

Hacker Tools and their Signatures, Part Three: Rootkits

Toby Miller 2001-08-14

Hacker Tools and their Signatures, Part Three: Rootkits

by Toby Miller

last updated August 14, 2001

This is the third installment of a series devoted to examining hacker tools and their signatures. In this installment we will be looking at some of the signatures related to the KOH rootkit. The purpose of this paper is to assist the reader in detecting the KOH rootkit. Through this process, it is hoped that the reader will also learn steps to take to defend against the installation of these types of rootkits.

KOH Rootkit

I first received this rootkit from a friend who asked me to look this over. After looking this over I realized that this rootkit was similar to t0rn and LRK but I figured that a write up on this rootkit was needed.

One of the main goals I had when looking at this rootkit in addition to gaining signatures, was also to look at this from a "how can I prevent this from being installed on my machine" approach. I will be covering this approach a little later in the paper.

The following figure is a list of files that come with the KOH rootkit:

```
total 1100
drwxr-xr-x    3 506      506      4096 Aug  5 21:57 .
drwxr-x---   37 root     root     4096 Aug  5 22:12 ..
drwxr-xr-x    2 506      506      4096 May 10 15:23 ...
-rwxr-xr-x    1 506      506      2979 May 10 15:18 clean
-rwxr-xr-x    1 506      506     138288 Nov  3  2000 dir
-rwxr-xr-x    1 506      506     101924 Nov  3  2000 du
-rwxr-xr-x    1 506      506     52984 Nov  3  2000 find
-rwxr-xr-x    1 root     root      194 Jun  8 19:12 hosts.allow
-rwxr-xr-x    1 506      506     3375 Jun  8 19:14 inetd.conf
-rwxr-xr-x    1 506      506     2802 May  9 19:17 install
-rwxr-xr-x    1 root     root      881 Jun  8 19:16 install7-rwxr-xr-x    1
506      506
9712 Nov  3  2000 killall
-rwxr-xr-x    1 506      506     14162 Nov  4  2000 klogd
-rwxr-xr-x    1 root     root     34286 May  7 14:48 knight
-rwxr-xr-x    1 506      506     71347 Nov  5  2000 login
-rwxr-xr-x    1 506      506     138283 Nov  3  2000 ls
-rwxr-xr-x    1 506      506     30968 Nov  3  2000 netstat
-rwxr-xr-x    1 506      506     28952 Nov  3  2000 ps
-rwxr-xr-x    1 root     root     14791 Jun 10 09:24 pss
-rwxr-xr-x    1 506      506     32281 Nov  3  2000 pstree
```

Figure 1. File listing of the KOH rootkit

Looking at figure 1 we see a few items that stand out like a sore thumb. First, all the way at the top of the figure, we see the three-dot directory (?). That alone should raise a few red flags. Second, we see our typical binaries and installation files (i.e. ls, ps, netstat, install) that KOH replaces and uses. One thing that caught my eyes when I was looking through this rootkit was the owner(s) of these files. Notice that the owner is not root, but is 506. Keep this in mind because when we install this rootkit we will see that the ownerships do not change. We also see in figure a hosts.allow file and an inetd.conf file. Both of these files will be covered in detail a little later in the paper.

Ok, we know what files this rootkit contains; now we want to begin looking for signs that this rootkit has been installed. The first sign and probably the easiest are comparing md5sums. For this paper we will not be looking at the known "good" sums that come with Redhat (or any Linux distro for that matter). Instead let's look at the md5sums of the KOH rootkit only.

1f0a7caf19a9d7521ac820c373d34447	clean
dcd9aaf29c011734bb1ba7e75320494e	dir
e9e8a46584fc6d46d74c33134b76ba3c	du
e1e6c4d13e3a953e5098817963c623ab	find
b24663403f6dfd8edd2bfcba1c7b86b2	hosts.allow
1a68893209db20a29331940165e47c37	inetd.conf
a22b62ed5b9ea18c99ca1cab0738cdfc	killall
eb61ab84ee4a58dc25d29663c142830a	klogd
92b39cae4c02a8578db2659e1c564b6a	knight
60a6f1e51a979f2b29e10ad78151dc6d	ls
d0071f34a1cbb2fd148fb568bb9f9d5e	netstat
6424dc5699b0e01172b884f7212fc289	ps
bfdc2fd2e6d8b13327cef00dded02691	pss
629fb28f22fb4fb30c07310c7cd21067	pstree
e204a2f4983b02e0d6d72c6d14920174	terminal
a75dfd0fd9135098eb43b58e6e190acd	top
3886124586863722fc3522affdcff638	vdir

Figure 2. KOH Md5sums

Figure 2 shows us the md5sums associated with the KOH rootkit. Again, this is a good starting point when searching for a rootkit similar to KOH.

The next step in my investigation is to look at the binaries KOH replaced. There are a couple of techniques that can be used to do this with Linux. The reason I look at the binaries first instead of looking for directories is that binaries hide keys that unlock doors to other directories a rootkit may use. Let's pick a couple of binaries to analyze and figure out more details.

In this discussion we will be talking about tools such as lsattr, ltrace, strace, stat and strings. All of these tools come FREE with most versions of Linux. Lets begin by looking at /bin/ps. Yes, we could identify that /bin/ps had been hacked by looking at the md5sum but that would ruin all of the fun. First, let refer back to figure 1. In Figure 1. we see that ps is owned by UID 506 and GID 506. Well, this holds true after installation of the rootkit as well. What does that mean to us? This means that if you were to run ls -la you would see that the ownership of the file is different than normally (normally /bin/ps is owned by root). That alone should set off a red flag. If not, remember that SIZE DOES MATTER. Normally, a typical /bin/ps is 64K in size, whereas a "rooted" /bin/ps is almost 29k in size. Once you have confirmed that your binaries have been "rooted" and want to get a head start on finding more rootkit files then you might run strings on a binary such as /bin/ps. Figure 3 shows us the results of running strings on /bin/ps:

```
6WVSz2Zt
},hg
D$$Phu
UWVS
[^_]
[^_]
[^_]
PRQShR
90t.
/var/.prc
NR  PID      STACK      ESP      EIP  TMOUT  ALARM  STAT  TTY    TIME  COMMAND
  PID  TTY  MAJFLT  MINFLT   TRS    DRS   SIZE   SWAP   RSS   SHRD   LIB   DT
COMMAND
  PID  TTY  STAT   TIME   PAGEIN  TSIZ  DSIZ   RSS    LIM  %MEM  COMMAND
  UID   PID  SIGNAL  BLOCKED  IGNORED  CATCHED  STAT  TTY    TIME  COMMAND
PPID   PID  PGID   SID  TTY  TPGID  STAT   UID    TIME  COMMAND
USER           PID  %CPU  %MEM  SIZE   RSS  TTY  STAT  START  TIME  COMMAND
  FLAGS  UID   PID  PPID  PRI  NI    SIZE   RSS  WCHAN          STA  TTY  TIME
COMMAND
  PID  TTY  STAT   TIME  COMMAND
sort
unrecognized long sort option
help
version
ps: unknown long option
```

Figure 3. KOH /bin/ps strings output

Figure 3 does not show the whole contents of strings, instead I have shown the part of strings that matters the most. What may that part be? Look at /var/.prc. /var/.prc instructs /bin/ps what processes to hide. Figure 4. shows us the contents of .prc:

```

3 scr
3 .m3 kni
3 kegg
3 kbn
3 aux
3 az
3 in.t
3 wu
3 bsh
3 term
3 klo

```

Figure 4. .prc

The next binary we will look at is /bin/lis. Again, lets check this binary owner:

```

-rwxr-xr-x  1 506      506      138283 Nov  3
2000
/bin/lis

```

Figure 5. ls results

Again, look at the owner of this file; it's 506 and not root. Also, look at the file size. The file size is 138283 bytes. Normally, /bin/lis is 46k in size. As discussed earlier this should raise some red flags. Let's take a look at the output of strings when ran against this version of /bin/lis:

```

/usr/local/share/locale
fileutils
GNU fileutils-3.13
vdir%s - %s
/var/.kls
//DIREDD//
//SUBDIREDD//
POSIXLY_CORRECT
COLUMNS
ignoring invalid width in environment variable COLUMNS: %s
TABSIZ
ignoring invalid tab size in environment variableTABSIZ: %s
abcdefghijklmnopqrstuwxABCDEFGHI:LNQRST:UX178

```

Figure 6. KOH /bin/lis strings output

Figure 6. shows us the output of strings when ran against /bin/lis with KOH. Again, for size sake I did not show

the whole output. But look at /var/.kls, that is not normal. Let's take a look at what is hidden in /var/.kls:

```
knet.tgz
knet
identd
identd.c.
koh.l.
koh.s.
koh.n.
koh.c.
koh.u.
koh.m.
koh.conf
koh.tcl
eggdrop
kegg
kohnet
.kohnet
koh.settings
scripts
.mk
screen
```

Figure 7. /var/.kls output

Figure 7. shows us a partial list of the files and directories this version of /bin/lis will hide. Before continuing I would like to point out that there are other tools that can be used instead of strings. The tools I use are ltrace and strace. I will use ltrace in our next binary that we will look at.

Netstat is our final binary that we will look at. A normal netstat binary is 80k in size. With KOH installed the size of netstat is 30k. We know from the previous two binaries that the UID for the file will be 506. Let's take a look at an ltrace example and figure out what we are looking at.

```
fopen("/var/.adr", "r"
open("/var/.adr", 0, 0666
SYS_open("/var/.adr", 0, 0666) = -2
```

Figure 8. ltrace example

Figure 8. shows us an ltrace example. This was taken from the KOH netstat. Here we that it points out a .adr file. Let's see what the file looks like:

```

1 63.24
1 63.15
192.168.1.1
2 192.168.1.10
2 192.168.1.111
3 6667
3 7000
3 22121
3 52121
3 42121
3 911
3 6669
3 1524
4 1524
3 666
4 666
3 65535
3 42069

```

Figure 9. /var/.adr results

Figure 9. is just a partial list of addresses or ports that this file will hide using the KOH's netstat (some addresses have been changed to protect the guilty or innocent). Now we know that KOH hides three files under the /var directory. Let's move on and see where else KOH hides files.

After looking through the install file I noticed that KOH hides file under /usr/bin/no. How can we find that out? Well, I was lucky and had the install file to look at but there is a nice little binary that comes with Linux that I like to use in these situations. The binary is lsattr. What is lsattr? Lsattr will list the attributes of the Linux EXT2 file system.

When I am searching for a rootkit like KOH I use lsattr -a. The -a list all files in directories. The following is an example of using lsattr -a /usr/bin | grep no to detect KOH:

```

----- ./emacs-nox
----- ./nohup
----- ./sgmlnorm
----- ./shownonascii
----- ./anno
----- ./whatnow
----- ./no

```

Figure 10. lsattr output

Ok, we now know that KOH hides a file named knight under /usr/bin/no. When I found that out I looked at the

file its self and found out that it looked like an IRC bot. While playing with this file I caused it to core dump. Just for FYI here's the results of the core dump:

```
GNU gdb 5.0rh-5 Red Hat Linux 7.1
Copyright 2001 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB. Type "show warranty" for details.
This GDB was configured as "i386-redhat-linux"...
Core was generated by `ps aux          '.
Program terminated with signal 5, Trace/breakpoint trap.
Reading symbols from /lib/i686/libc.so.6...done.
Loaded symbols for /lib/i686/libc.so.6
Reading symbols from /lib/ld-linux.so.2...done.
Loaded symbols for /lib/ld-linux.so.2
Reading symbols from /lib/libnss_files.so.2...done.
Loaded symbols for /lib/libnss_files.so.2
#0  0x4000e4f1 in _dl_debug_state () at dl-debug.c:56
      in dl-debug.c
(gdb) (gdb) quit
```

Figure 11. Core dump of the knight program

Let's take a look at some more files that KOH installs. KOH replaces the inetd.conf file. I instead of showing the whole file I am just going to share what is added to inetd.conf file.

```
shell  stream  tcp      nowait  root    /usr/sbin/tcpd  in.rshd
login  stream  tcp      nowait  root    /usr/sbin/tcpd
in.rlogind
telnet  stream  tcp      nowait  root    /usr/sbin/tcpd
in.telnetd
telnetd stream  tcp  nowait  root  /usr/sbin/tcpd  in.telnetd
```

Figure 12. Inetd.conf additions

Without going into to many details we can see that it opens up quite a few services. With netstat not working can we spell NMAP? KOH also adds some accounts into /etc/shadow. The following are the accounts that KOH add:

```
lpd:101:11235::99999:::135640292
admin:101:11235::99999:::135640292
```

Figure 13. Password entries and results

To save some space, figure 13 also shows the passwords that KOH uses to access the box.

Protection Against Rootkits

Rootkits are not at all new, but there are still a lot of people that have been "rooted" and have had one installed. This section will briefly cover some of the techniques that I recommend to use in order to protect yourself against rootkits. This is not a comprehensive list just a few recommendations.

1. Install patches. Yes, I know this is old news but it still is a big problem.
2. Use a securing script such as bastille. This script is a lifesaver for the busy administrator. Use it.
3. Make md5sums of many of the "commonly hacked binaries". This would allow you to compare those sums against the sums that you suspect.
4. Use chatr and then delete it. Chatr changes the attributes of files. If you use the -I option on those same commonly hacked files and then delete chatr you would be surprised with the results. I used this method and installed T0rn (version 6) and t0rn installs as normal and does not notify the person that all binaries did not install. This technique will probably give you 30 seconds to 1 minute depending on the attackers skill level.
5. Use xinetd. I love this program. I think it makes it easier to manage your ports with xinetd.

Conclusions

This rootkit was fun to analyze and interesting to learn about. There are many tools and procedures that can be used to detect these types of rootkits. This paper does not list them all. Hopefully, the techniques and tools I use here will help anyone who has to dissect a rootkit in future.

Relevant Links

[Hacker Tools, Part One: Hacker Tools and Their Signatures, Part One: bind8x.c](#)

Toby Miller

[Hacker Tools, Part Two: Juno and Unispl0it](#)

Toby Miller

[Privacy Statement](#)

Copyright 2006, SecurityFocus