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

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 an open source virtualization framework
- QEMU
  - Device virtualization (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.
Asynchronous Clock
Asynchronous Nature

● Why?
  ○ Avoid bloking

● How?
  ○ Multithreading
  ○ Timer(Asynchronous 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

```c
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 tlgs;

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 QEMU Timer 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**

```c
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

QEMUTimer

... *cb

opaque

gnome-calculator

... ...

active_timers

system@plt
Throttle -- Introduction

- 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](https://github.com/qemu/qemu/blob/master/docs/throttle.txt)
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

Commits on Nov 2, 2022

virtio-crypto: Support asynchronous mode

Lei He authored and mstirkin committed on Nov 2, 2022

Commits on Jun 16, 2022

crypto: Introduce RSA algorithm

pizhenwei authored and mstirkin committed on Jun 17, 2022
Why We Choose Virtio Crypto

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

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

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

```c
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

```c
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;
};
```

```c
struct virtio_crypto_ctrl_header {
    uint32_t opcode;
    uint32_t algo;
    uint32_t flag;
    /* data virtqueue id */
    uint32_t queue_id;
};
```
Request of Data Queue

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

```c
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;
};
```

```c
/* 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

```c
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

```c
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-backendbuiltin, id=cryptodev0, throttle-bps=32, throttle-ops=10
  -device virtio-crypto-pci, id=cyphero0, cryptodev=cryptodev0

- **Throttle**

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

Virtio-crypto → AES
Summary

Virtio-crypto → AES → Synchronous
Summary

Virtio-crypto

AES

Synchronous

Asynchronous
Summary

- Virtio-crypto
- AES
  - Synchronous
- RSA
  - Asynchronous
Summary

- Virtio-crypto
- AES
- RSA
- Synchronous
- Asynchronous
Summary

Virtio-crypto

AES

Synchronous

RSA

Asynchronous
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
The text and diagram describe a process involving V-SHUTTLE, which is a tool for automated fuzz testing in the context of device emulators and hypervisors.

**Before**

- **pci_dma_read (buffer_addr, &buf, size);**

**After**

- If (fuzzing_mode)
  
  - `read_from_testcase (&buf, size);`

The diagram illustrates the flow of data through different components:

- **Hypervisor**
- **Guest Memory**
- **Device Emulators**
- **DMA (Data Movement)**
- **Fuzzed Input**
- **Redirected DMA**

The process involves the use of API hooking to redirect DMA operations and handle fuzzy inputs.

One-dimensional vectors are mentioned as part of the redirected DMA operations.
Modified V-SHUTTLE

API Hooking

<Before>

Hypervisor

Guest Memory

Virtio Device Emulators

iov_to_buf(iov, &buf, size);

Vring

<After>

If (fuzzing_mode)
read_from_testcase (&buf, size);

DATA1
DATA2
DATA3
...

one-dimensional vectors

Redirected Vring

Fuzzed Input
Crash

<table>
<thead>
<tr>
<th>American fuzzy lop 2.52b (qemu-system-x86_64)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Process timing</strong></td>
</tr>
<tr>
<td>run time</td>
</tr>
<tr>
<td>last new path</td>
</tr>
<tr>
<td>last uniq path</td>
</tr>
<tr>
<td>cycle progress</td>
</tr>
<tr>
<td>now processing</td>
</tr>
<tr>
<td>paths timed out</td>
</tr>
<tr>
<td><strong>Stage progress</strong></td>
</tr>
<tr>
<td>now trying</td>
</tr>
<tr>
<td>stage execs</td>
</tr>
<tr>
<td>total execs</td>
</tr>
<tr>
<td>exec speed</td>
</tr>
<tr>
<td><strong>Fuzzing strategy yields</strong></td>
</tr>
<tr>
<td>bit flips</td>
</tr>
<tr>
<td>byte flips</td>
</tr>
<tr>
<td>arithmetics</td>
</tr>
<tr>
<td>known ints</td>
</tr>
<tr>
<td>dictionary</td>
</tr>
<tr>
<td>havoc</td>
</tr>
<tr>
<td>trim</td>
</tr>
</tbody>
</table>

| Overall results                               |
| cycles done                                    | 0                                             |
| total paths                                    | 228                                           |
| uniq crashes                                   | 0                                             |
| uniq hangs                                     | 0                                             |

| Map coverage                                   |
| map density                                    | 0.42% / 12.25%                                |
| count coverage                                 | 1.64 bits/tuple                               |

| Findings in depth                              |
| favored paths                                  | 15 (6.58%)                                    |
| new edges on                                   | 116 (50.88%)                                  |
| total crashes                                  | 0 (0 unique)                                  |
| total tmouts                                   | 0 (0 unique)                                  |

| Path geometry                                  |
| levels                                         | 2                                             |
| pending                                        | 228                                           |
| pend fav                                       | 15                                            |
| own finds                                      | 139                                           |
| imported                                       | n/a                                           |
| stability                                      | 4.96%                                         |

[-] PROGRAM ABORT: Unable to request new process from fork server (OOM?)

Location: run_target(), afl-fuzz.c:237
After Fuzzing

● Coverage

<table>
<thead>
<tr>
<th></th>
<th>Hit</th>
<th>Total</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lines</td>
<td>528</td>
<td>661</td>
<td>79.9 %</td>
</tr>
<tr>
<td>Functions</td>
<td>31</td>
<td>37</td>
<td>83.8 %</td>
</tr>
<tr>
<td>Branches</td>
<td>146</td>
<td>231</td>
<td>63.2 %</td>
</tr>
</tbody>
</table>

● 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

- Wrong encryption type
- Excessive length of the op_info

```c
/* 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);
```
2. NPD in cryptodev_backend_account

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

```c
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);
                ...
```
Patch

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

```c
--- a/backends/cryptodev.c
+++ b/backends/cryptodev.c
@@ -191,6 +191,11 @@ static int cryptodev_backend_account(CryptoDevBackend
    *backend,
    if (algtype == QCRYPTODEV_BACKEND_ALGASYM) {
      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:
```
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
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);
    }...
    ...
4. CVE-2023-3180: Heap-based Buffer Overflow

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

```c
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

![Diagram showing buffer overflow](image)
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

OOB write

- Malloc Primitive
  - Heap spray
  - Heap fengshui

Exploitable Structure
- Data pointer
- Function pointer

Information Leakage
Control Flow Hijack

Arbitrary Code Execution
Previous work

**virtio-gpu:** helps information leakage

- Leverage the uninitialized data in malloced chunk to leak

```c
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;
        }
        ...
    }
}
```
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
### 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

```c
/* 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;
    ...
}
```

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 */
...
};
```

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

```c
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
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.

```c
typedef struct CryptoDevBackendSymOpInfo {
    uint32_t aad_len;
    uint32_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

- 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 structure residing in memory.

- The vulnerable `sym_op_info` object could not overflow any other useful structures inside `virtio-crypto`. The whole process makes sense.

- All these structures mentioned above 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.

Solution: asynchronous clock
Timerkiller

Make use of asynchronous clock

virtio-cryptop

req 0 | allocate | use | free
req 1 | allocate | wait | use | free
req 2 | allocate | wait | use | free
... |
req N | allocate | wait | use | free

timeline
Timerkiller

Make use of asynchronous clock

virtio-crypto

req 0 → allocate → use → free

req 1 → allocate → wait → use → free

req 2 → allocate → wait → use → free

... 

req N → allocate → wait → use → free
Make use of asynchronous clock

virtio-crypto

req 0 → allocate → use → free

req 1 → allocate → wait → use → free

req 2 → allocate → wait → use → free

...  

req N → allocate → wait → use → free

time window

timeline
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 an 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

<table>
<thead>
<tr>
<th>...</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>request</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>request</td>
</tr>
<tr>
<td>sym_op_info</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>asym_op_info</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>asym_op_info</td>
<td>sym_op_info</td>
</tr>
<tr>
<td>...</td>
<td>asym_op_info</td>
</tr>
<tr>
<td>sym_op_info</td>
<td>...</td>
</tr>
<tr>
<td>sym_op_info</td>
<td>request</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

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
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
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

<table>
<thead>
<tr>
<th>src buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
</tr>
<tr>
<td>in</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>cb</td>
</tr>
<tr>
<td>opaque</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>vulnerable sym_op_info</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dst</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>overflowed dst buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>src</td>
</tr>
<tr>
<td>dst</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>victim request</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>victim asym_op_info</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
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
How to turn oob write into an oob read?

Information leakage -- our plan

• Prepare a vulnerable sym_op_info
• Put na asy_op_info next to the vulnerable sym_op_info
• Put a request next to the asy_op_info
• Oob write asy_op_info->src
• Get oob read when execute asymmetric encryption

src buffer

... dst buffer ...

... opaque ...

... cb ...

victim asy_op_info

overflowed

dst buffer

overflowed

src buffer

request

...
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
Information leakage -- details

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

```
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_INUSE)

[+] small_bins[3]: fw=0x55e0237bd930, bk=0x55e023763080
  → Chunk (addr=0x55e0237b93530, size=0x30, flags=PREV_INUSE)
  → Chunk (addr=0x55e023c64720, size=0x50, flags=PREV_INUSE)

[+] 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_INUSE)

[+] small_bins[9]: fw=0x55e023216570, bk=0x55e023708230
  → Chunk (addr=0x55e023216580, size=0xa0, flags=PREV_INUSE)
```
Information leakage -- details

1. Make an 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 virtio-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
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.
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

payload in guest memory

pointer

malicious timer

vulnerable sym_op_info

...  
dst  
...  
dst buffer  
...  
src  
dst  ...  ...

victim 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
• 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

payload in guest memory

pointer

malicious timer

QEMUTimerList *

main_loop_tlg

...
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

```
method: oob request

Payload in guest memory:
- cmd string
- system@plt
- cmd string addr

Overwritten victim request:
- system(cmd)
```
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 = 0x562a218cf74 <system@plt+4>,
    opaque = 0x778981652000,
    session_id = 0x3,
    u = {
        sym_op_info = 0x562a25033d90,
        asym_op_info = 0x562a25033d90
    },
    next = {
        tqe_next = 0x0,
        tqe_circ = {
            tql_next = 0x0,
            tql_prev = 0x9
        }
    }
}

gef> x/s op_info->opaque
0x778981652000: "gnome-calculator"
Conclusion
New Exploit Skill

- We propose two methods to help exploit heap overflow write vulnerabilities.
- Exploit conditions
  - heap overflow write vulnerability
  - overflow size $\geq 0x48$
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!