Rogue CDB: Escaping from VMware Workstation Through the Disk Controller

Wenxu Yin
Senior Vulnerability Researcher, Qihoo 360
About The Speaker

- Wenxu Yin @awxylitol
- Senior Vulnerability Researcher
- Alpha Lab, Vulnerability Research Institute, Qihoo 360
- Hypervisors: VMware Workstation/ESXi and QEMU
- Network Devices: Juniper, SonicWall, Ubiquiti and NETGEAR
About Vulnerability Research Institute

- OS, Browser and Hypervisor Security Research
- https://vul.360.net
- Over 2000 CVEs
- Pwn2Own 2017 and Tianfu Cup 2018/2019/2020 Champion
- Pwnie Awards 2019/2020
Agenda

I. Disk Controllers and VMware's Implementation

II. Root Cause and Exploit Primitives

III. The Exploitation Process

IV. Takeaways and Q&A
I. Disk Controllers and VMware's Implementation
SCSI and CDB

What is a disk controller?

- PCI/PCIe Interface
- SCSI (Small Computer System Interface)
- SATA (Serial AT Attachment)
- IDE (Integrated Drive Electronics)

SCSI and CDB

What is a disk controller?

➢ VMware Workstation 17.0 Pro
➢ Creating a 64 bit Linux Guest VM on a Windows Host
➢ SAS (Serial Attached SCSI)
What is a disk controller?

- A disk controller is typically plugged into one of the PCI/PCIe slots on the motherboard and sits between the driver in the OS and the disks.
- In the case of a hypervisor, the emulated disk controller is exposed to the Guest OS via the emulated PCI interface, and the hard disk itself is merely a large file stored on the Host OS.
SCSI and CDB

The SCSI specification

➢ SCSI is a protocol used principally to talk to storage devices such as hard disks and tape drives.
➢ The SCSI standards define commands, protocols, electrical, optical and logical interfaces.

SCSI and CDB

The SCSI specification

➢ Parallel SCSI (formally, SCSI Parallel Interface, or SPI) is the earliest of the interface implementations in the SCSI family.

➢ Serial Attached SCSI (SAS) is a point-to-point serial protocol. SAS replaces the older Parallel SCSI.
SCSI and CDB

SCSI and CDB

The Command Descriptor Block (CDB) protocol

➢ In SCSI standards for transferring data between computers and peripheral devices, often computer storage, commands are sent in a CDB.

➢ Each CDB can be a total of 6, 10, 12, or 16 bytes, but later versions of the SCSI standard also allow for variable-length CDBs.
## Table 2: Typical CDB for 6-byte commands

<table>
<thead>
<tr>
<th>Bit Byte</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>OPERATION CODE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Miscellaneous CDB information</td>
<td>(MSB)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>LOGICAL BLOCK ADDRESS (if required)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>(LSB)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>TRANSFER LENGTH (if required)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>PARAMETER LIST LENGTH (if required)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ALLOCATION LENGTH (if required)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>CONTROL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SCSI and CDB

The Command Descriptor Block (CDB) protocol

➢ The **first** byte of a SCSI CDB is an **operation code** that specifies the command that the application client is requesting the device server to **perform**

- [Group 0 - Six-byte commands (00 to 1F)](https://www.t10.org/lists/op-num.htm)
- [Group 1 - Ten-byte commands (20 to 3F)](https://www.t10.org/lists/op-num.htm)
- [Group 2 - Ten-byte commands (40 to 5F)](https://www.t10.org/lists/op-num.htm)
- Group 3 - reserved
- [Group 4 - Sixteen-byte commands (80 to 9F)](https://www.t10.org/lists/op-num.htm)
- [Group 5 - Twelve-byte commands (A0 to BF)](https://www.t10.org/lists/op-num.htm)
- Group 6 - vendor specific
- Group 7 - vendor specific
SCSI and CDB

<table>
<thead>
<tr>
<th>OP</th>
<th>Description</th>
<th>Device Column key</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>MMMMMMMMMMMM</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>M</td>
<td>REWIND</td>
</tr>
<tr>
<td>01</td>
<td>Z ZZZ</td>
<td>REZERO UNIT</td>
</tr>
<tr>
<td>02</td>
<td>V VVV V</td>
<td>TEST UNIT READY</td>
</tr>
<tr>
<td>03</td>
<td>MMMMMMMMMMMM MMMMM M</td>
<td>REQUEST SENSE</td>
</tr>
<tr>
<td>04</td>
<td>MC O</td>
<td>FORMAT UNIT</td>
</tr>
<tr>
<td>04</td>
<td>O</td>
<td>FORMAT MEDIUM</td>
</tr>
<tr>
<td>05</td>
<td>V MVV V</td>
<td>FORMAT</td>
</tr>
<tr>
<td>05</td>
<td>V MVV V</td>
<td>READ BLOCK LIMITS</td>
</tr>
<tr>
<td>06</td>
<td>O V O V</td>
<td>REASSIGN BLOCKS</td>
</tr>
<tr>
<td>07</td>
<td>O</td>
<td>INITIALIZE ELEMENT STATUS</td>
</tr>
<tr>
<td>08</td>
<td>Z M O V</td>
<td>READ(6)</td>
</tr>
<tr>
<td>08</td>
<td>O</td>
<td>RECEIVE</td>
</tr>
<tr>
<td>08</td>
<td></td>
<td>GET MESSAGE(6)</td>
</tr>
</tbody>
</table>

https://www.t10.org/lists/op-num.htm
VMware’s Implementation

How a virtual hard disk device works

➢ lspci -ktv
➢ LSI Logic / Symbios Logic 53c1030 PCI-X Fusion-MPT Dual Ultra320 SCSI
➢ SCSI Disk Controller from LSI Corporation
➢ VMware emulates it, the default hard disk controller for a 64 bit Linux Guest VM on VMware Workstation
VMware's Implementation

```
orboxes@orboxes:~$ lspci -ktv
-0000:00.0 Intel Corporation 440BX/ZX/DX - 82443BX/ZX/DX Host bridge
  +01.0-[01]--
  +07.0 Intel Corporation 82371AB/EB/MB PIIX4 ISA
  +07.1 Intel Corporation 82371AB/EB/MB PIIX4 IDE
  +07.3 Intel Corporation 82371AB/EB/MB PIIX4 ACPI
  +07.7 VMware Virtual Machine Communication Interface
  +0f.0 VMware SVGA II Adapter
  +10.0 LSI Logic / Symbios Logic 53c1030 PCI-X Fusion-MPT Dual Ultra320 SCSI
  +11.0-[02]---+00.0 VMware USB1.1 UHCI Controller
      |      +01.0 Intel Corporation 82545EM Gigabit Ethernet Controller (Copper)
      |      +02.0 VMware USB2 EHCI Controller
      \-04.0 VMware SATA AHCI controller
```
VMware's Implementation

How a virtual hard disk device works

- Driver on Linux is called **mptspi**
- BAR (Base Address Register)
- PMIO: BAR0, 0x1400, Size 256
- MMIO: BAR1, 0xFEB80000, Size 0x20000;
  - BAR3, 0xFEBA0000, Size 0x20000;
VMware's Implementation

```
vmboxes@osboxes:~$ lspci -kvvv -s 00:10.0
00:10.0 SCSI storage controller: LSI Logic / Symbios Logic 53c1030 PCI-X Fusion-MPT Dual Ultra320 SCSI (rev 01)
  Subsystem: VMware LSI Logic Parallel SCSI Controller
  Control: I/O+ Mem+ BusMaster+ SpecCycle- MemWINV- VGASnoop- ParErr- Stepping- SERR- FastB2B- DisINTx-
  Status: Cap+ 66MHz- UDF- FastB2B+ ParErr- DEVSEL=medium >TABort- <TABort- <MAbort- >SERR- <PERR- INTx-
  Latency: 64 (1500ns min, 63750ns max)
  Interrupt: pin A routed to IRQ 17
  Region 0: I/O ports at 1400 [size=256]
  Region 1: Memory at feb00000 (64-bit, non-prefetchable) [size=128K]
  Region 3: Memory at feba0000 (64-bit, non-prefetchable) [size=128K]
  [virtual] Expansion ROM at c0008000 [disabled] [size=16K]
  Capabilities: <access denied>
  Kernel driver in use: mptspi
  Kernel modules: mptspi
```
VMware's Implementation

How a virtual hard disk device works

- Linux Kernel 6.1.19
- drivers/message/fusion/lsi/mpi_init.h
- drivers/message/fusion/lsi/mpi.h
VMware’s Implementation

How a virtual hard disk device works

- VMware Workstation 17.0.0 Build 20800274
- RPC Handler for the LSI SCSI Controller
- a2 should be MSG_SCSI_IO_REQUEST from Guest
- v6 is malloced to store the overall SCSI CDB Request
VMware's Implementation

case 7:
    sub_14025B550(v2, (unsigned __int8 *)(a1 + 36), *((QWORD *)(a1 + 24)));
    break;

__int64 __fastcall sub_14025B550(__int64 a1, unsigned __int8 *a2, __int64 a3)
{
// [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

    v6 = sub_14071E390(a1 + 696);
    *((QWORD *)v6 + 16920) = a3;
    *((QWORD *)v6 + 16856) = *((QWORD *)a2);
    *((QWORD *)v6 + 16872) = *((QWORD *)a2 + 1);
    *((QWORD *)v6 + 16888) = *((QWORD *)a2 + 2);
    *((QWORD *)v6 + 16904) = *((QWORD *)a2 + 6);
    *((DWORD *)v6 + 16912) = *((DWORD *)a2 + 14);
    *((QWORD *)v6 + 32) = a3;
    *((QWORD *)v6 + 40) = v6 + 16880;
    *((DWORD *)v6 + 48) = a2[4];
    *((BYTE *)v6 + 60) = a2[13];
VMware's Implementation

How CDB commands are processed in VMware Workstation

➢ Then **v6** is passed to the **generic** SCSI CDB handler function
➢ This function **sub_1402129A0()** also handles SCSI CDB from **other** disk controllers like PVSCSI, BusLogic, etc.

```
LABEL_30:
   *(BYTE *)(v6 + 61) = v14;
   return sub_1402129A0(*(QWORD *)(a1 + 232), v6, *a2);
}

if ( !*(BYTE *)(a2 + 66) )
   return sub_1402129A0(v3, (__int64)v7, *((BYTE *)(v7 + 16908));
sub_1405FE110("PVSCSI: Failing request to bus=%u\n", *(unsigned __int8 *)(a2 + 66));
v14 = 17i64;
v7[2073] = 17i64;
```
VMware's Implementation

How CDB commands are processed in VMware Workstation

- Check is done in `sub_140211F30()`
- If it passes, the CDB is sent to the respective `handler` functions of different SCSI devices, like CD Drive or Hard Disk in `sub_14021BEC0()`
VMware's Implementation

LOBYTE(v7) = sub_140211F30(a1, v5, a2);
if ( (_BYTE)v7 )
{
    v8 = *((QWORD *)NtCurrentTeb()->ThreadLocalStoragePointer + (unsigned int)TlsIndex);
    if ( *((DWORD *)(v8 + 11776) )
        v6 = *((DWORD *)(v8 + 11776) - 1);
    *((DWORD *)(a2 + 24) ) = v6;
    v9 = sub_1405E98B0();
    if ( a1[2] != 5 )
        v9 += *((DWORD *)(v5 + 544));
    *((DWORD *)(a2 + 72) ) = v9;
    *((DWORD *)(a2 + 64) ) = sub_140094520();
    L_BYTE(v7) = sub_140211C80(v5, a2, 1);
    if ( (_BYTE)v7 )
    {
        ++*((DWORD *)(v5 + 192));
        v10 = *((int64 **)(v5 + 200));
        if ( v10 )
        {
            v11 = *v10;
            *((DWORD *)(a2 + 8) ) = v10;
            *((DWORD *)(a2 ) = v11;
            *((DWORD *)(v11 + 8) ) = a2;
            *v10 = a2;
        }
        else
        {
            *((DWORD *)(a2 + 8) ) = a2;
            *((DWORD *)(a2 ) = a2;
            *((DWORD *)(v5 + 200) ) = a2;
        }
    }
    L_BYTE(v7) = sub_14021BE0C0(v5, a2);
VMware's Implementation

```c
__int64 __fastcall sub_14021BEC0(__int64 a1, __int64 a2)
{
    // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

    v2 = a2;
    LOBYTE(a2) = 1;
    sub_140119000(_, ___QWORD ___)(a1 + 552), a2);
    if ( *(_QWORD *)(a1 + 496) && *(_BYTE *)(v2 + 40) != 3 )
        *(_QWORD *)(a1 + 496) = 0164;
    v4 = *(_BYTE *)(v2 + 40);
    v5 = *(unsigned int *)(v2 + 128);
    *(_DWORD *)(v2 + 56) = v5;
    if ( *v4 != 3 )
        return *(__int64 __fastcall **)(__int64, __int64, __int64 __fastcall *(__QWORD, __QWORD), __QWORD)(a1 + 24)(
            a1,
            v2,
            sub_14021BBF0,
            0164);
    v6 = *(_QWORD *)(a1 + 496);
    if ( !v6 )
        return *(__int64 __fastcall **)(__int64, __int64, __int64 __fastcall *(__QWORD, __QWORD), __QWORD)(a1 + 24)(
            a1,
            v2,
            sub_14021BBF0,
            0164);
```

VMware’s Implementation

What kind of check does it have

➢ CDB Length
➢ CDB Operation Code

Table 2: Typical CDB for 6-byte commands

<table>
<thead>
<tr>
<th>Bit</th>
<th>Byte</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

typedef struct _MSG_SCSI_IO_REQUEST
{
    UB            TargetID; /* 00h */
    UB            Bus;     /* 01h */
    UB            CIOffset;/* 02h */
    UB            Function; /* 03h */
    UB            CDBLength; /* 04h */
    UB            SBufferLength; /* 05h */
    UB            Reserved; /* 06h */
    UB            MsgFlags;  /* 07h */
    U32           MsgContext; /* 08h */
    UB            LUN[8];  /* 0Ch */
    UB            Control; /* 14h */
    UB            CDB[16]; /* 18h */
    UB            Datalength; /* 28h */
    U32           SenseBufferLowAddr; /* 2Ch */
    SGE_TO_UNION  SGL; /* 30h */
} MSG_SCSI_IO_REQUEST, MPI_POINTER PTR_MSG_SCSI_IO_REQUEST, 
SCSIIORequest_t, MPI_POINTER pSCSIIORequest_t;
VMware's Implementation

What kind of check does it have

➢ \( v5 = *(\text{unsigned int} *)(a3 + 48); \) is the **CDB Length** set by the Guest
➢ \( *(\text{unsigned } \_\text{int8 }**)(a3 + 40); \) is the CDB, and \( v7 = **(\text{unsigned } \_\text{int8 }**)(a3 + 40); \) is the **Operation Code**
➢ CDB Length and Operation Code have to be **consistent**

- **Group 0** - Six-byte commands (00 to 1F)
- **Group 1** - Ten-byte commands (20 to 3F)
- **Group 2** - Ten-byte commands (40 to 5F)
- Group 3 - reserved
- **Group 4** - Sixteen-byte commands (80 to 9F)
- **Group 5** - Twelve-byte commands (A0 to BF)
- Group 6 - vendor specific
- Group 7 - vendor specific

; unsigned _int8 byte_1409D9238[8]
byte_1409D9238 db 6, 2 dup(0Ah), 40h, 10h, 0Ch, 2 dup(41h)
VMware's Implementation

```c
char __fastcall sub_140211F30(_QWORD *a1, __int64 a2, __int64 a3)
{
    // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

    v27 = -1;
    v5 = *(unsigned int *)(a3 + 48);
    v7 = **(unsigned __int8 **)(a3 + 40);
    v8 = byte_1409D9238[(unsigned __int64)**(unsigned __int8 **)(a3 + 40) >> 5];
    if ( v8 != v5 )
    {
        if ( v8 == 0x40 )
        {
            v9 = (unsigned int)dword_140DFE484++;
            if ( (unsigned __int8)sub_1406044A0(v9, v5) )
                sub_1405FE110("
SCSI (%s): Operation rejected: reserved opcode %x, cdbLen %u\n",
(const char *)(a2 + 752),
    v7,
    *(unsigned int *)(a3 + 48));
LABEL_30:
    sub_14021B9D0(a3, 5, 32, 0, 112);
    goto LABEL_31;
    }
    if ( v8 == 0x41 )
```
II. Root Cause and Exploit Primitives
Root Cause

Why does this vulnerability exist?

➢ Assumption is broken with the introduction of newer specifications.

3d. Out-of-bounds read/write vulnerability (CVE-2023-20872)

Description

VMware Workstation and Fusion contain an out-of-bounds read/write vulnerability in SCSI CD/DVD device emulation. VMware has evaluated the severity of this issue to be in the Important severity range with a maximum CVSSv3 base score of 7.7.

https://www.vmware.com/security/advisories/VMSA-2023-0008.html
Root Cause

Why does this vulnerability exist?

- **a3** is the CDB Length, which can be 0x6, 0xA, 0xC, 0x10, **0x40**, 0x41
- **a2** is the CDB
- Clearly, the assumed maximum length of CDB is **0x10**
Root Cause

Why does this vulnerability exist?

➢ Page Heap enabled
➢ Crash at memcpy()
Root Cause

The Fix

➢ VMware Workstation 17.0.1 Build 21139696
➢ Check the Operation Code Group first
➢ Then check the consistency between the CDB Length and the Operation Code
Exploit Primitives

OOB Read

- Page Heap enabled
- dst/RCX is the 0x158 chunk(\texttt{v16}) + offset 0x138 malloced above
- src/RDX is the 0x4228 chunk(\texttt{v6}) + offset 0x41F8 malloced in the LSI Logic function
Exploit Primitives

Exploit Primitives
Exploit Primitives

OOB Read

- sub_14071E390() returns the src chunk + 8
- sub_140603000() is a wrapper of malloc()

```c
_int64 __fastcall sub_1402585500(__int64 a1, unsigned __int8 *a2, __int64 a3)
{
    // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

    v6 = sub_14071E390(a1 + 696);
    *(DWORD *)(v6 + 16920) = a3;
    *(DWORD *)(v6 + 16856) = *(DWORD *)a2;
    *(DWORD *)(v6 + 16872) = *((DWORD *)a2 + 1);
    *(DWORD *)(v6 + 16888) = *((DWORD *)a2 + 2);
    *(DWORD *)(v6 + 16904) = *((DWORD *)a2 + 6);
    *(DWORD *)(v6 + 16912) = *((DWORD *)a2 + 14);
    *(DWORD *)(v6 + 32) = a3;
    *(DWORD *)(v6 + 40) = v6 + 16880;
    *(DWORD *)(v6 + 48) = a2[4];
    *(BYTE *)(v6 + 60) = a2[13];

    v3 = sub_140603000(*(DWORD *)a1 + 8i64);
    if (*(_DWORD *)(v1 + 8))
        v1 = 0i64;
    *v3 = v1;
    return v3 + 1;
}
```
Exploit Primitives

OOB Read

- **0x20 bytes within** src chunk
- **0x41F8 to 0x4228**, minus CDB[16]
- DataLength(U32), SenseBufferLowAddr(U32), SGL(FlagsLength(U32), Address64(U64))
- Something at the end of the src chunk
Exploit Primitives

OOB Read

- **0x10** bytes from the following chunk
- src is **0x4228** chunk
- Non-LFH on Windows 10
Exploit Primitives

OOB Write

- **0x10** bytes *within* the **dst** chunk
- 0x138 to 0x158 minus CDB[0x10]

```c
struct v16 {
    char padding[0x138];
    char CDB[0x10];
    void *func_ptr;
    void *second_param;
};
```
Exploit Primitives

OOB Write

- **0x20** bytes into the following chunk
- dst is a **0x158** chunk
- May be on LFH
Exploit Primitives

OOB Write

➢ Arbitrary Call

➢ a9 is \texttt{sub\_14080DAA0()}

➢ a10 is \texttt{v16}, the \texttt{0x158} chunk

\begin{verbatim}
return sub_140839B60(
    (*DWORD *)(a1 + 64),
    DWORD) a2,
    a3,
    (DWORD)v17,
    v11 != 0,
    a7,
    v16 + 48,
    255i64,
    __int64 sub_14080DAA0, v16);
\end{verbatim}
Exploit Primitives

OOB Write
➢ Arbitrary Call
➢ Inside `sub_14080DAA0()`
➢ `func_ptr` is at `v16/RBX + 0x148`
➢ `second_param` is at `v16/RBX + 0x150`

```c
struct v16 {
    char padding[0x138];
    char CDB[0x10];
    void *func_ptr;
    void *second_param;
}
```
Exploit Primitives

OOB Write

- Arbitrary Call
- **RIP** and **RDX** are controlled by us
- if we overflow func_ptr with **0**, call will **not** happen
III. The Exploitation Process
Linear vmem

How is the guest physical memory implemented?

➢ On a 64 bit Linux Guest with 4GB memory, the address space of the physical memory is not 0x00000000 – 0xFFFFFFFF, but is divided into two parts: 0x00000000 – 0xBFFFFFFF, 0x100000000 – 0x3FFFFFFF
Linear vmem

How is the guest physical memory implemented?

➢ The physical memory of the Guest is mapped as the `.vmem` file at 0x7FFF0000 – 0x17FFF0000 **linearly**

➢ Read/Write a **HVA** of 0x7FFF0000 + 0x1000 is the same as a **GPA** of 0x0 + 0x1000
Exploit on Linux

What do we have?

- No CFG
- RIP and RSI (2nd parameter) controlled

```assembly
mov    rax, [rbp+148h]
test   rax, rax
jz     short loc_62D20B
mov    rsi, [rbp+150h]
mov    edi, r12d
call   rax
```
Exploit on Linux

The one gadget

➢ Tried searching for something like "mov rdi, rsi"
➢ ropper --file vmware-vmx --search "mov rdi, rsi"
➢ One more Arbitrary Call

```
.text:00000000000693BD8          mov   rdi, rsi
.text:00000000000693BDB          call   qword ptr [rsi+30h]
```
Exploit on Linux

The one gadget

➢ RSI points to "/usr/bin/gnome-calculator"
Exploit on Windows

Bypass CFG

➢ Without triggering this bug, the original handler function is `sub_14028EC90()`
Exploit on Windows

Bypass CFG

➢ I was playing with the Arbitrary Call primitive with the func_ptr overflowed with 0 when a crash happened since the OOB Write destroyed some chunks on the heap.

➢ This function looks interesting, if ONLY I could find one that uses the second parameter like this.
Exploit on Windows

```assembly
__int64 __fastcall sub_1406BA90(__int64 a1)
{
    __DWORD *v2; // rcx
    void (__fastcall *v3)(__DWORD, __DWORD, __DWORD, __DWORD); // rax
    void *v4; // rcx

    if ( *(__DWORD *)(a1 + 32) == 2 )
    {
        v2 = *(__DWORD **)(a1 + 96);
        if ( v2 )
            *v2 = *(__DWORD *)(a1 + 28);
    }
    v3 = *(void (__fastcall **)(__DWORD, __DWORD, __DWORD, __DWORD))(a1 + 8);
    if ( v3 )
        v3(__DWORD *)(a1 + 16), *(unsigned int *)(a1 + 24), *(unsigned int *)(a1 + 28), *(__DWORD *)(a1 + 328));
    if ( *(__DWORD *)(a1 + 32) <= 1u )
    {
        v4 = *(void **)(a1 + 64);
        if ( v4 )
            if ( (void *)(a1 + 72) != v4 )
            {
                free(v4);
                *(__DWORD *)(a1 + 64) = 0164;
            }
    }
    return sub_14071E450(a1);
}
```
Exploit on Windows

```c
void __fastcall sub_14028EC90(__int64 a1, __QWORD *a2)

    // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

    v2 = a2[1];
    v11 = *(_BYTE *)(*(_QWORD *)(v2 + 16) + 168164);
    v4 = **(unsigned __int8 **) (v2 + 40);
    sub_1401850D0(*(_QWORD *)(a2 + 960164));
    v5 = a2[1];
    v6 = *((unsigned __int8 *)a2 + 32);
    v7 = *((unsigned __int8 *)a2 + 33);
    v8 = *((unsigned __int8 *)a2 + 34);
    *(_DWORD *)(v5 + 56) = *(_DWORD *)(*(_QWORD *)(v5 + 96) + 8164) - *(_DWORD *)(v5 + 56);
    ...

LABEL_8:

    ((void (__fastcall *)__int64, __QWORD)a2[2])(v5, a2[3]);

    free(a2);
```
Exploit on Windows

Bypass CFG

➢ It is the **original** callback function!
➢ With the second parameter already under our control, we can make another Arbitrary Call
➢ We do not even have to control **RIP**
➢ **Data-Only** Exploitation
Exploit on Windows

Bypass CFG

➢ We can point **RDX** to **vmem** to arrange the required elements of the **a2** structure in the **Guest** directly

➢ Set a2 to 0x7FFF0000 + 0x1000, we can write at the **physical** address of 0x1000 in the Guest
Exploit on Windows

Bypass CFG

➢ a2[2] points to `KERNEL32!WinExec()
➢ a2[1] points to "calc.exe"
➢ a2[3] is `1(SW_SHOWNORMAL)
➢ a2[2](a2[1], a2[3]);
Exploit on Windows

The features of this kind of function

- One of its parameters points to a **structure** with a function **pointer** that will get called and the **parameters** of the function stored inside
- Turn one call into a call “chain”
Live Demonstration: Linux
Live Demonstration: Windows
IV. Takeaways and Q&A
Takeaways

➢ The disk controllers of VMware hypervisors are complex and may have more bugs;
➢ It pays to read the specifications when doing hypervisor bug hunting;
➢ When exploiting certain type of bugs, we can put the data in the guest physical memory directly.
Credits

➢ Lei SHI, mentor, encouragement and guidance

➢ Guang GONG, @oldfresher, director, freedom of research
THANK YOU!