ViDeZZo: Dependency-aware Virtual Device Fuzzing

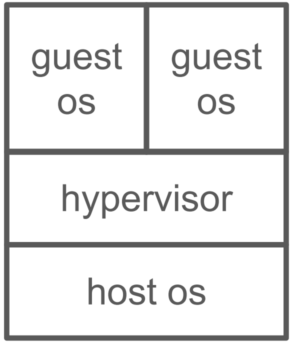
Qiang Liu (Zhejiang University; EPFL) Flavio Toffalini (EPFL)
Yajin Zhou (Zhejiang University) Mathias Payer (EPFL)

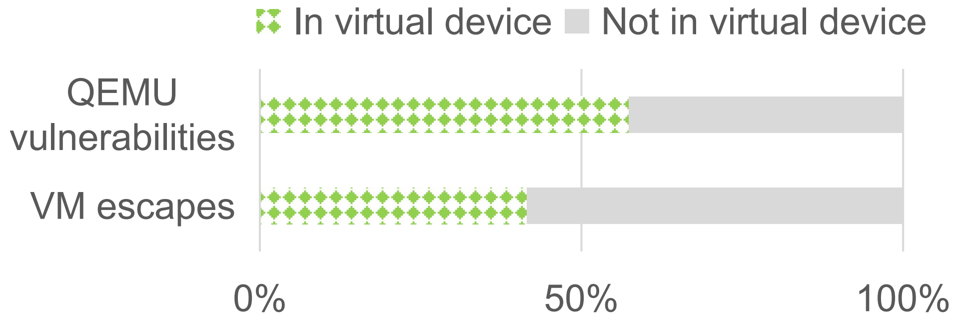
Virtual Device Security Matters!

Virtual device is software that emulates hardware

Hypervisor must isolate the host from the guest

Virtual device vulnerabilities are the biggest single type



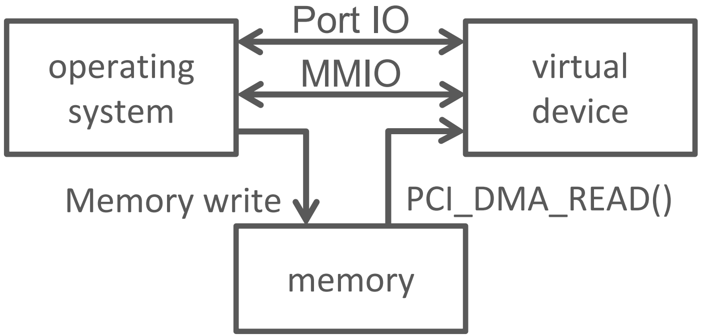


How to fuzz virtual devices in an efficient and scalable way?

Key Points

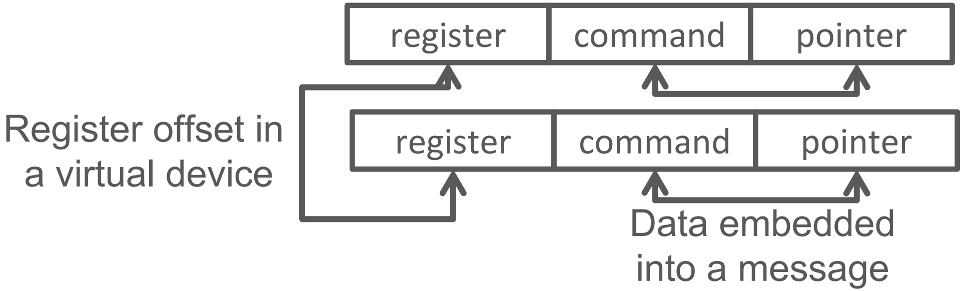
Virtual Device Messages

- Port IO read/write
- MMIO read/write
- DMA read/write



Message Sequence

- E.g., two MMIO messages



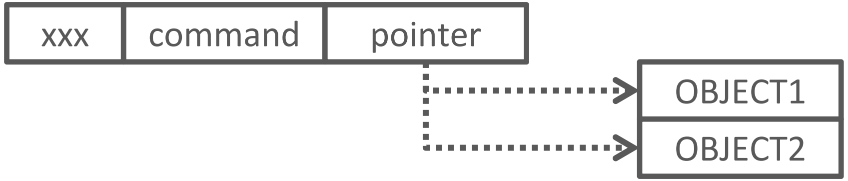
Key challenges: intra-message and inter-message dependency

Key Challenge 1: Intra-Message Dependency

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

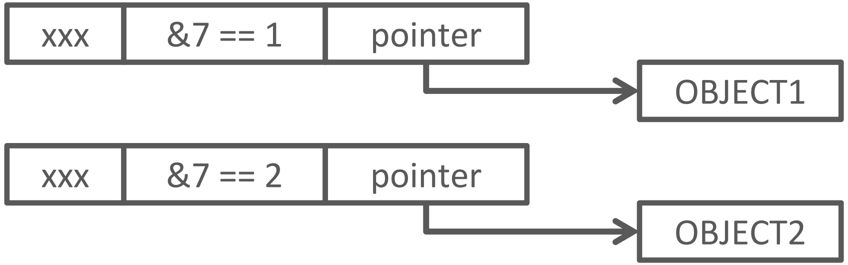
Message 1

- *Pointer* points to something



Field *pointer* should point to which object?

- It depends on the value of *command*



Key Challenge 2: Inter-Message Dependency

A message may depend on a previously issued message

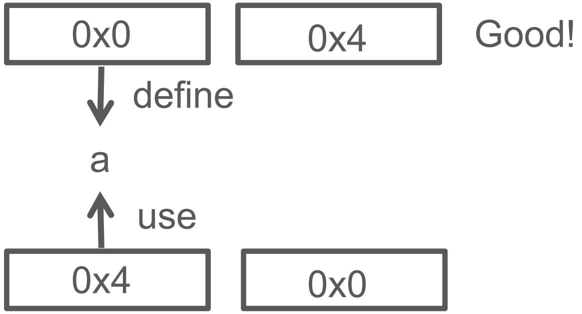
Message 2 and message 3

- Operate two related registers 0x0 and 0x4

0x0	xxx	xxx
0x4	xxx	xxx

Which message should be issued first?

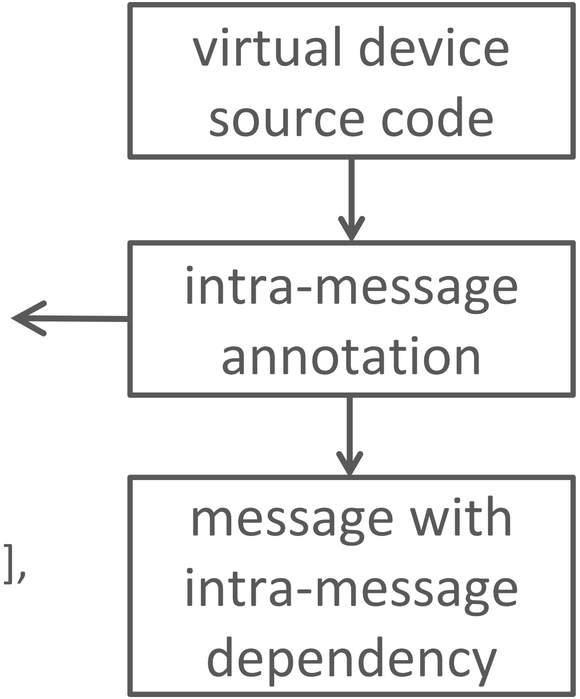
- Message 3 {0x4} depends on message 2 {0x0}



Solution 1: Intra-Message Annotation

```
vd0=Model('tx', 0)
vd0.add_struct('tx_t', {
  'command#0x4': FLAG,
  'address#0x4': POINTER})
vd0.add_flag(
  'tx_t.command', {0: 3})

vd0.add_point_to(
  'tx_t.address', [  ...,
  /*1*/object1,  /*2*/object2, ...],
  condition=['tx_t.command'])
```



Semi-automatically extract intra-message annotation from source code

Solution 2: Inter-Message Mutation

Message Level

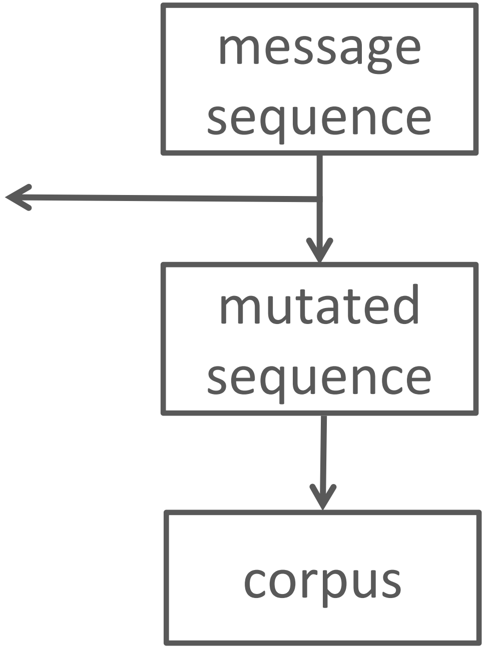
- ChangeValue: mutate the value of a message

Sequence Level

- ShuffleMessage: shuffle a sequence message

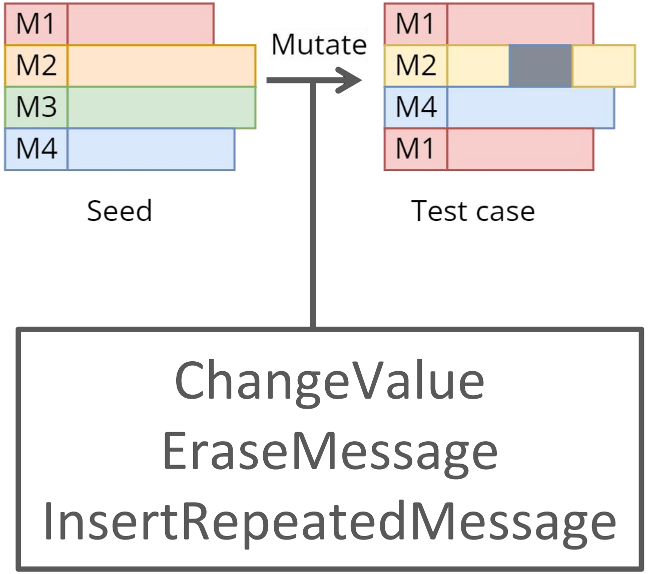
Group Level

- GroupMessage: group message for future re-use

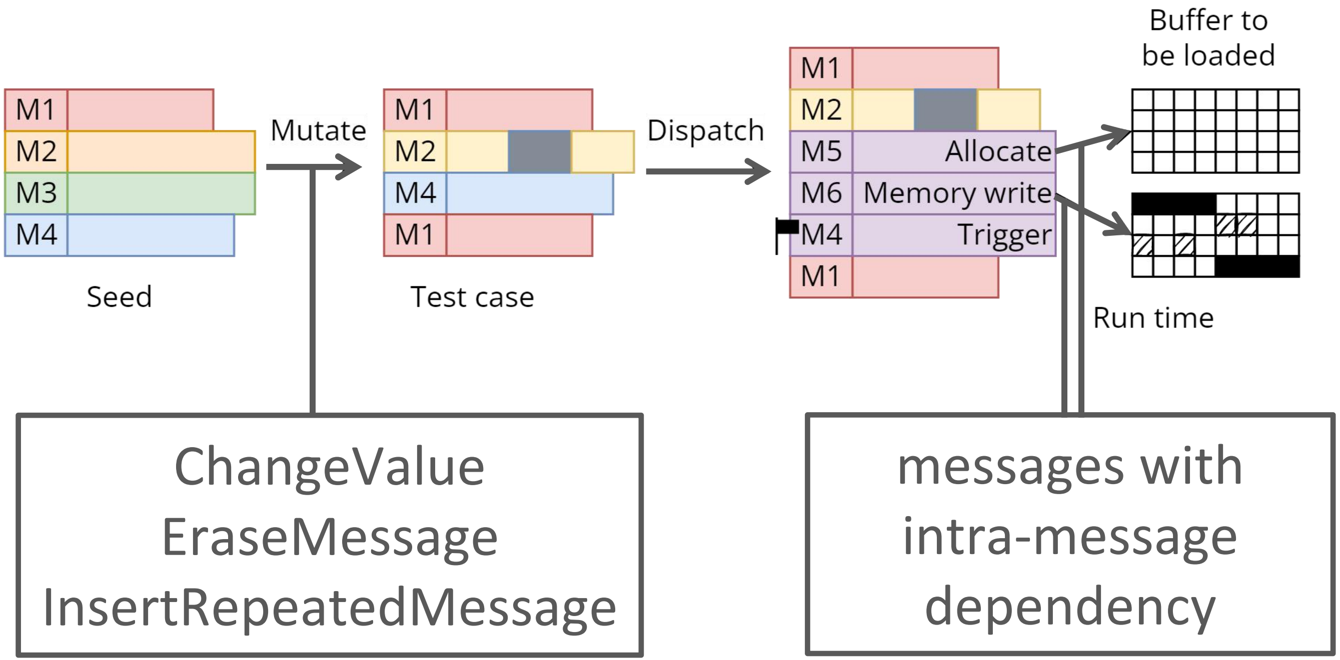


Automatically learn the dependency with new mutators during fuzzing

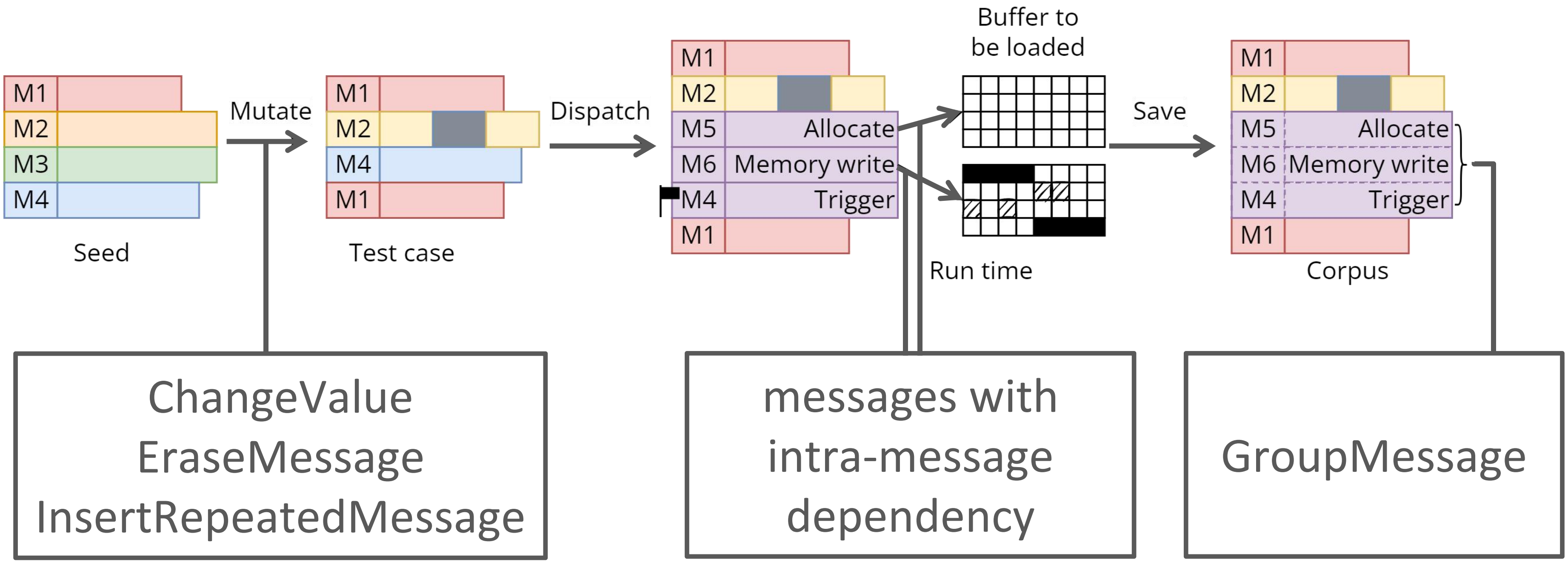
Fuzzing Workflow



Fuzzing Workflow



Fuzzing Workflow



Expressive Grammar Limits Manual Effort

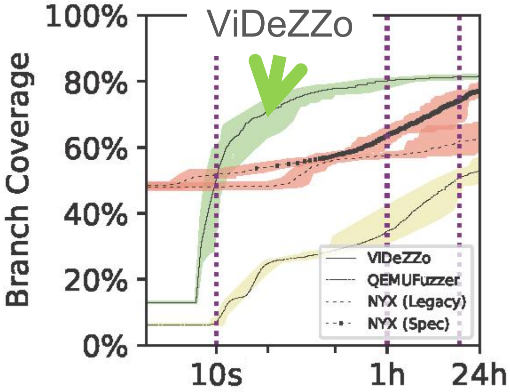
ViDeZZo semi-automatically models 18 QEMU virtual devices

- While Nyx models only 1 QEMU virtual device manually

Why do we need manual effort?

- Unnamed types (four cases)
- Disjointed control flow (four virtual devices)
- Context-aware dependencies (five virtual devices)

Coverage and Bugs



ViDeZZo scales to 28 virtual devices

- Covering 5 device categories, 4 archs, and 2 vmms
- Achieving competitive final coverage results faster

ViDeZZo discovers 24 existing bugs and 28 new bugs

- In both QEMU and VirtualBox
- In both virtio/non-virtio virtual devices
- Covering not only checks but also spatial/temporal memory corruption

We have seven patches accepted

ViDeZZo: Dependency-aware Virtual Device Fuzzing

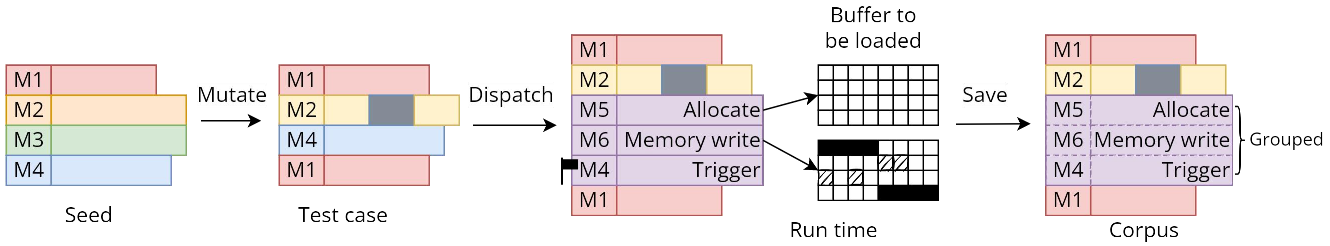
Fuzzing virtual device must consider

- Intra-message and inter-message dependencies

ViDeZZo addresses them with

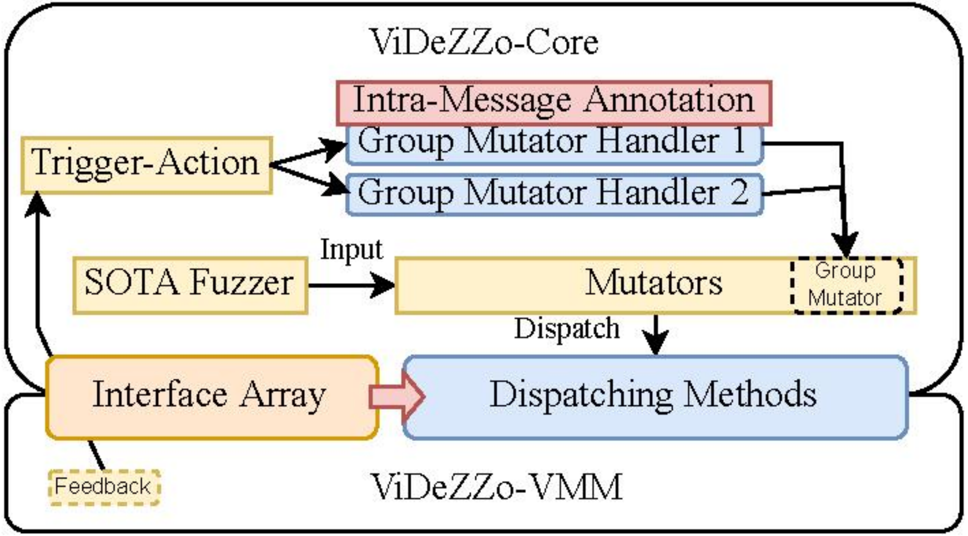
- Intra-message annotation and inter-message mutators

ViDeZZo found 28 new bugs in both QEMU and VirtualBox



Backup slides

System Design



Only One CVE?

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)

Summary of Manual Effort

Step (where in the text)	Manual effort	Estimated average time
Add a new VMM (Section 5.3)	Register a virtual device by searching its architecture, the launch command line, and the signature of PIO/MMIO regions.	10 minutes per virtual device
	Initialize a VMM and identify the testing interfaces by following the main() in an existing VMM frontend.	A week per VMM (up to two weeks for debugging)
	Decide and implement the dispatching methods by looking for guest memory access functions.	An hour per VMM
Finish the rest of the annotation extraction after scanning the source code of a virtual device with our static analysis engine (Section 6.1)	Extract the definition of unnamed types by looking at the source code.	Two minutes per case
	Match two taint analysis results touching the same variable due to disjointed control flow by reading the source code.	15 minutes per case
	Extract the head-tail pointer context by reading the source code.	10 minutes per case
	Extract the flag/tag pointer context by reading the source code.	20 minutes per case
	Extract the length and buffer context by reading the source code.	Five minutes per case
Add a new group mutator (end of Section 4.3)	Obtain the insight about what group mutator is necessary by fuzzing virtual devices.	N/A
	Decide the feedback and develop the handler with the help of our action-trigger protocol.	Hours per case (up to two days for debugging)

Summary of Scalability

Flexible System Design

- ViDeZZo-Core and ViDeZZo-VMM

Lightweight Annotation

Reuse the Same Annotation

- Same virtual devices of different hypervisors

Device	VDF	HyperCube	Nyx-Legacy	V-SHUTTLE	QEMUFuzzer	ViDeZZo
QEMU-x86 Audio						
AC97	53.0%	100%	94.04%	—	**95.93%**	95.90%
CS4231a	56.0%	74.76%	75.36%	85.80%	**94.06%**	92.61%
ES1370	72.7%	91.38%	89.69%	91.91%	88.40%	**91.36%**
Intel-HDA	58.6%	79.17%	62.61%	78.30%	**65.87%**	64.78%
SB16	81.0%	83.80%	83.12%	81.52%	84.15%	**87.54%**
QEMU-x86 Storage						
AHCI	—	—	—	61.60%	49.89%	**62.06%**
FDC	70.5%	84.51%	**70.06%**	—	69.23%	69.72%
Megasas	—	—	—	58.50%	58.67%	**76.74%**
SDHCI	90.5%	81.15%	**73.58%**	—	71.34%	68.52%
VirtIO-BLK	—	—	—	—	30.55%	**55.39%**
QEMU-x86 Network						
E1000	81.6%	66.08%	53.36%	74.50%	35.32%	**82.27%**
E1000E (1/2)[1]	—	—	—	—	**63.12%**	60.94%
E1000E (2/2)[1]	—	—	—	—	35.48%	**40.84%**
EEPro100	75.4%	83.32%	82.12%	—	82.13%	**90.46%**
NE2000	71.7%	71.89%	74.35%	71.90%	75.09%	**94.00%**
PCNET	36.1%	78.81%	78.87%	88.90%	**93.27%**	92.10%
RTL8139	63.0%	74.68%	**83.33%**	80.82%	83.06%	77.46%
QEMU-x86 Display						
ATI-VGA (1/2)[2]	—	—	—	79.40%	—	80.69%
ATI-VGA (2/2)[2]	—	—	—	—	—	85.67%
CIRRUS-VGA	—	—	—	—	88.65%	**89.68%**
QEMU-x86 USB						
EHCI	—	—	—	31.19%	71.84%	**71.96%**
OHCI	—	—	—	36.62%	77.33%	**83.99%**
UHCI	—	—	—	22.27%	55.90%	**72.00%**
XHCI	—	64.40%	63.24%	—	52.92%	**81.63%**
			Nyx-Spec			
XHCI			77.12%			
QEMU-x86_64						
VirtIO-BLK	—	—	—	—	—	55.39%
QEMU-AArch32						
PL041 (Audio)	—	—	—	—	—	83.91%
SMC91C111 (Net)	—	—	—	—	92.14%	92.98%
TC6393XB (Display)	—	—	—	—	—	76.38%
QEMU-AArch64						
XLNX-ZYNQMP-CAN	—	—	—	—	—	70.42%
XLNX-DP (Display)	—	—	—	—	—	90.42%
VirtualBox x86_64						
SB16	—	—	—	—	—	61.33%
FDC	—	—	—	—	—	39.32%
PCNET	—	—	—	—	—	48.35%
OHCI	—	—	—	—	—	36.13%