

# Windows rootkits of 2005, part three

James Butler, Sherri Sparks 2006-01-05

The third and final article in this series explores five different rootkit detection techniques used to discover Windows rootkit deployments. Additionally, nine different tools designed for administrators are discussed.

## 1. Introduction

Rootkits have become very sophisticated over the past few years, and in 2005 we have seen a surge in rootkit deployments in spyware, worms, botnets, and even music CDs. Although once a computer system has been subverted by a rootkit it is extremely difficult to detect or eradicate the rootkit, there are still some different methodologies that detect the rootkit that have worked to varying degrees. [Part one](#) looked at what Windows rootkits are and what makes them so dangerous. [Part two](#) examined the latest cutting-edge rootkit technologies and how they achieve stealth.

Now in part three, we explore five such detection techniques and, where possible, provide information about different rootkit detection tools.

## 2. Signature based detection

Signature based detection methods have been in use by antiviral products for years. The concept is simple. System files are scanned for a sequence of bytes that comprise a "fingerprint" that is unique to a particular rootkit. If the signature is found in a file on the user's system, it signals an infection. As signature scanning has traditionally been applied to the filesystem, its usefulness for rootkit detection is limited unless it is combined with some more advanced detection techniques. This is due to the rootkit's natural propensity to hide files using execution path hooking techniques.

Despite their antiquity, signature based detections are worth mentioning because they may be applied with success to scanning system memory in addition to filesystem scanning. Ironically, most public kernel rootkits are susceptible to signature scans of kernel memory. As kernel drivers, they typically reside in non-paged memory and few, if any, make an effort towards any kind of polymorphic code obfuscation. Thus, a scan of kernel memory should trivially identify most public kernel rootkits regardless of their underlying "bag of tricks" (DKOM, SSDT, IDT hooking and the like). The key words in that last sentence, however, are "public rootkits" because signature based detection is, by definition, useless against malware for which a known signature does not exist. Finally, signature based detection methods are useless against Virtual Memory Manager (VMM) hooking rootkits like Shadow Walker which are capable of controlling the memory reads of a scanner application. [\[ref 1\]](#)

## 3. Heuristic / Behavioral detection

Where signature based detections fall short, heuristic detections take over. Their primary advantage lies in their ability to identify new, previously unidentified rootkits. They work by recognizing deviations in "normal" system patterns or behaviors. Various heuristics have been proposed for identifying rootkits based upon execution path hooking. In this section we examine two such tools: VICE and Patchfinder.

### 3.1 VICE

VICE is a [freeware tool](#) written to detect hooks [ref 2]. It is a standalone program that installs a device driver to analyze both user mode applications and the operating system kernel. In the kernel, VICE checks the SSDT for function pointers that do not resolve to `ntoskrnl.exe`. Also, you can add devices to the file "driver.ini," and VICE will check the IRP major function table of the corresponding driver. If a function pointer in the IRP major function table of a driver does not consist of an address within the driver, then the IRP has been hooked by an outside driver or piece of kernel code. In user mode, VICE checks the address space of every application looking for IAT hooks in every DLL that the application uses. Inline function hooks are detected in DLL functions imported by applications and in the SSDT functions themselves. VICE will resolve what function is being hooked and the address of the hooking function. When possible, VICE will also display the full path on the filesystem of the DLL or device driver doing the hooking so that a System Administrator can remove the malicious software. Today, VICE will detect most publicly known Windows rootkits and any stealth related technology that uses hooking technologies. To run VICE, the host machine must have the Microsoft .NET Framework installed, which is free for download.

The current version of VICE has been targeted and subverted by at least one public rootkit. [ref 3] Rootkits have leveraged the fact that VICE always executes with a specific process name. When the rootkit detects the VICE process, it does not hook so VICE has nothing to detect. Another attack has targeted VICE's communication channel between the user mode portion and the device driver. However, VICE's biggest weakness may be the large number of false positives it returns. VICE was designed to detect hooks, but there are legitimate uses of hooks in the operating system. Microsoft themselves offer "hooks" in the form of hot patching and DLL forwarding. It is difficult to determine what is a malicious hook versus a benign, legitimate hook.

### 3.2 Patchfinder

Where VICE identifies rootkits based on static analysis of code and data structures, Joanna Rutkowska proposes a detection method based upon runtime execution path profiling [ref 4]. It is implemented in the proof of concept tool, [Patchfinder](#), and is available on her website, [invisiblethings.org](#). Patchfinder is built upon the observation that a rootkit must add code to a given execution path (i.e. to filter the results returned by a hooked service, for example). Thus, if one is able to count the number of instructions executed by some system services commonly hooked by rootkits, the instruction count should be greater in an infected system than the count observed by executing the same services on a clean system. Rutkowska uses the "single step" feature of the x86 processor to perform this instruction counting. When code is run in "single step" mode, the processor halts execution and calls a special Interrupt Service Routine (ISR) after each instruction is executed. The instruction

count is updated in this routine. Unfortunately, in a system as complex as Windows, there may be multiple execution paths which result in non-deterministic behavior. In other words, multiple executions of the tested service may return different counts under varying system conditions. Rutkowska deals with this problem statistically in Pathfinder by constructing a histogram and empirically demonstrating that the primary peak on the histogram remains constant regardless of system load, while the insertion of additional instructions by a rootkit causes a shift in the peak. Nevertheless, there remain cases where false positives can occur and the technique is vulnerable to rootkits which realize that they are being traced.

## 4. Cross view based detection

Cross view based detection techniques are relatively new and show a lot of promise. These techniques assume that the operating system has been subverted, but this method leverages the fact that there is usually more than one way to ask the same question. In a cross view based detection method, the detection software calls the common APIs to enumerate key elements within the computer system such as the list of files, processes, or Registry keys. However, to be successful, the detection software must also have an algorithm to generate a similar data set that does not rely upon the common APIs. Any difference in these two data sets reveals something hidden because it did not exist in the data set generated using the common APIs. Cross view based detection relies upon the fact that API hooking or manipulation of kernel data structures will taint the data returned by the operating system APIs, but the low level methods used to glean the same information will be designed in such a manner as not to be subverted by hooks or DKOM tricks.

### 4.1 Rootkit Revealer

[Rootkit Revealer](#) is available from Sysinternals. [ref 5] Sysinternals provides many free utilities for System Administrators and developers alike. Rootkit Revealer targets a subset of rootkits called persistent rootkits. Persistent rootkits are those that exist between reboots. In other words, a memory resident rootkit would not be persistent. In order to be persistent between reboots, Rootkit Revealer assumes that the rootkit will have to exist on the filesystem and in the Windows Registry. Rootkit Revealer uses a cross view based approach to detect these persistent rootkits. Rootkit Revealer calls the highest level APIs in order to enumerate the files on disk and the Registry keys. To compare against this, Rootkit Revealer parses the raw filesystem structure on disk and the bare files that comprise the Registry hives.

Like most rootkit detection tools, Rootkit Revealer can be subverted by a rootkit that blocks or diverts access to disk volumes or the Registry files. Rootkit Revealer also suffers from false positives if files or Registry keys are created, deleted, or otherwise altered between the high level and the low level enumeration. This is true for all cross view based techniques because of the nature of the algorithm.

### 4.2 Klister

[Klister](#) is another proof of concept rootkit detection tool developed by Joanna Rutkowska. [ref 4] Klister demonstrates a method useful for detecting rootkits that use DKOM

techniques to hide processes, like FU. FU hides processes by unlinking EPROCESS objects from the PsActiveProcessList (note: See [part one](#) for specific details). [ref 6] Intuitively, it seems that if one removes a process from the OS's process list it would cease to run (in other words cease to be scheduled for time on the CPU by the OS scheduler). Interestingly, this is not actually the case. The OS dispatcher maintains separate queues from which it schedules processes. Klister exploits this redundancy in OS data structures to detect processes hidden by unlinking the entries in the PsActiveProcessList. By comparing the active process list with the dispatch queues, it is possible to identify discrepancies. A process which appears in the dispatch queue but not in the active process list is assumed to be a hidden rootkit process. Although Rutkowska only applies this technique to process detection, comparisons between redundant kernel data structures can be a valid form of detection for other DKOM attacks as well.

### 4.3 Blacklight

Blacklight by F-Secure is one of the first commercial products aimed specifically at rootkit detection. [ref 7] Although Blacklight technology is being rolled into other products by F-Secure, Blacklight is available as a [free download](#) for now. Blacklight targets processes and files hidden by rootkits. It uses a cross view based method, enumerating all the processes with higher level APIs and comparing that list to a list generated using a much lower level method. Blacklight uses a similar method for detecting hidden files.

Over the last year, the algorithm Blacklight uses for the lower level enumeration has changed several times. Each time, Blacklight attempts to identify a critical piece of data that would cause the system to fail if a rootkit altered or subverted it. Blacklight and other cross view based detection methods have had varying degrees of success identifying those key elements that cannot be altered. Once the rootkit authors discover the algorithm being used, a carefully designed rootkit can usually alter the data without any negative consequences.

### 4.4 Strider GhostBuster

Similarly to Rootkit Revealer, Microsoft's Strider GhostBuster takes a cross-view-diff based approach to detect hidden files, registry entries, processes, and loaded modules. [ref 8] That is, it compares high level API queries with low level queries obtained by manually parsing the underlying operating / filesystem data structures (such as the NTFS Master File Table and the Windows Registry hive). Differences between the queries are taken to indicate the presence of "ghostware," a term coined by the MS research team to include rootkits, trojans, keyloggers, and other commercial adware / spyware that has a vested interest in concealing their presence on a system.

GhostBuster provides both inside-the-box and outside-the-box system scans. By inside-the-box, we mean that the scan can be performed on a running system. This is basically a scan very similar to Rootkit Revealer scan with added detection for hidden processes. To perform this scan, an inside-the-box scan is first performed on a live suspect system. Then, the system is booted to a clean operating system (WinPE is used) where a low level scan is then performed. The offline nature of the scan reduces the possibility of interference or attack of

the scanner application by a rootkit. The tradeoff, however, is in convenience since it is clearly impractical to reboot a server system for a routine malware scan. Nevertheless, it may be useful for providing forensic evidence if the system is strongly suspected or already known to be compromised.

Although the cross view approach seems to be state-of-the-art in current rootkit detection methodologies, it is worth mentioning that the approach is not invulnerable to existing rootkit attack methodologies. Indeed, its success depends in large part upon implementation, specifically the method which is used to obtain the "low level" view of the system. For example, in order to parse the NTFS disk layout, a detector must first obtain a handle to the volume and then subsequently read its disk sectors. If the detector uses an operating system API to perform these tasks, it may as well go back to the drawing board. In other words, a detector must assume that the API interface is compromised (API hooking is a tried and true rootkit technique for cheating detection). Thus, the strongest implementation of a cross view approach should only rely upon direct communication with the disk controller. [ref 9]

## 5. Integrity based detection

Integrity based detection provides an alternative to both signatures and heuristics. It relies upon comparing a current snapshot of the filesystem or memory with a known, trusted baseline. Differences between the current and baseline snapshots are taken as evidence of malicious activity. Unfortunately, however, the integrity checker is usually not capable of pinpointing the origin of the activity that has caused the changes.

### 5.1 Tripwire

Tripwire is a disk based integrity checker [ref 10]. It creates a trusted database of unique CRC hash values for each of the system files on a user's hard disk. When the user initiates a scan for malicious activity, it recalculates the CRC's for all files and compares them to the original CRC's in the database. It is based upon observation that system files should not change (except perhaps in the rare system update or service pack), thus a mismatch indicates compromise. Tripwire was very effective against early rootkits that simply replaced system files on the disk with trojanized versions. Unfortunately, rootkits adapted by moving their modifications from disk to memory. This renders Tripwire virtually useless for modern rootkit detection. Nevertheless, where disk based integrity checking fails, memory based integrity checking may succeed.

### 5.2 System Virginty Verifier

[System Virginty Verifier](#) by Joanna Rutkowska is yet another prototype rootkit detection tool. [ref 11] It combines heuristics similar to VICE with memory based integrity checking technology. Like VICE, it checks the integrity of operating system data structures (IAT / EAT / SSDT / IRP tables). It also incorporates some more advanced heuristics to help deal with the false positive problem resulting from benign hooking by legitimate applications such as antivirus scanners and personal firewalls. Memory integrity checking is provided by comparing the code sections for important system libraries and drivers on disk with their

corresponding loaded images in memory.

## 6. Hardware detection - Copilot

**Copilot** is unique in that it is a hardware based detection tool. It began as a project at the University of Maryland and has spawned an independent company. Currently, Copilot [ref 12] is in the form of a PCI card that is installed on the host being monitored for rootkit activity. The goal of the PCI card is to remain as independent of the potentially subverted operating system as possible. To do this, the PCI card has its own CPU and uses Direct Memory Access (DMA) to scan the physical memory of the computer looking for rootkit behavior such as hooks in the SSDT, alterations to kernel functions (using kernel integrity checks), and modifications to key data structures like those performed by a DKOM attack. The Copilot PCI board also has its own network interface to communicate in a secure fashion to an administrative component.

Because it is a hardware based solution, Copilot provides a high degree of assurance; however, this is not without a price. Any hardware based solution is going to be more costly to purchase and to maintain, but the added assurance is worth it for some. If coupled with a software monitoring component, Copilot would make an even more formidable adversary against the rootkit authors. Copilot could even verify the integrity of its own software running on the host.

## 7. Concluding part three

As we survey the existing rootkit detection techniques, it becomes apparent that all of the approaches have strengths and weaknesses. Every one of the described tools brings a piece of the puzzle to the table, but none of them yet provide a truly comprehensive solution to the problem. Many of the tools are still proof of concept and, ironically, some of them were written by the rootkit authors themselves. It is, however, encouraging to see anti-rootkit activity beginning in the commercial AV / Anti-malware arena (as we saw with Blacklight and Strider GhostBuster) as well as in the high assurance market (Copilot). In attempting to consider the "big picture," we can finally make a few observations based upon our firsthand observations of some of the battles that occur between rootkit authors and detectors. First, a number of the aforementioned tools show promise. Hybrid approaches which combine several of these techniques, however, will be stronger because they will maximize their strengths while minimizing the effects of weaknesses. Second, "lower is better" should be the mantra of the rootkit detector. Many detectors get themselves into trouble by relying upon operating systems API's which they should assume to be compromised. Finally, one must keep in mind that the rootkit still has the upper hand. Unlike a rootkit which rarely needs to concern itself with the stability of the victim's system, this concern is of paramount importance to the detector, especially commercial detectors with a paying client base. This often constrains the rootkit detection software by forcing it to avoid some useful but dangerous methods. Despite this, detectors are clearly "upping the ante." Now in early 2006, it will be interesting to see how the ongoing battle between rootkit detector and developer plays out over the coming year.

## 8. References

- [ref 1] Butler, James and Sparks, Sherri. "Shadow Walker: Raising The Bar For Windows Rootkit Detection", Phrack 63, July 2005.
- [ref 2] Butler, James, "VICE - Catch the hookers!" Black Hat, Las Vegas, July, 2004. [www.blackhat.com/presentations/bh-usa-04/bh-us-04-butler/bh-us-04-butler.pdf](http://www.blackhat.com/presentations/bh-usa-04/bh-us-04-butler/bh-us-04-butler.pdf)
- [ref 3] Holy Father, Hacker Defender. <http://www.hxdef.org/download/hxdef100r.zip>
- [ref 4] Rutkowska, Joanna. "Detecting Windows Server Compromises with Patchfinder 2", Jan 2004.
- [ref 5] Rootkit Revealer. <http://www.sysinternals.com/Files/RootkitRevealer.zip>
- [ref 6] Fuzen, FU Rootkit. <http://www.rootkit.com/project.php?id=12>
- [ref 7] BlackLight. <http://www.europe.f-secure.com/exclude/blacklight/>
- [ref 8] Y. Wang, D. Beck, R. Roussev, and C. Verbowski. "Detecting Stealth Software with Strider GhostBuster". Technical Report, Feb 2005.
- [ref 9] Rutkowska, Joanna. "Thoughts about Cross-View based Rootkit Detection", June 2005 [http://www.invisiblethings.org/papers/crossview\\_detection\\_thoughts.pdf](http://www.invisiblethings.org/papers/crossview_detection_thoughts.pdf) and [http://www.invisiblethings.org/papers/rootkits\\_detection\\_with\\_patchfinder2.pdf](http://www.invisiblethings.org/papers/rootkits_detection_with_patchfinder2.pdf)
- [ref 10] Tripwire, Inc. <http://www.tripwire.com>
- [ref 11] Rutkowska, Joanna. "System Virginty Verifier: Defining the Roadmap for Malware Detection on Windows Systems", Sept 2005. [http://www.invisiblethings.org/papers/hitb05\\_virginity\\_verifier.ppt](http://www.invisiblethings.org/papers/hitb05_virginity_verifier.ppt)
- [ref 12] N. Petroni, T. Fraser, J. Molina, and W. Arbaugh, "Copilot - a Coprocessor-based Kernel Runtime Integrity Monitor," in Proc. Usenix Security Symposium, Aug. 2004. On the Web at <http://www.komoku.com/pubs/USENIX-Copilot.pdf>.

## About the authors

James Butler is the CTO of [Komoku](http://www.komoku.com), which specializes in high assurance, host integrity monitoring and management. Before that, Mr. Butler was the Director of Engineering at HBGary, Inc. focusing on rootkits and other subversive technologies. He is the co-author and a teacher of "Aspects of Offensive Rootkit Technologies" and co-author of the newly released bestseller "[Rootkits: Subverting the Windows Kernel](#)."

Sherri Sparks is a PhD student at the University of Central Florida. Currently, her research interests include offensive / defensive malicious code technologies and related issues in digital forensic applications.

Copyright © 2006, SecurityFocus

[Privacy Statement](#)

Copyright 2006, SecurityFocus