Linnæus University
School of Computer Science, Physics and Mathematics

Degree Project

Rootkits

Jie Li & Yuting Lu
2010-09-04
Subject: Computer Science
Level: Bachelor
Course code: 2DV00E

Linnæus University
School of Computer Science, Physics and Mathematics

SE-351 95 Växjö / SE-391 82 Kalmar
Tel +46-772-28 80 00
dfm@lnu.se
Lnu.se

Abstract :

The kernel system of Windows is more thoroughly exposed to people. So, the kernel-level Rootkits techniques are now laid on greater emphasis. It is very important to maintain the security of computers and to conduct an in-depth research on the operational mechanism by using kernel-level Rootkits in hiding its traces. Since the involved core techniques are beginning to catch on nowadays, we should analyze some new key techniques employed for application of Rootkits, discuss the specific methods and propose a set of defense strategy for computer security.

Key words: Windows Rootkits; driver; hook; kernel; hide; detection

Content

"Rootkits are extremely powerful and are the next wave of attack technology. Like other types of malicious code, rootkits thrive on stealth. They hide away from standard system observers, employing hooks, trampolines, and patches to get their work done. "

—Gary McGraw, Ph.D., CTO, Cigital, coauthor of Exploiting Software (2004) and Building Secure Software (2002), both from Addison-Wesley

1. Introduction

Literally, a rootkit is a "kit" containing small and useful programs that allow an attacker to have access to "root," the most powerful user on a computer.

Rootkit is a kind of software or software combination whose major function is to hide other programs' traveling schedule .In other words, a rootkit is a set of programs and code that allows a permanent and undetectable presence on a computer. In general, Rootkit may be also regard as a technology.

1.1 Background

Rootkit was used in the good intentions in earlier time.

You can imagine that, if some day you backup files, you will save a number of important system information in the directory. You certainly do not want to delete them accidentally. Therefore, rootkit can properly hide the files you want to protect.

Also, if a system crashes, rootkits can help it quickly resume and recover from errors.

What makes rootkits significant now?

With the development of anti-spyware program , the Internet users become more aware of the threat to them. Thus, it is harder for the malware to spread .So now they are turning to rootkits for a new way [1].

What is more, rootkits are different from all the previous of this kind of insidious applications. They are virtually undetectable by current anti-virus scanners and spyware. Most of the technology and tricks employed by a rootkit are designed to hide data and code on a system. For example, many rootkits can hide files and directories [1].

1.2 Problem Formulation

In recent years, with deep-going development of network applications, network security seems to face enormous challenges. Especially, the rampant Trojan horses and Back door on the network during last couple of years, they tend to trap users' systems into more and more dangerous situation.

The most outstanding malware is Rootkit. It is a mixture of Trojan horse and back door. And because of the extensive application of Windows operating system, it is meaningful to research Windows Rootkit.

In order to prevent the impact of such malware procedure, it is essential for us to learn the principle of Rootkits. Thus, we can take specific measures to protect our personal computers.

Our main problem is to identify the mechanisms of Rootkits and the means for anti-Rootkit software. To reach this we will make a description of the principles behind a Rootkit, in particular the hiding technology used by this type of malware. A simple Rootkit will also be implementation to get deeper understanding of how they function. We also will give some suggestion on prevention.

1.3 Limitations

Our research is limited to Windows operating system.

1.4 Knowledge Required

Good knowledge of Operating System. There will be a lot of terms and something that runs in the kernel mode.

It is better if you know some skills of computer and network security.

1.5 Method

--Survey text

Read materials on Rootkits. We will read some materials on rootkits and try to find the most common way that the hackers use to hide rootkits. And we will also read some materials on security and find some methods to detect rootkits.

-- Analysis

Learn the principle. After we find the methods that are used to hide rootkits, we will try to learn its principles and analyze how they work. We also learn the principle of the security, state how it detects rootkits.

--Development

Write an implementation. In the chapter of implementation, we will write our own rootkits. We will use the principles that we learns. After we finish our rootkits, we will test it.

--Discussion

Search more ways to solve the problem. In this part we will discuss the benefits of different methods. And then we will search more ways to solve the problems.

1.6 Report structure

In the report we will give a brief introduction to rootkits first. And then we will state the principles of rootkis. After that we will analyze the hiding technique of rootkits. In the implement, we will write our rootkits and test it. Then we will list some ways to detect rootkits, write and analyze it. After finishing the targets above, we will have a discussion and search more ways to solve the problems. Finally we will have a conclusion.

2. Rootkits Principle

In this chapter, we will state the rootkits principles. It includes what is a rootkit, how rootkit works, kernel-level rootkit, features of hiding and techniques of hiding.

2.1 Trojaned System Commands

Before explaining what a rootkit is, we should know what the trojaned system commands are?

Trojaned system commands can be translated as "Trojan horse" (or, Trojan system instructions).

We believe that everybody should know the story of The Trojan War!

Seemingly, it always disguises as a normal procedure. But in fact, it secretly replaces the normal procedure and leaves the back door in order that you can secretly control the operation of the host, or even destroy the programs. That is to say, it is a Trojan horse program, commonly known as: Backdoor program (backdoor), or Trojan program (Trojan).

When there hides such a program in the system, we call it: in the Trojan.

The sources of Trojan, can be commonly divided into the following categories [10]:

a) The system has been invaded.
b) Infected with worms,
c) The implementation of programs of unknown origin.

In terms of the system invasion, most of the hackers will not make an immediate and obvious damage on invading the host.

The so-called rootkit, is that someone organizes these commonly-used Trojan horses and makes a suite of programs, to facilitate the cracker compile and install the Trojan program successfully in invasion.

There are many different types of rootkits. Usually, the Trojan program included in the rootkit distributes in the form of original program code.

Many of these programs migrate from the early BSD UNIX system step by step. Therefore, the trace of the rootkits not only exists in almost of the machine platforms, but they have diverse and multifarious patterns as well.

2.2 A Rootkit Is Not a Virus [1]

A virus program can spread automatically. By contrast, the rootkit does not make copies of itself, and it does not have its own mind. A rootkit is under full control of a human attacker, while a virus is not.

In most cases, it would be dangerous for an attacker to use a virus when she requires stealth and subversion. Viruses usually copy themselves. They have the possibility to damage the computer, data, or even the whole system. So they are easy to be detected by scanners. It is difficult to make self-control because of these features. Also, viruses are noisy.

But a rootkit enables an attacker to stay in complete control. Some kinds of attacking operations require rather strict controls, and using a virus would simply be out of the question.

2.3 How Rootkits Work?

File Scanning - One way that virus and spyware scanners find malware (usually used to describe any types of virus, trojan, spyware, etc.) is to scan your hard disk directory , retrieving the name of each file, opening that file, and searching its contents for known signatures that match the virus or spyware database in the scanner software.

Rootkits intercept all requests to file directories and delete the names of their own files from returned lists. Therefore, it comes undetectable.

2.4 Kernel-level Rootkit

Since we want to use Rootkit, it is necessary to know its principle and have a perceptual awareness of it.

Generally speaking, there are two ways to implement kernel-level rootkit. They are changing the system's data structure and changing the kernel execution path.

There are not too many rootkits that change their system data structures. FU_Rootkit [1] can be considered one type of them. It will hide the process by removing the object on PsActiveProcessList chain in nucleus. On the other hand, changing the kernel execution path is implemented by modifying or adding instructions within nuclear or system DLL (Dynamic Link Library).

a. Hiding process, directory, file, registry

To hide the process, directory, file, registry, they must intercept operating system from calling the function.

But they are not able to be modified freely, some relevant function should be written, and then modify the SYSCALL table to point to its function. Note that, in order to modify SYSCALL table, it must be in Ring0 (Kernel mode) state.

System Call Symbol	Windows NT				Windows 2000					Windows XP			Windows 2003 Server		Vista
	SP3	SP4	SP5	SP6	SP0	SP1	SP2	SP3	SP4	SP0	SP1	SP2	SP0	SP1	SP0
NtAcceptConnectPort	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000	0x0000
NtAccessCheck	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001	0x0001
NtAccessCheckAndAuditAlarm	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002	0x0002
NtAccessCheckByType					0x0003	0x0003	0x0003	0x0003	0x0003	0x0003	0x0003	0x0003	0x0003	0x0003	0x0003
NtAccessCheckByTypeAndAuditAlarm					0x0004	0x0004	0x0004	0x0004	0x0004	0x0004	0x0004	0x0004	0x0004	0x0004	0x0004
NtAccessCheckByTypeResultList					0x0005	0x0005	0x0005	0x0005	0x0005	0x0005	0x0005	0x0005	0x0005	0x0005	0x0005
NtAccessCheckByTypeResultListAndAuditAlarm					0x0006	0x0006	0x0006	0x0006	0x0006	0x0006	0x0006	0x0006	0x0006	0x0006	0x0006
NtAccessCheckByTypeResultListAndAuditAlarmByHandle					0x0007	0x0007	0x0007	0x0007	0x0007	0x0007	0x0007	0x0007	0x0007	0x0007	0x0007
NtAddAtom	0x0003	0x0003	0x0003	0x0003	0x0008	0x0008	0x0008	0x0008	0x0008	0x0008	0x0008	0x0008	0x0008	0x0008	0x0008
NtAddBootEntry										0x0009	0x0009	0x0009	0x0009	0x0009	0x0009
NtAddDriverEntry													0x000a	0x000a	0x000a
NtAdjustGroupsToken	0x0004	0x0004	0x0004	0x0004	0x0009	0x0009	0x0009	0x0009	0x0009	0x000a	0x000a	0x000a	0x000b	0x000b	0x000b
NtAdjustPrivilegesToken	0x0005	0x0005	0x0005	0x0005	0x000a	0x000a	0x000a	0x000a	0x000a	0x000b	0x000b	0x000b	0x000c	0x000c	0x000c
NtAlertResumeThread	0x0006	0x0006	0x0006	0x0006	0x000b	0x000b	0x000b	0x000b	0x000b	0x000c	0x000c	0x000c	0x000d	0x000d	0x000d
NtAlertThread	0x0007	0x0007	0x0007	0x0007	0x000c	0x000c	0x000c	0x000c	0x000c	0x000d	0x000d	0x000d	0x000e	0x000e	0x000e
NtAllocateLocallyUniqueId	0x0008	0x0008	0x0008	0x0008	0x000d	0x000d	0x000d	0x000d	0x000d	0x000e	0x000e	0x000e	0x000f	0x000f	0x000f
NtAllocateUserPhysicalPages					0x000e	0x000e	0x000e	0x000e	0x000e	0x000f	0x000f	0x000f	0x0010	0x0010	0x0010
NtAllocateUuids	0x0009	0x0009	0x0009	0x0009	0x000f	0x000f	0x000f	0x000f	0x000f	0x0010	0x0010	0x0010	0x0011	0x0011	0x0011
NtAllocateVirtualMemory	0x000a	0x000a	0x000a	0x000a	0x0010	0x0010	0x0010	0x0010	0x0010	0x0011	0x0011	0x0011	0x0012	0x0012	0x0012
NtAlpcAcceptConnectPort															0x0013
NtAlpcCancelMessage															0x0014
NtAlpcConnectPort															0x0015
NtAlpcCreatePort															0x0016
NtAlpcCreatePortSection															0x0017

Figure 2.1 SYSCALL table

SYSCALL table is a table that contains system functions. These functions will be called by OS or applications. Figure 2.1 is part of the SYSCALL table. The SYSCALL table is very important to rootkits. Some rootkits use this table to hide themselves. We will state how they work below.

b. Hiding connections and ports

Hiding connections and ports will use the technology of TDI query [1].

TDI refers to Transport Driver Interface. In Figure 2.2, you can see pros and cons of using TDI

Approach	PRO	CON
TDI	Allows you to have an interface very similar to sockets—which will be easier for many programmers Uses the local host TCP/IP stack and thus avoids issues with multiple IP or MAC addresses	It is more likely to be captured by desktop firewall software

Figure 2.2 TDI

In the TDI, there would be two equipments, \ \ Device \ \ TCP and \ \ Device \ \ UDP.

We can write code to enable users to Ring0 state and then modify the data in the device, in order to hide connections and ports.

2.5 Features of Hiding

In recent years, there is a great development in the protection technology of network security. The latest desktop anti-virus software and firewall software play an important role in protecting the target host from attacking by malicious code like Trojans, worms or leakage attacking procedures.

However, malicious code based on Rootkits technology in kernel-level malicious code will hide itself by virtue of the relevant underlying technology. Thus, in large part, it could escape the killing of firewall or antivirus software blocking, which does a grave threat on computer security.

Rootkit is a kind of software tool for attackers to control the target computer as system administrator. By using the tool, we can achieve a long-term, stable and a hidden control over the target computer.

This definition highlights the feature of hiding. Here hiding refers to that:

a. to evade the detection of anti-virus software
b. network communication through the general Firewall in the target system
c. to avoid leaving the log record in the target host

According to the working environment of Rootkits, it can be divided into user-level rookits and kernel-level rookits.

Kernel-level Rootkits refer to using driver or other relevant technology to attack into the Windows operating system kernel. And, through its own tamper of kernel-related structure or object in Windows operating system, achieving hiding and the execution of malicious code loading.

The following will present the typical process of kernel-level Rootkits Procedures, such as He4hook, NtRootkits etc. We will also make deep-going analysis of its hiding technology from self-hiding, communication hiding and other aspects of kernel-level rootkits on the target system.

2.6 Techniques of Hiding

As the kernel-level rootkits get into the operating system kernel through the carrier of driver.

The primary task of hiding technology must be hiding driver loading and the achievement of self-hiding of the driver procedures in the target system. On the other hand, it will try to hide memory.

2.6.1 Rootkits hide loading in the registry

Kernel-level rootkits exist in the form of drivers in target systems. Rootkits then bound to consider how to cover the load and how to hide in the registry afterwards.

In this regard, typical NtRootkits, He4hook have different implementations. There are two specific methods, such as modifying the system registry and the use of ZwSetSystemlnformation function to load the driver.

Rootkit now loads into kernel memory using a single interrupt call - an NT system call known as ZwSetSystemInformation(). Using this call we cause the Rootkit to be immediately loaded into memory and activated.

Modify registry

The Windows operating system has similar treatment mechanisms to services and drivers. Therefore, when the driver which contains the code of the kernel-level rootkits is going to run at the attacked machine, the corresponding content will also be registered to the registry key---
HKEY-LOCAL-MACHINE\SYSTEM\CurrentControlSet\Services [2].

Then, we think of the practice of modifying the data structure relevant to the registry in the system. So that the existence of rootkits cannot be found in the registry or the list of services.

But in practice, this method has a very obvious flaw: When the system starts and re-load all the services, the changes we have done before may easily be detected or corrected, thus, rootkits are very likely to be exposed.

Load driver by using function ZwSetSystemlnformation

Directly modifying the registry can easily be found, which obviously cannot meet the kernel-level.
rootkits' hiding requirements. Use ZwSetSystemlnformation (SystemLoadAndCallimage) function, and you can load driver directly without adding any registry entry, making the loading more subtle.

Rootkits use ZwSetSystemInformation, the timing and location of the loading is very important in the hiding.

If you make an application or service load a driver randomly, it is easy for the system administrator to find it.

If you use the application or thread to load, which uses a system-recognized and sufficiently-permitted driver to access, the loading movements could be more subtle. For instance, rootkits can use self-start service or application as a carrier of loading. Under normal circumstances, this type of service or application can not only call the driver, but also have sufficient permissions to use all the procedures and those dynamic link library associated with them in the user mode. It is a very favorable loading for rootkits.

What to do next is to modify it, so that it can hide the traces of the implementation and loading as well as loading rootkits.

We need to cover two aspects of work: hiding the changed process, and hiding the existence of original file. Specific method is:

Firstly, write a "null driver", which is to build a framework for a driver so that it can run in the driver mode.

In computing, a device driver or software driver is a computer program allowing

higher-level computer programs to interact with a hardware device. Here, driver is only a frame. Inside, it is Rootkits' code.

And then, place in the empty drive the Trojans in order to produce a Trojans running as a driver.

Based on this consideration, if the trace of the modifying to the driver can be hidden, you can replace any content of driver which is regarded completely safe in operating system (of course, it includes all of those drivers running in kernel mode).

After completing the work above, put rootkits inside, copy itself into memory before it starts, and then you can open the thread to carry out other work.

2.6.2 Memory hiding

When Rootkits are successfully loaded into the target system, they will be executed in memory.

The space of physical memory is limited, but Windows operating system provide users with four G's virtual memory space.

When the physical memory cannot meet user's requirements, some data that need not be dealt with immediately in memory will be written into hard disk temporarily. If the exchanged data contains Rootkits, it is easy to be found. So, the operating system is not allowed to exchange Rootkits code with hard disk data. In other words.

How to make Rootkits "Invisible" in memory? Then, the memory that stores Rootkits program must be anonymous and non-paged memory. Because it is very difficult to find a hiding driver among a large heap of anonymous memory paging. What is more, the content of non-paged memory will not be exchanged to the hard disk [2]. If Rootkits modify a kernel driver, the hiding will be better.

For example, with direct interception of certain low-level drivers' dispatch routine of IRPs (I/O Request Packets: handle several types of requests, such as reads, writes, and queries.).

Rootkits start work and make use of these two functions: IoQueueWorkItem and KelnsertQueueDpc.In this way, they can work without opening any routine of task manager, and greatly improve their feature of hiding.

After completing the work above, Rootkits will necessarily unload the driver where they are located, and then re-load the original driver and resume its normal function. So, we can avoid the possibility that some features of the system do not work or other problems occur that make them found by the system administrator. Also, the original drive must also be hidden in advance, or it is more likely to cause suspicion of the administrator because there are two same drivers in the system. The simplest way is, put the original driver source code and Rootkits program source code together (in anonymous memory), the procedures follow in the rear of Rootkits and execute immediately after the execution of Rootkits.

2.6.3 Rootkits hide files

Rootkits' feature of hiding in the file system refers to hiding I / O operation of the file.Mainstream of Rootkits use three methods: I / O reading and writing operation at

the system bottom,writing the file system to filter driver, direct interception routines of file system driver.

The first method is to let Rootkits have I / O operation at the bottom of the system. Then they are less likely to be found. Generally because programmers consider it is most reliable and secure to read or write the disk at the bottom part. But this method requires process of all formats of the disk file. It is feasible for FAT, but for NTFS, it is too difficult.

The second method is to write filter driver at upper file system driver. This method can be very convenient to monitor the reading and writing of files on the disk, thus having reading and writing control over the disk. The essence is, to intercept some important functions of the original file system driver by this layer of filter driver, and achieve monitoring and operations of reading and writing by use of Hook to add the function code.

The third method is to direct intercept the dispatched routines of file system driver. Compared with the second method, this method requires only interception of the dispatched routine of original file driver, and then complete I / O control. This is even more subtle and common. Specifically, first capture Patch DriverObject -> MajorFunction and FastIoDispatch structure in memory, then complete I / O operation by monitoring and modifying, and finally return the modified data to file system driver. This approach is comparatively more common and mature.

2.7 Sony DRM Event

In November, 2005, many anti-virus companies released new tools to identify, or sometimes even remove the anti-copy protection software in CDs that are produced by Sony BMG Music Company. So, many professional anti-virus software companies considered it a potential security threat. The copy-protection software found in Sony's music album released was triggered when using a computer to play music CD. The software can automatically install itself from CD drive to the computer hard disk, and then hide itself from the user. Security experts said the technology maybe used by virus writers to hide their malicious software. According to CA (Computer Associates International, Inc.), Sony Corporation's software would set itself as the default media player after installation. Then it could transmit users' network address and the material of CDs that have been played in the computer to Sony Corporation. No matter if Sony BMG Music Company did it intentionally, this model of software destroyed the simple intention that user computers play MP3 songs from non-copy protection CDs.

2.8 Microsoft Alureon Rootkit

Partial users had blue screen after having deployed Microsoft's patch renewed in February, 2010. Microsoft carried on the investigation to this situation, and announced the survey result that it was relevant to the influence a specific patch and malicious software.

Engineers of Microsoft confirmed blue screen problem was caused by MS10-015 deployment. It was a Windows kernel patch and had repaired two long-standing

essence cracks. Computers with blue screen problem were infected by Alureon rootkit. Alureon rootkit belonged to the data-stealing Trojan horse category. It could intercept computer's internet flow to steal user name, password and credit card data. Alureon could also avoid detection, thus carrying out the malicious procedure without disturbance.

3. Development of Hiding Technology

In this chapter, we will state specific methods that the hackers use to hide rootkits. And we also introduce a new way to hide rootkits.

3.1 Memory Hooking Technology

Hook technology is now very popular. Many programs, including rootkits use Hook technology to change the program flow in order to achieve their own purposes. And the general Hook mainly refers to Hook in a system table or a system call [7]. So, many of the firewall or anti-virus software will be monitoring purposefully, which will greatly reduce success rate of the Hook.

Currently, rootkits have better approach:
First ,rootkits find the function address that is going to Hook in the memory.
And then, directly replace to the memory region where this function locates with a long-jump instruction.

Rootkits will fulfill the implementation at the jump position, and then perform the original function, and finally jump back to the original memory.

3.2 Directly Kernel Object Modification Technology [1]

This technology directly modifies part of the data and structure in the core structure to achieve rootkits function. Modifying the contents of the kernel directly can hide rootkits to further extent.

It is very difficult, because the structure and the kernel is very complicated and it is closely related to system version. The slightest negligence may cause the system to crash. It is really hard to write generic code at the moment. But as attacks against specific targets, it is very effective.

3.3 Advanced Techniques for Hiding

If you want to make your rootkit more stealth and more powerful, Hardware manipulation may be a good way. It is very powerful. It puts your rootkit below all other things (including anti-rootkit software). It can directly access to hardware. So you can do what you want to do. But it is very difficult to use. It is platform-specific, so your rootkit can not work on many computers.

In order to use this method, you need to modify the firmware first. Firmware is small, programs and data structures that stored in memory chips. It can internally control various electronic devices. you can use an external device, or on-board software to write rootkit to BIOS.

3.4 More Dangerous Rootkit------Bootkit

The method mentioned is not a popular method. It is so difficult to achieve and very platform-specific. Here is another rootkit which is also very dangerous. It is called Bootkit.

Let us view some information about the Bootkit. Stoned Bootkit is a new Windows bootkit which attacks all Windows versions from 2000 up to 7. It is loaded

before Windows starts and is memory resident up to the Windows kernel. With Stoned Bootkit you can install any software (for example a trojan) on any computer running Windows without knowing any password, even when the hard disk is fully encrypted.

There is no difference between rootkit and bootkit in function. They are both used to get the whole control of computer. But the bootkit has a different way to get access. The bootkit's custom boot sector code hijacks the startup routine before the Master Boot Record (MBR) loads.

The Master Boot Record (MBR) is the information in the first sector of any hard disk or diskette that identifies how and where an operating system is located [17].

This concept bootkit was first introduced in 2005 by researchers from eEye Digital Security. It is a method of exploiting the BIOS during startup [18].

4. Implementation

In this chapter, we will write our own rootkits. We will use language C to implement them. We will analyze the methods we use and fulfill them. After that we will test them and detect them. We will analyze the methods of detection.

4.1 Hide Process

Hooking is a technique to change the behavior of an OS, of applications, or something else by intercepting function calls or messages or events passed between software components. [16]. Many programs will use this technique, such as protection software. Of course, our rootkit also use this method.

Kernel Hooks

Kernel memory is the high virtual address memory region.

In most cases, processes can not access kernel memory. Why will our rootkit access the kernel memory? Because the userland hooks are easy to detect and prevent. And we hope our rootkit can get the full control of the computer and can hide itself (avoid detecting by the detection software). So it is a good idea to let our rootkit survive in the kernel mode.

Our rootkit will access kernel memory by implementing a driver.

The first thing we want to do is to hide process. We will use the method hook to hide.

The first thing we need to hook is SSDT (System Service Descriptor Table). The windows executive runs in kernel mode and provides support to the OS's subsystems: Win32, POSIX, and OS/2 [1] (The three subsystems comprise a well-documented set of APIs. Through these APIs, a process can request the aid of the OS). These native system services's addresses are listed in a kernel structure. It is called System Service Descriptor Table (SSDT). This table can be used to locate the address of the function in memory. And another table, System Service Parameter Table (SSPT) specifies the number of bytes for the function parameters.

The third table is KeServiceDescriptorTable (View Figure 4.1). The table contains two pointers, one to the portion of the SSDT, another to the SSPT. This table is exported by the kernel. So the table is powerful and easily used by hackers. Hackers can use this table to call system functions and then hide the rooktis.

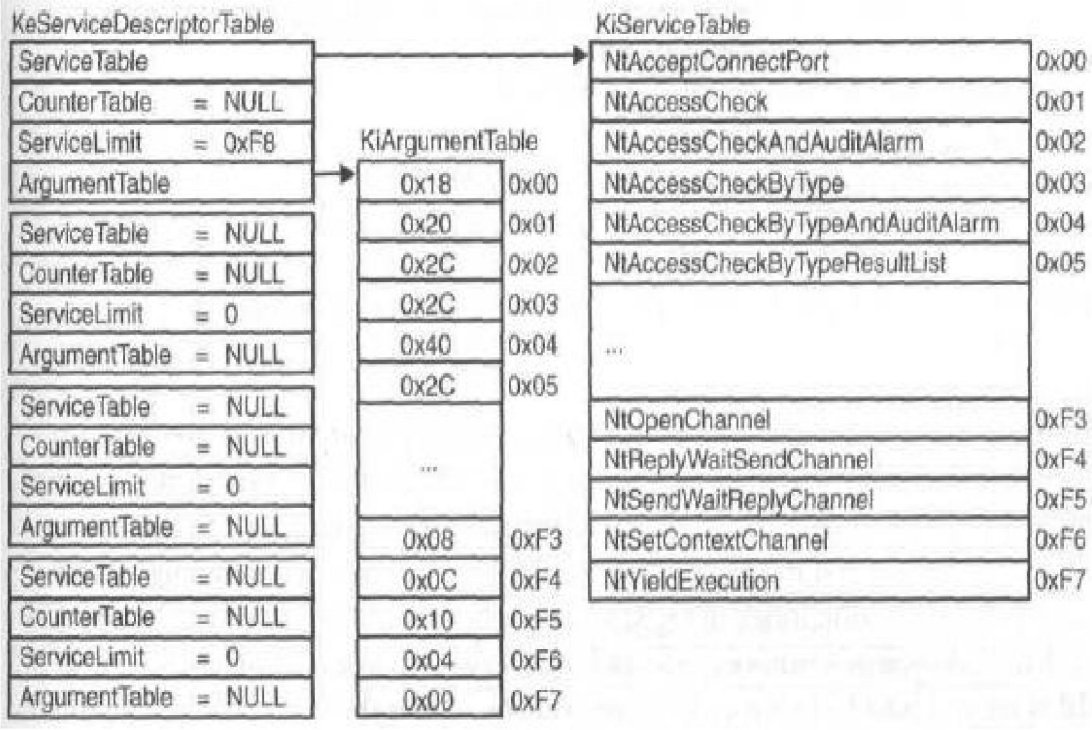


Figure 4.1 KeServiceDescriptorTable

http://www.bomi.gov.cn/Article/UploadFiles/200704/200748221514723.JPG

To call a specific function, the system service dispatcher, KiSystemService, simply takes the ID number of the desired function and multiplies it by four to get the offset into the SSDT. KeServiceDescriptorTable contains the number of services. This value is used to find the maximum offset into the SSDT or the SSPT.

An application can call the system service dispatcher. When our rootkit is loaded in the kernel mode as a device driver, it will change the SSDT to point to a function it provides instead of the original one. So if a application call the specific function, the rootkit function will be called.

The OS use the function ZwQuerySystemInformation to get information. Taskmgr.exe uses this function to get a list of processes and show it. (Figure 4.2). This program is the most common way to view the processes. If a process does not show in this program, It will not be found by user in most case.

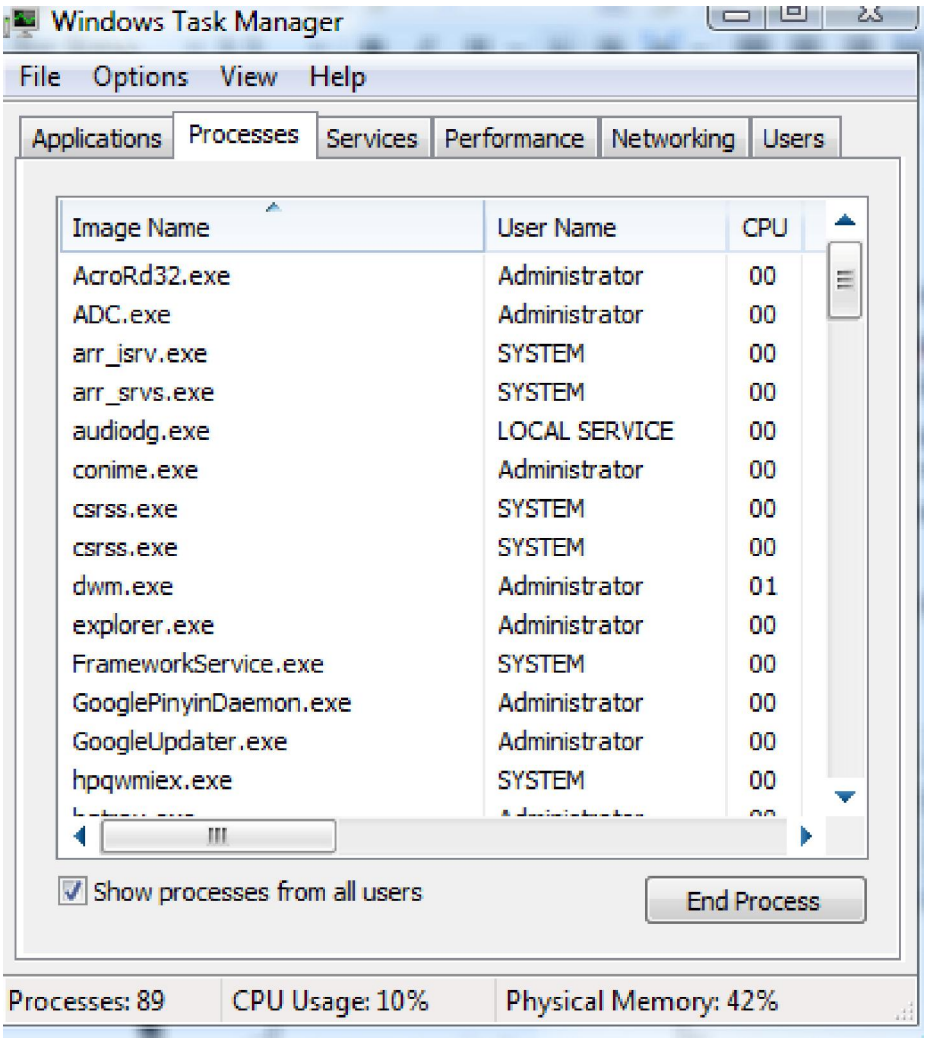


Figure 4.2 Taskmgr.exe

If we want to hide a specific process, we should use rootkit to replace the NtQuerySystemInformation (ZwQuerySystemInformation) function. Our hook can call the original function and filter the results.

Unfortunately, our rootkit can not hook the SSDT immediately because Windows has SSDT Memory Protection. They make the SSDT read-only because in most cases, an application does not need to modify this table. If we try to write to the read-only memory, the system will collapse and the Blue Screen will happen. So before we hook the SSDT, we need to change the SSDT Memory Protections.

There is a struct called MDL. The struct contains the information of memory. MDL describes pages in a virtual buffer in terms of physical pages. The pages associated with the buffer are described in an array that is allocated just after the MDL header structure itself.

In the struct MDL, the member MdlFlags indicates the state of memory. We can call MmCreateMdl to map the memory into our domain and build MDL. We can use Marco SYSTEMSERVICE to get function address which MmCreateMdl needs, After that we can change the MDlFlags. The last thing to do is to call

MmMapLockedPage to lock the MDL pages in memory. Now we can modify the SSDT.

When we hook the SSDT, there are two Macros we can use. They are HOOK_SYSCALL and UNHOOK_SYSCALL. The two macros can take the address of the Zw* function being hooked and exchange it with the address of the hook function.

We hook the ZwQuerySystemInformation with our function NewZwQuerySystemInformation. This function will filter the specific processes from the processes list. And the function will add the running of the filter processes to the Idle process. View Figure 4.3.

The result of Hiding Process

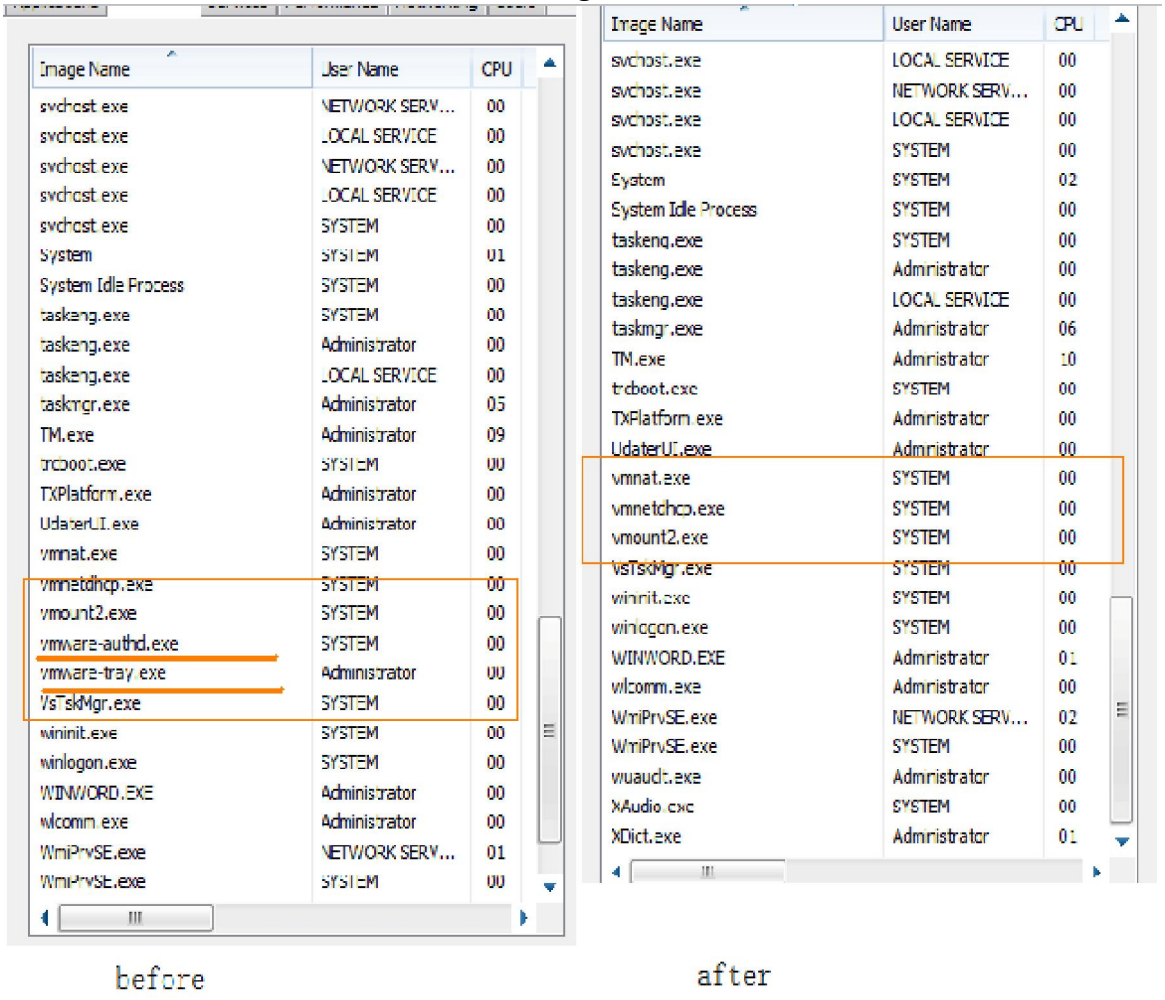
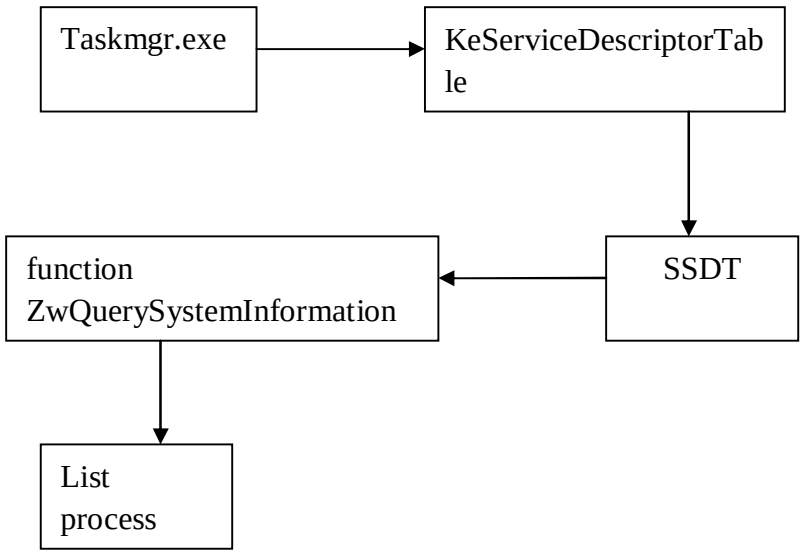


Figure 4.3 Hiding Process

Here are the processes of Hook .view Figure 4.4. In this figure, you will see the process of hook. Before hooking, the program will view the table SSDT, get the address of function ZwQuerySystemInformation and then call it. After hooking, we

use our address to replace the original address. So when the program tries to use the system function, it will call our rootkit's function.

Before hook:



After Hook

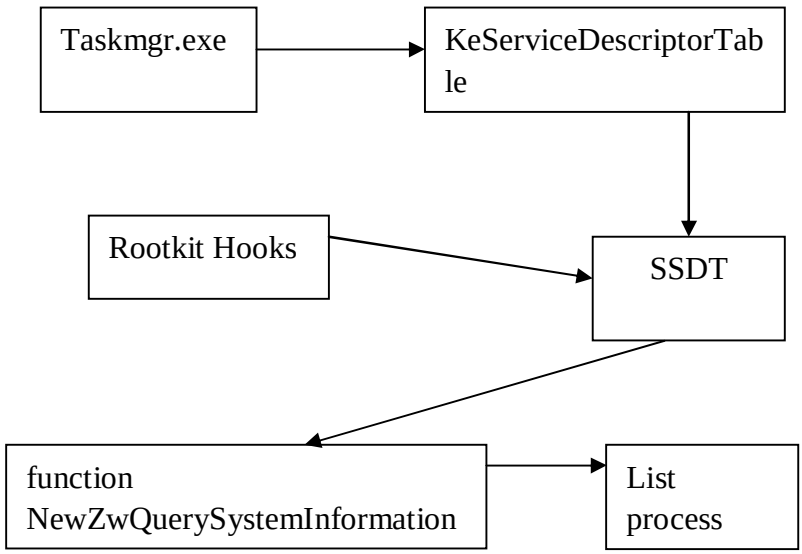


Figure 4.4 the processes of Hook

In our codes, there is a structure very important. It is _SYSTEM_PROCESSES struct _SYSTEM_PROCESSES. You can see in Appendix A struct _SYSTEM_PROCESSES.

And in this structure, we need to use two members. They are NextEntryDelta and ProcessName. NextEntryDelta points to next process. And we use ProcessName to find if the process is that we want to hide.

This is the function we use to hide:

```
NTSTATUS NewZwQuerySystemInformation(
                    IN ULONG SystemInformationClass,
                    IN PVOID SystemInformation,
                    IN ULONG SystemInformationLength,
                    OUT PULONG ReturnLength)
```

In this function, we will get the process list, compare their names and hide the specific processes. View Figure 4.5

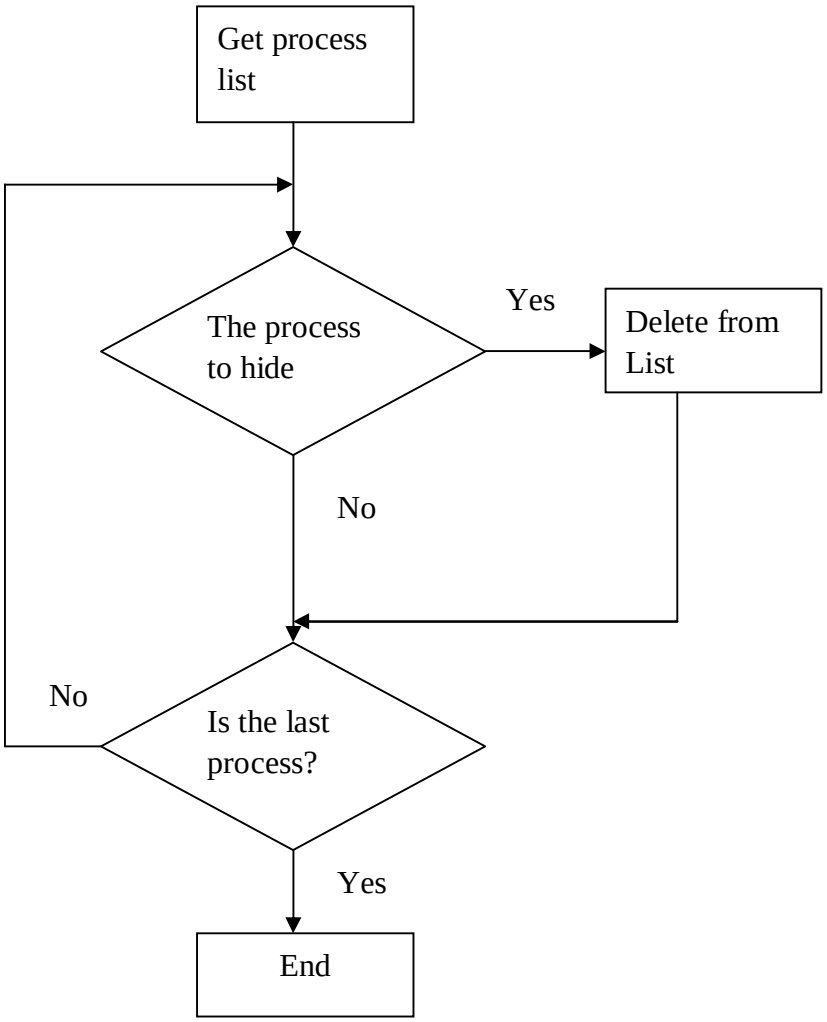


Figure 4.5 process of fuction NewZwQuerySystemInformation

The function OnUnload will be called when we want to uninstall our rootkit. The function will unhook the system call and free the memory we use.

The most important function is

```
NTSTATUS DriverEntry(IN PDRIVER_OBJECT theDriverObject,
    IN PUNICODE_STRING theRegistryPath)
```

It is the main function, the entrance of the driver.
It will do:

1. Register a dispatch function for Unload
2. save old system call locations
3. change the permissions on the MDL
4. hook system calls

4.2 Hide File

The second thing we need to hide is the specific file. Rootkit may have some files such as drivers, recording files, applications and so on. And we want to hide it. We do not want others (administrator or other users) find them. So we need to hide them.

4.2.1 Hook

One way to hide files is hook (Hook is very powerful).

Windows uses the method ‘enumerate’ to find a file. It will enumerate all files and subdirectories. And it is function NtQueryDirectoryFile to enumerate files

```
NTSTATUS NtQueryDirectoryFile(
    IN HANDLE FileHandle,
    IN HANDLE Event OPTIONAL,
    IN PIO_APC_ROUTINE ApcRoutine OPTIONAL,
    IN PVOID ApcContext OPTIONAL,
    OUT PIO_STATUS_BLOCK IoStatusBlock,
    OUT PVOID FileInformation,
    IN ULONG FileInformationLength,
    IN FILE_INFORMATION_CLASS FileInformationClass,
    IN BOOLEAN ReturnSingleEntry,
    IN PUNICODE_STRING FileName OPTIONAL,
    IN BOOLEAN RestartScan
);
```

If you want to know the parameters of the function you can view “The Undocumented Functions-Microsoft Windows NT_2000”

The parameters related to hiding are FileHandle, FileInformation, FileInformationClass.

Here are the descriptions for the three parameters (“The Undocumented Functions-Microsoft Windows NT_2000”)

FileHandle: HANDLE to File Object opened with FILE_DIRECTORY_FILE option and FILE_LIST_DIRECTORY access.

FileInformation: User's allocated buffer for output data.

FileInformationClass: Information class. Can be one of:

FileDirectoryInformation
FileFullDirectoryInformation
FileBothDirectoryInformation
FileNamesInformation
FileOleDirectoryInformation

You can see the structure in detail in Appendix B FileInformationClass.

In these structures, there are three members that are important to us. They are NextEntryOffset, FileName and FileNameLength. NextEntryOffset is the length of particular list item. First item can be found on address FileInformation + 0. So the second item is on address FileInformation + NextEntryOffset of first one. Last item has NextEntryOffset set on zero. FileName is a full name of the file. FileNameLength is a length of file name.

If we want to hide a file, we should compare.Some function:

DWORD getDirEntryLenToNext(IN PVOID FileInformationBuffer,
IN FILE_INFORMATION_CLASS FileInfoClass): to get NextEntryOffset.

void setDirEntryLenToNext(IN PVOID FileInformationBuffer,
IN FILE_INFORMATION_CLASS FileInfoClass,
IN DWORD value): to set NextEntryOffset.

PVOID getDirEntryFileName(IN PVOID FileInformationBuffer,
IN FILE_INFORMATION_CLASS FileInfoClass):
to get the filename of the specified directory entry.

ULONG getDirEntryFileLength(IN PVOID FileInformationBuffer,
IN FILE_INFORMATION_CLASS FileInfoClass): to the length of the filename of the specified directory.

If we want to hide one, we just need change the value of NextEntryOffset of previous record.

View Figure 4.6

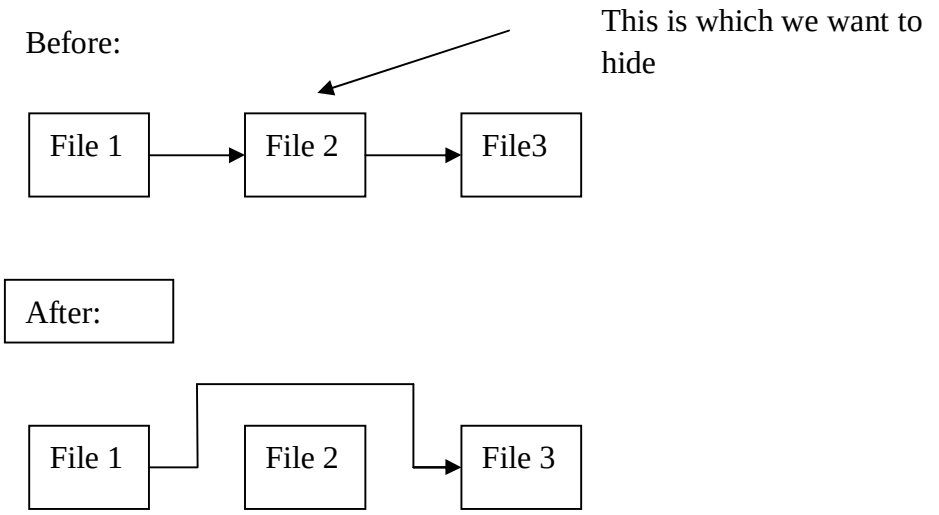


Figure 4.6 Hide File

If the one we try to hide is the last one, we just need to set the NextEntryOffset zero. View Figure 4.7

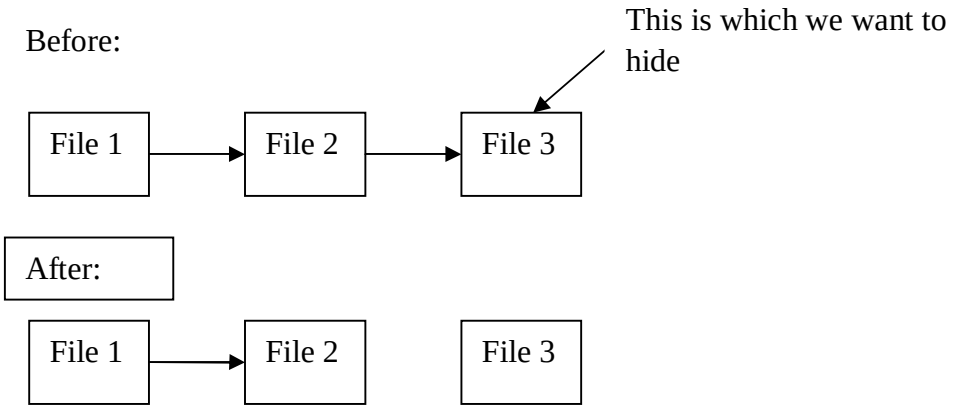


Figure 4.7 Hide File
(the last one)

If the one is the first one, we should let the next one be the first one. View Figure 4.8

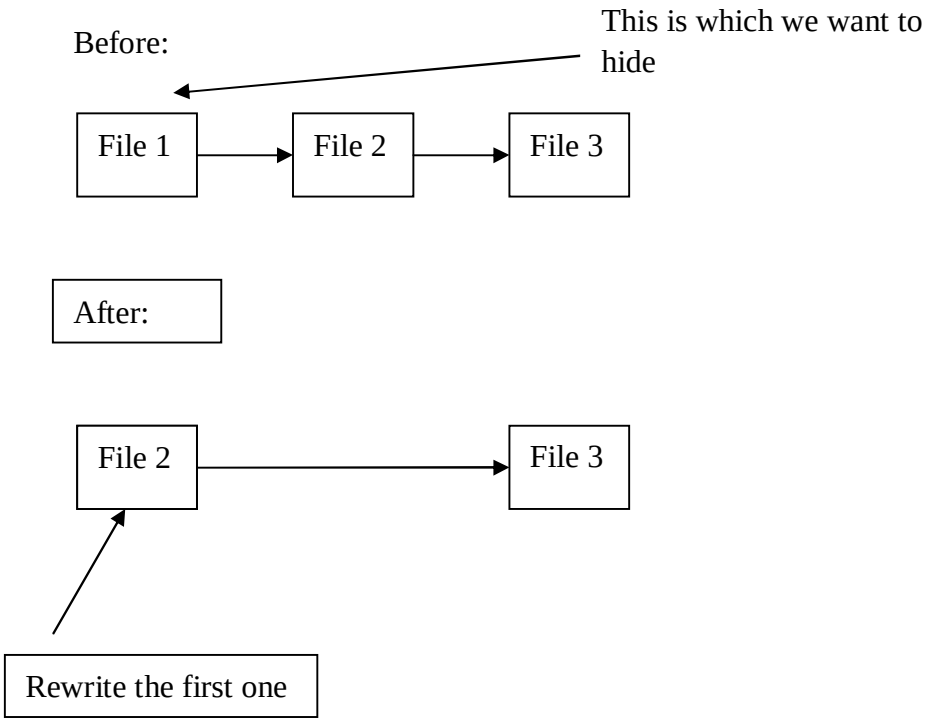


Figure 4.8 Hide File
(the first one)

We write assembly to unprotect memory. We set WP (It is written-protection) zero. We also can use MDL (used in hiding process) to change the memory protection.

4.2.2 File Filter

Another way to hide file is File Filter.

OS provides a lot for users. One of them is function. We can call this function to get some information. And we also can hook the system function to finish our targets. Another thing OS offers is driver. We can modify the behavior of an existing driver without coding a whole new driver. [1]

Driver chains exist for almost hardware devices. It works like this: (Figure 4.9)

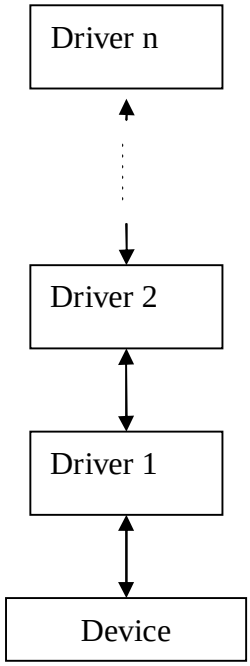


Figure 4.9 drivers chain(1)

The driver 1 will deal with Device. The driver 2 will deal with data that Driver 1 provides and format the data ····· Layer drivers will modify data before it pass on.

So we can make them work like this (Figure 4.10): we add our driver into the driver chain. Then we can modify the data.

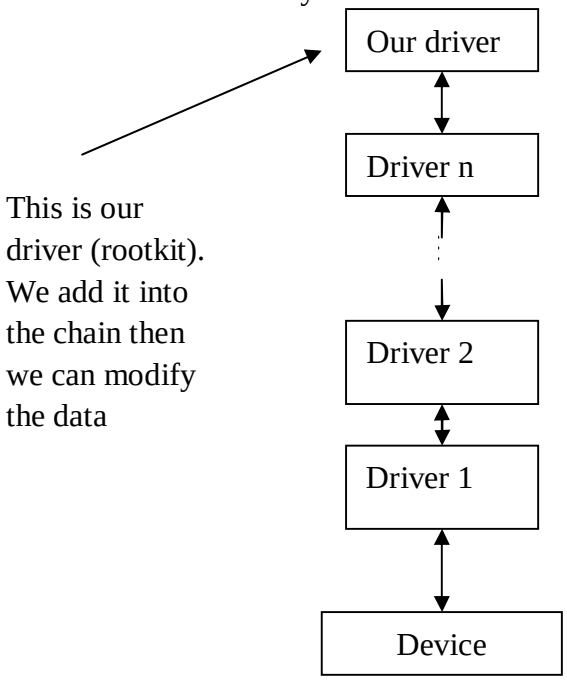


Figure 4.10 drivers chain(2)

Now let us get down to file filter.

First, of course is the “main function”------the entrance of the driver:

```
NTSTATUS
DriverEntry(
            IN PDRIVER_OBJECT DriverObject,
            IN PUNICODE_STRING RegistryPath)
{
```

Then we need to set the pass-through dispatch routine:

```
    for (i=0; i<=IRP_MJ_MAXIMUM_FUNCTION; i++)
    {
        DriverObject->MajorFunction[i] = MyIRP;
    }

    DriverObject->MajorFunction[IRP_MJ_CREATE] = FsDeviceCreate;
    DriverObject->MajorFunction[IRP_MJ_CLOSE] = FsDeviceClose;
    DriverObject->MajorFunction[IRP_MJ_CLEANUP] = FsDeviceClose;

    DriverObject->MajorFunction[IRP_MJ_DIRECTORY_CONTROL] =
FsDirectoryControl;
    DriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL] =
FsDeviceControl;
```

Before explain what these codes works. Let us explain something about IRP. IRP is I/O request packets. It is allocated by the I/O manager When the I/O manager allocate one IRP, it know the number of the drivers in the chain. And the IRP works like this (Figure 4.11):

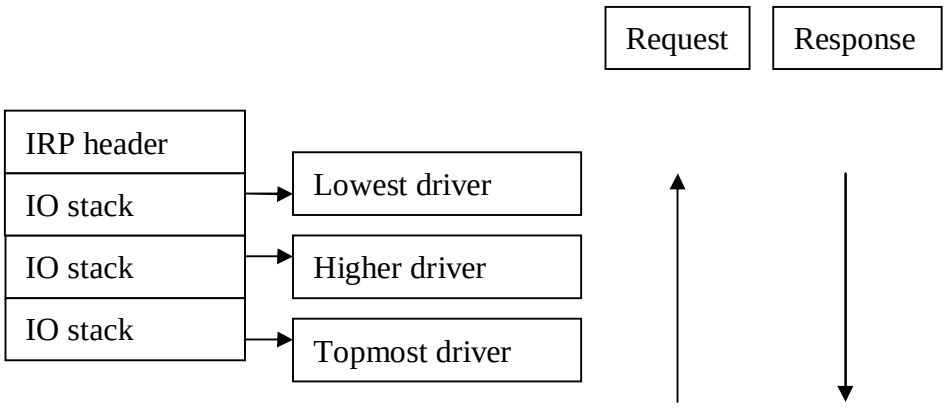


Figure 4.11 IRP stack

So we have chance to modify the data during the request and response. And we also know the number of devices. Then let us back to the codes. The function MyIRP will response all IRP requests. But we do not want to make our MyIRP so complex and we do not need to response all requests. We just need to response to parts of requests. So we add some small function to response. Then we use function AttachToDiskDevice() to add our device to the chain. After that the most important thing to do is to finish the dispatch routine function.

Some important functions:

The function:

NTSTATUS MyIRP(IN PDEVICE_OBJECT DeviceObject, IN PIRP Irp)

In fact it is not the function we use to hide the files. It will let the IRP that we do not need pass. Let those works successfully.

The function:

BOOLEAN AttachToDiskDevice(IN PUNICODE_STRING pDiskName, OUT PDEVICE_OBJECT *pOurDevice)

As we above, this function is to add our device to the chain. In this function, it will call system functions IoCreateDevice and IoAttachDeviceToDeviceStack to create and add device.

The function:

BOOLEAN HandleDirectory(IN OUT PFILE_BOTH_DIR_INFORMATION DirInfo, IN PULONG lpBufLenth)

This function is to hide file or directory.. In the function it will compare the file name and decide to hide it or not.

4.3 Hide Memory

As mentioned before, if we make rootkits invisible in memory, the memory that stores Rootkits program must be anonymous and non-paged memory. Let us get a brief idea of windows memory management.

First windows on 32 bit x86 systems can access up to 4GB of physical memory. This is because of the processor's address bus which is 32 bits. So the physical address range is from 0x00000000 to 0xFFFFFFFF which is 4GB. And windows allow each process have 4GB logical address space. How can windows make this? How can it give 4GB physical address to more than one process? The answer is paging. It allows the programs to use logical memory [8]. Graphically it looks something like Figure 4.12

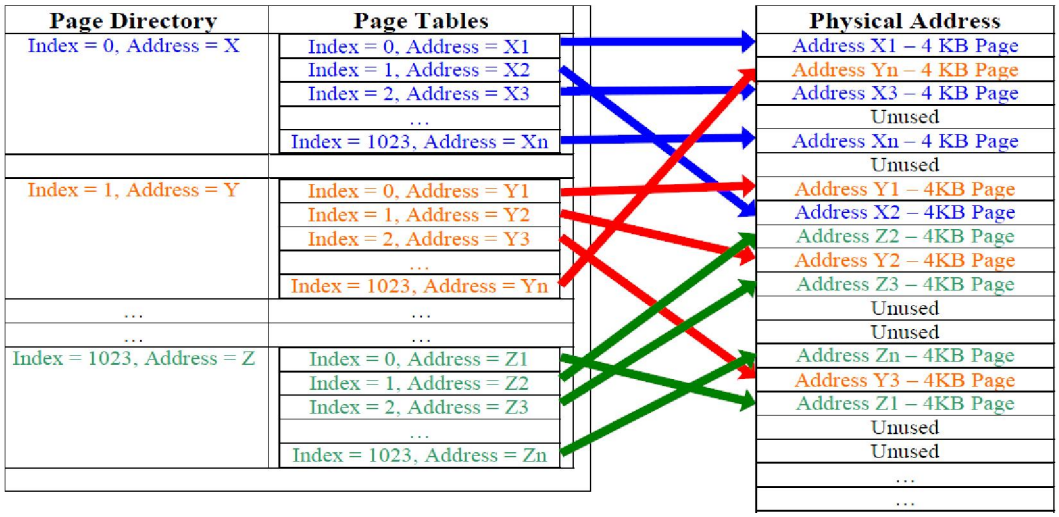


Figure 4.12 Paging technique

Here is a simple example. We think it may easy to understand. And in fact the windows works like this (More Complex but same principle).

Our physical memory is divided into 4 pages. Now we have two processes. They use some memory. View Figure 4.13

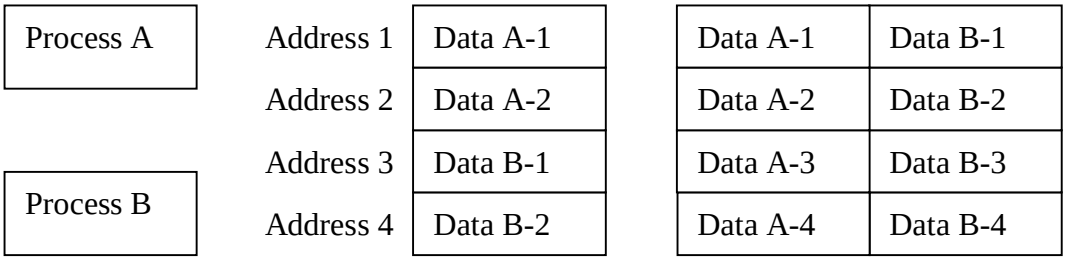


Figure 4.13 a simple example
(before paging)

Now Process A wants to use Data A-3. But the Data A-3 is not in the memory. So windows need to page it in. Windows calculates and finds it should put Data A-3 in the Address 3. And the Data B-1 (in Address 3 is no use now). So windows will page it out and put it on the hard Disk. View Figure 4.14

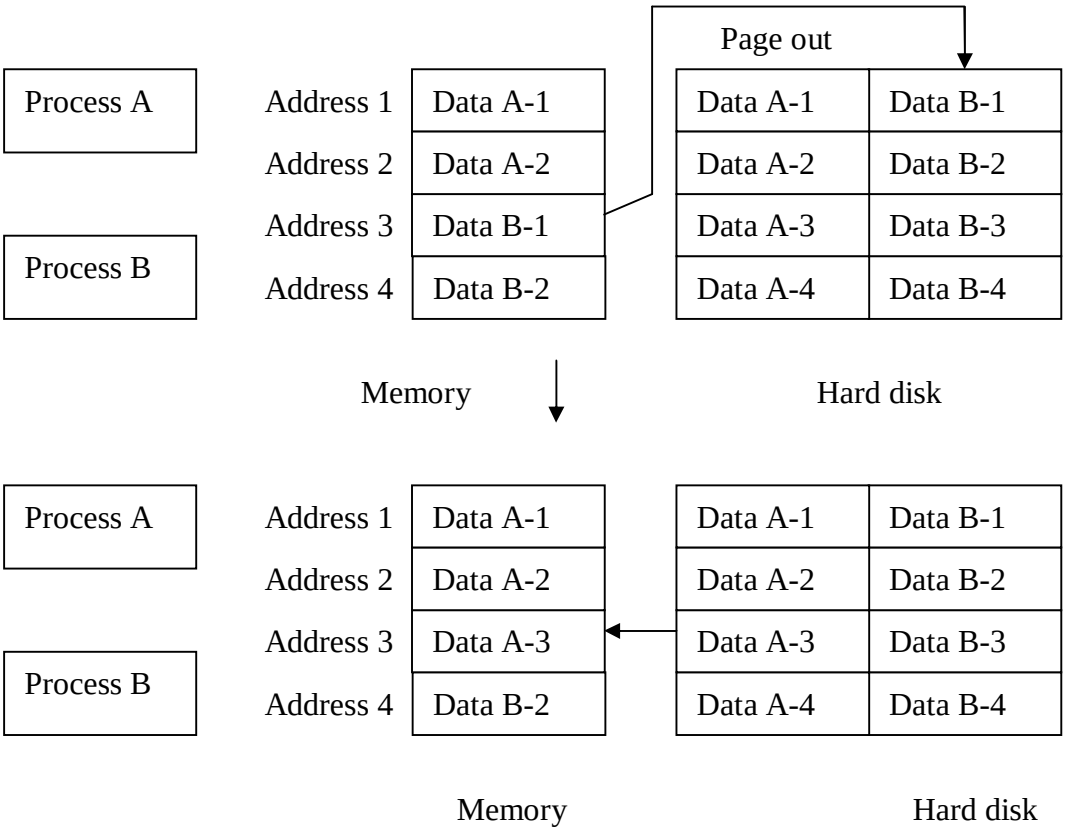


Figure 4.14 a simple example
(after paging)

So now Process A can use Data A-3.

Briefly speaking, windows will page out the memory that is no use now and page in the memory that the process want to use soon.

The paged memory may page out from the memory. If the processor wants to use the memory that has been paged out, the system will call a function to deal with the missing page interrupt. And this function is running on the dispatch_level. If the codes in the missing page also run on the dispatch_level, the function that deals with the missing page interrupt can not be called. And then Blue Screen happens [9]. (You can get more from windows DDK documentation)

Do we wish the memory that stores Rootkits program be paged out? Do we wish our rootkit cause Blue Screen? Of course not. So we may use the non-paged memory. The non-paged memory never be swapped out

We use the function MmBuildMdlForNonPagedPool to apply for non-paged memory.

And then the MmMapLockedPages routine maps the physical pages that are described by a given MDL.

Ok, now our rootkit can use non-paged memory and will not cause Blue Screen.

When we unload our rootkit, we need to free them. The function are MmUnmapLockedPages and IoFreeMdl.

4.4 Detection

When we installed our rootkit into my computer, my anti-virus software (McAfee) did not alarm. Then we use some specific software to detect the rootkit. We use IceSword. IceSword is a kind of software to Diagnose system and clean Malware.

We can view the result Figure 4.15 and Figure 4.16

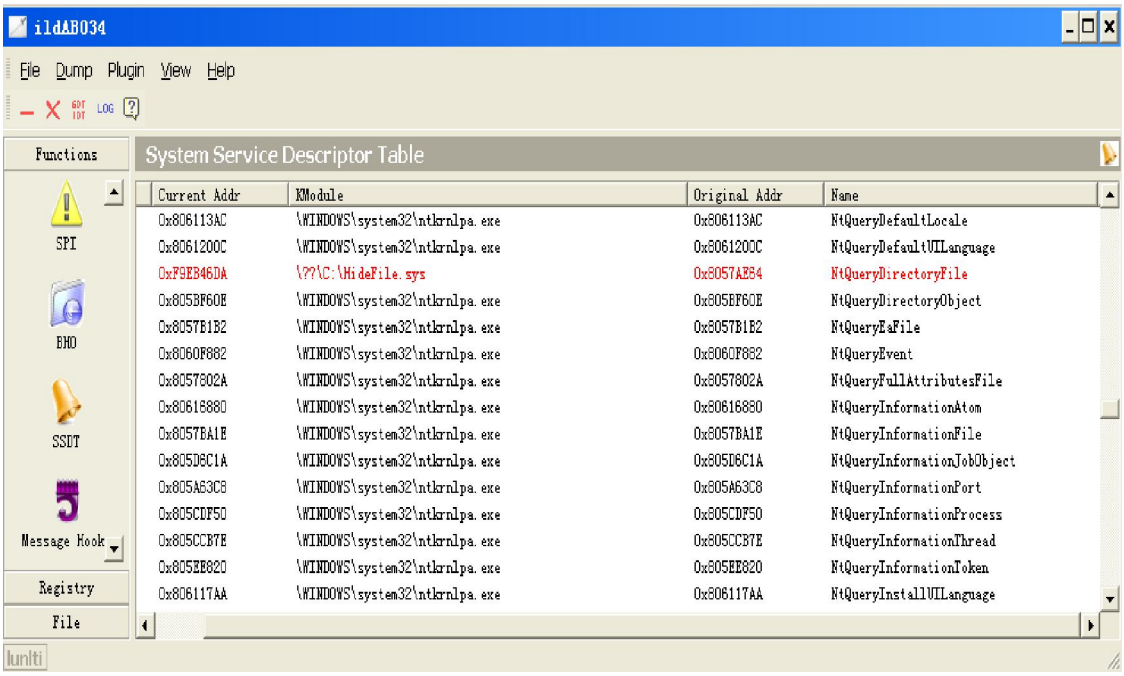


Figure 4.15 IceSword

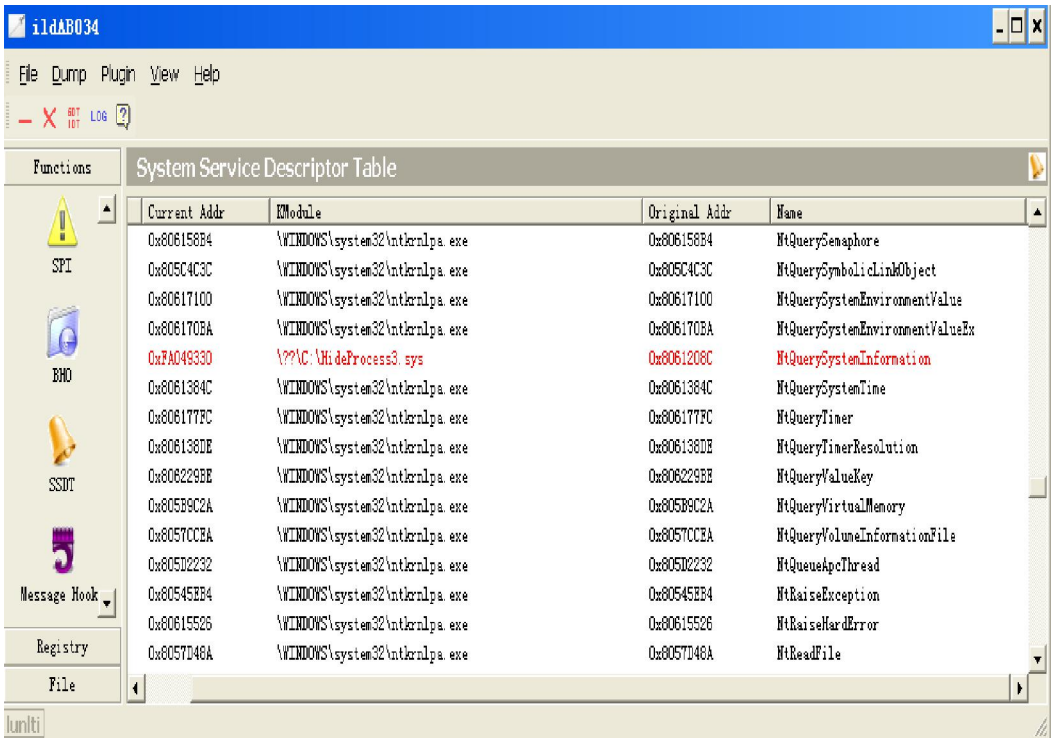


Figure 4.16 IceSword

The function NtQuerySystemInformation (we hook to hide processes) and function NtqueryDirectoryFile (we hook to hide files) became red. And it also shows our rootkits' location. So our rootkits were detected by this software.

How does it find the hook functions? [12]
IceSword will keep watch on the table SSDT. First it will analyze the SSDT and get the correct location of each function. Then it will check the current location of the function and compare it. If the correct location and current location are not same, it means the function is hooked.

How does it find the hiding processes?
IceSword uses table PspCidTable to list processes. PspCiDTable has a pointer to Handle_Table structure. It is used to store processes. The index is PID and TID. This table is not exported by kernel. It means the author must know how to get this table. The author calls function PsLookupProcessByProcessId to locate PspCidTable. But can the author get PspCidTable if someone hooks this function? The answer is yes. Because he does not use the traditional method to locate the function PsLookupProcessByProcessId. The traditional method is exporting by kernel. If he uses this method, the function may have been hooked. So he uses another way to locate. He opens the ntoskrnl.exe file, analyzes it and gets the function. After getting the function, the author can get PspCidTable successfully. This table has pointers (indexes) of all processes and threads. This HANDLE_TABLE does not belong to any process. It is independent. Now the IceSword can list all processes.

How does it find hiding files? [15]
The basic principle is that the IceSword creates an IRP and use IoCallDriver to send to FSD. The IoCallDriver routine sends an IRP to the driver associated with a specified device object [13]. FSD is file system driver [14]. But IceSword does more work. IceSword hooks \FileSystem\NTFS (for NTFS) and \FileSystem\FastFat (for FAT32). So it can get whole operations and IceSword write its own IoCallDriver.

.

5. Methodology of Detection

In this chapter, we will discuss some detection methods. We will state two common methods and some specific methods.

5.1 Check system files

There is some file integrity detecting program to achieve this method [4]. For example, Tripwire [5] , AIDE (Advanced Intrusion Detection Environment) and so on.

The main idea of these methods is to save a snapshot of key system files at first. And then check if the file is changed by file integrity testing programs.

Generally, user-level rootkit will modify the system program. Therefore, the user-level rootkit can be effectively detected by this method. However, the kernel-level rootkit will not modify any system files. This approach cannot be used to detect kernel-level rootkits.

5.2 Check system call table and the its function address

This approach has two ways of implementation: loadable kernel module and applications.

The application method is to get current address of the system call table from ‘kmem’. And detect kernel-level rootkits by comparison with the correct address [4]. Tools of this approach are such as chkrootkit [6] .

5.3 StreamArmor – Discover & Remove Alternate Data Streams (ADS) [11]

What is ADS?

A sophisticated hacker with more focused goals looks to a perimeter system breach as an opportunity to progress further inside a network or to establish a new anonymous base from which other targets can be attacked.

One popular method used in Windows Systems is the use of Alternate Data Streams (ADS). ADS provides hackers with a method of hiding root kits or hacker tools on a breached system and allows them to be executed without being detected by the systems administrator.

StreamArmor

StreamArmor is the sophisticated tool for discovering hidden alternate data streams (ADS) as well as clean them completely from the system.

5.4 How to detect system management mode (SMM) Rootkits

This kind of rootkit hides itself by running in a protected part of a computer's memory that can be locked and rendered invisible to the operating system.
You can see some strategies to detect them later.

5.5 Defense Strategy

As defense of other network security threats, defense of Windows Rootkit needs a suit

of comprehensive and reasonable strategy. So the defense will be effective, and the impact and the loss of the intrusion will drop to the lowest.

According to the technical principle and harmful feature of Windows Rootkit, the following will discuss the defense strategy that should be adopted from three aspects. Namely, they are prevention, detection and response management.

5.5.1 Prevention: Focus on defense

The weakness of Windows Rootkit is that the installation must have administrator's privilege. That is, the premise of installing Windows Rootkit is that the system is attacked. So, we could

a) Strengthen the system security, ensure that all operating systems and application software are the latest. What is more, there should be in-time patch installation, emergency repair and update program.
b) Also, prohibit useless service, do not run an application or network services that are not in use. Because unnecessary services and operation of the inappropriate software will generate unnecessary risk.
c) Select safe configuration method configuration when configuring server. Sometimes, convenient operation can bring about risks.
d) When necessary, use software of security assessments and vulnerability scanning to conduct a comprehensive security audit over the system .Thus enhancing system security.

Apart from basic methods to enhance system security, we could consider installing host-based intrusion prevention system (HIPS) tool [1]. It is considered the best way to prevent Rootkit currently.

HIPS can observe and prevent suspicious behavior related to system invasion. The suspicious behavior includes buffer overflow attacks, suspicious system service calling, illegal registry setting,etc.

5.5.2 Detection

Prevention is the key, but sometimes it cannot guarantee complete safety. So, detectability of Windows Rootkit should be enhanced. We have discussed detecting techniques. And the methods include real-time online detection and scanning detection started by the user.

File Integrity testing is one of the best way to test userland Rootkit. It seeks changes of key files by cryptographic hash of files and settings.

Nowadays, many anti-virus programs can also detect Rootkit. They focus on the feature of various Rootkits and update the virus database in time so as to effectively detect the Rootkits. You can also choose to install one or more reliable Anti-Rootkit software of the latest version. Thorough scanning on the system from time to time can help find Rootkits as early as possible.

5.5.3 Response Management

If the Windows system has been Installed Rootkit by the attacker, it is a very serious

security incident and the approach depends on information security strategy of different companies or organizations. But whatever the information security strategy is, this Windows operating system will not be trusted any more. We should find out why the system is attacked before recovering it. Because it is conductive to strengthen the system and network security later.

The best way to restore the system is to re-install the operating system, re-strengthen it, and install protective software. If the attacker has modified a small part of the operating system, the user can not know whether other components are modified. When dealing with the attacked system, should strictly inspect the systems nearby, and treat them the same way if there is problem. The secret information of the original system, such as account / password, also needs to be changes. Because it may have been stolen by the attacker or sniffed by network access. After the system recovery, it is better to strengthen protection and monitoring to the network and the system. The attacker may return to scene of crime again and try to re-enter the system.

6. Analysis

In this chapter we will analyze the advantages and disadvantages of the hiding methods. And then we will discuss some detection technique and state some ways to prevent rootkits.

In previous chapters, we talked something about rootkit, including rootkit principle, hiding technique, detection and so on.

We can know rootkit is just a technique at first. But because it is very powerful, some people try to use it and do something evil. Nowadays rootkit has become a very big problem. It is because that not only it can get whole control of system, but also it can hide. Rootkit can be combined with some other malwares, such as virus, Trojans. It can spread as virus. It can infect kernel drivers. It can gather users' information such as credit card code. And the most terrible, it can hide. It can not only hide itself but also hide other malwares. So maybe there is a rootkit that have been survived in your computer for a long time, and you never know.

We have mentioned before the most significant feature of rootkit is hiding. And there are many different hiding techniques for rootkit.

The basic technique for hiding is hooking. Some people call it age-old art. It is old, but it is not out of fashion. Hooking SSDT is still very powerful. It can hide from most protection software. Unless you use some specific anti-rootkits software, you can hardly find them. And you can use this technique to do most things such as hiding processes, hiding files. Compared with some other techniques, it is much easier to control. You can just use language C to finish your rootkit. And it is stable. But there is a obvious shortage for basic hooking (there is some advanced hooking, we will mentioned later). It is each to detection. You can use some specific anti-rootkits software to unhook. Because the basic hook (such as SSDT hooking) is easy to find, the rootkit developers try to use some other ways to hide. Then the advanced hook----inline hook appeared. What is inline hook? Inline hook is still hook. First the inline hook will save the code bytes of the target function and then overwrite with an immediate jump. The jump will lead to the hook function. Typically the place to overwrite is the first several bytes of the target function. We can see it is more difficult to detect than basic hook. Because it has hooked the inside of the function and most application consider the functions windows provides are reliable. But it is also more difficult to control than basic hook. It is more instable. It is much easier to cause Blue Screen. Because the place you hook is called many times. If you are not considerate enough, the hook may be called accidently. And then system crashes. So this hook has a strong dependence on system and the author's ability. The author must have a good command of system.

Another technique of hiding is DKOM (Direct Kernel Object Manipulation). It is different from hooks. Because it modifies some objects (data or structure). It has its advantages and disadvantages. On the one hand, it is very hard to detect. On the other hand, it is extremely fragile. You must understand some things about the object. It includes what objects looks like, how windows use object, when it is used and the version. Objects may change among different versions of windows or even among

different minor service-pack releases. So the person who wants to use DKOM must know the answers to these problems. Another drawback of DKOM is that it can not finish all purposes.

Some authors of rootkits put more emphasis on hardware. These rootkits are more dangerous and more evil. In most case if there is one rootkit in your system and you do not know how to clean it, you can reinstall your system to clean it. But you can not clean these rootkit even if you reinstall your system. Some rootkits will survive in the MBR. So they are not in the system. Protection software can not detect them. And these rootkits start before windows. When you start your computer, first BIOS will check which to start (hard disk ,CD or floppy disk), and then BIOS will get the MBR into memory. After that the MBR will check and find the system to start. So the rootkit that survive in the MBR can detect the system. If it finds the system starts, it can insert malware code into it.

How can we detect rootkits?

It is more difficult to write an anti-rootkit software than to write a rootkit. Developers need to consider more if they want to write an anti-rootkit software. They need to consider safety and stable. Because they may also use some hooks and it is easy to cause crash. Then they need to consider efficiency. The anti-rootkit software should find rootkits as soon as possible. They also need to confider protection. Some rootkits may disable the protection software. So the anti-rootkit software needs to prevent itself from infection or shutdown.

Now some softwares can detect rootkits. And as mentioned before, there are two main methods of rootkit detection. One is check system files. It will save the system state and make a snapshot before infection. Then compare the system with the snapshot. Another way is to check the system call table or function address to find rootkits. But do not think it is safe that you have installed protection software. A lot of software cannot detect the advanced rootkits (The rootkit uses more than one techniques to hide). And many rootkits will start before anti-rootkit software. So they can try to disable the anti-rootkit software.

We do not realize the safety problem of hardware. It is a tendency that rootkits try to survive outside of systems (It may survive in the MBR or even try to modify firmware).

Do you think your computer is safety enough?

We think prevention is important to detection.

1. install firewall and protection software
2. Lock or shutdown some system function in order to prevent system from modifying by rootkit
3. keep your system clean and backup your data.

7. Conclusion

Windows Rootkits are in a rapid development stage now. Technology of Windows Rootkits and Anti-Rootkits is developing competitively. With more and more awareness and excavation of the Windows operating system's kernel mechanism, there will emerge newer and more advanced technology of Windows Rootkits and Anti-Rootkits.

Rootkits usually use the hooking technology to hide itself. Certainly, some parts of high-level rookits use other technologies, including DKOM, dispatched routine and so on. A few more formidable rootkits can survive in MBR or modify firmware. This kind of rootkis are unable to eliminate even when re-instoring the system.

There are two main kinds of technologies in the rootkits detection.
First, check system files.
Second, check system call table and function address.

In the implementation of rootkit frame, we have fulfilled hiding process, files, and memory. We adopt some basic methods to hide process by hook ssdt, to hide files by hook ssdt and file filter, and to hide memory by using non-paged memory. Driver is the frame of the implementation, and it demonstrates how rootkit achieves its hiding technology.

Last but not least, in order to better defense Windows Rootkits and reduce the corresponding damage and impact, we should make a suit of complete and reasonable defense strategy. Also, technology of Windows Rootkits is widely used in anti-virus software, firewall, host intrusion prevention software and other security software. They will keep a secure network and system.

References

[1] Greg Hoglund and James Butler
ROOTKITS ：Subverting the Windows Kernel
Publisher: Addison Wesley Professional, 2005

[2] Chen Zhibo，Zhou Peng，He Yun.
Fast integer pel and fractional pel motion estimation for JVT[R]. Awaji Island,JP VT 6th

[3] Chen Zhibo，Du Cheng，Wang jinghua，
PPFPS—A paraboloid prediction based fractional pixel search strategy for H. 26L[J].
IEEE,International Symposium on，2002.

[4] Levine J，Grizzard J，Owen H.
A Metbodology to Detect and Characterize Kernel Level Rootkit Exploits Involving Redirection of the System Call TablelC]. Proceedings of Second IEEE International
Information Assurance Workshop，2004.

[5] Tripwire-Take Control of IT Security and Compliance Available at www.tripwire.org [Accessed 28 April 2010]

[6] Locally checks for signs of a rootkit Available at www.chkrootkit.org[Accessed 28 April 2010]

[7] Jeong Jechan g.
Fast sub—pixel motion estimation having lower complexity[c].
IEEE International Conference on Consumer Electronics，2003.

[8] Pankaj Garg
Windows Memory Management

[9] The Microsoft Windows DDK Docs Online Available at http://www.osronline.com/ddkx/ddk2.htm [Accessed 2 May 2010]

[10] Trojan Horse Attacks Available at http://www.irchelp.org/irchelp/security/trojan.html [Accessed 19 April 2010]

[11] StreamArmor – Discover & Remove Alternate Data Streams (ADS) Available at http://www.darknet.org.uk/2010/04/streamarmor-discover-remove-alternate-data-streams-ads/ [Accessed 10 May 2010]

[12] Icesword Available at http://blog.csdn.net/yaneng/archive/2009/06/18/4280066.aspx [Accessed 10 May 2010]

[13] IoCallDriver Available at http://www.osronline.com/ddkx/kmarch/k104_1agi.htm [Accessed 26 April 2010]

[14]FSD Structures Available at http://msdn.microsoft.com/en-us/library/aa915584.aspx [Accessed 9 May 2010]

[15] Icesword Available at http://clin003.com/rootkit/icesword-documents-hidden-achieve-breakthrough-37 [Accessed 10 May 2010]/

[16] Hooking Available at http://en.wikipedia.org/wiki/Hook_(programming) [Accessed 16 April 2010]

[17] Master Boot Record Available at http://searchcio-midmarket.techtarget.com/sDefinition/0,,sid183_gci214086,00.html [Accessed 15 May 2010]

[18] Building malware defenses: From rootkits to bootkits Available at http://searchsecurity.techtarget.com/tip/0,289483,sid14_gci1270250,00.html [Accessed 20 May 2010]

Appendices

Appendix A struct _SYSTEM_PROCESSES

_SYSTEM_PROCESSES

```
struct _SYSTEM_PROCESSES
{
    ULONG                       NextEntryDelta;
    ULONG                       ThreadCount;
    ULONG                       Reserved[6];
    LARGE_INTEGER               CreateTime;
    LARGE_INTEGER               UserTime;
    LARGE_INTEGER               KernelTime;
    UNICODE_STRING              ProcessName;
    KPRIORITY                   BasePriority;
    ULONG                       ProcessId;
    ULONG                       InheritedFromProcessId;
    ULONG                       HandleCount;
    ULONG                       Reserved2[2];
    VM_COUNTERS                 VmCounters;
    IO_COUNTERS                 IoCounters;
    struct _SYSTEM_THREADS      Threads[1];
};
```

Appendix B FileInformationClass

Here are structures:

```
typedef struct _FILE_DIRECTORY_INFORMATION {
   ULONG NextEntryOffset;
   ULONG FileIndex;
   LARGE_INTEGER CreationTime;
   LARGE_INTEGER LastAccessTime;
   LARGE_INTEGER LastWriteTime;
   LARGE_INTEGER ChangeTime;
   LARGE_INTEGER EndOfFile;
   LARGE_INTEGER AllocationSize;
   ULONG FileAttributes;
   ULONG FileNameLength;
   WCHAR FileName[1];
} FILE_DIRECTORY_INFORMATION,
*PFILE_DIRECTORY_INFORMATION;

typedef struct _FILE_FULL_DIR_INFORMATION {
   ULONG NextEntryOffset;
   ULONG FileIndex;
   LARGE_INTEGER CreationTime;
   LARGE_INTEGER LastAccessTime;
   LARGE_INTEGER LastWriteTime;
   LARGE_INTEGER ChangeTime;
   LARGE_INTEGER EndOfFile;
   LARGE_INTEGER AllocationSize;
   ULONG FileAttributes;
   ULONG FileNameLength;
   ULONG EaSize;
   WCHAR FileName[1];
} FILE_FULL_DIR_INFORMATION,
*PFILE_FULL_DIR_INFORMATION;

typedef struct _FILE_ID_FULL_DIR_INFORMATION {
   ULONG NextEntryOffset;
   ULONG FileIndex;
   LARGE_INTEGER CreationTime;
   LARGE_INTEGER LastAccessTime;
   LARGE_INTEGER LastWriteTime;
   LARGE_INTEGER ChangeTime;
   LARGE_INTEGER EndOfFile;
   LARGE_INTEGER AllocationSize;
   ULONG FileAttributes;
   ULONG FileNameLength;
```

```
        ULONG EaSize;
        LARGE_INTEGER FileId;
        WCHAR FileName[1];
    } FILE_ID_FULL_DIR_INFORMATION,
*PFILE_ID_FULL_DIR_INFORMATION;

    typedef struct _FILE_BOTH_DIR_INFORMATION {
        ULONG NextEntryOffset;
        ULONG FileIndex;
        LARGE_INTEGER CreationTime;
        LARGE_INTEGER LastAccessTime;
        LARGE_INTEGER LastWriteTime;
        LARGE_INTEGER ChangeTime;
        LARGE_INTEGER EndOfFile;
        LARGE_INTEGER AllocationSize;
        ULONG FileAttributes;
        ULONG FileNameLength;
        ULONG EaSize;
        CCHAR ShortNameLength;
        WCHAR ShortName[12];
        WCHAR FileName[1];
    } FILE_BOTH_DIR_INFORMATION,
*PFILE_BOTH_DIR_INFORMATION;

    typedef struct _FILE_ID_BOTH_DIR_INFORMATION {
        ULONG NextEntryOffset;
        ULONG FileIndex;
        LARGE_INTEGER CreationTime;
        LARGE_INTEGER LastAccessTime;
        LARGE_INTEGER LastWriteTime;
        LARGE_INTEGER ChangeTime;
        LARGE_INTEGER EndOfFile;
        LARGE_INTEGER AllocationSize;
        ULONG FileAttributes;
        ULONG FileNameLength;
        ULONG EaSize;
        CCHAR ShortNameLength;
        WCHAR ShortName[12];
        LARGE_INTEGER FileId;
        WCHAR FileName[1];
    } FILE_ID_BOTH_DIR_INFORMATION,
*PFILE_ID_BOTH_DIR_INFORMATION;

    typedef struct _FILE_NAMES_INFORMATION {
```

```
    ULONG NextEntryOffset;
    ULONG FileIndex;
    ULONG FileNameLength;
    WCHAR FileName[1];
} FILE_NAMES_INFORMATION, *PFILE_NAMES_INFORMATION;
```