現代 PC のブートキット対策の調査

松尾 和輝 / FFRI Security, Inc

Ring Minus Security

自己紹介: 松尾 和輝（@InfPCTechStack）

所属

FFRI セキュリティ 基礎技術研究室

研究分野

UEFI BIOS, SMM, ハイパーバイザー

研究発表

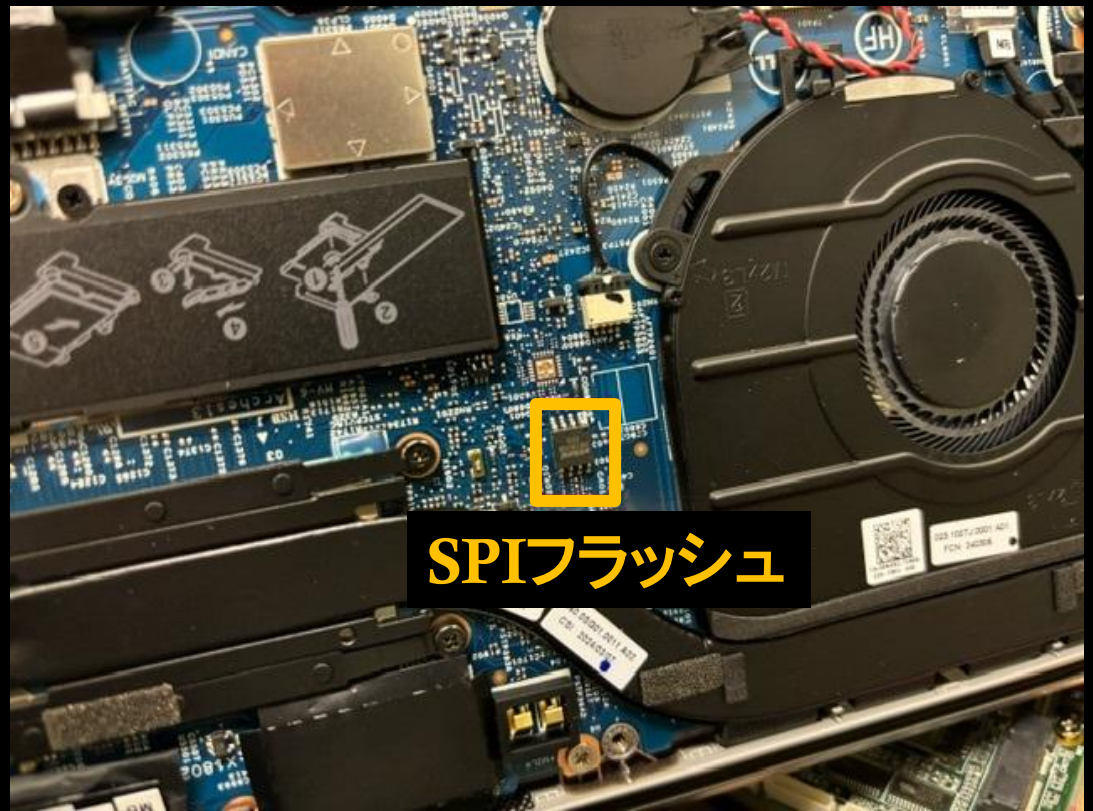
- SmmPack: Obfuscation for SMM Modules with TPM Sealed Key [DIMVA 2024]
- You've Already Been Hacked: What if There Is a Backdoor in Your UEFI OROM? [BHUSA 2024]
- Shade BIOS: Unleashing the Full Stealth of UEFI Malware [BHUSA 2025]

前提: UEFI セキュリティ 101

- PC のブート中、実行されるバイナリの格納場所

bootROM → SPIフラッシュ → ESP

  - bootROM は CPU の中にある
  - SPI フラッシュはマザボにある↓



SPIフラッシュ中身↓

UEFITool 0.27.0 - up2.bin

File Action Help

Structure

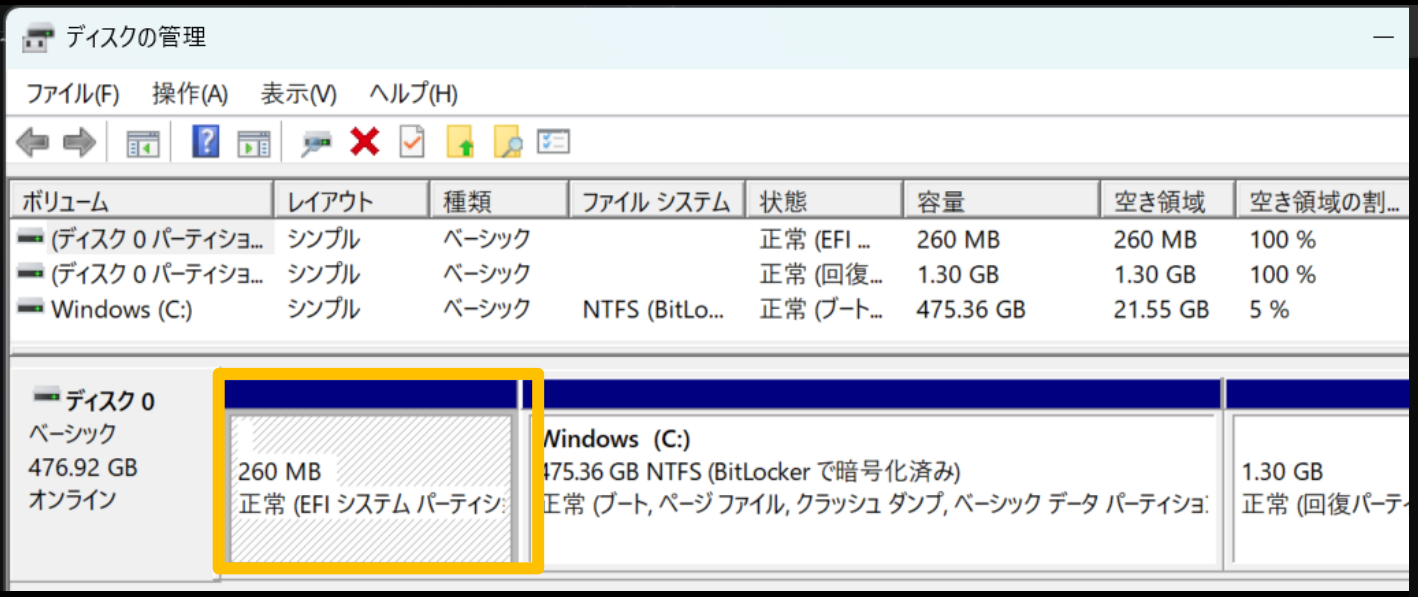
Name	Action	Type	Subtype	Text
Intel image		Image	Intel	
Descriptor region		Region	Descriptor	
BIOS region		Region	BIOS	
Padding		Padding	Non-empty	
8C8CE578-8A3D-4F1C-9935-896185C32DD3		Volume	FFSv2	
8C8CE578-8A3D-4F1C-9935-896185C32DD3		Volume	FFSv2	
Padding		Padding	Non-empty	
8C8CE578-8A3D-4F1C-9935-896185C32DD3		Volume	FFSv2	
8C8CE578-8A3D-4F1C-9935-896185C32DD3		Volume	FFSv2	
9E21FD93-9C72-4C15-8C4B-E77F1DB2D792		File	Volume image	
Compressed section		Section	Compressed	
Volume image section		Section	Volume image	
8C8CE578-8A3D-4F1C-9935-896185C32DD3		Volume	FFSv2	
FC510EE7-FFDC-11D4-BD41-0080C73C8881		File	Freeform	DXE apriori file
3374B583-7A96-496E-9B8C-2BEFE911ADB7		File	DXE driver	AaeonUartOverride
316A13C6-B0CA-4CC9-8599-5415BADF1923		File	DXE driver	AaeonUartOverridePrs
AB7ED12E-1D78-4635-AB87-23F00A911EC7		File	DXE driver	RomLayoutDxe
D6A2CB7F-6A18-4E2F-B43B-9920A733700A		File	DXE core	DxeCore
53BCC14F-C24F-434C-B294-8ED2D4CC1860		File	DXE driver	DataHubDxe
9B680FCE-AD6B-4F3A-B60B-F59899003443		File	DXE driver	DevicePathDxe
CD3BAEB6-50EB-4EE8-8E4E-AB74D2C1A600		File	DXE driver	EnglishDxe

前提: UEFI セキュリティ 101

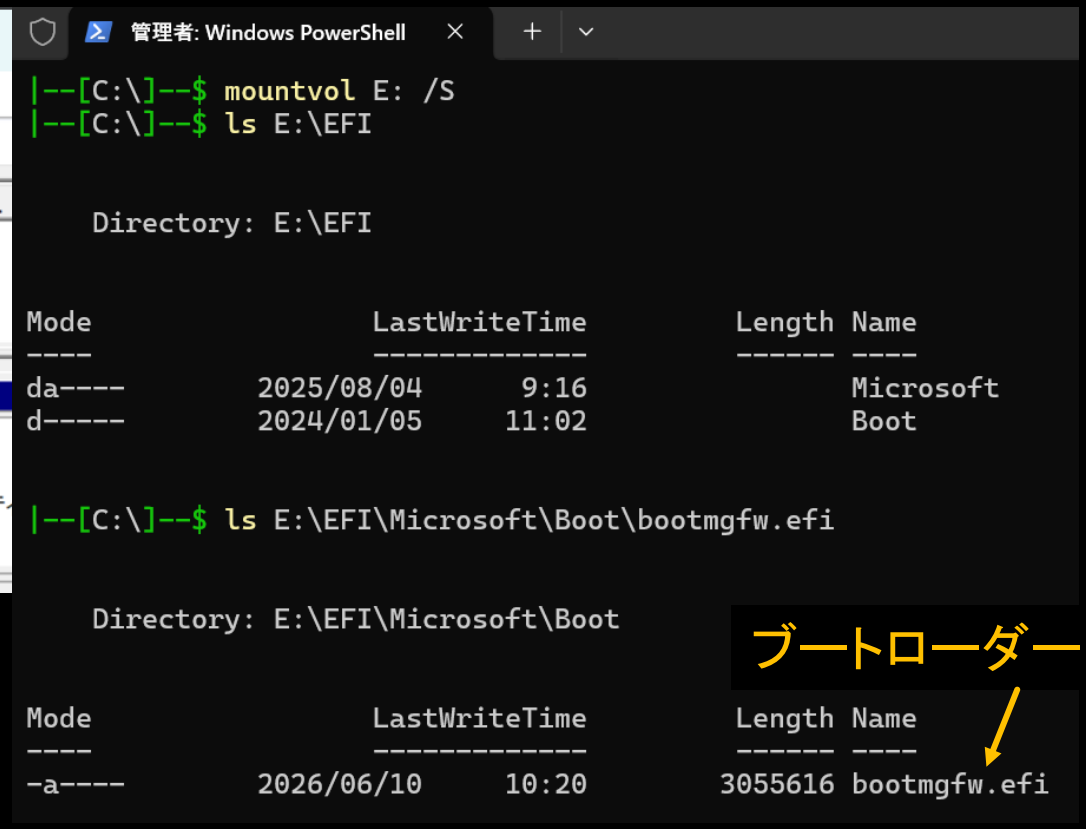
- PC のブート中、実行されるバイナリの格納場所

bootROM → SPIフラッシュ → ESP

  - ESP (Efi System Partition): ブートローダー(bootmgfw.efi) が置かれるディスク等のパーティション



ESP



ブートローダー

前提: UEFI セキュリティ 101

どこに UEFI マルウェア本体があるかで二種類に大別できる

- **ESP感染型**
  - ブートローダー (bootmgfw.efi) を置き換える
- **SPIフラッシュ感染型**
  - ソフトウェア経由 (PCIe の MMIO) あるいは物理的 (ROMライターなど) に書き込む

2018	Lojax (SPI)	悪性DXEモジュールがNTFSにユーザーランドマルウェアを入れる
2020	MosaicRegressor (SPI)	Lojaxと同じ
2021	FinSpy (ESP)	改竄したbootmgfw.efiからOSブートチェーンをフックしていき、exeをインジェクト
2021	ESPector (ESP)	FinSpyと同じだが、DSE無効化もしてる
2022	MoonBounce (SPI)	改竄したDXEコアからOSブートチェーンまでフックしていき、カーネルドライバーをメモリに入れる
2022	CosmicStrand (SPI)	改竄したDXEコアからOSブートチェーンをフックしていき、シェルコードをカーネルメモリに入れる
2023	BlackLotus (ESP)	bootmgfw.efi周辺のみをイジって、セキュアブート/HVCI等をバイパスしてカーネルドライバーを投下
2024	Bootkitty (ESP)	初の Linux 版で grubx64.efi を置き換える

In-the-wild の UEFI マルウェア一覧

概要

現代の PC は、BIOS の改竄からどのように守られているかを調査する

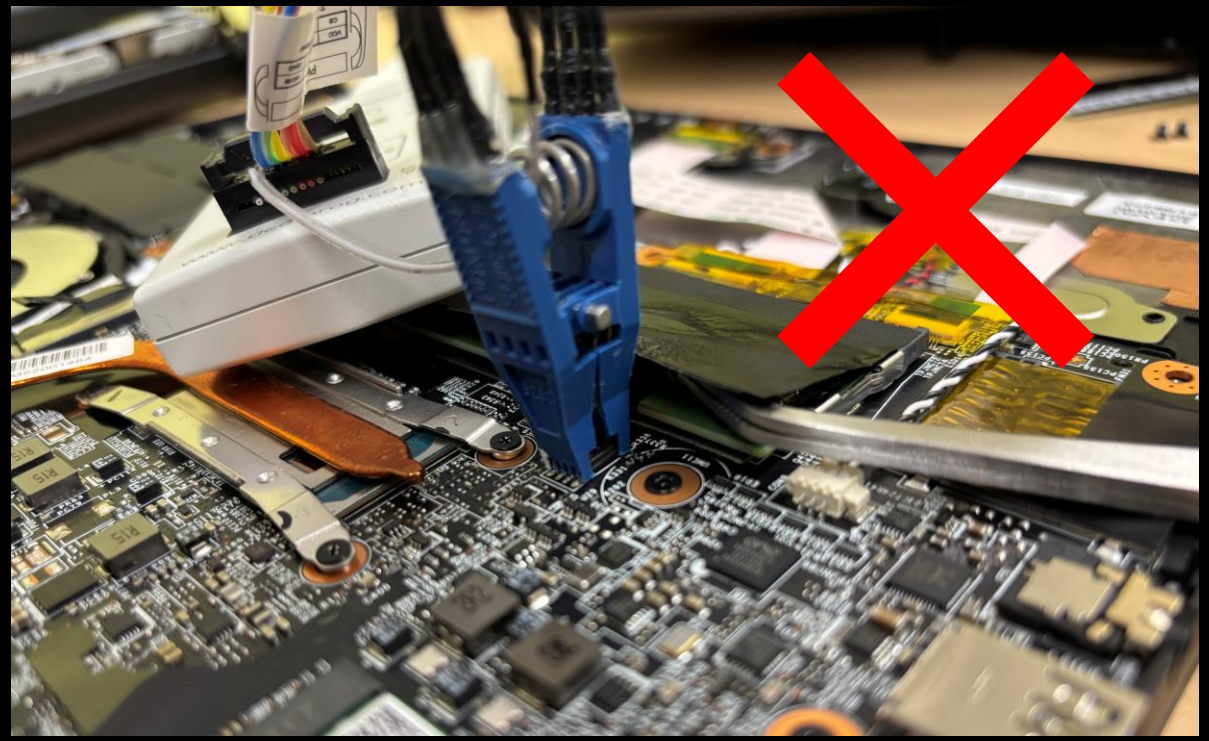
- 背景
  - ブートキットなどの「SPI フラッシュへの感染手法」はここ 7 年ぐらい聞かない
  - Lojax 時代のようなハードウェアロックの不備や SpeedRacer のような脆弱性はもう治っている
    - 補足: SPI フラッシュは OS 起動後の書き込みが SPI コントローラーによりロックされ再起動でしか解除できない
- 調査方法
  - 過去取り上げられた SPI フラッシュへの感染手法が、現代の PC で守られているかを実際に検証する
  - 脅威モデル: 物理アクセス/BIOS 設定書き換え 可能な攻撃者を想定する

1	0h RW/L	**Lock Enable (LE):** When set, setting the WP bit will cause SMI. When cleared, setting the WP bit will not cause SMI. Once set, this bit can only be cleared by a PLTRST#. When this bit is set, EISS - bit (5) of this register is locked down.
0	0h RW	**Write Protect Disable (WPD):** When set, access to the BIOS space is enabled for both read and write cycles to BIOS. When cleared, only read cycles are permitted to the FWH or SPI flash. When this bit is written from a 0 to a 1 and the LE bit is also set, an SMI# is generated. This ensures that only SMM code can update BIOS.

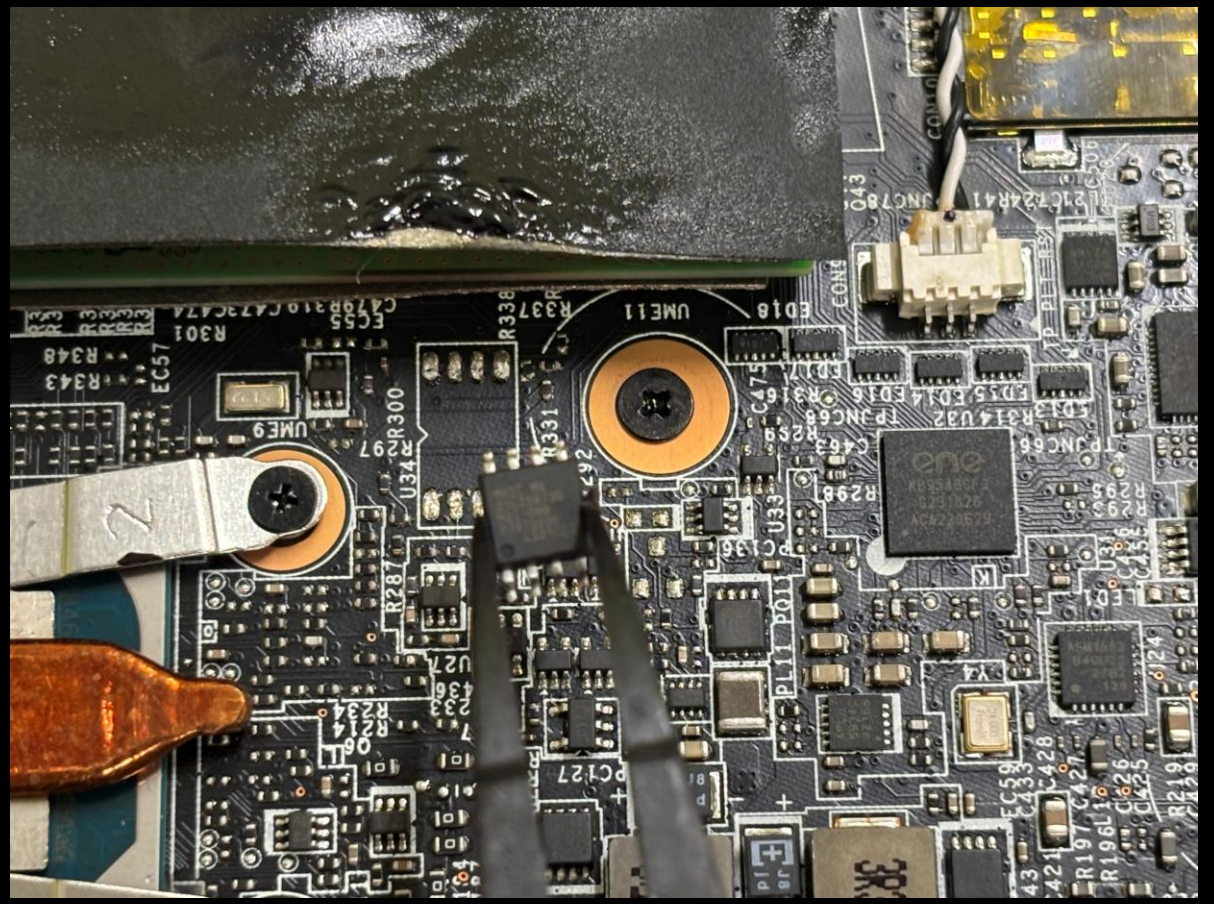
Intel® 500 Series Chipset Family On-Package Platform Controller Hub - Datasheet Volume 2
(2 Enhanced SPI Interface > 2.1.4 BIOS Control)

物理アクセス

最近のノートPC は ISP (In-System Programming) が使えない。
(※ おそらくSPI フラッシュの電源線が他のデバイスと共通で、ROMライターが他デバイスも起動してしまう。
故に他デバイスから SPI フラッシュへの通信と干渉する)

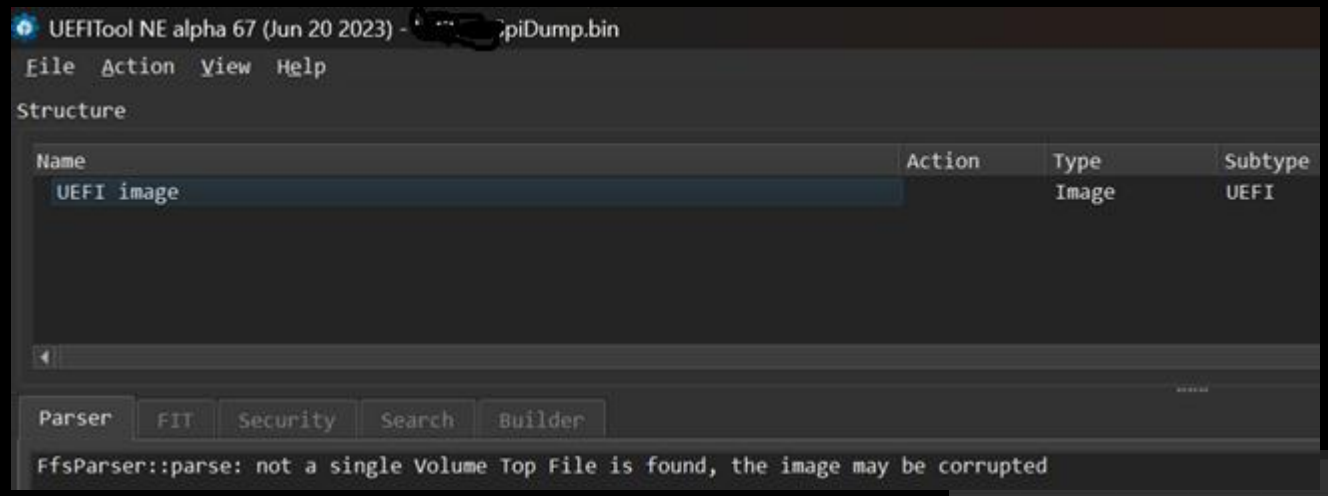


Desolder すれば読み書き可能 →



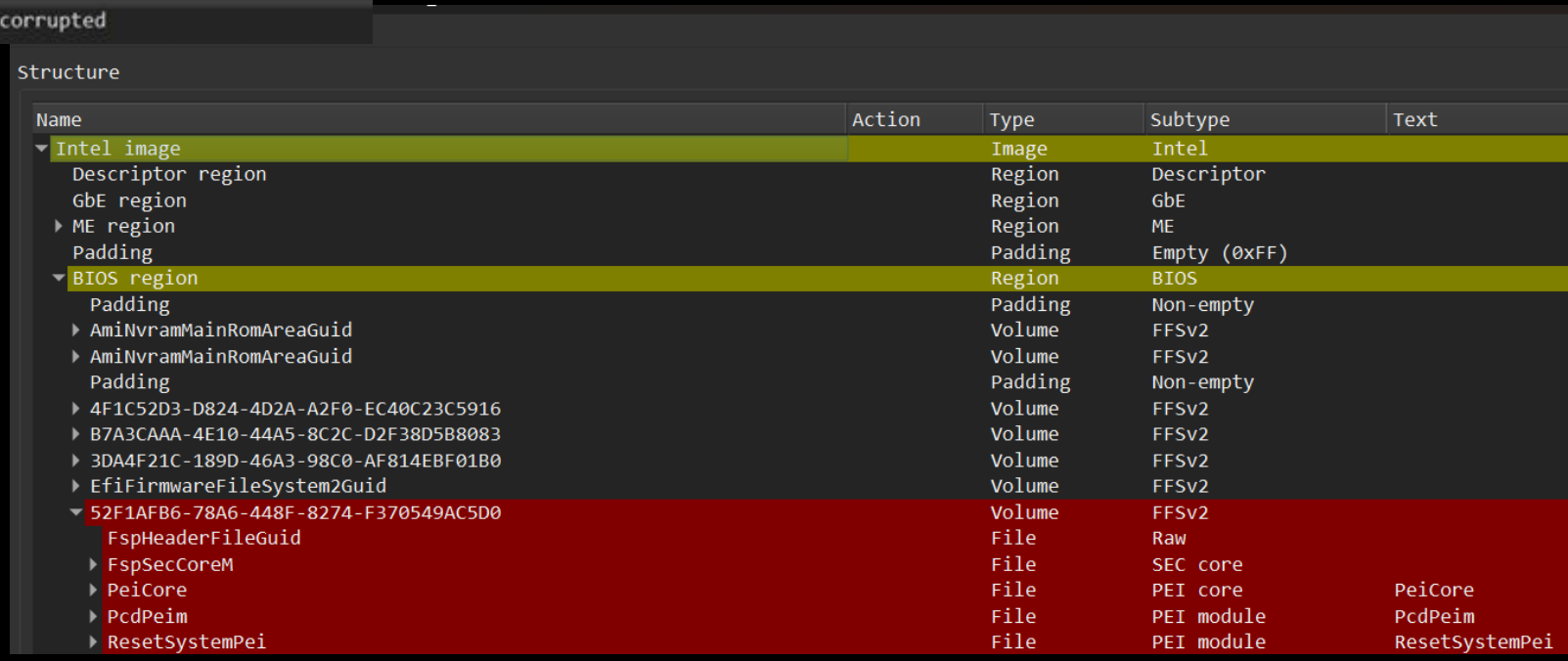
物理アクセス

最近のビジネス向け PC の SPI フラッシュは、さらに暗号化されてるっぽい



↑ビジネス向け PC (vPro 製品)

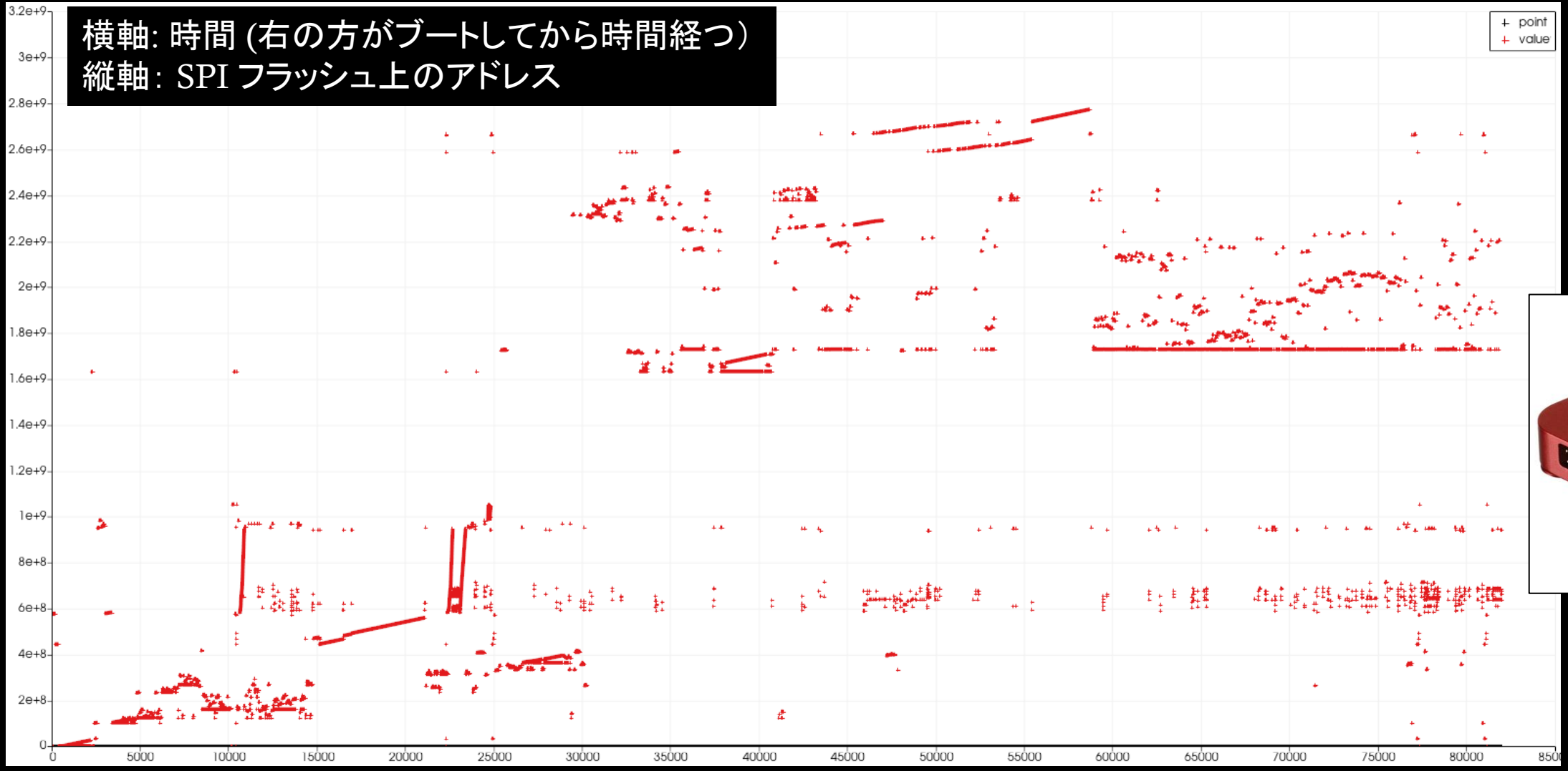
ビジネス用じゃない PC の BIOS ↓



物理アクセス

暗号化された SPI フラッシュは、アクセスもかなり複雑になってる模様
（普通フラッシュ上の BIOS 領域を上から読んでメモリにロードするので線形上がりになる）

横軸: 時間 (右の方がブートしてから時間経つ)
縦軸: SPI フラッシュ上のアドレス



Saleae ロジアナで解析

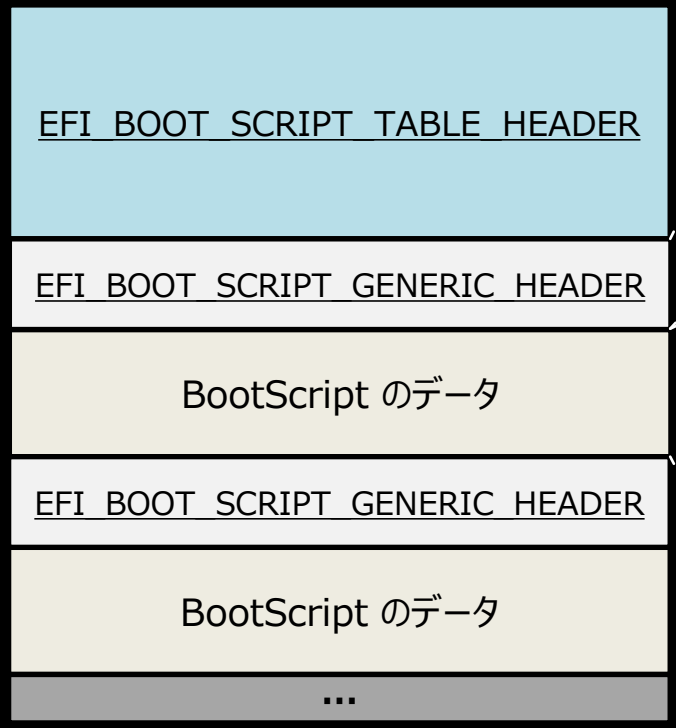
物理アクセス

結論: 物理アクセスによる読み書きは難しくなっている

S3 BootScript 攻撃

S3 BootScript 攻撃 / S3 Resume 攻撃 とは

- 一昔前のほとんどの研究において、OS 上から BIOS への感染に使われていた手法
- スリープから復帰する時にデバイスの設定等を戻すために実行する BootScript に SPI フラッシュに感染するコードを書く（スリープから復帰中は SPI フラッシュのロックが解除されているので）
- 詳細: BootScript で OpCode=0x08/09 を使うと指定したアドレスの任意コードを実行させられる



S3 BootScript

```
typedef struct {
  UINT16    OpCode;
  UINT8     Length;
} EFI_BOOT_SCRIPT_GENERIC_HEADER;
```

```
typedef struct {
  UINT16                OpCode;
  UINT8                 Length;
  EFI_PHYSICAL_ADDRESS  EntryPoint;
} EFI_BOOT_SCRIPT_DISPATCH;

typedef struct {
  UINT16                OpCode;
  UINT8                 Length;
  EFI_PHYSICAL_ADDRESS  EntryPoint;
  EFI_PHYSICAL_ADDRESS  Context;
} EFI_BOOT_SCRIPT_DISPATCH_2;
```

```
// ****************************************
// EFI Boot Script Opcode definitions
// ****************************************
#define EFI_BOOT_SCRIPT_IO_WRITE_OPCODE                0x00
#define EFI_BOOT_SCRIPT_IO_READ_WRITE_OPCODE           0x01
#define EFI_BOOT_SCRIPT_MEM_WRITE_OPCODE               0x02
#define EFI_BOOT_SCRIPT_MEM_READ_WRITE_OPCODE          0x03
#define EFI_BOOT_SCRIPT_PCI_CONFIG_WRITE_OPCODE        0x04
#define EFI_BOOT_SCRIPT_PCI_CONFIG_READ_WRITE_OPCODE   0x05
#define EFI_BOOT_SCRIPT_SMBUS_EXECUTE_OPCODE           0x06
#define EFI_BOOT_SCRIPT_STALL_OPCODE                   0x07
#define EFI_BOOT_SCRIPT_DISPATCH_OPCODE                0x08
#define EFI_BOOT_SCRIPT_DISPATCH_2_OPCODE              0x09
#define EFI_BOOT_SCRIPT_INFORMATION_OPCODE             0x0A
#define EFI_BOOT_SCRIPT_PCI_CONFIG2_WRITE_OPCODE       0x0B
#define EFI_BOOT_SCRIPT_PCI_CONFIG2_READ_WRITE_OPCODE  0x0C
#define EFI_BOOT_SCRIPT_IO_POLL_OPCODE                 0x0D
#define EFI_BOOT_SCRIPT_MEM_POLL_OPCODE                0x0E
#define EFI_BOOT_SCRIPT_PCI_CONFIG_POLL_OPCODE         0x0F
#define EFI_BOOT_SCRIPT_PCI_CONFIG2_POLL_OPCODE        0x10
```

https://github.com/tianocore/edk2/blob/master/MdePkg/Include/Pi/PiS3BootScript.h

S3 BootScript 攻撃

防御機構: SMM LockBox

- OS 起動後の BootScript への書き込みを禁止する機構
- BootScript は SMRAM に保存され、BootScript を編集する SMI ハンドラーは SMRAM ロックと同時に無効になる

```
254  /**
255    Dispatch function for a Software SMI handler.
270  EFI_STATUS
271  EFIAPI
272  SmmLockBoxHandler (
273    IN EFI_HANDLE  DispatchHandle,
274    IN CONST VOID  *Context         OPTIONAL,
275    IN OUT VOID    *CommBuffer      OPTIONAL,
276    IN OUT UINTN   *CommBufferSize  OPTIONAL
277    )
314    switch (LockBoxParameterHeader->Command) {
315      case EFI_SMM_LOCK_BOX_COMMAND_SAVE:
316        if (TempCommBufferSize < sizeof (EFI_SMM_LOCK_BOX_PARAMETER_SAVE)) {
317          DEBUG ((DEBUG_ERROR, "SmmLockBox Command Buffer Size for SAVE inva
318          break;
319        }
320
321        SmmLockBoxSave ((EFI_SMM_LOCK_BOX_PARAMETER_SAVE *)(UINTN)LockBoxParameterHeader);
```

```
45   VOID
46   SmmLockBoxSave (
47     IN EFI_SMM_LOCK_BOX_PARAMETER_SAVE  *LockBoxParameterSave
48     )
49   {
50     EFI_STATUS                      Status;
51     EFI_SMM_LOCK_BOX_PARAMETER_SAVE  TempLockBoxParameterSave;
52
53     //
54     // Sanity check
55     //
56     if (mLocked) {
57       DEBUG ((DEBUG_ERROR, "SmmLockBox Locked!\n"));
58       LockBoxParameterSave->Header.ReturnStatus = (UINT64)EFI_ACCESS_DENIED;
59       return;
60     }
```

SMRAM がロックされたら True になるグローバル変数

https://github.com/tianocore/edk2/blob/master/MdeModulePkg/Universal/LockBox/SmmLockBox/SmmLockBox.c

S3 BootScript 攻撃

防御機構: SMM LockBox

- OS 起動後の BootScript への書き込みを禁止する機構
- BootScript は SMRAM に保存され、BootScript を編集する SMI ハンドラーは SMRAM ロックと同時に無効になる

ちゃんと実装されてる？

```
254   /**
255     Dispatch function for a Software SMI handler
270   EFI_STATUS
271   EFIAPI
272   SmmLockBoxHandler (
273     IN EFI_HANDLE  DispatchHandle,
274     IN CONST VOID  *Context         OPTIONAL,
275     IN OUT VOID    *CommBuffer      OPTIONAL,
276     IN OUT UINTN   *CommBufferSize  OPTIONAL
277     )
314     switch (LockBoxParameterHeader->Command) {
315       case EFI_SMM_LOCK_BOX_COMMAND_SAVE:
316         if (TempCommBufferSize < sizeof (EFI_SMM_LOCK_BOX_PARAMETER_SAVE)) {
317           DEBUG ((DEBUG_ERROR, "SmmLockBox Command Buffer Size for SAVE inva
318           break;
319         }
320
321         SmmLockBoxSave ((EFI_SMM_LOCK_BOX_PARAMETER_SAVE *)(UINTN)LockBoxParameterHeader);
```

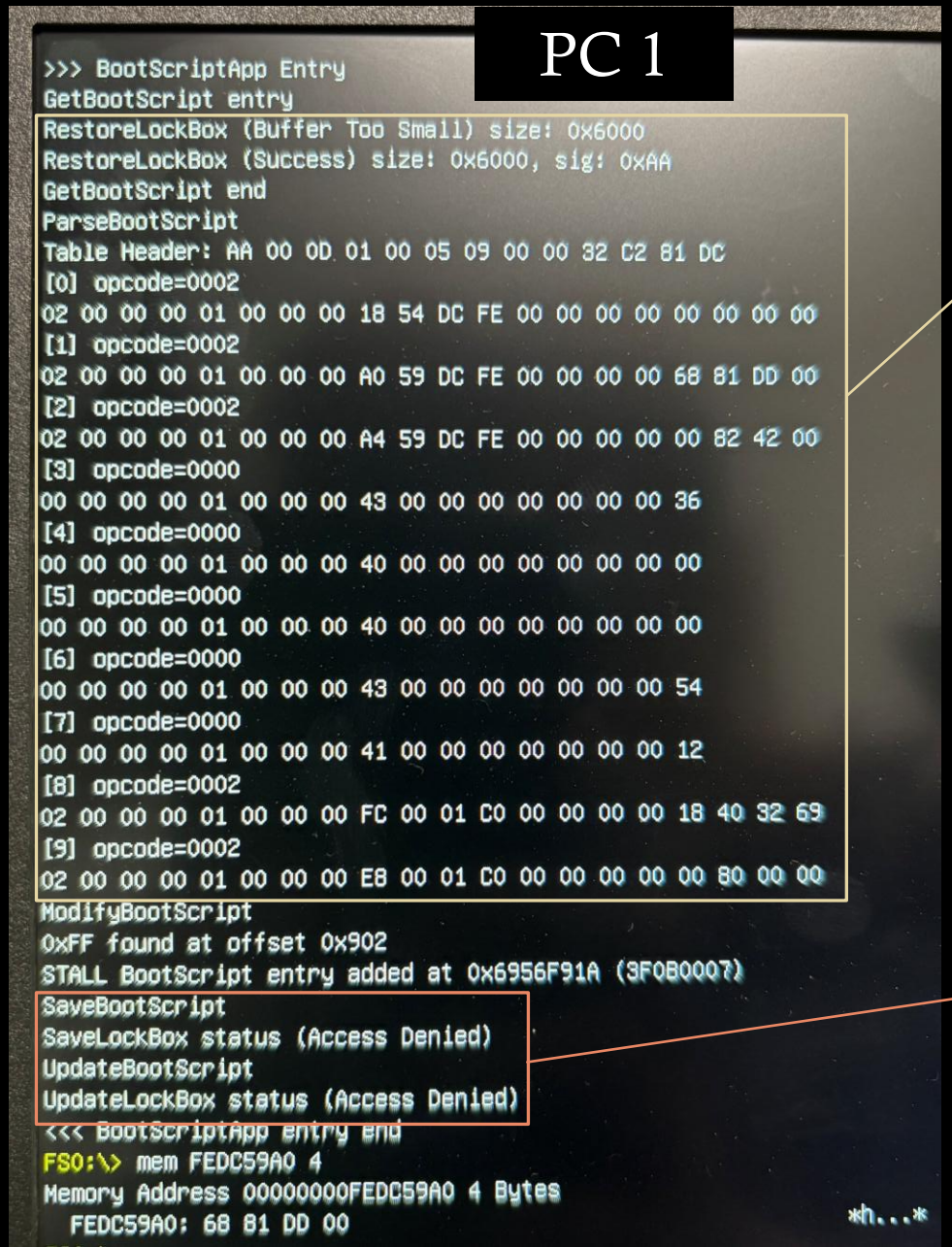
```
                                                    *LockBoxParameterSave
48      )
49    {
50      EFI_STATUS                      Status;
51      EFI_SMM_LOCK_BOX_PARAMETER_SAVE  TempLockBoxParameterSave;
52
53      //
54      // Sanity check
55      //
56      if (mLocked) {
57        DEBUG ((DEBUG_ERROR, "SmmLockBox Locked!\n"));
58        LockBoxParameterSave->Header.ReturnStatus = (UINT64)EFI_ACCESS_DENIED;
59        return;
60      }
```

SMRAM がロックされたら True になるグローバル変数

https://github.com/tianocore/edk2/blob/master/MdeModulePkg/Universal/LockBox/SmmLockBox/SmmLockBox.c

S3 BootScript 攻撃

PC 1



RestoreLockBox で BootScript を読む事はできる

SaveLockBox/UpdateLockBox で BootScript を書き込めるが、
“Access Denied” でちゃんと弾かれる

S3 BootScript 攻撃

PC 2

```
>>> BootScriptApp Entry
GetBootScript entry
RestoreLockBox (Buffer Too Small) size: 0x6000
RestoreLockBox (Success) size: 0x6000, sig: 0xAA
GetBootScript end
ParseBootScript
Table Header: AA 00 0D 01 00 09 09 00 00 C5 D7 C5 D7
[0] opcode=0000
00 00 00 00 01 00 00 00 B2 00 00 00 00 00 00 00 D0
[1] opcode=0002
02 00 00 00 02 00 00 00 30 03 D3 FE 00 00 00 00 41 00 00 9A 00 00 00 00
[2] opcode=0002
02 00 00 00 02 00 00 00 08 03 D3 FE 00 00 00 00 00 00 0F 00 00 00 00 00
[3] opcode=0002
02 00 00 00 02 00 00 00 00 00 03 D3 FE 00 00 00 00 00 00 F1 99 00 00 00 00
[4] opcode=0002
02 00 00 00 02 00 00 00 78 02 D3 FE 00 00 00 00 00 00 05 00 00 00 00 00
[5] opcode=0002
02 00 00 00 02 00 00 00 70 02 D3 FE 00 00 00 00 00 00 EC 99 00 00 00 00
[6] opcode=0002
02 00 00 00 01 00 00 00 84 1E 89 FD 00 00 00 00 F0 00 02 00
[7] opcode=0002
02 00 00 00 01 00 00 00 54 27 88 FD 00 00 00 00 F0 00 02 00
[8] opcode=000E
02 00 00 00 B0 1E 89 FD 00 00 00 00 01 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00 00 00
[9] opcode=0002
02 00 00 00 01 00 00 00 B0 1E 89 FD 00 00 00 00 01 FF FC 00
ModifyBootScript
0xFF found at offset 0x906
STALL BootScript entry added at 0x59A9891E (3F0B0007)
SaveBootScript
SaveLockBox status (Access Denied)
UpdateBootScript
UpdateLockBox status (Access Denied)
<<< BootScriptApp entry end
FS0:\> mem FED30300 8
Memory Address 00000000FED30300 8 Bytes
  FED30300: 00 00 F1 99 00 00 00 00-
FS0:\> _
```

PC 3

```
>>> BootScriptApp Entry
GetBootScript entry
RestoreLockBox (Buffer Too Small) size: 0x6000
RestoreLockBox (Success) size: 0x6000, sig: 0xAA
GetBootScript end
ParseBootScript
Table Header: AA 00 0D 01 00 36 05 00 00 00 00 00 00
[redacted]
[0] opcode=0002
02 00 00 00 01 00 00 00 84 1E 89 FD 00 00 00 00 F0 00 02 00
[1] opcode=0002
02 00 00 00 01 00 00 00 54 27 88 FD 00 00 00 00 F0 00 02 00
[2] opcode=000E
02 00 00 00 B0 1E 89 FD 00 00 00 00 01 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00 00 00 00 00 00 00
[3] opcode=0002
02 00 00 00 01 00 00 00 B0 1E 89 FD 00 00 00 00 01 20 FC 00
[4] opcode=000E
02 00 00 00 B0 1E 89 FD 00 00 00 00 01 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00 00 00 00 00 00 00
[5] opcode=0002
02 00 00 00 01 00 00 00 50 27 88 FD 00 00 00 00 01 20 FC 00
[6] opcode=0001
02 00 00 00 30 1B 00 00 00 00 00 00 00 20 00 00 FF FF FF FF
[7] opcode=0005
02 00 00 00 DC D0 0F 00 00 00 00 00 00 08 00 00 FF FE FF FF
[8] opcode=0002
02 00 00 00 01 00 00 00 FC 00 01 C0 00 00 00 00 18 50 CE 61
[9] opcode=0002
02 00 00 00 01 00 00 00 E8 00 01 C0 00 00 00 00 00 80 00 00
ModifyBootScript
0xFF found at offset 0x533
STALL BootScript entry added at 0x5DD6C54B (3F0B0007)
SaveBootScript
SaveLockBox status (Access Denied)
UpdateBootScript
UpdateLockBox status (Access Denied)
<<< BootScriptApp entry end
FS0:\> _
```

ちゃんと弾かれる

S3 BootScript 攻撃

結論: 流石に SMM LockBox はどの PC でも実装されていて、S3 BootScript 攻撃は難しい

まとめ

- SPI フラッシュの改竄は難しくなっている
  - 物理アクセスは SPI フラッシュの暗号化等で書き込めない
  - 一昔前に多様されていた S3 BootScript 攻撃の対策 SMM LockBox はちゃんと実装されている
- その他の既存手法もざっと見てるが、どれも難しい
  - BIOS アップデートの改竄 => 署名チェック不備/無し が前はあったが、今はちゃんと実装されてる
  - EC (Embedded Controller) の悪用 (例: BHUSA2019 Alex) => EC がホストに出来ること少なくなってる
  - DMA => IOMMU によって対策。TEE とか PC盗難 にも関わるのでその分対策は厳重
- Future Work
  - 現代 PC に通用する見落としている SPI フラッシュへの感染手法はあるか？
  - ESP (EFI System Partition) は簡単に改竄できるので、ESP からの攻撃可能性は重要になってきている
  - ESP に一度感染した後、より早いブートフェーズでコード実行できるエクスプロイトは無いか？ (S3BootScript みたく)
- ちなみに、HP の PC はかなり独自のセキュリティ機構が載っているっぽい
  - HP Endpoint Security Controller という独自チップがあり、それが SPI フラッシュの完全性を検証してる
  - https://jp.ext.hp.com/content/dam/jp-ext-hp-com/jp/ja/ec/business-solution/security/pdf/hp_sure_start_gen4.pdf