Alureon: The First ITW 64-Bit Windows Rootkit

Joe Johnson
Software Development Engineer
Microsoft Corporation

Microsoft

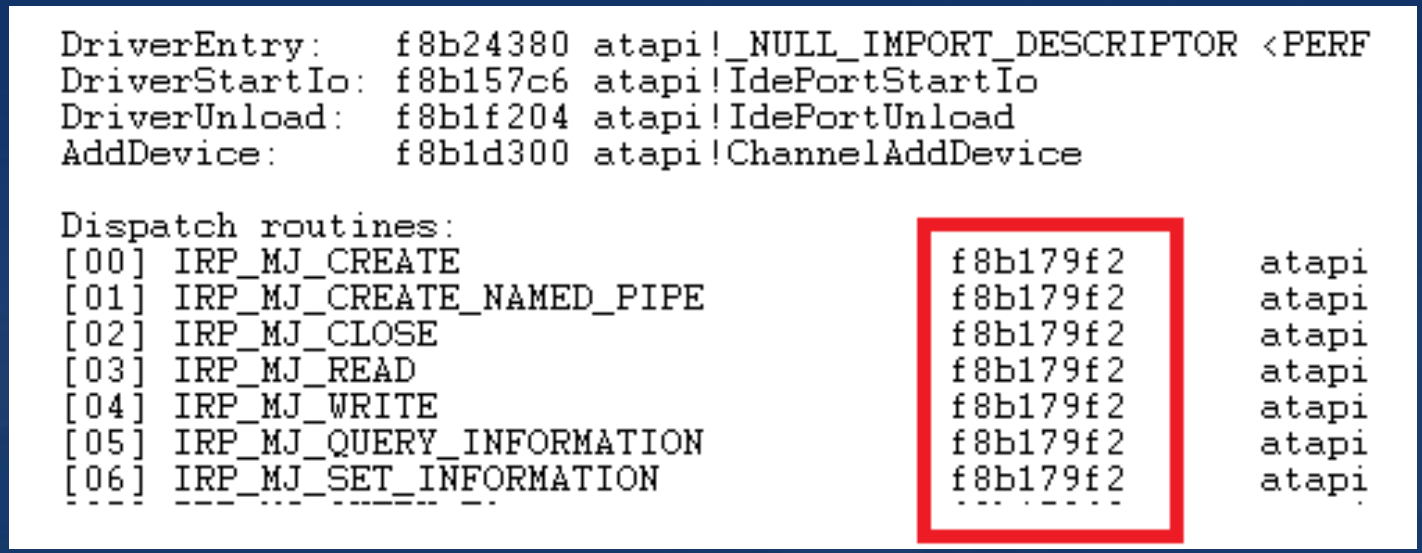
Overview

- A crash course on TDL3, pre-MBR
- A look at the initial MBR infector and its ominous ldr64
- A deep dive into the updated version that includes 64-bit support
- A quick comparison to Sinowal
- A look into how Alureon became the first public ITW kernel rootkit to affect 64 bit Windows

Microsoft

Alureon\TDL3 Evolution

- Initially, TDL3 infected the resource section of the miniport driver for \Systemroot and directly replaced all the IRP handlers for it



Microsoft

Alureon\TDL3 Evolution

- It evolved to only replace the device object in the relevant device stack and clean up the in memory image of the infected driver

```
kd> !drvobj partmgr
Driver object (82bdc3a0) is for:
 \Driver\PartMgr
Driver Extension List: (id , addr)
(f8e552d8 82bdc2b8)
Device Object list:
82b95900
kd> !devstack 82b95900
  !DevObj    !DrvObj            !DevExt    ObjectName
> 82b95900  \Driver\PartMgr     82b959b8
  82b0aab8  \Driver\Disk        82b0ab70   DR0
  82be0f18  \Driver\ACPI        82b98008   00000050
Invalid type for DeviceObject 0x82b91940
```

Microsoft

Alureon\TDL3 Evolution

- Eventually, it started infecting random drivers instead of the miniport driver it targeted in memory
- On 7/19/2010 we received our first sample of a new version that infects the MBR instead of a driver

Microsoft

First MBR-infecting Alureon

Something old, something new.

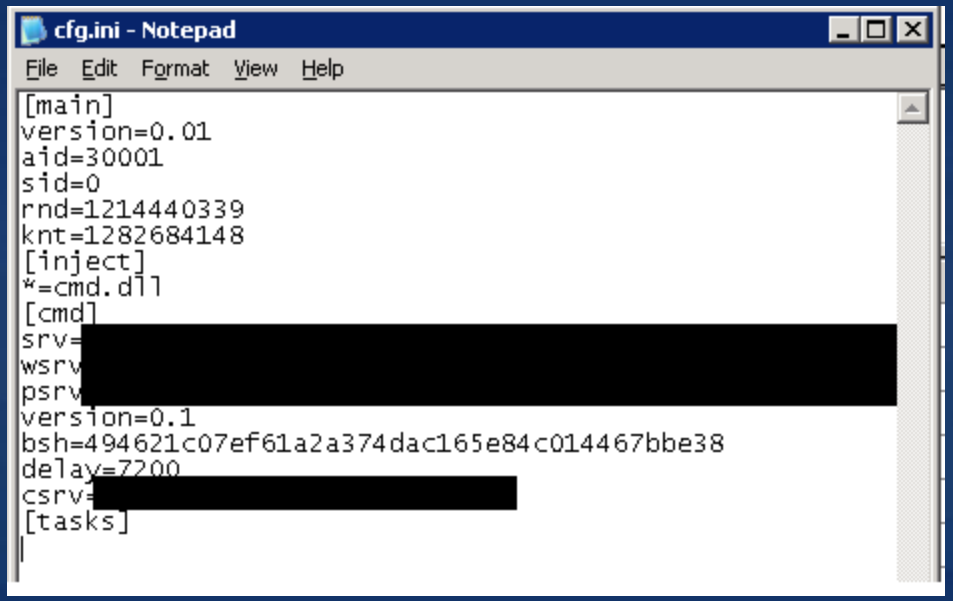
- Still infects the device stack for the drive containing \Systemroot
- Still has everything except the initial loading code in an encrypted virtual file system at the end of the disk
- Still installs via driver loaded by the spooler
- Now loads via an infected MBR instead of an infected driver

Microsoft

First MBR-infecting Alureon

Something old, something new.

- The MBR variant uses a config file as well, but now the version is 0.01 instead of 3.27.3, indicating a version under development



Microsoft

First MBR Infecting Alureon

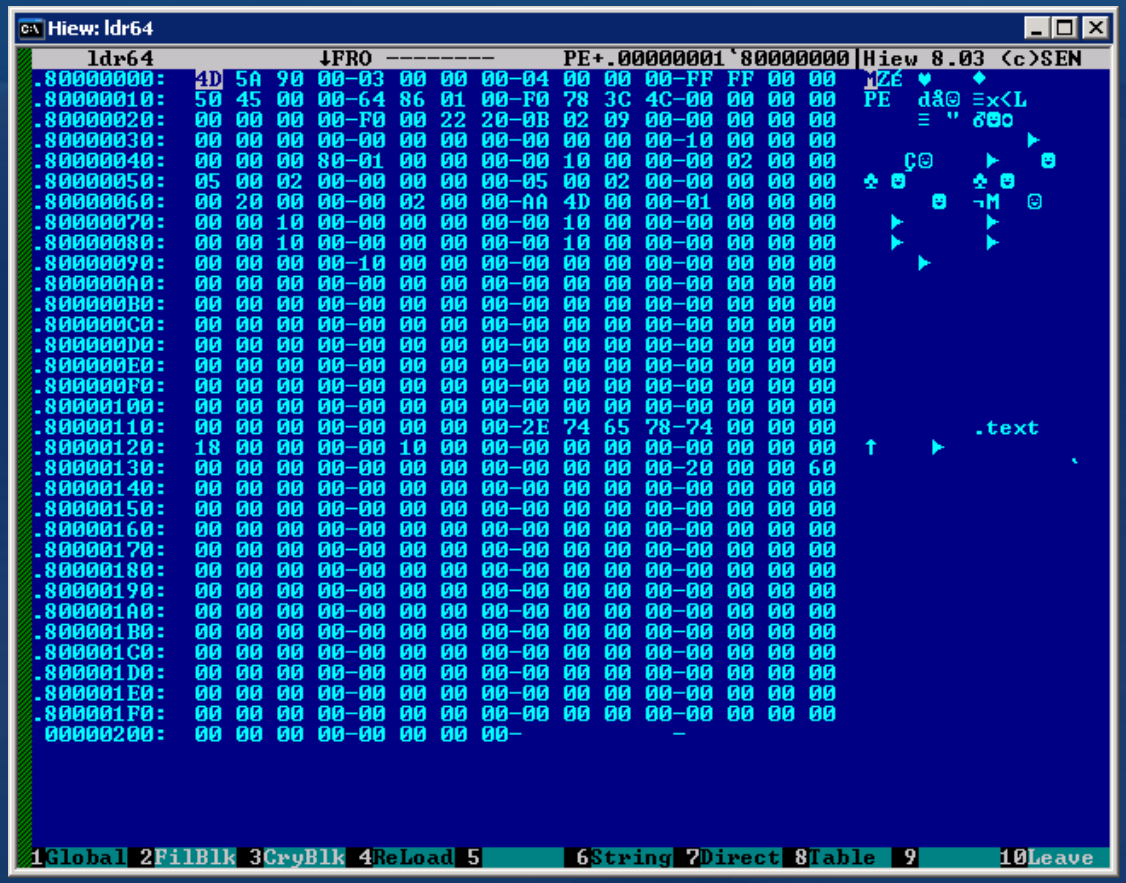
Contents of the virtual file system

- mbr – Copy of the original mbr
- ldr16 – Int 13h hook and loader for ldr32
- ldr32 – Fake KD (kernel debugger communication DLL)
- drv32 – Payload driver. Handles hooking device stack for \SystemRoot and injecting processes
- Cfg.ini – Configuration file
- ldr64 – Uh… **???**

Microsoft

First MBR Infecting Alureon

ldr64 empty! Phew!



Microsoft

MBR Alureon 0.02

The 64-bit Rootkit

- First appeared on 8/9/2010
- Ldr64 is no longer empty and 64 bit version of the payload driver is now also present
- Because of Code Integrity, the print spooler driver load no longer works, so it writes to disk the old fashioned way

3573	3:57:5...	DCFE.tmp.exe	2720	CreateFile	\Device\Harddisk0\DR0	SUCCESS	Desired Access: Generic Read/Write, Disposition: Open,..
3574	3:57:5...	DCFE.tmp.exe	2720	DeviceIoControl	\Device\Harddisk0\DR0	FAST IO DISALLO...	Control: IOCTL_SCSI_PASS_THROUGH_DIRECT
3575	3:57:5...	DCFE.tmp.exe	2720	DeviceIoControl	\Device\Harddisk0\DR0	SUCCESS	Control: IOCTL_SCSI_PASS_THROUGH_DIRECT
3576	3:57:5...	DCFE.tmp.exe	2720	DeviceIoControl	\Device\Harddisk0\DR0	FAST IO DISALLO...	Control: IOCTL_SCSI_PASS_THROUGH_DIRECT
3577	3:57:5...	DCFE.tmp.exe	2720	DeviceIoControl	\Device\Harddisk0\DR0	SUCCESS	Control: IOCTL_SCSI_PASS_THROUGH_DIRECT
3578	3:57:5...	DCFE.tmp.exe	2720	DeviceIoControl	\Device\Harddisk0\DR0	FAST IO DISALLO...	Control: IOCTL_SCSI_PASS_THROUGH_DIRECT
3579	3:57:5...	DCFE.tmp.exe	2720	DeviceIoControl	\Device\Harddisk0\DR0	SUCCESS	Control: IOCTL_SCSI_PASS_THROUGH_DIRECT
3580	3:57:5...	DCFE.tmp.exe	2720	DeviceIoControl	\Device\Harddisk0\DR0	FAST IO DISALLO...	Control: IOCTL_SCSI_PASS_THROUGH_DIRECT
3581	3:57:5...	DCFE.tmp.exe	2720	DeviceIoControl	\Device\Harddisk0\DR0	SUCCESS	Control: IOCTL_SCSI_PASS_THROUGH_DIRECT
3582	3:57:5...	DCFE.tmp.exe	2720	DeviceIoControl	\Device\Harddisk0\DR0	FAST IO DISALLO...	Control: IOCTL_SCSI_PASS_THROUGH_DIRECT
3583	3:57:5...	DCFE.tmp.exe	2720	DeviceIoControl	\Device\Harddisk0\DR0	SUCCESS	Control: IOCTL_SCSI_PASS_THROUGH_DIRECT
3584	3:57:5...	DCFE.tmp.exe	2720	DeviceIoControl	\Device\Harddisk0\DR0	FAST IO DISALLO...	Control: IOCTL_SCSI_PASS_THROUGH_DIRECT

Microsoft

How Does It Load?

MBR

- The MBR starts with boilerplate relocation code to move to 0x600
- After jumping to the relocated code, it runs a simple ROR decryption loop (only 12 bytes of code)
- It then decrypts and loads ldr16 from the VFS

How Does It Load?

Ldr16 – int13h hook

- The first thing ldr16 does after loading is to hook int13h, the BIOS disk read interrupt
- After starting the normal load sequence, it watches for the load of a KD communication extension (usually kdcom.dll), and replaces it with ldr32 or ldr64 depending on the PE

```
xor     si, si
mov     es, si
mov     eax, es:[si+4Ch]
mov     dword ptr ds:call_original_int13+1, eax
mov     ah, 48h ; 'H'
mov     si, 4F6h
mov     word ptr ds:4F6h, 1Eh
int     13h             ; DISK -
xor     di, di
mov     word ptr es:[di+4Ch], offset loc_3F
mov     word ptr es:[di+4Eh], cs
```

Name	RVA	Size
Export	00001060	000000FA
Import	000019FC	00000028
Resource	00000000	00000000

Microsoft

How Does It Load?

Finding kdcom.dll

```
seg000:0112 ReplaceKdcom:                           ; CODE XREF: seg000:00BA↑j
seg000:0112                 cmp     word ptr es:[bx], 5A4Dh ; MZ
seg000:0117                 jnz     DoHooks
seg000:011B                 mov     di, es:[bx+3Ch]
seg000:011F                 cmp     word ptr es:[bx+di], 4550h ;  PE
seg000:0124                 jnz     DoHooks
seg000:0128                 cmp     word ptr es:[bx+di+18h], 10Bh ; Look for 32bit optional header size
seg000:012E                 jnz     short ReplaceKdcom64
seg000:0130                 cmp     dword ptr es:[bx+di+7Ch], 0
seg000:0136                 jz      short loc_106
seg000:0138                 cmp     dword ptr es:[bx+di+7Ch], 0FAh ; '-'
seg000:0141                 jnz     DoHooks
seg000:0145                 mov     si, offset aLdr32 ; "ldr32"
seg000:0148                 mov     cx, 6
seg000:014B                 jmp     short ReadVfsFile
seg000:014D ; ---------------------------------------------------------------------------
seg000:014D
seg000:014D ReplaceKdcom64:                         ; CODE XREF: seg000:012E↑j
seg000:014D                 cmp     dword ptr es:[bx+di+8Ch], 0
seg000:0154                 jz      short loc_FA
seg000:0156                 cmp     dword ptr es:[bx+di+8Ch], 0FAh ; '-'
seg000:0160                 jnz     DoHooks
seg000:0164                 mov     si, 3BFh
seg000:0167                 mov     cx, 6
seg000:016A
seg000:016A ReadVfsFile:                            ; CODE XREF: seg000:014B↑j
seg000:016A                 cld
```

Microsoft

How Does It Load?

Other int13h patches

- The first block swaps the EmsEnabled library flag (0x16000020) in the BCD to be the WinPEMode OS Loader flag (0x26000022)

```
DoHooks:                                ; CODE XREF: s
                                        ; seg000:00D6
                movzx   cx, byte ptr ds:3E1h
                shl     cx, 7

PatchBcd2Pe:                            ; CODE XREF: s
                cmp     dword ptr es:[bx], '0061'
                jnz     short patchEmsToApp
                cmp     dword ptr es:[bx+4], '0200'
                jnz     short patchEmsToApp
                mov     dword ptr es:[bx], '0062'
                mov     dword ptr es:[bx+4], '2200'
```

Microsoft

How Does It Load?

Other int13h patches continued

- The second block tweaks the parent of the EmsEnabled Library flag to allow the WinPEMode OS loader flag to work

```
patchEmsToApp:                          ; CODE XREF: s
                                        ; seg000:0238↑
                cmp     dword ptr es:[bx], 1666Ch
                jnz     short KillMin
                cmp     dword ptr es:[bx+8], '0061'
                jnz     short KillMin
                mov     dword ptr es:[bx+8], '0062'
```

Microsoft

How Does It Load?

Other int13h patches continued

- The third block changes the string "/MIN" to "IN/M", hiding the normal registry option of "/MININT" that would be visible for a WinPE boot.

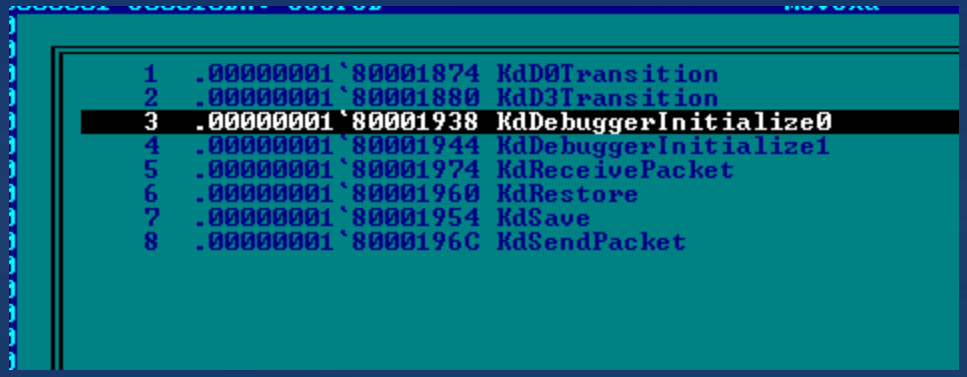
```
KillMin:                                ; CODE XREF: seg
                                        ; seg000:025E↑j
                cmp     dword ptr es:[bx], 'NIM/'
                jnz     short next
                mov     dword ptr es:[bx], 'M/NI'
```

Microsoft

How Does It Load?

Ldr64 fake KD communications DLL

- The kernel loads the fake version of kdcom thanks to the int13h hook
- Most of the debugging operations are set to return safe values, but KdDebuggerInitialize1 has the first link in the (long) loading chain



Microsoft

How Does It Load?

Ldr64 fake KD communications DLL

```
KdDebuggerInitialize1:                                  lea         rcx,[00000001`8000190C] --↑2
.00000001`8000194B: 48FF25F6F6FFFF                      jmp         PsSetLoadImageNotifyRoutine
.00000001`80001952: CC                                  int         3
.00000001`80001953: CC                                  int         3
```

```
.00000001`8000190C: 4883EC28                         5sub           rsp,028 ;'('
.00000001`80001910: 833DE100000000                     cmp          d,[00000001`800019F8],0 --↓1
.00000001`80001917: 7519                               jnz         .00000001`80001932 --↓2
.00000001`80001919: 488D156CFFFFFF                     lea          rdx,[00000001`8000188C] --↑3
.00000001`80001920: 33C9                               xor          ecx,ecx
.00000001`80001922: FF1518F7FFFF                       call         IoCreateDriver
.00000001`80001928: C705C600000001000000               mov          d,[00000001`800019F8],1 --↓1
.00000001`80001932: 4883C428                         2add           rsp,028 ;'('
.00000001`80001936: C3                                 retn ; _^_^_^_^_^_^_^_^_^_^_^_^_^_^_^_^_
```

```
.00000001`8000188C: 4C8BDC                           5mov           r11,rsp
.00000001`8000188F: 4883EC58                           sub          rsp,058 ;'X'
.00000001`80001893: B8BFB60000                         mov          eax,00000B6BF ;'  ╣'
.00000001`80001898: BA01000000                         mov          edx,1
.00000001`8000189D: 4C8BC9                             mov          r9,rcx
.00000001`800018A0: 6689442444                         mov          [rsp][044],ax
.00000001`800018A5: B8D0110000                         mov          eax,0000011D0 ;'  ◄ш'
.00000001`800018AA: 4D8D43E8                           lea          r8,[r11][-018]
.00000001`800018AE: 6689442446                         mov          [rsp][046],ax
.00000001`800018B3: 488D0536010000                     lea          rax,[00000001`800019F0] --↓1
.00000001`800018BA: C74424400763F553                   mov          d,[rsp][040],053F56307 ;'SJc•'
.00000001`800018C2: 498943D8                           mov          [r11][-028],rax
.00000001`800018C6: 49894BD0                           mov          [r11][-030],rcx
.00000001`800018CA: 488D05FFFCFFFF                     lea          rax,[00000001`800015D0] --↑2
.00000001`800018D1: 8D4A01                             lea          ecx,[rdx][1]
.00000001`800018D4: C644244894                         mov          b,[rsp][048],094 ;'ö'
.00000001`800018D9: C6442449F2                         mov          b,[rsp][049],0F2 ;'≥'
.00000001`800018DE: 498943C8                           mov          [r11][-038],rax
.00000001`800018E2: C644244A00                         mov          b,[rsp][04A],0
.00000001`800018E7: C644244BA0                         mov          b,[rsp][04B],0A0 ;'á'
.00000001`800018EC: C644244CC9                         mov          b,[rsp][04C],0C9 ;'╔'
.00000001`800018F1: C644244D1E                         mov          b,[rsp][04D],01E
.00000001`800018F6: C644244EFB                         mov          b,[rsp][04E],-5 ;'√'
.00000001`800018FB: C644244F8B                         mov          b,[rsp][04F],08B ;'ï'
.00000001`80001900: FF1532F7FFFF                       call         IoRegisterPlugPlayNotification
.00000001`80001906: 4883C458                           add          rsp,058 ;'X'
.00000001`8000190A: C3                                 retn ; _^_^_^_^_^_^_^_^_^_^_^_^_^_^_^_^_
```

Microsoft

How Does It Load?

Ldr64 fake KD communications DLL cont.

```
.00000001`800015D0: 4C8BDC          mov     r11,rsp
.00000001`800015D3: 49895B10        mov     [r11][010],rbx
.00000001`800015D7: 55              push    rbp
.00000001`800015D8: 56              push    rsi
.00000001`800015D9: 57              push    rdi
.00000001`800015DA: 4154            push    r12
.00000001`800015DC: 4155            push    r13
.00000001`800015DE: 4156            push    r14
.00000001`800015E0: 4157            push    r15
.00000001`800015E2: 4881EC80020000  sub     rsp,000000280 ;'  '
.00000001`800015E9: 488B4128        mov     rax,[rcx][028]
.00000001`800015ED: 33FF            xor     edi,edi
.00000001`800015EF: 4C8BFA          mov     r15,rdx
.00000001`800015F2: 4C8D4C2468      lea     r9,[rsp][068]
.00000001`800015F7: 4C8D442438      lea     r8,[rsp][038]
.00000001`800015FC: 498D4B18        lea     rcx,[r11][018]
.00000001`80001600: BA01001000      mov     edx,000100001
.00000001`80001605: C744242820000000 mov    d,[rsp][028],000000020 ;'     '
.00000001`8000160D: 4889442448      mov     [rsp][048],rax
.00000001`80001612: C744243830000000 mov    d,[rsp][038],000000030 ;'    0'
.00000001`8000161A: 48897C2440      mov     [rsp][040],rdi
.00000001`8000161F: C744245040020000 mov    d,[rsp][050],000000240 ;'  @'
.00000001`80001627: 48897C2458      mov     [rsp][058],rdi
.00000001`8000162C: 48897C2460      mov     [rsp][060],rdi
.00000001`80001631: C744242007000000 mov    d,[rsp][020],7
.00000001`80001639: FF15D1F9FFFF    call    ZwOpenFile
.00000001`8000163F: 3BC7            cmp     eax,edi
.00000001`80001641: 0F8C10020000    jl      .00000001`80001857 --↓1
.00000001`80001647: 488B8C24D0020000 mov    rcx,[rsp][0000002D0]
.00000001`8000164F: 4C8D4C2430      lea     r9,[rsp][030]
```

```
.00000001`80001700: 488BDF          mov     rbx,rdi
.00000001`80001703: 8BC2            mov     eax,edx
.00000001`80001705: 488D3D50FAFFFF  lea     rdi,[00000001`8000115C] ;'drv64'  --↑5
.00000001`8000170C: B906000000      mov     ecx,6
.00000001`80001711: 48C1E005        shl     rax,5
.00000001`80001715: 488DB40486000000 lea    rsi,[rsp][rax][0000000086]
.00000001`8000171D: F3A6            rep cmpsb
```

Microsoft

Differences From Sinowal

- While both follow the MBR -> 16 bit loader pattern, they take different approaches after that
- Sinowal hooks the kernel to install its driver loader, then hooks IoInitSystem to load its driver after the rest of the drivers are loaded
- Alureon replaces the debugging infrastructure and relies on built in kernel routines to do the work thereafter

Microsoft

How About Patchguard?

- 64 bit Alureon does not bypass Kernel Patch Protection (Patchguard)
- Patchguard only guards the code and structures used by the kernel, not all loaded drivers
- The drv64 payload, once loaded, behaves in a way consistent with normal third party extensibility in the kernel

Microsoft

What About Code Integrity?

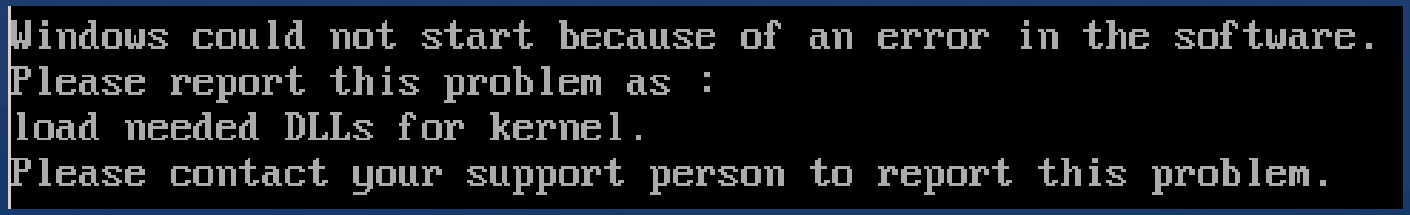
No, seriously, how *does* it load?

- In this case, the BCD was patched to inform winload that this is Windows PE booting, not a normal version of Windows
- Winload does not check code integrity in this case
- As seen before, the malware also trashes the "/MININT" string which prevents the rest of Windows from treating the OS instance as WinPE

Microsoft

What About Windows XP-64?

- In the case of Windows XP-64 and Server 2003 64-bit, the fake version of kdcom.dll breaks the boot sequence
- This renders the machine unbootable



Microsoft

Detection

- Detected as
  - Trojan:Win32/Alureon.DX (Dropper)
  - Trojan:DOS/Alureon.A (MBR\Rootkit)
- Infected machines also no longer list the system disk in diskpart

Microsoft

Cleanup

- Since the only trace of the infection outside the virtual file system is the MBR, that is all that needs to be disabled
- Online cleaning is problematic, but currently offline cleaning is trivial. Use fixmbr for XP/Server 2003 (32 and 64 bit), and bootrec for Vista+. Future variants may break this.

Microsoft

Conclusions

- As of July 2010, 46% of Windows 7 machines are 64-bit
- Server 2008 R2 only supports 64-bit
- As more machines move to 64-bit Windows, we can expect more malware to move into the 64-bit kernel

Thanks to Scott Molenkamp, Jimmy Kuo, Vincent Tiu, Mady Marinescu for their help on this presentation.

Microsoft


Questions?

More Reading

- MMPC blog – http://blogs.technet.com/b/mmpc/archive/2010/08/27/alureon-evolves-to-64-bit.aspx
- Threat/Cleaning details -- http://www.microsoft.com/security/portal/Threat/Encyclopedia/Entry.aspx?Name=Trojan%3aDOS%2fAlureon.A
- Active discussion on latest TDL3 developments -- http://www.kernelmode.info/forum/viewtopic.php?f=16&t=19
- Early TDL3 paper -- http://www.drweb.com/static/BackDoor.Tdss.565_%28aka%20TDL3%29_en.pdf

Microsoft

Microsoft®

Your potential. Our passion.™