Global Information Assurance Certification Paper

Windows Rootkits

Adam Gaydosh

GSEC Practical Assignment Version 1.4b Option 1

March 24, 2004

Abstract:

Rootkits are an emerging threat that responsible computer operators need to fully understand in order to protect the integrity of their systems and contribute to the overall stability of the internet. This paper focuses specifically on understanding offensive and defensive rootkit technologies for the Microsoft Windows NT, 2000, and XP Operating Systems (OS). We begin by reviewing common attack scenarios and Windows malware operation, followed by an overview of Windows rootkit technology and an examination of historical and contemporary examples. Finally, we discuss the Windows security architecture and examine some tools and strategies to prevent and detect a system intrusion.

TABLE OF CONTENTS:

1 Introduction

2 Know your Enemy

2.1 Windows Rootkits

2.2 Evolution

2.3 Current Threats

3 Windows Security Architecture

3.1 Host Hardening

3.2 Intrusion Detection

3.3 Rootkit Detection Tools

4 Conclusion

Appendix A: References

1 Introduction

In this paper we will examine protection and exploitation techniques of the contemporary Microsoft Windows OS family, including Windows NT, 2000, and XP. We will specifically focus our examination on the impact of emerging Windows rootkit technologies, including their operation, prevention, detection and removal. The technical content assumes a familiarity with basic system administration and network security, and is intended for system administrators, incident responders, and forensics analysts.

2 Know your Enemy

In order to effectively defend your host system, you should study the motives and intentions of potential attackers, to gain a solid understanding of their methods of attack. We can broadly categorize attacks as those targeted at specific systems and those targeted at random. The primary differences are the degree of effort exerted by the attacker in order to subvert your defenses, and their intentions of subsequent system use.

An common of example a targeted attack is the attempted theft, destruction, or manipulation of sensitive corporate intellectual property. This attacker may be a disgruntled employee launching attacks from inside your network perimeter, and perhaps even have legitimate privileges to his target. It is however just as likely the attacker is outside of your organization, attacking accross the Internet. In the case of a randomly targeted attack, objective of the attacker is often simply the use of system resources such as network bandwidth or storage space. In the case that either attack is successful, the attacker will want to ensure they can retain control of their victim's host, hide their presence, and remove all traces of the initial attack, and rootkits are the type of malware that can do exactly that. However, while rootkits may provide attacker's a desirable level of subterfuge, administrator or system account privileges are required for them to be installed, and this ultimately requires an independently exploitable vector for privilege escalation.

2.1 Windows Rootkits

In his classic paper on the original Windows rootkit, Greg Hoglund defines a rootkit as "…a set of programs which *PATCH* and *TROJAN* existing execution paths within the system (1)". As we touched on previously, rootkits are tools that are often used by intruders to conceal their control of a system, but are not the tool that is used to actually gain the system's control. There are some fundamentals to rootkit operation and design that must be discussed in order to clearly understand how they are used by attackers to conceal themselves.

The two primary types of Windows rootkits are user-mode and kernel-mode. Both share the objective of obfuscating system resource usage from legitimate users, which generally includes manipulating file system objects, services, and registry keys. User-mode rootkits modify or replaces binaries, which retain their restricted access to system resources through the application programming interface (API). A classic example is the modified *netstat.exe* which does not display activity on specified ports of the attackers choosing, such as the ones to which their remote shell and ftp server our bound! In fact, user-mode rootkits will often include a collection of altered system executables and other binaries provide the attacker with a wide array of particular services, while attempting to hide from authorized users any trace of their existence. However, with each iteration of the Windows OS family, the effectiveness of the native security controls has continually evolved to help defend against the classic user-mode rootkits, although they are not foolproof and may provide the administrator with a false sense of security. Take for example Window File Protection (WFP), which

"protects critical system files that are installed as part of Windows (for example, files with a .dll, .exe, .ocx, and .sys extension and some True Type fonts). WFP uses the file signatures and catalog files that are generated by code signing to verify if protected system files are the correct Microsoft versions (Microsoft)."

While enabling WFP certainly goes a long way in preventing the installation of user-mode rootkits by unauthorized users, an attacker who has sufficient privilege may simply disable WFP, in some cases by changing a single registry key.

Kernel-mode rootkits take the subterfuge deeper into the internal layers of the OS by effectively patching the system kernel. They can manipulate or replace APIs by employing device drivers that allow the malware to directly access the core system resources. Some kernel-mode rootkits may be sophisticated enough to escape detection by some, if not all of the tools available today, and are the focal point for most contemporary Windows rootkit research and development. The traditional ways kernel-mode rootkits hide various objects include hooking dynamic link library's (DLL's) functions (API/ Import Address Table [IAT] hooking), modifying DLL's functions (raw code change), hooking entries in Service Descriptor Table (SDT) / System Service Table (SST) / KiServiceTable (very popular), hooking interrupt descriptor table (IDT) 2Eh entry, and modifying kernel code (raw code change) (Rutkowska, November 2003). The end result of all these techniques is the ability to be able to control process calls to protected system resources.

2.1.1 Evolution

The very term 'rootkit' reveals the UNIX background of this class of malware, referring the hidden root level access they provide to attackers of the UNIX platform. The first publicly available kernel-mode Windows rootkit was *NTRootkit* by Greg Hoglund, which set the standard for functionality that most future generations of Windows rootkits have continued to provide, albeit employing progressively more sophisticated techniques to hide their presence.

This rootkit has been designed as a kernel mode driver that runs with system privileges right at the core of the system kernel. Given this fact, it has access to all resources of the operating system, thus having a broad field of action. In order to install it one requires the administrator's permissions whilst simple net start/net stop commands are sufficient to activate/disactivate (sp.) it respectively (Bobkiewicz, 4).

The hallmark features of Windows rootkits that *NTRootkit* provides include the ability to hide services, files, processes and registry keys. *NTRootkit* was never fully realized, but provided the framework for features that have been found in later generation Windows rootkits, such as serving a remote command shell or runnninga keyboard sniffer that can intercept all consol keystrokes (Kuepper, 17). *NTRookit* project is hosted at Hoglund's seminal web site, rootkit.com, which itself has provided a fertile environment helping fuel the emergence of many of the latest generations of Windows rootkits, such as *Vanquish, He4Hook*, and *Fu*.

2.1.2 Current Threats

The collaborative nature of malware development and knowledge sharing amongst computer researchers has resulted in windows rootkit technologies evolving comparably to the dissemination of information on the operation of the lowest layers of Windows. As such, contemporary Windows rootkits are sophisticated software designed to elude the most meticulous administrators. The latest generation of kernel-mode rootkits have evolved beyond hooking or dll injection, employing advanced techniques of resource manipulation that continue to test the current limits of rootkit detection. For example, by using Direct Kernel Object Manipulation (DKOM) in memory, "a device driver or loadable kernel module has access to kernel memory…and a sophisticated rootkit can modify the elements directly in memory in a relatively reliable fashion to hide (Butler, 12)."

As Windows rookits continue to evolve, the trend has been to peel away the layers of access restrictions in the OS in order to inference as directly as possible with the physical hardware. Future rootkit threats that may be beyond the capabilities of current detection methods may reside solely in peripheral hardware such as video card and hard drive controllers.

3 Windows Security Architecture

It is important to understand the internals of the Windows security architecture, in order to effectively enable it's native security controls as well as understand their potential for exploitation. What follows is a brief introduction to the Windows security architecture concepts that are imperative to understanding a technical analysis of any Windows rootkit. The OS is logically divided into two area's of privilege, the Userland and the Kernel. Useland contains applications, services and process and the environmental subsystems, provided as the user's interface to the OS. The Kernel contains the Executive API, Registry, and Hardware abstraction layer (HAL) that the userland components use to control the physical system hardware. Stepping through the execution of a user mode application, it's subsystem DLLs access the kernel through the appropriate API, which uses the HAL to ultimately control access to the physical hardware for execution. While a familiarity with the underpinnings of the kernel architecture is important for understanding the operation of some of the more advanced rootkits in use today, much of our defensive security posturing is concerned with the access controls available to Userland.

Like any large piece of complex software, Microsoft Windows has seemingly endless potential for security bugs and vulnerabilities, not to mention operator error. Regardless of whether or not you agree with the preceding statement, you are well advised to take a moment to consider it, because more often then not, it is the point of view of your attacker. Luckily, we can turn these sentiments back on him by applying them to our own system's security analysis. While we will leave it to the bug trackers and other security researcher to keep finding new vulnerabilities, there is plenty of opportunity for us to be proactive in operator error reduction. Considering the size and complexity of any OS, and particularly Windows, perhaps the most prudent approach to system security is that of least privilege. While logically this applies to using well defined Groups and Users to appropriately delegate permissions to all system objects, it should also be extended to include reducing which services and process run to the least number necessary. In theory, a system's level of security is inversely proportional to its usability, so that any additional process, service, or functionality will compromise your threat threshold. By reducing the complexity of your system you can go a long way in reducing its vulnerability to compromise.

3.1 Host Hardening

Due diligence in properly administering your system is the best defense against rootkits and other malware. There are already many fine resources available that provide extensive detail on effectively securing Windows hosts that administrators should review before installing a new system, so we will not

rehash the details extensively here. Two good places to start are the Microsoft Security homepage (http://www.microsoft.com/security/) for hardening checklists and other security information on your specific version of Windows, and the Center for Internet Security (http://www.cisecurity.org/), which provides independent security benchmarks and assessment software. However, it still is useful to discuss how some particular precautions, which although may be considered standard best practice of systems administration, are also strategies that particularly aide in preventing, or at least detecting, a successful rootkit intrusion.

First and foremost, be sure to enable strict Discretionary Access Control Lists (DACLS) on all system files and folders, shares and registry keys, so that unauthorized users cannot add additional services and other objects. By then enabling the corresponding System Audit Control Lists (SACLS), security events will be audited and available for investigation (Carvey, 2004). Diligent application of vendor issued patches for the OS and all applications is absolutely mandatory for any system to withstand the barrage of automated attacks to which internet-facing hosts are constantly exposed, and it should go without saying that all pre-deployment configuration and patching must be done on a protected network segment or entirely off-line. Services and processes should be operated under the principal of least privilege to reduce the potential for further exploitation in the event of a successful attack. Of course if an attacker is an unauthorized system user who has obtained root privileges through an independent attack vector, he will be able to roll back any security configuration and still install a rootkit. The importance of disciplined host hardening can not be overstated in limiting the exposure to attacks that may lead to total system compromise.

When the initial hardening of clean build has completed, it is imperative to get a snapshot of the system configuration in a known good state, to provide a base-line for future incident detection. For example, md5 hashes of the file system can then be compared to archived versions to ensure they have not changed. However, any data gathered on expected production system behavior may be used in the future to spot even subtle anomalies, so that it's use need not be limited to providing conclusive evidence of a compromise.

3.2 Intrusion Detection

Determining that a system has been compromised is not always a strait-forward task. Often times the administrator will notice an anomaly that is indicative of malicious system use and proceed to review the host. First a review of the system logs should indicate if any unauthorized activity has been attempted, assuming auditing has been enabled as discussed above in host hardening. Microsoft has provided a lush interface to the Window API in Windows Management Instrumentation (WMI) for effectively managing any system object.

WMI accommodates a variety of popular scripting languages such as PERL and VBS and provides Windows administrators the perfect opportunity to automate critical system reporting for accurate data correlation. There is no shortage of scripting resources on the Internet for system administrators. Carvdawg's Perl Page (http://patriot.net/~carvdawg/scripts) is one such location that provides several scripts specifically developed for Windows IR (*procdmp.pl*) and log monitoring (*WmiEvt.pl*).

While diligent system monitoring and testing may provide evidence of certain compromises, the rootkits we are examining have been specifically designed to hide their operation from other system users. For detecting the presence of user-mode rootkits on suspect systems, network port scanning, from either a remote host or locally running clean binaries on read-only media, can be extremely effective. Using the original example of the Trojaned *netstat.exe*, you can compare the local output to the results of a remote scan to quickly spot indiscrepancies and zero in on suspicious endpoints.

Because kernel-mode rootkits can effectively invalidate any reliance on the host OS for intrusion detection, examining the suspect system from an alternate OS burned to a bootable CD can effectively reveal their presence. This trick will not be able to detect all malware however, so that sniffing the host's network traffic may be required. In some cases, monitoring network traffic may be only way to detect malicious activity, and can substantiate evidence found on the victim computer (Casey, 29). Like most methods of intrusion detection, this method is most effective if you have an extensive baseline analysis of clean traffic patterns that can be used to detect deviations in live packet captures. Similarly, it should only be trusted to determine that there may be some malicious activity and never used to determine that the host is emphatically clean.

Execution Path Analysis (EPA) is an advanced Windows rootkit detection technique that provides reliable detection by exploiting "the processor stepping mode to measure the number of instructions executed in system kernel an DLLS, in order to detect additional instructions inserted by malicious code (Rutkowska, 2003)." Because EPA uses statistcical analysis to determine if the number of instructions execute on the live system is a suspicious deviation from the expected number, it share the weakness of many other rootkit detection techniques, and that it is only effective if a clean baseline has been established. As successful as EPA is detecting most rookits, it was not long before the *Fu* rootkit was released, which avoids EPA detection. EPA uses debug registers to protect the instruction counter in the IDT, but the debug registers do not protect physical memory. Essentially, *Fu* uses DKOM to write directly to physical memory and bypass the debug registers. (Barbosa, 1)

3.3 Rootkit Detection Tools

The deeper the particular malware interfaces with the victim operating system the more difficult detection becomes, which is why user-mode rootkits generally require less sophisticated detection tools then kernel-mode rootkits. User-mode rootkits are most effectively detected using standard system reporting tools, and the most important point is to ensure you are using known clean binaries. It is a good idea to download the latest copies of all the tools you'll want for an IR CD, from a secure host of course, and either compile your own from source, or ensure that the binaries are digitally signed. The first tool in any such kit should be a network port usage monitor. There are many fine such programs available for the Windows platform that you should familiarize yourself with, such as *open ports* (http://www.diamondcs.com.au/openports/index.php?page=download) and *fport* (http://www.foundstone.com/knowledge/proddesc/fport.html), in addition to the native *netstat*. Try using them all to collaborate their results. If you find any unfamiliar or know-bad processes bound to a network socket, or a legitimate process running on an unusual port, than it is cause for concern, and may be indicative of a Trojan horse infection or other malware. In the case of discovering a known Trojan or rootkit listening on your system, you may be satisfied with the evidence from your port monitor. But in the case of a suspicious system process listening on an unusual port (e.g. ftp on ephemeral port only, not tcp/21), you will want to investigate the process in more detail without using the compromised system binaries. Luckily there are also plenty of tools for reporting all the execution details of a running process that are very useful for analyzing it's behavior. For example, *ProcessExplorer* (http://www.sysinternals.com/ntw2k/freeware/procexp.shtml) reports on all system objects a particular process calls, and provides extensive data on system drivers. The Sysinternals web site (http://www.sysinternals.com/ntw2k/utilities.shtml) has many other useful monitoring utilities besides *ProcessExplorer* such as *CPUMon*, *DiskMon*, *FileMon*, *PortMon*, *RegMon*, and *TokenMon*, that provide real-time views of specific Windows subsystems.

Because kernel-mode rootkits directly modify system calls, regardless of the source of the binaries you use on the local system, their results will conceal the rootkits presence. One category of tools that can detect most kernel-mode rootkits are system integrity checkers. Integrity checkers such as Tripwire (http://www.tripwire.com/products/servers/index.cfm), operate in a manner similar to WFP, creating an cryptographic hash of system files and comparing them to a know good version. The drawback to integrity checkers is that they're effectiveness is dependent on ensuring that the initial hash generation uses clean files, which must be established before any suspected incident. Otherwise, more advanced detection techniques must be employed. As mentioned above,

booting a suspect host off a CD OS distribution such as *Knoppix-STD* (http://www.knoppix-std.org/download.html) and *F.I.R.E* (http://sourceforge.net/project/showfiles.php?group_id=46038&package_id=56574&release_id=159330) can be effective in determining if the host has been compromised. Both examples are extremely specialized 'NIX distributions that provide an extensive array of security tools that should prove to be an invaluable additional to your burgeoning security toolkit. With NTFS support built in, you are able to comb the file system for artifacts indicative on an intrusion that would otherwise be hidden by the rootkit. For example, illicit FTP servers (pubstro's) may be serving data from obfuscated directories on your file system that are easily detected from a clean OS.

A new rootkit detection tool using EPA named *Patchfinder* (PF) has been developed to target modern kernel-mode rootkits, and can detect *Hacker Defender*, *APX*, *Vaniquish*, *He4Hook* and many other modern kernel-mode rootkits (Rutkowska, 2004). Unlike some of the other tools we've discussed so far, PF provides reliable results from the live system without requiring a reboot. However, like integrity checkers, it must be previously installed on the clean system in order to detect system compromise. PF performs statistical analysis against the clean baseline and reports detections to the *EventViewer*.

The latest tool in the never-ending security software arms race is *Klister* (http://www.rootkit.com/vault/joanna/klister-0.4.zip), the only toolset known to detect rootkits that employ DKOM, such as *Fu*. *Klister* reads the internal kernel thread list, which is used by the kernel dispatcher to allocate CPU time, to enumerate all threads and processes running on a system.

4 Conclusion

It is critical to never assume your system's security is impenetrable, and one should always perform diligent security practices within a paradigm of calculated risk management. The Internet should is an extremely hostile environment, and systems that are connected to it must be prepared to defend against the onslaught of Worms, Trojan horses and Viruses that infesting it today.

In this paper we have presented a technical overview of Windows rootkit technologies. Weather an attacker is looking to surreptitiously manipulate sensitive data, disrupt system services, or even destroy critical assets, the latest generation of Windows rootkits makes it very difficult to detect their existence. Because Windows rootkits have evolved into such sophisticated toolsets, they enable even the least knowledgeable of attackers to effectively conceal their presence on your system. It is imperative that we all act as responsible netizens and develop an understanding of the tools and techniques used by

attackers so that we can appropriately defend our hosts and help stem the flow of malware at large.

Appendix A: References

Barbosa, Edgar. "Avoiding Windows Rootkit Detection." February 2004. http://packetstormsecurity.org/papers/bypass/bypassEPA.pdf

Bobkiewicz, Bartosz. "Hidden Backdoors, Trojan Horses and Rootkit Tools in a Windows Environment." January 23, 2003. http://www.windowsecurity.com/articles/Hidden_Backdoors_Trojan_Horses_and_Rootkit_Tools_in_a_Windows_Environment.html.

Butler, Jamie. "Direct Kernel Object Manipulation." 2004. http://www.blackhat.com/presentations/win-usa-04/bh-win-04-butler.pdf.

Carvey, Harlan. "Data Hiding on a Live System." 2004. http://www.blackhat.com/presentations/win-usa-04/bh-win-04-carvey.ppt.

Casey, Eoghan. "Network traffic as a source of evidence: tool strengths, weaknesses, and future needs." Digital Investigation February 2004 1(1): 28-43.

Hoglund, Greg. "A *REAL* NT Rootkit, patching the NT Kernel." Phrack Magazine September 9, 1999 9(55): 5.

Kuepper, Brian. "What You Don't See On Your Hard Drive." April 4, 2002. http://www.sans.org/rr/papers/27/653.pdf

Microsoft. "Description of the Windows File Protection Feature." December 15, 2003. http://support.microsoft.com/default.aspx?scid=kb;EN-US;222193

Rutkowska, Joanna. "Advanced Windows 2000 Rootkit Detection (Execution Path Analysis." July 2003. http://www.rootkit.com/vault/joanna/windows_rootkit_detection_joanna.pdf

Rutkowska, Joanna. "Detecting Windows Server Compromises." November 6, 2003. http://www.hivercon.com/conf/archive/hc03/Rutkowska_Win32RookitDetection_HC2003.ppt

Rutkowska, Joanna. "Detecting Windows Server Compromises with Patchfinder 2." January 2004. http://www.rootkit.com/vault/joanna/rootkits_detection_with_patchfinder2.pdf