Alureon: The First ITW 64-Bit Windows Rootkit

Joe Johnson
Software Development Engineer
Microsoft Corporation
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
Initially, TDL3 infected the resource section of the miniport driver for `\Systemroot` and directly replaced all the IRP handlers for it.

```
DriverEntry: f8b24380 atapi!_NULL_IMPORT_DESCRIPTOR <PERF
DriverStartIo: f8b157c6 atapi!IdePortStartIo
DriverUnload: f8b1f204 atapi!IdePortUnload
AddDevice: f8b1d300 atapi!ChannelAddDevice

Dispatch routines:
[00] IRP_MJ_CREATE          f8b179f2  atapi
[01] IRP_MJ_CREATE_NAMED_PIPE f8b179f2  atapi
[02] IRP_MJ_CLOSE            f8b179f2  atapi
[03] IRP_MJ_READ             f8b179f2  atapi
[04] IRP_MJ_WRITE            f8b179f2  atapi
[05] IRP_MJ_QUERY_INFORMATION f8b179f2  atapi
[06] IRP_MJ_SET_INFORMATION  f8b179f2  atapi
```
It evolved to only replace the device object in the relevant device stack and clean up the in memory image of the infected driver.
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.
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
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.
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... ???
First MBR Infecting Alureon

Idr64 empty! Phew!
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
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 kdccom.dll), and replaces it with Ldr32 or Ldr64 depending on the PE.
How Does It Load?

Finding kdcom.dll

```
seg000:0112  ReplaceKdcom:
seg000:0112  cmp    word ptr es:[bx], 5A40h ; M2
seg000:0117  jnz    DoHooks
seg000:0118  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+10h], 100h ; 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], OFAh ; '
seg000:0141  jnz    DoHooks
seg000:0145  mov    si, offset aLdr32 ; "ldr32"
seg000:0148  mov    cx, 5
seg000:0148  jmp    short ReadUvsFile
seg000:014D  ReplaceKdcom64:
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], OFAh ; '
seg000:0160  jnz    DoHooks
seg000:0164  mov    si, 3BFh
seg000:0167  mov    cx, 5
seg000:016A  ReadUvsFile:
seg000:016A  cld
```
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)
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.
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.
How Does It Load?

Ldr64 fake KD communications DLL

- The kernel loads the fake version of kdcocm thanks to the int13h hook
- Most of the debugging operations are set to return safe values, but KdDebuggerInitialize has the first link in the (long) loading chain
How Does It Load?
Ldr64  fake KD communications DLL
How Does It Load?
Ldr64 fake KD communications DLL cont.

```
 mov    r11, rsp
 mov    [r11+010],rbx
 push   rbp
 push   rsi
 push   rdi
 push   r12
 push   r13
 push   r14
 push   r15
 sub    rsp,00000020
 mov    rax,[rcx+1028]
 xor    edi,edi
 mov    r15,rdx
 lea    r9,[rsp][068]
 lea    r8,[rsp][038]
 lea    rcx,[r11+018]
 lea    edx,00010000
 mov    d,[rsp][028],00000020
 mov    d,[rsp][038],00000030
 mov    d,[rsp][048],rax
 mov    d,[rsp][058],rdi
 mov    d,[rsp][060],rdi
 call   ZwOpenFile
 cmp    eax,edi
 jle    .00000001`
 mov    rcx,[rsp][000002D0]
 lea    r9,[rsp][030]
 mov    eax,eul
 mov    rbx,rdi
 mov    ecx,edx
 lea    rdi,[00000001`8000115C] ;‘dsv64’ --t5
 lea    rsi,[rsp][rax][0000000086]
 rep  cmpsb
```
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.
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
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.
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.

Windows could not start because of an error in the software.
Please report this problem as:
load needed DLLs for kernel.
Please contact your support person to report this problem.
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
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.
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.
More Reading

- MMPC blog –

- Threat/Cleaning details --

- Active discussion on latest TDL3 developments --

- Early TDL3 paper --