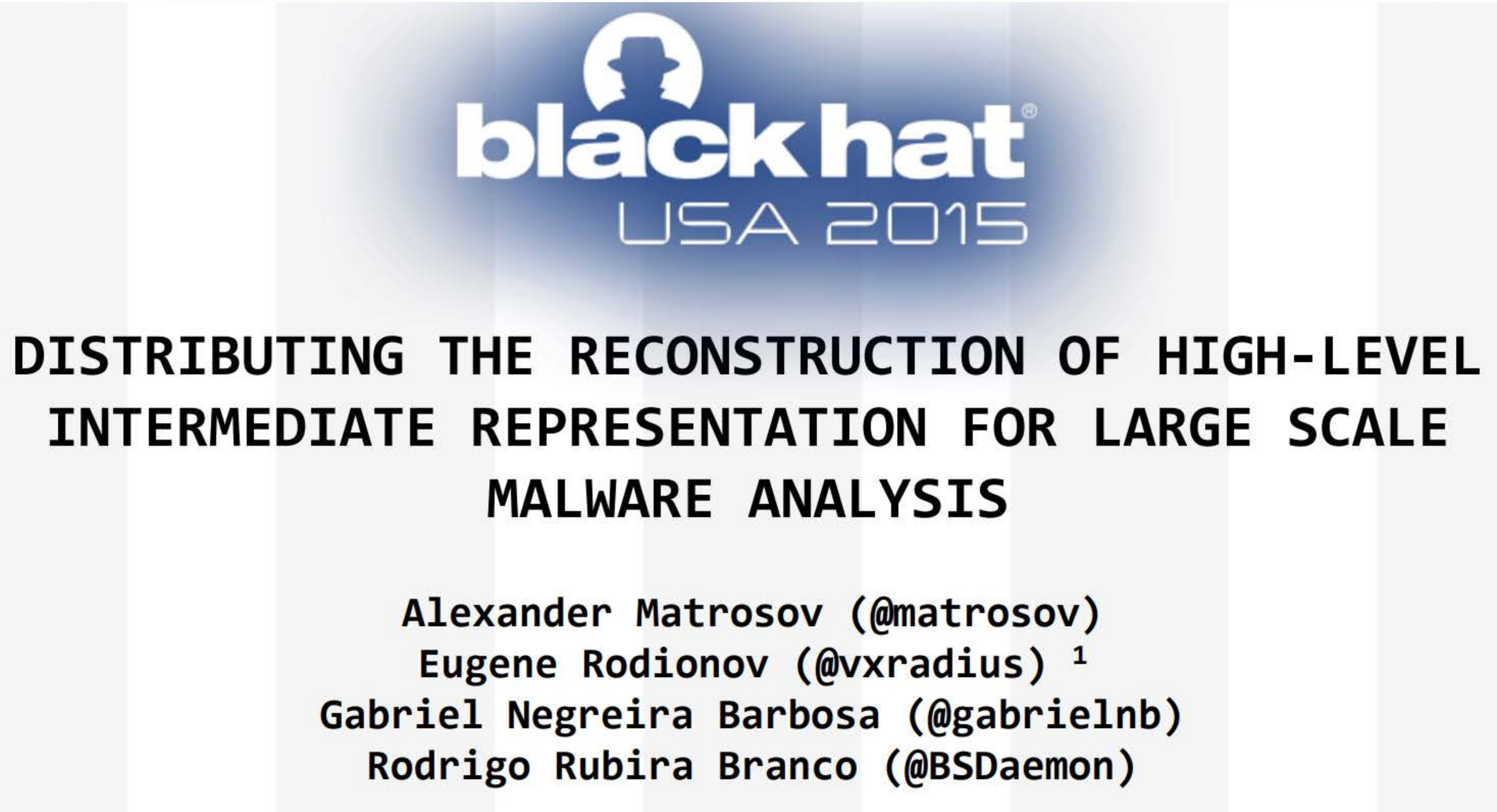
black hat®
EUROPE 2020
DECEMBER 9-10
BRIEFINGS

efiXplorer
Hunting for UEFI Firmware Vulnerabilities at Scale with Automated Static Analysis

Alex Matrosov, Andrey Labunets,
Yegor Vasilenko, Philip Lebedev



black hat USA 2015

DISTRIBUTING THE RECONSTRUCTION OF HIGH-LEVEL INTERMEDIATE REPRESENTATION FOR LARGE SCALE MALWARE ANALYSIS

Alexander Matrosov (@matrosov)
Eugene Rodionov (@vxradius) [1]
Gabriel Negreira Barbosa (@gabrielnb)
Rodrigo Rubira Branco (@BSDaemon)

https://www.blackhat.com/docs/us-15/materials/us-15-Branco-Distributing-The-Reconstruction-Of-High-Level-Intermediate-Representation-For-Large-Scale-Malware-Analysis.pdf

Outline

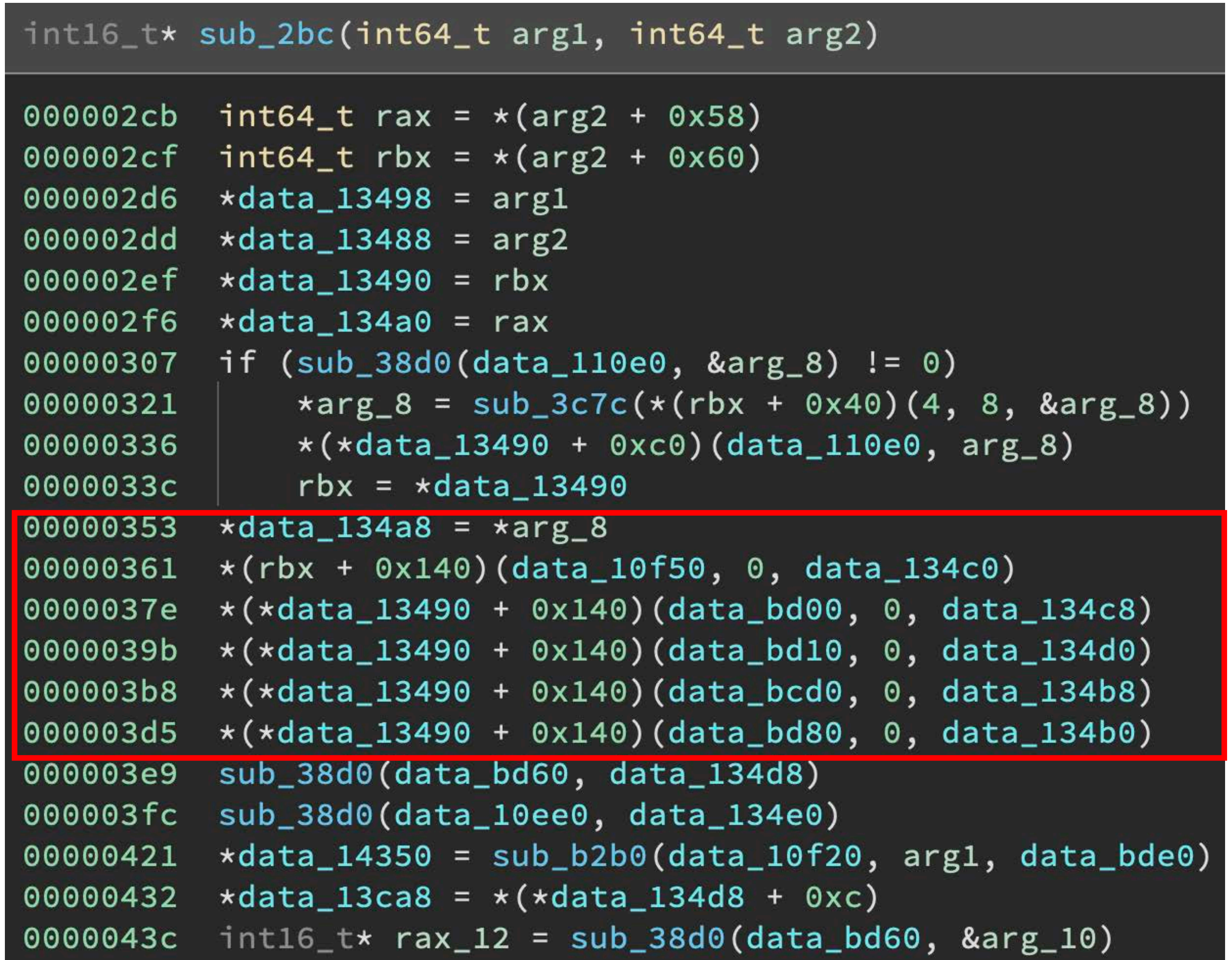
- ✓ Why have we created efiXplorer?
- ✓ Motivation
- ✓ Automated vulnerability search
  - Methodology
  - SMM Callout vuln pattern
  - GetVariable/SmmGetVariable vuln pattern
  - PPI GetVariable vuln pattern
- ✓ Final statistics
- ✓ Future Plans

BINARLY

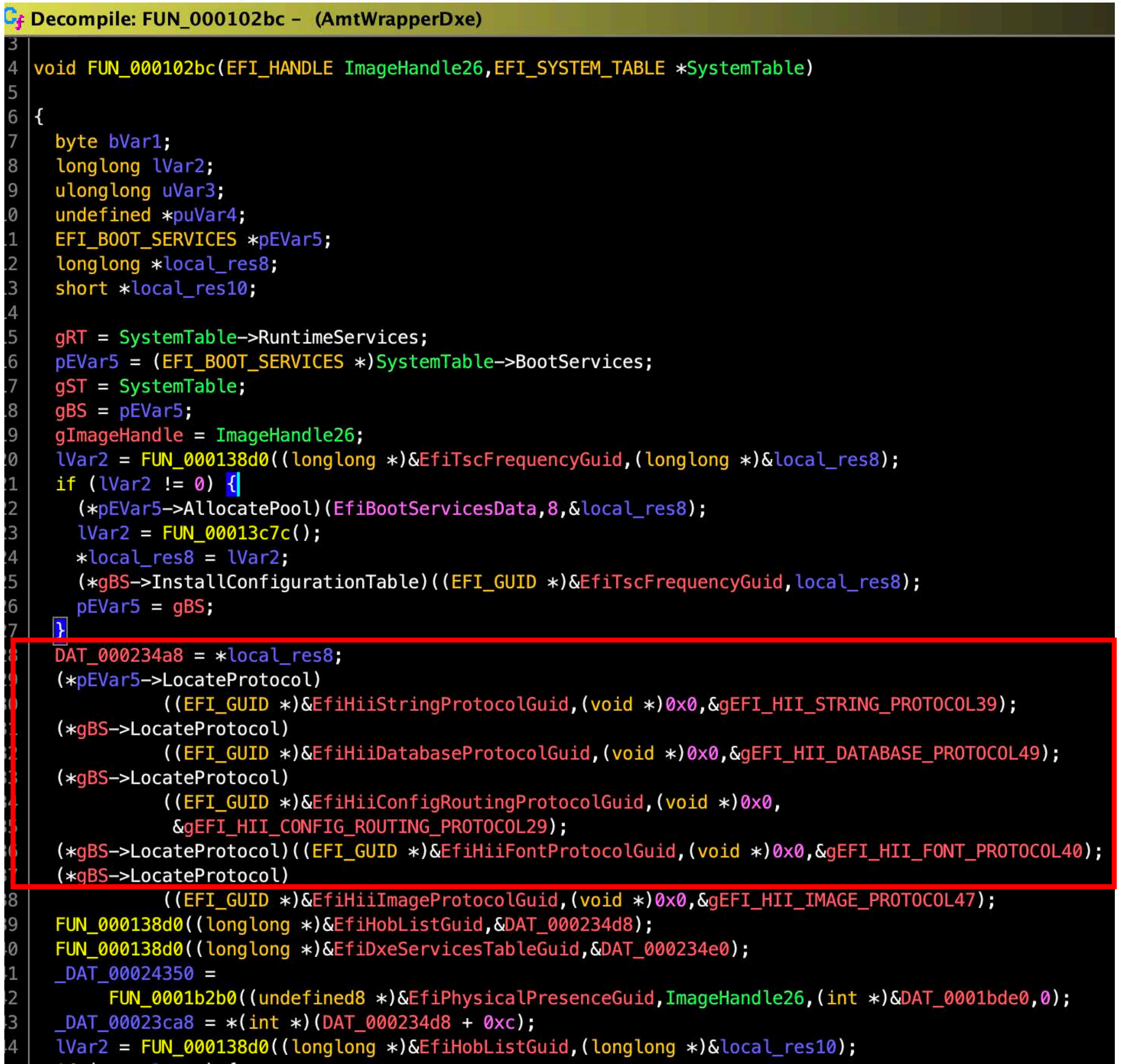
efiXplorer

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

The UEFI firmware code REconstruction limitations



```
int16_t* sub_2bc(int64_t arg1, int64_t arg2)

000002cb  int64_t rax = *(arg2 + 0x58)
000002cf  int64_t rbx = *(arg2 + 0x60)
000002d6  *data_13498 = arg1
000002dd  *data_13488 = arg2
000002ef  *data_13490 = rbx
000002f6  *data_134a0 = rax
00000307  if (sub_38d0(data_110e0, &arg_8) != 0)
00000321      *arg_8 = sub_3c7c(*(rbx + 0x40)(4, 8, &arg_8))
00000336      *(*data_13490 + 0xc0)(data_110e0, arg_8)
0000033c      rbx = *data_13490
00000353  *data_134a8 = *arg_8
00000361  *(rbx + 0x140)(data_10f50, 0, data_134c0)
0000037e  *(*data_13490 + 0x140)(data_bd00, 0, data_134c8)
0000039b  *(*data_13490 + 0x140)(data_bd10, 0, data_134d0)
000003b8  *(*data_13490 + 0x140)(data_bcd0, 0, data_134b8)
000003d5  *(*data_13490 + 0x140)(data_bd80, 0, data_134b0)
000003e9  sub_38d0(data_bd60, data_134d8)
000003fc  sub_38d0(data_10ee0, data_134e0)
00000421  *data_14350 = sub_b2b0(data_10f20, arg1, data_bde0)
00000432  *data_13ca8 = *(*data_134d8 + 0xc)
0000043c  int16_t* rax_12 = sub_38d0(data_bd60, &arg_10)
```



Decompile: FUN_000102bc - (AmtWrapperDxe)

```

void FUN_000102bc(EFI_HANDLE ImageHandle26,EFI_SYSTEM_TABLE *SystemTable)

{
  byte bVar1;
  longlong lVar2;
  ulonglong uVar3;
  undefined *puVar4;
  EFI_BOOT_SERVICES *pEVar5;
  longlong *local_res8;
  short *local_res10;

  gRT = SystemTable->RuntimeServices;
  pEVar5 = (EFI_BOOT_SERVICES *)SystemTable->BootServices;
  gST = SystemTable;
  gBS = pEVar5;
  gImageHandle = ImageHandle26;
  lVar2 = FUN_000138d0((longlong *)&EfiTscFrequencyGuid,(longlong *)&local_res8);
  if (lVar2 != 0) {
    (*pEVar5->AllocatePool)(EfiBootServicesData,8,&local_res8);
    lVar2 = FUN_00013c7c();
    *local_res8 = lVar2;
    (*gBS->InstallConfigurationTable)((EFI_GUID *)&EfiTscFrequencyGuid,local_res8);
    pEVar5 = gBS;
  }
  DAT_000234a8 = *local_res8;
  (*pEVar5->LocateProtocol)
            ((EFI_GUID *)&EfiHiiStringProtocolGuid,(void *)0x0,&gEFI_HII_STRING_PROTOCOL39);
  (*gBS->LocateProtocol)
            ((EFI_GUID *)&EfiHiiDatabaseProtocolGuid,(void *)0x0,&gEFI_HII_DATABASE_PROTOCOL49);
  (*gBS->LocateProtocol)
            ((EFI_GUID *)&EfiHiiConfigRoutingProtocolGuid,(void *)0x0,
             &gEFI_HII_CONFIG_ROUTING_PROTOCOL29);
  (*gBS->LocateProtocol)((EFI_GUID *)&EfiHiiFontProtocolGuid,(void *)0x0,&gEFI_HII_FONT_PROTOCOL40);
  (*gBS->LocateProtocol)
            ((EFI_GUID *)&EfiHiiImageProtocolGuid,(void *)0x0,&gEFI_HII_IMAGE_PROTOCOL47);
  FUN_000138d0((longlong *)&EfiHobListGuid,&DAT_000234d8);
  FUN_000138d0((longlong *)&EfiDxeServicesTableGuid,&DAT_000234e0);
  _DAT_00024350 =
       FUN_0001b2b0((undefined8 *)&EfiPhysicalPresenceGuid,ImageHandle26,(int *)&DAT_0001bde0,0);
  _DAT_00023ca8 = *(int *)(DAT_000234d8 + 0xc);
  lVar2 = FUN_000138d0((longlong *)&EfiHobListGuid,(longlong *)&local_res10);
```

Why we work on efiXplorer?

- ✓ Simplifying Reconstruction of UEFI-specific types and protocols

  - efiXplorer->efiAnalyzer

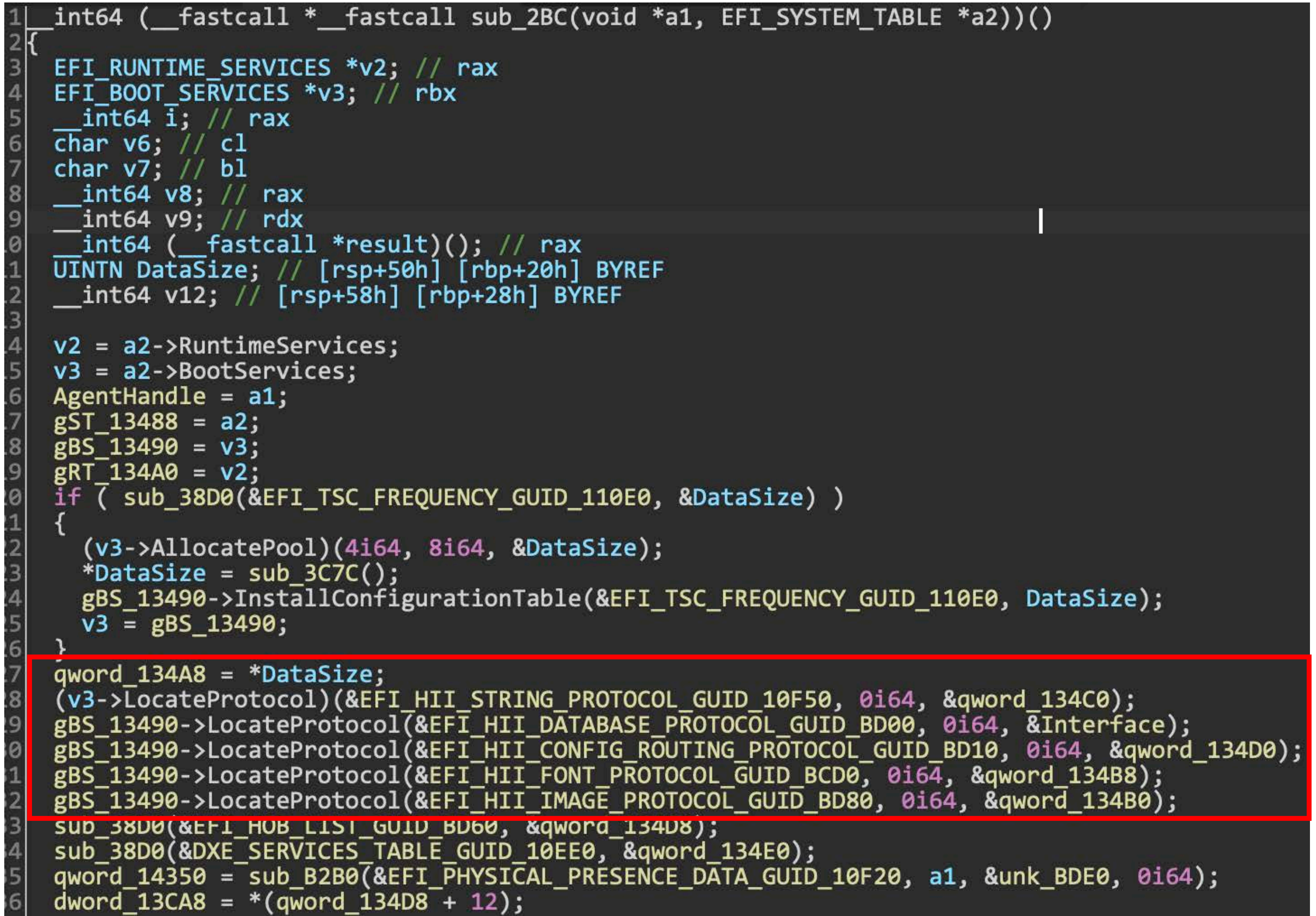
- ✓ Creating a unified loader for whole UEFI firmware image with rebuilt dependencies and cross-references between different DXE and PEI modules

  - efiXplorer->efiLoader

- ✓ Finding common types of vulnerabilities with UEFI specifics and power of static analysis

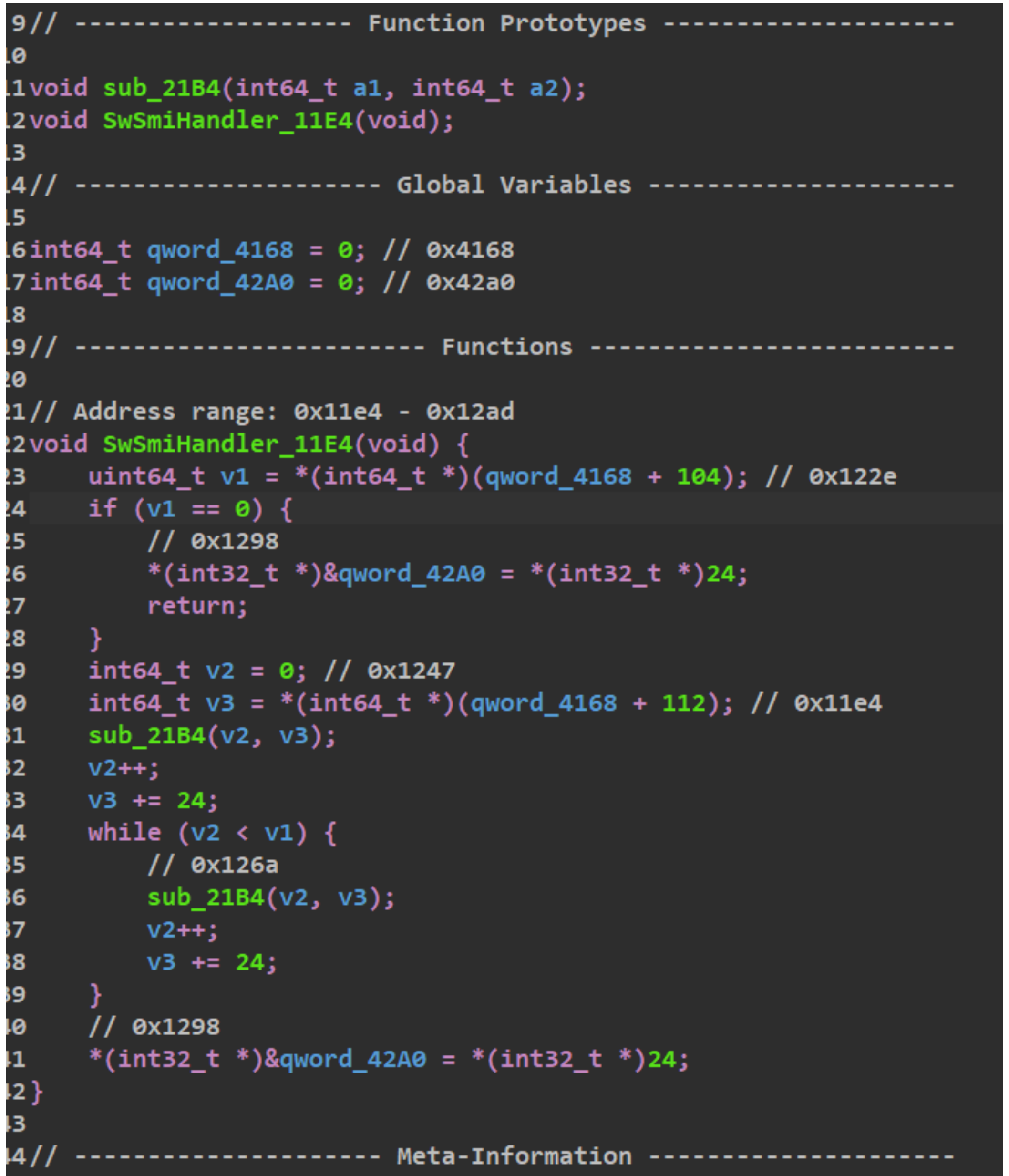
  - efiXplorer->efiAnalyzer->efiVulnHunt

Hex-Rays + efiXplorer

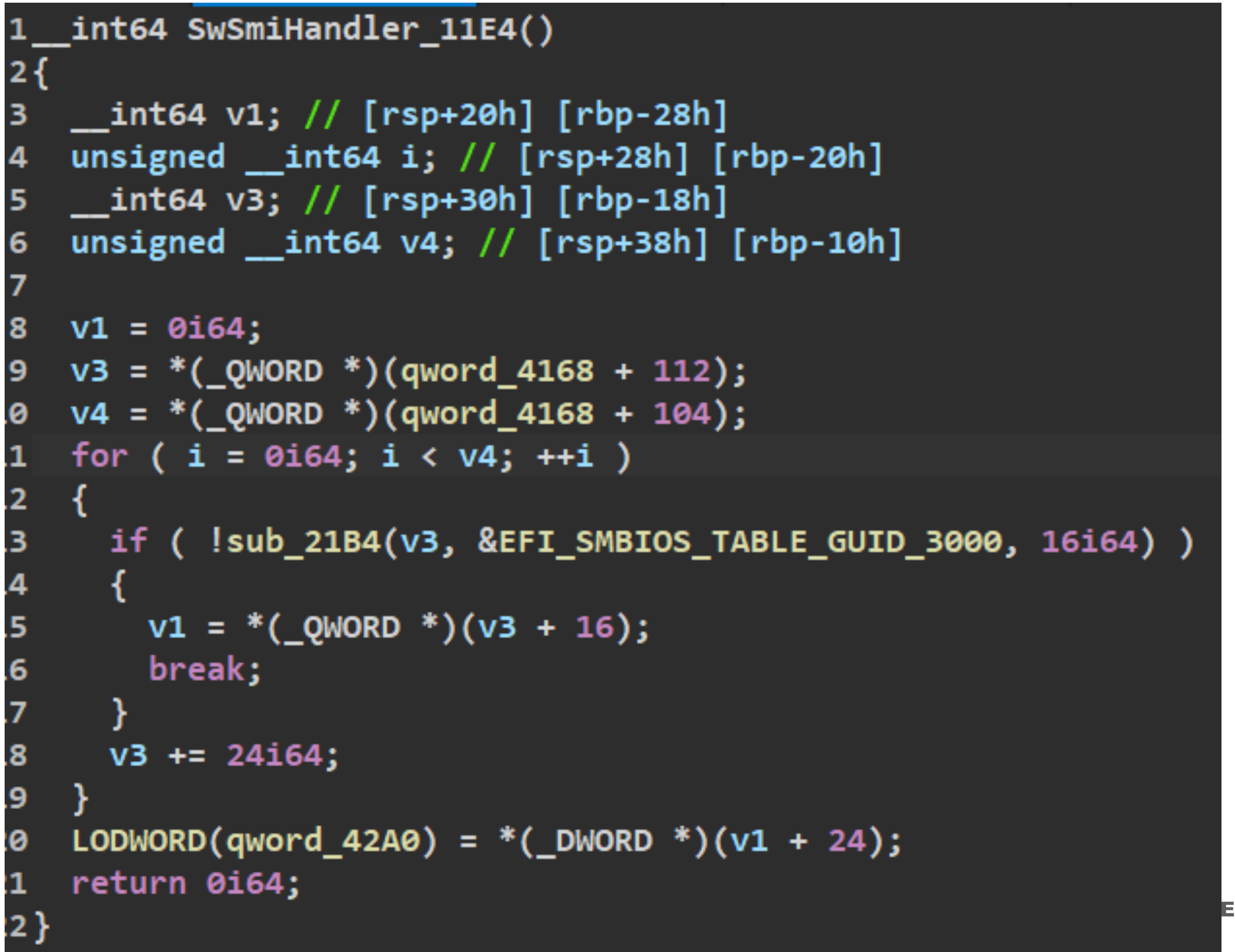


```
__int64 (__fastcall *__fastcall sub_2BC(void *a1, EFI_SYSTEM_TABLE *a2))()
{
  EFI_RUNTIME_SERVICES *v2; // rax
  EFI_BOOT_SERVICES *v3; // rbx
  __int64 i; // rax
  char v6; // cl
  char v7; // bl
  __int64 v8; // rax
  __int64 v9; // rdx
  __int64 (__fastcall *result)(); // rax
  UINTN DataSize; // [rsp+50h] [rbp+20h] BYREF
  __int64 v12; // [rsp+58h] [rbp+28h] BYREF

  v2 = a2->RuntimeServices;
  v3 = a2->BootServices;
  AgentHandle = a1;
  gST_13488 = a2;
  gBS_13490 = v3;
  gRT_134A0 = v2;
  if ( sub_38D0(&EFI_TSC_FREQUENCY_GUID_110E0, &DataSize) )
  {
    (v3->AllocatePool)(4i64, 8i64, &DataSize);
    *DataSize = sub_3C7C();
    gBS_13490->InstallConfigurationTable(&EFI_TSC_FREQUENCY_GUID_110E0, DataSize);
    v3 = gBS_13490;
  }
  qword_134A8 = *DataSize;
  (v3->LocateProtocol)(&EFI_HII_STRING_PROTOCOL_GUID_10F50, 0i64, &qword_134C0);
  gBS_13490->LocateProtocol(&EFI_HII_DATABASE_PROTOCOL_GUID_BD00, 0i64, &Interface);
  gBS_13490->LocateProtocol(&EFI_HII_CONFIG_ROUTING_PROTOCOL_GUID_BD10, 0i64, &qword_134D0);
  gBS_13490->LocateProtocol(&EFI_HII_FONT_PROTOCOL_GUID_BCD0, 0i64, &qword_134B8);
  gBS_13490->LocateProtocol(&EFI_HII_IMAGE_PROTOCOL_GUID_BD80, 0i64, &qword_134B0);
  sub_38D0(&EFI_HOB_LIST_GUID_BD60, &qword_134D8);
  sub_38D0(&DXE_SERVICES_TABLE_GUID_10EE0, &qword_134E0);
  qword_14350 = sub_B2B0(&EFI_PHYSICAL_PRESENCE_DATA_GUID_10F20, a1, &unk_BDE0, 0i64);
  dword_13CA8 = *(qword_134D8 + 12);
```

efiXloader: SMI handlers identification



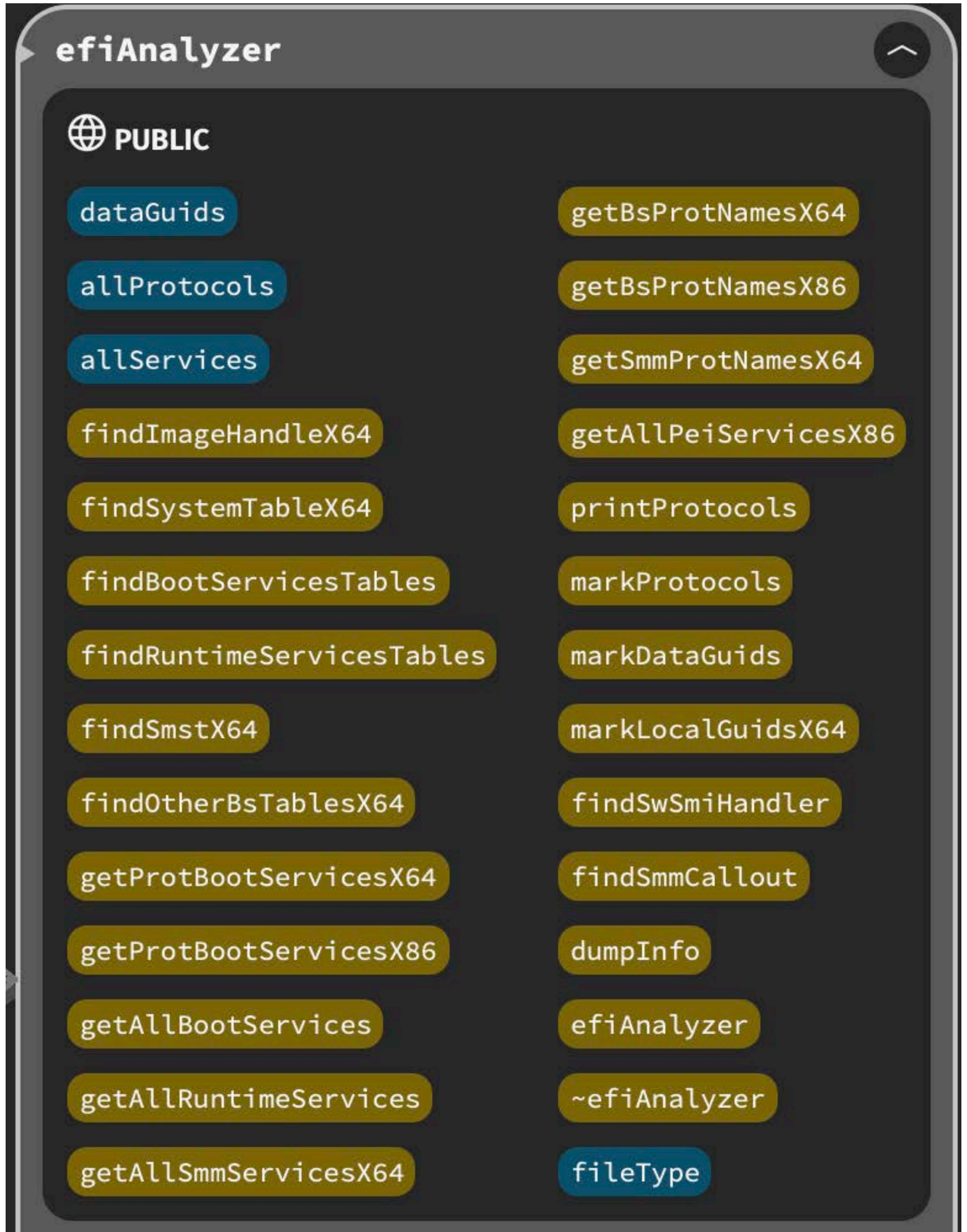
```
// ------------------- Function Prototypes -------------------

void sub_21B4(int64_t a1, int64_t a2);
void SwSmiHandler_11E4(void);

// -------------------- Global Variables --------------------

int64_t qword_4168 = 0; // 0x4168
int64_t qword_42A0 = 0; // 0x42a0

// ----------------------- Functions ------------------------

// Address range: 0x11e4 - 0x12ad
void SwSmiHandler_11E4(void) {
    uint64_t v1 = *(int64_t *)(qword_4168 + 104); // 0x122e
    if (v1 == 0) {
        // 0x1298
        *(int32_t *)&qword_42A0 = *(int32_t *)24;
        return;
    }
    int64_t v2 = 0; // 0x1247
    int64_t v3 = *(int64_t *)(qword_4168 + 112); // 0x11e4
    sub_21B4(v2, v3);
    v2++;
    v3 += 24;
    while (v2 < v1) {
        // 0x126a
        sub_21B4(v2, v3);
        v2++;
        v3 += 24;
    }
    // 0x1298
    *(int32_t *)&qword_42A0 = *(int32_t *)24;
}

// -------------------- Meta-Information --------------------
```



```
__int64 SwSmiHandler_11E4()
{
  __int64 v1; // [rsp+20h] [rbp-28h]
  unsigned __int64 i; // [rsp+28h] [rbp-20h]
  __int64 v3; // [rsp+30h] [rbp-18h]
  unsigned __int64 v4; // [rsp+38h] [rbp-10h]

  v1 = 0i64;
  v3 = *(_QWORD *)(qword_4168 + 112);
  v4 = *(_QWORD *)(qword_4168 + 104);
  for ( i = 0i64; i < v4; ++i )
  {
    if ( !sub_21B4(v3, &EFI_SMBIOS_TABLE_GUID_3000, 16i64) )
    {
      v1 = *(_QWORD *)(v3 + 16);
      break;
    }
    v3 += 24i64;
  }
  LODWORD(qword_42A0) = *(_DWORD *)(v1 + 24);
  return 0i64;
}
```

black hat
EUROPE 2020

How it started, and how it's going?

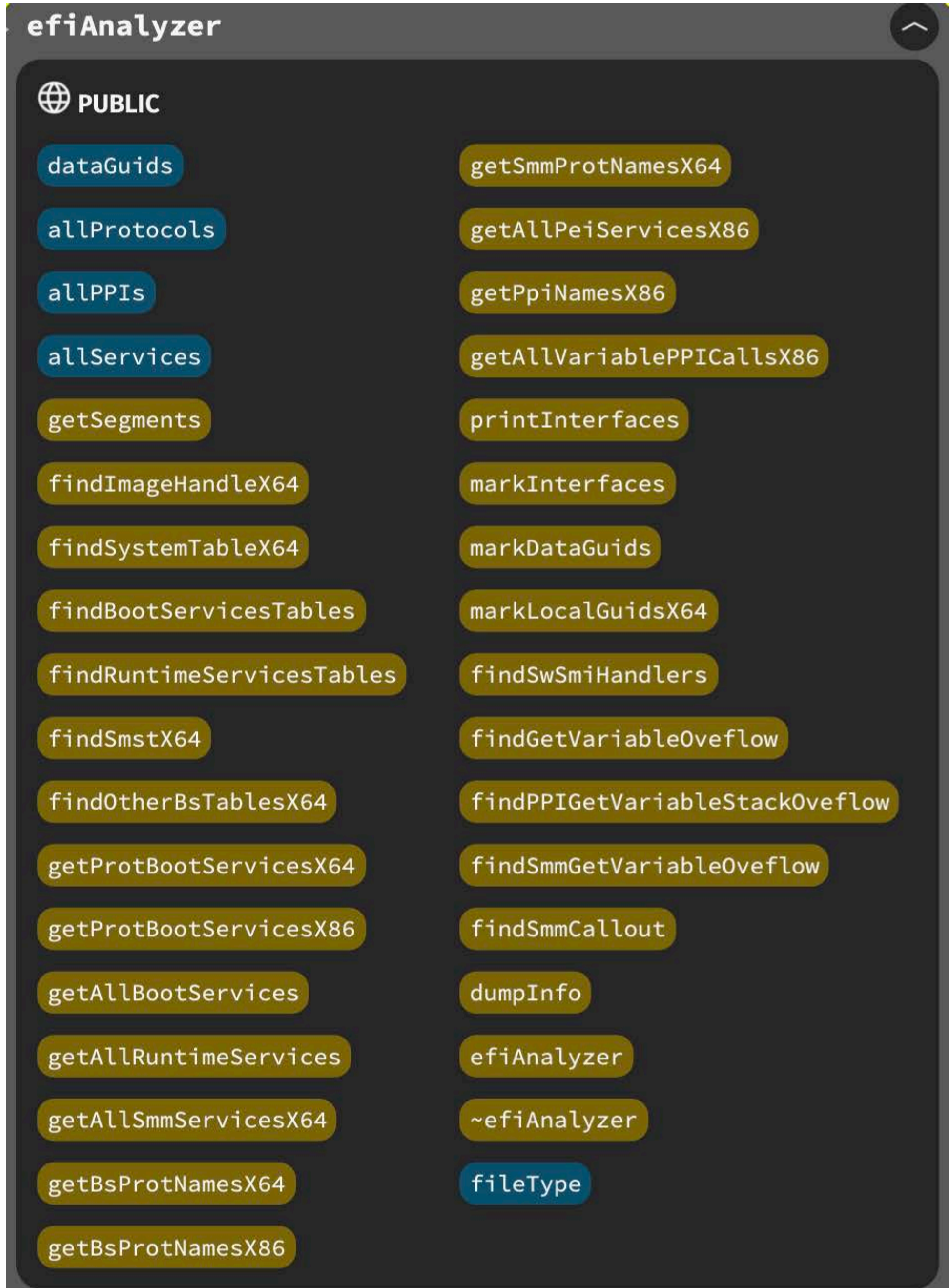


efiAnalyzer

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



efiAnalyzer

PUBLIC

dataGuids
allProtocols
allPPIs
allServices
getSegments
findImageHandleX64
findSystemTableX64
findBootServicesTables
findRuntimeServicesTables
findSmstX64
findOtherBsTablesX64
getProtBootServicesX64
getProtBootServicesX86
getAllBootServices
getAllRuntimeServices
getAllSmmServicesX64
getBsProtNamesX64
getBsProtNamesX86

getSmmProtNamesX64
getAllPeiServicesX86
getPpiNamesX86
getAllVariablePPICallsX86
printInterfaces
markInterfaces
markDataGuids
markLocalGuidsX64
findSwSmiHandlers
findGetVariableOveflow
findPPIGetVariableStackOveflow
findSmmGetVariableOveflow
findSmmCallout
dumpInfo
efiAnalyzer
~efiAnalyzer
fileType

Hex-Rays

STATE-OF-THE-ART BINARY CODE ANALYSIS SOLUTIONS

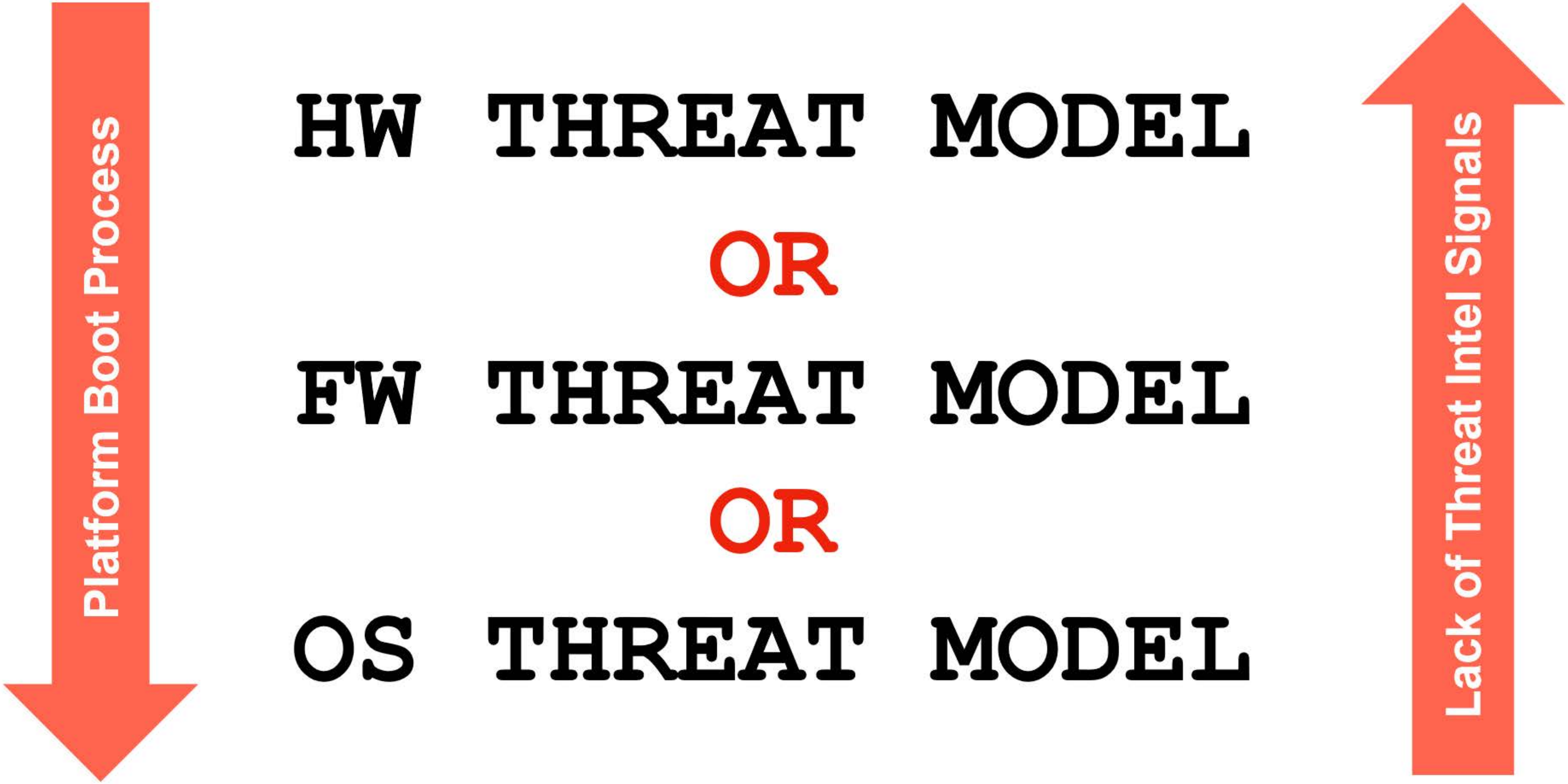
First prize DynDataResolver

Second prize Lucid and grap

Third prize efiXplorer

https://www.hex-rays.com/contests_details/contest2020/

Motivation of this REsearch



Platform Boot Process

HW THREAT MODEL

OR

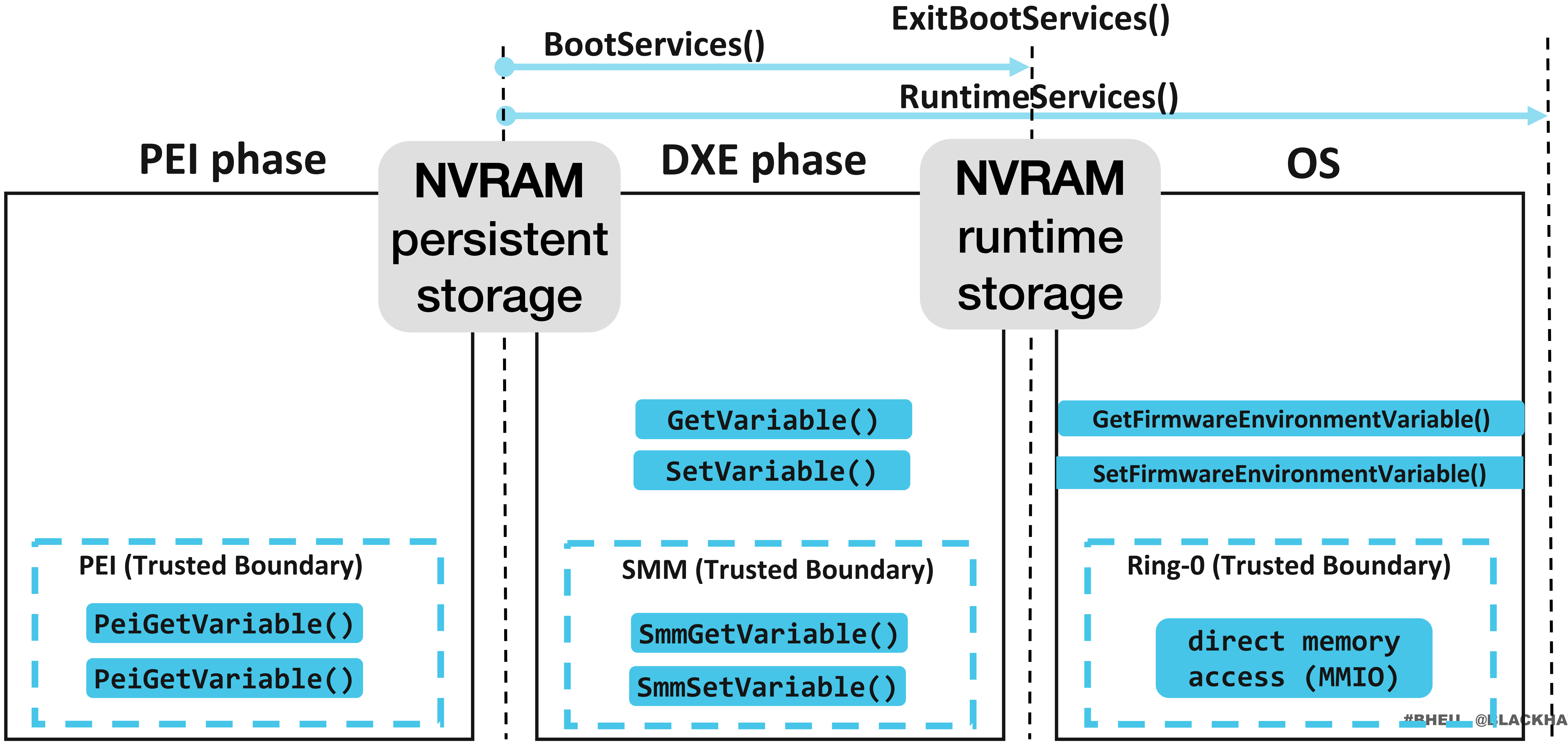
FW THREAT MODEL

OR

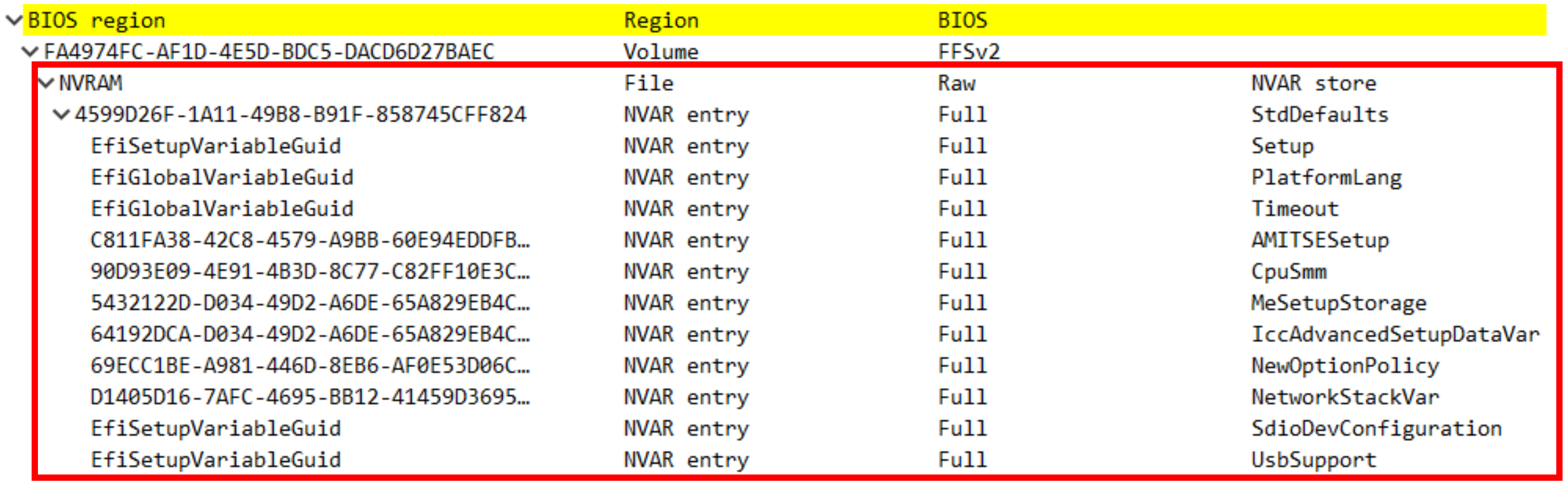
OS THREAT MODEL

Lack of Threat Intel Signals

NVRAM Variables access during Boot Flow



NVRAM persistence on SPI flash

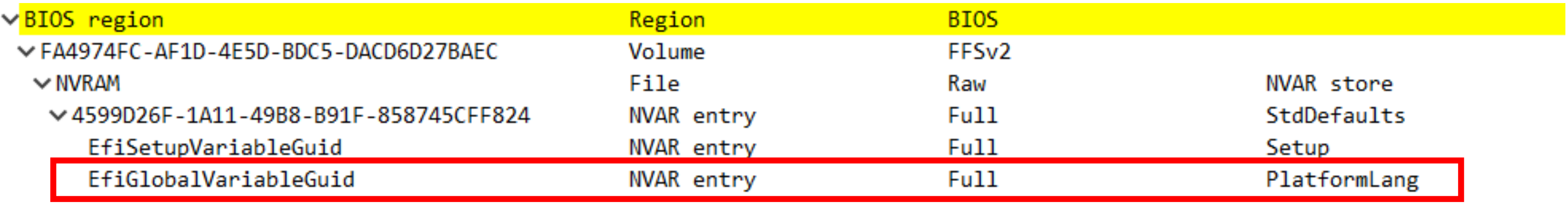


BIOS region	Region	BIOS	
FA4974FC-AF1D-4E5D-BDC5-DACD6D27BAEC	Volume	FFSv2	
NVRAM	File	Raw	NVAR store
4599D26F-1A11-49B8-B91F-858745CFF824	NVAR entry	Full	StdDefaults
EfiSetupVariableGuid	NVAR entry	Full	Setup
EfiGlobalVariableGuid	NVAR entry	Full	PlatformLang
EfiGlobalVariableGuid	NVAR entry	Full	Timeout
C811FA38-42C8-4579-A9BB-60E94EDDFB…	NVAR entry	Full	AMITSESetup
90D93E09-4E91-4B3D-8C77-C82FF10E3C…	NVAR entry	Full	CpuSmm
5432122D-D034-49D2-A6DE-65A829EB4C…	NVAR entry	Full	MeSetupStorage
64192DCA-D034-49D2-A6DE-65A829EB4C…	NVAR entry	Full	IccAdvancedSetupDataVar
69ECC1BE-A981-446D-8EB6-AF0E53D06C…	NVAR entry	Full	NewOptionPolicy
D1405D16-7AFC-4695-BB12-41459D3695…	NVAR entry	Full	NetworkStackVar
EfiSetupVariableGuid	NVAR entry	Full	SdioDevConfiguration
EfiSetupVariableGuid	NVAR entry	Full	UsbSupport

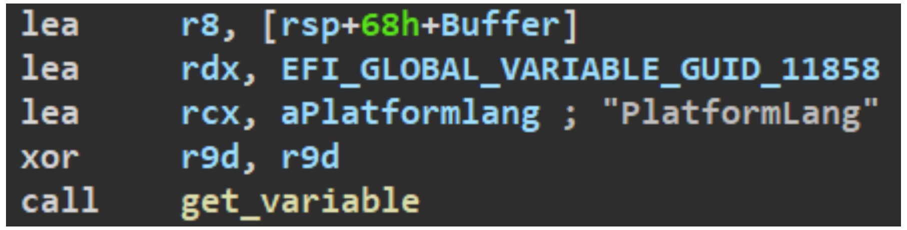
- ✓ **NVRAM region is not protected by Intel Boot Guard and can be abused by attacker with physical access (supply chain vector).**
- ✓ **Arbitrary code execution via *GetVariable()* is common, attacker can modify persistent NVRAM storage and install fileless DXE/SMM/PEI implant (shellcode).**

Most security solutions inspect only UEFI drivers!

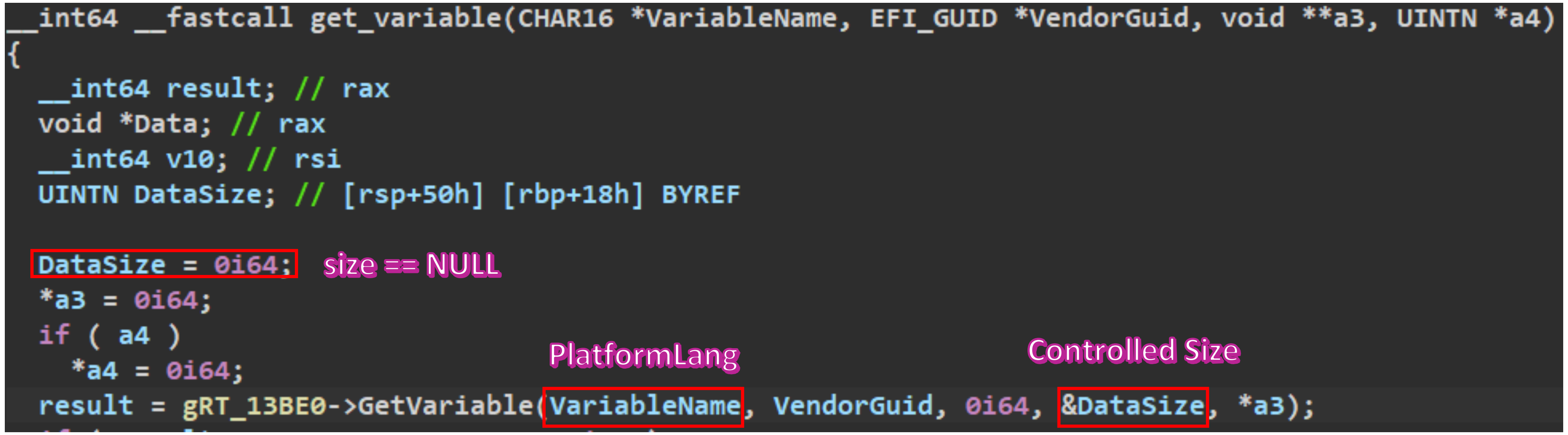
NVRAM persistence on SPI flash



```
∨ BIOS region                                  Region        BIOS
  ∨ FA4974FC-AF1D-4E5D-BDC5-DACD6D27BAEC       Volume        FFSv2
    ∨ NVRAM                                    File          Raw           NVAR store
      ∨ 4599D26F-1A11-49B8-B91F-858745CFF824   NVAR entry    Full          StdDefaults
          EfiSetupVariableGuid                 NVAR entry    Full          Setup
          EfiGlobalVariableGuid                NVAR entry    Full          PlatformLang
```



```
lea     r8, [rsp+68h+Buffer]
lea     rdx, EFI_GLOBAL_VARIABLE_GUID_11858
lea     rcx, aPlatformlang ; "PlatformLang"
xor     r9d, r9d
call    get_variable
```



```
__int64 __fastcall get_variable(CHAR16 *VariableName, EFI_GUID *VendorGuid, void **a3, UINTN *a4)
{
  __int64 result; // rax
  void *Data; // rax
  __int64 v10; // rsi
  UINTN DataSize; // [rsp+50h] [rbp+18h] BYREF

  DataSize = 0i64;
  *a3 = 0i64;
  if ( a4 )
    *a4 = 0i64;
  result = gRT_13BE0->GetVariable(VariableName, VendorGuid, 0i64, &DataSize, *a3);
```

size == NULL

PlatformLang

Controlled Size

NVRAM persistence: previous work

✓ Linux NVRAM runtime persistence (not SPI storage)

https://media.defcon.org/DEF%20CON%2027/DEF%20CON%2027%20presentations/DEFCON-27-Michael-Leibowitz-and-Topher-Timzen-EDR-Is-Coming-Hide-Yo-Sht.pdf

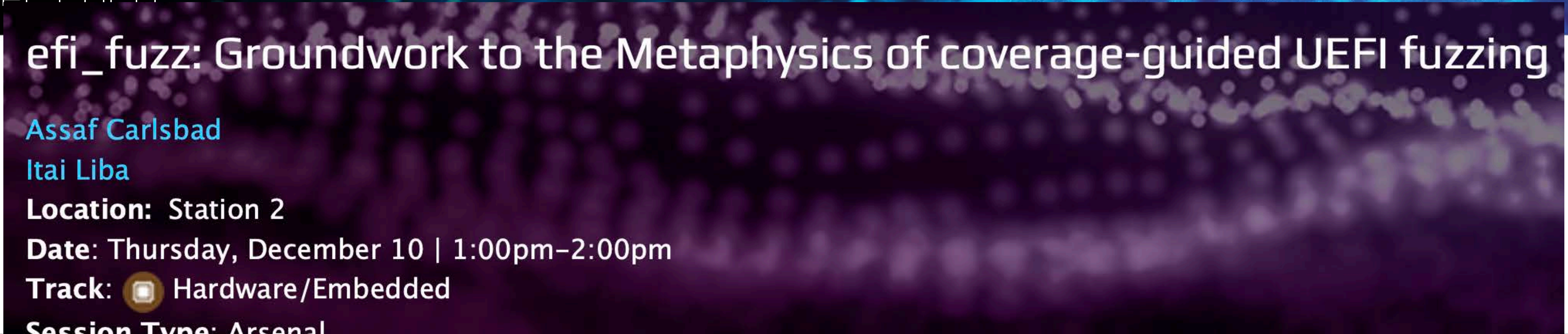
https://github.com/perturbed-platypus/LinooxMalware

✓ MS Win NVRAM runtime persistence (not SPI storage)

https://slaeryan.github.io/posts/midnighttrain.html

https://github.com/slaeryan/MIDNIGHTTRAIN

* NVRAM persistent storage (with physical access to the target machine) also mentioned in CIA Vault7 leak



efi_fuzz: Groundwork to the Metaphysics of coverage-guided UEFI fuzzing

Assaf Carlsbad

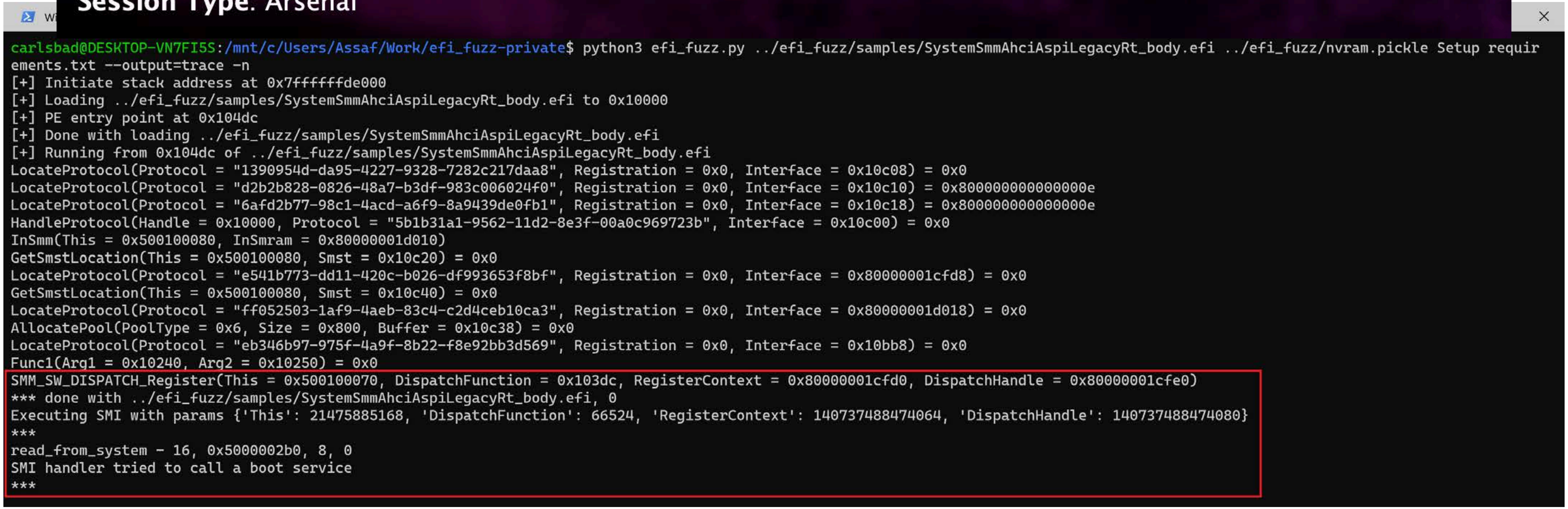
Itai Liba

Location: Station 2

Date: Thursday, December 10 | 1:00pm–2:00pm

Track: Hardware/Embedded

Session Type: Arsenal



```
carlsbad@DESKTOP-VN7FI5S:/mnt/c/Users/Assaf/Work/efi_fuzz-private$ python3 efi_fuzz.py ../efi_fuzz/samples/SystemSmmAhciAspiLegacyRt_body.efi ../efi_fuzz/nvram.pickle Setup requir
ements.txt --output=trace -n
[+] Initiate stack address at 0x7ffffffde000
[+] Loading ../efi_fuzz/samples/SystemSmmAhciAspiLegacyRt_body.efi to 0x10000
[+] PE entry point at 0x104dc
[+] Done with loading ../efi_fuzz/samples/SystemSmmAhciAspiLegacyRt_body.efi
[+] Running from 0x104dc of ../efi_fuzz/samples/SystemSmmAhciAspiLegacyRt_body.efi
LocateProtocol(Protocol = "1390954d-da95-4227-9328-7282c217daa8", Registration = 0x0, Interface = 0x10c08) = 0x0
LocateProtocol(Protocol = "d2b2b828-0826-48a7-b3df-983c006024f0", Registration = 0x0, Interface = 0x10c10) = 0x800000000000000e
LocateProtocol(Protocol = "6afd2b77-98c1-4acd-a6f9-8a9439de0fb1", Registration = 0x0, Interface = 0x10c18) = 0x800000000000000e
HandleProtocol(Handle = 0x10000, Protocol = "5b1b31a1-9562-11d2-8e3f-00a0c969723b", Interface = 0x10c00) = 0x0
InSmm(This = 0x500100080, InSmram = 0x80000001d010)
GetSmstLocation(This = 0x500100080, Smst = 0x10c20) = 0x0
LocateProtocol(Protocol = "e541b773-dd11-420c-b026-df993653f8bf", Registration = 0x0, Interface = 0x80000001cfd8) = 0x0
GetSmstLocation(This = 0x500100080, Smst = 0x10c40) = 0x0
LocateProtocol(Protocol = "ff052503-1af9-4aeb-83c4-c2d4ceb10ca3", Registration = 0x0, Interface = 0x80000001d018) = 0x0
AllocatePool(PoolType = 0x6, Size = 0x800, Buffer = 0x10c38) = 0x0
LocateProtocol(Protocol = "eb346b97-975f-4a9f-8b22-f8e92bb3d569", Registration = 0x0, Interface = 0x10bb8) = 0x0
Func1(Arg1 = 0x10240, Arg2 = 0x10250) = 0x0
SMM_SW_DISPATCH_Register(This = 0x500100070, DispatchFunction = 0x103dc, RegisterContext = 0x80000001cfd0, DispatchHandle = 0x80000001cfe0)
*** done with ../efi_fuzz/samples/SystemSmmAhciAspiLegacyRt_body.efi, 0
Executing SMI with params {'This': 21475885168, 'DispatchFunction': 66524, 'RegisterContext': 140737488474064, 'DispatchHandle': 140737488474080}
***
read_from_system - 16, 0x5000002b0, 8, 0
SMI handler tried to call a boot service
***
```

https://labs.sentinelone.com/moving-from-dynamic-emulation-of-uefi-modules-to-coverage-guided-fuzzing-of-uefi-firmware/

Limitations of blackbox AFL fuzzing

- ✓ Lack of code-coverage-based feedback loop means test generation can rely only static corpus.
- ✓ Random input mutations with little initial knowledge may need extra RE work to create more precise/valid corpus
- ✓ Platform simulation like Simics with combination of Symbolic Execution* can improve input corpus generation and test coverage in general.
- ✓ **efiXplorer** can also fill that gap by providing the coverage and helping with corpus generation for potential targets.

* https://software.intel.com/content/www/us/en/develop/articles/finding-bios-vulnerabilities-with-symbolic-execution-and-virtual-platforms.html

Vendors disclosure Details

Intel/Dell Timeline (discovered by Nvidia Offensive Research):

- **Sep 2020:** Initial Disclosure
- **Oct 2020:** Issues confirmed

  GetVariable() – 2 stack overflow issues with **SMM code execution impact**

  SmmGetVariable() – 2 stack overflow issues with **SMM code execution impact**

  CommBuffer – 1 heap overflow issue with **SMM code execution impact**
- **Nov 2020:** Security fixes confirmed in update cycle
- **April 2020:** Disclosure date 🍿

efiXplorer Automated vulnerability search at scale

Automated vulnerability search methodology

We used 3 datasets with firmware images only released in 2020:

- ✓ **ASRock** - 450 firmware images
- ✓ **ASUS** - 820 firmware images
- ✓ **Lenovo** - 84 firmware images

Automated vulnerability search methodology

We evaluated efiXplorer at automated vulnerability search in three ways:

- ✓ **Measuring objects and structures recovery**
    - ✓ Function calls recovery precision 0.94 / recall 0.88 (at DXE stage)
    - ✓ For more info: https://github.com/binarly-io/Research_Publications/tree/main/EKO_2020

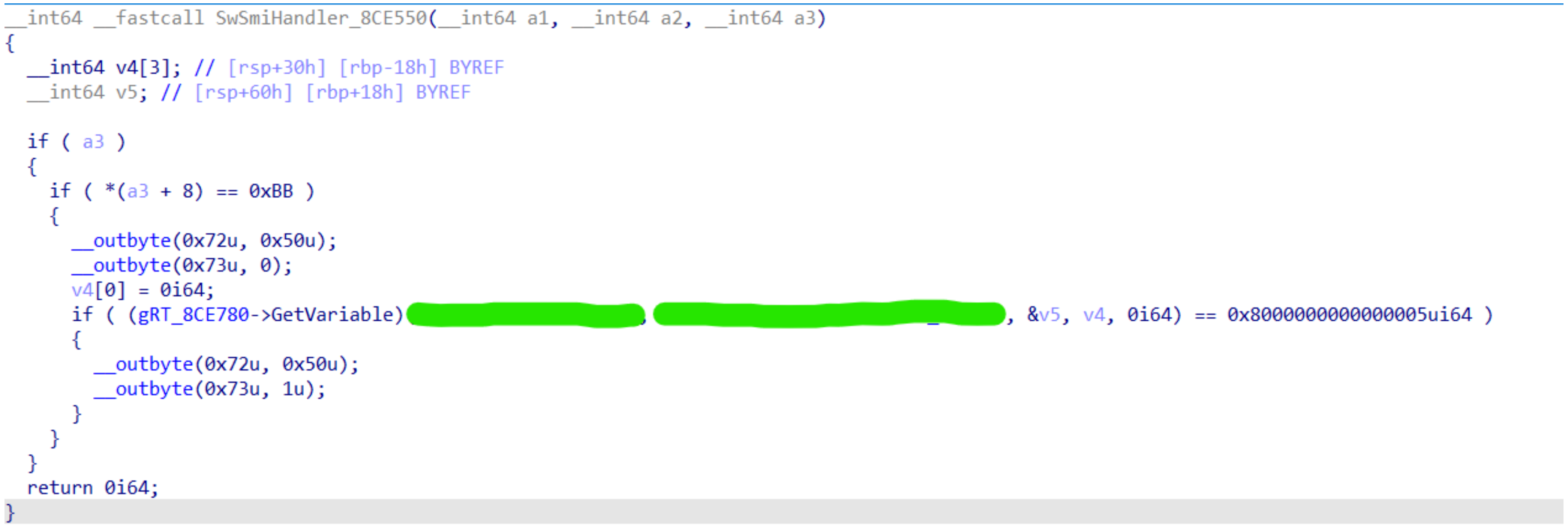
- ✓ **Measuring attack surface: number of SMI handlers and GetVariable calls**

- ✓ **Running automated vulnerability checks and validating results semi-manually**

efiXplorer
SMI callout automated search

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
- 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], [redacted], &v5, v4, 0i64) == 0x8000000000000005ui64 )
      {
        __outbyte(0x72u, 0x50u);
        __outbyte(0x73u, 1u);
      }
    }
  }
  return 0i64;
}
```

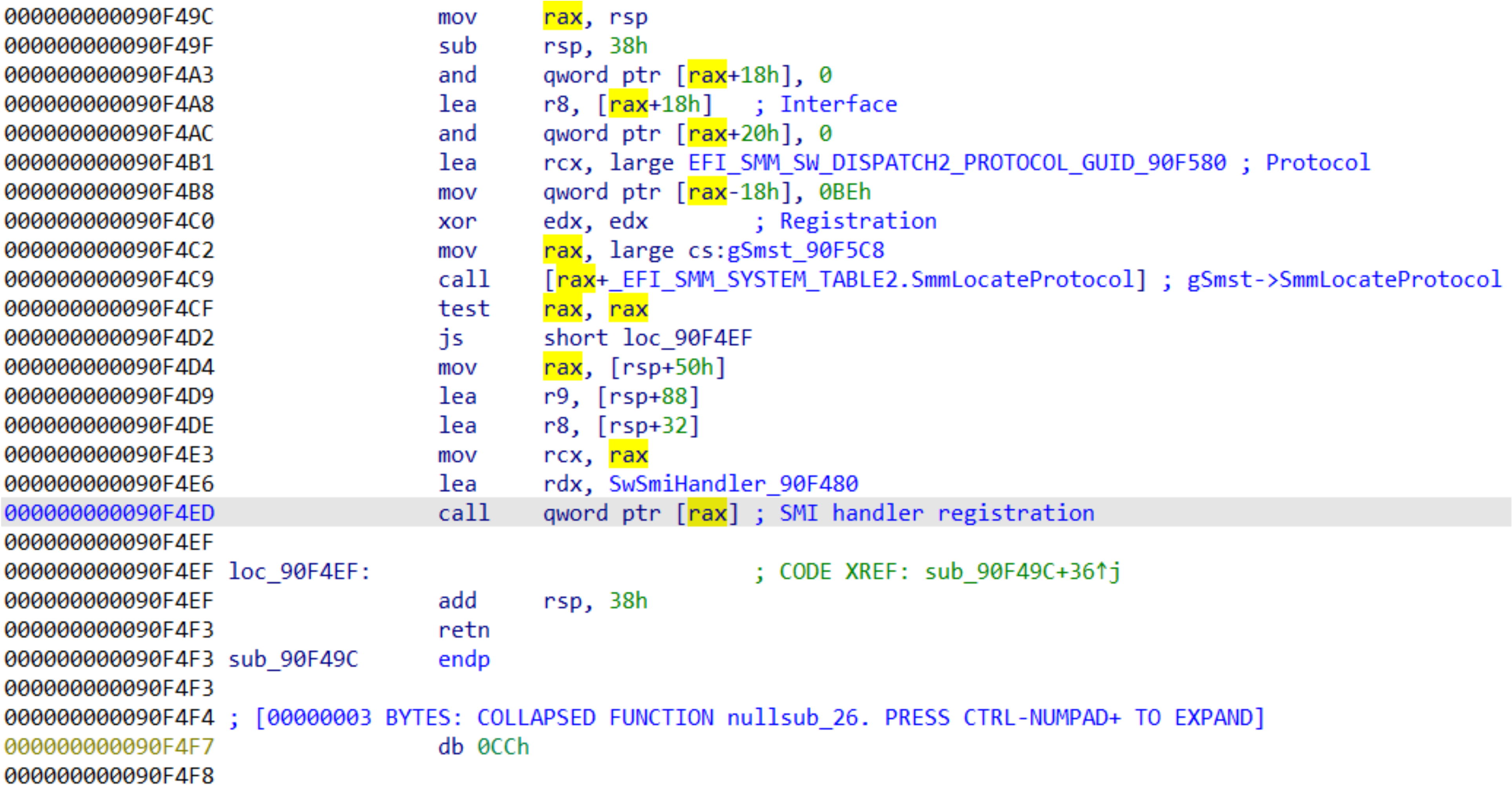
efiXloader: SMM callouts identification

- 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 inside SMM

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

efiXloader: SMM callouts identification

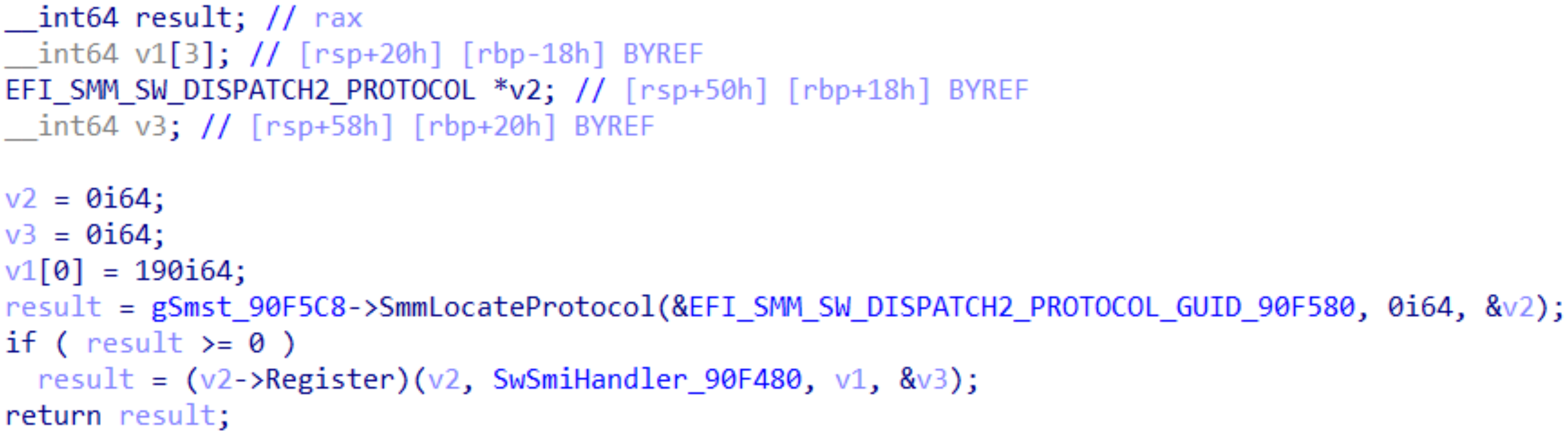
- Iterate through EFI_SMM_SW_DISPATCH2_PROTOCOL.Register() within each SMM driver and collect pointer to SMI handler



```
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
```

efiXloader: SMM callouts identification

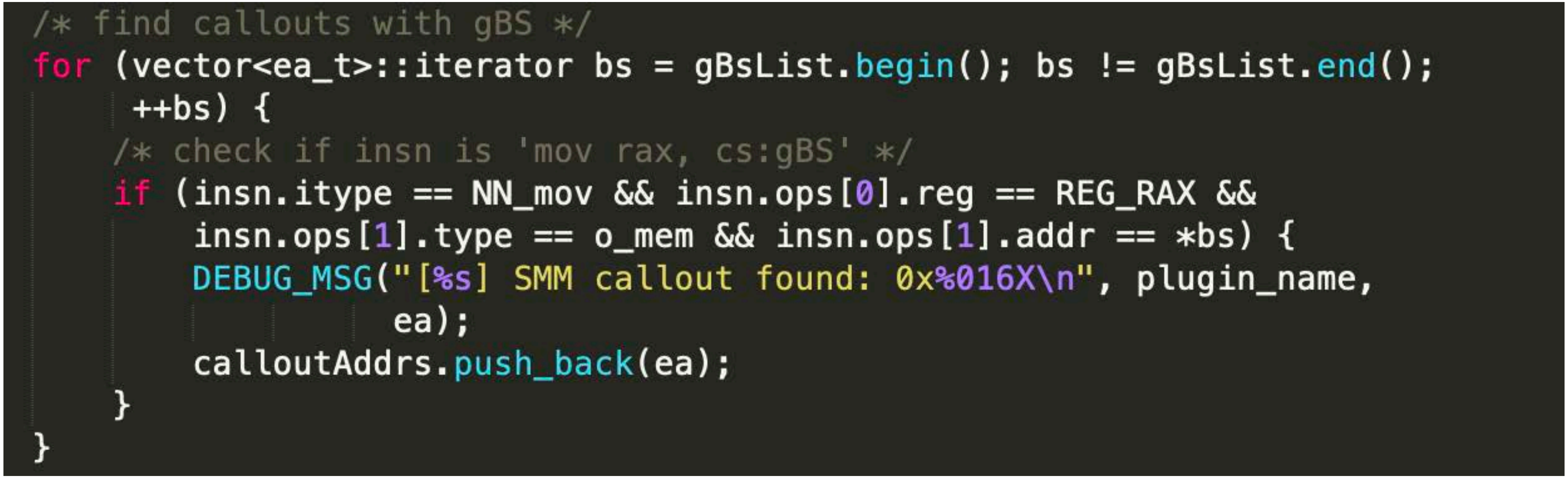
- **Iterate through EFI_SMM_SW_DISPATCH2_PROTOCOL.Register() within each SMM driver and collect pointer to SMI handler**



```
__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;
```

efiXloader: SMM callouts identification

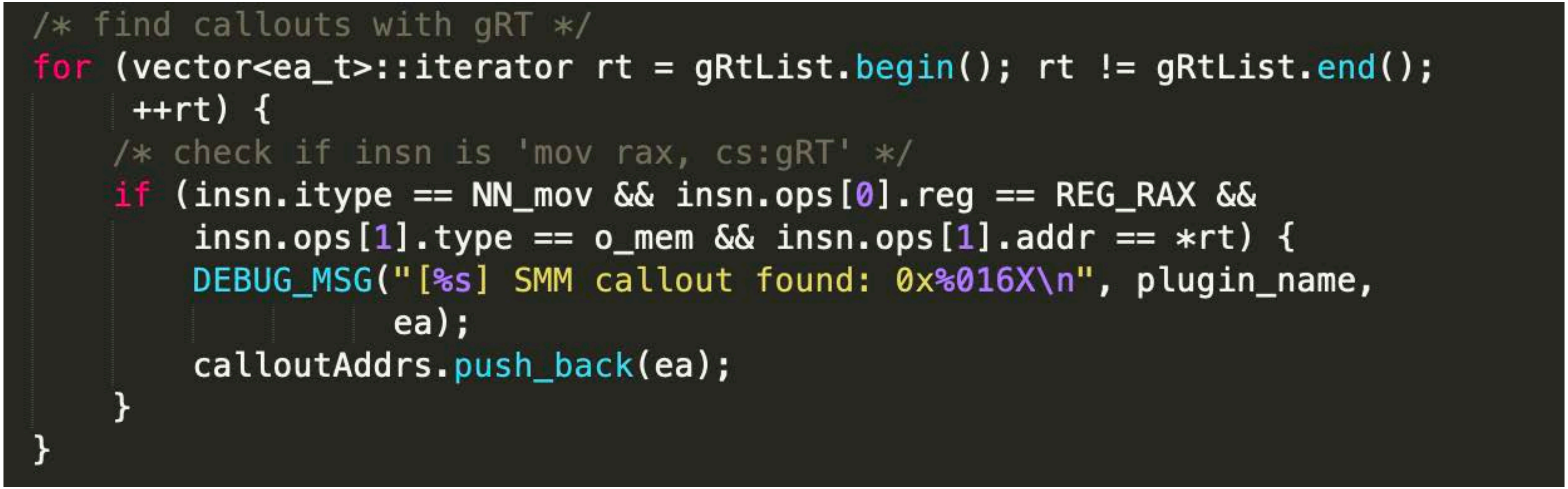
- BootServices



```
/* find callouts with gBS */
for (vector<ea_t>::iterator bs = gBsList.begin(); bs != gBsList.end();
     ++bs) {
    /* check if insn is 'mov rax, cs:gBS' */
    if (insn.itype == NN_mov && insn.ops[0].reg == REG_RAX &&
        insn.ops[1].type == o_mem && insn.ops[1].addr == *bs) {
        DEBUG_MSG("[%s] SMM callout found: 0x%016X\n", plugin_name,
                  ea);
        calloutAddrs.push_back(ea);
    }
}
```

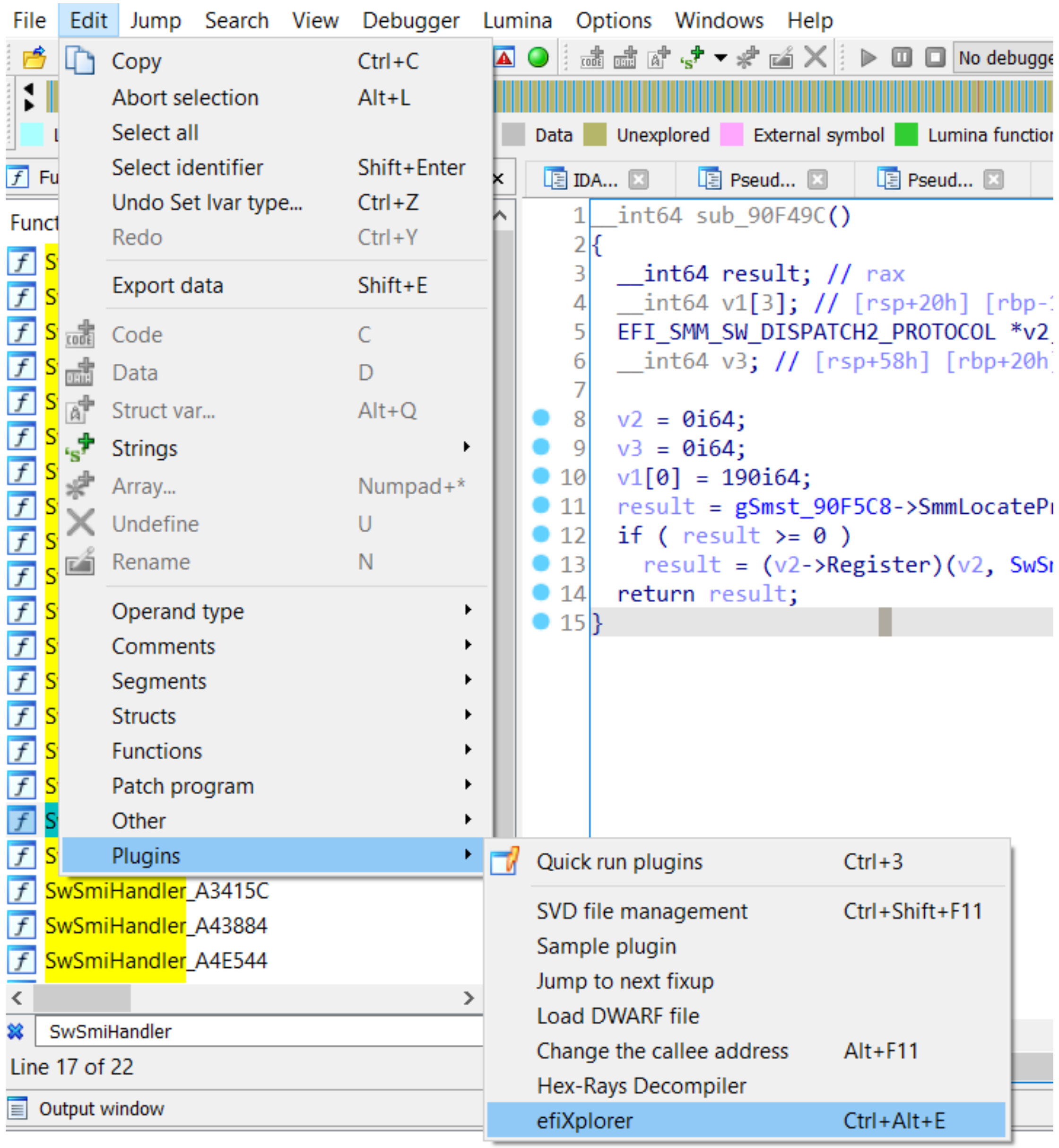
efiXloader: SMM callouts identification

- RuntimeServices



```
/* find callouts with gRT */
for (vector<ea_t>::iterator rt = gRtList.begin(); rt != gRtList.end();
     ++rt) {
    /* check if insn is 'mov rax, cs:gRT' */
    if (insn.itype == NN_mov && insn.ops[0].reg == REG_RAX &&
        insn.ops[1].type == o_mem && insn.ops[1].addr == *rt) {
        DEBUG_MSG("[%s] SMM callout found: 0x%016X\n", plugin_name,
                    ea);
        calloutAddrs.push_back(ea);
    }
}
```

efiXloader: SMM callouts identification

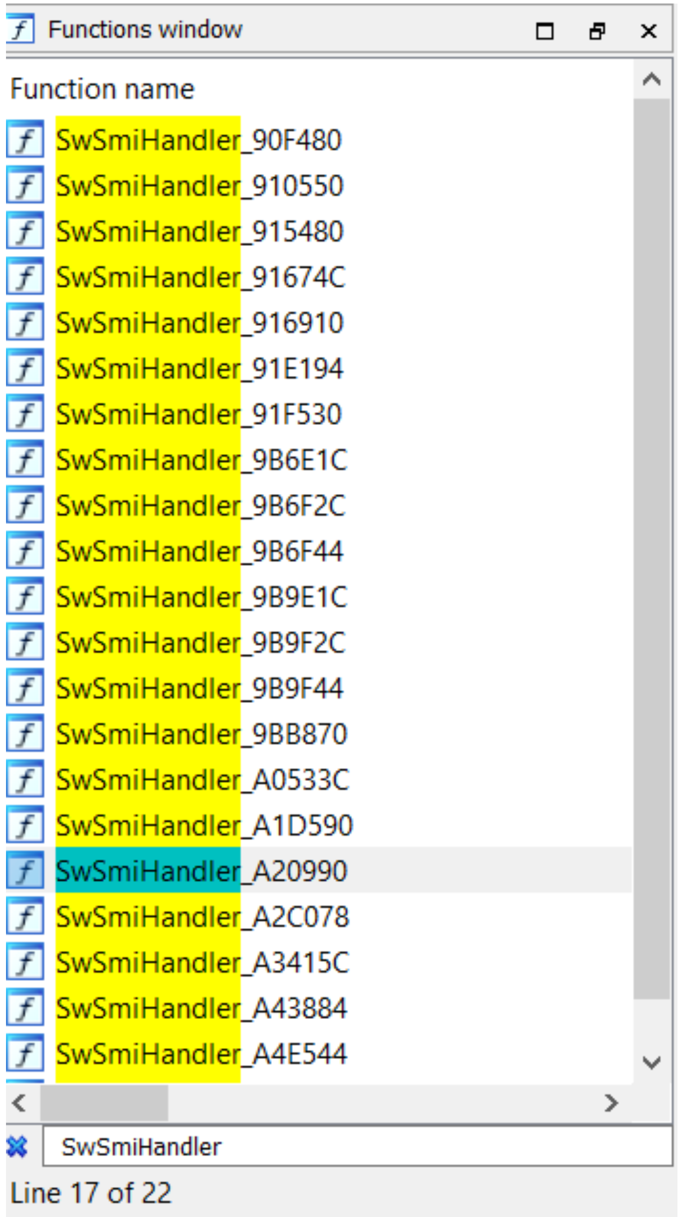


```
__int64 sub_90F49C()
{
  __int64 result; // rax
  __int64 v1[3]; // [rsp+20h] [rbp-
  EFI_SMM_SW_DISPATCH2_PROTOCOL *v2
  __int64 v3; // [rsp+58h] [rbp+20h]

  v2 = 0i64;
  v3 = 0i64;
  v1[0] = 190i64;
  result = gSmst_90F5C8->SmmLocatePr
  if ( result >= 0 )
    result = (v2->Register)(v2, SwSm
  return result;
}
```

SwSmiHandler_A3415C
SwSmiHandler_A43884
SwSmiHandler_A4E544

SwSmiHandler
Line 17 of 22

efiXplorer



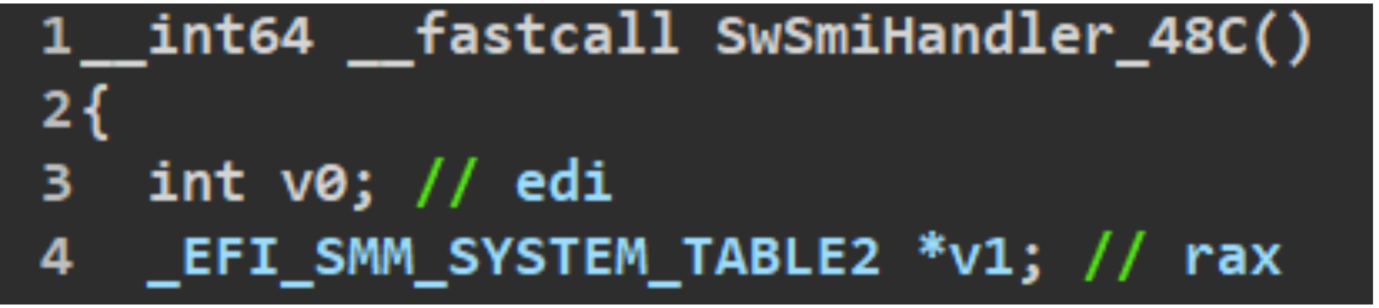
Function name
SwSmiHandler_90F480
SwSmiHandler_910550
SwSmiHandler_915480
SwSmiHandler_91674C
SwSmiHandler_916910
SwSmiHandler_91E194
SwSmiHandler_91F530
SwSmiHandler_9B6E1C
SwSmiHandler_9B6F2C
SwSmiHandler_9B6F44
SwSmiHandler_9B9E1C
SwSmiHandler_9B9F2C
SwSmiHandler_9B9F44
SwSmiHandler_9BB870
SwSmiHandler_A0533C
SwSmiHandler_A1D590
SwSmiHandler_A20990
SwSmiHandler_A2C078
SwSmiHandler_A3415C
SwSmiHandler_A43884
SwSmiHandler_A4E544

SwSmiHandler
Line 17 of 22

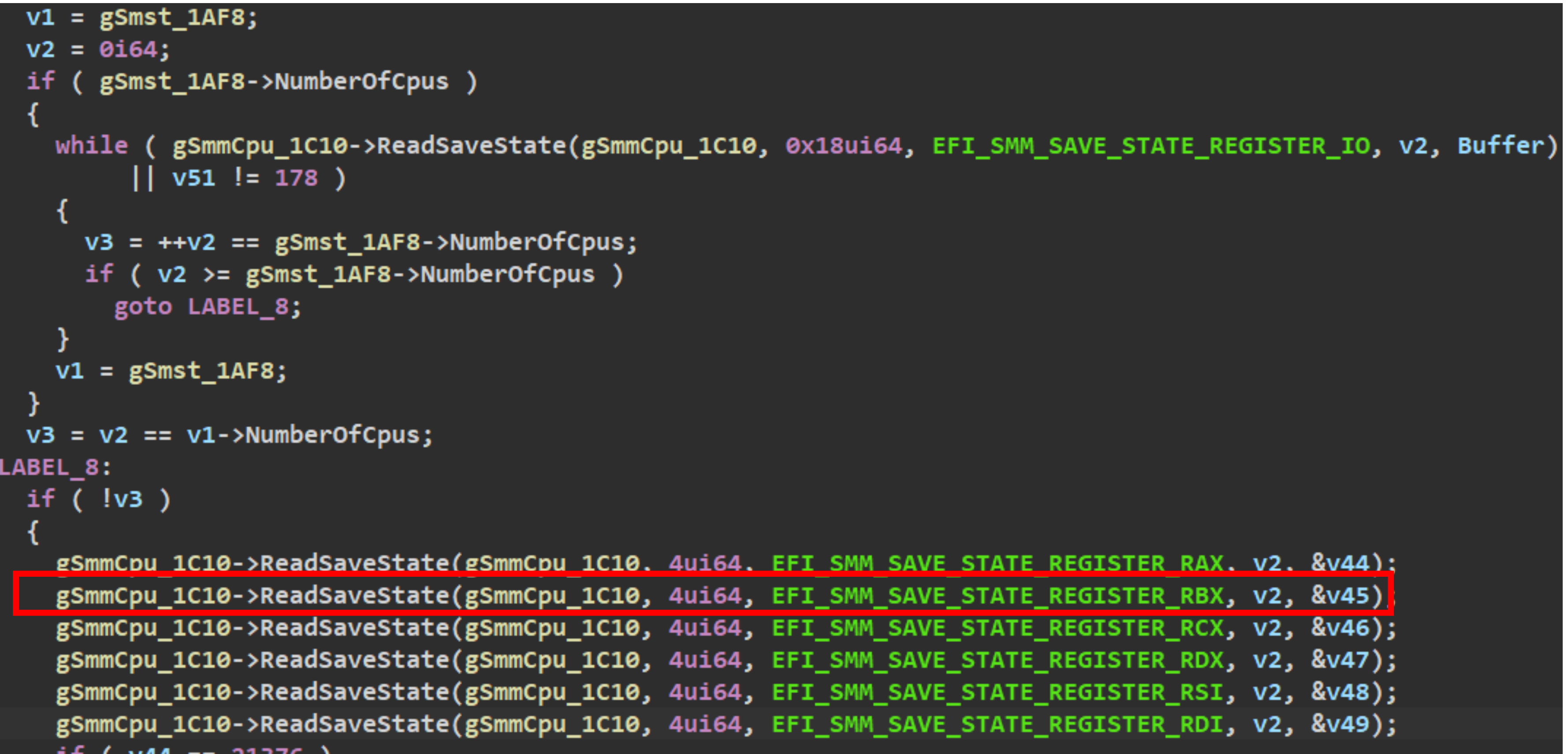
SMM callouts identification: statistics

Vendor Name	Avg number of SMI calls per firmware	Avg number of SMM callout pattern is triggered per firmware
ASRock	51	72
ASUS	42	80
Lenovo	20	3

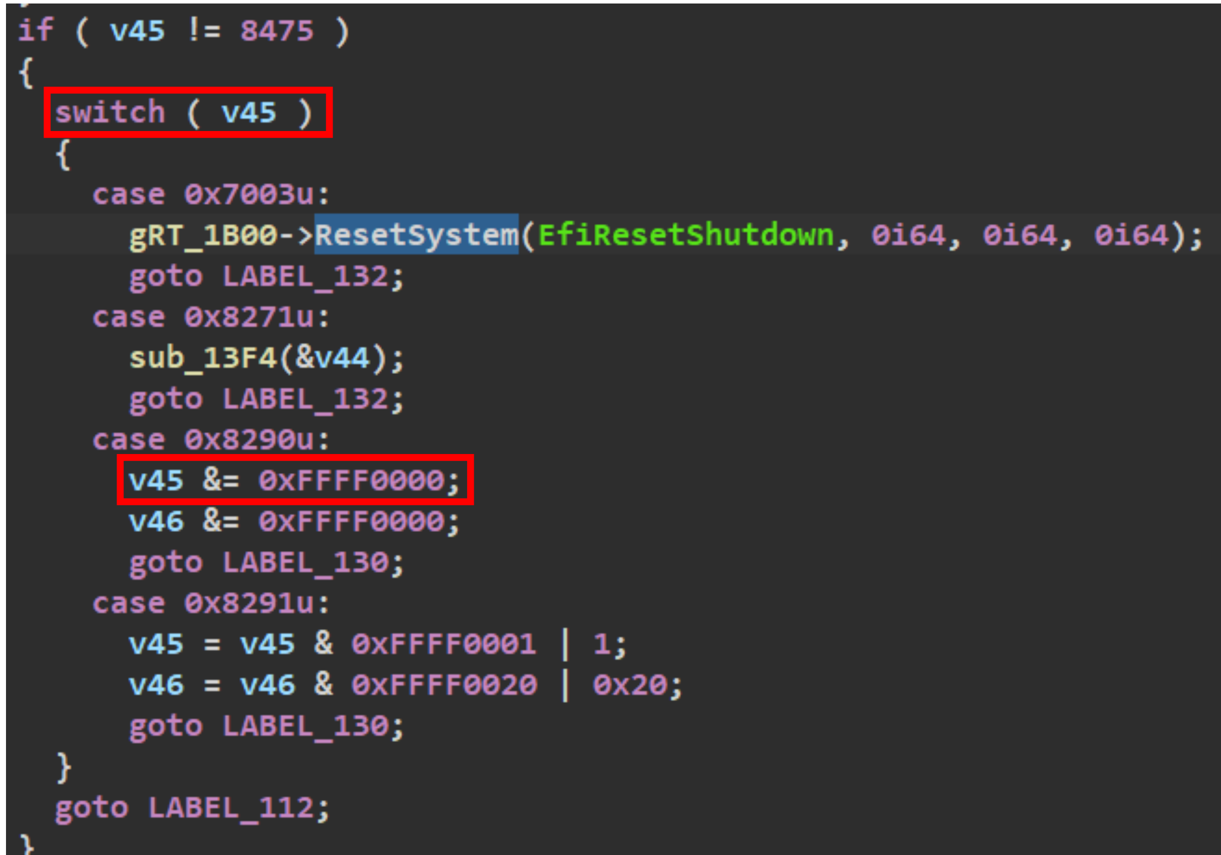
SMM callouts identification: results



```
__int64 __fastcall SwSmiHandler_48C()
{
  int v0; // edi
  _EFI_SMM_SYSTEM_TABLE2 *v1; // rax
```



```
  v1 = gSmst_1AF8;
  v2 = 0i64;
  if ( gSmst_1AF8->NumberOfCpus )
  {
    while ( gSmmCpu_1C10->ReadSaveState(gSmmCpu_1C10, 0x18ui64, EFI_SMM_SAVE_STATE_REGISTER_IO, v2, Buffer)
         || v51 != 178 )
    {
      v3 = ++v2 == gSmst_1AF8->NumberOfCpus;
      if ( v2 >= gSmst_1AF8->NumberOfCpus )
        goto LABEL_8;
    }
    v1 = gSmst_1AF8;
  }
  v3 = v2 == v1->NumberOfCpus;
LABEL_8:
  if ( !v3 )
  {
    gSmmCpu_1C10->ReadSaveState(gSmmCpu_1C10, 4ui64, EFI_SMM_SAVE_STATE_REGISTER_RAX, v2, &v44);
    gSmmCpu_1C10->ReadSaveState(gSmmCpu_1C10, 4ui64, EFI_SMM_SAVE_STATE_REGISTER_RBX, v2, &v45);
    gSmmCpu_1C10->ReadSaveState(gSmmCpu_1C10, 4ui64, EFI_SMM_SAVE_STATE_REGISTER_RCX, v2, &v46);
    gSmmCpu_1C10->ReadSaveState(gSmmCpu_1C10, 4ui64, EFI_SMM_SAVE_STATE_REGISTER_RDX, v2, &v47);
    gSmmCpu_1C10->ReadSaveState(gSmmCpu_1C10, 4ui64, EFI_SMM_SAVE_STATE_REGISTER_RSI, v2, &v48);
    gSmmCpu_1C10->ReadSaveState(gSmmCpu_1C10, 4ui64, EFI_SMM_SAVE_STATE_REGISTER_RDI, v2, &v49);
    if ( v44 == 21376 )
```

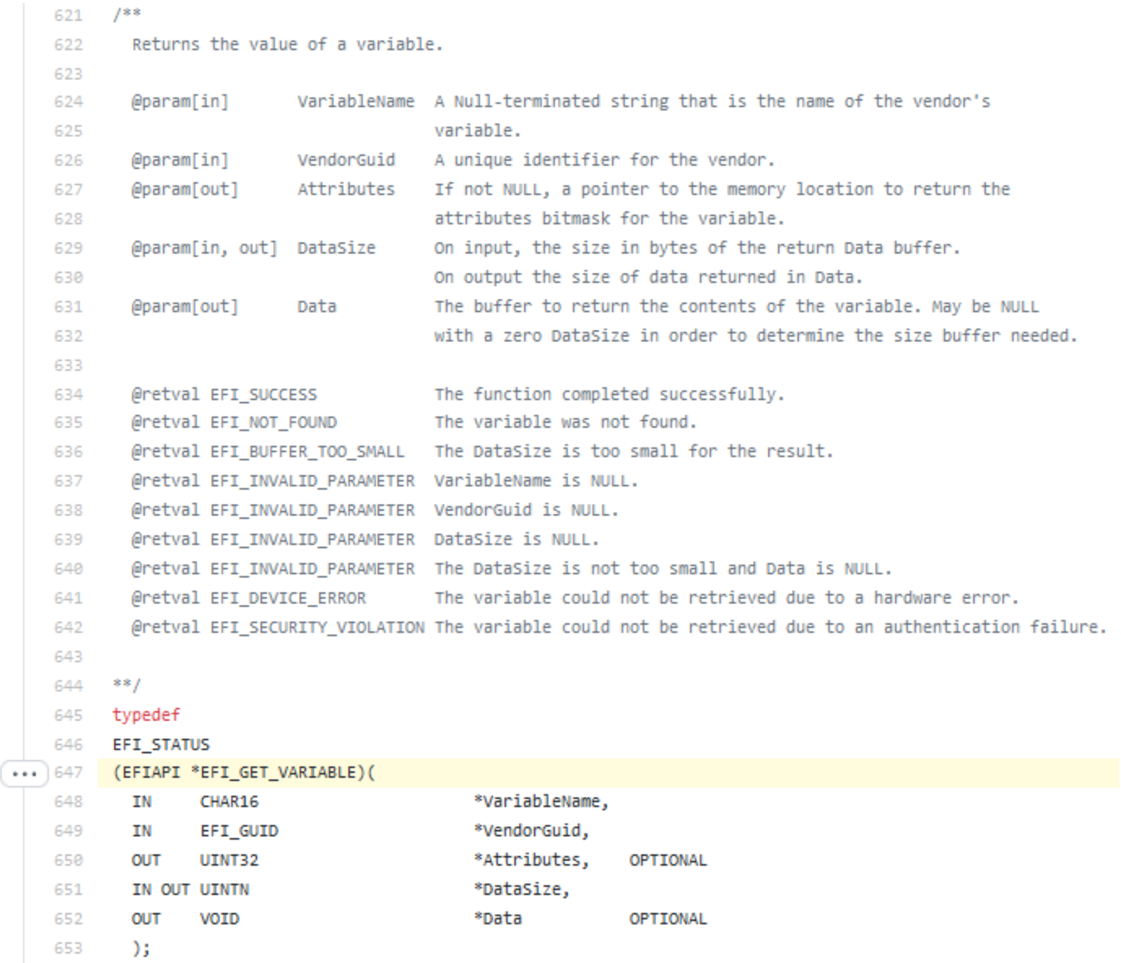


```
if ( v45 != 8475 )
{
  switch ( v45 )
  {
    case 0x7003u:
      gRT_1B00->ResetSystem(EfiResetShutdown, 0i64, 0i64, 0i64);
      goto LABEL_132;
    case 0x8271u:
      sub_13F4(&v44);
      goto LABEL_132;
    case 0x8290u:
      v45 &= 0xFFFF0000;
      v46 &= 0xFFFF0000;
      goto LABEL_130;
    case 0x8291u:
      v45 = v45 & 0xFFFF0001 | 1;
      v46 = v46 & 0xFFFF0020 | 0x20;
      goto LABEL_130;
  }
  goto LABEL_112;
}
```

efiXplorer
GetVariable vuln search

efiXplorer: GetVariable vuln search

EFI_GET_VARIABLE EFI_RUNTIME_SERVICES::GetVariable definition

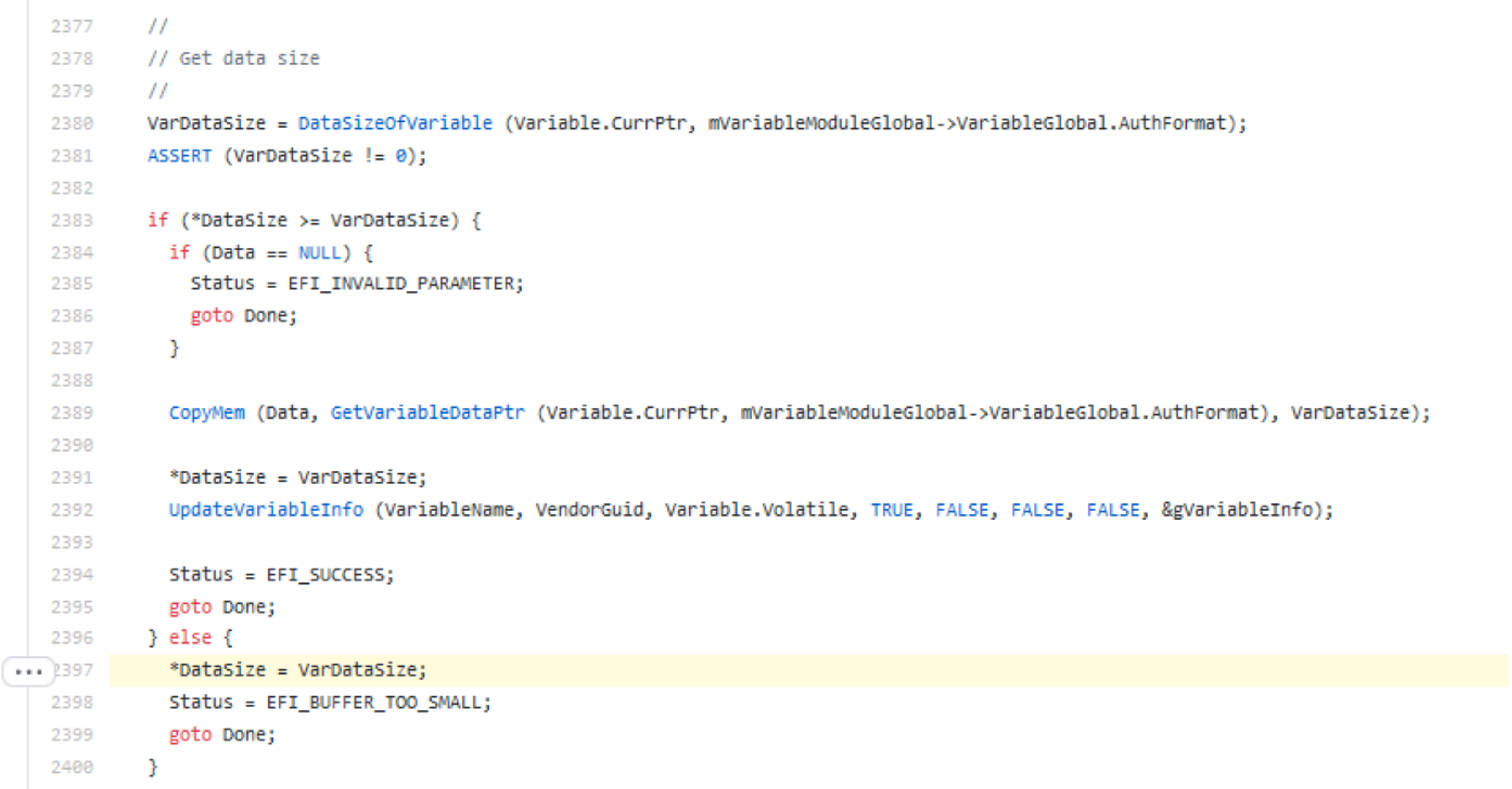


```
/**
  Returns the value of a variable.

  @param[in]       VariableName  A Null-terminated string that is the name of the vendor's
                                 variable.
  @param[in]       VendorGuid    A unique identifier for the vendor.
  @param[out]      Attributes    If not NULL, a pointer to the memory location to return the
                                 attributes bitmask for the variable.
  @param[in, out]  DataSize      On input, the size in bytes of the return Data buffer.
                                 On output the size of data returned in Data.
  @param[out]      Data          The buffer to return the contents of the variable. May be NULL
                                 with a zero DataSize in order to determine the size buffer needed.

  @retval EFI_SUCCESS            The function completed successfully.
  @retval EFI_NOT_FOUND          The variable was not found.
  @retval EFI_BUFFER_TOO_SMALL   The DataSize is too small for the result.
  @retval EFI_INVALID_PARAMETER  VariableName is NULL.
  @retval EFI_INVALID_PARAMETER  VendorGuid is NULL.
  @retval EFI_INVALID_PARAMETER  DataSize is NULL.
  @retval EFI_INVALID_PARAMETER  The DataSize is not too small and Data is NULL.
  @retval EFI_DEVICE_ERROR       The variable could not be retrieved due to a hardware error.
  @retval EFI_SECURITY_VIOLATION The variable could not be retrieved due to an authentication failure.

**/
typedef
EFI_STATUS
(EFIAPI *EFI_GET_VARIABLE)(
  IN     CHAR16                      *VariableName,
  IN     EFI_GUID                    *VendorGuid,
  OUT    UINT32                      *Attributes,    OPTIONAL
  IN OUT UINTN                       *DataSize,
  OUT    VOID                        *Data           OPTIONAL
  );
```

https://github.com/tianocore/edk2/blob/3806e1fd139775610d8f2e7541a916c3a91ad989/MdePkg/Include/Uefi/UefiSpec.h#L647

efiXplorer: GetVariable vuln search

If DataSize smaller than VarDataSize, just change DataSize and return EFI_BUFFER_TOO_SMALL status code (according to the implementation of VariableServiceGetVariable from EDK2)



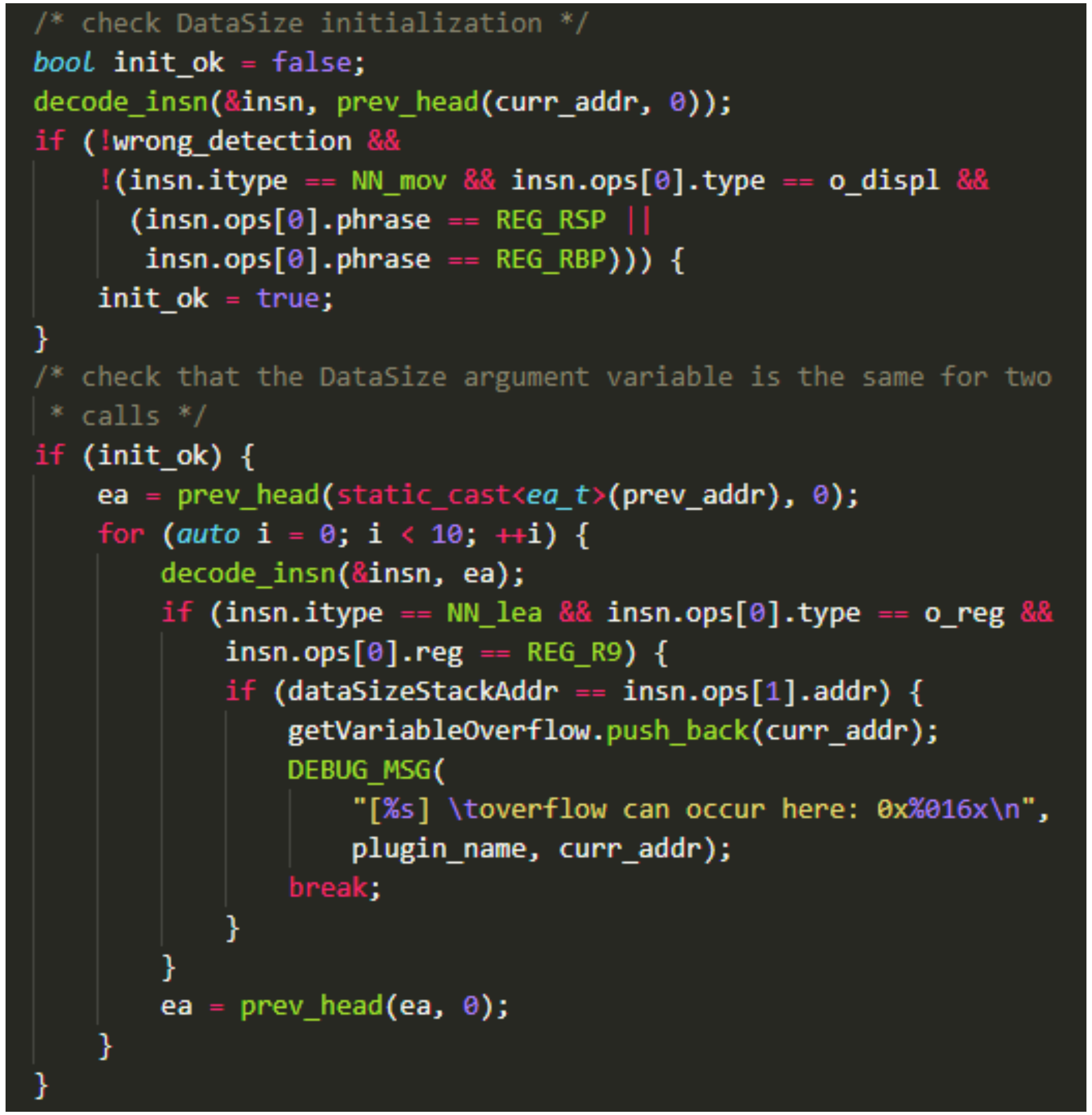
```
//
// Get data size
//
VarDataSize = DataSizeOfVariable (Variable.CurrPtr, mVariableModuleGlobal->VariableGlobal.AuthFormat);
ASSERT (VarDataSize != 0);

if (*DataSize >= VarDataSize) {
  if (Data == NULL) {
    Status = EFI_INVALID_PARAMETER;
    goto Done;
  }

  CopyMem (Data, GetVariableDataPtr (Variable.CurrPtr, mVariableModuleGlobal->VariableGlobal.AuthFormat), VarDataSize);

  *DataSize = VarDataSize;
  UpdateVariableInfo (VariableName, VendorGuid, Variable.Volatile, TRUE, FALSE, FALSE, FALSE, &gVariableInfo);

  Status = EFI_SUCCESS;
  goto Done;
} else {
  *DataSize = VarDataSize;
  Status = EFI_BUFFER_TOO_SMALL;
  goto Done;
}
```

https://github.com/tianocore/edk2/blob/master/MdeModulePkg/Universal/Variable/RuntimeDxe/Variable.c#L2397

efiXplorer: GetVariable vuln search

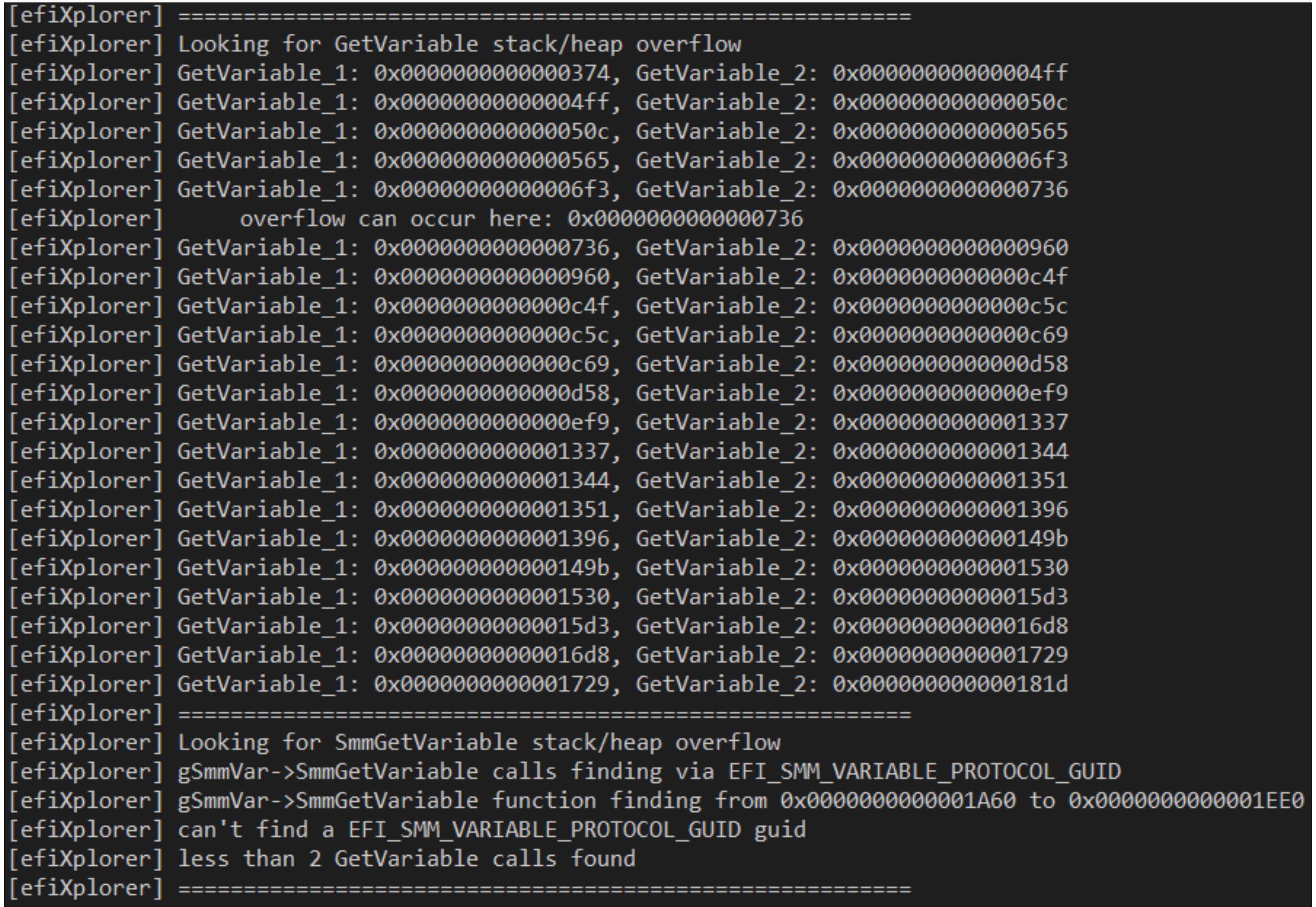
Algorithm and implementation

- loop through all the pairs of **GetVariable** calls and get the address of the **DataSize** stack variable on the first call
- check that the data size is not initialized before the second call to **GetVariable**
- check that the **DataSize** argument variable is the same for two calls



```
/* check DataSize initialization */
bool init_ok = false;
decode_insn(&insn, prev_head(curr_addr, 0));
if (!wrong_detection &&
    !(insn.itype == NN_mov && insn.ops[0].type == o_displ &&
      (insn.ops[0].phrase == REG_RSP ||
       insn.ops[0].phrase == REG_RBP))) {
    init_ok = true;
}
/* check that the DataSize argument variable is the same for two
 * calls */
if (init_ok) {
    ea = prev_head(static_cast<ea_t>(prev_addr), 0);
    for (auto i = 0; i < 10; ++i) {
        decode_insn(&insn, ea);
        if (insn.itype == NN_lea && insn.ops[0].type == o_reg &&
            insn.ops[0].reg == REG_R9) {
            if (dataSizeStackAddr == insn.ops[1].addr) {
                getVariableOverflow.push_back(curr_addr);
                DEBUG_MSG(
                    "[%s] \toverflow can occur here: 0x%016x\n",
                    plugin_name, curr_addr);
                break;
            }
        }
        ea = prev_head(ea, 0);
    }
}
```

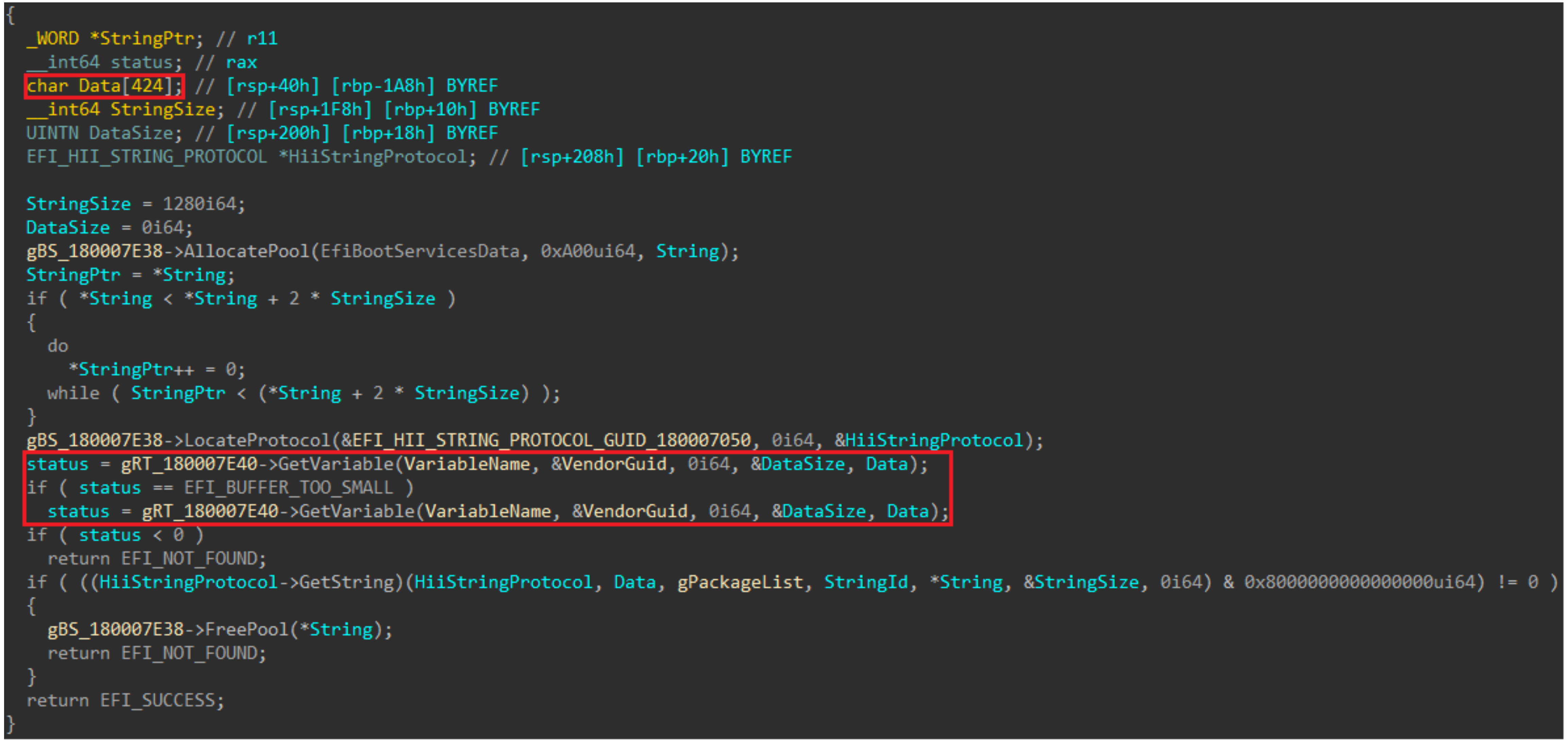
efiXplorer: GetVariable vuln search



```
[efiXplorer] ========================================================
[efiXplorer] Looking for GetVariable stack/heap overflow
[efiXplorer] GetVariable_1: 0x0000000000000374, GetVariable_2: 0x00000000000004ff
[efiXplorer] GetVariable_1: 0x00000000000004ff, GetVariable_2: 0x000000000000050c
[efiXplorer] GetVariable_1: 0x000000000000050c, GetVariable_2: 0x0000000000000565
[efiXplorer] GetVariable_1: 0x0000000000000565, GetVariable_2: 0x00000000000006f3
[efiXplorer] GetVariable_1: 0x00000000000006f3, GetVariable_2: 0x0000000000000736
[efiXplorer]      overflow can occur here: 0x0000000000000736
[efiXplorer] GetVariable_1: 0x0000000000000736, GetVariable_2: 0x0000000000000960
[efiXplorer] GetVariable_1: 0x0000000000000960, GetVariable_2: 0x0000000000000c4f
[efiXplorer] GetVariable_1: 0x0000000000000c4f, GetVariable_2: 0x0000000000000c5c
[efiXplorer] GetVariable_1: 0x0000000000000c5c, GetVariable_2: 0x0000000000000c69
[efiXplorer] GetVariable_1: 0x0000000000000c69, GetVariable_2: 0x0000000000000d58
[efiXplorer] GetVariable_1: 0x0000000000000d58, GetVariable_2: 0x0000000000000ef9
[efiXplorer] GetVariable_1: 0x0000000000000ef9, GetVariable_2: 0x0000000000001337
[efiXplorer] GetVariable_1: 0x0000000000001337, GetVariable_2: 0x0000000000001344
[efiXplorer] GetVariable_1: 0x0000000000001344, GetVariable_2: 0x0000000000001351
[efiXplorer] GetVariable_1: 0x0000000000001351, GetVariable_2: 0x0000000000001396
[efiXplorer] GetVariable_1: 0x0000000000001396, GetVariable_2: 0x000000000000149b
[efiXplorer] GetVariable_1: 0x000000000000149b, GetVariable_2: 0x0000000000001530
[efiXplorer] GetVariable_1: 0x0000000000001530, GetVariable_2: 0x00000000000015d3
[efiXplorer] GetVariable_1: 0x00000000000015d3, GetVariable_2: 0x00000000000016d8
[efiXplorer] GetVariable_1: 0x00000000000016d8, GetVariable_2: 0x0000000000001729
[efiXplorer] GetVariable_1: 0x0000000000001729, GetVariable_2: 0x000000000000181d
[efiXplorer] ========================================================
[efiXplorer] Looking for SmmGetVariable stack/heap overflow
[efiXplorer] gSmmVar->SmmGetVariable calls finding via EFI_SMM_VARIABLE_PROTOCOL_GUID
[efiXplorer] gSmmVar->SmmGetVariable function finding from 0x0000000000001A60 to 0x0000000000001EE0
[efiXplorer] can't find a EFI_SMM_VARIABLE_PROTOCOL_GUID guid
[efiXplorer] less than 2 GetVariable calls found
[efiXplorer] ========================================================
```

efiXplorer: GetVariable vuln examples

- In this case, changing the value of the variable can lead to the execution of arbitrary code

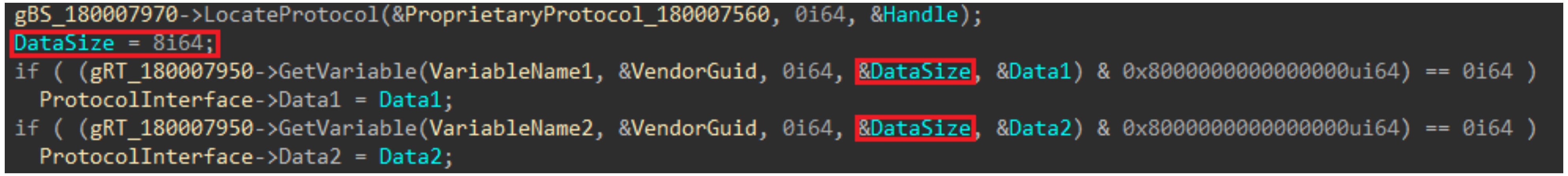


```
{
  _WORD *StringPtr; // r11
  __int64 status; // rax
  char Data[424]; // [rsp+40h] [rbp-1A8h] BYREF
  __int64 StringSize; // [rsp+1F8h] [rbp+10h] BYREF
  UINTN DataSize; // [rsp+200h] [rbp+18h] BYREF
  EFI_HII_STRING_PROTOCOL *HiiStringProtocol; // [rsp+208h] [rbp+20h] BYREF

  StringSize = 1280i64;
  DataSize = 0i64;
  gBS_180007E38->AllocatePool(EfiBootServicesData, 0xA00ui64, String);
  StringPtr = *String;
  if ( *String < *String + 2 * StringSize )
  {
    do
      *StringPtr++ = 0;
    while ( StringPtr < (*String + 2 * StringSize) );
  }
  gBS_180007E38->LocateProtocol(&EFI_HII_STRING_PROTOCOL_GUID_180007050, 0i64, &HiiStringProtocol);
  status = gRT_180007E40->GetVariable(VariableName, &VendorGuid, 0i64, &DataSize, Data);
  if ( status == EFI_BUFFER_TOO_SMALL )
    status = gRT_180007E40->GetVariable(VariableName, &VendorGuid, 0i64, &DataSize, Data);
  if ( status < 0 )
    return EFI_NOT_FOUND;
  if ( ((HiiStringProtocol->GetString)(HiiStringProtocol, Data, gPackageList, StringId, *String, &StringSize, 0i64) & 0x8000000000000000ui64) != 0 )
  {
    gBS_180007E38->FreePool(*String);
    return EFI_NOT_FOUND;
  }
  return EFI_SUCCESS;
}
```

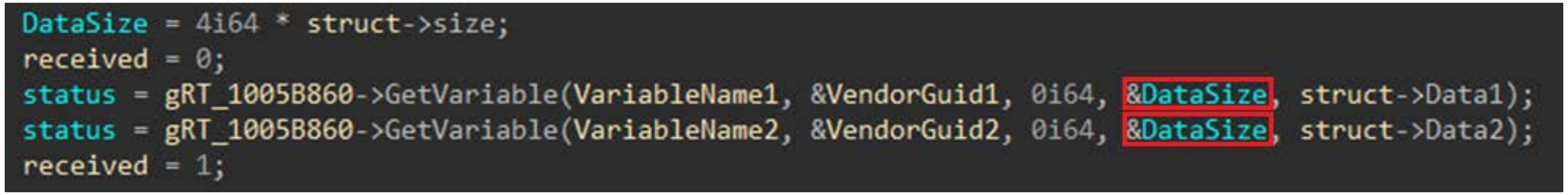
efiXplorer: GetVariable vuln examples

- **The sequence of multiple GetVariable calls may cause the buffer overflow as follows**

1. First call is required to update DataSize value
2. Second call — trigger OOB write



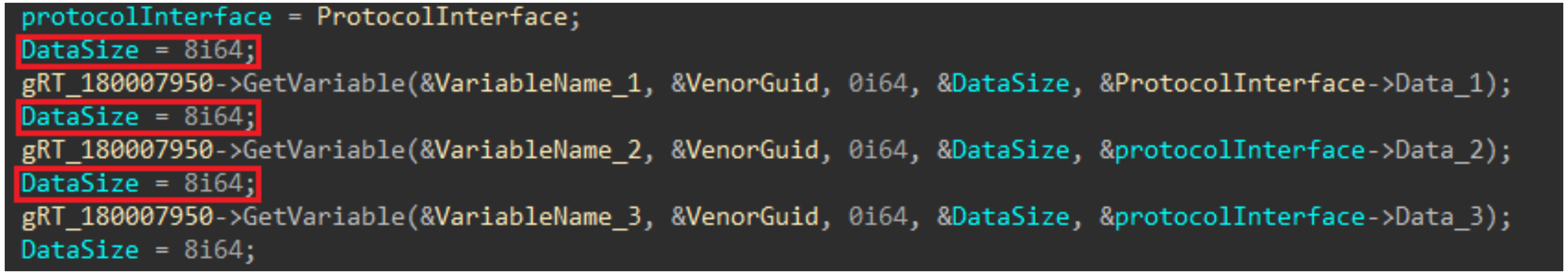
```
gBS_180007970->LocateProtocol(&ProprietaryProtocol_180007560, 0i64, &Handle);
DataSize = 8i64;
if ( (gRT_180007950->GetVariable(VariableName1, &VendorGuid, 0i64, &DataSize, &Data1) & 0x8000000000000000ui64) == 0i64 )
  ProtocolInterface->Data1 = Data1;
if ( (gRT_180007950->GetVariable(VariableName2, &VendorGuid, 0i64, &DataSize, &Data2) & 0x8000000000000000ui64) == 0i64 )
  ProtocolInterface->Data2 = Data2;
```



```
DataSize = 4i64 * struct->size;
received = 0;
status = gRT_1005B860->GetVariable(VariableName1, &VendorGuid1, 0i64, &DataSize, struct->Data1);
status = gRT_1005B860->GetVariable(VariableName2, &VendorGuid2, 0i64, &DataSize, struct->Data2);
received = 1;
```

- **Correct usage**

Initializing data size before each call

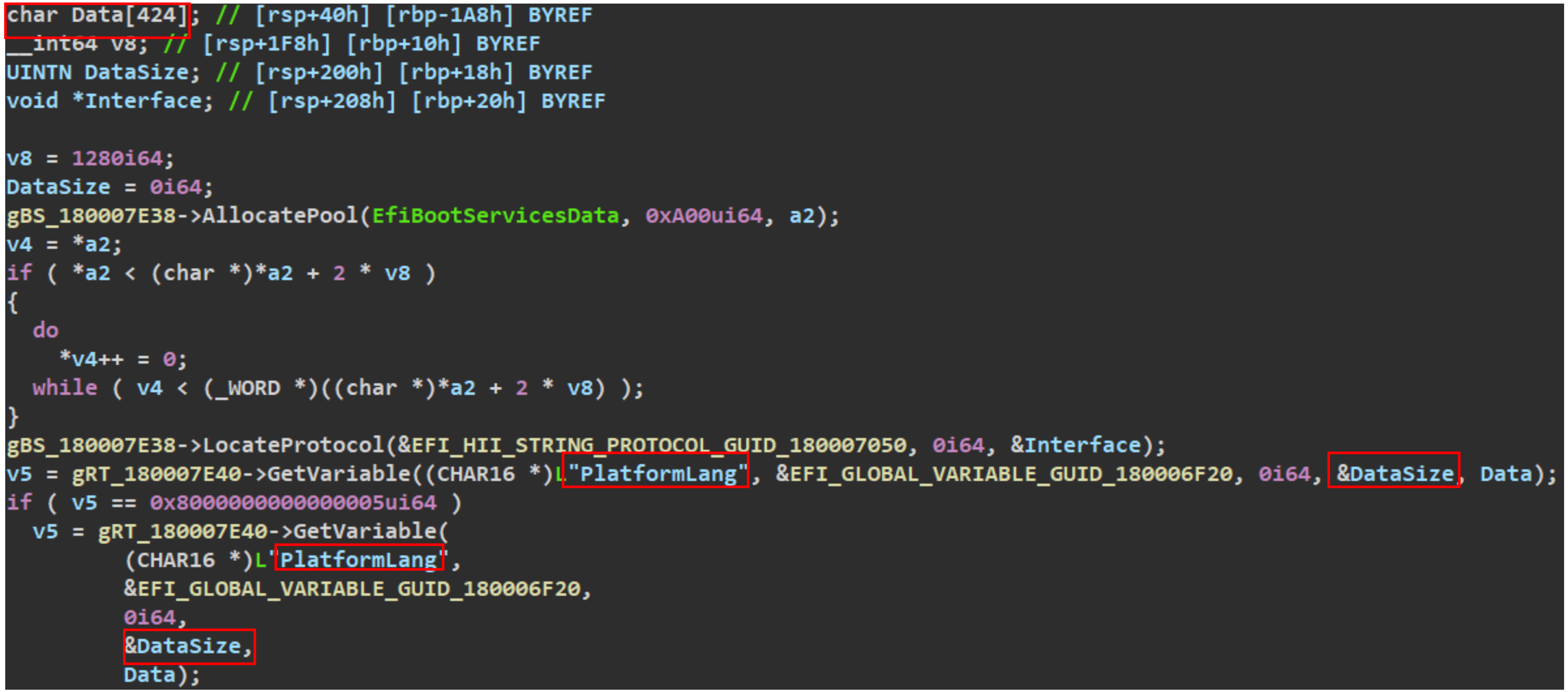


```
protocolInterface = ProtocolInterface;
DataSize = 8i64;
gRT_180007950->GetVariable(&VariableName_1, &VenorGuid, 0i64, &DataSize, &ProtocolInterface->Data_1);
DataSize = 8i64;
gRT_180007950->GetVariable(&VariableName_2, &VenorGuid, 0i64, &DataSize, &protocolInterface->Data_2);
DataSize = 8i64;
gRT_180007950->GetVariable(&VariableName_3, &VenorGuid, 0i64, &DataSize, &protocolInterface->Data_3);
DataSize = 8i64;
```

GetVariable vuln search: statistics

Vendor Name	Avg number of calls per firmware	Avg number of vuln pattern is triggered per firmware
ASRock	735	2
ASUS	697	5
Lenovo	466	20

DXE GetVariable vuln search: results

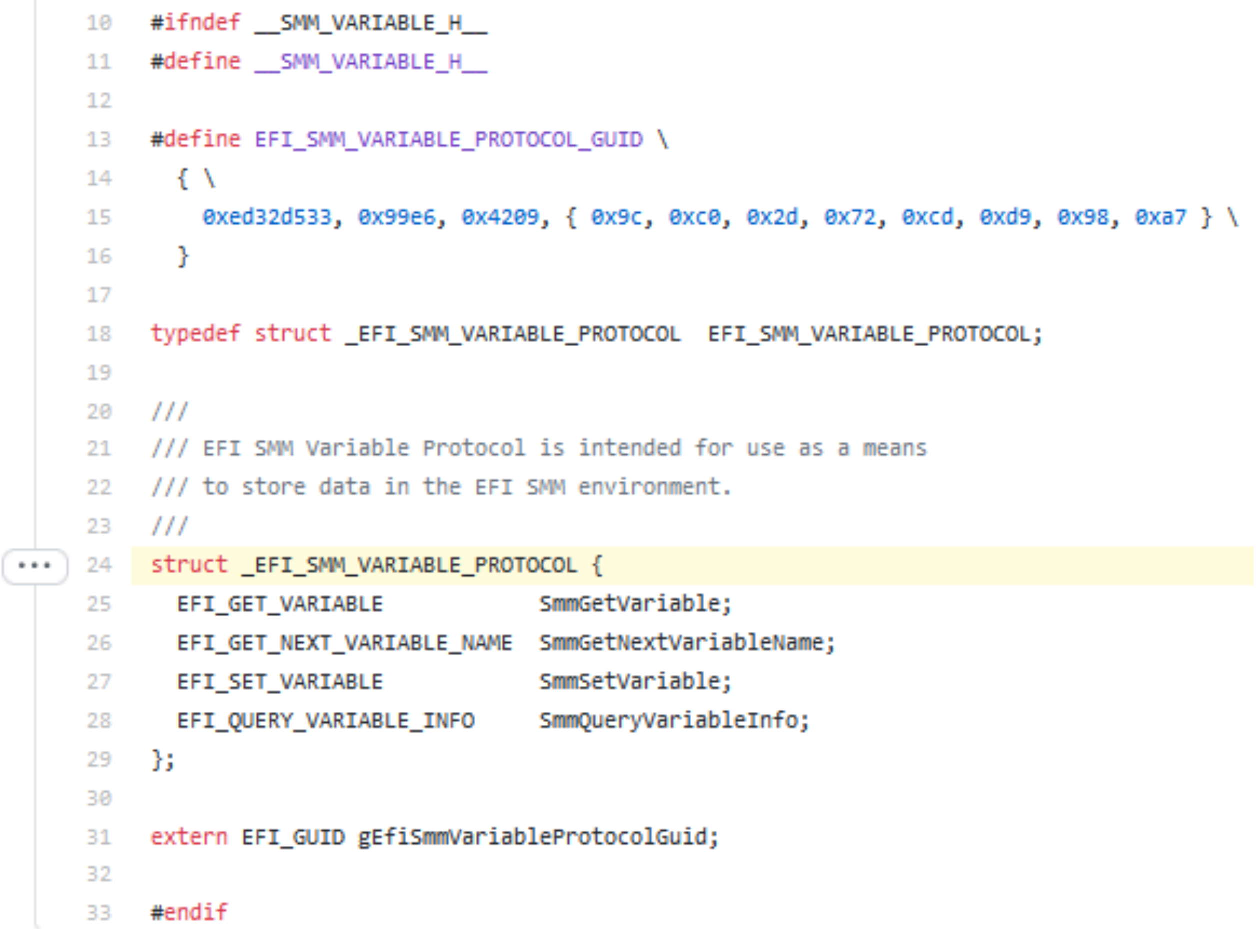


```
char Data[424]; // [rsp+40h] [rbp-1A8h] BYREF
__int64 v8; // [rsp+1F8h] [rbp+10h] BYREF
UINTN DataSize; // [rsp+200h] [rbp+18h] BYREF
void *Interface; // [rsp+208h] [rbp+20h] BYREF

v8 = 1280i64;
DataSize = 0i64;
gBS_180007E38->AllocatePool(EfiBootServicesData, 0xA00ui64, a2);
v4 = *a2;
if ( *a2 < (char *)*a2 + 2 * v8 )
{
  do
    *v4++ = 0;
  while ( v4 < (_WORD *)((char *)*a2 + 2 * v8) );
}
gBS_180007E38->LocateProtocol(&EFI_HII_STRING_PROTOCOL_GUID_180007050, 0i64, &Interface);
v5 = gRT_180007E40->GetVariable((CHAR16 *)L"PlatformLang", &EFI_GLOBAL_VARIABLE_GUID_180006F20, 0i64, &DataSize, Data);
if ( v5 == 0x8000000000000005ui64 )
  v5 = gRT_180007E40->GetVariable(
         (CHAR16 *)L"PlatformLang",
         &EFI_GLOBAL_VARIABLE_GUID_180006F20,
         0i64,
         &DataSize,
         Data);
```

efiXplorer SmmGetVariable vuln search

efiXplorer: SmmGetVariable vuln search

- **SmmGetVariable** - function from **EFI_SMM_VARIABLE_PROTOCOL**
- functionality is like **EFI_RUNTIME_SERVICES::GetVariable**



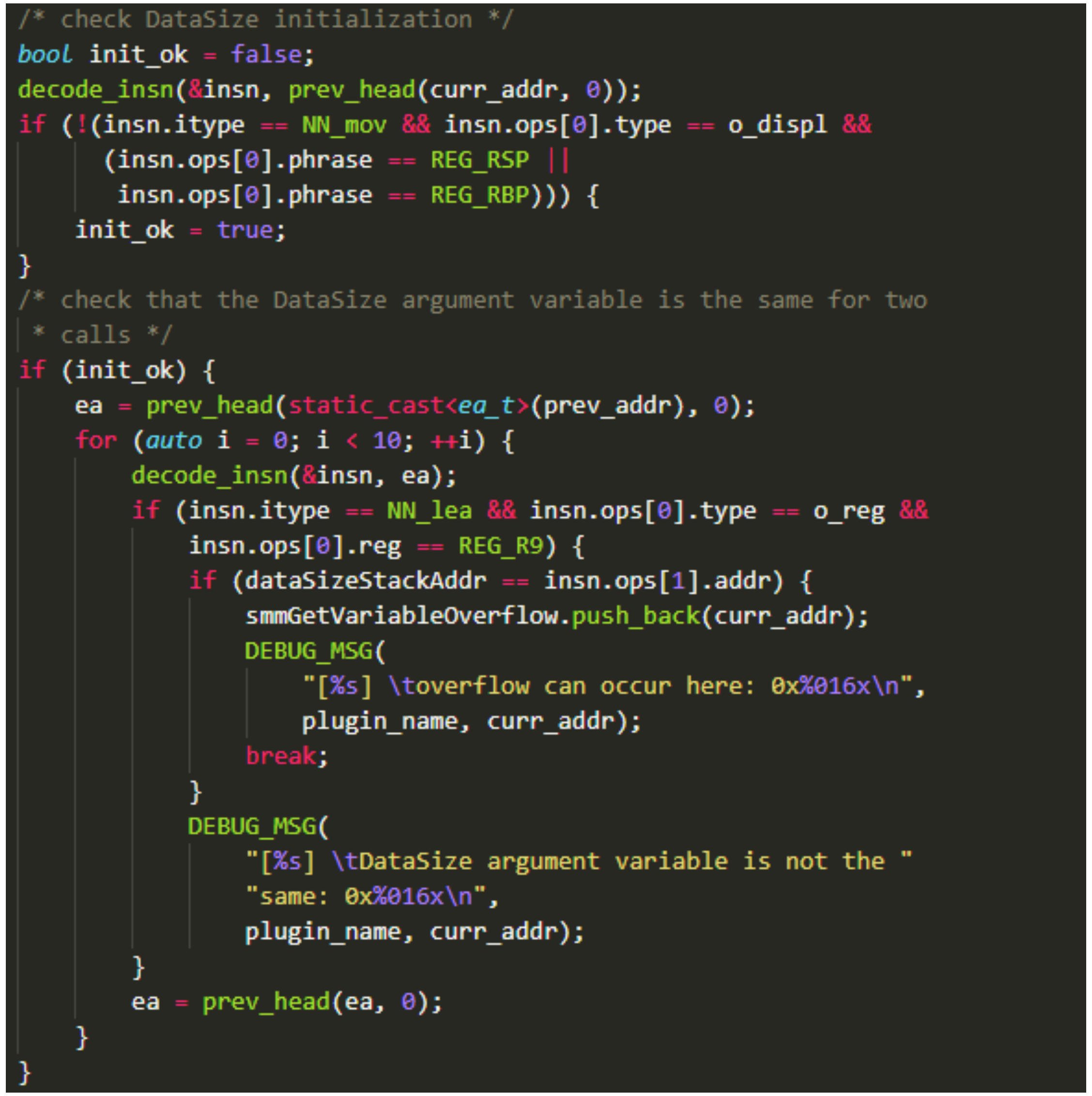
```
#ifndef __SMM_VARIABLE_H__
#define __SMM_VARIABLE_H__

#define EFI_SMM_VARIABLE_PROTOCOL_GUID \
  { \
    0xed32d533, 0x99e6, 0x4209, { 0x9c, 0xc0, 0x2d, 0x72, 0xcd, 0xd9, 0x98, 0xa7 } \
  }

typedef struct _EFI_SMM_VARIABLE_PROTOCOL  EFI_SMM_VARIABLE_PROTOCOL;

///
/// EFI SMM Variable Protocol is intended for use as a means
/// to store data in the EFI SMM environment.
///
struct _EFI_SMM_VARIABLE_PROTOCOL {
  EFI_GET_VARIABLE            SmmGetVariable;
  EFI_GET_NEXT_VARIABLE_NAME  SmmGetNextVariableName;
  EFI_SET_VARIABLE            SmmSetVariable;
  EFI_QUERY_VARIABLE_INFO     SmmQueryVariableInfo;
};

extern EFI_GUID gEfiSmmVariableProtocolGuid;

#endif
```

https://github.com/tianocore/edk2/blob/3806e1fd139775610d8f2e7541a916c3a91ad989/MdeModulePkg/Include/Protocol/SmmVariable.h#L24

efiXplorer: SmmGetVariable vuln search

Algorithm and implementation (similar to GetVariable vuln search)

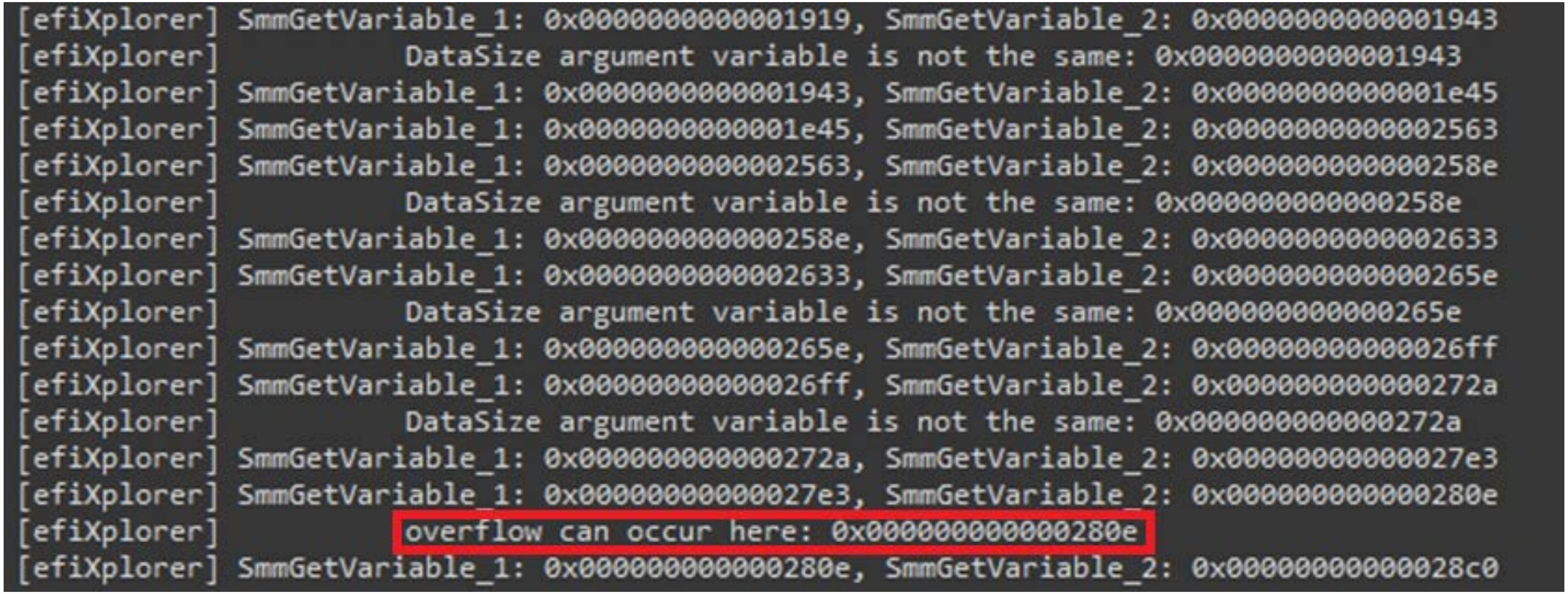
- loop through all the pairs of **SmmGetVariable** calls and get the address of the **DataSize** stack variable on the first call
- check that the data size is not initialized before the second call to **SmmGetVariable**
- check that the **DataSize** argument variable is the same for two calls



```
/* check DataSize initialization */
bool init_ok = false;
decode_insn(&insn, prev_head(curr_addr, 0));
if (!(insn.itype == NN_mov && insn.ops[0].type == o_displ &&
      (insn.ops[0].phrase == REG_RSP ||
       insn.ops[0].phrase == REG_RBP))) {
    init_ok = true;
}
/* check that the DataSize argument variable is the same for two
 * calls */
if (init_ok) {
    ea = prev_head(static_cast<ea_t>(prev_addr), 0);
    for (auto i = 0; i < 10; ++i) {
        decode_insn(&insn, ea);
        if (insn.itype == NN_lea && insn.ops[0].type == o_reg &&
            insn.ops[0].reg == REG_R9) {
            if (dataSizeStackAddr == insn.ops[1].addr) {
                smmGetVariableOverflow.push_back(curr_addr);
                DEBUG_MSG(
                    "[%s] \toverflow can occur here: 0x%016x\n",
                    plugin_name, curr_addr);
                break;
            }
            DEBUG_MSG(
                "[%s] \tDataSize argument variable is not the "
                "same: 0x%016x\n",
                plugin_name, curr_addr);
        }
        ea = prev_head(ea, 0);
    }
}
```

efiXplorer: SmmGetVariable vuln search

- **Static analyzer messages in the IDA output window**

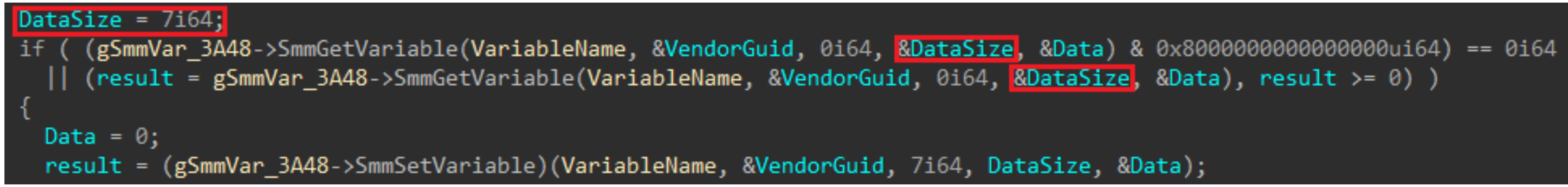


```
[efiXplorer] SmmGetVariable_1: 0x0000000000001919, SmmGetVariable_2: 0x0000000000001943
[efiXplorer]          DataSize argument variable is not the same: 0x0000000000001943
[efiXplorer] SmmGetVariable_1: 0x0000000000001943, SmmGetVariable_2: 0x0000000000001e45
[efiXplorer] SmmGetVariable_1: 0x0000000000001e45, SmmGetVariable_2: 0x0000000000002563
[efiXplorer] SmmGetVariable_1: 0x0000000000002563, SmmGetVariable_2: 0x000000000000258e
[efiXplorer]          DataSize argument variable is not the same: 0x000000000000258e
[efiXplorer] SmmGetVariable_1: 0x000000000000258e, SmmGetVariable_2: 0x0000000000002633
[efiXplorer] SmmGetVariable_1: 0x0000000000002633, SmmGetVariable_2: 0x000000000000265e
[efiXplorer]          DataSize argument variable is not the same: 0x000000000000265e
[efiXplorer] SmmGetVariable_1: 0x000000000000265e, SmmGetVariable_2: 0x00000000000026ff
[efiXplorer] SmmGetVariable_1: 0x00000000000026ff, SmmGetVariable_2: 0x000000000000272a
[efiXplorer]          DataSize argument variable is not the same: 0x000000000000272a
[efiXplorer] SmmGetVariable_1: 0x000000000000272a, SmmGetVariable_2: 0x00000000000027e3
[efiXplorer] SmmGetVariable_1: 0x00000000000027e3, SmmGetVariable_2: 0x000000000000280e
[efiXplorer]          overflow can occur here: 0x000000000000280e
[efiXplorer] SmmGetVariable_1: 0x000000000000280e, SmmGetVariable_2: 0x00000000000028c0
```

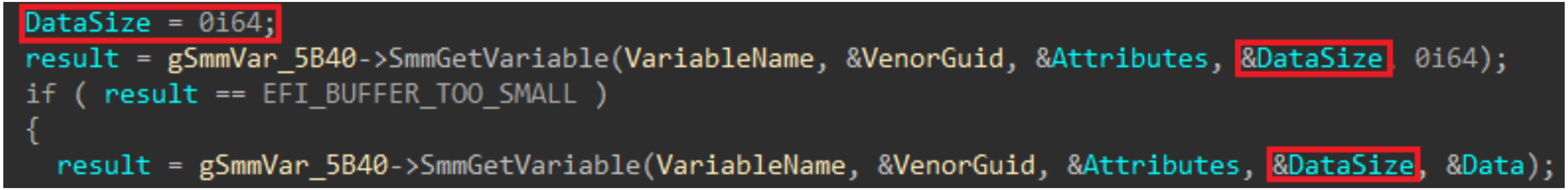
efiXplorer: SmmGetVariable vuln examples

- **The sequence of multiple SmmGetVariable calls may cause the buffer overflow inside SMM**

1. First call is required to update DataSize value
2. Second call — trigger OOB write



```
DataSize = 7i64;
if ( (gSmmVar_3A48->SmmGetVariable(VariableName, &VendorGuid, 0i64, &DataSize, &Data) & 0x8000000000000000ui64) == 0i64
  || (result = gSmmVar_3A48->SmmGetVariable(VariableName, &VendorGuid, 0i64, &DataSize, &Data), result >= 0) )
{
  Data = 0;
  result = (gSmmVar_3A48->SmmSetVariable)(VariableName, &VendorGuid, 7i64, DataSize, &Data);
```



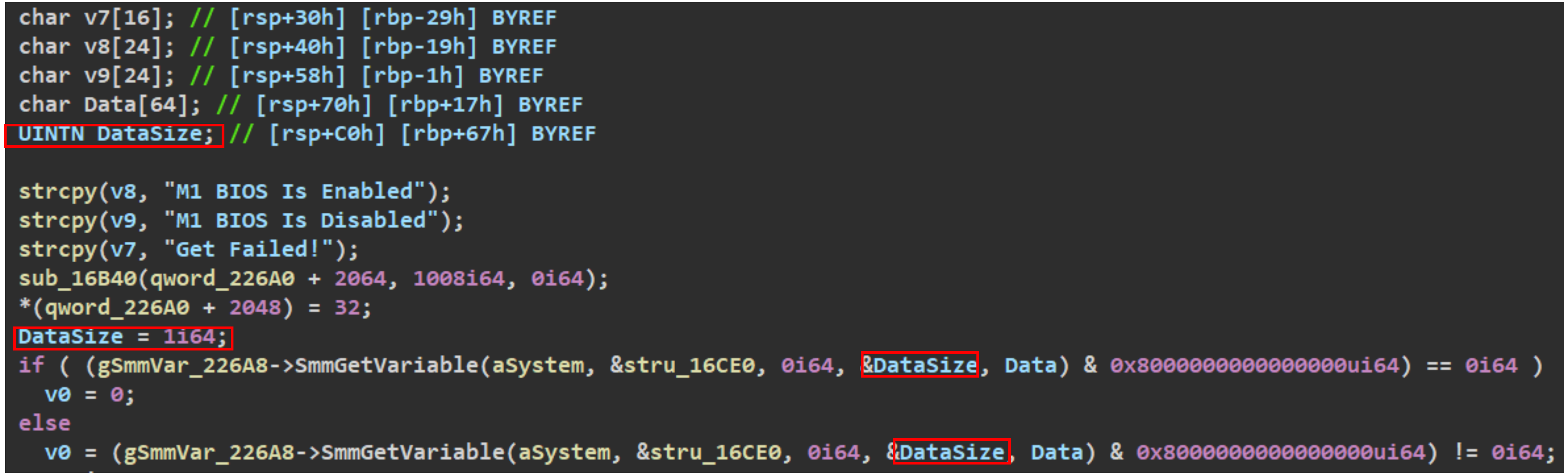
```
DataSize = 0i64;
result = gSmmVar_5B40->SmmGetVariable(VariableName, &VenorGuid, &Attributes, &DataSize, 0i64);
if ( result == EFI_BUFFER_TOO_SMALL )
{
  result = gSmmVar_5B40->SmmGetVariable(VariableName, &VenorGuid, &Attributes, &DataSize, &Data);
```

SmmGetVariable vuln search: statistics

Vendor Name	Avg number of calls per firmware	Avg number of vuln pattern is triggered per firmware
ASRock	8	0
ASUS	7	0*
Lenovo	15	1

* 3 cases among 820 firmware images

SmmGetVariable vuln search: results

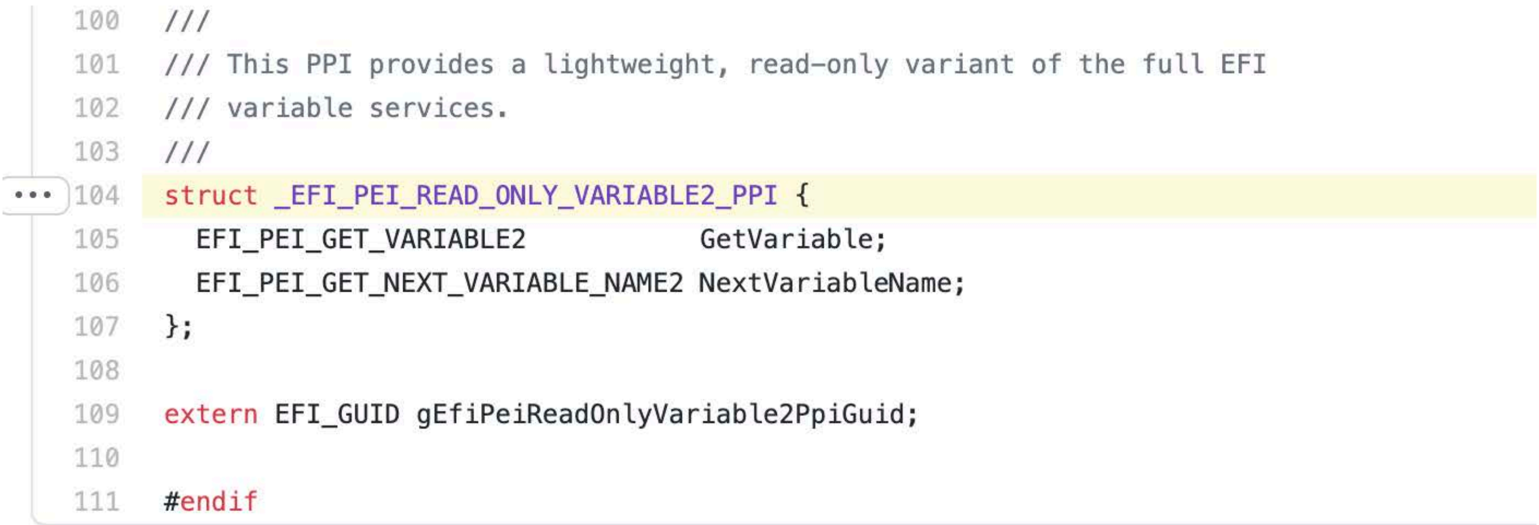


```
char v7[16]; // [rsp+30h] [rbp-29h] BYREF
char v8[24]; // [rsp+40h] [rbp-19h] BYREF
char v9[24]; // [rsp+58h] [rbp-1h] BYREF
char Data[64]; // [rsp+70h] [rbp+17h] BYREF
UINTN DataSize; // [rsp+C0h] [rbp+67h] BYREF

strcpy(v8, "M1 BIOS Is Enabled");
strcpy(v9, "M1 BIOS Is Disabled");
strcpy(v7, "Get Failed!");
sub_16B40(qword_226A0 + 2064, 1008i64, 0i64);
*(qword_226A0 + 2048) = 32;
DataSize = 1i64;
if ( (gSmmVar_226A8->SmmGetVariable(aSystem, &stru_16CE0, 0i64, &DataSize, Data) & 0x8000000000000000ui64) == 0i64 )
  v0 = 0;
else
  v0 = (gSmmVar_226A8->SmmGetVariable(aSystem, &stru_16CE0, 0i64, &DataSize, Data) & 0x8000000000000000ui64) != 0i64;
```

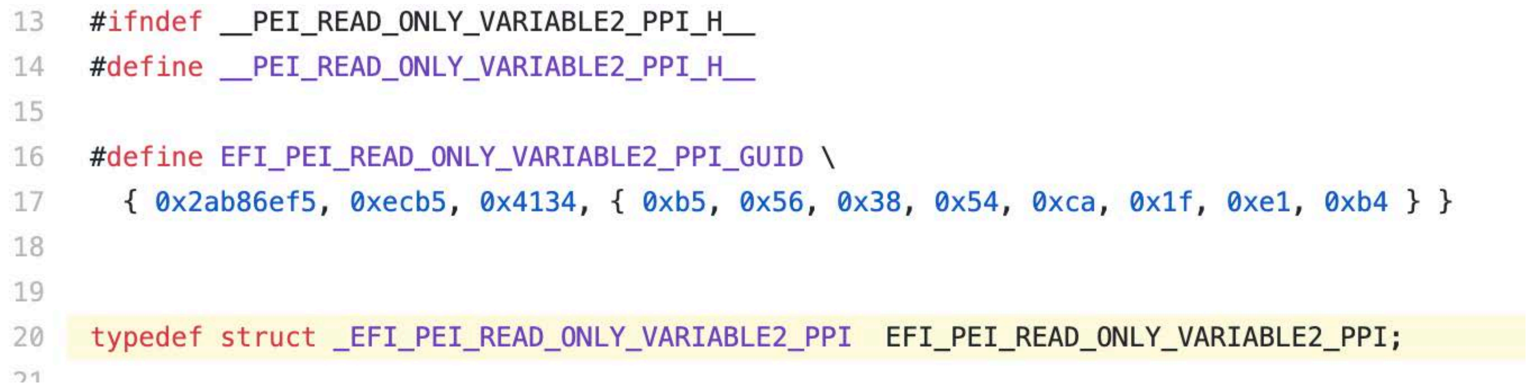
efiXplorer PPI GetVariable vuln search

efiXplorer: PPI GetVariable vuln search

Similar to GetVariable in SMM, PEI modules rely on EFI_PEI_READ_ONLY_VARIABLE2_PPI service to read nvram variables



```
///
/// This PPI provides a lightweight, read-only variant of the full EFI
/// variable services.
///
struct _EFI_PEI_READ_ONLY_VARIABLE2_PPI {
  EFI_PEI_GET_VARIABLE2           GetVariable;
  EFI_PEI_GET_NEXT_VARIABLE_NAME2 NextVariableName;
};

extern EFI_GUID gEfiPeiReadOnlyVariable2PpiGuid;

#endif
```



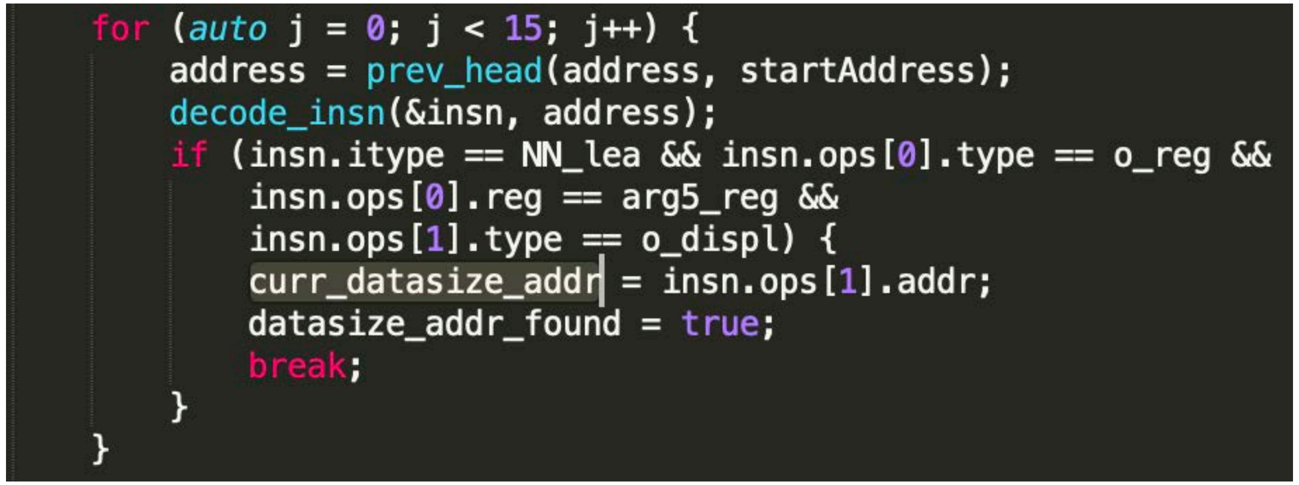
```
#ifndef __PEI_READ_ONLY_VARIABLE2_PPI_H__
#define __PEI_READ_ONLY_VARIABLE2_PPI_H__

#define EFI_PEI_READ_ONLY_VARIABLE2_PPI_GUID \
  { 0x2ab86ef5, 0xecb5, 0x4134, { 0xb5, 0x56, 0x38, 0x54, 0xca, 0x1f, 0xe1, 0xb4 } }


typedef struct _EFI_PEI_READ_ONLY_VARIABLE2_PPI  EFI_PEI_READ_ONLY_VARIABLE2_PPI;
```

https://github.com/tianocore/edk2/blob/3806e1fd139775610d8f2e7541a916c3a91ad989/MdePkg/Include/Ppi/ReadOnlyVariable2.h#L104

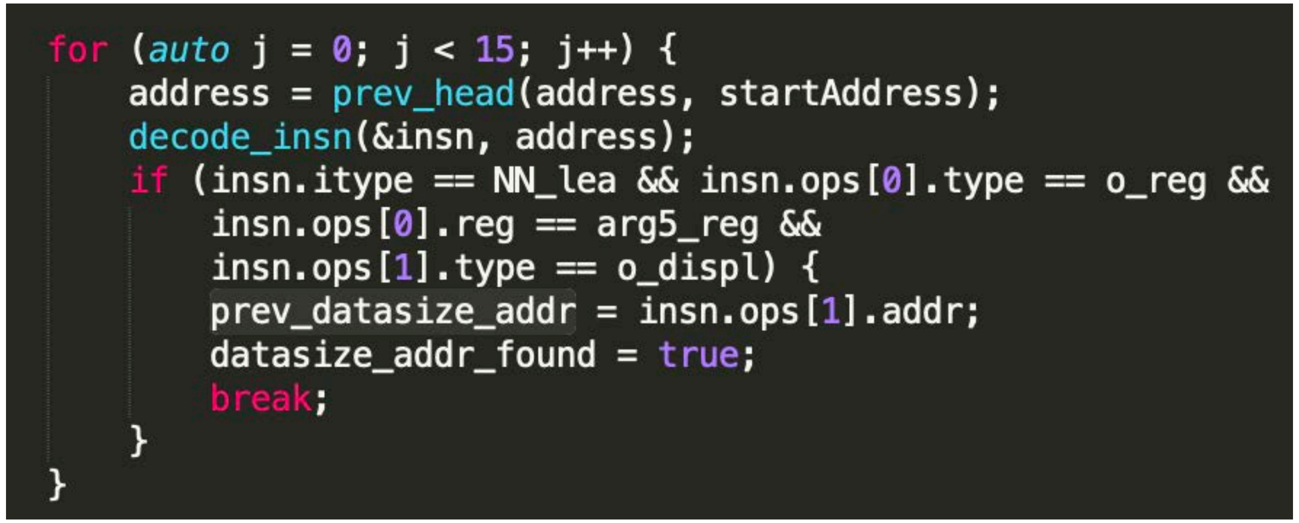
efiXplorer: PPI GetVariable vuln search

Algorithm and implementation (similar to SmmGetVariable vuln search)

- loop through all the pairs of **VariablePPI->GetVariable** calls and get the address of the **DataSize** stack variable on the first call
- check that the **DataSize** argument is the same for both calls



```
for (auto j = 0; j < 15; j++) {
    address = prev_head(address, startAddress);
    decode_insn(&insn, address);
    if (insn.itype == NN_lea && insn.ops[0].type == o_reg &&
        insn.ops[0].reg == arg5_reg &&
        insn.ops[1].type == o_displ) {
        curr_datasize_addr = insn.ops[1].addr;
        datasize_addr_found = true;
        break;
    }
}
```

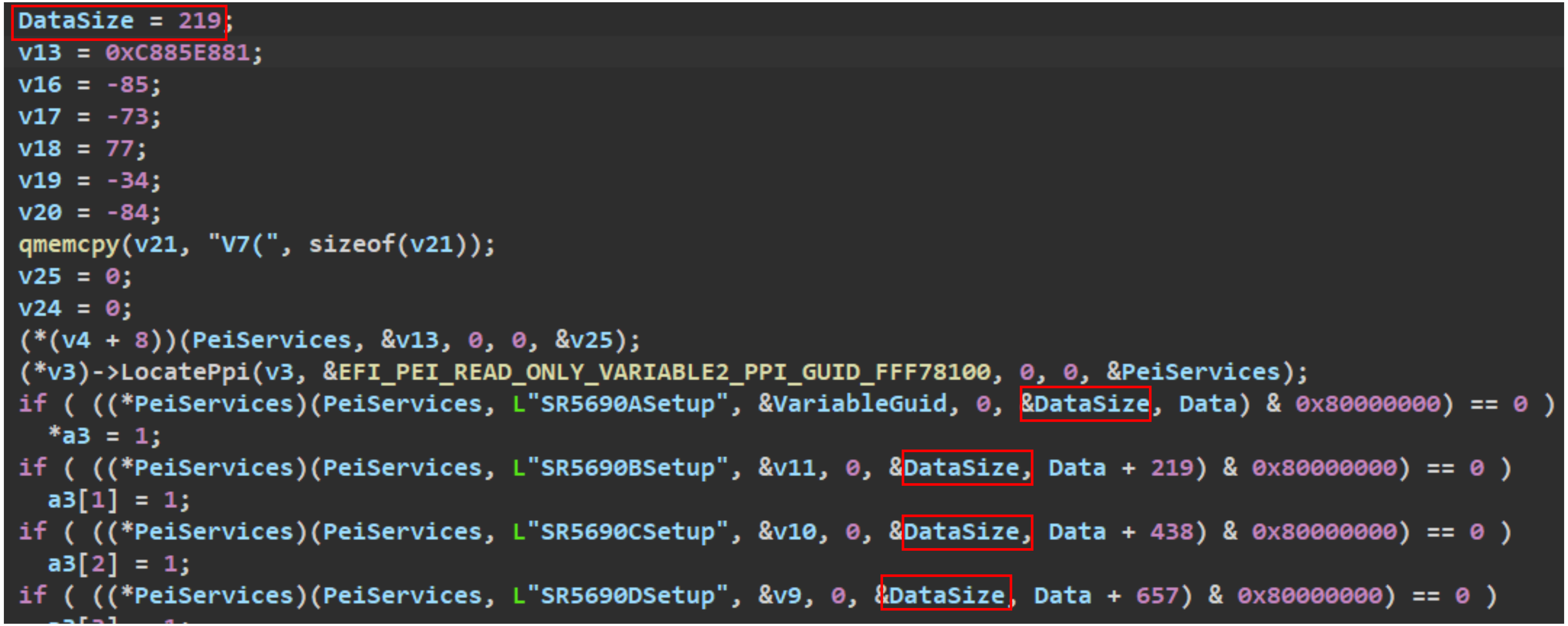


```
for (auto j = 0; j < 15; j++) {
    address = prev_head(address, startAddress);
    decode_insn(&insn, address);
    if (insn.itype == NN_lea && insn.ops[0].type == o_reg &&
        insn.ops[0].reg == arg5_reg &&
        insn.ops[1].type == o_displ) {
        prev_datasize_addr = insn.ops[1].addr;
        datasize_addr_found = true;
        break;
    }
}
```

PPI GetVariable vuln search: statistics

Vendor Name	Avg number calls per firmware	Avg number of vuln pattern is triggered per firmware
ASRock	122	12
ASUS	176	17
Lenovo	77	8

PPI GetVariable vuln search: results



```
DataSize = 219;
v13 = 0xC885E881;
v16 = -85;
v17 = -73;
v18 = 77;
v19 = -34;
v20 = -84;
qmemcpy(v21, "V7(", sizeof(v21));
v25 = 0;
v24 = 0;
(*(v4 + 8))(PeiServices, &v13, 0, 0, &v25);
(*v3)->LocatePpi(v3, &EFI_PEI_READ_ONLY_VARIABLE2_PPI_GUID_FFF78100, 0, 0, &PeiServices);
if ( ((*PeiServices)(PeiServices, L"SR5690ASetup", &VariableGuid, 0, &DataSize, Data) & 0x80000000) == 0 )
  *a3 = 1;
if ( ((*PeiServices)(PeiServices, L"SR5690BSetup", &v11, 0, &DataSize, Data + 219) & 0x80000000) == 0 )
  a3[1] = 1;
if ( ((*PeiServices)(PeiServices, L"SR5690CSetup", &v10, 0, &DataSize, Data + 438) & 0x80000000) == 0 )
  a3[2] = 1;
if ( ((*PeiServices)(PeiServices, L"SR5690DSetup", &v9, 0, &DataSize, Data + 657) & 0x80000000) == 0 )
  a3[3] = 1;
```

Vuln hunting at scale: results and statistics

Vuln hunting at scale: vendor stats

Attack surface and potential vulnerabilities: average numbers per 1 firmware for each of the 3 vendors

Vendor name	SMI handlers num.	Potential SMM callouts num.	PEI GetVariable calls num.	Potential PEI GetVariable vuln num	DXE GetVariable calls num.	Potential DXE GetVariable vuln num	SMM GetVariable calls num.	Potential SMM GetVariable vuln num
ASRock	51	72	122	12	735	2	8	0
ASUS	42	80	176	17	697	5	7	0.003
Lenovo	20	3	78	8	466	2	15	1

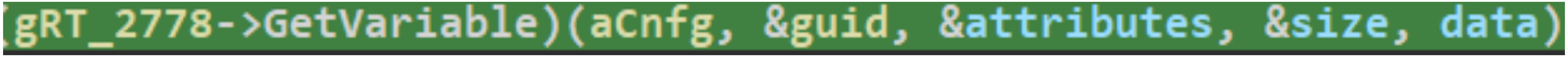
Vuln hunting at scale: Attack Surface stats

Attack surface and potential vulnerabilities: average numbers per 1 firmware for each boot phase (PEI/SMM/DXE)

Metric	PEI	SMM	DXE
GetVariable	152.00	8.00	695.00
GetVar Vuln	15.00	0.06	4.00

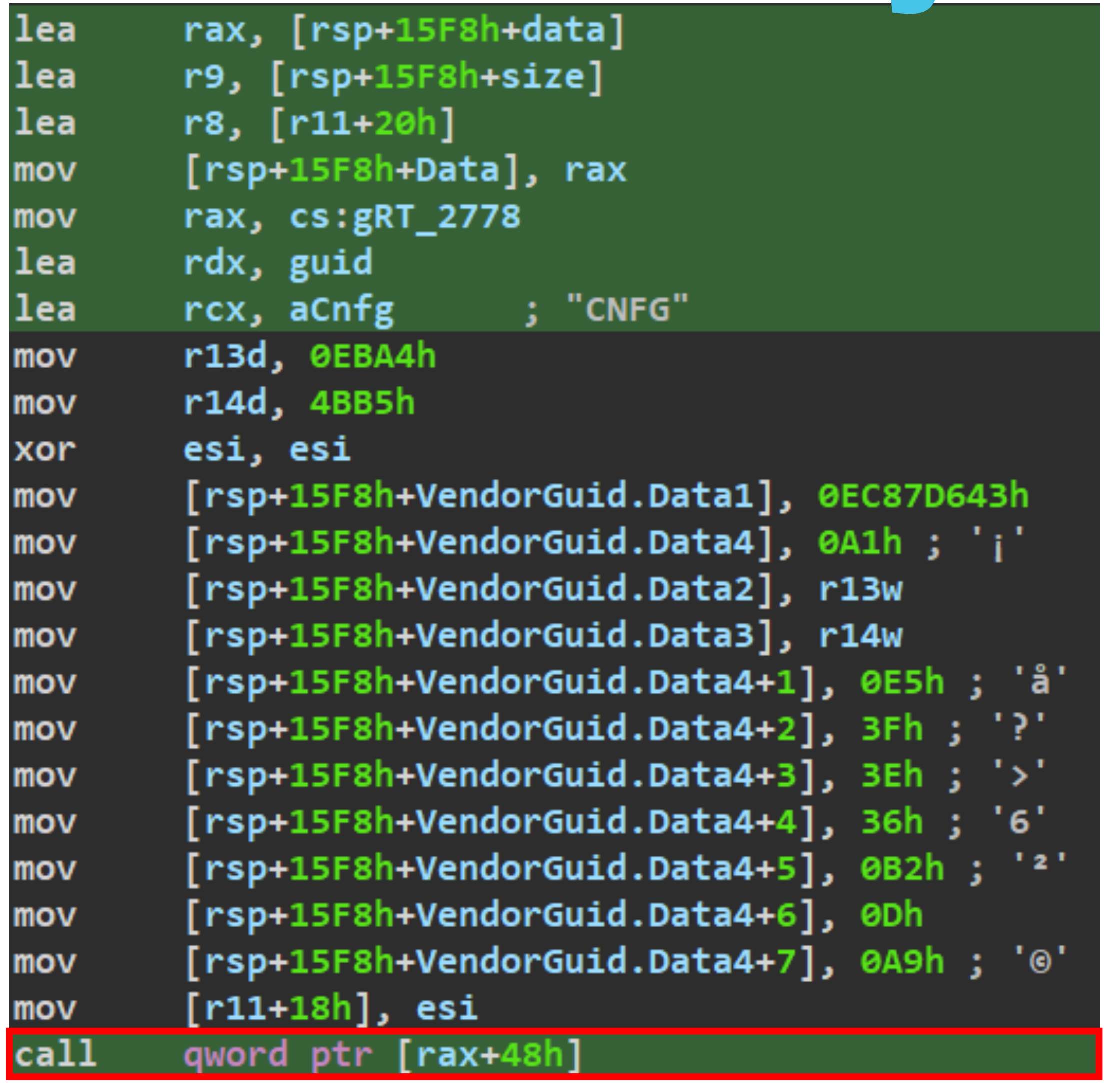
efiXplorer: future plans

Decompiler



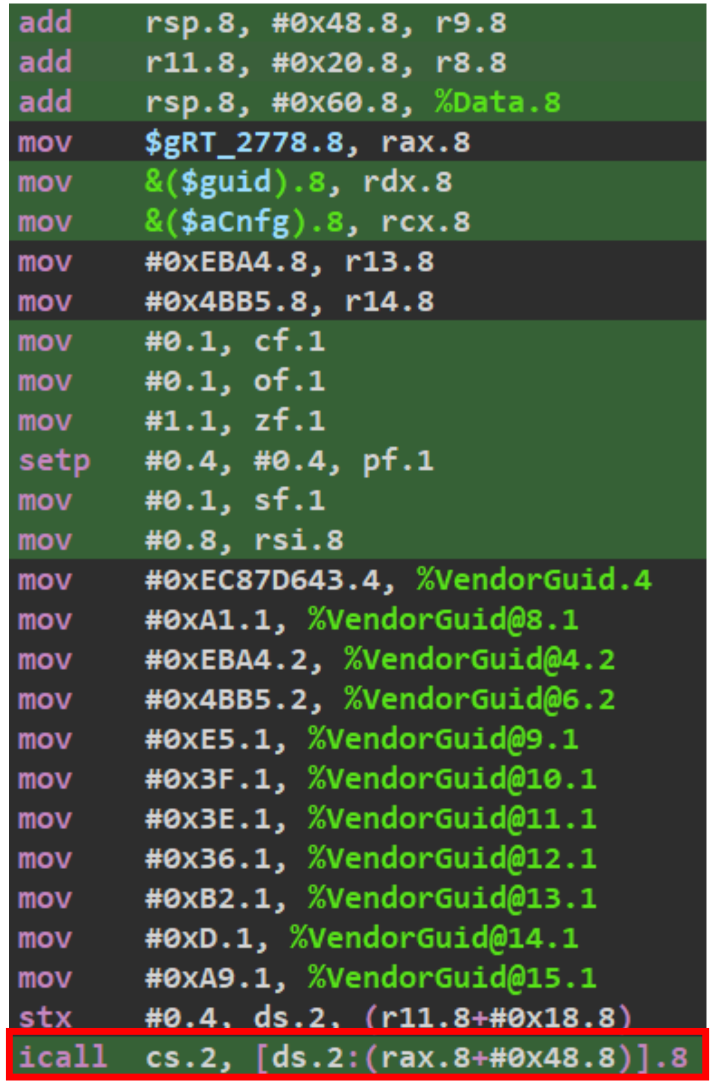
```
(gRT_2778->GetVariable)(aCnfg, &guid, &attributes, &size, data)
```

Disassembly

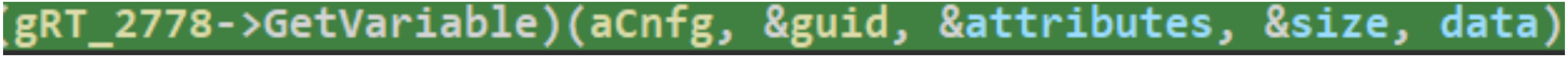


```
lea     rax, [rsp+15F8h+data]
lea     r9, [rsp+15F8h+size]
lea     r8, [r11+20h]
mov     [rsp+15F8h+Data], rax
mov     rax, cs:gRT_2778
lea     rdx, guid
lea     rcx, aCnfg          ; "CNFG"
mov     r13d, 0EBA4h
mov     r14d, 4BB5h
xor     esi, esi
mov     [rsp+15F8h+VendorGuid.Data1], 0EC87D643h
mov     [rsp+15F8h+VendorGuid.Data4], 0A1h ; '¡'
mov     [rsp+15F8h+VendorGuid.Data2], r13w
mov     [rsp+15F8h+VendorGuid.Data3], r14w
mov     [rsp+15F8h+VendorGuid.Data4+1], 0E5h ; 'å'
mov     [rsp+15F8h+VendorGuid.Data4+2], 3Fh ; '?'
mov     [rsp+15F8h+VendorGuid.Data4+3], 3Eh ; '>'
mov     [rsp+15F8h+VendorGuid.Data4+4], 36h ; '6'
mov     [rsp+15F8h+VendorGuid.Data4+5], 0B2h ; '²'
mov     [rsp+15F8h+VendorGuid.Data4+6], 0Dh
mov     [rsp+15F8h+VendorGuid.Data4+7], 0A9h ; '©'
mov     [r11+18h], esi
call    qword ptr [rax+48h]
```

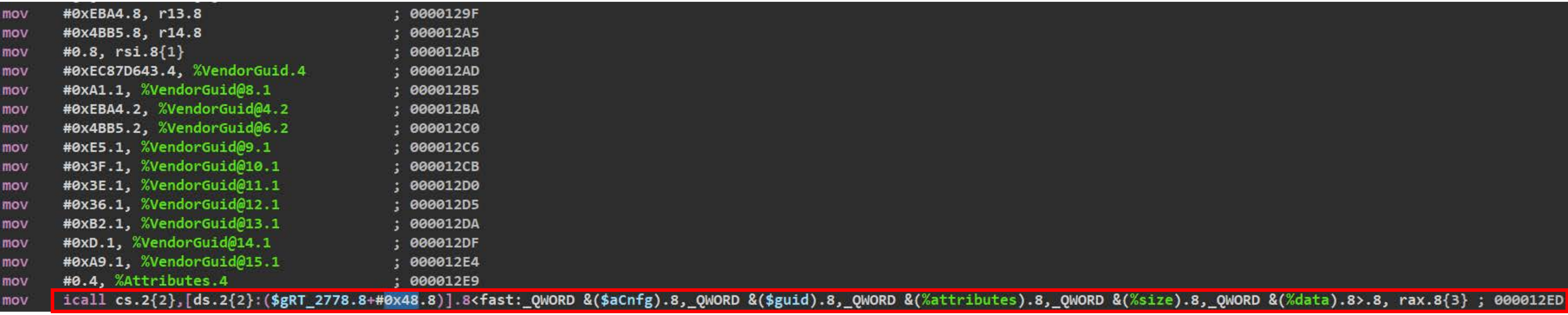
Microcode



```
add    rsp.8, #0x48.8, r9.8
add    r11.8, #0x20.8, r8.8
add    rsp.8, #0x60.8, %Data.8
mov    $gRT_2778.8, rax.8
mov    &($guid).8, rdx.8
mov    &($aCnfg).8, rcx.8
mov    #0xEBA4.8, r13.8
mov    #0x4BB5.8, r14.8
mov    #0.1, cf.1
mov    #0.1, of.1
mov    #1.1, zf.1
setp   #0.4, #0.4, pf.1
mov    #0.1, sf.1
mov    #0.8, rsi.8
mov    #0xEC87D643.4, %VendorGuid.4
mov    #0xA1.1, %VendorGuid@8.1
mov    #0xEBA4.2, %VendorGuid@4.2
mov    #0x4BB5.2, %VendorGuid@6.2
mov    #0xE5.1, %VendorGuid@9.1
mov    #0x3F.1, %VendorGuid@10.1
mov    #0x3E.1, %VendorGuid@11.1
mov    #0x36.1, %VendorGuid@12.1
mov    #0xB2.1, %VendorGuid@13.1
mov    #0xD.1, %VendorGuid@14.1
mov    #0xA9.1, %VendorGuid@15.1
stx    #0.4, ds.2, (r11.8+#0x18.8)
icall  cs.2, [ds.2:(rax.8+#0x48.8)].8
```

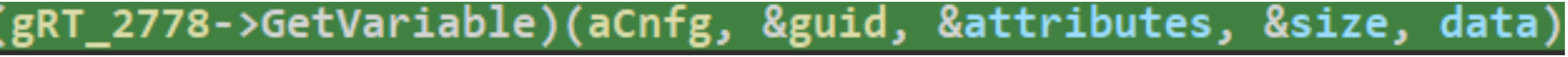


```
(gRT_2778->GetVariable)(aCnfg, &guid, &attributes, &size, data)
```

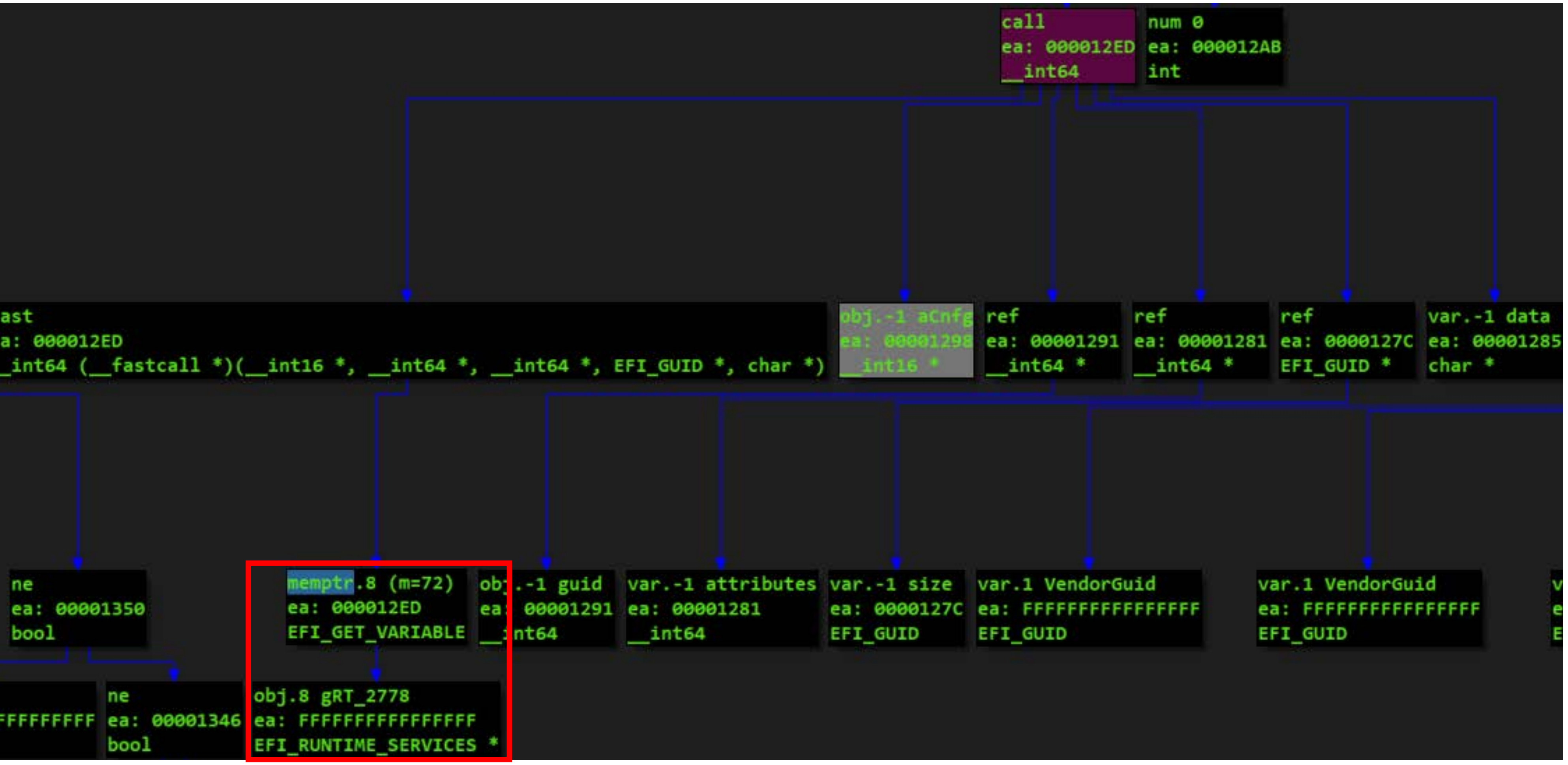


```
mov     #0xEBA4.8, r13.8                            ; 0000129F
mov     #0x4BB5.8, r14.8                            ; 000012A5
mov     #0.8, rsi.8{1}                              ; 000012AB
mov     #0xEC87D643.4, %VendorGuid.4                ; 000012AD
mov     #0xA1.1, %VendorGuid@8.1                    ; 000012B5
mov     #0xEBA4.2, %VendorGuid@4.2                  ; 000012BA
mov     #0x4BB5.2, %VendorGuid@6.2                  ; 000012C0
mov     #0xE5.1, %VendorGuid@9.1                    ; 000012C6
mov     #0x3F.1, %VendorGuid@10.1                   ; 000012CB
mov     #0x3E.1, %VendorGuid@11.1                   ; 000012D0
mov     #0x36.1, %VendorGuid@12.1                   ; 000012D5
mov     #0xB2.1, %VendorGuid@13.1                   ; 000012DA
mov     #0xD.1, %VendorGuid@14.1                    ; 000012DF
mov     #0xA9.1, %VendorGuid@15.1                   ; 000012E4
mov     #0.4, %Attributes.4                         ; 000012E9
mov     icall cs.2{2},[ds.2{2}:($gRT_2778.8+#0x48.8)].8<fast:_QWORD &($aCnfg).8,_QWORD &($guid).8,_QWORD &(%attributes).8,_QWORD &(%size).8,_QWORD &(%data).8>.8, rax.8{3} ; 000012ED
```

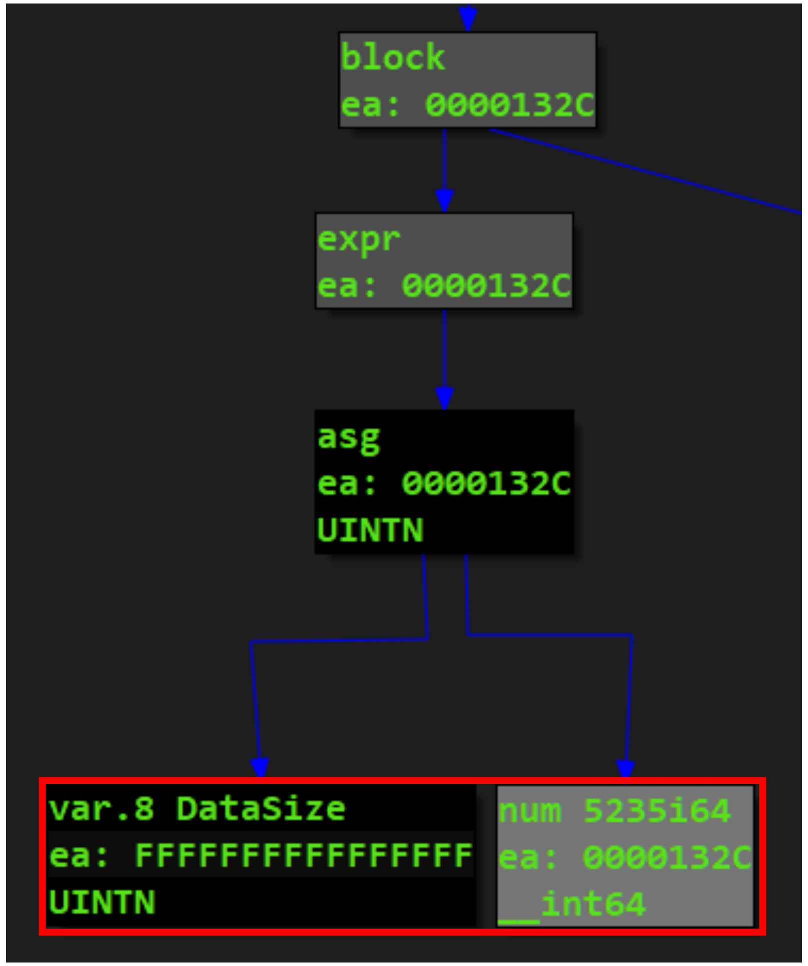
Power of dataflow analysis



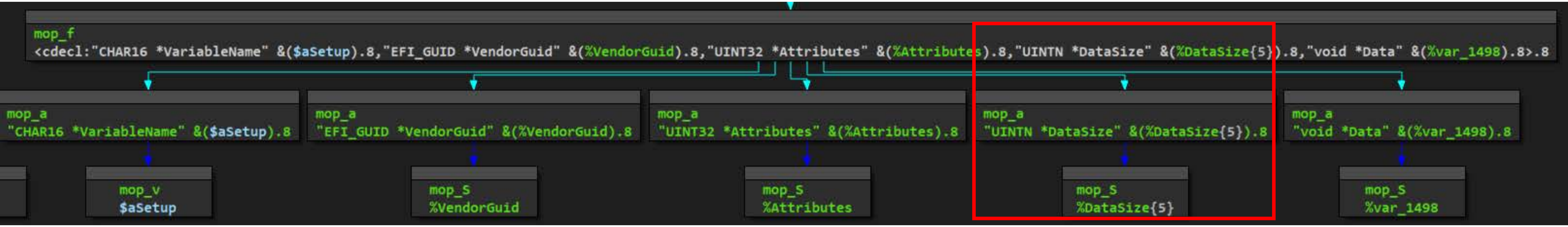
```
(gRT_2778->GetVariable)(aCnfg, &guid, &attributes, &size, data)
```



```
call
ea: 000012ED
__int64

num 0
ea: 000012AB
int

ast
a: 000012ED
_int64 (__fastcall *)(__int16 *, __int64 *, __int64 *, EFI_GUID *, char *)

obj.-1 aCnfg
ea: 00001298
__int16 *

ref
ea: 00001291
__int64 *

ref
ea: 00001281
__int64 *

ref
ea: 0000127C
EFI_GUID *

var.-1 data
ea: 00001285
char *

ne
ea: 00001350
bool

memptr.8 (m=72)
ea: 000012ED
EFI_GET_VARIABLE

obj.-1 guid
ea: 00001291
__int64

var.-1 attributes
ea: 00001281
__int64

var.-1 size
ea: 0000127C
EFI_GUID

var.1 VendorGuid
ea: FFFFFFFFFFFFFFFF
EFI_GUID

var.1 VendorGuid
ea: FFFFFFFFFFFFFFFF
EFI_GUID

FFFFFFFF

ne
ea: 00001346
bool

obj.8 gRT_2778
ea: FFFFFFFFFFFFFFFF
EFI_RUNTIME_SERVICES *
```



```
block
ea: 0000132C

expr
ea: 0000132C

asg
ea: 0000132C
UINTN

var.8 DataSize
ea: FFFFFFFFFFFFFFFF
UINTN

num 5235i64
ea: 0000132C
__int64
```



```
mop_f
<cdecl:"CHAR16 *VariableName" &($aSetup).8,"EFI_GUID *VendorGuid" &(%VendorGuid).8,"UINT32 *Attributes" &(%Attributes).8,"UINTN *DataSize" &(%DataSize{5}).8,"void *Data" &(%var_1498).8>.8

mop_a
"CHAR16 *VariableName" &($aSetup).8

mop_a
"EFI_GUID *VendorGuid" &(%VendorGuid).8

mop_a
"UINT32 *Attributes" &(%Attributes).8

mop_a
"UINTN *DataSize" &(%DataSize{5}).8

mop_a
"void *Data" &(%var_1498).8

mop_v
$aSetup

mop_S
%VendorGuid

mop_S
%Attributes

mop_S
%DataSize{5}

mop_S
%var_1498
```

Conclusion

✓ Well-tuned heuristics work surprisingly well for UEFI security analysis

- recovery of important structures
- automated attack surface measurement (!)
- automated potential vulnerability finding (!)

✓ Firmware vendors have worked on attack surface reduction, but well-known attack vectors is still a problem in 2020, such as: SMM callouts, GetVariable misuse

✓ We need more open, usable, and working instruments for UEFI security, including: Vuln research, RE and automation, Forensics and Data Science

✓ It's about right time for a much broader audience to look into the problem of UEFI implants

✓ Who knows what else we'll find there?

black hat EUROPE 2020

DECEMBER 9-10

BRIEFINGS

Thank you!

@matrosov, @isciurus,

@yeggorv, @p41ll