Rootkits

What is a rootkit?

A rootkit is a special variant of a Trojan, a.k.a. a RAT (Remote Administration Tool). What separates a rootkit from a regular Trojan is that a rootkit, by definition, occupies Ring 0, also known as root or kernel level, the highest run privilege available, which is where the OS (Operating System) itself runs. Non-rootkit trojans typically run in Ring 3, or user level, which is where ordinary applications run, though some sources refer to userland trojans as "rootkits" also. Usually, but not always, a rootkit will actively obfuscate and attempt to hide its presence from the user and any security software present.

Rootkits subvert the OS through the kernel (core operating system) or privileged drivers. This enables a rootkit to operate as a part of the OS itself rather than a program being run by the OS. This high level of sophistication makes rootkits extremely difficult to detect and remove. Often anti-virus products will be unable to detect or remove a rootkit once it has taken over the OS and more specialized detection and removal procedures are required.

What kinds of rootkits are there?

Rootkits may use a variety of techniques to gain control of the operating system and hide from both the user and security software. Different techniques may be used in combination to increase overall effectiveness. There are many variations and not every technique will be discussed. Only those most relevant and common will be discussed here. [1] Some common techniques include:

MBR infection

The MBR or Master Boot Record is the portion of the hard drive that tells the BIOS (Basic Input Output System) where to find the OS (Operating System). This is a critical handoff of responsibility between the BIOS which does the initial boot sequence when the computer is started and the OS which takes over. By subverting this process the rootkit (sometimes called bootkit) is able to inject itself between the computer's hardware and OS, subtly altering data sent back and forth to mask its presence and take over the system.

Every time the OS tries to read files from the hard drive the rootkit intercepts the attempt and substitutes either fake data to hide itself or modified data to trick the OS into loading and executing infected files. By selectively intercepting attempts to read and execute kernel drivers the rootkit loads itself into memory and takes over the OS. If the user attempts to view the rootkit files, the rootkit can give a false report of there being no trace of its files. Since the rootkit often never actually modifies the OS files on the hard drive itself, but only gives modified data when the file is being loaded into memory, it becomes even harder to detect. It can also detect and intercept any attempt to delete the rootkit itself or any portion thereof. Even if the rootkit is deleted, since it is loaded in the MBR, the system can be reinfected when it is rebooted.

1. OS subversion techniques used by ordinary trojans, such as IAT and EAT hooks, malicious App_Init DLLs, etc are out of scope for this document. Only kernel level attacks are presented, so additional information regarding these and other user mode attacks are left to the reader.

Newer versions of Windows incorporate protections to prevent the MBR from being written to. However, rootkits have evolved to overcome this by writing directly to the disk using SRB (SCSI Request Blocks). Not all computers use the MBR method of booting the OS. Newer PCs may use EFI (Extensible Firmware Interface) or UEFI (Unified Extensible Firmware Interface) which will be discussed later in the document.

Hypervisor

A hypervisor is a virtual machine manager, which when used for legitimate purposes allows a single physical computer to host and run more than one OS simultaneously by creating multiple virtual machines, each of which appear to the OS to be a physical computer. It simulates hardware and intercepts attempts by the OS to access the hardware, then translates the request, and passes it to the actual hardware. Hypervisors have many legitimate uses in computing, however a rootkit can create a malicious hypervisor to hide its existence from the OS and the user.

There are several types of hypervisor rootkits. Some modify the bootloader to create the malicious hypervisor during the bootup process in a way very similar to an MBR rootkit. Others can subvert the OS and migrate it into a virtual machine while it is still running, without any indication to the user and without requiring a reboot. This is possible due to hardware support for virtualization built into most modern CPUs. Intel's virtualization architecture is called VT-x and AMD’s is called Pacifica.

A hypervisor rootkit would subvert a running OS by first checking to see whether the hardware supports virtualization using a function such as *vmx_init*. It would then reallocate system memory and split system resources using a function such as *vmx_fork* which will put the rootkit into a privileged Supervisor mode beyond Ring 0. It will then put the running OS and all active processes into a non-privileged non-root mode where they cannot see or interact with the actual hardware or the processes of the rootkit.

The hypervisor rootkit emulates virtual hardware for the OS, which the OS cannot detect to be any different from the actual hardware. In such a situation the rootkit becomes almost impossible to detect from within the compromised OS, because it controls what the OS "sees". The only certain way is to do a forensic exam of the hard drive to look for backdoors or modifications to the bootloader which would allow the rootkit to reload after a reboot.[2]

The rootkit can also suspend its operation and even temporarily exit out of virtualization mode if it detects the OS is attempting an operation which may uncover its existence, such as by checking to see if virtualization extensions are active or attempting to detect timing irregularities in certain system calls such as CPUID. Because a hypervisor introduces a certain amount of latency in addition to what would normally be expected without a hypervisor it may be possible to detect some less sophisticated hypervisor rootkits. This is not reliable however for advanced rootkits which can suspend or exit the virtual mode temporarily.

2. Hypervisor rootkits which are injected into memory and do not modify the file structure on the hard drive will not be able to be detected by an examination of the hard disk, but will also not survive reboot.

Alternate Data Streams

Alternate Data Streams or ADS are a little known function of NTFS, a popular file system used by Microsoft Windows products. ADS allows the OS to store metadata about a file without changing the file itself. ADS are not viewable by Windows Explorer or other common file viewers. They make a very good hiding place for rootkits because there is no limit to the number or size of files that can be stored invisibly in ADS. An executable may be stored in ADS and executed without ever showing up on a file or directory listing. More and more AV (anti-virus) products are now scanning ADS, so this is no longer widely used for modern trojans, but is still common in rootkits a few years old. However, if you examine an infected hard drive on a non-infected computer, you may be unable to detect the rootkit files using standard file explorers and will need specialized tools which can scan ADS. It is possible to manually create and read ADS streams, but only if you know the exact stream identifier expressed in the form *"drive letter:\path\filename:stream"*. For example *c:\temp\tmpfile.tmp:hidden.exe*

Slack space

Every file on a hard drive is allocated a certain amount of space. Because space is allocated in fixed size "chunks" or disk clusters, most often the file that has been allocated the space doesn't use all of its allocated space and there is a little bit left over. This is known as slack space. Rootkits have long been known to hide in such areas of the disk, spread out over the slack space allocated to several normal files. Ordinary attempts to read the portions of the disk where rootkit resides will simply show the file to which those disk clusters have been allocated. It takes specialized tools to read these sections of the disk, and even then it is difficult to tell a rootkit in slack space from the random junk data that would normally be there anyway.

A rootkit taking advantage of this method will most likely store itself in the slack space of protected system files that will not change much or ever, because of the risk of having itself overwritten when the file to which the space is allocated grows in size. Most AV tools and even AR (anti-rootkit) tools are not able to scan slack space, which makes this an excellent hiding place for malware which will enable it to remain undetected even when the hard drive is examined on a non-compromised system.

Bad Sectors

Over time a hard drive may develop sectors (storage units) which can no longer be reliably read from or written to, these are called bad sectors or bad blocks. The OS keeps a record of these bad sectors in the MFT in Windows and the bad blocks inode in Linux so it will not try to write to them in the future. Sectors marked as bad are generally not readable because in most modern drives they are transparently mapped to a pool of spare sectors either by the drive controller hardware or in some cases the OS.

Because of this bad sectors make a favored hiding place for rootkits, preferred over slack space because there is no danger of data in bad sectors being overwritten. The rootkit simply marks the locations on disk where its files are stored as bad, making those sectors inaccessible without direct disk access. Most software uses APIs (Application Programming Interface) to access hardware, which requires the hardware access request to go through the OS. This data hiding technique makes the rootkit invisible to both regular AV and even specialized AR tools which use standard APIs for scans. Forensic software capable of direct disk access and reading raw sector data would be required to locate data stored in bad sectors, and often rootkits using this method of hiding will intercept direct disk access requests requiring the disk to be examined on a non-compromised system.

Hidden Partition

A partition is a logical division of the physical hard drive used for data access. Some rootkits create a hidden partition within an existing disk partition. In order to do this the rootkit has to create a disk object and a disk driver to access the new hidden disk. In a Windows system this would either involve copying the existing *disk.sys* driver object and modifying the dispatch function and device object to point to the hidden partition or creating a whole new device object and driver set from scratch.

Usually the IRP table will also be hooked to monitor and control access to the hidden disk object and prevent the OS from accidentally overwriting the hidden data since it overlaps the ordinary disk partition the OS already knows about. The rootkit may also create a fake file and allocate the portion of the disk used by the hidden partition to the fake file to prevent the OS from trying to allocate that space for another purpose.

Commonly the hidden partition will be allocated a section of the hard drive at the very end as this is the least likely to already have data. Any existing data will be moved and the rootkit will intercept access attempts and transparently redirect them to wherever it has moved the data. Modern rootkits will also encrypt the hidden partition making it impossible to read without the correct encryption key and encryption algorithm.

Interrupt Hooks

The OS uses a set of basic commands to interface with the computer hardware as mediated by the BIOS. These commands are known as interrupt calls and given numbers in hexadecimal. A rootkit which is able to intercept and modify these calls is said to have hooked that call. Depending on how the interrupt is hooked it may be known as an INT hook or IDT hook. Since interrupt calls are the most basic, a rootkit which is able to hook them has control over the hardware at a very low level. This technique is most commonly seen in MBR rootkits because INT calls are used in the boot process. Specifically *INT 13h*, which enables direct access to the hard drive, is commonly hooked by MBR rootkits. This enables a rootkit to modify the disk directly, subverting any access control on the part of the OS. It also enables the rootkit to intercept any attempt by the OS to read or modify data on the disk and prevent or alter attempted data reads or modifications.

Message Hooks

Programs running in memory use messages to communicate changes and user input to other programs and the OS. A message hook is used to either monitor or intercept messages before they reach the intended system process. For Windows OS they are created by calling the *SetWindowsHook* function with appropriate parameters. Rootkits will often set message hooks because all user input, keystrokes and mouse movements, creates messages. A rootkit which has hooked these messages will be able to read and record all user activity on the PC. Since there are many different messaging subroutines, it allows very fine grained control over which functions will be monitored. Some common message hooks used by rootkits are *WH_KEYBOARD*, *WH_KEYBOARD_LL*, *WH_MSGFILTER*, and *WH_MOUSE*.

SSDT Hooks

The System Service Descriptor Table or SSDT is used by Windows OS to locate system services which are crucial to the functioning of the OS. In Linux OS this function is held by the System Call Table. A rootkit which hooks this table can alter it so that important system calls are routed to the rootkit. Any program which attempts to use the SSDT will instead be funneled to the rootkit, and since the SSDT is fundamental to the OS, every program must use it. SSDT hooks are very powerful and commonly used by rootkits for stealth.

For example, if the *NtQueryDirectoryFile* function is hooked, the rootkit can return false information to requesting programs, such as AV, about files and directories on the hard drive, making itself invisible. In the same way, a rootkit may hide its running processes, network activity, or Registry entries, such as with *NtEnumerateKey* and *NtEnumerateValueKey* or for Linux *sys_getdents* and *sys_write*.

Because of the frequent use of SSDT hooks, many anti-rootkit programs scan the SSDT for modifications, however rootkits are able to hide changes to the SSDT in a variety of ways, such as by modifying the *KTHREAD* structure or modifying the SSDT "on the fly" without leaving permanent traceable changes. Newer Microsoft OS and 64bit OS have made hooking the SSDT much more difficult, however this is still very common on Windows XP rootkits.

IRP Hooks

Any time a program needs to send or receive data from the computer hardware an I/O Request Packet (IRP) is used as an intermediary between hardware and software. This includes reading and writing data from the hard drive, RAM, video, audio, and network. Hooking IRP generally involves modifying or replacing hardware drivers. Rootkits use this method as another way of gaining privileged access to hardware, while intercepting other access attempts.

A rootkit which has modified disk driver *disk.sys* or the low level disk driver *atapi.sys* can control what other programs see on the hard drive, while *tcpip.sys* allows a rootkit to hide network traffic. Initially few rootkits used these techniques, but as other techniques came under more scrutiny, more and more rootkits began using IRP hooks and coming up with novel ways to hook the IRP subsystem without leaving obvious hooks in place.

For example by modifying the lowest level device driver for the hard drive *\Device\Harddisk0\DR0* to no longer point to the default IRP handling subsystem via *IRP_MJ_INTERNAL_DEVICE_CONTROL* routine but a parallel system controlled by the malware, or adding a malicious device into a target device's IRP chain via *IoAttachDevice*. These are both sneaky ways to redirect IRPs without having to modify the IRP dispatch table itself. Since there are many different drivers for hardware, this makes detecting hooks that much harder for anti-rootkit software, especially since, unlike SSDT, pointers in the IRP table are not all expected to point back to the kernel, since there are many 3rd party drivers in use.

Other commonly hooked procedures include: *IRP_MJ_READ*, *IRP_MJ_WRITE*, *IRP_MJ_SCSI*, and *DriverStartIo*. Since some AR products began using passthrough IOCTLs to directly access the disk and bypass the rootkit hooks, newer rootkits are additionally hooking *IRP_MJ_DEVICE_CONTROL* subcontrols such as *IOCTL_ATA_PASS_THROUGH* and *IOCTL_ATA_PASS_THROUGH_DIRECT* or *SCSIOP_READ* and *SCSIOP_WRITE*.

DKOM

A kernel object is a virtual placeholder for a resource that contains information about it. Everything on a computer will have an associated kernel object, every file, every process, every port, etc. When a kernel object is created, it is given an index number called a handle, through which it is accessed. When a program wants to make a change (e.g. create or destroy a process), it makes a request to change the kernel object, and the kernel itself (Object Handler) decides whether to grant or deny the request.

Normally the kernel itself is the only one able to directly change kernel objects, however, in the last few years, rootkits have appeared which are able to access kernel objects directly in what is called Direct Kernel Object Manipulation (DKOM). It is another tool in the toolbox of the malware writer to be able to hide thier own processes and drivers while interfering with other processes and files. But it is much more stealthy than other methods such as replacing device drivers and hooking tables for 2 reasons: 1. Because changes occur in memory only, there is no record of them, and 2. Because no other program, not even AV, can access the kernel objects, what happens in this reserved memory region is somewhat "behind the curtain".

In Linux DKOM can be accomplished by writing to */dev/mem* or */dev/kmem*. A DKOM rootkit in Windows XP will use the undocumented API *NtSystemDebugControl* a hidden API used to directly access kernel memory. However, it must open a handle to the memory at *\\Device\PhysicalMemory*, which is one method of detecting it.

By modifying the *EPROCESS* structure, a DKOM rootkit can hide running processes. Other often modified kernel objects are *ETHREAD*, *TOKEN*, and *DRIVER*. These attacks allow the rootkit to hide processes and device drivers and change process access tokens. A rootkit that modifies the kernel object of the page fault handler can hide the contents of RAM from any other program. This means such a rootkit can hide its own existence even from a scan of objects in memory or running processes.

However, rules for manipulating kernel objects will change from one version of the OS to another, making manipulation of those objects challenging, also because of the delicacy of the operations involved any mistake will result in a system crash, which can be a giveaway. Despite the difficulties in DKOM, it is expected more and more rootkits will be using them in the future, since advances in OS security are rendering hooks more difficult and because all OS must use kernel objects.

The latest versions of some rootkits are using DKOM to great effect by blending it with IRP hooking, using DKOM to create phony devices and setting IRP hooks on the phony device while using DKOM to link the phony and real device by modifying the *OBJECT_HEADER* structure. In this way, the actual device is not shown as being hooked, so it can evade anti-rootkit techniques. There is a great deal of innovation occurring with DKOM rootkits and more creative methods of using them to manipulate and hide data is to be expected.

Rootkit Trends - 2011

Rootkits are increasingly developed by professional malware developers working in teams and accordingly are becoming highly sophisticated and complex, comparable in many ways to the AV and AR products devoted to catching them. Modern rootkits are highly obfuscated to confuse forensics and frustrate reverse engineering, incorporate encrypted files, encrypted communications, and a modular design that allows different types of malware from different designers to work together by exporting malicious APIs and syscalls. This modular design allows malware developers to specialize in one particular area: initial infection, hiding malware files and activity, payload functionality, ie botnet, search engine results modification, sending spam emails, capturing sensitive user data, etc, and specialized plugin functions, ie keylogging, HTTPS, etc. These trends are making rootkits more flexible and powerful as well as harder to detect and remove.

Rootkit detection

Since rootkits go to great pains to hide, they can be quite difficult to detect. Additionally, since kernel rootkits run in Ring 0, they can subvert any other software running, including tools trying to find them. For this reason, it is a good idea to take the hard drive out of the suspected infected machine and attach it to a known clean machine for examination.

One of the first indicators of a rootkit infection is system instability. Since rootkits often replace core system drivers, any malfunction will crash the system. Since rootkit drivers are not subject to the same quality standards of an OS vendor bugs and system crashes are common, though this is becoming less true over time as professional level rootkits become more common. Additionally, often rootkits are designed to work with a very specific patch level for an OS. So if the OS is patched and some dll is replaced that the rootkit has modified, it can cause serious system problems, such as lockups and crashes. But then there are a few rootkits that don't even try to be stealthy and pop up advertisements for pornography as well. All of these can be potential indicators that a deeper examination is needed.

Prior to making any changes to a potentially rootkit compromised system, it is a good idea to learn as much through passive observation as possible. Many rootkits monitor system activity very closely and are programmed to look for anti-rootkit programs running in memory and attempts to read or change sensitive areas of the OS and hard drive which may represent attempts to detect or remove the rootkit.

Rootkits with an observer process will usually have some self defense code which will activate if it detects any attempt to remove the rootkit. This can be anything from terminating the process, to unhooking hooked tables and drivers, to moving its code around in memory or on disk in an attempt to thwart investigation. For this reason, it is a good idea to make a clone of the hard disk of the potentially infected machine to examine without running the risk of alerting the rootkit on the running machine that it is being investigated. With a clone you can safely kill processes, modify files, and generally poke into the suspected rootkit and observe if there is unusual behavior in response to this.

Kernel Mode Signing

One of the major security flaws of past Windows OS is that device drivers were loaded in Ring 0. This is a major problem because device drivers often come from 3rd parties and are unverified, meaning they could be buggy or include malicious code. This was a common way for rootkits to load themselves into kernel memory in the past. 64 bit versions of newer Windows, Vista and later, incorporate a security measure called kernel mode signing. This requires all kernel mode drivers to be cryptographically signed, certifying their origin and trusted status.

Modern rootkits have found ways to overcome this security control. Rootkits which subvert the MBR may use functions normally used for debugging purposes, *BcdLibraryBoolean_DisableIntegrityCheck* and *BcdLibraryBoolean_AllowPrereleaseSignatures*. Since an MBR rootkit controls the boot process it is able to set either of these options at boot time to disable code signing requirements and load malicious, unsigned kernel drivers.

Kernel Mode Patch Protection

Another security feature found in 64 bit versions of Windows, XP and newer, is kernel mode patch protection (KPP) also known as PatchGuard. It prevents modifications to the SSDT, IDT, GDT, and MSRs, creation of kernel stacks, and inline patching of the kernel or kernel libraries. However, PatchGuard has several well known bypass techniques, including hooking and/or modifying the PatchGuard code itself or supporting system functions like the exception handler. Because the code PatchGuard is attempting to regulate runs in Ring 0, it has full access to the kernel and there is an ongoing cycle of attacks to disable or evade PatchGuard's protections and updates to PatchGuard to counter those attacks.

Unified Extensible Firmware Interface

The security design flaw exploited by MBR rootkits is that if they can get direct access to the hardware at boot time all future software checks become meaningless. UEFI includes a security control to eliminate this threat in the hardware itself called secure boot. Secure boot requires cryptographic signatures on all code loaded at boot time. The signatures create a chain of trust from the software developer up to the certifying authority which certifies the software as trusted. Any unauthorized modifications to a signed bootloader will cause the integrity check to fail and prevent the system from booting. While this is not fool proof it does provide a high degree of protection against rootkits and other malware which may attempt to modify the bootloader or key boot components, i.e. *NTLDR, bootmgr, winload.exe, winresume.exe,* or *kdcom.dll*. UEFI is becoming more commonplace and is widely supported by hardware manufacturers and most modern OS. As of this writing, there have been no verified instances of malware able to bypass UEFI protections.[3]

Hardware Assisted Security

A major stumbling block to anti-rootkit efforts is the fact that all software running in privileged execution mode (ring 0) on the CPU and with direct access to hardware is effectively on equal terms with the OS, meaning a rootkit can alter or disable the AR software hunting for it. Several attempts have been made to incorporate AR technology directly into the hardware to give more of an advantage. One of these was a PCI card called copilot which contained rootkit hunting code burned into the firmware, able to monitor the host's memory and filesystem at the hardware level. This technology never caught on in the private sector but was popular in the government sector.

Another hardware assisted security technology is called DeepSAFE. This relies on virtualization, creating a hypervisor that runs at a higher level of privilege than the OS and kernel level code within the OS, including rootkits. This means that the scans running from within the hypervisor based security code cannot be easily bypassed because it is not vulnerable to hooking from the OS layer. It can also freeze the running system and examine the contents of RAM directly without having to rely on the OS, which may have been subverted.

Compare Integrity Assurance Snapshot

If you have a snapshot of the hard drive from a known clean state using one of the many intergrity assurance software products, such as Tripwire, Samhain, OSSEC, AFICK, or AIDE, you can use it to track changes to the hard drive. This will show you files and registry settings added, removed and altered, which is a good first step to trying to track down changes made by a rootkit.

3. Secure boot depends on the chain of trust established by certificate authorities, which has been successfully broken in rare instances. PKI and chain of trust attacks are outside the scope of this paper.

Be aware that the registry changes frequently as a matter of course and temp files are regularly created and deleted in the appropriate folders. Rootkit authors are aware of this and may try to mimic these normal patterns by hiding a rootkit in /tmp or a .tmp file for example. Look for changes in any critical OS directories and cross reference with the logs to determine if those were authorized changes. Registry entries which could be used to load a rootkit into memory should also be given special attention, some examples would be:

HKLM\SYSTEM\CurrentControlSet\Services,
HKLM\Software\Microsoft\Windows\CurrentVersion\*
HKCU\Software\Microsoft\Windows\CurrentVersion\*
*HKLM\Software\Microsoft\Internet Explorer**
*HKCU\Software\Microsoft\Internet Explorer**
HKCR\exefile\shell\open\command
HKLM\Software\Classes\exefile\shell\open\command
HKLM\Software\Microsoft\ActiveSetup\InstalledComponents

Anti-Rootkit Products

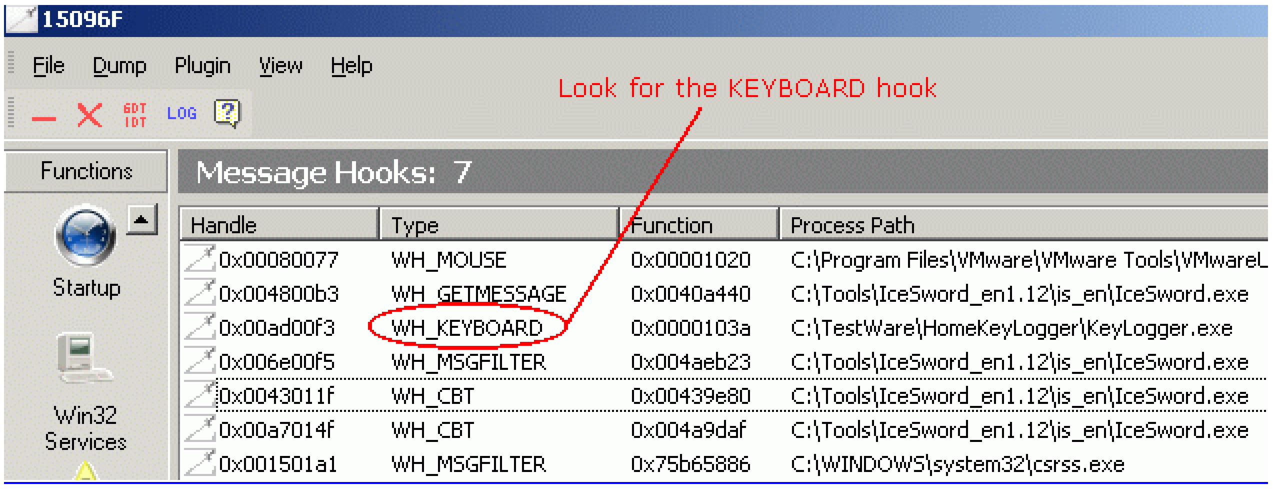
There are a number of specialized anti-rootkit (AR) software products available, some free and some commercial products. Some Windows AR include: Rootkit Revealer, Blacklight, Rootkit Unhooker, GMER, Icesword, RAIDE, and Helios. Some Linux AR include: chkrootkit, Rkdetector, rkhunter, Zeppoo, kstat, elfstat, and KsID. While none of them are capable of detecting every rootkit, they can provide some very useful information about the state of the OS.

Many older rootkits use direct SSDT and IAT hooks. In other words they modify the tables to point directly to the rootkit code. These types of changes are trivially easy for a scanner to detect. The AR scanner simply scans the IAT and SSDT tables for pointers which don't point to the kernel itself. It then presents a list of these hooks to the user for examination.

However, just because a hook is present, doesn't mean there is a rootkit. There are other legitimate software applications which may also install hooks. System security software such as AV and firewall will often hook SSDT tables. Poorly programmed software which should use hooks limited to its own process, may instead install global keyboard or mouse hooks which an AR scanner will flag as suspicious. AV and firewalls will often hook the network stack or device drivers (ie chained or filtered device drivers) to protect the system. ADS is used by jpeg image files and saved webpages. Software debuggers will often hook exception handling APIs. In Linux systems, SE_Linux will often hook the sys call table. In theory, there should be few enough hooks in an OS to carefully examine each one to determine whether it is malicious or part of a known process. However, in order to counter rootkits which become ever deeply buried in the OS, modern AV and AR products often embed themselves just as deeply into the OS, in some cases using live kernel patching techniques. In effect become benign rootkits themselves. The documentation of system modifications for many of these products is woefully incomplete or non-existent and because of this in some cases it may not be possible to determine whether a given hook or kernel patch is a sign of a rootkit or an undocumented AV or firewall function without removing the software.

Several examples of both benign and malicious hooks and kernel patches will be shown to provide reference for your own investigations.

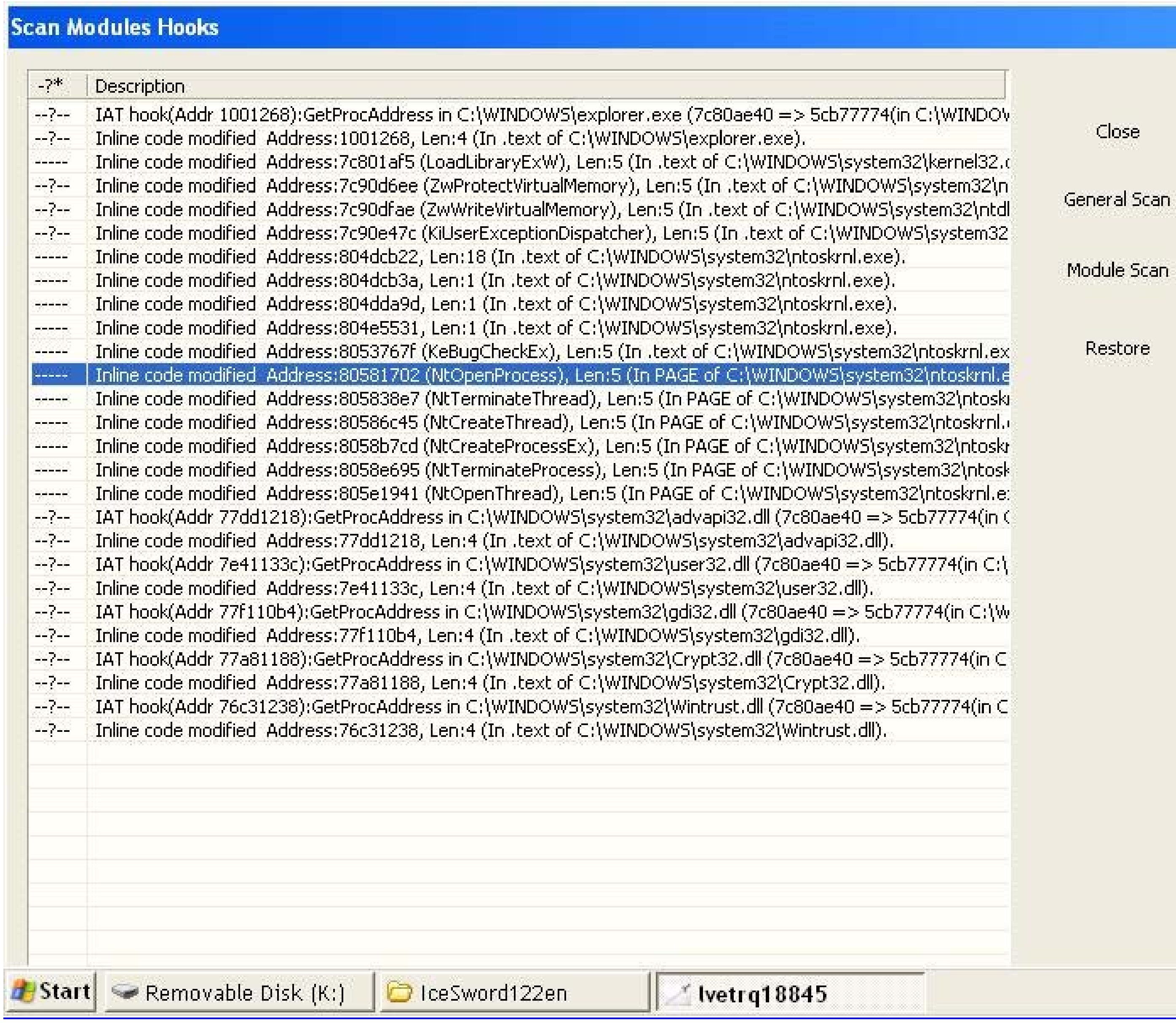
This screenshot shows Icesword reporting a global keyboard hook.



Handle	Type	Function	Process Path
0x00080077	WH_MOUSE	0x00001020	C:\Program Files\VMware\VMware Tools\VMwareL
0x004800b3	WH_GETMESSAGE	0x0040a440	C:\Tools\IceSword_en1.12\is_en\IceSword.exe
0x00ad00f3	WH_KEYBOARD	0x0000103a	C:\TestWare\HomeKeyLogger\KeyLogger.exe
0x006e00f5	WH_MSGFILTER	0x004aeb23	C:\Tools\IceSword_en1.12\is_en\IceSword.exe
0x0043011f	WH_CBT	0x00439e80	C:\Tools\IceSword_en1.12\is_en\IceSword.exe
0x00a7014f	WH_CBT	0x004a9daf	C:\Tools\IceSword_en1.12\is_en\IceSword.exe
0x001501a1	WH_MSGFILTER	0x75b65886	C:\WINDOWS\system32\csrss.exe

Newer rootkits do not directly hook tables, but instead modify the code of the legitimate API handler or dll to insert a *JMP* instruction within the file header that points to the rootkit. This leaves the table intact and unmodified, but any process which attempts to call that API will get redirected to the rootkit. In some cases the file on disk may be left intact as well and only running code modified.

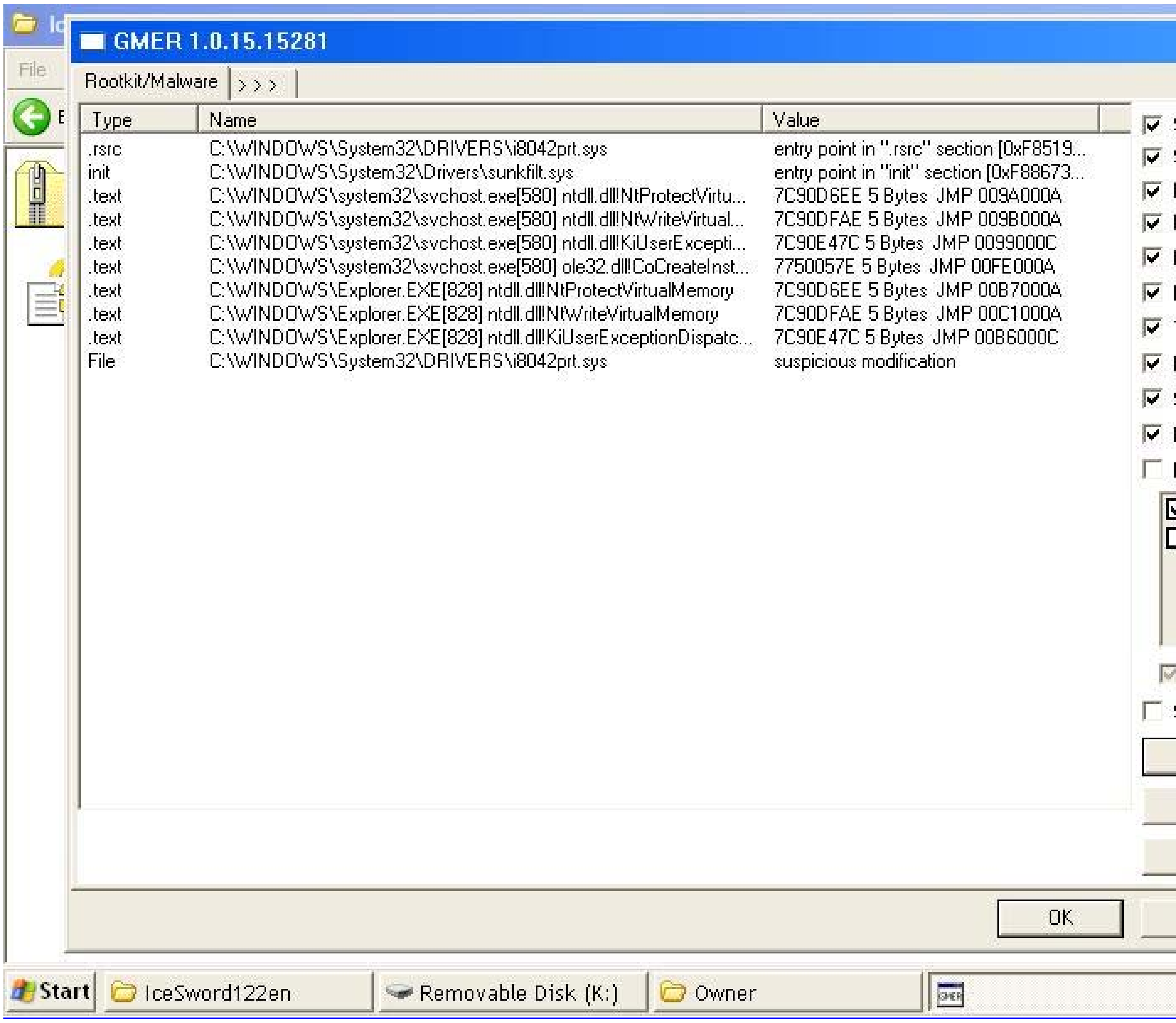
This screenshot shows Icesword reporting a number of kernel hooks. Of particular note is that malicious code has been injected into *ntoskrnl.exe* the OS kernel for Windows, which has hooked the SSDT APIs for *NtOpenProcess*, *NtTerminateThread*, *NtCreateThread*, *NtCreateProcessEx*, *NtTerminateProcess*, and *NtOpenThread*. This particular rootkit is able to monitor and control any attempt to start a new process or kill an existing one.



Scan Modules Hooks

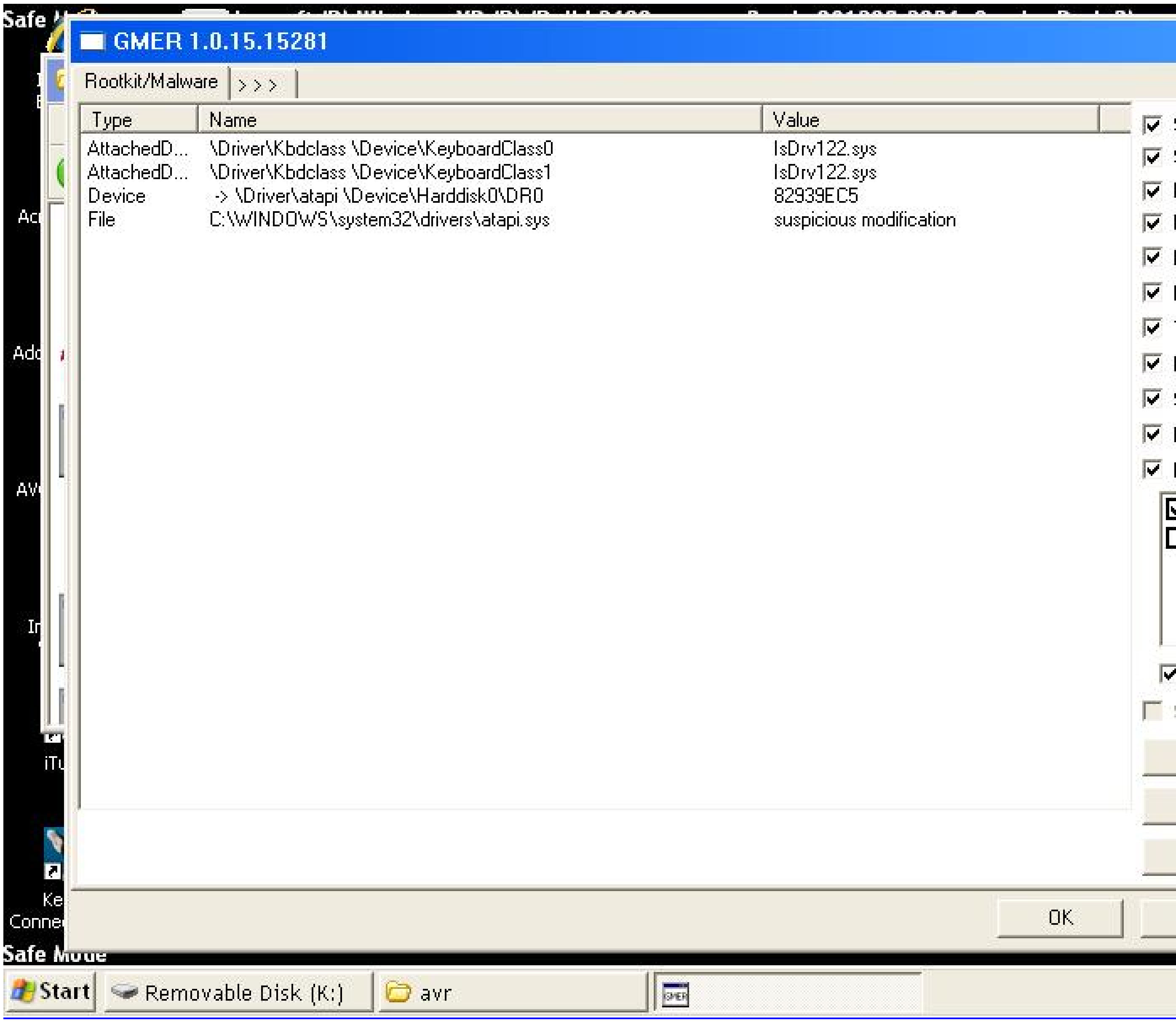
-?*	Description
--?--	IAT hook(Addr 1001268):GetProcAddress in C:\WINDOWS\explorer.exe (7c80ae40 => 5cb77774(in C:\WINDOV
--?--	Inline code modified Address:1001268, Len:4 (In .text of C:\WINDOWS\explorer.exe).
-----	Inline code modified Address:7c801af5 (LoadLibraryExW), Len:5 (In .text of C:\WINDOWS\system32\kernel32.d
--?--	Inline code modified Address:7c90d6ee (ZwProtectVirtualMemory), Len:5 (In .text of C:\WINDOWS\system32\n
--?--	Inline code modified Address:7c90dfae (ZwWriteVirtualMemory), Len:5 (In .text of C:\WINDOWS\system32\ntdl
--?--	Inline code modified Address:7c90e47c (KiUserExceptionDispatcher), Len:5 (In .text of C:\WINDOWS\system32
-----	Inline code modified Address:804dcb22, Len:18 (In .text of C:\WINDOWS\system32\ntoskrnl.exe).
-----	Inline code modified Address:804dcb3a, Len:1 (In .text of C:\WINDOWS\system32\ntoskrnl.exe).
-----	Inline code modified Address:804dda9d, Len:1 (In .text of C:\WINDOWS\system32\ntoskrnl.exe).
-----	Inline code modified Address:804e5531, Len:1 (In .text of C:\WINDOWS\system32\ntoskrnl.exe).
-----	Inline code modified Address:8053767f (KeBugCheckEx), Len:5 (In .text of C:\WINDOWS\system32\ntoskrnl.ex
-----	Inline code modified Address:80581702 (NtOpenProcess), Len:5 (In PAGE of C:\WINDOWS\system32\ntoskrnl.e
-----	Inline code modified Address:805838e7 (NtTerminateThread), Len:5 (In PAGE of C:\WINDOWS\system32\ntoskı
-----	Inline code modified Address:80586c45 (NtCreateThread), Len:5 (In PAGE of C:\WINDOWS\system32\ntoskrnl.
-----	Inline code modified Address:8058b7cd (NtCreateProcessEx), Len:5 (In PAGE of C:\WINDOWS\system32\ntoskr
-----	Inline code modified Address:8058e695 (NtTerminateProcess), Len:5 (In PAGE of C:\WINDOWS\system32\ntosk
-----	Inline code modified Address:805e1941 (NtOpenThread), Len:5 (In PAGE of C:\WINDOWS\system32\ntoskrnl.e:
--?--	IAT hook(Addr 77dd1218):GetProcAddress in C:\WINDOWS\system32\advapi32.dll (7c80ae40 => 5cb77774(in C
--?--	Inline code modified Address:77dd1218, Len:4 (In .text of C:\WINDOWS\system32\advapi32.dll).
--?--	IAT hook(Addr 7e41133c):GetProcAddress in C:\WINDOWS\system32\user32.dll (7c80ae40 => 5cb77774(in C:\
--?--	Inline code modified Address:7e41133c, Len:4 (In .text of C:\WINDOWS\system32\user32.dll).
--?--	IAT hook(Addr 77f110b4):GetProcAddress in C:\WINDOWS\system32\gdi32.dll (7c80ae40 => 5cb77774(in C:\W
--?--	Inline code modified Address:77f110b4, Len:4 (In .text of C:\WINDOWS\system32\gdi32.dll).
--?--	IAT hook(Addr 77a81188):GetProcAddress in C:\WINDOWS\system32\Crypt32.dll (7c80ae40 => 5cb77774(in C
--?--	Inline code modified Address:77a81188, Len:4 (In .text of C:\WINDOWS\system32\Crypt32.dll).
--?--	IAT hook(Addr 76c31238):GetProcAddress in C:\WINDOWS\system32\Wintrust.dll (7c80ae40 => 5cb77774(in C
--?--	Inline code modified Address:76c31238, Len:4 (In .text of C:\WINDOWS\system32\Wintrust.dll).

This screenshot shows GMER reporting inline or "hidden" hooks in the *ntdll.dll* process which is used to handle translating user mode applications (Ring 3) API requests to the kernel. In addition to hooking the virtual memory handler, this rootkit has also hooked *i8042prt.sys* and *sunkfilt.sys* a keyboard and mouse driver respectively.



Type	Name	Value
.rsrc	C:\WINDOWS\System32\DRIVERS\i8042prt.sys	entry point in ".rsrc" section [0xF8519...
init	C:\WINDOWS\System32\Drivers\sunkfilt.sys	entry point in "init" section [0xF88673...
.text	C:\WINDOWS\system32\svchost.exe[580] ntdll.dll!NtProtectVirtu...	7C90D6EE 5 Bytes JMP 009A000A
.text	C:\WINDOWS\system32\svchost.exe[580] ntdll.dll!NtWriteVirtual...	7C90DFAE 5 Bytes JMP 009B000A
.text	C:\WINDOWS\system32\svchost.exe[580] ntdll.dll!KiUserExcepti...	7C90E47C 5 Bytes JMP 0099000C
.text	C:\WINDOWS\system32\svchost.exe[580] ole32.dll!CoCreateInst...	7750057E 5 Bytes JMP 00FE000A
.text	C:\WINDOWS\Explorer.EXE[828] ntdll.dll!NtProtectVirtualMemory	7C90D6EE 5 Bytes JMP 00B7000A
.text	C:\WINDOWS\Explorer.EXE[828] ntdll.dll!NtWriteVirtualMemory	7C90DFAE 5 Bytes JMP 00C1000A
.text	C:\WINDOWS\Explorer.EXE[828] ntdll.dll!KiUserExceptionDispatc...	7C90E47C 5 Bytes JMP 00B6000C
File	C:\WINDOWS\System32\DRIVERS\i8042prt.sys	suspicious modification

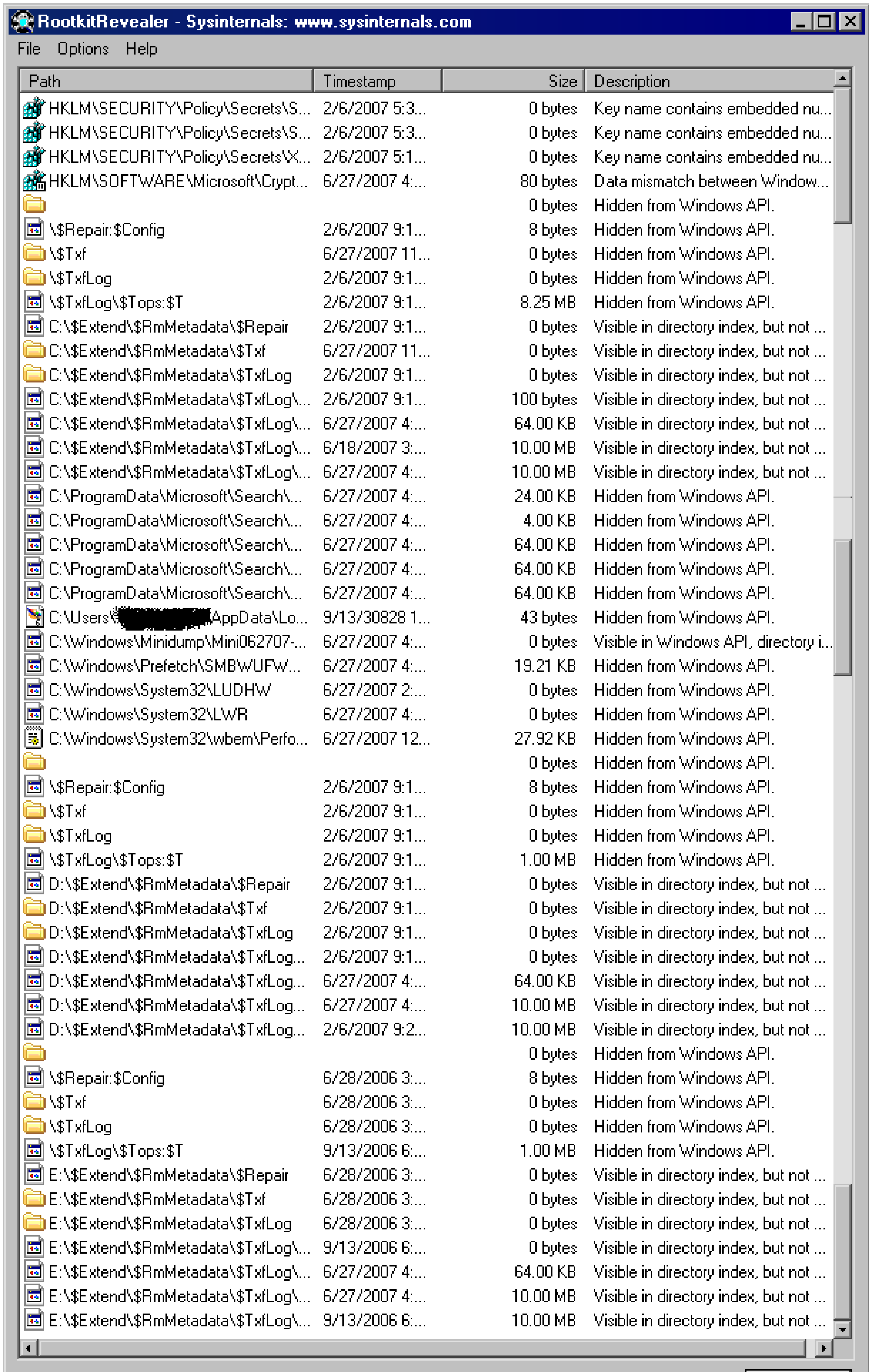
This screenshot shows GMER reporting a keyboard hook and an IRP hook in *atapi.sys*, a low level hard disk driver. This is not a sure sign in itself as some change rollback or shadow copy software may use IRP hooks in the disk driver, but it should be examined very carefully.



Type	Name	Value
AttachedD...	\Driver\Kbdclass \Device\KeyboardClass0	IsDrv122.sys
AttachedD...	\Driver\Kbdclass \Device\KeyboardClass1	IsDrv122.sys
Device	-> \Driver\atapi \Device\Harddisk0\DR0	82939EC5
File	C:\WINDOWS\system32\drivers\atapi.sys	suspicious modification

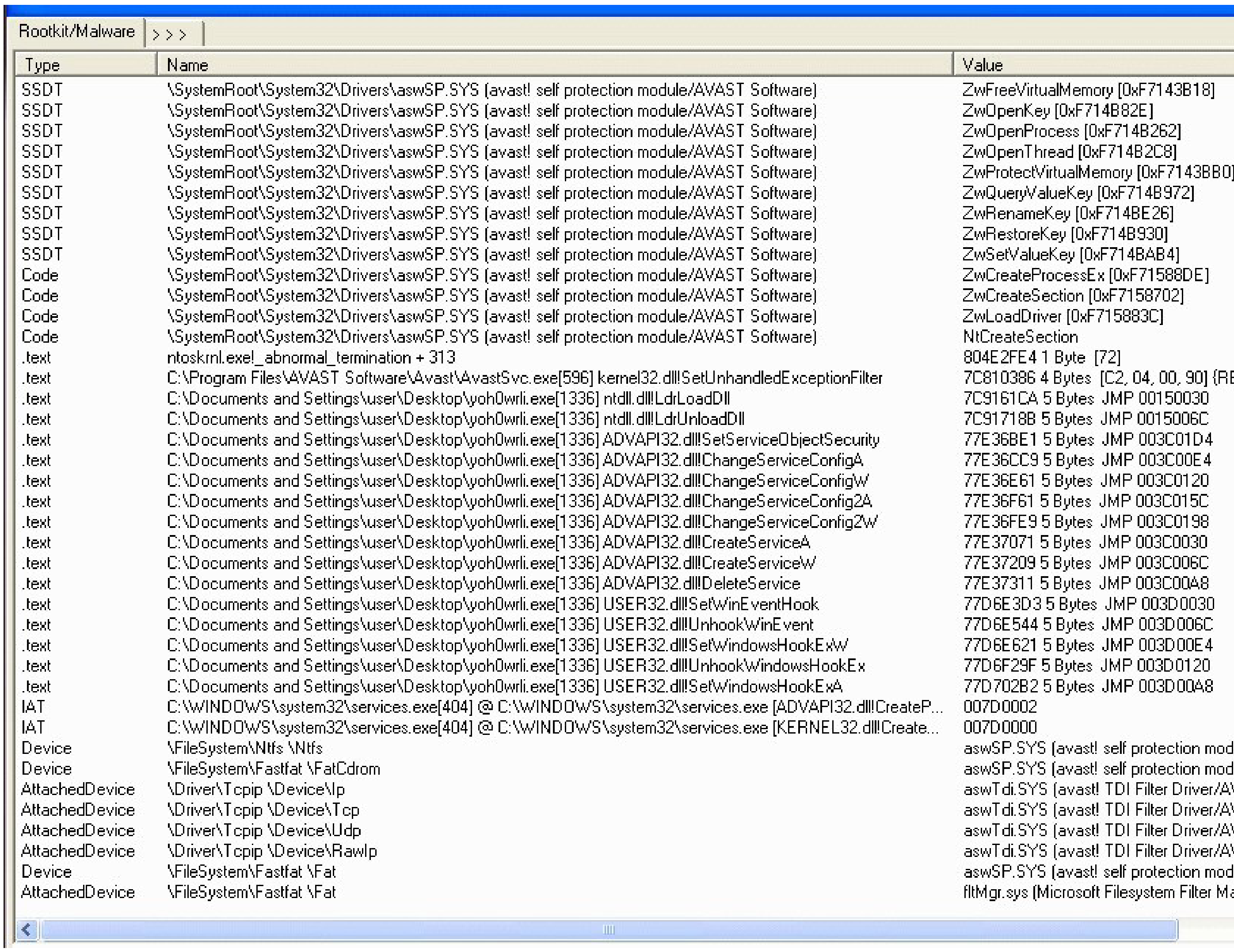
Another common technique among AR products is to examine raw disk data and compare it to data reported by APIs, or comparing the processes listed in PsActiveProcessHead with the processes listed by Task Manager. Discrepancies are reported as hidden processes and files.

This screenshot shows Rootkit Revealer reporting a number of hidden files. However, these all appear to be false positives. Any file which changes between the time the first (raw) scan is done and the comparative (API) scan is done will show up as discrepancies



Path	Timestamp	Size	Description
HKLM\SECURITY\Policy\Secrets\S...	2/6/2007 5:3...	0 bytes	Key name contains embedded nu...
HKLM\SECURITY\Policy\Secrets\S...	2/6/2007 5:3...	0 bytes	Key name contains embedded nu...
HKLM\SECURITY\Policy\Secrets\X...	2/6/2007 5:1...	0 bytes	Key name contains embedded nu...
HKLM\SOFTWARE\Microsoft\Crypt...	6/27/2007 4:...	80 bytes	Data mismatch between Window...
		0 bytes	Hidden from Windows API.
\$Repair:$Config	2/6/2007 9:1...	8 bytes	Hidden from Windows API.
\$Txf	6/27/2007 11...	0 bytes	Hidden from Windows API.
\$TxfLog	2/6/2007 9:1...	0 bytes	Hidden from Windows API.
\$TxfLog\$Tops:$T	2/6/2007 9:1...	8.25 MB	Hidden from Windows API.
C:\$Extend\$RmMetadata\$Repair	2/6/2007 9:1...	0 bytes	Visible in directory index, but not ...
C:\$Extend\$RmMetadata\$Txf	6/27/2007 11...	0 bytes	Visible in directory index, but not ...
C:\$Extend\$RmMetadata\$TxfLog	2/6/2007 9:1...	0 bytes	Visible in directory index, but not ...
C:\$Extend\$RmMetadata\$TxfLog\...	2/6/2007 9:1...	100 bytes	Visible in directory index, but not ...
C:\$Extend\$RmMetadata\$TxfLog\...	6/27/2007 4:...	64.00 KB	Visible in directory index, but not ...
C:\$Extend\$RmMetadata\$TxfLog\...	6/18/2007 3:...	10.00 MB	Visible in directory index, but not ...
C:\$Extend\$RmMetadata\$TxfLog\...	6/27/2007 4:...	10.00 MB	Visible in directory index, but not ...
C:\ProgramData\Microsoft\Search\...	6/27/2007 4:...	24.00 KB	Hidden from Windows API.
C:\ProgramData\Microsoft\Search\...	6/27/2007 4:...	4.00 KB	Hidden from Windows API.
C:\ProgramData\Microsoft\Search\...	6/27/2007 4:...	64.00 KB	Hidden from Windows API.
C:\ProgramData\Microsoft\Search\...	6/27/2007 4:...	64.00 KB	Hidden from Windows API.
C:\ProgramData\Microsoft\Search\...	6/27/2007 4:...	64.00 KB	Hidden from Windows API.
C:\Users\[redacted]AppData\Lo...	9/13/30828 1...	43 bytes	Hidden from Windows API.
C:\Windows\Minidump\Mini062707-...	6/27/2007 4:...	0 bytes	Visible in Windows API, directory i...
C:\Windows\Prefetch\SMBWUFW...	6/27/2007 4:...	19.21 KB	Hidden from Windows API.
C:\Windows\System32\LUDHW	6/27/2007 2:...	0 bytes	Hidden from Windows API.
C:\Windows\System32\LWR	6/27/2007 4:...	0 bytes	Hidden from Windows API.
C:\Windows\System32\wbem\Perfo...	6/27/2007 12...	27.92 KB	Hidden from Windows API.
		0 bytes	Hidden from Windows API.
\$Repair:$Config	2/6/2007 9:1...	8 bytes	Hidden from Windows API.
\$Txf	2/6/2007 9:1...	0 bytes	Hidden from Windows API.
\$TxfLog	2/6/2007 9:1...	0 bytes	Hidden from Windows API.
\$TxfLog\$Tops:$T	2/6/2007 9:1...	1.00 MB	Hidden from Windows API.
D:\$Extend\$RmMetadata\$Repair	2/6/2007 9:1...	0 bytes	Visible in directory index, but not ...
D:\$Extend\$RmMetadata\$Txf	2/6/2007 9:1...	0 bytes	Visible in directory index, but not ...
D:\$Extend\$RmMetadata\$TxfLog	2/6/2007 9:1...	0 bytes	Visible in directory index, but not ...
D:\$Extend\$RmMetadata\$TxfLog...	2/6/2007 9:1...	0 bytes	Visible in directory index, but not ...
D:\$Extend\$RmMetadata\$TxfLog...	6/27/2007 4:...	64.00 KB	Visible in directory index, but not ...
D:\$Extend\$RmMetadata\$TxfLog...	6/27/2007 4:...	10.00 MB	Visible in directory index, but not ...
D:\$Extend\$RmMetadata\$TxfLog...	2/6/2007 9:2...	10.00 MB	Visible in directory index, but not ...
		0 bytes	Hidden from Windows API.
\$Repair:$Config	6/28/2006 3:...	8 bytes	Hidden from Windows API.
\$Txf	6/28/2006 3:...	0 bytes	Hidden from Windows API.
\$TxfLog	6/28/2006 3:...	0 bytes	Hidden from Windows API.
\$TxfLog\$Tops:$T	9/13/2006 6:...	1.00 MB	Hidden from Windows API.
E:\$Extend\$RmMetadata\$Repair	6/28/2006 3:...	0 bytes	Visible in directory index, but not ...
E:\$Extend\$RmMetadata\$Txf	6/28/2006 3:...	0 bytes	Visible in directory index, but not ...
E:\$Extend\$RmMetadata\$TxfLog	6/28/2006 3:...	0 bytes	Visible in directory index, but not ...
E:\$Extend\$RmMetadata\$TxfLog\...	9/13/2006 6:...	0 bytes	Visible in directory index, but not ...
E:\$Extend\$RmMetadata\$TxfLog\...	6/27/2007 4:...	64.00 KB	Visible in directory index, but not ...
E:\$Extend\$RmMetadata\$TxfLog\...	6/27/2007 4:...	10.00 MB	Visible in directory index, but not ...
E:\$Extend\$RmMetadata\$TxfLog\...	9/13/2006 6:...	10.00 MB	Visible in directory index, but not ...

This screenshot shows a GMER scan reporting a huge list of system modifications entirely caused by either the AVAST anti-virus package installed or GMER itself. The GMER executable in this case is named yoh0wrli.exe.

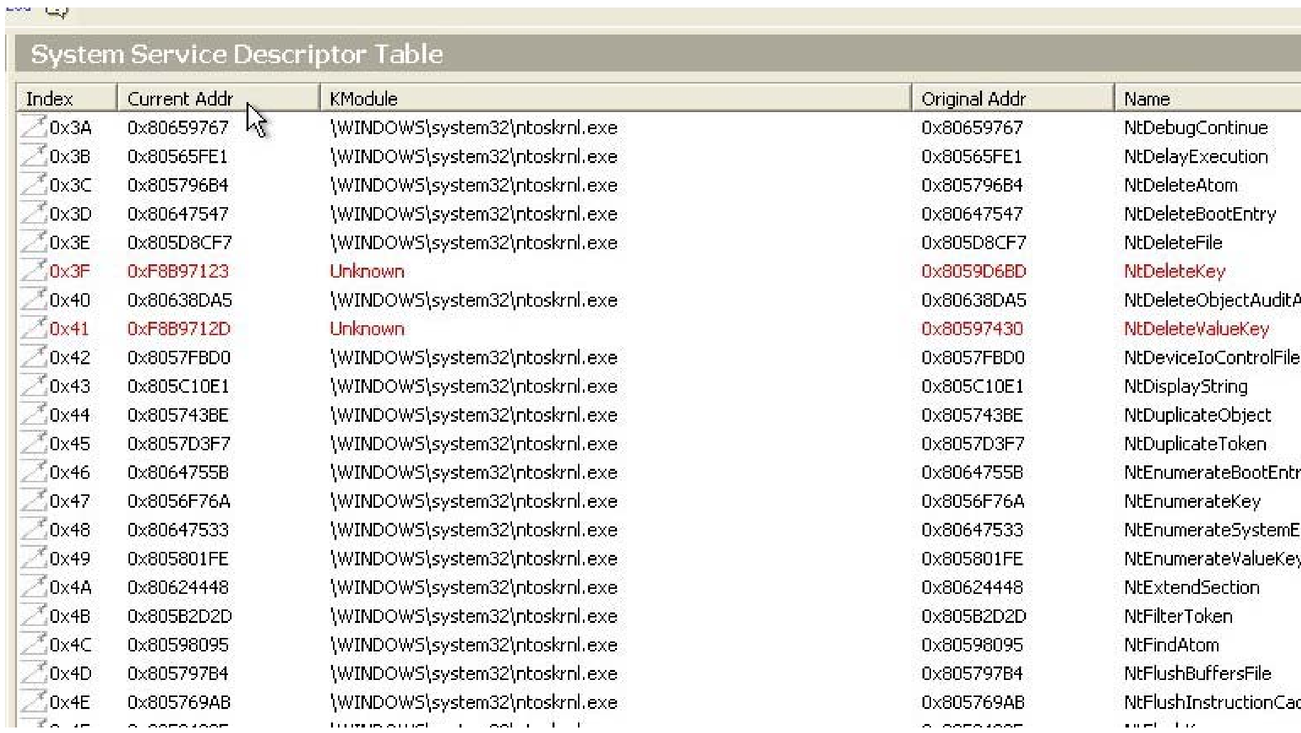


Rootkit/Malware | >>>

Type	Name	Value
SSDT	\SystemRoot\System32\Drivers\aswSP.SYS (avast! self protection module/AVAST Software)	ZwFreeVirtualMemory [0xF7143B18]
SSDT	\SystemRoot\System32\Drivers\aswSP.SYS (avast! self protection module/AVAST Software)	ZwOpenKey [0xF714B82E]
SSDT	\SystemRoot\System32\Drivers\aswSP.SYS (avast! self protection module/AVAST Software)	ZwOpenProcess [0xF714B262]
SSDT	\SystemRoot\System32\Drivers\aswSP.SYS (avast! self protection module/AVAST Software)	ZwOpenThread [0xF714B2C8]
SSDT	\SystemRoot\System32\Drivers\aswSP.SYS (avast! self protection module/AVAST Software)	ZwProtectVirtualMemory [0xF7143BB0]
SSDT	\SystemRoot\System32\Drivers\aswSP.SYS (avast! self protection module/AVAST Software)	ZwQueryValueKey [0xF714B972]
SSDT	\SystemRoot\System32\Drivers\aswSP.SYS (avast! self protection module/AVAST Software)	ZwRenameKey [0xF714BE26]
SSDT	\SystemRoot\System32\Drivers\aswSP.SYS (avast! self protection module/AVAST Software)	ZwRestoreKey [0xF714B930]
SSDT	\SystemRoot\System32\Drivers\aswSP.SYS (avast! self protection module/AVAST Software)	ZwSetValueKey [0xF714BAB4]
Code	\SystemRoot\System32\Drivers\aswSP.SYS (avast! self protection module/AVAST Software)	ZwCreateProcessEx [0xF71588DE]
Code	\SystemRoot\System32\Drivers\aswSP.SYS (avast! self protection module/AVAST Software)	ZwCreateSection [0xF7158702]
Code	\SystemRoot\System32\Drivers\aswSP.SYS (avast! self protection module/AVAST Software)	ZwLoadDriver [0xF715883C]
Code	\SystemRoot\System32\Drivers\aswSP.SYS (avast! self protection module/AVAST Software)	NtCreateSection
.text	ntoskrnl.exe!_abnormal_termination + 313	804E2FE4 1 Byte [72]
.text	C:\Program Files\AVAST Software\Avast\AvastSvc.exe[596] kernel32.dll!SetUnhandledExceptionFilter	7C810386 4 Bytes [C2, 04, 00, 90] {RE
.text	C:\Documents and Settings\user\Desktop\yoh0wrli.exe[1336] ntdll.dll!LdrLoadDll	7C9161CA 5 Bytes JMP 00150030
.text	C:\Documents and Settings\user\Desktop\yoh0wrli.exe[1336] ntdll.dll!LdrUnloadDll	7C91718B 5 Bytes JMP 0015006C
.text	C:\Documents and Settings\user\Desktop\yoh0wrli.exe[1336] ADVAPI32.dll!SetServiceObjectSecurity	77E36BE1 5 Bytes JMP 003C01D4
.text	C:\Documents and Settings\user\Desktop\yoh0wrli.exe[1336] ADVAPI32.dll!ChangeServiceConfigA	77E36CC9 5 Bytes JMP 003C00E4
.text	C:\Documents and Settings\user\Desktop\yoh0wrli.exe[1336] ADVAPI32.dll!ChangeServiceConfigW	77E36E61 5 Bytes JMP 003C0120
.text	C:\Documents and Settings\user\Desktop\yoh0wrli.exe[1336] ADVAPI32.dll!ChangeServiceConfig2A	77E36F61 5 Bytes JMP 003C015C
.text	C:\Documents and Settings\user\Desktop\yoh0wrli.exe[1336] ADVAPI32.dll!ChangeServiceConfig2W	77E36FE9 5 Bytes JMP 003C0198
.text	C:\Documents and Settings\user\Desktop\yoh0wrli.exe[1336] ADVAPI32.dll!CreateServiceA	77E37071 5 Bytes JMP 003C0030
.text	C:\Documents and Settings\user\Desktop\yoh0wrli.exe[1336] ADVAPI32.dll!CreateServiceW	77E37209 5 Bytes JMP 003C006C
.text	C:\Documents and Settings\user\Desktop\yoh0wrli.exe[1336] ADVAPI32.dll!DeleteService	77E37311 5 Bytes JMP 003C00A8
.text	C:\Documents and Settings\user\Desktop\yoh0wrli.exe[1336] USER32.dll!SetWinEventHook	77D6E3D3 5 Bytes JMP 003D0030
.text	C:\Documents and Settings\user\Desktop\yoh0wrli.exe[1336] USER32.dll!UnhookWinEvent	77D6E544 5 Bytes JMP 003D006C
.text	C:\Documents and Settings\user\Desktop\yoh0wrli.exe[1336] USER32.dll!SetWindowsHookExW	77D6E621 5 Bytes JMP 003D00E4
.text	C:\Documents and Settings\user\Desktop\yoh0wrli.exe[1336] USER32.dll!UnhookWindowsHookEx	77D6F29F 5 Bytes JMP 003D0120
.text	C:\Documents and Settings\user\Desktop\yoh0wrli.exe[1336] USER32.dll!SetWindowsHookExA	77D702B2 5 Bytes JMP 003D00A8
IAT	C:\WINDOWS\system32\services.exe[404] @ C:\WINDOWS\system32\services.exe [ADVAPI32.dll!CreateP...	007D0002
IAT	C:\WINDOWS\system32\services.exe[404] @ C:\WINDOWS\system32\services.exe [KERNEL32.dll!Create...	007D0000
Device	\FileSystem\Ntfs \Ntfs	aswSP.SYS (avast! self protection mod
Device	\FileSystem\Fastfat \FatCdrom	aswSP.SYS (avast! self protection mod
AttachedDevice	\Driver\Tcpip \Device\Ip	aswTdi.SYS (avast! TDI Filter Driver/A\
AttachedDevice	\Driver\Tcpip \Device\Tcp	aswTdi.SYS (avast! TDI Filter Driver/A\
AttachedDevice	\Driver\Tcpip \Device\Udp	aswTdi.SYS (avast! TDI Filter Driver/A\
AttachedDevice	\Driver\Tcpip \Device\RawIp	aswTdi.SYS (avast! TDI Filter Driver/A\
Device	\FileSystem\Fastfat \Fat	aswSP.SYS (avast! self protection mod
AttachedDevice	\FileSystem\Fastfat \Fat	fltMgr.sys (Microsoft Filesystem Filter Ma

AVAST has not only hooked IAT and SSDT, but also has created filtered device drivers for the network card, hard drive, and CDROM.

This screenshot shows Icesword reporting an apparently alarming finding that Unknown executable has hooked several important SSDT entries including NtDeleteKey and NtDeleteValueKey. These hooks were created by the Avira anti-virus software which then obfuscated the hook, probably to prevent malware from interfering but making it all the more suspicious looking.



System Service Descriptor Table

Index	Current Addr	KModule	Original Addr	Name
0x3A	0x80659767	\WINDOWS\system32\ntoskrnl.exe	0x80659767	NtDebugContinue
0x3B	0x80565FE1	\WINDOWS\system32\ntoskrnl.exe	0x80565FE1	NtDelayExecution
0x3C	0x805796B4	\WINDOWS\system32\ntoskrnl.exe	0x805796B4	NtDeleteAtom
0x3D	0x80647547	\WINDOWS\system32\ntoskrnl.exe	0x80647547	NtDeleteBootEntry
0x3E	0x805D8CF7	\WINDOWS\system32\ntoskrnl.exe	0x805D8CF7	NtDeleteFile
0x3F	0xF8B97123	Unknown	0x8059D6BD	NtDeleteKey
0x40	0x80638DA5	\WINDOWS\system32\ntoskrnl.exe	0x80638DA5	NtDeleteObjectAuditA
0x41	0xF8B9712D	Unknown	0x80597430	NtDeleteValueKey
0x42	0x8057FBD0	\WINDOWS\system32\ntoskrnl.exe	0x8057FBD0	NtDeviceIoControlFile
0x43	0x805C10E1	\WINDOWS\system32\ntoskrnl.exe	0x805C10E1	NtDisplayString
0x44	0x805743BE	\WINDOWS\system32\ntoskrnl.exe	0x805743BE	NtDuplicateObject
0x45	0x8057D3F7	\WINDOWS\system32\ntoskrnl.exe	0x8057D3F7	NtDuplicateToken
0x46	0x8064755B	\WINDOWS\system32\ntoskrnl.exe	0x8064755B	NtEnumerateBootEntr
0x47	0x8056F76A	\WINDOWS\system32\ntoskrnl.exe	0x8056F76A	NtEnumerateKey
0x48	0x80647533	\WINDOWS\system32\ntoskrnl.exe	0x80647533	NtEnumerateSystemE
0x49	0x805801FE	\WINDOWS\system32\ntoskrnl.exe	0x805801FE	NtEnumerateValueKey
0x4A	0x80624448	\WINDOWS\system32\ntoskrnl.exe	0x80624448	NtExtendSection
0x4B	0x805B2D2D	\WINDOWS\system32\ntoskrnl.exe	0x805B2D2D	NtFilterToken
0x4C	0x80598095	\WINDOWS\system32\ntoskrnl.exe	0x80598095	NtFindAtom
0x4D	0x805797B4	\WINDOWS\system32\ntoskrnl.exe	0x805797B4	NtFlushBuffersFile
0x4E	0x805769AB	\WINDOWS\system32\ntoskrnl.exe	0x805769AB	NtFlushInstructionCac

Eliminating false positives

After having examined all the hooks present in the OS, the investigator should try to eliminate any false positives by examining all the software loaded on the system to determine whether any legitimate applications may have placed the hooks. In some cases it may be possible to simply disable the software being tested temporarily and run another scan. In other cases it will be required to completely uninstall the software to remove all of its hooks. This process should be completed methodically and the system rescanned after each change to see which hooks, if any, disappear. Ideally, a scan of the system in a known clean state would have been done to allow a comparison to be made.

Once all legitimate software which may have hooked the OS has been disabled or removed, the remaining hooks can be assumed to either be part of the OS itself or a rootkit. Research into the OS design will tell whether it has placed its own hooks or not. These Microsoft dlls are known to hook other dlls as part of their normal function: *setupapi.dll, mswsock.dll, sfc_os.dll, adsldpc.dll, advapi32.dll, secur32.dll, ws2_32.dll, iphlpapi.dll, ntdll.dll, kernel32.dll, user32.dll, gdi32.dll*.

Inline code modifications of kernel files are generally extremely suspicious, however, Microsoft has released a set of APIs called "detours" for inline code modifications for use in hot patching live systems without needing to reboot. The changes made by applications using these APIs would show up as hooks in a rootkit scanner. Properly implemented these types of hooks should be temporary and rare, however there is no way to be completely certain whether any given inline kernel modification is malicious or not without examining the memory location referenced by the hook. If you have a tool to enumerate dlls called by processes, such as Process Explorer, you can check to see if *detoured.dll* is listed. If so, this is generally a sign that the Detours API is in use and has hooked the process.

Linux and MacOS also use what's called runtime patching or runtime memory barrier patching which replaces instructions in the .text section of the kernel. Generally this is done to optimize the kernel for the specific instruction set of the CPU without having to compile and release a binary for every type of CPU, though sometimes it is done to apply kernel patches to a system that cannot be rebooted. All runtime changes should be documented in the .altinstructions or .altstr_replace section of the kernel. Any changes not documented there should be considered malicious, but even documented changes may show up on a running kernel modification scan from a tool like elfstat. And ultimately there is nothing stopping rootkit authors from properly documenting modifications to make the rootkit seem more legit.

Examine automatic program execution entries

Rootkits, like any other complex software are generally composed of several interrelated files, which may include device drivers, executables, and dependent dlls. Often there will be dozens of such files, each of which has a specialized function, stored in different folders all over the hard drive. The rootkit needs to get all of them into memory to function properly. This job falls to loader files, which only serve to load the other rootkit components into memory. There may be a half dozen or more distinct loaders, each capable of kickstarting the rootkit in case the others are deleted.

Rootkit authors tend to favor the "belt and suspenders" approach to making sure their rootkit is loaded on boot. Often, despite having loaders specified in:

HKLM\Software\Microsoft\WindowsNT\CurrentVersion\Windows\AppInit_DLLs
HKU\.DEFAULT\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders
HKLM\SYSTEM\CurrentControlSet\Control\SafeBoot\Minimal\

all of which are stealthy places to hide, the rootkit will still have loaders in all the obvious places you would think of to look for malware, like:

HKLM\Software\Microsoft\Windows\CurrentVersion\Run

and user Startup folders, found at

C:\Documents and Settings\%username%\Start Menu\Programs\Startup for XP or *C:\Users\%username%\Start Menu\Programs\Startup* for Vista and later.

Checking common locations for automatic program execution is a good step to take in any investigation, even where something as stealthy as a rootkit is concerned.

Remember to check System Services, Task Scheduler, and for Linux, cron jobs also. The presence of a file with a name consisting of a series of random letters and numbers in any of these places is a dead giveaway. Checking obvious locations for a loader is often a quick and easy way to unmask the presence of a rootkit which may have other more stealthy components elsewhere.

Another quick check which surprisingly often yields results is to check the system directories for hidden files, ie

C*:\Windows\system32*

Hidden files generally show up as "grayed out". Despite being buried in a huge pile of legitimate files which would otherwise make finding any file which didn't belong difficult, rootkits quite often mark their files as hidden as "extra protection", which only serves to make them stand out to an investigator. Very few real OS files are marked hidden in the filesystem, so it is fairly easy to check online whether any hidden files that turn up are legitimate or not. In Linux OS check *lsmod* and */proc/modules* for unknown or suspicious kernel modules.

Memory Scan

Even though there are methods for a rootkit to hide its code in memory, not all rootkits use these techniques, so a good AV or AR program which includes memory scanning is a good step to take, as is using Task Manager or ps to look at running processes for anything that looks suspicious. Examples of suspicious processes would include unusual filenames or applications which should not be running or which should have a visible window but do not. Many rootkits hook the default browser and run it in a hidden context, so if the web browser is shown as running when it is not visible as such, that can be a sign of infection. Most linux distributions using kernel 2.6 or newer enable CONFIG_STRICT_DEVMEM which disables the ability to read physical RAM, which may be required for some rootkit scanning tools.

Open Ports

It is also a good idea to check all the open TCP and UDP ports, using a tool like TCPView or netstat. Even though some rootkits hide network connections, not all do, so it is worthwhile to check. The computer being tested will need to have internet access for any attempts by the rootkit to "phone home" to show up. Close all other programs which may have an active internet connection to more easily spot unauthorized connections. Then do a reverse DNS lookup on any IPs which show up to determine if there is a legitimate reason for that connection.

For rootkits which use hidden connections, having another computer sniffing network traffic using Wireshark and a hub or network shunt can be a useful way to expose a rootkit's communications to its remote command and control server. Rootkits may connect back on any port or protocol. What's more important is the connection end point. A lack of communication should not be construed as no infection, since some rootkits only phone home very infrequently, but unexpected connections are a good indicator of infection.

Taking Notes

Keep meticulous notes of all information uncovered during an investigation. Rootkits are known to behave erratically. A registry entry which points to one of the rootkit files may disappear the next time the registry is examined. Open network connections may be brief and infrequent. Take screenshots where possible and in every case make note of file names and locations, memory offsets, registry entries, IP addresses, and disk sector addresses.

Making A Diagnosis

After all the above steps have been done, make copies of any files which are suspicious and upload them to a multi-AV site such as *virustotal.com* or *novirusthanks.org*. Most rootkits use encryption or other obfuscation techniques and are only likely to have been previously identified by a handful of AV vendors. Running a scan using a large number of AV signature databases is more likely to result in a positive match should any of the files actually belong to a rootkit. In the absence of a direct match in one of the AV databases, a malware sandbox such as Anubis *anubis.iseclab.org* or *camas.comodo.com* may be useful for automated heuristic behavior analysis and comparison to known rootkit profiles. This will often catch variants of popular rootkits that have simply had minor modifications to evade AV.

However, many rootkits monitor the execution environment and will refuse to run in a virtualized or sandboxed environment. In this case the investigator is forced to make an independent evaluation of the heuristic behavior of the computer as to whether it is consistent with an infection. There is no sure standard, but most rootkit infections will exhibit multiple signs, such as hooks, hidden processes, files, and network connections.

Other Traces Of Malicious Activity

A rootkit compromised machine may function as a staging area for the rootkit user to launch additional attacks on other machines on the network. If this is the case, evidence of this activity may be found on the computer hard drive which can point to the underlying rootkit. Tools for malicious activity can be considered a sign of an infection.

Hackers often code attacks in Perl, Ruby, and Python scripts, so support libraries for these programming languages may be an indicator of malicious activity. Network scanners such as Nmap, sniffers such as Wireshark, password crackers such as John the Ripper, and exploit frameworks such as Metaplsoit may also be indicators of malicious activity. Whether they have a legitimate reason to be on the machine will depend on its regular use and role in the network. Logs of other machines on the network, including IDS, which indicate a pattern of malicious activity originating from the suspect machine may also be a sign of infection.

Advanced Rootkit Detection

Even though hypervisor rootkits, memory only rootkits, and BIOS based firmware rootkits have not been found in the wild so far, they cannot be ruled out, particularly as various nation-state actors become involved in the development of targeted malware. The existence of these kinds of advanced rootkits adds a greater element of uncertainty to rootkit detection.

While it is possible to use commands like *dmesg* to tell if virtualization components are loaded in the OS, along with other techniques such as examining the IDT (Interrupt Descriptor Table), if the machine is already known to be running a virtual environment as part of its normal function this gives no information whether there is any additional hypervisor other than the expected one.

Forensic analysis of the hard drive on a known clean system may show signs of a hypervisor rootkit which resides on the hard disk but not one which is only memory resident or any rootkit in hardware flash memory. Hardware based rootkit scanners, such as copilot, may be able to unmask these advanced types of rootkits, but even that may not be able to catch all of them or may itself be vulnerable to compromise. Due to the highly sophisticated nature of the threats, 100% certainty that a rootkit is not present on a system is not possible. Even a brand new computer never before used can be compromised as there have been instances of malware infected software provided direct from the manufacturer.

Rootkit Removal

The most reliable and efficient method of removing a rootkit is to low level format the infected hard drive using manufacturer's software or firmware for that purpose and reload the OS from known good backups. In cases where computer firmware is suspected of compromise, the additional step of re-flashing all BIOS firmware using firmware cryptographically signed by the manufacturer may be necessary. For real certainty, every writable space, including all drives and firmware, would need to be flushed.

If this is impractical, steps may be taken to attempt to manually remove the rootkit piecemeal, however success cannot be guaranteed. The key to a manual rootkit removal is to have accurately and thoroughly mapped out all its functions, hooks, and files. Often a rootkit will be programmed to check whether its hooks and files are intact and replace them if they are modified or deleted. In order to fully remove a rootkit, all its files, hooks, and registry entries must be removed while the computer is offline to prevent the rootkit from detecting the changes and undoing them.

Additionally, any device drivers and kernel files which have been modified by the rootkit will need to be restored from backup as they are critical for the operation of the OS and cannot be simply deleted. It will be crucial when restoring damaged drivers and kernel files to ensure they are the same version as the original. If known good backups are not available, OS files may be restored from the original installation source.

Before attempting a removal, it is advisable to observe the rootkit in operation on a clone drive using advanced debugging tools, such as SoftICE and Ollydbg, which monitors heap and stack, traces registers, recognizes procedures, loops, API calls, switches, tables, constants and strings. This is to make sure that all hidden components are uncovered to the fullest extent possible. However, many rootkits watch memory space for known debuggers and will attempt to confuse the process by shutting down, falsifying data, or terminating the debugger.

It is important to gather as much information as possible before attempting removal because if even one component is missed, the rootkit may still be operable and either recreate deleted components or download them fresh from its control server. It may be necessary to fully reverse engineer the rootkit to determine how to completely remove it.

In cases of an MBR infection, the MBR will need to be overwritten with a clean copy using the *fdisk* utility, *fixmb* or for Linux *grub-install*. In cases of slack space infection, the slack space can be overwritten without damaging the files on disk. This is done with a specialized utility like Eraser or bmap. In fact, if the whole hard drive is not going to be wiped, it is probably a good idea to at least wipe slack space and free space, even if there is no concrete indication the rootkit is storing files there, simply because it doesn't harm the filesystem and there just might be some backup copy of the rootkit there waiting to spring back into action. For cases of ADS infection, a different set of specialized tools will be required to clean them. Some of these tools include: ADSSpy, Streams, and StreamArmor.

If the rootkit has been positively identified by an AV vendor, it may be possible to use that vendor's AV software to remove some or all of the rootkit files automatically. For this reason multi-AV scan sites will be valuable in identifying which AV vendor has detection signatures for the rootkit. In addition, there may be information online or available directly from the AV vendor which more fully describes the operation of the rootkit and exact removal instructions. Even if no AV vendor has signatures for the rootkit, it may still be useful to run an AV scan which includes good heuristic detection, to complement other efforts and make sure nothing is missed.