A Disaster Caused By A Bug: A Black Box Escape Of Qemu Based On The USB Device

f1yyy@Chaitin.Tech
Lingni.Kong@Ocean University of China
Haipeng.Qu@Ocean University of China
About us

● Beijing Chaitin Tech Co., Ltd(@ChaitinTech)
  ○ https://chaitin.cn/en
  ○ https://realworldctf.com/

● Chaitin Security Research Lab
  ○ Pwn2Own 2017 3rd place
  ○ GeekPwn 2015/2016/2018/2019 awardees
    ■ PS4 Jailbreak, Android rooting, IoT Offensive Research, ESXi Escape
  ○ CTF players from team b1o0p, Tea Deliverers
    ■ 2nd place at DEFCON 2016
    ■ 3rd place at DEFCON 2019
    ■ 1st place at HITCON 2019
About us

- Information Security Lab of Ocean University of China
  http://security.ouc.edu.cn/

- OUC Security Research Lab
  - BCTF 2020 online round 1st place
  - WCTF World Hacker Masters 2019 3rd place
What is Virtual Machine Escape

Normally, all of the sensitive behaviors of guest OS will be sanitized by the hypervisor
What is Virtual Machine Escape

- **Guest OS 0**
- **Guest OS 1**
- **Guest OS N**

**VMM**

**Host OS**
What is Virtual Machine Escape

exploitation

Host OS

VMM

Guest OS 0

... Guest OS N

Guest OS 1
What is Virtual Machine Escape

- VMM
- Guest OS 0
- Guest OS 1
- Guest OS N

Exploitation

Guest OS 0 → ... → Guest OS 1 → ... → Guest OS N

VMM

Execute arbitrary codes on the host

Host OS

#!/bin/bash

Vork connection
Introduction of Qemu-KVM
Qemu

- Open source software
- Emulator

User Space

Linux Kernel
- Kernel-based Virtual Machine
- Encapsulates VMX or SVM
Qemu-KVM VM

- Qemu
  - Emulates other devices
- KVM
  - Emulates CPU and memory

User Space

QEMU

Linux Kernel

KVM
Qemu-KVM VM

- Qemu
  - Uses ioctl and /dev/kvm
- KVM
  - Provides a series of APIs to create and run VM
Qemu-KVM VM

- **Qemu**
  - Uses ioctl and /dev/kvm

- **KVM**
  - Provides a series of APIs to create and run VM
Libvirt

- A set of open source APIs, daemons and management tools for managing hardware virtualization
- Used by most public cloud providers.
Attack Surface of A Libvirt VM

Guest OS

Virtio-net  Virtio-blk  UHCI  Cirrus VGA

HID
Attack Surface of A Libvirt VM

- Virtio
  - Simple
  - Few code
  - Few CVEs

Guest OS

- Virtio-net
- Virtio-blk
- UHCI
- Cirrus VGA
- HID
Attack Surface of A Libvirt VM

- Cirrus VGA
  - Many CVEs
  - Hard to exploit
Attack Surface of A Libvirt VM

- UHCI
  - Universal Host Controller Interface
  - USB 1.0

- HID
  - Human Interface Device
  - mouse/keyboards

Guest OS

Virtio-net  Virtio-blk  UHCI  Cirrus VGA

HID
Why hard to escape from the public cloud?

- Lack of good vulnerabilities
- Lack of further information

Guest OS

- Virtio-net
- Virtio-blk
- UHCI
- Cirrus VGA
- HID
CVE-2020-14364 Timeline

- Reported at 2020.8.13
- Redhat fixed it and disclosed it at 2020.8.24
How Does Guest OS send usb packets?

Universal Host Controller Interface

 registers
   flbase_addr_low
   flbase_addr_high
   frnum
   cmd

Guest OS
How Does Guest OS send USB packets?

Universal Host Controller Interface

- Registers
  - flbase_addr_low
  - flbase_addr_high
  - frnum
  - cmd
How Does Guest OS send usb packets?

Universal Host Controller Interface

- write
- flbase_addr_low
- flbase_addr_high
- fnnum
- cmd

USB Packets
How Does Guest OS send USB packets?

- Guest OS
  - flbase_addr_low
  - flbase_addr_high
  - fnnum
  - cmd

Universal Host Controller Interface

Write

USB Packets
How Does Guest OS send USB packets?

Guest OS

- flbase_addr_low
- flbase_addr_high
- frnum
- cmd

write

Universal Host Controller Interface

USB Packets

Control Endpoint

USB Device

Data Endpoint
How Does Guest OS send usb packets?

Guest OS

write

flbase_addr_low
flbase_addr_high
frnum
cmd

Universal Host Controller Interface

USB Packets

Control Endpoint

Data Endpoint

USB Device
How Do Qemu transfer USB packets?

USB Device

Control Endpoint

USB Device

Data Endpoint

USB Packets

SETUP

DATA_IN

DATA_OUT
How Do Qemu transfer USB packets?

USB Packets

SETUP

DATA_IN

DATA_OUT

Control Endpoint

USB Device

Data Endpoint
How Do Qemu transfer USB packets?

USB Packets

- SETUP
- DATA_IN
- DATA_OUT

DIR_OUT

DIR_IN

Control Endpoint

USB Device

Data Endpoint
How Do Qemu transfer USB packets?

USB Packets

- SETUP
- DATA_IN
- DATA_OUT

DIR_OUT

You want to **set** control information

Control Endpoint

USB Device

Data Endpoint
How Do Qemu transfer USB packets?

You want to **get** control information.
How Do Qemu transfer USB packets?

USB Packs → SETUP → DIR_OUT → DIR_IN → DATA_IN → DATA_OUT

USB Device

Control Endpoint

Data Endpoint
How Do Qemu transfer USB packets?

USB Packets

SETUP

DIR_OUT

Waiting Data

Control Endpoint

USB Device

Data Endpoint

DIR_IN

DATA_IN

DATA_OUT
How Do Qemu transfer USB packets?

USB Packets

- SETUP
- DATA_IN
- DATA_OUT
- DIR_IN
- DIR_OUT
- Waiting Data

Another Packet

Control Endpoint

USB Device

Data Endpoint
How Do Qemu transfer USB packets?

USB Packets

SETUP

DATA_IN

DATA_OUT

DIR_OUT

Waiting Data

Control Endpoint

USB Device

Data Endpoint

Another Packet
How Do Qemu transfer USB packets?

USB Packets
- SETUP
- DATA_OUT
- DATA_IN
- DIR_OUT
- Waiting Data

Another Packet

Control Endpoint
USB Device
Data Endpoint
How Do Qemu transfer USB packets?

USB Packets

- SETUP
- DATA_IN
- DATA_OUT

USB Device

- Control Endpoint
- Data Endpoint

USB Packets flow to the USB Device through the SETUP, DATA_IN, and DATA_OUT endpoints.
How Do Qemu transfer USB packets?

USB Packets

 SETUP

 DIR_IN

 DIR_OUT

 DATA_IN

 DATA_OUT

 Control Endpoint

 USB Device

 Data Endpoint
How Do Qemu transfer USB packets?

USB Packets

SETUP

DIR_IN

DIR_OUT

Control Endpoint

USB Device

Data Endpoint

DATA_IN

DATA_OUT
How Do Qemu transfer USB packets?

USB Packets

SETUP

DIR_IN

BACK Data

Control Endpoint

USB Device

DATA_IN

DATA_OUT

DATA_IN

DATA_OUT
How Do Qemu transfer USB packets?

USB Packets

- SETUP
- DATA_IN
- DIR_IN
- Back Data
- DIR_OUT

Control Endpoint

USB Device

Data Endpoint

Another Packet
How Qemu transfer USB packets?

USB Packets

- SETUP
- DATA_IN
- DIR_IN
- Back Data

Control Endpoint
- USB Device
  - Data Endpoint

Data Path: USB Packets → Setup → DATA_IN → BACK Data → Control Endpoint
How Do Qemu transfer USB packets?

USB Packets

DATA_IN

DATA_OUT

SETUP

Control Endpoint

USB Device

Data Endpoint
How Do Qemu transfer USB packets?

USB Packets → DATA_IN → Control Endpoint

DATA_IN → DATA_OUT → Data Endpoint
How Do Qemu transfer USB packets?

USB Packets

SETUP

DATA_IN

DATA_OUT

Control Endpoint

USB Device

Data Endpoint
How Do Qemu transfer USB packets?

USB Packets

SETUP

DATA_IN

DATA_OUT

Control Endpoint

USB Device

Data Endpoint
CVE-2020-14364

USB Packets

SETUP

DIR_OUT

Waiting Data

Control Endpoint

USB Device

Data Endpoint

DATA_IN

DATA_OUT

DATA_IN

DATA_OUT
USB Packets

<table>
<thead>
<tr>
<th>SETUP</th>
<th>DIR_OUT</th>
<th>Waiting Data</th>
</tr>
</thead>
</table>

1. static void do_token_setup(USBDevice *s, USBPacket *p){
2.     ...  
3.     usb_packet_copy(p, s->setup_buf, p->iov.size);
4.     s->setup_index = 0;
5.     p->actual_length = 0;
6.     s->setup_len   = (s->setup_buf[7] << 8) | s->setup_buf[6];
7.     if (s->setup_len > sizeof(s->data_buf)) {
8.         fprintf(stderr,
9.             "usb_generic_handle_packet: ctrl buffer too
10.               small (%d > %zu)\n",
11.               s->setup_len, sizeof(s->data_buf));
12.         p->status = USB_RET_STALL;
13.         return;
14.     }  
15.     if (s->setup_buf[0] & USB_DIR_IN) {
16.         ...  
17.     }else{
18.         s->setup_state = SETUP_STATE_DATA;
1. static void do_token_setup(USBDevice *s, USBPacket *p){
2.     ...
3.     usb_packet_copy(p, s->setup_buf, p->iov.size);
4.     s->setup_index = 0;
5.     p->actual_length = 0;
6.     s->setup_len   = (s->setup_buf[7] << 8) | s->setup_buf[6];
7.     if (s->setup_len > sizeof(s->data_buf)) {
8.         fprintf(stderr,
9.                 "usb_generic_handle_packet: control buffer too small (%d > %zu)\n",
10.            s->setup_len, sizeof(s->data_buf));
11.            p->status = USB_RET_STALL;
12.            return;
13.        }
14.        if (s->setup_buf[0] & USB_DIR_IN) {
15.            ...
16.        }else{
17.            s->setup_state = SETUP_STATE_DATA;
1. static void do_token_setup(USBDevice *s, USBPacket *p){
2.     ...
3.     usb_packet_copy(p, s->setup_buf, p->iov.size);
4.     s->setup_index = 0;
5.     p->actual_length = 0;
6.     s->setup_len   = (s->setup_buf[7] << 8) | s->setup_buf[6];
7.     if (s->setup_len > sizeof(s->data_buf)) {
8.         fprintf(stderr,
9.             "usb_generic_handle_packet: ctrl buffer too
small (%d > %zu)\n",
10.            s->setup_len, sizeof(s->data_buf));
11.            p->status = USB_RET_STALL;
12.            return;
13.        }
14.    if (s->setup_buf[0] & USB_DIR_IN) {
15.        ...
16.    }else{
17.        s->setup_state = SETUP_STATE_DATA;
1. static void do_token_setup(USBDevice *s, USBPacket *p) {
2.     ...
3.     usb_packet_copy(p, s->setup_buf, p->iov.size);
4.     s->setup_index = 0;
5.     p->actual_length = 0;
6.     s->setup_len = (s->setup_buf[7] << 8) | s->setup_buf[6];
7.     if (s->setup_len > sizeof(s->data_buf)) {
8.         fprintf(stderr,
9.             "usb_generic_handle_packet: ctrl buffer too small (%d > %zu)\n",
10.            s->setup_len, sizeof(s->data_buf));
11.         p->status = USB_RET_STALL;
12.         return;
13.     }
14.     if (s->setup_buf[0] & USB_DIR_IN) {
15.         ...
16.     } else {
17.         s->setup_state = SETUP_STATE_DATA;
18.     }
19. }

Get the length of setting data
1. static void do_token_setup(USBDevice *s, USBPacket *p) {
2.     ...
3.     usb_packet_copy(p, s->setup_buf, p->iov.size);
4.     s->setup_index = 0;
5.     p->actual_length = 0;
6.     s->setup_len = (s->setup_buf[7] << 8) | s->setup_buf[6];
7.     if (s->setup_len > sizeof(s->data_buf)) {
8.         fprintf(stderr, "usb_generic_handle_packet: ctrl buffer too small (%d > %zu)\n",
9.                 s->setup_len, sizeof(s->data_buf));
10.        p->status = USB_RET_STALL;
11.        return;
12.    }
13.    else{
14.        if (s->setup_buf[0] & USB_DIR_IN) {
15.            ...
16.        }else{
17.            s->setup_state = SETUP_STATE_DATA;
18.        }
19.    }
20.}
```c
static void do_token_setup(USBDevice *s, USBPacket *p) {
    ...
    usb_packet_copy(p, s->setup_buf, p->iov.size);
    s->setup_index = 0;
    p->actual_length = 0;
    s->setup_len = (s->setup_buf[7] << 8) | s->setup_buf[6];
    if (s->setup_len > sizeof(s->data_buf)) {
        fprintf(stderr,

            "usb_generic_handle_packet: ctrl buffer too small (%d > %zu)\n",

        s->setup_len, sizeof(s->data_buf));
        p->status = USB_RETSTALL;
        return;
    } else {
        s->setup_state = SETUP_STATE_DATA;
    }
    ...}
```
CVE-2020-14364

USB Packets

SETUP

DATA_OUT

Waiting Data

Another Packet
1. static void do_token_out(USBDevice *s, USBPacket *p){
2.     ...
3.     switch(s->setup_state) {
4.         ...
5.         case SETUP_STATE_DATA:
6.             if (!(s->setup_buf[0] & USB_DIR_IN)) {
7.                 int len = s->setup_len - s->setup_index;
8.                 if (len > p->iov.size) {
9.                     len = p->iov.size;
10.                }
11.                usb_packet_copy(p, s->data_buf + s->setup_index, len);
12.                s->setup_index += len;
13.                if (s->setup_index >= s->setup_len) {
14.                    s->setup_state = SETUP_STATE_ACK;
15.                }
16.             return;
17.         }
18.     }

First, we send an 8-byte SETUP packet to make the s->setup_state be SETUP_STATE_DATA
Next, we send another 8-byte SETUP packet to make the s->setup_len big enough.
Finally, we send another N-byte DATAOUT packet. We will overflow-write to the data buffer.
Black Box Escape
Why We Need Black Box Escape

- It’s hard for an attacker to get following information
  - Qemu’s version
  - The binary file of Qemu
What do we have now

- We can do out-of-bound read and write of the databuf between 0-0xffff.

```c
struct USBDevice {
    DeviceState qdev;
    ...
    uint8_t setup_buf[8];
    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;
    /* Overrides class usb_desc if not NULL */
    ...}
```
What do we have now

- We can do out-of-bound read and write of the databuf between 0-0xffff.
- How to leak some key information?

```c
struct USBDevice {
    DeviceState qdev;
    ... 
    uint8_t setup_buf[8];
    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;
    /* Overrides class usb_desc if not NULL */
    ...
```
What do we have now

- We can do out-of-bound read and write of the databuf between 0-0xffff.
- How to leak some key information?
- The usb_desc contains the description of this USB device.

```c
struct USBDevice {
    DeviceState qdev;
    ... 
    uint8_t setup_buf[8];
    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;
    /* Overrides class usb_desc if not NULL */
    ... 
```
We can get the USBDescID by sending some USB packets.
● We can get the USBDescID by sending some USB packets.

● Arbitrary Address Read
  ○ Overwrite the pointer of USBDesc.
  ○ Get the USBDescID back.

```c
1. struct USBDesc {
2.     USBDescID                id;
3.     const USBDescDevice     *full;
4.     const USBDescDevice     *high;
5.     const USBDescDevice     *super;
6.     const char* const       *str;
7.     const USBDescMSOS       *msos;
8. }

9. struct USBDescID {
10.    uint16_t                  idVendor;
11.    uint16_t                  idProduct;
12.    uint16_t                  bcdDevice;
13.    uint8_t                   iManufacturer;
14.    uint8_t                   iProduct;
15.    uint8_t                   iSerialNumber;
16.    uint8_t                   iProduct;
17. }
```
struct USBDevice {
    DeviceState qdev;
    ...
    uint8_t setup_buf[8];
    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;
    /* Overrides class usb_desc if not NULL */
    ...
}
struct USBEndpoint {
    uint8_t nr;
    uint8_t pid;
    uint8_t type;
    uint8_t ifnum;
    int max_packet_size;
    int max_streams;
    bool pipeline;
    bool halted;
    USBDevice *dev;
    QTAILQ_HEAD(, USBPacket) queue;
};

• We get the address of USBDevice by reading the USBEndpoint(ep_ctl, ep_in or ep_out)
We get the address of `USBDevice` by reading the `USBEndpoint(ep_ctl, ep_in or ep_out)`

`DeviceState` has a free function pointers
1. struct DeviceState {
2.   /*< private >*/
3.   Object parent_obj;
4.   ...
5. };
6. 
7. struct Object {
8.   { /*< private >*/
9.   ObjectClass *class;
10.  ObjectFree *free;
11.  GHashTable *properties;
12.  uint32_t ref;
13.  Object *parent;
14. };
15. 
16. }

- We get the address of USBDevice by reading the USBEndpoint(ep_ctl, ep_in or ep_out)
- **DeviceState** has a free function pointers
1. struct DeviceState {
2.     /*< private >/*/
3.     Object parent_obj;
4.     ... 
5. }
6. 
7. struct Object {
8.     { 
9.         /*< private >/*/
10.         ObjectClass *class;
11.         ObjectFree *free;
12.         GHashTable *properties;
13.         uint32_t ref;
14.         Object *parent;
15.     };
16. }

- We get the address of USBDevice by reading the USBEndpoint(ep_ctl, ep_in or ep_out)
- DeviceState has a free function pointers
- We will finally get the free address in libc
Leak libc → Leak system

- After getting address of free, we get the address of system like pwntools.Dynelf does.
  - Search ELF magic number forward first to get the base address of libc
  - Find `.dynstr` and `.dynsym` section
  - Find “system” in `.dynstr` and get the offset in `.dynsym`
Control PC

USBHIDState

USBDevice dev;
...
HIDState hid;

HIDState

HIDMouseState ptr;
...
HIDEventFunc event
...

USBDevice

... 
uint8_t data_buf[4096];
...

1. static void hid_idle_timer(void *opaque)
2. {
3.     HIDState *hs = opaque;
4.     hs->idle_pending = true;
5.     hs->event(hs);
6. }

HIDMouseState ptr;
...
HIDEventFunc event
...
```c
Control PC

USBHIDState

USBDevice dev;
...
HIDState hid;

USBDevice

...
uint8_t data_buf[4096];
...

HIDState

HIDMouseState ptr;
...
HIDEventFunc event
...
```
Control PC

USBHIDState

USBDevice

... 
uint8_t data_buf[4096];
...

HIDState

“cat /etc/passwd > ./test”
...

... 
system
...
Conclusion

- Sandboxes are necessary even in public cloud environments
- Good vulnerabilities can do a lot of interesting things
- The skills used in CTF are helpful
Thanks!

@flyYY__