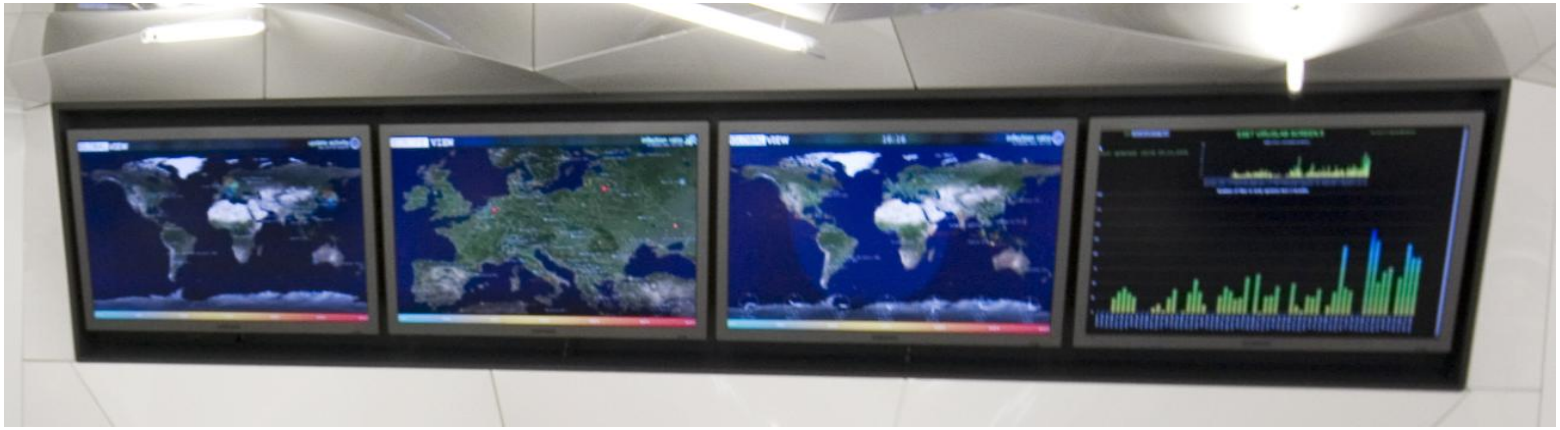
Defeating x64:
Modern Trends of Kernel-Mode Rootkits

Aleksandr Matrosov

Eugene Rodionov

Who we are?

- **Malware researchers at ESET**
  - **rootkits analysis**
  - **development of cleaning tools**
  - **tracking new rootkit techniques**
  - **investigation of cybercrime groups**



http://www.joineset.com/

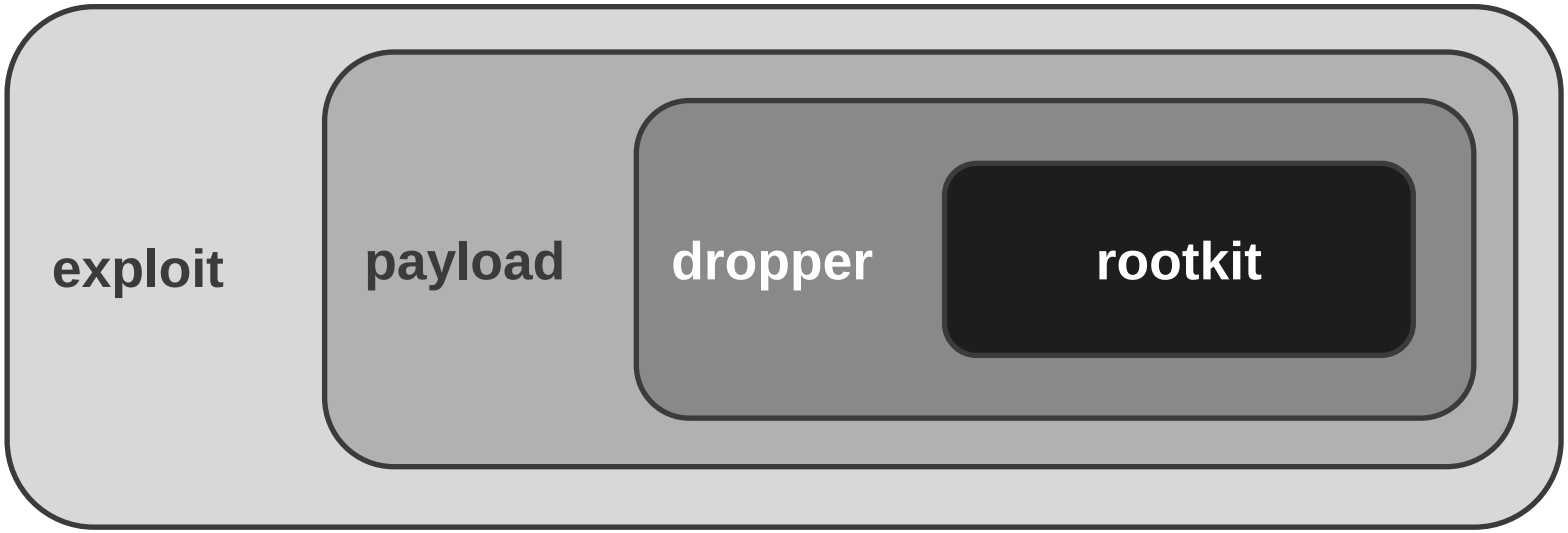
Agenda

- ✓ Evolution of payloads and rootkits
- ✓ Bypassing code integrity checks
- ✓ Attacking Windows Bootloader
- ✓ Modern Bootkit details:
  - Win64/Olmarik
  - Win64/Rovnix
- ✓ How to debug bootkit with Bochs emulator

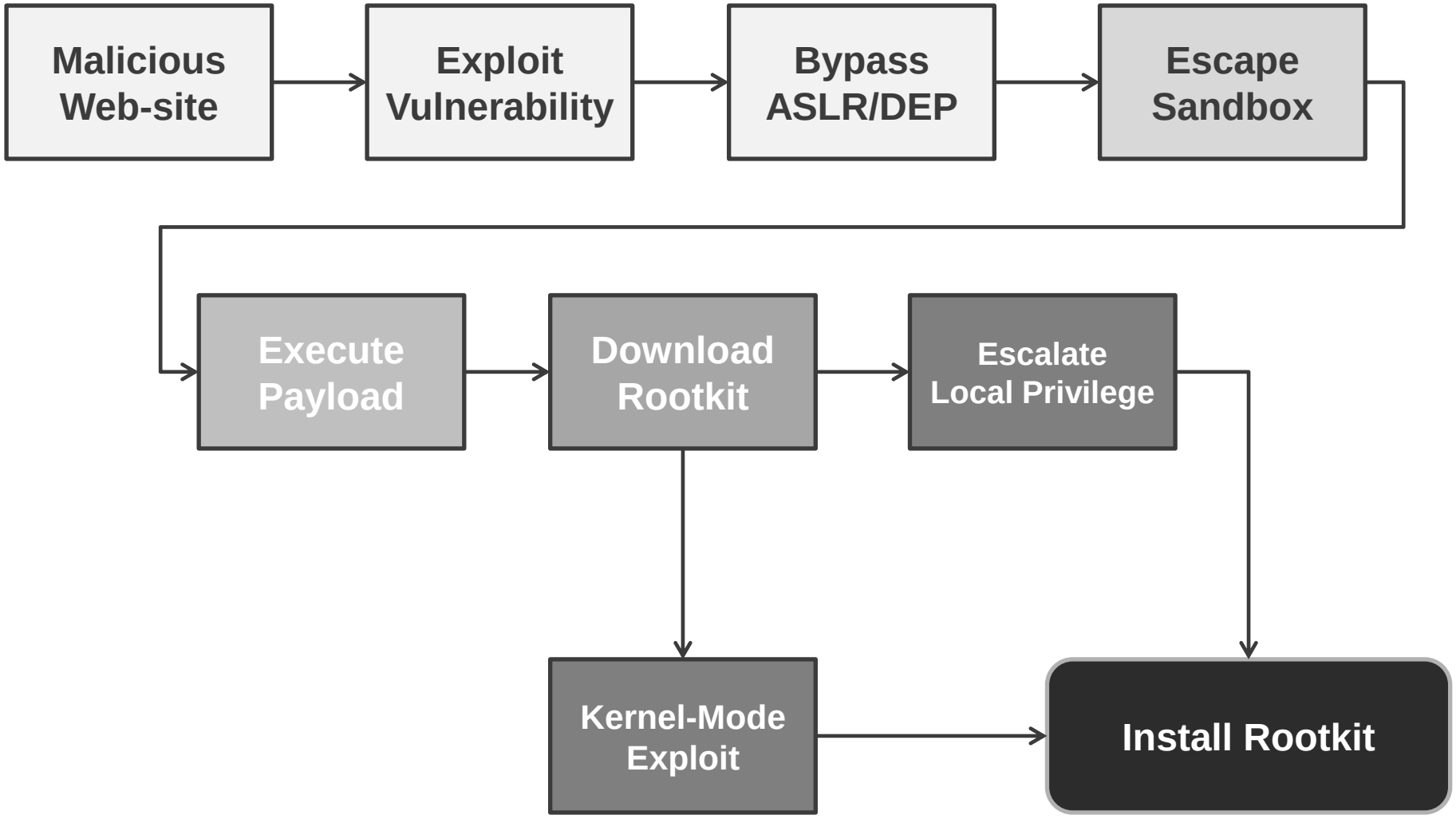
- ✓ HiddenFsReader as a forensic tool

Evolution of Rootkits

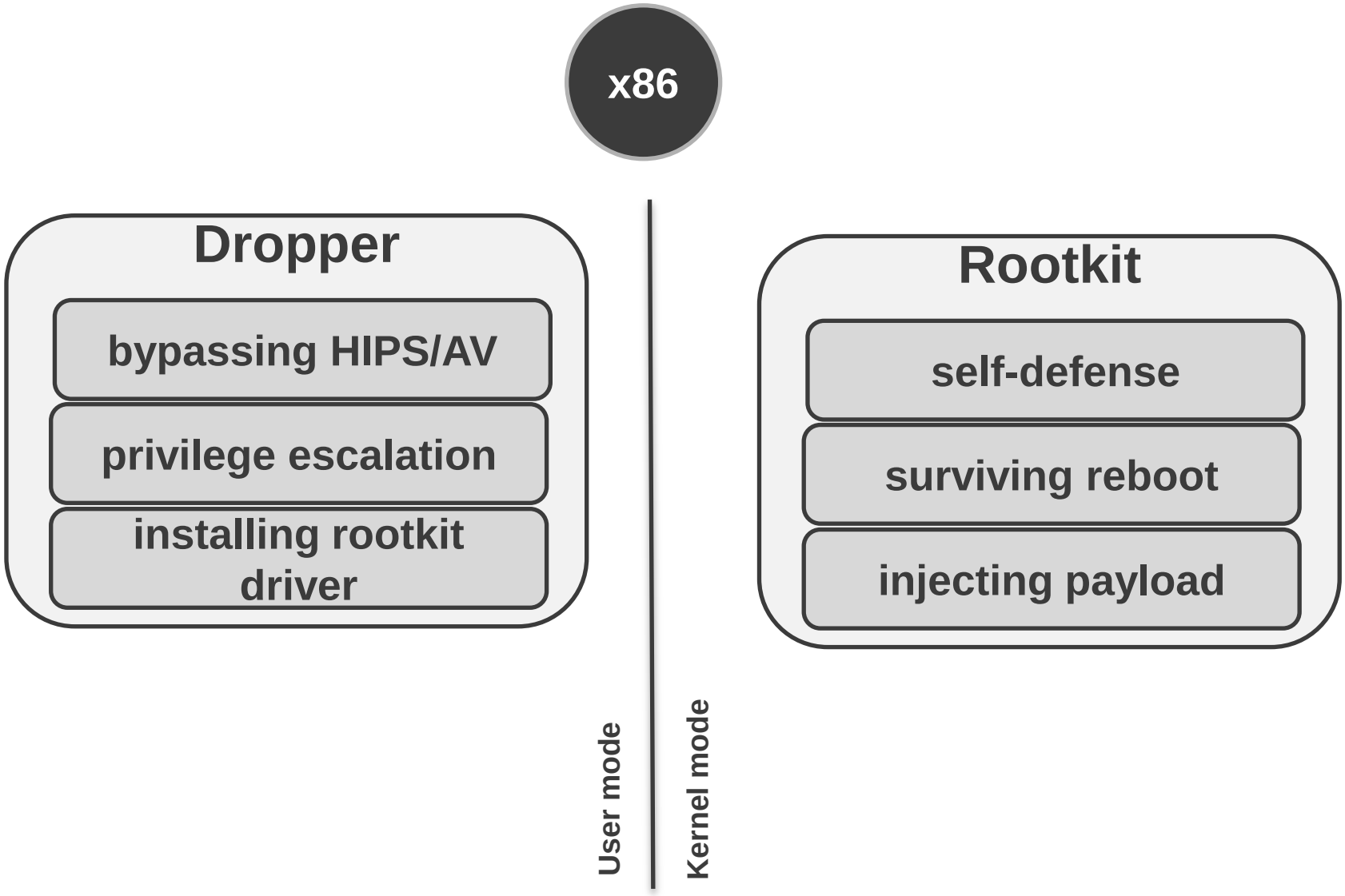
Evolution of Rootkit Installation



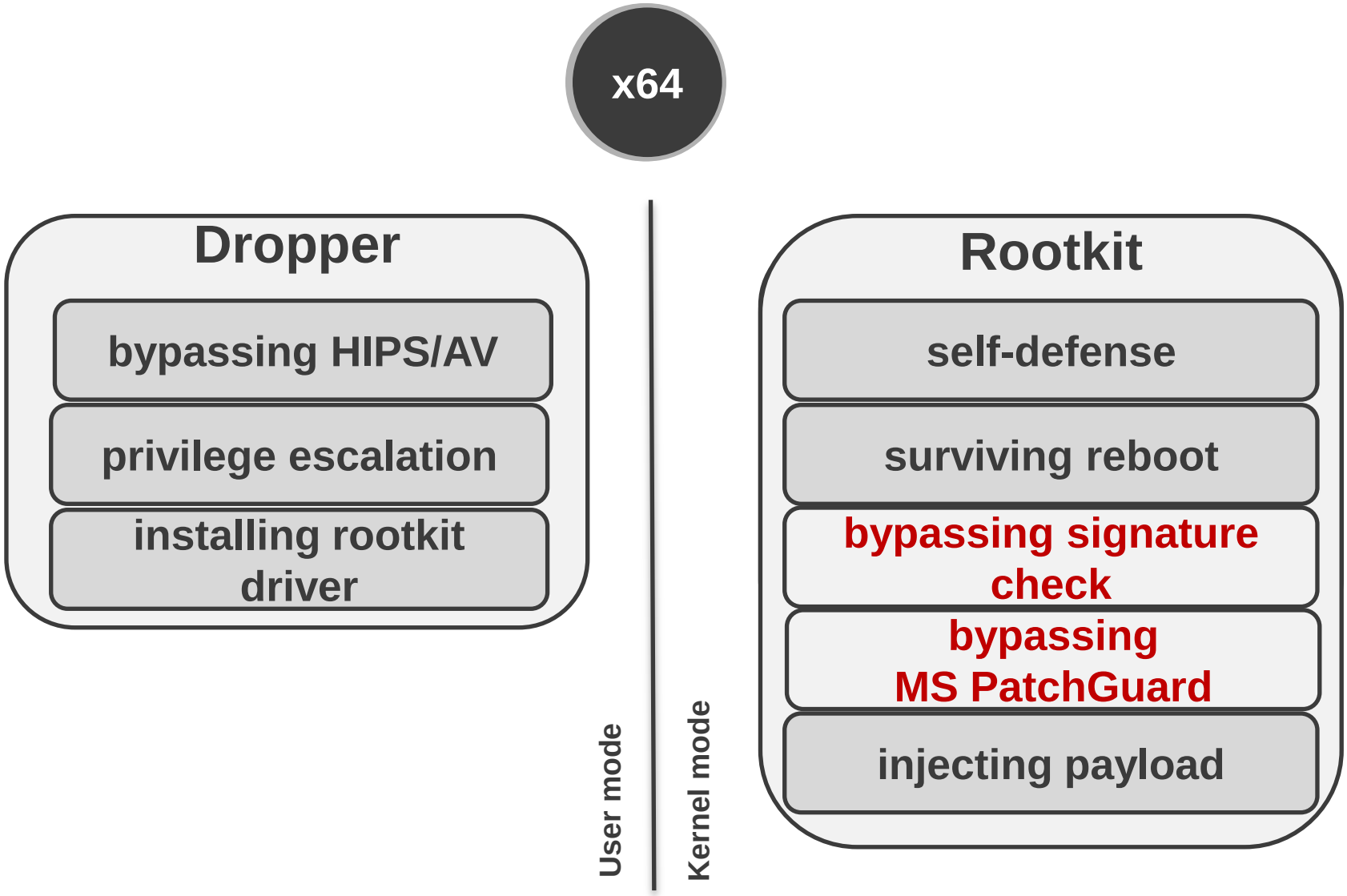
Evolution of Rootkit Installation



Evolution of Rootkit Features



Evolution of Rootkit Features



Obstacles for 64-bit Rootkits

- **Kernel-Mode Code Signing Policy:**
  - ✓ **It is “difficult” to load unsigned kernel-mode driver**
- **Kernel-Mode Patch Protection (Patch Guard):**
  - ✓ **SSDT (System Service Dispatch Table)**
  - ✓ **IDT (Interrupt Descriptor Table)**
  - ✓ **GDT (Global Descriptor Table)**
  - ✓ **MSRs (Model Specific Registers)**

Bypassing Code Integrity Checks

Types of Integrity Checks

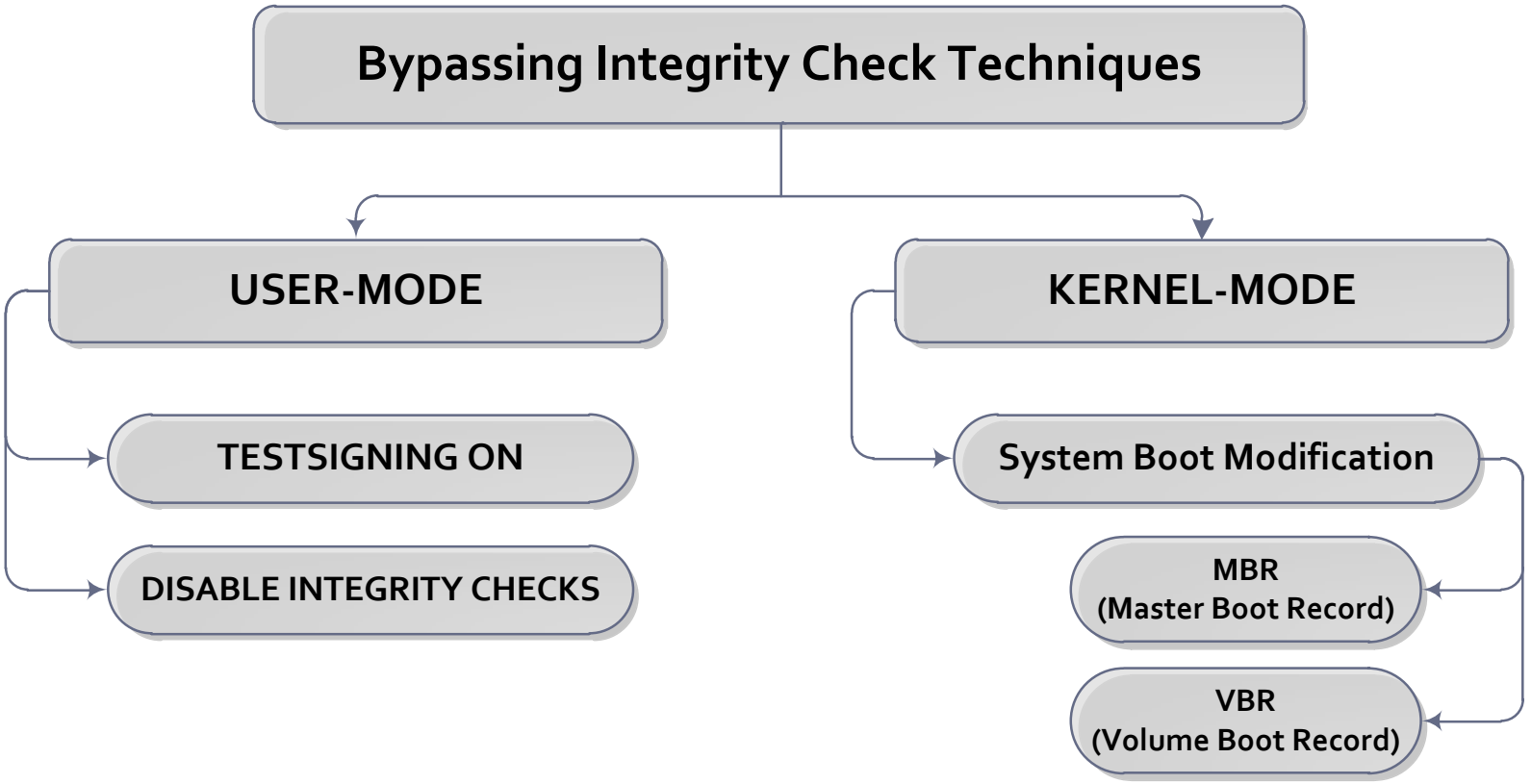
- PnP Device Installation Signing Requirements
- Kernel-Mode Code Signing Policy
  - ✓Enforced on 64-bit version of Windows Vista and later versions

	64-bit Windows Vista and later	32-bit Windows Vista and later
Boot-start driver	☑	☑
Non boot-start PnP driver	☑	☒
Non boot-start, non-PnP driver	☑	☒ (except stream protected media drivers)

Subverting KMCSP

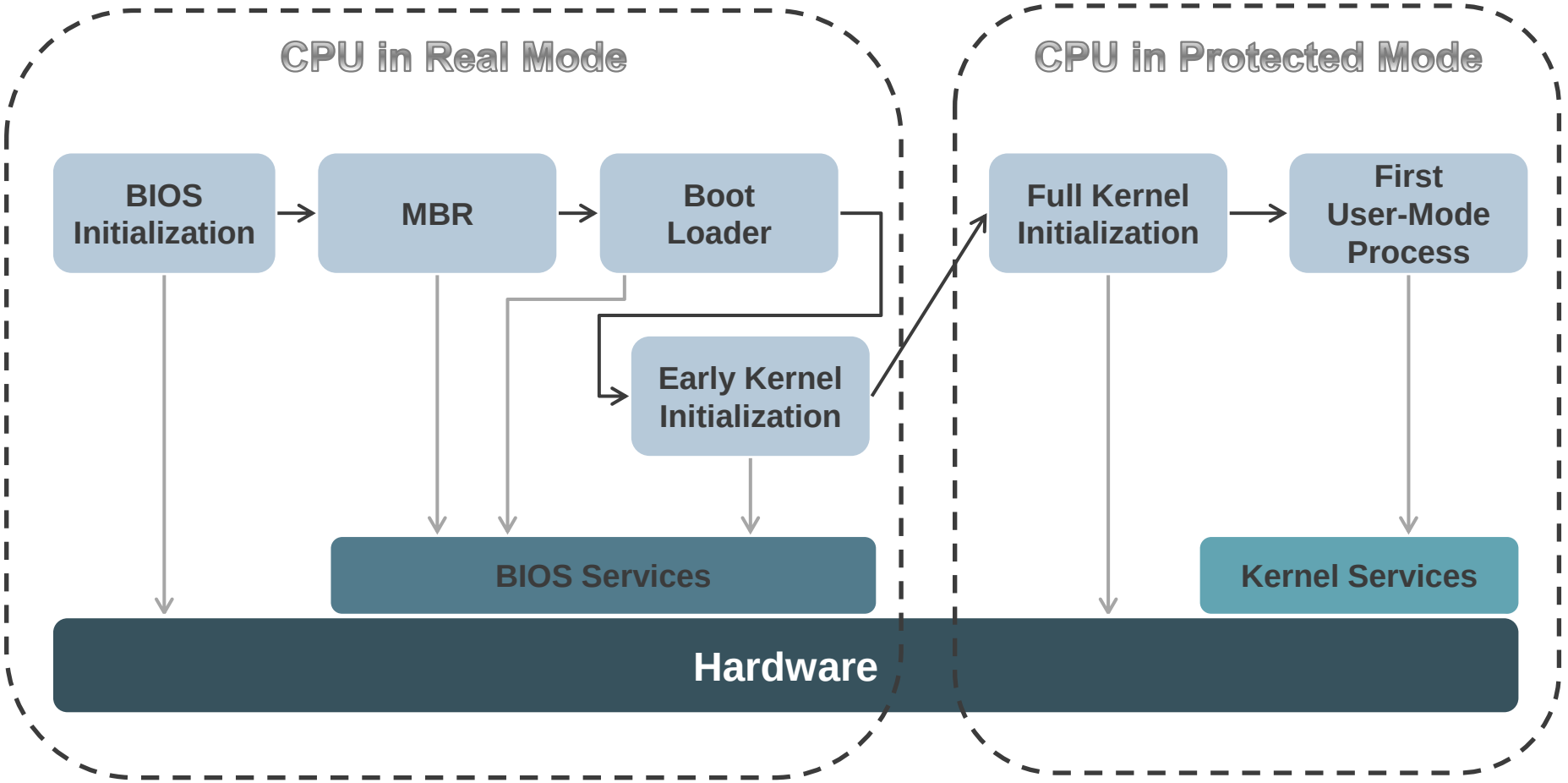
- Abusing vulnerable signed legitimate kernel-mode driver
- Switching off kernel-mode code signing checks by altering BCD data:
  - ✓abusing WinPe Mode
  - ✓disabling signing check
  - ✓enabling test signing
- Patching Bootmgr and OS loader

Bypassing Integrity Checks

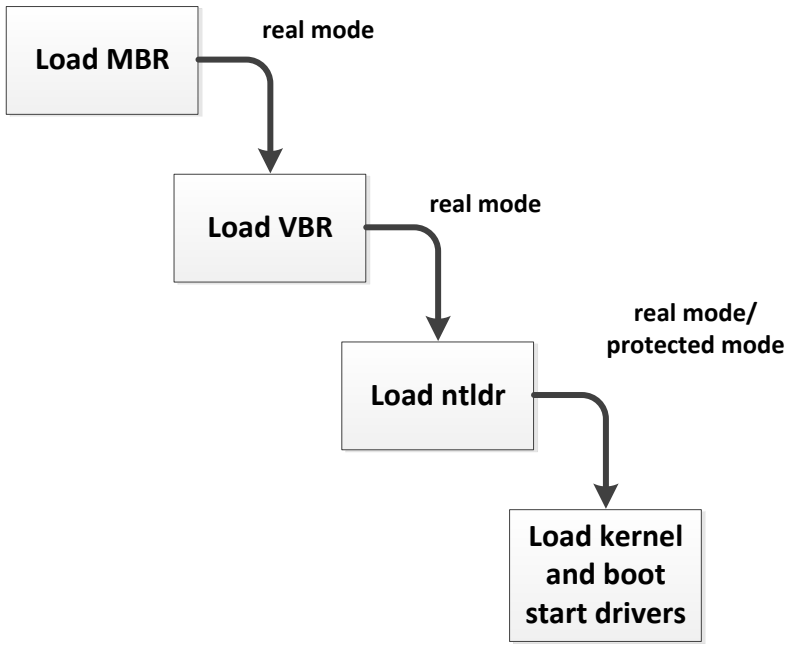


Attacking Windows Bootloader

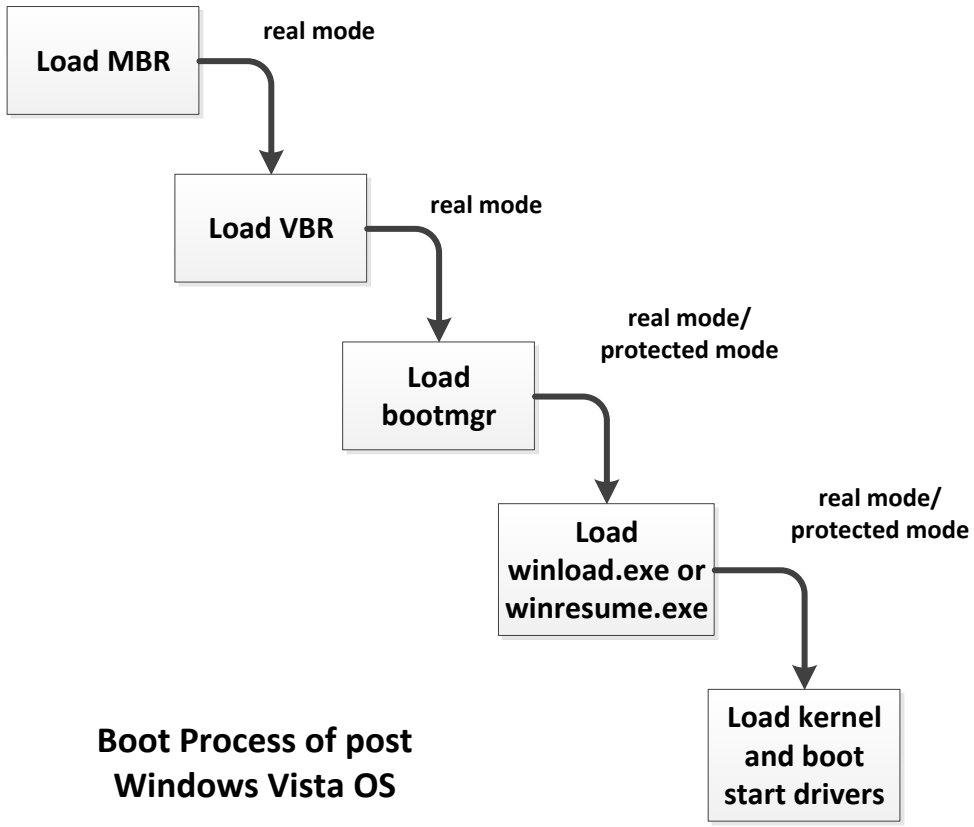
Boot Process



Boot Process of Windows OS



Boot Process of pre Windows Vista OS

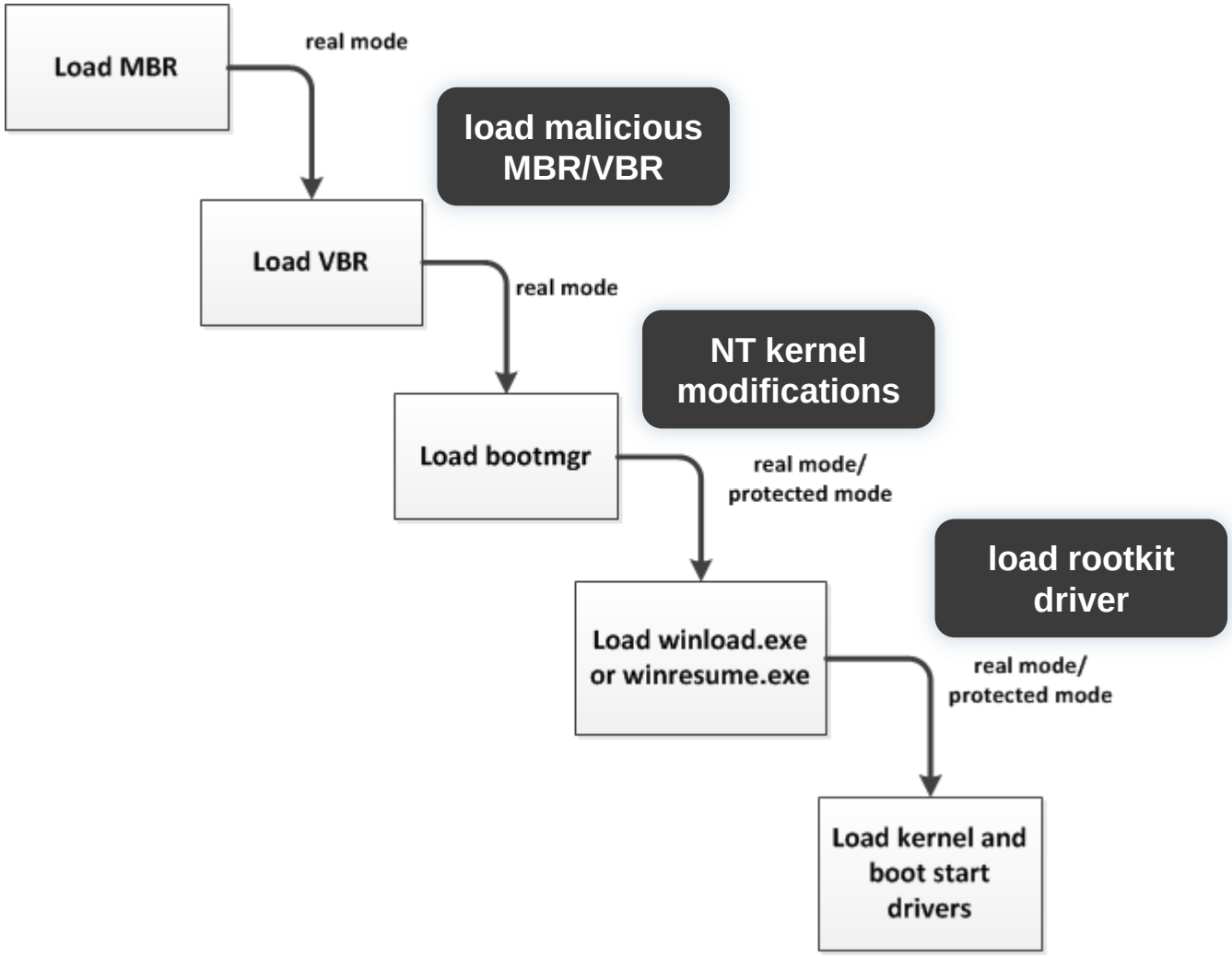


Boot Process of post Windows Vista OS

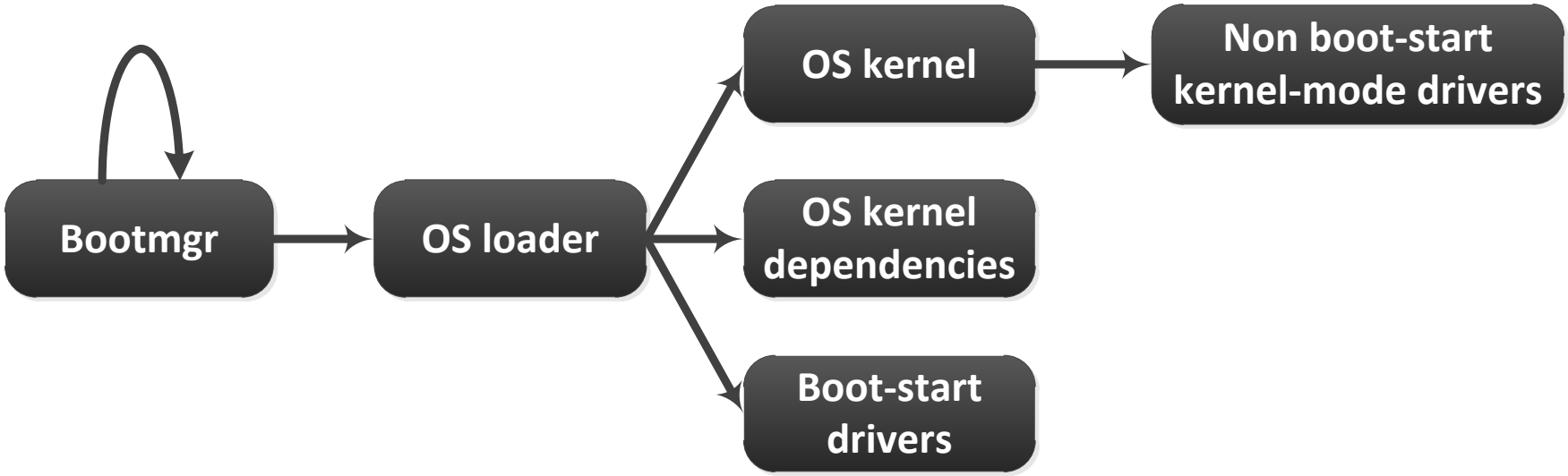
MBR – Master Boot Record

VBR – Volume Boot Record

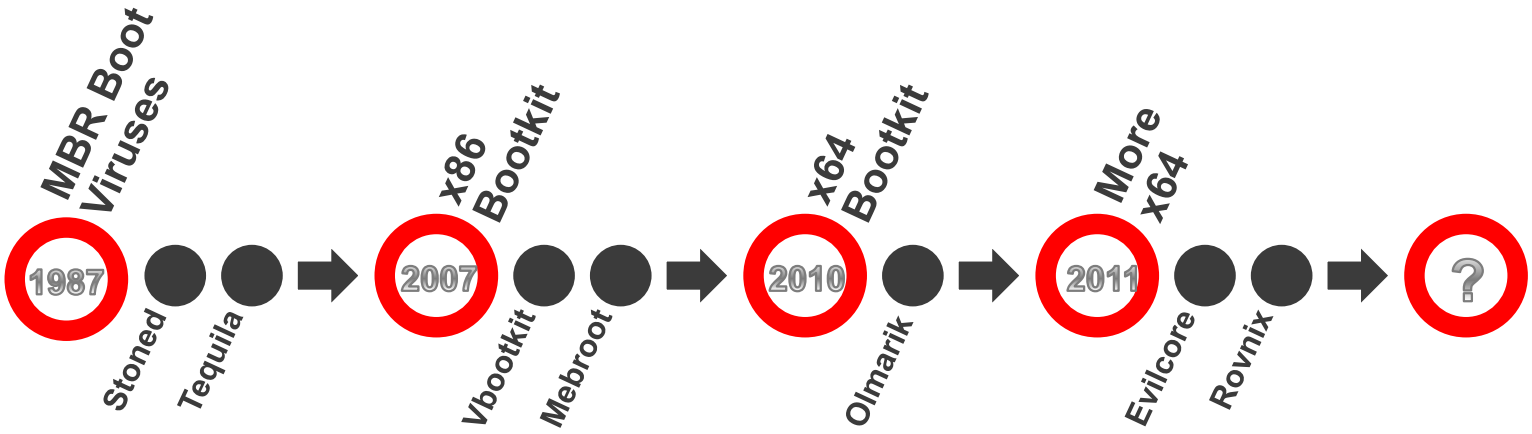
Boot Process with Bootkit Infection



Code Integrity Check



Evolution of Bootkits



- **Bootkit PoC evolution:**
  - ✓ **eEye Bootroot (2005)**
  - ✓ **Vbootkit (2007)**
  - ✓ **Vbootkit v2 (2009)**
  - ✓ **Stoned Bootkit (2009)**
  - ✓ **Evilcore x64 (2011)**

- **Bootkit Threats evolution:**
  - ✓ **Win32/Mebroot (2007)**
  - ✓ **Win32/Mebratix (2008)**
  - ✓ **Win32/Mebroot v2 (2009)**
  - ✓ **Win64/Olmarik (2010/11)**
  - ✓ **Win64/Rovnix (2011)**

Win64/Olmarik

wanted

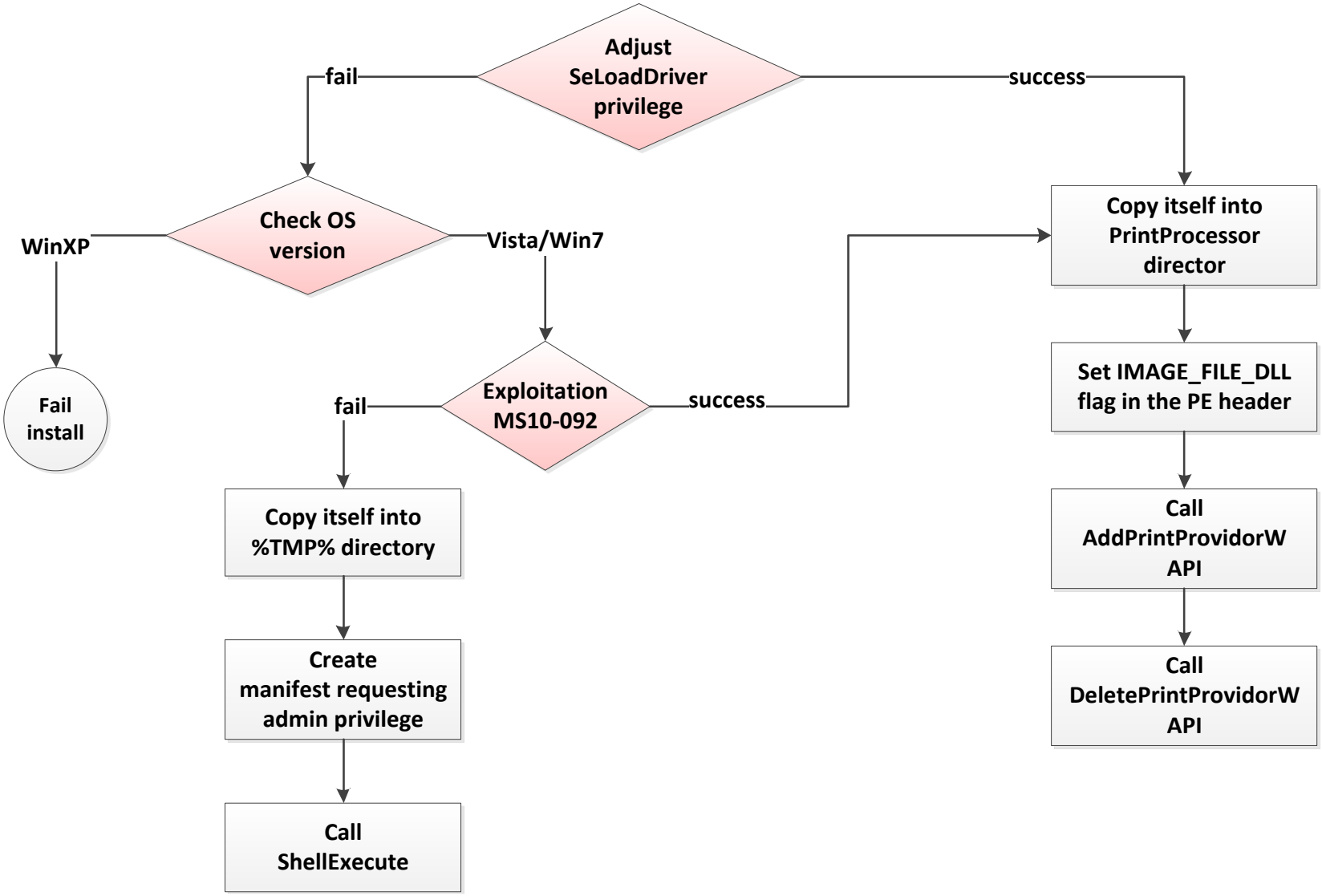
DEAD OR ALIVE

wanted

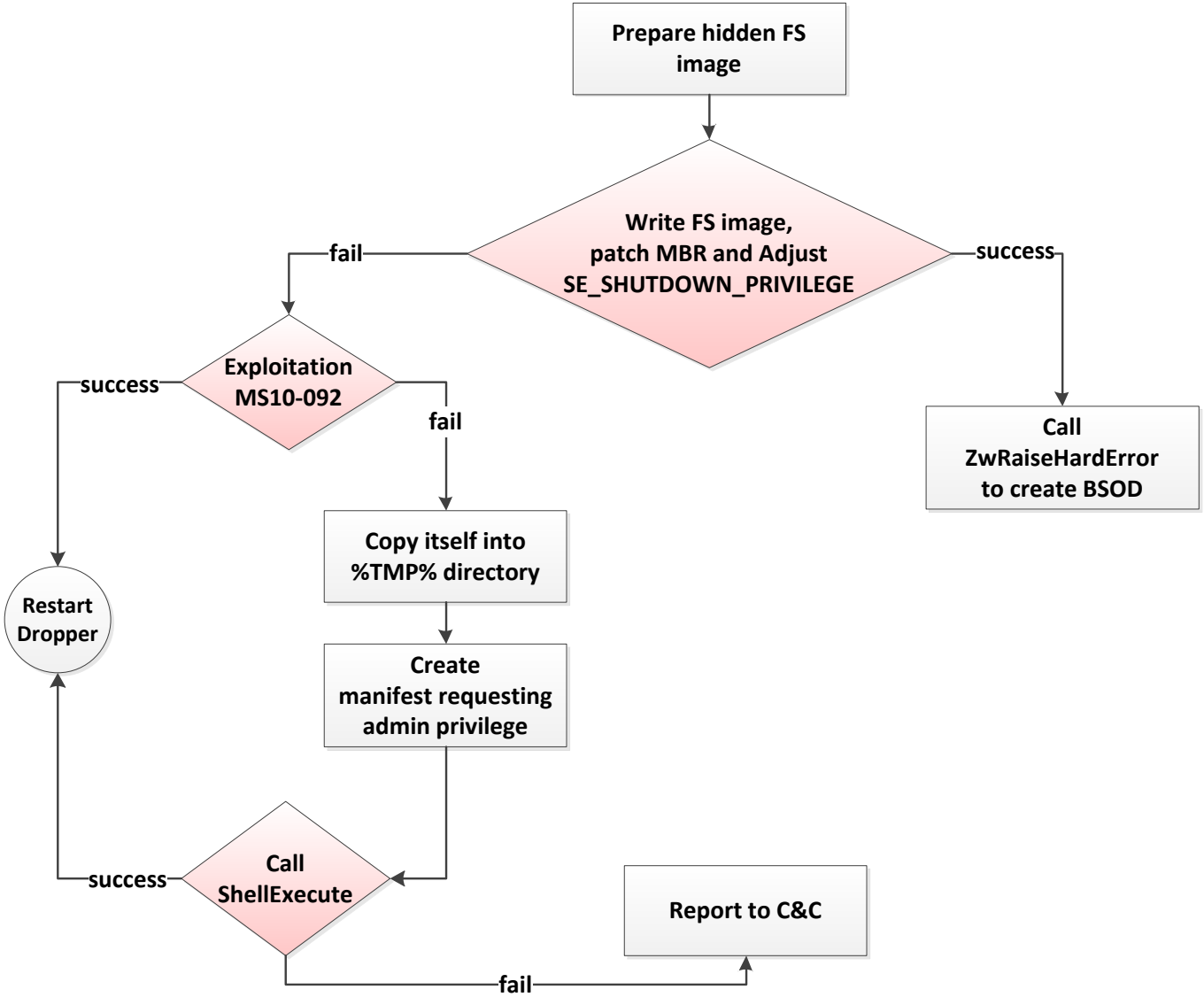
DEAD OR ALIVE

Installation on x86 vs. x64

TDL4 Installation on x86

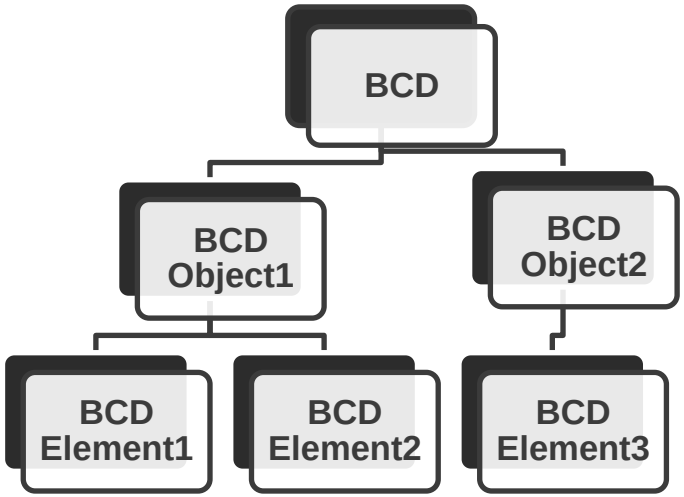


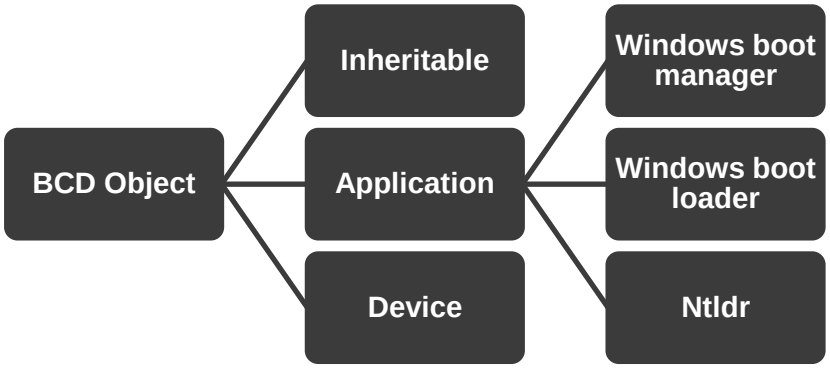
TDL4 Installation on x64



Boot Configuration Data (BCD)

- HKEY_LOCAL_MACHINE
  - BCD00000000
    - Description
    - Objects
      - {0ce4991b-e6b3-4b16-b23c-5e0d9250e5d9}
      - {1afa9c49-16ab-4a5c-901b-212802da9460}
      - {4636856e-540f-4170-a130-a84776f4c654}
      - {5189b25c-5558-4bf2-bca4-289b11bd29e2}
      - {6567fef2-0e15-11e0-a000-cb5e17119bb9}
      - {6567fef3-0e15-11e0-a000-cb5e17119bb9}
      - {6567fef4-0e15-11e0-a000-cb5e17119bb9}
      - {6567fef5-0e15-11e0-a000-cb5e17119bb9}
      - {6efb52bf-1766-41db-a6b3-0ee5eff72bd7}
      - {7ea2e1ac-2e61-4728-aaa3-896d9d0a9f0e}
      - {7ff607e0-4395-11db-b0de-0800200c9a66}
      - {9dea862c-5cdd-4e70-acc1-f32b344d4795} **Windows Boot Manager**
        - Description
        - Elements
          - 11000001 **BcdLibraryDevice_ApplicationDevice**
          - 12000004 **BcdLibraryString_Description**
          - 12000005 **BcdLibraryString_PreferredLocale**
          - 14000006 **BcdLibraryObjectList_InheritedObjects**
          - 23000003 **BcdBootMgrObject_DefaultObject**
          - 23000006 **BcdBootMgrObject_ResumeObject**
          - 24000001 **BcdBootMgrObjectList_DisplayOrder**
          - 24000010 **BcdBootMgrObjectList_ToolsDisplayOrder**
          - 25000004 **BcdBootMgrInteger_Timeout**
      - {b2721d73-1db4-4c62-bf78-c548a880142d} **Windows Memory Tester**





BCD Elements determining KMCSP
(before KB2506014)

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

BCD Elements determining KMCSP (before KB2506014)

BCD e
BcdLibra… (0x1600…
BcdOSL… (0x2600…
BcdLibra… (0x1600…

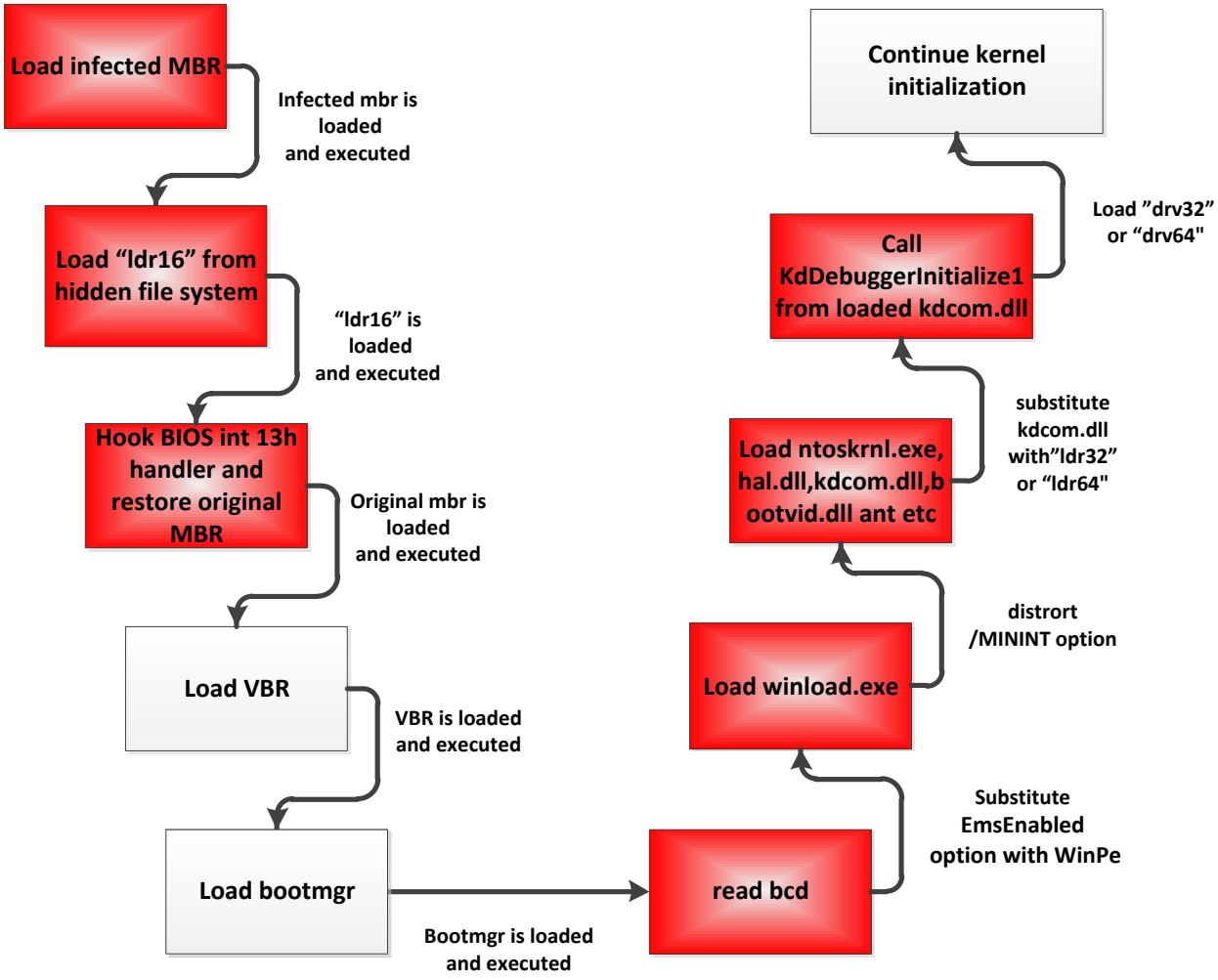
```
; BYTE __stdcall BlImgQueryCodeIntegrityBootOptions(BYTE *a2, BYTE *pOption, BYTE *AllowPreReleaseSign)
_BlImgQueryCodeIntegrityBootOptions@12 proc near
                mov     edi, edi
                push    ebp
                mov     ebp, esp
                push    ecx
                push    esi
                mov     esi, [edx+14h]
                lea     eax, [ebp+var_1]
                push    eax
                push    BcdLibraryBoolean_DisableIntegrityChecks
                push    esi
                call    _BlGetBootOptionBoolean@12 ; BlGetBootOptionBoolean(x,x,x)
                test    eax, eax
                jge     short loc_428742
                mov     [ebp+var_1], 0

loc_428742:                             ; CODE XREF: BlImgQueryCodeIntegrityBootOptions(x,x,x)+1B↑j
                test    byte ptr [edx], 4
                jz      short loc_428764
                cmp     [ebp+var_1], 0
                jnz     short loc_428764
                lea     eax, [ebp+var_1]
                push    eax
                push    BcdLibraryBoolean_WinPEEnabled
                push    esi
                call    _BlGetBootOptionBoolean@12 ; BlGetBootOptionBoolean(x,x,x)
                test    eax, eax
                jge     short loc_428764
                mov     [ebp+var_1], 0

loc_428764:                             ; CODE XREF: BlImgQueryCodeIntegrityBootOptions(x,x,x)+24↑j
                                        ; BlImgQueryCodeIntegrityBootOptions(x,x,x)+2A↑j ...
                mov     eax, [ebp+pOption]
                mov     cl, [ebp+var_1]
                mov     [eax], cl
                lea     eax, [ebp+var_1]
                push    eax
                push    BcdLibraryBoolean_AllowPrereleaseSignatures
                push    esi
                call    _BlGetBootOptionBoolean@12 ; BlGetBootOptionBoolean(x,x,x)
```

Abusing Win PE mode: TDL4 modules

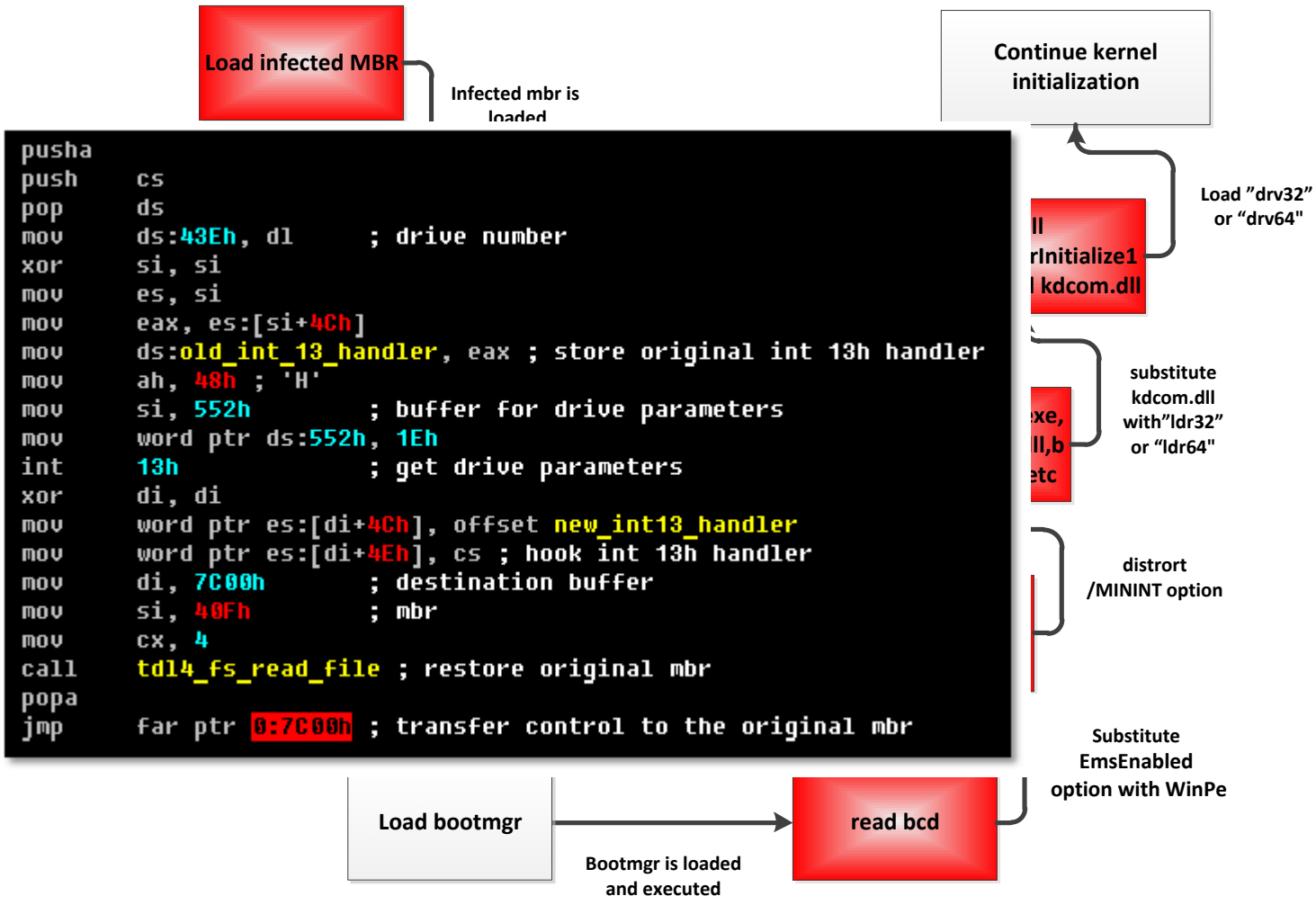
Module name	Description
mbr (infected)	infected MBR loads *ldr16* module and restores original MBR in memory
ldr16	hooks 13h interrupt to disable KMCSP and substitute *kdcom.dll* with *ldr32* or *ldr64*
ldr32	reads TDL4's kernel-mode driver from hidden file system and maps it into kernel-mode address space
ldr64	implementation of *ldr32* module functionality for 64-bit OS

int 13h – service provided by BIOS to communicate with IDE HDD controller

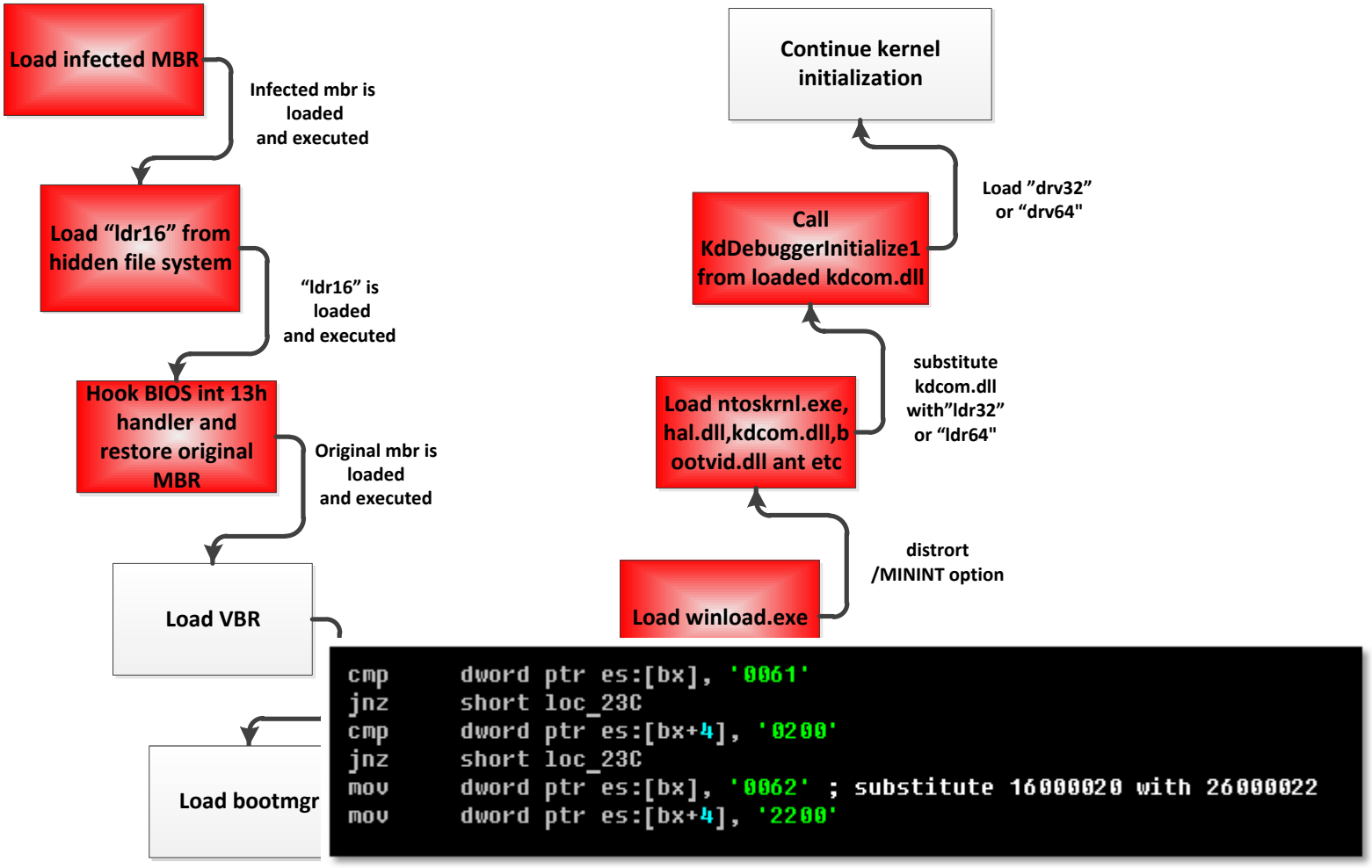
Abusing Win PE mode: Workflow



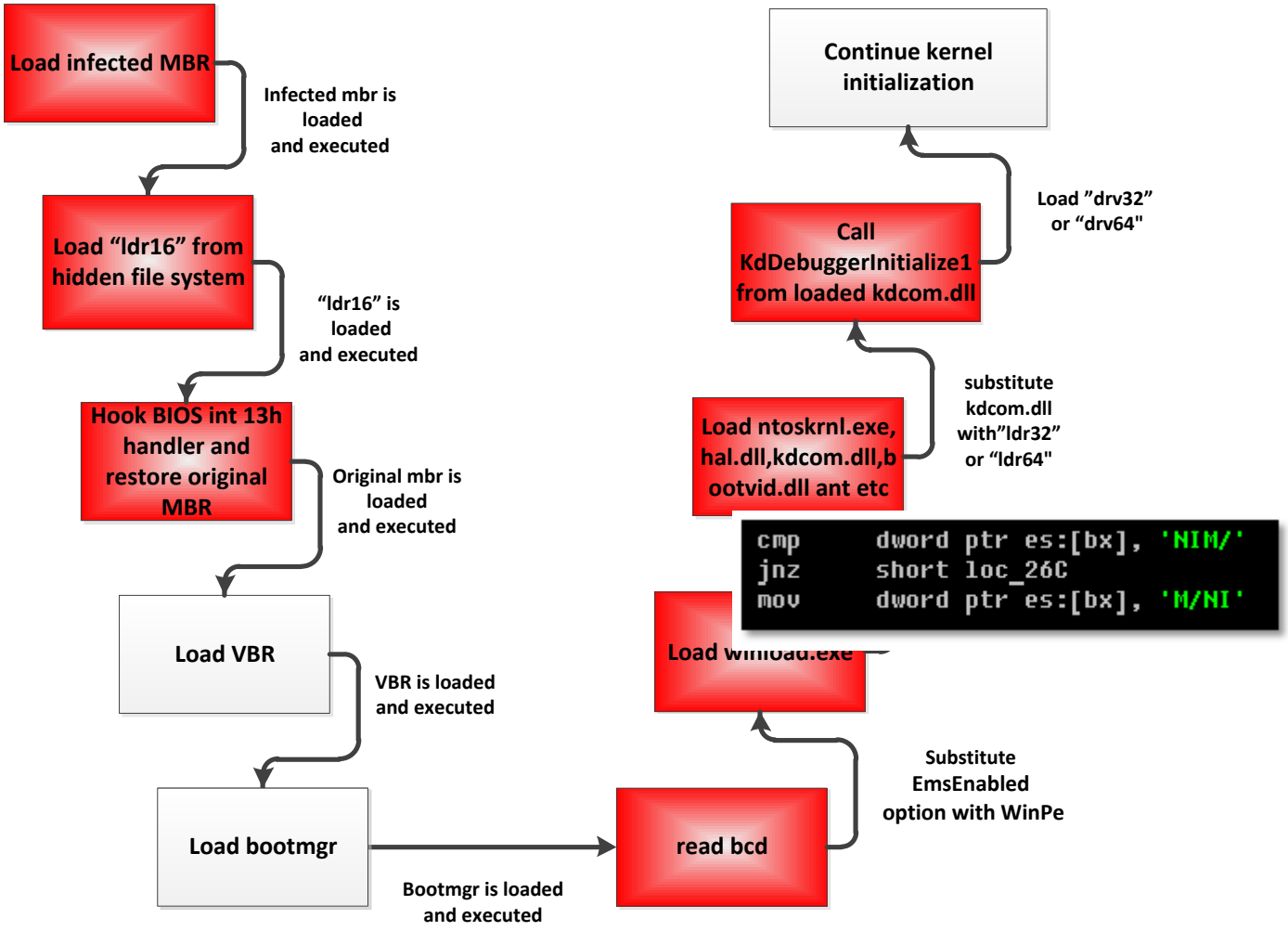
Abusing Win PE mode: Workflow



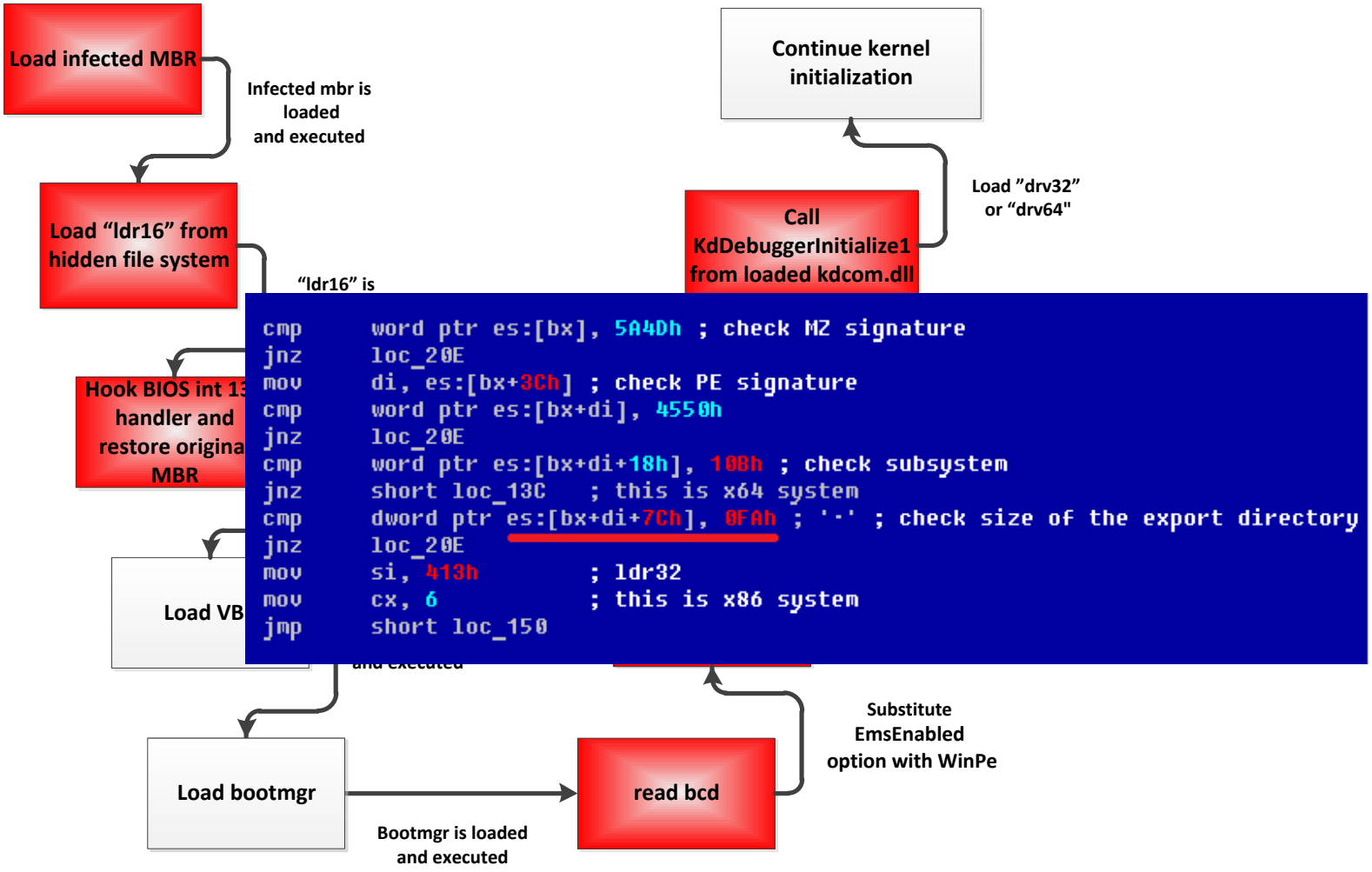
Abusing Win PE mode: Workflow



Abusing Win PE mode: Workflow



Abusing Win PE mode: Workflow

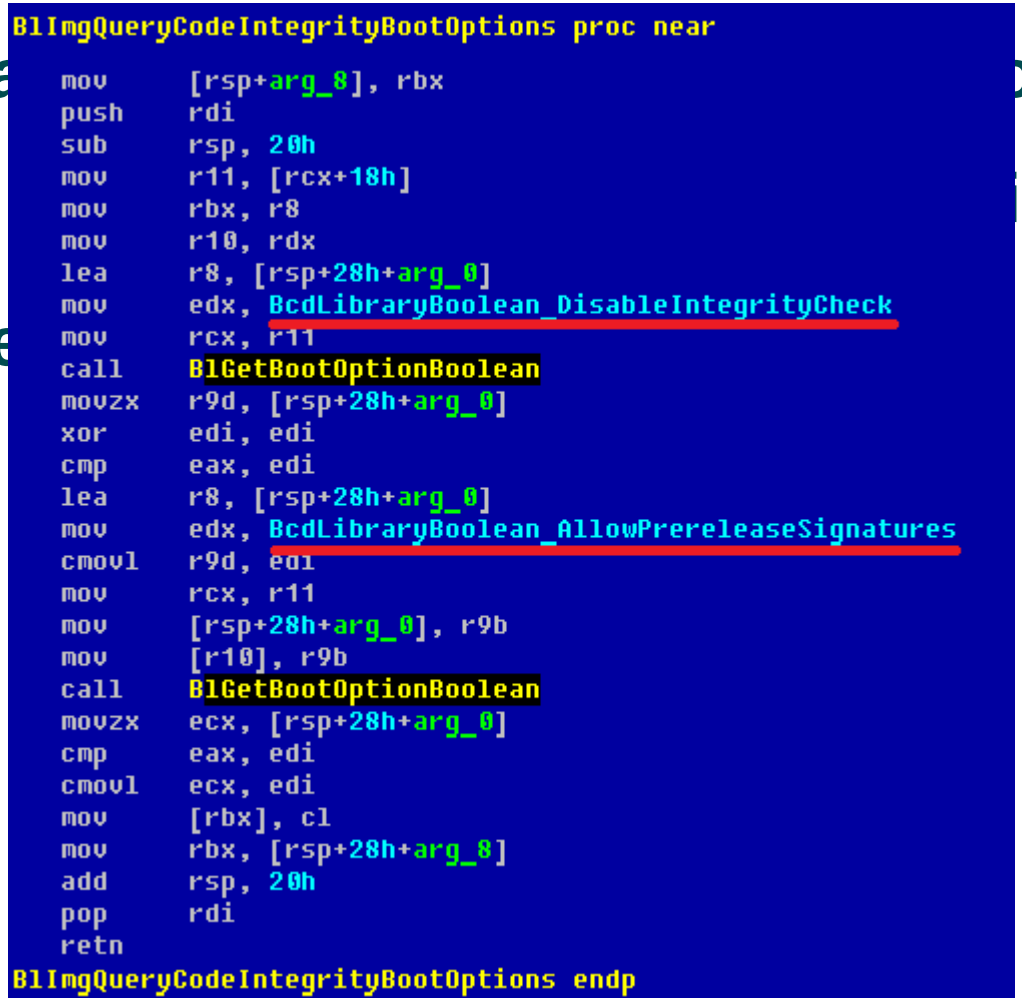


MS Patch (KB2506014)

- ***BcdOsLoaderBoolean_WinPEMode*** **option no longer influences kernel-mode code signing policy**
- **Size of the export directory of *kdcom.dll* has been changed**

MS Patch (KB2506014)

- ***BcdOsLoa*** **…on no longer influences … …icy**
- **Size of the … has been changed**



```
BlImgQueryCodeIntegrityBootOptions proc near
    mov     [rsp+arg_8], rbx
    push    rdi
    sub     rsp, 20h
    mov     r11, [rcx+18h]
    mov     rbx, r8
    mov     r10, rdx
    lea     r8, [rsp+28h+arg_0]
    mov     edx, BcdLibraryBoolean_DisableIntegrityCheck
    mov     rcx, r11
    call    BlGetBootOptionBoolean
    movzx   r9d, [rsp+28h+arg_0]
    xor     edi, edi
    cmp     eax, edi
    lea     r8, [rsp+28h+arg_0]
    mov     edx, BcdLibraryBoolean_AllowPrereleaseSignatures
    cmovl   r9d, edi
    mov     rcx, r11
    mov     [rsp+28h+arg_0], r9b
    mov     [r10], r9b
    call    BlGetBootOptionBoolean
    movzx   ecx, [rsp+28h+arg_0]
    cmp     eax, edi
    cmovl   ecx, edi
    mov     [rbx], cl
    mov     rbx, [rsp+28h+arg_8]
    add     rsp, 20h
    pop     rdi
    retn
BlImgQueryCodeIntegrityBootOptions endp
```

MS Patch (KB2506014)

- *BcdOsLoa* …on no longer influences … …icy
- Size of the … has been changed

```
BlImgQueryCodeIntegrityBootOptions proc near
    mov     [rsp+arg_8], rbx
    push    rdi
    sub     rsp, 20h
    mov     r11, [rcx+18h]
    mov     rbx, r8
    mov     r10, rdx
    lea     r8, [rsp+28h+arg_0]
    mov     edx, BcdLibraryBoolean_DisableIntegrityCheck
    mov     rcx, r11
    add     rsp, 20h
    pop     rdi
    retn
BlImgQueryCodeIntegrityBootOptions endp
```

Ordinal	Function RVA	Name Ordinal	Name RVA	Name
N/A	00001E3C	00001E7A	00001E60	00001EE6
(nFunctions)	Dword	Word	Dword	szAnsi
00000001	00001014	0000	0000608C	KdD0Transition
00000002	00001014	0001	0000609B	KdD3Transition
00000003	00001020	0002	000060AA	KdDebuggerInitialize0
00000004	00001104	0003	000060C0	KdDebuggerInitialize1
00000005	00001228	0004	000060D6	KdReceivePacket
00000006	00001008	0005	000060E6	KdReserved0
00000007	00001158	0006	000060F2	KdRestore
00000008	00001144	0007	000060FC	KdSave
00000009	00001608	0008	00006103	KdSendPacket

Bypassing KMCSP: Another Attempt

Patch Bootmgr and OS loader (*winload.exe*) to disable KMCSP:

```
                        ; CODE XREF: seg000:0330↑j
    cmp     byte ptr es:[bx], 0BFh ; '¬'
    jnz     short loc_354
    cmp     dword ptr es:[bx+1], 0C0000428h ; mov edi, C0000428h
    jnz     short loc_354
    mov     dword ptr es:[bx+1], 0C428h ;     mov edi, 0000C428h
```

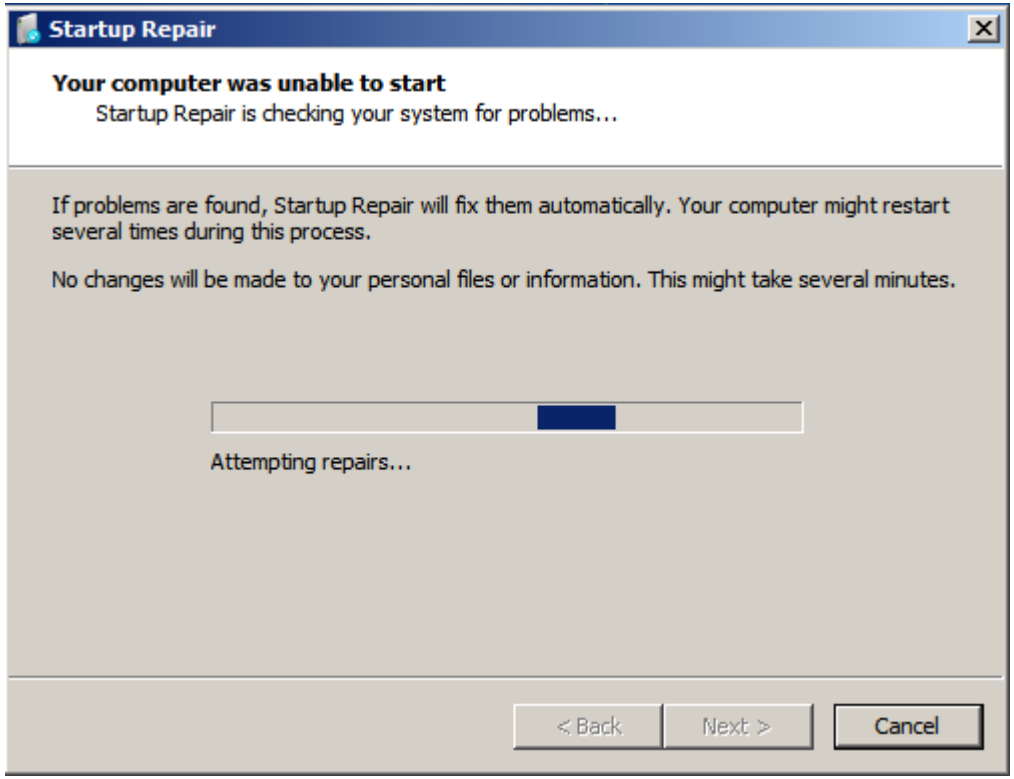
Bypassing KMCSP: Another Attempt

Patch Bootmgr and OS loader (*winload.exe*) to disable KMCSP:

```
                        ; __int32 __stdcall I_CheckImageHashInCatalog(struct _CRYPTOAPI_BLOB *, unsigned __int8 *const )
                        ?I_CheckImageHashInCatalog@@YAJPEAU_CRYPTOAPI_BLOB@@QEAE@Z proc near
                                                ; CODE XREF: MinCrypL_CheckImageHash+2C↑p
                                                ; MinCrypL_CheckImageHash+52↑p
                                                ; DATA XREF: ...

                        var_88          = dword ptr -88h
                        var_80          = qword ptr -80h
                        var_78          = byte ptr -78h
                        var_68          = dword ptr -68h
                        var_28          = dword ptr -28h
                        Source2         = qword ptr -20h
                        var_18          = byte ptr -18h
                        arg_0           = dword ptr  8
                        arg_8           = qword ptr  10h
                        arg_10          = qword ptr  18h

48 89 5C 24 10                          mov     [rsp+arg_8], rbx
48 89 6C 24 18                          mov     [rsp+arg_10], rbp
56                                      push    rsi
57                                      push    rdi
41 54                                   push    r12
48 81 EC 90 00 00 00                    sub     rsp, 90h
8B 19                                   mov     ebx, [rcx]
48 8B 69 08                             mov     rbp, [rcx+8]
4C 8B E2                                mov     r12, rdx
85 DB                                   test    ebx, ebx
BF 28 04 00 C0                          mov     edi, 0C0000428h ; STATUS_INVALID_IMAGE_HASH
```

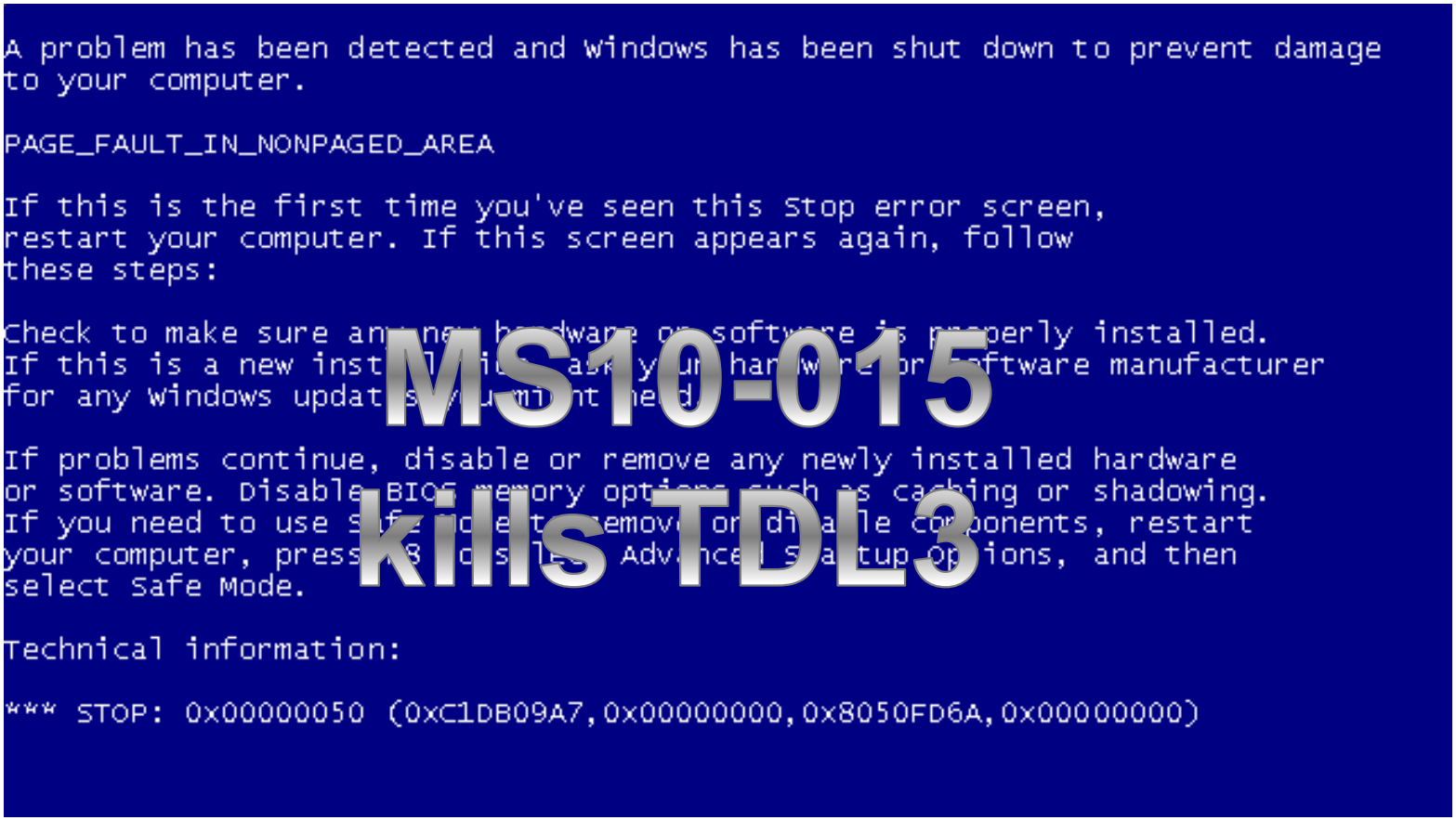
Bypassing KMCSP: Result

Bootmgr fails to verify OS loader's integrity



Bypassing KMCSP: Result

Bootmgr fails to verify OS loader's integrity

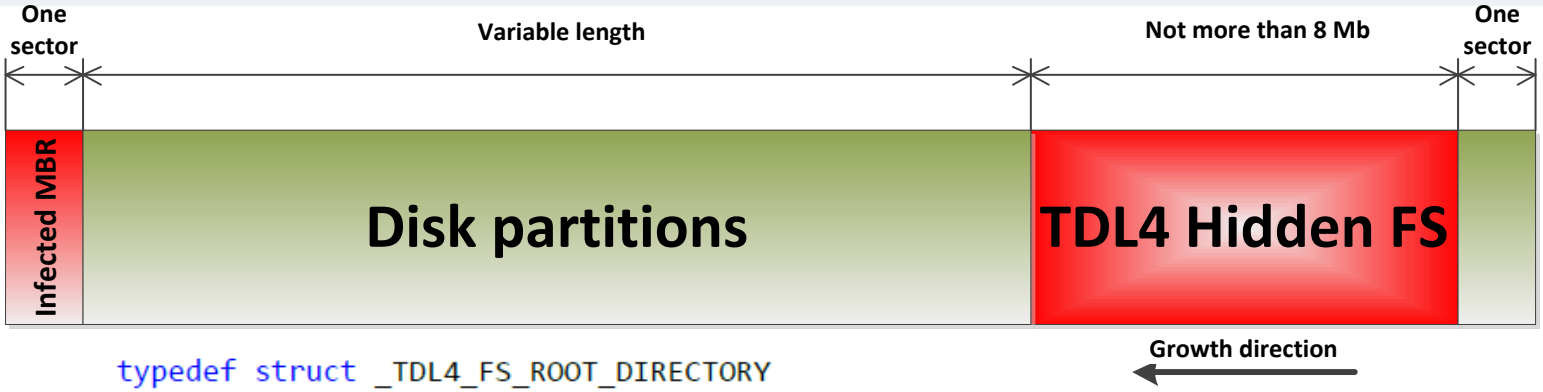


TDL4 Hidden File Systems

TDL’s Hidden Storage

- **Reserve space in the end of the hard drive (not visible at file system level analysis)**
- **Encrypted contents (stream cipher: RC4, XOR-ing)**
- **Implemented as a hidden volume in the system**
- **Can be accessed by standard APIs (*CreateFile, ReadFile, WriteFile, SetFilePointer, CloseHandle*)**

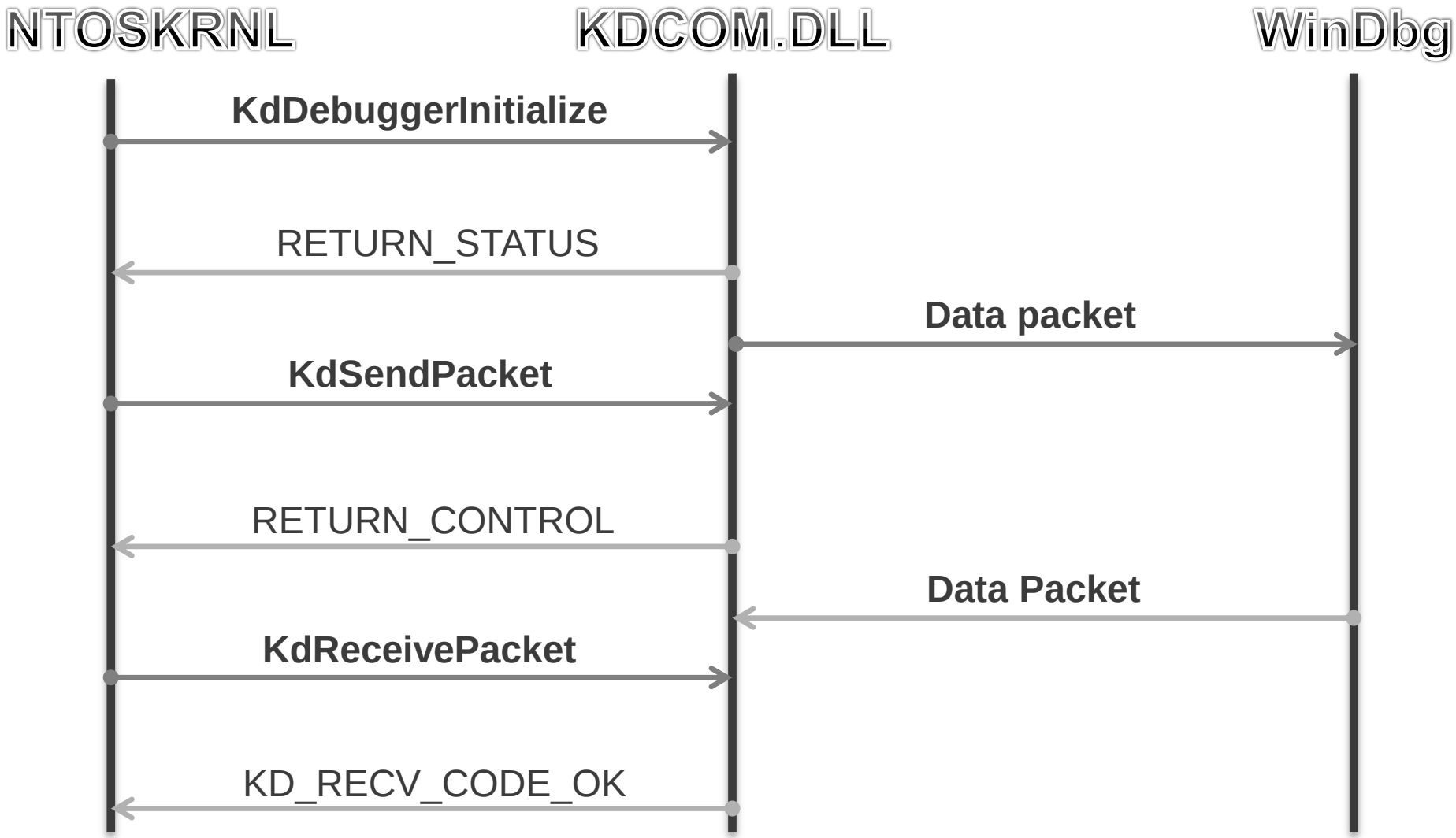
TDL4 File System Layout




```
typedef struct _TDL4_FS_ROOT_DIRECTORY
{
        // Signature of the block
        // DC - root directory
        WORD Signature;
        // Set to zero
        DWORD Reserved;
        // Array of entries corresponding to files in FS
        TDL4_FS_FILE_ENTRY FileTable[15];
}TDL4_FS_ROOT_DIRECTORY, *PTDL4_FS_ROOT_DIRECTORY;
typedef struct _TDL4_FS_FILE_ENTRY
{
        // File name - null terminated string
        char FileName[16];
        // Offset from beginning of the file system to file
        DWORD FileBlockOffset;
        // Reserved
        DWORD dwFileSize;
        // Time and Date of file creation
        FILETIME CreateTime;
}TDL4_FS_FILE_ENTRY, *PTDL4_FS_FILE_ENTRY;
```

Debugging Bootkit with WinDbg

WinDbg and kdcom.dll



TDL4 and kdcom.dll

original routine

```
; __stdcall KdDebuggerInitialize1(x)
                public _KdDebuggerInitialize1@4
_KdDebuggerInitialize1@4 proc near      ; DATA XREF: .edata:off_80011328↓o
                call    _KdCompInitialize1@0 ; KdCompInitialize1()
                xor     eax, eax
                retn    4
_KdDebuggerInitialize1@4 endp
```

modified routine

```
                public KdDebuggerInitialize1
KdDebuggerInitialize1 proc near         ; DATA XREF: .text:off_10001058↑o
                push    offset NotifyRoutine ; NotifyRoutine
                call    PsSetCreateThreadNotifyRoutine
                retn    4
KdDebuggerInitialize1 endp
```

```
; void __stdcall NotifyRoutine(HANDLE, HANDLE, BOOLEAN)
NotifyRoutine   proc near               ; DATA XREF: CallbackRoutine+1CE↑o
                                        ; KdDebuggerInitialize1↓o
                cmp     dword_100017F0, 0
                jnz     short locret_1000179D
                push    offset DriverEntry
                push    0
                call    IoCreateDriver
                xor     ecx, ecx
                test    eax, eax
                setns   cl
                mov     dword_100017F0, ecx

locret_1000179D:                        ; CODE XREF: NotifyRoutine+7↑j
                retn    0Ch
NotifyRoutine   endp
```

TDL4 and kdcom.dll

original export table

Name	Address	Ordinal
KdD0Transition	80010386	1
KdD3Transition	80010386	2
KdDebuggerInitialize0	800103A6	3
KdDebuggerInitialize1	8001044C	4
KdReceivePacket	80010F4C	5
KdRestore	80010460	6
KdSave	80010456	7
KdSendPacket	800111B2	8
HalInitSystem(x,x)	80010CE6	

modified export table

Name	Address	Ordinal
KdD0Transition	1000171A	1
KdD3Transition	10001724	2
KdDebuggerInitialize0	100017A0	3
KdDebuggerInitialize1	100017AC	4
KdReceivePacket	100017DC	5
KdRestore	100017C6	6
KdSave	100017BA	7
KdSendPacket	100017D2	8
DriverEntry	1000172E	

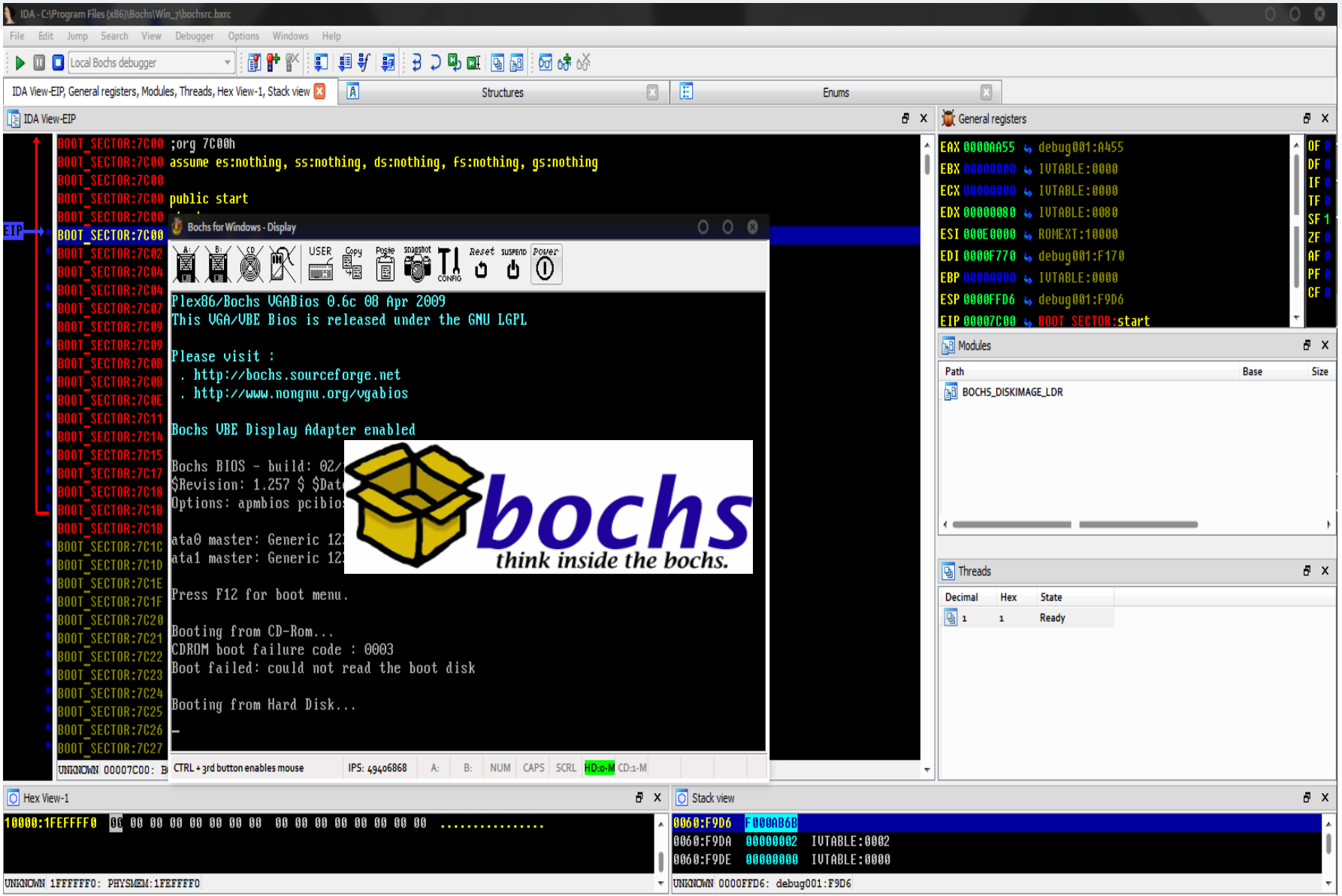
```
; void __stdcall NotifyRoutine(HANDLE, HANDLE, BOOLEAN)
NotifyRoutine   proc near               ; DATA XREF: CallbackRoutine+1CE↑o
                                        ; KdDebuggerInitialize1↓o
                cmp     dword_100017F0, 0
                jnz     short locret_1000179D
                push    offset DriverEntry
                push    0
                call    IoCreateDriver
                xor     ecx, ecx
                test    eax, eax
                setns   cl
                mov     dword_100017F0, ecx

locret_1000179D:                        ; CODE XREF: NotifyRoutine+7↑j
                retn    0Ch
NotifyRoutine   endp
```

How to Debug TDL4 with WinDbg

- **Patch *ldr16* to disable *kdcom.dll* substitution**
- **Reboot the system and attach to it with WinDbg**
- **Manually load *drv32/drv64***

“TDL4 Analysis Paper: a brief introduction and How to Debug It”, Andrea Allievi
http://www.aall86.altervista.org/TDLRootkit/TDL4_Analysis_Paper.pdf

Debugging Bootkits with Bochs

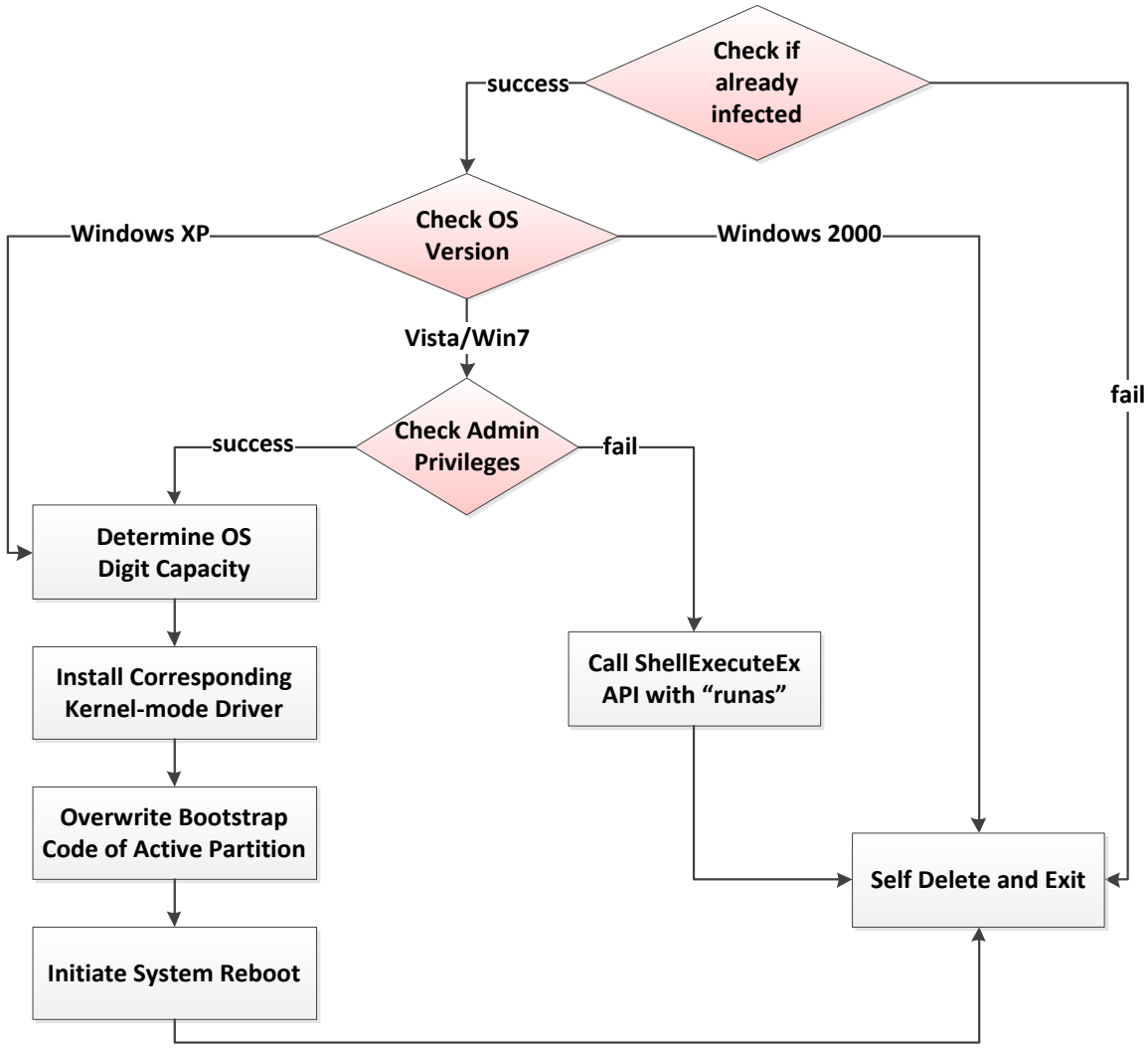


DEMO

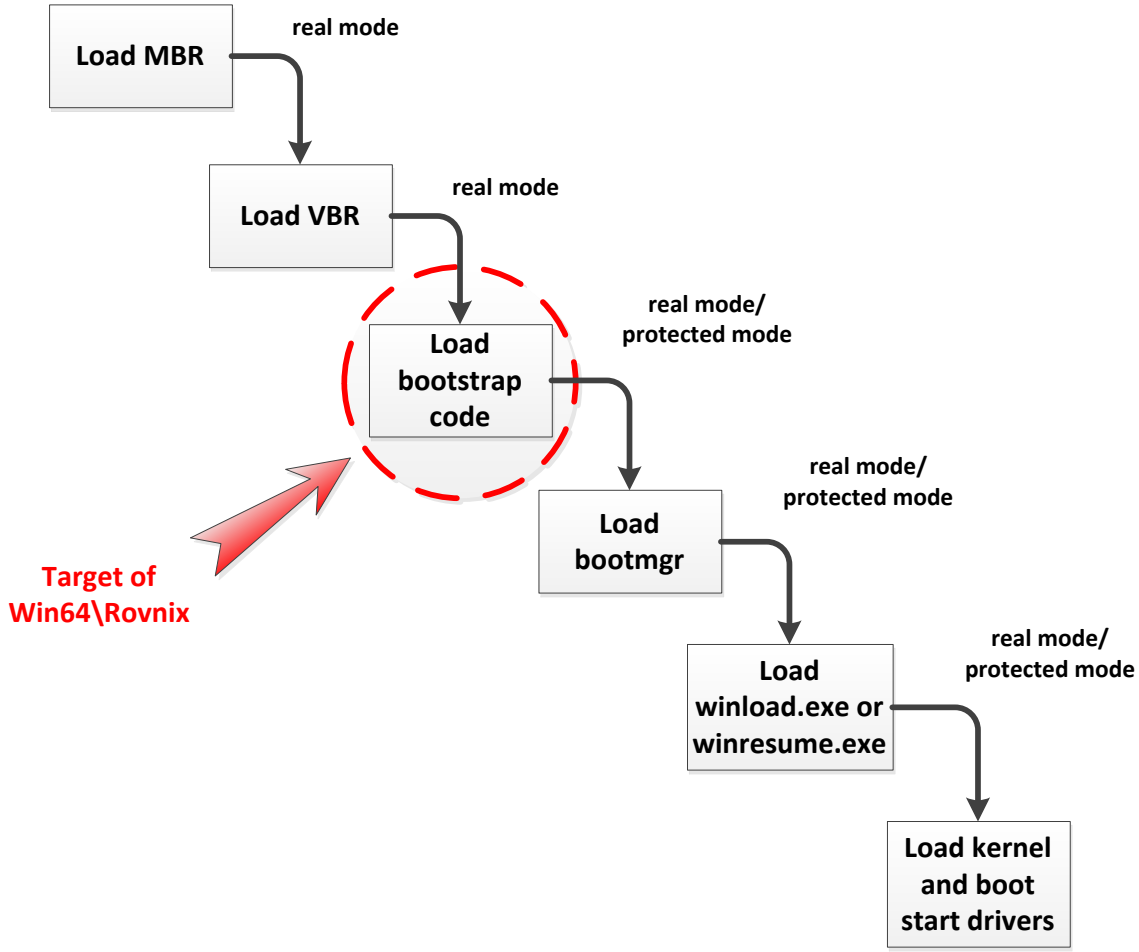
http://www.youtube.com/watch?v=sT6N7Dr-G6s

Win64/Rovnix

Win64/Rovnix: Installation

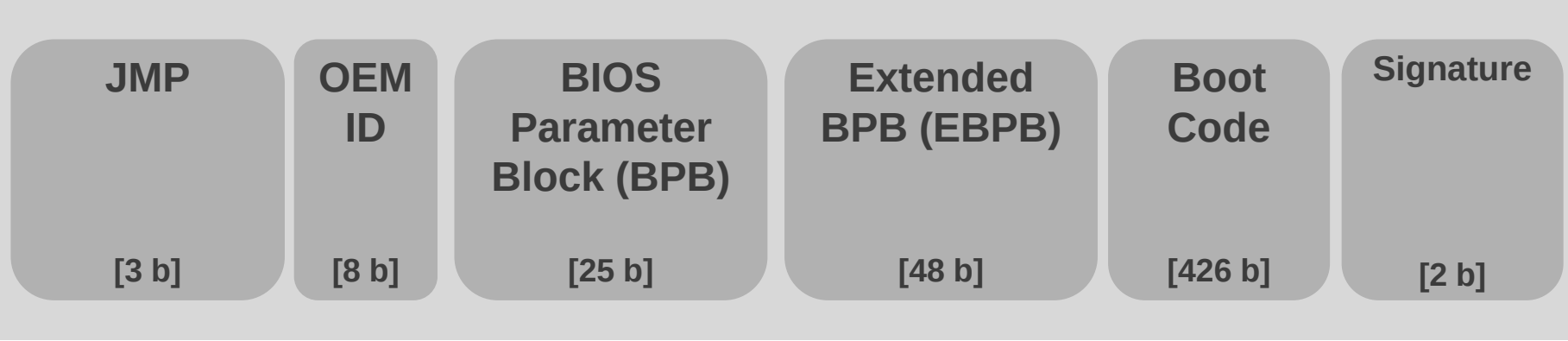


Win64/Rovnix: Bootkit Overview



NTFS Bootstrap Code

NTFS Boot Sector (Volume Boot Record)

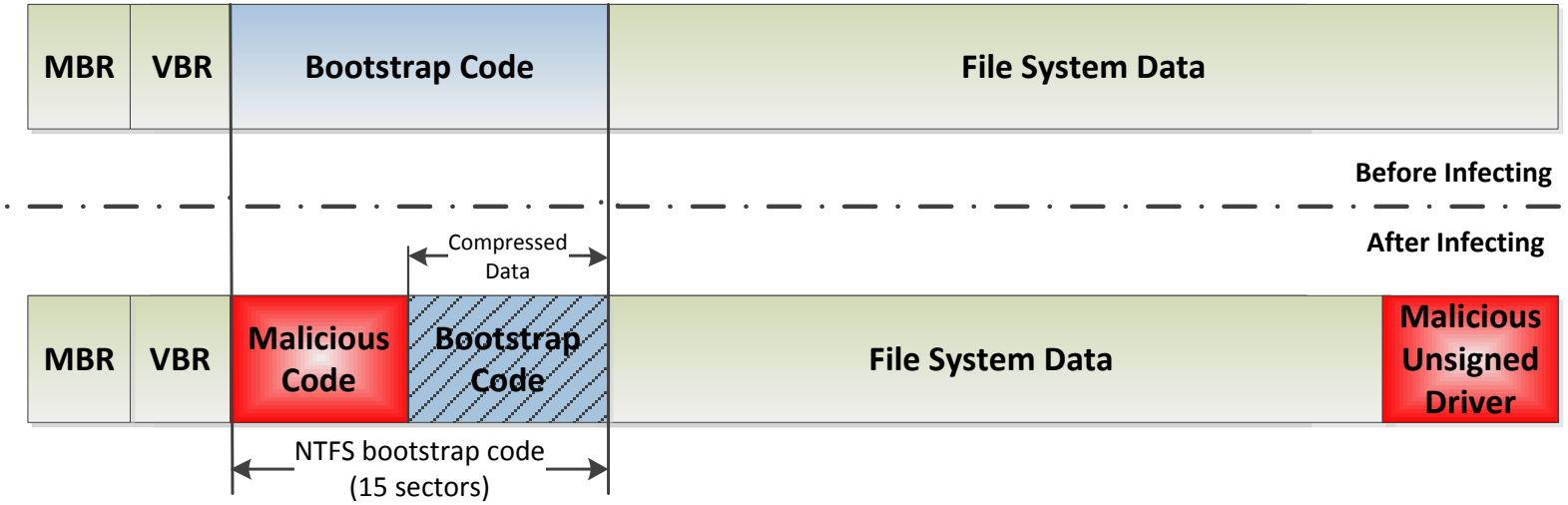


NTFS Bootstrap Code

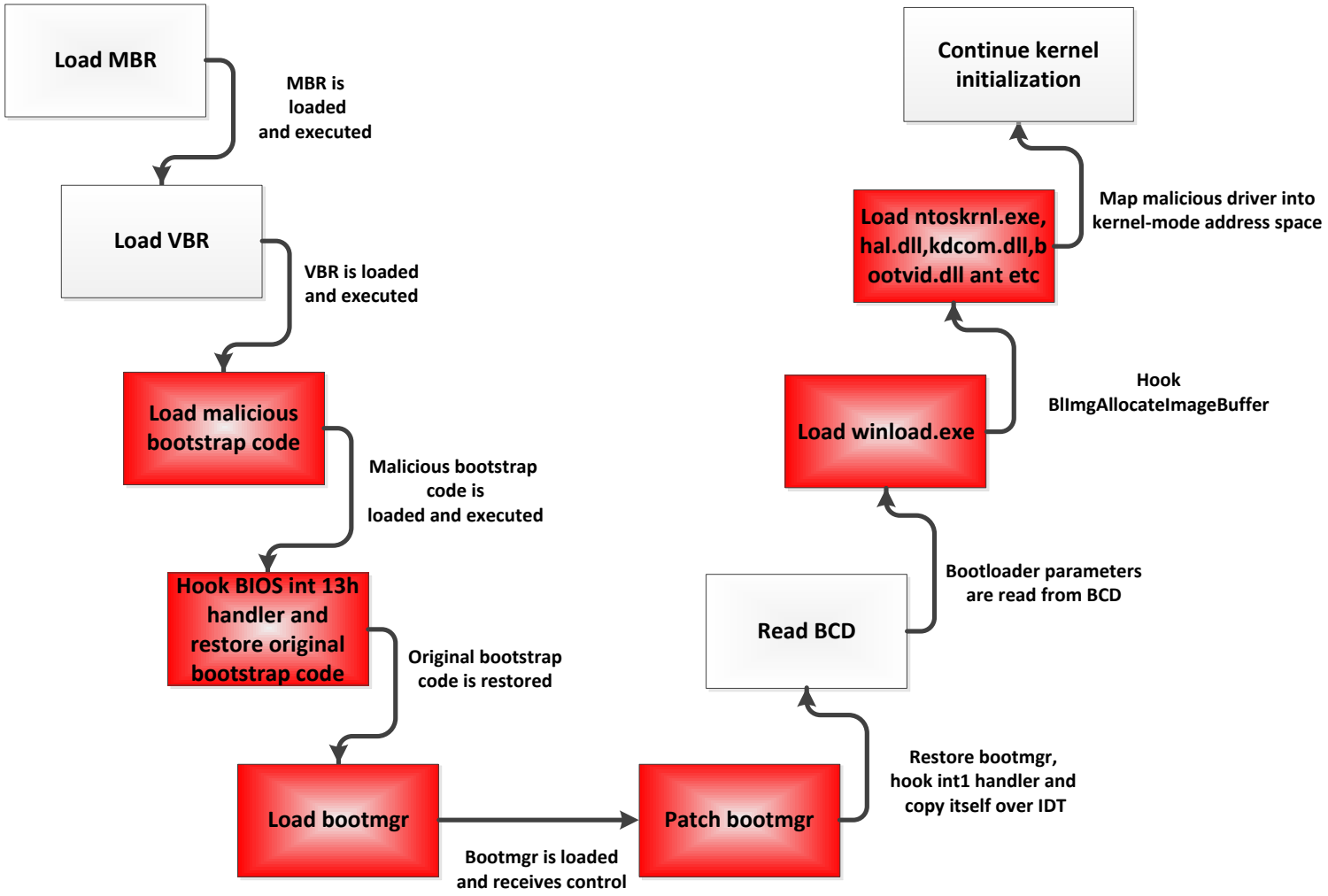

```
typedef struct BOOTSTRAP_CODE {
    BYTE code_1[8]       <fgcolor=cGreen>;
    BYTE IPL[32]         <fgcolor=cAqua>;
    BYTE IPL_EXT[7]      <fgcolor=cYellow>;
    BYTE code_2[387]     <fgcolor=cGreen>;
    BYTE endOfSector[2] <fgcolor=cWhite>;
};

typedef struct _BOOTSECTOR {
    BYTE jmp[3]                          <fgcolor=cGreen>;
    BYTE oemID[8]                        <fgcolor=cWhite>;
    BIOS_PARAMETR_BLOCK bpb              <fgcolor=cYellow>;
    EXT_BIOS_PARAMETR_BLOCK ebpb         <fgcolor=cPurple>;
    BOOTSTRAP_CODE bootstrap;
    NTLDR_CODE ntldr;
};
```

Win64/Rovnix: Infected Partition Layout

- Win64/Rovnix overwrites bootstrap code of the active partition
- The malicious driver is written either:
  - ✓ before active partition, in case there is enough space
  - ✓ in the end of the hard drive, otherwise

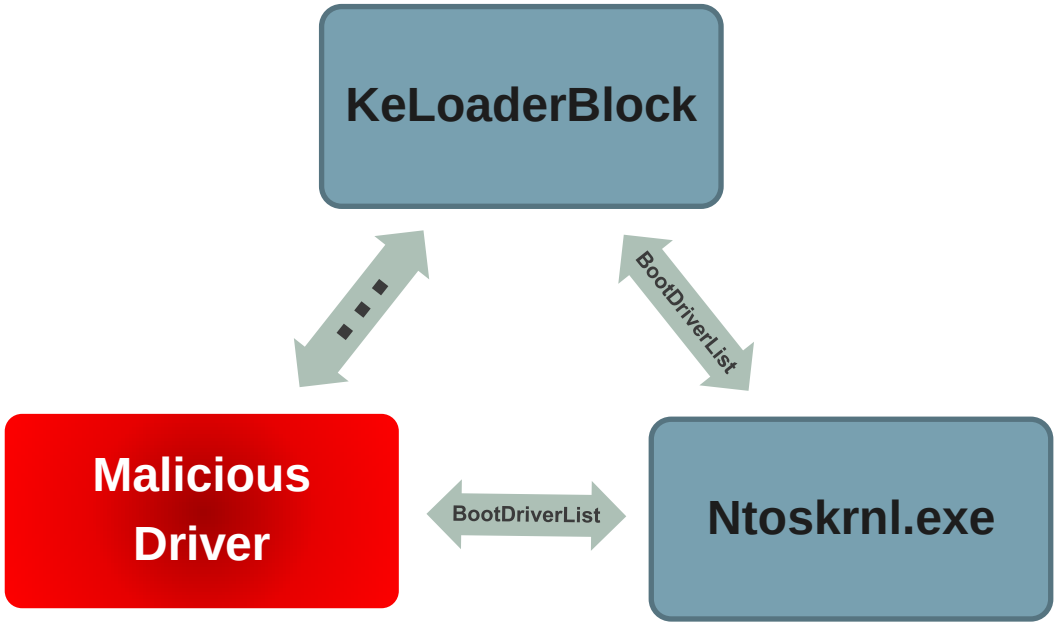


Win64/Rovnix: Bootkit Details



Win64/Rovnix: Loading Unsigned Driver

- Insert malicious driver in *BootDriverList* of *KeLoaderBlock* structure
- When kernel receives control it calls entry point of each module in the *BootDriverList*



Win64/Rovnix: Abusing Debugging Facilities

Win64/Rovnix:

- hooks Int 1h
  - ✓ tracing
  - ✓ handles hardware breakpoints (DR0-DR7)
- overwrites the last half of IDT (*Interrupt Descriptor Table*)
  - ✓ is not used by OS

As a result the malware is able to:

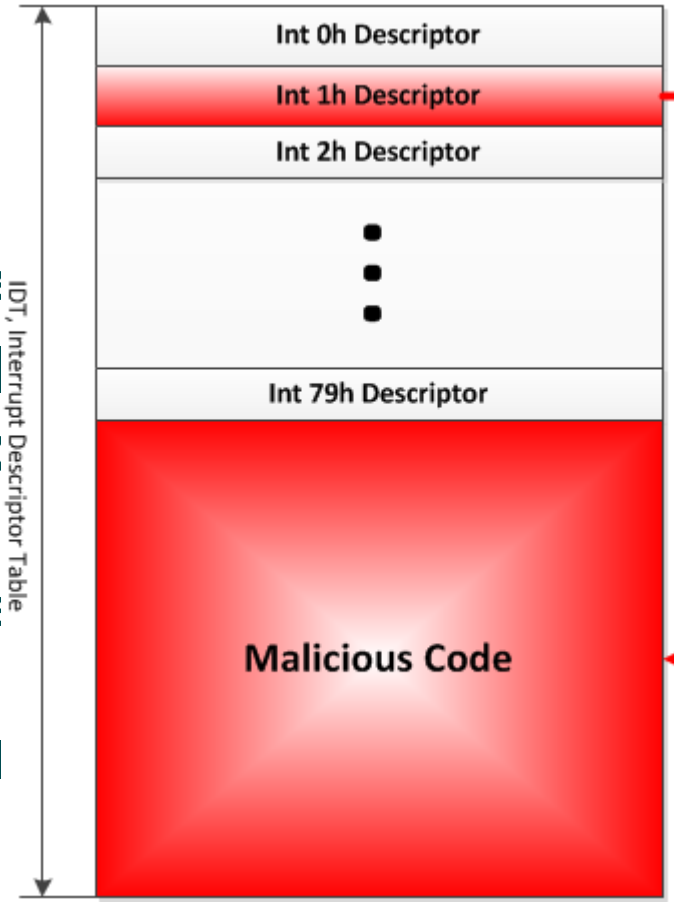
- ✓ set up hooks without patching bootloader components
- ✓ retain control after switching into protected mode

Win64/Rovnix: Abusing Debugging Facilities

Win64/Rovnix:

- hooks Int 1h
  - ✓ tracing
  - ✓ handles hardwa
- overwrites the I *Descriptor Table*)
  - ✓ is not used by (

As a result the ma
- ✓ set up hooks :omponents
- ✓ retain control mode



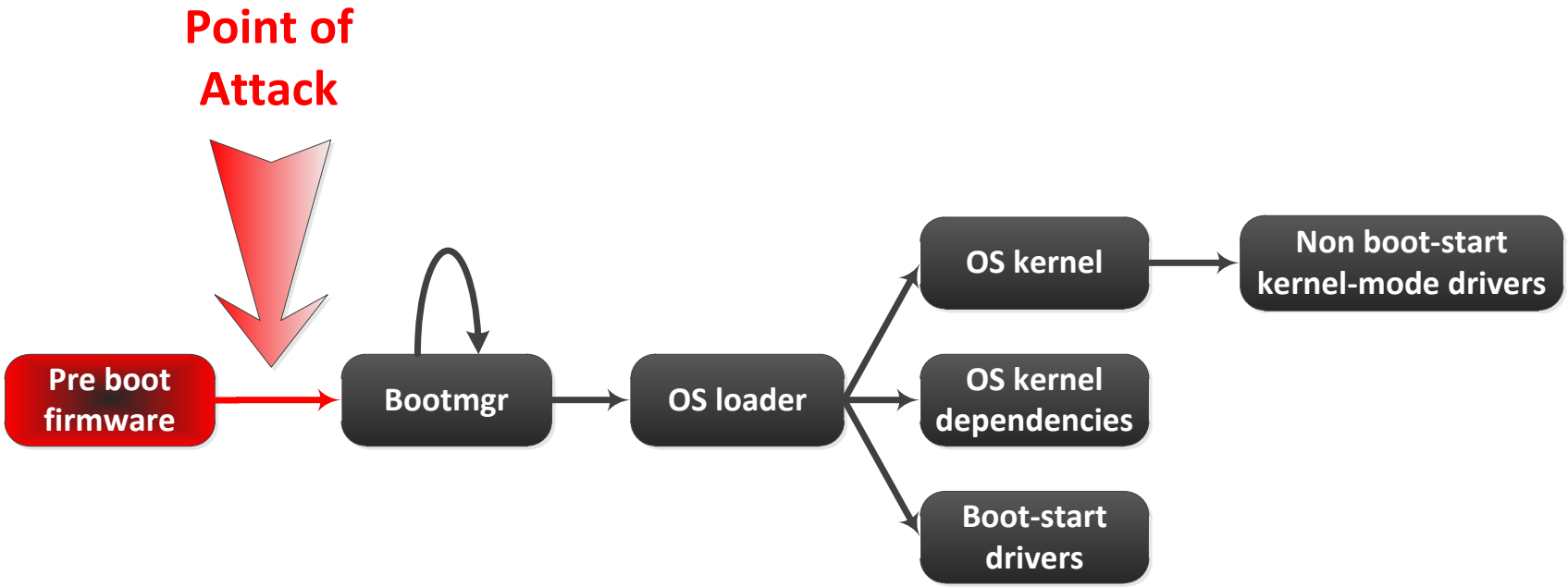
DEMO

Olmarik vs Rovnix

Characteristics	Win64/Olmarik	Win64/Rovnix
Privilege escalation	MS10-092	☒
Reboot technique	*ZwRaiseHardError* API	*ExitWindowsEx* API
MBR/VBR infection	MBR	VBR (bootstrap code)
Loading driver	*ZwCreateDriver* API	Inserting into boot driver list of *KeLoaderBlock* structure
Payload injection	*KeInitializeApc/ KeInstertQueueApc* APIs	*KeInitializeApc/ KeInstertQueueApc* APIs
Self-defense	Kernel-mode hooks, MBR monitoring	☒
Number of modules	10	2
Stability of code	■■■■□	■■■□□
Threat complexity	■■■□□	■■■■□

What Facilitates the Attack Vector?

- **Untrusted platform problem**
  - ✓ **BIOS controls boot process, but who controls it?**
  - ✓ **The trust anchor is below point of attack**



HiddenFsReader as a Forensic Tool

HiddenFsReader as a Forensic Tool

Retrieves content of the malware hidden file system.

Supported malware: TDL3/TDL3+,TDL4;

ZeroAccess (will be added soon)

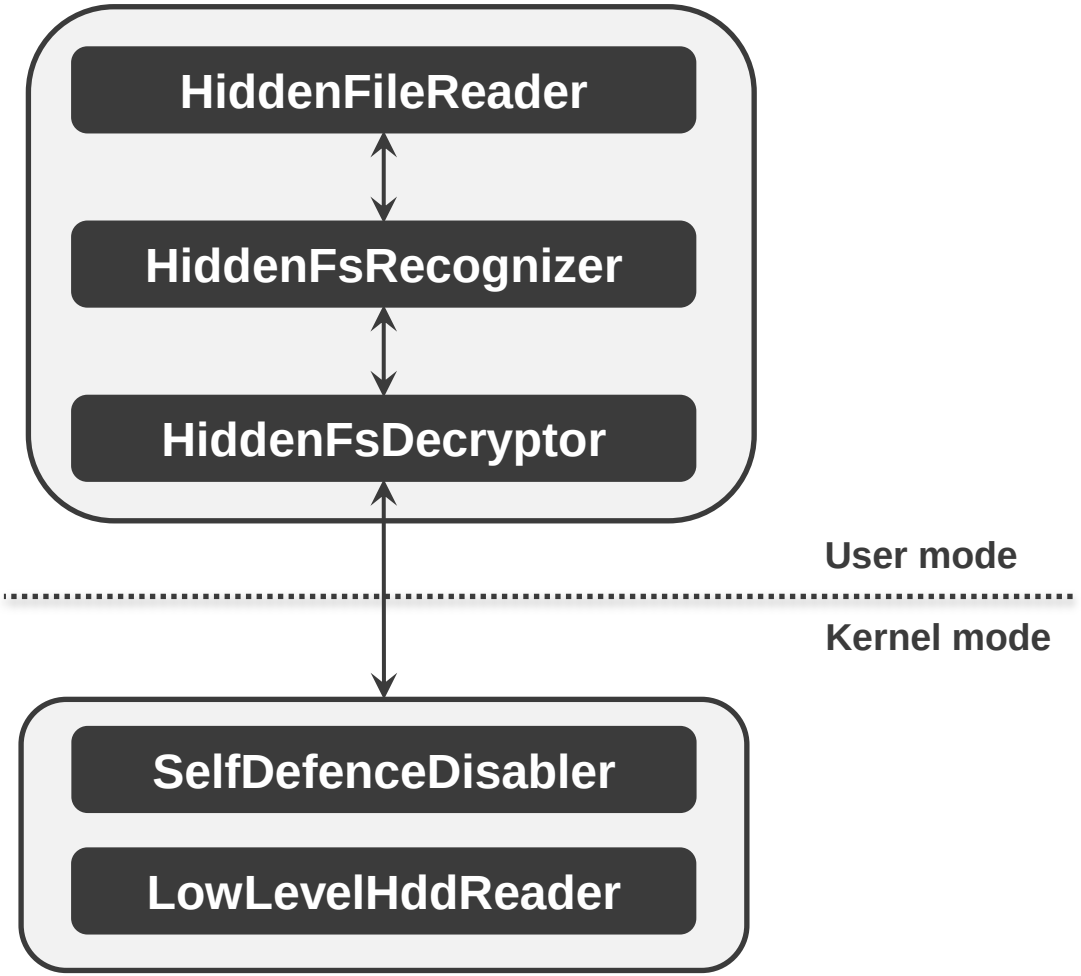
```
C:\>TdlFsReader.exe
Contents of TDL file system:
         cfg.ini MD5: B8C8B1B5C01EBF2F48760F2E06C402E6
             mbr MD5: AF1EC9B9C5CE1D74D3D9CA3BBE0FA941
       bckfg.tmp MD5: 6AD76461EEB59A1D77529B595D3672ED
         cmd.dll MD5: 4DADED6C7EFF7230D68AE48B02A847FA
           ldr16 MD5: 4FB9748189F6688ADA9A51A5901406FA
           ldr32 MD5: C078E5EA19F853AC0830D2F6088F7161
           ldr64 MD5: ADEB890D564913F11301C648A5FB6220
           drv64 MD5: 87A462D034192EDD60ACE835E91B930B
       cmd64.dll MD5: BC3B9FB8EAFD440D43B76DC12FB445C7
           drv32 MD5: 1EF0E0C265DA7F727E1FB8FF38D02FF1
C:\>_
```

http://eset.ru/tools/TdlFsReader.exe

DEMO

http://www.youtube.com/watch?v=iRpp6vn2DAE

HiddenFsReader (HFR) Architecture



Conclusion

- ✓ The bootkit technique allows malware to bypass KMCSP
- ✓ Return to old-school techniques of infecting MBR
- ✓ Win64/Olmarik (TDL4) is the first widely spread rootkit targeting Win x64
- ✓ Win64/Rovnix relies on debugging facilities of the platform to subvert KMCSP
- ✓ The only possible way of debugging bootkits is to use emulators (Bochs, QEMU)
- ✓ The untrusted platform facilitates bootkit techniques
- ✓ HiddenFsReader is shared amongst malware researchers

References

- ✓ **“The Evolution of TDL: Conquering x64”**

  http://www.eset.com/us/resources/white-papers/The_Evolution_of_TDL.pdf

- ✓ **“Defeating x64: The Evolution of the TDL Rootkit”**

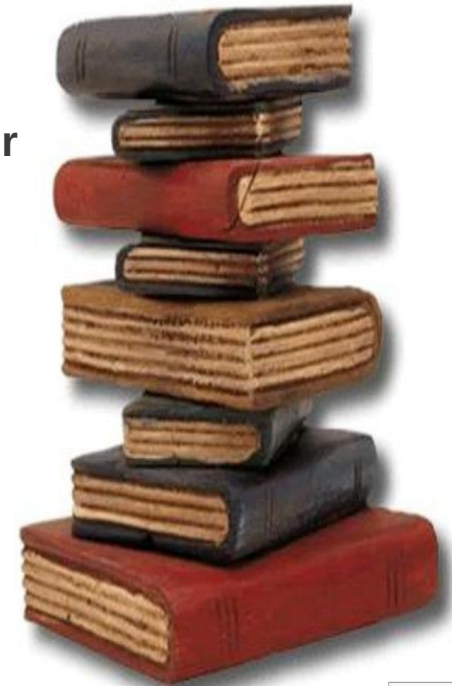
  http://www.eset.com/us/resources/white-papers/TDL4-CONFidence-2011.pdf

- ✓ **“Hasta La Vista, Bootkit: Exploiting the VBR”**

  http://blog.eset.com/2011/08/23/hasta-la-vista-bootkit-exploiting-the-vbr

- ✓ **Follow ESET Threat Blog**

  http://blog.eset.com



Questions

Thank you for your attention ;)

Aleksandr Matrosov
matrosov@eset.sk
@matrosov

Eugene Rodionov
rodionov@eset.sk
@vxradius