HITB SECCONF 2023 Phuket

Timekiller: Leveraging Asynchronous Clock to Escape from QEMU/KVM

Yongkang Jia
Xiao Lei
Yiming Tao
Gaoning Pan (Zhejiang University)
Chunming Wu (Zhejiang University)

Zhejiang University & Singular Security Lab

About us

Yongkang Jia (kangel)

- Master of Zhejiang University (graduated)
- Security researcher at Singular Security Lab
- Former CTF player at AAA team
- Research interest: Virtualization security
- (@J_kangel)

Xiao Lei (Nop)

- Master of Zhejiang University
- Intern at Singular Security Lab
- CTF player at AAA team
- Research interest: Virtualization security

Yiming Tao

- Master of Zhejiang University
- Research interest: Virtualization security

Agenda

- Background
- Asynchronous Clock
- Virtio Crypto
- Virtio Device Fuzzing
- Vulnerabilities
- Exploit
- Conclusion

Background

QEMU/KVM Introduction

- QEMU/KVM is a open source virtualization framework
- QEMU
  - Device virtulization(network, display, usb, cryptography, etc)
- KVM
  - CPU virtualization
  - Memory virtualization
  - Interrupt virtualization

The Research Surface of QEMU

- New attack surface
  - GPU virtualization
  - race condition bugs
- New exploit skill
  - common exploitation skills for stack overflow vulnerabilities
  - common exploitation skills for heap overflow vulnerabilities
  - common exploitation skills for Use-After-Free vulnerabilities

Why We Start Our Research?

- There are more race condition bugs in other virtualization products but less in QEMU

Do race condition bugs widely exist in QEMU?

Asynchronous Clock

Asynchronous Nature

- Why?
  - Avoid bloking
- How?
  - Multithreading
  - Timer(Asychronous Clock)

QEMU's Threading Model

- I/O thread(just one)
  - poll, alarm signal, event, callback function
  - BH
  - Timer
- Vcpu thread
  - Each vcpu has its own thread
- Other worker thread
  - VNC, spice, migration...

QEMUTimer

- Real time clock
  - runs even when the VM is stopped
- Virtual clock
  - runs when the VM is running
- Host clock
  - runs when the VM is suspend, but is sensitive to time changes to the system clock
- Realtime clock used for icount warp
  - the same as @QEMU_CLOCK_VIRTUAL outside icount mode

```
typedef enum {
    QEMU_CLOCK_REALTIME = 0,
    QEMU_CLOCK_VIRTUAL = 1,
    QEMU_CLOCK_HOST = 2,
    QEMU_CLOCK_VIRTUAL_RT = 3,
    QEMU_CLOCK_MAX
} QEMUClockType;
```

QEMUTimer

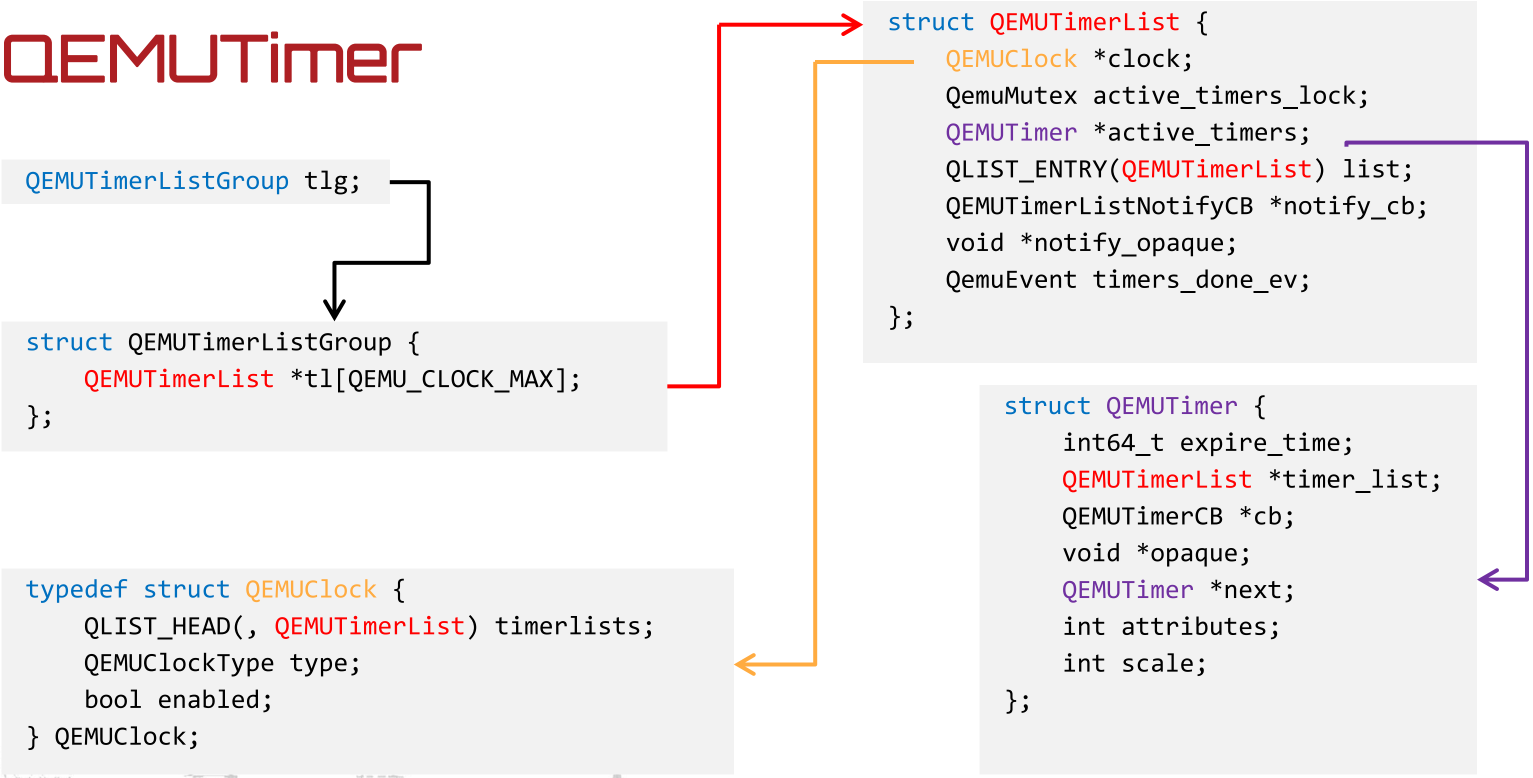
```
QEMUTimerListGroup tlg;
```

```
struct QEMUTimerListGroup {
    QEMUTimerList *tl[QEMU_CLOCK_MAX];
};
```

```
typedef struct QEMUClock {
    QLIST_HEAD(, QEMUTimerList) timerlists;
    QEMUClockType type;
    bool enabled;
} QEMUClock;
```

```
struct QEMUTimerList {
    QEMUClock *clock;
    QemuMutex active_timers_lock;
    QEMUTimer *active_timers;
    QLIST_ENTRY(QEMUTimerList) list;
    QEMUTimerListNotifyCB *notify_cb;
    void *notify_opaque;
    QemuEvent timers_done_ev;
};
```

```
struct QEMUTimer {
    int64_t expire_time;
    QEMUTimerList *timer_list;
    QEMUTimerCB *cb;
    void *opaque;
    QEMUTimer *next;
    int attributes;
    int scale;
};
```



What Can QEMUTimer Do ?

- Handle request(Network,USB,Disk,Crypto,etc)
- Fuzzing(V-SHUTTLE)
- Exploit

Handle Request

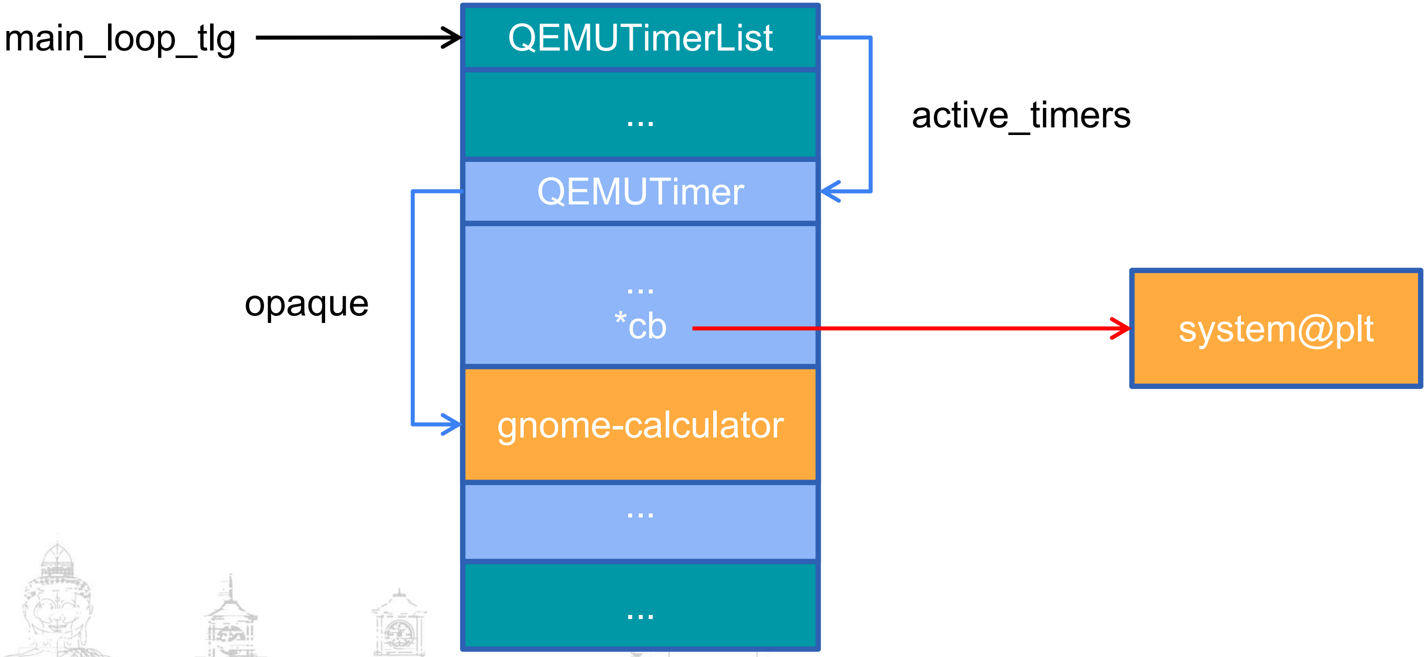
- Network
  - e1000
- USB
  - ehci, uhci, xhci
- Disk
  - fdc
- Crypto
  - virtio-crypto

Fuzzing

- V-SHUTTLE
  - https://github.com/hustdebug/v-shuttle/blob/main/V-Shuttle-M/fuzz-util.h#L311

```
void setup_process_mode(void) {
    ...
        if(is_fuzzing()) {
            sleep(1);
            _afl_init_forkserver();
            fuzz_timer = timer_new_ns(QEMU_CLOCK_VIRTUAL, fuzzing_entry, NULL);
            timer_mod(fuzz_timer, qemu_clock_get_ns(QEMU_CLOCK_VIRTUAL));
        }
    ...
}
```

Exploit


main_loop_tlg
QEMUTimerList
...
active_timers
QEMUTimer
opaque
...
*cb
system@plt
gnome-calculator
...
...

Throttle -- Introdution

- QEMU includes a throttling module that can be used to set limits to I/O operations.
- It is currently used to limit the number of bytes per second and operations per second (IOPS) when performing disk I/O.

Throttle -- Using

- Bytes per second(throttle-bps)
- Operation per second(throttle-ops)
- For detail
    - https://github.com/qemu/qemu/blob/master/docs/throttle.txt

Virtio Crypto

Introduction

- A virtual cryptography device under virtio device framework
- Provides a set of unified operation interfaces for different cryptography services
- For more information about virtio-crypto
    - https://wiki.qemu.org/Features/VirtioCrypto

Why We Choose Virtio Crypto

Why We Choose Virtio Crypto

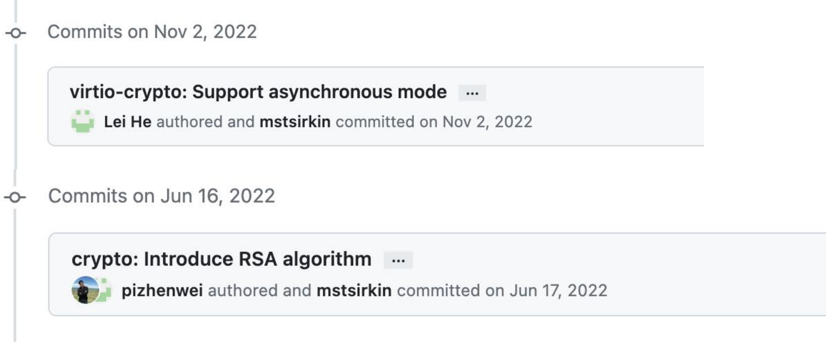
- Cryptography used widely
  - Wireless, telecom, data center, enterprise systems

Why We Choose Virtio Crypto

- Cryptography used widely
  - Wireless, telecom, data center, enterprise systems
- Continuously updating

Why We Choose Virtio Crypto

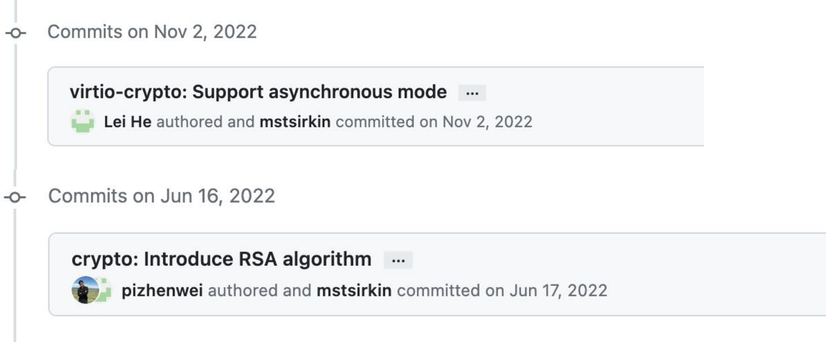
- Cryptography used widely
  - Wireless, telecom, data center, enterprise systems
- Continuously updating



Why We Choose Virtio Crypto

- Cryptography used widely
  - Wireless, telecom, data center, enterprise systems
- Continuously updating

New Features May
Mean New Bugs



Why We Choose Virtio Crypto

- Cryptography used widely
  - Wireless, telecom, data center, enterprise systems
- Continuously updating
- lack of research recently

Commits on Nov 2, 2022

virtio-crypto: Support asynchronous mode ...

Lei He authored and **mstsirkin** committed on Nov 2, 2022

Commits on Jun 16, 2022

Vulnerability Details : CVE-2017-5931

Integer overflow in hw/virtio/virtio-crypto.c in QEMU (aka Quick Emulator) allows local guest OS privileged users to cause a denial of service (QEMU process crash) or possibly execute arbitrary code on the host via a crafted virtio-crypto request, which triggers a heap-based buffer overflow.

Publish Date : 2017-03-27 Last Update Date : 2023-02-12

Why We Choose Virtio Crypto

- Cryptography used widely
  - Wireless, telecom, data center, enterprise systems
- Continuously updating
- lack of research recently
- Asynchronous nature

Commits on Nov 2, 2022

virtio-crypto: Support asynchronous mode ...

Lei He authored and **mstsirkin** committed on Nov 2, 2022

Commits on Jun 16, 2022

Vulnerability Details : CVE-2017-5931

Integer overflow in hw/virtio/virtio-crypto.c in QEMU (aka Quick Emulator) allows local guest OS privileged users to cause a denial of service (QEMU process crash) or possibly execute arbitrary code on the host via a crafted virtio-crypto request, which triggers a heap-based buffer overflow.

Publish Date : 2017-03-27 Last Update Date : 2023-02-12

From Virtio's Perspective

- Control queue （one）
  - Session management for symmetric or asymmetric service
  - Facilitate control operations for device
- Data queue （1 - 1023）
  - Transport channel for crypto service requests

```
static void virtio_crypto_device_realize
    (DeviceState *dev, Error **errp)
{
    VirtIODevice *vdev = VIRTIO_DEVICE(dev);
    for (i = 0; i < vcrypto->max_queues; i++) {
        vcrypto->vqs[i].dataq =
           virtio_add_queue(vdev, 1024,
           virtio_crypto_handle_dataq_bh);
           ...
        }

        vcrypto->ctrl_vq = virtio_add_queue(vdev,
        1024, virtio_crypto_handle_ctrl);
        ...
}
```

Request of Control Queue

- General header: virtio_crypto_ctrl_header
- The opcode defines the type of session

```
struct virtio_crypto_ctrl_header {
    uint32_t opcode;
    uint32_t algo;
    uint32_t flag;
    /* data virtqueue id */
    uint32_t queue_id;
};
```

```
/* The request of the control virtqueue's packet */
struct virtio_crypto_op_ctrl_req {
    struct virtio_crypto_ctrl_header header;
    union {
    struct virtio_crypto_sym_create_session_req
        sym_create_session;
    ...
    struct virtio_crypto_akcipher_create_session_req
        akcipher_create_session;
    ...
    } u;
};
```

Request of Data Queue

- General header: virtio_crypto_op_header
- The opcode defines the type of request

```
struct virtio_crypto_op_header {
    uint32_t opcode;
    /* algo should be service-specific algorithms */
    uint32_t algo;
    /* session_id should be service-specific algorithms */
    uint64_t session_id;
    /* control flag to control the request */
    uint32_t flag;
    uint32_t padding;
};
```

```
/* The request of the data virtqueue's packet */
struct virtio_crypto_op_data_req {
    struct virtio_crypto_op_header header;
    union {
        struct virtio_crypto_sym_data_req sym_req
        struct virtio_crypto_hash_data_req hash_req;
        struct virtio_crypto_mac_data_req mac_req;
        struct virtio_crypto_aead_data_req aead_req;
        struct virtio_crypto_akcipher_data_req
akcipher_req;
        uint8_t padding[48];
    } u;
};
```

Symmetric Crypto Service

- Support algorithm
  - AES
- Operation info structure
  - CryptoDevBackendSymOpInfo

```
typedef struct CryptoDevBackendSymOpInfo {
    uint32_t aad_len;
    uint32_t iv_len;
    uint32_t src_len;
    uint32_t dst_len;
    uint32_t digest_result_len;
    uint32_t hash_start_src_offset;
    uint32_t cipher_start_src_offset;
    uint32_t len_to_hash;
    uint32_t len_to_cipher;
    uint8_t op_type;
    uint8_t *iv;
    uint8_t *src;
    uint8_t *dst;
    uint8_t *aad_data;
    uint8_t *digest_result;
    uint8_t data[];
} CryptoDevBackendSymOpInfo;
```

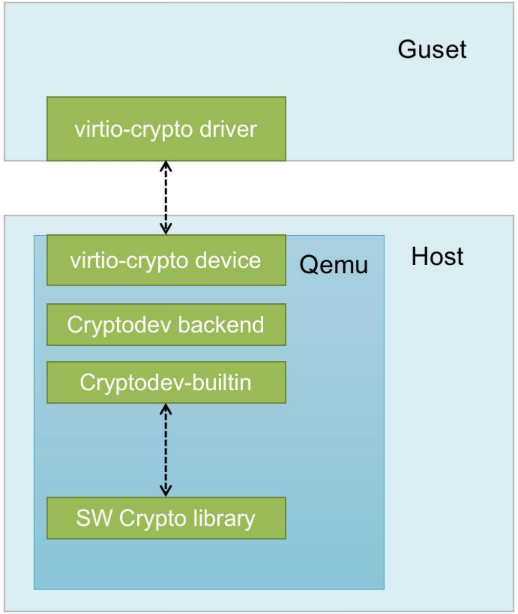
Asymmetric Crypto Service

- Support algorithm
  - RSA
- Operation info structure
  - CryptoDevBackendAsymOpInfo
- TODO
  - support DSA&ECDSA until qemu crypto framework support these

```
typedef struct CryptoDevBackendAsymOpInfo {
    uint32_t src_len;
    uint32_t dst_len;
    uint8_t *src;
    uint8_t *dst;
} CryptoDevBackendAsymOpInfo;
```

Overview of Virtio-crypto

- Guest
  - virtio-crypto user space pmd driver
  - LKCF based kernel space driver
- Host
  - virtio-crypto device inside QEMU
  - Finally call SW Crypto library， such as qcrypto builtin driver, libgcrypt, libnettle, etc



Virtio crypto Mode

- Synchronous mode
  1. Get op_info from guest
  2. Do operation immediately
  3. Free op_info

- Asynchronous mode
  1. Get op_info from guest
  2. Add op_info into queue
  3. Keep op_info chunk

```c
int cryptodev_backend_crypto_operation(
CryptoDevBackend *backend, CryptoDevBackendOpInfo *op_info)
{
    int ret;
    if (!throttle_enabled(&backend->tc)) {
    goto do_account;
    }

    if (throttle_schedule_timer(&backend->ts, &backend->tt, true) ||
    !QTAILQ_EMPTY(&backend->opinfos)) {
    QTAILQ_INSERT_TAIL(&backend->opinfos, op_info, next);
    return 0;
    }
do_account:
    ...
    return cryptodev_backend_operation(backend, op_info);
}
```

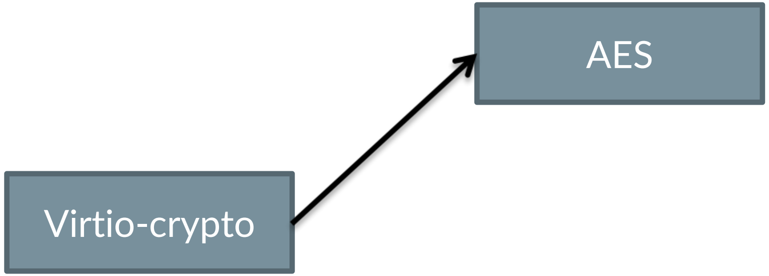
Asynchronous Mode

- Command

```
-object cryptodev-backend-builtin,id=cryptodev0,throttle-bps=32,throttle-ops=10
-device virtio-crypto-pci,id=crypto0,cryptodev=cryptodev0
```

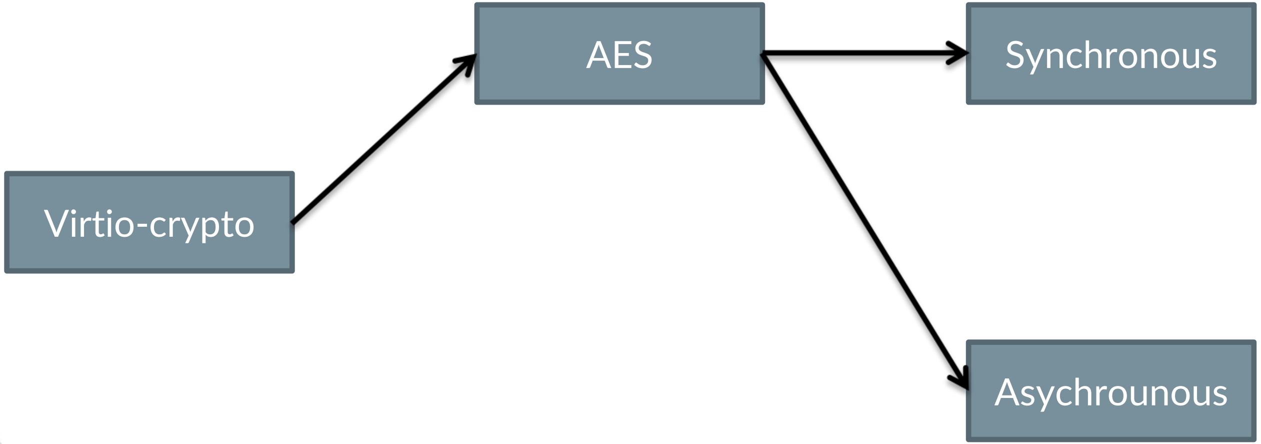
- Throttle
  - throttle-bps： the number of bytes per second
  - throttle-ops： the number of operations per second (IOPS).

Summary

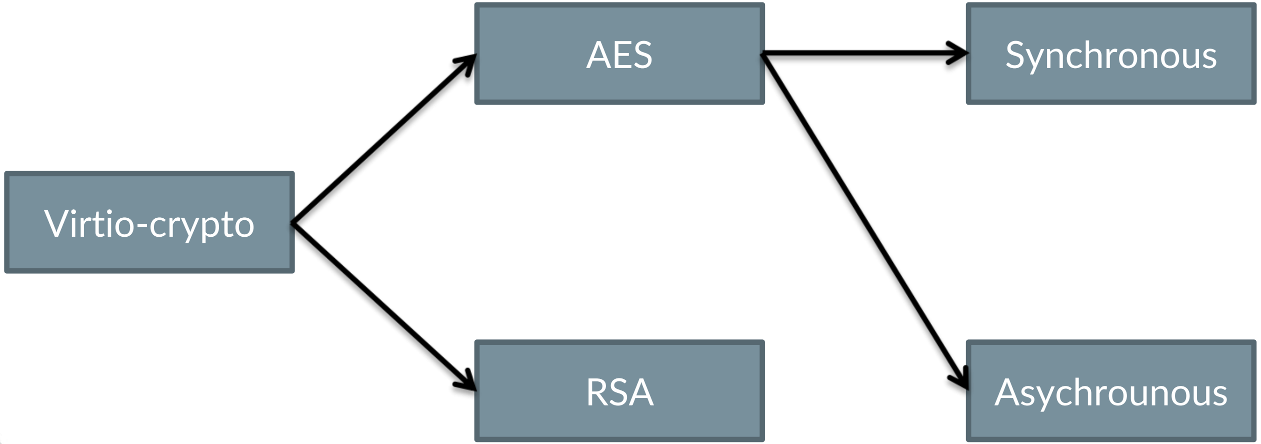


Summary

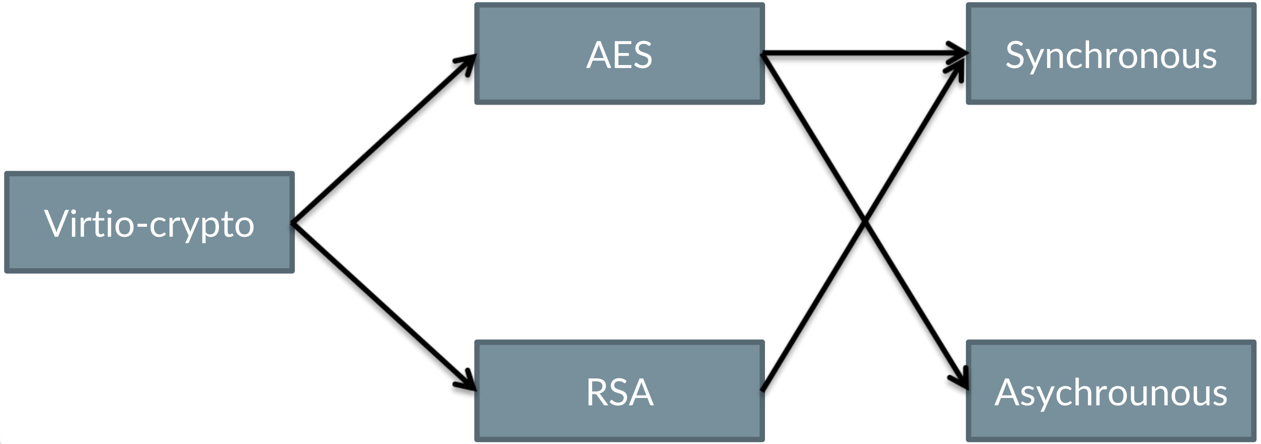
Summary



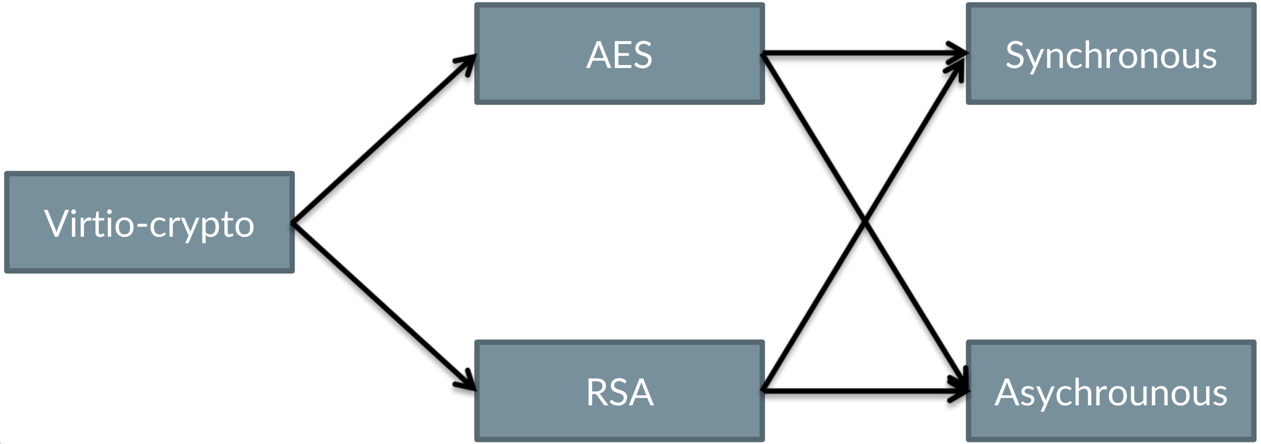
Summary



Summary



Summary



Virtio Device Fuzzing

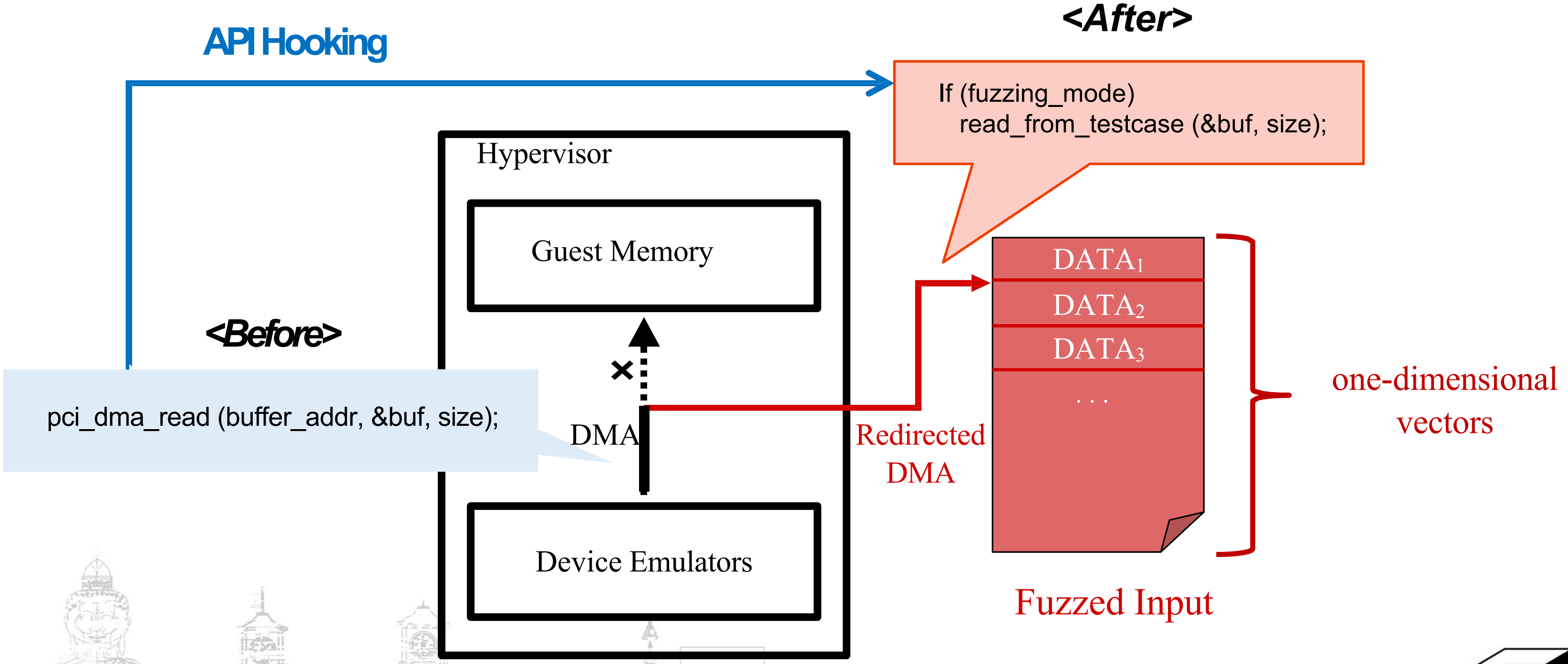
Before Fuzzing

- Which Fuzzer？
  - libfuzzer in qemu (Unfamiliar)
  - AFL (More modification need)
  - V-Shuttle (My favorite， just need less modification)
- How many target our Fuzzer can adapt？
  - Just virtio-crypto(too limited)
  - Whole virtual device (more work)
  - Virtio device

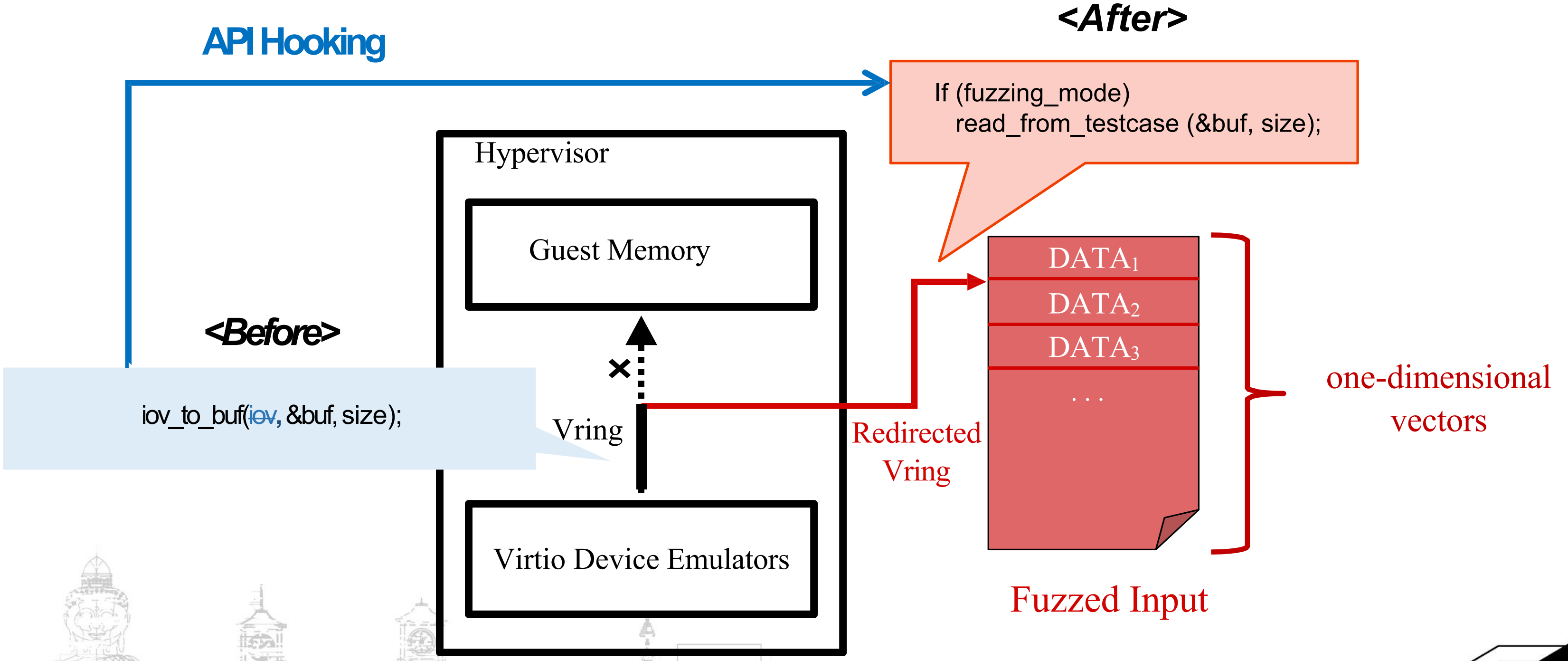
Modify V-SHUTTLE

- Initial operation
  - crate vring buffer
  - init virtio by call a series virtio_pci_common_write
- Hook data interaction
  - iov to buf
- Log redirection
  - stdout, stderr -> log_file

V-SHUTTLE


API Hooking
<After>
If (fuzzing_mode)
read_from_testcase (&buf, size);
Hypervisor
Guest Memory
<Before>
pci_dma_read (buffer_addr, &buf, size);
DMA
Redirected DMA
DATA1
DATA2
DATA3
. . .
one-dimensional vectors
Device Emulators
Fuzzed Input

Modified V-SHUTTLE



Crash

```
                 american fuzzy lop 2.52b (qemu-system-x86_64)

┌─ process timing ─────────────────────────────────────┬─ overall results ─────┐
│        run time : 0 days, 0 hrs, 0 min, 18 sec       │  cycles done : 0      │
│   last new path : 0 days, 0 hrs, 0 min, 0 sec        │  total paths : 228    │
│ last uniq crash : none seen yet                      │ uniq crashes : 0      │
│  last uniq hang : none seen yet                      │   uniq hangs : 0      │
├─ cycle progress ────────────────────┬─ map coverage ─┴───────────────────────┤
│  now processing : 1 (0.44%)         │    map density : 0.42% / 12.25%        │
│ paths timed out : 0 (0.00%)         │ count coverage : 1.64 bits/tuple       │
├─ stage progress ────────────────────┼─ findings in depth ────────────────────┤
│  now trying : havoc                 │ favored paths : 15 (6.58%)             │
│ stage execs : 4838/32.8k (14.76%)   │  new edges on : 116 (50.88%)           │
│ total execs : 23.8k                 │ total crashes : 0 (0 unique)           │
│  exec speed : 2115/sec              │  total tmouts : 0 (0 unique)           │
├─ fuzzing strategy yields ───────────┴───────────────┬─ path geometry ────────┤
│   bit flips : 18/480, 10/479, 8/477                 │    levels : 2          │
│  byte flips : 3/60, 0/59, 0/57                      │   pending : 228        │
│ arithmetics : 43/3360, 4/3916, 3/3258               │  pend fav : 15         │
│  known ints : 1/138, 1/506, 9/1022                  │ own finds : 139        │
│  dictionary : 0/0, 0/0, 0/137                       │  imported : n/a        │
│       havoc : 0/0, 0/0                              │ stability : 4.96%      │
│        trim : 0.00%/14, 0.00%                       ├────────────────────────┘
└─────────────────────────────────────────────────────┘          [cpu000:  9%]
[-] PROGRAM ABORT : Unable to request new process from fork server (OOM?)
         Location : run_target(), afl-fuzz.c:2377
```

After Fuzzing

- Coverage

	Hit	Total	Coverage
Lines:	528	661	79.9 %
Functions:	31	37	83.8 %
Branches:	146	231	63.2 %

- Vulnerability
  - 4 bugs reported, 1 CVE assigned

Vulnerabilities

Vulnerabilities

- NPD in virtio_crypto_free_request
- NPD in cryptodev_backend_account
- NPD in cryptodev_builtin_operation
- CVE-2023-3180 : Heap-based buffer overflow

1. NPD in virtio_crypto_free_request

```
--- a/hw/virtio/virtio-crypto.c
+++ b/hw/virtio/virtio-crypto.c
@@ -476,15 +476,17 @@ static void virtio_crypto_free_request(VirtIOCryptoReq
*req)
         size_t max_len;
         CryptoDevBackendSymOpInfo *op_info = req->op_info.u.sym_op_info;

-        max_len = op_info->iv_len +
-                  op_info->aad_len +
-                  op_info->src_len +
-                  op_info->dst_len +
-                  op_info->digest_result_len;
-
-        /* Zeroize and free request data structure */
-        memset(op_info, 0, sizeof(*op_info) + max_len);
-        g_free(op_info);
+        if (op_info) {
+            max_len = op_info->iv_len +
+                      op_info->aad_len +
+                      op_info->src_len +
+                      op_info->dst_len +
+                      op_info->digest_result_len;
+
+            /* Zeroize and free request data structure */
+            memset(op_info, 0, sizeof(*op_info) + max_len);
+            g_free(op_info);
+        }
```

- This function trigger in the end of the encrypt/decrypt process
- Root cause:no check for the op_info
- This flaw results in a denial of service

How to Trigger

```
/* Plain cipher */
if (cipher_para) {
    ...
    } else if (alg_chain_para) { /* Algorithm chain */
    ...
    } else {
        return NULL;
    }
    max_len = (uint64_t)iv_len + aad_len + src_len + dst_len +
                hash_result_len;
    if (unlikely(max_len > vcrypto->conf.max_size)) {
        virtio_error(vdev, "virtio-crypto too big length");
        return NULL;
    }

op_info = g_malloc0(sizeof(CryptoDevBackendSymOpInfo) + max_len);
```

- Wrong encryption type
- Excessive length of the op_info

2. NPD in cryptodev_backend_account

- Root cause： no addition of the library for RSA when complie the QEMU
  - e.g: --enable-gcrypt

```
static int cryptodev_backend_account(CryptoDevBackend
*backend,
                 CryptoDevBackendOpInfo *op_info)
{...
    if (algtype == QCRYPTODEV_BACKEND_ALG_ASYM) {
        CryptoDevBackendAsymOpInfo *asym_op_info = op_info-
>u.asym_op_info;
        len = asym_op_info->src_len;
        switch (op_info->op_code) {
        case VIRTIO_CRYPTO_AKCIPHER_ENCRYPT:
            CryptodevAsymStatIncEncrypt(backend, len);
...
```

```
#define CryptodevSymStatInc(be, op, bytes)
do { \
   be->sym_stat->op##_bytes += (bytes); \
   be->sym_stat->op##_ops += 1; \
} while (/*CONSTCOND*/0)
```

may be NULL

Patch

- Add a check for the value of backend->asym_stat

```
--- a/backends/cryptodev.c
+++ b/backends/cryptodev.c
@@ -191,6 +191,11 @@  static int cryptodev_backend_account(CryptoDevBackend
*backend,
     if (algtype == QCRYPTODEV_BACKEND_ALG_ASYM) {
         CryptoDevBackendAsymOpInfo *asym_op_info = op_info->u.asym_op_info;
         len = asym_op_info->src_len;
+
+        if (unlikely(!backend->asym_stat)) {
+            error_report("cryptodev: Unexpected asym operation");
+            return -VIRTIO_CRYPTO_NOTSUPP;
+        }
         switch (op_info->op_code) {
         case VIRTIO_CRYPTO_AKCIPHER_ENCRYPT:
             CryptodevAsymStatIncEncrypt(backend, len);
```

3. NPD in cryptodev_builtin_operation

- Builtin backend : support AES/RSA encrypt/decrypt
- Both AES/RSA sessions are share the same structure(contain cipher&akcipher) and in the same array
- Only one structure(cipher&akcipher) in session can be initialized while the other is set as NULL
- Root cause : Incorrect matching between encryption/decryption algorithm and session

NPD in cryptodev_builtin_operation

```
static int cryptodev_builtin_operation(
                 CryptoDevBackend *backend,
                 CryptoDevBackendOpInfo *op_info)
{
...
    if (op_info->session_id >= MAX_NUM_SESSIONS ||
              builtin->sessions[op_info->session_id] == NULL) {
        error_setg(&local_error, "Cannot find a valid session id: %" PRIu64
"",
                   op_info->session_id);
        return -VIRTIO_CRYPTO_INVSESS;
    }
    sess = builtin->sessions[op_info->session_id];
    if (algtype == QCRYPTODEV_BACKEND_ALG_SYM) {
        sym_op_info = op_info->u.sym_op_info;
        status = cryptodev_builtin_sym_operation(sess, sym_op_info,
&local_error);
    } else if (algtype == QCRYPTODEV_BACKEND_ALG_ASYM) {
        asym_op_info = op_info->u.asym_op_info;
        status = cryptodev_builtin_asym_operation(sess, op_info->op_code,
asym_op_info, &local_error);
    }
...
```

no check for the type of session

sess->cipher or sess->akcipher may be NULL

4. CVE-2023-3180:Heap-based Buffer Overflow

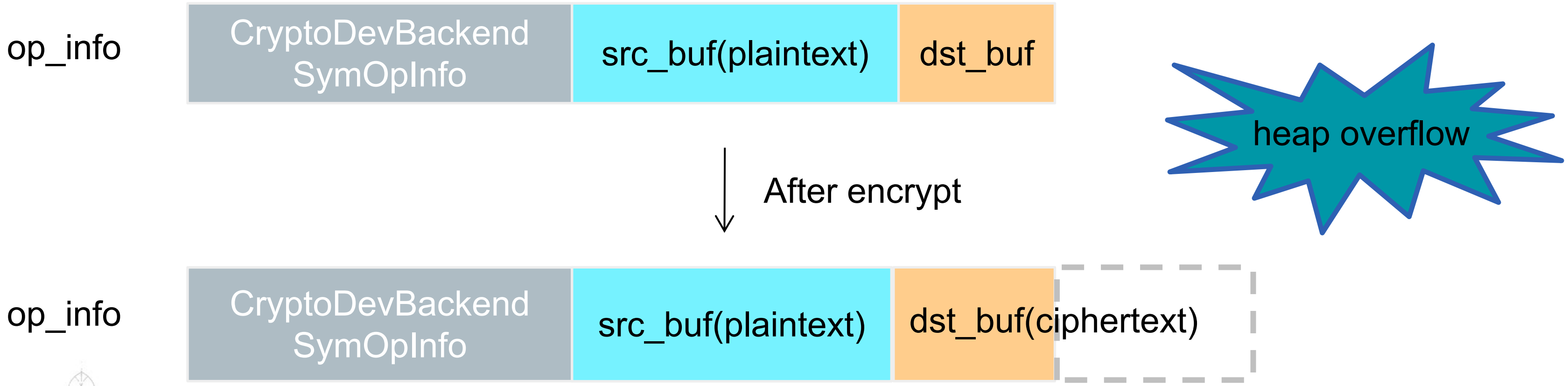
- No check for src_len and dst_len when do symmetric encryption/decryption

```
max_len = (uint64_t)iv_len + aad_len + src_len + dst_len + hash_result_len;
if (unlikely(max_len > vcrypto->conf.max_size)) {
    virtio_error(vdev, "virtio-crypto too big length");
    return NULL;
}

op_info = g_malloc0(sizeof(CryptoDevBackendSymOpInfo) + max_len);
op_info->iv_len = iv_len;
op_info->src_len = src_len;
op_info->dst_len = dst_len;
```

4. CVE-2023-3180:Heap-based Buffer Overflow

- Config
    - iv_len = 0, src_len = 0x80, dst_len = 0x40, hash_result_len = 0



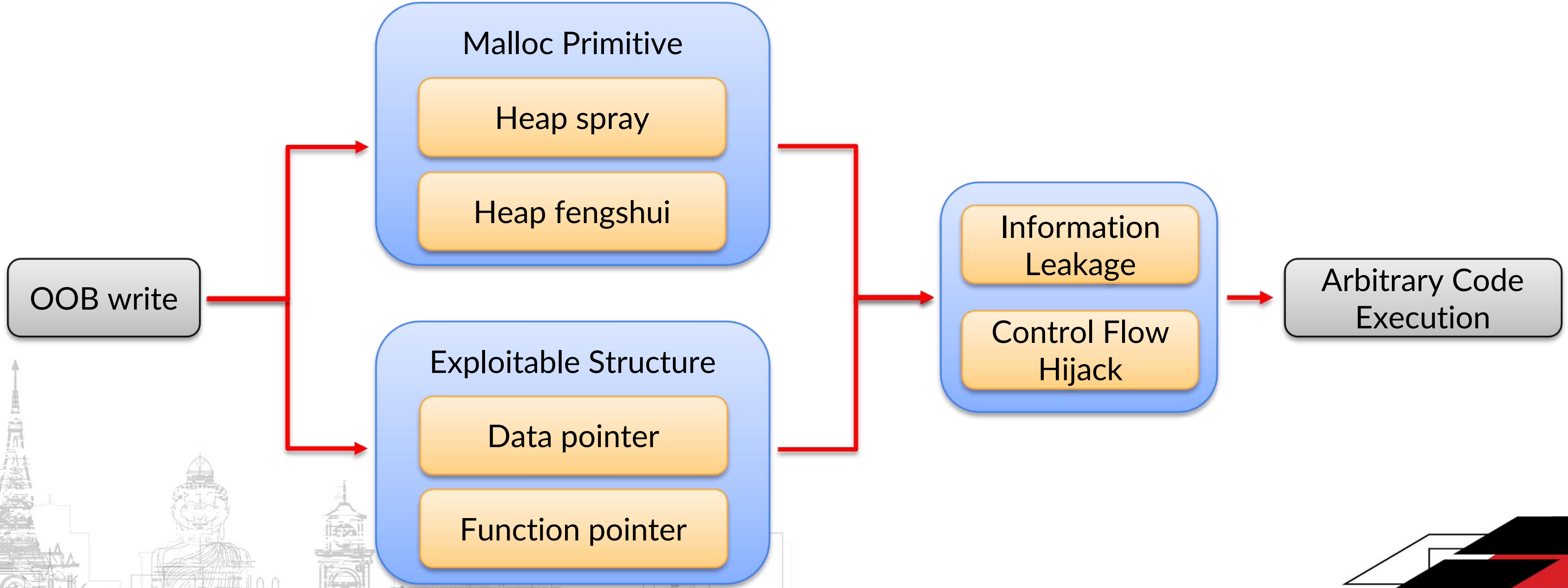
Patch

```
diff --git a/hw/virtio/virtio-crypto.c b/hw/virtio/virtio-crypto.c
index 44faf5a522..13aec771e1 100644
--- a/hw/virtio/virtio-crypto.c
+++ b/hw/virtio/virtio-crypto.c
@@ -634,6 +634,11 @@ virtio_crypto_sym_op_helper(VirtIODevice *vdev,
         return NULL;
     }

+    if (unlikely(src_len != dst_len)) {
+        virtio_error(vdev, "sym request src len is different from dst len");
+        return NULL;
+    }
+
     max_len = (uint64_t)iv_len + aad_len + src_len + dst_len + hash_result_len;
     if (unlikely(max_len > vcrypto->conf.max_size)) {
         virtio_error(vdev, "virtio-crypto too big length");
```

Exploit

Exploit development



Previous work

virtio-gpu: helps information leakage

- Leverage the uninitialized data in malloced chunk to leak

```
int vrend_renderer_resource_create(
                struct vrend_renderer_resource_create_args *args,
                struct iovec *iov, uint32_t num_iovs, void *image_oes)
{
    struct vrend_resource *gr;
    int ret;
...
    gr = (struct vrend_resource *)CALLOC_STRUCT(vrend_texture);
...
    if (args->bind == VIRGL_BIND_CUSTOM) {
        assert(args->target == PIPE_BUFFER);
        /* use iovec directly when attached */
        gr->storage = VREND_RESOURCE_STORAGE_GUEST_ELSE_SYSTEM;
        gr->ptr = malloc(args->width);
        if (!gr->ptr) {
            FREE(gr);
            return ENOMEM;
        }
    }
...
}
```

https://i.blackhat.com/asia-20/Thursday/asia-20-Shao-3D-Red-Pill-A-Guest-To-Host-Escape-On-QEMUKVM-Virtio-Device.pdf

Previous work

virtio-gpu: helps information leakage

- Leverage the uninitialized data in malloced chunk to leak
- New version code changes the malloc to calloc, so that this bug has been fixed already
- Not available any more

```
static int
vrend_resource_alloc_buffer(struct vrend_resource *gr,
                            uint32_t flags)
{
    const uint32_t bind = gr->base.bind;
    const uint32_t size = gr->base.width0;

    if (bind == VIRGL_BIND_CUSTOM) {
        /* use iovec directly when attached */
        gr->storage_bits |= VREND_STORAGE_HOST_SYSTEM_MEMORY;
        gr->ptr = calloc(1, size);
        if (!gr->ptr)
            return -ENOMEM;
    }
...
}
```

https://i.blackhat.com/asia-20/Thursday/asia-20-Shao-3D-Red-Pill-A-Guest-To-Host-Escape-On-QEMUKVM-Virtio-Device.pdf

Previous work

usb: convert oob read and write into AAR and AAW

- The oob read and write happens inside the USBDevice structure
- Nearly impossible to make heap manipulation
- Not suitable for us

```
/* definition of a USB device */
struct USBDevice {
    DeviceState qdev;
...
    uint8_t data_buf[4096];
    int32_t remote_wakeup;
    int32_t setup_state;
    int32_t setup_len;
    int32_t setup_index;

    USBEndpoint ep_ctl;
    USBEndpoint ep_in[USB_MAX_ENDPOINTS];
    USBEndpoint ep_out[USB_MAX_ENDPOINTS];

    QLIST_HEAD(, USBDescString) strings;
    const USBDesc *usb_desc;
...
}
```

https://conference.hitb.org/hitbsecconf2021ams/materials/D2T2%20-%20A%20Black%20Box%20Escape%20Of%20Qemu%20Based%20On%20The%20USB%20Device%20-%20L.%20Kong,%20Y.%20Zhang%20&%20H.%20Qu.pdf

Previous work

slirp: leverage IP fragment to AAR and AAW

- Partial overwrite m_data to get bypass ASLR
- Overwrite m_data and m_len to get AAW and AAR
- Not very friendly

```
struct mbuf {
    /* XXX should union some of these! */
    /* header at beginning of each mbuf: */
    struct mbuf *m_next; /* Linked list of mbufs */
    struct mbuf *m_prev;
    struct mbuf *m_nextpkt; /* Next packet in queue/record */
    struct mbuf *m_prevpkt; /* Flags aren't used in the output queue */
    int m_flags; /* Misc flags */

    int m_size; /* Size of mbuf, from m_dat or m_ext */
    struct socket *m_so;

    char *m_data; /* Current location of data */
    int m_len; /* Amount of data in this mbuf, from m_data */
...
};
```

https://github.com/0xKira/qemu-vm-escape/blob/master/Tensec2019-Vulnerability_Discovery_and_Exploitation_of_Virtualization_Solutions_for_Cloud_Computing_and_Desktops.pdf

Our solution -- virtio-crypto

Malloc primitives

- Guest simply make a symmetric encryption request
- Argument *_len are all controllable
- Malloc size vary from 0x60 to max_size depended by the configuration

```
static CryptoDevBackendSymOpInfo *
virtio_crypto_sym_op_helper(VirtIODevice *vdev,
           struct virtio_crypto_cipher_para *cipher_para,
           struct virtio_crypto_alg_chain_data_para *alg_chain_para,
           struct iovec *iov, unsigned int out_num)
{
...
    if (cipher_para) {
        iv_len = ldl_le_p(&cipher_para->iv_len);
        src_len = ldl_le_p(&cipher_para->src_data_len);
        dst_len = ldl_le_p(&cipher_para->dst_data_len);
    }
...
    max_len = (uint64_t)iv_len + aad_len + src_len + dst_len + hash_result_len;
    if (unlikely(max_len > vcrypto->conf.max_size)) {
        virtio_error(vdev, "virtio-crypto too big length");
        return NULL;
    }
    op_info = g_malloc0(sizeof(CryptoDevBackendSymOpInfo) + max_len);
...
}
```

Our solution -- virtio-crypto

Malloc primitives

- Guest simply make a asymmetric encryption request
- Argument src_len and dst_len are all controllable with no size limitation
- Malloc size could be truly arbitrary

```
static int
virtio_crypto_handle_asym_req(VirtIOCrypto *vcrypto,
        struct virtio_crypto_akcipher_data_req *req,
        CryptoDevBackendOpInfo *op_info,
        struct iovec *iov, unsigned int out_num)
{
...
    asym_op_info = g_new0(CryptoDevBackendAsymOpInfo, 1);
    src_len = ldl_le_p(&req->para.src_data_len);
    dst_len = ldl_le_p(&req->para.dst_data_len);

   if (src_len > 0) {
        src = g_malloc0(src_len);
...
    }

   if (dst_len > 0) {
        dst = g_malloc0(dst_len);
...
    }
...
}
```

Our solution -- virtio-crypto

Exploitable structures

- When making an encryption request, these structures will be allocated
- Overwrite the member src_len, we could make further oob read
- Overwrite the member dst, we could make arbitrary write

```
typedef struct CryptoDevBackendAsymOpInfo
{
        uint32_t src_len;
        uint32_t dst_len;
        uint8_t *src;
        uint8_t *dst;
} CryptoDevBackendAsymOpInfo;
```

```
typedef struct CryptoDevBackendSymOpInfo
{
        uint32_t aad_len;
        uint32sr_t iv_len;
        uint32_t c_len;
        uint32_t dst_len;
        uint32_t digest_result_len;
        uint32_t hash_start_src_offset;
        uint32_t cipher_start_src_offset;
        uint32_t len_to_hash;
        uint32_t len_to_cipher;
        uint8_t op_type;
        uint8_t *iv;
        uint8_t *src;
        uint8_t *dst;
        uint8_t *aad_data;
        uint8_t *digest_result;
        uint8_t data[];
} CryptoDevBackendSymOpInfo;
```

Our solution -- virtio-crypto

Exploitable structures

```
typedef struct VirtIOCryptoReq {
    VirtQueueElement elem;
    /* flags of operation, such as type of algorithm */
    uint32_t flags;
    struct virtio_crypto_inhdr *in;
    struct iovec *in_iov; /* Head address of dest iovec */
    unsigned int in_num; /* Number of dest iovec */
    size_t in_len;
    VirtQueue *vq;
    struct VirtIOCrypto *vcrypto;
    CryptoDevBackendOpInfo op_info;
} VirtIOCryptoReq;

typedef struct CryptoDevBackendOpInfo {
    QCryptodevBackendAlgType algtype;
    uint32_t op_code;
    uint32_t queue_index;
    CryptoDevCompletionFunc cb;
    void *opaque; /* argument for cb */
...
} CryptoDevBackendOpInfo;
```

- When Making an encryption request, the structure will be allocated
- Member in helps leak the guest memory space address. And member cb and opaque help leak the qemu image and heap address.
- Overwrite the member cb and opaque to hijack control flow and overwrite the member in_iov to make AAW.

However...

Every encryption / decryption process is synchronous by default

- There will be only one instance of each exploitable structure residing in memory.
- The vulnerable sym_op_info object could not overflow any other useful structures inside virtio-crypto.
- All these structures mentioned before will be freed after the process, which means we could only prepare a chunk hole ahead to make oob write.
- We could not get any time window of the malloc-use-free process and therefore we could not make heap spray and manipulation.

However...

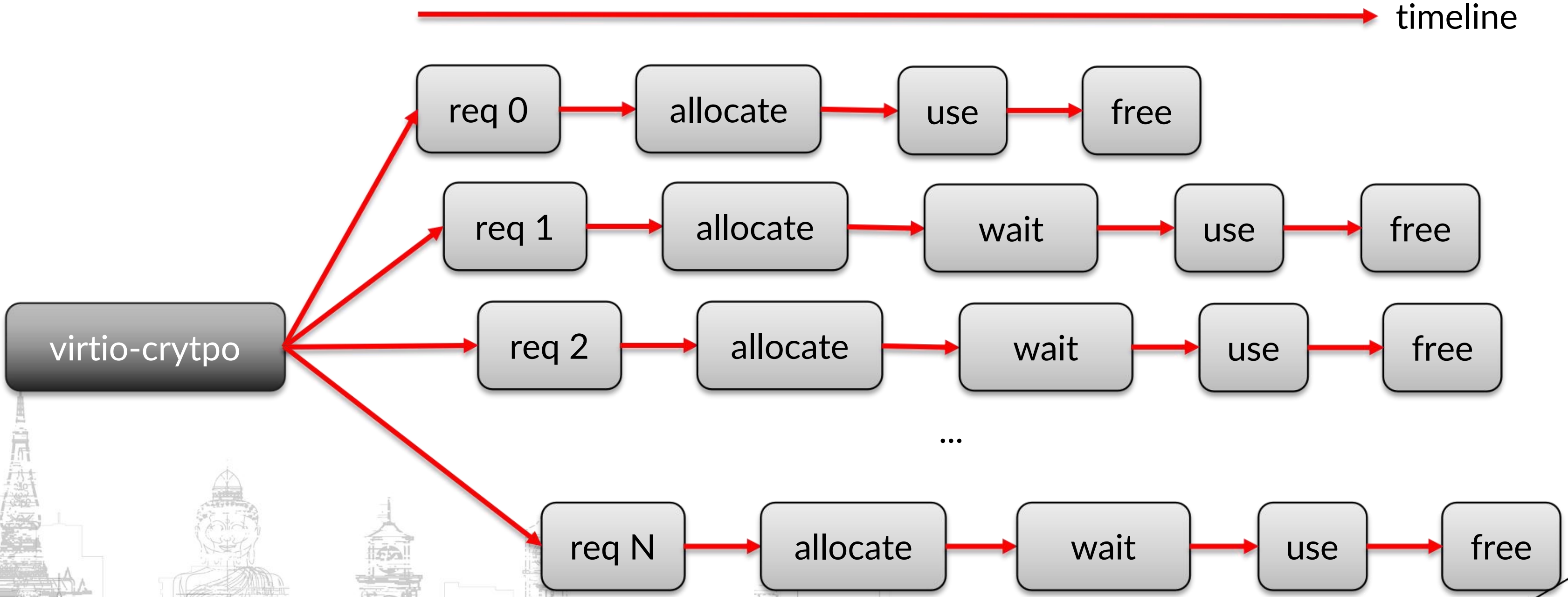
Every encryption / decryption process is synchronous by default

- There will be only one instance of each exploitable ture residing in memory.
- The vulnerable sym_op_info object any other useful structures inside virtio-crypto
- All these structures mer will be freed after the process, which means we could o are a chunk hole ahead to make oob write.
- We could not get any time window of the malloc-use-free process and therefore we could not make heap spray and manipulation.

Solution: asynchronous clock

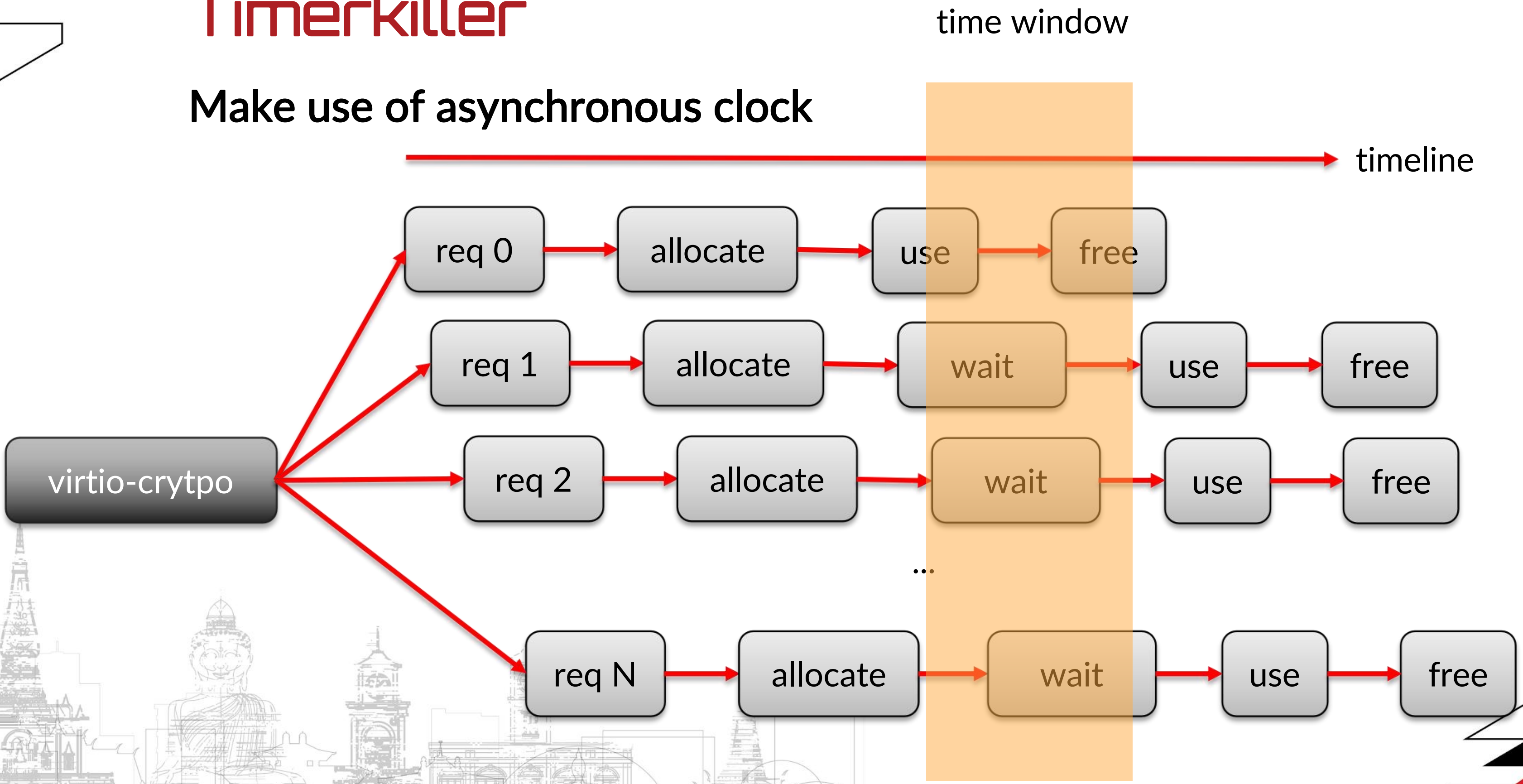
Timerkiller

Make use of asynchronous clock



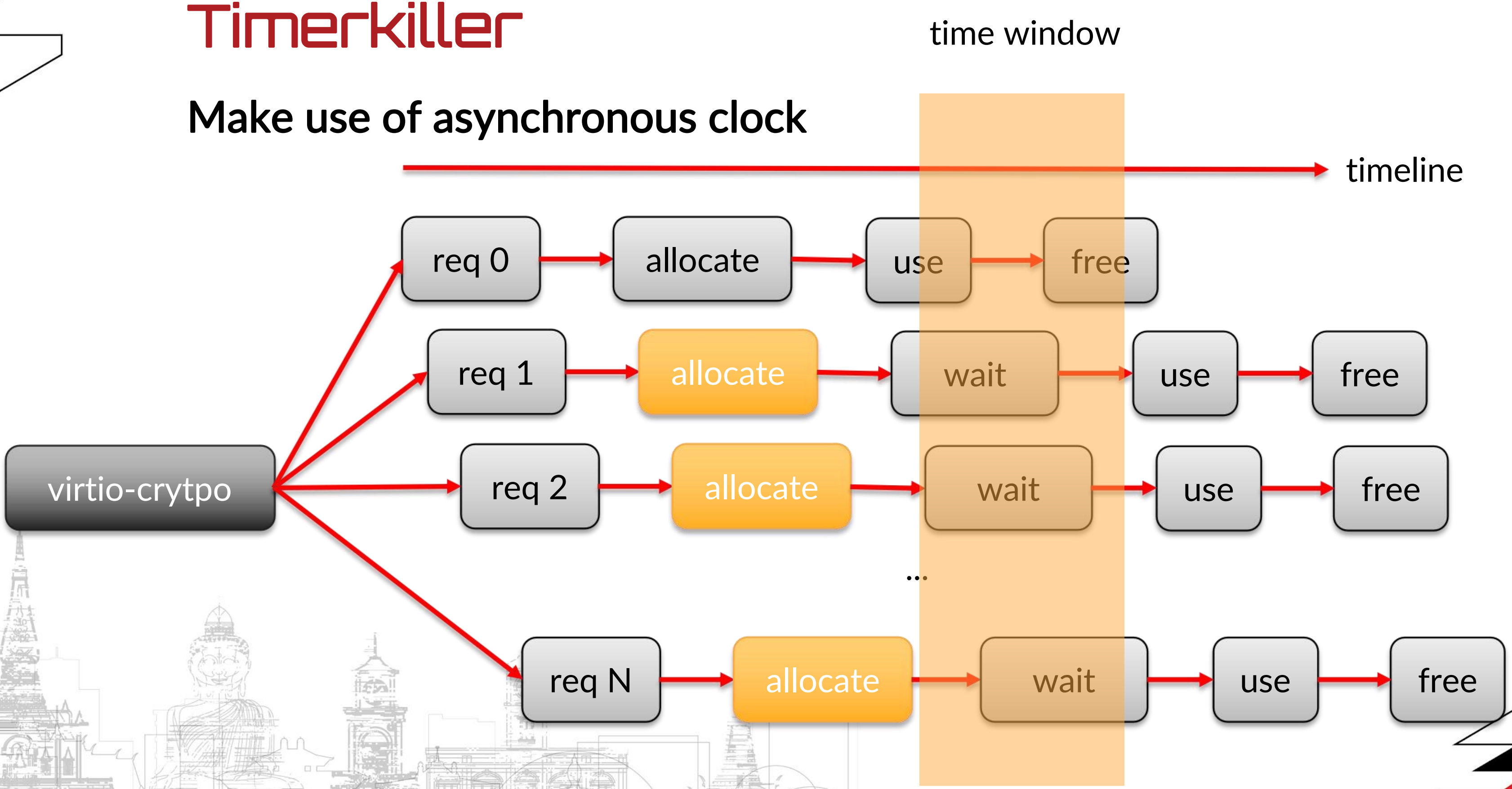
Timerkiller

Make use of asynchronous clock



Timerkiller

Make use of asynchronous clock



Timerkiller

Make use of asynchronous clock

- Multiple requests, sym_op_info and asym_op_info could stay in heap memory at the same time
- The size of time windows could be controlled by making a encryption request that the data is of certain size
- It’s very easy to do so, since all we need to do is to prepare the arguments and make a request

…	…
request	…
…	request
sym_op_info	…
…	asym_op_info
…	…
asym_op_info	sym_op_info
…	asym_op_info
sym_op_info	…
sym_op_info	request
…	…

qemu process heap layout

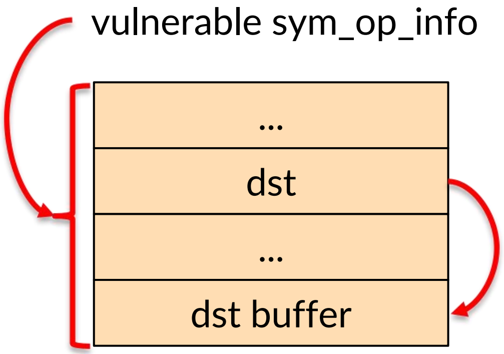
Information leakage -- our plan

How to turn oob write into an oob read ?

Information leakage -- our plan

How to turn oob write into an oob read ?

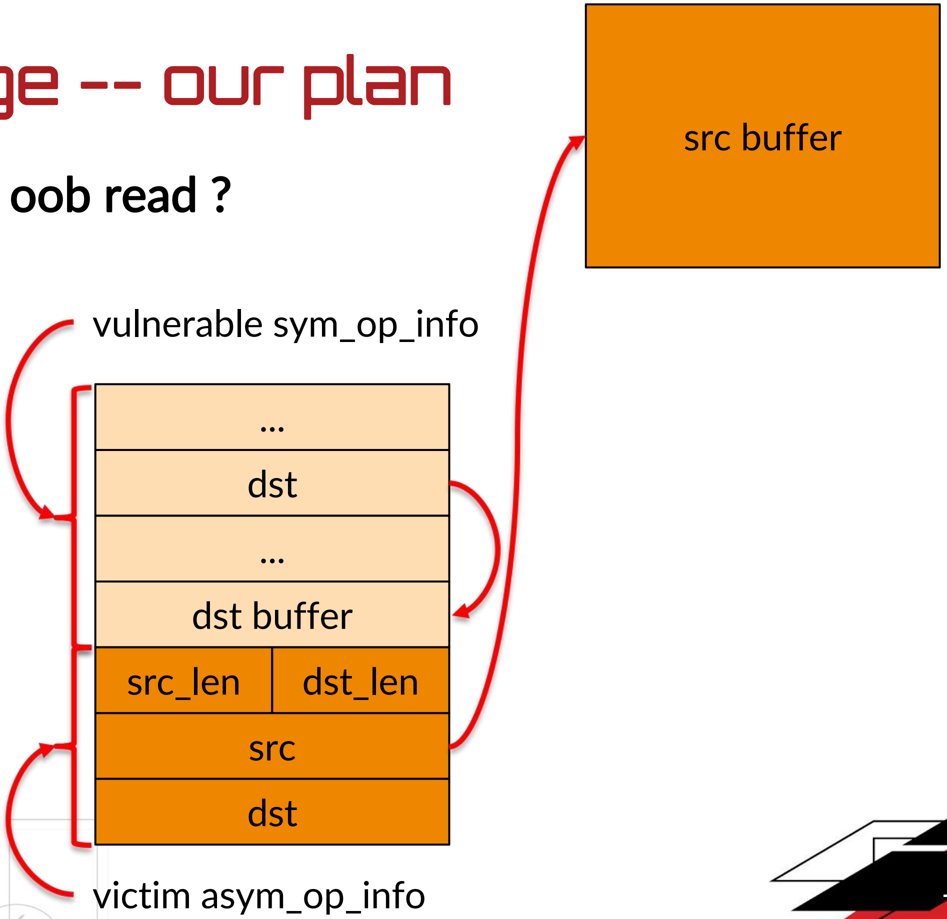
- Prepare a vulnerable sym_op_info



Information leakage -- our plan

How to turn oob write into an oob read ?

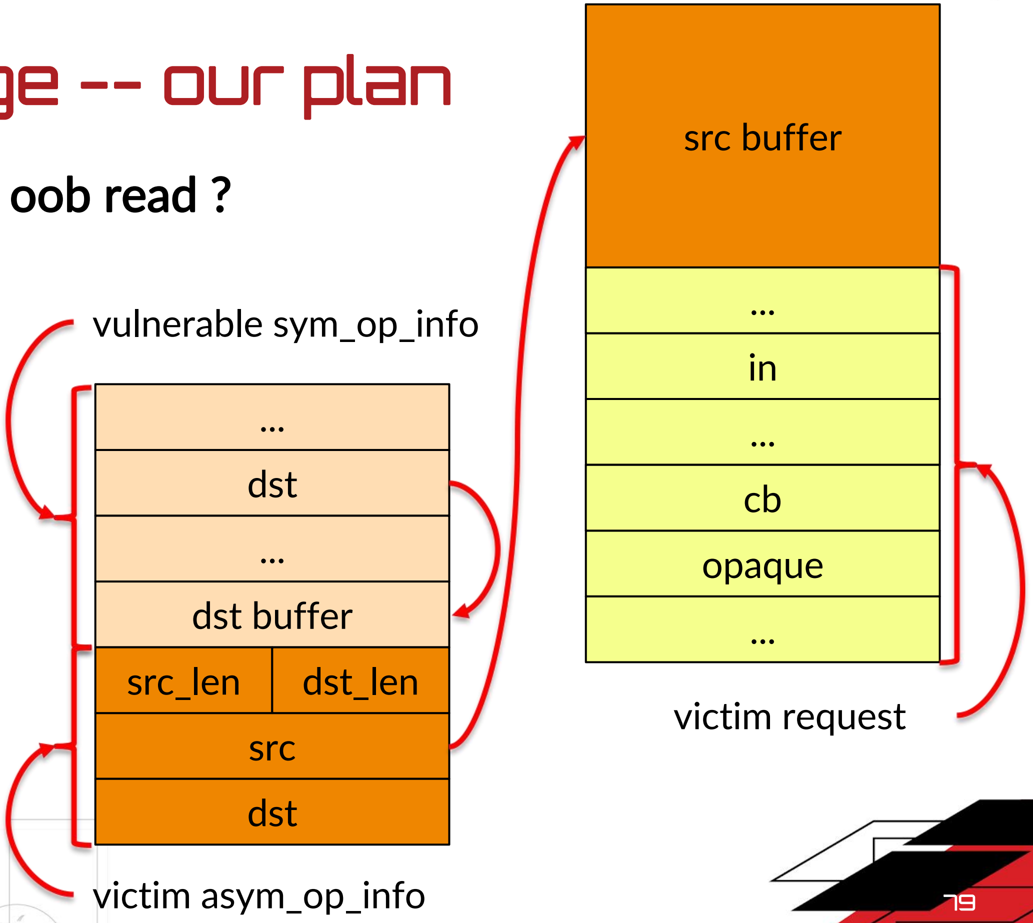
- Prepare a vulnerable sym_op_info
- Put an asym_op_info next to the vulnerable sym_op_info



Information leakage -- our plan

How to turn oob write into an oob read ?

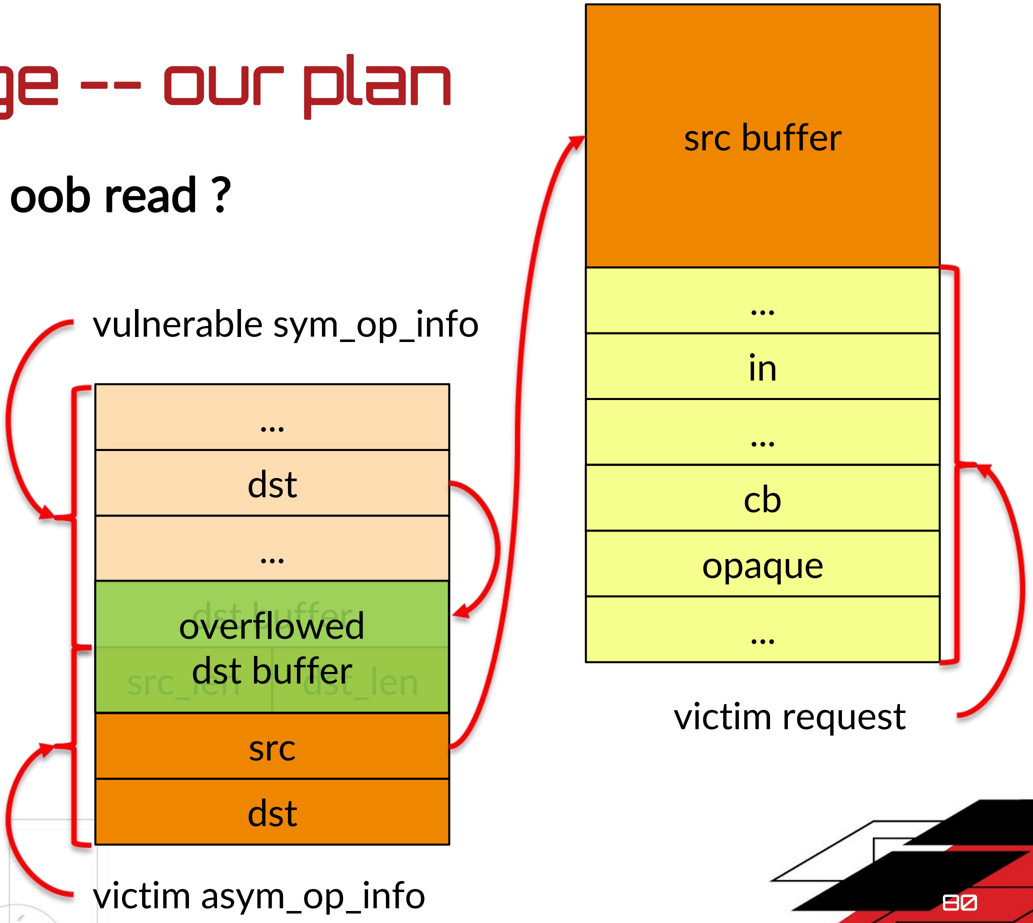
- Prepare a vulnerable sym_op_info
- Put an asym_op_info next to the vulnerable sym_op_info
- Put a request next to the asym_op_info->src



Information leakage -- our plan

How to turn oob write into an oob read ?

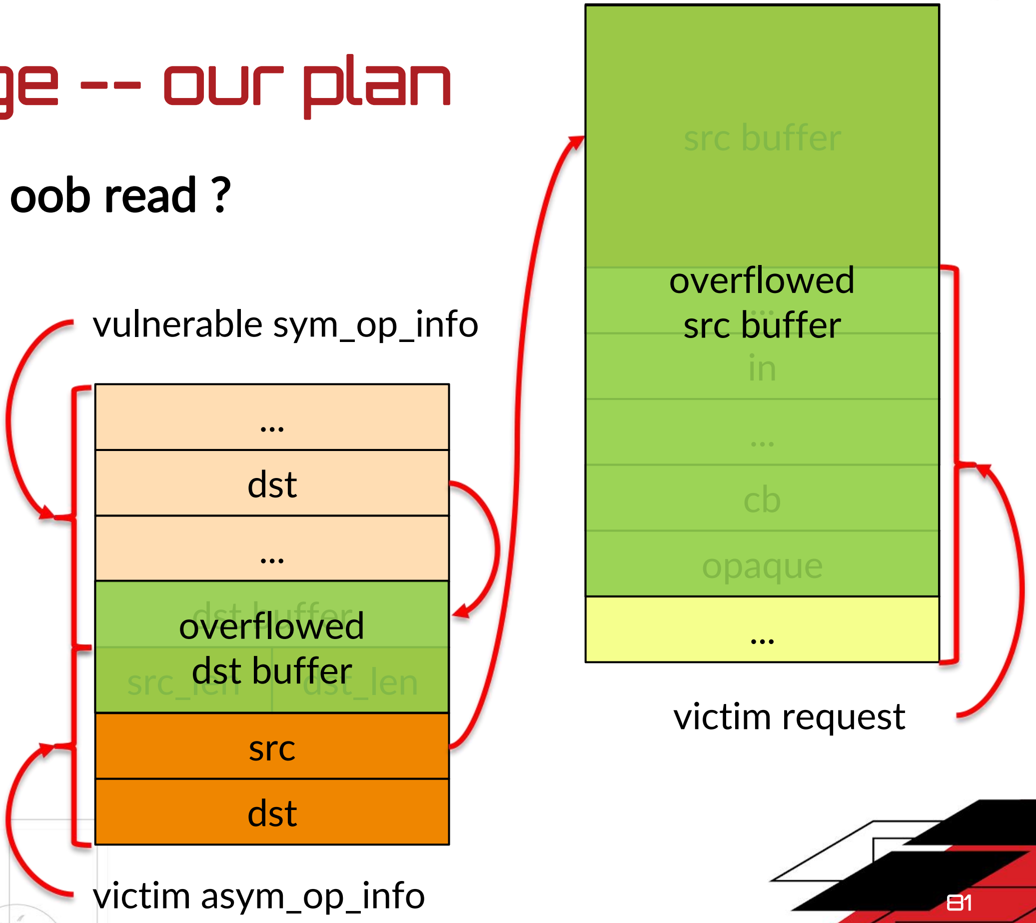
- Prepare a vulnerable sym_op_info
- Put an asym_op_info next to the vulnerable sym_op_info
- Put a request next to the asym_op_info->src
- Oob write asym_op_info->src_len



Information leakage -- our plan

How to turn oob write into an oob read ?

- Prepare a vulnerable sym_op_info
- Put an asym_op_info next to the vulnerable sym_op_info
- Put a request next to the asym_op_info->src
- Oob write asym_op_info->src_len
- Get oob read when execute asymmetric encryption



```
gef➤  p *request
$1 = {
  elem = {
    index = 0x0,
    len = 0x7f87,
    ndescs = 0x1,
    out_num = 0x3,
    in_num = 0x26,
    in_addr = 0x561b35123eb0,
    out_addr = 0x561b35123fe0,
    in_sg = 0x561b35123ff8,
    out_sg = 0x561b35124258
  },
  flags = 0x0,
  in = 0x7f86f9fa1040,
  in_iov = 0x561b34c29990,
  in_num = 0x26,
  in_len = 0x9a6,
  vq = 0x7f875b279010,
  vcrypto = 0x561b35ab7e80,
  op_info = {
    algtype = QCRYPTODEV_BACKEND_ALG_SYM,
    op_code = 0x0,
    queue_index = 0x0,
    cb = 0x561b31ed6d80 <virtio_crypto_req_complete>,
    opaque = 0x561b35123e00,
    session_id = 0x5,
    u = {
      sym_op_info = 0x561b35be9000,
      asym_op_info = 0x561b35be9000
    },
    next = {
      tqe_next = 0x0,
      tqe_circ = {
        tql_next = 0x0,
        tql_prev = 0x0
      }
    }
  }
}
```

```
gef➤  xinfo request->in
Page: 0x00007f86d3e00000  →  0x00007f8753e00000 (size=0x80000000)
Permissions: rw-
Pathname:
Offset (from page): 0x261a1040
Inode: 0
```

oob read ?

```
gef➤  xinfo request->op_info.cb
Page: 0x0000561b31c29000  →  0x0000561b32486000 (size=0x85d000)
Permissions: r-x
Pathname: /usr/local/bin/qemu-system-x86_64
Offset (from page): 0x2add80
Inode: 5641119
Segment: .text (0x0000561b31c32190-0x0000561b32485a50)
Offset (from segment): 0x2a4bf0
Symbol: virtio_crypto_req_complete
```

```
gef➤  xinfo request->op_info.opaque
Page: 0x0000561b34b2a000  →  0x0000561b35e31000 (size=0x1307000)
Permissions: rw-
Pathname: [heap]
Offset (from page): 0x5f9e00
Inode: 0
```

vulnerable sym_op_info

overflowed src buffer

...

opaque

overflowed

dst

request

victim asym_op_info

Information leakage -- main steps

1. Make a encryption request and occupy the vrtio-crypto device for a certain time
2. Prepare a chunk with size N
3. Heap spray and clear small bins with size 0x20, Y and N-0x20 to N
4. Free the chunk with size N mentioned above
5. Allocate the vulnerable sym_op_info with size N-0x20, and leave a small bin with size 0x20
6. Allocate the victim asym_op_info with the 0x20 small bin, and allocate the asym_op_info->src with size Y so that it will allocate from unsorted bin
7. Allocate the victim request from large bin and thus it's adjacent to the victim asym_op_info->src
8. Overwrite the asym_op_info->src_len and request will then be leaked

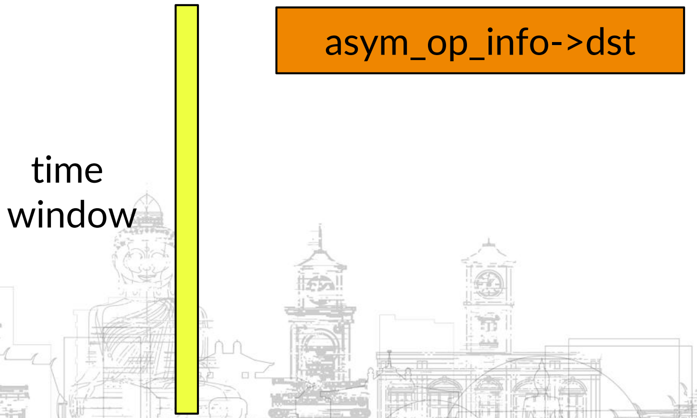
Information leakage -- details

1. Make a encryption request and occupy the vrtio-crypto device for a certain time

time
window

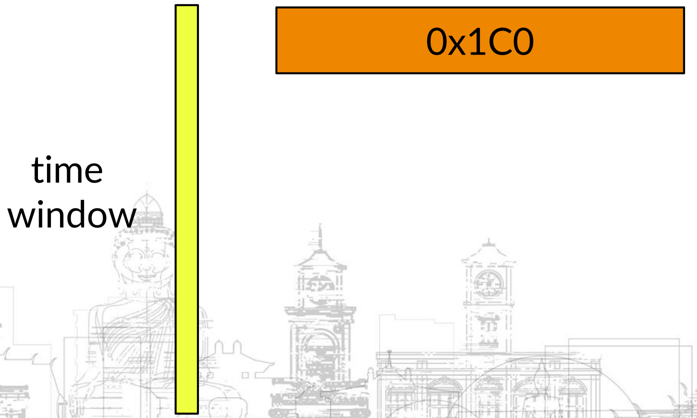
Information leakage -- details

1. Make a encryption request and occupy the vrtio-crypto device for a certain time
2. Prepare a chunk with size 0x1C0



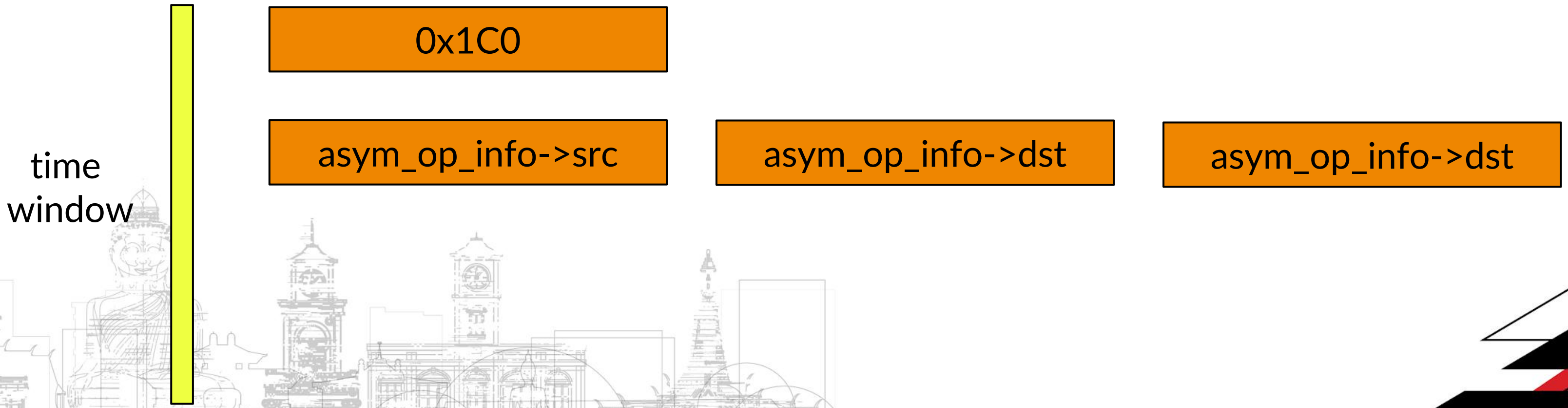
Information leakage -- details

1. Make a encryption request and occupy the vrtio-crypto device for a certain time
2. Prepare a chunk with size 0x1C0



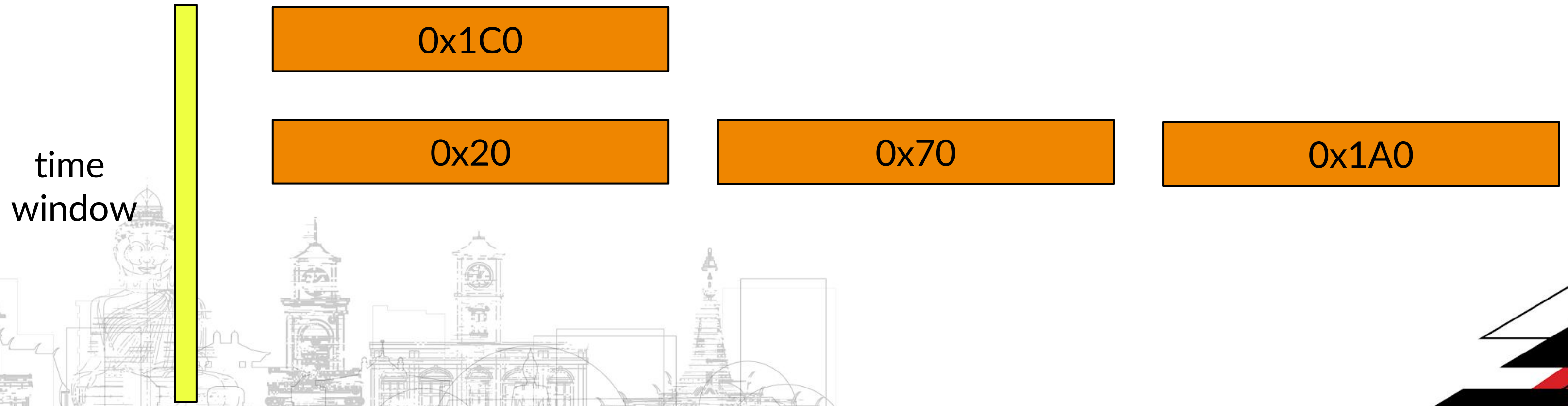
Information leakage -- details

1. Make a encryption request and occupy the vrtio-crypto device for a certain time
2. Prepare a chunk with size 0x1C0
3. Heap spray and clear small bins with size 0x20, 0x70, 0x1A0 to 0x1C0



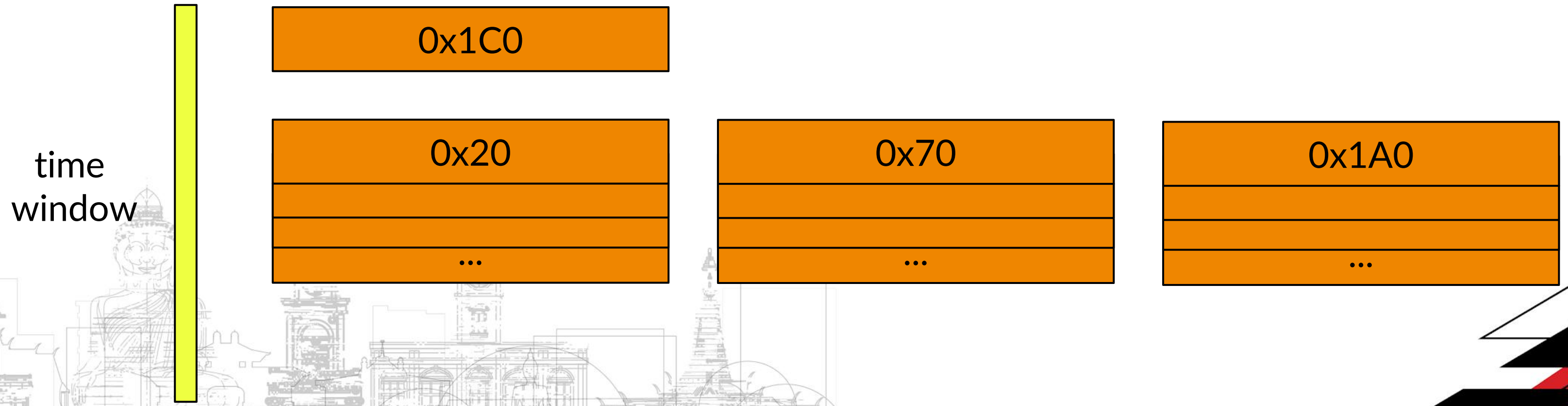
Information leakage -- details

1. Make a encryption request and occupy the vrtio-crypto device for a certain time
2. Prepare a chunk with size 0x1C0
3. Heap spray and clear small bins with size 0x20, 0x70, 0x1A0 to 0x1C0



Information leakage -- details

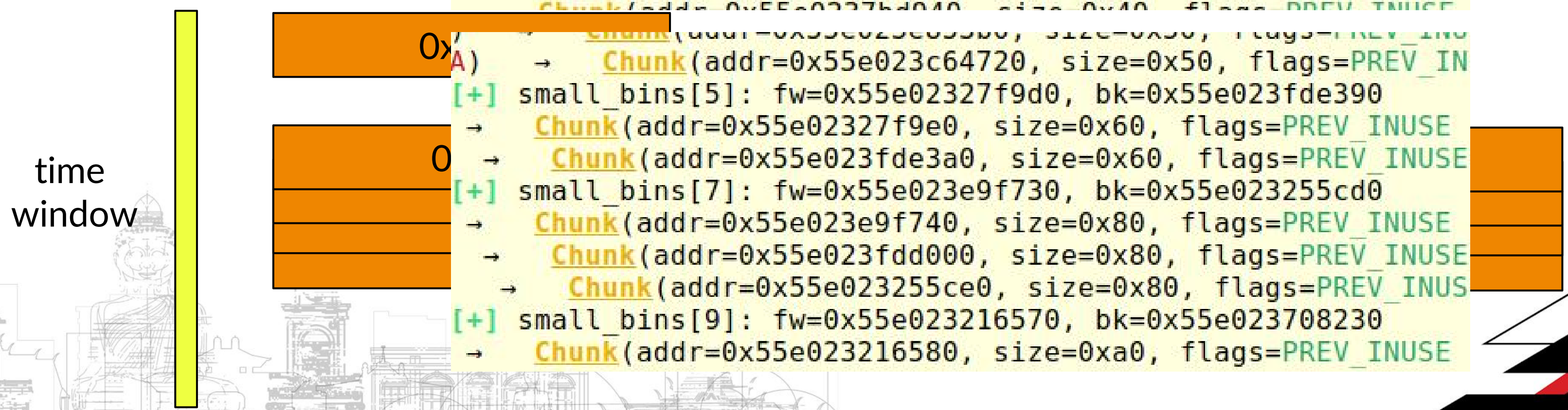
1. Make a encryption request and occupy the vrtio-crypto device for a certain time
2. Prepare a chunk with size 0x1C0
3. Heap spray and clear small bins with size 0x20, 0x70, 0x1A0 to 0x1C0



Information leakage -- details

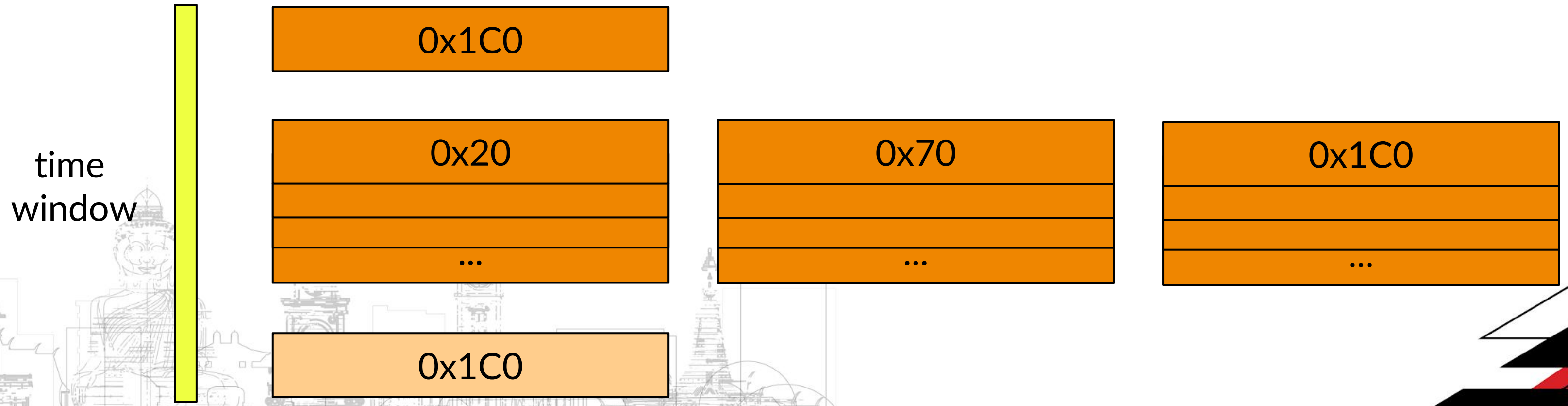
1. Make a encrypt
   time
2. Prepare a chunk
3. Heap spray and

```
gef➤  heap bin small
[+] small_bins[2]: fw=0x55e023fdd9c0, bk=0x55e023eacb90
 →   Chunk(addr=0x55e023fdd9d0, size=0x30, flags=PREV_INUSE
  →   Chunk(addr=0x55e023d4efd0, size=0x30, flags=PREV_INUSE
   →   Chunk(addr=0x55e023feebb0, size=0x30, flags=PREV_INUS
)    →   Chunk(addr=0x55e023ff63d0, size=0x30, flags=PREV_INU
A)    →   Chunk(addr=0x55e0236890c0, size=0x30, flags=PREV_IN
NA)    →   Chunk(addr=0x55e023eacba0, size=0x30, flags=PREV_I
[+] small_bins[3]: fw=0x55e0237bd930, bk=0x55e023763080
A)    →   Chunk(addr=0x55e023c64720, size=0x50, flags=PREV_IN
[+] small_bins[5]: fw=0x55e02327f9d0, bk=0x55e023fde390
 →   Chunk(addr=0x55e02327f9e0, size=0x60, flags=PREV_INUSE
  →   Chunk(addr=0x55e023fde3a0, size=0x60, flags=PREV_INUSE
[+] small_bins[7]: fw=0x55e023e9f730, bk=0x55e023255cd0
 →   Chunk(addr=0x55e023e9f740, size=0x80, flags=PREV_INUSE
  →   Chunk(addr=0x55e023fdd000, size=0x80, flags=PREV_INUSE
   →   Chunk(addr=0x55e023255ce0, size=0x80, flags=PREV_INUS
[+] small_bins[9]: fw=0x55e023216570, bk=0x55e023708230
 →   Chunk(addr=0x55e023216580, size=0xa0, flags=PREV_INUSE
```



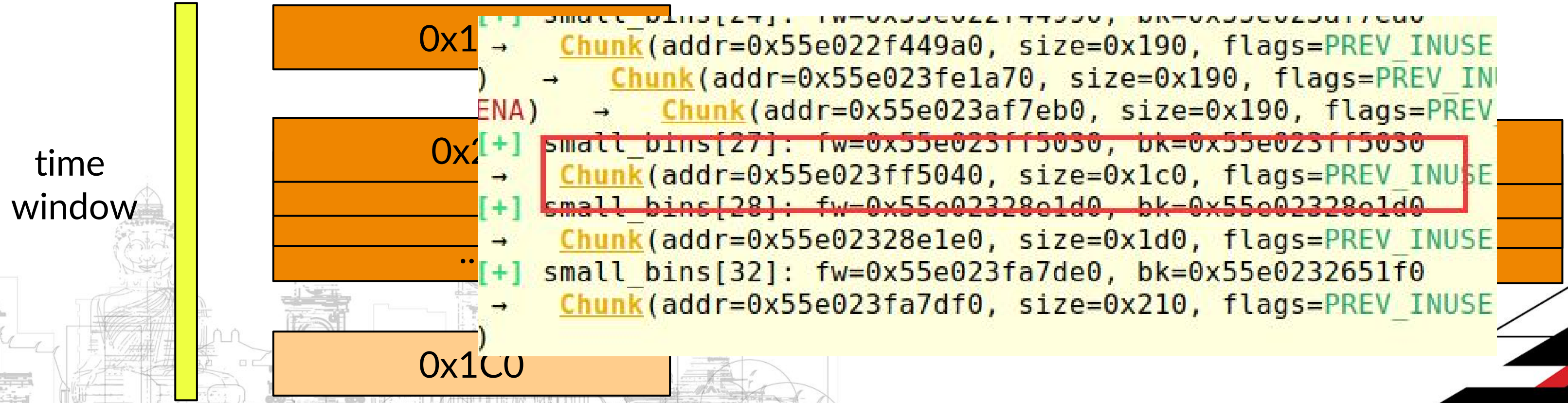
Information leakage -- details

1. Make a encryption request and occupy the vrtio-crypto device for a certain time
2. Prepare a chunk with size 0x1C0
3. Heap spray and clear small bins with size 0x20, 0x70, 0x1A0 to 0x1C0
4. Free the chunk with size 0x1C0 mentioned above



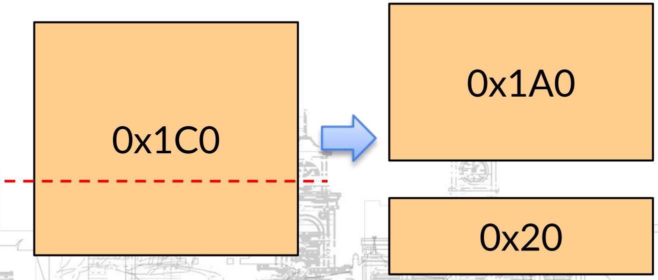
Information leakage -- details

1. Make a encryption request and occupy the vrtio-crypto device for a certain time
2. Prepare a chunk with size 0x1C0
3. Heap spray and clear small bins with size 0x20, 0x70, 0x1A0 to 0x1C0
4. Free the chunk with size 0x1C0 mentioned above



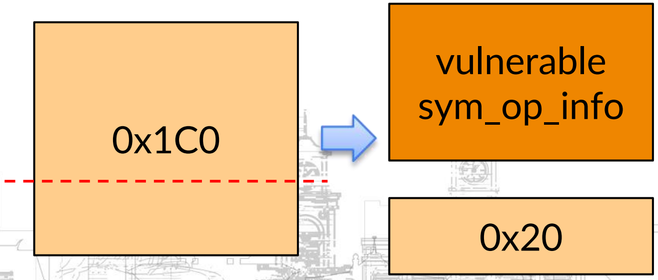
Information leakage -- details

5. Allocate the vulnerable sym_op_info with size 0x1A0, and leave a small bin with size 0x20



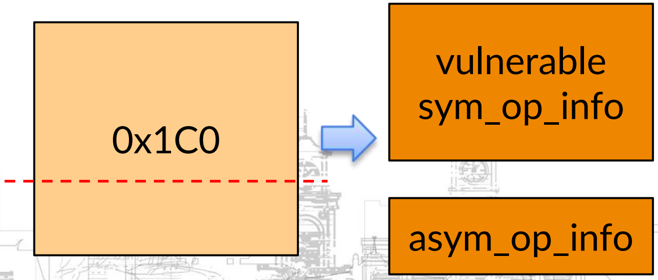
Information leakage -- details

5. Allocate the vulnerable sym_op_info with size 0x1A0, and leave a small bin with size 0x20



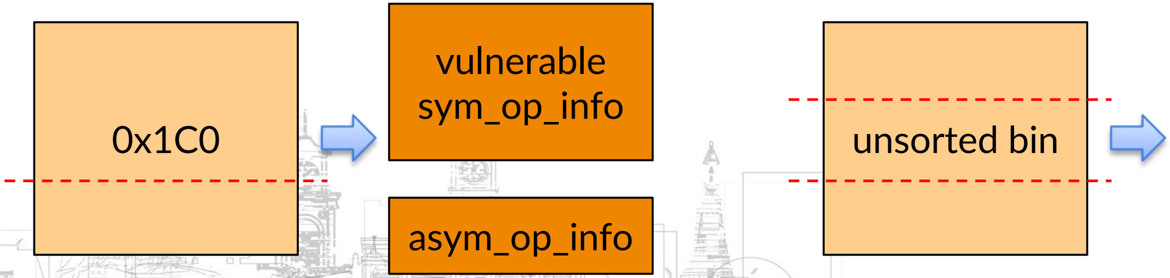
Information leakage -- details

5. Allocate the vulnerable sym_op_info with size 0x1A0, and leave a small bin with size 0x20
6. Allocate the victim asym_op_info with the 0x20 small bin, and allocate the asym_op_info->src with size 0x70 so that it will allocate from unsorted bin



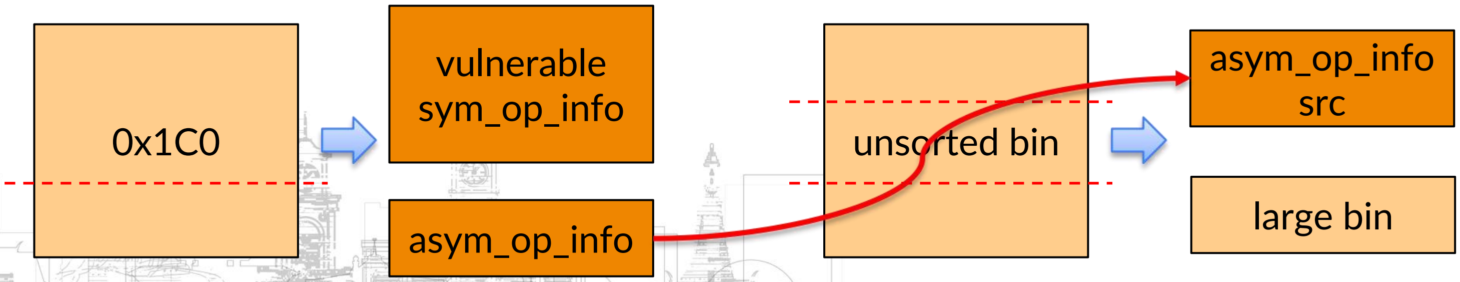
Information leakage -- details

5. Allocate the vulnerable sym_op_info with size 0x1A0, and leave a small bin with size 0x20
6. Allocate the victim asym_op_info with the 0x20 small bin, and allocate the asym_op_info->src with size 0x70 so that it will allocate from unsorted bin



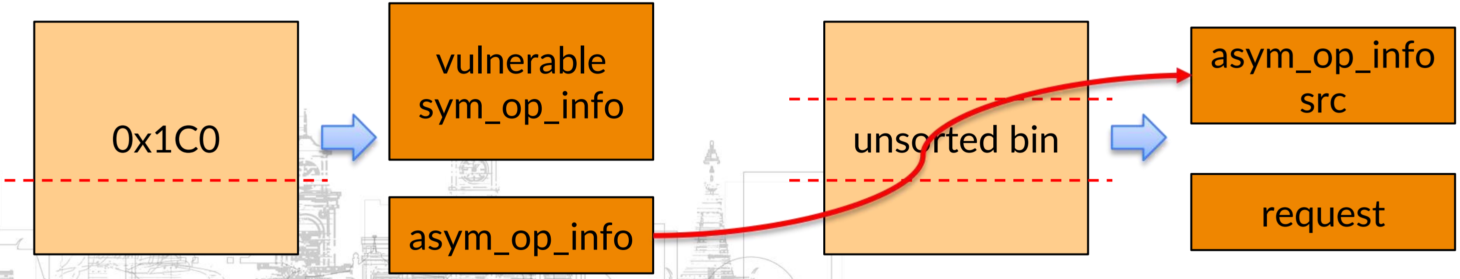
Information leakage -- details

5. Allocate the vulnerable sym_op_info with size 0x1A0, and leave a small bin with size 0x20
6. Allocate the victim asym_op_info with the 0x20 small bin, and allocate the asym_op_info->src with size 0x70 so that it will allocate from unsorted bin



Information leakage -- details

5. Allocate the vulnerable sym_op_info with size 0x1A0, and leave a small bin with size 0x20
6. Allocate the victim asym_op_info with the 0x20 small bin, and allocate the asym_op_info->src with size 0x70 so that it will allocate from unsorted bin
7. Allocate the victim request from large bin and thus it's adjacent to the victim asym_op_info->src



Information leakage

Get the leaked information

Then we just wait for the ciphertext of the oob data transferred to the guest.

And later decrypt it to get the address information to bypass aslr.

```
0x0000000000000491
0x00007f1100000000
0x0000000300000001
0x0000561600000026
0x00005616e3a00940
0x00005616e3a00a70
0x00005616e3a00a88
0x00005616e3a00ce8
0000000000000000000
0x00007f1115147008
0x00005616e3703d90
0x0000000000000026
0x0000000000000156
0x00007f1178c42010
0x00005616e45fa960
0000000000000000000
0000000000000000000
0x00005616e1a3ad80
0x00005616e3a00890
------------------------------------------------------
[+] guest_memory_base: 0x7f10ebe00000
[+] qemu_base: 0x5616e147f000
[+] system_plt: 0x5616e1793c74
------------------------------------------------------
```

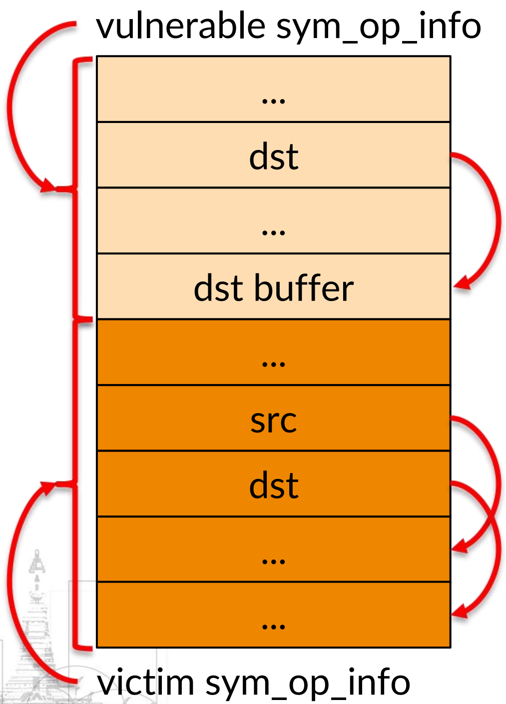
Control flow hijack -- our plan

Method 1: oob write sym_op_info to make AAW

Control flow hijack -- our plan

Method 1: oob write sym_op_info to make AAW

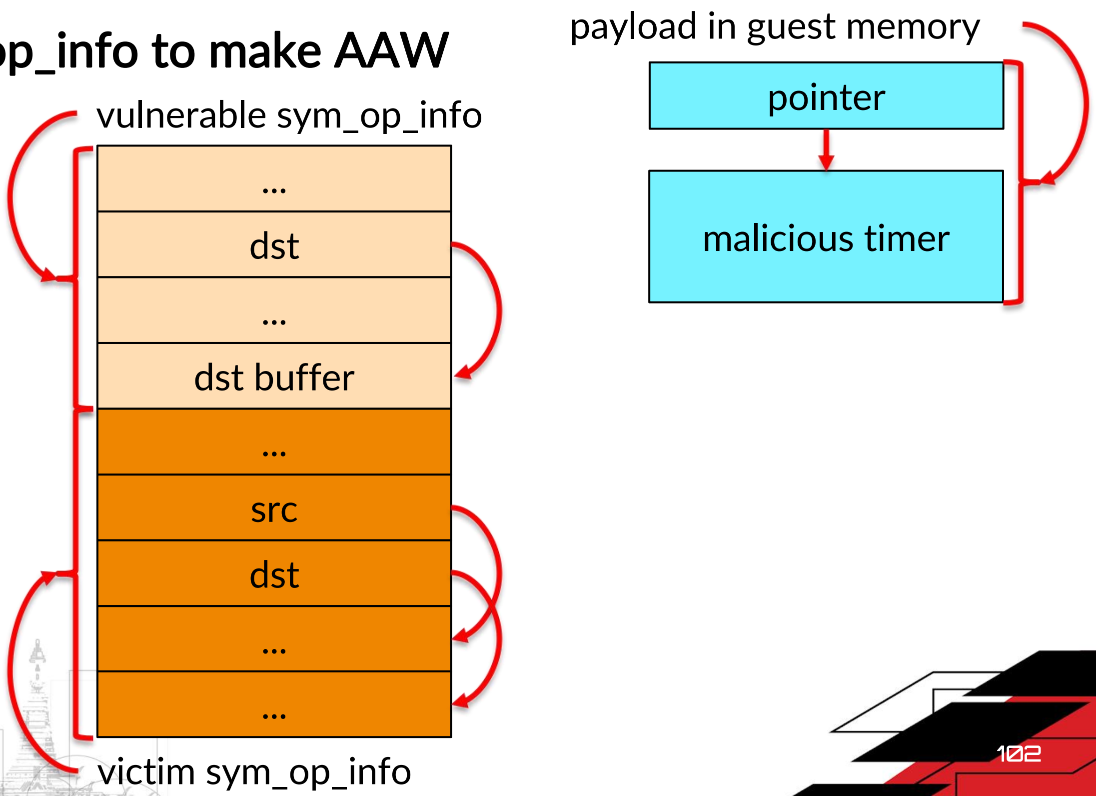
- Prepare a vulnerable sym_op_info
- Put another victim sym_op_info next to the vulnerable sym_op_info



Control flow hijack -- our plan

Method 1: oob write sym_op_info to make AAW

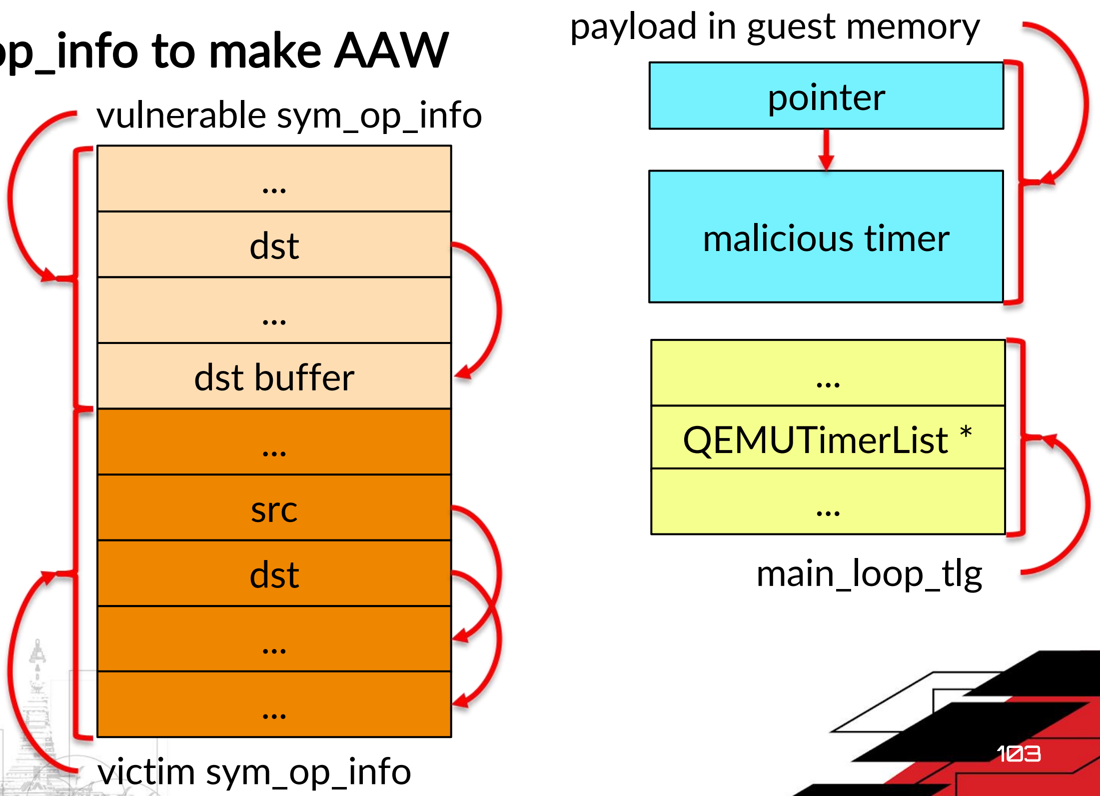
- Prepare a vulnerable sym_op_info
- Put another victim sym_op_info next to the vulnerable sym_op_info
- Prepare payload in guest memory space



Control flow hijack -- our plan

Method 1: oob write sym_op_info to make AAW

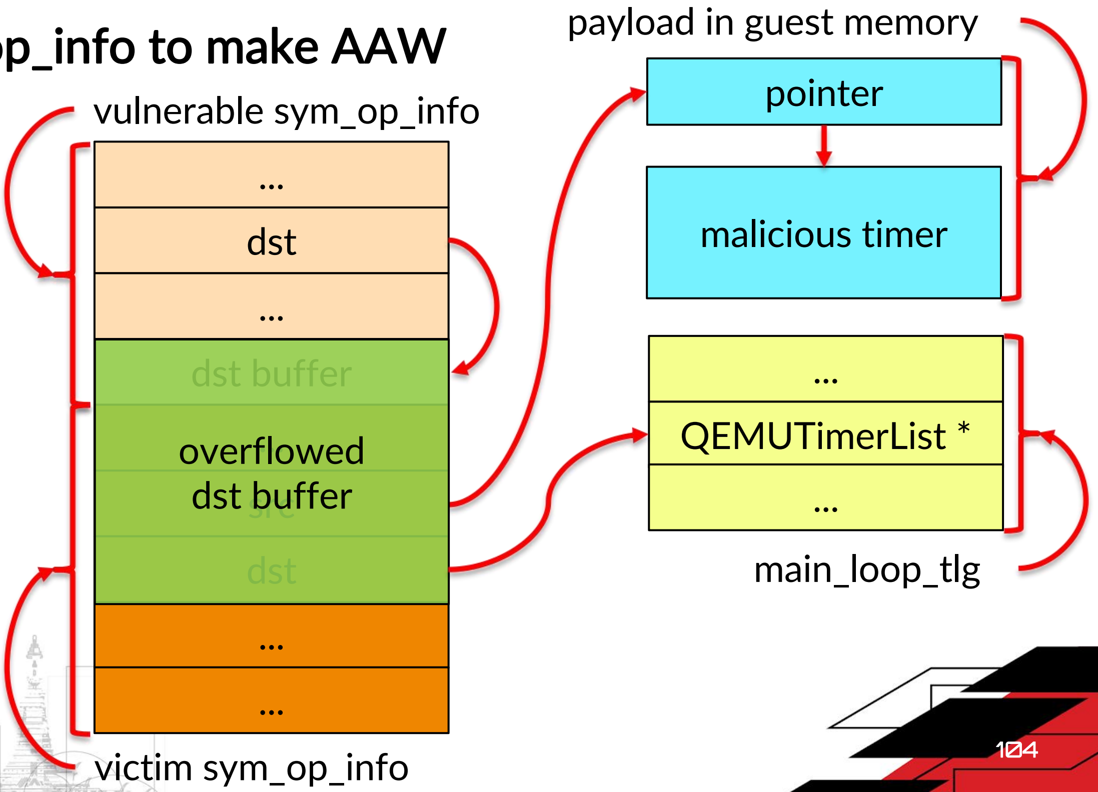
- Prepare a vulnerable sym_op_info
- Put another victim sym_op_info next to the vulnerable sym_op_info
- Prepare payload in guest memory space
- Overwrite the victim sym_op_info->src and victim sym_op_info->dst



Control flow hijack -- our plan

Method 1: oob write sym_op_info to make AAW

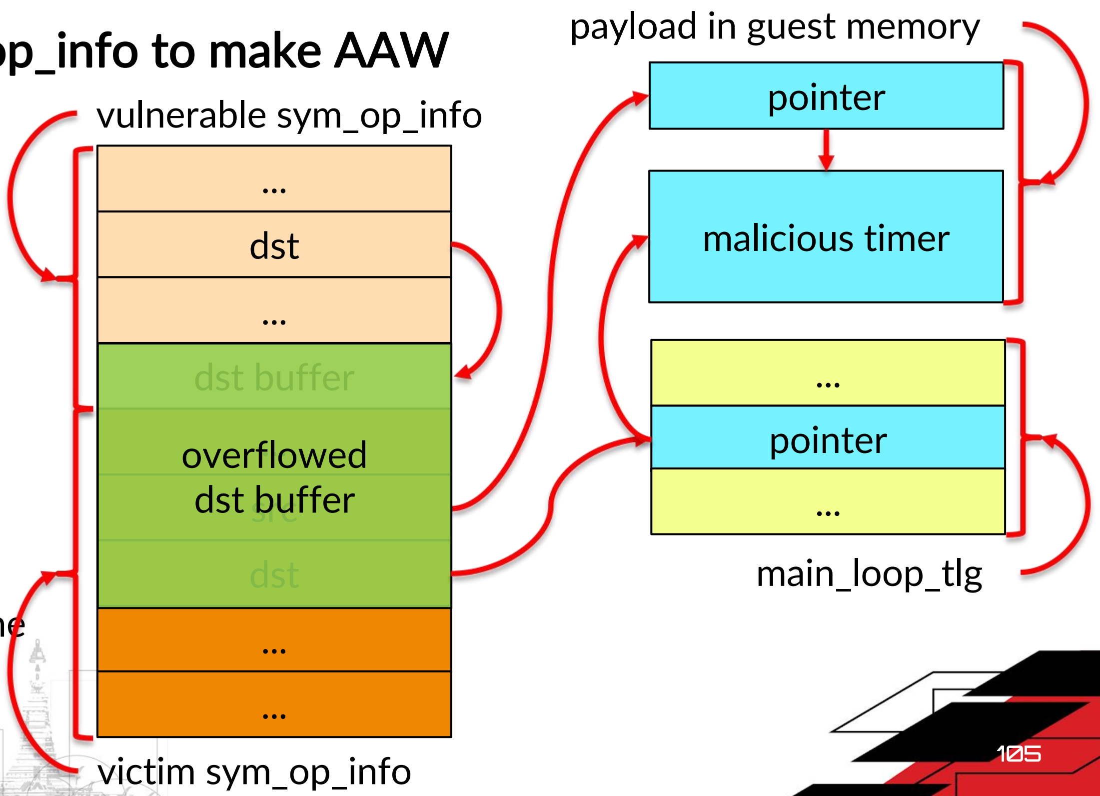
- Prepare a vulnerable sym_op_info
- Put another victim sym_op_info next to the vulnerable sym_op_info
- Prepare payload in guest memory space
- Overwrite the victim sym_op_info->src and victim sym_op_info->dst



Control flow hijack -- our plan

Method 1: oob write sym_op_info to make AAW

- Prepare a vulnerable sym_op_info
- Put another victim sym_op_info next to the vulnerable sym_op_info
- Prepare payload in guest memory space
- Overwrite the victim sym_op_info->src and victim sym_op_info->dst
- Wait for encryption process and hijack the QEMUTimerList



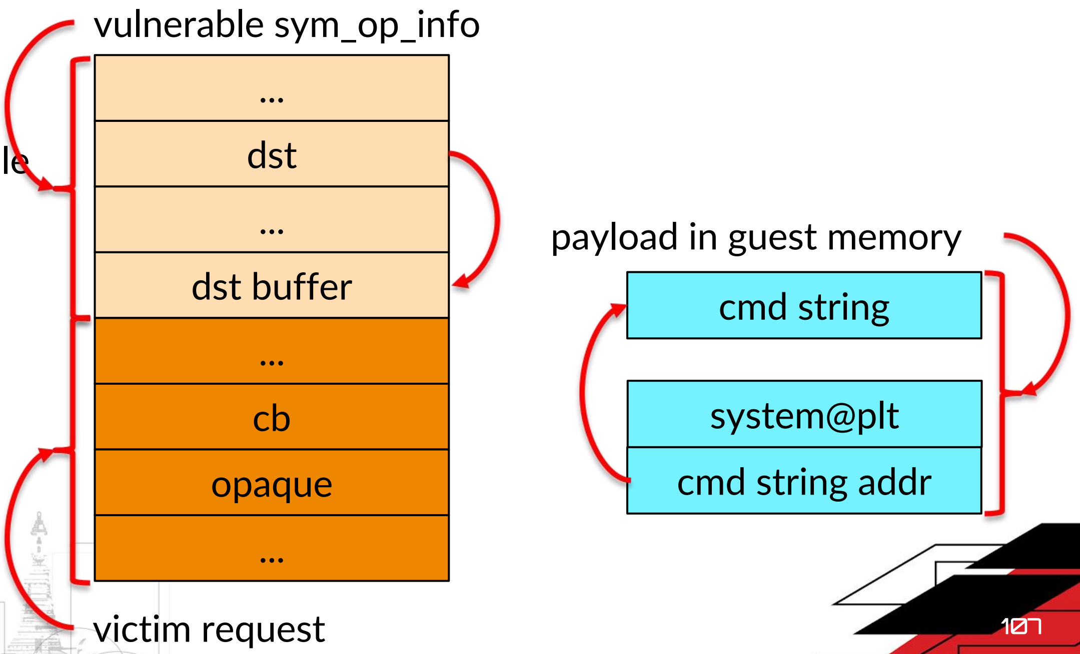
Control flow hijack -- our plan

Method 2: oob write request and hijack cb (*)

Control flow hijack -- our plan

Method 2: oob write request and hijack cb (*)

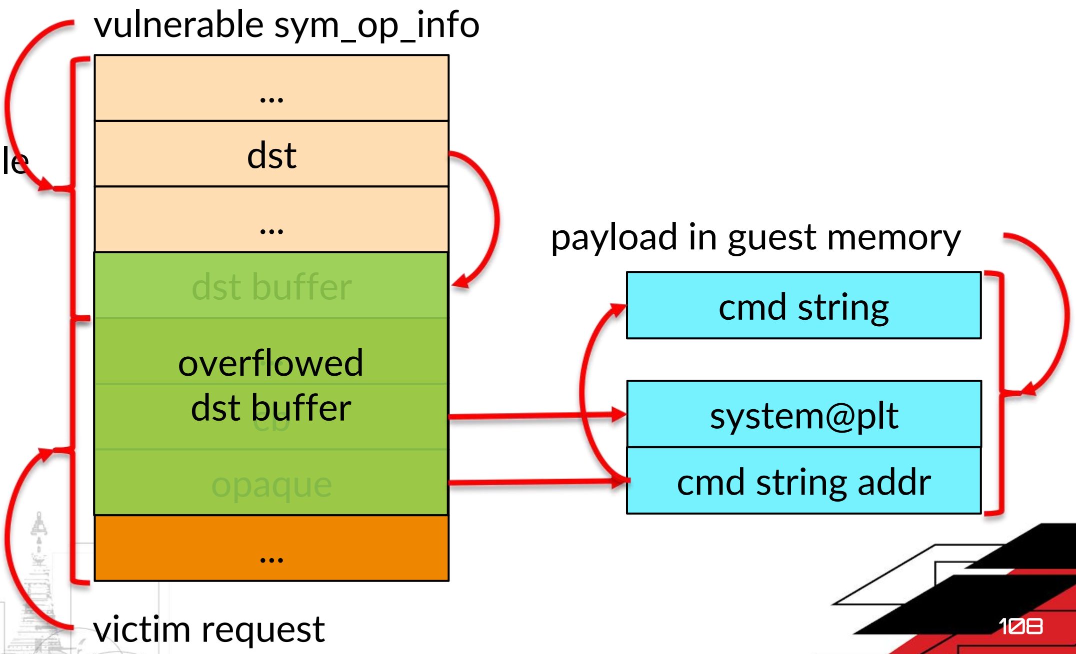
- Prepare a vulnerable sym_op_info
- Put a victim request next to the vulnerable sym_op_info
- Prepare payload in guest memory space



Control flow hijack -- our plan

Method 2: oob write request and hijack cb (*)

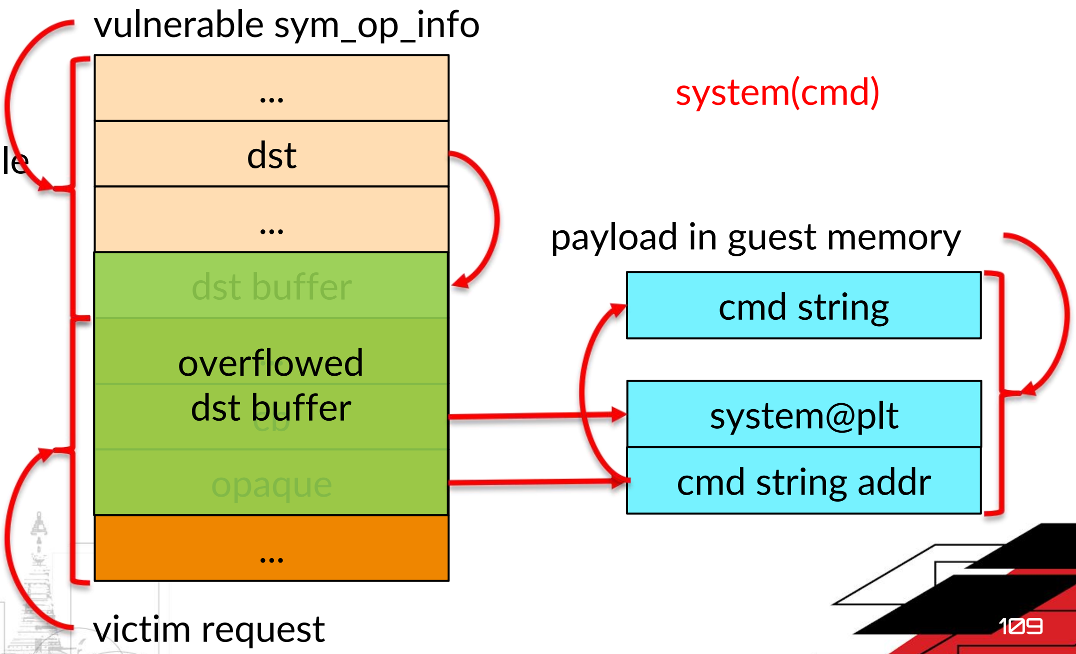
- Prepare a vulnerable sym_op_info
- Put a victim request next to the vulnerable sym_op_info
- Prepare payload in guest memory space
- Overwrite the victim request->cb and request->opaque



Control flow hijack -- our plan

Method 2: oob write request and hijack cb (*)

- Prepare a vulnerable sym_op_info
- Put a victim request next to the vulnerable sym_op_info
- Prepare payload in guest memory space
- Overwrite the victim request->cb and request->opaque
- Wait for the victim request to be done

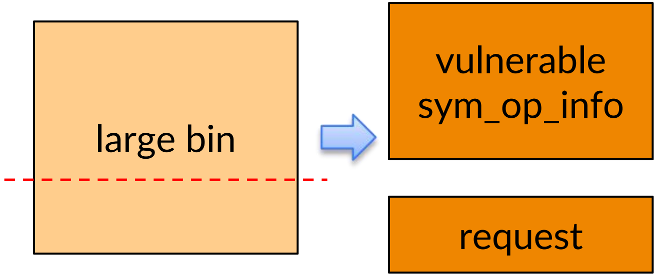


Control flow hijack -- main steps

1. Make a encryption request and occupy the vrtio-crypto device for a certain time
2. Heap spray and clear those large bins with small size
3. Allocate the vulnerable sym_op_info with size in range of large bin, so that it will split from a large bin and leave the remainder as a large bin
4. Allocate the victim request with size in range of large bin, so that it will malloc from the remainder large bin and be next to the vulnerable sym_op_info
5. Overwrite the request->cb and request->opaque to hijack control flow

Control flow hijack -- details

Same as what we do in “Information Leakage” to make heap manipulation.



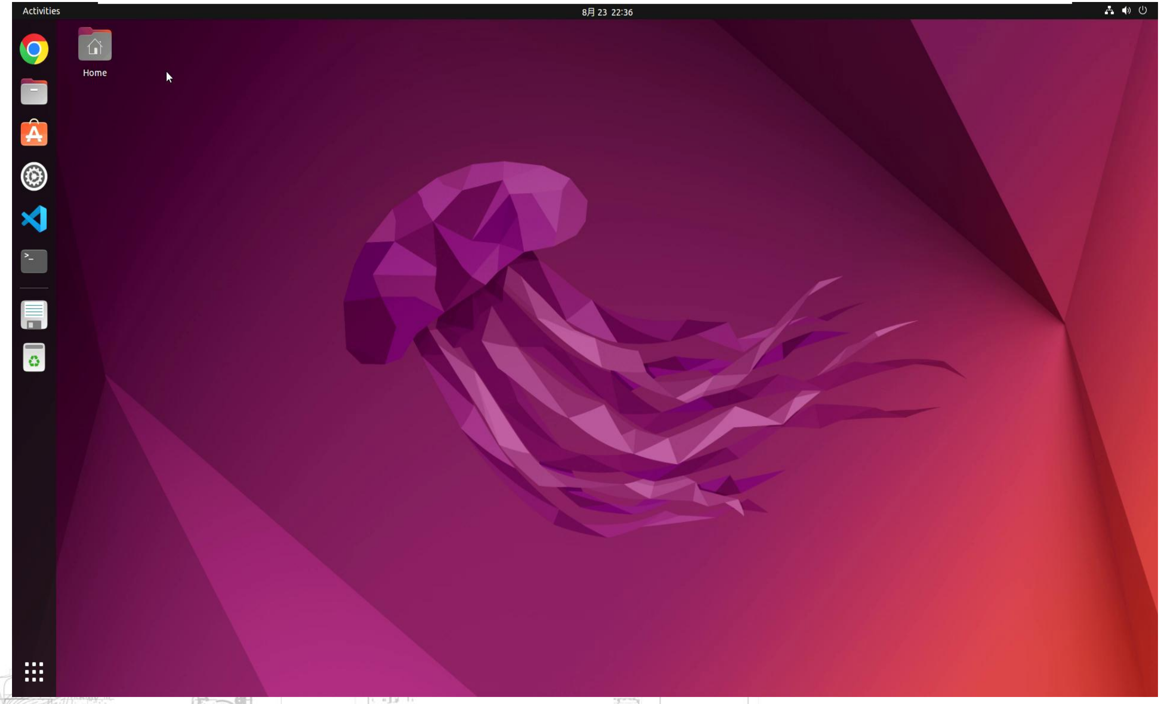
Control flow hijack -- details

When the cb is called, we just make a control flow hijack

```
gef➤  p *op_info
$1 = {
  algtype = QCRYPTODEV_BACKEND_ALG_SYM,
  op_code = 0x0,
  queue_index = 0x0,
  cb = 0x562a218cfc74 <system@plt+4>,
  opaque = 0x7f8981652000,
  session_id = 0x3,
  u = {
    sym_op_info = 0x562a25033d90,
    asym_op_info = 0x562a25033d90
  },
  next = {
    tqe_next = 0x0,
    tqe_circ = {
      tql_next = 0x0,
      tql_prev = 0x0
    }
  }
}
gef➤  x/s op_info->opaque
0x7f8981652000: "gnome-calculator"
gef➤
```

```
[#0] 0x7f89e7e50d60 → __libc_system(line=0x7f8981652000 "gnome-calculator")
[#1] 0x562a21bd02b2 → cryptodev_builtin_operation(backend=0x562a243322f0, op_info=0x562a25096b00)
[#2] 0x562a21bd0b9d → cryptodev_backend_operation(backend=0x562a243322f0, op_info=0x562a25096b00)
[#3] 0x562a21bd0f54 → cryptodev_backend_throttle_timer_cb(opaque=0x562a243322f0)
[#4] 0x562a22082e9e → timerlist_run_timers(timer_list=0x562a240c5d50)
[#5] 0x562a22083007 → timerlistgroup_run_timers(tlg=0x562a240c5c40)
[#6] 0x562a2205f64b → aio_dispatch(ctx=0x562a240c5b40)
[#7] 0x562a2207caed → aio_ctx_dispatch(source=0x562a240c5b40, callback=0x0, user_data=0x0)
[#8] 0x7f89e91b5d3b → g_main_context_dispatch()
[#9] 0x562a2207e0fc → glib_pollfds_poll()
```

Demo



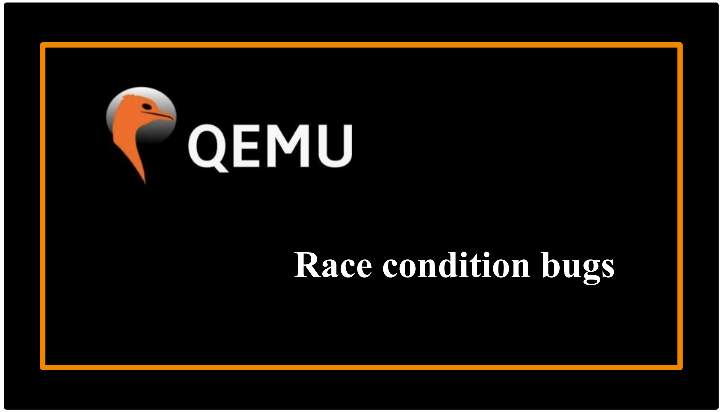
Conclusion

New Exploit Skill

- We propose two methods to help exploit heap overflow write vulnerabilities.
- Exploit conditions
  - heap overflow write vulnerability
  - overflow size >= 0x48

Begin of Story

- Find some race condition bugs in QEMU



End of Story

- We failed to find race conditional bugs in QEMU
- We find a new exploit skill in QEMU

A watched flower never blooms, but an untended willow grows.

THANK YOU!