ViDeZZo: Dependency-aware Virtual Device Fuzzing

Qiang Liu (Zhejiang University; EPFL) Flavio Toffalini (EPFL)
Yajin Zhou (Zhejiang University) Mathias Payer (EPFL)

浙江大學
ZHEJIANG UNIVERSITY

EPFL

Who I Am

Qiang Liu

Visiting Ph.D. student -> Post.Doc. from 2023.11

- HexHive, led by Prof. Mathias Payer, EPFL, Switzerland
- Topics: **Hypervisor Fuzzing (S&P'23)**, OS Fuzzing, Network Protocol Fuzzing

Ph.D. student since 2018.09

- BlockSec, led by Prof. Yajin Zhou, Zhejiang University, China
- Topics: Rehosting (FirmGuide ASE'21, ECMO CCS'21)

ViDeZZo: Dependency-aware Virtual Device Fuzzing

Part 3

Part 2

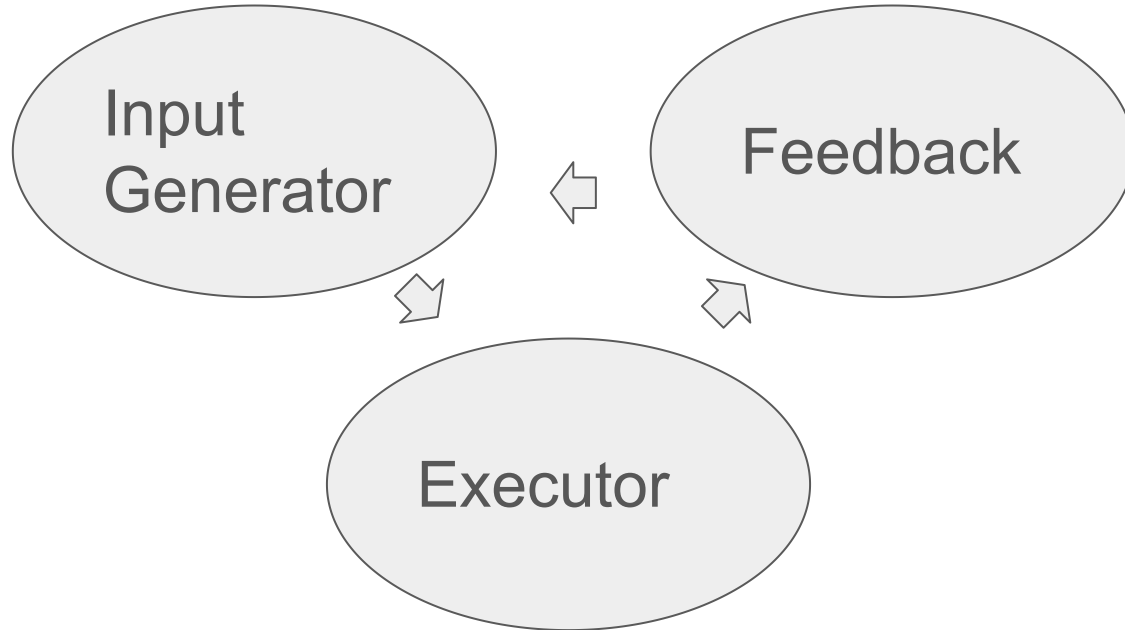
Part 1

浙江大學
ZHEJIANG UNIVERSITY

EPFL

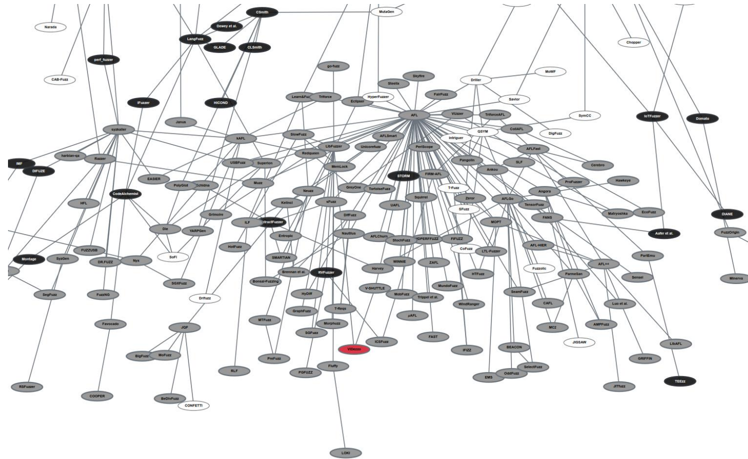
hexhive

Fuzzing or fuzz testing generates a lot of test cases and monitors the executions for defects[1]

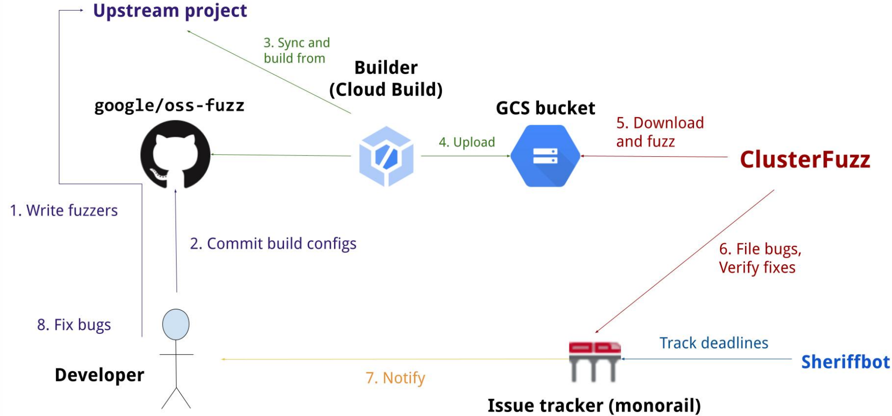


[1] Xiaogang Zhu, et. al. 2022. Fuzzing: A Survey for Roadmap. ACM Comput. Surv.

Fuzzing has been very successful in looking for vulnerabilities for user applications

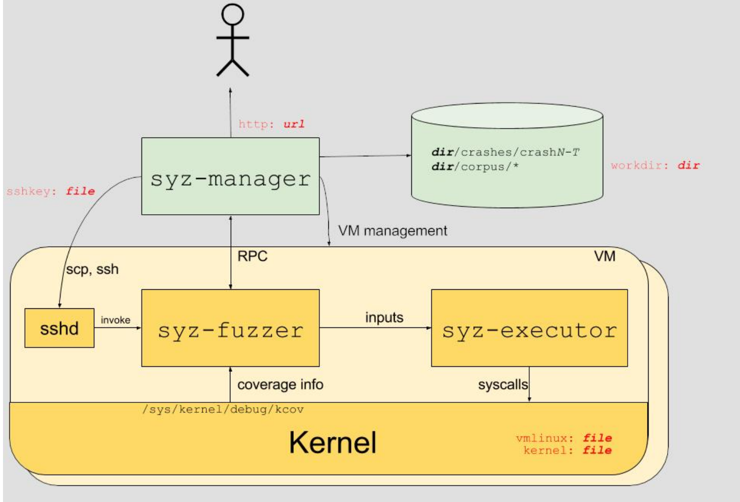


https://fuzzing-survey.org/

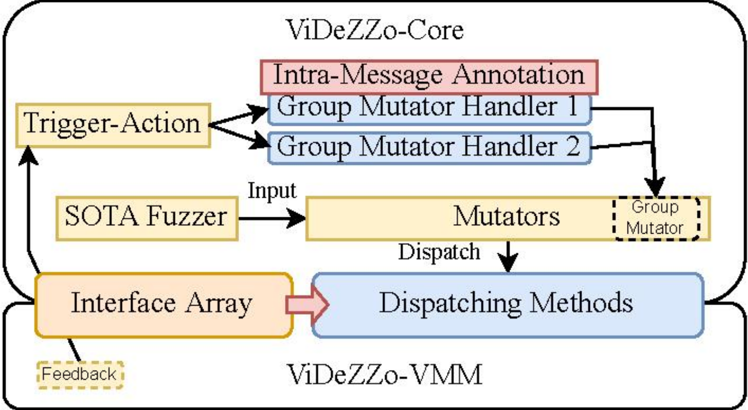


https://github.com/google/oss-fuzz

Fuzzing has been heavily applied to low-level system software (Operating System, Hypervisor, etc.)

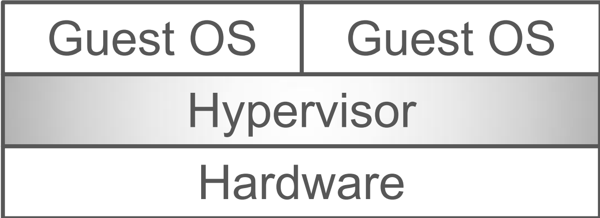


https://github.com/google/syzkaller

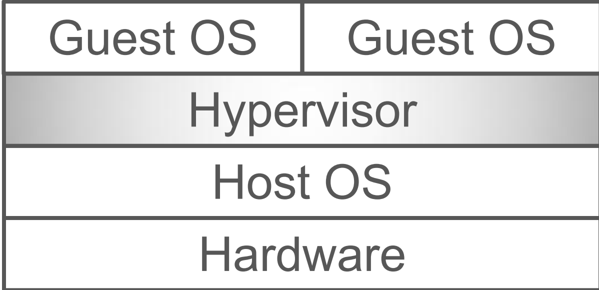


ViDeZZo

Hypervisor must guarantee the isolation between the guests and the host



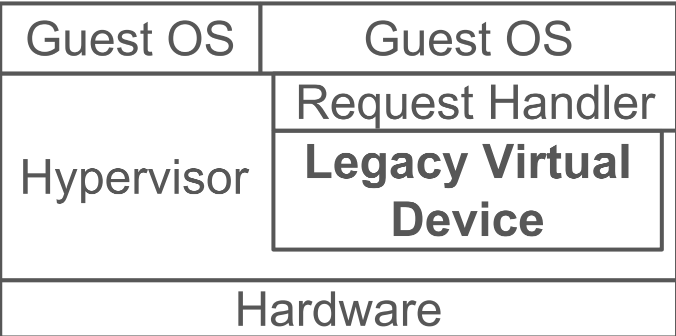
Type-I Hypervisor runs directly on hardware, e.g., Hyper-V, VMware ESXi



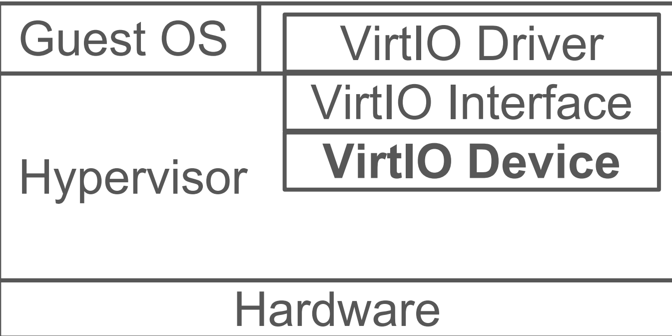
Type-II Hypervisor runs on the host OS, e.g., QEMU/KVM, VirtualBox, VMWare Workstation

What virtual devices are

Virtual Device = Legacy Virtual Device + VirtIO Device



Legacy Virtual Device

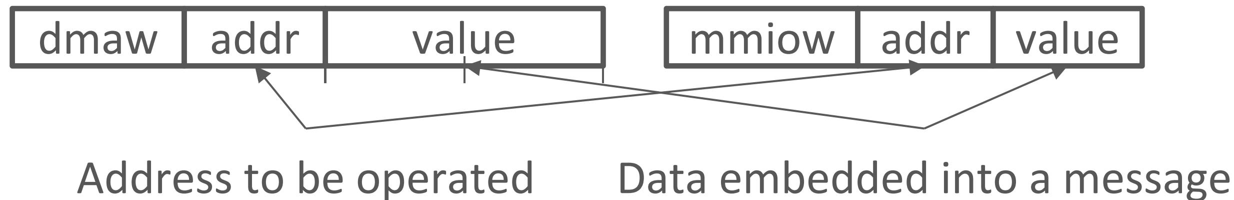


VirtIO Device

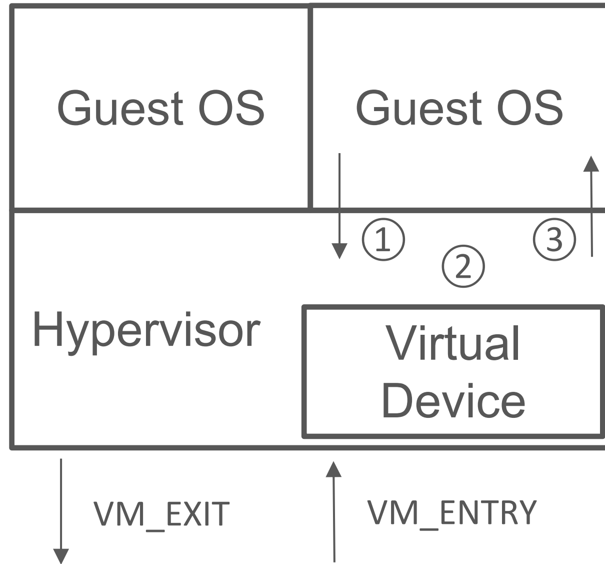
Guests interact with virtual devices through structural and sequential virtual device messages

There are different ways to interact with virtual devices

- I/O Accesses
  - Memory-Mapped I/O (MMIO)
  - Port-Mapped I/O (PIO)
  - DMA channels
- Time Management Operations



How a virtual device work



① dma_write(addr=**0x1000**, value=**’\xde\xed\xbe\xef’**)
mmio_write(addr=**0xffff00b0**, value=**0x1000**)

② Virtual device handles the messages

```
void mmio_write_handler(uint64_t addr, uint64_t value) {
        switch(addr) {
                case 0xb0;
                        dma_start=value;
                        dma_read(dma_start, &deadbeef);
                        break;
```

③ Return to the Guest OS

Virtual device remains the biggest attack surface

There are so many vulnerabilities in virtual devices

- 57.4% (252/439) QEMU vulnerabilities were in virtual devices
- 41.5% (22/53) of VM escapes were due to vulnerabilities in virtual devices

Outline

Key Challenges

- Intra-Message Dependency
- Inter-Message Dependency

Corresponding Solutions

- Intra-Message Annotation
- Inter-Message Mutation
- Fuzzing Workflow

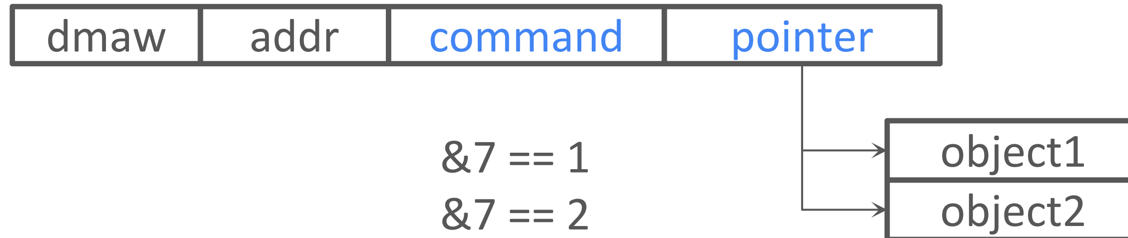
Evaluation

Key Challenge 1: Intra-Message Dependency

A field in a virtual device message may be dependent on another field

Example 1

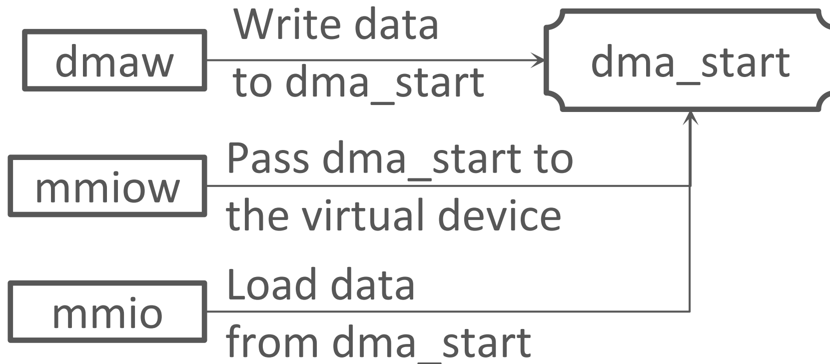
- Pointer points to something
- It depends on the value of command



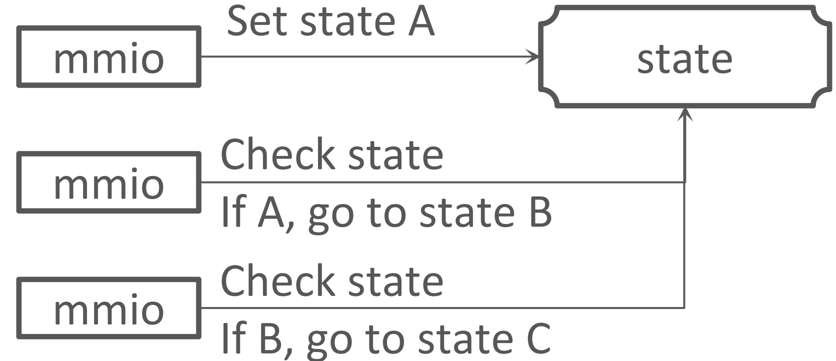
Key Challenge 2: Inter-Message Dependency

A message may depend on a previously issued message

Example 2



Example 3



Solution 1: Semi-automatically construct intra-message annotation from source code

```
vd0=Model(‘tx’, 0)
vd0.add_struct(‘tx_t’, {})
vd0.add_flag(‘tx_t.command’, {})

vd0.add_point_to(‘tx_t.address’, …)
```

Intra-Message Annotation

Virtual Device Source Code

Generated Messages w/
Intra-Message Dependency

Intra-message annotation

- Type system: FLAG, POINTER, etc.
- APIs for intra-message dependencies
- Programming model: Model()

```
1  vd0 = Model('tx', 0)
2  vd0.add_head(['tx_t'])
3  vd0.add_struct('tx_t', {
4    'command#0x4': 'FLAG',
5    'array_addr#0x4': 'POINTER'})
6  vd0.add_flag('tx_t.command', {0: 3})
7  vd0.add_point_to('tx_t.array_addr',
8    [None, 'macaddr', 'config', None, None, None,
9  //if 0      1          2        3     4     5
10   None, None], condition=['tx_t.command.0'])
11 //  6     7               == tx_t.command.0
```

Automatic extraction is based on taint analysis

- Start from pci_dma_read()
- Get the type of the destination buffer
- Decide a flag if a field flows to binary operations
- Decide a pointer if a field flows to specific functions

```
pci_dma_read(from=addr, to=&tx);

structs/enums

tx.command & 0x3

pci_dma_[read|write](
  from=tx.array_addr)
```

Current intra-message annotation is a good start, but can be improved

- Be more formal
- Support C structs/enums
- Support more types of dependencies
- Simplify the programming model

Solution 2: Automatically learn the dependency with new mutators during fuzzing

Message Level	ChangeAddr, etc.
Sequence Level	ShuffleMessages, etc.
Group Level	GroupMessage

Multi-level Mutators

Raw Virtual Device Messages

Mutated Messages (saved if interesting)

Inter-message mutators are beneficial but can be improved

Benefits

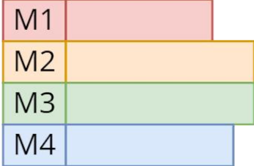
- Go beyond the byte mutators
- Capture different granularities

Some ideas

- Support weighted mutators
- Support dictionaries
- Support better mutator scheduling

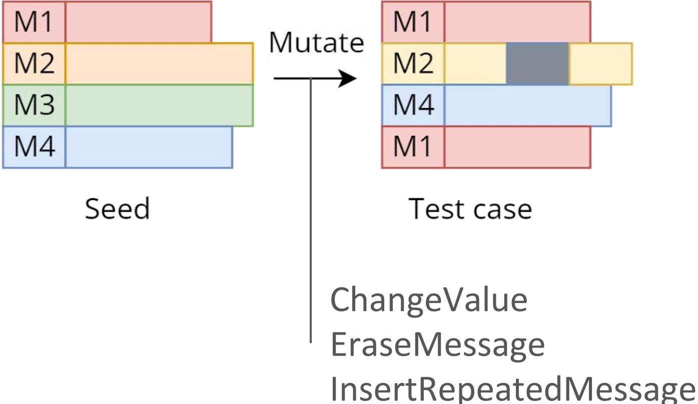
Inter-message dependency is far away being addressed (will discuss later)

Fuzzing Workflow

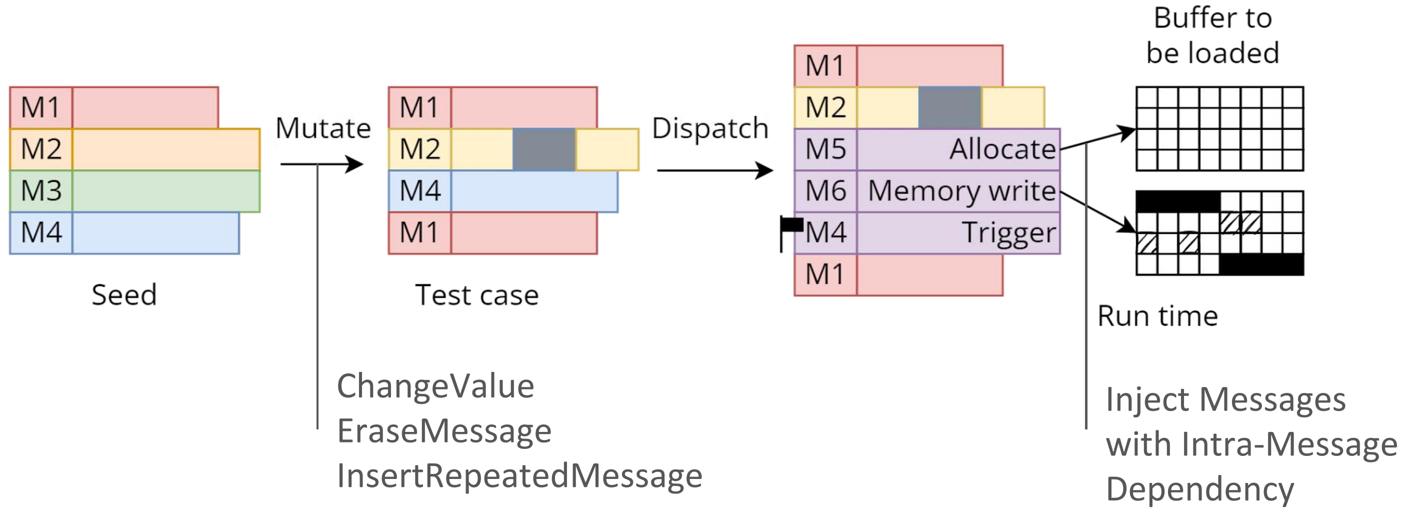


Seed

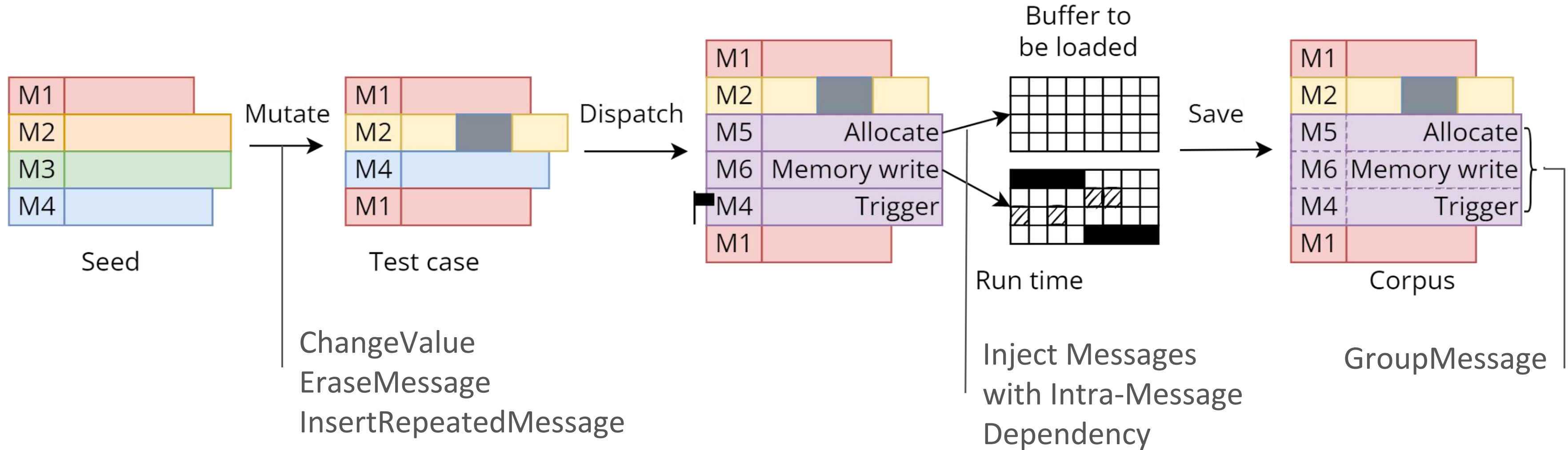
Fuzzing Workflow



Fuzzing Workflow



Fuzzing Workflow



Evaluation

Be scalable and efficient

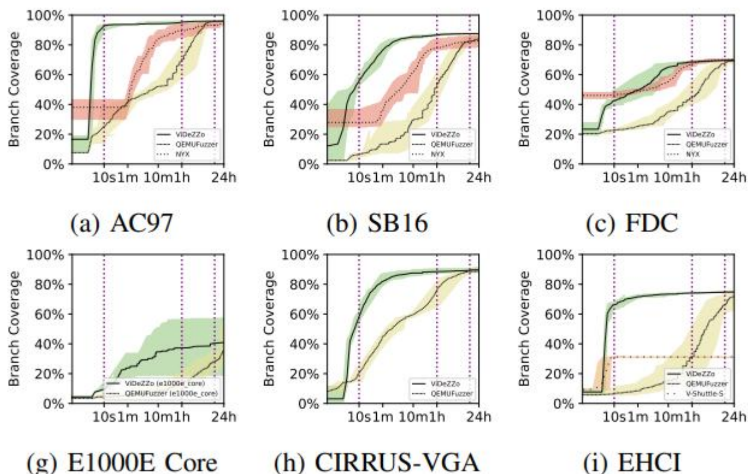


Figure 11: Branch coverage over 24 hours of virtual devi
VIDEZZO. The shadows show the minimum/maximum cov

Be effective

Id	Target	Category	VMM	Version	Arch	Short Description	# of Messages	Reported By	Status
1	ac97	audio	qemu	7.0.94	i386	Abort in audio_calloc()	1	An, Vi	Fixed
2	am53c974	storage	qemu	6.1.50	i386	Null pointer access in do_busid_cmd()	N/A	An	Fixed
3	ati	display	qemu	6.1.50	i386	Out of bounds write in ati_2d_blt()	N/A	VS	Fixed
4	ati	display	qemu	7.0.94	i386	Out of bounds write in ati_2d_blt()	4	Vi	*Fixed*
5	cadence_uart	serial	qemu	7.2.50	aarch64	Devision by zero in uart_parameters_setup()	2	Vi	*Fixed*
6	dwc2	usb	qemu	6.1.50	aarch32	Assertion failed in hw/usb/core.c from dwc2	N/A	Vi	Patch submitted
7	e1000	net	qemu	6.1.50	i386	Infinite loop in process_tx_desc()	N/A	Ny, VS, QF	Fixed
8	ehci	usb	qemu	6.1.50	i386	Abort in usb_ep_get()	N/A	Ny, VS, QF	Patch submitted
9	ehci	usb	qemu	6.1.50	i386	Assertion failure in address_space_unmap()	N/A	Ny, VS, QF	Fixed

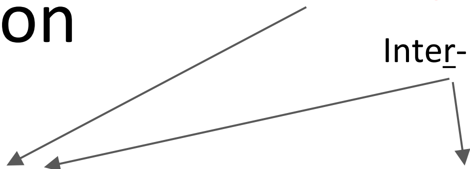
Bug	Description	V-SHUTTLE	QEMFUZZER	VIDEZZO
CVE-2020-11869	ATI-VGA IO	35.6M	—	782K (98.0K–2.85M)
CVE-2020-25084	EHCI UAF	79.4M	1.80M (1.36–2.23M)	44.0M (11.7M–88.8M)
CVE-2020-25085	SDHCI HBO	8.88M	1.58M (1.28M–1.85M)	32.3M (1.74M–114M)
CVE-2020-25625	OHCI IL	40.5M	TIMEOUT	2.22K (1.02K–6.22K)
CVE-2021-20257	E1000 IL	235K	TIMEOUT	283K (101K–618K)

Id	Target	Category	VMM	Version	Arch	Short Description	# of Messages	Reported By	Status
27	sb16	audio	qemu	6.1.50	i386	Assertion failure in audio_calloc() caused by sb16	N/A	An	Fixed
28	sb16	audio	qemu	6.1.50	i386	Abort in audio_calloc()	4	Vi	*Fixed(us)*
29	sdhci	storage	qemu	7.1.50	i386	Heap buffer overflow in sdhci_read_dataport()	9	QF	Fixed
30	smc91c111	net	qemu	7.1.93	aarch32	Out of bounds read/write in smc91c111	5	Vi	Open
31	tc6393xb	display	qemu	7.2.50	aarch32	negative-size-param in nand_blk_load_512()	23	Vi	Open
32	tc6393xb	display	qemu	7.2.50	aarch32	Heap buffer overflow in nand_blk_write_512()	7	Vi	Open
33	virtio-blk	storage	qemu	7.0.94	i386	Assertion failure in address_space_stw_le_cached()	5	An	Fixed
34	virtio-blk	storage	qemu	7.0.94	i386	Infinite loop in virtio_blk_handle_vq()	16	An	Fixed
35	vmxnet3	net	qemu	6.1.50	i386	Code should not be reached vmxnet3_io_bar1_write()	N/A	VS, Vi	Fixed
36	vmxnet3	net	qemu	6.1.50	i386	Three hw_error() in vmxnet3_validate_queues()	N/A	QF	Fixed
37	vmxnet3	net	qemu	6.1.50	i386	Assertion failed in vmxnet3_io_bar0_write()	N/A	QF	Fixed
38	vmxnet3	net	qemu	6.1.50	i386	Out of memory net_tx_pkt_init()	N/A	QF, VS	Fixed
39	vmxnet3	net	qemu	6.1.50	i386	Assertion failure in net_tx_pkt_reset()	N/A	QF	Fixed
40	vmxnet3	net	qemu	6.1.50	i386	eth_get_gso_type: code should not be reached	N/A	QF, VS	Fixed
41	xhci	usb	qemu	7.0.94	i386	Abort in xhci_find_stream()	56	Vi	*Fixed(us)*
42	xlnx_dp	display	qemu	7.0.91	aarch64	Abort in xlnx_dp_aux_set_command()	1	Vi	*Fixed(us)*
43	xlnx_dp	display	qemu	6.1.50	aarch64	Out of bounds read in xlnx_dp_read()	1	Vi	*Fixed(us)*
44	xlnx_dp	display	qemu	6.1.50	aarch64	Out of bounds in xlnx_dp_vblend_read()	N/A	An	Fixed
45	xlnx_dp	display	qemu	7.2.50	aarch64	Overflow in xlnx_dp_aux_push_rx_fifo()	3	Vi	Patch submitted
46	xlnx_dp	display	qemu	7.2.50	aarch64	Abort in xlnx_dp_change_graphic_fmt()	1	Vi	Patch submitted
47	xlnx_dp	display	qemu	7.2.50	aarch64	Underflow in xlnx_dp_aux_pop_tx_fifo()	1	Vi	Patch submitted
48	xlnx_dp	display	qemu	7.2.50	aarch64	Overflow in xlnx_dp_aux_push_tx_fifo()	17	Vi	Patch submitted
49	xlnx_zynqmp_can	net	qemu	7.2.50	aarch64	Fifo underflow in transfer_fifo()	2	Vi	Open
50	xlnx_zynqmp_can	net	qemu	7.2.50	aarch64	Fifo overflow in transfer_fifo()	291	Vi	Open
51	xlnx_zynqmp_qspips	spi	qemu	7.2.50	aarch64	Out of bound in xilinx_spips_write()	1	Vi	Open
52	xlnx_zynqmp_qspips	spi	qemu	7.2.50	aarch64	Underflow in xlnx_dp_aux_push_rx_fifo()	2	Vi	Open

Evaluation

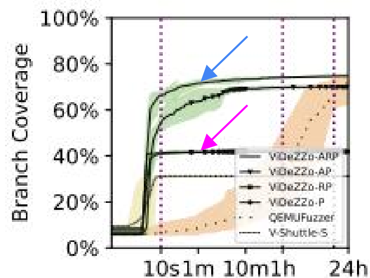
Intra-Message Dependency

Inter-Message Dependency

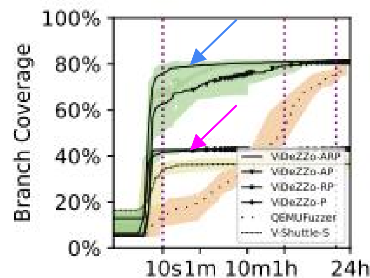


ViDeZZo-ARP v.s. ViDeZZo-RP

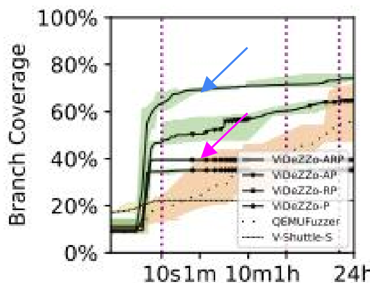
- Intra-message annotation contributes!



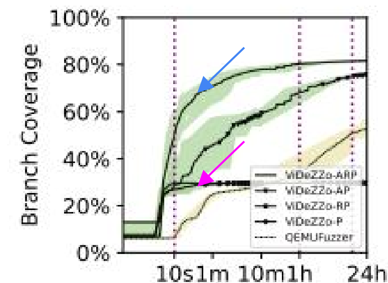
(a) EHCI



(b) OHCI



(c) UHCI

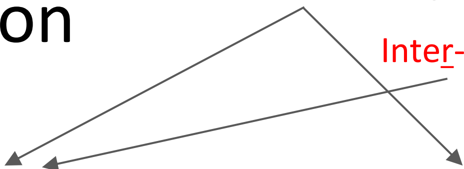


(d) XHCI

Evaluation

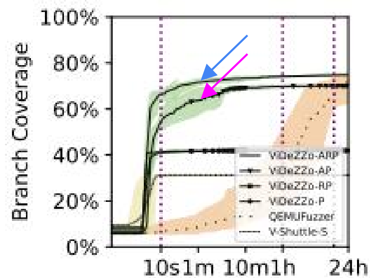
Intra-Message Dependency

Inter-Message Dependency

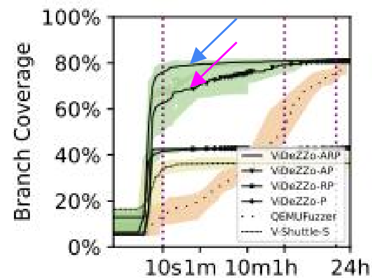


ViDeZZo-ARP v.s. ViDeZZo-AP

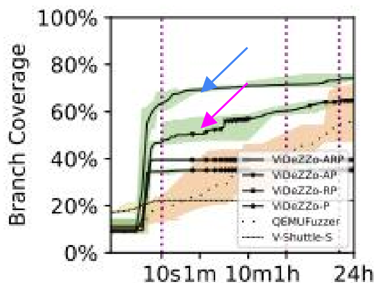
- Inter-message mutators contribute!



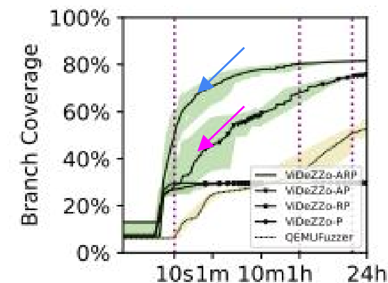
(a) EHCI



(b) OHCI



(c) UHCI



(d) XHCI

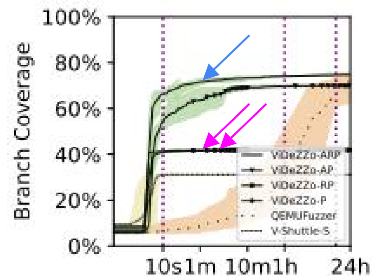
Evaluation

Intra-Message Dependency

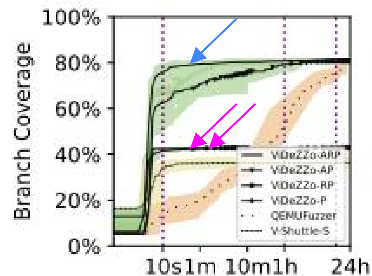
Inter-Message Dependency

ViDeZZo-ARP and ViDeZZo-RP/P

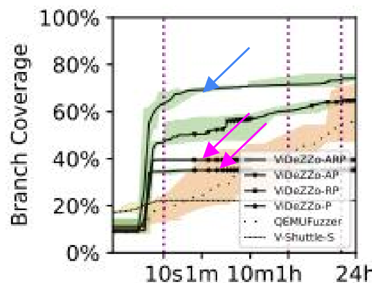
- ARP > RP=P
- Inter-message mutators are more effective when the intra-message annotation is supported



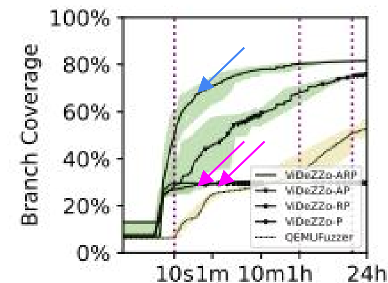
(a) EHCI



(b) OHCI

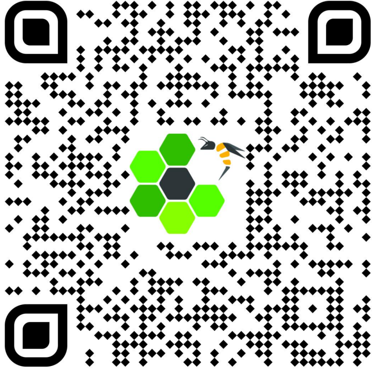


(c) UHCI



(d) XHCI

ViDeZZo: Dependency-aware Virtual Device Fuzzing



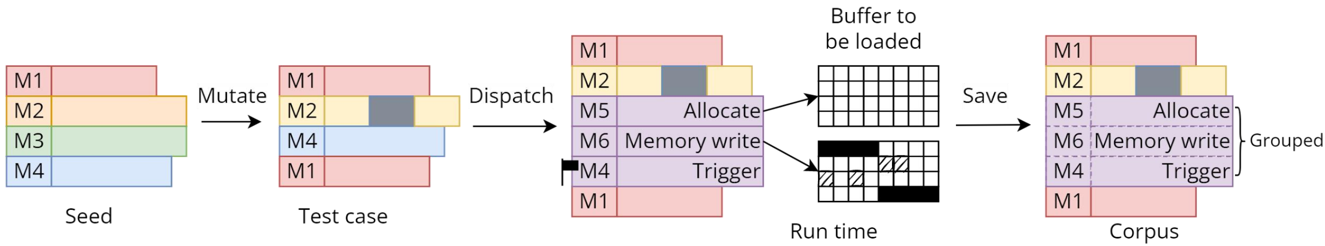
Fuzzing virtual device must consider

- Intra-message and inter-message dependencies

ViDeZZo addresses them with

- Intra-message annotation and inter-message mutators

ViDeZZo found 28 new bugs in both QEMU and VirtualBox



EPFL

hexhive

Backup Slides

Encoded intra-message dependency: flag/tag-pointer dependency

```
1  typedef struct {
2    uint32_t command; uint32_t array_addr; } tx_t;
3  void action_command(physaddr addr) {
4    tx_t tx;
5    MacAddr macaddr;
6    TxConfig config;
7    dma_read(/*addr=*/addr, /*dst=*/&tx);
8    switch (tx.command & COMMAND/*=7*/) {
9    case CmdIASetup/*=1*/:
10     dma_read(tx.array_addr, &macaddr); break;
11   case CmdConfigure/*=2*/:
12     dma_read(tx.array_addr, &config); break;
```

```
1  vd0 = Model('tx', 0)
2  vd0.add_head(['tx_t'])
3  vd0.add_struct('tx_t', {
4    'command#0x4': 'FLAG',
5    'array_addr#0x4': 'POINTER'})
6  vd0.add_flag('tx_t.command', {0: 3})
7  vd0.add_point_to('tx_t.array_addr',
8    [None, 'macaddr', 'config', None, None, None,
9  //if 0       1           2        3     4     5
10   None, None], condition=['tx_t.command.0'])
11 //  6     7               == tx_t.command.0
```

Encoded intra-message dependency: head-tail-pointers dependency

```
1  typedef struct {
2    uint32_t head; uint32_t tail;
3  } ed_t;
4  typedef struct {
5    uint32_t next;
6  } td_t;
7  void handle_end_descriptor(physaddr head)
8    ed_t ed;
9    dma_read(/*addr=*/head, /*dst=*/&ed)
10   while ((ed.head & 0xffffff00) != ed.tail) {
11     td_t td;
12     phsyaddr addr = ed->head & 0xffffff00;
13     if (ed.head & 0x1) /* be invalid and return */
14     dma_read(/*addr=*/addr, /*dst=*/&td);
15     ed->head |= td.next & 0xffffff00;
16 //---------
17 vd1 = Model('ed', 1)
18 vd1.add_head(['ed_t'])
19 vd1.add_struct('ed_t', {
20   'head#0x4': POINTER|FLAG,
21   'tail#0x4': 'POINTER'})
22 vd1.add_flag('ed_t.head', {0: 1@0})
23 vd1.add_struct('td_t', {'next#0x4': 'POINTER'})
24 vd1.add_linked_list(
25   'ed_t.head', 'ed_t.tail',
26     ['td_t'], ['next'], alignment=8)
27 vd1.add_head(['ed_t'])
```

Encoded intra-message dependency: len-buffer dependency

```
1  typedef {
2    uint64_t addr1; uint32_t len;
3  } bpl_t;
4  void handle_hda(physaddr addr0)
5    bpl_t bpl;
6    dma_read(/*addr=*/addr0, /*dst=*/&bpl);
7    int n_copied = custom_memcpy(
8      /*src=*/bpl.addr1, /*dst=*/buf);
9    if (bpl.len == n_copied) {
10     // do something
11 //----------
12 vd2 = Model('bpl', 2)
13 vd2.add_head('bpl_t')
14 vd2.add_struct('bpl_t', {
15   'addr1#0x8': 'POINTER', 'len#0x4': 'CONSTANT'})
16 vd2.add_strcut('bpl_buf', {
17   'buf#0x1000': 'RANDOM'})
18 vd2.add_point_to('bpl_t.addr1', ['bpl_buf'])
19 vd2.add_constant('bpl_buf.len', [0x1000])
```

Encoded intra-message dependency: dependency in MMIO accesses

```
1  static void xhci_doorbell_write(
2      void *ptr, hwaddr reg,
3      uint64_t val, unsigned size) {
4    reg >>= 2;
5    if (reg == 0) {
6      if (val == 0) {
7        xhci_process_commands(xhci);
8    } else {
9      epid = val & 0xff;
10     streamid = (val >> 16) & 0xffff;
11     xhci_kick_ep(xhci, reg, epid, streamid);
12 }
13 //----------
14 vd3 = Model('xhci_doorbell_write_0', 3)
15 vd3.add_head('mmio_write')
16 vd3.add_struct('mmio_write', {
17   'addr#0x8': 'CONSTANT', 'len#0x4': 'CONSTANT',
18   'valu#0x8': 'CONSTNAT'})
19 vd3.add_constant('mmio_write.addr', 0x0)
20 vd3.add_constant('mmio_write.len', 0x4)
21 vd3.add_constant('mmio_write.valu', 0x0)
22
23 vd4 = Model('xhci_doorbell_write_!0', 4)
24 vd4.add_head('mmio_write')
25 vd4.add_struct('mmio_write', {
26   'addr#0x8': 'CONSTANT', 'len#0x4': 'CONSTANT',
27   'valu#0x8': 'FLAG'})
28 vd4.add_constant('mmio_write.addr', [
29   i for i in range(4, 0x20)])
30 vd3.add_constant('mmio_write.len', 0x4)
31 vd4.add_flag('mmio_write.valu', {
32   0: 8, 8: 16, 16: 32, 32: 64@0})
```