Chapter 1

TIMELY ROOTKIT DETECTION DURING LIVE RESPONSE

Daniel Molina, Matthew A. Zimmermann, Gregory R. Roberts, Marnita T. Eaddie, Gilbert L. Peterson

Abstract The ever evolving nature of the cyber domain presents a unique set of challenges for today's forensic analysts. One such challenge comes in the form of programs called rootkits. These programs attempt to provide stealth to an attacker by manipulating a computer's operating system. The surreptitious environment these programs create increases the level of difficulty and the time that it takes to perform a computer forensic investigation.

Current forensic utilities provide limited rootkit detection capabilities. In many cases, an analysts cannot determine the presence of a rootkit on a system until after a detailed review of all the collected evidence has been done. This research effort seeks to shorten this time by providing a simple tool that can identify the presence of a rootkit, during a live response investigation, while minimizing destruction of evidence. This tool provides similar results to those identified by Todd, et al.[1] for live response investigations. Their research pointed out that, at the time, there were no tools that performed timely rootkit detection without sacrificing evidence integrity.

Keywords: Forensic analysis, rootkits, rootkit detection, live response.

1. Introduction

The forensic community has always been concerned with preserving evidence while avoiding crime scene contamination. The ever changing nature of computer systems makes this a even big challenge for forensic investigators. Adding to this is the fact that there are programs that aid intruders in hiding their tracks on a system. These programs, known as rootkits, increase the level of difficulty in a forensic investigation in two ways. First, they prevent certain forensic tools from gathering necessary

evidence; secondly, they force investigators to run more intrusive tools that can alter the current state of the system.

This paper describes a methodology for accurately identifying a rootkit while minimizing system modification. Studying current rootkits and their hiding techniques reveals the type of rootkits that are detectable using a collection of system information gathering tools in user space and an output correlation algorithm that compares different outputs and flags discrepancies. The automated analysis tool was tested against five different rootkits; they were hiding a backdoor's process identifier (PID) and a folder containing four files. The rootkits under test were: Hacker Defender[16], AFX[19], Vanquish[17], FU[18], and FUto[20]. The backdoor under test was Back Orifice 2000[21].

The first portion of this paper provides background information categorizing rootkits and describing hiding, and detection techniques. The latter part of the paper explains the detection approach, tool developed, test methodology, and test results.

2. Rootkit Taxonomy

A rootkit is a program that allows an attacker unhindered and undetected access to a computer. The term 'root' comes from the UNIX world where root is the highest level of privilege a user can have. Originally written for UNIX, rootkits have expanded their reach and are now written for other operating systems. Rootkits are frequently a collection of "commonly Trojaned system processes and scripts that automate many of the actions attackers take when they compromise a system."[9] Rootkits can hide files, network connections, memory addresses, and registry entries. Rootkits can be embedded in other programs or media similar to the rootkit found in Sony's CDs in 2005[5].

In Windows OSs, as in UNIX, rootkits seek the highest privilege level possible. Windows runs on the Intel x86 architecture, which employs a memory protection scheme represented using four rings. The layers are labeled from zero to three; ring 0 has the highest level of privilege while ring 3 has the lowest. In addition, ring 0 represents the memory space where the operating system kernel and drivers reside, and ring 3 represents the memory space where user applications reside. In order for a rootkit to achieve a high level of stealth, the malicious program must operate at a lower layer (i.e. ring 0) than where rootkit detection or prevention software operates (i.e. ring 3).

In today's systems, the lower layers control the upper layers. Rootkit researchers such as Hoglund[8] and Rutkowska[15] have pointed out that if a rootkit detector is at a lower layer than a rootkit the detection rate

is much higher. On the other hand, if the rootkit executes at a lower layer than the detector the rootkit's ability to hide improves dramatically. By going to a lower layer the rootkit can control the information gathered by the detector which in turn gives the rootkit the ability to hide malicious data. This research addresses the problem by correlating the output of multiple system information gathering utilities to determine if there is enough evidence to expose a rootkit's presence. The utilities are executed from user space without prior installation.

2.1 Rootkit Categories

Rootkits fall into one or more of the following categories: kernel, library, user-level, hardware, and virtualized rootkits. For example, some rootkits have user-level and kernel-level components. This section explains the different categories in more detail.

- Kernel level rootkits add additional code and/or replace a portion of kernel code with modified code to allow stealthy control of the system.
- Library rootkits modify system libraries used by user or kernel applications to achieve stealth[12].
- Application level rootkits (user-level) are programs that alter system files or binaries on disk[14].
- Hardware level rootkits attempt to subvert the system from the lowest level possible. Although they are very difficult to implement, the possibility exists as demonstrated by John Heasman [7]. Our research does not attempt to detect this type of rootkit as it is very difficult to implement and there are no known and available hardware-based rootkits at the time.
- Virtual machine based rootkits(VMBR) attempt to take control of the virtual machine monitor (VMM) which sits between an OS and hardware. The rootkit thereby controls requests to hardware from the upper level. VMBRs modify the boot sequence and load themselves up instead of the chosen virtual machine monitor or operating system. After the program is loaded into memory, the rootkit loads the host operating system as a virtual machine.

  A VMBR is difficult to detect during live analysis because forensic tools are executed from within a VM. In addition, it is difficult to access the VMBR's state by software running on the target machine[3]. From the user's perspective, once the VMBR is running,

the system is in a hidden VMM where malware can exist without interference. The VMBR becomes the master controller of the system with the ability to view/know all keystrokes, network packets, memory allocations, system events, etc. An example of a VMBR is SubVirt[3]. The tool developed here does not try to detect this type of rootkit due to its complexity and lack of availability of a known VMBR.

2.2 Rootkit Hiding Techniques

Kernel and user level rootkits apply a number of hiding techniques to achieve their goals, used individually or in combination. Some of these techniques are:

- *Patching* - This refers to any modification to a binary done statically or dynamically. Static patching is also used by software crackers to get around software protection and registration methods.

- *Hooking* - Hooking is a method used to redirect, or alter, a program's normal flow of execution. This can be done by modifying a function call in memory. An example of a rootkit that uses this method is Hacker Defender.

- *Direct Kernel Object Manipulation* - This method takes advantage of the way Windows schedules processes. It tries to hide processes by de-linking the malicious process from the doubly linked list used by the object manager[8]. Examples of rootkits that use this technique are FU and FUto.

2.3 Rootkit Detection

Rootkit detectors fall under one or more of the following categories [1].

1 Signature-based: system files are scanned to look for a rootkit fingerprint

2 Heuristic/behavioral based: checks for deviations from normal system behavior

3 Cross-view based: comparing system parameters in at least 2 ways

4 Integrity-based: current snapshot is compared to a known trusted snapshot

5 Hardware-based: hardware detection mechanism that employs direct memory access to acquire memory and scan retrieved data looking for rootkit fingerprint.

At the time of this writing, software-based rootkit detectors have components that execute from user space, kernel space, or from both. The best detection capability exist if the detector runs below the rootkit. For example, if the rootkit only executes in user space then the detector has a better chance of detecting the rootkit from kernel space.

Kernel level rootkits are harder to identify for two reasons; the detector must coexist with the rootkit in kernel space, or it must be hardware-based. These approaches are forbidden during live response due to evidence integrity issues.

Some anti-virus programs include rootkit detection features. For example, "F-secure Internet Security 2005 has a feature called 'Manipulation Control'. It is a behavioral blocking mechanism that prevents malicious processes from manipulating other processes [6].

Some examples of Windows-based rootkit detectors are:

- Blacklight by F-Secure: "detects files, folders and processes but not hidden registry keys"[2]. It offers a removal option for detected rootkits. This feature, if used, must be executed cautiously to avoid problems with the system.
- RootkitRevealer, detects rootkits by doing a high level scan, from user space, a raw disk scan, and then comparing the results. For instance, it reports differences in the Windows registry and file system by scanning the registry and file system from user and kernel levels. It does not have any rootkit removal capabilities[2].
- IceSword is a suite of tools that includes a process viewer, startup analyzer, port enumerator, and other tools. It does not identify the rootkit but leaves task of identification up to user.[2]

Although these are effective rootkit detectors, their main forensic drawback is that most require installation.

UNIX-based rootkit detectors include the following:

- Chkrootkit: a "shell script that checks specific system binaries to determine if a rootkit has been installed on the system."[11]
- Rkhunter (rootkit hunter) created by Michael Boelen does MD5 hash comparisons of critical system files, looks for known rootkit files, checks file permissions, looks for suspicious information in loadable kernel modules, tries to find hidden files, and scans plain text and binary files for specific strings[13].

3. Rootkits and Live Response Analysis

During a live response investigation, analysts must be aware of how user-level and kernel-level rootkits affect the integrity of the system. Rootkits running at user-level break the integrity of the system by altering the security subsystem and displaying inaccurate information. "They intercept system calls and filter output application programming interfaces (APIs) to hide processes, files, system drivers, network ports, registry keys and paths, and system services"[4]. Rootkits running at kernel level, can usurp systems calls, hide processes, hide registry keys, hide files, and redirect calls to Trojan functions[10]. For this reason, as part of their analysis, some rootkit detectors check the integrity of critical OS data structures. Unfortunately, this is a more disruptive task than what forensic analysts are typically willing to risk in order to get useful evidence.

Due to the increasing threat of rootkit technology and its potential impact on computer systems, computer forensics analysis must include methods for detecting rootkits in a timely manner. Generally, forensic investigators employ the following methods for rootkit detection: live analysis tools (i.e. FRED, HELIX), install rootkit detection programs, or imaging the hard drive for off-line analysis.[1]

Volatile forensic evidence can be destroyed if a computer is turned off. For live response, information must be carefully collected and documented by the investigators. However, even if the information is diligently collected, it may be compromised by active rootkits. The integrity of live response data is therefore directly impacted by rootkits.

Rootkit detection tools can aid the investigator in determining what rootkits are present and what may have been affected but need to be preinstallation or installed after the system has been compromised altering system state dramatically. Even if data is hidden by a rootkit, live response analysis can either directly find the rootkit or detect suspicious behavior, letting the investigator know that the rootkit is running.

Once the system is turned off, the rootkit cannot actively hide itself or other information. An analysis of the image off-line may reveal a rootkits presence, especially when examining file signatures; however, even if a rootkit is detected, an examiner may still have difficulty identifying what the rootkit was hiding or if the rootkit was even running. The methodology presented in the next section assists the examiner with this task.

4. Automated Analysis

This paper focuses on windows rootkits, and uses available open source utilities that require no installation to perform a system scan from user space. These utilities are executed via a batch script that runs each utility and sends their output to individual files. After the batch script completes, the output files are parsed by an automated analysis program that performs the correlation and identification of discrepancies.

While rootkit detection is possible from user-level by identifying discrepancies in the output of command-line utilities, it is advantageous to automate this process. To help with the automation, a Java-based Graphical User Interface (GUI) with a file parser is used to generate a report of the discrepancies. Because the examination utilities are executed from trusted media, it is desirable that the GUI reporting and parser utility also run from trusted media. The reporting and parsing utility is able to run from a trusted source by using a Java Runtime Environment (JRE) located on the trusted media source.

In order to simplify an examiner's task in detecting rootkits, the rootkit detector provides the capability of initiating a batch job that invokes all the command-line utilities and scans the output files for potential threats. This interface demonstrates that the proposed methodology for detecting rootkits using only the combined output of user level utilities is easily automated and is a viable solution to the problem of locating rootkits in a timely manner in an environment with restricted privileges.

In order to examine the command-line output for the presence of rootkits, the utility focuses on two primary methods of identifying potential threats: differences between output files that should display identical information and combinations of these discrepancies that indicate possible threats. While it is simple to classify all of the differences between the outputs, it is difficult to categorize and assess all possible combinations of these discrepancies according to their potential threat to a system. Therefore, the utility is restricted to observing only a select number of key combinations of differences.

To identify the differences, the application scans all output files for usable system information and correlates that information with the output of other utilities that should generate the same data. A conceptual diagram is shown in Figure 1. The diagram represents a scenario where there is a hidden file called "secret.hide", the goal is to identify its presence on the system by looking at the output of two directory listings that are generated using two different tools (i.e. dir and ls). For example, the

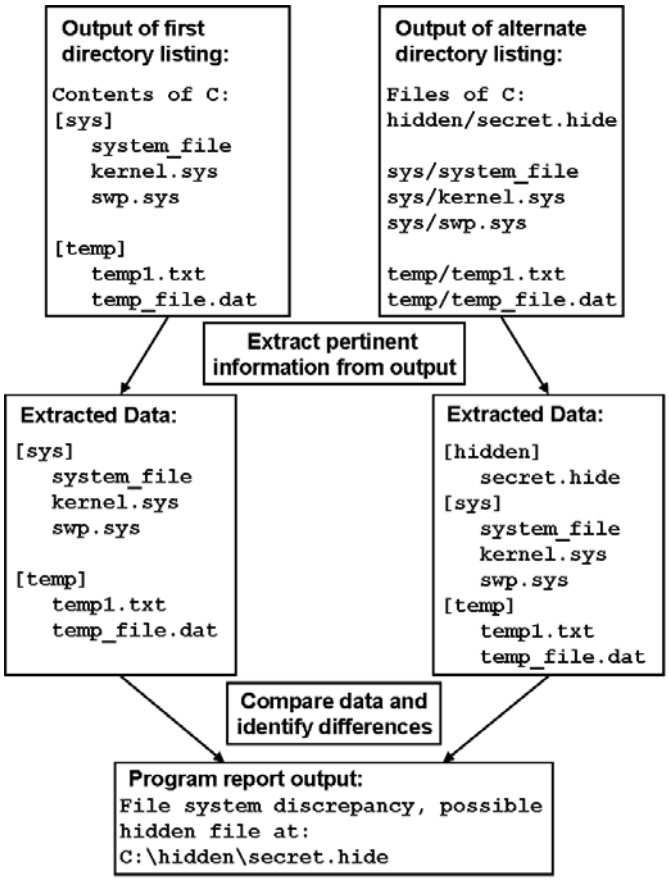

Figure 1. Discrepancy Identification Example

rootkit detector uses this method to discover files that Vanquish tries to hide.

In order to achieve proper parsing of the output files, the parser uses a strictly-formatted parsing scheme to ensure that the proper information is extracted. This type of scheme assumes that the formatting of the output files is constant (i.e. one utility always generates a consistently-formatted output file), this implies that any future additions or version changes to the command-line utilities require a reexamination of the output format and possible modification of the parser code.

Once all the discrepancies are identified by the parsing function, the utility then considers some combination detection rules to determine if any of the detected differences indicate the presence of a rootkit, Figure 2 depicts this step in the overall detection process. These detection

rules are not all encompassing; rather they serve as a demonstration that rootkits can be detected when they fail to completely hide themselves from system users. Also, by considering the meaning of various combinations of results indicated by the command-line utilities rootkits are detectable from user space with minimal system impact.

The diagram depicted in Figure 2 presents a scenario where three discrepancies are being targeted with the goals of determining if they are related to one another and if they indicate the presence of a rootkit. In this scenario, the three pieces of information come from earlier comparisons of different outputs. For example, Discrepancy A comes from a comparison between dir and ls.exe, Discrepancy B comes comparing the outputs of pslist.exe and handle.exe, and Discrepancy C comes from comparing dir with handle.exe. In the end, all this information goes through a set of predefined rules which determine if a rootkit is present or not.

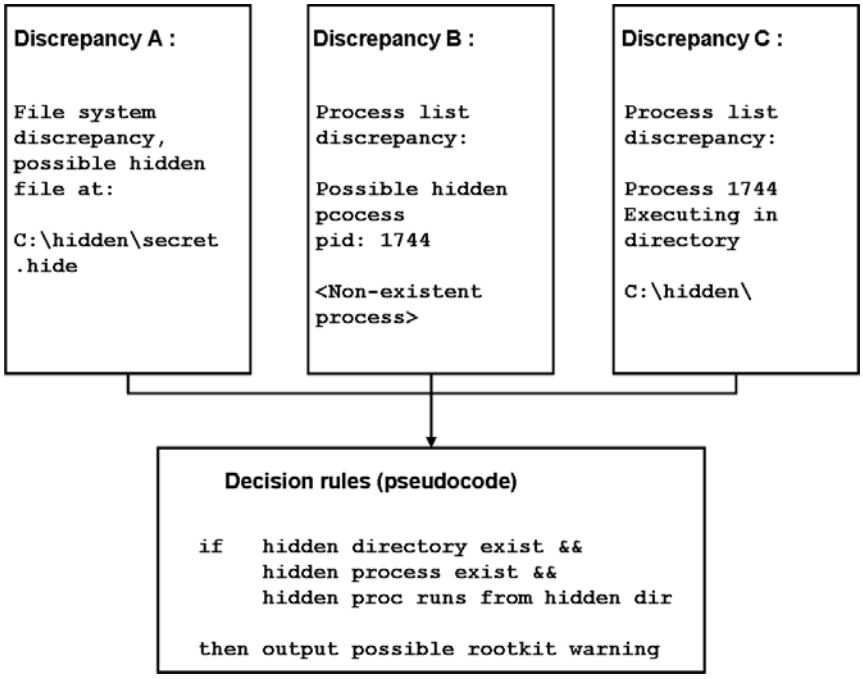

Figure 2. Example of Discrepancy Combinations Used to Detect a Rootkit

5. Results

In order to make the testing process consistent, the victim system is run on VMWare which allows the flexibility of running malicious code

Table 1. Experimental Results

Rootkit	Rootkit PID	Bo2k PID	Bo2k Port	Hidden Data
Hacker Defender	Found	Found	Not Found	Found
AFX	Found	Found	Not Found	Found
Vanquish	N/A	N/A	N/A	Found
FU	Not Found	Found	Not Found	N/A
FUto	Not Found	Found	Not Found	N/A

(i.e. rootkits) without infecting the host operating system. In addition, it allows a user to halt or start an image quickly and reliably, and go back to previous snapshots. As mentioned in Section 1.4, the automated analysis tool targets Windows-based rootkits, more specifically, the OS under test is Windows XP with Service Pack 2.

Initial tests were performed to determine if the detection approach would yield useful results. This meant running the batch script and analyzing the output files one by one in order to determine if there was any evidence of a rootkit. Surprisingly, there was evidence that indicated the presence of a rootkit on the system. For instance, the output file of handle.exe clearly showed a non-existent process with a PID and an object named Hacker Defender. Similar results were also achieved with AFX and Vanquish rookit.

After obtaining the aforementioned results, the next step was to perform two tests against rootkits that employed more sophisticated hiding techniques (i.e. kernel level subversion). For this part of the experiment the rootkits under test were FU and FUto; both hiding BackOrifice's PID. During this tests the detector identified BackOrifice's PID but was unable to identify FU's nor FUto's PID. Notably though, by performing a directory listing of the folder C:\Windows \Prefetch it was determined that bo2K.exe and FU.exe had prefetch files that indicated that the two programs had been executed on the system. This signature could be added to the rootkit detector but similar to any other signature-based detector it is easily defeated by renaming the executable. Table 1 shows a summary of the experimental results. One the most notable result was the fact that BackOrifice was detected with all the rootkits under test (user and kernel level). On the other hand, the detector was unable to locate BackOrifice's open ports.

6. Conclusion

This article presents an automated utility that identifies a rootkit during a live response investigation scenario. The utility effectively identified rootkit evidence in all of the test cases, some with more accuracy

than others. For instance, when testing the detector against Hacker Defender, AFX, and Vanquish, the utility identified each rootkit's PID and the PID of the backdoor they were hiding as well as the folder that they were attempting to hide. In the case of FU and FUto with Back Orifice, the only evidence that pointed to the presence of a rootkit was the name of the executable FU.exe in the prefetch folder. On the other hand, Back Orifice's PID and file name were easily detected even though FU and FUto were trying to hide that information.

Finally, there are a number of areas this paper does not address and should be considered in future research efforts. These areas include: performing an exhaustive examination of the Windows API in order to identify all the different avenues for determining parallel system information (i.e. different ways of gathering network port information), perform the same type of analysis done in this paper in a UNIX environment, and perform similar experiments with other rootkits and backdoors.

7. Acknowledgement

This research was sponsored by the Anti-Tamper Software Protection Initiative Technology Office, Sensors Directorate, of the U.S. Air Force Research Laboratory. The views expressed in this article are those of the authors and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U.S. Government.

References

[1] A. Todd, J. Benson, G. Peterson, T. Franz, M. Stevens and R. Raines, An Analysis of Forensic Tools in Detecting Rootkits and Hidden Processes, *Proceedings of the Advances in Digital Forensics III*, pp. 89–106, 2007.

[2] Alert, Tech Support, Rootkit Detection and Removal, (http://www.pcsupportadvisor.com/rootkits.htm), February, 2006.

[3] King, ST and Chen, PM, SubVirt: implementing malware with virtual machines, *Proceedings of the Security and Privacy, 2006 IEEE Symposium on*, 2006.

[4] K. Dillard, What are user-mode vs kernel-mode rootkits?, (www.searchwindowssecurity.com), 2005.

[5] J. Evers, Microsoft Will Wipe Sony's Rootkit, (www.news.com), November, 2005.

[6] F-Secure, The Threat - Rootkits, (www.f-secure.com).

[7] J. Heasman, Implementing and Detecting a PCI Rootkit, (http://www.ngssoftware.com), Novemver, 2006.

[8] G. Hougland and J. Butler, Rootkits: Subverting the Windows Kernel, Addison-Wesley Professional, 2005.

[9] C. Prosise and K. Mandia, Incident Response & Computer, Forensics,McGraw-Hill Osborne Media, 2003,

[10] S. Joel, S. McClure and G. Kurtz, Hacking exposed: network security secrets and solutions, Berkley: McGraw Hill, 2001.

[11] J. Levine, B. Culver and H. Owen, A Methodology for Detecting New Binary Rootkit Exploits, *Proceedings of the IEEE SouthEastCon* 2003.

[12] A. Chuvakin, An Overview of Unix Rootkits, iALERT White Paper, iDefense Labs, (www.megasecurity.org), February, 2003.

[13] M. Boelen, Rootkit Hunter, (http://www.rootkit.nl/projects/rootkit_hunter.html).

[14] J. Levine, J. Grizzard and H. Owen, Detecting and categorizing kernel-level rootkits to aid future detection, *Proceedings of the IEEE Security & Privacy Magazine*, pp. 24–32, 2006.

[15] J. Rutkowska, Introducing Stealth Malware Taxonomy, Technical Report, COSENIC Advanced Malware Labs, November, 2006.

[16] Holy Father, Hacker Defender, (www.hxdef.org/download.php), 2005.

[17] Rootkit.com, Vanquish Rootkit, (www.rootkit.com).

[18] Rootkit.com, FU Rootkit, (www.rootkit.com).

[19] Rootkit.com, AFX Rootkit, (www.rootkit.com).

[20] Rootkit.com, FUto Rootkit, (www.rootkit.com).

[21] Sourceforge.net, Back Orifice 2000, (http://bo2k.sourceforge.net).