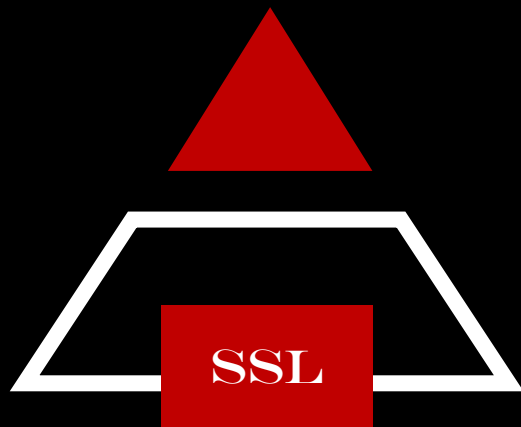


Bug Tamer: Turning Limited Heap Overflow into Full VMware Escape



Yongkang Jia & Xiao Lei

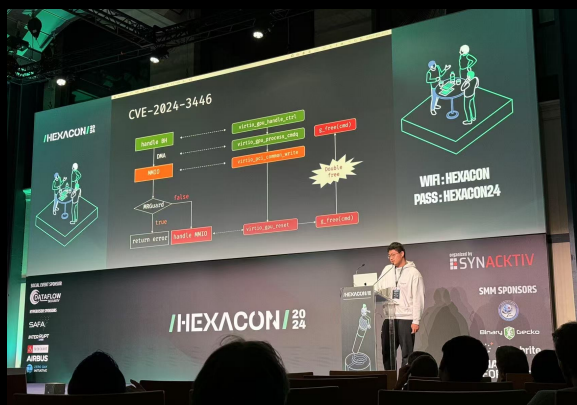
About us



HITB2023HKT



GeekCon2024 DAF



HexaCon2024



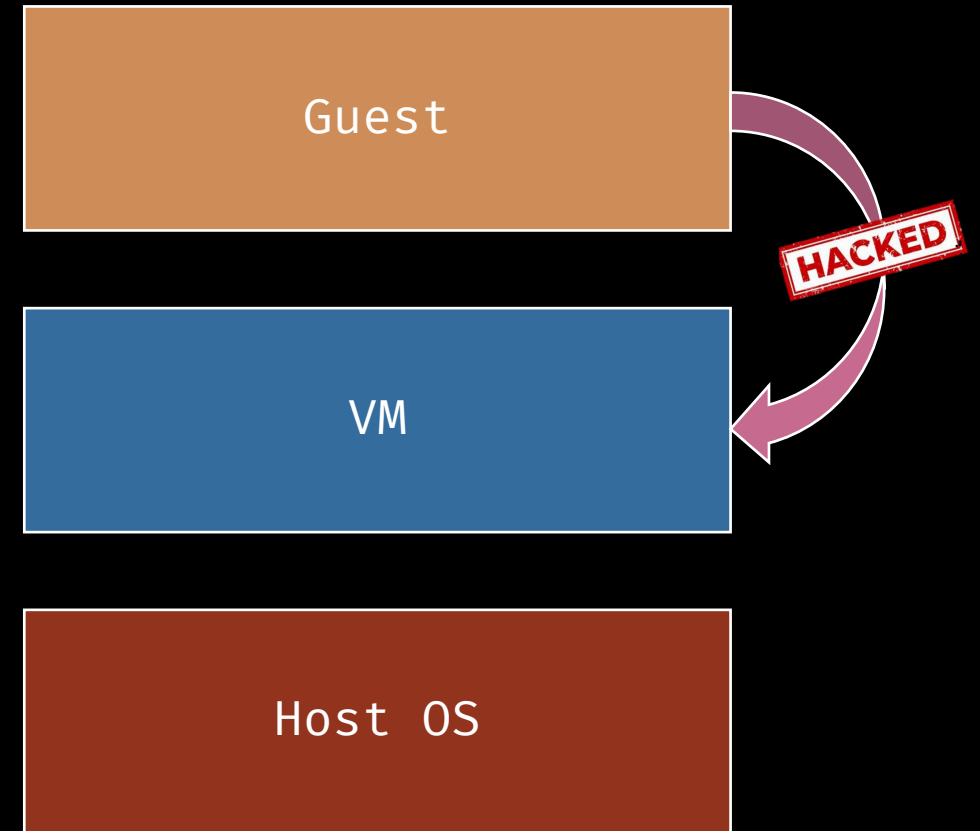
NoHat2025

Agenda

- Introduction
- A TOCTOU bug
- Exploitation development
- Conclusion

VM Escape

- QEMU Escape
 - CVE-2023-3180
 - CVE-2024-3446 & CVE-2024-8612
- VirtualBox Escape
 - CVE-2023-21987 & CVE-2023-21991
 - CVE-2024-21121
 - CVE-2025-30712
- VMware Escape
 - CVE-2023-20869 & CVE-2023-20870
 - CVE-2024-22267 & CVE-2024-22270



Why VMware

- A hot topic in virtualization research
 - Pwn2Own
- Some technical article could be referenced
 - Security Confs, technical blogs, etc.
- More challenging
 - Close source
- Would be much interesting! 😊

Reverse engineering

- Got function symbols from ESXi ARM fling

MINDSHARE: ANALYSIS OF VMWARE WORKSTATION AND ESXI USING DEBUG SYMBOLS FROM FLINGS

January 07, 2021 | Reno Robert

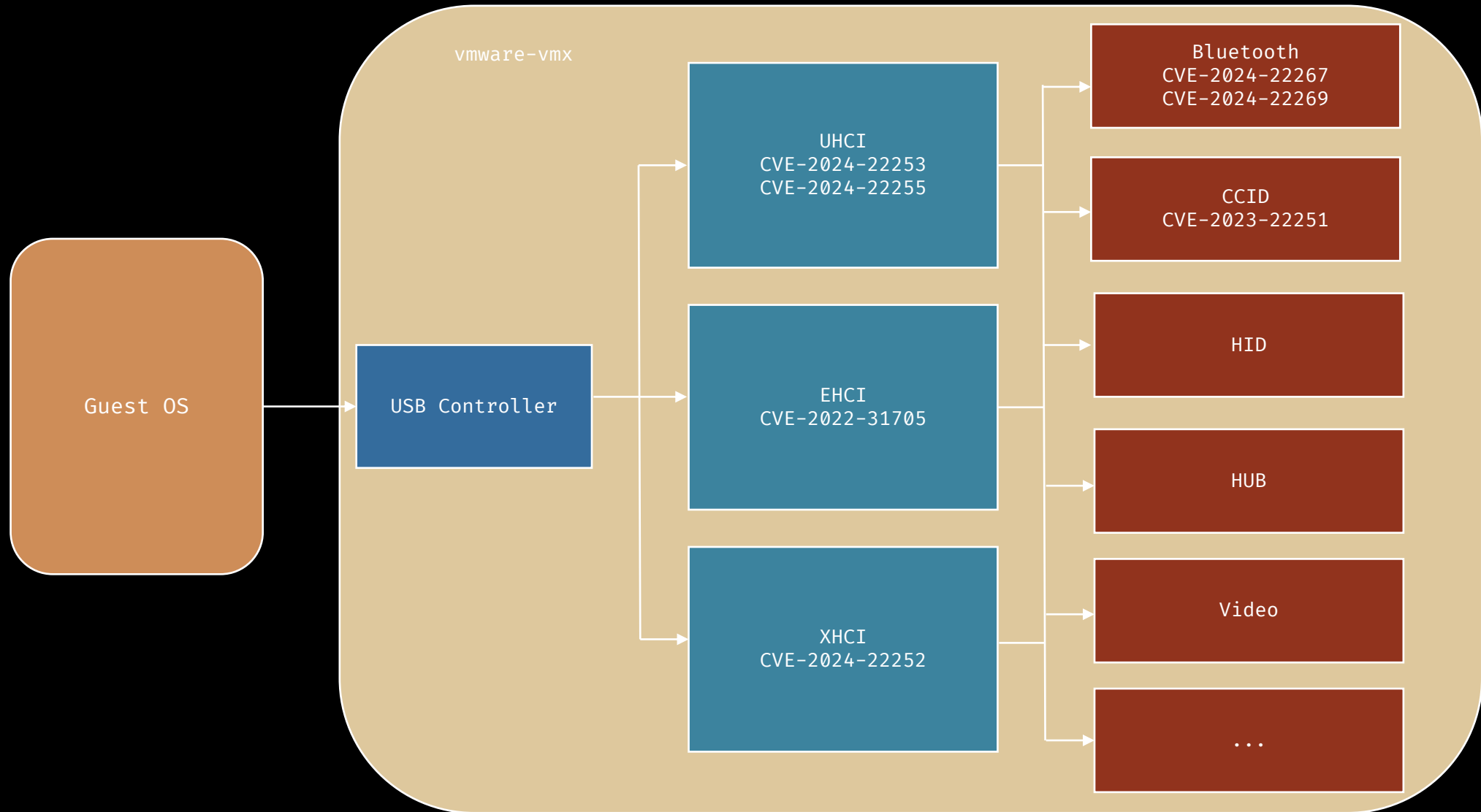
- Reproduce disclosed bugs



Guest Revolution: Our Story of Compromising the Host Kernel from the VMware Guest

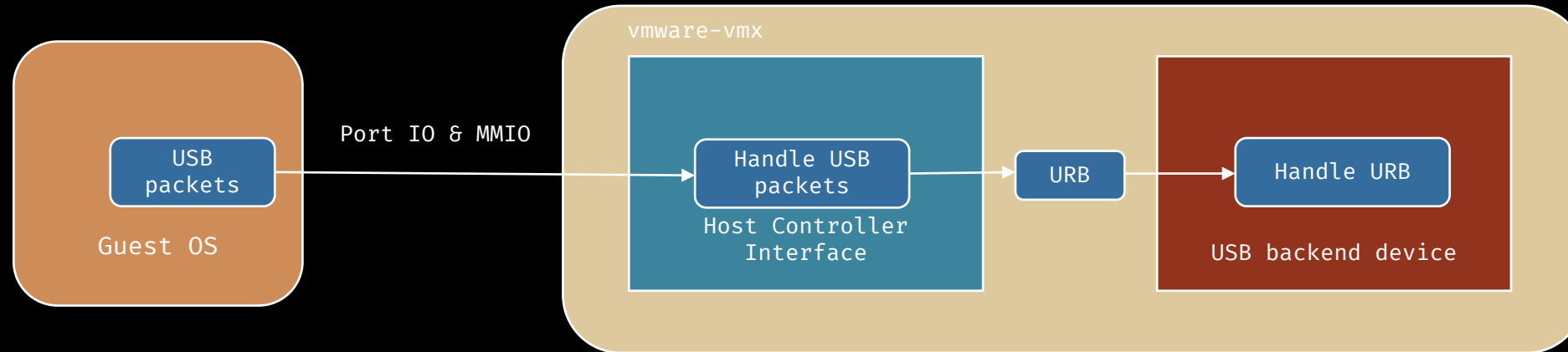
Junoh Lee, Gwangun Jung

USB Bug Map

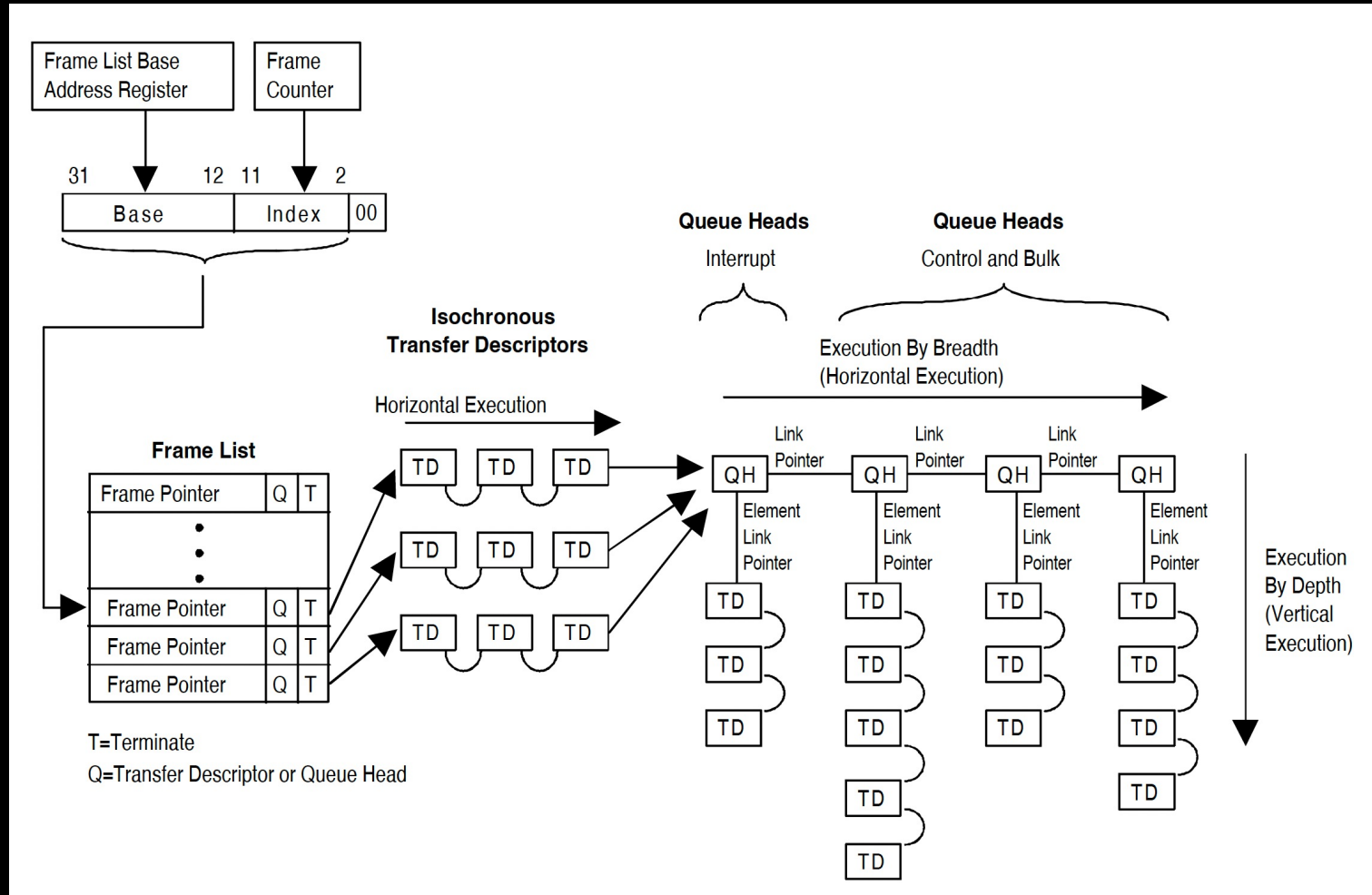


How USB bug occurs

- Handle USB packets
 - Handle uhci, handle ehci, handle xhci
- Handle URB(USB Request Block)
 - Handle Bluetooth protocol, handle video params, etc.



Take a look at UHCI & EHCI



Our Plan

Reproduce a virtual Bluetooth bug

- Reverse uhci & ehci handle process
- Reverse Bluetooth protocol handle process
- Find some new bugs

But, at the beginning

- Some interesting things happened

Agenda

- Introduction
- A TOC-TOU bug
- Exploitation development
- Conclusion

A TOC-TOU bug

pseudocode snippets of function Ehci_HandleQH

```
...
PhysMem_Get(bufferPointerGPA, bytesToTransfer, 1u, 8u, &physMemDesc);
_bytesToTransfer = _totalBytesToTransfer;
if ( physMemDesc.singelPage == 1 )
{
    setupPacket = *(VUsbSetup *)physMemDesc.hostVA;
}
else
{
    PhysMemReadSlow(&physMemDesc, 0LL, 8uLL, (char *)&setupPacket);
    _bytesToTransfer = _totalBytesToTransfer;
}
bmRequestType = setupPacket.bmRequestType;
wLength = setupPacket.wLength;
...
length = wLength + _bytesToTransfer;
setupUrb = VUsb_NewUrb(pipe_, 0, wLength + _bytesToTransfer);
urbData = setupUrb->urbDataPtr;
setupUrb->numPackets = 1;
setupUrb->dataLen = length;
v24 = physMemDesc.singelPage == 1;
setupUrb->tokenType = isHostToDevice;
if ( v24 )
    memcpy(urbData, (const void *)physMemDesc.hostVA, bytesToTransfer);
else
    PhysMemReadSlow(&physMemDesc, 0LL, bytesToTransfer, (char *)urbData);
...
```

A TOC-TOU bug

- Read SETUP packet data from guest's buffer pointer for the first time

pseudocode snippets of function Ehci_HandleQH

```
...
PhysMem_Get(bufferPointerGPA, bytesToTransfer, 1u, 8u, &physMemDesc);
_bytesToTransfer = _totalBytesToTransfer;
if ( physMemDesc.singelPage == 1 )
{
    setupPacket = *(VUsbSetup *)physMemDesc.hostVA;
}
else
{
    PhysMemReadSlow(&physMemDesc, 0LL, 8uLL, (char *)&setupPacket);
    _bytesToTransfer = _totalBytesToTransfer;
}
bmRequestType = setupPacket.bmRequestType;
wLength = setupPacket.wLength;
...
length = wLength + _bytesToTransfer;
setupUrb = VUsb_NewUrb(pipe_, 0, wLength + _bytesToTransfer);
urbData = setupUrb->urbDataPtr;
setupUrb->numPackets = 1;
setupUrb->dataLen = length;
v24 = physMemDesc.singelPage == 1;
setupUrb->tokenType = isHostToDevice;
if ( v24 )
    memcpy(urbData, (const void *)physMemDesc.hostVA, bytesToTransfer);
else
    PhysMemReadSlow(&physMemDesc, 0LL, bytesToTransfer, (char *)urbData);
...
```

A TOC-TOU bug

- Read SETUP packet data from guest's buffer pointer for the first time
- Allocate a urb object, whose size is $0x98 + wLength + 8$

pseudocode snippets of function Ehci_HandleQH

```
...
PhysMem_Get(bufferPointerGPA, bytesToTransfer, 1u, 8u, &physMemDesc);
_bytesToTransfer = _totalBytesToTransfer;
if ( physMemDesc.singelPage == 1 )
{
    setupPacket = *(VUsbSetup *)physMemDesc.hostVA;
}
else
{
    PhysMemReadSlow(&physMemDesc, 0LL, 8uLL, (char *)&setupPacket);
    _bytesToTransfer = _totalBytesToTransfer;
}
bmRequestType = setupPacket.bmRequestType;
wLength = setupPacket.wLength;
...
length = wLength + _bytesToTransfer;
setupUrb = VUsb_NewUrb(pipe_, 0, wLength + _bytesToTransfer);
urbData = setupUrb->urbDataPtr;
setupUrb->numPackets = 1;
setupUrb->dataLen = length;
v24 = physMemDesc.singelPage == 1;
setupUrb->tokenType = isHostToDevice;
if ( v24 )
    memcpy(urbData, (const void *)physMemDesc.hostVA, bytesToTransfer);
else
    PhysMemReadSlow(&physMemDesc, 0LL, bytesToTransfer, (char *)urbData);
...
```

A TOC-TOU bug

- Read SETUP packet data from guest's buffer pointer for the first time
- Allocate a urb object, whose size is $0x98 + wLength + 8$
- Read SETUP packet data **for the second time** and store them in the allocated urb's *dataBuf*

pseudocode snippets of function Ehci_HandleQH

```
...
PhysMem_Get(bufferPointerGPA, bytesToTransfer, 1u, 8u, &physMemDesc);
_bytesToTransfer = _totalBytesToTransfer;
if ( physMemDesc.singelPage == 1 )
{
    setupPacket = *(VUsbSetup *)physMemDesc.hostVA;
}
else
{
    PhysMemReadSlow(&physMemDesc, 0LL, 8uLL, (char *)&setupPacket);
    _bytesToTransfer = _totalBytesToTransfer;
}
bmRequestType = setupPacket.bmRequestType;
wLength = setupPacket.wLength;
...
length = wLength + _bytesToTransfer;
setupUrb = VUsb_NewUrb(pipe_, 0, wLength + _bytesToTransfer);
urbData = setupUrb->urbDataPtr;
setupUrb->numPackets = 1;
setupUrb->dataLen = length;
v24 = physMemDesc.singelPage == 1;
setupUrb->tokenType = isHostToDevice;
if ( v24 )
    memcpy(urbData, (const void *)physMemDesc.hostVA, bytesToTransfer);
else
    PhysMemReadSlow(&physMemDesc, 0LL, bytesToTransfer, (char *)urbData);
...
```

A TOC-TOU bug

- Read SETUP packet data from guest's buffer pointer for the first time
- Allocate a urb object, whose size is $0x98 + wLength + 8$
- Read SETUP packet data **for the second time** and store them in the allocated urb's *dataBuf*
- A TOC-TOU bug occurs

pseudocode snippets of function Ehci_HandleQH

```
...
PhysMem_Get(bufferPointerGPA, bytesToTransfer, 1u, 8u, &physMemDesc);
_bytesToTransfer = _totalBytesToTransfer;
if ( physMemDesc.singelPage == 1 )
{
    setupPacket = *(VUsbSetup *)physMemDesc.hostVA;
}
else
{
    PhysMemReadSlow(&physMemDesc, 0LL, 8uLL, (char *)&setupPacket);
    _bytesToTransfer = _totalBytesToTransfer;
}
bmRequestType = setupPacket.bmRequestType;
wLength = setupPacket.wLength;
...
length = wLength + _bytesToTransfer;
setupUrb = VUsb_NewUrb(pipe_, 0, wLength + _bytesToTransfer);
urbData = setupUrb->urbDataPtr;
setupUrb->numPackets = 1;
setupUrb->dataLen = length;
v24 = physMemDesc.singelPage == 1;
setupUrb->tokenType = isHostToDevice;
if ( v24 )
    memcpy(urbData, (const void *)physMemDesc.hostVA, bytesToTransfer);
else
    PhysMemReadSlow(&physMemDesc, 0LL, bytesToTransfer, (char *)urbData);
...
```



A TOC-TOU bug

- Read SETUP packet data from guest's buffer pointer for the first time
- Allocate a urb object, whose size is $0x98 + wLength + 8$
- Read SETUP packet data **for the second time** and store them in the allocated urb's *dataBuf*
- A TOC-TOU bug occurs
- Both in UHCI & EHCI devices

pseudocode snippets of function Ehci_HandleQH

```
...
PhysMem_Get(bufferPointerGPA, bytesToTransfer, 1u, 8u, &physMemDesc);
_bytesToTransfer = _totalBytesToTransfer;
if ( physMemDesc.singelPage == 1 )
{
    setupPacket = *(VUsbSetup *)physMemDesc.hostVA;
}
else
{
    PhysMemReadSlow(&physMemDesc, 0LL, 8uLL, (char *)&setupPacket);
    _bytesToTransfer = _totalBytesToTransfer;
}
}

bmRequestType = setupPacket.bmRequestType;
wLength = setupPacket.wLength;
...
length = wLength + _bytesToTransfer;
setupUrb = VUsb_NewUrb(pipe_, 0, wLength + _bytesToTransfer);
urbData = setupUrb->urbDataPtr;
setupUrb->numPackets = 1;
setupUrb->dataLen = length;
v24 = physMemDesc.singelPage == 1;
setupUrb->tokenType = isHostToDevice;
if ( v24 )
    memcpy(urbData, (const void *)physMemDesc.hostVA, bytesToTransfer);
else
    PhysMemReadSlow(&physMemDesc, 0LL, bytesToTransfer, (char *)urbData);
...

```



Exploit Capability Exploration

- Available USB backend devices
 - Virtual Hub, Virtual Storage, Virtual Bluetooth, Virtual Video, ...

Virtual Video

- Limited oob read / write primitive

```
setupPacket = urb->setupPacket;
if ((setupPacket.bmRequestType & 0x7F) == 0x21) {
    if ((setupPacket.wIndex == 1) && (setupPacket.wValue == 0x100 || setupPacket.wValue == 0x200)) {
        if (setupPacket.bRequest == 1) {
            // After some checks
            wLength = setupPacket.wLength >= 0x22 ? 0x22 : setupPacket.wLength;
            memcpy(&vusbDev->uvcParams, urb->extraData, wLength);
        } else if (setupPacket.bRequest == 0x81) {
            wLength = setupPacket.wLength >= 0x22 ? 0x22 : setupPacket.wLength;
            memcpy(urb->extraData, &vusbDev->uvcParams, wLength);
        } else {
            ...
        }
    }
}
```

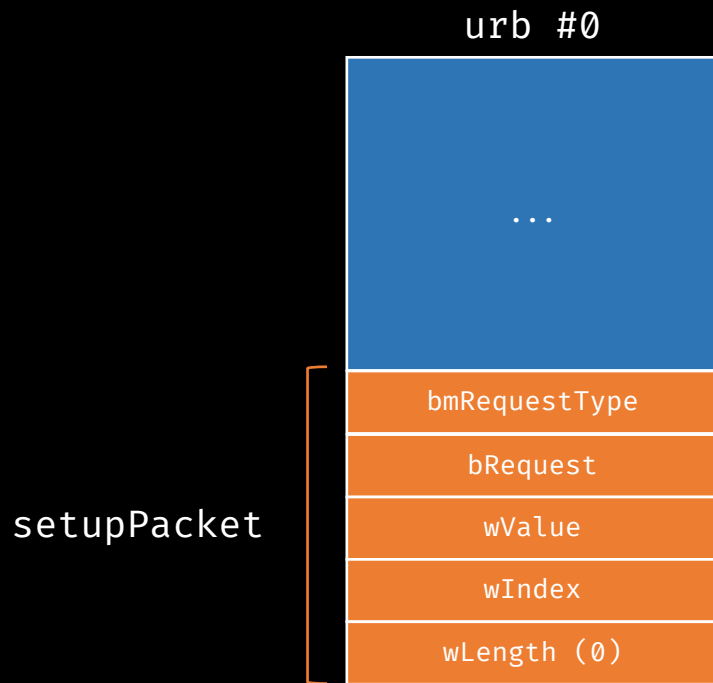
Virtual Video

- Limited oob read / write primitive

```
setupPacket = urb->setupPacket;
if ((setupPacket.bmRequestType & 0x7F) == 0x21) {
    if ((setupPacket.wIndex == 1) && (setupPacket.wValue == 0x100 || setupPacket.wValue == 0x200)) {
        if (setupPacket.bRequest == 1) {
            // After some checks
            wLength = setupPacket.wLength >= 0x22 ? 0x22 : setupPacket.wLength;
            memcpy(&vusbDev->uvcParams, urb->extraData, wLength); set the payload data
        } else if (setupPacket.bRequest == 0x81) {
            wLength = setupPacket.wLength >= 0x22 ? 0x22 : setupPacket.wLength;
            memcpy(urb->extraData, &vusbDev->uvcParams, wLength); perform oob read / write
        } else {
            ...
        }
    }
}
```

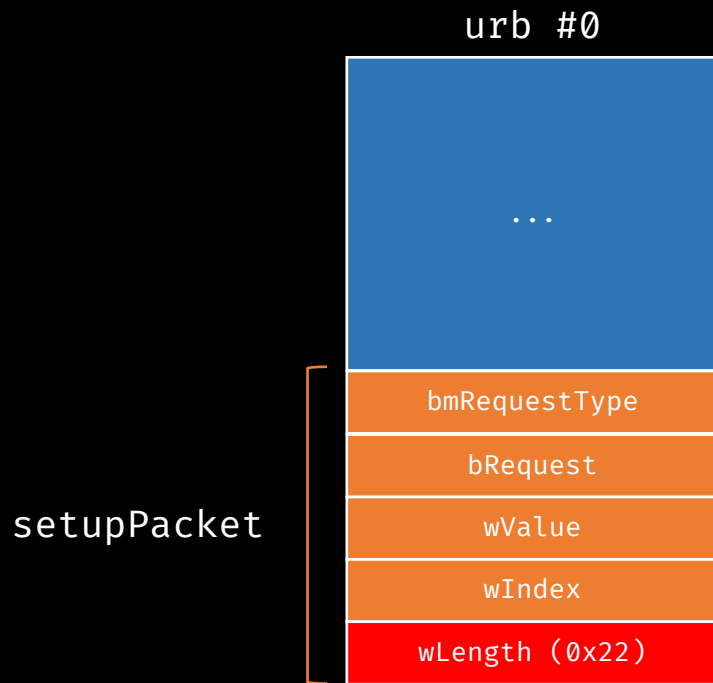
Virtual Video

- Limited oob read / write primitive



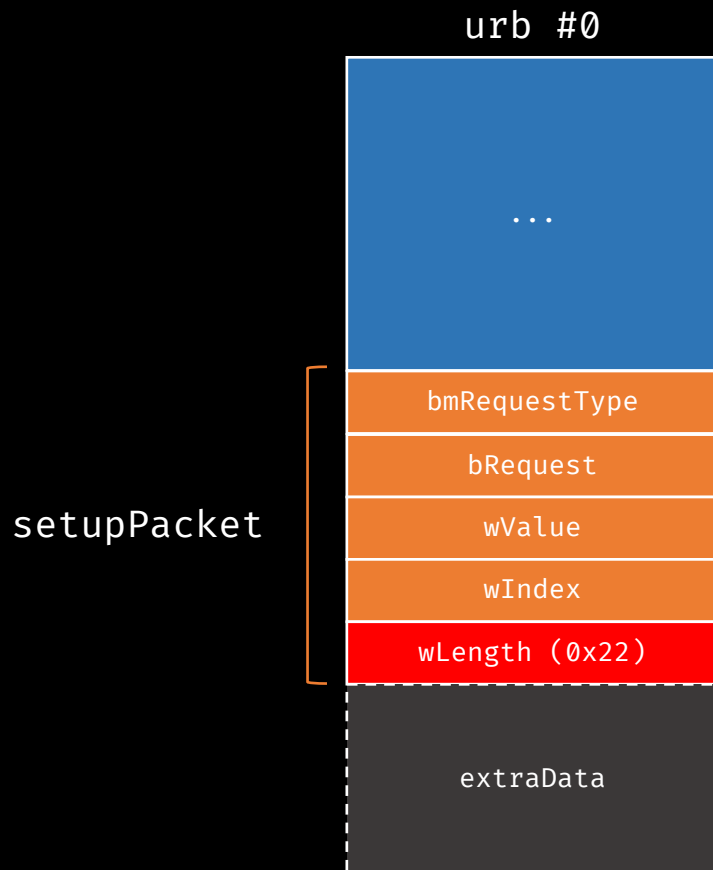
Virtual Video

- Limited oob read / write primitive



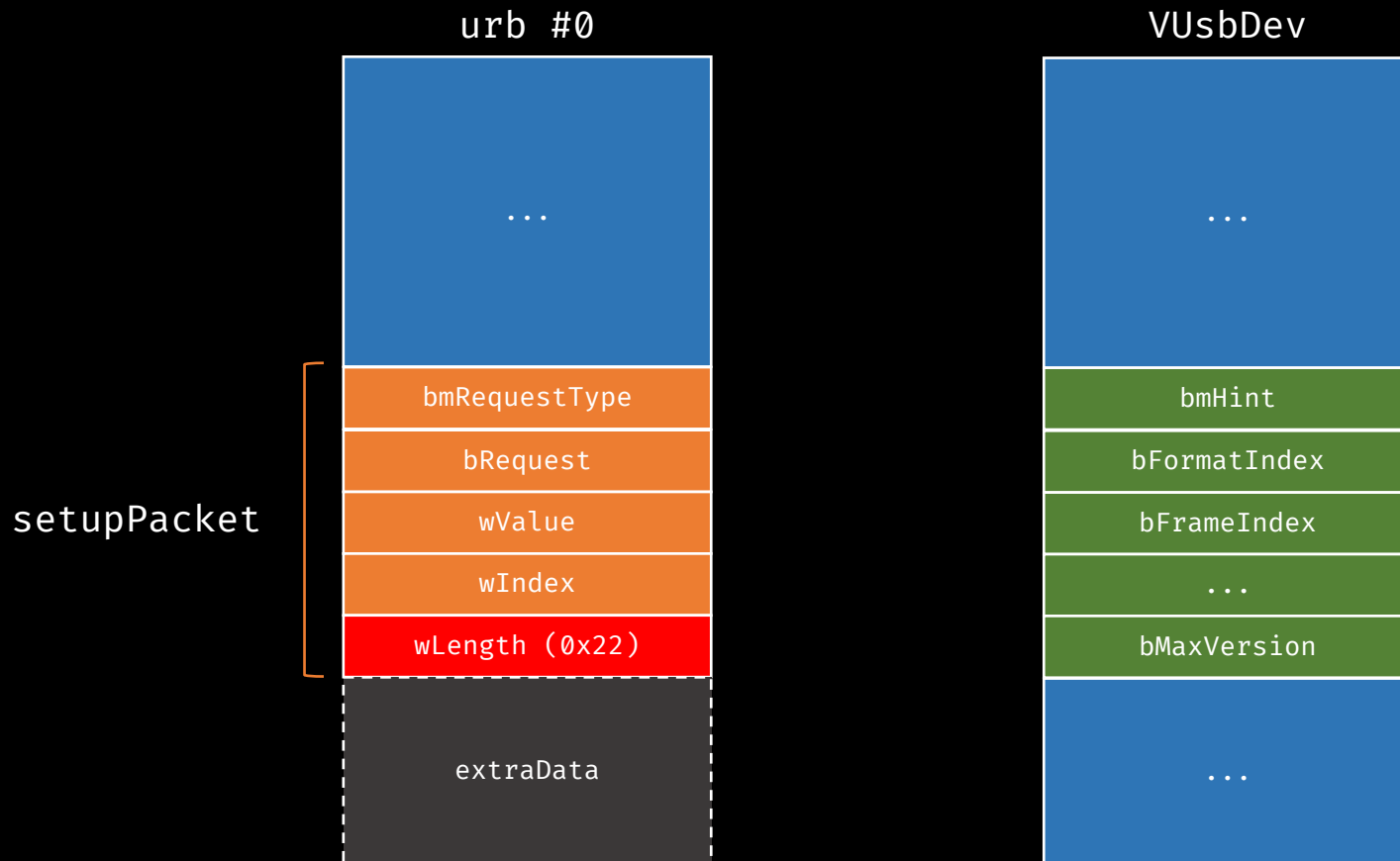
Virtual Video

- Limited oob read / write primitive



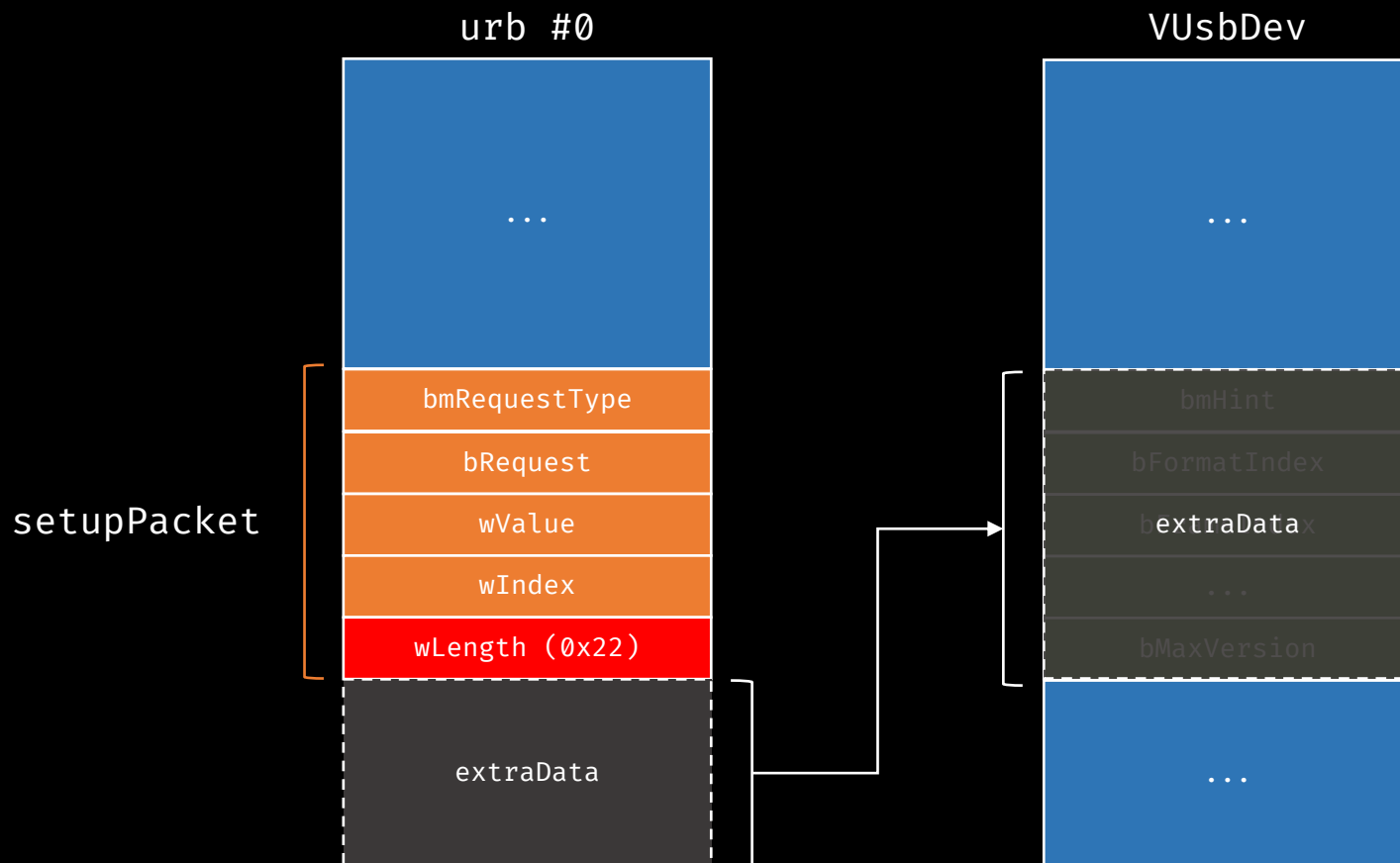
Virtual Video

- Limited oob read / write primitive



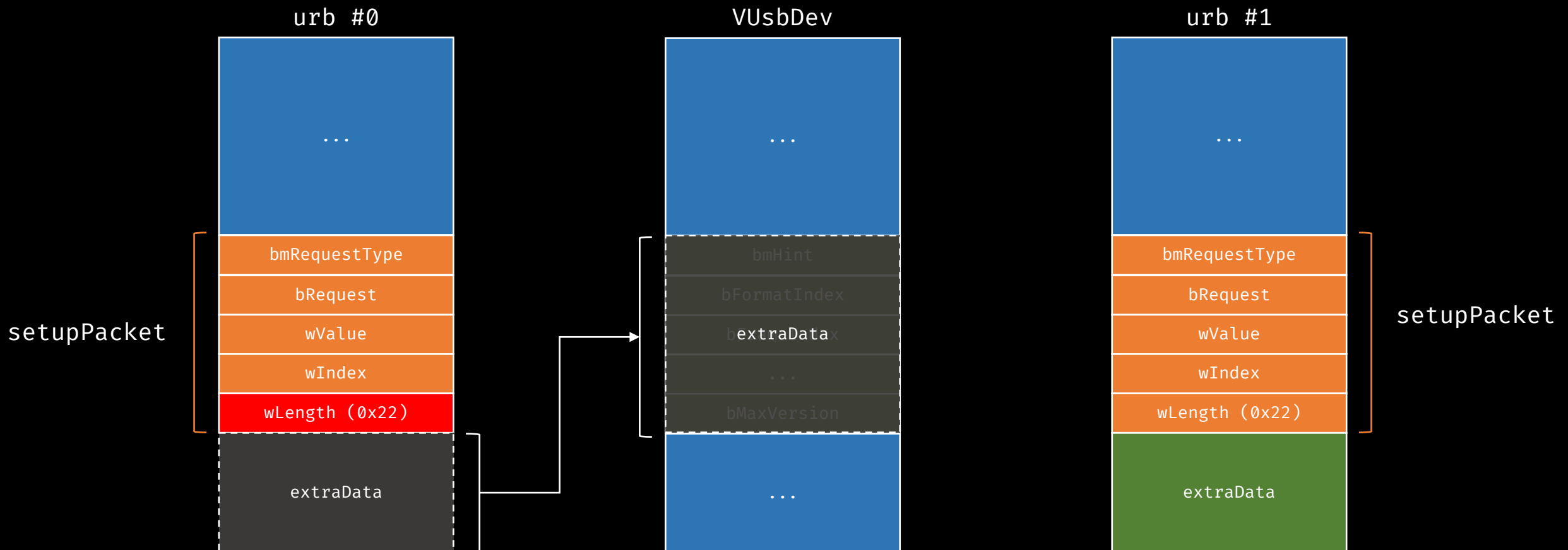
Virtual Video

- Limited oob read / write primitive



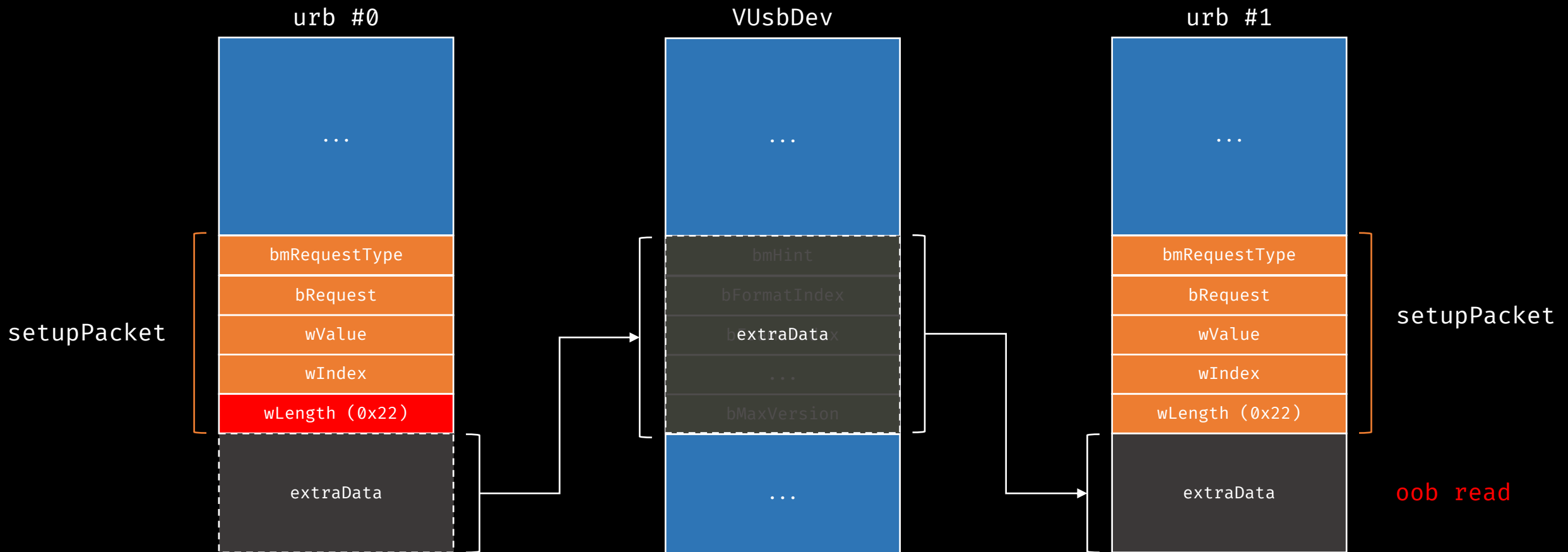
Virtual Video

- Limited oob read / write primitive



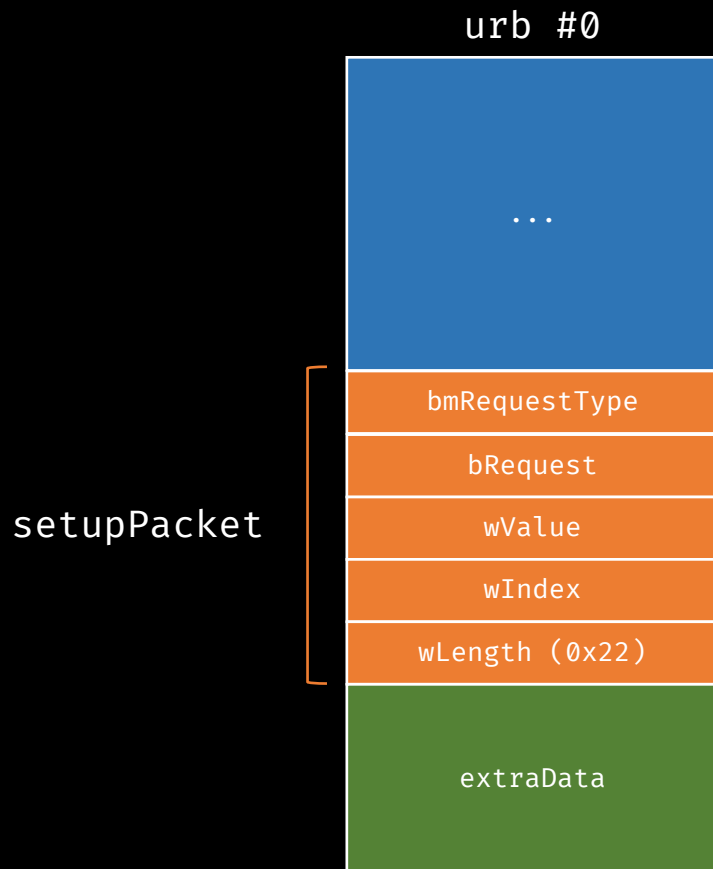
Virtual Video

- Limited oob read / write primitive



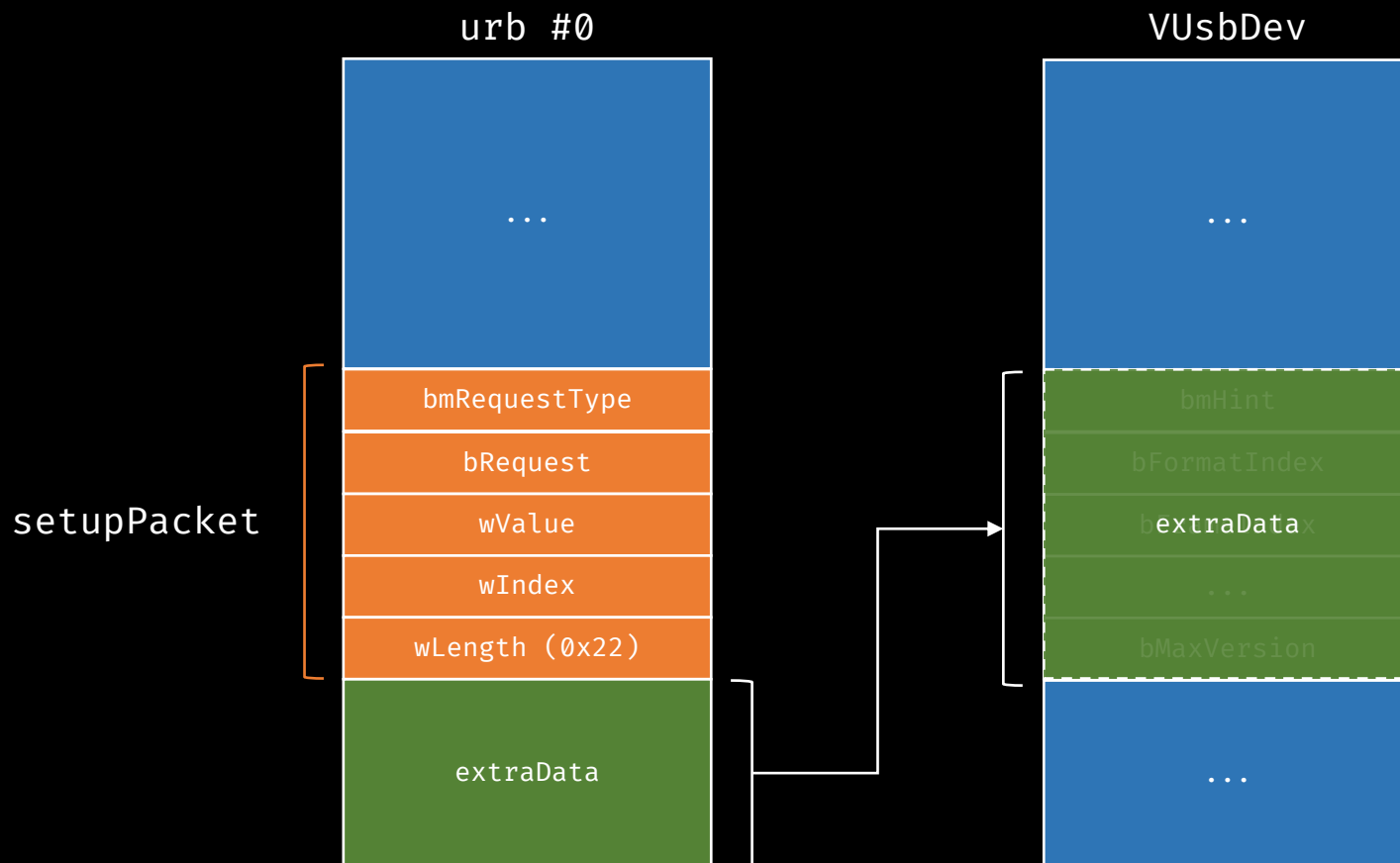
Virtual Video

- Limited oob read / write primitive



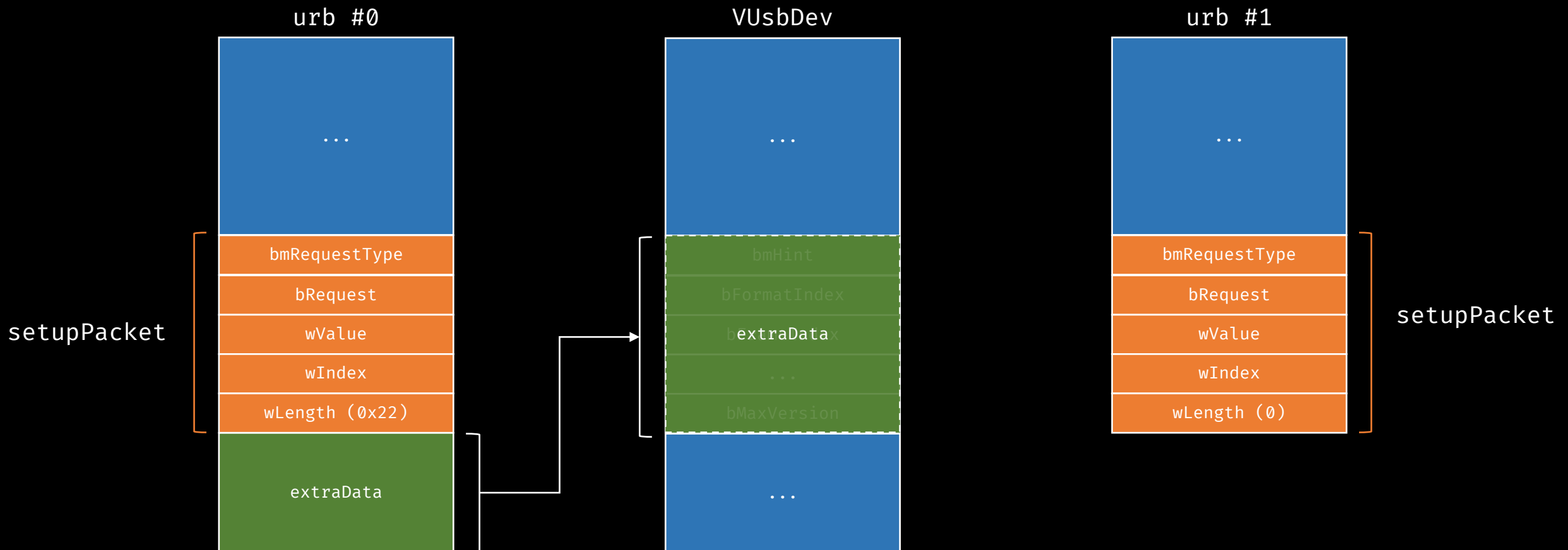
Virtual Video

- Limited oob read / write primitive



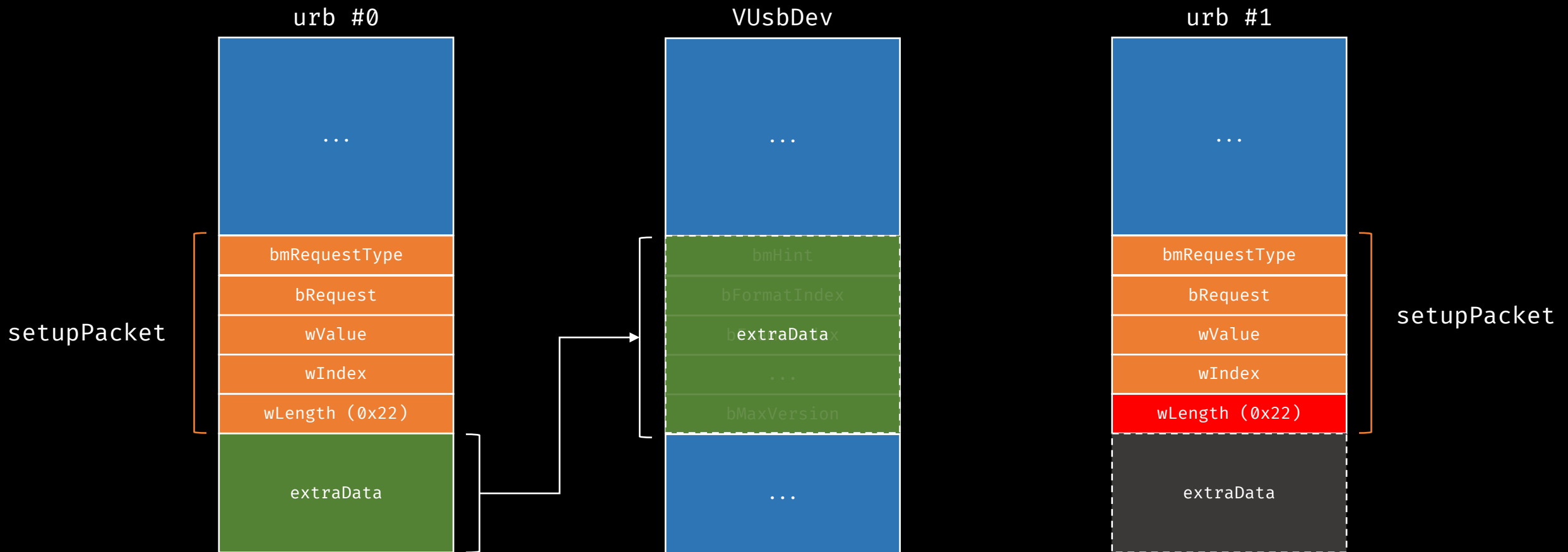
Virtual Video

- Limited oob read / write primitive



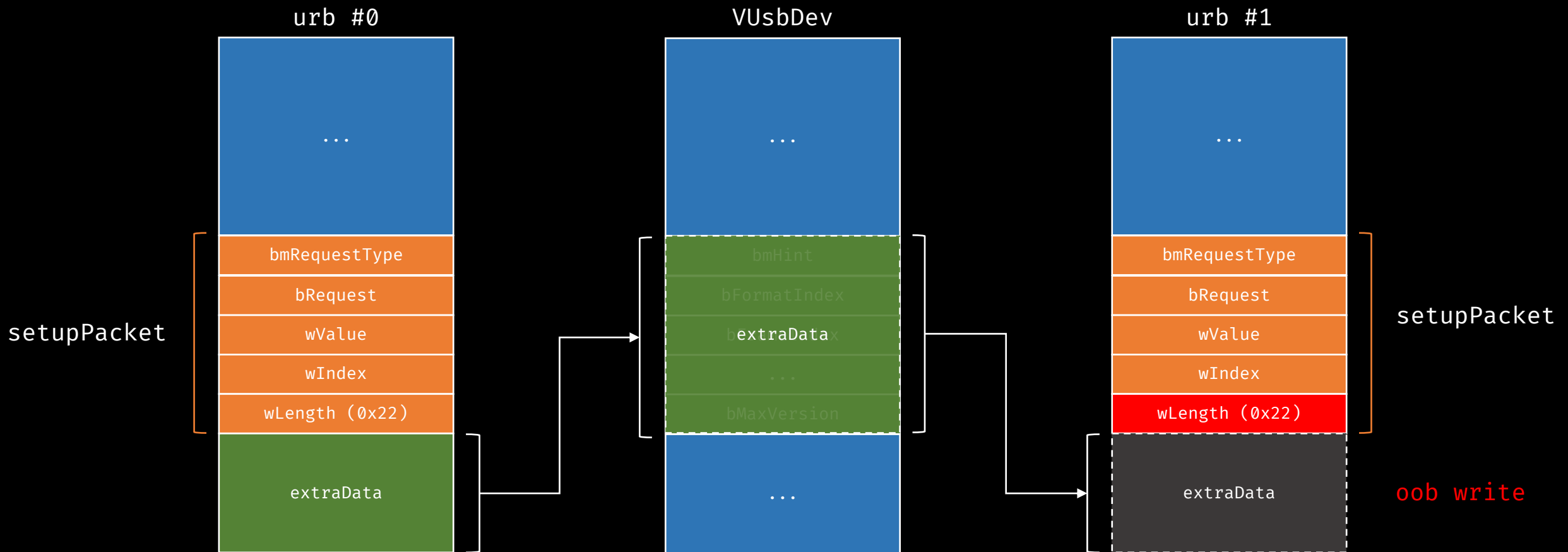
Virtual Video

- Limited oob read / write primitive



Virtual Video

- Limited oob read / write primitive



Exploit Capability Exploration

- Available USB backend devices
 - Virtual Hub, Virtual Storage, Virtual Bluetooth, **Virtual Video**, ...
- Vulnerable targets
 - VMWare Workstation, VMWare ESXi, VMWare fusion

Exploit Capability Exploration

- Available USB backend devices
 - Virtual Hub, Virtual Storage, Virtual Bluetooth, **Virtual Video**, ...
- Vulnerable targets
 - VMWare Workstation, VMWare ESXi, **VMWare fusion**

Exploit Capability Exploration

- Available USB backend devices
 - Virtual Hub, Virtual Storage, Virtual Bluetooth, **Virtual Video**, ...
- Vulnerable targets
 - VMWare Workstation, VMWare ESXi, **VMWare fusion**
- Host environment
 - MacOS (Intel & M-Chips)
 - VMWare fusion (13.5.2)
 - USB virtual video device (one or more virtual cameras)

Agenda

- Introduction
- A TOCTOU bug
- Exploitation development
- Conclusion

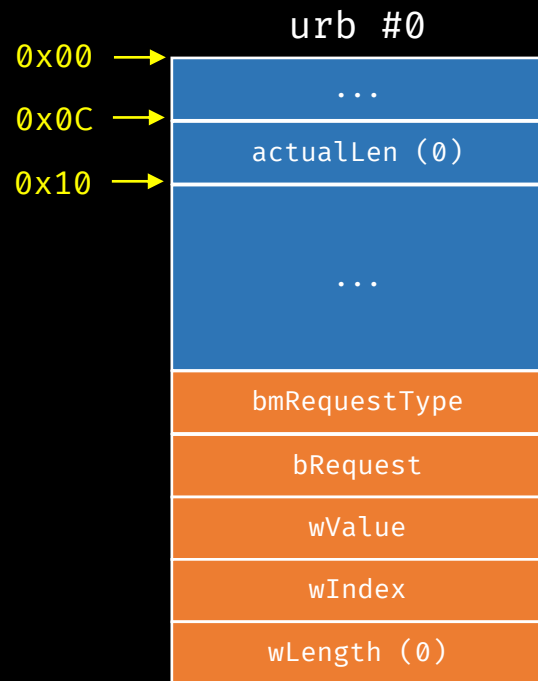
Exploit on Mac M1

Info Leak

- Limited oob write + urb object -> more powerful oob read

Info Leak

- Limited oob write + urb object -> more powerful oob read



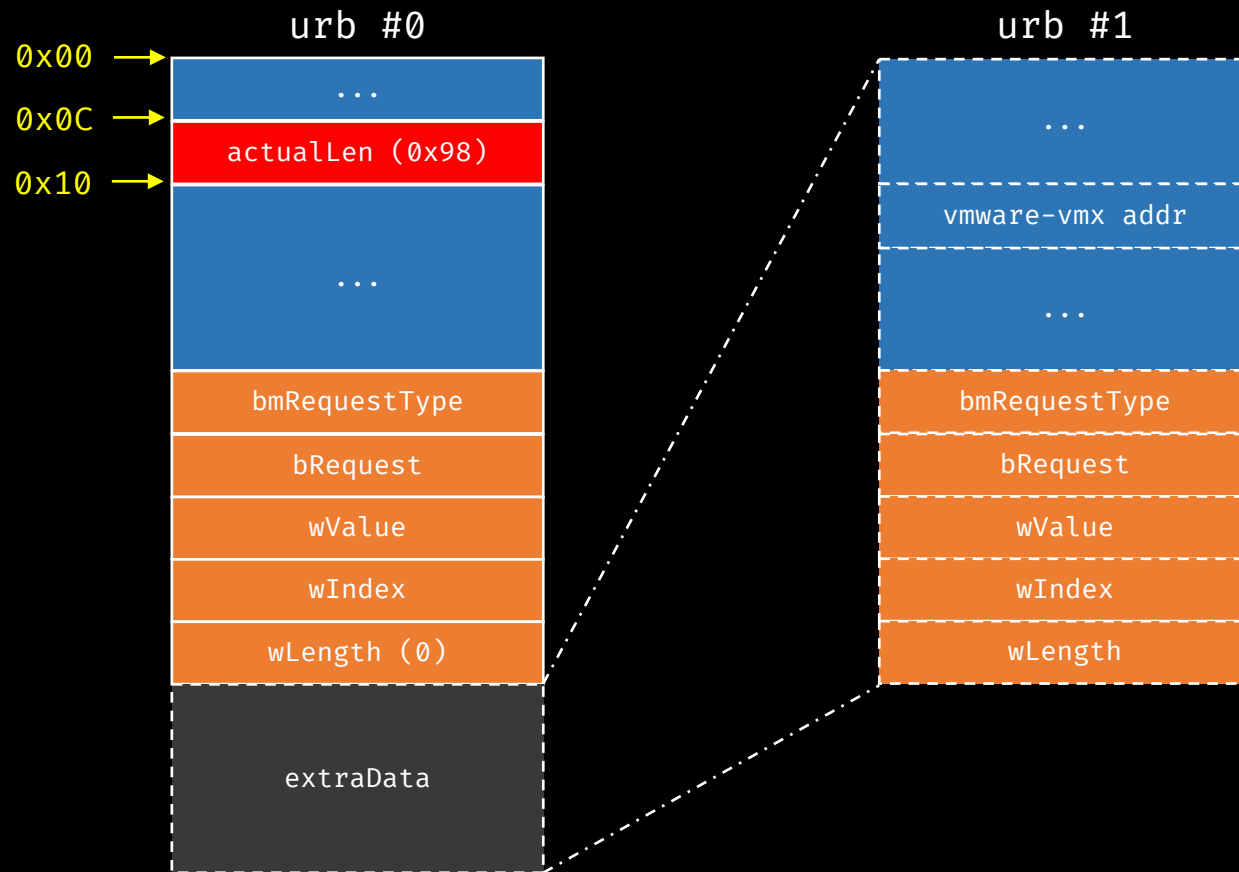
Info Leak

- Limited oob write + urb object -> more powerful oob read



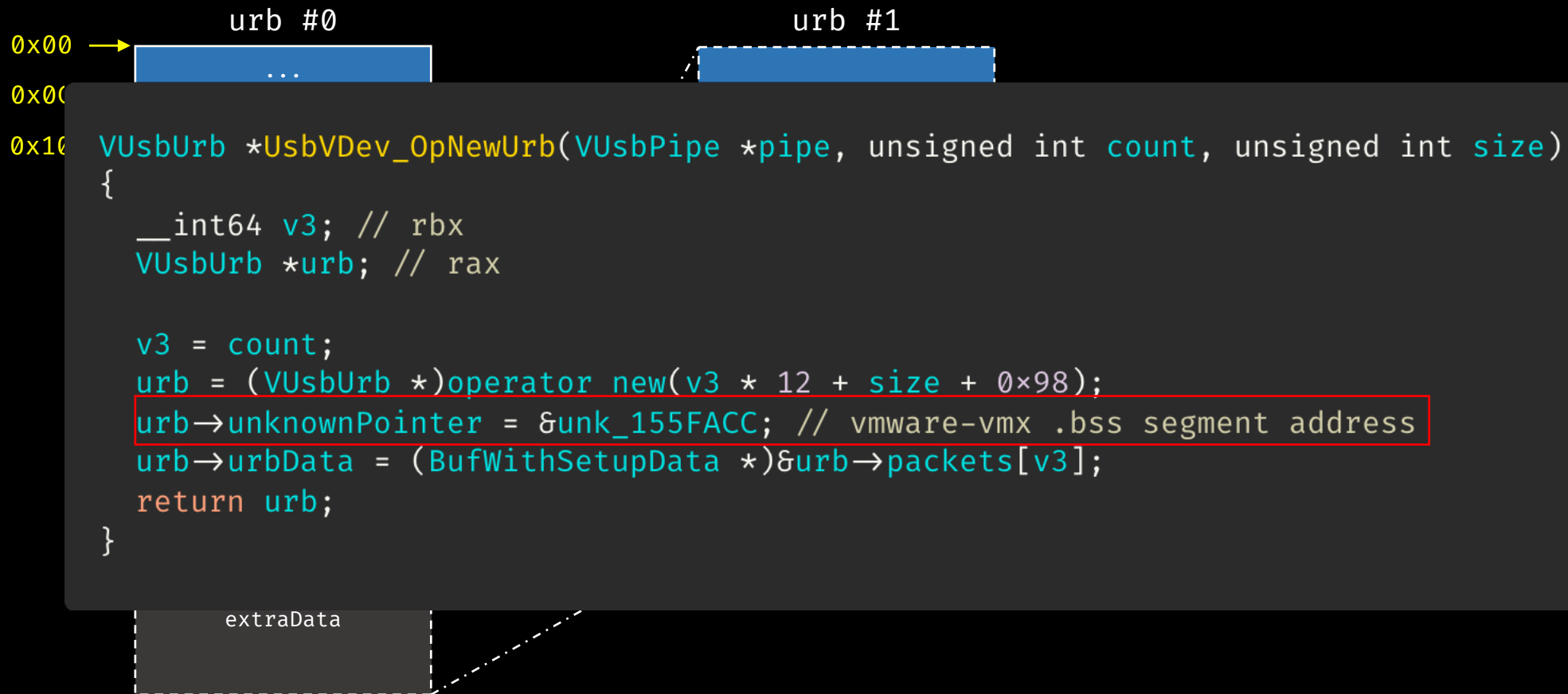
Info Leak

- Limited oob write + urb object -> more powerful oob read



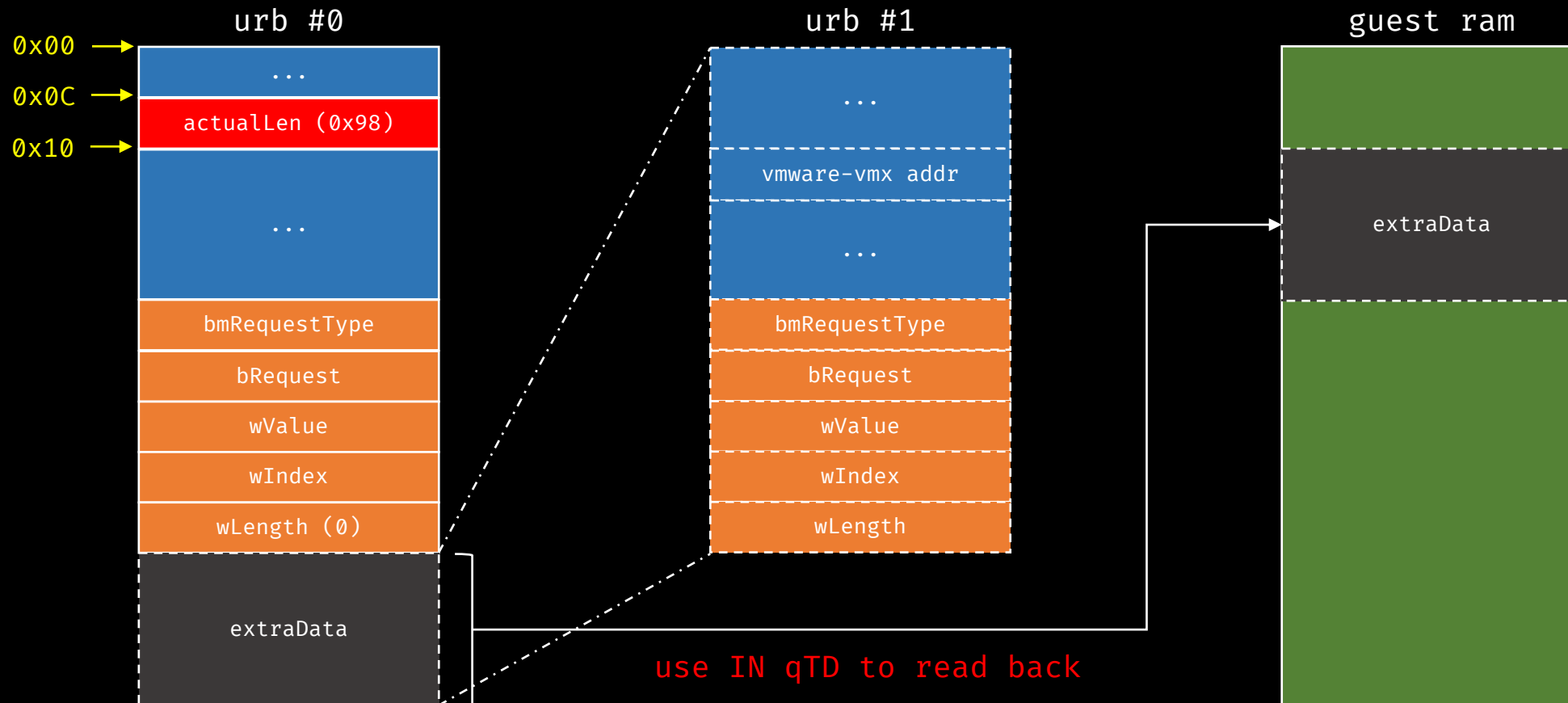
Info Leak

- Limited oob write + urb object -> more powerful oob read



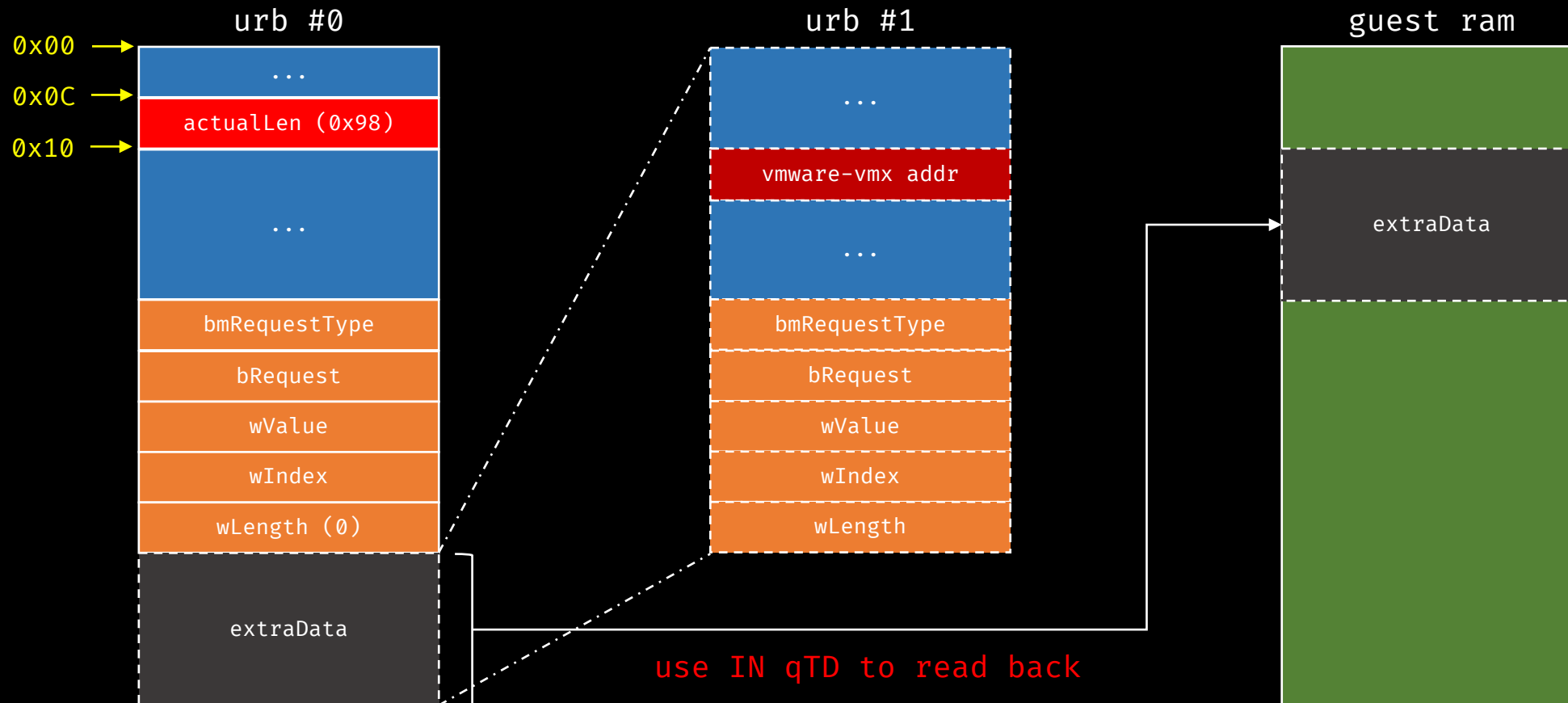
Info Leak

- Limited oob write + urb object -> more powerful oob read



Info Leak

- Limited oob write + urb object -> more powerful oob read

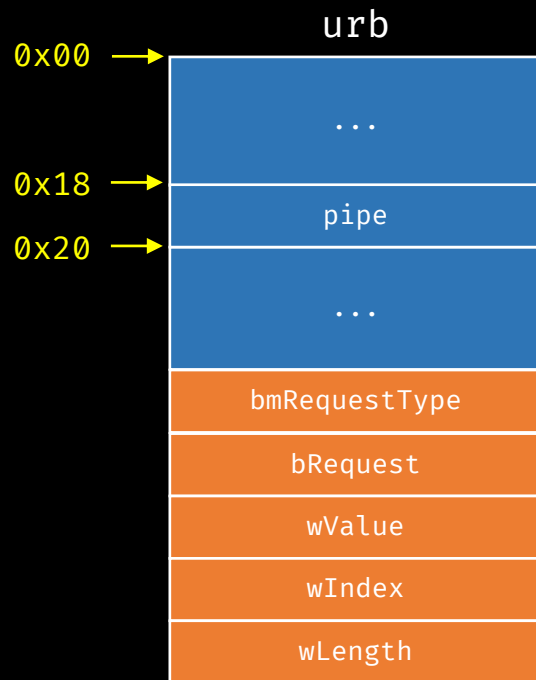


Hijack Control Flow

- Leverage *VUsb_CancelPipe*

Hijack Control Flow

- Leverage *VUsb_CancelPipe*

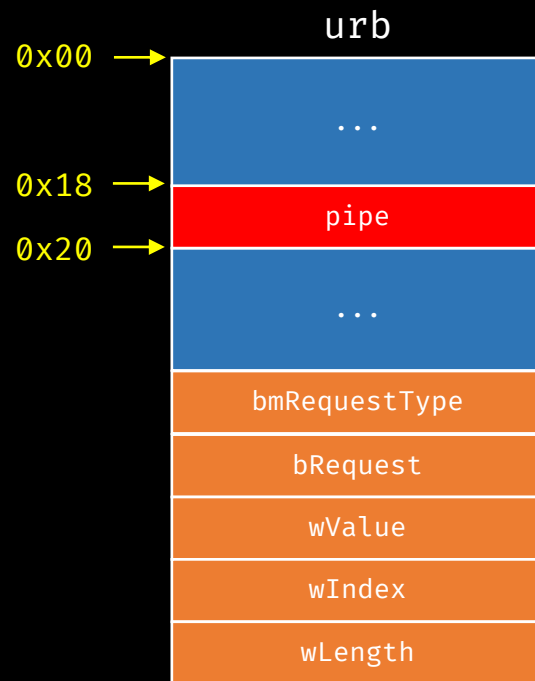


```
void VUsb_CancelPipe(VUsbPipe *pipe)
{
    VUsbPipe *next; // rbx
    void *v2; // rax
    VUsbUrb *v3; // rbp

    LogWithLevel(6u, "UsbDev: DevID(%lx): Cancel pipe(%p).\n", pipe->vusbDev->devID, pipe);
    ((void (__fastcall *)(VUsbDev *, _QWORD))pipe->vusbDev->backendBase->urbOps->opCancelPipe)(
        pipe->vusbDev,
        pipe->endpoint);
    next = (VUsbPipe *)pipe->urbList.next;
    while ( next != (VUsbPipe *)&pipe->urbList )
    {
        v2 = next;
        v3 = (VUsbUrb *)((char *)&next[-1].pipeList.prev + 1);
        next = (VUsbPipe *)next->usbDeviceInterface;
        VUsb_UrbRemoveFromPipe(v3, *((_DWORD *)v2 - 8), *((_DWORD *)v2 + 10));
        VUsb_PutUrb(v3);
    }
    pipe->pipeSize = 0;
    *((_QWORD *)&pipe->cBulkUrbs = 0LL;
}
```

Hijack Control Flow

- Leverage *VUsb_CancelPipe*



```
void VUsb_CancelPipe(VUsbPipe *pipe)
{
    VUsbPipe *next; // rbx
    void *v2; // rax
    VUsbUrb *v3; // rbp

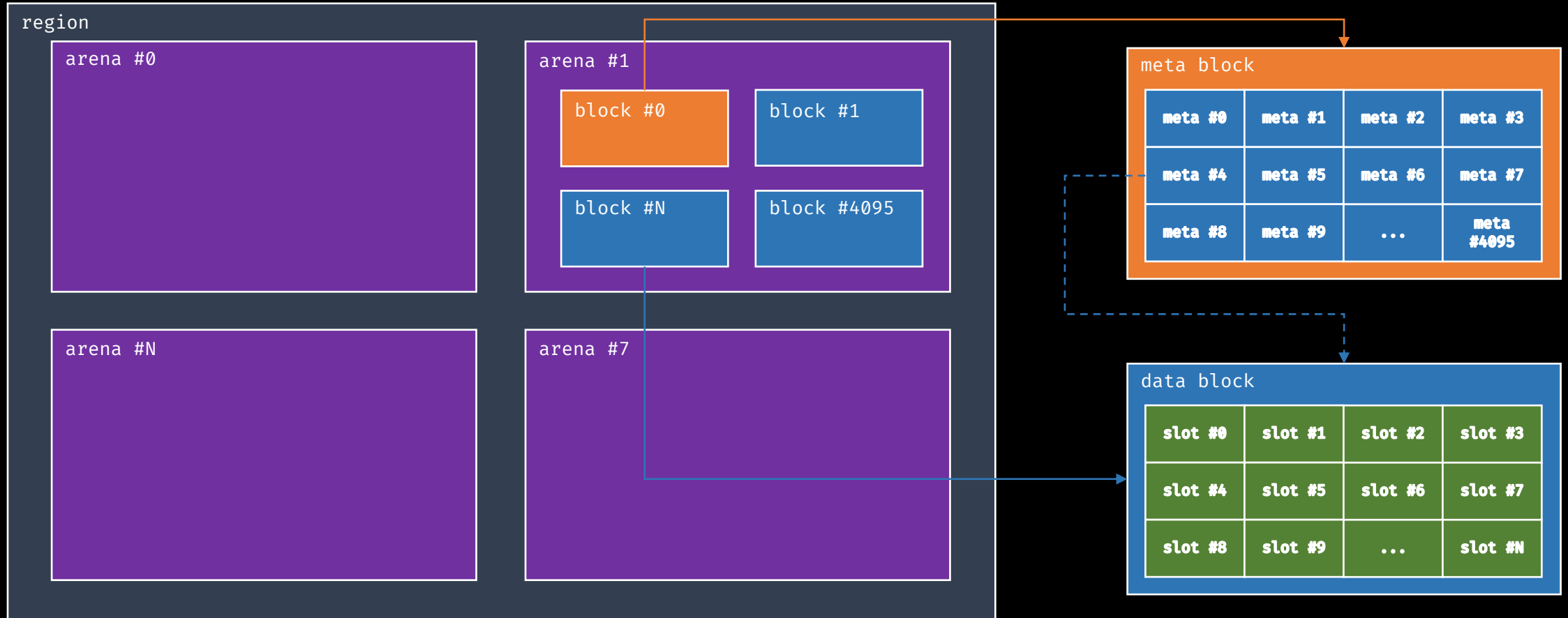
    LogWithLevel(6u, "UsbDev: DevID(%lx): Cancel pipe(%p).\n", pipe->vusbDev->devID, pipe);
    ((void (__fastcall *)(VUsbDev *, _QWORD))pipe->vusbDev->backendBase->urbOps->opCancelPipe)(
        pipe->vusbDev,
        pipe->endpoint);
    next = (VUsbPipe *)pipe->urbList.next;
    while ( next != (VUsbPipe *)&pipe->urbList )
    {
        v2 = next;
        v3 = (VUsbUrb *)((char *)&next[-1].pipeList.prev + 1);
        next = (VUsbPipe *)next->usbDeviceInterface;
        VUsb_UrbRemoveFromPipe(v3, *((_DWORD *)v2 - 8), *((_DWORD *)v2 + 10));
        VUsb_PutUrb(v3);
    }
    pipe->pipeSize = 0;
    *((_QWORD *)&pipe->cBulkUrbs = 0LL;
}
```

arbitrary function call

Challenges

- How to perform heap manipulation on MacOS
- Where to prepare the crafted pipe object

libmalloc (nano zone)



libmalloc (nano zone)

- The size class ranges from 0x10 to 0x100, aligned to 0x10
- Allocation of the same size might come from different blocks due to cpu switches
- Blocks are shuffled which brings certain randomization

	CPU #0	CPU #1	CPU #2	...	CPU #N
0x10	meta #0		meta #3023	...	meta #1092
0x20	meta #22	meta #188		...	meta #891
...	...	...	...	...	...
0x100	meta #489			...	meta #49

libmalloc (nano zone)

- Slot is allocated sequentially when all slots are available
- A free list (LIFO) is used to chain the freed slots when block is partially full

block meta
next_slot (0x7FB)
free_count
gen_count
in_use

slot #1	slot #2	slot #3	...	slot #N

block meta
next_slot (3)
free_count
gen_count
in_use

slot #1	slot #2	slot #3	...	slot #N
next_slot (N)		next_slot (1)		next_slot (0x7FB)
double_free_guard		double_free_guard		double_free_guard

Heap manipulation

1. Heap spray to take up the slot fragments of all cpu's blocks

block meta
next_slot (0x7FB)
free_count
gen_count
in_use

slot #1	slot #2	slot #3	...	slot #N

Heap manipulation

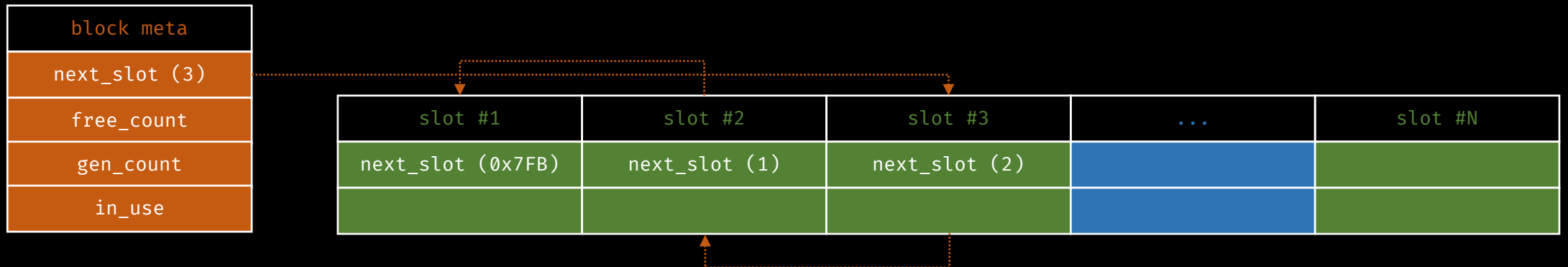
1. Heap spray to take up the slot fragments of all cpu's blocks
2. Allocate urbs on new blocks of all cpus

block meta
next_slot (0x7FB)
free_count
gen_count
in_use

urb #0	urb #1	urb #2	...	slot #N

Heap manipulation

1. Heap spray to take up the slot fragments of all cpu's blocks
2. Allocate urbs on new blocks of all cpus
3. Free some of the urbs in step 2



Heap manipulation

1. Heap spray to take up the slot fragments of all cpu's blocks
2. Allocate urbs on new blocks of all cpus
3. Free some of the urbs in step 2
4. Allocate three urbs from the free list

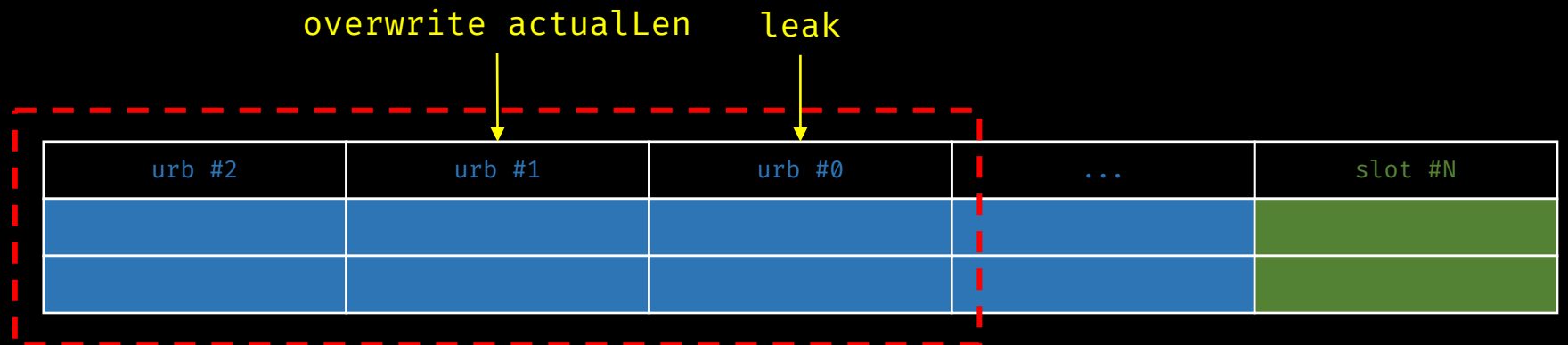
block meta
next_slot (0x7FB)
free_count
gen_count
in_use

urb #2	urb #1	urb #0	...	slot #N

Heap manipulation

1. Heap spray to take up the slot fragments of all cpu's blocks
2. Allocate urbs on new blocks of all cpus
3. Free some of the urbs in step 2
4. Allocate three urbs from the free list
5. Overwrite urb #1's *actualLen* to leak urb #0's object

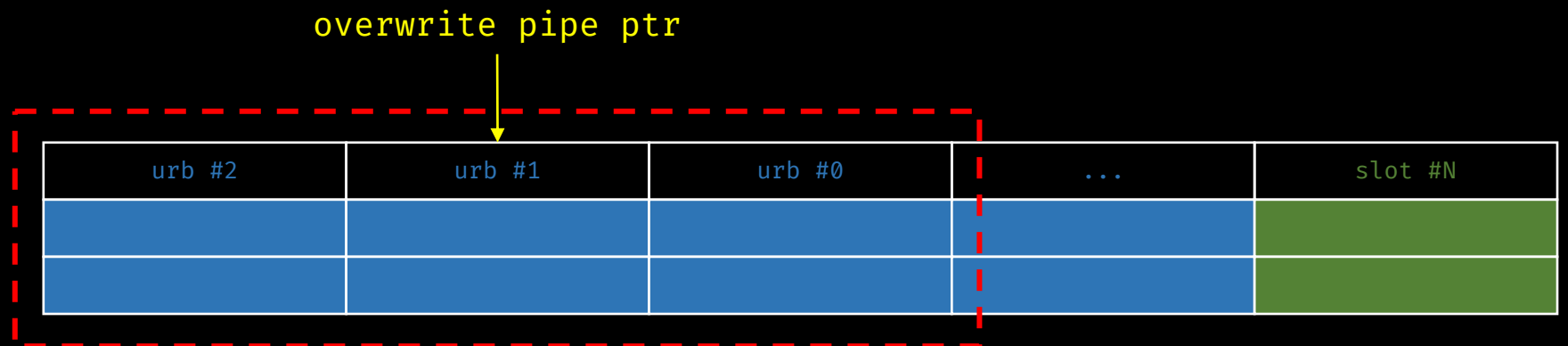
block meta
next_slot (0x7FB)
free_count
gen_count
in_use



Heap manipulation

1. Heap spray to take up the slot fragments of all cpu's blocks
2. Allocate urbs on new blocks of all cpus
3. Free some of the urbs in step 2
4. Allocate three urbs from the free list
5. Overwrite urb #1's *actuaLen* to leak urb #0's object
6. Overwrite urb #1's *pipe* pointer to hijack control flow

block meta
next_slot (0x7FB)
free_count
gen_count
in_use



Home for pipe object

Weak ASLR for Guest RAM

- Guest RAM always starts from 0x300000000



shared memory	300000000-308000000	[128.0M	960K	960K	127.1M]	rw-/rwx	SM=SHM
shared memory	308000000-310000000	[128.0M	1248K	1248K	126.8M]	rw-/rwx	SM=SHM
shared memory	310000000-318000000	[128.0M	768K	768K	127.2M]	rw-/rwx	SM=SHM
shared memory	318000000-320000000	[128.0M	1776K	1776K	126.3M]	rw-/rwx	SM=SHM
shared memory	320000000-328000000	[128.0M	1424K	1424K	126.6M]	rw-/rwx	SM=SHM
shared memory	328000000-330000000	[128.0M	960K	960K	127.1M]	rw-/rwx	SM=SHM
shared memory	330000000-338000000	[128.0M	752K	752K	127.3M]	rw-/rwx	SM=SHM
shared memory	338000000-340000000	[128.0M	928K	928K	127.1M]	rw-/rwx	SM=SHM
shared memory	340000000-348000000	[128.0M	992K	992K	127.0M]	rw-/rwx	SM=SHM
shared memory	348000000-350000000	[128.0M	608K	608K	127.4M]	rw-/rwx	SM=SHM
shared memory	350000000-358000000	[128.0M	928K	928K	127.1M]	rw-/rwx	SM=SHM
shared memory	358000000-360000000	[128.0M	704K	704K	127.3M]	rw-/rwx	SM=SHM
shared memory	360000000-368000000	[128.0M	1680K	1680K	126.4M]	rw-/rwx	SM=SHM
shared memory	368000000-370000000	[128.0M	1648K	1648K	126.4M]	rw-/rwx	SM=SHM
shared memory	370000000-378000000	[128.0M	1008K	1008K	127.0M]	rw-/rwx	SM=SHM
shared memory	378000000-380000000	[128.0M	1152K	1152K	126.9M]	rw-/rwx	SM=SHM
shared memory	380000000-388000000	[128.0M	2688K	2688K	125.4M]	rw-/rwx	SM=SHM
shared memory	388000000-390000000	[128.0M	4512K	4512K	123.6M]	rw-/rwx	SM=SHM
shared memory	390000000-398000000	[128.0M	26.7M	26.7M	101.3M]	rw-/rwx	SM=SHM
shared memory	398000000-3a0000000	[128.0M	45.0M	45.0M	83.0M]	rw-/rwx	SM=SHM
shared memory	3a0000000-3a8000000	[128.0M	128.0M	128.0M	0K]	rw-/rwx	SM=SHM
shared memory	3a8000000-3b0000000	[128.0M	128.0M	128.0M	0K]	rw-/rwx	SM=SHM
shared memory	3b0000000-3b8000000	[128.0M	128.0M	128.0M	0K]	rw-/rwx	SM=SHM
shared memory	3b8000000-3c0000000	[128.0M	128.0M	128.0M	0K]	rw-/rwx	SM=SHM
shared memory	3c0000000-3c8000000	[128.0M	128.0M	128.0M	0K]	rw-/rwx	SM=SHM
shared memory	3c8000000-3d0000000	[128.0M	128.0M	128.0M	0K]	rw-/rwx	SM=SHM
shared memory	3d0000000-3d8000000	[128.0M	128.0M	128.0M	0K]	rw-/rwx	SM=SHM
shared memory	3d8000000-3e0000000	[128.0M	128.0M	128.0M	0K]	rw-/rwx	SM=SHM
shared memory	3e0000000-3e8000000	[128.0M	128.0M	128.0M	0K]	rw-/rwx	SM=SHM
shared memory	3e8000000-3f0000000	[128.0M	128.0M	128.0M	0K]	rw-/rwx	SM=SHM
shared memory	3f0000000-3f8000000	[128.0M	128.0M	128.0M	0K]	rw-/rwx	SM=SHM
shared memory	3f8000000-400000000	[128.0M	128.0M	128.0M	0K]	rw-/rwx	SM=SHM

Weak ASLR for Guest RAM

```
bool __fastcall MemHostMap(__int64 a1, __int64 a2)
{
    ...
    v6 = mach_vm_allocate(mach_task_self_, &address, *(_QWORD *)(a2 + 24), v3 == 0);
    if ( v6 )
    {
        v7 = v6;
        Log("%s: Could not allocate %llu bytes: %x.\n", "MemHostMap", *(_QWORD *)(a2 + 24), v6);
        Msg_Append("0&!*0*0(msg.memMacOS.vmAllocateFailed)Could not allocate %llu bytes: %x.\n", *(_QWORD *)(a2 + 24),
v7);
        return 0;
    }
    ...
    *(_QWORD *)(a2 + 8) = address;
    return 1;
}
```

api: mach_vm_allocate

Weak ASLR for Guest RAM

```
# encoding: utf-8
import os

c_code_tmpl = '''
#include <mach/mach.h>
#include <mach/mach_vm.h>
#include <string.h>
#include <stdio.h>

#define MiB (1024ULL * 1024)
#define GiB (1024ULL * MiB)

int main(void) {
    mach_vm_address_t addr;
    kern_return_t kr = mach_vm_allocate(mach_task_self(), &addr, <<SIZE_FIELD>>, VM_FLAGS_ANYWHERE);

    printf("Allocate 2G memory at: %p\\n", (void *)addr);

    return 0;
}
'''

size_class = ['512 * MiB', '1 * GiB', '1 * GiB + 64 * MiB', '1 * GiB + 512 * MiB', '2 * GiB', '4 * GiB', '8 * GiB']

TIMES = 1000
for size in size_class:
    count = 0
    c_code = c_code_tmpl.replace('<<SIZE_FIELD>>', size)
    with open('test.c', 'w+', encoding='utf-8') as f:
        f.write(c_code)
    os.system('/usr/bin/clang test.c -o test')
    for i in range(TIMES):
        res = os.popen("./test", 'r').read()
        if "0x300000000" in res:
            count += 1
    print(f'[+] For size {size}, the probability is: {count / 1000 * 100}%')
```

```
> python3.12 analyze.py
[+] For size 512 * MiB, the probability is: 0.0%
[+] For size 1 * GiB, the probability is: 42.6%
[+] For size 1 * GiB + 64 * MiB, the probability is: 50.8%
[+] For size 1 * GiB + 512 * MiB, the probability is: 100.0%
[+] For size 2 * GiB, the probability is: 100.0%
[+] For size 4 * GiB, the probability is: 100.0%
[+] For size 8 * GiB, the probability is: 100.0%
```

Weak ASLR for Guest RAM

```
# encoding: utf-8
import os

c_code_tmpl = '''
#include <mach/mach.h>
#include <mach/mach_vm.h>
#include <string.h>
#include <stdio.h>

#define MiB (1024ULL * 1024)
#define GiB (1024ULL * MiB)

int main(void) {
    mach_vm_address_t addr;
    kern_return_t kr = mach_vm_allocate(mach_task_self(), &addr, <<SIZE_FIELD>>, VM_FLAGS_ANYWHERE);

    printf("Allocate 2G memory at: %p\\n", (void *)addr);

    return 0;
}
'''

size_class = ['512 * MiB', '1 * GiB', '1 * GiB + 64 * MiB', '1 * GiB + 512 * MiB', '2 * GiB', '4 * GiB', '8 * GiB']

TIMES = 1000
for size in size_class:
    count = 0
    c_code = c_code_tmpl.replace('<<SIZE_FIELD>>', size)
    with open('test.c', 'w+', encoding='utf-8') as f:
        f.write(c_code)
    os.system('/usr/bin/clang test.c -o test')
    for i in range(TIMES):
        res = os.popen("./test", 'r').read()
        if "0x300000000" in res:
            count += 1
    print(f'[+] For size {size}, the probability is: {count / 1000 * 100}%')
```

```
> python3.12 analyze.py
[+] For size 512 * MiB, the probability is: 0.0%
[+] For size 1 * GiB, the probability is: 42.6%
[+] For size 1 * GiB + 64 * MiB, the probability is: 50.8%
[+] For size 1 * GiB + 512 * MiB, the probability is: 100.0%
[+] For size 2 * GiB, the probability is: 100.0%
[+] For size 4 * GiB, the probability is: 100.0%
[+] For size 8 * GiB, the probability is: 100.0%
```

As long as the size is bigger enough, mach_vm_allocate will always return memory from a fixed address.

Pipe object's home

1. Prepare the crafted pipe object at Guest RAM
2. Set the pipe object pointer as *0x280000000 + pipe's GPA*
3. Hijack control flow without further address information 

Demo on Mac M1 Pro



Exploit on Mac Intel

- UHCI is available -- an extra virtual video device is not needed

```
kangel@kangel-virtual-machine:~$ lsusb -tv
/: Bus 02.Port 1: Dev 1, Class=root_hub, Driver=uhci_hcd/2p, 12M
   ID 1d6b:0001 Linux Foundation 1.1 root hub
   |__ Port 1: Dev 2, If 0, Class=Human Interface Device, Driver=usbhid, 12M
       ID 0e0f:0003 VMware, Inc. Virtual Mouse
   |__ Port 2: Dev 3, If 0, Class=Hub, Driver=hub/7p, 12M
       ID 0e0f:0002 VMware, Inc. Virtual USB Hub
       |__ Port 1: Dev 4, If 0, Class=Wireless, Driver=btusb, 12M
           ID 0e0f:0008 VMware, Inc.
       |__ Port 1: Dev 4, If 1, Class=Wireless, Driver=btusb, 12M
           ID 0e0f:0008 VMware, Inc.
/: Bus 01.Port 1: Dev 1, Class=root_hub, Driver=ehci-pci/6p, 480M
   ID 1d6b:0002 Linux Foundation 2.0 root hub
   |__ Port 1: Dev 2, If 0, Class=Video, Driver=uvcvideo, 480M
       ID 0e0f:000b VMware, Inc.
   |__ Port 1: Dev 2, If 1, Class=Video, Driver=uvcvideo, 480M
       ID 0e0f:000b VMware, Inc.
```

Exploit on Mac Intel

- UHCI is available -- an extra virtual video device is not needed
- Guest RAM address is randomized

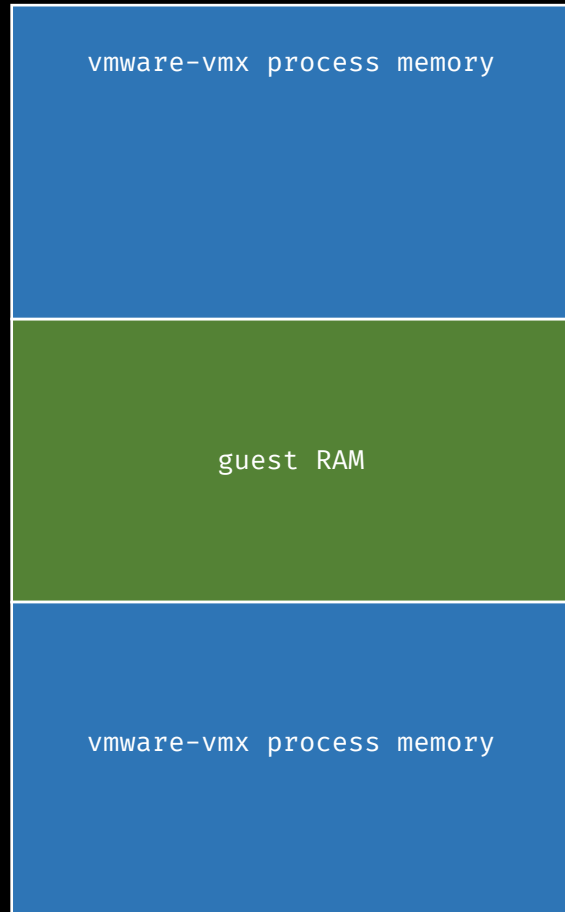
__TEXT	1037f9000-104919000	[17.1M	8496K	0K	0K]	r-x/r-x	SM=COW	/Applications/VMware Fusion.app/Contents/Library/vmware-vmx
shared memory	10891b000-11091b000	[128.0M	127.9M	127.9M	0K]	rw-/rwx	SM=SHM	
shared memory	11091b000-11891b000	[128.0M	128.0M	128.0M	0K]	rw-/rwx	SM=SHM	
shared memory	11891b000-12091b000	[128.0M	128.0M	128.0M	0K]	rw-/rwx	SM=SHM	
shared memory	12091b000-12891b000	[128.0M	128.0M	128.0M	0K]	rw-/rwx	SM=SHM	
shared memory	12891b000-13091b000	[128.0M	128.0M	128.0M	0K]	rw-/rwx	SM=SHM	
shared memory	13091b000-13891b000	[128.0M	128.0M	128.0M	0K]	rw-/rwx	SM=SHM	
shared memory	13891b000-14091b000	[128.0M	128.0M	128.0M	0K]	rw-/rwx	SM=SHM	
shared memory	14091b000-14891b000	[128.0M	127.2M	127.2M	0K]	rw-/rwx	SM=SHM	
shared memory	14891b000-15091b000	[128.0M	124.0M	124.0M	0K]	rw-/rwx	SM=SHM	
shared memory	15091b000-15891b000	[128.0M	128.0M	128.0M	0K]	rw-/rwx	SM=SHM	
shared memory	15891b000-16091b000	[128.0M	128.0M	128.0M	0K]	rw-/rwx	SM=SHM	
shared memory	16091b000-16891b000	[128.0M	128.0M	128.0M	0K]	rw-/rwx	SM=SHM	
shared memory	16891b000-17091b000	[128.0M	128.0M	128.0M	0K]	rw-/rwx	SM=SHM	
shared memory	17091b000-17891b000	[128.0M	28.0M	28.0M	0K]	rw-/rwx	SM=SHM	
shared memory	17891b000-18091b000	[128.0M	0K	0K	0K]	rw-/rwx	SM=SHM	
shared memory	18091b000-18891b000	[128.0M	63.6M	63.6M	0K]	rw-/rwx	SM=SHM	

Defeat Guest RAM's ASLR

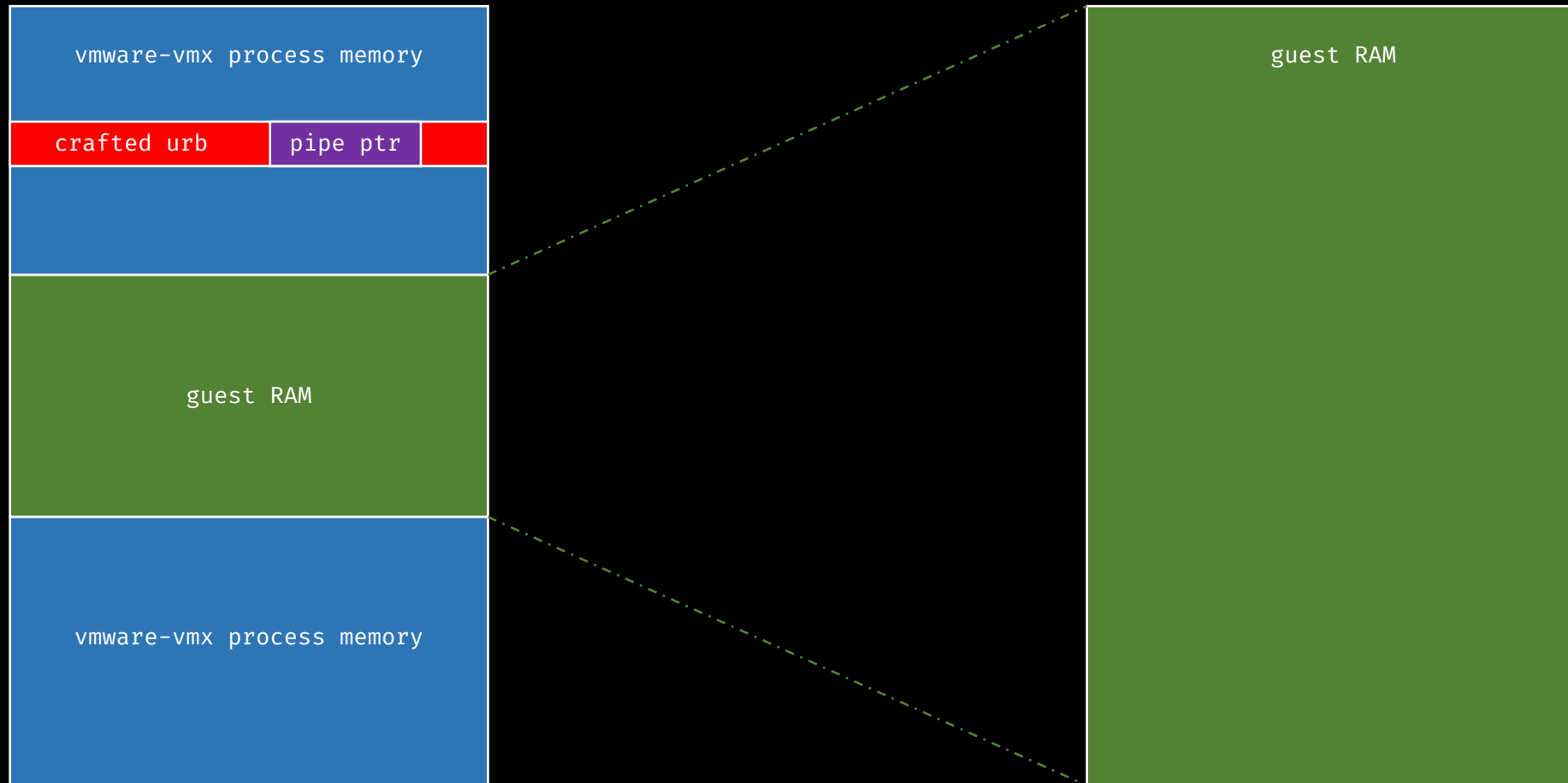
- We could get the vmware-vmx's base address
- The offset between vmware-vmx and guest RAM ranges roughly from *0x5000000 to 0x5000000 + 0x1000000* on every startup

__TEXT	1037f9000-104919000	[17.1M	8496K	0K	0K]	r-x/r-x	SM=COW	/Applications/VMware Fusion.app/Contents/Library/vmware-vmx
shared memory	10891b000-11091b000	[128.0M	127.9M	127.9M	0K]	rw-/rwx	SM=SHM	
shared memory	11091b000-11891b000	[128.0M	128.0M	128.0M	0K]	rw-/rwx	SM=SHM	
shared memory	11891b000-12091b000	[128.0M	128.0M	128.0M	0K]	rw-/rwx	SM=SHM	
shared memory	12091b000-12891b000	[128.0M	128.0M	128.0M	0K]	rw-/rwx	SM=SHM	
shared memory	12891b000-13091b000	[128.0M	128.0M	128.0M	0K]	rw-/rwx	SM=SHM	
shared memory	13091b000-13891b000	[128.0M	128.0M	128.0M	0K]	rw-/rwx	SM=SHM	
shared memory	13891b000-14091b000	[128.0M	128.0M	128.0M	0K]	rw-/rwx	SM=SHM	
shared memory	14091b000-14891b000	[128.0M	127.2M	127.2M	0K]	rw-/rwx	SM=SHM	
shared memory	14891b000-15091b000	[128.0M	124.0M	124.0M	0K]	rw-/rwx	SM=SHM	
shared memory	15091b000-15891b000	[128.0M	128.0M	128.0M	0K]	rw-/rwx	SM=SHM	
shared memory	15891b000-16091b000	[128.0M	128.0M	128.0M	0K]	rw-/rwx	SM=SHM	
shared memory	16091b000-16891b000	[128.0M	128.0M	128.0M	0K]	rw-/rwx	SM=SHM	
shared memory	16891b000-17091b000	[128.0M	128.0M	128.0M	0K]	rw-/rwx	SM=SHM	
shared memory	17091b000-17891b000	[128.0M	28.0M	28.0M	0K]	rw-/rwx	SM=SHM	
shared memory	17891b000-18091b000	[128.0M	0K	0K	0K]	rw-/rwx	SM=SHM	
shared memory	18091b000-18891b000	[128.0M	63.6M	63.6M	0K]	rw-/rwx	SM=SHM	

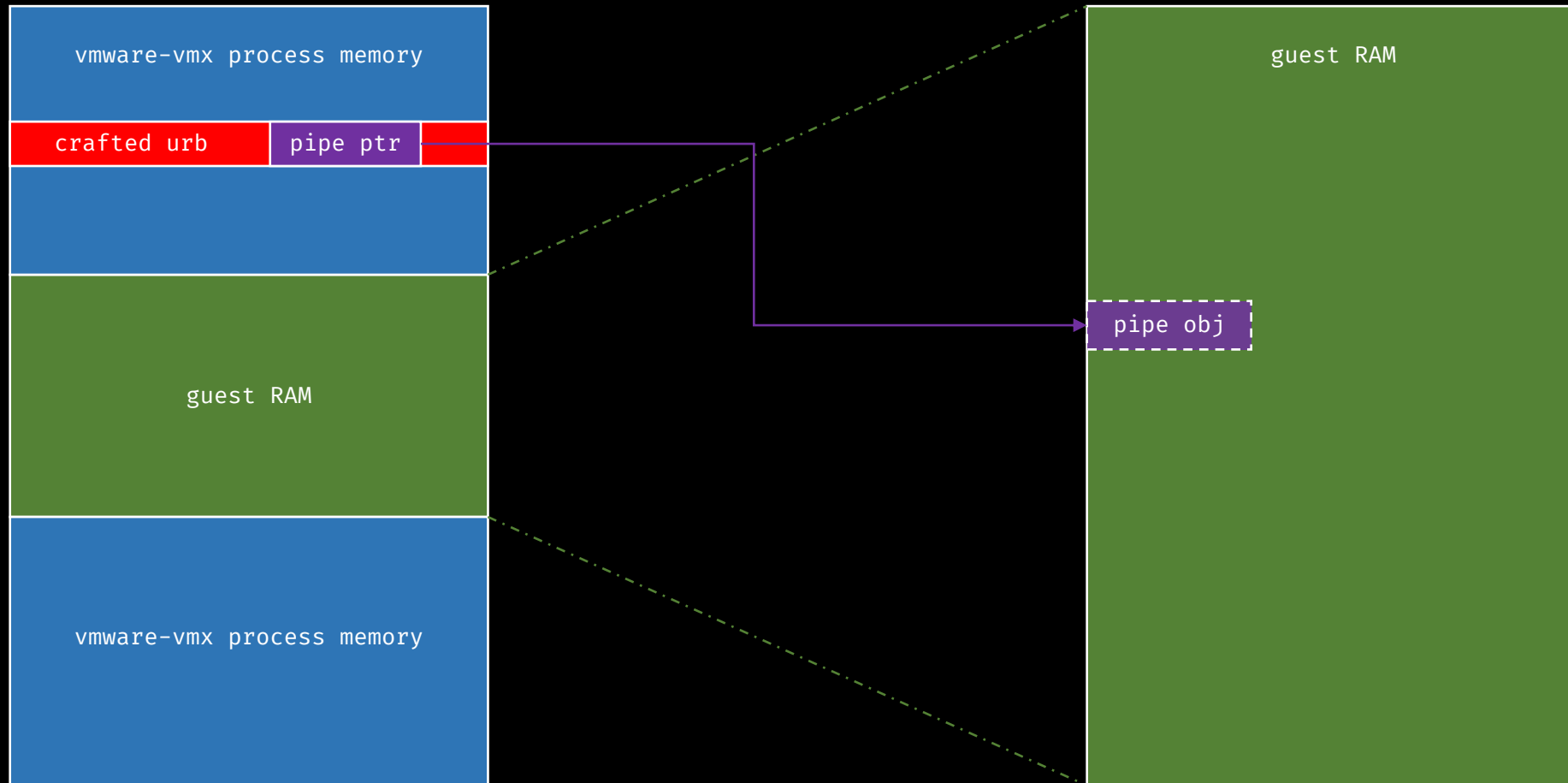
Defeat Guest RAM's ASLR



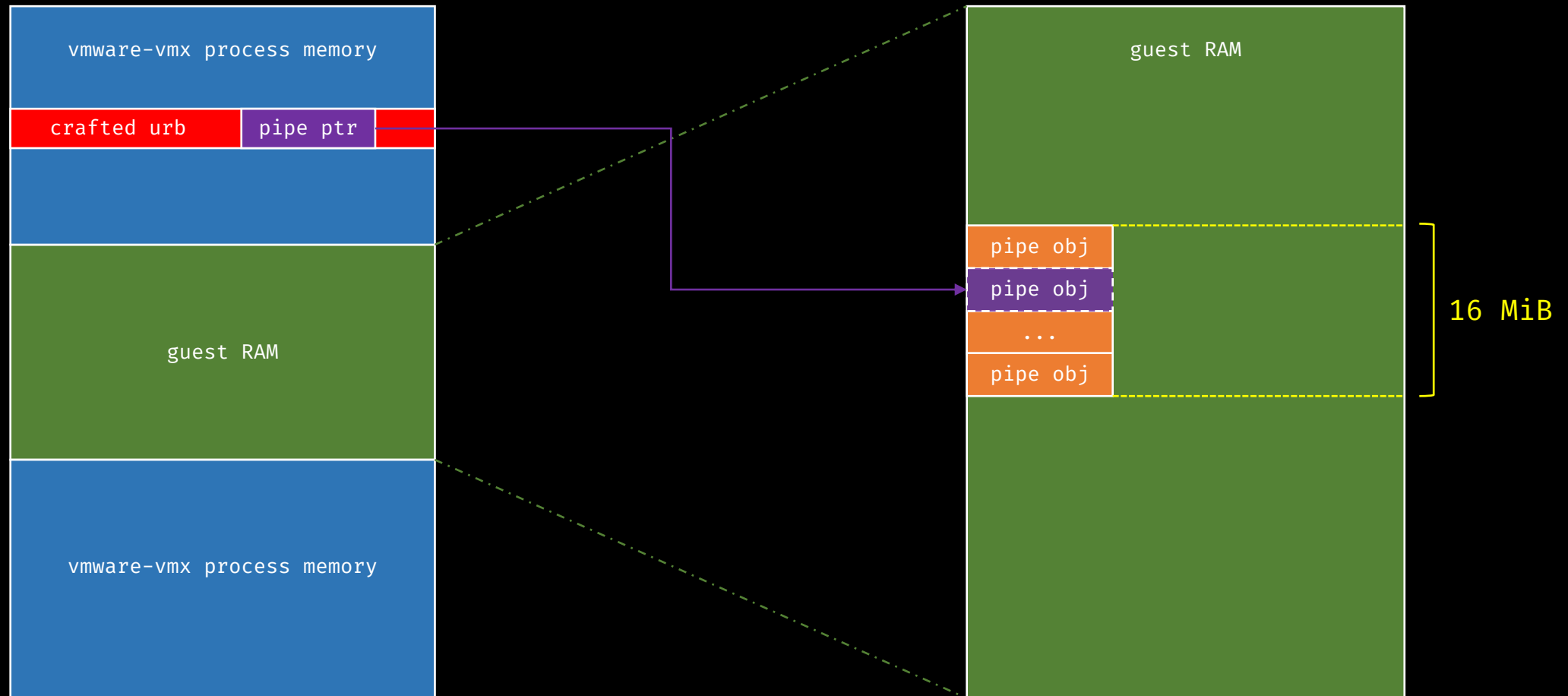
Defeat Guest RAM's ASLR



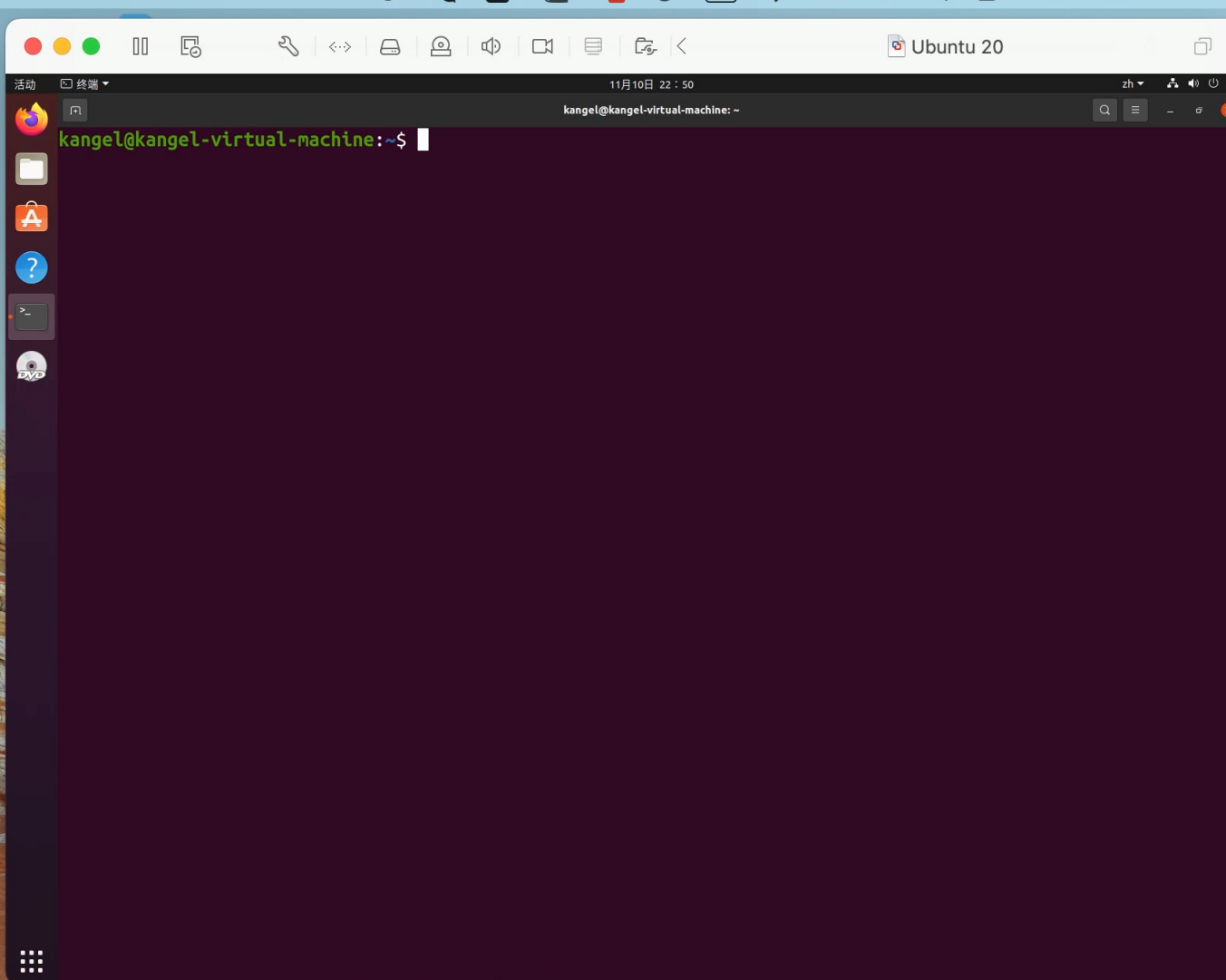
Defeat Guest RAM's ASLR



Defeat Guest RAM's ASLR



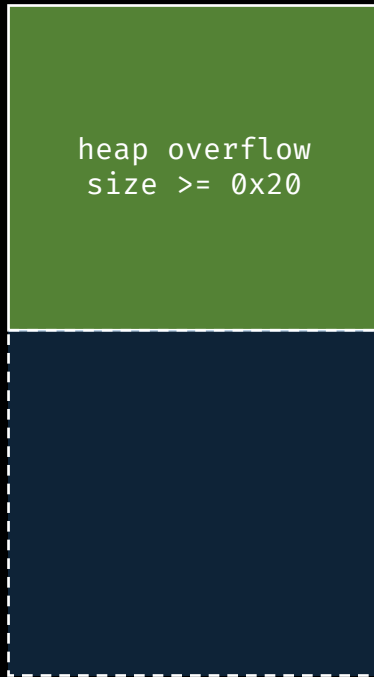
Demo on Mac Intel



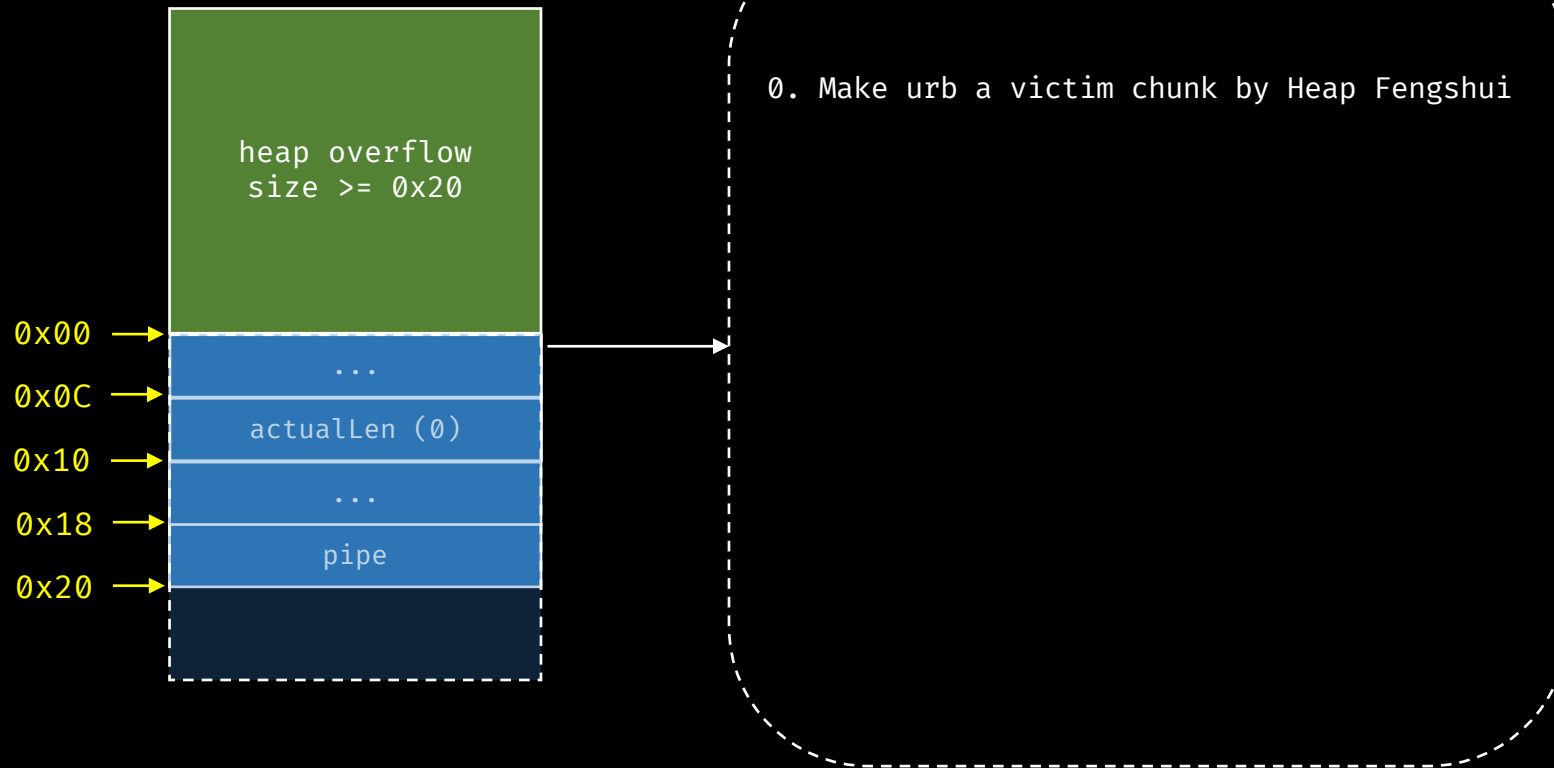
Agenda

- Introduction
- A TOCTOU bug
- Exploitation development
- Conclusion

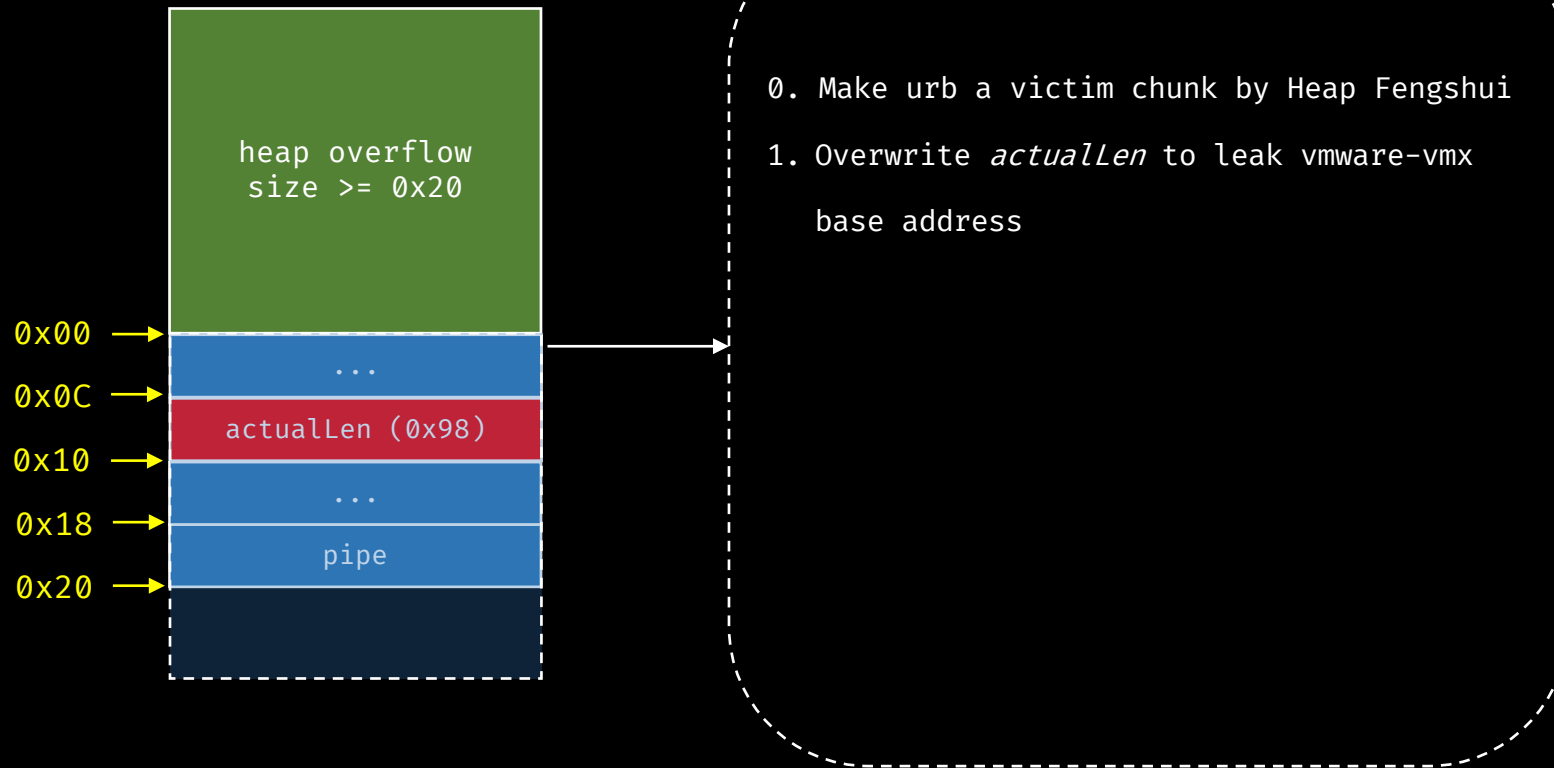
Conclusion



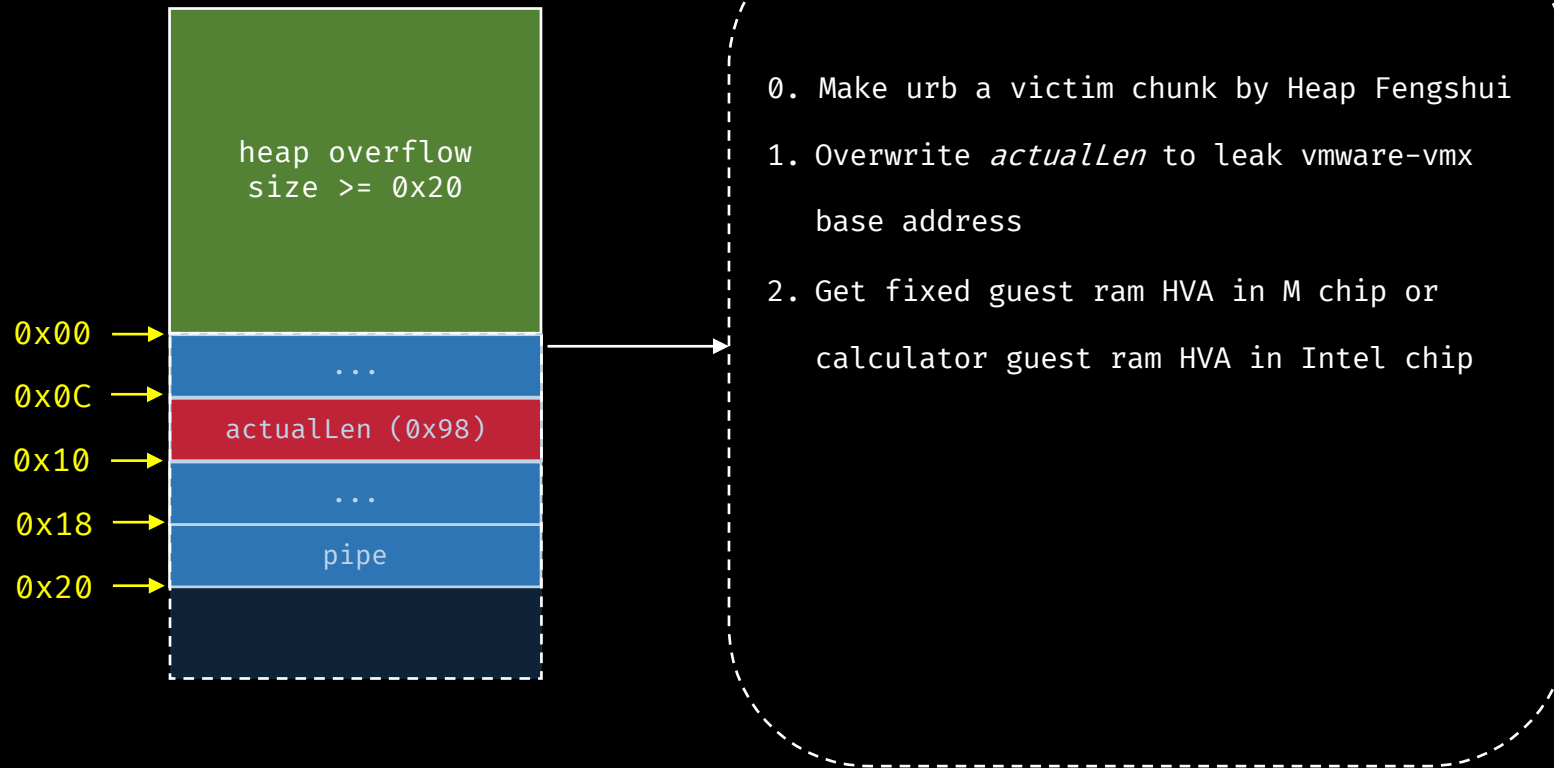
Conclusion



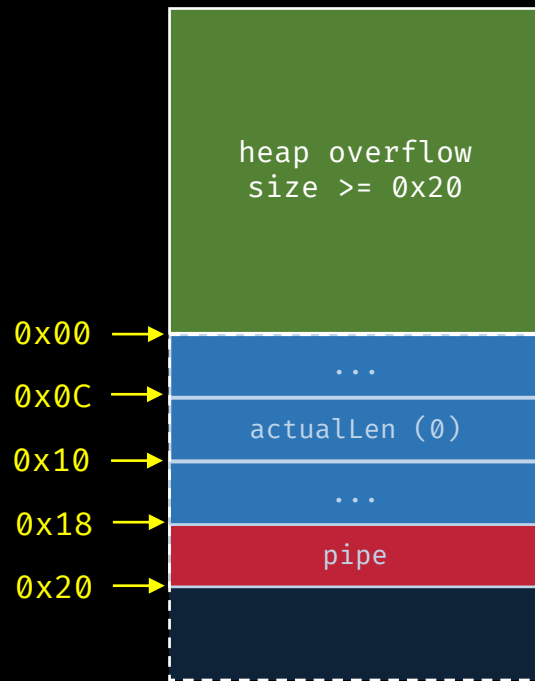
Conclusion



Conclusion

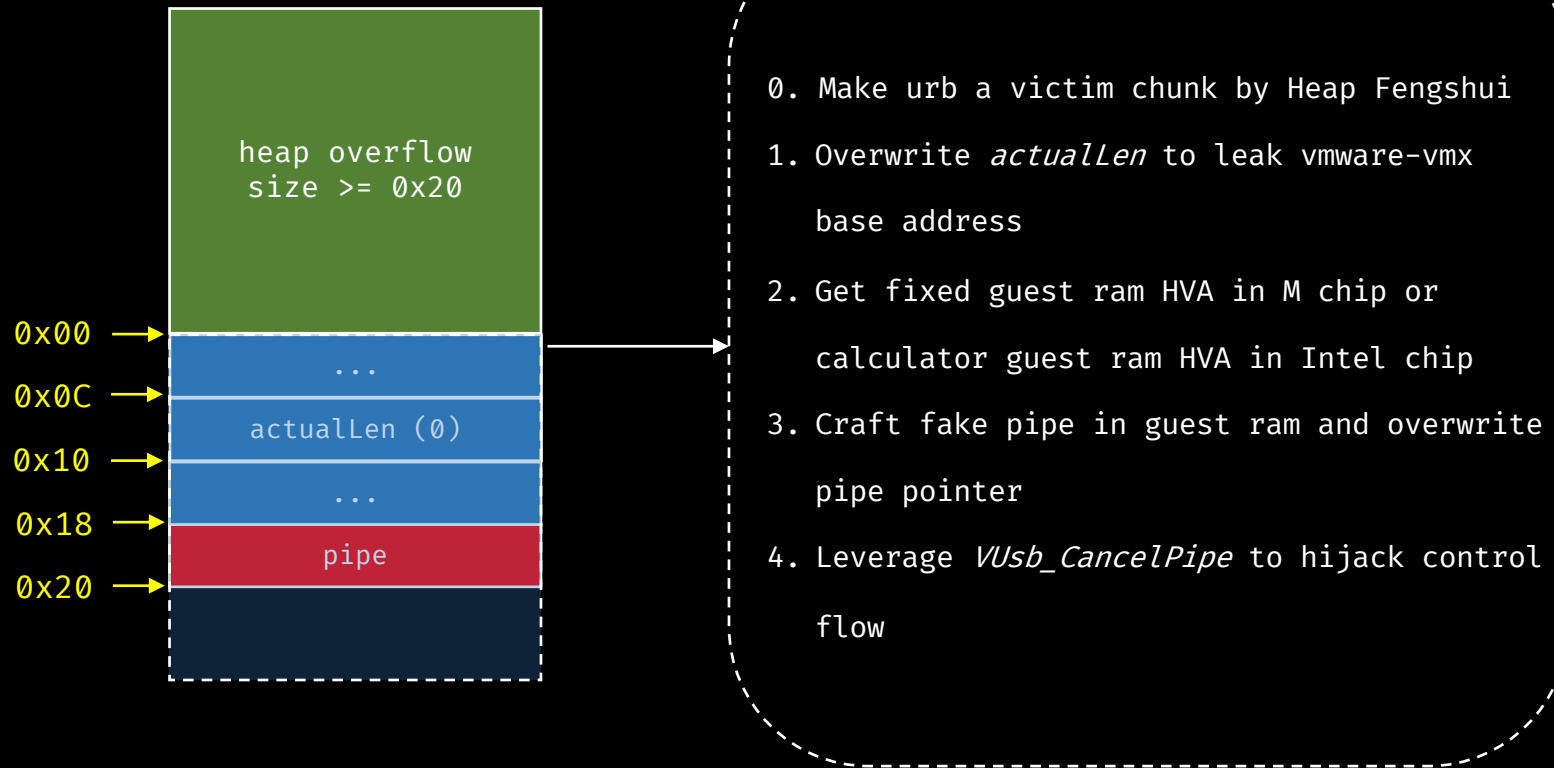


Conclusion

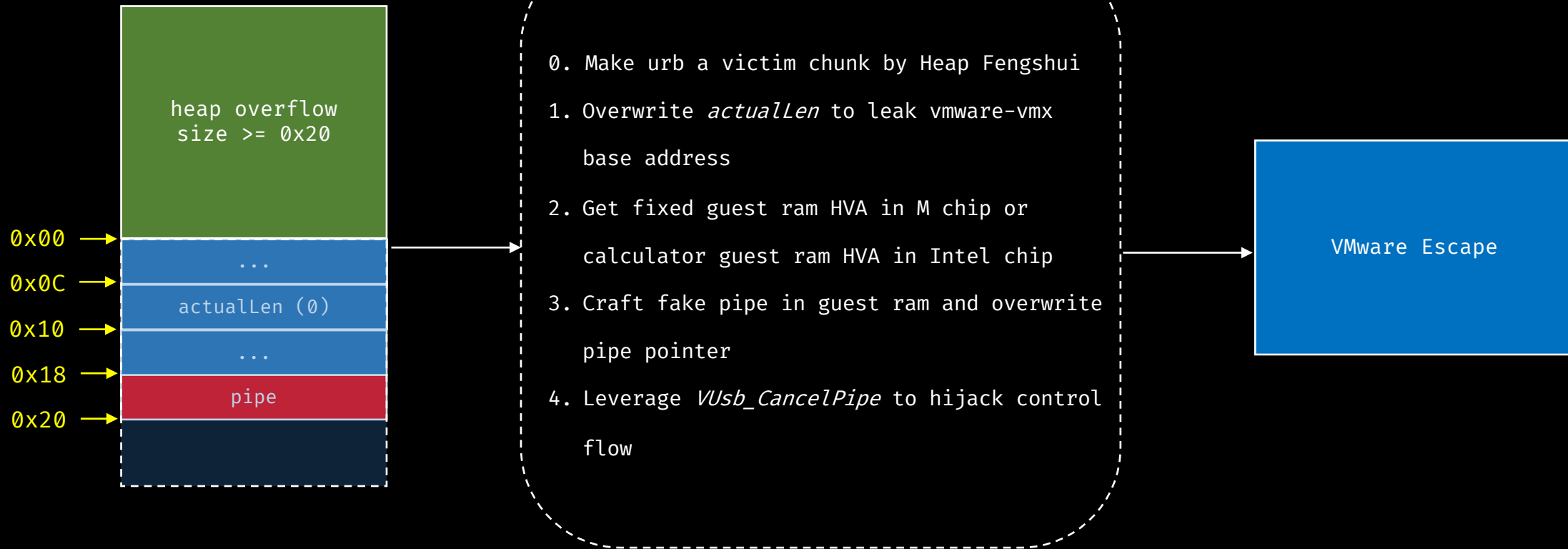


0. Make urb a victim chunk by Heap Fengshui
1. Overwrite *actualLen* to leak vmware-vmx base address
2. Get fixed guest ram HVA in M chip or calculator guest ram HVA in Intel chip
3. Craft fake pipe in guest ram and overwrite pipe pointer

Conclusion

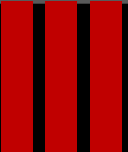


Conclusion

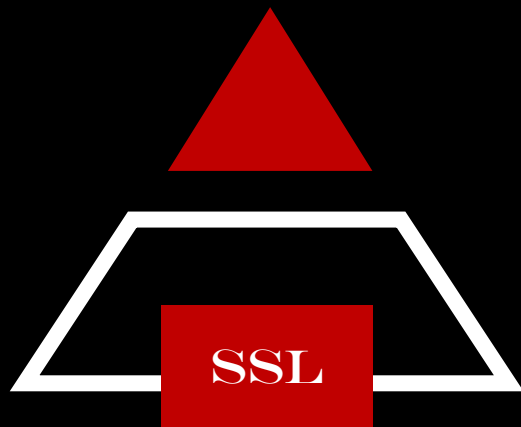


Takeaways

- Reproducing old bugs is a good way to find new bugs
- Don't care too much about the results, just enjoy the research process



Thank you for listening!



Contact us:

Yongkang Jia(kangel@zju.edu.cn, @j_kangel)

Xiao Lei(nop.leixiao@gmail.com, @N0p_0rz)