Guest Revolution

Our Story of Compromising the Host Kernel from the VMware Guest

Junoh Lee & Gwangun Jung

theori

Index

1. Introduction

theori

Who are we ?

theori

Junoh Lee

Researcher

@bbbig12

Gwangun Jung

Researcher

@pr0ln

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:

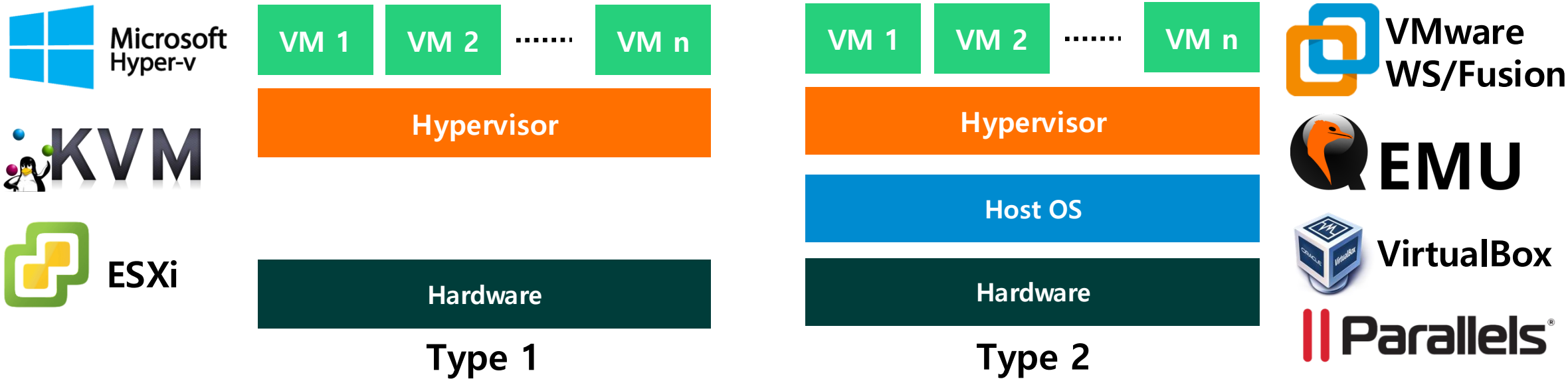
Add-on Prize	Prize	Master of Pwn Points
Escalation of privilege leveraging a Windows kernel vulnerability on the host operating system.	$50,000	5

2. Pwning VMware Workstation

theori

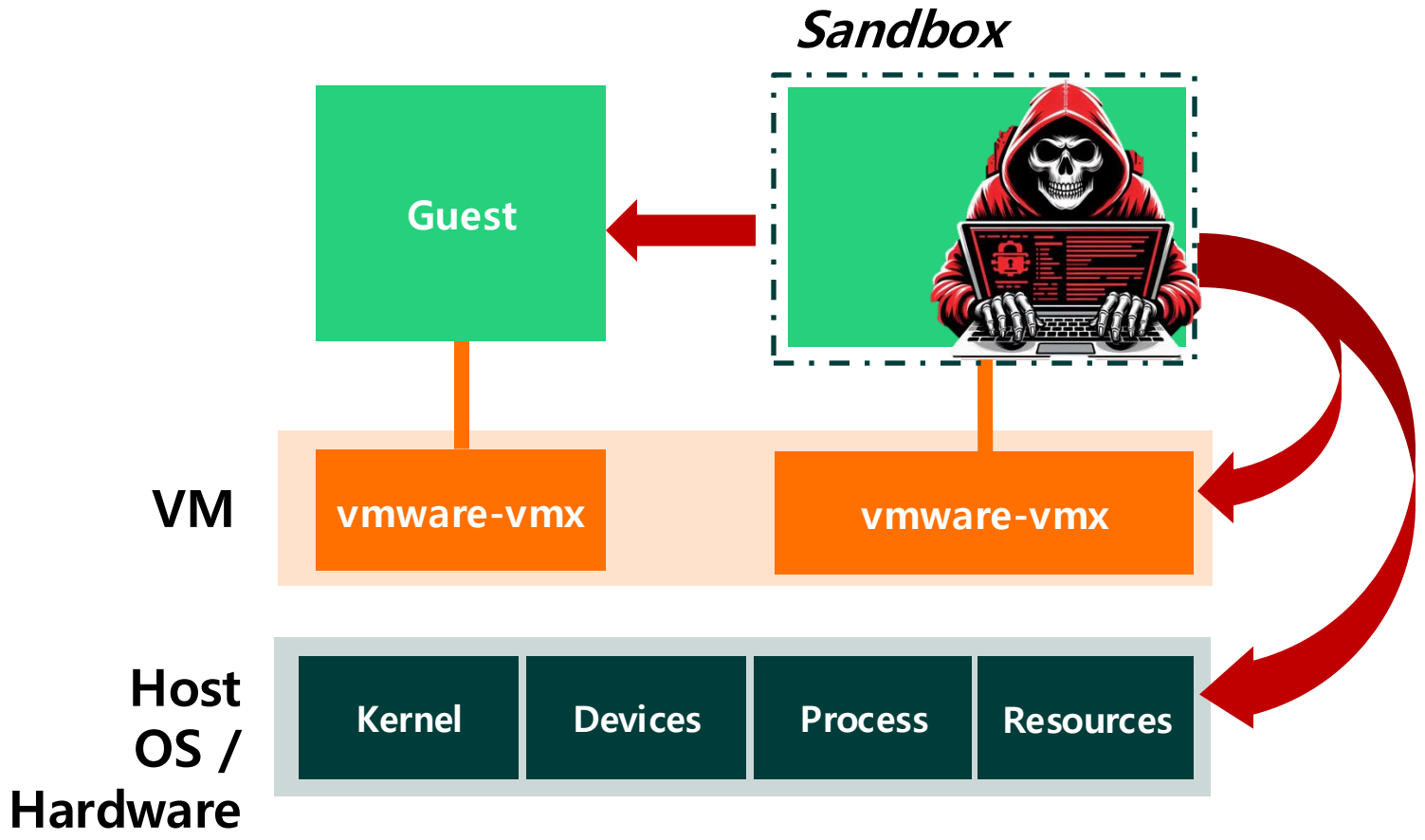
VM escape overview

Type 1 and Type 2 hypervisors



VM escape overview

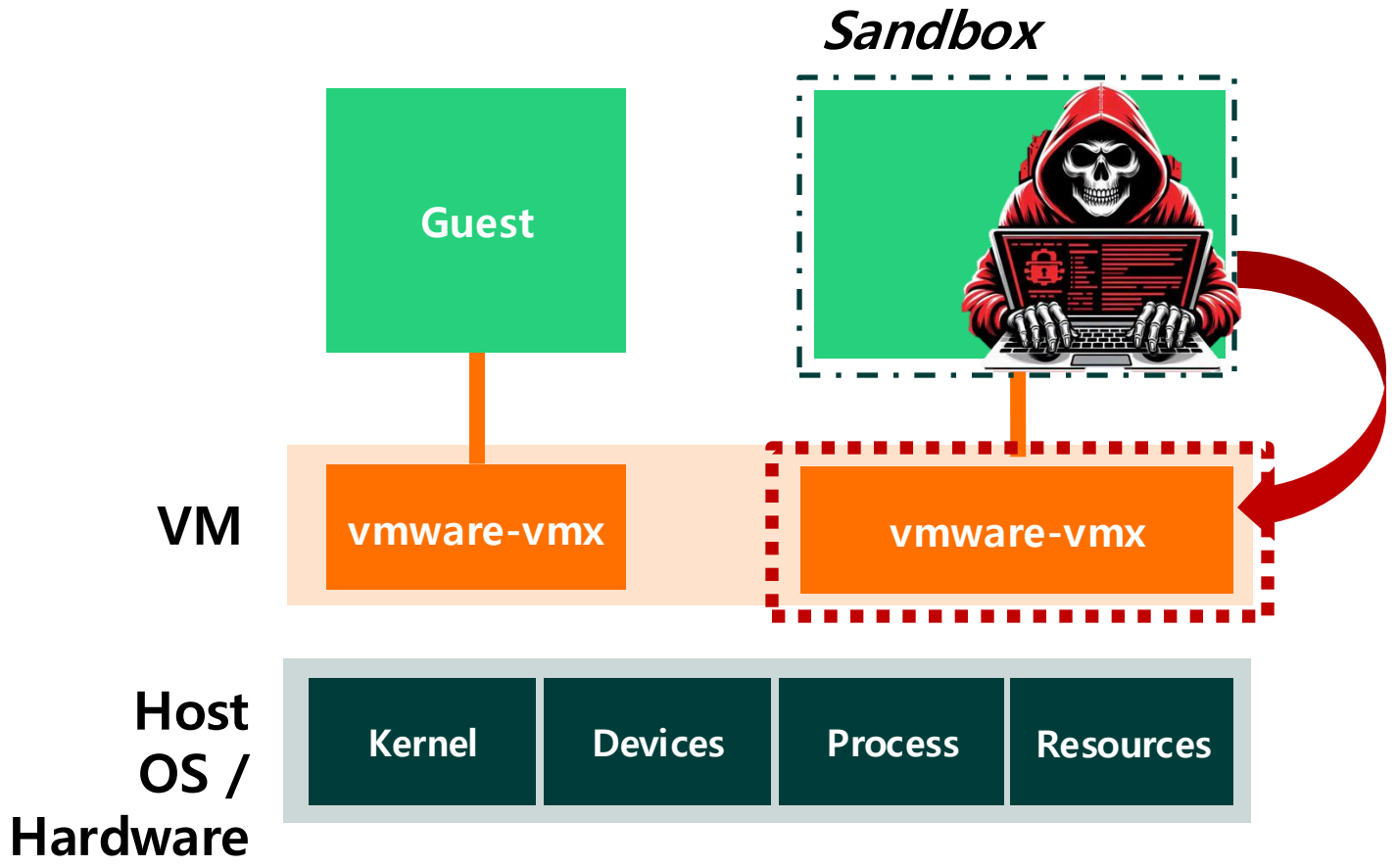
What is VM escape?



- *Code Execution*
- *File read/write*
- *Information leak*
- *Denial of Service*
- *Etc ..*

VM escape overview

What is VM escape? – Our research



- *Code Execution*
- *File read/write*
- *Information leak*
- *Denial of Service*
- *Etc ..*

VM escape overview

VM Escape CVEs (2023~2024)

VMware VirtualBox QEMU

Virtual USB Device

- CVE-2024-22269 - Bluetooth
- CVE-2024-22267 - Bluetooth
- CVE-2023-34044 - Bluetooth
- CVE-2023-22251 - USB CCID
- CVE-2023-20870 - Bluetooth
- CVE-2023-20869 - Bluetooth

Network Device

- CVE-2024-21113 - E1000
- CVE-2024-6505 - VirtIO
- CVE-2024-4693 - VMXNET3
- CVE-2023-6693 - VirtIO
- CVE-2023-4387 - VirtIO
- CVE-2023-3567 - Net

USB Controller

- CVE-2024-22255 - UHCI
- CVE-2024-22252 - XHCI
- CVE-2024-21121 - OHCI
- CVE-2023-21990 - OHCI
- CVE-2023-21989 - OHCI
- CVE-2023-21989 - EHCI

Disk Controller & Disk

- CVE-2024-21112 - AHCI
- CVE-2024-22273 - SCSI
- CVE-2024-20872 - SCSI
- CVE-2023-42467 - SCSI
- CVE-2023-4135 - NVMe

Graphic

- CVE-2024-22268 - SVGA
- CVE-2024-21991 - VGA
- CVE-2024-21115 - VGA

Etc

- CVE-2024-22270 - HGFS
- CVE-2023-21987 - TPM
- CVE-2023-21988 - GPA

.....

VM escape overview

VM Escape CVEs (2023~2024)

VMware VirtualBox QEMU

Virtual USB Device

- CVE-2024-22269 - Bluetooth
- CVE-2024-22267 - Bluetooth
- CVE-2023-34044 - Bluetooth
- CVE-2023-22251 - USB CCID
- CVE-2023-20870 - Bluetooth
- CVE-2023-20869 - Bluetooth

Network Device

- CVE-2024-21113 - E1000
- CVE-2024-6505 - VirtIO
- CVE-2024-4693 - VMXNET3
- CVE-2023-6693 - VirtIO
- CVE-2023-4387 - VirtIO
- CVE-2023-3567 - Net

USB Controller

- CVE-2024-22255 - UHCI
- CVE-2024-22252 - XHCI
- CVE-2024-21121 - OHCI
- CVE-2023-21990 - OHCI
- CVE-2023-21989 - OHCI
- CVE-2023-21989 - EHCI

Disk Controller & Disk

- CVE-2024-21112 - AHCI
- CVE-2024-22273 - SCSI
- CVE-2024-20872 - SCSI
- CVE-2023-42467 - SCSI
- CVE-2023-4135 - NVMe

Graphic

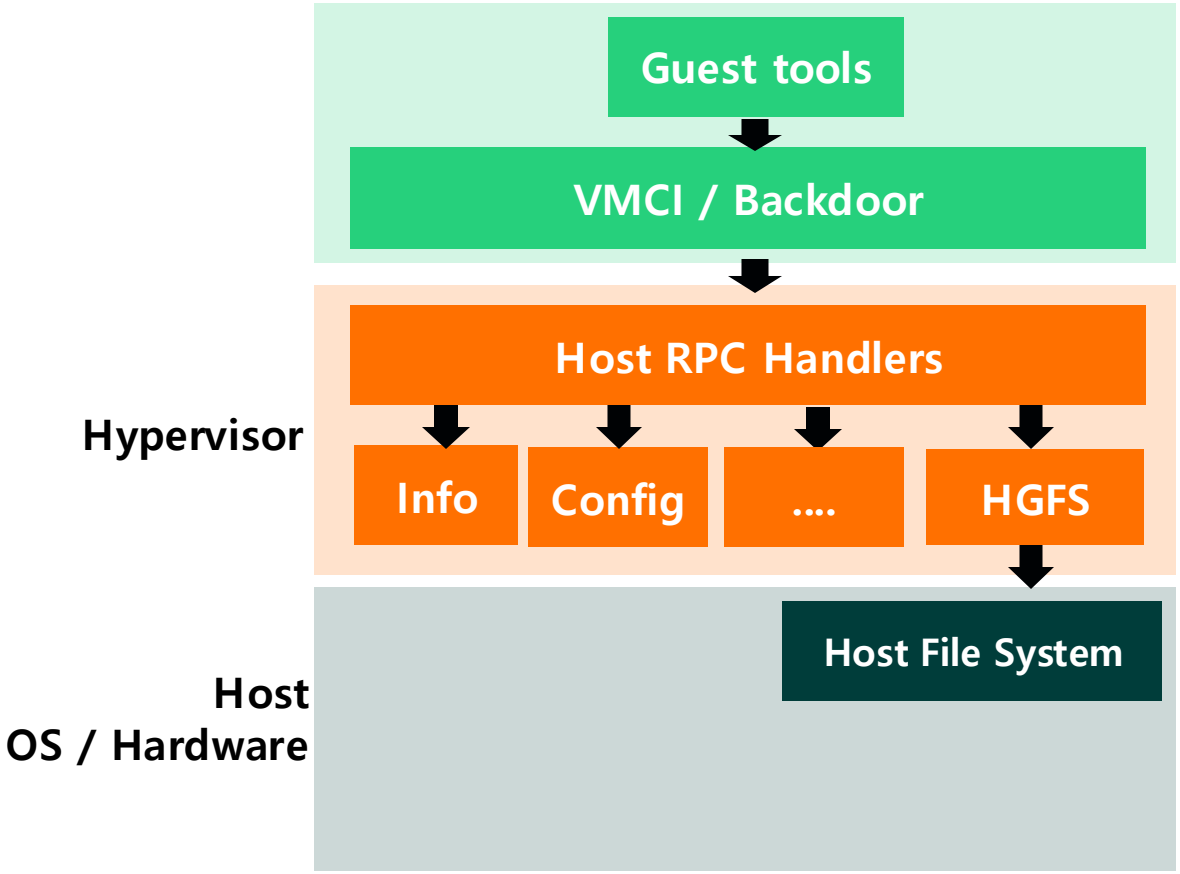
- CVE-2024-22268 - SVGA
- CVE-2024-21991 - VGA
- CVE-2024-21115 - VGA

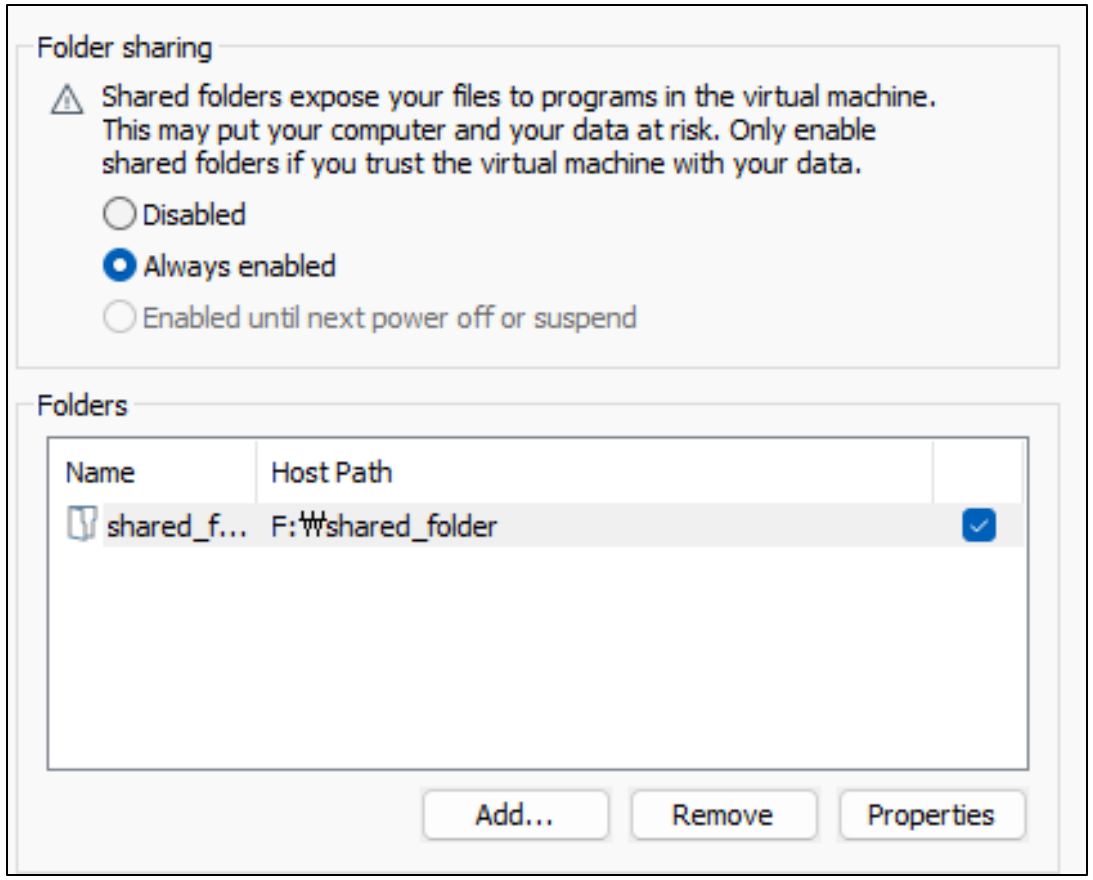
Etc

- CVE-2024-22270 - HGFS
- CVE-2023-21987 - TPM
- CVE-2023-21988 - GPA
- .....

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

Host Guest File Sharing (HGFS)



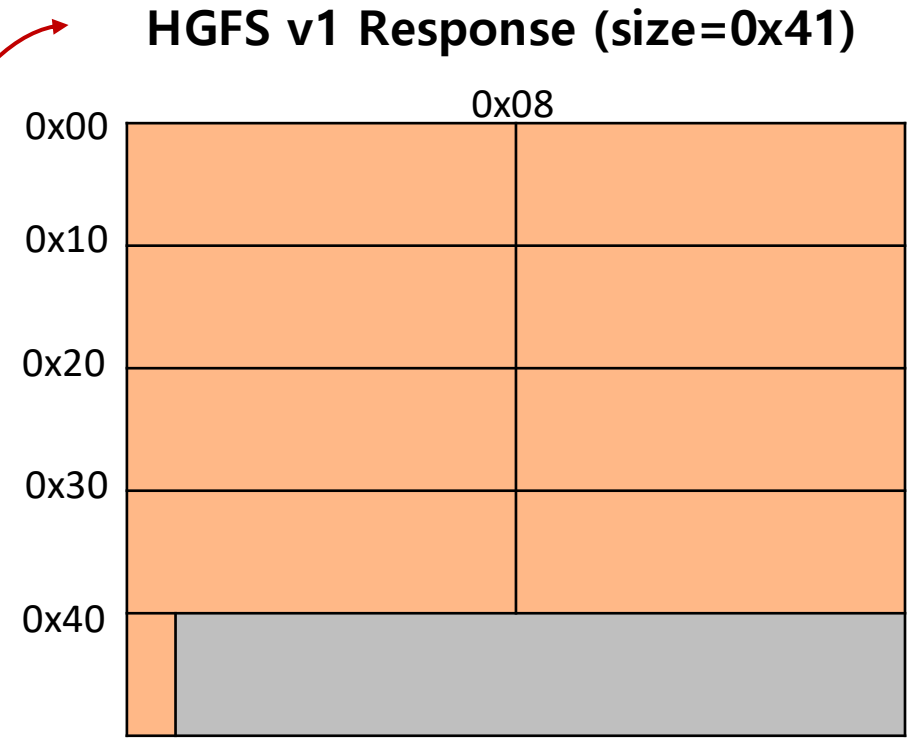


HGFS Uninitialized heap data leakage (CVE-2024-22270)

Root cause analysis (1)

```
char hgfs_fileread(struct hgfs_req *req, _BYTE *a2){
  Block = 0LL;
  rep = 0LL;
  // ...
  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 = 0LL;
    // Allocate a HGFS Response buffer
    resp = _malloc(data_size + 0x18);
```



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

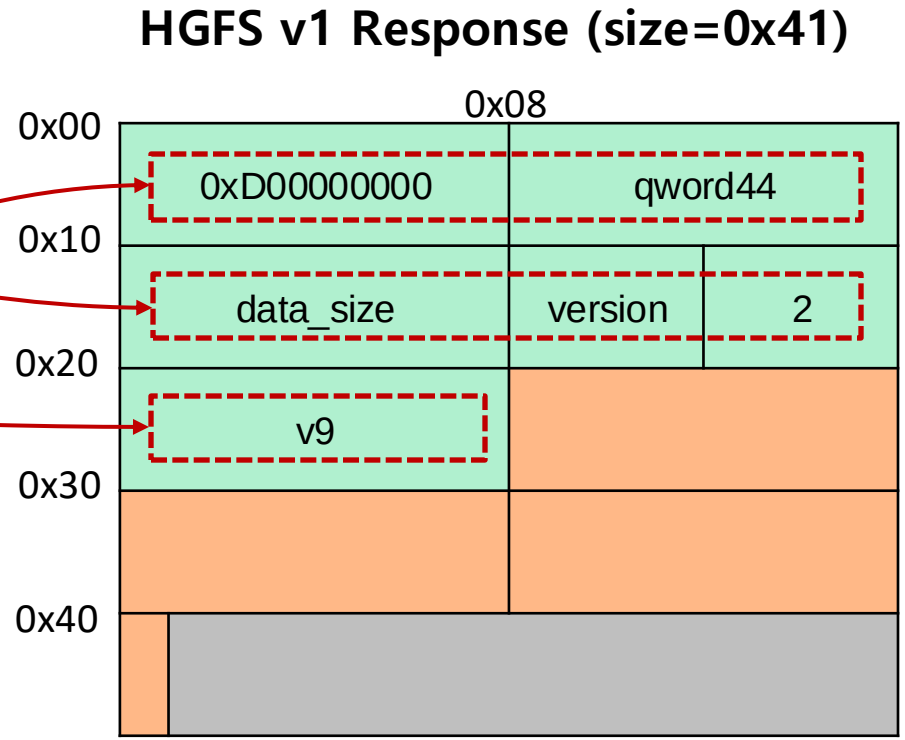
HGFS Uninitialized heap data leakage (CVE-2024-22270)

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.



25 bytes of heap are not initialized!!

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

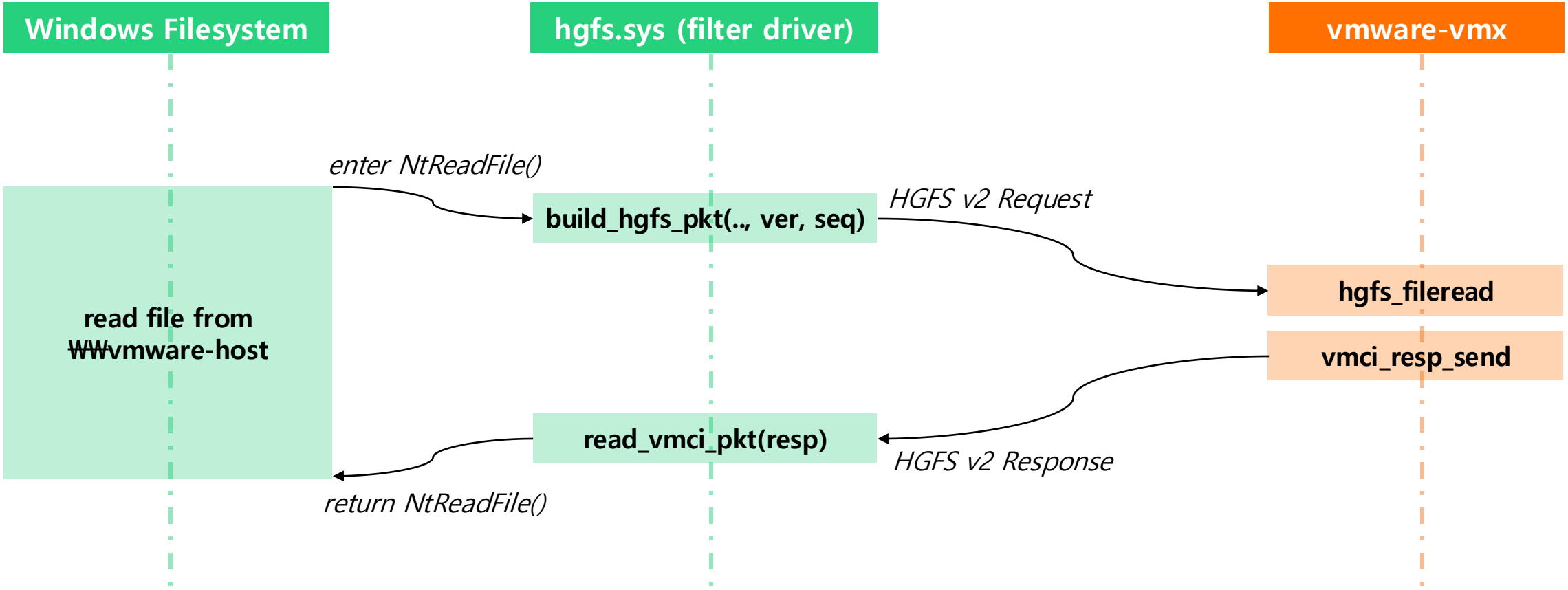
Trigger and exploit in Windows guest

- **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**

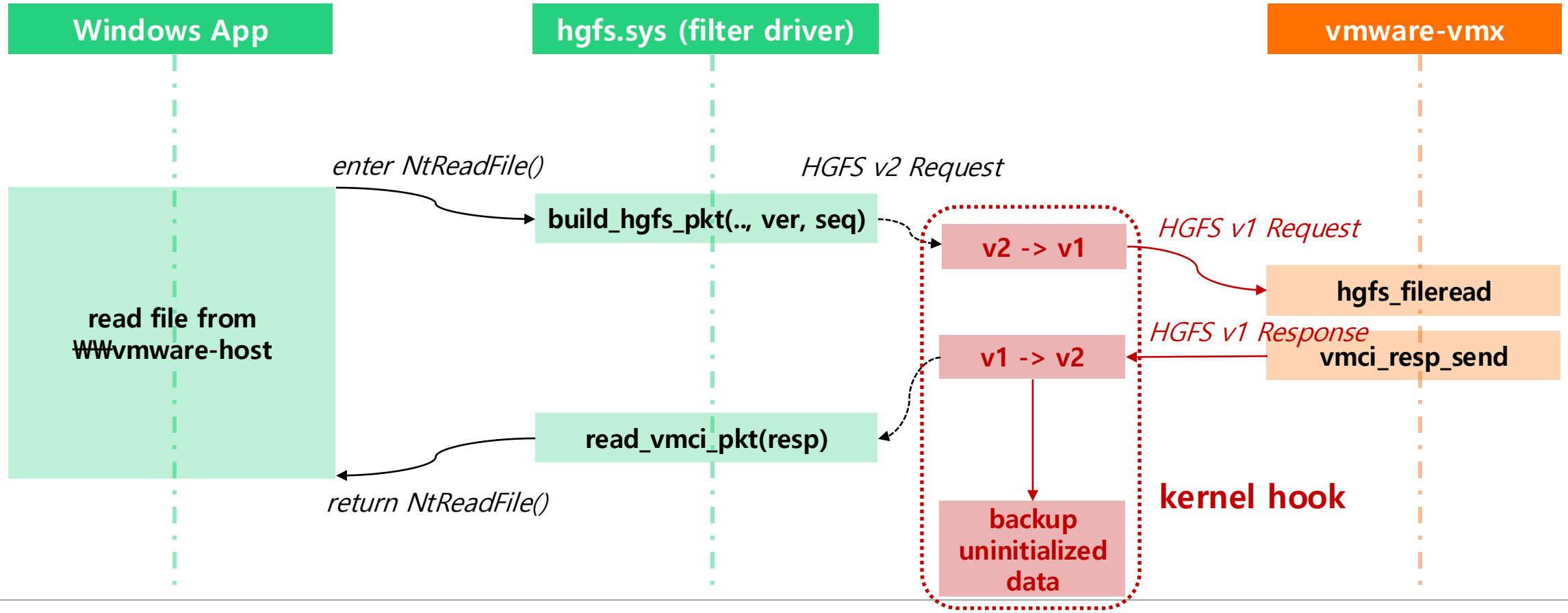
HGFS Uninitialized heap data leakage (CVE-2024-22270)

Trigger and exploit in Windows guest



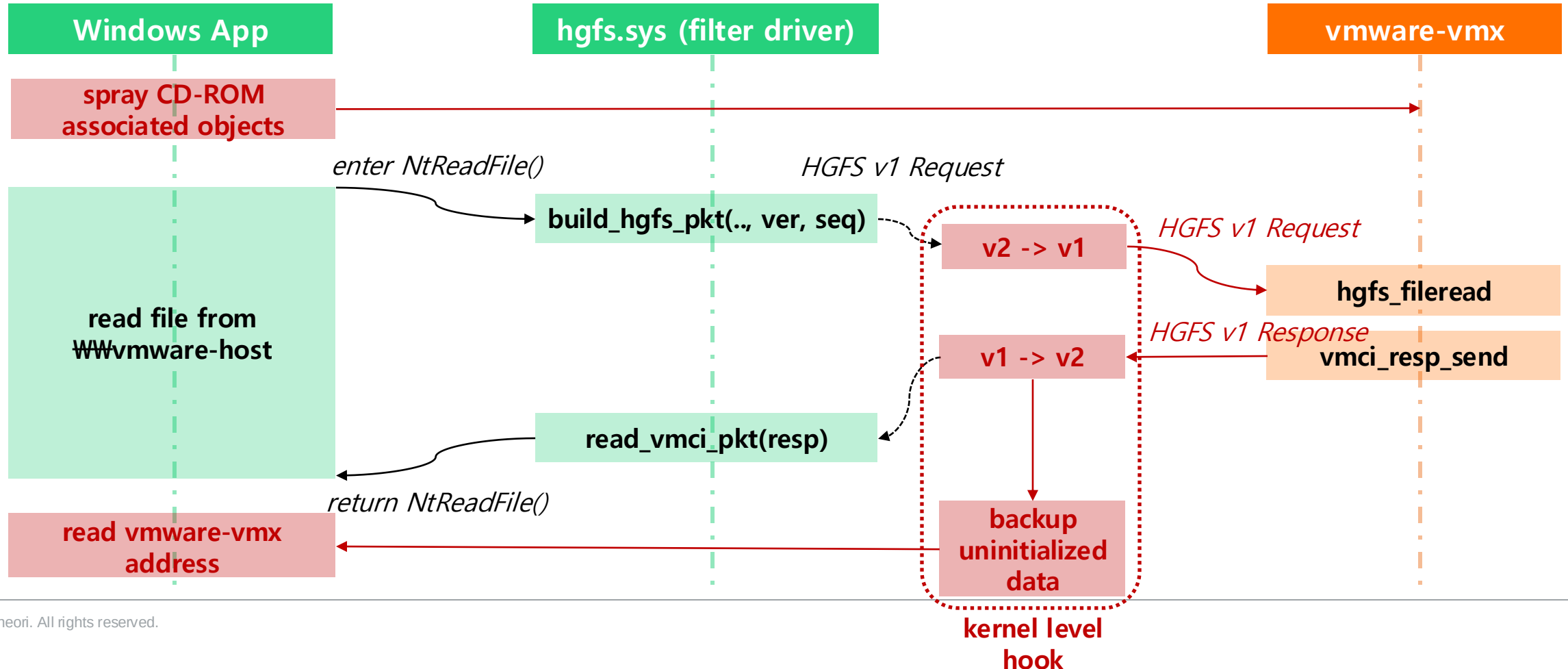
HGFS Uninitialized heap data leakage (CVE-2024-22270)

Trigger and exploit in Windows guest



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

Trigger and exploit in Windows guest



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

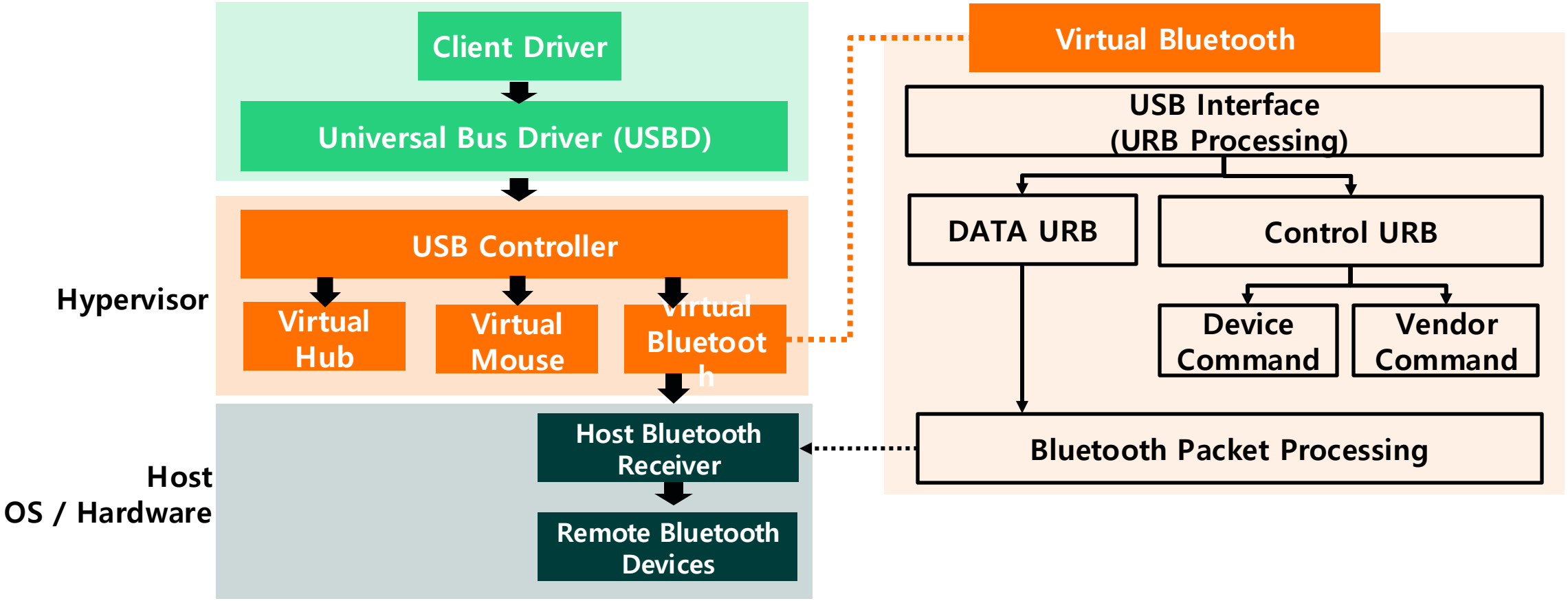
Trigger and exploit in Windows guest

```
[+] Patch VM drivers to trigger
  [-] Patch vmhgfs.sys Offset 0x1000
  [-] Patch vmhgfs.sys Offset 0xabb0
  [-] Patch vmhgfs.sys Offset 0x1053
  [-] Patch vmhgfs.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 😍

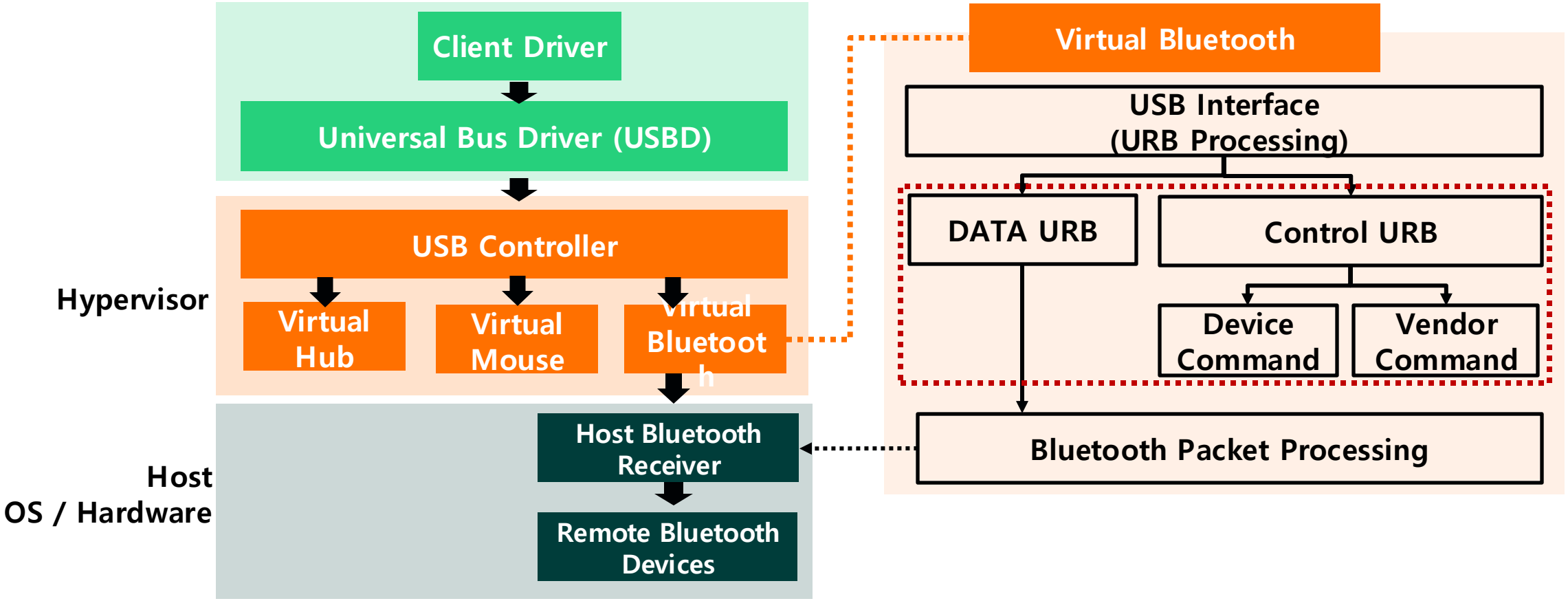
VBluetooth URB Use-After-Free (CVE-2024-22267)

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);
 }

 return VUsbQueue_HandleURBs(pool);
}
```

No increase the reference counter of URB object

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

Root cause analysis (2)

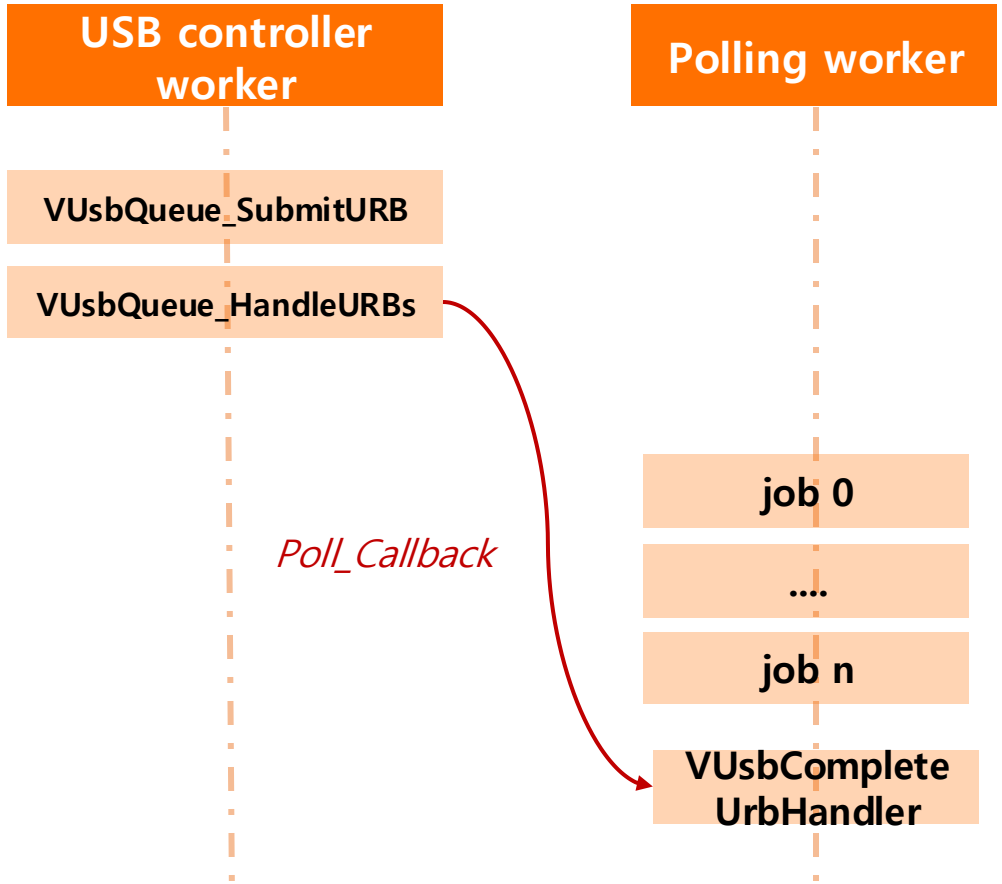
```
rbufQueue VUsbQueue_HandleURBs(pools *pool){
 // ...
 curr_node = pool->head;
 if ( curr_node ){
  while ( 1 ){
   urb = curr_node->pUrb;
   // handle a urb object

   PooledLinkList_FreeNode(curr_node, pool);

   Poll_Callback(1, 2, VUsbCompleteUrb, urb, ...);

   curr_node = (rbufQueue *)pool->head;
   if ( !curr_node )
    return curr_node;
  }
```

Register a job to polling queue

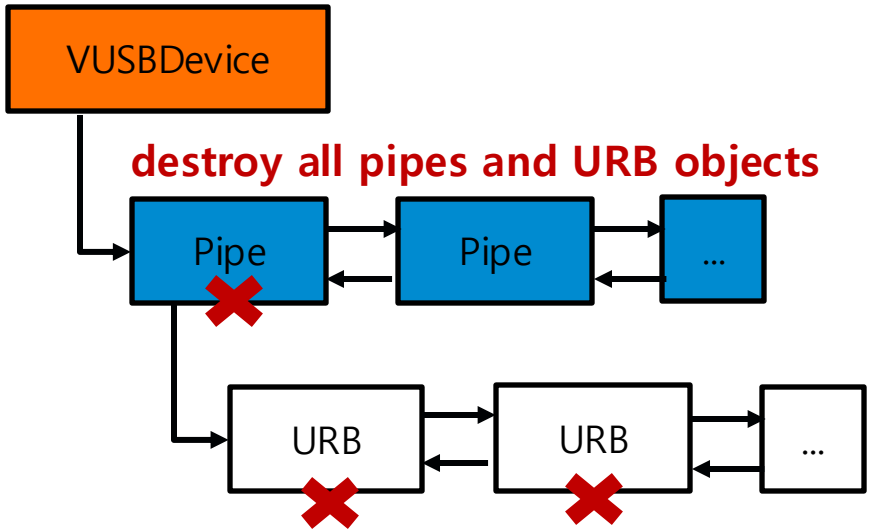


VBluetooth URB Use-After-Free (CVE-2024-22267)

Reset the USB device

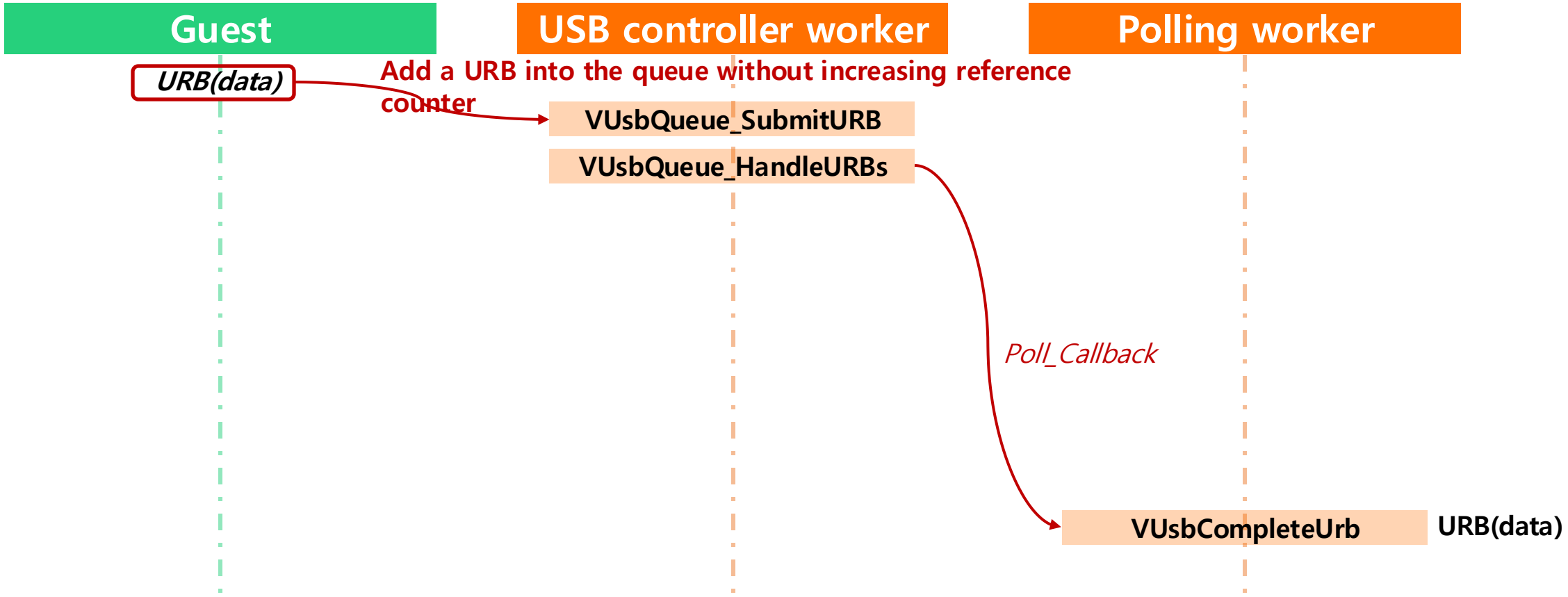
```
_int64 __fastcall VBluetooth_SubmitUrb(vurb *urb)
{
 // ...
 if ( endpointAddress )
 {
  /* handle a non control URB and return */
 }
 /* handle a control URB */
 if ( (urbData->bmRequestType & 0x60) != 0x20 )
 {
  // ...
  if ( requestType == 9 )
  {
   if ( urbData->wValue <= 1u )
   {
    sub_1407BA7E0(urb->pipe->dev,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



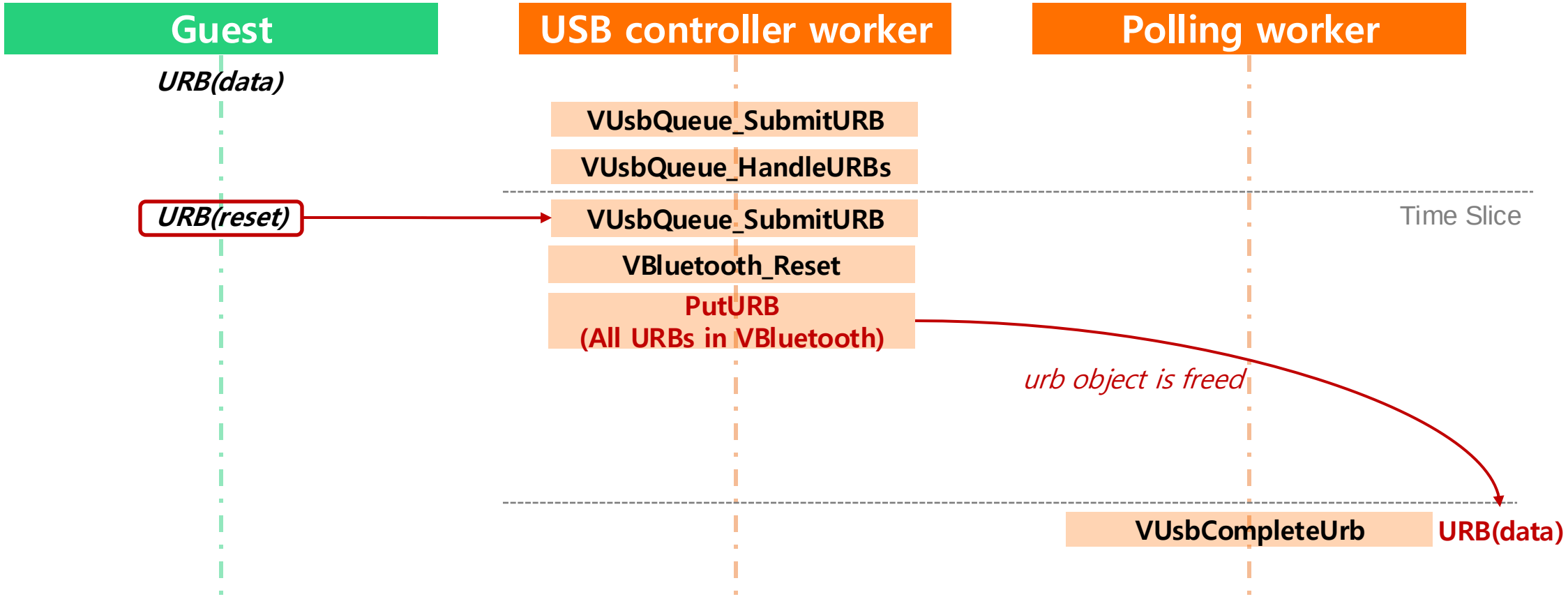
VBluetooth URB Use-After-Free (CVE-2024-22267)

Trigger Use-After-Free (1)



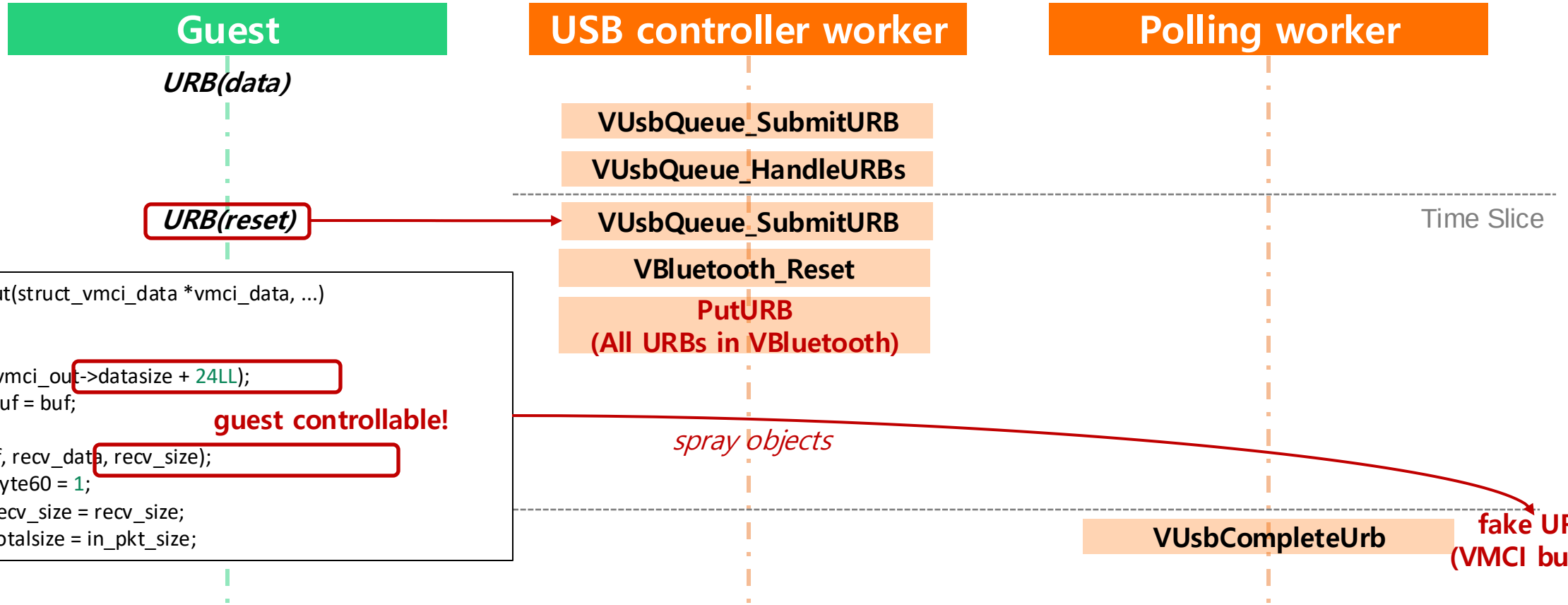
VBluetooth URB Use-After-Free (CVE-2024-22267)

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)
{                                  called by VUsbCompleteUrb
  // ...
  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;
  }
```

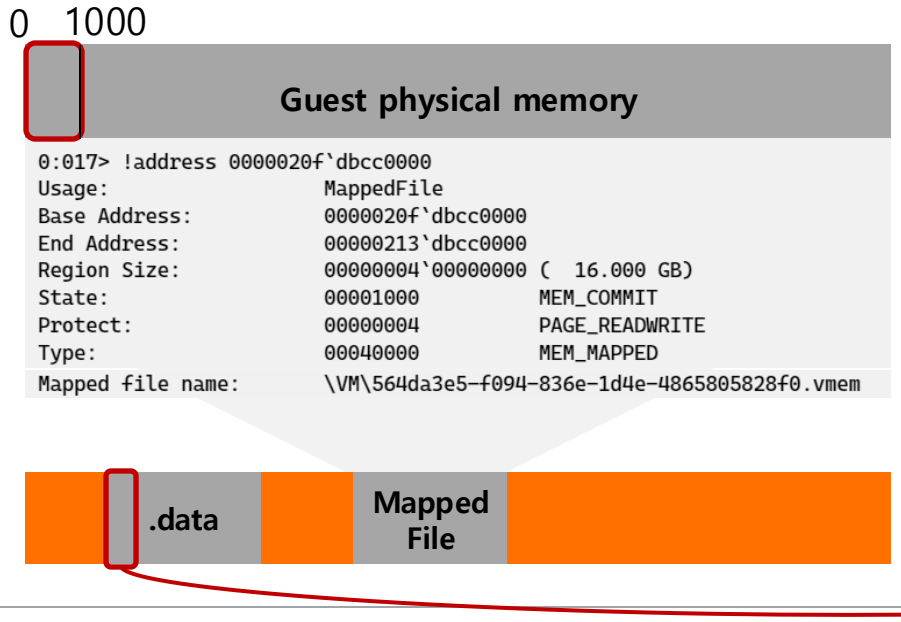
Name	Base address	Size	CF Guard
vmware-vmx.exe	**0x7ff6e9b...**	**28.0...**	**CF Guard**
advapi32.dll	0x7ffe54600...	712 kB	CF Guard
cfgmgr32.dll	0x7ffe53850...	312 kB	CF Guard
crypt32.dll	0x7ffe53cd0...	1.4 MB	CF Guard
dsound.dll	0x7ffd93550...	648 kB	CF Guard
libcrypto-3-x64.dll	0x7ffd635c0...	5.01 MB	
libssl-3-x64.dll	0x7ffd93480...	780 kB	
msvcp140.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**



First page of physical memory is not used after boot

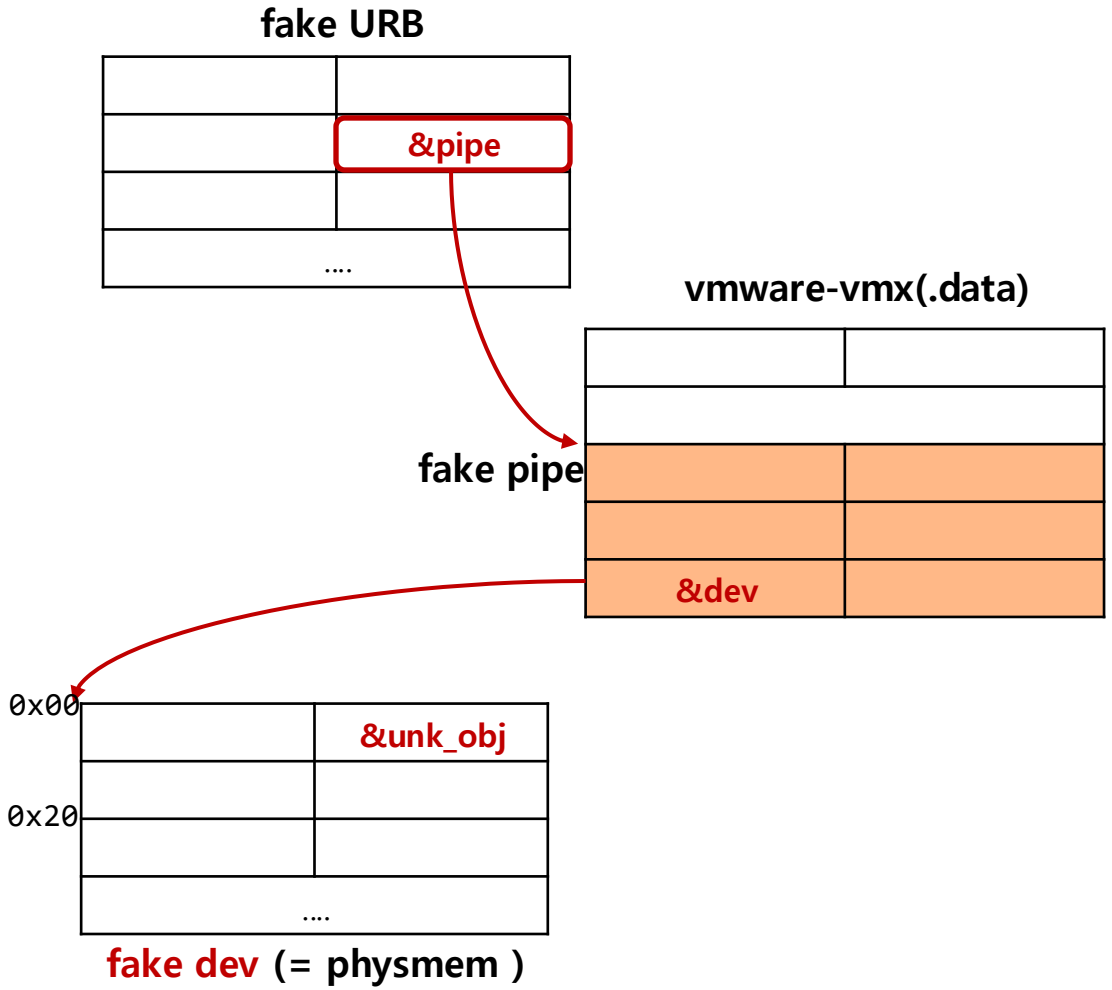
Guest' physical memory base is in .data section of vmware-vmx

```
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

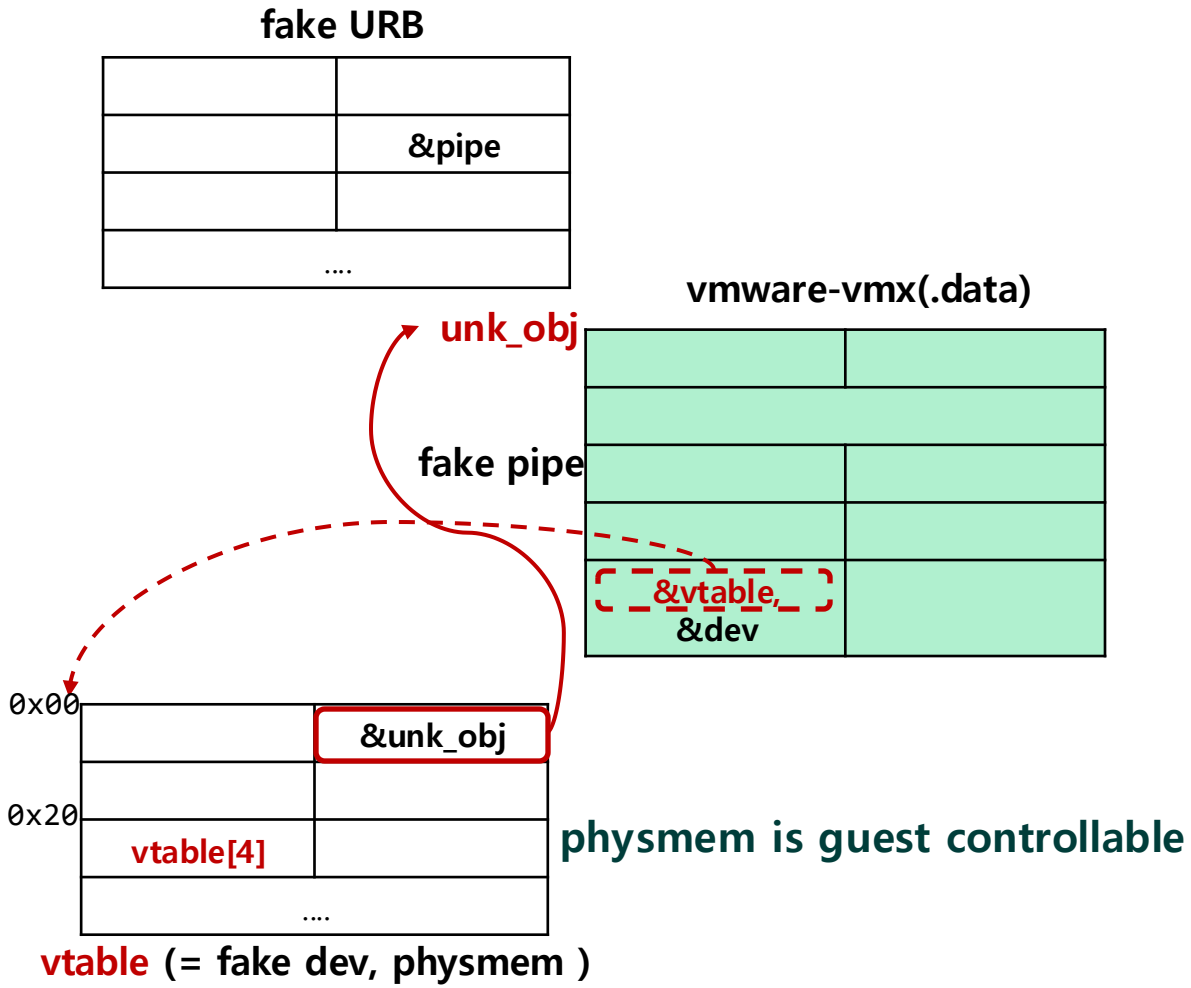
```
char __fastcall VUsb_CompleteUrbAddBatch(vurb *urb)
{
  // ...
  pipe = urb->pipe; // [urb + 18h]
  data = urb->data;
  dev = pipe->dev;  // [pipe + 20h]
  if ( unknown_flag && urb->type &&
       pipe->stalled && urb->status == 3 )
    urb->status = 4;
  if ( urb->status == 6 )
  {
    if ( urb->hcpriv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

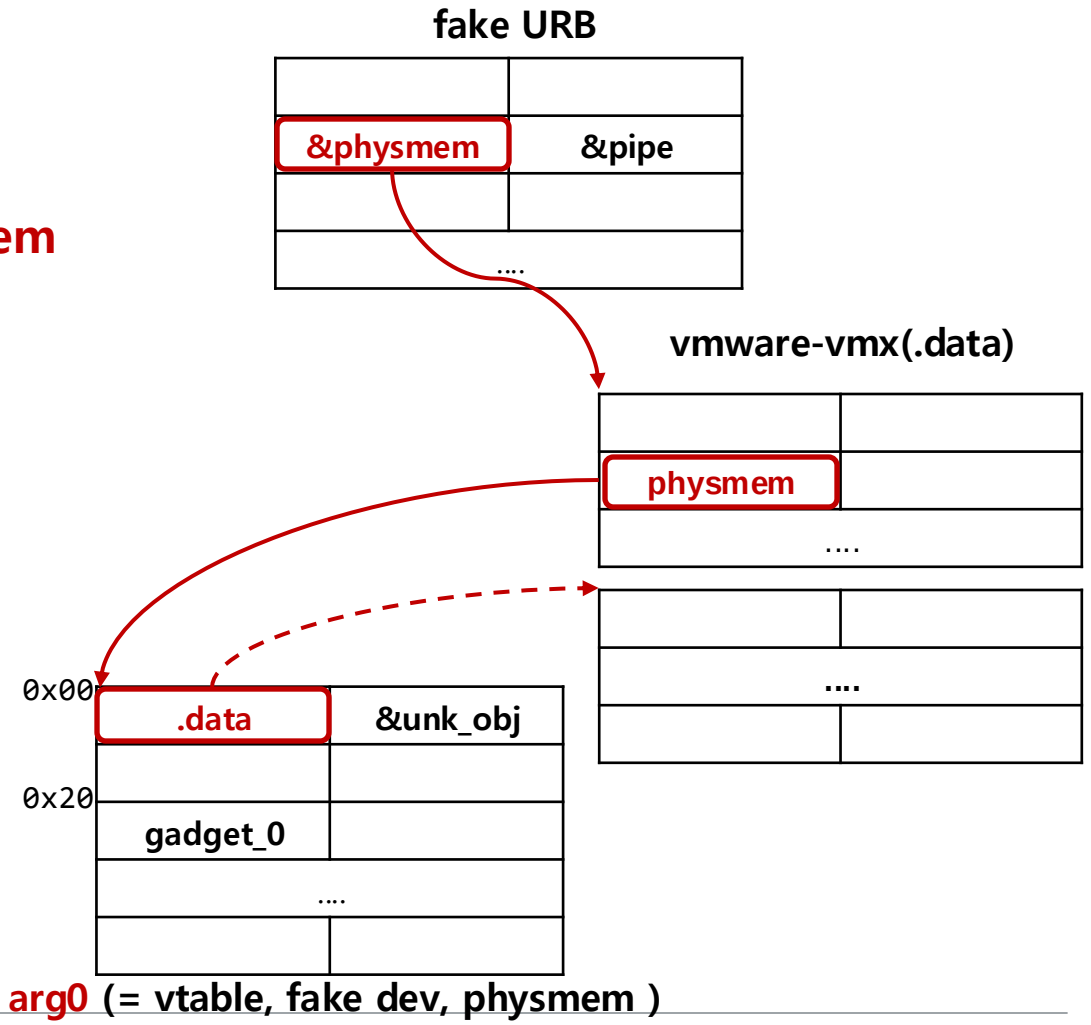
```
__int64 gadget_0(__int64 urb) {// sub_140323F50
 // mov    rax, [rcx+10h]
 // mov    rcx, [rax]

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

rcx == next arg0 == physmem

rax point to .data section

.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]

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

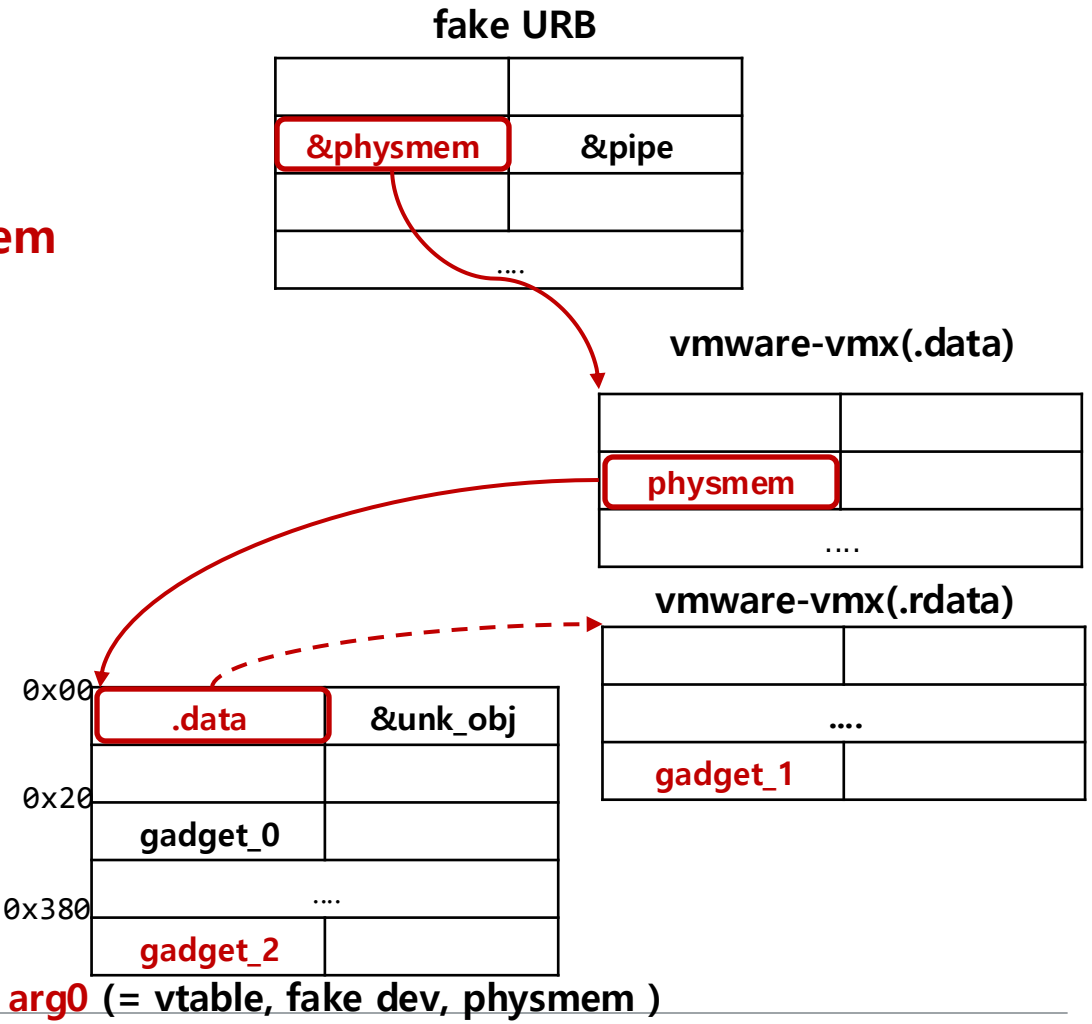
rcx == next arg0 == physmem

rax point to .data section

```
.rdata:0000000140A0D808 dq offset sub_140295230
__int64 gadget_1(__int64 phys) {// sub_140295230

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

gadget_2 = *(phys+0x380)

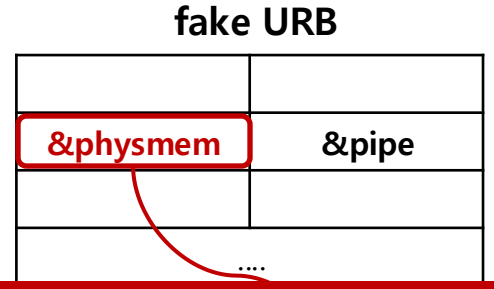


VBluetooth URB Use-After-Free (CVE-2024-22267)

Pivoting arg0 to Guest's physmem

```
__int64 gadget_0(__int64 urb) {// sub_140323F50
// mov    rax, [rcx+10h]
// mov    rcx, [rax]
```

rcx == next arg0 == physmem

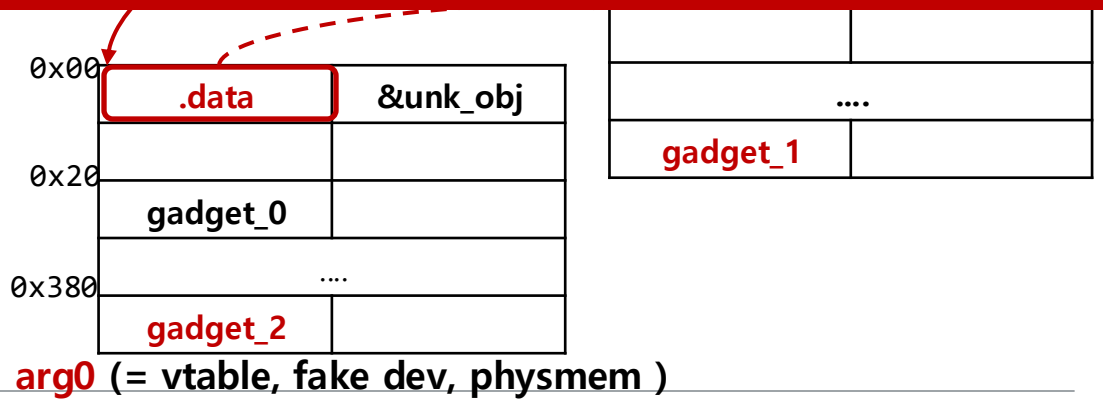


Almost data of arg0 are controllable !! 😍

```
.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)

Run Shellcode with CFG based ROP

- 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
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	

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(physmem+off, IAT+off, ...);
   - ReadPhys(physmem+off, ...) from guest
3. Call VirtualProtect(unused space, ..., PAGE_EXECUTE_READWRITE, ...);
4. Copy shellcode to unused space
   - WritePhys(physmem+off, ...) from guest
   - memcpy(unused space , physmem + off, ...);
5. Finally, jump to shellcode using **libcrypto's gadget.**

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

3. Windows Kernel Exploit

theori

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)
  - …

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

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

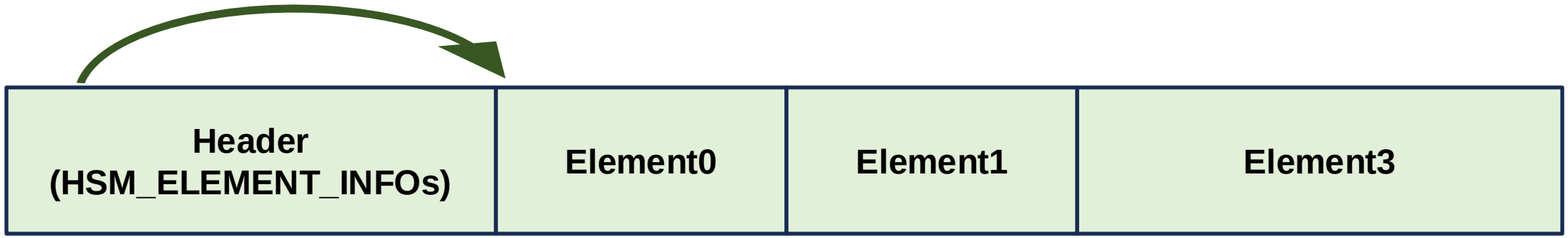
```
buffer->ReparseTag = IO_REPARSE_TAG_CLOUD_3;
...
NtFsControlFile(hDir,
                FSCTL_SET_REPARSE_POINT,
                buffer,
                buffer_len);
```

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 HsmIBitmapNORMALOpen() {
 ...
 if ( elem4_buffer && elem4_length - 1 <= 0xFFE )
 {
  ...
 }
 else
 {
  v40 = ExAllocatePoolWithTag(PagedPool, 0x1000ui64, 0x6D427348u);
  if ( v40 )
  {
   memmove(v40, elem4_buffer, elem4_length);
   goto LABEL_87;
  }
 }
```

Meaningless Condition 😕

Overflow if elem4_length > 0x1000

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 cldflt reparse point data

What Can We Do?

- Heap based buffer overflow on fixed size chunk (0x1000)
  - On paged pool

Exploit Time

```
                0i64, 0, PoolWithTag, 0x4000u, 0i64);
  ...
}
```

- We can overflow with arbitrary data
  - By setting cldflt 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...

```
__int64 __fastcall NtCreateWnfStateName(
   ...
   unsigned int MaxDataSize,
   PSECURITY_DESCRIPTOR a7)
{
  ...
  if ( MaxDataSize > 0x1000 )
  {
    StateName = 0xC000000D;
    goto ERROR;
  }
  ...
  ExpWnfCreateNameInstance(
    ...
    pMaxDataSize,
    ...);
```

```
__int64 __fastcall ExpWnfCreateNameInstance(
_WNF_SCOPE_INSTANCE *a1,
_WNF_STATE_NAME_STRUCT a2,
__int32 *MaxDataSize,
_EPROCESS *a4,
_WNF_NAME_INSTANCE **a5)
{

WNI->StateNameInfo.MaxStateSize = *MaxDataSize;
...
}
```

WNF_STATE_DATA – Big Chunk OOB Write

NtUpdateWnfStateData

```
__int64 __fastcall ExpNtUpdateWnfStateData(
  ...
  )
{
  ...
  res = ExpWnfValidatePubSubPreconditions(
    2u,
    &Instance->StateNameInfo,
    Size,
    v40,
    v34);
  if ( res < 0 )
  goto ERROR;
  ...
}
```

Write Size

```
__int64 __fastcall ExpWnfValidatePubSubPreconditions(
  ...)
{
  ...
  TypeId = StateInfo->TypeId;
  if ( !TypeId )
  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(
 ...
  v40,
  v34);
 if ( res < 0 )
 goto ERROR;
 ...
}
```

Write Size

```
int64 __fastcall ExpWnfValidatePubSubPreconditions(
...
}
```

OOB Write is not possible for chunks of size 0x1000

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,
    v36,
    v21);
  if ( res < 0 )
  goto ERROR;
  ...
}
```

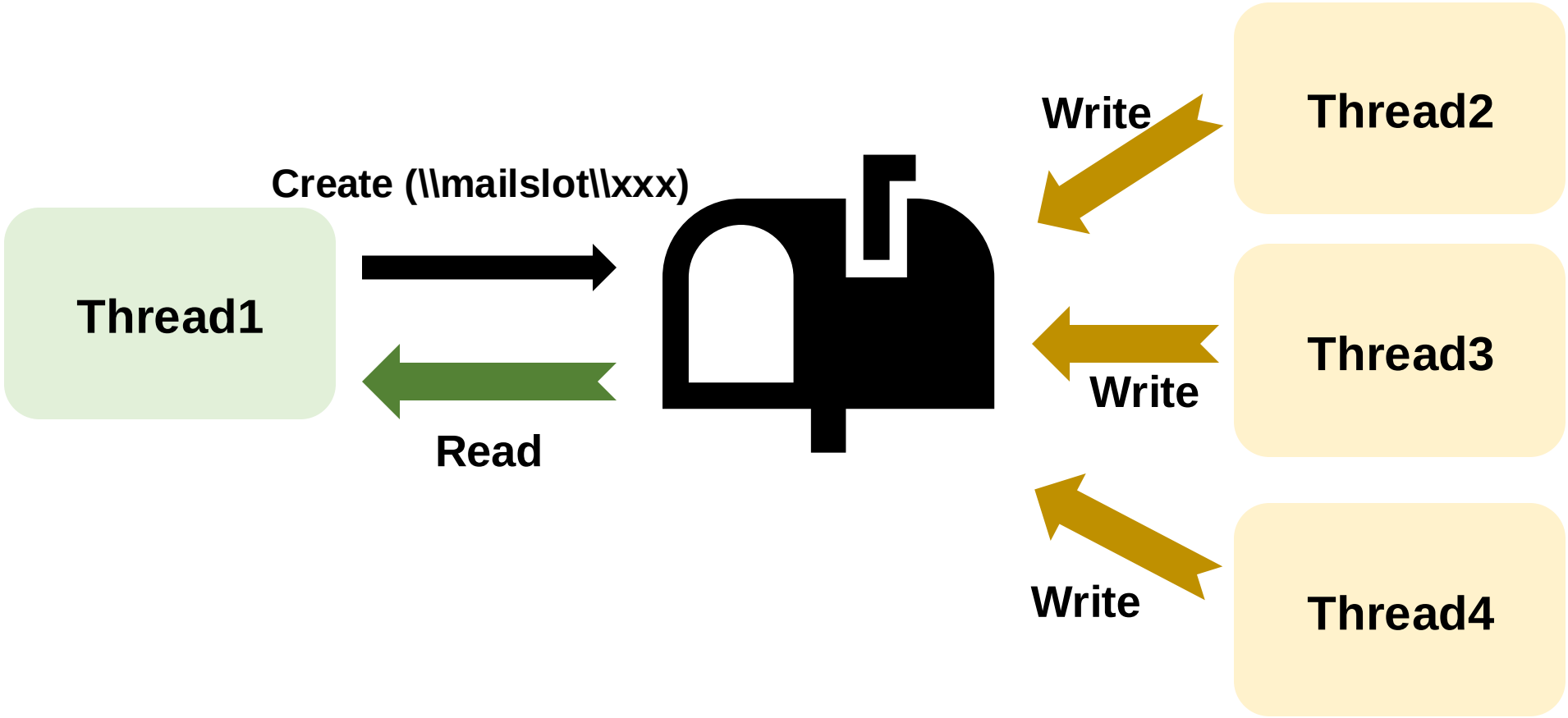
Read Size == 0 !!?

```
__int64 __fastcall ExpWnfValidatePubSubPreconditions(
  ...)
{
  ...
  TypeId = StateInfo->TypeId;
  if ( !TypeId )
  return StateInfo->MaxStateSize < Size ? 0xC000000D : 0;
  ...
}
```

Always pass the size validation

Mailslot

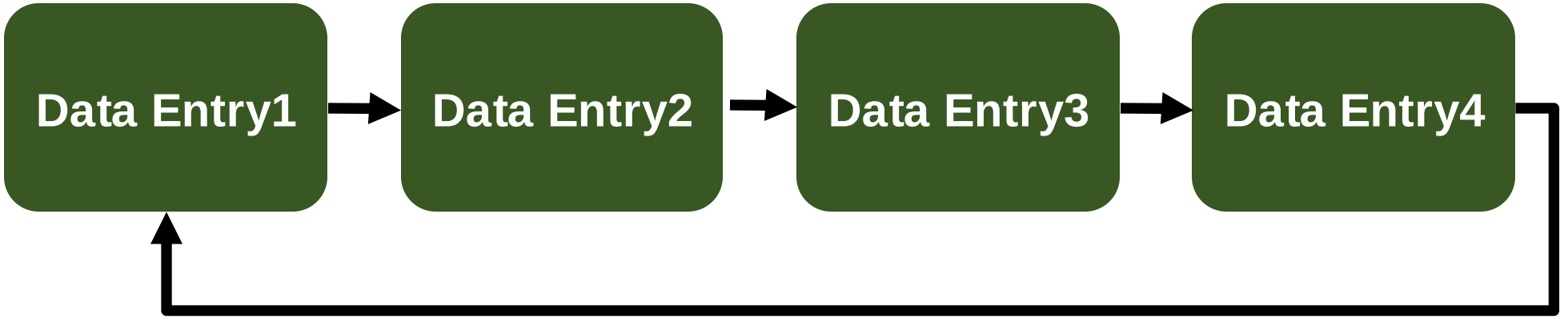
Mechanism for **one-way** interprocess communications



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

Mailslot Data Entry

Mailslot manages multiple received data using Data Entry Queue

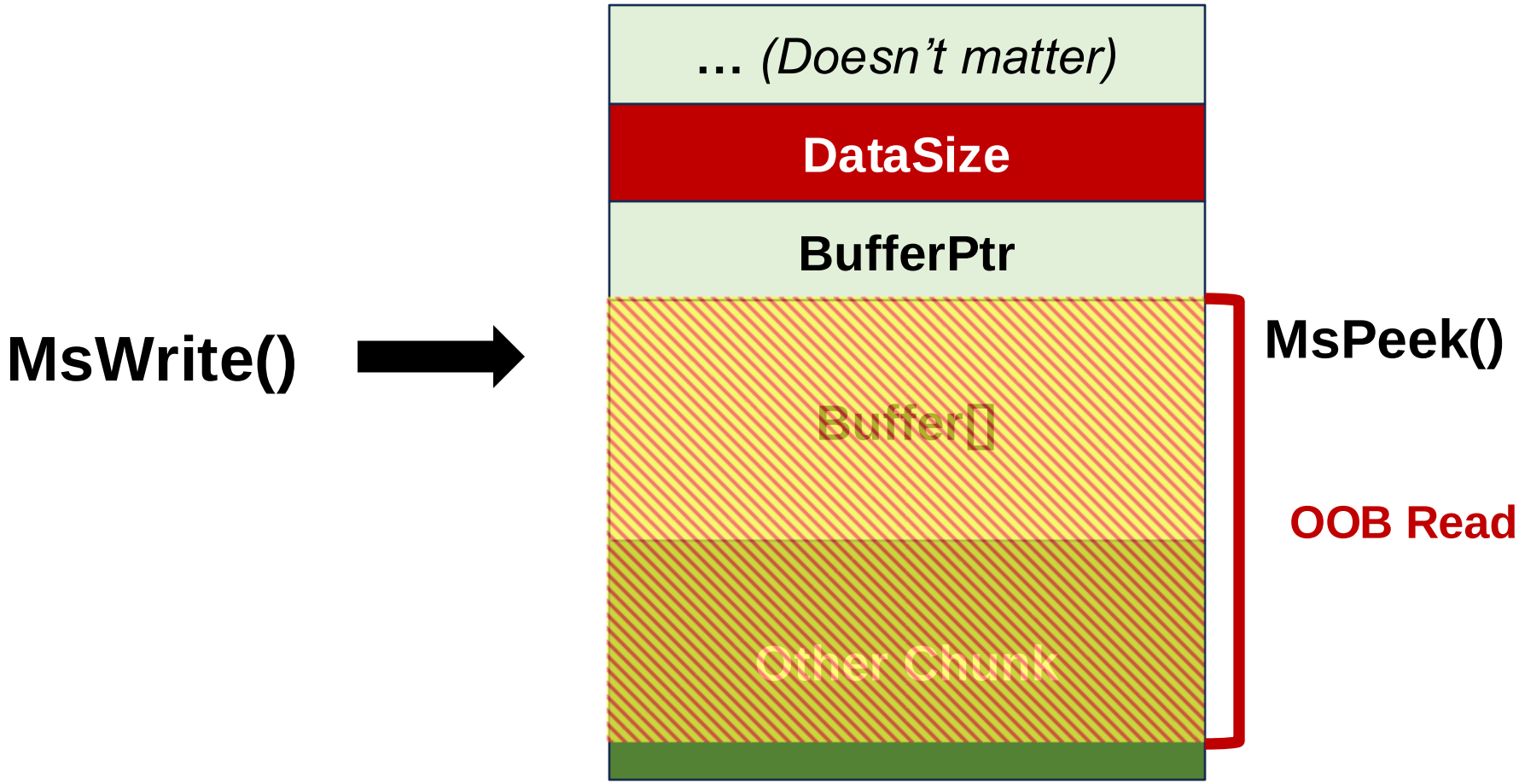


Mailslot Data Entry Object

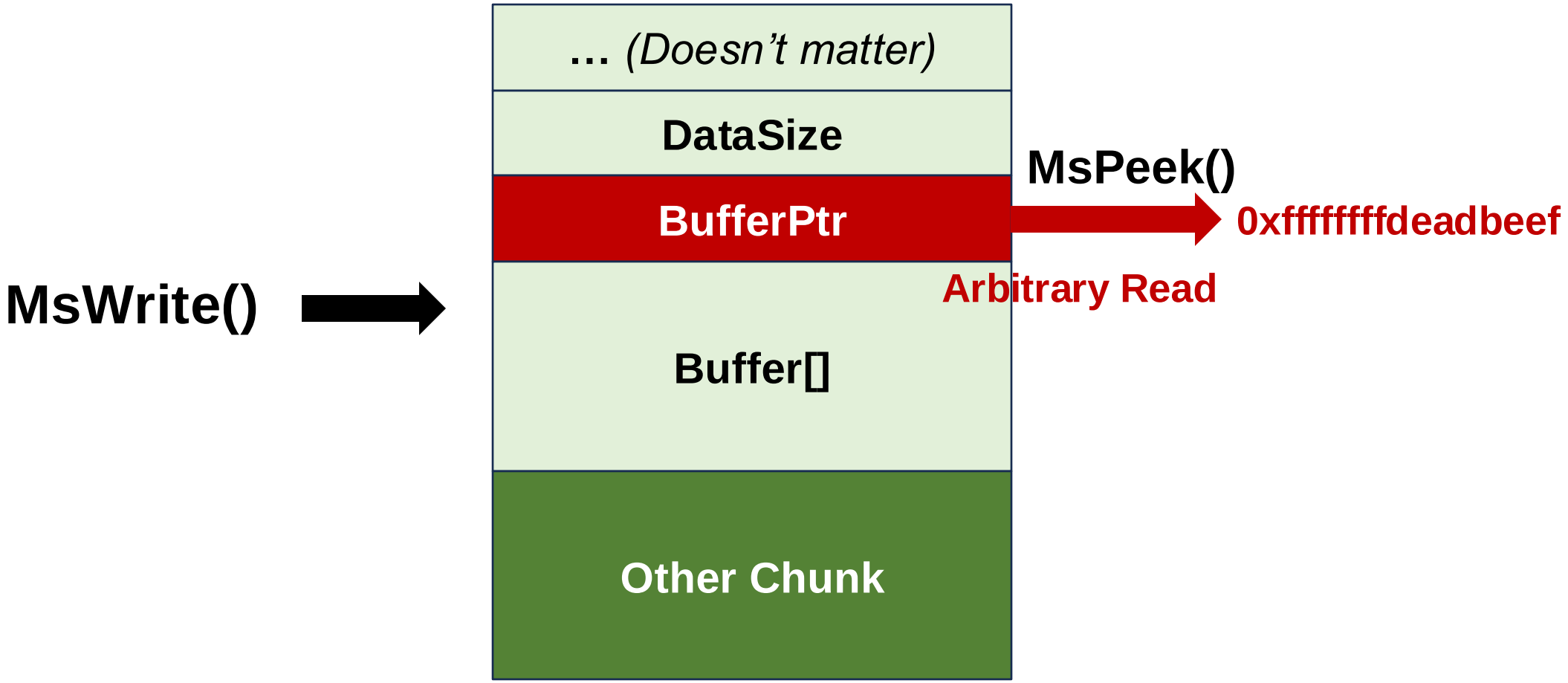
- **mutable object** and contains **useful members**
- **Sprayable Object**
  - By calling *WriteFile* multiple times from another thread
  - By creating multiple mailslot handles
- Almost **no size limit** (< 4GB)
- Allocated in **paged pool**

```
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[];
};
```

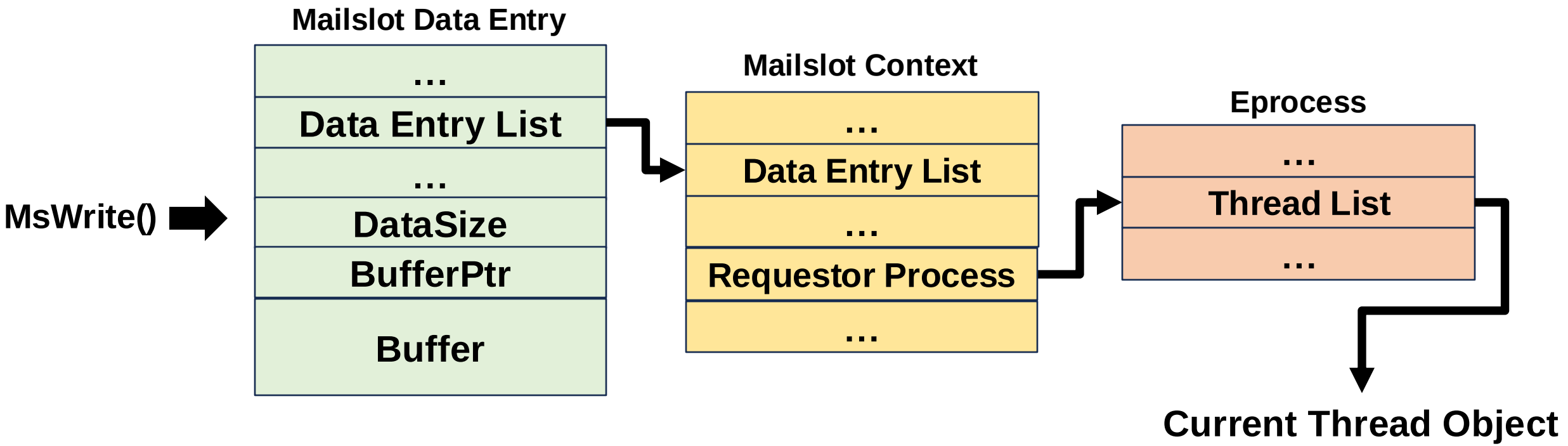
Out-of-Bound Read



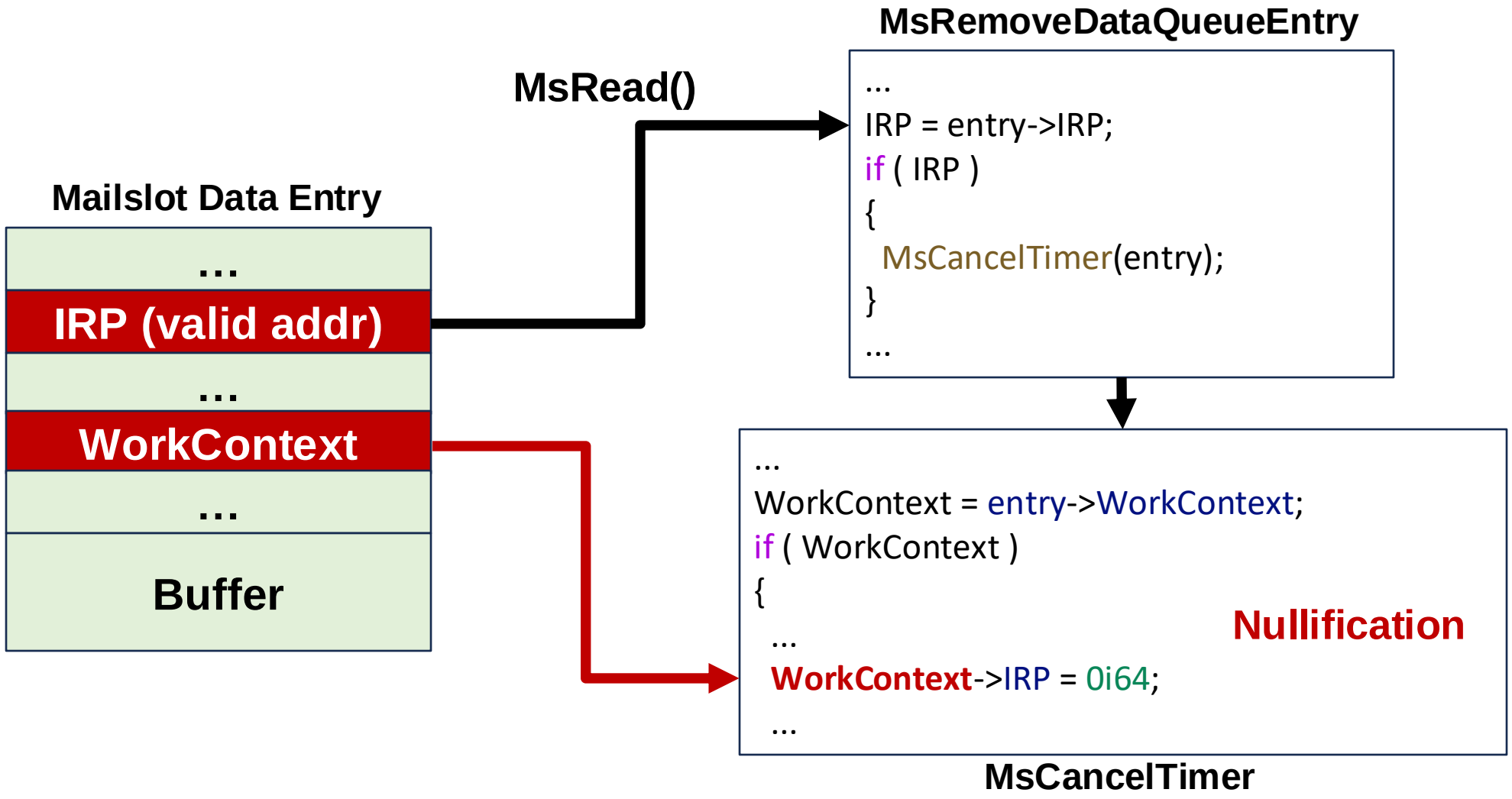
Arbitrary Read



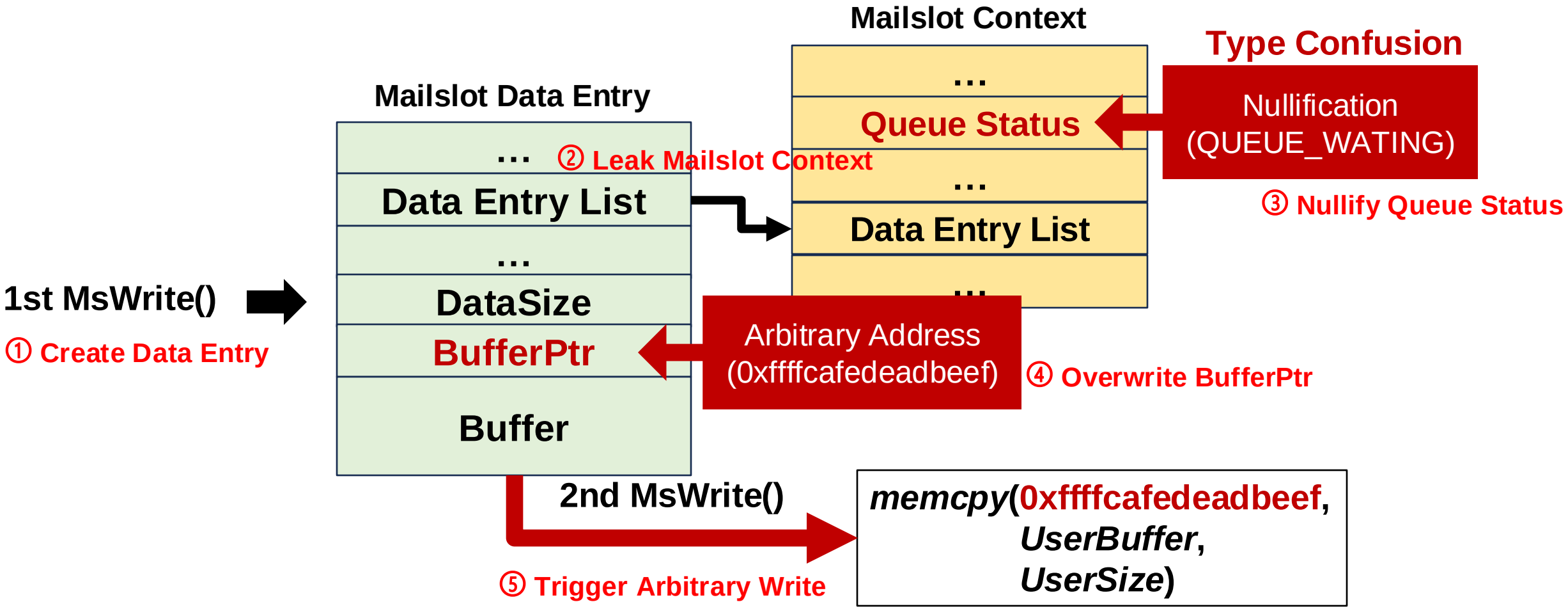
Leaking Critical Object Address



Arbitrary Nullification

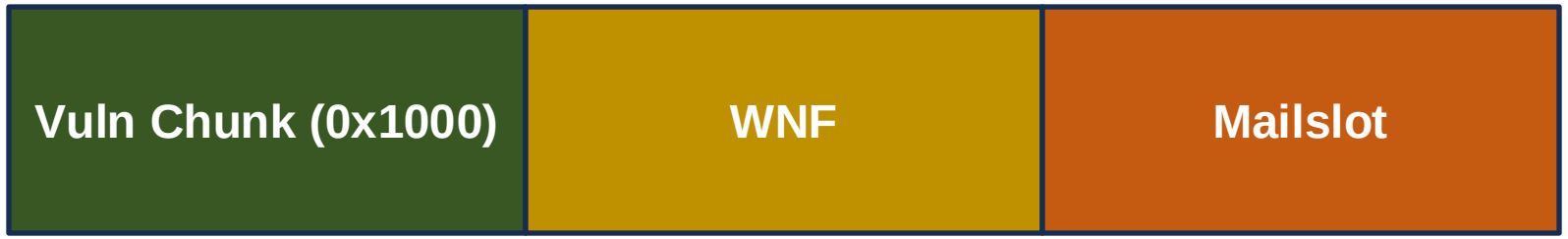


Arbitrary Write

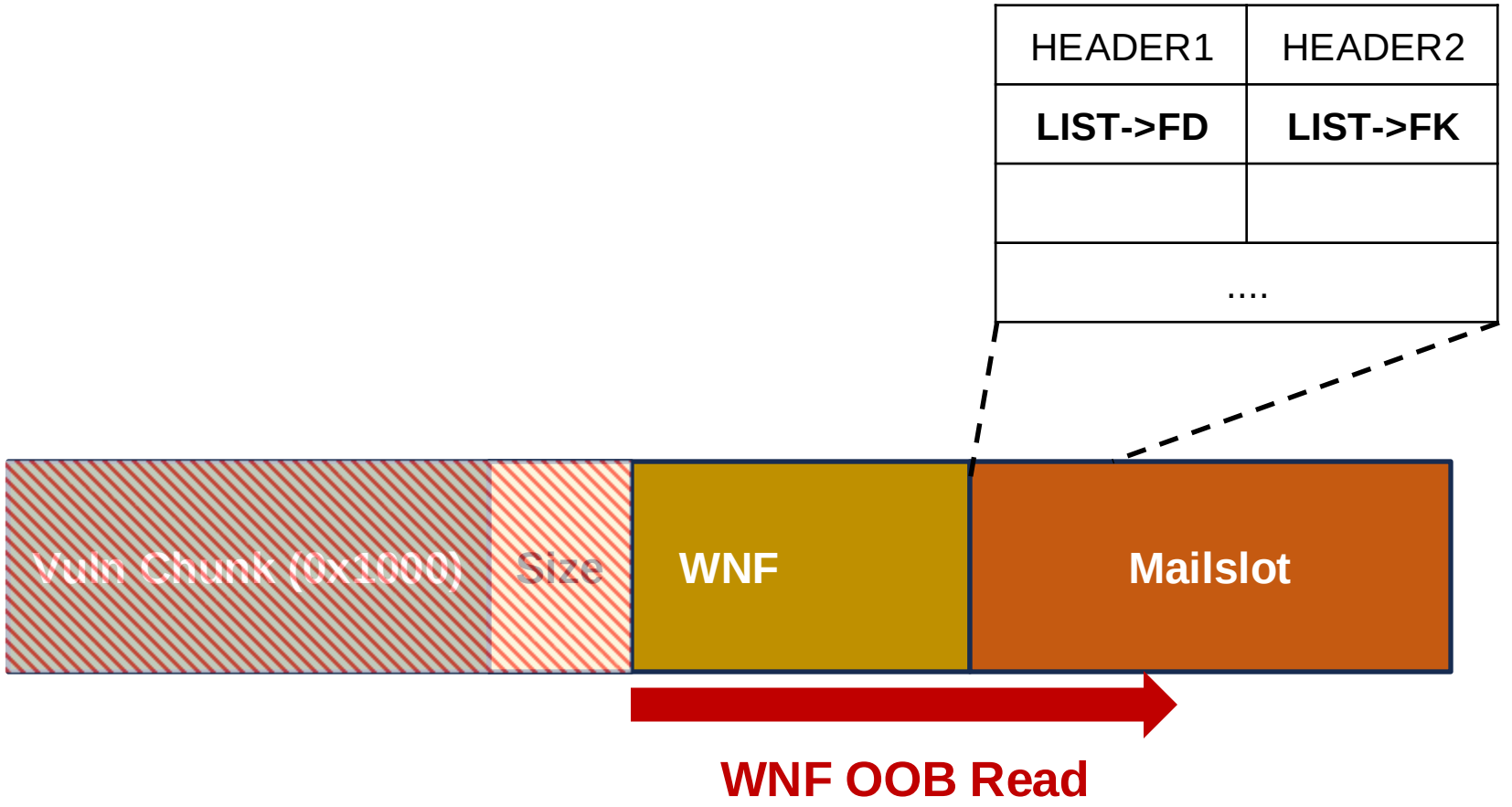


Exploitation - Layout

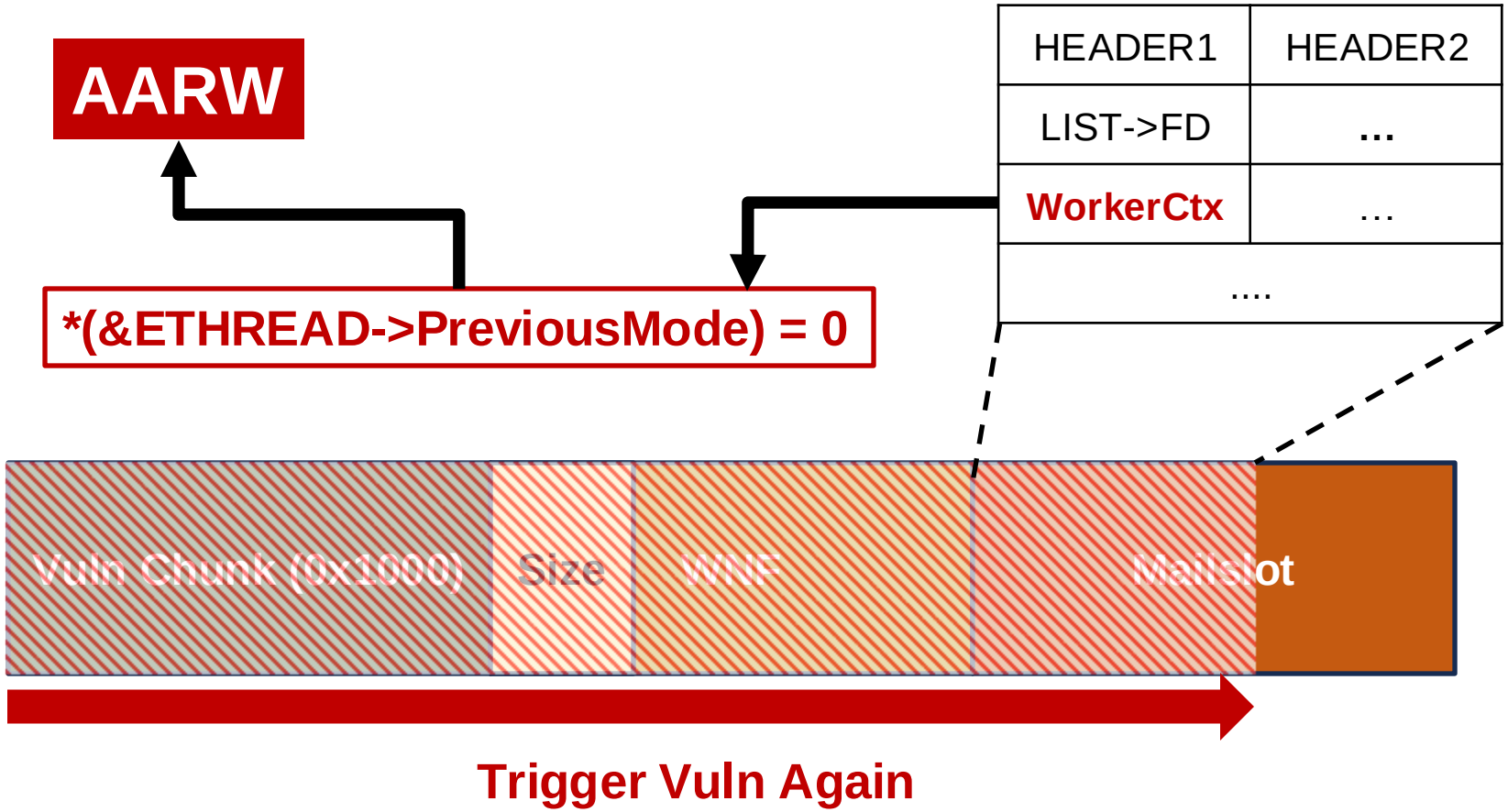
* Heap Spray needed



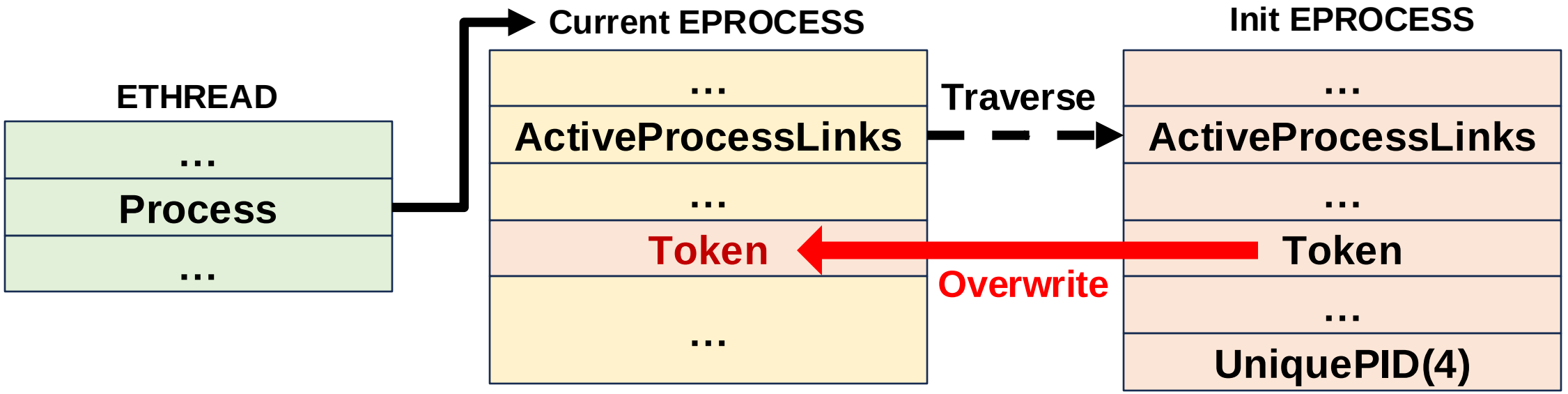
Exploitation – Mailslot Header Info Leak



Exploitation – Leveraging Arbitrary Nullification



Exploitation – Token Overwriting

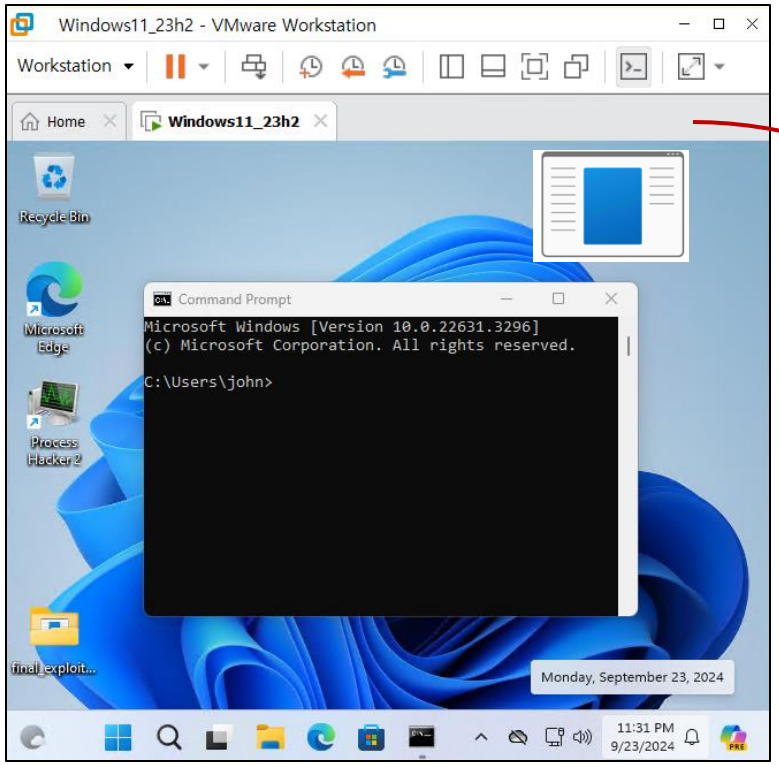


4. Chaining Exploits

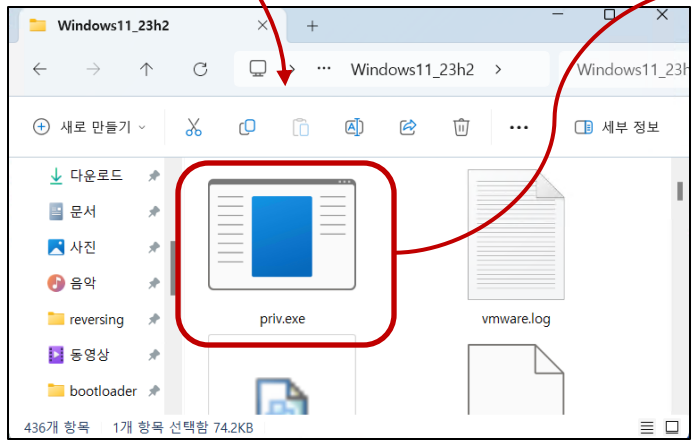
theori

Run the Windows LPE exploit

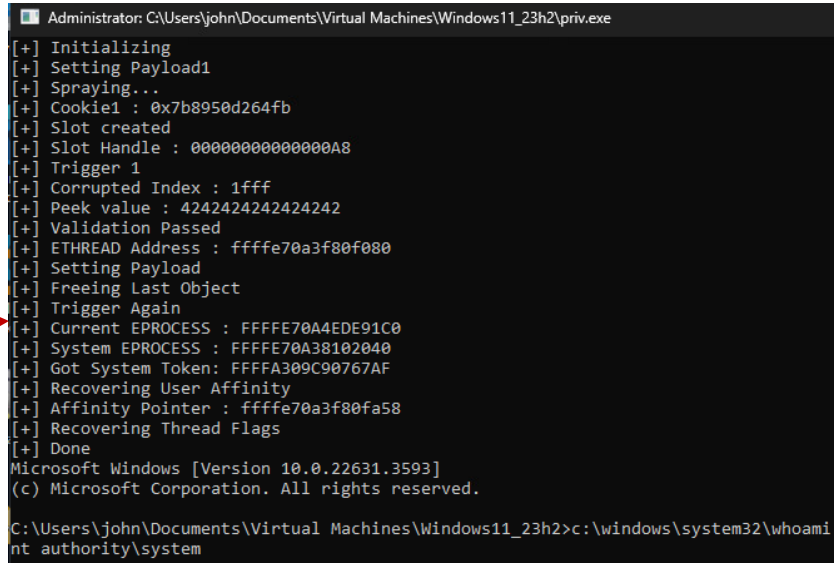
1. shellcode runs in vmware-vmx



2. Drop the Windows LPE on host



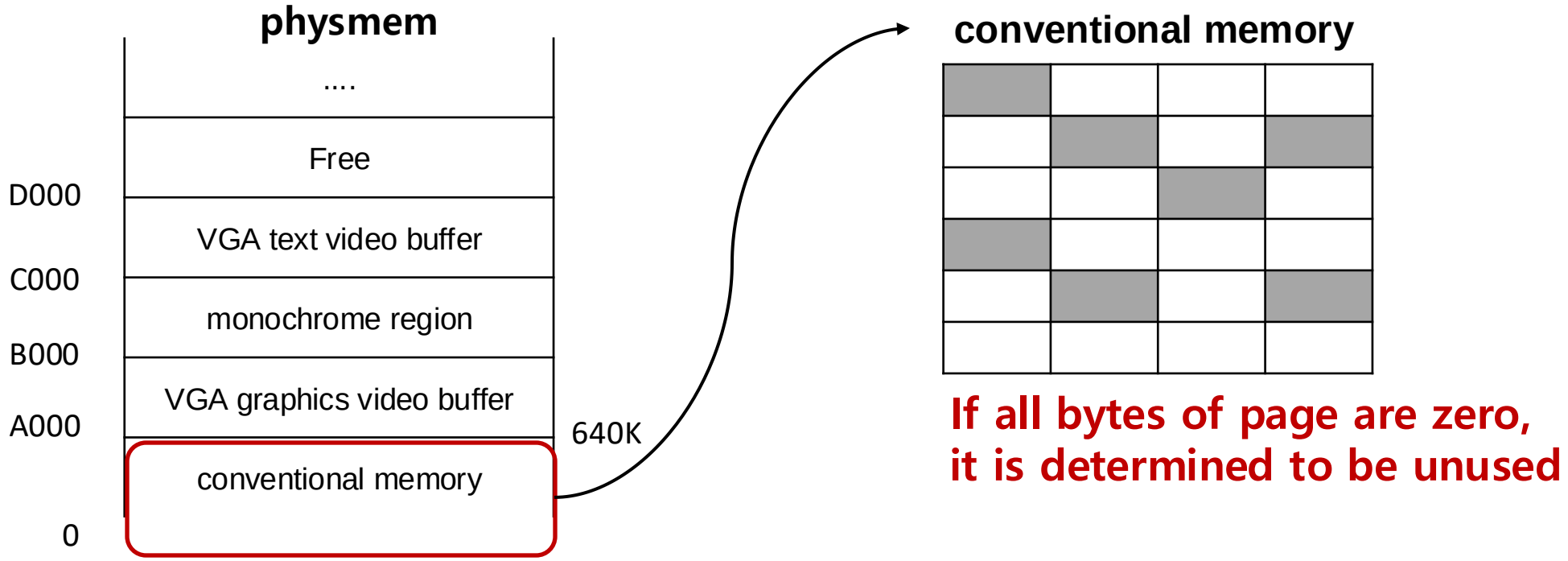
3. Run the Windows LPE on host



Drop the Windows LPE exploit

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

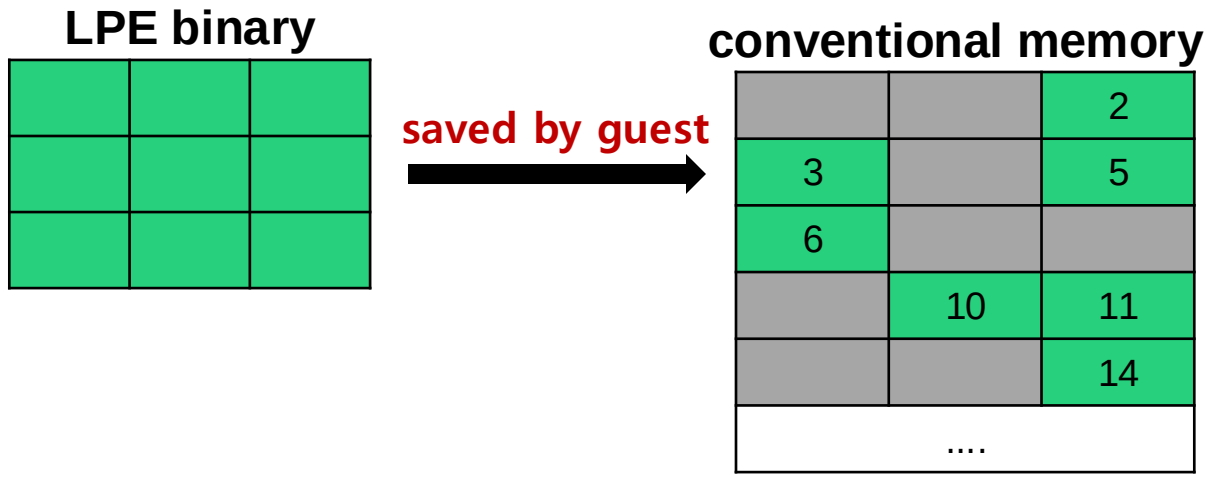
- Conventional memory has enough unused pages to store LPE Exploit



Run the Windows LPE exploit

Build a shellcode

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

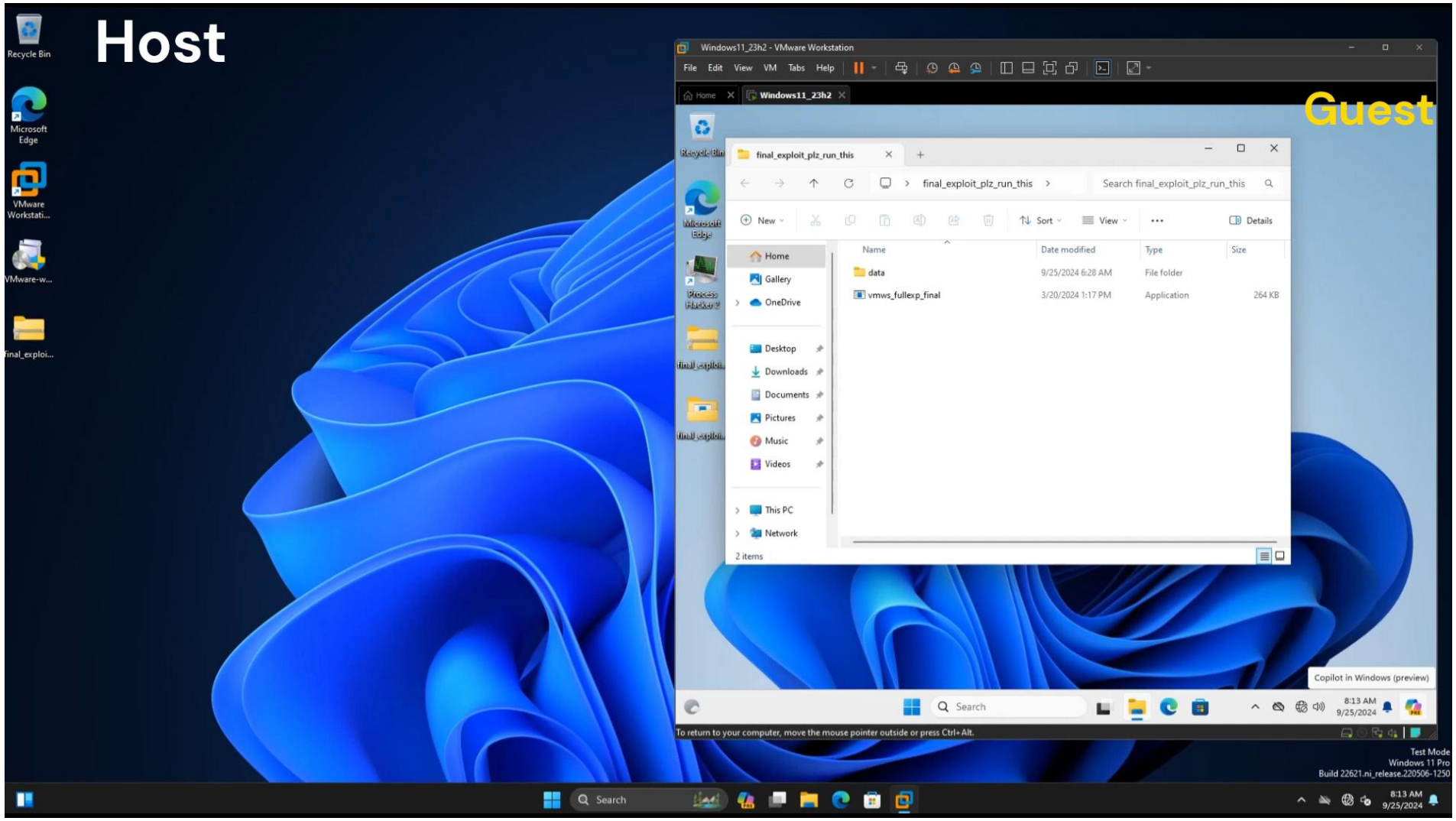



```
block_table = [2,3,5,6,10,11,14, ...];

for(i=0; i<nblock; i++) {
    readPhysmem(block_table[i] * PAGE_SIZE, buf);
    WriteFile(hFile, buf, PAGE_SIZE);
}
```

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

Demo



5. Conclusion

theori

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
  - @bbbig12
- @theori_io

End Of Document

theori