Quarkslab

For Science!

Using an Unimpressive Bug in EDK II To Do Some Fun Exploitation

Gwaby

Whoami

- Reverse Engineer
  - @ Quarkslab [*]
- Desktop & Virtualization team
  - Vuln research
  - R&D

Agenda

1. Some Generalities (a.k.a Boring Section)
2. The Bug (a.k.a Kind of Okay Stuff)
3. Exploitation (a.k.a Wanna Be Fun Part)

[*]: We're hiring & the job is fun!

Introduction

Some Generalities

UEFI

- Unified Extensible Firmware Interface
- Replace old (16-bit x86) BIOS technology
  - Initialize the platform hardware
  - Report information to the OS
- EDK II
  - Maintained by TianoCore
  - Main implementation of UEFI standard
  - Code base for various OEMs
  - Open source, mainly in C

> *https://github.com/tianocore/edk2*

System Management Mode (SMM)

- Special purpose and isolated operating mode (**Ring -2**)
  - Defined in IA CPU architecture
  - Highest privilege
    - Greatest access to system memory and hardware resources
- Handle critical functions
  - Partially in charge of protecting the boot using hardware resources
- Code and data running in **SMM** located in **SMRAM**
  - Special protected memory region

System Management Mode (SMM)

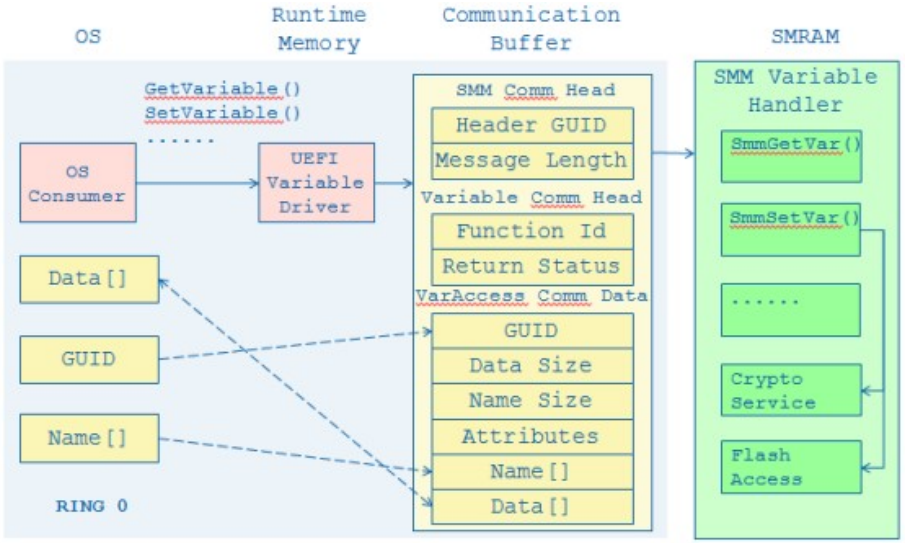
Entering and Exiting SMM

- **S**ystem **M**anagement **I**nterrupt (SMI)
  - CPU switchs into SMM
  - Jump to pre-defined entry vector
  - Save previous context (`save states`)
  - Returns to normal world with `RSM` instruction
- 2 ways of communications between SMM and normal world
  - through ACPI table
  - through `EFI_SMM_COMMUNICATION_PROTOCOL` protocol
    - API-like function in EFI

System Management Mode (SMM)

EFI_SMM_COMMUNICATION_PROTOCOL protocol

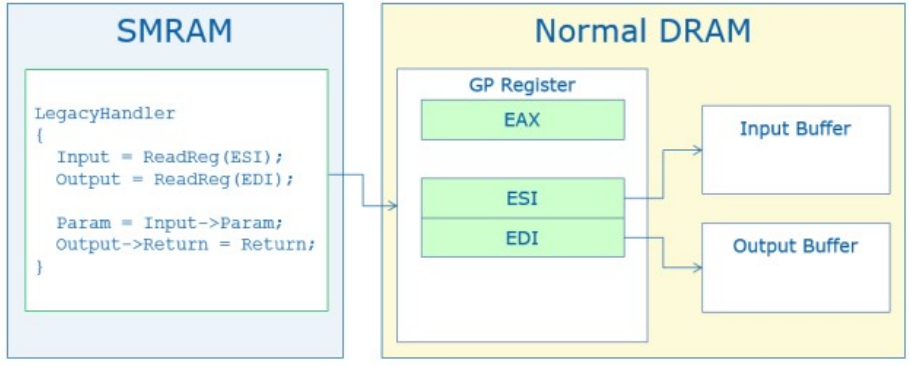
- Provides runtime communication services between drivers outside of SMM and a SMI handler



System Management Mode (SMM)

UEFI ACPI table

- Describes a special software **SMI**
- Generated using I/O resources or CPU instructions
  - Used by any non-firmware component
- Data address is recorded in the ACPI table or via a general purpose register



Tcg2Smm

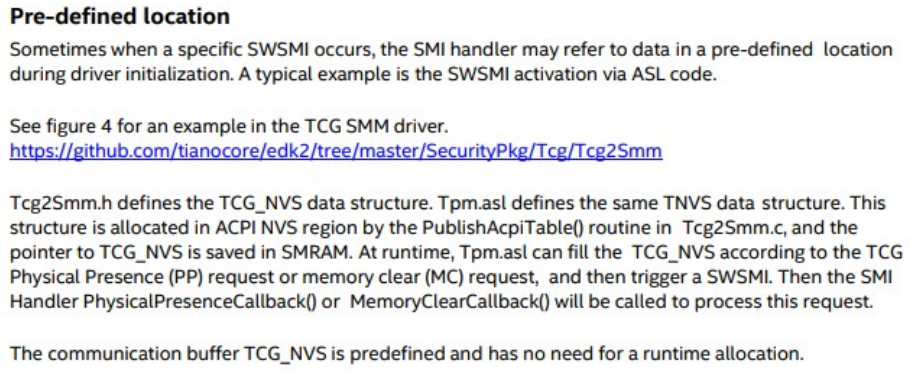
Securing SMM Communications

- Two "main" best practices when developping a SMI handler
  - Copy of the comm buffer in a temporary variable
    - To prevent TOC/TOU attacks
  - Use `SmmIsBufferOutsideSmmValid()` API
    - check if a comm buffer is valid per processor architecture and not overlap with SMRAM.

The Bug!

Tcg2Smm

- (Continuing) reading about SMM communication
  - *A Tour Beyond BIOS Secure SMM Communication* white paper (page 8)



Pre-defined location

Sometimes when a specific SWSMI occurs, the SMI handler may refer to data in a pre-defined location during driver initialization. A typical example is the SWSMI activation via ASL code.

See figure 4 for an example in the TCG SMM driver.
https://github.com/tianocore/edk2/tree/master/SecurityPkg/Tcg/Tcg2Smm

Tcg2Smm.h defines the TCG_NVS data structure. Tpm.asl defines the same TNVS data structure. This structure is allocated in ACPI NVS region by the PublishAcpiTable() routine in Tcg2Smm.c, and the pointer to TCG_NVS is saved in SMRAM. At runtime, Tpm.asl can fill the TCG_NVS according to the TCG Physical Presence (PP) request or memory clear (MC) request, and then trigger a SWSMI. Then the SMI Handler PhysicalPresenceCallback() or MemoryClearCallback() will be called to process this request.

The communication buffer TCG_NVS is predefined and has no need for a runtime allocation.

- **TL.DR.**
  - Data structure allocated in ACPI NVS region by `PublishAcpiTable()`
  - Pointer saved in SMRAM and used in other SWSMI
  - Check the git (`Tcg2Smm.c`) for more details
  - (*ASL == ACPI Source Language*)

Tcg2Smm

```
$ git clone edk2
$ cd edk2/SecurityPkg/Tcg/Tcg2Smm
$ grep PublishAcpiTable Tcg2Smm.c
```

Tcg2Smm

```
$ git clone edk2
$ cd edk2/SecurityPkg/Tcg/Tcg2Smm
$ grep PublishAcpiTable Tcg2Smm.c
$
```

```
...  :|
```

- No `PublishAcpiTable()` in `Tcg2Smm.c`

Tcg2Smm

```
$ git clone edk2
$ cd edk2/SecurityPkg/Tcg/Tcg2Smm
$ grep PublishAcpiTable Tcg2Smm.c
$
```

```
... :|
```

- No `PublishAcpiTable()` in `Tcg2Smm.c`
- But it was here!
  - commit cd64301 on Jun 8, 2016

```
EFI_STATUS PublishAcpiTable (VOID)
{
// [...]

mTcgNvs = AssignOpRegion (Table, SIGNATURE_32 ('T', 'N', 'V', 'S'),
                          (UINT16) sizeof (TCG_NVS));
ASSERT (mTcgNvs != NULL);
}
```

Tcg2Smm

Ok, but how about now?

```
//Communication service SMI Handler entry.
//This handler takes requests to exchange Mmi channel and Nvs address between MM and
//DXE.

//Caution: This function may receive untrusted input.
//Communicate buffer and buffer size are external input, so this function will do basic
//validation.
EFI_STATUS EFIAPI TpmNvsCommunciate (/* [...] */)
{
  // [...]
  if (!IsBufferOutsideMmValid ((UINTN)CommBuffer, TempCommBufferSize)) {
    return EFI_ACCESS_DENIED;
  }
  CommParams = (TPM_NVS_MM_COMM_BUFFER *)CommBuffer;
  mTcgNvs = (TCG_NVS *)(UINTN)CommParams->TargetAddress;
  // [...]
}
```

- Done through `EFI_SMM_COMMUNICATION_PROTOCOL` protocol

Tcg2Smm

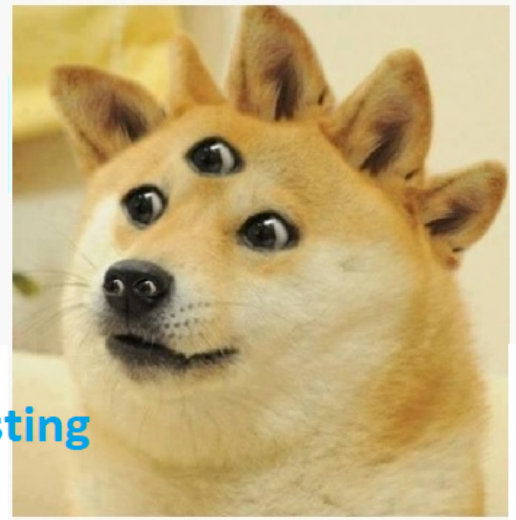
Ok, but how about now?

```
//Communication service SMI Handler entry.
//This handler takes requests to exchange Mmi channel and Nvs address between MM and
//DXE.

//Caution: This function may receive untrusted input.
//Communicate buffer and buffer size are external input, so this function will do basic
//validation.
EFI_STATUS EFIAPI TpmNvsCommunciate (/* [...] */)
{
  // [...]
  if (!IsBufferOutsideMmValid ((UINTN)CommBuffer, TempCommBufferSize)) {
    return EFI_ACCESS_DENIED;
  }
  CommParams = (TPM_NVS_MM_COMM_BUFFER *)CommBuffer;
  mTcgNvs = (TCG_NVS *)(UINTN)CommParams->TargetAddress;
  // [...]
}
```

Wow!

Much interesting



TCG_NVS

```
#pragma pack(1)

typedef struct {
  PHYSICAL_PRESENCE_NVS PhysicalPresence;
  MEMORY_CLEAR_NVS   MemoryClear;
  UINT32        PPRequestUserConfirm;
  UINT32        TpmIrqNum;
  BOOLEAN       IsShortFormPkgLength;
} TCG_NVS;
```

- Used by two other SWSMI callbacks
  - `PhysicalPresence`
  - `MemoryClear`

```
typedef struct {
  UINT8     SoftwareSmi;
  UINT32    Parameter;
  UINT32    Response;
  UINT32    Request;
  UINT32    RequestParameter;
  UINT32    LastRequest;
  UINT32    ReturnCode;
} PHYSICAL_PRESENCE_NVS;

typedef struct {
  UINT8     SoftwareSmi;
  UINT32    Parameter;
  UINT32    Request;
  UINT32    ReturnCode;
} MEMORY_CLEAR_NVS;
```

Tcg2Smm SWSMI callbacks

- Can be resumed as two big switchs
  - Actions depends on the `Parameter` field
- Example

```
EFI_STATUS EFIAPI MemoryClearCallback (/* [...] */)
{
  EFI_STATUS  Status;
  UINTN       DataSize;
  UINT8       MorControl;

  mTcgNvs->MemoryClear.ReturnCode = MOR_REQUEST_SUCCESS;
  if (mTcgNvs->MemoryClear.Parameter == ACPI_FUNCTION_DSM_MEMORY_CLEAR_INTERFACE) {
    MorControl = (UINT8)mTcgNvs->MemoryClear.Request;
  } else if (mTcgNvs->MemoryClear.Parameter == ACPI_FUNCTION_PTS_CLEAR_MOR_BIT) {
    // [...]
  }
}
```

Tcg2Smm SWSMI callbacks - Outcome

PhysicalPresence callback

PHYSICAL_PRESENCE_NVS.Parameter = 2 or 7	
PHYSICAL_PRESENCE_NVS.Request	0x000000XX
PHYSICAL_PRESENCE_NVS.ReturnCode	0x00000001
Leak few bytes in TcgPhysicalPresence nvs variable	

PHYSICAL_PRESENCE_NVS.Parameter = 5	
PHYSICAL_PRESENCE_NVS.Response	0xXXXXXXXX
PHYSICAL_PRESENCE_NVS.LastRequest	0x000000XX
PHYSICAL_PRESENCE_NVS.ReturnCode	0x00000001

PHYSICAL_PRESENCE_NVS.Parameter = 8	
PHYSICAL_PRESENCE_NVS.ReturnCode	0x00000001

MemoryClear callback

MEMORY_CLEAR_NVS.Parameter = 1	
MEMORY_CLEAR_NVS.ReturnCode	0x00000000

MEMORY_CLEAR_NVS.Parameter = 2	
MEMORY_CLEAR_NVS.ReturnCode	0x00000000

MEMORY_CLEAR_NVS.Parameter = ??	
MEMORY_CLEAR_NVS.ReturnCode	0x00000001

Where XX indicates that the value is retrieved from a non-volatile variable (`Tcg2PhysicalPresence`).

Tcg2Smm SWSMI callbacks - Outcome

PhysicalPresence callback

PHYSICAL_PRESENCE_NVS.Parameter = 2 or 7	
PHYSICAL_PRESENCE_NVS.Request	0x000000XX
PHYSICAL_PRESENCE_NVS.ReturnCode	0x00000001
Leak few bytes in TcgPhysicalPresence nvs variable	

PHYSICAL_PRESENCE_NVS.Parameter = 5	
PHYSICAL_PRESENCE_NVS.Response	0xXXXXXXXX
PHYSICAL_PRESENCE_NVS.LastRequest	0x000000XX
PHYSICAL_PRESENCE_NVS.ReturnCode	0x00000001

PHYSICAL_PRESENCE_NVS.Parameter = 8	
PHYSICAL_PRESENCE_NVS.ReturnCode	0x00000001

Where XX indicates that the value is retrieved from a non-volatile variable (`Tcg2PhysicalPresence`).

MemoryClear callback

MEMORY_CLEAR_NVS.Parameter = 1	
MEMORY_CLEAR_NVS.ReturnCode	0x00000000

MEMORY_CLEAR_NVS.Parameter = 2	
MEMORY_CLEAR_NVS.ReturnCode	0x00000000

MEMORY_CLEAR_NVS.Parameter = ??	
MEMORY_CLEAR_NVS.ReturnCode	0x00000001

Woot loot!

Sooooo... Arbitrary write in SMRAM

==> God mode (almost) activated?? \o/

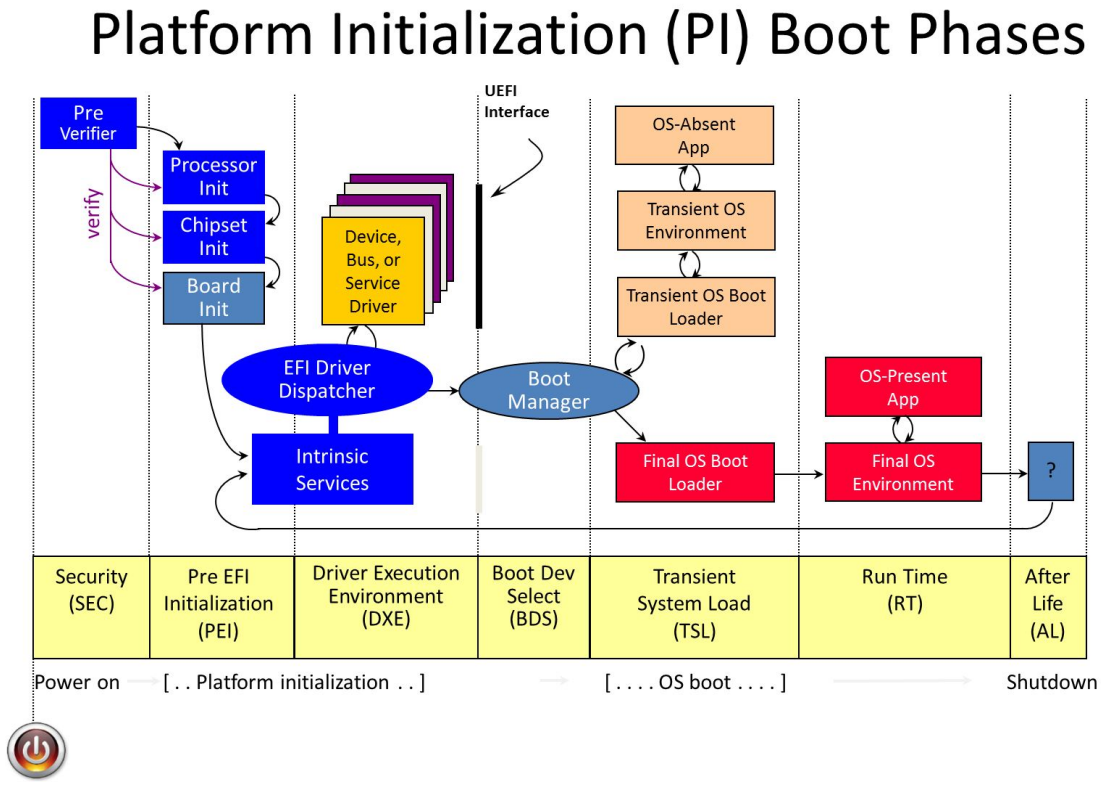
Tcg2Smm SWSMI callbacks - Outcome

Errrh, yes and no...

- `TpmNvsCommunciate` SMI unregistered when `gEfiMmReadyToLockProtocolGuid` is published :'(

- Registers notification callback for the "ready to lock" protocol
  - Prevent use by the third party code
  - Happens just after the SMM End of DXE Protocol
- Completly removes the SMI handler
  - Cannot modify the `mTcgNvs` after that

UEFI Boot Phases



UEFI Boot Phases

Let's forget about that

- Primitive quite limited
  - Can only write 4 fixed bytes
- Depending on the value present in the "Parameter" field
  - Easiest to control -> default case while triggering `MemoryClearCallback`
  - Write `0x00000001` (almost) anywhere in SMRAM

Predicates

- We have another vuln allowing us to block the deletion of the SMI handler
- SecureBoot is disabled
  - We can load an arbitrary UEFI application

Exploitation

Target

- We need to find a firmware with the flaw inside...

Target

- We need to find a firmware with the flaw inside...
- We can just use OVMF

Target

- We need to find a firmware with the flaw inside...
- We can just use OVMF
- The SWSMI are not implemented in OVMF

Target

- We need to find a firmware with the flaw inside...
- We can just use OVMF
- The SWSMI are not implemented in OVMF
- Let's use some development board

Target

- We need to find a firmware with the flaw inside...
- We can just use OVMF
- The SWSMI are not implemented in OVMF
- Let's use some development board
- This is related to a hardware component

Target

- We need to find a firmware with the flaw inside...
- We can just use OVMF
- The SWSMI are not implemented in OVMF
- Let's use some development board
- This is related to a hardware component
- Let's buy a f*cking TPM for the dev board then!

Target

- We need to find a firmware with the flaw inside...
- We can just use OVMF
- The SWSMI are not implemented in OVMF
- Let's use some development board
- This is related to a hardware component
- Let's buy a f*cking TPM for the dev board then!
- Most of the boards are sold out and it is so looong to order stuff, I want to test it naow! ;_;

Target

- We need to find a firmware with the flaw inside...
- We can just use OVMF
- The SWSMI are not implemented in OVMF
- Let's use some development board
- This is related to a hardware component
- Let's buy a f*cking TPM for the dev board then!
- Most of the boards are sold out and it is so looong to order stuff, I want to test it naow! ;_;

> **F*ck this shit**
>
> - Let's adapt the SMM driver for OVMF

Target

Adaptation

- Change both SMI callbacks into root MMI handlers
  - Called for every SMI event
- Hardcode the SMI IDs in the SMM driver
- Filter the requests by checking the SMI IDs
- Change the load dependancies
  - Remove the SWSMI dispatcher
  - Add the module responsible for catching the SMI IDs

4-byte Write Primitive to Arbitrary Read-Write Primitive

What we have

- Can write `0x00000001` anywhere in SMRAM
  - Change the value of `mTcgNvs` with the SMI callback
  - Trigger the `MemoryClear` SWSMI callback

4-byte Write Primitive to Arbitrary Read-Write Primitive

What we have

- Can write `0x00000001` anywhere in SMRAM
  - Change the value of `mTcgNvs` with the SMI callback
  - Trigger the `MemoryClear` SWSMI callback

Goal

- Read and Write anything anywhere in SMRAM \o/

4-byte Write Primitive to Arbitrary Read-Write Primitive

What we have

- Can write `0x00000001` anywhere in SMRAM
  - Change the value of `mTcgNvs` with the SMI callback
  - Trigger the `MemoryClear` SWSMI callback

Goal

- Read and Write anything anywhere in SMRAM \o/

Restriction

- Only use what is provided by EDK2

4-byte Write Primitive to Arbitrary Read-Write Primitive

What we have

- Can write `0x00000001` anywhere in SMRAM
  - Change the value of `mTcgNvs` with the SMI callback
  - Trigger the `MemoryClear` SWSMI callback

Goal

- Read and Write anything anywhere in SMRAM \o/

Restriction

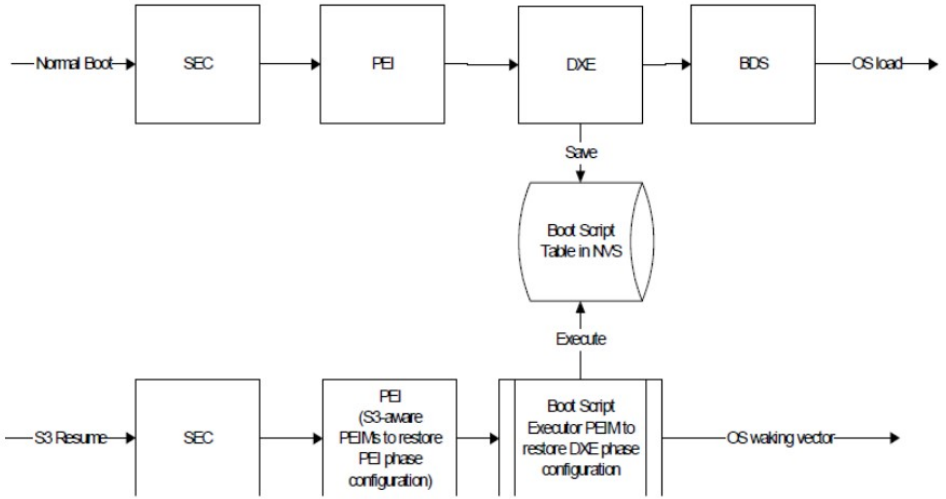
- Only use what is provided by EDK2

- Let's corrupt some global variables
  - SmmLockBox!

SmmLockBox

S3 Resume

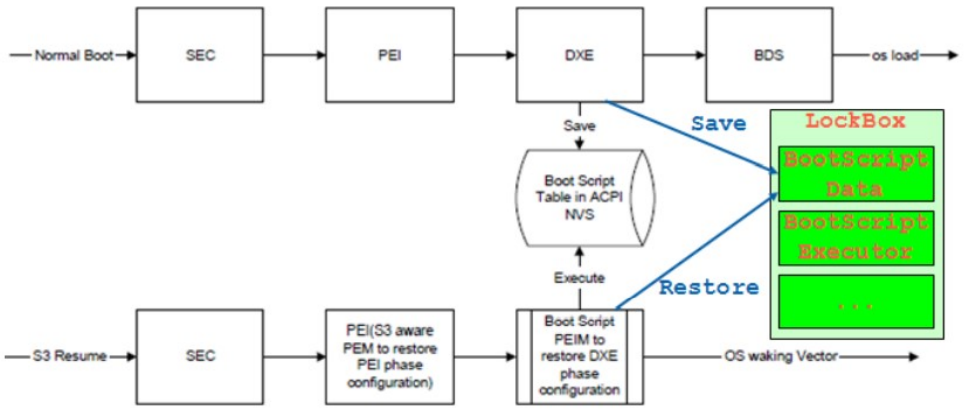
- Power saving feature
  - Set of power state of transition (defined in ACPI specification)
  - 4 states in the sleeping group
  - S3 sleeping state -> "suspend to memory"
- Restore the platform to its pre-boot configuration
  - Avoid dealing with the DXE phase



SmmLockBox

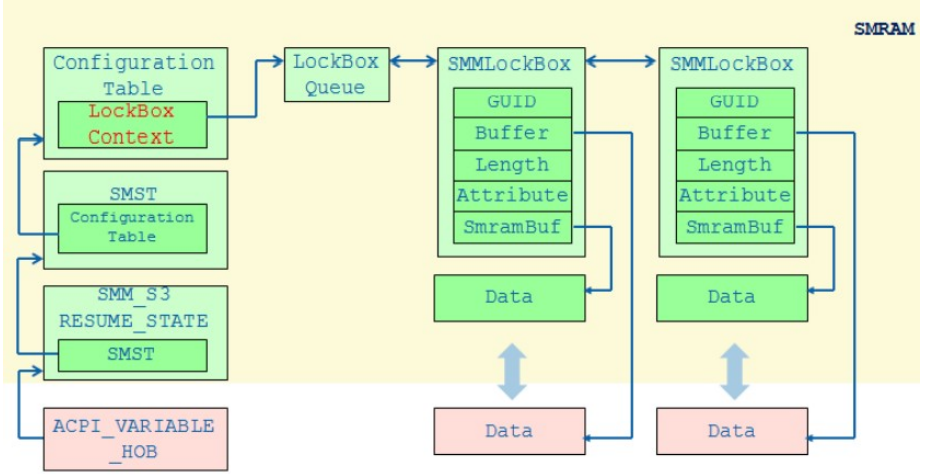
LockBox

- Data stored in memory might be tampered if left unprotected
- Container that maintains the integrity of data
  - But not the confidentiality
- EDKII implementation based on SMM



SmmLockBox

- Useful API
  - `SaveLockBox()` - copy data to LockBox
  - `UpdateLockBox()` - update data in LockBox
  - `SetLockBoxAttributes()` - set LockBox attributes
  - `RestoreLockBox()` – get data from LockBox and copy it in a buffer, or at its original address
- Reachable throught the communicate protocol
  - `gEfiSmmLockBoxCommunicationGuid`



> Could become a perfect R/W primitive <3

SmmLockBox

4-byte Write Primitive to Arbitrary Read-Write Primitive

- Want to use SmmLockBox API to R/W in SMRAM

Problem

- SmmLockBox API protected with `SmmIsBufferOutsideSmmValid`
- `SaveLockBox()`, `SmmLockBoxSetAttributes` & `UpdateLockBox()` locked at the end of DXE phase

SmmIsBufferOutsideSmmValid

```
BOOLEAN EFIAPI SmmIsBufferOutsideSmmValid (
  IN EFI_PHYSICAL_ADDRESS  Buffer,
  IN UINT64                Length
  )
```

- Implemented in `SmmMemLib` library
  - Statically linked in SMM modules using it
- Ensures that the buffer:
  1. Is within a valid range of address
  2. Doesn't overlap with SMRAM
  3. Is inside the region intended for communication buffer
  4. Is not in a memory region labelled as "untested"
  5. Is not on a RO memory page

SmmIsBufferOutsideSmmValid

```
BOOLEAN EFIAPI SmmIsBufferOutsideSmmValid (
  IN EFI_PHYSICAL_ADDRESS  Buffer,
  IN UINT64                Length
  )
```

- Implemented in `SmmMemLib` library
  - Statically linked in SMM modules using it
- Ensures that the buffer:
  1. Is within a valid range of address
  2. Doesn't overlap with SMRAM
  3. Is inside the region intended for communication buffer
  4. Is not in a memory region labelled as "untested"
  5. Is not on a RO memory page

SmmIsBufferOutsideSmmValid

SMRAM Overlap Verification

```
BOOLEAN EFIAPI SmmIsBufferOutsideSmmValid (
  IN EFI_PHYSICAL_ADDRESS  Buffer,
  IN UINT64                Length
  )
{
// [...]
  for (Index = 0; Index < mSmmMemLibInternalSmramCount; Index++) {
    if (((Buffer >= mSmmMemLibInternalSmramRanges[Index].CpuStart)
    && (Buffer < mSmmMemLibInternalSmramRanges[Index].CpuStart + mSmmMemLibInternalSmramRanges[Index].PhysicalSize))
    || ((mSmmMemLibInternalSmramRanges[Index].CpuStart >= Buffer)
    && (mSmmMemLibInternalSmramRanges[Index].CpuStart < Buffer + Length)))
    {
      return FALSE;
    }
  }
// [...]
}
```

- Loops through all entries in `mSmmMemLibInternalSmramRanges`
  - Quits if the buffer overlaps one region
- Continue with other tests if no match found

SmmIsBufferOutsideSmmValid

SMRAM Overlap Verification

- `mSmmMemLibInternalSmramRanges`
  - `EFI_SMRAM_DESCRIPTOR`: describing a SMRAM region and its accessibility attributes

```
typedef struct {
  EFI_PHYSICAL_ADDRESS     PhysicalStart;
  EFI_PHYSICAL_ADDRESS     CpuStart;
  UINT64                   PhysicalSize;
  UINT64                   RegionState;
} EFI_MMRAM_DESCRIPTOR;

// RegionState == accessibility attributes of the SMRAM

typedef EFI_MMRAM_DESCRIPTOR EFI_SMRAM_DESCRIPTOR;
```

- Table content
  - *(dumped when running OVMF)*

PhysicalStart	CpuStart	PhysicalSize	RegionState
0x7000000	0x7000000	0x001000	`EFI_ALLOCATED` `EFI_CACHEABLE`
0x7001000	0x7001000	0xFFF000	`EFI_CACHEABLE`

SmmIsBufferOutsideSmmValid

SMRAM Overlap Verification

- `mSmmMemLibInternalSmramRanges`
  - `EFI_SMRAM_DESCRIPTOR`: describing a SMRAM region and its accessibility attributes

```
typedef struct {
  EFI_PHYSICAL_ADDRESS     PhysicalStart;
  EFI_PHYSICAL_ADDRESS     CpuStart;
  UINT64                   PhysicalSize;
  UINT64                   RegionState;
} EFI_MMRAM_DESCRIPTOR;

// RegionState == accessibility attributes of the SMRAM

typedef EFI_MMRAM_DESCRIPTOR EFI_SMRAM_DESCRIPTOR;
```

- Table content
  - *(dumped when running OVMF)*

PhysicalStart	CpuStart	PhysicalSize	RegionState
0x7000000	0x7000000	0x001000	`EFI_ALLOCATED` `EFI_CACHEABLE`
0x7001000	0x7001000	0xFFF000	`EFI_CACHEABLE`

- Overwrite `mSmmMemLibInternalSmramCount` with `0x00000001` to dodge the check :D

SmmLockBox

4-byte Write Primitive to Arbitrary Read-Write Primitive

Problem

- ~~SmmLockBox API protected with `SmmIsBufferOutsideSmmValid`~~
- `SaveLockBox()`, `SmmLockBoxSetAttributes` & `UpdateLockBox()` locked

SmmLockBox

mLocked Variable

- Prevent data manipulation after on runtime
- Same notification event as `Tcg2Smm.efi`
  - Smm Ready To Lock event (`gEfiSmmReadyToLockProtocolGuid`)

```
EFI_STATUS
EFIAPI
SmmReadyToLockEventNotify (
  IN CONST EFI_GUID  *Protocol,
  IN VOID            *Interface,
  IN EFI_HANDLE      Handle
  )
{
  mLocked = TRUE;
  return EFI_SUCCESS;
}
```

```
VOID
SmmLockBoxSave ( /* [...] */  )
{
  // [...]

  if (mLocked) {
    DEBUG ((DEBUG_ERROR, "SmmLockBox Locked!\n"));
    LockBoxParameterSave->Header.ReturnStatus = \
                                          EFI_ACCESS_DENIED;
    return;
  }

  // [...]
}
```

SmmLockBox

mLocked Variable

No worries

- We can just overwrite it too and voilà!

SmmLockBox

mLocked Variable

No worries

- We can just overwrite it too and voilà!

Not quite...

```
0000070E0                dw 415Fh                ; Data3
0000070E0                db 87h, 46h, 8Fh, 0B5h, 0B8h, 9Ch, 0E4h, 0ACh; Data4
0000070F0 ; _LIST_ENTRY mLockBoxQueue
0000070F0 mLockBoxQueue  _LIST_ENTRY <offset mLockBoxQueue, offset mLockBoxQueue>
0000070F0                                        ; DATA XREF: SmmLockBoxMmConstructor+14F↑o
0000070F0                                        ; .data:mLockBoxQueue↓o
0000007100 ; unsigned __int8 mLocked
0000007100 mLocked       db 0                    ; DATA XREF: SmmLockBoxHandler:loc_1DBD↑r
0000007100                                       ; SmmLockBoxHandler:loc_1E34↑r ...
```

SmmLockBox

mLocked Variable

No worries

- We can just overwrite it too and voilà!

Not quite...

```
0000070E0                db 87h, 46h, 8Fh, 0B5h, 0B8h, 9Ch, 0E4h, 0ACh; Data4
0000070F0 ; _LIST_ENTRY mLockBoxQueue
0000070F0 mLockBoxQueue  _LIST_ENTRY <offset mLockBoxQueue, offset mLockBoxQueue>
0000070F0                                        ; DATA XREF: SmmLockBoxMmConstructor+14F↑o
0000070F0                                        ; .data:mLockBoxQueue↓o
0000007100 ; unsigned __int8 mLocked
0000007100 mLocked       db 0                    ; DATA XREF: SmmLockBoxHandler:loc_1DBD↑r
0000007100                                       ; SmmLockBoxHandler:loc_1E34↑r ...
```

- Should we just recompile it?
  - Nah, that's cheated...
  - Need to find something else

Interlude

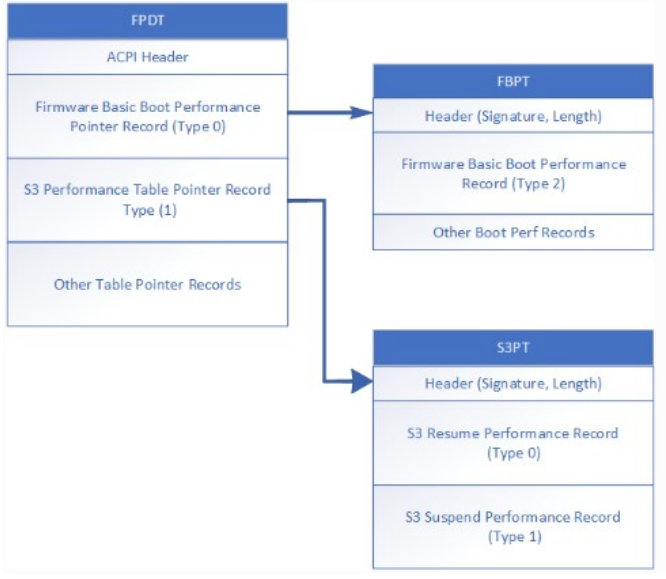
> **New goal:** Transform the "write 4-fixed-bytes anywhere" into "write **zero** anywhere"

Interlude

> **New goal:** Transform the "write 4-fixed-bytes anywhere" into "write **zero** anywhere"

ACPI Firmware Performance Data Table

- Provides information on platform initialization performance records during boot
- Used to track performance of each UEFI phase
- Also useful for tracking impacts from changes in hardware/software configuration



ACPI Firmware Performance Data Table

- Table in SMRAM
- Registers a SMI handler
  - `gEfiFirmwarePerformanceGuid`

```
EFI_STATUS
EFIAPI
FpdtSmiHandler (
  IN     EFI_HANDLE  DispatchHandle,
  IN     CONST VOID  *RegisterContext,
  IN OUT VOID        *CommBuffer,
  IN OUT UINTN       *CommBufferSize
  )
```

```
typedef struct {
  UINTN          Function;
  EFI_STATUS     ReturnStatus;
  UINTN          BootRecordSize;
  VOID           *BootRecordData;
  UINTN          BootRecordOffset;
} SMM_BOOT_RECORD_COMMUNICATE;
```

- Returns (depends on `Function` field)
  - FPDT size
  - Chunks of the table
    - By specifying the offset and size requested

ACPI Firmware Performance Data Table

```
FS0:\EFI\Tools\tcg2\> Python368.efi dump_fpdt.py
  ----- Get Performance Data Size -----
  FPDT Size: 0x4c0
  ----- Dump Performance Data -----
  00000000: 13 10 2A 01 03 00 00 00  00 00 BD B0 D0 ED 00 00  ..*.............
  00000010: 00 00 D8 E2 7E A4 0E F6  FD 42 8E 58 7B D6 5E E4  ....~....B.X{.^.
  00000020: C2 9B 01 00 00 00 00 00  00 00 13 10 2A 01 04 00  ............*...
  00000030: 00 00 00 00 47 49 CF F0  00 00 00 00 D8 E2 7E A4  ....GI........~.
  00000040: 0E F6 FD 42 8E 58 7B D6  5E E4 C2 9B 01 00 00 00  ...B.X{.^.......
  00000050: 00 00 00 00 10 10 22 01  01 00 00 00 00 00 81 92  ......".........
  00000060: 46 F2 00 00 00 00 D8 E2  7E A4 0E F6 FD 42 8E 58  F.......~....B.X
  ...
```

ACPI Firmware Performance Data Table

```
FS0:\EFI\Tools\tcg2\> Python368.efi dump_fpdt.py
  ----- Get Performance Data Size -----
  FPDT Size: 0x4c0
  ----- Dump Performance Data -----
  00000000: 13 10 2A 01 03 00 00 00  00 00 BD B0 D0 ED 00 00  ..*.............
  00000010: 00 00 D8 E2 7E A4 0E F6  FD 42 8E 58 7B D6 5E E4  ....~....B.X{.^.
  00000020: C2 9B 01 00 00 00 00 00  00 00 13 10 2A 01 04 00  ............*...
  00000030: 00 00 00 00 47 49 CF F0  00 00 00 00 D8 E2 7E A4  ....GI........~.
  00000040: 0E F6 FD 42 8E 58 7B D6  5E E4 C2 9B 01 00 00 00  ...B.X{.^.......
  00000050: 00 00 00 00 10 10 22 01  01 00 00 00 00 00 81 92  ......".........
  00000060: 46 F2 00 00 00 00 D8 E2  7E A4 0E F6 FD 42 8E 58  F.......~....B.X
  ...
```

- Plenty of `0x00` \o/
- Possibility to ask for 1 byte at any offset in the table

ACPI Firmware Performance Data Table

```
FS0:\EFI\Tools\tcg2\> Python368.efi dump_fpdt.py
  ----- Get Performance Data Size -----
  FPDT Size: 0x4c0
  ----- Dump Performance Data -----
  00000000: 13 10 2A 01 03 00 00 00  00 00 BD B0 D0 ED 00 00  ..*.............
  00000010: 00 00 D8 E2 7E A4 0E F6  FD 42 8E 58 7B D6 5E E4  ....~....B.X{.^.
  00000020: C2 9B 01 00 00 00 00 00  00 00 13 10 2A 01 04 00  ............*...
  00000030: 00 00 00 00 47 49 CF F0  00 00 00 00 D8 E2 7E A4  ....GI........~.
  00000040: 0E F6 FD 42 8E 58 7B D6  5E E4 C2 9B 01 00 00 00  ...B.X{.^.......
  00000050: 00 00 00 00 10 10 22 01  01 00 00 00 00 00 81 92  ......".........
  00000060: 46 F2 00 00 00 00 D8 E2  7E A4 0E F6 FD 42 8E 58  F.......~....B.X
  ...
```

- Plenty of `0x00` \o/
- Possibility to ask for 1 byte at any offset in the table

- Need to get rid of `SmmIsBufferOutsideSmmValid` again

Let's Rewind

4-fixed byte Write Primitive to (almost) Arbitrary Write Primitive

- Bypass of `SmmIsBufferOutsideSmmValid` in `PiSmmCore.efi`
  - Used by Firmware Performance Data Table SMI handler
    - `gEfiFirmwarePerformanceGuid`

> Overwrite of `mSmmMemLibInternalSmramCount` with Tcg2Smm bug

(almost) Arbitrary Write Primitive to Arbitrary R/W Primitive

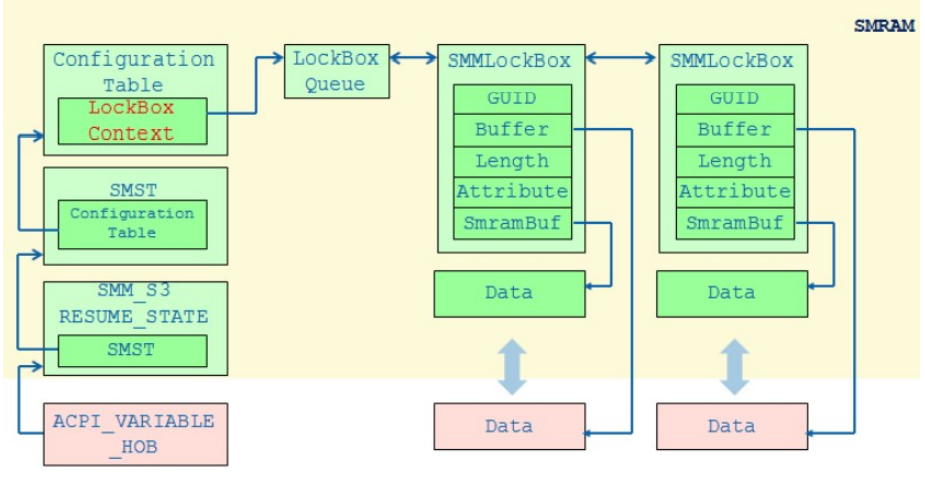
- Unlock SmmLockBox API
- Bypass of `SmmIsBufferOutsideSmmValid` in `SmmLockBox.efi`

> - Use `gEfiFirmwarePerformanceGuid` SMI handler to write `0x00` in `mLocked`
> - Overwrite of `mSmmMemLibInternalSmramCount` with either Tcg2Smm bug or `gEfiFirmwarePerformanceGuid` SMI handler

Arbitrary R/W to Code Execution

Shellcode Location

- SmmLockBox module reuse
  - Buffer allocated and copied in SMRAM
- Doubly-linked list of saved LockBox
  - stored in `mLockBoxQueue` globale variable



- Perfect way to store a shellcode :D
  - Get `mLockBoxQueue`
  - Retrieve the last inserted LockBox data buffer
  - Execute & hourray

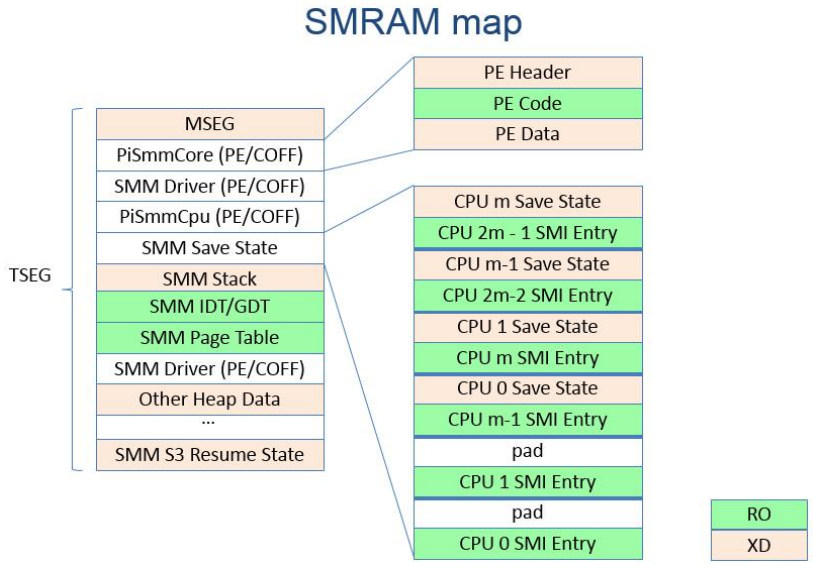
Arbitrary R/W to Code Execution

Shellcode Location

Small issue

- LockBox buffer not executable :/

- Memory access protection
  - Depending on the page usage
- Types allowed for allocation in SMM
  - `EfiRuntimeServicesData` - access: `RW-`
  - `EfiRuntimeServicesCode` - access: `R-X`
- Implemented at the page table entry level
- Activated if the SMM image is page aligned



Arbitrary R/W to Code Execution

Shellcode Location

Small issue

- LockBox buffer not executable :/

- Memory access protection
  - Depending on the page usage
- Types allowed for allocation in SMM
  - `EfiRuntimeServicesData` - access: `RW-`
  - `EfiRuntimeServicesCode` - access: `R-X`
- Implemented at the page table entry level
- Activated if the SMM image is page aligned

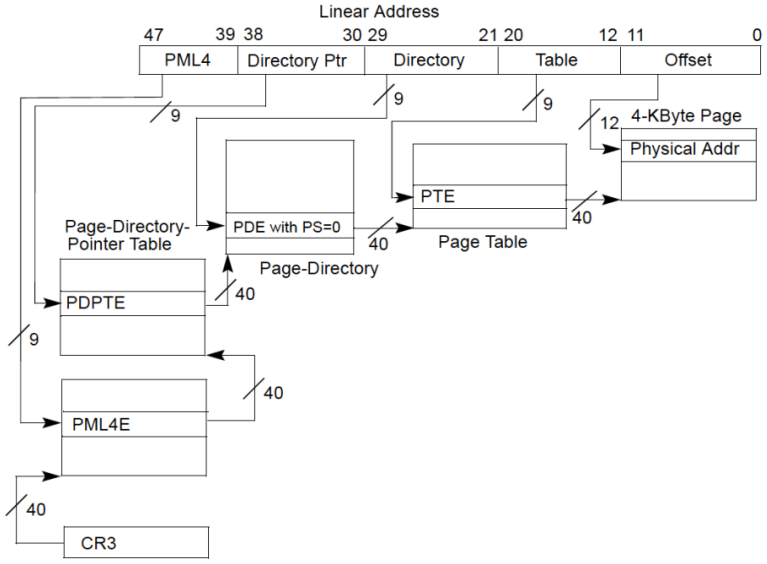
Fun Fact

- Not the case on OVMF compiled with MSFT toolchain ;)
  - Missing `/ALIGN:4096` build option

Fix The Access Right

Find the page table entry

- CR3 value stored in `mSmmProfileCr3`
  - Located in `PiSmmCpuDxeSMM.efi`
- # of level depending on the page size
  - 4 levels for regular pages
- Entries can be protected with a mask
  - AMD Secure Encrypted Virtualization
  - may also be found in `mAddressEncMask`




```
AddressEncMask = PcdGet64 (PcdPteMemoryEncryptionAddressOrMask) & PAGING_1G_ADDRESS_MASK_64;
// ...
PageTable = Entry & ~AddressEncMask & PAGING_4K_ADDRESS_MASK_64;
```

Fix The Access Right

Protection Removal

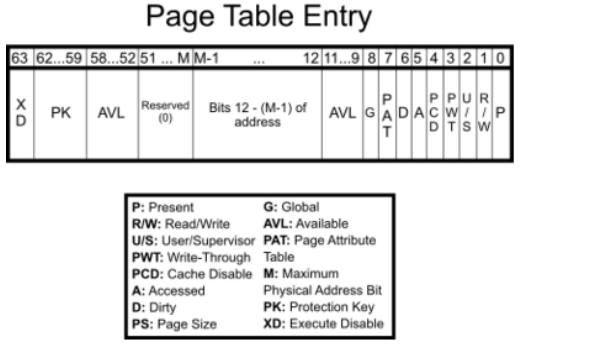
Write Protect

- Page table entries in read only
- Bit 16 (WP) in CR0
  - Can use `AsmWriteCr0` function to fix it

```
UINTN EFIAPI AsmWriteCr0( UINTN Cr0 )
{
  __asm__ __volatile__ (
    "movl %0, %%cr0"
    :
    : "r" (Cr0)
  );
  return Cr0;
}
```

No Execute

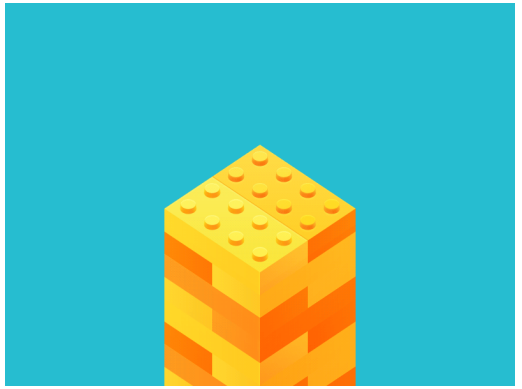
- Bit 63 (NX) of the page entry value
- Need to set it to `0`
  - No shiny way beside doing it by hand :|



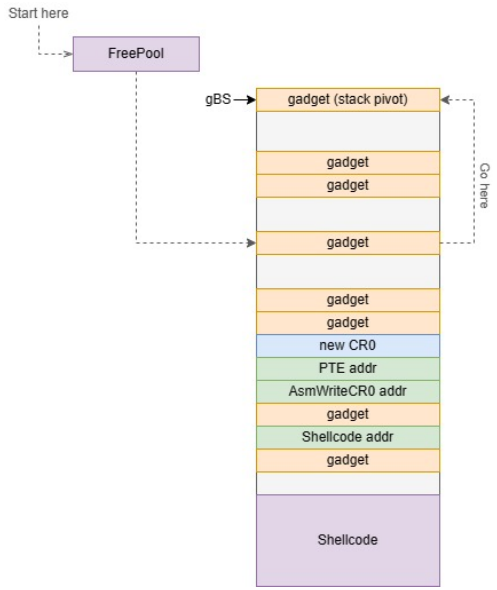
Fix The Access Right

Protection Removal

> Wibbly-wobbly-grubby-magicky part of the exploit



- `ROPGadget`[1] on SMM modules
  - Only focused on `PiSmmCpuDxeSMM.efi` actually
- ROPchain crafting
  - 8 gadgets
  - 2 function calls
  - 1 globale variable corruption
- Et voila!

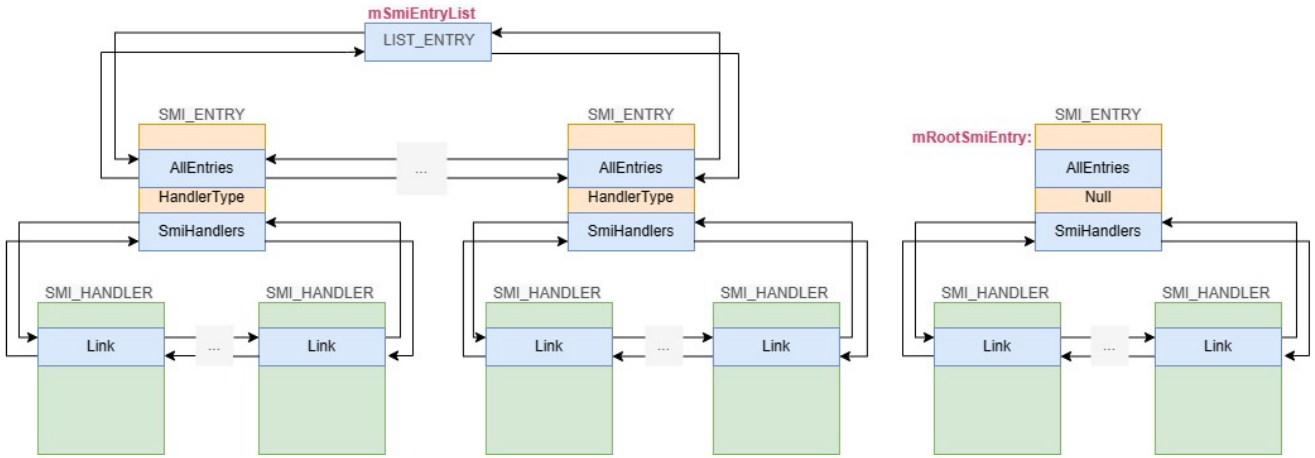


[1]: https://github.com/JonathanSalwan/ROPgadget

Execution

SMI Handler Registration

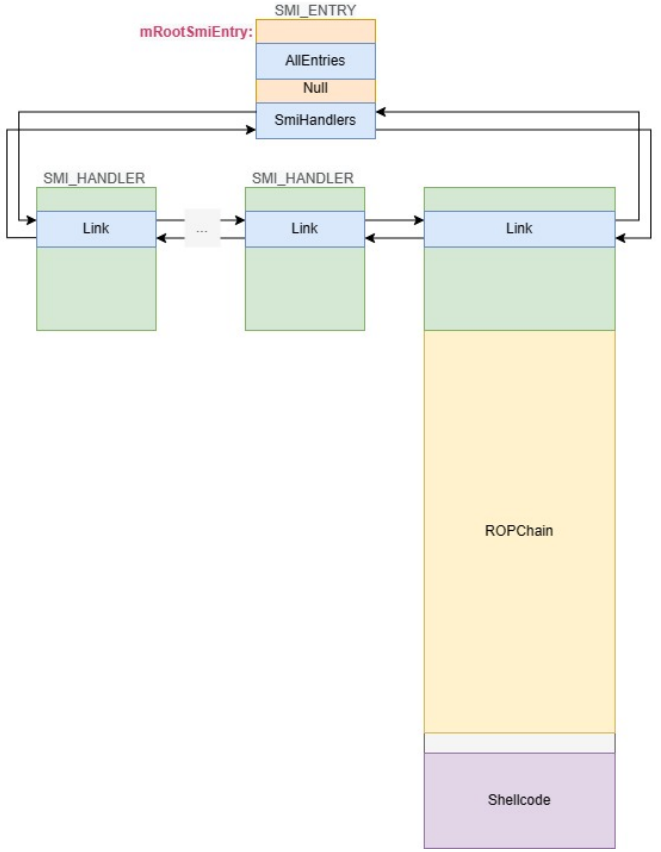
- SMI handlers registered with `SmmiHandlerRegister` (provided by the SMM System Table)
- Create a `SMI_HANDLER` object
- Add it to the double-linked list corresponding to its type
  - defined by a `SMI_ENTRY` object in `PiSmmCore.efi`

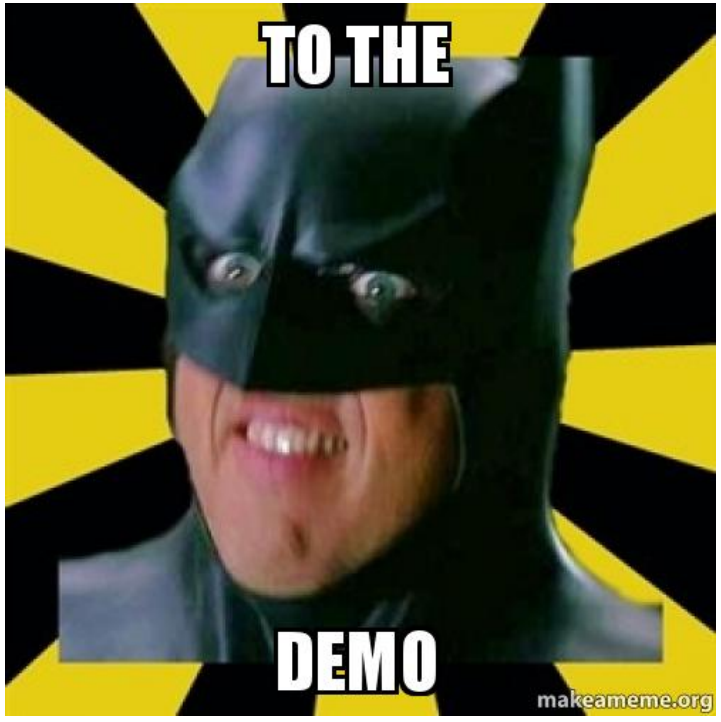


Execution

Fake SMI handler

- Simply add a fake object in the `SMI_ENTRY` list
- Wait for a couple of (milli) seconds
  - If in the root list
  - Otherwise, need to call it
- Clean every thing
- Profit \o/




TO THE
DEMO
makeameme.org

Conclusion

Conclusion

- Meh bug...

Conclusion

- Meh bug...
- Exploitation part really fun

Conclusion

- Meh bug...
- Exploitation part really fun

> - Even better exploit presented at BlueHatIL by Benny Zeltser & Jonathan Lusky
>   - *"RingHopper – Hopping from User-space to God Mode"*

Conclusion

- Meh bug...
- Exploitation part really fun

> - Even better exploit presented at BlueHatIL by Benny Zeltser & Jonathan Lusky
>   - *"RingHopper – Hopping from User-space to God Mode"*

- Thanks for listening anyway :)

Quarkslab

Questions?

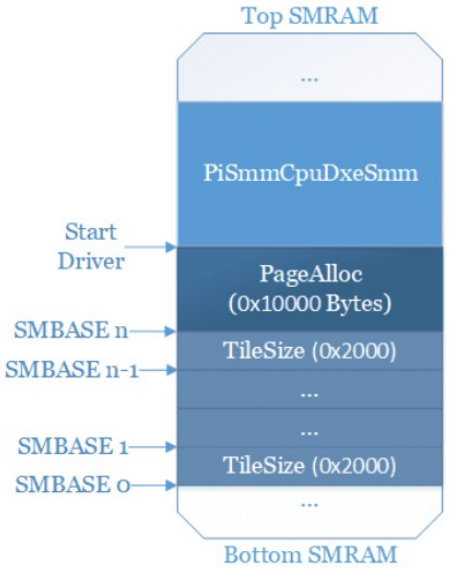
SmmLockBox

Lockation (pun intended)

Reuse of a wonderful technique[1] from Bruno Pujos (@BrunoPujos) to find the `SMBASE`

- Initialized in the `PiSmmCpuDxeSMM` module
  - Calculates the size necessary to reserve
    - `0x10000 + TileSize * (number_of_cpu - 1)`
  - Allocates the `SMBASE` just after the module
    - Use of `SmmAllocatePages`
    - Takes the highest available page of memory
      - Because nothing in the free list for now
- Get `PiSmmCpuDxeSMM` base address
  - Through its protocol registration

    `gSmmCpuPrivate->SmmConfiguration`

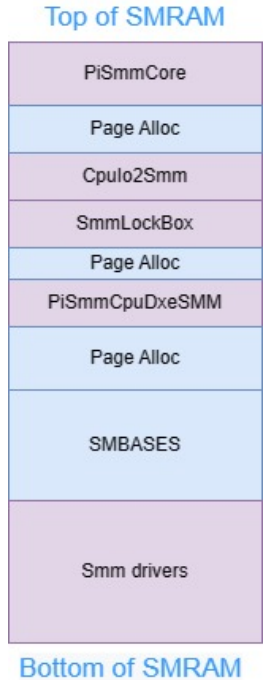


[1]: *https://www.synacktiv.com/en/publications/through-the-smm-class-and-a-vulnerability-found-there.html*

SmmLockBox

Lockation (pun intended)

- Actually we don't care about the SMBASE...
- But we do care about `PiSmmCpuDxeSMM`!
  - One of the first SMM modules to be loaded at boot time
  - `SmmLockBox.efi` loaded just before
- `SmmLockBox.efi` base address can be calculated
  - `delta = Pe.SizeofImage + Pe.fileAlignment + [Lockbox allocated data]`



ACPI Firmware Performance Data Table

Location (no pun this time)

- SMI published by SMM foundation
  - Part of `PiSmmCore.efi`
- Location calculated the same way as for SmmLockBox
  - Just need to take into account CpuIo2Smm