FEATURE 3

INSIDE SONY'S ROOTKIT

Mark Russinovich
Sysinternals, USA

In late October 2005, as I was performing scans of my computer systems with a test version of *RootkitRevealer* – a rootkit detection tool I had co-authored with Bryce Cogswell – I was stunned to see evidence of the presence of a rootkit on one of my computers.

RootkitRevealer displayed a number of cloaked Registry keys and files and a hidden directory – Figure 1 shows the results. The cloaked objects were not connected in any obvious way to software that I had installed, so I launched an investigation using several tools. Eventually I came to the conclusion that the rootkit had been installed when I had accepted a EULA presented to me by the autorun program on a *Sony* CD I had purchased: *Get Right With the Man*, by Van Zant.

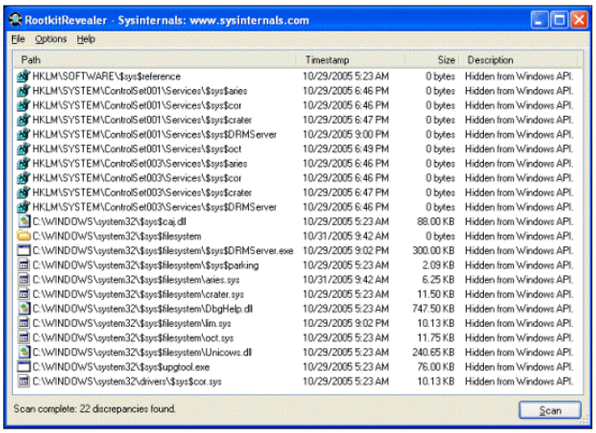

Figure 1: RootkitRevealer results on a system with XCP installed.

I documented the findings in my blog (http://www.sysinternals.com/blog), and a firestorm of criticism aimed at *Sony* ensued. Less than 24 hours after publishing my findings hundreds of comments had been posted to the blog and the story was picked up by *Slashdot.org*.

The furor centred on the fact that *Sony*'s Digital Rights Management (DRM) software, which it had licensed from UK-based *First4Internet*, used the rootkit's cloaking to hide its presence from computer users without asking them explicitly for consent to such behaviour, or even noting it in the EULA.

As the story unfolded more problems came to light, including the 'phone home' behaviour of the player that ships on the CD, possible use of LGPL software in the player, and security problems in the ActiveX control that *Sony* provided originally as its uninstaller. Eventually, *Sony* discontinued the production of CDs using *First4Internet*'s 'XCP' DRM technology, recalled all of the CDs containing XCP, offered existing customers an exchange for non-XCP CDs, and provided a stand-alone executable uninstaller.

There are many interesting angles to this story, but in this article I focus on the XCP rootkit's implementation and discuss the use of rootkits in commercial software.

SYSTEM CALL HOOKING

Aries.sys is the device driver that implements the rootkit functionality of the *First4Internet* DRM software. The CD installation software starts and loads the driver when a user accepts its EULA and configures the driver as an auto-start driver, which means that it loads early each time *Windows* boots.

Under most circumstances, *Windows* presents end users with an acceptance dialog box before installing unsigned drivers. Unsigned drivers have not been subject to reliability testing by *Microsoft*'s Windows Hardware Quality Laboratories (WHQL), so *Microsoft* cannot know what level of testing the drivers have undergone.

Neither Aries nor the other drivers included in the XCP are signed, but *Windows* performs signature checks only when it installs drivers via its Plug and Play system. Since the CD installation configures the XCP hardware-related drivers manually and starts all the XCP drivers, including Aries, using the Windows Service Control Manager's StartService API, no check occurs and no warning is presented. (*Microsoft* is considering implementing a check for a signature in all driver load paths in *Windows Vista.*)

The Aries driver relies on system call hooking, a technique I pioneered with Bryce Cogswell in 1996 with the first

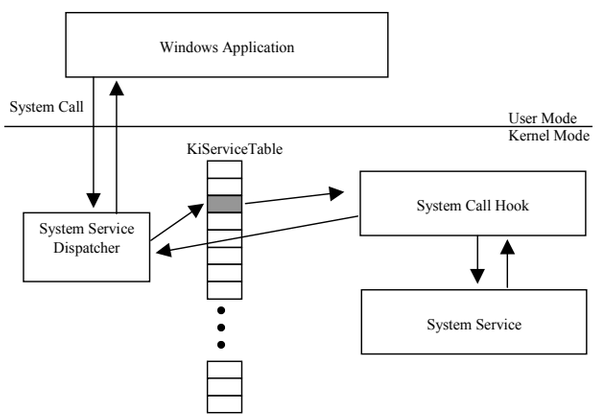

Figure 2: System call hook control flow.

implementation of the Regmon registry monitoring application. When a user-mode process invokes a kernel-mode system service it does so by loading the index number of the service into a processor register and then transitioning to the kernel-mode system service dispatcher, by executing either a software interrupt or a processor-supported system call instruction. The system service dispatcher locates the target kernel-mode service by calling indirectly through the specified index in the system service table, which is identified internally as KiServiceTable (see *Windows Internals* by Russinovich and Solomon for more information).

A system service is hooked when a driver replaces the function pointer in KiServiceTable for the service with a pointer to its own function. Subsequent calls to the service route to the hooking driver, which can examine and manipulate input parameters, invoke the original routine, manipulate output parameters, or even process the service without the use of the original service routine. Figure 2 depicts a system call hook.

The list of functions that Aries hooks when it initializes, along with their *Windows* API wrappers that export the services to user-mode, are shown in Table 1. Modern rootkits can cloak operating system objects ranging from TCP/IP ports to *Windows* services, but a cursory examination of the system services that Aries hooks reveals that it cloaks only file system, Registry, and objects returned by ZwQuerySystemInformation.

System service	Windows API wrapper	Description
ZwCreateFile	CreateFile	Opens a file or directory
ZwQueryDirectoryFile	FindFirstFile, FindNextFile	Lists directory contents
ZwOpenKey	RegOpenKeyEx	Opens a Registry key
ZwEnumerateKey	RegEnumKeyEx	Lists a key's subkeys
ZwQuerySystemInformation		Queries system information

Table 1: System services hooked by Aries.

When I performed my initial investigation of the rootkit I looked for evidence of system-call hooking by examining the system service table using local kernel debugging. Local kernel debugging describes the use of a kernel debugger running on a system to examine the kernel code and data of the same system.

LiveKd, a tool I released on the CD accompanying *Inside Windows 2000* and now available from *Sysinternals*, provides local kernel debugging capability for the standard *Microsoft* kernel debuggers, Kd and Windbg, on *Windows NT 4.0* and higher, and *Microsoft* added local kernel debugging support in the *Windows XP* kernel. Because all system services reside within the core operating system image file, Ntoskrnl.exe, hooking is visible as offsets in the system service table that fall outside this image. Figure 3 shows two easily-identifiable hooks inserted by Aries.

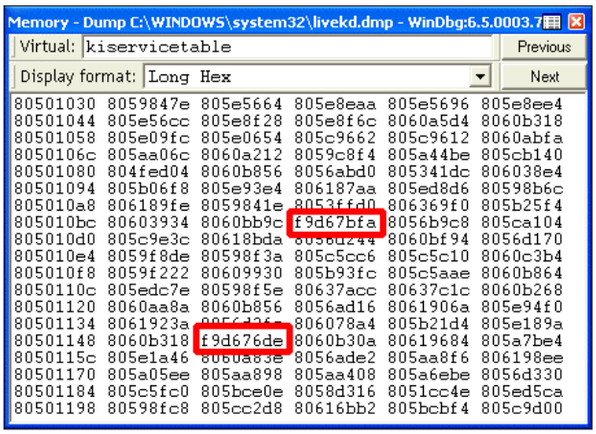

Figure 3: Evidence of Aries hooks.

Until a few years ago we made the source code to Regmon available publicly, which led to the use of our hooking functions and support routines in the NTRootkit example that's published on http://www.rootkit.com/. The structure of the code in Aries indicates that it's likely to be derived from NTRootkit code.

As an aside, system call hooking is prevented by *Windows 64-bit Editions* for the x64 platform through a technology *Microsoft* calls 'patch guard'. Patch guard monitors the system service table and other system structures that are commonly altered by rootkits, and crashes the system when it detects a modification. The alternative to system call hooking on these platforms for Registry operations is the Registry callback interface that the 5.2 version of the kernel introduced; file system filtering is the kernel framework available for monitoring and altering file system operations.

HOOK IMPLEMENTATION

Like most rootkits, Aries cloaks objects for all processes except for those that it considers privileged enough to see an accurate view of the system. The first step of each of the hooks, therefore, is to query the name of the process executing the hooked service.

Windows stores the image name of a process in the executive process block (EPROCESS). To determine the

version-dependent offset of the process name field in the EPROCESS structure the Aries initialization code uses Regmon-based code to search for the name 'System' in the EPROCESS of the System process, which is the process in which it executes.

If the process name is prefixed with the string 'sys', Aries allows execution to proceed unaltered by executing the system service function it hooked and returning the result. The Sony DRM software includes one process, sysDRMService.exe, that receives an unfiltered view of the system.

When a process does not have a privileged name the hook functions simply filter objects that also have names with the 'sys' prefix. The hook for one of the two file-related functions hooked by Aries, ZwCreateFile, returns the *Windows* native error code, STATUS_OBJECT_NOT_FOUND, which translates to the user-mode ERROR_PATH_NOT_FOUND error, for attempted opens of files and directories with such names.

The other file-related service, ZwQueryDirectoryFile, returns a list of child files and directories of a particular directory that match specified search criteria and the Aries hook removes entries with the 'sys' prefix.

The ZwRegEnumerateKey hook behaves in a manner similar to that of ZwQueryDirectoryFile, stripping sys-prefixed keys from the result buffer returned by the underlying service. ZwOpenKey's hook is different, however, because it does not modify the functionality of ZwOpenKey.

It appears that the developer originally intended to model the ZwOpenKey hook on that of ZwCreateFile, but realized that doing so would prevent the Service Control Manager, Services.exe, from opening the keys corresponding to DRM-related services and drivers, such as sysDRMServer, and therefore prevent those services and drivers from loading. The reason that the hook remains is likely to be an oversight.

The final hook function, that for ZwQuerySystemInformation, cares about only one particular type of system query: SystemProcessInformation. This is the query that process diagnostic tools like Task Manager use to obtain a list of active processes. Operating the same way as the other hooks, ZwQuerySystemInformation filters processes that have names starting with 'sys' from the list returned by the kernel function.

The effect of the Aries hooks is to hide the presence of directories, Registry keys and processes that have the 'sys' prefix from any process that doesn't have that prefix in its own name. As I've described, the *Sony* DRM software takes advantage of this behaviour by naming one if its services 'sysDRMServer'. The software consists of several other drivers that it names similarly, and the Registry key name for Aries itself is 'sysaries', so that it is not visible to applications like Regedit and security software when they enumerate the contents of HKLM\System\CurrentControlSet\Services.

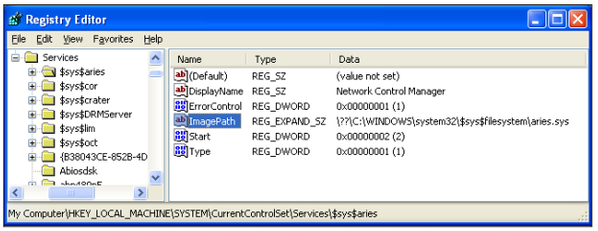

Figure 4: Aries Registry key.

Figure 4 shows the contents of the sysaries key after the Aries driver is disabled and the key is visible.

The Aries driver registers an unload function and takes other steps that suggest that the developer believed a hook-driver can be unloaded safely.

In fact, *Sony*'s decloaking patch unloads the driver while the system is running, which can lead to a crash. Each of the hook functions calls the Plug and Play Manager function IoAcquireRemoveLockEx on entry and IoReleaseRemoveLockEx on exit and the driver's unload function executes the two functions in sequence and then pauses for a short time before exiting. The names of the functions imply that they synchronize safe driver unloading with the I/O system, but in reality they simply increment and decrement a reference count. Standard Plug and Play drivers leverage the reference to unload only when their devices are not in use, but Aries is a non-Plug and Play driver so the counter plays no role in its unload logic.

There's no way for a driver that hooks system calls to guarantee that other threads in the system will not attempt to execute the hook functions after the driver unloads. A race condition exists where, for example, a thread is pre-empted just as it is about to execute the first few instructions of a hook function, the driver unloads, and then the thread executes invalid memory the driver just occupied. This scenario is unlikely, but the Aries unload logic demonstrates an ignorance of both *Windows* device driver and multi-threaded programming.

THE SECURITY AND RELIABILITY OF ARIES

An obvious security problem created as a side effect of the Aries cloak is that other applications can take advantage of its general nature to hide their own objects from end users.

Malware authors released several viruses shortly after I disclosed the existence of Aries that creates objects with the magic 'sys' prefix, and World of Warcraft (WoW) gamers published a way to circumvent the WoW anti-cheat system by hiding executable images behind the Aries cloak.

The media was quick to claim that Aries opens huge security holes, but the fact is that viruses could just as easily deploy their own rootkits instead of piggy-backing on Aries. The security implications of rootkits are complicated and are tied directly to the visibility of the software that they cloak.

In general, I believe that rootkits pose a security risk not because of potential errors in their cloaking, but because the software they cloak is generally completely invisible to systems administrators. Virtually all software has security flaws, but systems administrators can check periodically or use patch-management software to ensure that systems are updated with the latest fixes, or they can uninstall any software that they deem to be a risk. However, there is no advertisement to users of the presence of the *Sony* DRM software by way of auto-updater or bundled uninstall utility, so it can't be patched or uninstalled. As a result, a security problem in any of its components is permanent and undetectable.

The other concern about cloaked kernel-mode code is reliability. Bugs in the Aries driver could impact the stability of *Windows* and lead to system crashes.

My analysis shows that the Aries driver contains at least one bug that can lead to a crash. All but one of the hook functions filter data coming out of the kernel and so can trust the validity of the buffers on which they operate. The hook for ZwCreateFile, however, checks the file name input parameter for the 'sys' prefix and therefore accesses pointers that are passed by applications and potentially invalid. Aries omits validation of the input buffer, however, and so can easily be directed to perform an invalid memory access from kernel-mode, which is a violation that causes the Windows Memory Manager to crash the system. The NtCrash2 program that I wrote in 1997 to stress the input parameter validation of the *Windows* kernel triggers this bug, as can be seen in the bluescreen example shown in Figure 5.

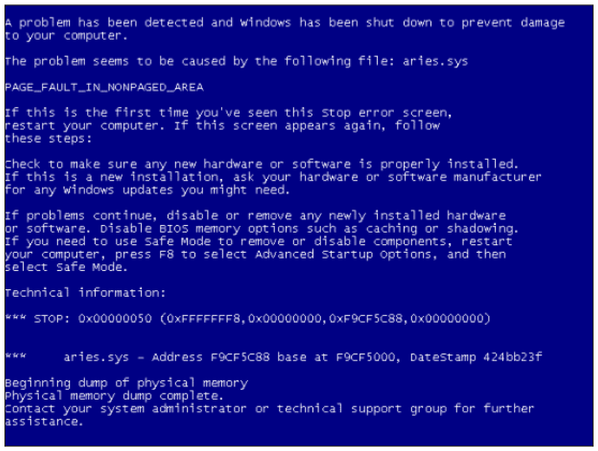

Figure 5: Aries blue screen crash.

The reliability risks caused by rootkits are similar to their security risks. While bugs in user-mode processes manifest as isolated crashes of those processes, bugs in drivers, including Aries and the other XCP drivers, result in *Windows* crashes. Crash analysis of resulting dump files might identify the problematic driver, but because the drivers and the Registry keys that configure them to load are cloaked while *Windows* is online, an administrator has no way of disabling or removing the drivers. If the sequence of execution that triggers a bug is unavoidable then the installation becomes unusable. The XCP software exacerbates this problem by configuring most of its drivers so that they also load in Safe Mode.

COMMERCIAL ROOTKITS

The furor over *Sony*'s rootkit centres on lack of disclosure during the installation process and the rootkit's use in concealing associated software from users. The resulting security and reliability risks only highlight the negative impacts of hidden software. However, are rootkits always bad?

Two other commercial products, both sold by security vendors, utilize rootkit technology: *Symantec*'s *Norton Undelete* and *Kaspersky Antivirus* (*KAV*). *Norton* uses a rootkit to hide the presence of directories in which it stores backup copies of files deleted by end users so that users cannot accidentally delete the backups. *KAV* stores a file's scanning information in an NTFS alternate data stream that it attaches to the file. It hides the streams so that they don't perturb the appearance of files when the files are read by stream-aware applications.

These examples differ greatly from the *Sony* case: *Symantec* and *Kaspersky* use rootkits in a way that is intended to benefit the consumer and not the software vendor. Also, the software that utilizes the rootkit advertises its presence to the user, implements auto-update features, and can easily be uninstalled. However, although the security and reliability risks of these rootkits are minimal, there is still potential for exploitation by the same type of opportunistic malware that uses the Aries cloak. In the end, I believe that the benefits of even benign cloaking are generally outweighed by the potential risks, and that software vendors should use alternative technologies if possible.