Fast User-Mode Rootkit Scanner for the Enterprise

Yi-Min Wang and Doug Beck – Microsoft Research, Redmond

ABSTRACT

User-mode resource hiding through API interception and filtering is a well-known technique used by malware programs to achieve stealth. Although it is not as powerful as kernel-mode techniques, it is more portable and reliable and, as a result, widely used. In this paper, we describe the design and implementation of a fast scanner that uses a cross-view diff approach to detect all user-mode hiding Trojans and rootkits. We also present detection results from a large-scale enterprise deployment to demonstrate the effectiveness of the tool.

Introduction

The term "rootkit" generally refers to the class of stealth malware programs that hide "resources" from the operating system resource-enumeration APIs. For example, a rootkit may be used to hide critical executable files from anti-virus scanners, to hide critical Windows Registry entries from experienced system administrators using programs such as RegEdit, and to hide critical processes from average users running Windows Task Manager. By making critical resources "invisible" to the APIs and the system utilities that make use of such APIs, rootkits have a much better chance of evading detection and maintaining full control of infected machines for an extended period of time.

The techniques used by rootkits to achieve stealth can be broadly divided into two categories. Rootkits in the first category intercept resource-enumeration APIs in user mode or kernel mode, and remove selected entries before the results are returned to the API caller. Rootkits in the second category perform Direct Kernel Object Manipulation (DKOM) to remove selected resource entries from a cached list (such as the ActiveProcessList on Windows) that is designed specifically for answering resource queries and not critical to the actual functions of the resources or to the functioning of the operating system.

Cross-View Diff-Based Rootkit Detection

The traditional signature-based anti-virus approach cannot effectively deal with rootkit infections in the enterprise for three reasons. First, viruses usually have well-defined bundles and scope of impact, which can be analyzed in a lab to generate fixed signatures. In contrast, rootkits are merely "resource hiders" that can be used to hide any hacker tools, keyloggers, spyware programs, FTP servers, etc., so each rootkit infection can potentially involve a customized bundle with a different scope of impact.

Second, sophisticated hackers who attack large enterprise are less likely to use common tools or malware programs for which commercial anti-virus scanners already have signatures. Finally, while the major strength of anti-virus software is to detect and *automatically remove* known-bad malware programs without user intervention, corporate security organizations in large enterprises often need to investigate every rootkit infection case to assess potential damages and prevent future infections; automatic removal is often not desirable.

We previously proposed a non-signature, diff-based approach to rootkit detection, called *Ghost-Buster* [WVR+04]. The basic idea is to get "the lie" from inside the box, get "the truth" from outside the box, and take a diff to detect hidden resources. Specifically, we get "the lie" by enumerating files and Registry entries through infected APIs inside the operating system. Then we boot into a clean CD and scan the files and Registry on the infected drive as a data drive. Since the rootkit is not running, we obtain "the truth" that includes the resources that the rootkit was trying to hide so that the diff between "the lie" and "the truth" will reveal precisely those hidden entries. Such a diff-based approach essentially turns the hiding behavior into its own detection mechanism and turns one of the most difficult anti-malware problems into one of the easiest problems to solve.

Fast User-Mode Rootkit Scanner for the Enterprise

Although this CD-boot-based solution can cover a broad range of rootkits, no matter how they are operating in user mode or kernel mode, it is inconvenient, requires user cooperation, and is difficult to deploy on an enterprise scale as a scanner. Since the statistics from a major Product Support Service (PSS) organization indicates that *user-mode rootkits* account for over 90% of the reported enterprise rootkit cases, it is desirable to have a scalable rootkit scanner that can be deployed in the enterprise to detect all user-mode rootkits, which intercept and filter resource API calls in the address space of each user-mode process [YH03, YN04].

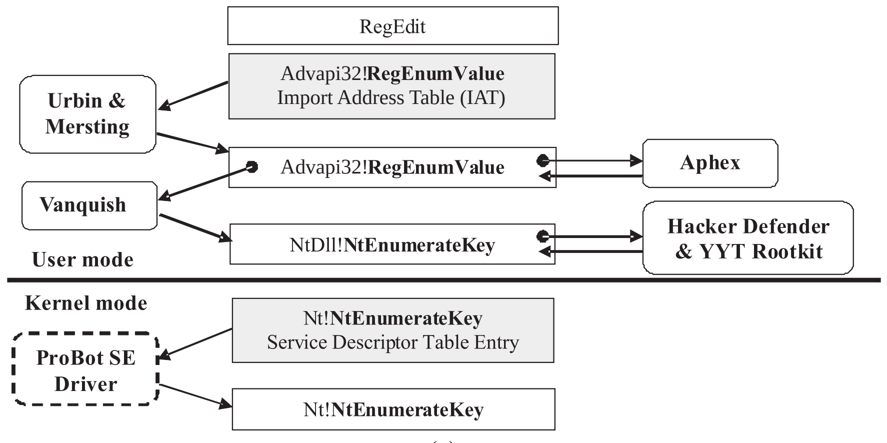
We have developed such a rootkit scanner for Windows platforms. It is based on the key observation that, when considering only user-mode rootkits, "the truth" can be obtained from *the lowest level of user mode* by properly preparing the call stack parameters and using a few lines of assembly code to directly invoke the transition into the kernel, without going through the regular user-mode Win32 API code. In our current implementation, we use the difference to detect hidden processes and to detect hidden hooks to Auto-Start Extensibility Points (ASEPs), which are those Registry locations most frequently attacked by spyware, Trojans, and rootkits based on an extensive study [WRV+04, WBV+05]. If any hidden processes or ASEP hooks are detected, we then look for potentially hidden files associated with those hidden entries. This allows us to detect user-mode rootkit infections in a few seconds, without requiring a reboot or any user participation.

Implementation

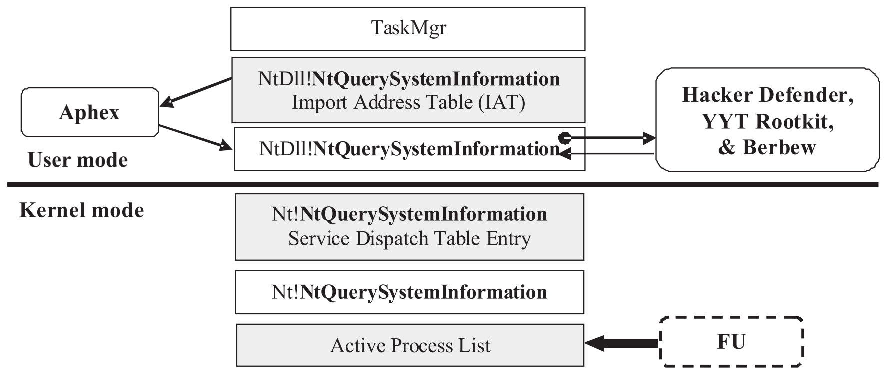
Figure 1 (a) and (b) illustrates how several real-world Trojans and rootkits hook into the Registry and process enumeration API calling chains, respectively, to hide their resources. Urbin and Mersting are Trojan DLLs that make modifications at the highest level by altering the per-process Import Address Table (IAT) entries of the Registry enumeration APIs to point to their Trojan functions (an IAT contains pointers to functions exported by loaded DLLs [HB05]). In contrast, Vanquish directly modifies the loaded, in-memory API code to interject its code. 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 RegEnumValue API code inside Apvapi32.dll (denoted by Advapi32!RegEnumValue), while Hacker Defender modifies the lower-level NtEnumerateKey API exported by NtDll.dll. YYT rootkit operates very similarly to Hacker Defender. ProBot SE in Figure 1(a) and FU in Figure 1(b) are kernel-mode stealth programs that cannot be detected by the scanner described in this paper.

Our tool performs the following steps to obtain "the truth" from underneath all the user-mode Trojans and rootkits shown in Figure 1.



(a)


TaskMgr
NtDll!NtQuerySystemInformation
Import Address Table (IAT)
Aphex
NtDll!NtQuerySystemInformation
Hacker Defender, YYT Rootkit, & Berbew
User mode
Kernel mode
Nt!NtQuerySystemInformation
Service Dispatch Table Entry
Nt!NtQuerySystemInformation
Active Process List
FU

(b)

Figure 1: Trojans and rootkits that (a) hide Registry entries and (b) hide processes.

1. Set up the user-mode stack with the parameters required by the operating system; this is easily achieved by creating a function with a signature that exactly matches the desired Native API, such as *NtDll!NtEnumerateKey()* and *NtDll!NtQuerySystemInformation()*;
2. Populate the EAX register with the index that indicates to the operating system which system function you wish to call;
3. Populate the EDX register with a pointer to the user mode's stack parameters;
4. Execute an "int 2e" instruction to signal a kernel-mode transition;
5. Return to the caller.

The previous steps basically describe what the code in *NtDll.dll* does when calling into the operating system. Given that the steps require direct manipulation of X86 registers, a portion of the code is written in assembly – this is easily obtained by disassembling *NtDll.dll* and searching for the desired function by name. Below is an example of the *NtQuerySystemInformation* call for retrieving "the truth" of the list of processes:

```
__declspec(naked)
NTSTATUS
NTAPI
MyNtQuerySystemInformation(
    SYSTEM_INFORMATION_CLASS
                     SystemInformationClass,
    PVOID SystemInformation,
    ULONG SystemInformationLength,
    PULONG ReturnLength)
{
    __asm
    {
        mov eax, 0xAD
        lea edx, [esp+0x4]
        int 2eh
        ret 10h
    }
}
```

Several important points need to be made regarding the code above. First, notice the `__declspec (naked)` compiler directive. This prevents the compiler from generating prolog code for the function in order to ensure that the user-mode stack is in the right form at the time of a kernel-mode transition. Second, the function is labeled as `NTAPI` to ensure that the right C calling convention is used (in this case `__stdcall`). Third, the value that is moved into the EAX register is the index into a kernel-mode function dispatch table that tells the operating system which function to call: this value is unique to the function and varies from version to version of the operating system. Fourth, the "int 2eh" instruction sends a signal to the operating system to tell it to initiate a kernel-mode transition. Finally, the parameters exactly match the original query API because the operating system will pass them directly to the kernel-mode API.

Rootkits & Trojans	**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
YYT Rootkit	HKLM\SYSTEM\CurrentControlSet\Services\NPF → npf.sys HKLM\SYSTEM\CurrentControlSet\Services\TSSERVER → comine.exe HKLM\SYSTEM\CurrentControlSet\Services\Udfs
Hacker Defender 1.0	HKLM\SYSTEM\CurrentControlSet\Services\HackerDefender100 → hxdef100.exe HKLM\SYSTEM\CurrentControlSet\Services\HackerDefenderDrv100 → hxdefdrv.sys
Aphex	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run → <user defined name>.exe
Vanquish	HKLM\SYSTEM\CurrentControlSet\Services\Vanquish → vanquish.exe

(a)

Rootkits & Trojans	**Hidden Processes Detected**
Berbew	<random name>.exe
YYT Rootkit	comine.exe
Hacker Defender 1.0	hxdef100.exe, and any other processes with names matching the patterns specified in hxdef100.ini
Aphex	By default, any process with a "~~"-prefixed name (which is configurable)

(b)

Figure 2: Hidden resources detected: (a) hidden Registry ASEP hooks; (b) hidden processes.

Once the tool obtains "the truth" and then uses a regular Win32 API call to obtain "the lie", it compares the two scans and declares as hidden resources those that appear only in "the truth". It is possible to see false positives due to the creation or deletion (depending on the order the scans are performed) of resources in the time window between the scans. In practice this is usually not an issue. Also, this is easily mitigated by performing successive scans and taking the intersection of the missing resource sets to create the final result set.

Experimental Results

Results from Lab Tests

We have tested the tool in our lab against the seven user-mode rootkits and Trojans shown in Figure 1. Although the implementations of these malware programs are quite different, our tool was able to efficiently and effectively detect all of them in a uniform way. Figure 2(a) and (b) show the detected hidden ASEP hooks and hidden processes, respectively, for each malware. The Urbin, Mersting, Berbew, and YYT Rootkit samples were captured from the wild, while the Hacker Defender, Aphex, and Vanquish samples were downloaded from the Web. The "AppInit_DLLs" ASEP allows auto-loading of one or more DLLs into every Windows-based application that is running in the current log-on session [AID]; the "Services" ASEP allows installations of always-running services and drivers; the "Run" ASEP allows additional processes to be auto-started near log-in time.

Results from Actual Deployment

We have deployed the tool on over 200,000 desktop and server machines. The executable file is copied from a central machine to each target machine at scan time; the scan results are reported back to the central machine, and the executable file is removed. Figure 3 gives some examples of detected infections. They can be broadly classified into two categories: hiding Trojans and full-fledged rootkits. Figures 3 (a), (b), and (c) show three types of hiding Trojans: they were most likely installed by malicious Web servers that exploit visiting browsers' vulnerabilities [WBJ+05]. The Trojans in Figure 3 (a) hide their hooks to the "AppInit_DLLs" ASEP, while the Trojans in Figure 3 (b) and (c) hide their randomly-named processes.

Figure 3 (d), (e), (f) show three cases of infections with full-fledged rootkits. We make the following observations: first, it is common for rootkits to create malware programs that have the same or similar filenames as some system programs but reside in a different directory; for example, one of the lsass.exe processes in Figure 3 (f) was instantiated from the lsass.exe malware program located in the "drivers" directory. Second, full-fledged rootkits tend to hook the "Services" ASEP to install services and drivers and they tend to hide multiple ASEP hooks and processes.

Rootkit Investigation Tool

Once a rootkit-infected machine is identified in an enterprise, it is important to investigate the hacker's intention and the damage that has been done, without disturbing the malware because some are designed to erase all traces of themselves once they realize that they have been detected. It is therefore highly desirable to have a tool that allows such non-intrusive investigations.

We have observed that configurable rootkits, such as the most popular Windows rootkit "Hacker Defender," typically support the notion of "root processes". A root process is not infected by the rootkit and can see "the truth". It is provided for the convenience of the rootkit users. For example, it can be very awkward for hackers if the resources are hidden from their tools

```
Hidden ASEP Hook: HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows\
Value: "AppInit_DLLs" Data: "C:\WINDOWS\system32\log.dll"
Hidden ASEP Hook: HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows\
Value: "AppInit_DLLs" Data: "C:\WINDOWS\System32\winpgfd.dll"
Hidden ASEP Hook: HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows\
Value: "AppInit_DLLs" Data: "C:\WINDOWS\System32\winpgfd.dll"
```

(a)

```
Hidden Procs: \Device\HarddiskVolume1\WINDOWS\system32\Hbkqjd32.exe
Hidden Procs: \Device\HarddiskVolume2\WINDOWS\system32\Mdnpdf32.exe
Hidden Procs: \Device\HarddiskVolume1\WINDOWS\System32\Ppnlan32.exe
Hidden Procs: \Device\HarddiskVolume1\WINDOWS\system32\Ljoocp32.exe
Hidden Procs: \Device\HarddiskVolume1\WINDOWS\system32\Gnaeplam.exe
```

(b)

```
Hidden Procs: \Device\HarddiskVolume1\WINDOWS\System32\elitesig32.exe
Hidden Procs: \Device\HarddiskVolume1\WINDOWS\system32\elitegct32.exe
Hidden Procs: \Device\HarddiskVolume2\WINDOWS\system32\eliteyzx32.exe
Hidden Procs: \Device\HarddiskVolume1\WINNT\system32\eliteohl32.exe
Hidden Procs: \Device\HarddiskVolume2\WINNT\system32\elitemfu32.exe
Hidden Procs: \Device\HarddiskVolume2\WINNT\system32\eliteaee32.exe
```

(c)

Figures 3a-3c: Actual infection cases.

as well; when a rootkit is used to hide a spyware browser add-on executable file, it gets really tricky if the file is also hidden from the browser process that is supposed to load it. With "root process" support, the hacker tools and the browser can be declared as root processes to ensure smooth operation, while the resources are still hidden from all the other processes, system utilities, and anti-malware scanners. In the case of Hacker Defender, "root processes" are listed in a configuration file via filenames specified with regular expressions.

We have developed a technique to take advantage of the support for root processes to allow non-intrusive investigations. It works as follows [WB05]. First, the fast user-mode rootkit scanner is used to identify hidden processes. Very often, one or more of the hidden processes are also root processes (because these are the most critical processes for the infection). Suppose process "foo.exe" is a detected hidden, root process. In the second step, we make a copy of the command-window program "cmd.exe", rename it to "foo.exe", and launch it through "start foo.exe". Now we have a command window that is a root process that can see all previously hidden resources. In particular, the "dir" command from this window can see all hidden files, including the Hacker Defender configuration file which reveals critical information about the hacker's intention.

```
Hidden processes detected: 2
   PID  Name
   2168 \Device\HarddiskVolume1\WINDOWS\system32\scrss.exe
   2460 \Device\HarddiskVolume1\WINDOWS\system32\taskmgnr.exe
Hidden service keys detected: 2
Key:  HKEY_LOCAL_MACHINESYSTEM\CurrentControlSet\Services\scrss
      ImagePath: C:\WINDOWS\system32\scrss.exe
Key:  HKEY_LOCAL_MACHINESYSTEM\CurrentControlSet\Services\tskmgr
      ImagePath: C:\WINDOWS\system32\taskmgnr.exe
```

(d)

```
Hidden processes detected: 11
      PID   Name
      436   \Device\HarddiskVolume2\WINDOWS\system32\smss.exe
      564   \Device\HarddiskVolume2\WINDOWS\system32\lsass.exe
      1396  \Device\HarddiskVolume2\WINDOWS\system32\svchosts.exe
      1428  \Device\HarddiskVolume2\WINDOWS\system32\csschk.exe
      1744  \Device\HarddiskVolume2\WINDOWS\system32\DNTUS26.EXE
      1976  \Device\HarddiskVolume2\WINDOWS\system32\ShellExt\_tmp\IPSSvc.exe
      248   \Device\HarddiskVolume2\ SYSTEM~1\_system\lsass.exe
      308   \Device\HarddiskVolume2\WINDOWS\smss.exe
      364   \Device\HarddiskVolume2\ SYSTEM~1\_system\system\ioFTPD.exe
      2052  \Device\HarddiskVolume2\ SYSTEM~1\_system\lsass.exe
      2164  \Device\HarddiskVolume2\ SYSTEM~1\_system\bot\eggdrop.exe
Hidden service keys detected: 2
Key:  HKEY_LOCAL_MACHINESYSTEM\CurrentControlSet\Services\IPSdrv
      ImagePath: \??\c:\windows\system32\shellext\_tmp\IPSdrv.sys
Key:  HKEY_LOCAL_MACHINESYSTEM\CurrentControlSet\Services\NetSecc
      ImagePath: c:\windows\system32\shellext\_tmp\ipssvc.exe naslib.dll
```

(e)

```
Hidden processes detected: 4
      PID   Name
      556   \Device\HarddiskVolume2\WINDOWS\system32\services.exe
      568   \Device\HarddiskVolume2\WINDOWS\system32\lsass.exe
      1768  \Device\HarddiskVolume2\WINDOWS\java\lspool.exe
      484   \Device\HarddiskVolume2\WINDOWS\system32\drivers\lsass.exe
Hidden service keys detected: 2
Key:  HKEY_LOCAL_MACHINESYSTEM\CurrentControlSet\Services\Lspool
              ImagePath: c:\windows\java\lspool.exe
Key:  HKEY_LOCAL_MACHINESYSTEM\CurrentControlSet\Services\r_server
              ImagePath: c:\windows\system32\drivers\lsass.exe /service
```

(f)

Figures 3d-3f: Actual infection cases.

In the third step, we launch Task Manager, RegEdit, anti-virus scanner, anti-spyware scanner, etc. from this command window. Since this root process is not infected it cannot infect its child processes; as a result, all these utilities are now running as root processes that can see all previously hidden resources. After the investigation, malware processes, Registry entries, and files can all be terminated/deleted through these utilities.

Related Work

There are two different approaches to rootkit 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 rootkit 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, security 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]. Our user-mode rootkit detector belongs to the second category.

There is a subtle but important difference between the "cross-view diff" used in our proposal 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 infected path 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 hiding malware and usually has zero or very few false positives because legitimate programs rarely hide.

Ideally, "the truth" should be obtained from "outside the box" to eliminate the possibility of any malware intervention. The WinPE-based GhostBuster tool [WBV+05] took such an approach to obtain "the truth" of the file system and Registry. The PCI-add-in card described in the Copilot paper [PFM+04] or the Myrinet NIC described in the Bookdoors paper [BNG+04] can be used to obtain "the truth" of the process list through Direct Memory Access (DMA) without the knowledge or intervention of the potentially infected OS. Instead of targeting comprehensiveness, our user-mode rootkit detector targets efficiency, scalability, and ease of use with good coverage.

In response to the increasing popularity of stealth techniques among Windows malware, several rootkit detection tools have been released in recent months, including RootkitRevealer from Sysinternals, Blacklight Rootkit Eliminator from F-Secure, and IceSword from Xfocus.net. RootkitRevealer uses the same cross-view diff technique described in our previous paper on Inside-the-box GhostBuster [WBV+05]: it performs high- and low-level scans and reports the differences between these scans. For the file scans, it performs two low-level scans by reading the NTFS Master File Table and the NTFS on-disk directory index structures. For the low-level Registry scan, the tool reads the raw Registry hive files. A file discrepancy is reported if a file does not appear in all three scans. A Registry discrepancy is reported if the data, length, or type of a Registry value differs or if an entry is missing.

Blacklight is designed to detect hidden processes and files via a kernel-mode driver. In addition to running as a standalone process, it incorporates detection evasion technology: it may perform its scans through the Windows Explorer process. IceSword uses kernel-mode technology to detect hidden processes, hidden ports, hidden services, hidden auto-start programs, hidden files, hidden Browser Helper Objects, and hidden Registry entries. It monitors process creation and deletion and is able to disable filter drivers that prevent file creation and deletion. In order to achieve its functionality, the program loads a kernel-mode driver and then disables kernel-level debugging. Unlike the other rootkit tools, it has anti-rootkit attack technology to keep it from being disabled by malicious software. It achieves this by trapping keyboard strokes and requiring that the user hit Ctrl+Alt+D in order to put the program into a mode where it may be shutdown.

Similar rootkit 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 rootkits 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 discussed cross-view diff-based hidden resource detection for Linux/UNIX platforms in our previous paper [WBV+05].

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 cross-view diff approach.

Summary

User-mode rootkits are popular because they are more portable and reliable than kernel-mode rootkits. We have shown that there is a quick and easy way to detect all user-mode rootkits: by performing a cross-view diff between a high-level infected scan above rootkit interception and a low-level clean scan below the interception, our tool can precisely detect hidden Registry entries and processes within a few seconds. The simplicity and efficiency make it an attractive tool for scalable deployment in large enterprises to provide protection against new or customized rootkits that escape common signature-based anti-malware scanning. Detection results from actual deployment suggested that hiding Trojans, most likely installed through malicious Web sites, may be an even more serious concern than rootkits in terms of prevalence.

Author Information

Yi-Min Wang manages the Cybersecurity and Systems Management Research Group and leads the Strider project at Microsoft Research, Redmond. He received his Ph.D. in Electrical and Computer Engineering from University of Illinois at Urbana-Champaign in 1993, worked at AT&T Bell Labs from 1993 to 1997, and joined Microsoft in 1998. His research interests include security, systems management, dependability, home networking, and distributed systems.

Doug Beck is a senior developer at Microsoft where he has worked for the past six years. During his career at Microsoft Doug has focused on developing Systems Management software. He received a Ph.D. in Theoretical Physical Chemistry from the University of Washington in 1996 where he developed simulation software for solving partial differential equations as part of his research. Doug is currently working at Microsoft Research and can be reached at Doug.Beck@ microsoft.com .

References

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

[AS] Microsoft Windows Anti-Spyware, http://www.microsoft.com/spyware .

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

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

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

[HB05] Hoglund, G. and J. Butler, *Rootkits: Subverting The Windows Kernel*, Addison-Wesley, 2005.

[J01] Jones, K., "Loadable kernel modules," *;login:*, http://www.usenix.org/publications/login/2001-11/pdfs/jones2.pdf , Nov., 2001.

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

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

[MSRT] Windows Malicious Software Removal Tool, http://www.microsoft.com/security/malwareremove/ .

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

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

[WB05] Wang, Yi-Min and Doug Beck, "How to 'Root' a Rootkit That Supports Root Processes Using Strider GhostBuster Enterprise Scanner," *Microsoft Research Technical Report MSR-TR-2005-21*, February 11, 2005.

[WBJ+05] Wang, Yi-Min, Doug Beck, Xuxian Jiang, and Roussi Roussev, "Automated Web Patrol with Strider HoneyMonkeys: Finding Web Sites That Exploit Browser Vulnerabilities," *Microsoft Research Technical Report MSR-TR-2005-72*, August, 2005.

[WBV+05] Wang, Yi-Min, Doug Beck, Binh Vo, Roussi Roussev, and Chad Verbowski, "Detecting Stealth Software with Strider GhostBuster," *Proc. DSN*, June, 2005.

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

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

[WVR+04] Wang, Yi-Min, 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.

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

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

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

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

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

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

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

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

[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.

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

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