SEGFUZZ: Segmentizing Thread Interleaving to Discover Kernel Concurrency Bugs through Fuzzing

Dae R. Jeong¹, Byoungyoung Lee², Insik Shin¹, Youngjin Kwon¹

¹Korea Advanced Institute of Science & Technology
²Seoul National University
Kernel concurrency bugs

- Kernel concurrency bugs manifest depending on thread interleavings
Kernel concurrency bugs

- Kernel concurrency bugs manifest depending on thread interleavings

**Interleaving 1**

```c
Syscall A

flag = 1;

Syscall B

if (flag)
    init_ptr(ptr);

if (flag)
    access_ptr(ptr);
```
Kernel concurrency bugs

- Kernel concurrency bugs manifest depending on thread interleavings

**Interleaving 1**

```c
Syscall A
flag = 1;
if (flag)
    init_ptr(ptr);
if (flag)
    access_ptr(ptr);
```

**Interleaving 2**

```c
Syscall A
Syscall B
flag = 1;
if (flag)
    init_ptr(ptr);
if (flag)
    access_ptr(ptr);
Uninitialized access!
```
Fuzzing explores the search space of the program by running random inputs

- Conventionally focusing on exploring *execution paths*
  - Symbolic/concolic execution, static analysis, ...
Fuzzing explores the search space of the program by running random inputs
- Conventionally focusing on exploring *execution paths*
  - Symbolic/concolic execution, static analysis, ...

Recent approaches to identify concurrency bugs
- Exploring *execution path & thread interleavings*
  - Razzer [S&P’19], Krace [S&P’20], Snowboard [SOSP’21], Conzzer [NDSS’22], ...
- *Controlling thread interleavings* by overriding the kernel scheduler
Coverage-guided fuzzing

- **Coverage metric**
  - Expressing the search space of the program
  - Guiding the generation of new test cases
Coverage-guided fuzzing

- **Coverage metric**
  - Expressing the search space of the program
  - Guiding the generation of new test cases

- **Code coverage**
  - Expressing the search space of *execution paths*
  - Ex) Branch coverage
Coverage-guided fuzzing

- **Code coverage**
  - Limited in expressing the search space of thread interleavings

Interleaving 1

<table>
<thead>
<tr>
<th>Thread 1</th>
<th>Thread 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A = 1;</td>
<td>A = 2;</td>
</tr>
<tr>
<td>if (A != 0)</td>
<td></td>
</tr>
<tr>
<td>print(A);</td>
<td></td>
</tr>
</tbody>
</table>

Interleaving 2

<table>
<thead>
<tr>
<th>Thread 1</th>
<th>Thread 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A = 1;</td>
<td>A = 2;</td>
</tr>
<tr>
<td>if (A != 0)</td>
<td></td>
</tr>
<tr>
<td>print(A);</td>
<td></td>
</tr>
</tbody>
</table>

The same branch coverage but different outcomes
Coverage-guided fuzzing

**Coverage metric**
- Expressing the search space of the program
- Guiding the generation of new test cases

**Code coverage**
- Expressing the search space for *execution paths*
- Ex) Branch coverage

**Interleaving coverage**
- Expressing the search space for *thread interleavings*
- *Not well-studied area*
Coverage-guided fuzzing

- **Coverage metric**
  - Expressing the search space of the program
  - Guiding the generation of new test cases

**We want to design and utilize interleaving coverage**

- **Interleaving coverage**
  - Expressing the search space for *thread interleavings*
  - *Not well-studied area*
Coverage metric for thread interleavings

- **Challenge**
  - A large search space of thread interleavings
Coverage metric for thread interleavings

- **Challenge**
  - A large search space of thread interleavings

100 inst. for each syscall
Coverage metric for thread interleavings

- **Challenge**
  - A large search space of thread interleavings

There are a huge number of interleavings (e.g., more than $10^{58}$)

100 inst. for each syscall
Coverage metric for thread interleavings

- **Challenge**
  - A large search space of thread interleavings

There are a huge number of interleavings (e.g., more than $10^{58}$)

Only a small number of interleavings cause a concurrency bug.

100 inst. for each syscall
Coverage metric for thread interleavings

- **Challenge**
  - A large search space of thread interleavings

- Our interleaving coverage should
  1) reduce the search space
  2) capture “interesting” interleavings
Characteristic of concurrency bugs

- Observation from a previous study [1]
  - Most of concurrency bugs (97 out of 105) manifest depending on the execution order of at most four memory accesses

Characteristic of concurrency bugs

- **Observation from a previous study [1]**
  - Most of concurrency bugs (97 out of 105) manifest depending on the execution order of *at most four memory accesses*

```c
if (flag)
    init_ptr(ptr);
flag = 1;
if (flag)
    access_ptr(ptr);
```

Uninitialized access!

---

Characteristic of concurrency bugs

- **Observation from a previous study [1]**
  - Most of concurrency bugs (97 out of 105) manifest depending on the execution order of *at most four memory accesses*

The uninitialized access bug manifests depending *only on three instructions*

Characteristic of concurrency bugs

- **Observation from a previous study [1]**
  - Most of concurrency bugs (97 out of 105) manifest depending on the execution order of *at most four memory accesses*

- **Our strategy: Segmentizing thread interleaving**
  - Decomposing thread interleaving into small interleaving segments that consists of at most four memory accesses

---

Key idea: decomposing thread interleaving

100 inst. for each syscall
Key idea: decomposing thread interleaving

100 inst. for each syscall
Key idea: decomposing thread interleaving

Interleaving segment
Key idea: decomposing thread interleaving

**Benefits**
- Reducing the search space
- Tracking interesting interleavings
Key idea: decomposing thread interleaving

Our interleaving coverage is based on interleaving segments
SegFuzz

**Single-thread fuzzing**

- Inputs
- `cfg`

**Multi-thread fuzzing**

- Syscall A
- Syscall B
Single-thread fuzzing

- Explore execution paths
- Identify two system calls that potentially cause a concurrency bug

Please check our paper!
Our approach: SegFuzz

- **Multi-thread fuzzing**
  - Explore thread interleavings
  - *Utilizing interleaving coverage*
    - called *interleaving segment coverage*
Multi-thread fuzzing of SegFuzz

- Mutation-based Interleaving generator
- Unexplored interleaving
- Executor
- Interleaving segment coverage
- Single-thread fuzzing
- System calls
- Bug
- Single-thread fuzzing
- System calls
- Bug
Multi-thread fuzzing of SegFuzz

**Tracking explored interleavings**

- Mutation-based Interleaving generator
- Unexplored interleaving
- Executor
- System calls
- Bug

**Interleaving segment coverage**
Interleaving segment coverage

Syscall A

if (flag)
init_ptr(ptr);

Syscall B

flag = 1;

if (flag)
access_ptr(ptr);
Interleaving segment coverage

Whole thread interleaving

Syscall A

2
if (flag)

3
if (flag)

4

Syscall B

1
flag = 1;

Execution

Syscall A

flag = 1;

if (flag)
init_ptr(ptr);

if (flag)
access_ptr(ptr);
Interleaving segment coverage

Syscall A

if (flag)

if (flag)

Syscall B

flag = 1;

Whole thread interleaving
Interleaving segment coverage

Syscall A

1
flag = 1;

2
if (flag)

3
if (flag)

Syscall B

Whole thread interleaving

Segment #1

1
flag = 1;

2
if (flag)

3
if (flag)

Interleaving segments
(each contains at most 4 inst.)
Interleaving segment coverage

Segment #1

1. flag = 1;
2. if (flag)
3. if (flag)

Segment #2

1. flag = 1;
2. if (flag)
3. if (flag)

 Syscall A

1. flag = 1;
2. if (flag)
4. if (flag)

 Syscall B

1. flag = 1;
3. if (flag)

Whole thread interleaving

Interleaving segments
(each contains at most 4 inst.)
Interleaving segment coverage

Syscall A

Syscall B

Segment #1

flag = 1;

if (flag)

Segment #2

flag = 1;

if (flag)

Segment #3

flag = 1;

if (flag)

Whole thread interleaving

Interleaving segments
(each contains at most 4 inst.)

1

2

3

4

5

1

2

3

4

5
Interleaving segment coverage

- *Interleaving segment coverage*
  - Collection of segments

Interleaving segments
(each contains at most 4 inst.)
Interleaving segment coverage

- **Interleaving segment coverage**
  - Collection of segments
Interleaving segment coverage

- Interleaving segment coverage
  - Collection of segments

There are more interleavings of these instructions that we have not explored (including the offending interleaving).
Multi-thread fuzzing of SegFuzz

**Searching for unexplored interleavings**

- **Mutation-based Interleaving generator**
- **Unexplored interleaving**
- **Executor**
- **Bug**
- **Interleaving segment coverage**
- **Single-thread fuzzing**
- **System calls**
Mutation-based interleaving generator

- Mutating interleavings within segments to generate unexplored interleavings

Segment #1

1. flag = 1;
2. if (flag)
3. if (flag)
Mutation-based interleaving generator

- Mutating interleavings within segments to generate unexplored interleavings

Segment #1

Mutated segment #1-1

Mutated segment #1-2
Mutating interleavings within segments to **generate unexplored interleavings**

Segment #1

1. if (flag)
2. if (flag)
3. if (flag)

flag = 1;

Mutated segment #1-1

1. if (flag)
2. if (flag)
3. if (flag)

flag = 1;

The concurrency bug occurs when exploring this mutated segment
Mutation-based interleaving generator

- Mutating interleavings within segments to generate unexplored interleavings

- Testing multiple mutated segments at one execution
  - Recomposing mutated segments to determine how to schedule instructions
  - Please check our paper!
Evaluation

- 21 new concurrency bugs in the Linux kernel

<table>
<thead>
<tr>
<th>Crash Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>general protection fault in vmci_host_poll</td>
</tr>
<tr>
<td>KASAN: use-after-free Read in cfusb1_device_notify</td>
</tr>
<tr>
<td>KASAN: use-after-free Read in slcan_RECEIVE_BUF</td>
</tr>
<tr>
<td>general protection fault in cttimeout_net_exit</td>
</tr>
<tr>
<td>KASAN: use-after-free Read in raw_notifier_call_chain</td>
</tr>
<tr>
<td>INFO: task hung in blk_trace_remove</td>
</tr>
<tr>
<td>INFO: task hung in blk_trace_setup</td>
</tr>
<tr>
<td>kernel BUG in pfkey_send_acquire</td>
</tr>
<tr>
<td>general protection fault in add_wait_queue exclusive</td>
</tr>
<tr>
<td>KASAN: use-after-free Read in slip_ioctl</td>
</tr>
<tr>
<td>general protection fault in add_wait_queue</td>
</tr>
<tr>
<td>WARNING in isotp_tx_timer_handler</td>
</tr>
<tr>
<td>KASAN: use-after-free Read in snd_pcm_plugin_read_transfer</td>
</tr>
<tr>
<td>Kernel BUG in find_lock_entries</td>
</tr>
<tr>
<td>KASAN: use-after-free Read in tcp_write_timer_handler</td>
</tr>
<tr>
<td>KASAN: use-after-free Read in event_sched_out</td>
</tr>
<tr>
<td>general protection fault in soft_cursor</td>
</tr>
<tr>
<td>KASAN: use-after-free Read in perf_event_groups_insert</td>
</tr>
<tr>
<td>BUG: unable to handle kernel paging request in usb_start_wait_urb</td>
</tr>
<tr>
<td>BUG: unable to handle kernel paging request in __kernfs_new_node</td>
</tr>
<tr>
<td>general protection fault in raw_seq_start</td>
</tr>
</tbody>
</table>
Evaluation

- 21 new concurrency bugs in the Linux kernel

Use-after-free

<table>
<thead>
<tr>
<th>Crash Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>general protection fault in vmci_host_poll</td>
</tr>
<tr>
<td>KASAN: use-after-free Read in cfusbl_device_notify</td>
</tr>
<tr>
<td>KASAN: use-after-free Read in slcan_receive_buf</td>
</tr>
<tr>
<td>general protection fault in ctimeout_net_exit</td>
</tr>
<tr>
<td>KASAN: use-after-free Read in raw_notifier_call_chain</td>
</tr>
<tr>
<td>INFO: task hung in blk_trace_remove</td>
</tr>
<tr>
<td>INFO: task hung in blk_trace_setup</td>
</tr>
<tr>
<td>kernel BUG in pfkey_send_acquire</td>
</tr>
<tr>
<td>general protection fault in add_wait_queue_exclusive</td>
</tr>
<tr>
<td>KASAN: use-after-free Read in slip_ioctl</td>
</tr>
<tr>
<td>general protection fault in add_wait_queue</td>
</tr>
<tr>
<td>WARNING in isotp_tx_timer_handler</td>
</tr>
<tr>
<td>KASAN: use-after-free Read in snd_pcmplug_read_transfer</td>
</tr>
<tr>
<td>Kernel BUG in find_lock_entries</td>
</tr>
<tr>
<td>KASAN: use-after-free Read in tcp_write_timer_handler</td>
</tr>
<tr>
<td>KASAN: use-after-free Read in event_sched_out</td>
</tr>
<tr>
<td>general protection fault in soft_cursor</td>
</tr>
<tr>
<td>KASAN: use-after-free Read in perf_event_groups_insert</td>
</tr>
<tr>
<td>BUG: unable to handle kernel paging request in usb_start_wait_urb</td>
</tr>
<tr>
<td>BUG: unable to handle kernel paging request in __kernfs_new_node</td>
</tr>
<tr>
<td>general protection fault in raw_seq_start</td>
</tr>
</tbody>
</table>
Compare against Snowboard, KRace, and Syzkaller with 9 kernel concurrency bugs
Evaluation - Comparison study

- Compare against Snowboard, KRace, and Syzkaller with 9 kernel concurrency bugs

*SegFuzz discovers concurrency bugs 4.1x faster than previous approaches*
Conclusion

- **SegFuzz**, a fuzzing framework to effectively discover kernel concurrency bugs
  - Applying the problem decomposition strategy based on the previous finding

- A novel thread interleaving coverage called **interleaving segment coverage**
  - Tracking explored thread interleavings
  - Efficiently exploring unexplored thread interleavings

- Discovered 21 new concurrency bugs in the Linux kernel
SEGFUZZ: Segmentizing Thread Interleaving to Discover Kernel Concurrency Bugs through Fuzzing

Thank You!