WHITE PAPER: SYMANTEC SECURITY RESPONSE

When Malware Meets Rootkits

Elia Florio
Symantec Security Response, Dublin

Originally published by Virus Bulletin, December 2005. Copyright held by Virus Bulletin, Ltd., but is made available free of charge by permission of Virus Bulletin. For more information on Virus Bulletin, please visit http://virusbtn.com/

When Malware Meets Rootkits

Contents

Abstract

There was a time when Windows rootkits were just stand-alone applications, but today it's very common to find advanced rootkit technologies used in worms and Trojans – and sometimes even in non-malicious programs. Although Windows rootkits were introduced only few years ago, the number of programs that currently use stealth technology, or that will use it in the future, is growing very quickly, sometimes with unexpected consequences. This article will not cover all the techniques of rootkits, since the topic is huge. For information on rootkits and how they work on Windows operating systems, refer to [1]. This paper deals only with a specific rootkit technique known as 'DKOM using \Device\PhysicalMemory'. This technique was observed recently in the worm W32/Fanbot.A@mm [2], which spread worldwide in October 2005. The paper will also present some data on rootkit usage in malicious threats.

Introduction – the art of hiding expressed in many forms

Rootkits are usually divided in two categories: user-mode rootkits that work in Ring 3 mode, and kernel-mode rootkits that operate in Ring0. The latter represents a more sophisticated piece of code, which requires a lot of programming knowledge and familiarity with the Windows kernel.

Kernel-mode techniques are very powerful and the most advanced rootkits are able to subvert the Windows kernel [3] and hide files, folders, registry keys, ports and processes. This type of rootkit needs to operate as a system driver to manipulate the kernel because this interaction requires Ring0 privileges, which are not available for normal executables in userland space.

The major drawback of this implementation is that the rootkit always comes with two different binaries (one SYS driver and one EXE that installs the driver) and this fact raises some barriers to the practical integration of this type of threat into real applications. Even if the SYS driver can hide everything (including itself), it needs to keep static structures installed in kernel memory which can be detected [4]. Moreover, the installation process requires interaction with the Windows Service Control Manager (SCM), or alternatively uses the undocumented API ZwSetSystemInformation. Both methods can create some evidence of the threat's presence or can be blocked during the installation phase.

Ghost processes in the system

For these reasons, the next generation of rootkits started to approach the Windows kernel in a different way, avoiding the need for a SYS driver and system hooks. This goal is achieved by mixing the idea introduced by the FU rootkit (known as DKOM, Direct Kernel Object Manipulation) with another technique that involves the manipulation of the \Device\PhysicalMemory object and does not require any additional driver. The method of 'playing' with the physical memory object was imported from the Linux world, where another (in)famous rootkit known as 'SucKIT' [5] is gaining a lot of popularity.

DKOM rootkits are able to manipulate kernel structures and can hide processes and ports, change privileges, and fool the Windows event viewer without many problems. This type of rootkit hides processes by manipulating the list of active processes of the operating system, changing data inside the EPROCESS structures. This method is well documented and was first implemented by the FU rootkit [6].

Essentially, the Windows operating system maintains two different lists of all process and thread information (PID, name, token, etc.). Every process has an associated EPROCESS structure, which is linked to the previous and the following process (double-linked list) using some pointers. Figure 1 shows, with a simplified diagram, how EPROCESS structures are interconnected.

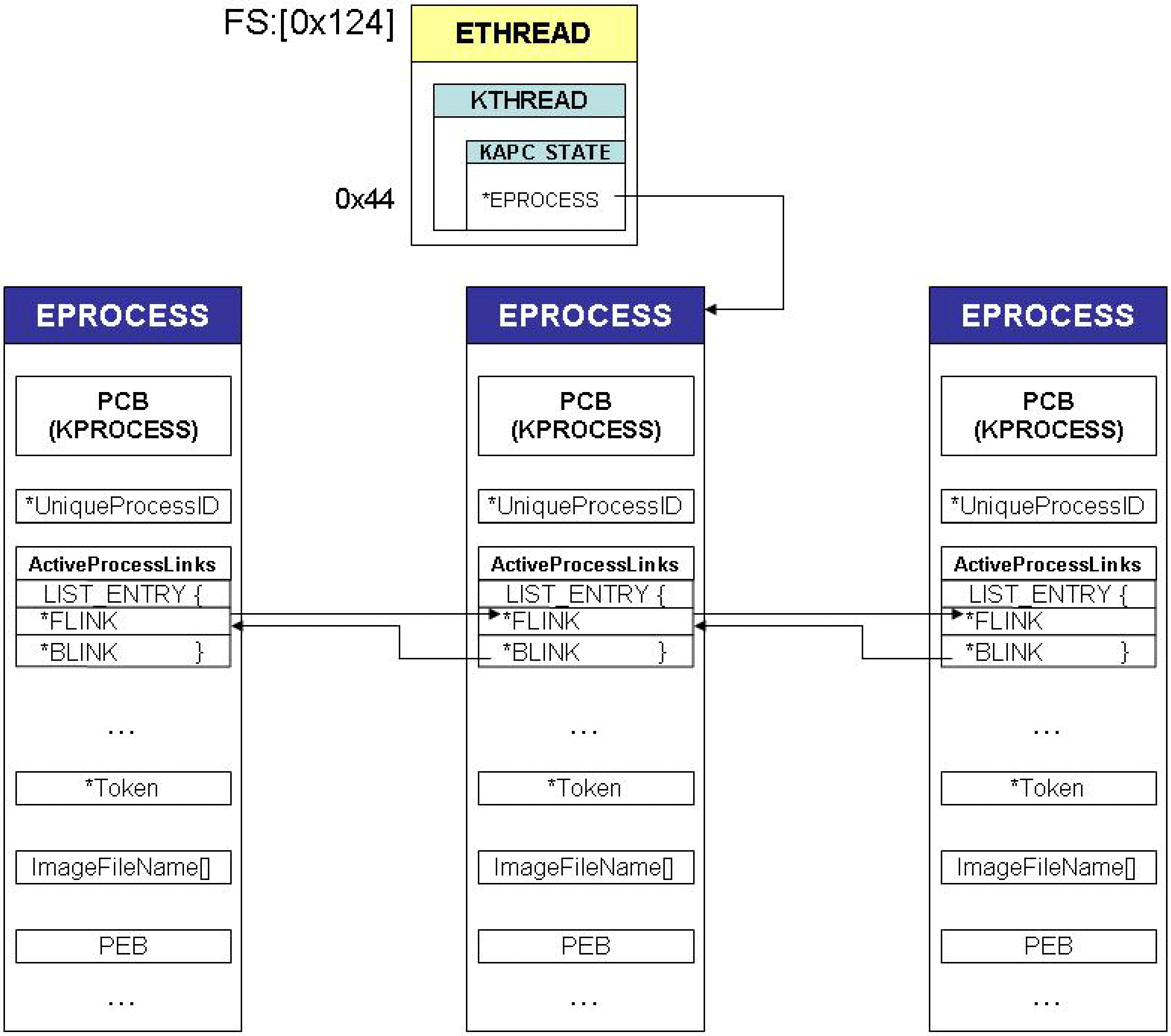


Figure 1: Windows EPROCESS structures are connected to each other by a double-linked list.

However, many people don't realize that processes don't run; only threads run. The Windows operating system uses a pre-emptive, priority-based, round robin method of scheduling threads, swapping the active status from one thread to another (process structures are not involved in the switch).

Considering this fact, DKOM rootkits exploit a very simple trick: they unlink their own EPROCESS from this list, connecting the pointers of the previous and of the next EPROCESS in a way that will skip the 'ghost' process.

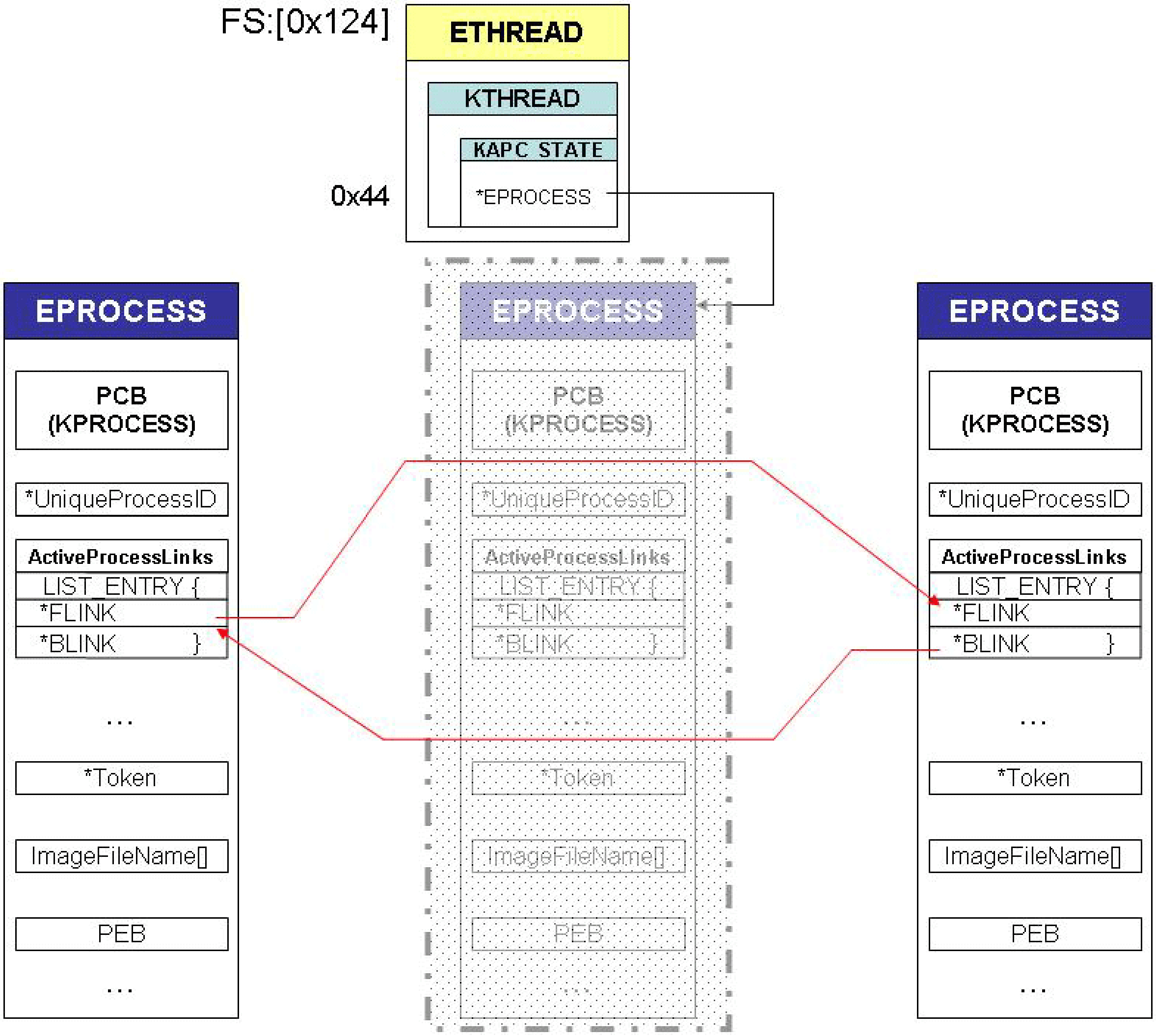


Figure 2: To hide a process the DKOM rootkit simply unlinks it from the list, linking its previous process with the next one. It's just a swap of a few pointers.

With this simple change, a process become invisible to the task manager and other common process manager tools, but it still runs in the system as all its threads are still active. Only advanced tools (e.g. KProcCheck [7]) can detect the presence of the hidden process by traversing the handle table list or the scheduler thread list.

This kind of threat (DKOM rootkit that uses \Device\PhysicalMemory) is quite hard to code because it requires the following abilities:

1. The ability to obtain read/write access to the \Device\PhysicalMemory object.
2. The ability to manipulate the EPROCESS/ETHREAD structure correctly (these structures differ greatly between Windows 2000, XP and 2003).
3. The ability to locate the 'System' process in kernel memory and patch it.
4. The ability to translate the virtual address of a process to a physical address in memory.

While there have been good examples of the first three steps [8] in the past, the last step is the most difficult as the Windows addressing scheme is based on a complex layer of multiple arrays. Contiguous virtual addresses of a process may have different physical addresses mapped into kernel memory [9].

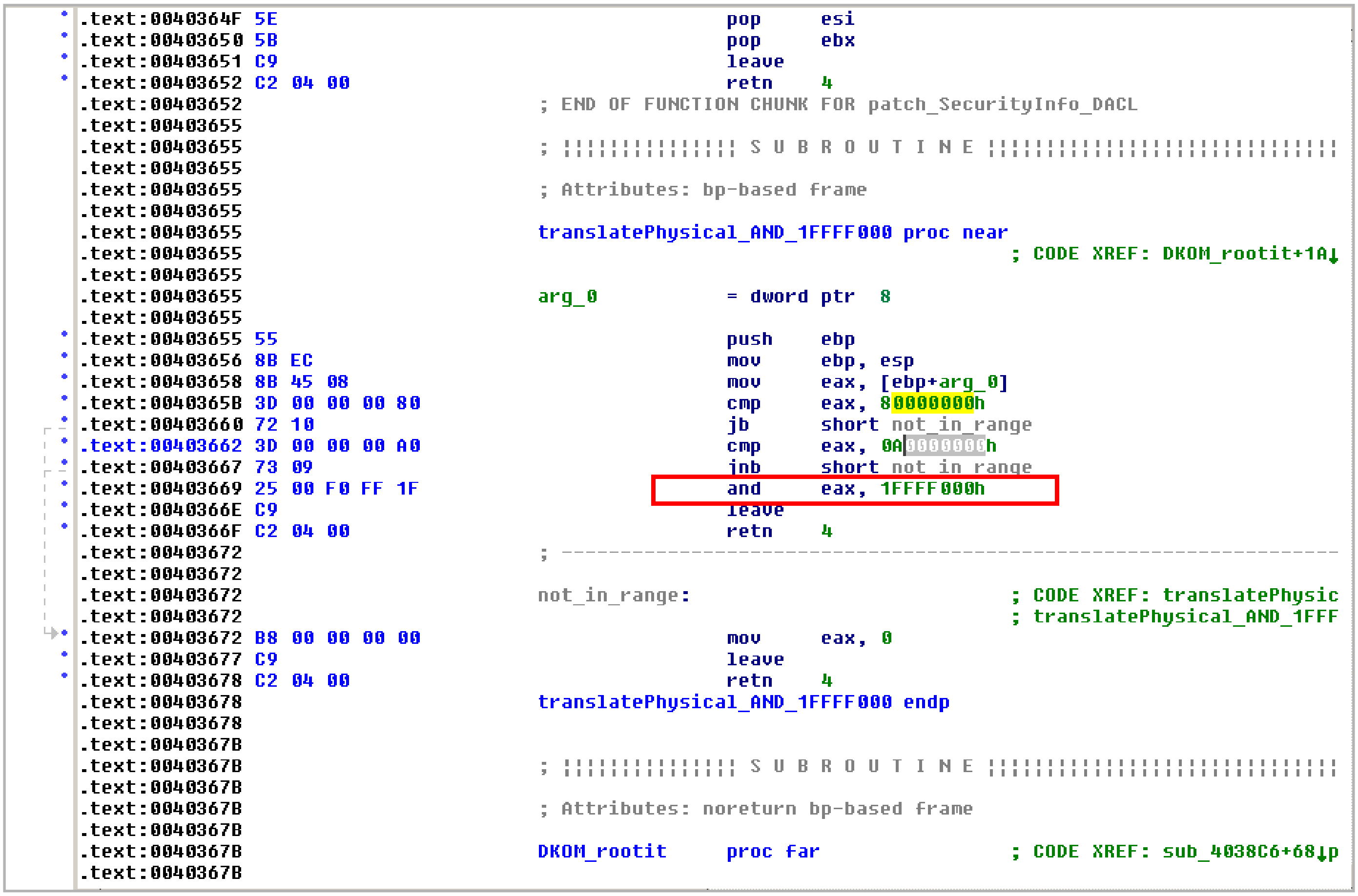
Worms using \Device\PhysicalMemory

It was surprising to find a practical (and well written) implementation of this rootkit technique inside the W32/Fanbot.A@mm code. W32/Fanbot.A is not the only worm that uses the DKOM and \Device\PhysicalMemory technique. The first worm that tried to achieve this was W32/Myfip.H. However, the routine observed in this worm was a little buggy and did not work well under XP and 2003 systems as it used a simplified memory model (the trick introduced in [8]) to map logical addresses to physical addresses.

W32/Myfip.H tried to 'emulate' the kernel API MmGetPhysicalAddress by checking if the virtual address was in the range (0x80000000 – 0xA0000000) and applying to it an AND mask of 0x1FFFF000. However, MmGetPhysicalAddress changes a lot from Windows 2000 to XP, so the correct way to translate the virtual address is to use the page tables of the specific process that owns the virtual address to be translated.

Instead, W32/Fanbot.A implements a good algorithm for address translation that considers the Page Directory and the Page Table (including tests for large pages).

It also follows all the basic memory management rules: it extracts PDindex from the virtual address, gets the correct PDE, locates the corresponding PTE, and finally calculates the correct physical address. The only limitation of the W32/Fanbot.A code is that it does not work on Windows versions with PAE (Page Address Extension), because it makes the assumption of four-byte entries for PD and PT.



```
.text:0040364F 5E                              pop     esi
.text:00403650 5B                              pop     ebx
.text:00403651 C9                              leave
.text:00403652 C2 04 00                        retn    4
.text:00403652                         ; END OF FUNCTION CHUNK FOR patch_SecurityInfo_DACL
.text:00403655
.text:00403655                         ; ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦ S U B R O U T I N E ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦
.text:00403655
.text:00403655                         ; Attributes: bp-based frame
.text:00403655
.text:00403655                         translatePhysical_AND_1FFFF000 proc near
.text:00403655                                                                 ; CODE XREF: DKOM_rootit+1A↓
.text:00403655
.text:00403655                         arg_0           = dword ptr  8
.text:00403655
.text:00403655 55                              push    ebp
.text:00403656 8B EC                           mov     ebp, esp
.text:00403658 8B 45 08                        mov     eax, [ebp+arg_0]
.text:0040365B 3D 00 00 00 80                  cmp     eax, 80000000h
.text:00403660 72 10                           jb      short not_in_range
.text:00403662 3D 00 00 00 A0                  cmp     eax, 0A0000000h
.text:00403667 73 09                           jnb     short not_in_range
.text:00403669 25 00 F0 FF 1F                  and     eax, 1FFFF000h
.text:0040366E C9                              leave
.text:0040366F C2 04 00                        retn    4
.text:00403672                         ; ---------------------------------------------------------------------
.text:00403672
.text:00403672                         not_in_range:                           ; CODE XREF: translatePhysic
.text:00403672                                                                 ; translatePhysical_AND_1FFF
.text:00403672 B8 00 00 00 00                  mov     eax, 0
.text:00403677 C9                              leave
.text:00403678 C2 04 00                        retn    4
.text:00403678                         translatePhysical_AND_1FFFF000 endp
.text:00403678
.text:0040367B
.text:0040367B                         ; ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦ S U B R O U T I N E ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦
.text:0040367B
.text:0040367B                         ; Attributes: noreturn bp-based frame
.text:0040367B
.text:0040367B                         DKOM_rootit     proc far                ; CODE XREF: sub_4038C6+68↓p
.text:0040367B
```

Figure 3: The Myfip.H variant implemented a DKOM routine patching physical memory object, but it uses a simplified translation algorithm for addresses.

A closer look at the rootkit code used by Fanbot

W32/Fanbot.A@mm is a worm that has all the typical mass-mailing techniques. This variant comes packed with NsPack and installs itself as a service. It can spread by email, copy itself into P2P folders, and exploit the universal plug-and-play vulnerability (MS05-039).

Once unpacked (276 KB of code), it's possible to locate the DKOM routine by searching for the Unicode string '\Device\PhysicalMemory' and tracing its reference back to the virtual address 0x40F8A5, where the rootkit code begins. The nice thing (for malware writers) is that the rootkit routine of the worm is written in a modular way so that it can easily be extracted and reused in any other malware.

First, the worm loads the NTDLL.DLL library and gets the APIs that are necessary to operate (RtlInitUnicodeString and ZwOpenSection).

Next, it checks the OS version and uses an interesting technique to locate the PDB (Page Directory Base) of the ‘System’ process. DKOM rootkits need to locate the System process in order to get its PDB (which is necessary for physical address translation). For example, the FU rootkit tries to locate System by iterating all the EPROCESS structures and looking for the ‘System’ string in the name. Other rootkits find the System process by checking UniqueProcessID, because on Microsoft systems the following assumption is usually true:

- Windows NT / 2000 => ‘System’ PID = 8
- Windows XP / 2003 => ‘System’ PID = 4

However, W32/Fanbot.A uses a completely different method: it does not scan for a string or PID – it only checks the OS version and locates the PDB of the System process directly using one of the following offsets (as explained in [10]):

- Windows 2000 => ‘System’ PDB = 0x30000
- Windows XP => ‘System’ PDB = 0x39000

At this stage the worm is ready to open ‘\Device\PhysicalMemory’ using ZwOpenSection. If it fails (usually because the current user has no rights to manipulate this object) then it uses the trick (described by Crazylord) of changing ACLs (adding Read/Write permissions) for the physical memory.

Once the worm has located the System page directory, it reads the PDB from memory and keeps a copy of it for all the address translations. The rootkit routine follows this procedure:

1. Locate the current running ETHREAD structure at 0xFFDFF124 (FS:0x124).
2. From ETHREAD jump to EPROCESS, using the pointer at offset 0x44 of the structure.
3. Read FLINK and BLINK from ActiveProcessLinks of the current EPROCESS structure (these offsets change from 2000 to XP).
4. Unlink the current EPROCESS from the ActiveProcessLinks list by connecting the previous process with the next one (just a swap of a few DWORDS!).

The Fanbot worm works under Windows 2000 and XP because the author implemented all the necessary checks for different OS versions, and because it uses the right offsets to handle the EPROCESS structures correctly, according to the following table:

Table 1: Some important offsets of the EPROCESS structure that change for different Windows versions. After the end of the rootkit routine, the worm executable is completely hidden and disappears from the process list.

	Windows 2000	**Windows XP**	**Windows 2003**
PID offset	0x94	0x9C	0x84
FLINK offset	0xA0	0x88	0x88
BLINK offset	0xA4	0x8C	0x8C

After the end of the rootkit routine, the worm executable is completely hidden and disappears from the process list.

```
.text:0040F626                         var_4           = dword ptr -4
.text:0040F626
.text:0040F626 55                                      push    ebp
.text:0040F627 89 E5                                   mov     ebp, esp
.text:0040F629 83 EC 24                                sub     esp, 24h
.text:0040F62C 56                                      push    esi
.text:0040F62D 57                                      push    edi
.text:0040F62E C7 05 14 9C 42 00 94 00+                mov     ds:VersionInformation.dwOSVersionInfoSize, 148
.text:0040F638 68 14 9C 42 00                          push    offset VersionInformation ; lpVersionInformation
.text:0040F63D E8 32 1C 00 00                          call    GetVersionExA
.text:0040F642 83 3D 18 9C 42 00 05                    cmp     ds:VersionInformation.dwMajorVersion, 5 ;
.text:0040F642                                                                  ; dwMajorVersion=5 for Win2K/XP/.NET
.text:0040F649 74 07                                   jz      short goodVersionForRookit_calculatePDBoffset
.text:0040F64B 31 C0                                   xor     eax, eax
.text:0040F64D E9 ED 00 00 00                          jmp     exit
.text:0040F652                         ; ---------------------------------------------------------------------------
.text:0040F652
.text:0040F652                         goodVersionForRookit_calculatePDBoffset:
.text:0040F652                                                                  ; CODE XREF: getAccessToPhysMem_and_map_PDB_of_SYSTEM+23↑j
.text:0040F652 A1 1C 9C 42 00                          mov     eax, ds:VersionInformation.dwMinorVersion
.text:0040F657 09 C0                                   or      eax, eax         ; dwMinorVersion=0 for WinNT/2K
.text:0040F657                                                                  ; dwMinorVersion=1 for WinXP/.NET
.text:0040F659 74 07                                   jz      short case_WIN2K_PDB@30000
.text:0040F65B 83 F8 01                                cmp     eax, 1
.text:0040F65E 74 0B                                   jz      short case_WINXP_PDB@39000
.text:0040F660 EB 12                                   jmp     short exit_EAX_0
.text:0040F662                         ; ---------------------------------------------------------------------------
.text:0040F662
.text:0040F662                         case_WIN2K_PDB@30000:                    ; CODE XREF: getAccessToPhysMem_and_map_PDB_of_SYSTEM+33↑j
.text:0040F662 C7 45 DC 00 00 03 00                    mov     [ebp+offset_of_PageDirectoryBase], 30000h
.text:0040F669 EB 10                                   jmp     short open_PhysMem
.text:0040F66B                         ; ---------------------------------------------------------------------------
.text:0040F66B
.text:0040F66B                         case_WINXP_PDB@39000:                    ; CODE XREF: getAccessToPhysMem_and_map_PDB_of_SYSTEM+38↑j
.text:0040F66B C7 45 DC 00 90 03 00                    mov     [ebp+offset_of_PageDirectoryBase], 39000h
.text:0040F672 EB 07                                   jmp     short open_PhysMem
.text:0040F674                         ; ---------------------------------------------------------------------------
.text:0040F674
.text:0040F674                         exit_EAX_0:                              ; CODE XREF: getAccessToPhysMem_and_map_PDB_of_SYSTEM+3A↑j
.text:0040F674 31 C0                                   xor     eax, eax
.text:0040F676 E9 C4 00 00 00                          jmp     exit
.text:0040F67B                         ; ---------------------------------------------------------------------------
.text:0040F67B
.text:0040F67B                         open_PhysMem:                            ; CODE XREF: getAccessToPhysMem_and_map_PDB_of_SYSTEM+43↑j
```

Figure 4: The rootkit routine of W32/Fanbot.A worm is able to work under Windows 2000 and XP, as it knows all the correct offsets of several kernel structures.

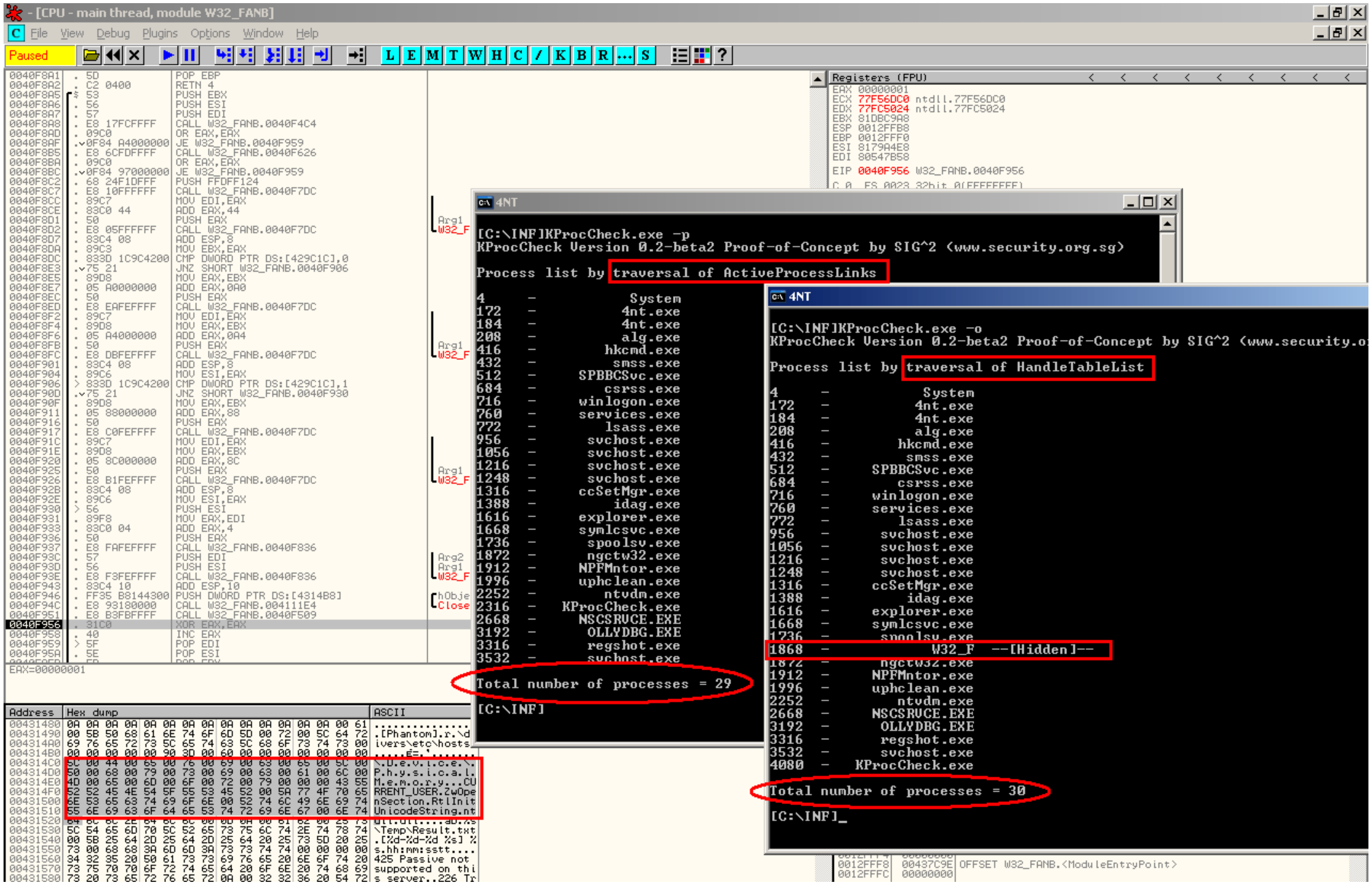
Rootkit technologies in the wild

The recent Sony digital rights management case is evidence of how mature rootkit technology has become a commercial entity ([11] and p.11). This rootkit has caused general consumer uproar as can be seen simply on Amazon's feedback pages for several Sony CDs that ship with the rootkit (see http://www.amazon.com/). But if rootkits have gained this much popularity in the software industry, what's been happening in the 'malware industry'? A process of rootkit integration has already started and many examples of different rootkit techniques can be seen in Trojans, worms, and now also in spyware and adware programs. Malware writers have learned the lesson and they know that the hardest enemy to fight is the one that nobody can see!

Table 2: List of malware and security risks that use rootkit techniques to hide files, processes or registry keys. In some cases it is possible to observe completely different rootkit techniques used by variants of the same family (e.g. Backdoor/Graybird). Some malware, like W32/Loxbot.A@mm, contain a modified copy of FU rootkit (msdirectx.sys) embedded in their code.

Name	**Threat Category**			**Rootkit Characteristics**				
	Worm /Virus	Backdoor /Trojan	Adware/ Spyware	DLL/IAT hooking	SDT/IDT hooking	DKOM	Use SYS driver	Use "Physical Memory"
Adware/Elitebar			X	X				
Adware/CommonName			X		X		X	
Spyware/Search			X		X		X	
Spyware/Elpowkeylogger			X		X		X	
Spyware/Apropos.C			X	X	X		X	
Backdoor/Graybird [a]		X			X		X	
Backdoor/Haxdoor [a]		X			X		X	
Backdoor/Darkmoon [a]		X			X		X	
Backdoor/Berbew [a]		X		X	X		X	
Backdoor/Ryejet [a]		X			X		X	
Trojan/Drivus		X			X		X	
PWSteal/Raidys		X			X		X	
W32/Spybot.NLX	X				X		X	
W32/Theals.A@mm	X			X				
W32/Tdiserv.A	X				X		X	
W32.Mytob.AR@mm					X		X	
W32.Loxbot.A@mm	X				X		X	
W32.Myfip.H@mm	X					X		X
W32.Fanbot.A@mm	X					X		X

a - Data refers to the threat family, not just an individual threat.



```
[C:\INF]KProcCheck.exe -p
KProcCheck Version 0.2-beta2 Proof-of-Concept by SIG^2 (www.security.org.sg)

Process list by traversal of ActiveProcessLinks

4       -              System
172     -             4nt.exe
184     -             4nt.exe
208     -             alg.exe
416     -           hkcmd.exe
432     -            smss.exe
512     -        SPBBCSvc.exe
684     -           csrss.exe
716     -        winlogon.exe
760     -        services.exe
772     -           lsass.exe
956     -         svchost.exe
1056    -         svchost.exe
1216    -         svchost.exe
1248    -         svchost.exe
1316    -        ccSetMgr.exe
1388    -            idag.exe
1616    -        explorer.exe
1668    -        symlcsvc.exe
1736    -         spoolsv.exe
1872    -         ngctw32.exe
1912    -        NPFMntor.exe
1996    -        uphclean.exe
2252    -           ntvdm.exe
2316    -      KProcCheck.exe
2668    -        NSCSRVCE.EXE
3192    -         OLLYDBG.EXE
3316    -         regshot.exe
3532    -         svchost.exe

Total number of processes = 29

[C:\INF]
```

```
[C:\INF]KProcCheck.exe -o
KProcCheck Version 0.2-beta2 Proof-of-Concept by SIG^2 (www.security.o

Process list by traversal of HandleTableList

4       -              System
172     -             4nt.exe
184     -             4nt.exe
208     -             alg.exe
416     -           hkcmd.exe
432     -            smss.exe
512     -        SPBBCSvc.exe
684     -           csrss.exe
716     -        winlogon.exe
760     -        services.exe
772     -           lsass.exe
956     -         svchost.exe
1056    -         svchost.exe
1216    -         svchost.exe
1248    -         svchost.exe
1316    -        ccSetMgr.exe
1388    -            idag.exe
1616    -        explorer.exe
1668    -        symlcsvc.exe
1736    -         spoolsv.exe
1868    -               W32_F   --[Hidden]--
1872    -         ngctw32.exe
1912    -        NPFMntor.exe
1996    -        uphclean.exe
2252    -           ntvdm.exe
2668    -        NSCSRVCE.EXE
3192    -         OLLYDBG.EXE
3316    -         regshot.exe
3532    -         svchost.exe
4080    -      KProcCheck.exe

Total number of processes = 30

[C:\INF]_
```

Figure 5: Fanbot process totally disappears from task list and cannot be detected (and killed) using standard tools.

References

[1] Patrick Runald, ‘The trouble with rootkits’, Virus Bulletin, September 2005, p.4.
[2] Description of W32/Fanbot.A@mm, http://securityresponse.symantec. com/avcenter/venc/data/w32.fanbot.a@mm.html.
[3] Greg Hoglund and Jamie Butler, Rootkits: Subverting the Windows Kernel, Addison-Wesley Professional, 2005.
[4] ‘modGREPER’, a hidden module detector created by Joanna Rutkowska, http://invisiblethings.org/tools/ modGREPER/modGREPER-0.2-bin.zip.
[5] Sd and Devik, ‘Linux on-the-fly kernel patching without LKM’, Phrack #58, Article 7, http://www.phrack.org/show.php?p=58&a=7.
[6] FU Rootkit, http://www.rootkit.com/ project.php?id=12.
[7] ‘Win2K Kernel Hidden Process/Module Checker’ KprocCheck by SIG>2 http://www.security.org.sg/code/kproccheck.html.
[8] Crazylord, ‘Playing with Windows /dev/(k)mem’, Phrack #58, Article 16, http://www.phrack.org/show.php?p=59&a=16.
[9] Pankaj Garg, ‘Windows Memory Management’, http://www.intellectualheaven.com/Articles/WinMM.pdf.
[10] Mark Russinovich and David Solomon, Microsoft Windows Internals, Fourth Edition, Microsoft Press, 2004.
[11] XCP (eXtended Copy Protection), the digital audio protection used by Sony which makes use of rootkit technology http://www.xcp-aurora.com/.

About the Author

Elia Florio is a software engineer with the Symantec Security Response team, based in Dublin, Ireland. Elia graduated from the University of Calabria (UNICAL), Italy with a Bachelor of Computer Engineering in 2003. Elia previously worked for Value Partner and for Accenture on a variety of projects, including security-related consulting. Elia has written several articles for industry magazines and has contributed to a number of vulnerability announcements.

About Symantec

Symantec is the global leader in information security, providing a broad range of software, appliances, and services designed to help individuals, small and mid-sized businesses, and large enterprises secure and manage their IT infrastructure. Symantec's Norton™ brand of products is the worldwide leader in consumer security and problem-solving solutions. Headquartered in Cupertino, California, Symantec has operations in 35 countries. More information is available at www.symantec.com.

Symantec has worldwide operations in 35 countries. For specific country offices and contact numbers, please visit our Web site. For product information in the U.S., call toll-free 1 800 745 6054.

Symantec Corporation
World Headquarters
20330 Stevens Creek Boulevard
Cupertino, CA 95014 USA
408 517 8000
800 721 3934
www.symantec.com