Derek Soeder
Ryan Permeh

BLACK HAT BRIEFINGS

eEye BootRoot

This presentation will cover the eEye BootRoot project, an exploration of technology that boot sector code can use to subvert the Windows NT-family kernel and retain the potential for execution, even after Windows startup—a topic made apropos by the recent emergence of Windows rootkits into mainstream awareness. We will provide some brief but technical background on the Windows startup process, then discuss BootRoot and related technology, including a little-known stealth technique for low-level disk access. Finally, we will demonstrate the proof-of-concept BootRootKit, loaded from a variety of bootable media.

Derek Soeder *is a Software Engineer and after-hours researcher at eEye Digital Security. In addition to participating in the ongoing development of eEye's Retina Network Security Scanner product, Derek has also produced a number of internal technologies and is responsible for the discovery of multiple serious security vulnerabilities. His main areas of interest include operating system internals and machine code-level manipulation.*

Ryan Permeh *is a Senior Software Engineer at eEye Digital Security. He focuses mainly on the Retina and SecureIIS product lines. He has worked in the porting of nmap and libnet to Windows, as well as helping with disassembly and reverse engineering, and exploitation efforts within the eEye research team.*

eEye Digital Security®

eEye BootRoot:
A Basis for Bootstrap-Based Windows Kernel Code

Derek Soeder, Software Engineer
Ryan Permeh, Senior Software Engineer

Introduction

- **Explores the capabilities of custom boot sector code on NT-family Windows**
  - What can it do? Anything – it's privileged code on the CPU
  - The trick is keeping control while allowing the OS to function
- **Overview**
  - BIOS boot process and Windows startup
  - eEye BootRoot: how it works, capabilities and shortcomings
  - Demo: eEye BootRootKit backdoor
- **Required Knowledge**
  - x86 real and protected modes, some Windows kernel

Booting Up

BIOS Handoff to Bootstrap Code

Booting Up – Summary

- **BIOS transfers execution to code from some other medium**
  - Disk drive (fixed or removable)
  - CD-ROM
  - Network boot
- **Windows startup from a hard drive installation**
  - Hard drive Master Boot Record
  - Windows bootstrap loader
  - NTLDR
  - OSLOADER.EXE
  - NTDETECT.COM
  - NTOSKRNL.EXE, HAL.DLL, boot drivers

Booting Up – Disk Drive

- **BIOS loads first sector of drive (200h bytes) at 0000h:7C00h**
  - Executes in real mode
  - SS:SP < 0000h:0400h, DS = 0040h (BIOS data area)
- **For hard drives, the first sector is the Master Boot Record**
  - Copies itself to 0000h:0600h
  - Locates a bootable partition in the partition table
  - Executes the first sector of the boot partition at 0000h:7C00h
- **Partition boot sector is always part of the operating system**
  - Loads and executes the next boot stage of the OS

Booting Up – MBR Partition Table

Master Boot Record Layout

```
0000  xx xx xx xx xx xx xx xx-xx xx xx xx xx xx xx
0010  xx xx xx xx xx xx xx xx-xx xx xx xx xx xx xx
 ...
01B0  xx xx xx xx xx xx xx xx-xx xx xx xx xx xx BI SH
01C0  SS SC ID EH ES EC L0 L1-L2 L3 S0 S1 S2 S3 BI SH
01D0  SS SC ID EH ES EC L0 L1-L2 L3 S0 S1 S2 S3 BI SH
01E0  SS SC ID EH ES EC L0 L1-L2 L3 S0 S1 S2 S3 BI SH
01F0  SS SC ID EH ES EC L0 L1-L2 L3 S0 S1 S2 S3 55 AA
```

- Partition 1 (offset 01BEh)
- Partition 2 (offset 01CEh)
- Partition 3 (offset 01DEh)
- Partition 4 (offset 01EEh)

Partition Table Entry Format

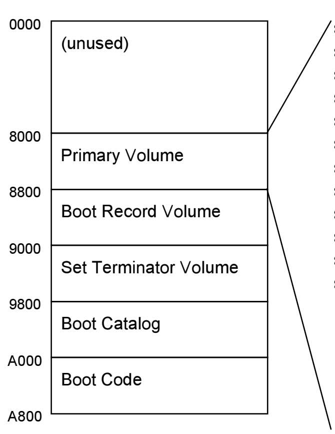
```
+00  BYTE   Boot Indicator
             -- bit 7: partition bootable
+01  BYTE   Starting Head
+02  BYTE   Starting Sector / Cylinder
             -- bits 5..0: sector
             -- bits 7..6: cylinder (bits 9..8)
+03  BYTE   Starting Cylinder (bits 7..0)
+04  BYTE   System ID (volume type)
+05  BYTE   Ending Head
+06  BYTE   Ending Sector / Cylinder
             -- bits 5..0: sector
             -- bits 7..6: cylinder (bits 9..8)
+07  BYTE   Ending Cylinder (bits 7..0)
+08  DWORD  Linear sector number of partition
+0C  DWORD  Size in sectors of partition
```

Source: *NTFS.com Hard Drive Partition - Partition Table.*
http://www.ntfs.com/partition-table.htm

Booting Up – CD-ROM

- **Differences from disks and diskettes**
  - Sector size is 800h bytes (2KB)
  - Data format is more complicated (ECMA-119 / ISO 9660)
  - Bootable CD format dictated by “El Torito” Specification
- **Boot sector (only first 200h bytes) loads at 07C0h:0000h**
  - Executes in real mode
  - SS:SP = 0000h:0400h, DS = 0040h (BIOS data area)
- **Additional disc contents are accessed via INT 13h**
  - Boot catalog entry indicates “emulation mode” (floppy or HD)

Booting Up – Bootable CD Layout (1)




```
8000  BYTE   Volume Descriptor Type = 1
8001  [ 5]   Standard Identifier = "CD001"
8006  BYTE   Volume Descriptor Version = 1
8050  DWORD  Volume Space Size (sectors) = 15h
8054  DWORD  Volume Space Size (big-endian)
8078  WORD   Volume Set Size = 1
807A  WORD   Volume Set Size (big-endian)
807C  WORD   Volume Sequence Number = 1
807E  WORD   Volume Sequence Number (big-endian)
8080  WORD   Logical Block Size = 0800h
8082  WORD   Logical Block Size (big-endian)
809C  [ 22h] Directory Record for Root Directory
  809C  BYTE   Length of Directory Record = 1
  80B5  BYTE   File Flags = 02h (Directory)
  80B8  WORD   Volume Sequence Number = 1
  80BA  WORD   Volume Sequence Number (big-endian)
  80BB  BYTE   Length of File Identifier = 1
  80BC  [ 1]   File Identifier = { 0}
```

Source: *ECMA-119: Volume and File Structure of CDROM for Information Interchange*.
http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-119.pdf

Source: *“El Torito” Bootable CD-ROM Format Specification, Version 1.0*.
http://www.phoenix.com/NR/rdonlyres/98D3219C-9CC9-4DF5-B496-A286D893E36A/0/specscdrom.pdf

Booting Up – Bootable CD Layout (2)

0000	(unused)
8000	Primary Volume
8800	Boot Record Volume
9000	Set Terminator Volume
9800	Boot Catalog
A000	Boot Code
A800	

```
8800  BYTE   Volume Descriptor Type = 0
8801  [5]    Standard Identifier = "CD001"
8806  BYTE   Volume Descriptor Version = 1
8807  [20h]  Boot System Identifier =
             "EL TORITO SPECIFICATION", {0}
8847  DWORD  Pointer to First Sector of Boot Catalog
```

```
9000  BYTE   Volume Descriptor Type = 0FFh
9001  [5]    Standard Identifier = "CD001"
9006  BYTE   Volume Descriptor Version = 1
```

Source: *ECMA-119: Volume and File Structure of CDROM for Information Interchange*.
http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-119.pdf

Source: *"El Torito" Bootable CD-ROM Format Specification, Version 1.0*.
http://www.phoenix.com/NR/rdonlyres/98D3219C-9CC9-4DF5-B496-A286D893E36A/0/specscdrom.pdf

Booting Up – Bootable CD Layout (3)

0000	(unused)
8000	Primary Volume
8800	Boot Record Volume
9000	Set Terminator Volume
9800	Boot Catalog
A000	Boot Code
A800	

```
9800  [20h]  Validation Entry
  9800  BYTE   Header ID = 1
  9801  BYTE   Platform ID = 0
  981C  WORD   Checksum = 55AAh
  981E  WORD   Key = AA55h
9820  [20h]  Initial/Default Entry
  9820  BYTE   Boot Indicator = 88h
  9821  BYTE   Boot Media Type = 2 (1.44MB floppy)
  9822  WORD   Load Segment = 0
  9824  BYTE   System Type = 0
  9826  WORD   Sector Count (virtual sectors) = 1
  9828  DWORD  Load RBA (sector) = 14h
```

Source: *ECMA-119: Volume and File Structure of CDROM for Information Interchange*.
http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-119.pdf

Source: *"El Torito" Bootable CD-ROM Format Specification, Version 1.0*.
http://www.phoenix.com/NR/rdonlyres/98D3219C-9CC9-4DF5-B496-A286D893E36A/0/specscdrom.pdf

Booting Up – Network Boot

- **PXE: Preboot eXecution Environment**
  - Network boot via BOOTP (basis for DHCP) and TFTP
  - BIOS PXE boot agent requests configuration over BOOTP
    - Requires an IP address, server's IP address, and boot file name
    - BOOTP server receives on UDP/67, client on UDP/68
  - Downloads boot file from TFTP service on server
    - TFTP server receives on UDP/69
- **Executes boot file in real mode at 0000h:7C00h**
  - Up to ~500KB of data will be downloaded and stored there
  - Register values should be considered undefined

Booting Up – Network Boot Traffic Example


```
Client IP        Port   Packet               Server IP        Port
0.0.0.0          68     DHCP Discovery    -> 255.255.255.255  67
255.255.255.255  68  <- DHCP Offer           (server IP)      67
  [Server Identifier = (server IP); Boot File Name = "..."]
0.0.0.0          68     DHCP Request      -> 255.255.255.255  67
255.255.255.255  68  <- DHCP Ack             (server IP)      67
  [Server Identifier = (server IP); Boot File Name = "..."]

(client IP)      (var)  TFTP Read Req     -> (server IP)      69
  [File: (boot file name); Mode: "octet"; "tsize" = 0; "blksize" = (block size)]
(client IP)      (var) <- TFTP Option ACK    (server IP)      69
  ["tsize" = (size of boot file); "blksize" = (supported block size)]
(client IP)      (var)  TFTP ACK          -> (server IP)      69
  [Block: 0]
(client IP)      (var) <- TFTP Data          (server IP)      69
  [Block: 1; file data]
(client IP)      (var)  TFTP ACK          -> (server IP)      69
  [Block: 1]
...
```

Windows Startup

Windows Boot Sector to NTOSKRNL Execution

Windows Startup – Boot Loader

- **Windows partition boot sector**
  - Loads first 16 sectors (itself is first) at 0D00h:0000h
  - Uses IBM/MS INT 13h Extensions if available
  - Passes execution to next stage of Windows boot loader
- **Windows boot loader**
  - Loads and executes NTLDR at 2000h:0000h in real mode
  - Does not export any functionality to NTLDR
  - Only uses ~40% of its allotted 8KB (room for our code?)

Windows Startup – NTLDR

- **Enters 16-bit protected mode**
  - Creates GDT and IDT for use throughout Windows startup
  - Wraps real mode BIOS interrupt functionality that subsequent protected mode startup code will invoke:
    - INT 10h: Video
    - INT 13h: Disk
    - INT 14h: Serial
    - INT 15h: System Configuration, Power Management
    - INT 16h: Keyboard
    - INT 19h: Reboot
    - INT 1Ah: Clock (Date and Time)
- **Maps OSLOADER.EXE at its preferred image base**
  - OSLOADER.EXE is a PE image embedded in NTLDR
  - No MZ header or PE signature prior to Windows 2003
  - NTLDR executes its entry point in 32-bit protected mode

Windows Startup – NTLDR GDT


```
#0008: Limit=FFFFFFFF Base=00000000 DPL=0 P=1 A=0 Code32    KGDT_R0_CODE
#0010: Limit=FFFFFFFF Base=00000000 DPL=0 P=1 A=0 Data32    KGDT_R0_DATA
#0018: Limit=FFFFFFFF Base=00000000 DPL=3 P=1 A=0 Code32    KGDT_R3_CODE
#0020: Limit=FFFFFFFF Base=00000000 DPL=3 P=1 A=0 Data32    KGDT_R3_DATA
#0028: Limit=00000077 Base=00024460 DPL=0 Task Gate         KGDT_TSS
#0030: Limit=00001000 Base=00000000 DPL=0 P=1 A=0 Data32    KGDT_R0_PCR
#0038: Limit=00000FFF Base=00000000 DPL=3 P=1 A=1 Data32    KGDT_R3_TEB
#0040: Limit=0000FFFF Base=00000400 DPL=3 P=1 A=0 Data16    KGDT_VDM_TILE
#0048: (reserved)                                           KGDT_LDT
#0050: Limit=0000006F Base=00023B7E DPL=0 Task Gate         KGDT_DF_TSS
#0058: Limit=0000FFFF Base=00020000 DPL=0 P=1 A=0 Code16    (NTLDR code)
#0060: Limit=0000FFFF Base=00022F30 DPL=0 P=1 A=0 Data16    (NTLDR data)
#0068: Limit=00003FFF Base=000B8000 DPL=0 P=1 A=0 Data16    (text memory)
#0070: Limit=00003FFF Base=FFFF7000 DPL=0 P=1 A=0 Data16
#0078: Limit=0000FFFF Base=80400000 DPL=0 P=1 A=0 Data16    (NTOSKRNL code)
#0080: Limit=0000FFFF Base=80400000 DPL=0 P=1 A=0 Data16    (NTOSKRNL data)
#0088: Limit=00000000 Base=00000000 DPL=0 P=1 A=0 Data16
```

Windows Startup – OSLOADER.EXE (1)

17

- **OSLOADER.EXE loads the operating system**
  - Processes \BOOT.INI
  - Executes NTDETECT.COM in real mode at 1000h:0000h
  - Enables paging
    - Applies /3GB BOOT.INI option
    - Sets typical virtual addresses for GDT, IDT, and page tables
  - Loads HAL.DLL and NTOSKRNL.EXE, and any import dependencies (BOOTVID.DLL), at their preferred virtual addresses, and applies relocations
  - Loads the registry (system32\config\system)
  - Loads NLS code pages and required fonts

Windows Startup – OSLOADER.EXE (2)

18

- **OSLOADER.EXE loads boot drivers**
  - Loads drivers with a Start type of Boot (0)
    - Creates a PsLoadedModuleList-format list (*_BlLoaderBlock)
    - Does not realign image sections prior to Windows 2003: in-memory image is the raw file contents!
  - Drivers do not execute at this stage
- **Transfers execution to NTOSKRNL.EXE entry point**

Windows Startup – NTOSKRNL.EXE

19

- **NTOSKRNL and HAL.DLL finish initializing machine state**
  - NTOSKRNL assumes control of TSS, IDT, and GDT
  - Initializes processor(s) and ABIOS support
- **Kernel subsystems initialize in two passes or "phases"**
  - Phase 0 initialization
    - KiSystemStartup calls KiInitializeKernel, which calls ExpInitializeExecutive
  - Phase 1 initialization
    - Phase1Initialization executes as a separate system thread
    - Boot drivers execute during this phase
    - Finishes kernel initialization and starts user-mode SMSS.EXE
  - "Phase 2" mostly deals with licensing (ExInitSystemPhase2)

Windows Startup – Phase 0 Initialization

20

- **NTOSKRNL.EXE!KiSystemStartup**
  - HAL.DLL!HalInitializeProcessor
  - KiInitializeKernel
    - KiInitSystem (*initializes* _KeServiceDescriptorTable *and* _KeServiceDescriptorTableShadow)
    - KeInitializeProcess (_KiIdleProcess), KeInitializeThread (P0BootThread)
    - ExpInitializeExecutive
      - HAL.DLL!HalInitSystem
      - ExInitSystem
      - MmInitSystem (0)
      - ObInitSystem
      - SeInitSystem
      - PsInitSystem (*creates* _PsInitialSystemProcess *and* Phase1Initialization *thread*)
      - PpInitSystem

Windows Startup – Phase 1 Initialization

21

- **NTOSKRNL.EXE!Phase1Initialization**
  - HAL.DLL!HalInitSystem
  - PoInitSystem (0)
  - ObInitSystem
  - ExInitSystem
  - KeInitSystem
  - SeInitSystem
  - MmInitSystem (1)
  - CmInitSystem
  - FsRtlInitSystem
  - PpInitSystem
  - LpcInitSystem
  - ExInitSystemPhase2
  - IoInitSystem (IopInitializeSystemDrivers *runs boot drivers,* PsLocateSystemDll *loads NTDLL.DLL*)
  - MmInitSystem (2) (*makes executive pageable*)
  - PoInitSystem (1)
  - PsInitSystem (*locates certain NTDLL exports*)

eEye BootRoot

22

Technology for Windows Kernel Pre-Subversion

eEye BootRoot – The Problem

23

- **We execute after the BIOS but before the operating system**
- **Advantages**
  - Our code is privileged – real mode is "ring 0"
  - We can control all subsequent code execution
- **Disadvantages**
  - No part of the operating system is loaded yet
  - We need the system to function normally, except with a few of our own "adjustments"
  - OS startup will bring about dramatic machine state changes

eEye Digital Security

eEye BootRoot – Playing Field

24

- **Real mode environment features**
  - Interrupt Vector Table (100h doublewords at 0000h:0000h)
    - Hooking BIOS interrupt services is like hooking APIs
  - BIOS data area (100h bytes at 0040h:0000h)
    - See Ralf Brown's MEMORY.LST for more information
  - 640KB conventional memory
- **CPU and hardware settings**
  - CR*n*, DR*n*, GDTR, IDTR, MSRs, etc.
  - Chipset: e.g., Programmable Interrupt Controller
  - *Any* hardware device
  - Other processors...?

eEye Digital Security

eYe BootRoot – Game Plan

25

- **Windows startup will assume exclusive control over *almost* every facet of machine state...**
  - CPU state, IRQs, chipset, eventually most hardware
  - Eventually all other CPUs in a multiprocessor system
  - Unused memory
- **...But its weakness is reliance upon the BIOS**
  - It uses BIOS interrupts, so IVT is mostly preserved
  - It has to respect memory ranges reserved by BIOS

We can exploit this trust to function like a BIOS "hook"

eEye BootRoot – Our Solution

26

- **"Go resident" – reserve memory for a copy of our code**
  - Reduce conventional memory KB reported by 0040h:0013h
    - Boot virii have used this technique forever
- **Hook INT 13h (Disk) to "patch" OS files as they load**
  - Scan for a code signature in OSLOADER and patch there
  - Must handle INT 13h/AH=02h (Read Sectors) and INT 13h/AH=42h (IBM/MS Extensions – Extended Read)
- **OSLOADER patch gives us an intermediate point to regain execution and modify OS further (i.e., patch boot drivers)**

eEye BootRoot – Other Possibilities

27

- **Modify system files on disk before Windows startup**
  - Intrusive; requires code to navigate FAT and NTFS
    - Could we piggyback off Windows boot loader code?
- **Hook INT 15h to reserve any amount of extended memory**
  - OSLOADER calls INT 15h/AX=E820h to get memory map
- **Regain execution by hooking an interrupt called late in Windows startup**
  - More of OS is loaded – more available to modify
  - Our hook runs in real mode, so we must re-enter protected mode to modify OS memory above 1MB

eEye Digital Security

eEye BootRoot – System Memory Map Example

28

Base Address	Length	Type
0000000000000000	000000000009F800	1 Available
000000000009F800	0000000000000800	2 (Reserved)
00000000000CA000	0000000000002000	2 (Reserved)
00000000000DC000	0000000000004000	2 (Reserved)
00000000000E4000	000000000001C000	2 (Reserved)
0000000000100000	0000000007DF0000	1 Available
0000000007EF0000	000000000000C000	3 (ACPI Reclaimable)
0000000007EFC000	0000000000004000	4 (ACPI NVS)
0000000007F00000	0000000000100000	1 Available
00000000FEC00000	0000000000010000	2 (Reserved)
00000000FEE00000	0000000000001000	2 (Reserved)
00000000FFFE0000	0000000000020000	2 (Reserved)

System memory map generated using INT 15h/AX=E820h on a VMWare 4.5 system with 128MB RAM.

eEye Digital Security

eYe BootRootKit

29

“Finally, someone implemented it.”

eEye Digital Security

eEye BootRootKit – Overview

30

- **Proof-of-concept for eEye BootRoot technology**
  - Loads from many bootable media
  - Installs INT 13h hook to “patch” OSLOADER on load
  - OSLOADER patch locates module list, hooks NDIS.SYS
  - NDIS backdoor inspects incoming packets for code to run
- **Features**
  - Works on Windows 2000 and later
  - Fits into 512 bytes!
- **The idea is simple, but there are always hidden complexities**

eEye Digital Security

eEye BootRootKit – INT 13h Hook

31

- **Move to reserved conventional memory and hook INT 13h**
  - Warning: Don't assume value of CS!
    - Executed from disk – CS:IP = 0000h:7C00h
    - Executed from CD – CS:IP = 07C0h:0000h
- **INT 13h hook scans read sectors for a code signature**
  - INT 13h hook must be able to handle INT 13h/AH=02h *and* INT 13h/AH=42h extended reads (required for large disks)
  - Signature should be unique, cross-version, and must not be split across a read boundary (i.e., across two sectors)
  - Warning: OSLOADER verifies PE checksums (except itself)
    - Could disable checksum checking code... ("`CMP reg1, [reg2+58h]`")

eEye Digital Security

eEye BootRootKit – OSLOADER Patch

32

- **We patch 6 bytes executed after boot driver load:**

```
0031ADF1 8B F0    MOV  ESI, EAX
0031ADF3 85 F6    TEST ESI, ESI
0031ADF5 74 21    JZ   $+23h
0031ADF7 80 ...   ; not modified, only used as part of signature
```

  - Hook must be absolute – we don't know where code will load
    - "`CALL seg:ofs32`" is 7 bytes
    - "`CALL DWORD PTR [ofs32]`" is 6 bytes – perfect for this patch site
  - We use "`CALL DWORD PTR [addr1]`", where `[addr1] = addr2`, and both `addr1` and `addr2` are addresses in our resident code
  - Paging is not a concern – OSLOADER will map low 16MB virtual memory to low 16MB physical memory

eEye Digital Security

eEye BootRootKit – OSLOADER Patch Function

33

- **Scan OSLOADER for address of _BlLoaderBlock**
  - Assume OSLOADER begins on a 1MB boundary
    - 00300000h for 2000 and XP, 00400000h for 2003
  - Use CALL hook return address as pointer into OSLOADER
  - Scan for the following code signature:

```
00301888 C7 46 34 00 40 00 00     MOV  DWORD PTR [ESI+34h], 4000h
 ...
00301895 A1 xx xx xx xx           MOV  EAX, [_BlLoaderBlock]
```

  - `[[_BlLoaderBlock]+0]` points to base of module list

- **Search module list for NDIS.SYS**
  - Name is usually uppercase, but don't assume

eEye BootRootKit – OSLOADER Module List

34

```
+00h  LIST_ENTRY      module list links
+08h  [10h]           ???
+18h  PTR             image base address
+1Ch  PTR             module entry point
+20h  DWORD           size of loaded module in memory
+24h  UNICODE_STRING  full module path and file name
+2Ch  UNICODE_STRING  module file name
```

Format of loaded module list nodes used by OSLOADER and based at [[_BlLoaderBlock]+0].
Structure is identical to that used by NTOSKRNL in PsLoadedModuleList.

eEye BootRootKit – Hooking NDIS (1)

35

- **Scan NDIS.SYS for code signature**
  - This signature within ndisMLoopbackPacketX:

```
BFECEE7E 50                      PUSH EAX
BFECEE7F 53                      PUSH ECX
BFECEE80 C7 46 10 0E 00 00 00    MOV  DWORD PTR [ESI+10h], 0Eh
BFECEE87 E8 xx xx xx xx          CALL ethFilterDprIndicateReceivePacket
```

    Leads to ethFilterDprIndicateReceivePacket, which we hook
  - Note: These two functions are in different PE sections

- **In 2000 and XP, boot drivers' sections aren't aligned yet!**
  - We must translate raw offsets into Relative Virtual Addresses, and vice versa, to find actual CALL destination and then store our own relative JMP hook there
  - If listed module size is 64KB multiple, sections are aligned(?)

eEye Digital Security

eEye BootRootKit – Hooking NDIS (2)

36

- **Hook ethFilterDprIndicateReceivePacket**
  - Store a relative JMP at function entry point, or two bytes afterward if first instruction is "MOV EDI, EDI"
  - Assume overwritten instructions will always be:
    `PUSH EBP / MOV EBP, ESP / SUB ESP, imm` (exact value is irrelevant, we just subtract a lot)
  - Write protection not enabled yet, modify away!
  - Code is not pageable so it will never be reloaded from disk

- **Store hook function code**
  - We overwrite DOS "MZ" code at (image base + 40h)
  - This hook function provides a remote kernel backdoor

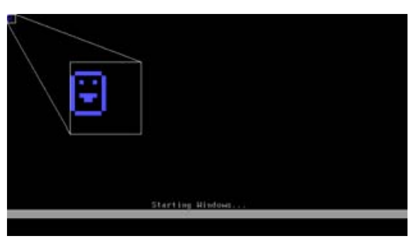
eEye Digital Security

eEye BootRootKit – NDIS Backdoor

- **Hook function checks received packets for signature**
  - ethFilterDprIndicateReceivePacket sees all incoming frames
  - arg_4 0 8 0C is pointer to Ethernet frame data
  - arg_4 0 8 14 is frame size
  - Check offset 55h within frame for ‘eBR\xEE’ signature
    - Should be beyond IP and TCP/UDP headers, even with options
    - If present, execute code directly from frame at offset 59h
- **For large payloads, send “mini-payloads” to construct code**
  - SharedUserData (FFDF0000h) is universal and writable, and visible in user-land at 7FFE0000h

Demonstration

- **From a floppy disk**
- **From a CD-RW**
- **Via network boot**



Look for the blue smiley!

To-Do

39

- **Adapt for more traditional rootkit functionality**
- **Explore other methods of retaining execution potential besides INT 13h hook-based patching**
- **Investigate bootable USB storage and other bootable media**

Bonus Material!

40

- **A little something extra for those who thought this talk would be entirely boring... (you may still be right)**
- **Did you know:**
  - You can perform raw disk operations without entering the kernel?
  - It's not an NT kernel vulnerability!
  - It's...

IOPL Technique

41

It's a Feature, Not a Vulnerability

IOPL Technique – Overview

42

- **EFlags contains an IOPL (I/O Privilege Level) field**
  - CPL <= IOPL (numerically less; greater privileges) can use:
    - IN / INSB / INSW / INSD
    - OUT / OUTSB / OUTSW / OUTSD
    - CLI / STI
  - Only ring 0 can modify IOPL
- **Some CSRSS threads run with IOPL=3 (prior to 2003)**
  - These threads can be hijacked, or
  - NtSetInformationProcess(ProcessUserModeIOPL)
    - Must have SeTcbPrivilege
  - Either way requires SYSTEM-equivalent privileges

IOPL Technique – Disk Access

43

- **Allows low-level disk access using port I/O**
  - Possible on IDE drives with only port I/O
    - No DMA required, no IRQs generated, etc.
    - For sample code, see [Kaze] in References
  - So what?
    - Evade anti-virus boot sector protection?
    - Evade system integrity assurance software?
    - Defeat machine state preservation software?
    - Fun way to install eEye BootRootKit on a hard drive

IOPL Technique – Local Kernel Backdoor?

44

- **Could it allow kernel subversion?**
  - DMA has provisions for memory-to-memory transfers
  - PIC can be reprogrammed
    - Could a spurious software interrupt / exception / IRQ get EIP to an address < MmUserProbeAddress?
    - Some fault handlers expect CPU to push error code after EIP
      - Fault: `Error, EIP, CS, EFlags, ESP, SS`
      - IRQ: `EIP, CS, EFlags, ESP, SS`
  - Arbitrary disk contents can be modified...
    - ...And we can violently reboot (“`MOV AL, 0FEh / OUT 64h, AL`”)
  - Much harder to monitor than “\Device\PhysicalMemory”, ZwSystemDebugControl [randnut], or loading a driver

References

Brown, Ralf. *Ralf Brown's Interrupt List.* `http://www.cs.cmu.edu/~ralf/files.html`

ECMA. *Standard ECMA-119: Volume and File Structure of CDROM for Information Interchange.* `http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-119.pdf`

Intel Corporation. *Preboot Execution Environment (PXE) Specification, Version 2.1.* `ftp://download.intel.com/labs/manage/wfm/download/pxespec.pdf`

Kaze <Kaze_0mx@yahoo.fr>. "FATdoc#1.txt: Lire le Fat via les Ports." `http://fat.lyua.org/frm/data/fatdoc1.txt`

NTFS.com. "Hard Drive Partition – Partition Table." `http://www.ntfs.com/partition-table.htm`

randnut@hotmail.com. "Multiple WinXP kernel vulns can give user mode programs kernel mode privileges." `http://lists.grok.org.uk/pipermail/full-disclosure/2004-February/017545.html`

Russinovich, Mark. "Inside the Boot Process, Part 1." `http://www.windowsitpro.com/Article/ArticleID/3952/3952.html`

Stevens, Curtis E., and Stan Merkin. *"El Torito" Bootable CD-ROM Format Specification, Version 1.0.* `http://www.phoenix.com/NR/rdonlyres/98D3219C-9CC9-4DF5-B496-A286D893E36A/0/specscdrom.pdf`

Questions?

Questions? Comments? E-mail me! dsoeder@eeye.com