V-SHUTTLE: Scalable and Semantics-Aware Hypervisor
Virtual Device Fuzzing

Gaoning Pan  Xingwei Lin  Xuhong Zhang  Yongkang Jia
Shouling Ji  Chunming Wu  Xinlei Ying  Jiashui Wang  Yanjun Wu

CCS 2021
Virtual machine architecture
- uses hypervisor, a.k.a. VMM
- provides strong isolation with virtualized hardware
- has each execution environment
- Virtual Machine Escape
  - A hypervisor has lots of code for services and hardware emulations
  - Privilege escalation: Guest -> Host

Virtual Machine Architecture

Applications

Guest OS

Hypervisor

Host OS

Hardware and Firmware
Background

Existing VM escape

- Storage device: Scavenger [Blackhat Asia’ 21], VENOM
- Graphics device: 3d Red Pill [Blackhat Asia’ 20]
- ......

- High bounty target in famous PWN competitions, like Pwn2Own and TianfuCup

TianfuCup @TianfuCup · Nov 7, 2020
The escape from #qemu is confirmed! Two bugs exploited: a uaf and an information-disclosure bug. $60,000 awarded to 360 ESG Vulnerability Research Institute @XiaoWei__ Congrats!

TianfuCup @TianfuCup · Nov 7, 2020
Wow, @XiaoWei__ contributed another successful entry, it's against target Ubuntu + qemu-kvm. VM escape achieved. Excelleeeeeeent!
Virtual Device Transaction

Guest

Guest Kernel Driver

① Access Register through MMIO

② Copy Pre-allocated Buffer in Guest through DMA

Host

Command Register

Base Address Register

…

MMIO Space

Hypervisor

Command Ring Buffer

Data

③ Execute Transactions

Memory Space

Main functionality

Complex structures
5 most popular QEMU device categories used in virtualization scenarios

<table>
<thead>
<tr>
<th>Category</th>
<th>Device (support DMA)</th>
<th>Number</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>USB</td>
<td>uhci, ehci, ohci, xhci</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Storage</td>
<td>esp, ahci, lsi53c810, megasas, mptsas, nvme, pvscsi, sdhci, virtio-blk, virtio-scsi, virtio-9p</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>Network</td>
<td>e1000, e1000e, eepro100, pcnet, rocker, rtl8139, tulip, vmxnet3, virtio-net</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>Display</td>
<td>(null)</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>Audio</td>
<td>ac97, cs4231a, es1370, intel-hda, sb16</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>Avg</td>
<td></td>
<td>29(72.5%)</td>
<td>40(100%)</td>
</tr>
</tbody>
</table>

72.5% of the devices support DMA and use it to transfer complex data
Core Challenge – Nested Structures

Feature 1: Nested Form Construction

- **Overall Level**: Higher-level tree structures and recursively defined.
- **Node Level**: Unknown pointer offset and unknown following node’s address.

Feature 2: Node Type Awareness

- **Overall Level**: Precise pointing relationships can only be known at runtime.
- **Node Level**: Fine-grained semantics of referred node types.
Motivating Example – USB UHCI

QEMU source code

```c
pci_dma_read (&s->dev, s->frame_addr, &link, 4);
...
for (; is_valid (link);) {
  ...
  if (is_qh (link)) {
    pci_dma_read (&s->dev, link, &qh, sizeof(qh));
    ...
    continue;
  }
  pci_dma_read (&s->dev, link, &td, sizeof(td));
  ...
pci_dma_read (&s->dev, td.buffer, buf, td.len);
/* main usb packet processing */
link = td.link;
}
```

Guest memory

Q. How can such hierarchically nested structures be generated exactly?
Related Work

- Random fuzzing to basic interfaces (MMIO, DMA, etc.):
  - VDF [Andrew et al., RAID’17]
  - Hyper-Cube [Schumilo et al., NDSS’20]

  **No knowledge** of the protocol implementation about DMA structures

- Apply expert-defined specifications to bridge the gap
  - Build structure-aware fuzzing against specifications that describe structures
  - Nyx [Schumilo et al., Security’21]

  Structure-specific rules heavily rely on domain knowledge (**time-consuming**)

Related Work

- Random fuzzing to basic interfaces (MMIO, DMA, etc.):
  - VDF [Andrew et al., RAID’17]
  - Hyper-Cube [Schumilo et al., NDSS’20]

Can we **avoid** such complex data structures building issues and make the fuzzing process **fully automatic** as well as domain knowledge free?

- Build structure-aware fuzzing against specifications that describe structures
- Nyx [Schumilo et al., Security’21]

Structure-specific rules heavily rely on domain knowledge (**time-consuming**).
Key Insight

Nested DMA Structures

Decoupled DMA Structures
Overview of V-SHUTTLE

**Fuzzer**
- Runs in host system
- **Persistent** mode to enable long-term fuzzing
- Collect coverage feedback
- Semantics-aware fuzzing via seedpools

**Fuzzing Agent**
- Runs in the hypervisor
- Emulate malicious drivers of the guest kernel
- Intercept all DMA and I/O accesses
1. DMA Redirection

Before:

```c
pci_dma_read(buffer_addr, &buf, size);
```

After:

```c
If (fuzzing_mode)
read_from_testcase(&buf, size);
```

API Hooking

Hypervisor

Guest Memory

Device Emulators

Fuzzed Input

one-dimensional vectors

Redirected DMA
1. DMA Redirection

Eliminate the **pointer** in each node while leaving the structure’s **semantics** intact.
Recall DMA Feature2: Dynamic Node Type

DMA sequence1

A
B
C

t1

DMA sequence2

A
C
B

t2

inconsistency

Learn

seed corpus

Learn

Finer-grained node-level semantics is required for coverage-guided fuzzing
2. Semantics-Aware Fuzzing via Seedpools

Static Analysis to Label DMA Objects

```
1. void uhci_process_frame ( ... ) {
2.     UHCI_QH qh;
3.     ... 
4.     if (is_qh) {
5.         pci_dma_read (&qh, sizeof(qh));
6.     }
7.     UHCI_TD td;
8.     ...
9.     uhci_read_td (&td);
10.    uhci_handle_td (...);
11.    ...
12. }
```

```
1. void uhci_read_td (UHCI_TD *td) {
2.     pci_dma_read (td, sizeof(*td));
3.     ...
4. }
```

```
1. void uhci_handle_td ( ... ) {
2.     UHCI_TD last_td;
3.     ...
4.     uhci_read_td (&last_td);
5.     ...
6. }
```
2. Semantics-Aware Fuzzing via Seedpools

Static Analysis to Label DMA Objects

1. void uhci_process_frame ( … ) {
2.      UHCI_QH qh;
3.      …
4.      if (is_qh) {
5.          pci_dma_read (&qh, sizeof(qh), 1);
6.      }
7.      UHCI_TD td;
8.      …
9.      uhci_read_td (&td, 2);
10.     uhci_handle_td (…);
11.     …
12. }

1. void uhci_read_td (UHCI_TD *td, id) {
2.     pci_dma_read (td, sizeof(*td), id);
3.     …
4. }

1. void uhci_handle_td (…) {
2.     UHCI_TD last_td;
3.     …
4.     uhci_read_td (&lastTd, 3);
5.     …
6. }

Control-Flow
Backward Data-Flow
2. Semantics-Aware Fuzzing via Seedpools

- Static Analysis to Label DMA Objects
- DMA Redirection with Type Constraints

```c
pci_dma_read(buffer_addr, &buf, size, id);
```

**<Before>**

**<After>**

If (fuzzing_mode) read_from_testcase (&buf, size, type_id);

Ø Static Analysis to Label DMA Objects
Ø DMA Redirection with Type Constraints
2. Semantics-Aware Fuzzing via Seedpools

- Static Analysis to Label DMA Objects
- DMA Redirection with Type Constraints
- Seedpool-Based Fuzzer Design

Learn from scratch

Fuzzed Input

Seed Pool

Fuzzer

Seedpool-Based Fuzzing
2. Semantics-Aware Fuzzing via Seedpools

- Static Analysis to Label DMA Objects
- DMA Redirection with Type Constraints
- Seedpool-Based Fuzzer Design
- Semantics-aware Fuzzing Process

Algorithm 1 Main semantics-aware fuzzing loop of V-SHUTTLE

**Input:** Initial seeds queues $Seedpool[]$, Target Hypervisor $H$

1. // setup each basic seed queues and global information
2. for all queue of the $Seedpool[]$ do
3. queue.setup();
4. end for
5. $GlobalMap$.init();
6. repeat
7. $id = H$.request()
8. $seed = Mutate(Seedpool[id])$;
9. $Cover = H$.feed($seed$);
10. if $Cover$.haveNewCoverage() then
11. $Seedpool[id]$.push($seed$)
12. end if
13. until timeout or abort-signal;

**Output:** Crashing seeds crashes
3. Lightweight Fuzzing Loop

Device Instance Initialization → Fuzzing Entry Setup → Main Fuzzing Loop → MMIO/PIO Callbacks

① Retrieve MMIO/PIO Operation Callbacks  ② Start Fuzzing  ③ Invoke Device Callbacks explicitly  ④ Transaction Processed  ⑤ Repeat periodically

Fuzzing Agent

Virtual Device

Hypervisor

Ring3 harness - --> Lightweight, Driver-free, Easily implemented
Evaluations
Experiment settings

- Two well-known hypervisors: QEMU 5.1.0, VirtualBox 6.1.14
- Build with ASAN to discover bugs
- Gcov-based coverage measurement
- Each hypervisor instance is tested for 24 hours
Scalability

- Code coverage on **16** popular QEMU devices: Audio, Graphics, Network, USB, Storage
- Our solution has **tolerable overhead** as compared to the traditional dumb fuzzing

<table>
<thead>
<tr>
<th>Device</th>
<th>Line Coverage</th>
<th>Functions Coverage</th>
<th>Branches Coverage</th>
<th>Speed(exec/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Initial</td>
<td>Total</td>
<td>Initial</td>
<td>Total</td>
</tr>
<tr>
<td>Audio</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CS4231a</td>
<td>30.00%</td>
<td>96.10%</td>
<td>57.10%</td>
<td>100.00%</td>
</tr>
<tr>
<td>Intel-HDA</td>
<td>68.30%</td>
<td>95.00%</td>
<td>78.60%</td>
<td>95.20%</td>
</tr>
<tr>
<td>ES1370</td>
<td>54.20%</td>
<td>99.62%</td>
<td>73.70%</td>
<td>100.00%</td>
</tr>
<tr>
<td>SoundBlaster</td>
<td>12.30%</td>
<td>99.19%</td>
<td>28.60%</td>
<td>100.00%</td>
</tr>
<tr>
<td>Graphics</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ATI-VGA</td>
<td>27.40%</td>
<td>86.00%</td>
<td>66.70%</td>
<td>80.00%</td>
</tr>
<tr>
<td>Network</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E1000</td>
<td>36.20%</td>
<td>94.20%</td>
<td>46.90%</td>
<td>96.90%</td>
</tr>
<tr>
<td>NE2000</td>
<td>6.70%</td>
<td>89.60%</td>
<td>28.60%</td>
<td>100.00%</td>
</tr>
<tr>
<td>PCNET</td>
<td>24.60%</td>
<td>97.40%</td>
<td>44.80%</td>
<td>100.00%</td>
</tr>
<tr>
<td>RTL8139</td>
<td>28.10%</td>
<td>97.60%</td>
<td>59.10%</td>
<td>97.70%</td>
</tr>
<tr>
<td>USB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UHCI</td>
<td>81.30%</td>
<td>89.10%</td>
<td>86.10%</td>
<td>88.90%</td>
</tr>
<tr>
<td>EHCI</td>
<td>40.70%</td>
<td>82.70%</td>
<td>53.40%</td>
<td>89.00%</td>
</tr>
<tr>
<td>OHCI</td>
<td>46.90%</td>
<td>83.70%</td>
<td>65.10%</td>
<td>86.00%</td>
</tr>
<tr>
<td>Storage</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NVME</td>
<td>38.60%</td>
<td>72.40%</td>
<td>47.30%</td>
<td>76.40%</td>
</tr>
<tr>
<td>Lsi53c895a</td>
<td>26.90%</td>
<td>79.00%</td>
<td>46.70%</td>
<td>71.10%</td>
</tr>
<tr>
<td>Megasas</td>
<td>58.10%</td>
<td>63.80%</td>
<td>68.30%</td>
<td>70.00%</td>
</tr>
<tr>
<td>AHCI</td>
<td>75.30%</td>
<td>81.80%</td>
<td>78.60%</td>
<td>82.10%</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>40.98%</strong></td>
<td><strong>87.95%</strong></td>
<td><strong>58.10%</strong></td>
<td><strong>89.58%</strong></td>
</tr>
</tbody>
</table>
Code Coverage Enhancement

- Comparison of Dumb Fuzzing, Structure-Aware Fuzzing, V-SHUTTLE Main Framework, V-SHUTTLE with Semantics-Aware Fuzzing Mode

- V-SHUTTLE performs better with the semantics-aware fuzzing mode

(a) UHCI
(b) OHCI
(c) EHCI
Code Coverage Enhancement

- Compared with state-of-the-art hypervisor fuzzers
  - VDF [RAID'17], Hyper-Cube [NDSS'20], Nyx [Sec'21]

- V-SHUTTLE presents coverage improvement over the others

<table>
<thead>
<tr>
<th>Device</th>
<th>VDF</th>
<th>Hyper-Cube</th>
<th>Nyx</th>
<th>V-Shuttle</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cov</td>
<td></td>
<td>Cov</td>
<td>Cov</td>
</tr>
<tr>
<td>CS4231a</td>
<td>56.00%</td>
<td>74.76%</td>
<td>74.76%</td>
<td><strong>85.80%</strong></td>
</tr>
<tr>
<td>Intel-HDA</td>
<td>58.60%</td>
<td>79.17%</td>
<td>78.33%</td>
<td>78.30%</td>
</tr>
<tr>
<td>ES1370</td>
<td>72.70%</td>
<td>91.38%</td>
<td>91.38%</td>
<td>91.91%</td>
</tr>
<tr>
<td>SoundBlaster</td>
<td>81.00%</td>
<td>83.80%</td>
<td>81.34%</td>
<td>81.52%</td>
</tr>
<tr>
<td>E1000</td>
<td>81.60%</td>
<td>66.08%</td>
<td>54.55%</td>
<td><strong>74.50%</strong></td>
</tr>
<tr>
<td>NE2000</td>
<td>71.70%</td>
<td>71.89%</td>
<td>71.89%</td>
<td>71.90%</td>
</tr>
<tr>
<td>PCNET</td>
<td>36.10%</td>
<td>78.71%</td>
<td>89.49%</td>
<td>88.90%</td>
</tr>
<tr>
<td>RTL8139</td>
<td>63.00%</td>
<td>74.68%</td>
<td>79.28%</td>
<td><strong>88.40%</strong></td>
</tr>
</tbody>
</table>
Vulnerability Discovery

- Discovered new vulnerabilities
  - 35 new vulnerabilities found in QEMU and VirtualBox with 17 CVE assigned
  - UAF, Integer overflow, OOB access, etc., including high-impact exploitable vulnerabilities

- Reasonable time to rediscover previously known vulnerabilities

---

### Hypervisor

<table>
<thead>
<tr>
<th>Bug</th>
<th>Description</th>
<th>Device Type</th>
<th>CVE/Issue-ID</th>
<th>CVSS Score</th>
<th>Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2020-25625</td>
<td>OHCI infinite loop</td>
<td>USB</td>
<td>CVE-2020-25624</td>
<td>5.0</td>
<td>DoS</td>
</tr>
<tr>
<td>CVE-2020-25805</td>
<td>SDHCI Heap buffer overflow</td>
<td>USB</td>
<td>confirmed</td>
<td>-</td>
<td>DoS</td>
</tr>
<tr>
<td>CVE-2021-20257</td>
<td>E1000 infinite loop</td>
<td>USB</td>
<td>CVE-2020-25617</td>
<td>3.8</td>
<td>DoS</td>
</tr>
<tr>
<td>CVE-2020-25804</td>
<td>EHCI use-after-free</td>
<td>USB</td>
<td>CVE-2020-25661</td>
<td>3.8</td>
<td>DoS</td>
</tr>
<tr>
<td>CVE-2020-25804</td>
<td>ATI-VGA integer overflow</td>
<td>USB</td>
<td>CVE-2020-25617</td>
<td>3.8</td>
<td>DoS</td>
</tr>
<tr>
<td>CVE-2021-20257</td>
<td>E1000 infinite loop</td>
<td>USB</td>
<td>CVE-2020-25661</td>
<td>3.8</td>
<td>DoS</td>
</tr>
<tr>
<td>CVE-2021-20257</td>
<td>E1000 infinite loop</td>
<td>USB</td>
<td>CVE-2020-25661</td>
<td>3.8</td>
<td>DoS</td>
</tr>
<tr>
<td>CVE-2020-25625</td>
<td>OHCI infinite loop</td>
<td>USB</td>
<td>CVE-2020-25624</td>
<td>5.0</td>
<td>DoS</td>
</tr>
<tr>
<td>CVE-2020-25805</td>
<td>SDHCI Heap buffer overflow</td>
<td>USB</td>
<td>CVE-2020-25624</td>
<td>5.0</td>
<td>DoS</td>
</tr>
<tr>
<td>CVE-2021-20257</td>
<td>E1000 infinite loop</td>
<td>USB</td>
<td>CVE-2020-25624</td>
<td>5.0</td>
<td>DoS</td>
</tr>
<tr>
<td>CVE-2020-25804</td>
<td>EHCI use-after-free</td>
<td>USB</td>
<td>CVE-2020-25624</td>
<td>5.0</td>
<td>DoS</td>
</tr>
<tr>
<td>CVE-2020-25804</td>
<td>ATI-VGA integer overflow</td>
<td>USB</td>
<td>CVE-2020-25624</td>
<td>5.0</td>
<td>DoS</td>
</tr>
<tr>
<td>CVE-2020-25625</td>
<td>OHCI infinite loop</td>
<td>USB</td>
<td>CVE-2020-25624</td>
<td>5.0</td>
<td>DoS</td>
</tr>
<tr>
<td>CVE-2020-25805</td>
<td>SDHCI Heap buffer overflow</td>
<td>USB</td>
<td>CVE-2020-25624</td>
<td>5.0</td>
<td>DoS</td>
</tr>
<tr>
<td>CVE-2021-20257</td>
<td>E1000 infinite loop</td>
<td>USB</td>
<td>CVE-2020-25624</td>
<td>5.0</td>
<td>DoS</td>
</tr>
<tr>
<td>CVE-2020-25804</td>
<td>EHCI use-after-free</td>
<td>USB</td>
<td>CVE-2020-25624</td>
<td>5.0</td>
<td>DoS</td>
</tr>
<tr>
<td>CVE-2020-25804</td>
<td>ATI-VGA integer overflow</td>
<td>USB</td>
<td>CVE-2020-25624</td>
<td>5.0</td>
<td>DoS</td>
</tr>
</tbody>
</table>

---

### QEMU

- Heap buffer overflow (write) in ohci_copy_iso_td
- Stack buffer overflow (read) in ohci_service_iso_td
- Heap buffer overflow (read) in ohci_service_td
- Infinite loop in e1000e_write_packet_to_guest
- OOB access in ati_2d_blt
- Reachable assert failure via eth_get_gso_type
- Divide by zero in dwc2_handle_packet
- Integer Overflow in smbd2_operation
- Infinite loop in xhci_ring_length
- Heap-use-after-free in nic_reset
- Heap buffer overflow (write) in dp8393x_do_transmit_packets
- Failed malloc in omap_rfb1_transfer_start
- Infinite loop in allwinner_sun8i_emac_get_desc
- Divide by zero in exynos4210_lttick_cnt_get_cnto
- Divide by zero in zynq_sclr_compute_pll
- Failed malloc in vmxnet3_activate_device
- NULL pointer dereference in fdctrl_read
- Heap-use-after-free in ehci_flush_qh
- NULL pointer dereference in ls153c935a
- NULL pointer dereference in import_oport_read
- NULL pointer dereference in a9_timer_get_current_cpu
- Assertion in usb_msd_send_status
- Assertion in usb_ep_get
- Assertion in ohci_frame_boundary
- Assertion in vmxnet3_io_bar1_write
- Assertion in ls1_do_dma

---

### VirtualBox

- Heap buffer overflow (write) in xhciR3WriteEvent
- Heap buffer overflow (write) in xhcr3ServiceQDH
- OOB Read in ehciR3ServiceQDH
- Divide by zero in e1kTxDloadMore
- Integer overflow in e1kGetTxLen
- Heap buffer overflow (write) in buslogicRegisterWrite
- Divide by zero in atar3SetSector
- NULL pointer dereference in blk_read
- Uninitialized stack object in ls1LogicCS1

---

### Details

- CVE-2020-25625: OHCI infinite loop
- CVE-2020-25805: SDHCI Heap buffer overflow
- CVE-2021-20257: E1000 infinite loop
- CVE-2020-25804: EHCI use-after-free
- CVE-2020-11869: ATI-VGA integer overflow
QEMU: **USB-OHCI** Out-of-Bounds Access

```c
static int ohci_service_iso_td(OHCIState *ohci, struct ohci_ed *ed,
    int completion)
{
    if (ohci_read_iso_td(ohci, addr, &iso_td)) {
        if (ohci_read_iso_td(ohci, addr, &iso_td)) {
            trace_usb_ohci_iso_td_read_failed(addr);
            ohci_die(ohci);
            return 1;
        }
    }
    if ((start_addr & OHCI_PAGE_MASK) != (end_addr & OHCI_PAGE_MASK)) {
        len = (end_addr & OHCI_OFFSET_MASK) - 0x1001
              - (start_addr & OHCI_OFFSET_MASK);
    } else {
        len = end_addr - start_addr + 1;
    }
    if (len & dir) { /* OHCI_TD_DIR_IN */
        if (ohci_copy_iso_td(ohci, start_addr, end_addr, ohci->usb_buf,
            len, 0, DMA_DIRECTION_TO_DEVICE)) {
            ohci_die(ohci);
            return 1;
        }
    }
}
```

**Reading iso_td**

**Vulnerable point**

**Crash point**

**Process schedule**

**Endpoint descriptor list**

**Transfer descriptor queues**
Deployment and Application

- V-SHUTTLE performs better with the semantics-aware fuzzing mode
- V-SHUTTLE’s can be ported to Ant Group’s commercial platform with little efforts
  - Lightweight: Takes about an hour to implement V-SHUTTLE into a new hypervisor via static analysis, some simple configurations and instrumentation
Limitation and future work

- Automatic PoC reconstruction under persistent fuzzing.
- Supporting closed-source hypervisors by applying binary analysis technique.
- Fine-grained awareness of hypervisor internal states.
We systematically study the driver-device interaction in virtual machine transaction and reveal that the data structures transferred via DMA have nested features.

The first hypervisor fuzzer that automatically handles nested structures by semantics-aware DMA redirection

Discovered 35 vulnerabilities with 17 CVEs assigned, and presented the better code coverage, compared to state-of-the-arts

V-SHUTTLE: https://github.com/hustdebug/v-shuttle