JAMES BUTLER AND SHERRI SPARKS

spyware and rootkits

THE FUTURE CONVERGENCE

Jamie Butler is the director of engineering at HBGary, Inc. and is co-author of the upcoming book Rootkits: The Day After. He is also a frequent speaker at computer security conferences, presenting his research on kernel reverse engineering, host-based intrusion prevention, and rootkits.

james.butler@hbgary.com

Sherri Sparks is a Ph.D. student of Computer Science at the University of Central Florida. Her current research interests are in software security, reverse engineering, and intrusion detection.

ssparks@longwood.cs.ucf.edu

ALL OF A SUDDEN YOUR PENTIUM 4, 3.2GHz desktop with 3GB of memory takes half an hour to boot into Windows. What's more, you can't seem to open Internet Explorer without being escorted to a home page you'd rather die than let your mother see. Of course, that is to say nothing of the unsolicited pop-up advertisements bombarding you at every click. And if all of that wasn't indication enough, you know there's a problem when your machine starts complaining about being "out of memory"...and the only program running is Notepad! Welcome, dear reader, to the modern worldwide scourge: spyware.

Understanding the Threat

WHAT IS SPYWARE?

The term "spyware" encompasses a large class of software capable of covertly monitoring a system and transmitting collected data to third parties. Such data may range from visited URLs to passwords and other confidential information. As of the 2003 publication of *Emerging Internet Threats Survey*,[1] one-third of companies have been affected by spyware-infected systems, and that number is growing. The implications are alarming on both commercial and private fronts. Of primary concern are the violations to personal privacy and the protection of intellectual property. Secondary issues relate to the degradation of system performance, network bandwidth, and utilization of IT personnel as they are forced to deal with application conflicts and system instability.

INFECTION VECTORS

A spyware infection is most often unintentional; however, there are cases where the software is purposely installed on someone's system to gather personal information or to monitor their Internet browsing habits. Commercial key loggers and parental control software both fall into this category. Unintentional infection may result from the exploitation of unpatched browser vulnerabilities and social engineering. It is not uncommon for unsuspecting users to be tricked into downloading and installing software they believe to be something else. Spyware may also piggy-back on the installa-

tion of another seemingly legitimate application. For instance, peer-to-peer networking software such as Kazaa and Bear Share, most of which contains embedded spyware, is proliferating. It goes without saying that the popularity of these programs ensures a continuing supply of infected hosts.

SPYWARE CHARACTERISTICS

We can divide spyware characteristics into primary and secondary traits, where primary traits relate to actively spying on the target system and secondary traits relate to concealing the spyware presence from the system's users. Primary traits include taking screenshots or logging keystrokes, running applications, and capturing URLs of visited Web pages. Sending logged information to a third party could also be considered a primary characteristic. In contrast, functionality for hiding registry entries, files, and running processes would be considered secondary traits. Behaviors that make the application difficult to remove also fall into this category. These types of behaviors include loading during the boot process, disabling detection programs, and reinstallation after removal.

Current-generation spyware is defined by a vast number of primary characteristics, but except for specific "hacker tools" (e.g., trojans), secondary characteristics have been either nonexistent or relatively primitive up to this point. In other words, most of this spyware is very efficient at spying but not very efficient at hiding. Consequently, it has, for the most part, been detectable with simple file- and registry-scanning techniques. Nonetheless, information security is a co-evolutionary process, and spyware development/detection is no exception. A few leading-edge spyware developers such as CoolWebSearch[2] are adapting and forcing us to an essential juncture. We note that the primary traits of spyware fundamentally describe a software trojan, while the secondary traits essentially define a rootkit. And we ask ourselves, What is the next generation in spyware? . . . Trojan meets Rootkit?

The Next Generation of Spyware: Trojan Meets Rootkit?

The functional requirements of a successful spyware application and a successful trojan or rootkit are remarkably similar. First, spyware programs, like rootkits and trojans, need to intercept user data such as keystrokes and network communications. Secondly, they must hide their presence from the user and/or make uninstallation difficult. Currently lacking sophistication in this second area, it is reasonable to expect that primitive spyware applications will continue to evolve their ability to conceal themselves. Rootkits have already mastered this ability. By understanding the application of advanced rootkit techniques to spyware, we may be better prepared to deal with the threat of an impending spyware epidemic.

Regardless of the goal, whether it be hiding presence or intercepting communication, spyware and rootkits must both wedge themselves between the legitimate calling program, such as Internet Explorer, and the end communication point, either another node on the network or the underlying operating system. This involves altering normal program control flow. In order to accomplish this, rootkits use two primary technologies: hooks and layers. Once these technologies are in place, the spyware or rootkit can capture keystrokes as someone logs into her/his online bank account, or they can hide the presence of a particular process or network port from appearing on the local machine. In the remainder of this section we give an overview of rootkit techniques that are applicable to current and future spyware

developments.

USER MODE TECHNIQUES (IMPORT HOOKING AND BROWSER HELPER OBJECTS)

Without a doubt, techniques such as import address table (IAT) hooking and browser helper objects (BHOs) are the most common methods of program subversion used by rootkits and trojans. Unfortunately, the Windows architecture makes these types of attacks accessible to even the lowest, most humble user mode applications. This is due to the fact that user programs and upper-level operating system components coexist at the same privilege level. Ultimately, spyware and rootkits have complete control over an application because they run with the same rights as the application they are hijacking. Some forms of spyware, such as key loggers and browser hijackers, already employ these techniques. Nevertheless, we expect to see an increase in their utilization as primitive spyware applications take on more rootkit-like characteristics to evade detection and removal.

In user mode, an attacker generally 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. For example, if a spyware program is able to intercept a user mode scanner application's effort to open its files, it can return errors indicating those files don't exist. Subsequently, the scanner will falsely report that the system is uninfected. Windows APIs are implemented as dynamically linked libraries (DLLs) and are the basis of IAT hooking attacks.

The design of DLLs facilitates the attack. When an application uses an API function exported from a DLL such as InternetConnect in Wininet.dll, the compiler creates an `IMAGE_IMPORT_DESCRIPTOR` data structure in the application's binary file. The `IMAGE_IMPORT_DESCRIPTOR` contains the name of the DLL from which the function is exported and a pointer to a table containing all of the functions exported by the DLL that are used by the application. Each member of this table is an `IMAGE_THUNK_DATA` structure which is filled in at load time, by the Windows loader, with the memory address of the desired function. We can summarize the flow of execution as follows. Suppose an application makes a call to InternetConnect. First, the program code calls into the Import Address Table (IAT). From there, the IAT contains a jump that is taken to the destination address of the real function. It is easy to see that the IAT is a likely target for a rootkit or spyware. By changing a single function pointer, the attacker can re-route program execution through his/her own code, thereby capturing data, altering data, or even hiding the attacker's presence (see Figure 1). A more comprehensive explanation of the Windows portable executable (PE) format and IAT structure can be found in Matt Pietrek's "Inside Windows."[3]

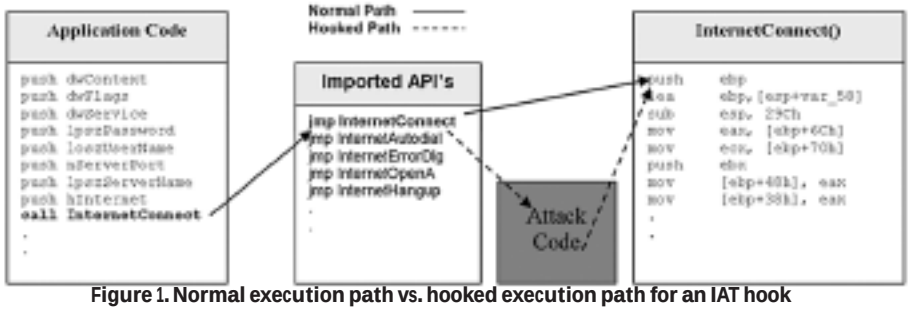

Figure 1. Normal execution path vs. hooked execution path for an IAT hook

Browser Helper Objects (BHOs) demonstrate another DLL-based userland attack technique. While BHOs are specifically designed to customize and extend Internet Explorer, many browsers provide similar features that can be maliciously exploited by spyware applications. A BHO is suspect if a user suddenly notices that his home page has been redirected, "new" toolbars have suddenly appeared in his browser, or his list of "favorites" has been modified. The risk of BHOs, however, extends beyond the mere inconvenience of having one's homepage hijacked. By definition, BHOs are in-process Component Object Model (COM) DLLs which Internet Explorer loads on startup. The result is that a BHO has complete access to Explorer's process address space. In practical terms, this means a BHO can intercept all of the events occurring in the user's browser. For example, the `BEFORENAVIGATE2` event is triggered before Internet Explorer navigates to a Web page. This means that a BHO has access to the URL before the page is even downloaded. More alarming is the fact that BHOs are not limited to acting on browser events. Really, anything is possible within the constraints of the permissions of the user who launched Internet Explorer. This includes creating or deleting files, executing programs, reading email, and recording and sending private Internet banking information. What makes BHOs particularly troubling is that it is not obvious that they are running. Since they run as a DLL within Internet Explorer, it is almost impossible to distinguish a malicious BHO from a completely benign one.

KERNEL MODE TECHNIQUES (CALL TABLE HOOKING AND FILTER DRIVERS)

In our discussion of user mode we noted that user applications coexist with some portions of the operating system. We now extend that statement to kernel mode. At this highest of privilege levels, drivers coexist with the Windows kernel itself. This means that a malicious driver has the power to usurp complete control of the operating system environment. Indeed, modern rootkits have reached an alarming degree of sophistication in their employment of kernel mode hooking techniques. Fortunately, writing a kernel driver is something of a "black art," so that, with the exception of a very few advanced key loggers, most spyware developers haven't caught up to the rootkit developers *yet*. Nevertheless, as spyware continues to evolve in complexity and stealth, these techniques may become a very real threat to information security. In the following discussion we cover three of the most common kernel hooking techniques: system call table hooking, filter drivers, and IRP table hooking.

The system call table (SCT) is one of the simplest, most effective places to hook in the kernel. It provides the gateway into the kernel through which all user mode API calls must pass. In most operating systems the SCT is implemented as a table of pointers to the functions the kernel exports to user mode applications. The Windows system call mechanism is also based on this concept. Calls to the Kernel32.dll and Ntdll.dll API functions pass through the kernel function called KiSystemService, which does some sanity checking on the function parameters and then calls the referenced SCT function. By modifying the function pointer in the table to point to attack code, an attacker has total control over the operation of the function (see Figure 2). Spyware and rootkits can use this trick to filter information they do not want the user or system administrator to see. For example, by hooking NtQuerySystemInformation in the SCT and filtering its response, the attacker can hide any file or directory in Windows. Although the idea of modifying a table of function pointers is reminiscent of userland IAT hooking, kernel SCT hooking is a much more powerful technique. Where IAT

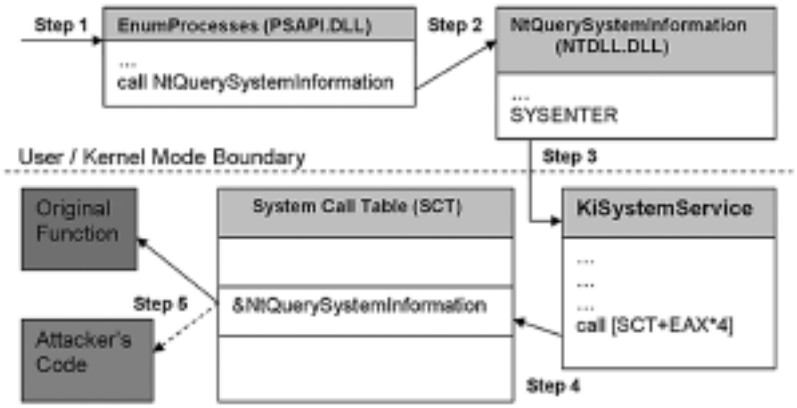

Figure 2. Normal execution path vs. hooked execution path for an SCT hook

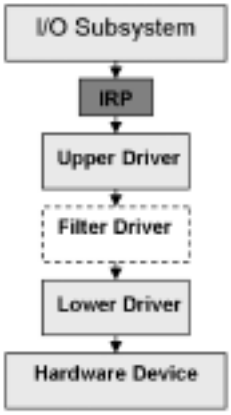

Figure 3. Windows has a layered driver architecture

hooking is local to the application process being hooked, an SCT hook will globally intercept functions across all processes, including the operating system itself.

In Windows, the drivers for a system's hardware devices are layered into a hierarchal "device stack." Furthermore, a given hardware device may have one or more drivers associated with it, which we can visualize as a vertical stack of layered components (see Figure 3). These drivers communicate with each other and the operating system by means of I/O request packets (IRPs). Filter drivers and IRP hooking techniques exploit the layered nature of Windows' driver architecture. Unlike normal drivers, filter drivers are transparently inserted on top of or in between existing drivers in the stack, using the kernel API, IoAttachDevice. Although they are sometimes legitimately used to add functionality to an existing lower-level driver, an attacker will typically use them to either modify or intercept data. Key loggers and network sniffers typically use them to capture user passwords and other sensitive information.

By design, layered filter drivers require a lot of code to implement. Furthermore, they may be more easily detected than a driver that does direct IRP hooking. Instead of installing a filter, an attacker can directly hook the functions exported by a target device driver in its IRP major function table. The IRP major function table is simply an array of 28 function pointers to handler functions in response to notifications and requests which the driver receives from either a client application or the operating system. An application typically sends an IRP to a driver to request a specific service. For example, `IRP_MJ_DIRECTORY_CONTROL` is sent to file system drivers to

request the list of directories and files. By intercepting and altering different I/O requests, spyware can easily hide on the file system or eavesdrop on network communications. Some commercial software such as ZoneAlarm[4] even uses this technique to intercept and regulate network traffic. It is not difficult for an attacker to find a particular driver object in memory. Windows' kernel provides the function called IoGetDeviceObjectPointer which spyware or a rootkit can call to get a pointer to the named device object. This device object contains a pointer to its corresponding driver object which the attacker can use to reference the target's IRP function table. Figure 4 shows the relationship among these objects in memory.

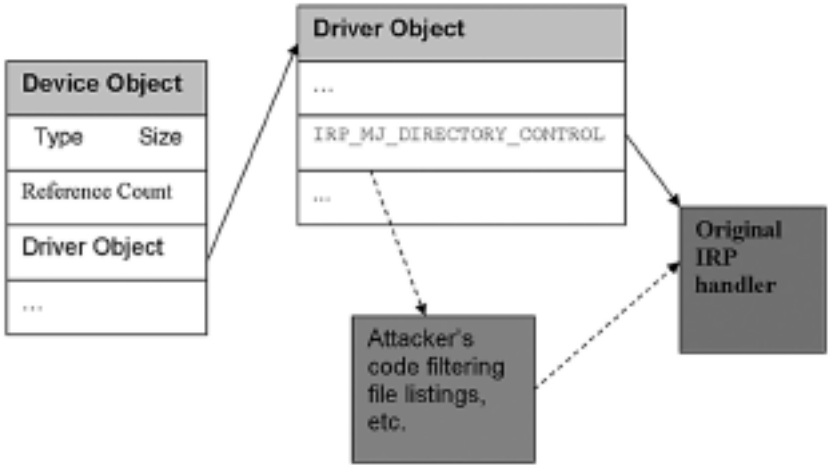

Figure 4. Illustration of hooking a driver's IRP table

HYBRID TECHNIQUES (INLINE FUNCTION HOOKING)

Inline function hooking can be considered a hybrid technique, since it can be applied either to a user mode application or to the kernel. This technique is a bit more advanced and harder to detect than the methods previously mentioned. Rather than simply replacing a pointer in a table, an inline hook alters the target function itself. Normally this is done by replacing the first few bytes of the function with an unconditional jump to the rootkit or spyware code. Before this overwrite occurs, the attacker saves the bytes being replaced, so the semantics of the original function are maintained. The trick here is in the fact that instructions are variable length on an X86 processor. Therefore, although an unconditional jump is only five bytes on a 32-bit architecture, the instructions being overwritten may have a different length. Inline function hooking becomes extremely difficult to detect if the jump is embedded deeply in the target function. Complicating matters further, the destination of the jump may be nondeterministic except at the moment of execution. A clever piece of spyware has the full breadth of the assembly language and all of its potential permutations within which to hide.

Managing the Threat: Spyware Detection

There are a number of commercial and freeware spyware detection tools available. Like most things free, some of them are better than others. We found BHO Demon very useful in detecting and disabling BHOs in Internet Explorer.[5] It installs a service on the local machine to watch for future attempts to inject BHOs into Internet Explorer. Spybot Search & Destroy was also very useful during our research.[6] It not only detects BHOs, but additionally detects and removes other forms of spyware and adware.

SPYWARE DETECTION CHALLENGES

Current spyware detection tools are primarily based upon signature scanning techniques. Signature scans have been used heavily by antivirus (AV) engines and are quite reliable for detecting known strains of spyware. Unfortunately, they are ineffective against unknown strains, which must first be caught, analyzed, and sampled for a usable signature. With new spyware variants emerging almost daily, it is difficult for detection engines to keep pace. Indeed, there are even a handful of spyware applications which utilize a rudimentary form of polymorphism to randomize their file names and registry keys, so that every infected machine contains a slightly different version of the program. This makes it more difficult for a detection program to obtain a consistent signature for the application. Some detectors have turned to heuristics to deal with these issues. A further problem with current detections lies in the fact that many of them run in user mode right alongside the spyware applications they are attempting to apprehend. Using the aforementioned hooking techniques, a malicious spyware application is capable of intercepting the function calls of a user mode detection engine as easily as it hijacks the user's Internet browser. In this manner, clever spyware may trick the detector into believing the machine is uninfected. A detection engine implemented in kernel mode will provide defense against this attack as long as spyware remains a mostly user mode phenomenon.

VICE

VICE is a freeware tool designed to detect hooks.[7] It is based upon heuristic analysis of hooking behaviors rather than exact signatures. The benefit of this approach is that VICE is capable of pinpointing suspicious activity related to previously unknown rootkits or spyware. It is implemented as a stand-alone program capable of analyzing both user mode applications and the operating system. VICE checks the address space of every application looking for IAT hooks in every DLL that those applications use. It also checks the kernel SCT for function pointers that do not resolve to ntoskrnl.exe and the IRP major function tables for a list of user defined drivers. A user can add devices to this list by editing the driver.ini file. Inline function hooks are detected in DLL functions imported by applications, as well as in the kernel SCT functions themselves. When possible, VICE will display the full path on the file system of the DLL or device driver doing the hooking, so that a system administrator can examine and remove the malicious software. It should be noted, however, that VICE is not an end-user spyware detection and removal tool. Some legitimate applications such as firewalls and antivirus products also use these techniques to filter and examine data, so an operator of VICE will need to have experience enough to recognize those cases. As stated previously, many current spyware applications are immature and do not utilize advanced hooking techniques. Nevertheless, as spyware evolves, VICE stands to become an increasingly useful tool, as it has proved to be in the battle against Windows rootkits. Today, VICE will detect most publicly known Windows rootkits and any spyware that currently uses these more intrusive hooking technologies. To run VICE, the host machine must have the Microsoft .NET Framework installed, which is free for download.

Conclusion

We hope to advance VICE as rootkit and spyware techniques continue to evolve. The sophistication of the disassembly engine logic can be improved

to identify more complex and/or deeply embedded inline function hooks. In future versions, VICE will become more of an active forensics tool, with enhanced capabilities for detecting anomalous registry accesses, file accesses, and network communications on systems suspected of being compromised. Although it is still an open-ended topic of research, host-based systems tend to have a better understanding of their environment than do Network Intrusion Detection Systems (NIDS). This translates into an advantage for heuristic systems like VICE that rely on their ability to differentiate between "normal" and "abnormal" behaviors.

Spyware has become a threat to corporate and personal information security. With the combined goals of data interception and stealth, a spyware application is well suited to leverage both trojan and rootkit technologies. Although current spyware lacks the sophistication of modern rootkits for hiding its presence on a system, we can expect that to change with the advent of more advanced detection and removal tools. In the next generation of spyware we expect to see more complex hooking, polymorphic techniques, and kernel mode components. By understanding the potential application of rootkit stealth techniques to spyware, hopefully we will be better prepared to meet the coming challenges in detection and removal.

REFERENCES

1. Websense, "Emerging Internet Threats Survey 2003": http://www.websense.com/company/news/research/Emerging_Threats_2003_EMEA.pdf.

2. CoolWebSearch: http://www.coolwebsearch.com.

3. Matt Pietrek, "Inside Windows: An In-Depth Look into the Win32 Portable Executable File Format." *MSDN Magazine*, vol. 17, no. 2 (February 2002).

4. ZoneAlarm Pro: http://www.zonelabs.com.

5. BHO Demon: http://www.definitivesolutions.com/.

6. Spybot Search & Destroy, by Patrick M. Kolla: http://www.safer-networking.org/.

7. VICE, by James Butler, HBGary Inc.: http://www.rootkit.com/vault/fuzen_op/vice.zip.