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
History of BIOS rootkits

In the Wild

1998
WinCIH

2006
ACPI Rootkit

2007
POI Option Rootkit

2008
IceLord Rootkit

2009
SMM Rootkit

2011
BIOS Patching

2012
Computrace

2013
Mebromi/BIOSkit

2014
Rakshasa

2015
Dream Boot

2016
1st SecureBoot Bypass

Proof of Concept

Move to UEFI world with Secure Boot

MS Win10: Virtualization Based Security Era
History of BIOS rootkits

Low Threat Activity

In the Wild

Proof of Concept

Low Research Activity

Move to UEFI world with Secure Boot

MS Win10: Virtualization Based Security Era
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)
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

Signed BIOS Updates Are Rare

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

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

SEC

S-CRTM; Init caches/MTRRs; Cache-as-RAM (NEM); Recovery; TPM Init

Pre-EFI Init (PEI)

S-CRTM: Measure DXE/BDS
Early CPU/PCH Init
Memory (DIMMs, DRAM) Init, SMM Init

Driver Exec Env (DXE)

Continue initialization of platform & devices
Enum FV, dispatch drivers (network, I/O, service..)
Produce Boot and Runtime Services

Boot Dev Select (BDS)

Boot Manager (Select Boot Device)
EFI Shell/Apps; OS Boot Loader(s)

ExitBootServices. Minimal UEFI services (Variable)

Runtime / OS

ACPI, UEFI SystemTable, SMBIOS table
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

EFI_SYSTEM_TABLE

Pointers

- EFI_RUNTIME_SERVICES
- EFI_BOOT_SERVICES
- EFI_DXE_SERVICES

Module: hal.dll

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>HalpEfiRuntimeActive</td>
<td>FFFF800476329E0</td>
</tr>
<tr>
<td>HalpEfiRuntimeServicesBlock</td>
<td>FFFF800476690C0</td>
</tr>
<tr>
<td>HalpEfiBugcheckCallbackNextRuntimeServiceIndex</td>
<td>FFFF8004766910B</td>
</tr>
<tr>
<td>HalpEfiRuntimeServicesTable</td>
<td>FFFF80047669118</td>
</tr>
<tr>
<td>HalpEfiRuntimeCallbackRecord</td>
<td>FFFF8004766BC58</td>
</tr>
</tbody>
</table>
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

OS Kernel-Mode

Exploit

MODIFY

U/EFI System Firmware

S3 Boot Script Table
Restores hardware config

Script Engine

NORMAL BOOT

SPI Flash

Exploit tricks SMI handler to write to an address \textit{inside SMRAM}

\textbf{Attacking and Defending BIOS in 2015}
Exploiting firmware SMI handler

OS Kernel-Mode

Exploit invoke SMI handlers (grants access to SW SMI I/O port 0xB2)

SMI Handlers
System Firmware

Injects SMM payload through the input pointer vulnerability in SMI handler

SMM firmware payload install a persistent rootkit

Hardware
Memory CPU Graphics
I/O Network

Exploit
What about Secure Boot?
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

---

Madness, as you know, is a lot like gravity, all it takes is a little push.
Going deeper or bypass still possible?

Windows 10 Secure Boot
UEFI Secure Boot
Measured Boot

Signed BIOS Update

DXE Driver
DXE Driver

OS Driver
OS Exploit
OS Kernel
UEFI OS Loaders

Modify Secure Boot FW or config in ROM

UEFI Boot Loader
Bootx64.efi
Bootmgfw.efi

UEFI DXE Core / Dispatcher
System Firmware (SEC/PEI)
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

- **rkloader**
  - DXE module
  - bootkit trigger

- **fsbg**
  - UEFI application
  - main bootkit functionality

- **ntfs**
  - DXE module
  - NTFS driver
Hacking Team UEFI Implant: How It Works

- **RkLoader** is executed at DXE phase by Firmware

- Load and execute main bootkit module *fsbg*

- Application *fsbg* is executed

  - Initialize NTFS protocol by loading NTFS driver

- Drop malware onto NTFS volume

  - Application *fsbg* installs malware onto NTFS volume
Hacking Team UEFI Implant: How It Works

```c
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 =
{ 0x3aa9ace,
  0xc9c1,
  0x46e2,
};

NewFilePathProtocol = (EFI\_DEVICE\_PATH\_PROTOCOL *) (((UINT8 *) NewDevicePathProtocol + DevicePathLength) + 4);
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 EFI\_API
\_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;
}
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 CAL_C_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"kdfas7835jfwte09j29FKFLDOR3r35fJR";
UINT16 g_NAME_ELITE[] = L"eorpkek3904kLDKQ0023iosdn93smMXK";
#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
DEITYBOUNCE
Only Snowden-leaked documentation is available for analysis

Safe to assume that servers use legacy BIOS

DEITYBOUNCE Workflow

ARKSTREAM → DEITYBOUNCE → SMRAM → Malicious SMI handler

System Initialization

Execute OS bootloader

Patch OS kernel/Inject malicious module

OS Kernel is initialized
BANANABALLOT and JETPLOW (Equation Group)
(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.

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

(TS/CI/REL) 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: (CI/REL) Released. Has been widely deployed. Current availability restricted based on OS version (request for details).

POC: S32222, nsa.ic.gov

Unit Cost: $0

Derived From: NSA/CSSR 1-52
Date Issued: 06/24/08
Declassify On: 20321036
File: BBALL_AM29F4-2131.exe
Name: biosModule_AM29F4
Version: 0x02010301
Priority: 10
ID: 65793
chain: 0x10000000
Command: handler_readBIOS
Command: handler_writeBIOS
MNUCE
FINAL
.Interface
.menu
.menuItem
  .itemText Read BIOS AM29F4 Memory
  .querylist
    .query Enter Bios Address:
    .query Enter number of bytes to read:
  .querylist
  .progmName LM_readBIOS
  .handler handler_readBIOS
  .argList
    .arg --biosaddr
    .arg --bioso
  .argList
</menuItem>
</menuItem>
.menuItem
  .itemText Write a file to BIOS AM29F4 memory
  .querylist
    .query Address to write data:
    .query Enter filename of binary data to write:
  .querylist
  .progmName LM_writeBIOS
  .handler handler_writeBIOS
  .argList
    .arg --biosAddr
    .arg --writeFile
  .argList
</menuItem>
</menu>

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>writeBios.asaBios</td>
<td>00001350</td>
</tr>
<tr>
<td>chipRead.asaBios</td>
<td>000017C0</td>
</tr>
<tr>
<td>kmmodData</td>
<td>000021C0</td>
</tr>
<tr>
<td>reverse6</td>
<td>00001EC0</td>
</tr>
<tr>
<td>reverse4</td>
<td>00001E90</td>
</tr>
<tr>
<td>sizeof_kmmodData</td>
<td>000021A0</td>
</tr>
<tr>
<td>comparePixOSVersion</td>
<td>00001F70</td>
</tr>
<tr>
<td>checksum_uint32</td>
<td>00001D20</td>
</tr>
<tr>
<td>cmosReadByte</td>
<td>00001B50</td>
</tr>
<tr>
<td>writeBios</td>
<td>00000410</td>
</tr>
<tr>
<td>readBIos.asaBios</td>
<td>00001300</td>
</tr>
<tr>
<td>checksum_bios</td>
<td>00000080</td>
</tr>
<tr>
<td>handler_writeBIOS</td>
<td>00000E00</td>
</tr>
<tr>
<td>handler_readBIOS</td>
<td>00000800</td>
</tr>
<tr>
<td>isPixOS</td>
<td>00001F00</td>
</tr>
<tr>
<td>fix_ip_csum_incr</td>
<td>00001E20</td>
</tr>
<tr>
<td>setupTable</td>
<td>000000F0</td>
</tr>
<tr>
<td>reverse2</td>
<td>00001E70</td>
</tr>
<tr>
<td>unsetupTable</td>
<td>000001F0</td>
</tr>
<tr>
<td>readBios</td>
<td>000002D0</td>
</tr>
<tr>
<td>Platform_5505</td>
<td>00002160</td>
</tr>
<tr>
<td>chipWrite.asaBios</td>
<td>000018D0</td>
</tr>
<tr>
<td>determineBIos</td>
<td>00000940</td>
</tr>
<tr>
<td>unlock_asaBios</td>
<td>00001380</td>
</tr>
<tr>
<td>NewChecksum</td>
<td>00001D50</td>
</tr>
<tr>
<td>compareNetscreenOSVersion</td>
<td>00002050</td>
</tr>
<tr>
<td>__i686.get_pc_thunk.x</td>
<td>00002125</td>
</tr>
<tr>
<td><em>GLOBAL_OFFSET_TABLE</em></td>
<td>00002DC0</td>
</tr>
<tr>
<td>handler_setCmos</td>
<td>00001D10</td>
</tr>
<tr>
<td>unlock_asaBios_5505</td>
<td>000015A0</td>
</tr>
<tr>
<td>entryPoint</td>
<td>00000E00</td>
</tr>
<tr>
<td>getPhysicalAddress</td>
<td>00000200</td>
</tr>
<tr>
<td>free</td>
<td>00002F00</td>
</tr>
<tr>
<td>cmosWriteByte</td>
<td>00001B70</td>
</tr>
<tr>
<td>OS_VER</td>
<td>00000065</td>
</tr>
<tr>
<td>_text</td>
<td>00002DAA</td>
</tr>
<tr>
<td>_start</td>
<td>00000000</td>
</tr>
<tr>
<td>GOT_START</td>
<td>00000070</td>
</tr>
</tbody>
</table>
if ( !isPixOS(x(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
{
    u2 = NET;
    u3 = x(NET + 4) < 0x7000u;
    u4 = x(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'
    u5 = &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) )
{
    u5 = &stdout;
    fwrite("Bios Write Address information could not be read\n", 1, 49, stdout);
    goto LABEL_7;
}
a1[7] = temp1;
return 1;
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
<table>
<thead>
<tr>
<th>Name</th>
<th>Action</th>
<th>Type</th>
<th>Subtype</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>8FEEECF1-BCFD-4A78-9231-4801566E..</td>
<td>File</td>
<td>Application</td>
<td></td>
<td>AbsoluteComputraceInstaller</td>
</tr>
<tr>
<td>PE32 image section</td>
<td></td>
<td>PE32 image</td>
<td></td>
<td></td>
</tr>
<tr>
<td>User Interface section</td>
<td></td>
<td>User Interface</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Version section</td>
<td></td>
<td>Version</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4EFCS1DA-23A6-4790-A292-4985C7F..</td>
<td>File</td>
<td>Dxe driver</td>
<td></td>
<td>LenovoComputraceEnablerDxe</td>
</tr>
<tr>
<td>Dxe dependency section</td>
<td></td>
<td>Dxe dependency</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PE32 image section</td>
<td></td>
<td>PE32 image</td>
<td></td>
<td></td>
</tr>
<tr>
<td>User Interface section</td>
<td></td>
<td>User Interface</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Version section</td>
<td></td>
<td>Version</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4589CBF3-03F0-4008-8D6F-26343C6..</td>
<td>File</td>
<td>Dxe driver</td>
<td></td>
<td>LenovoComputraceloaderDxe</td>
</tr>
<tr>
<td>Dxe dependency section</td>
<td></td>
<td>Dxe dependency</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PE32 image section</td>
<td></td>
<td>PE32 image</td>
<td></td>
<td></td>
</tr>
<tr>
<td>User Interface section</td>
<td></td>
<td>User Interface</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Version section</td>
<td></td>
<td>Version</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18578B75-D073-4203-90D2-8768A87..</td>
<td>File</td>
<td>Smm module</td>
<td></td>
<td>LenovoComputraceSmiServices</td>
</tr>
<tr>
<td>Smm dependency section</td>
<td></td>
<td>Smm dependency</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PE32 image section</td>
<td></td>
<td>PE32 image</td>
<td></td>
<td></td>
</tr>
<tr>
<td>User Interface section</td>
<td></td>
<td>User Interface</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Version section</td>
<td></td>
<td>Version</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4CD6586-CF78-4676-A079-16F7F96..</td>
<td>Smm</td>
<td>Smm module</td>
<td></td>
<td>LenovoSecuritySmmDispatch</td>
</tr>
<tr>
<td>621DE6CD-BF52-4EE3-A162-0BDE769..</td>
<td>Dxe</td>
<td>Dxe driver</td>
<td></td>
<td>LenovoRemoteConfigUpdateDxe</td>
</tr>
</tbody>
</table>

parseRegion: ME region is empty
parseVolume: unknown file system FFF12BBD-7696-4C88-A985-27478758F50
parseFile: non-empty pad-file contents will be destroyed after volume modifications.
Computrace/LoJack

**UEFI Environment**
- Computrace Configuration & Activation
  - LenovoComputraceLoaderDxe
  - LenovoComputraceEnablerDxe
  - AbsoluteComputraceInstallerDxe
  - LenovoComputraceSmiServices

**OS Environment**
- Computrace Agent
- OS Process
- OS Process
- OS Process
- Network Interface
- Computrace C&C Servers

**OS NTFS Volume**
BIOS Update Issues
Lenovo BIOS Update on MS Win10 with Device Guard
Forensic Approaches
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


```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" }
}
```


```
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

**Reader**
- Write start address to FADDR
- Write size of data to read to HSFC
- Write read command to HSFC
- Set FGO (0x0001) bit in HSFC
- Wait for SPI read cycle completion
- Read data from FDATAX registers

**SPI Controller**
- FADDR: Flash Linear Address
- HSFC:
  - DMA Channel (FDBC)
  - Flash Cycle Count (FCYCLE)
  - FGO
- HSFS:
  - Status Registers
- FDATAX: Data
### SPI Flash Dump – Attacker’s Possibilities

<table>
<thead>
<tr>
<th>HSFC:</th>
<th>FSMIE</th>
<th>FDBC</th>
<th>FCYCLE</th>
<th>FGO</th>
</tr>
</thead>
</table>

**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

**Reader**
- Write start address to FADDR
- Write size of data to read to HSFC
- Write read command to HSFC
- Set FGO (0x0001) bit in HSFC
- Wait for SPI read cycle completion
- Read data from FDATAX registers

**SPI Controller**

**Attacker**
- Set FSMIE bit to 1 in HSFC
- Once FDONE is set to 1 SMI is triggered
- Write fake data to FDATAX registers
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?

How Debug UEFI Firmware?

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

How to enter SMM

Debugger Commands

SPECIAL BREAK 0 ON "SMM Entry Break": enabled (S=0,C=0)
SPECIAL BREAK 1 ON "SMM Exit Break": enabled (S=0,C=0)
INFO: Resetting target, this may take a moment...
execution stopped by "Halt Command break"
	db> IA32CPU "read msr 0x9e"
ERROR: Couldn’t read MSR 0x9e: The CPU faulted when accessing an MSR.
	-db> SET PORT 0xB2 = 1
WARNING: Multiple breaks, context is set to the most interesting.
program stopped: SPECIAL BREAK 'SMM Entry Break' (ID=0) at "0x0900:0x00000000"
Few words about UEFI Firmware Mitigations
Exploiting AMI Aptio firmware on example of Intel NUC

Rootkits and Bootkits

Reversing Modern Malware and Next Generation Threats

Alex Matrosov, Eugene Rodionov, and Sergey Bratus

nostarch.com/rootkits
ZERONIGHTS 2016

HACK O'CLOCK

Thank you for your attention!

Y SO MANY QUESTIONS?

Alex Matrosov
@matrosov

Eugene Rodionov
@vxradius