

Firewalls and Internet Security Second Edition

Repelling the Wily Hacker

William R. Cheswick
Steven M. Bellovin
Aviel D. Rubin



ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

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 Addison-Wesley was aware of a trademark claim, the designations have been printed in initial capital letters or in all capitals.

The authors and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

The publisher offers discounts on this book when ordered in quantity for bulk purchases and special sales. For more information, please contact:

U.S. Corporate and Government Sales
(800) 382-3419
corpsales@pearsontechgroup.com

For sales outside of the U.S., please contact:

International Sales
(317) 581-3793
international@pearsontechgroup.com

Visit Addison-Wesley on the Web: www.awprofessional.com

Library of Congress Cataloging-in-Publication Data

Cheswick, William R.
Firewalls and Internet security : repelling the wily hacker / William R. Cheswick, Steven M. Bellovin and Aviel D. Rubin.— 2nd ed.
p. cm.
Includes bibliographical references and index.
ISBN 020163466X
1. Firewalls (Computer security) I. Bellovin, Steven M. II. Rubin, Aviel D. III. Title.

TK5105.875.I57C44 2003
005.8—dc21

2003000644

Copyright © 2003 by AT&T and Lumeta Corporation

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher. Printed in the United States of America. Published simultaneously in Canada.

For information on obtaining permission for use of material from this work, please submit a written request to:

Pearson Education, Inc.
Rights and Contracts Department
75 Arlington Street, Suite 300
Boston, MA 02116
Fax: (617) 848-7047

ISBN: 0-201-63466-X
Text printed on recycled paper
1 2 3 4 5 6 7 8 9 10—CRS—0706050403
First printing, February 2003

6

The Hacker's Workbench, and Other Munitions

It's a poor atom blaster that doesn't point both ways.

Salvor Hardin in *Foundation*
—ISAAC ASIMOV

6.1 Introduction

This chapter describes some hacking tools and techniques in some detail. Some argue that these techniques are best kept secret, to avoid training a new generation of hackers. We assert that many hackers already know these techniques, and many more (see Sidebar).

System administrators need to know the techniques and tools used in attacks to help them detect and deal with attacks. More importantly, the network designer needs to know which security efforts are most likely to frustrate an attacker. Much time and money is wasted tightening up some area that is not involved in most attacks, while leaving other things wide open.

We believe it is worthwhile to describe the techniques used because an informed system administrator has a better chance to beat an informed hacker. Small defensive measures can frustrate elaborate and sophisticated attacks. In addition, many of these tools are useful for ordinary maintenance, tiger-team testing, and legitimate hardening of a network by authorized administrators.

While most of the tools we discuss originated on UNIX platforms, the programs are often distributed in source code form, and many have been ported to Windows (e.g., *nmapNT* from eEye Digital Security). For the hackers, the same *class of service* is now available from virtually any platform.

Should We Talk About Security Holes? An Old View

A commercial, and in some respects a social, doubt has been started within the last year or two, whether or not it is right to discuss so openly the security or insecurity of locks. Many well-meaning persons suppose that the discussion respecting the means for baffling the supposed safety of locks offers a premium for dishonesty, by showing others how to be dishonest. This is a fallacy. Rogues are very keen in their profession, and already know much more than we can teach them respecting their several kinds of roguery. Rogues knew a good deal about lockpicking long before locksmiths discussed it among themselves, as they have lately done. If a lock—let it have been made in whatever country, or by whatever maker—is not so inviolable as it has hitherto been deemed to be, surely it is in the interest of *honest* persons to know this fact, because the *dishonest* are tolerably certain to be the first to apply the knowledge practically; and the spread of knowledge is necessary to give fair play to those who might suffer by ignorance. It cannot be too earnestly urged, that an acquaintance with real facts will, in the end, be better for all parties.

Some time ago, when the reading public was alarmed at being told how London milk is adulterated, timid persons deprecated the exposure, on the plea that it would give instructions in the art of adulterating milk; a vain fear—milk men knew all about it before, whether they practiced it or not; and the exposure only taught purchasers the necessity of a little scrutiny and caution, leaving them to obey this necessity or not, as they pleased.

... The unscrupulous have the command of much of this kind of knowledge without our aid; and there is moral and commercial justice in placing on their guard those who might possibly suffer therefrom. We employ these stray expressions concerning adulteration, debasement, roguery, and so forth, simply as a mode of illustrating a principle—the advantage of publicity. In respect to lock-making, there can scarcely be such a thing as dishonesty of intention: the inventor produces a lock which he honestly thinks will possess such and such qualities; and he declares his belief to the world. If others differ from him in opinion concerning those qualities, it is open to them to say so; and the discussion, truthfully conducted, must lead to public advantage: the discussion stimulates curiosity, and curiosity stimulates invention. Nothing but a partial and limited view of the question could lead to the opinion that harm can result: if there be harm, it will be much more than counterbalanced by good.

Rudimentary Treatise on the Construction of Locks, 1853

—CHARLES TOMLINSON

6.2 Hacking Goals

Though it may be difficult to break into a host, it is generally easy to break into a given site if there are no perimeter defenses. Most sites have many hosts, which share trust: They live in the same security boat. Internet security relies on a long chain of security assumptions, and the attacker need only find the weakest link. A generic hacker has the following goals:

1. Identify targets with a network scan
2. Gain access to the proper host or hosts
3. Gain control of those hosts (i.e., *root* access for a UNIX system)
4. Cover evidence of the break-in
5. Install back doors to facilitate future re-entry and
6. Repeat the preceding steps for other hosts that trust the “owned” host

The hardest step for the hacker is the second, and it is where we concentrate most of our security efforts. Often an exploit used in Step 2 gives the Bad Guy control of the host (Step 3) without further effort. This is why we strip all network services we can off a host (see Section 14.4.) It is also why we install firewalls: to try to limit access to network services that might be insecure.

6.3 Scanning a Network

Obscurity should not be the sole basis of your security, but rather one of many layers. An attacker needs to learn about your networks, your hosts, and network services. The most direct way is to scan your network and your hosts. An attacker can locate hosts directly, through network scanners, and indirectly, perhaps from DNS or inverse DNS information. They may find targets in the host files on other machines, from chat rooms, or even in newspaper reports.

Numerous programs are available for host and port scanning. The simplest ones are nearly trivial programs, easily written in a few lines of Perl or C. An intrusion detection system of any sort easily detects these scans, so they are run from stolen accounts on hacked computers.

ICMP pings are the most common host detection probes, but firewalking packets (see Section 11.4.5) may reach more hosts. And be consistent: One major military network we know blocked pings to some of its networks, but allowed in UDP packets in the *traceroute* port range.

An attacker may scan an entire net host by host—the Internet equivalent of war dialing for the phone system—or they may send directed broadcast packets. Directed broadcasts are more efficient, but are often blocked because of Smurf attacks. Scans can be much slower and more subtle to avoid detection. There are numerous scanning tools; see Table 6.1.

Once located, hosts may be fingerprinted to determine the operating system, version, and even patch level. These programs examine idiosyncrasies of the TCP/IP stack—and we have heard reports that they can crash some hosts. Fingerprinting programs use arcane details that were once

Table 6.1: Some Common Scanning Tools

Tool	Networks	Ports	Fingerprint
<i>nmap</i>	X	X	X
<i>fping</i>	X		
<i>hping</i>	X		
<i>pinger</i>	X		
<i>queso</i>			X
<i>strobe</i>	X	X	

of interest only to the propeller-heads who wrote TCP/IP stacks. Now they have actually helped improve the security and robustness of some of this software.

Hosts are also scanned for active ports. They seek active network services, and often identify the server software and versions. Port scanners can be very subtle. For example, if they send a TCP SYN packet, but follow the computer's response with an RST to clear the connection instead of sending an ACK to complete the three-way handshake, a normal kernel will not report the connection attempt to a user-level program. A simple alarm program in `/etc/inetd.conf` will miss the probe, but the attacker can use the initial response to determine if the port has a listener, available for further probes.

Carefully crafted TCP packets can also probe some firewalls without creating log entries. It is important that packet monitoring systems log packets, not just completed connections, to make sure they detect everything. Table 6.1 lists port scanners, too.

6.4 Breaking into the Host

There are three approaches to breaking into a host from the Internet:

- Exploit a security hole in the network services offered by the host
- Duplicate the credentials of an authorized user or
- Hijack an existing connection to the host

In the early days of the Internet, the first two were most common; now we see all three. There are other ways to break into machines, such as social engineering or gaining physical access to the console or host itself. One paper [Winkler and Dealy, 1995] describes a typical approach using a corporate telephone directory.

Security flaws are numerous. They are announced by various CERT organizations and vendors, usually without details. Other groups, such as Bugtraq, include detailed descriptions and "exploits" (also known as "sploits"), programs that exercise the flaw. The hacking community discovers their own security holes as well, and sometimes exchanges them like baseball cards.

Many conferences run open wireless networks with Internet connectivity these days—a substantial convenience. But even at security conferences, *dsniff* catches a surprising range of passwords, some obviously not intended to be guessable.

Strong encryption, such as found in IPsec, *ssh* (we hope), and SSL completely foils sniffing, but sometimes it can be inconvenient to use, or tunnels may not be used properly. For some systems (like your *New York Times* password), you may choose to use a junk password you don't care about, but make sure you don't use that password elsewhere.

6.8.3 *Nmap*—Find and Identify Hosts

We mentioned *nmap* earlier. It has an extensive database of TCP/IP stack idiosyncrasies for many versions of various operating systems. If you point it to a system it doesn't recognize, it displays the new fingerprint and asks to submit it to the database managers, to appear in future versions.

The database can be quite useful on its own—companies are quite interested in inventory and version control, and *nmap* has the best database we know of for *host fingerprinting*, or identifying the operating system and version from afar. It does need to find closed and open TCP ports to help identify a host. A safe host of the kind we recommend can have such restricted responses to network accesses that *nmap* does not perform well. In addition, there are now programs, such as *iplog* [Smart *et al.*, 2000] and *honeyd* [Spitzner, 2002], that will deceive *nmap* and other scanners about the operating system you are running. This can be useful for honeypots and similar projects.

It has been reported that *nmap* probes have crashed some versions of Microsoft Windows, and many stacks embedded in devices like hubs and printers. This limits the value of *nmap* for auditing important networks. Many network administrators have been burnt by *nmap* and won't run it.

6.8.4 *Nbaudit*—Check NetBIOS Share Information

Nbaudit (also called *nat*, unfortunately) retrieves information from systems running NetBIOS file and printer sharing services. It can quickly find hosts with shared disks and printers that have no password protection. It also tries a list of common usernames, which unfortunately is often successful.

6.8.5 *Juggernaut*—TCP Hijack Tool

Until the mid-1990s, TCP hijacking was a theoretical attack. We knew practical attacks were coming, but the tools hadn't been written. In 1995, Joncheray [1995] described in detail how to do it; in early 1997, *Phrack* released the source code for *Juggernaut* [daemon9, 1997]. As with many hacking tools, the user doesn't really need to know the details of the attack. In fact, an interactive mode enables the attacker to watch a number of TCP sessions at once.

The program permits eavesdropping, of course. It can also let you substitute text in specific packets, or hijack the session while running a daemon that suppresses the original user. To that user, it appears that the Internet is down, again. It would be illogical to suspect that an attack is occurring unless there is other evidence: TCP connections go away quite often. Storms of ACK packets might be noticed, but those aren't visible to end-users.

The attacker does need to run this program on a host that has access to the packet flow, usually near one of the endpoints. Suitable hosts are rare near the main packet flows in the “middle” of the Internet, and the packet rates are probably too high.

Sessions can be hijacked after authentication is completed—which renders the authentication useless. Good encryption completely frustrates this tool and all TCP hijacking attacks.

6.8.6 *Nessus*—Port Scanning

The first port scanner we are aware of was a set of shell scripts written by Mike Muus around 1988. ISS followed in the early 1990s, and then SATAN. Now *Nessus* is available from <http://www.nessus.org>. The network and host probes are run by a server, to which clients may connect from afar. Public key encryption and user accounts are used to restrict these connections.

The various tests *nessus* uses are modularized; and new tests are created often and are available for download. Like the fingerprint descriptions for *nmap*, these modules make it easy to extend and expand the capabilities.

6.8.7 DDoS Attack Tools

Trinoo is a set of tools for performing distributed denial-of-service attacks. There is a master program that can issue attack or even update instructions to slave programs installed on a wide variety of hosts. The communications can be encrypted, and the master’s instructions sent with a spoofed address to make traceback difficult. A number of other programs with similar capabilities are available.

DDoS attacks are discussed further in Section 5.8.3.

6.8.8 Ping of Death—Issuing Pathological Packets

This program was one of the first to attack hosts by sending pathological TCP/IP packets. This particular attack involved sending packets longer than the maximum length expected by the software. Fragmentation packet processing was used to confuse the software.

There are many other programs with similar goals. TCP/IP is quite complicated and there are only a few original implementations of it.

6.8.9 Virus Construction Kits

There are a wide variety of virus construction kits. Some are so sophisticated, we are surprised that they don’t come with user help lines and shrink-wrap agreements.

Most kits include a GUI of some sort, and a variety of options: what kind of virus to create, when it should be activated, how it is transported, and so on. All the popular virus transports are available: Word macros, boot sectors, palmtop downloads, to name just a few. Polymorphism and encryption are options as well.

If you wish to experiment with these, we advise great caution. Isolated nets and virtual machines are your friends.

Intrusion Detection

Behold, the fool saith, “Put not all thine eggs in the one basket”—which is but a manner of saying, “Scatter your money and your attention”; but the wise man saith, “Put all your eggs in the one basket and—*watch that basket!*”

—PUDDIN’HEAD WILSON’S CALENDAR

It is important to post sentries near things you wish to protect, and an *intrusion detection system (IDS)* helps perform this function. As commercial products, these security tools have been promoted as the ultimate solution to network intrusions, and many IT managers have proclaimed that their network was secure because they had installed the latest firewall and IDS. These can help, but they’re far from a panacea.

There are several types of intrusion detection systems. *Network IDSs (NIDSs)* eavesdrop on network traffic, looking for an indication of an intrusion. Various host-based systems scan files or traffic for incoming viruses; some analyze system call patterns or scan for changed files.

IDSs are plagued by several inherent limitations. False positives (false alarms) occur when an IDS incorrectly concludes that an intrusion occurred. False negatives are actual intrusions that are missed by the IDS. For most intrusion detection systems, both of these are unavoidable and occur with such frequency as to greatly limit their value. It usually requires human intervention to determine evilness or the lack thereof, and some of the sources of weird packets may be too difficult to fix.

Finally, network IDS systems usually work by sniffing the network traffic and gluing the packets together into streams of data. It is easy to do a fair job of this—it seems almost trivial. Most sniffers do just this, but a number of papers, such as [Ptacek and Newsham, 1998], [Paxson, 1998], and especially [Handley *et al.*, 2001], point out that this job is nearly impossible to get exactly right. The problem is that a sniffing program needs to know the states of the TCP/IP stacks at both ends of the communication, plus the idiosyncrasies of their implementation details. For example, suppose that two packets arrive containing overlapping data. It is TCP’s job to

reassemble the stream of data, and now it has two versions for the overlapping region. Should it use the first copy or the last? The RFCs are silent on this, and implementations vary. If the overlapping data in the two packets doesn't match, which version should the NIDS assume was delivered?

The overlapping data problem may seem to be contrived, and it is rare in the wild, but programs such as *fragrouter* [Song *et al.*, 1999] intentionally modify TCP/IP streams to confuse eavesdroppers. *Fragrouter* takes scripts written in a little language that define the kinds of pathologies desired on the packet stream. Outgoing packet streams can be distorted so badly that the monitoring host may be incapable of decoding the data stream.

Four places need to process pathological TCP/IP packet streams correctly: clients, servers, firewalls, and NIDSs. [Handley *et al.*, 2001] propose an intervening device to normalize the packet stream. One can imagine adding such functionality to a firewall, making it behave more like a circuit-level gateway. Some firewalls already do some of this; they reassemble fragmented packets to protect against short fragment attacks. True circuit-level gateways (see Section 9.3) cleanse IP streams as well.

This sort of issue was the basis of our recommendation in the first edition that corporations avoid direct IP connectivity between their corporate networks and the Internet, and use application- or circuit-level firewalls instead.

These odd, contrived packets are rare in normal Internet traffic. IDS software should notice when an unusual number of packets are fragmented, contain small TTL values, or have other unusual pathologies. Unfortunately, these do occur in legitimate traffic, and can't be used as the sole indicia of malicious activity. To give just one example, *traceroute*—a very normal network diagnostic program—is a leading cause of small TTLs.

False negatives are an obvious problem: An attack occurred and we missed it. False positives are a particular problem, because they are very hard to avoid without disabling the desired features of the IDS. People are expensive. People competent enough to do a good job of monitoring these alarms quickly tire of them, and step quickly through the reports or ignore them entirely. False alarms can also come from configuration errors.

When evaluating an IDS system, always check out the false-positive rates.

15.1 Where to Monitor

It is important to understand the limitations of IDSs before you consider installing one. The most important question to ask is "What is the purpose of the IDS?" One legitimate reason to install an IDS outside of your firewall is to justify funding requests to your boss (this is a threat model in which management is the enemy). There is no point in monitoring the outside of your network to see if you are under attack—you are. That's not to say that you should ignore the outside, but it is probably more valuable to record and store the outside traffic for later examination than to attempt real-time intrusion detection. If you are a researcher trying to learn about new attacks, such information is invaluable. However, there is too much traffic going by, and an IDS is too weak a tool to do real-time analysis. It is a fine place to train people who are learning about networks and IDS devices.

IDS devices become more useful when deployed near important assets, inside the various security layers. They are like a video camera installed in a bank vault, a final layer of assurance that all is well. The more restricted the normal access to a network or a machine is, the more sensitive the rules should be for the detectors. People probably shouldn't be issuing Web queries from the payroll computer.

15.2 Types of IDSs

Different kinds of IDSs have different strengths and weaknesses. Signature-based IDSs have a database of known attacks; anything matching a database entry is flagged. You don't get many false positives if the system is properly tuned, but you are likely to experience false negatives because they only recognize what is in the database. Unfortunately, the sweet spot between overly broad signatures (which match normal traffic) and overly narrow signatures (which are easy to code around) is hard to find. At the very least, signature-based systems should incorporate context, and not just rely on string matches.

Anomaly-based IDSs, which look for *unusual* behavior, are likely to get false positives and false negatives. They work best in an environment with a narrowly defined version of *normal*, where it is easy to determine when something is not supposed to occur. The more special purpose a machine is, the more constrained normal behavior is, and the less prone it is to false positives.

Anomaly detection is an interesting area of research, but so far has yielded little in the way of practical tools. [Forrest *et al.*, 1996] and [Ko *et al.*, 2000] have produced some interesting results. Forrest developed a tool that monitors processes running on a computer and examines the system calls. The tool has a notion of what a normal pattern of calls is, and recognizes when something happens that is not supposed to. The tool uses *n*-grams of system calls and slides a window across the sequence of calls that a process has executed. If the behavior of a process varies beyond a certain threshold from known trace behavior, an anomaly is signaled. The key feature that the tool looks for is the order of the calls in a sequence. Certain calls are preceded by others, and if enough calls are preceded by the wrong calls, it assumes that there's trouble. [Somayaji and Forrest, 2000] describe how to slow down or abort processes that behave too badly.

As noted, IDSs can be host-based or network-based. The two are complementary, not mutually exclusive; each has its strengths and weaknesses. Host-based systems tend to know the state of their own machine, which simplifies the processing of the data flows, but the software can be subverted if the host is compromised. Network-based devices are stand-alone units and are presumably more resistant to attack or even detection. On several occasions we have advised IDS designers to cut the transmit lead on their Ethernet cables, or at least suppress the emission of packets in the software. That's hard to do with today's twisted-pair Ethernets on some platforms; however, there are dedicated hardware devices designed to tap networks without any possibility of transmitting onto them.

For some environments, such as DMZs, our favorite kind of IDS is a *honeypot*—a machine that nobody is supposed to touch. Any source of traffic to that machine is at the very least misbehaving, and more likely evil. A honeypot might not work in an open corporate environment, but is well suited to a dedicated network, which should not have anything except dedicated machines.

A honeypot on the public Internet can be useful for studying hacker behavior, though some hackers have learned to avoid them. One of the prettiest examples is Niels Provos' *honeyd* [Spitzner, 2002, Chapter 8]. It mimics an entire network, populated by many different sorts of machines. However, you can't rely on this for determining if someone has penetrated a single machine; at most, it can detect scans. To cope with the false positives and false negatives, some people use multiple IDSs whose outputs are correlated. Time correlation can be used to detect "low and slow" attacks.

15.3 Administering an IDS

An intrusion detection system requires a significant amount of resources. IDSs have to be installed in strategic locations, configured properly, and monitored. In most environments, they will have to deal with an amazing amount of broken network traffic. For example, an HP printer driver we used tried to find everything on the subnet without knowing about masks, so it scanned an entire /16 network looking for an HP printer. Network management software sometimes does the same. Someone running an IDS has to be able to deal with this kind of traffic and must be tolerant of a lot of noise [Bellovin, 1993]. They also have to make sure they do not become too complacent because IDSs tend to cry wolf.

15.4 IDS Tools

Many IDS tools are available, both free and commercial. Sniffers, such as *snort* (see the following section), *ethereal*, and *bro* [Paxson, 1998], are very useful. *Ethereal* provides a nice GUI that enables you to reproduce TCP streams so that you can view application-level data. It can also dump network traffic to a file for later investigation.

Commercial products range from pure snake oil to fairly useful tools. Some products try to apply AI techniques to the problem. Others collect distributed information and try to assemble an overall view of an attack.

15.4.1 Snort

Perhaps the most popular free intrusion detection program is *snort*, developed by Martin Roesch. *Snort* is open source, and there is an active community of users and contributors; see <http://www.snort.org/>. The program is available on a wide variety of platforms—it works anywhere that *libpcap* runs.

Snort can be used in several ways. It can sniff a network and produce *tcpdump*-formatted output. It can also be used to log packets so that data mining tools and third-party programs can do after-the-fact analysis on network traffic. The most interesting feature of *snort* is its ability to design a ruleset that recognizes certain traffic patterns. Many rules are available for *snort*, and they are often shared among users and posted on the Internet. *Snort* can be configured to recognize *nmap* probes, known buffer overflow attacks, known CGI exploits, reconnaissance traffic, such as