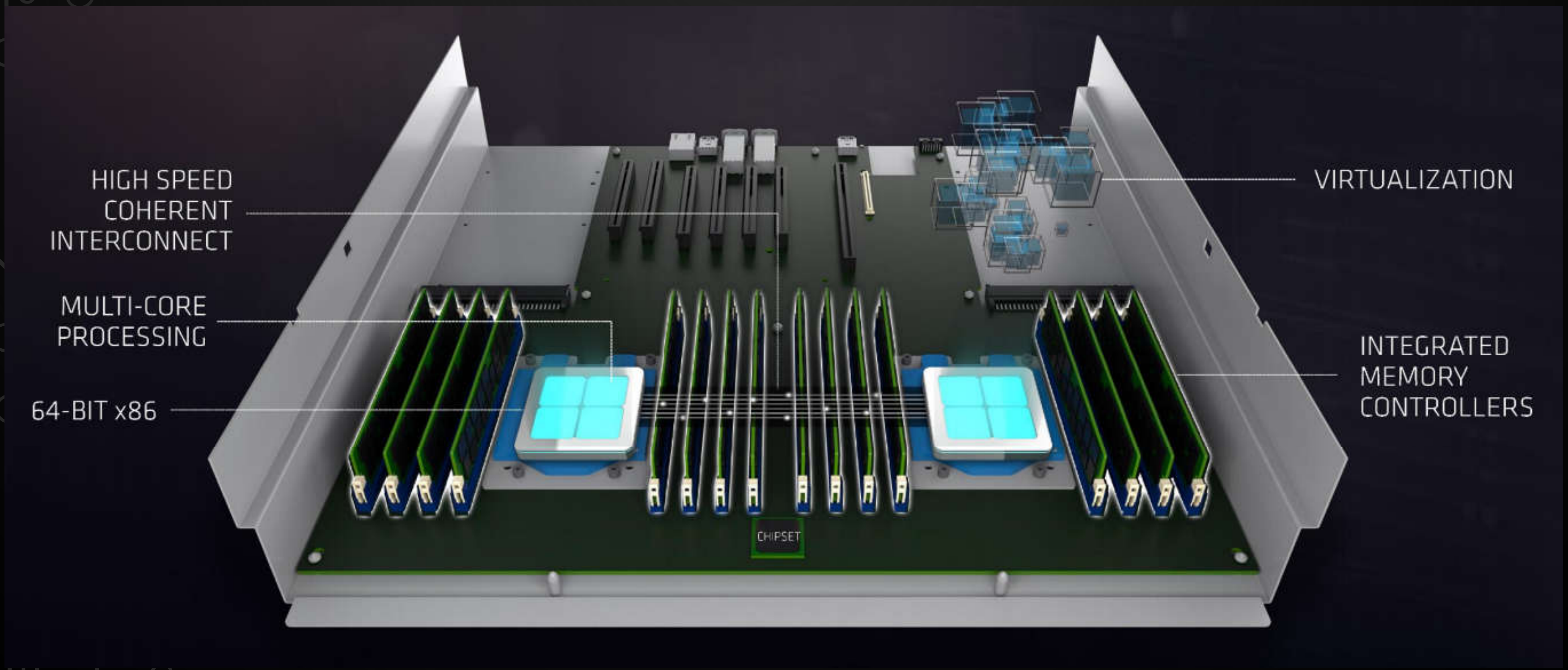
AMDFlaws

A Technical Deep Dive

Bio

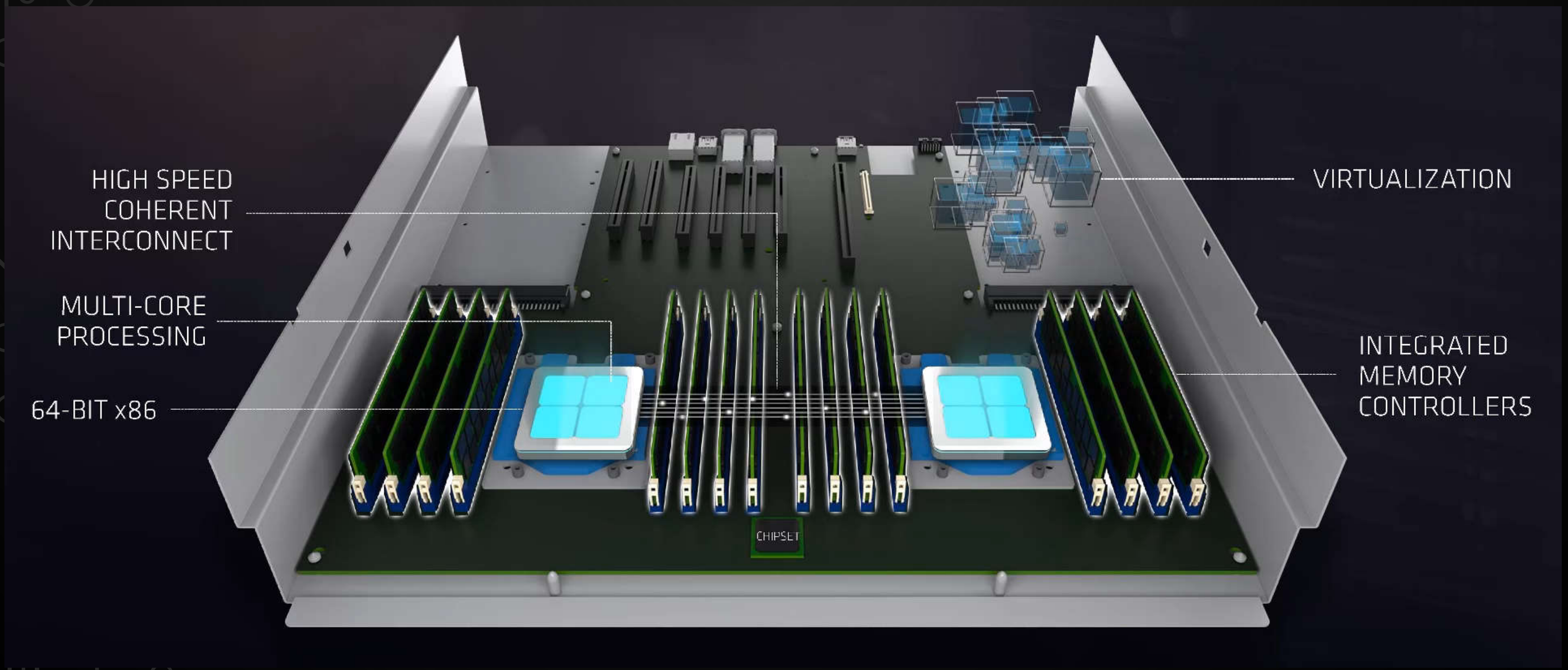
- Ido Li On
  - CEO @ CTS Labs (2017-2018)
- Uri Farkas
  - VP R&D @ CTS Labs (2017-2018)

Research Overview: 13 Vulnerabilities

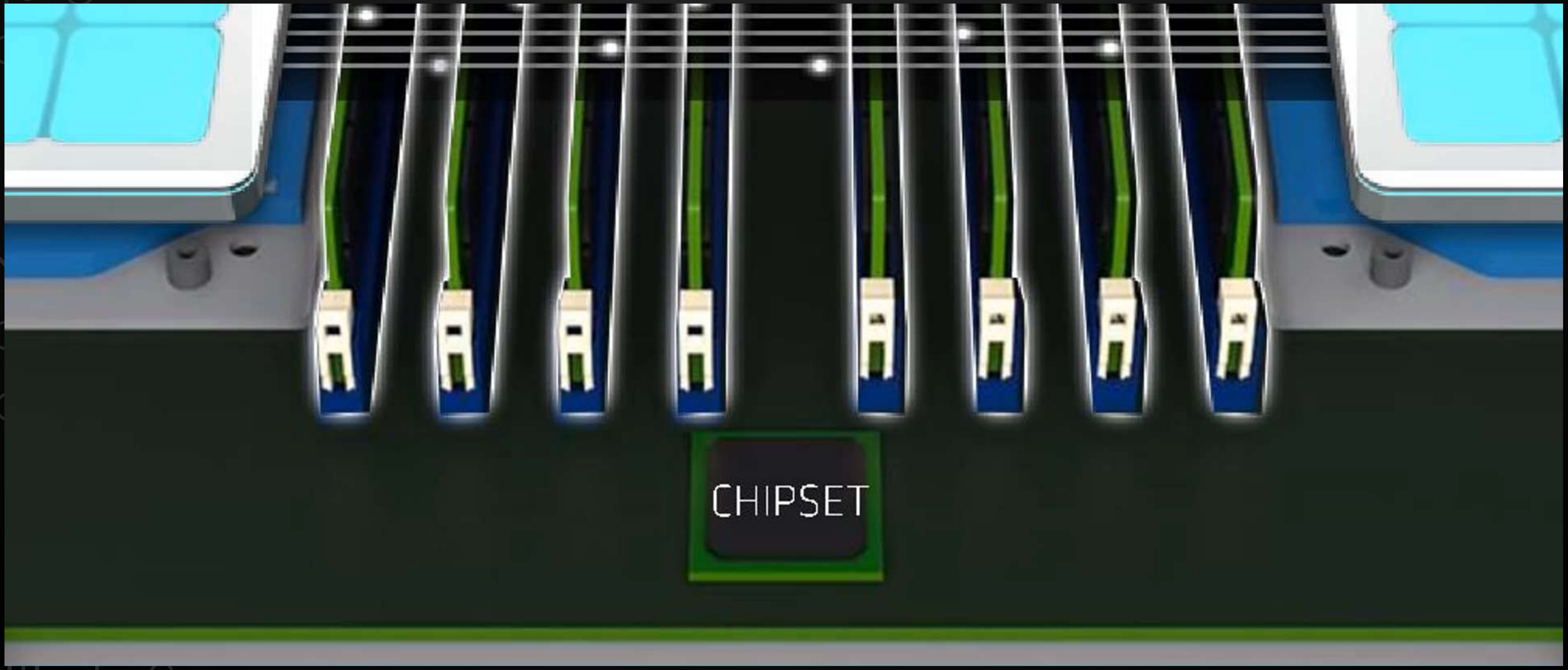


Chipset Vulnerabilities

The chipset contains USB, SATA, and PCIe bridge controllers



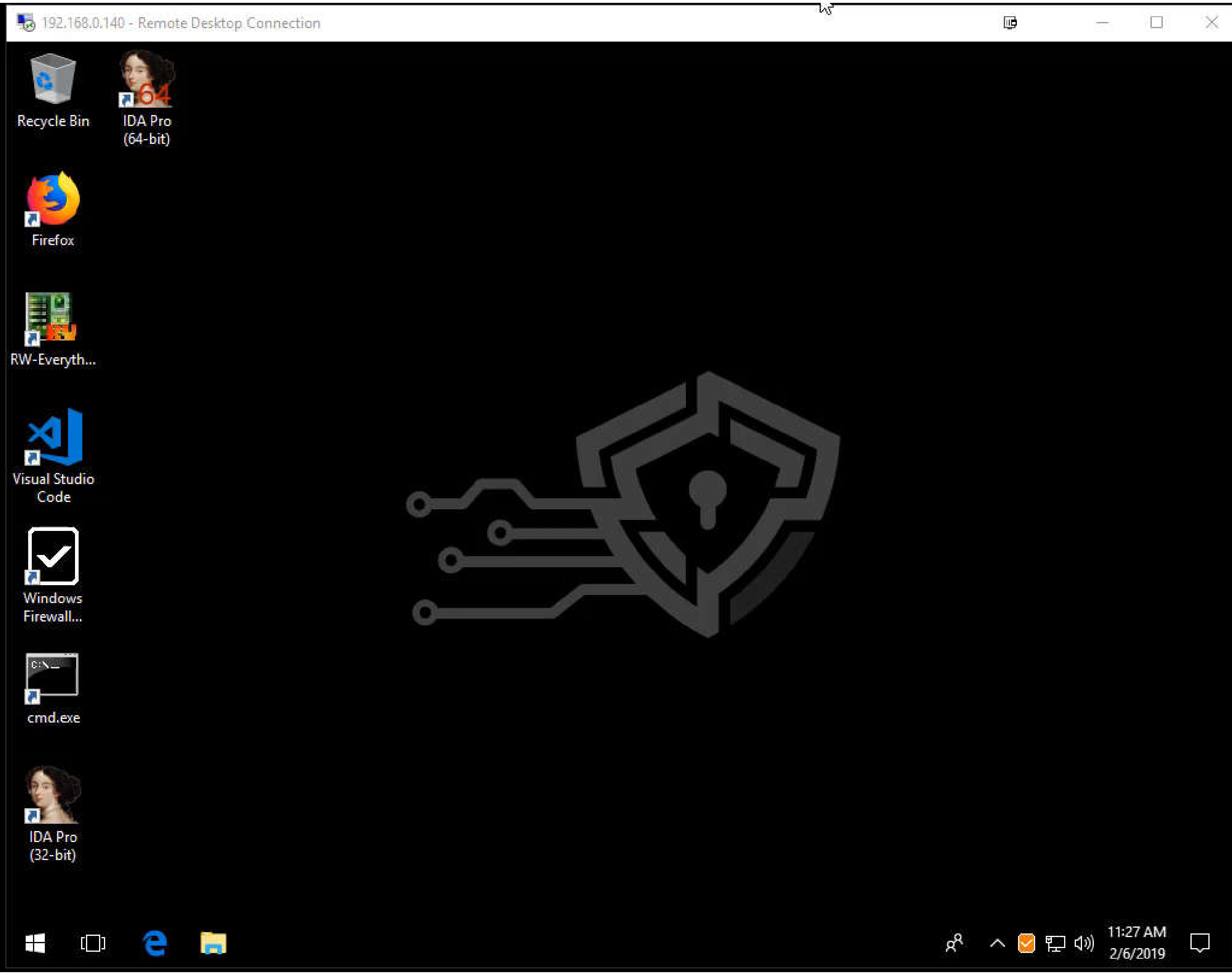
Platform Security Processor Vulnerabilities



Demo: Ryzen Desktop machine

Ryzen 5

Biostar B350GT3

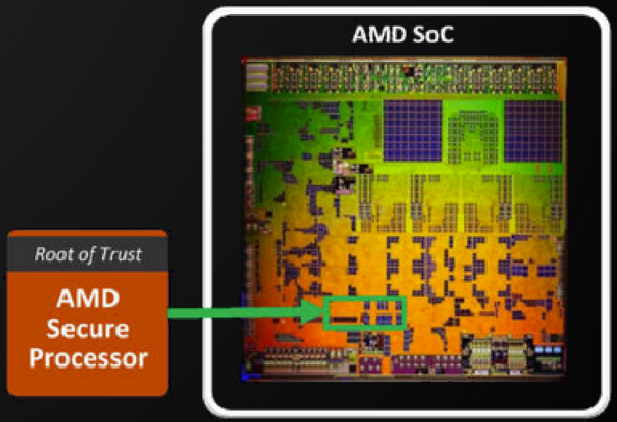

192.168.0.140 - Remote Desktop Connection
Recycle Bin
IDA Pro (64-bit)
Firefox
RW-Everyth...
Visual Studio Code
Windows Firewall...
cmd.exe
IDA Pro (32-bit)
11:27 AM
2/6/2019

What we're here to talk about

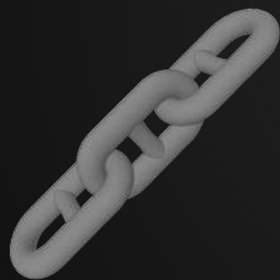
- The Platform Security Processor (PSP) and why care about it
- How we researched the PSP
- Three of the most interesting vulnerabilities we found

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)



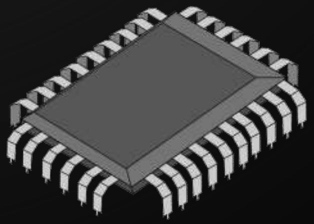
The PSP has powerful capabilities



Hardware Root of Trust

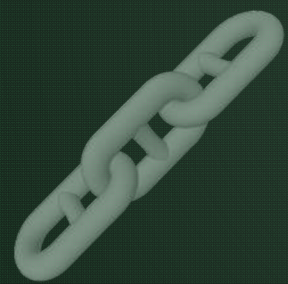
Direct Memory Access

TPM



Completely Independent

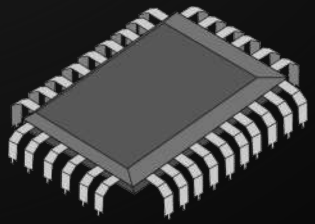
The PSP has powerful capabilities



Hardware Root of Trust

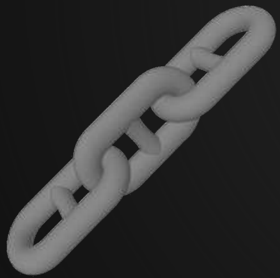
Direct Memory Access

TPM



Completely Independent

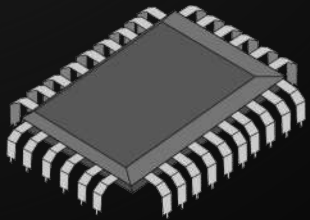
The PSP has powerful capabilities



Hardware Root of Trust

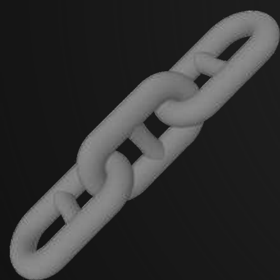
Direct Memory Access

TPM

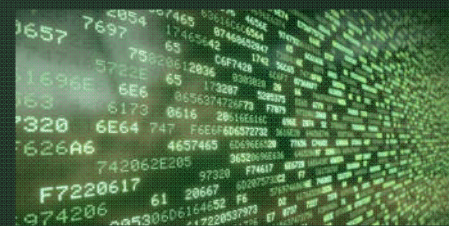


Completely Independent

The PSP has powerful capabilities

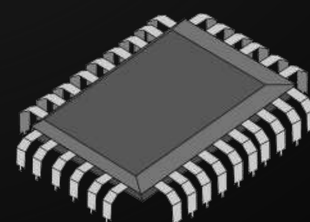


Hardware Root of Trust



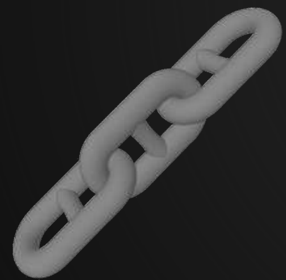
Direct Memory Access

TPM



Completely Independent

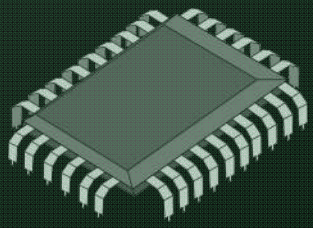
The PSP has powerful capabilities



Hardware Root of Trust

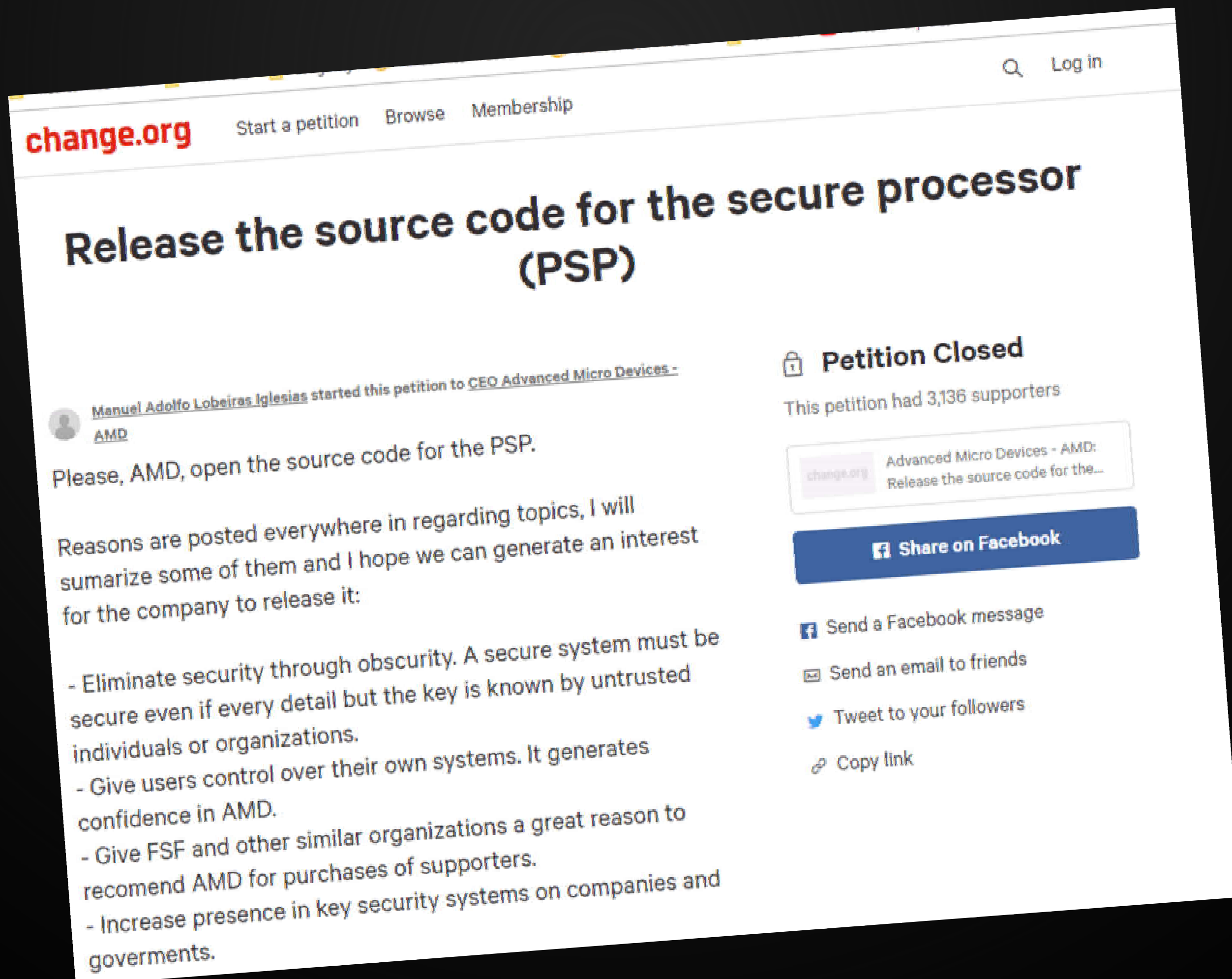
Direct Memory Access

TPM



Completely Independent

The PSP is controversial



The PSP is everywhere

Ryzen PCs

Ryzen Mobile Laptops

EPYC Servers

Vega GPUs

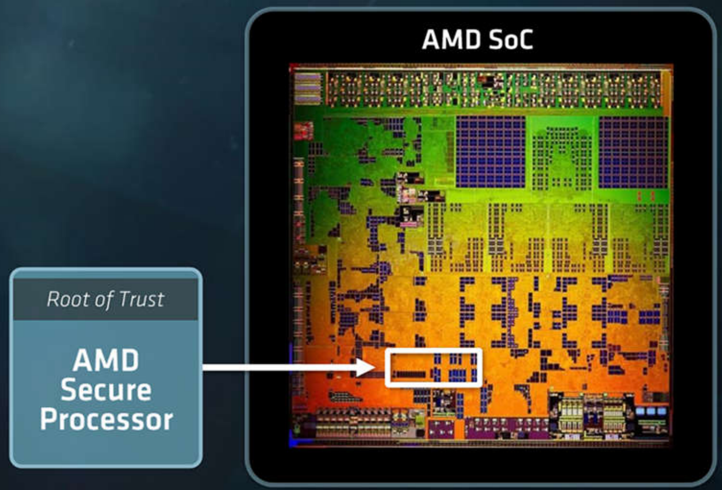
Pwning the PSP

The PSP Firmware resides in SPI ROM (BIOS Image)

AMD SECURE PROCESSOR

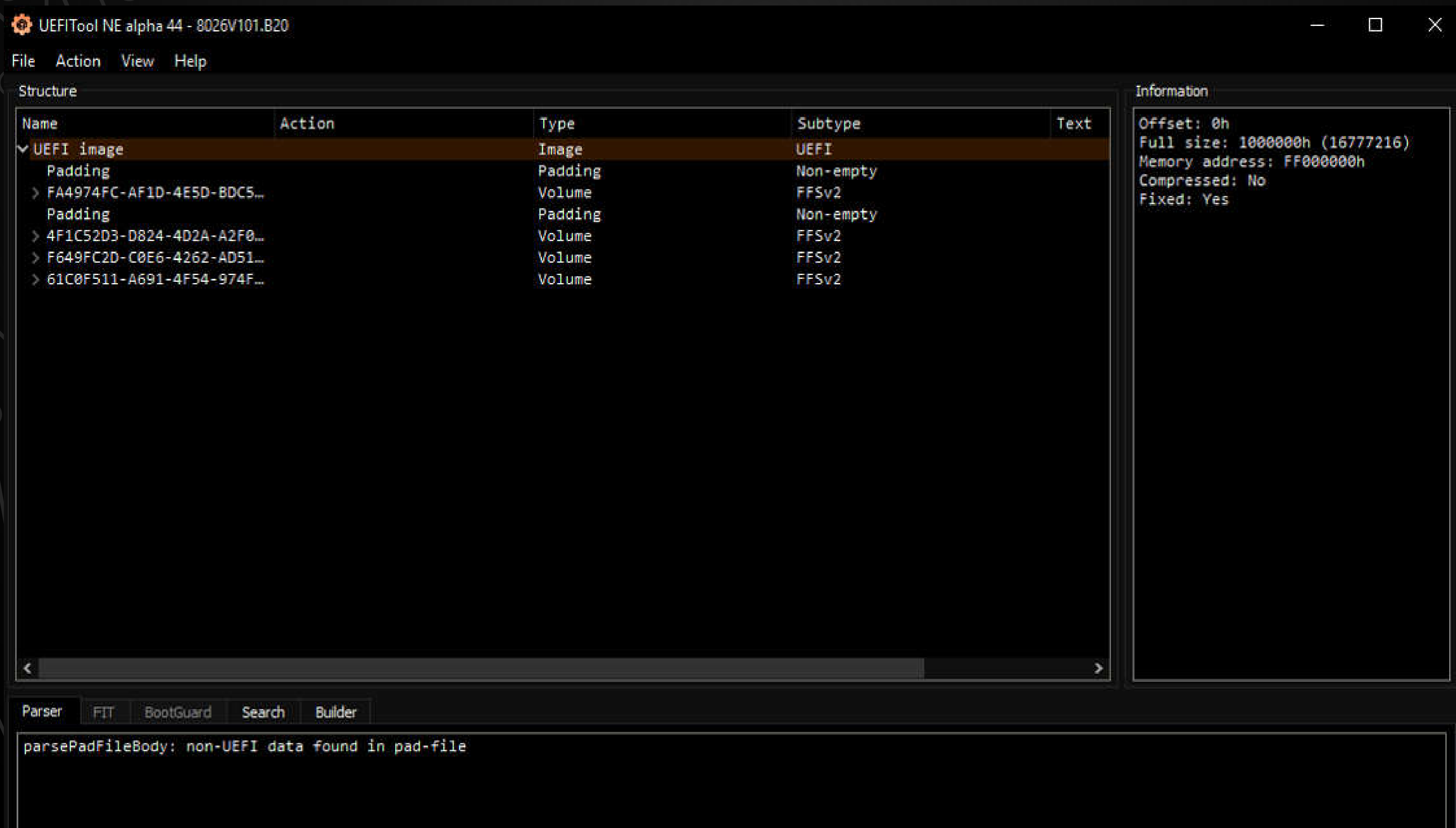
A Dedicated Security Subsystem

- 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

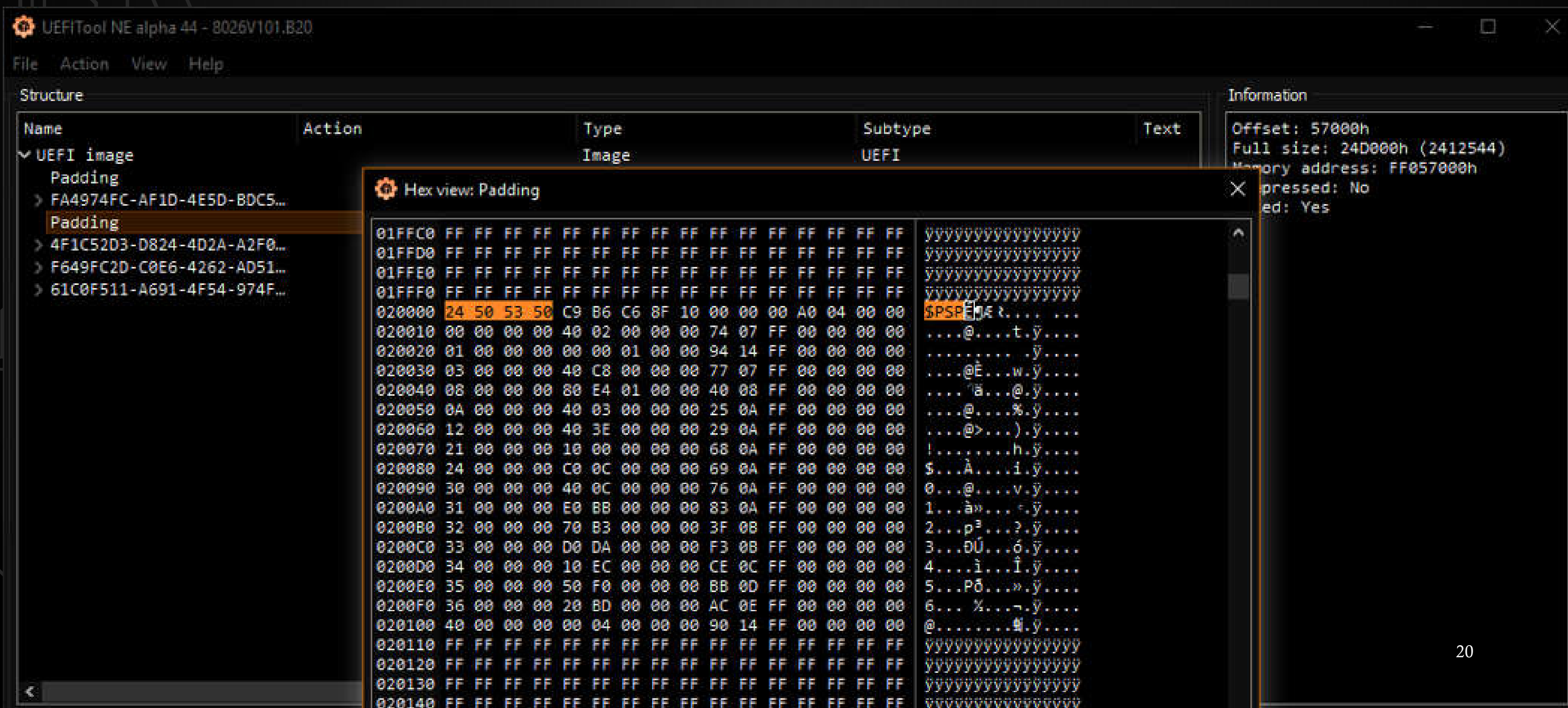


Hardware Root of Trust Provides Foundation for Platform Security

UEFITool lets us examine BIOS images



The PSP firmware resides in a “padding” area

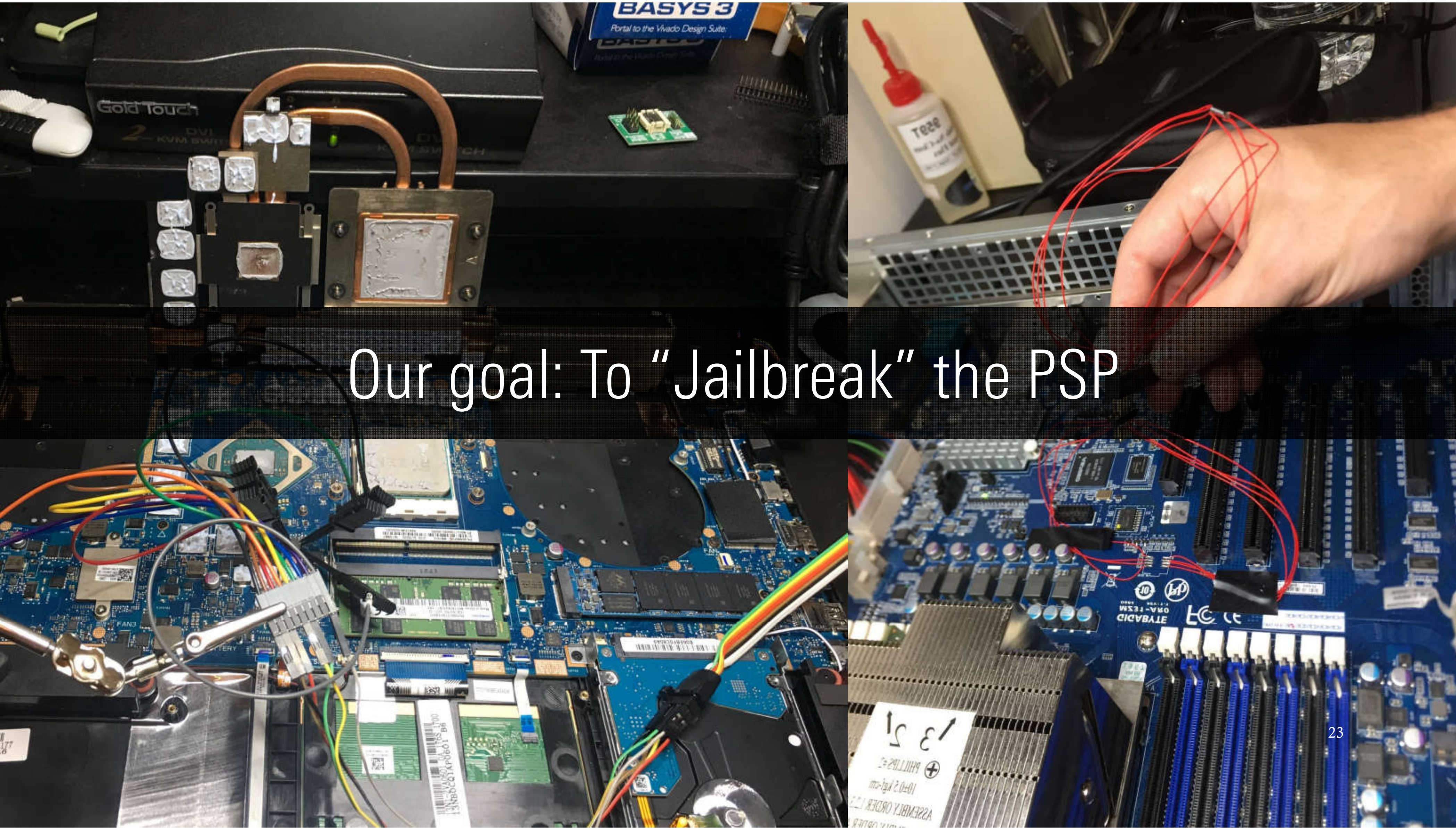


ARM code inside

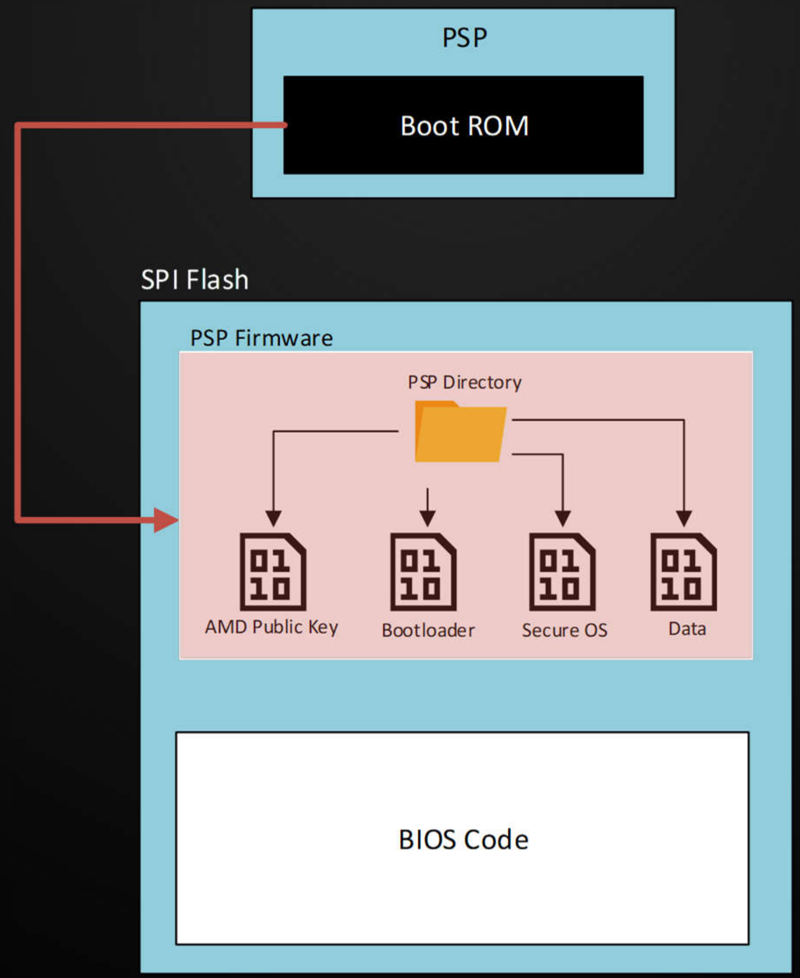
```
uri@Uri:~$ binwalk -A padding.bin

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+24A↑o
VOLATILE:00000FB9                 DCB 0, 0, 0
VOLATILE:00000FBC off_FBC         DCD dword_A2C4           ; DATA XREF: f_bootloader_routine:loc_E24↑r
VOLATILE:00000FC0 dword_FC0       DCD 0x5A870              ; DATA XREF: f_bootloader_routine+2C8↑r
VOLATILE:00000FC4 dword_FC4       DCD 0x18002FD0           ; DATA XREF: f_bootloader_routine:loc_E60↑r
VOLATILE:00000FC8
VOLATILE:00000FC8 ; =============== S U B R O U T I N E =======================================
VOLATILE:00000FC8
VOLATILE:00000FC8
VOLATILE:00000FC8 sub_FC8                                  ; CODE XREF: sub_2DD4+60↓p
VOLATILE:00000FC8                                          ; sub_333C+24↓p ...
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
VOLATILE:00000FDE loc_FDE                                  ; CODE XREF: sub_FC8+12↑j
VOLATILE:00000FDE                 ORR.W           R1, R1, #0x700000
VOLATILE:00000FE2                 ORRS            R1, R4
VOLATILE:00000FE4                 STR             R1, [R0]
VOLATILE:00000FE6                 MOVW            R1, #0x240
VOLATILE:00000FEA                 STRD.W          R1, R3, [R0,#4]
VOLATILE:00000FEE                 MOV.W           R1, #0x20000
VOLATILE:00000FF2                 STRD.W          R1, R2, [R0,#0xC]
```

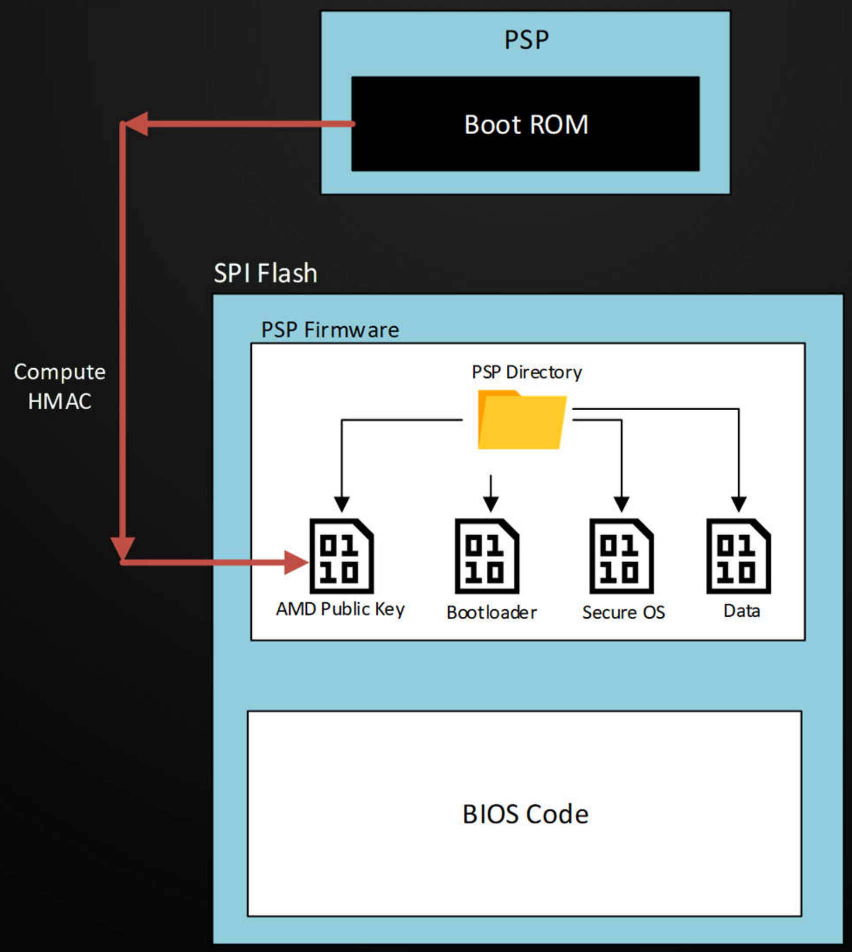
Our goal: To "Jailbreak" the PSP



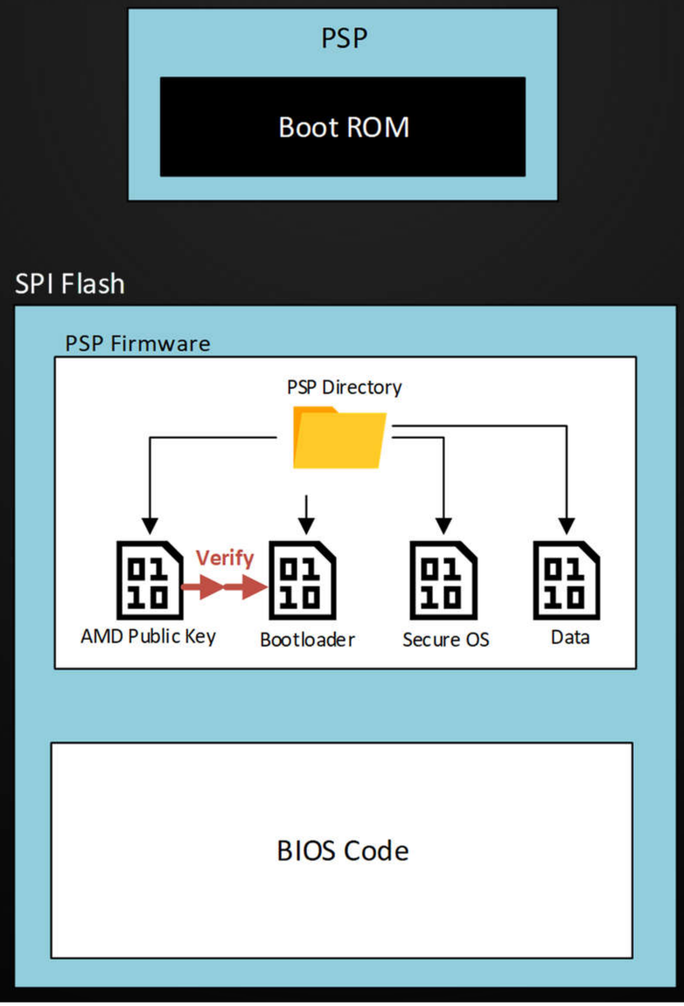
The Boot Process is a chain of verification



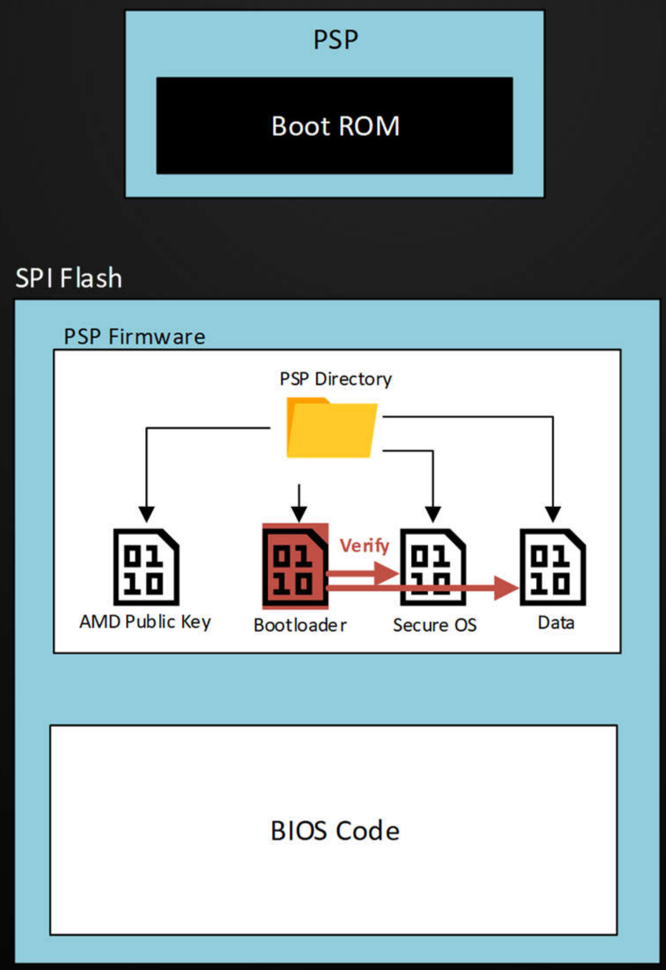
The Boot Process is a chain of verification



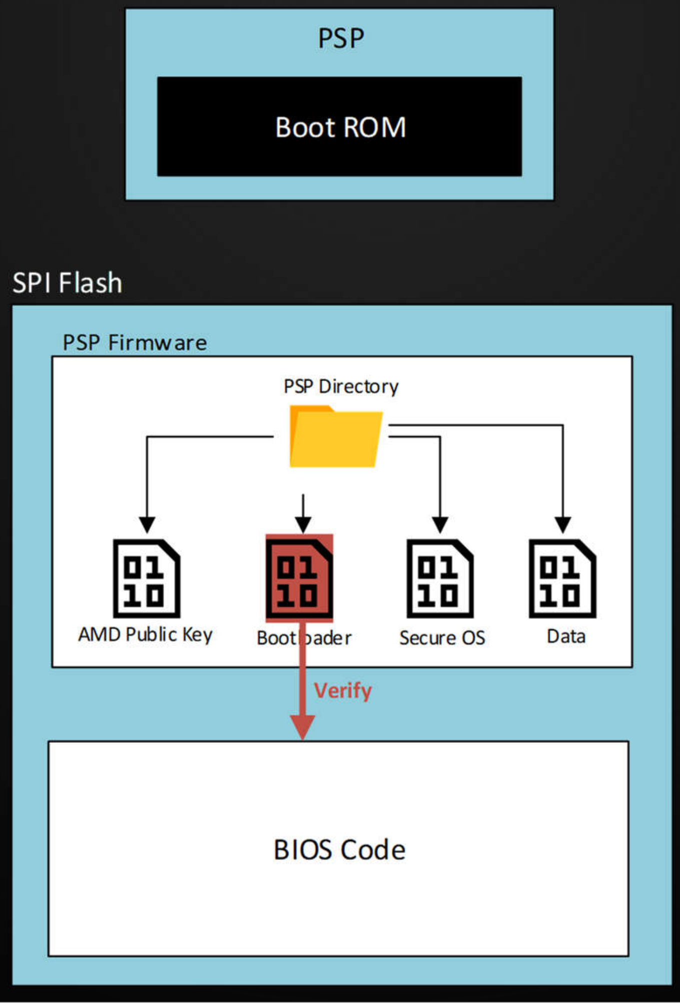
The Boot Process is a chain of verification



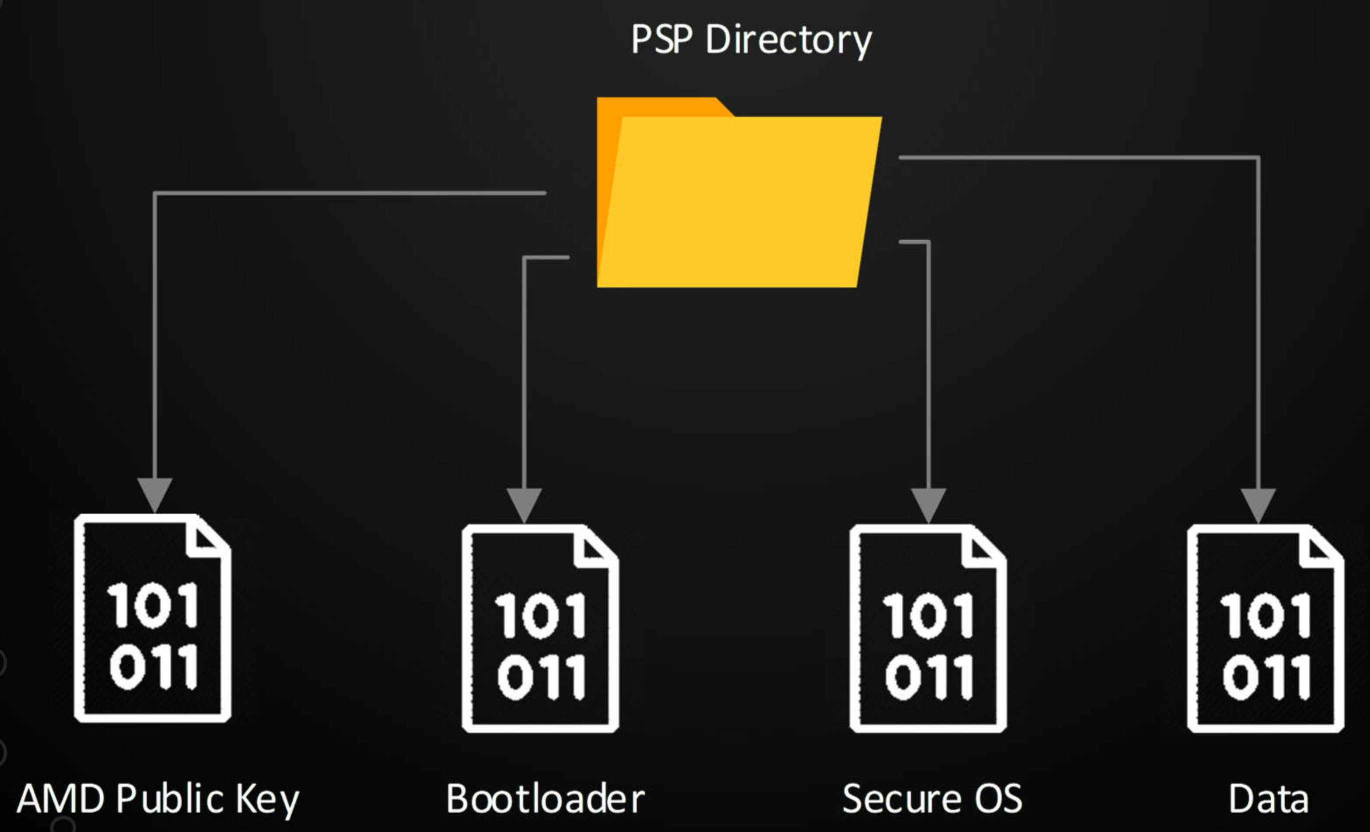
The Boot Process is a chain of verification



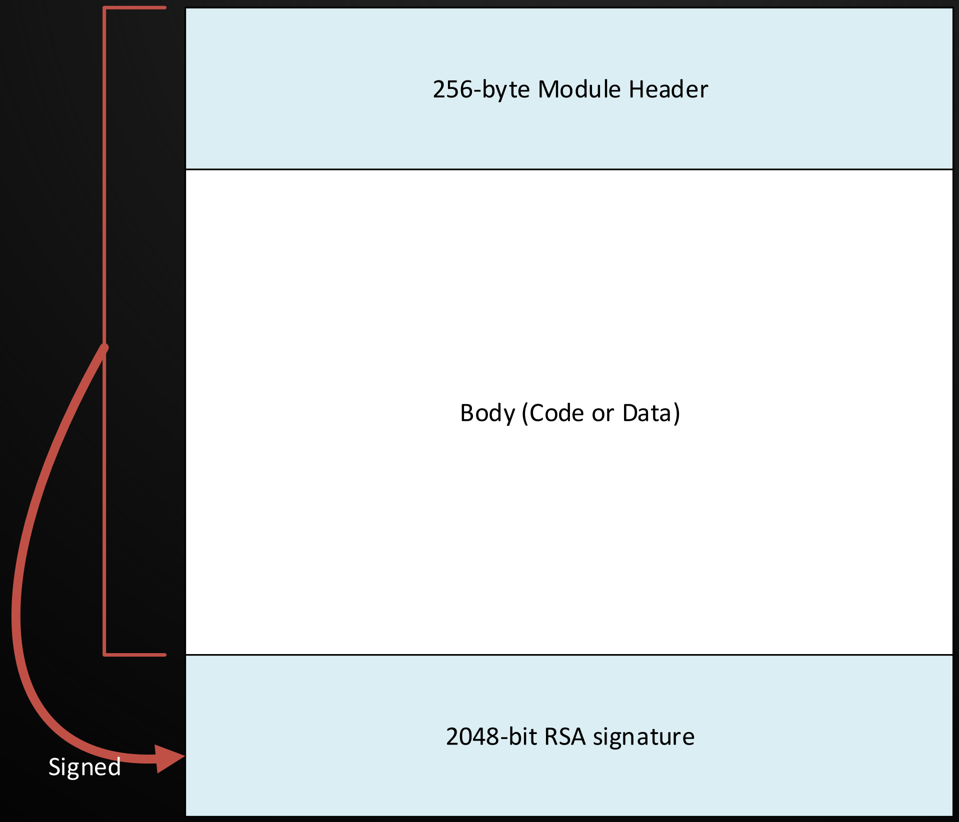
The Boot Process is a chain of verification



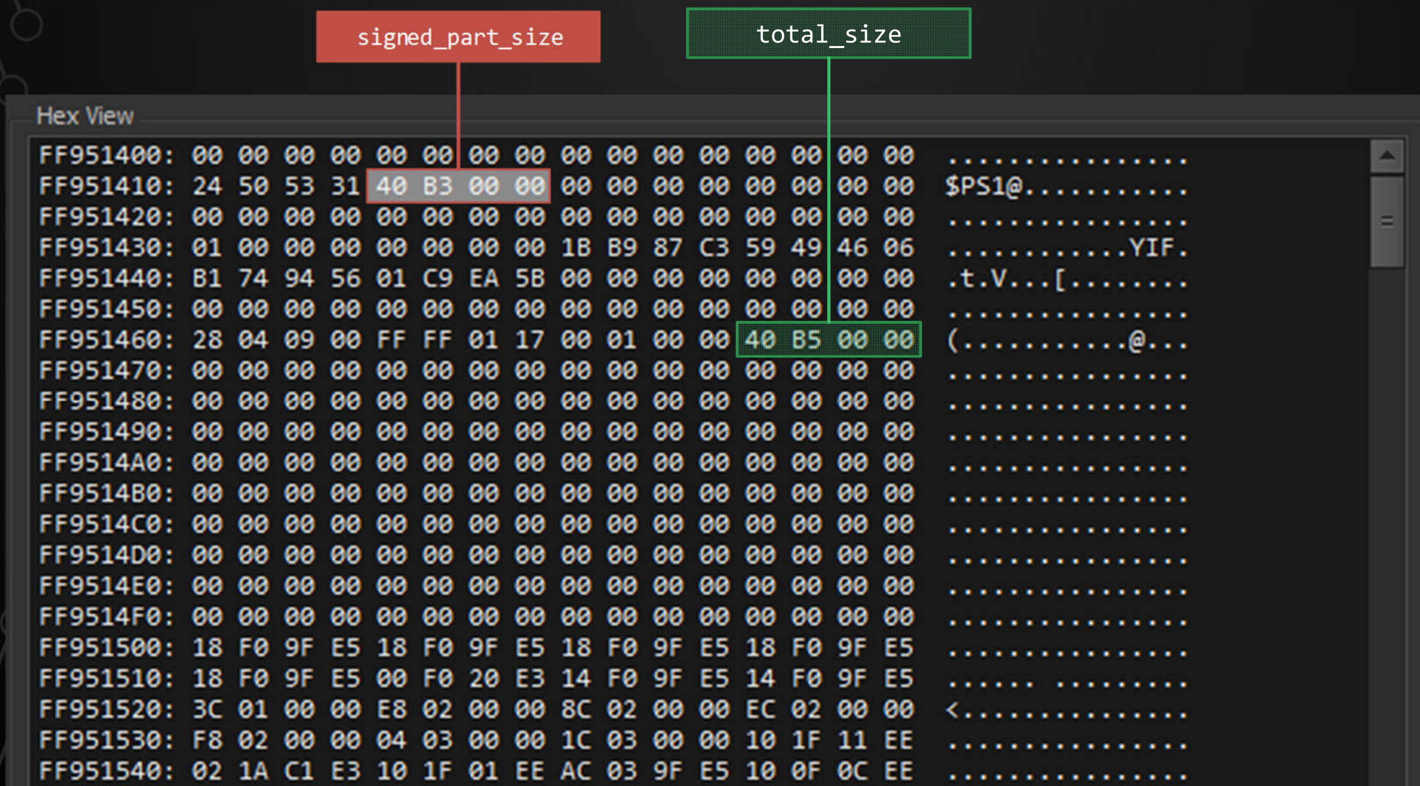
PSP firmware is comprised of individually signed modules



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

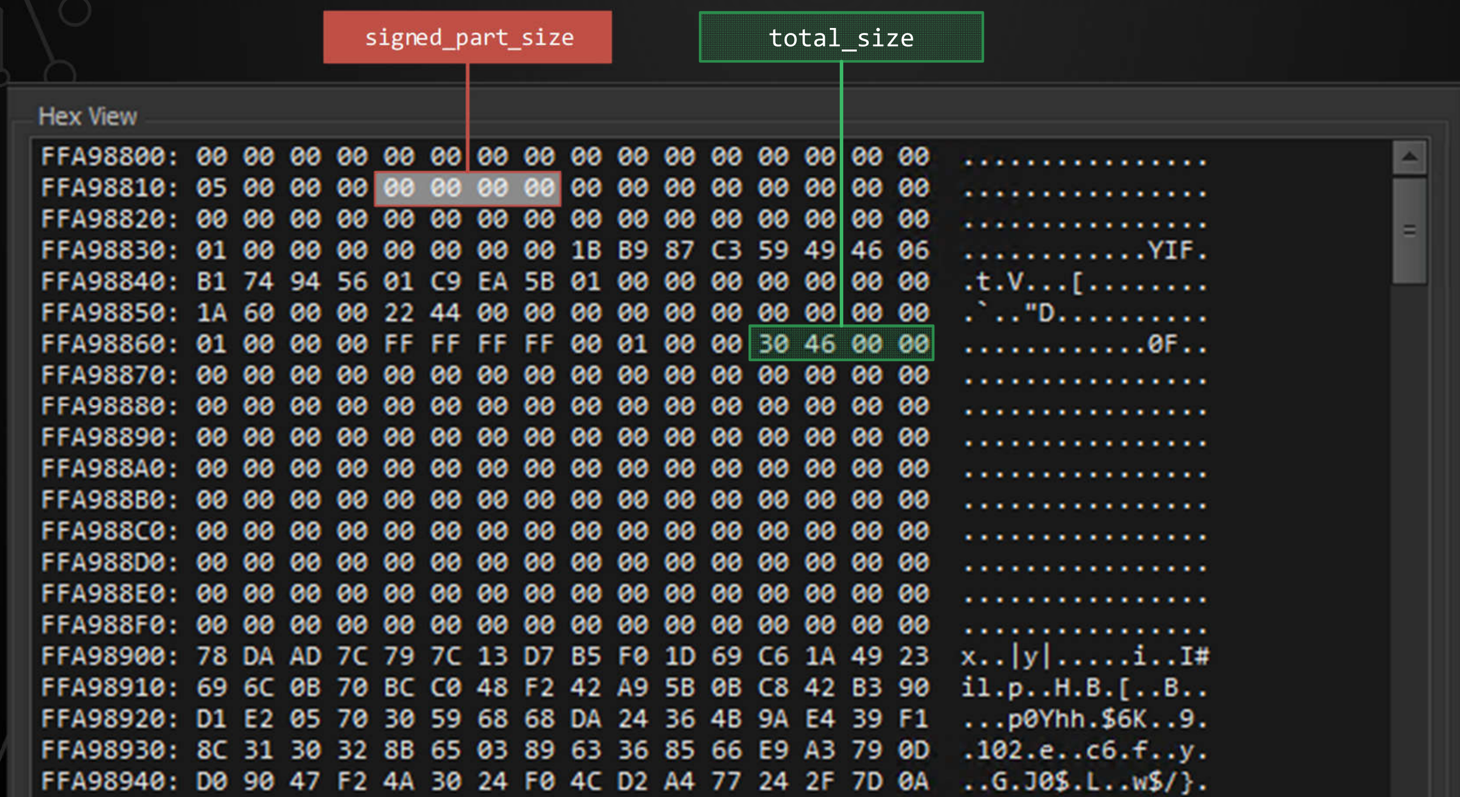


PSP Module (Code)

Some modules have signed_part_size = 0

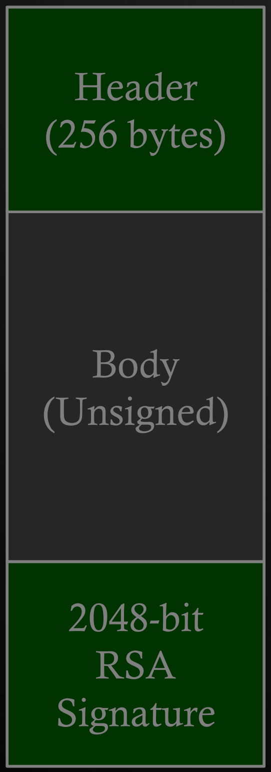
```
uri@Uri:~$ python find_issues.py

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

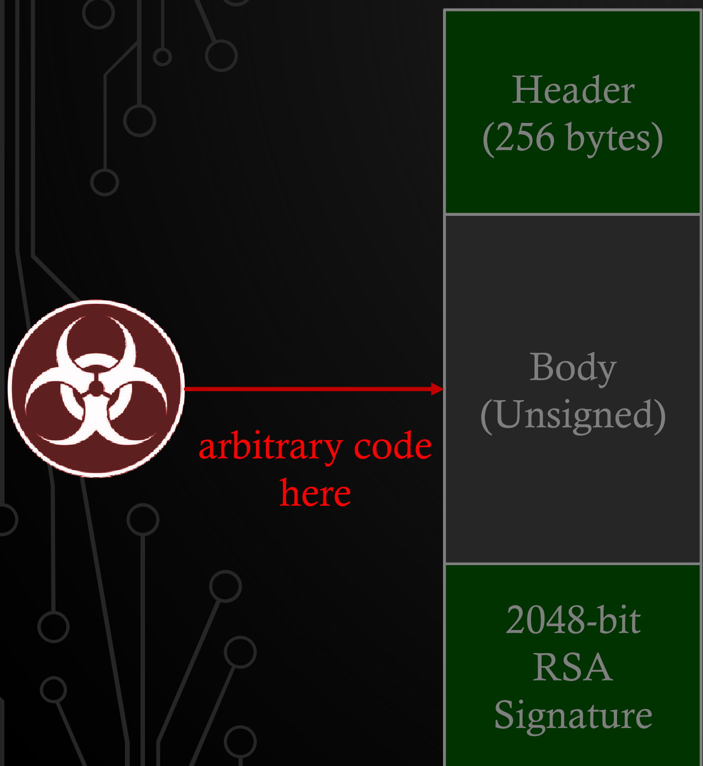


PSP Module (Data)

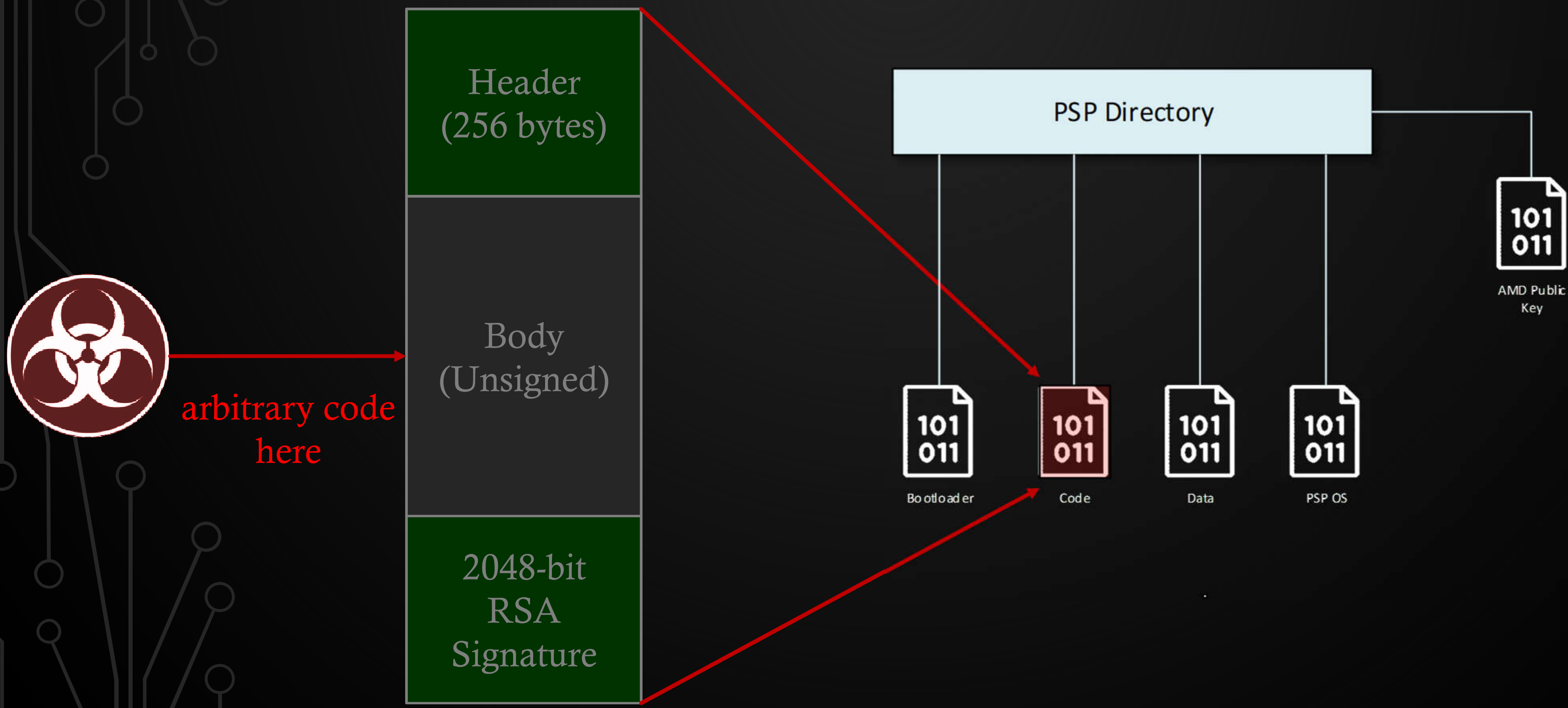
In data modules – the signature only covers the header!

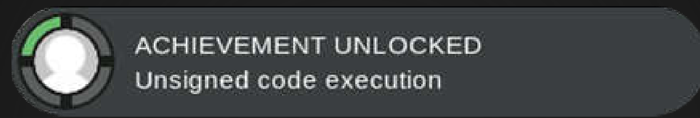


We can use the header and its signature to create any legitimate PSP module



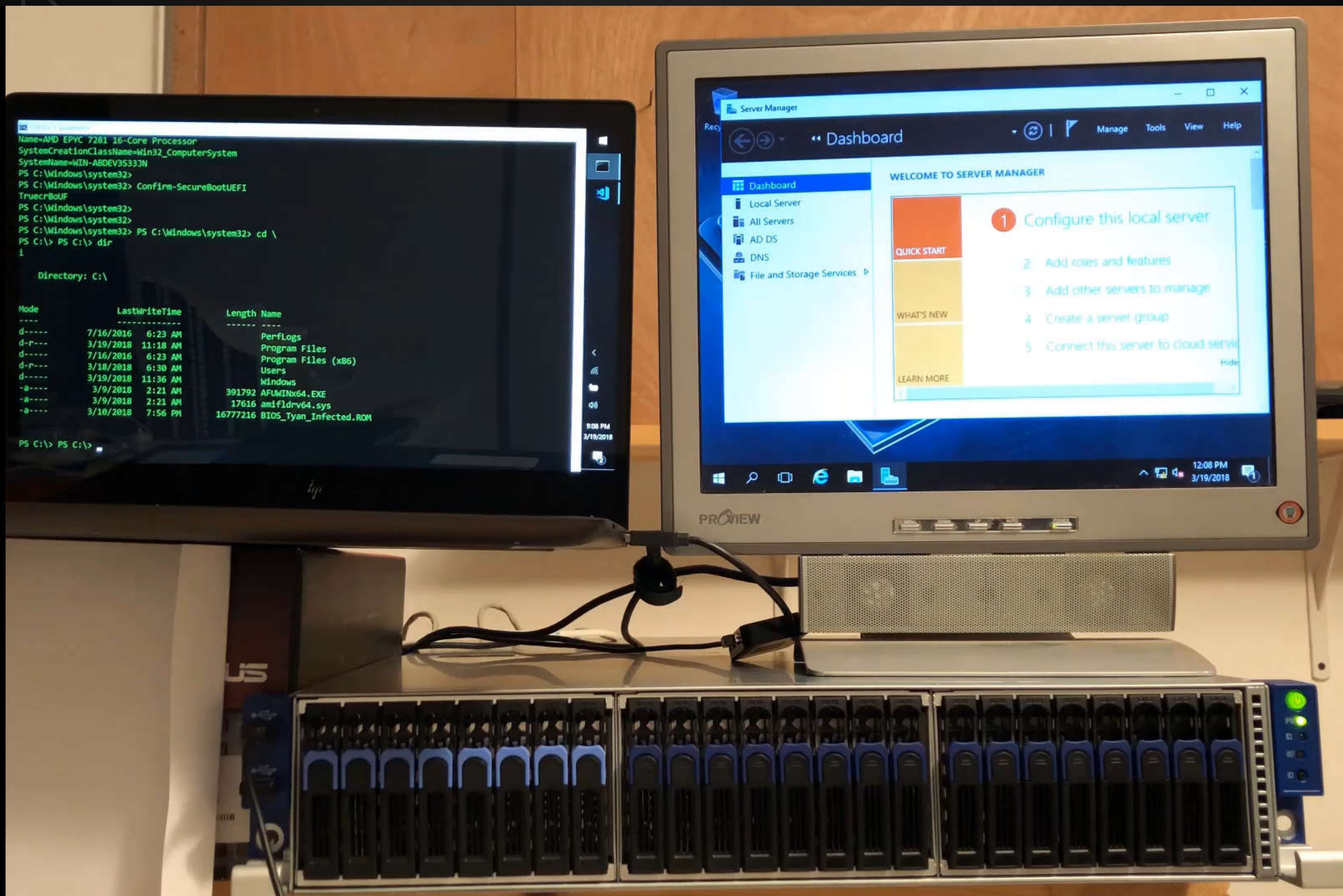
We can replace an existing module, and pass all signature checks!



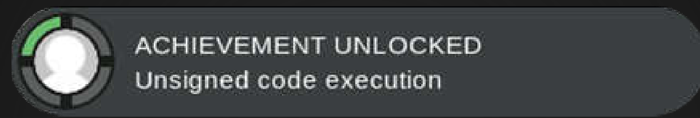


Masterkey-1

DEMO



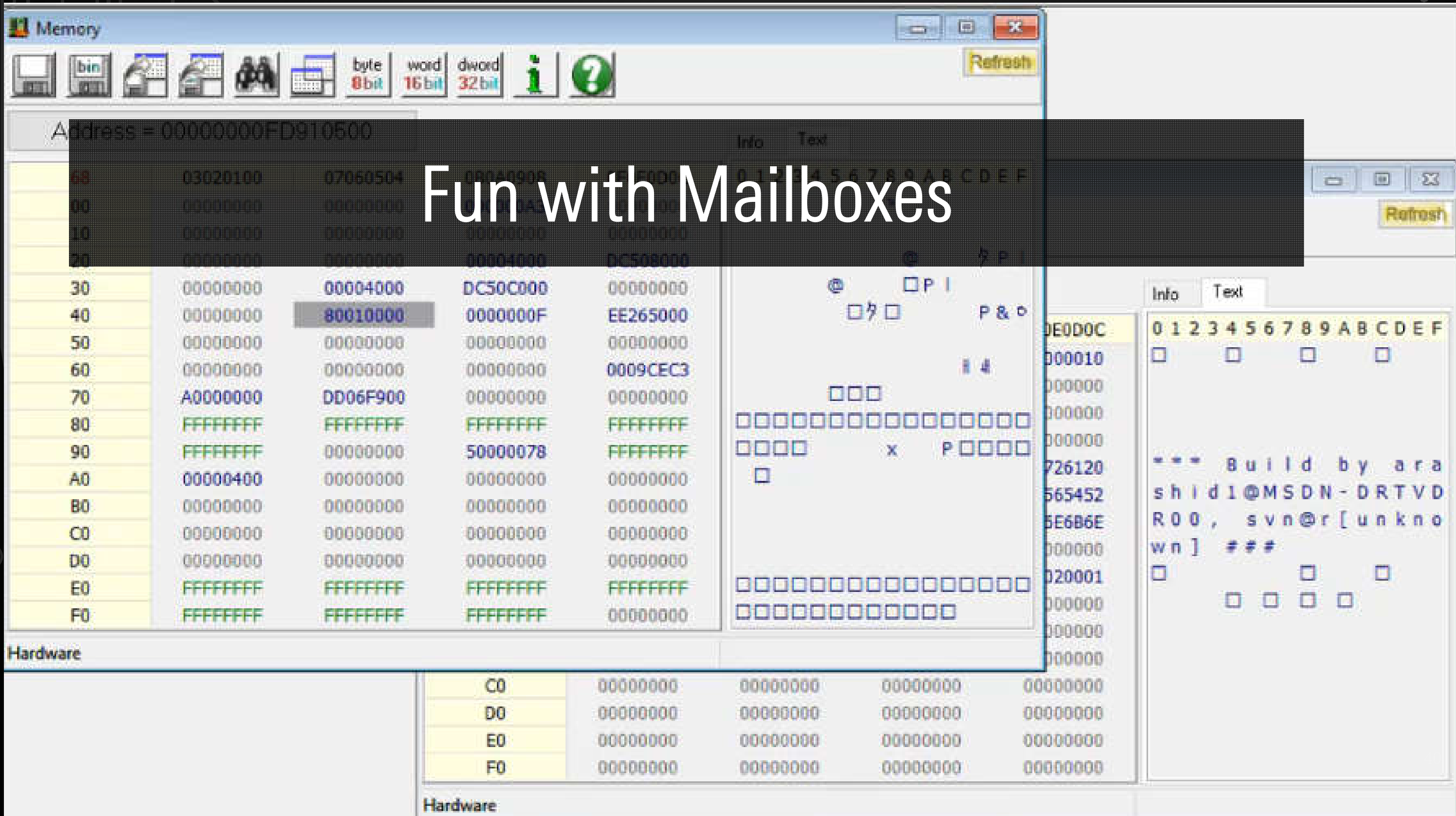
Name=AMD EPYC 7281 16-Core Processor
SystemCreationClassName=Win32_ComputerSystem
SystemName=WIN-ABDEV3S33JN
PS C:\Windows\system32>
PS C:\Windows\system32> Confirm-SecureBootUEFI
TruecrBoUF
PS C:\Windows\system32>
PS C:\Windows\system32>
PS C:\Windows\system32> PS C:\Windows\system32> cd \
PS C:\> PS C:\> dir
Directory: C:\
Mode LastWriteTime Length Name
d----- 7/16/2016 6:23 AM PerfLogs
d-r--- 3/19/2018 11:18 AM Program Files
d----- 7/16/2016 6:23 AM Program Files (x86)
d-r--- 3/18/2018 6:30 AM Users
d----- 3/19/2018 11:36 AM Windows
-a---- 3/9/2018 2:21 AM 391792 AFUWINx64.EXE
-a---- 3/9/2018 2:21 AM 17616 amifldrv64.sys
-a---- 3/10/2018 7:56 PM 16777216 BIOS_Tyan_Infected.ROM
PS C:\> PS C:\>
Server Manager
Dashboard
Manage
Tools
View
Help
Dashboard
Local Server
All Servers
AD DS
DNS
File and Storage Services
WELCOME TO SERVER MANAGER
1 Configure this local server
QUICK START
2 Add roles and features
3 Add other servers to manage
WHAT'S NEW
4 Create a server group
5 Connect this server to cloud servi
LEARN MORE
Hide
12:08 PM
3/19/2018
PROVIEW



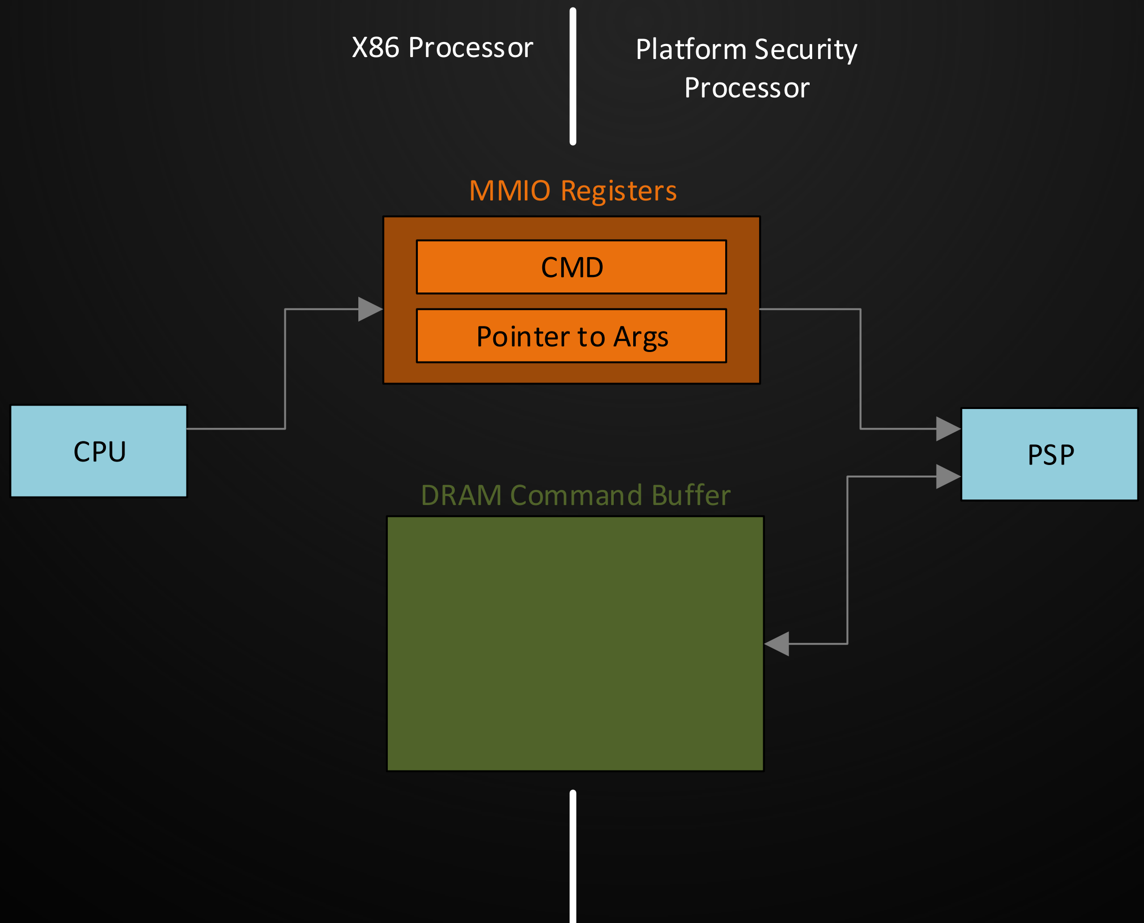
Masterkey-1

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

Fun with Mailboxes



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
```

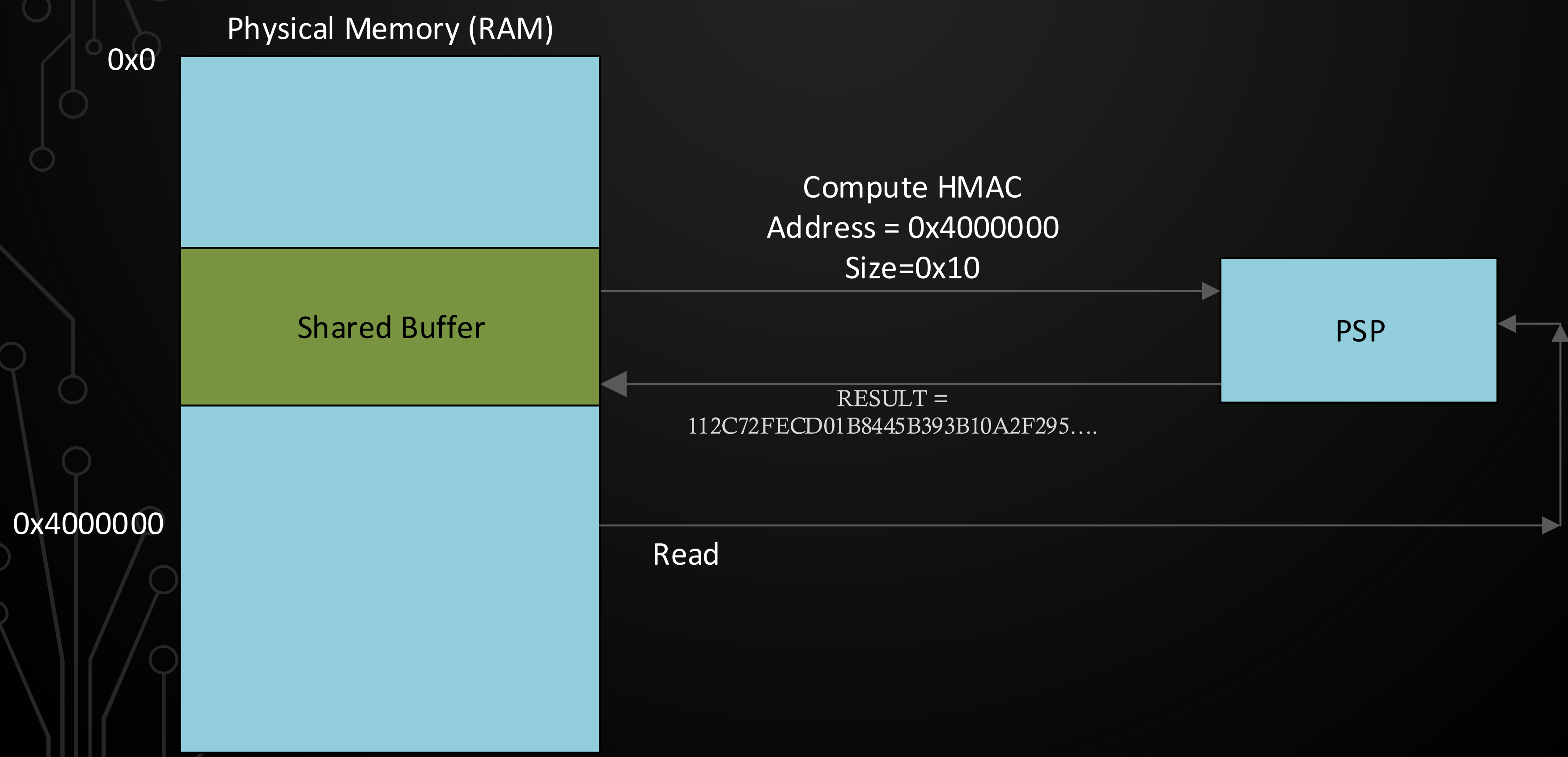

† *https://github.com/coreboot/coreboot/blob/master/src/soc/amd/common/block/include/amdblocks/psp.h*

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];
}
```

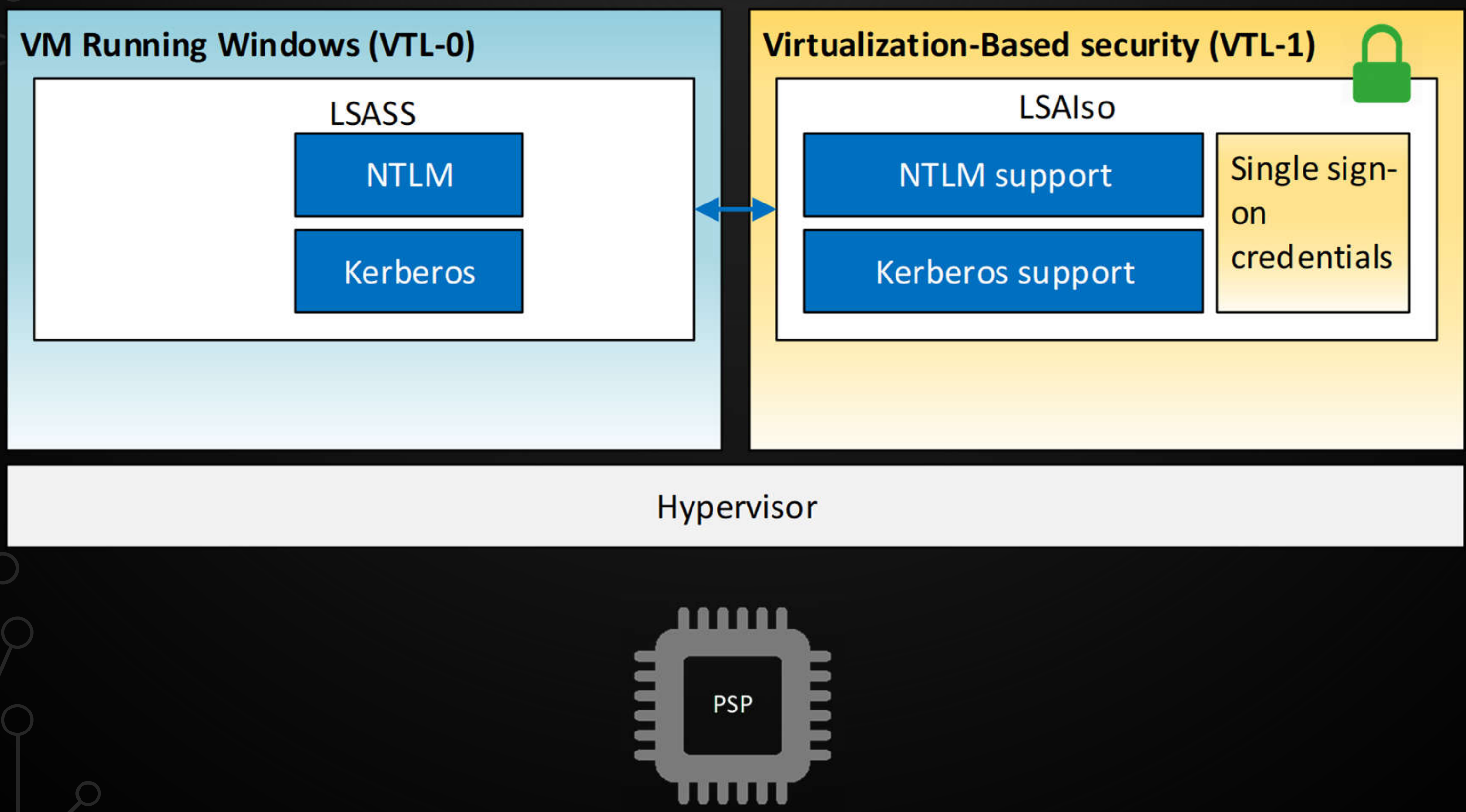
PSP reads memory for you



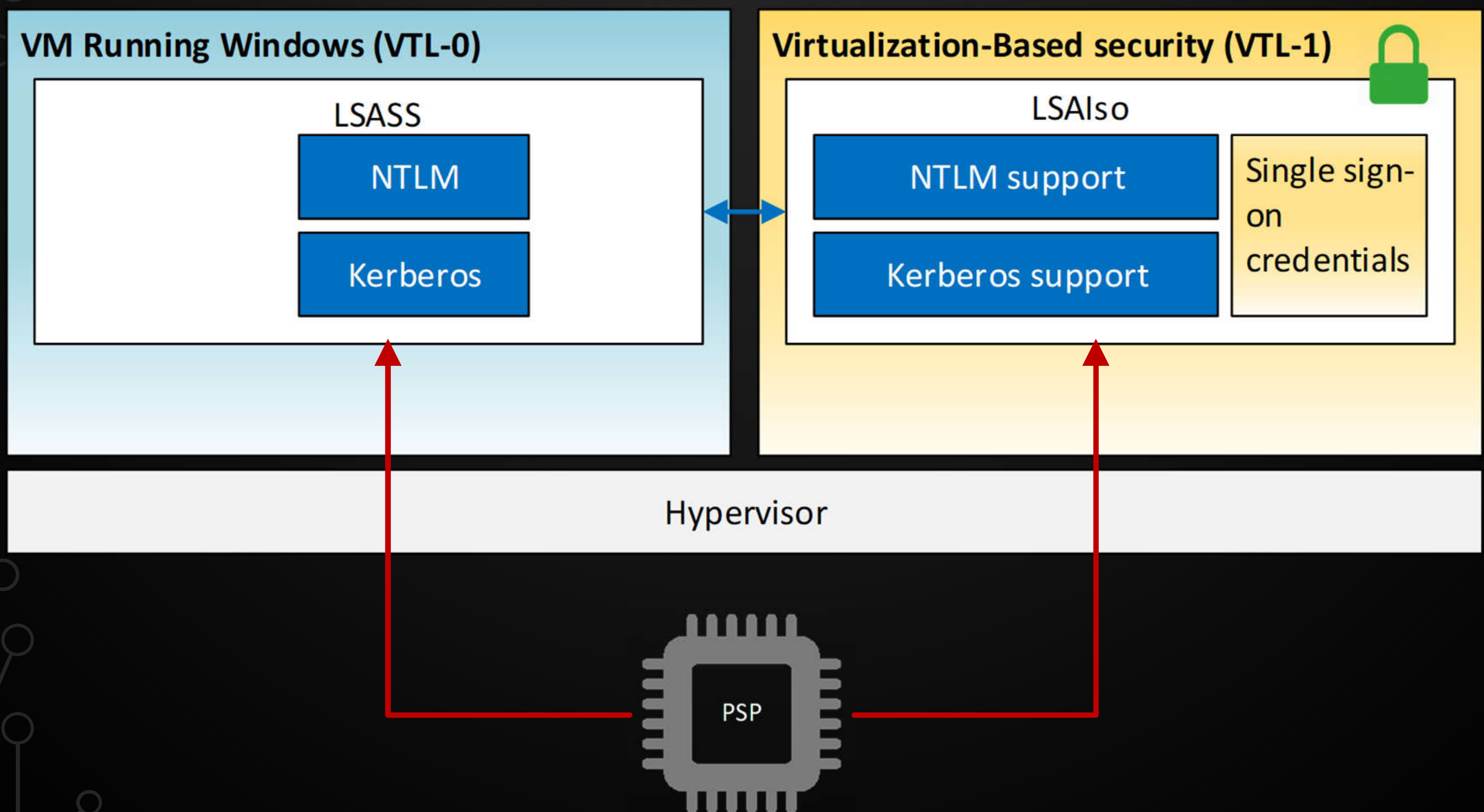
Privileged Memory Locations

- SMM
- Other virtual machines
  - Credential Guard ?

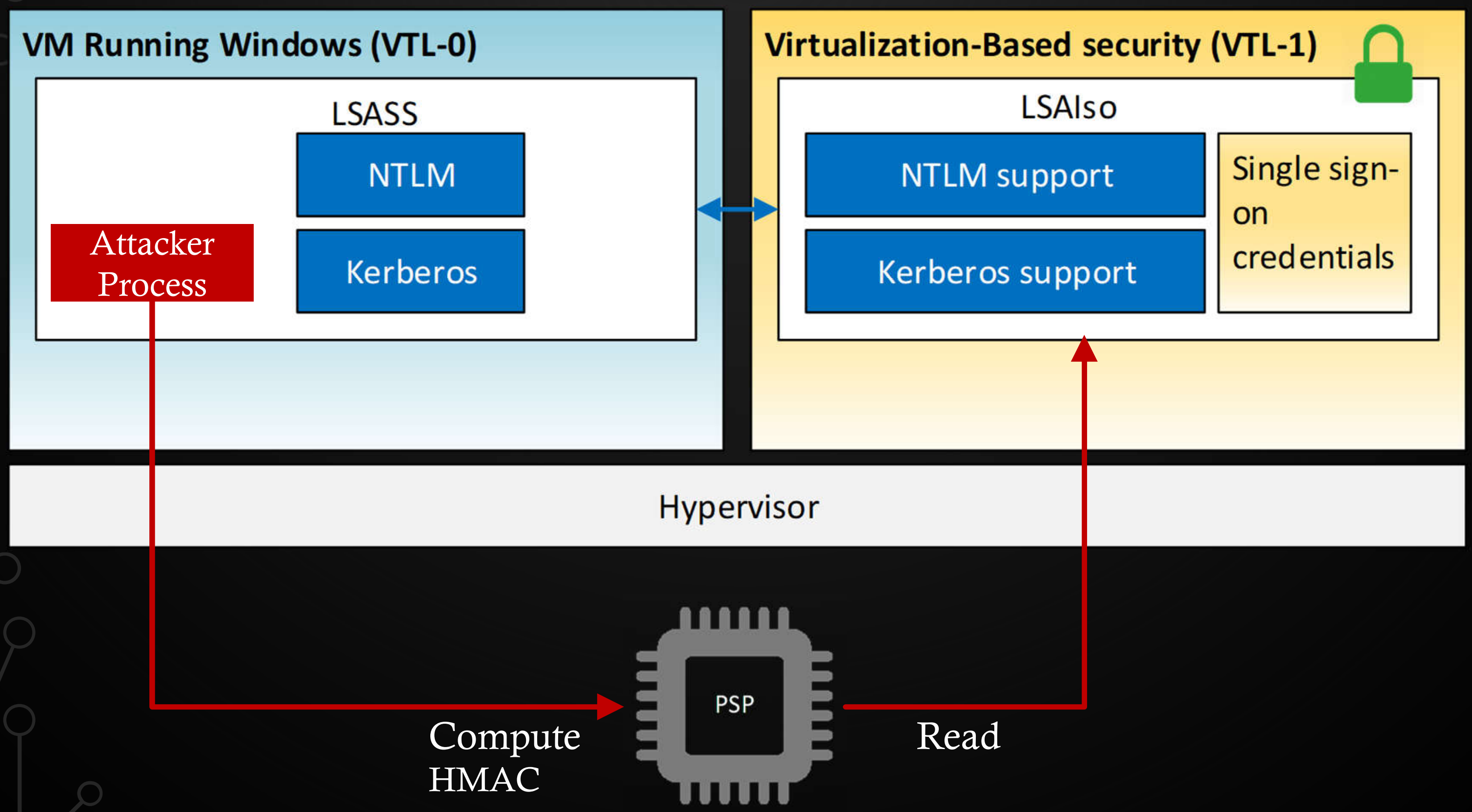
Credential Guard - Credentials are stored on isolated Virtual Machine



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



The PSP will read privileged memory for you

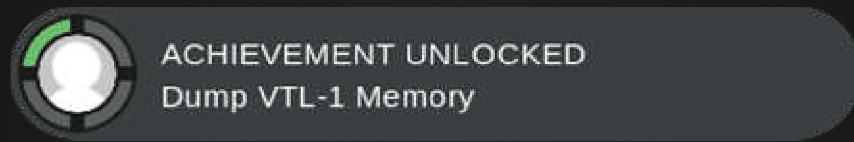


What if we compute HMAC on a single-byte?

0x00 = 743771B093E5F45526274762E2723D56B6E82273402C4D67C65D48563C658502

We can build a translation table!

0x00 = 743771B093E5F45526274762E2723D56B6E82273402C4D67C65D48563C658502
0x01 = 93F17EA603A23F3A55BCC80206EE7D8B8CEA7B5868470FFE2DF67DB17E2FA391
0x02 = 4749A3A56843611245E3C7070CE396C6225546EB03E0EC9D6F6423815E98E901
0x03 = FC5B2C165C3BC3CA503E62CE800712F02FB814B44AFE429915C4C1BE2B3B1B2F
0x04 = 112C72FECD01B8445B393B10A2F29552528386185522913BE44684DE78965679
0x05 = 80429C59A5704A361112A27FF6C72899AB2E32AA57F32411AACDC468CB180B3E
0x06 = 1BA839DE8C412DF5D5368A7209C29F30D976B0CFE3D95925FEC1156AA0C02E24
0x07 = C906BB0687AD289C4E063ACA3355A84CCDF4528CC45E970E23B779F32C0435D7
0x08 = CE52670F68F0CA109128E1F57E271BD470EB983A430D438E71BDFD0B9228EA6D
0x09 = 629DA6A43E2D9B5C18189013C486BCB16DA10F2F411338D070B112E8332946CC
0x0A = 158F90A2B67747F15F5697FC3E9E66D86CBFD9CB87839C2C0C735080C2D5BC19
0x0B = AA69F104AFF8761B45A62F3F002C58A8B0B181FFCE745A84E8AD75946DDAC7A1
0x0C = CAB9E96F2E35373EA22AC46F3E4352B6A60C5A5E0C0F4A29BC29FDC7FDCEEB3
...



Ryzenfall-1

- Dump passwords & hashes in VTL-1 memory

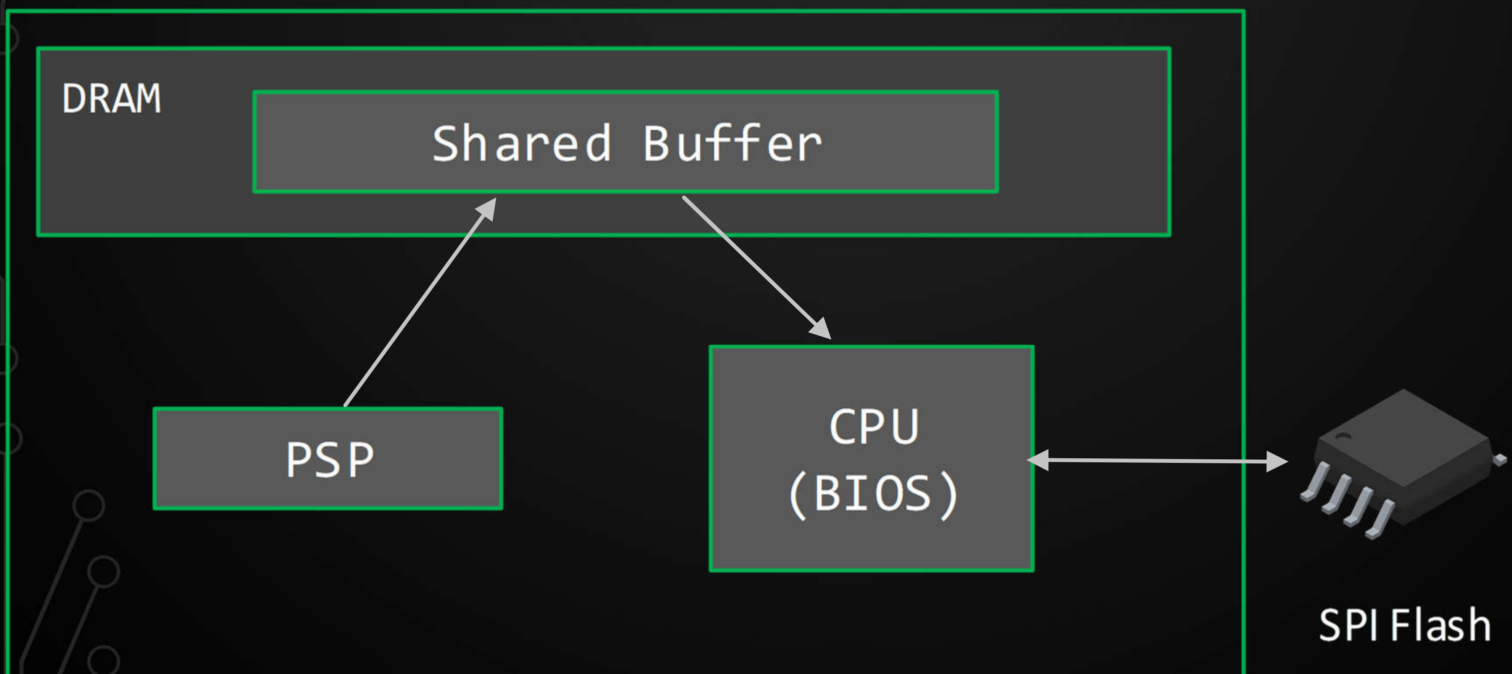
Ryzenfall-1

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

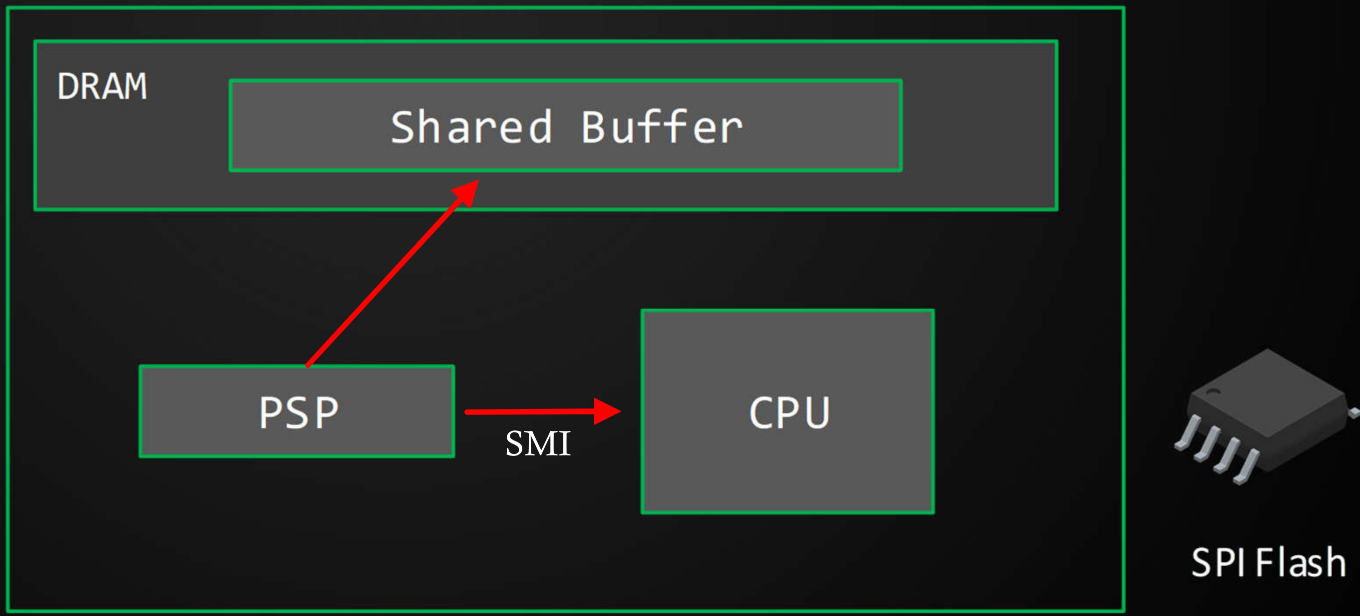
SMM_INFO – Initialize PSP-to-BIOS communications

```
/* 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



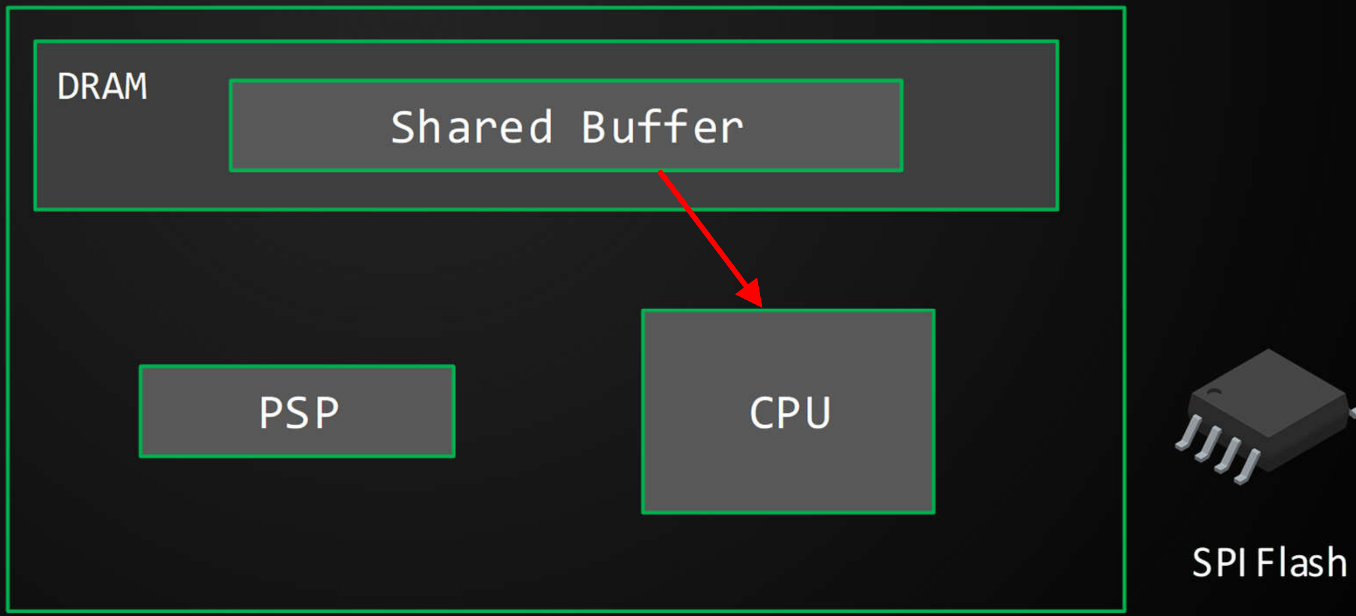
[1] The PSP first writes its request to the P2C shared buffer.



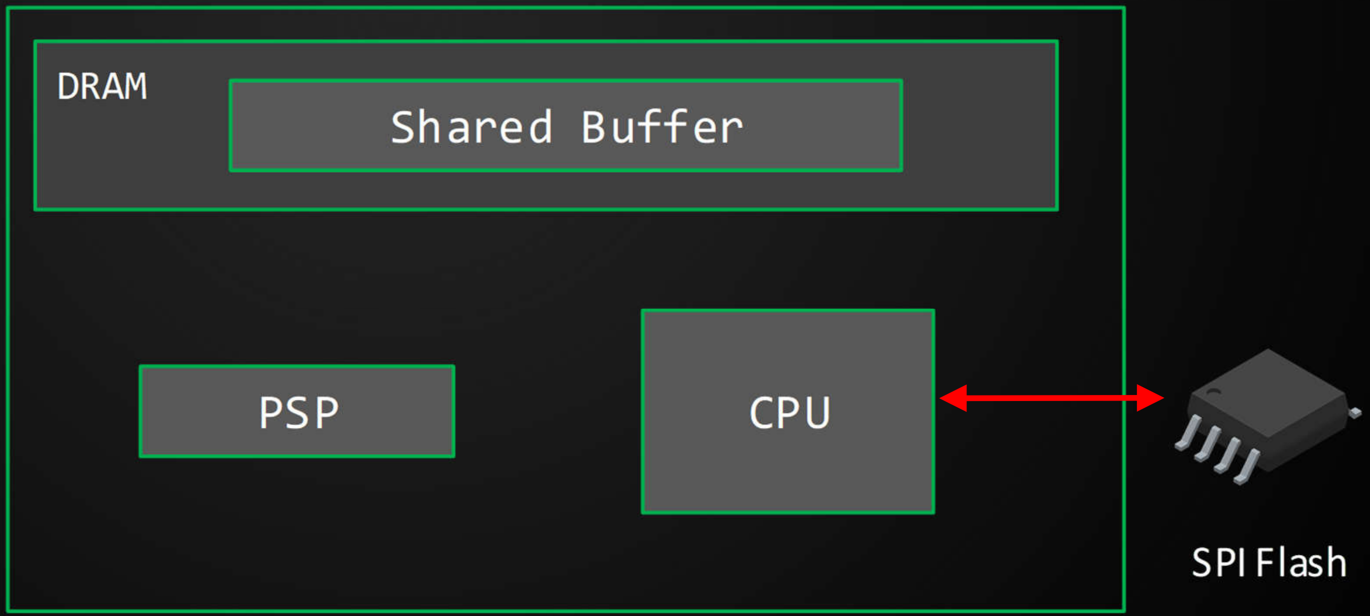
P2C Interface Flow

P2C Interface Flow

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

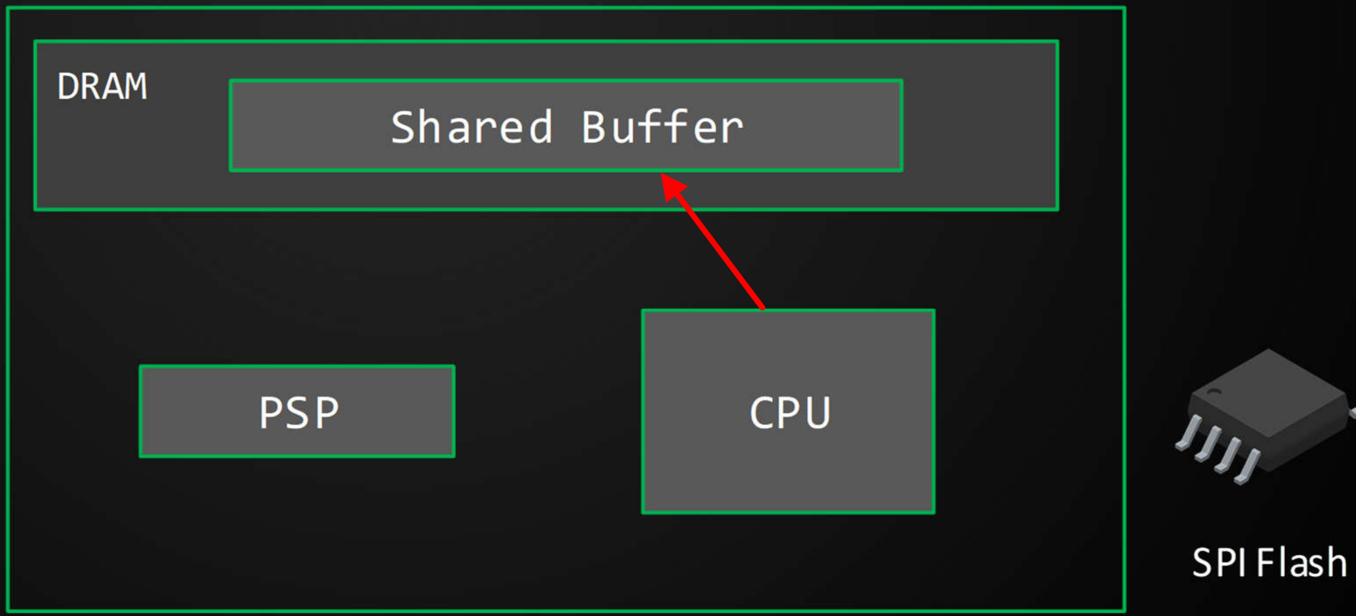


[3] SMM code reads/writes data to/from SPI flash on behalf of the PSP.



P2C Interface Flow

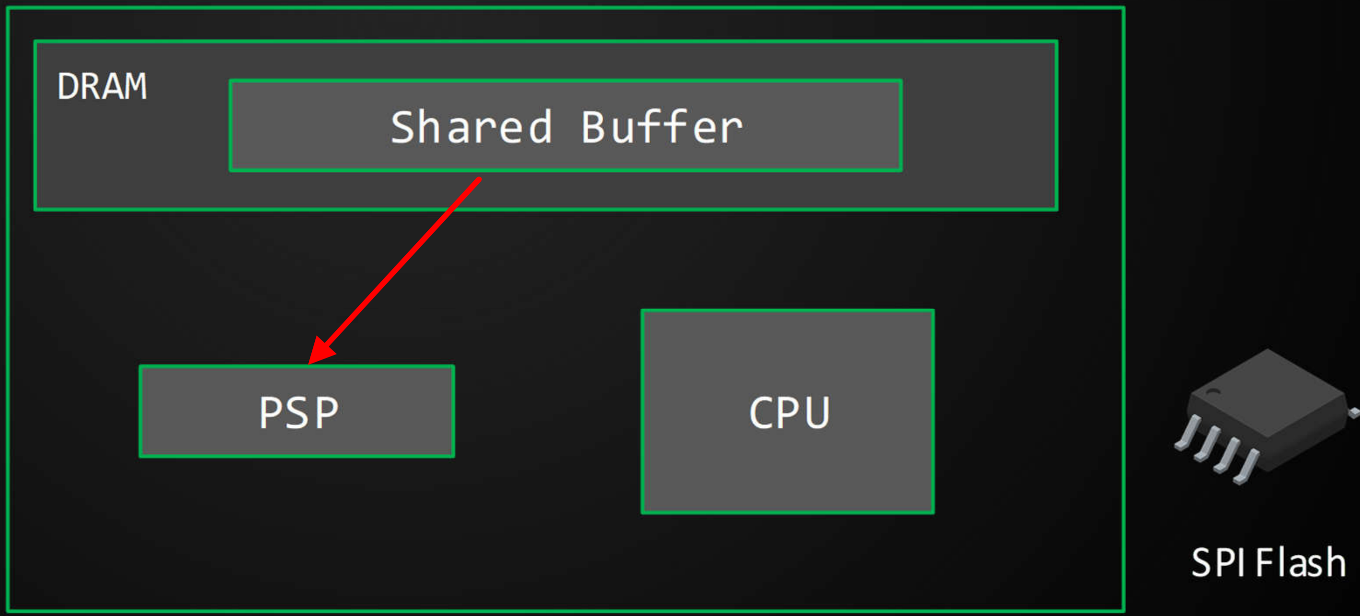
[4] Results are written back to the shared buffer, and the PSP is signaled.



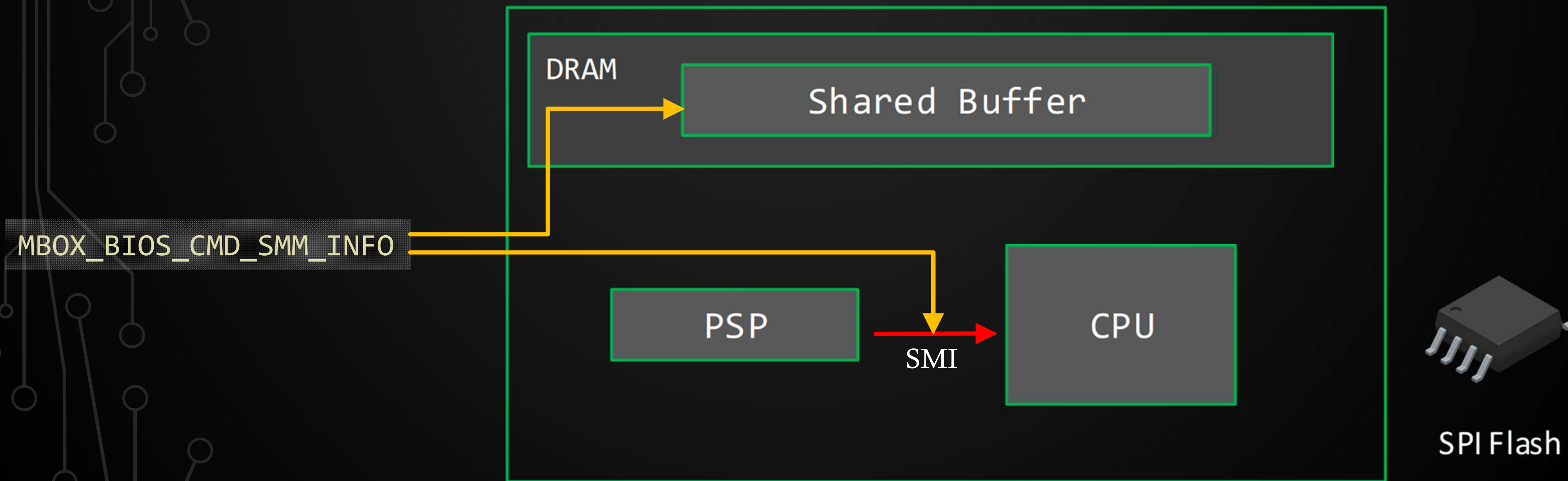
P2C Interface Flow

P2C Interface Flow

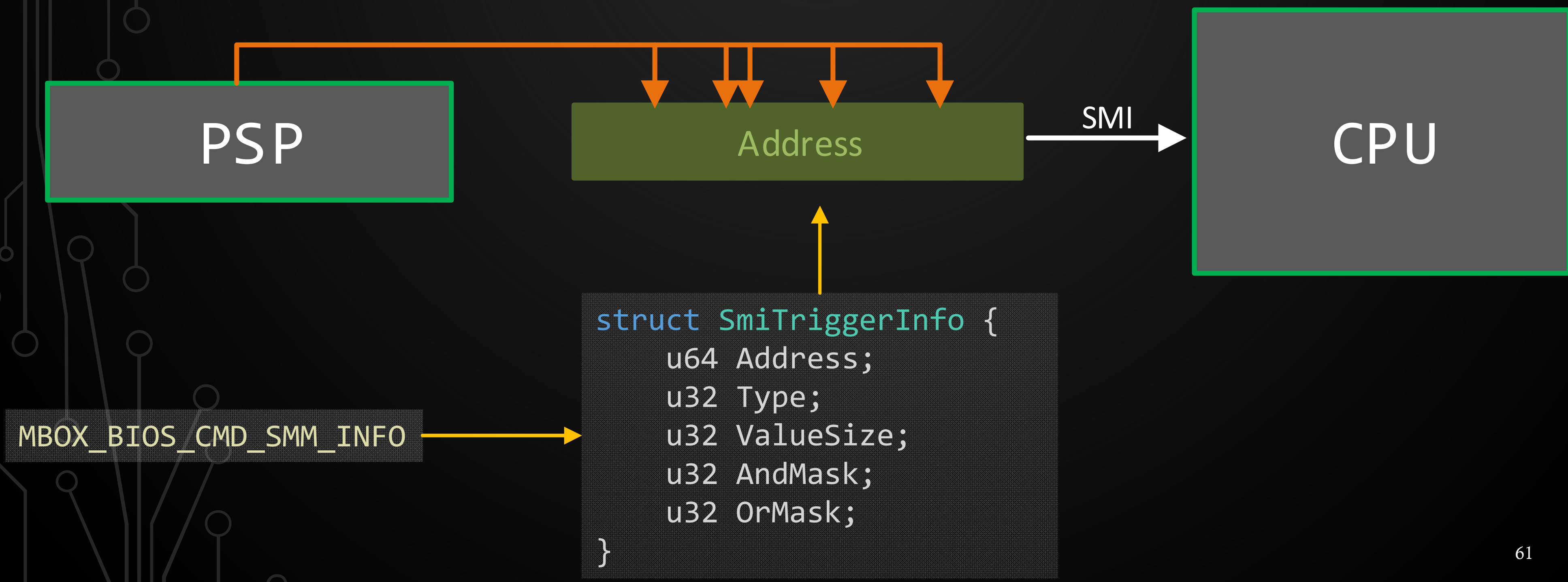
[5] PSP reads the result from the same shared buffer.



SMM_INFO sets the shared buffer and SMI information



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 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);
    // ….
}
```

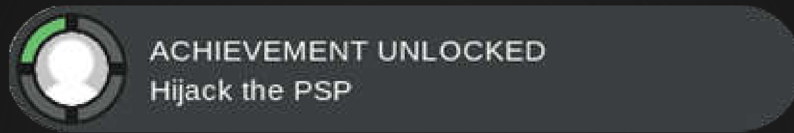
Double fetch -- We can switch ValueSize under its feet!

```
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);
    // ….
}
```

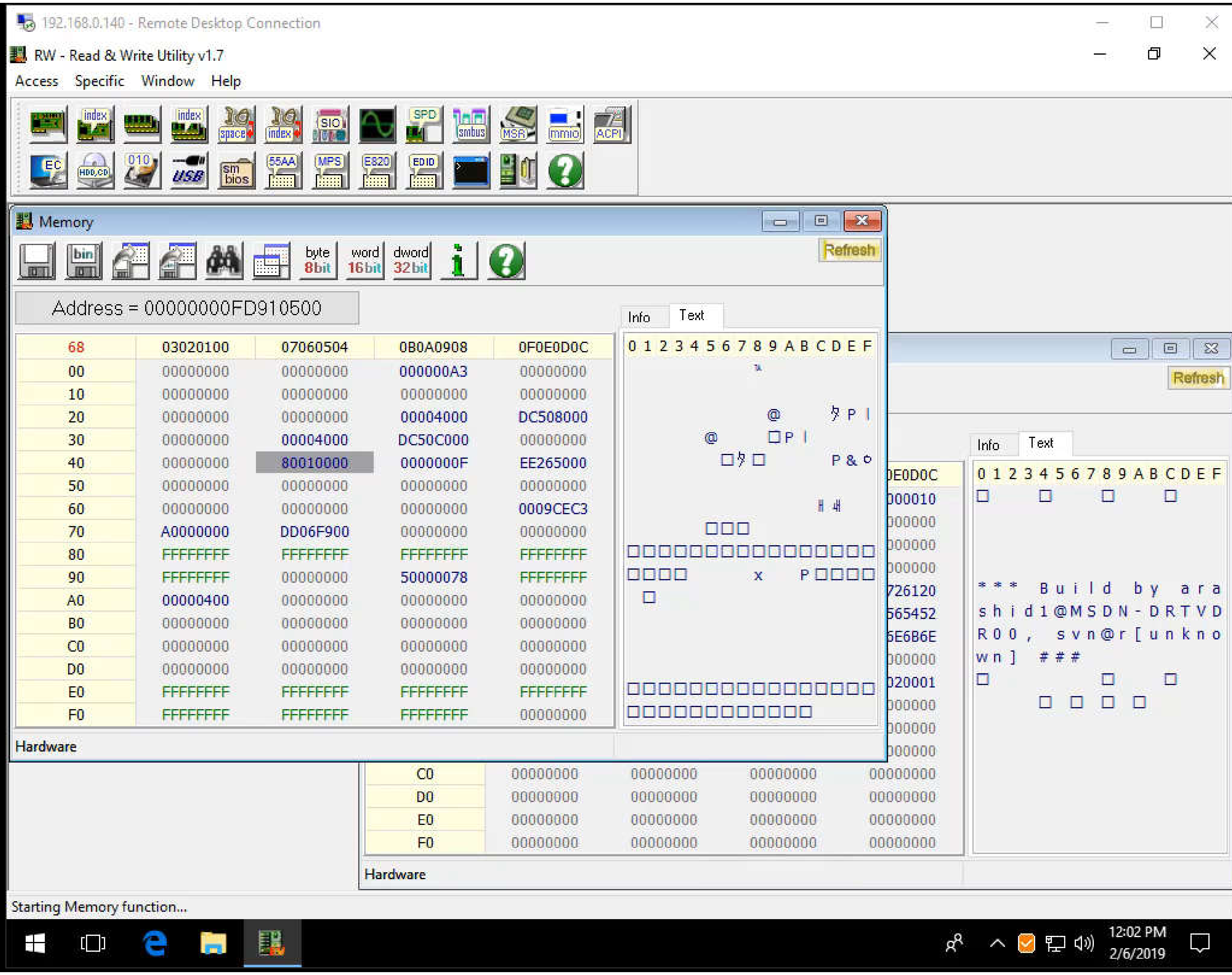


Ryzenfall-4

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

Ryzenfall-4

DEMO



192.168.0.140 - Remote Desktop Connection
RW - Read & Write Utility v1.7
Access Specific Window Help
Memory
byte 8bit
word 16bit
dword 32bit
Refresh
Address = 00000000FD910500
Info Text
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
Hardware
*** Build by arashid1@MSDN-DRTVDR00, svn@r[unknown] ###
Starting Memory function...
12:02 PM
2/6/2019

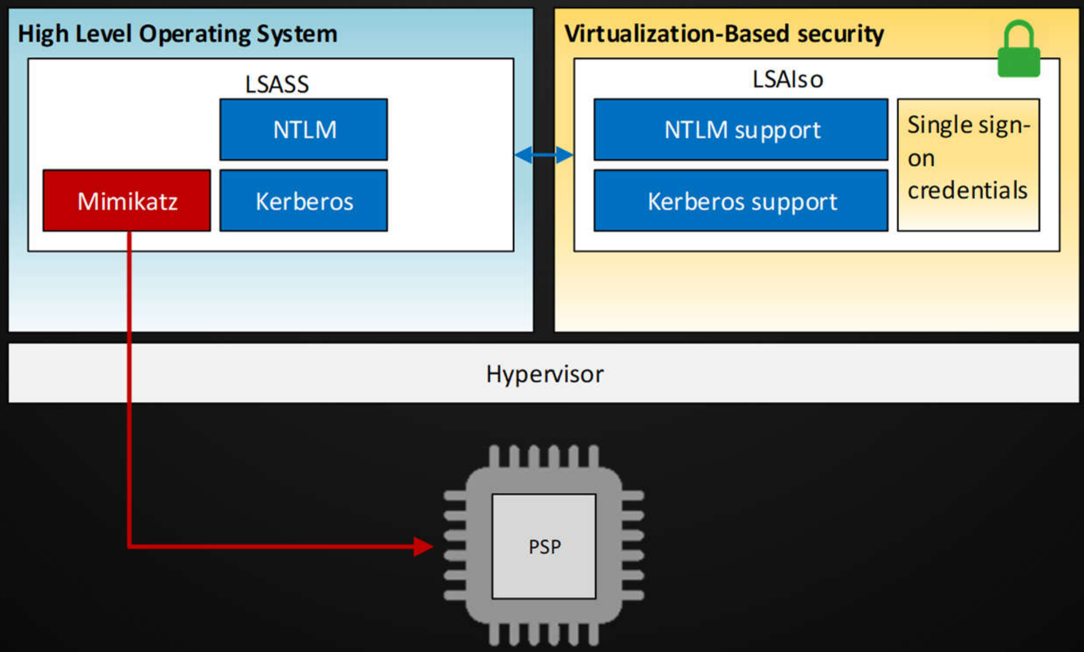
Problem: Credential Guard Breaks Mimikatz

Credential Guard causes Mimikatz to fail

Solution: Use PSP to Break Into Isolated VM

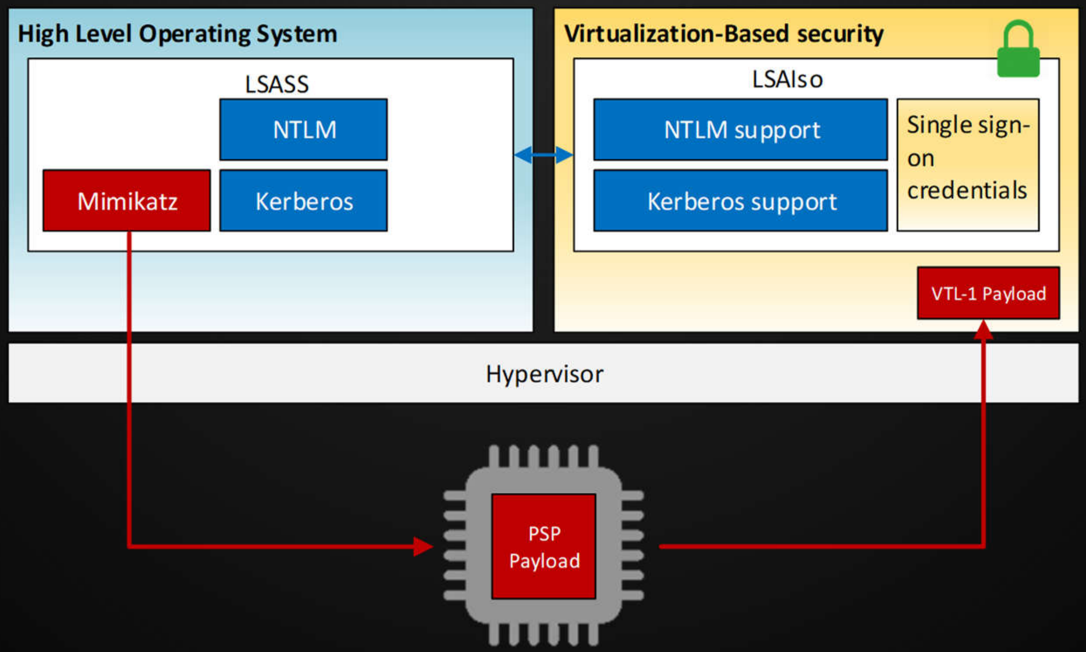
Bypass Credential Guard in 3 simple steps

Step 1: Inject payload into PSP

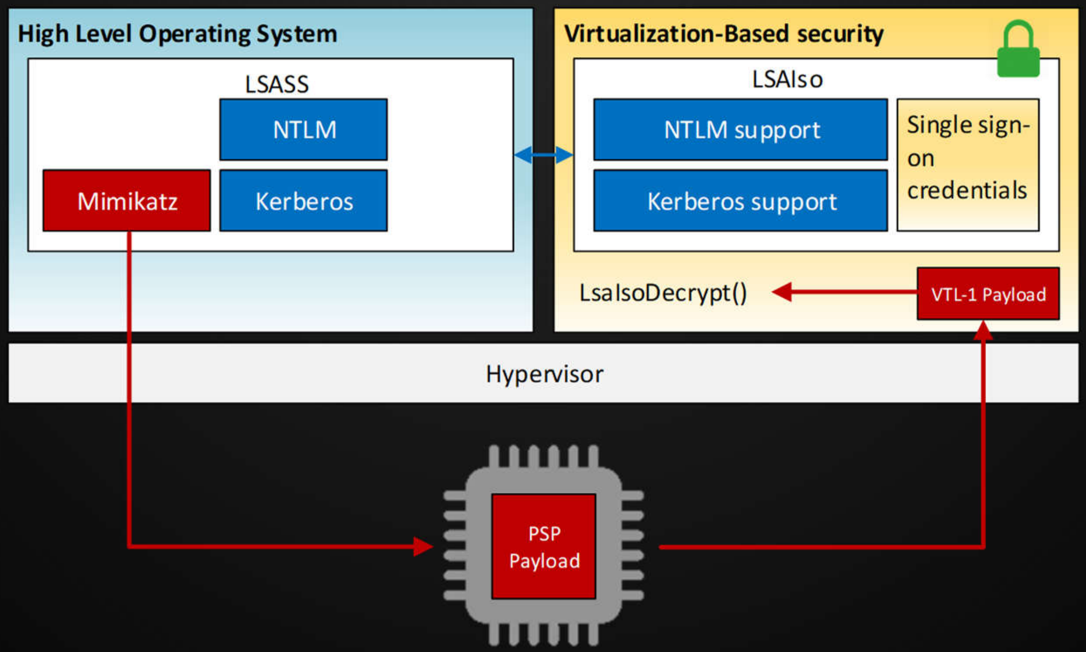


- Either Masterkey or Ryzenfall will do

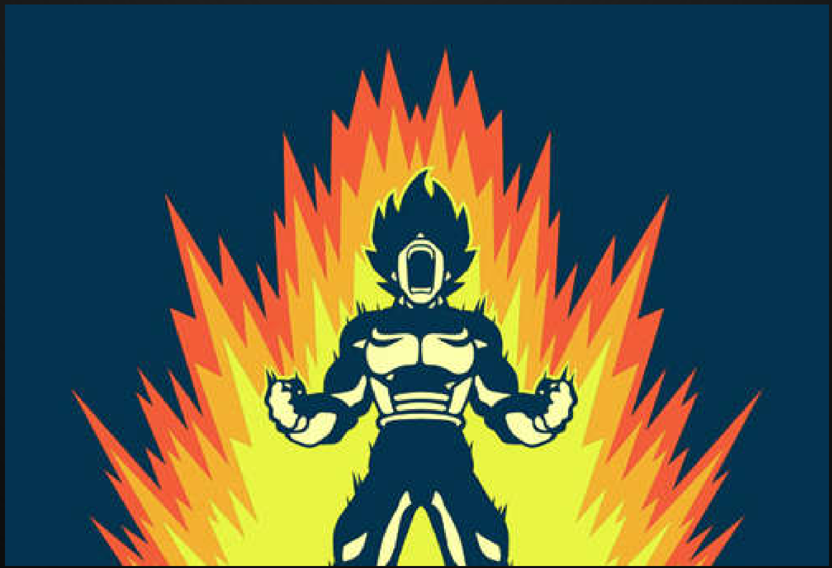
Step 2: From PSP, Inject into Isolated LSA



Step 3: Profit!

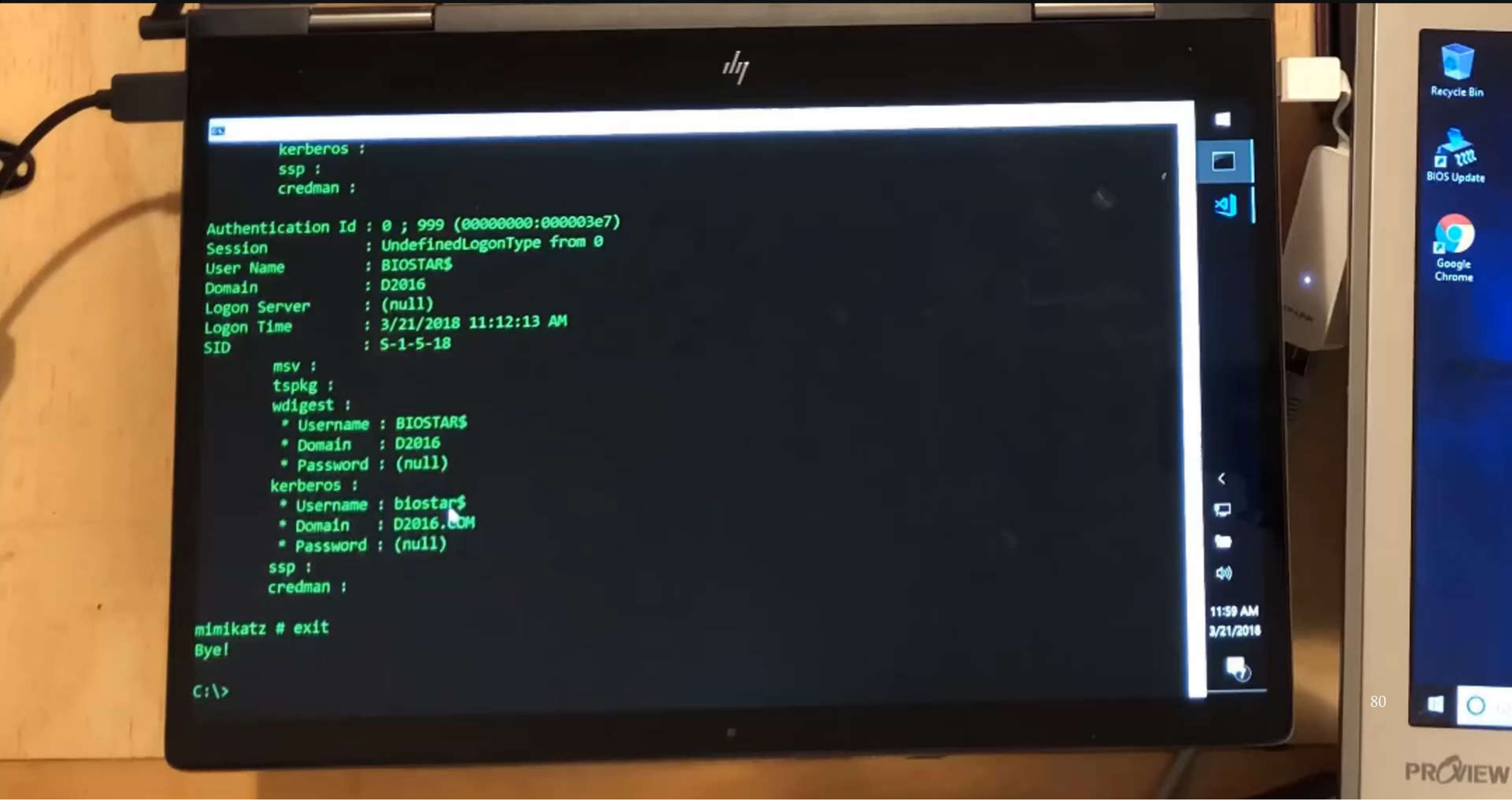


Mimikatz



Power-Up

Mimikatz -- PSP Power-Up Edition



Thank you!

idolion@cts-labs.com | uri@cts-labs.com

Side note: Debugging PSP signature exploits