Security of BIOS/UEFI System Firmware from Attacker and Defender Perspectives

7. Hands-On System Firmware Forensics

Yuriy Bulygin *
Alex Bazhaniuk *
Andrew Furtak *
John Loucaides **

* Advanced Threat Research, McAfee
** Intel

License

Training materials are shared under Creative Commons “Attribution” license CC BY 4.0

Provide the following attribution:

Derived from “Security of BIOS/UEFI System Firmware from Attacker and Defender Perspective” training by Yuriy Bulygin, Alex Bazhaniuk, Andrew Furtak and John Loucaides available at https://github.com/advanced-threat-research/firmware-security-training

Section 7. Hands-on System Firmware Forensics

7.1 Live Forensic / Incident Response

Important: software based forensics is not reliable to detect firmware compromise. Firmware rootkits can interfere with software based forensics.

Live System Forensics

To perform system firmware forensics, the following components can be extracted & analyzed:

1. Layout and entire contents of SPI Flash memory
2. BIOS/UEFI firmware including EFI binaries and NVRAM
3. Runtime or Boot UEFI Variables (non-volatile and volatile)
4. UEFI Secure Boot certificates (PK, KEK, db/dbx ..)
5. UEFI system and configuration tables (Runtime, Boot and DXE services)
6. UEFI S3 resume boot script table
7. PCIe option (expansion) ROMs

Live System Forensics

8. Settings stored in RTC-backed CMOS memory
9. ACPI tables
10. SMBIOS table
11. HW protection settings (e.g. SPI W/P)
12. System firmware protection settings (Secure Boot, etc.)
13. MBR/VBR or UEFI GUID Partition Table (GPT)
14. Files on EFI system partition (boot loaders)
15. Contents of TPM Platform Configuration Registers (PCR)
16. Firmware images from other components such as HDD/SSD, NIC, Embedded Controller, etc.

Live System Forensics

SPI Flash Memory Contents

```
chipsec_util spi info
chipsec_util spi dump SPI.bin
chipsec_util spi read <start> <size> BIOS.bin
```

Understanding Layout of SPI Flash Memory

```
# chipsec_util.py spi info

. . .
================================================================
SPI Flash Map
----------------------------------------------------------------

BIOS Flash Primary Region
----------------------------------------------------------------
BFPREG = 0BFF0500:
  Base  : 00500000
  Limit : 00BFF000
  Shadowed BIOS Select: 0

----------------------------------------------------------------
Flash Region             | FREGx Reg | Base     | Limit
----------------------------------------------------------------
0 Flash Descriptor       | 00000000  | 00000000 | 00000FFF
1 BIOS                   | 0BFF0500  | 00500000 | 00BFFFFF
2 Intel ME               | 04FF0003  | 00003000 | 004FFFFF
3 GBe                    | 00020001  | 00001000 | 00002FFF
4 Platform Data          | 00001FFF  | 01FFF000 | 00000FFF
```

List of Flash Regions

BIOS region

Extracting Contents of SPI Flash Memory...

Read/Write/Erase SPI flash memory

CHIPSEC SPI command line interface

```
# chipsec_util.py spi

chipsec_util spi info|dump|read|write|erase [flash_address] [length] [file]
Examples:
  chipsec_util spi info
  chipsec_util spi dump rom.bin
  chipsec_util spi read 0x700000 0x100000 bios.bin
  chipsec_util spi write 0x0 flash_descriptor.bin
```

Full dump of SPI flash

```
# chipsec_util.py spi dump spi.dump.bin

# dir spi.dump.bin
Directory of C:\Users\user\Desktop\chipsec\tool

03/05/2015  11:31 PM         8,388,608 spi.dump.bin
               1 File(s)      8,388,608 bytes
               0 Dir(s)  209,732,562,944 bytes free
```

SPI flash image size: 8M

Live System Forensics

BIOS and UEFI System Firmware

```
chipsec_util uefi var-list
chipsec_util uefi var-find fTA
chipsec_util uefi var-read db D719B2CB.. db.bin
chipsec_util uefi tables
chipsec_util uefi s3bootscript
chipsec_util acpi list
chipsec_util acpi table FADT
```

Extracting Persistent EFI Configuration…

```
# chipsec_util.py uefi var-list
```

Name	Ext	Size
AcpiGlobalVariable_C020489E-6DB2-4EF2-9AA5-CA06FC11D36A_NV+BS+RT_1	bin	8
AMITSESetup_C811FA38-42C8-4579-A9BB-60E94EDDFB34_NV+BS+RT_0	bin	81
Boot0000_8BE4DF61-93CA-11D2-AA0D-00E098032B8C_NV+BS+RT_0	bin	136
Boot0001_8BE4DF61-93CA-11D2-AA0D-00E098032B8C_NV+BS+RT_0	bin	300
BootCurrent_8BE4DF61-93CA-11D2-AA0D-00E098032B8C_BS+RT_0	bin	2
BootOptionSupport_8BE4DF61-93CA-11D2-AA0D-00E098032B8C_BS+RT_0	bin	4
BootOrder_8BE4DF61-93CA-11D2-AA0D-00E098032B8C_NV+BS+RT_0	bin	10
db_D719B2CB-3D3A-4596-A3BC-DAD00E67656F_NV+BS+RT+TBAWS_0	bin	3,143
dbx_D719B2CB-3D3A-4596-A3BC-DAD00E67656F_NV+BS+RT+TBAWS_0	bin	76
DimmSPDdata_A09A3266-0D9D-476A-B8EE-0C226BE16644_NV+BS+RT_0	bin	8
DmiData_70E56C5E-280C-44B0-A497-09681ABC375E_NV+BS+RT_0	bin	397
FastBootOption_B540A530-6978-4DA7-91CB-7207D764D262_NV+BS+RT_0	bin	284
FlashInfoStructure_82FD6BD8-02CE-419D-BEF0-C47C2F123523_NV+BS+RT_0	bin	7
Guid1394_F9861214-9260-47E1-BCBB-52AC033E7ED8_NV+BS+RT_0	bin	8
KEK_8BE4DF61-93CA-11D2-AA0D-00E098032B8C_NV+BS+RT+TBAWS_0	bin	1,560
LastBoot_B540A530-6978-4DA7-91CB-7207D764D262_NV+BS+RT_0	bin	10
LegacyDevOrder_A56074DB-65FE-45F7-BD21-2D2BDD8E9652_NV+BS+RT_0	bin	16
MaintenanceSetup_EC87D643-EBA4-4BB5-A1E5-3F3E36B20DA9_NV+BS+RT_0	bin	410
MEFWVersion_9B875AAC-36EC-4550-A4AE-86C84E96767E_NV+BS+RT_0	bin	20
MemorySize_6F20F7C8-E5EF-4F21-8D19-EDC5F0C496AE_NV+BS+RT_0	bin	8
MemoryTypeInformation_4C19049F-4137-4DD3-9C10-8B97A83FFDFA_NV+BS+RT_0	bin	64
MrcS3Resume_87F22DCB-7304-4105-BB7C-317143CCC23B_NV+BS+RT_0	bin	4,052
NBPlatformData_EC87D643-EBA4-4BB5-A1E5-3F3E36B20DA9_BS+RT_		
OsIndications_8BE4DF61-93CA-11D2-AA0D-00E098032B8C_NV+BS+R		
OsIndicationsSupported_8BE4DF61-93CA-11D2-AA0D-00E098032B8C_		
PasswordInfo_6320A8C8-9C93-4A71-B529-9F79C8761B8D_NV+BS+RT_		
PchS3Peim_E6C2F70A-B604-4877-85BA-DEEC89E117EB_BS+RT_0		
PK_8BE4DF61-93CA-11D2-AA0D-00E098032B8C_NV+BS+RT+TBAWS_		
PKDefault_8BE4DF61-93CA-11D2-AA0D-00E098032B8C_NV+BS+RT_0		
SecureBoot_8BE4DF61-93CA-11D2-AA0D-00E098032B8C_BS+RT_0		
SecurityTokens_6320A8C8-9C93-4A71-B529-9F79C8761B8D_NV+BS+RT_0	bin	17
Setup_EC87D643-EBA4-4BB5-A1E5-3F3E36B20DA9_NV+BS+RT_0	bin	410
SetupDefault_EC87D643-EBA4-4BB5-A1E5-3F3E36B20DA9_NV+BS+RT_0	bin	410
SetupMode_8BE4DF61-93CA-11D2-AA0D-00E098032B8C_BS+RT_0	bin	1
SetupPlatformData_EC87D643-EBA4-4BB5-A1E5-3F3E36B20DA9_BS+RT_0	bin	16
SignatureSupport_8BE4DF61-93CA-11D2-AA0D-00E098032B8C_BS+RT_0	bin	80
TpmDeviceSelectionUpdate_EC87D643-EBA4-4BB5-A1E5-3F3E36B20DA9_NV+BS..	bin	1
TrEEPhysicalPresence_F24643C2-C622-494E-8A0D-4632579C2D5B_NV+BS+RT_0	bin	12
UsbSupport_EC87D643-EBA4-4BB5-A1E5-3F3E36B20DA9_NV+BS+RT_0	bin	32

AcpiGlobalVariable

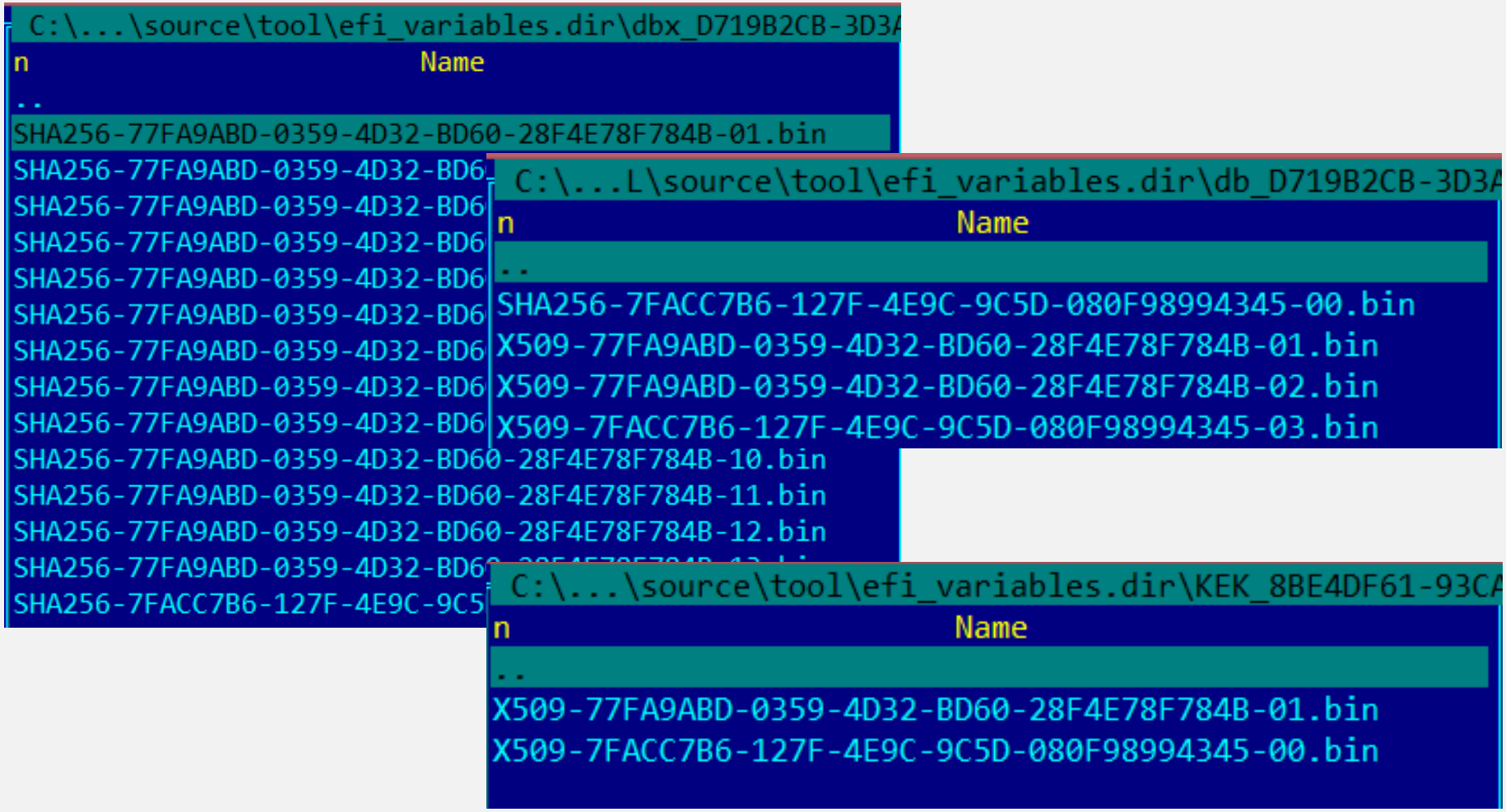
BootOrder vars

Secure Boot certificates (PK, KEK, db, dbx)

Setup Variable

Extracting UEFI Secure Boot keys…

```
# chipsec_util.py uefi keys db.bin / dbx.bin / kek.bin
```



Locating UEFI System Tables…

```
[uefi] EFI System Table:
49 42 49 20 53 59 53 54 1f 00 02 00 78 00 00 00 | IBI SYST    x
33 15 11 86 00 00 00 00 98 33 45 ff ff ff ff ff | 3        3E
70 22 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | p"
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
00 00 00 00 00 00 00 00 18 ae bf ff ff ff ff ff |
00 00 00 00 00 00 00 00 08 00 00 00 00 00 00 00 |
18 9e bf ff ff ff ff ff                         |
Header:
  Signature     : IBI SYST
  Revision      : 2.31
  HeaderSize    : 0x00000078
  CRC32         : 0x86111533
  Reserved      : 0x00000000
EFI System Table:
  FirmwareVendor      : 0xFFFFFFFFFF453398
  FirmwareRevision    : 0x0000000000002270
  ConsoleInHandle     : 0x0000000000000000
  ConIn               : 0x0000000000000000
  ConsoleOutHandle    : 0x0000000000000000
  ConOut              : 0x0000000000000000
  StandardErrorHandle : 0x0000000000000000
  StdErr              : 0x0000000000000000
  RuntimeServices     : 0xFFFFFFFFFFBFAE18
  BootServices        : 0x0000000000000000
  NumberOfTableEntries: 0x0000000000000008
  ConfigurationTable  : 0xFFFFFFFFFFBF9E18

[uefi] UEFI appears to be in Runtime mode
```

```
# chipsec_util.py uefi tables
```

Locating Runtime UEFI Services…

```
[uefi] EFI Runtime Services Table:
52 55 4e 54 53 45 52 56 1f 00 02 00 88 00 00 00 | RUNTSERV
6f aa 42 cb 00 00 00 00 2c 2b e0 fe ff ff ff ff | o B     ,+
bc 2c e0 fe ff ff ff ff 20 2e e0 fe ff ff ff ff |  ,       .
0c 30 e0 fe ff ff ff ff dc 14 65 da 00 00 00 00 |  0        e
00 14 65 da 00 00 00 00 34 0b d6 fe ff ff ff ff |   e     4
e0 0c d6 fe ff ff ff ff 3c 0e d6 fe ff ff ff ff |         <
ec e3 e0 fe ff ff ff ff 60 96 d4 fe ff ff ff ff |         `
f8 fa e0 fe ff ff ff ff 9c fd e0 fe ff ff ff ff |
cc 0f d6 fe ff ff ff ff                         |
Header:
  Signature     : RUNTSERV
  Revision      : 2.31
  HeaderSize    : 0x00000088
  CRC32         : 0xCB42AA6F
  Reserved      : 0x00000000
Runtime Services:
  GetTime                   : 0xFFFFFFFFFEE02B2C
  SetTime                   : 0xFFFFFFFFFEE02CBC
  GetWakeupTime             : 0xFFFFFFFFFEE02E20
  SetWakeupTime             : 0xFFFFFFFFFEE0300C
  SetVirtualAddressMap      : 0x00000000DA6514DC
  ConvertPointer            : 0x00000000DA651400
  GetVariable               : 0xFFFFFFFFFED60B34
  GetNextVariableName       : 0xFFFFFFFFFED60CE0
  SetVariable               : 0xFFFFFFFFFED60E3C
  GetNextHighMonotonicCount: 0xFFFFFFFFFEE0E3EC
  ResetSystem               : 0xFFFFFFFFFED49660
  UpdateCapsule             : 0xFFFFFFFFFEE0FAF8
  QueryCapsuleCapabilities : 0xFFFFFFFFFEE0FD9C
  QueryVariableInfo         : 0xFFFFFFFFFED60FCC
```

```
# chipsec_util.py uefi tables
```

Locating S3 Resume Boot Script Table…

AcpiGlobalVariable UEFI variable points to a structure in memory (**ACPI_VARIABLE_SET_COMPATIBILITY**)

```
[CHIPSEC] Reading EFI variable Name='AcpiGlobalVariable'..
[uefi] EFI variable AF9FFD67-EC10-488A-9DFC-
6CBF5EE22C2E:AcpiGlobalVariable:
18 be 89 da
```

```
[CHIPSEC] Reading: PA = 0x00000000DA89BE18, len = 0x100, output:
00 c0 6e da 00 00 00 00 00 40 08 00 00 00 00 00 |    n       @
00 00 00 00 00 00 00 00 18 a0 88 da 00 00 00 00 |
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
80 c0 89 da 00 00 00 00 40 c0 89 da 00 00 00 00 |         @
00 00 20 fa 00 00 00 00 00 00 00 00 00 00 00 00 |
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
```

```
# chipsec_util.py uefi s3bootscript
```

Extracting S3 Boot Script Table…

```
[CHIPSEC] Reading: PA = 0x00000000DA88A018, len = 0x100, output:
00 00 00 00 21 00 00 00 02 00 0f 01 00 00 00 00 |      !
00 00 c0 fe 00 00 00 00 01 00 00 00 00 00 00 00 |
00 01 00 00 00 24 00 00 00 02 02 0f 01 00 00 00 |       $
00 04 00 c0 fe 00 00 00 00 01 00 00 00 00 00 00 |
00 00 00 00 08 02 00 00 00 21 00 00 00 02 00 0f |           !
01 00 00 00 00 00 00 c0 fe 00 00 00 00 01 00 00 |
00 00 00 00 00 10 03 00 00 00 24 00 00 00 02 02 |            $
0f 01 00 00 00 00 04 00 c0 fe 00 00 00 00 01 00 |
00 00 00 00 00 00 00 07 00 00 04 00 00 00 24 00 |                 $
00 00 02 02 07 07 07 07 07 07 04 f4 d1 fe 00 00 |
00 00 01 00 00 00 00 00 00 00 80 00 00 00 05 00 |
00 00 28 00 00 00 03 02 00 00 00 00 00 00 14 90 |    (
d1 fe 00 00 00 00 00 00 00 00 00 00 00 00 01 00 |
00 00 00 00 00 00 06 00 00 00 28 00 00 00 03 00 |            (
00 00 00 00 00 00 04 90 d1 fe 00 00 00 00 01 00 |
00 00 00 00 00 00 f8 00 00 00 00 00 00 00 07 00 |
```

```
# chipsec_util.py uefi s3bootscript
```

Decoding S3 Boot Script Opcodes…

```
[000] Entry at offset 0x0000 (length = 0x21):
Data:
02 00 0f 01 00 00 00 00 00 00 c0 fe 00 00 00 00
01 00 00 00 00 00 00 00 00
Decoded:
  Opcode : S3_BOOTSCRIPT_MEM_WRITE (0x02)
  Width  : 0x00 (1 bytes)
  Address: 0xFEC00000
  Count  : 0x1
  Values : 0x00
..

[359] Entry at offset 0x2F2C (length = 0x20):
Data:
01 02 30 04 00 00 00 00 21 00 00 00 00 00 00 00
de ff ff ff 00 00 00 00
Decoded:
  Opcode : S3_BOOTSCRIPT_IO_READ_WRITE (0x01)
  Width  : 0x02 (4 bytes)
  Address: 0x00000430
  Value  : 0x00000021
  Mask   : 0xFFFFFFDE
```

 # chipsec_util.py uefi s3bootscript

Extracting CMOS Settings…

```
[CHIPSEC] Dumping CMOS memory..
Low CMOS contents:
....0...1...2...3...4...5...6...7...8...9...A...B...C...D...E...F
00..06  33  28  46  10  11  04  16  06  16  26  02  50  80  00  09
10..00  FF  FF  FF  0E  80  02  00  3C  FF  FF  FF  FF  FF  00  FF
20..FF  FF  FF  FF  FF  FF  FF  FF  FF  FF  FF  FF  FF  FF  17  B5
30..00  3C  20  FF  FF  E1  0C  FF  00  00  00  00  00  00  00  00
40..FF  FF  FF  FF  00  9F  00  00  00  00  00  00  00  00  00  00
50..00  00  00  00  FF  FF  FF  FF  3F  FF  FF  00  FF  FF  FF  FF
60..00  FF  FF  FF  FF  FF  FF  FF  FE  FF  00  30  7C  FF  FF  FF
70..FF  FF  FF  FF  FF  FF  FF  FF  FF  5A  FF  FF  49  53  B2  00
High CMOS contents:
....0...1...2...3...4...5...6...7...8...9...A...B...C...D...E...F
00..FF  FF  FF  FF  FF  FF  FF  FF  FF  FF  FF  FF  FF  FF  FF  FF
10..FF  FF  FF  00  FF  FF  FF  FF  FF  FF  FF  FF  FF  FF  32  3F
20..FF  FF  FF  FF  FF  FF  FF  FF  FF  FF  FF  FF  FF  FF  00  00
30..00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
40..00  00  FF  FF  FF  FF  FF  FF  FF  FF  FF  FF  FF  FF  FF  FF
50..FF  FF  FF  FF  FF  FF  FF  FF  FF  FF  FF  FF  FF  FF  FF  FF
60..FF  FF  FF  FF  EF  FF  FF  FF  FF  FF  FF  FF  FF  FF  FF  FF
70..FF  FF  FF  FF  FF  FF  FF  FF  FF  FF  FF  FF  FF  FF  FF  FF
[CHIPSEC] (cmos) time elapsed 0.011
```

```
# chipsec_util.py cmos dump
```

Locating ACPI Tables…

```
[acpi] found RSDP in EFI memory: 0x00000000DA871000
==============================================================================
  Root System Description Pointer (RSDP)
==============================================================================
  Signature         : RSD PTR
  Checksum          : 0x4C
  OEM ID            : _ASUS_
  Revision          : 0x02
  RSDT Address      : 0xDA871028
  Length            : 0x00000024
  XSDT Address      : 0x00000000DA871098
  Extended Checksum: 0xD3
  Reserved          : 00 00 00

[acpi] found XSDT at PA: 0x00000000DA871098
[CHIPSEC] Enumerating ACPI tables..
 - MSDM: 0x00000000DA61EE18
 - BGRT: 0x00000000DA887718
 - HPET: 0x00000000DA885420
 - XSDT: 0x00000000DA871098
 - ECDT: 0x00000000DA8831E0
 - FPDT: 0x00000000DA883198
 - APIC: 0x00000000DA883120
 - FACP: 0x00000000DA883010
 - MCFG: 0x00000000DA8832A8
 - SSDT: 0x00000000DA8873D0
```

```
# chipsec_util.py acpi list
```

Live System Forensics

Platform Hardware Configuration

```
chipsec_util pci enumerate
chipsec_util mmio list
chipsec_util mmio dump GTTMMADR
chipsec_util io list
chipsec_util cpu info
chipsec_util cpu cupid
chipsec_util ucode id
```

Live System Forensics

Operating System

```
chipsec_util idt
chipsec_util gdt
chipsec_util cpu pt
chipsec_util mem read 0x41E 0x20 kbrd_buffer.bin
chipsec_util mem pagedump 0xFED00000 0x100000
```

Live System Forensics

Platform Components

```
chipsec_util cmos dump
chipsec_util ec dump
chipsec_util ec read 0x2F
chipsec_util tpm state
chipsec_util tpm command pccrread 0
chipsec_util spd detect
chipsec_util spd dump
chipsec_util smbus read 0xA0 0x0 0x100
```

Live System Forensics

Virtual Machine Monitor (VMM) Configuration

```
chipsec_util iommu list
chipsec_util iommu config
chipsec_util iommu pt
chipsec_util vm status
chipsec_util vm pt
```

Live System Forensics

Checking Platform Configuration

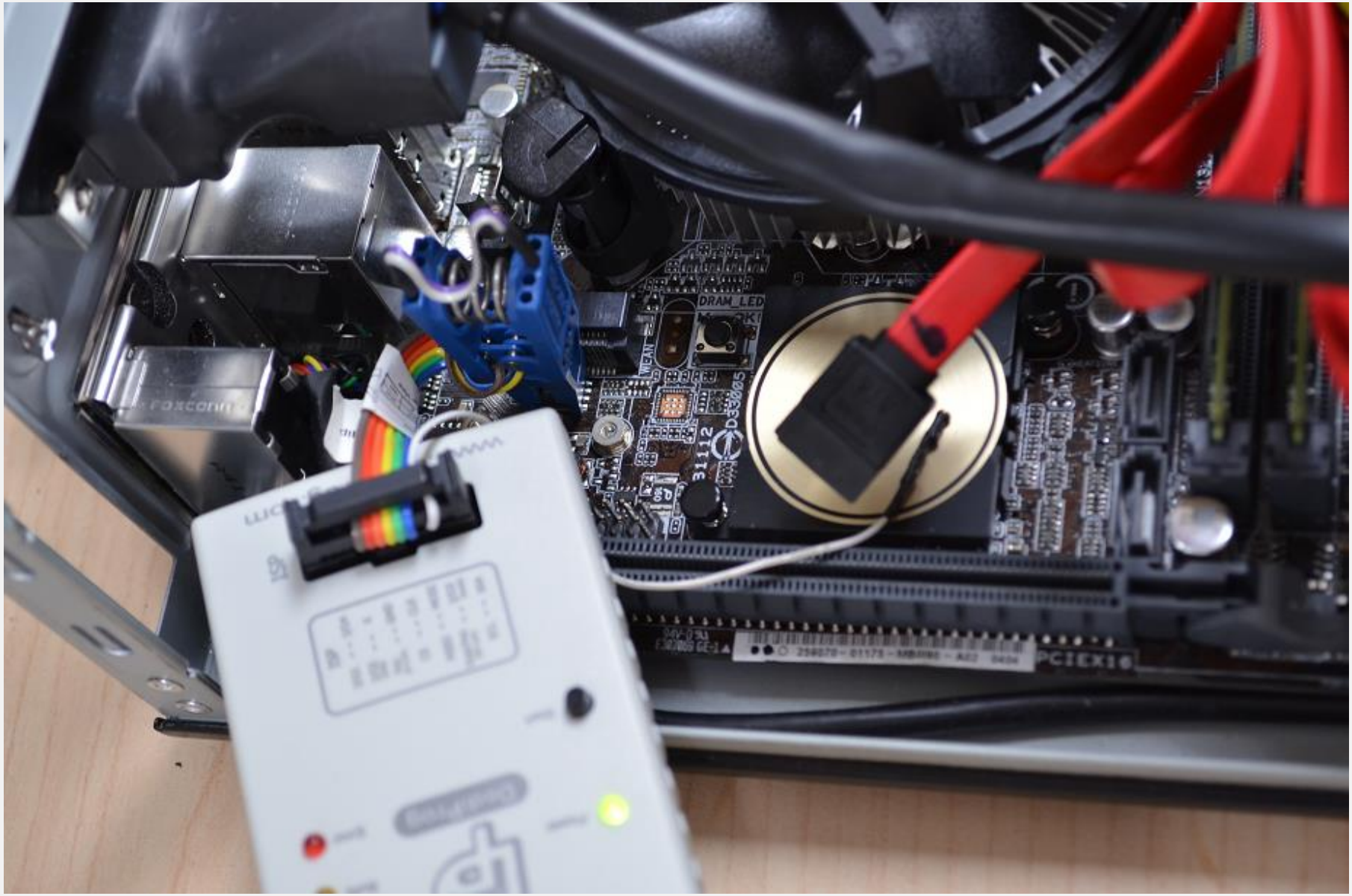
```
chipsec_main
chipsec_main -m common.bios_wp
chipsec_main -m common.smrr
chipsec_main -m common.spi_lock
chipsec_main -m common.spi_desc
chipsec_main -m common.secureboot.variables
chipsec_main -m common.uefi.s3bootscript
chipsec_main -m remap
chipsec_main -m smm_dma
```

Exercise 7.1

Online Forensics (UEFI Firmware and SPI Flash Contents)

7.2 Offline Forensic of Firmware Images

Important: ensure that firmware images were obtained using trusted hardware tools (that haven't been tampered with)


DRAM_LED
PCIEX16

Where to Start From?

1. Option 1: Find original firmware image
   - Check BIOS update (capsule) image or the BIOS image on the platform manufacturer’s web-site
   - Check BIOS security advisories to understand how the firmware could be compromised and infected
   - Compare multiple images including suspect and clean
2. Option 2: Extract a known good SPI memory image from a clean system (or from multiple systems)
3. Option 3: Make SPI memory dumps before and after the infection if you have an infector/dropper

Analyzing firmware images

1. Decode two images:

```
# chipsec_util.py decode original_bios.bin

# chipsec_util.py decode current_bios.bin
```

2. Extracted binaries and other artefacts extracted from the images are stored in directories: `original_bios.bin.dir`, `current_bios.bin.dir`

3. Compare the contents of the two directories with any tool of choice:

```
# diff -qr original_bios.bin.dir
current_bios.bin.dir
```

4. Analyze/compare extracted EFI executables and analyze/compare extracted NV variables
   - Contents of NVRAM (NV EFI variables) are dynamic and will differ between platform reboots (and even within the same boot)
   - BIOS update images downloaded from the manufacturer's web-site typically don't contain NVRAM
   - Update images may not contain non-BIOS regions of SPI flash memory (e.g. flash descriptor)

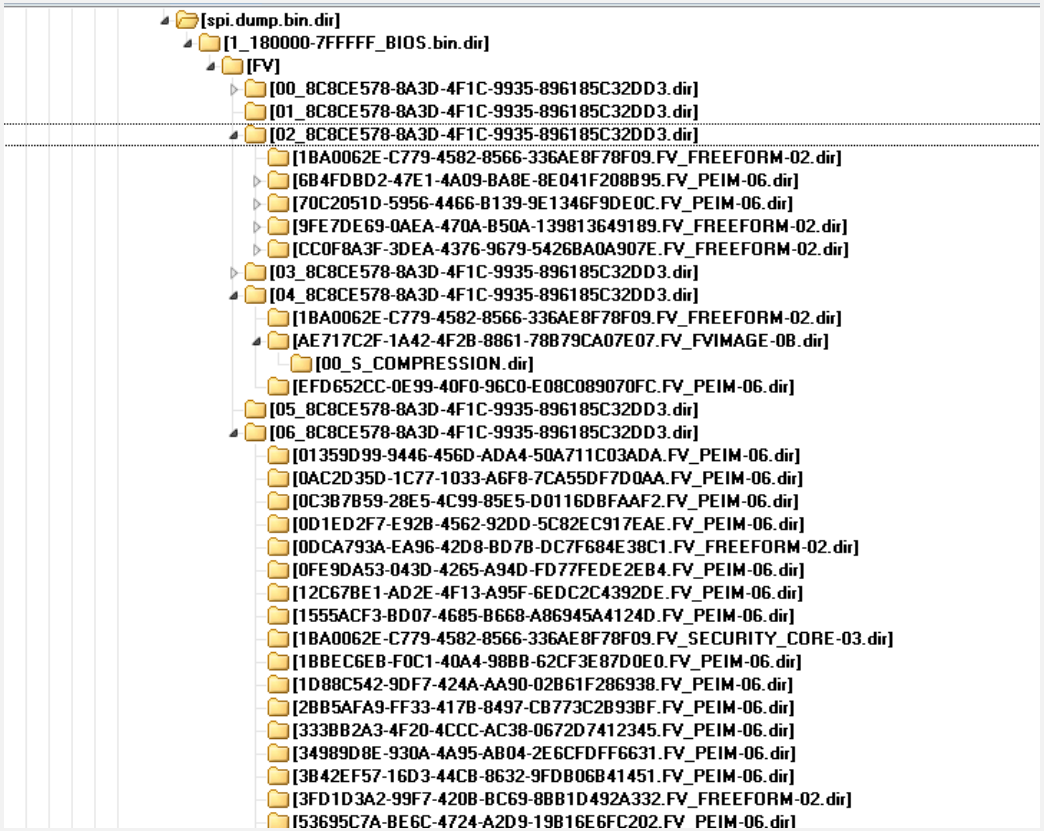
BIOS/Firmware Forensics: Offline

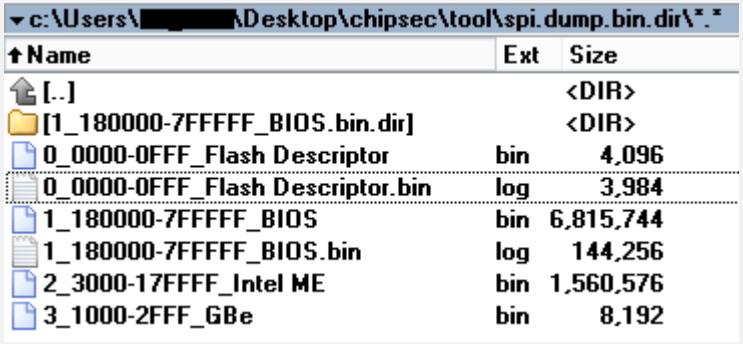
Offline system firmware analysis

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

Extracting EFI Executables

```
# chipsec_util.py decode spi.dump.bin
```






```
spi.dump.bin.dir
        1_180000-7FFFFF_BIOS.bin.dir                                    - SPI Flash Region
                FV
                  06_8C8CE578-8A3D-4F1C-9935-896185C32DD3.dir           - FV (FW volume)
                     F7D22BCA-1BCA-5591-CC8B-1CA98F2890FE.FV_PEIM-06.dir - EFI Executable
                          01_S_PE32.pe32.efi                            - Section
```

UEFI FW Volume (FV) Structure

```
Volume offset           : 0x00600000
File system GUID        : 8C8CE578-8A3D-4F1C-9935-896185C32DD3
Volume length           : 0x00080000 (524288)
Attributes              : 0x0003FEFF
Header length           : 0x00000048
Checksum                : 0xE6AE  (0xE6AE)
Extended Header Offset  : 0x00000000
     File offset : 0x00600048
     Name        : 92685943-D810-47FF-A112-CC8490776A1F
     Type        : 0x04
     Attributes  : 0x00000040
     State       : 0xF8
     Checksum    : 0xEAA3 (0xEAA3)
     Size        : 0x00E6DC (59100)
          Section offset : 0x00600060
          Name           : S_PE32
          Type           : 0x10

     File offset : 0x0060E728
     Name        : DF8556F0-3A61-11DE-8A39-0800200C9A66
     Type        : 0x06
     Attributes  : 0x00000040
     State       : 0xF8
     Checksum    : 0xA86D (0xA86D)
     Size        : 0x001E04 (7684)
          Section offset : 0x0060E740
          Name           : S_PEI_DEPEX
          Type           : 0x1B

          Section offset : 0x0060E748
          Name           : S_PE32
          Type           : 0x10
```

Firmware Volume

EFI Binary

Section

Flash Descriptor

```
# chipsec_util.py spidesc fd.bin

[spi_fd] Valid SPI flash descriptor found at offset 0x00000000

+ 0x0000 Reserved : FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
+ 0x0010 Signature: 0x0FF0A55A
...
Flash Regions
-------------------------------------------------------
 Region                 | FLREGx    | Base     | Limit
-------------------------------------------------------
0 Flash Descriptor      | 00000000  | 00000000 | 00000FFF
1 BIOS                  | 07FF0180  | 00180000 | 007FFFFF
2 Intel ME              | 017F0003  | 00003000 | 0017FFFF
3 GBe                   | 00020001  | 00001000 | 00002FFF
4 Platform Data         | 00007FFF  | 07FFF000 | 00000FFF (not used)

+ 0x0060 Master Section:
=======================================================
+ 0x0060 FLMSTR0    : 0x0A0B0000
+ 0x0064 FLMSTR1    : 0x0C0D0000
+ 0x0068 FLMSTR2    : 0x08080118

Master Read/Write Access to Flash Regions
-------------------------------------------------------
 Region                 | CPU/BIOS | ME       | GBe
-------------------------------------------------------
0 Flash Descriptor      | R        | R        |
1 BIOS                  | RW       |          |
2 Intel ME              |          | RW       |
3 GBe                   | RW       | RW       | RW
4 Platform Data         |          |          |
```

Access permissions to SPI flash regions

Reading EFI Configuration

```
# chipsec_util.py uefi nvram nvar rom.dump.bin
```

Path to extracted/parsed NVRAM contents:

NVRAM dump: `rom.dump.bin.dir\nvram_nvar.nvram.bin`

Decoded variables: `rom.dump.bin.dir\nvram_nvar.nvram.lst`

Format of NVRAM and variables are platform/BIOS specific.

CHIPSEC supports multiple types of NVRAM (evsa, nvar, vss, vss_new)

Decoding NVRAM from SPI dump

```
# type nvram_nvar.nvram.lst
...
--------------------------------
EFI Variable (offset = 0x4ec4):
--------------------------------
Name       : Setup
Guid       : EC87D643-EBA4-4BB5-A1E5-3F3E36B20DA9
Attributes: 0x7 ( NV+BS+RT )
Data:
01 5b 00 53 00 5b 00 01 01 72 00 68 00 72 00 01 |  [ S [   r h r
01 50 00 46 00 50 00 01 01 50 00 46 00 50 00 01 |  P F P   P F P
01 f8 11 18 15 b8 0b 10 0e b8 0b 10 0e 90 01 d0 |
07 00 00 00 00 e8 03 d0 07 02 fa 00 00 28 64 00 |              (d
03 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
00 00 00 00 00 00 00 00 00 01 00 00 00 00 00 00 |
00 00 00 01 00 00 01 01 01 01 01 01 00 00 00 00 |
00 00 00 00 01 02 00 01 00 00 00 00 00 00 00 00 |
00 00 00 00 00 01 02 01 08 00 00 00 00 00 00 00 |
00 00 01 00 00 00 00 00 00 00 00 00 00 00 00 00 |
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |
00 00 00 00 00 00 00 00 01 00 00 01 01 00 00 00 |
00 00 00 00 00 00 00 00 00 01 00 00 00 00 00 00 |
00 01 01 20 00 00 00 00 00 00 00 00 00 00 00 00 |
00 00 00 00 00 00 01 01 00 00 00 01 01 00 01 00 |
01 00 00 00 00 00 00 00 00 00 00 00 01 00 0a 00 |
00 01 00 00 02 01 00 01 00 00 00 01 00 00 00 00 |
...
```

Exercise 7.2

Offline Firmware Image Forensics

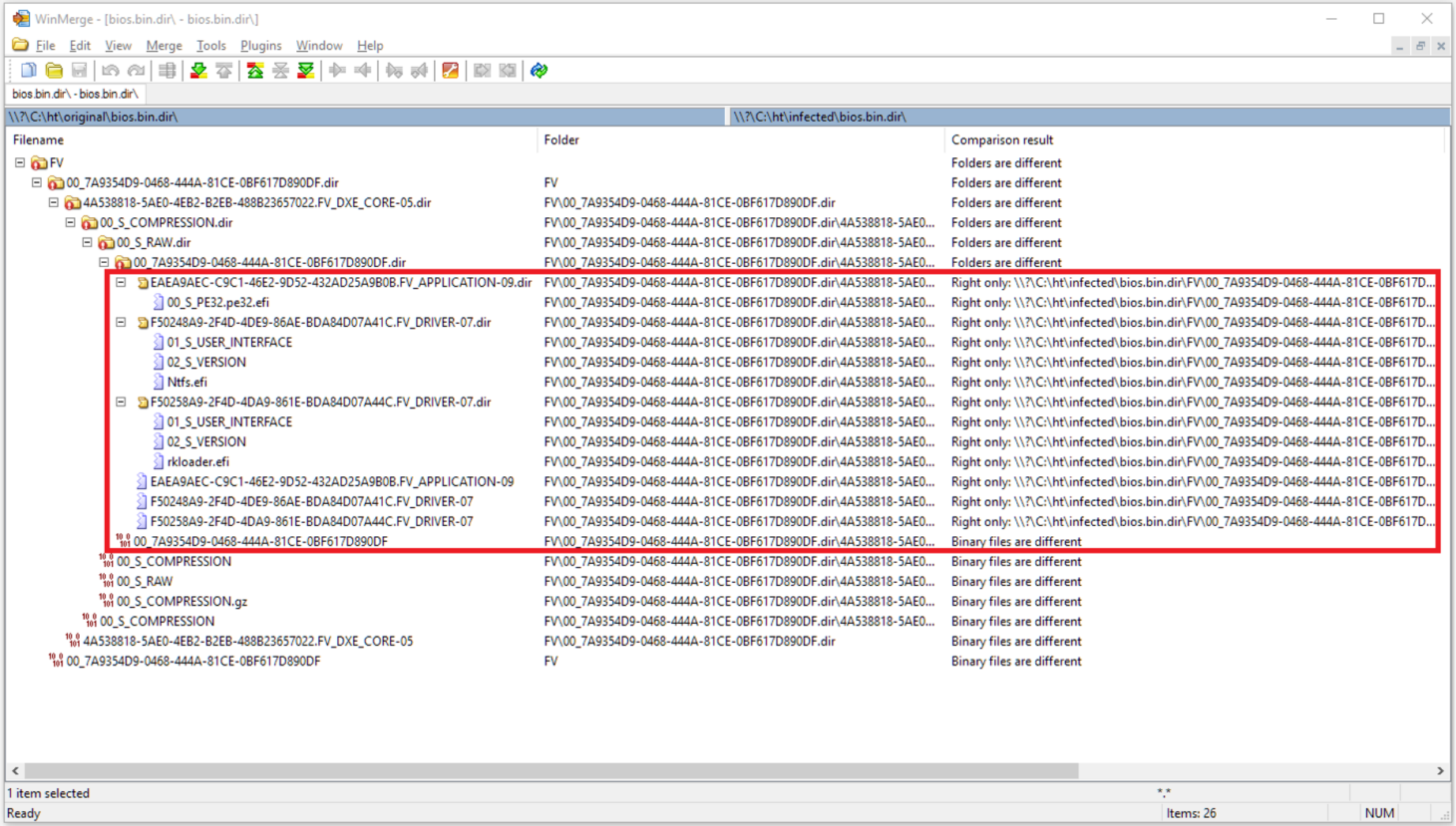
Case Study:
]HackingTeam[UEFI Rootkit

Case Study: HackingTeam's UEFI Rootkit

- Leaked emails reveal `Uefi_windows_persistent.zip` with UEFI based firmware image
- The image contains unexpected sections:
  1. *rkloader* DXE driver executable
  2. NTFS DXE driver executable
  3. Unnamed PE executable

Initial analysis by ATR from July 2015

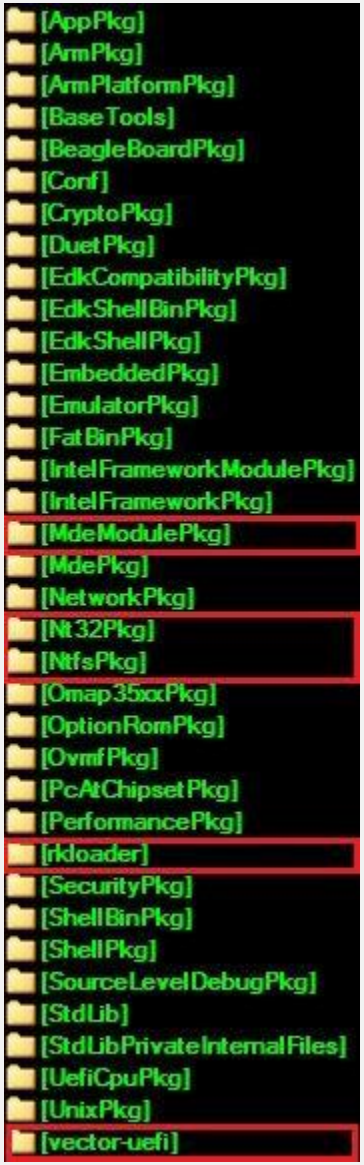
Case Study: HackingTeam's UEFI Rootkit



Case Study: HackingTeam's UEFI Rootkit

- From leaked source code, we can see that the "rkloader" is a DXE driver that is automatically executed during boot
- The module simply registers a callback (on the "READY_TO_BOOT" event) to execute the malicious payload

```
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;
}
```



Case Study: HackingTeam's UEFI Rootkit

The callback then loads a UEFI application, which does the following:

- Check for infection by looking for UEFI variable "fTA"

```
/**
    Leggo in NvRam la variabile fTA
**/
BOOLEAN
EFIAPI
CheckfTA()
{
    EFI_STATUS                  Status = EFI_SUCCESS;

    UINTN  VarDataSize;
    UINT8  VarData;


    VarData=0;
    VarDataSize=sizeof(VarData);
    Status=gRT->GetVariable(L"fTA", &gEfiGlobalFileVariableGuid, NULL, &VarDataSize, (UINTN*)&VarData);
```

- Use NTFS module to drop a backdoor (scoute.exe) and RCS agent (soldier.exe) onto the filesystem

```
#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)
```

Case Study: HackingTeam's UEFI Rootkit

Installation options

- Physical Access and a SPI programmer
- Booting a USB image to erase and reprogram firmware. Requires unlocked (vulnerable) firmware

Impact

- Automatic reinfection after removal of remote access components

Detection

- Look for `fTA` UEFI variable with GUID

  `8BE4DF61-93CA-11d2-aa0d-00e098302288`

- Examine SPI image for additional DXE modules

Exercise 7.3

Solve UEFI Crack Me

Training materials are available on Github

https://github.com/advanced-threat-research/firmware-security-training

Yuriy Bulygin	@c7zero
Alex Bazhaniuk	@ABazhaniuk
Andrew Furtak	@a_furtak
John Loucaides	@JohnLoucaides