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
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(  ?  )

The Input Space is Enormous!
State re-winds

- 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 0xcf8 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);
                break;
        }
    }
}
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

- 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

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
OSS-Fuzz

You are allowed to see jobs/libfuzzer_asan_qemu/honggfuzz_asan_qemu/libfuzzer_asan_qemu (including security)

Out-of-memory  Sun, Oct 4, 2020, 10:05 PM  [Image]
quemufuzz-1386-target-1440fx-qos-noreset-fuzz

Abctr  Wed, Sep 9, 2020, 7:06 AM  [Image]
guest_malloc
virtio_scsi_fuzz
virtio_scsi_fork_fuzz

ASSERT  Wed, Aug 8, 2020, 5:51 AM  [Image]
offset -- 0
iov_from_buf_full
iov_from_buf

ASSERT  Wed, Aug 8, 2020, 2:46 AM  [Image]
(g_get_monotonic_time() - start_time <= QVIRTIO_NET_TIMEOUT_US)
virtio_net_fuzz_multi
virtio_net_fork_fuzz_check_used

Out-of-memory  Thu, Jun 25, 2020, 2:05 AM  [Image]
quemufuzz-1386-target-1440fx-qtest-reboot-fuzz

static ssize_t virtio_net_receive_rcu(NodeClientState *nc, const uint8_t *buf, size_t size, bool mc_rx)
{
    VirtIONet *v = qemu_get_node_queue(nc);
    VirtIODevice *dev = VirtIO_DEVICE(n);
    struct iovc ndev[QVIRTIO_MAX_SIZE];
    struct virtio_net_hdr_net_rxbuf ndev;
    unsigned mdrcnt = 0;
    size_t offset, i, guest_offset;

    if (!virtio_net_can_receive(nc)) {
        return -5;
    }

    if (mc_rx & node_data_enabled)
    {
        int index = virtio_net_process_rxbufs(nc, buf, size);
        if (index >= 0) {
            NodeClientState *nc2 = qemu_get_subqueue(n->nic, index);
            return virtio_net_receive_rcu(nc2, buf, size, true);
            }
    }

    /* hdr_len refers to the header we supply to the guest */
    if (!virtio_net_has_buffers(q, size + n->guest_hdr_len - n->host_hdr_len)) {
        return 0;
    }

    if (receive_filter(n, buf, size))
        return size;

    if (offset = i = 0;
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

Dima Stepanov

Philippe Mathieu-Daudé

Li Qiang

Paolo Bonzini

Jon Maloy

Lidong Chen