Windows Rootkits: Attacks and Countermeasures

Desmond Lobo
Internet Commerce Security Laboratory
Graduate School of Information Technology and Mathematical Sciences
University of Ballarat, Australia
desmondlobo@students.ballarat.edu.au

Paul Watters
Internet Commerce Security Laboratory
Graduate School of Information Technology and Mathematical Sciences
University of Ballarat, Australia
p.watters@ballarat.edu.au

Xin-Wen Wu
School of Information and Communication Technology
Griffith University, Australia
x.wu@griffith.edu.au

Li Sun
School of Mathematical and Geospatial Sciences
RMIT University, Australia
lisun01@gmail.com

Abstract - **Windows XP is the dominant operating system in the world today and rootkits have been a major concern for XP users. This paper provides an in-depth analysis of the rootkits that target that operating system, while focusing on those that use various hooking techniques to hide malware on a machine. We identify some of the weaknesses in the Windows XP architecture that rootkits exploit and then evaluate some of the anti-rootkit security features that Microsoft has unveiled in Vista and 7. To reduce the number of rootkit infections in the future, we suggest that Microsoft should take full advantage of Intel's four distinct privilege levels.**

Keywords – computer security; malicious software (malware); rootkits; Microsoft Windows; Intel's ring architecture

I. INTRODUCTION

Rootkits refer to software that is used to hide the presence and activity of viruses, worms, Trojans and other forms of malware, and permit an attacker to take control of a computer system [21]. Installing a rootkit is usually the first thing that an attacker will do after gaining access to a system, as this will ensure that the attack will remain undetected [1].

Rootkits also often open a backdoor through which the attacker can spy on the system's activities [2]. The attacker can then proceed to capture personal data, such as bank account details, passwords, and credit card numbers.

In this paper, we concentrate on rootkits that target the Windows XP operating system. We focus on this particular operating system for obvious reasons: the Windows family accounts for approximately 90% of the operating systems in use today and, within the Windows family, XP is by far the most popular [3].

There are clearly two very good reasons why it is important to conduct research in the area of Windows rootkits:

1. It is estimated that 85% of malicious software is being written today with the intention of generating profit for the malware's author [4]. We are no longer dealing with script kiddies just trying to create malware for fun, but instead are targeted by organized criminal gangs that want to steal money. The Symantec Corporation even claims that "cyber crime has surpassed illegal drug trafficking as a criminal moneymaker" [5].
2. There has been an increase of several hundred percent in both the number and complexity of rootkits over the last few years [6]. Malicious software is already a very big worldwide problem and the proliferation of rootkits is only going to serve to escalate this problem.

These two trends are illustrated in Figure 1.

This paper discusses some of the rootkits that use hooking techniques to hide malware on a computer system. We explain how these rootkits were able to exploit the weaknesses of the Windows XP architecture.

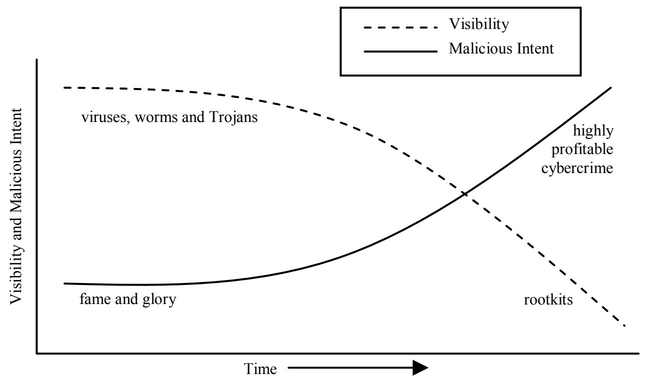

Figure 1. Visibility of Malware versus Malicious Intent [13]

XP was released in 2001; Microsoft has recently released two other versions of Windows, namely Vista and 7. This paper also outlines the steps that Microsoft has taken in order to address the weaknesses of the XP architecture. In particular, we identify some of the anti-rootkit security features that Microsoft has unveiled in Vista and 7.

We conclude the paper by suggesting that Microsoft should make better use of the available hardware memory protection mechanisms that are provided by Intel's four protection modes (also known as rings) in order to reduce the incidence rate of rootkit infections in the future.

The rest of this paper is organized as follows:

- We describe how rootkits evolved in section II.
- In sections III and IV, we explain some of the Windows XP rootkit hooking techniques and Intel's ring architecture, respectively.
- In section V, we assess some of the countermeasures that Microsoft has deployed in Vista and 7 to guard against rootkit attacks.
- Finally, we provide a discussion of and conclusion to the paper in section VI.

II. Evolution of Rootkits

The Pakistani Brain, the first computer virus for the PC that appeared in 1986, is also considered to be the first type of malware that used stealth techniques to avoid detection. During the following year, in 1987, another virus called Lehigh was released into the wild. Lehigh, however, was not nearly as successful as the Pakistani Brain virus and was very quickly contained, primarily because it did not utilize stealth techniques to remain hidden. Having noticed the effectiveness of the Lehigh virus in comparison to the Pakistani Brain, virus writers thereafter realized the importance of making use of stealth techniques. [6]

The first true rootkits started to appear in the early 1990s. During that period, malicious hackers often managed to penetrate computer systems and they would then use those compromised systems to launch attacks against other computers. To ensure that they would be able to take advantage of those compromised systems for an indefinite period of time, what they needed was a way to conceal their presence on the system. Thus, these hackers started developing rootkits for this purpose. [7]

1997 marked a milestone in terms of rootkit technologies as it saw the release of Cabanas, the first virus developed for the Windows NT system. Cabanas is considered to be the forefather of many of the rootkits in the wild today that hook two API (Application Programming Interface) functions, namely FindFirstFile and FindNextFile, in order to hide some files in a folder [8]. On a Windows machine, files in a folder are stored in a linked list. In order to display the files, the FindFirstFile function is called to find the first file and the FindNextFile function is called to locate subsequent files. If these two functions have been hooked, it is then possible to ensure that certain files in the folder are never displayed and remain hidden. [15]

A. *Rootkits Grab the Headlines*

It was in 2005 that rootkits really grabbed the headlines with an incident involving Sony BMG Music Entertainment, the world's second largest record label. Sony was concerned about users making illegal copies of their music files. Thus, they employed some stealth (rootkit) technologies on their music CDs that would hide digital rights management files and processes on a user's computer. This would prevent users from making illegal copies of the music files on their computers. [17]

This was achieved by using some code that would hide any file, folder or process that started with the string "sys" [19]. However, this meant that similarly named malware could, unfortunately, also be hidden from anti-virus scanners [17]. It was Mark Russinovich, a security expert, who had identified the rootkit and he subsequently made this knowledge public.

When the two million customers who purchased these CDs found out that their machines had been compromised, they were outraged. After listening to lawfully purchased music on their computers, these individuals learned that some software had been installed on their machine without their permission. On top of that, Sony had failed to provide these customers with an uninstaller to completely remove this software. [18, 19]

B. *Attacks on Financial Institutions*

In recent years, rootkits have been successfully used in attacks on financial institutions. In January 2007, for example, a large Swedish bank was attacked using Haxdoor, a Trojan with rootkit capabilities. Phishing emails were first sent to the bank customers urging them to download and run an anti-spam application [9, 10, 11]. The application had the Haxdoor malware embedded in it. When installed on the victim's system, the malware then proceeded to install a keylogger to capture keystrokes [12, 10]. This keylogger lay dormant on the user's computer until the victim visited the bank's website. This action triggered the keylogger to begin capturing keystrokes, and the stolen data was eventually sent to servers in Russia for further processing [12, 9, 10].

The Swedish bank reported that 250 customers had been attacked and a total of $1.17 million had been lost to the Russian organized criminals [12, 10, 11].

This incident was successful for two reasons:

- Firstly, the criminals managed to stay below the radar of security software by targeting a relatively small number of customers, by using a series of small withdrawals, and by spreading the attack over a period of 15 months [12, 11]. By staying below the radar in this way, the criminals made sure that no alarms were triggered.

- Secondly, the Haxdoor Trojan that was used in the attack contained a rootkit that ensured that the Trojan was not detected by the system.

C. *Proof-of-Concept Attacks on Smart Phones*

The latest news about rootkits involves a group of researchers at Rutgers University. In their research [20], they have been able to show that smart phones are just as vulnerable to rootkit attacks as desktop computers. The operating systems on these phones have evolved to the point that they are now almost as complex as those of PCs. Detecting and removing rootkits from desktop system has already proven to be a very challenging task, and the users of these smart phones are undoubtedly going to face similar challenges.

The main concern about smart phones is that these devices can access a number of interfaces that are not available on regular desktop computers, such as GSM, GPS and the battery:

1. The researchers at Rutgers demonstrated how rootkits could be used to listen in on private telephone conversations on a GSM network, and this could potentially lead to the leaking of sensitive information.
2. Since most of these phones are equipped with GPS tracking capabilities, rootkits could also be use in an attack that aims to compromise the victim's current location.
3. Lastly, rootkits could be used in an attack to drain the phone's battery and this would result in a denial of service for the user.

III. HOOKING WINDOWS XP

Having described how rootkits evolved in the previous section, we now explain how rootkits manage to hook the operating system. Hooking a Windows machine has several malicious purposes [24]:

- By hooking the functions that list the files in a directory, a rootkit can ensure that certain files remain hidden on a system.
- By hooking the appropriate functions in an antivirus program, a rootkit can guarantee that certain files are not scanned.
- By hooking the functions involved in keyboard input, a rootkit would have the ability to capture keystrokes.
- Hooking certain functions can cause a process to open a port on a system, thereby providing a backdoor for an attacker.

Hiding files is the most common action taken by rootkits and we now give an example to illustrate how a rootkit created a hook in order to accomplish this on an XP system. Figure 2 describes one of the many hooks that had been created by the Feebs malware. Feebs is actually a worm with rootkit capabilities. It attempts to harvest information from an infected computer and send this stolen data to a remote user [25]. A machine can become infected when a user executes an email attachment containing this malware [23].

The hook in figure 2 is called an inline function hook [7]. These types of hooks are created when a rootkit overwrites the first five bytes of an API function with a JUMP instruction. The first byte in the function is replaced with the value E9, the opcode for a JUMP in assembly language, and the remaining four bytes contain a 32-bit address of some malicious code. Therefore, the process will jump to this address to execute malicious code whenever this API function is called.

From figure 2, it is evident that the FindNextFileA API function in the Kernel32 DLL (Dynamic-Link Library) file had been hooked. As mentioned earlier, the Cabanas virus is considered to be the forefather of many of the rootkits in the wild today (such as Feebs) that hook this function in order to hide files.

The FindNextFileA API function had been exported by the Kernel32 DLL file to the Ctfmon Windows process (Process IDentifier 1776 in figure 2). This process runs in the background on Windows XP systems:

> "Ctfmon.exe monitors the active windows and provides text input service support for speech recognition, handwriting recognition, keyboard, translation, and other alternative user input technologies [22]."

Whenever this particular API function was called, some malicious code at address 10010822 was run. This code was contained within the mscf32 DLL file; Anson and Bunting [24] refer to such a file as a "rogue DLL" that an attacker had managed to inject into the memory address space of the Ctfmon process. Since this malicious code would be executed whenever the FindNextFileA function was called, the rootkit can ensure that certain files are never displayed.

A. *Windows XP Architectural Weaknesses*

It is possible to create a hook like that in figure 2 because there are some weaknesses in the Windows XP architecture that rootkits are able to take advantage of. In this subsection, we identify some of these weaknesses.

Rootkits are capable of redirecting the flow of execution of an XP system. In the example given in figure 2, the rootkit had managed to overwrite the first five bytes of the FindNextFileA API function and then redirect the flow of execution to the mscf32 DLL file. How was this achieved?

Most of the system DLL files are stored in the \Windows\System32\ directory. These critical files are protected by the Windows File Protection (WFP) feature [26]. This means that it is not possible to overwrite or replace these files, except in certain situations such as during a Windows update. This feature was incorporated in Windows 2000 and XP to help improve the stability of these systems.

```
McAfee(R) Rootkit Detective 1.1 scan report
On 21-02-2010 at 01:52:57
OS-Version 5.1.2600
Service Pack 3.0

==========================================

Object-Type: IAT/EAT-hook
PID: 1776
Details: Export : Function  :
  kernel32.dll!FindNextFileA =>
  C:\WINDOWS\system32\mscf32.dll:10010822
Object-Path: C:\WINDOWS\system32\mscf32.dll
Status: Hooked
```

Figure 2. Inline Function Hook Created by a Feebs Worm/Rootkit

The necessary DLL files from the \Windows\System32\ folder are then copied into memory. These copies are marked as 'read-only' and all the processes have access to these shared files. If a particular process needs to modify one of these DLL files, a second copy of the file is created, while the other processes can still access the original DLL copy. This procedure is called Copy-on-Write: it protects processes from damaging each other while consuming as few resources as possible [27].

Rootkits exploit the fact that these DLL files can be overwritten in memory and are thus able to change the flow of execution. Even having modified the flow of execution, rootkits still need to perform one additional step: in the figure 2 example, the rootkit had managed to inject the mscf32 DLL file, containing some malicious code, into the Ctfmon process. The injection of this DLL file into the address space of the Ctfmon process could have been accomplished using, for instance, the LoadLibrary API function.

In addition to this function, there are others (such as CreateRemoteThread and WriteProcessMemory) that make it trivial to inject malicious code into a running process. These functions transcend the normal barriers that have been put in place to protect processes from modifying each other. Sparks et al. [34] facetiously refer these functions as the "Rootkit API" that is provided by Windows since they simplify the job of a rootkit writer.

B. *Legitimate Uses of Hooking*

The next question that arises is: why does Microsoft allow this to occur? There are actually legitimate reasons for deploying hooks in a Windows machine.

Way back in 1999, Microsoft provided a situation in which hooking the operating system would be a suitable option [33]. A company had created a DLL file that it would inject into a database product in order to enhance the capabilities of that product. When an attempt was made to terminate the product, a DLL_Process_Detach notification was sent, with the objective of unloading the DLL file from the address space of a certain process. The DLL would then proceed to close socket connections, files, and other resources. However, by the time the DLL file received the notification, other DLLs in the process' address space would have already received their DLL_Process_Detach notifications. Thus, many functions that the DLL called would fail because the other DLLs had already been unloaded.

Microsoft's suggestion was to hook the ExitProcess API function. It is the ExitProcess function that ends the process and causes the system to send the DLL_Process_Detach notifications. If the ExitProcess function was hooked, it could be arranged so that the company's DLL file would be the first one to be unloaded. The hook could then redirect the flow of execution back to the original ExitProcess function and the remaining DLLs could be unloaded without any concerns.

Today, a lot of legitimate software, including several security packages such as antivirus and firewall applications, deploys hooks in the operating system to receive notifications about events like file creation and opening of ports [47]. Unfortunately, as we stated earlier, there are scores of rootkits that also exploit these techniques.

IV. INTEL'S RING ARCHITECTURE

In the previous section, we identified some of XP's architectural flaws that rootkits could take advantage of. Another weakness of the Windows design relates to the hardware that this software runs on. Intel processors were created with four protection rings, as illustrated in figure 3. These rings could be used to help separate user applications, operating system services, device drivers and the operating system's kernel. The inner rings have more privileges than the outer rings; in other words, the inner rings have full access to the outer rings and follow the principle of least privilege.

On the other hand, there are special gates between the rings that control the access that an outer ring has to an inner ring. This helps improve security since an outer ring would not be able to gain access to an inner ring at will. For example, a user mode rootkit running in ring 3 would be prevented from turning on a web camera since the drivers for this hardware would only be accessible from ring 1 [37]. Furthermore, this design ensures that misbehaving applications, services or drivers will not disrupt the stability of the operating system's kernel [28].

Even though Intel processors offered four distinct rings, the OS/2 operating system, which was initially created by Microsoft and IBM, made use of just three of these: ring 3 for user applications and OS/2 services, ring 2 for device drivers, and ring 0 for the OS/2 kernel [38].

For the Windows family of operating systems, Microsoft reduced that number further and only made use of rings 3 and 0: ring 3 was used for user applications while ring 0 was used for Windows services, device drivers and the Windows kernel. There were some concerns with this design; for example, there would be a need for driver signing (which we will elaborate on in the following section).

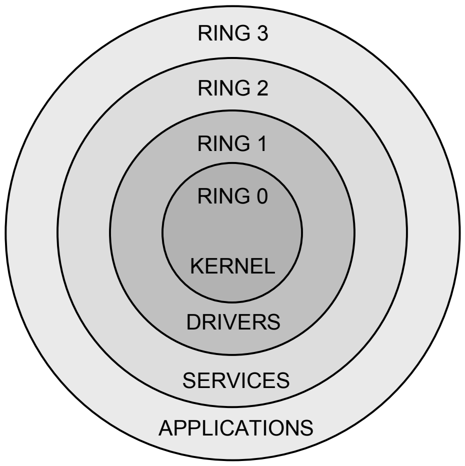

Figure 3. Intel's Ring Architecture [28, 37, 24, 14]

Microsoft did provide a reason for this decision to only use two rings:

> "Some hardware that was supported in the past (such as Compaq Alpha and Silicon Graphics MIPS) implemented only two privilege levels [28]."

It is clear that Windows was first designed for a single-user PC without a network connection, and security features were not built in from the outset; Microsoft was more concerned about compatibility. Their strategy was to get their product to market as fast as possible and, given the success that they have had with Windows, you have to applaud them for that. The trouble was that rootkit writers began to take advantage of this architecture. With only two rings in use, there was no obstacle or barrier between the kernel space and user land. It was quite possible for rootkits to gain access to ring 0 and, once they did, they would then be able to take full control of the computer system.

V. DEFENSE-IN-DEPTH

The Haxdoor Trojan that was used in the attack on the Swedish bank is typical of the malware that we can expect to witness more often in the future. In addition to its keylogging abilities, it also has screen capturing and form grabbing capabilities. These features will ensure that the malware will somehow manage to gather the personal data that it is after. Furthermore, its rootkit features are also proving that this strategy of quietly stealing information without raising any alarms is very effective.

It is, therefore, logical to take the necessary steps to prevent infection of the system in the first place, and the best strategy to tackle such an attack-in-depth is to develop an effective defense-in-depth approach. This was a term that was originally associated with a military tactic that tried to buy some time for soldiers when they came under attack. The defense-in-depth method, in terms of information security, uses multiple layers of security to ensure that confidential data is protected, even if the attackers manage to circumvent some of the layers.

After enduring years of attacks on its operating systems, Microsoft has in recent years started to focus more on security and has invested considerable resources to protect its operating systems [16]. They have adopted a defense-in-depth strategy to prevent rootkits from infecting their operating systems. Their multi-layered approach includes items such as:

A. Kernel Patch Protection (KPP)

B. Data Execution Prevention (DEP) and Address Space Layout Randomization (ASLR)

C. Driver Signing

D. Windows Service Hardening

While it might be possible to circumvent each individual defensive method, the cumulative effect of several layers of defenses will make the job of the attacker much more difficult. These defensive features are described in the following sections.

A. Kernel Patch Protection (KPP)

The kernel is the central component of the operating systems and can be considered to be a bridge between application programs and the hardware. This is illustrated in figure 4. Having such a key role to play in an operating system, Microsoft introduced Kernel Patch Protection (KPP), also known as Patchguard, to help protect the kernel and to improve the overall reliability, performance and security of Windows [31].

In particular, KPP can help prevent modifications of the System Service Descriptor Table (SSDT), which is often hooked by rootkits. For instance, figure 5 describes one of several SSDT hooks that had been created by Haxdoor, the Trojan/rootkit that was used in the attack on the Swedish bank.

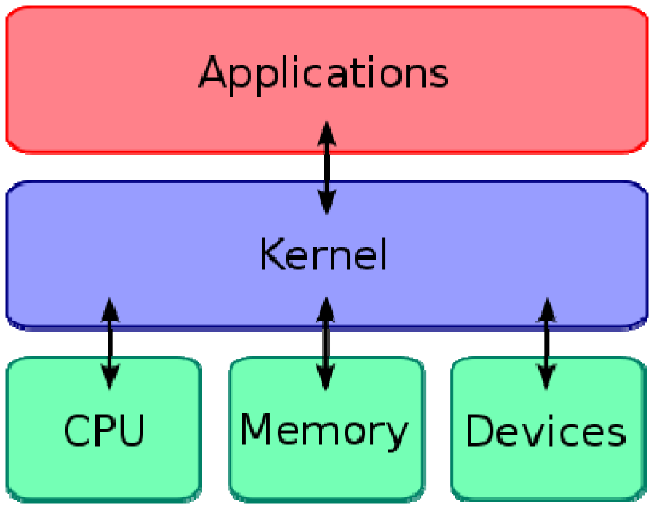

Figure 4. Kernel of the Operating System [30]

Native system services refer to undocumented API functions for the Windows operating system that are callable from user mode [28]. In figure 5, for example, ZwCreateProcess is an internal system service that the CreateProcess API function calls to create a new process. A native system service call is thus a mechanism that allows a user mode application to access the operating system's kernel [8].

The SSDT contains a list of pointers with the addresses of the internal kernel function that implements the corresponding service [28, 8, 7]. A rootkit can intercept calls that are made to a specific native system service by replacing the SSDT entry with the address of its own code. After this rootkit code is executed, the original native system service can be called or some fabricated data can be returned instead [7, 29]. A graphical representation of the status of an SSDT hook is given in figure 6.

Microsoft introduced KPP in 2005 and this consequently prevented rootkits from modifying any part of the kernel, such as the SSDT, the interrupt descriptor table (IDT), the global descriptor table (GDT), etc. [31]. Every five to ten minutes, KPP checks to confirm that these critical components of the kernel have not been modified. It does this by comparing against known good copies or signatures. If KPP determines that one of these components has been altered, it forces the system to crash. [39]

KPP was only available for 64-bit versions of Windows, though, and there is a perfectly good reason for this. In 2005, 32-bit versions of Windows were ubiquitous and, hence, there was an abundance of application programs available for these versions of Windows. Since Microsoft had not implemented any sort of KPP prior to 2005, many of the developers of these application programs took the liberty of hooking the kernel. This included products from several prominent security vendors, such as McAfee, Symantec and Kaspersky [32]. If Microsoft had employed KPP on 32-bit versions of Windows, these products that were hooking the kernel would have ceased functioning. This is not something that Microsoft would have wanted: they strongly encourage compatibility between their operating systems and third-party application software.

```
McAfee(R) Rootkit Detective 1.1 scan report
On 16-11-2009 at 03:55:05
OS-Version 5.1.2600
Service Pack 3.0

==========================================

Object-Type: SSDT-hook
Object-Name: ZwCreateProcess
Object-Path: C:\WINDOWS\system32\vdnt32.sys
```

Figure 5. SSDT Hook Created by a Haxdoor Trojan/Rootkit

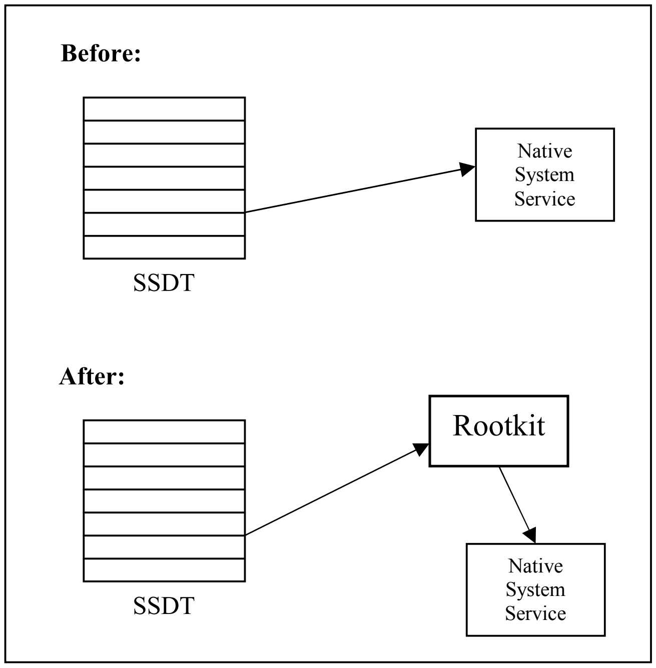

Figure 6. Before and After an SSDT Hook is Implemented [29]

On the other hand, a relatively small percentage of 64-bit versions of Windows were running in 2005. Thus, Microsoft took the step to implement KPP just on 64-bit versions as this would raise an insignificant number of compatibility issues. As a final point, to appease vendors such as McAfee, Symantec and Kaspersky, Microsoft created additional APIs to ensure that these companies would still be able to develop 64-bit versions of their products without having to hook the kernel.

We conclude that this is an indication that Microsoft has definitely started taking security very seriously with the development of KPP. When it comes to protesting the kernel, Microsoft has, in essence, given up on 32-bit versions of its operating systems. Nevertheless, the software giant has taken a strong stance to ensure that the 64-bit kernel will be protected. Not only will a 64-bit machine be able to accommodate significantly more RAM, users of these systems can rest assured that they will be provided with increased security as well.

B. Data Execution Prevention (DEP) and Address Space Layout Randomization (ASLR)

Prior to deploying a rootkit in a computer system, the attacker must gain access to that system. Often, attackers manage to exploit the software on the system by using buffer overflows [7]. In response to these attacks, Microsoft has recently introduced two defensive techniques: Data Execution Prevention (DEP) and Address Space Layout Randomization (ASLR). We first briefly explain how buffer overflow attacks work and then describe the two countermeasures.

1) Buffer Overflows

Basically, attackers who make use of buffer overflows take advantage of the fact that most high-level language programmers do not fully understand what is occurring at the assembly level. Suppose, for example, a program has been designed to request some type of input and this information is stored on the stack in a buffer of maximum size 100. To cause the buffer to overflow, the attacker could possibly send 1000 characters, with the remaining 900 characters overwriting the adjacent memory on the stack. [35]

A well-designed buffer overflow attack would ensure that the 900 excess characters, in this case, contained some malicious rootkit code that would be stored on memory. Once this code has been executed, the attacker would be able to take control of the system. [36]

It should be pointed out that Linux- and Unix-based operating systems are susceptible to these types of attacks as well. The telnet daemon (telnetd), for instance, which allows users to remotely log in to a machine, was found to contain a vulnerability that would allow an attacker to trigger a buffer overflow and cause a denial of service (DoS) or possibly execute malicious code [50].

2) DEP and ASLR Countermeasures

Windows machines that have enabled DEP, which is also referred to as eXecute Disable (XD) by Intel and No eXecute (NX) by AMD, are less vulnerable to buffer overflow attacks. DEP first identifies which memory locations in a process contain data and which contain code. After this has been established, DEP can then block the execution of any code in the data content area [16]. Thus, an attacker might, for instance, endeavor to use a buffer overflow exploit to inject some malicious code and overwrite data on the stack. If an attempt is then made to execute that code, DEP immediately triggers an alarm and the program is terminated. [40]

ASLR also helps prevent buffer overflow attacks. With ASLR, modules are loaded into random locations whenever a system boots and this makes it difficult for shellcode to operate successfully [42, 39]. Suppose, for example, an attacker had managed to introduce some malicious code onto the stack. Next, suppose that this code then attempts to inject a rogue DLL into a process by calling the LoadLibrary API function in the Kernel32 DLL file. The location of the LoadLibrary function would need to be determined. This task would be more complicated with ASLR because the Kernel32 DLL file might have been loaded into any one of 256 different locations.

A security expert from Microsoft demonstrated how ASLR works [41]. He first determined the location of several DLL files on his laptop:

• wsock32.dll	0x73ad0000
• winhttp.dll	0x74020000
• user32.dll	0x779b0000
• kernel32.dll	0x77c10000
• gdi32.dll	0x77a50000

After rebooting his machine, he then found that, because of ASLR, those same DLLs had moved to the following locations:

• wsock32.dll	0x73200000
• winhttp.dll	0x73760000
• user32.dll	0x770f0000
• kernel32.dll	0x77350000
• gdi32.dll	0x77190000

DEP and ASLR are most effective when they are used together [39, 42]. If DEP is used without ASLR, the code that has been injected onto the stack could be used to redirect the flow of execution to a known function address. Conversely, if ASLR is used without DEP, the attacker could simply execute code off the stack.

C. Driver Signing

As noted earlier, the Windows family of operating systems only uses two of the four protection rings offered by the processor. Intel had originally anticipated that device drivers would operate in ring 1 but, because ring 1 is not used in Windows machines, these drivers execute instead in ring 0. This means that they have full access to the computer system, and this raises some concerns. The driver might possibly contain some malicious code, such as a rootkit, that could be used to take control of the machine.

This vulnerability was one of the reasons that Microsoft introduced a driver signing mechanism, where the computer user would be warned whenever an attempt was made to install an unauthorized driver [28].

D. Windows Service Hardening

Windows services refer to programs that run quietly in the background on a Windows machine [28, 24]. As mentioned previously, these services, device drivers and the kernel all operate in ring 0, and this causes some concerns. If a rootkit manages to grab control of one of these services, it could execute with unrestricted privileges and take over the whole computer system [14]. Another reason that these services are attractive to rootkit writers is that they are normally running from the time the machine boots up until it shuts down [43].

Thus, Microsoft introduced Windows service hardening to restrict the privileges that were available to these services, thereby removing any privileges that each service did not require. Furthermore, there were procedures put into place to ensure that the services were isolated from each other, consequently protecting each of these services from the other services and applications. [43]

VI. Discussion and Conclusion

This paper has outlined some of the anti-rootkit features that Microsoft has introduced in the last few years: Kernel Patch Protection, Data Execution Prevention, Address Space Layout Randomization, Driver Signing, Windows Service Hardening, etc. Essentially, Microsoft has had to resort to these measures because of the decision that they took to make use of only two of the four protection rings that were available on the processor:

- If the kernel had been isolated in ring 0, there might not have been any need for implementing Kernel Patch Protection.
- If device drivers had been installed in ring 1 as Intel had planned, there might not have been any need for requiring Driver Signing.
- If Windows services had been executing in ring 2 as Intel had intended, there might not have been any need for employing Windows Service Hardening.

Was it then a good decision for Microsoft to use just two of the four protection rings? There is no question that because Microsoft made this decision, their operating systems are not as secure as they could have been. On the other hand, because Microsoft was concerned about compatibility from the very beginning, they now have 90% of the operating systems market share, and you can't really argue with those results.

So, where does Microsoft go from here? We feel that this is the time for Microsoft to completely redesign their Windows operating system and adopt a four-ring architecture as Intel had originally proposed. We conclude this paper by providing some justification for making this statement:

- Microsoft needs to take back control of the kernel. Having device drivers and Windows services also operating in ring 0 should not be permissible. Ring 0 needs to be reserved exclusively for the kernel.

  Because Microsoft does not have full control of the kernel, they have had to resort to deploying strategies such as Kernel Patch Protection. These short term solutions do not address the real problem. In fact, Authentium and Uniformed [44, 45] have already demonstrated that it is possible to circumvent Kernel Patch Protection.
- Microsoft is a well established company with a strong brand name and an abundance of resources available to them. They could invest some of these resources into developing a new version of Windows, based on a four-ring architecture, from scratch. The demand is there for a top quality product.
- Microsoft is facing some formidable competition in Google. Google is definitely more than just a search engine, offering products and services such as Gmail, Google Docs, YouTube and Google Maps. Most importantly, Google is about to unveil a brand new operating system that, unlike Windows, has been designed from the ground up with security in mind [46].

  Google's new operating system will also appeal to financial institutions, such as the Swedish bank and its 250 customers that were attacked. One of Google's guiding principles is, "Don't scapegoat the users" [46]. The burden of ensuring that the computer system is secure should not be the responsibility of the bank's customers and they should not be held accountable when attacked. In fact, David Shroyer, vice president of online security and enrollment at Bank of America, goes on to point out that "customer education is less powerful of a weapon against stealthy malware that is constantly finding ways to avoid detection" [48].

  Google has already entered the smartphone market and, if the statistics in table 1 are anything to go by, Microsoft needs to take note and should be concerned about the impending release of Google's desktop operating system. In the three-month period from the end of November 2009 until the end of February 2010, Google increased their market share of smartphone operating systems by 5.2 points, in large part at the expense of Microsoft.

  Security is certainly a very high priority for computer users today and Microsoft does not have a very good reputation when it comes to security. If these users are not satisfied with the security that is being provided by Windows and if Google offers a better (and cheaper) alternative, they will surely make the switch.

Table 1: Top Smartphone Platforms in the US [49]

	Share (%) of Smartphone Subscribers		
	Three Month Avg. Ending Nov. 2009	Three Month Avg. Ending Feb. 2010	Point Change
RIM	40.8%	42.1%	1.3
Apple	25.5%	25.4%	-0.1
Microsoft	19.1%	15.1%	-4.0
Google	3.8%	9.0%	5.2
Palm	7.2%	5.4%	-1.8

ACKNOWLEDGMENT

This research was funded by the University of Ballarat, the Australian Government and the Internet Commerce Security Laboratory (which includes Westpac Banking Corporation, IBM Australia, the Australian Federal Police and the State Government of Victoria). We would like to thank the anonymous reviewers for their helpful comments.

REFERENCES

[1] M. Alvarez, M. Vucelich, and L. Johnson, "IBM Internet Security Systems X-Force Threat Insight Monthly", July 2008, IBM Corporation

[2] N. A. Quynh, and Y. Takefuji, "Towards a Tamper-Resistant Kernel Rootkit Detector", Symposium on Applied Computing, Proceedings of the 2007 ACM Symposium on Applied Computing, pp. 276-283, Seoul, South Korea

[3] W3Schools, "OS Platform Statistics", Retrieved from http://www.w3schools.com on 10 March 2010

[4] L. Wang and P. Dasgupta, "Kernel and Application Integrity Assurance: Ensuring Freedom from Rootkits and Malware in a Computer System", Proceedings of the 21st International Conference on Advanced Information Networking and Applications Workshops, 2007, IEEE Computer Society

[5] Symantec, "Cyber Crime has Surpassed Illegal Drug Trafficking as a Criminal Moneymaker; 1 in 5 will become a Victim", Symantec Corporation Press Release, Retrieved from http://www.symantec.com on 5 December 2009

[6] McAfee, "Rootkits - Part 1 of 3: The Growing Threat", McAfee Inc., Apr. 2006

[7] G. Hoglund and J. Butler, "Rootkits: Subverting the Windows Kernel", Addison-Wesley Software Security Series, Pearson Education Inc., 2006

[8] K. Kasslin, M. Stahlberg, S. Larvala, and A. Tikkanen, "Hide 'N Seek Revisited - Full Stealth is Back", Proceedings of the 15th International Virus Bulletin Conference, 2005, Dublin, Ireland

[9] D. Ladd, "News Briefs", IEEE Security and Privacy, March/April 2007, IEEE Computer Society

[10] Y. Ben-Itzhak, "Defending Your Organization Against the New Generation of Web-Based Hybrid", Infosecurity, Volume 4, Number 3, 2007, pp. 42-43

[11] X. Zhang and K. C. Tadi, "Modeling Virus and Antivirus Spreading Over Hybrid Wireless Ad Hoc and Wired Networks", Proceeding of the IEEE Global Telecommunications Conference, 2007, USA

[12] P. Wollacott, "Cybercrime Comes of Age", ITNOW, Vol. 49, No. 2, 2007, pp. 6-7, The British Computer Society, Oxford University Press

[13] OECD, "Malicious Software (Malware): A Security Threat to the Internet Economy", Organization for Economic Co-operation and Development, June 2008, OECD Ministrial Meeting on the Future of the Internet Economy

[14] M. Davis, S. Bodmer and A. LeMasters, "Hacking Exposed Malware and Rootkits: Malware and Rootkits Secrets and Solutions", McGraw-Hill Osborne Media, 2010

[15] J. Butler and S. Sparks, "Windows Rootkits of 2005", Security Focus, Retrieved from http://www.securityfocus.com on 12 March 2010

[16] J. Allchin, "Security Features Versus Convenience", The Windows Blog, Microsoft Corporation, 23 January 2007, Retrieved from http://windowsteamblog.com on 12 March 2010

[17] McAfee, "Rootkits - Part 1 of 3: The Growing Threat", McAfee Inc., Apr. 2006

[18] D. K. Mulligan and A. K. Perzanowski, "The Magnificance of the Disaster: Reconstructing the Sony BMG Rootkit Incident", Berkley Technology Law Journal, Vol. 22, p. 1157, 2007

[19] J. A. Halderman and E. W. Felten, "Lessons from the Sony DRM Episode", Proceedings of the 15th USENIX Security Symposium, pp. 77-92, 2006

[20] J. Bickford, R. O'Hare, A. Baliga, V. Ganapathy and L. Iftode, "Rootkits on Smart Phones: Attacks, Implications and Opportunities", Proceedings of the Eleventh Workshop on Mobile Computing Systems & Applications, ACM, pp. 49-54, Annapolis, Maryland, USA, 2010

[21] A. Emigh, "The Crimeware Landscape: Malware, Phishing, Identity Theft and Beyond", Journal of Digital Forensic Practice, Vol. 1, No. 3, September 2006, pp. 245-260

[22] Microsoft Support, "Frequently asked questions about Ctfmon.exe", 29 January 2007, Retrieved from http://support.microsoft.com on 15 March 2010

[23] McAfee, "W32/Feebs!rootkit", Retrieved from http://vil.nai.com on 15 March 2010

[24] S. Anson and S. Bunting, "Mastering Windows Network Forensics and Investigation", Wiley Publishing, 2007

[25] Sophos, "W32/Feebs-Gen", Retrieved from http://www.sophos.com on 15 March 2010

[26] Microsoft Support, "Description of the Windows File Protection Feature", 11 September 2009, Retrieved from http://support.microsoft.com on 18 March 2010

[27] P. Dabak, S. Phadke and M. Borate, "Undocumented Windows NT", Hungry Minds, 1999

[28] M. E. Russinovich and D. A. Solomon, "Microsoft Windows Internals", 4th Edition, Microsoft Press, 2005

[29] C. Ries, "Inside Windows Rootkits", VigilantMinds, 2006

[30] Daymix, "Kernel Computing", Retrieved from http://daymix.com on 12 April 2010

[31] Windows Hardware Developer Central, "Kernel Patch Protection: Frequently Asked Questions", 22 January 2007, Retrieved from http://www.microsoft.com on 19 March 2010

[32] M. Oiaga, "Windows vs. Rootkits: The root(kit) of all evil", 20 February 2010, Retrieved from http://news.softpedia.com on 20 March 2010

[33] J. Richter, "Programming Applications for Microsoft Windows", Microsoft Press, 1999

[34] S. Sparks, S. Embleton and C. Zou, "Windows Rootkits: A Game of Hide and Seek", School of Electrical Engineering and Computer Science, University of Central Florida, USA [n.d.]

[35] McAfee, "Buffer Overflow Exploits: The Why and How", McAfee System Protection Solutions, April 2005

[36] H. M. Deitel, P. J. Deitel and D. R. Choffnes, "Operating Systems, Third Edition", Prentice Hall, 2004

[37] C. Mitchell, "Trusted Computing Platforms: Intel's Trusted eXectuion Technology (TXT)", Information Security Group, Royal Holloway University of London, Retrieved from http://www.isg.rhul.ac.uk on 1 April 2010

[38] WarpSpeed Computers, "Presentation Device Driver Reference for OS/2", Retrieved from http://www.warpspeed.com.au on 2 April 2010

[39] B. Blunder, "The Rootkit Arsenal: Escape and Evasion in the Dark Corners of the System", Wordware Publishing, 2009

[40] K. Kubicki, "CPU & Chipset: A bit about the NX bit; Virus Protection Woes", AnandTech Incorporated, 11 October 2004, Retrieved from http://www.anandtech.com on 8 April 2010

[41] M. Howard, "Address Space Layout Randomization in Windows Vista", Microsoft Corporation, 26 May 2006, Retrieved from http://blogs.msdn.com on 8 April 2010

[42] M. Howard and M. Thomlinson, "Windows Vista ISV Security", Microsoft Corporation, April 2007, Retrieved from http://msdn.microsoft.com on 26 March 2010

[43] W. Moses, "Security Watch: Services Hardening in Windows Vista", TechNet Magazine, Microsoft Corporation, January 2007

[44] Authentium, “Microsoft Patchguard and Authentium”, Authentium Virus Blog: Authentium Malware Information Exchange Portal, 25 October 2006, Retrieved from http://blogs.authentium.com on 11 April 2010

[45] Skywing, “PatchGuard Reloaded: A Brief Analysis of PatchGuard Version 3”, Uninformed Journal, Volume 8, September 2007

[46] Google, “The Chromium Projects: Security Overview”, Retrieved from http://www.chromium.org on 11 April 2010

[47] M. Jakobsson and Z. Ramzan, “Crimeware: Understanding New Attacks and Defenses”, Addison Wesley, Symantec Press, 2008

[48] M. Savage, “The Banking Malware Scourge”, Information Security, May 2010

[49] comScore “comScore Reports February 2010 U.S. Mobile Subscriber Market Share”, 5 April 2010, Retrieved from http://www.comscore.com on 18 May 2010

[50] Cisco, “Linux/Unix: Telnet Daemon Buffer Overflow Vulnerability”, Cisco Systems Inc., 6 October 2004, Retrieved from http://cisco.com on 12 June 2010