



360
互联网安全中心

Virtualization System Vulnerability Discovery Framework

Speaker: Qinghao Tang

Title : 360 Marvel Team Leader

360 Marvel Team

Established in May 2015, the first professional cloud computing and virtualization security team in China. Focusing on attack and defense techniques in virtualization system.

- fuzzing framework
- guest machine escape technology
- Hypervisor risk defense technology

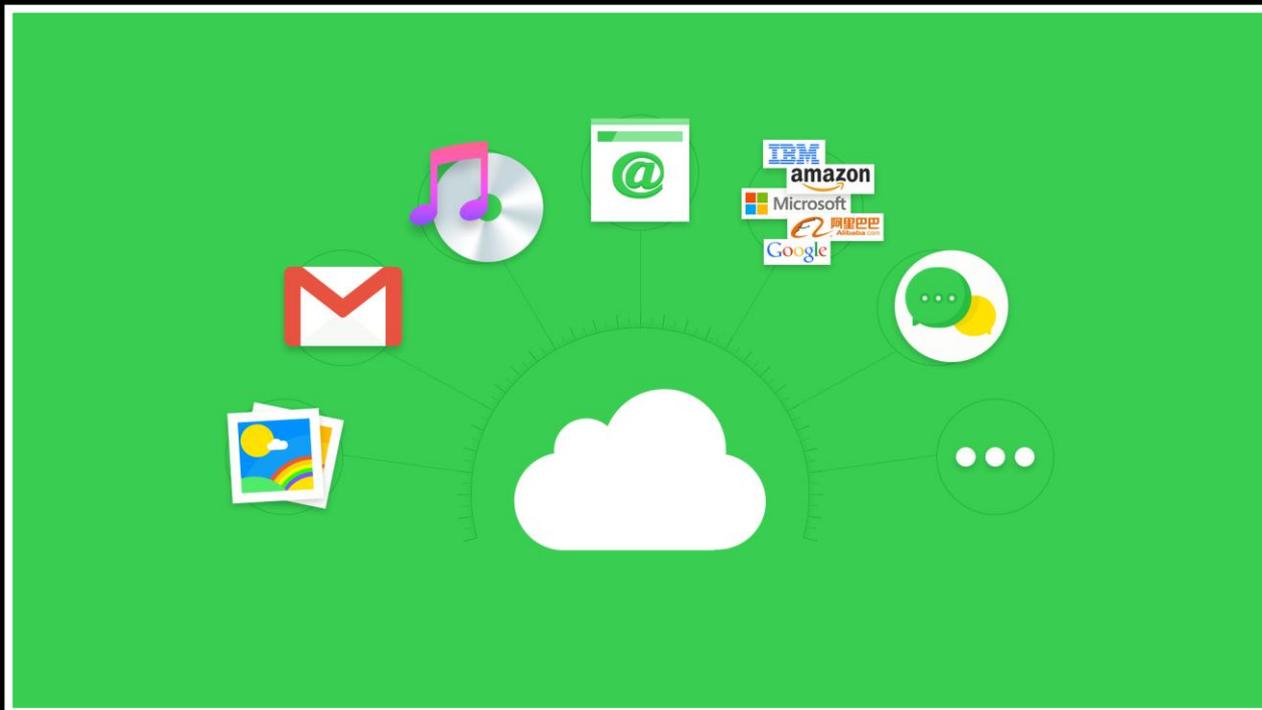


Agenda

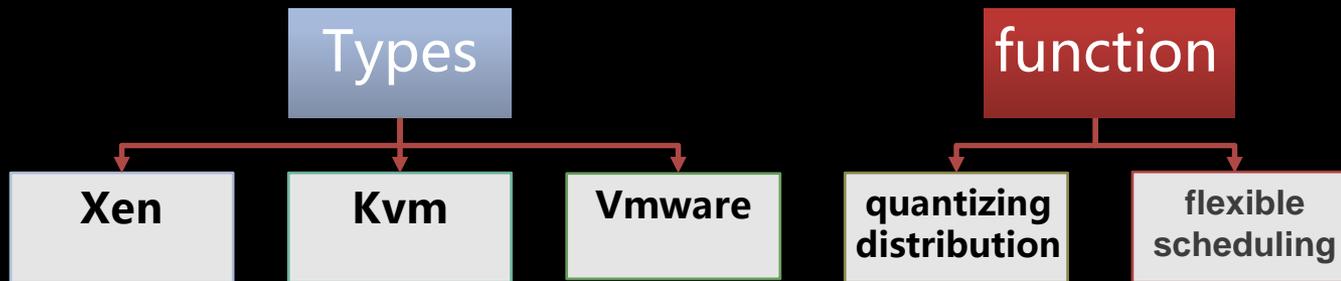
- **Virtualization System Attack Surface**
- **The fuzzing framework**
- **Case study**

Virtualization System Attack Surface

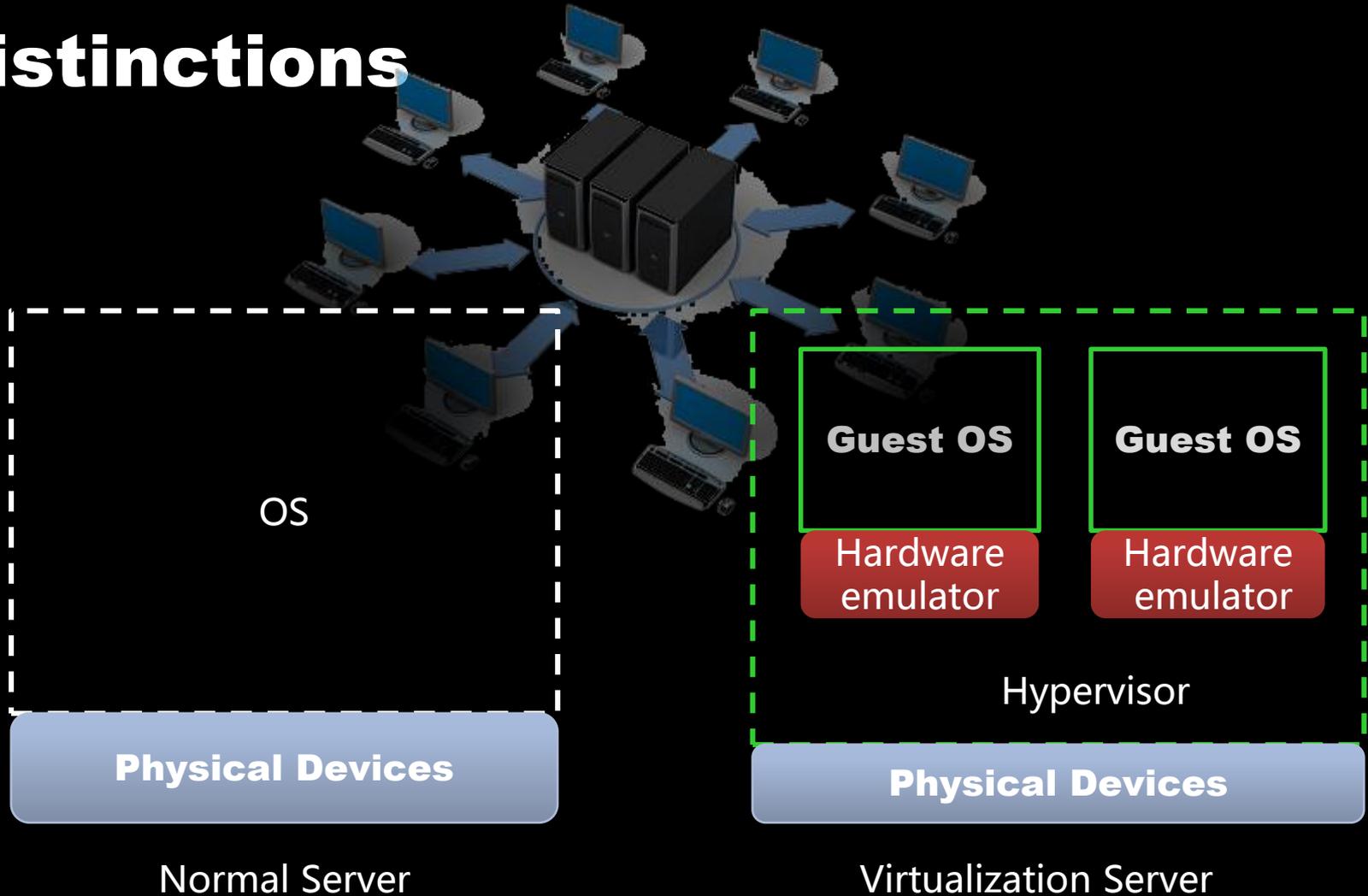
Cloud Computing



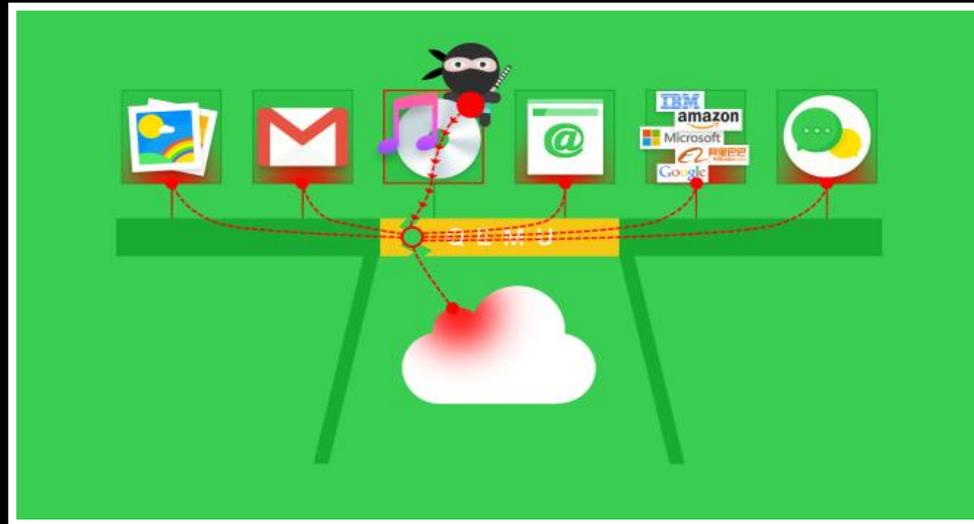
Hypervisor



Distinctions

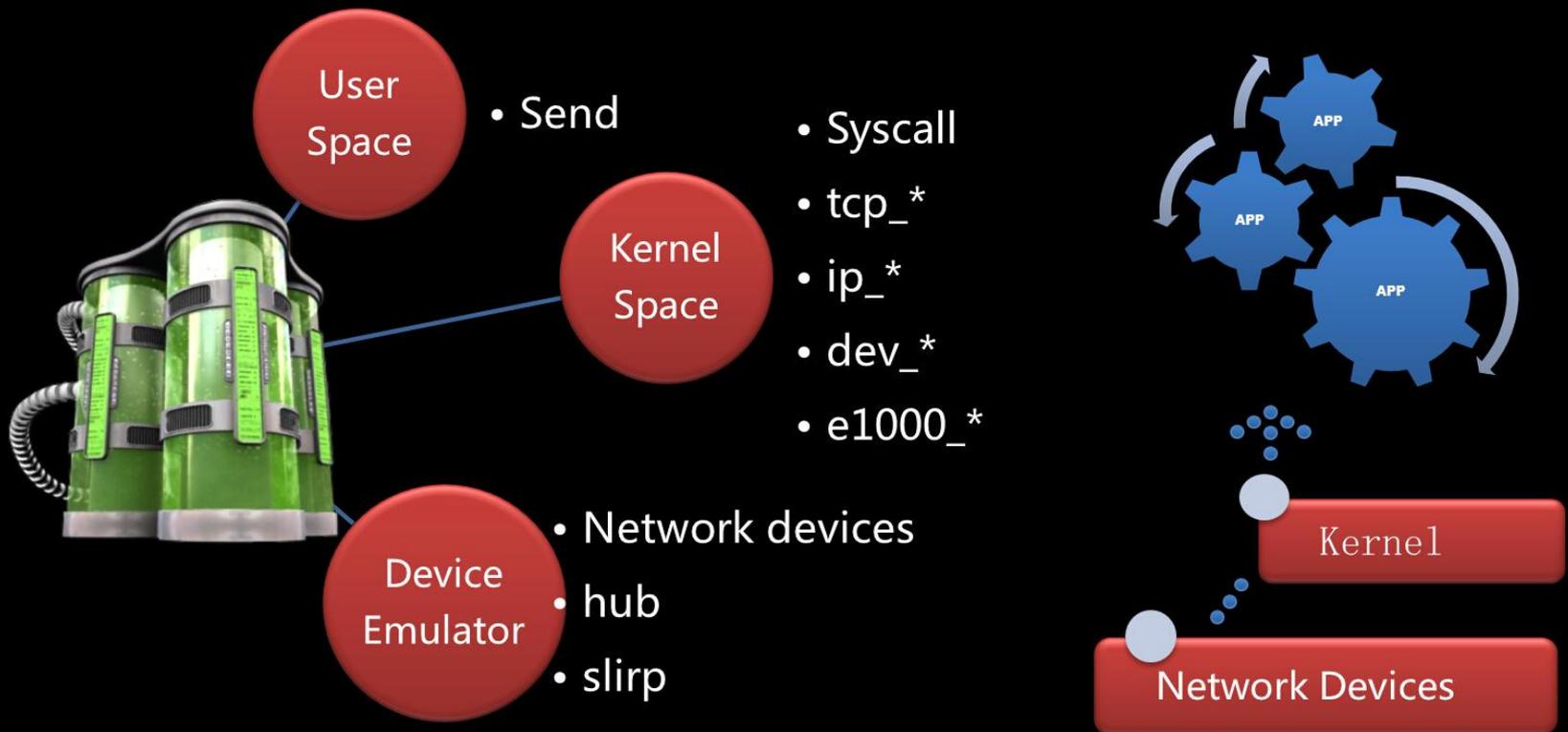


Attacking Processes in cloud computing



1. Enter VM via web or other devices
2. Exploit virtualization system vulnerabilities to escape VM
3. lateral movements to others VMs on host
4. Access to host network

Operation Principles of device emulators



The attack surface

- Hardware virtualization components' diversity
 - Qemu : 30+
 - Vmware : 20+
- Bridge between inside-outside
 - VM os -> device emulators -> Host os
- Related Vulnerabilities result big dangers

Compare to traditional targets

- Hardware virtualization focus on lower



System Kernel

Hypervisor

- Testing data totally different

Vulnerabilities found by us

```
CVE-2015-5225 CVE-2015-5279 CVE-2015-6815
CVE-2015-6855 CVE-2015-8345 CVE-2015-7504
CVE-2015-7549 CVE-2015-8567 CVE-2015-8568
CVE-2015-8558 CVE-2015-8613 CVE-2015-8701
CVE-2016-1568 CVE-2016-1570 CVE-2016-2392
```

Fuzzing Framework

Basic intro

Attack surface : hardware virtual components

Environment : qemu , vmware

Testing results : more than 25 vulnerabilities

Challenges : lower layers hard to predict ;

Methods for testing hardware virtual components

1. Analyze data which flowed to components
2. Change flowed-in data' s contents and timing
3. Recording all of tiny abnormal activities
4. Analyze abnormal activities, find reasons
5. optimize fuzz framework

Other factors of fuzz framework

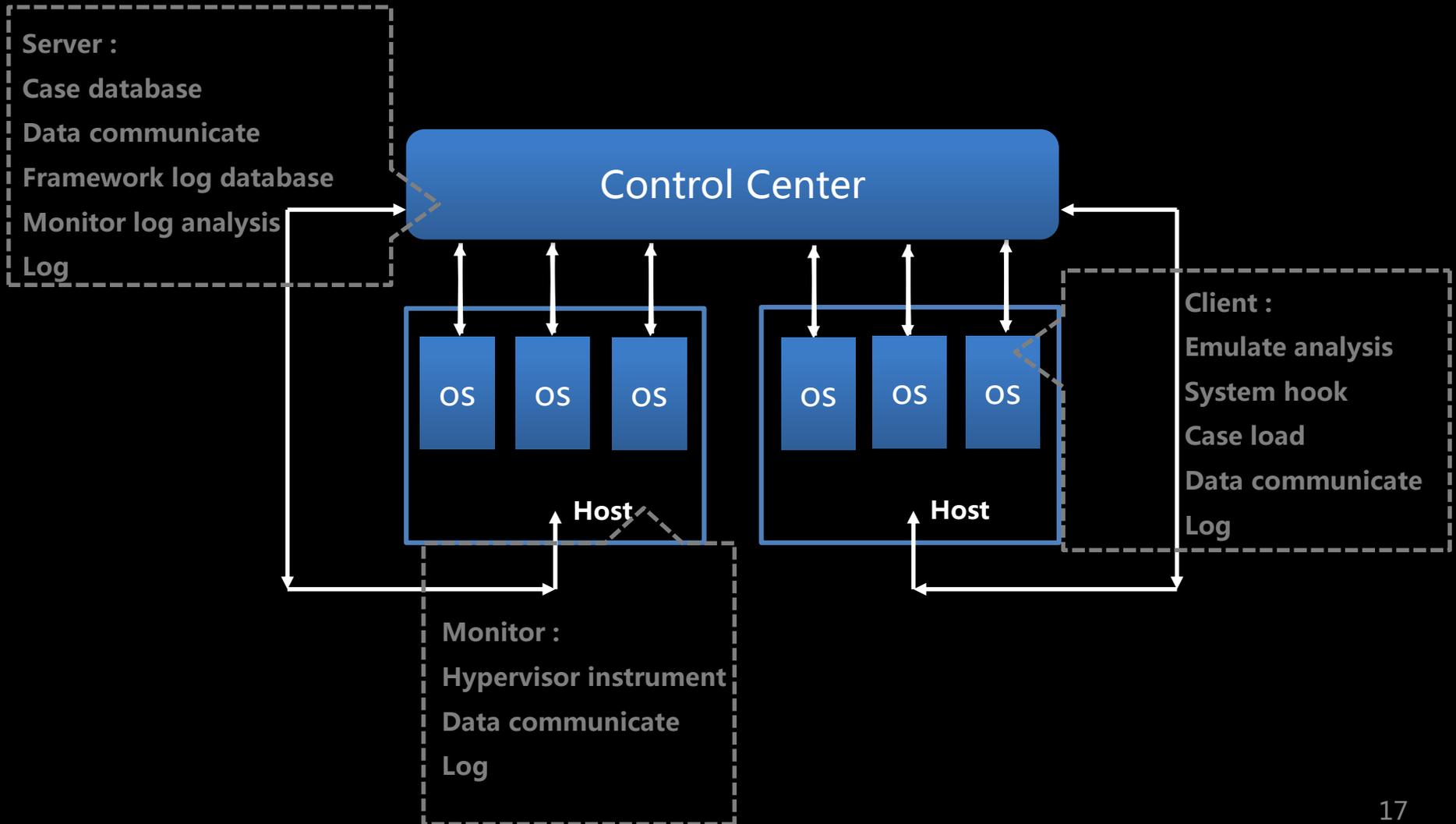
1. Flexibility (other OS)

- vm in Linux
- coding in C and Python

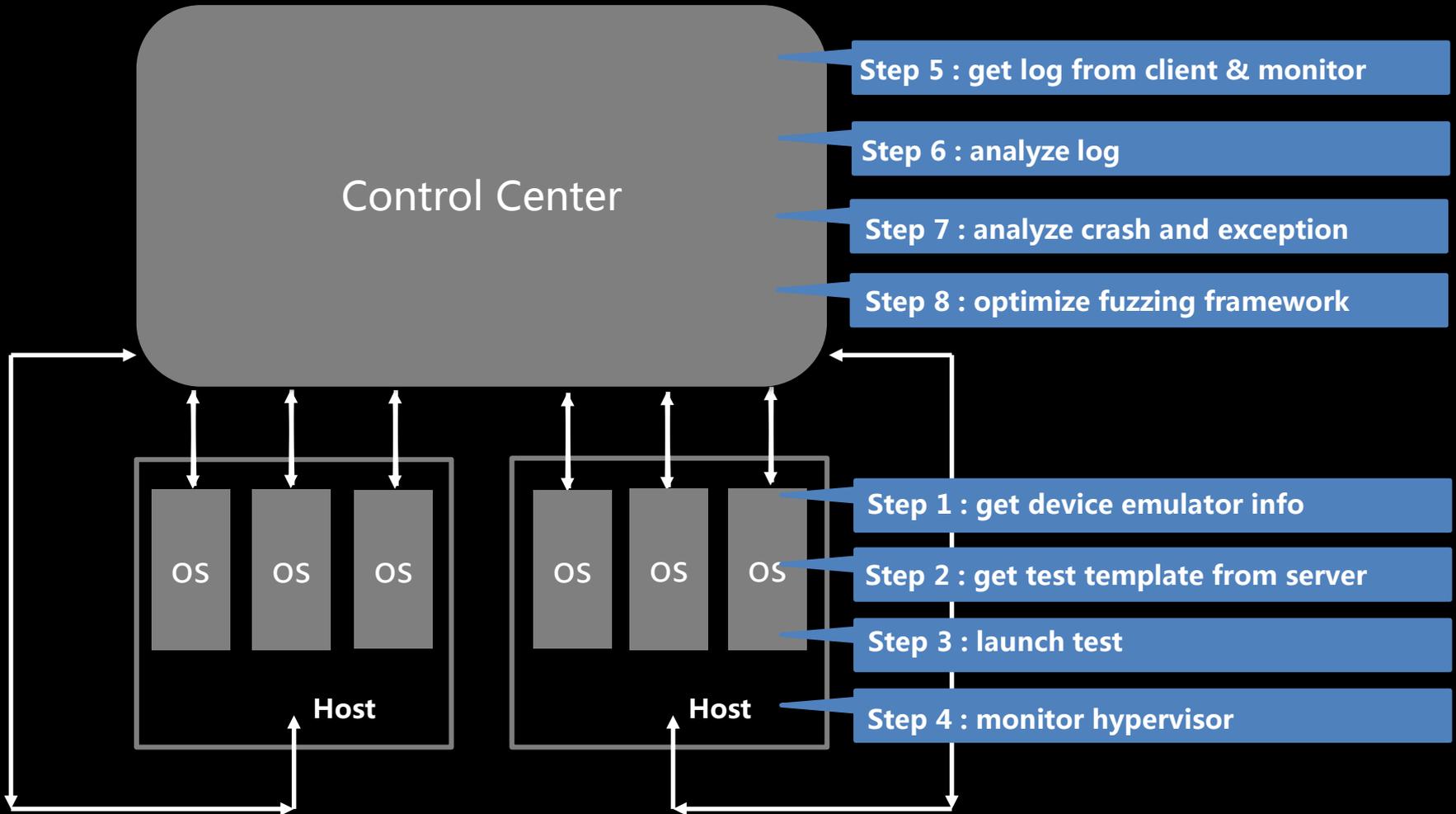
2. Deeply understand VM system

- language for coding
- development environment
- coding style

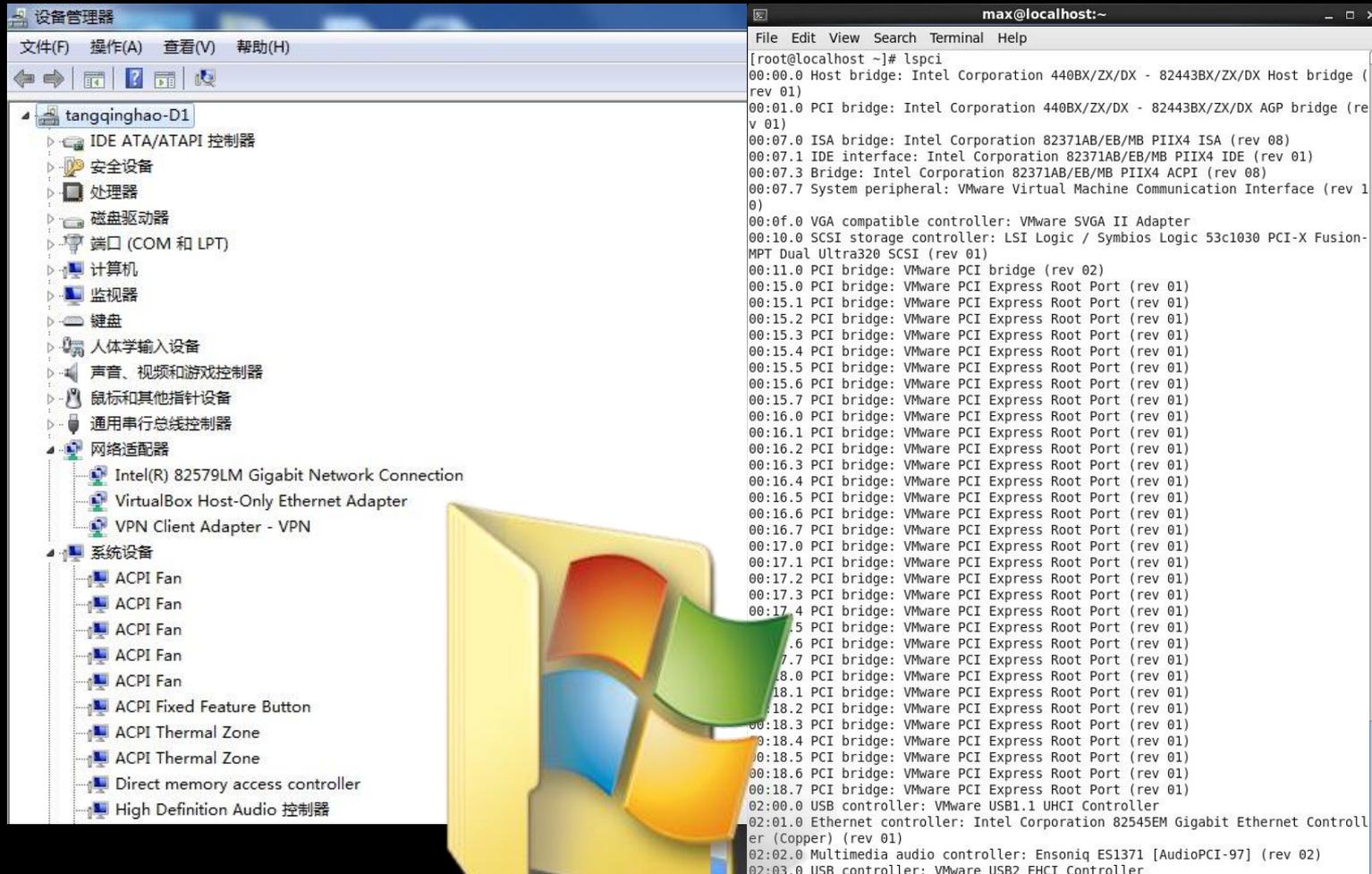
Fuzz framework structure



Fuzz framework working flow



Get target components info



The image displays two windows side-by-side. The left window is the Windows Device Manager for a system named 'tangqinghao-D1'. It shows a tree view of hardware categories, with 'Network adapters' expanded to show 'Intel(R) 82579LM Gigabit Network Connection', 'VirtualBox Host-Only Ethernet Adapter', and 'VPN Client Adapter - VPN'. The 'System devices' category is also expanded, showing various components like ACPI Fans, buttons, and audio controllers.

The right window is a terminal window titled 'max@localhost:~'. It shows the output of the 'lspci' command, listing hardware components and their PCI IDs. The output includes:

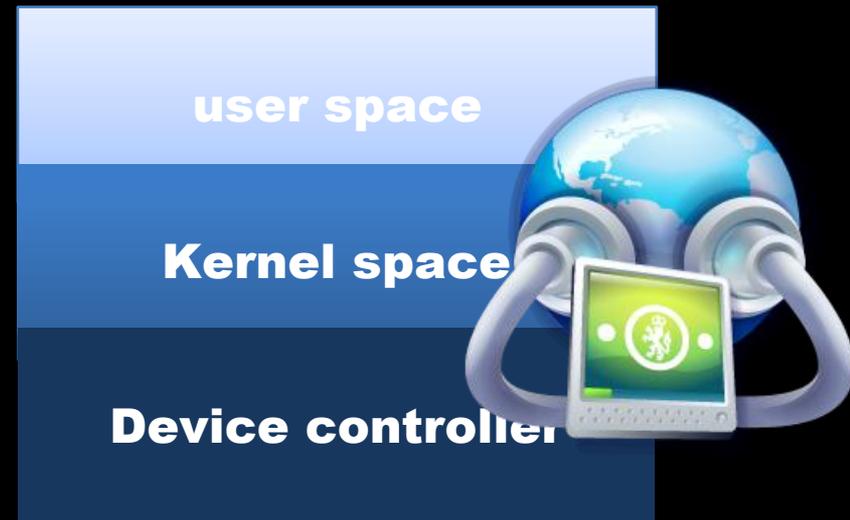
```
[root@localhost ~]# lspci
00:00.0 Host bridge: Intel Corporation 440BX/ZX/DX - 82443BX/ZX/DX Host bridge (rev 01)
00:01.0 PCI bridge: Intel Corporation 440BX/ZX/DX - 82443BX/ZX/DX AGP bridge (rev 01)
00:07.0 ISA bridge: Intel Corporation 82371AB/EB/MB PIIX4 ISA (rev 08)
00:07.1 IDE interface: Intel Corporation 82371AB/EB/MB PIIX4 IDE (rev 01)
00:07.3 Bridge: Intel Corporation 82371AB/EB/MB PIIX4 ACPI (rev 08)
00:07.7 System peripheral: VMware Virtual Machine Communication Interface (rev 10)
00:0f.0 VGA compatible controller: VMware SVGA II Adapter
00:10.0 SCSI storage controller: LSI Logic / Symbios Logic 53c1030 PCI-X Fusion-MPT Dual Ultra320 SCSI (rev 01)
00:11.0 PCI bridge: VMware PCI bridge (rev 02)
00:15.0 PCI bridge: VMware PCI Express Root Port (rev 01)
00:15.1 PCI bridge: VMware PCI Express Root Port (rev 01)
00:15.2 PCI bridge: VMware PCI Express Root Port (rev 01)
00:15.3 PCI bridge: VMware PCI Express Root Port (rev 01)
00:15.4 PCI bridge: VMware PCI Express Root Port (rev 01)
00:15.5 PCI bridge: VMware PCI Express Root Port (rev 01)
00:15.6 PCI bridge: VMware PCI Express Root Port (rev 01)
00:15.7 PCI bridge: VMware PCI Express Root Port (rev 01)
00:16.0 PCI bridge: VMware PCI Express Root Port (rev 01)
00:16.1 PCI bridge: VMware PCI Express Root Port (rev 01)
00:16.2 PCI bridge: VMware PCI Express Root Port (rev 01)
00:16.3 PCI bridge: VMware PCI Express Root Port (rev 01)
00:16.4 PCI bridge: VMware PCI Express Root Port (rev 01)
00:16.5 PCI bridge: VMware PCI Express Root Port (rev 01)
00:16.6 PCI bridge: VMware PCI Express Root Port (rev 01)
00:16.7 PCI bridge: VMware PCI Express Root Port (rev 01)
00:17.0 PCI bridge: VMware PCI Express Root Port (rev 01)
00:17.1 PCI bridge: VMware PCI Express Root Port (rev 01)
00:17.2 PCI bridge: VMware PCI Express Root Port (rev 01)
00:17.3 PCI bridge: VMware PCI Express Root Port (rev 01)
00:17.4 PCI bridge: VMware PCI Express Root Port (rev 01)
00:17.5 PCI bridge: VMware PCI Express Root Port (rev 01)
00:17.6 PCI bridge: VMware PCI Express Root Port (rev 01)
00:17.7 PCI bridge: VMware PCI Express Root Port (rev 01)
00:18.0 PCI bridge: VMware PCI Express Root Port (rev 01)
00:18.1 PCI bridge: VMware PCI Express Root Port (rev 01)
00:18.2 PCI bridge: VMware PCI Express Root Port (rev 01)
00:18.3 PCI bridge: VMware PCI Express Root Port (rev 01)
00:18.4 PCI bridge: VMware PCI Express Root Port (rev 01)
00:18.5 PCI bridge: VMware PCI Express Root Port (rev 01)
00:18.6 PCI bridge: VMware PCI Express Root Port (rev 01)
00:18.7 PCI bridge: VMware PCI Express Root Port (rev 01)
02:00.0 USB controller: VMware USB1.1 UHCI Controller
02:01.0 Ethernet controller: Intel Corporation 82545EM Gigabit Ethernet Controller (Copper) (rev 01)
02:02.0 Multimedia audio controller: Ensoniq ES1371 [AudioPCI-97] (rev 02)
02:03.0 USB controller: VMware USB2 EHCI Controller
```

Testing data

- Device access ports
- Device deal with structures used by data.
- Device data processing

Testing data attacks

- User space: generate testing data, send request to client kernel
- Kernel space: apply for memory, fill memory, send info to ports
- Device emulator : testing data flow inside , trigger exceptions



Monitor

VM management

- Snapshot
- Reboot
- VM device editing

Dynamic debugging

- Debugging Mode on Start
- Load Debugging Plugin

VM processing log

- User space
- Kernel space

Exceptions occur in device emulator

- VM os crash
- Hypervisor crash
- Invisible results

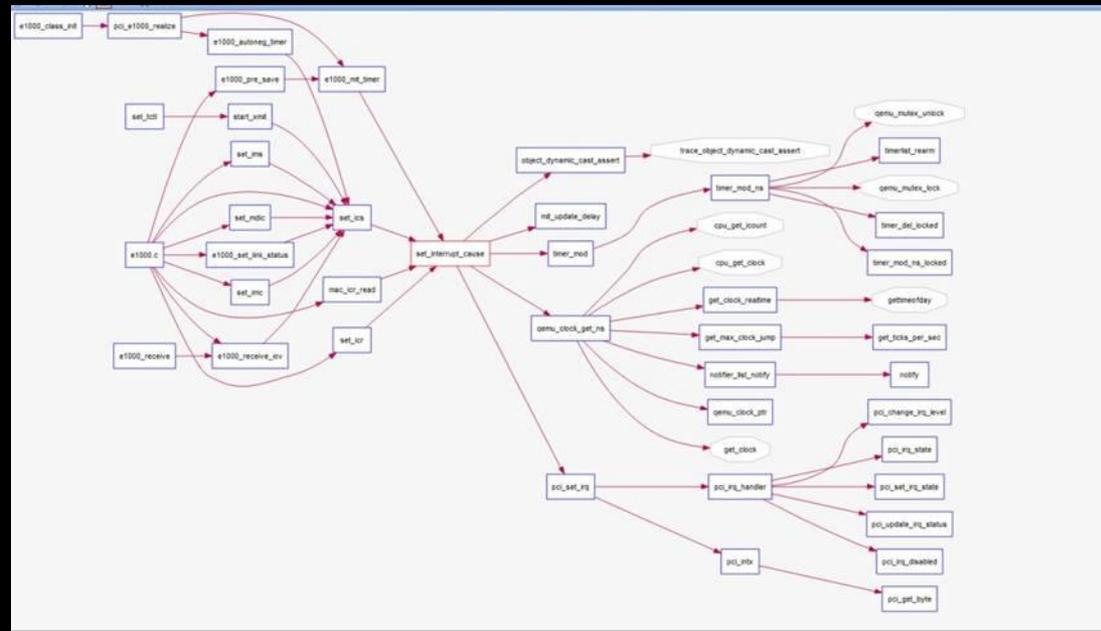
```
(gdb) c
Continuing.
[Thread 0x7ffffa4ece700 (LWP 64212) exited]
[Thread 0x7ffff97b7b700 (LWP 64211) exited]

Breakpoint 1, ne2000_receive (nc=0x7ffff9c2f7f0, buf=0x7ffffffffffd390 "RT",
    size =126) at hw/net/ne2000.c:180
180 {
(gdb) disable
(gdb) c
Continuing.
[New Thread 0x7ffff97b7b700 (LWP 64229)]

Program received signal SIGSEGV, Segmentation fault.
0x00007ffff5c022dc in _int_malloc () from /lib64/libc.so.6
(gdb) bt
#0 0x00007ffff5c022dc in _int_malloc () from /lib64/libc.so.6
#1 0x00007ffff5c036b1 in malloc () from /lib64/libc.so.6
#2 0x00007ffff786c676 in malloc_and_trace (n bytes=49280) at vl.c:2724
#3 0x00007ffff6923cd5 in g_malloc () from /lib64/libglib-2.0.so.0
#4 0x00007ffff6938e0a in g_slice_alloc () from /lib64/libglib-2.0.so.0
#5 0x00007ffff776a614 in virtio_blk_alloc_request (s=0x7ffff8894230)
    at /home/max/qemu-2.4.0/hw/block/virtio-blk.c:33
#6 0x00007ffff776ad06 in virtio_blk_get_request (s=0x7ffff8894230)
    at /home/max/qemu-2.4.0/hw/block/virtio-blk.c:192
#7 0x00007ffff776beed in virtio_blk_handle_output (vdev=0x7ffff8894230, vq=
    0x7ffff9924ff0) at /home/max/qemu-2.4.0/hw/block/virtio-blk.c:603
#8 0x00007ffff77aa921 in virtio_queue_notify_vq (vq=0x7ffff9924ff0)
    at /home/max/qemu-2.4.0/hw/virtio/virtio.c:151
#9 0x00007ffff77aca59 in virtio_queue_host_notify (vq=0x7ffff9924ff0)
    at /home/max/qemu-2.4.0/hw/virtio/virtio.c:1536
#10 0x00007ffff7a780f2 in qemu_iohandler_poll (pollfds=0x7ffff87c1240)
    at iohandler.c:126
#11 0x00007ffff7a77d64 in main_loop_wait (nonblocking=0) at main-loop.c:
#12 0x00007ffff7868e44 in main_loop () at vl.c:1902
#13 0x00007ffff7870f9e in main (argc=14, argv=0x7ffffffffffe238, envp=
    0x7ffffffffffe2b0) at vl.c:4653
(gdb) █
```

Advanced monitoring skills

- Dynamic
- Static



Optimize fuzz framework by using log data

- Client log

Decrease invalid combinations

- Monitor log

Promote coverage

- Server log

Limitation & Future

- Permission limitation
- More kinds of virtualization systems :
Hyper-V ; VMWARE
- More attack surfaces :
hypercall ; virtio ; guest machine client
- About open source project

Case Study

Principle of e1000 Network Device

•Initialization

Port Allocation , Address Mapping
Device Status Setting, Resource Allocation

•Data Transfer

'Write Command' to device TDT register
process of descriptor
3 types descriptor : context , data , legacy
data xfer
set status , wait for next instruction

•Processing Details

Circular Memory
TSO : tcp segmentation/flow control.

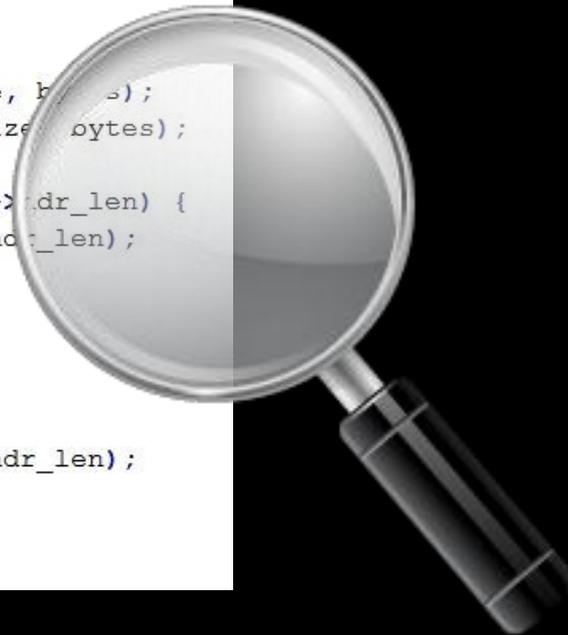


E1000 vulnerability analysis

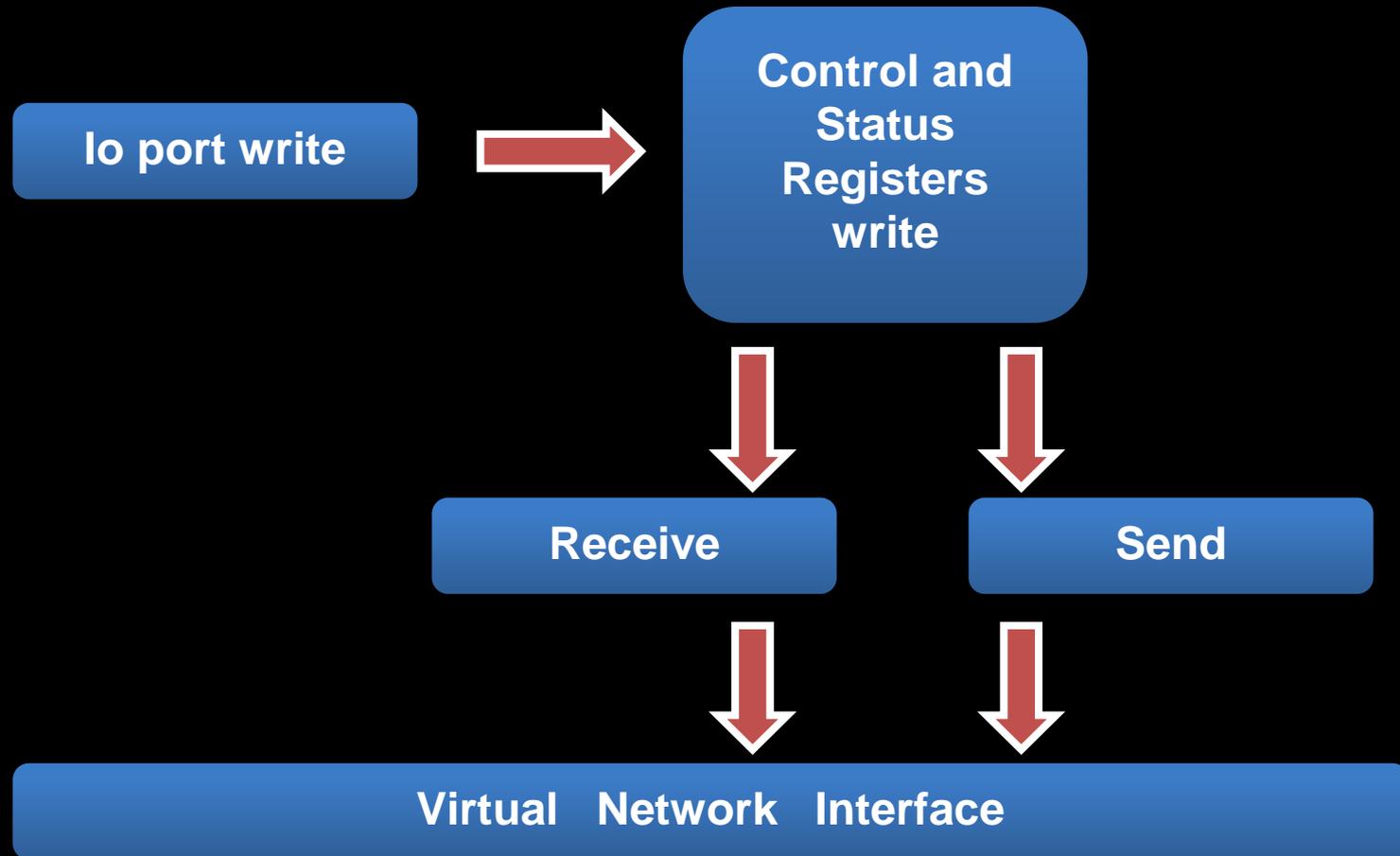
- Qemu e1000 Network Device
- Vmware e1000 Network Device

```
do {
    bytes = split_size;
    if (tp->size + bytes > msh)
        bytes = msh - tp->size;

    bytes = MIN(sizeof(tp->data) - tp->size, bytes);
    pci_dma_read(d, addr, tp->data + tp->size, bytes);
    sz = tp->size + bytes;
    if (sz >= tp->hdr_len && tp->size < tp->hdr_len) {
        memmove(tp->header, tp->data, tp->hdr_len);
    }
    tp->size = sz;
    addr += bytes;
    if (sz == msh) {
        xmit_seg(s);
        memmove(tp->data, tp->header, tp->hdr_len);
        tp->size = tp->hdr_len;
    }
} while (split_size -- bytes);
```



Pcnet network card emulator working processes



Pcnet vulnerability analysis

- Qemu pcnet Network Device

```
} else if (s->looptest == PCNET_LOOPTEST_CRC ||
           !CSR_DXMTFCS(s) || size < MIN_BUF_SIZE+4) {
    uint32_t fcs = ~0;
    uint8_t *p = src;

    while (p != &src[size])
        CRC(fcs, *p++);
    *(uint32_t *)p = htonl(fcs);
    size += 4;
}
```



Summary

**Stay tuned for more achievements by
360 Marvel Team**



Email : tangqinghao@360.cn
QQ : 702108451