Beware of Geeks Bearing Gifts: A Windows NT Rootkit Explored

**Reproduced from www.rootkit.com*

FREDRIC J CIBELLI

Incident Handling and Hacker Exploit Practical

4 April 2001

Preface: Greg Hoglund has unveiled a surreal and frightening truth about the present computer security landscape; a working, true to definition rootkit for the most abundantly used operating system in the world, Microsoft Windows NT. This paper briefly explores the exploit's code, capabilities and techniques while discussing the impact it can have on computer systems and a society that depends on them. This is the first, true Trojan for the Windows NT platform.

Name: NTroot, most current version 0.43, www.rootkit.com, windows rootkit

Variants: None, yet, a one of a kind

Operating System: Windows NT/2000

Protocols/Services: Microsoft Windows NT Operating System kernel, registry/drivers, etc.

Brief Description: A Windows NT rootkit that can hide processes, hide files, perform Registry key hiding, keyboard sniffing, and redirect EXE files, all the while, providing a shell that allows an attacker to connect remotely. It is built as a kernel mode driver, running with full privileges on a machine.

Protocol Description:

The 'protocol' in this exploit is Microsoft Windows NT Operating System (a network-integrated operating system). This exploit has been accomplished for both Windows NT 4.0 and Windows NT 5.0 (2000).

Windows, which needs little introduction, has been the dominant operating system of personal computers for the latter half of the last century. Before NT, Unix and Novell NetWare were the dominant platforms for Local Area Networks that businesses used to communicate. Microsoft had developed a product called LAN Manager (LanMan), which was to become Windows NT. LanMan introduced an improved 'single' logon capability and a 'shares' concept, however, did not make any lead way into the market until Microsoft upgraded it (interweaving an operating system, and calling it Windows NT). Windows NT slowly grew into a dominant product, initially as a client, then making inroads into the server market. By 2001, NT had entrenched itself into the minds and operating environments of security focused solutions. Today, Windows NT commands a 92% stronghold on the PC operating system market. Furthermore, Microsoft has plans to integrating its Windows 9x OS with Windows NT. The fact that Microsoft's OS are so widely used, and relied upon, exponentially increases the importance of such a powerful and undermining exploit as NT Rootkit.

'Trusted Computing Base'

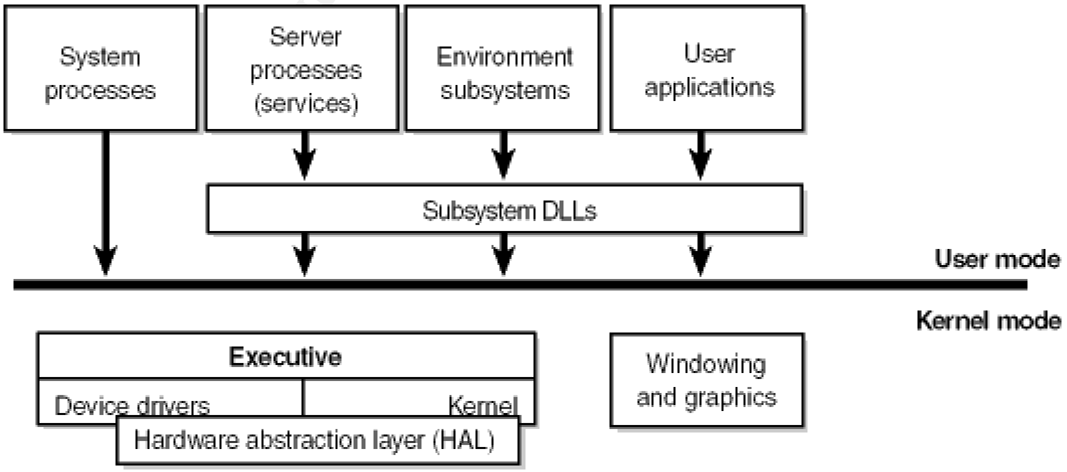
In Phrack Magazine, Greg Hoglund first introduced a form of NT rootkit. There he began his discussion by exploring the concept of 'trusted computing base.' I believe this is a valid and important point that needs to discussed here, because this single concept is the impetus behind rootkit's power:

The fear amongst information security professionals, and those who rely on information technology to survive, is an ability to undermine the 'Trusted Computing Base' (TCB) for single machines and the 'Network Trusted Computing Base' (NTCB) for networked systems. The TCB for a machine are those users and processes that can work at the operating system level, i.e. 'administrators' or 'superusers' or processes/software that have these privileges. These rights are known in Windows NT code as the security privilege flag SE_TCB_PRIVILEDGE. Those who operate with this can, pretty much, do anything to the operating system. NTCB is analogous to TCB, however, provides users the ability to do anything to a network of systems. Furthermore, it is a safe assumption that if one is compromised, so is the other. (These concepts and terminology were first defined in the National Computer Security Center's Rainbow Series)

User access to TCB is reserved for a selective community that is entrusted with the information and security of a system, i.e. system administrators, software engineers, etc. They are corporately tracked and scrutinized, because of the access power they have to alter, manipulate and destroy information. However, they are 'trusted'. What if this tightly held community had a leak? Decisions based on the information the system delivered could become detrimental. I will not delve into the damage this scenario can deliver, at this time, but the possibilities are as endless as the human imagination. As a current example, the FBI has recently implemented policy that requires system administrators to undergo the controversial and same lie detector tests that agents and other highly secure positions do (Verton). This demonstrates a corporate acknowledgement of the importance of the Trusted Computing Base.

'Quick Intro to Windows NT Architecture'

To understand some key points of Windows NT security, and how rootkit exploits them, we must begin with an introduction of NT architecture, and how the operating system is set up. Reference the diagram below for a barebones representation of the Windows NT/2000 architecture (Reproduced from Microsoft Press Online):



Starting from the bottom up, you have the hardware, which is your computer processor, physical memory, monitor, etc. Interacting with this equipment is the Hardware Abstraction Layer (HAL). This module provides a layer of abstraction, so Windows NT can run on different types of hardware. Next is the heart of the OS, the kernel. The kernel keeps all the processes in line, determines access control, management of every aspect of the OS, and so on. Windows NT also calls this part the 'executive layer;' it is slightly higher than the kernel, organizationally. Here in lies a key Windows NT component to this paper, the Security Reference Monitor, which will be discussed in full, later. Device drivers allow outside processes to run in Kernel mode, and effectively interact with the hardware (key aspect to rootkits deployment). Above that are the User mode modules that are used to interact with the user. These are not of importance when understanding rootkit's capabilities. Rootkit primarily exploits kernel level processes. A point to notice is the distinction of User mode and Kernel mode.

'User Mode vs. Kernel Mode'

A key strategy to building an operating system is a layered approach to security. Application of this strategy provides the capability to separate the 'Kernel' execution environment, where interaction with the hardware occurs, and the user's environment. Intel based x86 processors supports multiple operating modes, called rings. They range from 0-3, where 0 has full control of memory, or most privileged and 3 being the least privileged. Windows NT only uses ring 0 and ring 3, Kernel mode and User mode respectively.

The true distinction between Kernel mode and User mode is access to memory. Objects are defined in Windows NT as: every file, directory, synchronization object, process, thread, pipe, access token, Registry key, etc. Objects are placed into memory and are market by a 'privilege level.' However, anything in Kernel mode has access to all objects in memory and is trusted to play nice. Processes that run in Kernel mode are trusted because Microsoft, or a system administrator through the use of drivers, generated them. The trust of Kernel mode processes is a fundamental concept of rootkit's ability to undermine the TCB of Windows NT.

'Security Descriptors and Access Control Lists'

Now you may be asking how exactly does Windows NT create these concepts, and manage them? Me too. At the heart of Windows NT security are Access Control Lists (ACLs) and Security Descriptors (SDs). A SD is a list of attributes that Windows NT uses to determine the 'worthiness' or 'privilege level' (as discussed previously) of a process or thread to access objects. It contains a Security Identifier (SID) contains object owners, and other information about the process or thread. SIDs are number representations of accounts, groups, machines and domains. These tell the kernel what access you are allotted.

Now, ACLs act as a table or database to reference permissions assigned to an object. An ACL consists of Access Control Entries (ACEs). There are two types of ACLs discretionary ACLs (DACLs) and system ACLs (SACLs). DACLs define access permissions to the object they secure, and SACLs contain audit instructions for the system. For the purpose of this introduction, we will only concern ourselves with DACLs. ACLs are created, via an algorithm, every time an object is generated.

‘Security Reference Monitor’

Simply put, the Security Reference Monitor (SRM) determines if a process or user has access to the Kernel mode. It is part of the *executive* services or Kernel mode services available to all Windows NT processes. It is one of the most integral components of the NT operating system’s security architecture. SRM is the Windows NT component that implements the ACL, SDs and SIDs structures.

The SRM is the police that ensure processes have proper SDs to access objects. An administrator account may not have access to every object, but has the capability to take ownership and rewrite the DACL to allow access as desired. The SRM also manages security events, alerting the user whenever access has been denied (any user of Windows NT is familiar with this window).

A key point here is that the SRM does not manage or protect code running in Kernel mode. Processes running in kernel mode (as stated earlier) can directly modify an object without supervision of the SRM. Applying SRM type conditions and management to the kernel would cause serious performance degradation, and is deemed unnecessary as long as proper security is in place to ensure only the TCB has access to the kernel.

‘Driver Model’

One last key component of Windows NT is the driver model (reference architecture model in previous section). When a user-mode program wants to do something it calls an application-programming interface (API), which is passed to the Win32 module. From here it is passed to the Kernel Mode. There, a cloud of operating system services surrounds a driver. A request is made for an interaction with some hardware resource (i.e. read from a hard drive). The device driver then uses the hardware abstraction layer (HAL) (drivers operate in kernel mode, and therefore can talk directly to their hardware, however use the HAL to provide a level of abstraction, and hence portability, configurable, etc.) to process the request. This seems complicated, but the driver is just a part of the OS, in Kernel mode that has specific instructions on how to interact with a piece of hardware.

Description of Variants:

None, one of a kind project accomplished by Greg Hoglund and other very smart people.

However, this team has developed multiple ways to load the malicious code into the kernel, which is discussed under “How to protect against it,” “Integrity Protection Driver.” The signature of the exploit can be changed, simply by finding other weaknesses in Windows NT, making it hard to protect from at the OS abstraction.

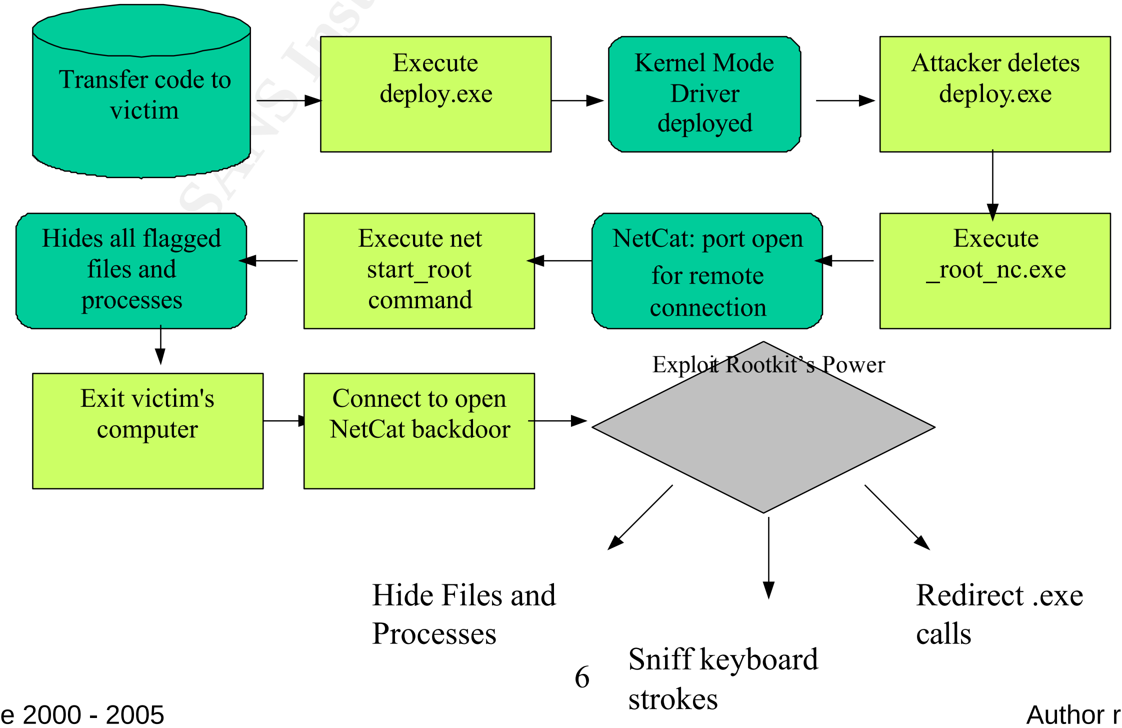
How the Exploit Works:

Greg Hoglund's trek to undermining the TCB of a Windows NT Operating System surrounds the executable code, known as the **ntoskrnl.exe**, which defines all the SRM functions. It is there that he begins the deconstruction of the TCB. As discussed earlier, SRMs have the responsibility of checking IDs before a user/process can have access to an object. What NT Rootkit has done is patch the SRM. Greg Hoglund learned that the SID is passed to the SRM every time a process tries to access an object. This is very restricting.

To escape these boundaries set by Windows NT security, NT Rootkit works to undermine a set of functions known as the Native Call Interface (NCI) (I will only brush over this part of the exploit due to the programming intensive tricks employed by very smart people, and offer only a quick introduction. For more information please visit www.rootkit.com or reference Greg Hoglund's paper "A *REAL* NT Rootkit…"). Through reverse engineering, Greg Hoglund was able to learn that functions can be added to the NCI. Next, he determined that a table used by **ntoskernel.exe** function used to determine other function calls could also be altered. Ultimately, what NT Rookit needed to accomplish is create a table with it's own functions and add it. Through proper memory allocation, making room for NT Rootkit's table of functions within the existing tables, Greg Hoglund developed a patch that ensured the new table was recognized. He was in. He patched the kernel giving any process access to any object in memory. From there, with a team of programmers, NT Rootkit was born.

Building upon Greg Hoglund's original patch of the kernel, many options were added. A recent and interesting function is the keyboard sniffer. It is implemented through a driver hook. It forces the victim's machine to redirect all keystrokes through itself first. This technique can be mirrored to implement network sniffers, and other detection devices.

Diagram:


Transfer code to victim
Execute deploy.exe
Kernel Mode Driver deployed
Attacker deletes deploy.exe
Execute _root_nc.exe
NetCat: port open for remote connection
Execute net start_root command
Hides all flagged files and processes
Exit victim's computer
Connect to open NetCat backdoor
Exploit Rootkit's Power
Hide Files and Processes
Sniff keyboard strokes
Redirect .exe calls

How to Use the Exploit:

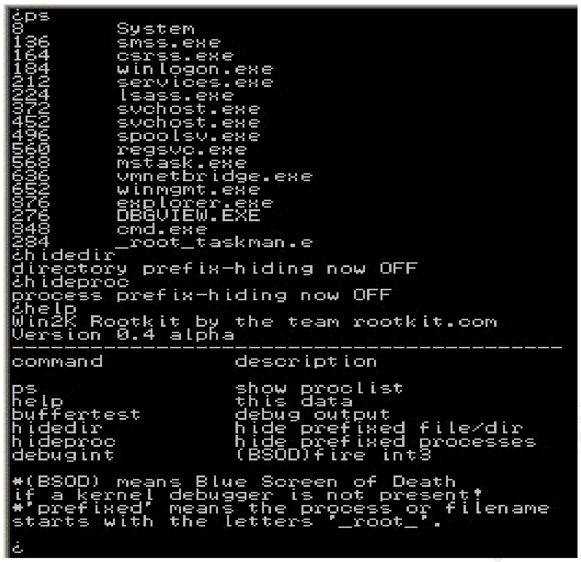
Rootkit is in a proof of concept stage, and has only been around for a few years. Much of the exploit's code has been used to prove theory, developed by Greg Hoglund. Rootkit is in no way a fully functional hacker tool that every script kiddy can deploy, like Sub Seven or Back Orifice. I have seen very little indication of a fully developed user interface being explored to exploit the capabilities of the rootkit through a user-friendly medium. However, it still can be a lethal weapon in a hacker's toolbox.

'Getting Rooted'

So you may be wondering how do I take down my company with this? (Or whatever deployment of this exploit you desire, I am not the moral police here) In order to load the device driver required to employ this exploit, one must first have administrative access (Windows NT requires administrative privileges to load drivers). I will not go into how to obtain this, as this is not an anatomy of an attack, just a description of one part or tool of an attack. There are numerous well-documented methods to obtain administrative access to a Windows NT machine. From this point, an attacker will need copy and execute the appropriate code.

NT Rootkit, version 0.43, contains a device driver, **_root_.sys** and an executable **deploy.exe**. To launch rootkit, an attacker must copy these files to the target computer, and then execute **deploy.exe**. **Deploy.exe** will install the driver and start NT rootkit automatically. At this point some may say that the victim has been 'rooted'. The attacker must delete **deploy.exe** so not to tip off the victim. However, all other traces of rootkit, as we will see, will become hidden. To start and stop the program, an attacker can execute the **net start_root_** and **net stop_root_** commands in the rootkit shell. Now it is time to see what this baby can do.

I mentioned the 'rootkit shell' in the previous paragraph. This is the user interface that wraps all the capabilities of the NT rootkit together. The following is a rootkit-shell command line screenshot taken from www.rootkit.com for NT rootkit version 0.4a (Hoglund):



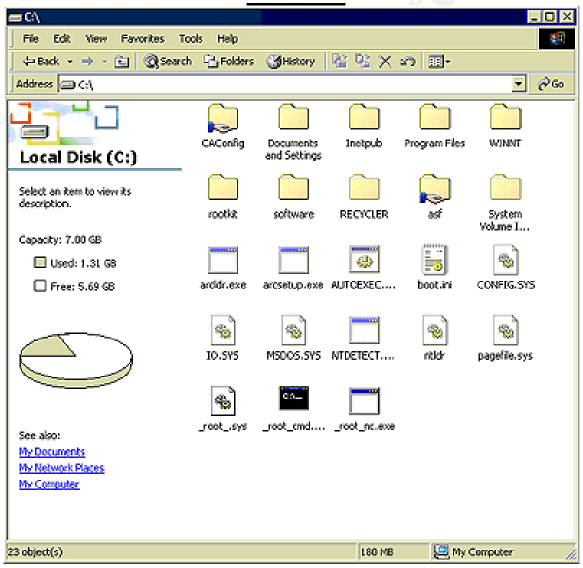
```
¿ps
8           System
136         smss.exe
164         csrss.exe
184         winlogon.exe
212         services.exe
224         lsass.exe
372         svchost.exe
452         svchost.exe
496         spoolsv.exe
560         regsvc.exe
568         mstask.exe
636         vmnetbridge.exe
652         winmgmt.exe
876         explorer.exe
276         DBGVIEW.EXE
848         cmd.exe
284        _root_taskman.e
¿hidedir
directory prefix-hiding now OFF
¿hideproc
process prefix-hiding now OFF
¿help
Win2K Rootkit by the team rootkit.com
Version 0.4 alpha
-----------------------------------------
command                 description

ps                      show proclist
help                    this data
buffertest              debug output
hidedir                 hide prefixed file/dir
hideproc                hide prefixed processes
debugint                (BSOD)fire int3

*(BSOD) means Blue Screen of Death
if a kernel debugger is not present!
*'prefixed' means the process or filename
starts with the letters '_root_'.

¿
```

**Reproduced from www.rootkit.com*

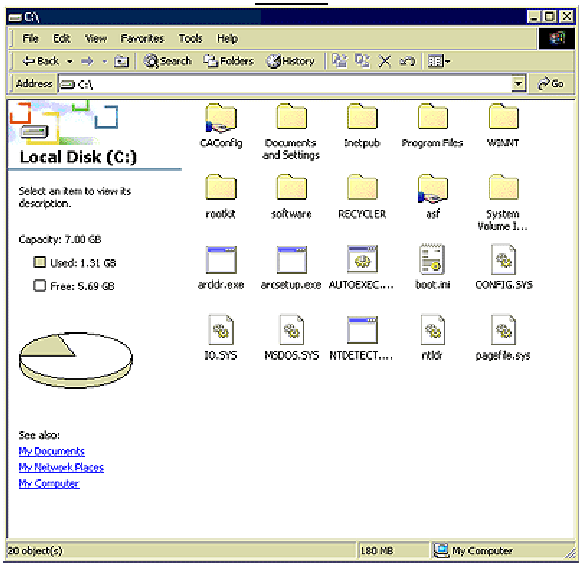
'Hiding Files'

One of NT Rootkit's key features is its capability to hide any file an attacker flags properly. Whenever rootkit is running, a file beginning with **_root_** is hidden from a directory listing through either the 'dir' command, or Windows Explorer. Any files an attacker does not want a user to discover can be renamed to meet rootkit's nomenclature. Furthermore, because the device driver, and other files that support rootkit, already begin with **_root_**, the exploit leaves no traceable evidence (that is if the attacker remembers to delete **deploy.exe**, as stated earlier). The following are before and after screenshots, taken from *Hackers' rootkit for NT* (Prosise):

Before



After



'Hiding Processes'

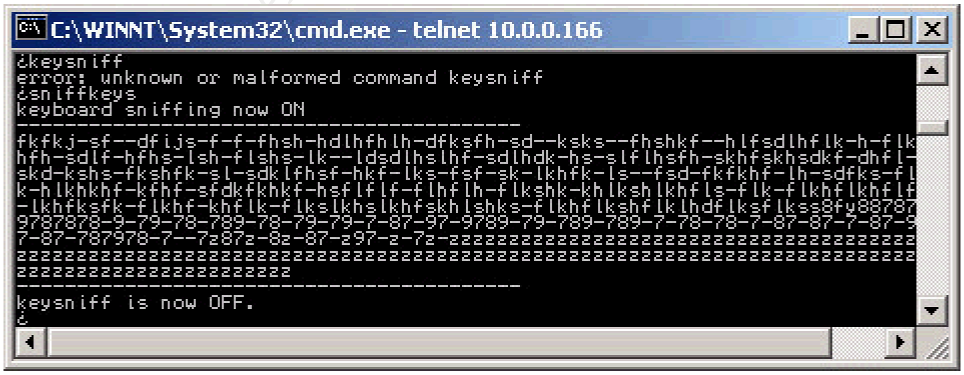
A process is an executing program, (synonymous with task) utilizing computer bookkeeping resources (Webopedia). When entering into a system, an attacker wants to hide his or her tracks. NT Rootkit provides the capability to remove executing programs or processes from any table, i.e. the NT Task Manager. Processes would be able to run in the background, utilize computer resources, but be hidden from a user or system administrator. NT Rootkit tags processes in the same format as files. Any executable program that begins with **_root_** will have its process number and generic indications removed. This is a very powerful tool, and fundamental to any rootkit. With this capability an attackers can operate in a full stealth mode, destroying and manipulating information at leisure.

'Remote Control'

In order to administer the rootkit shell remotely, a slightly modified Netcat solution has been developed. Netcat, known as the Swiss army knife of hacker tools, delivers a key capability to NT Rootkit by providing a remote access connection to its shell. An attacker utilizes the modified Netcat to connect to a TCP port (usually 99), which is bound to a command shell. This means when the attacker connects to the port on the victim's machine, the attacker will obtain a C:> prompt, existing within the victim's command shell. To hide this process, an attacker can rename Netcat as **_root_nc.exe**, thereby hiding the file and process when NT Rootkit is enabled. Even more noteworthy, the attacker has opened a fully hidden backdoor, unbeknown to the victim. Telnet capabilities have also been explored as another NT Rootkit remote access solution.

'Keyboard Sniffer'

Imagine an administrator, wakes up one morning, feeling good about the day, heads into work early to get a head start on the server migration for his company. He executes the trusty Ctrl-Alt-Delete key sequence so he or she can log onto the network then enters his or her login name and password. Just like that an attacker has an administrator name and password. NT Rootkit's first sniffer focuses on keyboard strokes. A recent addition (officially unveiled 3 February 2001), the keyboard sniffer provides the attacker the capability to grab any keystroke made on a victim's machine. Control of the keyboard sniffer comes from the shell, and can be turned on and off. The following is a screen shot taken from www.rootkit.com (Hoglund):



```
¿keysniff
error: unknown or malformed command keysniff
¿sniffkeys
keyboard sniffing now ON
------------------------------------------
fkfkj-sf--dfijs-f-f-fhsh-hdlhfhlh-dfksfh-sd--ksks--fhshkf--hlfsdlhflk-h-flk
hfh-sdlf-hfhs-lsh-flshs-lk--ldsdlhslhf-sdlhdk-hs-slflhsfh-skhfskhsdkf-dhfl-
skd-kshs-fkshfk-sl-sdklfhsf-hkf-lks-fsf-sk-lkhfk-ls--fsd-fkfkhf-lh-sdfks-fl
k-hlkhkhf-kfhf-sfdkfkhkf-hsflflf-flhflh-flkshk-khlkshlkhfls-flk-flkhflkhflf
-lkhfksfk-flkhf-khflk-flkslkhslkhfskhlshks-flkhflkshflklhdflksflkss8fy88787
9787878-9-79-78-789-78-79-79-7-87-97-9789-79-789-789-7-78-78-7-87-87-7-87-9
7-87-787978-7--7z87z-8z-87-z97-z-7z-zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz
zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz
zzzzzzzzzzzzzzzzzzzzzzz
------------------------------------------
keysniff is now OFF.
¿
```

**Reproduced from www.rootkit.com*

‘Other Capabilities and the Future’

Other capabilities of NT Rootkit range from the option to deliver a debug interrupt, in turn causing a blue screen of death on a victim's machine, to the ability to hide registry entries. However, one of the most powerful capability of NT Rootkit is the doors it opened. Programmers now can develop code, built upon rootkit's core theory and design that can truly exploit a Window NT box. For example, the keyboard sniffer developed a technique that can be ported to other peripherals, USB, Ethernet network connections, etc. A capability to exercise a sniffer on a network connection, for example, brings the scope of this exploit out of the Operating System of one machine and into LANs, and from there, who knows. A note to repeat here is that this exploit is only in its proof of concept stage; many more smart minds can build upon this, making for some really nasty stuff.

Signature of the Attack:

Because of the stealth focus of this exploit, no auditing devices can detect NT Rootkit. There is no visible disruption to an operating system that can be detected, easily. If events begin to occur that user or a system administrator feels out of place, this may be the first indication of a rootkit attack. From there in-depth investigating must occur to determine if NT Rootkit has rooted your system. Some of those techniques are discussed in the following section.

How to Protect Against It:

To protect against NT Rootkit, successfully, security professionals should focus their efforts on defending against and preventing from attackers gaining access to their machines. As we know, administrator rights are necessary to load the rootkit driver, and once this occurs, it becomes exponentially more difficult to detect and defend against any malicious events. This said, if NT Rootkit has rooted you, all is not lost. However, the TCB has been violated and decisions must be made accordingly.

'Finding the Hidden Files?'

First you may be wondering if there is a way to uncover hidden files and processes. The simple answer is no. Using other executable programs other than ‘dir’ or Task Manager to list files and processes, respectively, will not work. Even perfect, uncorrupted code of programs such as POSIX compliant **ls.exe** and **pulist.exe** would not reveal objects flagged **_root_** (Prosise). Chris Prosise found that by mapping a share to the suspected victim with administrative rights, all hidden files flagged **_root_** could be viewed and copied. However, this is not a common detection technique, but one that allows you to obtain information, once you believe you have been rooted. Also, look for the obvious **deploy.exe**, **_root_.sys** and other programs that an attacker may have left out in the open.

'Backdoor Detection'

What about detecting a back door? The netstat command that comes with windows will detect all open ports. One in particular to notice is port 99. If it is open, most likely, Netcat may be running on your system. Netstat will not tie the open port to a program, so we can only speculate. However, **fport** and **insider** will map the open port to a program (Prosise). If a port is open with Netcat as the associated program, we have indicated a backdoor has been loaded on your machine, but is there a rootkit associated?

Loading of the NT Rootkit, via the driver, will not create a log if successful, only unsuccessful rootkit installs show up. Also, do not look for any other information in NT auditing systems because rootkit by passes them all. Remember it operates within the freedom of Kernel mode processes.

‘Integrity Protection Driver’

Another thread of discussion occurring on Bugtraq, is concerned with solutions that protect from rootkits by defending against malicious drivers being loaded. One such product, Integrity Protection Driver (IPD) from Pedestal Software protects Windows from being victim to these malicious drivers. However, such protection is analogous to viruses, and such anti-virus software. Anti-virus software protects you from ‘knowns’ and so does IPD. Greg Hoglund argues that there are multiple ways to load code into the kernel, other than a driver, making such a security solution that fixes only a ‘known’ hole. A true defense against such NT Rootkits would be an OS structure overhaul. Furthermore, many other points of entry have been demonstrated that undermines the reliance on driver-loaded code into the kernel. IPD has been argued that it *should* be treated as an anti-virus program, in which updates are required to protect against this certain type of malicious logic.

Outside of directly protecting you from an NT Rootkit, security professionals should: incorporate code check sums; programs such as Tripwire; ensure they have well documented redeployment software; and other good security practices that will minimize the effect and damage an NT Rootkit can do.

Source Code/Pseudo Code:

Updated code is available at www.rootkit.com and the latest build will be submitted with this paper. It is not compiled and requires Microsoft Driver Development Kit to generate the exploit from the attached code.

Conclusion:

NT Rootkit's proof of concept, in my opinion, is complete. Greg Hoglund has accomplished something that will be the catalyst for many others. However lethal this exploit may be, defending your network and computer systems relies heavily on techniques outside this paper. Ensuring an attacker does not gain administrative access to any of your systems should be the primary focus of security professionals. NT Rootkit only provides hackers a way to truly manipulate and destroy your information. Techniques such as perimeter defense, intrusion detection, well-defined and enforced user policies, etc (all that is taught at SANs and other security communities) will ultimately be the key to defending against NT Rootkit. What this exploit will evolve into is unknown. However, the possibilities are frightening. How far it has come since inception is a tribute to how fast the information security field is moving. Furthermore, its impact is already being felt.

Greg Hoglund and his team have demonstrated to the computer security world a dark and powerful exploit that undermines Windows NT security. They have forced a necessary paradigm shift in Windows Security theory. Greg Hoglund has publicly expressed that the only true way to protect your information from the likes of NT Rootkit, is for Microsoft to rethink their security model. I hope such projects as this do have the impetus and influence to change such behemoths as Microsoft, so that information security continues to evolve, forward, keeping information safe, at least for a little while…

Bibliography

"Inside Windows NT, Second Edition." Microsoft Press Online. <http://mspress.microsoft.com/prod/books/sampchap/1312a.htm>

Hoglund, Greg. "A *REAL* NT Rootkit, Patching the NT Kernel." Phrack Magazine 9 Sep 1999.

Hoglund, Greg. "Loading Rootkit using SystemLoadAndCallImage." BugTraq 29 Aug. 2000.

National Computer Security Center. A Guide to Understanding Discretionary Access Control in Trusted Systems. Maryland, National Computer Security Center, 1987.

Prosise, Chris, Shah, Saumil Udayan. "Hackers' rootkit for NT." CNET.com. 22 Feb. 2001. <http://ciscom.cnet.com/>

Prosise, Chris, Shah, Saumil Udayan. "Stop Windows hackers." CNET.com. 8 Mar. 2001. <http://ciscom.cnet.com/>

Scambray, Joel, Stuart McClure, George Kurtz. Hacking Exposed Second Edition. McGraw-Hill, 2001

Schmidt, Jeff. Microsoft Windows 2000 Security Handbook. Que Corporation, 2000.

Schultz, E. Eugene. Windows NT/2000 Network Security. Macmillan Technical Publishing, 2000

Verton, Dan. "FBI to use lie detectors on its IT workers." CNN.com. 26 Mar 2001< www.cnn.com>

<www.webopedia.com>