CHAPTER

18

REGAINING CONTROL OF YOUR SYSTEM

In this chapter, techniques are explored that enable you quickly to regain control of your system after having discovered that someone has cracked it. This can be a delicate and complex operation if you are is to minimize damage and maximize the amount of knowledge to be learned about what happened and how. Because you are dealing with unknown software (the cracker's), there is no one right answer and there are no guarantees.

Part IV should first be read before you actually suffer a break-in so that you have an understanding of what to do to recover and have made some preparations in advance. On a test system, conduct some practice sessions recovering from a simulated attack. Some suggestions on how to do this are offered in "Fire Drills" on page 462.

Plan to disable important credentials quickly. This includes PGP, SSH, and SSL keys that might have been compromised. Change any passwords that might have been compromised through sniffing or social engineering. If the cracker might have gotten control of financial systems that print checks, ship merchandise, handle credit cards, or the like, be sure to block the flow of goods and money. This might include closing bank accounts, stopping outgoing shipments, etc. until a detailed analysis is completed.

The topics covered in this chapter include:

18.1 Finding the Cracker's Running Processes

Once you have detected that your system has been broken into, it would be very helpful to try to find any running processes that the cracker has left behind. Remember that any program on the system might have been compromised.

It is preferable to operate as an unprivileged user account that does not have access to anything important. This is because you do not know what programs have been compromised. For example, perhaps the cracker discovered that `/usr/local/bin` was mode 777 (world-writable) and he placed a compromised version of `date` in it. He might be waiting for something as innocent as root invoking `date` to get full control of the system—that is, "own it" in cracker parlance. By operating as that unprivileged user, clearly you limit the consequences of this or other actions.

Keep a "stealth" version of `ps` under an unassuming name. If you run an application, call it by that name. Do a `ps` of your system and note root programs such as `lpd` and `sendmail` as possible names to call your "stealth ps" executable. Some crackers might notice that `sendmail` should not have an argument of `axlww` so if you are feeling ambitious, grab the source of `ps` and tweak it to create a custom version that defaults to these flags. (The "`a`" flag requests all processes, not just yours; the "`x`" flag also includes daemons not associated with a terminal; the "`l`" flag requests long format to give more details; the "`w`" flag allows longer lines; and the second "`w`" allows unlimited lines.)

Back during my gray hat student days at Berkeley, one time I was operating covertly as root, repeatedly doing a `ps`[a] to watch for system administrators trying to detect us. I was "riding shotgun" while Doug Merritt installed a Trojan.

I noticed someone logged in on an administrator account do a `ps` and knew that we were detected. Twenty seconds later the administrator was using the `write` command to contact me, asking who I was.

Fortunately we were in a little-used terminal room and we were logged off and out of Evans Hall in 60 seconds flat! The lesson is that each side in this war should have been using stealth versions of `ps`. The better crackers now do this; you should too!

---

a. Most people do not know that the "command name" that `ps` (process status) displays is specified by the command's parent process and is arbitrary. In other words when you ask the shell to start `ps`, the shell *chooses* to claim that the child process name is ps.

A common cracker technique is to specify a different name such as `cc` or `assignment7`. The name (or inode number in pre-2.2 kernels) of the executable that Linux's `/proc/`*PID*`/exe` symbolic link points to cannot be spoofed, however, except with a kernel hack. A useful enhancement to `ps` would to be to warn if `exe` does not match the command name (recognizing that the shell may show the program name as `ps` instead of `/bin/ps`).

18.1.1 Handling Deleted Executables

One cracker trick is to remove the executable of a running program from the file system. Recall that this will cause the reference to the name of the file in its directory to be removed *but* the file still will exist until all programs that have the file open (as open file descriptors) close it.

A running program "in execution" is treated as an open file. Crackers know that the first thing most SysAdmins do when they detect an intrusion is to shut down the system, either to copy the disk for evidence or analysis or in the hope that the problem will go away. Of course, on a clean shutdown the program's execution will be stopped and the program's data blocks and inode freed. If the system is shut down abruptly, the cleanup will be done upon reboot by `fsck`.

The method for detecting these executing programs and making copies of them automatically (for analysis) is discussed in "Detecting Deleted Executables" on page 517. Any of these executables that are found are almost certainly Trojans, unless one of them is a program under development by a programmer whose building of a new version caused the deleted version to be removed from disk.

A very useful feature in the kernel is that the symbolic link in `/proc` to the executable is good *even though the original file has been removed from the file system!* This allows you to make a copy of the file for analysis as simply as

```
cp /proc/479/exe /home/samspade/del_cracker
```

In other words, if the cracker did

```
cd /tmp
.genie&
rm .genie
```

this technique still will recover a copy for you to analyze and present as evidence.

18.2 Handling Running Cracker Processes

At this point, it is assumed that you ran a covert and trusted `ps` program and it shows two processes that you are suspicious of, `/bin/ls` and `wizbang`. You are suspicious of `/bin/ls` because it has been running for a long time and there is no reason for a user to be doing something like

```
/bin/ls -R /
```

or similar that could explain this program running for so long. You are suspicious of `wizbang` because you are not aware of an application of this name.

The PID (process ID) of `/bin/ls` is 16887 so you use your covert `ls` command, say, monthly, to issue the command

```
cd /proc/16887
monthly -l
```

and it might show

```
-r--r--r--   1 root  root  0 May 17 00:49 cmdline
lrwx------   1 root  root  0 May 17 00:49 cwd -> /tmp
-r--------   1 root  root  0 May 17 00:49 environ
lrwx------   1 root  root  0 May 17 00:49 exe -> /tmp/.genie
dr-x------   2 root  root  0 May 17 00:49 fd
pr--r--r--   1 root  root  0 May 17 00:49 maps
-rw-------   1 root  root  0 May 17 00:49 mem
lrwx------   1 root  root  0 May 17 00:49 root -> /
-r--r--r--   1 root  root  0 May 17 00:49 stat
-r--r--r--   1 root  root  0 May 17 00:49 statm
-r--r--r--   1 root  root  0 May 17 00:49 status
```

Observe that the `exe` file is a symbolic link to the executable program that was invoked and it certainly does not point to `/bin/ls`. Very likely, this is a Trojan horse. Note that because these files are owned by root, this process is running as root. Note that the name of the executable, `/tmp/.genie`, is extremely suspicious. This is because it is highly unusual for root to be invoking executables that are found in `/tmp` and that the name begins with a ".", which means that a normal `ls` command will not show this file. You also could do a binary comparison with `/bin/ls` to convince yourself that it really is a different program with the following command:

```
cmp exe /bin/ls
```

The following output would be typical:

```
exe /bin/ls differ: char 25, line 1
```

Clearly, it is a different program. This is a Trojan horse!

It is assumed that you have a notebook and are taking notes of all your actions and discoveries. You will want to log the date and time in this notebook because it may be introduced into evidence in court at some future date. You may want to sign your entries too.

At this point you will want to note the PID of this Trojan horse and its executable name, `/tmp/.genie`. You will want to make a copy of this Trojan horse. If it is convenient, media, such as magnetic tape, floppy, or CD-RW, is recommended. This is because after it is written to, the media may be write-protected, labeled, and set aside. This way, it will survive even if some other cracker Trojan destroys the data on your disk.

It is very helpful if you already have a stealth copy of `tar` or some other program that is useful to copy files to your backup media. Assuming that your stealth version of `tar` is called

```
/home/larry/bin/feather
```

and you will be backing up to `/dev/fd0`. Issue the following command:

```
/home/larry/bin/feather -chvf /dev/fd0 /proc/16887/exe
```

The "`h`" flag causes `tar` to back up the file that any symbolic link, such as `/proc/16887/exe`, points to. This will back up the cracker's program even if `/tmp/.genie` (the copy in the disk-based file system) was removed. Remove this floppy from the drive, write protect it, and label it something like

```
/tmp/.genie -> /proc/16887/exe
cracker-deleted running program
2000/07/29
Trojan horse on
www.pentacorp.com
(signed Joe SysAdmin date)
```

At this point, you have several options regarding how to proceed and there is no one right answer. You simply could kill the process. It is suggested that you *not* do a `kill 16887` because that will send a terminate signal to the process and give it a chance to catch the signal and do whatever it wants. It might remove all evidence of itself. It might send e-mail to its owner warning him that he has been discovered. It might remove all of your data from the disk. Instead, use the following that will terminate it with no warning and without offering the Trojan horse a chance to take any action at all:

```
kill -9 16887
```

A really good cracker will have another process monitor this process and detect its demise. This could be done by the other process being this process's parent and using a `wait()` system call or SIGCHLD signal. It simply could do `kill(16887, 0)` periodically until it returns `-1`. It could set up a pipe, named or unnamed, with `16887` and detect the broken pipe when `16887` dies.

A second opportunity, for the daring, is to attach to the cracker's running process with a debugger and attempt to analyze it. (You might want to back up critical files first.)

You first might see if the binary has a symbol table. The command to do this would be the following:

```
file /proc/16887/exe
```

The result would be

```
/proc/16887/exe: symbolic link to /tmp/.genie
```

Oops, forgot that it is a symbolic link. You will use `find`'s dash flag to work around this in just a moment. First, make a copy of it, because it might try to remove its own disk copy to escape analysis and for possible use as evidence in court. The following will work, even if the copy on disk already has been removed:

```
cp /proc/16887/exe $HOME/Trojan
```

Using the `strings` program on it will display all ASCII strings in the file; this will give clues about what it does. The command would be

```
strings $HOME/Trojan | more
```

Try the following:

```
file -L /proc/16887/exe
```

The result might be the following:

```
/proc/16887/exe: ELF 32-bit LSB executable, Intel 80386,
version 1, dynamically linked (uses shared libs), not stripped
```

The `not stripped` is what we are hoping for. It means that the executable has not been stripped of its symbol table. (If it has been stripped of symbols, the analysis will be much more difficult.) The symbols in it may be listed with the following command:

```
nm $HOME/Trojan | more
```

An experienced programmer will get a good idea as to what it is doing by seeing which standard Linux functions and system calls it is using.

To have the standard Linux debugger, `gdb`, attach to a running process, you need to pass the executable name and the PID (process ID) to it. In our example, the command to issue would be

```
gdb /proc/16887/exe 16887
```

The `.genie` program will be stopped and *you* will be in control of it from this point on. The typical output from this `gdb` command might be the following:

```
GNU gdb 4.18
...
Attaching to program: /tmp/.genie, Pid 16887
Reading symbols from /lib/libdb.so.3...done.
Reading symbols from /lib/libresolv.so.2...done.
Reading symbols from /lib/libnsl.so.1...done.
Reading symbols from /lib/libc.so.6...done.
Reading symbols from /lib/ld-linux.so.2...done.
Reading symbols from /lib/libnss_files.so.2...done.
Reading symbols from /lib/libnss_nisplus.so.2...done.
Reading symbols from /lib/libnss_nis.so.2...done.
0x4012354e in __select () from /lib/libc.so.6
(gdb)
```

At this point, the first command that you would want to issue is `bt` to generate a backtrace. This will show which routine is being called by which. Frequently, this will give a good idea of what might be going on inside the program. This is what you might see:

```
(gdb) bt
#0  0x4012354e in __select () from /lib/libc.so.6
#1  0x5 in _wish ()
#2  0x400901eb in __libc_start_main (main=0x805eed0 argc=3,
    argv=0xbffff9b4, init=0x804a054,
    rtld_fini=0x4000a610 <_dl_fini>, stack_end=0xbffff9ac)
    at ../sysdeps/generic/libc-start.c:90
(gdb)
```

This tells us a number of interesting things. The `__select ()` routine is the one currently running. This would be the `select()` system call that causes the program to wait until I/O completes on any of a specified set of open files (file descriptors). Usually, at least one of these open files would be a network file. The `select()` system call was invoked by a routine called `wish()`.

This executable has a program name of `.genie`, a routine called `wish()`, and it is probably waiting for a connection from the network. A good guess would be that it is waiting for a cracker to connect to it via TCP or UDP and give it commands to execute. But wait, there's more. From another window, let us see what files it has open. Issue the commands

```
cd /proc/16887
monthly -l fd
```

The following would be typical:

```
lr-x------ 1 root  root  64 May 17 01:55 0 -> /dev/null
l-wx------ 1 root  root  64 May 17 01:55 1 -> /dev/null
l-wx------ 1 root  root  64 May 17 01:55 2 -> /dev/null
lrwx------ 1 root  root  64 May 17 01:55 3 -> socket:[17095]
```

File descriptors (open file numbers) `0`, `1`, and `2` are standard input, standard output, and standard error. All of them are directed to `/dev/null`. The presence of `/dev/null` as standard input and standard output indicates that the program is operating as a daemon; that is, a long running process that is not associated with any user tty. File descriptor `3` is a socket, e.g., a network connection. Issue a `netstat -avp` command. The `netstat` command gives network status information. The `-a` flag lists all network ports that are open, even those that are not currently connected to a remote system. The `-v` flag adds verbosity. The `-p` flag will cause `netstat` to list the PID and name of each process (program) that has a network port open. The `-p` flag is a new and very useful but many people do not know that it is available. The `-p` flag does require root access. When `netstat -avp` is issued the following is shown:

```
Proto Recv-Q Send-Q Local Address Foreign Address State
PID/Program name
...
tcp        0      0 *:1243        *:*          LISTEN
16887//bin/ls
...
```

This shows that the Trojan is listening on TCP port 1243.

> The `Foreign Address` field identifies the remote host and port that the program is communicating with, if it has an established TCP connection. If this is shown for a cracker process, this would be the system either that he is attacking *from* or that he is attacking from your system.

In this example, all that is shown in the `Foreign Address` field is `*:*`, indicating that there is no such connection. Because the protocol is shown as `tcp`, this means that this program is operating as a server waiting for a client to connect. Be *very* suspicious of programs using ports above 1023 that are *not* connected to well-known ports on remote systems. Thus, this program is suspicious. There only are a few legitimate widely used services on ports above 1023. (1080 for SOCKS, 6000 for X, 6010 for SSH-wrapped X, and 2049 for NFS are common.) Double-check this port by issuing the following command:

```
grep 1243 /etc/services
```

The `grep` did not find anything. Run the ports program that is discussed in “Turn Off Unneeded Services” on page 82 and observe the output.

```
TCP
Lcl port        Rmt port  Status     Rmt IP   Rmt host
 ...
* 1243=subseven    0=zero  0A=LISTEN  0.0.0.0  local
*** cracker server
...
```

The `ports` program instantly identified the Trojan horse from its default port number. Had your cracker chosen to alter the port that your version of it listened on, this might have been more difficult, though `ports` will flag any TCP connection on a high port in a listen state. Most script kiddies do not bother even to strip the symbol table from the executable. In this case you can have a look at the symbols using the `nm` program in the usual invocation.

```
nm suspicious_file | less
```

Even if the symbol table was stripped out, almost every program has ASCII strings in it that will give clues to what it does and what its origin is. The `strings` program searches for sequences of printable characters and prints these out. The `-a` flag will print out all strings, not just those in the text and data portions. Typical usage would be

```
strings -a suspicious_file | less
```

It can be useful for running programs too. To analyze running process number 86, use the following command:

```
strings -a /proc/86/exe | less
```

Many of the fancier cracker tools have help messages that give clues to their capabilities.

You could get braver and actually step through the Trojan horse. Prior to doing this, it would be a good idea to back up the system because you are playing with a live "bomb" at this point and it might go off. Once you have attached to it with `gdb` (or my favorite debugger, `ddd`), it is stopped ("frozen") until and unless you allow it to continue. The `ddd` debugger is available from

```
http://www.gnu.org/software/ddd/
```

While `.genie` is stopped, it is not possible for it to restart on its own, so it is somewhat safe to create a backup of the current system or continue with other things at this time. There might be Trojans anywhere, so you should not trust the system until all of these are analyzed, as discussed in "Finding Cracker-Altered Files" on page 559.

If you do want to step through it, you could `telnet` to it. In this example, you would do this via the following command from a different window:

```
telnet www.pentacorp.com 1243
```

Then, in the `gdb` (or `ddd` window) step through the code. Check each instruction before it is executed to ensure that it will not do something harmful like

```
execl("rm", "/bin/rm", "-rf", "/", 0);
```

If in doubt, terminate the debugging session. The safest way is to first issue the following command. (In our example, the Trojan's PID is `16887`.)

```
kill -9 16887
```

A similar analysis could be done of the `wizbang` process.

18.2.1 Popular Trojan Horses

Some of the most commonly seen Trojan horses are discussed here. They give a starting place for searching for Trojans if you suspect that you might have one or more and they also gives a "feel" for types of Trojans to expect. Following the security mailing lists, news groups, and Web sites (all covered in Appendix A) is critical as new exploits are discovered weekly.

One way to detect Trojans is with the use of Tripwire, which is discussed in "Tripwire" on page 511. The periodic use of `tar -d` or `rpm` works well too. These latter two methods are discussed in "Finding Cracker-Altered Files" on page 559. Additionally, scanning your system for open ports with a careful comparison to past results from `netstat` or `ports` should show any suspicious ports that have not been open in the past.

1. The `fingerd` program commonly is replaced with a version containing a Trojan. Tripwire normally should detect this because the full pathname for `fingerd` should be specified in `/etc/inetd.conf` or implied by its use of `tcpd` and it is assumed that Tripwire is configured to watch all system directories.

2. The `inetd` process is the Internet "superserver" daemon that starts most network services based on requests from remote system. It is extremely easy and fast to create a Trojan with `inetd` once a cracker has root access. All a cracker running as root needs to do is

   ```
   echo ingreslock stream tcp wait root /bin/sh -i > /tmp/tim
   /usr/sbin/inetd /tmp/tim
   /bin/rm /tmp/tim
   ```

   The ingreslock service seems popular; perhaps some firewalls allow it. However, any unused TCP service may be used. Certainly, because a cracker needs to be root to to create this back door, he could add new service names to `/etc/services` or alter the port number of an existing service.

   The best way to detect this compromise is to know what services *should* be running on your systems and use `ports` or `netstat -atuvp` daily or weekly, possibly comparing the results against previous results stored in a file.

   Certainly, you can test the exploit the way crackers use it via

   ```
   telnet yoursys.com ingreslock
   ```

   and see if you get a root shell but this could be dangerous. Although the Trojan just described is unsophisticated (though far from harmless) a later version may require a password and damage the system if the correct password is not provided. Thus, are the risks of using untrusted software.

Tripwire will *not* detect this compromise because `inetd` and `/etc/inetd.conf` have not been altered. You could count the number of `inetd` processes running via

```
ps -axlww | grep inetd | grep -v grep | wc -l
```

but expect periodic false positives when `inetd` forks prior to doing an exec to start a requested service.

Note that the use of a second running `inetd` process using a rogue `inetd.conf` file allows many other exploits with little effort or thought required by a cracker. He could, for example, install on the system a compromised `fingerd` as discussed earlier in this section.

However, to avoid detection by Tripwire and other file system comparison methods he removed `/tmp/tim`.

3. Some crackers will create a `/usr/sbin/inetd` executable with a Trojan built into `inetd` itself. Usually this version will listen on an additional TCP port, such as ingreslock, and provide a root shell to anyone who uses `telnet` to connect to the magic port. This Trojan would be detected with Tripwire.

However, the cracker could hide this Trojan even from Tripwire or `tar -d`. This is done by invoking the Trojan version as `/usr/sbin/inetd` and then removing it from disk and putting the "real" version back in its place. Thus, a

```
ls -l /proc/PID/exe
```

will point back to `/usr/sbin/inetd` and Tripwire will show that `/usr/sbin/inetd` is identical to Tripwire's stored checksum; likewise, `tar -d` will show that `/usr/sbin/inetd` matches the backup copy if the cracker sets the create time back, though some crackers will not bother. One way to detect this subtle problem is to issue the command

```
cmp /proc/PID/exe /usr/sbin/inetd
```

If the cracker actually removed the executable from disk then the `getdel` script will detect this automatically and alert you. This script is discussed in "Detecting Deleted Executables" on page 517.

4. Some crackers will install versions of `/bin/ls` and `/bin/ps` with Trojans that will not list any of the cracker's files nor show the cracker's processes. This technique is used by crackers quite often. Even the script kiddies use it. Tripwire will detect these Trojans, of course.

It is helpful for you to have backup versions of these programs under different names stored in some innocuous place. In a pinch, you could use different programs to perform these same functions such as the `file` program for `ls`. Instead of using a possibly compromised `ps`,

```
file /proc/[0-9]*/exe
```

will list processes.

5. Sometimes `/bin/login` is replaced with one that has a "back door" that allows a cracker to become root by entering some word in place of an account listed in `/etc/passwd`.

18.3 Drop the Modems, Network, Printers, and System

Once you have detected that your system has been broken into, you must decide how much data to gather on the intrusion before stopping further damage. It is assumed that you already have gathered as much data as you dare by this point. Now you need to get the intruder out of your system. There are two parts to this. The first is preventing him from accessing your system, and I will address that here. (Later you need to remove what he has left behind in the way of compromised programs and plug security holes.)

Many system administrators forget that the most effective way to throw the intruder out is to sever the connections between the computer and the outside world. For most, this means disconnection from the network and modems.

You should plan in advance for your response to an intrusion because you will want to act quickly when an intrusion is discovered. A formal *Security Procedures* or *Intrusion Response* manual is recommended. Writing this document in advance is valuable because in the "heat of battle" you might not have the luxury of time nor be able to think as clearly.

This is why every pilot carries an *Emergency Procedures* document on his aircraft that lists the responses to common emergencies that have been thought out carefully in advance and are based on past experiences. On your next airline flight, on the way out you might ask the flight crew to show you theirs. It will be instructional.

In many cases, the fast way to sever connectivity is to disconnect the modems from either the phone lines or electrical power. For a small setup, simply unplugging the phone cable from the modem or phone jack will do fine. I do this when I am under attack. For larger setups, having all the modems' power plugs in one or two power strips or UPS (Uninterruptible Power Supply) units will allow throwing one or two power switches to turn them off, this being easier than unplugging lots of phone cables and then later trying to figure out which one went where. (Having two sets of power strips or UPS units provides redundancy.)

If the intruder might have gotten in through your LAN, simply unplug the network cable from the computer. I recommend this solution so that you do not disrupt the rest of

the network. If you have any local users through serial connections who could be the culprit, you might need to unplug these cables. Keep in mind that it is likely that the intruder has broken into other of your systems too, particularly if they are configured similarly. If you think that this is likely, it might be better to disconnect your entire network from the Internet.

Many SysAdmins forget that the fastest way to shut out intruders is to shut down the system or take it to single-user mode. Once you capture evidence of the break-in, either of these is strongly preferred. The advantage of first dropping the modems or network is that it prevents most of the possible further harm to your system while allowing you to see what processes the cracker left running. Seeing these processes, obviously, is important to tracking down the cracker's methods, damage, and origin.

Now that you have collected all the information that you can about the intruder's current connections, it is time to shut the system down and boot from a disk or tape that is known to have no Trojan horses. An orderly shutdown might alert any Trojan horses that you have missed. It is hard to be confident that you have detected and killed all of them. Because of this, it might be better to stop the system abruptly.

First try to close any database operations becaues these can be delicate. Then issue the `sync` command from a nonprivileged account and wait two seconds. Then press the computer's reset button or interrupt power. The slight risk of file system corruption probably is less than the risk of alerting a Trojan horse that might destroy the entire file system or send e-mail to the cracker alerting him that he might have been discovered.

Before coming up multi-user, inspect `/var/spool/mqueue` for possible cracker-generated e-mail that he might be using to alert himself that he has been discovered. If you suspect that he could be using an idle account, issue the command

```
ls -ltr /var/spool/mail
```

and observe which accounts have the most recent e-mail. Are any of these accounts unused or accounts of people on vacation? Certainly, someone could be receiving e-mail while on vacation. Personal accounts' e-mail should not be looked at unless the "owner" is unavailable and only with *written* permission from management.

There might even be laws in your jurisdiction forbidding this on the basis of "privacy." Having a written policy in advance that "all e-mail and disk files are subject to inspection as needed for system administration" might grant you the authority. This is another issue to work out with management, Human Resources (Personnel), and the Legal Department in advance.

After shutting the system down it is time to switch to Auxiliary Control. Setting this up was discussed in "Switch to Auxiliary Control (Hot Backups)" on page 437. If you do not have it then use Tripwire or the `tar` technique to find what was altered and correct. Failing this, it is time for backup tapes or CD-RWs.