Detecting Stealth Software with Strider GhostBuster

Yi-Min Wang, Doug Beck, Binh Vo, Roussi Roussev, and Chad Verbowski
Microsoft Research, Redmond

Abstract

Stealth malware programs that silently infect enterprise and consumer machines are becoming a major threat to the future of the Internet [XZ04]. Resource hiding is a powerful stealth technique commonly used by malware to evade detection by computer users and anti-malware scanners. In this paper, we focus on a subclass of malware, termed "ghostware", which hide files, configuration settings, processes, and loaded modules from the operating system's query and enumeration Application Programming Interfaces (APIs). Instead of targeting individual stealth implementations, we describe a systematic framework for detecting multiple types of hidden resources by leveraging the hiding behavior as a detection mechanism. Specifically, we adopt a ***cross-view diff-based approach*** *to ghostware detection by comparing a high-level infected scan with a low-level clean scan and alternatively comparing an inside-the-box infected scan with an outside-the-box clean scan. We describe the design and implementation of the Strider GhostBuster tool and demonstrate its efficiency and effectiveness in detecting resources hidden by real-world malware such as rootkits, Trojans, and key-loggers.*

1. Introduction

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 malware 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 video card EEPROM and bad disk sectors, from user-mode API interception to kernel-mode data structure manipulation, and from individual trojanized OS utilities to OS patching with system-wide effect. Stealth software presents a major challenge to trustworthy computing by making it extremely difficult for computer users to answer the question: *"Has my machine been compromised?"* Information on stealth techniques [YN04] and easy-to-use, configurable tools for providing stealth capabilities [ZH,ZA] are becoming increasingly available. Installations of such unwanted software on user machines through vulnerability exploits [XS04,YB04], spam emails [XG04], and bundling with freeware are becoming widespread [XZ04]. The increasing uses of malware-infected machines in computer crimes such as phishing, spamming, DOS attacks, keystroke logging, etc. [XG03,XG04,YC04,XA04,XS04,XP04,XW04] pose a serious threat to the future of the Internet and the computing industry.

It is very difficult to reason about the general stealth software problem and to create solutions because stealth behavior is not well-defined. In this paper, we focus on an important subclass of stealth software, which we call *"ghostware"* [W04] for ease of presentation. 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. The hiding behavior is typically achieved through API interception and filtering [YN04] (e.g., intercepting file enumeration API calls and removing the to-be-hidden entries from the returned result set) or Direct Kernel Object Manipulation (DKOM, e.g., removing to-be-hidden processes from the Active Process List data structure) [YV04]. Ghostware encompasses at least three types of malware and commercial software: (1) rootkits [PFM+04,ZR,YO03,XP03] and Trojans that hide their executable files and process instances [ZH,ZV]; (2) commercial key-loggers that hide their log files containing keystrokes and screenshots; and (3) commercial file-hiders that hide user-specified files [ZHF]. Although some of these ghostware programs may have legitimate uses, resource-hiding behavior is generally considered highly undesirable from the user's perspective.

There are two different approaches to ghostware detection. The first approach targets the hiding mechanism by, for example, detecting the presence of API interceptions [YI,ZVI,YK,YKS,YV04]. It has at least two disadvantages: first, it cannot catch ghostware programs that do not use the targeted mechanism; second, it may catch as false positives legitimate uses of API interceptions for in-memory software patching, fault-tolerance wrappers, etc. The second approach targets the hiding behavior by detecting any discrepancies between "the truth" and "the lie". For example, comparing the output of "ls" and "echo *" can detect an infected "ls" program [B99].

In this paper, we propose a framework for the second approach, called GhostBuster, that systematically accommodates multiple resource types and provides a convenient inside-the-box solution as well as a more fundamental outside-the-box solution. To detect each type of hidden resource, we divide the problem into two parts.

First, we perform both a high-level and a low-level scan of the resources in an inside-the-box solution. When a

Proceedings of the 2005 International Conference on Dependable Systems and Networks (DSN'05)
0-7695-2282-3/05 $20.00 © 2005 IEEE

ghostware program implements its hiding mechanism between the two levels as shown in Figure 1, the high-level scan contains "the lie" and the low-level scan contains "the truth" so that their difference exposes the hidden resources. Specifically, the "Master File Table", the "Raw Hive Files", and the "Kernel Process List" are the low-level resources that we scan to detect hidden files, Registry entries, and processes, respectively. A major advantage of such a completely inside-the-box solution is that it is convenient, efficient and scalable: users can quickly scan their machines daily or as needed without having to reboot and corporate IT organizations can remotely deploy the solution on a large number of desktops without requiring user cooperation. A disadvantage is that a ghostware program running with sufficient privilege can always try to defeat the solution by interfering with the low-level scan. Another related issue occurs due to the discrepancies between the truth and the "truth approximation", which we discuss later.

Second, to avoid scan interference from a ghostware-infected OS, our framework aims at exporting the truth so that it can be scanned outside the box from a clean OS; the scan is then compared against the inside-the-box generated high-level scan to expose hidden resources, as shown in Figure 1. While persistent-state resources such as files and Registry entries are naturally available outside, volatile-state resources such as processes and loaded modules require a mechanism to persist relevant kernel data structures (see Section 4). Our current implementation uses Windows Preinstallation Environment (WinPE) CD [WPE] as the clean OS. Since the ghostware programs are not running when we perform a scan from WinPE, there will not be any hiding or malicious interference. This implies that an outside-the-box solution is more fundamental. However, this solution is less convenient and therefore users will only be willing to run it on an infrequent basis or when they suspect that their machines have been compromised.

There is a subtle but important difference between the *"cross-view diff"* used in GhostBuster and the more common *"cross-time diff"* used in Tripwire [KS94] and the Strider Troubleshooter [WVS03,WVD+03]. The goal of a cross-time diff is to capture changes made to persistent state by essentially comparing snapshots from two different points in time (one before the changes and one after). In contrast, the goal of a cross-view diff is to detect hiding behavior by *comparing two snapshots of the same state at exactly the same point in time, but from two different points of view* (one through the ghostware and one not). Cross-time diff is a more general approach for capturing a broader range of malware programs, hiding or not; the downside is that it typically includes a significant number of false positives stemming from legitimate changes and thus requires additional noise filtering, which has a negative impact on usability. In contrast, cross-view diff targets only ghostware and usually has zero or very few false positives because legitimate programs rarely hide.

This paper is organized as follows. Section 2 describes stealth techniques that are used to hide files, analyzes the implementations of actual file-hiding ghostware programs, presents the design and implementation of GhostBuster for hidden-file detection, and evaluates its performance. Sections 3 and 4 apply the same framework to the detection of ghostware programs that hide Windows Registry entries, processes, and modules, respectively. Section 5 presents GhostBuster extensions designed to avoid being targeted by malware and for automating the outside-the-box detection process. Although the main focus of this paper is on Windows ghostware, we show in Section 5 that the GhostBuster approach can be applied to detecting actual Linux/Unix ghostware as well. Section 6 concludes the paper and outlines future work.

2. Detection of File-hiding Ghostware

Between a user-mode file-query program (such as the "dir" command in a cmd window) and the physical disk, there exist many layers where ghostware programs can insert themselves to intercept and filter resource queries. Figure 2 illustrates the six different techniques employed by the 10 file-hiding ghostware programs in our collection. Of these 10 programs, Urbin, Mersting, Vanquish, Aphex, and Hacker Defender are rootkits and Trojans, ProBot SE is a key-logger, and Hide Files, Hide Folders XP, Advanced Hide Folders, and File & Folder Protector are commercial file hiders.

Urbin and Mersting make modifications at the highest level by altering the per-process Import Address Table (IAT) [R00] entries of file enumeration APIs to point to their Trojan import functions. In contrast, Vanquish directly modifies the loaded, in-memory API code so that its function is called and then it calls the next OS function. Both techniques cause the Trojan functions to appear in the call stack trace of a kernel or user-mode debugging session. To achieve better stealth, Aphex and Hacker Defender modify the in-memory API code with a jump to the Trojan code along with a Trojan code jump back to the next instruction after the API detour [HB99]; the Trojan code modifies the return address on the stack to cause its code to be executed in the return path. The only difference is that Aphex modifies the FindFirstFile and FindNextFile API code inside Kernel32.dll [YN04] (denoted by Kernel32!FindFirst(Next)File), while Hacker Defender modifies the lower-level NtQueryDirectoryFile API exported by NtDll.dll [YH03].

System-wide file hiding can also be achieved by running a centralized, kernel-mode intercepting program, without modifying per-process memory. ProBot SE

hijacks kernel-mode file-query APIs by modifying their dispatch entries in the Service Dispatch Table [YT04]. All four commercial file hiders use a filter driver [IFS] that is inserted into the OS file system stack to intercept all file operations. The filter driver can scope the file-hiding behavior to specific processes by examining the IRP (I/O Request Packet) for the I/O operation to determine the originating process.

Another form of file hiding exploits the restrictions on filenames enforced by some Win32 API, but not the NTFS file system. If a ghostware program creates a file using low-level APIs in an NTFS file system that breaks any of the restrictions, Win32 API calls will not be able to access them. Examples include long full pathnames, filenames with trailing dots or spaces, filenames containing special characters, reserved filenames, etc. [MSDN].

To detect files hidden by ghostware, our GhostBuster tool performs the high-level scan using either the FindFirst(Next)File APIs or the "dir /s /b" command, and performs the low-level scan by reading the Master File Table (MFT), which is the core of the NFTS volume structure [SR00], and other NTFS metadata directly. A comparison of the two scans reveals hidden files. We note that not all files revealed by the comparison necessarily come from ghostware programs; for example, some benign software bugs may generate filenames that are too long, or files may be created in the very small time window between when the high- and low-level scans are taken. However, in practice the noise level from this is extremely low. For the outside-the-box solution, our tool scans the potentially infected hard drive from a clean WinPE CD boot using either the Win32 APIs, the "dir /s /b" command, or the low-level scanning code, and compares the result with the inside high-level scan.

Experimental Results

Figure 3 shows the GhostBuster detection results for the 10 file-hiding ghostware programs. The Urbin and Mersting Trojan samples were captured from actual infected machines. The remaining programs were obtained from various Web sites. The hidden files are divided into three categories: (1) ghostware binaries including EXEs, DLLs, and drivers; (2) ghostware data files such as .ini configuration files and .log files; and (3) other target files specified to be hidden by either rootkits/Trojans or file hiders. The results clearly demonstrate the major advantage of the GhostBuster cross-view diff approach: it can uniformly detect files hidden by ghostware programs implemented with a wide variety of interception techniques.

The execution time for hidden-file detection depends on the disk size, speed, and usage. We tested GhostBuster on 8 machines including 4 corporate desktops, 3 home machines, and 1 laptop. Seven machines had disk usage ranging from 5 to 34GB and CPU speed ranging from 550MHz to 2.2GHz. For these machines the inside-the-box solution took between 30 seconds and 7 minutes. (On the 8th machine, which is a dual-proc 3GHz workstation with 95GB of the 111GB hard drive utilized, the scan took 38 minutes.) The outside-the-box solution typically adds 1.5 to 3 minutes for booting into the WinPE CD.

We did not observe any false positives on any inside-the-box scans. However, in the outside-the-box solution, the larger time gap between the two scans and the file activities during reboot did introduce some false positives. They were mostly log files generated by always-running services (such as anti-virus real-time scanners and Change and Configuration Management (CCM) services), System Restore [SR] file-change log entries, OS prefetched files [PF], and browser temporary files. On all but one machine, the number of false positives was two or less and they were easily filtered out through manual inspection. On the one machine that had 7 false positives, we disabled the CCM service, re-ran the scan, and saw the number of false positives reduced to 2.

3. Detection of Registry-hiding Ghostware

The Windows Registry is a centralized, hierarchical store for configuration data containing name-value pairs. A Registry key is like a file-system folder and can contain one or more Registry items (or values). The Registry is composed of several "hives" [SR00], each of which is backed by a file; for example, "C:\windows\system32\config\system" stores the HKLM\system hive, and "ntuser.dat" in the user profile folder stores the per-user sub-hive under the HKU hive.

Most Windows ghostware programs we studied do not modify OS files, presumably for two reasons: the Windows system source code is not widely available, and there are many easy-to-use Auto-Start Extensibility Points (ASEPs) [WRV+04] that applications can "hook" to get automatically started as essentially "part of the system". Most of the ASEPs reside in the Registry. Examples include the HKLM\SYSTEM\CurrentControlSet\Services Registry key for auto-starting drivers and services, the HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run key for auto-starting additional processes, and the HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects for auto-loading DLLs into the Internet Explorer browser.

By extensively studying 120 real-world spyware programs, we have shown that the ASEP-based monitoring and scanning technique is effective for detecting spyware programs [WRV+04]. In a similar study of 30 malware programs, we found that each hooked at least one Registry-based ASEP. Since ASEP hooks are critical for their continued operation across reboots, many ghostware

programs hide their hooks to evade detection and resist removal.

The first six file-hiding ghostware programs that we analyzed in the previous section, namely Urbin, Mersting, Vanquish, Aphex, Hacker Defender, and ProBot SE, also hide their ASEP hooks. The hiding techniques are similar to those illustrated in Figure 2 except that the file-related APIs Kernel32!FindFirst(Next)File and NtDll!NtQueryDirectoryFile are replaced by the Registry-related APIs Advapi32!RegEnumValue and NtDll!NtEnumerateKey, respectively. Alternatively, ghostware programs can use the kernel-level Registry callback functionality to intercept and filter Registry query results.

Another form of Registry hiding exploits differences in the way Win32 API and the Native API interpret Registry entry names: the former assumes NULL-terminated strings, while the latter uses counted Unicode strings [YH]. As a result, Registry entries created with the Native API can be hidden from most of the Registry editors that use the Win32 API by embedding NULL characters as part of the name. Yet another form of Registry hiding exploits some Registry editors' software bugs in handling long names that allow certain entries to become invisible. The GhostBuster hidden-Registry detection tool described next can detect these two forms of hiding as well.

GhostBuster uses either the standard Win32 Registry enumeration APIs or the RegEdit program for the high-level scan of all ASEP hooks. Since each Registry hive is simply a file with a well-defined schema [SR00], our low-level scan copies and parses each hive file directly to retrieve all ASEP hooks thus bypassing the APIs. These copies are "truth approximation" instead of the truth itself, as shown in Figure 1, because some ghostware programs may eventually be able to interfere with the copying process. For the outside scan, GhostBuster mounts Registry hive files from the potentially infected system drive under the live Registry loaded from the WinPE CD, and uses the Win32 APIs or RegEdit to scan all ASEP hooks to extract the truth.

Detection of hidden ASEP hooks is particularly useful for ghostware removal: it locates the Registry keys that can be deleted to disable the ghostware after a reboot, even if the ghostware files still remain on the machine. It also reveals the pathnames of the associated program files; the user can locate and remove those files once the machine is rebooted and those files are no longer hidden. Alternatively, on-demand anti-virus scan can be invoked to remove those key files as well as other auxiliary files installed by the ghostware.

Experimental Results

Figure 4 shows the results of GhostBuster detecting six Registry-hiding ghostware programs. Both of the Trojans that came from the wild, Urbin and Mersting, hook the AppInit_DLLs ASEP to allow their DLL to be loaded into every process that loads User32.dll [AID]; they both hide the ASEP hook. Hacker Defender hides both of its ASEP hooks, one for the service hxdef100.exe and the other for the driver hxdefdrv.sys. Vanquish and ProBot SE similarly hide their service and driver hooks. ProBot SE and Aphex hide their Run key hooks for starting additional user-mode processes.

On the 8 machines we tested, inside-the-box hidden-ASEP detection took between 18 to 63 seconds. In all the experiments, we observed only one false positive on one machine: the data field of the AppInit_DLLs entry contained corrupted data that did not show up in RegEdit, but appeared in the raw hive parsing. The problem was fixed by exporting the parent key (to a text file without the corrupted data), by deleting the parent key, and then by re-importing the exported key.

4. Detection of Process/Module-hiding Ghostware

In addition to hiding persistent state such as files and Registry entries, some ghostware programs hide processes from the commonly used Task Manager utility and the tlist command-line utility that is popular among systems administrators. In many environments, process hiding is considered more important than file and Registry hiding because, while there are typically hundreds of thousands of files and Registry entries [WVD+03], there are usually only tens of processes running on a machine and so it may be feasible for the user to go through the entire list in an attempt to identify suspicious entries.

Figure 5 illustrates the different methods used by the four process-hiding ghostware programs in our collection. Aphex intercepts process list queries by modifying the IAT entry for the NtDll!NtQuerySystemInformation API, while Hacker Defender and Berbew hijack the queries by putting a "jmp" instruction inside the NtDll!NtQuerySystemInformation in-memory code. The DKOM implementation of the FU rootkit presents a unique challenge: it hides a process by removing its corresponding entry from the Active Process List kernel data structure. Since the list is only a "truth approximation" for responding to process list queries and does not necessarily reflect all the processes that own at least one schedulable thread [YK04], a process can be absent from the list while remaining fully functional. One can even use the FU rootkit to hide the other process-hiding ghostware programs to increase their stealth.

Although less popular, administrator tools such as Process Explorer [PE], AskStrider [WR+04] and tlist can be used to enumerate all modules (e.g., DLLs) loaded by each process and all drivers loaded by the system to detect any suspicious entries. For example, AskStrider can be used to quickly detect a Hacker Defender infection today by revealing its unhidden hxdefdrv.sys driver. To hide its modules from these tools and the underlying Kernel32!Process32First(Next)→NtDll!NtQueryInformationProcess calling chain, the Vanquish rootkit blanks out the pathname of vanquish.dll from the Process Environment Block (PEB) module list in the per-process memory.

For inside-the-box hidden-process detection, our tool performs the high-level scan using the NtDll!NtQuerySystemInformation API, and uses a driver to perform the low-level scan by traversing the Active Process List. A diff between these two scans can reveal API-intercepting ghostware, but it cannot detect processes hidden through DKOM because the low-level scan does not contain the truth. To detect the latter, our tool supports an advanced mode that performs the low-level scan by traversing another kernel data structure that maintains the process list to support OS functionalities other than responding to enumeration queries. Similarly, to detect hidden modules, our low-level scan extracts the truth of all modules loaded by all processes from a kernel data structure and compares that with the high-level API scan.

Outside-the-box scanning of volatile resources such as processes and modules requires an additional step of making the image of the relevant memory address space available outside. Ideally, a PCI-add-in card as described in the Copilot paper [PFM+04] or a Myrinet NIC as described in the Bookdoors paper [BNG+04] should be used to retrieve volatile data through Direct Memory Access (DMA) without the knowledge or intervention of the potentially infected OS. To allow users without the extra hardware to use GhostBuster today, we obtain a "truth approximation" by inducing a blue screen (i.e., kernel crash) to generate a memory dump file, and apply similar kernel data structure traversal code to the dump file to perform the outside-the-box scan. This is only an approximation because future ghostware programs can potentially trap the blue-screen events and remove all traces of themselves from the memory dump.

Experimental Results

Figure 6 shows the results of GhostBuster detecting four process-hiding and one module-hiding ghostware programs. The first three, namely Aphex, Hacker Defender, and Berbew, can be detected by using the Active Process List as the truth, while FU can only be detected by running GhostBuster in the advanced mode. Since the hidden vanquish.dll is injected into many processes, the GhostBuster report contains many such entries.

The inside-the-box scanning and diff for the combined hidden-process and hidden-module detection took between 1 and 5 seconds. It is conceivable that false positives can be introduced if any process happens to get started or terminated during that short interval, but we have not encountered any false positives in our experiments so far. For the outside-the-box scan, the kernel memory dump through blue screen added 15 to 45 seconds.

5. Extensions

Ghostware Targeting Issues

It is possible for ghostware to target specific OS utilities; for example, a process-hiding ghostware program may choose to hide processes only from Task Manager and tlist. The GhostBuster design described so far will not detect such ghostware because the tool cannot experience the hiding behavior. It is also possible for ghostware to target GhostBuster so that resources are hiding from all running programs except the GhostBuster process.

To address these two issues, we have implemented a GhostBuster extension in the form of a DLL. Instead of running the GhostBuster EXE that can be easily targeted, we inject the GhostBuster DLL into every running process and perform the scans and diff from inside each process, essentially turning every process into a GhostBuster. In particular, OS utilities such as Windows Explorer, Task Manager, and RegEdit are now all GhostBusters. This makes it more difficult for ghostware programs to target GhostBuster, while hiding from common utilities. Injecting GhostBuster into an anti-virus scanner is particularly interesting. As a demonstration, we set up a Hacker Defender-infected machine, installed the eTrust anti-virus software, and ran the on-demand scanner. The scanner could not detect Hacker Defender, even though it did have the known-bad signatures. We then injected the GhostBuster DLL into the scanner process InocIT.exe; that allowed InocIT.exe to detect Hacker Defender's hidden entries. This combination creates a dilemma for malware programs: they will be detected by GhostBuster if they hide from InocIT.exe and by the eTrust signatures if they do not hide.

Another potential attack on GhostBuster is to hide a large number of innocent files, together with the ghostware files. Although this would make it more difficult to identify ghostware files, the GhostBuster tool still provides effective detection of ghostware infection in this case because the existence of a large number of hidden files is a serious anomaly.

Automating the Outside-the-box Solution

There are two extensions that can be implemented to automate the outside-the-box ghostware detection and greatly enhance its usability. In an enterprise environment, the CD boot can be replaced by a network boot through the Remote Installation Service (RIS) [RIS]: upon a reboot, a client machine contacts the RIS server to obtain a network boot loader, which then performs the outside-the-box scan and diff.

Another approach is to use a Virtual Machine (VM): we include a clean version of VM on the GhostBuster WinPE CD and ask the user to boot the machine from the CD. The GhostBuster process will automatically start and insert an ASEP hook to the potentially infected boot drive to enable auto-start of a high-level scan, and then launch a VM instance to boot from that drive. Inside the VM instance, the scanning code will save the scan result file and notify the host machine of its completion. The host machine then "powers down" the VM instance, grabs the released infected drive, executes the outside-the-box scanning code, removes the added ASEP hook, and invokes the comparison code. If no hidden resource is detected, the machine is rebooted normally into its boot drive.

A similar technique can be used when the ghostware-infected machine is in fact a VM. As a demonstration, we set up a Hacker Defender-infected VM using the Virtual PC software and performed an infected scan inside the VM. We then "powered down" the VM and scanned the virtual disk drive from the host machine using a utility that allows a virtual drive to appear as a normal drive on the host machine. A diff of the two scans revealed all the hidden files and contained zero false positive because the two scans were performed on exactly the same drive image in such a VM setting.

Detecting Linux/Unix Ghostware

Similar ghostware problems exist on the Linux/Unix platforms as well [PFM+04,YKS,YC,YW98,B99,YA03]. (In fact, the term "rootkit" originated from the root privilege concept on Unix platforms.) A common technique used by Linux/Unix ghostware programs to hide resources is to intercept system calls to the kernel via a Loadable Kernel Module (LKM) [ZK,YJ,J01]. For example, some rootkits are known to hook read, write, close, and the getdents (get directory entries) system calls. More advanced rootkits can directly patch the kernel in memory [YC98,YL01].

We have experimented with several file-hiding rootkits including Darkside 0.2.3 [ZD] for FreeBSD, and Superkit [ZS] and Synapsis for Linux. For the inside-the-box high-level scan, we used the "ls" command to scan all mounted partitions. For the outside-the-box scan, we used the same command from the clean, bootable CD distribution of the OS to scan the same set of partitions. Our results showed that the cross-view diff reports contained zero or very few false positives: in all cases, the number of false positives was four or less, and they were mostly temporary files and log files generated by system daemons such as FTP. We also experimented with the T0rnkit rootkit [ZT] that replaces OS utility programs with trojanized versions. The GhostBuster approach could detect its hidden files as well.

6. Conclusions

Stealth malware programs are becoming a serious threat to the future of the Internet, and yet they have been dealt with mostly in an ad-hoc fashion. In this paper, we have described a cross-view diff-based framework for systematic detection of ghostware programs that hide files, Registry, processes, and loaded modules. We have proposed using the inside-the-box diff of a high-level scan and a low-level scan to provide an efficient, automatic solution that can be run frequently to detect most of today's ghostware programs. Experimental results have shown that it takes only seconds to detect hidden processes and modules, tens of seconds to detect hidden critical Registry entries, and a few minutes to detect hidden files. In the case of Hacker Defender, the most popular Windows rootkit today according to Product Support Service engineers, we were able to deterministically detect its presence within 5 seconds through hidden-process detection, locate its hidden auto-start Registry keys within one minute, remove the keys to disable the malware, and reboot the machine to delete the now-visible files.

We have also proposed an outside-the-box, CD-boot solution to detect more advanced ghostware that may interfere with the inside-the-box scans. Experimental results based on 12 real-world ghostware programs showed that, while they employ a wide variety of resource-hiding techniques, they can all be uniformly detected by GhostBuster's diff-based approach that targets the hiding behavior and effectively turns the problem into its own solution. False positives in a cross-view diff report are minimal and can be easily filtered out.

As we pointed out in the Introduction, the problem space of stealth software is broader than that of ghostware, which has been our focus so far. Stealth software may hide their persistent state in a form for which current OS does not provide query/enumeration APIs or does not provide common utilities that make use of such APIs. Examples include hiding executable code inside the BIOS [YB], video card EEPROM, boot sectors [D], bad disk sectors, Alternate Data Streams (ADS), etc. Stealth software can also hide their active running code in a form that cannot be revealed by the process/module query APIs; they can inject code into an existing process and hijack a thread to execute that code. Detection of these advanced hiding

techniques is beyond the scope of this paper and we plan to pursue them as future work.

As a final note, most of today's Windows rootkits do not modify OS files or memory image; rather, they "extend" the OS through ASEP hooking in a way that is indistinguishable from many other good software programs that also extend the OS. Therefore, it is difficult to apply the genuinity tests and software-based attestation techniques that detect deviations from a known-good hash of a well-defined OS memory range [KJ03,SPDK04]. On the other hand, these techniques can detect both hiding and non-hiding malware programs that modify the OS and are complementary to the GhostBuster approach.

Acknowledgement

We would like to express our sincere thanks to David Brumley, Aaron Johnson, Lee Yan, Bill Arbaugh, Dan Simon, and Brad Daniels for their valuable discussions and to the reviewers for their valueable comments. The MFT-based technique was inspired by discussions with Robert Hensing, and the implementation was based on the code provided by Ed Elliott and Takefumi Kakimoto.

References

[AID] Working with the AppInit_DLLs registry value, http://support.microsoft.com/kb/q197571/.

[B99] D. Brumley, "Invisible Intruders: Rootkits In Practice," ;login: The Magazine of USENIX and SAGE, http://www.usenix.org/publications/login/1999-9/features/rootkits.html, 1999.

[BNG+04] A. Bohra, I. Neamtiu, P. Gallard, F. Sultan, and L. Iftode, "Remote Repair of Operating System State Using Backdoors," in *Proc. Int. Conf. on Autonomic Computing (ICAC)*, pp. 256-263, May 2004.

[D] Chapter 7 - Disk, File System, and Backup Utilities, Microsoft TechNet, http://www.microsoft.com/technet/prodtechnol/winntas/support/utilitys.mspx.

[HB99] Galen Hunt and Doug Brubacher. "Detours: Binary Interception of Win32 Functions," in *Proc. the 3rd Usenix Windows NT Symposium*, pp. 135-143, July 1999 (http://research.microsoft.com/sn/detours/).

[IFS] IFS Kit - Installable File System Kit, http://www.microsoft.com/whdc/devtools/ifskit/default.mspx.

[J01] K. Jones, "Loadable kernel modules," ;login: The Magazine of USENIX and SAGE, http://www.usenix.org/publications/login/2001-11/pdfs/jones2.pdf, Nov. 2001.

[KJ03] Rick Kennell and Leah H. Jamieson, "Establishing the Genuinity of Remote Computer Systems," In *Proc. USENIX Security Symposium*, August 2003.

[KS94] G. H. Kim and E. H. Spafford, "The Design and Implementation of Tripwire: A File System Integrity Checker," in *Proc. of the 2nd ACM Conf. on Computer and Communications Security*, pp. 18-29, Nov. 1994.

[MSDN] Naming a File, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/fileio/base/naming_a_file.asp.

[PE] Process Explorer, http://www.sysinternals.com/ntw2k/freeware/procexp.shtml.

[PF] How to Disable the Prefetcher Component in Windows XP, http://support.microsoft.com/?kbid=307498.

[PFM+04] Nick L. Petroni, Jr., Timothy Fraser, Jesus Molina, and William A. Arbaugh, "Copilot - a Coprocessor-based Kernel Runtime Integrity Monitor," in *Proc. Usenix Security Symposium*, Aug. 2004.

[R00] John Robbins, Debugging Applications, 2000.

[RIS] Remote Installation Services, http://www.microsoft.com/windows2000/en/datacenter/help/default.asp?url=/windows2000/en/datacenter/help/sag_RIS_Default_topnode.htm.

[SPDK04] A. Seshadri, A. Perrig, L. van Doorn, and P. Khosla, "SWATT: SoftWare-based ATTestation for Embedded Devices," in *Proc. IEEE Symp. on Security and Privacy*, May 2004.

[SR] Windows XP System Restore, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwxp/html/windowsxpsystemrestore.asp.

[SRM] System Restore Monitored File Extensions, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/sr/sr/monitored_file_extensions.asp.

[SR00] D. A. Solomon and M. E. Russinovich, Inside Microsoft Windows 2000, Third Edition, 2000.

[W04] "Strider GhostBuster: Why It's A Bad Idea For Stealth Software To Hide Files," Yi-Min Wang, *Usenix Security Symposium*, Work-in-Progress Report presentation, http://www.usenix.org/events/sec04/tech/wips/, Aug. 2004.

[WPE] Microsoft Windows Preinstallation Environment (Windows PE), http://www.microsoft.com/licensing/programs/sa/support/winpe.mspx.

[WR+04] Yi-Min Wang, et al., "AskStrider: What Has Changed on My Machine Lately?", Microsoft Research Technical Report MSR-TR-2004-03, Jan. 2004.

[WRV+04] Yi-Min Wang, Roussi Roussev, Chad Verbowski, and Aaron Johnson, "Gatekeeper: Monitoring Auto-Start Extensibility Points (ASEPs) for Spyware Management," in *Proc. Usenix LISA*, Nov. 2004.

[WVD+03] Yi-Min Wang, et al., "STRIDER: A Black-box, State-based Approach to Change and Configuration Management and Support," *Proc. Usenix Large Installation Systems Administration (LISA) Conference*, pp. 159-171, October 2003.

[WVS03] Yi-Min Wang, Chad Verbowski, and Daniel R. Simon, "Persistent-state Checkpoint Comparison for Troubleshooting Configuration Failures", in *Proc. IEEE DSN*, June 2003.

[WVR+04] Yi-Min Wang, Binh Vo, Roussi Roussev, Chad Verbowski, and Aaron Johnson, "Strider GhostBuster: Why It's A Bad Idea For Stealth Software To Hide Files," Microsoft Research Technical Report MSR-TR-2004-71, July 2004.

[XA04] "Alarm growing over bot software," (DOS attacks), CNET News.com, http://news.zdnet.com/2100-1009_22-5202236.html, April 2004.

[XG03] "Guilty Plea in Kinko's Keystroke Caper," (stealing online banking passwords), Security Focus, July 18, 2003.

[XG04] "Gone Phishing: Web Scam Takes Dangerous Turn," (stealing online banking passwords), Wall Street Journal, May 27, 2004.

[XP03] Kevin Poulsen, "Windows Root Kits a Stealthy Threat," SecurityFocus, http://www.securityfocus.com/news/2879, Mar 5 2003.

[XP04] "Phishers tapping botnets to automate attacks", (phishing), The Register, http://www.theregister.co.uk/2004/11/26/anti-phishing_report/, Nov. 26, 2004.

[XS04] "Spreading Web Virus Aims to Steal Financial Data," Reuters, June 25, 2004.

[XW04] "White collar virus writers make cash from chaos," (dialers, spamming), The Register, http://www.theregister.co.uk/2004/12/07/sophos_av_review_2004/, Dec. 2004.

[XZ04] "Zombie PCs: Silent, Growing Threat," PC World, http://www.pcworld.com/news/article/0,aid,116841,00.asp, July 2004.

[YA03] A. Chuvakin, "An Overview of Unix Rootkits," iALERT White Paper, iDefense Labs, http://www.megasecurity.org/papers/Rootkits.pdf, February 2003.

[YB] BIOS and Flash Utilities, http://h20000.www2.hp.com/bizsupport/TechSupport/DriverDownload.jsp?pnameOID=100870&locale=en_US&taskId=135&refresh=true&prodTypeId=12454&prodSeriesId=96495&swEnvOID=1093#2663.

[YB04] C. Boyd, "Xpire/Splitinfinity.info Server Hack and Malware injection using IFRAMES Vulnerability – Condensed Version," http://www.spywarewarrior.com/xpire-splitinfinity-serverhack_malwareinstall-condensed.pdf.

[YC] The chkrootkit tool, http://www.chkrootkit.org/.

[YC04] "Nasty New Parasite," (stealth spyware), Spyware Weekly Newsletter, http://www.spywareinfo.com/newsletter/archives/0604/8.php, June 8, 2004.

[YC98] Silvio Cesare, "Runtime kernel kmem patching," http://vx.netlux.org/lib/vsc07.html, Nov. 1998.

[YH] Hidden Registry Keys, http://www.sysinternals.com/ntw2k/info/tips.shtml#registryhidden.

[YH03] "How to become unseen on Windows NT," http://rootkit.host.sk/knowhow/hidingen.txt, May 8, 2003.

[YI] Ivo Ivanov, "API hooking revealed", http://www.codeproject.com/system/hooksys.asp.

[YJ] A. R. Jones, "A Review of Loadable Kernel Modules," http://www.giac.org/practical/gsec/Andrew_Jones_GSEC.pdf.

[YK] Tan Chew Keong, "ApiHookCheck Version 1.01," http://www.security.org.sg/code/apihookcheck.html, April 15, 2004.

[YK04] Tan Chew Keong, "Win2K Kernel Hidden Process/Module Checker 0.1 (Proof-Of-Concept)," http://www.security.org.sg/code/kproccheck.html, May 23, 2004.

[YKS] KSTAT - Kernel Security Therapy Anti-Trolls, http://s0ftpj.org/en/tools.html.

[YL01] "Linux on-the-fly kernel patching without LKM", http://www.phrack.org/phrack/58/p58-0x07, Phrack Magazine, Dec. 2001.

[YN04] "NTIllusion -- A portable Win32 userland rootkit.txt," Phrack Magazine, July 13, 2004.

[YO03] OpioN, "Kernel Rootkits Explained", http://www.ebcvg.com/articles.php?id=124, March 2003.

[YT04] C. K. Tan, "Defeating Kernel Native API Hookers by Direct Service Dispatch Table Restoration," http://www.security.org.sg/code/SIG2_DefeatingNativeAPIHookers.pdf, July 2004.

[YV04] VICE – Catch the hookers! http://www.blackhat.com/presentations/bh-usa-04/bh-us-04-butler/bh-us-04-butler.pdf.

[YW98] "Weakening the Linux Kernel," Phrack Magazine, http://www.phrack.org/phrack/52/P52-18, Jan. 1998.

[ZA] API Hook SDK 2.12, http://www.devarchive.com/f1709.html.

[ZAF] Aphex – AFX Windows Rootkit 2003, http://www.iamaphex.cjb.net.

[ZAH] Advanced Hide Folders, http://www.hide-folders.biz/.

[ZB] Berbew, http://securityresponse.symantec.com/avcenter/venc/data/backdoor.berbew.j.html.

[ZD] Darkside rootkit, http://www.antiserver.it/backdoor-rootkit/.

[ZF] File & Folder Protector, http://www.softheap.com/ffp.html.

[ZFU] The FU Rootkit, http://www.rootkit.com/vault/fuzen_op/FU_Rootkit.zip.

[ZH] Hacker Defender, http://rootkit.host.sk/.

[ZHF] Hide Files 3.3, http://www.tomdownload.com/new_add/new20031128/hide_files_folders.htm.

[ZHO] Hide Folders XP, http://www.fspro.net/downloads.html.

[ZK] Knark LKM-rootkit, http://www.sans.org/resources/idfaq/knark.php.

[ZP] ProBot SE Monitoring Software, http://www.nethunter.cc/index.php?id=14.

[ZR] RootKits, http://www.rootkit.com/.

[ZS] Superkit rootkit, http://www.remoteassessment.com/darchive/191006794.html.

[ZT] The T0rnkit rootkit, http://www.europe.f-secure.com/v-descs/torn.shtml.

[ZU] The Urbin Trojan, http://vil.nai.com/vil/content/v_125663.htm.

[ZV] Vanquish, https://www.rootkit.com/project.php?id=9.

[ZVI] Vice, http://www.rootkit.com/project.php?id=20.

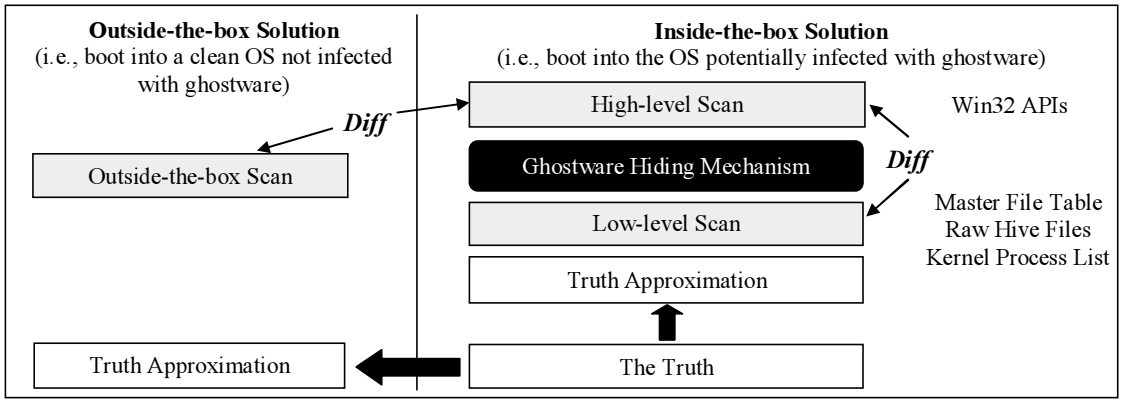

Figure 1. Strider GhostBuster: Combining Inside-the-box and Outside-the-box Scans and Diffs.

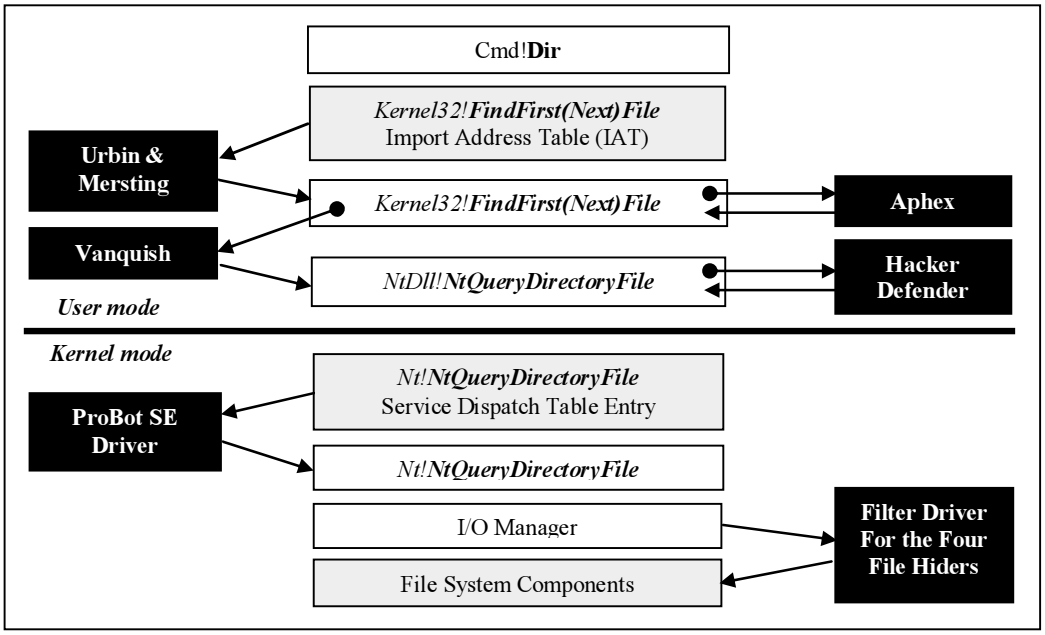

Figure 2. How Ghostware Programs Hide Files

Ghostware	Hidden Files Detected
Urbin [ZU]	1 (C:\windows\system32\msvsres.dll)
Mersting	1 (C:\windows\system32\kbddfl.dll)
Vanquish [ZV]	3+ (C:\windows\vanquish.exe, C:\windows\vanquish.dll, C:\vanquish.log, and any other "*vanquish*" files)
Aphex [ZAF]	Any files with names matching a configurable prefix
Hacker Defender 1.0 [ZH]	3+ (hxdef100.exe, hxdefdrv.sys, hxdef100.ini, and any other files with names matching the patterns specified in hxdef100.ini)
ProBot SE [ZP]	4 (C:\windows\system32\<random name>.exe, C:\windows\system32\<random name>.dll, and two C:\windows\system32\drivers\<random name>.sys files)
File hiders [ZHF,ZHO,ZAH,ZF]	Any user-selected folders and files

Figure 3. Experimental Results for GhostBuster Hidden-File Detection

Ghostware	Hidden ASEP Hooks Detected
Urbin	*HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows\AppInit_DLLs* → msvsres.dll
Mersting	*HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows\AppInit_DLLs* → kbddfl.dll
Hacker Defender 1.0	*HKLM\SYSTEM\CurrentControlSet\Services\HackerDefender100* → hxdef100.exe *HKLM\SYSTEM\CurrentControlSet\Services\HackerDefenderDrv100* → hxdefdrv.sys
Vanquish	*HKLM\SYSTEM\CurrentControlSet\Services\Vanquish* → vanquish.exe
ProBot SE	*HKLM\SYSTEM\CurrentControlSet\Services\<random name>* → System32\drivers\<random name>.sys *HKLM\SYSTEM\CurrentControlSet\Services\<random name>* → <random name>.sys keyboard driver *HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run* → <random name>.exe
Aphex	*HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run* → <user defined name>.exe

Figure 4. Experimental Results for GhostBuster Hidden ASEP Hook Detection.

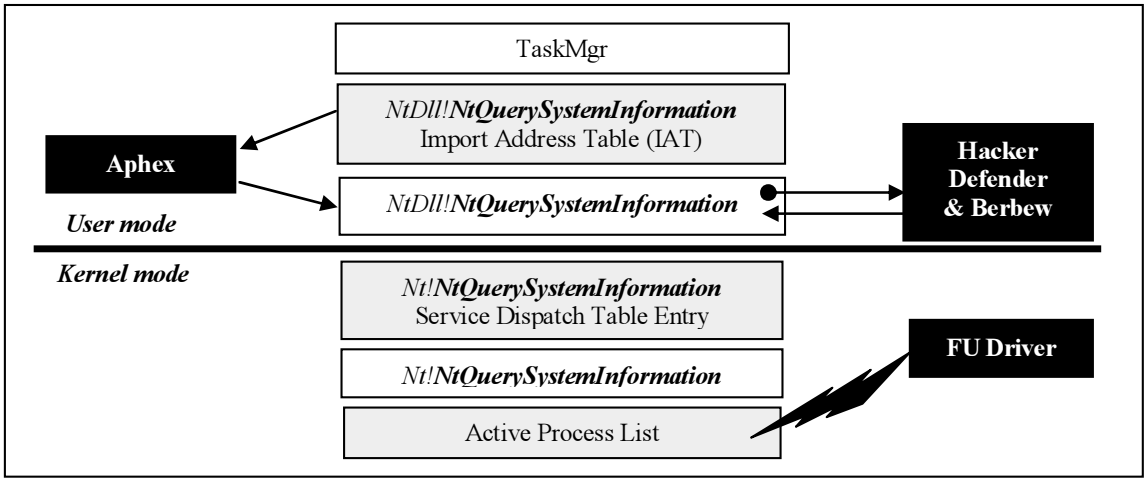

Figure 5. How Ghostware Programs Hide Processes.

Ghostware	Hidden Processes/Modules Detected
Aphex	By default, any process with a "~~"-prefixed name (which is configurable)
Hacker Defender 1.0	hxdef100.exe, and any other processes with names matching the patterns specified in hxdef100.ini
Berbew [ZB]	<random name>.exe
FU [ZFU]	Any process hidden by the *"fu –ph <PID>"* command
Vanquish	vanquish.dll (hidden inside many processes)

Figure 6. Experimental Results for GhostBuster Hidden Processes/Modules Detection.