qemu-pwn-DefconQuals-2018-EC3

raycp / 2019-11-21 09:15:24 /

这题是qemu逃逸是一道堆题，实际环境的堆题还是和普通的pwn题有一定区别的，同时这题还是把符号去掉了，增加了逆向的难度。

描述

在官方的描述中，还是逃逸读flag。

```
there's a vulnerable PCI device in the qemu binary. players have to write a kernel driver for the ubuntu kernel that is there
```

文件下载下来以后，文件结构如下：

```
$ ll
-rw-r--r-- 1 raycp raycp 256K May 10  2018 bios-256k.bin
-rw-r--r-- 1 raycp raycp 235K May 10  2018 efi-e1000.rom
-rw-rw-r-- 1 raycp raycp 1.8M Aug 13 19:10 initramfs-busybox-x86_64.cpio.gz
-rw-r--r-- 1 raycp raycp 9.0K May 10  2018 kvmvapic.bin
-rw-r--r-- 1 raycp raycp 1.5K May 10  2018 linuxboot_dma.bin
-rwxr-xr-x 1 raycp raycp  13M May 11  2018 qemu-system-x86_64
-rwxr-xr-x 1 raycp raycp  170 May 10  2018 run.sh
-rw-r--r-- 1 raycp raycp  38K May 10  2018 vgabios-stdvga.bin
-rw------- 1 raycp raycp 6.9M May 10  2018 vmlinuz-4.4.0-119-generic
```

`run.sh`里面的内容是：

```
#!/bin/sh
./qemu-system-x86_64 -initrd ./initramfs-busybox-x86_64.cpio.gz -nographic -kernel ./vmlinuz-4.4.0-119-generic -append "priori
```

通过`-device ooo`知道了目标应该主要是ooo这个pci设备。

```
$ file qemu-system-x86_64
qemu-system-x86_64: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/l, for GNU/Lin
```

可以看到`qemu-system-x86_64`是`stripped`，符号是去掉了的。

分析

环境安装

我是在ubuntu18上面尝试`sudo ./run.sh`把虚拟机跑起来的，但是各种报错，折腾了很久才跑起来，因此在这里也记录一下。

一开始报错：

```
./qemu-system-x86_64: error while loading shared libraries: libiscsi.so.2: cannot open shared object file: No such file or dir
```

解决办法，安装`libiscsi`：

```
git clone https://github.com/sahlberg/libiscsi.git
./autogen.sh
./configure
make
sudo make install
cp /usr/lib/x86_64-linux-gnu/libiscsi.so.7 /lib/libiscsi.so.2
```

在运行`./autogen.sh`的时候，报错：

```
configure.ac:9: error: possibly undefined macro: AC_PROG_LIBTOOL
```

解决方法，安装`libtool`和`libsysfs-dev`：

```
sudo apt-get install libtool
sudo apt-get install libsysfs-dev
```

安装完`libiscsi`后，再跑`sudo ./run.sh`，仍然报错：

```
./qemu-system-x86_64: error while loading shared libraries: libpng12.so.0: cannot open shared object file: No such file or dir
```

解决方法，安装libpng12：

```
sudo wget -O /tmp/libpng12.deb http://mirrors.kernel.org/ubuntu/pool/main/libp/libpng/libpng12-0_1.2.54-1ubuntu1_amd64.deb
sudo dpkg -i /tmp/libpng12.deb
sudo rm /tmp/libpng12.deb
```

再跑run.sh，报错：

```
./qemu-system-x86_64: error while loading shared libraries: libxenctrl-4.6.so: cannot open shared object file: No such file or
```

解决方法，安装libxen4.6：

```
sudo wget  -O /tmp/libxen.deb http://mirrors.kernel.org/ubuntu/pool/main/x/xen/libxen-4.6_4.6.5-0ubuntu1.4_amd64.deb
sudo dpkg -i /tmp/libxen.deb
sudo rm /tmp/libxen.deb
```

然后终于可以运行起来了。。。。

```
sudo ./run.sh
...
[    3.609675] Write protecting the kernel read-only data: 14336k
[    3.615441] Freeing unused kernel memory: 1696K
[    3.618437] Freeing unused kernel memory: 100K

Boot took 3.82 seconds


break out of the vm, but don't forget to have fun!

/bin/sh: can't access tty; job control turned off
/ # [    4.444675] clocksource: Switched to clocksource tsc

/ #
```

逆向分析

把qemu-system-x86_64拖进ida进行分析，由于符号去掉了，所以不能像之前一样直接搜索ooo相关的函数来寻找设备函数。

因此为了将该设备相关的函数和结构体找出来，我对照的是edu.c以及hitb2018
babyqemu的idb文件，通过ooo_class_init字符串定位0x6E67DE地址的函数为ooo_class_init；确定0x6E64A5函数为pci_ooo_realize；确定0x47D731函数为

通过pci_ooo_realize函数可以确定mmio的空间大小为0x1000000。

接下来详细分析ooo_mmio_write函数以及ooo_mmio_read函数。

首先是ooo_mmio_write函数，关键代码如下：

```
__int64 __fastcall ooo_mmio_read(struct_a1 *a1, int addr, unsigned int size)
{
 unsigned int idx; // [rsp+34h] [rbp-1Ch]
 __int64 dest; // [rsp+38h] [rbp-18h]
 struct_a1 *v6; // [rsp+40h] [rbp-10h]
 unsigned __int64 v7; // [rsp+48h] [rbp-8h]

 v7 = __readfsqword(0x28u);
 v6 = a1;
 dest = 0x42069LL;
 idx = (addr & 0xF0000u) >> 16;
 if ( (addr & 0xF00000u) >> 20 != 15 && global_buf[idx] )
   memcpy(&dest, (char *)global_buf[idx] + (signed __int16)addr, size);
 return dest;
}
```

可以看到(addr & 0xF0000u)为idx，addr的低16位为offset。当(addr & 0xF00000u) >> 20不为15时，将global_buf[idx] +
offset中的数据拷贝出来赋值给dest，否则dest为0x42069，返回dest。

接着看ooo_mmio_write函数，代码如下：

```
void __fastcall ooo_mmio_write(struct_a1 *opaque, __int64 addr, __int64 value, unsigned int size)
{
 unsigned int cmd; // eax MAPDST
 int n[3]; // [rsp+4h] [rbp-3Ch]
```

```
__int16 v8; // [rsp+22h] [rbp-1Eh]
int i; // [rsp+24h] [rbp-1Ch]
unsigned int idx; // [rsp+2Ch] [rbp-14h] MAPDST

*(_QWORD *)n = value;
cmd = ((unsigned int)addr & 0xF00000) >> 20;
cmd = ((unsigned int)addr & 0xF00000) >> 20;
switch ( cmd )
{
  case 1u:
    free(global_buf[((unsigned int)addr & 0xF0000) >> 16]);
    break;
  case 2u:
    idx = ((unsigned int)addr & 0xF0000) >> 16;
    v8 = addr;
    memcpy((char *)global_buf[idx] + (signed __int16)addr, &n[1], size);
    break;
  case 0u:
    idx = ((unsigned int)addr & 0xF0000) >> 16;
    if ( idx == 15 )
    {
      for ( i = 0; i <= 14; ++i )
        global_buf[i] = malloc(8LL * *(_QWORD *)&n[1]);
    }
    else
    {
      global_buf[idx] = malloc(8LL * *(_QWORD *)&n[1]);
    }
    break;
}
}
```

从该函数中可以看出`addr & 0xF00000`为`cmd`，根据`cmd`进行相应的`case`选择。`addr & 0xF0000`为`idx`，这似乎变成了一个堆的菜单题：

1. cmd
   为0时，进行malloc分配，分配的size为传入的`value`值（IDA反编译出来的是value的高32位，看汇编代码可以确定为value的低32位），分配出来的指针保存到全局变量
2. cmd为1时，调用free函数释放掉`global_buf[idx]`。
3. cmd为2时，将value写入到`global_buf[idx] + offset`中。

很明显可以看到这里的`uaf`漏洞，释放了以后并没有清空指针，形成漏洞。

利用

因为是在ubuntu18上面跑的，`glibc2.27`有`tcache`，所以利用起来比较简单。

同时可以看到`sub_6E65F9`函数包含后门，该函数调用`system("cat ./flag")`。因此只要控制rip为`0x6E65F9`即可。

利用过程为：申请堆块并释放到tcache中，利用uaf将`tcache`的`fd`改为`free got`，连续申请，将`free got`申请出来并改写成`0x6E65F9`，最后触发free拿到flag。

有一点需要指出的是，由于qemu在启动过程中会形成很多堆块，使得管理链表中存在很多堆块，可能会导致控制释放的顺序与申请的顺序无法像预期的一样控制。我的解决

一开始exp中也遇到一个错误：`exp[85]: segfault at 7f9dbdfc6000 ip 0000000000400b8e`，查看了`400b8e`地址为：`0x400b8e <mmio_write_byte+31>: mov BYTE PTR [rdx],al`，是访存错误。意识到是我一开始mmap文件`/sys/devices/pci0000:00/0000:00:04.0/resource0`的size过小，导致`mmio_write`的时候访存越界，所以

同时题目当时的环境是ubuntu16，由于没有tcache，利用起来比较复杂。根据已有的wp，有两种解法：

1. 根据DefconQuals 2018 - EC3解法：申请0x70大小的堆块，利用fastbin attack将fd改到`global_buf`地址处，因为堆指针地址开头会为`0x7f`，所以可以绕过size检查，从而将`global_buf`申请出来，覆盖地址实现任意读写，再修改got地址
2. 根据EC3 write-up (DEF CON CTF 2018 Quals)解法：利用堆溢出，将堆中内容都覆盖成后门的地址，再利用命令`echo mem > /sys/power/state`将虚拟机休眠，唤醒的时候会劫持控制流拿到flag。

感觉如果没有后门以及开了PIE的话，也可以利用`mmio_read`先泄露libc地址和堆地址，再做利用也是可行的。

还有一点是如何将exp传入到虚拟机中，一种方式是将exp编译好后base64编码，粘贴到虚拟机中再解码。另一种是看到文件系统是`initramfs-busybox-x86_64.cpio`

```
gunzip initramfs-busybox-x86_64.cpio.gz
cpio -idmv < initramfs-busybox-x86_64.cpio
```

然后make将exp编译出来并重打包文件系统再启动qemu虚拟机，就可以看到exp在里面了，makefile内容如下：

```
ALL:
	gcc -O0 -static -o exp exp.c
	-rm ../initramfs-busybox-x86_64.cpio.gz
	#-rm ../initramfs-busybox-x86_64.cpio
	find . | cpio -o --format=newc > ../initramfs-busybox-x86_64.cpio
	cd .. && gzip initramfs-busybox-x86_64.cpio
```

小结

第一次看没有符号的题，还是有一定的挑战的，修复运行环境也搞了半天，学到了不少。

最后在github里面找到了题目的源码，逆了半天有点儿尴尬，不过看完没符号的反编译代码并尽量把它修复也是对自己的一点挑战吧。

相关脚本和文件链接

参考链接

1. DefconQuals 2018 - EC3
2. EC3 write-up (DEF CON CTF 2018 Quals)
3. oooverflow.c
4. linux系统的休眠与唤醒简介