Chapter 6

ANALYSIS OF TOOLS FOR DETECTING ROOTKITS AND HIDDEN PROCESSES

A. Todd, J. Benson, G. Peterson, T. Franz, M. Stevens and R. Raines

Abstract Rootkits pose a dilemma in forensic investigations because hackers use them surreptitiously to mislead investigators. This paper analyzes the effectiveness of online and offline information analysis techniques in detecting rootkits and determining the processes and/or files hidden by rootkits. Five common rootkits were investigated using a live analysis tool, five rootkit detection tools (RDTs) and four offline analysis tools. The experimental results indicate that, while live analysis techniques provide a surprising amount of information and offline analysis provides accurate information, RDTs are the best approach for detecting rootkits and hidden processes.

Keywords: Rootkits, rootkit detection, live analysis, offline analysis

1. Introduction

A rootkit is a program that provides the means to create an undetectable presence on a computer [14]. It is an executable that hides processes, files and registry values from users and other programs. The presence of a rootkit can have an adverse effect on the outcome of an investigation. In legal proceedings, several individuals have claimed ignorance of their crimes, instead laying the blame on an unknown hacker who left a rootkit on their machines [1].

Forensic investigators may detect rootkits using live analysis techniques, rootkit detection tools (RDTs) or by forensically imaging the drive and performing an offline analysis. Live analysis provides valuable, volatile information that may not be available during an offline analysis of a hard drive, e.g., dynamic data such as running processes and open ports, which are only available during system operation. However, when one of the processes on a machine includes a rootkit, the

investigator must question if the results of a live analysis are tainted. When rootkits are running, the data and processes they have modified are still in use by the operating system, but this fact is not visible to the user. In fact, our experiments demonstrate that even if a live response cannot detect what is hidden, it can still provide an investigator with details about what was available to the user. This information can also help the investigator determine what the rootkit was hiding.

RDTs, which detect rootkits and/or files hidden by rootkits, may be employed as part of a live analysis. However, there are limitations associated with any program that attempts to detect rootkits while a rootkit is running. The fundamental problem is the lack of trust in the operating system (and system files and functions). Most RDTs cross-compare information provided by corrupted system calls with system information gathered by the RDT, identifying anomalies as possible rootkits.

A rootkit hides information by manipulating operating system function calls. This information cannot be hidden from an offline scan that does not use the corrupted system calls and searches the entire hard drive for rootkit signatures. However, using offline analysis to determine what a rootkit was attempting to hide is much more difficult.

This paper examines the efficacy of live response, RDTs and offline analysis in detecting rootkits and determining what they conceal. Experiments were conducted on five rootkits: AFXRootkit [22], Vanquish [22], Hacker Defender [15], FU [22] and FUTo [22], which use both user-level and kernel-level hiding techniques [9, 14]. The experimental results demonstrate that live analysis techniques perform better than expected in rootkit-tainted environments, but rootkits obscure the data enough to make investigation difficult. Offline analysis can identify all the data from a hard drive, but finding a rootkit and what it is hiding is nearly impossible. Finally, rootkit detectors provide the most information about a system when a rootkit is running, but the investigator must be willing to accept some loss of forensic integrity of the hard drive.

2. Related Work

Relatively few experiments have been conducted to measure the effectiveness of rootkit detection tools (RDTs). Two exceptions are the studies conducted by Claycomb [6] and CMS Consulting [7].

Claycomb [6] analyzed five publicly-available RDTs (IceSword, GhostBuster, BlackLight, RootkitRevealer, Vice) and one custom RDT (Bluestone). These RDTs were tested against three user-level rootkits (Hacker Defender, AFXRootkit2005, Vanquish) and three kernel-level rootkits (SonyBMG, FU, FUTo) running on Windows XP and Windows Server

2003 machines. The study determined that IceSword (v1.12) and GhostBuster had the best detection capabilities; BlackLight and RootkitRevealer had above average but lesser reliability; and Vice and Bluestone were the least reliable of the RDTs investigated.

The CMS Consulting study [7] analyzed the effectiveness of eight publicly-available RDTs (IceSword, GhostBuster, BlackLight, RootkitRevealer, Flister, Keensense, Process Guard, GMER) against three user-level rootkits (Hacker Defender, AFXRootkit2005, Vanquish) and one kernel-level rootkit (FU). IceSword (v1.18), BlackLight, Keensense, ProcessGuard and GMER were determined to be 100% effective in detecting the rootkits. Flister, RootkitRevealer and Ghostbuster demonstrated above average but lesser reliability.

2.1 Rootkits

With the exception of the Claycomb and CMS Consulting studies, information about rootkit detection is mainly gleaned from chat rooms and online forums rather than from published articles. Most of the claims are largely unproven or anecdotal in nature, but the discussions about rootkit detection and evasion techniques are very valuable. Some of the prominent contributors are Hoglund [14], Butler [3, 4], BPSparks [3, 4, 28], Rutkowska [23], Rutkowski [24, 25], Levine [17, 18], and individuals from Microsoft [29] and Symantec [10].

2.1.1 Traditional Rootkit Techniques. Rootkits are categorized by the techniques they use to hide themselves and their protected files and applications. Traditional (user-level) rootkits operate at the same privilege level as user programs; they intercept API calls and change the outputs of functions by removing references to the files and processes hidden by the rootkit. This is done by either hooking the API call, which redirects the function call to a compromised version of the function, or by patching the correct function with malicious code that modifies the correct output before returning the output to the calling process. Kernel-level rootkits operate in kernel memory space and are, therefore, protected from user-level programs such as certain anti-virus scanners and rootkit detectors. Kernel-level rootkits can also hook API calls, but at the higher privilege level of the kernel. Additionally, a kernel-level rootkit may utilize direct kernel object manipulation (DKOM) and affect the data structures that track system processes [14].

2.1.2 Modern Rootkit Techniques. Kernel data manipulation, which originated in 2003, is generally considered to be the first "modern" rootkit technique. Instead of changing kernel code, a mod-

ern rootkit unlinks its process from the *PsActiveProcessLinkHead* list. This list is not used by the scheduler, so the hidden process still gets CPU time. The technique can be detected by comparing the *PsActiveProcessLinkHead* list to a scheduler list (e.g., *KiDispatcherReadyListHead*) [24]. A more sophisticated technique called "Innocent Threads" evades detection by unlinking from the *PsActiveProcessLinkHead* list and having the malicious thread masquerade as one belonging to a legitimate system process (e.g., `winlogon.exe`) [23].

2.1.3 Advanced Rootkit Techniques. Some of the newest rootkit techniques have been implemented during the past year. These include using polymorphic code, using covert channels and subverting virtual memory. Rootkits with polymorphic code generators allow the creation of a different binary image for every system on which they are installed [23]. Covert channel communications permit rootkits to communicate with their hidden applications without being detected [27]. Finally, although rootkits are becoming more proficient at controlling their execution, they still find it difficult to conceal their code and memory-based modifications within operating system components. This leaves them vulnerable to detection by even the most primitive in-memory signature scans. Virtual memory subversion enables rootkits to control memory reads by the operating system and other processes, allowing them to see only what the rootkit wants them to see [3].

2.2 Rootkit Detection

Rootkit detectors are typically categorized into five classes [4]:

- **Signature-Based Detectors:** These detectors scan system files for byte sequences representing rootkit "fingerprints."
- **Heuristic/Behavior-Based Detectors:** These detectors identify deviations from "normal" system patterns or behavior.
- **Cross-View-Based Detectors:** These detectors enumerate system parameters in at least two different ways and compare the results. Typically, this means invoking APIs for information and using an algorithm specific to the RDT to obtain the same information without going through the APIs.
- **Integrity-Based Detectors:** These detectors compare a current snapshot of the filesystem or memory with a trusted baseline.
- **Hardware-Based Detectors:** These detectors are independent of the potentially subverted operating system. They typically have

their own CPU and use Direct Memory Access (DMA) to scan physical memory for rootkit signatures such as hooks in the System Service Descriptor Table (SSDT), alterations to kernel functions, and modifications to key data structures.

3. Experimental Methodology

To determine which of the three analysis techniques (live analysis, RDTs or offline analysis) provides the most information about rootkit behavior, three sets of experiments were conducted against six target hosts. The first set of experiments (live response) involved one system under test (SUT), the second set (RDTs) involved five SUTs, and the third (offline analysis) involved four SUTs. A full-factorial design consisting of sixty total experiments was conducted. Five different metrics were employed. The metrics were designed to capture forensic information about rootkits, i.e., what they were concealing from the user in terms of processes, services, drivers, ports and files.

Table 1. Target host configurations.

Configuration	Operating System	Rootkit Installed	Hiding Technique
A	Windows XP SP2	None	n/a
B	Windows XP SP2	AFXRootkit (2005)	User-level
C	Windows XP SP2	Vanquish (0.2.1)	User-level
D	Windows XP SP2	Hacker Defender (1.0.0r)	User-level
E	Windows XP SP2	FU (2004)	Kernel-level
F	Windows XP SP2	FUTo (2006)	Kernel-level

3.1 Target Host Configurations

The baseline target host (Configuration A in Table 1) was a clean installation of Microsoft Windows XP Professional with Service Pack 2 running on a virtual machine using VMWare 5.5. Five publicly available rootkits were used as the workload to the SUTs, consisting of three user-level rootkits (AFXRootkit, Vanquish, Hacker Defender) and two kernel-level rootkits (FU, FUTo). AFXRootkit uses API patching to hide the resident directory of the executable, as well as files in the directory, registry keys with the same name as the directory, and processes owned by programs running from the directory [2]. Hacker Defender uses API hooking to hide files, processes, registry settings and ports, as long as they are in the `hxdef.ini` file. Additionally, it creates a

password-protected backdoor on the host and redirects packets to help mask the source of traffic for an attack against a third machine [15]. Vanquish uses DLL injection for API hooking to hide all files, services and registry settings containing the magic string "vanquish." It also has a password logging function bundled into the executable [31]. Both FU and FUTo use DKOM to manipulate the handle table to hide processes; they may also raise the privileges of processes by manipulating the table. FUTo, the successor to FU, was created to address certain weaknesses in IceSword and BlackLight. It evades detection by manipulating the process handle table without using function calls [26].

Table 2. Experiment Set I.

Experiment Set	Live Response SUTs
I-1	Windows Forensic Toolchest [19]

3.2 Experiment Set I (Live Response)

Experiment Set I involved performing a live response on each workload configuration using one SUT in order to determine the SUT's effectiveness in detecting the workload rootkits (Table 2). The SUT used was the Windows Forensic Toolchest (WFT) [19]. WFT is specifically designed for detecting processes, services, drivers, ports and files. WFT functions that may be used for rootkit detection are listed in Table 3.

3.3 Experiment Set II (Rootkit Detectors)

Experiment Set II involved the execution of five rootkit detectors (Table 4) on each workload configuration (Table 1) to determine the effectiveness of each SUT in detecting the workload rootkits. RootkitRevealer provides user-level and kernel-level detection by comparing information returned by the Windows API and raw FAT/NTFS structures [8]. RKDetector [20] uses its own NTFS/FAT32 filesystem driver to provide a filesystem browser, rootkit detector, ADS scanner, registry browser and hidden registry key scanner [21]. BlackLight loops through all possible process IDs attempting to open each process, and compares the information it gathers with that returned by the Windows API [12]. IceSword [30] detects hidden process, services, drivers, files, ports and registry settings, and identifies hooked system calls and open TCP/UDP ports [26].

Table 3. Functions packaged in WFT.

WFT Function	Target
PSLIST	Processes
PS	Processes
LISTDLLS	Processes
PSTAT	Processes
TLIST	Processes
CMDLINE	Processes
HANDLE	Processes
PSSERVICE	Services
SC	Services
NET START	Services
SERVICELIST	Services
DRIVERS	Drivers
NET STAT	Ports
FPORT	Ports
OPENPORTS	Ports
TREE	Files

Table 4. Experiment Set II.

Experiment Set	Rootkit Detector SUTs
II-1	RootkitRevealer (1.7) [8]
II-2	RKDetector (Beta 2.0) [20]
II-3	BlackLight (Beta 2.2.1046) [12]
II-4	IceSword (1.12) [30]
II-5	IceSword (1.18) [30]

Table 5. Experiment Set III.

Experiment Set	Offline Analysis SUTs	Tool Type
III-1	Clam-AV [16]	Signature-Based AV
III-2	F-Prot [11]	Signature-Based AV
III-3	EnCase [13]	Forensic Investigation
III-4	Autopsy [5]	Forensic Investigation

3.4 Experiment Set III (Offline Analysis)

Experiment Set III involved performing an offline analysis on each of the workload configurations using four different SUTs (Table 5) to deter-

mine the effectiveness of each SUT in detecting the workload rootkits. Two SUTs are signature-based anti-virus tools and the other two are forensic tools that were used to visually inspect the target workloads. Rootkit information cannot be hidden from an offline analysis. Therefore, the study focused on determining what was hidden and when it was hidden (if the rootkit was operating when the machine was seized).

4. Experimental Results and Observations

This section presents the results and observations from the three sets of experiments that covered ten SUTs and six target configurations.

4.1 Workload Capabilities and Limitations

Each of the five rootkit workloads hides different processes, services, drivers, ports and/or files.

- **AFXRootkit:** This user-level rootkit hides two running processes (rootkit `root.exe` and backdoor `bo2k.exe`), one running service (`treasure`), one port (`54320`), and several files including its current directory (`c:\treasure`). No drivers are used to root the system.

- **Vanquish:** This user-level rootkit is not a constantly running process. It starts, hides its payload using patching, and then stops. Vanquish hides the rootkit service (`vanquish`), port (`54320`), and all files and directories with "vanquish" in their names.

- **Hacker Defender:** This user-level rootkit hides two running processes (rootkit `hxdef100.exe` and backdoor `bo2k.exe`), one running service (`Hacker Defender100`), one port (`54320`), and the files specified in the rootkit's initialization file (`hxdef100.ini`). Hacker Defender also hides a driver (`hxdefdrv.sys`), which it uses to gain system access.

- **FU:** This kernel-level rootkit only hides processes. FU is not a constantly running process, so it only hides the one process it is directed to hide (backdoor `bo2k.exe`). FU does not hide ports or files, and it does not create a system service. FU uses a driver (`msdirectx.sys`) to gain system access, but it does not hide this driver.

- **FUTo:** This kernel-level rootkit only hides processes. Like its predecessor FU, FUTo is not a constantly running process, so it only hides one running process (backdoor `bo2k.exe` in our tests). FUTo

does not hide ports or files, and it does not create a system service. FUTo uses a driver (`msdirectx.sys`) to gain system access, but it does not hide it.

Table 6. Experiment Set I results.

Rootkit	Processes	Services	Drivers	Ports	Files
AFXRootkit	Y	Y	n/a	N	N
Vanquish	Y[2]	N	n/a	Y[2]	N
Hacker Defender	Y[1]	N	Y	N	N
FU	Y[1]	n/a	Y[1]	Y[2]	Y[2]
FUTo	N	n/a	Y[1]	Y[2]	Y[2]

[1] WFT detected the rootkit, but provided limited information.
[2] Rootkit did not attempt to hide this information.

4.2 Experiment Set I Results

The first set of experiments performed a live analysis using WFT on each of the five rootkits. The results are summarized in Table 6.

Since rootkits subvert the system functions that list active process, open ports and other files, it was surprising that WFT's incident response tools were able to detect so much information hidden by the rootkits. WFT provided considerable information associated with running processes and system drivers. FUTo was the only rootkit that could completely hide information about the backdoor's process. However, although the results indicate a high rootkit detection rate, WFT was unable to provide reliable information about system files, open ports and system services. Nevertheless, WFT provided useful information and, when it could not detect a rootkit, still gave an accurate depiction of what the user would normally see.

Note that an investigator must look closely at the PSTAT and PROCESS HANDLES utilities when using WFT to detect rootkit processes. PSTAT provides generic process information, which helps detect simple rootkits, e.g., AFXRootkit. PROCESS HANDLES provides more detailed system information that was scrubbed clean only by FUTo. The information listed here may only identify a "<Non-existent Process>," but this indicates to the investigator that a covert process is running.

When using WFT to detect items (e.g., drivers, ports and files) that a rootkit may be hiding, the investigator should use the NET START utility to identify services and DRIVERS to identify drivers. NET START only works against basic rootkits (e.g., AFXRootkit). DRIVERS lists

system drivers, but it was able to detect all the rootkit drivers in our experiment. Rootkits also hide ports and files; however, WFT did not reveal any information about ports and files that were actively hidden.

Table 7. Experiment Set II results.

RDTs	AFXRootkit	Vanquish	Hacker Defender	FU	FUTo
RootkitRevealer	Y	N	Y	N[1]	N[1]
RKDetector	Y	Y	Y	N[1]	N[1]
BlackLight	Y	Y	Y	Y[3]	Y[23]
IceSword 1.12	Y	Y	Y	Y[3]	N
IceSword 1.18	Y	Y	Y	Y[3]	Y[23]

[1] RDT did not look for hidden processes, services, drivers and ports.
[2] RDT detected a discrepancy, but could not provide all the process information.
[3] RDT did not detect the actual rootkit, just the hidden processes.

4.3 Experiment Set II Results

Table 7 summarizes the results of using rootkit detection tools (RDTs) in a live analysis and their success in detecting rootkits.

The results show that the four RDTs fall into two groups: one group (RootkitRevealer and RKDetector) only detects hidden files and is not effective against all rootkits, while the second group (BlackLight and IceSword) is highly effective against all rootkits. BlackLight detects the same rootkits as IceSword. However, IceSword is a more robust tool that identifies hooked system calls, hidden drivers and open TCP/UDP ports. This information enables an investigator to classify the anomalous behavior identified by the detector as an actual rootkit. The following subsections summarize the four RDTs used in our study.

4.3.1 RootkitRevealer. RootkitRevealer primarily identifies hidden files and registry settings; it does not look for hidden processes, ports, services and drivers. It can detect rootkits that hide executables, images and other illegal content, but not those that can hide running processes. Specifically, RootkitRevealer detected all the hidden files in the current directory of AFXRootkit and all the files and registry settings listed in the Hacker Defender initialization file (`hxdef100.ini`), but it was unable to detect the files hidden by Vanquish. RootkitRevealer listed no discrepancies when executed against FU and FUTo because they only hide processes, but this is misleading because the rootkits were hiding information from the user. RootkitRevealer had mixed results for

the five rootkits that were tested, and it provided the least amount of information.

4.3.2 RKDetector. RKDetector primarily identifies hidden files; it does not look for hidden processes, ports, services and drivers. Like RootkitRevealer, it can detect rootkits that hide executables, images and other illegal content, but not those that can hide running processes. It had good results when executed against the five rootkits, detecting the three that hide files. RKDetector detected all the hidden files in the same directory as AFXRootkit, all the hidden files containing the keyword "vanquish," and all the hidden files and registry settings listed in the Hacker Defender initialization file (`hxdef100.ini`). However, it was ineffective against FU and FUTo because they do not hide files. RKDetector presents its findings in an easy-to-understand file tree that marks directories containing hidden files and directories, and highlights the actual hidden files and directories. In addition to standard rootkit detection, it allows the investigator to explore the registry and alternate data streams on the hard drive.

4.3.3 BlackLight. BlackLight scans a system for hidden items, providing a list of hidden items and options to remove or clean them. However, the list of hidden items does not include services, drivers, ports and registry settings. BlackLight identified all the files hidden by AFXRootkit, Vanquish and Hacker Defender, but it did not list the folder containing these files which was, in fact, hidden. It detected FU's hidden process (`bo2k.exe`). However, when executed against FUTo, BlackLight detected the hidden process, but was unable to provide its name. Against both FU and FUTo, BlackLight detected the process hidden by the rootkit, but was unable to identify the rootkit. Nevertheless, BlackLight performed extremely well in our experiments, detecting all five rootkits.

4.3.4 IceSword. IceSword identifies hidden processes and services, and attempts to provide a clean list of all files, registry settings, ports, items set to start up automatically, browser helper objects and message hooks. IceSword detected all the files, directories, services, ports and processes hidden by AFXRootkit, Vanquish and Hacker Defender. IceSword v1.12 and v1.18 both detected FU's hidden process (`bo2k.exe`). IceSword v1.12 could not detect FUTo's hidden process; on the other hand, IceSword v1.18 detected the process and provided its name. However, when executed against both FU and FUTo, IceSword could not identify the rootkit that was responsible for hiding the process.

Nevertheless, IceSword performed extremely well in our experiments, detecting all five rootkits (v1.18).

Table 8. Anti-virus scan results.

AV Scanner	AFXRootkit	Vanquish	Hacker Defender	FU	FUTo
Clam-AV	N	N	Y	Y	N
F-Prot	Y	N	Y	Y	N

4.4 Experiment Set III Results

Table 8 presents the results produced by anti-virus scans during the offline analysis. The results are surprising, since the tools are specifically designed for UNIX and Linux environments, not Windows. Visual analysis of the images using EnCase [13] and Autopsy [5] could not determine when the rootkit was running and what it was hiding. However, some information was obtained during offline analysis. In particular, identifying the rootkit can help determine what is hidden, e.g., knowing that the Vanquish rootkit was present could lead an investigator to search for items using the keyword "vanquish." AFXRootkit and Vanquish also provide some relative time information. They update the last access date and time stamp of their DLL files (`hook.dll` and `vanquish.dll`, respectively) upon startup. These findings indicate that it may be difficult to detect when a rootkit was running via offline analysis; this is discernable from DLL time stamps only in some instances. This is a concern with the FU and FUTo rootkits, as they must be started by a user on the machine and there is no mechanism to determine if they were running.

5. Case Study

The investigative techniques and tools identified in this paper were used in the 2006 Cyber Defense Exercise (CDX), an annual network defense competition between student teams from the five U.S. service academies and the Air Force Institute of Technology. The competition teams designed, implemented and defended a realistic network that provided services based on an overall architectural concept of operations. As part of the initial network setup, each team received preconfigured systems of unknown security state. This required each system to be analyzed to determine vulnerabilities.

Table 9. Preconfigured systems.

Configuration	Operating System	System Role	Installed Rootkit
A	Windows Server 2003 SP1	Exchange Server	Hacker Defender
B	Windows Server 2003 SP1	Domain Controller	Hacker Defender
C	Fedora Core 2	Web Server	F-ck'it Rootkit
D	Fedora Core 2	SMB Server	F-ck'it Rootkit
E	Windows XP SP2	Workstation 1	Vanquish
F	Windows XP SP2	Workstation 2	None
G	Windows XP SP2	Workstation 3	AFXRootkit
H	Windows XP SP2	Workstation 4	None

5.1 Preconfigured Systems

Each competition team was given eight preconfigured systems (Table 9). The systems were virtual machines operating on a Windows XP Service Pack 2 host system using VMWare 5.5. Most of the preconfigured systems contained malicious software, e.g., rootkits, backdoors, keyloggers, as shown in Table 9.

Table 10. Investigation tools.

Investigation Tool	Tool Type
Clam-AV	Signature-Based AV
F-Prot	Signature-Based AV
IceSword v1.12	RDT (Windows)
RootkitRevealer v1.7	RDT (Windows)
BlackLight (Beta 2.2.1046)	RDT (Windows)
Rootkit Hunter v1.2.8	RDT (Linux)
Chkrootkit	RDT (Linux)

5.2 Investigation Methodology

Each preconfigured system was analyzed for malicious software including rootkits. A combination of offline and live response tools, including RDTs, was used (Table 10).

Table 11. Investigation results.

Tool	Exchange Server	Domain Controller	Web Server	SMB Server	Client	Client
Clam-AV	Y	Y	N	N	N	N
F-Prot	Y	N	N	N	N	Y
IceSword v1.12	Y	Y	n/a	n/a	N[2]	Y
RootkitRevealer v1.7	Y	Y	n/a	n/a	N	Y
BlackLight (Beta 2.2.1046)	Y	Y	n/a	n/a	N	Y
Rootkit Hunter v1.2.8	n/a	n/a	Y	Y	n/a	n/a
Chkrootkit	n/a	n/a	Y[1]	Y[1]	n/a	n/a

[1] Information was discovered, but the rootkit was not identified.
[2] The presence of the rootkit was not revealed, but the hidden files could be viewed.

5.3 Analysis Results

The results of using the various tools for offline and live analysis and their success at detecting hidden process and rootkits on the six pre-configured systems are shown in Table 11. Note that all the systems contained a rootkit, but the specific rootkit varied between machines.

The results indicate that RDTs are effective even in unconstrained environments and without prior knowledge of the rootkits involved. All three Windows RDTs appeared to be equally effective. However, in practice, IceSword is preferred as it provides additional information that permits the viewing of all hidden files. Rootkit Hunter was the most effective and robust tool of the two Linux RDTs tested. Clearly, RDTs are the best tools for conducting forensic investigations of rootkits.

6. Conclusions

Rootkits hide processes, files and other system information from users and often obscure malicious activity in digital forensic investigations. However, forensic investigators may apply three techniques to detect rootkits and hidden processes: live response, rootkit detection tools (RDTs) and offline analysis.

Our experiments indicate that using a rootkit detection tool during a live response generates the largest amount of data and the most useful information. It is still possible to detect rootkits using a non-invasive live response, but live response procedures are subject to the effects of the rootkits and it can very time consuming to sift through the resulting

data. Offline analysis has the advantage of being able to examine all the data from the hard drive, but finding a rootkit, what it was hiding, and when it was hiding are nearly impossible. RDTs are very effective at determining the presence of rootkits and identifying their targets, but the investigator should be willing to pay the penalty of lower forensic integrity of the evidentiary hard drive.

Our future research will attempt to determine the exact effects of RDTs and how they might compromise the overall forensic integrity of digital evidence. This may make it possible to create an offline RDT, with the benefit that the rootkit would not be able to run and, therefore, not be able to hide any information. An alternative would be to perform anomaly detection on the generated live response data to identify data items associated with the rootkit, thereby speeding up live response analysis.

Acknowledgements

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

References

[1] E. Abreu, Hackers get novel defense; the computer did it (www.forbes.com/markets/newswire/2003/10/27/rtr1124430.html), 2003.

[2] Aphex, `ReadMe.txt` (www.iamaphex.net), 2006.

[3] J. Butler and S. Sparks, Windows rootkits of 2005: Part two (www.securityfocus.com/infocus/1851), 2005.

[4] J. Butler and S. Sparks, Windows rootkits of 2005: Part three (www.securityfocus.com/infocus/1854), 2006.

[5] B. Carrier, *File System Forensic Analysis*, Addison-Wesley, Boston, Massachusetts, 2005.

[6] C. Claycomb, Analysis of Windows Rootkits, M.S. Thesis, Department of Electrical and Computer Engineering, Air Force Institute of Technology, Wright-Patterson Air Force Base, Ohio, 2006.

[7] CMS Consulting, Hidden rootkits in Windows (www.task.to/events/presentations/TASK_Hidden_Rootkits_in_Windows.pdf), 2005.

[8] B. Cogswell and M. Russinovich, RootkitRevealer v1.71 (www.sysinternals.com/Utilities/RootkitRevealer.html).

[9] K. Dillard, What are user-mode vs. kernel-mode rootkits? (search windowssecurity.techtarget.com/originalContent/0,289142,sid45_gci1086469,00.html), 2005.

[10] E. Florio, When malware meets rootkits, *Virus Bulletin*, 2005.

[11] Frisk Software International, F-Prot Antivirus Scanner (www.f-prot.com/products/home_use/linux).

[12] F-Secure Corporation, Blacklight (www.f-secure.com/blacklight/blacklight.html).

[13] Guidance Software, EnCase (v.4) (www.guidancesoftware.com).

[14] G. Hoglund and J. Butler, *Rootkits: Subverting the Windows Kernel*, Addison-Wesley, Boston, Massachusetts, 2005.

[15] Holy_Father, Hacker Defender (hxdef.org/download.php).

[16] T. Kojm, Clam AntiVirus (www.clamav.net).

[17] J. Levine, B. Culver and H. Owen, A methodology for detecting new binary rootkit exploits, *Proceedings of the IEEE SouthEastCon*, 2003.

[18] J. Levine, J. Grizzard, P. Hutto and H. Owen, A methodology to characterize kernel level rootkit exploits that overwrite the system call table, *Proceedings of the IEEE SoutheastCon*, pp. 25–31, 2004.

[19] M. McDougal, Windows Forensic Toolchest (WFT) (www.foolmoon.net/security/wft), 2005.

[20] RKDetector.com, RKDetector v2.0 (www.rkdetector.com).

[21] RKDetector.com, RKDetector v2.0 Engine (www.rkdetector.com).

[22] Rootkit.com (www.rootkit.com/download.php).

[23] J. Rutkowska, Concepts for the Stealth Windows Rootkit (The Chameleon Project) (invisiblethings.org/papers/chameleon_concepts.pdf), 2003.

[24] J. Rutkowski, Advanced Windows 2000 rootkit detection (hxdef.org/knowhow/rutkowski.pdf), 2003.

[25] J. Rutkowski, Execution path analysis: Finding kernel rootkits (doc.bughunter.net/rootkit-backdoor/execution-path.html), 2004.

[26] P. Silberman, FUTo (uninformed.org/?v=3&a=7), 2006.

[27] Simple Nomad, Covering your tracks: Ncrypt and Ncovert, presented at *Black Hat USA 2003* (www.blackhat.com/html/bh-media-archives/bh-archives-2003.html), 2003.

[28] S. Sparks, Shadow Walker: Raising the bar for rootkit detection, presented at *Black Hat USA 2005* (www.blackhat.com/presentations/bh-jp-05/bh-jp-05-sparks-butler.pdf), 2005.

[29] Y. Wang, B. Vo, R. Roussev, C. Verbowski and A. Johnson, Strider Ghostbuster: Why it's a bad idea for stealth software to hide files, Microsoft Research Technical Report, MSR-TR-2004-71, Microsoft Corporation, Redmond, Washington, 2004.

[30] XFocus.net, IceSword (v1.12 and v1.18) (www.xfocus.net).

[31] XShadow, Vanquish v0.2.1 (www.rootkit.com/vault/xshadoe/readme.txt), 2005.