The Apollo Incident: Evidence.

Thomas Roessler <roessler@does-not-exist.org>

ABSTRACT

apollo.honeyp.edu was attacked on Nov 7 2000, 23:11:51, exploiting a format bug present in **rpc.statd**. The exploit used is known as **statdx**. The attacker, operating from the IP addresses 216.216.74.2 and 24.12.200.186, then installed a collection of trojaned programs from various root kits. A packet sniffer was started, and a version of **sshd** containing a back-door and a password logging facility was installed. Also, the **BitchX** IRC client was run, and an attempt was made to install the **eggdrop** IRC bot. The intruder installed various tools suitable for attacking other systems. At the same time, he tried to make sure that apollo was itself not vulnerable to these attacks.

1. Tools used for the investigation

The most important tools used in the investigation of this incident were:

- The Unix **strings**(1) command, which was applied in order to extract human-readable information from all kinds of binary files, including the hard disk images provided.
- Various commands from *The Coroner's Toolkit* by Wietse Venema and Dan Farmer. In particular, **grave-robber**(1), **ils**(1), **icat**(1), and **mactime**(1) were very helpful.
- The **rpm** package manager. This program helped to identify files which had been altered by the intruder.
- Standard Unix board means, including **awk**(1), **grep**(1), **sed**(1), and **sort**(1).
- **objdump**(1), and **gdb**(1) were used to disassemble one binary.

The system used to perform the actual analysis was a standard Debian GNU/Linux 2.2 system.

2. Given information

The following pieces of evidence and informations on the system were known beforehand:

- apollo was a standard RedHat 6.2 installation.
- An intrusion detection system installed on the same network had logged the information published at [http://project.honeynet.org].

As a basis for the investigation, dd'ed disk images from the system had been made available.

3. Modified system files: The root kit (and other backdoors).

As a very first step, the disk images were mounted as mnt/, and consistency of the system was verified using **rpm**(8), specifically looking for md5 checksum mismatches:

```
# rpm -V -a --root=`pwd`/mnt/ | grep ^..5
```

The output [see file `md5sum.mismatches` for full results] in particular contains the following lines:

```
S.5.....   /bin/ls
S.5.....   /bin/netstat
S.5.....   /sbin/ifconfig
SM5.....   /usr/sbin/in.identd
S.5.....   /bin/ps
```

```
S.5.....   /usr/bin/top
S.5.....   /usr/sbin/tcpd
```

This implies that each of these programs has a size and md5 checksum different from the ones present in the package manager's database, but that the time stamps, — with the exception of in.identd — file modes, and ownerships are identical to the ones listed in the package manager's database.

This observation leads to the suspicion that the attacker has installed a root kit on the system, and modified the files' time stamps. It is expected that these programs will hide network connections, processes, and files.

We investigate each of the binaries with the **strings**(1) command (using the `-a` command line switch), looking for unusual strings. We find the following remarkable facts:

/bin/ls

This file is — quite unusual for a glibc-based Linux distribution — a binary linked against libc5, using **ld-linux.so.1**. Further inspecting the output of **strings**(1), we find a reference to a file named **/usr/man/r**, which is certainly untypical for **ls**. The file does, however, exist on apollo, and contains a list of files most likely hidden by this version of **ls**. Additionally, we find the following strings in the **ls** binary (which is not stripped):

```
/home/aarons2/. /.d/lrk4/fileutils-3.13/src/
/home/aarons2/. /.d/lrk4/fileutils-3.13/intl/
/home/aarons2/. /.d/lrk4/fileutils-3.13/lib/
```

From this, we conclude that the trojaned files most likely come from `lrk4`— the Linux Root Kit version 4. They also give some hints on where this version of the root kit was built.

/bin/netstat

This file is once again a binary linked against libc5. We find a reference to a file named **/usr/libexec/awk/addy.awk**, which is present on apollo, and consists of pairs of one-digit numbers and IP addresses. This file is contained in the list of files hidden by the trojaned **/bin/ls**. The published source code of lrk4 yields the following interpretation:

- Any connections from or to IP addresses which contain — when written in dotted-quad notation — the strings `65.1` or `216.149` are not displayed in **netstat**'s output.
- The two lines

  ```
  1 134518464.134518444
  2 134518464.134518444
  ```

  do not, however, make any immediate sense from the root kit's source code.

/sbin/ifconfig

This is another binary linked against libc5. There are no evidently strange strings in this file. However, we do *not* see the string `PROMISC` in the **strings**(1) output produced. This string is normally present in **/sbin/ifconfig**. We conclude that this version of **ifconfig** is unable to report that an interface is in promiscuous mode — probably to cover up a packet sniffer installed on the target system.

/usr/sbin/in.identd

This program was linked against glibc, and thus most likely does not come from the same source as the root kit programs. Interesting strings we wouldn't expect here include these:

```
-i
sh
/bin/sh
/tmp/.fileMeYV0p
open exec file
execute program
envoye_don_le_trojan
```

A web search for some of these specific strings is not successful. In order to find out the function of the program, we take a look at the disassembly output generated by the command:

```
# objdump -CdRS
```

[See file `in.identd.S` for the raw output.]

We note in particular the following facts:

- **in.identd** first invokes the function `envoye_don_le_trojan`. This function checks the origin of an incoming connection using **getpeername**(2). If the connection's source port is 49239 or 49240, an interactive shell is started.
- Otherwise, **in.identd** copies a large part of its data segment to a file named **/tmp/.fileMeYV0p**, chmod's this file to mode 0700, and then uses **execve**(2) to run that program. [A copy is included as `dot-fileMeYV0p`. It was generated by actually running the fake **in.identd**.]

See file `in.identd.s` for an annotated version of the relevant code sections' disassembler listings. It's worth noting that the coding style used is at times bizarre: For instance, the actual binary is written using calls of the type:

```
fprintf (fp, "%c", mem[i])
```

This is not very efficient.

We now turn to **.fileMeYV0p**. Running **strings**(1) on it reveals, in particular, the following character strings:

```
/.noident
/.fakeid
parse.c: trying to do masqueraded ident: %lX %X %X %lX %X
parse.c: succesful determined masqueraded ident
/usr/local/etc/identd.masq
/proc/net/ip_masquerade
2.8.5fm
```

These strings are consistent with, for instance, the description of a modified version of **pidentd** which is at least distributed with Mandrake Linux, see http://www.sjc.com/LinuxSoft/rpms/-RPMS/mandrake7.0/rpms/pidentd-2.8.5-7mdk.i586.html for details.

The overall structure observed can best be described as a generic backdoor wrapper for servers which are expected to be executed from **inetd**(8).

We finally note that the trojan features of this program were not active, since the ident daemon used is **/usr/bin/identd**, and since this service is not run from **inetd**(8).

/bin/ps

/usr/bin/top

These are libc5 binaries, too. The most remarkable fact is a reference to **/dev/ptyp**, which exists in both programs. This file exists on apollo, and consists of pairs of one-digit numbers and what looks like fragments of program names. According to the published source code of lrk4, the lines prefixed with the digit 2 contain strings which are matched exactly against a process name (and, if matched, hidden), while lines prefixed with the digit 3 contain fragments which are matched against process names.

/usr/bin/tcpd

This is a binary linked against glibc, and thus may come from a different source . Looking at **strings**(1) output, we find the following interesting string:

```
/usr/man/.a
```

Looking at the file **/usr/man/.a** on apollo, we once again find pairs of one-digit integers and strings. These strings include what looks like fragments of IP addresses and ports. The one-digit integers range from 1 to 4. Looking at the public source of lrk4, we notice that this version of the root kit only supports a "magic" value of 1 here.

We note, however, that the file's format would make sense for lrk5's modified version of **netstat**. We also note that, with a "default build" of either version 4 or 5 of lrk, **tcpd** and **netstat** use the same configuration file.

Apparently, the root kit binaries and configuration files actually used by the intruder come from different sources, and builds of root kit versions.

We now turn to other files which were changed on the system, or added to system directories.

/usr/sbin/named

/usr/local/sbin/named

/usr/local/sbin/named-bootconf

/usr/local/sbin/ndc

/usr/local/sbin/irpd

We note that **/usr/sbin/named** and **/usr/local/sbin/named** are identical binaries, and that the system's default **named** binary has been changed. Applying **strings**(1) to this binary, we find the following interesting strings:

```
@(#)named 8.2.2-P5 Thu Nov 25 16:18:38 CST 1999       \
     root@zagnut.goobe.net:/dev/.oz/src/bin/named
$Id: version.c,v 8.3 1999/01/02 06:05:14 vixie Exp $
named 8.2.2-P5 Thu Nov 25 16:18:38 CST 1999
root@zagnut.goobe.net:/dev/.oz/src/bin/named
8.2.2-P5
```

(Lines broken for readability.)

This does, in particular, tell us that this version of **named** was built on the host `zagunt.goobe.net` in directory **/dev/.oz**. It is likely that `zagunt.goobe.net` was a system attacked earlier, and that the **/dev/.oz** directory was the intruder's working directory. The build was performed on Nov 25 1999. 8.2.2-P5 is the version of **named** which was released by the ISC on Nov 12 1999 in response to the security holes covered in CERT advisory CA-99.14. It seems likely that this version of **bind** was installed in an attempt to secure the system against other intruders trying to exploit these errors.

We finally note that there are more files in **/usr/local/** which belong to a newly installed bind distribution.

/etc/rc.d/rc.local

/usr/local/sbin/sshd1

The attacker has installed an ssh1 server. We will come back to this server in the dumpster diving section of this report. It contains a back-door and password logging code.

/usr/sbin/in.telnetd

We'll later find out that this md5 mismatch is actually caused by a conflict between a package installed by the attacker and a package which was part of the default installation.

/usr/tmp/nap

This file contains what looks like the output of a password-logging backdoor. We'll later find out that it has been generated by the ssh1 server on the system.

We now use **find** to search for recently-modified directories. The output can be found in `dirlist`. At this point, we just mention the **/usr/man/.Ci** directory, and the file **/bin/bx**. **/usr/man/.Ci** was the intruder's tool chest. We'll cover it in detail in the timeline section of this report. **/bin/bx** is some version of the BitchX IRC client which gives us a hint about the intruder's motives.

Finally, we mention that four files are stored in the directory **/usr/man**, namely **.a**, **.p**, **p**, and **r**. We have already seen that **r** is the configuration file for the root kit's **ls** binary, and that **.a** is the configuration file for the trojan horse version of **tcpd**(8). We'll later see that **.p** is used by an unused root kit version of **pstree** which is available in the attacker's tool chest. **p**'s contents are identical to **.p**'s, and we'll later find a reference for a file with the same name, but in a different directory in a root kit version of **killall**.

Finally, we unmount the **/usr**, **/var**, **/boot**, and **/home** partitions from apollo, and look for any files hidden in the mount point directories of the root partition. We find nothing.

4. Dumpster diving 1: Looking for log files.

In this section, we look mainly for **syslog**-generated, and **.bash_history** files.

4.1. Shell histories.

find(1) reveals the following files and symbolic links:

```
./.bash_history -> /dev/null
./home/drosen/.bash_history
./root/.bash_history -> /dev/null
./tmp/.bash_history -> /dev/null
./usr/games/.bash_history -> /dev/null
```

[See `bash-histories` for full output including time stamps.]

We note that all symbolic links have ctimes and mtimes dating on Nov 8 08:52 CST, while **/home/drosen/.bash_history**'s mtime and ctime are both 08:59 CST. The contents of this file heavily look like someone installed (and possibly compiled) software using the `drosen` account. We'll come back to this file in the dumpster diving 3 and timeline sections of this report.

4.2. System log files and snippets.

Inspecting **/var/log/messages**, we note that this file doesn't show any signs of the RPC-based attack mentioned in the intrusion detection system's log. We can therefore assume that this file was edited. To find traces of unedited versions of this file, we apply the **strings**(1) and **grep**(1) commands to `honeypot.hda7.dd`. More precisely, we use the following command in order to find log file snippets:

```
# strings < /attic/forensics/honeypot.hda7.dd | \
     grep 'Nov  ' | sort -u
```

The full results can be found in the file `logsnippets`. In particular, we notice the following entries:

```
Nov  8 00:08:40 apollo in.telnetd[2077]: connect from 216.216.74.2
Nov  8 00:08:40 apollo in.telnetd[2078]: connect from 216.216.74.2
Nov  8 00:08:41 apollo inetd[408]: pid 2077: exit status 1
Nov  8 00:08:41 apollo inetd[408]: pid 2078: exit status 1
Nov  8 00:09:00 apollo rpc.statd[270]: SM_MON request for      \
     hostname containing '/': ^D
```

The pattern we can observe here is familiar from the IDS output available: First, two connections to **in.telnetd** are attempted from 216.216.74.2 (ATHM-216-216-xxx-2.home.net). 20 seconds later, an RPC request is made. Although the time stamps differ by almost an hour, it is very likely that the log file entires and the IDS output actually describe the same sequence of events.

We therefore conclude that the system's clock was actually 57:09 minutes ahead of NTP time. Additionally, from the combination of IDS entries and log file entries, we conclude that the **rpc.statd** format problem initially described by *Daniel Jacobowitz <drow@FALSE.ORG>* on the **bugtraq** mailing list on Jul 16 2000 was exploited. From the nature of the format string used, we further conclude that the exploit actually used was **statdx**, posted to **bugtraq** by *ron1n — <shellcode@HOTMAIL.COM>* on Aug 05 2000.

Finally, we apply a simliar method to the swap image:

```
# strings < /attic/forensics/honeypot.hda9.dd | \
```

```
grep -A3 'Nov '
```

The full output is available as `logsnippets.swap`. The following snippets are of particular interest (wrapped for convenience):

```
<30>Nov  8 20:54:25 named[2965]: XSTATS 973738465 973695265 \
     RR=3 RNXD=0 RFwdR=1 RDupR=0 RFail=0 RFErr=0 RErr=0  \
     RAXFR=0 RLame=1 ROpts=0 SSysQ=1 SAns=37 SFwdQ=2     \
     SDupQ=8 SErr=0 RQ=39 RIQ=0 RFwdQ=0 RDupQ=0 RTCP=0   \
     SFwdR=1 SFail=0 SFErr=0 SNaAns=36 SNXD=0
--
<85>Nov  8 20:37:37 login: ROOT LOGIN ON tty1
sion o
d for user root by LOGIN(uid=0)
 of type %s
--
<30>Nov  8 08:54:25 named[2964]: Forwarding source address is       \
     [0.0.0.0].1037
<30>Nov  8 08:54:25 named[2964]: listening on [172.16.1.107].53 (eth0)
<30>Nov  8 08:54:25 named[2964]: master zone "0.0.127.in-addr.arpa"      \
     (IN) loaded (serial 1997022700)
<28>Nov  8 08:54:25 named[2964]: Zone "0.0.127.in-addr.arpa"        \
     (file named.local): No default TTL set using SOA minimum instead
<28>Nov  8 08:59:52 inetd[408]: pid 2387: exit status 1
<86>Nov  8 08:28:41 login: LOGIN ON 0 BY adm1 FROM             \
     c871553-b.jffsn1.mo.home.com
--
<38>Nov  8 0
@pwdb[2404]: (su) session opened for user own by adm1(uid=5000)
```

It is interesting to note that most of these entries were not even present on the **hda7** image. This most likely implies that **syslogd** has been shut down before these events occured. (According to **/etc/syslog.conf**, **syslogd** was running in synchronuous mode.)

5. Dumpster diving 2: Grepping for some random debris

5.1. Looking for environments on the swap.

We now look for program environments in the swap space. We use the following command:

```
# strings /attic/forensics/honeypot.hda9.dd | \
     egrep -A3 '[A-Z]+='
```

Full output is available as `swap-environments`. In particular, we find several environments like this:

```
LESSOPEN=|/usr/bin/lesspipe.sh %s
HISTSIZE=1000
HOSTNAME=apollo.honeyp.edu
LOGNAME=adm1
REMOTEHOST=c871553-b.jffsn1.mo.home.com
MAIL=/var/spool/mail/adm1
TERM=vt100
HOSTTYPE=i386
PATH=/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin
HOME=/root
INPUTRC=/etc/inputrc
SHELL=/bin/bash
```

```
USER=adm1
LANG=en_US
OSTYPE=Linux
_=/usr/sbin/named
```

Elements common to these environments are the `vt100` terminal, the `REMOTEHOST` variable's value, and the `adm1` `USER` or `LOGNAME` settings. The _ environment variable is set by **bash**(1) to contain the last typed command. We do, in particular, find the following strings:

```
_=/bin/su
_=/usr/local/sbin/sshd
_=/usr/sbin/named
```

Each of these strings can be found in an environment which mentions the `adm1` user.

Finally, we digress and find some environment fragments which give some indication of how the file system dumps were generated in the aftermath of the intrusion:

```
LESSOPEN=|/usr/bin/lesspipe.sh %s
USERNAME=root
ENV=/root/.bashrc
HISTSIZE=1000
HOSTNAME=apollo.honeyp.edu
LOGNAME=root
MAIL=/var/spool/mail/root
TERM=linux
HOSTTYPE=i386
HOME=/root
INPUTRC=/etc/inputrc
:*.xpm=
SHELL=/bin/bash
tif=
USER=root
LANG=en_US
SHLVL=1
_=./nc
```

The following snippet is even more revealing, although it's possibly not from a buffer used for an environment (wrapped for convenience):

```
[root@apollo linux]# ./dd bs=1024 < /dev/hda8 |      \
     ./nc 192.168.1.10 10000 -w 3
LANG
en_US
LANG
```

5.2. What accounts have ever existed on the system? — and another discovery.

We have, so far, seen some evidence of accounts named `own` and `adm1` which seem to have existed on the system. Still, none of these accounts occur in the **/etc/passwd** and **/etc/shadow** files on apollo. Since it's typical for entries in these files to contain colon-enclosed numbers, we simply apply **strings**(1) and **grep**(1) to the raw image, looking for such patterns:

```
# strings /attic/forensics/honeypot.hda8.dd | egrep ':[0-9]+:'
```

The full output can be found in file `passwd.suspects`. Investigating that file, we find lots of strings which contain time stamps for source code (which is something we are going to look for in Dumpster Diving 3). Still, we can isolate several passwd-like entries which are stored in `passwd.snippets`. Even better, we find what looks like the remainder of a **.bash_history** file which informs us how the `own` and `adm1` accounts were most likely created:

```
echo own:x:0:0::/root:/bin/bash >> /etc/passwd
echo adm1:x:5000:5000:Tech Admin:/tmp:/bin/bash >> /etc/passwd
echo own::10865:0:99999:7:-1:-1:134538460 >> /etc/shadow
echo adm1:Yi2yCGHo0wOwg:10884:0:99999:7:-1:-1:134538412 >> /etc/shadow
```

We revisit the root partition's dd'ed image with **strings**, and recover what looks like fragments from `root`'s **.bash_history** file, see `bash_history.root`. We reproduce the section of this file which may give us a look at the intruder's early activities:

```
uptime
rm -rf /etc/hosts.deny
touch /etc/hosts.deny
rm -rf /var/log/wtmp
touch /var/log/wtmp
killall -9 klogd
killall -9 syslogd
rm -rf /etc/rc.d/init.d/*log*
echo own:x:0:0::/root:/bin/bash >> /etc/passwd
echo adm1:x:5000:5000:Tech Admin:/tmp:/bin/bash >> /etc/passwd
echo own::10865:0:99999:7:-1:-1:134538460 >> /etc/shadow
echo adm1:Yi2yCGHo0wOwg:10884:0:99999:7:-1:-1:134538412 >> /etc/shadow
cat /etc/inetd.conf | grep tel
exit
```

In particular, this confirms our earlier claim that **syslogd** and **klogd** were killed by the intruder at a very early stage. The final **grep** can be read to indicate that the intruder planned to revisit the system using **telnet**. Finally, we notice that `own` was a password-less `root` account, and that `adm1` was a password-protected user account used to telnet into the system. This is consistent with the **login** and **pwdb** log file snippets we recovered from the system's swap.

6. Dumpster diving 3: Reconstructing and analyzing free inodes.

Finally, we apply the **ils**(1) and **icat**(1) programs from *The Coroner's Toolkit* to extract free, but non-empty inodes from the disk images. In what follows, we will use the notation hda*N*/*inum* for the files recovered that way. We extract the inodes like this:

```
# for f in hda1 hda5 hda6 hda7 hda8 ; do \
  mkdir $f ; \
  ils -r /attic/forensics/honeypot.${f}.dd | \
    awk -F '|' '(NF == 13 && $2 == "f" && $11 > 0) { print $1 }' | \
    while read inode ; do \
      icat /attic/forensics/honeypot.${f}.dd $inode > $f/$inode ; \
   done ; \
done
```

We first note that no free inodes were reconstructed on **hda1** and **hda6**.

6.1. hda8: The Root Partition.

Using **file**(1) to inspect the files we just gathered, we find several shell scripts, lots of C source code, and finally a GNU tar archive in inode no. 8133. Inspecting this archive, we see that it contains a directory whose name is a single space character, and that this directory apparently contains an **eggdrop** IRC bot. Browsing some of the files, we see references to `tPACK`, and a script named `install`. With this archive, we can interpret the **.bash_history** file we found in `drosen`'s home directory: Apparently, the `drosen` account was used to compile and "install" an eggdrop bot.

Taking md5sums of the free inodes and the extracted contents of the **eggdrop** tar ball, we can use standard shell tools to create a list of the free inodes which did not belong to the eggdrop source. We find:

60502

A compiled version of **encrypt.c** from the **eggdrop** distribution.

60531

A time stamp file, most likely from building **eggdrop.** The string contained is:

```
Wed Nov  8 08:58:56 CST 2000
```

Most of the other inodes seem to contain directories or intermediate data from building eggdrop. Some inodes contain arbitrary data garbage (22200, 22193, 22194). The contents of these files may be encrypted data, which may merit further inspection, but is not covered in this report.

6.2. hda7: The /var Partition.

We have only two free, but nonempty inodes on this partition, 22184 and 4040. 22184 contains 75 null bytes. By inspecting **ils**(1) output, we see that 4040 is owned by user `root`, group `slocate`. We further note that 4040 contains what looks like file name information. We conclude that this file has been left over by some update process for the **slocate** package. We'll investigate this further in the timeline section.

6.3. hda5: The /usr Partition.

We finally turn to the **/usr** partition which contains ther attacker's tool chest, and may reveal some interesting details. Once again, we first inspect the extracted inodes with **file**(1). Inodes 109791 and 109861 contain GNU tar archives.

Investigating 109861, we find a pre-built **bind** distribution, and an installation script named **install**, which installs the relevant binaries in **/usr/local**, and finally replaces **/usr/sbin/named** by the newly-installed version. We note that this script nicely explains the **bind**-related files installed on apollo.

109791 contains the complete source archive for ssh-1.2.27, including object files. We compare the contents of this archive to a known-good distribution tar ball fetched from `ftp.fu-berlin.de` using **diff**(1). We find that the source code found on apollo contains provisions for a back-door using a universal password (whose MD5 checksum is stored), and for a password logger. Looking at the **config.status** file contained in the tar ball, we suppose that the attacker has most likely made use of these features. The pre-built version of **ssh** installed on the system was configured with the following parameters:

```
./configure  --enable-global=d33e8f1a6397c6d2efd9a2aae748eb02  \
     --enable-sshd-log=/usr/tmp/nap
```

To verify that these parameters were indeed used for the **sshd1** installed on `apolllo`, we investigate that binary with **strings**(1). The md5 checksum of the global password and the path to the log file are both found.

Further, we note that the tar ball found on apollo actually contains two copies of the **ssh** source code. The only difference between these two versions of the code is in a comment. In `ssh.diff`, we include a diff between the known-good version of the source and the "top-level copy" found on apollo.

Finally, we have another look at **/usr/tmp/nap**, and MD5-hash the password which was stored there:

```
# echo -n "tw1Lightz0ne" | md5sum
d33e8f1a6397c6d2efd9a2aae748eb02
```

We conclude that `tw1Lightz0ne` was indeed the password used for the back-doored **ssh** server.

We further find two deleted Red Hat packages:

```
109865: RPM v3 bin i386 nfs-utils-0.1.9.1-1
109866: RPM v3 bin i386 wu-ftpd-2.6.0-14.6x
```

In 109801, we find a shell script which unpacks, builds, and installs the **ssh** distribution from 109791. It is, at this point, not likely that this script was actually used: It creates a copy of **/usr/local/sbin/sshd1** in **/usr/sbin/sshd**. This file does not exist on apollo, and an inspection of inode hda5/92360 (which contains the **/usr/sbin** directory) with **icat** does not reveal any traces of **sshd**.

In 109802, we find a slightly modified version of this script, which is indeed consistent with what we see in the rest of the file system.

109803 contains another small shell script which contains instructions for unpacking and installing the **bind** distribution from inode 109861.

109864 contains a shell script which uses **rpm** to upgrade the system's **nfs-utils** installation to **nfs-utils-0.1.9.1-1.i386.rpm**. (See inode 109865.) The script echoes the string `statd patch`.

109867 contains a script which echos the string `patching wuftpd`, and installs an (updated?) **wuftpd.rpm** (see inode 109866).

Verifying MD5 hashes, we notice that the recovered packages actually match the security upgrades announced in RHSA (Red Hat Security Advisory) 2000:039-02 and RHSA 2000:043-03, respectively. The newly installed **nfs-utils** package actually fixes the hole which was used by the intruder to originally gain access to the system.

Most of the other files can be readily identified as either source files from the ssh-1.2.27 distribution, as files created when **ssh** was built, or as files from the **bind** binary distribution found on the system. One detail is particularly worth noting: In 109958 (and other files from the **ssh** tar-ball), we find the following string:

```
/dev/.oz/.nap/rkit/terror/.oz/ssh-1.2.27/gmp-2.0.2-ssh-2/mpq/
```

This is an indication on where this version of **ssh** was built, and hints at yet another root kit which may have belonged to the attacker's tool chest. It should be noted that neither lrk4, nor lrk5 contains a directory named `terror`.

7. The attack's time line.

This section contains a possible time line for the attack. It's based on the information gathered so far, and on output from the **mactime**(1) utility from *The Coroner's Toolkit*. In order to use **mactime**(1), **grave-robber**(1) was run against the mounted file system. **grave-robber**'s `body` file and output from manual invocations of **ils** and TCT's **ils2mac** utility were then combind into the file `bigbody`, which is attached to this report. The result was then processed with **mactime**(1), and slightly changed with a simple **sed** script, so information about free inodes could more easily be read. The **mactime**(1) output can be found in its entirety in the file `timeline.txt`.

All time stamps in what follows are — unless explicitly noted otherwise — in the system's own time zone, i.e., 57:09 minutes ahead of actual CST.

7.1. Preliminaries: Explaining hda7/4040

The part of the timeline we consider begins on Nov 07 04:02. At this point of time, hda7/4040 and hda7/26218 are accessed, and their contents modified. The following entries, which are dated Nov 08 04:02, reveal the cause for this behaviour: We are seeing side-effects from the system's daily routine clean-up. This part of the timeline explains the free inodes we observed on hda7.

7.2. The attack.

00:09 (23:11 CST)

From the IDS logs and log file snippets recovered from **hda7**, the initial attack happened on Nov 07 23:11:50 CST, or Nov 08 00:09:00 system time. The intruder, coming from 216.216.74.2, exploited the **rpc.statd** format bug described in section 4.2, and installed a "shell server" on port 4545 using a simple modification to **/etc/inetd.conf**.

08:25 (07:28 CST)

Inspecting `timeline.txt`, we find the next interesting activities at Nov 08 08:25 system time.

```
Nov 08 00 08:25:53     2836 .a. -r-xr-xr-x root     root     mnt/usr/bin/uptime
Nov 08 00 08:26:15        0 m.c -rw-r--r-- root     root     mnt/etc/hosts.deny
Nov 08 00 08:26:51     1024 .a. drwxr-xr-x root     root     mnt/etc/rc.d/init.d
Nov 08 00 08:29:27    63728 .a. -rwxr-xr-x root     root     mnt/usr/bin/ftp
Nov 08 00 08:33:42     1024 .a. drwx------ daemon   daemon   mnt/var/spool/at
Nov 08 00 08:45:18    31376 .a. -rwxr-xr-x root     root     <hda5/93839>
                        161 .a. -rw-r--r-- root     root     mnt/etc/hosts.allow
                          0 .a. -rw-r--r-- root     root     mnt/etc/hosts.deny
Nov 08 00 08:45:19       63 .a. -rw-r--r-- root     root     mnt/etc/issue.net
Nov 08 00 08:45:24     1504 .a. -rw-r--r-- root     root     mnt/etc/security/console.perms
Nov 08 00 08:51:37  2129920 m.. -rw-r--r-- drosen   drosen   <hda8/8133>
Nov 08 00 08:51:53     1153 .a. -rwxr-xr-x 1010     users    <hda5/109801>
```

First, **/usr/bin/uptime** is read (i.e., executed). Contents and inode of **/etc/hosts.deny** are modified, and something is changed in **/etc/rc.d/init.d**. The **.bash_history** fragments recovered in section 5.2 are consistent with these time stamps. We note that the time stamps we can link to the individual events from that **.bash_history** file are at distances of 10 and more seconds. From this, we conclude that the attacker most likely typed in the commands manually, or cutted and pasted them from somewhere. It's unlikely that we are seeing script activities at this point.

For convenience, we reproduce the relevant sections from the recovered **.bash_history**:

```
uptime
rm -rf /etc/hosts.deny
touch /etc/hosts.deny
rm -rf /var/log/wtmp
touch /var/log/wtmp
killall -9 klogd
killall -9 syslogd
rm -rf /etc/rc.d/init.d/*log*
echo own:x:0:0::/root:/bin/bash >> /etc/passwd
echo adm1:x:5000:5000:Tech Admin:/tmp:/bin/bash >> /etc/passwd
echo own::10865:0:99999:7:-1:-1:134538460 >> /etc/shadow
echo adm1:Yi2yCGHo0wOwg:10884:0:99999:7:-1:-1:134538412 >> /etc/shadow
cat /etc/inetd.conf | grep tel
exit
```

We can speculate that this was the first and only time the attacker used his original and uncomfortable back-door.

We also note that the recovered **.bash_history** ends at this point with an `exit` command, which implies that the intruder left the system at this point.

08:28 (07:31 CST)
According to one of the log file snippets found in the system's swap partition, the intruder returned at 08:28:41 using telnet from 24.12.200.186 (c871553-b.jffsn1.mo.home.com), and logging in as user `adm1`.

08:29 (07:32 CST)
He then invoked **/usr/bin/ftp** at 08:29:27 (see the timeline excerpts reproduced above).

08:45 (07:48 CST)
The **mactime**-generated time line then reveals a read access to hda5/93839 at 08:45:18. This inode contains a copy of the telnet daemon, which implies that the intruder's last telnet session before installing the new daemon began at 08:45. Investigating time stamps on the **telnetd** installed later on, we conclude that the attacker indeed did not telnet into the machine after this point of time. This is further supported by the facts that **hosts.allow** and **hosts.deny** files were last read at this point of time. In order to further confirm this, we dump apollo's **/var/log/lastlog** file using a small **lastlog**(8)-like tool which dumps all records from lastlog which have a non-zero

ll_time field (see file `lastlog.c`). We obtain the following output:

```
   0   tty1             Wed Nov  8 20:37:37 2000
5000   1                Wed Nov  8 08:45:24 2000
               (from c871553-b.jffsn1.mo.home.com)
```

This confirms several facts: First, the intruder indeed used user id 5000 (adm1) to log in over the network. Second, he came from c871533-b.jffsn1.mo.home.com. Third, his last login as adm1 was indeed at 08:45 system time. Also note that the time stamp recorded in **/var/log/lastlog** corresponds to the atime of the file **/etc/security/console.perms**, which is inspected by **/lib/security/pam_console.so**, which is in turn used by **login**.

08:51 (07:54 CST)

The next time stamp we find refers to hda8/8133, the eggdrop bot's source we investigated in 6.8. This file's contents was last modified at 08:51:37. We can read this as a sign for a finished download.

The next events we observe are at 08:51:53-08:51:56: Various files in **/usr/man/.Ci** are first accessed, and then their inode data are changed. In order to understand this behaviour, it is instructive to unpack, as root, an arbitrary tar ball, and then investigate time stamps using **mactime**(1). The basic pattern — access first, then a change to the inode data --- is the same. Additionally, the fact that these files belong to user 1010 which left no other traces on the system may indicate that we have the results of an instance of **tar** running as root here, unpacking some archive which contained the entire **.Ci** directory.

08:52 (07:55 CST) — subverting the system

At 08:52:09 system time, the attacker begins to quickly replace **.bash_history** files by symbolic links. His speed at this point makes it likely that a script was used which, apparently, did not survive the activities to follow. In order to find out more details, we engage in dumpster diving another time, and run

```
# unrm /attic/forensics/honeypot.hda5.dd | strings
```

Browsing through the output data and searching for **.bash_history**, we find what looks like fragments of a tar archive. In fact, we can reconstruct the script the attacker has most likely run. It is contained in the file `trojan.sh`. (The entire output of the command run is 31 MB of data; we don't include this for space reasons.)

We note that the script expects to be run in the **.Ci** directory; we can assume that, at some point between 08:51:56 and 08:52:09 system time, the attacker changed directories to **.Ci**.

The script begins by removing various **.bash_history** files and replacing them with symbolic links to **/dev/null**.

The sequence of events which follows at 08:52:10 system time can entirely be explained by looking at this script, too: It creates a directory named **backup**, and tries to copy the system's **ps**, **top**, **syslogd**, **netstat**, **ifconfig**, and **tcpd** binaries to that directory.

The path used for **syslogd** is, however, wrong, so this file is neither copied (and found in the backup directory later on), nor is it replaced by a trojan horse version.

The script then prints the message `Trojaning in progress` and uses **./fix** to replace these programs by the trojan versions included in the root kit tar ball. **./fix** is a standard part of lrk4, which installs trojan horses, preserving file mode, ownership, and time stamps. Additionally, **./fix** tries to fix CRC checksums under certain conditions. Comparing **sum**(1) checksums of the files' backup versions and the installed versions reveals that this did not happen.

At this point, the script tries to restart **syslogd** (and fails in this case, since the intruder has killed that daemon before).

It then calls (and later removes) a script named **addbd**. We can reconstruct this file from the dumpster diving we performed above on **hda5**. It's included as `addbd.sh`, and creates the files **/usr/man/p**, **/usr/man/.p**, **/usr/man/r**, and **/usr/man/.a**. We find this sequence of activities in `timeline.txt` at 08:52:12 system time.

Note that only **/usr/man/r** and **/usr/man/.a** are used by the trojan horses actually installed on the system. The other configuration files are unused, which further supports the impression that the attacker's tool chest came from a variety of sources, and was possibly not fully understood by the individual using it.

08:52:13 (07:55 CST) — Starting the sniffer

Next, the program **./snif** is started by `trojan.sh`. We find corresponding entries in `timeline.txt` at 08:52:13. In particular, **sniff.pid** and **tcp.log** were created. Both file names occur in the **snif** binary. Further inspecting the binary with **strings**(1), and comparing with an independently-built instance of the **linsniffer** included with lrk4, we can safely state that **./snif** is indeed an instance of **linsniffer**.

Note that **tcpd.log** was not modified after this poit of time. This implies that no TCP connections logged by **linsniffer** came in after the sniffer was started.

In order to better understand this fact, we first list the relevant lines in function **filter**() from the published C source code (see file `linsniffer.c`):

```
   if(ntohs(tcp->dest)==21)  p=1; /* ftp */
   if(ntohs(tcp->dest)==23)  p=1; /* telnet */
/*   if(ntohs(tcp->dest)==110) p=1;  pop3 */
/*   if(ntohs(tcp->dest)==109) p=1;  pop2 */
   if(ntohs(tcp->dest)==143) p=1; /* imap2 */
   if(ntohs(tcp->dest)==513) p=1; /* rlogin */
/*   if(ntohs(tcp->dest)==106) p=1;  poppasswd */
   if(victim.active == 0)
      if(p == 1)
         if(tcp->syn == 1)
         {
```

Evidently, **linsniffer** is activated by TCP connection setup on certain ports. It's easy to customize this section of the code. For this reason, we disassemble the **snif** binary found on apollo (see file `snif.s`), and try to extract the list of ports watched. In the **filter**() function, we find the following assembly code:

```
804892a:       86 c4                   xchg   %al,%ah
804892c:       0f b7 d0                movzwl %ax,%edx
804892f:       89 d0                   mov    %edx,%eax
8048931:       66 83 f8 15             cmp    $0x15,%ax
8048935:       75 05                   jne    804893c
                              <filter+0x74>
8048937:       bb 01 00 00 00          mov    $0x1,%ebx
804893c:       89 d0                   mov    %edx,%eax
804893e:       66 83 f8 17             cmp    $0x17,%ax
8048942:       75 05                   jne    8048949
                              <filter+0x81>
8048944:       bb 01 00 00 00          mov    $0x1,%ebx
8048949:       89 d0                   mov    %edx,%eax
804894b:       66 3d 8f 00             cmp    $0x8f,%ax
804894f:       75 05                   jne    8048956
                              <filter+0x8e>
8048951:       bb 01 00 00 00          mov    $0x1,%ebx
8048956:       89 d0                   mov    %edx,%eax
8048958:       66 3d 01 02             cmp    $0x201,%ax
804895c:       75 05                   jne    8048963
                              <filter+0x9b>
804895e:       bb 01 00 00 00          mov    $0x1,%ebx
```

```
8048963:       80 3d 08 a4 04 08 00    cmpb   $0x0,0x804a408
804896a:       75 59                   jne    80489c5
                                    <filter+0xfd>
804896c:       83 fb 01                cmp    $0x1,%ebx
```

Note that

```
xchg %al, %ah
```

is the equivalent of a **ntohs**(3) function call, applied to %ax. The **cmp**, **jne**, **mov** sequence which is then found several times nicely fits the C code quoted. The **cmp** instruction at 804896c can be identified with the statement

```
if (p == 1)
```

in line 137 of the C source code. Thus, we can find the list of ports monitored by examining the **cmp** statements at 8048931, 804893e, 804894b, and 8048958. We confirm that the sniffer installed on apollo matches the publicly known source code in this aspect: The ports monitored are 0x15 (21, ftp), 0x17 (23, telnet), 0x8f (143, imap), and 0x201 (513, rlogin).

This analysis in particular confirms the conclusions drawn from the 08:45 time stamps: No new **telnet** connections to apollo were opened after 08:52:13. Since the replacement telnet daemon wasn't installed before 08:52:33 (see below), we can conclude that the connection observed at 08:45 was indeed the last **telnet** to apollo before the disk images were made.

The starting of the sniffer gives us an important hint to reconstruct the time stamp on the **pwdb** log message we could partially reproduce from the system's swap: From **sniff.pid**, we know that the sniffer got the process id 2485 at 08:52:13, which permits us to conclude that the log snippet describes an event which happened not too long before this point of time.

08:52:14 (07:55 CST) — Cleaning the log files

Back to `trojan.sh`, **./clean** is executed at 08:52:14 system time. This script, which is still present on apollo, creates various temporary files which are then used as input for another script, **./snap**. This script is used to remove certain lines from various log files.

On apollo, any lines matching the strings `echoing ip's and shit`, `sshd`, `log`, `games`, `209.86`, `own`, `owned`, `Pro`, `snif`, `ident`, `splitrock`, `209.255`, `echo`, and `snap'ping` were supposed to be removed from the files **/var/log/messages**, **/var/log/secure**, and **/var/log/xferlog** at this point. It's worth noting that this was not the last manipulation of these files, since we do not see any "m" entries in `timeline.txt` at this point of time.

08:52:15 (07:55 CST) — More root kit configuration; hardening the system.

The next timeline entries nicely fit with `trojan.sh`: At 08:52:15, inode information for **/dev/ptyp** is modified, and **a.sh** is read. This script kills and removes various system services, most of which have a recent history of security holes. In particular, this includes **rpc.statd**, which was used for the original attack.

The next — long — line in `trojan.sh` uncompresses and unpacks a file named **rpms.tgz**, which apparently contained various Red Hat packages. These packages are then installed using:

```
rpm -Uvh --force *.rpm
```

The files **rpms.tgz**, **rpms.tar**, and the **rpms** directory are then removed.

We do not have access to **rpms.tgz** or **rpms.tar**. However, running

```
# rpm --root=`pwd`/mnt -q -i -a
```

reveals a list of package installation dates. We summarize the recent entries from this list, sorting by time stamps:

am-utils	Wed Nov 8 08:52:26 2000
lpr	Wed Nov 8 08:52:32 2000

make	Wed Nov 8 08:52:32 2000
ypserv	Wed Nov 8 08:52:33 2000
screen	Wed Nov 8 08:52:33 2000
telnet	Wed Nov 8 08:52:33 2000
wu-ftpd	Wed Nov 8 08:53:41 2000
nfs-utils	Wed Nov 8 08:53:49 2000

This list is consistent with `timeline.txt`, and reveals that **rpms.tgz** contained the **am-utils**, **lpr**, **make**, **ypserv**, **screen**, and **telnet** packages. We come back later to the **wu-ftpd** and **nfs-utils** packages, for which we have found specific installation scripts in section 6.3.

Comparing package versions on apollo with the versions supplied with a default RedHat 6.2 installation, we see that only the **wu-ftpd** and **nfs-utils** packages installed were actual upgrades. All other packages were downgrades from the default installs of RedHat 6.2. Still, with the exception of the **make** package, the newly-installed packages had been released as security upgrades in the past, so this step would have counted as "securing the system" on an older RedHat installation.

Additionally, the version of the **telnet** package installed by the intruder actually contains **telnet** clients and servers, and conflicts with the **telnet-server** package installed by default on a RedHat 6.2 system. This resolves the md5 checksum mismatch for the **/usr/sbin/in.telnetd** binary observed earlier: The binary installed on the system is consistent with the **telnet** package, but inconsistent with the **telnet-server** package.

We summarize the security updates attempted by the intruder. In the RHSA column, we list the Red Hat security advisory in which the package version used by the intruder was announced; in the release date column, you find that advisory's date.

Package	RH's version	Intruder's version	RHSA	release date
screen	3.9.5-4	3.9.4-3	1999:042-01	1999-10-20
am-utils	6.0.3-1	6.0.1s11-1.6.0	1999:032	1999-08-30
lpr	0.50-4	0.48-1	2000:002-1	2000-01-08
ypserv	1.3.9-3	1.3.9-1	1999:046-01	1999-10-27
telnet	0.16-6	0.10-29	1999:029-01	1999-08-19

Fetching the relevant packages from RedHat's server, we can use **rpm** to finally verify that the intruder did not install trojan horse versions of these packages:

```
# ls *.rpm
am-utils-6.0.1s11-1.6.0.i386.rpm  telnet-0.10-29.i386.rpm
lpr-0.48-1.i386.rpm               ypserv-1.3.9-1.i386.rpm
screen-3.9.4-3.i386.rpm
# for f in *.rpm ; do rpm --root=`pwd`/mnt --verify -p $f ; done
..?..... c /etc/amd.conf
..?..... c /etc/amd.net
......G.   /var/spool/lpd
#
```

We return to **trojan.sh**. The script next restarts **inetd** and **lpd**, and cleans **/var/log/wtmp**. The only trace we see of these activities is the atime (08:52:33) on **/var/log/wtmp**, which was written to at later times.

At 08:52:34, we see write and inode access to **/bin/bx**. This implies that the copying of **bx** to **/bin**, and the chmod we find in `trojan.sh`, were running or finished at this point of time. We note, however, that there is no corresponding read access for **/usr/man/.Ci/bx**, and that this file was read at a later point of time.

The next lines in `trojan.sh` are supposed to replace **/usr/sbin/in.ftpd**. While we do find some traces of an **in.ftpd** when inspecting **/usr/man/.Ci** with **icat**(1), it is not clear whether this file was actually installed at this point of time. In any event, it would have been overwritten shortly thereafter.

At this point, `trojan.sh` just spits out some more messages, and then terminates.

(Note that these messages permit us to assign names to the scripts in hda5/109801 and hda5/109802: 109801 was **install-sshd1**, and 109802 was **install-sshd**.)

08:52-08:53 (07:55-07:56 CST)

The next events we observe can be found in `timeline.txt`, and start at 08:52:53. We observe activities which we suspect to be related to the unpacking of another tar ball in the **/usr** partition. These activities end for a short moment at 08:52:59, when hda5/109791 was last accessed. We have, in section 6.3, identified this file as the tar archive containing the source code of the trojaned version of **ssh** which was installed on apollo. We suppose that this is the tar archive unpacked at this point of time. In order to further confirm our assumption, we look at some of the files which were extracted and recovered later on. We confirm that these files belong to the **ssh** distribution.

We suspect that **install-sshd** (hda5/109802) was running at this point of time, and indeed hda5/109802 was last read at 08:53:13, which is consistent with this assumption.

After untarring the source, that script would issue the command

```
make install
```

in the source directory. Looking at `timeline.txt`, we notice that this is indeed consistent: We apply **file**(1) to a random sample of the files read at 08:53:03-08:53:05, and notice that apparently most files read during these seconds were object files. Additionally, all files written during this interval of time were executable, which further supports the hypothesis that what we're seeing here is actually the process of linking together the trojan ssh version.

The access patterns we observe between 08:53:06 and 08:53:13 are consistent with an actual installation of the trojaned **ssh**, which is what we expect to happen at this point of time.

At 08:53:13, we observe write access to **/etc/sshd_config** and **/etc/rc.d/rc.local** (which is consistent with an execution of **install-sshd**), and read access for **/usr/bin/awk** (which is used by **install-sshd** in order to parse **ps**(1) output).

At 08:53:28, we observe what looks like an invocation of the **whereis** command, most likely in an attempt to find the just-installed instance of ssh. Then, at 08:53:33, netstat is invoked, and **/usr/local/sbin/sshd** started:

```
Nov 08 00 08:53:33    32816 .a. -rwxr-xr-x root     root     mnt/bin/netstat
                     202709 .a. -rw-r--r-- root     root     mnt/boot/System.map-2.2.14-5.0
                        537 .a. -rw------- root     root     mnt/etc/ssh_host_key
                        512 .a. -rw------- root     root     mnt/etc/ssh_random_seed
                        684 .a. -rw-r--r-- root     root     mnt/etc/sshd_config
                      47008 .a. -rwxr-xr-x root     root     mnt/lib/libutil-2.1.3.so
                         16 .a. lrwxrwxrwx root     root     mnt/lib/libutil.so.1 -> libutil-2.1.3.
                          5 .a. lrwxrwxrwx root     root     mnt/usr/local/sbin/sshd -> sshd1
                     643674 .a. -rwxr-xr-x root     root     mnt/usr/local/sbin/sshd1
                          5 mac -rw-r--r-- root     root     mnt/var/run/sshd.pid
```

The next block of activities lasts from 08:53:40 until 08:53:43, and covers the installation of **wu-ftpd.rpm**. Note that this is consistent with the rpm time table generated above, which dated this installation at 08:53:41. Indeed, at 08:53:41, we find the last read access to hda5/109866, which contains **wu-ftpd-2.6.0-14.6x.rpm**. (Named **wu-ftpd.rpm** by the attacker.) The last read access to hda5/109867, the script responsible for installing the new **wu-ftpd** package, is at 08:53:43, which is consistent with our hypthesis.

08:53-08:54 (07:56-07:57 CST)

Next, we observe the installation of the updated **nfs-utils** package. The relevant entries last from 08:53:47, when **rpm** was last read on apollo, until 08:54:05, when hda5/109864 was last read —

the script responsible for the installation of this package. Note that, at 08:54:05, we also see read access to **/etc/rc.d/init.d/nfslock** and related files. These patterns are consistent with the fact that hda5/109864 calls

```
/etc/rc.d/init.d/nfslock restart
```

in the end.

According to the table generated from **rpm**'s output, the **nfs-utils** package was installed at 08:53:49. This is consistent with our observations.

From 08:54:18 until 08:54:43, we can observe the access patterns generated while the new name server package was installed using hda5/109803. The name server was started at 08:54:25, and the installation proper was over at 08:54:28, when the script issued a **dig** command in order to verify the installation. The last access to this script was at 08:54:43, when the unpacked contents of the tar archive, and the archive itself were removed. Note that this interpretation is consistent with the name server's startup messages which were recovered from apollo's swap. These messages date the start of **named** at 08:54:25, too.

08:55 (07:58 CST)

From 08:55:30 until 08:55:51 system time, the intruder used the **/usr/man/.Ci/addn** program to add one or more /16 networks to **/usr/libexec/awk/addy.awk**. The fact that the ctime of **/usr/libexec/awk** was modified at 08:55:30 makes it highly likely that **addy.awk** wasn't installed before this moment. (Which is what we expect from the observations made so far.)

At 08:55:58, the intruder used the **do** script to remove his traces (the `adm1` and `own` accounts) from **/etc/passwd** and **/etc/shadow**. It's interesting to note that this script actually leave a rather visible trace themselves: While removing `own`, the system's `shutdown` account is removed from the user database.

08:56 (07:59 CST)

At 08:56:02-08:56:04, the intruder used — again — the **snap** script to remove traces from various log files. It is not known for what traces he was looking here.

The file system activities from 08:56:05 until 08:56:10 show the intruder removing the various source distributions he had — most likely — unpacked in **/usr/man/.Ci**. This clean-up was performed by the script **rmS**, which was last read at 08:56:11.

At 08:56:25, we observe how the intruder starts the instance of **bx** he left in **/usr/man/.Ci**. This explains the atime discrepancy observed above when he installed the program to **/bin**. **bx** in turn uses **/usr/bin/uncompress**, which can be confirmed by running **strings**(1) on the **bx** binary.

08:56-08:58 (07:59-08:01 CST)

At 08:56:59, we observe the traces of the **chmod-it** script, which sets rather restrictive permissions on various binaries on the system which normally run with special privileges.

At 08:57:06, the intruder runs **mkdir**, followed by **chown** at 08:58:19.

The fact that **/tmp** was accessed at 08:58:08 may be a trace of a

```
cd /tmp
```

at this point of time. This is, however, not certain.

At 08:58:26, the traces hint that the intruder typed:

```
su drosen
```

We note that `drosen`'s **.bashrc** is the only startup file in his home directory which was accessed at this point of time. All other "dot-files" were last accessed on Nov 5. This means that the shell started was interactive, but not a login shell (which would have been spawned by typing `su − drosen`). Note that the access to **/lib/security/pam_xauth.so** at this point of time is justified by a look at **/etc/pam.d/su**.

For the activities which follow (08:58:26-08:59:07), we have once again two sources of information: In addition to the file time stamps, we can use `drosen`'s **.bash_history** file, which we quote in its entirety.

```
gunzip *
tar -xvf *
rm tpack*
cd " "
./install
exit
```

Note that there is no indication that the bot is actually run.

08:59 (08:02 CST)

At 08:59:14, the intruder is `root` again, and removes the files he left back. The mtime and ctime on **/dev**, and the atime on **/bin/rm**, are hints that the intruder may have had his files reside in a subdirectory of **/dev**. Inspecting **/dev**'s inode with **icat**, we find the string `tpacker` in **/dev**, which further supports this hypothesis.

09:02 (08:05 CST)

As one of his last activities, the intruder tested his **ssh** backdoor at 09:02:22. We look at the time stamps in a bit more detail:

```
Nov 08 00 09:02:22       26 .a. -rw-r--r-- root     root     mnt/etc/host.conf
                      67580 .a. -rwxr-xr-x root     root     mnt/lib/libnss_dns-2.1.3.so
                         19 .a. lrwxrwxrwx root     root     mnt/lib/libnss_dns.so.2 -> libnss_dns-
                     169720 .a. -rwxr-xr-x root     root     mnt/lib/libresolv-2.1.3.so
                         18 .a. lrwxrwxrwx root     root     mnt/lib/libresolv.so.2 -> libresolv-2.
Nov 08 00 09:02:23       68 .a. -rw-r--r-- root     root     mnt/etc/hosts
                       1567 .a. -rw-r--r-- root     root     mnt/etc/protocols
Nov 08 00 09:02:28       10 .a. lrwxrwxrwx root     root     mnt/usr/tmp -> ../var/tmp
                       1024 m.c drwxrwxrwx root     root     mnt/var/tmp
                        184 mac -rw-r--r-- root     root     mnt/var/tmp/nap
```

We observe that, at 09:02:22, the dynamic library responsible for DNS lookups was read. This is a strong sign for an incoming connection which has triggered a (reverse) lookup. At 09:02:23, **/etc/hosts** and **/etc/protocols** are read, what supports our hypotehsis.

At 09:02:28 (after the user entered his password), **/usr/tmp/nap** is written. The file's contents fully confirm our hypothesis:

```
+-[ User Login ]-------------------- --- --- - -
| username: root password: tw1Lightz0ne hostname: c871553-b.jffsn1.mo.home.com
+----------------------------------- ----- --- -- -- -
```

At 09:02:30, the intruder looks who's logged in using **w**(1). At 09:02:31, he most likely verifies the proper functioning of his root-kitted **ps** replacement.

09:02-09:03 (08:05-08:06 CST)

Then — from 09:02:42 until 09:03:12 —, he edits **/etc/inetd.conf** using the **pico** editor, most likely removing his original back door.

At 09:03:12, **killall** was most likely executed. The reason for this is unclear.

Finally, at 09:03:15, we observe the last read access to `root`'s **.bash_logout** file:

```
Nov 08 00 09:03:15       24 .a. -rw-r--r-- root     root     mnt/root/.bash_logout
                       3124 .a. -rwxr-xr-x root     root     mnt/usr/bin/clear
                       1143 .a. -rw-r--r-- root     root     mnt/usr/share/terminfo/v/vt100
                       1143 .a. -rw-r--r-- root     root     mnt/usr/share/terminfo/v/vt100-am
```

/usr/bin/clear is executed from **.bash_logout**, and the access to the terminfo file reveals that the user's `TERM` environment variable was set to `vt100` in this session, too. (See also 5.1.)

7.3. Securing the evidence

The remainder of `timeline.txt` shows how the evidence was secured. We do not examine this in detail.

8. More tools from the attacker's chest.

In this section, we examine the unused tools the attacker left back in his chest in **/usr/man/.Ci**.

addps
This simple shell script adds entries to **/dev/ptyp**, the configuration file of the trojaned **top** and **ps** utilities.

find Inspecting the binary with **strings**(1), we find more references to lrk4:

```
# strings -a find | fgrep lrk
/home/aarons2/. /.d/lrk4/findutils/find/
/home/aarons2/. /.d/lrk4/findutils/lib/
```

Thus, we claim that this binary is the well-known trojaned **find** implementation from lrk4.

inetd
This file is statically linked and stripped. Our hypothesis is that this is another trojan from lrk4. Investigating this binary with **strings**(1), we find a reference to **/bin/sh**, in the immediate neighborhood of the string `lpd`, and finally — and nearby — the string `rfe`. All these are typical for lrk4's (or lrk5's) trojaned version of **inetd**.

killall
This is another libc5 binary. Applying **strings**(1), we find the following interesting character sequences:

```
hackinit
hack_list
hackcheck
fp_hack
/dev/.oz/p
```

We claim that this is an lrk version of **killall**.

needz
This simple shell script echoes the URLs of **screen** and **pico** packages.

pstree
This is a binary which was linked against glibc. It contains a reference to **/usr/man/.p**, and the following strings:

```
h_st:T(0,20)=s136next:(0,21)=*(0,20),0,32;hack_type:(0,1),32,32;\
hack_cmd:(0,22)=ar(0,1);0;127;(0,2),64,1024;;
hackinit:F(0,19)
hackcheck:F(0,1)
hack_list:G(0,21)
fp_hack:G(0,42)
hackinit
hack_list
hackcheck
fp_hack
```

We claim that this is another lrk trojan.

q

qs According to some strings found in the binaries, these are the client and server for some secure connection system by *Mixter*. More precisely, this seem to be compiled versions of the client and server from the **Q2** package which is available from http://mixter.warrior2k.com/progs.html.

sp.pl

This is a perl script which postprocesses **linsniffer** output.

syslogd

This binary references **/usr/man/.l**. It's statically linked, and stripped. Our hypothesis is that this is another lrk trojan, and comes from the same source as the trojaned **inetd**.

<space>/Anap

This simple shell script purges **/usr/tmp/nap**.

paki/stream.c

This is the **stream.c** denial of service attack which was discussed on the **bugtraq** mailing list in January 2000.

paki/slice2

It may seem likely that this program is related to the SYN-flooding tool at http://www.computec.ch/exploits/dos/slice2.c. However, a look with **strings**(1) quickly reveals that the **slice2** binary found on apollo was built from a different source, but takes similar parameters.

We notice that this binary was dynamically linked against libc5. The plan for analyzing this program is, thus, to use the dynamic linker in order to create a laboratory environment in which we can easily observe any packets crafted, without actually sending anything to the network. We then apply the same method to the known **slice2**, and analyze the differences between both programs.

We first use **nm**(1) to get an overview of the functions used. Apparently, the key system calls and library functions used are **sendto**(2), **rand**(3), **random**(3), and **socket**(2). The known **slice2.c**, additionally invokes **fork**(2) in order to create many processes which are run in parallel. We prepare a simple shared library (see `libfake.tar`) which provides fake versions of these functions. In particular, **rand**(3) and **random**(3) are changed to return reproducible data (and can now be used interchangeably), **socket**(2) just returns increasing positive numbers, **fork**(2) simulates a single successful fork call from the child's perspective, and **sendto**(2) logs any data transmitted to a file given in the environment variable `SENDTO_LOG`. The format used is similar to the one produced by

```
tcpdump -x
```

and is, in particular, understood by Wietse Venema's **tcpdumpx** utility.

Now, we run the following commands (with **slice2.apollo** being a copy of the binary found on apollo):

```
$ LD_PRELOAD=./fake.so SENDTO_LOG=sendto.log \
     ./slice2 127.0.0.1 127.0.0.1 1 200 1
$ LD_PRELOAD=./fake.so SENDTO_LOG=sendto.log.apollo \
     ./slice2.apollo 127.0.0.1 127.0.0.1 1 200 1
```

After a short time, we interrupt the programs by sending SIGINT. The log files produced are included (the log file generated by the apollo **slice2** was, however, manually truncated afterwards). We process them with **tcpdumpx**, extracting packet headers:

```
$ tcpdumpx < sendto.log.apollo | fgrep -A3 HDR > headers.apollo
$ tcpdumpx < sendto.log | fgrep -A3 HDR > headers
```

(The files produced are included.) A typical packet header generated by the publicly known **slice2** looks like this:

```
IP_HDR=20 IP_OPT=0 TCP_HDR=20 TCP_OPT=-20 DATA=20 FLAGS=SYN

IP_HDR   45  00  00  28  00  00  00  00  ff  06
        vhl tos len len id  id  off off ttl pro
IP_HDR   bd  cd  7f  00  00  01  7f  00  00  01
```

```
        sum sum src src src src dst dst dst dst
TCP_HDR  01  00  00  01  02  00  00  00  03  00
        src src dst dst seq seq seq seq ack ack
TCP_HDR  00  00  00  02  ff  ff  f9  fc  00  00
        ack ack off flg win win sum sum urp urp
```

On the other hand, a packet header generated by the version found on apollo looks like this:

```
IP_HDR=20 IP_OPT=0 TCP_HDR=20 TCP_OPT=0 DATA=10200 FLAGS=SYN

IP_HDR   45  00  28  00  00  01  00  00  1e  06
        vhl tos len len id  id  off off ttl pro
IP_HDR   75  12  7f  00  00  01  7f  00  00  01
        sum sum src src src src dst dst dst dst
TCP_HDR  00  02  00  01  00  00  00  03  00  00
        src src dst dst seq seq seq seq ack ack
TCP_HDR  00  00  50  02  ff  ff  b1  da  00  00
        ack ack off flg win win sum sum urp urp
```

We immediately notice that both programs generate TCP packets with the SYN flag set, and with varying source and destination ports. Thus, both programs are indeed being used to SYN flood a target.

Looking more closely at the individual entries in the header, we note that apollo's version of **slice2** writes out the packet length as 0x2800 (10240), while the publicly available version writes it out as 0x0028 (40). The latter value correctly describes the packet sent. This may point to a missing **htons** call in apollo's version of **slice2**. On the other hand, the publicly known version of **slice2** produces packets with the TCP header length set to 0, while apollo's version correctly sets the header length nibble to 5, corresponding to a header length of 20 bytes. Finally, we note that the TTL of the packets sent differs between the two versions of **slice2**.

scan/amd/a.sh

This script is used to probe the networks 206.110, 206.111, ... on port 111, using **scan/amd/pscan**.

scan/amd/pscan

scna/amd/pscan.c

The port scanner just mentioned, and its source code

scan/amd/amdx

From the strings found in this binary, this is a close relative of the amd exploit posted to the **bugtraq** mailing list by *duke <duke@VIPER.NET.AU>* on Thu, 2 Sep 1999. This vulnerability is also covered by CERT Advisory CA 99.12, and in RHSA 1999:032. It was fixed in **am-utils-6.0.1**, which is the version of the package which was installed by the intruder on apollo.

scan/amd/ben

scan/amd/ben.c

This program launches an RPC info query against a target host, and is used to find out whether or not a certain RPC service exists on that host. In this case, the `RPCID` macro is defined to be 300019, which corresponds to the automounter. (See also **/etc/rpc**.)

scan/bind/ibind.sh

This script uses **scan/bind/pscan** in order to look for name servers. It then uses **dig**(1) to find those name servers' versions, and classifies them as vulnerable or not vulnerable according to the version numbers. Versions above 8.2.2 are not considered vulnerable.

scan/daemon/lscan2.c

This is a port scanner which looks for hosts with running name servers, POP, IMAP or FTP daemons, or **mountd**.

scan/daemon/z0ne

This is a tool which performs an AXFR query for a domain given on the command line, and writes all IP addresses found in that zone to `stdout`. Source code for **z0ne** can be found in current versions of **bind-contrib.tar.gz** as distributed by the Internet Software Consortium at ftp.isc.org.

scan/port

This directory contains the well-known port scanner **strobe**.

scan/statd/classb

From the strings contained, and a short glance at a disassembler listing generated with **gdb**, this program is most likely used to quickly create a list of IP addresses to be scanned.

scan/statd/r

From the strings contained within this file, this program knows itself as `rpcscan`, and is used to scan for hosts on which a certain RPC service is running. It is likely that another instance of this tool was responsible for the RPC Info Query observed by **snort** immediately before the break-in occured.

It is likely that this program comes from source code very close to the one which can be found at http://www.self-evident.com/security/scanners/rpcscan.c.gz. However, when applying **strings**(1) to the binary which was found on apollo, we notice the strings

```
: %i PORT:%d
, %i PORT:%d
```

which are not present in the publicly available source code.

scan/statd/statdx

Investigating this binary with **strings**(1), this seems to be a compiled version of the **statdx** exploit posted to **bugtraq** by *ron1n - <shellcode@HOTMAIL.COM>* on Aug 5 2000. This is the same program which was most likely used to attack apollo. For RedHat linux, the problem fixed in **nfs-utils-0.1.9.1-1**, which is the version of the package which was installed on apollo by the intruder. This package was announced in RHSA 2000:043-03.

scan/wu/wu

From the strings in this binary, this is a compiled version of the attack originally posted to the **bugtraq** maling list by *tf8 <tf8@zolo.freelsd.net>* on Jun 23 2000. The vulnerability is covered by CERT Advisory CA-2000-1 and by RHSA 2000:039-02. We note that this problem was fixed for RedHat in **wu-ftpd-2.6.0-14.6x**, which is precisely the version of the package which was installed by the intruder on apollo.

scan/wu/fs

From the strings in this binary, this is a program which scans networked hosts for various vulnerabilities. The binary contains a reference to `www.r0xcrew.org`, which does not exist.

scan/x/pscan

scan/x/pscan.c

This is another instance on the port scanner which is also present in other subdirectories of **scan/**.

scan/x/xscan

This shell script uses **pscan** to look for hosts which are running an X server on port 6000.

scan/x/x

From the strings in this binary, this program is intended to work as an X11 key logger for either individual hosts or entire subnets.

scan/x/xfil

This shell script is apparently intended to scan files for "interesting" areas, containing keywords such as `telnet`, `ssh`, `rsh`, `rlogin`, or `ftp`. This may be intended to help attackers find passwords in keyboard logs generated by **scan/x/x**.

9. References

CERT advisories can be found at http://www.cert.org/.

Red Hat Security Advisories (RHSA's) can be found at http://www.redhat.com/.

An archive of the **bugtraq** mailing list is available at http://www.securityfocus.com/.

More information on the internals of TCP/IP packets can be found in W. Richard Stevens, *TCP/IP Illustrated, Volume 1*, Addison-Wesley 1994.

The published source code for linux root kits is available from ftp://ftp.technotronic.com/unix/trojans/.

tcpdumpx is available from ftp://ftp.porcupine.org/pub/debugging/.

The Coroner's Toolkit is available from http://www.porcupine.org/forensics/.

Appendix

[Q1] Identify the intrusion method, its date, and time.

See sections 4.2 and 7.2.

[Q2] identify as much as possible about the intruder(s).

The closest thing to identifying information about the intruder are the time stamps related to network activities:

- At 23:11 CST on Nov 7, 2000, the intruder attacked the system from 216.216.74.2.
- At 07:31 CST on Nov 8, 2000, and at 07:45 and 08:05 CST on the same day, the intruder logged into the hacked system from 24.12.200.186, using telnet and ssh clients.

We do not know whether or not these IP addresses identify systems abused by the intruder, or whether they directly lead to him.

Additionally, there are some conjectures we can make about the attacker from the traces he left behind on apollo, and from his behaviour on the system:

- We can identify various sources from which the intruder's tools come. At least three can be distinguished: There are programs from lrk4 which were linked dynamically against libc5 (such as **/bin/ls** and **/sbin/ifconfig).** Then, there are programs which were linked dynamically against glibc, such as the generic backdoor wrapper found in **/usr/sbin/in.identd**, or **/usr/sbin/tcpd**. Finally, there are statically linked (and stripped) binaries such as **/usr/man/.Ci/syslogd**.
- The attacker used scripts which didn't match the trojan horse binaries he installed on the system. Note the unused rootkit configuration file **/usr/man/p**.
- When the Linux root kit is built from scratch, several binaries share the same configuration file, for instance **netstat** and **tcpd**. This is not the case with the binaries found on apollo.
- The intruder installed a set of RedHat security updates which would apply to earlier versions of the distribution - but actually turn out to be downgrades on more recent RedHat distributions.

We conclude that the intruder most likely did not build his collection of attack tools from source code, but obtained the tools in binary form from various sources.

Additionally, from the installation of **BitchX** (which was actually run), and from the fact that the attacker performed steps to install an **eggdrop** IRC bot, it seems likely that one of his interests is Internet Relay Chat, and possibly abuse of that network.

[Q3] List all the files that were added/modified by the intruder. [...]

See various sections of the report.

[Q4] Was a sniffer or password harvesting program installed?

See sections 3, 6.3, and 7.2 of the report.

[Q5] Was there a "rootkit" or other post-concealment trojan horse programs installed on the system?

See section 3. The rootkit was no obstacle for the investigation, since all tools used resided on a known-good system.

[Q6] What is publicly known about the source of any programs found on the system?

See the report, in particular sections 3, 6.3, 7.2 (08:52:15), and 8.

[Q7] Build a time line of events.

See section 7 of the report.