Exploiting the Rootkit Paradox with Windows Memory Analysis

Jesse D. Kornblum
ManTech CFIA

Abstract

Rootkits are malicious programs that silently subvert an operating system to hide an intruder's activities. Although there are a number of tools designed to detect rootkits, these programs are competing with the rootkit for system resources and allowing the rootkit to actively evade detection. By taking a memory image of the system, a forensic examiner can conduct a more thorough search for rootkits and even without discovering one directly, infer the presence of one. This paper explores how an examiner can create such a memory image and use the inherent properties of rootkits to find them in those memory images.

Background

Rootkits are programs designed to hide processes, files, and activity from the operating system and legitimate users of a computer. Normally used only by intruders, they subvert the operating system and prevent it from functioning normally. The rootkit can modify, delete, or insert data into any of the operating system's processes, and as a result, have complete control over what the operating system does or does not see. Intruders use rootkits to hide malicious activity such as opening back doors for unauthorized access, recording keystrokes, or launching attacks against other systems.

By their very nature, rootkits are difficult to detect because they hide their own activities. For example, the Hacker Defender rootkit offers its owner the ability to hide itself, selected files, processes, and registry keys from the operating system and thus any user [HOLY]. Traditional malware detection techniques are not effective against rootkits as these programs cannot flag processes that they cannot see.

The Rootkit Paradox

All rootkits obey two basic principles:

1. They want to remain hidden.
2. They need to run.

Taken together, these rules create a paradox. In order to remain hidden, the rootkit needs to minimize its footprint on the system. However, in order to run, the operating system, a deterministic process, has to be able to find and execute the rootkit. If a deterministic process like the operating system can find the rootkit, then an examiner can find it as well.

Imaging Windows Memory

A memory image is similar to a disk image (also known as a bit-stream or "dd" image). It is a stream of bytes that represents an exact duplicate of an original data source. A memory image is just a bit-stream image of the contents of physical memory. Unlike a disk image, however, capturing a memory image is not a repeatable process. Because the contents of physical memory are volatile, changing during the imaging process and *as a result of the imaging process*, there is no way to preserve the exact contents of memory as they stood at any given moment. That being said, a memory image can capture a blurry snapshot of the computer's memory. A memory image is similar to a long exposure photograph from a camera slowly turning on a pivot point. Although most objects in the camera's view will appear in the photograph as they truly existed, other objects may not appear or may appear more than once. With the right analytical techniques, however, it is possible to retrieve a wealth of information from such a snapshot.

The Windows operating system allows programs to manipulate some kernel objects directly. This technique, called Direct Kernel Object Modification (DKOM) can be used to access the physical memory of the machine [CRAZ]. Forensics investigators have used this technique to record the contents of physical memory for later analysis [GARN04]. At the present time, an examiner can use DKOM or one of three other ways to acquire an image of physical memory. Each of them has specific advantages and drawbacks, described below:

<u>DKOM</u>

A Windows userland program called dd, adapted from the GNU program of the same name, can be used to image physical memory using DKOM [GARN04] . For example:

```
C:\> dd if=\\.\PhysicalMemory of=memory.img conv=noerror
```

Note that this version of dd does not support the sync option in the conv argument that is commonly used in forensic acquisition. This technique is also subject to the blurry snapshot problem described above. Running dd causes the program to be loaded into memory, changing the state of physical memory. Because this program requires either the System or Administrator privileges and requests access to the rarely used Physical Memory object, a rootkit could easily detect its use and deny access to the object.

<u>Virtual Machine Suspension</u>

When running the operating system on a virtual machine such as VMWare, the examiner can suspend the virtual machine to freeze the state of the target operating system. The contents of physical memory are normally written to a file so that they can be reloaded when the target operating system is resumed. An examiner could run the target operating system in a virtual machine, suspend it, and then use the saved contents of physical memory as an image file. The drawback to this method is that any rootkit can easily detect that it is running in a virtual machine environment and either refuse to run or delete itself.

Hibernation Mode

Microsoft Windows 2000, XP, and 2003 all support hibernation mode which saves the operating system state, to include the contents of physical memory, to a file on the disk before powering off the computer [MSFT]. When the computer is powered back on, the operating system state can be restored from the file. A clever examiner can use the file on the disk to recover a complete memory image of physical memory, although parsing the hibernation file format is left as an exercise to the reader. Conversely, a rootkit could intercept the instruction to begin the Hibernation process and hide itself before allowing hibernation to begin. But in accordance with Rootkit Principle #2, the rootkit would have to leave a hook of some kind so that it would be executed when the operating system resumed; a hook that an examiner could find. A rootkit that did not survive hibernation, although stealthy, would not survive for long.

Hardware Imagers

A number of commercial hardware products such as hardware debuggers exist today that allow an examiner to suspend the operating system and access physical memory directly. These devices will halt the operating system and allow the examiner to grab a pristine copy of the physical memory. Because they operate outside of the rootkit's view, the rootkit is powerless to interfere. On the other hand, these physical devices usually must be attached to the system before power is connected; they are difficult to use as part of an incident response procedure.

Why Smart Rootkits Won't Interfere with Memory Imaging

Previous researchers have noted that it would be easy for a rootkit to interfere with the Physical Memory kernel object and thus distort the examiner's picture of the rootkit [CARR]. However, such manipulations would clearly violate Rootkit Principle #1. There is no normal circumstance when legitimate access to the Physical Memory object is obstructed, and thus such an obstruction can *only* point to the existence of a rootkit. Although the examiner might not know the nature of the rootkit, its existence could be easily inferred.

Accordingly, it would be possible for a rootkit to put itself between the Physical Memory object and the memory imaging process. In that way the rootkit could filter the data and remove any references to the rootkit from the memory image. Unfortunately for the rootkit, imaging physical memory is an extremely fast operation. The author's informal testing has shown memory can be imaged at upwards of 50MB/sec. Sanitizing physical memory before giving it to an imaging process would require considerable CPU time as all traces of the rootkit would have to be eliminated. This could include device drivers, kernel objects, userland programs, kernel hooks, page fault handlers, and whatever other operating system components the rootkit has modified either directly or indirectly. Because physical memory is not organized in a fixed structure, the rootkit would have to dynamically find these structures and replace them with clean data.

The extra time required for this purging would be a dead giveaway to an examiner that something unusual was going on. Again, like denying access to the physical memory object, a slow memory imaging process would not reveal the nature of the rootkit, but would clearly indicate the presence of one.

Memory Image Analysis

Once a good memory image has been obtained, the examiner's search for the rootkit can begin in earnest. Unlike trying to find a rootkit on a live system, the rootkit is unable to take any action to hide itself in the memory image. As with most things, it is easier to find a rootkit when it is not moving. Techniques that would cause real problems for a live system analysis tool, such as marking every page of the rootkit's process as being paged out to the disk [SPAR], serve as conspicuous clues for a memory image based rootkit detector.

Analyzing a memory image allows an examiner to see the state of the operating system without the operating system as a filter. That is, the examiner can see the data without the operating system, or the rootkit, interpreting the data for them. Of course this puts the burden on the examiner to make sense of the operating system's state, but reverse engineering data structures are not new in the computer forensics field.

Although memory image analysis is in its infancy, the results from the Digital Forensic Research Workshop's 2005 Memory Analysis Challenge [BETZ][GARN05] showed great promise for recovering operating system structures from memory images. In fact, Chris Betz's analysis indicated the presence of the Hacker Defender rootkit using only a memory image. Obviously more work needs to be done to improve rootkit detection in memory images, but the capability is there.

Conclusion

Having a procedure to indicate the presence of a rootkit is good, but is not a final solution. If a legitimate user discovers that they are unable to image physical memory, they will be tipped off that *something* is wrong, but may not be able to discover exactly what has happened. Further, more research into memory image analysis in order to find anomalies in the operating system components is needed. The research conducted for the Digital Forensic Research Workshop's 2005 Memory Analysis Challenge was an excellent first step that should be continued as we strive to develop rootkit detection methods.

About the Author

Jesse D. Kornblum is a Senior Computer Forensics Engineer for ManTech's Computer Forensics and Intrusion Analysis Division. Based in the Washington DC area, his research focuses on computer forensics and computer security. He has authored a number of computer forensics tools including the widely used md5deep suite of cryptographic hashing programs and the First Responder's Evidence Disk. For more please visit http://research.jessekornblum.com/ or http://www.mantech.com/sma/s_cfia.asp. Please send all correspondence to jesse.kornblum@mantech.com.

References

[BETZ] Betz, Chris, *DFRWS 2005 Challenge Report*, Digital Forensic Research Workshop 2005 Memory Analysis Challenge, 2005, http://www.dfrws.org/2005/challenge/ChrisBetz-DFRWSChallengeOverview.html

[CARR] Carrier, Brian and Grand, Joe, *A Hardware-Based Memory Acquisition Procedure for Digital Investigations*, *Digital Investigation* Journal, 2004, http://www.digital-evidence.org/papers/tribble-preprint.pdf

[CRAZ] Crazylord, *Playing with Windows /dev/(k)mem*, *Phrack* Magazine Volume 0xb, Issue 0x3b, 2002, published, but offline

[GARN04] Garner Jr., George M. *Forensic Acquisition Utilities*, 2004, http://users.erols.com/gmgarner/forensics/

[GARN05] Garner Jr., George M and Mora, Robert-Jan, *Preliminary Analysis Of 2005 DFRWS Forensic Challenge*, Digital Forensic Research Workshop 2005 Memory Analysis Challenge, 2005, http://www.dfrws.org/2005/challenge/rossettoecioccolato-DFRWSChallengeOverview.pdf

[HOLY] Holy Father, *The Hacker Defender Project*, 2005, http://www.hxdef.org/

[MSFT] Microsoft Windows® XP Professional Resource Kit, 2nd Edition, Microsoft Press, 2003.

[SPAR] Sparks, Sherri and Butler, Jamie, *Raising The Bar For Windows Rootkit Detection*, *Phrack* Magazine Volume 0x0b, Issue 0x3d, 2005.