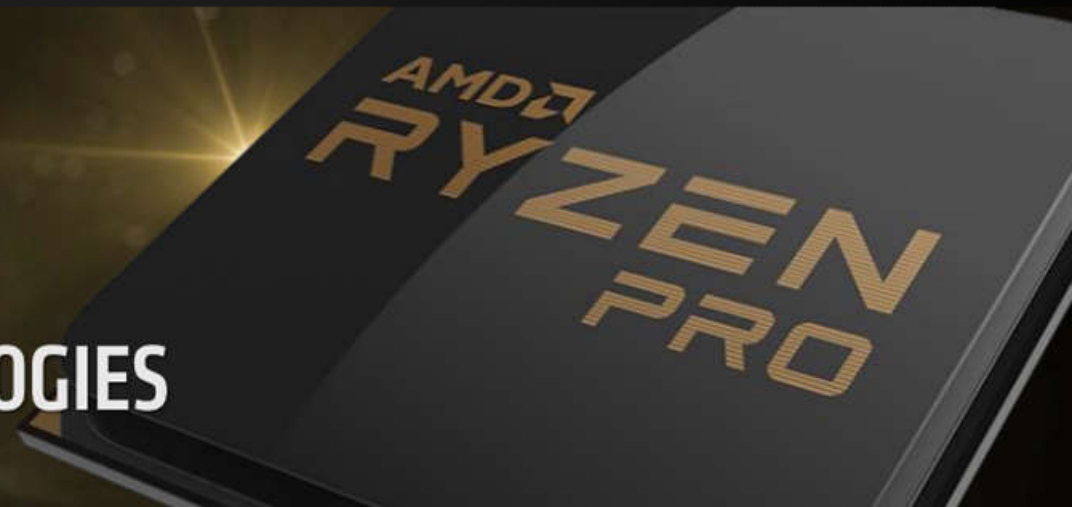




AMDFlaws

A Technical Deep Dive

INSECURITY AT THE
SILICON LEVEL
WITH AMD SECURE TECHNOLOGIES



Bio

- ◆ Ido Li On

 - ◆ CEO @ CTS Labs (2017-2018)

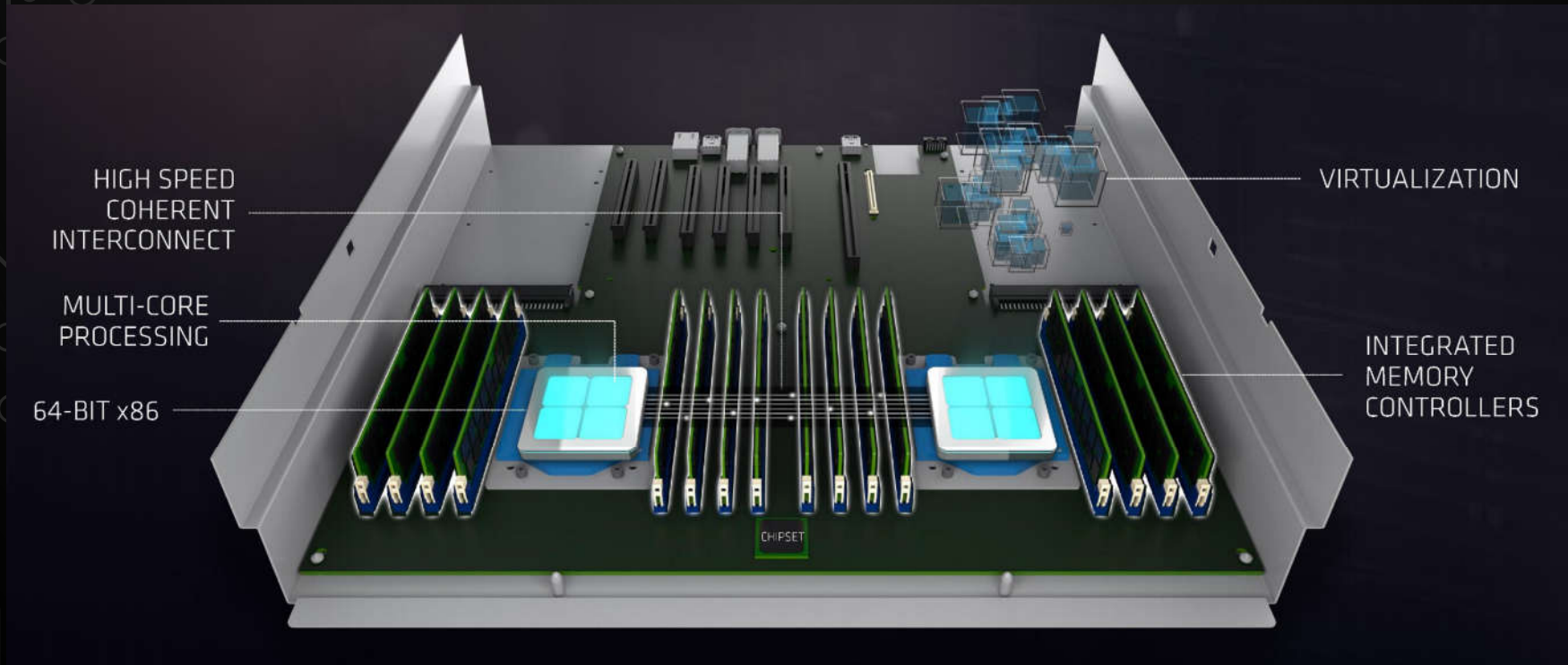
- ◆ Uri Farkas

 - ◆ VP R&D @ CTS Labs (2017-2018)





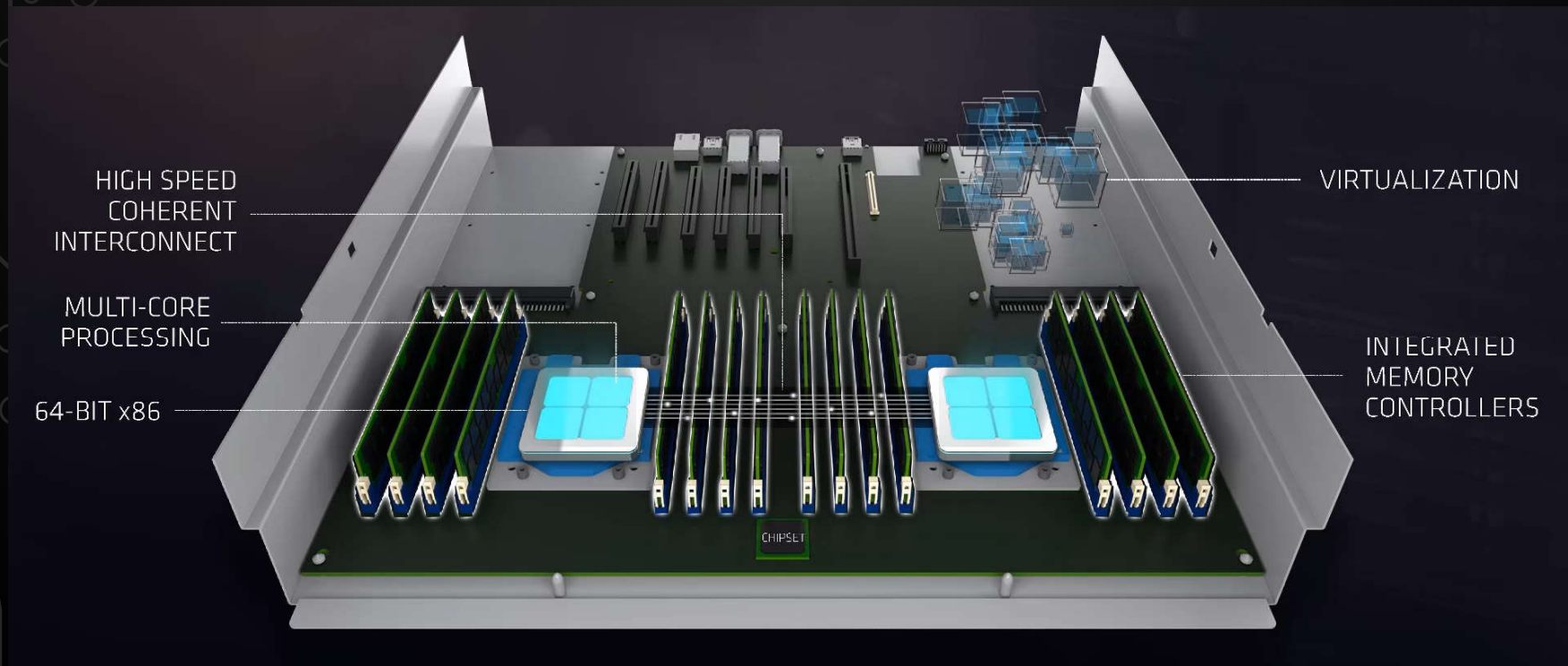
Research Overview: 13 Vulnerabilities



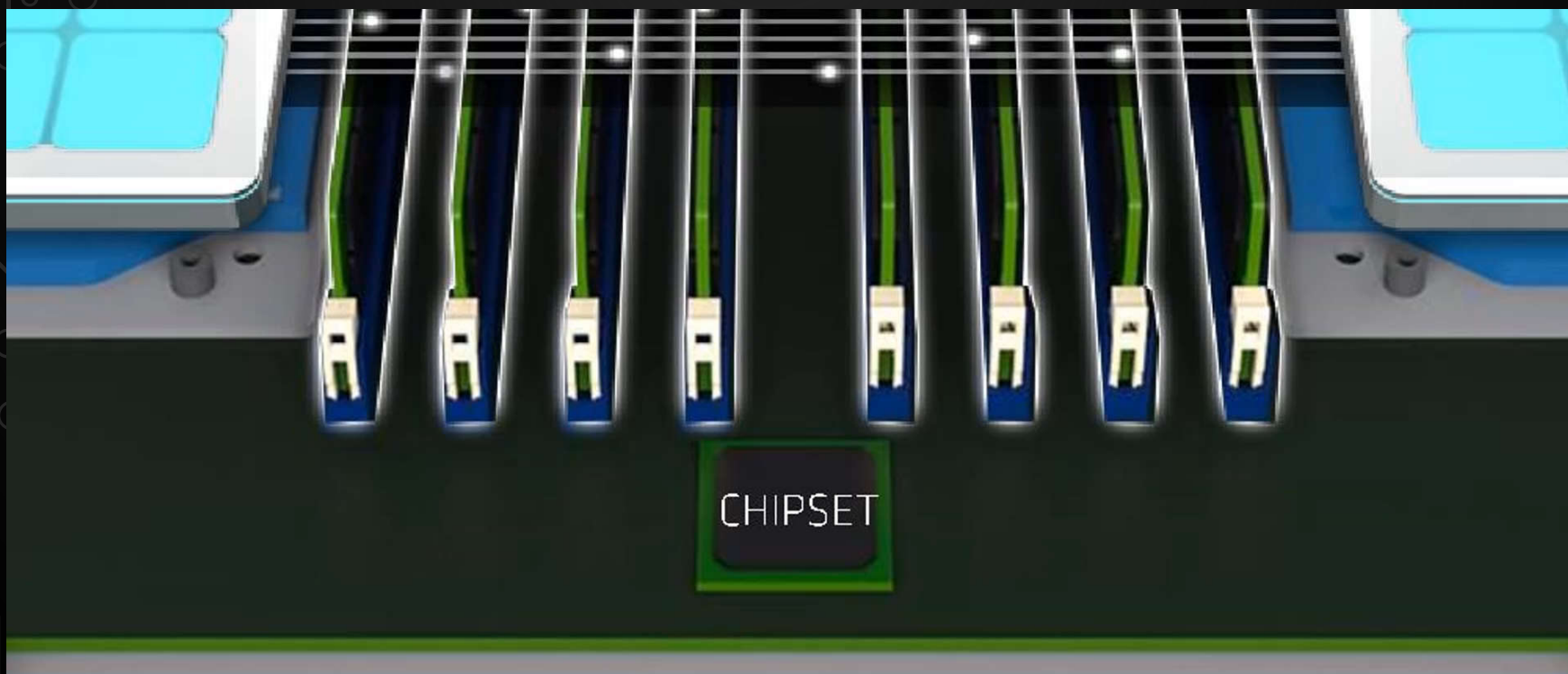


Chipset Vulnerabilities

The chipset contains USB, SATA, and PCIe bridge controllers



Platform Security Processor Vulnerabilities





Demo: Ryzen Desktop machine

Ryzen 5

Biostar B350GT3



192.168.0.140 - Remote Desktop Connection

Recycle Bin
IDA Pro (64-bit)

Firefox

RW-Everyth...

Visual Studio Code

Windows Firewall...

cmd.exe

IDA Pro (32-bit)



11:27 AM
2/6/2019





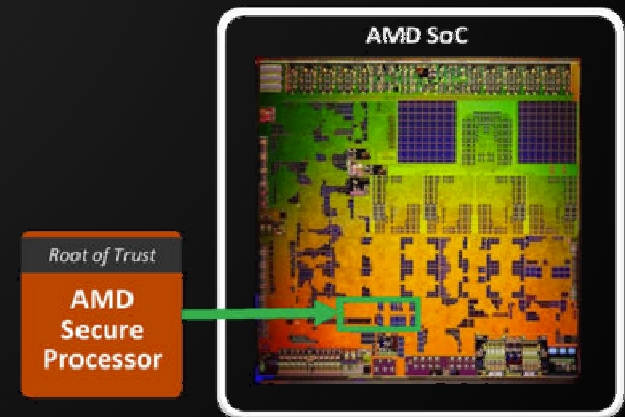
What we're here to talk about

- ◆ The Platform Security Processor (PSP) and why care about it
- ◆ How we researched the PSP
- ◆ Three of the most interesting vulnerabilities we found

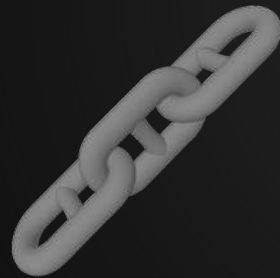


The Platform Security Processor (PSP)

- ◇ Security Subsystem similar to Intel ME / Apple Secure Enclave
- ◇ First version introduced by AMD in 2013
- ◇ Massively revised in the Zen architecture (2017)



The PSP has powerful capabilities



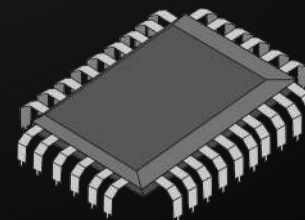
Hardware Root of Trust



Direct Memory Access



TPM



Completely Independent



The PSP has powerful capabilities



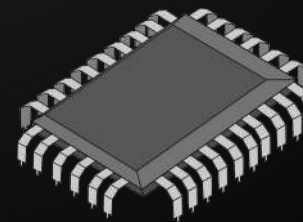
Hardware Root of Trust



Direct Memory Access



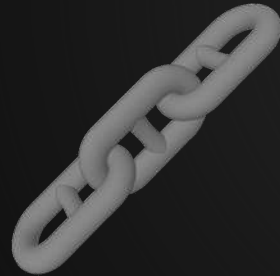
TPM



Completely Independent



The PSP has powerful capabilities



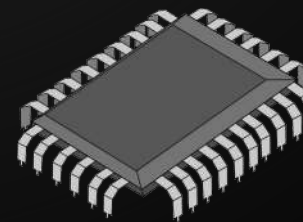
Hardware Root of Trust



Direct Memory Access



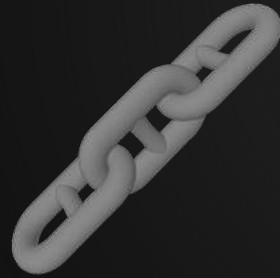
TPM



Completely Independent



The PSP has powerful capabilities



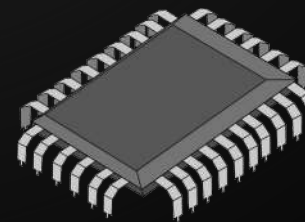
Hardware Root of Trust



Direct Memory Access



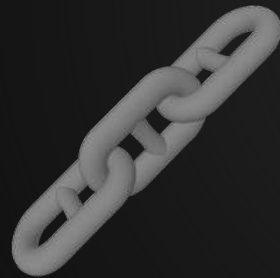
TPM



Completely Independent



The PSP has powerful capabilities



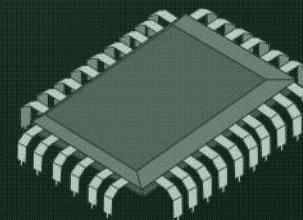
Hardware Root of Trust



Direct Memory Access



TPM



Completely Independent

The PSP is controversial



change.org Start a petition Browse Membership

Log In

Release the source code for the secure processor (PSP)

Manuel Adolfo Lobeiras Iglesias started this petition to CEO Advanced Micro Devices - AMD

Please, AMD, open the source code for the PSP.

Reasons are posted everywhere in regarding topics, I will summarize some of them and I hope we can generate an interest for the company to release it:

- Eliminate security through obscurity. A secure system must be secure even if every detail but the key is known by untrusted individuals or organizations.
- Give users control over their own systems. It generates confidence in AMD.
- Give FSF and other similar organizations a great reason to recomend AMD for purchases of supporters.
- Increase presence in key security systems on companies and governments.

Petition Closed
This petition had 3,136 supporters

Advanced Micro Devices - AMD:
Release the source code for the...

Share on Facebook

- Send a Facebook message
- Send an email to friends
- Tweet to your followers
- Copy link

The PSP is everywhere



Ryzen PCs



Ryzen Mobile Laptops



EPYC Servers



Vega GPUs



Pwning the PSP

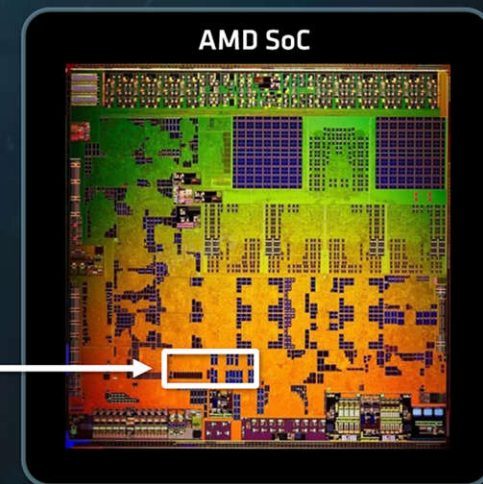
The PSP Firmware resides in SPI ROM (BIOS Image)



AMD SECURE PROCESSOR

A Dedicated Security Subsystem

- AMD Secure Processor integrated within SoC
 - 32-bit microcontroller (ARM Cortex-A5)
- Runs a secure OS/kernel
- Secure off-chip NV storage for firmware and data (i.e. SPI ROM)
- Provides cryptographic functionality for secure key generation and key management
- Enables hardware validated boot



Hardware Root of Trust Provides Foundation for Platform Security

UEFITool lets us examine BIOS images



UEFITool NE alpha 44 - 8026V101.B20

File Action View Help

Structure

Name	Action	Type	Subtype	Text
UEFI image		Image	UEFI	
Padding		Padding	Non-empty	
> FA4974FC-AF1D-4E5D-BDC5...		Volume	FFSv2	
Padding		Padding	Non-empty	
> 4F1C52D3-D824-4D2A-A2F0...		Volume	FFSv2	
> F649FC2D-C0E6-4262-AD51...		Volume	FFSv2	
> 61C0F511-A691-4F54-974F...		Volume	FFSv2	

Information

Offset: 0h
Full size: 1000000h (16777216)
Memory address: FF000000h
Compressed: No
Fixed: Yes

Parser FIT BootGuard Search Builder

```
parsePadFileBody: non-UEFI data found in pad-file
```



The PSP firmware resides in a "padding" area

UEFITool NE alpha 44 - 8026V101.B20

File Action View Help

Structure

Name	Action	Type	Subtype	Text
UEFI image		Image	UEFI	
Padding				
FA4974FC-AF1D-4E5D-BDC5...				
Padding				
4F1C52D3-D824-4D2A-A2F0...				
F649FC2D-C0E6-4262-AD51...				
61C0F511-A691-4F54-974F...				

Information

Offset: 57000h
Full size: 24D000h (2412544)
Memory address: FF057000h
Compressed: No
Signed: Yes

Hex view: Padding

Address	Hex	ASCII
01FFC0	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	yyyyyyyyyyyyyyyy
01FFD0	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	yyyyyyyyyyyyyyyy
01FFE0	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	yyyyyyyyyyyyyyyy
01FFF0	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	yyyyyyyyyyyyyyyy
020000	24 50 53 50 C9 B6 C6 8F 10 00 00 00 A0 04 00 00	\$PSP\$M& r... ..
020010	00 00 00 00 40 02 00 00 00 74 07 FF 00 00 00 00@....t.y....
020020	01 00 00 00 00 00 01 00 00 94 14 FF 00 00 00 00y....
020030	03 00 00 00 40 C8 00 00 00 77 07 FF 00 00 00 00@È...w.y....
020040	08 00 00 00 80 E4 01 00 00 40 08 FF 00 00 00 00'a...@.y....
020050	0A 00 00 00 40 03 00 00 00 25 0A FF 00 00 00 00@....%.y....
020060	12 00 00 00 40 3E 00 00 00 29 0A FF 00 00 00 00@>...).y....
020070	21 00 00 00 10 00 00 00 00 68 0A FF 00 00 00 00	!.....h.y....
020080	24 00 00 00 C0 0C 00 00 00 69 0A FF 00 00 00 00	\$...À....i.y....
020090	30 00 00 00 40 0C 00 00 00 76 0A FF 00 00 00 00	0...@....v.y....
0200A0	31 00 00 00 E0 BB 00 00 00 83 0A FF 00 00 00 00	1...à»...r.y....
0200B0	32 00 00 00 70 B3 00 00 00 3F 0B FF 00 00 00 00	2...p³...?.y....
0200C0	33 00 00 00 D0 DA 00 00 00 F3 0B FF 00 00 00 00	3...ÐÚ...ó.y....
0200D0	34 00 00 00 10 EC 00 00 00 CE 0C FF 00 00 00 00	4...i...Î.y....
0200E0	35 00 00 00 50 F0 00 00 00 BB 0D FF 00 00 00 00	5...Pð...»y....
0200F0	36 00 00 00 20 BD 00 00 00 AC 0E FF 00 00 00 00	6...%...r.y....
020100	40 00 00 00 00 04 00 00 00 90 14 FF 00 00 00 00	@.....h.y....
020110	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	yyyyyyyyyyyyyyyy
020120	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	yyyyyyyyyyyyyyyy
020130	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	yyyyyyyyyyyyyyyy
020140	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	yyyyyyyyyyyyyyyy



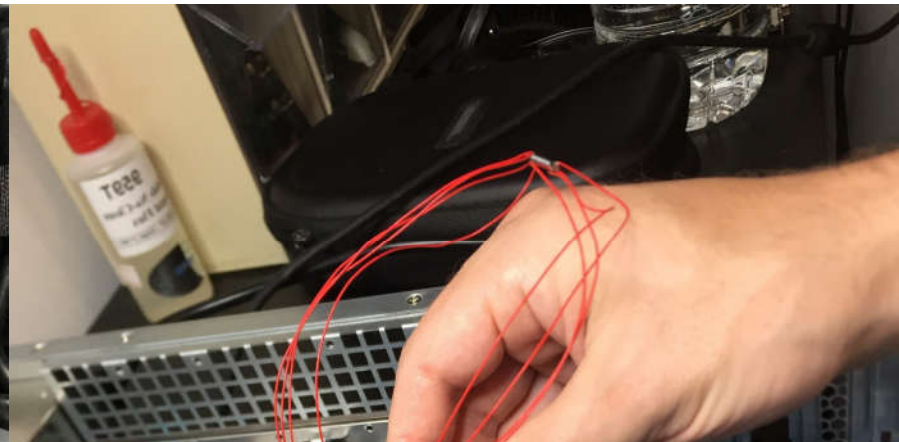
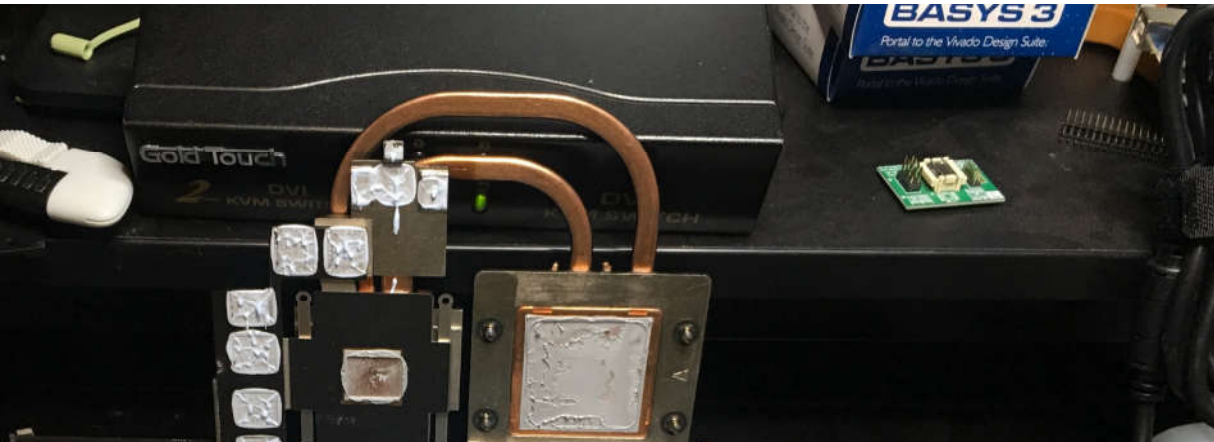
ARM code inside

```
uri@Uri:~$ binwalk -A padding.bin
```

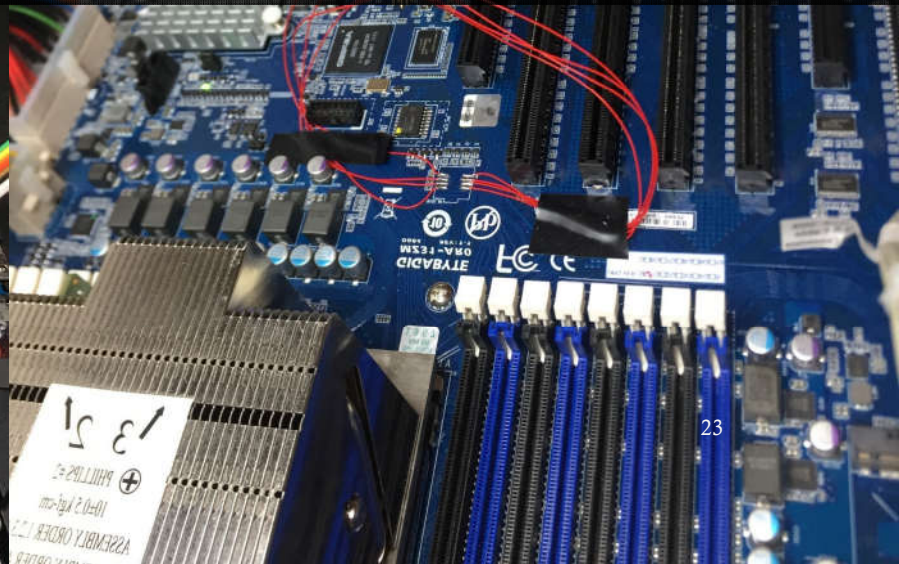
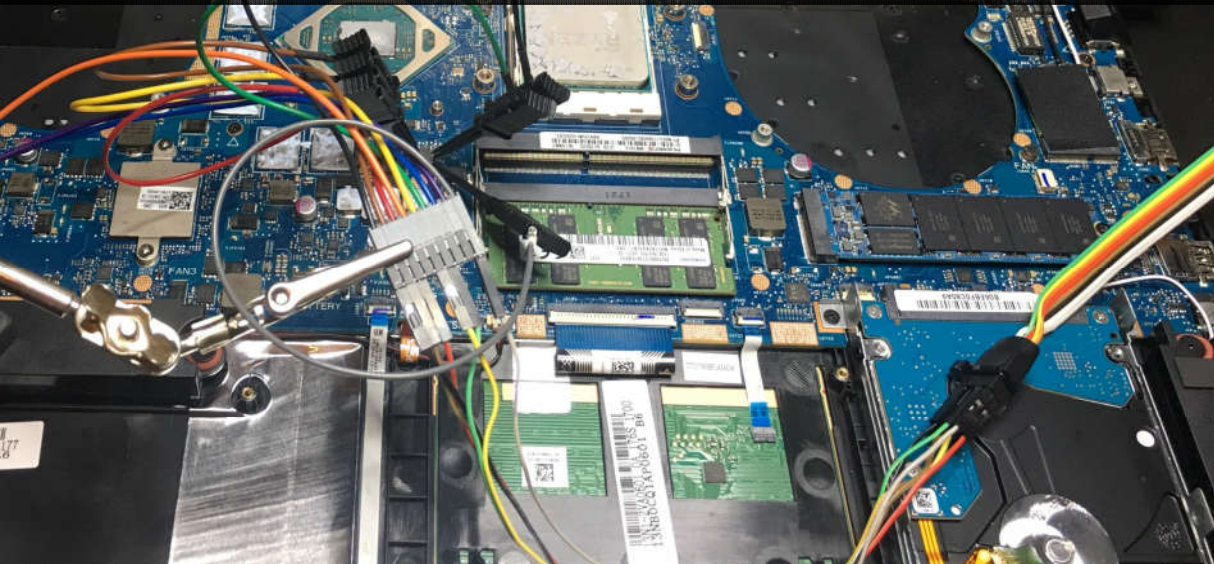
DECIMAL	HEXADECIMAL	DESCRIPTION
206712	0x32778	ARM instructions, function prologue
268580	0x41924	ARM instructions, function prologue
268652	0x4196C	ARM instructions, function prologue
268668	0x4197C	ARM instructions, function prologue
268684	0x4198C	ARM instructions, function prologue
268780	0x419EC	ARM instructions, function prologue
268792	0x419F8	ARM instructions, function prologue
270372	0x42024	ARM instructions, function prologue
520402	0x7F0D2	ARMEB instructions, function prologue
989088	0xF17A0	ARM instructions, function prologue
1906212	0x1D1624	ARM instructions, function prologue
...		
...		



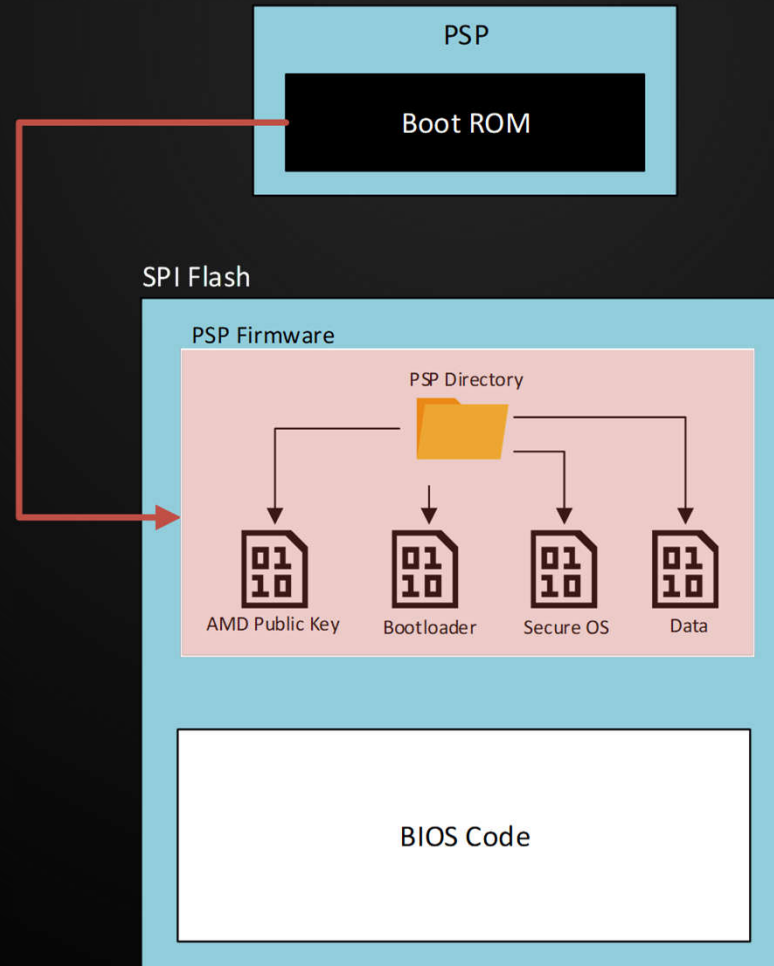
```
VOLATILE:00000FA0 aPspfwBootloade DCB "PSPFW Bootloader Version",0
VOLATILE:00000FA0 ; DATA XREF: f_bootloader_routine+24A7a
VOLATILE:00000FB9 DCB 0, 0, 0
VOLATILE:00000FBC off_FBC DCB dword_A2C4 ; DATA XREF: f_bootloader_routine:loc_E247r
VOLATILE:00000FC0 dword_FC0 DCB 0x5A870 ; DATA XREF: f_bootloader_routine+2C87r
VOLATILE:00000FC4 dword_FC4 DCB 0x18002FD0 ; DATA XREF: f_bootloader_routine:loc_E607r
VOLATILE:00000FC8 ; ----- S U B R O U T I N E -----
VOLATILE:00000FC8
VOLATILE:00000FC8 sub_FC8 ; CODE XREF: sub_2004+607p
VOLATILE:00000FC8 ; sub_333C+247p ...
VOLATILE:00000FC8 PUSH {R4-R6,LR}
VOLATILE:00000FCA LDRD.W R4, R6, [SP,#0x10]
VOLATILE:00000FCE LSLS R5, R4, #3
VOLATILE:00000FD0 ORR.W R1, R5, R1,LSL#12
VOLATILE:00000FD4 MOVS R4, #0x18
VOLATILE:00000FD6 LSLS R1, R1, #5
VOLATILE:00000FD8 LSLS R5, R6, #0x1F
VOLATILE:00000FDA BEQ loc_FDE
VOLATILE:00000FDC MOVS R4, #0x1A
VOLATILE:00000FDE loc_FDE ; CODE XREF: sub_FC8+127j
VOLATILE:00000FDE ORR.W R1, R1, #0x70000
VOLATILE:00000FE2 ORRS R1, R4
VOLATILE:00000FE4 STR R1, [R0]
VOLATILE:00000FE6 MOVS R1, #0x240
VOLATILE:00000FEA STRD.W R1, R3, [R0,#4]
VOLATILE:00000FEE MOV.W R1, #0x20000
VOLATILE:00000FF2 STRD.W R1, R2, [R0,#0xC]
```

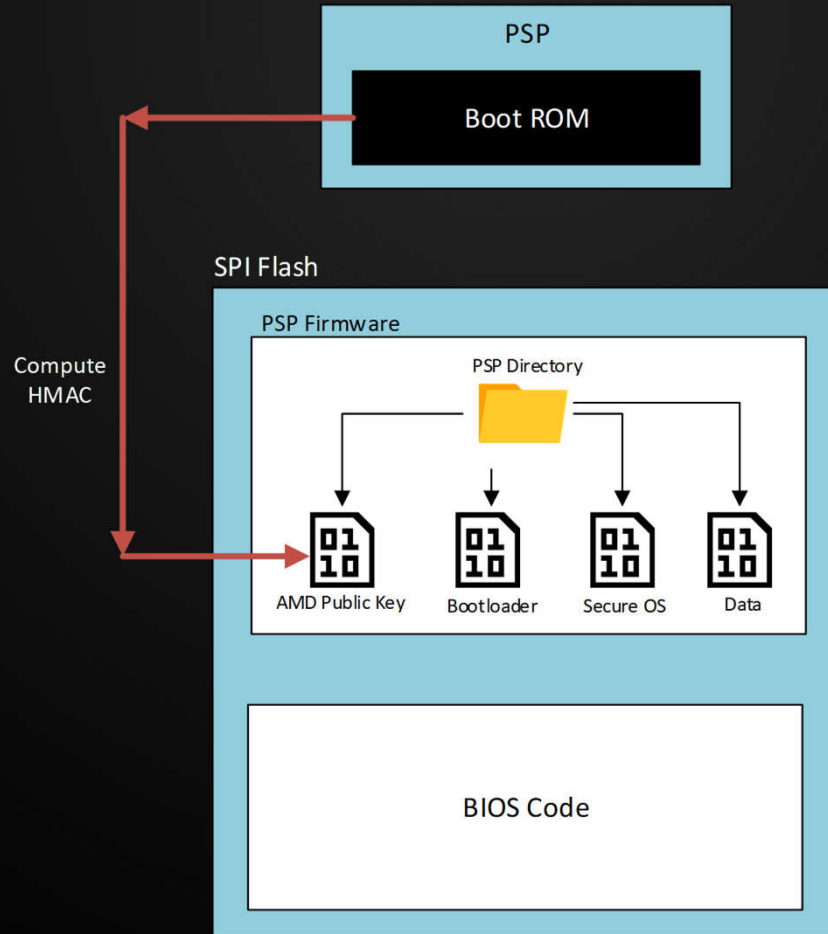
Our goal: To "Jailbreak" the PSP



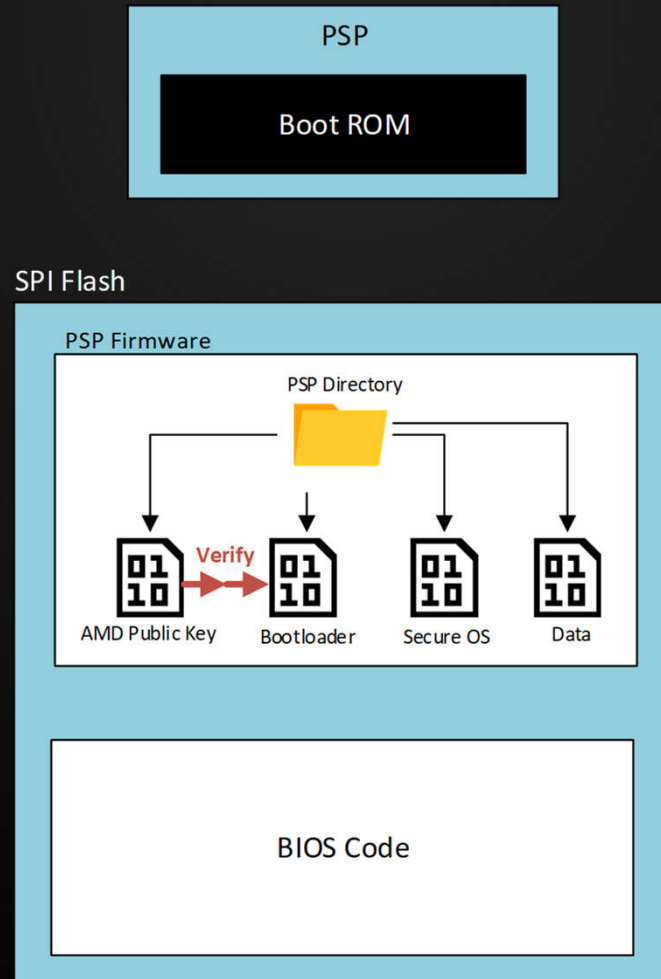
The Boot Process is a chain of verification



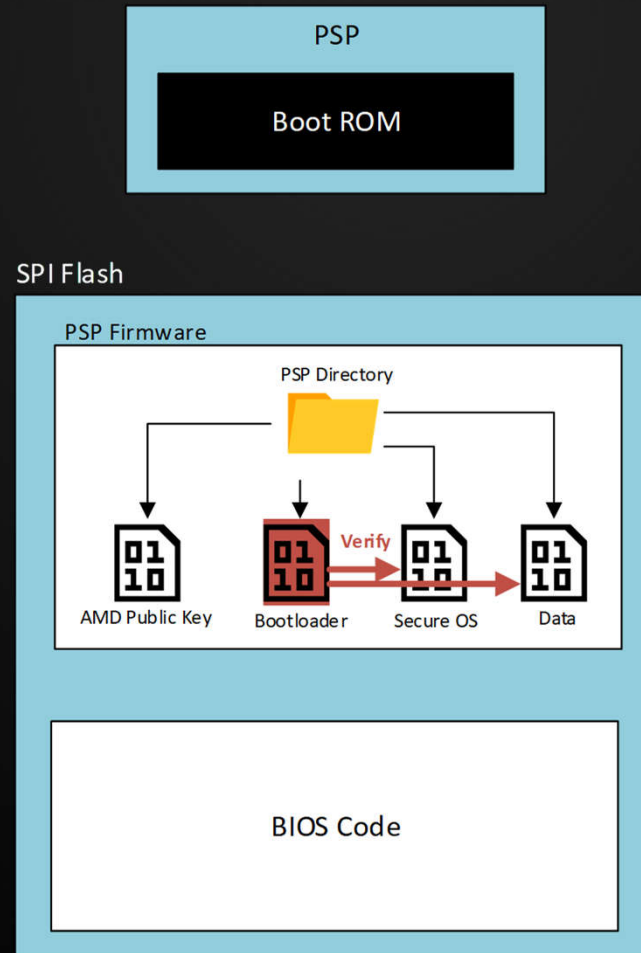
The Boot Process is a chain of verification



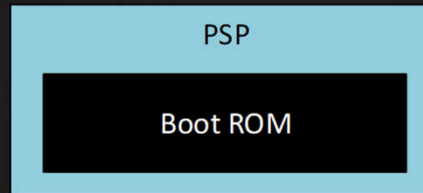
The Boot Process is a chain of verification



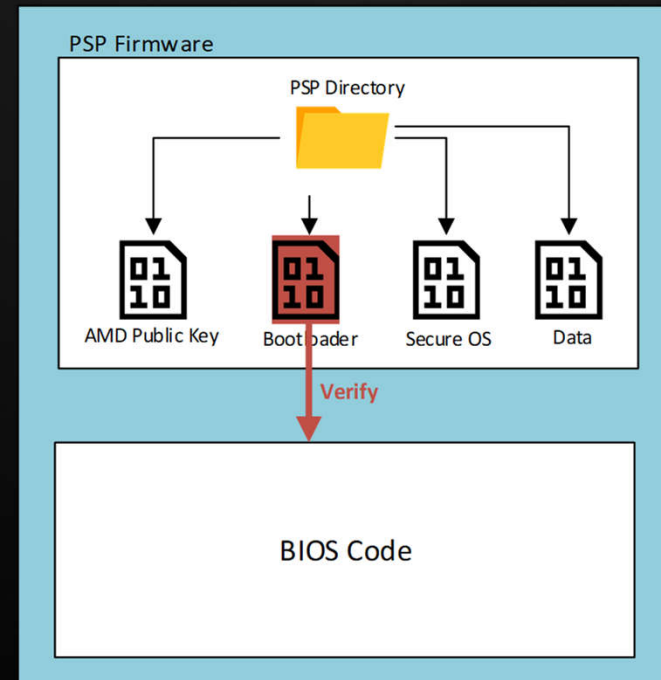
The Boot Process is a chain of verification



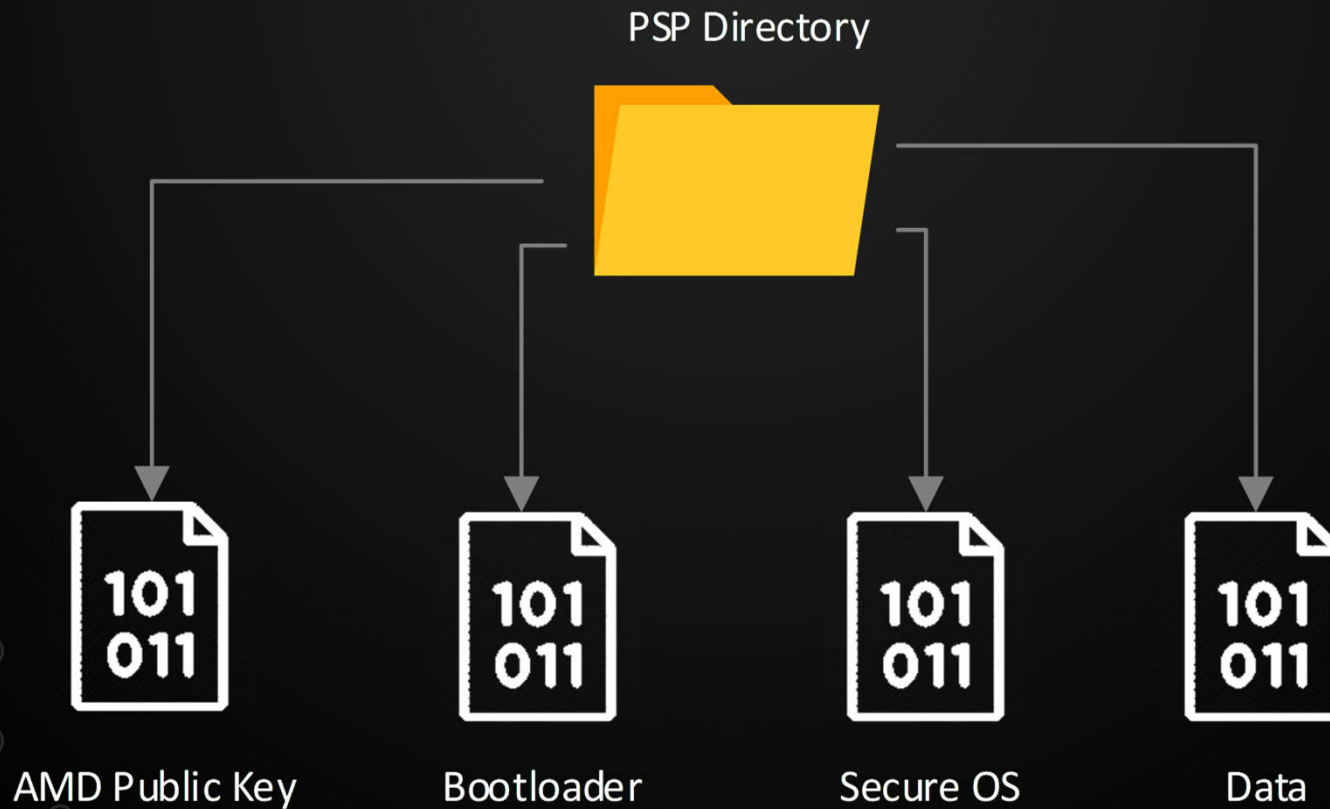
The Boot Process is a chain of verification



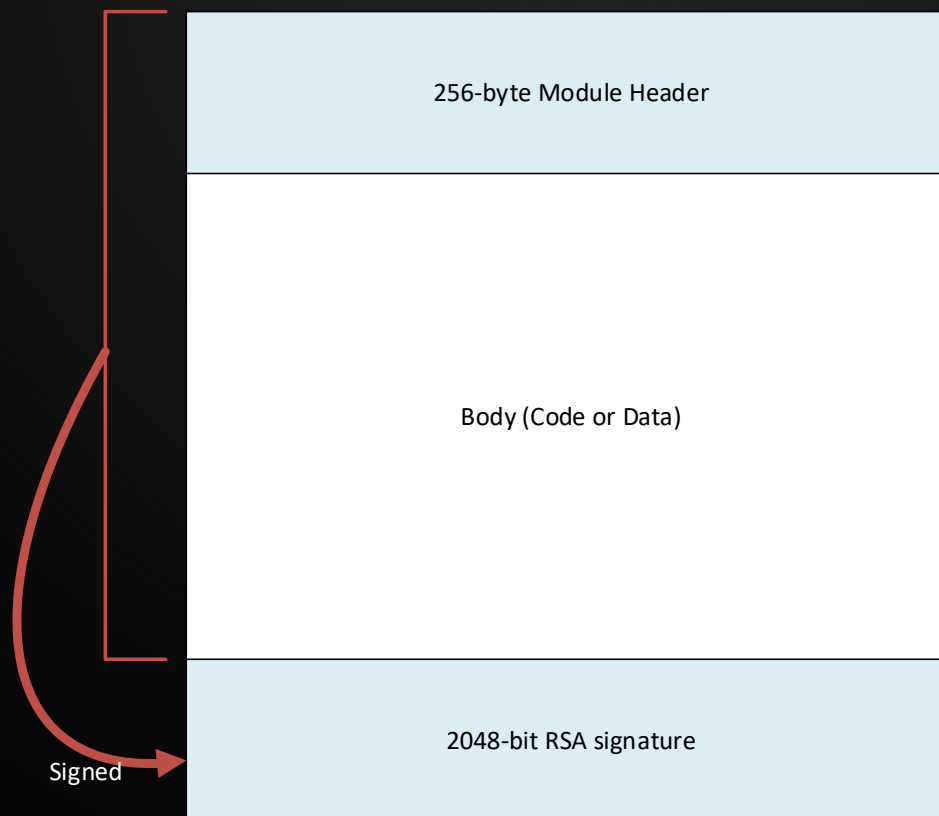
SPI Flash



PSP firmware is comprised of individually signed modules



Every PSP Module has a header and body, which are signed by the AMD Public Key





A field in the module's header determines what is signed

signed_part_size total_size

```
Hex View
FF951400: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
FF951410: 24 50 53 31 40 B3 00 00 00 00 00 00 00 00 00 00 $PS1@.....
FF951420: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
FF951430: 01 00 00 00 00 00 00 00 1B B9 87 C3 59 49 46 06 .....YIF.
FF951440: B1 74 94 56 01 C9 EA 58 00 00 00 00 00 00 00 00 .t.V...[.....
FF951450: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
FF951460: 28 04 09 00 FF FF 01 17 00 01 00 00 40 B5 00 00 (...@...
FF951470: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
FF951480: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
FF951490: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
FF9514A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
FF9514B0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
FF9514C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
FF9514D0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
FF9514E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
FF9514F0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
FF951500: 18 F0 9F E5 18 F0 9F E5 18 F0 9F E5 18 F0 9F E5 .....
FF951510: 18 F0 9F E5 00 F0 20 E3 14 F0 9F E5 14 F0 9F E5 .....
FF951520: 3C 01 00 00 E8 02 00 00 8C 02 00 00 EC 02 00 00 <.....
FF951530: F8 02 00 00 04 03 00 00 1C 03 00 00 10 1F 11 EE .....
FF951540: 02 1A C1 E3 10 1F 01 EE AC 03 9F E5 10 0F 0C EE .....
```



Some modules have signed_part_size = 0

```
uri@Uri:~$ python find_issues.py
```

```
Checking FF149400.bin.....OK
Checking FF159400.bin.....OK
Checking FF178100.bin.....OK
Checking FF17C000.bin.....OK
Checking FF17F000.bin.....OK
Checking FF262100.bin.....ERROR! signed_part_size=0x0 but signature available
Exiting..
```

```
uri@Uri:~$
```



An example PSP data module

signed_part_size total_size

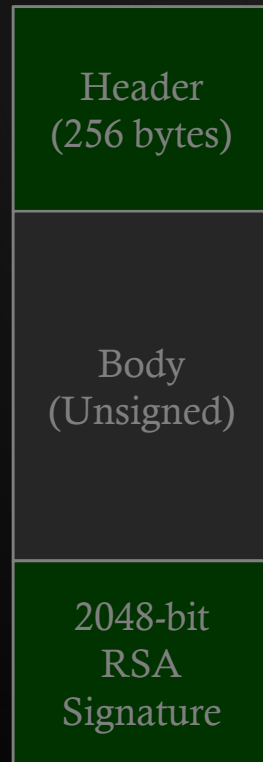
Hex View

FFA98800:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
FFA98810:	05 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
FFA98820:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
FFA98830:	01 00 00 00 00 00 00 00 1B B9 87 C3 59 49 46 06YIF.
FFA98840:	B1 74 94 56 01 C9 EA 5B 01 00 00 00 00 00 00 00	.t.V...[.....
FFA98850:	1A 60 00 00 22 44 00 00 00 00 00 00 00 00 00 00	..`..D.....
FFA98860:	01 00 00 00 FF FF FF FF 00 01 00 00 30 46 00 000F..
FFA98870:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
FFA98880:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
FFA98890:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
FFA988A0:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
FFA988B0:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
FFA988C0:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
FFA988D0:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
FFA988E0:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
FFA988F0:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
FFA98900:	78 DA AD 7C 79 7C 13 D7 B5 F0 1D 69 C6 1A 49 23	x.. yi..I#
FFA98910:	69 6C 0B 70 BC C0 48 F2 42 A9 5B 0B C8 42 B3 90	il.p..H.B.[..B..
FFA98920:	D1 E2 05 70 30 59 68 68 DA 24 36 4B 9A E4 39 F1	..p0Yhh.\$6K..9.
FFA98930:	8C 31 30 32 8B 65 03 89 63 36 85 66 E9 A3 79 0D	.102.e..c6.f..y.
FFA98940:	D0 90 47 F2 4A 30 24 F0 4C D2 A4 77 24 2F 7D 0A	..G.J0\$.L..w\$/}.

PSP Module (Data)



In data modules – the signature only covers the header!

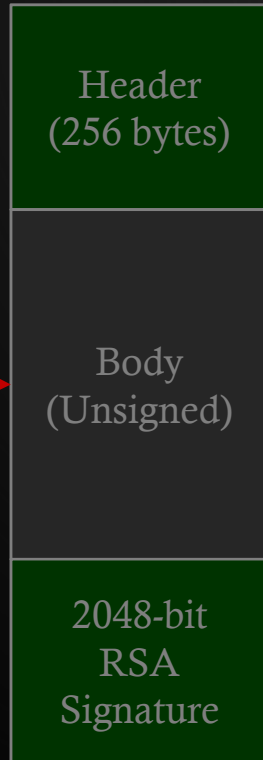




We can use the header and its signature to create any legitimate PSP module



arbitrary code
here

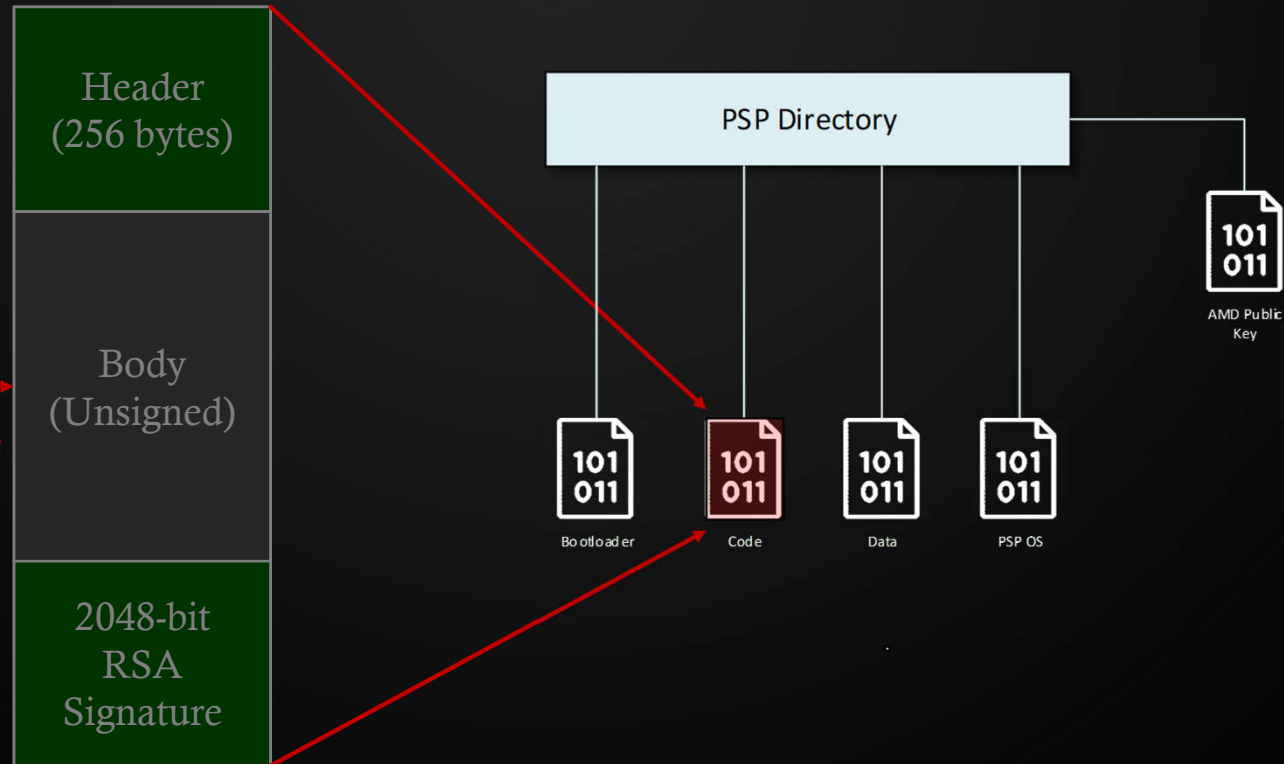




We can replace an existing module, and pass all signature checks!



arbitrary code here





ACHIEVEMENT UNLOCKED
Unsigned code execution

Masterkey-1

DEMO



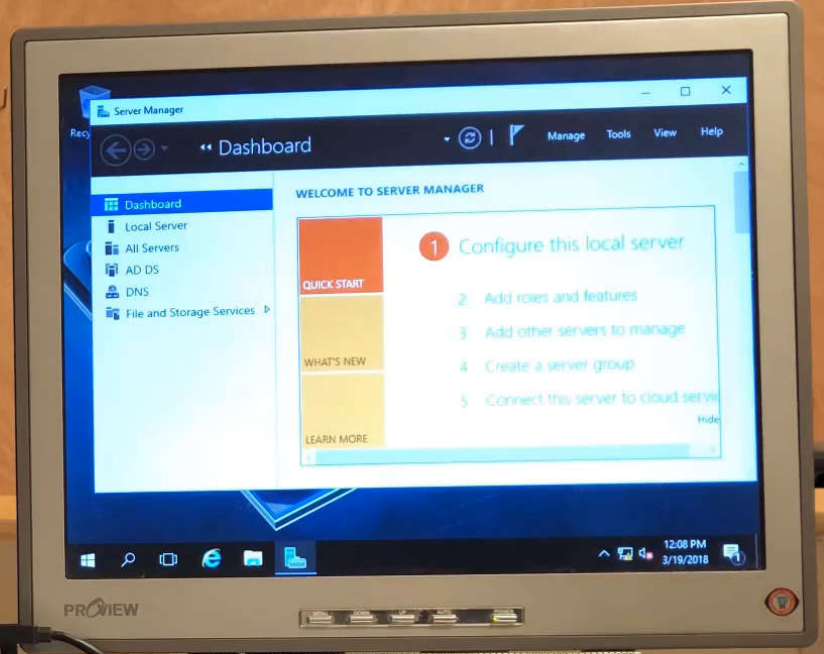
```

Name=AMD EPYC 7261 16-Core Processor
SystemCreationClassName=Win32_ComputerSystem
SystemName=WIN-AB06V3533JN
PS C:\Windows\system32>
PS C:\Windows\system32> Confirm-SecureBootUEFI
True
PS C:\Windows\system32>
PS C:\Windows\system32>
PS C:\Windows\system32> PS C:\Windows\system32> cd \
PS C:\> PS C:\> dir
.
Directory: C:\

Mode                LastWriteTime         Length Name
----                -
d-----          7/16/2016   6:23 AM             PerfLogs
d-----          3/19/2018  11:18 AM             Program Files
d-----          7/16/2016   6:23 AM             Program Files (x86)
d-----          3/18/2018   6:30 AM             Users
d-----          3/19/2018  11:36 AM             Windows
-a-----          3/9/2018   2:21 AM             391792 AFLMINx64.EXE
-a-----          3/9/2018   2:21 AM             17616  amf1drv64.sys
-a-----          3/10/2018   7:56 PM             16777216 BIOS_Tyan_Infected.ROM

PS C:\> PS C:\>

```





ACHIEVEMENT UNLOCKED
Unsigned code execution



Masterkey-1

- ◆ Nice and all, but
 - ◆ Requires restart or entering sleep state before our code can be executed
- ◆ What else can we find?



Fun with Mailboxes

The screenshot displays a memory viewer window with the following data:

Address	Value 1	Value 2	Value 3	Value 4	Info	Text
68	03020100	07060504	00000000	00000000		
70	A0000000	DD06F900	00000000	00000000		
80	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF		
90	FFFFFFFF	00000000	50000078	FFFFFFFF		
A0	00000400	00000000	00000000	00000000		
B0	00000000	00000000	00000000	00000000		
C0	00000000	00000000	00000000	00000000		
D0	00000000	00000000	00000000	00000000		
E0	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF		
F0	FFFFFFFF	FFFFFFFF	FFFFFFFF	00000000		

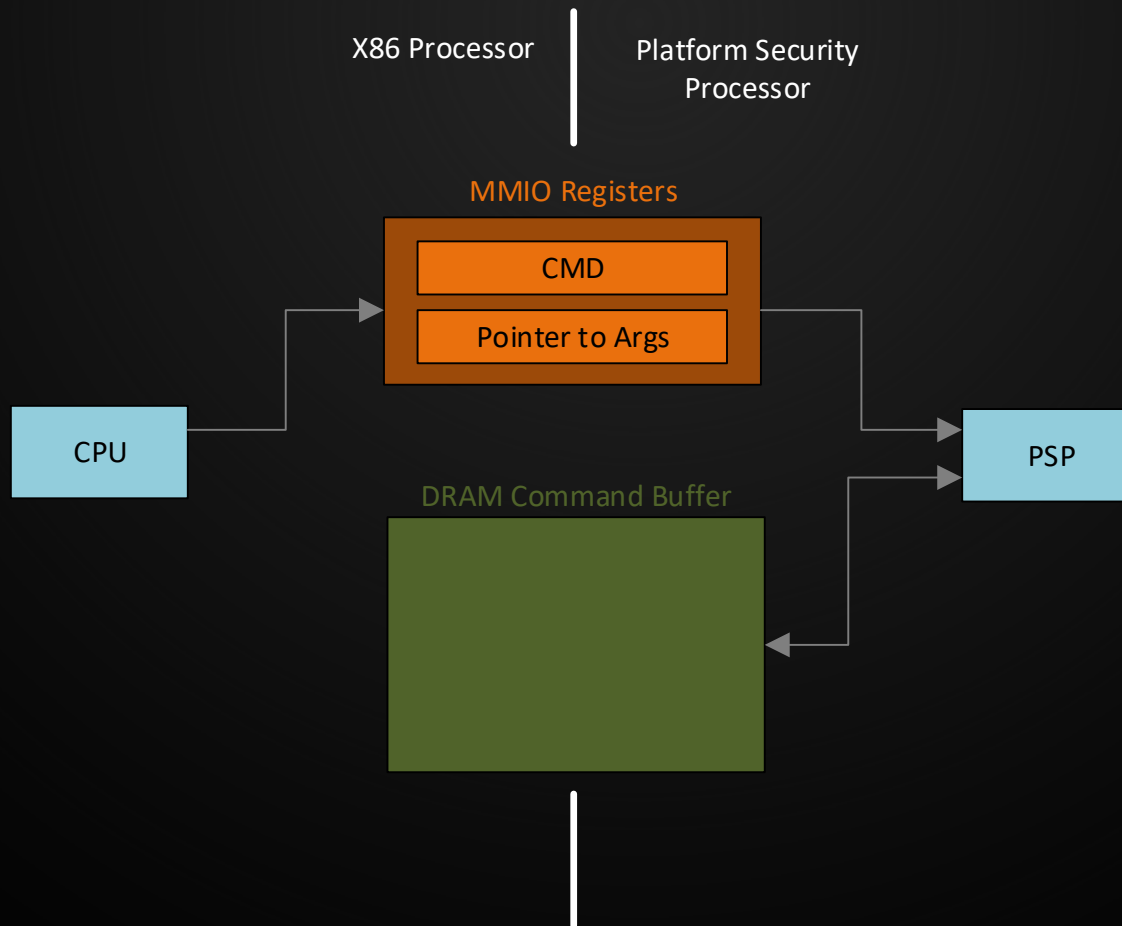
Hardware section:

Hardware	Value 1	Value 2	Value 3	Value 4
C0	00000000	00000000	00000000	00000000
D0	00000000	00000000	00000000	00000000
E0	00000000	00000000	00000000	00000000
F0	00000000	00000000	00000000	00000000

Text view details:

```
0 1 2 3 4 5 6 7 8 9 A B C D E F
[ ] [ ] [ ] [ ]
*** Build by ara
shid1@MSDN-DRTVD
R00, svn@[unkno
wn] ###
[ ] [ ] [ ] [ ]
```


Mailbox communication is done through MMIO and shared buffers in RAM





Partial list of Mailbox commands

```
/* x86 to PSP commands */  
#define MBOX_BIOS_CMD_DRAM_INFO      0x01  
#define MBOX_BIOS_CMD_SMM_INFO      0x02  
#define MBOX_BIOS_CMD_SX_INFO      0x03  
#define MBOX_BIOS_CMD_RSM_INFO      0x04  
#define MBOX_BIOS_CMD_PSP_QUERY     0x05  
#define MBOX_BIOS_CMD_BOOT_DONE     0x06  
#define MBOX_BIOS_CMD_CLEAR_S3_STS  0x07  
#define MBOX_BIOS_CMD_S3_DATA_INFO  0x08  
#define MBOX_BIOS_CMD_NOP           0x09  
#define MBOX_BIOS_CMD_SMU_FW        0x19  
#define MBOX_BIOS_CMD_SMU_FW2      0x1a  
#define MBOX_BIOS_CMD_ABORT        0xfe
```



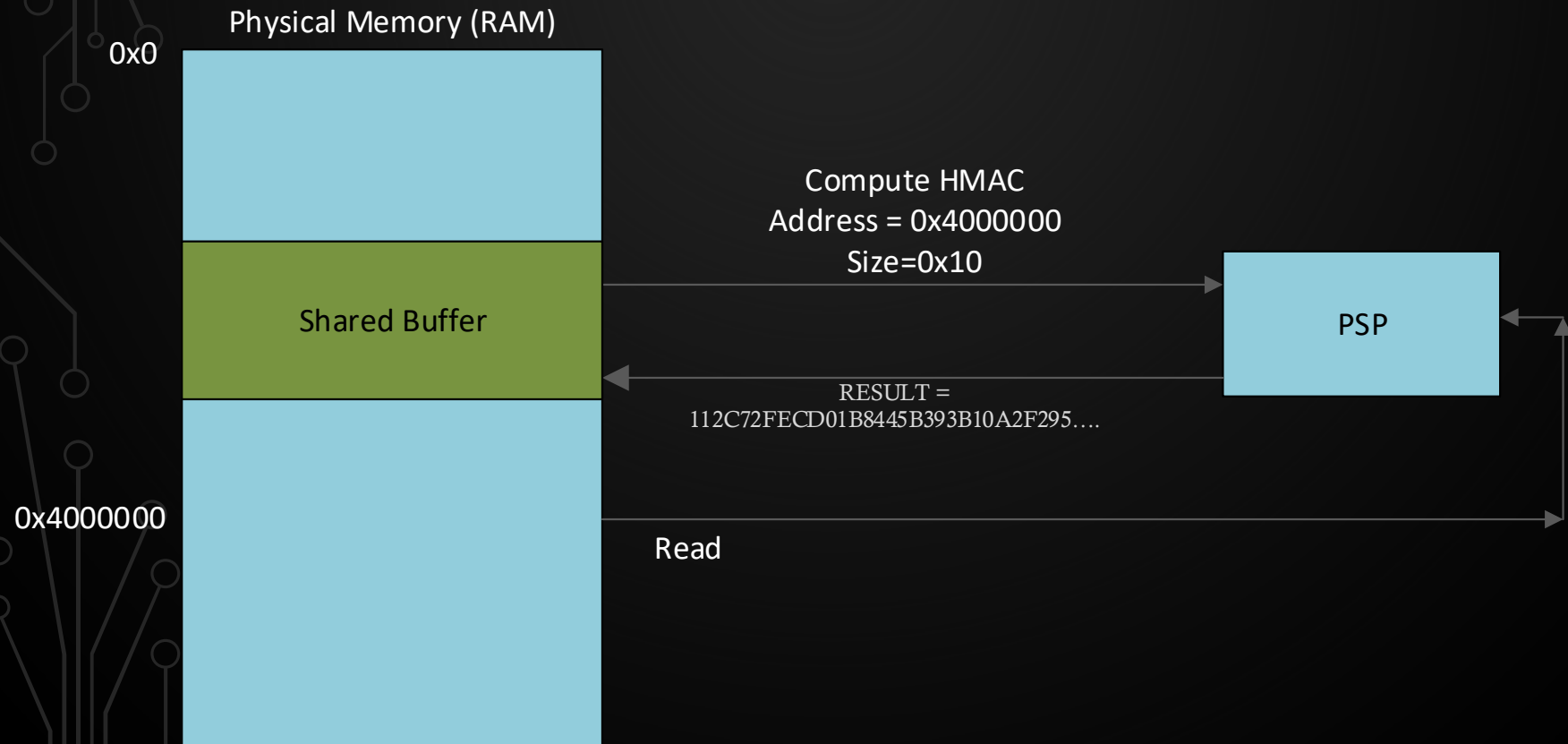
Command 08h = Compute HMAC

```
/* x86 to PSP commands */  
#define MBOX_BIOS_CMD_DRAM_INFO      0x01  
#define MBOX_BIOS_CMD_SMM_INFO      0x02  
#define MBOX_BIOS_CMD_SX_INFO      0x03  
#define MBOX_BIOS_CMD_RSM_INFO      0x04  
#define MBOX_BIOS_CMD_PSP_QUERY      0x05  
#define MBOX_BIOS_CMD_BOOT_DONE      0x06  
#define MBOX_BIOS_CMD_CLEAR_S3_STS  0x07  
#define MBOX_BIOS_CMD_S3_DATA_INFO  0x08  
#define MBOX_BIOS_CMD_NOP            0x09  
#define MBOX_BIOS_CMD_SMU_FW         0x19  
#define MBOX_BIOS_CMD_SMU_FW2        0x1a  
#define MBOX_BIOS_CMD_ABORT          0xfe
```

```
struct mbox_s3_data_info_buffer {  
    QWORD address;  
    QWORD size;  
    unsigned char hmac_out[32];  
}
```



PSP reads memory for you

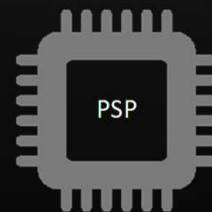
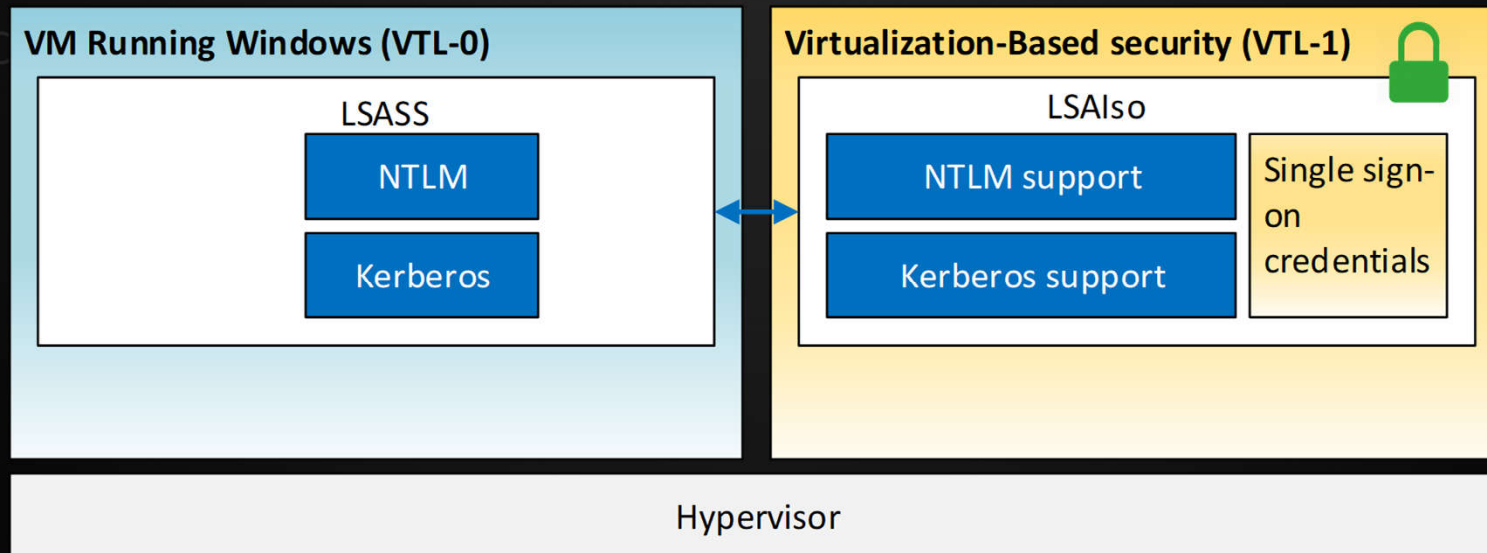




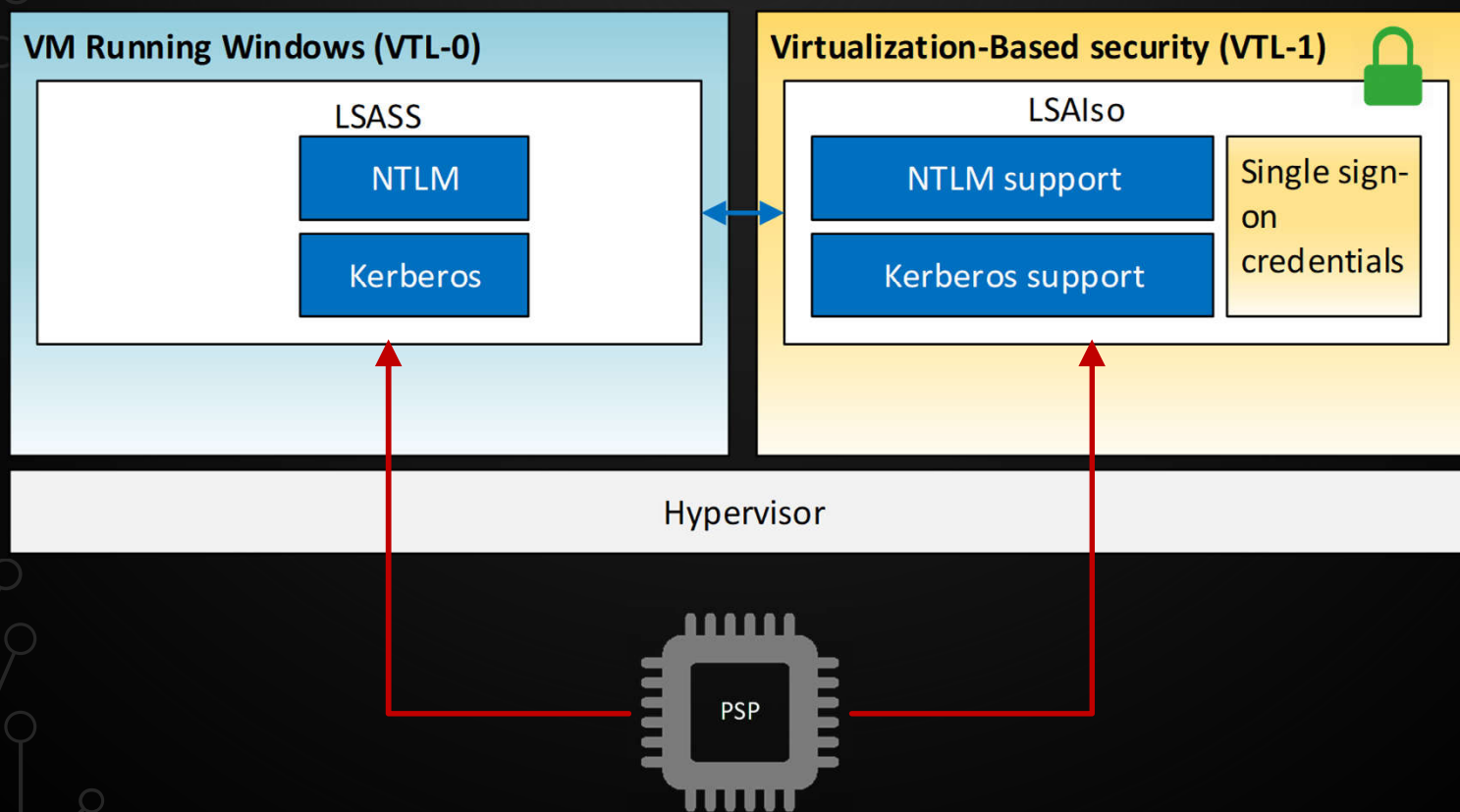
Privileged Memory Locations

- ◆ SMM
- ◆ Other virtual machines
 - ◆ Credential Guard ?

Credential Guard - Credentials are stored on isolated Virtual Machine

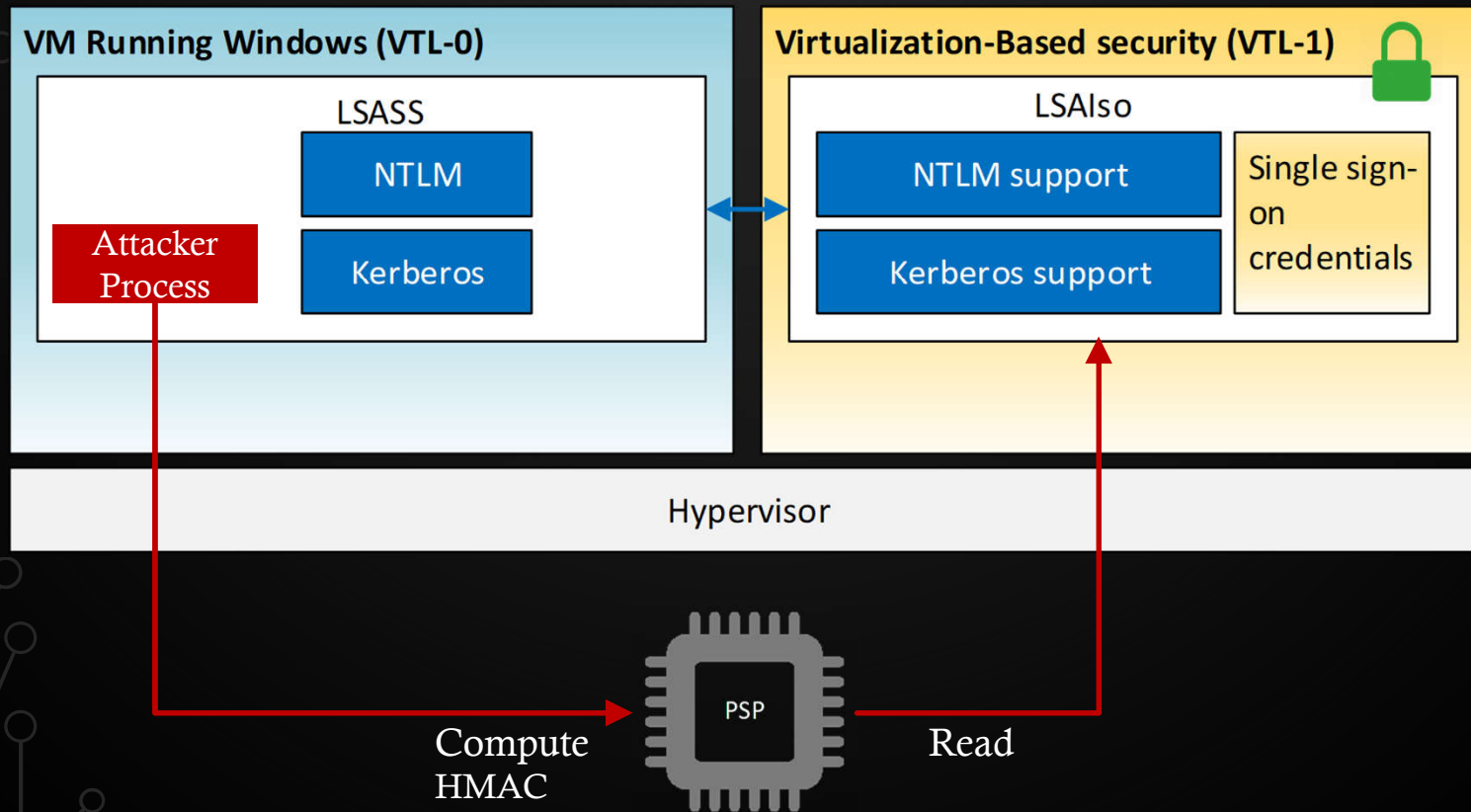


The PSP has full access to physical memory, including VTL-1





The PSP will read privileged memory for you





What if we compute HMAC on a single-byte?

`0x00 = 743771B093E5F45526274762E2723D56B6E82273402C4D67C65D48563C658502`



We can build a translation table!

0x00 = 743771B093E5F45526274762E2723D56B6E82273402C4D67C65D48563C658502
0x01 = 93F17EA603A23F3A55BCC80206EE7D8B8CEA7B5868470FFE2DF67DB17E2FA391
0x02 = 4749A3A56843611245E3C7070CE396C6225546EB03E0EC9D6F6423815E98E901
0x03 = FC5B2C165C3BC3CA503E62CE800712F02FB814B44AFE429915C4C1BE2B3B1B2F
0x04 = 112C72FEC01B8445B393B10A2F29552528386185522913BE44684DE78965679
0x05 = 80429C59A5704A361112A27FF6C72899AB2E32AA57F32411AACDC468CB180B3E
0x06 = 1BA839DE8C412DF5D5368A7209C29F30D976B0CFE3D95925FEC1156AA0C02E24
0x07 = C906BB0687AD289C4E063ACA3355A84CCDF4528CC45E970E23B779F32C0435D7
0x08 = CE52670F68F0CA109128E1F57E271BD470EB983A430D438E71BDFD0B9228EA6D
0x09 = 629DA6A43E2D9B5C18189013C486BCB16DA10F2F411338D070B112E8332946CC
0x0A = 158F90A2B67747F15F5697FC3E9E66D86CBFD9CB87839C2C0C735080C2D5BC19
0x0B = AA69F104AFF8761B45A62F3F002C58A8B0B181FFCE745A84E8AD75946DDAC7A1
0x0C = CAB9E96F2E35373EA22AC46F3E4352B6A60C5A5E0C0F4A29BC29FDC7FDCCEEB3

...



ACHIEVEMENT UNLOCKED
Dump VTL-1 Memory



Ryzenfall-1

- ◆ Dump passwords & hashes in VTL-1 memory



Ryzenfall-1

- ◇ Cool. But can we hijack the PSP in runtime?

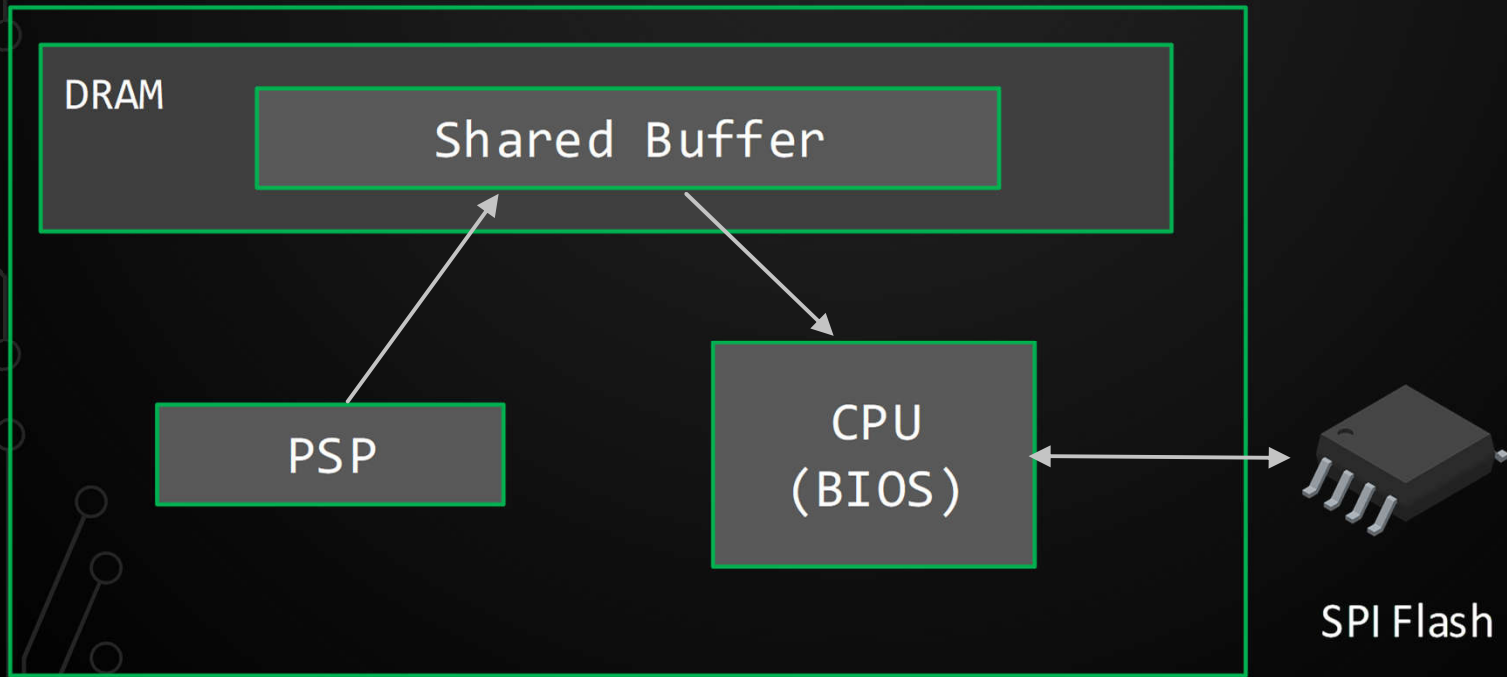


SMM_INFO – Initialize PSP-to-BIOS communications

```
/* x86 to PSP commands */  
#define MBOX_BIOS_CMD_DRAM_INFO      0x01  
#define MBOX_BIOS_CMD_SMM_INFO      0x02  
#define MBOX_BIOS_CMD_SX_INFO       0x03  
#define MBOX_BIOS_CMD_RSM_INFO      0x04  
#define MBOX_BIOS_CMD_PSP_QUERY     0x05  
#define MBOX_BIOS_CMD_BOOT_DONE     0x06  
#define MBOX_BIOS_CMD_CLEAR_S3_STS  0x07  
#define MBOX_BIOS_CMD_S3_DATA_INFO  0x08  
#define MBOX_BIOS_CMD_NOP           0x09  
#define MBOX_BIOS_CMD_SMU_FW        0x19  
#define MBOX_BIOS_CMD_SMU_FW2       0x1a  
#define MBOX_BIOS_CMD_ABORT         0xfe
```

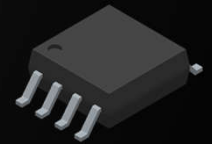
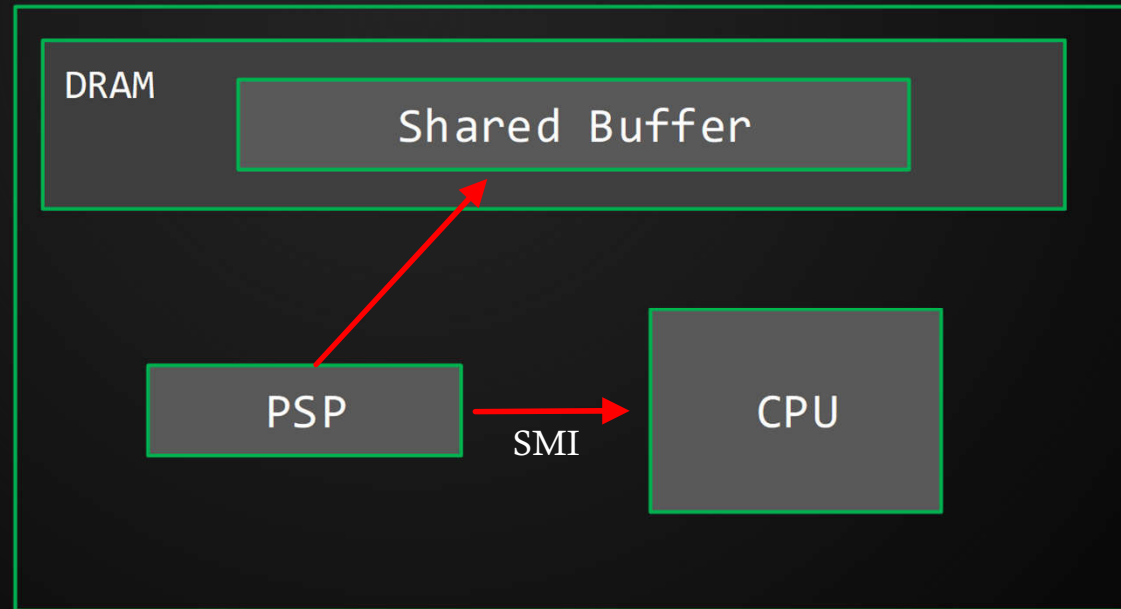


PSP uses PSP-to-BIOS to access SPI Flash





[1] The PSP first writes its request to the P2C shared buffer.

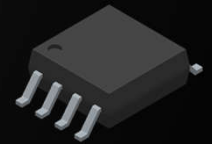
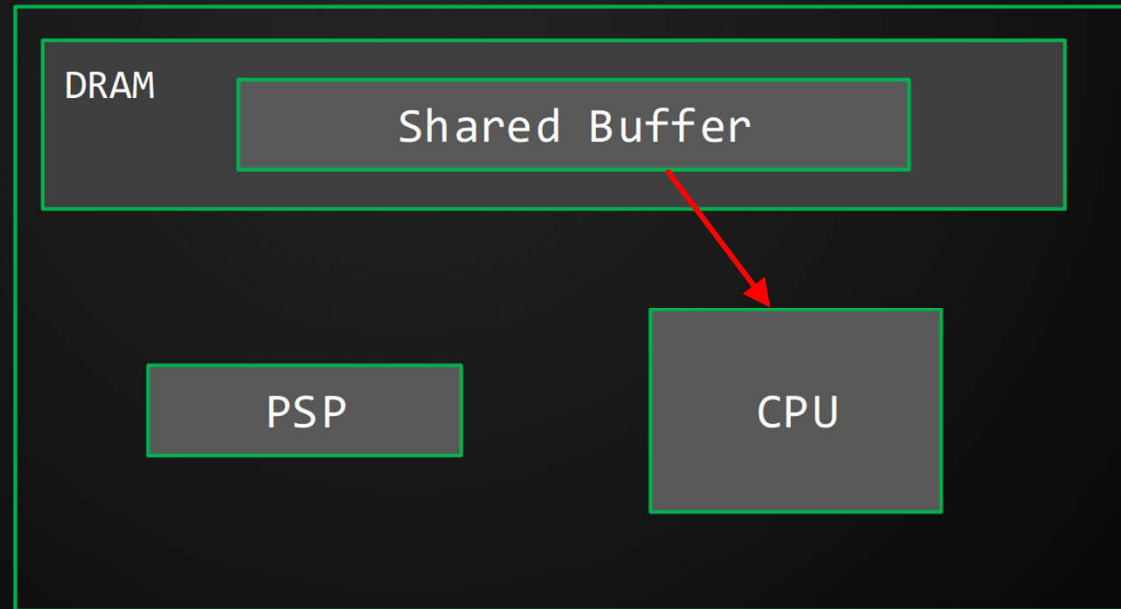


SPI Flash

P2C Interface Flow



[2] SMI triggers SMM code to process the request from the shared buffer.

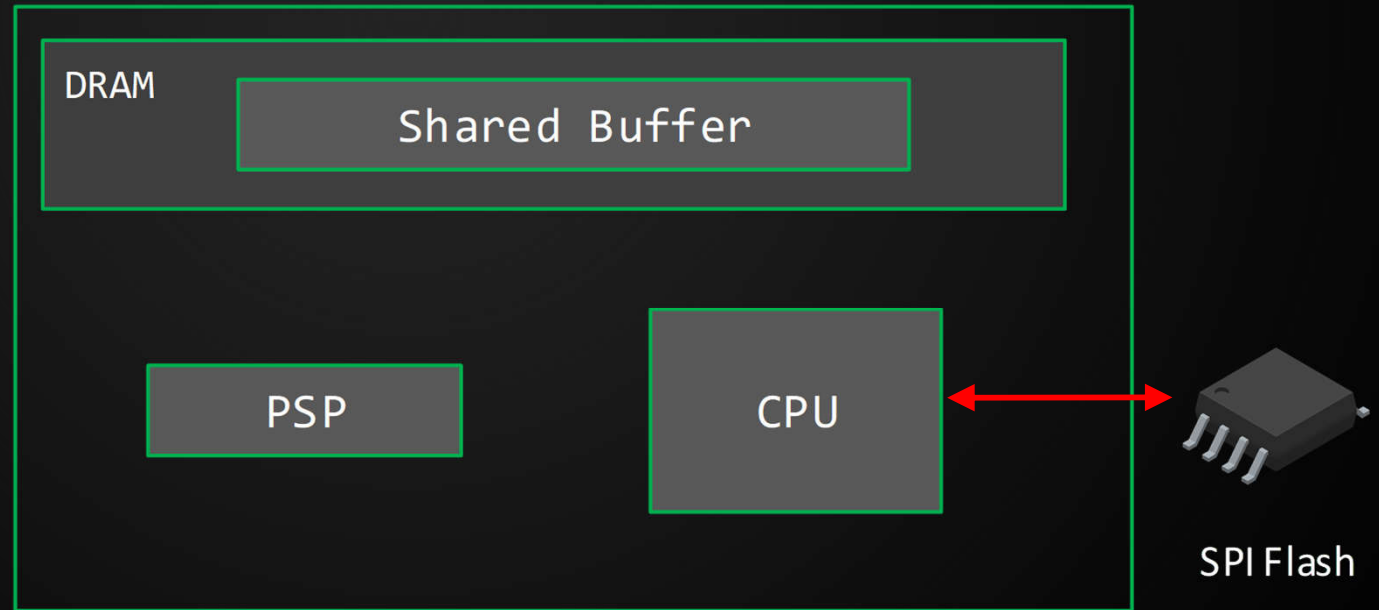


SPI Flash

P2C Interface Flow



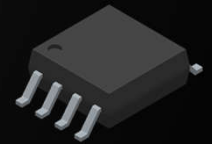
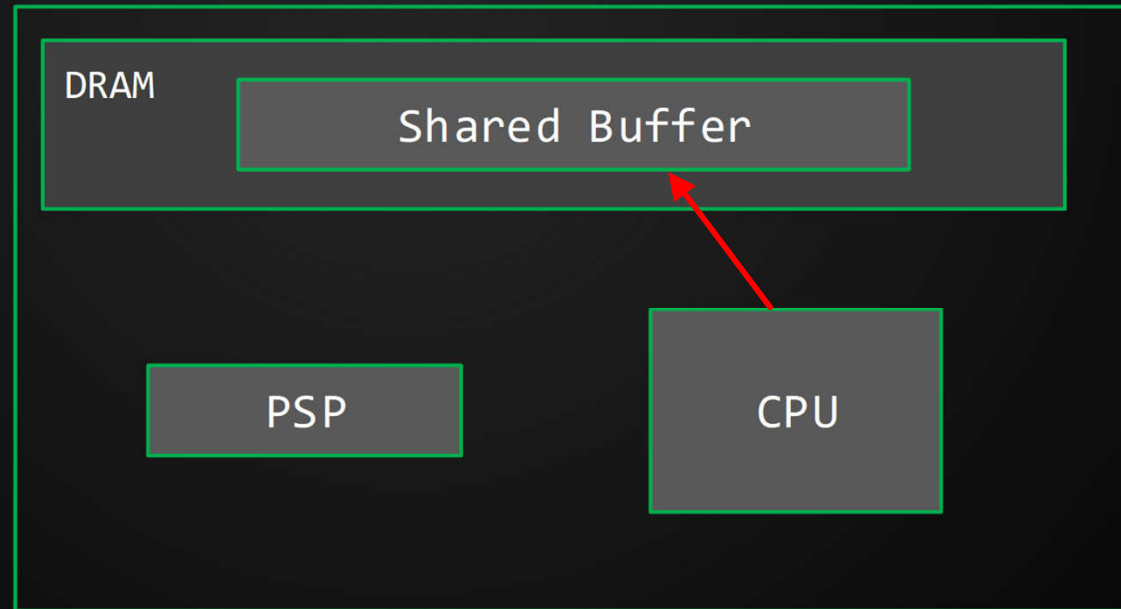
[3] SMM code reads/writes data to/from SPI flash on behalf of the PSP.



P2C Interface Flow



[4] Results are written back to the shared buffer, and the PSP is signaled.

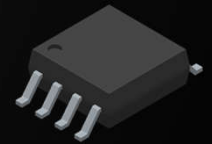
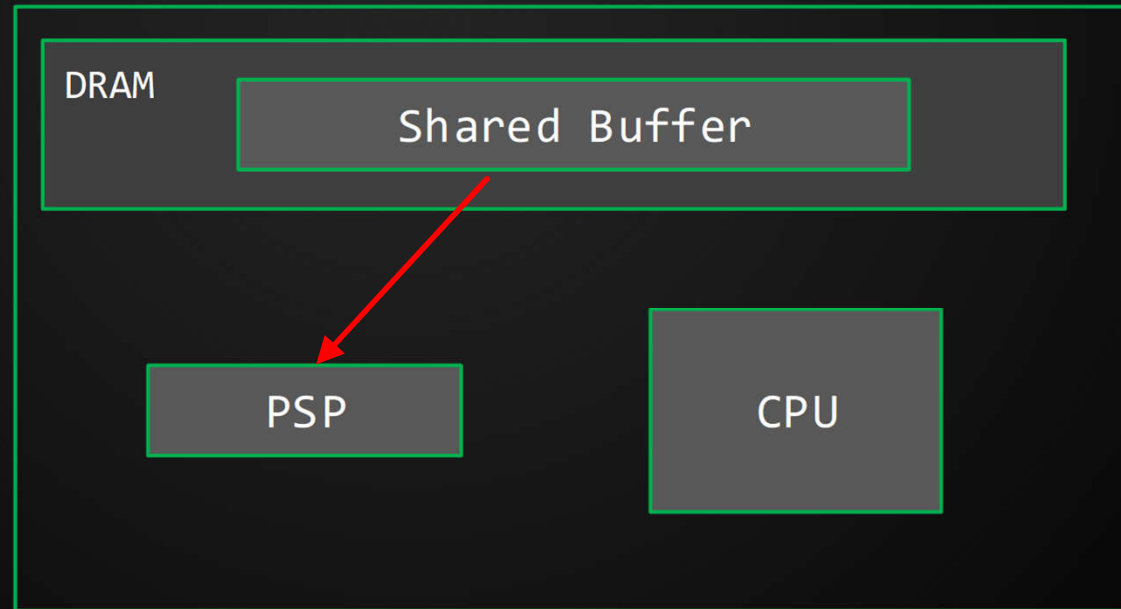


SPI Flash

P2C Interface Flow



[5] PSP reads the result from the same shared buffer.

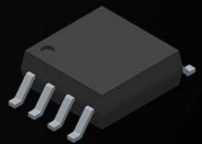
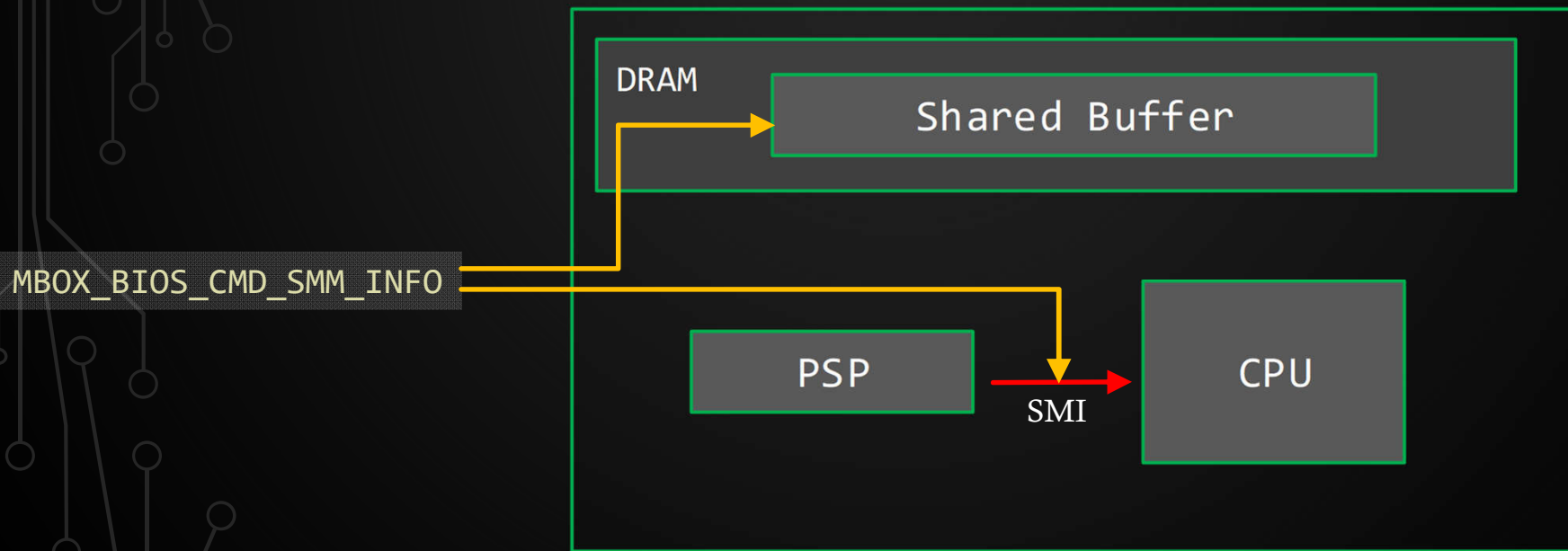


SPI Flash

P2C Interface Flow

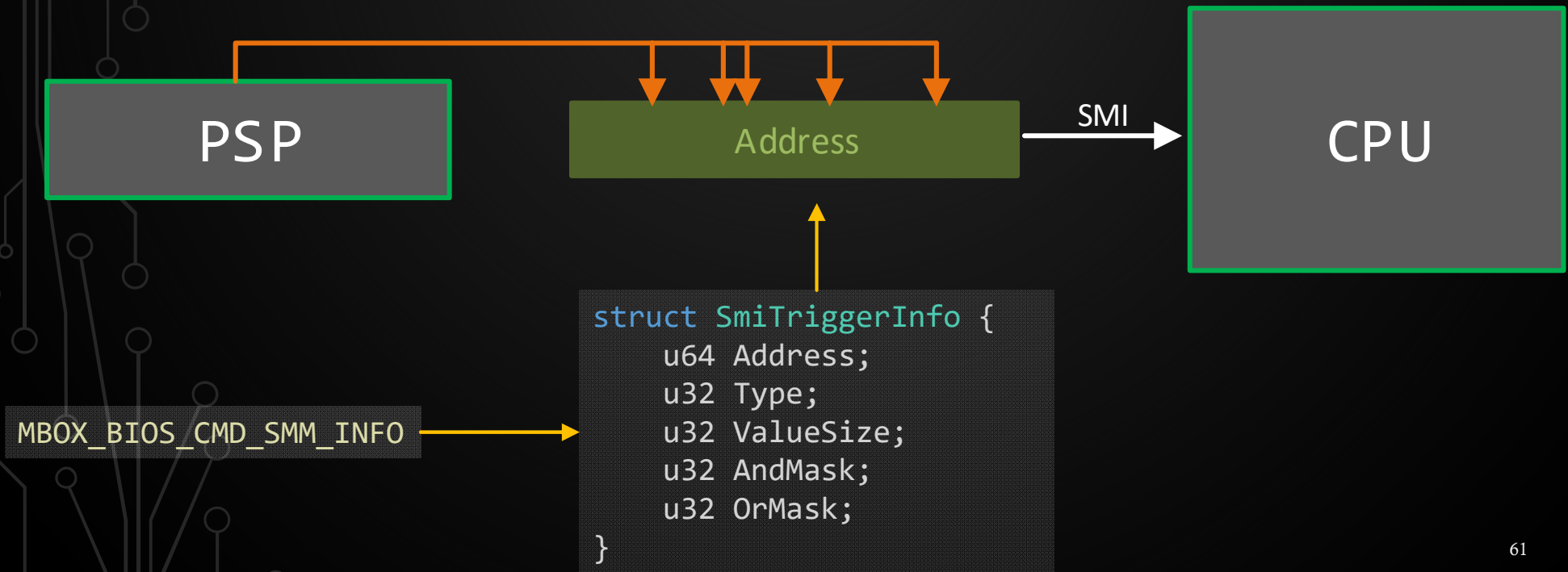


SMM_INFO sets the shared buffer and SMI information



SPI Flash

To trigger an SMI, the PSP applies a bitmask to an address in memory





SMM_INFO Handler initializes PSP-to-BIOS in PSP Memory

```
VOID HandleSmmInfo(struct SmmInfoReqBuffer *pSmmInfo)
{
    *GLOBAL_PSP_TO_BIOS_BASE = pSmmInfo->PspToBiosBase;
    memcpy(&GLOBAL_SMI_TRIGGER_INFO, pSmmInfo->SmiTrigInfo,
        sizeof(struct SmiTriggerInfo));
}
```



GenerateSMI - SMI trigger routine in PSP Firmware

```
VOID GenerateSMI() // Flip some bits in memory to generate an SMI
{
    DWORD dwSmiValue;
    DWORD dwTriggerAddress;

    struct SmiTriggerInfo *pSmiTriggerInfo = &GLOBAL_SMI_TRIGGER_INFO;
    if (1 << pSmiTriggerInfo->ValueSize != 4)
        goto err;

    /// ... redacted for brevity...
    DWORD size = 1 << pSmiTriggerInfo->ValueSize;
    dwTriggerAddress = dma_map(pSmiTriggerInfo->Address, size, &dwTriggerAddress);

    memcpy(&dwSmiValue, dwTriggerAddress, size);
    dwSmiValue = dwSmiValue & pSmiTriggerInfo->AndMask;
    dwSmiValue = dwSmiValue | pSmiTriggerInfo->OrMask;
    memcpy(dwTriggerAddress, &dwSmiValue, size);
    // ...
}
```



ValueSize is sanity-checked

```
VOID GenerateSMI() // Flip some bits in memory to generate an SMI
{
    DWORD dwSmiValue;
    DWORD dwTriggerAddress;

    struct SmiTriggerInfo *pSmiTriggerInfo = &GLOBAL_SMI_TRIGGER_INFO;
    if (1 << pSmiTriggerInfo->ValueSize != 4)
        goto err;

    /// ... redacted for brevity...
    DWORD size = 1 << pSmiTriggerInfo->ValueSize;
    dwTriggerAddress = dma_map(pSmiTriggerInfo->Address, size, &dwTriggerAddress);

    memcpy(&dwSmiValue, dwTriggerAddress, size);
    dwSmiValue = dwSmiValue & pSmiTriggerInfo->AndMask;
    dwSmiValue = dwSmiValue | pSmiTriggerInfo->OrMask;
    memcpy(dwTriggerAddress, &dwSmiValue, size);
    /// ...
}
```




pSmiTriggerInfo->Address is mapped into PSP address space

```
VOID GenerateSmi() // Flip some bits in memory to generate an SMI
{
    DWORD dwSmiValue;
    DWORD dwTriggerAddress;

    struct SmiTriggerInfo *pSmiTriggerInfo = &GLOBAL_SMI_TRIGGER_INFO;
    if (1 << pSmiTriggerInfo->ValueSize != 4)
        goto err;

    /// ... redacted for brevity...
    DWORD size = 1 << pSmiTriggerInfo->ValueSize;
    dwTriggerAddress = dma_map(pSmiTriggerInfo->Address, size, &dwTriggerAddress);

    memcpy(&dwSmiValue, dwTriggerAddress, size);
    dwSmiValue = dwSmiValue & pSmiTriggerInfo->AndMask;
    dwSmiValue = dwSmiValue | pSmiTriggerInfo->OrMask;
    memcpy(dwTriggerAddress, &dwSmiValue, size);
    // ...
}
```



The PSP applies the bitmask and writes back to RAM

```
VOID GenerateSMI() // Flip some bits in memory to generate an SMI
{
    DWORD dwSmiValue;
    DWORD dwTriggerAddress;

    struct SmiTriggerInfo *pSmiTriggerInfo = &GLOBAL_SMI_TRIGGER_INFO;
    if (1 << pSmiTriggerInfo->ValueSize != 4)
        goto err;

    /// ... redacted for brevity...
    DWORD size = 1 << pSmiTriggerInfo->ValueSize;
    dwTriggerAddress = dma_map(pSmiTriggerInfo->Address, size, &dwTriggerAddress);

    memcpy(&dwSmiValue, dwTriggerAddress, size);
    dwSmiValue = dwSmiValue & pSmiTriggerInfo->AndMask;
    dwSmiValue = dwSmiValue | pSmiTriggerInfo->OrMask;
    memcpy(dwTriggerAddress, &dwSmiValue, size);
    /// ...
}
```



ValueSize is dereferenced twice! Hmm..

```
VOID GenerateSmi() // Flip some bits in memory to generate an SMI
{
    DWORD dwSmiValue;
    DWORD dwTriggerAddress;

    struct SMI_TRIGGER_INFO *pSmiTriggerInfo = &GLOBAL_SMI_TRIGGER_INFO;
    if (1 << pSmiTriggerInfo->ValueSize != 4)
        goto err;

    /// ... redacted for brevity...
    DWORD size = 1 << pSmiTriggerInfo->ValueSize;
    dwTriggerAddress = dma_map(pSmiTriggerInfo->Address, size, &dwTriggerAddress);

    memcpy(&dwSmiValue, dwTriggerAddress, size);
    dwSmiValue = dwSmiValue & pSmiTriggerInfo->AndMask;
    dwSmiValue = dwSmiValue | pSmiTriggerInfo->OrMask;
    memcpy(dwTriggerAddress, &dwSmiValue, size);
    // ...
}
```



Double fetch -- We can switch ValueSize under its feet!

```
VOID GenerateSmi() // Flip some bits in memory to generate an SMI
{
    DWORD dwSmiValue;
    DWORD dwTriggerAddress;

    struct SmiTriggerInfo *pSmiTriggerInfo = &GLOBAL_SMI_TRIGGER_INFO;
    if (1 << pSmiTriggerInfo->ValueSize != 4)
        goto err;

    /// ... redacted for brevity...
    DWORD size = 1 << pSmiTriggerInfo->ValueSize;
    dwTriggerAddress = dma_map(pSmiTriggerInfo->Address, size, &dwTriggerAddress);

    memcpy(&dwSmiValue, dwTriggerAddress, size);
    dwSmiValue = dwSmiValue & pSmiTriggerInfo->AndMask;
    dwSmiValue = dwSmiValue | pSmiTriggerInfo->OrMask;
    memcpy(dwTriggerAddress, &dwSmiValue, size);
    // ...
}
```

Window of
opportunity



Stack overflow!

```
VOID GenerateSMI() // Flip some bits in memory to generate an SMI
{
    DWORD dwSmiValue;
    DWORD dwTriggerAddress;

    struct SmiTriggerInfo *pSmiTriggerInfo = &GLOBAL_SMI_TRIGGER_INFO;
    if (1 << pSmiTriggerInfo->ValueSize != 4)
        goto err;

    /// ... redacted for brevity...
    DWORD size = 1 << pSmiTriggerInfo->ValueSize;
    dwTriggerAddress = dma_map(pSmiTriggerInfo->Address, size, &dwTriggerAddress);

    memcpy(&dwSmiValue, dwTriggerAddress, size);
    dwSmiValue = dwSmiValue & pSmiTriggerInfo->AndMask;
    dwSmiValue = dwSmiValue | pSmiTriggerInfo->OrMask;
    memcpy(dwTriggerAddress, &dwSmiValue, size);
    /// ...
}
```



ACHIEVEMENT UNLOCKED
Hijack the PSP



Ryzenfall-4

- ◆ Double fetch leads to stack overflow
- ◆ No stack cookies, no ASLR or other exploit mitigations



Ryzenfall-4

DEMO



Memory

Address = 00000000FD910500

Address	03020100	07060504	0B0A0908	0F0E0D0C
68	03020100	07060504	0B0A0908	0F0E0D0C
00	00000000	00000000	000000A3	00000000
10	00000000	00000000	00000000	00000000
20	00000000	00000000	00004000	DC508000
30	00000000	00004000	DC50C000	00000000
40	00000000	80010000	0000000F	EE265000
50	00000000	00000000	00000000	00000000
60	00000000	00000000	00000000	0009CEC3
70	A0000000	DD06F900	00000000	00000000
80	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF
90	FFFFFFFF	00000000	50000078	FFFFFFFF
A0	00000400	00000000	00000000	00000000
B0	00000000	00000000	00000000	00000000
C0	00000000	00000000	00000000	00000000
D0	00000000	00000000	00000000	00000000
E0	FFFFFFFF	FFFFFFFF	FFFFFFFF	FFFFFFFF
F0	FFFFFFFF	FFFFFFFF	FFFFFFFF	00000000

Info Text

```

*
@ P I
@ P I
P & o
#
x P

```

Info Text

```

*** Build by ara
shid1@MSDN-DRTVD
R00, svn@[unkno
wn] ###

```

Hardware

C0	00000000	00000000	00000000	00000000
D0	00000000	00000000	00000000	00000000
E0	00000000	00000000	00000000	00000000
F0	00000000	00000000	00000000	00000000

Hardware

Starting Memory function...





Problem: Credential Guard Breaks Mimikatz

Credential Guard causes Mimikatz to fail



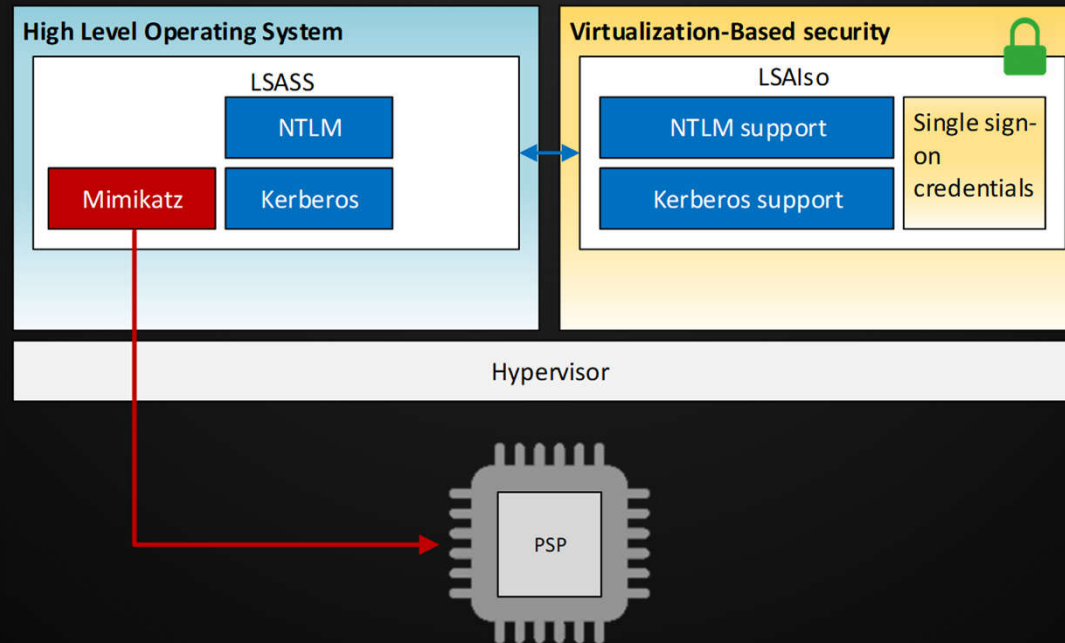


Solution: Use PSP to Break Into Isolated VM

Bypass Credential Guard in 3 simple steps



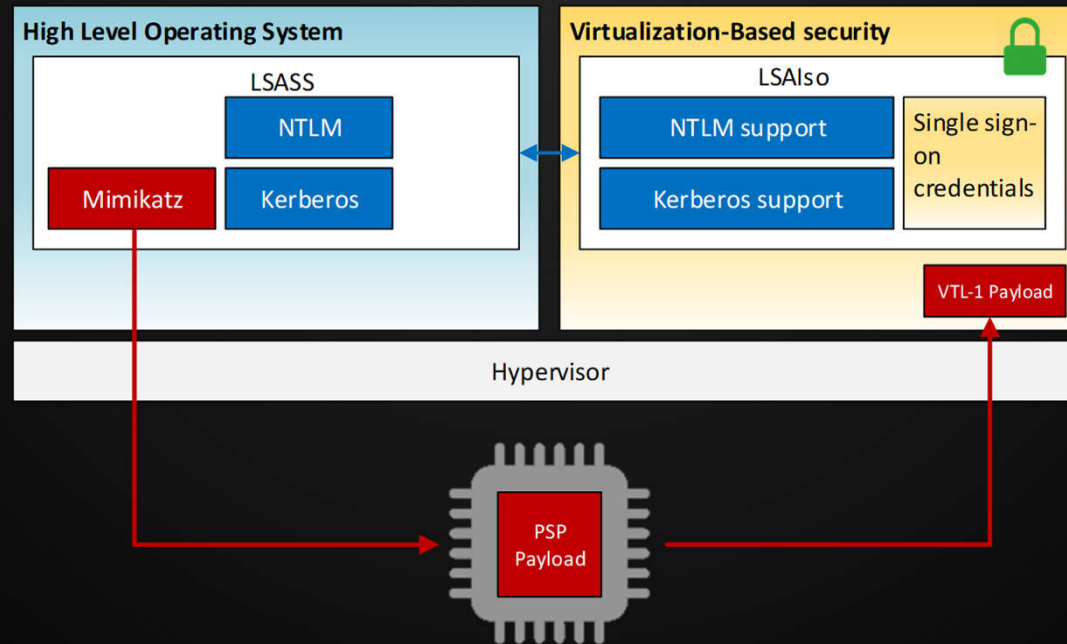
Step 1: Inject payload into PSP



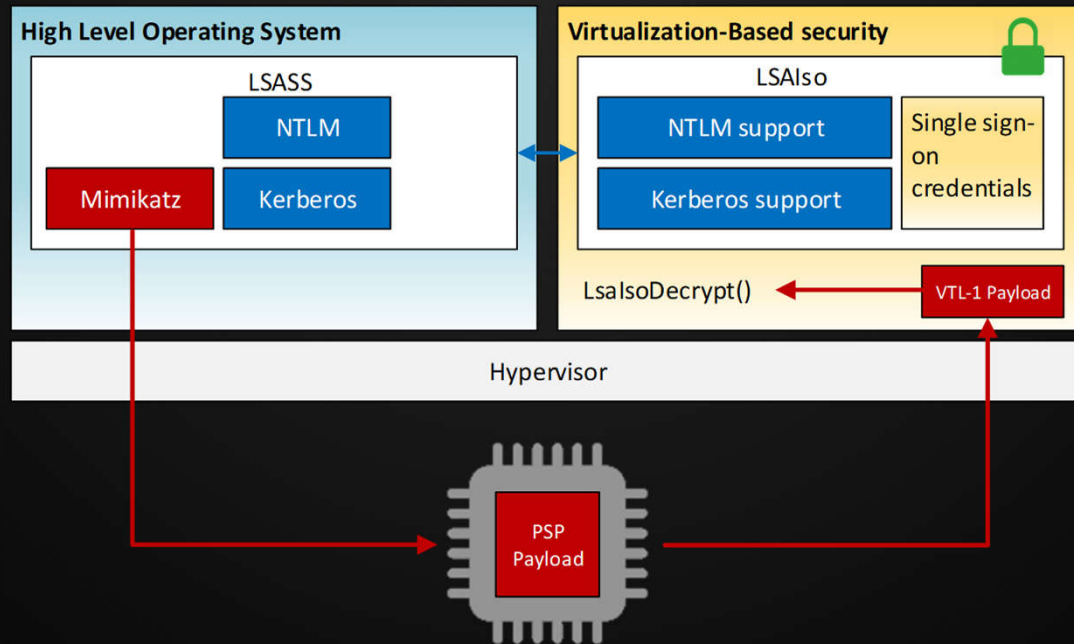
◆ Either Masterkey or Ryzenfall will do



Step 2: From PSP, Inject into Isolated LSA

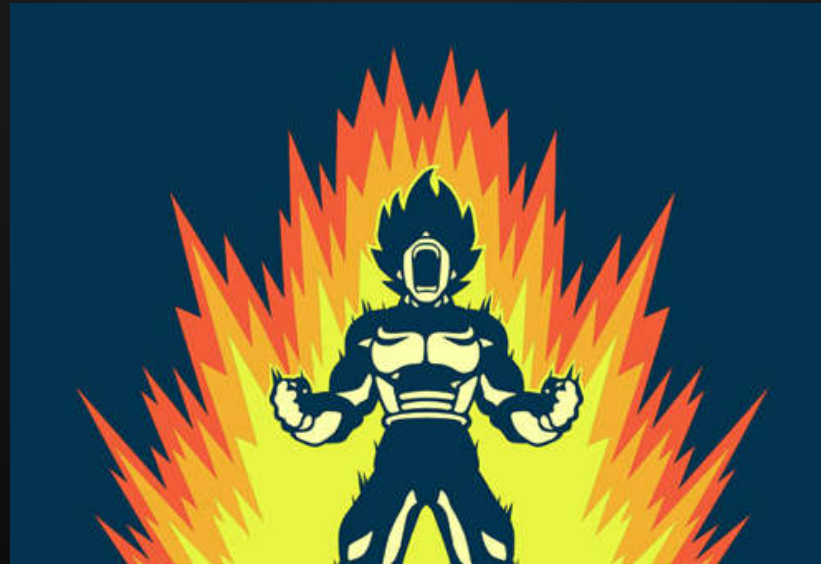


Step 3: Profit!





Mimikatz



Power-Up

Mimikatz -- PSP Power-Up Edition

```
kerberos :
ssp :
credman :

Authentication Id : 0 ; 999 (00000000:000003e7)
Session           : UndefinedLogonType from 0
User Name         : BIOSTAR$
Domain            : D2016
Logon Server      : (null)
Logon Time        : 3/21/2018 11:12:13 AM
SID               : S-1-5-18

msv :
tspkg :
wdigest :
 * Username : BIOSTAR$
 * Domain   : D2016
 * Password : (null)
kerberos :
 * Username : biostar$
 * Domain   : D2016.COM
 * Password : (null)
ssp :
credman :

mimikatz # exit
Bye!

C:\>
```

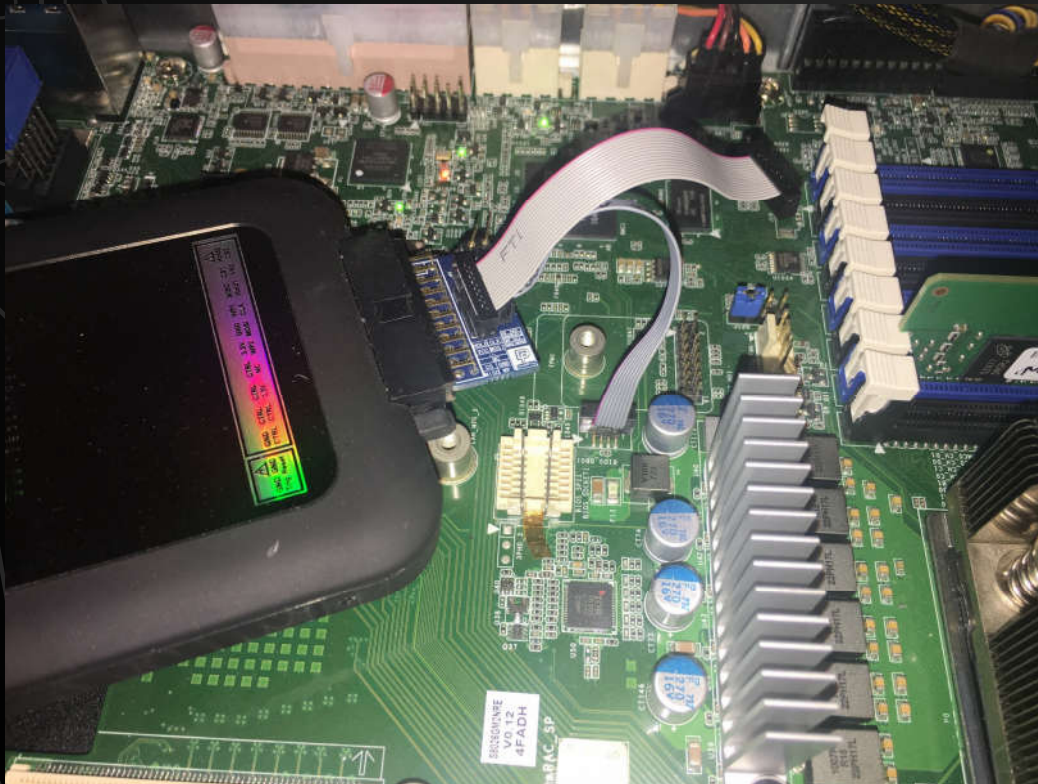
11:59 AM
3/21/2018



Thank you!



Side note: Debugging PSP signature exploits



SPI Trace Viewer

Save Trace Display Trace Clear Buffer Translation...

TIMESTAMP(μ)	CNT	CMD	ADDRESS	DATA
2.41564066	1	03		FF FF FF FF
2.42601388	2	03	00 00 00	FF FF FF FF
2.89880151	3	03	1F FF C0	E9 A4 FC 8D A4 24 00 00 00 00 8D 98 00 00 00 00 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF DF 07 FC FF 10 00 8D A4 24 00 00 00 00 8D 49 00 0F 09 E9 23 FF FF FF 00 00 20 00 00 00 00 FC FF
2.89913081	4	03	1F FF 00	FF 02 50 02 58 02 59 02 68 02 69 02 6A 02 68 02 6C 02 6D 02 6E 02 6F 02 FA B0 01 E6 80 88 00 F0 8E D8 BE F0 FF 80 3C EA 75 05 EA 58 E0 00 F0 B0 02 E6 80 66 2E 0F 01 16 A8 FF 0F 20 C0 0C 01 0F
2.89916629	5	03	1F FF F0	0F 09 E9 23
2.89917044	6	03	1F FF 00	FF 02 50 02 58 02 59 02 68 02 69 02 6A 02 68 02 6C 02 6D 02 6E 02 6F 02 FA B0 01 E6 80 88 00 F0 8E D8 BE F0 FF 80 3C EA 75 05 EA 58 E0 00 F0 B0 02 E6 80 66 2E 0F 01 16 A8 FF 0F 20 C0 0C 01 0F
2.89920418	7	03	1F FF A8	47 00 80 FF
2.89920829	8	03	1F FF A8	FF 02 50 02 58 02 59 02 68 02 69 02 6A 02 68 02 6C 02 6D 02 6E 02 6F 02 FA B0 01 E6 80 88 00 F0 8E D8 BE F0 FF 80 3C EA 75 05 EA 58 E0 00 F0 B0 02 E6 80 66 2E 0F 01 16 A8 FF 0F 20 C0 0C 01 0F
2.89921423	9	03	1F FF 00	47 00 80 FF
2.89924653	10	03	1F FF 40	22 C0 FC B8 08 0E 8E D8 BE C0 BE D0 8E E0 BE E8 66 EA B5 FC FF 10 00 8D A4 24 00 00 00 00 90 00 00 00 00 00 00 00 00 FF FF 00 00 00 93 CF 00 FF FF 00 00 98 CF 00 FF FF 00 00 93 CF 00
2.89928545	11	03	1F FC 80	E4 71 0A C0 75 1A 66 B9 EE 00 00 0F 32 66 25 02 00 00 80 75 08 66 00 04 00 80 0F 30 EB 11 66 88 06 00 00 66 33 D2 66 B9 50 02 00 00 0F 30 FA F4 EB FC B9 A0 01 00 00 0F 32 0F BA F0 16 73 02 0F 30 89 18 00 00 0F 32 83 E2 F0 25 FF
2.89931790	12	03	1F FC C0	0F 00 00 00 00 00 E0 FE 0F 30 E9 90 FD FF FF B0

Filter

Write Enable(0x06) Normal Read(0x03) Chip Erase(0xC7) Address Range: Start: 0x0 End: 0xFFFFF Mask Non Significant Address Bits

Write Disable(0x04) Fast Read(0x0B) Chip Erase(0x60)

Read Status Register(0x05) Page Program(0x02) Read ID(0x9F) Others: 0x

Write Status Register(0x01) Sector Erase(0xD6)