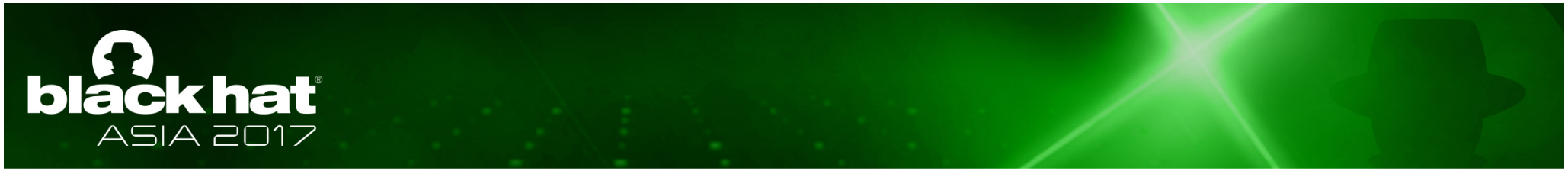


UEFI Firmware Rootkits: Myths and Reality

(Revisited for Black Hat Asia)

Alex Matrosov
@matrosov

Eugene Rodionov
@vxradius

Who We Are: Alex Matrosov

Have fun with UEFI Security and RE at CYLANCE

Former Firmware Security Researcher @Intel

@matrosov

Who We Are: Eugene Rodionov

Senior Security Researcher @Intel

Previously @ESET

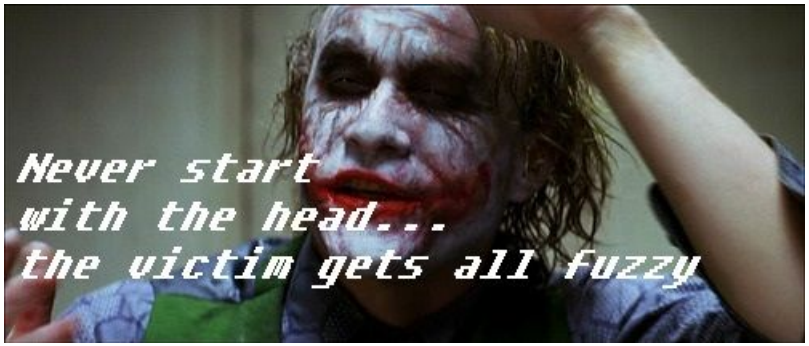
@vxradius

I don’t speak for my employer. All the opinions and information presented here is my responsibility

What is different with previous talks?

Agenda

- Intro
- UEFI Rootkit Infection
- BIOS Rootkits In-The-Wild
  - ✓ HackingTeam Rootkit
  - ✓ BIOS Implants
  - ✓ Computrace/LoJack
- UEFI Ransomware Story
  - ✓ Vulns
  - ✓ Disclosure
  - ✓ DEMO
- MS Device Guard bypass from UEFI (CVE-2016-8222)
- Forensic Approaches
- Mitigations



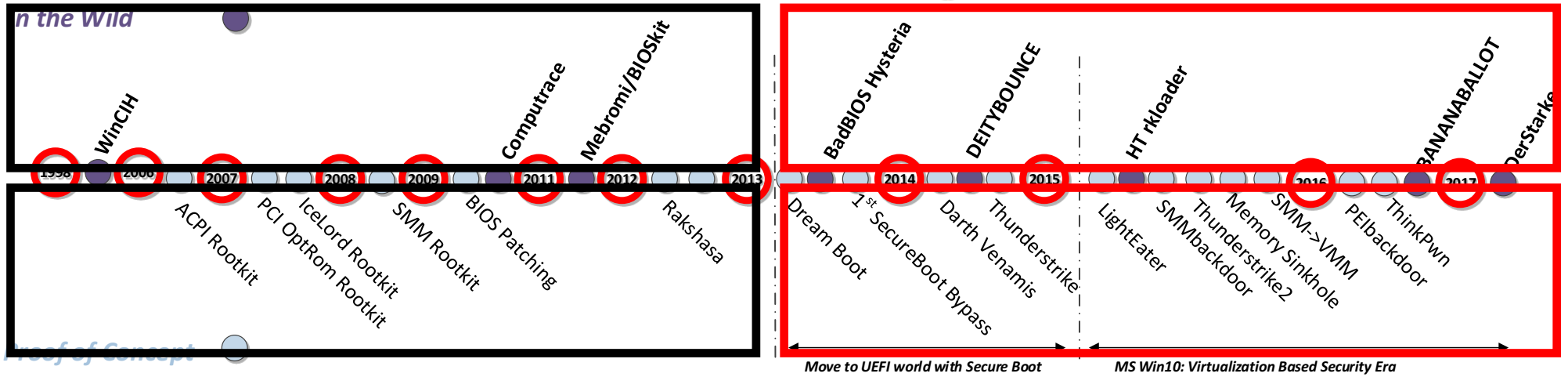
History of BIOS rootkits


Damaged

History of BIOS rootkits

Low Threat Activity

Targeted Attacks



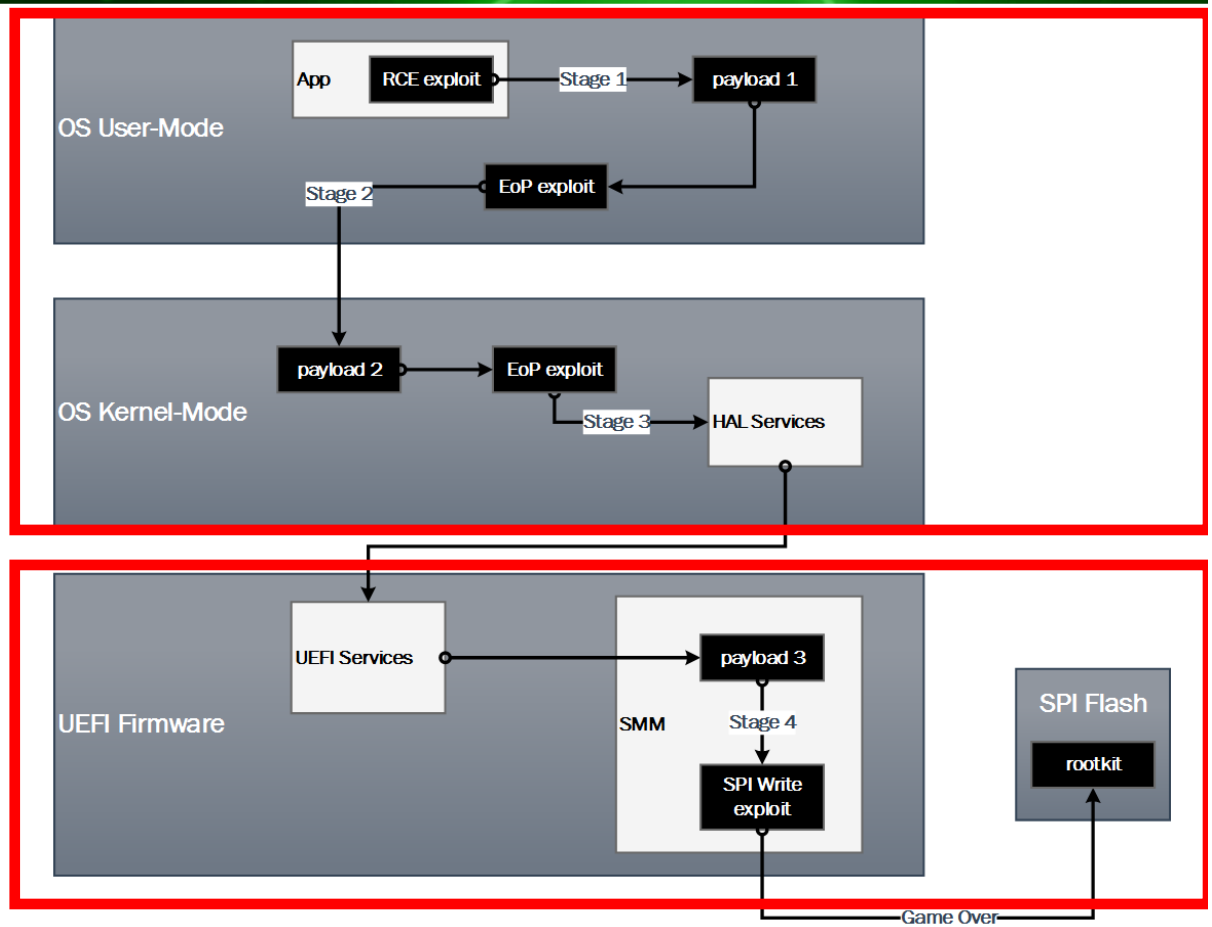
Low Research Activity

High Research Activity

UEFI Rootkit Infection

Firmware Rootkits Infection

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



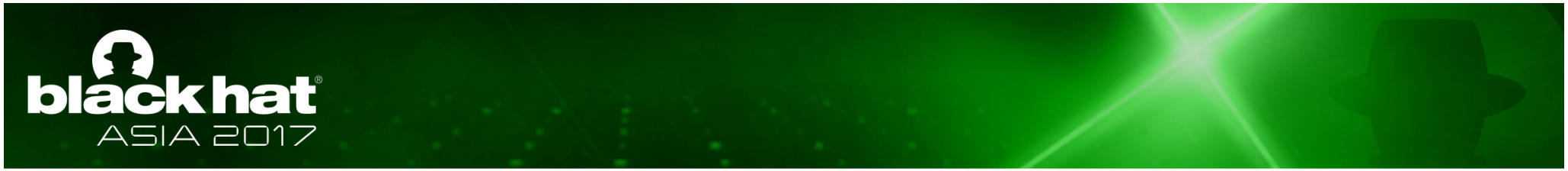
Recent Vulns For Persistent Infection

- **SMI Handlers(always an issue) –** memory corruption vulnerabilities can lead arbitrary SMM code execution.

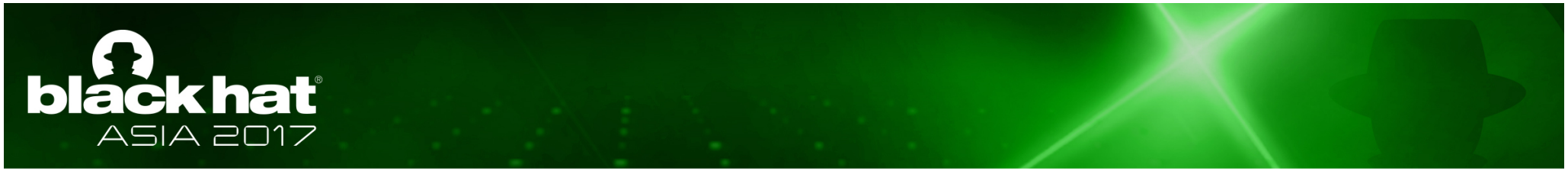
- **S3BootScript(VU#976132) -** arbitrary modification of platform firmware. Allows attacker arbitrarily read/write to the SMRAM region.

- **ThinkPwn(LEN-8324) –** arbitrary SMM code execution exploit for multiple BIOS vendors. Allows attacker to disable flash write protection and modify platform firmware

- **Aptiocalypsis(INTEL-SA-00057) -** arbitrary SMM code execution exploit for AMI Aptio based firmware. Allows attacker to disable flash write protection and modify platform firmware



BIOS Rootkits In-The-Wild



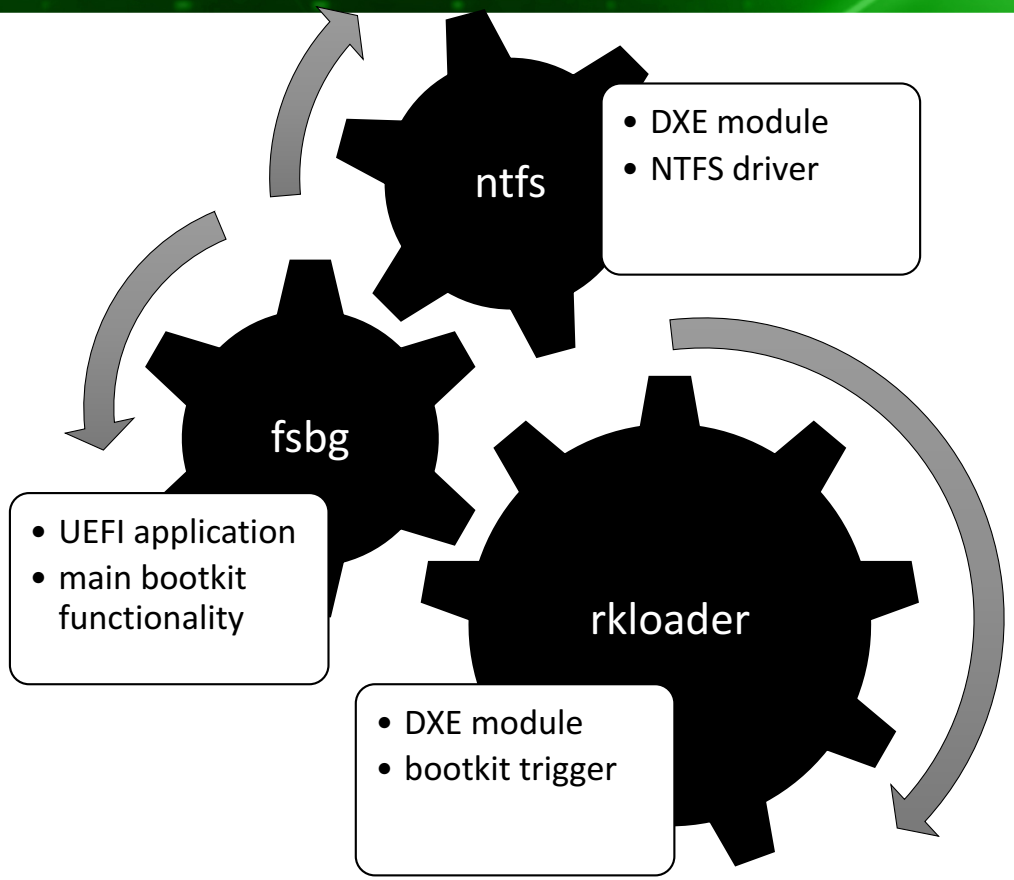
HakingTeam Vector-EDK

Hacking Team UEFI Implant

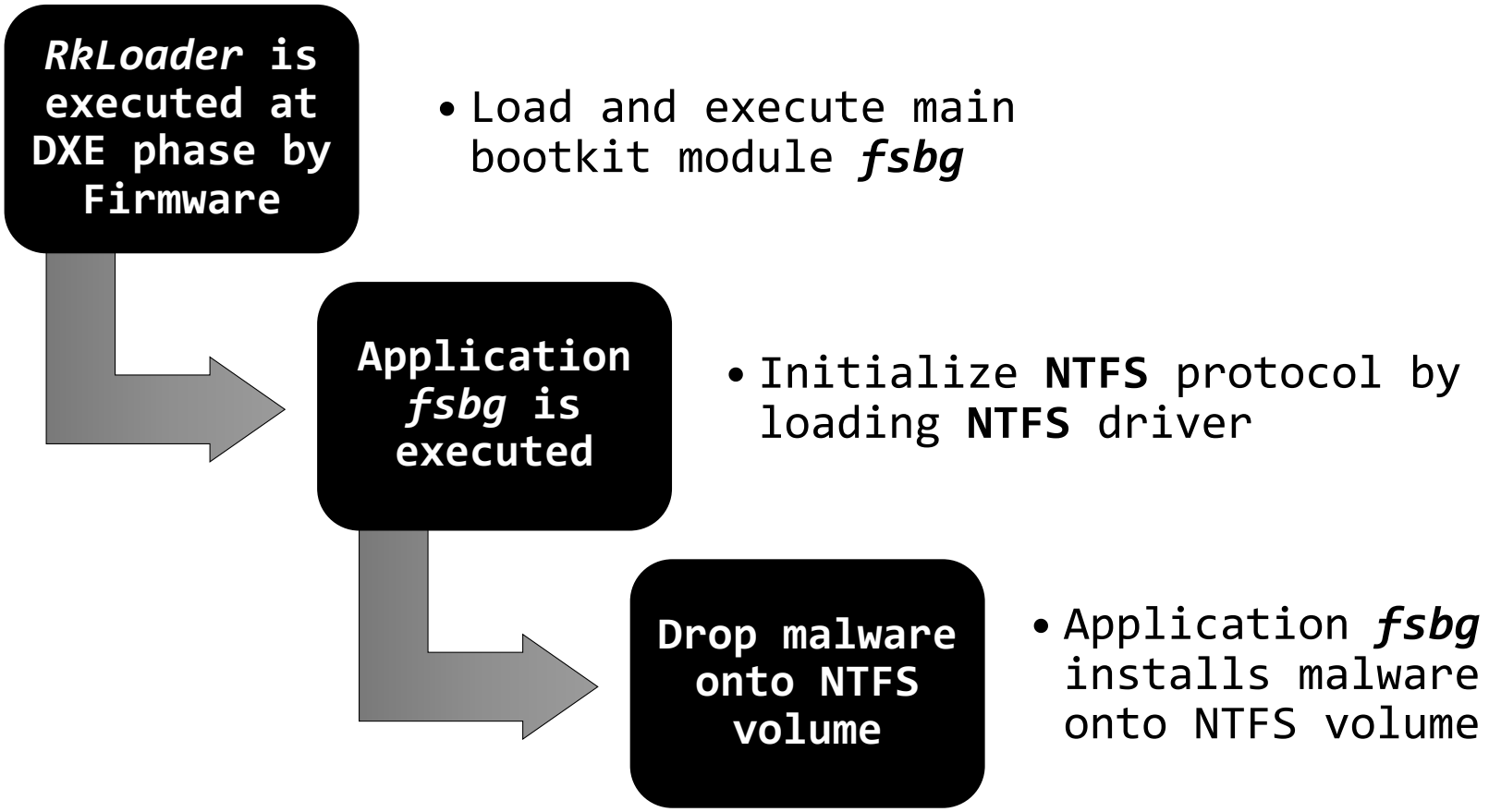
- First* discovery of non-PoC UEFI Rootkit/Implant
- Persistent copy of malicious agent inside SPI flash
- Infect OS from UEFI firmware

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

Hacking Team UEFI Implant : Modules



HT UEFI Implant: How It Works

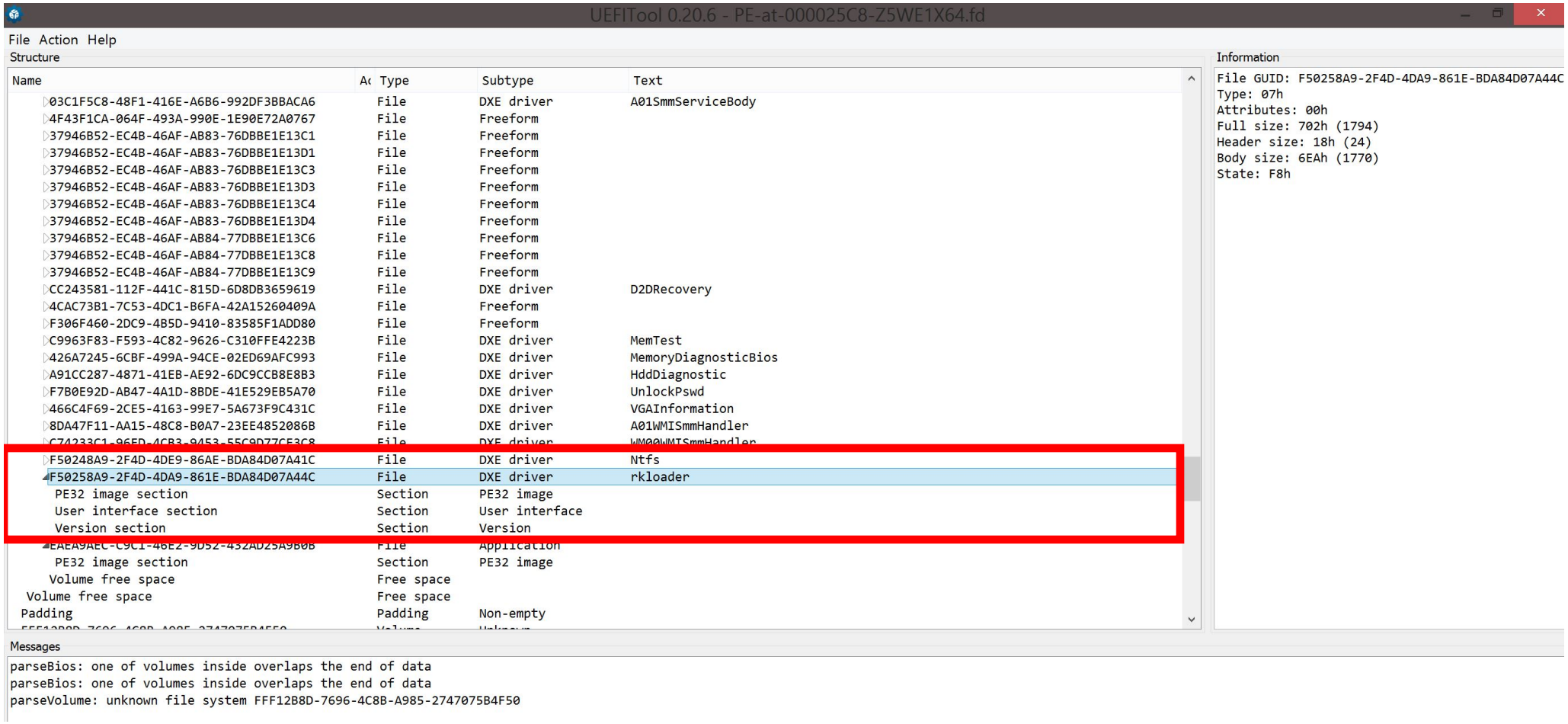


HT Implant Deployment

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[


UEFITool 0.20.6 - PE-at-000025C8-Z5WE1X64.fd
File Action Help
Structure
Name
Type
Subtype
Text
03C1F5C8-48F1-416E-A6B6-992DF3BBACA6 File DXE driver A01SmmServiceBody
4F43F1CA-064F-493A-990E-1E90E72A0767 File Freeform
37946B52-EC4B-46AF-AB83-76DBBE1E13C1 File Freeform
37946B52-EC4B-46AF-AB83-76DBBE1E13D1 File Freeform
37946B52-EC4B-46AF-AB83-76DBBE1E13C3 File Freeform
37946B52-EC4B-46AF-AB83-76DBBE1E13D3 File Freeform
37946B52-EC4B-46AF-AB83-76DBBE1E13C4 File Freeform
37946B52-EC4B-46AF-AB83-76DBBE1E13D4 File Freeform
37946B52-EC4B-46AF-AB84-77DBBE1E13C6 File Freeform
37946B52-EC4B-46AF-AB84-77DBBE1E13C8 File Freeform
37946B52-EC4B-46AF-AB84-77DBBE1E13C9 File Freeform
CC243581-112F-441C-815D-6D8DB3659619 File DXE driver D2DRecovery
4CAC73B1-7C53-4DC1-B6FA-42A15260409A File Freeform
F306F460-2DC9-4B5D-9410-83585F1ADD80 File Freeform
C9963F83-F593-4C82-9626-C310FFE4223B File DXE driver MemTest
426A7245-6CBF-499A-94CE-02ED69AFC993 File DXE driver MemoryDiagnosticBios
A91CC287-4871-41EB-AE92-6DC9CCB8E8B3 File DXE driver HddDiagnostic
F7B0E92D-AB47-4A1D-8BDE-41E529EB5A70 File DXE driver UnlockPswd
466C4F69-2CE5-4163-99E7-5A673F9C431C File DXE driver VGAInformation
8DA47F11-AA15-48C8-B0A7-23EE4852086B File DXE driver A01WMISmmHandler
F50248A9-2F4D-4DE9-86AE-BDA84D07A41C File DXE driver Ntfs
F50258A9-2F4D-4DA9-861E-BDA84D07A44C File DXE driver rkloader
PE32 image section Section PE32 image
User interface section Section User interface
Version section Section Version
PE32 image section Section PE32 image
Volume free space Free space
Volume free space Free space
Padding Padding Non-empty
Information
File GUID: F50258A9-2F4D-4DA9-861E-BDA84D07A44C
Type: 07h
Attributes: 00h
Full size: 702h (1794)
Header size: 18h (24)
Body size: 6EAh (1770)
State: F8h
Messages
parseBios: one of volumes inside overlaps the end of data
parseBios: one of volumes inside overlaps the end of data
parseVolume: unknown file system FFF12B8D-7696-4C8B-A985-2747075B4F50


I'M NOT A
MONSTER

DEITYBOUNCE

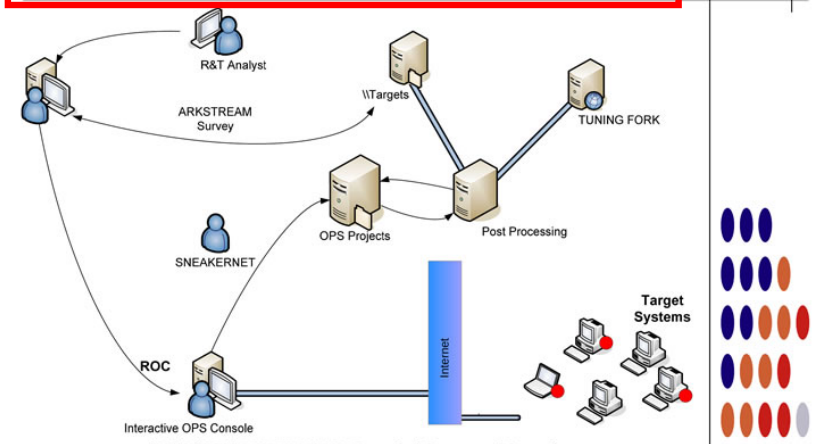
SECRET//COMINT//REL TO USA, FVEY

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

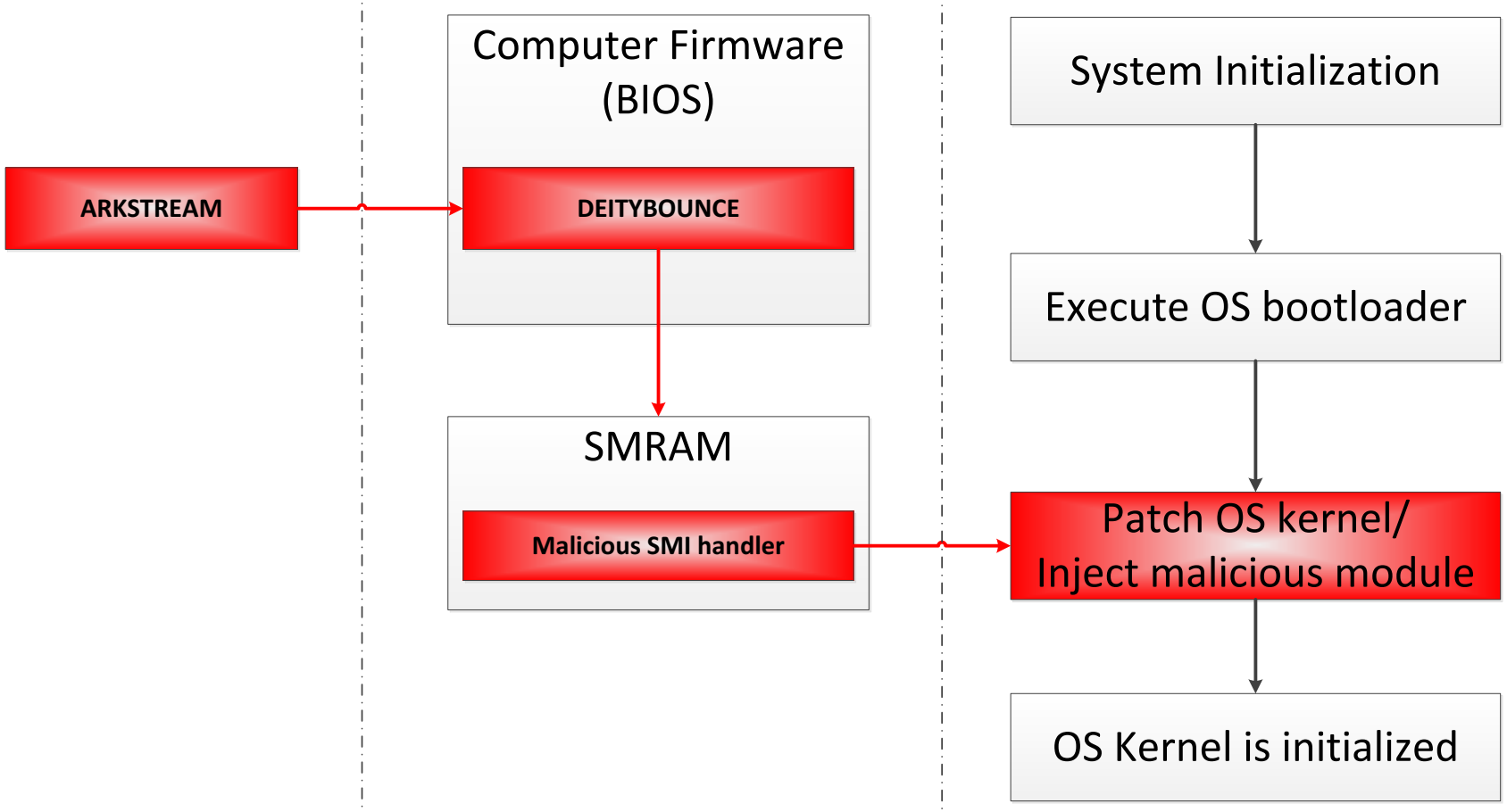
SECRET//COMINT//REL TO USA, FVEY

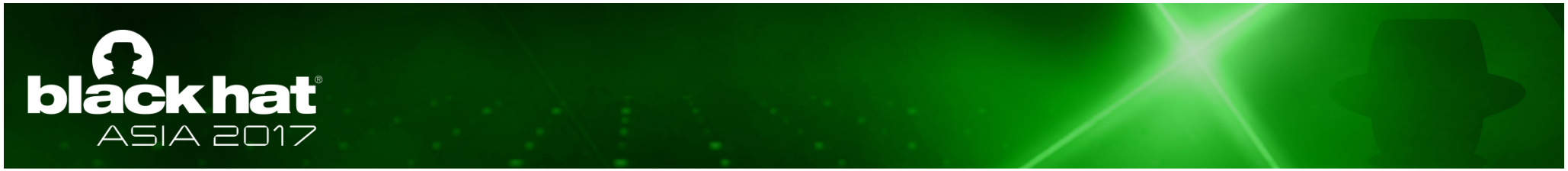
- Only Snowden-leaked documentation is available for analysis
- Safe to assume that servers use legacy BIOS[1]

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

black hat ASIA 2017

DEITYBOUNCE Workflow





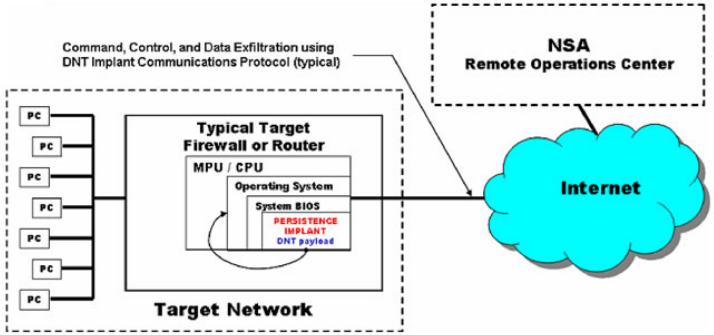
BANANABALLOT and JETPLOW (Equation Group)

JETPLOW

ANT Product Data

06/24/08

(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//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).

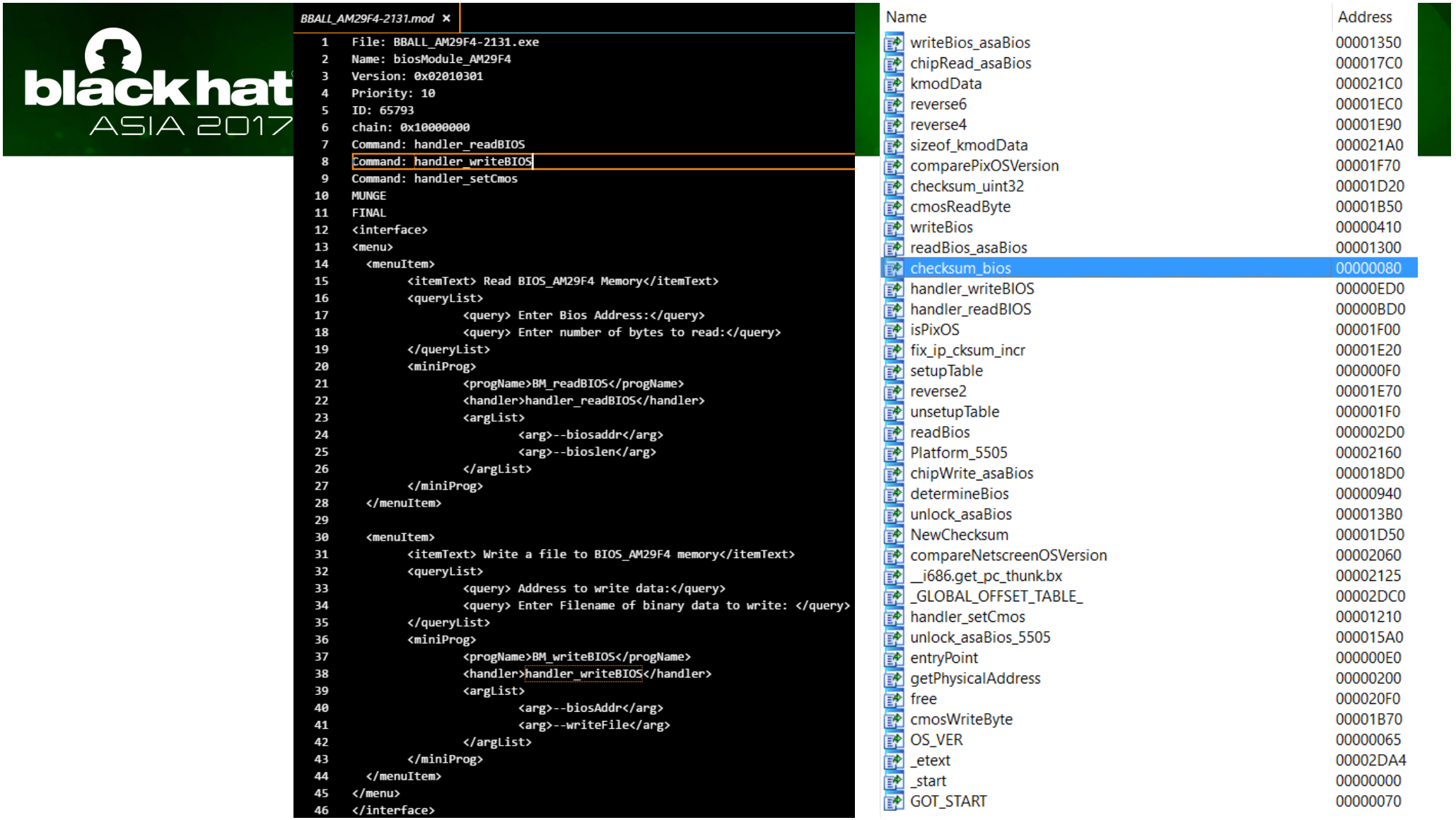
(TS//SI//REL) A typical JETPLOW deployment on a target firewall with an exfiltration path to the Remote Operations Center (ROC) is shown above. JETPLOW is remotely upgradeable and is also remotely installable provided BANANAGLEE is already on the firewall of interest.

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


black hat
ASIA 2017
BBALL_AM29F4-2131.mod
1 File: BBALL_AM29F4-2131.exe
2 Name: biosModule_AM29F4
3 Version: 0x02010301
4 Priority: 10
5 ID: 65793
6 chain: 0x10000000
7 Command: handler_readBIOS
8 Command: handler_writeBIOS
9 Command: handler_setCmos
10 MUNGE
11 FINAL
12 <interface>
13 <menu>
14 <menuItem>
15 <itemText> Read BIOS_AM29F4 Memory</itemText>
16 <queryList>
17 <query> Enter Bios Address:</query>
18 <query> Enter number of bytes to read:</query>
19 </queryList>
20 <miniProg>
21 <progName>BM_readBIOS</progName>
22 <handler>handler_readBIOS</handler>
23 <argList>
24 <arg>--biosaddr</arg>
25 <arg>--bioslen</arg>
26 </argList>
27 </miniProg>
28 </menuItem>
29
30 <menuItem>
31 <itemText> Write a file to BIOS_AM29F4 memory</itemText>
32 <queryList>
33 <query> Address to write data:</query>
34 <query> Enter Filename of binary data to write: </query>
35 </queryList>
36 <miniProg>
37 <progName>BM_writeBIOS</progName>
38 <handler>handler_writeBIOS</handler>
39 <argList>
40 <arg>--biosAddr</arg>
41 <arg>--writeFile</arg>
42 </argList>
43 </miniProg>
44 </menuItem>
45 </menu>
46 </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;
}
```

Vault7: DerStarke

DerStarke Implant

- As infection method used DarkDream (S3BootScript exploit VU#976132) which is patched in 2015
- Can hook firmware update process on-the-fly to keep malicious code active after firmware update
- According to leaked information it is focused on attacks against Apple hardware
- Contains UEFI BIOS components to perform its activities

DerStarke Components

- Loader (L.efi) - main loader module that delivers/cleans implant components on/from OS
- VerboseInstaller (VI.efi) - DXE application that is responsible for writing DXE/PEI drivers to firmware image or recovery partitions onto SPI image
- AppInstaller (AI.efi) - DXE application that uses DarkDream exploit to execute VerboseInstaller

DerStarke Components

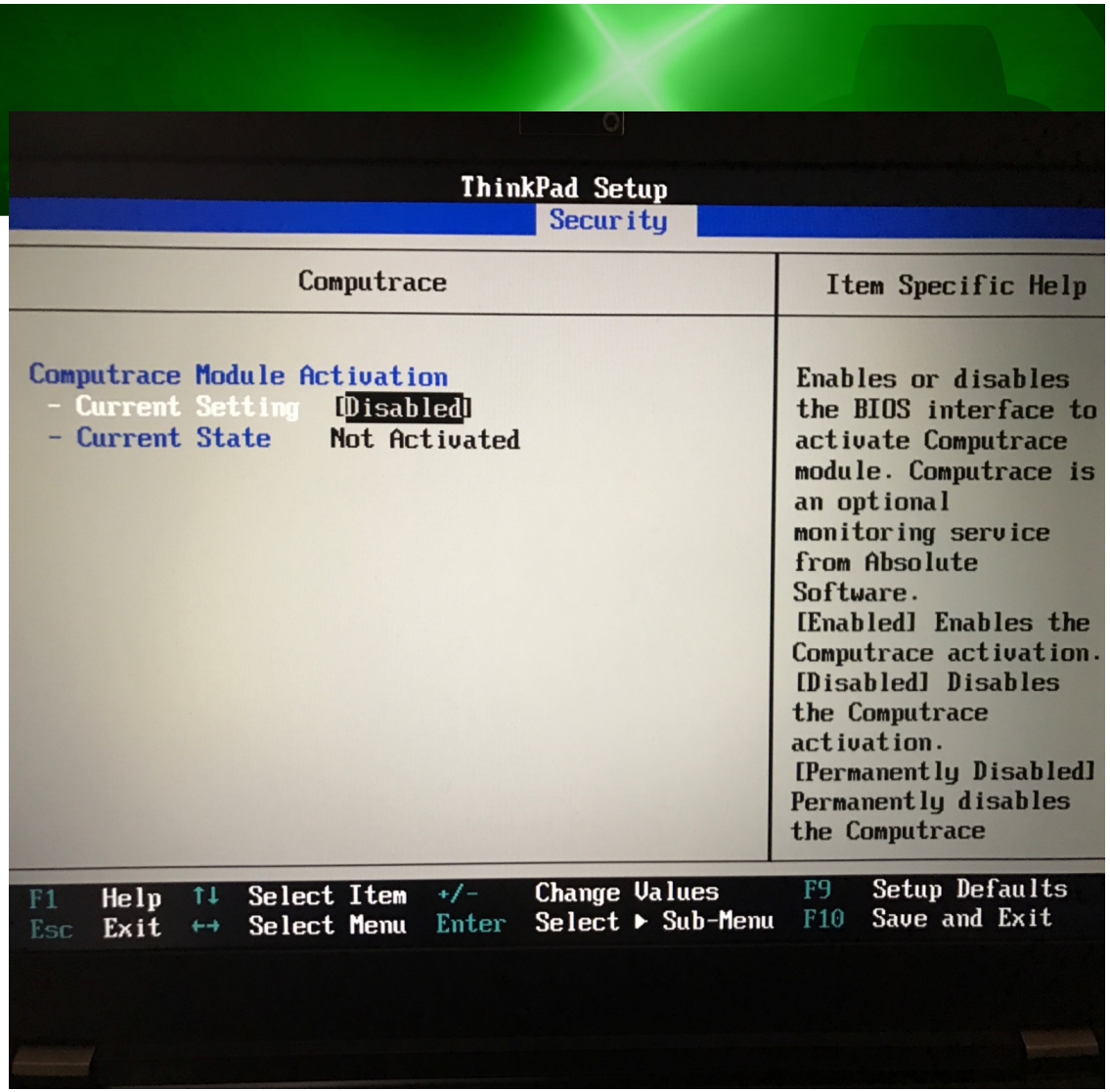
- S3Sleep (S.efi) - DXE driver with DarkDream exploit
- PeiUnlock (PU.efi) - PEI driver that keeps SPI flash unlocked
- PeiLoader (PL.efi) - PEI driver used to hook firmware updates
- DxeInjector (DI.efi) - DXE driver that is used by PeiLoader to inject the old implants from firmware image into the new update capsule
- FwUpdate (FW.efi) - DXE driver for remote update of Loader and PeiLoader into SPI flash

Computrace/LoJack

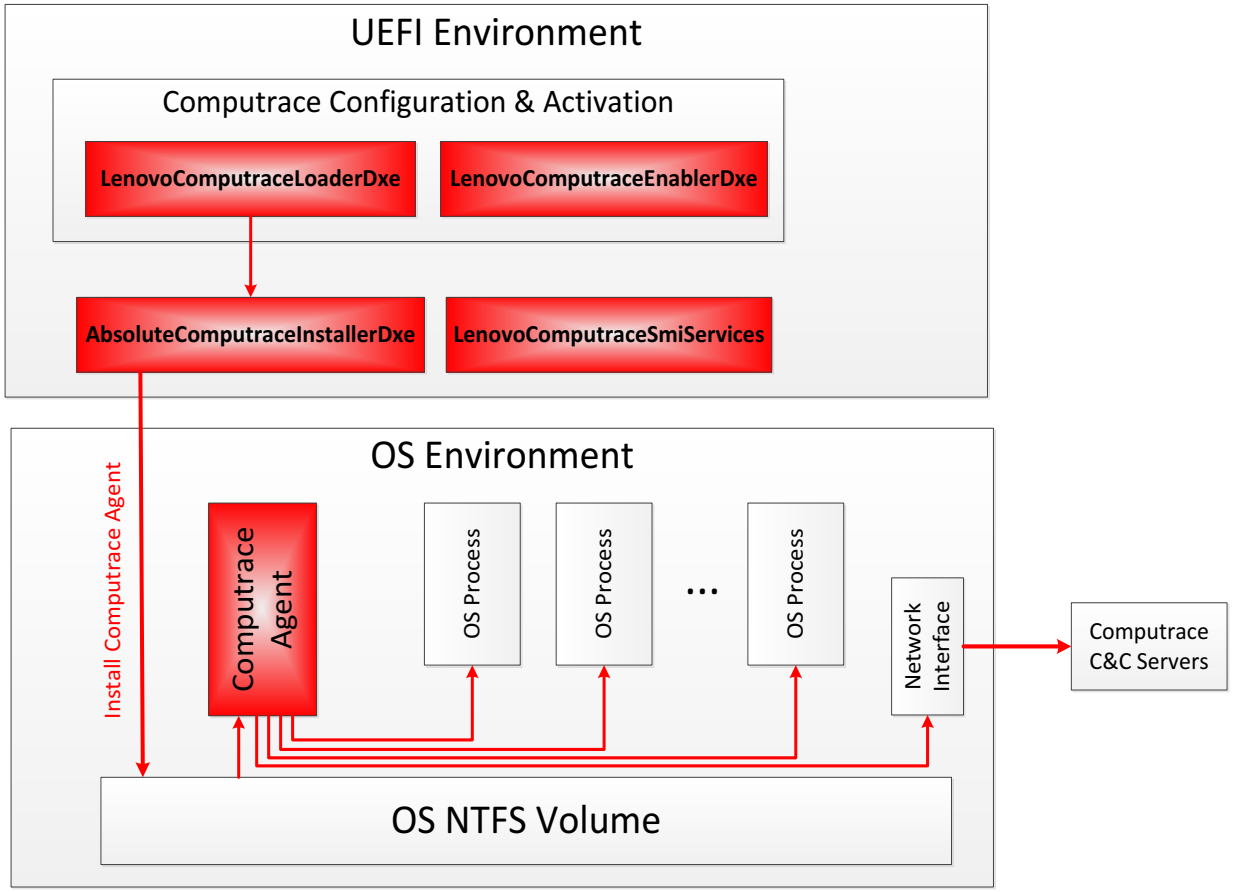
Computrace/LoJack

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

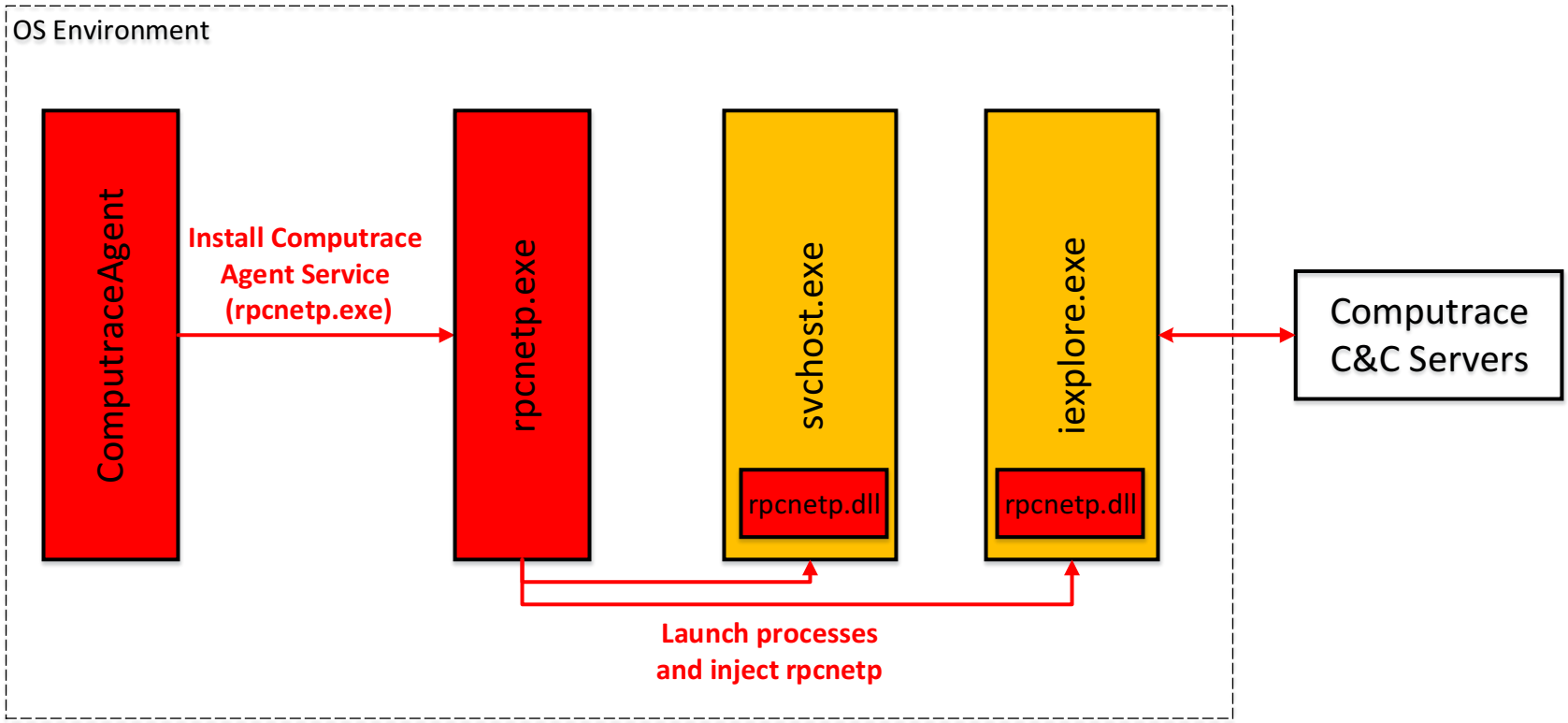

AND HERE WE GO!


ThinkPad Setup
Security
Computrace
Item Specific Help
Computrace Module Activation
- Current Setting [Disabled]
- Current State Not Activated
Enables or disables the BIOS interface to activate Computrace module. Computrace is an optional monitoring service from Absolute Software.
[Enabled] Enables the Computrace activation.
[Disabled] Disables the Computrace activation.
[Permanently Disabled] Permanently disables the Computrace
F1 Help ↑↓ Select Item +/- Change Values F9 Setup Defaults
Esc Exit ↔ Select Menu Enter Select ▶ Sub-Menu F10 Save and Exit

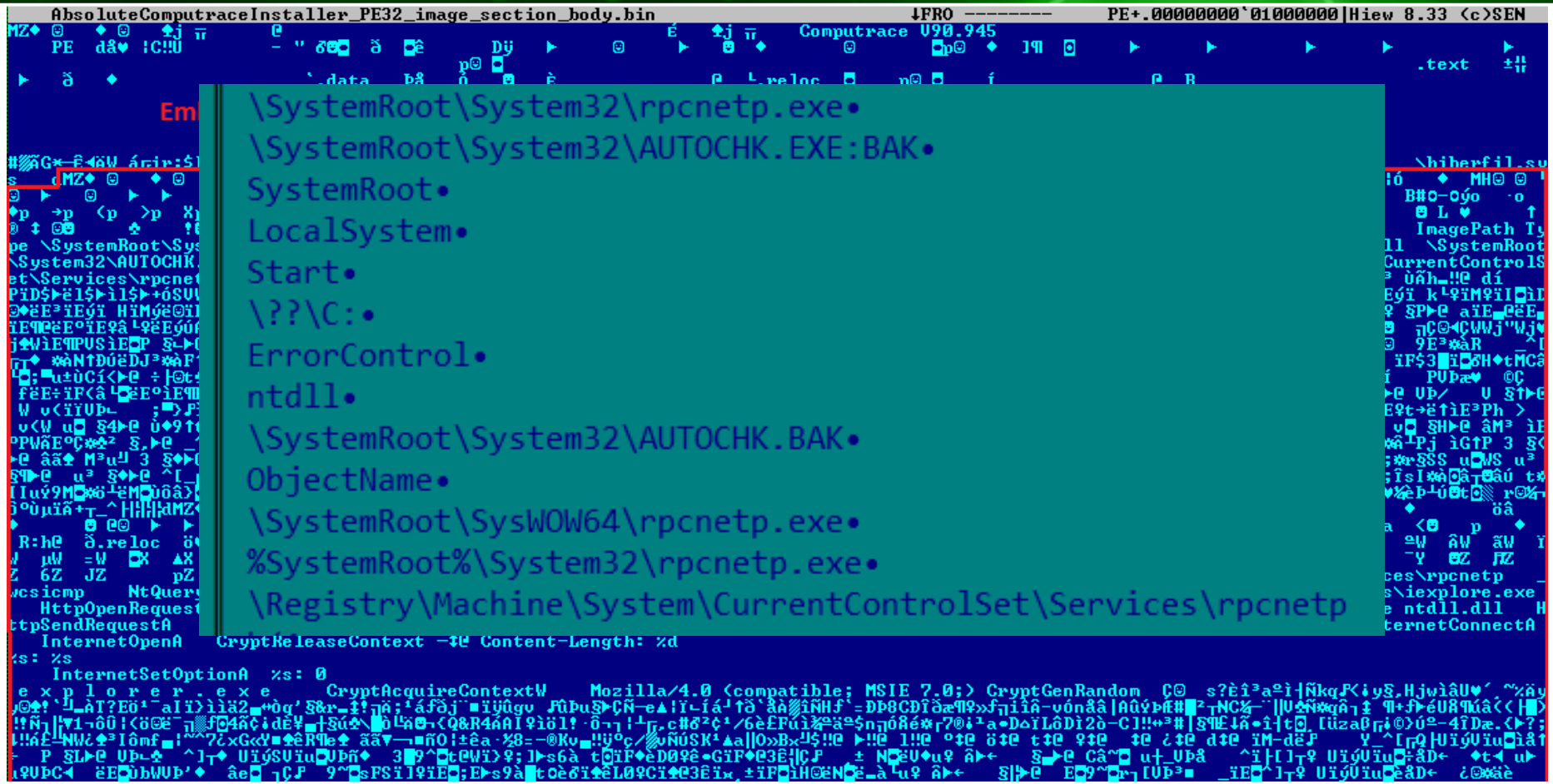
Computrace/LoJack Components



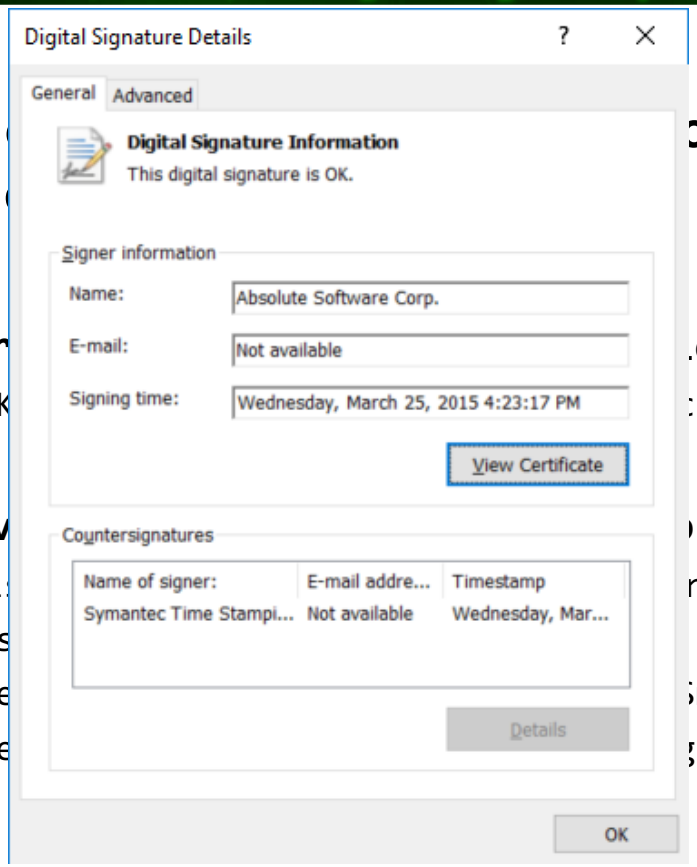
Computrace/LoJack: OS Components

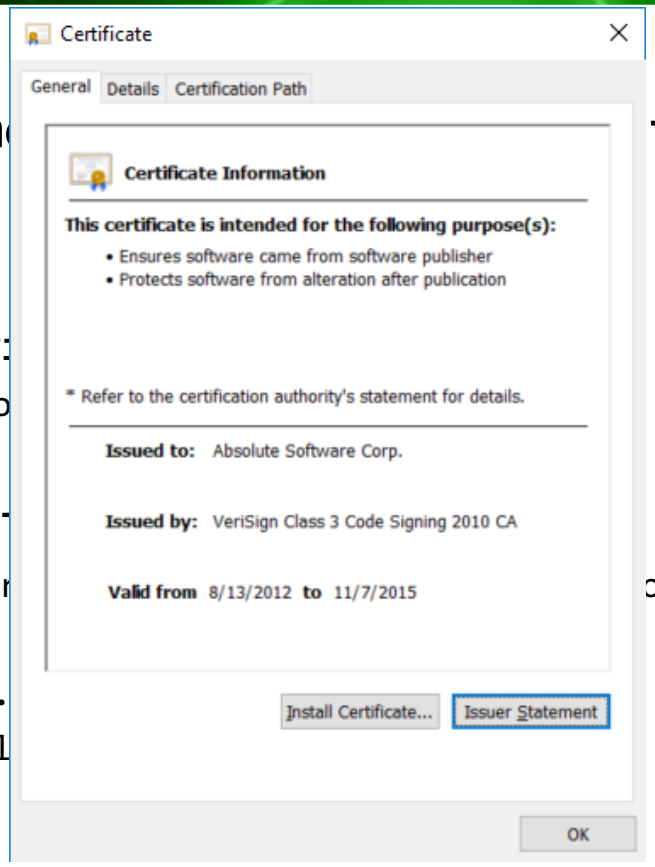


Computrace/LoJack Layout



ComputraceAgentInstaller





Computrac **o m** **trace**
Agent int

- **Legacy: r** **e wi**
  - ✓ AUTOCHK **catio**
- **Modern: v** **rm**
  - ✓ WBPT i **rmwar** **or execution**
  - ✓ WBPT is
  - ✓ The exe **smss.**
  - ✓ The exe **gital**

** Alex Ionescu, ACPI 5.0 Rootkit Attacks "Againts" Windows 8 // http://alex-ionescu.com/Publications/SyScan/syscan2012.pdf*

ComputraceAgentInstaller: Replacing AUTHOCK.EXE

- Implements custom NTFS parsing functionality to install the “agent” onto NTFS volumes

```
struct_1 *install_agent_on_system_volume()
{
  v0 = 0;
  get_block_protocol(0, &v3);
  bld_str_cat_2(&v2, &aMulti0Disk0Rdi[35], "\\System32");// WINDOWS\System32
  do
    resu
  while
  aMulti
  return
}

__int64
{
  install_agent_on_system_volume();
  find_file_volume(0i64, 0i64, "\\BOOT.INI", install_agent_on_volume);
  find_file_volume(0i64, 0i64, "\\MSDOS.SYS", install_agent_on_volume);
  return 0i64;
}
```

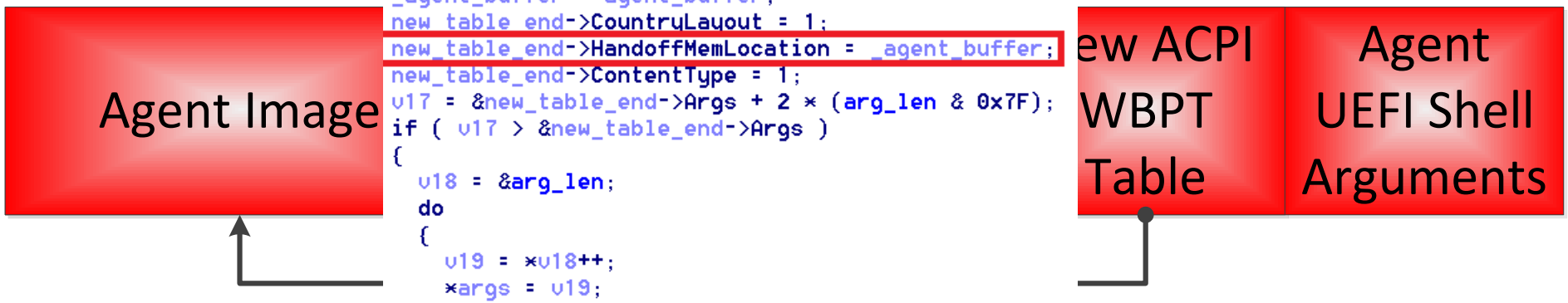
```
if ( v14 != autochk_magic )
{
  result = find_file_in_fs(v3, v2, &v11, "AUTOCHK.EXE");
  if ( result )
  {
    v8 = 4i64;
    result = replace_file(v2, &v14, &v10, &v8, 0);
```

ComputraceAgentInstaller Using WBPT

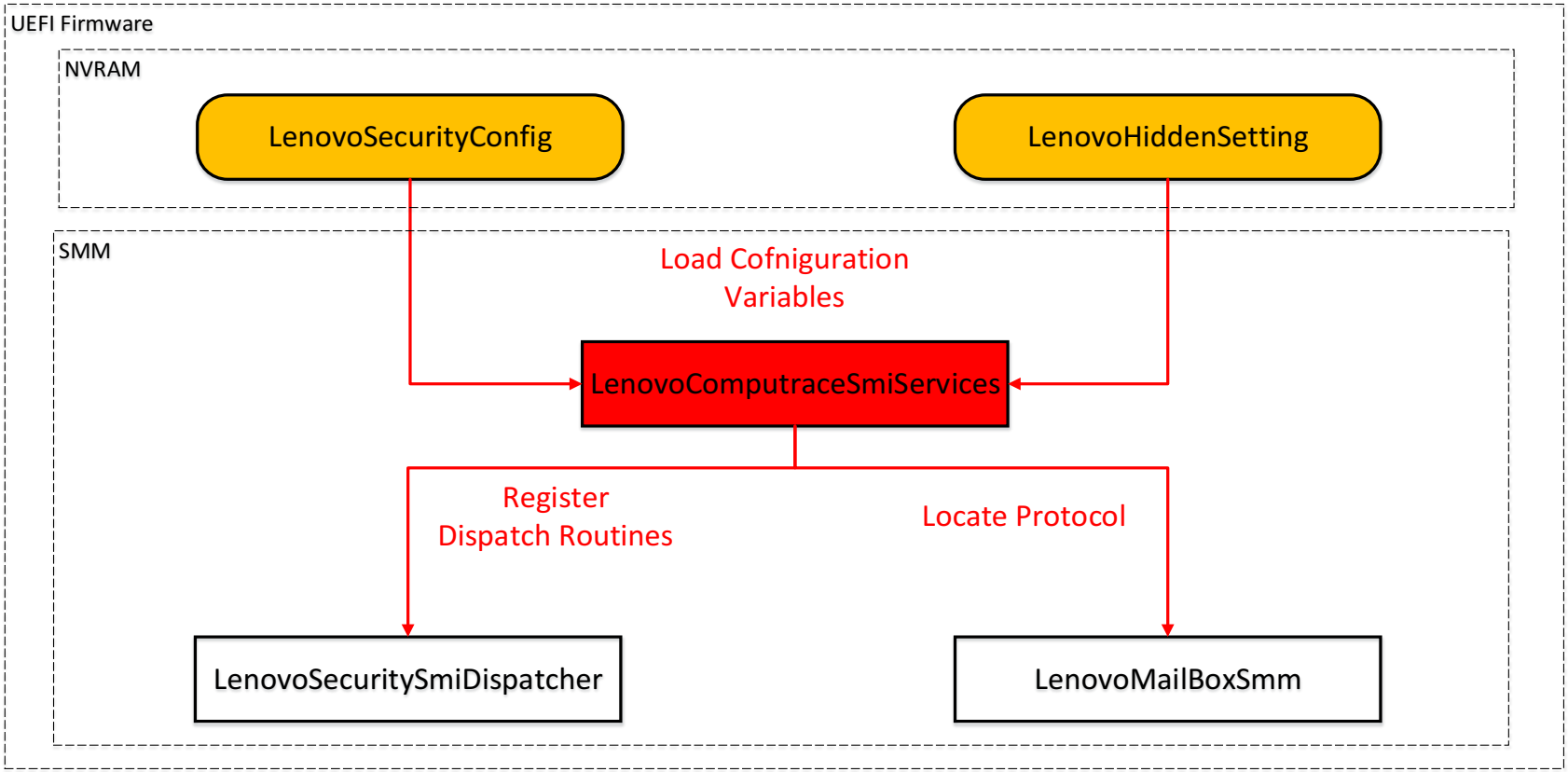
- Installs WBI .e

BootServic *COL->InstallAcpiTable*

```
*&new_table_end->OemId[0] = '-TBA';
*&new_table_end->OemTableId[0] = '-TBA';
new_table_end->Revision = 1;
new_table_end->Checksum = 0;
new_table_end->Signature = 'TBPW';
*&new_table_end->OemTableId[4] = 'TBPW';
*&new_table_end->OemId[4] = 'TN';
new_table_end->OemRevision = 1;
new_table_end->CreatorId = 'WTBA';
new_table_end->CreatorRevision = 0x20120402;
args = &new_table_end->Args;
new_table_end->HandoffMemSize = 0x8570;
_agent_buffer = agent_buffer;
new_table_end->CountryLayout = 1;
new_table_end->HandoffMemLocation = _agent_buffer;
new_table_end->ContentType = 1;
v17 = &new_table_end->Args + 2 * (arg_len & 0x7F);
if ( v17 > &new_table_end->Args )
{
  v18 = &arg_len;
  do
  {
    v19 = *v18++;
    *args = v19;
    args += 2;
  }
  while ( args <= v17 );
}
```



ComputraceSmiServices: Components



ComputraceSmiServices: Initialization

```
signed __int64 __fastcall InitializeComputraceSmi(__int64 a1, EFI_SYSTEM_TABLE *a2)
{
  v2 = a2->BootServices;
  Imagehandle = a1;
  v3 = a2;
  SystemTable = a2;
  BootServices = v2;
  (v2->LocateProtocol)(&EFI_SMM_BASE2_PROTOCOL_GUID, 0i64, &EFI_SMM_BASE2_PROTOCOL_0);
  (EFI_SMM_BASE2_PROTOCOL_0->InSmm)(EFI_SMM_BASE2_PROTOCOL_0, &inSmm);
  if ( inSmm )
    (EFI_SMM_BASE2_PROTOCOL_0->GetSmstLocation)(EFI_SMM_BASE2_PROTOCOL_0, &SmstLocation);
  (BootServices->LocateProtocol)(&EFI_SMM_ACCESS2_PROTOCOL_GUID, 0i64, &EFI_SMM_ACCESS2_PROTOCOL);
  SmramMemoryMapSize = 0i64;
  (EFI_SMM_ACCESS2_PROTOCOL->GetCapabilities)(EFI_SMM_ACCESS2_PROTOCOL, &SmramMemoryMapSize, 0i64);
  SmramMap = AllocSmmMemory(v4, SmramMemoryMapSize);
  EFI_SMM_ACCESS2_PROTOCOL->GetCapabilities(EFI_SMM_ACCESS2_PROTOCOL, &SmramMemoryMapSize, SmramMap);
  gSmramMemoryMapSize = SmramMemoryMapSize >> 5;
  v5 = v3->RuntimeServices;
  VendorTable = 0i64;
  v6 = SystemTable->NumberOfTableEntries;
  RuntimeServices = v5;
}
```

ComputraceSmiServices: Register Callbacks

```
signed __int64 RegisterComputraceSmi()
{
  v0 = 0i64;
  if ( SmstLocation )
    v0 = SmstLocation;
  qword_EB8 = v0;
  if ( (SmstLocation->SmmLocateProtocol
          &LENOVO_SECURITY_SMI_DISPATCH
          0i64,
          &LENOVO_SECURITY_SMI_DISPATCH
    return 0x8000000000000003i64;
  if ( (SmstLocation->SmmLocateProtocol
    return 0x8000000000000003i64;
  zeromem(&security_settings, 7ui64);
  if ( InitializeSecurityConfiguration(
    return 0x8000000000000003i64;
  v3 = Handler_1;
  v4 = 0i64;
  if ( Handler_1 )
  {
    v5 = 0i64;
    do
    {
      (*LENOVO_SECURITY_SMI_DISPATCH_PROTOCOL)(LENOVO_SECURITY_SMI_DISPATCH_PROTOCOL, ServicesTable[v5], v3);
      v5 = 2 * ++v4;
      v3 = ServicesTable[2 * v4 + 1];
    }
    while ( v3 );
  }
  return 0i64;
}
```

```
text:0000000000000300 qword_300       dq 4CC90D4FA2C1808Fh, 499DD341E6D119A6h
text:0000000000000300
text:0000000000000310 ; __int64 ServicesTable[]
text:0000000000000310 ServicesTable   dq 85h
text:0000000000000318 off_318         dq offset Handler_1
text:0000000000000320                 dq 87h
text:0000000000000328                 dq offset Handler_2
text:0000000000000330                 dq 88h
text:0000000000000338                 dq offset Handler_3
text:0000000000000340                 db 0Bh, 0
text:0000000000000342                 align 8
```

UEFI Ransomware Story

Disclosure Timeline
(CLVA-2016-12-001/002)

- ✓ Discovery Date: 2016-12-15
- ✓ Gigabyte Notification Date: 2017-01-03
- ✓ CERT/CC Contact Date: 2017-01-17
- ✓ CERT/CC Acknowledgement (VU#507496) Date: 2017-01-20
- ✓ Disclosure Notification Date: 2017-01-26
- ✓ AMI Contact Date: 2017-02-10
- ✓ AMI Response Date: 2017-02-20
- ✓ Disclosure Revised Date: 2017-03-06
- ✓ CERT/CC Acknowledged Revised Disclosure Date: 2017-03-07
- ✓ Gigabyte Working on the Patch: 2017-03-21
- ✓ Public Disclosure Date: 2017-03-31

https://github.com/CylanceVulnResearch/disclosures

Target Platform

- **Gigabyte (GB-BSi7HA-6500)**
  - ✓ Intel 6th generation Core i7 CPU (Skylake)
  - ✓ BIOS Lock (BLE) – ENABLED (not enabled by default)

- **MS Windows 10 Enterprise**
  - ✓ All latest updates installed
  - ✓ Virtualization Based Security (VBS) - ENABLED
  - ✓ Device Guard - ENABLED
  - ✓ Secure Boot – ENABLED (OS + BIOS)

Vertical Markets

- School
- University computer labs
- Libraries
- Hospital / Medical equipment
- Governmental

Powerful Commercial Applications

- Factory testing machine
- Bank ATM system
- Gaming equipment
- Vending machine
- Security system

copy from Gigabyte official website


HA
HA
HA
HA
WHY SO SERIOUS?
JOKER
JOKER

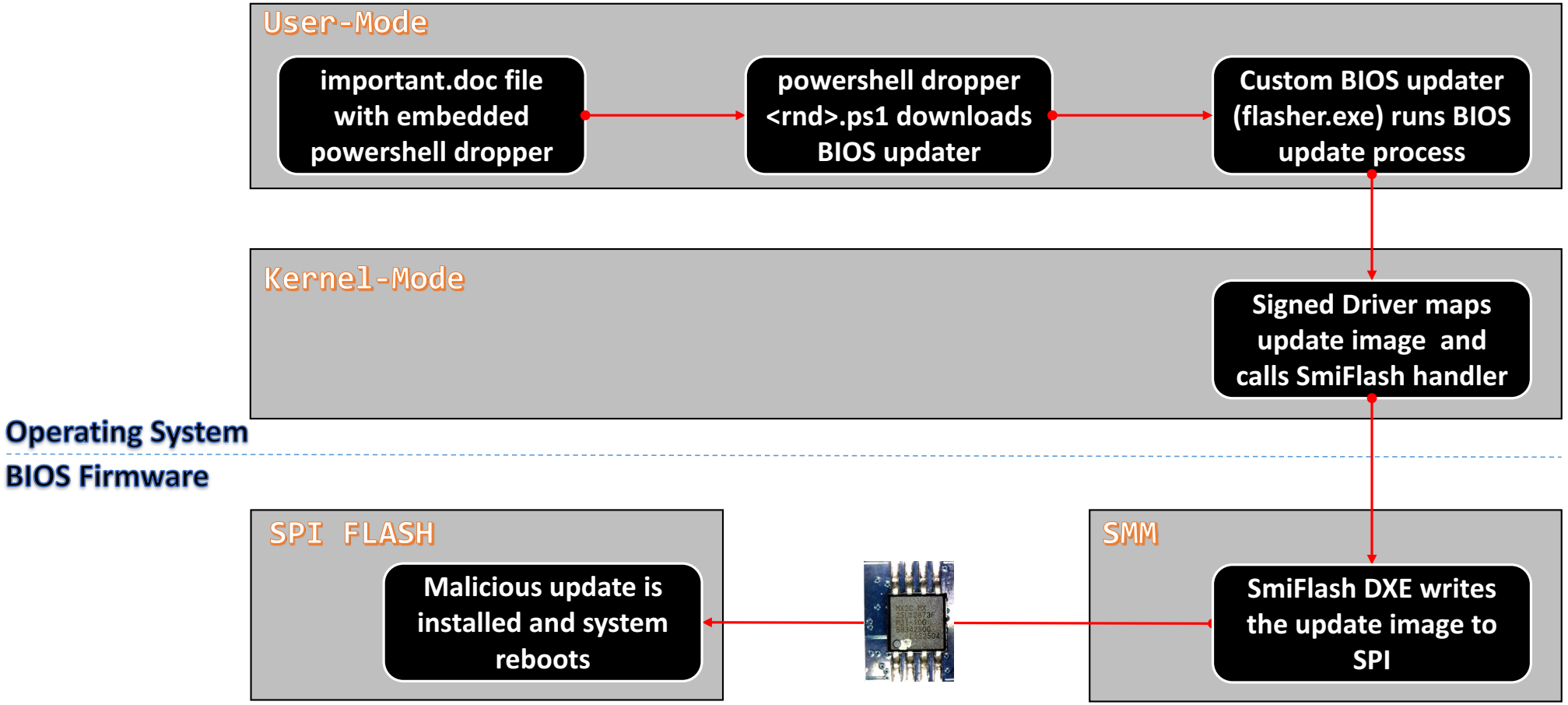
Why so vulnerable?

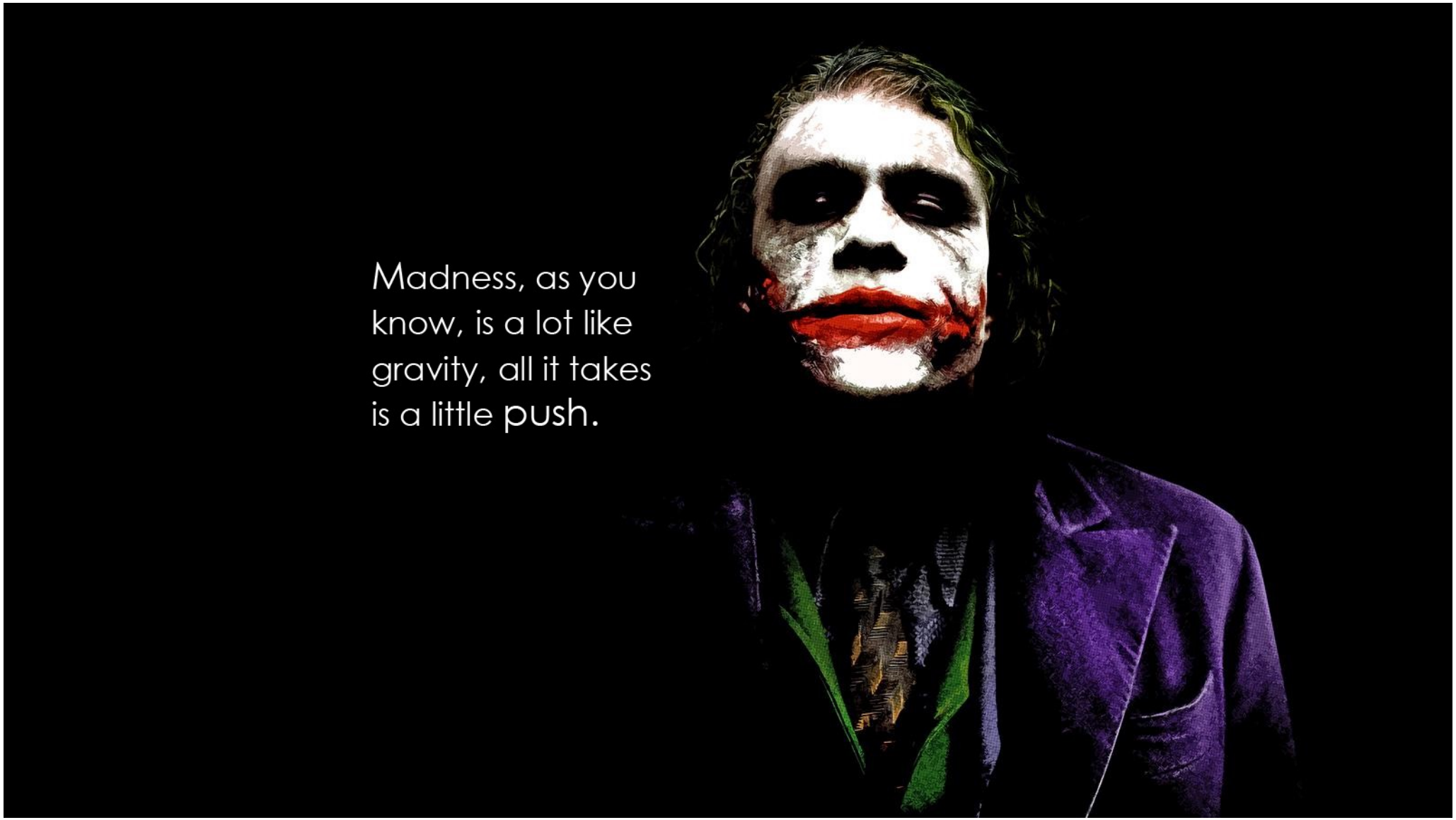
- **BIOS LOCK (BLE) not enabled**
  (CLVA-2016-12-001/CVE-2017-3197)
  - ✓ Attacker is able to modify BIOSWE bit
  - ✓ Attacker can arbitrary write to SPI flash from OS
- **FW update process don't verify signature**
  - ✓ Attacker is able to abuse BIOS updater with signed driver
- **SmiFlash Handler multiple vulns**
  (CLVA-2016-12-002/CVE-2017-3198)
  - ✓ Attacker can elevate privileges to SMM (ring -2)

CLVA-2016-12-001

```
[x][ ================================================================
[x][ Module: BIOS Interface Lock (including Top Swap Mode)
[x][ ================================================================
[*] BiosInterfaceLockDown (BILD) control = 1
[*] BIOS Top Swap mode is disabled (TSS = 0)
[*] RTC TopSwap control (TS) = 0
[+] PASSED: BIOS Interface is locked (including Top Swap Mode)

[*] running module: chipsec.modules.common.bios_wp
[*] Module path: c:\Chipsec\chipsec\modules\common\bios_wp.pyc
[x][ ================================================================
[x][ Module: BIOS Region Write Protection
[x][ ================================================================
[*] BC = 0x08 << BIOS Control (b:d.f 00:31.0 + 0xDC)
    [00] BIOSWE           = 0 << BIOS Write Enable
    [01] BLE              = 0 << BIOS Lock Enable
    [02] SRC              = 2 << SPI Read Configuration
    [04] TSS              = 0 << Top Swap Status
    [05] SMM_BWP          = 0 << SMM BIOS Write Protection
[-] BIOS region write protection is disabled!

[*] BIOS Region: Base = 0x00A00000, Limit = 0x00FFFFFF
SPI Protected Ranges
------------------------------------------------------------
PRx (offset) | Value    | Base     | Limit    | WP? | RP?

PR0 (74)     | 00000000 | 00000000 | 00000000 | 0   | 0
PR1 (78)     | 00000000 | 00000000 | 00000000 | 0   | 0
PR2 (7C)     | 00000000 | 00000000 | 00000000 | 0   | 0
PR3 (80)     | 00000000 | 00000000 | 00000000 | 0   | 0
PR4 (84)     | 00000000 | 00000000 | 00000000 | 0   | 0

[!] None of the SPI protected ranges write-protect BIOS region
```

Remote Installation Persistent UEFI Ransomware




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

black hat ASIA 2017

DEMO TIME

```
[*] running module: chipsec.modules.common.bios_wp
[*] Module path: c:\Chipsec\chipsec\modules\common\bios_wp.pyc
[x][ =============================================================
[x][ Module: BIOS Region Write Protection
[x][ =============================================================
[*] BC = 0x2A << BIOS Control (b:d.f 00:31.0 + 0xDC)
    [00] BIOSWE           = 0 << BIOS Write Enable
    [01] BLE              = 1 << BIOS Lock Enable
    [02] SRC              = 2 << SPI Read Configuration
    [04] TSS              = 0 << Top Swap Status
    [05] SMM_BWP          = 1 << SMM BIOS Write Protection
[+] BIOS region write protection is enabled (writes restricted to SMM)

[*] BIOS Region: Base = 0x00A00000, Limit = 0x00FFFFFF
SPI Protected Ranges
------------------------------------------------------------
PRx (offset) | Value    | Base     | Limit    | WP? | RP?
------------------------------------------------------------
PR0 (74)     | 00000000 | 00000000 | 00000000 | 0   | 0
PR1 (78)     | 00000000 | 00000000 | 00000000 | 0   | 0
PR2 (7C)     | 00000000 | 00000000 | 00000000 | 0   | 0
PR3 (80)     | 00000000 | 00000000 | 00000000 | 0   | 0
PR4 (84)     | 00000000 | 00000000 | 00000000 | 0   | 0

[!] None of the SPI protected ranges write-protect BIOS region

[+] PASSED: BIOS is write protected
```

How does attack work without physical access?

- ✓ The BIOS update is not authenticated (inside BIOS)

  → can accept malicious BIOS image

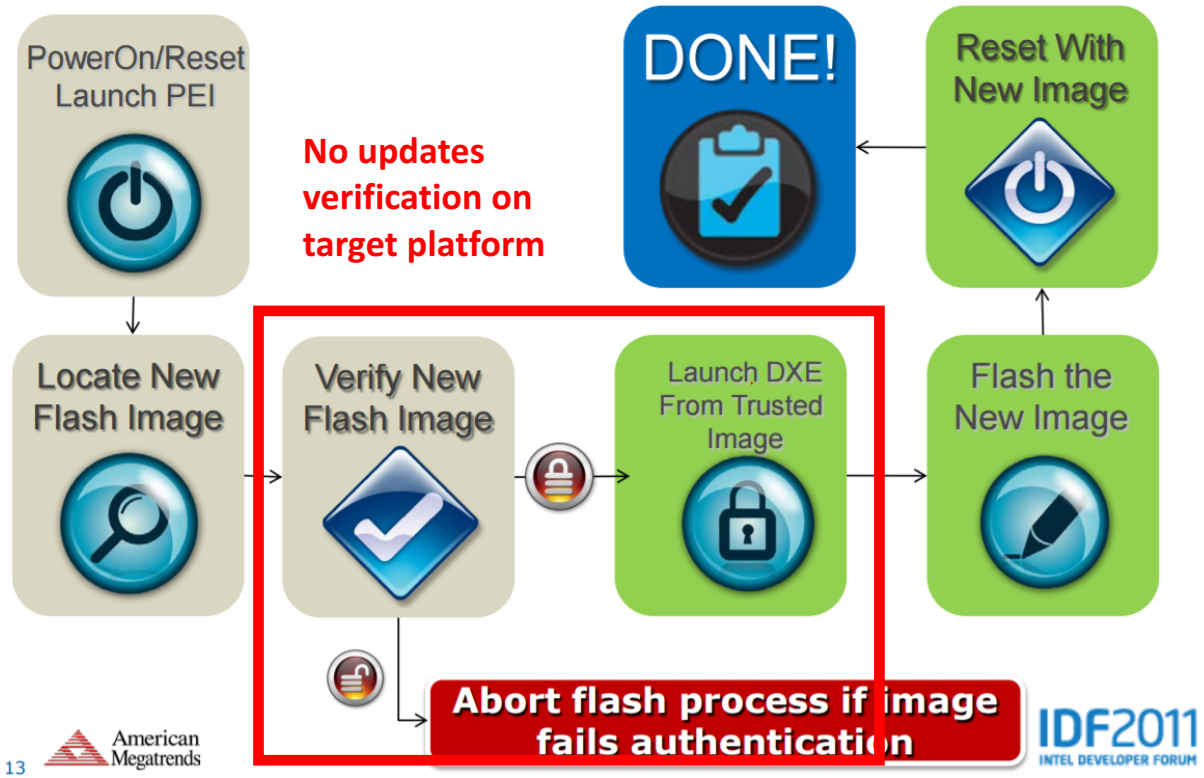
- ✓ The BIOS update software (flasher.exe) does not authenticate updates

  → can install malicious update

- ✓ The BIOS update software uses digitally signed kernel mode driver with unverified caller

  → can deliver malicious update

BIOS Update Process Issues

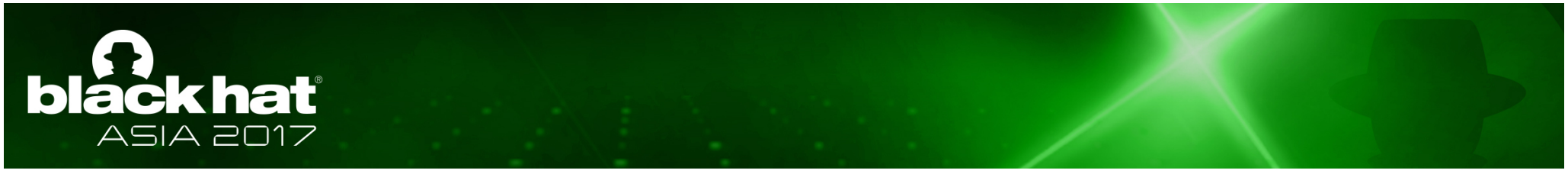


https://firmware.intel.com/sites/default/files/SF11_EFIS002_100.pdf

Gigabyte SMIFlashHandler Issue (CLVA-2016-12-002)

- SMIFLASH HANDLERS (SMiFlash.efi)
  - ✓ ENABLE 0x20
  - ✓ READ 0x21
  - ✓ ERASE 0x22
  - ✓ WRITE 0x23
  - ✓ DISABLE 0x24
  - ✓ GET_INFO 0x25

- No checks for the input pointers
  SmmIsBufferOutsideSmmValid()

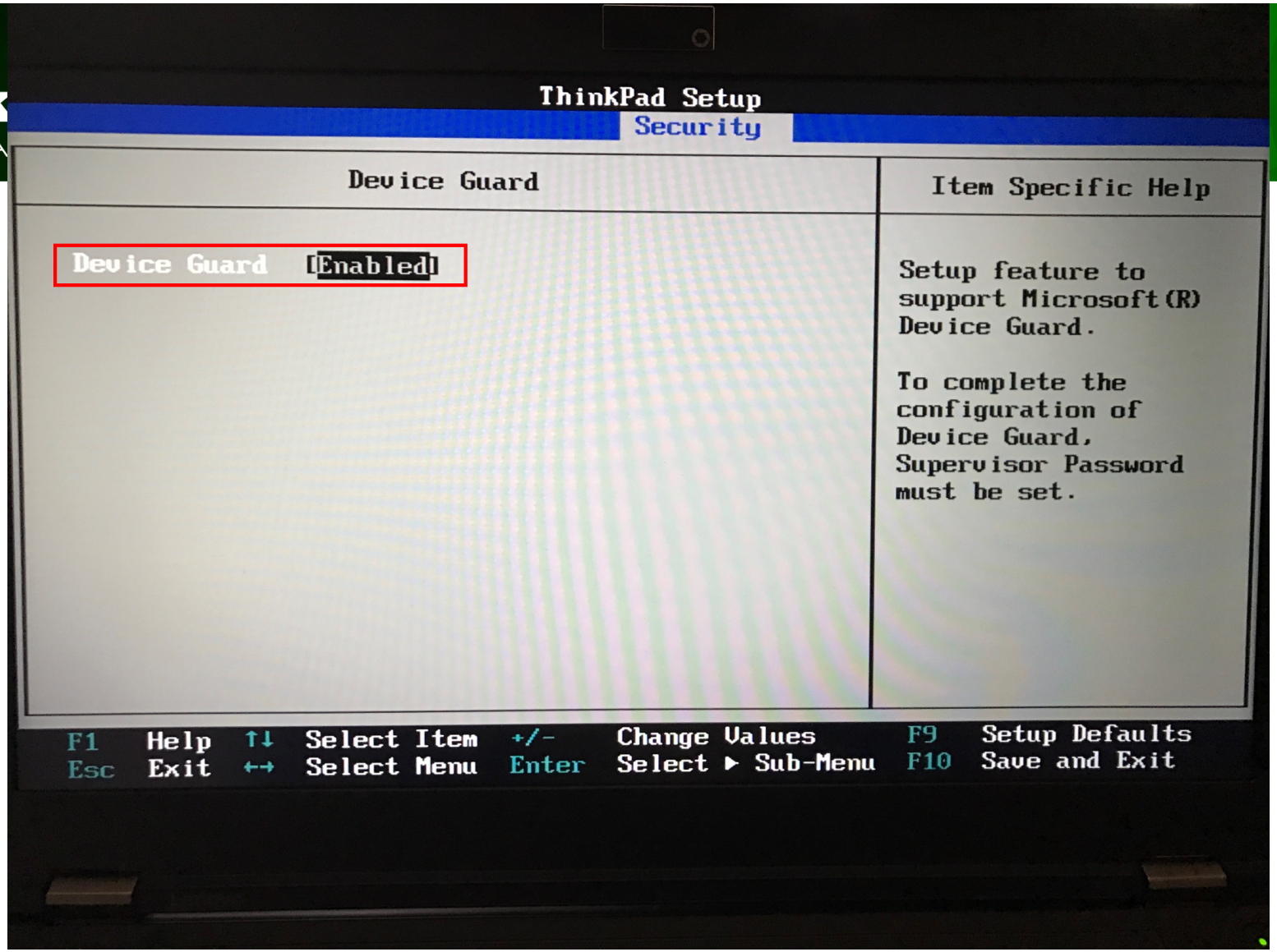


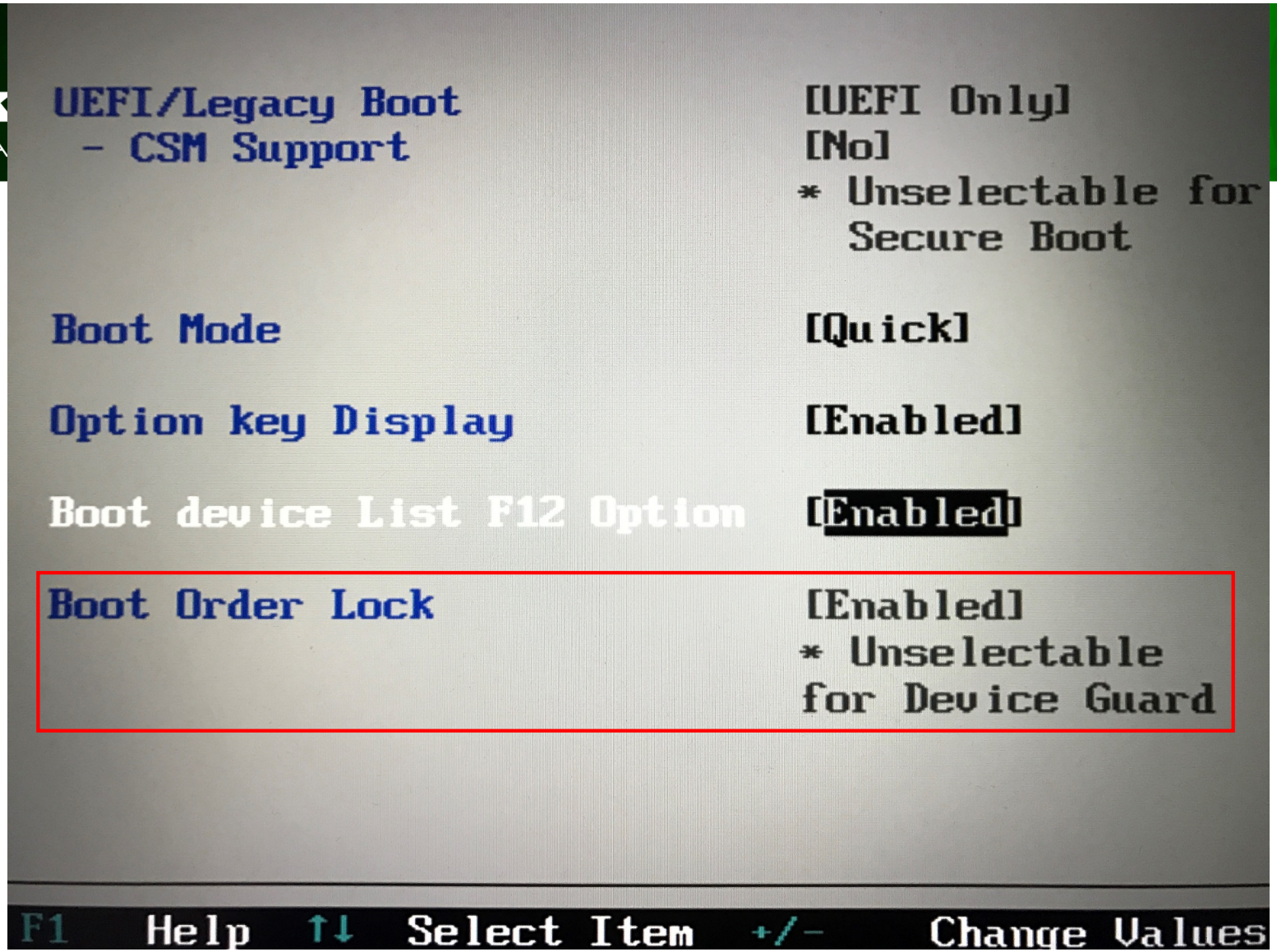
MS Device Guard bypass from UEFI (CVE-2016-8222)

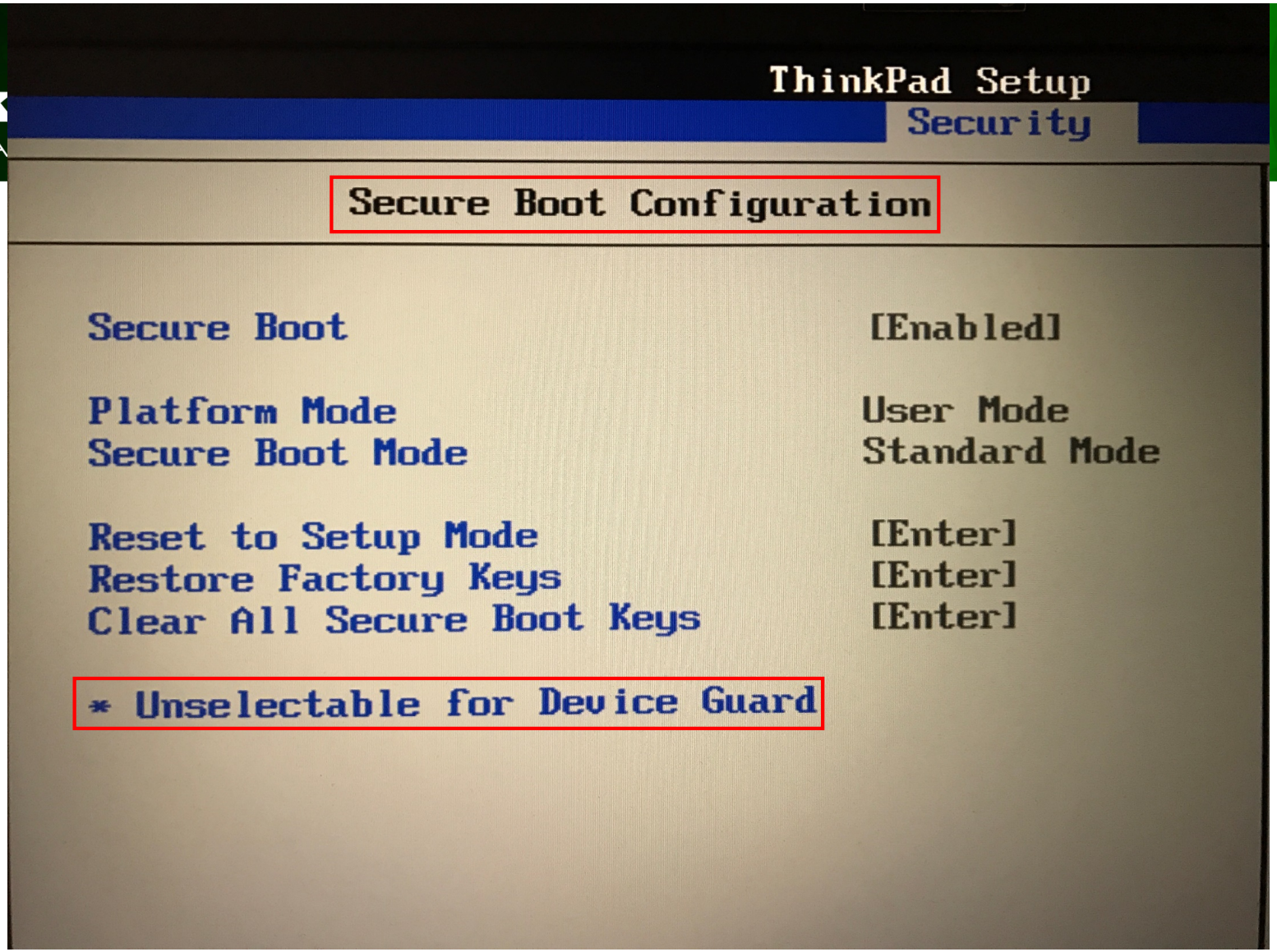
CVE-2016-8222 (LEN-8327): Device Guard protection bypass

"A vulnerability has been identified in a signed kernel driver for the BIOS of some ThinkPad systems that can allow an attacker with Windows administrator-level privileges to call System Management Mode (SMM) services. This could lead to a denial of service attack or allow certain BIOS variables or settings to be altered (such as boot sequence). The setting or changing of BIOS passwords is not affected by this vulnerability."

https://support.lenovo.com/us/ru/solutions/len_8327


ThinkPad Setup
Security
Device Guard
Item Specific Help
Device Guard [Enabled]
Setup feature to support Microsoft(R) Device Guard.
To complete the configuration of Device Guard, Supervisor Password must be set.
F1 Help ↑↓ Select Item +/- Change Values F9 Setup Defaults
Esc Exit ↔ Select Menu Enter Select ▶ Sub-Menu F10 Save and Exit


UEFI/Legacy Boot
[UEFI Only]
- CSM Support
[No]
* Unselectable for
Secure Boot
Boot Mode
[Quick]
Option key Display
[Enabled]
Boot device List F12 Option
[Enabled]
Boot Order Lock
[Enabled]
* Unselectable
for Device Guard
F1 Help ↑↓ Select Item +/- Change Values


ThinkPad Setup
Security
Secure Boot Configuration
Secure Boot
[Enabled]
Platform Mode
User Mode
Secure Boot Mode
Standard Mode
Reset to Setup Mode
[Enter]
Restore Factory Keys
[Enter]
Clear All Secure Boot Keys
[Enter]
* Unselectable for Device Guard

LenovoDeviceGuardDxe

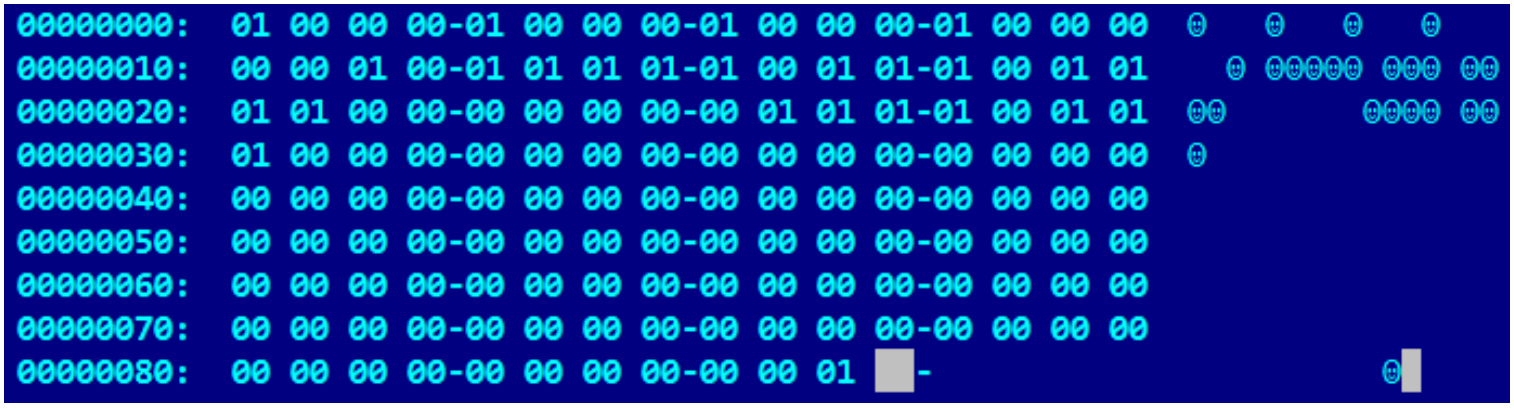
```
__int64 __fastcall SetupDeviceGuardCfg(EFI_CONFIGURATION_TABLE *a1, __int64 Attr)
{
  __int64 v2; // rbx@1
  __int64 _result; // rax@3
  char Value; // [rsp+30h] [rbp-98h]@1
  char v6; // [rsp+3Ch] [rbp-8Ch]@2
  char v7; // [rsp+6Ah] [rbp-5Eh]@1
  __int64 Size; // [rsp+D0h] [rbp+8h]@1
  __int64 Attrib; // [rsp+D8h] [rbp+10h]@1

  Attrib = Attr;
  Size = 139i64;
  (RuntimeServices->GetVariable)(L"LenovoSecurityConfig", &LenovoSecurityConfigGuid, &Attrib, &Size, &Value);
  v2 = 0i64;
  if ( v7 && v6 == 1 )
  {
    v6 = 0;
    (RuntimeServices->SetVariable)(L"LenovoSecurityConfig", &LenovoSecurityConfigGuid, Attrib, Size, &Value);
    _result = (BootServices->LocateProtocol)(&LenovoSecurityVariableProtocolGuid, 0i64, &LenovoSecurityVariableProtocol);
    if ( _result >= 0 )
    {
      (LenovoSecurityVariableProtocol->CheckLenovoSecurityConfigCrc32)(LenovoSecurityVariableProtocol);
      (RuntimeServices->ResetSystem)(0i64, 0i64, 0i64, 0i64);
    }
    else
    {
      v2 = _result;
    }
  }
  return v2;
}
```

LenovoDeviceGuardDxe

```
__int64 CheckLenovoSecurityConfigCrc32()
{
  __int64 result; // rax@1
  __int64 crc32_byte; // r9@1
  signed int eeprom_offset; // ebx@2
  char *pcrc32; // rdi@2
  int crc32; // [rsp+38h] [rbp+10h]@1

  crc32 = 0;
  result = GetCrc32OfLenovoSecurityConfigVar(&crc32);
  if ( result >= 0 )
  {
    eeprom_offset = 92;
    pcrc32 = &crc32;
    do
    {
      LOBYTE(crc32_byte) = *pcrc32;
      result = EmulatedEepromProtocol(0i64, 87i64, eeprom_offset, crc32_byte);
      if ( result < 0 )
        break;
      ++eeprom_offset;
      ++pcrc32;
    }
    while ( eeprom_offset - 92 < 4 );
  }
  return result;
}
```

LenovoSecurityConfig (T-540p without DXE)

- VARIBLE NAME: LenovoSecurityConfig
- GUID: A2C1808F-0D4F-4CC9-A619-D1E641D39D49
- Attributes:
  - ✓ **NV** – stored in non-volatile memory (NVRAM)
  - ✓ **BS** – BOOTSERVICE ACCESS
  - ✓ **RT** – RUNTIME ACCESS
  - ✓ **AWS** – AUTHENTICATED WRITE ACCESS

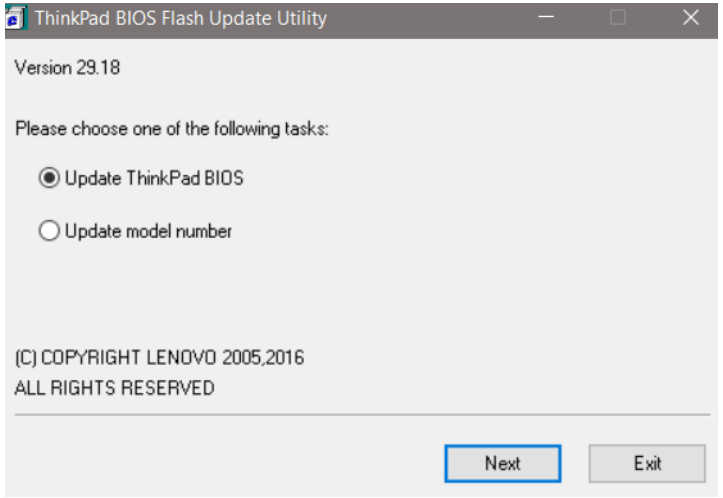


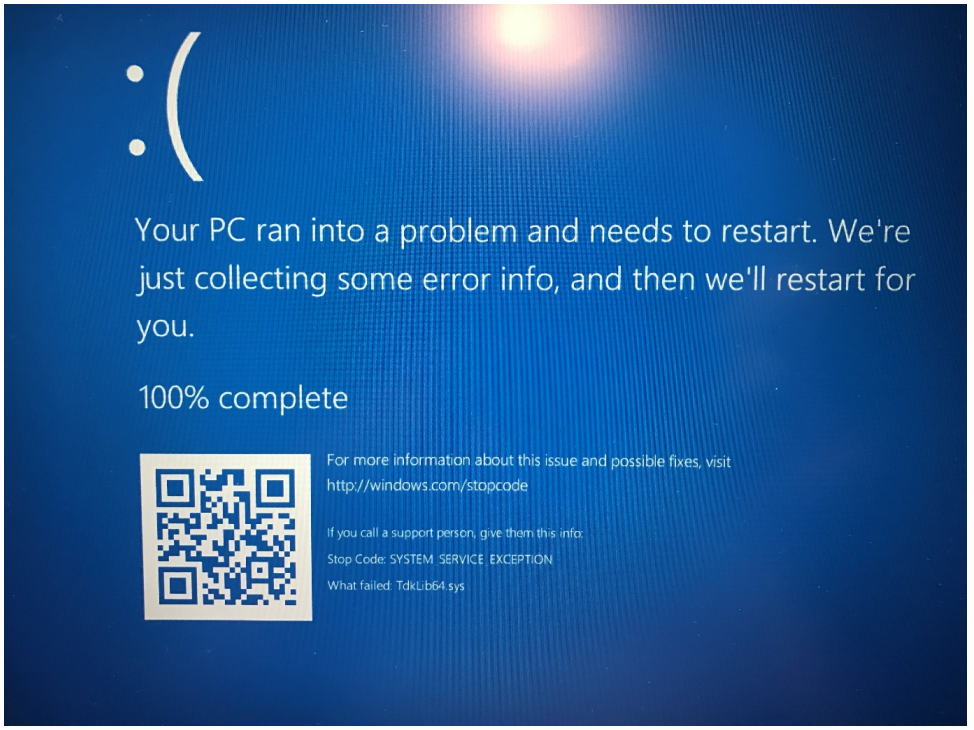
LenovoSecurityConfig (P50 with DXE)

- VARIBLE NAME: LenovoSecurityConfig
- GUID: 2A4DC6B7-41F5-45DD-B46F-2DD334C1CF65
- Attributes: **NV+BS+RT+AWS**


LET'S
NOT BLOW
THIS OUT OF
PROPORTION

BIOS Update with Device Guard





Forensic Approaches


GOOD
OR
BAD
BIOS

SMM Memory Forensics

- Diff original and infected SMRAM to hunt for malicious SMI
  - ✓ Parse *EFI_SMM_SYSTEM_TABLE* to get SMM memory layout
  - ✓ Find all SMI handlers based on *DATABASE_RECORD* signature
- Also another useful information can be extracted from SMRAM:
  - ✓ List of SMM drivers
  - ✓ List of SMM Protocols
- Kudos to Cr4sh for SMRAM forensic parser

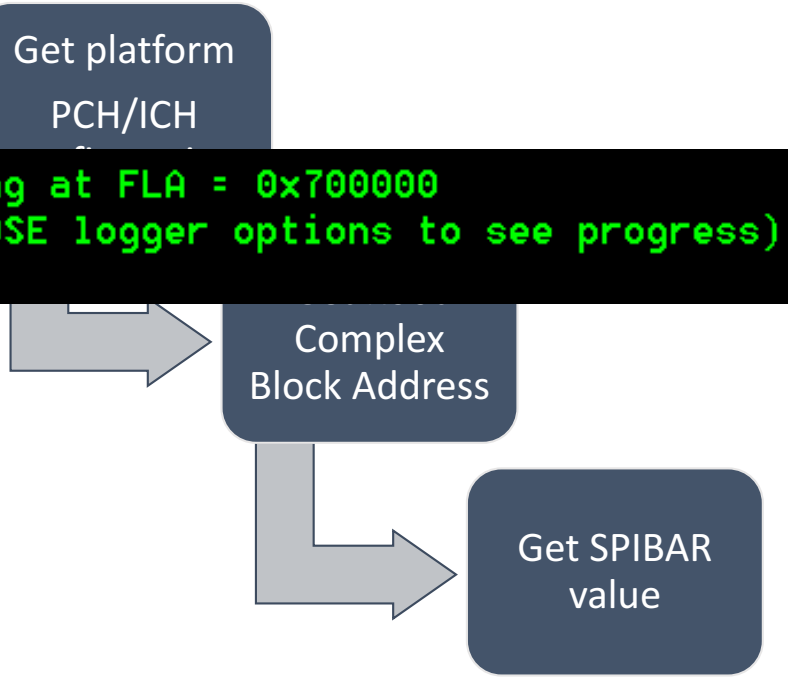
  https://github.com/Cr4sh/smram_parse/blob/master/smram_parse.py

How to dump SPI Flash?

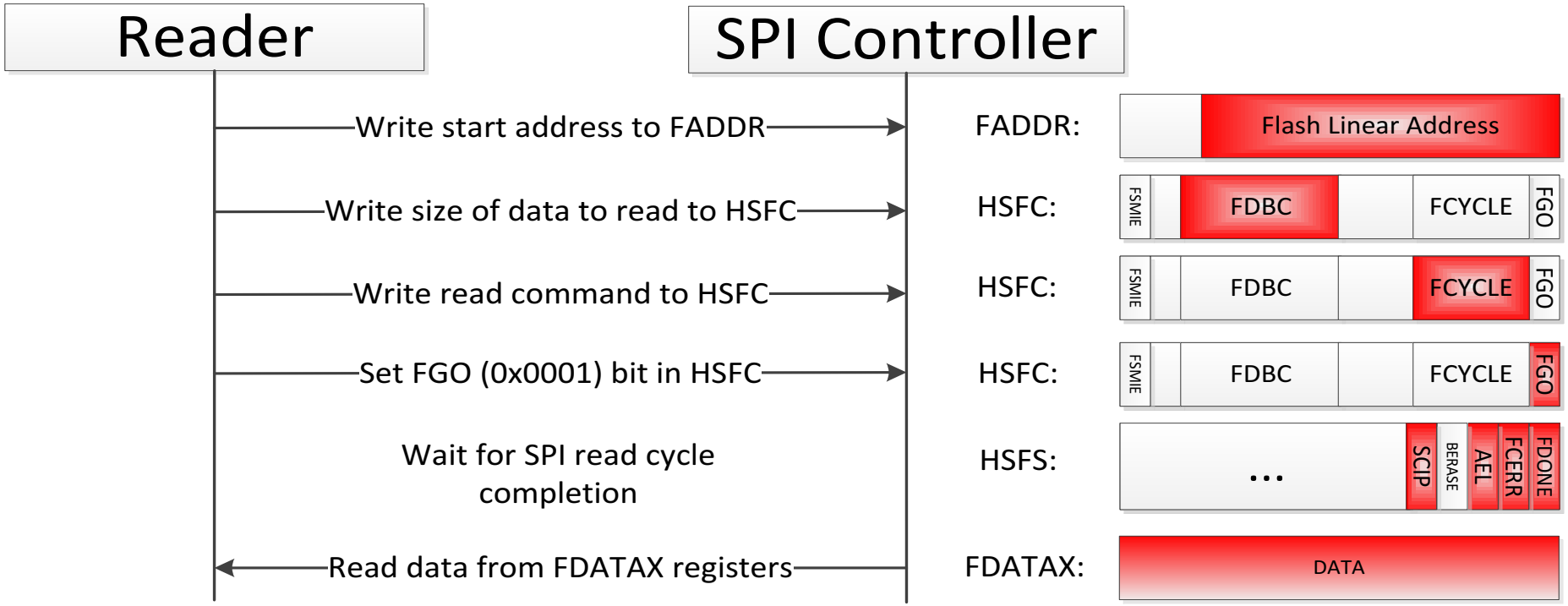
SPI Flash – Reading from OS

- 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

```
[CHIPSEC] reading 0x100000 bytes from SPI Flash starting at FLA = 0x700000
[CHIPSEC] it may take a few minutes (use DEBUG or VERBOSE logger options to see progress)
ERROR: HSFS.FDV is 0, hardware sequencing is disabled
```



SPI Flash – Reading from OS

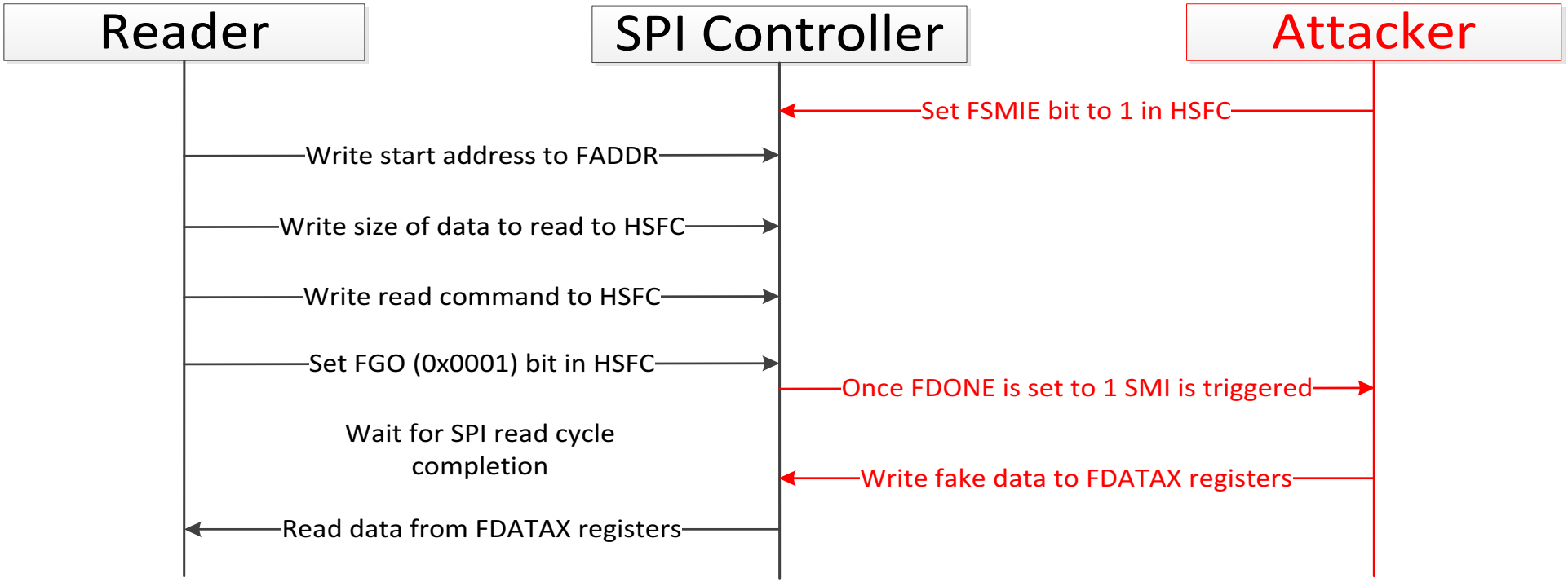


SPI Flash – Anti-Forensics

HSFC:

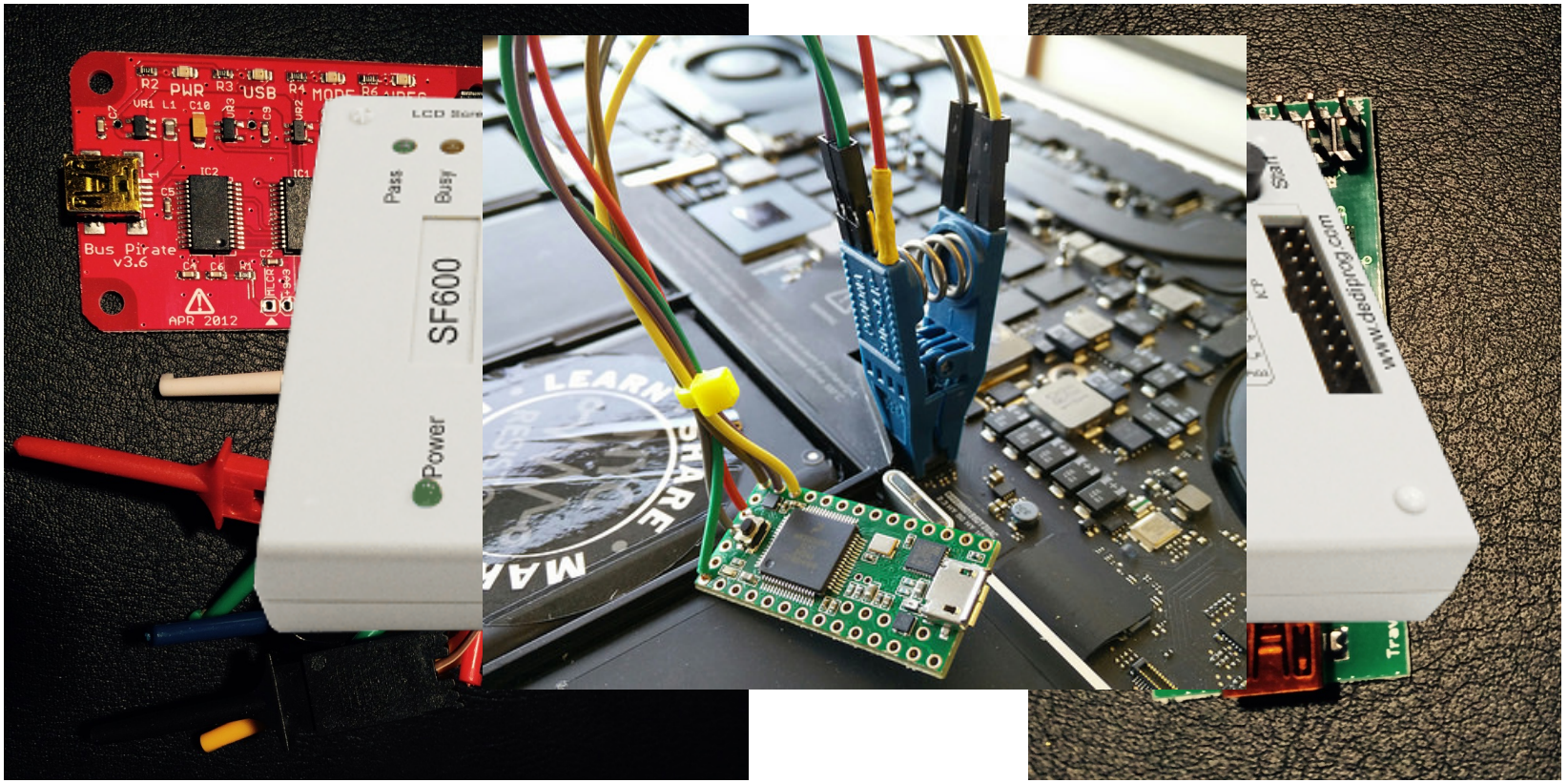
Flash SPI SMI# Enable (FSMIE) — R/W. When set to 1, the SPI asserts an SMI# request whenever the Flash Cycle Done (FDONE) bit is 1.

SPI Flash – Anti-Forensics



black hat ASIA 2017

How to dump BIOS?



Intel Virtual Platform

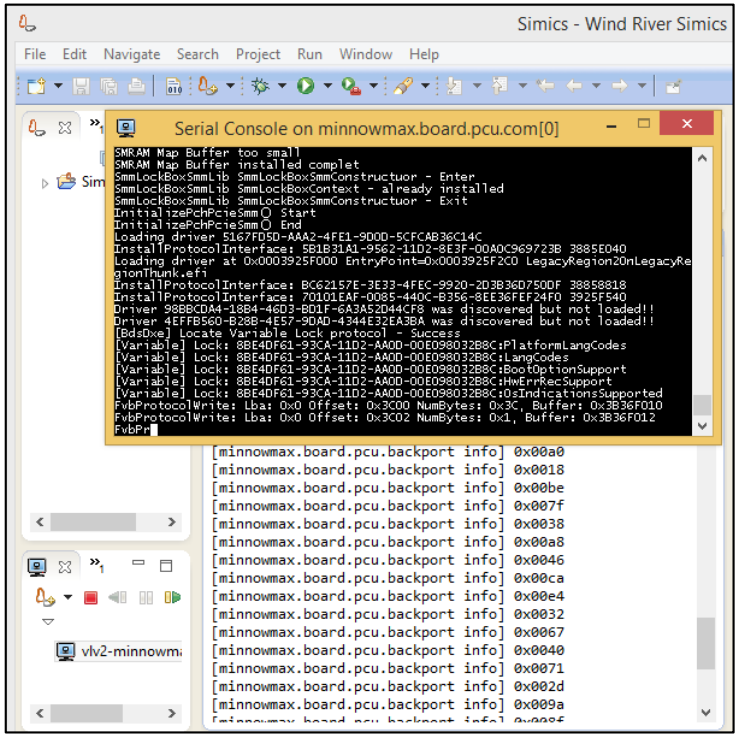
black hat ASIA 2017

WIND

AN INTEL COMPANY

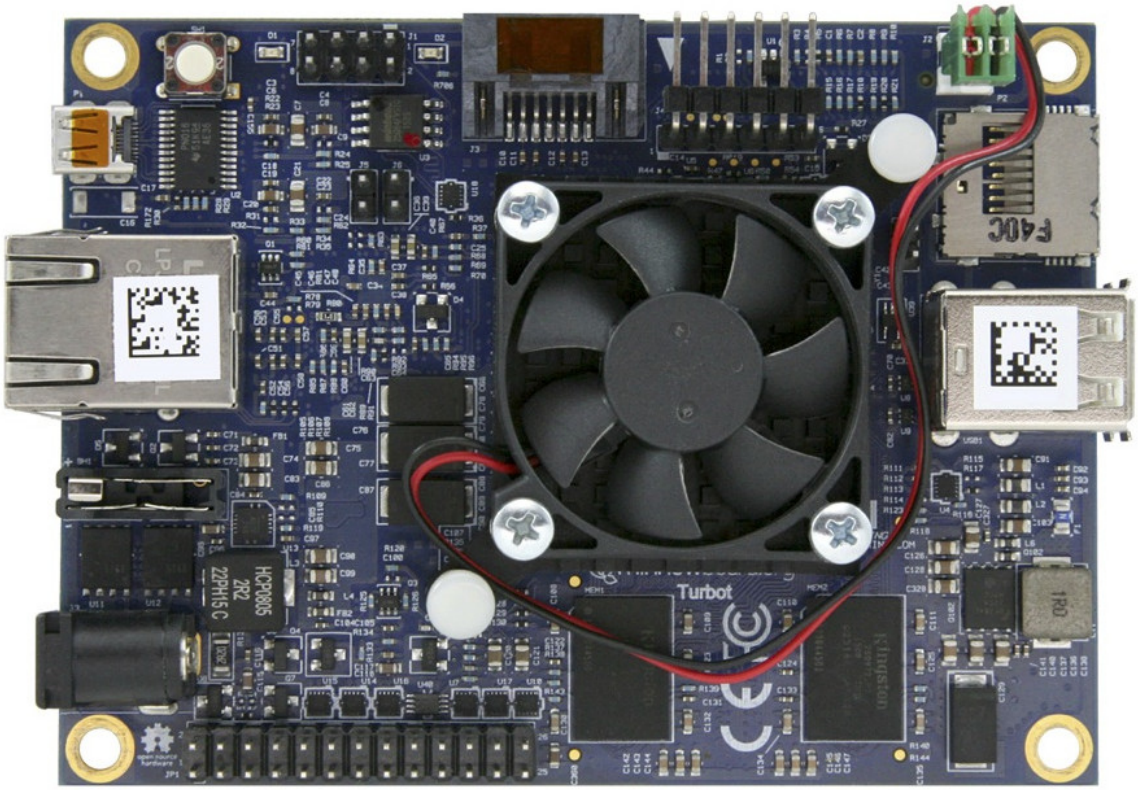
WIND RIVER SIMICS

Simulate Anything, Chip to System



- 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/


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

black hat ASIA 2017

Intel XDP Hardware Debuggers

A few words about UEFI Firmware Mitigations


LETS PUT
A SMILE
ON THAT FACE!

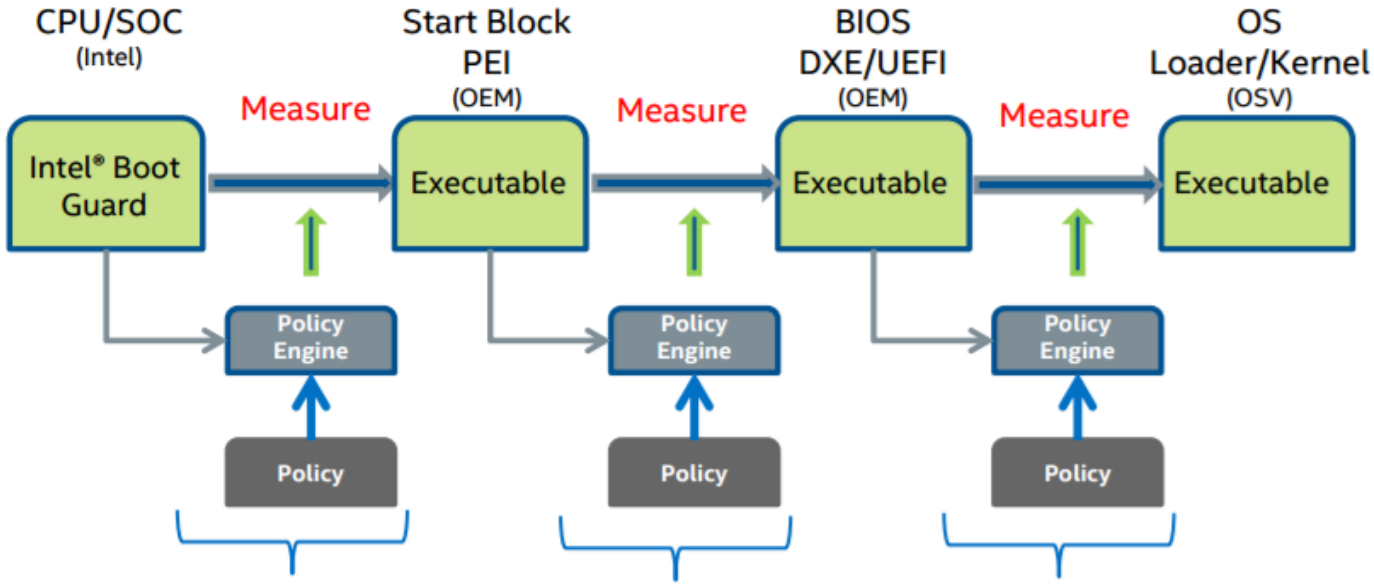
Intel Boot Guard Technology

- Intel Boot Guard – authenticated code module ACM-based secure boot (verified boot) that's verifies a known and trusted BIOS is booting the platform

- Protect Secure Boot Root of Trust from firmware-based attacks

CPU reset -> bootrom -> ACM -> initial boot block -> BIOS -> boot loader -> OS

Intel Boot Guard Technology



Intel® Device Protection Technology with Boot Guard

http://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/4th-gen-core-family-mobile-brief.pdf

OEM PI Verification Using PI Signed Firmware Volumes

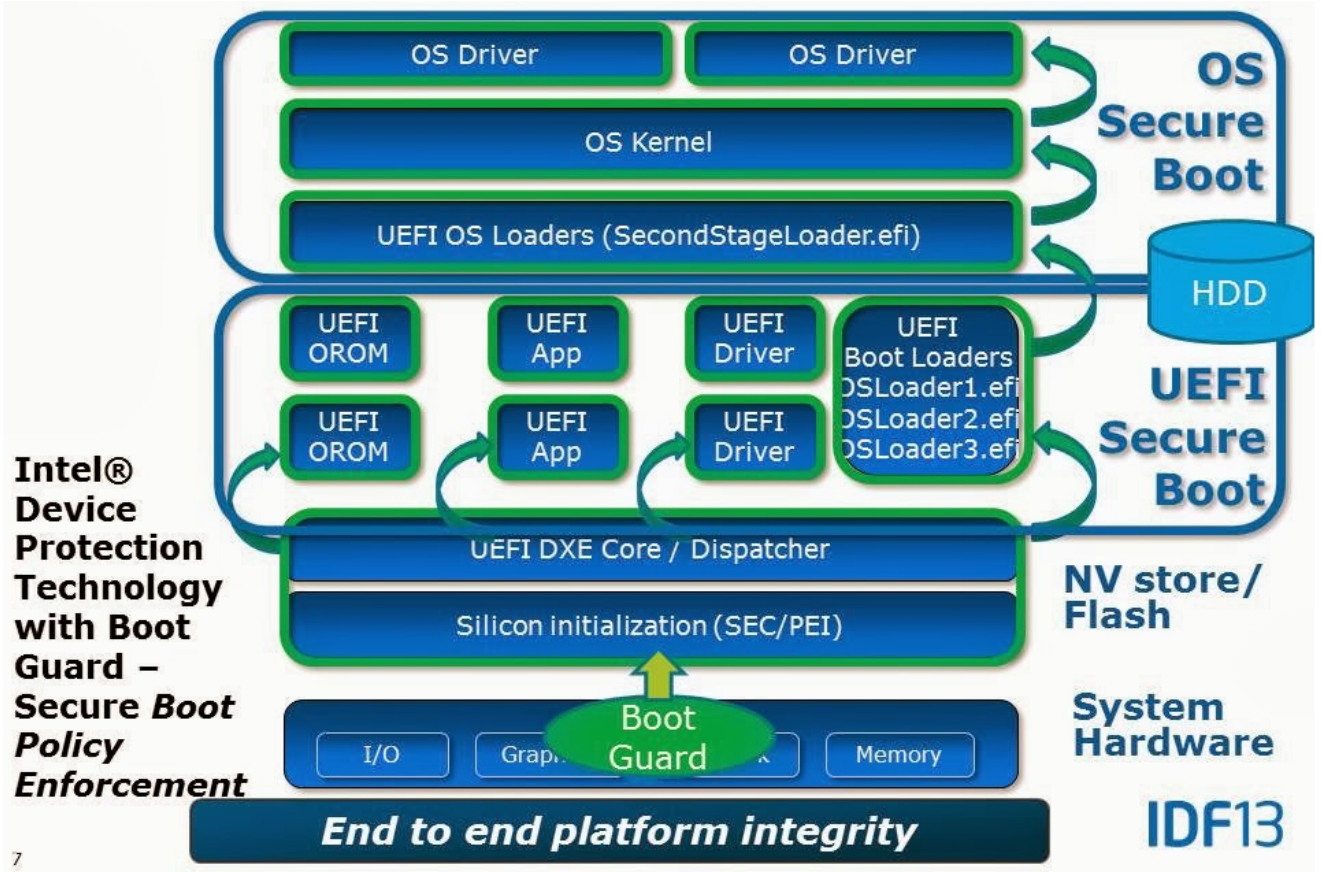
Vol 3, section 3.2.1.1 of PI 1.3 Specification

OEM UEFI 2.4 Secure Boot

Chapter 27.2 of The UEFI 2.4 Specification

https://firmware.intel.com/sites/default/files/STTS003%20-%20SF15_STTS003_100f.pdf

Intel Boot Guard Technology

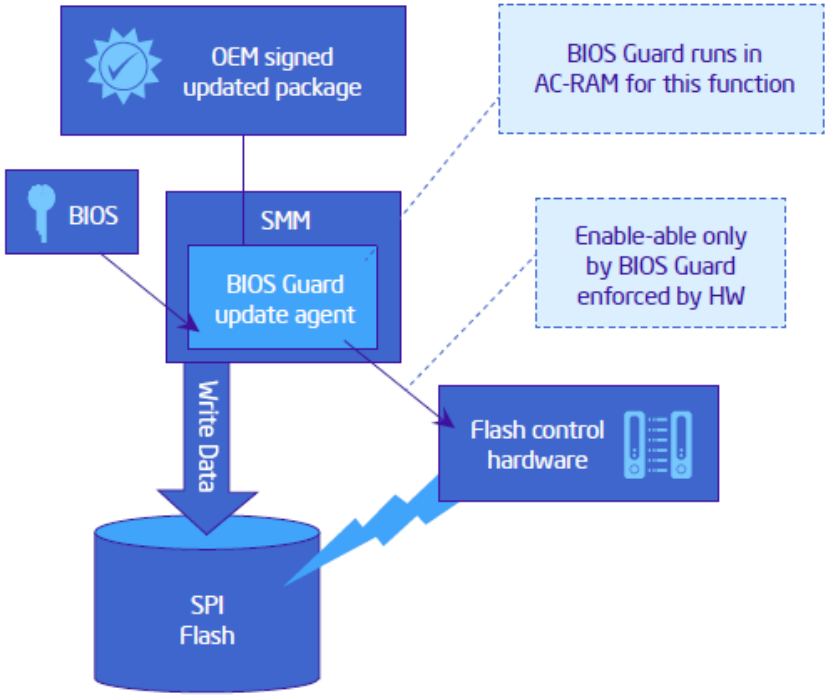


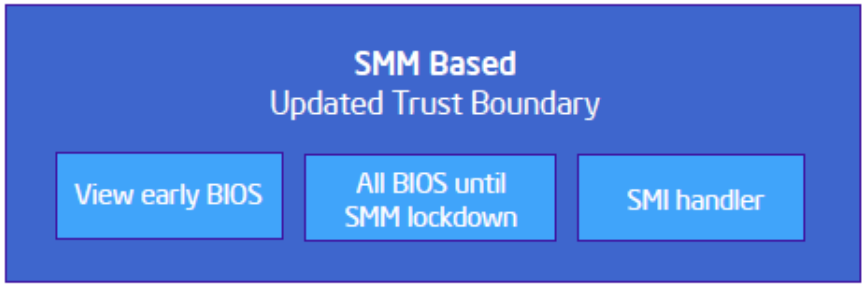
http://vzimmer.blogspot.com/2013/09/where-do-i-sign-up.html

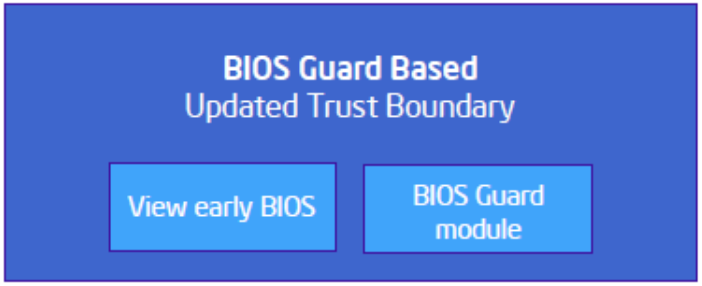
Intel BIOS Guard Technology

- BIOS Guard – hardware-assisted authentication and protection against BIOS recovery attacks
- BIOS Guard can reduces SMI handler attack surface because of one signed and authenticated code module (ACM)
- BIOS Guard module is authenticated code module (ACM) executing in internal processor (isolated CPU) before letting the machine boot
- With active BIOS Guard only guarded module is able to modify SPI flash memory
- BIOS Guard can be useful protection from persistent rootkit infection

Intel BIOS Guard Technology







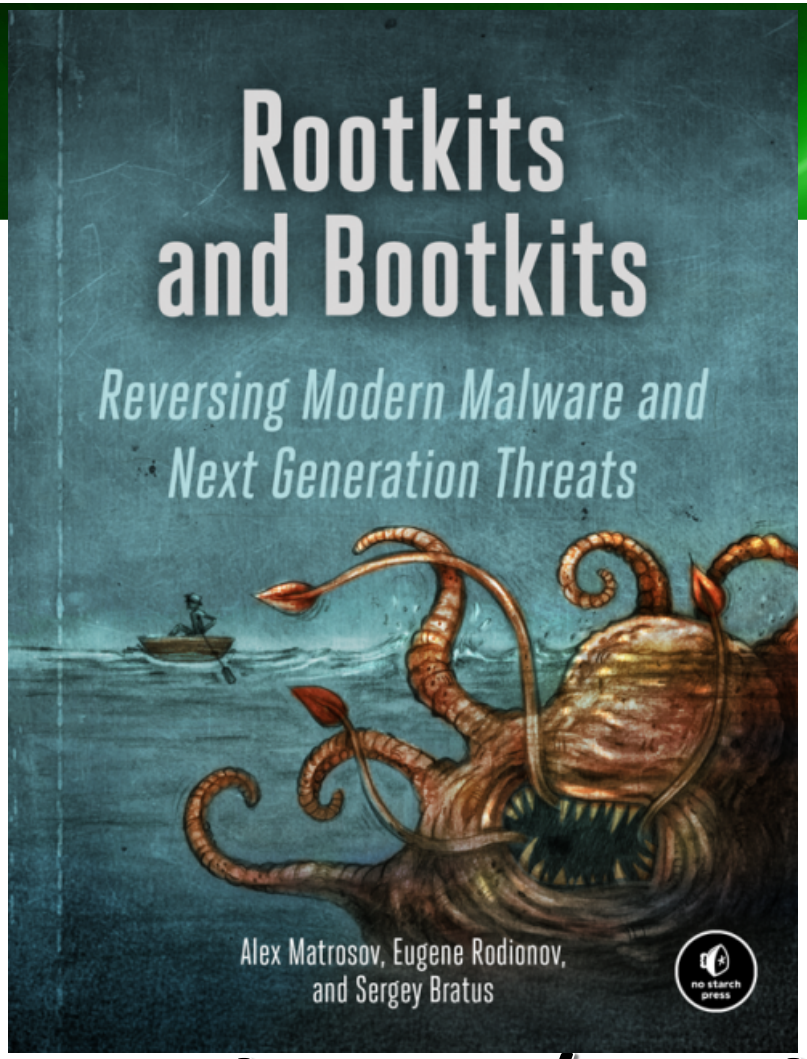
http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/security-technologies-4th-gen-core-retail-paper.pdf

Windows SMM Security Mitigation Table (WSMT)

```
/** @file
  Defines Windows SMM Security Mitigation Table
  @ https://msdn.microsoft.com/windows/hardware/drivers/bringup/acpi-system-description-tables#wsmt

  Copyright (c) 2016, Intel Corporation. All rights reserved.<BR>
  This program and the accompanying materials
  are licensed and made available under the terms and conditions of the BSD License
  which accompanies this distribution.  The full text of the license may be found at
  http://opensource.org/licenses/bsd-license.php

  THE PROGRAM IS DISTRIBUTED UNDER THE BSD LICENSE ON AN "AS IS" BASIS,
  WITHOUT WARRANTIES OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

**/


#ifndef _WINDOWS_SMM_SECURITY_MITIGATION_TABLE_H_
#define _WINDOWS_SMM_SECURITY_MITIGATION_TABLE_H_

#include <IndustryStandard/Acpi.h>

#define EFI_ACPI_WINDOWS_SMM_SECURITY_MITIGATION_TABLE_SIGNATURE  SIGNATURE_32('W', 'S', 'M', 'T')

#pragma pack(1)

#define EFI_WSMT_TABLE_REVISION  1

typedef struct {
  EFI_ACPI_DESCRIPTION_HEADER Header;
  UINT32                      ProtectionFlags;
} EFI_ACPI_WSMT_TABLE;

#define EFI_WSMT_PROTECTION_FLAGS_FIXED_COMM_BUFFERS                  0x1
#define EFI_WSMT_PROTECTION_FLAGS_COMM_BUFFER_NESTED_PTR_PROTECTION   0x2
#define EFI_WSMT_PROTECTION_FLAGS_SYSTEM_RESOURCE_PROTECTION          0x4

#pragma pack()

#endif
```

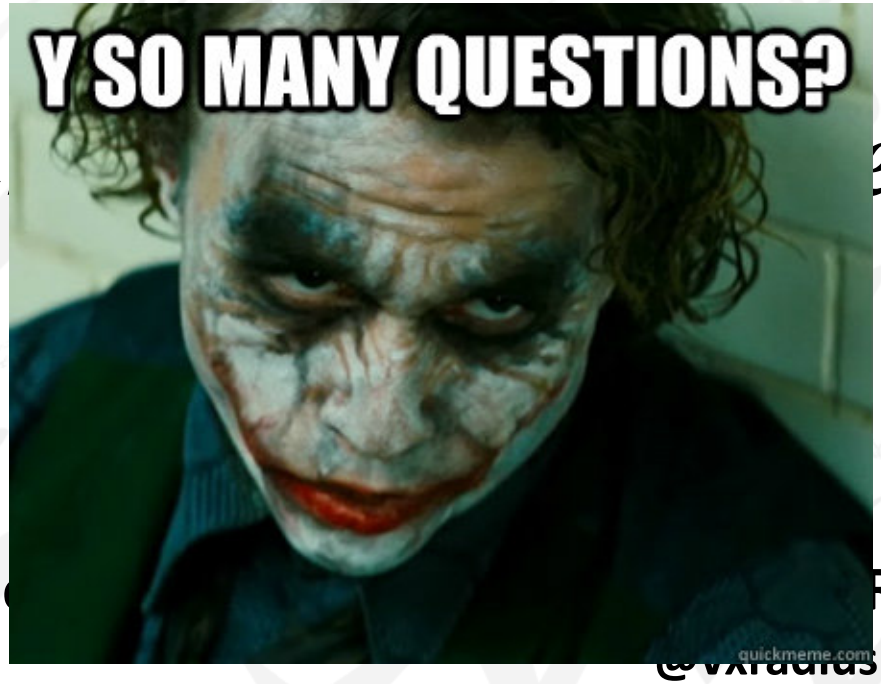
https://msdn.microsoft.com/en-us/windows/hardware/drivers/bringup/acpi-system-description-tables#wsmt

black hat ASIA 2017



nostarch.com/rootkits

Than ention!



Alex Matros Rodionov
@matrosov
@vxradius