black hat®
ASIA 2024

APRIL 18-19, 2024
BRIEFINGS

URB Excalibur: The New VMware All-Platform VM Escapes

Yuhao Jiang (@danis_jiang)

Xinlei Ying (@0x140ce)

Who are we?

Security researchers at Ant Group Light-Year Security Lab

Escaped from virtual machine many times

Won the Pwnie Awards🦄 at 2023

蚂蚁安全实验室
Ant Security Lab

蚂蚁光年
Ant Light-Year

Yuhao Jiang
(@danis_jiang)

Xinlei Ying
(@0x140ce)

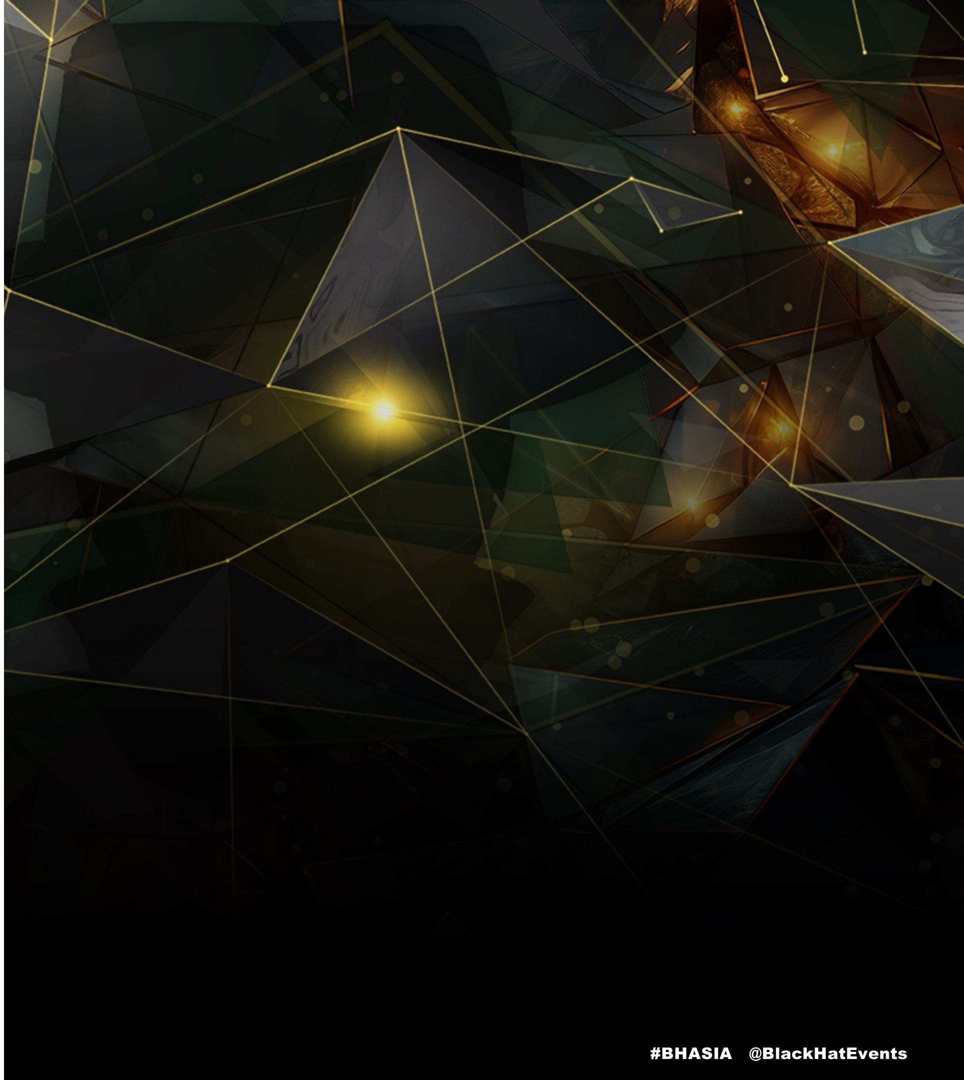
Talk Roadmap

1. Introduction

2. A journey of finding vulnerabilities in VMware’s hypervisor

3. Exploit development of VMware VM escape

Introduction



What is Virtual Machine escape and the danger of it

- Escape from the isolation sphere
- Take control over the whole hypervisor
- Network escape

One of the most catastrophic threats to the Cloud

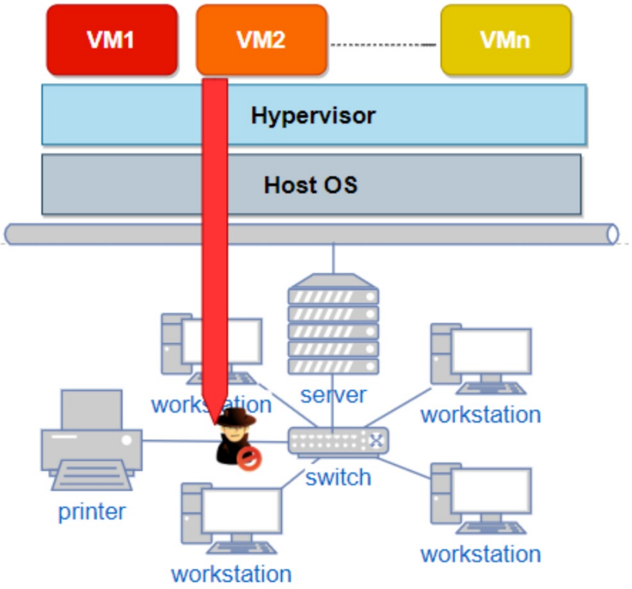


Fig. 6. Demonstration of VM Escape to Network.

VMware's Architecture

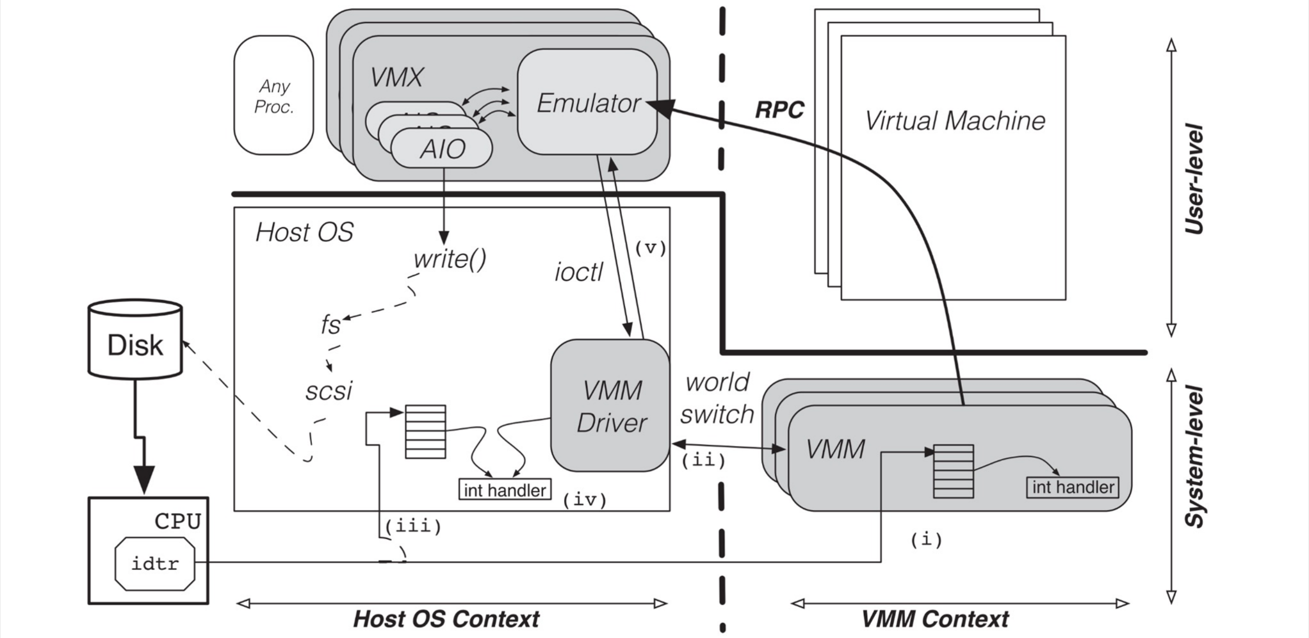


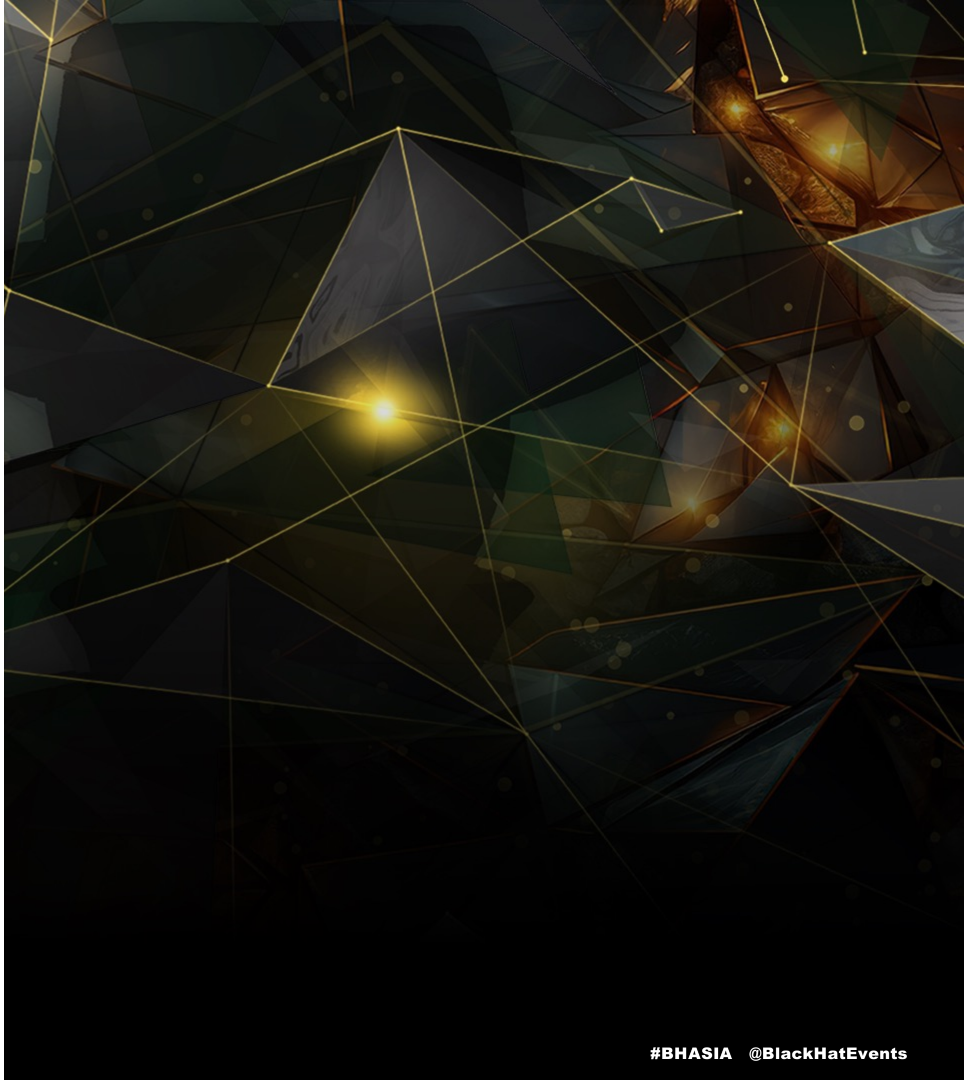
Fig. 2. The VMware Hosted Architecture. VMware Workstation consists of the three shaded components.

VMware hypervisors’ attack surface

Virtual Device	Hard Disk	LSI Logic	
		NVME	
	Network Adapter	E1000/E1000e	
		VMXNET3	
	USB Controller	UHCI	Tianfu Cup 2021 Workstation (CVE-2021-22041), Tianfu Cup 2023 Workstation (CVE-2024-22253, CVE-22255)
		EHCI	GeekPwn 2022 Fusion (CVE-2022-31705)
		XHCI	Tianfu Cup 2021 ESXi (CVE-2021-22040), Tianfu Cup 2023 ESXi (CVE-2024-22252)
	USB Device	HID (mouse)	
		Bluetooth	Pwn2Own 2023 Workstation (CVE-2023-20869, CVE-2023-20870)
		…	
	GPU	SVGA 2D	
		SVGA 3D	
	Sound Card	ES1371	
	TPM	vTPM	
		…	
GuestRPC		Backdoor	
VMM			

Vulnerability Discovery

A journey of finding vulnerabilities in VMware’s hypervisor



Start vulnerability discovery in VMware

First encounter with VMware, closed-source hypervisor

1. **Focusing on an interesting and potentially risky attack surface**
   - Having studied QEMU EHCI vulnerabilities
   - Interested in VMware's EHCI implementation

2. **Reverse engineering**
   - Using string search as an entry point
   - Understanding EHCI specification and QEMU code while reverse engineering VMware

EHCI / USB 2.0 Controller

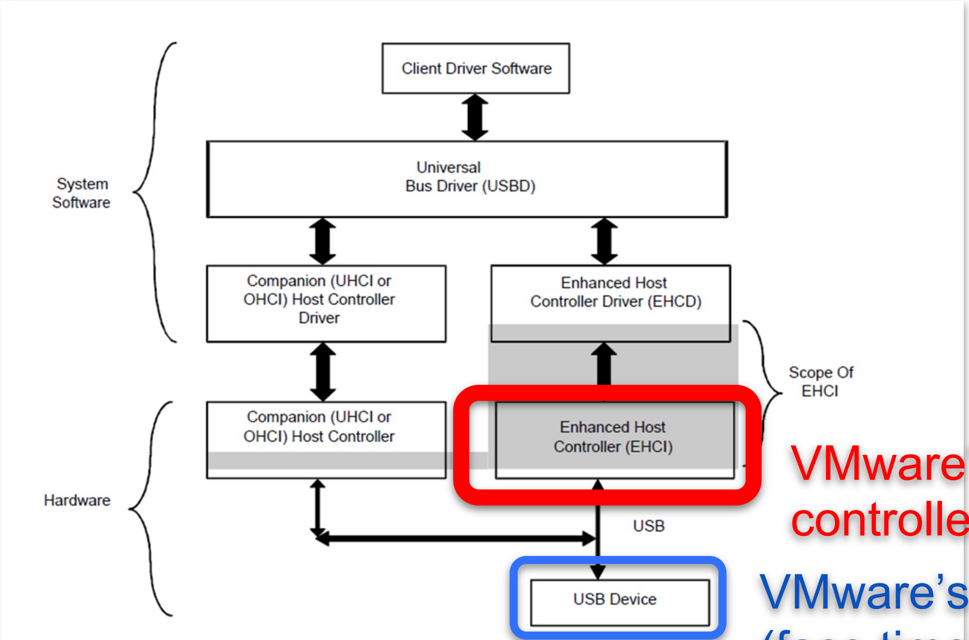
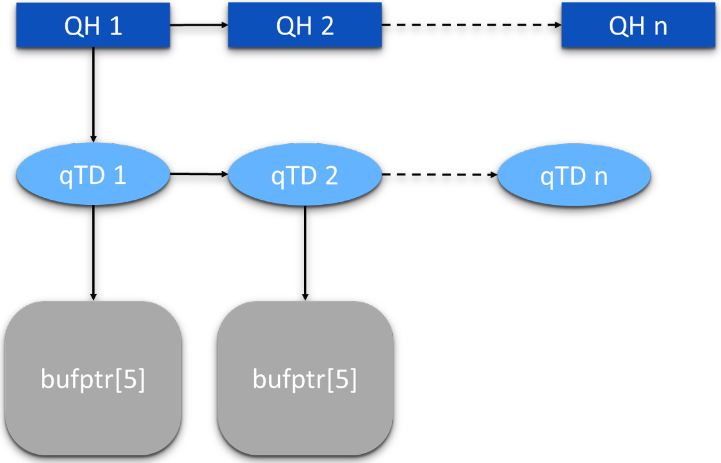

Client Driver Software
Universal Bus Driver (USBD)
System Software
Companion (UHCI or OHCI) Host Controller Driver
Enhanced Host Controller Driver (EHCD)
Scope Of EHCI
Companion (UHCI or OHCI) Host Controller
Enhanced Host Controller (EHCI)
Hardware
USB
USB Device

Figure 1-1. Universal Serial Bus, Revision 2.0 System Block Diagram

VMware's virtual EHCI controller

VMware's virtual video device (face time)

EHCI / USB 2.0 Controller



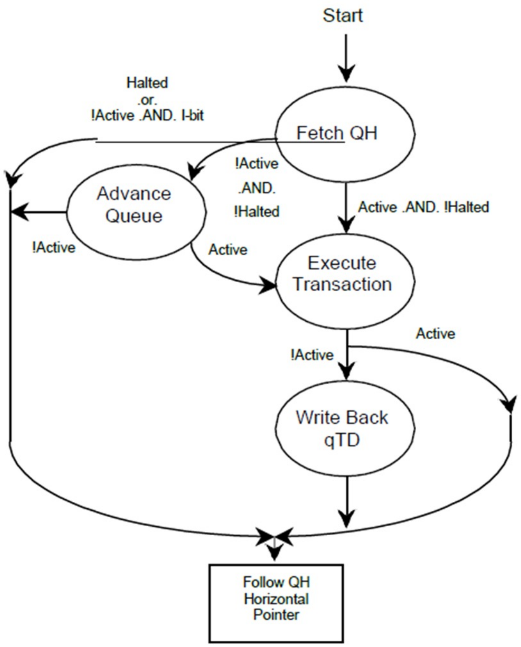


Figure 4-14. Host Controller Queue Head Traversal State Machine

EHCI / USB 2.0 Controller

Endpoint/Pipe:

- Control
- Bulk
- Interrupt
- Isochronous

Token:

- Setup
- In: Device -> Software
- Out: Software -> Device

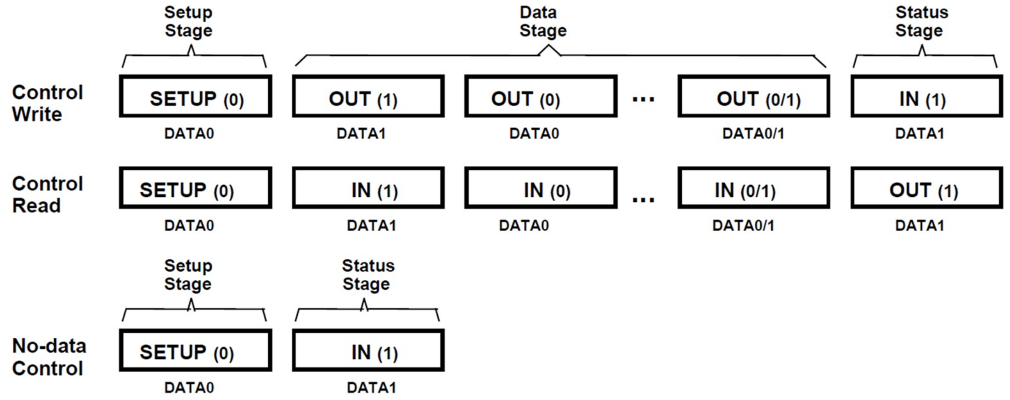
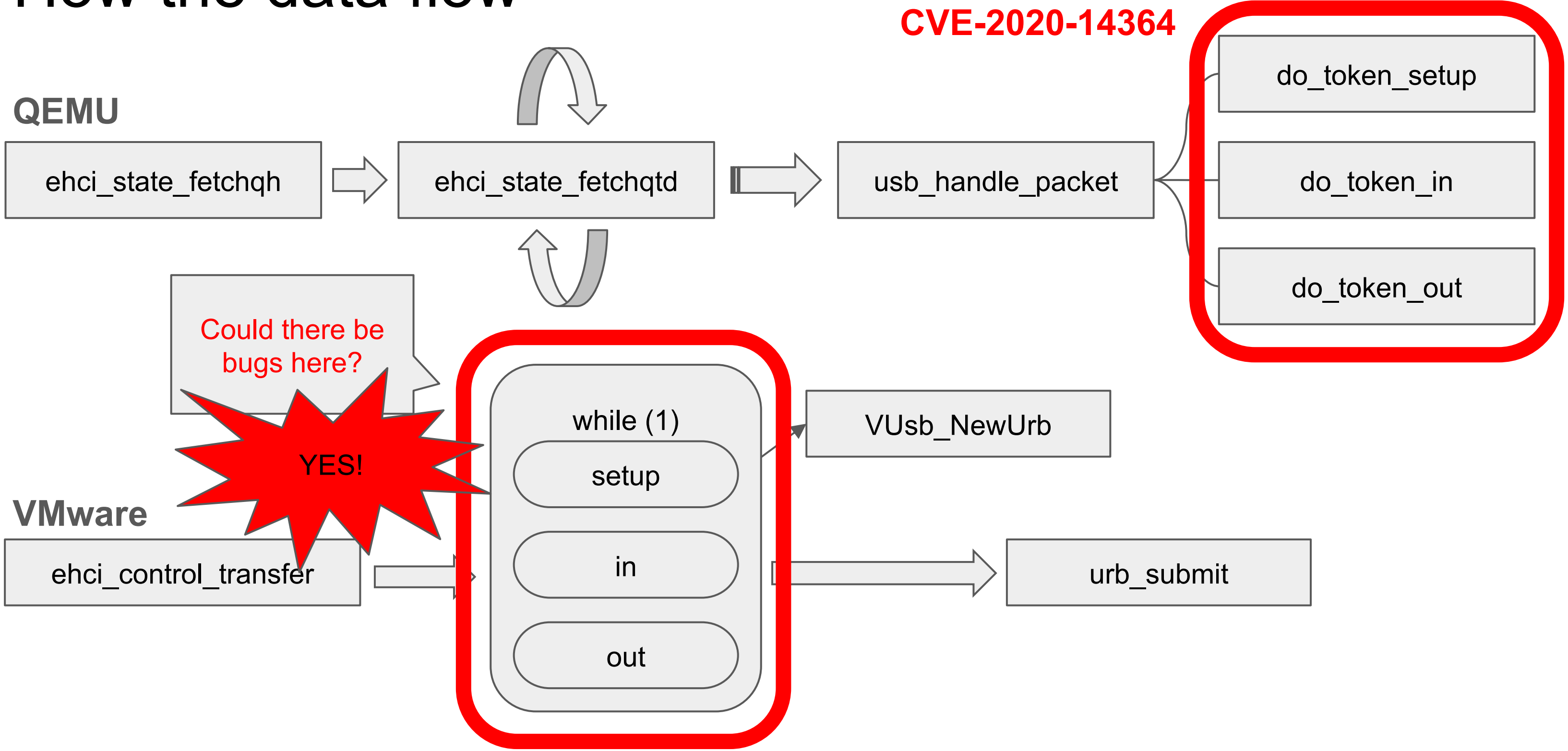
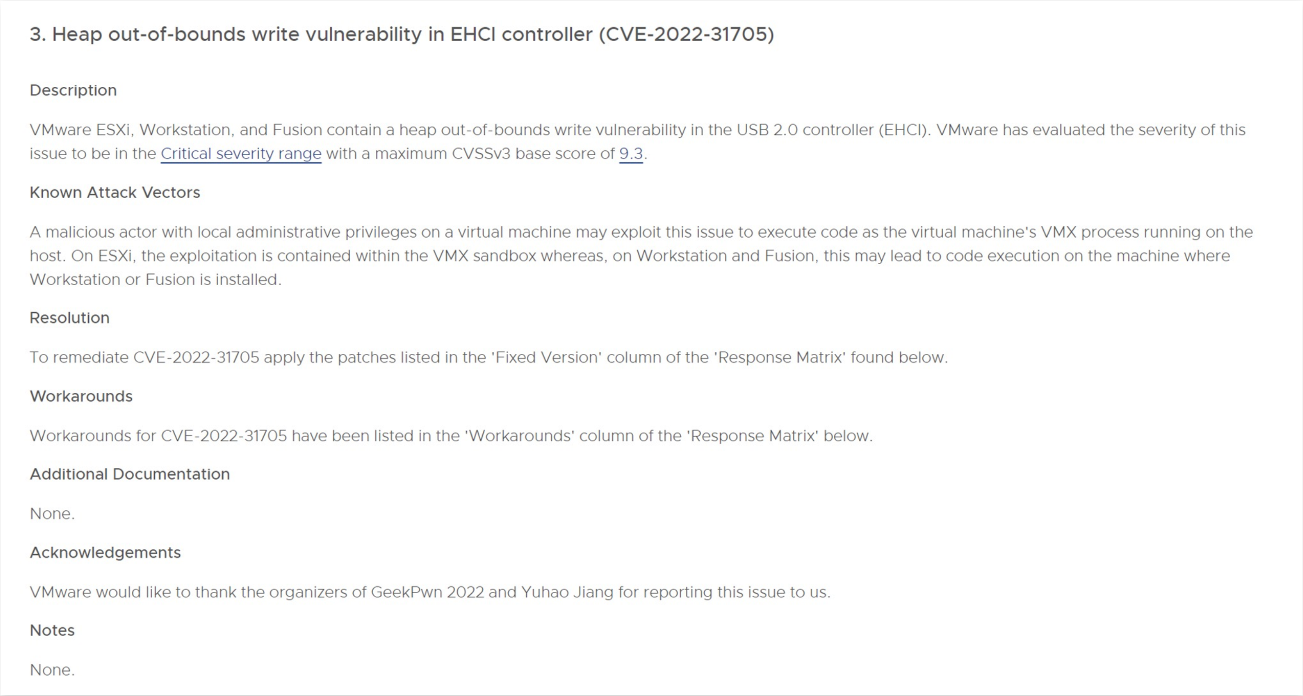


Figure 8-37. Control Read and Write Sequences

How the data flow



CVE-2022-31705



3. Heap out-of-bounds write vulnerability in EHCI controller (CVE-2022-31705)

Description

VMware ESXi, Workstation, and Fusion contain a heap out-of-bounds write vulnerability in the USB 2.0 controller (EHCI). VMware has evaluated the severity of this issue to be in the Critical severity range with a maximum CVSSv3 base score of 9.3.

Known Attack Vectors

A malicious actor with local administrative privileges on a virtual machine may exploit this issue to execute code as the virtual machine's VMX process running on the host. On ESXi, the exploitation is contained within the VMX sandbox whereas, on Workstation and Fusion, this may lead to code execution on the machine where Workstation or Fusion is installed.

Resolution

To remediate CVE-2022-31705 apply the patches listed in the 'Fixed Version' column of the 'Response Matrix' found below.

Workarounds

Workarounds for CVE-2022-31705 have been listed in the 'Workarounds' column of the 'Response Matrix' below.

Additional Documentation

None.

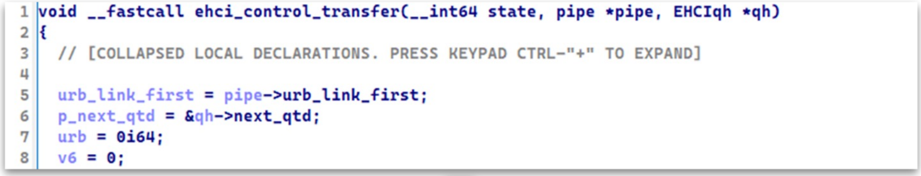
Acknowledgements

VMware would like to thank the organizers of GeekPwn 2022 and Yuhao Jiang for reporting this issue to us.

Notes

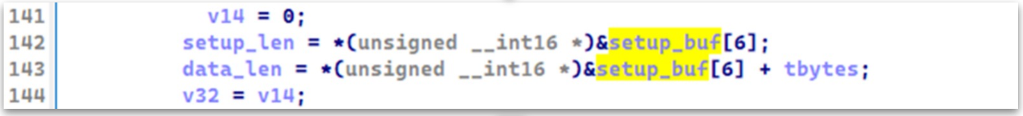
None.

CVE-2022-31705

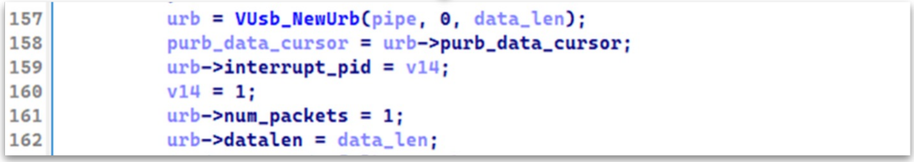


```
void __fastcall ehci_control_transfer(__int64 state, pipe *pipe, EHCIqh *qh)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  urb_link_first = pipe->urb_link_first;
  p_next_qtd = &qh->next_qtd;
  urb = 0i64;
  v6 = 0;
```

SETUP qTD



```
        v14 = 0;
      setup_len = *(unsigned __int16 *)&setup_buf[6];
      data_len = *(unsigned __int16 *)&setup_buf[6] + tbytes;
      v32 = v14;
```

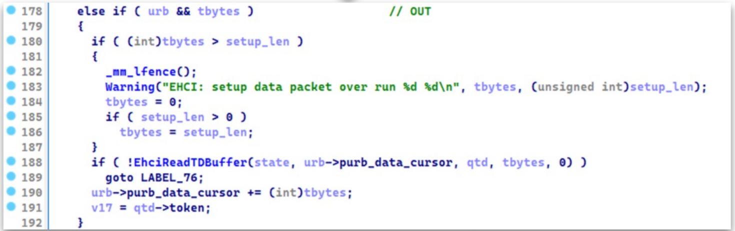


```
        urb = VUsb_NewUrb(pipe, 0, data_len);
        purb_data_cursor = urb->purb_data_cursor;
        urb->interrupt_pid = v14;
        v14 = 1;
        urb->num_packets = 1;
        urb->datalen = data_len;
```

urb’s size =
0x98 + 8 + setup_len

CVE-2022-31705

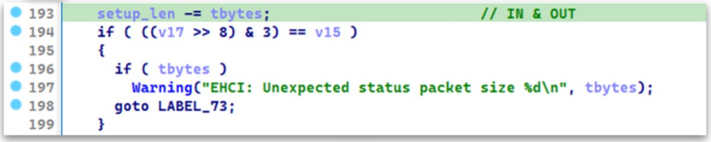
OUT qTD



```
    else if ( urb && tbytes )                           // OUT
    {
      if ( (int)tbytes > setup_len )
      {
        _mm_lfence();
        Warning("EHCI: setup data packet over run %d %d\n", tbytes, (unsigned int)setup_len);
        tbytes = 0;
        if ( setup_len > 0 )
          tbytes = setup_len;
      }
      if ( !EhciReadTDBuffer(state, urb->purb_data_cursor, qtd, tbytes, 0) )
        goto LABEL_76;
      urb->purb_data_cursor += (int)tbytes;
      v17 = qtd->token;
    }
```

IN qTD

Where is the BUG?



```
    setup_len -= tbytes;                                // IN & OUT
    if ( ((v17 >> 8) & 3) == v15 )
    {
      if ( tbytes )
        Warning("EHCI: Unexpected status packet size %d\n", tbytes);
      goto LABEL_73;
    }
```

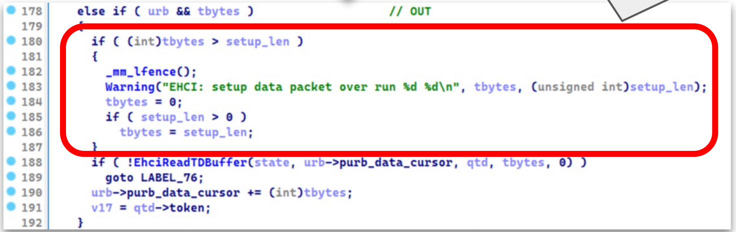
Next qTD

CVE-2022-31705

OUT qTD

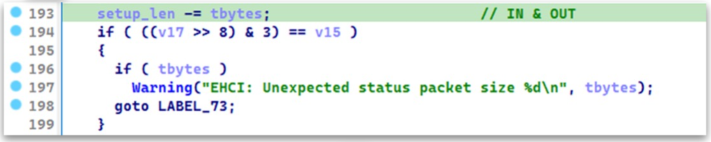
Check if qTD->tbytes is valid

IN qTD



```
else if ( urb && tbytes )                         // OUT
{
  if ( (int)tbytes > setup_len )
  {
    _mm_lfence();
    Warning("EHCI: setup data packet over run %d %d\n", tbytes, (unsigned int)setup_len);
    tbytes = 0;
    if ( setup_len > 0 )
      tbytes = setup_len;
  }
  if ( !EhciReadTDBuffer(state, urb->purb_data_cursor, qtd, tbytes, 0) )
    goto LABEL_76;
  urb->purb_data_cursor += (int)tbytes;
  v17 = qtd->token;
}
```

Where is the BUG?



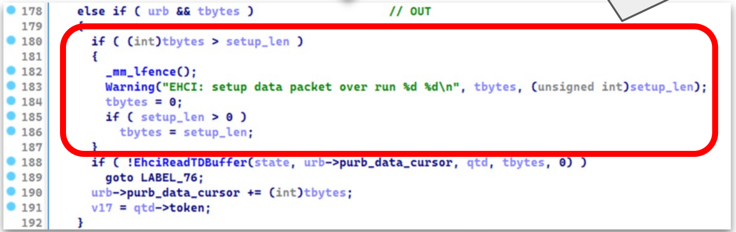
```
setup_len -= tbytes;                              // IN & OUT
if ( ((v17 >> 8) & 3) == v15 )
{
  if ( tbytes )
    Warning("EHCI: Unexpected status packet size %d\n", tbytes);
  goto LABEL_73;
}
```

Next qTD

CVE-2022-31705

OUT qTD

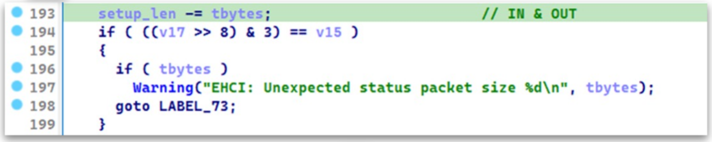
Check if qTD->tbytes is valid



```
else if ( urb && tbytes )                               // OUT
{
  if ( (int)tbytes > setup_len )
  {
    _mm_lfence();
    Warning("EHCI: setup data packet over run %d %d\n", tbytes, (unsigned int)setup_len);
    tbytes = 0;
    if ( setup_len > 0 )
      tbytes = setup_len;
  }
  if ( !EhciReadTDBuffer(state, urb->purb_data_cursor, qtd, tbytes, 0) )
    goto LABEL_76;
  urb->purb_data_cursor += (int)tbytes;
  v17 = qtd->token;
}
```

IN qTD

NO CHECK!
Directly subtract tbytes from setup_len

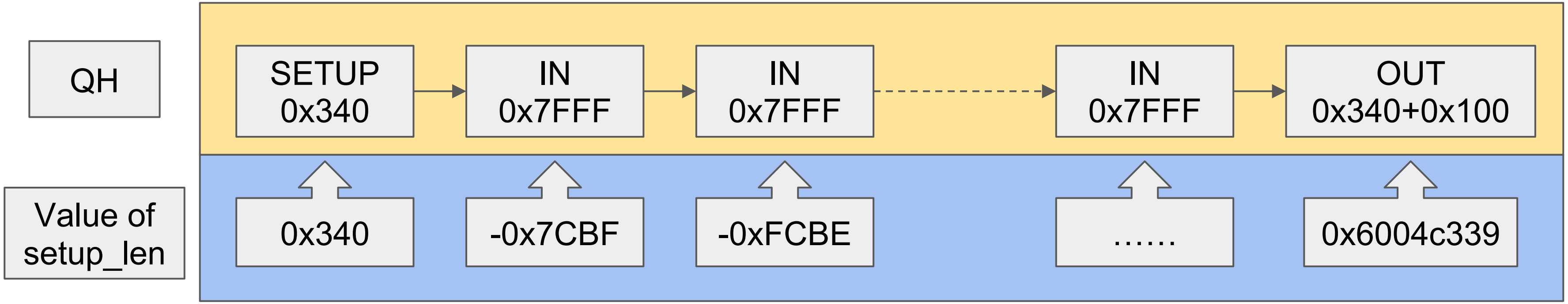


```
setup_len -= tbytes;                               // IN & OUT
if ( ((v17 >> 8) & 3) == v15 )
{
  if ( tbytes )
    Warning("EHCI: Unexpected status packet size %d\n", tbytes);
  goto LABEL_73;
}
```

Next qTD

CVE-2022-31705

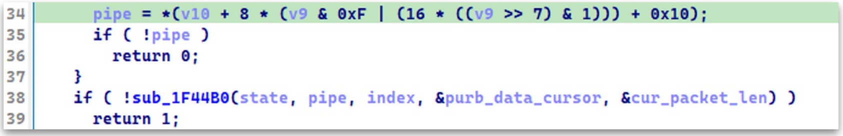
- Missing tbytes check when handling IN qTD
- setup_len downward integer overflow



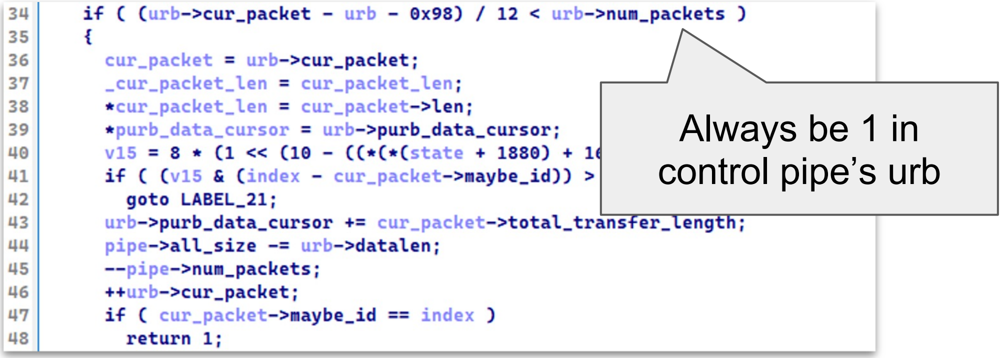
- setup_len is much larger than the size of urb
- Use OUT qTD to obtain heap out-of-bounds write

What else did we find? BUG 1: Out-of-bounds read vulnerability

- Pipe type confusion (Control ⇐⇒ ISOC)
- Handle urb incorrectly



```
    pipe = *(v10 + 8 * (v9 & 0xF | (16 * ((v9 >> 7) & 1))) + 0x10);
    if ( !pipe )
      return 0;
  }
  if ( !sub_1F44B0(state, pipe, index, &purb_data_cursor, &cur_packet_len) )
    return 1;
```



```
  if ( (urb->cur_packet - urb - 0x98) / 12 < urb->num_packets )
  {
    cur_packet = urb->cur_packet;
    _cur_packet_len = cur_packet_len;
    *cur_packet_len = cur_packet->len;
    *purb_data_cursor = urb->purb_data_cursor;
    v15 = 8 * (1 << (10 - ((*(*(state + 1880) + 16
    if ( (v15 & (index - cur_packet->maybe_id)) >
      goto LABEL_21;
    urb->purb_data_cursor += cur_packet->total_transfer_length;
    pipe->all_size -= urb->datalen;
    --pipe->num_packets;
    ++urb->cur_packet;
    if ( cur_packet->maybe_id == index )
      return 1;
```

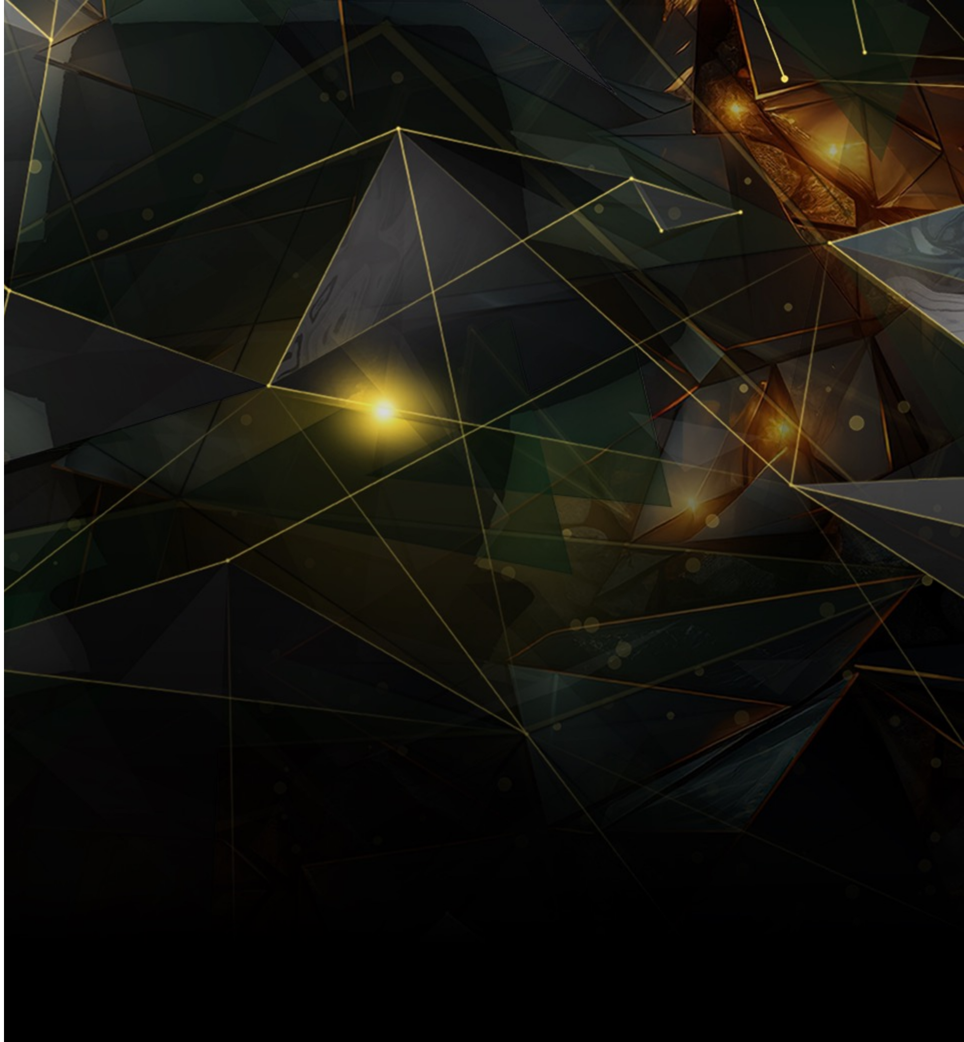
Always be 1 in control pipe’s urb

What else did we find?

BUG 2: Information disclosure vulnerability

- In many virtual USB devices (USB Audio, USB Video, USB RNG…)
- No memset, writeback_len is set to the data size of urb.

Exploit Development



The problem

- **[Again]** Closed-source
- No public exploit code and rarely disclosed exploit flow
- Most of past exploit primitives have been patched
- Few code paths that can be controlled with in the guest OS.

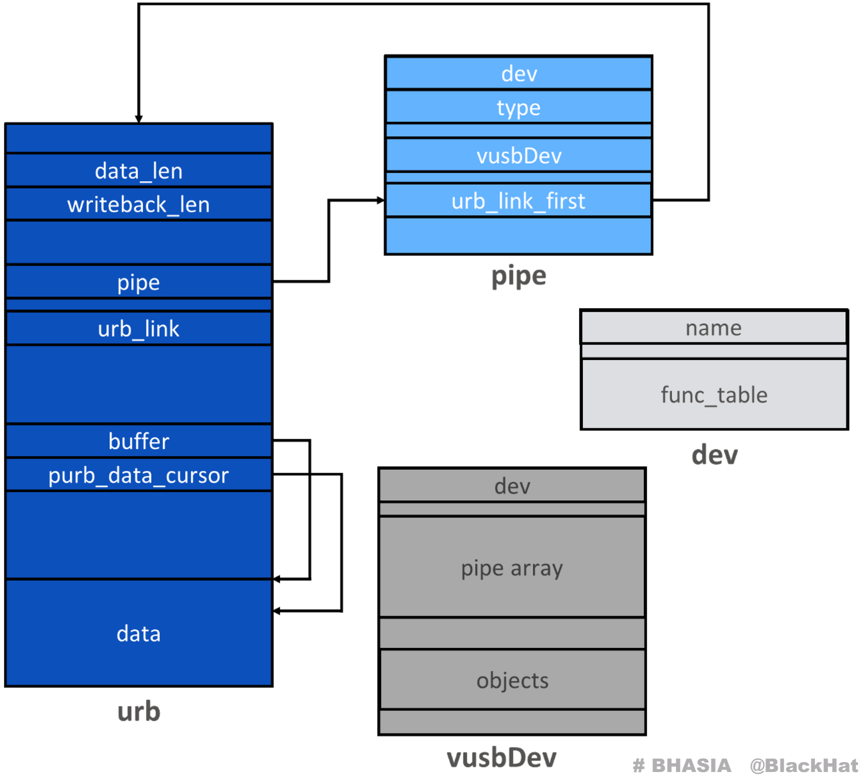
Some patches for old primitives

- DnD/CP objects in **backdoor module** (2017)
  1. VMware remove dynamic allocation and release of DnD/CP objects
- ResourceContainer in **SVGA backend module** (2018)
  1. VMware first removed the function pointer table in the ResourceContainer in 15.5.7
  2. VMware moves SVGA backend module into sandbox (mksSandbox) in 16
- GMR in **SVGA front-end module** (2021)
  1. VMware adds a check at the head of GMR chunk (MKSMemMgrSafeMalloc)
- ……

URB: Powerful Excalibur

- USB Request Block
- Used by all virtual USB controllers

- Dynamic allocate and free
- Has:
  - A variable length data array
  - A member to control length to read
  - A data pointer
  - A pipe pointer
- …

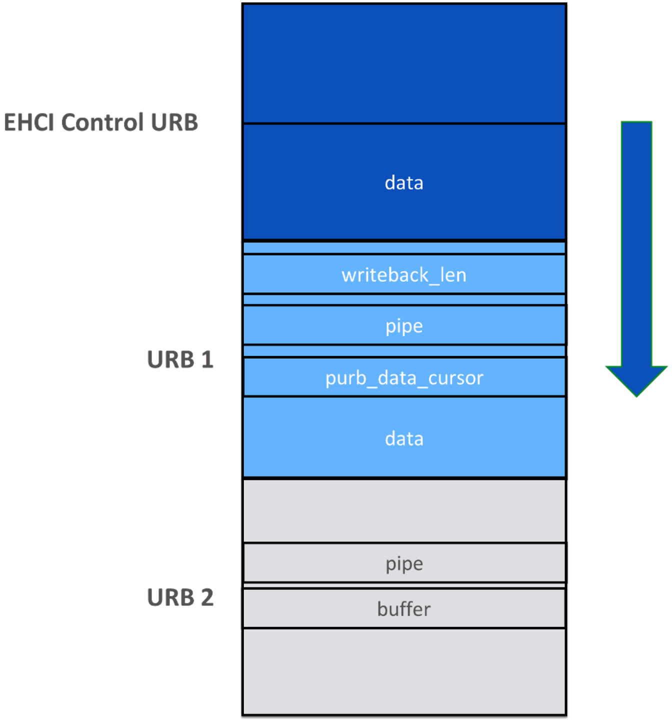


Out-of-bounds write -> Out-of-bounds Read

1. Allocate URB1 and URB2, leaving space for EHCI Control URB

2. Allocate EHCI Control URB, then overwrites writeback_len of URB1

3. Read back URB1, we can read the buffer address and pipe address

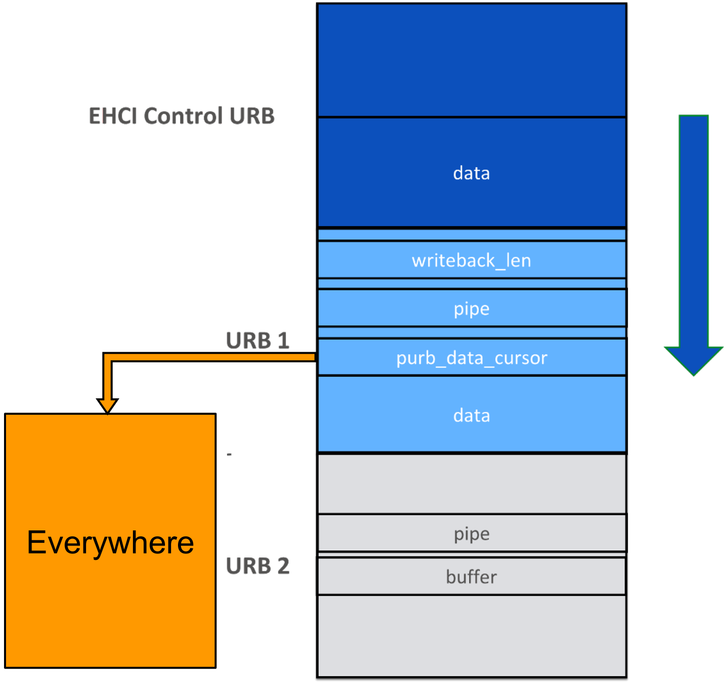


Arbitrary Address Read

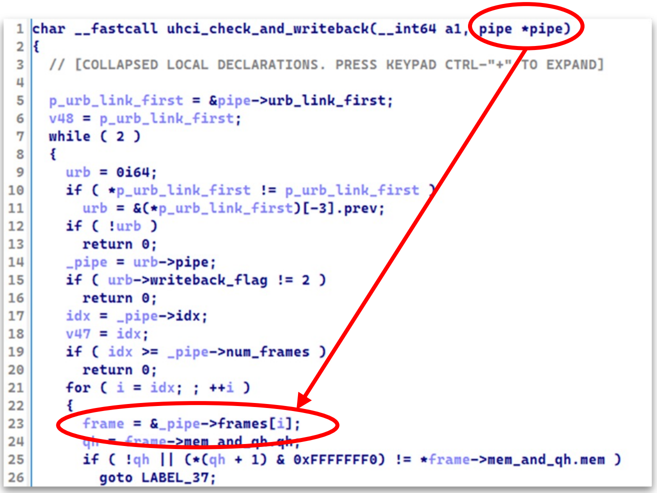
1. Allocate EHCI Control URB again

2. This time overwrite purb_data_cursor to any location

3. Read back URB1

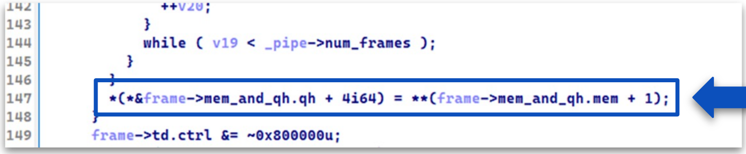


Arbitrary Address Write

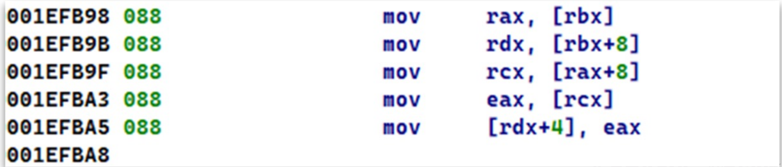


```
char __fastcall uhci_check_and_writeback(__int64 a1, pipe *pipe)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  p_urb_link_first = &pipe->urb_link_first;
  v48 = p_urb_link_first;
  while ( 2 )
  {
    urb = 0i64;
    if ( *p_urb_link_first != p_urb_link_first )
      urb = &(*p_urb_link_first)[-3].prev;
    if ( !urb )
      return 0;
    _pipe = urb->pipe;
    if ( urb->writeback_flag != 2 )
      return 0;
    idx = _pipe->idx;
    v47 = idx;
    if ( idx >= _pipe->num_frames )
      return 0;
    for ( i = idx; ; ++i )
    {
      frame = &_pipe->frames[i];
      qh = frame->mem_and_qh.qh;
      if ( !qh || (*(qh + 1) & 0xFFFFFFF0) != *frame->mem_and_qh.mem )
        goto LABEL_37;
```

- Write from a pointer in frame to another pointer in frame
- frame is a member in pipe
- We can fake the pipe in urb using out-of-bonds write

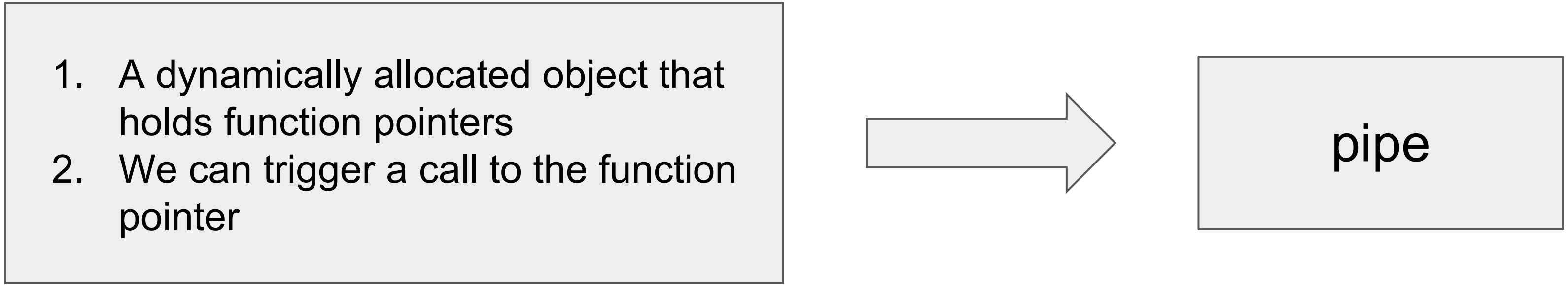


```
              ++v20;
          }
          while ( v19 < _pipe->num_frames );
        }
      }
      *(*&frame->mem_and_qh.qh + 4i64) = **(frame->mem_and_qh.mem + 1);
    }
    frame->td.ctrl &= ~0x800000u;
```

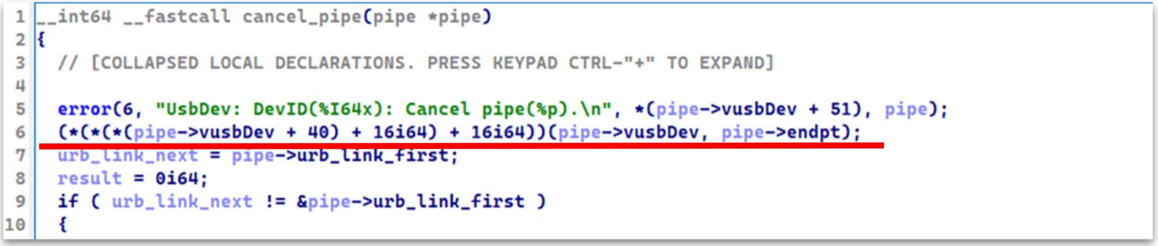


```
001EFB98 088                    mov     rax, [rbx]
001EFB9B 088                    mov     rdx, [rbx+8]
001EFB9F 088                    mov     rcx, [rax+8]
001EFBA3 088                    mov     eax, [rcx]
001EFBA5 088                    mov     [rdx+4], eax
001EFBA8
```

Control the RIP



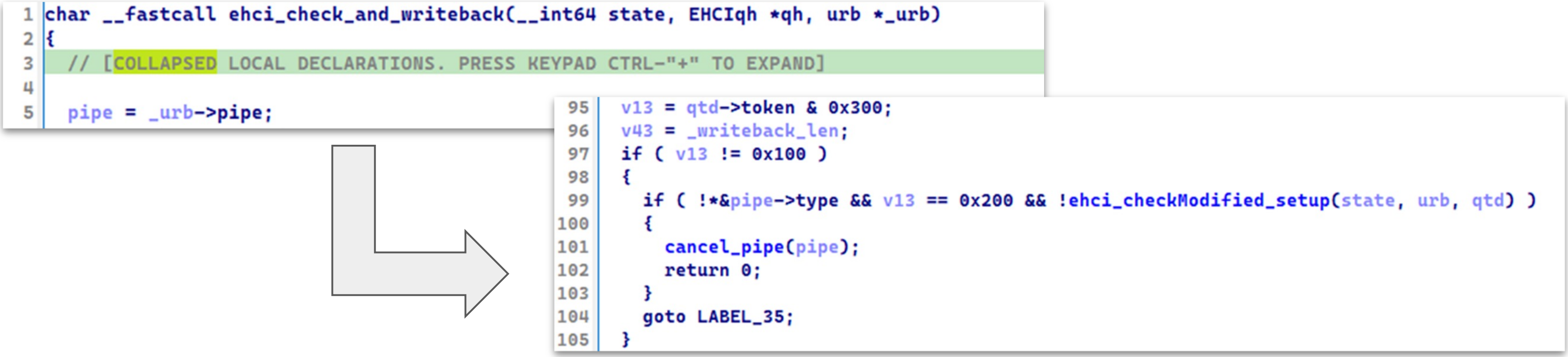
1. A dynamically allocated object that holds function pointers
2. We can trigger a call to the function pointer



```
__int64 __fastcall cancel_pipe(pipe *pipe)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  error(6, "UsbDev: DevID(%I64x): Cancel pipe(%p).\n", *(pipe->vusbDev + 51), pipe);
  (*(*(*(pipe->vusbDev + 40) + 16i64) + 16i64))(pipe->vusbDev, pipe->endpt);
  urb_link_next = pipe->urb_link_first;
  result = 0i64;
  if ( urb_link_next != &pipe->urb_link_first )
  {
```

Control the RIP: Path 1

- The pipe when calling cancel_pipe in ehci_check_and_writeback comes from the pointer of urb
- We can use out-of-bounds write to forge the urb->pipe to implement arbitrary address calls.



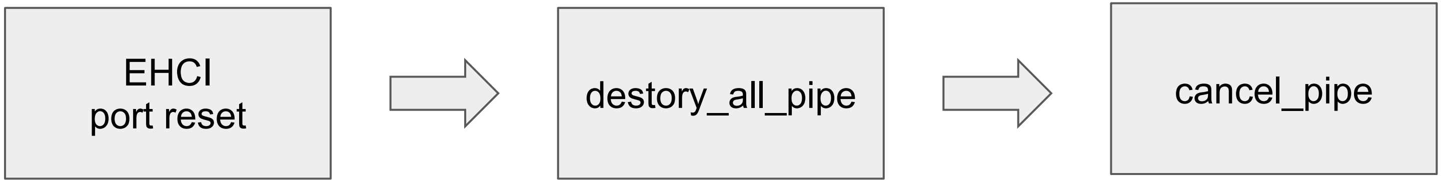
```
char __fastcall ehci_check_and_writeback(__int64 state, EHCIqh *qh, urb *_urb)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  pipe = _urb->pipe;
```

```
  v13 = qtd->token & 0x300;
  v43 = _writeback_len;
  if ( v13 != 0x100 )
  {
    if ( !*&pipe->type && v13 == 0x200 && !ehci_checkModified_setup(state, urb, qtd) )
    {
      cancel_pipe(pipe);
      return 0;
    }
    goto LABEL_35;
  }
```

Control the RIP

Path 2

- Fake a new pipe directly in vusbDev by arbitrary address write



Use Path 2 when we can’t reserve EHCI urb, although it needs more actions

What’s more? We need heap grooming

Heap spraying and grooming primitive: **SVGA_3D_CMD_SET_SHADER**

Allocate and free in large quantities, the heap size is sizeInBytes+8

```
svga_3d_cmd_define_gb_shader(shid, SVGA3D_SHADERTYPE_MIN, sizeInBytes);

svga_3d_cmd_bind_gb_shader(shid, mobid, 0);

svga_3d_cmd_set_shader(cid, SVGA3D_SHADERTYPE_MIN, shid);


svga_3d_cmd_destroy_gb_shader(shid);
```

https://census-labs.com/media/straightouttavmware-wp.pdf

CENSUS
IT Security Works

Straight outta VMware:
Modern exploitation of the SVGA device for
guest-to-host escape exploits

Black Hat Europe 2018

Zisis Sialveras
zisis@census-labs.com

CENSUS S.A., Stadiou 33, 10559, Athens, Greece. Tel: +30-2110-128355, Email: info@census-labs.com, Web: https://www.census-labs.com

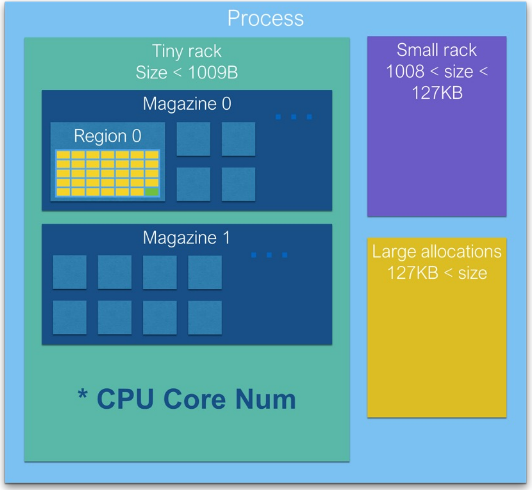
Try on the VMware Fusion!

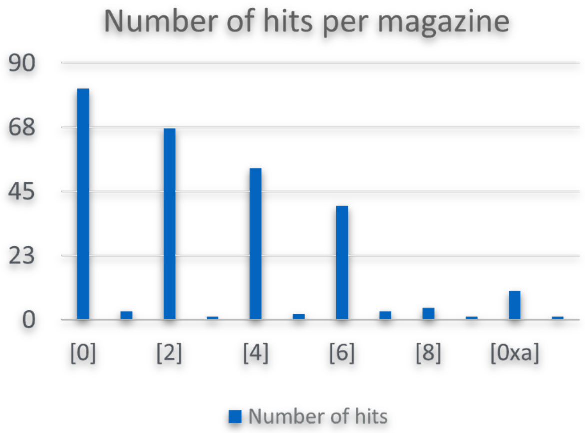
1 out-of-bounds read, 3 arbitrary address reads, and 2 arbitrary address writes

1. Heap grooming
2. Leak pipe address and heap address
3. Leak the program base address (pipe->dev)
4. Leak ehci state address (in .data)
5. Leak vusbdev address (in ehci state)
6. Write the upper 4 bytes of the fake pipe to vusbdev
7. Write the lower 4 bytes of the fake pipe to vusbdev
8. Trigger cancel pipe
9. Escape

Big problem. Magazine

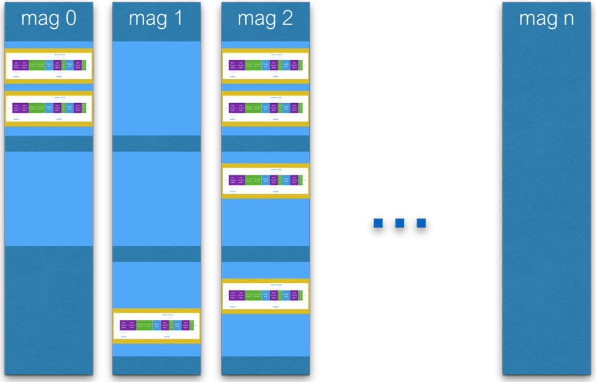
- MacOS's libmalloc uses magazines to manage heap blocks
- Each CPU core will have a unique corresponding magazine





Big problem. Magazine. How we deal with it

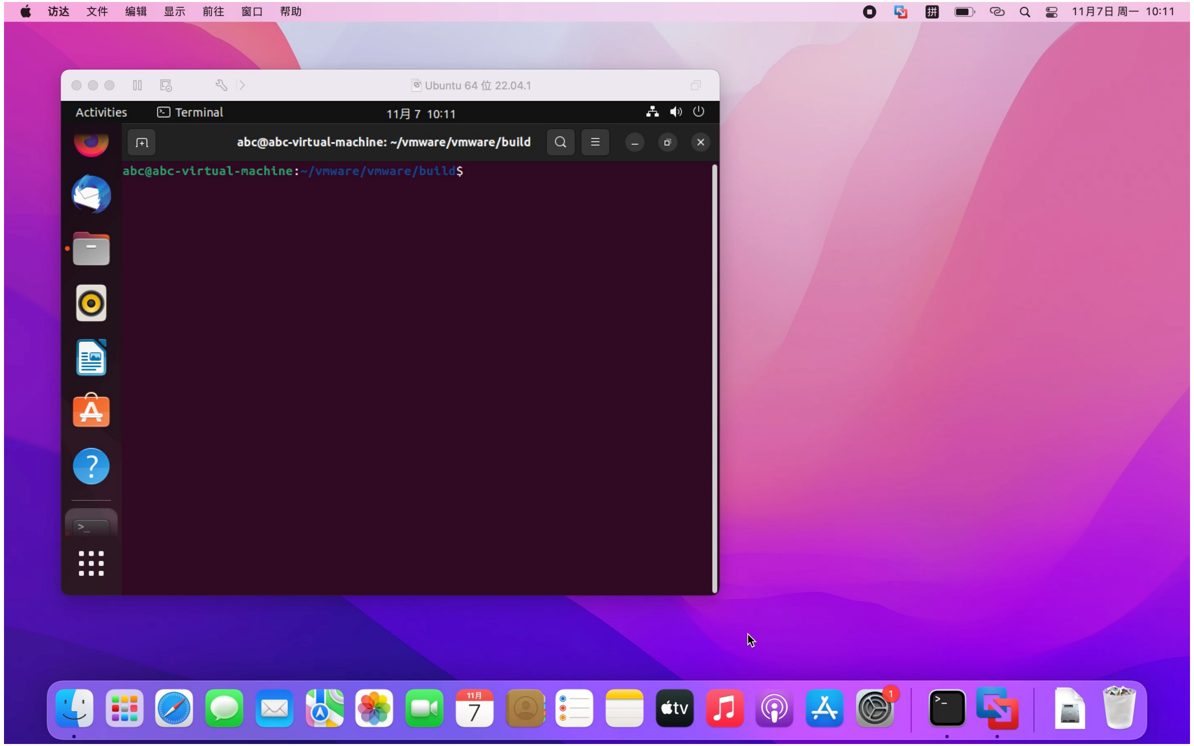
- Repeat the basic heap layout, and try to have at lease one layout on each magazine
- Try a large number of times for every step (place objects, do oob read…)
- How to ensure that all magazines are occupied?
  - Add sleep between each allocation X
  - Increase cpu's occupancy and try to increase cpu core switching X



- Remove sleep, speed up exploit
- Use a huge number of spray rounds (0x1000)

Success rate > 80%

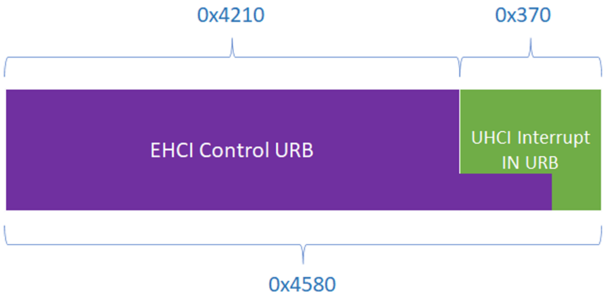
Demo


访达 文件 编辑 显示 前往 窗口 帮助
11月7日 周一 10:11
Ubuntu 64 位 22.04.1
Activities
Terminal
11月 7 10:11
abc@abc-virtual-machine: ~/vmware/vmware/build

```
abc@abc-virtual-machine:~/vmware/vmware/build$
```

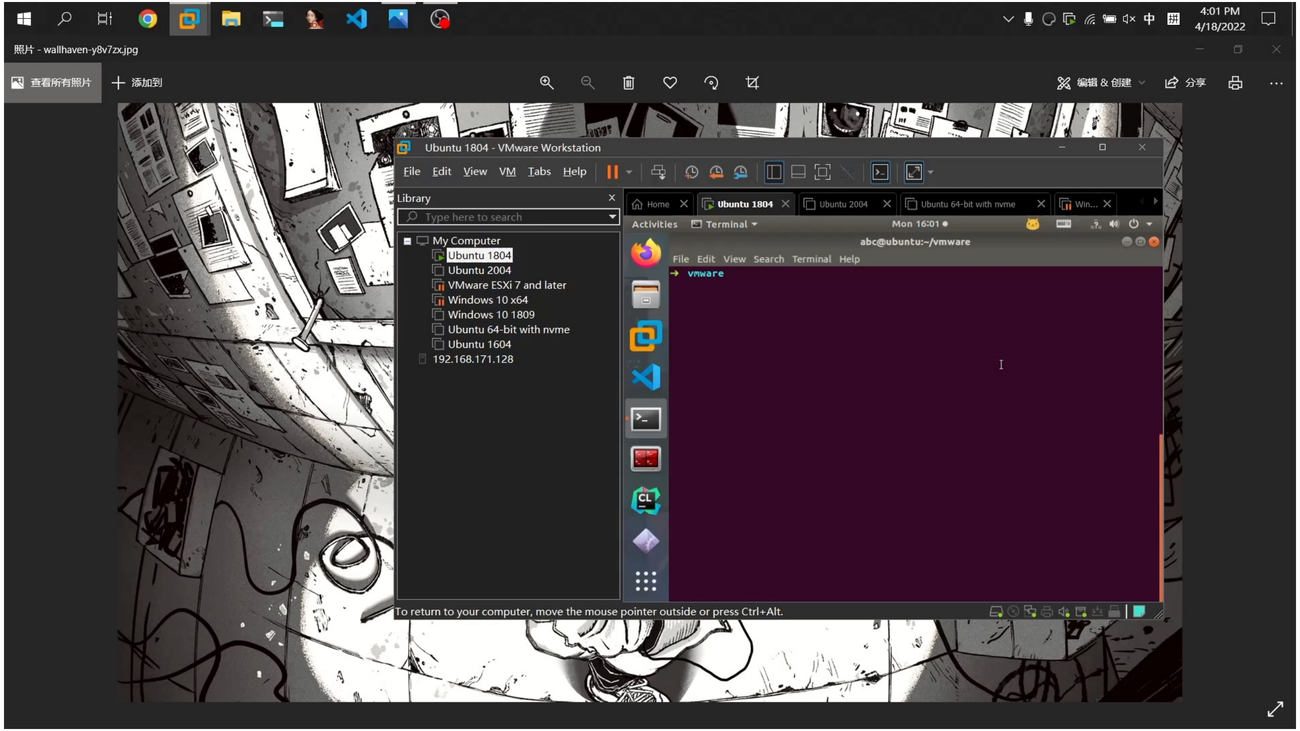
On VMware Workstation

- In the default configuration, there will be no device on the EHCI
  - Plug in a usb device to connect to ehci
- To avoid the randomization of LFH:
  - Use chunks larger than 0x4000
  - Select a size that has not been used by LFH when we can’t allocate larger than 0x4000



1. Leak heap address
2. Leak process base address
3. Leak the address of createProcessW (KERNEL32.dll)
4. Call WinExec

Demo



- My Computer
  - Ubuntu 1804
  - Ubuntu 2004
  - VMware ESXi 7 and later
  - Windows 10 x64
  - Windows 10 1809
  - Ubuntu 64-bit with nvme
  - Ubuntu 1604
- 192.168.171.128

```
→ vmware
```

To return to your computer, move the mouse pointer outside or press Ctrl+Alt.

On ESXi

- Same as Workstation, no default device on EHCI
- Similar to CentOS 7, use very old glibc-2.17 (2.28 after ESXi 8.0.2)
- Basically the same as on Fusion (No need to face magazines)
- Use GMR instead of Shader

Takeaways

- Where bugs have arisen with similar software, there may be new bugs
- When looking for exploit primitives, try to look for objects related to the vulnerability
- Virtual devices, especially USB-related devices, are now a popular attack surface

Questions?