VMware Workstation: Escaping via a New Route - Virtual Bluetooth

Nguyen Hoang Thach (@hi_im_d4rkn3ss)

STAR Labs SG Pte. Ltd.

About me

- Nguyen Hoang Thach (@hi_im_d4rkn3ss)
- Senior Security Researcher at STAR Labs SG Pte. Ltd.
- Focusing on Browser / Virtual Machine / IOT bug hunting.
- Participated in multiple Pwn2Own events.
- Speak at conferences: POC2022

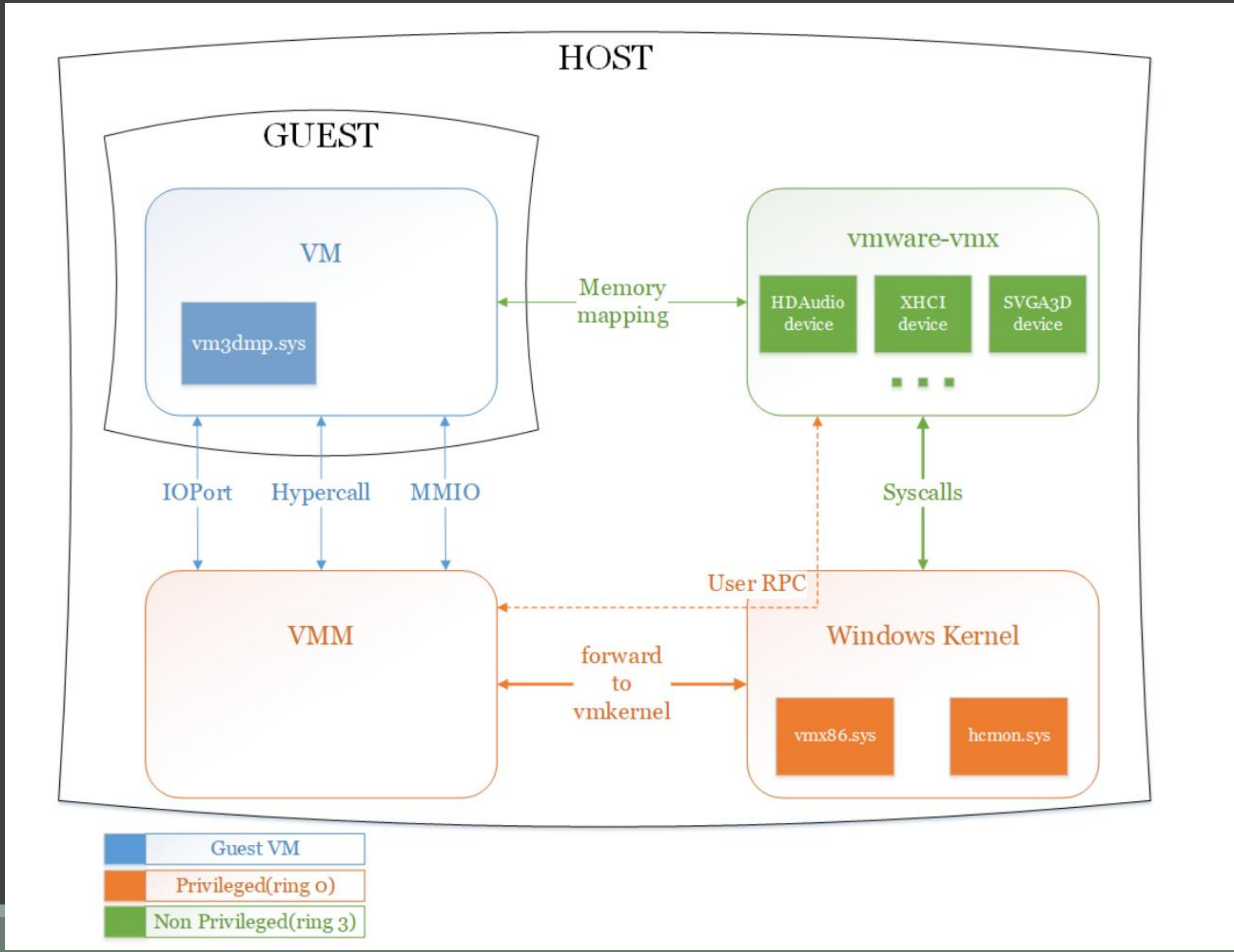
Agenda

- Backgrounds
- Virtual Bluetooth Device
- Infoleak bugs
- 2023 bug: Stack Overflow
- 2024 bug: Use-after-free
- Summary

Backgrounds

Backgrounds

Vmware Workstation Architecture (Windows host)



Credit: https://www.synacktiv.com/sites/default/files/2020-10/Speedpwning_VMware_Workstation.pdf

Backgrounds

Vmware Workstation Architecture (Windows host)

- `vmware-vmx.exe` binary is the main binary which run guest OS. It implements many emulated devices (RAM, CPU, Disk, Network, Audio, USB, ...)

- It communicates to host OS (Windows) via Hypercall (HyperV enabled). In case HyperV is disabled, it has separated driver to communicated to host OS

- It communicates to guest OS via MMIO / IOPort / Socket

Backgrounds

Attack surface : Device Emulation

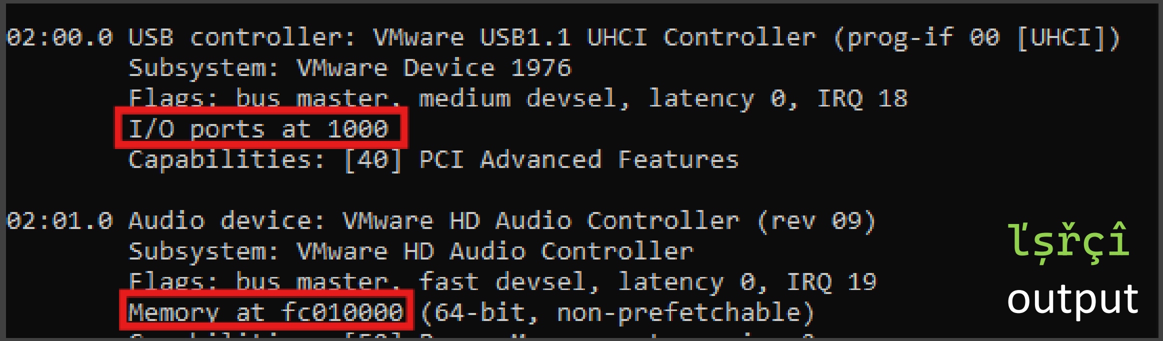
- 2024:
  - **VBluetooth** (CVE-2024-22267, CVE-2024-22269, demonstrated in Pwn2Own)
  - **Host Guest File Sharing (HGFS)** (CVE-2024-22270),
  - **Shader** (CVE-2024-22268)
  - Storage Controller (CVE-2024-22273)
- 2023:
  - **VBluetooth** (CVE-2023-20869, CVE-2023-20870, demonstrated in Pwn2Own, CVE-2023-34044)
  - **UHCI** (CVE-2024-22255, CVE-2024-22253, demonstrated in Tianfu Cup)
  - **XHCI** (CVE-2024-22252, demonstrated in Tianfu Cup)
  - **SCSI** (CVE-2023-20872)
- 2022:
  - **ThinPrint** (CVE-2022-22938)
  - **CD-ROM** (CVE-2021-22045)
  - **UHCI** (CVE-2021-22041, demonstrated in Tianfu Cup)
  - **XHCI** (CVE-2021-22040, demonstrated in Tianfu Cup)
  - **EHCI** (CVE-2022-31705, demonstrated in Geekpwn)

=> Most bugs were in Device Emulation Implementation

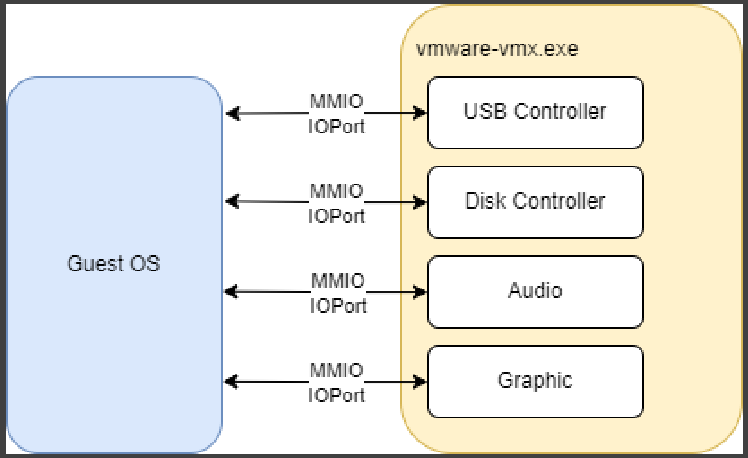
Backgrounds

Attack surface : Device Emulation

- Most of Emulated Devices is communicated to guest OS via MMIO / IOPort



- For better understand, refer to Specification and other opensource virtual machine source code (QEMU / VirtualBox)

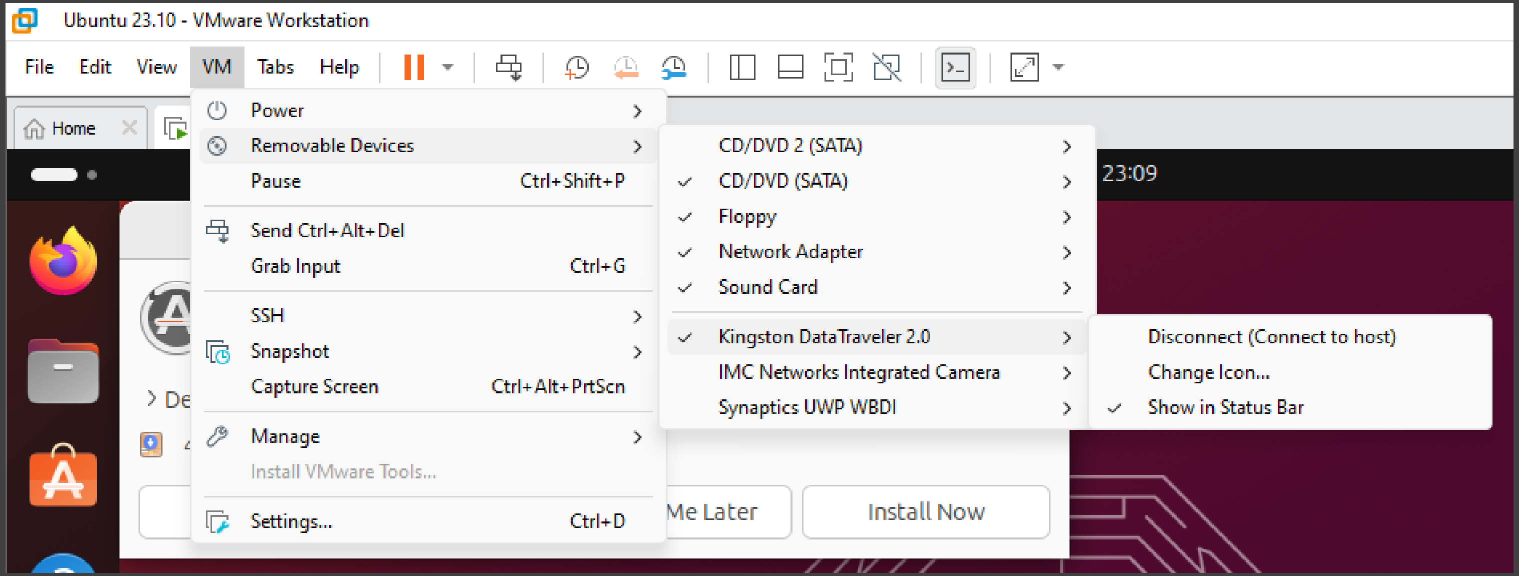


Virtual Bluetooth Device

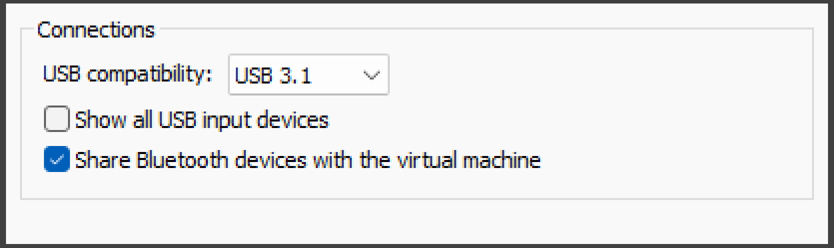
Virtual Bluetooth Device

USB Controller

- Connect external USB device to Guest OS



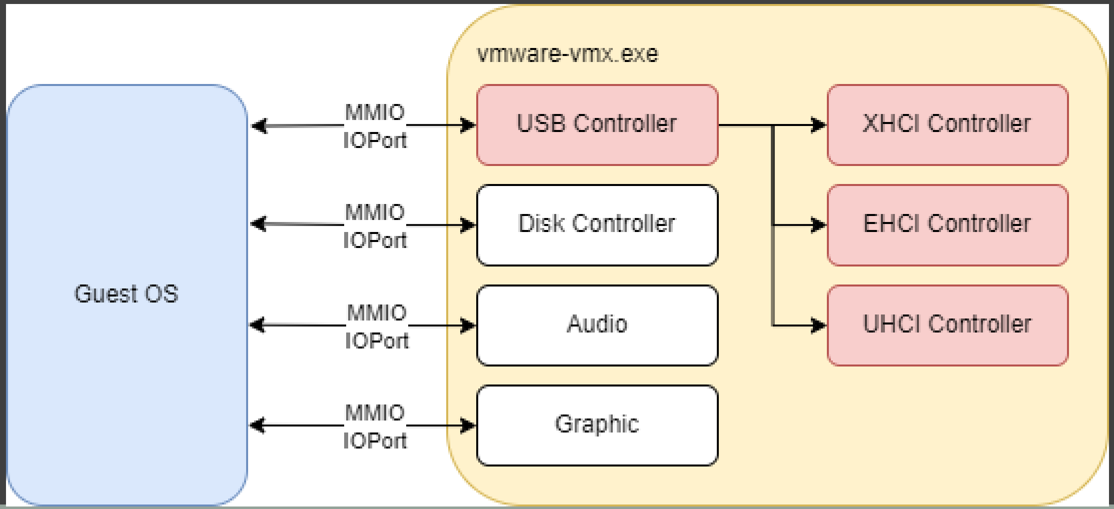
- USB Controller config



Virtual Bluetooth Device

USB Controller

- In recent years, there are many exploitable bugs found in USB Controller.
- Windows guest: UHCI (usb 1.0) + EHCI (usb 2.0) + XHCI (usb 3.1) are enabled by default
- Linux guest: UHCI (usb 1.0) + XHCI (usb 3.1) are enabled by default
- Specification is complicated -> potential for bugs



Virtual Bluetooth Device

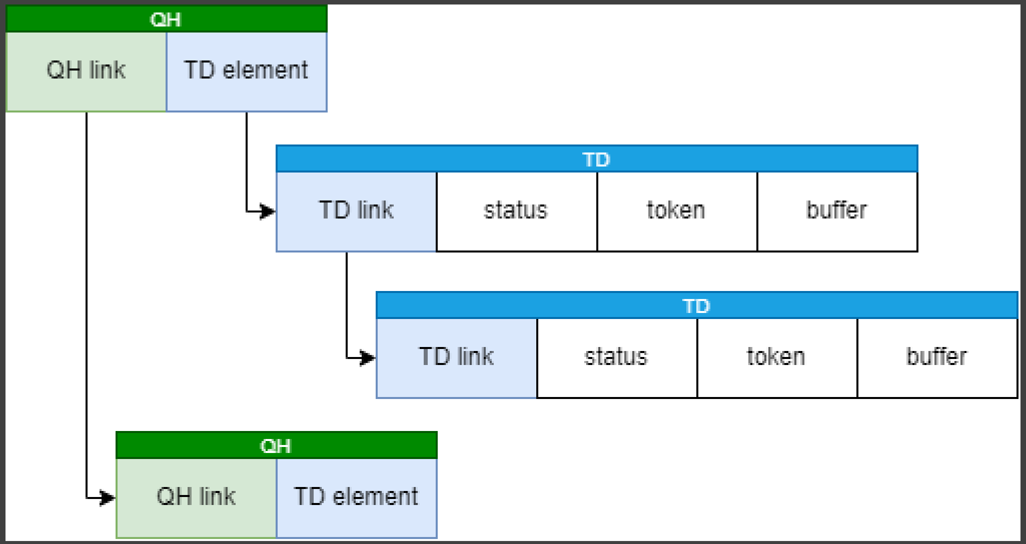
USB Controller: URB

- Most of communications rely on URB (USB Request Block) message
- There are 4 types of URB: Control URB , Bulk URB, Interrupt URB, Isochronous URB.
- Guest OS communicate to USB controller via URB message
- Each message identified by
  - `buffer` : URB data
  - `attribute` (`buffer` len, type, device address, device endpoint, eof bit, …),

Virtual Bluetooth Device

USB Controller: URB

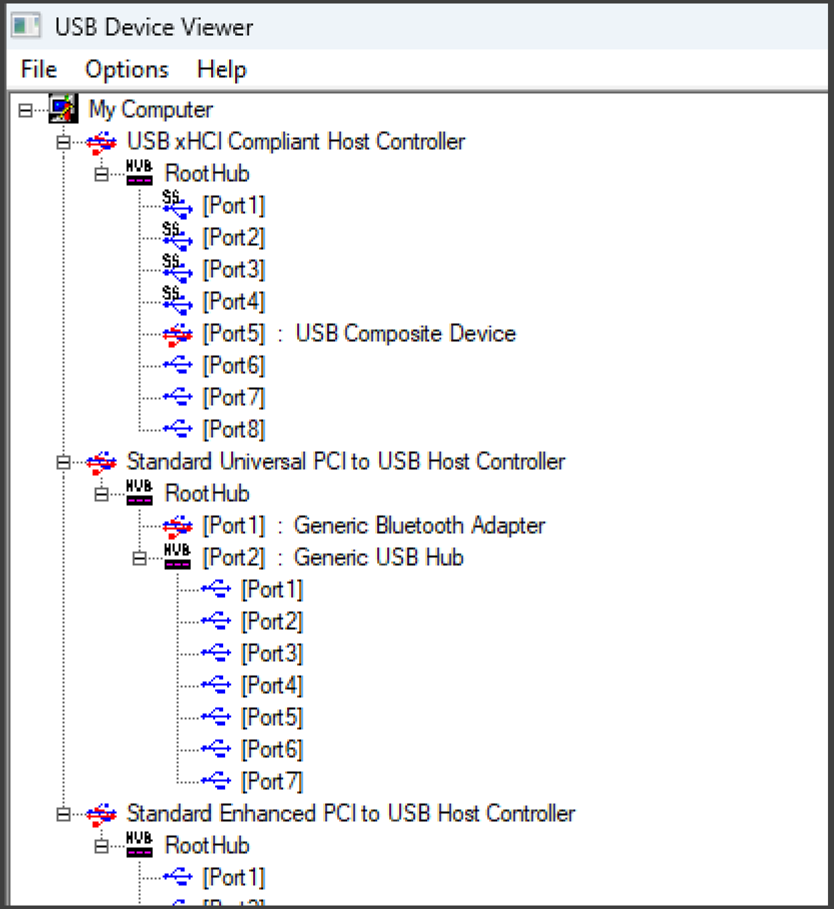
- In UHCI, URB message is represented in memory by 2 structs : QH (Queue-Head) and TD (Transfer Descriptor)

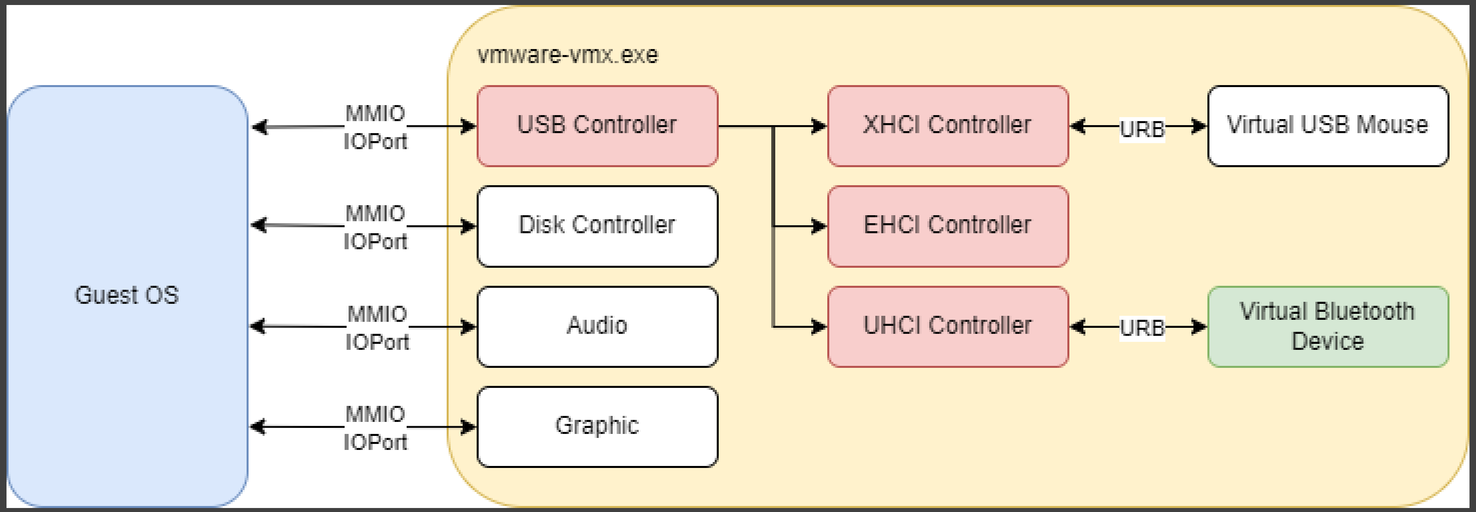


- `QH.link`: point to the next QH
- `QH.element`: point to next TD
- `TD.link`: point to next TD
- `TD.token`: encode URB atrribute
- `TD.buffer`: point to physical address of URB’s buffer

Virtual Bluetooth Device

Virtual Bluetooth Device





Virtual Bluetooth Device

Virtual Bluetooth Device

- It is enabled by default in both Linux and Windows (disabled in ESXi)

- Bluetooth is a complicated protocol

- Vmware implement Bluetooth basic feature:
  - Controller stack: HCI, LMP, SCO, ACL.
  - Host stack: L2CAP, RFCOMM, SDP.

- It communicates to UHCI Controller via URB message

<table>
<tr><th colspan="4">URB</th></tr>
<tr><td colspan="3">attribute</td><td rowspan="2">buffer</td></tr>
<tr><td>dev addr</td><td>dev enpt</td><td>…</td></tr>
</table>

Virtual Bluetooth Device

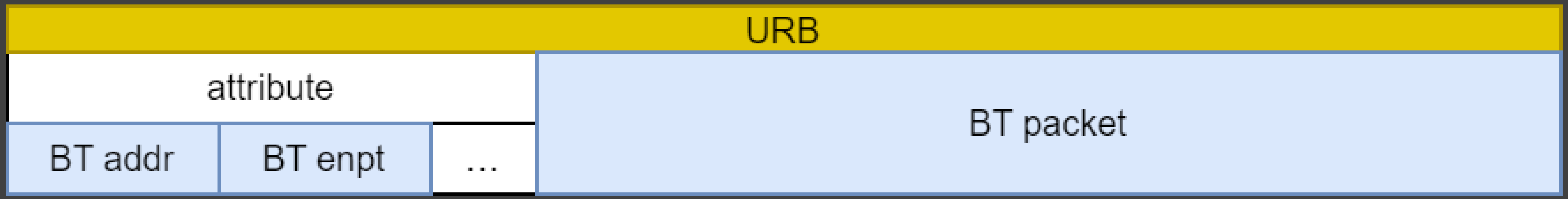
Virtual Bluetooth Device

- It is enabled by default in both Linux and Windows (disabled in ESXi)

- Bluetooth is a complicated protocol

- Vmware implement Bluetooth basic feature:
  - Controller stack: HCI, LMP, SCO, ACL.
  - Host stack: L2CAP, RFCOMM, SDP.

- It communicates to UHCI Controller via URB message



Infoleak bugs

Infoleak bugs

Bugs patterns

- In my journey, I found 3-4 infoleak bugs which have same pattern
- Memory Uninitialized -> leak uninitialized memory in heap
- Using heap spray technique to leak
  - .text -> rop chain
  - .heap -> easy to bypass other mitigations

Infoleak bugs

Bugs patterns

- Guest OS communicate to Virtual Device via MMIO / IOPort

- The process looks like following:

```
char *buffer = malloc(size);
uint32_t in_size;
uint32_t out_size;

copy_from_guest(buffer, &in_size);
out_size = process(buffer, in_size);
copy_to_guest(buffer, out_size);
```

- **buffer**: is allocated to store guest’s request data
- **in_size**: size of guest’s request data
- **out_size**: size of host’s response data, it is set when process guest’s request

Infoleak bugs

Bugs patterns

- `buffer` is not zero-out -> contains uninitialized memory in heap
- `in_size` is controllable by guest

If we can set `out_size` > `in_size`
-> memory disclosure

Infoleak bugs

CVE-2023-20869 root cause

Trigger by sending a Control URB message

```
__int64 __fastcall VirtualBluetooth_ProcessUrb(URB *urb)
{
  urb->status = 0;
  urb->out_size = guest_submitted_size;
  urb_data = urb->urb_data;
  endpt = pipe->endpt;

  if ( endpt ) {
    /* ... process urb_data and set `urb->out_size` */
  }
  if ( (urb_data->bmRequestType & REQ_MASK) == REQ_CLASS ) {
    /* ... process urb_data and set `urb->out_size` */
  }
  if ( USB_ProcessNonDeviceURB(urb) )  // process urb_data and set `urb->out_size`
    return UHCI_SendResponse(urb);
  bRequest = urb_data->bRequest;
  if ( bRequest == REQ_SET_CONFIGURATION ) {
    /* ... process urb_data and set `urb->out_size` */
  }
  if ( bRequest != REQ_SET_INTERFACE ) {
    urb->status = 4;    // error: invalid bRequest
    return UHCI_SendResponse(urb);
  }
}
```

Infoleak bugs

CVE-2023-20869 root cause

- `buffer`: `urb_data`
- `in_size`: not shown here
- `out_size`: `urb->out_size`

How about `out_size` ?

- Set by `guest_submitted_size` (controllable by guest) at start
- Will be set properly when processing `urb_data` -> safe
- But it forgets to set in case `bRequest` is invalid -> guest can control `out_size`.
- No check `out_size` vs `in_size`

-> leads to Memory Disclosure

Infoleak bugs

CVE-2023-20869 patch

- Correct the `out_size` in case `bRequest` is invalid

-> do not patch the ultimate root cause.

Infoleak bugs

A 20869's variant root cause

Trigger by sending a Control URB message

```
__int64 __fastcall VirtualHID_ProcessUrb(URB *urb)
{
    urb_data = urb->urb_data;
    hid_message = urb_data + 1;
    hid_message_size = urb->guest_submitted_size - 8;
    bmRequestType = urb_data->bmRequestType;
    switch ( urb_data->bRequest ) {
        case GET_REPORT:
            if ( hiddev->vtable->get_report_func )  // `get_report_func` is NULL in default
                hid_message_size = get_report_func(urb_data->wIndex, HIBYTE(urb_data->wValue), urb_data->wValue, hid_message, hid_message_size);
            else if ( urb_data != -8i64 )
                memset(hid_message, 0, hid_message_size);
        break;
        case SET_REPORT:
            if ( hiddev->vtable->set_report_func ) // `set_report_func` is NULL in default
                set_report_func(urb_data->wIndex, HIBYTE(urb_data->wValue), urb_data->wValue, hid_message, hid_message_size);
            break;
        default:
            urb->status = 3;
            goto SEND_RESPONSE_LABEL;
    }
    if ( hid_message_size >= 0 ) {
        urb->status = 0;
        urb->out_size = hid_message_size + 8;
        goto SEND_RESPONSE_LABEL;
    }
}
```

Infoleak bugs

A 20869’s variant root cause

Found in Virtual Human Interface Device (HID)

- `buffer`: `urb_data`
- `in_size`: not shown here
- `out_size`: `urb->out_size`

Although we can control `out_size` and it doesn’t check `out_size` vs `in_size`, it does `memset` at `GET_REPORT` case to prevent memory disclosure -> safe

But it missed `memset` at `SET_REPORT` case

-> Memory Disclosure

Infoleak bugs

A 20869's variant patch

- Many variants were found (reported to Vmware program, used in Tianfu …)
- Vmware still only add check `out_size` vs `in_size` or correct `out_size`
- Until 03/2024, they finally patched the root cause

```
URB* __fastcall AllocateNewURB(__int64 pipe, __int64 num_packets, __int64 total_buffer_size)
{
  /* ... */
  urb->hcpriv = 0i64;
+ if ( !type && urb->urb_data )
+   memset(urb->urb_data, 0, allocLen);
  urb->pipeLink.next = &urb->pipeLink;
  /* ... */
}
```

Infoleak bugs

A 20869's variant patch

- It only does `memset` if `pipe_type` = 0 (Control URB)
- What about other endpoints (Bulk, Isoc, Intr) ?

-> Still not complete

Infoleak bugs

CVE-2024-22267 root cause

Trigger by sending a Bulk URB message

```
void __fastcall L2CAP_HandleSignalChannel(__int64 *l2cap, l2cap_struct a2, __int64 in_UrbBuf) {

    case L2CAP_CMD_ECHO_REQ:
      L2CAP_Response(l2cap, L2CAP_CMD_ECHO_RSP, a2.id, in_UrbBuf);
      break;
    case L2CAP_CMD_INFO_REQ:
      if ( !UrbBuf_CopyOut(in_UrbBuf, &req_type, 2i64) )
        goto LABEL_9;
      warn("Bluetooth-L2CAP: Unsupported Info Request, type=%04x\n", req_type);
      v28 = *l2cap;
      resp_data = req_type;
      LOWORD(v32) = 1;
      v13 = UrbBuf_NewWithData(*(v28 + 24), &resp_data, 4i64);
      L2CAP_Response(l2cap, L2CAP_CMD_INFO_RSP, v29.id, v13);
      UrbBuf_Release(v13);
      break;

}
```

Infoleak bugs

CVE-2024-22267 root cause

- Case `L2CAP_CMD_INFO_REQ`
  - `buffer`: `in_UrbBuf`
  - `in_size`: `sizeof(in_UrbBuf)`
  - `out_size`: 4

-> It is safe

- Case `L2CAP_CMD_ECHO_REQ`
  - `buffer`: `in_UrbBuf`
  - `in_size`: `sizeof(in_UrbBuf)`
  - `out_size`: `sizeof(in_UrbBuf)`

Is it safe ?

Infoleak bugs

CVE-2024-22267 root cause

Let see how Bulk URB is create:

```
__int64 __fastcall UHCI_CreateBulkURB(__int64 a1, __int64 a2) {
    /* ... */
    if ( td.token == USB_TOKEN_PID_OUT && td.size ) {
        if ( !td.buffer || (!PhysMem_CopyFrom(td.buffer, urb->urb_data_cursor, td.size)) ) {
            Log("UHCI: Bad %s pointer %#I64x.\n", "TDBuf", v14->td.buffer);
            return;
        }
    }
    urb->in_size += td.size;
    urb->urb_data_cursor += td.size;
    /* ... */
}
```

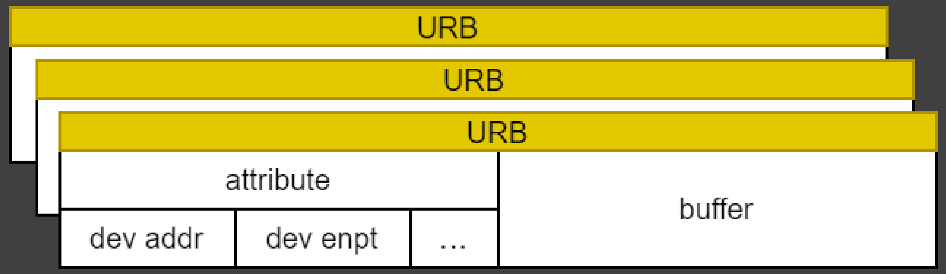
- Guest could fully control TD struct -> `token` is controllable
- It only copy data from guest to `urb_data` if `USB_TOKEN_PID_OUT`
- Malformed TD with `USB_TOKEN_PID_SETUP` -> skip copying but still increase `in_size`

-> cause `buffer` contains uninitialized data -> Memory Disclosure

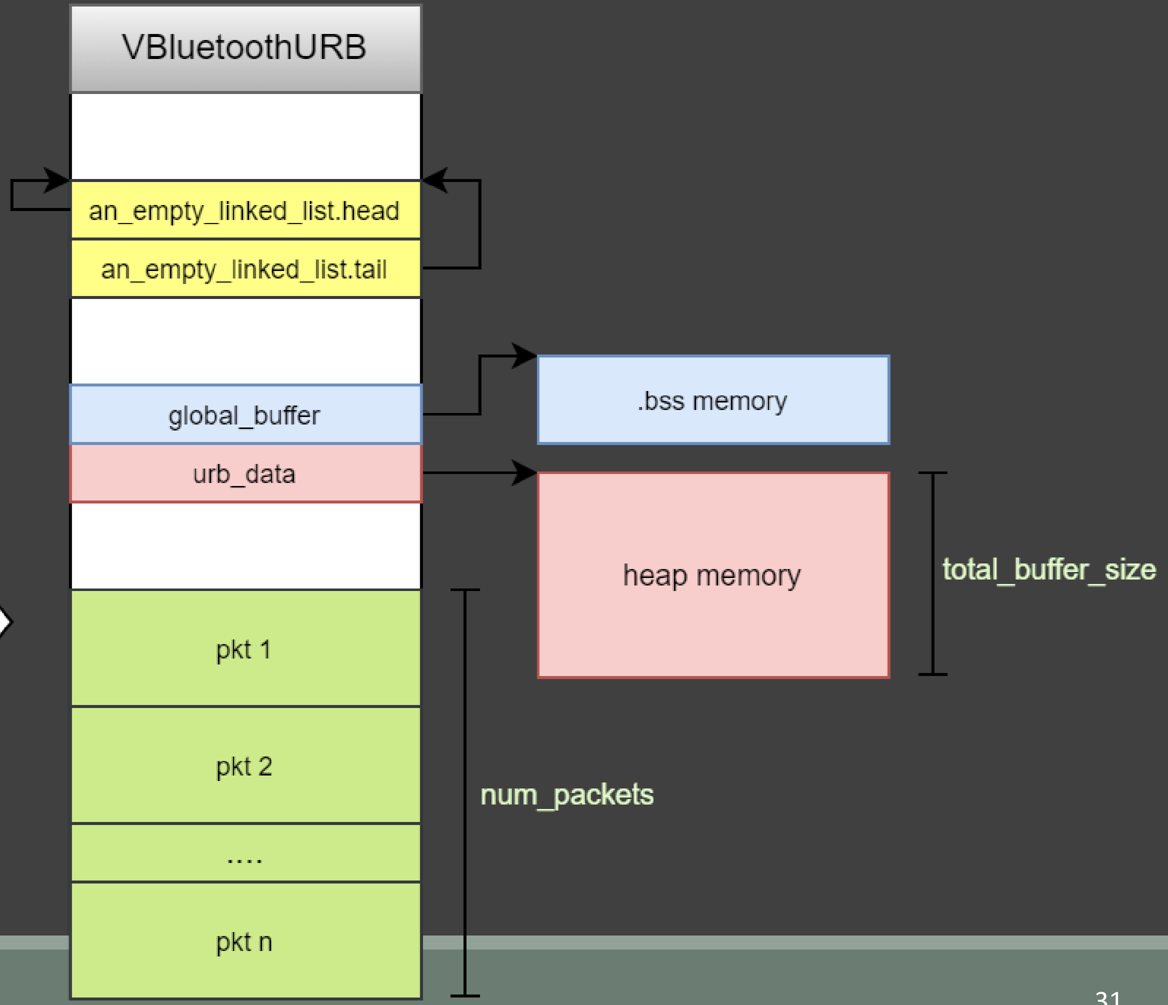
Infoleak bugs

Exploitation

VBluetooth URB object




```
URB* __fastcall AllocateNewURB(
    __int64 pipe,
    __int64 num_packets,
    __int64 total_buffer_size
)
```

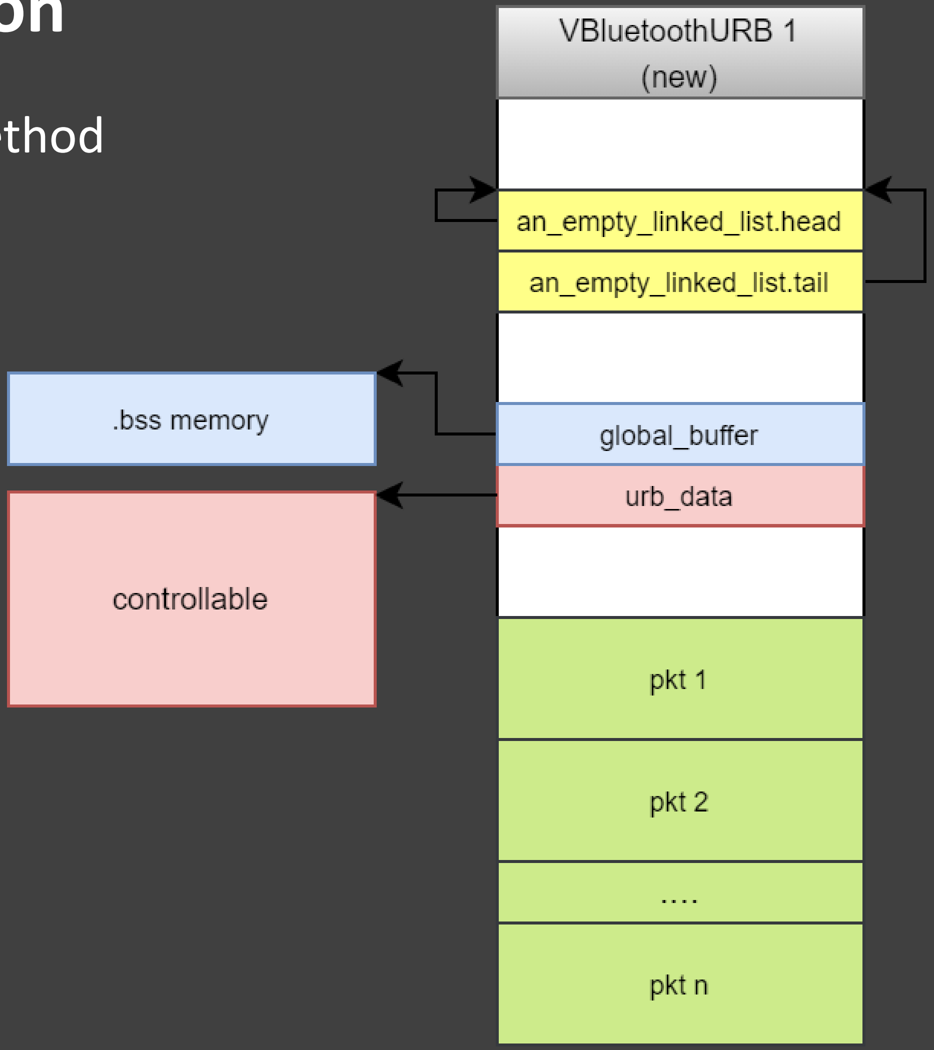


Infoleak bugs

Exploitation

My exploit method

1. Allocate

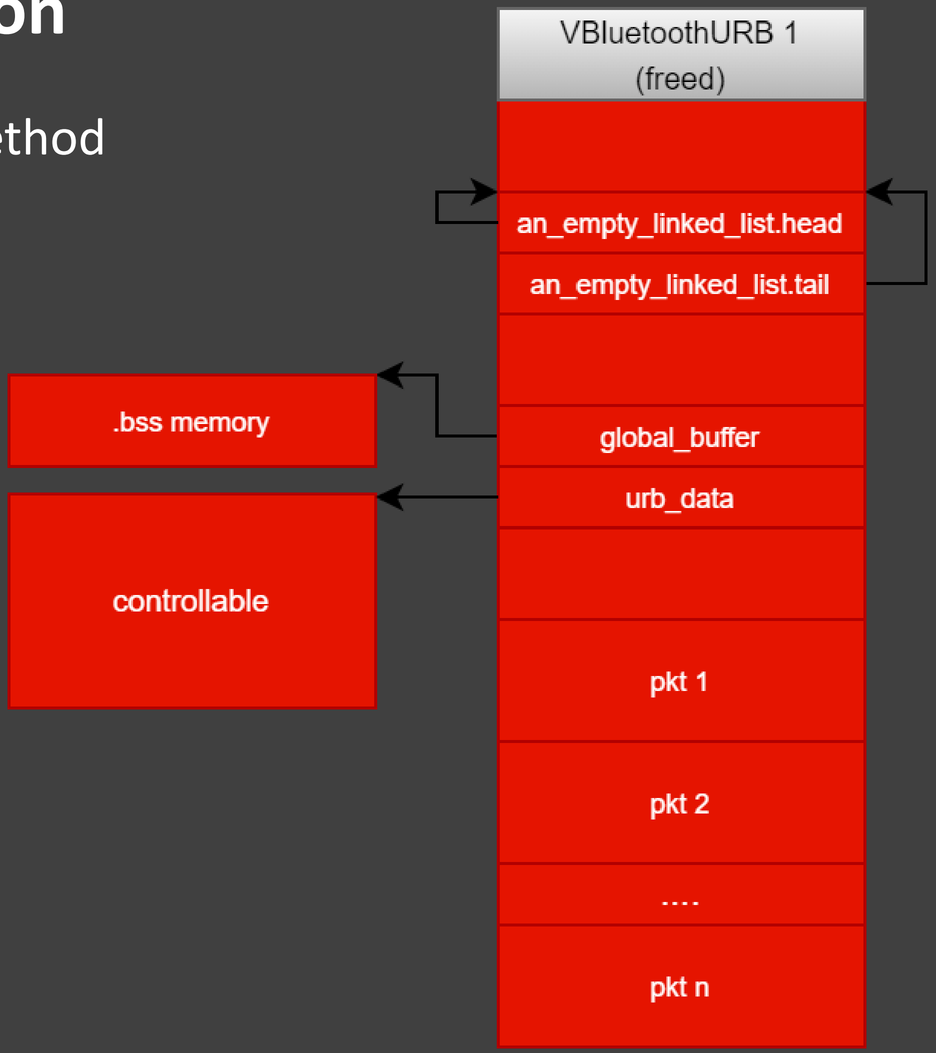


Infoleak bugs

Exploitation

My exploit method

2. Free

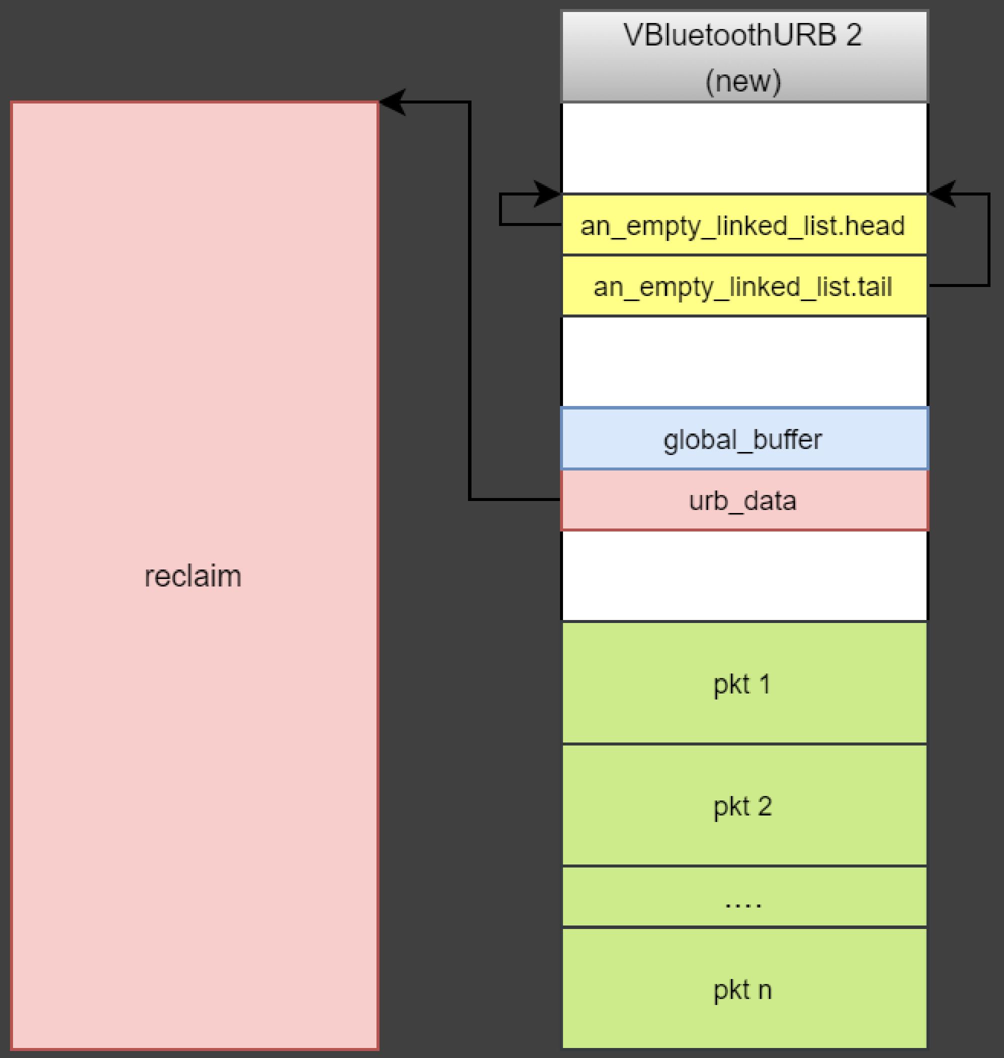


Infoleak bugs

Exploitation

My exploit method

3. Reclaim

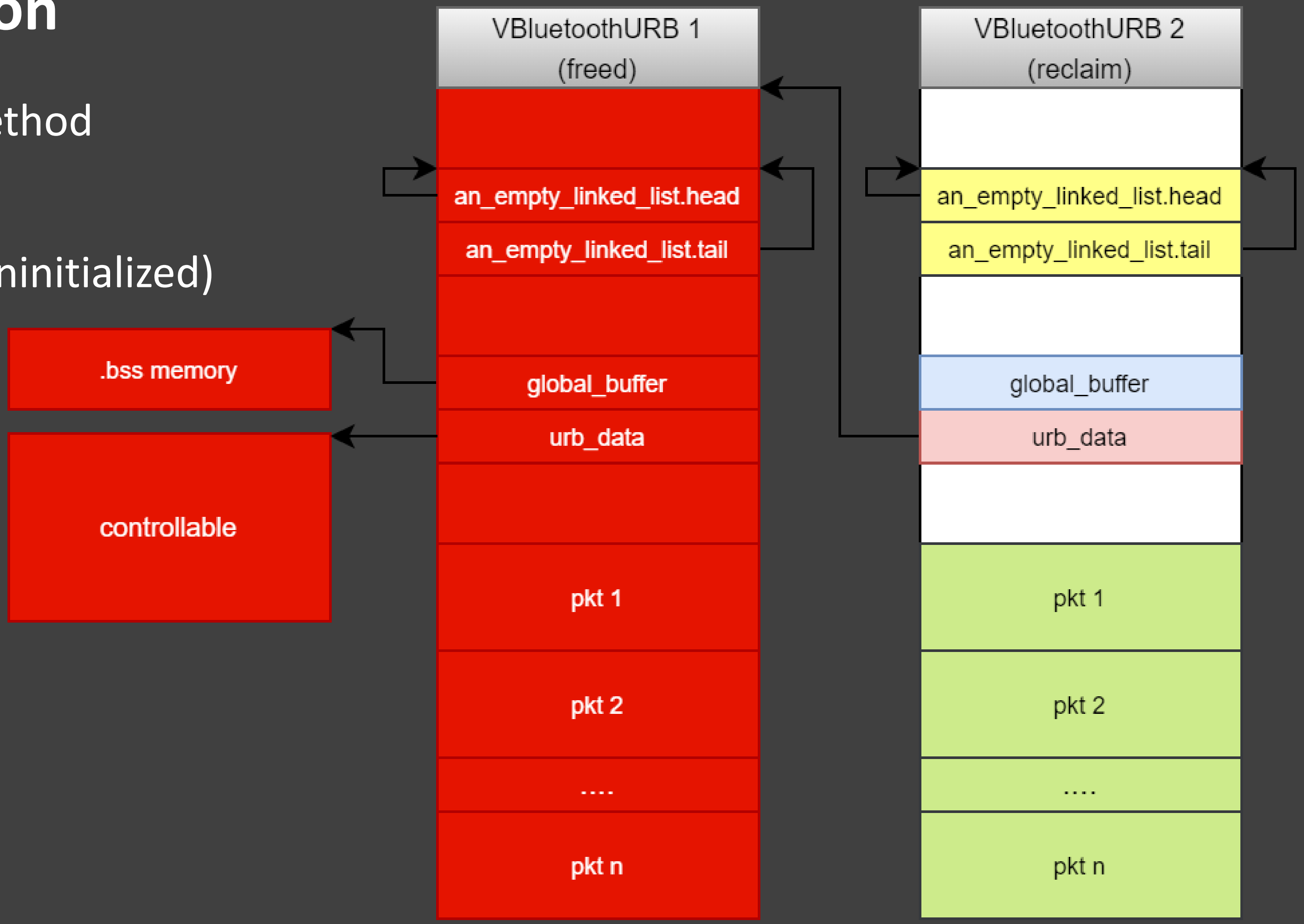


Infoleak bugs

Exploitation

My exploit method

3. Reclaim
(urb_data is uninitialized)



Infoleak bugs

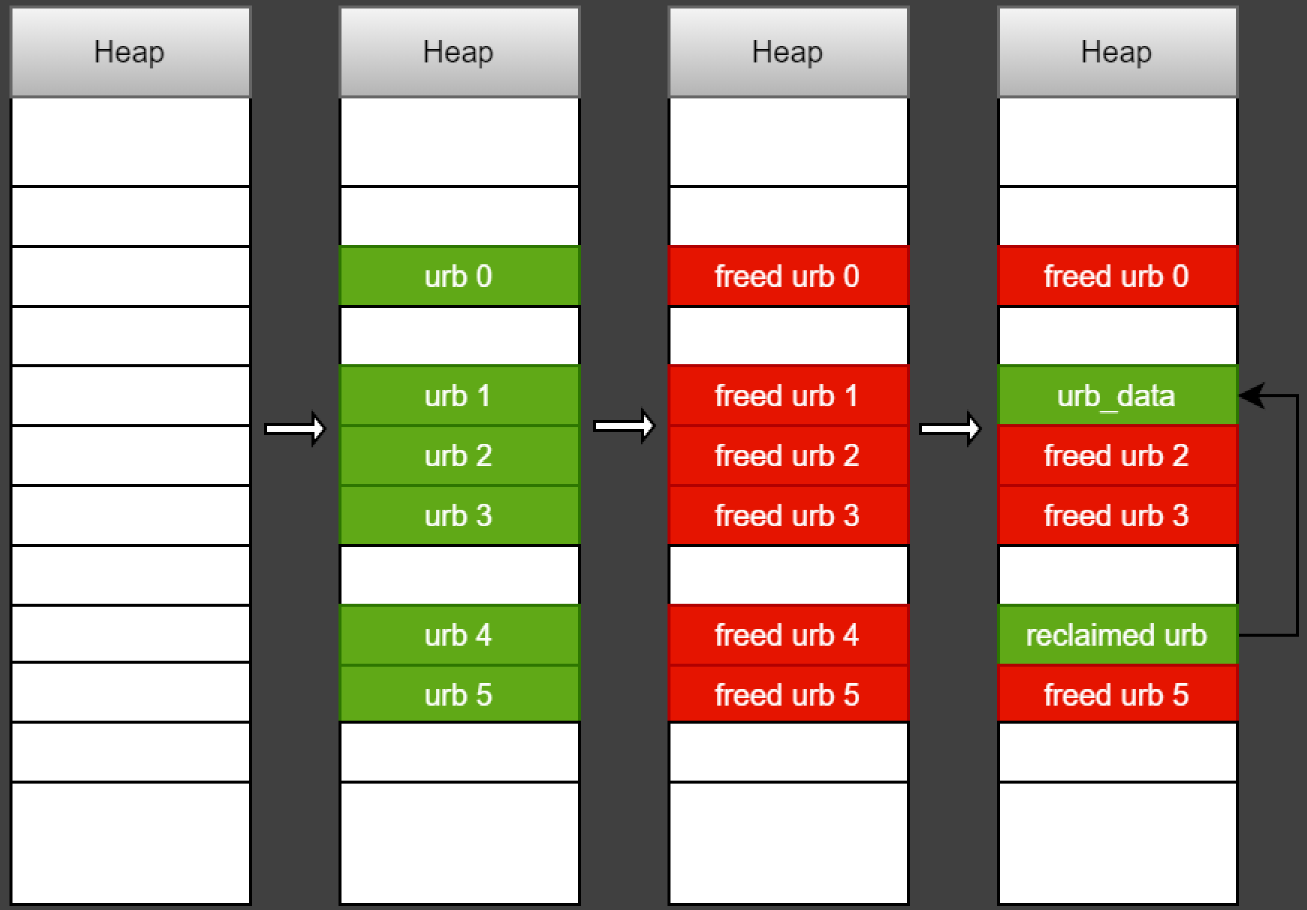
Exploitation – CVE-2023-20869

- Abuse `Control URB` message

- **urb_data** buffer size is flexible, controllable by guess

- Control `VBluetoothURB` object size is fixed, = 0xA0 (LFH enable)

-> leak:
- **global_buffer** ptr -> .text base



Infoleak bugs

Exploitation – CVE-2024-22267

- Abuse `Bulk URB` message. It is more strict than `Control URB`

- `urb_data` buffer size must be multiple of 0x40,
- Normal `VBluetoothURB` object size is 0xA0

-> can’t use `urb_data` buffer to reclaim `VBluetoothURB`

Infoleak bugs

Exploitation – CVE-2024-22267

- Luckily, I found that `Isochronous  URB` is special: can send multiple Isoc URB messages in one time

-> increase size of `Isochronous  URB` object, each msg add 0xC bytes to `Isochronous  URB` obj

- 0x54 msg: create an `Isoc URB` object size: 0xA0 + 0x54*0xC = 0x490

-> Send a `Bulk URB` message with data size 0x480 can reclaim the above freed `Isoc URB` object

Infoleak bugs

Exploitation – CVE-2024-22267

- Try the same exploitation method as the previous one:

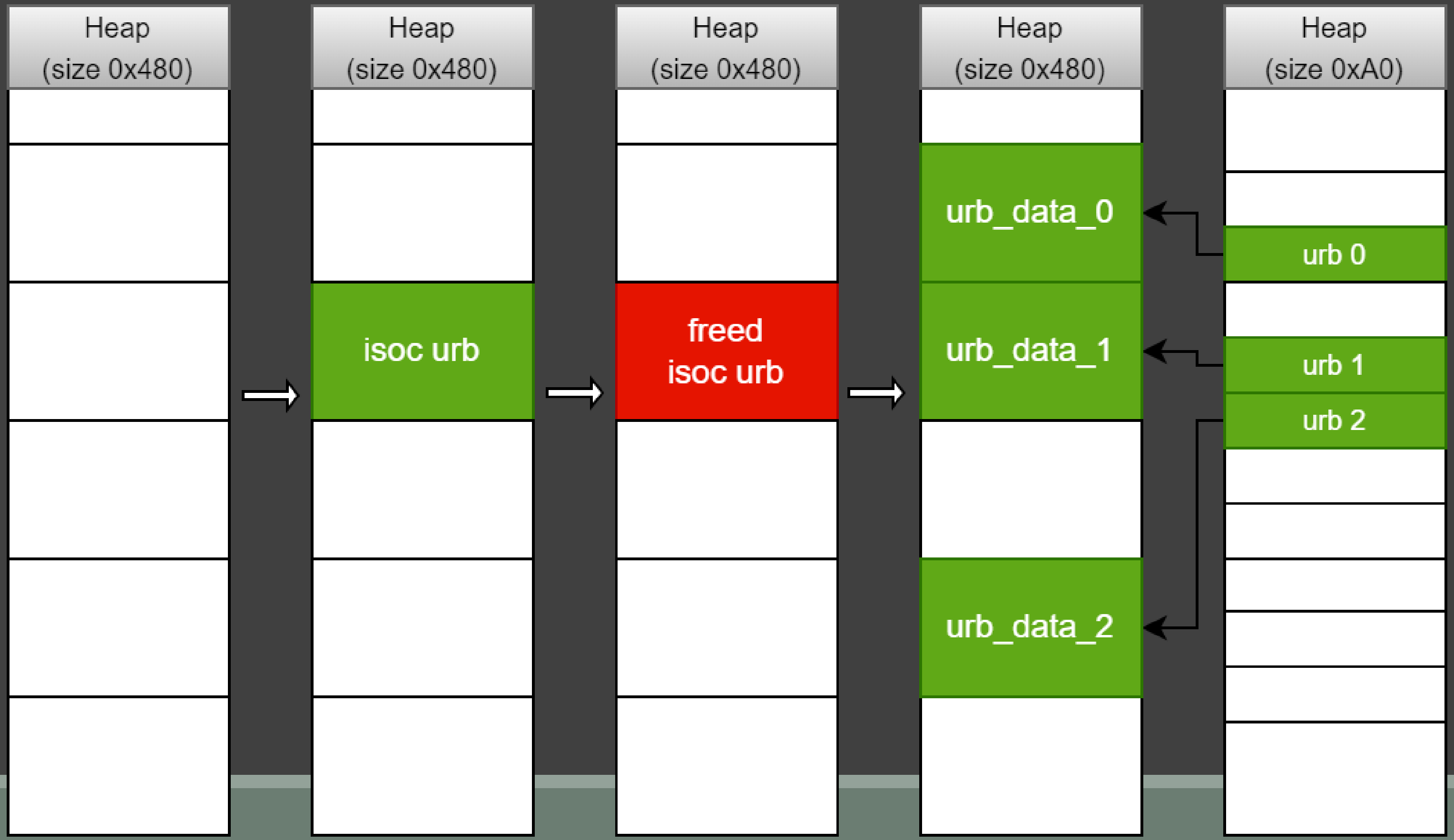
-> success, but stability is low, maybe because the heap size is larger -> unstable

- Stability is very important here.

Infoleak bugs

Exploitation – CVE-2024-22267

- After a few experimentations, I found a stable method to exploit it:
  - Prepare: Enable LFH first



Infoleak bugs

Exploitation – CVE-2024-22267

-> leak:

- `global_buffer` ptr -> .text base
- `an_empty_linked_list.head` and `tail` -> .heap address

- Result is even better than expected
  - Address of buffer size 0x480 is very stable
  - Subsequence allocations size 0x480 -> return same address

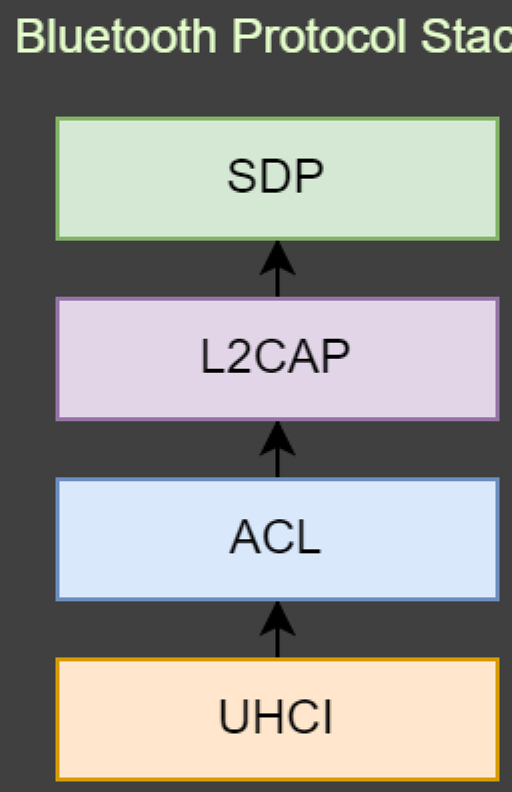
-> could send more URB to write data to known heap memory
-> It is a very powerful primitive

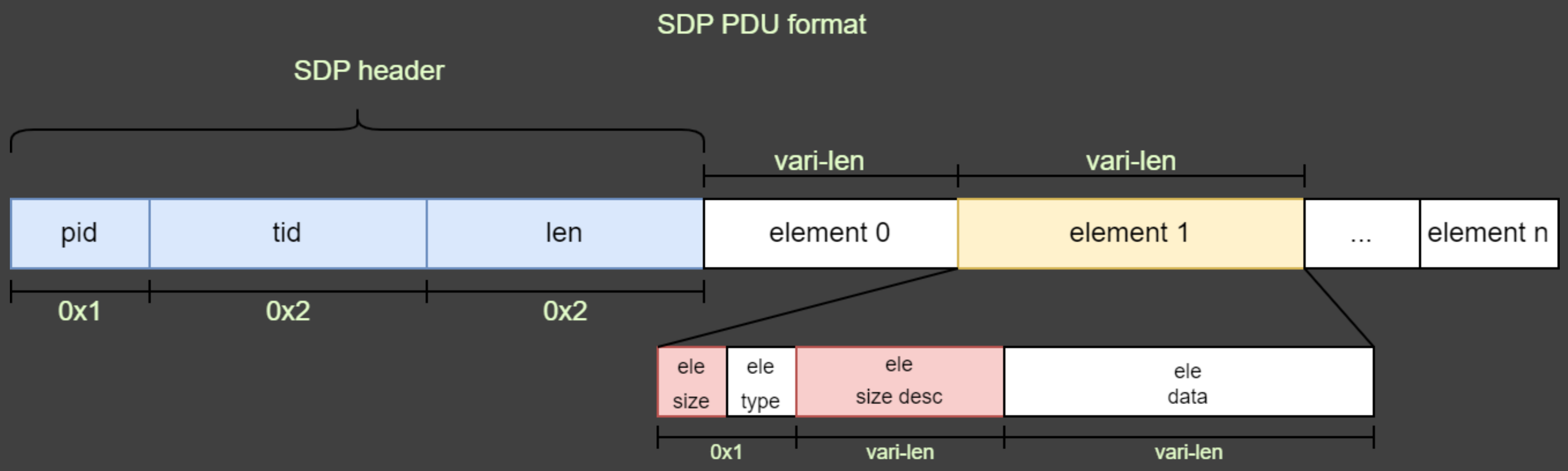
2023 bug: Stack overflow

2023 bug: Stack Overflow

CVE-2023-20869 Root Cause

- SDP Protocol





SDP element:
+ TLV format
+ L – length is variable (not fixed 2/4 bytes)

2023 bug: Stack Overflow

CVE-2023-20869 Root Cause

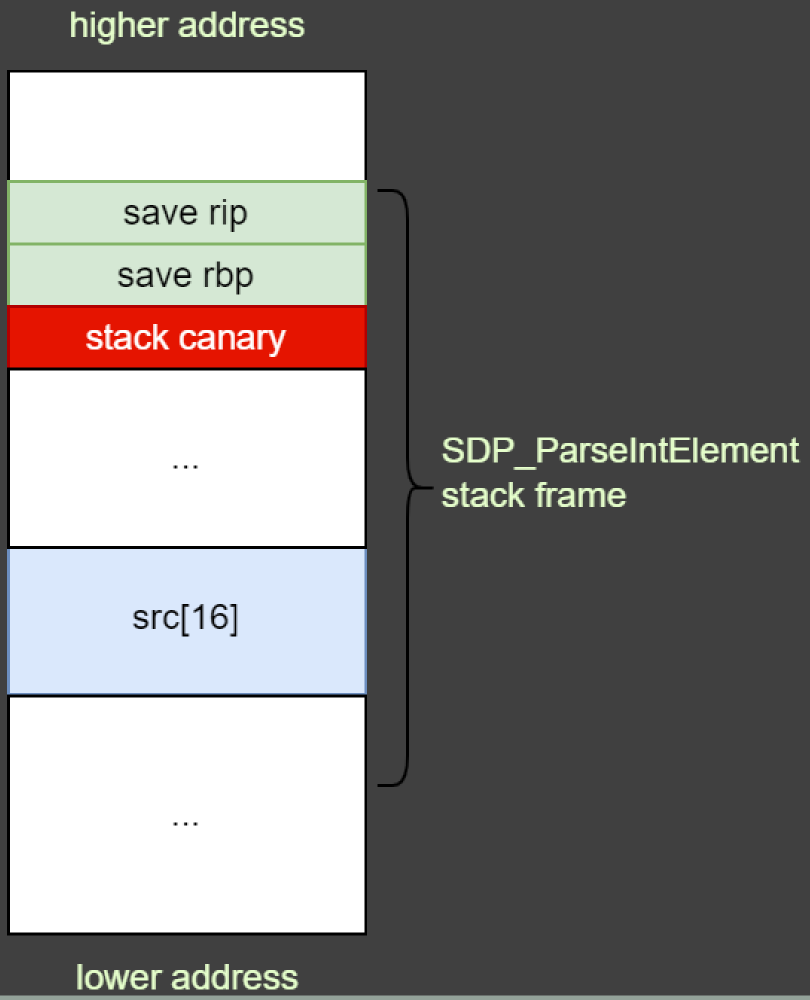
- SDP_ParseIntElement function parse element type Integer

```
char __fastcall SDP_ParseIntElement(buf *sdp_buf, unsigned int ele_size, _QWORD *a3, _QWORD *a4) {
  char src[16];
  result = buf_copy(*sdp_buf, tmp, ele_size);
  if ( result ) {
    memcpy(&src[-ele_size], src, ele_size);
    *a3 = 0i64;
    if ( a4 )
      *a4 = 0i64;
    return SDP_Slice(in_rbuf, v4);
  }
  return result;
}
```

- len is controllable -> stack overflow in buf_copy and memcpy

2023 bug: Stack Overflow

CVE-2023-20869 Exploitation



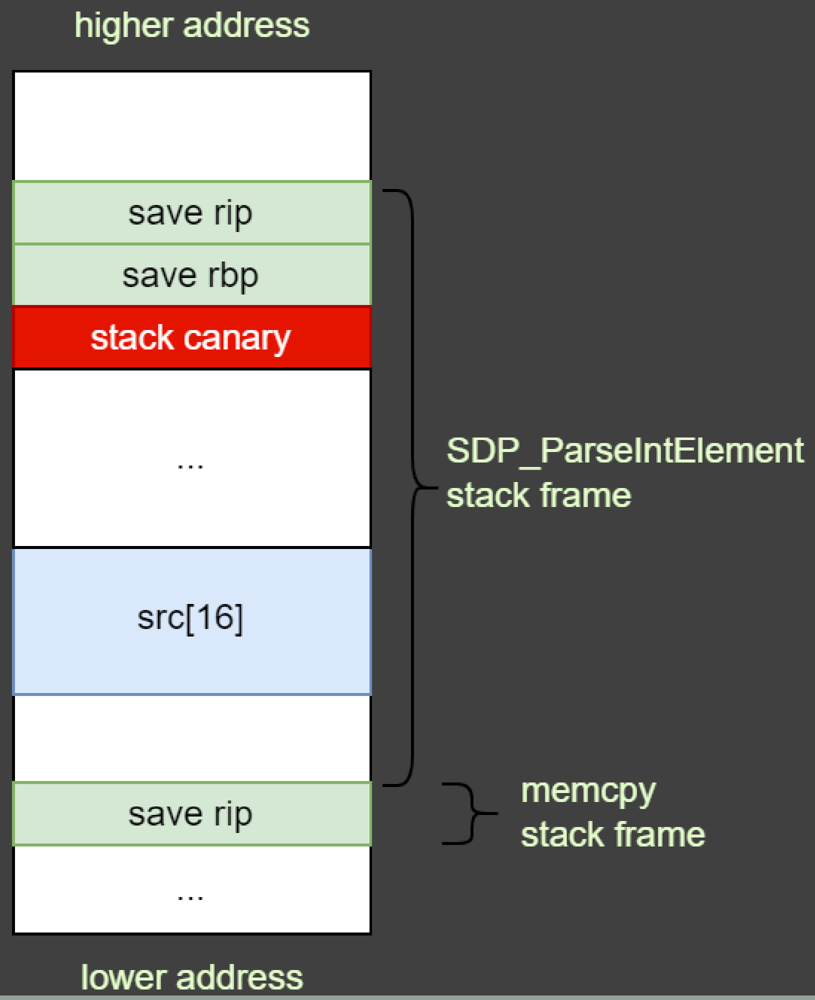
Stack bof due to `buf_copy`

- Stack canary protect

-> unexploitable

2023 bug: Stack Overflow

CVE-2023-20869 Exploitation



Stack bof due to `memcpy`

- `memcpy` function doesn’t use stack -> no canary
- can downward the `src` buffer to overwrite

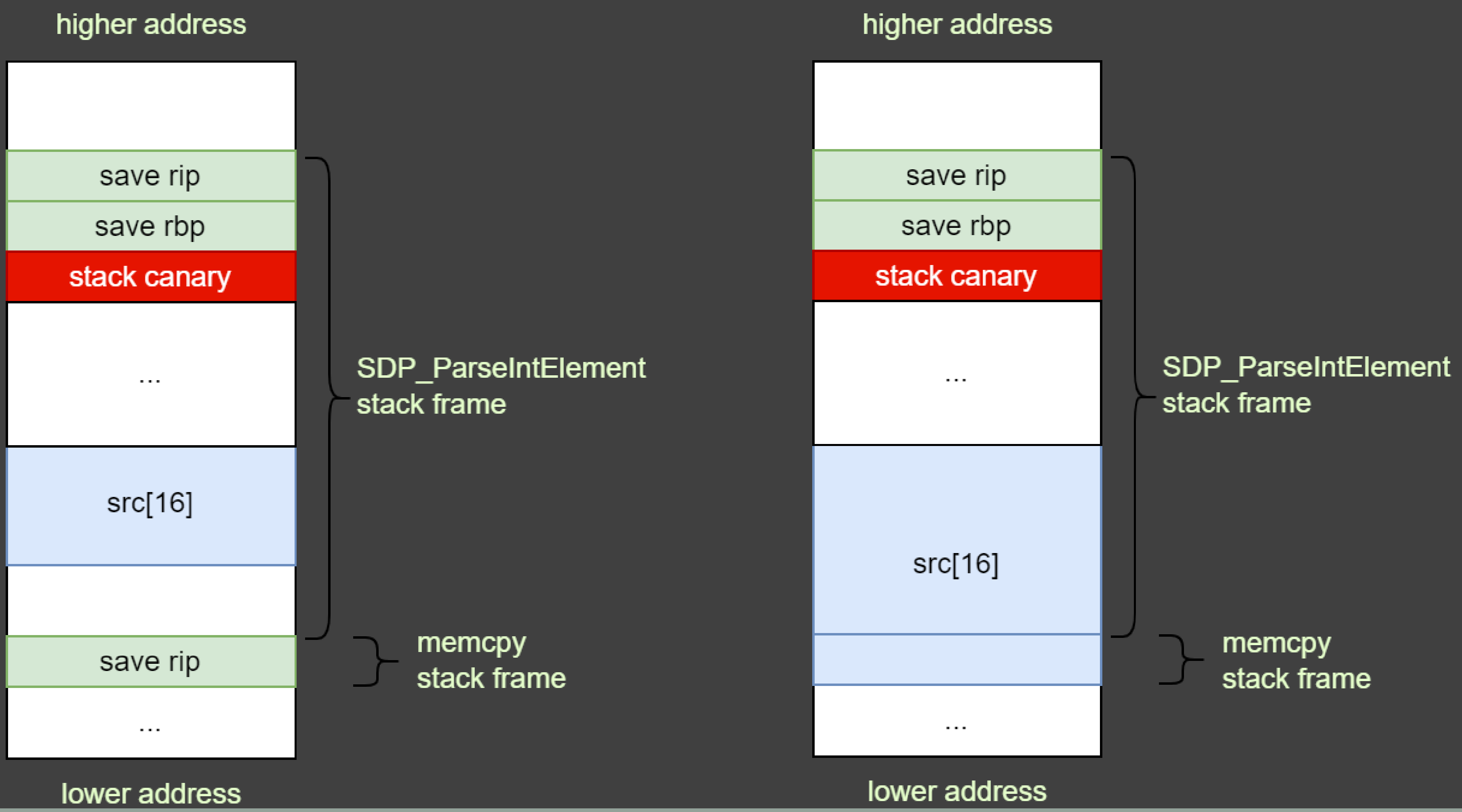
`memcpy` ‘s save rip

- execute rop chain directly

-> exploitable

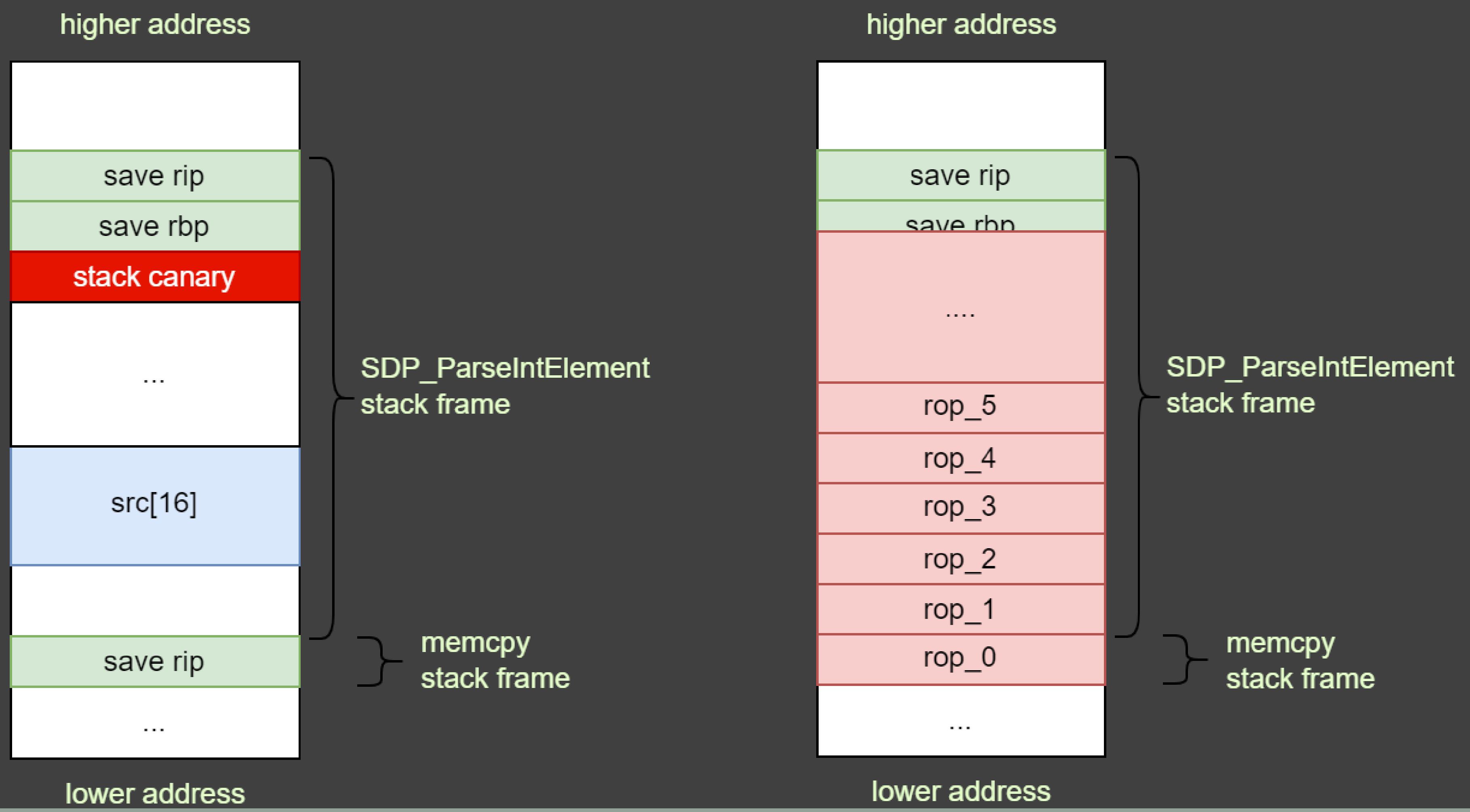
2023 bug: Stack Overflow

CVE-2023-20869 Exploitation



2023 bug: Stack Overflow

CVE-2023-20869 Exploitation

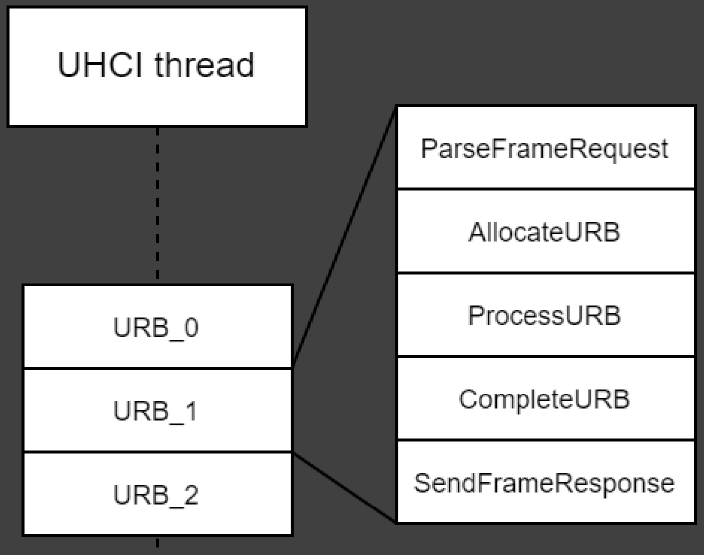


2024 bug: Use-after-free

2024 bug: Use-after-free

CVE-2024-22269 root cause

UHCI thread worker



- `UHCI_CreateBulkURB`
- `AllocateNewUrb`
- `VirtualBluetooth_ProcessURB`

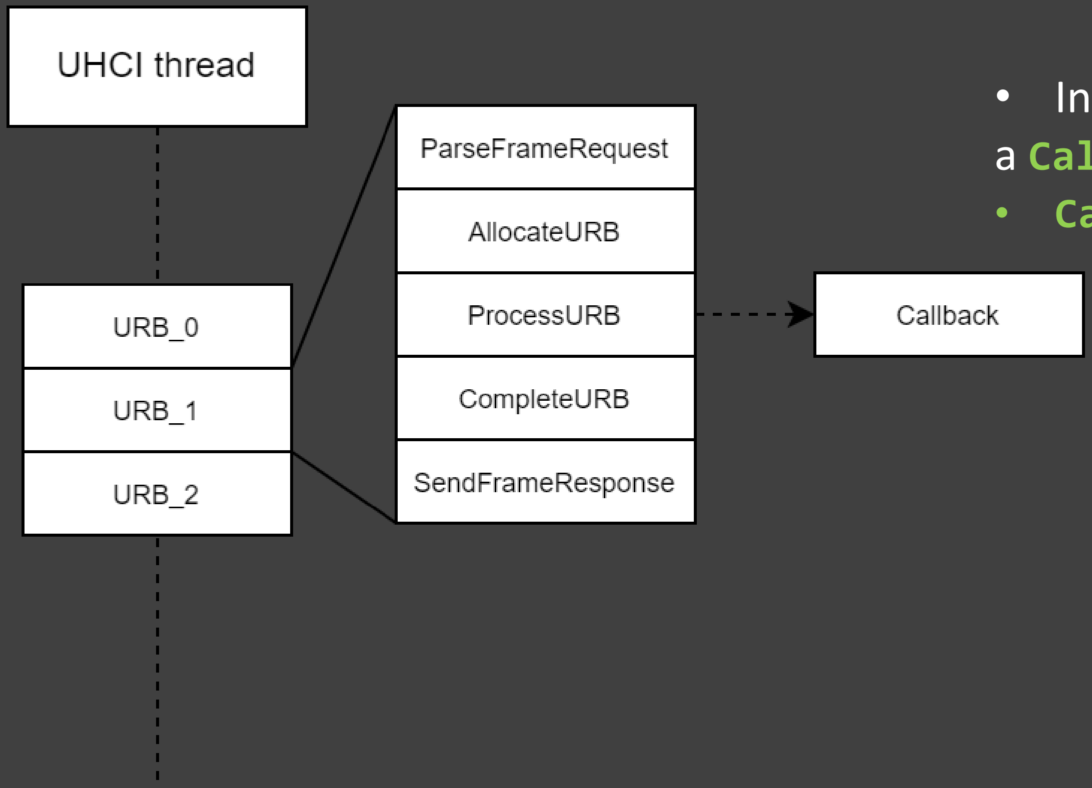
- `UHCI_SendResponse`

- URB messages are processed sequentially

2024 bug: Use-after-free

CVE-2024-22269 root cause

UHCI thread worker handle callback

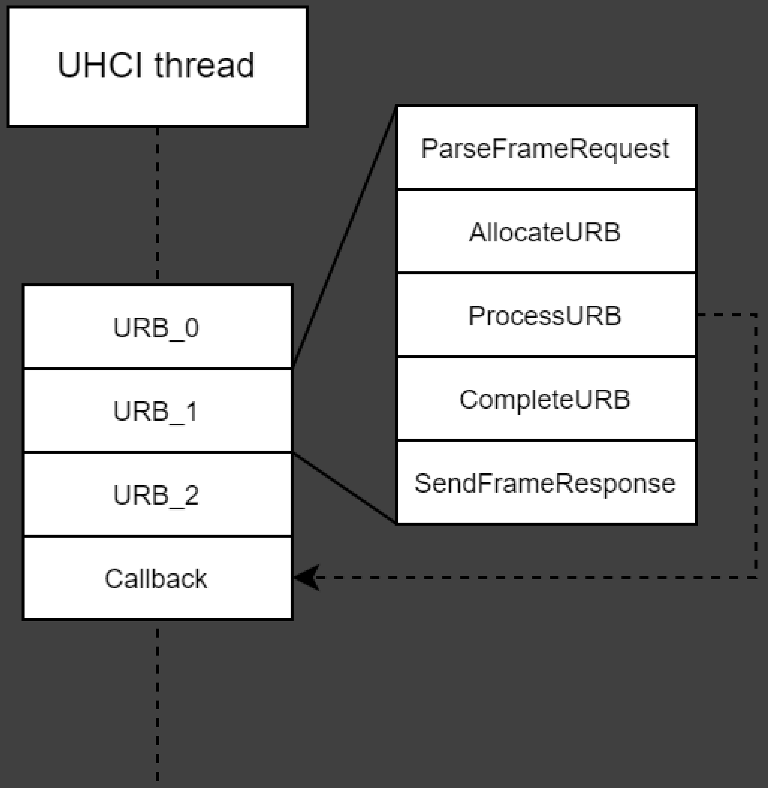


- In `ProcessURB` function, it might schedule a `Callback` function
- `Callback` function is executed in same thread

2024 bug: Use-after-free

CVE-2024-22269 root cause

UHCI thread worker handle callback

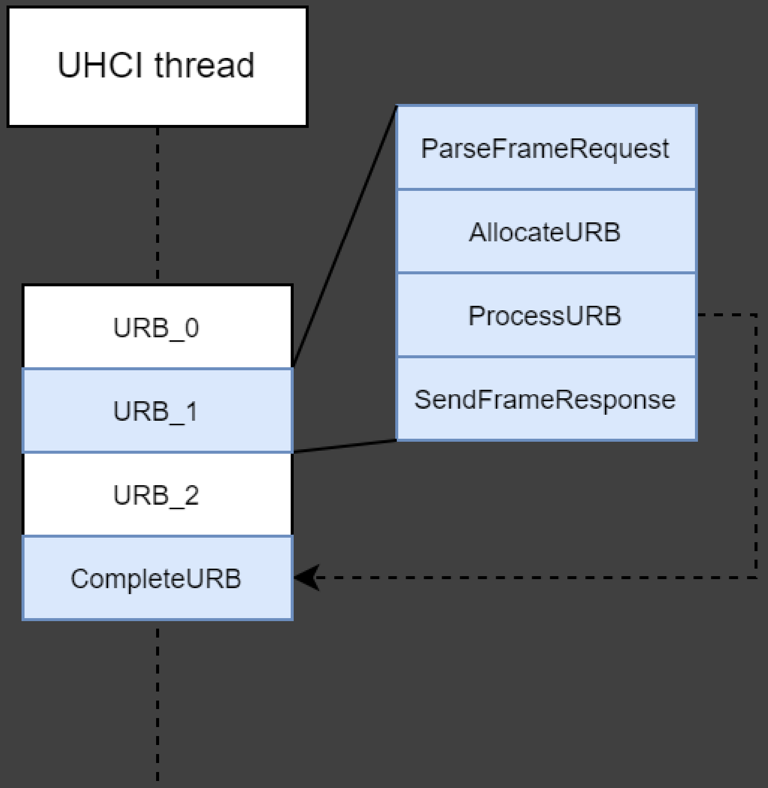


- In **`ProcessURB`** function, it might schedule a **`Callback`** function
- **`Callback`** function is executed in same thread -> It must wait for processing other urb messages

2024 bug: Use-after-free

CVE-2024-22269 root cause

UHCI thread worker handle callback

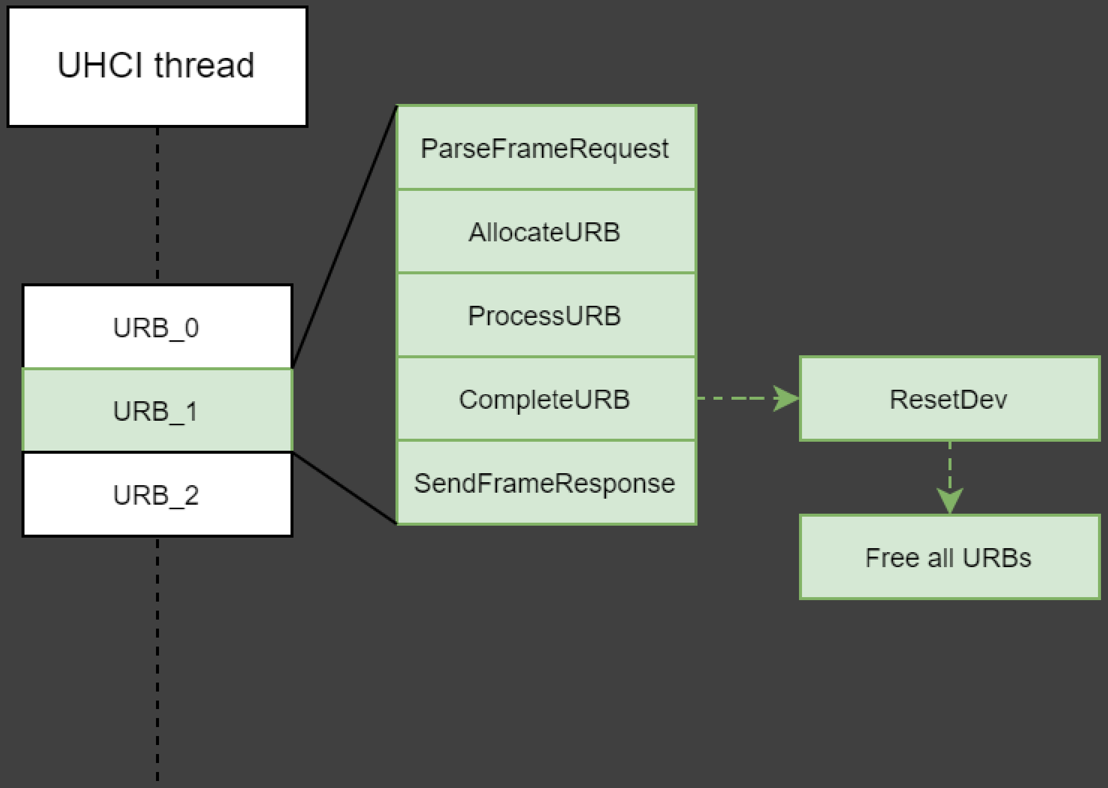


- Bulk URB msg
- Bluetooth Device
- `L2CAP_ProcessRequest`

2024 bug: Use-after-free

CVE-2024-22269 root cause

CompleteURB phase

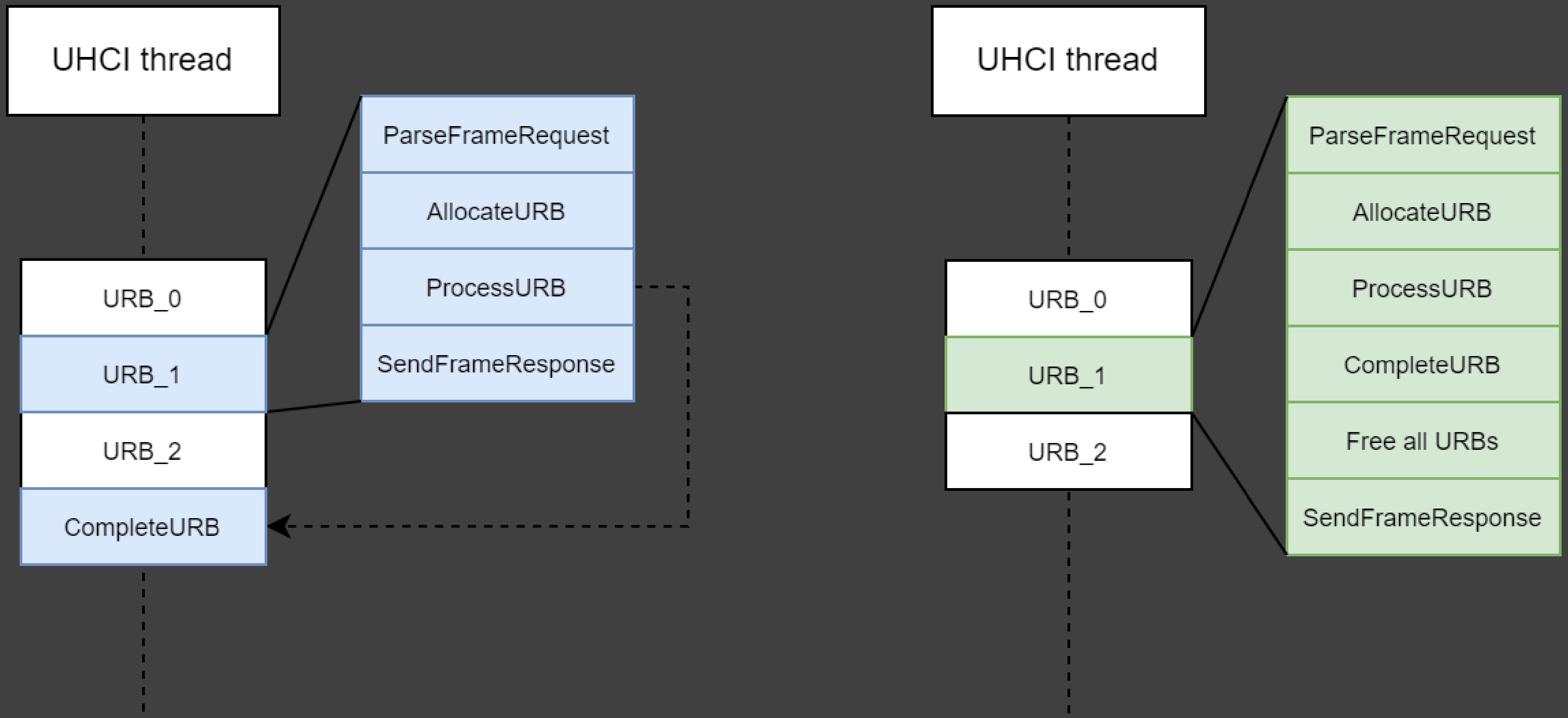


- Control URB msg + command `USB_REQ_SET_CONFIGURATION`

2024 bug: Use-after-free

CVE-2024-22269 root cause

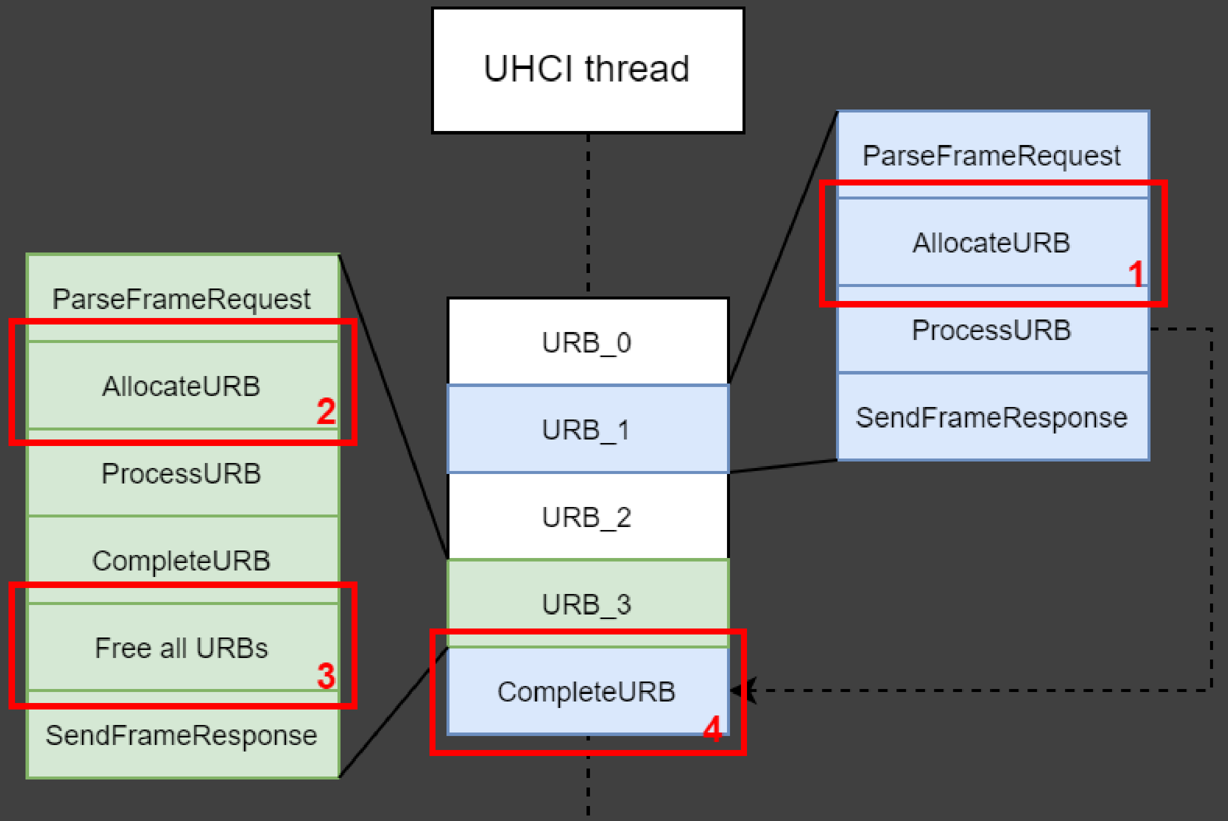
Use-After-Free Scenario



2024 bug: Use-after-free

CVE-2024-22269 root cause

Use-After-Free Scenario



1. Allocate Bulk URB object
2. Allocate Control URB
3. Free all URB objects,
   include Bulk URB object (1)
4. Trigger callback with parameter
Bulk URB object (1).

-> Use-After-Free

2024 bug: Use-after-free

CVE-2024-22269 exploitation

CộņřľêʧêÛŖB function

```
bool __fastcall Usb_CompleteUrb(urb *urb)
{
  /* ... */
  pipe = urb->UsbPipe;
  dev = pipe->UsbDevice;
  if ( urb->status == 0x6 )
  {
    if ( urb->field_88 )
      (dev->UsbHostCtl->DevicePrivateData.vtable->field_20)(urb);
    urb->field_60 = 0x2;
    Usb_Remove(urb, urb->field_8, urb->field_50);
    Usb_ReleaseUrb(urb);
    return 0;
  }
  /* ... */
}
```

What we got

- .text address
- 1st parameter is freed `urb` object

Exploit method:

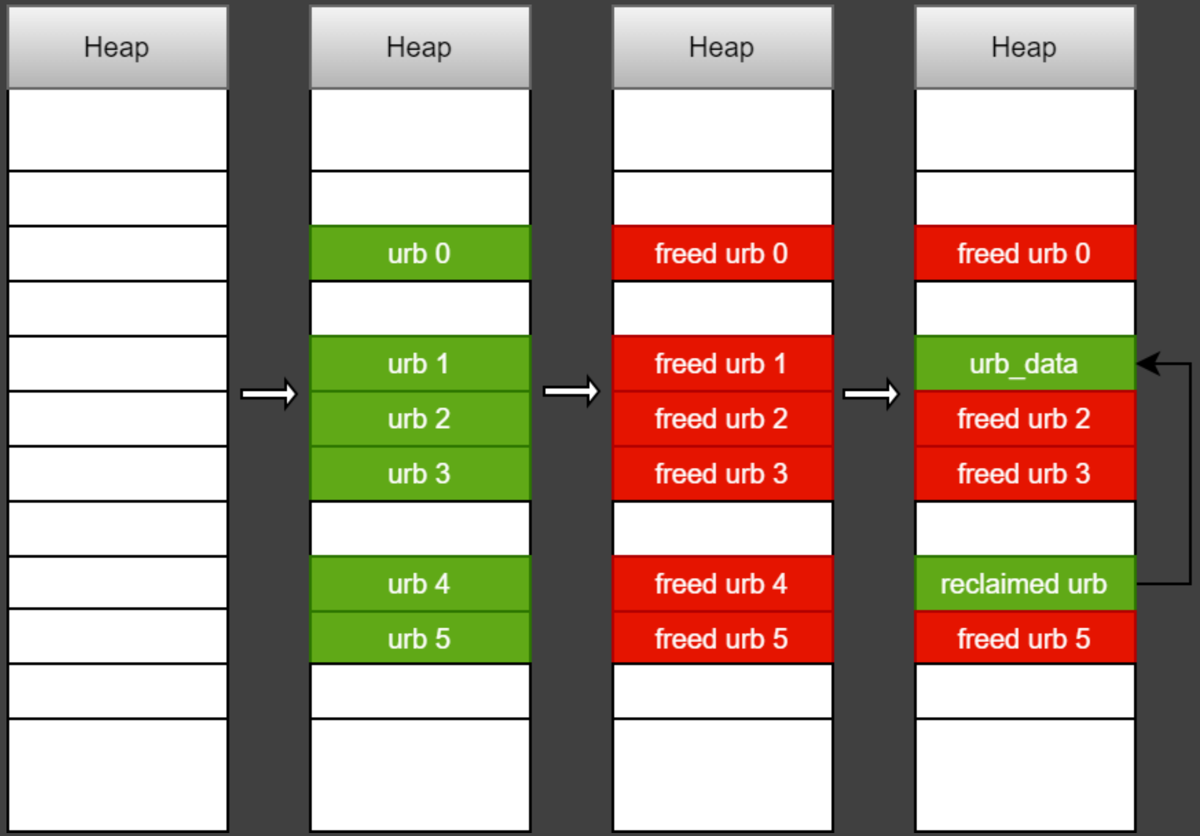
- Reclaim freed `urb` object
- Fake `UsbPipe` object -> `vtable`

-> Stack pivot to `urb` object memory

2024 bug: Use-after-free

CVE-2024-22269 exploitation

Reclaim freed ụsč object



- Bulk **urb** object size is 0xA0

-> reuse CVE-2023-20869's exploit method

2024 bug: Use-after-free

CVE-2024-22269 exploitation

Stack pivot and execute rop chain

```
.text:000000014024E6C4                 mov     rcx, rbx
.text:000000014024E6C7                 call    cs:__guard_dispatch_icall_fptr
.text:000000014024E6CD
```

- it means CFG (Control Flow Guard) is enabled -> cant call arbitrary rop gadgets

2024 bug: Use-after-free

CVE-2024-22269 exploitation

Ideas ?

- trigger the uaf mutiple times, to call allowed functions
- chain functions together to bypass CFG

2024 bug: Use-after-free

CVE-2024-22269 CFG bypass

Trigger the uaf mutiple times

- It require our exploit must be stable, no crash after triggering the bug
- When I started developing my exploit, the stability is not too good, about 50-60%. I need to improve it more. I still use the same method, but just tried improving the stability
- No ultimate method, just **trial and error**
  - Change number of spray urb messages
  - Change order of sending urb messages
  - Change type of urb messages

2024 bug: Use-after-free

CVE-2024-22269 CFG bypass

Trigger the uaf mutiple times

-> Final approach:

- Send 8 bulk urb message -> allocate 8 `urb` objects in heap
- Send 1 control urb message with cmd `USB_REQ_SET_CONFIGURATION` -> free 8 `urb` objects
- Send 8 control urb messages -> reclaim freed `urb` object
- When `CompleteURB` function callback is fired -> call our function

2024 bug: Use-after-free

CVE-2024-22269 CFG bypass

Chain functions together to bypass CFG

- Purpose ?
  - Leak `WinExec` function address
  - Call `WinExec` to pop calc

2024 bug: Use-after-free

CVE-2024-22269 CFG bypass

Chain functions together to bypass CFG

```
__int64(__fastcall* __fastcall sub_140346D40(exp_struct_urb* a1))(_QWORD)
{
    exp_struct_1* exp_struct_1; // rcx
    exp_struct_3* exp_struct_3; // rdx
    __int64(__fastcall * result)(_QWORD); // rax

    exp_struct_1 = a1->exp_struct_1;
    exp_struct_3 = exp_struct_1->exp_struct_2->exp_struct_3;
    result = exp_struct_3->func;
    if (result)
        return (result)(exp_struct_1->dst, exp_struct_3);
    return result;
}
```

```
signed __int64 __fastcall sub_140065650(volatile signed __int64 *a1, signed __int64 *a2)
{
    signed __int64 result; // rax

    result = _InterlockedCompareExchange64(a1, a2[1], *a2);
    *a2 = result;
    return result;
}
```

Function 1: Control parameter a1 and a2
Function 2: Exchange value in a1 and a2

-> write-what-where primitive

But it requires write data to a known address
We got it, when we exploit the leak bug

2024 bug: Use-after-free

CVE-2024-22269 CFG bypass

Chain functions together to bypass CFG

```
__int64 __fastcall GuestRPCHandler_tools_pkg_version(
        __int64 a1,
        __int64 a2,
        __int64 a3,
        int a4,
        __int64 a5,
        __int64 a6)
{
  if ( a4 )
    return GuestRpc_SetResult(a5, a6, "No argument expected", 0LL);
  if ( !qword_140D968D0 )
  {
    qword_140D968D0 = sub_1406A5FF0(0LL, "%d", 12389LL);
    if ( !qword_140D968D0 )
      return GuestRpc_SetResult(a5, a6, "Failed allocation", 0LL);
  }
  v7 = sub_1400EE960();
  v8 = "0";
  LOBYTE(v9) = 1;
  if ( v7 )
    v8 = qword_140D968D0;
  return GuestRpc_SetResult(a5, a6, v8, v9);
}
```

backdoor handler function
"`vmx.capability.tools_pkg_version`"

read data in a pointer store at `qword_140D968D0`

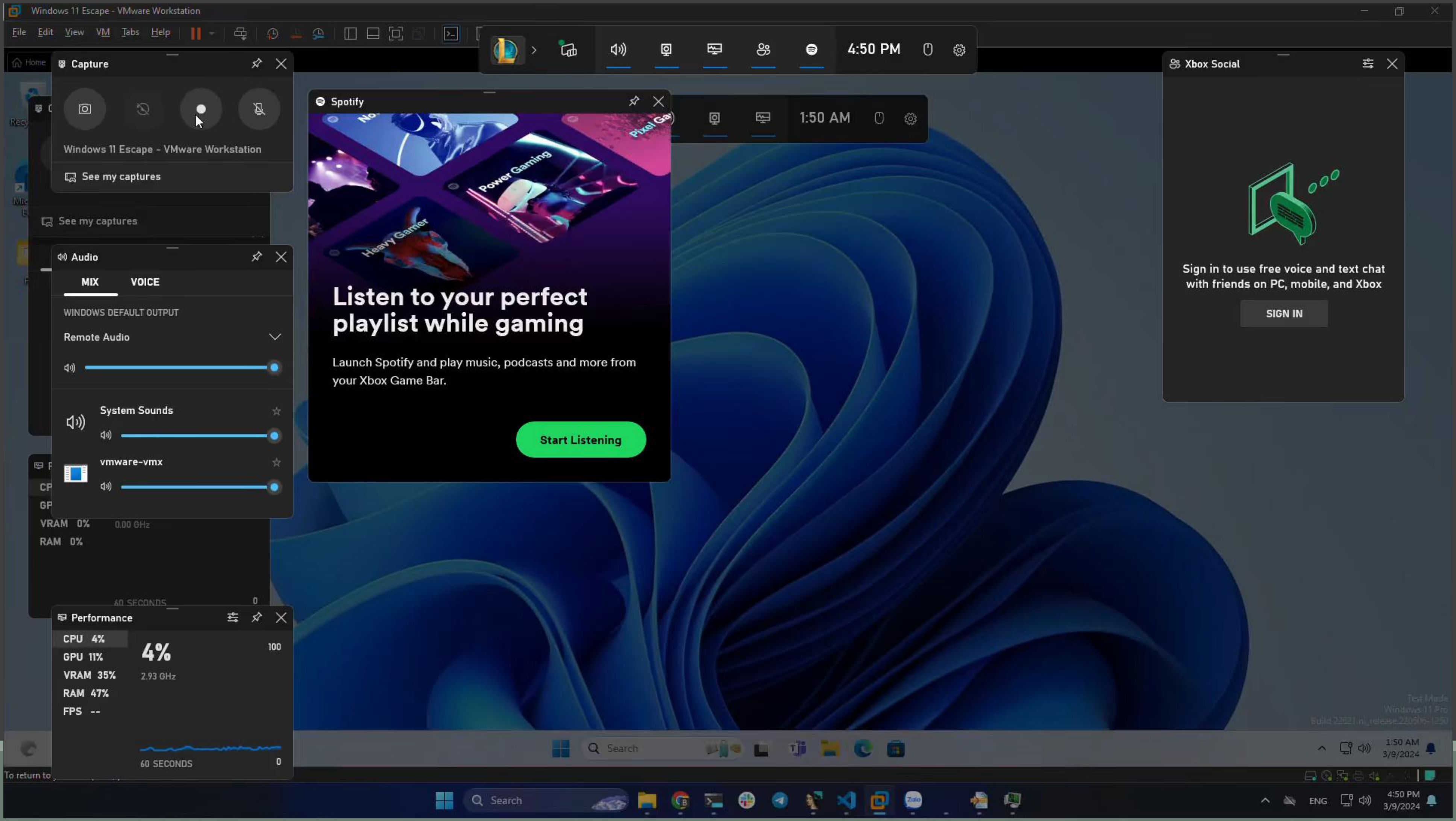
-> write-what-where -> read-what-where

We can bypass CFG now:

1. leak `WriteFile` function address
2. Calculate `WinExec` function address
3. Trigger uaf again to call `WinExec` to pop calc

2024 bug: Use-after-free

CVE-2024-22269 Demo



Summary

Summary

1. New Attack surface:
   - Virtual Bluetooth Device
2. Bugs pattern:
   - Memory uninitialized leads to memory disclosure
   - UAF when schedule callback function
3. Exploit tips
   - Experiment more to find the most stable method
   - CFG bypass method

Thanks for listening

Nguyen Hoang Thach (@hi_im_d4rkn3ss)

STAR Labs SG Pte. Ltd.