长亭科技
CHAITIN

# 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/

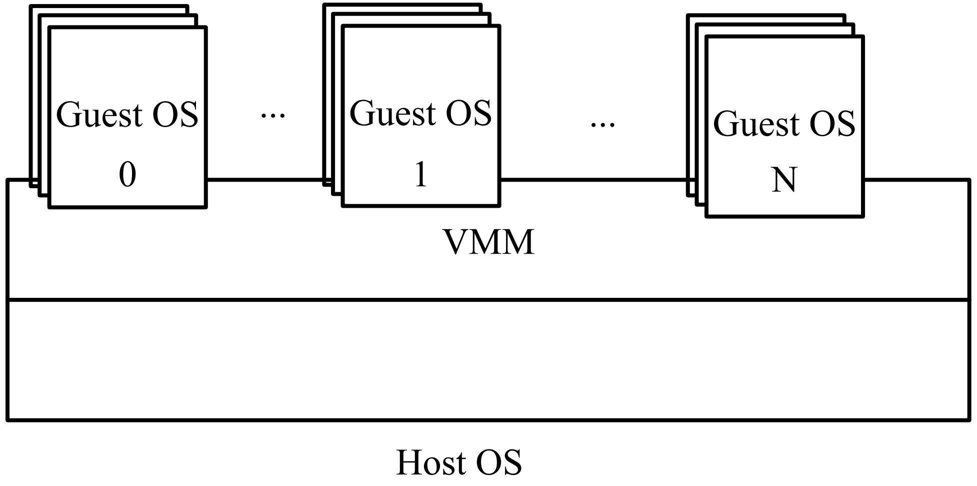
- Chaitin Security Research Lab
    - Pwn2Own 2017 3rd place
    - GeekPwn 2015/2016/2018/2019 awardees
        - PS4 Jailbreak, Android rooting, IoT Offensive Research, ESXi Escape
    - CTF players from team b1o0p, Tea Deliverers
        - 2nd place at DEFCON 2016
        - 3rd place at DEFCON 2019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

## About us

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

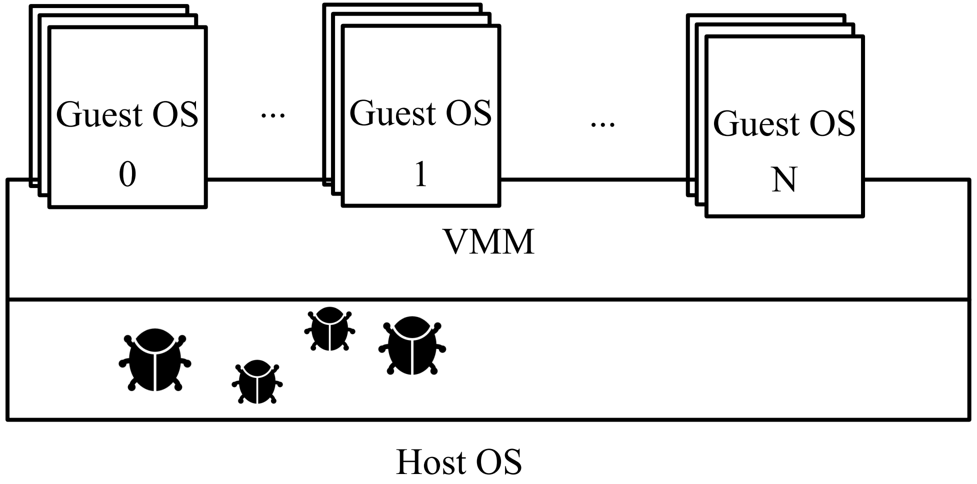
- OUC Security Research Lab
  - BCTF 2020 online round 1st place
  - WCTF World Hacker Masters 2019 3rd place

## What is Virtual Machine Escape

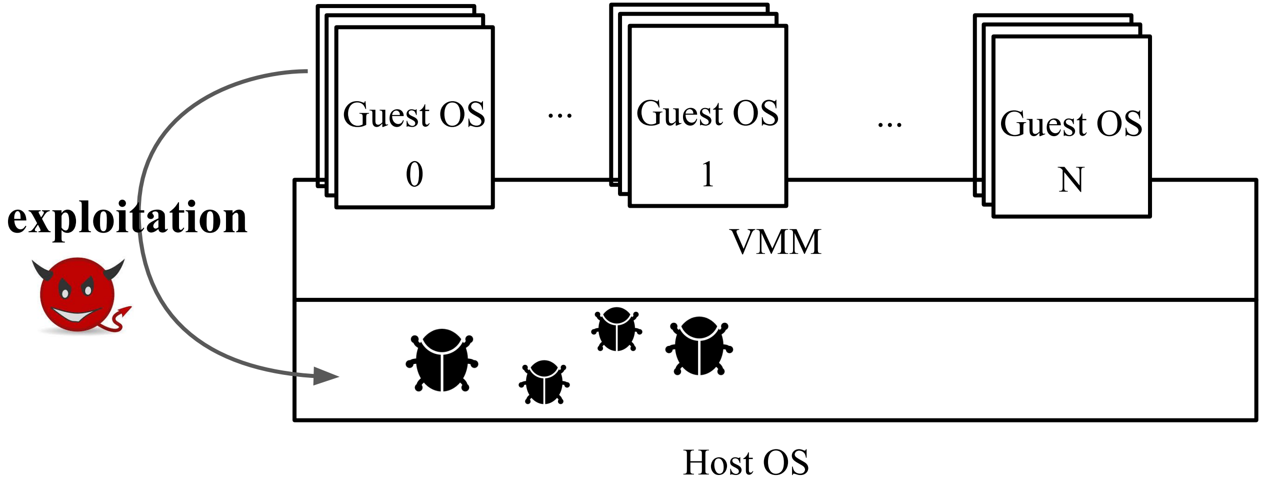


Normally, all of the sensitive behaviors of guest OS will be sanitized by the hypervisor

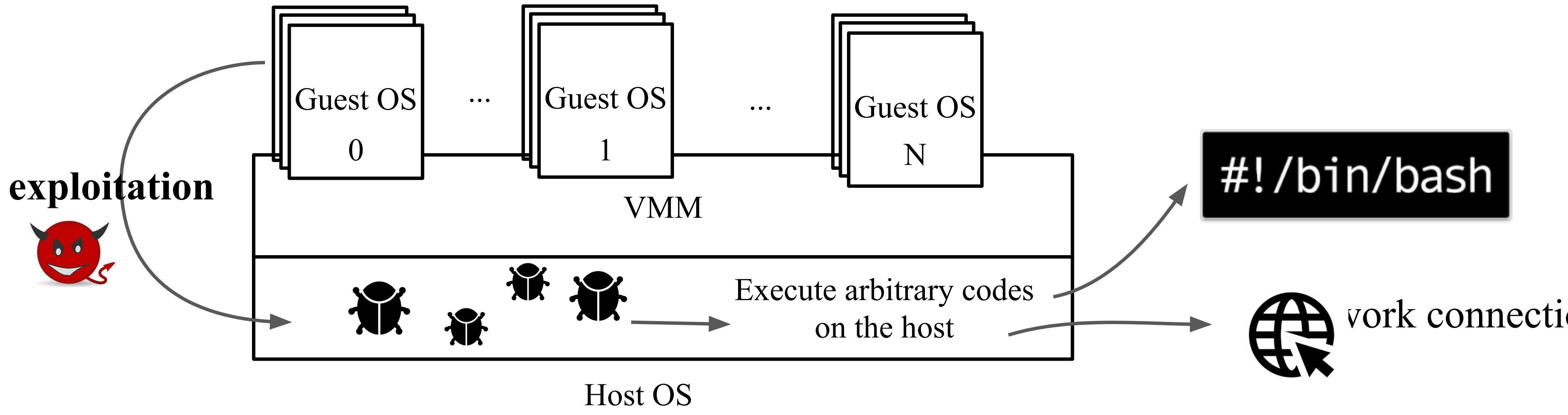
# What is Virtual Machine Escape



# What is Virtual Machine Escape



# What is Virtual Machine Escape

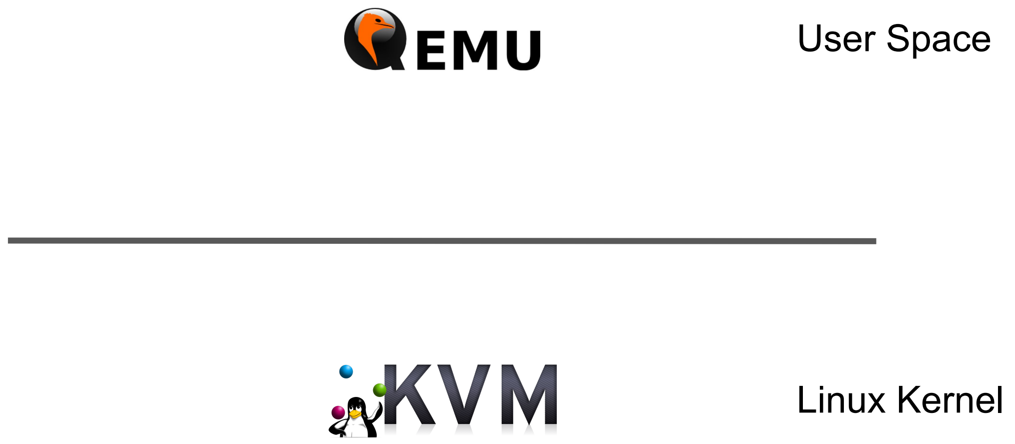


长亭科技
CHAITIN

# Introduction of Qemu-KVM

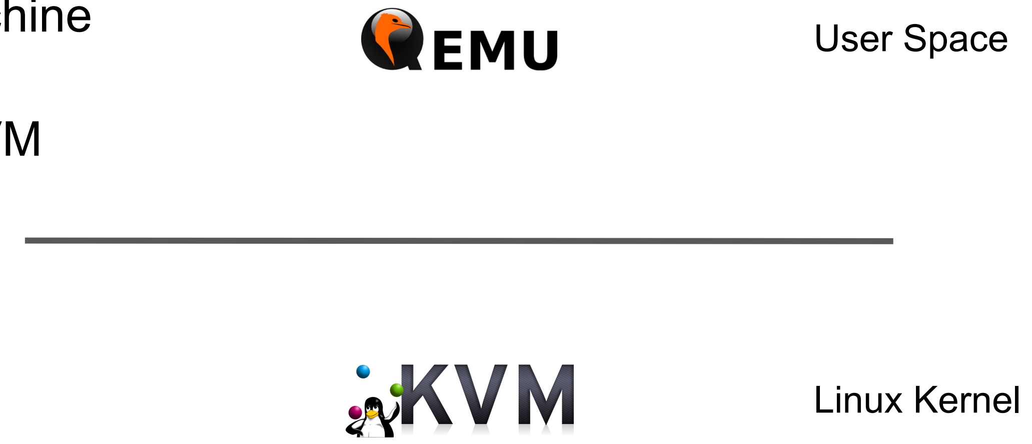
# Qemu

- Open source software
- Emulator



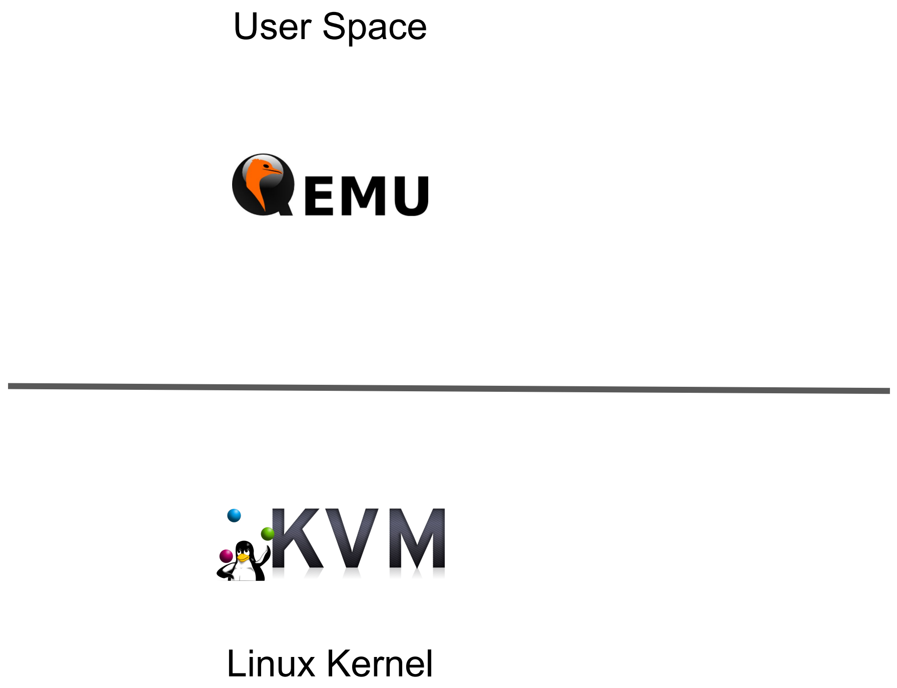
# KVM

- Kernel-based Virtual Machine
- Encapsulates VMX or SVM



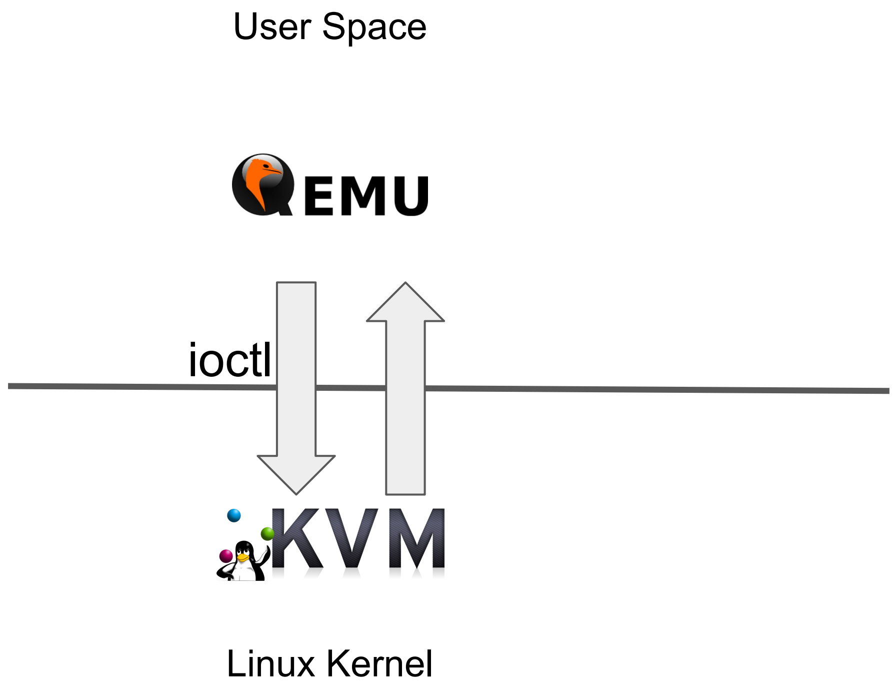
## Qemu-KVM VM

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



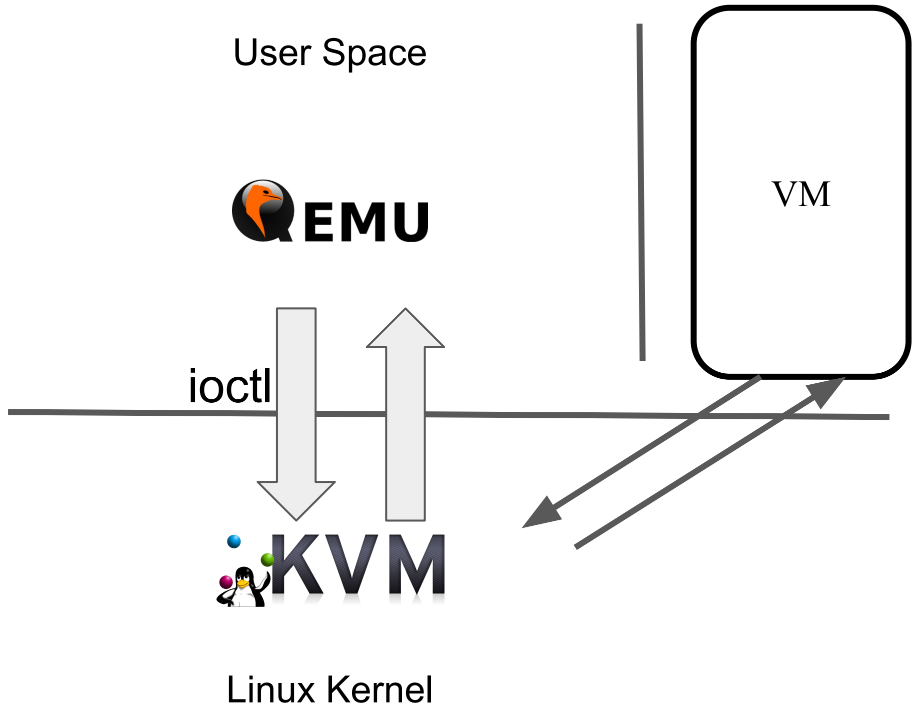
# Qemu-KVM VM

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



# Qemu-KVM VM

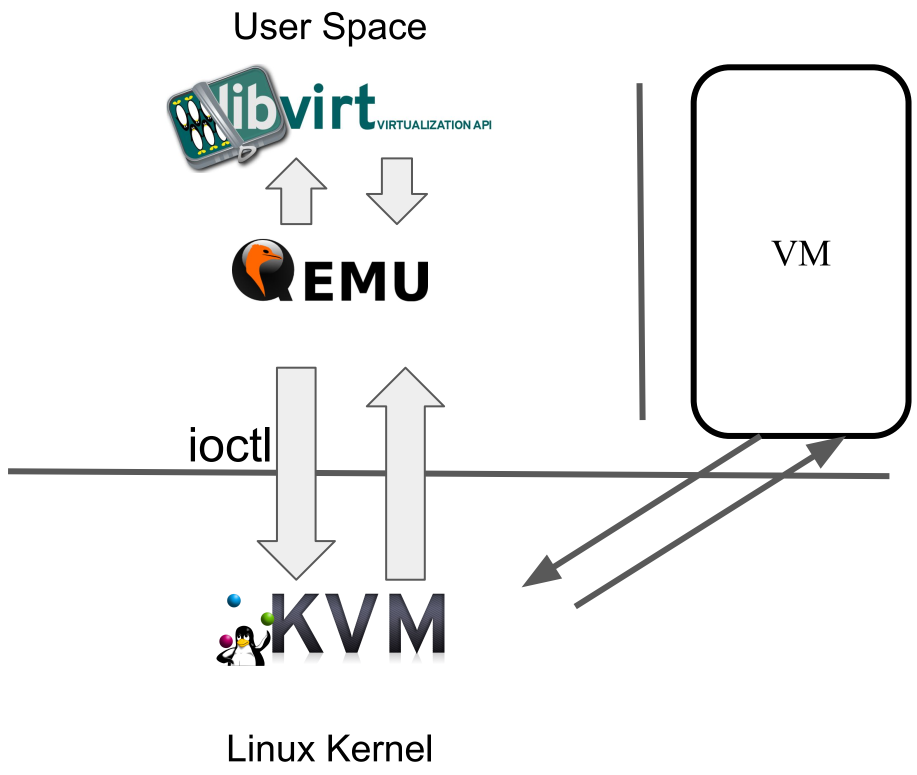
- Qemu
  - Uses ioctl and /dev/kvm
- KVM
  - Provides a series of APIs to create and run VM



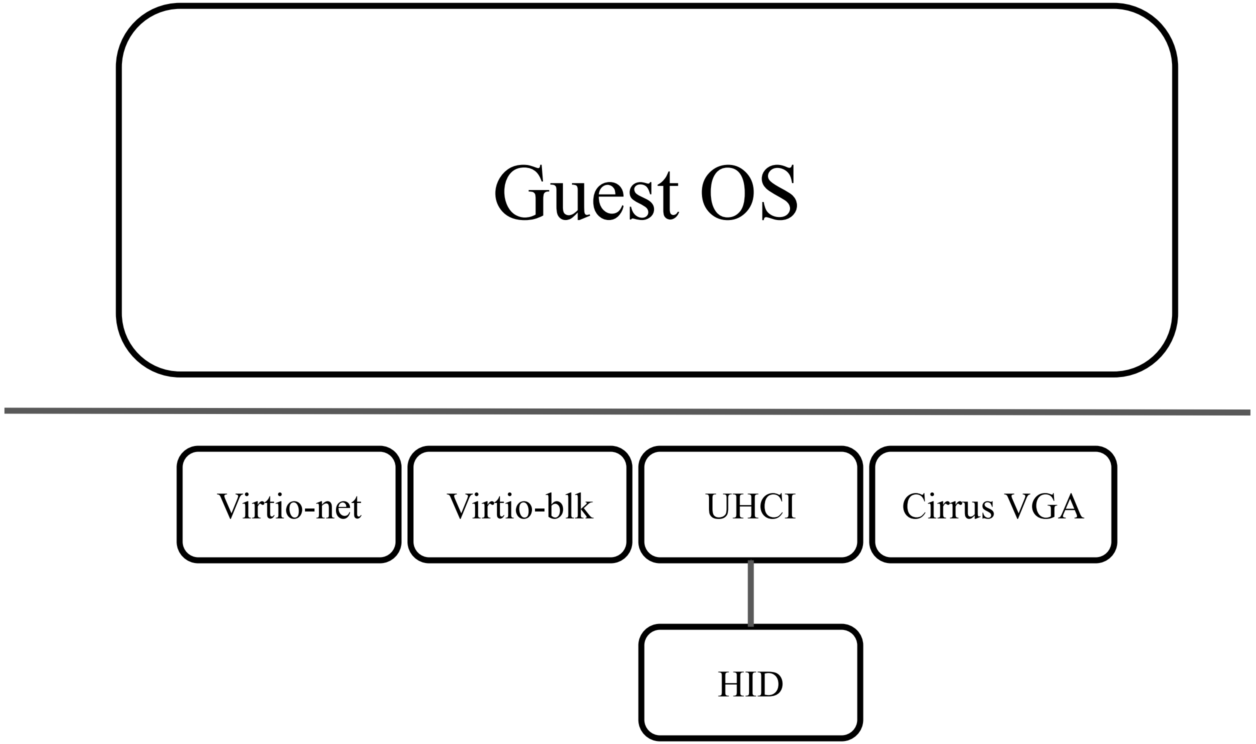
## Libvirt

- A set of open source APIs, daemons and management tools for managing hardware virtualization

- Used by most public cloud providers.

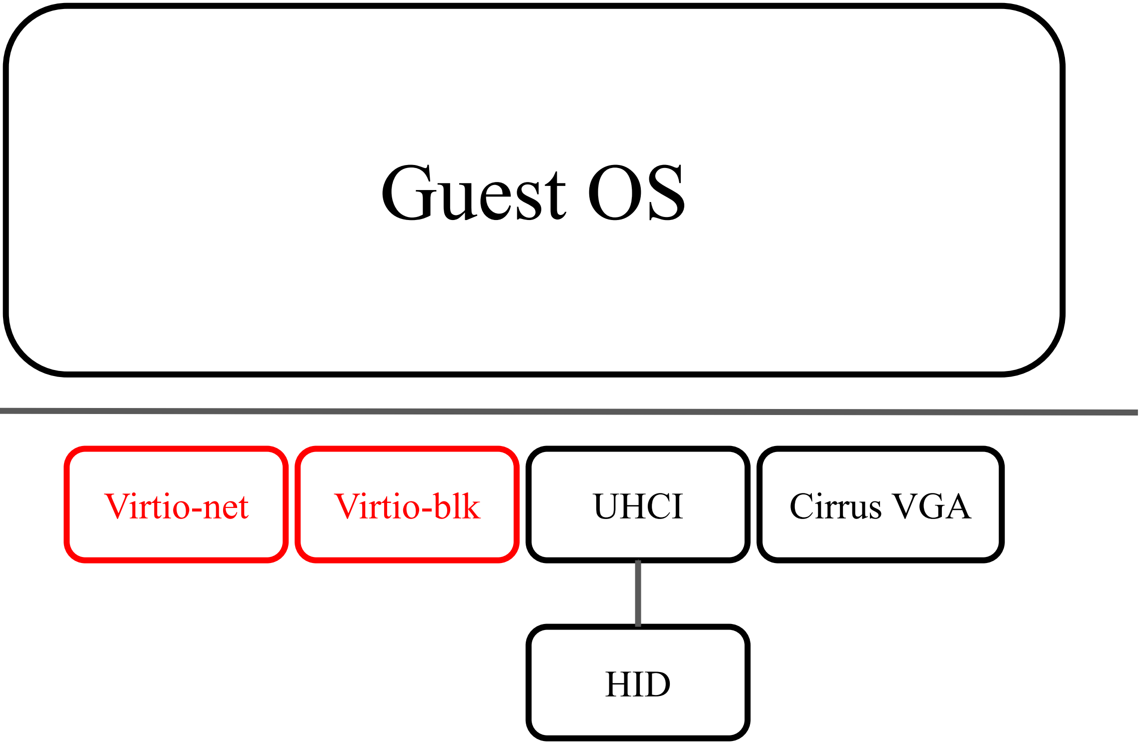


## Attack Surface of A Libvirt VM



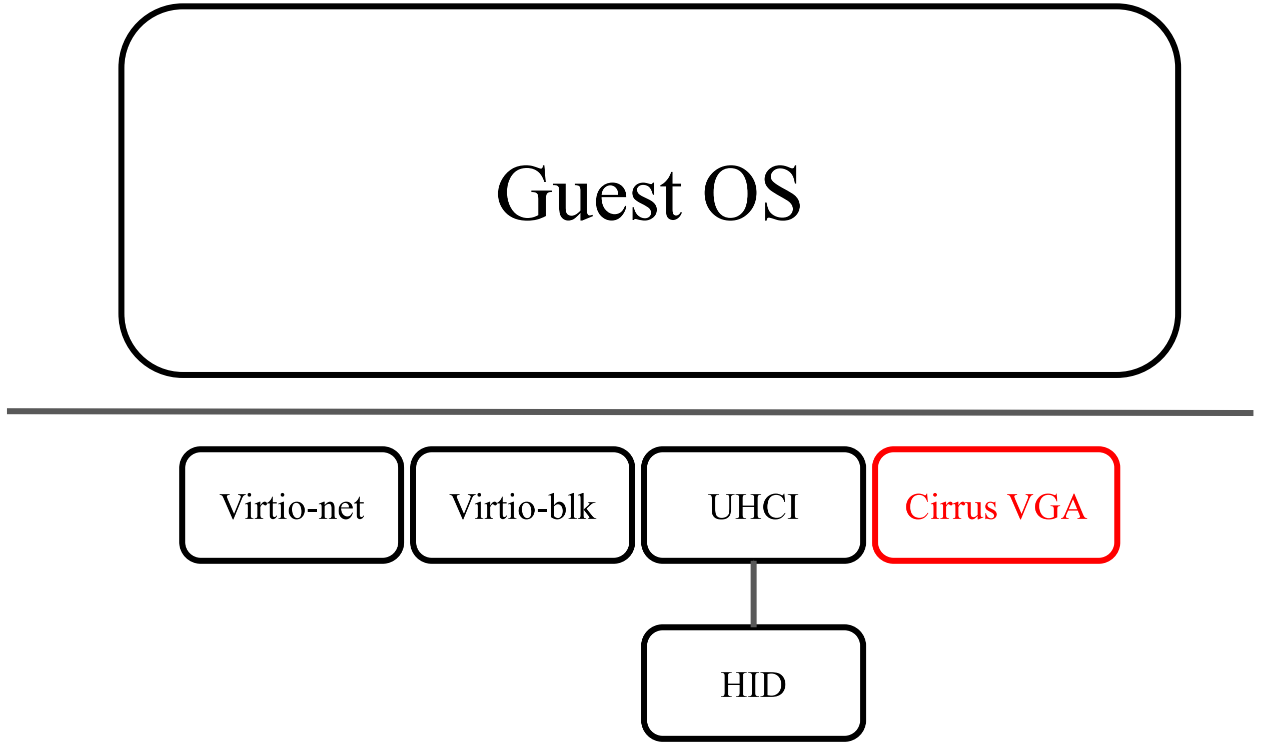
# Attack Surface of A Libvirt VM

- Virtio
  - Simple
  - Few code
  - Few CVEs



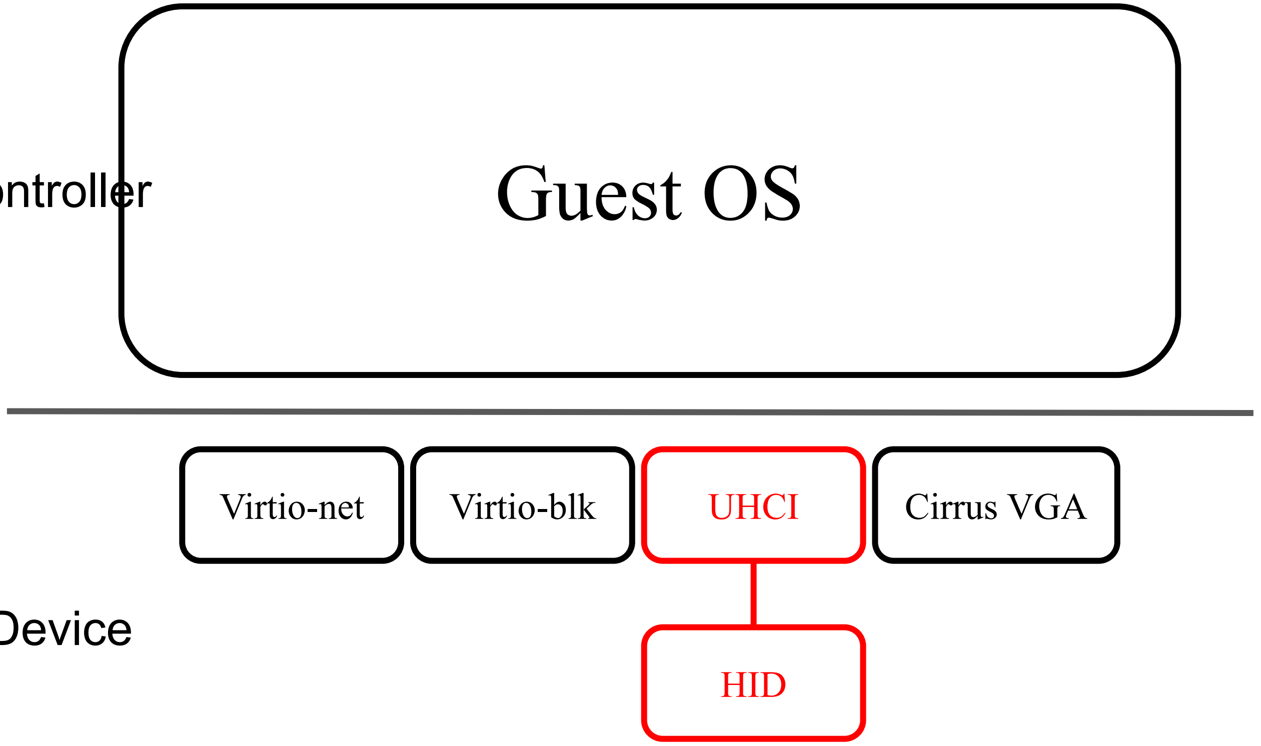
## Attack Surface of A Libvirt VM

- Cirrus VGA
  - Many CVEs
  - Hard to exploit



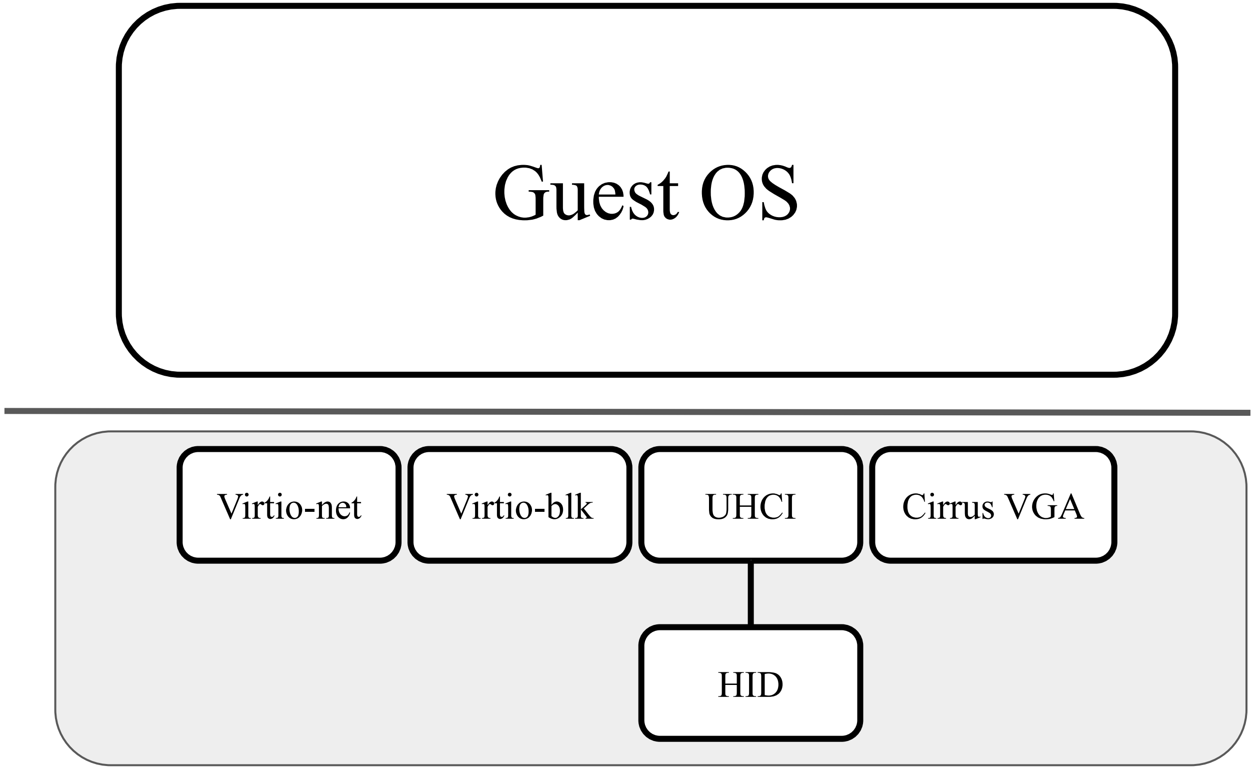
# Attack Surface of A Libvirt VM

- UHCI
  - Universal Host Controller Interface
  - USB 1.0
- HID
  - Human Interface Device
  - mouse/keyboard



# Why hard to escape from the public cloud?

- Lack of good vulnerabilities
- Lack of further information



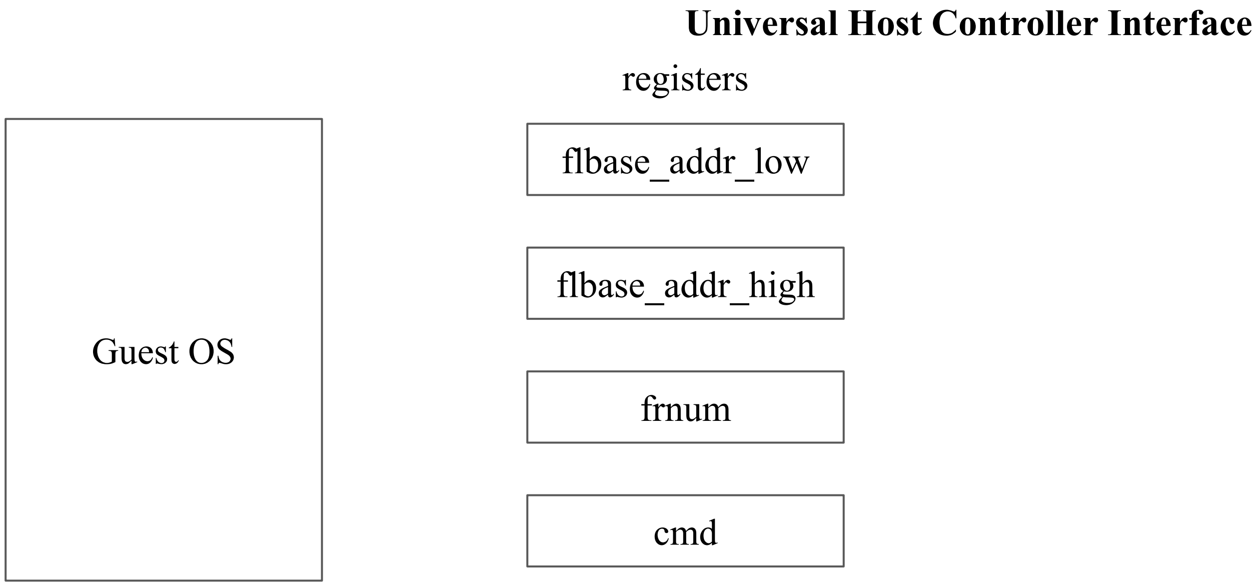
长亭科技
CHAITIN

## CVE-2020-14364

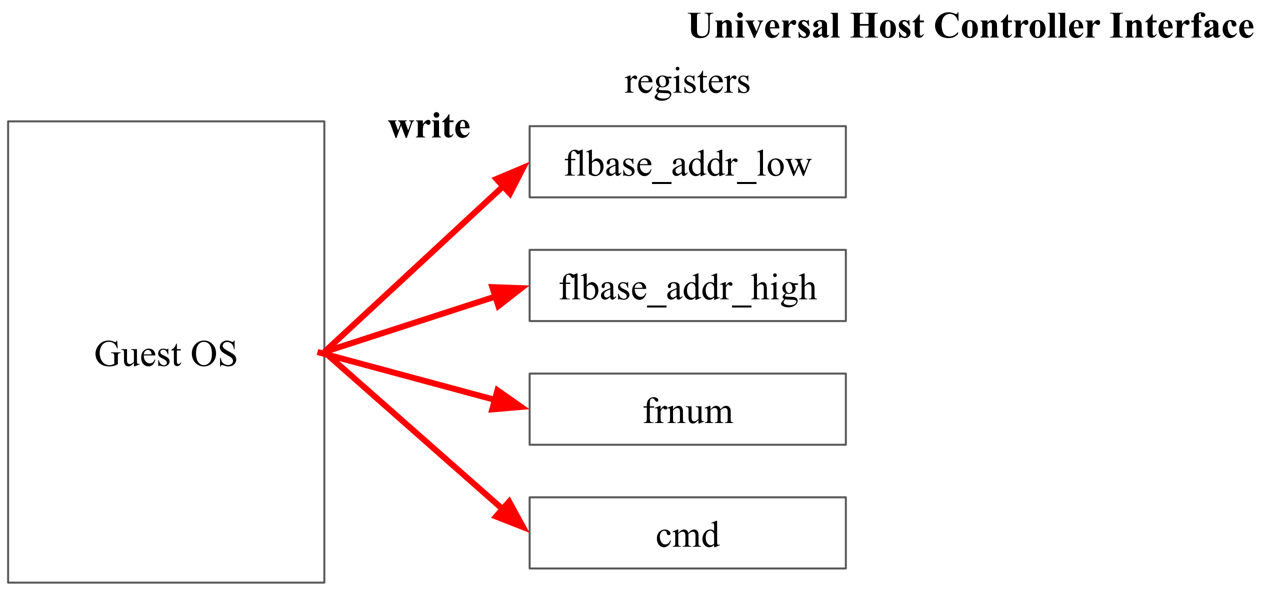
## CVE-2020-14364 Timeline

- Reported at 2020.8.13
- Redhat fixed it and disclosed it at 2020.8.24

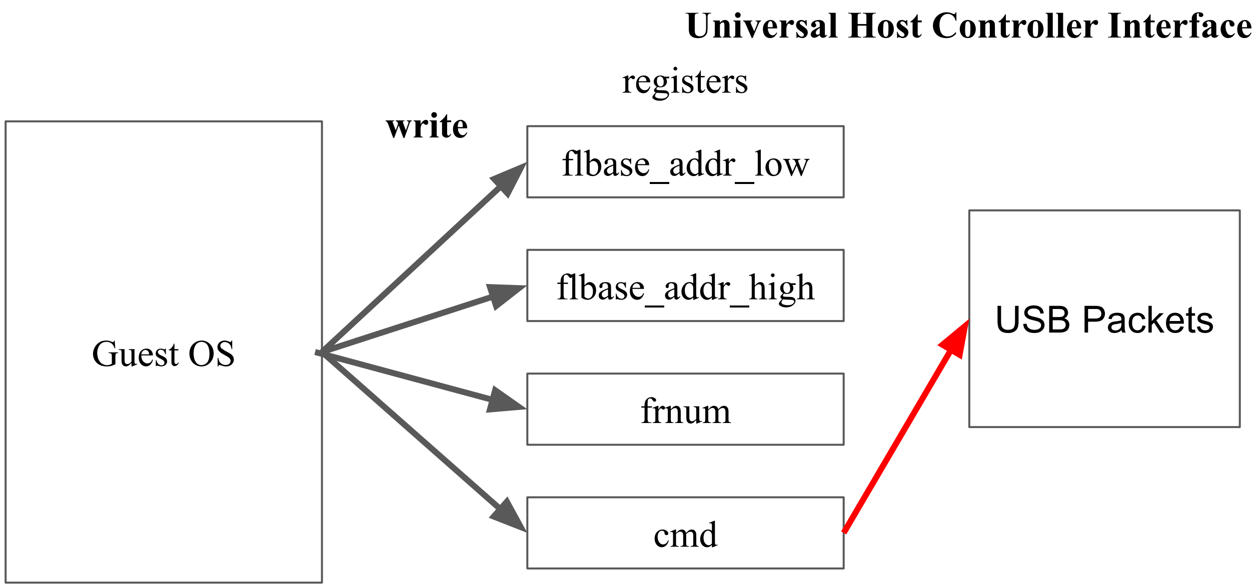
# How Does Guest OS send usb packets ?



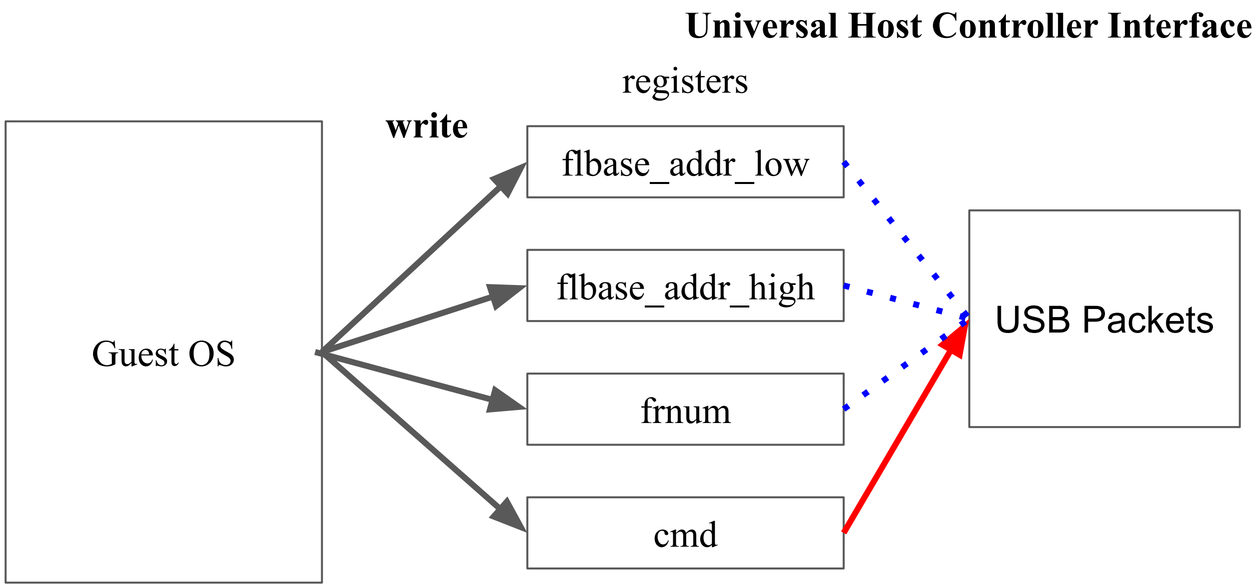
# How Does Guest OS send usb packets ?



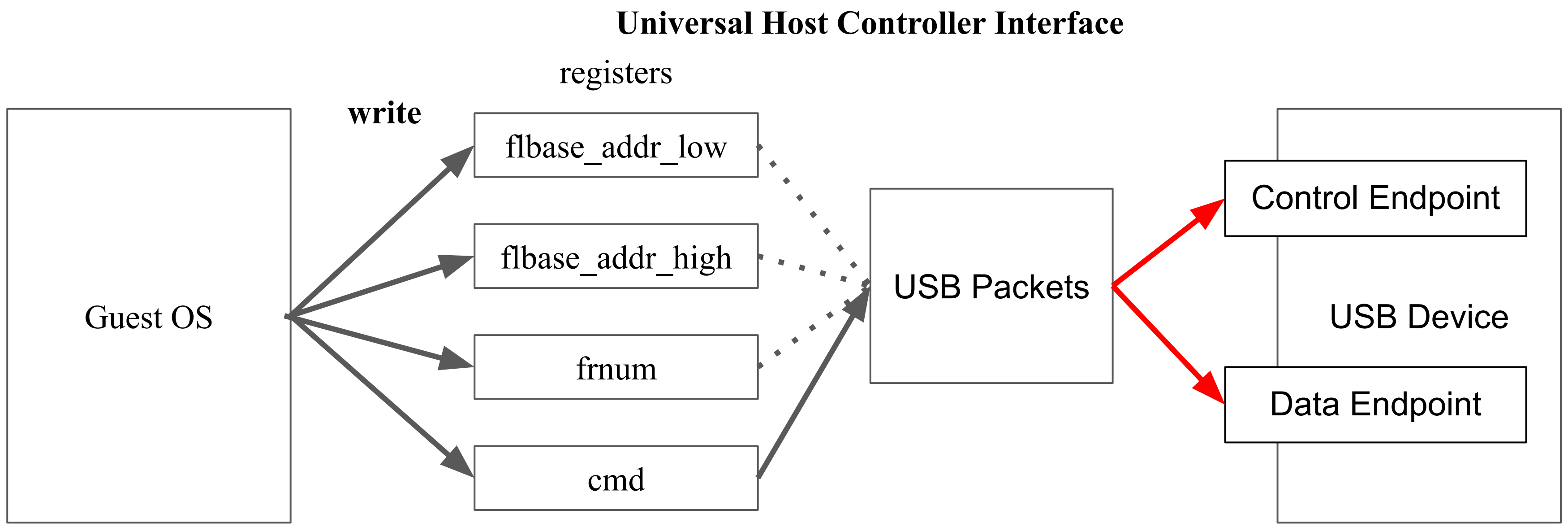
## How Does Guest OS send usb packets ?



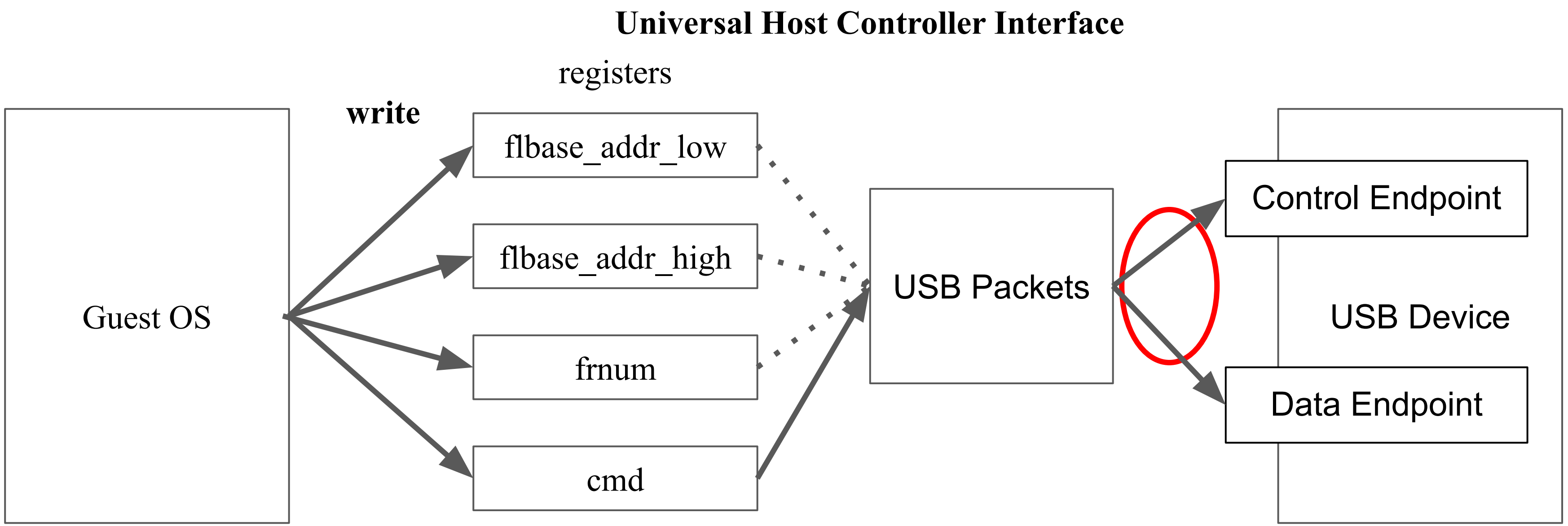
# How Does Guest OS send usb packets ?



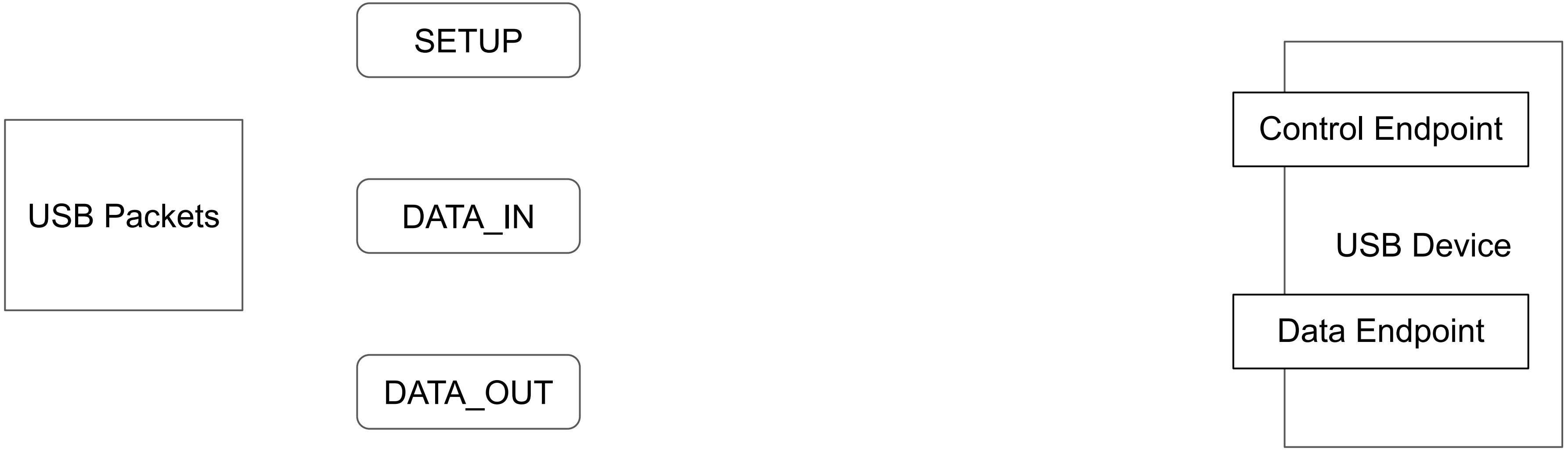
# How Does Guest OS send usb packets ?



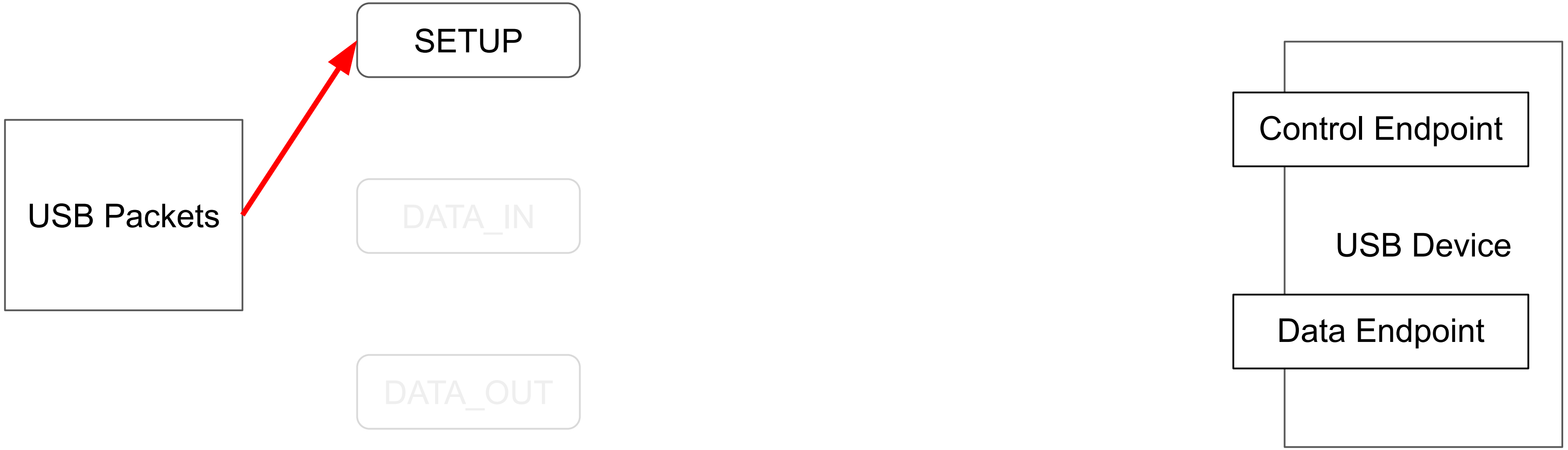
# How Does Guest OS send usb packets ?



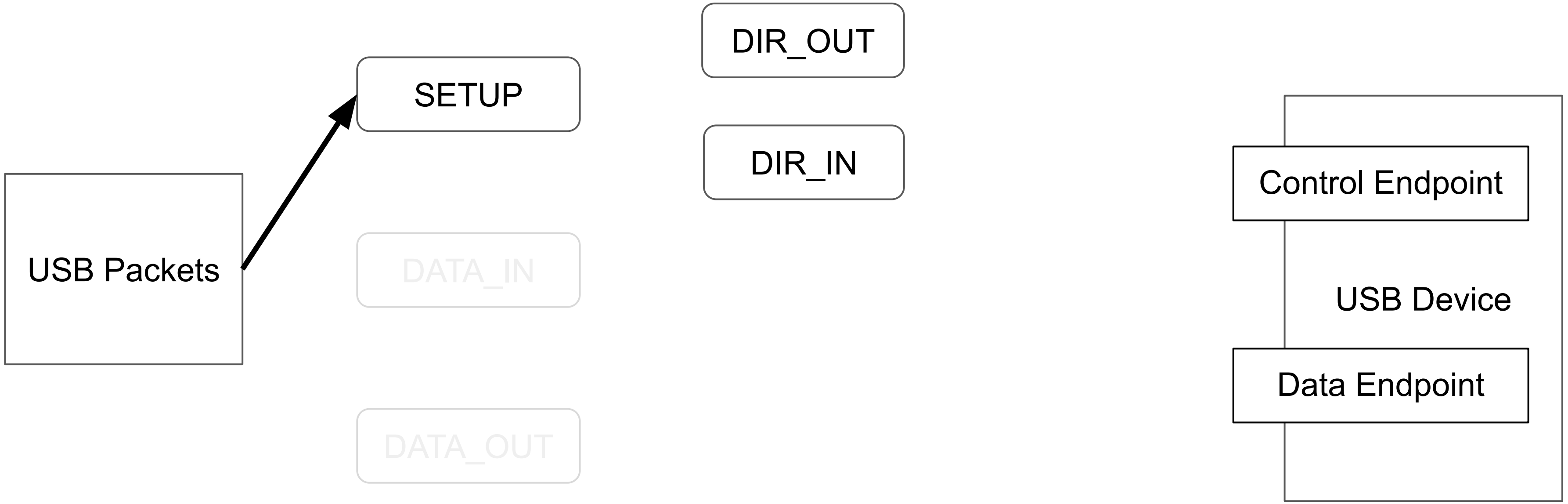
# How Do Qemu transfer USB packets?



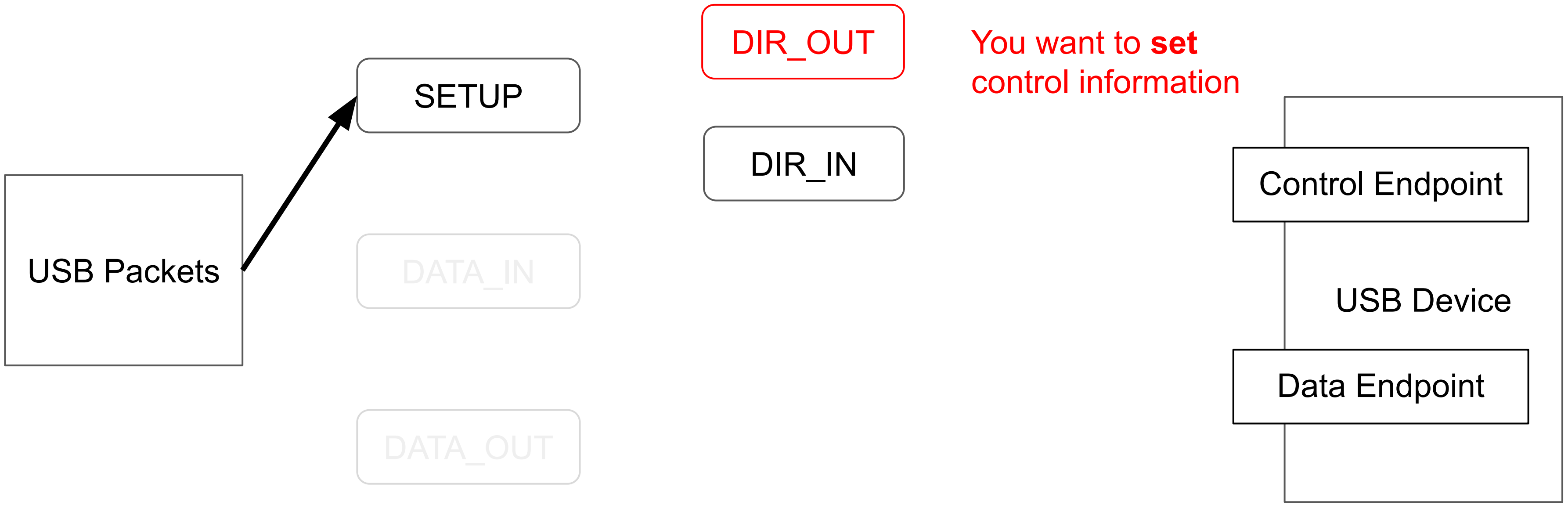
## How Do Qemu transfer USB packets?



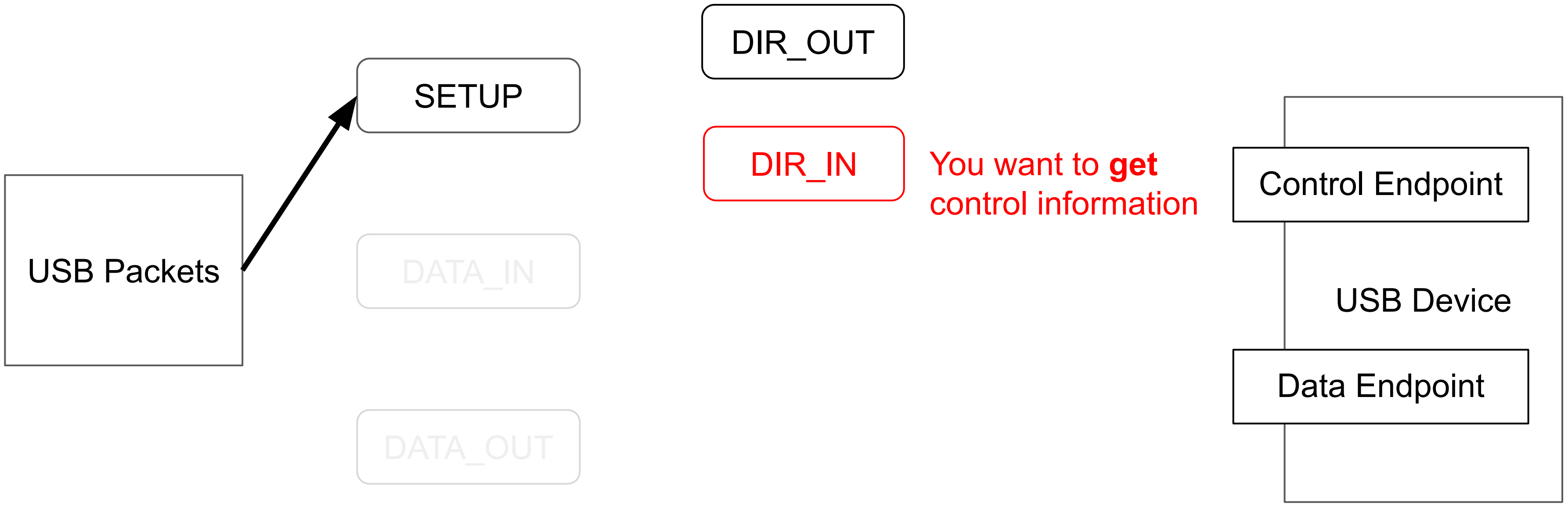
# How Do Qemu transfer USB packets?



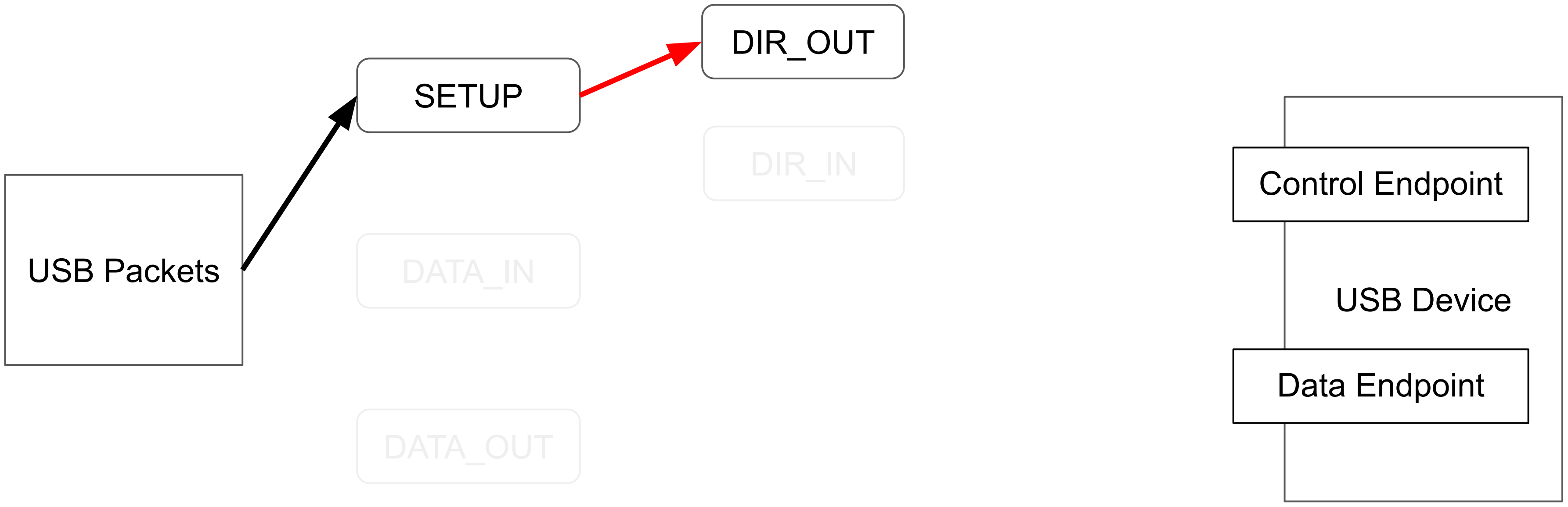
# How Do Qemu transfer USB packets?



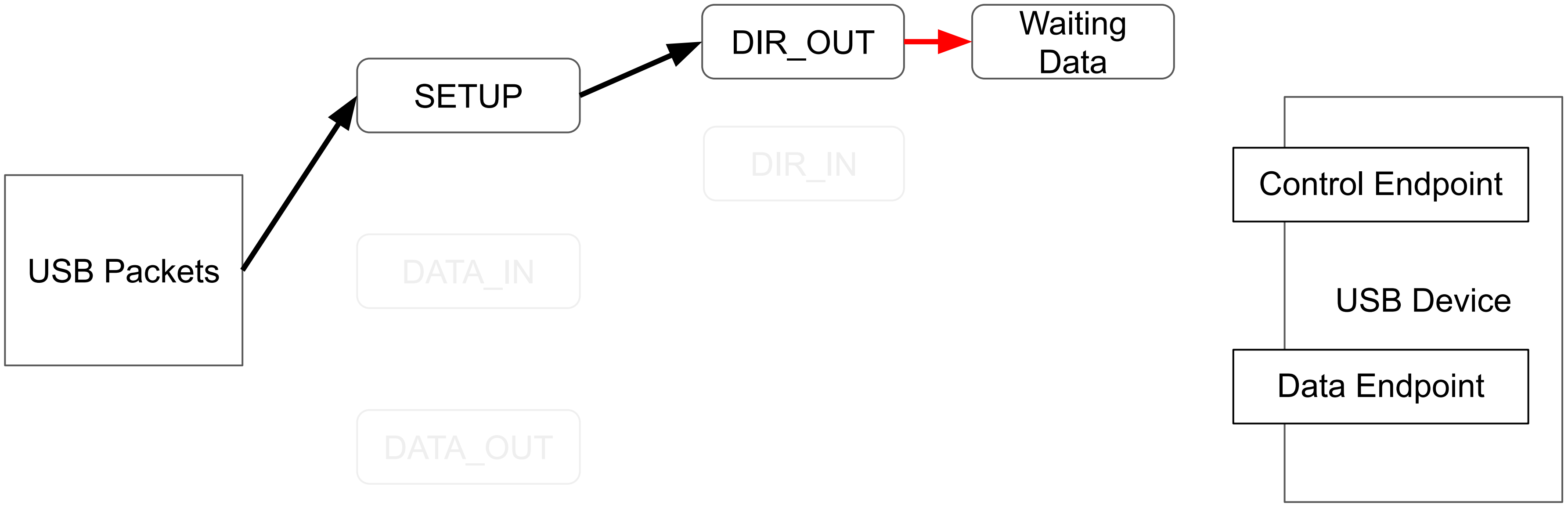
# How Do Qemu transfer USB packets?



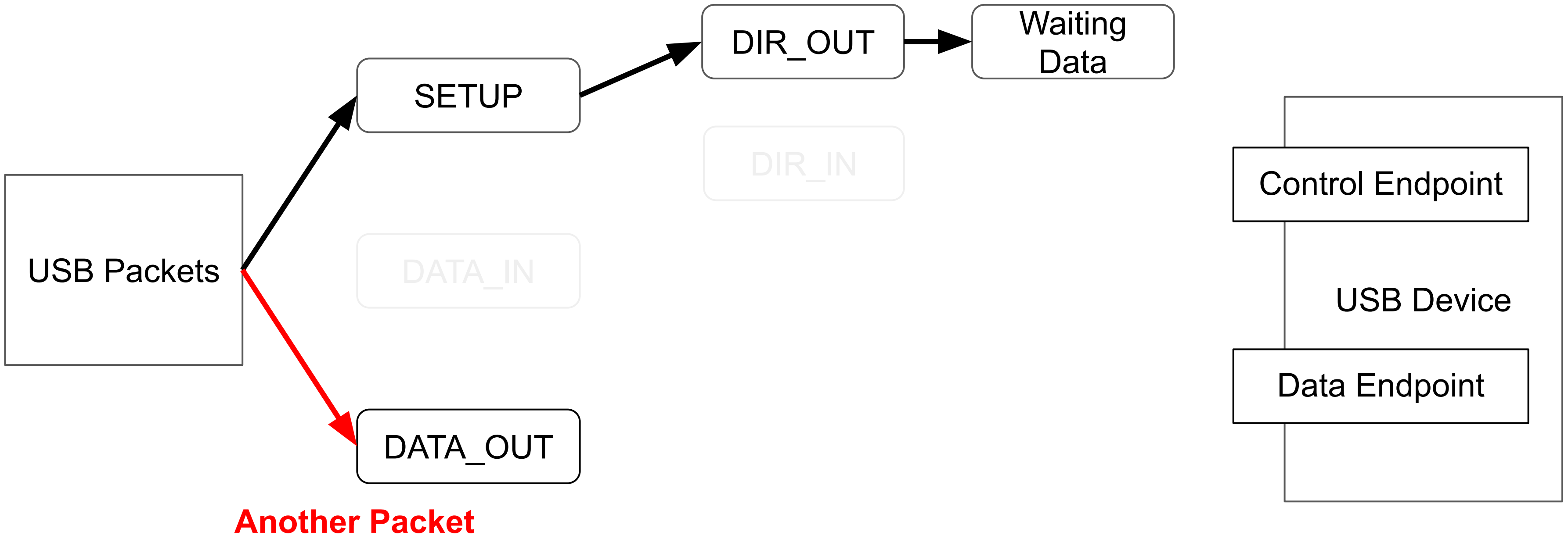
## How Do Qemu transfer USB packets?



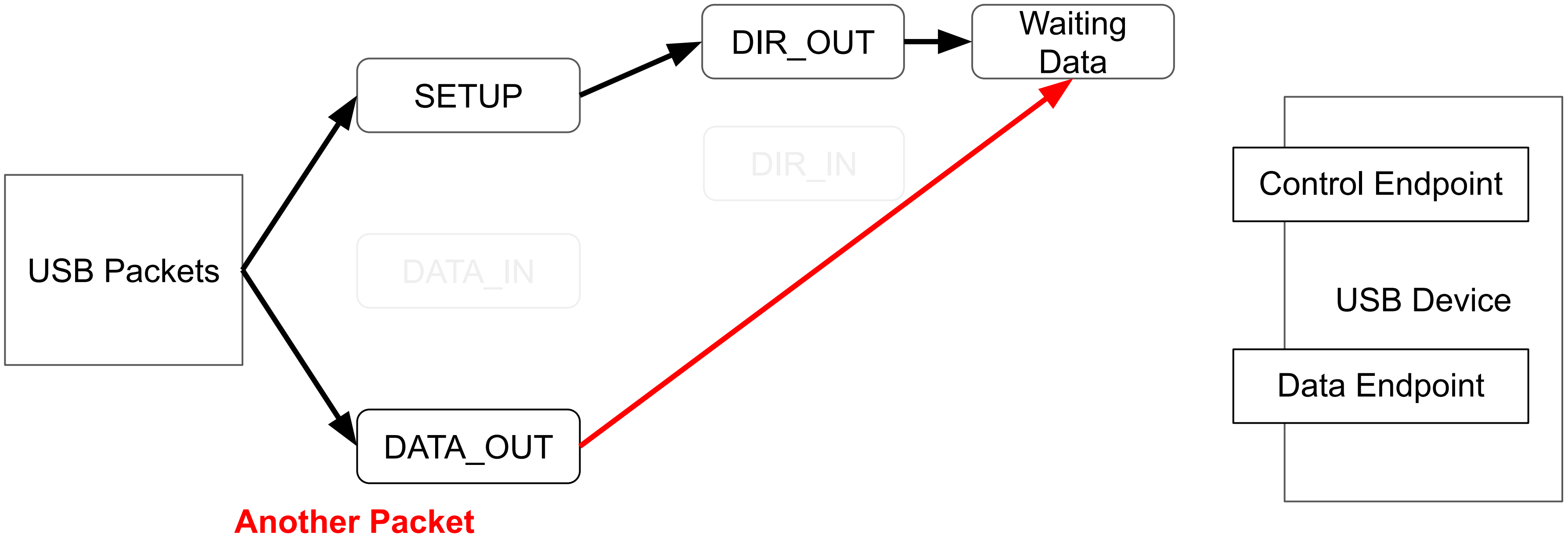
# How Do Qemu transfer USB packets?



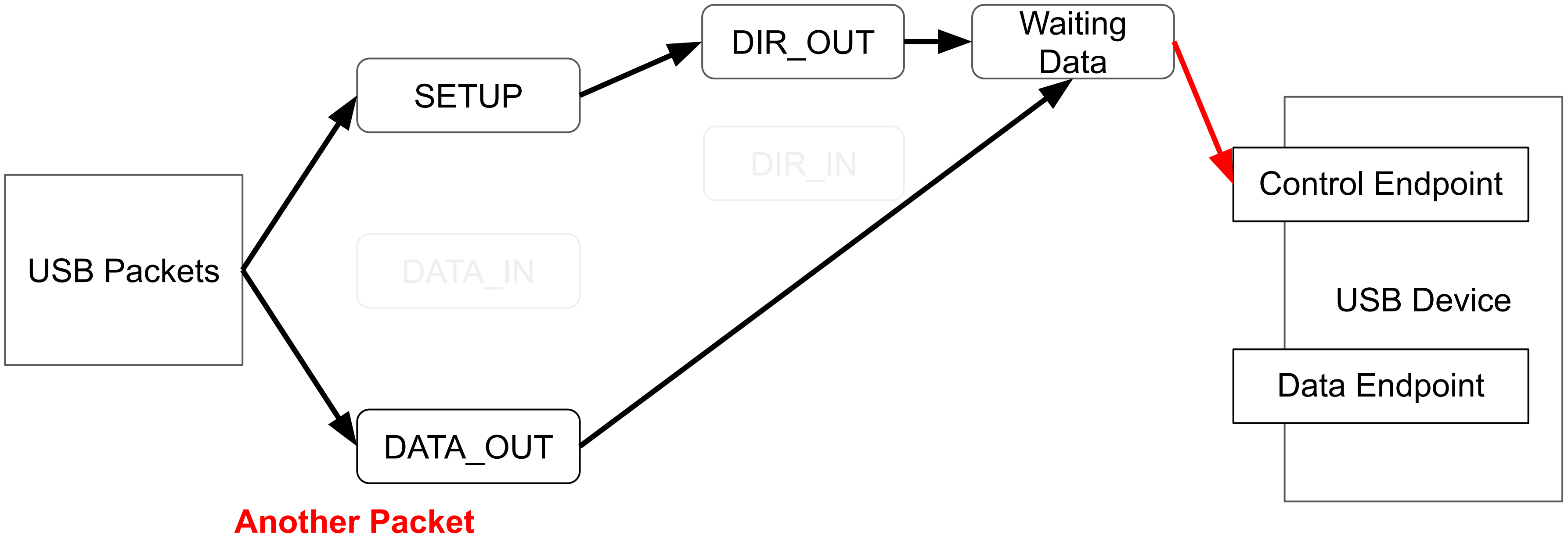
# How Do Qemu transfer USB packets?



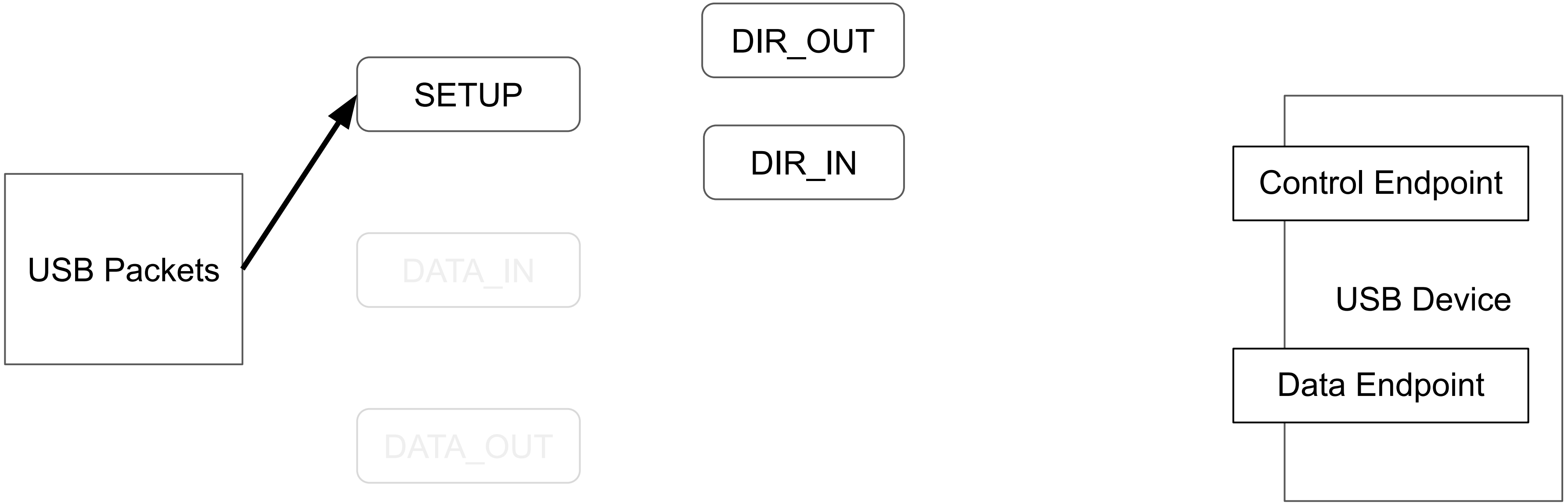
# How Do Qemu transfer USB packets?



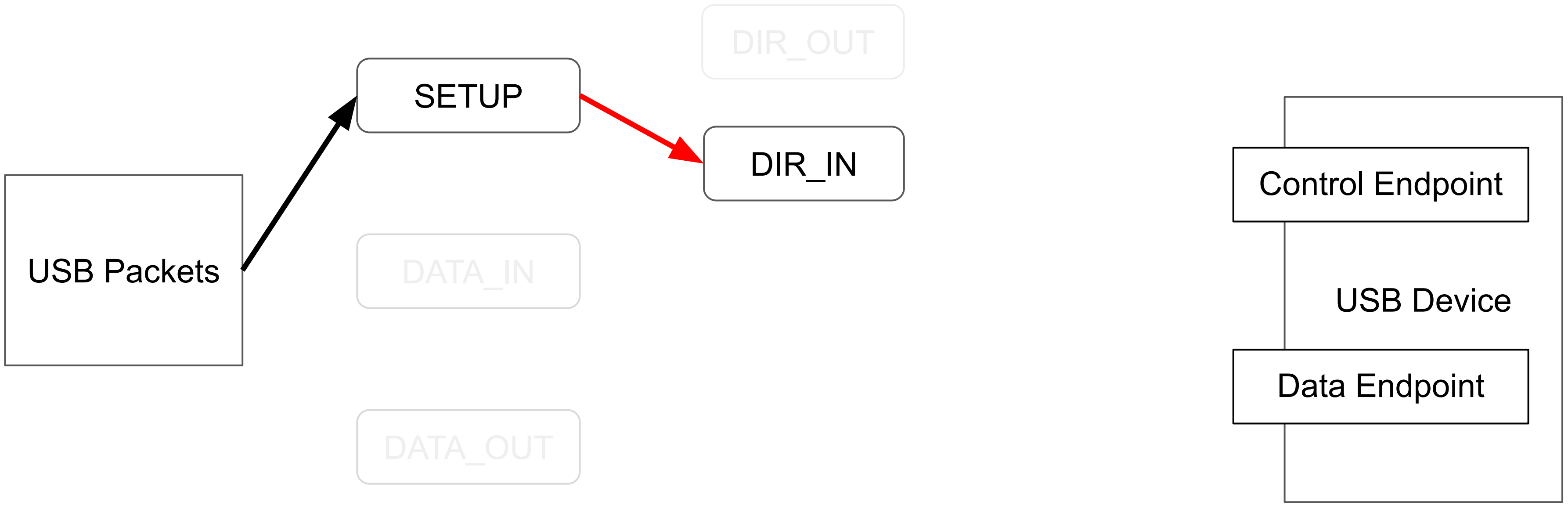
# How Do Qemu transfer USB packets?



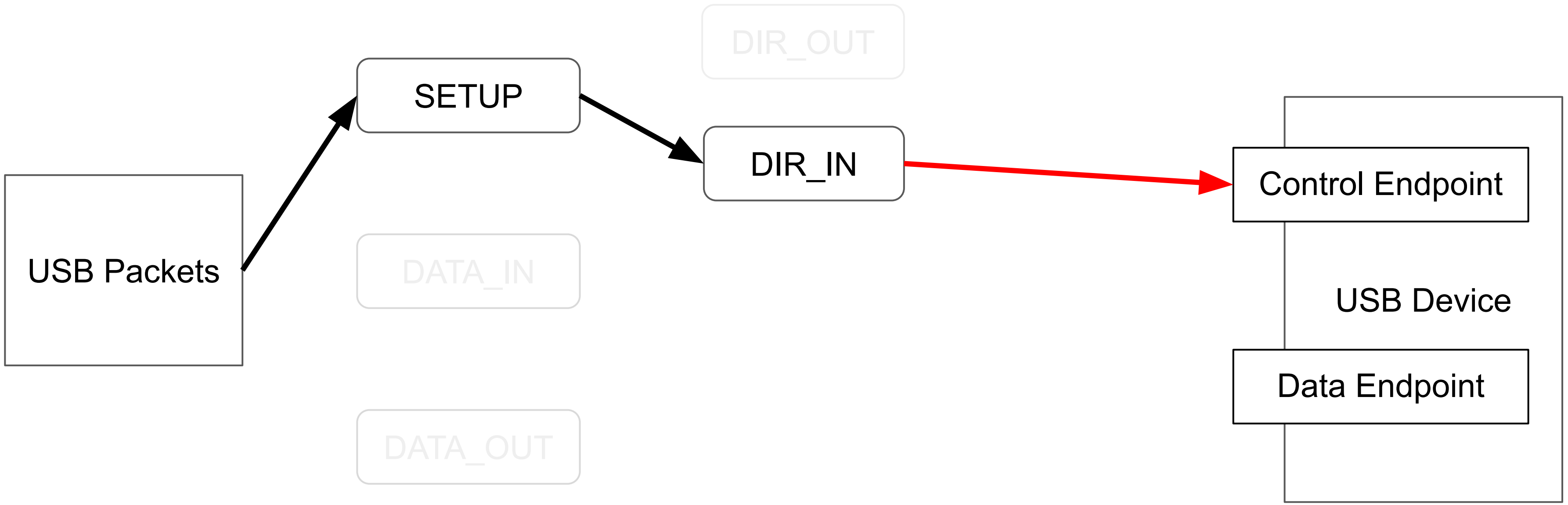
# How Do Qemu transfer USB packets?



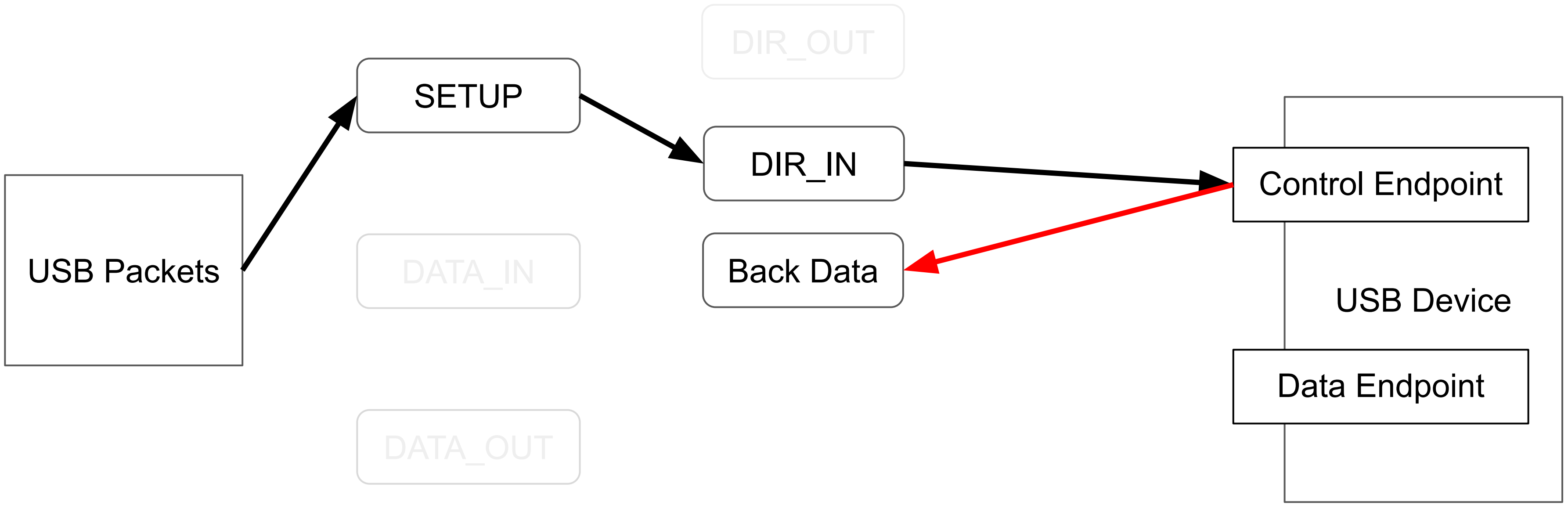
# How Do Qemu transfer USB packets?



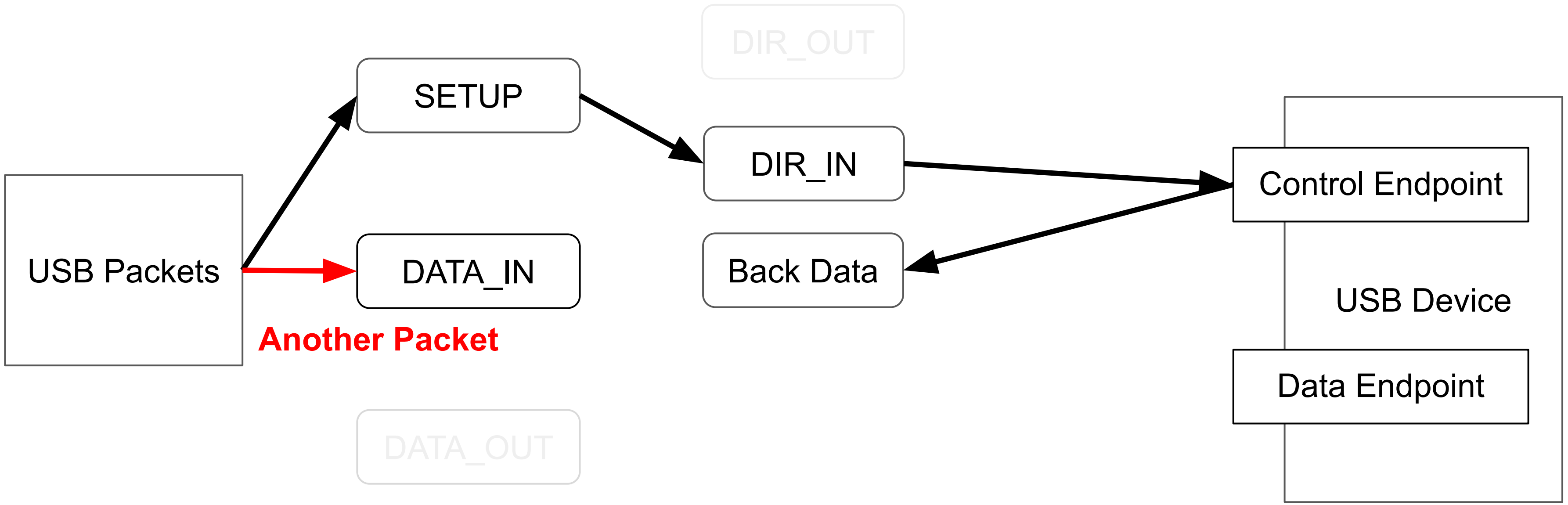
# How Do Qemu transfer USB packets?



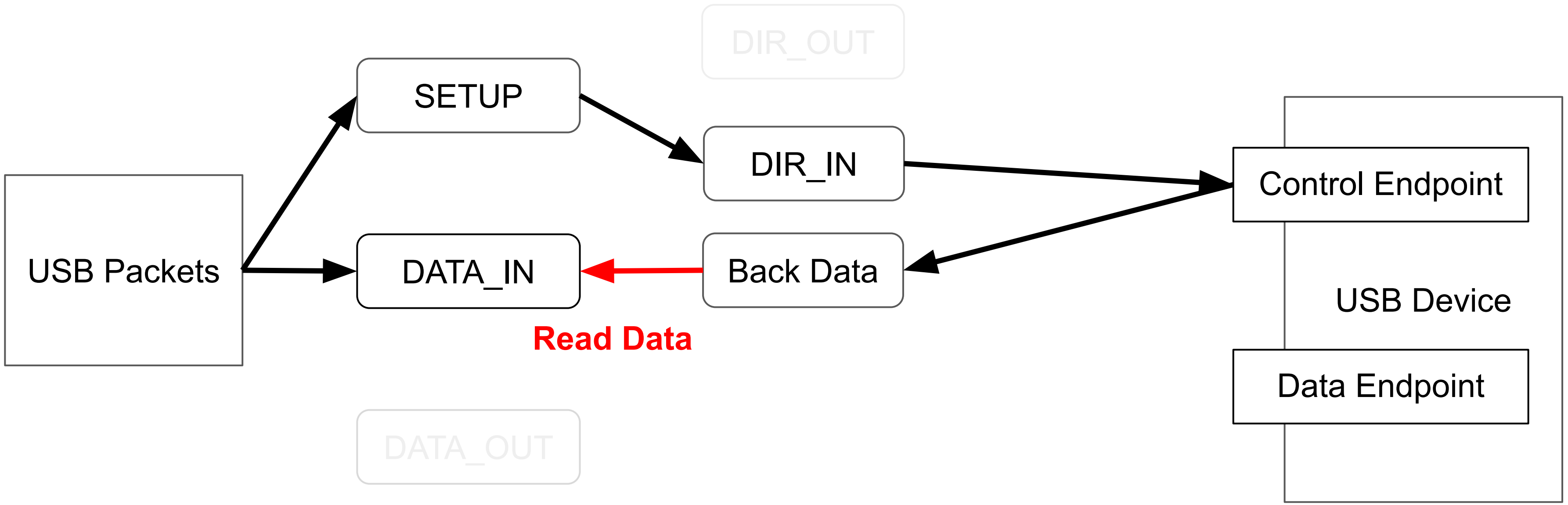
# How Do Qemu transfer USB packets?



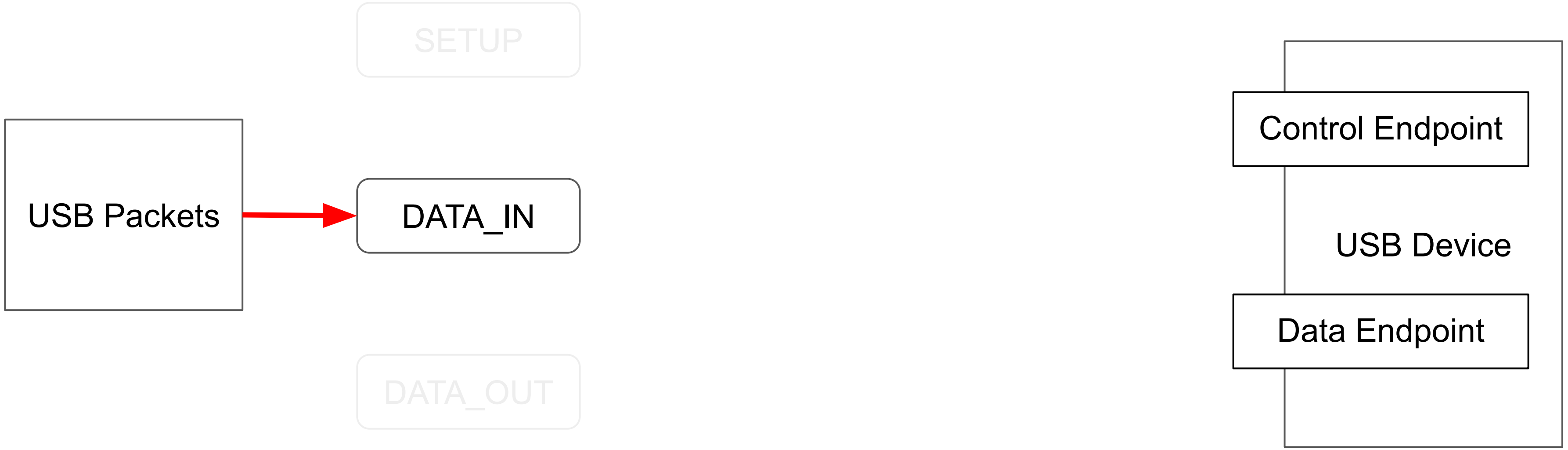
# How Do Qemu transfer USB packets?



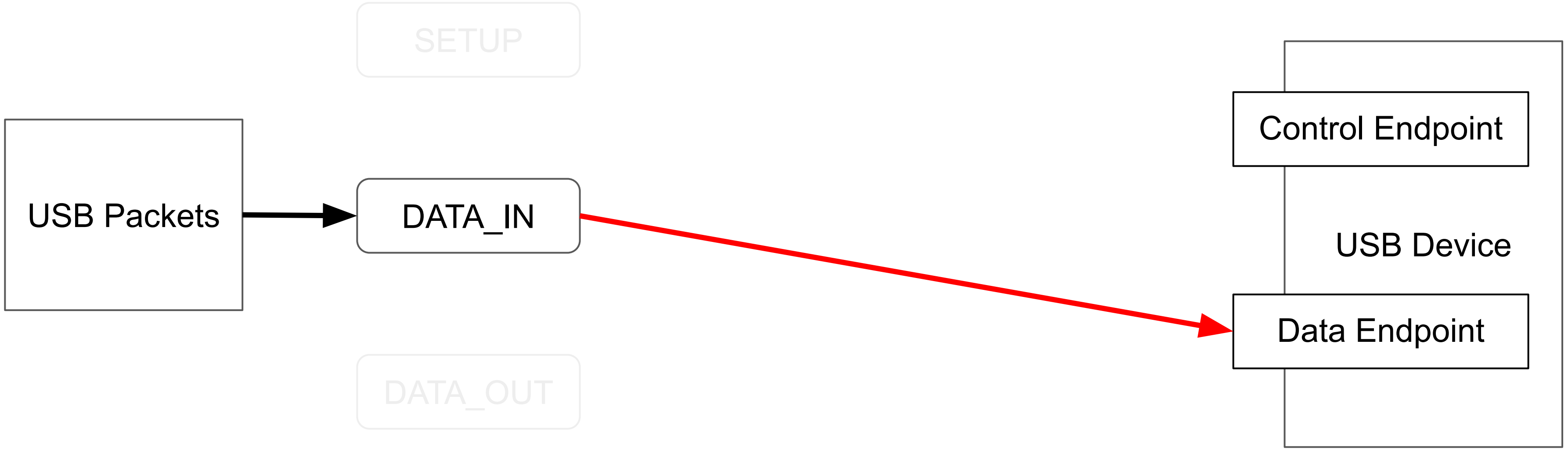
# How Qemu transfer USB packets?



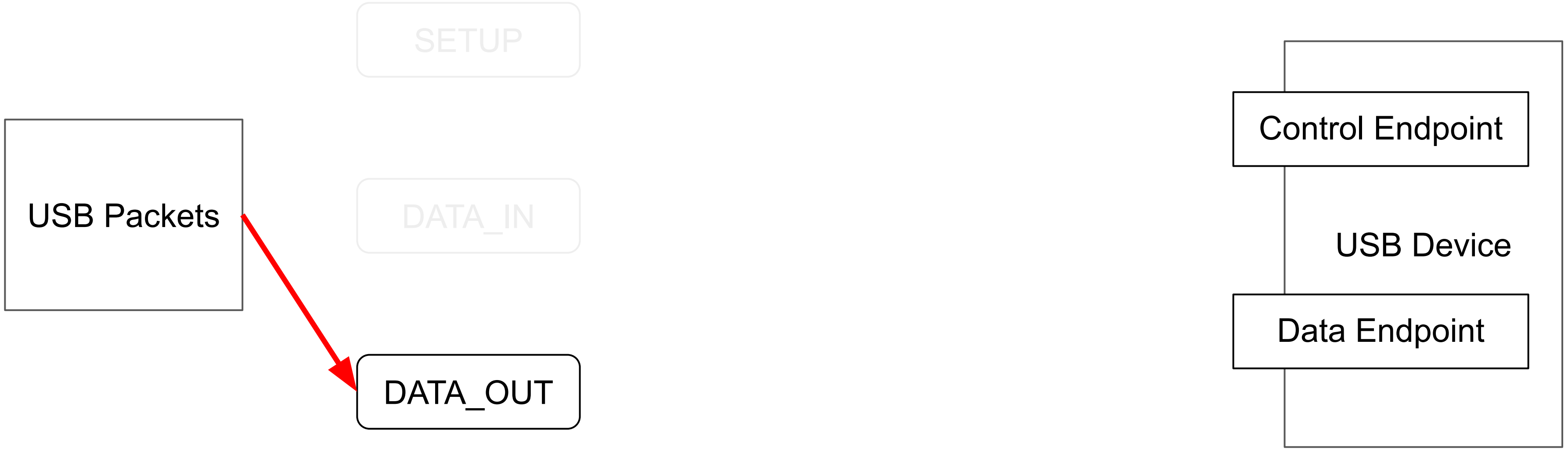
# How Do Qemu transfer USB packets?



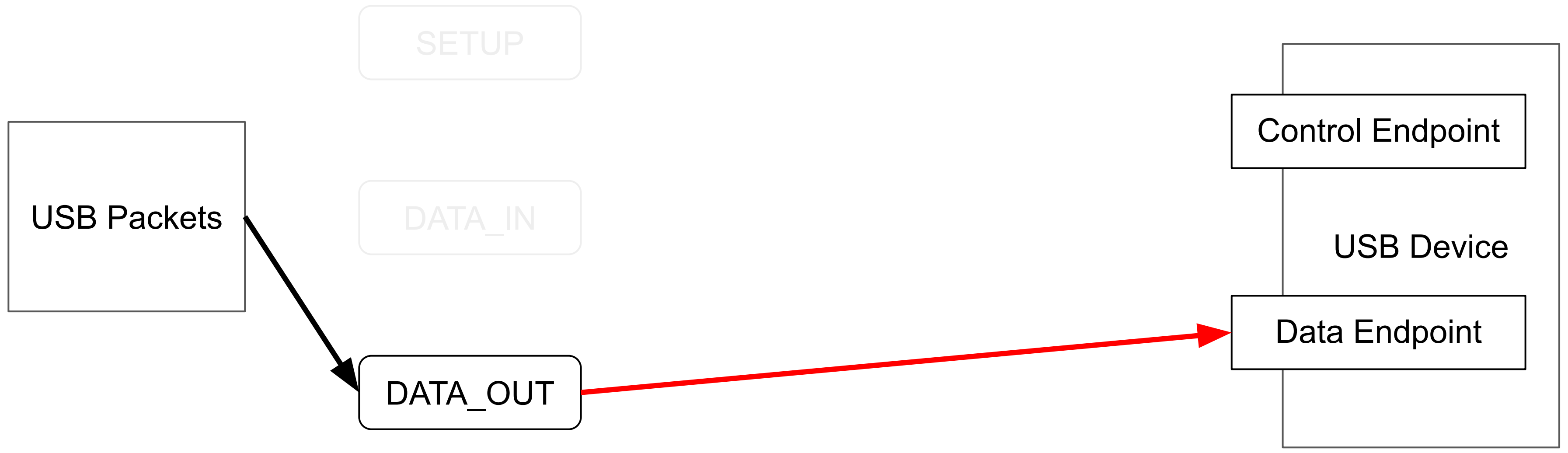
# How Do Qemu transfer USB packets?



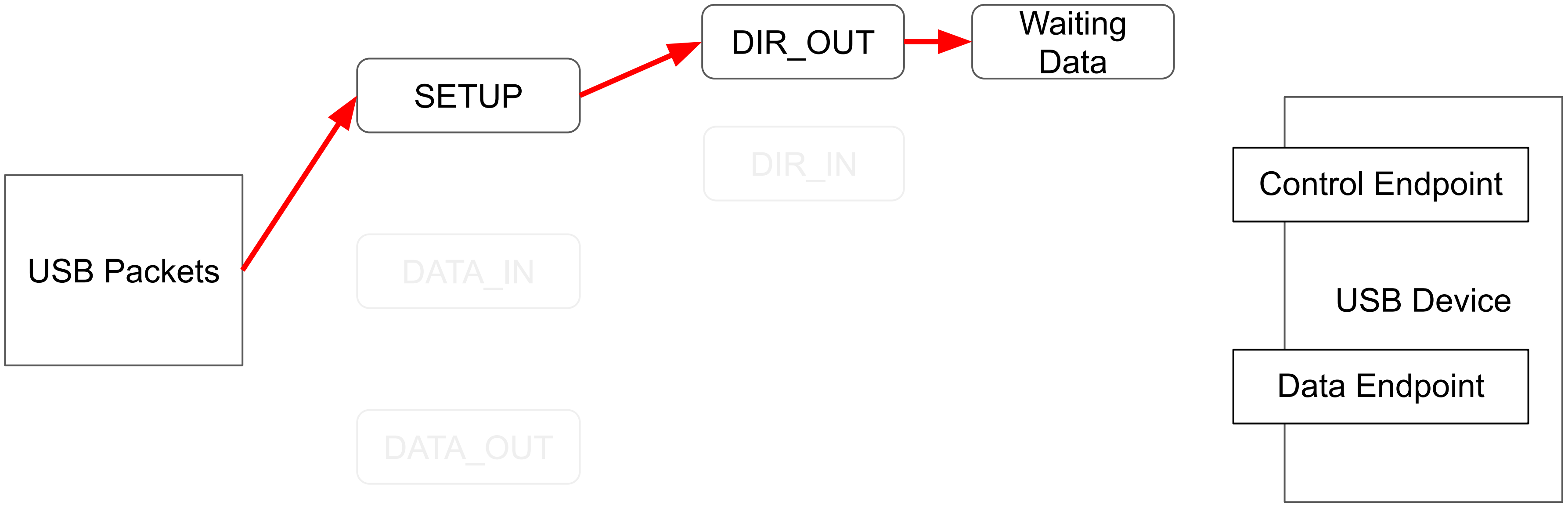
# How Do Qemu transfer USB packets?



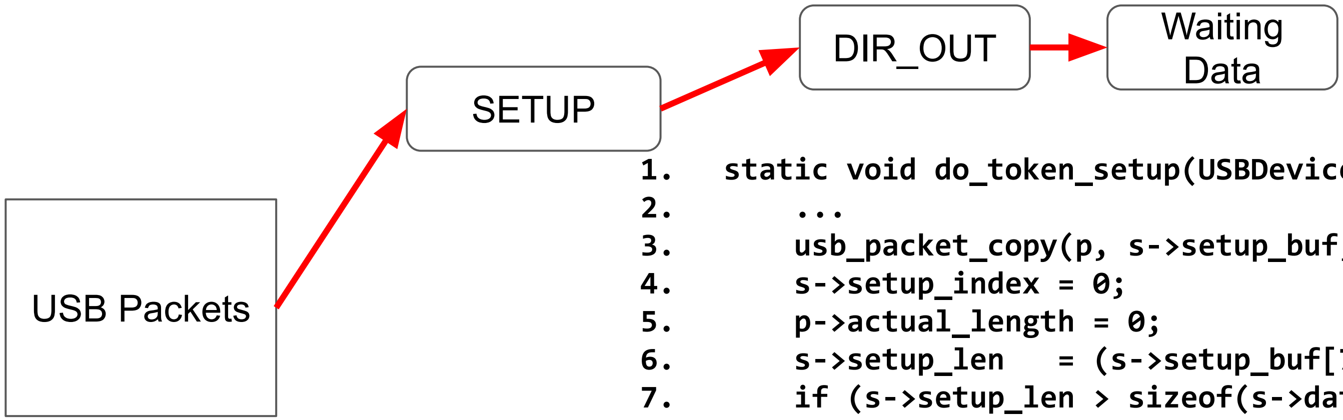
## How Do Qemu transfer USB packets?



# CVE-2020-14364

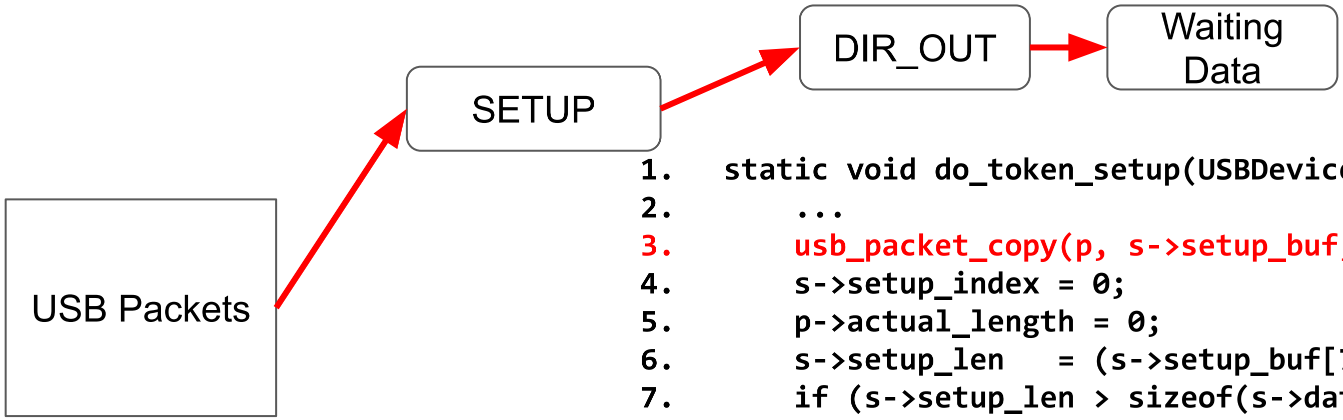


# CVE-2020-14364



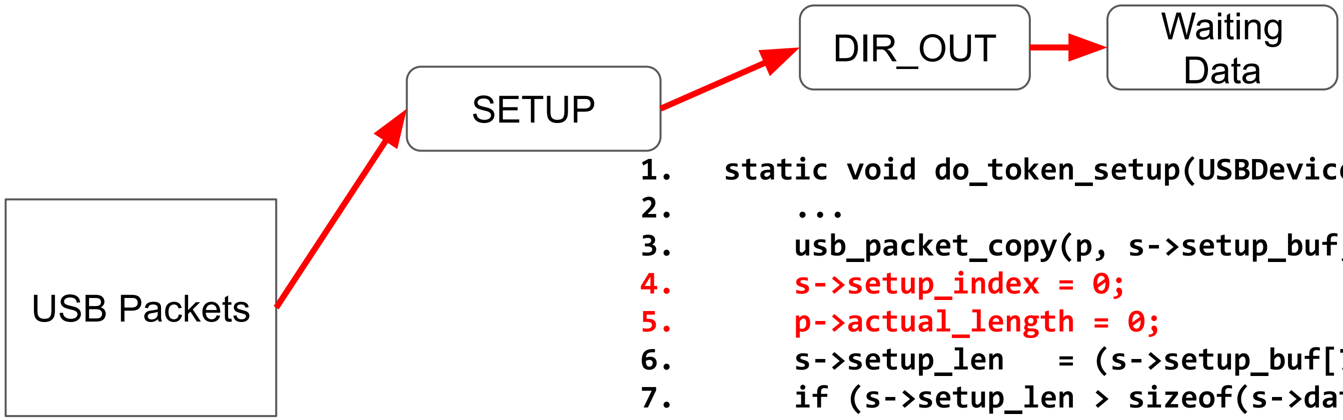
```c
static void do_token_setup(USBDevice *s, USBPacket *p){
    ...
    usb_packet_copy(p, s->setup_buf, p->iov.size);
    s->setup_index = 0;
    p->actual_length = 0;
    s->setup_len   = (s->setup_buf[7] << 8) | s->setup_buf[6];
    if (s->setup_len > sizeof(s->data_buf)) {
        fprintf(stderr,
                "usb_generic_handle_packet: ctrl buffer too small (%d > %zu)\n",
                s->setup_len, sizeof(s->data_buf));
        p->status = USB_RET_STALL;
        return;
    }
    if (s->setup_buf[0] & USB_DIR_IN) {
    ...
    }else{
        s->setup_state = SETUP_STATE_DATA;
```

# CVE-2020-14364



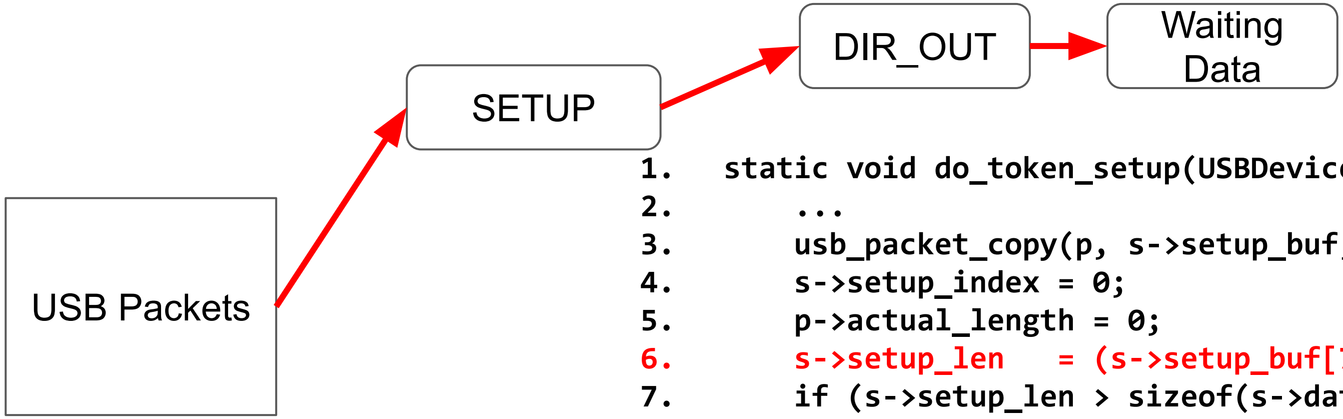
```
static void do_token_setup(USBDevice *s, USBPacket *p){
    ...
    usb_packet_copy(p, s->setup_buf, p->iov.size);
    s->setup_index = 0;
    p->actual_length = 0;
    s->setup_len   = (s->setup_buf[7] << 8) | s->setup_buf[6];
    if (s->setup_len > sizeof(s->data_buf)) {
        fprintf(stderr,
                "usb_generic_handle_packet: ctrl buffer too small (%d > %zu)\n",
                s->setup_len, sizeof(s->data_buf));
        p->status = USB_RET_STALL;
        return;
    }
    if (s->setup_buf[0] & USB_DIR_IN) {
    ...
    }else{
        s->setup_state = SETUP_STATE_DATA;
```

# CVE-2020-14364



```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_RET_STALL;
        return;
    }
    if (s->setup_buf[0] & USB_DIR_IN) {
    ...
    }else{
        s->setup_state = SETUP_STATE_DATA;
```

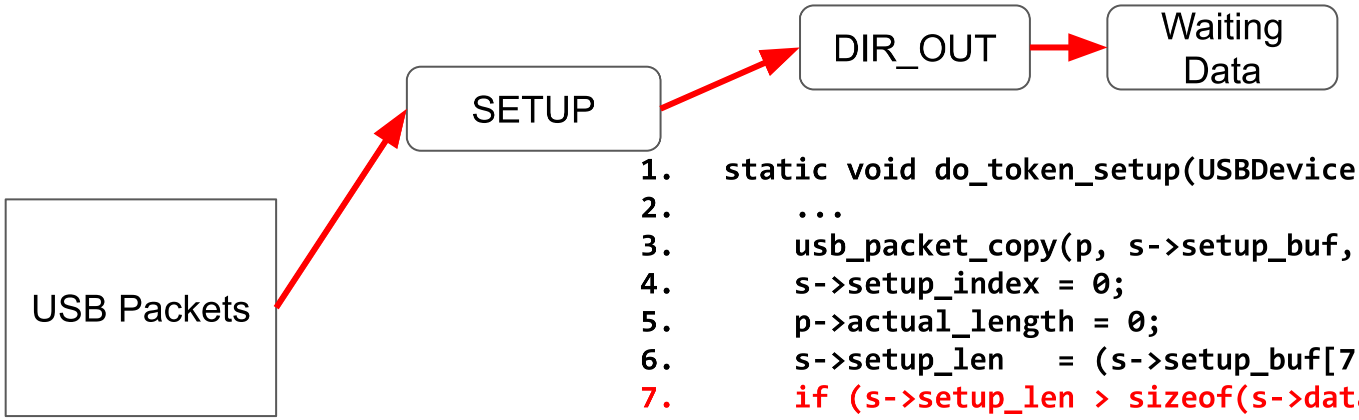
# CVE-2020-14364



Get the length of setting data

```
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_RET_STALL;
        return;
    }
    if (s->setup_buf[0] & USB_DIR_IN) {
    ...
    }else{
        s->setup_state = SETUP_STATE_DATA;
```

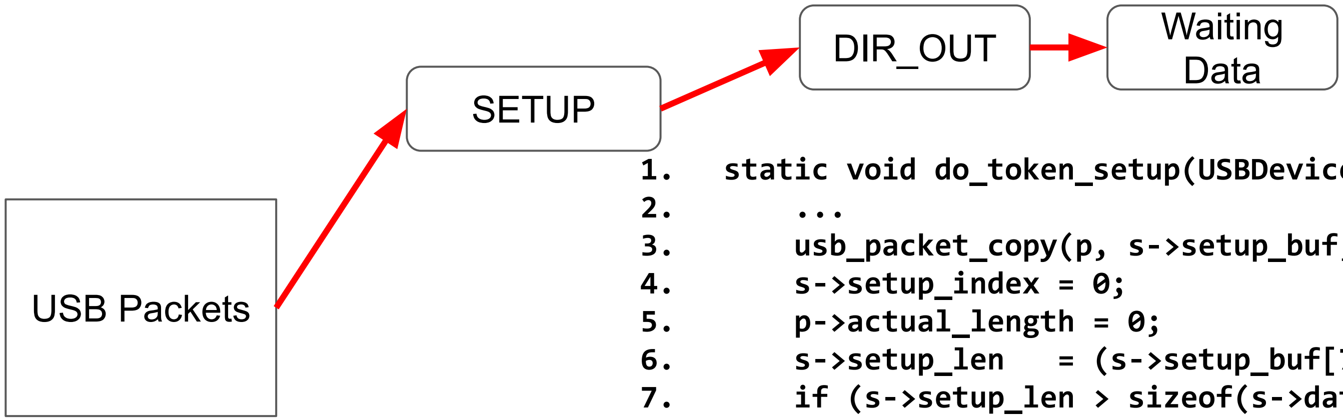
# CVE-2020-14364



```
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_RET_STALL;
        return;
    }
    if (s->setup_buf[0] & USB_DIR_IN) {
    ...
    }else{
        s->setup_state = SETUP_STATE_DATA;
```

**Check the input length if bigger than the buffer**

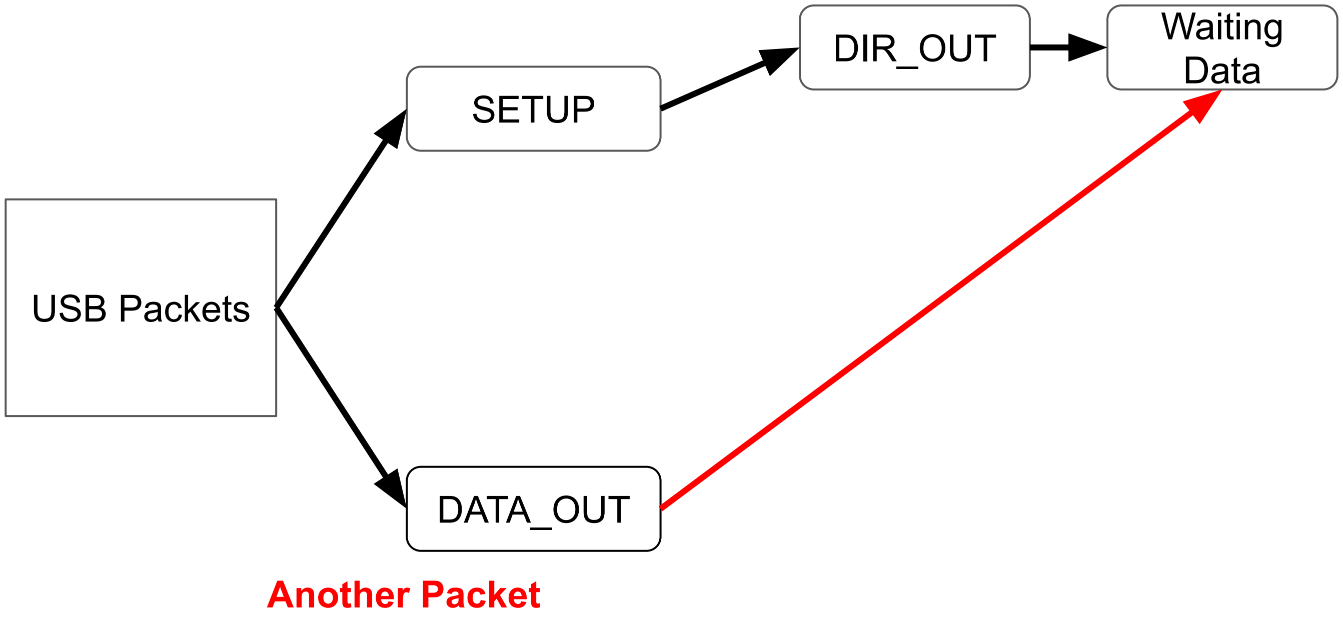
# CVE-2020-14364



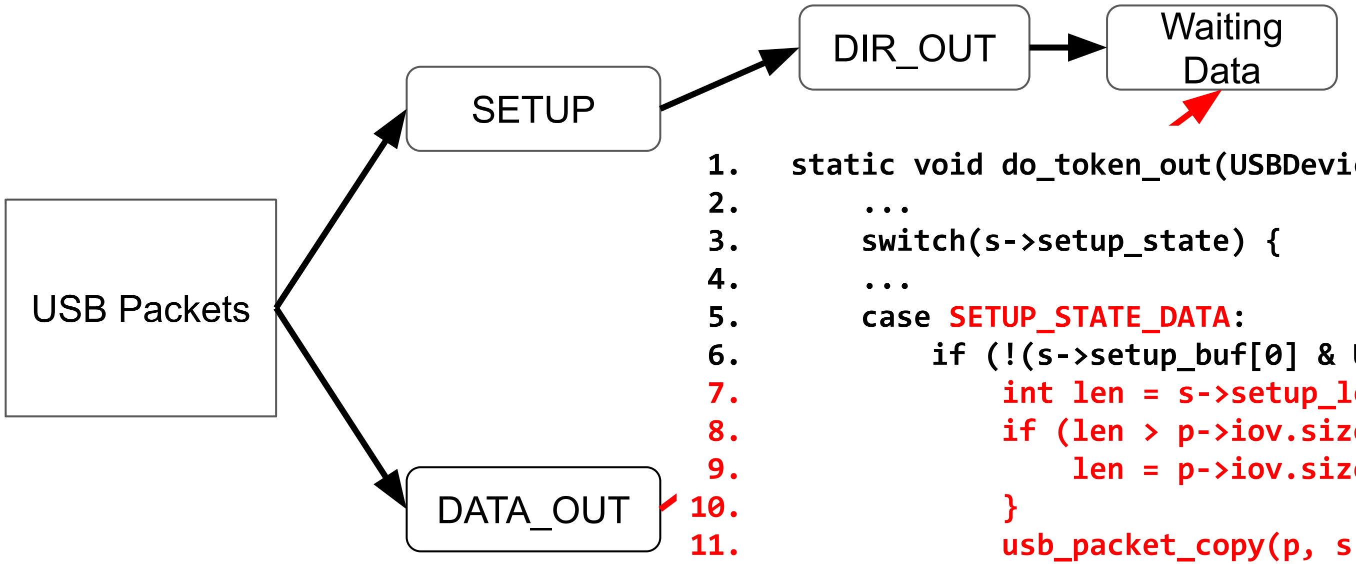
**Return if check fails without clearing s->setup_len ! ! !**

```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_RET_STALL;
        return;
    }
    if (s->setup_buf[0] & USB_DIR_IN) {
    ...
    }else{
        s->setup_state = SETUP_STATE_DATA;
```

# CVE-2020-14364



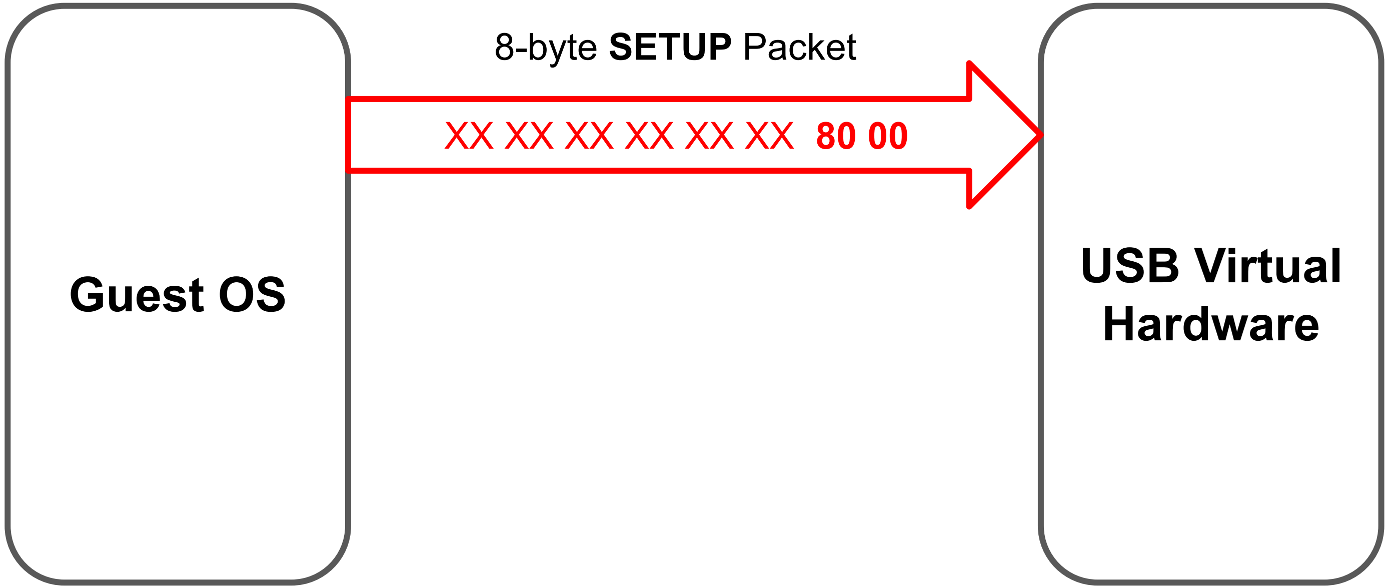
# CVE-2020-14364



```
static void do_token_out(USBDevice *s, USBPacket *p){
    ...
    switch(s->setup_state) {
    ...
    case SETUP_STATE_DATA:
        if (!(s->setup_buf[0] & USB_DIR_IN)) {
            int len = s->setup_len - s->setup_index;
            if (len > p->iov.size) {
                len = p->iov.size;
            }
            usb_packet_copy(p, s->data_buf + s->setup_index, len);
            s->setup_index += len;
            if (s->setup_index >= s->setup_len) {
                s->setup_state = SETUP_STATE_ACK;
            }
            return;
        }

```

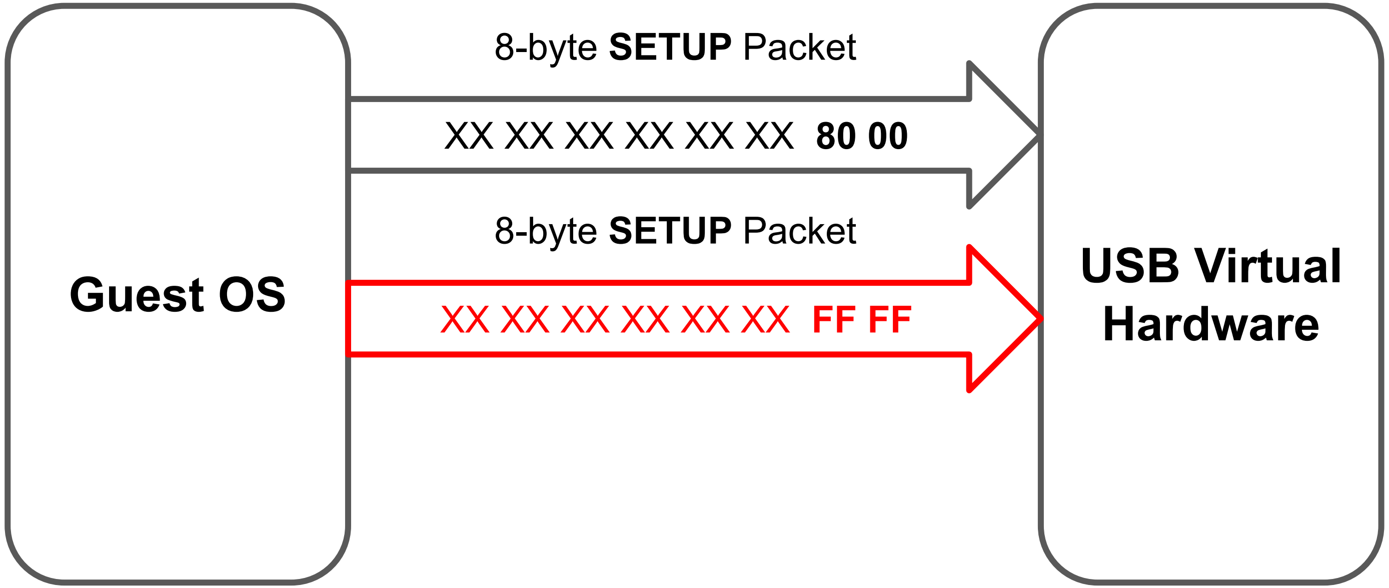
**s->setup_len will be used in copying data**

# Build A POC



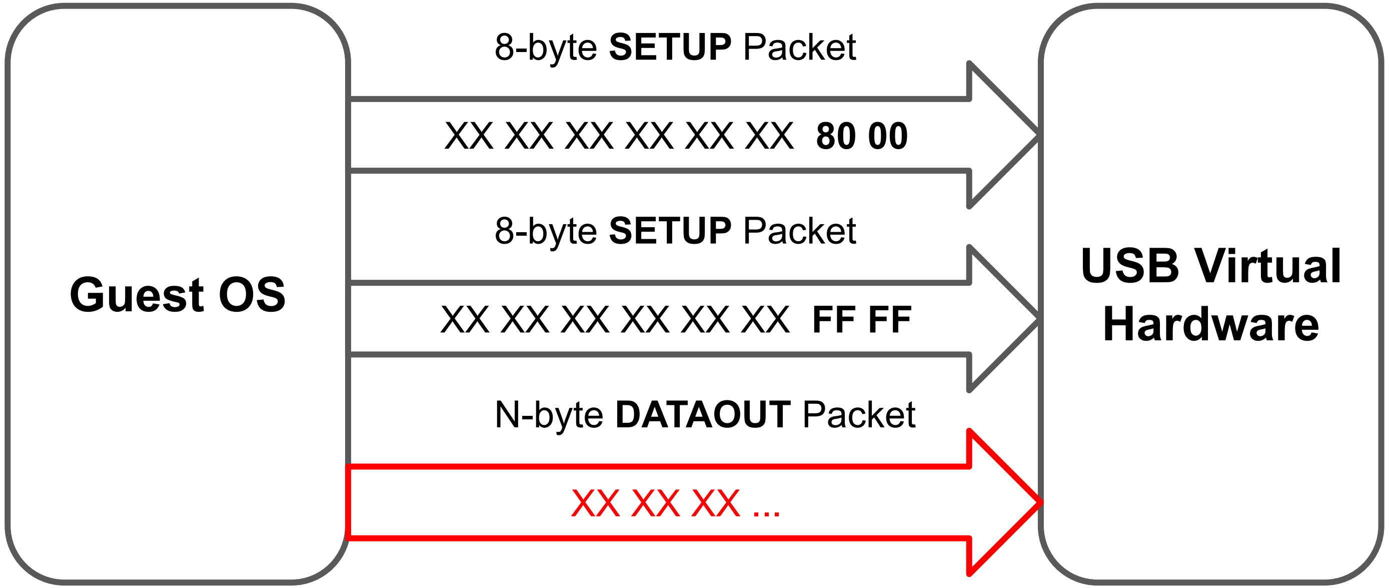
**First, we send an 8-byte SETUP packet to make the s->setup_state be SETUP_STATE_DATA**

# Build A POC



Next, we send another 8-byte SETUP packet to make the s->setup_len big enough

# Build A POC



**Finally, we send another N-byte DATAOUT packet. We will overflow-write to the data buffer.**

长亭科技
CHAITIN

## 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.

```
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?

```
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.

```
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 */
    ...
```

## OOB Read and Write → Arbitrary Address Read

- We can get the USBDescID by sending some USB packets.

```
struct USBDesc {
    USBDescID                   id;
    const USBDescDevice         *full;
    const USBDescDevice         *high;
    const USBDescDevice         *super;
    const char* const           *str;
    const USBDescMSOS           *msos;
};

struct USBDescID {
    uint16_t                    idVendor;
    uint16_t                    idProduct;
    uint16_t                    bcdDevice;
    uint8_t                     iManufacturer;
    uint8_t                     iProduct;
    uint8_t                     iSerialNumber;
};
```

## OOB Read and Write → Arbitrary Address Read

- We can get the USBDescID by sending some USB packets.
- Arbitrary Address Read
  - Overwrite the pointer of USBDesc.
  - Get the USBDescID back.

```
struct USBDesc {
    USBDescID                     id;
    const USBDescDevice           *full;
    const USBDescDevice           *high;
    const USBDescDevice           *super;
    const char* const             *str;
    const USBDescMSOS             *msos;
};

struct USBDescID {
    uint16_t                      idVendor;
    uint16_t                      idProduct;
    uint16_t                      bcdDevice;
    uint8_t                       iManufacturer;
    uint8_t                       iProduct;
    uint8_t                       iSerialNumber;
};
```

## Arbitrary Address Read → Leak libc

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

```
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 */
    ...

```

## Arbitrary Address Read → Leak libc

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

```
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;
};
```

# Arbitrary Address Read → Leak libc

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

- **DeviceState** has a free function pointers

```
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 */
    ...

```

# Arbitrary Address Read → Leak libc

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

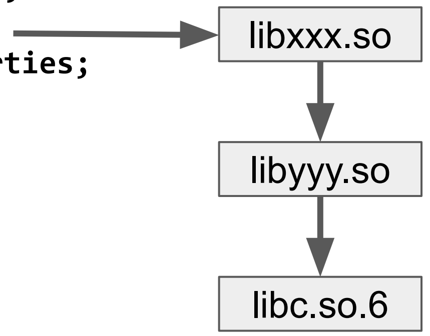
- **DeviceState** has a free function pointers

```
struct DeviceState {
    /*< private >*/
    Object parent_obj;
    ...
};

struct Object
{
    /*< private >*/
    ObjectClass *class;
    ObjectFree *free;
    GHashTable *properties;
    uint32_t ref;
    Object *parent;
};

```

## Arbitrary Address Read → Leak libc

- 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

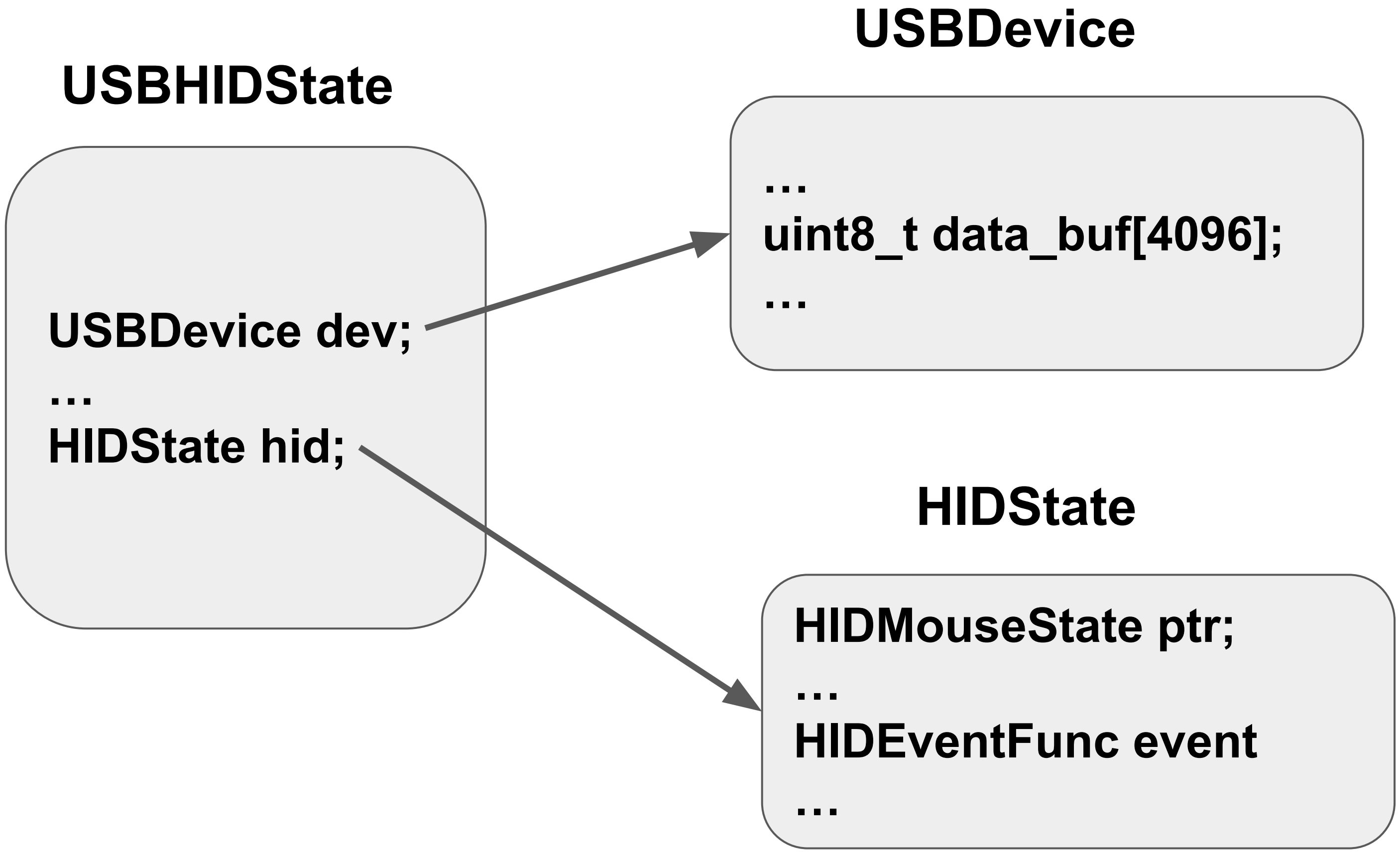
```
struct DeviceState {
    /*< private >*/
    Object parent_obj;
    ...
};

struct Object
{
    /*< private >*/
    ObjectClass *class;
    ObjectFree *free;
    GHashTable *properties;
    uint32_t ref;
    Object *parent;
};

```



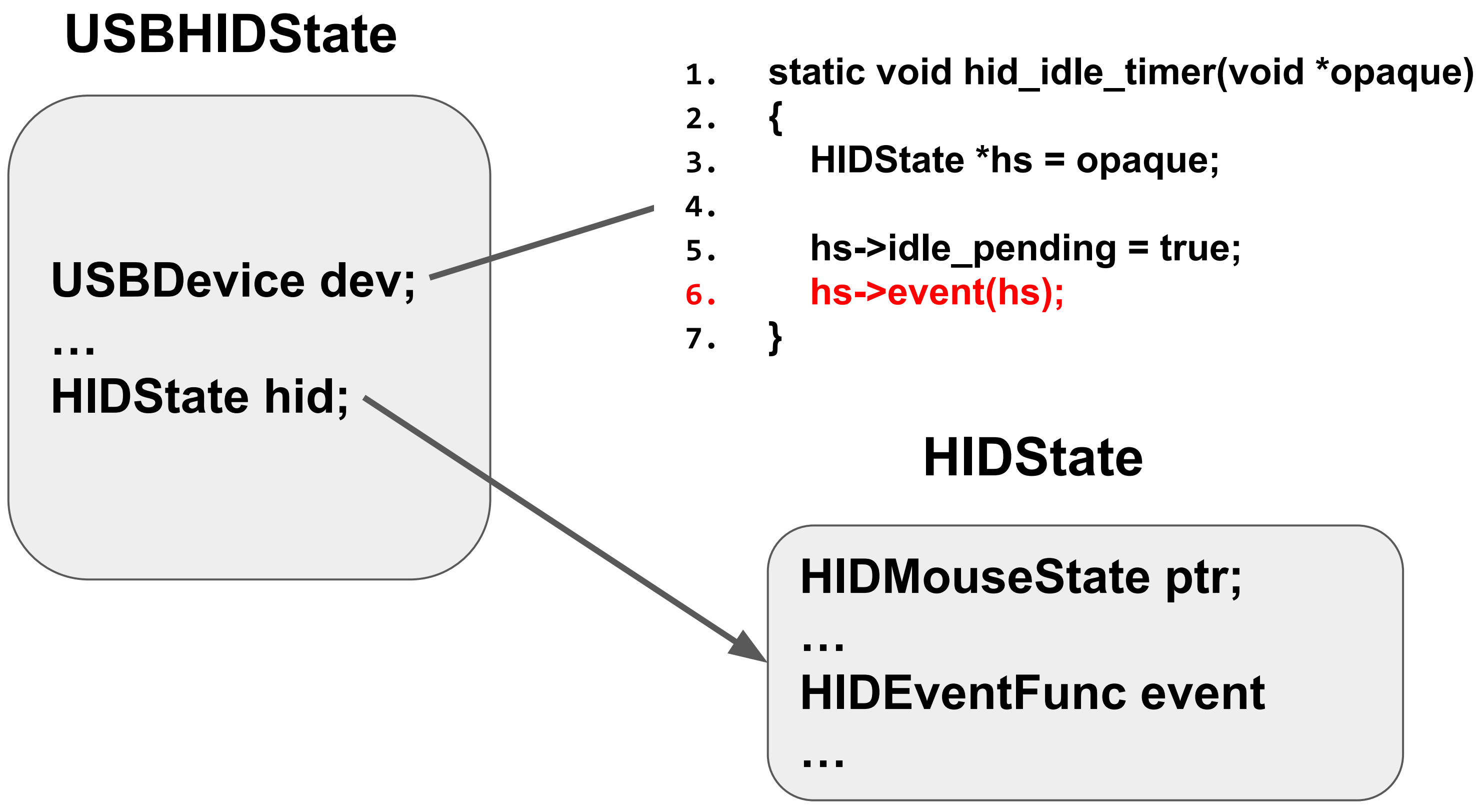
## Leak libc → Leak system

- 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

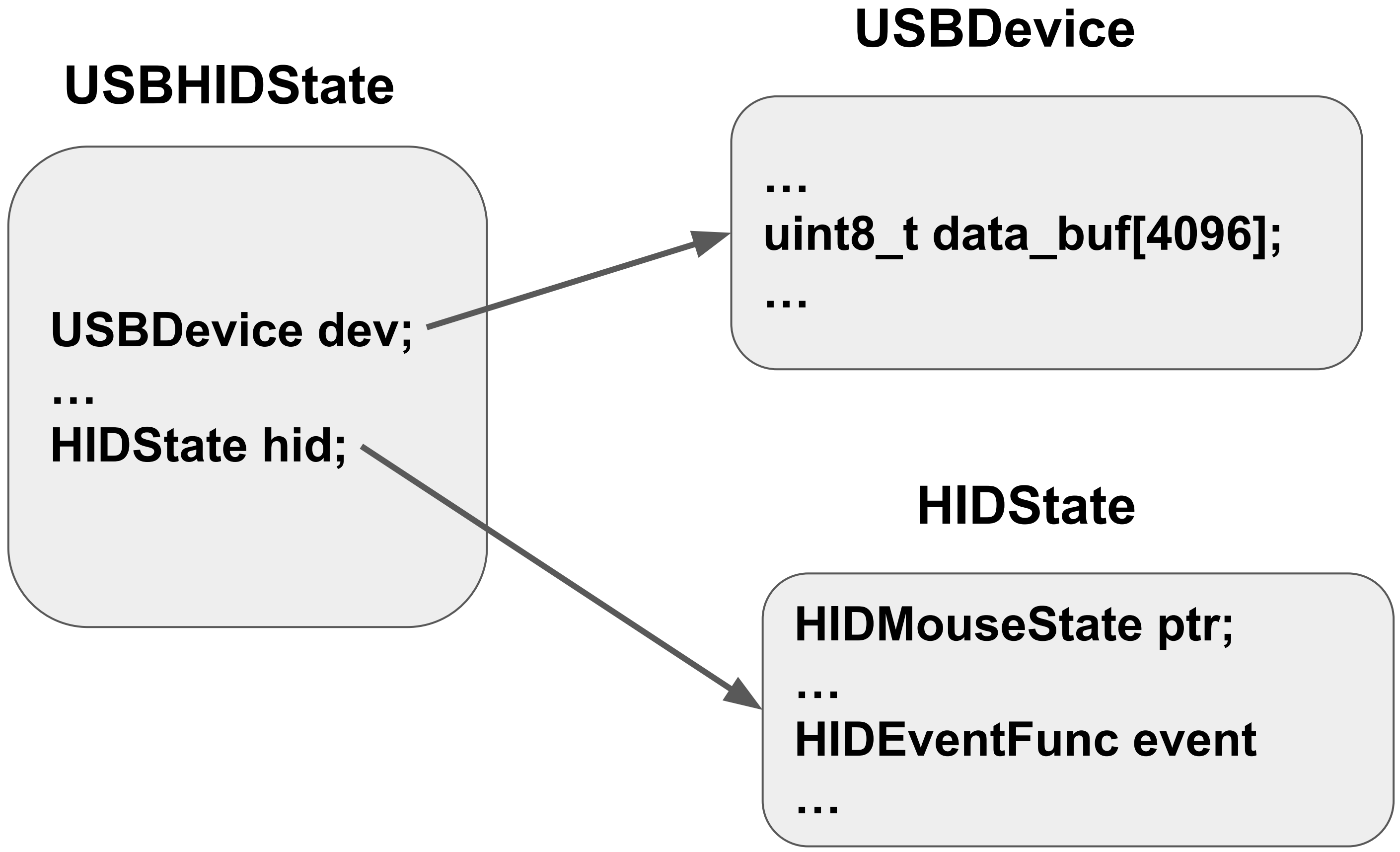

USBHIDState
USBDevice dev;
…
HIDState hid;
USBDevice
…
uint8_t data_buf[4096];
…
HIDState
HIDMouseState ptr;
…
HIDEventFunc event
…

## Control PC

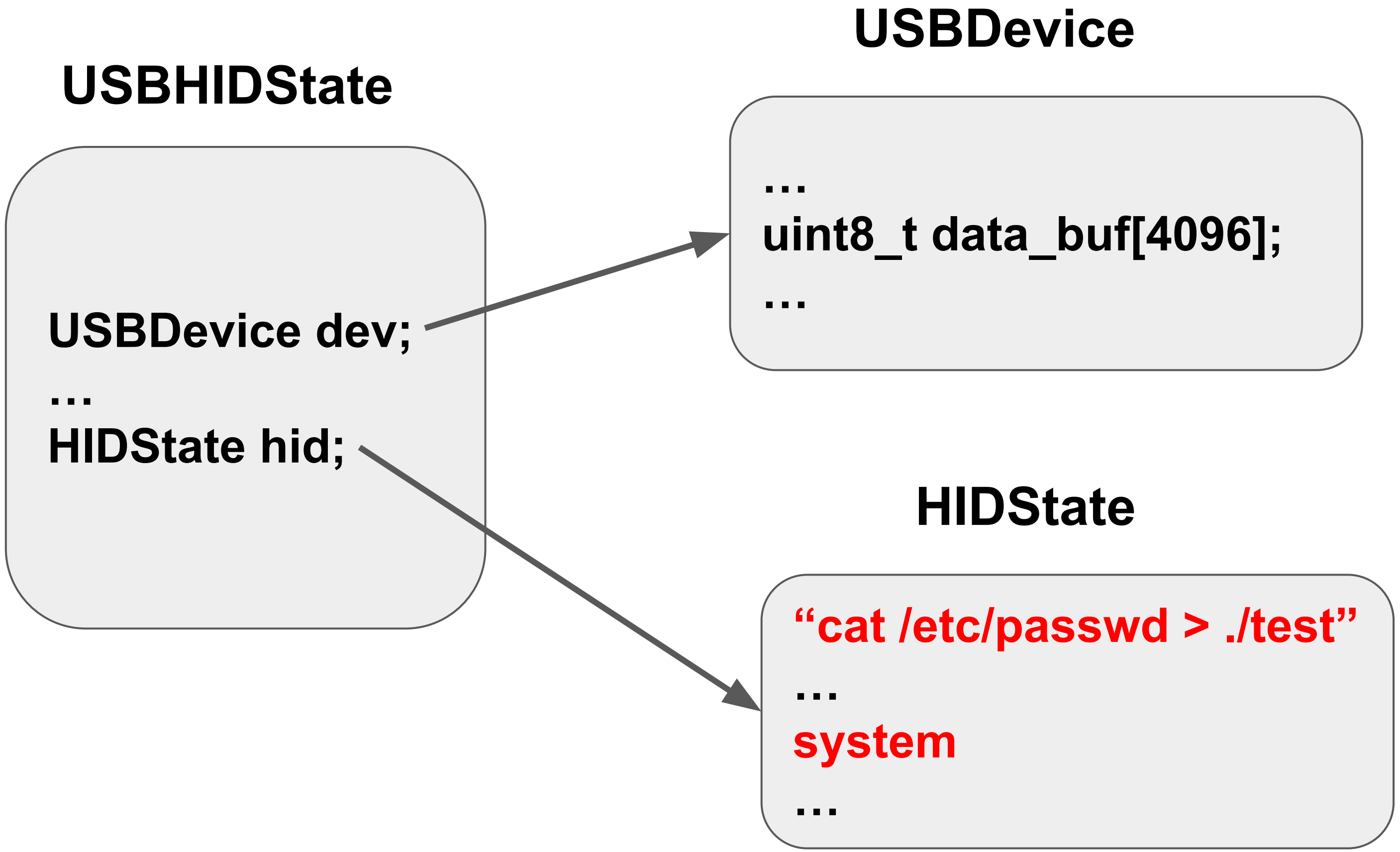


```c
static void hid_idle_timer(void *opaque)
{
    HIDState *hs = opaque;

    hs->idle_pending = true;
    hs->event(hs);
}
```

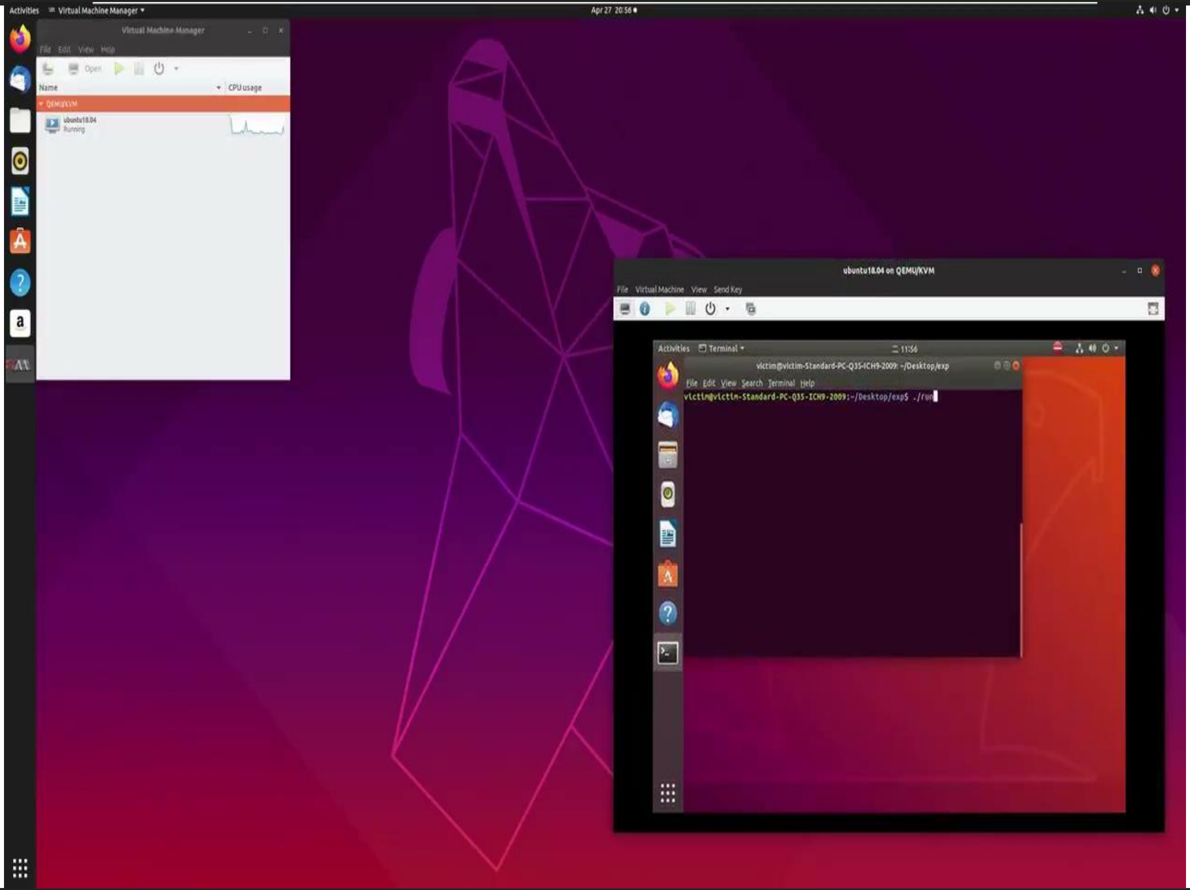
## Control PC


USBHIDState
USBDevice dev;
…
HIDState hid;
USBDevice
…
uint8_t data_buf[4096];
…
HIDState
HIDMouseState ptr;
…
HIDEventFunc event
…

## Control PC


USBHIDState
USBDevice dev;
...
HIDState hid;
USBDevice
...
uint8_t data_buf[4096];
...
HIDState
“cat /etc/passwd > ./test”
...
system
...

## Demo



```
victim@victim-Standard-PC-Q35-ICH9-2009:~/Desktop/exp$ ./run
```

## 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!

@f1yYY__

长亭科技

CHAITIN