Boot What?

Why tech invented by IBM in 1983 is relevant today

June 22nd, 2017
$ WHOAMI

- Christopher Glyer
- Chief Security Architect, FireEye
- Incident Responder
- Forensic Analyst
- Wanna-be sailor
MBR/VBR BOOT ANTICS

- Disks, Partitions and Volumes
- Boot like it’s 1983!
- Known Attack Vectors
- FIN1 Bootcode Case Study
- Results at Scale
BACK IN 1983...
• Lotus 1-2-3
• Chicken McNuggets
• Redskins won first Super Bowl
• First season of The A-Team
• Thriller was best selling record
• IBM releases Master Boot Record
DISKS, PARTITIONS, VOLUMES, OH MY!

Lin: /dev/hda
Win: //./PhysicalDrive0\Device\Harddisk0\Partition0

Lin: /dev/hda1
Win: \Device\Harddisk0\Partition1

Lin: /mnt/blah
Win: C:\Device\HarddiskVolume1
TERMINOLOGY & DEFINITIONS

Basic Input Output System (BIOS)

Master Boot Record (MBR)

Volume Boot Record (VBR)

BIOS Parameter Block (BPB)

Initial Program Loader (IPL)
BOOT LIKE IT’S 1983
BOOT LIKE IT’S 1983

- POST
- Find the active drive (BIOS config)
- Load first sector of active drive (512 bytes, MBR) into 0x7C00
- Execute MBR
BOOT LIKE IT’S 1983

• **First sector of disk**
• Loaded at 0x7C00, but relocates itself to load VBR
• Mix of code and data (partition table)
• Executes in 16-bit real-mode
• **Reads Partition Table**
  • Locates *active* partition
• Loads first sector of *active partition* (512-bytes VBR) into 0x7C00
• Executes VBR
BOOT LIKE IT’S 1983

- First sector of active partition
- Loaded at 0x7C00
- Mix of code and data (BPB)
- Executes in 16-bit real-mode
- Loads IPL, 15-sectors after VBR
- Executes IPL
BOOT LIKE IT’S 1983

- **15-sectors following VBR**
- Frequently loaded at 0x7E00 (after VBR) or 0xD000
- Starts in 16-bit real-mode, but transitions to protected-mode
- Executes NTLDR/BOOTMGR
KNOWN ATTACK VECTORS
KNOWN ATTACK VECTORS

- Direct MBR Code Modification (TDL4)
- Partition Table Manipulation (Olmasco)

BIOS → MBR → VBR → IPL → OS

KNOWN ATTACK VECTORS

- Direct MBR Code Modification (TDL4)
- Partition Table Manipulation (Olmasco)

- Direct VBR Code Modification (Rovnix)
- BPB Manipulation (GapZ)

BIOS → MBR → VBR → IPL → OS

KNOWN ATTACK VECTORS

- Direct MBR Code Modification (TDL4)
- Partition Table Manipulation (Olmasco)

- Direct VBR Code Modification (Rovnix)
- BPB Manipulation (GapZ)

- Direct IPL Modification

# MBR REPLACEMENT

<table>
<thead>
<tr>
<th>MBR</th>
<th>Active Partition</th>
<th>Free Partition</th>
<th>Free Partition</th>
<th>Free Partition</th>
<th>Hidden Sectors</th>
</tr>
</thead>
<tbody>
<tr>
<td>VBR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IPL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FILE SYSTEM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Original MBR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Malicious Stage2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Replace MBR with malicious code
- Samples:
  - TDL4 – hooks FS DEVICE_OBJ to hide modifications
  - MebRoot – hooks disk.sys to hide modifications
  - XPAJ
PARTITION ADDITION

- Replace, modify or add partition table entries
- Samples:
  - Olmasco
VBR REPLACEMENT

<table>
<thead>
<tr>
<th>MBR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active Partition</td>
</tr>
<tr>
<td>Free Partition</td>
</tr>
<tr>
<td>Free Partition</td>
</tr>
<tr>
<td>Free Partition</td>
</tr>
<tr>
<td>Hidden Sectors</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BPB</th>
</tr>
</thead>
<tbody>
<tr>
<td>VBR</td>
</tr>
<tr>
<td>IPL</td>
</tr>
<tr>
<td>FILE SYSTEM</td>
</tr>
<tr>
<td>Original VBR</td>
</tr>
<tr>
<td>Malicious Code</td>
</tr>
</tbody>
</table>

- Replace VBR with malicious code
- Samples:
  - Rovnix (Cidox)
  - BOOTRASH (nemesis – VBR+IPL)
### VBR BPB MANIPULATION

<table>
<thead>
<tr>
<th>MBR</th>
<th>Active Partition</th>
<th>Free Partition</th>
<th>Free Partition</th>
<th>Free Partition</th>
<th>Hidden Sectors</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPB</td>
<td>VBR</td>
<td>IPL</td>
<td>FILE SYSTEM</td>
<td>VBR/Malicious Code</td>
<td></td>
</tr>
</tbody>
</table>

- Replace BPB values that will cause VBR to load from different location
- Samples:
  - GapZ – Modifies HiddenSectors to redirect VBR load
IPL REPLACEMENT

- Replace IPL with malicious code
- Samples:
  - Rovnix (Cidox)
  - BOOTRASH (nemesis VBR+IPL)
MALICIOUS ACTIONS

- Malicious code will commonly:
  - Hook IVT/IDT
  - Modify bytes on disk
  - Backup original MBR/VBR/IPL bytes
  - Hook kernel to hide modifications (disk.sys, miniport..)
FIN1 BOOTCODE CASE STUDY
WHO IS FIN1?

Notable cases:

• 2008: RBS WorldPay - $9 million ATM Fraud
• 2011: Fidelity Information Services - $13 million ATM fraud

Opsec & sophistication have significantly improved

• No backdoors – only web shells
• Commodity backdoors (e.g. poison ivy)
• All custom backdoors including linux and boot record manipulation
FINDING BOOTRASH

- Investigation identified system beaconing to bad domain
- Need to find the code that launched the backdoor
- Searched common persistence mechanisms
  - Services
  - Run keys
  - Scheduled tasks
  - Startup folders
FINDING BOOTRASH

• Searched for advanced persistence mechanisms
  • WMI Event filters/consumers
  • Search order hijacking/DLL side loading
  • Hijacked MBR
FINDING BOOTRASH

• Searched for advanced persistence mechanisms
  • WMI Event filters/consumers
  • Search order hijacking/DLL side loading
  • Hijacked MBR

• ***Found nothing***
GOOD RULE OF THUMB

*If you can’t find the persistence mechanism for malware – take the time to figure it out!*

Mandiant has found multiple “new” persistence techniques

- MBR modification
- WMI event filters/consumers
- Search order hijacking (first, second, and tertiary)
- DLL side loading
- Legitimate file patching
FINDING BOOTRASH

- Performed memory analysis and identified two processes injected with malware
  - wininet.exe
  - svchost.exe
- Attacker help menu!

VBR?!? – we’ve learned about that!
GETTING THE VBR

• Ran RedLine disk listing and volume listing audits
• Identified offset of the first volume on disk, researched the length of a VBR
• Ran RedLine disk acquisition audit to acquire the exact 512-bytes on disk containing the VBR

RedLine Disk Acquisition Audit Configuration

• **Drive:** \\.
• **Path:** PhysicalDrive0
• **Offset:** 63*512 or 2048*512 or ??*512
• **Size:** 512 or 16*512 to include IPL
TEAMWORK BIG WILLI STYLE

- Worked with malware analyst to disassemble the code to determine if it was malicious
  - Shout out to Willi Ballenthin
- Identified where other components may be stored
  - Backup copy of VBR
  - Location of the virtual file system
- Acquired malware components and put together full picture of how the malware operated
BOOTRASH DETAILS

1. BIOS loads MBR, MBR loads malicious VBR
2. Malicious VBR loads components from the custom Virtual File System (VFS)
   • VFS could be stored either in:
     • Registry
     • Unallocated space on disk
3. Malicious VBR loads legitimate VBR
4. Legitimate VBR loads IPL
BOOTRASH DETAILS

5. Patches Interrupt Vector Table entry
   • Intercept memory queries once the operating system loader gains control

6. Patches Interrupt Descriptor Table each time the CPU changes from real mode to protected mode
   • Redirects control to the bootkit every time a specific address is executed

7. Allows bootkit to intercept operating system loader execution and inject Nemesis components as part of the normal kernel loading
RESULTS AT SCALE
MBR @SCALE - RESULTS

Problem:
• Hashing entire MBR not effective due to timestamps included at offset 218-223

Solution:
• Hash the code section of MBR at offset mbr[:218] + mbr[224:416]
6663 unique MBR hashes across ~265K systems
MBR @SCALE - RESULTS

Why >6000 MBR Hashes?

• Numerous legitimate applications modify MBR (Altiris, SafeBoot, PGPGuard…)
• Lots of minor variations to known good. (ex: jmp instructions differing lengths)
• Strings “opErating system”
• MBR backup utilities – stores multiple backup copies of MBR and loads
• Loads 4 sectors of VBR instead of 1
MBR @SCALE - RESULTS

• Areas to Explore
  • Hamming distance calculation useful for finding variants
  • Emulation of 16-bit code (vivisect/unicorn) – loop/structure detection, hooking, instruction frequency
  • Taint Analysis
VBR @SCALE - RESULTS

- **VBR bootstrap hashes**
  - Currently stacks quite well with 165 unique VBR hashes across ~265K systems
  - Vast majority of infrequent hashes have “mProtect!” in the header

### VBR Bootstrap Hash Frequency

<table>
<thead>
<tr>
<th>Hash</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>EBA2D2E048E3A1FA0DE78F454B628C12</td>
<td>107894</td>
</tr>
<tr>
<td>D63F3089E6D54924701D1AEBA0887C95</td>
<td>86367</td>
</tr>
<tr>
<td>E7A27EB31DA26173C18D58F000F74B07</td>
<td>2839</td>
</tr>
<tr>
<td>A959459A09BC3C7A1E8C9BF2212741D1</td>
<td>1620</td>
</tr>
<tr>
<td>88304B05661A08E258B002A25F84A13D</td>
<td>360</td>
</tr>
<tr>
<td>3B9BC8D3DD94BC7367A85B677A14827B</td>
<td>226</td>
</tr>
<tr>
<td>3447D07FE447OB1AB1A8EA8EF888199</td>
<td>155</td>
</tr>
<tr>
<td>2D5423E2A9AF88B0A18607FC21DF245</td>
<td>108</td>
</tr>
</tbody>
</table>
VBR @SCALE - RESULTS

- **VBR BPB Metadata Stacking** - Hidden Sectors & IPL offset
  - 210 unique values (63/2048/81920/499505152 most common) with long tail depending on disk partitions
VBR @SCALE - RESULTS

• VBR BPB Metadata Stacking
  • JMP Instruction – 99.9% consistent (“EB5290”) jumps past BPB (0x54-bytes)
  • start_sector_lba (Partition table) was always equal to hidden_sectors (BPB)
  • nfatcopies, maxroot, numsectorsfat, sectorsper, numsectorspart, drivernumbers – all stack to 1 value.
One more thing...
VBR: BIOS PARAMATER BLOCK (BPB) OVERVIEW

- BPB describes layout of the storage volume
- GapZ – Modifies HiddenSectors value to redirect where VBR is loaded from disk

<table>
<thead>
<tr>
<th>Byte Offset</th>
<th>Field Length</th>
<th>Field Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0B</td>
<td>WORD</td>
<td>Bytes Per Sector</td>
</tr>
<tr>
<td>0x0D</td>
<td>BYTE</td>
<td>Sectors Per Cluster</td>
</tr>
<tr>
<td>0x0E</td>
<td>WORD</td>
<td>Reserved Sectors</td>
</tr>
<tr>
<td>0x10</td>
<td>3 BYTES</td>
<td>always 0</td>
</tr>
<tr>
<td>0x13</td>
<td>WORD</td>
<td>not used by NTFS</td>
</tr>
<tr>
<td>0x15</td>
<td>BYTE</td>
<td>Media Descriptor</td>
</tr>
<tr>
<td>0x16</td>
<td>WORD</td>
<td>always 0</td>
</tr>
<tr>
<td>0x18</td>
<td>WORD</td>
<td>Sectors Per Track</td>
</tr>
<tr>
<td>0x1A</td>
<td>WORD</td>
<td>Number Of Heads</td>
</tr>
<tr>
<td>0x1C</td>
<td>DWORD</td>
<td>Hidden Sectors</td>
</tr>
<tr>
<td>0x20</td>
<td>DWORD</td>
<td>not used by NTFS</td>
</tr>
<tr>
<td>0x24</td>
<td>DWORD</td>
<td>not used by NTFS</td>
</tr>
<tr>
<td>0x28</td>
<td>LONGLONG</td>
<td>Total Sectors</td>
</tr>
<tr>
<td>0x30</td>
<td>LONGLONG</td>
<td>Logical Cluster Number for the file $MFT</td>
</tr>
<tr>
<td>0x38</td>
<td>LONGLONG</td>
<td>Logical Cluster Number for the file $MFTMirr</td>
</tr>
<tr>
<td>0x40</td>
<td>DWORD</td>
<td>Clusters Per File Record Segment</td>
</tr>
<tr>
<td>0x44</td>
<td>DWORD</td>
<td>Clusters Per Index Block</td>
</tr>
<tr>
<td>0x48</td>
<td>LONGLONG</td>
<td>Volume Serial Number</td>
</tr>
<tr>
<td>0x50</td>
<td>DWORD</td>
<td>Checksum</td>
</tr>
</tbody>
</table>

### VBR: BIOS PARAMATER BLOCK (BPB) OVERVIEW

- What are the unused/null values??

<table>
<thead>
<tr>
<th>Byte Offset</th>
<th>Field Length</th>
<th>Field Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0B</td>
<td>WORD</td>
<td>Bytes Per Sector</td>
</tr>
<tr>
<td>0x0D</td>
<td>BYTE</td>
<td>Sectors Per Cluster</td>
</tr>
<tr>
<td>0x0E</td>
<td>WORD</td>
<td>Reserved Sectors</td>
</tr>
<tr>
<td>0x10</td>
<td>3 BYTES</td>
<td>always 0</td>
</tr>
<tr>
<td>0x13</td>
<td>WORD</td>
<td>not used by NTFS</td>
</tr>
<tr>
<td>0x15</td>
<td>BYTE</td>
<td>Media Descriptor</td>
</tr>
<tr>
<td>0x16</td>
<td>WORD</td>
<td>always 0</td>
</tr>
<tr>
<td>0x18</td>
<td>WORD</td>
<td>Sectors Per Track</td>
</tr>
<tr>
<td>0x1A</td>
<td>WORD</td>
<td>Number Of Heads</td>
</tr>
<tr>
<td>0x1C</td>
<td>DWORD</td>
<td>Hidden Sectors</td>
</tr>
<tr>
<td>0x20</td>
<td>DWORD</td>
<td>not used by NTFS</td>
</tr>
<tr>
<td>0x24</td>
<td>DWORD</td>
<td>not used by NTFS</td>
</tr>
<tr>
<td>0x28</td>
<td>LONGLONG</td>
<td>Total Sectors</td>
</tr>
<tr>
<td>0x30</td>
<td>LONGLONG</td>
<td>Logical Cluster Number for the file $MFT</td>
</tr>
<tr>
<td>0x38</td>
<td>LONGLONG</td>
<td>Logical Cluster Number for the file $MFTMirr</td>
</tr>
<tr>
<td>0x40</td>
<td>DWORD</td>
<td>Clusters Per File Record Segment</td>
</tr>
<tr>
<td>0x44</td>
<td>DWORD</td>
<td>Clusters Per Index Block</td>
</tr>
<tr>
<td>0x48</td>
<td>LONGLONG</td>
<td>Volume Serial Number</td>
</tr>
<tr>
<td>0x50</td>
<td>DWORD</td>
<td>Checksum</td>
</tr>
</tbody>
</table>

VBR BPB MANIPULATION

- What are the unused/null values??
- Variables left over from FAT file systems
- What happens if you enter a value?

<table>
<thead>
<tr>
<th>Byte Offset</th>
<th>Field Length</th>
<th>Field Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0B</td>
<td>WORD</td>
<td>Bytes Per Sector</td>
</tr>
<tr>
<td>0x0D</td>
<td>BYTE</td>
<td>Sectors Per Cluster</td>
</tr>
<tr>
<td>0x0E</td>
<td>WORD</td>
<td>Reserved Sectors</td>
</tr>
<tr>
<td>0x10</td>
<td>3 BYTES</td>
<td>Number of FATs &amp; Root Entries</td>
</tr>
<tr>
<td>0x13</td>
<td>WORD</td>
<td>Number of Sectors</td>
</tr>
<tr>
<td>0x15</td>
<td>BYTE</td>
<td>Media Descriptor</td>
</tr>
<tr>
<td>0x16</td>
<td>WORD</td>
<td>Sectors per FAT</td>
</tr>
<tr>
<td>0x18</td>
<td>WORD</td>
<td>Sectors Per Track</td>
</tr>
<tr>
<td>0x1A</td>
<td>WORD</td>
<td>Number Of Heads</td>
</tr>
<tr>
<td>0x1C</td>
<td>DWORD</td>
<td>Hidden Sectors</td>
</tr>
<tr>
<td>0x20</td>
<td>DWORD</td>
<td>Big number of Sectors</td>
</tr>
<tr>
<td>0x24</td>
<td>DWORD</td>
<td>Big sectors per FAT</td>
</tr>
<tr>
<td>0x28</td>
<td>LONGLONG</td>
<td>Total Sectors</td>
</tr>
<tr>
<td>0x30</td>
<td>LONGLONG</td>
<td>Logical Cluster Number for the file $MFT</td>
</tr>
<tr>
<td>0x38</td>
<td>LONGLONG</td>
<td>Logical Cluster Number for the file $MFTMirr</td>
</tr>
<tr>
<td>0x40</td>
<td>DWORD</td>
<td>Clusters Per File Record Segment</td>
</tr>
<tr>
<td>0x44</td>
<td>DWORD</td>
<td>Clusters Per Index Block</td>
</tr>
<tr>
<td>0x48</td>
<td>LONGLONG</td>
<td>Volume Serial Number</td>
</tr>
<tr>
<td>0x50</td>
<td>DWORD</td>
<td>Checksum</td>
</tr>
</tbody>
</table>

VBR BPB MANIPULATION

- Number of FAT copies
  - Read sectors further into disk than should
- Root directory entries
  - Used in same was as above, just by different set of VBRs, read sectors further into disk than should
- Sectors Per Fat
  - Can cause VBR to read more than one sector first time through read loop, making it read more than 15-sectors from disk
**VBR BPB MANIPULATION (TOKEN IDA SLIDE)**

```assembly
read_IPL_from_disk proc near
  ; CODE XREF: seg000:00CF1p
  pushad
  push ds
  push es
  push bp
  mov eax, dword ptr ds:vbr.bpb.max_root_dir_entries_zero
  add eax, ds:vbr.bpb.hidden_sectors
  push large offset vbr
  push eax
  push es
  push bx
  push bx
  push 10h
  mov ah, 42h
  mov dl, ds:vbr.bpb.reserved1
  mov ss
  pop ds
  mov si, sp
  int 13h
  ; DISK - IBM/MS Extension - EXTENDED READ (DL - drive
  pop ecx
  pop bx
  pop dx
  pop ecx
  pop ds
  jnb error_strings
  inc dword ptr ds:vbr.bpb.max_root_dir_entries_zero
  add dx, word ptr ds:vbr.bpb.reserved2 ; new sector offset in memory
  push dh
  push dl
  dec ds:vbr.bpb.sectors_per_FAT_zero
```
QUESTIONS?
REFERENCES

http://thestarman.pcministry.com/asm/mbr/W7MBR.htm