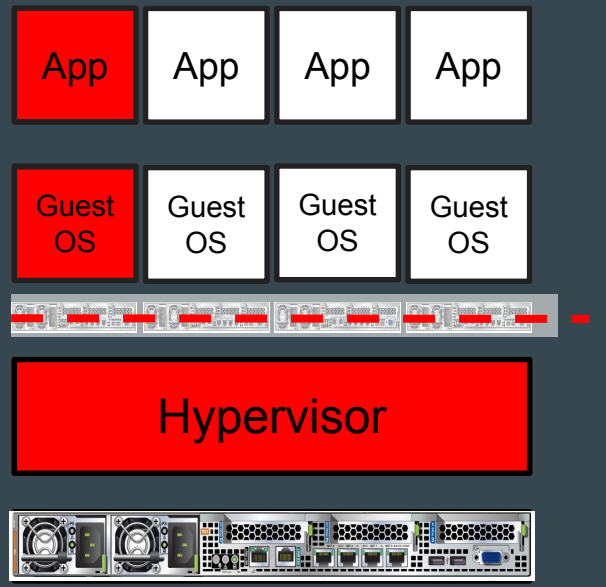
Virtual Device Fuzzing in QEMU

Alexander Bulekov, Boston University
Bandan Das, Red Hat

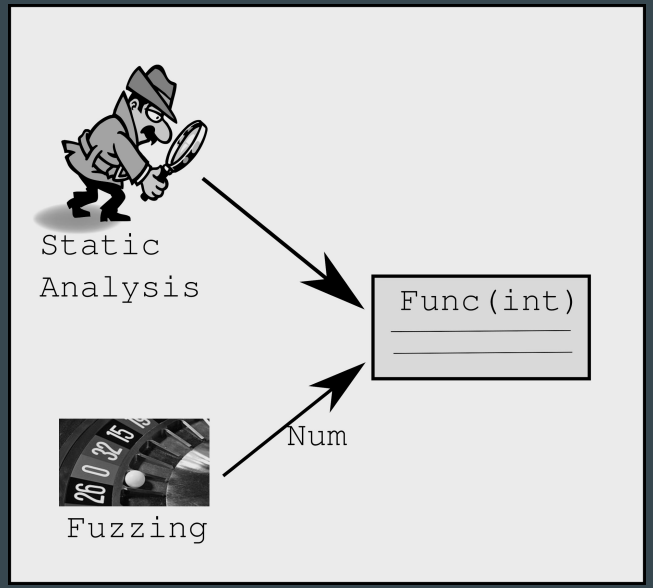
QEMU and Virtual Devices

- Virtual Devices enable guest I/O
- LOC: ~500k compared to TLOC: 1.7M
- VIRTIO, emulated Device Fuzzing in QEMU
- hardware devices
- A potential attack surface
    - Hardening - an ongoing challenge



Code Analysis

- Vulnerabilities are the foundations for attacks!
- Static Analysis
  - Check for Syntax, Semantics
  - Offline run
  - False positives
- Dynamic Analysis
  - Fuzzing
  - Feeding “random” data at runtime
  - Integration challenges
  - False positives ?
- Complementary



Fuzzing in QEMU - an outline

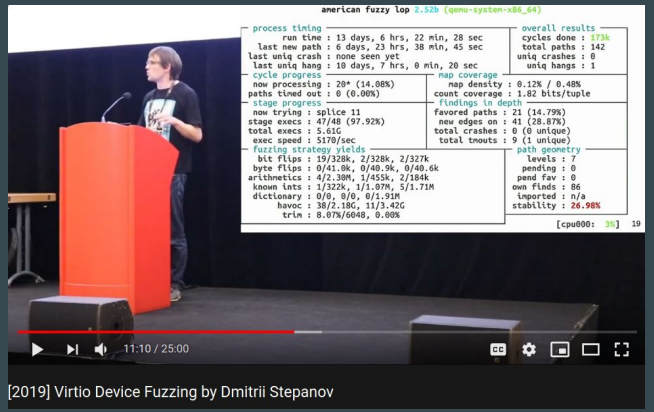
- qcow2 fuzzer
  - Maria Kustova, 2014
- Megasas MMIO write segfault
  - AFL, Salva Peiró, 2015
- Virtio Device Fuzzing using AFL
  - Dmitrii Stepanov, KVM Forum, 2019
- Google Summer of Code 2019

[Qemu-devel] [PATCH 0/5] tests: Add the image fuzzer with qcow2 support

From: Maria Kustova
Subject: [Qemu-devel] [PATCH 0/5] tests: Add the image fuzzer with qcow2 support
Date: Mon, 30 Jun 2014 15:48:35 +0400

```
This patch series introduces the image fuzzer, a tool for stability and
reliability testing.
Its approach is to run large amount of tests in background. During every test a
program (e.g. qemu-img) is called to read or modify an invalid test image.
A test image has valid inner structure defined by its format specification with
some fields having random invalid values.

Patch 1 contains documentation for the image fuzzer, patch 2 is the test runner
and remaining ones relate to the image generator for qcow2 format.
```



Google Summer of Code

Missing pieces

- QEMU integration
  - Making it easy for developers to fuzz
- Continuous Integration
  - Catch bugs in new code, prior to the next release
- Hardware
  - For fuzzing runs

QEMU fuzzing - Challenges

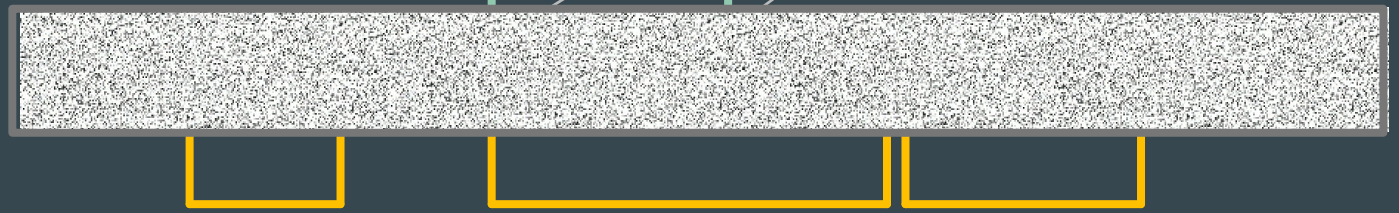
- Large input space
  - Devices IO happens in across multiple channels
- Fuzzing framework
  - Need something familiar, but reasonably performant
- State changes
  - Devices accumulate state
  - If we want to reliably reproduce bugs, the fuzzer needs to start with a clean slate for each input

Fuzzing framework

- American Fuzzy Lop
- libfuzzer
  - LLVM based and integrated into Google’s oss-fuzz infrastructure
- Prebuilt fuzzers are well suited for fuzzing single or few file inputs
- A custom fuzzer for QEMU ?
  - Better integration
  - Parallel problem: Kernel fuzzing and syzkaller

virtual_device(?)

Port IO



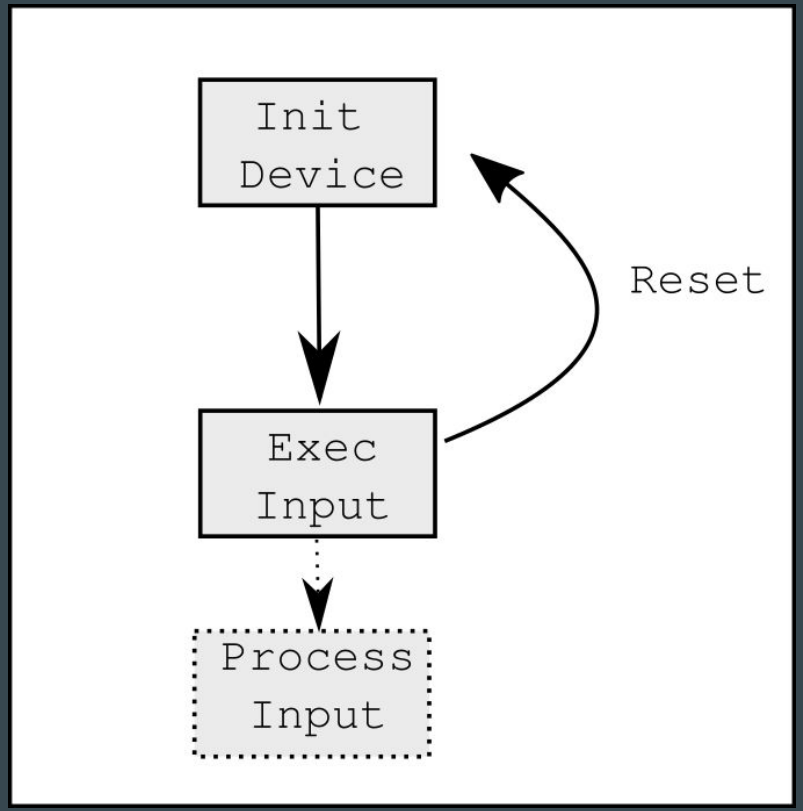
MMIO

DMA

The Input Space is Enormous!

State rewinds

- To get reliable results, fuzzing input data needs a consistent state
  - Reboot ?
  - Snapshots ?
  - Fork ?



Recap: Testing Devices in QEMU

- qtest
  - QEMU system process listens to commands from a qtest client, such as inw, outl, readq, clock_step
  - QTest test-cases usually use libqtest to configure, initialize and send qtest commands to the QEMU system process.
- libqos
  - Testing complex devices is difficult with the qtest IO primitives. Manually perform PCI enumeration, device initialization, allocation of space for DMA data...
  - libqos builds upon libqtest to implement standardized driver-like interfaces for common needs, such as bus-access, RAM allocation, etc.

```
outl 0xcf8 0x80001018
outl 0xcfc 0xe0800000
outl 0xcf8 0x80001020
outl 0xcf8 0x80001004
outw 0xcfc 0x7
writeq 0xe0801024 0x10646c00776c6cff
writeq 0xe080102d 0xe0801000320000
writeq 0xe0801015 0x12b2901ba000000
write 0x10646c02 0x1 0x2c
write 0x999 0x1 0x25
...
```

```
QE1000E_PCI *d = (QE1000E_PCI *) obj;
 uint32_t val;

 /* Enable the device */
 qpci_device_enable(&d->pci_dev);

 /* Reset the device */
 val = e1000e_macreg_read(&d->e1000e,
                          E1000E_CTRL);
 ...
```

Fuzzing a Device ≈ Writing a new QTest test

```
/* Uses simple qtest commands and reboots to reset state */
fuzz_add_target(&(FuzzTarget){
        .name = "i440fx-qtest-reboot-fuzz",
        .description = "Fuzz the i440fx using raw qtest commands and "
        "rebooting after each run",
        .get_init_cmdline = i440fx_argv,
        .fuzz = i440fx_fuzz_qtest});

/* Uses libqos and forks to prevent state leakage */
fuzz_add_qos_target(&(FuzzTarget){
        .name = "i440fx-qos-fork-fuzz",
        .description = "Fuzz the i440fx using raw qtest commands and "
        "rebooting after each run",
        .pre_vm_init = &fork_init,
        .fuzz = i440fx_fuzz_qos_fork,},
        "i440FX-pcihost",
        &(QOSGraphTestOptions){}
        );
```

```
static void ioport_fuzz_qtest(QTestState *s,
        const unsigned char *Data, size_t Size) {
    /*
     * loop over the Data, breaking it up into actions. each action has an
     * opcode, address offset and value
     */
    struct {
        uint8_t opcode;
        uint8_t addr;
        uint32_t value;
    } a;

    while (Size >= sizeof(a)) {
        memcpy(&a, Data, sizeof(a));
        uint16_t addr = a.addr % 2 ? I440FX_PCI_HOST_BRIDGE_CFG :
                                     I440FX_PCI_HOST_BRIDGE_DATA;
        switch (a.opcode % ACTION_MAX) {
        case WRITEB:
            qtest_outb(s, addr, (uint8_t)a.value);
            break;
        case WRITEW:
            qtest_outw(s, addr, (uint16_t)a.value);
            break;
...
```

```
static void pciconfig_fuzz_qos(QTestState *s, QPCIBus *bus,
        const unsigned char *Data, size_t Size) {
    /*
     * Same as ioport_fuzz_qtest, but using QOS. devfn is incorporated into the
     * value written over Port IO
     */
    struct {
        uint8_t opcode;
        uint8_t offset;
        int devfn;
        uint32_t value;
    } a;

    while (Size >= sizeof(a)) {
        memcpy(&a, Data,  sizeof(a));
        switch (a.opcode % ACTION_MAX) {
        case WRITEB:
            bus->config_writeb(bus, a.devfn, a.offset, ( uint8_t)a.value);
            break;
        case WRITEW:
            bus->config_writew(bus, a.devfn, a.offset, ( uint16_t)a.value);
```

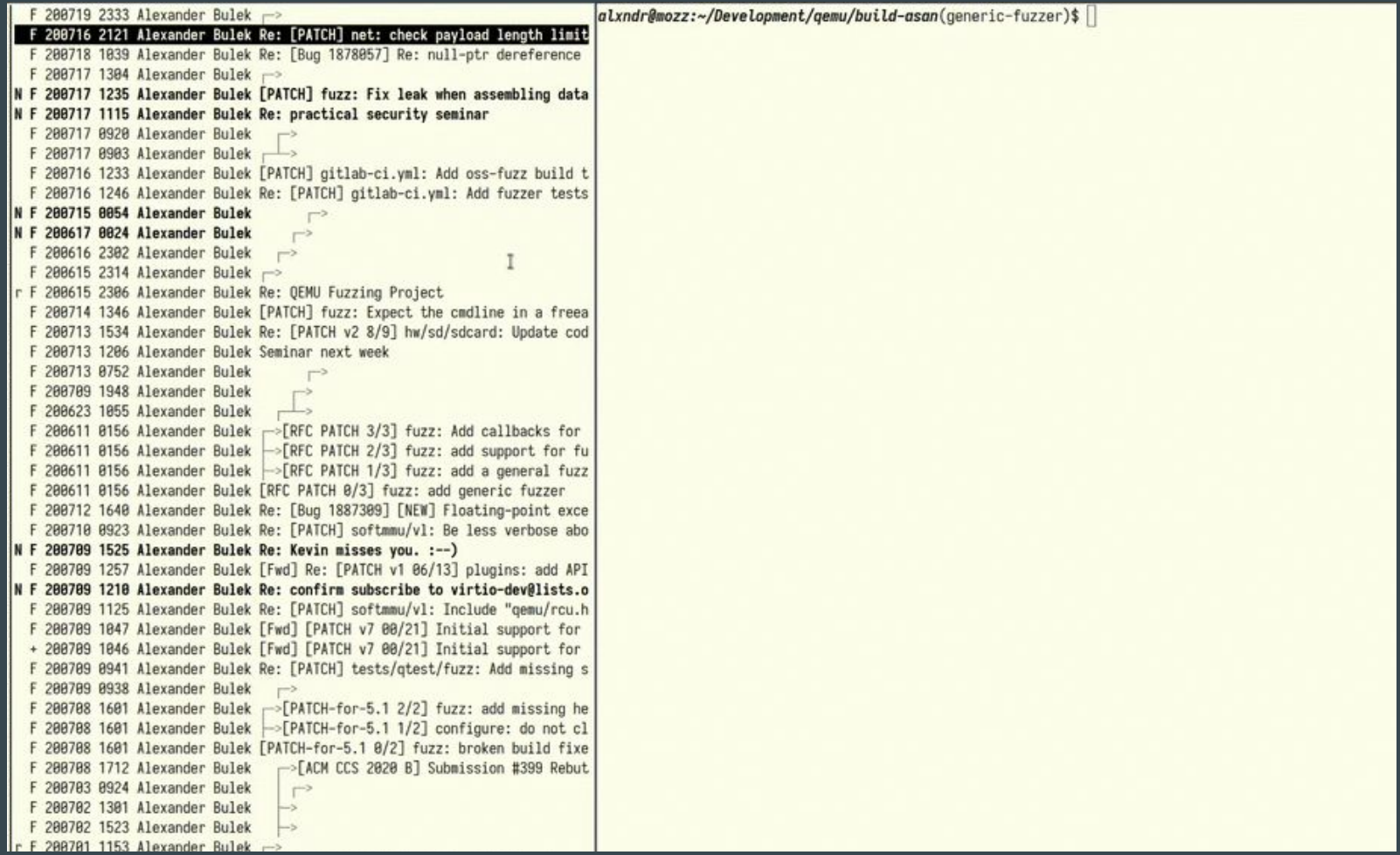
Generic Device Fuzzer

- Sometimes writing a fuzzer tailored for a device is tough
- We built a General Device Fuzzer that will fuzz devices over MMIO, Port IO and DMA
- To use, simply specify the arguments and object/MemoryRegion names you want to fuzz
  - `QEMU_FUZZ_ARGS="-M q35 -nodefaults -device e1000,netdev=net0 -netdev user,id=net0"`
  - `QEMU_FUZZ_OBJECTS='e100*'`

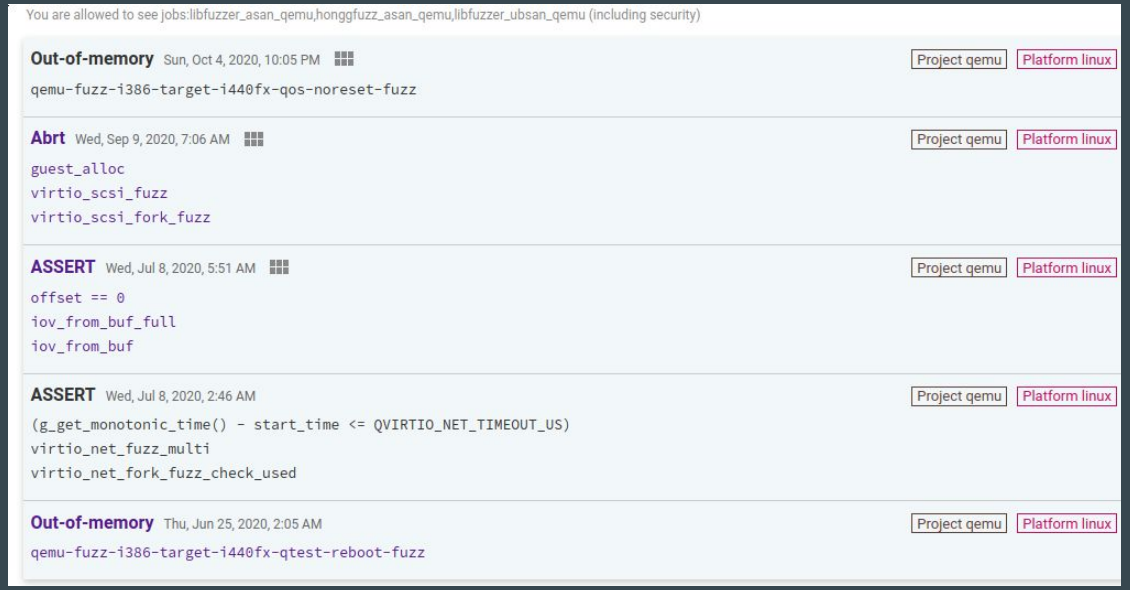
Generic Device Fuzzer

```
outl 0xcf8 0x80001010
outl 0xcfc 0xe1020000
outl 0xcf8 0x80001014
outl 0xcf8 0x80001004
outw 0xcfc 0x7
outl 0xcf8 0x800010a2
writeb 0xe102003b 0xff
writel 0xe1020118 0xffffffff
writel 0xe1020420 0xffffffff
writel 0xe1020424 0xffffffff
writeb 0xe102042b 0xff
write 0xe1020429 0x5 0x0055c5e5c0
write 0x5c041 0x3 0x0402e1
write 0x5c048 0x1 0x8a
write 0x5c04a 0x1 0x31
write 0x5c04b 0x1 0xff
write 0xe1020403 0x1 0xff
```

- Comes with scripts to convert crashing inputs into QTest scripts
- Automatically minimize the crash
- Crash reproducers can be included in email text, or even in a commit message


F 200719 2333 Alexander Bulek
F 200716 2121 Alexander Bulek Re: [PATCH] net: check payload length limit
F 200718 1039 Alexander Bulek Re: [Bug 1878057] Re: null-ptr dereference
F 200717 1304 Alexander Bulek
N F 200717 1235 Alexander Bulek [PATCH] fuzz: Fix leak when assembling data
N F 200717 1115 Alexander Bulek Re: practical security seminar
F 200717 0920 Alexander Bulek
F 200717 0903 Alexander Bulek
F 200716 1233 Alexander Bulek [PATCH] gitlab-ci.yml: Add oss-fuzz build t
F 200716 1246 Alexander Bulek Re: [PATCH] gitlab-ci.yml: Add fuzzer tests
N F 200715 0054 Alexander Bulek
N F 200617 0024 Alexander Bulek
F 200616 2302 Alexander Bulek
F 200615 2314 Alexander Bulek
r F 200615 2306 Alexander Bulek Re: QEMU Fuzzing Project
F 200714 1346 Alexander Bulek [PATCH] fuzz: Expect the cmdline in a freea
F 200713 1534 Alexander Bulek Re: [PATCH v2 8/9] hw/sd/sdcard: Update cod
F 200713 1206 Alexander Bulek Seminar next week
F 200713 0752 Alexander Bulek
F 200709 1948 Alexander Bulek
F 200623 1055 Alexander Bulek
F 200611 0156 Alexander Bulek [RFC PATCH 3/3] fuzz: Add callbacks for
F 200611 0156 Alexander Bulek [RFC PATCH 2/3] fuzz: add support for fu
F 200611 0156 Alexander Bulek [RFC PATCH 1/3] fuzz: add a general fuzz
F 200611 0156 Alexander Bulek [RFC PATCH 0/3] fuzz: add generic fuzzer
F 200712 1640 Alexander Bulek Re: [Bug 1887309] [NEW] Floating-point exce
F 200710 0923 Alexander Bulek Re: [PATCH] softmmu/vl: Be less verbose abo
N F 200709 1525 Alexander Bulek Re: Kevin misses you. :--)
F 200709 1257 Alexander Bulek [Fwd] Re: [PATCH v1 06/13] plugins: add API
N F 200709 1210 Alexander Bulek Re: confirm subscribe to virtio-dev@lists.o
F 200709 1125 Alexander Bulek Re: [PATCH] softmmu/vl: Include "qemu/rcu.h
F 200709 1047 Alexander Bulek [Fwd] [PATCH v7 00/21] Initial support for
+ 200709 1046 Alexander Bulek [Fwd] [PATCH v7 00/21] Initial support for
F 200709 0941 Alexander Bulek Re: [PATCH] tests/qtest/fuzz: Add missing s
F 200709 0938 Alexander Bulek
F 200708 1601 Alexander Bulek [PATCH-for-5.1 2/2] fuzz: add missing he
F 200708 1601 Alexander Bulek [PATCH-for-5.1 1/2] configure: do not cl
F 200708 1601 Alexander Bulek [PATCH-for-5.1 0/2] fuzz: broken build fixe
F 200708 1712 Alexander Bulek [ACM CCS 2020 B] Submission #399 Rebut
F 200703 0924 Alexander Bulek
F 200702 1301 Alexander Bulek
F 200702 1523 Alexander Bulek
r F 200701 1153 Alexander Bulek
alxndr@mozz:~/Development/qemu/build-asan(generic-fuzzer)$

OSS-Fuzz




```
      static ssize_t virtio_net_receive_rcu(NetClientState *nc, const uint8_t *buf,
                                            size_t size, bool no_rss)
17.4k {
17.4k     VirtIONet *n = qemu_get_nic_opaque(nc);
17.4k     VirtIONetQueue *q = virtio_net_get_subqueue(nc);
17.4k     VirtIODevice *vdev = VIRTIO_DEVICE(n);
17.4k     struct iovec mhdr_sg[VIRTQUEUE_MAX_SIZE];
17.4k     struct virtio_net_hdr_mrg_rxbuf mhdr;
17.4k     unsigned mhdr_cnt = 0;
17.4k     size_t offset, i, guest_offset;
17.4k
17.4k     if (!virtio_net_can_receive(nc)) {
    0         return -1;
    0     }
17.4k
17.4k     if (!no_rss && n->rss_data.enabled) {
    0         int index = virtio_net_process_rss(nc, buf, size);
    0         if (index >= 0) {
    0             NetClientState *nc2 = qemu_get_subqueue(n->nic, index);
    0             return virtio_net_receive_rcu(nc2, buf, size, true);
    0         }
17.4k     }
17.4k
          /* hdr_len refers to the header we supply to the guest */
17.4k     if (!virtio_net_has_buffers(q, size + n->guest_hdr_len - n->host_hdr_len)) {
13.3k         return 0;
13.3k     }
4.09k
4.09k     if (!receive_filter(n, buf, size))
    0         return size;
4.09k
4.09k     offset = i = 0;
4.09k
```

Bugs

The fuzzer has found:

- Old bugs that did not have reliable reproducers:
  - lsi_scsi bug from 2011.
  - UHCI bug from 2015
- Bugs that revealed architectural issues:
  - DMA re-entrancy issues
  - Memory access API
- > 50 reports on launchpad. 6 CVEs to date
- Combine with sanitizers to find heap-overflow, UAF, alignment, etc bugs

Open question: How to handle automated bug reports from oss-fuzz?

Fuzzing QEMU: The Future

- Fuzzing device backend code (SPICE, VNC, SLiRP, ...)
- Fuzzing migration/{Save, LoadVM} handlers and reboots
- Fuzzing to find timing-sensitive/double-fetch bugs
- Bugs that require more interactions than can fit in a single input
- Improving kernel fuzzing of virtualization-related components.
- Regression testing based on bug reproducers
- Multiprocess QEMU, vhost-user, vfio...
- Better ways to reset state between inputs

Interested?

`{alxndr,bsd} on #qemu`

alxndr@bu.edu

bsd@redhat.com

Stefan Hajnoczi

Darren Kenny

Thomas Huth

Li Qiang

Dima Stepanov

Philippe Mathieu-Daudé

Lidong Chen

Paolo Bonzini

Jon Maloy