

GHOSTWARE AND ROOTKIT DETECTION TECHNIQUES FOR WINDOWS

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING
AND THE INSTITUTE OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

By

Cumhur Doruk Bozağaç

September, 2006

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assist. Prof. Dr. A. Aydın Selçuk (Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assist. Prof. Dr. İbrahim Körpeoğlu

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assist. Prof. Dr. Alper Şen

Approved for the Institute of Engineering and Science:

Prof. Dr. Mehmet B. Baray
Director of the Institute

ABSTRACT

GHOSTWARE AND ROOTKIT DETECTION TECHNIQUES FOR WINDOWS

Cumhur Doruk Bozağaç

M.S. in Computer Engineering

Supervisor: Assist. Prof. Dr. A. Aydın Selçuk

September, 2006

Spyware is a significant problem for most computer users. In public, the term spyware is used with the same meaning as adware, a kind of malicious software used for showing advertisements to the user against his will. Spyware programs are also known for their tendency to hide their presence, but advanced stealth techniques used to be either nonexistent or relatively primitive in terms of effectiveness. In other words, most of the spyware programs were efficient at spying but not very efficient at hiding. This made spyware easily detectable with simple file-scanning and registry-scanning techniques. New spyware programs have merged with rootkits and gained stealth abilities, forming spyware with advanced stealth techniques. In this work we focus on this important subclass of spyware, namely ghostware. Ghostware programs hide their resources from the Operating System Application Programming Interfaces that were designed to query and enumerate them. The resources may include files, Windows Registry entries, processes, and loaded modules and files. In this work, we enumerated these hiding techniques and studied the stealth detection methodologies. We also investigated the effectiveness of the hiding techniques against popular anti-virus programs and anti-spyware programs together with publicly available ghostware detection and rootkit detection tools. The results show that, anti-virus programs or anti-spyware programs are not effective for detecting or removing ghostware applications. Hidden object detection or rootkit detection tools can be useful, however, these tools can only work after the computer is infected and they do not provide any means for removing the ghostware. As a result, our work shows the need for understanding the potential dangers and applications of ghostware and implementing new detection and prevention tools.

Keywords: spyware, ghostware, rootkit, stealth, detection.

ÖZET

WINDOWS İŞLETİM SİSTEMİ İÇİN GHOSTWARE VE ROOTKIT YAKALAMA TEKNİKLERİ

Cumhur Doruk Bozağaç

Bilgisayar Mühendisliği, Yüksek Lisans

Tez Yöneticisi: Yrd. Doç. Dr. A. Aydın Selçuk

Eylül, 2006

Spyware programları bilgisayar kullanıcıları için önemli bir problem teşkil etmektedirler. Genel olarak "spyware" terimi kullanıcılara reklam göstermek veya internet tarayıcılarından alışkanlıklarını takip etmek için kullanılan "adware" adlı kötü niyetli programlar ile aynı anlamda kullanılır. Bu özelliklerine ek olarak spyware programları varlıklarını gizleme eğilimleri ile de bilinirler, fakat bugüne kadar bu konudaki yeteneklerini ya kullanmadılar ya da oldukça kısıtlı kullandılar. Diğer bir deyişle, kullanıcıyı takip etmede oldukça gelişmiş olan spyware programları kendilerini saklama konusunda bu kadar başarılı değillerdi. Bu sebepten dolayı da dosya tarama veya windows kütüğü tarama teknikleri ile kolayca yakalanabiliyorlardı. Yeni spyware programları "rootkit" denilen kendilerini saklama konusunda uzman programlarla birleşerek kendilerini ustaca saklayabilen spyware'ler haline geldiler. Kısaca "ghostware" adını verdiğimiz bu programlar işletim sistemlerinin uygulamalara sunduğu programlama arayüzlerini etkileyerek kendilerini ve kaynaklarını saklayabilmektedirler. Bu çalışmamızda ghostware programlarının kullandıkları teknikleri ve onlara karşı kullanılacak karşı teknikleri inceledik. Ayrıca popüler anti-virüs ve anti-spyware programlarına ve karşı teknik kullanan araçlara karşı etkililiklerini araştırdık. Sonuçlara göre anti-virüs ve anti-spyware programları ghostware programlarını yakalamada ve kaldırmada yetersiz kaldı. Karşı teknik kullanan araçlar nispeten başarılıydı fakat bu araçlar sadece enfeksiyon sonrası kullanılabilirdiğinden ve sorundan kurtulmak için herhangi bir yöntem içermediğinden, ghostware programlarının tehlikelerini ve kullanım alanlarını anlayarak yeni yakalama teknikleri geliştirilmesi zorunluluğunu gösterdik.

Anahtar sözcükler: spyware, ghostware, rootkit.

To my family...

Acknowledgement

I would like to express my gratitude to my supervisor Assist. Prof. Dr. A. Aydın Selçuk for his supportive, encouraging and constructive approach throughout my masters study and his efforts and patience during supervision of the thesis.

I would like to express my sincere appreciation to the jury members, Assist. Prof. Dr. İbrahim Körpeođlu, and Assist. Prof. Dr. Alper Ően for reviewing and evaluating my thesis.

I want to thank to TÜBİTAK UEKAE /ILTAREN for the understanding and support during my academic studies. I also want to thank to the people in work, especially to my team leaders and to my teammates in Software Development Team, for their ambition and enthusiasm. It was a pleasure to work with them, and this let me continue my academic work together with my job.

Finally special thanks to my family for bringing me up and making me who I am; and to the love of my life, Gülin, for her endless patience, encouragement and support during my thesis work.

Contents

- 1 Introduction** **1**
 - 1.1 Spyware Types 2
 - 1.2 Problem Definiton 4
 - 1.3 Related Work 5
 - 1.4 Structure Of This Thesis 6

- 2 Ghostware Techniques** **7**
 - 2.1 Operating System Kernel and Windows API 8
 - 2.2 User Mode Hooking Techniques 14
 - 2.2.1 Import Address Table Hooking 15
 - 2.2.2 Inline Function Hooking 16
 - 2.2.3 DLL Injection 17
 - 2.3 Kernel Mode Hooking Techniques 19
 - 2.3.1 System Service Dispatch Table (SSDT) Hooking 20
 - 2.3.2 Interrupt Descriptor Table (IDT) Hooking 21

- 2.3.3 Input/Output Request Packet (IRP) Function Table Hooking 21
- 2.4 Direct Kernel Object Memory Access 22
- 3 Ghostware Detection 26**
 - 3.1 Hook Detection 28
 - 3.2 Hidden File Detection 29
 - 3.3 Hidden Process Detection 31
- 4 Experiments and Evaluation 32**
 - 4.1 Methodology 32
 - 4.2 Test Results 35
 - 4.3 Data Mining Approach 42
- 5 Conclusion 45**

List of Figures

2.1	Ring Structure	8
2.2	General Kernel Structure	9
2.3	Windows Kernel Structure	11
2.4	Passing Control from User Mode to Kernel Mode	13
2.5	System Service Dispatching General Structure	14
2.6	Import Address Table Hooking	15
2.7	Inline Function Hooking	16
2.8	DLL Injection	18
2.9	Kernel Mode Hooking Techniques	19
2.10	System Service Dispatch Table Hooking	20
2.11	I/O Request Packet Function Table Hooking	22
2.12	Process List Before Modification using DKOM	23
2.13	Process List After Modification using DKOM	24
3.1	Cross-view Detection Mechanism	27

3.2	System Virginty Verifier	30
4.1	Test Case Kernel-Mode Ghostware Structure	33
4.2	Vice detecting SSDT hooking	38
4.3	System Virginty Verifier by Ruanna Rutkowska	39
4.4	Hidden Process Detected by Swap Context-Detour technique . . .	41
4.5	IceSword showing hidden files	42
4.6	RAIDE detecting FUTo hooking	43

List of Tables

4.1	Anti-Virus Programs Detection Test Results	36
4.2	Anti-Spyware Detection Test Results	37
4.3	Ghostware and Rootkit Detection Tools Test Results	40
4.4	Data Mining Technique Test Results	40

Chapter 1

Introduction

At the end of the 90's, when Internet became popular, banner advertising companies started using new techniques for showing advertisements. Pictures at the top or bottom of a web page were simply ignored by the Internet users, if advertisements were not blocked by firewalls or other security products. Consequently spyware was born, with the aim of showing more advertisements to the user, sometimes without the need of having a website. The first samples were installed as bundled into freeware and shareware applications such as peer-to-peer file sharing programs. Their purpose was to display advertisements through pop-up windows while the user is surfing the web.

The term “Spyware” first began to be used in the computer software context in 1999 when Zone Labs used it in a press release for its Zone Alarm firewall product [37]. In 2000, Gibson Research launched the first anti-spyware product, OptOut. Steve Gibson, the developer of OptOut, states that “Spyware is any software that employs a users Internet connection in the background (the so-called *backchannel*) without their knowledge or explicit permission” [32]. Consequently, spyware refers to software that was installed without the knowledge and consent of users and that operates in stealth.

The actions of many spyware programs go beyond simply facilitating advertisements or gathering non-personal data [9]. Today, they are using techniques

similar to malicious threats ranging from silent installation to exploiting vulnerabilities in operating system components. Furthermore, once installed on the system, they are trying to stay resident and maintain their covert operations. Many spyware programs are able to gather personal or confidential data and transmit this data over network.

1.1 Spyware Types

A problem regarding spyware is the lack of a standard definition and categorization. Some prefer a narrow definition that focuses on the surveillance aspects of spyware and its ability to collect, store and communicate information about users and their behavior. Others use a broad definition that include pop-up advertiser applications, toolbars, search tools, browser hijackers and dialers. Definitions for spyware also include hacker tools for remote access and administration, keylogging and cracking passwords. In general, we can divide the spyware into two parts:

- **Adware:** These programs are used for showing advertisements to the user. We can categorize the types of adware like this:
 - *Browser Hijackers:* Once installed in a users web browser, changes its default start, search, and error page settings to alternative sites. Browser redirection inflates the websites traffic gaining higher advertising revenues, referral fees, and purchase commissions made through the redirected website.
 - *Internet Explorer Toolbars:* Internet Explorer allows users customise the interface through dynamic loaded plug-ins called Browser Helper Objects (BHO). Some plug-ins perform necessary functions, such as the *Yahoo Toolbar* or *Google Toolbar*. However, spyware applications can install and display themselves as toolbars, search bars, or task buttons incorporated into Internet Explorer through browser plug-ins. Various spyware toolbars spy, modify, and redirect web requests or

- cause indecent pop-ups and send information from the host, such as *XXXToolbar*.
- *Pop-up Advertiser Applications*: Display advertisements based on entered website URLs while surfing the web or specific keywords entered through a search engine. Some spyware applications like Cydoor download the advertisement database to a users workstation in the form of a list of URLs during installation. Gator fetch advertisements based on the users web surfing activity and some criteria programmed in the application.
 - *Drive-by Downloads*: Internet Explorer uses ActiveX controls to enhance the browsers functionality and provide interactive programs for Internet like Shockwave and Flash. A *drive-by download* is a program that automatically downloads to a users computer, often without the users consent or knowledge and having full access to the Windows operating system using exploits in browser.
- **Stealth Malware**: The term “stealth malware” refers to a large class of software programs that try to hide their presence from operating system (OS) utilities commonly used by computer users and malicious code detection software such as anti-virus and anti-spyware programs. Stealth techniques range widely from the simple use of hidden file attributes to sophisticated code hiding in bad disk sectors, from user-mode API interception to kernel-mode data structure manipulation, and from individual trojan OS utilities to OS patching with system-wide effect.
 - *Rootkits*: A number of tools available to the owner of the tool, making it available for connection and providing stealth features for the attacker. *Rootkit* originally referred to a set of recompiled Unix tools such as *ps*, *netstat* and *passwd* that would carefully hide any trace of the intruder that those commands would normally display, thus allowing the intruders to maintain “root” privilege on the system without the system administrator even seeing them. Now the term is not restricted to Unix-based operating systems, as tools that perform a similar set of tasks now exist for Microsoft Windows operating system.

A rootkit is often used to hide utilities. They are also used to abuse a compromised system, by helping the attacker hide his access. For example, the rootkit may hide an application that spawns a shell when the attacker connects to a particular network port on the system.

- *Trojans*: Similar to rootkits, they are used to abuse a compromised system, by opening *backdoors* to help the attacker subsequently access the system and track victim activity. In general trojans include a set of tools for data gathering. These tools can intercept network packets, capture display and log key strokes. However, they do not include advanced stealth features like a rootkit.
- *Winsock Hijackers*: A layered service provider (LSP) is between a computers Winsock layer and TCP layer and can modify all data that passes through the system. Spyware can install malicious LSPs to this layer called Winsock Hijackers. They can monitor the network, accessing all data passing through the desktop, capable of redirecting web requests to affiliate websites. Any attempt to remove these Winsock hijackers can break the LSP chain and cause the Internet connection to stop working.
- *Man-in-the-Middle Proxies*: Redirects all web surfing activity, including secure connections, to a man-in-the-middle proxy under the disguise of Internet connection accelerator. Can harvest sensitive information such as passwords, credit card numbers, bank account information, health care records, and confidential data.

1.2 Problem Definiton

In public, the term spyware is used with the same meaning as adware, since stealth malware properties used to be either nonexistent or relatively primitive in terms of effectiveness. In other words, most of the spyware programs were efficient at spying but not very efficient at hiding. This made spyware to be easily detectable

with simple file-scanning and registry-scanning techniques. Afterwards leading-edge spyware developers such as *CoolWebSearch* evolved, and they employed hiding techniques similar to windows rootkits, in order to avoid detection. These program consists of two parts. First, they gather user data such as keystrokes and network communications. Secondly, they hide their presence from the user and/or make uninstallation difficult.

When stealth features first appeared in computer viruses their main purpose was to make the work of anti-virus researchers and applications as difficult as possible. Today the apparent blending of malicious code writing and hacking gives stealth code a whole new perspective. One of the most important things for any attacker after compromising a host on the Internet is to operate covertly on the host as long as possible. This is where stealth code becomes useful, it makes both the intruder and its backdoor operations be invisible to user and detection tools. This emerged spyware with rootkit techniques, or as we call it ghostware.

Ghostware programs hide their resources from the OS-provided Application Programming Interfaces (APIs) that were designed to query and enumerate them. The resources may include files, Windows Registry entries, processes, and loaded modules. Ghostware programs are the next generation of information security problem. In this work we will show their potential, what they can do and how they can escape being detected, to shed light on this new type of malicious software.

1.3 Related Work

There is not much academic work on spyware and ghostware. The academic research on spyware began with the articles attempting to define the problem and warn ordinary users for the potential dangers and capabilities of spyware programs. For example, [53] and [13] enumerated some methodologies for spyware infection, while [19] and [11] defined a categorization of spyware. It becomes a controversial issue when ACM Communications reserved August 2005 volume, which included articles like [23], [1] and [61], however the volume does not

include any research on this subject, the articles were about the discussion of legal and ethical issues.

[62] and [2] explained how peer-to-peer programs and popular free programs are used for the distribution of bundled spyware programs. [46] made experiments on the effectiveness of spyware communication in a university campus network.

After all this noise, Microsoft took the problem seriously and formed a spyware research group by buying Giant Anti-Spyware Networks. [59] explains their offered technique for stopping spyware programs by monitoring Windows auto-start points.

The rootkits were not taken seriously in the Windows world, as they were considered as a problem for Unix users only. [40], [24], [21], [7] and [45] are the pioneers of Windows rootkits. They defined hooking methods for Windows API and showed numerous ways of alternating the data flow in memory space. [25], [14], [22] and [49] also included a history of rootkit methodologies and explanations on hooking methodologies.

Our problem, that spyware using rootkit techniques can be extremely effective, was first introduced by [51]. [17] also points out this problem and [60] and [58] is a research on the detection of ghostware, however their solution is not a public release yet.

1.4 Structure Of This Thesis

In this work we focus on this important subclass of spyware, namely ghostware. In Chapter 2, we will reveal their hiding techniques. In Chapter 3, we will enlist the detection methods against these techniques. In Chapter 4 we will investigate the effectiveness of their hiding techniques against popular anti-virus programs and anti-spyware programs together with publicly available ghostware detection and rootkit detection tools. We will also test detection techniques.

Chapter 2

Ghostware : Spyware with Stealth Techniques

The main purpose of a spyware program is allowing continued access to the computer without being detected during this time. As we explained in Chapter 1, spyware programs employ stealth techniques for this purpose. Software using these techniques are generally called “stealth malware”, which refers to programs that try to hide their presence from operating system (OS) utilities commonly used by computer users and malware detection software such as anti-virus and anti-spyware programs. In this paper, we will use the term “ghostware” [60] for these type of programs.

Main idea behind stealth features is to alter the execution path of any functionality in operating system such that calls to system services or returns of the system calls are diverted to the ghostware. This is called “hooking”. After the diversion, ghostware can modify the functionality or filter the return values in order to hide any information which can reveal its existence. This includes function calls used to enlist processes, files, services or ports. In order to understand the concept of hooking first we need to understand how operating system responds to kernel functionality requests.

2.1 Operating System Kernel and Windows API

The Intel x86 family of microchips use a concept called rings for access control. The term ring appears to refer to the original 80386 architecture reference manual's drawing of the four levels of protection. There are four rings, with Ring Zero being the most privileged and allowed total control of the processor, while Ring Three being the least privileged, providing the most processor level protection as shown in Figure 2.1. Internally, each ring is stored as a number and there are not actually physical rings on the microchip [8].

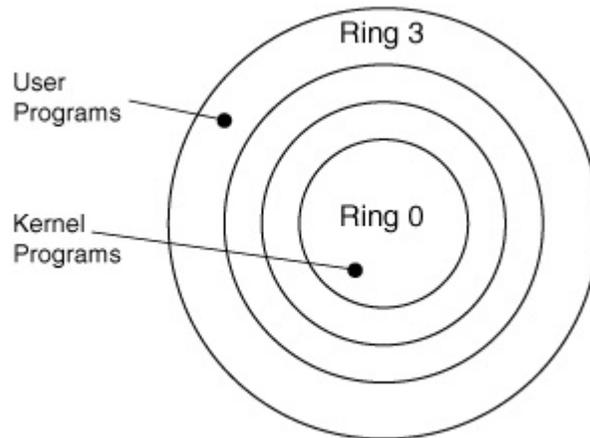


Figure 2.1: Ring Structure

For the X86 processor systems, the use of other priority levels has been deprecated. Paging only has the concept of user or system access (priority 3 or priority 0, respectively). The processor and operating system work together to handle transitions between the priority levels. The kernel itself, in both Linux and Windows, runs in Ring 0, and a process running in Ring 0 is said to be at kernel level. If a process runs in Ring 0, it can access all of the kernel's memory structures, and are therefore at the same level as the kernel code. User mode processes run in Ring 3, are not able to access kernel space directly. By relying on Ring 0 and Ring 3, all software on the machine is really carved up into two different worlds:

kernel mode (running in Ring 0) and user mode (running in Ring 3) [22].

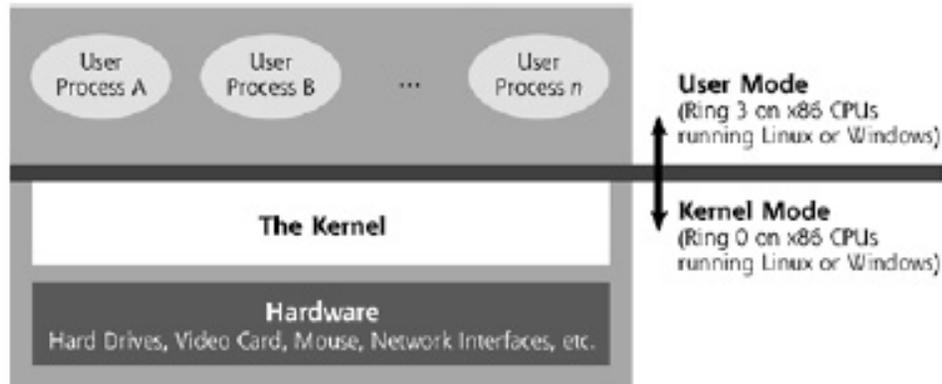


Figure 2.2: General Kernel Structure

The user mode is what users typically see and interact with on a day-to-day basis on your system, as it includes the programs they run, such as a mail program, office program, text editor or game. The kernel mode, is beneath this system controlling the whole operation managing access to the hardware and OS functions. When a system boots up, the kernel is loaded into memory and begins execution in Ring 0. After the kernel gets itself set up in memory, it activates various user-mode processes that allow individual users to access the system and run programs.

The kernel is special software that controls various extremely important elements of the machine. We are interested in Windows malicious code, so from this point, the word kernel will be used as Windows kernel. As illustrated in Figure 2.3, the kernel sits between individual running programs and the hardware itself. Performing various critical housekeeping functions for the operating system and acting as a liaison between user-level programs and the hardware, the kernel serves a critical role. It includes the following core features [49]:

- Process and thread control: The kernel dictates which programs run and when they run by creating various processes and threads within those

processes. A process is nothing more than some memory allocated to a running program, and the threads are individual streams of execution within a process. The kernel orchestrates various processes and their threads so that multiple programs can run simultaneously and transparently on the same machine. Interprocess communication control. When one process needs to send data to another process or the kernel itself, it can utilize various interprocess communication features of most kernels to send signals and data.

- Memory control: The kernel allocates memory to running programs, and frees that memory when it is no longer required. This memory control is implemented in the kernel's virtual memory management function, which utilizes physical RAM and hard drive space to store information for running processes.
- File system control: The kernel controls all access to the hard drive, abstracting the raw cylinders and sectors of the drive into a file system structure.
- Other hardware control: The kernel manages the interface between various hardware elements, such as the keyboard, mouse, video, audio, and network devices so various programs can utilize them for input and output operations.
- Interrupt control: When various hardware components of the machine need attention (e.g., a packet arriving on the network interface) or a program encounters an unusual event (e.g., division by zero), the kernel is responsible for determining how to handle the resulting interrupts. By taking care of the interrupt itself using kernel code or sending information to a particular process to deal with it, the kernel keeps the system operating smoothly.

When most programs run, control may have to pass from user mode into kernel mode, such as when the program needs to interact with hardware or use some other kernel functionality. For this purpose control will be passed from user mode to kernel mode, through tightly controlled interfaces. The software that

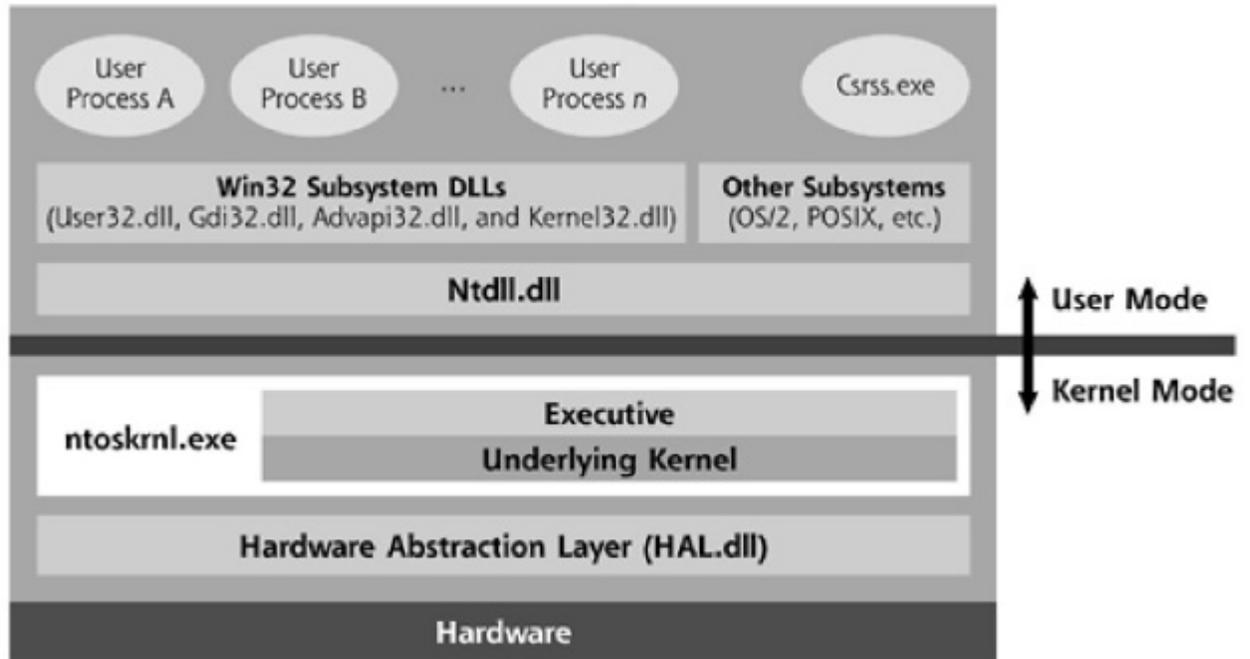


Figure 2.3: Windows Kernel Structure

implements this transition from Ring 3 to Ring 0 is referred to as a call gate, as it acts as a gate for user-mode processes into software living in kernel mode [38].

The programs that are run on a day-to-day basis, such as an internet browser, make function calls into various Win32 subsystem DLLs to interact with the operating system. When developers create programs to run on Windows, these programs include Win32 function calls to the Windows itself, implementing the OS API. Win32 functions have all kinds of capabilities, such as manipulating the screen, opening files in hard disk or running other programs [36]. These functions are grouped into several different files, each with its own lump of code to accomplish certain tasks, including *User32.dll*, *Gdi32.dll*, *Advapi32.dll*, and *Kernel32.dll*.

Function calls in user mode can do one of three things [50]:

First, if they don't require kernel-level interaction with hardware or other processes, a Win32 user mode function could just handle the request and send

a response. For example, the *GetCurrentProcessId* function, returns a process's own process ID number.

Secondly, if handling of a function call from a user-mode application involves the Win32 DLL needing information from a very special user-mode process called *Csrss.exe*, which is responsible for keeping the Win32 subsystem running. The “csrss” is an abbreviation for Client/Server Run-Time Subsystem, and the executable keeps the Win32 subsystem operating by invoking user processes and maintaining the state associated with each process. User-mode processes can ask *Csrss.exe* for information about themselves or other processes without calling the kernel.

Thirdly, a user-mode application could ask a Win32 DLL to take some action that requires invoking a kernel function. For example, reading or writing file means using the *ReadFile* or *WriteFile* function calls, which in turn requires the corresponding Win32 DLL, ‘Kernel32.dll’, to interact with the hardware. *Kernel32.dll* will map the *ReadFile* and *WriteFile* function calls to another Win32 DLL called *Ntdll.dll*, which is quite an unknown and internal API. *Ntdll.dll* takes the highly documented function calls of the Win32 API (like *ReadFile* and *WriteFile*), and convert them into the underlying function calls understood by the kernel (called *NtReadFile* and *NtWriteFile*, respectively).

Ntdll.dll is responsible for making the transition from user mode to kernel mode, jumping through a call gate into the kernel. Basically it invokes a kernel functionality called the Executive, which has kernel mode responsibilities such as making kernel function calls available to user mode, making various kernel-level data structures available to other kernel-level processing, and managing certain kernel state and global variables [49]. The *Executive* is placed in a file called *Ntoskrnl.exe*. After being invoked it will determine which piece of underlying kernel code is needed to handle a request and it will pass the execution to the corresponding code. In the case of reading or writing a file, the *Executive* needs to interact with hardware(hard disk), and it will accomplish this by using the Hardware Abstraction Layer (HAL) of kernel. This layer is composed of device drivers and the interface for reaching these drivers are implemented in a file called

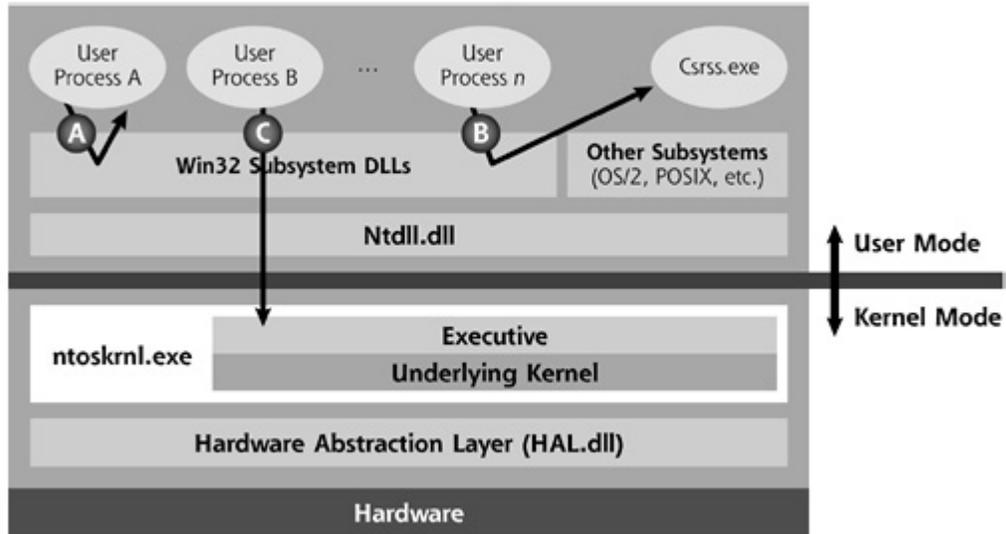


Figure 2.4: Passing Control from User Mode to Kernel Mode

HAL.dll, which makes various different vendor hardware products look consistent to the kernel.

The transition from user mode to kernel mode, or from *Ntdll.dll* to the *Executive* is called system service dispatching. As shown in Figure 2.5, invoking is accomplished by use of a CPU interrupt signal. *Ntdll.dll* triggers interrupt number 0x2E on x86-compatible processors to invoke this transition [49]. Inside the *Executive*, there is a part called the system service dispatcher, looks at the parameters and type of the system call and looking up a table called system service dispatch table it determines here the appropriate system service code to handle the request is located in kernel memory. After that, execution flow is transitioned to the appropriate kernel code. This is how a user-mode process can read from or write to a file, or perform other interactions with the hardware.

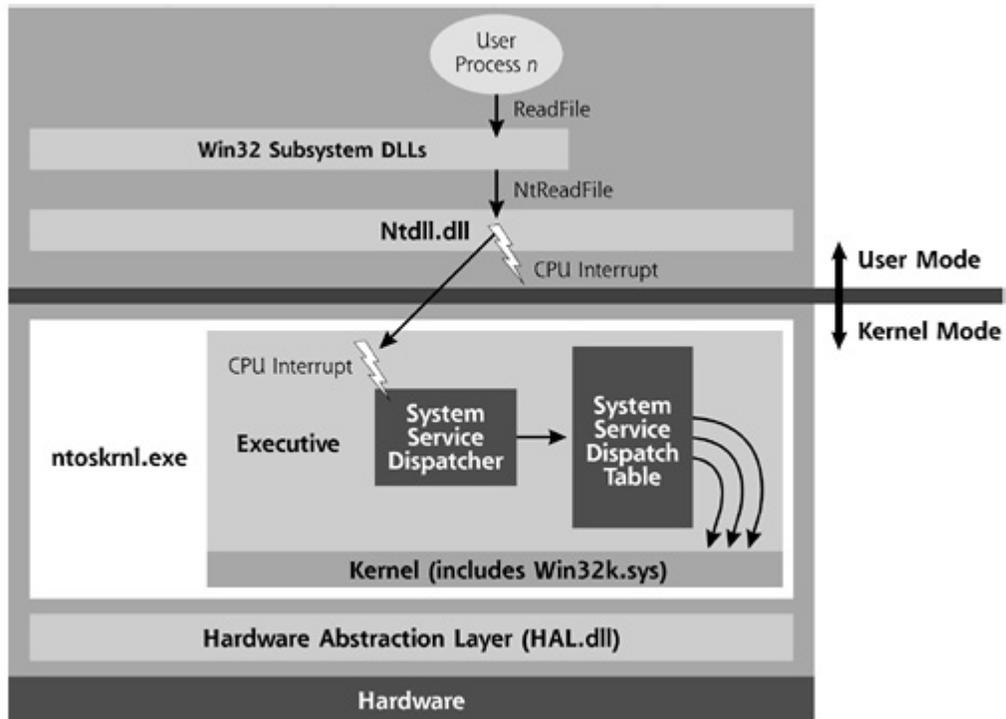


Figure 2.5: System Service Dispatching General Structure

2.2 User Mode Hooking Techniques

Hooking techniques are categorized according to where the hook is placed. (i.e., inside user space or kernel space) In the case of user mode hooking, ghostware does not bother dealing with kernel code and directly attack user processes or Win32 subsystem DLLs as shown in Figure 2.3. Instead it targets the APIs a program uses. This makes sense when you consider that user applications must rely upon the operating system to provide valuable functions such as opening files and writing to the registry.

2.2.1 Import Address Table Hooking

Windows uses a format called Windows Portable Executable (PE) format [35] for executable files. This standard allows programs to run in any version of Windows without the need of recompiling them. In order to provide this flexibility, PEs use dynamic symbol loading by using indirect addresses. Any call to external functions, such as from the ones in Win32 DLLs, are compiled so that the CALL uses a memory address to take the call address from. When the operating system loads the executable, it resolves all the external symbols and writes their addresses to these memory locations [14]. Whenever an application uses an API function exported from a DLL such, the compiler creates data structure called IMAGE_IMPORT_DESCRIPTOR in the application. this structure contains the name of the DLL from which the function is exported and a pointer to the Import Address Table (IAT), which contains all of the functions exported by the DLL that are used by the application. The table is filled with IMAGE_THUNK_DATA structures, which consists of the memory addresses of the desired functions all of which are filled by the Windows loader. When an application wants to make a call to an imported function, first the program code calls into the IAT. After reading destination address of the real function, another jump is made. By simply modifying this table, a ghostware can re-route program execution through itself, which allows filtering or manipulation of data [51].

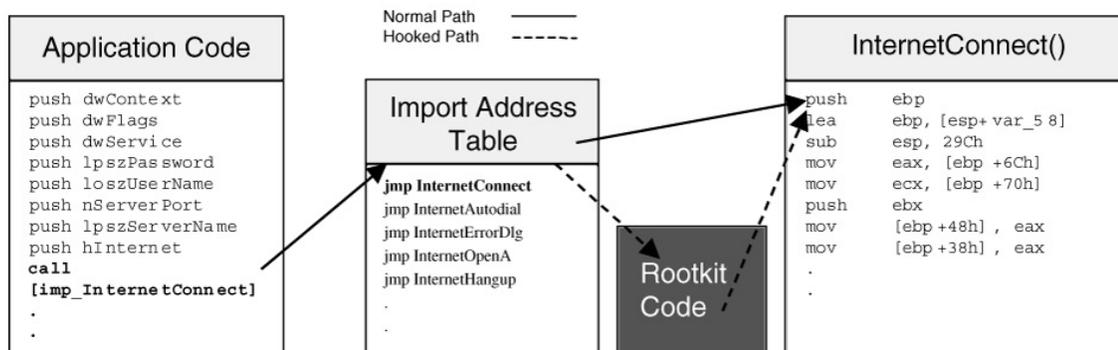


Figure 2.6: Import Address Table Hooking

2.2.2 Inline Function Hooking

An inline function hook is made by overwriting the code bytes of a target function in a process so that whenever the function is called, it will first call the attacker's function, which in turn calls target function. There are two approaches in inline function hooking. The first one is overwriting the actual executables or libraries on hard disk. However, this approach requires disabling the Windows File Protection (WFP), if the target function is a Win32 DLL or executable. When any directory containing sensitive Windows files (e.g., the System32 directory) is changed, the system signals WFP, invoking its functionality to check the digital signature of the changed file. If the signature does not match a Microsoft-approved value stored in the registry, WFP replaces the file with the proper Microsoft version of the file [30]. The second approach is modifying the code in memory. First attacker saves the first several bytes of the target function in what is called a trampoline. Then he/she replaces these bytes with an immediate jump to a place called detour. Detour calls the trampoline, which jumps to the target function plus saved bytes. When the target function does its job and returns, the detour can modify or filter the results and return to the source function that originally called the target function [24]. The sequence is illustrated in Figure 2.7

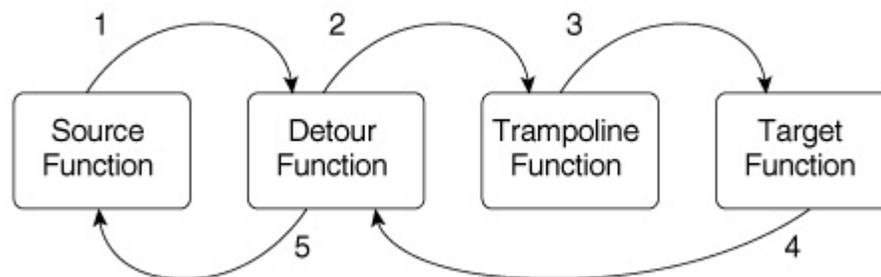


Figure 2.7: Inline Function Hooking

2.2.3 DLL Injection

DLL injection is forcing an unsuspecting running EXE process to accept a DLL that it never requested. The malicious DLL will be loaded into the memory space of the executable which allows the DLL to reach all data of the target. The idea is first brought up by [39]. There are two techniques used for DLL injection.

First one is to use hooks. Windows API has a function called *SetWindowsHookEx*, which makes it possible to hook window messages in another process, which will effectively load any malicious DLL into the address space of that other process. *SetWindowsHookEx* is defined like this:

```
HHOOK SetWindowsHookEx(  
    int idHook,  
    HOOKPROC lpfn,  
    HINSTANCE hMod,  
    DWORD dwThreadId  
);
```

The first parameter is the type of event message that will trigger the hook. This can be a hook procedure that monitors keystroke messages. The second parameter identifies the address to be called when a hook is triggered. The virtual-memory address of the DLL that contains this function is the third parameter. The last parameter is the thread to hook. Using this method, whenever a process is about to receive a keyboard event, the specified DLL will be loaded [22].

The second technique is using remote threads. [12] provides a step by step explanation of implementation and using Windows API for this purpose:

- Allocate space in the victim process for the DLL code to occupy. There is a built-in function in the Windows API to accomplish this task, called *VirtualAllocEx*.

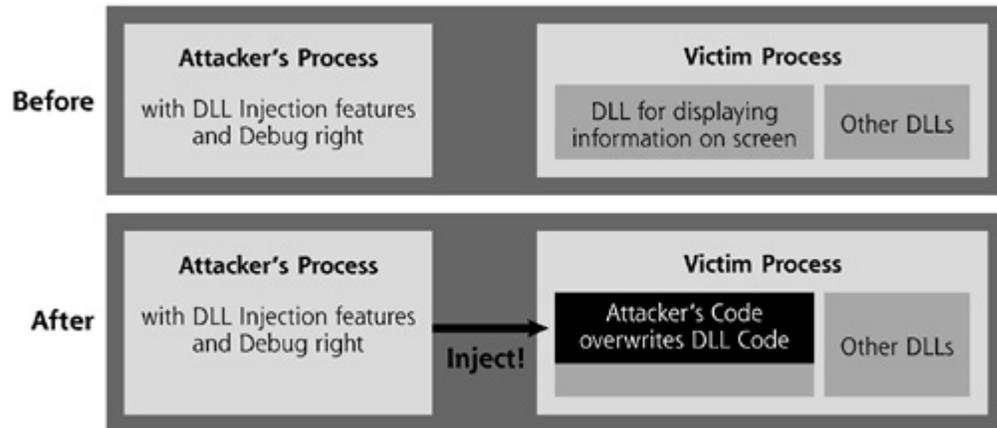


Figure 2.8: DLL Injection

- Allocate space in the victim process for the parameters required by the DLL to be injected. This can be done using the built-in Windows *VirtualAllocEx* function call, too.
- Write the name and code of the DLL into the memory space of the victim process. The *WriteProcessMemory* function of Windows API call can be used to write arbitrary data into the memory of a running process.
- Create a thread in the victim process to actually run the newly injected DLL. The *CreateRemoteThread* function in Windows API starts an execution thread in another process, which will run any code already in that process, including a newly injected DLL.
- Free up resources in the victim process after execution is completed. The resources consumed by this technique can be freed after the victim thread or process finishes running, using the *VirtualFreeEx* function of Windows API.

2.3 Kernel Mode Hooking Techniques

User mode hooking is useful and easy to implement, but also it is relatively easy to detect and prevent. With a user mode ghostware, the attacker has to break into the system and modify a number of programs to stay resident and consider all possible ways that can reveal itself. Moreover, since anti-malicious programs run in kernel mode, they will be one step ahead and ghostware will have a lower chance of survivability. A better solution would be installing a kernel memory hook. A kernel mode hook and consequently kernel memory access rights will make the ghostware on equal footing with any detection software. What is more, the attacker just modifies the kernel so that it lies to any particular command or program run by the administrator looking for that file. In this way, kernel mode hooking is far more efficient.

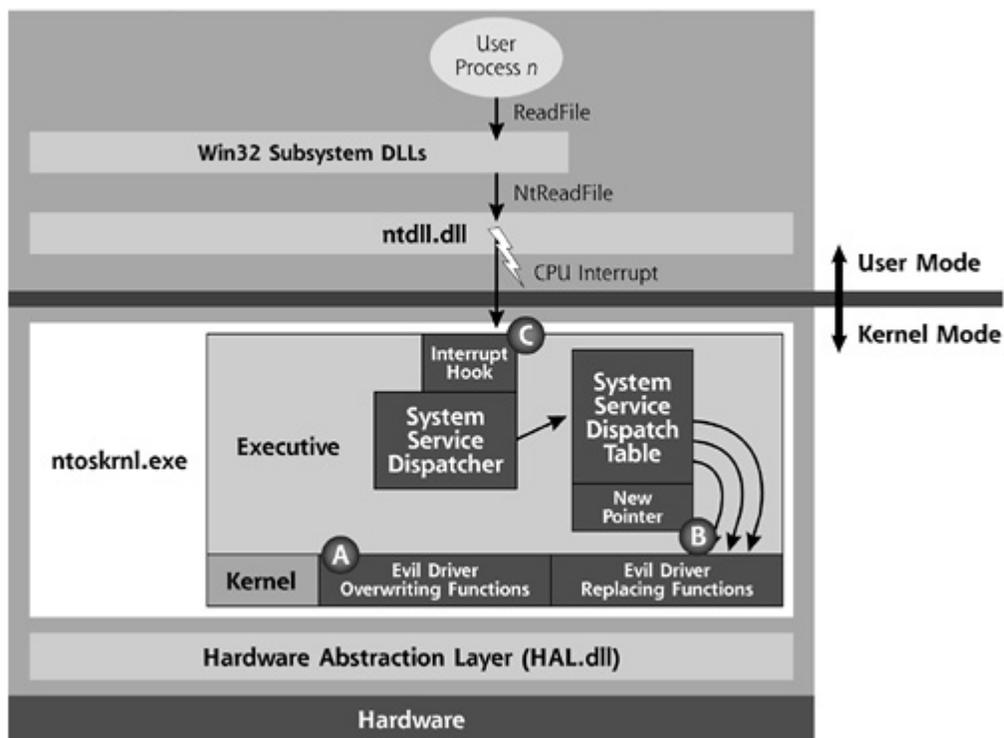


Figure 2.9: Kernel Mode Hooking Techniques

2.3.1 System Service Dispatch Table (SSDT) Hooking

As we have explained in section 2.1, System Service Dispatch Table (SSDT) plays an important role in system service call mechanism provided by the kernel, for letting the user-mode code use its services. For example, whenever a user-mode application needs access to files, registry or process objects, it calls the appropriate Windows API call, which eventually generates a system service call that is then handled by the kernel. First introduced by [40], SSDT hooking is a powerful and widely adopted kernel-mode technique. Attacker simply changes the content of the table and puts the hook function's address instead of the address of the internal kernel function that implements the corresponding service. After that, any call to the specific system service can be intercepted. [25] The process is illustrated in Figure 2.10.

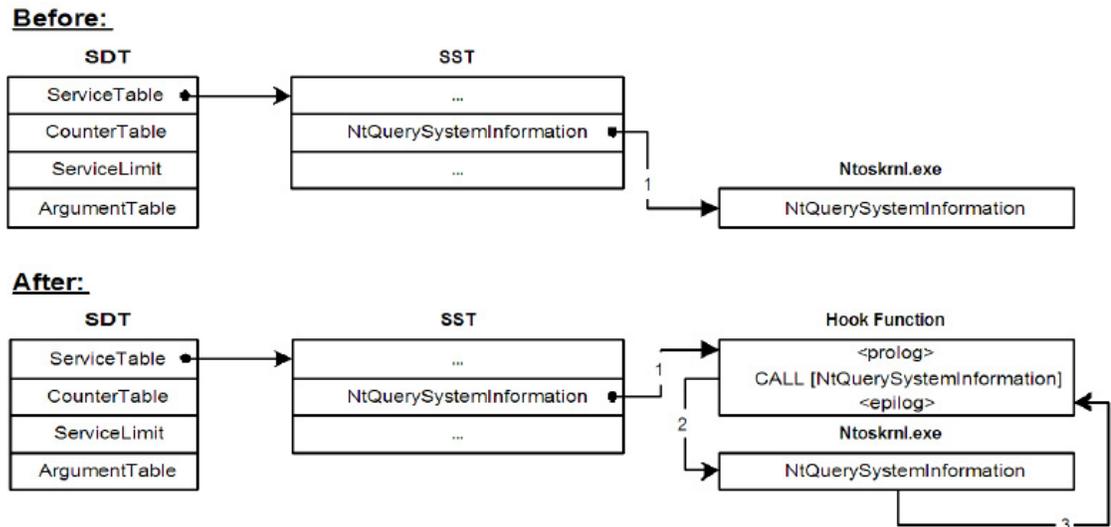


Figure 2.10: System Service Dispatch Table Hooking

Some versions of Windows come with write protection enabled for certain portions of memory. This becomes more common with later versions, such as Windows XP and Windows 2003. These later versions of the operating system make the SSDT read-only because it is unlikely that any legitimate program would need to modify this table. However, as we commented before, a kernel

mode ghostware is on equal footing with any kernel program, so this was not enough to stop hooking. By simply modifying the WP bit in the processor's CR0 register, write protection can be overwritten [41].

2.3.2 Interrupt Descriptor Table (IDT) Hooking

Interrupt Descriptor Table (IDT) is used to handle interrupts. Interrupts can originate from software or hardware. In the system call flow, *Ntdll.dll* triggers interrupt number 0x2E to invoke kernel, which happens right before usage of SSDT. The IDT specifies how to process interrupts, so a kernel mode hooking can be realized by writing address of the hook function for interrupt number 0x2E. Unlike other hooks, execution control does not return to the IDT handler, so the typical hook technique of calling the original function, filtering the data, and then returning from the hook will not work. The IDT hook is just a pass-through function and will never regain control, so it cannot filter data. Consequently this technique can only be used for identifying or blocking requests [22].

2.3.3 Input/Output Request Packet (IRP) Function Table Hooking

Another possible location for kernel mode hooking is function tables in device drivers. Whenever a driver is installed, it initializes a table of function pointers that have the addresses of its functions to handle different types of I/O Request Packets (IRPs). IRPs are used for several types of requests, such as reads, writes, and queries.

The driver and IRP type to hook depends on attackers purpose. For example, this technique can be used for hiding TCP or UDP ports. you could hook the functions dealing with file system writes or TCP queries. However, just like the IDT, major IRP handling functions do no return the execution flow, so normally filtering the results is not possible. In order to use IRP for altering a query, attacker changes the control flags, which enables execution of a callback function

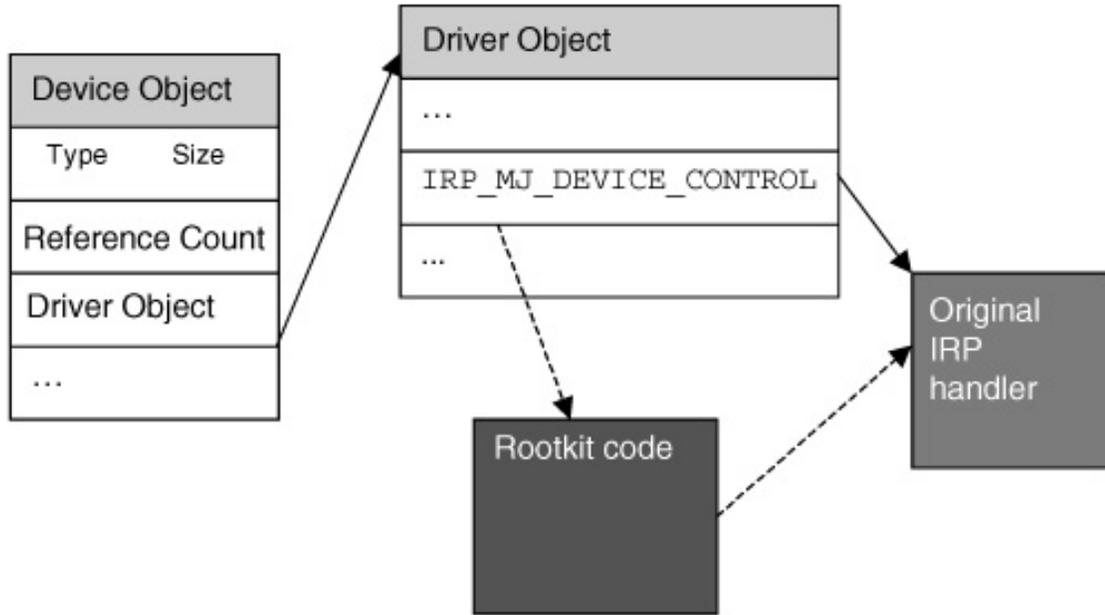


Figure 2.11: I/O Request Packet Function Table Hooking

after IRP handler. I/O Manager will call the *IoCompletionRoutine* of the IRP and after the handler has successfully finished processing the IRP and filling in the output buffer with the requested information. By hooking the *IoCompletionRoutine* attacker can filter query results [22].

2.4 Direct Kernel Object Memory Access

Instead of using a device driver, an attacker could directly patch the kernel in the memory of the victim machine, a technique first described in detail by [21]. The technique is built upon the memory handling in Windows, specifically with regard to the CPU running in Ring 0 and Ring 3. The Global Descriptor Table (GDT) contains information about how memory is divided into various segments, allocated to user programs and the kernel itself. All memory locations between 0x80000000 and 0xC0000000 are for use by the kernel and restricted to user-mode

processes. The GDT stores data about how various memory segments are placed and access rights for each memory segment. The tricky part of GDT is, same range of memory addresses can be defined in multiple segments. By use of some Windows API weaknesses, a malicious process can add a new entry to the GDT, thereby describing a new segment that maps to a whole memory range, that is, a memory space starting at 0x00000000 and going to 0xFFFFFFFF. Using this technique, the malicious process can access everywhere in kernel memory.

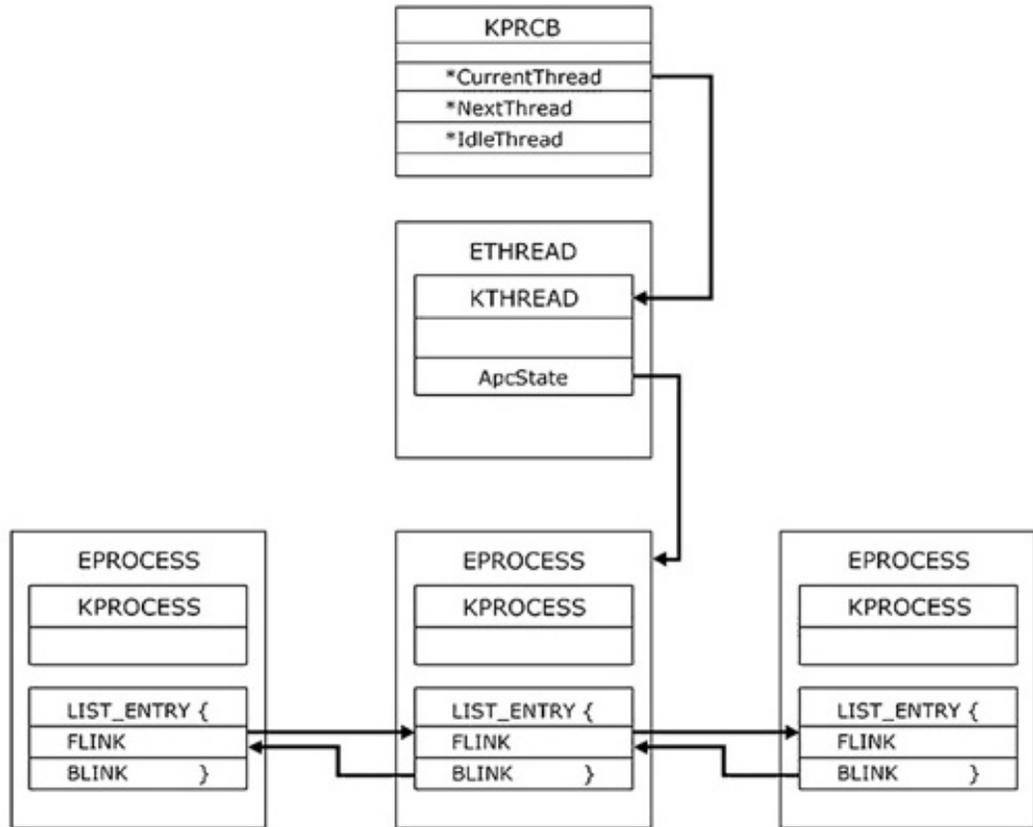


Figure 2.12: Process List Before Modification using DKOM

After gaining access rights for reaching every part of memory, a malicious process can hide processes, elevate their privilege levels, or perform other modifications. As an example, the Windows operating system’s list of active processes is obtained by traversing a doubly linked list referenced in the **EPROCESS** structure of each process. When a user-mode application sends a request for the process

list, the appropriate system service function traverses the linked list and sends the data back to the client. Specifically, a process's EPROCESS structure contains a LIST_ENTRY structure that has the members FLINK and BLINK. FLINK and BLINK are pointers to the processes in front of and behind the current process descriptor, as shown in Figure 2.12 [7].

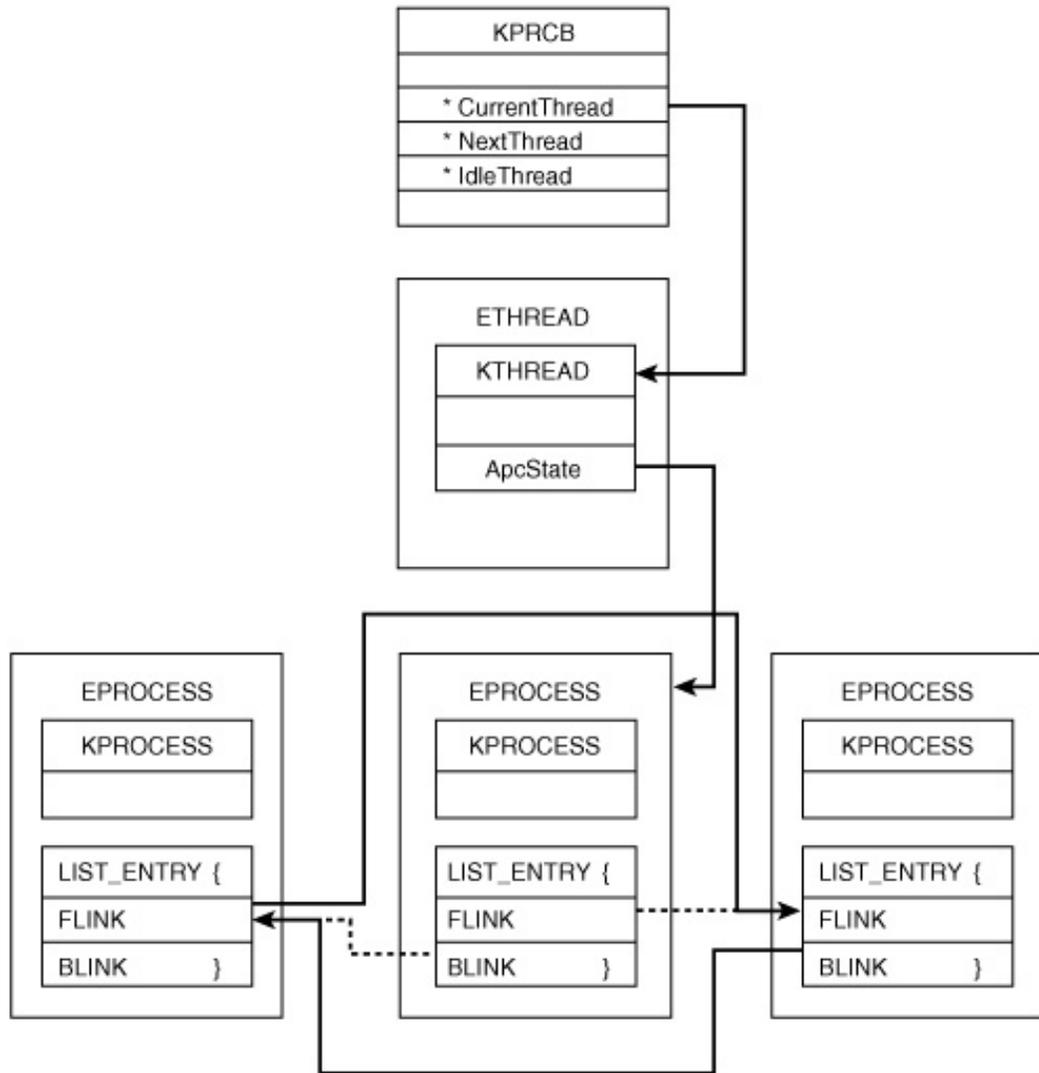


Figure 2.13: Process List After Modification using DKOM

By simply changing the FLINK and BLINK pointer values of the forward and rearward EPROCESS blocks to by-pass a process, we can make that process

invisible, as shown in Figure 2.13.

Chapter 3

Ghostware Detection Techniques

Most of the malicious software detection techniques are based on the signature matching approach used by anti-virus software. Each file execution and program installation is investigated to determine file signatures for use by malicious code scanner. In the case of spyware, this technique will only work during the loading phase, which allows the anti-virus program to quarantine the file so that it cannot be executed or installed. If the spyware is using stealth techniques and it was loaded before the activation of anti-virus program, then standard detection techniques will fail, since a ghostware can potentially by-pass the scanner. With the appearance of the stealth techniques, detection softwares become pretty weak and developers decided to improve detectors by adding forensic tools and behavioral detectors. In this chapter we will explore the stealth detection techniques used/usable for detecting ghostware.

There are two basic approaches to stealth detection. The first approach looks for the hiding mechanism such as API interceptions or hooks. There are three problems with this approach. First, there are many loading methods using different entry points for hooking. The detection software need to look for all of these points and updated frequently. This was the problem with 'Pedestal Software's Integrity Protection Driver' [34], which is no longer supported by the developer. The second problem is, it may catch many false positives due to legitimate uses of API hooking by detectors, debuggers or in-memory software patching. And

finally, if ghostware is capable of tracking detection attempts, it may use its hooks for deceiving the detector.

Second technique is 'cross-view based detection'. Detector snapshots two view of the system, a "high level" view and a "low level" view. The high level view will show what the stealth software wants the user to see, with hidden files, processes and other objects out of sight. The low level view is will contain everything actually present on the system, including the hidden objects. The difference of these two views, if exists, will be hidden objects.

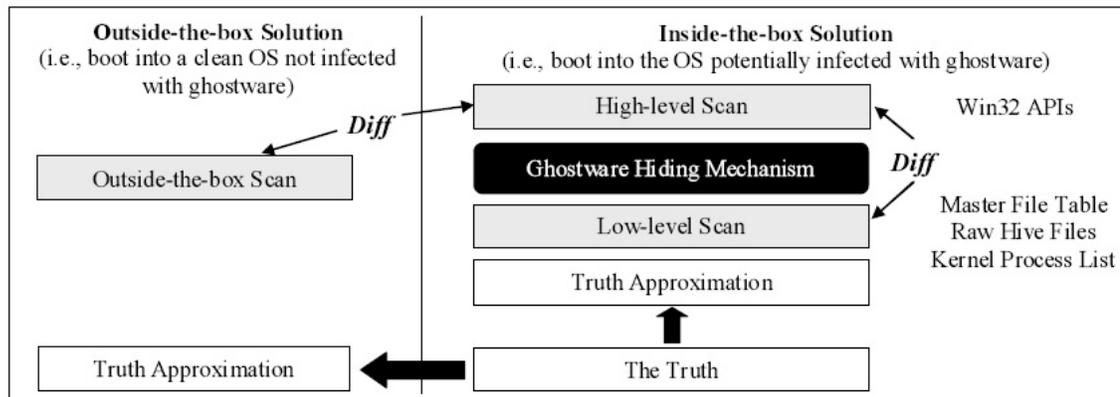


Figure 3.1: Cross-view Detection Mechanism

The main problem with this approach is getting the views. First of all, getting a low level view may not be easy especially for every object type. The operating system need to be clean and yet there is not enough documentation for getting a low level view. The technique must work deeper than any hooking technique to get the data untampered. This can mean replicating or manipulating operating system functionality or taking advantage of undocumented data structures to acquire this view. Moreover, the 'high level' view must be tainted. One API function may not see a hidden file and yet others may be unaffected, so the the file may be hidden only from a certain perspective, which high level view must consider. Also, a ghostware can temporarily reveal the objects it was hiding, after noticing a detection attempt, detector should not reveal itself.

3.1 Hook Detection

There are many places where a hook can be hidden, as we explained in Chapter 2, including:

- Import Address Table (IAT)
- System Service Dispatch Table (SSDT)
- Interrupt Descriptor Table (IDT) with one per CPU
- Drivers' I/O Request Packet (IRP) handler
- In-line function hooks

The basic algorithm for identifying a hook is tracing the execution flow and looking for branches that fall outside of an acceptable range. Such branches would be produced by instructions like *call* or *jmp*. Defining the acceptable range maybe a problem, depending on the situation.

In a process Import Address Table (IAT), the name of the module containing imported functions is listed. This module has a defined start address in memory, and a size which defines the acceptable range. All legitimate I/O Request Packet (IRP) handlers should exist within a given driver's address range, and all entries in the System Service Dispatch Table (SSDT) should be within the address range of the kernel process, *Ntoskrnl.exe*. Interrupt Descriptor Table (IDT) hooks can be a problem since there is no well defined address range here except the INT 2E handler, which should point to the kernel, *Ntoskrnl.exe*. In-line hooks are quite hard with this technique, since imported functions can be in another module. Detector may need a complete disassembly of the function in order to find check validity of address. VICE [5] uses this technique for detecting ghostware [22].

Another way for finding hooks in APIs and in system services is tracing and counting execution calls. The idea is quite simple and elegant, if the ghostware has compromised the system and trying to hide something by changing some execution path, then system will be executing extra instructions, during some typical

system and library calls. In order to implement instruction counting, detector can use a nice feature of Intel processors, the so called single stepping mode. When the processor is working in this mode, it will generate debug exception after every instruction which was executed [45]. After recording execution path in a clean system, the detector can perform execution tests checking to see whether additional instructions have been executed in subsequent calls when compared to the baseline. This method is implemented in Patchfinder [44].

Another detection technique is employed by modGREPER [42], which searches through whole kernel memory (0x80000000 0xffffffff) in order to find structures which looks like a valid module description objects. It recognizes specific structure types used in hooking like `_DRIVER_OBJECT` and `_MODULE_DESCRIPTION` and builds a list of found objects. After that, modGREPER matches them to each other and finally compares this list against the list of kernel modules obtained with documented API.

Another detection technique is based on the fact that code sections are read-only in kernel memory, so programs should not modify their code under normal circumstances. System Virginty Verifier (SVV) [43] uses this technique for detecting ghostware. SVV checks if the code sections of important system DLLs and system drivers (kernel modules) are exactly same in memory and on disk. The difference can reveal a hook as shown in Figure 3.2. However this can also be possible due to page faults and memory relocations, so this technique needs careful handling of all possible OS modifications.

3.2 Hidden File Detection

Windows File Protection (WFP) provides some protection against file changes, but employing additional file integrity checking tools for intrusion detection or hidden file detection is a good practice against ghostware. The most famous file integrity checking tool is Tripwire [56]. These tools look for changes to critical system files and registry settings based on cryptographic hashes of known

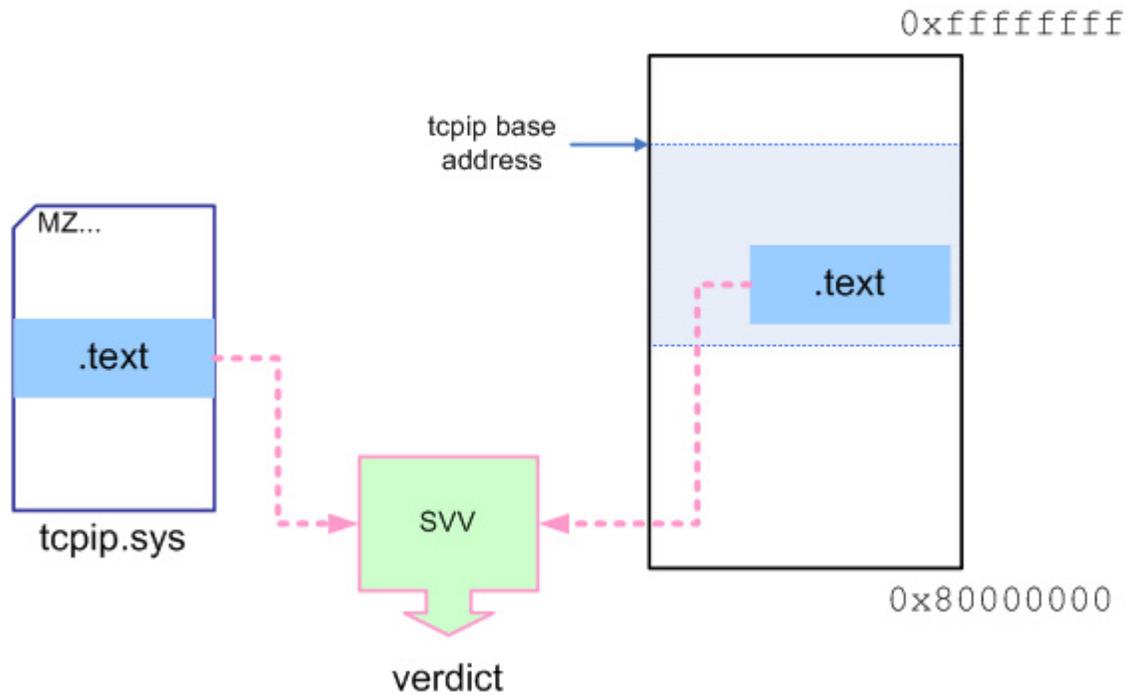


Figure 3.2: System Virginty Verifier

good files and settings. By this way, they can spot file changes made to the operating system files, probably by a kernel-mode ghostware.

Another tool for ghostware detection is Rootkit-Revealer [10]. It can detect hidden registry entries as well as hidden files using cross-view difference. For the low-level view, Rootkit-Revealer parses the files that correspond to the different Registry hives without the aide of the standard Win32 API calls, such as *RegOpenKeyEx* and *RegQueryValueEx*. It also parses the file system at a very low level, avoiding the typical API calls. High-level view is taken by the system API task manager uses. Comparison of these views reveal hidden objects.

Another detection tool is developed by Microsoft Research, GhostBuster [60], which implements the cross-view difference based detection for files, registry and processes. GhostBuster can perform low-level and high-level scans as an inside-the-box solution, while it can also be used as an outside-the-box solution, performing low-level scan during a Windows Preinstallation Environment [28] CD

boot. GhostBuster uses the *Master File Table*, the *Raw Hive Files*, and the *Kernel Process List* as the low-level resources to detect hidden files, registry entries, and processes, respectively [58].

3.3 Hidden Process Detection

One way of detecting hidden process is through 'CSRSS.EXE'. Detector can track the handles in CSRSS.EXE and identify the processes to which they refer and use this as a low-level view for cross-view difference based detection. Every process's EPROCESS block contains a pointer to a structure that is its HANDLE_TABLE. [41] These handle table structures are linked by a LIST_ENTRY, similarly to the way all processes are linked by a LIST_ENTRY. By finding the handle table for any process and then walking the list of handle tables, detector can identify every process on the system. This technique is used by F-Secure BlackLight [16].

Last detection technique is specially effective against hidden processes hidden by DKOM as explained in Chapter 2.4. The *SwapContext* function in *Ntoskrnl.exe* is called to swap the currently running thread's context with the thread's context that is resuming execution. When *SwapContext* has been called, the current thread address and next thread address are saved. By using the Detour technique in Chapter 2.2.2 for verifying that the KTHREAD of the thread to be swapped in points to an EPROCESS block that is appropriately linked to the doubly linked list of EPROCESS blocks, one can detect hidden processes [7].

Chapter 4

Experiments and Evaluation

After explaining the stealth techniques in Chapter 2 and detection techniques in Chapter 3, now we will investigate the practical effectiveness of detection techniques against current ghostware programs.

4.1 Methodology

The test case consists of both kernel-mode ghostware and user-mode ghostware. For the kernel-mode ghostware, we implement a modular structure for injecting hooks. The hook codes are implemented as system drivers and there is a loader/unloader system at the core using the Service Control Manager (SCM). When a driver is loaded using the SCM, it is non-pageable and IRP-handling functions, and other important code will be resident on the memory. Figure 4.1 shows the design structure.

The kernel-mode ghostware includes components for hiding processes, files and ports. For process hiding it uses the SSDT hooking technique explained in Chapter 2.3.1 and DKOM technique used by FU rootkit [18] as explained in Chapter 2.4. For both techniques we define a keyword, which is “_cool_” in our case, and hide every process that includes this string in its name. For file hiding we

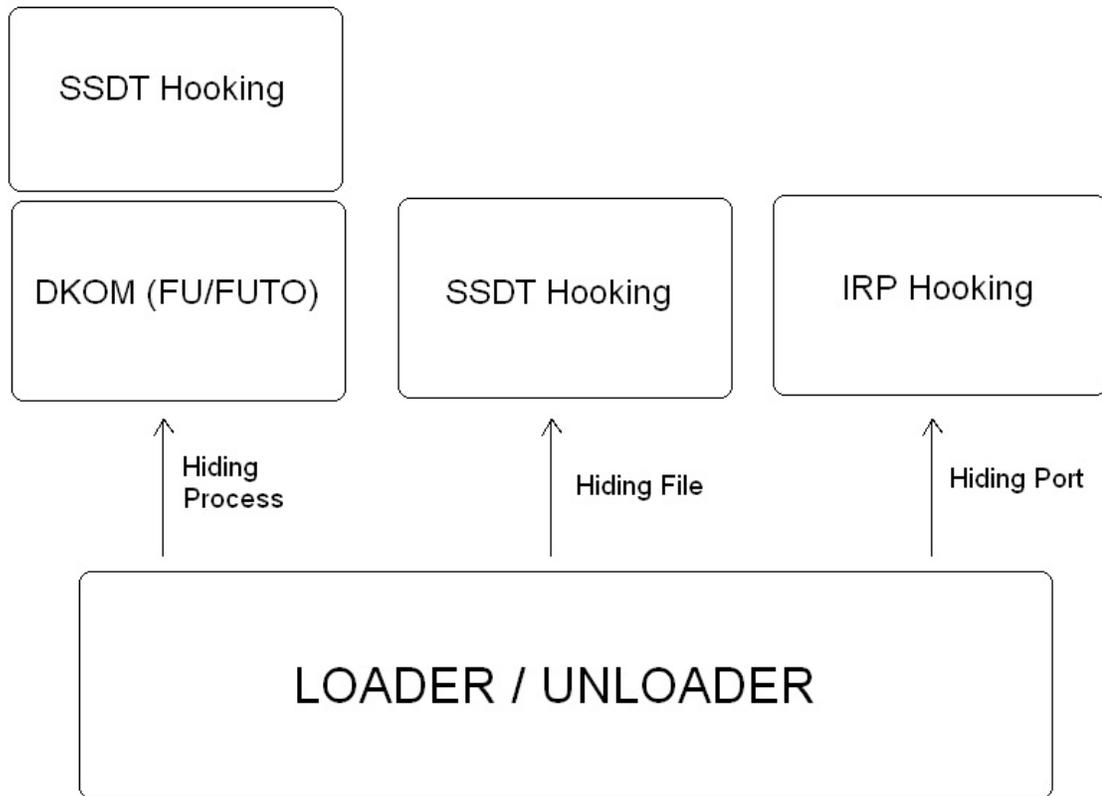


Figure 4.1: Test Case Kernel-Mode Ghostware Structure

also use the SSDT hooking technique. We hide every file or folder which includes “_cool_” in its name and of course all file names of this program starts with “_cool_”. For the port hiding case, we used the IRP hooking technique explained in Chapter 2.3.3, hiding port 80 in TCP and UDP connections. Furthermore, we also include FUTO [48] in our tests, an expansion to FU rootkit which handles some weaknesses in FU by manipulating more tables, like PspCidTable, for hiding objects.

In the case of user-mode ghostware, instead of implementation, we decided to use the best user-mode hacker toolkit, Hacker Defender [20], for our tests. It is capable of hiding processes, files, ports, services and registry keys using DLL injection as explained in Chapter 2.2.3. This is an open source project and we recompiled all the code before testing it. We also changed driver name,

service names, process names and polymorph the executable with a polymorphic encryptor called Morphine [29] for avoiding easy detection. Similar to the kernel-mode ghostware, this hides every process or file or folder with the name `hxdef` in it.

The test bed is a virtual machine using VMWare [57] software loaded with Windows XP operating system. The test cases can be categorized into three parts. We first used the best three anti-virus software, which are claiming to be highly effective in detection and prevention of spyware. (namely, Symantec Norton Anti-Virus 2006 [54], McAfee Anti-Virus/Anti-Spyware Program [27] and Panda Anti-Virus+Anti-Spyware Titanium 2006 [33])

The tests are performed in three phases. First we load ghostware, install anti-virus program and make a full scan to see if the program can detect previous infections. We call this a “pre-load” test. Then we restart the computer and run ghostware while anti-virus program is active and monitoring the computer, to see if the program can detect any hooking or loading attempt. We call this a “load” test. Finally we scan the folders of ghostware, without running them, in other words make a manual scanning directly to the malicious code files to see if the program can detect the malicious code by signature.

We also run tests with anti-spyware programs by picking the three most popular commercial ones, which are Lavasoft Adaware SE Professional 1.06 [26], Bulletproofsoft Spyware And Adware Remover v9.3 [4] and Etrust Pestpatrol Anti-Spyware 2005 [15]. We also include a widespread free tool called Spyware Doctor v4.0 [52]. We apply the same testing approach to these files. Finally we try detection with the tools mentioned in Chapter 3 such as: VICE, Rootkit-Revealer, System Virginty Verifier and Flister. The implementers of FU rootkit suggested another detection technique that we can call as Swap Context-Detour Patching and provided some coding, which we included in our tests. We also included F-Secure BlackLight [16], a commercial hidden process and hidden file detector. Furthermore we made tests with IceSword, which is a ghostware detector in development phase originating from China and RAIDE [6], an academic work for rootkit detection. All these tools can only work as detectors so we run

and test them after the ghostware is loaded and running.

4.2 Test Results

The anti-virus test results, is shown in Table 4.1. The tests show that anti-virus (AV) programs are weak against ghostware. First of all one would expect the signature matching algorithms of these programs to be stronger. We used the exact FU rootkit hooking code for process hiding with DKOM, and we only recompiled it, yet none of the AV programs managed to detect it when we scan the files. On the contrary, quite surprisingly, Norton managed to detect FUTO, an expansion of FU. Furthermore, McAfee and Norton managed to detect system driver and the config file of Hacker Defender, which we could not polymorph. In our opinion, standard signature matching algorithms are not reliable for detecting ghostware and this is why they need detection by behavior. Only Norton anti-virus employed this technique and it did not catch the hooking behavior, instead it caught the encryption/decryption code we used for Hacker Defender. Norton quarantined the executable saying it may contain malicious code, which allowed detection of Hacker-Defender while loading. None of the anti-virus programs include any method for scanning hidden files or processes, so none of them worked in the pre-load test.

The anti-spyware test results, are shown in Table 4.2. The tests show that anti-spyware (AS) programs, in general, are not capable of removing or detecting ghostware. These programs seem to be designed for detection of adware. They include features like Browser Helper Object (BHO) scanning, registry scanning, or file scanning for known adware signatures. None of them was able to detect our samples except Spyware Doctor, which detected user-mode ghostware and FU code. Furthermore, Spyware Doctor managed to detect the hidden process by FU in load and pre-load tests.

	User-Mode Hooking			Kernel-Mode Hooking			DKOM	
	Inj Proc.	Inj File	Inj Port	SSDT Proc.	SSDT File	IRP Port	DKOM FU	DKOM FUTo
Norton AV Pre-Load	X	X	X	X	X	X	X	X
Norton AV Load	✓	✓	✓	X	X	X	X	X
Norton AV Signature	✓	✓	✓	X	X	X	X	✓
McAfee AV Pre-Load	X	X	X	X	X	X	X	X
McAfee AV Load	X	X	X	X	X	X	X	X
McAfee AV Signature	✓	✓	✓	X	X	X	X	X
Panda AV Pre-Load	X	X	X	X	X	X	X	X
Panda AV Load	X	X	X	X	X	X	X	X
Panda AV Signature	X	X	X	X	X	X	X	X

Table 4.1: Anti-Virus Programs Detection Test Results

	User-Mode Hooking			Kernel-Mode Hooking			DKOM		
	Inj Proc.	Inj File	Inj Port	SSDT Proc.	SSDT File	IRP Port	DKOM FU	DKOM FUI	DKOM FUIo
Lavasoft AS Pre-Load	X	X	X	X	X	X	X	X	X
Lavasoft AS Load	X	X	X	X	X	X	X	X	X
Lavasoft AS Signature	X	X	X	X	X	X	X	X	X
BulletPS AS Pre-Load	X	X	X	X	X	X	X	X	X
BulletPS AS Load	X	X	X	X	X	X	X	X	X
BulletPS AS Signature	X	X	X	X	X	X	X	X	X
PestP. AS Pre-Load	X	X	X	X	X	X	X	X	X
PestP. AS Load	X	X	X	X	X	X	X	X	X
PestP. AS Signature	X	X	X	X	X	X	X	X	X
Spy.Doc. AS Pre-Load	X	X	X	X	X	X	√	√	√
Spy.Doc. AS Load	X	X	X	X	X	X	√	√	√
Spy.Doc. AS Signature	√	√	√	X	X	X	√	√	√

Table 4.2: Anti-Spyware Detection Test Results

Table 4.3 shows the ghostware and rootkit detection tools test results. Some tools are made for specific purposes, such as Rootkit-Revealer only searches for hidden files and hidden registry entries. As a result it is not included during tests of techniques that hide process and this is shown as 'n.a.' in the results. The techniques in the table (the columns) can be thought to be more complex and hard to detect from left to right. In this context, the trend of having more successful detection on the left and having less detection on the right makes sense. VICE, SVV and ModGrepper are old tools and they were released before DKOM techniques were invented. As a result, although they are quite successful in user-mode techniques and kernel-mode techniques, they have no use against DKOM techniques. ModGrepper did not work at all, so we think it was build for specific purposes, i.e., to detect specific techniques, which we did not test.

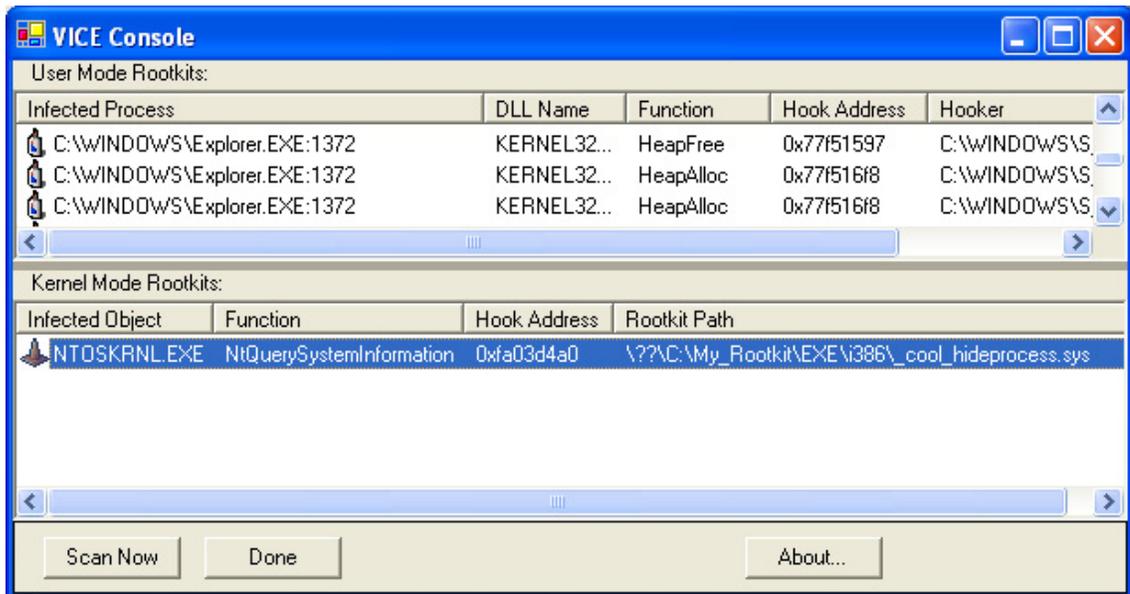


Figure 4.2: Vice detecting SSDT hooking

```

C:\WINDOWS\System32\cmd.exe

C:\Detectors>svu.exe check /m
ntoskrnl.exe (804d0000 - 806b3f00)... innocent hooking (verdict = 2).
module ntoskrnl.exe [0x804d0000 - 0x806b3f00]:
  0x804d4f8e (section .text) 1 byte(s): exclusion filter: KiSystemCallExitBranch() [05->06]
  file :05
  memory :06
  verdict = 1

  0x804f01d2 (section .text) 18 byte(s): exclusion filter: KeFlushCurrentTb()
  file :d8 0f 22 d8 c3 0f 20 e0 25 7f ff ff ff 0f 22 e0 0d 80
  memory :e0 25 7f ff ff ff 0f 22 e0 0d 80 00 00 0f 22 e0 c3
  verdict = 1

  0x804f01ea (section .text) 1 byte(s): exclusion filter: KeFlushCurrentTb() [c3->00]
  file :c3
  memory :00
  verdict = 1

  0x804fc8d8 [KiServiceTable[173]] 4 byte(s):
  KiServiceTable HOOK:
  address 0xfa03d4a0 is inside _cool_hideprocess.sys module [0xfa03d000-0xfa03e000]
  target module path: \??\C:\My_Rootkit\EXE\i386\_cool_hideprocess.sys
  file :ba a7 57 80
  memory :a0 d4 03 fa
  verdict = 2

  0x8050fa89 (section .text) [RtlPrefetchMemoryNonTemporal()+0] 1 byte(s): exclusion filter: RtlPrefetchMemoryNonTemporal() [c3->90]
  file :c3
  memory :90
  verdict = 1

module ntoskrnl.exe: end of details

SYSTEM INFECTION LEVEL: 2
 0 - BLUE
 1 - GREEN
--> 2 - YELLOW
 3 - ORANGE
 4 - RED
 5 - DEEPRED

```

Figure 4.3: System Virginty Verifier by Ruanna Rutkowska

	User-Mode Hooking			Kernel-Mode Hooking			DKOM	
	Inj Proc.	Inj File	Inj Port	SSDT Proc.	SSDT File	IRP Port	DKOM FU	DKOM FUTo
VICE	√	√	√	X	√	X	X	X
Rootkit-Revealer	n.a.	√	n.a.	n.a.	√	n.a.	n.a.	n.a.
SVV	√	√	√	√	√	X	X	X
ModGrepper	X	X	X	X	X	X	X	X
Flister	n.a.	X	n.a.	X	n.a.	n.a.	n.a.	n.a.
SwC-Detour	√	n.a.	n.a.	√	n.a.	n.a.	√	X
BlackLight	√	√	n.a.	√	√	n.a.	√	X
IceSword	√	√	√	√	√	√	√	X
RAIDE	x	x	x	√	√	n.a.	√	√

Table 4.3: Ghostware and Rootkit Detection Tools Test Results

	User-Mode Hooking			Kernel-Mode Hooking			DKOM	
	Inj Proc.	Inj File	Inj Port	SSDT Proc.	SSDT File	IRP Port	DKOM FU	DKOM FUTo
Data Mining	√	√	√	√	X	X	X	X

Table 4.4: Data Mining Technique Test Results

Swap Context-Detour Patching technique was quite successful for detecting all kinds of hidden processes except the one used by FUTo. Figure 4.4 shows hidden process detection by this technique.

```

c:\ Command Prompt
Process list by traversal of HandleTableList
4 - System
228 - cmd.exe --[Hidden]--
464 - dllhost.exe
516 - smss.exe
580 - csrss.exe
604 - winlogon.exe
652 - services.exe
664 - lsass.exe
844 - svchost.exe
944 - svchost.exe
1036 - svchost.exe
1064 - svchost.exe
1332 - msdtc.exe
1344 - wuaclt.exe
1372 - explorer.exe
1460 - spoolsv.exe
1548 - VMwareTray.exe
1568 - VMwareUser.exe
1576 - ctfmon.exe
1584 - msmsgs.exe
1808 - VMwareService.e
3660 - ProcessDetector

```

Figure 4.4: Hidden Process Detected by Swap Context-Detour technique

Blacklight and Icesword also shine in our tests and they detected all techniques except FUTo, but we need to add the fact that FUTo was developed against these two tools. Especially IceSword offers various options to the user for detecting not only ghostware but also techniques used by other kinds of spyware such as BHOs. While Blacklight can only detect hidden processes and files, IceSword can detect registry entries, services, drivers, ports etc. In Figure 4.5, we can see IceSword revealing hidden files and folders.

Finally, we were particularly impressed by RAIDE, an academic research project on beta phase. RAIDE not only detects the hooks, it also offers options for removing the hook, deleting the hidden object etc., though, removal algorithms do not work well and we received a number of Blue Screen of Deaths(BSoD) during our testing. Figure 4.6 shows RAIDE command window detecting FUTo technique.

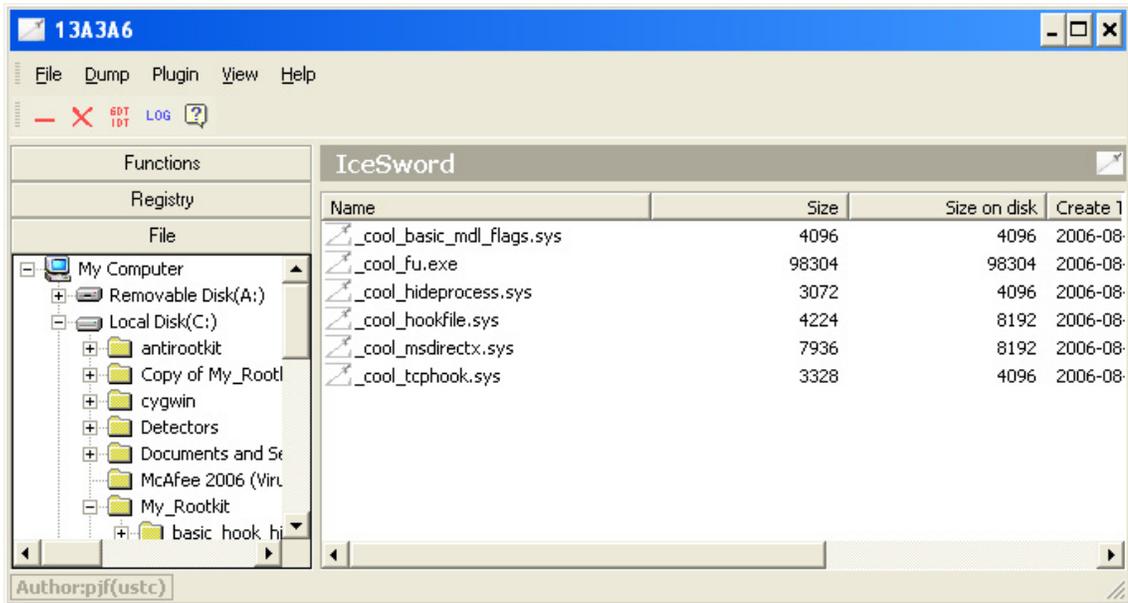


Figure 4.5: IceSword showing hidden files

The test results show us that anti-virus programs and anti-spyware programs, as they exist today, are ineffective against the ghostware threat. Even specific tools made for detecting hidden objects or hooks in system are not capable of detecting all of the rootkit or ghostware attack techniques.

4.3 Data Mining Approach

We have seen that signature based approaches are weak against new threads, so we also investigated the use of data mining based heuristic scanning technique proposed by [47]. The idea is, malicious executables have common intentions and they may have similar byte code. He claims that, we can detect malicious executables by looking at the frequency analysis of byte code in a file. We need to first form a model for byte frequencies by forming a two distinct datasets of benign and malicious executables. First we train our model using these datasets, and then we test it on new executables.

```

C:\WINDOWS\System32\cmd.exe
[-] Found 6 hidden process(es) on the system
[-] Check the system for hooks
[-] Finished checking the system for hooks
[-] Found 49 hook(s) on the system
[-] Found a hidden process. logonui.exe:2568 is hidden using PspCidTable Remove
method. This method is commonly used by FUTO.
[-] What action do you want to take against the process hidden using DKOM method
?
    1.      Dump the process address space to files on disk for furt
her analysis.
    2.      Do nothing
2
[-] Found a hidden process. msmsgs.exe:168 is hidden using PspCidTable Remove me
thod. This method is commonly used by FUTO.
[-] What action do you want to take against the process hidden using DKOM method
?
    1.      Dump the process address space to files on disk for furt
her analysis.
    2.      Do nothing
2
[-] Found a hidden process. msmsgs.exe:1732 is hidden using Direct Kernel Object
Manipulation (DKOM) method. This method is commonly used by FU.
[-] What action do you want to take against the process hidden using DKOM method
?
    1.      Relink the process
    2.      Close the process
    3.      Dump the process address space to files on disk for furt
her analysis.
    4.      Do nothing
2
[-] Found a hidden process. VMwareUser.exe:0 is hidden using PspCidTable Remove
method. This method is commonly used by FUTO.

```

Figure 4.6: RAIDE detecting FUTO hooking

We applied this technique on spyware programs by forming our own datasets in [3]. The dataset consisted of 312 benign(non-malicious) executables and 555 spyware executables. The malicious collection was formed by using a malicious code collection at [31] and manually collecting spywares by crawling the internet using a sandboxed operating system. The benign executables were collected from the system files in Windows XP operating system and from programs of a stereotype user. Byte sequences were extracted using the hexdump tool. For each file in the dataset, using this tool a hexdump file is formed. The tests were made using the 5-fold cross validation technique. According to the results, we had 91.28% accuracy and 4.92% false positive rate in our dataset, which was quite promising and data mining based heuristic scheme has the potential to be used for detecting new spyware programs.

Using the exact byte sequence model in our previous tests, we applied this technique on our ghostware samples. The test results in Figure 4.4 show that, our model was not very successful on detecting ghostware programs. This is probably due to the fact that our training dataset included not only ghostware type of spyware, but all kinds of spyware. However, the technique relies on the idea that executables with common intentions and will have similar byte code. We do not have enough number of ghostware samples to form a ghostware specific byte frequency model.

Chapter 5

Conclusion

When stealth features first appeared in computer viruses their main purpose was to make the work of anti-virus researchers and applications as difficult as possible. Today the apparent blending of malicious code writing and hacking gives stealth code a whole new perspective. Spyware has become a threat to corporate and personal information security. With the combined techniques of data interception and stealth, a spyware application can include all the functionality needed for a perfect information theft. Most of the current spyware lacks the advanced stealth techniques employed by modern windows rootkits, thus they are incapable of hiding their presence on a system. Nevertheless signature matching detection algorithms greatly reduced their effectiveness and this emerged spyware with rootkit techniques, or as we call it ghostware.

In this work we tested the most popular security tools along with specific detection tools against ghostware. The results show that, anti-virus programs or anti-spyware programs are not capable of detecting or removing ghostware applications. Hidden object detection or rootkit detection tools can be useful, however, these tools work after the computer is infected and they do not provide any means for removing the ghostware. As a result, we need understand the potential application of ghostware and implement new detection and prevention tools.

There is so much to be done in this field. First of all an official spyware definition should be announced which includes the ghostware programs. The lack of information misleads people and prevents them from taking precautions. The adware type of spyware hit users very hard because most of the ordinary users did not see what was coming. It is estimated that spyware infected 67% to 90% of computers connected to the Internet in year 2005. Panda Anti-Virus released an online Anti-Virus Scanner that can also detect Spyware in April 2005. According to their reports [55], in the first 24 hours of the operation, 84 percent of malicious code detected was spyware and the 74 most detected malicious code were all spyware programs.

We need new detection and prevention tools for fighting ghostware. As we explained in Chapter 3, the main approach for detecting a hidden object is comparing high level (infected) and low level (uninfected) views of the system. But getting a low level is hard, because there is not enough documentation for getting a low level view and the technique must work deeper than any hooking technique to get the data untampered. At this point, we can get help from the Operating System. If OS could export some lists for integrity checking (and the lists can include checksums for its own integrity checking), then detectors work could be quite easy.

Preventing an infection is quite tricky, because one needs to watch all the entry points to detect an intrusion. However, this is not possible due of the lack of documentation about this subject. Programs trying to close gates all failed when new entry points are found. The documentation of the Windows API should be better, similar to Unix, so that one can enumerate the possible ways for hooking APIs. This would allow creation of detectors that simply watch the gates and avoid intrusion.

Bibliography

- [1] N. Awad and K. Fitzgerald. The deceptive behaviors that offend us most about spyware. *Communications of the ACM*, 48(8):55–60, 2005.
- [2] M. Boldt and J. Wieslander. Investigating Spyware in Peer-to-Peer Tools. *Blekinge Institute of Technology*, 2003.
- [3] D. Bozağaç. Application of Data Mining based Malicious Code. Detection Techniques for Detecting new Spyware. Available online as www.cs.bilkent.edu.tr/~guvenir/courses/cs550/Workshop/Cumhur_Doruk_Bozagac.pdf, 2005.
- [4] Bulletproofsoft Spyware Adware Remover. Online at: <http://www.bulletproofsoft.com>.
- [5] J. Butler and G. Hoglund. VICECatch the Hookers. *Black Hat USA*, 2004.
- [6] J. Butler and P. Silberman. RAIDE : Rootkit Analysis Identification Elimination. Available online as www.blackhat.com/presentations/bh-europe-06/bh-eu-06-Silberman-Butler.pdf, 2006.
- [7] J. Butler, J. Undercoffer, and J. Pinkston. Hidden processes: the implication for intrusion detection. *Information Assurance Workshop, 2003. IEEE Systems, Man and Cybernetics Society*, pages 116–121, 2003.
- [8] B. Caelli. What Are Rings. Available online as <http://www.osronline.com/article.cfm?article=224>, 2003.
- [9] E. Chien and S. Response. Techniques of Adware and Spyware. *the Proceedings of the Fifteenth Virus Bulletin Conference, Dublin Ireland*, 2005.

- [10] J. Cogswell and M. Russinovich. Rootkit Revealer. Available online as www.sysinternals.com/ntw2k/freeware/rootkitreveal.shtml.
- [11] C. Cordes. Monsters in the Closet: Spyware Awareness and Prevention. *EDUCAUSE Quarterly*, 28(2), 2005.
- [12] Z. Csizmadia. Injecting a DLL into Another Process's Address Space. Available online as www.codeguru.com/dll/LoadDll.shtml, 2000.
- [13] B. Edelman. Methods and Effects of Spyware. *Response to FTC Call for Comments on Spyware*, Mar, 2004.
- [14] G. Erdélyi. Hide and Seek, Anatomy of Stealth Malware. *Proceedings of the 2004 Black Hat Europe*, pages 147–167, 2004.
- [15] Etrust Pestpatrol Anti-Spyware. Online at: <http://www.pestpatrol.com>.
- [16] F-Secure BlackLight. Online at: www.f-secure.com/blacklight.
- [17] E. Florio. When Malware Meets Rootkits.
- [18] FU Rootkit Project. Online at: <http://www.rootkit.com/project.php?id=12>.
- [19] C. Gutzman, S. Sweep, and A. Tambo. Differences and Similarities of Spyware and Adware. *University of Minnesota Morris*, 2003.
- [20] Hacker Defender Project. Online at: <http://hxdef.org>.
- [21] G. Hoglund. A Real NT Rootkit, Patching the NT Kernel. *Phrack Magazine*, 9.
- [22] G. Hoglund and J. Butler. *Rootkits: Subverting the Windows Kernel*. Addison-Wesley, 2006.
- [23] Q. Hu and T. Dinev. Is spyware an Internet nuisance or public menace? *Communications of the ACM*, 48(8):61–66, 2005.
- [24] G. Hunt and D. Brubacher. Detours: Binary interception of Win32 functions. *Proceedings of the 3rd USENIX Windows NT Symposium*, pages 135–143, 1999.

- [25] K. Kasslin, M. Ståhlberg, S. Larvala, and A. Tikkanen. HIDE AND SEEK REVISITED - FULL STEALTH IS BACK. *Virus Bulletin Conference 2005*, 2005.
- [26] Lavasoft Anti-Spyware. Online at: <http://www.lavasoft.com>.
- [27] McAfee Anti-Spyware. Online at: <http://www.mcafee.com>.
- [28] Microsoft Windows Preinstallation Environment(Windows PE). Online at: <http://www.microsoft.com/licensing/programs/sa/support/winpe.msp>.
- [29] Morphine Project. Online at: <http://hxdef.org>.
- [30] MSDN. Description of the Windows File Protection Feature. Available online as <http://support.microsoft.com/kb/222193/>, 2003.
- [31] Netlux. Online at: <http://vx.netlux.org>.
- [32] OptOut. Online at: <http://www.grc.com/optout.htm>.
- [33] Panda Anti-Spyware. Online at: <http://www.pandasoftware.com>.
- [34] Pedestal Software. Online www.pedestalsoftware.com.
- [35] M. Pietrek. Peering Inside the PE: A Tour of the Win32 Portable Executable File Format. *Microsoft Systems Journal*, 9(3):15-34, 1994.
- [36] B. Rector. *Developing Windows 3 Applications with Microsoft SDK*. SAMS Carmel, Ind, 1992.
- [37] F. Report. Monitoring Software on Your PC: Spyware, Adware, and Other Software. 2005.
- [38] S. Response. Windows Rootkit Overview. Technical report, Symantec, 2005. Security Response Whitepaper.
- [39] J. Richter. Load Your 32-bit DLL into Another Processes Address Space Using INJLIB. *Microsoft Systems Journal*, 1994.

- [40] M. Russinovich and B. Cogswell. Windows NT System-Call Hooking. *Dr. Dobbs Journal*, 22(1):42–46, 1997.
- [41] M. Russinovich and D. Solomon. *Microsoft Windows internals: Microsoft Windows server 2003, Windows XP, and Windows 2000*. Microsoft Press, 2005.
- [42] J. Rutkowska. modGREPER. Available online as <http://www.invisiblethings.org/tools/modGREPER/modGREPER-0.3-bin.zip>.
- [43] J. Rutkowska. System Virginty Verifier. Available online as <http://www.invisiblethings.org/tools/svv/svv-2.3-bin.zip>.
- [44] J. Rutkowska. Detecting Windows Server Compromises with Patchfinder 2. Available online as www.invisiblethings.org/papers/rootkits_detection_with_patchfinder2.pdf, 2004.
- [45] J. Rutkowski. Advanced Windows 2000 Rootkit Detection (Execution Path Analysis). *present at Black Hat USA*, pages 28–31, 2003.
- [46] S. Saroiu, S. Gribble, and H. Levy. Measurement and Analysis of Spyware in a University Environment. *Proceedings of the ACM/USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2004.
- [47] M. Schultz, E. Eskin, E. Zadok, and S. Stolfo. Data Mining Methods for Detection of New Malicious Executables. *IEEE Symposium on Security and Privacy*, 1:207–1, 2001.
- [48] P. Silberman. FUTo. Available online as <http://www.uninformed.org>.
- [49] E. Skoudis. *Malware: Fighting Malicious Code*. Prentice Hall PTR, 2003.
- [50] D. Solomon and M. Russinovich. *Inside Microsoft Windows 2000*. Microsoft Press Redmond, WA, USA, 2000.
- [51] S. Sparks and J. Butler. Spyware and Rootkits, The Future Convergence. *Login*, 29(6):8–15, 2004.
- [52] Spyware Doctor. Online at: <http://www.pctools.com/spyware-doctor/>.

- [53] T. Stafford and A. Urbaczewski. SPYWARE: THE GHOST IN THE MACHINE. *Communications of the Association for Information Systems (Volume14, 2004)*, 291(306):291.
- [54] Symantec Norton Anti-Virus. Online at: <http://www.symantec.com>.
- [55] A.-S. Team. 84% of Malware on Computers Worldwide is Spyware. Technical report, Panda Anti-Virus, 2005. Available online as <http://www.pandasoftware.com/about/press/viewnews.aspx?noticia=5968>.
- [56] The tripwire software package. Online at: <http://www.tripwire.com>.
- [57] VMware - Virtualization Software. Online at: www.vmware.com/.
- [58] Y. Wang, D. Beck, B. Vo, R. Roussev, and C. Verbowski. Detecting Stealth Software with Strider GhostBuster. *Proc. DSN*, 2005.
- [59] Y. Wang, R. Roussev, C. Verbowski, A. Johnson, M. Wu, Y. Huang, and S. Kuo. Gatekeeper: Monitoring Auto-Start Extensibility Points (ASEPs) for Spyware Management. *Usenix LISA: 18th Large Installation System Administration Conference*, 2004.
- [60] Y. Wang, B. Vo, R. Roussev, C. Verbowski, and A. Johnson. Strider GhostBuster: Why Its A Bad Idea For Stealth Software To Hide Files. Technical report, Microsoft Research Technical Report MSR-TR-2004, 2004.
- [61] M. Warkentin, X. Luo, and G. Templeton. A framework for spyware assessment. *Communications of the ACM*, 48(8):79–84, 2005.
- [62] J. Wieslander, M. Boldt, and B. Carlsson. Investigating Spyware on the Internet. *Proceedings of the Seventh Nordic Workshop on Secure IT Systems*.