MundoFuzz: Hypervisor Fuzzing with Statistical Coverage Testing and Grammar Inference

Cheolwoo Myung†, Gwangmu Lee‡, and Byoungyoung Lee†
Seoul National University†, EPFL‡
Hypervisor: Manager of Virtual Machine

- Allow **remote users** to run guest VMs
Hypervisor can be attacked by **Malicious VM**

- One of guest VMs can be **malicious**
Fuzzing: Feed **Random Inputs** to Hypervisor

Combination of Low-level IO operation

- PIO
- MMIO
- DMA
- etc.
Motivation: Too many devices, too many formats

- Hypervisor controls **many virtual devices**
  - Each device has its **own input formats**

![Virtual Devices Diagram]

**virtual devices**

- SCSI
- NIC
- USB
- NVMe
- Display
- Timer
- Sound
- SATA
- APIC

Input

Hypervisor
Limitations of Current Hypervisor Fuzzing

#1. Generating **random inputs** per device

**Limitation** ⇒ Cannot explore deep states of the devices

#2. Relying on **manual input grammars** per device

**Limitation** ⇒ Require unacceptable manual work to specify grammar rules

Let’s fuzz hypervisor with **grammar-awareness** using **automatic grammar inference**!
Overview of MundoFuzz

- Augment hypervisor fuzzing capability with automatic grammar inference

- **Challenges** in inferring hypervisor grammars
  - #1. Hypervisor grammars have *hidden input semantics* per device
  - #2. Hardware features of hypervisor introduce *coverage noises*

- **Our approach**
  - Statistical and differential learning with coverage
Challenge 1: Hidden Input Semantics

- Too difficult to infer **hidden input semantics** behind the hypervisor input

Example: SCSI command input
Challenge 1: Hidden Input Semantics

- Too difficult to infer hidden input semantics behind the hypervisor input
  - IO address semantics: correct semantic command should be given

Example: SCSI command input
Challenge 1: Hidden Input Semantics

- Too difficult to infer **hidden input semantics** behind the hypervisor input
  - **IO address semantics**: correct semantic command should be given

Example: SCSI command input

<table>
<thead>
<tr>
<th>Control Type (bar+4)</th>
<th>Data Type (bar+0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>: invoke the desired function</td>
<td>: transfers the data parameter</td>
</tr>
</tbody>
</table>

Hypervisor Operation

- Find Sector 0
- Write Data

--MMIO[bar+0] ← 0x0
--MMIO[bar+4] ← 0x4
--MMIO[bar+0] ← 0xc0af
--MMIO[bar+4] ← 0x8
Challenge 1: Hidden Input Semantics

- Too difficult to infer hidden input semantics behind the hypervisor input
  - IO address semantics: correct semantic command should be given

Example: SCSI command input

Control Type (bar+4)
: invoke the desired function

Data Type (bar+0)
: transfers the data parameter

| MMIO[bar+0] ← 0x0 |
| MMIO[bar+4] ← 0x4 |
| MMIO[bar+0] ← 0xc0ffee |
| MMIO[bar+4] ← 0x8 |

Hypervisor Operation

Invoke the “Find Sector” func. (0x4) with the parameter (0x0)

Find Sector 0

Write Data
Challenge 1: Hidden Input Semantics

- Too difficult to infer **hidden input semantics** behind the hypervisor input
  - **IO address semantics**: correct semantic command should be given

Invoke the "Find Sector" func. (0x4) with the parameter (0x0)

Input

<table>
<thead>
<tr>
<th>Control Type (bar+4)</th>
<th>Data Type (bar+0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>: invoke the desired function</td>
<td>: transfers the data parameter</td>
</tr>
</tbody>
</table>

Example: SCSI command input

<table>
<thead>
<tr>
<th>MMIO[bar+0]</th>
<th>MMIO[bar+4]</th>
</tr>
</thead>
<tbody>
<tr>
<td>← 0x0</td>
<td>← 0x4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MMIO[bar+0]</th>
<th>MMIO[bar+4]</th>
</tr>
</thead>
<tbody>
<tr>
<td>← 0xc0fe</td>
<td>← 0x8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hypervisor Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find Sector 0</td>
</tr>
<tr>
<td>Write Data</td>
</tr>
</tbody>
</table>

- **Example:**

Invoke the “Write Data” func. (0x8) with the parameter (0x8)
Challenge 1: Hidden Input Semantics

- Too difficult to infer **hidden input semantics** behind the hypervisor input
  - **IO address semantics**: correct semantic command should be given
  - **IO order semantics**: correct semantic order should be given

**Example: SCSI command input**

- MMIO[bar+0] ← 0x0
- MMIO[bar+4] ← 0x4
- MMIO[bar+0] ← 0xcafe
- MMIO[bar+4] ← 0x8

**Hypervisor Operation**

- Find Sector 0
- Write Data
Challenge 1: Hidden Input Semantics

- Too difficult to infer **hidden input semantics** behind the hypervisor input
  - **IO address semantics**: correct semantic command should be given
  - **IO order semantics**: correct semantic order should be given

Example: SCSI command input

- Input
  - SCSI Command

- Hypervisor Operation
  - Find Sector 0
  - Write Data

```
MMIO[bar+0] ← 0x0
MMIO[bar+4] ← 0x4
MMIO[bar+0] ← 0xcafe
MMIO[bar+4] ← 0x8
```
Challenge 1: Hidden Input Semantics

- Too difficult to infer **hidden input semantics** behind the hypervisor input
  - **IO address semantics:** correct semantic command should be given
  - **IO order semantics:** "Find Sector" should be performed before "Write Data"

Example: SCSI command input

```
Input
SCSI Command

MMIO[bar+0] ← 0x0
MMIO[bar+4] ← 0x4
MMIO[bar+0] ← 0xcafe
MMIO[bar+4] ← 0x8

Hypervisor Operation
Find Sector 0
Write Data
```
Solution 1: Differential Learning on Input Semantics

#1. IO address semantics

- Different IO address types react to IO address values differently
Solution 1: Differential Learning on Input Semantics

#1. IO address semantics

- Different IO address types react to IO address values differently
  - control type ⇒ exhibits a different coverage

```
Correct input
MMIO[bar+0] ← 0x0
MMIO[bar+4] ← 0x4

Incorrect input
MMIO[bar+0] ← 0x0
MMIO[bar+4] ← 0xfb
```
Solution 1: Differential Learning on Input Semantics

#1. IO address semantics

- Different IO address types react to IO addresses differently
  - control type ⇒ exhibits a different coverage

Correct input:
- MMIO[bar+0] ← 0x0
- MMIO[bar+4] ← 0x4

Incorrect input:
- MMIO[bar+0] ← 0x0
- MMIO[bar+4] ← 0xfb

Control Type!

Different!!
Solution 1: Differential Learning on Input Semantics

#1. IO address semantics

- Different IO address types react to IO address values differently
  - control type ⇒ exhibits a different coverage
  - data type ⇒ exhibits a same coverage

Correct input
- \text{MMIO}[\text{bar+0}] \leftarrow 0x0
- \text{MMIO}[\text{bar+4}] \leftarrow 0x4

Incorrect input
- \text{MMIO}[\text{bar+0}] \leftarrow 0xff
- \text{MMIO}[\text{bar+4}] \leftarrow 0x4

Coverage
- Same!!

Diagram:
- Correct input: \text{MMIO}[\text{bar+0}] \leftarrow 0x0, \text{MMIO}[\text{bar+4}] \leftarrow 0x4
- Incorrect input: \text{MMIO}[\text{bar+0}] \leftarrow 0xff, \text{MMIO}[\text{bar+4}] \leftarrow 0x4
- Hypervisor output: Same!!

Legend:
- Red: Incorrect input
- Green: Correct input
Solution 1: Differential Learning on Input Semantics

#1. IO address semantics

- Different IO address types react to IO address values differently:
  - control type ⇒ exhibits a different coverage
  - data type ⇒ exhibits a same coverage

Correct input:
- $\text{MMIO[bar+0]} \leftarrow 0x0$
- $\text{MMIO[bar+4]} \leftarrow 0x4$

Incorrect input:
- $\text{MMIO[bar+0]} \leftarrow 0xff$
- $\text{MMIO[bar+4]} \leftarrow 0x4$

Coverage:

Find Sector 0

Find Sector 255
Solution 1: Differential Learning on Input Semantics

#2. IO order semantics

- IO operations wouldn’t work correctly without prerequisite IO operations
  - absence of IO operations $\Rightarrow$ may distort some following coverage
Solution 1: Differential Learning on Input Semantics

#2. IO order semantics

- IO operations wouldn’t work correctly without prerequisite IO operations
  - absence of IO operations ⇒ may distort \textit{some following coverage}

\begin{itemize}
  \item Correct input
    \begin{align*}
    \text{MMIO}_{\text{bar}+0} & \leftarrow 0x0 \\
    \text{MMIO}_{\text{bar}+4} & \leftarrow 0x4 \\
    \text{MMIO}_{\text{bar}+0} & \leftarrow 0xcafe \\
    \text{MMIO}_{\text{bar}+4} & \leftarrow 0x8 \\
    \end{align*}

  \item Incorrect input
    \begin{align*}
    \text{MMIO}_{\text{bar}+0} & \leftarrow 0x0 \\
    \text{MMIO}_{\text{bar}+4} & \leftarrow 0x4 \\
    \text{MMIO}_{\text{bar}+0} & \leftarrow 0xcafe \\
    \text{MMIO}_{\text{bar}+4} & \leftarrow 0x8 \\
    \end{align*}
\end{itemize}
Solution 1: Differential Learning on Input Semantics

#2. IO order semantics

- IO operations wouldn’t work correctly without prerequisite IO operations
  - absence of IO operations ⇒ may distort some following coverage

Correct input:

- $\text{MMIO}[\text{bar}+0] \leftarrow 0x0$
- $\text{MMIO}[\text{bar}+4] \leftarrow 0x4$
- $\text{MMIO}[\text{bar}+0] \leftarrow 0\text{cafe}$
- $\text{MMIO}[\text{bar}+4] \leftarrow 0x8$

Incorrect input:

- $\text{MMIO}[\text{bar}+0] \leftarrow 0x0$
- $\text{MMIO}[\text{bar}+4] \leftarrow 0x4$
- $\text{MMIO}[\text{bar}+0] \leftarrow 0\text{cafe}$
- $\text{MMIO}[\text{bar}+4] \leftarrow 0x8$

Coverage:
Solution 1: Differential Learning on Input Semantics

#2. IO order semantics

- IO operations wouldn’t work correctly without prerequisite IO operations
  - absence of IO operations ⇒ may distort some following coverage

Correct input:

<table>
<thead>
<tr>
<th>Input</th>
<th>MMIO[bar+0] ← 0x0</th>
<th>MMIO[bar+4] ← 0x4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MMIO[bar+0] ← 0xcafe</td>
<td>MMIO[bar+4] ← 0x8</td>
</tr>
</tbody>
</table>

Incorrect input:

<table>
<thead>
<tr>
<th>Input</th>
<th>MMIO[bar+0] ← 0x0</th>
<th>MMIO[bar+4] ← 0x4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MMIO[bar+0] ← 0xcafe</td>
<td>MMIO[bar+4] ← 0x8</td>
</tr>
<tr>
<td>Skip</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Dependent!

Input:

| MMIO[bar+0] ← 0x0 | MMIO[bar+4] ← 0x4 |
| MMIO[bar+0] ← 0xCAFE | MMIO[bar+4] ← 0x8 |

Coverage:

Different!!

Match!
Challenge 2: Coverage Noises

- The measured input coverage includes **unwanted coverage**
  - due to the asynchronous event handling (e.g., timer, interrupt event)
  - asynchronous event introduces **non-deterministic (noise) coverage**
Challenge 2: Coverage Noises

- The measured input coverage includes **unwanted coverage**
  - due to the asynchronous event handling (e.g., timer, interrupt event)
  - asynchronous event introduces **non-deterministic (noise) coverage**

**Example:** SCSI command input
The measured input coverage includes **unwanted coverage**
- due to the asynchronous event handling (e.g., timer, interrupt event)
- asynchronous event introduces **non-deterministic (noise) coverage**

**Example: SCSI command input**

**Hypervisor Operation**

- **Interrupt event**
  - **Find Sector**
  - **Write Data**

**Expected coverage**: 6

**Measured coverage**: 5

**# of execution**: 1

Cov. #1
Challenge 2: Coverage Noises

- The measured input coverage includes **unwanted coverage**
  - due to the asynchronous event handling (e.g., timer, interrupt event)
  - asynchronous event introduces **non-deterministic (noise) coverage**

Example: SCSI command input

Hypervisor Operation

Input

SCSI Command

Find Sector

Write Data

Timer event

Expected coverage:

# of execution: 2

Measured coverage:

Cov. #1

Cov. #2
Challenge 2: Coverage Noises

- The measured input coverage includes **unwanted coverage**
  - due to the asynchronous event handling (e.g., timer, interrupt event)
  - asynchronous event introduces **non-deterministic (noise) coverage**

---

Example: SCSI command input

<table>
<thead>
<tr>
<th>Input</th>
<th>Hypervisor Operation</th>
<th>expected coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCSI Command</td>
<td>Find Sector</td>
<td></td>
</tr>
<tr>
<td></td>
<td>APIC event</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Write Data</td>
<td></td>
</tr>
</tbody>
</table>

# of execution: \( N \)

measured coverage

<table>
<thead>
<tr>
<th>Cov. #1</th>
<th>Cov. #2</th>
<th>...</th>
<th>Cov. #N</th>
</tr>
</thead>
</table>

: target
: noise
Solution 2: Statistical Differential Coverage Measurement

 measured coverage

coverage #1

coverage #2

::

coverage #N

: target
: noise
Solution 2: Statistical Differential Coverage Measurement

- **Target coverage** (■)
  - is always captured for all execution
Solution 2: Statistical Differential Coverage Measurement

- **Target coverage** (■)
  - is always captured for all execution

- **Noise coverage** (□)
  - is captured differently for each execution
Solution 2: Statistical Differential Coverage Measurement

- Remove noise coverage by intersecting all measured coverages

![Diagram showing the intersection of measured coverages to remove noise coverage.](image)
Solution 2: Statistical Differential Coverage Measurement

- Remove noise coverage by intersecting all measured coverages
  - the result only contains target coverage
Architecture of MundoFuzz

- Hypervisor
- MundoFuzzOS
- Correct
- Incorrect
- Statistical noise filter
- Cov. w/o noise
- Cov. w/ noise
- IO addr.
- IO order
- Analysis
- Grammar
- MundoFuzz-Fuzzer
- MundoFuzzOS
- Hypervisor
What MundoFuzz Found?

- MundoFuzz found new 40 bugs in QEMU and Bhyve
  - 23 bugs in QEMU
  - 17 bugs in Bhyve
  - 9 of these were acknowledged as CVEs

<table>
<thead>
<tr>
<th>Hypervisor</th>
<th>Bug Types</th>
<th>Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>QEMU</td>
<td>Use-after-free</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Heap Overflow</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Segmentation Fault</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Infinite Loop</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Stack Overflow</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Assertion</td>
<td>11</td>
</tr>
<tr>
<td>Bhyve</td>
<td>Segmentation Fault</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Floating Point Exception</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Assertion</td>
<td>12</td>
</tr>
</tbody>
</table>
Our result

- Overall coverage: MundoFuzz outperforms state-of-art hypervisor fuzzer
  - HyperCube: +4.91%
  - Nyx: +6.60%
- MundoFuzz shows higher coverage than Nyx+ (with manual grammar rule)
  - for USB-XHCI device (48 hours)
Conclusion

- Proposed MundoFuzz, a hypervisor fuzzing technique
  - statistically removes noise coverage in raw coverage
  - automatically learns the grammar using two hidden semantics

- MundoFuzz discovered 40 new bugs (including 9 CVEs)

- MundoFuzz presented better coverage, compared to state of the arts.
Thank you!

Q & A

Contact Cheolwoo Myung
Ph.D. Student at Seoul National University (SNU)
cwmyung@snu.ac.kr