

Application of a Methodology to Characterize Rootkits Retrieved from Honeynets

John Levine, Julian Grizzard, Henry Owen, *Members, IEEE*
levine@ece.gatech.edu, grizzard@ece.gatech.edu, owen@ece.gatech.edu

Abstract-- Techniques and methods currently exist to detect if a certain type of rootkit has exploited a computer systems. However, these current techniques and methods can only indicate that a system has been exploited by a rootkit. We are currently developing a methodology to indicate if a rootkit is previously known or if it is a modified or entirely new rootkit. We present in this paper an application of our methodology against a previously unseen rootkit that was collected from the Georgia Tech Honeynet. We conduct our analysis process against this rootkit and are able to identify specific characteristics for subsequent detections of this rootkit. This ability will provide system administrators, researchers, and security personnel with the information necessary in order to take the best possible recovery actions. This may also help to detect and fingerprint additional instances and prevent further security instances involving rootkits.

Index Terms-- forensics, Honeynet, rootkits, signatures

I. INTRODUCTION

Rootkits are a phenomenon that has recently drawn attention. Prior to rootkits, system utilities could be trusted to provide a system administrator with accurate information. Modern crackers have developed methods to conceal their activities and programs to assist in this concealment [1]. Rootkits are a serious threat to the security of a networked computer system.

Modern operating systems are subject to a variety of exploits that allow hackers to gain root access on networked computer systems. This in turn, provides hackers with the ability to install rootkits on these compromised systems. System administrators need to be aware of the threats that their computers face from rootkits as well as the ability to recognize if a particular rootkit has been installed on their computer system.

Part of our efforts is the use of a Honeynet to collect new rootkit type exploits. Rootkits are also available from other sources including the Internet. The Honeynet, however, offers us an opportunity to collect rootkits that may not have been previously seen by other researchers [2]. These rootkits are targeted against actual live systems on the Honeynet that have been compromised by a hacker. We believe that the Honeynet offers us an actual opportunity to collect existing, modification to existing, and entirely new rootkit exploits.

On 1 June 2003 a system installed on the Georgia Tech Honeynet was compromised, allowing a hacker to gain root level access. The hacker then installed a rootkit on this system. No traffic should have been going to or come from this system since it is a Honeynet machine [3]. By following the principles of data capture and data control we were able to capture the exploit that the hacker executed against this system and prevent this system from being used to compromise any other systems.

A. Target System Description

The target system that was employed on the Honeynet was a standard version of the Red Hat Linux 6.2 operating system running the Linux 2.2 kernel. This system was configured to install all available packages and no special modifications were made to this system. The install process was a default installation for this configuration. No additional services besides those that were started by the default installation were enabled on the target system. The following ports were opened on this system:

21 ftp	23 telnet	25 smtp
79 finger	98 Linux conf	111 sun rpc
113 auth	513 login	514 shell
515 printer	954 unknown	1024 kdm
1025 listen	1033 net info	6000 x11

B. Method of Compromise

At 10:34 AM ZULU on June 1, 2003 an exploit was launched against the target system on port 21 (ftp daemon) to attempt to gain root level access. The ftp server running on Red Hat Linux 6.2 is wu-ftpd2.6.0(1) ftp daemon, the default ftp server. Exploits that allow a hacker to gain root level access have been published against this particular service and are available on the Internet. This attack was successful and the hacker was able to gain root level access on the target system. Figure 1 shows the start of the top stream that was extracted from the Honeynet data concerning this attack. The string RNFR ./ is a signature of the WU-FTP exploit for the ftp server that is running on this system [4].

2. Install a kernel level debugger on this system. The installation of the kernel level debugger will probably require the system to be recompiled with a custom kernel.
3. Install and run a file integrity checker program on this system. Select target directories based on an analysis of the installation that occurred when the rootkit was originally acquired.
4. Install a rootkit detection program such as chkrootkit on the target system. This will detect many existing system utility rootkit exploits and may help to identify modification to existing as well as entirely new rootkit exploits.
5. Install a program such as kern_check [7] to check the integrity of the system call table within the kernel. Run this program on the target system to establish a baseline and ensure that the kernel integrity has not been compromised on this initial installation.
6. Make a copy of the kernel text segment of memory via /dev/kmem for future comparison. We have presented a method to do this in our paper titled "A Methodology to Characterize Kernel Level Rootkit Exploits that Overwrite the System Call Table" to be present at SoutheastCon 2004 [8]. The kernel text segment of kernel memory should remain consistent for a particular kernel build. Any deviation between this copy and a future copy could indicate that the kernel may have been compromised by a kernel level rootkit exploit. A more detailed analysis of kernel space can then be conducted via the kernel debugger (kdb) program that has previously been installed.
7. Run the file integrity checker program and the rootkit detection program on the target system prior to infecting the system with the rootkit to establish a baseline for comparison between a clean and infected system.
8. Install the rootkit on the target system. Follow the installation steps that were used when the rootkit was initially acquired for analysis.
9. Run the file integrity checker program on the system and note the results. The presence of certain types of rootkits should be indicated by the results of the file integrity checker program but other types of rootkits, specifically those that target the kernel, may not be detected by this type of program.
10. Run the rootkit detection program (chkrootkit [9]) on the system that has been infected with the rootkit that is being analyzed. If this is a previously known system utility rootkit then a program such as chkrootkit should be able to detect the presence of this rootkit. If the file integrity checker program detected a change to a system utility binary program file but the rootkit detection program did not detect the presence of a rootkit then we can assume that we are dealing with either a modification to an existing system utility rootkit or an entirely new system utility rootkit.
11. Run the kernel integrity check program on the target

system. If the system call table was modified or redirected, then this program should be able to detect what has been modified. Make note of each system call that is indicated as being modified as well as a total count of the number of system calls that have been modified by this rootkit. The system calls that a rootkit modifies can establish a signature for a specific kernel level rootkit [10]. This program may not be able to detect some other modification to the kernel text code segment of the kernel.

12. Make a copy of the kernel text code and compare it against the original copy of the kernel text code that was prepared before infecting this system with the rootkit. A difference between these two files may indicate the presence of a kernel level rootkit. This is especially significant if the existing kernel integrity checker programs failed to detect any modification to the kernel. This would indicate that one is dealing with a new type of kernel level rootkit that does not target the system call table. The system call table is the normal avenue of attack for hackers who are attempting to create a kernel level rootkit.

The results of these steps can be used to classify a rootkit exploit as an existing, modification to an existing, or an entirely new rootkit.

III. ANALYSIS PROCESS

The Georgia Tech HoneyNet was able to capture the entire exploit session of a system compromise including the downloaded files as well as the remote machines that the hacker connected to from the compromised machine. This provided us with the scripts and files that were used by the hacker to install the r.tgz rootkit. Preliminary analysis of these files gave us an indication of how this rootkit could be installed on a target system similar to the system on the HoneyNet. We then set out to install the r.tgz rootkit on a target system to analyze and classify it.

We have initially set up a baseline system that consists of the same operating system as the system that was compromised on the HoneyNet. In this case it is Red Hat 6.2 running the Linux 2.2.14 kernel. Following the procedures outlined in Section II, we installed a kernel level debugger (kdb) on this system as well as a file integrity checker program (AIDE [11]). We then installed a known rootkit detection program (chkrootkit [9]) and made a copy of the kernel text segment via /dev/kmem. Prior to infecting this system with the r.tgz rootkit we ran the AIDE and chkrootkit program to establish a clean baseline for analysis and classification. We then infected this system with the r.tgz rootkit.

The first check that we ran on the infected system is the file integrity check to determine what files have been added, changed, or deleted. Running the AIDE program on the infected system indicated that 2 files had been added to the infected system and 178 files had been changed by the r.tgz program. This is a large number of files and initial analysis

of the install scripts for this rootkit does not indicate that all of these files are being modified. Follow on analysis was conducted on these modified files to determine the nature of these changes. Figure 4 shows the results of running AIDE on the infected system.

```

root@localhost.localdomain: /check
File Edit Settings Help
@aide_db"AIDE found differences between database and filesystem!!
Start timestamp: 2003-12-08 11:55:53
Summary:
Total number of files=2106,added files=2,removed files=0,changed files=178

Added files:
added:/usr/bin/strings
added:/sbin/sendmail
  
```

Figure 4: AIDE results on r.tgz infected system

The next step was to run the known rootkit detection program (chkrootkit) on the target system. Running this program utilizing the binaries that are currently installed on the target system only results in the identification of one system binary as being infected. The binary that is indicated as being infected is *ifconfig*. The chkrootkit program also detects five suspicious files and possible infections by the “Showtee” and “Romanian” rootkits.

The recommended method of using the chkrootkit program is to use known good binary files. Known good binary files can be copied to a read-only disk and the files on this disk can be accessed by the chkrootkit program using the `-p` switch. Using known good binary to check the system results in the identification of the same changes identified in the previous paragraph plus 5 additional binaries being identified as being infected on the target system. These five binary files are: *du*, *ifconfig*, *killall*, *ls*, and *psree*. The check using known good binaries also indicates the following under the lkm check: 1 process hidden from readdir command, 15 processes hidden from ps command. The differences between the output using known good binaries and the output using the binaries currently installed on the system indicate that the r.tgz rootkit modified some of the system binary files that are used by the chkrootkit program to check the system status. As a result we can conclude that known good binaries should always be used while running the chkrootkit program. The five files detected as being changed by the chkrootkit program are also detected as being changed by the AIDE file integrity checker program. These results are used in our methodology to classify this rootkit exploit.

The next step in the methodology is to verify the integrity of the kernel. Running the kernel check program (*kern_check*) on the target system utilizing a known good `/boot/System.map` file indicates that there is a mismatch of 21 system calls between the kernel and the known good `/boot/System.map` file. The results of the *kern_check* program are shown in figure 5.

```

root@localhost.localdomain: /check
c[root@localhost /check]# ./kern_check /boot/System.map
kaddr = c31ac000
WARNING: (kernel) 0xc31ad284 != 0xc0108ca8 (map) [sys_fork]
WARNING: (kernel) 0xc31ad6a4 != 0xc0125ef0 (map) [sys_read]
WARNING: (kernel) 0xc31ad5d8 != 0xc0125fac (map) [sys_write]
WARNING: (kernel) 0xc31ada01 != 0xc01259f4 (map) [sys_open]
WARNING: (kernel) 0xc31ada99 != 0xc0125b44 (map) [sys_close]
WARNING: (kernel) 0xc31ae02b != 0xc0125a88 (map) [sys_create]
WARNING: (kernel) 0xc31ae0af != 0xc012d8c8 (map) [sys_unlink]
WARNING: (kernel) 0xc31ad9e0 != 0xc0108cf4 (map) [sys_execve]
WARNING: (kernel) 0xc31adbe4 != 0xc012af4c (map) [sys_stat]
WARNING: (kernel) 0xc31adcef != 0xc012b0cc (map) [sys_fstat]
WARNING: (kernel) 0xc31adb60 != 0xc0125014 (map) [sys_utime]
WARNING: (kernel) 0xc31adaf0 != 0xc01109f8 (map) [sys_kill]
WARNING: (kernel) 0xc31ad0cc != 0xc010e45c (map) [sys_olduname]
WARNING: (kernel) 0xc31adc68 != 0xc012b00c (map) [sys_lstat]
WARNING: (kernel) 0xc31ae12f != 0xc012b1c8 (map) [sys_readlink]
WARNING: (kernel) 0xc31add47 != 0xc012afac (map) [sys_newstat]
WARNING: (kernel) 0xc31addd1 != 0xc012b06c (map) [sys_newlstat]
WARNING: (kernel) 0xc31ade5b != 0xc012b148 (map) [sys_newfstat]
WARNING: (kernel) 0xc31ad214 != 0xc01080bc (map) [sys_clone]
WARNING: (kernel) 0xc31ad361 != 0xc012f0fc (map) [sys_getdents]
WARNING: (kernel) 0xc31ad2f1 != 0xc0108cdc (map) [sys_vfork]
[root@localhost /check]#
  
```

Figure 5: Results of kern_check program

This is an indication that the kernel of the target system may have been compromised by the r.tgz rootkit. Checking the current kernel text segment code against the previously archived version of the kernel text segment code indicates that the kernel has been compromised. The previously archived version of the kernel text segment code was built when the target system was first compiled.

Analysis of the kernel using kdb indicates that the pointer to the system call table is being redirected to a new instance of this table. The correct system call table address is 0xc0248928 and can be retrieved from the `/boot/System.map` file. The current system call table address as displayed by kdb in kernel memory is 0xc31ac000. The following is the results of running a kdb query on the system call interrupt within kernel space. The returned call statement should refer to the address of the system call statement that is stored in the `/boot/System.map` file (0xc024928) and it does not.

```
kdb> id 0xc0109d84 ~ (address of system call interrupt from /boot/System.map)
```

```
system_call + 0x2d: call *0xc31ac000(,%eax,4)
```

This is an indication that the system call table is being redirected by a kernel rootkit

Since the kernel is a fundamental part of the computer operating system we will first examine this aspect of the r.tgz rootkit to determine the method that this rootkit used to compromise the target system. You can not trust any of the system output if the kernel has been compromised.

As previously mentioned, the Honeynet allowed us to retrieve the install scripts and code that is utilized by the r.tgz rootkit. The main install file for the r.tgz rootkit calls a series of script files to install the rootkit. Analysis of these scripts indicates that the startfile script is the script that

compromises the kernel on the target system. The replace script is used to replace the system binaries.

The startfile script copies the r.tgz file *init* to the */etc/rc.d/init.d* directory. Analysis of the r.tgz *init* file indicated that this is the script that compromises the kernel. The *init* script file executes several binary files named *sendmail* (executed as a daemon), *write* (executed as a daemon), and two instances of the executable *all* with the 'i' switch and a pid. The *sendmail* binary file is actually another instance of the *all* program that is copied over in the *createdir* script file that is executed by the r.tgz startup script file. This analysis resulted in the identification of three instances of the *all* binary executable file being executed by the r.tgz rootkit. Two of these instances of the *all* binary file have pid's associated with them. The fact that the *init* file, which calls these three instances of the *all* file, has been copied into the */etc/rc.f/init.d* directory is an indication that the r.tgz rootkit developer wanted this code to be executed upon system reboot, making the kernel compromise portion of this rootkit resident within memory. The use of file names such as *init*, *sendmail*, and *write* are examples of a direct masquerade as described by Thimbleby, Anderson, and Cairns. Direct masquerades are files that pretend to be normal programs [12]. As a result of this analysis, we choose to examine the *all* program. This program is a binary executable file and we do not have the underlying source code that was used to create this rootkit. We choose to use a tool such as *strings* on this file initially in conducting our analysis. An segment of the results of running the *strings* command on the r.tgz *all* binary file are indicated in Figure 6.

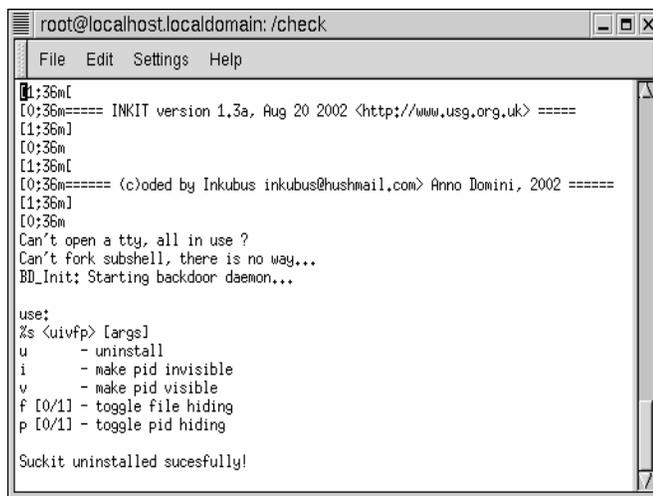


Figure 6: strings output of r.tgz all program

The strings output of this program indicates that the *all* program is a kernel level rootkits known as INKIT. A search on the Internet for a kernel rootkit called INKIT does not result in any references to this particular rootkit. According to the use statement that is output by the *strings* command, the 'i' switch that is used in the *init* script with a particular pid is used to make that pid invisible. The last string displayed in Figure 6 is significant to note. This text

string makes reference to the SuckIT rootkit. We presented an in depth analysis of the SuckIT kernel level rootkit in [6]. The fact that this string appears in INKIT, including the misspelling of successfully just like the actual SuckIT rootkit, leads us to believe that INKIT is a modification or a copy of the SuckIT kernel level rootkit. Next we attempted to uninstall the INKIT kernel level rootkit using the 'u' switch as indicated by the use statement in Figure 7.

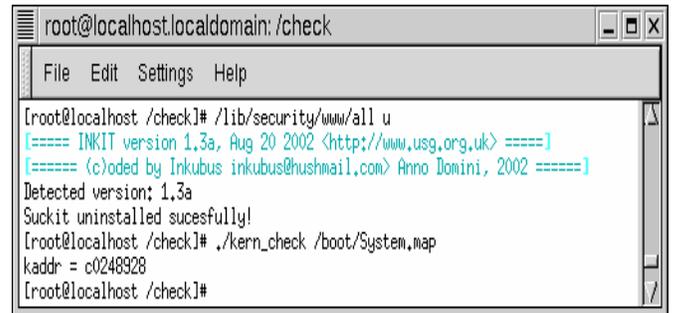


Figure 7: Uninstall of INKIT kernel rootkit

This command was successful in uninstalling the kernel level rootkit. We then verified the integrity of the kernel with the *kern_check* kernel integrity check program. The indication was that there are no system calls currently being redirected by the kernel. An examination of the kernel using *kdb* indicated that the system call interrupt is now referencing the correct system call table. As a final check, we compared the current kernel text segment against the original archived text segment. These files now match. At this point the kernel is no longer compromised and we reran the file integrity checker and known rootkit detection programs on the target system since we can now trust the kernel output. The output from the AIDE file integrity checker program now indicates that 196 files have changed. This is an increase of 19 files from the previous check of the AIDE program where 177 files were detected as being changed by the r.tgz rootkit. It appears that the kernel element of the r.tgz rootkit was hiding these 19 changed programs from the AIDE program. Next in the analysis process we analyzed these changed files and the r.tgz install scripts to determine how these files had been changed. The results of the new instance of the AIDE program are indicated in Figure 8.

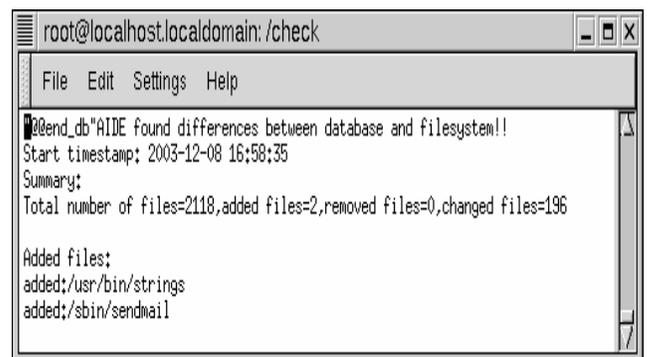


Figure 8: New AIDE results on target system.

Analysis of the install scripts for the r.tgz rootkit does not indicate that 196 files are being changed when the rootkit is installed. Analysis of the AIDE results indicates that all of the executable files in the /bin directory are changing. Comparison of the files in the /bin directory with known good files indicates that the files that are not being changed by the r.tgz rootkit are increasing in size by 8759 bytes. This increase in file size is a signature of the Linux.OSF.8759 virus. This virus is associated with the hax.tgz rootkit [13]. Application of the methodology thus far has indicated that r.tgz is composed of elements of two rootkits; the INKIT kernel level rootkit which is based on SuckIT, and the hax.tgz binary level rootkit.

Another signature of the Linux.OSF.8759 virus is that a trojan port is opened on the target system at 3049. This trojan port is detected by the chkrootkit program which checks for processes listening on ports with the use of the *netstat* command with the *-anp* switch to detect open ports on the system in question. Other Trojan ports can be detected in a similar fashion.

There is a utility named *clean.OSF.8759-ps* that can be used to clean infection of the Linux.OSF.8759 virus [14]. However, these files can not be cleaned by the root user after the r.tgz rootkit is installed on the target system. This is a result of the *socklist* script within the r.tgz rootkit changing the attributes on all of the files in the /bin directory with the *chattr +ASacdisu* command. Attributes on selected files in the /sbin and /usr/bin directories are also reset by the *replace* script within the r.tgz rootkit after these files are infected with the Linux.OSF.8759 virus. Files and directories with their attributes reset could be used as an indication that a possible rootkit is installed on the target system. These attributes must be turned off using the *chattr -ASacdisu* command to disinfect these files.

To produce an accurate count of the number of binary files that are added, deleted, or changed by the r.tgz rootkit the attributes of the files within the /bin, /sbin, and /usr/bin directories must first be reset. Then these directories can be disinfecting with the *clean.OSF.9759-ps* utility. Once these files and directories have been disinfecting the AIDE program is run to provide an accurate count of the files that have been changed on the target system by the r.tgz rootkit. Figure 9 shows the results of this instance of the AIDE program on the target system. The results are that 2 files have been added and 14 files have been changed by r.tgz. These results correspond with the install scripts for the r.tgz rootkit.

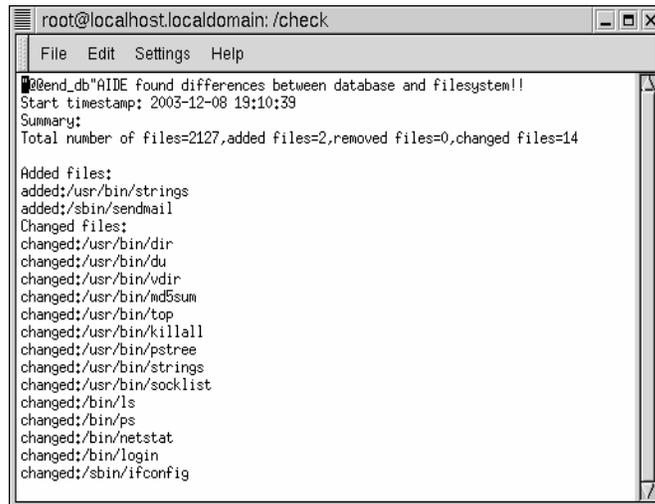


Figure 9: Accurate AIDE count of changed files

IV. ROOTKIT CHARACTERISTICS

The chkrootkit program only detects 5 of the 14 files that are detected as being changed by the AIDE program. It may be possible to develop new signatures for the 9 changed files that are not detected by chkrootkit. These files are: *dir*, *vdir*, *md5sum*, *top*, and *strings* in the /usr/bin directory; *ps*, *netstat*, and *login* in the /bin directory; and *ifconfig* in the /sbin directory. Two other possible signatures are the two added files, /usr/bin/strings and /sbin/sendmail. There are currently two instances of the *strings* command in the /usr/bin directory. The *socklist* script of the r.tgz rootkit copies the original /usr/bin/strings file to the following file: /usr/bin/strings<blank space>. This would make it hard to detect that this file has been tampered with under a visual examination. However, this is a signature that can be used to detect the presence of either the r.tgz rootkit or the binary elements that make up the r.tgz rootkit on the target system.

Using the elements of the methodology to detect unique rootkit string signatures that we have presented in our paper “A Methodology for Detecting New Binary Rootkit Exploits” we are able to detect some possible unique string signatures in the binary files that are replaced by the r.tgz rootkit [14]. The following are potential string signatures that can be used by a program such as chkrootkit to detect the presence of the r.tgz rootkit binary file replacements.

/usr/bin/dir	“stpcpy”
/usr/bin/vdir	“/usr/include/file.h”
/usr/bin/md5sum	“/usr/local/share/locale”
/usr/bin/top	“proc_hackinit”
/usr/bin/strings	“/bin/su -”
/usr/bin/socklist	“bin/egrep -v”
/bin/ps	“/tmp/extfsRNV23Z”
/bin/netstat	“__bzero”
/bin/login	“cococola”

We have already characterized the kernel rootkit that is an

element of r.tgz. This kernel rootkit is one that redirects the system call table to an entirely new system call table as previously described by the authors in [6]. Based on other analysis that we have done, we were able to uninstall and reinstall this rootkit on the target system. It is significant to note that every new reinstallation of this kernel level rootkit will result in a new address in kernel space for the new instance of the compromised system call table.

As previously stated, we concluded that the r.tgz rootkit is a blended rootkit that contains elements of the INKIT kernel rootkit and the hax.tgz binary rootkit. The INKIT rootkit is based on SuckIT. The hax.tgz rootkit is based on bigwar.tgz rootkit [13].

V. SUMMARY

We have applied our methodology to a rootkit that we were able to acquire on the Honeynet research network that we established at Georgia Tech. Our methodology enabled us to identify the binary elements that this rootkit replaced on the target system. New signatures were identified that can help to detect the presence of this rootkit.

The methodology enabled us to characterize the kernel element of this rootkit as a modification to an already existing kernel rootkit. We were able to uninstall this kernel rootkit on the target system.

We have also applied this methodology to nine additional rootkits. This analysis enabled us to characterize these rootkits and identify specific signatures that could be used in subsequent detection and analysis [15].

The analysis that we have presented concerning the application of our methodology to a rootkit would benefit network administrators, researchers, and network security personnel in characterizing rootkits as well as provide methods to detect these rootkits.

VI. REFERENCES

- [1] D. Dettrich, (2002, 5 JAN) “*Root Kits*” and hiding files directories/processes after a break-in, [Online]. Available: <http://staff.washington.edu/dittrich/misc/faqs/rootkits.faq>
- [2] L. Spitzner, *Honeypots- Tracking Hackers*, Indianapolis, IN: Addison-Wesley, 2003, p. 69.
- [3] The Honeynet Project, *Know Your Enemy*, Indianapolis, IN: Addison-Wesley, 2002, p. 19.
- [4] http://www.linuxsecurity.com/feature_stories/feature_story-141.html, Nov 2003.
- [5] <http://www.packetfu.org/hpa.html>, Nov 2003.
- [6] J. Levine, J. Grizzard, H. Owen, “A Methodology to Characterize Kernel Level Rootkit Exploits Involving Redirection of the System Call Table”, to be presented at the 2nd International Information Assurance Workshop, Charlotte, NC, 8-9 Apr 2004.
- [7] <http://la-samhna.de/library/rootkits/detect.html>, Sep 2003

- [8] J. Levine, J. Grizzard, P. Hutto, H. Owen, “A Methodology to Characterize Kernel Level Rootkit Exploits that Overwrite the System Call Table”, to be presented at SoutheastCon 2004, Greensboro NC, 26-28 Mar, 2004.
- [9] <http://www.chkrootkit.org>, Dec 2003.
- [10] Zovi, D., “Kernel Rootkits”, <http://www.cs.unm.edu/~ghandi/lkr.pdf>, 3 July 2001, Oct 2003.
- [11] <http://www.cs.tut.fi/~rammer/aide.html>, Sep 2002.
- [12] Thimbleby, S. Anderson, p. Cairns, “A Framework for Modeling Trojans and Computer Virus Infections,” *The Computer Journal*, vol. 41, no.7 pp. 444-458, 1998.
- [13] <http://www.honeylux.org.lu/project/honeyluxR1/result/sub01/report/hax.html>, Aug 2003.
- [14] <http://packetstormsecurity.nl/trojans/indexdate.shtml/lean-osf.8759.tgz/README>, Dec 2003.
- [15] J. Levine, H. Owen, B. Culver, “A Methodology for Detecting New Binary Rootkit Exploits”, presented at the 2003 IEEE SoutheastCon 2003, Ocho Rios, Jamaica, 4-6 Apr 2003.
- [16] J. Levine, A Methodology for Detecting and Classifying Rootkit Exploits, PhD Thesis, Georgia Institute of Technology, Atlanta, GA, to be published.