京麒
网络安全大会

再看云虚拟化安全
QEMU通用漏洞挖掘新思路

主讲人：钱文祥　Tencent Blade Team高级安全研究员

钱文祥 (@leonwxqian) Tencent Blade Team高级安全研究员

《白帽子讲浏览器安全》作者 | 云、IoT、浏览器等安全研究

发现Amazon Echo、Google Home智能音箱及浏览器中"麦哲伦"SQLite、Curl的多个远程代码执行漏洞等

DEF CON 26 & 27、Blackhat 2019 USA、HITB 2020 Singapore等演讲者

- Tencent Blade Team由腾讯安全平台部在2017年底成立
- 专注于AIoT，移动互联网，云虚拟化技术，区块链等前沿领域的安全技术研究
- 向Google、Microsoft、 Apple、Amazon、 Huawei等诸多国际知名公司报告过200+安全漏洞
- 研究成果多次入选BlackHat、DEFCON、CanSecWest、HITB、POC、Xcon、CSS等顶级安全大会
- 团队官网：https://blade.tencent.com

目录

- 云虚拟化背景介绍
- 云虚拟化的安全探索
  - QEMU结构简介
  - Fuzzer的制作
  - 发现的安全问题
- 安全研究展望和企业Hypervisor开发安全思考

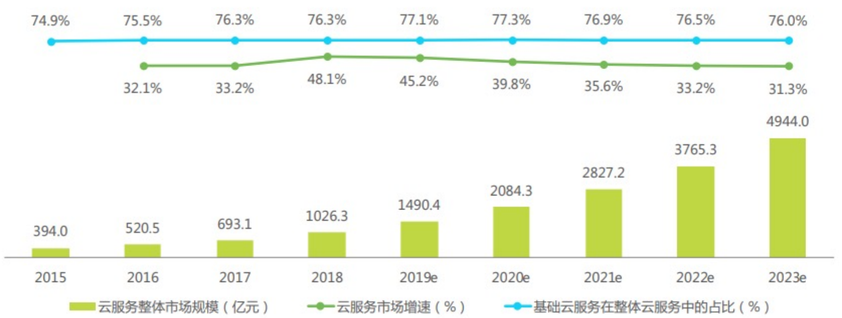
背景介绍

云服务处于快速增长期

- 2020 Q2 全球云服务产业市场增速进一步增大
- 我国整体云服务市场保持高速增长

  至2018年形成逾千亿（1026.3亿元）的市场体量
- AI、大数据、IoT等新兴科技为云服务带来了更加充足的想象空间

2015-2023年中国云服务市场规模与增速及基础云服务的市场占比变化

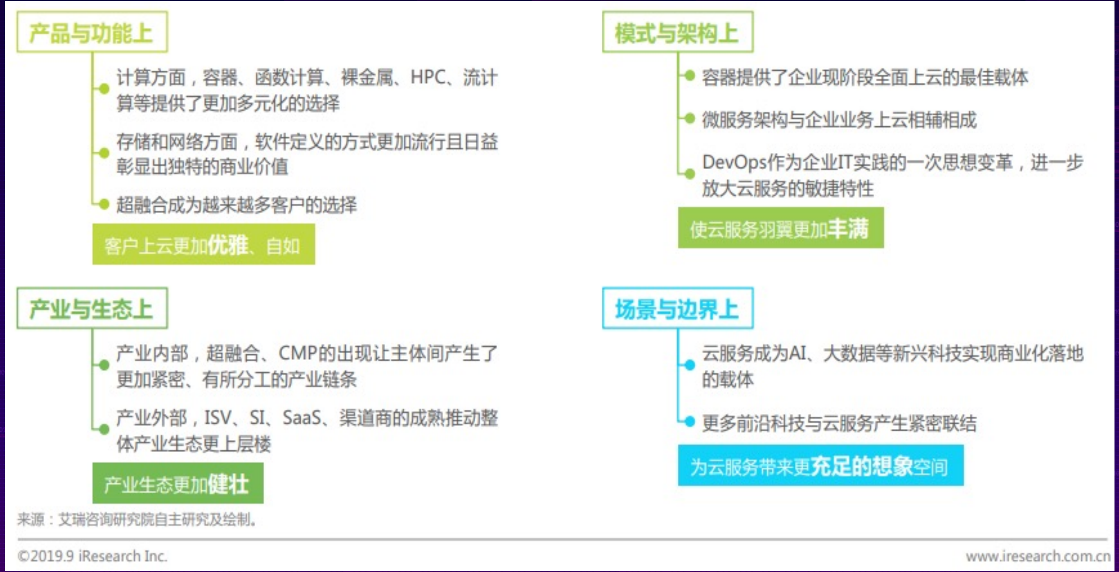


注释：云服务整体市场包含公有云、私有云、专有云、混合云等各类部署模式下的服务。
来源：根据公开资料、企业访谈，结合艾瑞统计模型核算。

©2019.9 iResearch Inc.   www.iresearch.com.cn

基础云服务不断发展演进

- 基于云或虚拟化的、以软件为主进行交付的IT基础资源服务
- 提供的是以计算、存储、网络等IT基础资源为核心能力的服务
- 虚拟化是基础云服务大环境中，许多服务的重要技术支撑



产品与功能上

- 计算方面，容器、函数计算、裸金属、HPC、流计算等提供了更加多元化的选择
- 存储和网络方面，软件定义的方式更加流行且日益彰显出独特的商业价值
- 超融合成为越来越多客户的选择

客户上云更加**优雅**、自如

模式与架构上

- 容器提供了企业现阶段全面上云的最佳载体
- 微服务架构与企业业务上云相辅相成
- DevOps作为企业IT实践的一次思想变革，进一步放大云服务的敏捷特性

使云服务羽翼更加**丰满**

产业与生态上

- 产业内部，超融合、CMP的出现让主体间产生了更加紧密、有所分工的产业链条
- 产业外部，ISV、SI、SaaS、渠道商的成熟推动整体产业生态更上层楼

产业生态更加**健壮**

场景与边界上

- 云服务成为AI、大数据等新兴科技实现商业化落地的载体
- 更多前沿科技与云服务产生紧密联结

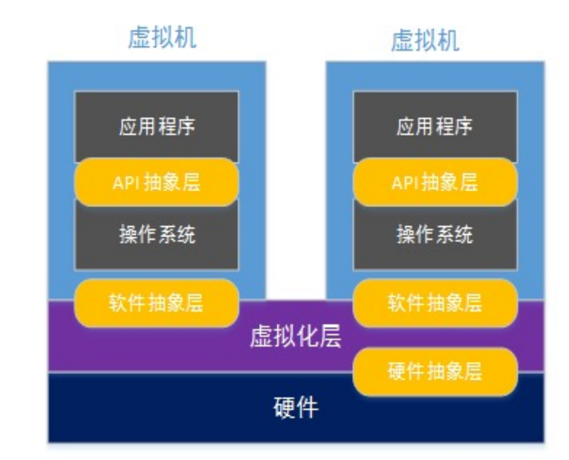
为云服务带来更**充足的想象**空间

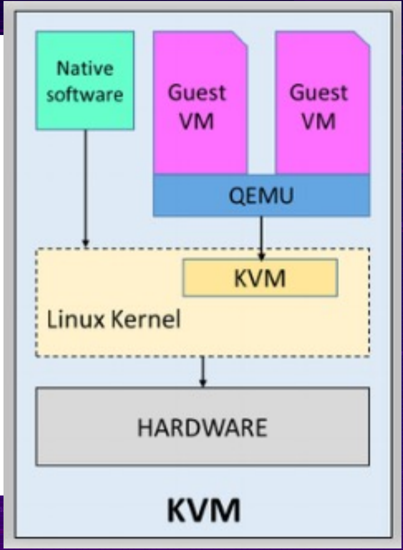
来源：艾瑞咨询研究院自主研究及绘制。

©2019.9 iResearch Inc. www.iresearch.com.cn

云虚拟化

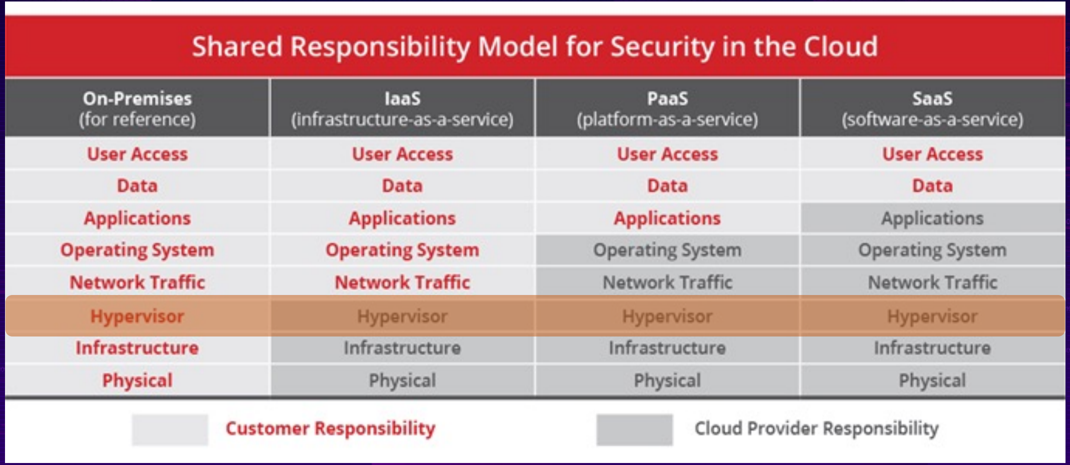
- 许多种云服务需要从一个物理硬件系统创建多个模拟环境
- 实现这个技术的核心便是虚拟化，具体则是Hypervisor层
- Hypervisor是一种运行在基础物理服务器和操作系统之间的中间软件层，可允许多个guest操作系统和应用共享硬件
- 典型的代表有：QEMU-KVM，Xen，VMware，Hyper-V等等





Hypervisor —— 必须守住的阵地

- Hypervisor是用户（guest）和云服务提供商的“边缘”
- Host是总管理员，上面运行大量虚拟化机器
- 用户看来，Guest就是一台电脑，用户可以完全控制guest
- 但是如果有漏洞允许用户从guest中穿透到host上，
  则会对云主机的隐私性、安全性造成极大危害



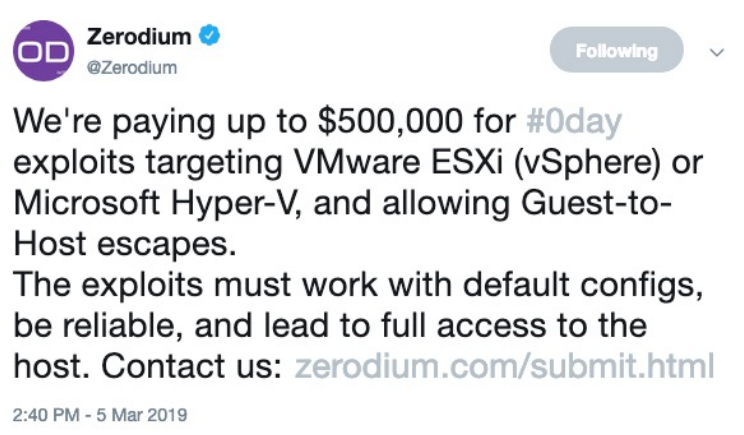
Shared Responsibility Model for Security in the Cloud

On-Premises (for reference)	IaaS (infrastructure-as-a-service)	PaaS (platform-as-a-service)	SaaS (software-as-a-service)
User Access	User Access	User Access	User Access
Data	Data	Data	Data
Applications	Applications	Applications	Applications
Operating System	Operating System	Operating System	Operating System
Network Traffic	Network Traffic	Network Traffic	Network Traffic
Hypervisor	Hypervisor	Hypervisor	Hypervisor
Infrastructure	Infrastructure	Infrastructure	Infrastructure
Physical	Physical	Physical	Physical

Customer Responsibility   Cloud Provider Responsibility

Hypervisor的安全一向受到各方重视

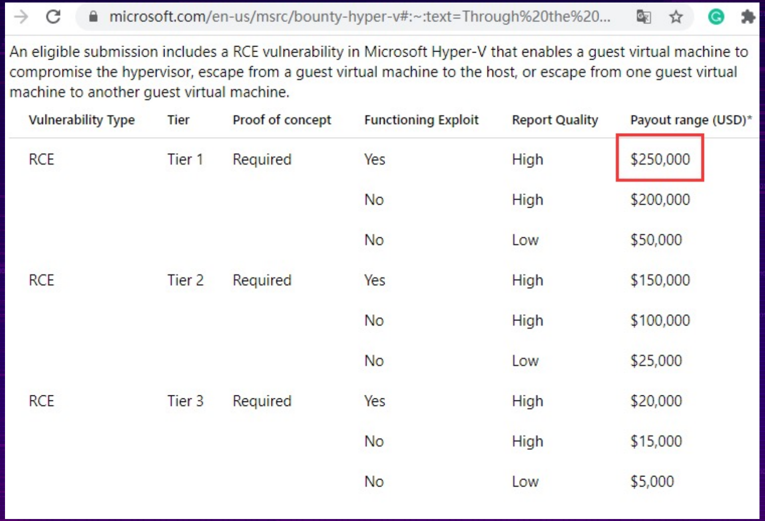
- 长期以来，Zerodium等企业高价收购Hypervisor的漏洞
- Microsoft等企业也开出高价鼓励安全研究人员上报漏洞



Zerodium
@Zerodium

We're paying up to $500,000 for #0day exploits targeting VMware ESXi (vSphere) or Microsoft Hyper-V, and allowing Guest-to-Host escapes.
The exploits must work with default configs, be reliable, and lead to full access to the host. Contact us: zerodium.com/submit.html

2:40 PM - 5 Mar 2019



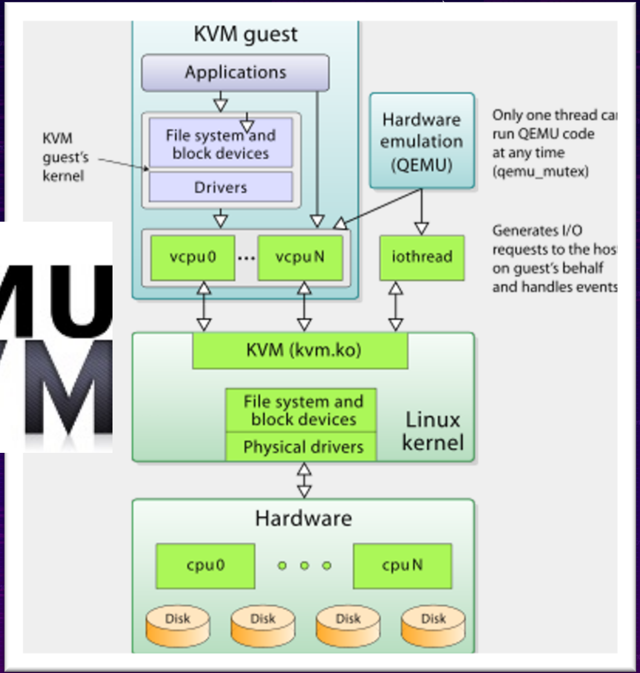
microsoft.com/en-us/msrc/bounty-hyper-v#:~:text=Through%20the%20...

An eligible submission includes a RCE vulnerability in Microsoft Hyper-V that enables a guest virtual machine to compromise the hypervisor, escape from a guest virtual machine to the host, or escape from one guest virtual machine to another guest virtual machine.

Vulnerability Type	Tier	Proof of concept	Functioning Exploit	Report Quality	Payout range (USD)*
RCE	Tier 1	Required	Yes	High	$250,000
			No	High	$200,000
			No	Low	$50,000
RCE	Tier 2	Required	Yes	High	$150,000
			No	High	$100,000
			No	Low	$25,000
RCE	Tier 3	Required	Yes	High	$20,000
			No	High	$15,000
			No	Low	$5,000

最热门的Hypervisor组合之一：QEMU + KVM

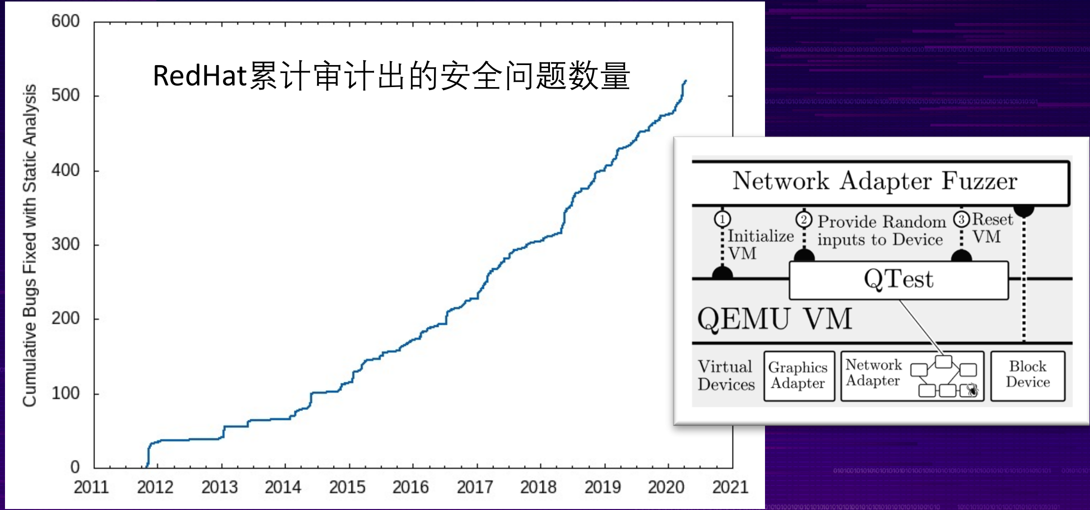
- QEMU允许使用KVM作为加速器，以便可以使用vCPU
- 通过直接在主机CPU上执行客户代码来实现近乎本机的性能
- QEMU: 用户态的Type 2 Hypervisor(即在主机操作系统上运行)
  - 用于应用硬件虚拟化
- KVM: Linux内核模块，Type 1 Hypervisor
  - Linux的完全虚拟化解决方案
  - vCPU的指令可以直接在物理CPU上执行



云虚拟化的安全探索

RedHat的安全尝试

- 定期运行的自动扫描可确保要合并的新代码没有已知缺陷
- 基于Qtest的QEMU Fuzz框架
- 接入谷歌的OSS-Fuzz
- Coverity等静态扫描发现了1,000多个错误
- Fuzzer发现了100多个错误
- https://www.redhat.com/en/blog/hardening-qemu-through-continuous-security-testing

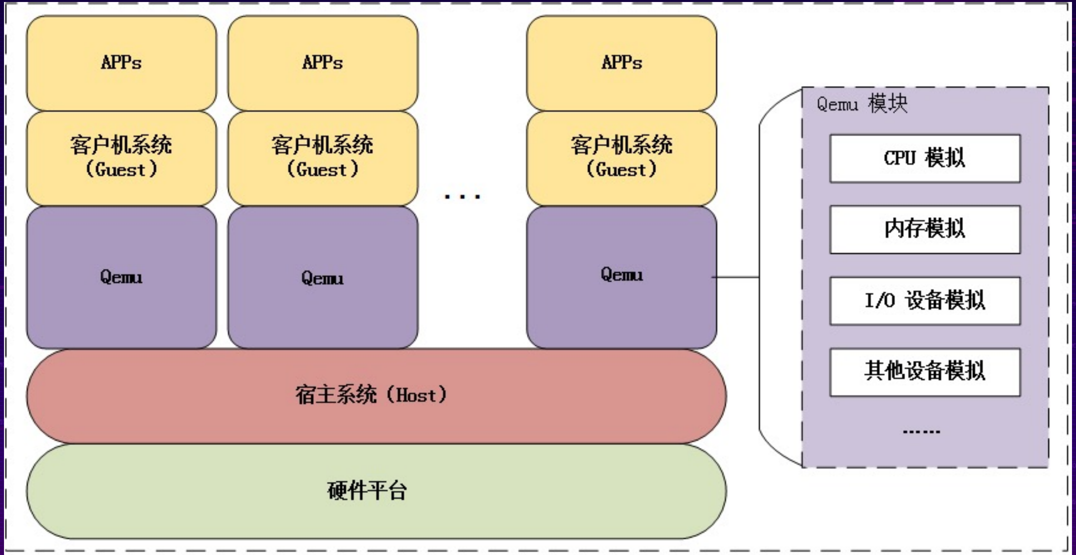


对QEMU Fuzz的一些早前研究

- When virtualization encounter AFL-A Portable virtual device fuzzing framework with AFL(2016)
- VDF: Targeted Evolutionary Fuzz Testing of Virtual Devices （ 2017)
- Virtio Device Fuzzing by Dmitrii Stepanov (AFL + virtio & SPDK, 2019)
- Qtests (Unittest) with libfuzzer in QEMU Projects (2020, https://fossies.org/linux/qemu/docs/devel/fuzzing.txt)
- HYPER-CUBE: High-Dimensional Hypervisor Fuzzing(2020)

当Fuzz遇上了2020

- 如何做出更通用的Fuzzer？
- QEMU很适合整体进行Fuzz
  - QEMU本质上就是一个用户态进程
  - 将QEMU整体改造成一个Fuzz对象
    - Fuzzer的输入/输出是什么？



VM EXIT

- 特定情况会导致VM EXIT
  - Intel手册包含68种情况（Intel® 64 and IA-32 Architectures Software Developer ’s Manual 3A, 3B, 3C, and 3D: System Programming Guide – 附录 C）
- 简单而言它们包括:
  - 部分特权指令的执行
  - 发生了某种处理器异常或中断
  - 开关机或严重错误（如triple-fault）

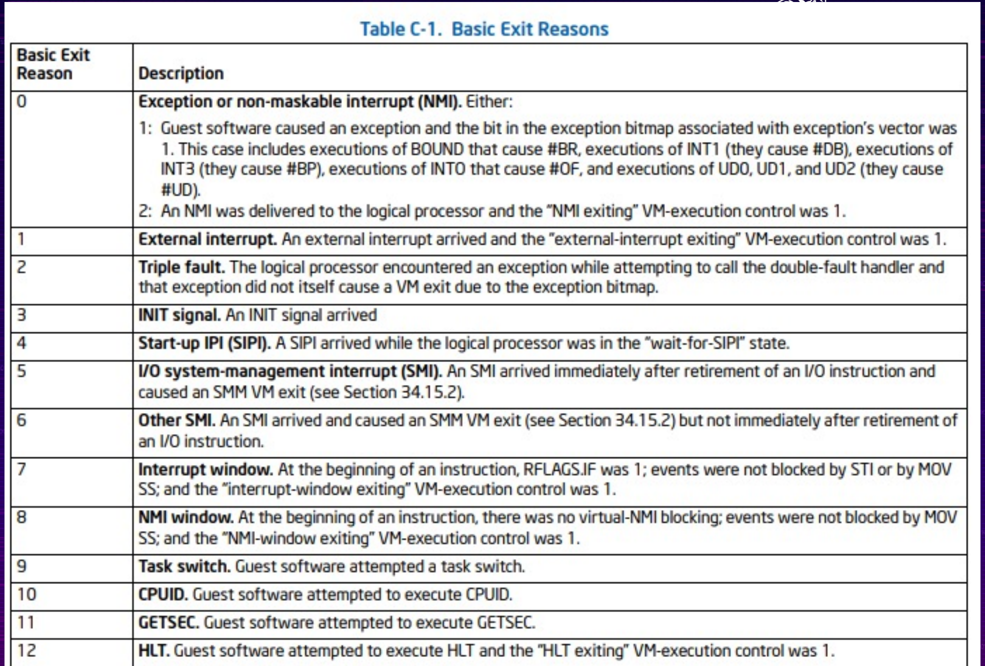
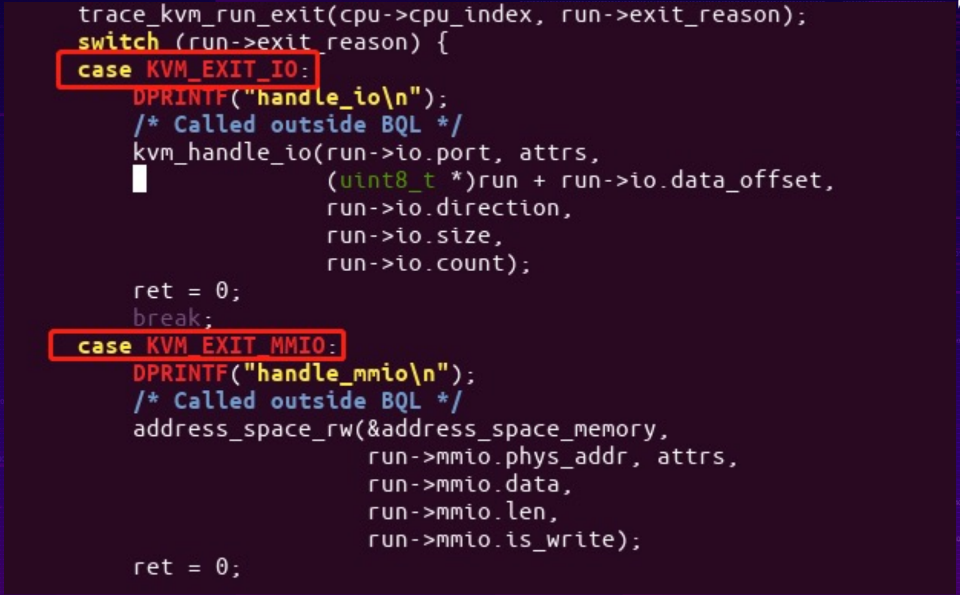


Table C-1. Basic Exit Reasons

Basic Exit Reason	Description
0	**Exception or non-maskable interrupt (NMI).** Either: 1: Guest software caused an exception and the bit in the exception bitmap associated with exception's vector was 1. This case includes executions of BOUND that cause #BR, executions of INT1 (they cause #DB), executions of INT3 (they cause #BP), executions of INTO that cause #OF, and executions of UD0, UD1, and UD2 (they cause #UD). 2: An NMI was delivered to the logical processor and the "NMI exiting" VM-execution control was 1.
1	**External interrupt.** An external interrupt arrived and the "external-interrupt exiting" VM-execution control was 1.
2	**Triple fault.** The logical processor encountered an exception while attempting to call the double-fault handler and that exception did not itself cause a VM exit due to the exception bitmap.
3	**INIT signal.** An INIT signal arrived
4	**Start-up IPI (SIPI).** A SIPI arrived while the logical processor was in the "wait-for-SIPI" state.
5	**I/O system-management interrupt (SMI).** An SMI arrived immediately after retirement of an I/O instruction and caused an SMM VM exit (see Section 34.15.2).
6	**Other SMI.** An SMI arrived and caused an SMM VM exit (see Section 34.15.2) but not immediately after retirement of an I/O instruction.
7	**Interrupt window.** At the beginning of an instruction, RFLAGS.IF was 1; events were not blocked by STI or by MOV SS; and the "interrupt-window exiting" VM-execution control was 1.
8	**NMI window.** At the beginning of an instruction, there was no virtual-NMI blocking; events were not blocked by MOV SS; and the "NMI-window exiting" VM-execution control was 1.
9	**Task switch.** Guest software attempted a task switch.
10	**CPUID.** Guest software attempted to execute CPUID.
11	**GETSEC.** Guest software attempted to execute GETSEC.
12	**HLT.** Guest software attempted to execute HLT and the "HLT exiting" VM-execution control was 1.

QEMU如何处理VMEXIT

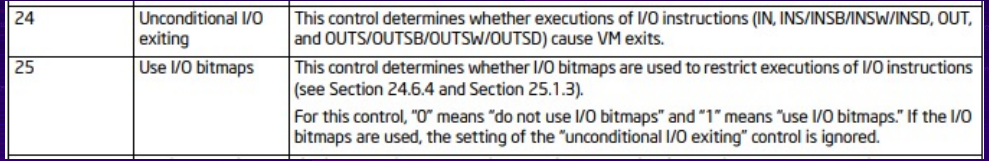
- accel/kvm/kvm-all.c
- 最关键的两个退出理由：
  - KVM_EXIT_IO
  - KVM_EXIT_MMIO
- 以及其余几个我们不太关心的理由



```
trace_kvm_run_exit(cpu->cpu_index, run->exit_reason);
switch (run->exit_reason) {
case KVM_EXIT_IO:
    DPRINTF("handle_io\n");
    /* Called outside BQL */
    kvm_handle_io(run->io.port, attrs,
                  (uint8_t *)run + run->io.data_offset,
                  run->io.direction,
                  run->io.size,
                  run->io.count);
    ret = 0;
    break;
case KVM_EXIT_MMIO:
    DPRINTF("handle_mmio\n");
    /* Called outside BQL */
    address_space_rw(&address_space_memory,
                     run->mmio.phys_addr, attrs,
                     run->mmio.data,
                     run->mmio.len,
                     run->mmio.is_write);
    ret = 0;
```

I/O Port和MMIO

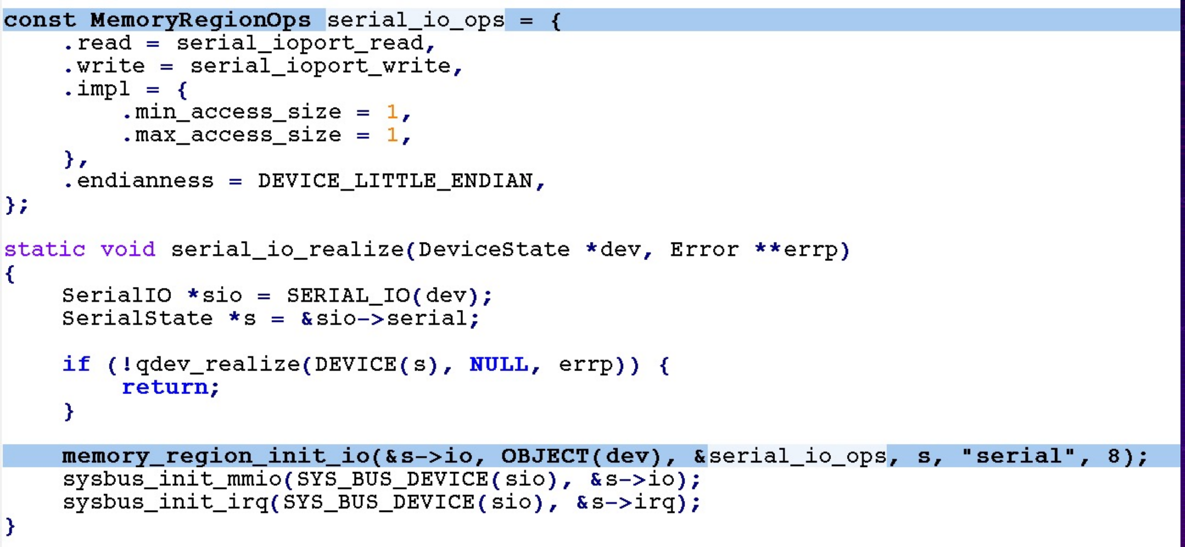
- CPU与外部设备、存储器的连接和数据交换都需要通过接口设备来实现
  - 不过QEMU的I/O 端口也是模拟的，因此有“24　无条件I/O退出”
- 实际就是Port Mapped I/O和Memory Mapped I/O
  - IN/OUT指令均会引起VM EXIT
  - MMIO常见的是mmap /dev/mem后进行读写操作
  - VMEXIT后，QEMU捕获退出原因并交由具体设备处理



24	Unconditional I/O exiting	This control determines whether executions of I/O instructions (IN, INS/INSB/INSW/INSD, OUT, and OUTS/OUTSB/OUTSW/OUTSD) cause VM exits.
25	Use I/O bitmaps	This control determines whether I/O bitmaps are used to restrict executions of I/O instructions (see Section 24.6.4 and Section 25.1.3). For this control, “0” means “do not use I/O bitmaps” and “1” means “use I/O bitmaps.” If the I/O bitmaps are used, the setting of the “unconditional I/O exiting” control is ignored.

设备中的I/O Port和MMIO

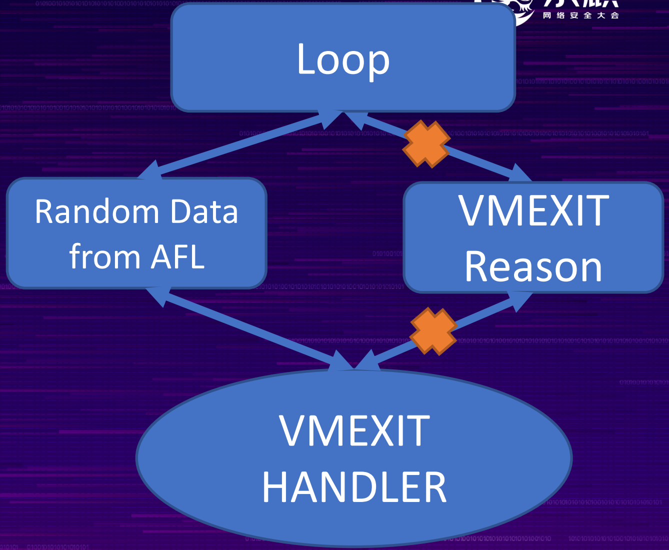
- 设备的realize函数（初始化用）会注册虚拟的端口
- 端口回调由.read/.write指定
- .min/max_access_size指定单次访问最小/大长度
- PORT和MMIO是guest内设备输入的
  最主要的、最单一的入口
- 历史上guest内触发的漏洞几乎都以这两个为入口



```
const MemoryRegionOps serial_io_ops = {
    .read = serial_ioport_read,
    .write = serial_ioport_write,
    .impl = {
        .min_access_size = 1,
        .max_access_size = 1,
    },
    .endianness = DEVICE_LITTLE_ENDIAN,
};

static void serial_io_realize(DeviceState *dev, Error **errp)
{
    SerialIO *sio = SERIAL_IO(dev);
    SerialState *s = &sio->serial;

    if (!qdev_realize(DEVICE(s), NULL, errp)) {
        return;
    }

    memory_region_init_io(&s->io, OBJECT(dev), &serial_io_ops, s, "serial", 8);
    sysbus_init_mmio(SYS_BUS_DEVICE(sio), &s->io);
    sysbus_init_irq(SYS_BUS_DEVICE(sio), &s->irq);
}
```

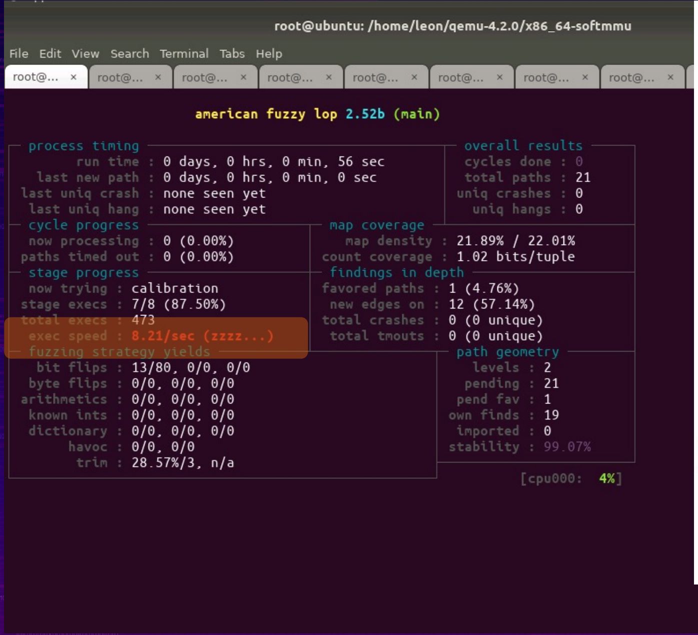
改造KVM的大处理循环

- 测试的目标是各个虚拟设备
- 不涉及vCPU，甚至基本都不涉及guest
- 改造vl.c，传入指定命令行，让QEMU初始化我们要Fuzz的设备
- 设备初始化完成后，就开始模拟VMEXIT，不断提交PORT/MMIO请求
- AFL提供输入，用于控制循环次数以及传给处理函数的参数

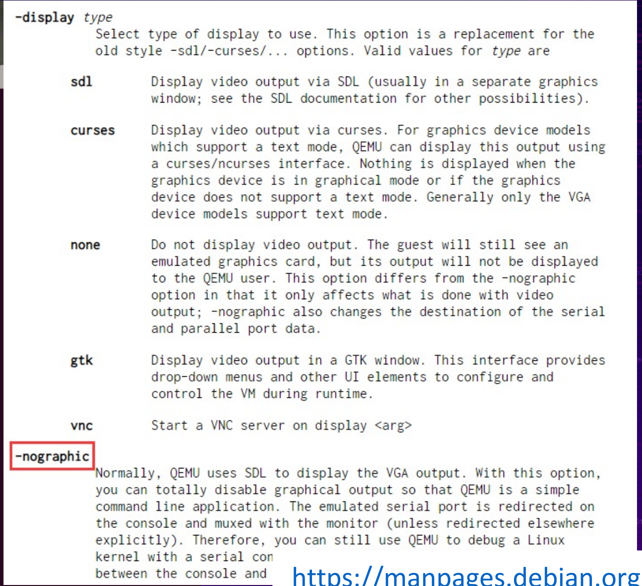


问题来了

- 命令行过于复杂，难以确认
- Fuzz速度很慢
- 效率过低
- 需要以无窗口模式启动


root@ubuntu: /home/leon/qemu-4.2.0/x86_64-softmmu
File Edit View Search Terminal Tabs Help
root@... × root@... × root@... × root@... × root@... × root@... × root@... × root@... ×

```
                    american fuzzy lop 2.52b (main)

┌─ process timing ─────────────────────────────────────┬─ overall results ─────┐
│        run time : 0 days, 0 hrs, 0 min, 56 sec        │  cycles done : 0       │
│   last new path : 0 days, 0 hrs, 0 min, 0 sec         │  total paths : 21      │
│ last uniq crash : none seen yet                       │ uniq crashes : 0       │
│  last uniq hang : none seen yet                       │   uniq hangs : 0       │
├─ cycle progress ────────────────────┬─ map coverage ─┴──────────────────────┤
│  now processing : 0 (0.00%)          │    map density : 21.89% / 22.01%      │
│ paths timed out : 0 (0.00%)          │ count coverage : 1.02 bits/tuple      │
├─ stage progress ────────────────────┼─ findings in depth ───────────────────┤
│  now trying : calibration            │ favored paths : 1 (4.76%)             │
│ stage execs : 7/8 (87.50%)           │  new edges on : 12 (57.14%)           │
│ total execs : 473                    │ total crashes : 0 (0 unique)          │
│  exec speed : 8.21/sec (zzzz...)     │  total tmouts : 0 (0 unique)          │
├─ fuzzing strategy yields ───────────┴───────────────┬─ path geometry ────────┤
│   bit flips : 13/80, 0/0, 0/0                        │    levels : 2          │
│  byte flips : 0/0, 0/0, 0/0                          │   pending : 21         │
│ arithmetics : 0/0, 0/0, 0/0                          │  pend fav : 1          │
│  known ints : 0/0, 0/0, 0/0                          │ own finds : 19         │
│  dictionary : 0/0, 0/0, 0/0                          │  imported : 0          │
│       havoc : 0/0, 0/0                               │ stability : 99.07%     │
│        trim : 28.57%/3, n/a                          ├────────────────────────┘
└──────────────────────────────────────────────────────┘          [cpu000:  4%]
```

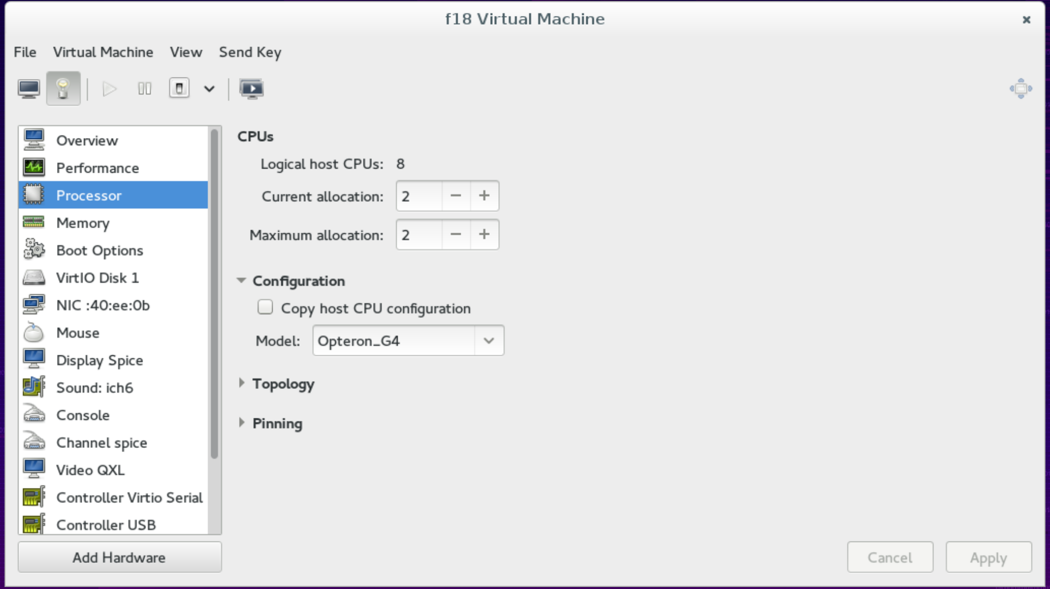


```
-display type
       Select type of display to use. This option is a replacement for the
       old style -sdl/-curses/... options. Valid values for type are

    sdl        Display video output via SDL (usually in a separate graphics
               window; see the SDL documentation for other possibilities).

    curses     Display video output via curses. For graphics device models
               which support a text mode, QEMU can display this output using
               a curses/ncurses interface. Nothing is displayed when the
               graphics device is in graphical mode or if the graphics
               device does not support a text mode. Generally only the VGA
               device models support text mode.

    none       Do not display video output. The guest will still see an
               emulated graphics card, but its output will not be displayed
               to the QEMU user. This option differs from the -nographic
               option in that it only affects what is done with video
               output; -nographic also changes the destination of the serial
               and parallel port data.

    gtk        Display video output in a GTK window. This interface provides
               drop-down menus and other UI elements to configure and
               control the VM during runtime.

    vnc        Start a VNC server on display <arg>

-nographic
       Normally, QEMU uses SDL to display the VGA output. With this option,
       you can totally disable graphical output so that QEMU is a simple
       command line application. The emulated serial port is redirected on
       the console and muxed with the monitor (unless redirected elsewhere
       explicitly). Therefore, you can still use QEMU to debug a Linux
       kernel with a serial con
       between the console and
```

https://manpages.debian.org/jessie/qemu-system-x86/qemu-system-x86_64.1.en.html

解决方式

- libvirt + virt-manager
  - 图形化配置后启动，获取需要的命令行
- 尽量配置足够多的设备，让测试程序有足够多的目标
- 将用到的东西尽量全部移动到RAMDISK上
- 限制访问有效的IOPORT或MMIO的端口/内存



f18 Virtual Machine

CPUs

Logical host CPUs: 8

Current allocation: 2

Maximum allocation: 2

Configuration

□ Copy host CPU configuration

Model: Opteron_G4

Topology

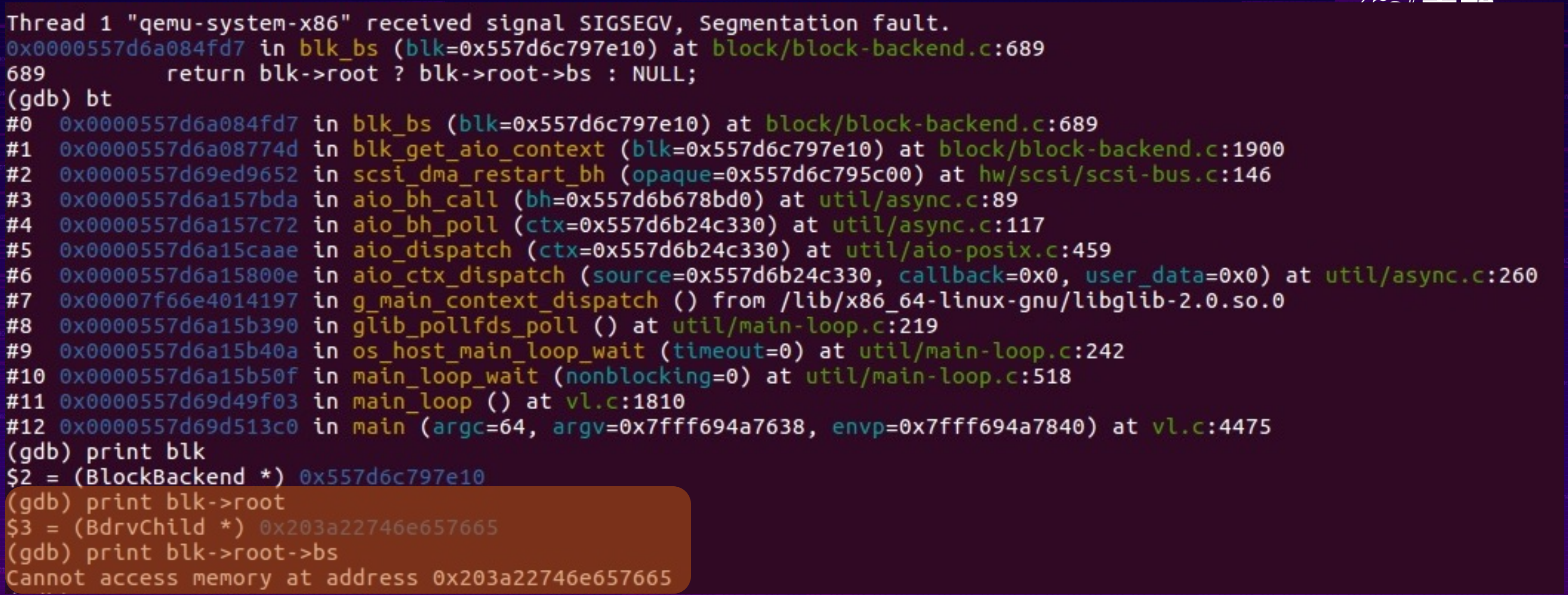
Pinning

测试结果

- 这是一个Fuzzer雏形
- 主要是为了初期配合人工审计，探索性质的Fuzzer
- 中途针对覆盖问题和速度问题进行了很多次修改
- 很快发现了一个漏洞

第1代Fuzzer的成果

- SCSI-BUS中的 Use-after-free
- 最坏情况下可能导致代码执行



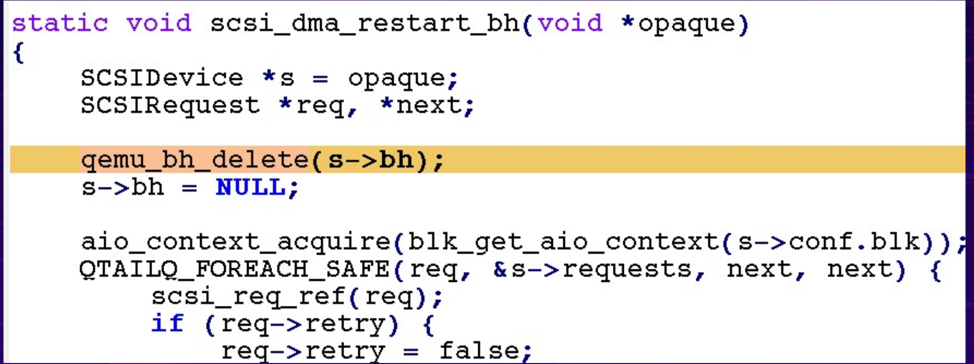
```
Thread 1 "qemu-system-x86" received signal SIGSEGV, Segmentation fault.
0x0000557d6a084fd7 in blk_bs (blk=0x557d6c797e10) at block/block-backend.c:689
689         return blk->root ? blk->root->bs : NULL;
(gdb) bt
#0  0x0000557d6a084fd7 in blk_bs (blk=0x557d6c797e10) at block/block-backend.c:689
#1  0x0000557d6a08774d in blk_get_aio_context (blk=0x557d6c797e10) at block/block-backend.c:1900
#2  0x0000557d69ed9652 in scsi_dma_restart_bh (opaque=0x557d6c795c00) at hw/scsi/scsi-bus.c:146
#3  0x0000557d6a157bda in aio_bh_call (bh=0x557d6b678bd0) at util/async.c:89
#4  0x0000557d6a157c72 in aio_bh_poll (ctx=0x557d6b24c330) at util/async.c:117
#5  0x0000557d6a15caae in aio_dispatch (ctx=0x557d6b24c330) at util/aio-posix.c:459
#6  0x0000557d6a15800e in aio_ctx_dispatch (source=0x557d6b24c330, callback=0x0, user_data=0x0) at util/async.c:260
#7  0x00007f66e4014197 in g_main_context_dispatch () from /lib/x86_64-linux-gnu/libglib-2.0.so.0
#8  0x0000557d6a15b390 in glib_pollfds_poll () at util/main-loop.c:219
#9  0x0000557d6a15b40a in os_host_main_loop_wait (timeout=0) at util/main-loop.c:242
#10 0x0000557d6a15b50f in main_loop_wait (nonblocking=0) at util/main-loop.c:518
#11 0x0000557d69d49f03 in main_loop () at vl.c:1810
#12 0x0000557d69d513c0 in main (argc=64, argv=0x7fff694a7638, envp=0x7fff694a7840) at vl.c:4475
(gdb) print blk
$2 = (BlockBackend *) 0x557d6c797e10
(gdb) print blk->root
$3 = (BdrvChild *) 0x203a22746e657665
(gdb) print blk->root->bs
Cannot access memory at address 0x203a22746e657665
```

漏洞成因

- [1] 虚拟机可以处于重启/暂停状态
- [2] 向虚拟机挂载一块新硬盘，此时scsi_dma_restart_cb回调注册
- [3] 虚拟机进入已启动/恢复状态，此时scsi_dma_restart_bh被计划在**主线程**中由glib_pollfds_poll调用运行，这个回调会访问设备对象
- [4] Guest内向PCI总线写入热插拔请求弹出硬盘，此时acpi_pcihp_eject_slot会在**另一个线程**删除该设备对象
- [3]/[4]在不同线程对同一个对象进行操作，因此会产生条件竞争。如果设备先被[4]删除，就会出现UaF。

RedHat的理由

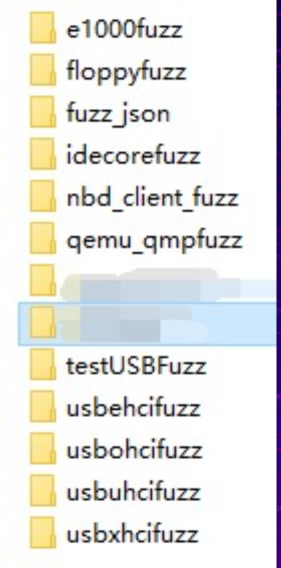
- 很可惜，这个问题，RedHat并没有给CVE
- 理由是：需要有管理人员去执行暂停虚拟机、挂载硬盘的操作
- 不过这里还有个问题就是，这两个操作的时隙不太长
- 如果利用成功，认为还是有危害的
- qemu_bh_delete是free的包装，而整个s这个对象都可以被占据
- https://bugzilla.redhat.com/show_bug.cgi?id=1854811



```
static void scsi_dma_restart_bh(void *opaque)
{
    SCSIDevice *s = opaque;
    SCSIRequest *req, *next;

    qemu_bh_delete(s->bh);
    s->bh = NULL;

    aio_context_acquire(blk_get_aio_context(s->conf.blk));
    QTAILQ_FOREACH_SAFE(req, &s->requests, next, next) {
        scsi_req_ref(req);
        if (req->retry) {
            req->retry = false;
```

“第二代” Fuzzer

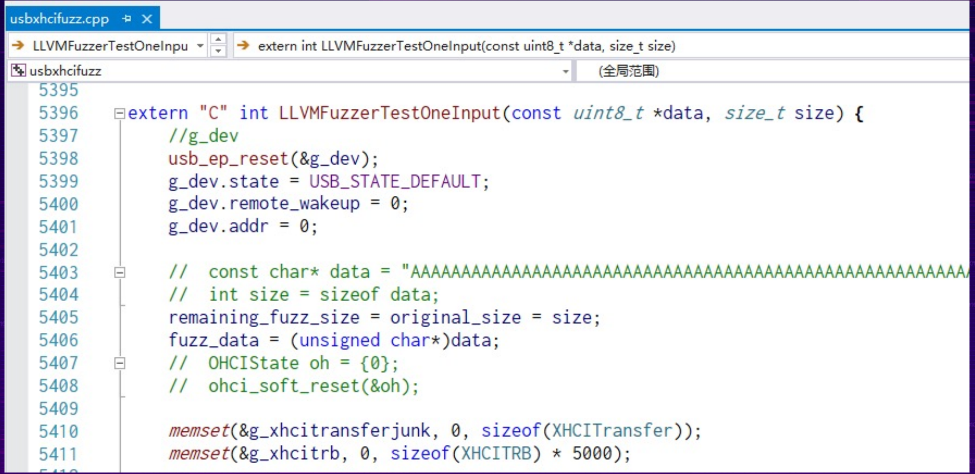
- 为了克服第一代Fuzzer的种种问题
- 从QEMU的单元测试中得到了一些灵感
- 因为虚拟设备的输入非常明确
  - PMIO/MMIO
- 不再将整个QEMU作为Fuzz目标，而是把单个设备作为目标



e1000fuzz
floppyfuzz
fuzz_json
idecorefuzz
nbd_client_fuzz
qemu_qmpfuzz
[redacted]
[redacted]
testUSBFuzz
usbehcifuzz
usbohcifuzz
usbuhcifuzz
usbxhcifuzz

基本思路

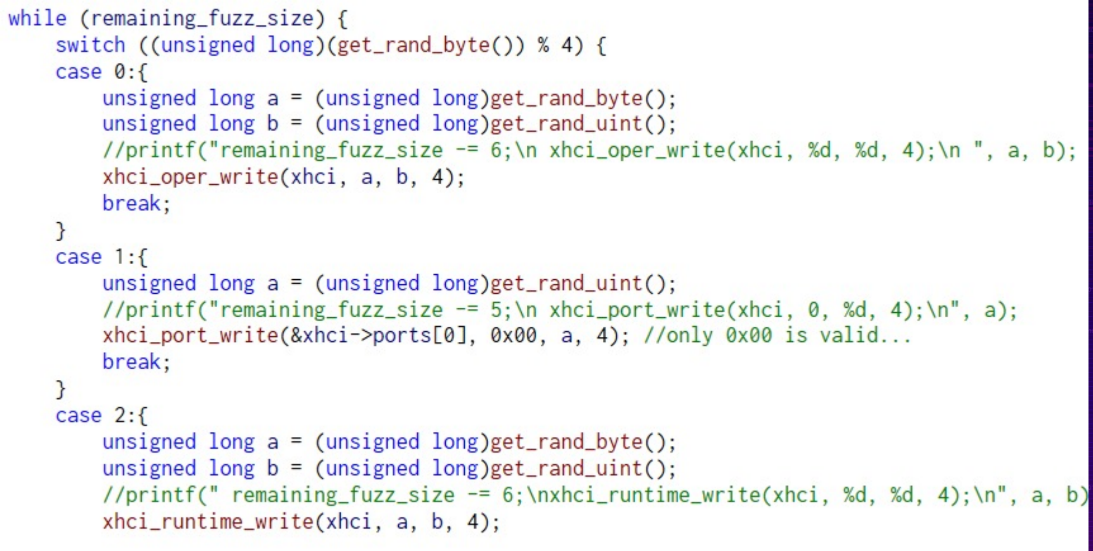
- 制作一个外壳，用于调用PMIO/MMIO接口
- 将虚拟设备的代码当成某种代码逻辑片段
- 虚拟设备不再发挥其原来的功能
  - 只是抽象的、没有特定目的的代码
- 去除一切I/O操作（比如模拟硬盘时，回写硬盘等操作）
- 精简一切不必要的操作（基于底层操作无安全问题的假设上开发）


usbxhcifuzz.cpp
LLVMFuzzerTestOneInpu
extern int LLVMFuzzerTestOneInput(const uint8_t *data, size_t size)
usbxhcifuzz
(全局范围)

```
extern "C" int LLVMFuzzerTestOneInput(const uint8_t *data, size_t size) {
    //g_dev
    usb_ep_reset(&g_dev);
    g_dev.state = USB_STATE_DEFAULT;
    g_dev.remote_wakeup = 0;
    g_dev.addr = 0;

    //  const char* data = "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
    //  int size = sizeof data;
    remaining_fuzz_size = original_size = size;
    fuzz_data = (unsigned char*)data;
    //  OHCIState oh = {0};
    //  ohci_soft_reset(&oh);

    memset(&g_xhcitransferjunk, 0, sizeof(XHCITransfer));
    memset(&g_xhcitrb, 0, sizeof(XHCITRB) * 5000);
```

基本思路2

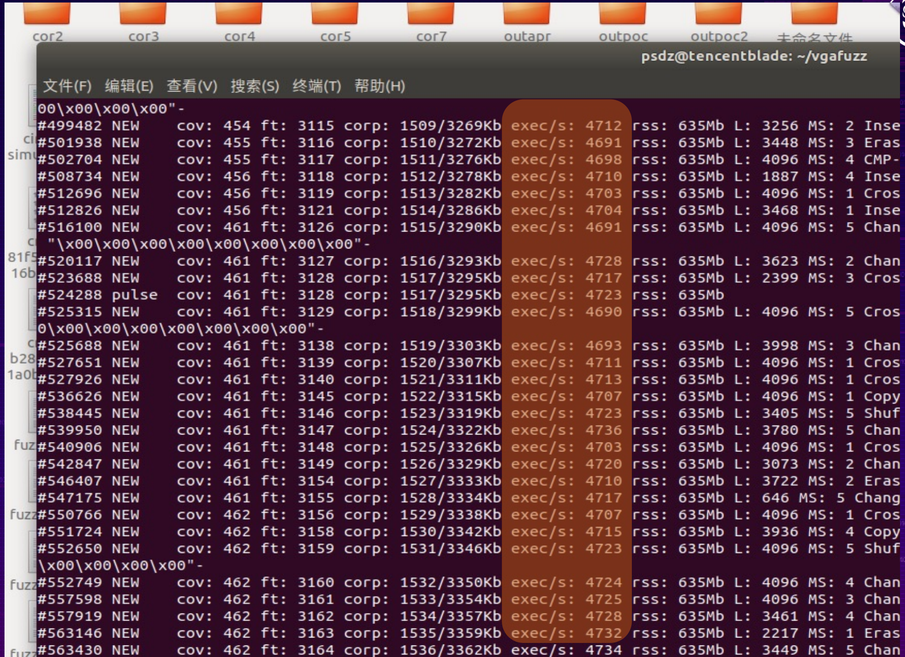
- 尽量按设备制作出各种小型的测试程序
- 不再有“总线” “设备管理器”等装置，取而代之使用全局变量和函数模拟（即假设系统只有这一个设备）
- 根据QEMU代码的要求，手动初始化设备数据
- 根据Fuzz数据调用PMIO/MMIO函数
- 一般搭配libFuzzer，以求达到最快的测试循环速度



```
while (remaining_fuzz_size) {
    switch ((unsigned long)(get_rand_byte()) % 4) {
    case 0:{
        unsigned long a = (unsigned long)get_rand_byte();
        unsigned long b = (unsigned long)get_rand_uint();
        //printf("remaining_fuzz_size -= 6;\n xhci_oper_write(xhci, %d, %d, 4);\n ", a, b);
        xhci_oper_write(xhci, a, b, 4);
        break;
    }
    case 1:{
        unsigned long a = (unsigned long)get_rand_uint();
        //printf("remaining_fuzz_size -= 5;\n xhci_port_write(xhci, 0, %d, 4);\n", a);
        xhci_port_write(&xhci->ports[0], 0x00, a, 4); //only 0x00 is valid...
        break;
    }
    case 2:{
        unsigned long a = (unsigned long)get_rand_byte();
        unsigned long b = (unsigned long)get_rand_uint();
        //printf(" remaining_fuzz_size -= 6;\nxhci_runtime_write(xhci, %d, %d, 4);\n", a, b)
        xhci_runtime_write(xhci, a, b, 4);
```

“第二代” Fuzzer 的优势

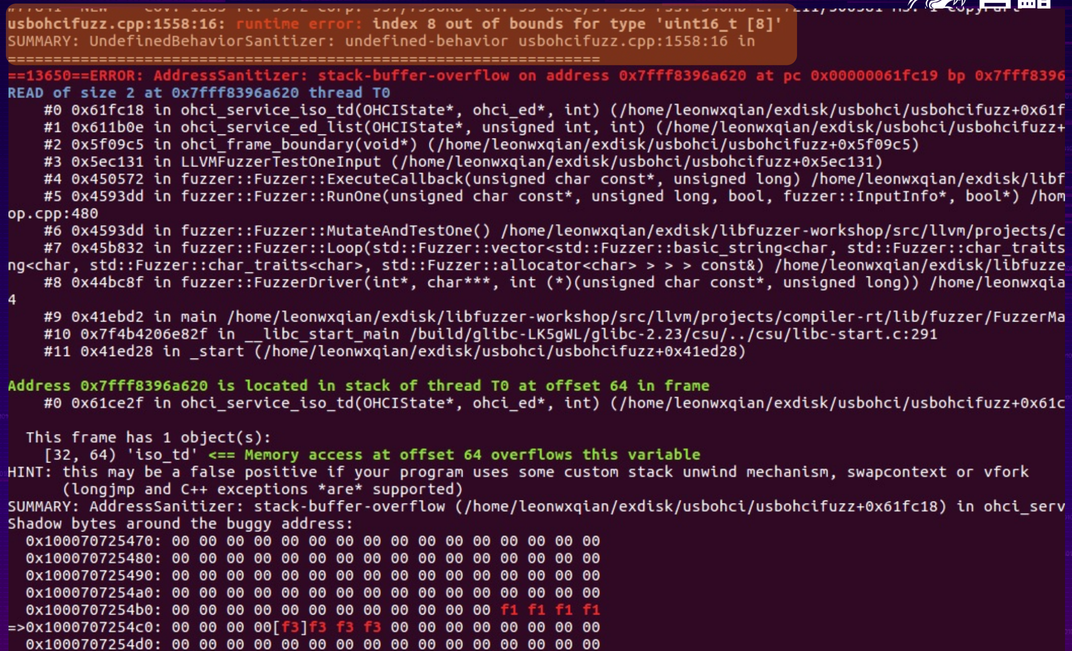
- 最早为了测试vga设备而构建
- 因为测试集小，没有I/O之类的操作，通常可以到每秒4k~13k的测试速度
- 容易确认覆盖不到的代码，代码维护容易，覆盖提升快
- 对某一个设备的Fuzz深度非常高


cor2
cor3
cor4
cor5
cor7
outapr
outpoc
outpoc2
未命名文件
psdz@tencentblade: ~/vgafuzz
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

```
00\x00\x00\x00"-
#499482 NEW    cov: 454 ft: 3115 corp: 1509/3269Kb exec/s: 4712 rss: 635Mb L: 3256 MS: 2 Inse
#501938 NEW    cov: 455 ft: 3116 corp: 1510/3272Kb exec/s: 4691 rss: 635Mb L: 3448 MS: 3 Eras
#502704 NEW    cov: 455 ft: 3117 corp: 1511/3276Kb exec/s: 4698 rss: 635Mb L: 4096 MS: 4 CMP-
#508734 NEW    cov: 456 ft: 3118 corp: 1512/3278Kb exec/s: 4710 rss: 635Mb L: 1887 MS: 4 Inse
#512696 NEW    cov: 456 ft: 3119 corp: 1513/3282Kb exec/s: 4703 rss: 635Mb L: 4096 MS: 1 Cros
#512826 NEW    cov: 456 ft: 3121 corp: 1514/3286Kb exec/s: 4704 rss: 635Mb L: 3468 MS: 1 Inse
#516100 NEW    cov: 461 ft: 3126 corp: 1515/3290Kb exec/s: 4691 rss: 635Mb L: 4096 MS: 5 Chan
 "\x00\x00\x00\x00\x00\x00\x00\x00"-
#520117 NEW    cov: 461 ft: 3127 corp: 1516/3293Kb exec/s: 4728 rss: 635Mb L: 3623 MS: 2 Chan
#523688 NEW    cov: 461 ft: 3128 corp: 1517/3295Kb exec/s: 4717 rss: 635Mb L: 2399 MS: 3 Cros
#524288 pulse  cov: 461 ft: 3128 corp: 1517/3295Kb exec/s: 4723 rss: 635Mb
#525315 NEW    cov: 461 ft: 3129 corp: 1518/3299Kb exec/s: 4690 rss: 635Mb L: 4096 MS: 5 Cros
0\x00\x00\x00\x00\x00\x00\x00"-
#525688 NEW    cov: 461 ft: 3138 corp: 1519/3303Kb exec/s: 4693 rss: 635Mb L: 3998 MS: 3 Chan
#527651 NEW    cov: 461 ft: 3139 corp: 1520/3307Kb exec/s: 4711 rss: 635Mb L: 4096 MS: 1 Cros
#527926 NEW    cov: 461 ft: 3140 corp: 1521/3311Kb exec/s: 4713 rss: 635Mb L: 4096 MS: 1 Cros
#536626 NEW    cov: 461 ft: 3145 corp: 1522/3315Kb exec/s: 4707 rss: 635Mb L: 4096 MS: 1 Copy
#538445 NEW    cov: 461 ft: 3146 corp: 1523/3319Kb exec/s: 4723 rss: 635Mb L: 3405 MS: 5 Shuf
#539950 NEW    cov: 461 ft: 3147 corp: 1524/3322Kb exec/s: 4736 rss: 635Mb L: 3780 MS: 5 Chan
#540906 NEW    cov: 461 ft: 3148 corp: 1525/3326Kb exec/s: 4703 rss: 635Mb L: 4096 MS: 1 Cros
#542847 NEW    cov: 461 ft: 3149 corp: 1526/3329Kb exec/s: 4720 rss: 635Mb L: 3073 MS: 2 Chan
#546407 NEW    cov: 461 ft: 3154 corp: 1527/3333Kb exec/s: 4710 rss: 635Mb L: 3722 MS: 2 Eras
#547175 NEW    cov: 461 ft: 3155 corp: 1528/3334Kb exec/s: 4717 rss: 635Mb L: 646 MS: 5 Chang
#550766 NEW    cov: 462 ft: 3156 corp: 1529/3338Kb exec/s: 4707 rss: 635Mb L: 4096 MS: 1 Cros
#551724 NEW    cov: 462 ft: 3158 corp: 1530/3342Kb exec/s: 4715 rss: 635Mb L: 3936 MS: 4 Copy
#552650 NEW    cov: 462 ft: 3159 corp: 1531/3346Kb exec/s: 4723 rss: 635Mb L: 4096 MS: 5 Shuf
\x00\x00\x00\x00"-
#552749 NEW    cov: 462 ft: 3160 corp: 1532/3350Kb exec/s: 4724 rss: 635Mb L: 4096 MS: 4 Chan
#557598 NEW    cov: 462 ft: 3161 corp: 1533/3354Kb exec/s: 4725 rss: 635Mb L: 4096 MS: 3 Chan
#557919 NEW    cov: 462 ft: 3162 corp: 1534/3357Kb exec/s: 4728 rss: 635Mb L: 3461 MS: 4 Chan
#563146 NEW    cov: 462 ft: 3163 corp: 1535/3359Kb exec/s: 4732 rss: 635Mb L: 2217 MS: 1 Eras
#563430 NEW    cov: 462 ft: 3164 corp: 1536/3362Kb exec/s: 4734 rss: 635Mb L: 3449 MS: 5 Chan
```

Fuzz的结果

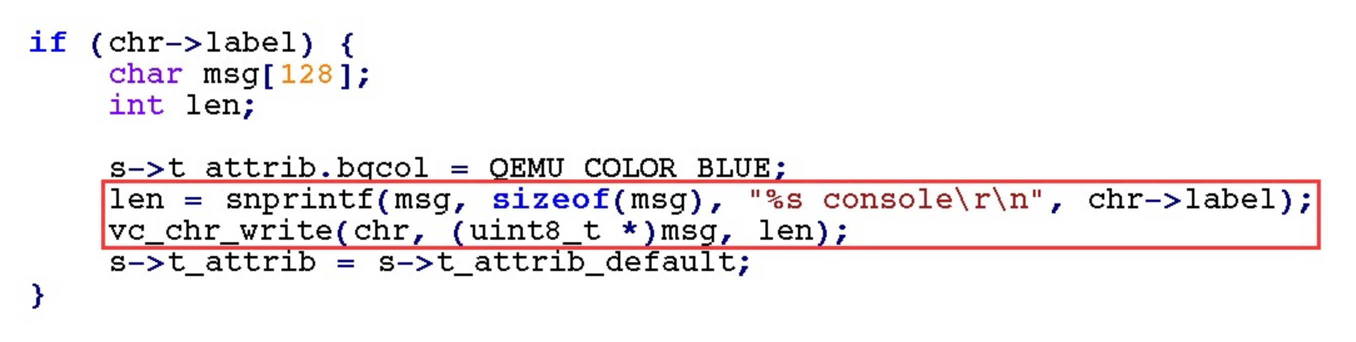
- 发现了8处问题
- 但QEMU官方(RedHat)自己也在Fuzz
- 很多不幸被撞



```
usbohcifuzz.cpp:1558:16: runtime error: index 8 out of bounds for type 'uint16_t [8]'
SUMMARY: UndefinedBehaviorSanitizer: undefined-behavior usbohcifuzz.cpp:1558:16 in
=================================================================
==13650==ERROR: AddressSanitizer: stack-buffer-overflow on address 0x7fff8396a620 at pc 0x00000061fc19 bp 0x7fff8396
READ of size 2 at 0x7fff8396a620 thread T0
    #0 0x61fc18 in ohci_service_iso_td(OHCIState*, ohci_ed*, int) (/home/leonwxqian/exdisk/usbohci/usbohcifuzz+0x61f
    #1 0x611b0e in ohci_service_ed_list(OHCIState*, unsigned int, int) (/home/leonwxqian/exdisk/usbohci/usbohcifuzz+
    #2 0x5f09c5 in ohci_frame_boundary(void*) (/home/leonwxqian/exdisk/usbohci/usbohcifuzz+0x5f09c5)
    #3 0x5ec131 in LLVMFuzzerTestOneInput (/home/leonwxqian/exdisk/usbohci/usbohcifuzz+0x5ec131)
    #4 0x450572 in fuzzer::Fuzzer::ExecuteCallback(unsigned char const*, unsigned long) /home/leonwxqian/exdisk/libf
    #5 0x4593dd in fuzzer::Fuzzer::RunOne(unsigned char const*, unsigned long, bool, fuzzer::InputInfo*, bool*) /hom
op.cpp:480
    #6 0x4593dd in fuzzer::Fuzzer::MutateAndTestOne() /home/leonwxqian/exdisk/libfuzzer-workshop/src/llvm/projects/c
    #7 0x45b832 in fuzzer::Fuzzer::Loop(std::Fuzzer::vector<std::Fuzzer::basic_string<char, std::Fuzzer::char_traits
ng<char, std::Fuzzer::char_traits<char>, std::Fuzzer::allocator<char> > > > const&) /home/leonwxqian/exdisk/libfuzze
    #8 0x44bc8f in fuzzer::FuzzerDriver(int*, char***, int (*)(unsigned char const*, unsigned long)) /home/leonwxqia
4
    #9 0x41ebd2 in main /home/leonwxqian/exdisk/libfuzzer-workshop/src/llvm/projects/compiler-rt/lib/fuzzer/FuzzerMa
    #10 0x7f4b4206e82f in __libc_start_main /build/glibc-LK5gWL/glibc-2.23/csu/../csu/libc-start.c:291
    #11 0x41ed28 in _start (/home/leonwxqian/exdisk/usbohci/usbohcifuzz+0x41ed28)

Address 0x7fff8396a620 is located in stack of thread T0 at offset 64 in frame
    #0 0x61ce2f in ohci_service_iso_td(OHCIState*, ohci_ed*, int) (/home/leonwxqian/exdisk/usbohci/usbohcifuzz+0x61c

  This frame has 1 object(s):
    [32, 64) 'iso_td' <== Memory access at offset 64 overflows this variable
HINT: this may be a false positive if your program uses some custom stack unwind mechanism, swapcontext or vfork
      (longjmp and C++ exceptions *are* supported)
SUMMARY: AddressSanitizer: stack-buffer-overflow (/home/leonwxqian/exdisk/usbohci/usbohcifuzz+0x61fc18) in ohci_serv
Shadow bytes around the buggy address:
  0x100070725470: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x100070725480: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x100070725490: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x1000707254a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x1000707254b0: 00 00 00 00 00 00 00 00 00 00 00 00 f1 f1 f1 f1
=>0x1000707254c0: 00 00 00 00[f3]f3 f3 f3 00 00 00 00 00 00 00 00
  0x1000707254d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

ui/console.c 的信息泄露问题

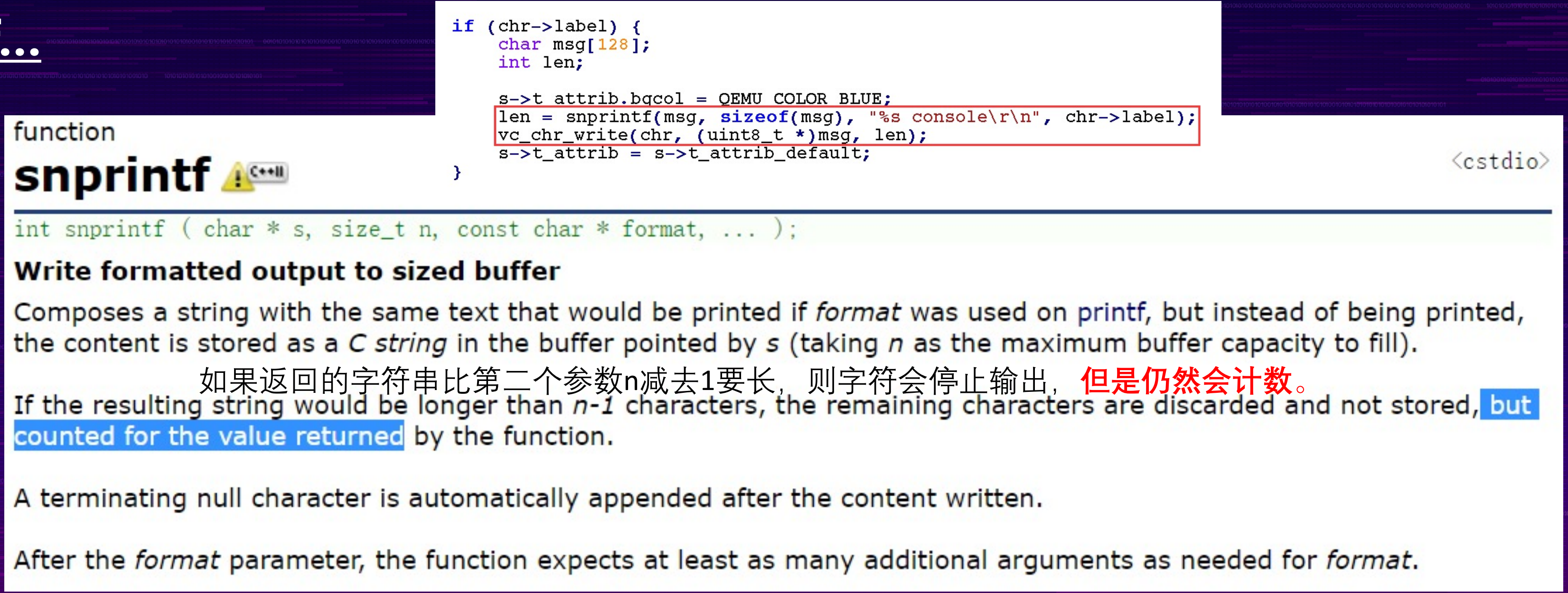
- -chardev vc,id=[ID HERE]
- 通过上述命令行可以创建一个字符设备
- -mon chardev=[SAME ID HERE]
- 通过上述命令行可以引用该设备



```
if (chr->label) {
    char msg[128];
    int len;

    s->t_attrib.bgcol = QEMU_COLOR_BLUE;
    len = snprintf(msg, sizeof(msg), "%s console\r\n", chr->label);
    vc_chr_write(chr, (uint8_t *)msg, len);
    s->t_attrib = s->t_attrib_default;
}
```

- 设备名会被snprintf输出到msg[128](max = 128)中
- 然后vc_chr_write()会再次从msg中输出snprintf的返回值个字符到console中

snprintf...



```
if (chr->label) {
    char msg[128];
    int len;

    s->t_attrib.bgcol = QEMU_COLOR_BLUE;
    len = snprintf(msg, sizeof(msg), "%s console\r\n", chr->label);
    vc_chr_write(chr, (uint8_t *)msg, len);
    s->t_attrib = s->t_attrib_default;
}
```

function

snprintf

<cstdio>

```
int snprintf ( char * s, size_t n, const char * format, ... );
```

Write formatted output to sized buffer

Composes a string with the same text that would be printed if *format* was used on printf, but instead of being printed, the content is stored as a *C string* in the buffer pointed by *s* (taking *n* as the maximum buffer capacity to fill).

如果返回的字符串比第二个参数n减去1要长，则字符会停止输出，**但是仍然会计数。**

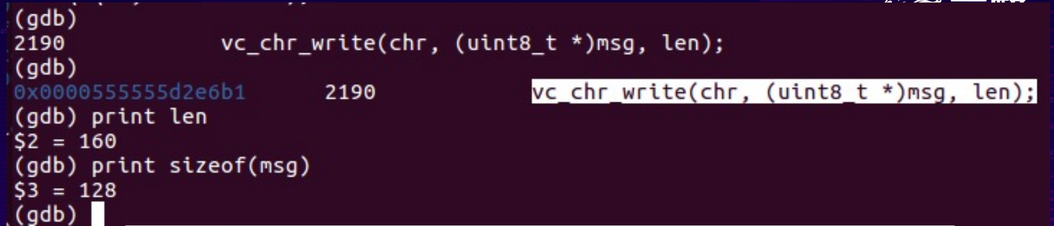
If the resulting string would be longer than *n-1* characters, the remaining characters are discarded and not stored, but counted for the value returned by the function.

A terminating null character is automatically appended after the content written.

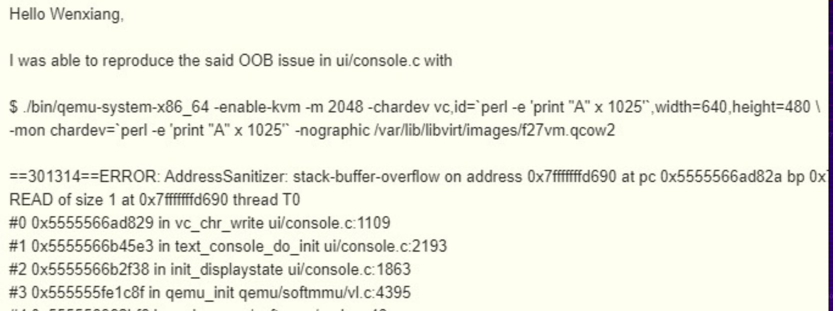
After the *format* parameter, the function expects at least as many additional arguments as needed for *format*.

ui/console.c 的信息泄露问题

- 几乎同时被我们的Fuzzer和静态扫描发现
- 指定过长的ID即可泄露数据
- 可以泄露栈上保存的函数指针，用于后续漏洞利用
- 但RedHat坚持只有有控制权限的人才能设置ID，因此未发CVE
- https://patchew.org/QEMU/20200701181801.27935-1-kraxel@redhat.com/



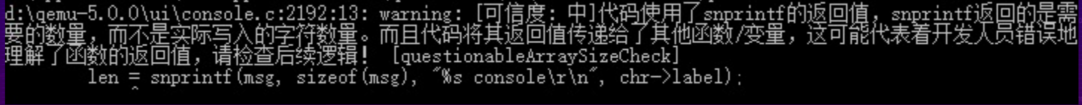
```
(gdb)
2190                vc_chr_write(chr, (uint8_t *)msg, len);
(gdb)
0x0000555555d2e6b1      2190                vc_chr_write(chr, (uint8_t *)msg, len);
(gdb) print len
$2 = 160
(gdb) print sizeof(msg)
$3 = 128
(gdb) █
```



Hello Wenxiang,

I was able to reproduce the said OOB issue in ui/console.c with

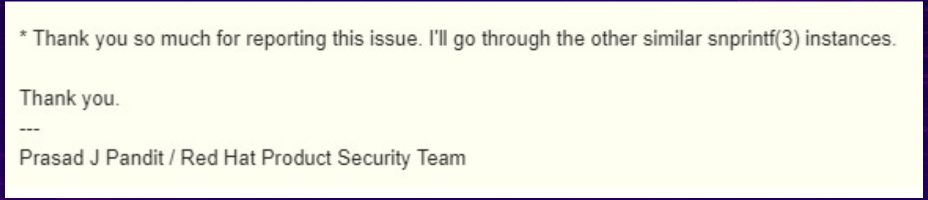
```
$ ./bin/qemu-system-x86_64 -enable-kvm -m 2048 -chardev vc,id=`perl -e 'print "A" x 1025'`,width=640,height=480 \
-mon chardev=`perl -e 'print "A" x 1025'` -nographic /var/lib/libvirt/images/f27vm.qcow2

==301314==ERROR: AddressSanitizer: stack-buffer-overflow on address 0x7fffffffd690 at pc 0x5555566ad82a bp 0x
READ of size 1 at 0x7fffffffd690 thread T0
#0 0x5555566ad829 in vc_chr_write ui/console.c:1109
#1 0x5555566b45e3 in text_console_do_init ui/console.c:2193
#2 0x5555566b2f38 in init_displaystate ui/console.c:1863
#3 0x555555fe1c8f in qemu_init qemu/softmmu/vl.c:4395
```



```
d:\qemu-5.0.0\ui\console.c:2192:13: warning: [可信度: 中]代码使用了snprintf的返回值, snprintf返回的是需要的数量, 而不是实际写入的字符数量。而且代码将其返回值传递给了其他函数/变量, 这可能代表着开发人员错误地理解了函数的返回值, 请检查后续逻辑!  [questionableArraySizeCheck]
        len = snprintf(msg, sizeof(msg), "%s console\r\n", chr->label);
            ^
```

发现的其他问题

- 某模块越界读写已报告，尚在披露保密期
- 某模组的提权问题已报告，尚在披露保密期
- Snprintf等类似的问题通过静态扫描也发现了几处



* Thank you so much for reporting this issue. I'll go through the other similar snprintf(3) instances.

Thank you.

Prasad J Pandit / Red Hat Product Security Team

安全研究和企业开发的一些安全思考

QEMU中常见的、容易发现的安全问题

- 早期：非常显眼的错误代码，在接入静态扫描后基本消失
  - 仍然可以扫描到，只是会在一些非常冷门、接近停止维护的设备代码中
- 过渡期：红帽意图将结构体替换为无符号类型，但现在QEMU中仍存在大量的有符号、无符号混用
  - 整数溢出，以及因此导致的各种问题
  - QEMU一部分代码编写时间很早，当时考虑的多为32位的情况，在64位环境下可能出问题
- 基础的函数问题，如之前所述的snprintf问题，以及如9pfs等文件系统中的逻辑问题

企业开发的一些常用安全实践总结

- 将静态扫描的流程集中在开发环节每一个提交中
- 设置专门人员负责安全审计
- 测试人员或开发人员应当对复杂功能编写Fuzzer并及时更新
- 提供足够算力的机器对代码进行模糊测试
- 及时处理崩溃或告警信息，合并PATCH并回报官方
- 研制热补丁系统，以方便修补类似于CVE-2020-14364这样补丁不需要改动很多代码的问题

BLADE
Tencent Blade

腾讯宙斯盾
DDoS防护系统

門神

腾讯蓝军
-Tencent Force

TSRC
腾讯安全应急响应中心

洞犀
INSIGHT SCANNER

Onion EDR
洋葱反入侵系统

腾讯铁将军

K
金刚系统
KING KONG

京麒
网络安全大会

THANKS