A Methodology for Detecting New Binary Rootkit Exploits

John Levine*, Brian Culver**, Henry Owen*,
*School of Electrical and Computer Engineering
**Office of Information Technology
Georgia Institute of Technology
Atlanta, Georgia 30332-0250

***Abstract*—Hackers who gain root privilege on a computer system usually want to maintain this level of privilege for future exploits. They do not want to have to go through the steps to regain this level of privilege because of the effort involved and the increased risk of being discovered as well as the possibility that the original exploit used to gain root access gets patched. A hacker who gains access to a system will install some method for use at a later time allowing the hacker to come back onto the system with root privilege. The hacker accomplishes this by the installation of some type of software known as a backdoor or a Trojan. One type of Trojan is known as a rootkit, in which specific system binaries necessary for the day-to-day operation of the computer system are modified or replaced by the hacker. These binaries still maintain their original functionality while allowing the hacker to maintain the ability to operate clandestinely on the host system. We propose a methodology for determining the unique signatures of common rootkits and how to determine if a compromised system is infected with a new unknown or modified previously known rootkit.**

***Index Terms*— Computer crime, hacking, information assurance, rootkits, signature analysis, Trojan.**

I. Introduction

Computers on today's Internet are vulnerable to a variety of exploits that can compromise their intended operations. Systems can be subject to Denial of Service Attacks that prevents other computers from connecting to them for their provided service (e.g. web server) or prevent them from connecting to other computers on the Internet. They can be subject to attacks that cause them to cease operations either temporary or permanently. A hacker may be able to compromise a system and gain root access, i.e. the ability to control that system as if the hacker was the system administrator. A hacker who gains root access on a computer system may want to maintain that access for the foreseeable future. One way for the hacker to do this is by the use of a rootkit. A rootkit enables the hacker to access the compromised computer system at a later time with root level privileges. System administrators have a continuing need for techniques in order to determine if a hacker has installed a rootkit on their systems.

Techniques currently exist for a system administrator to monitor the status of systems. Intrusion detection systems operate at numerous levels throughout the network to detect malicious activity by hackers. At the system or host level, a file integrity checker program can be run on the computer system in question. There are several host based IDS tools that look at changes to the system files. These programs take a snapshot of the trusted file system state and use this snapshot as a basis for future scans. The system administrator must tune this system so that only relative files are considered in the snapshot. Two such candidate systems are TRIPWIRE and AIDE (Advanced Intrusion Detection Environment) [1]. AIDE is a General Public License (GPL) program that is available for free on the Internet. This program operates by creating a database of specified files. This database contains attributes such as: permissions, inode number, user, group, file size, creation time (ctime), modification time (mtime), access time (atime), growing

size and number of links [2]. However, a program like AIDE does have shortcomings. Rami Lehti, in the AIDE manual, states "Unfortunately, Aide can not provide absolute sureness about changes in files. Like any other system files, Aide's binary files and/or database can be altered."

There is another free program that checks a system for rootkit detection. This program is known as chkrootkit and is available at http://www.chkrootkit.org. This program runs a shell script that checks specific system binaries to determine if a rootkit has been installed on the system. This program also checks to see if the network interfaces on the computer have been set to the promiscuous mode, which is a common ploy used by hackers in order to capture network traffic. The program also checks the system logs [3]. The shell script is signature based, therefore the signature must be known in order to detect if a rootkit has been installed on a system. Programs such as chkrootkit may not detect new rootkits, as well as modifications to existing rootkits.

In this paper we discuss a methodology for determining if a system has been infected by an existing rootkit or if the system has been infected with a new or modification to an existing rootkit. New signatures can be created for these new or modified rootkits in order to detect them.

II. EXISTING METHODOLOGIES TO DETECT ROOTKITS

A. What is a RootKit?

A rootkit can be considered as a "Trojan Horse" introduced into a computer operating system. According to Thimbleby, Anderson, and Cairns, there are four categories of trojans. They are: *direct masquerades*, i.e. pretending to be normal programs; *simple masquerades*, i.e. not masquerading as existing programs but masquerading as possible programs that are other than what they really are; *slip masquerades*, i.e. programs with names approximating existing names; and *environmental masquerades*, i.e. already running programs not easily identified by the user [4]. We are primarily interested in the first category of Trojans, that of direct masquerades.

A hacker must already have root level access on a computer system before he can install a rootkit. Rootkits do not allow an attacker to gain access to a system. Instead, they enable the attacker to get back into the system with root level permissions [5]. Once a hacker has gained root level access on a system, a trojan program that can masquerade as an existing system function can then be installed on the compromised computer system.

Rootkits are a fairly recent phenomenon. Systems used to have utilities that could be trusted to provide a system administrator with accurate information. Modern hackers have developed methods to conceal their activities and programs to assist in this concealment [6]. Traditional RootKits alter or replace existing system binary components. These replaced or modified programs allow backdoor access to a system as well as the ability to hide the hacker's presence on the system [7]. Rootkits are a serious threat to the security of a networked computer system.

A skilled hacker with programming experience most likely has the ability to create a rootkit for a Linux type system. It is very easy to create a rootkit. First you need a sniffer program. A sniffer program can be fashioned from a program like tcpdump. This program will be used for password recording after placing the Ethernet connection in promiscuous mode. Next you need the source code for the standard system binaries [8]. A skilled hacker can modify the source code to include a backdoor and compile a trojan binary. Even for a hacker without the requisite programming ability, there exists numerous rootkits targeted for specific operating systems available on the Internet today.

The vulnerabilities that exist in modern operating systems as well the proliferation of exploits that allow hackers to gain root access on networked computer systems provide hackers with the ability to install rootkits System administrators need to be aware of the threats that their computers face from rootkits as well as the ability to recognize if a rootkit has been installed on their computer system.

B. Running AIDE on a computer system.

We decided that some form of host based intrusion detection was to be used in our methodology to detect rootkits. Running a file integrity checker such as TRIPWIRE or AIDE when a system is initially built in order to get a file integrity baseline is highly recommended [9]. We chose AIDE over TRIPWIRE for several reasons. According to the AIDE documentation, AIDE is written as a replacement and extension for TRIPWIRE. It includes more features and is not a closed product. It can utilize multiple integrity checking algorithms and has the ability to output the database to stdout or a file. The current version of AIDE, as well as previous versions, is available on the Internet [10]. There is, however, an open source version of TRIPWIRE available for download. We felt that either program would suit our methodology accordingly.

We have chosen Linux Red Hat version 6.2 as our operating system. Red Hat 6.2 is a stable operating system that has been available for a number of years. We chose a workstation installation with all available packages installed on the system. There are known exploits available for Red Hat version 6.2 that will allow a hacker to gain root access on an unpatched system. This paper will not address how such a hacker would gain root access only how a hacker might keep such access after gaining it.

We set out to install the file integrity checker AIDE. To install AIDE it is necessary to first install the mhash library. The mhash library is necessary in order to run the additional integrity checking algorithms. The mhash library is available on the Internet [11]. Once installed, it is now possible to install the AIDE program.

A customized aide.conf file is necessary in order to tell the AIDE program what characteristics need to be checked

on the specific files you want to fingerprint. Also one must specify in this file which directories contain the files that will be tracked for changes. We are only checking for changes to md5 and sha1 checksums on the system binaries in the /bin directory because this is where some of the most critical binary rootkit components are usually installed. It is significant to note that some other rootkits may install trojaned binaries into different directories. There are numerous other file parameters that AIDE can check, including several different checksums. A listing of the aide.conf file that we are using is given in Figure 1.

Figure 1 – the aide.conf file

The first time that the AIDE program is run with the –init switch as indicated in the AIDE manual the file aide.db.new is created in the directory where the aide.conf is located, in our case this is the /usr/local/etc directory. This database file contains the fingerprint of all of the parameters for the files that were selected in the aide.conf file. This file needs to be renamed to aide.db so that subsequent executions of the AIDE program can use it.

The AIDE manual also recommends that the configuration file (aide.conf), the AIDE binary (aide), and the database file be maintained in a secure location such as a read-only media. This is to prevent a hacker from altering any of these files in order to present false information to the system administrator.

C. The Georgia Institute of Technology Office of Information Technology (OIT) Methodology for Detecting Rootkit Exploits.

The Georgia Institute of Technology, or Georgia Tech is an engineering and research institutes in the United States [12]. There are over 15,000 undergraduate and graduate students enrolled at the university as well as approximately 5,000 staff and faculty. Undergraduate and graduate degrees are offered in the Colleges of Architecture, Engineering, Sciences, Computing, Management, and the Ivan Allen College of Liberal Arts.

The Georgia Tech Office of Information Technology has the primary mission of providing technology leadership and support to Georgia Tech students, educators, researchers, administrators, and staff. OIT consists of seven directorates including the Information Security Directorate [13].

The Information Security Directorate is responsible for numerous tasks including: educating the campus community about security related issues, assessing current policies and developing new policies, assisting in strengthening technical measures to protect campus resources, and developing mechanisms to react to incidents and events that endanger the Institute's information assets [14]. There are 69 separate departments at Georgia Tech with between 30,000-35,000 networked computers installed on campus. The campus has two OC-12's and one OC-48 connection to the Internet with an average throughput of 600Mbps. Over four terabytes of data are processed by Georgia Tech on a daily basis.

Because of the high data throughput as well as the requirement for academic freedom and the research requirements of the various departments, the Information Security Directorate does not run a firewall at the Internet connection to the campus. However, individual departments and campus agencies do run firewalls designed to meet their security requirements.

The Information Security Directorate does at present run an Intrusion Detection System (IDS) at the campus gateway in order to monitor possible exploits against campus computer systems. This monitoring is done out of band and suspicious traffic is not terminated when detected. Suspicious activity will undergo a follow-on investigation.

Prior to installing the IDS the Information Security Directorate would on average investigate 5 possible compromises a week. The Information Security Directorate would normally receive reports of these compromises from concerned computer users. Since installing the IDS, the Information Security Directorate on average investigates 5 compromises a day.

When a system that may have been compromised is identified, the administrator responsible for that computer is notified that the computer system is to be investigated. This is to prevent the computer from being tampered with which could result in the investigation being impeded.

The system will be booted with a known good media disk and the hard drive will be mounted in a read-only manner. The use of a know good media disk is also recommended by Chris Kuethe in his paper on detecting hacked systems [15]. A duplicate copy of the hard disk may be produced with a signature checksum in the event that the hard disk ends up being used in a criminal investigation. The techniques used by the Security Directorate personnel at Georgia Tech may be unique to Georgia Tech. The current state of the art in Computer Forensics Analysis does not provide a formal methodology for investigation [16].

The investigation will begin by examining various directories that a hacker may have manipulated to hide his exploits on the computer system in question. The log files will first be examined to see if there are any records of what was done on the system. It is not uncommon for the log files to have no record of system modifications or to be deleted from the system. If the log files are deleted, steps will be

taken to try to retrieve them. Next, previously known directories where a hacker may choose to hide exploit files will then be examined. The chkrootkit tool may be run to check if a rootkit has been installed on the system. If these checks prove unsuccessful the Security Directorate Personnel will then conduct a more detailed examination of the system. For example, on a UNIX or LINUX operating system commands such as *find* or *locate* will be used to try and find directories that may have been used by the hacker when the system was exploited.

If such a directory is located then a listing of that directory will occur to see what files are present in that directory. The *file* and *strings* command will be used on these files to examine them. The *file* command will be run in order to try and determine the file type. The output of the *strings* command will be read in order to try and recognize any suspicious text strings that may indicate what exploit was done to the computer.

The /proc/ directory will then be checked to see if a program is running in memory. The pid's (process id numbers) will be compared between those listed by the *ps* (report process status) command (using the –ef switch) and those listed in the /proc/ directory. A difference between these two listings indicates that the *ps* command was most likely modified by the hacker to hide the processes that the hacker has running on the computer. The Information Security Directorate Personnel uses the /proc/ directory as a true listing of what is currently running on the system being investigated. The processes that show up in the /proc/ directory but which are not listed by the *ps* command will be examined using the *file* and *strings* command.

The *strace* command may be used to trace system calls for suspicious programs binaries left on the system by the hacker. The *ldd* (print shared library dependencies) command may also be used to check on shared library dependencies of the suspicious programs, especially for those suspect programs that have the same name as known good system binaries. A difference in library listing is determined to be a direct indication that hacked version of system binaries are installed on the system.

A similar methodology is used for other operating systems in order to determine if the system has been exploited by a rootkit. We believe it is the case that in general, information security personnel have no formal methodology to determine if a computer has been infected with a new unknown or previously modified known rootkit without conducting an exhaustive manual investigation of the exploit.

III. A METHODOLOGY TO DETECT NEW ROOTKIT SIGNATURES

A. Comparison of source code

We utilized lrk4 to test our methodology. The Linux RootKit IV (lrk4) was released in November of 1998 by Lord Somer. It includes the usual rootkit components to include: a sniffer, utilities to edit and erase log files, and Trojan replacement system utility programs [17]. More recent versions of the lrk rootkit exist. The source code for Version 5 is also available on the Internet in addition to lrk4 source code for systems with and without shadow passwords. There is also a precompiled version of lrk4 that is available for downloading [18]. The lrk4 code continues to be modified and improved upon. An update to lrk4 was posted on the Internet as recently as 11 May 2000 [19].

Although newer versions of the lrk exploit exist (ver 5 & 6) lrk4 is recognized as the most stable version of the lrk exploit. In order to use the precompiled version of the lrk4 exploit, it is necessary to install the previous version of several libraries since the compiled version of lrk4 was built against these earlier libraries [20].

A comparison can be made between the source code files of the clean and lrk4 version of the login.c file. Lord Somer had to add to the original login.c program from the Shadow-Suite in order to allow for Trojan password access and the disabling of the logging function. A comparison can by made either manually or by using the *diff* utility that is available on the Linux system. The following 2 figures show the result of this comparison on the two files in question.

Figure 2 – *diff* output screen 1

Figure 3 – *diff* output screen –2

Theses screens show all of the code that was input by Lord Somer into the login.c source code. Much of the logic behind this code has already been addressed in a separate paper. However, it is often the case that the source code for an exploit is no longer available on the target system. Therefore it is necessary to find another method to recognize that a specific rootkit has been installed on a system.

B. Comparison of binary files

We propose a methodology to uniquely identify the different binary level rootkits. It is necessary to have a clean copy of each binary file that was replaced by the rootkit program. The listing of the files that were replaced would be available as a result of running AIDE on the target system. Copies of the infected binary files are available on the target system. For example, on our target Red Hat 6.2 system, the clean login binary exists in the bin_bu directory, which was created when the operating system was first installed. The infected login program currently exists in the /bin directory as indicated by the AIDE program (see figure 3). This methodology to identify unique rootkits is as follows:

1. Run the *strings* command on each file in question and pipe the results into a file for further comparison.
2. As an additional check run the *diff* command against these two files for a check to see if the strings contained in the two files are different. Use the –q switch so that the output only reflects if the files are different.
3. Run the following command:

   fgrep –v –f login.clean login.infected

   The fgrep command outputs a line-matching pattern. The -v switch is an invert-matching switch which tells the fgrep function to only output those lines that do not match. The –f switch tells the fgrep command to get the patterns to use for matching in the second file (login.infected) from the first file (login.clean).

The following screen shows this series of commands being executed.

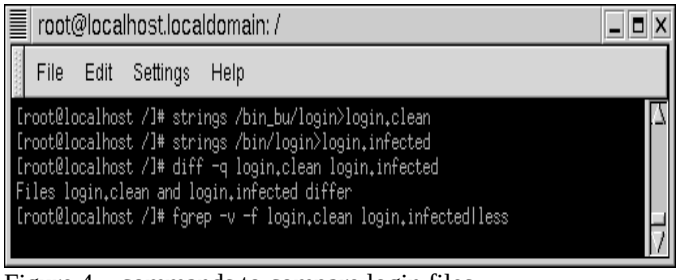

Figure 4 – commands to compare login files

This series of commands compares the any strings that exists in both files and outputs only those existing strings that are different between the two files. Using the clean login file as the string source and the infected login file as the target file will result in the output of those strings that exist in the hacked version of the login program but do not exist in the clean login file. The output of this command is displayed in the following screen with the string 'root' highlighted.

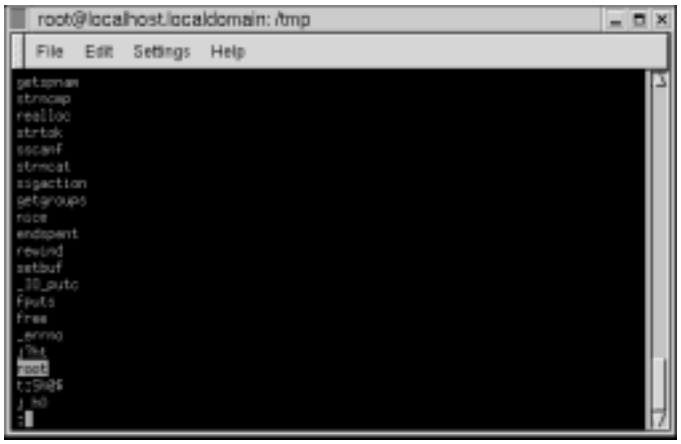

Figure 5 – output of *fgrep* function

There are numerous strings that differ between the clean and hacked login file. The primary reason for this is that the hacked login program is based on the Shadow-Suite login.c code and the clean login program is based on the BSD login.c code. However, some of these strings are potential signatures for the chkrootkit program to use to check for the existence of an infected rootkit program. The fact that we had the clean login.c code from the Shadow-Suite made it easy to determine what code, to include what strings, had been added by Lord Somer. All that is then required is to check these added strings against the original clean login.c binary file to ascertain the validity of using these strings as a signature.

By using our methodology a system administrator could build a library of files that may contain the test strings that exist in the system binaries. Even if these files do not contain any unique strings they can still serve as a unique signature for a specific rootkit. As various rootkits are discovered additional unique binary files can be added to the library. A system administrator that determined that a binary rootkit had been installed on a system could follow our methodology to compare the infected system binaries with the files that exist in the library.

Thus, if an infected binary did not match with the existing binaries in the library, the system administrator could make the determination that the system has been infected with a new or modified rootkit since it does not match any of the existing signature files.

The text strings that exist in this new or modified rootkit can be examined for unique strings to identify this new trojan exploit. This unique string could be used by the chkrootkit program to identify this specific rootkit exploit. A common text string could also be sought so that the chkrootkit program would be able to detect the greatest number of exploits with the least number of signatures. In either case, this signature can also be provided to a signature–based IDS system for detection of this exploit.

C. Modifications made to the chkrootkit program

We initially installed the Lord Somer's lrk4 rootkit on a clean Red Hat 6.2 system. We then ran chkrootkit-0.36, which was the current available version of chkrootkit, against the system. This version of chkrootkit detected that some of the binaries had been infected, but it did not detect that the login binary had been infected. The lrk4 rootkit that we installed did contain a source login.c program with a trojan capability as previously discussed in the last section.

Upon analysis, we discovered an error in the logic of the chkrootkit program. The chkrootkit suite is a script called chkrootkit that calls a routine called chk_login. This routine performs signature analysis on the login program by looking for the appearance of various strings within the binary file. One of the strings used by the chkrootkit program to detect infected login programs is the string "root". The lrk4 login binary file has 2 instances of the string "root" within it. The clean login program does not contain any reference to the string 'root'. The chk_login routine was written to allow for the appearance of 2 or less instances of "root" in the login binary program. We contacted Nelson Murilo, who is one of the authors of the chkrootkit program, about our discovery. The chkrootkit code was modified to only allow for the appearance of the string "root" in the login file for those specific operating systems that have the string "root" appear in the clean version of their login files. A new version of the chkrootkit program, chkrootkit-0.37, was quickly released that now detects that the lrk4 login file is infected.

IV. Conclusion

System administrators have a continuing need for tools, techniques and procedures in order to determine if their computer systems have been compromised. Various tools exist to help a system administrator make this determination on a daily basis. We propose a methodology in this paper for the system administrator to not only be able to determine a computer system has been infected with a binary level rootkit, which he can currently accomplish with existing tools, but also to be able to determine if the rootkit is a variant of already established exploits or a totally new exploit.

We examined the current methodology for detecting rootkit exploits by defining the characteristics of a rootkit. We addressed the employment of a current tool to detect rootkits. We also addressed the current methodology used to detect rootkit exploits utilized at a major public research university in the Southeastern United States

The methodology we developed to identify new or modified rootkit exploits was then outlined. These techniques will not only assist the system administrator in identifying new rootkit exploits but may also provide string signatures for IDS's and the chkrootkit program to use in the detection of rootkit exploits.

References

[1] S. Northcut, L. Zeltser, S. Winters, K. Kent Fredericks, R. Ritchey, *Inside Network Perimeter Security.* Indianapolis, In: New Riders, 2003, pp. 283-286.

[2] R. Lehti , " The Aide Manual", www.cs.tut.fi ~rammer /aide /manual.html , SEP 2002

[3] N. Murilo, K. Steding-Jones, "chkrootkit V. 0.36" www.chkrootkit .org/README .

[4] H. 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.

[5] E. Cole, *Hackers Beware,* Indianapolis, In: New Riders, 2002, pp. 548-553.

[6] 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

[7] E. Skoudis, *Counter Hack,* Upper Saddle River , NJ: Prentice Hall PTR: 2002, pp. 422-430.

[8] O'Brian, D. Recognizing and Recovering from Rootkit Attacks. Sys Admin 5,11 (Nov 1996), pp. 8-20.

[9] S. Northcutt, J. Novak, *Network Intrusion Detection An Analyst's Handbook* . Indianapolis, New Riders, 2001, p. 207.

[10] http://www.cs.tut.fi/~rammer/aide.html.

[11] http://mhash.sourceforge.net/

[12] http://www.gatech.edu SEP 2002

[13] http://www.oit..gatech.edu, SEP 2002.

[14] http://security.gatech.edu, SEP 2002

[15] C. Kuethe, "Through the Looking Glass: Finding Evidence of Your Cracker" *The Linux Gazette,* issue 36, Jan 1999, available at: http://www.linuxgazette.com/ issue36/kuenthe.html, SEP 2002 .

[16] R. Di Pietro, L. Mancini, "A Methodology of Computer Forensics Analysis" presented at the 2002 IEEE Workshop on Information Assurance, West Point, NY, 17-20 June 2002.

[17] S. Hawkins, "Understanding the Attackers Toolkit" *The SANS Institute Reading Room,* January 13, 2001, available at: http://rr.sans.org/linux/toolkit.php. AUG 2002.

[18] http://packetstormsecurity.org/UNIX/penetration/rootkits, AUG 2002

[19] *http://packetstormsecurity.nl/UNIX/penetration/rootkits/lrk-4.1.tar.gz*, SEP 2002

[20] E. Skoudis, *The Hack-Counter Hack Training Course,* Upper Saddle River , NJ: Hall PTR: 2002, p 74.