

8

The Hacker's Workbench

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

Salvor Hardin in Foundation
—ISAAC ASIMOV

8.1 Introduction

Sometimes, breaking into a system is easy. You just guess at an account and password. More often one has to poke around a little to find a target host and to locate its vulnerabilities. The Bad Guys are well equipped with automated tools to help them perform these tasks. Such tools allow them to scan networks automatically, locating possible victims and twisting the doorknobs. We've seen many examples of such tools. They have left them behind on various compromised computers, and our alarms have detected automated attacks on our own external networks. (In fact, we've detected them on our internal network as well. So far, all these have come from machines used by AT&T corporate security.) We believe it is worthwhile to describe the techniques used because an informed system administrator can beat an informed hacker any day. Also, many of these tools are useful for ordinary maintenance and legitimate hardening of an internal network by authorized administrators.

Some argue that these techniques are best kept secret, to avoid training a new generation of hackers. We'd like to point out that many hackers already have them, and they are relatively unavailable to legitimate users.

As always, we prefer a tool-based—that is to say, a UNIX-like—approach. A set of reusable modules, plus some shell scripts for glue, are far more flexible than massive, hard-coded Things. To illustrate this point, we have included the source to some of our programs.

Incidentally, in most cases it is not worth taking extraordinary precautions to protect any security software you've acquired or built. At best, you'll deter a few pokes from curious local

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—milkmen 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

users. Hackers encrypt their tools not so much to keep them from you, as to avoid detection and prosecution.

8.2 Discovery

The first step in a break-in is surveillance: figuring out whom to attack. It is not hard to discover the networks and hosts belonging to a particular target. Large sites have computers listed in `hosts.txt`, which is available to all from the NIC. Many novice hackers start with this list. (Indeed, we believe that our gateway machines are probed more often than most in part because they appear in this file.) The primary goal of a search is to find new networks. Once found, these can be scanned easily for hosts. Then hosts are scanned for weaknesses. One compromised host is usually the door to many more.

Whois can also provide a starting point. We were surprised to find the number of entries reported by

```
whois 'at&t'
```

Many of our internal network numbers are listed, because they are duly registered. *Whois* also displays a number of domain names and possible connections belonging to our many business units. This list is useful for us: we can monitor the various networks listed to see if any Inside ones get announced on the Outside, or vice versa. If so, we know we have a leak, either accidental or deliberately installed by some organization not conversant with our security concerns.

Networks can be found in `/etc/networks` (also derived from NIC data), although the list is incomplete. They are also announced as routes, but it is difficult to associate a simple numeric address with a particular target site. The *traceroute* command (or equivalent) can reveal intermediate networks in a path to a host. One fine source of network information is a router that is willing to chat using SNMP. Network management information includes a dump of the routing table. Routers are often configured to provide this information to group “public” on a read-only basis. (Some regional networks block SNMP access to their members’ machines from external sources.)

The name server can supply more complete information. As noted earlier, many name servers are configured to dump their entire database to anyone who asks for it. You can limit the damage by blocking TCP access to the name server port, but that won’t stop a clever attacker. Either way provides a list of important hosts, and the numeric IP addresses give network information. *Dig* can supply this data:

```
dig axfr zone @target.com +pfset=0x2020
```

Specifying `+pfset=0x2020` suppresses most of the extraneous information *dig* generates, making it more suitable for use in pipelines.

When a network is selected, the individual machines on it must be identified. A subnet can be scanned several ways for hosts. These scans are the Internet equivalent of *Wargames dialing* or *demon dialing*, where a computer calls every possible phone number in a target exchange to detect modems. Any that answer are subject to further probes. (To the homeowner these probes

```
#!/bin/sh
#
# usage: cscan [-n] a.b.c
#

TMP1=/tmp/cscan$$
TMP2=/tmp/cscan$$a

case $1 in
-n)    opt="-n"; shift;;
*)    opt=""
esac

network=$1; shift

{
    seq 1 254 |
    sed 's/./& '$network'.& 30/'
    sleep 2
}
pinglist -d 5 $opt |
sed 's/ exceeded$//' >$TMP1

responses=`cat $TMP1 | wc -l`
sort -nu <$TMP1 | tee $TMP2

found=`cat $TMP2 | wc -l`
echo $responses responses, $found found >&2

rm -f $TMP1 $TMP2
```

Figure 8.1: A tool for scanning a Class C network.

appear as those annoying phone calls with no one at the other end. On the Internet such probes are seldom noticed.)

There are several ways to find a host. The most obvious is to *ping* the desired address with an ICMP Echo Request. It can also be useful to try to connect to a host's *telnet* or SMTP ports. These tests can supply useful information like the host's domain name, operating system, and manufacturer.

For each probe of a new or unused numeric IP address, the router at the far end must issue an ARP request. These ARPs take time to satisfy, which, coupled with network transit time and local kernel load, limit the minimum probe time to about 10 to 20 ms for a nearby network. At this rate, a nearby Class C network can be scanned in a few seconds. A Class B takes less than an hour; much less if you skip all subnets that generate local `Destination Unreachable` messages.

An inverse name server lookup translates a numeric IP address into a domain name. Although many sites fail to keep this information current, it can offer a quick list of the important hosts on a net as well:

```
for i in `seq 1 254`
do
  dig +pfset=0x2020 -x 192.20.225.$i
done
```

All these tests can be used to build a database of networks and hosts. Subsequent scans should start with these databases and enhance them. DNS HINFO records, if present, provide splendid clues as to what forms of attack are likely to succeed. Since networks change and hosts go down, it is useful to timestamp new information, and try again later to augment the list.

A few things are worth noting about the test sequence given above. First, the list of addresses—1 through 254—is generated by the *seq* command, a useful tool for writing all sorts of shell scripts. The addresses are mapped to names by the *dig* command. The *host* command is better behaved, but less well known. Both programs will accept a list of requests on `stdin`, which is useful for large networks:

```
seq 1 254 | sed 's/^/192.20.225./' | host -x -i 2>/dev/null
```

8.2.1 Pinglist

To map a net, as opposed to learning what hosts are listed in the DNS for that network, one must actually send out packets to the candidate hosts. Our tool for scanning a Class C network is shown in Figure 8.1. It's based on the *pinglist* program, our analogue to *traceroute*. *Pinglist* reads a list of message identifiers, IP addresses, and time-to-live fields, and sends out UDP packets, although it could just as easily send out ICMP Echo Request messages. The returning packets are written to standard output, as a proper UNIX filter should. The output is captured and analyzed by other programs; we've even built a version of *traceroute* using it (Figure 8.2).

We don't use ICMP *ping* requests, although we could. *Pinglist* was derived from Van Jacobson's *traceroute*, which uses UDP packets. It would be easy to add a ping packet type. They are more likely to penetrate a packet filtering router than a random UDP packet.

The *pinglist* approach moves the critical privileged network access code to a single small program that can be carefully examined and run `setuid` to *root*. The network applications themselves can be written and run by unprivileged users. The installer of *traceroute* can set a minimum delay between packet transmission, which can help prevent network flooding.

8.2.2 Mapping Tools: Fremont

Of course, network mapping has been done before. Several commercial products are available for network administration that automate mapping. They usually have a fancy graphical interface. *Fremont* [Wood *et al.*, 1993] is a publicly available network mapping system. Each of these tools uses the techniques mentioned here, and others as well. But be careful about how you use a mapper: from time to time we have detected probes from these programs when their owners misconfigure them so that they try to map the entire Internet.

```

#!/bin/sh
#
#      traceroute, like the real thing, but using pinglist.

case $1 in
-n)    opt="-n"; sleep=0; shift;;
*)    opt="";   sleep=3
esac

case $# in
1)    host="$1";;
*)    echo "usage: $0 [-n] host"
      exit 1;;
esac

{
      seq 1 10 | sed 's/.*/& '$host' &/'
      sleep $sleep
} |
pinglist $opt | sed '/ reached$/q'

```

Figure 8.2: Traceroute implemented using pinglist.

8.3 Probing Hosts

We don't know of any commercial products that probe remote hosts for network weaknesses. There are plenty of home grown versions, and we are surprised that we haven't encountered more.

These tools are the most controversial of the lot. But they are very important. A corporation has a legitimate interest in hardening the hosts on their internal networks. Even gateway administrators appreciate a sanity check from such software before going on-line, or after an upgrade. Corporate *tiger teams* use tools like these to keep ahead of the hackers, or even to detect hosts that hackers have weakened.

The list of known network services weaknesses is fairly short. Most are easy to detect with short programs. Most are specific to certain versions of certain operating systems on certain hosts. Older bugs hang around for a long time: many people don't bother to upgrade their system software. They just buy new systems, or continue to use the old ones.

Mike Muuss incorporated a nice host test design in his network sweep program. Each test is incorporated in a separate shell script. The scripts' return code indicates that the host is or is not vulnerable or is unreachable. A shell script loops through a list of machines to be tested. Here are some of the most common and effective tests:

- Can TFTP deliver an `/etc/passwd` file? If so, save the file for subsequent testing. Some systems still come out of the box configured this way.
- Does the `/etc/passwd` file in the FTP directory contain real passwords for the host?

- Are any interesting file systems exported via NFS? In particular, is the `/etc/passwd` file exported?
- Is the old *fingerd* hole present? Generally, it has been fixed, but the test doesn't take long to run.
- Test for *guest* and *visitor* accounts. These tests can use *expect* [Libes, 1991] or similar technology.

Other tests have been incorporated into newer programs. Klaus's *ISS*, the *Internet Security Scanner*, scans an entire domain or a subnet and looks for various holes:

- Checks for a variety of commonly unprotected logins or mail aliases, such as *sync*, *guest*, *lp*, etc.
- Connects to the mail port, and logs information such as the apparent operating system and mailer types and versions.
- Attempts an anonymous FTP connection; if it succeeds, see if the home directory is writable. *ISS* could try to grab the `/etc/passwd` file, but doesn't.
- Uses *rpcinfo* to see if certain suspect services are running. But it doesn't actually try any attacks itself.
- Looks for file systems exported to the world.
- Invokes *ypx* to find detailed information about cracking NIS.

The last item is in some sense the most interesting, because it demonstrates just how powerful a specialized hacking (or auditing) tool can be. *Ypx* uses RPC directly to try to grab the password map via NIS. If you wish, it can grab other maps as well. For these operations to work, it needs to know the NIS domain name. You can either supply it with a list of guesses of your own or it can try to deduce it. It does this by looking at the host name, the host address, and even the initial greeting from *sendmail*. Or, although neither *ypx* nor *ISS* tries it, it is sometimes possible to inquire of the *bootp* daemon; often, it will tell you the domain name. (Hackers have that tool, too, although *ISS* doesn't try to invoke or simulate it.)

Both of these programs are already fairly potent, and it seems all but certain that *ISS*, at least, will be enhanced. The *ISS* documentation mentions an unimplemented option to scan hosts' TCP port number space à la our *scanports* program. Regardless, they define a minimum level of security that you must meet. The programs were posted to netnews. Rest assured that most of your neighborhood hackers have them.

The output of the sweep programs is a list of possibly vulnerable machines, plus several `/etc/passwd` files. Run a password cracker on the password files. *Crack* [Muffett, 1992] is often used. It is fast and easily available. (Many hackers use spare cycles on the machines they have already invaded to crack other password files.)

SATAN, the *Security Analysis Tool for Auditing Networks* [Farmer and Venema, 1993], is similar in intent to *ISS*. It's modular in design; any file in the *SATAN* directory with a `.sat` suffix

is executed when probing hosts. One of its most interesting features is a transitive trust analyzer. It uses the *finger* command to learn the source of logins and then recursively probes those sites. Very often, even tightly controlled sites have users who log in from easily penetrated machines.

As of late 1994, the code for *SATAN* has not yet been released, but we were told it would be made available soon.

8.4 Connection Tools

As we have discussed, it is difficult to set up filter rules correctly. To validate configurations, we use a small set of tools to connect to arbitrary ports. For many services, one can use *telnet*, but it's hard to pipe things to and from *telnet*. It is designed for human input/output, and it emits *telnet* escape codes onto the net.

Our programs establish a network connection to the desired host and service and then execute some arbitrary command, with `stdin` and `stdout` pointing to the network connection. That makes it easy to build shell scripts that use network services.

They also have the ability to bind to a particular local host address and port number. We use that feature to probe packet filters for correctness; many will blithely let in packets from low-numbered ports or from specific host addresses.

A complementary tool is a port scanner. Again, we use a modular approach:

```
scanports targethost `seq 1 1023`
```

It does the obvious: it reports any TCP services available on `TARGETHOST`. It also rings alarms on booby-trapped machines like ours. The code to *scanports* is shown in Figure 8.3. Other useful variants include UDP and RPC scanners; the latter should attempt to determine the server listening on each port.

Obviously, packet filters are a good defense against port scanners. Proper configuration can make the defense even stronger. If your filter returns `ICMP Destination Unreachable` messages when it bounces a packet, the scanner will receive immediate notification of the failure, and will try the next port fairly quickly. But if the probe packet is silently discarded, the sending TCP will have to suffer through a comparatively long timeout period before proceeding. (This suggests that trap programs should wait awhile before dropping the connection. But that leaves you more prone to denial-of-service attacks from the outside.) More sophisticated scanners can overcome this problem by exploiting parallelism and hand-crafted packets. Still, it helps.

8.5 Routing Games

As we have described, playing games with routing information is a fruitful way to attack hosts. Fortunately, it's not too hard to detect some of the vulnerabilities.

You can find out if your gateway passes source-routed packets by trying it. Recent versions of *telnet* and *traceroute* support specification of such routes on the command line. The code should compile and run on most systems without too much trouble. Be careful, though: many hosts have a buggy implementation of source routing. You are as likely to crash the machine as penetrate it.


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

int
main(int argc, char *argv[])
{
    int debug = 0;
    int s;
    char **cp, *path;
    extern char *ipcpath();

    if (argc > 1 && strcmp(argv[1], "-d") == 0) {
        argv++;
        debug = 1;
    }
    if (argc < 3) {
        fprintf(stderr,
            "usage: scanports ipchost port...\n");
        return 99;
    }

    for (cp = &argv[2]; *cp; cp++) {
        path = ipcpath(argv[1], "tcp", *cp);
        if (debug) fprintf(stderr, "%s\n", path);
        s = ipcopen(path, "");
        if (s < 0) {
            if (debug)
                ipcpererror("scanports");
            continue;
        }
        printf("%s ", *cp);
        close(s);
    }
    printf("\n");
    return 0;
}
```

Figure 8.3: Scanning for servers.

To detect attempts at confusion via ICMP `Redirects`, write a program that listens for such packets on a raw ICMP socket. All programs that have an open ICMP socket will receive copies of almost all ICMP packets. It's not hard to select just the interesting ones. The source to the `ping` command is readily available and provides a good starting point.

Before raising alarms, make sure that the `Redirect` is actually due to enemy action. Apart from genuine `Redirect` messages, which may be possible with your configuration (though not with ours), there are buggy routers on the Internet that send totally bogus ICMP messages without any malicious intent at all [Bellovin, 1993].

It is possible to write a program that will generate `Redirect` messages, but it's harder than it seems to test if your hosts and routers are vulnerable to them. A fair amount of sanity-checking can and should be done before such messages are honored. If you perceive no effect, it may be that your target was smarter than you thought, even if it isn't necessarily smart enough to resist all such attacks.

8.6 Network Monitors

Sooner or later, you may find that you need to monitor the raw network traffic itself. This is one of the first tools a site writes or acquires when they discover an active hacker attack on their system. (You may want to get one *before* the attack.) We use `tcpdump` on a spare workstation, with the Ethernet interface in promiscuous mode. This allows us to pick up address space probes and the like. If you prefer, there are many PC-based tools to do this, too.

If possible, do *not* provide the necessary drivers on an exposed machine. Hackers love to find hosts with such things; they are marvelous tools for password collection, as explained later. We cut the transmit lead on our monitoring station's drop cable; this effectively prevents attacks on the machine. Unfortunately, that also interferes with its ability to monitor for certain address-space scans. Before the scan packets themselves are transmitted onto the wire, the router must first see a response to an ARP request. For nonexistent machines, no one will answer, so the IP packet itself will not be sent; that deprives the monitor of source address information. Accordingly, we have another machine—in fact, the gateway itself—act as a *proxy ARP* server for a variety of likely addresses: some dummy entries in the DNS, addresses at the beginning and end of the address space, and a few random entries. (We'd cover the entire address space, but the ARP table is too small for that. We could put the entries in our regional network's router.)

8.7 Metastasis

Once an account is secured on a machine, the hacker has several hacking goals:

- destroy evidence of the successful invasion,
- obtain passwords for accounts on the machine,
- obtain *root* access to the invaded machine,

Is UNIX an Insecure Operating System?

UNIX has a reputation for being an insecure operating system. Though there are certainly many insecure machines running UNIX, we think this charge is imprecise at best.

The operating system kernel isn't the problem; the simple set of security primitives works well on most versions. Large commercial implementations slip up sometimes and have problems, such as the security hole in Sun's divide trap processing. But those are rare; few security problems stem from the kernel itself.

UNIX, however, is a terrific platform for making tools, including tools that help hackers. A shell script that emulates *login* and captures passwords is only a few lines long. (It's like giving prisoners access to a machine shop that is so good they can make the keys needed to unlock their cells.)

It is also true that UNIX is an administrative nightmare. Vital files are scattered over numerous directories. `Setuid` programs can change the rules in unexpected ways if the capability is misused.

Many of the early networking programs were written with little consideration for security. The Internet was new, the authors of the programs were not primarily interested in security, and it was marvelous that the programs existed at all. These were incorporated uncritically into commercial releases, and have caused much trouble since. Very few vendors have revisited these basic design decisions.

Finally, many commercial distributions of UNIX have reflected the early traditions of a friendly user community, or a somewhat misguided sense of what makes a system easy to administer. The machines come out of the box with the permissions wide open. We would prefer another stance: that they come tight and secure, and the installer would have to run a program *expose-me* to open up the host. Progress has been made on this point, but there's still a lot of room for improvement.

- set up password-sniffing tools, if possible,
- open new security holes or backdoors in the invaded machine, in case the original entry is discovered and closed, and
- find other hosts that trust the invaded host.

The last two goals are the most important; they usually allow the hacker to spread to a large number of hosts from a single security slip.

The hackers use a number of tools to effect these goals. Many things are simple manual checks:

- What hosts are trusted? Check `/etc/hosts.equiv` and all the users' `.rhosts` files.
- Check the mail alias database and log files. If user *foo* forwards mail to some other machine, or receives mail from account *foo* on some machine, that user probably has an account on that machine and may have a `.rhosts` file there pointing back to the first machine. (Trust is often symmetric.) Login accounting records are almost as valuable.
- Fetch and crack the `/etc/passwd` file if it hasn't been done already. If *root* access is obtained, fetch and crack the shadow or adjunct password file, if used.
- If appropriate permissions exist, install Trojan horses to collect passwords. Programs such as *login*, *telnet*, *ftp*, and even *su* are likely candidates. The hacker may even try to exploit typographical errors like *telent*; the real *telnet* command may be properly protected, but a directory later in the victim's `$PATH` may be writable.
- Try to force *root* execution of commands like

```
cp /bin/sh /tmp/.gift
chmod 4775 /tmp/.gift
```

- If possible, run a network monitor and collect passwords from *telnet* and FTP sessions. This is much easier than password cracking!

A program such as *COPS* [Farmer and Spafford, 1990] works for the cracker, too. It 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.

The Bad Guys do 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*. We saw Berferd break into a host, then use *sendmail* to become *uucp* or *bin*, and then become *root* from there. Many leave Trojan programs around for the system administrator to execute inadvertently. (Make sure the current directory doesn't appear in your administrator's execution path.)

It is alarmingly easy to obtain privileged access on many machines.

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

A truly evil attacker has further goals than simple subversion of hosts, such as searching through private files or reading queued email, or destroying or corrupting important files. Even mild-mannered hackers have been known to destroy systems if they thought they had been detected.


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.

A recent 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), etc.

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.

We do not claim sufficient wisdom to answer the question of whether or not 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 or not the people involved have learned better. In short—can you trust them? There is no universal answer to that question.

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

Once a machine is suspected of harboring an alien with *root* privileges, the administrator has no choice but to reload the system from the distribution media. User accounts should be reloaded from dumps taken *before* the attack occurred if it is possible to determine when that was. Executable files should be cleared from user accounts, and files like `.rhosts` cleared. The system must receive the latest security patches before it is connected to the network. Sometimes a reloaded machine is quickly reattacked and subverted again.

8.8 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 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 email 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.

8.9 Further Reading

This book, once finished, is a static construct; there is no way for us to update your copy with information on new holes and new tools. You have to assume the responsibility for staying current.

The Internet itself is a useful tool for doing this. There are a number of security-related newsgroups and mailing lists that you may want to follow. We list the ones we know of in Appendix A.

Another source of information is the hacker community itself. You may want to read *2600 Magazine*, the self-styled "Hacker Quarterly." It contains a lot of bragging and is somewhat focused on *phone-phreaking*, but it has a lot on computer security as well. Useful online publications include *Phrack* and the *Computer Underground Digest*.

A better source is the hacker bulletin boards. They are not that hard to find—ask around, or check some of the listings in *Boardwatch*. Once you find one, you'll see pointers to many more. The signal-to-noise ratio on these systems can be rather low, especially if you don't like the poor or variant spelling of the "d00dz" in the subculture, or if you aren't interested in "warez"—stolen PC software—but you can also learn amazing things about how to penetrate some systems. On the Internet, there is *Internet Relay Chat (IRC)*, a real time conferencing system. Some of the "channels" are dedicated to hacking, but participation is not necessarily open to all comers.

If you're going to participate in some of these forums, you need to make some ethical decisions. Who are you going to claim to be? Would you lie? You may have to prove yourself. Would you contribute sensitive information of your own? You can get remarkably far even if you admit that you are a corporate security person, especially if the other participants believe that you want information, not criminal convictions. (One friend of ours, who *has* participated in various raids, has been asked by various hackers for his autograph.)