Unix Forensics

Purpose

This paper explains and demonstrates popular forensics techniques on Unix based systems. The goal is to enable an experienced systems administrator to determine if a compromise has taken place, rebuild the events, and adjust security policies accordingly. The paper is split into three sections:

I. The Beginning of a Compromise

A. Port Scanning, Buffer Overflows, Rootkits and Loadable Kernel Modules

The following articles are taken from the Honeynet Project and describe the most common methods of attacking a Unix system.

Know Your Enemy
The Tools and Methodologies of the Script Kiddie
Honeynet Project
http://project.honeynet.org

My commander used to tell me that to secure yourself against the enemy, you have to first know who your enemy is. This military doctrine readily applies to the world of network security. Just like the military, you have resources that you are trying to protect. To help protect these resources, you need to know who your threat is and how they are going to attack. This article, the first of a series, does just that, it discusses the tools and methodology of one of the most common and universal threats, the *Script Kiddie*. If you or your organization has any resources connected to the Internet, this threat applies to you.

The Know Your Enemy series is dedicated to teaching the tools, tactics, and motives of the blackhat community. Know Your Enemy: II focuses on how you can detect these threats, identify what tools they are using and what vulnerabilities they are looking for. Know Your Enemy: III focuses on what happens once they gain root. Specifically, how they cover their tracks and what they do next. Know Your Enemy: Forensics covers how you can analyze such an attack. Know Your Enemy: Motives, uncovers the motives and psychology of some members of the black-hat community by capturing their communications amongst each other. Finally, Know Your Enemy: Worms at War covers how automated worms attack vulnerable Window systems.

Who is the Script Kiddie

The script kiddie is someone looking for the easy kill. They are not out for specific information or targeting a specific company. Their goal is to gain root the easiest way possible. They do this by focusing on a small number of exploits, and

then searching the entire Internet for that exploit. Sooner or later they find someone vulnerable.

Some of them are advanced users who develop their own tools and leave behind sophisticated backdoors. Others have no idea what they are doing and only know how to type "go" at the command prompt. Regardless of their skill level, they all share a common strategy, randomly search for a specific weakness, then exploit that weakness.

The Threat

It is this random selection of targets that make the script kiddie such a dangerous threat. Sooner or later your systems and networks will be probed, you cannot hide from them. I know of admins who were amazed to have their systems scanned when they had been up for only two days, and no one knew about them. There is nothing amazing here. Most likely, their systems were scanned by a script kiddie who happened to be sweeping that network block.

If this was limited to several individual scans, statistics would be in your favor. With millions of systems on the Internet, odds are that no one would find you. However, this is not the case. Most of these tools are easy to use and widely distributed, anyone can use them. A rapidly growing number of people are obtaining these tools at an alarming rate. As the Internet knows no geographic bounds, this threat has quickly spread throughout the world. Suddenly, the law of numbers is turning against us. With so many users on the Internet using these tools, it is no longer a question of if, but when you will be probed.

This is an excellent example of why security through obscurity can fail you. You may believe that if no one knows about your systems, you are secure. Others believe that their systems are of no value, so why would anyone probe them? It is these very systems that the script kiddies are searching for, the unprotected system that is easy to exploit, the easy kill.

The Methodology

The script kiddie methodology is a simple one. Scan the Internet for a specific weakness, when you find it, exploit it. Most of the tools they use are automated, requiring little interaction. You launch the tool, then come back several days later to get your results. No two tools are alike, just as no two exploits are alike. However, most of the tools use the same strategy. First, develop a database of IPs that can be scanned. Then, scan those IPs for a specific vulnerability.

For example, lets say a user had a tool that could exploit imap on Linux systems, such as imapd_exploit.c. First, they would develop a database of IP addresses that they could scan (i.e., systems that are up and reachable). Once this database of IP addresses is built, the user would want to determine which systems were running Linux. Many scanners today can easily determine this by sending bad packets to a system and seeing how they respond, such as Fyodor's nmap. Then, tools would be used to determine what Linux systems were running imap. All that is left now is to exploit those vulnerable systems.

You would think that all this scanning would be extremely noisy, attracting a great deal of attention. However, many people are not monitoring their systems, and do not realize they are being scanned. Also, many script kiddies quietly look for a single system they can exploit. Once they have exploited a system, they now use this system as a launching pad. They can boldly scan the entire Internet without fear of retribution. If their scans are detected, the system admin and not the black-hat will be held liable.

Also, these scan results are often archived or shared among other users, then used at a later date. For example, a user develops a database of what ports are open on reachable Linux systems. The user built this database to exploit the current imap vulnerability. However, lets say that a month from now a new Linux exploit is identified on a different port. Instead of having to build a new database (which is the most time consuming part), the user can quickly review his archived database and compromise the vulnerable systems. As an alternative, script kiddies share or even buy databases of vulnerable systems from each other. You can see examples of this in Know Your Enemy: Motives. The script kiddie can then exploit your system without even scanning it. Just because your systems have not been scanned recently does not mean you are secure.

The more sophisticated black-hats implement trojans and backdoors once they compromise a system. Backdoors allow easy and unnoticed access to the system whenever the user wants. The trojans make the intruder undetectable. He would not show up in any of the logs, systems processes, or file structure. He builds a comfortable and safe home where he can blatantly scan the Internet. For more information on this, check out Know Your Enemy: III.

These attacks are not limited to a certain time of the day. Many admins search their log entries for probes that happen late at night, believing this is when black-hats attack. Script kiddies attack at any time. As they are scanning 24hrs a day, you have no idea when the probe will happen. Also, these attacks are launched throughout the world. Just as the Internet knows no geographical bounds, it knows no time zones. It may be midnight where the black-hat is, but it is 1pm for you.

This methodology of scanning for vulnerable systems can be used for a variety of purposes. Recently, new Denial of Service attacks have been reported, specifically DDoS (Distributed Denial of Service attacks). These attacks are based on a single user controlling hundreds, if not thousands of compromised systems throughout the world. These compromised systems are then remotely coordinated to execute Denial of Service attacks against a victim or victims. Since multiple compromised systems are used, it is extremely difficult to defend against and identify the source of the attack. To gain control of so many systems, script kiddie tactics are often employed. Vulnerable systems are randomly identified and then compromised to be used as DDoS launching pads. The more systems compromised, the more powerful the DDoS attack. One example of such an attack is 'stacheldraht',. To learn more about Distributed Denial of Service attacks and how to protect yourself, check out Paul Ferguson's site Denialinfo

The Tools

The tools used are extremely simple in use. Most are limited to a single purpose with few options. First come the tools used to build an IP database. These tools are truly random, as they indiscriminently scan the Internet. For example, one tool has a single option, A, B, or C. The letter you select determines the size of the network to be scanned. The tool then randomly selects which IP network to scan. Another tool uses a domain name (z0ne is an excellent example of this). The tools builds an IP database by conducting zone transfers of the domain name and all sub-domains. User's have built databases with over 2 million IPs by scanning the entire .com or .edu domain. Once discovered, the IPs are then scanned by tools to determine vulnerabilities, such as the version of named, operating system, or services running on the system. Once the vulnerable systems have been identified, the black-hat strikes. For a better understanding of how these tools are used, check out Know Your Enemy: Forensics.

How to Protect Against This Threat

There are steps you can take to protect yourself against this threat. First, the script kiddie is going for the easy kill, they are looking for common exploits. Make sure your systems and networks are not vulnerable to these exploits. Both www.cert.org and www.ciac.org are excellent sources on what a common exploit is. Also, the listserv bugtraq (archived at securityfocus.com) is one of the best sources of information. Another way to protect yourself is run only the services you need. If you do not need a service, turn it off. If you do need a service, make sure it is the latest version. For examples on how to do this, check out Armoring Solaris, Armoring Linux or Armoring NT.

As you learned from the tools section, DNS servers are often used to develop a database of systems that can be probed. Limit the systems that can conduct zone transfers from your Name Servers. Log any unauthorized zone transfers and follow up on them. We highly recommend upgrading to the latest version of BIND (software used for Domain Name Service), which you can find at www.isc.org/bind.html. Another option is to use djbdns as a replacement for BIND. Last, watch for your systems being probed. Once identified, you can track these probes and gain a better understanding of the threats to your network and react to these threats.

Conclusion

The script kiddie poses a threat to all systems. They show no bias and scan all systems, regardless of location and value. Sooner or later, your system will be probed. By understanding their motives and methods, you can better protect your systems against this threat.

Related Articles

“Solaris Loadable Kernel Modules”
http://packetstormsecurity.org/groups/thc/slkm-1.0.html

“Kernel Rootkits Explained”
http://www.itsolvers4u.com/security/tools/rootkits/kernel_rootkits_explained.htm

“Buffer Overflows: Attacks and Defenses for the Vulnerability of the Decade”
http://www.cert-rs.tche.br/docs/discex00.pdf

“Know Your Enemy: Motives”
http://www.honeynet.org/papers/motives/

“Analysis of DdoS Utilities”
http://staff.washington.edu/dittrich/misc/ddos/

II. The Signs of a Compromise

Depending on the attack, there are a couple of quick checks a system administrator can make on a running system. If any one of these turns out to be true, then it is possible that NO OUTPUT can really be trusted on the server. With this in mind, it is a good idea to have a clean set of binaries available. It is VERY DIFFICULT to compile static binaries for Solaris. Some GNU binaries can be compiled while others can't. Below are statically (GNU) compiled **ls** (coreutils) and **find** (findutils) commands.

```
bash-2.05b# cd /cdrom/forensic/gnu/bin
bash-2.05b# file find ls
find:       ELF 32-bit MSB executable SPARC Version 1, statically linked, not stripped
ls:         ELF 32-bit MSB executable SPARC Version 1, statically linked, not stripped
```

A rescue disk should contain the following binaries (static if at all possible):

- ls
- ps
- find
- netstat
- lsof
- apptrace or truss
- md5
- ssh
- wget
- mdb

A good paper on how to compile static binaries (using a **stubs.o** file) can be found at:
http://bullwinkle.deer-run.com/~hal/sol-static.txt

You can also download an ISO image of the forensics disc I created:
http://www.ufsdump.org

A. The Intrusion

More often than not, attackers do not do the best job of cleaning up after themselves. Here is an entry in the **/var/adm/messages** from what appears to be a buffer overflow attempt on the **dtspc** daemon.

```
bash-2.05b# more messages
Jul  9 03:59:56 seatac inetd[5744]: [ID 161378 daemon.error] dtspc/tcp: bind: Address
already in use
Jul  9 03:59:57 seatac inetd[5746]: [ID 801587 daemon.error] /tmp/x: No such file or
directory
Jul  9 04:09:57 seatac inetd[5744]: [ID 161378 daemon.error] dtspc/tcp: bind: Address
already in use
Jul  9 04:09:57 seatac syslogd: /var/adm/utmpx not owned by root or not mode 644.\nThis
file must be owned by root and not writable by\nanyone other than root.  This alert is
being dropped because of\nthis problem.
```

It appears that the attacker tried the attack first at 3:59:56. It failed because the file (**/tmp/x**) was not available. This looks like it was attempting to bind a shell to an arbitrary port (usually 1524) using another instance of **inetd** and then try to connect to that port. But, the attacker had a typo somewhere because most buffer overflow attacks like this create a **/tmp/.x**.

Here is the **snort** log:

```
bash-2.05b# cd /var/log/snort/alert/12.22.215.168
bash-2.05b# more TCP:6112-4633
```

```
[**] EXPLOIT CDE dtspcd exploit attempt [**]
07/09-03:59:46.589981 212.93.142.186:4633 -> 12.22.215.168:6112
TCP TTL:46 TOS:0x0 ID:56 IpLen:20 DgmLen:1500 DF
***A**** Seq: 0xCD2DA989  Ack: 0x24E42EA6  Win: 0x16D0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 8694576 7290334
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

B. Bound Ports

Remote attacks require a control connection in order for an attacker or DDoS client to communicate back with a master server.

```
bash-2.03# /usr/ucb/netstat

TCP: IPv4
   Local Address        Remote Address    Swind Send-Q Rwind Recv-Q  State
-------------------- -------------------- ----- ------ ----- ------ -------
nelson.32773         216.115.95.70.6669    8167      0 24820      0 ESTABLISHED
nelson.32774         nonpublic.vhosts.de.28890 32120      0 24820      0 ESTABLISHED
nelson.telnet        12.22.215.58.43612   24820      1 24820      0 ESTABLISHED
```

The first two connections here are suspect. This server only serves out HTTP (port 80) connections. The two connections on ports 32773 and 32774 appear to be made by a DDoS agent.

Here is another example:

```
westwood.33174   Amsterdam2.NL.EU.undernet.org.6667  8760      0 24820      0 ESTABLISHED
```

Using **lsof**, we can determine what processes are bound to those ports.

```
bash-2.03# lsof -i
<snip>

ttymon 368 root 5u IPv4 0x30000b73a90 0t11894 TCP nelson:32773->216.115.95.70:6669
(ESTABLISHED)
ttymon 368 root 6u IPv4 0x30000a7ccc0 0t2360  TCP nelson:32774->nonpublic.vhosts.de:28890
(ESTABLISHED)

<snip>
```

The **ttymon** daemon is a serial port monitor, and should not have anything to do with a TCP/IP port. Therefore, we can assume that this version of **ttymon** may not be what it appears to be.

C. Rogue Processes

Most attackers have some kind of processes running for collection of data (packet sniffing, keystrokes) and communications.

```
bash-2.05b# ps
<snip>

root  1587  1553  0 17:50:35 ?        0:07 /opt/SUNWut/jre/bin/../
root  1521  1487  0 17:50:30 ?        0:00 /opt/SUNWut/lib/utxexec
root  1522  1485  0 17:50:30 ?        0:00 /opt/SUNWut/lib/utxexec /
root  1523  1486  0 17:50:30 ?        0:00 /opt/SUNWut/lib/utxexec /
root  1553  1521  0 17:50:32 ?        0:00 /bin/ksh -p /etc/opt/SUNWut/insertcard.start
root  1570  1553  0 17:50:34 ?        0:00 /usr/openwin/bin/xlogo -geometry 1x1+0+0
root  1686     1  0 00:13:51 ?        0:05 /usr/java/bin/../bin/../
root  1643     1  0 22:42:17 ?        0:03 /usr/local/sbin/sshd -q -p 22
```

```
root  1649     1  0 22:42:37 ?        0:00 pageout
root  1700  1643  0 00:14:44 ?        0:01 /usr/local/sbin/sshd -q -p 22
root  1647     1  0 22:42:26 ?        0:02 /usr/local/sbin/sshd -q -p 202
root  1645     1  0 22:42:21 ?        0:03 /usr/local/sbin/sshd -q -p 119

<snip>
```

The **pageout** process (PID 1649) is suspect here. The "real" **pageout** process is a kernel system thread which runs in a different scheduling class (SYS) as opposed to this **pageout** which ruins in TS/IA (user).

Back to the **ttymon** process mentioned in **Part B** of this section. The trojaned **ps** only shows two instances of **ttymon**.

```
bash-2.05b# ps -ef
root   343   338  0 13:16:43 ?        0:00 /usr/lib/saf/ttymon
root   379     1  0 13:20:01 ?        0:02 /usr/lib/saf/ttymon
```

Using **lsof**, we see 3 instances of ttymon.

```
bash-2.05b# lsof
<snip>
ttymon     343    root  txt   VREG          32,0    71168  278183 /usr/lib/saf/ttymon
<snip>
ttymon     379    root  txt   VREG          32,0    71168  278183 /usr/lib/saf/ttymon
<snip>
ttymon     713    root  cwd   VDIR          32,0      512   48208 /
usr/share/man/.man3/white
ttymon     713    root  txt   VREG          32,0   914300   48216 /
usr/share/man/.man3/white/ttymon
```

The third instance of **ttymon** is opening a file in an obscure location (**.man3**?). Here are the contents of that directory:

```
bash-2.05b# ls -l
total 2656
-rw-r--r--   1 root     other       4524 Jan 27  2003
-rw-r--r--   1 root     other         16 Jan 29  2003 Resolve.ck
-rw-r--r--   1 root     other         16 Jan 29  2003 Resolve.ck~bak
-rwxr-xr-x   1 root     root        8584 Jan 10  2002 a.out
-rwxr-xr-x   1 root     root        1256 Apr 24  2002 cfbotchk
-rwxr-xr-x   1 root     root          79 Apr 24  2002 crom
-rw-------   1 root     other        492 Feb  7  2003 filechan
drwxr-xr-x   3 root     root         512 Jun 18  2002 filesys
-rw-------   1 root     root        1676 Feb  7  2003 fileuser
-rw-------   1 root     other       3612 Feb  7  2003 fileuser~bak
drwxr-xr-x   3 root     root         512 Jun 18  2002 help
-rwxr-xr-x   1 root     root         400 Jun 12  2002 motd
-rwxr-xr-x   1 root     root      222608 Mar 20  2002 pico
-rw-r--r--   1 root     other          4 Feb  7  2003 pid.white
drwxr-xr-x   2 root     root         512 Jun 18  2002 text
drwxr-xr-x   2 root     root         512 Feb  7  2003 tmp
-rwxr-xr-x   1 root     root      914300 Dec 13  2002 ttymon
-rwxr-xr-x   1 root     root      154372 Oct 14  2002 white.tcl.ice
```

It appears that ttymon is some type of IRC bot. After some googling, it appears that the (ICE) protocol is used to send command strings to IRC bots (egg drops that act as a arc user, and keep the channel open also used to automatically respond to specific commands like "**!info**"). This server was idle when it was discovered. It is possible that it could have been nothing more than a chat server.

III. The Forensics

Unix Forensics

Once an intrusion has been detected, the forensics process can begin. The following are scenarios in which you must determine how to do forensics:

- production system/test system
- disaster recovery
- schedule downtime
- “clean” comparison system or trusted utilities

A. Searching for Rootkits and Trojan Binaries

Chances are that the attacker has placed a root kit on the machine. Many of the system auditing binaries can't be trusted. Using some utilities, we can determine if the binaries are trojaned. Here are the contents of a standard root kit:

```
bash-2.05b# ls
README          find            l0gin.new       me              packet          setup.sh        sys222
bd2             fix             le              mech.levels     pico            sl4             sys222.conf
bot2            idrun           log             mech.pid        ps              snif            tcpd
check           idsol           ls              mech.session    sec             sniff-100mb     zap3
emech233.users  l0gin.kit       m               netstat         secure.sh       sniff-10mb
```

A.1 Quick testing with ls and ldd:

```
bash-2.05b# -> ldd netstat
      libc.so.1 =>       /usr/lib/libc.so.1
      libdl.so.1 =>      /usr/lib/libdl.so.1
      /usr/platform/SUNW,Ultra-5_10/lib/libc_psr.so.1
bash-2.05b# -> ldd /bin/netstat
      libdhcpagent.so.1 =>     /usr/lib/libdhcpagent.so.1
      libcmd.so.1 =>     /usr/lib/libcmd.so.1
      libsocket.so.1 =>  /usr/lib/libsocket.so.1
      libnsl.so.1 =>     /usr/lib/libnsl.so.1
      libkstat.so.1 =>   /usr/lib/libkstat.so.1
      libc.so.1 =>       /usr/lib/libc.so.1
      libdl.so.1 =>      /usr/lib/libdl.so.1
      libmp.so.2 =>      /usr/lib/libmp.so.2
      /usr/platform/SUNW,Ultra-5_10/lib/libc_psr.so.1
```

The trojan binary links to less shared libraries.

```
bash-2.05b# ls -l netstat
-r-xr-sr-x   1 root     other      55168 Feb  5  2003 netstat
bash-2.05b ls -l /bin/netstat
-r-xr-sr-x   1 root     sys        61912 May 23  2002 /bin/netstat
```

The trojan version of **netstat** is NOT the same size as the good version of **netstat**.

A.2 Advanced testing with truss

Trojan rootkits will often call hidden config files. Tracing system calls is a good way to discover where these files exist which, in turn, can give more information about the intrusion.

```
bash-2.05b# truss -f -t open,stat -o out ./netstat
bash-2.05b# more out
<snip>

open("/usr/lib/libX.a/bin/netstat", O_RDONLY)  = 4
open("/usr/lib/libX.a/uconf.inv", O_RDONLY)    = 5
```

```
<snip>
```

The problem with this is that **netstat** is appears to look for a directory called **libX.a**. This should be a file! The trojan **netstat** calls the real **netstat** which has been copied to the l**ibX.a/bin** directory. Here is a listing of the **libX.a** directory:

```
bash-2.05b# ls libX.a
bin          loadbnc     passgen     patch.sol7  syn          utime
crt          new         patch.sol5  patch.sol8  td           wipe
l            oldsuper    patch.sol6  ssh-dxe     uconf.inv
```

After some googling, it appears that **libX.a** is a default install directory for the "X-Org SunOS Rootkit v2.5D". Particulary alarming is the "syn" program which is a scanner that is part of the DDoS "Tribal Flood Network" (TFN).

More info on TFN:
http://www.packetstormsecurity.org/distributed/TFN_toolkit.htm

Here is a snippet from the install script for the kit:

```
echo "${WHI}***${DWHI} Insert Rootkit Password : "
read PASSWD
echo "${WHI}***${DWHI} Using Password $PASSWD"
./pg $PASSWD >/etc/lpd.config
PASS=$PASSWD
echo "su_pass=`./rpass`" >>x.conf2
echo "${WHI}***${DWHI} Insert Rootkit SSH Port : "
read PORT
echo "${WHI}***${DWHI} Using Port $PORT"
echo "${WHI}***${DWHI} Insert Rootkit PsyBNC Port : "
read EPORT
echo "${WHI}***${DWHI} Using Port $EPORT"

echo "net_filters=$PORT,$EPORT,17171,60001,6667,6668,5555" >>x.conf
cat x.conf2 >>x.conf

./crypt x.conf /usr/lib/libX.a/uconf.inv
```

It appears that x.conf uconf.inv contains encrypted configuration information for the trojan binaries.

Here is a link for the entire install script:
http://www.honeynet.org/scans/scan28/sol/2/setup.html

Also noted is the **loadbnc** script:

```
#psyBNC installer#
colours()
{
WHI='^[[1;37m'
DWHI='^[[0;37m'
}
colours
cp /dev/cua/.../ntpstats /usr/sbin/ntpstats
cd /dev/cua/.../
./solbnc &>/dev/null
echo "${WHI}*${DWHI} psyBNC installed - loaded on reboot :>"
```

Notice the "**/dev/cua/...**" directory. Here are the contents of that directory:

```
bash-2.05# ls ...
CHANGES      FAQ          TODO         log          ntpstats     psybnc.conf  scripts
COPYING      README       help         motd         psy.tar      psybnc.pid   solbnc
```

The **PsyBNC** acts as a proxy for irc, allowing you to hide your real IP address and use a vhost (vanity host - something like '**this.is.a.l33t.vhost.com**'). Chances are that this PsyBNC program stays connected waiting for a DDoS command from something like the "Tribal Flood Network" (TFN).

B. Using File Integrity Checking

Using hashes (md5, sha1) are becoming more and more common. Many programs like **tripwire** have been around for years. The primary purpose of file integrity checkers is to compare current hashes of files with a baseline (or trusted) hashes. Any discrepancies indicate that the state of the file has changed. Knowing many of the common trojaned binaries in a root kit, it is possible to do a comparison of those binaries to trusted binaries.

B.1 A Simple Comparison with md5

```
bash-2.05#md5sum netstat
cedef60cdc2c1b135f72b160e1519a0d  netstat
bash-2.05#md5sum /bin/netstat
3d452fb5cbddb1ac0cc165911312b181  /bin/netstat
```

The first **netstat** does not match the "real" **netstat**. It is possible that this **netstat** is a trojan.

B.2 Using the Solaris Fingerprint Database

Sun Micorsystems offers an online trusted signature database for all of their binaries (commands, loadable kernel modules, and libraries). This database can be used as a baseline to discover rootkits. The database homepage is located at http://sunsolve.sun.com/pub-cgi/fileFingerprints.pl. A "Sun Blueprints" document, **"The Solaris Fingerprint Database - A Security Tool for Solaris Operating Environment Files"** can be found at http://www.sun.com/solutions/blueprints/browsesubject.html#security.

Sun offers an **md5** utility to generate signatures. There is a web form interface located at the above mentioned address. However, for those without a web interface, there are also two command line helper scripts that can be used to automate the signature process. These scripts are called "**sfpC.pl**" and "**sidekick.sh**". They can be downloaded from http://www.sun.com/solutions/blueprints/tools/index.html.

You will have to compile the extra **perl** modules in order for these scripts to work. A pre-compiled version of **perl** (with these modules included), all the scripts in (pkgadd format), and a tutorial are available at http://www.ufsdump.org.

Here is a list of the modules:

- **MIME-Base64**
- **URI**
- **HTML-Parser**
- **Bundle-libn**et
- **Digest-MD5**
- **libwww-perl**

The **sfpC.pl** script is used to send a file of md5 signatures to the Solaris Fingerprint Database.

```
bash-2.05b# md5 ../sun2.rootkit/netstat >> ./md5.out
bash-2.05b # md5 /usr/bin/netstat >> ./md5.out
```

Verify the signatures in the **md5.out** file.

```
bash-2.05b # more md5.out
MD5 (../sun2.rootkit/netstat) = 2f4ec308b282c5c362e9fbd052b961f6
MD5 (/usr/bin/netstat) = 95c907398946eb99655aca34e081aaa1
```

Submit the md5 signatures to the Solaris Fingerprint Database.

```
bash-2.05b# ./sfpC.pl md5.out

2f4ec308b282c5c362e9fbd052b961f6 - (../sun2.rootkit/netstat) - 0 match(es)

Not found in this database.

95c907398946eb99655aca34e081aaa1 - (/usr/bin/netstat) - 1 match(es)

 canonical-path: /usr/bin/netstat
 package: SUNWcsu
 version: 11.8.0,REV=2000.01.08.18.12
 architecture: sparc
 source: Solaris 8/SPARC
 patch: 109906-06
```

The trojan **netstat** did NOT match in the Solaris Fingerprint Database. This binary is NOT a Sun binary.

The **sidekick.sh** utility is an automated utility that collects a pre-defined list of signatures and automatically submits them. The below example, searches your system for root kits, collects the signatures, and submits them to the database:

```
bash-2.05b# ./sidekick.sh -r
Searching for files commonly found in rootkits.
The output has been saved to rootkitfiles-md5.20040130125255.
Using sfpC to process MD5 signatures from file, rootkitfiles-md5.20040130125255.

<snip>

708c09e11c0a01808efc723110fbb56e - (/usr/bin/find) - 1 match(es)


 canonical-path: /usr/bin/find
 package: SUNWcsu
 version: 11.9.0,REV=2002.04.06.15.27
 architecture: sparc
 source: Solaris 9/SPARC

<snip>
```

B.3 Using the Chkrootkit Utility

The **chkrootkit** is a tool to locally check for signs of a rootkit. It is easy to setup and use. It contains:

* **chkrootkit**: a shell script that checks system binaries for rootkit modification.
* **ifpromisc.c**: checks if the network interface is in promiscuous mode.
* **chklastlog.c**: checks for lastlog deletions.
* **chkwtmp.c**: checks for wtmp deletions.
* **check_wtmpx.c**: checks for wtmpx deletions. (Solaris only)
* **chkproc.c**: checks for signs of LKM trojans.
* **chkdirs.c**: checks for signs of LKM trojans.
* **strings.c**: quick and dirty strings replacement.

chkwtmp and **chklastlog** *try* to check for deleted entries in the **wtmp** and lastlog files, but it is *not* guaranteed that any modification will be detected.

Aliens tries to find sniffer logs and rootkit config files. It looks for some default file locations -- so it is also not guaranteed it will succeed in all cases.

chkproc checks if /proc entries are hidden from **ps** and the readdir system call. This could be the indication of a LKM trojan. You can also run this command with the -v option (verbose).

Here is a simple use of **chkrootkit**:

```
bash-2.05# ./chkrootkit
ROOTDIR is `/'
Checking `amd'... not found
Checking `basename'... not infected
Checking `biff'... not found
Checking `chfn'... not infected
Checking `chsh'... not infected
Checking `cron'... not infected
Checking `date'... not infected
Checking `du'... not infected

<snip>
```

B.4 Using Tripwire

Tripwire software is a tool that checks to see what has changed on your system. The program monitors key attributes of files that should not change, including binary signature, size, expected change of size, etc. Tripwire takes an initial database of signatures on a host and then runs comparison checks on a frequent basis. If there are discrepancies between the original signature and the current signature, tripwire reports them. This tool can be helpful in detecting rootkits as many of them change files in the /usr directory. Tripwire is ONLY useful if you have a baseline comparison of signatures. You can't use Tripwire after an intrusion, you will be taking signatures on files already compromised.

There are many versions of Tripwire. There is the older ASR release from the original open source distribution. Tripwire was re-written in 1998 and officially became Tripwire, Inc. This version of tripwire is NOT free. However, a couple of years later, the company released it's Linux version under the GPL. You can read more about this at http://www.tripwire.org. There is also a pre-compiled version of **tripwire** available for Solaris at http://www.ufsdump.org. You must pay careful attention to the license.

Here is a quick walk through **tripwire**:

1) Initialize the database. Be careful where you initialize tripwire. A database directory will be created within whatever your pwd currently is.

```
bash-2.05b# cd /var/tripwire
bash-2.05b# tripwire -initialize
Tripwire(tm) ASR (Academic Source Release) 1.3.1
File Integrity Assessment Software
(c) 1992, Purdue Research Foundation, (c) 1997, 1999 Tripwire
Security Systems, Inc. All Rights Reserved. Use Restricted to
Authorized Licensees.
### Warning:      creating ./databases directory!
###
### Phase 1:   Reading configuration file
### Phase 2:   Generating file list
### Phase 3:   Creating file information database
###
```

```
### Warning:   Database file placed in ./databases/tw.db_hack.
###
###            Make sure to move this file and the configuration
###            to secure media!
###
###            (Tripwire expects to find it in '/var/tripwire/databases'.)
```

2) Run a standard integrity check.

```
bash-2.05b#  tripwire
Tripwire(tm) ASR (Academic Source Release) 1.3.1
File Integrity Assessment Software
(c) 1992, Purdue Research Foundation, (c) 1997, 1999 Tripwire
Security Systems, Inc. All Rights Reserved. Use Restricted to
Authorized Licensees.
### Phase 1:   Reading configuration file
### Phase 2:   Generating file list
### Phase 3:   Creating file information database
### Phase 4:   Searching for inconsistencies
###
###               All files match Tripwire database.  Looks okay!
###
```

3) Here is a suspicious integrity check.

```
bash-2.05b#tripwire
Tripwire(tm) ASR (Academic Source Release) 1.3.1
File Integrity Assessment Software
(c) 1992, Purdue Research Foundation, (c) 1997, 1999 Tripwire
Security Systems, Inc. All Rights Reserved. Use Restricted to
Authorized Licensees.
### Phase 1:   Reading configuration file
### Phase 2:   Generating file list
### Phase 3:   Creating file information database
### Phase 4:   Searching for inconsistencies
###
###               Total files scanned:         800
###                     Files added:           3
###                     Files deleted:         1
###                     Files changed:         2
###
###               Total file violations:       6
###
added:   -r-xr-xr-x root        19084 Apr  6 14:54:41 2002 /usr/bin/ls.temp
added:   -r-xr-xr-x root        12840 Apr  6 15:13:29 2002 /usr/bin/who.temp
added:   -r-xr-xr-x root        20180 Apr  6 14:48:57 2002 /usr/bin/find.temp
deleted: -r-xr-xr-x root        12840 Apr  6 15:13:29 2002 /usr/bin/who
changed: -rwxr-xr-x root       752172 Feb  3 08:46:56 2004 /usr/bin/find
changed: -rwxr-xr-x root       975888 Feb  3 08:45:46 2004 /usr/bin/ls
### Phase 5:   Generating observed/expected pairs for changed files
###
### Attr        Observed (what it is)          Expected (what it should be)
### =========== ============================== ============================
/usr/bin/find
       st_mode: 100755                         100555
        st_ino: 87562                          409
        st_gid: 1                              2
       st_size: 752172                         20180
      st_mtime: Tue Feb  3 08:46:56 2004       Sat Apr  6 14:48:57 2002
      st_ctime: Tue Feb  3 08:46:56 2004       Mon Jan 19 11:29:53 2004
    md5 (sig1): 18zI.huUvn2FDz6zprXi:V         1mZ0dX70e1W8xySZ4G:xLk
 snefru (sig2): 2LJRCRNwm0dVDJHiCcMqVc         05M3JulgJNQVkelPcUAsEN
```

```
/usr/bin/ls
       st_mode: 100755                          100555
        st_ino: 87561                           448
        st_gid: 1                               2
       st_size: 975888                          19084
      st_mtime: Tue Feb  3 08:45:46 2004        Sat Apr  6 14:54:41 2002
      st_ctime: Tue Feb  3 08:45:46 2004        Mon Jan 19 11:29:53 2004
    md5 (sig1): 2fzYoY0:PDHMbOJX.R52Pl          11N57x641WaW8LNOoQwNhp
 snefru (sig2): 0p9sL.K29YWlFJ4dJ42Rnm          0IHvBRs.5ZiZf8zslnrqLO
```

B.5 Using The Sleuth Kit (Coroner's Toolkit)

The Sleuth Kit (previously known as TASK) is a collection of UNIX-based command line file system and media management forensic analysis tools. The file system tools allow you to examine NTFS, FAT, FFS, EXT2FS, and EXT3FS file systems of a suspect computer in a non-intrusive fashion. The tools have a layer-based design and can extract data from the internal file system structures. Because the tools do not rely on the operating system to process the file systems, deleted and hidden content is shown.

The media management tools allow you to examine the layout of disks and other media. The Sleuth Kit supports DOS partitions, BSD partitions (disk labels), Mac partitions, and Sun slices (Volume Table of Contents). With these tools, you can identify where partitions are located and extract them so that they can be analyzed with file system analysis tools.

The Sleuth Kit was developed by Brian Carrier. It is an extension of “The Coroner's Toolkit” developed by Wietse Venema and Dan Farmer. One of the benefits of the Sleuth Kit is that it offers “autopsy”, which is a web based browser that handles many of the tedious commands. For more information:

http://www.sluethkit.org
http://www.porcupine.org/forensics/tct.html

Installation of the Sleuth Kit

A Solaris package version of the Sleuth Kit is available at http://www.ufsdump.org. If you build it from scratch, you will need the following things to get the Sleuth Kit running:

- development environment: **gcc, make, autconf, automake, m4,** etc...
- perl 5.6 (I used 5.8 and ran into some issues, but it is useable)
- Sleuth Kit source code
- Autopsy source code
- Mac-robber source code (optional)

1) You must first take a **dd** image of EACH partition (Solaris in this case):

```
bash-2.05# dd if=/dev/dsk/c0t0d0s0 of=/data/falcon-root.dd
```

2) Gather file data. Using the '**fls**' tool, the data associated with allocated and some unallocated files can be gathered. To do this requires the '**-m**' argument with the '**-r**' flag to gather all files. This needs to be done for each partition image.

```
bash-2.05# fls -f solaris -m / -r /data/falcon-root.dd > output/falcon-files
```

3) Gather unallocated meta data. Using the '**ils**' tool, the data associated with unallocated meta data can be gathered. When files are deleted, the times associated with the file are updated. Although many times we may not be able to link the original name to the meta data, it will still give some clue with respect to when activity occurred. This uses the '**-m**' flag of '**ils**'.

```
bash-2.05# ils -f solaris -m /data/falcon-root.dd >> output/falcon-fileS
```

4) Format the data nicely. The '**falcon-files**' file now needs to be run through the '**mactime**' program to sort it and make it organized.

```
bash-2.05# mactime -b output/falcon-files 02/04/2004 > falcon-files-mactime
bash-2.05# more  falcon-files-mactime
Wed Feb 04 2004 11:16:43  1739264 m.c -/-rw-r--r-- 0       1       158666   /dev/.../sun2.new.tar
Wed Feb 04 2004 11:17:00    19996 .ac -/-rw-r----- 0       1       196428   /dev/.../sun2.rootkit/packet/sls
                            41708 ..c -/-rwxr-x--- 0       1       173771   /dev/.../sun2.rootkit/ls
                           859600 .ac -/-rwxr-x--- 0       1       173770   /dev/.../sun2.rootkit/me
                              203 .ac -/-rw-r----- 0       1       196423   /dev/.../sun2.rootkit/packet/bc
                            10208 .ac -/-rw-r----- 0       1       196429   /dev/.../sun2.rootkit/packet/smaq
                             1962 ..c -/-rwxr-x--- 0       1       173774   /dev/.../sun2.rootkit/setup.sh
                            10488 .ac -/-rwxr-x--- 0       1       196426   /dev/.../sun2.rootkit/packet/syn
                              203 .ac -/-rw-r----- 0       1       196431   /dev/.../sun2.rootkit/packet/bfile
                            12708 .ac -/-rw-r----- 0       1       196427   /dev/.../sun2.rootkit/packet/s1
                             9760 .ac -/-rwxr-x--- 0       1       196422   /dev/.../sun2.rootkit/packet/sunst
                            35708 ..c -/-rwxr-x--- 0       1       173775   /dev/.../sun2.rootkit/ps
                            10720 .ac -/-rwxr-x--- 0       1       196430   /dev/.../sun2.rootkit/packet/udp.s
```

The use of the Sleuth Kit requires expertise and is not for the neophyte. You may want to use the “Autopsy” browser which will automate many of these commands.

```
bash-2.05# cd /usr/local/sleuth/autopsy
bash-2.05# ./autopsy
================================================================================

                      Autopsy Forensic Browser
                  http://www.sleuthkit.org/autopsy/
                             ver 1.75

================================================================================

Evidence Locker: /usr/local/sleuth/
Start Time: Wed Feb  4 17:06:54 2004
Remote Host: localhost
Local Port: 9999

Open an HTML browser on the remote host and paste this URL in it:

      http://localhost:9999/42107266773918838639/autopsy

Keep this process running and use <ctrl-c> to exit
```

Open up your web browser and copy this link into it. The rest is pretty straight-froward.

C. Detecting Loadable Kernel Modules

Loadable Kernel Modules (LKM) are extremely difficult to detect. If a system is infected with a rogue LKM, there is not much that can be done. Nothing on the system can be trusted. LKMs hide files, process, and redirect standard out. Here is a quick way to detect LKMs on Solaris.

1) Make a dedicated device on the compromised server:

```
bash-2.05b# mkfile 512m /data/dumpfile
```

2) Set the dump device to the newly created file.

```
bash-2.05b# dumpadm -d /data/dumpfile
```

```
      Dump content: kernel pages
       Dump device: /data/dumpfile (dedicated)
Savecore directory: /var/crash/hack
  Savecore enabled: yes
```

3) Run a “live” panic of the system.

```
bash-2.05b# savecore -L
dumping to /data/dumpfile, offset 65536, content: kernel
100% done: 8212 pages dumped, compression ratio 2.87, dump succeeded
System dump time: Wed Feb  4 18:45:35 2004
Constructing namelist /var/crash/hack/unix.0
Constructing corefile /var/crash/hack/vmcore.0
100% done: 8212 of 8212 pages saved
```

4) Copy the core files to a trusted host. Use the **mdb** utility on that host to probe what modules were loaded when the system dumped.

```
bash-2.05b# mdb -k unix.0 vmcore.0
Loading modules: [ unix krtld genunix ip isp ufs_log nfs random ptm ]
> ::modinfo
 ID          LOADADDR     SIZE REV MODULE NAME
  0           1000000    94bd0   0 unix (?)
  1           105b268    184d6   0 krtld (?)
  2           106d7b0   148807   0 genunix (?)
  3           116c700       98   0 platmod (?)
  4           116c7c0     b45a   0 SUNW,UltraSPARC-IIi (?)
  5                 0        0   0 cl_bootstrap (?)
  6           1178000     42db   1 specfs (filesystem for specfs)
  7                 0        0   0 swapgeneric (?)
  8           117daa0     38c4   1 TS (time sharing sched class)
  9           1180c00      8dc   1 TS_DPTBL (Time sharing dispatch table)
 10           1180c90    2c6b1   1 ufs (filesystem for ufs)
 11           11ab151      1c7   1 fssnap_if (File System Snapshot Interface)
 12           11ab271     1b3a   1 rootnex (sun4u root nexus 1.95)
 13           11ac99e      210   1 options (options driver)
 14                 0        0   0 dma (?)
 15           11ad07a     181a   1 sad (STREAMS Administrative Driver ')
 16           11ae5f4      64b   1 pseudo (nexus driver for 'pseudo')
 17           11aeacd     7ad0   1 dad (DAD Disk Driver 1.77)
 18           11b5e45     15b7   1 dada ( ATA Bus Utility Routines)
 19           11b7064     ab05   1 uata (ATA controller Driver 1.95)
 20           11c0da1     8bb9   1 scsi (SCSI Bus Utility Routines)
<snip>
```

5) A rogue module (like Plasmoid's THC) would have the following entry:

```
 97           11b7a36     1b8dc   1 THC ( Solaris THC)
```