FireEye

Boot What?

Why tech invented by IBM in 1983 is relevant today

June 22rd, 2017

SECURITY REIMAGINED

$ WHOAMI

- Christopher Glyer
- Chief Security Architect, FireEye

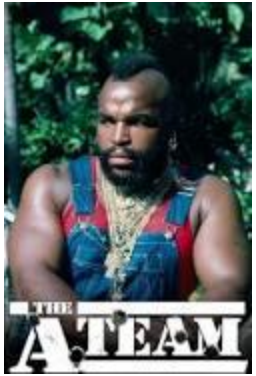
- Incident Responder
- Forensic Analyst
- Wanna-be sailor

MBR/VBR BOOT ANTICS

- Disks, Partitions and Volumes
- Boot like it's 1983!
- Known Attack Vectors
- FIN1 Bootcode Case Study
- Results at Scale

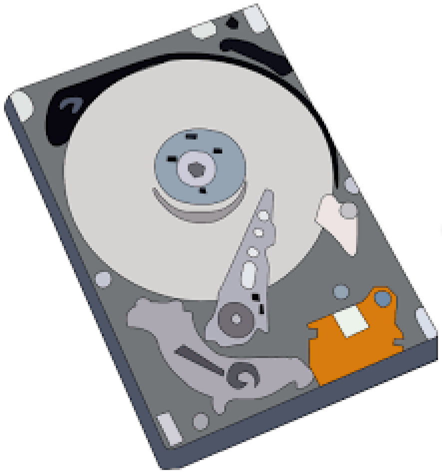
BACK IN 1983…

- 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



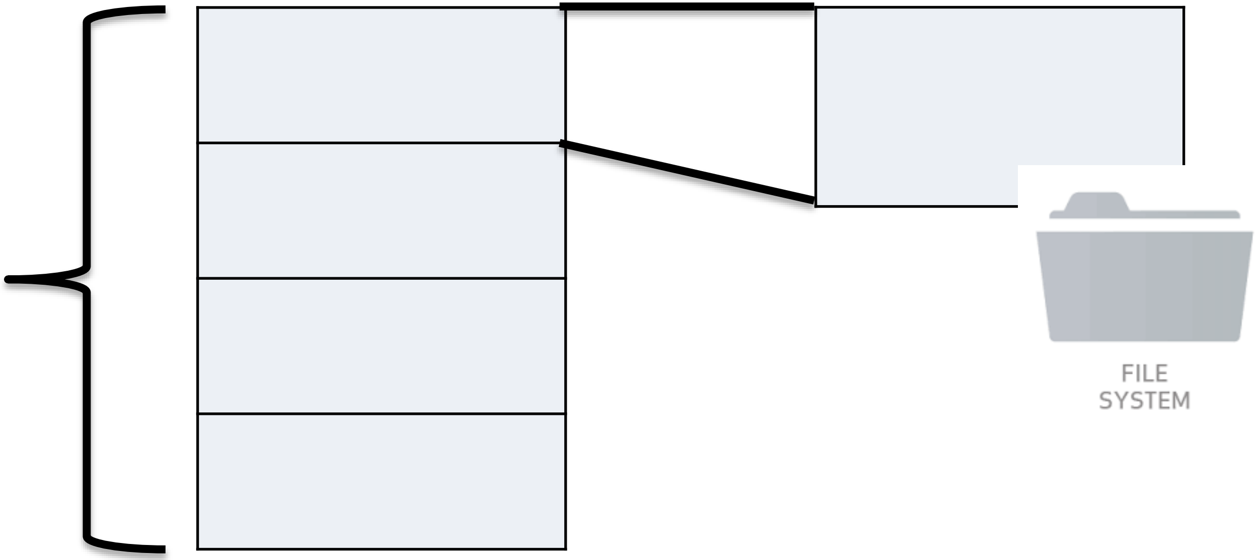
DISKS, PARTITIONS, VOLUMES, OH MY!

Disk



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

Partitions



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

Volume

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

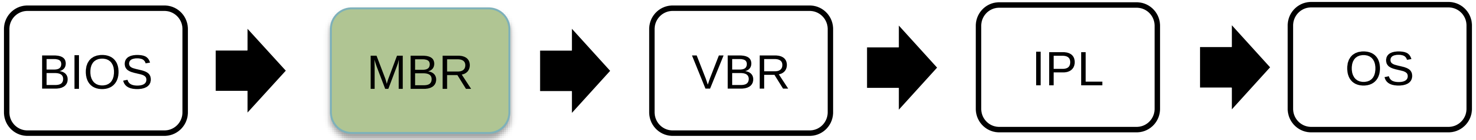
BOOT LIKE IT'S 1983

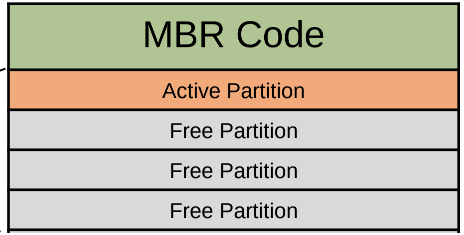
BIOS → MBR → VBR → IPL → OS

MBR

- POST
- Find the active d
- Load first sector MBR) into **0x7C**
- Execute MBR

BOOT LIKE IT’S 1983

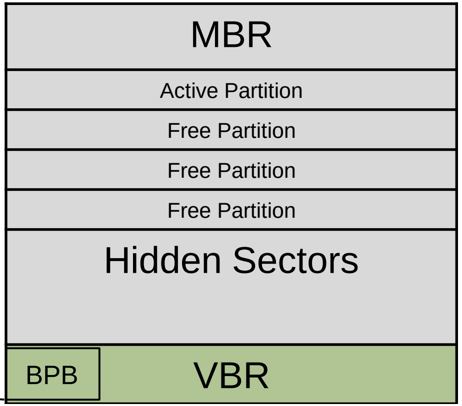




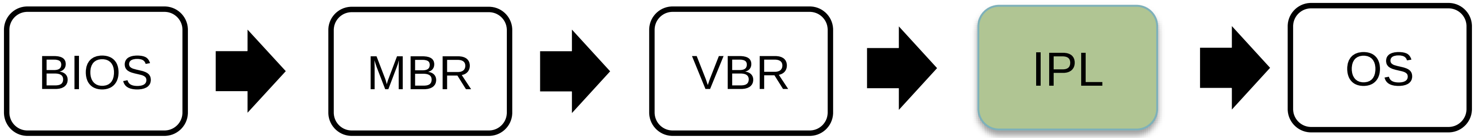
- **First sector of disk**
- Loaded at 0x7C00, but relocates itself to load VBR
- Mix of code and data (partition table)
- Executes in 16-bit real-mode
- Reads Partition Table
    - Locates active partition
- Loads first sector of active partition (512-bytes VBR) into 0x7C00
- Executes VBR

BOOT LIKE IT'S 1983

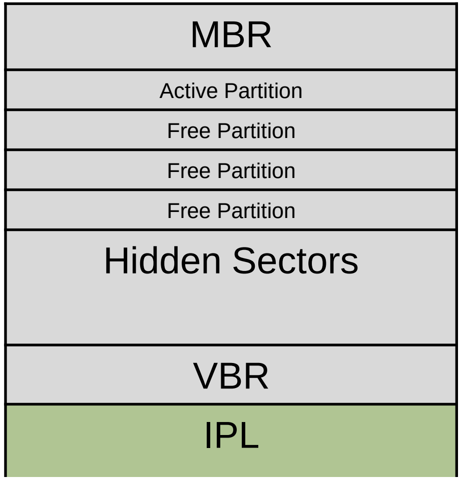
- **First sector of active partition**
- Loaded at 0x7C00
- Mix of code and data (BPB)
- Executes in 16-bit real-mode
- Loads IPL, 15-sectors after VBR
- Executes IPL



BOOT LIKE IT’S 1983

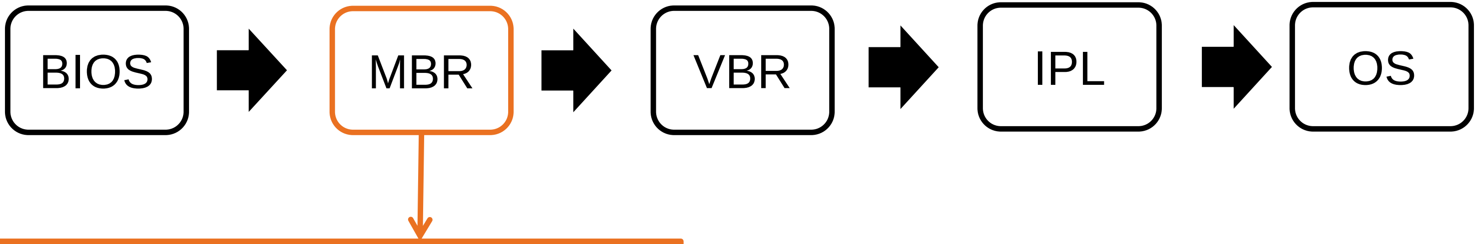


- **15-sectors following VBR**
- Frequently loaded at 0x7E00 (after VBR) or 0xD000
- Starts in 16-bit real-mode, but transitions to protected-mode
- Executes NTLDR/BOOTMGR



KNOWN ATTACK VECTORS

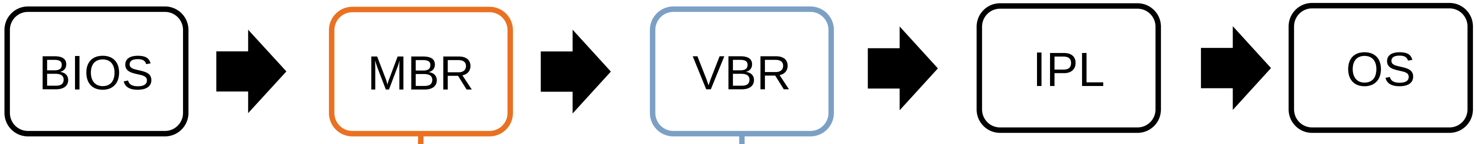
KNOWN ATTACK VECTORS



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

https://www.virusbulletin.com/uploads/pdf/conference/vb2014/VB2014-RodionovMatrosov.pdf

KNOWN ATTACK VECTORS

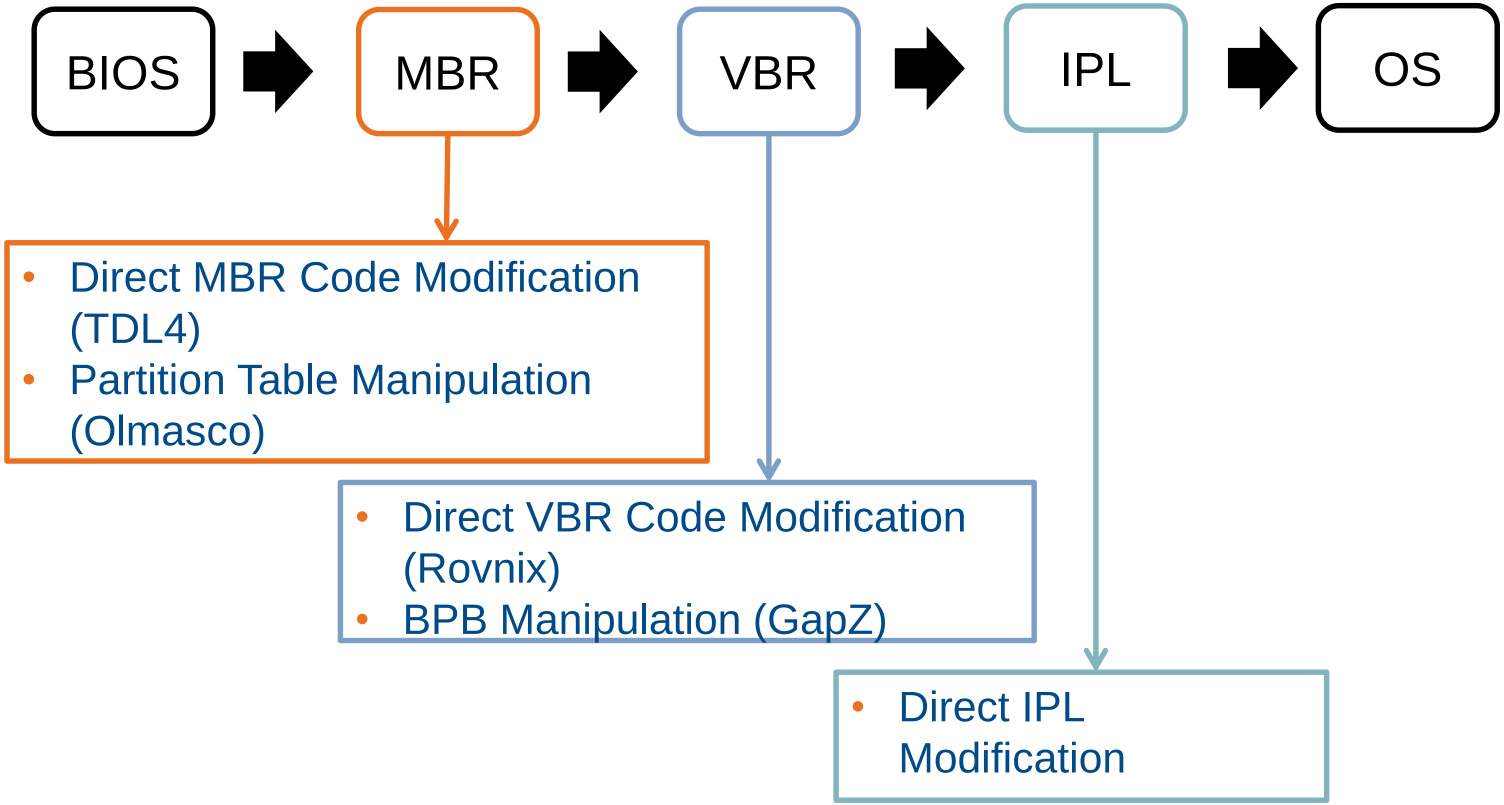


- Direct MBR Code Modification (TDL4)
- Partition Table Manipulation (Olmasco)

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

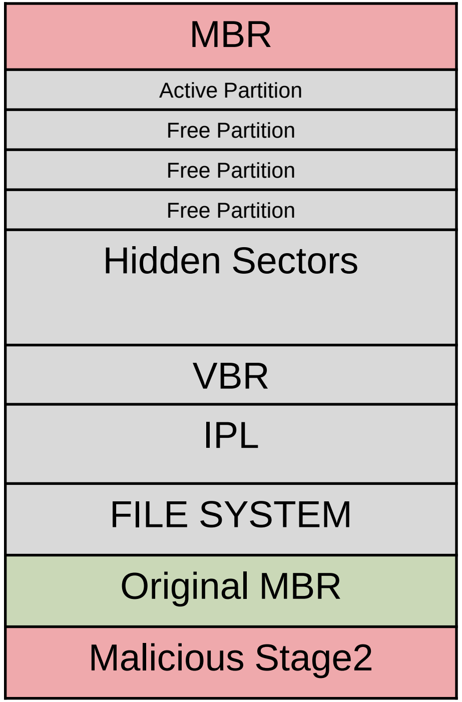
https://www.virusbulletin.com/uploads/pdf/conference/vb2014/VB2014-RodionovMatrosov.pdf

KNOWN ATTACK VECTORS



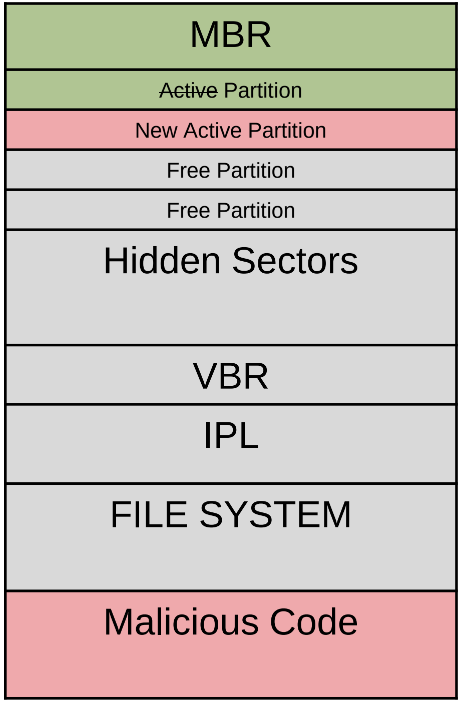
https://www.virusbulletin.com/uploads/pdf/conference/vb2014/VB2014-RodionovMatrosov.pdf

MBR REPLACEMENT



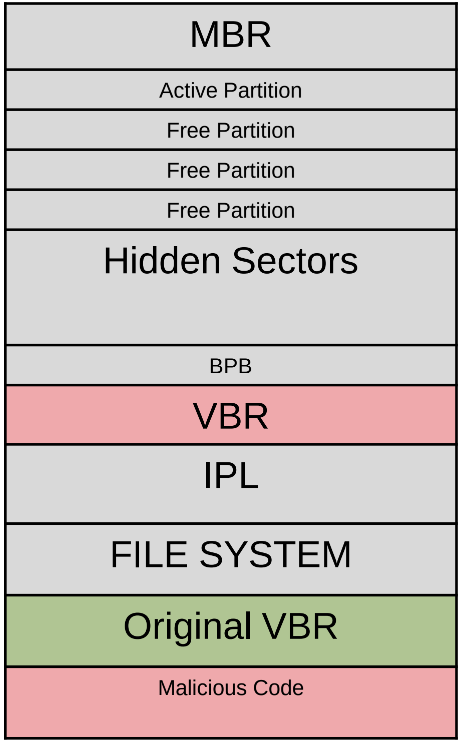
- Replace MBR with malicious code
- Samples:
  - TDL4 – hooks FS DEVICE_OBJ to hide modifications
  - MebRoot – hooks disk.sys to hide modifications
  - XPAJ

PARTITION ADDITION



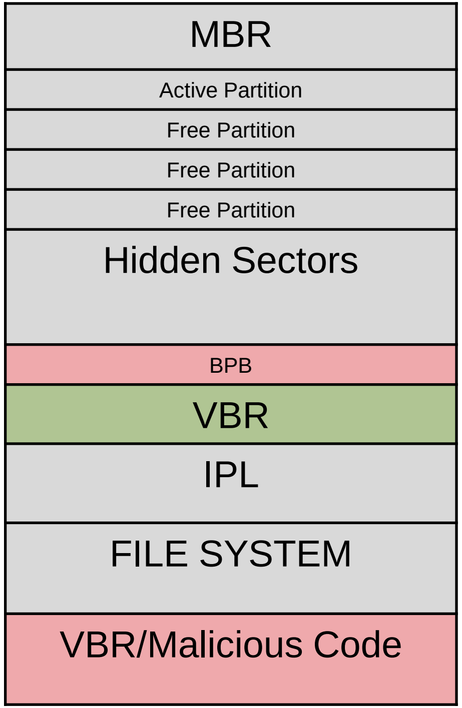
- Replace, modify or add partition table entries
- Samples:
  - Olmasco

VBR REPLACEMENT



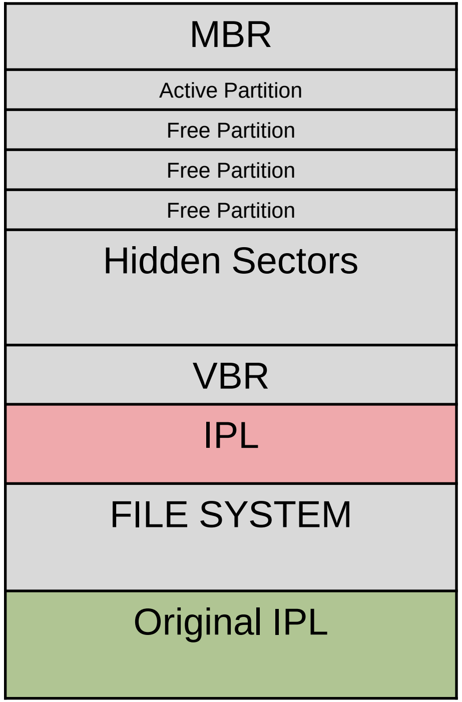
- Replace VBR with malicious code
- Samples:
  - Rovnix (Cidox)
  - BOOTRASH (nemesis – VBR+IPL)

VBR BPB MANIPULATION



- 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

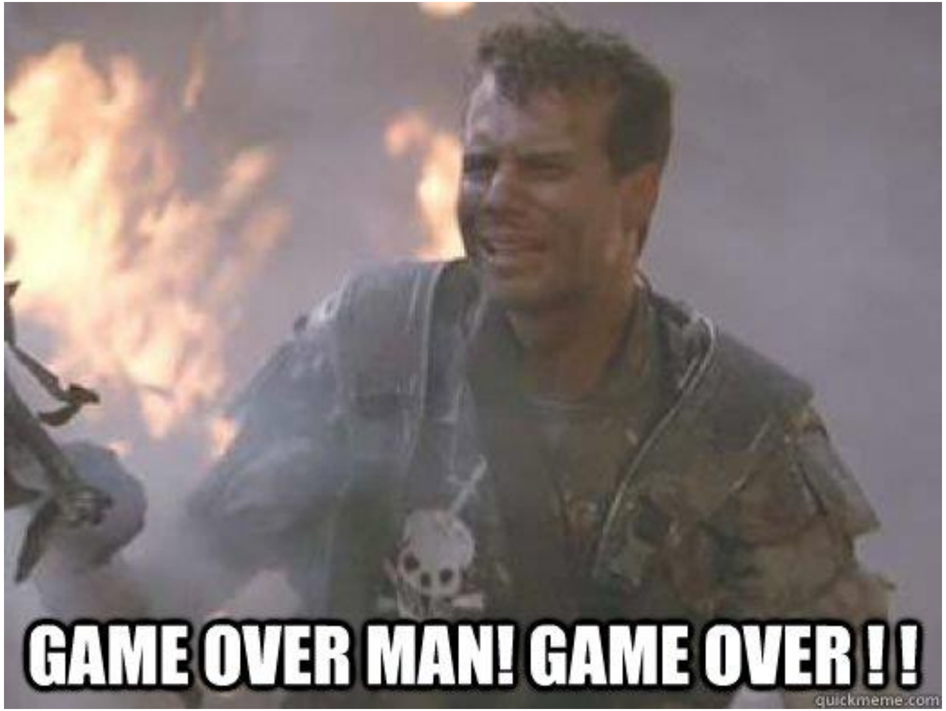
- Investigation identified system beaconing to bad domain
- Need to find the code that launched the backdoor
- Searched common persistence mechanisms
  - Services
  - Run keys
  - Scheduled tasks
  - Startup folders

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!**

```
Update
   Update nemesis
se !nmsupdate <file> in powerterminal to upload file
   <f / file> <FileName> [/r] [/nd]
       <FileName>    File on hdd to take updates from
       [/r]          Restart nemesis after update
        [/nd]         Do not delete update package after update
        [/flt:x]      File filter: vbr, boot, core, vfs, nmsdrv, inj, ldr, nms, dwml, all (default - all
   <Flush>     Flush updates to disk
   <Drop>      Drop updates
   <ru>        Initiate userland restart
   <rf>        Initiate full restart
    <create> <OutFileName> [/src:dir] [/vfs] [/flt:x] Create update package from files on disk or from y
       <OutFileName> Output update package name
       [/src:dir]    Optional directory with update files
       [/vfs]        Make it from current version on vfs
       [/y]          Overwrite output file if it exists
```

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]

MBR @SCALE - RESULTS

6663 unique MBR hashes across **~265K** systems

MBR @SCALE - RESULTS

Why >6000 MBR Hashes?

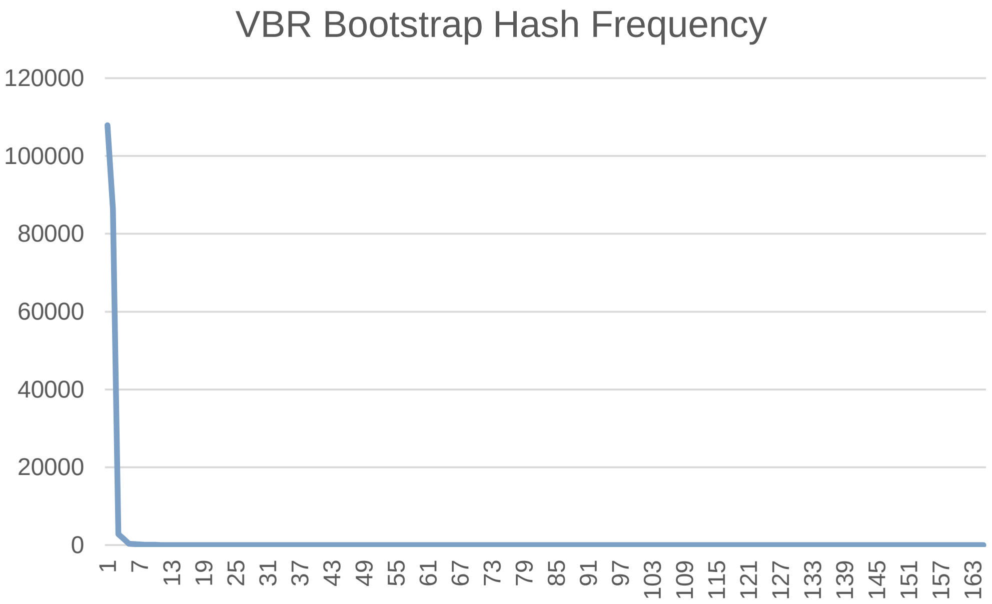
- Numerous legitimate applications modify MBR (Altiris, SafeBoot, PGPGuard…)
- Lots of minor variations to known good. (ex: jmp instructions differing lengths)
- Strings “opErating system”
- MBR backup utilities – stores multiple backup copies of MBR and loads
- Loads 4 sectors of VBR instead of 1

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

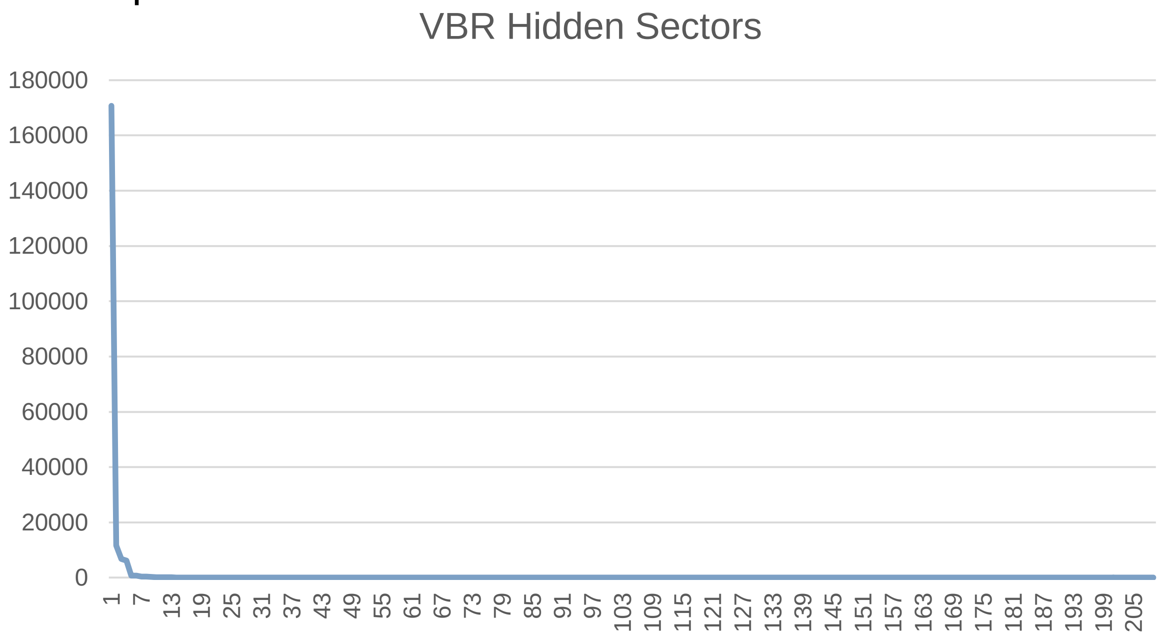
- **VBR bootstrap hashes**
  - Currently stacks quite well with 165 unique VBR hashes across ~265K systems
  - Vast majority of infrequent hashes have “мProtect!” in the header



Hash	Frequency
EBA2D2E048E3A1FA0DE78F454B628C12	107894
D63F30B9E6D54924701D1AEBA0887C95	86367
E7A27EB31DA26173C18D58F000F74B07	2839
A959459A09BC3C7A1E8C9BF2212741D1	1620
88304B05661A0BE258B002A25F84A13D	360
3B9BC8D3DD94BC7367A85B677A14827B	226
3447D07FE4470BB1AB1A8EA8EF888199	155
2D5423E2A9AF88B0AB18607FC21DF245	108

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

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

NTFS BIOS Parameter Block - https://technet.microsoft.com/en-us/library/cc977221.aspx

VBR: BIOS PARAMATER BLOCK (BPB) OVERVIEW

- What are the unused/null values??

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

NTFS BIOS Parameter Block - https://technet.microsoft.com/en-us/library/cc977221.aspx

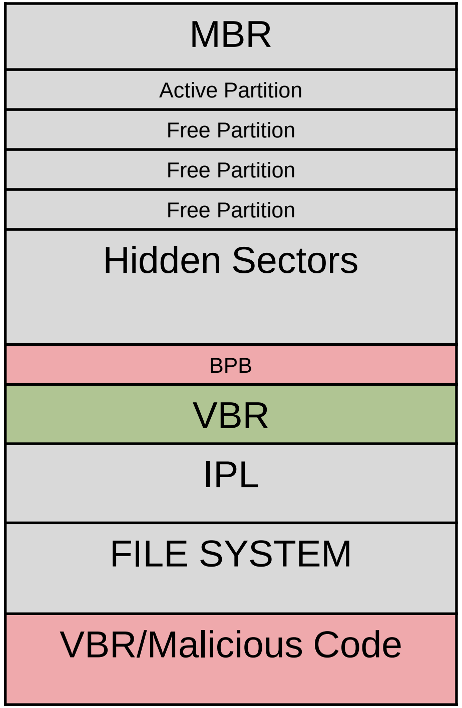
VBR BPB MANIPULATION

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

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

NTFS BIOS Parameter Block - https://technet.microsoft.com/en-us/library/cc977221.aspx

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)

```
                read_IPL_from_disk proc near            ; CODE XREF: seg000:00CF↑p
66 60                           pushad
1E                              push    ds
06                              push    es

                loc_121:                                ; CODE XREF: read_IPL_from_disk+46↓j
66 A1 11 00                     mov     eax, dword ptr ds:vbr.bpb.max_root_dir_entries_zero
66 03 06 1C 00                  add     eax, ds:vbr.bpb.hidden_sectors
1E                              push    ds
66 68 00 00 00+                 push    large offset vbr
66 50                           push    eax             ; sector offset to read
06                              push    es
53                              push    bx              ; memory offset
68 01 00                        push    1
68 10 00                        push    10h
B4 42                           mov     ah, 42h ; 'B'
8A 16 0E 00                     mov     dl, ds:vbr.bpb.reserved1
16                              push    ss
1F                              pop     ds
8B F4                           mov     si, sp          ; DAP is on stack
CD 13                           int     13h             ; DISK - IBM/MS Extension - EXTENDED READ (DL - drive
66 59                           pop     ecx
5B                              pop     bx
5A                              pop     dx
66 59                           pop     ecx
66 59                           pop     ecx
1F                              pop     ds
0F 82 16 00                     jb      error_strings
66 FF 06 11 00                  inc     dword ptr ds:vbr.bpb.max_root_dir_entries_zero
03 16 0F 00                     add     dx, word ptr ds:vbr.bpb.reserved2 ; new sector offset in memory
8E C2                           mov     es, dx
FF 0E 16 00                     dec     ds:vbr.bpb.sectors_per_FAT_zero
```

QUESTIONS?

REFERENCES

https://en.wikipedia.org/wiki/BIOS_parameter_block

https://www.f-secure.com/weblog/archives/00001393.html

https://recon.cx/2013/slides/Recon2013-Aleksandr%20Matrosov%20and%20Eugene%20Rodionov-Reconstructing-Gapz%20Position-Independent%20Code%20Analysis%20Problem.pdf

http://www.welivesecurity.com/2012/01/03/bootkit-threat-evolution-in-2011-2/

https://www.blackhat.com/docs/us-14/materials/us-14-Haukli-Exposing-Bootkits-With-BIOS-Emulation-WP.pdf

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