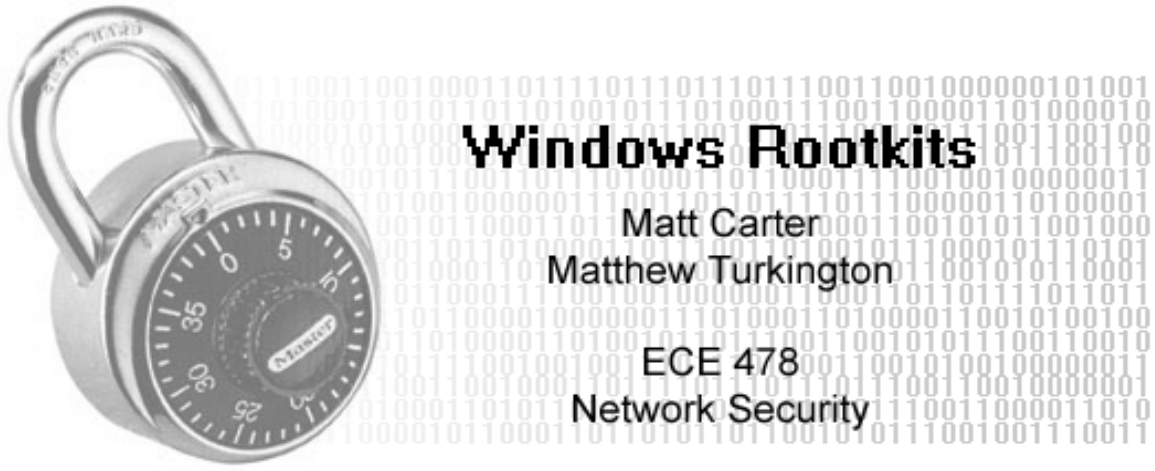


Windows Rootkits

Matt Carter
Matthew Turkington

ECE 478
Network Security

"*Hackers are using vastly more sophisticated techniques to secretly control the machines they've cracked, and experts say it's just the beginning*"

SecurityFocus' Kevin Poulsen

March 6, 2005
Oregon State University

Table of Contents

Introduction

The day started as any other day, network administrator Bob walked into the office and checked the outgoing traffic list, which shows the computers on the network which are using most bandwidth, and noticed something that wasn't there the day before. Alice's computer was at the top of the list, registering as having uploaded 8 GB of data over the weekend.

Bob went over to Alice and sat down at her computer to look for signs of compromise. There were no unrecognized processes running, no files out of place, no new user accounts, nothing that seemed out of place. He then went back to his office and ran **nmap** on Alice's computer to scan for open ports. The only open ports were the usual ones for communicating on the network.

Bob, finding nothing wrong, went about his day and decided not to worry about it anymore. Little did Bob know that he has just been tricked by a rootkit running on Alice's computer. Everything looked for had been hidden by the rootkit and Alice's computer had been turned into a FTP server and was uploading movies around the world all weekend.

Rootkits are becoming an ever increasing reality of computer security today. Where before a virus might drop a few annoying programs, a virus scanner could detect and remove them with ease. Now some viruses are coming with advanced rootkits which are

able to evade virus scanners, port scanners and even the most competent system administrators.

A Brief History

Rootkits have been around for quite a while. Before rootkits there were Trojan horses, exploit programs to elevate privileges, programs to clear log files, and other ways to hide the compromised state of a machine. These were bundled together to form what is known today as a rootkit.

A root kit is a set of tools used by an intruder after cracking[1] a computer system. These tools can help the attacker maintain their access to the system and use it for malicious purposes. They tools can also allow an attacker to easily access the system by opening a backdoor, installing a variety of Trojan horses, or installing a key logger. But by themselves rootkits are only to hide the existence of itself and other tools.

The first rootkit was created for Sun's Berkeley flavor of UNIX (SunOS 4) then later came rootkits for different distributions of Linux. According to Dave Dittrich:

> Linux Root Kit version 3 (lrk3), released in December of 1996, further added tcp wrapper trojans and enhanced the programs in the kit. This

[1] The act of breaking into a computer system; what a {cracker} does. Contrary to widespread myth, this does not usually involve some mysterious leap of hackerly brilliance, but rather persistence and the dogged repetition of a handful of fairly well-known tricks that exploit common weaknesses in the security of target systems. From **The Hackers' Dictionary of Computer Jargon**

> was the most common method of concealing activity and stealing passwords by sniffing on the new favorite target of intruders, x86 compatible PCs running Linux.[2]

A traditional UNIX rootkit consists of a set of modified binary files which can are used to replace the tools on the compromised system. For example, one such system could have the program “ps”, which will list running processes, with a modified one which will not list a particular backdoor that is currently running. Tools are also available which will make sure the rootkit binaries will have the same CRC checksum as the originals.

Modern rootkits for Microsoft Windows take a different approach in hiding themselves and other malicious programs. Instead of replacing binary files, they embed themselves in the system code and redirect API calls, incoming network traffic, or any other calls any programs or Windows itself can make. Similar to modifying “ps” in Linux, a rootkit can change the information that the Windows Task Manager is receiving from Windows so it will not list the running backdoor or keylogger.

[2] "Root Kits" and hiding files/directories/processes after a break in

Getting a Rootkit

A rootkit in and of itself does not possess the ability in install itself on a system. In order to be installed, an attacker must first gain root access to a system, then a rootkit can be installed, guaranteeing unrestricted future access to that system. Any vulnerability which results in the ability to install software can be utilized to install a rootkit. The three most common vulnerabilities for Windows are WINS, RPC, "Stupid User Attacks."

WINS (Windows Internet Naming Service) is a component of Windows which resolves host names to IP addresses, in a manner similar to DNS. Unlike DNS, WINS allows for a machine to be accessed explicitly without using a fully-qualified domain name. As part of the WINS host resolution process, a validation request executed, comparing the result from the potential host to what the server knows the host should be. In the most common of WINS exploits (such as MS04-045), a malformed validation packet is sent to the host which results in a buffer overflow. This buffer overflow in turn allows remote execution of code at the attacker's bidding.[3]

The **RPC** (Remote Procedure Call) service on Windows is an interface of the Distributed Component Object Model (DCOM). RPC is used to provide inter-process communication between distributed systems, allowing remote code execution seamlessly on remote systems. While this service is restricted to system-level access, it listens full-time on a standard TCP port. In an RPC attack (such as MS03-026), a malformed request is sent to

[3] Microsoft Security Bulletin MS04-045: Vulnerability in WINS Could Allow Remote Code Execution

the RPC server on a particular system which, in turn, instructs the system to execute the code under Local System privileges. The infamous Blaster worm was an RPC exploit.[4]

Finally, "Stupid User Attacks" are probably the most abundant. A vast majority of common users have no concept of basic computer security principles, and inadvertently put themselves at risk. Failure to create a password, let a lone a strong password, makes the job of an attacker substantially easier. Furthermore, with the prevalence of high-speed Internet connections, failure to secure file shares on an average desktop machine can quite easily result in a wide-open system.

In any of these cases, once a method of obtaining root access to a machine has been established, the attacker is free to install a rootkit ensuring future access. Oftentimes additional software will be installed in conjunction with the rootkit, such as FTP servers, IRC bots, or other tools making the compromised system "useful" to the attacker.

[4] Microsoft Security Bulletin MS03-026: Buffer Overrun In RPC Interface Could Allow Code Execution

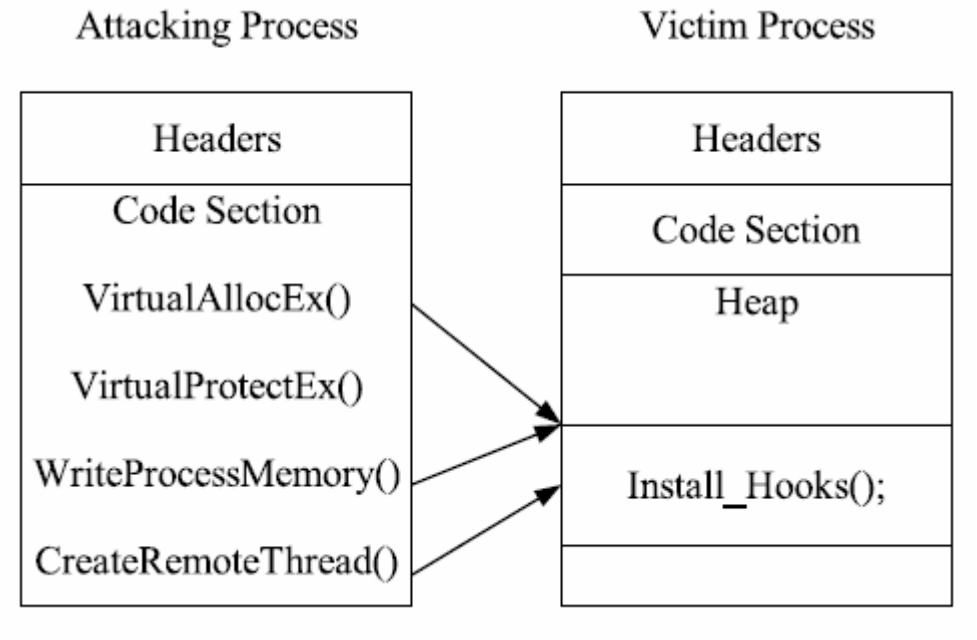
How do they work?

There many popular ways for rootkits to interfere with the normal Windows API. In this section I will explore these techniques in depth.

The first place to start modifying the API calls is within the user space layer in windows. The main idea of modifying the API is to execute the malicious code then continue executing the API so nothing looks suspicious. This method is known as hooking. If a malicious code was hooking into the Windows Logon API, it could intercept the logon password of a user in plaintext, record it, then pass it along to the real API which would initiate the logon.

One method of hooking into a process in Windows NT is called **Direct Memory Writing.** This method involves using the VirtualAllocEx() and WriteProcessMemory() API to do the dirty work. First the rootkit will use VirtualAllocEx() to allocate memory in the address space of another process, it then uses WriteProcessMemory() to inject any code it likes in the 'host' process. Then using CreateRemoteThread() the injected code is started and the thread is compromised.[5]

[5] Hide 'n' Seek? Anatomy of Stealth Malware



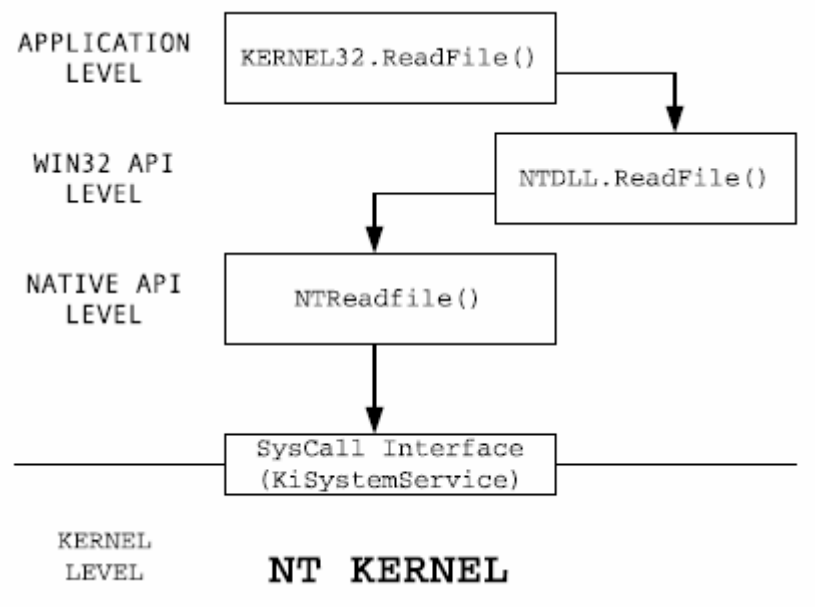
Direct Memory Writing [Hide 'n' Seek?]

There are more ways to modify API calls, such as **DLL Injection**, but newer, more advanced rootkits are moving away from accessing code in the user space and have moved on into modifying the kernel mode processes.

Any process that is running in kernel space has much more power at its disposal a process running in user mode. Hooking malicious code into kernel space processes has given rootkits the ability to affect every aspect of the compromised computer, from low level hard drive mappings, to the TCP/IP layer.

Windows NT (with that Windows 2000 and Windows XP) is built upon layers. The lowest layers are the kernel and the HAL (Hardware Abstraction Layer), followed by the Win32, OS/2 and POSIX subsystems. User applications get run under the Win32 layer,

which in turn calls the Win32 API which is a wrapper for the lower level Native NT API. The Native NT API is where the power is at; hooking into that layer lets you affect every layer above it. An interesting fact is that none of the Native NT API is documented by Microsoft, but dedicated researchers have reverse engineered about 90% of it.[6]



Example Win32 API Call [Hide 'n' Seek?]

Again, like user space, there are many ways for a process to hook into the kernel space. The preferred method is to change the pointer in the **System Service Descriptor Table.** Each process has an ID in this table and by knowing the ID of any process, you can hook into it. With the knowledge of the process ID you can store the location of the pointer (which points to what code to execute) then place your malicious process in the same spot in the SSDT. When the API calls for the process with the ID you just took, you can parse the arguments, pass them along to the location of the pointer you stored before then

[6] Hooking Windows NT System Services

retrieve the output from the API call and modify it. You can modify anything you want as long as it conforms to the expected output. The process could remove certain files, directories, ports or anything else from a listing before returning it to the above API.

Detection and Removal

Being as a key tenet of rootkit design is the ability for it to hide itself, detection and removal proves to be very difficult. Often the first signs of a rootkit come not from the kit itself, but from any of the various programs the attacker then executes on the compromised system. For example, a user may discover a FTP server full of pirated movies while completely missing the rootkit which allows the attacker continuous access to this service.

Oftentimes a good place to start is checking the system for abnormal activity which may indicate the presence of hidden software protected by a rootkit. Examining a listing of ports in use and bandwidth logs can reveal the presence of a hidden program. Further investigation often reveals a rootkit. Other abnormalities may include the inability to access certain registry keys and hidden folders in the file system. Typical antivirus and anti-spyware programs do not often detect rootkits, and in the rare case they do, are unable to remove them due to the system hooks.

Several programs have recently emerged which try and help expose the presence of rootkits. These utilities can be grouped into three main categories: those listing threads

apart from the API, those performing online API-less system scans, and those performing offline API-less system scans. While none of these methods of detection are 100% accurate all the time, they provide a good starting point (and in most cases are all you *can* do).

Tools such as **KLister** seek to expose rootkits by generating a list of running processes apart from the standard Windows API. This method is required because the hooks rootkits place in the Windows subsystem will cause them to be absent from any listing of processes generated by Windows directly. By accessing the internal kernel data structures in the scheduler core directly, KLister can return an accurate list of all running threads, from which a process list can be extrapolated. This list will show any hidden rootkit processes which are not displayed in the same listed generated via the Windows API.

The second method of rootkit detection is online API-less system scans. These tools, such as Sysinternals' **RootkitRevealer** and **Patchfinder2** operate by performing a typical system resource scan using the Windows API, then a low-level API-less scan. This scan is performed by directly accessing the FAT or NTFS file system, registry hives, and process lists without making any system calls to Windows. If the two lists don't match, this can be evidence of rootkits actively hiding files and processes on the system. It is possible for a rootkit to still hide from these tools, but they would have to intercept the utilities threads directly to prevent them from accessing hidden information.

Finally, the most reliable way of detecting rootkits is through an offline API-less scan. Microsoft Research is currently pursuing a project of this type named **Strider**. The goal of the Strider project is to provide a complete bootable "lite" environment into which a system can be booted. A complete system scan will then be performed, and the results compared to the results of a typical online scan. Because this offline operating system, although still a Windows derivative, is considered "clean," it will not have the system hooks present for the rootkit to use. The result will be an accurate list of files and registry keys present on that hard drive. When compared to a normal scan using the Windows API, a quick comparison will reveal any inconsistencies indicating the presence of a rootkit.

Removing rootkits is as complicated as detecting them. Even if you are able to locate the exact files and registry keys employed by the rootkit, because it has hooks in the system API which you must use to try and delete those files. This proves futile. The most effective removal technique is to simply remove the compromised hard drive and place it in another system. While another Windows box may be used, using another operating system, such as Linux, can increase your chances of success. Utilizing one of the above methods to determine the rootkit's files, it is then possible to remove them from the system from this "clean" environment.

Many tools have emerged claiming to remove rootkits. While these may work for specific variants of specific rootkits, due to the ever-chancing dynamic nature of the "computer underground," it is not realistic to expect these tools to be able to rescue a system every

time. To date, software manufacturers are lagging behind the rootkit makers with detection and removal. In much the same way antivirus companies lagged behind virus makers, it will take a high-profile exploit to raise public awareness and demand for effective rootkit detection and removal tools.

Prevention

Because rootkits are dependent upon existing vulnerabilities to enter a system, protecting against these vulnerabilities is the most practical way to prevent rootkit infection. Maintaining a current patch level is critical, as vulnerabilities are frequently discovered and subsequently patched. Although antivirus and anti-spyware tools do not target rootkits themselves, performing regular system scans with these tools will help to ensure rootkits do not have an easy way into the system. Effective management of shares and password complexity also greatly reduces the ease at which an attacker can compromise a system. Finally, network security tools, such as firewalls, can additionally remove the attacker from direct access to the system.

While rootkits run prevalent across the Internet, following basic computer security best practices will greatly reduce the likelihood of infection.

Appendix

DLL

Dynamic Link Library. A file of functions, compiled, linked, and saved separately from the processes that use them. Functions in DLLs can be used by more than one running process. The operating system maps the DLLs into the process's address space when the process is started up or while it is running. Dynamic link libraries are stored in files with the DLL file extension.
[http://www.gtscompanies.com/abbrev.html]

IRC

Internet Relay Chat - A protocol and a program type that allows participants to "chat" online in a live forum that usually centers around a common interest. IRC is the earliest form of online chat.
[http://www.getnetwise.org/glossary.php]

KeyLogger

A keylogger is a type of surveillance software that has the capability to record every keystroke you make to a log file (usually encrypted).
[http://www.webopedia.com/TERM/K/keylogger.html]

KLister

Klister is a POC which shows how to detect hidden processes. Although it is very simple tool, it is able to detect processes hidden by all known Windows rootkits.
[http://www.invisiblethings.org/tools.html]

NMAP

Nmap ("Network Mapper") is a free open source utility for network exploration or security auditing. It was designed to rapidly scan large networks, although it works fine against single hosts. [http://www.insecure.org/nmap]

Patchfinder2

Patchfinder (PF) is a sophisticated diagnostic utility designed to detected system libraries and kernel compromises. Its primary use is to check if the given machine has been attacked with some modern rootkits.
[http://www.invisiblethings.org/tools.html]

RPC

Remote procedure call. Much as its name suggests, this approach to concurrency works as follows: A process requests another process to do work by executing a remote procedure call, using a well-defined interface for the service to be performed. The processor executing the RPC starts a process to service the request, which performs the desired work, returns the result, and terminates.

Remote procedure call is a popular model for implementing distributed client-server computing environments. It is an alternative to interprocess communication (IPC) which allows remote systems to execute a set of procedures to share information.
[www.cise.ufl.edu/research/ParallelPatterns/PatternLanguage/Background/Glossary.htm]

RootkitRevealer

RootkitRevealer is an advanced patent-pending root kit detection utility. It runs on Windows NT 4 and higher and its output lists Registry and file system API discrepancies that may indicate the presence of a user-mode or kernel-mode rootkit.
[http://www.sysinternals.com/ntw2k/freeware/rootkitreveal.shtml]

Strider

Strider GhostBuster detects API-hiding rootkits by doing a *"cross-view diff"* between "the truth" and "the lie". It's not based on a known-bad signature, and it does not rely on a known-good state. It targets the fundamental weakness of hiding rootkits, and turns the hiding behavior into its own detection mechanism.
[http://research.microsoft.com/rootkit/]

WINS

Windows Internet Naming Service. A name resolution service that resolves Windows networking computer names to IP addresses in a routed environment. A server using this service handles name registrations, queries and releases.
[www.microsoft.com/technet/prodtechnol/ie/reskit/ie5/glossary.asp]

Works Cited

Dabak, Prasad. Hooking Windows NT System Services. -1 Oct. 1999. Windows IT Library. 5 Mar. 2005. <http://www.windowsitlibrary.com/Content/356/06/1.html>.

Dittrich, David. "Root Kits" and hiding files/directories/processes after a break in. 17 Apr. 2002. Unversity of Washington. 5 Mar. 2005. <http://staff.washington.edu/dittrich/misc/faqs/rootkits.faq>.

Erdelyi, Gergely. "Hide 'n' Seek? Anatomy of Stealth Malware." BlackHat Conference, Amsterdam, Netherlands, Europe. 19 Apr. 2004. [Speech]

Erdelyi, Gergely. "Hide 'n' Seek? Anatomy of Stealth Malware." F-Secure Coporation (2004): . [Paper]

Microsoft Corporation. “Microsoft Security Bulletin MS04-04” Microsoft TechNet (December 14, 2004). Microsoft TechNet. March 5, 2005 < http://www.microsoft.com/technet/security/Bulletin/MS04-045.mspx >

Microsoft Corporation. “Microsoft Security Bulletin MS03-026” Microsoft TechNet (September 10, 2003). Microsoft TechNet. March 5, 2005 < http://www.microsoft.com/technet/security/bulletin/MS03-026.mspx >

Microsoft Corporation. “MSR Strider Project” Microsoft Research (February 2005). Microsoft Research. March 5, 2005 < http://research.microsoft.com/sm/strider >