Razzer: Finding Kernel Race Bugs through Fuzzing

Dae R. Jeong†  Kyungtae Kim*  Basavesh Shivakumar*  Byoungyoung Lee‡*  Insik Shin†

†Korea Advanced Institute of Science and Technology
‡Seoul National University
*Purdue University
Kernel Vulnerability
Kernel Vulnerability
Kernel Vulnerability

Attacker can control the entire system
Fuzzing: Focused to Extend Coverage

• Fuzzing
  • One of the most practical approaches in finding vulnerabilities

• Coverage-guided fuzzing
  • It gathers **interesting** inputs that extend code coverage.
  • The more coverage, the more vulnerabilities
Race Bugs

• Assumption: Race condition between two threads

• Race condition occurs if following three conditions meet
  • Two instructions access the same memory location
  • At least one of two is a write instruction
  • These two are executed concurrently

• If a race occurs, the computational results may vary depending on the execution order
  • A race vulnerability is caused by the execution order unintended by developers.
Inefficient Fuzzing for Race Bugs

• Traditional fuzzers are inefficient to find race bugs
  • Instructions should be executed within a specific time window
    • Called as race window
  • Execution orders are not determined by the fuzzer
    • Execution orders are determined by the kernel scheduler
Inefficient Fuzzing for Race Bugs: Example

Thread 1

```
Syscall: open()

len = strlen(file_name);
buf = kmalloc(len);
```

```
strcpy(buf, file_name);
```

Thread 2

```
Syscall: rename()

strcpy(file_name, longer_name);
```
Inefficient Fuzzing for Race Bugs: Example

Thread 1

- `Syscall: open()`
  - `len = strlen(file_name);`
  - `buf = kmalloc(len);`
  - `strcpy(buf, file_name);`

Thread 2

- `Syscall: rename()`
  - `strcpy(file_name, longer_name);`
Inefficient Fuzzing for Race Bugs: Example

Thread 1

Syscall: open()

len = strlen(file_name);
buf = kmalloc(len);

strcpy(buf, file_name);

Thread 2

Syscall: rename()

strcpy(file_name, longer_name);
Inefficient Fuzzing for Race Bugs: Example

Thread 1

- Syscall: open()
  - len = strlen(file_name);
  - buf = kmalloc(len);
- strcpy(buf, file_name);

Thread 2

- Syscall: rename()
  - strcpy(file_name, longer_name);
Inefficient Fuzzing for Race Bugs: Example

Thread 1

Syscall: open()

len = strlen(file_name);
buf = kmalloc(len);

strcpy(buf, file_name);

Thread 2

Syscall: rename()

strcpy(file_name, longer_name);

file_name is longer than the allocated buffer
Inefficient Fuzzing for Race Bugs: Example

Thread 1

Syscall: open()

len = strlen(file_name);
buf = kmalloc(len);

strcpy(buf, file_name);

Buffer overflow!

Thread 2

Syscall: rename()

strcpy(file_name, longer_name);

file_name is longer than the allocated buffer
Inefficient Fuzzing for Race Bugs: Syzkaller

• Syzkaller
  • A kernel syscall fuzzer developed by Google

• Run Syzkaller to find three race bugs with limited set of syscalls
  • CVE-2016-8655
  • CVE-2017-17712
  • CVE-2017-2636

• None of CVEs was found within 10 hours
  • Traditional fuzzing is inefficient to find race bugs
  • Razzer can find all of them within 7~30 minutes
Our approach: Razzer

Code coverage

Thread interleaving

\[
\begin{align*}
\text{len} &= \text{strlen(}file\_name)\text{);} \\
\text{buf} &= \text{kmalloc(}len\text{);} \\
\text{strcpy(}file\_name, \ longer\_name)\text{;} \\
\text{strcpy(}buf, \ file\_name)\text{;}
\end{align*}
\]
Our approach: Razzer

Thread 1

Syscall: open()

len = strlen(file_name);
buf = kmalloc(len);

strcpy(buf, file_name);

Thread 2

Syscall: rename()

strcpy(file_name, longer_name);
Our approach: Razzer

Thread 1

- **Syscall: open()**
  - `len = strlen(file_name);`
  - `buf = kmalloc(len);`
  - `strcpy(buf, file_name);`

Thread 2

- **Syscall: rename()**
  - `strcpy(file_name, longer_name);`

Race window
Our approach: Razzer

Thread 1

Syscall: open()

len = strlen(file_name);
buf = kmalloc(len);
strcpy(buf, file_name);

Thread 2

Syscall: rename()

strcpy(file_name, longer_name);
Our approach: Razzer

Thread 1

```
len = strlen(file_name);
buf = kmalloc(len);
strcpy(buf, file_name);
```

Thread 2

```
Syscall: rename()

strcpy(file_name, longer_name);
```
Our approach: Razzer

Thread 1

Syscall: open()

len = strlen(file_name);
buf = kmalloc(len);

strcpy(buf, file_name);

Thread 2

Syscall: rename()

strcpy(file_name, longer_name);
Our approach: Razzer

Thread 1

Syscall: open()

len = strlen(file_name);
buf = kmalloc(len);
strcpy(buf, file_name);

Buffer overflow!

Thread 2

Syscall: rename()

strcpy(file_name, longer_name);
Design Overview

Offline analysis

Source code → Static analysis → Over-approximated data races

Online testing

Multi-thread input → Single-thread fuzzing → Multi-thread fuzzing
Design Overview

Offline analysis

Source code → Static analysis

Over-approximated data races

Online testing

Single-thread fuzzing → Multi-thread input

Multi-thread fuzzing
Static Analysis

• Identifying instructions that may race
  • Teaching Razzer where to install breakpoints to trigger race

• Inclusion-based points-to analysis
  • Also known as Andersen-style points-to analysis

• This static analysis certainly has false positives
  • Next phases (fuzzing) takes care of this issue because it is “fuzzing”
Static Analysis: Example

Source code

len = strlen(file_name);
buf = kmalloc(len);

strcpy(file_name);

strcpy(file_name, longer_name);

strcpy(buf, file_name);
Razzer identified **3.4M** race candidates over the entire Linux kernel.

```c
len = strlen(file_name);
buf = kmalloc(len);
strcpy(buf, file_name);
strcpy(file_name, longer_name);
```
Design Overview

- Offline analysis
  - Source code
  - Static analysis
  - Over-approximated data races

- Online testing
  - Multi-thread input
  - Single-thread fuzzing
  - Multi-thread fuzzing
Single-thread Fuzzing

Single-thread input

```
.. open()
rename()
..```


Single-thread Fuzzing

Thread 1

Syscall: open()

len = strlen(file_name);
buf = kmalloc(len);

strncpy(buf, file_name);

Syscall: rename()

strncpy(file_name, longer_name);
Transformation to Multi-thread Input

```plaintext
... open() ...
rename() ...
...
Transformation to Multi-thread Input

Thread 1

\[ \text{open()} \]

Thread 2

\[ \text{rename()} \]
Transformation to Multi-thread Input

Thread 1

```
open()
len = strlen(file_name);
buf = kmalloc(len);
strcpy(buf, file_name);
```

Thread 2

```
rename()
strcpy(file_name, longer_name);
```
Design Overview

Offline analysis

Source code

Over-approximated data races

Static analysis

Online testing

Single-thread fuzzing

Multi-thread input

Multi-thread fuzzing
```c

buf = kmalloc(len);
len = strlen(file_name);
... 

CPU 1

Thread 1

Syscall \textit{n} 

... 

strcpy(buf, file_name);

CPU 2

Thread 2

Syscall \textit{m} 

... 

strcpy(file_name, longer_name);

Guest VM

```

Multi-thread Fuzzing

...
Multi-thread Fuzzing

```
strcpy(buf, file_name);
len = strlen(file_name);
buf = kmalloc(len);

Thread 1

Syscall n

```

```
Thread 2

Syscall m

strcpy(file_name, longer_name);

```

Guest VM

Hypervisor
```c
strcpy(buf, file_name);
strcpy(file_name, longer_name);
len = strlen(file_name);
buf = kmalloc(len);
```

Multi-thread Fuzzing

CPU 1

Thread 1
```
Hypercall
Syscall n
``` 

CPU 2

Thread 2
```
Hypercall
Syscall m
``` 

Guest VM
Hypervisor

Thread 1
```
Hypercall
Syscall n
```
**Multi-thread Fuzzing**

```
strcpy(buf, file_name);
```
Multi-thread Fuzzing

```
strcpy(buf, file_name);
len = strlen(file_name);
buf = kmalloc(len);
strcpy(buf, file_name);
```

```
strcpy(file_name, longer_name);
```
Multi-thread Fuzzing

Two threads access the same memory => A race condition is occurred

CPU 1

Thread 1

:\n
Hypercall

Syscall $n$

len = strlen(file_name);

buf = kmalloc(len);

strcpy(buf, file_name);

CPU 2

Thread 2

Hypercall

Syscall $m$

\[\text{COPY} \begin{array}{c}
\text{file\_name} \\
\text{other\_name}
\end{array}\]
Implementation

• **Static analysis**
  • Implemented using SVF which is based on LLVM compiler suite

• **Single-thread/Multi-thread fuzzing**
  • Implemented based on Syzkaller
  • Deterministic scheduler
    • Implemented using QEMU/KVM
    • Exposing hypercall interfaces to support per-core breakpoint
Evaluation

- 30 new races in the Linux kernel
- 15 were fixed
### Evaluation

- 30 new races in the Linux kernel
- 15 were fixed

<table>
<thead>
<tr>
<th>Kernel crash summary</th>
<th>Crash type</th>
</tr>
</thead>
<tbody>
<tr>
<td>KASAN: slab-out-of-bounds write in try_insert_flip_string_flag</td>
<td>Use-After-Free</td>
</tr>
<tr>
<td>WARNING in __static_key_slow_dec</td>
<td>Reachable Warning</td>
</tr>
<tr>
<td>Kernel BUG at net/packet/af_packet.c.LINE!</td>
<td>Reachable Assertion</td>
</tr>
<tr>
<td>WARNING in refcount_dec</td>
<td>Reachable Warning</td>
</tr>
<tr>
<td>unable to handle kernel paging request in snd_seq_oss_readq_puts</td>
<td>Page Fault</td>
</tr>
<tr>
<td><strong>KASAN: use-after-free Read in loopback_active_get</strong></td>
<td>Use-After-Free</td>
</tr>
<tr>
<td>KASAN: null.ptr.deref Read in rds_ib_get_nr</td>
<td>Null ptr deref</td>
</tr>
<tr>
<td>KASAN: null.ptr.deref Read in list_tru_del</td>
<td>Null ptr deref</td>
</tr>
<tr>
<td>BUG: unable to handle kernel NULL ptr dereference in corrupted</td>
<td>Null ptr deref</td>
</tr>
<tr>
<td><strong>KASAN: use-after-free Read in nd_jump_root</strong></td>
<td>Use-After-Free</td>
</tr>
<tr>
<td><strong>KASAN: use-after-free Read in link_path_walk</strong></td>
<td>Use-After-Free</td>
</tr>
<tr>
<td>BUG: unable to handle kernel paging request in __inet_check_established</td>
<td>Page Fault</td>
</tr>
<tr>
<td>KASAN: null.ptr.deref Read in ata_pio_sector</td>
<td>Null ptr deref</td>
</tr>
<tr>
<td>WARNING in ip_recv_error</td>
<td>Reachable Warning</td>
</tr>
<tr>
<td>WARNING in remove_proc_entry</td>
<td>Reachable Warning</td>
</tr>
<tr>
<td>KASAN: null.ptr.deref Read in ip6gre_exit_batch_net</td>
<td>Null ptr deref</td>
</tr>
<tr>
<td>KASAN: slab-out-of-bounds Write in __register_sysctl_table</td>
<td>Heap overflow</td>
</tr>
<tr>
<td><strong>KASAN: use-after-free Write in skb_release_data</strong></td>
<td>Use-After-Free</td>
</tr>
<tr>
<td>KASAN: invalid-free in plock_free</td>
<td>Double free</td>
</tr>
<tr>
<td>Kernel BUG at lib/list_debug.c.LINE!</td>
<td>Reachable Assertion</td>
</tr>
<tr>
<td>INFO: trying to register non-static key in __handle_rmm_fault</td>
<td>Reachable INFO</td>
</tr>
<tr>
<td><strong>KASAN: use-after-free Read in rtnet_itr_write_lite</strong></td>
<td>Use-After-Free</td>
</tr>
<tr>
<td>BUG: soft lockup in vmemdup_user</td>
<td>Soft lockup</td>
</tr>
<tr>
<td><strong>KASAN: use-after-free Read in rds_tcp_accept_one</strong></td>
<td>Use-After-Free</td>
</tr>
<tr>
<td>WARNING in sg_rq_end_io</td>
<td>Reachable Warning</td>
</tr>
<tr>
<td>BUG: soft lockup in snd_virmidi_output_trigger</td>
<td>Soft lockup</td>
</tr>
<tr>
<td>KASAN: null.ptr.deref Read in smc_ioctl</td>
<td>Null ptr deref</td>
</tr>
<tr>
<td>KASAN: null.ptr.deref Write in binderf_update_page_range</td>
<td>Null ptr deref</td>
</tr>
<tr>
<td>WARNING in port_delete</td>
<td>Reachable Warning</td>
</tr>
<tr>
<td>KASAN: null.ptr.deref in inode_permission</td>
<td>Null ptr deref</td>
</tr>
</tbody>
</table>
Evaluation

- 30 new races in the Linux kernel
- 15 were fixed
Evaluation

- 30 new races in the Linux kernel
- 15 were fixed

<table>
<thead>
<tr>
<th>Kernel crash summary</th>
<th>Crash type</th>
</tr>
</thead>
<tbody>
<tr>
<td>KASAN: slab-out-of-bounds write in try_insert_flip_string_flag</td>
<td>Use-After-Free</td>
</tr>
<tr>
<td>WARNING in __static_key_slow_dec</td>
<td>Reachable Warning</td>
</tr>
<tr>
<td>Kernel BUG at net/packet/af_packet.c.LINE!</td>
<td>Reachable Assertion</td>
</tr>
<tr>
<td>WARNING in refcount_dec</td>
<td>Reachable Warning</td>
</tr>
<tr>
<td>WARNING in refcount_dec</td>
<td></td>
</tr>
<tr>
<td>unable to handle kernel paging request in snd_seq_oss_readq_puts</td>
<td>Page Fault</td>
</tr>
<tr>
<td>KASAN: use-after-free Read in loopback_active_get</td>
<td>Use-After-Free</td>
</tr>
<tr>
<td>KASAN: null-ptr-deref Read in rdq_ib_get_mr</td>
<td>Null ptr deref</td>
</tr>
<tr>
<td>KASAN: null-ptr-deref Read in list_tru_del</td>
<td>Null ptr deref</td>
</tr>
<tr>
<td>BUG: unable to handle kernel NULL ptr dereference in corrupted</td>
<td>Null ptr deref</td>
</tr>
<tr>
<td>KASAN: use-after-free Read in nd_jump_root</td>
<td>Use-After-Free</td>
</tr>
<tr>
<td>KASAN: use-after-free Read in link_path_walk</td>
<td>Use-After-Free</td>
</tr>
<tr>
<td>BUG: unable to handle kernel paging request in __inet_check_established</td>
<td>Page Fault</td>
</tr>
<tr>
<td>KASAN: null-ptr-deref Read in ata_pio_sector</td>
<td>Null ptr deref</td>
</tr>
<tr>
<td>WARNING in ip_recv_error</td>
<td>Reachable Warning</td>
</tr>
<tr>
<td>WARNING in remove_proc_entry</td>
<td>Reachable Warning</td>
</tr>
<tr>
<td>WARNING in remove_proc_entry</td>
<td></td>
</tr>
<tr>
<td>KASAN: null-ptr-deref Read in ip6grep_exit_batch_net</td>
<td>Null ptr deref</td>
</tr>
<tr>
<td>KASAN: slab-out-of-bounds Write in __register_systctl_table</td>
<td>Heap overflow</td>
</tr>
<tr>
<td>KASAN: use-after-free Write in skb_release_data</td>
<td>Use-After-Free</td>
</tr>
<tr>
<td>KASAN: undefined pointer in plock_free</td>
<td>Double free</td>
</tr>
<tr>
<td>Kernel BUG at lib/list_debug.c.LINE!</td>
<td>Reachable Assertion</td>
</tr>
<tr>
<td>INFO: trying to register non-static key in __handle_mm_fault</td>
<td>Reachable INFO</td>
</tr>
<tr>
<td>KASAN: use-after-free Read in vhost_ehr_write_liter</td>
<td>Use-After-Free</td>
</tr>
<tr>
<td>BUG: soft lockup in vmemdup_user</td>
<td>Soft lockup</td>
</tr>
<tr>
<td>KASAN: use-after-free Read in rds_tcp_accept_one</td>
<td>Use-After-Free</td>
</tr>
<tr>
<td>WARNING in sg_rq_end_io</td>
<td>Reachable Warning</td>
</tr>
<tr>
<td>BUG: soft lockup in snd_virmidi_output_trigger</td>
<td>Soft lockup</td>
</tr>
<tr>
<td>KASAN: null-ptr-deref Read in smc_ioctl</td>
<td>Null ptr deref</td>
</tr>
<tr>
<td>KASAN: null-ptr-deref Write in binderf_update_page_range</td>
<td>Null ptr deref</td>
</tr>
<tr>
<td>WARNING in port_delete</td>
<td>Reachable Warning</td>
</tr>
<tr>
<td>KASAN: null-ptr-deref in inode_permission</td>
<td>Null ptr deref</td>
</tr>
</tbody>
</table>
Evaluation: Comparison with Syzkaller

- Run Razzer and Syzkaller with limited set of syscalls

- Razzer found race bugs 23~85 faster than Syzkaller
  - Razzer found 3 race bugs within short time
  - Syzkaller didn’t find 3 race bugs within 10 hours

<table>
<thead>
<tr>
<th>Race bugs</th>
<th>Syzkaller</th>
<th>Razzer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># of exec</td>
<td>Time</td>
</tr>
<tr>
<td>CVE-2016-8655</td>
<td>29 M</td>
<td>10 hrs</td>
</tr>
<tr>
<td>CVE-2017-17712</td>
<td>37 M</td>
<td>10 hrs</td>
</tr>
<tr>
<td>CVE-2017-2636</td>
<td>5 M</td>
<td>10 hrs</td>
</tr>
</tbody>
</table>
Conclusion

• Razzer, a new fuzzer focusing on race bugs

• Taming non-deterministic behavior of races

• Combining static analysis and fuzzing

• Source code (by May 25, 2019)
  • https://github.com/compsec-snu/razzer
Thank you

Dae R. Jeong
threearcat@gmail.com