

URB Excalibur: The New VMware All-Platform VM Escapes

Yuhao Jiang (@danis_jiang)
Xinlei Ying (@0x140ce)

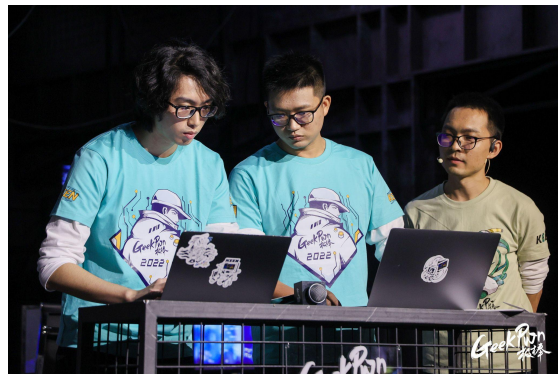


Who are we?

Security researchers at Ant Group Light-Year Security Lab

Escaped from virtual machine many times

Won the Pwnie Awards 🦄 at 2023



Yuhao Jiang
(@danis_jiang)

Xinlei Ying
(@0x140ce)



蚂蚁安全实验室
Ant Security Lab

蚂蚁光年
Ant Light-Year

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

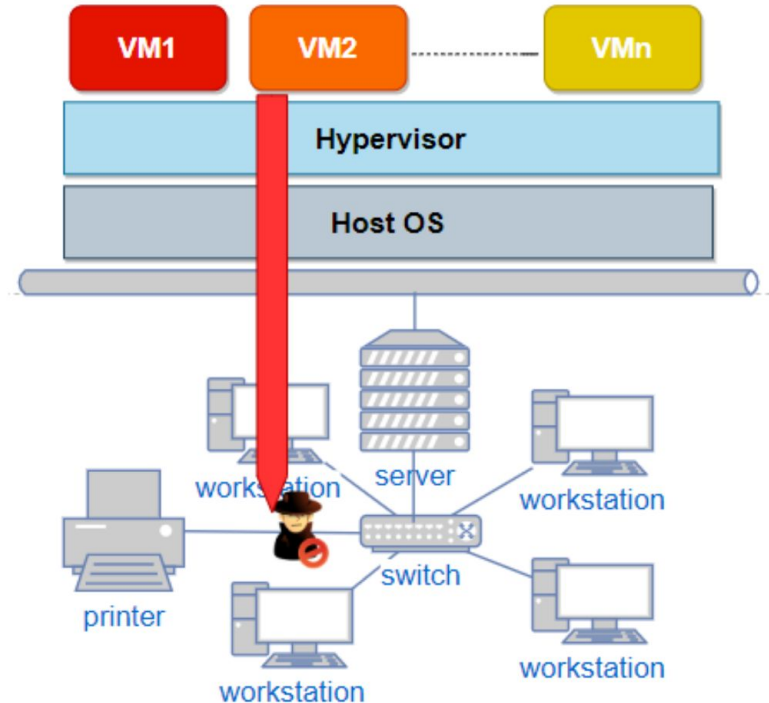


Fig. 6. Demonstration of VM Escape to Network.

VMware's Architecture

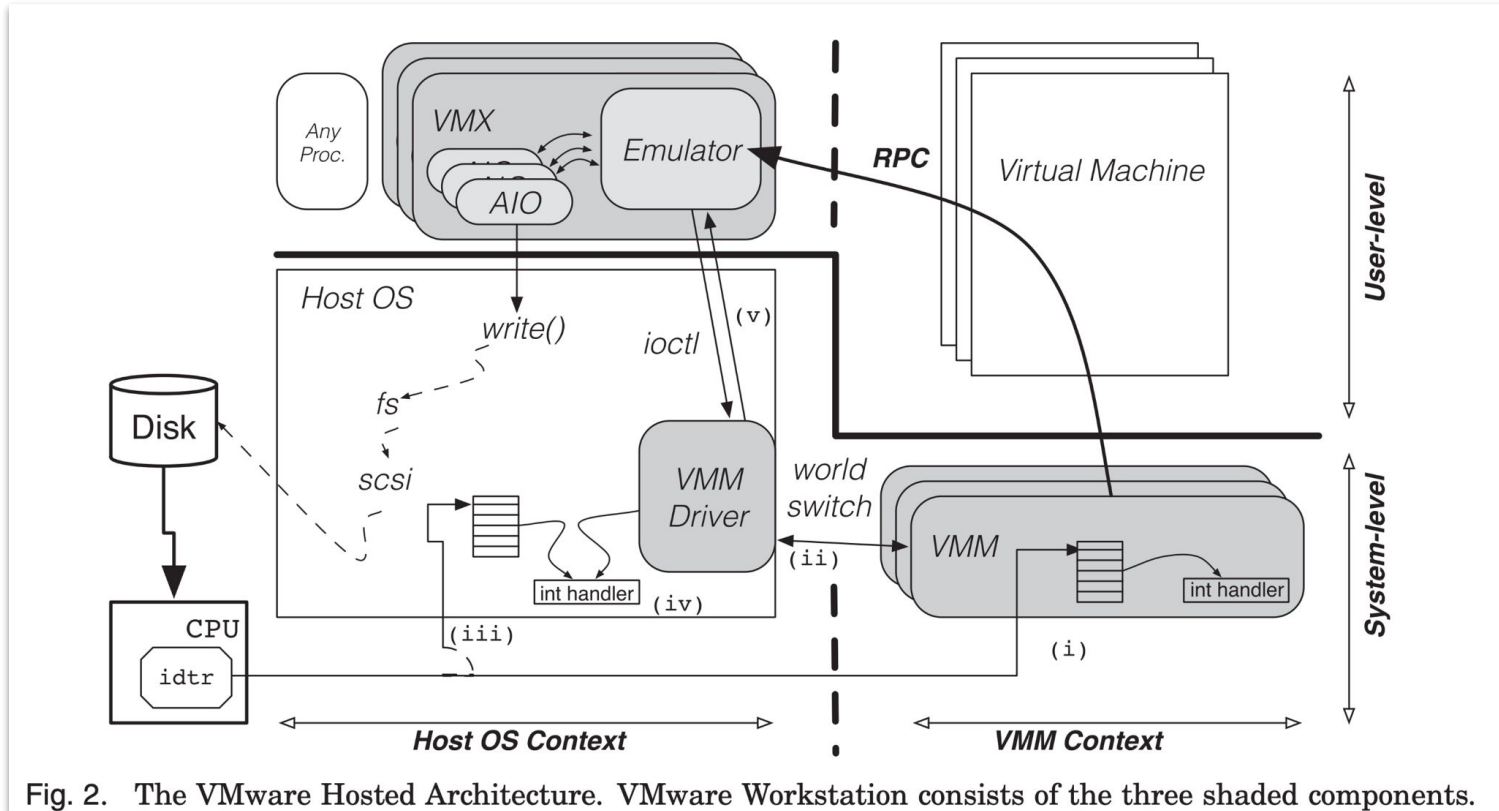


Fig. 2. The VMware Hosted Architecture. VMware Workstation consists of the three shaded components.

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

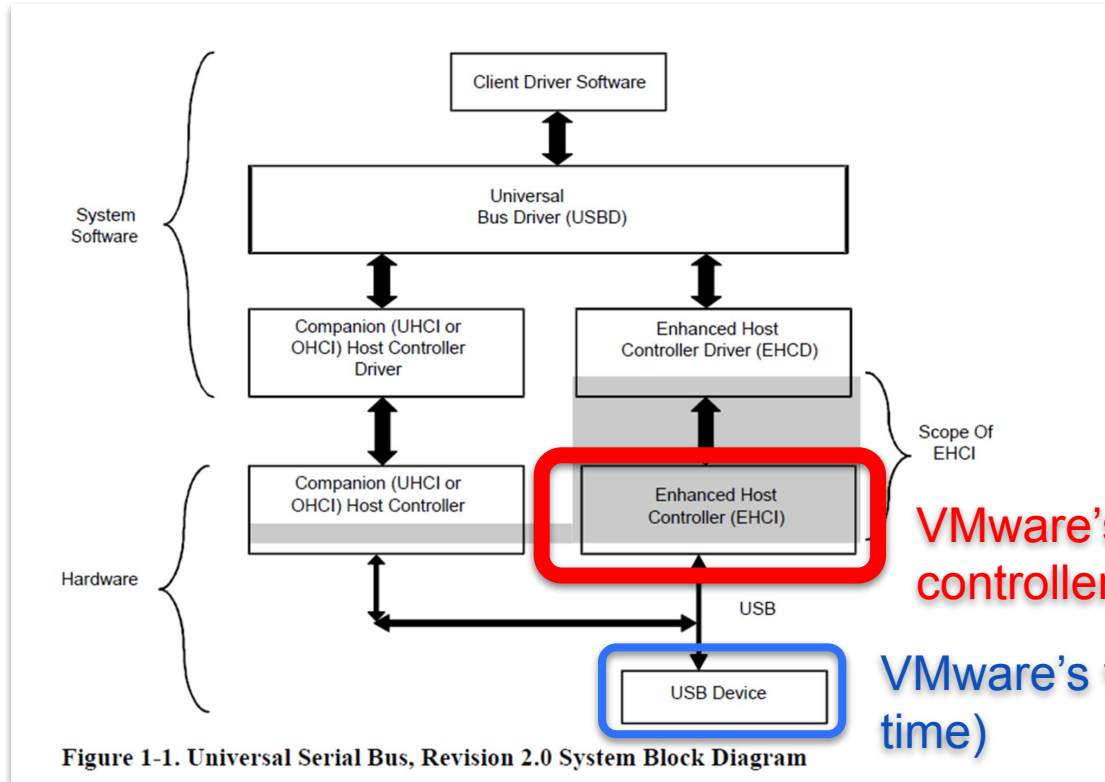
Start vulnerability discovery in VMware

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



EHCI / USB 2.0 Controller

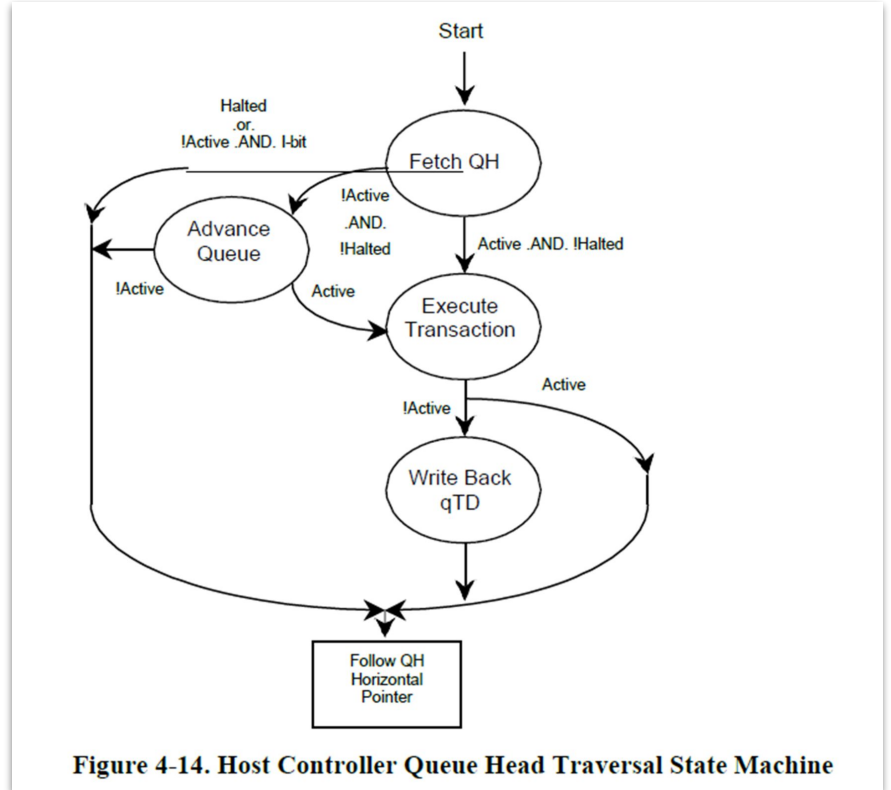
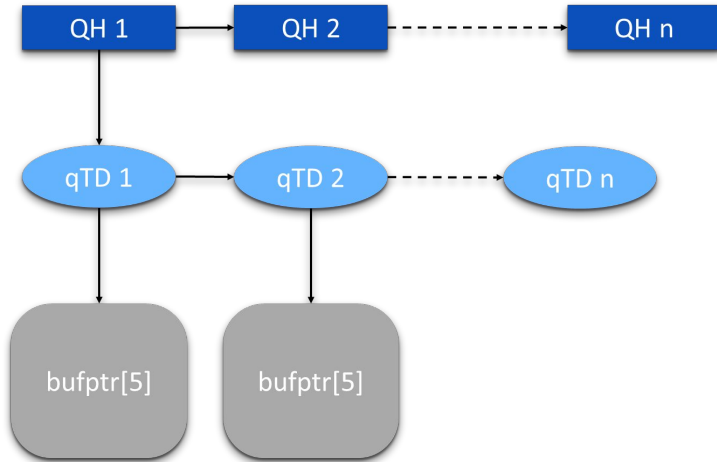


Figure 4-14. Host Controller Queue Head Traversal State Machine

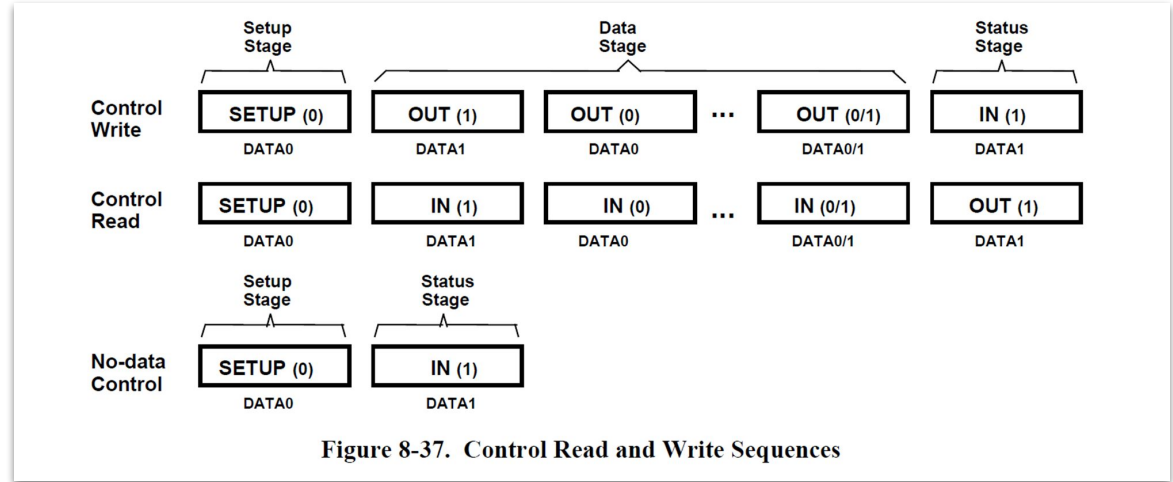
EHCI / USB 2.0 Controller

Endpoint/Pipe:

- Control
- Bulk
- Interrupt
- Isochronous

Token:

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



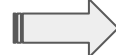
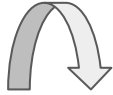
How the data flow

QEMU

ehci_state_fetchqh

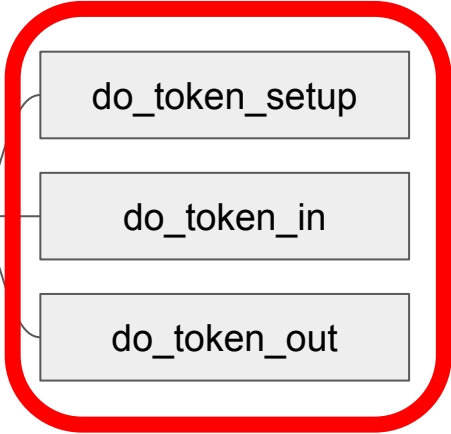


ehci_state_fetchqtd



usb_handle_packet

CVE-2020-14364

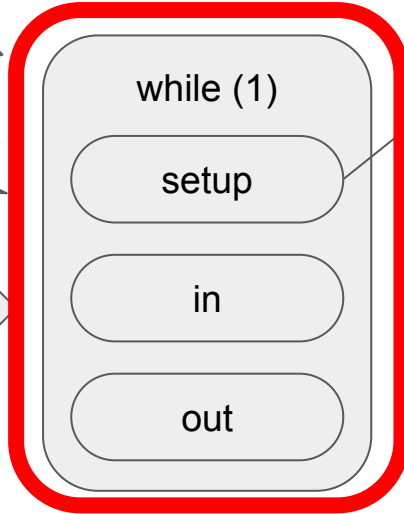


Could there be bugs here?



VMware

ehci_control_transfer



VUsb_NewUrb



urb_submit

urb_submit

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

urb's size =
0x98 + 8 + setup_len

```
1 void __fastcall ehci_control_transfer(__int64 state, pipe *pipe, EHCIqh *qh)
2 {
3     // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]
4
5     urb_link_first = pipe->urb_link_first;
6     p_next_qtd = &qh->next_qtd;
7     urb = 0i64;
8     v6 = 0;
```

SETUP qTD

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

```
157     urb = VUsb_NewUrb(pipe, 0, data_len);
158     purb_data_cursor = urb->purb_data_cursor;
159     urb->interrupt_pid = v14;
160     v14 = 1;
161     urb->num_packets = 1;
162     urb->datalen = data_len;
```

CVE-2022-31705

OUT qTD

```
178 else if ( urb && tbytes ) // OUT
179 {
180     if ( (int)tbytes > setup_len )
181     {
182         _mm_lfence();
183         Warning("EHCI: setup data packet over run %d %d\n", tbytes, (unsigned int)setup_len);
184         tbytes = 0;
185         if ( setup_len > 0 )
186             tbytes = setup_len;
187     }
188     if ( !EhciReadTDBuffer(state, urb->purb_data_cursor, qtd, tbytes, 0 )
189         goto LABEL_76;
190     urb->purb_data_cursor += (int)tbytes;
191     v17 = qtd->token;
192 }
```

IN qTD

```
193 setup_len -= tbytes; // IN & OUT
194 if ( ((v17 >> 8) & 3) == v15 )
195 {
196     if ( tbytes )
197         Warning("EHCI: Unexpected status packet size %d\n", tbytes);
198     goto LABEL_73;
199 }
```

Where is the BUG?

```
200 LABEL_59: // next qtd
201 LODWORD(v34) = qtd->next;
202 HIWORD(v34) = *(DWORD *)(&state + 1880 + 32i64);
203 if ( (qtd->next & 1) != 0 || (v34 & 0xFFFFFFFFFFFFFFF0ui64) == 0 )
204     goto LABEL_69;
205 mem = get_mem(state + 1328, v34 & 0xFFFFFFFFFFFFFFF0ui64, *(unsigned int *)(&state + 1868), 8);
206 if ( !mem )
207     break;
```


CVE-2022-31705

OUT qTD

Check if
qTD->tbytes is
valid

IN qTD

Where is the BUG?

```
178 else if ( urb && tbytes ) // OUT
179
180 if ( (int)tbytes > setup_len )
181 {
182     _mm_lfence();
183     Warning("EHCI: setup data packet over run %d %d\n", tbytes, (unsigned int)setup_len);
184     tbytes = 0;
185     if ( setup_len > 0 )
186         tbytes = setup_len;
187 }
188 if ( !EhciReadTDBuffer(state, urb->purb_data_cursor, qtd, tbytes, 0) )
189     goto LABEL_76;
190 urb->purb_data_cursor += (int)tbytes;
191 v17 = qtd->token;
192 }
```

```
193 setup_len -= tbytes; // IN & OUT
194 if ( ((v17 >> 8) & 3) == v15 )
195 {
196     if ( tbytes )
197         Warning("EHCI: Unexpected status packet size %d\n", tbytes);
198     goto LABEL_73;
199 }
```

```
200 LABEL_59: // next qtd
201 LODWORD(v34) = qtd->next;
202 HIDWORD(v34) = *(DWORD *)(*(DWORD *)(state + 1880) + 32i64);
203 if ( (qtd->next & 1) != 0 || (v34 & 0xFFFFFFFFFFFFFFF0ui64) == 0 )
204     goto LABEL_69;
205 mem = get_mem(state + 1328, v34 & 0xFFFFFFFFFFFFFFF0ui64, *(unsigned int *)(state + 1868), 8);
206 if ( !mem )
207     break;
```

CVE-2022-31705

OUT qTD

Check if
qTD->tbytes is
valid

IN qTD

```
178 else if ( urb && tbytes ) // OUT
179
180 if ( (int)tbytes > setup_len )
181 {
182     _mm_lfence();
183     Warning("EHCI: setup data packet over run %d %d\n", tbytes, (unsigned int)setup_len);
184     tbytes = 0;
185     if ( setup_len > 0 )
186         tbytes = setup_len;
187 }
188 if ( !EhciReadTDBuffer(state, urb->purb_data_cursor, qtd, tbytes, 0) )
189     goto LABEL_76;
190 urb->purb_data_cursor += (int)tbytes;
191 v17 = qtd->token;
192 }
```

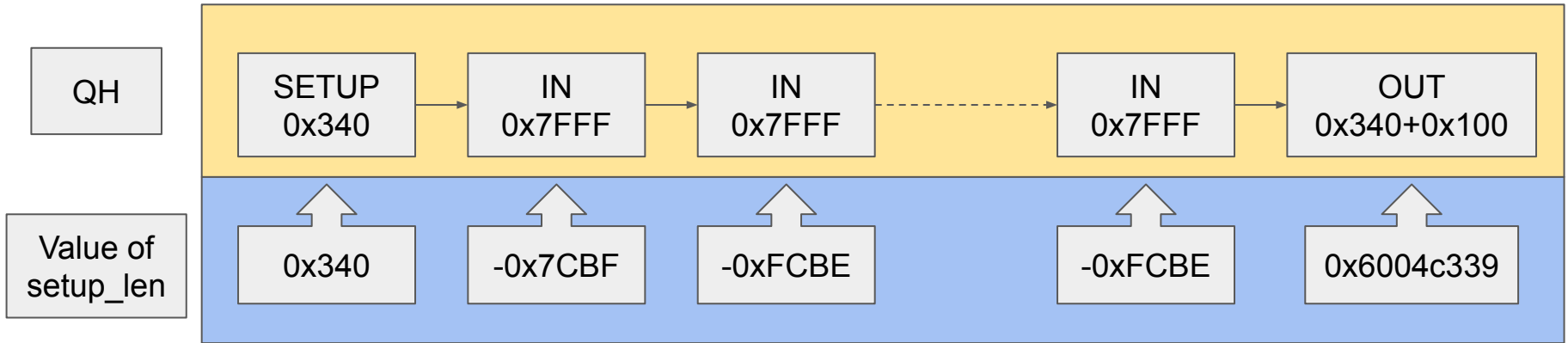
NO CHECK!
Directly subtract
tbytes from setup_len

```
193 setup_len -= tbytes; // IN & OUT
194 if ( ((v17 >> 8) & 3) == v15 )
195 {
196     if ( tbytes )
197         Warning("EHCI: Unexpected status packet size %d\n", tbytes);
198     goto LABEL_73;
199 }
```

```
200 LABEL_59: // next qtd
201 LODWORD(v34) = qtd->next;
202 HIDWORD(v34) = *(DWORD *)(*(_DWORD *) (state + 1880) + 32i64);
203 if ( (qtd->next & 1) != 0 || (v34 & 0xFFFFFFFFFFFFFFF0ui64) == 0 )
204     goto LABEL_69;
205 mem = get_mem(state + 1328, v34 & 0xFFFFFFFFFFFFFFF0ui64, *(unsigned int *) (state + 1868), 8);
206 if ( !mem )
207     break;
```

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

How VMware fixed it

OUT qTD

```
178 else if ( urb && tbytes ) // OUT
179 {
180     if ( (int)tbytes > setup_len )
181     {
182         _mm_lfence();
183         Warning("EHCI: setup data packet over run %d %d\n", tbytes, (unsigned int)setup_len);
184         tbytes = 0;
185         if ( setup_len > 0 )
186             tbytes = setup_len;
187     }
188     if ( !EhciReadTDBuffer(state, urb->purb_data_cursor, qtd, tbytes, 0 )
189         goto LABEL_76;
190     urb->purb_data_cursor += (int)tbytes;
191     v17 = qtd->token;
192 }
```

IN qTD

```
191     if ( (int)tbytes > setup_len )
192     {
193         _mm_lfence();
194         error(99, "EHCI: Setup data packet over run %d %d\n", tbytes, (unsigned int)setup_len);
195         tbytes = 0;
196         if ( setup_len > 0 )
197             tbytes = setup_len;
198     }
```

```
193     setup_len -= tbytes; // IN & OUT
194     if ( ((v17 >> 8) & 3) == v15 )
195     {
196         if ( tbytes )
197             Warning("EHCI: Unexpected status packet size %d\n", tbytes);
198         goto LABEL_73;
199     }
```

```
200 LABEL_59: // next qtd
201     LODWORD(v34) = qtd->next;
202     HIWORD(v34) = *(DWORD *)(*C_QWORD *)(state + 1880) + 32i64;
203     if ( (qtd->next & 1) != 0 || (v34 & 0xFFFFFFFFFFFFFFF0ui64) == 0 )
204         goto LABEL_69;
205     mem = get_mem(state + 1328, v34 & 0xFFFFFFFFFFFFFFF0ui64, *(unsigned int *)(state + 1868), 8);
206     if ( !mem )
207         break;
```

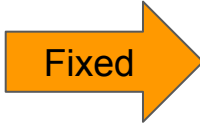
What else did we find?

BUG 1: Out-of-bounds read vulnerability

- Pipe type confusion (Control \Leftrightarrow ISOC)
- Handle urb incorrectly

```
34 pipe = *(v10 + 8 * (v9 & 0xF | (16 * ((v9 >> 7) & 1))) + 0x10);
35 if ( !pipe )
36     return 0;
37 }
38 if ( !sub_1F44B0(state, pipe, index, &purb_data_cursor, &cur_packet_len) )
39     return 1;
```

```
34 if ( (urb->cur_packet - urb - 0x98) / 12 < urb->num_packets )
35 {
36     cur_packet = urb->cur_packet;
37     _cur_packet_len = cur_packet_len;
38     *cur_packet_len = cur_packet->len;
39     *purb_data_cursor = urb->purb_data_cursor;
40     v15 = 8 * (1 << (10 - ((*state + 1880) +
41     if ( (v15 & (index - cur_packet->maybe_id))
42         goto LABEL_21;
43     urb->purb_data_cursor += cur_packet->total_transfer_length;
44     pipe->all_size -= urb->datalen;
45     --pipe->num_packets;
46     ++urb->cur_packet;
47     if ( cur_packet->maybe_id == index )
48         return 1;
```



Always be 1 in control pipe's urb

```
37 if ( !v10 || (pipe = *(pipe **)(v10 + 8 * (v9 & 0xF | (16 *
38 {
39     error(99, "EHCI: Nonexistent pipe for device %d, endpt %x.
40     return 0;
41 }
42 if ( *(DWORD *)&pipe->type != 1 )
43     return 0;
44 }
```

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.

Fixed

```
59 urb->pipe_type = type;
60 urb->endpt = pipe->endpt;
61 urb->hci_state = (_QWORD *)((_QWORD *)pipe->vusbDev + 40);
62 urb->interrupt_pid = 0;
63 urb->streamctx_struct2058 = 0i64;
64 if ( !type && buffer )
65     memset(buffer, 0, _datalen);
66 urb->urb_link.next = &urb->urb_link;
67 urb->urb_link.prev = &urb->urb_link;
```

Exploit Development

The problem

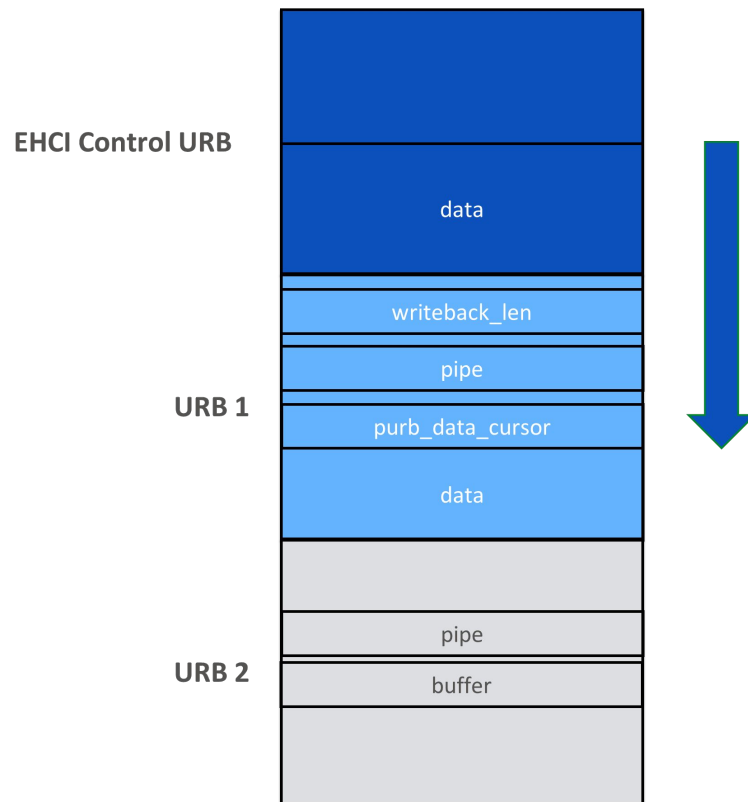
- **[Again]** Closed-source
- No public exploit code and rarely disclosed exploit flow
- Most of past exploit primitives have been patched
- Few code paths that can be controlled with in the guest OS.

Some patches for old primitives

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

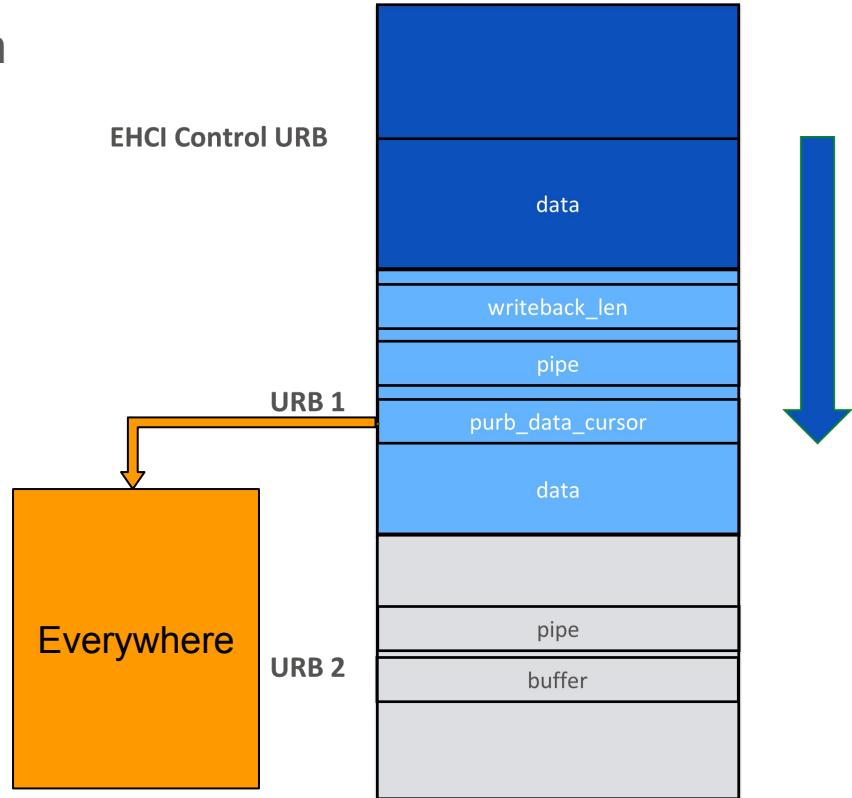
Out-of-bounds write -> Out-of-bounds Read

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 `pub_data_cursor` to any location
3. Read back URB1



Arbitrary Address Write

```
1 char __fastcall uhci_check_and_writeback(__int64 a1, pipe *pipe)
2 {
3     // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]
4
5     p_urb_link_first = &pipe->urb_link_first;
6     v48 = p_urb_link_first;
7     while ( 2 )
8     {
9         urb = 0i64;
10        if ( *p_urb_link_first != p_urb_link_first )
11            urb = &(*p_urb_link_first)[-3].prev;
12        if ( !urb )
13            return 0;
14        _pipe = urb->pipe;
15        if ( urb->writeback_flag != 2 )
16            return 0;
17        idx = _pipe->idx;
18        v47 = idx;
19        if ( idx >= _pipe->num_frames )
20            return 0;
21        for ( i = idx; ; ++i )
22        {
23            frame = &_pipe->frames[i];
24            qh = frame->mem_and_qh.qh;
25            if ( !qh || (*(qh + 1) & 0xFFFFFFFF) != *frame->mem_and_qh.mem )
26                goto LABEL_37;
```



```
142         ++v20;
143     }
144     while ( v19 < _pipe->num_frames );
145 }
146 }
147 }
148 }
149 }
150 }
151 }
152 }
153 }
154 }
155 }
156 }
157 }
158 }
159 }
160 }
161 }
162 }
163 }
164 }
165 }
166 }
167 }
168 }
169 }
170 }
171 }
172 }
173 }
174 }
175 }
176 }
177 }
178 }
179 }
180 }
181 }
182 }
183 }
184 }
185 }
186 }
187 }
188 }
189 }
190 }
191 }
192 }
193 }
194 }
195 }
196 }
197 }
198 }
199 }
200 }
201 }
202 }
203 }
204 }
205 }
206 }
207 }
208 }
209 }
210 }
211 }
212 }
213 }
214 }
215 }
216 }
217 }
218 }
219 }
220 }
221 }
222 }
223 }
224 }
225 }
226 }
227 }
228 }
229 }
230 }
231 }
232 }
233 }
234 }
235 }
236 }
237 }
238 }
239 }
240 }
241 }
242 }
243 }
244 }
245 }
246 }
247 }
248 }
249 }
250 }
251 }
252 }
253 }
254 }
255 }
256 }
257 }
258 }
259 }
260 }
261 }
262 }
263 }
264 }
265 }
266 }
267 }
268 }
269 }
270 }
271 }
272 }
273 }
274 }
275 }
276 }
277 }
278 }
279 }
280 }
281 }
282 }
283 }
284 }
285 }
286 }
287 }
288 }
289 }
290 }
291 }
292 }
293 }
294 }
295 }
296 }
297 }
298 }
299 }
300 }
301 }
302 }
303 }
304 }
305 }
306 }
307 }
308 }
309 }
310 }
311 }
312 }
313 }
314 }
315 }
316 }
317 }
318 }
319 }
320 }
321 }
322 }
323 }
324 }
325 }
326 }
327 }
328 }
329 }
330 }
331 }
332 }
333 }
334 }
335 }
336 }
337 }
338 }
339 }
340 }
341 }
342 }
343 }
344 }
345 }
346 }
347 }
348 }
349 }
350 }
351 }
352 }
353 }
354 }
355 }
356 }
357 }
358 }
359 }
360 }
361 }
362 }
363 }
364 }
365 }
366 }
367 }
368 }
369 }
370 }
371 }
372 }
373 }
374 }
375 }
376 }
377 }
378 }
379 }
380 }
381 }
382 }
383 }
384 }
385 }
386 }
387 }
388 }
389 }
390 }
391 }
392 }
393 }
394 }
395 }
396 }
397 }
398 }
399 }
400 }
401 }
402 }
403 }
404 }
405 }
406 }
407 }
408 }
409 }
410 }
411 }
412 }
413 }
414 }
415 }
416 }
417 }
418 }
419 }
420 }
421 }
422 }
423 }
424 }
425 }
426 }
427 }
428 }
429 }
430 }
431 }
432 }
433 }
434 }
435 }
436 }
437 }
438 }
439 }
440 }
441 }
442 }
443 }
444 }
445 }
446 }
447 }
448 }
449 }
450 }
451 }
452 }
453 }
454 }
455 }
456 }
457 }
458 }
459 }
460 }
461 }
462 }
463 }
464 }
465 }
466 }
467 }
468 }
469 }
470 }
471 }
472 }
473 }
474 }
475 }
476 }
477 }
478 }
479 }
480 }
481 }
482 }
483 }
484 }
485 }
486 }
487 }
488 }
489 }
490 }
491 }
492 }
493 }
494 }
495 }
496 }
497 }
498 }
499 }
500 }
```

- 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

```
001EFB98 038      mov     rax, [rbx]
001EFB9B 038      mov     rdx, [rbx+8]
001EFB9F 038      mov     rcx, [rax+8]
001EFBA3 038      mov     eax, [rcx]
001EFBA5 038      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



pipe

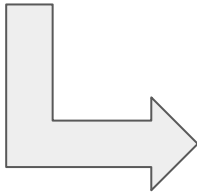
```
1 __int64 __fastcall cancel_pipe(pipe *pipe)
2 {
3     // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]
4
5     error(6, "UsbDev: DevID(%I64x): Cancel pipe(%p).\n", *(pipe->vusbDev + 51), pipe);
6     (**(*(pipe->vusbDev + 40) + 16i64) + 16i64))(pipe->vusbDev, pipe->endpt);
7     urb_link_next = pipe->urb_link_first;
8     result = 0i64;
9     if ( urb_link_next != &pipe->urb_link_first )
10    {
```

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.

```
1 char __fastcall ehci_check_and_writeback(__int64 state, EHCIqh *qh, urb *_urb)
2 {
3     // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]
4
5     pipe = _urb->pipe;
```

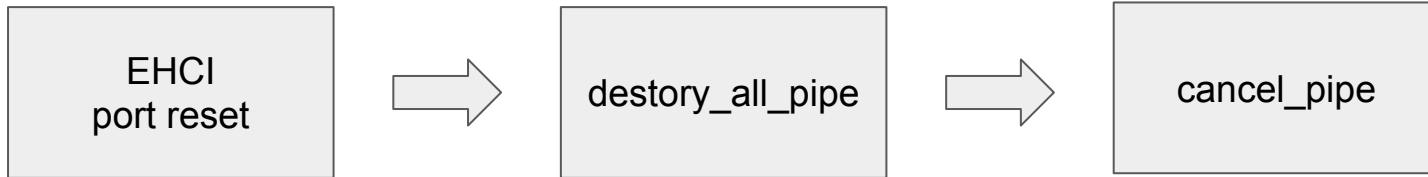


```
95     v13 = qtd->token & 0x300;
96     v43 = _writeback_len;
97     if ( v13 != 0x100 )
98     {
99         if ( !*&pipe->type && v13 == 0x200 && !ehci_checkModified_setup(state, urb, qtd) )
100         {
101             cancel_pipe(pipe);
102             return 0;
103         }
104         goto LABEL_35;
105     }
```

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`

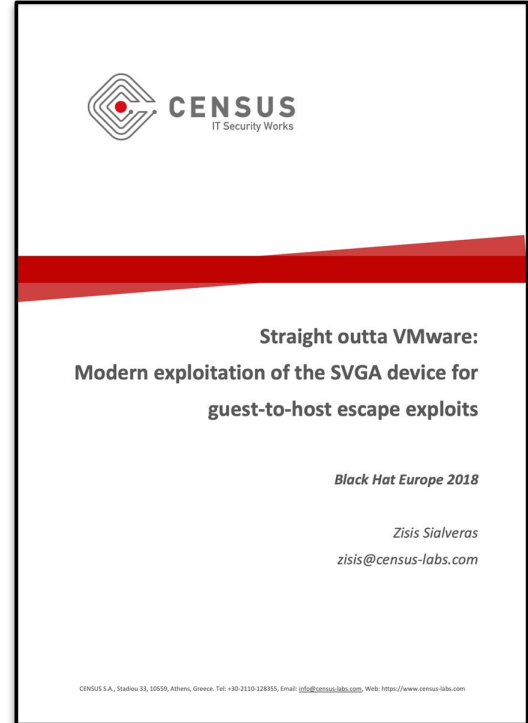
```
svga_3d_cmd_define_gb_shader(shid, SVGA3D_SHADERTYPE_MIN, sizeInBytes);
```

```
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>



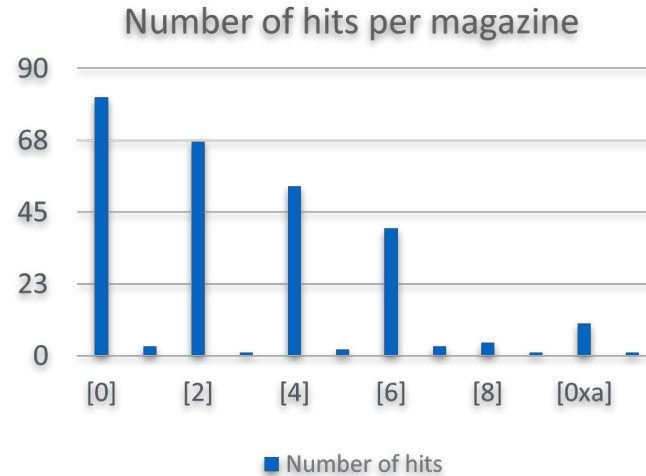
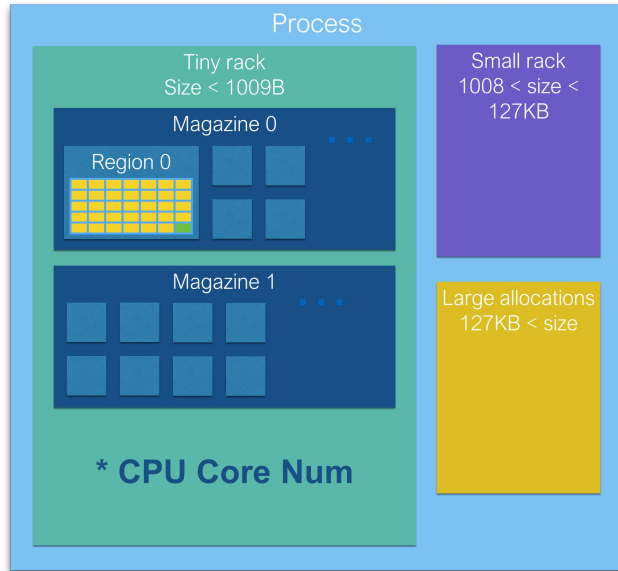
Try on the VMware Fusion!

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

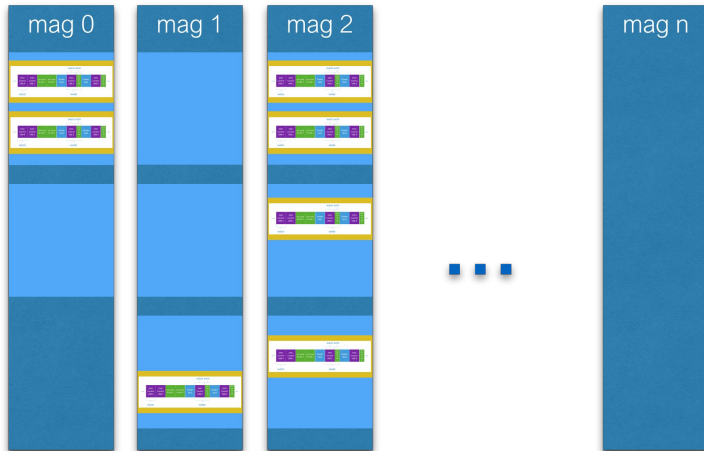
Big problem. Magazine

- 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 least 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)

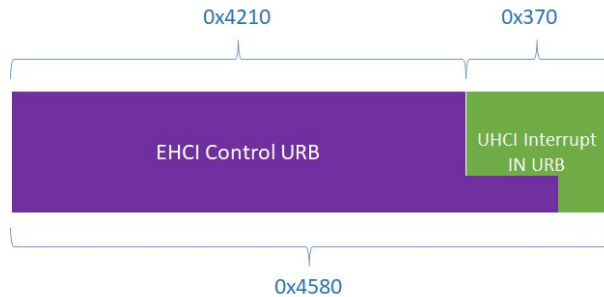
Success rate > 80%

Demo



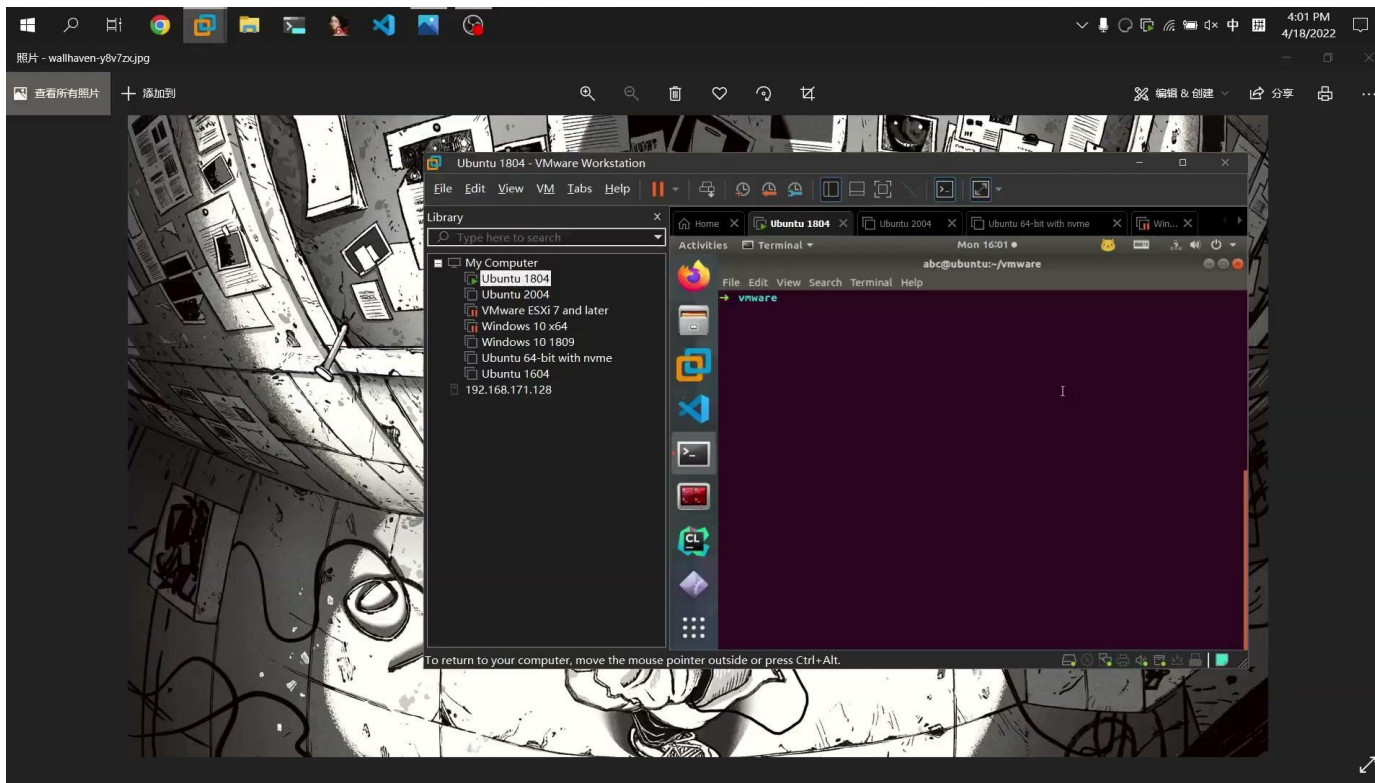
On VMware Workstation

- 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



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?