Breakout Script Of the Westworld

Tang Tianwen(nickname VictorV)

Cyber Security Researcher at Vulcan Team, 360 Security

Xiao Wei

Cyber Security Researcher at Vulcan Team, 360 Security

HITB LOCKDOWN 002 livestream

About Us

VictorV

○ Cyber security researcher at 360 Security Vulcan Team.

○ Found several critical vulnerabilities on VMware products.

CVE-2017-4902, CVE-2018-6981, CVE-2018-6983 …

○ Focus on Virtualization Security.

○ Found two critical vulnerabilities on Hyper-V

CVE-2019-1230, CVE-2019-0723

○ Escape from VMware Workstation in public on Tianfu Cup 2018.

About Us

Xiao Wei

- ○ Cyber security researcher at 360 Security Vulcan Team.
- ○ Focus on Virtualization Security and Web browser Security.
- ○ Escape from VMware Workstation, vSphere, VirtualBox, QEMU for several times
- ○ PoC 2016 speaker
- ○ Escape from VMware Workstation on Pwn2Own 2017
- ○ Escape from QEMU, VirtualBox, ESXi on Tianfu Cup 2019

Agenda

- ○ Overview of VM network device architecture
- ○ Exploitation primitives on VMware Workstation & ESXi
- ○ Attack Case of ESXi
- ○ Attack Case of Workstation
- ○ Live demo of escaping
- ○ Conclusion

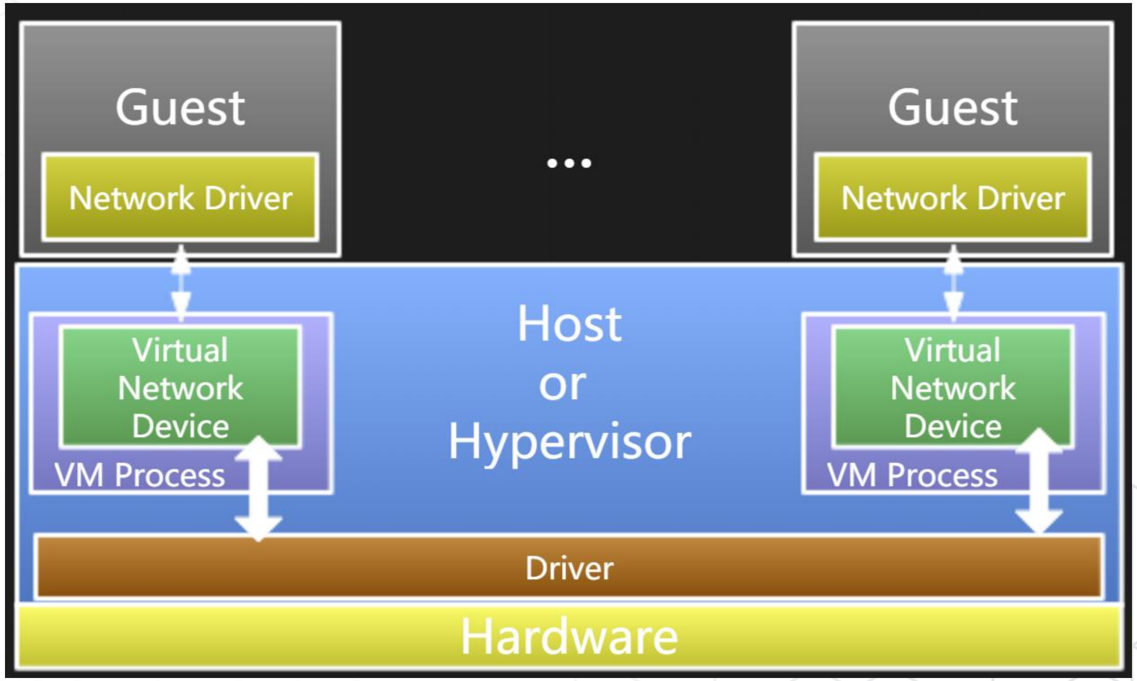
Overview of Virtual Net Device

○ Virtual Network Devices Architecture

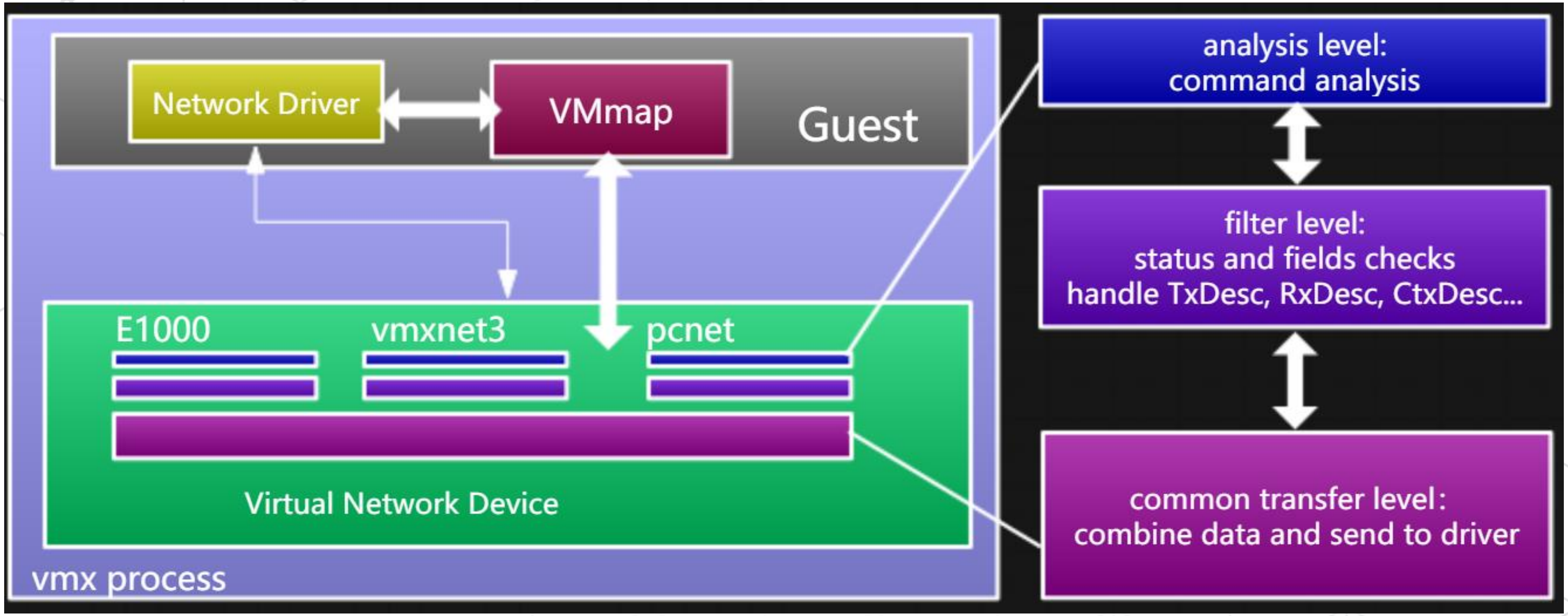
○ Attack Surfaces

Devices Architecture

- Guest Driver sends commands and data via IO port or IO memory
- Each Guest is created by a vmx process in host
- Virtual Device filters data from IO and transmits to physical device



Devices Architecture



Attack Surfaces

- Incorrect handling network command data

  CVE-2018-3294, CVE-2018-6983, CVE-2018-6973...

- Incorrect handling Guest address translation

  CVE-2018-6981, CVE-2018-6982...

- Incomplete checks of socket fields

  VMCI host driver integer overflow

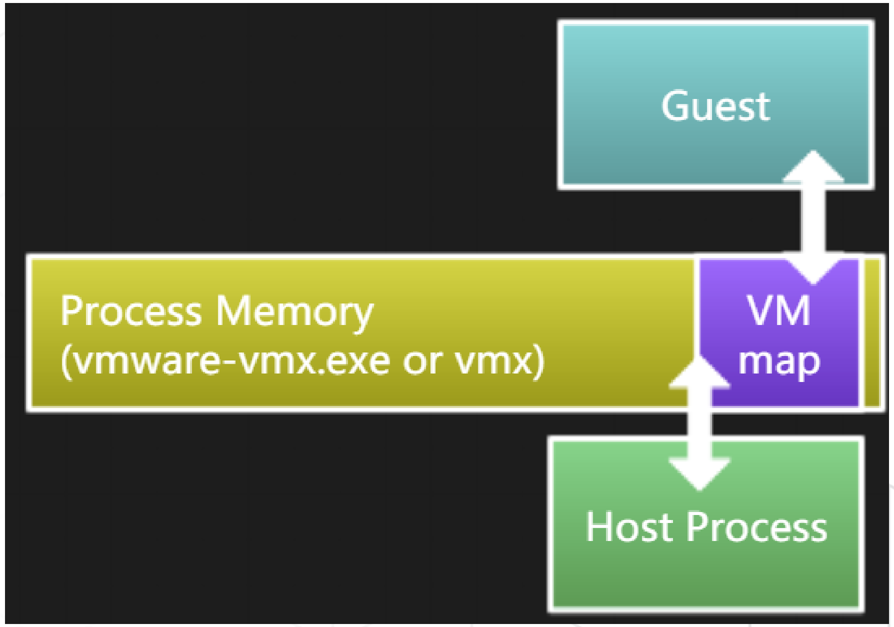
Exploitation Primitives

- Basic information of data transfer
- Heap Spray
- R/W related structures
- Bypass CFG

Basic information of data transfer

Guest Memory

- Guest's physical memory is a map space in vmx process's memory space.
- Vmx process needs to translate a Guest's memory address(as phys) into process address
- If the phys or size is illegal, translation function will return a 4k heap memory, or an array to store translated addresses

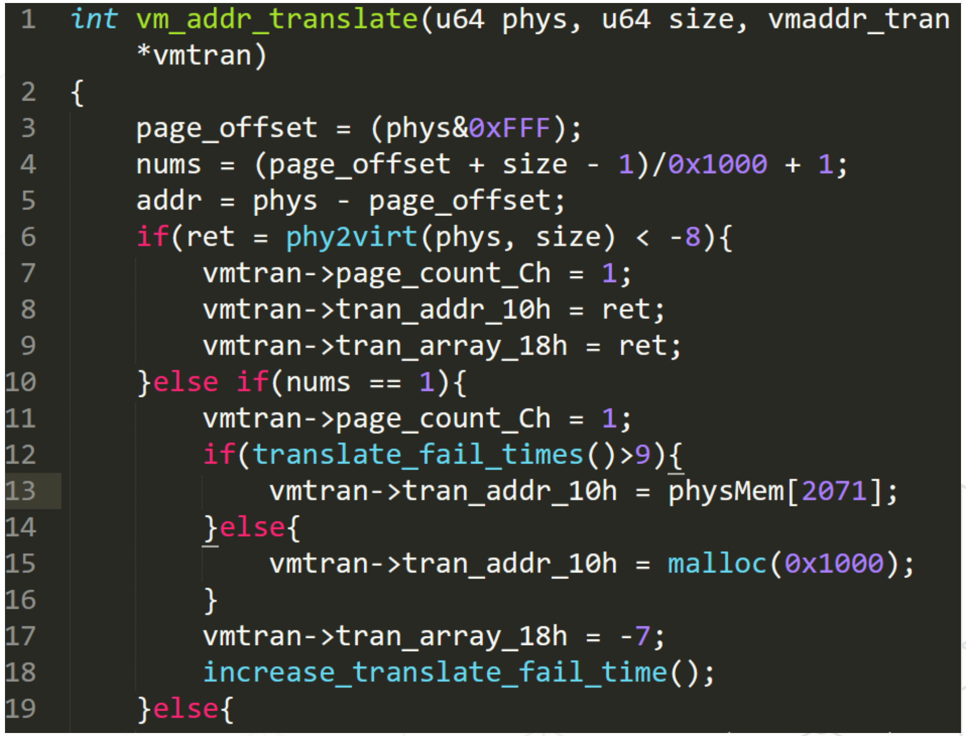


Basic information of data transfer

Translation

```
struct vmaddr_tran {
    _QWORD translated_size_0h;
    _DWORD page_offset_8h;
    _DWORD page_count_Ch;
    _QWORD tran_addr_10h;
    _QWORD tran_array_18h;
    …
};
```

Mark physmem[2071] as H1 at line 13



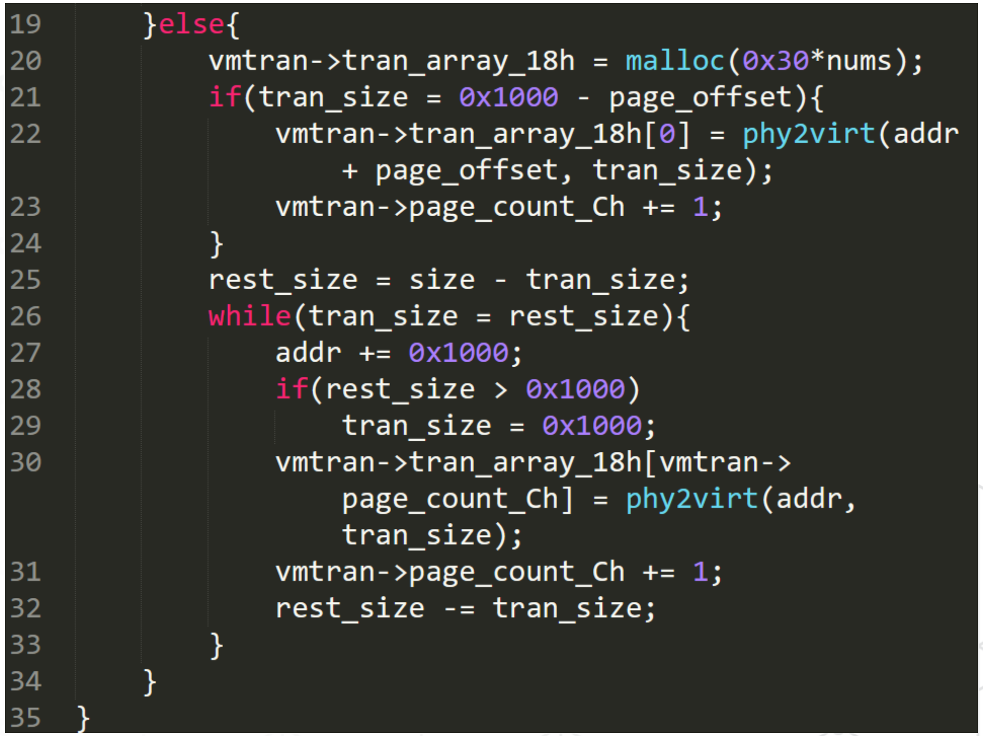
```
1  int vm_addr_translate(u64 phys, u64 size, vmaddr_tran
       *vmtran)
2  {
3      page_offset = (phys&0xFFF);
4      nums = (page_offset + size - 1)/0x1000 + 1;
5      addr = phys - page_offset;
6      if(ret = phy2virt(phys, size) < -8){
7          vmtran->page_count_Ch = 1;
8          vmtran->tran_addr_10h = ret;
9          vmtran->tran_array_18h = ret;
10     }else if(nums == 1){
11         vmtran->page_count_Ch = 1;
12         if(translate_fail_times()>9){
13             vmtran->tran_addr_10h = physMem[2071];
14         }else{
15             vmtran->tran_addr_10h = malloc(0x1000);
16         }
17         vmtran->tran_array_18h = -7;
18         increase_translate_fail_time();
19     }else{
```

Basic information of data transfer

Translation

```
struct vmaddr_tran {
    _QWORD translated_size_0h;
    _DWORD page_offset_8h;
    _DWORD page_count_Ch;
    _QWORD tran_addr_10h;
    _QWORD tran_array_18h;
    …
};
```

Array stores results for each PFN

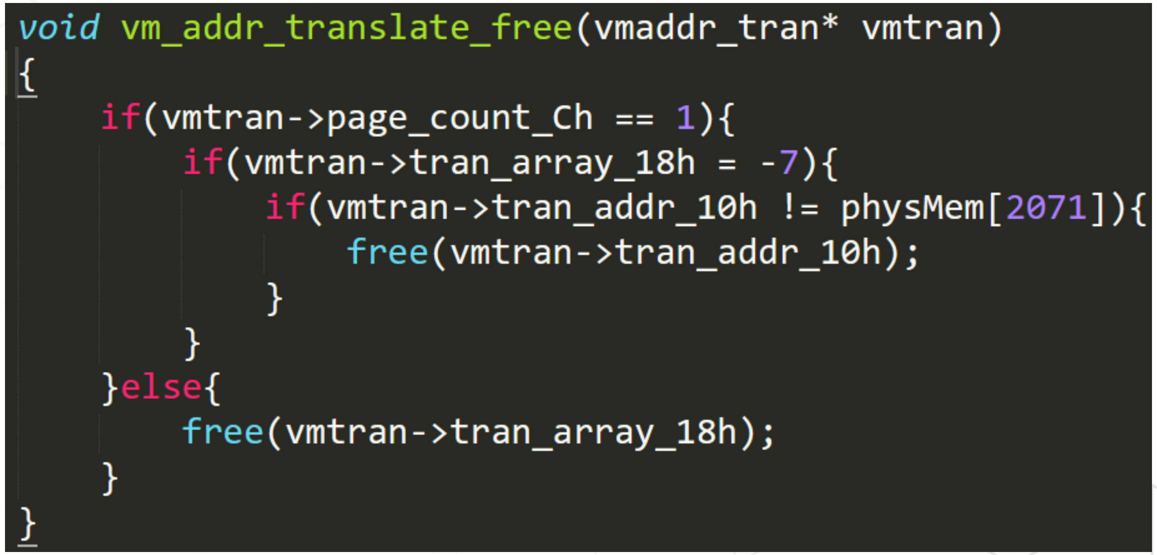


```
    }else{
        vmtran->tran_array_18h = malloc(0x30*nums);
        if(tran_size = 0x1000 - page_offset){
            vmtran->tran_array_18h[0] = phy2virt(addr
                + page_offset, tran_size);
            vmtran->page_count_Ch += 1;
        }
        rest_size = size - tran_size;
        while(tran_size = rest_size){
            addr += 0x1000;
            if(rest_size > 0x1000)
                tran_size = 0x1000;
            vmtran->tran_array_18h[vmtran->
                page_count_Ch] = phy2virt(addr,
                tran_size);
            vmtran->page_count_Ch += 1;
            rest_size -= tran_size;
        }
    }
}
```

Basic information of data transfer

Free translated result

```
struct vmaddr_tran {
    …
    _DWORD page_count_Ch;
    _QWORD tran_addr_10h;
    _QWORD tran_array_18h;
    …
};
```



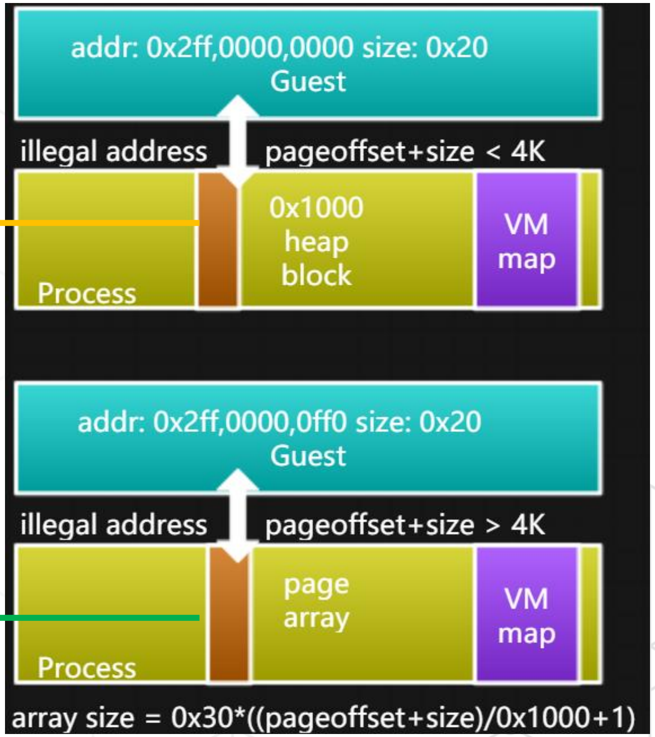
```
void vm_addr_translate_free(vmaddr_tran* vmtran)
{
    if(vmtran->page_count_Ch == 1){
        if(vmtran->tran_array_18h = -7){
            if(vmtran->tran_addr_10h != physMem[2071]){
                free(vmtran->tran_addr_10h);
            }
        }
    }else{
        free(vmtran->tran_array_18h);
    }
}
```

Structure **vmaddr_tran** will be cleaned by vm_addr_translate_free.

Basic information of data transfer

Examples

```
struct vmaddr_tran {
    …
    _DWORD page_count_Ch;
    _QWORD tran_addr_10h;
    _QWORD tran_array_18h;
    …
};
```



Heap Spray

We can use SVGA’s shader buffer to stores controlled data with controlled size. The number of this buffer is almost unlimited.

We can allocate it by svga command SVGA_3D_CMD_SET_SHADER

Notes: the details of how to send a svga command, you can watch this “Straight outta Vmware, Zisis Sialveras”

R/W related structures

○ SVGA MOB structure

+0x50 **guestbuffer**;// = vmaddr_tran->tran_addr_10h

+0x54 size;// size of guestbuffer

SVGA command *SVGA_3D_CMD_DX_SURFACE_COPY_AND_READBACK* allows us to copy data between mobs.

○ vmxnet3 mfTable

it can be used to write an arbitrary data from guest to a process heap. We can control its allocation and release.

R/W related structures

○ SVGA GMR buffer

It’s a MKS heap with tag 0xA0017. Each MKS heap has an extra heap header.

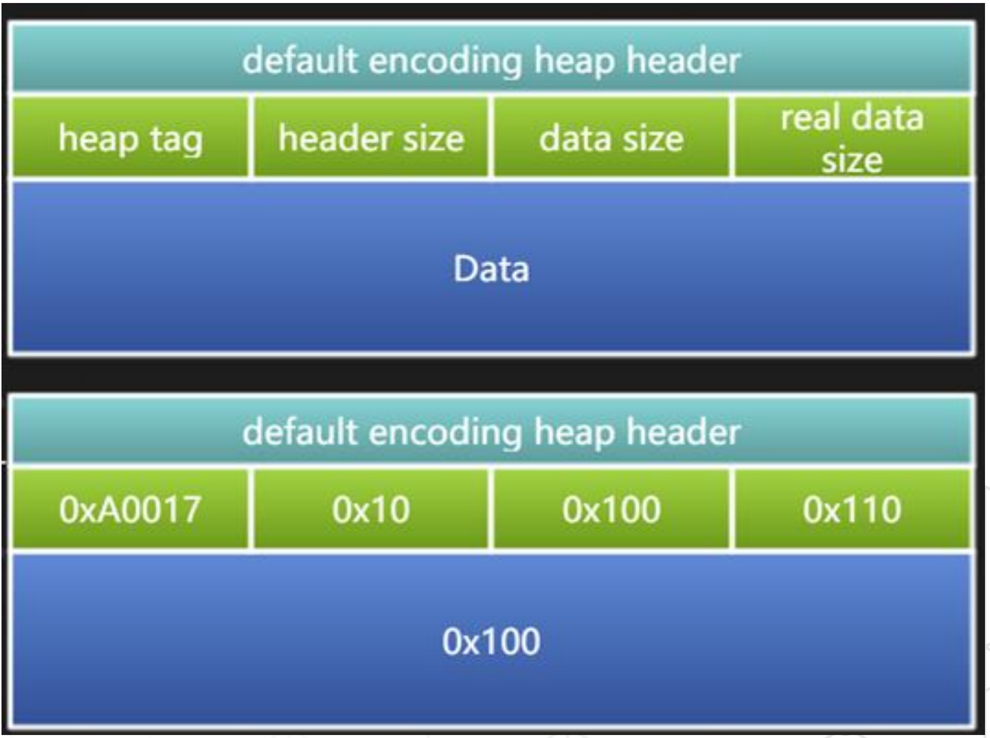
Calculate real heap header:

Heap header = buff - *(u32 *)(buff-0xc)

MKSheader -->

Real heap header -->

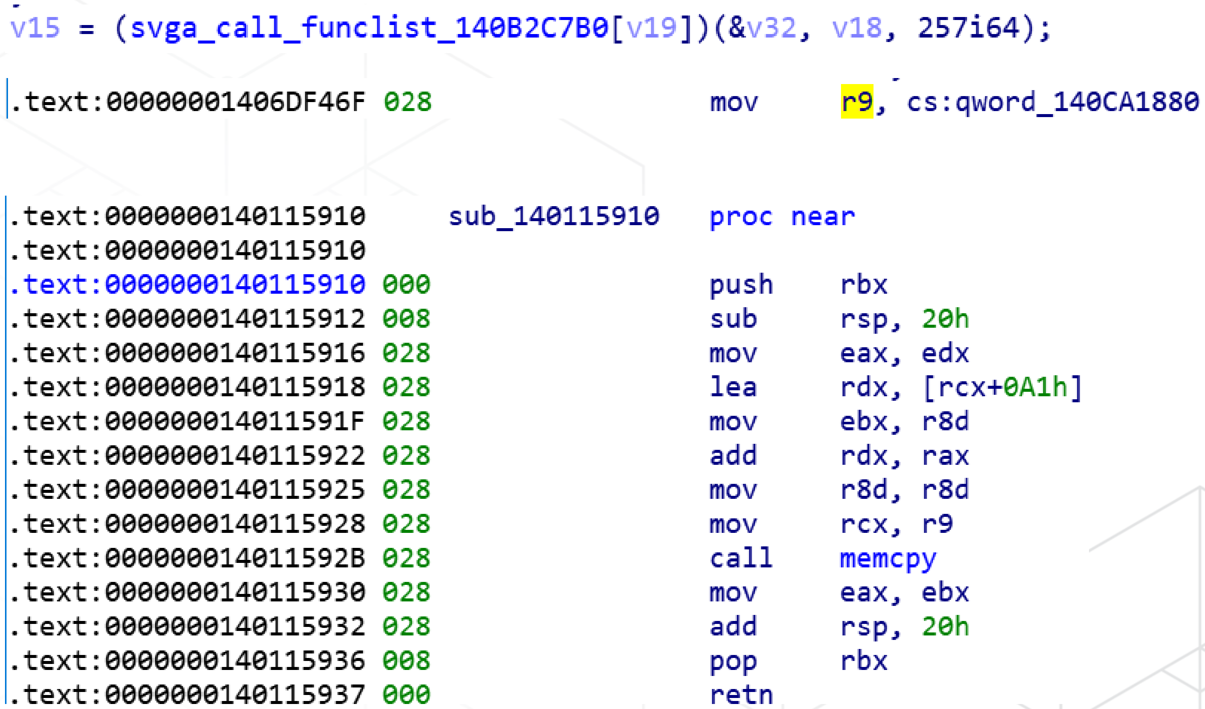
Return buffer address -->



Bypass CFG

Base on 15.0.1

1. change dynamic function list to function 0x1406DF450 which let R9 points to a variable at 0x140ca1880 of .rdata segment.



```
v15 = (svga_call_funclist_140B2C7B0[v19])(&v32, v18, 257i64);
```

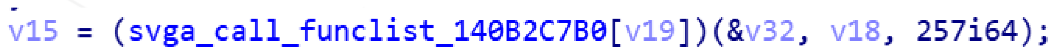
```
.text:00000001406DF46F 028                 mov     r9, cs:qword_140CA1880
```

```
.text:0000000140115910     sub_140115910   proc near
.text:0000000140115910
.text:0000000140115910 000                 push    rbx
.text:0000000140115912 008                 sub     rsp, 20h
.text:0000000140115916 028                 mov     eax, edx
.text:0000000140115918 028                 lea     rdx, [rcx+0A1h]
.text:000000014011591F 028                 mov     ebx, r8d
.text:0000000140115922 028                 add     rdx, rax
.text:0000000140115925 028                 mov     r8d, r8d
.text:0000000140115928 028                 mov     rcx, r9
.text:000000014011592B 028                 call    memcpy
.text:0000000140115930 028                 mov     eax, ebx
.text:0000000140115932 028                 add     rsp, 20h
.text:0000000140115936 008                 pop     rbx
.text:0000000140115937 000                 retn
```

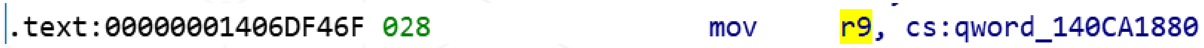
Bypass CFG

Base on 15.0.1

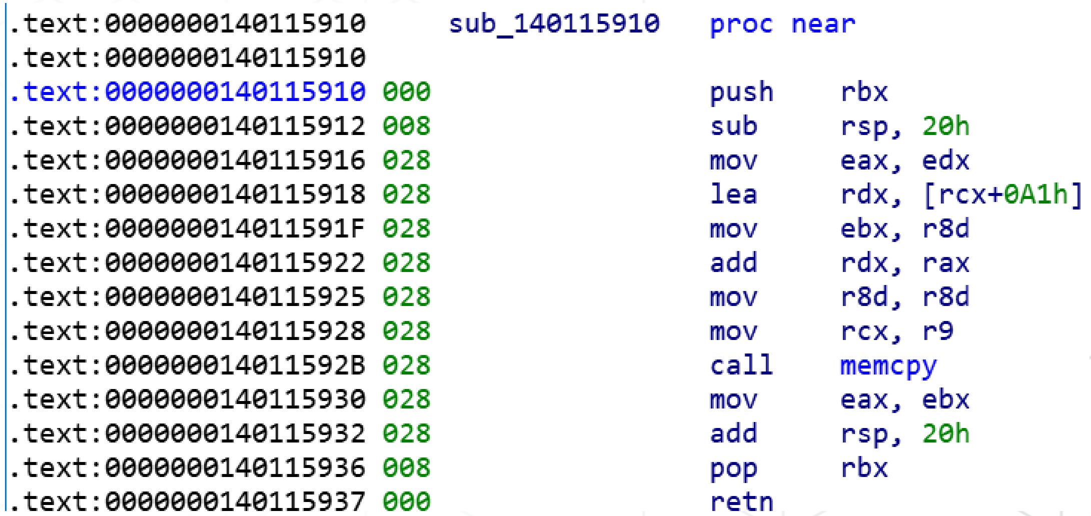
2. change pointer to function 0x140115910, It will save data of a1 to where the pointer in r9 indicates.



```
v15 = (svga_call_funclist_140B2C7B0[v19])(&v32, v18, 257i64);
```



```
.text:00000001406DF46F 028                 mov     r9, cs:qword_140CA1880
```



```
.text:0000000140115910     sub_140115910   proc near
.text:0000000140115910
.text:0000000140115910 000                 push    rbx
.text:0000000140115912 008                 sub     rsp, 20h
.text:0000000140115916 028                 mov     eax, edx
.text:0000000140115918 028                 lea     rdx, [rcx+0A1h]
.text:000000014011591F 028                 mov     ebx, r8d
.text:0000000140115922 028                 add     rdx, rax
.text:0000000140115925 028                 mov     r8d, r8d
.text:0000000140115928 028                 mov     rcx, r9
.text:000000014011592B 028                 call    memcpy
.text:0000000140115930 028                 mov     eax, ebx
.text:0000000140115932 028                 add     rsp, 20h
.text:0000000140115936 008                 pop     rbx
.text:0000000140115937 000                 retn
```

Attack Case of ESXi

○ Bug

○ Uninitialized to UAF

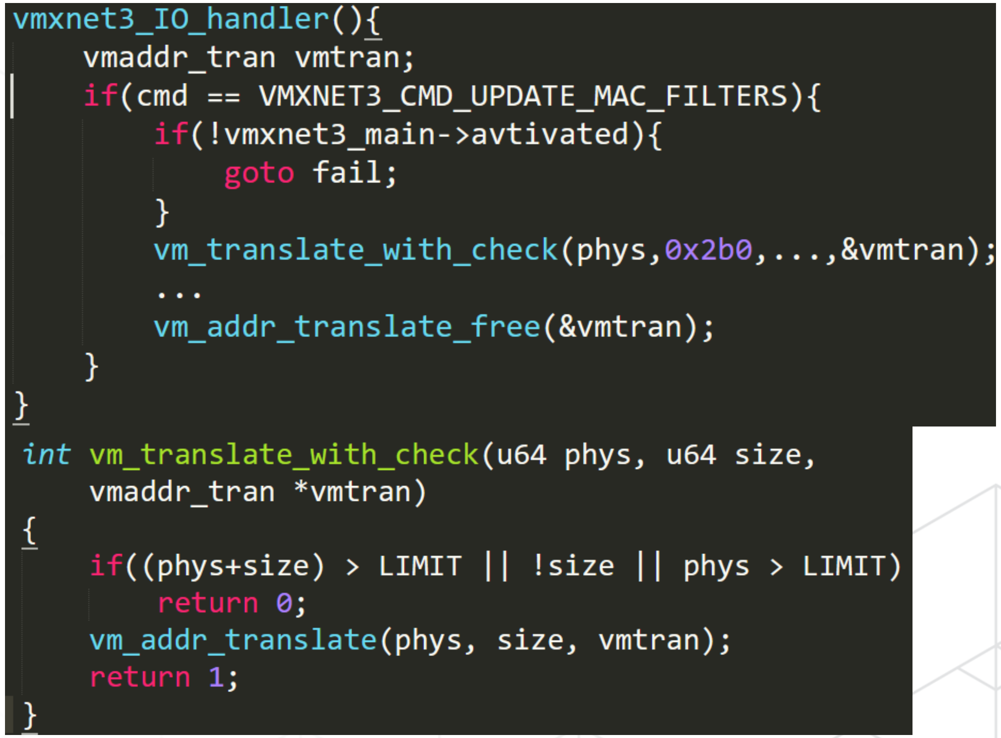
○ R/W everywhere

○ Control rip

based on ESXi-ver8941472

Bug: Uninitialized variable

- ○ Vmtran is a stack variable of structure vmaddr_tran
- ○ When handling command VMXNET3_CMD_UPDATE_MAC_FILTERS, it doesn’t check return value

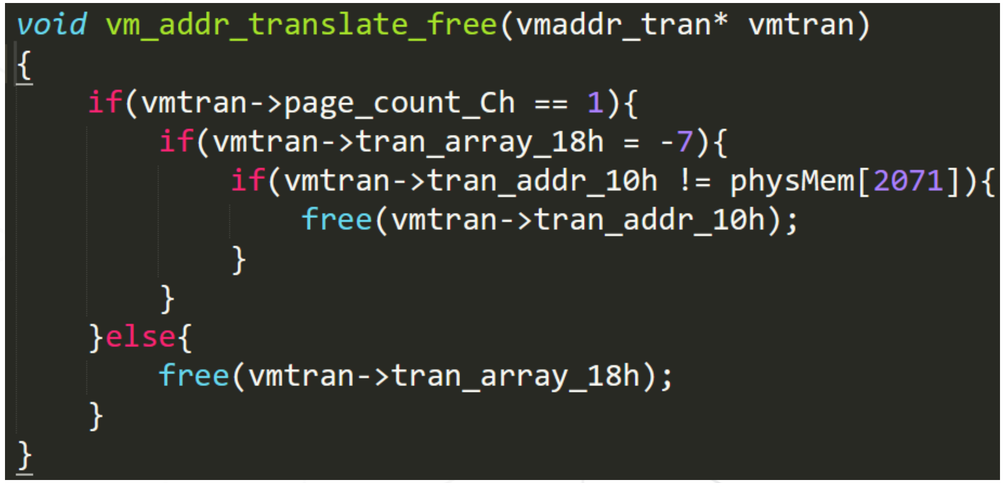


```
vmxnet3_IO_handler(){
    vmaddr_tran vmtran;
    if(cmd == VMXNET3_CMD_UPDATE_MAC_FILTERS){
        if(!vmxnet3_main->avtivated){
            goto fail;
        }
        vm_translate_with_check(phys,0x2b0,...,&vmtran);
        ...
        vm_addr_translate_free(&vmtran);
    }
}
```

```
int vm_translate_with_check(u64 phys, u64 size,
    vmaddr_tran *vmtran)
{
    if((phys+size) > LIMIT || !size || phys > LIMIT)
        return 0;
    vm_addr_translate(phys, size, vmtran);
    return 1;
}
```

Bug: Uninitialized variable

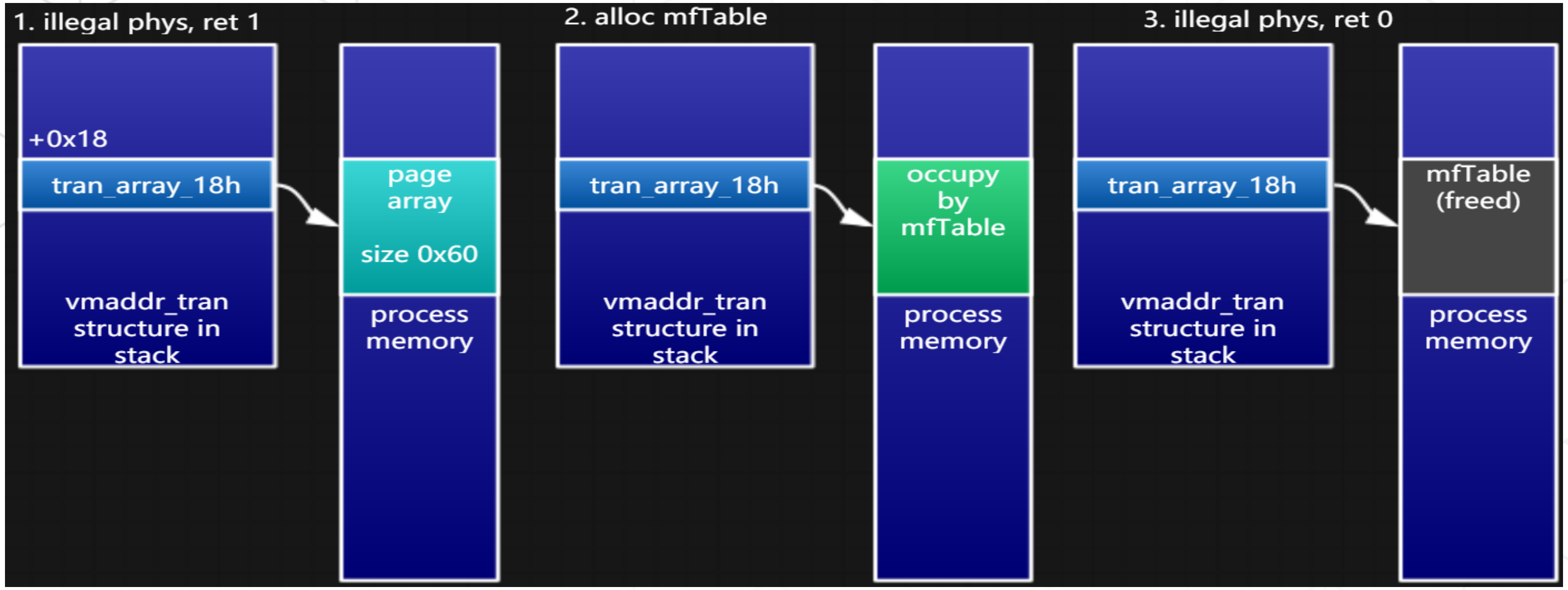
- Vmtran is a stack variable of structure vmaddr_tran
- When handling command VMXNET3_CMD_UPDATE_MAC_FILTERS, it doesn’t check return value



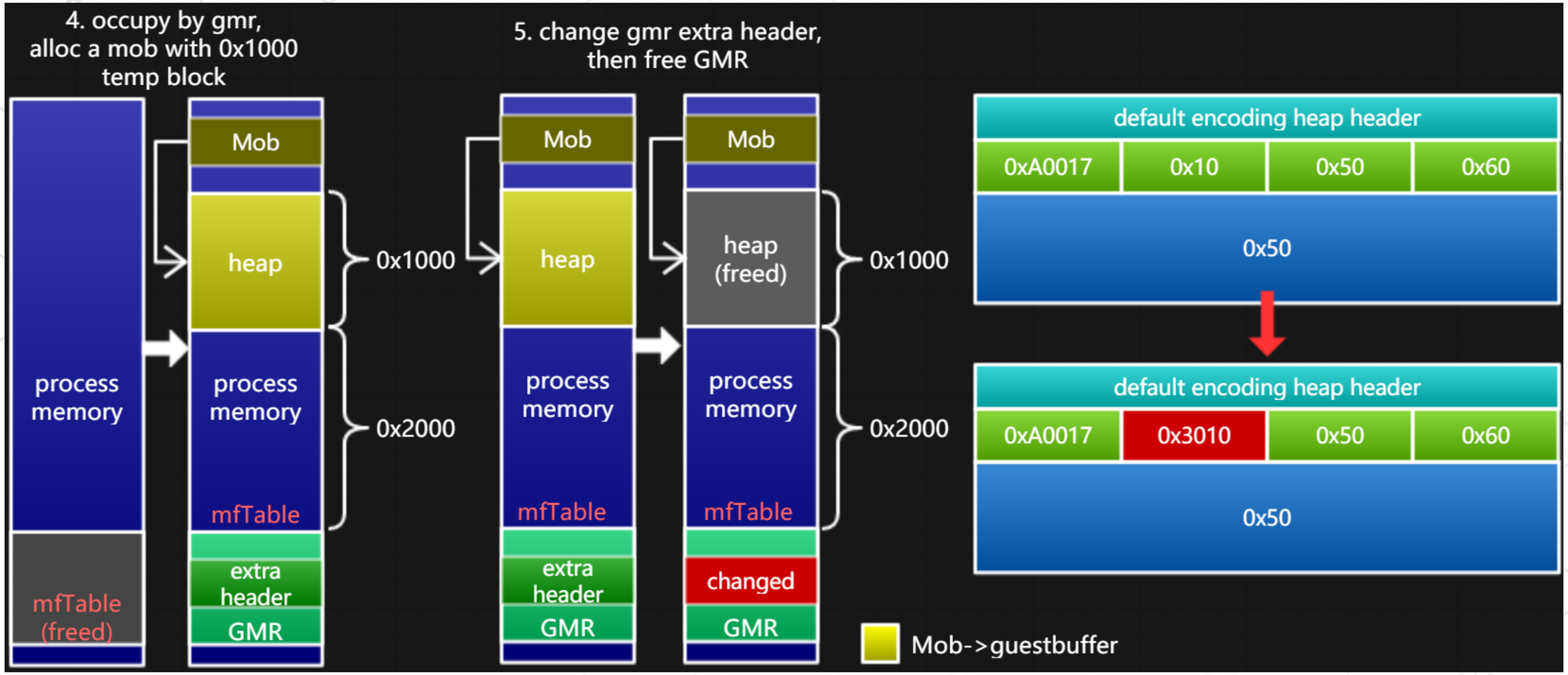
```
void vm_addr_translate_free(vmaddr_tran* vmtran)
{
    if(vmtran->page_count_Ch == 1){
        if(vmtran->tran_array_18h = -7){
            if(vmtran->tran_addr_10h != physMem[2071]){
                free(vmtran->tran_addr_10h);
            }
        }
    }else{
        free(vmtran->tran_array_18h);
    }
}
```

Transfer BUG to UAF

- In step 1.Addr = 0x2FF,XXXX,XF00; size is 0x2B0; array size = 0x30 * ((0xF00+0x2B0-1)/0x1000+1) = 0x60

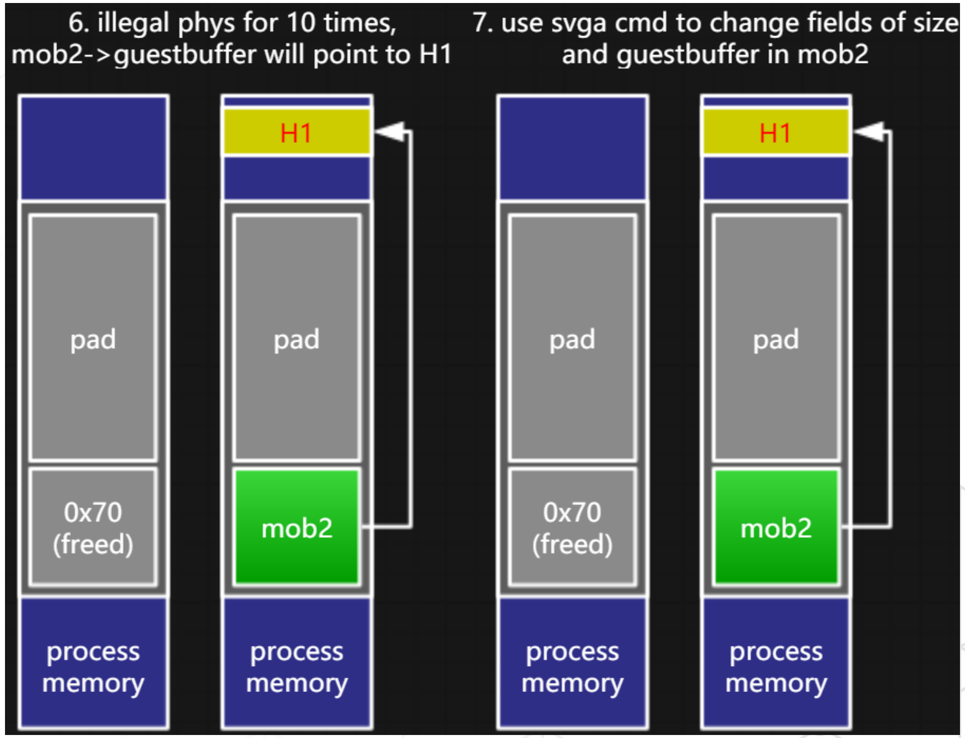


R/W everywhere

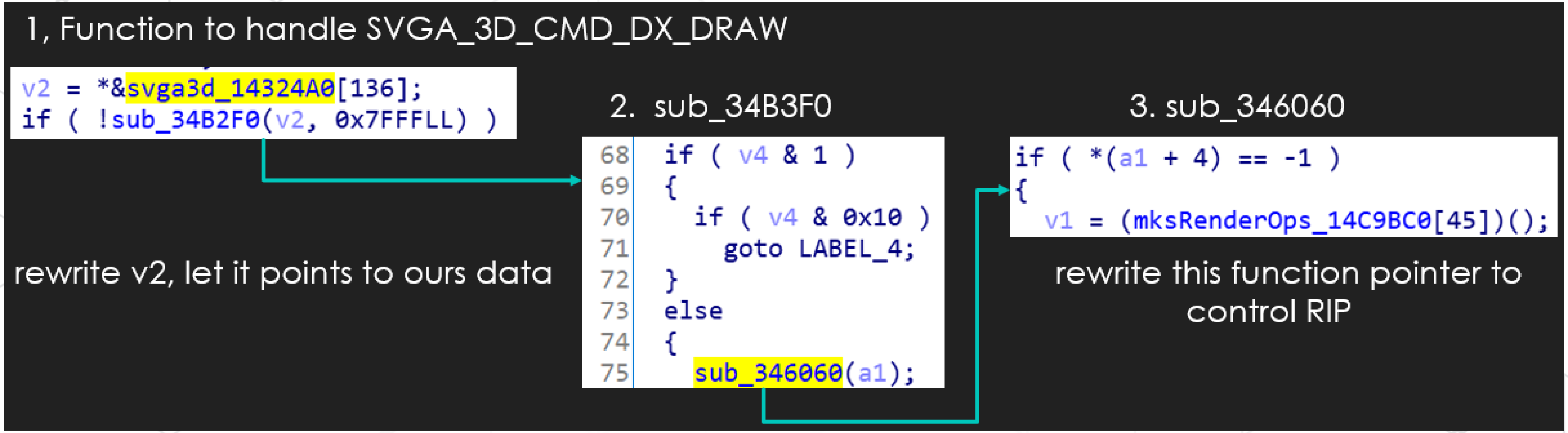


R/W everywhere

- ○ Pad 0x1000-0x70 memory, let heap split a 0x70 block.
- ○ Address translation fails over 9 times, then H1 is returned.
- ○ Use mob1 to change mob2's size.
- ○ Use SVGA command to read and write data from a normal mob to mob2.


6. illegal phys for 10 times, mob2->guestbuffer will point to H1
7. use svga cmd to change fields of size and guestbuffer in mob2
H1
H1
pad
pad
pad
pad
0x70 (freed)
mob2
0x70 (freed)
mob2
process memory
process memory
process memory
process memory

Control RIP



1, Function to handle SVGA_3D_CMD_DX_DRAW

```
v2 = *&svga3d_14324A0[136];
if ( !sub_34B2F0(v2, 0x7FFFLL) )
```

rewrite v2, let it points to ours data

2. sub_34B3F0

```
if ( v4 & 1 )
{
  if ( v4 & 0x10 )
    goto LABEL_4;
}
else
{
  sub_346060(a1);
```

3. sub_346060

```
if ( *(a1 + 4) == -1 )
{
  v1 = (mksRenderOps_14C9BC0[45])();
```

rewrite this function pointer to control RIP

Attack Case of Workstation

○ Bug

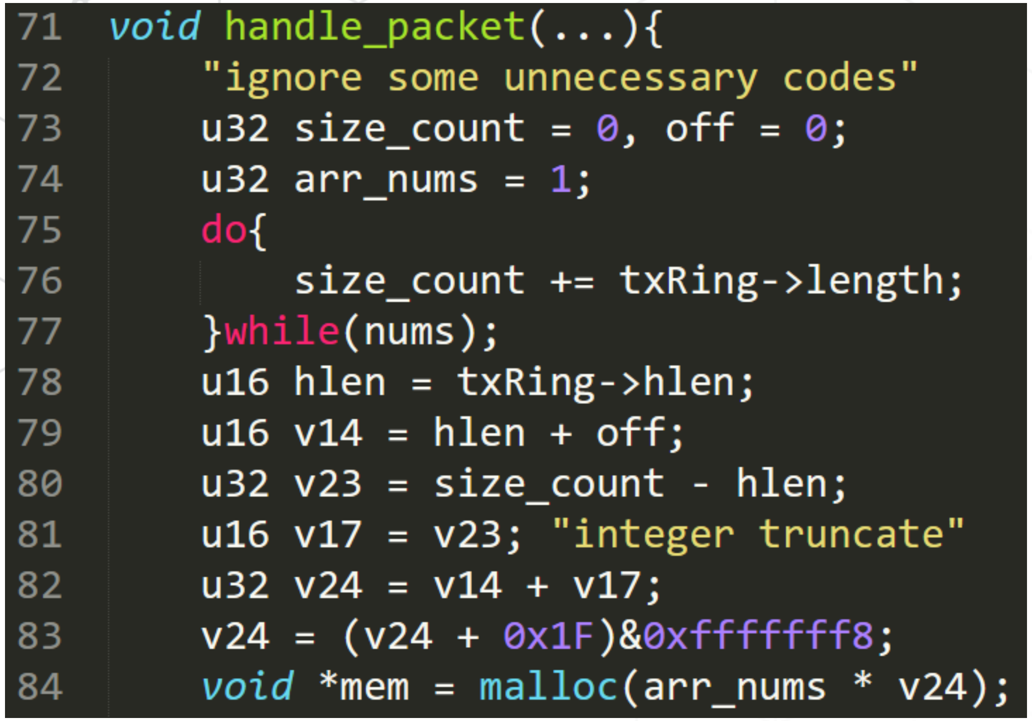
○ Leak information

○ R/W everywhere

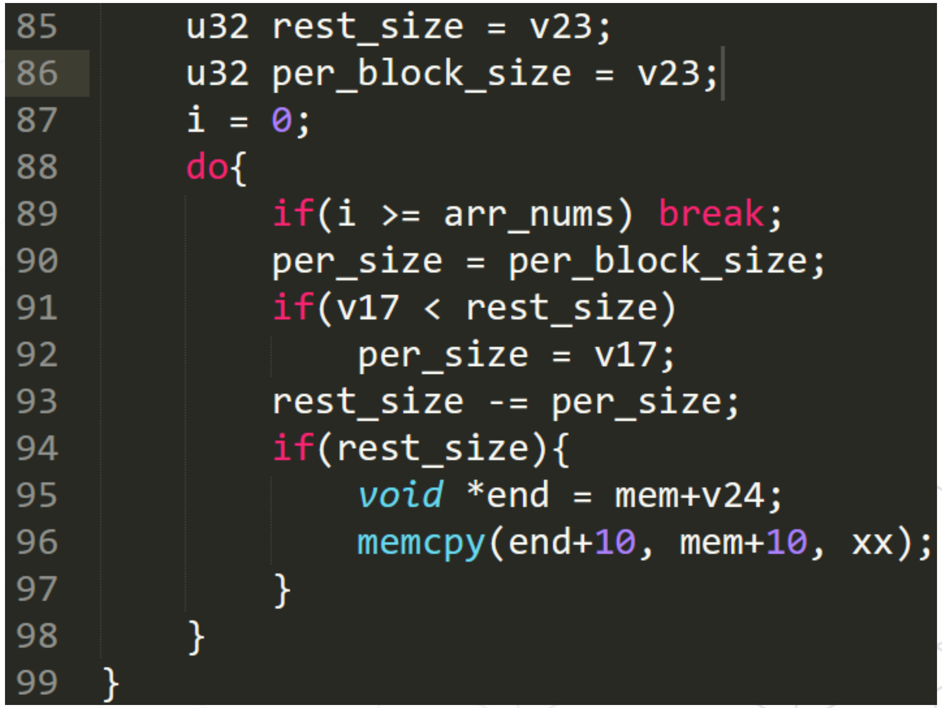
○ Bypass CFG

based on workstation 15.0.1

Bug: Integer Truncated



```
void handle_packet(...){
    "ignore some unnecessary codes"
    u32 size_count = 0, off = 0;
    u32 arr_nums = 1;
    do{
        size_count += txRing->length;
    }while(nums);
    u16 hlen = txRing->hlen;
    u16 v14 = hlen + off;
    u32 v23 = size_count - hlen;
    u16 v17 = v23; "integer truncate"
    u32 v24 = v14 + v17;
    v24 = (v24 + 0x1F)&0xfffffff8;
    void *mem = malloc(arr_nums * v24);
```



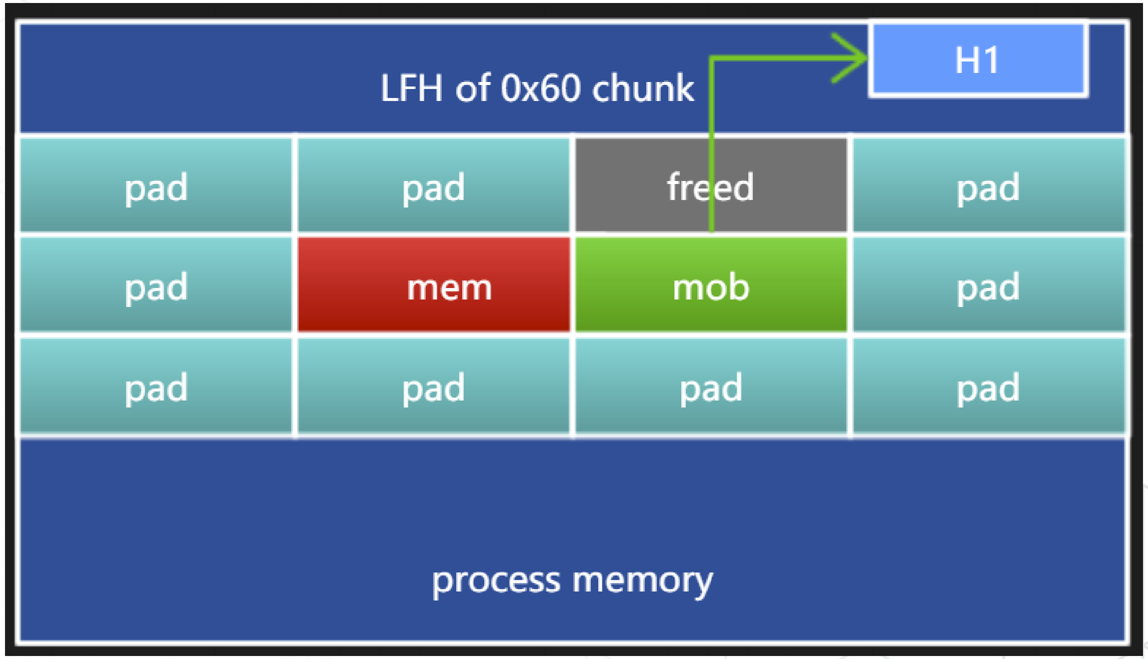
```
    u32 rest_size = v23;
    u32 per_block_size = v23;
    i = 0;
    do{
        if(i >= arr_nums) break;
        per_size = per_block_size;
        if(v17 < rest_size)
            per_size = v17;
        rest_size -= per_size;
        if(rest_size){
            void *end = mem+v24;
            memcpy(end+10, mem+10, xx);
        }
    }
}
```

Leak Information

Leak process related Addr

- ○ Allocate many 0x60 blocks and try to free several blocks. It has a good possibility that mem and mob are adjacent.

- ○ Overflow mem to overwrite mob's size, then use svga command to overflow read to leak process related address from the memory after H1.



R/W everywhere

Fake a moblist

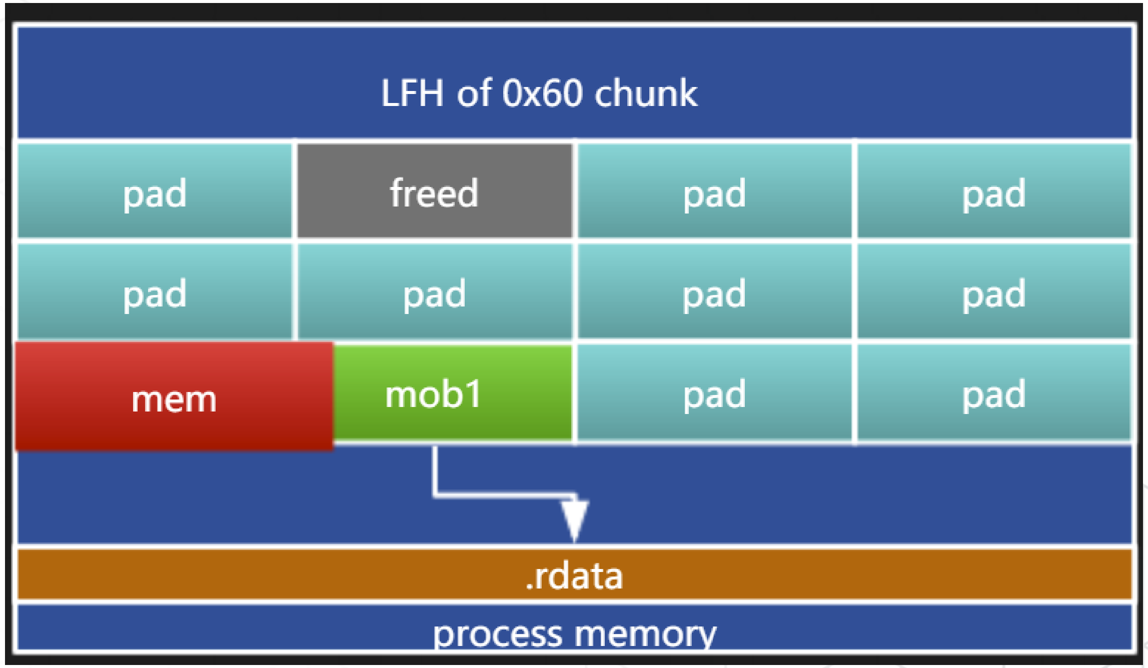
○ Overflow again

mob1->guestbuffer => moblist of .rdata segment.

○ Fake a moblist

mob1->guestbuffer => svgaFifoCmdScratchSpace (It's a svga command buffer at .rdata segment).

○ Use cmd SVGA_3D_CMD_PRESENT to write data to svgaFifoCmdScratchSpace.



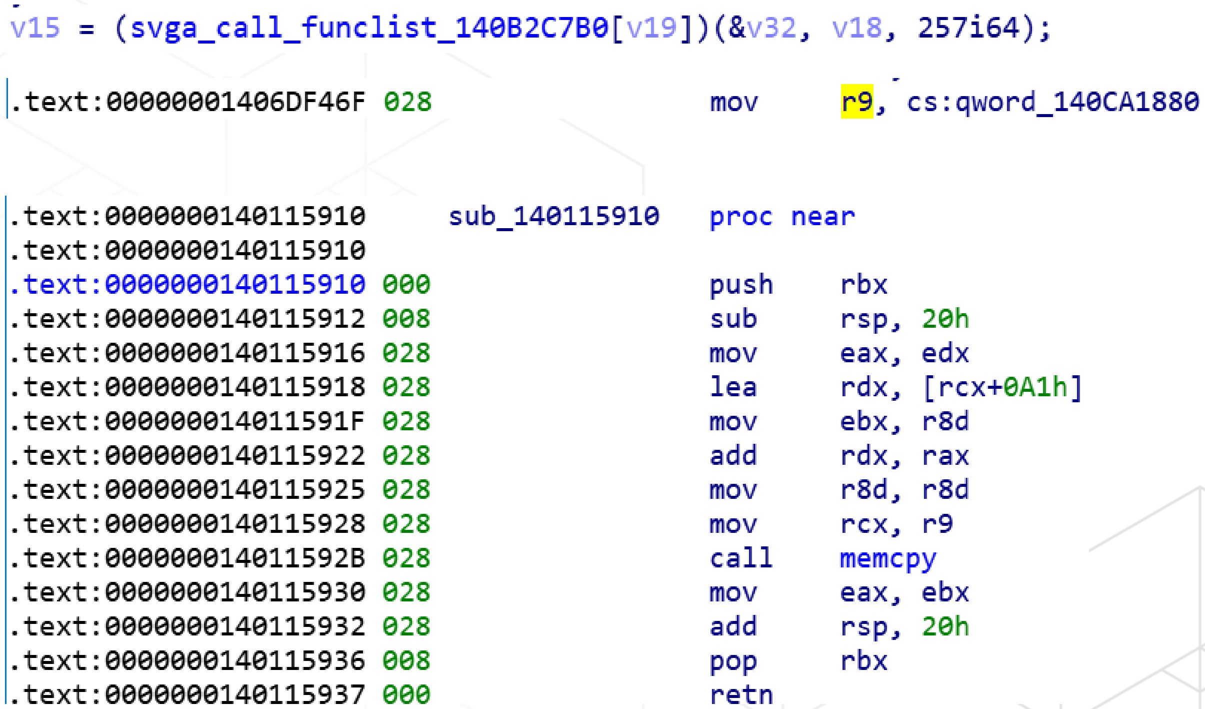
R/W everywhere

Fake mobs to r/w between Guest with process

- Fake mob points to VMmap offset
- SVGA_3D_CMD_SURFACE_COPY to read data from a mob to a svga buffer
- SVGA_3D_CMD_SURFACE_DMA to read data from a svga buffer to VM’s memory
- Faking two mobs. one points to somewhere we want to r/w, one points to our VM’s memory.

Bypass CFG

Use this skill to bypass CFG

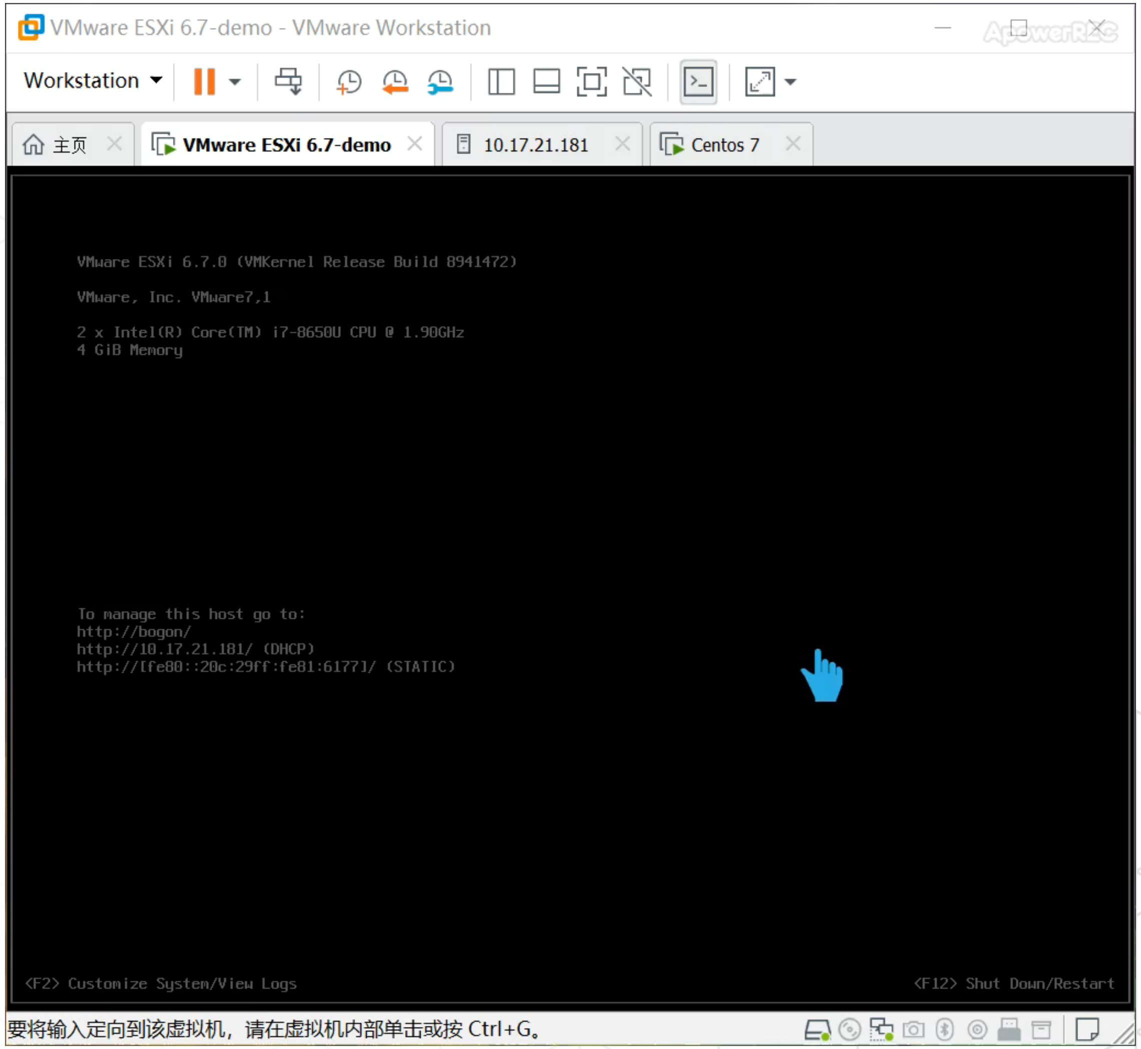


```
v15 = (svga_call_funclist_140B2C7B0[v19])(&v32, v18, 257i64);
```

```
.text:00000001406DF46F 028                 mov     r9, cs:qword_140CA1880
```

```
.text:0000000140115910     sub_140115910   proc near
.text:0000000140115910
.text:0000000140115910 000                 push    rbx
.text:0000000140115912 008                 sub     rsp, 20h
.text:0000000140115916 028                 mov     eax, edx
.text:0000000140115918 028                 lea     rdx, [rcx+0A1h]
.text:000000014011591F 028                 mov     ebx, r8d
.text:0000000140115922 028                 add     rdx, rax
.text:0000000140115925 028                 mov     r8d, r8d
.text:0000000140115928 028                 mov     rcx, r9
.text:000000014011592B 028                 call    memcpy
.text:0000000140115930 028                 mov     eax, ebx
.text:0000000140115932 028                 add     rsp, 20h
.text:0000000140115936 008                 pop     rbx
.text:0000000140115937 000                 retn
```

Demo of ESXi



```
VMware ESXi 6.7.0 (VMKernel Release Build 8941472)

VMware, Inc. VMware7,1

2 x Intel(R) Core(TM) i7-8650U CPU @ 1.90GHz
4 GiB Memory




To manage this host go to:
http://bogon/
http://10.17.21.181/ (DHCP)
http://[fe80::20c:29ff:fe81:6177]/ (STATIC)



<F2> Customize System/View Logs                                              <F12> Shut Down/Restart
```

要将输入定向到该虚拟机，请在虚拟机内部单击或按 Ctrl+G。

Conclusion

○ Programmers should care about the returned function results.

○ Creating an extra heap header without encoding is not a smart idea.

○ Manufactures should add modern mitigation measures to their products.

○ VM escape is not as hard as we expect.

Virtualization security is still a serious problem at present. We should be careful ☺

New Changes

To avoid to abuse mob structure, VMware Workstation 15.5.x stores mob structures in .rdata segment instead of allocating a heap. But other primitives still work.

It’s easy to find a similar structure in svga ;)

Thank You!

HITB LOCKDOWN 002 livestream

VictorV(vv.ttw@outlook.com)   Xiao Wei(xiaowei-c@360.cn)