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

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

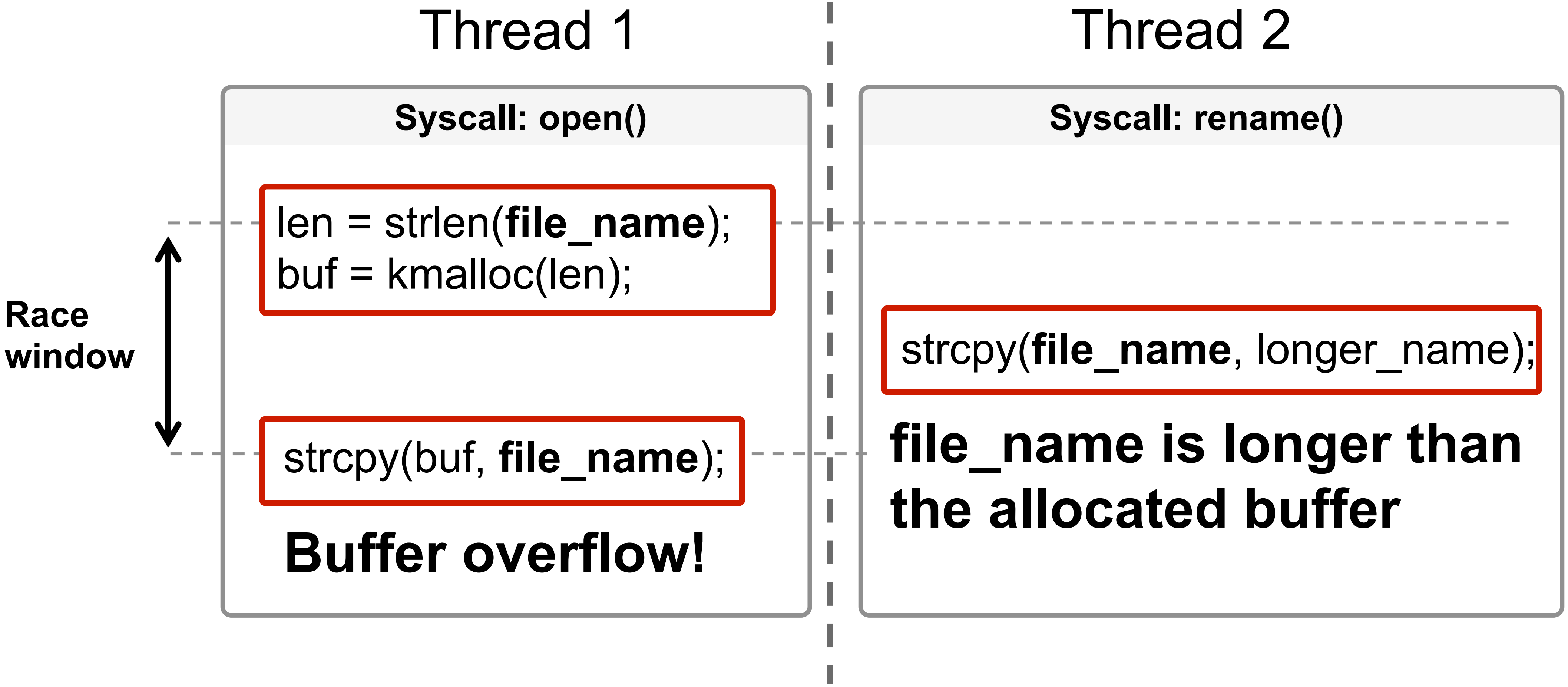
- Race condition occurs if following three conditions meet
  - Two instructions access the same memory location
  - At least one of two is a write instruction
  - These two are executed concurrently

- 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 dev elopers.

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);
```

Buffer overflow!

Race window

Thread 2

Syscall: rename()

```
strcpy(file_name, longer_name);
```

file_name is longer than the allocated buffer

Inefficient Fuzzing for Race Bugs: Syzkall er

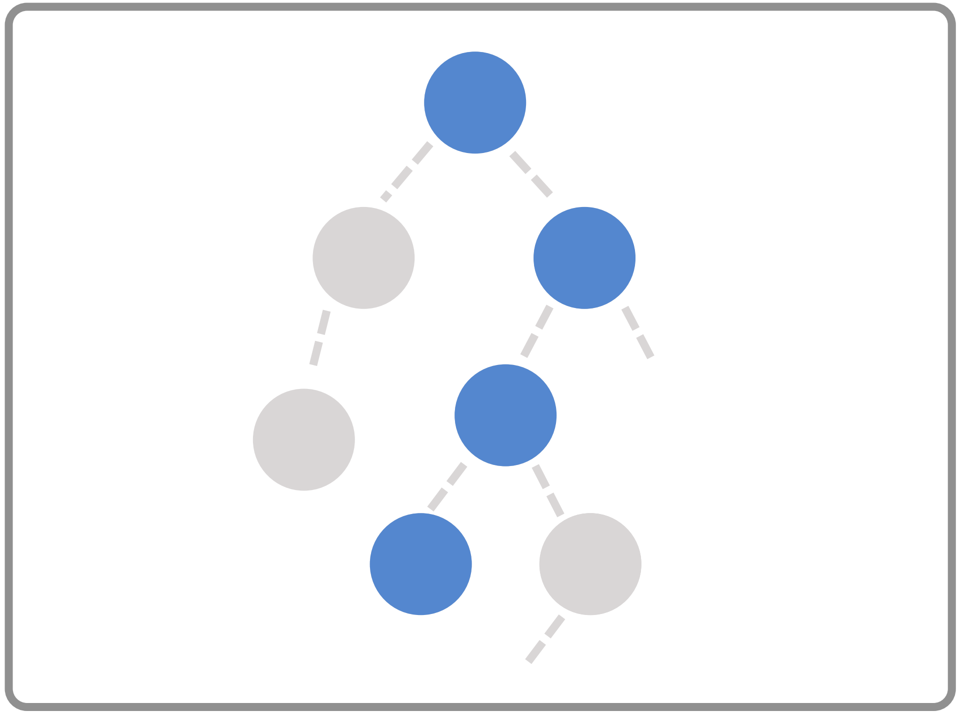
- Syzkaller
  - A kernel syscall fuzzer developed by Google

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

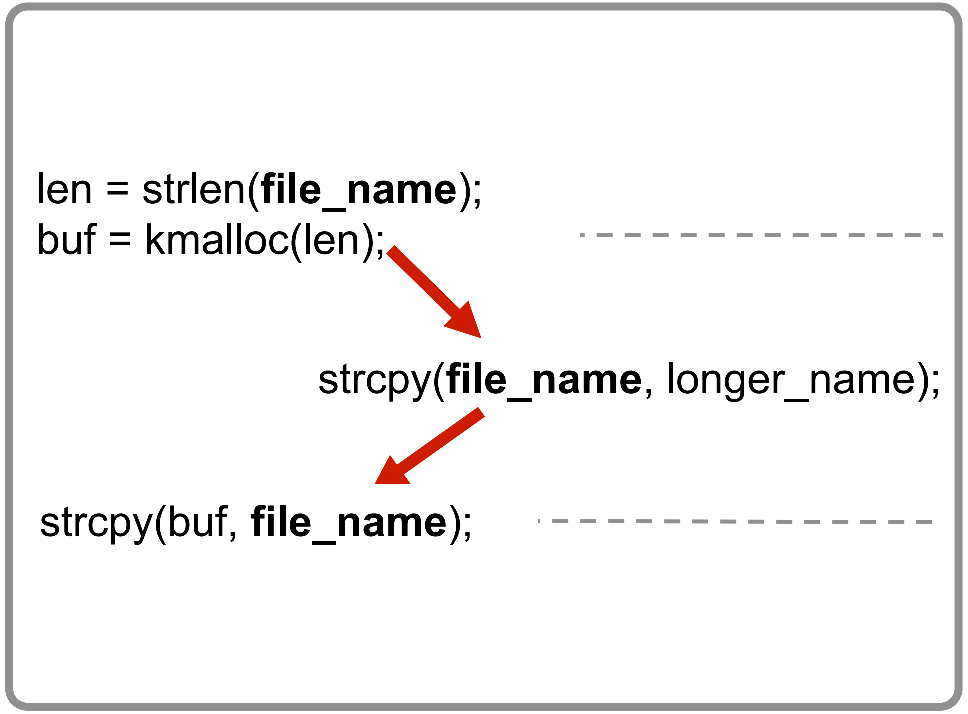
- None of CVEs was found within 10 hours
  - Traditional fuzzing is inefficient to find race bugs
  - Razzer can find all of them within 7~30 minutes

Our approach: Razzer

Code coverage

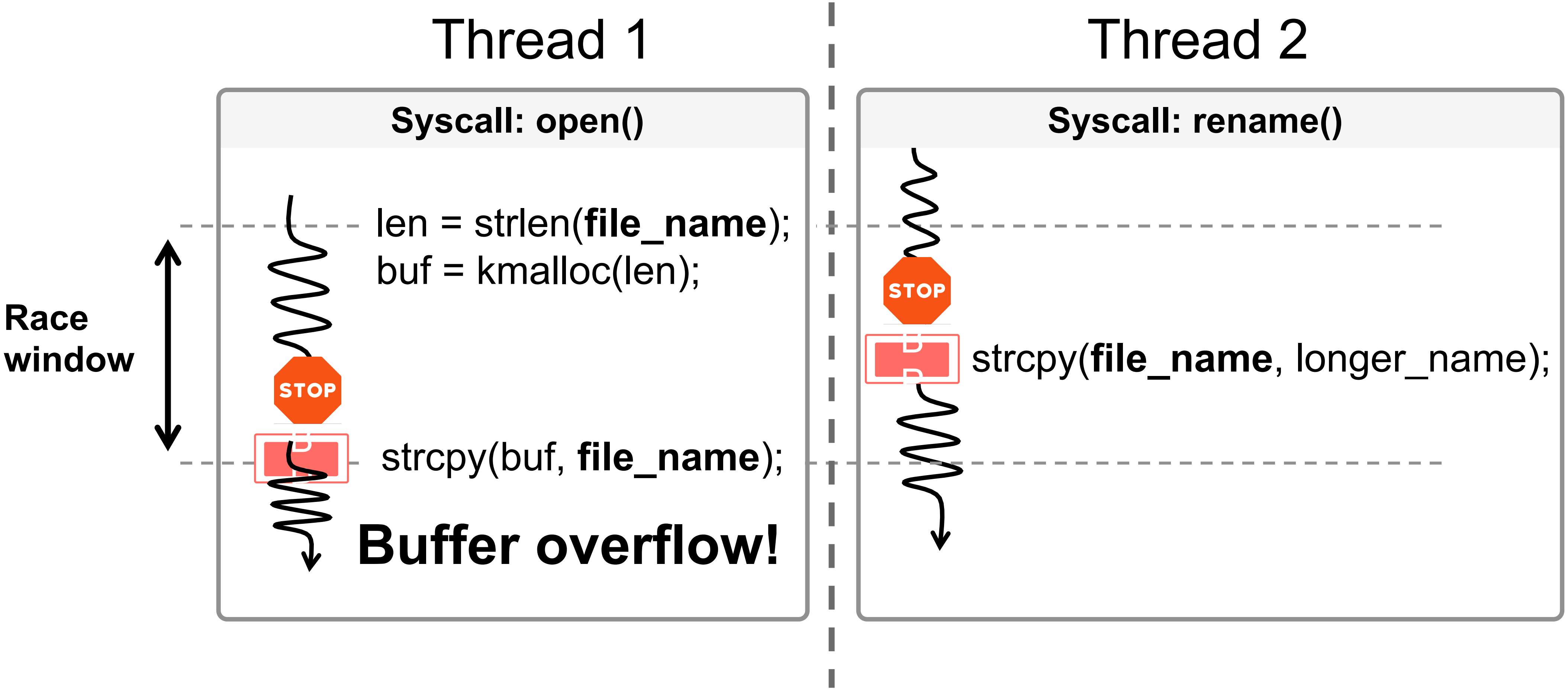


Thread interleaving



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

Our approach: Razzer



Thread 1

Syscall: open()

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

strcpy(buf, file_name);
```

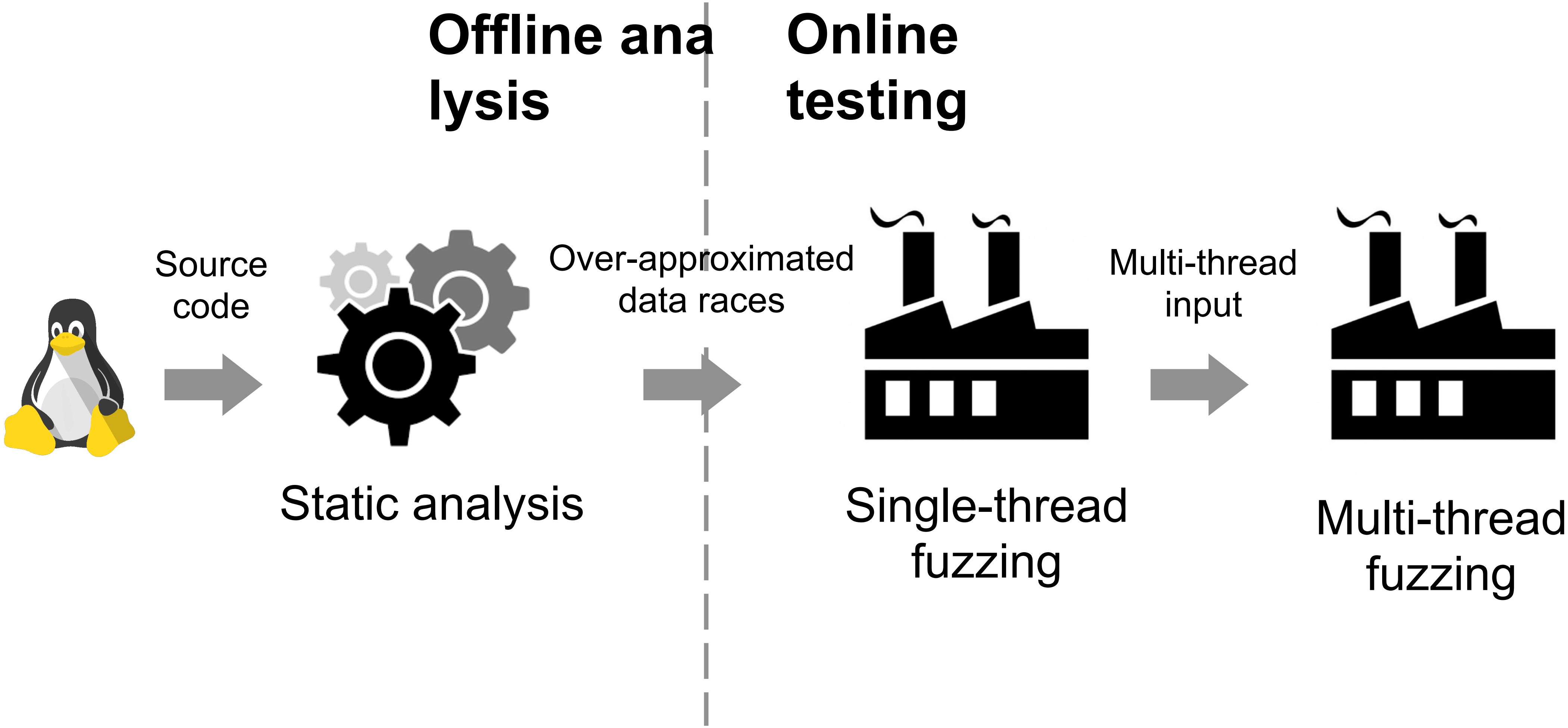
Buffer overflow!

Thread 2

Syscall: rename()

```
strcpy(file_name, longer_name);
```

Design Overview



Offline analysis

Online testing

Source code

Over-approximated data races

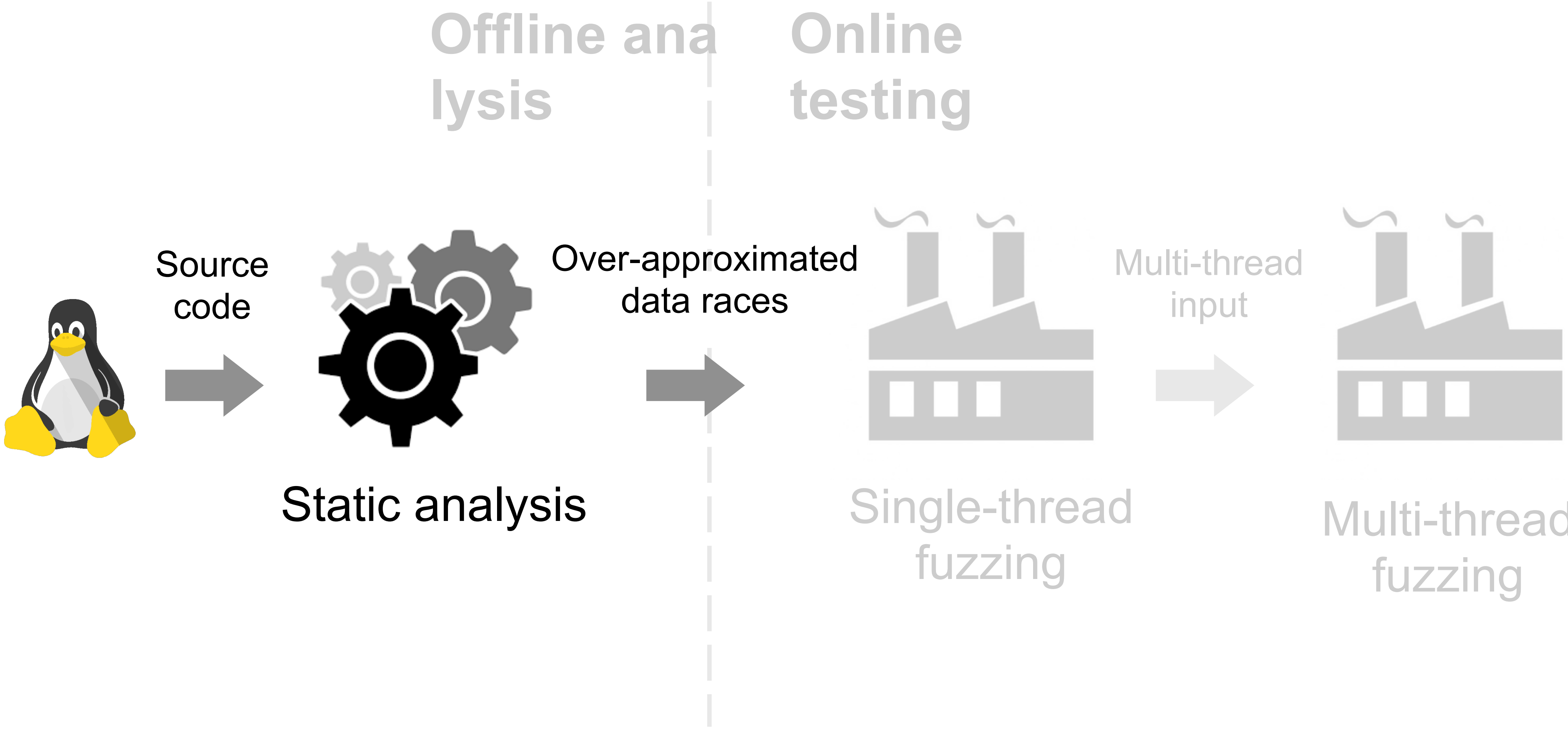
Multi-thread input

Static analysis

Single-thread fuzzing

Multi-thread fuzzing

Design Overview



Static Analysis

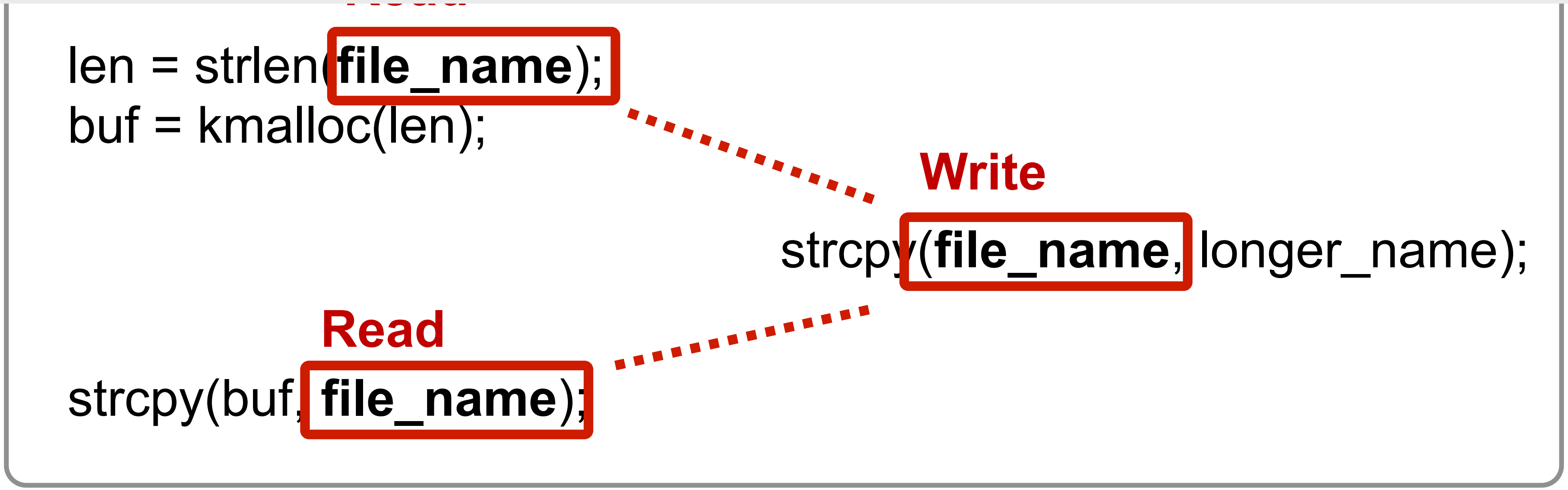
- Identifying instructions that may race
  - Teaching Razzer where to install breakpoints to trigger race

- 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

Razzer identified **3.4M** race candidates over the entire Linux kernel

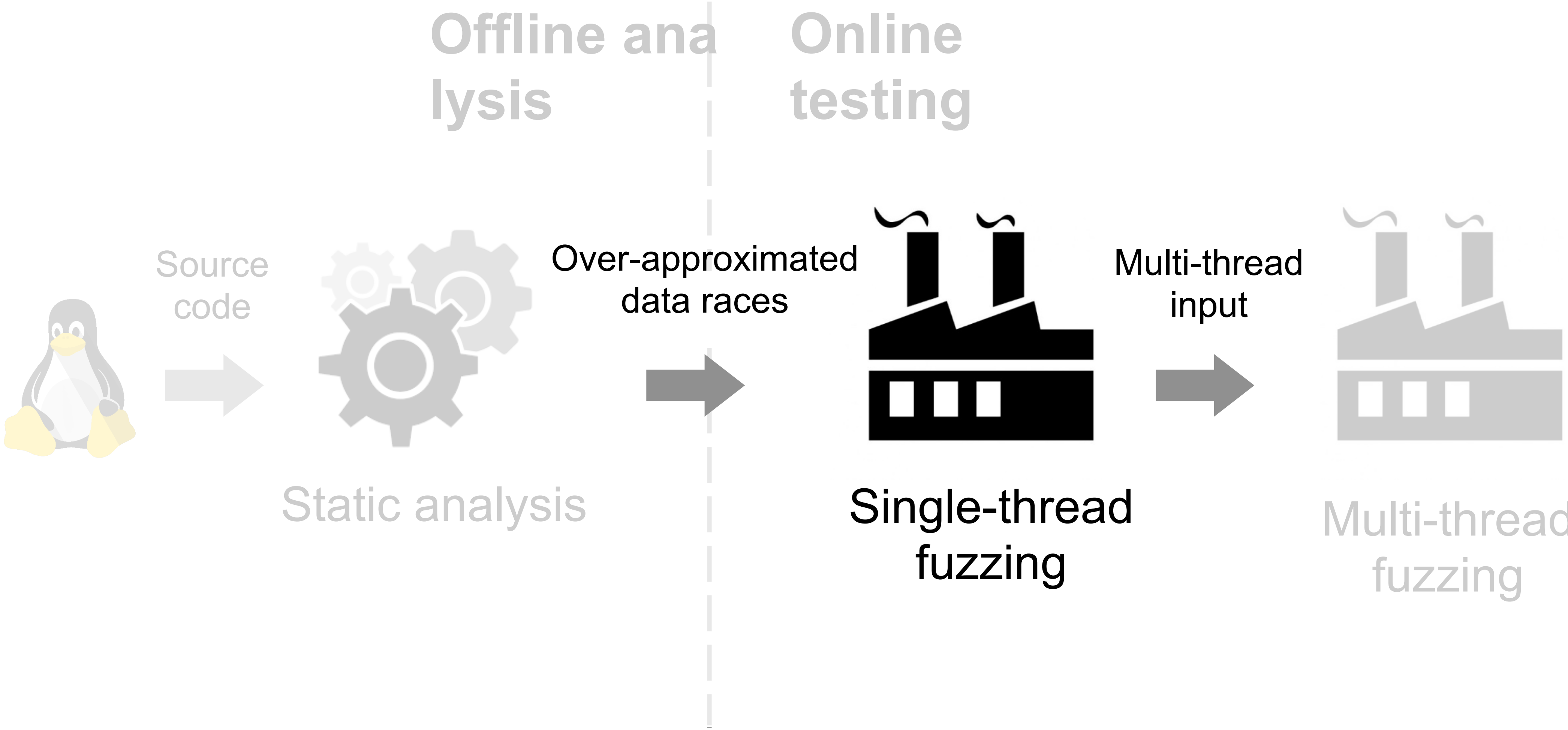


```
len = strlen(file_name);
buf = kmalloc(len);

                                    strcpy(file_name, longer_name);

strcpy(buf, file_name);
```

Design Overview



Offline ana lysis

Online testing

Source code

Static analysis

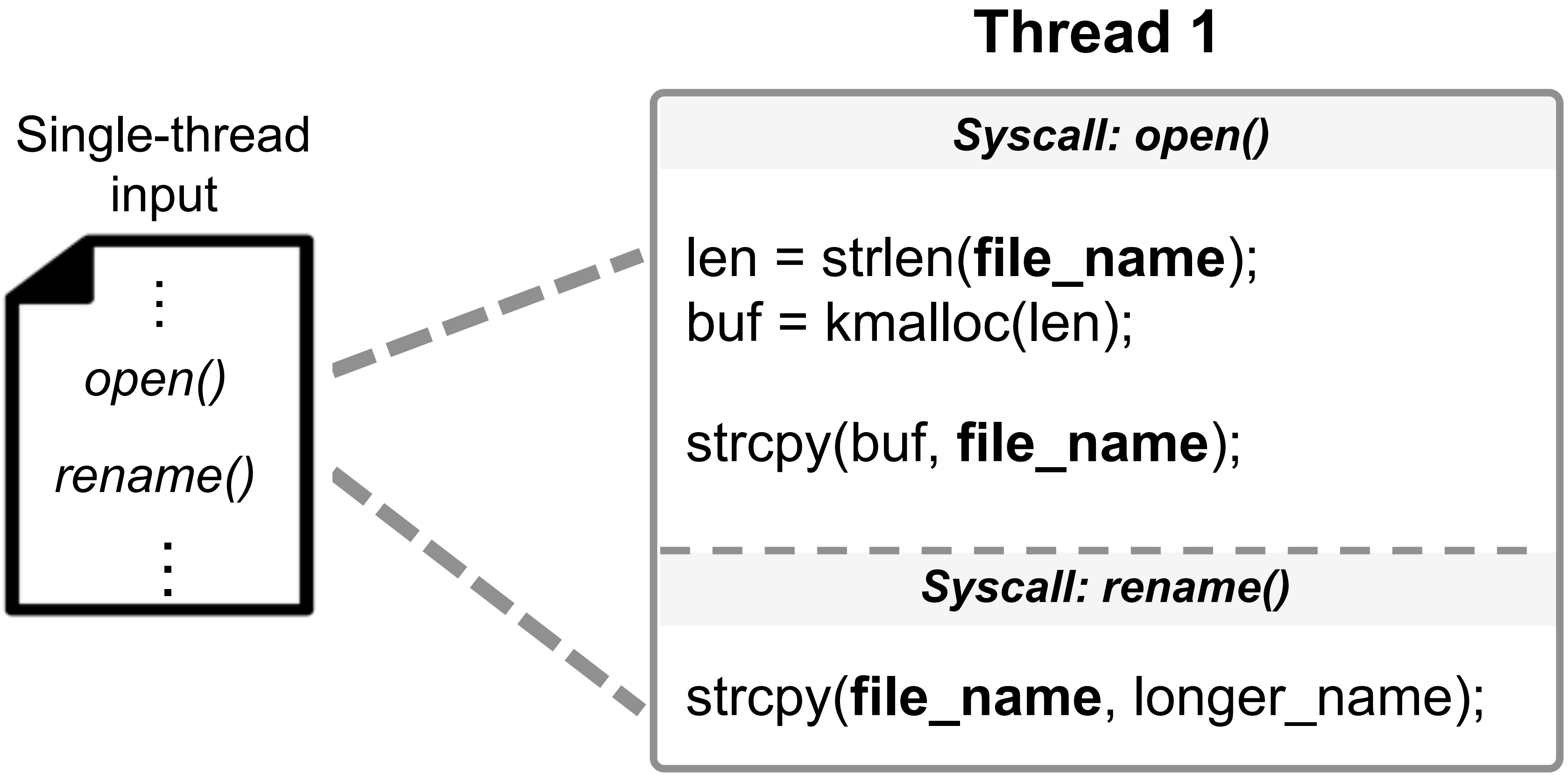
Over-approximated data races

Single-thread fuzzing

Multi-thread input

Multi-thread fuzzing

Single-thread Fuzzing



Thread 1

Syscall: open()

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

strcpy(buf, file_name);
```

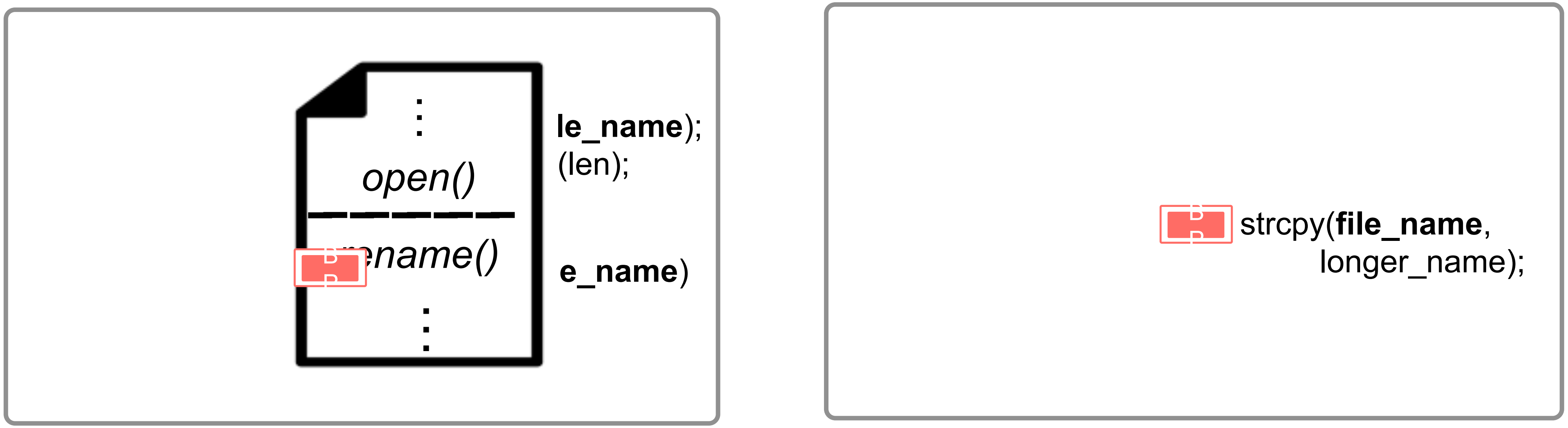
Syscall: rename()

```
strcpy(file_name, longer_name);
```

Transformation to Multi-thread Input

Thread 1

Thread 2

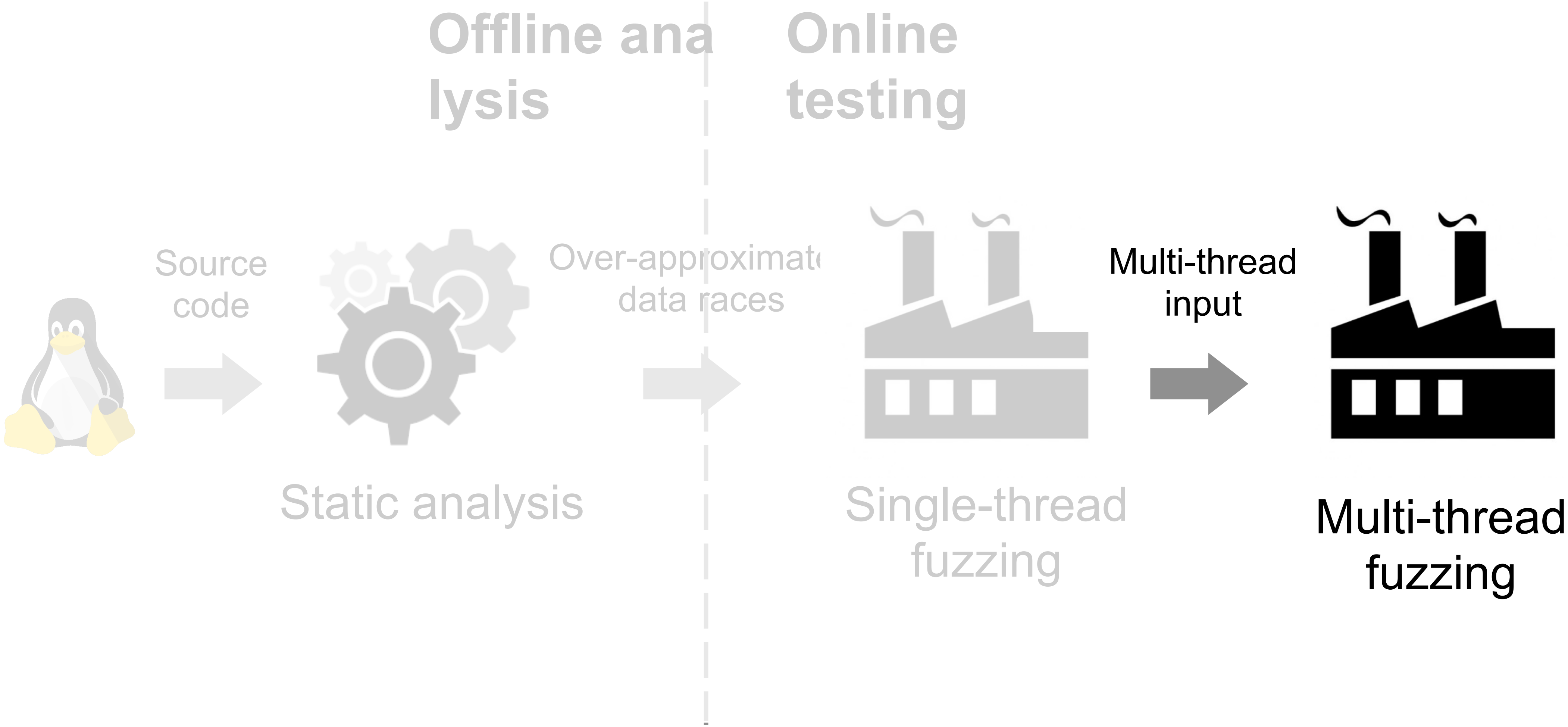


le_name);
(len);

e_name)

strcpy(**file_name**,
longer_name);

Design Overview



Offline analysis

Online testing

Source code

Static analysis

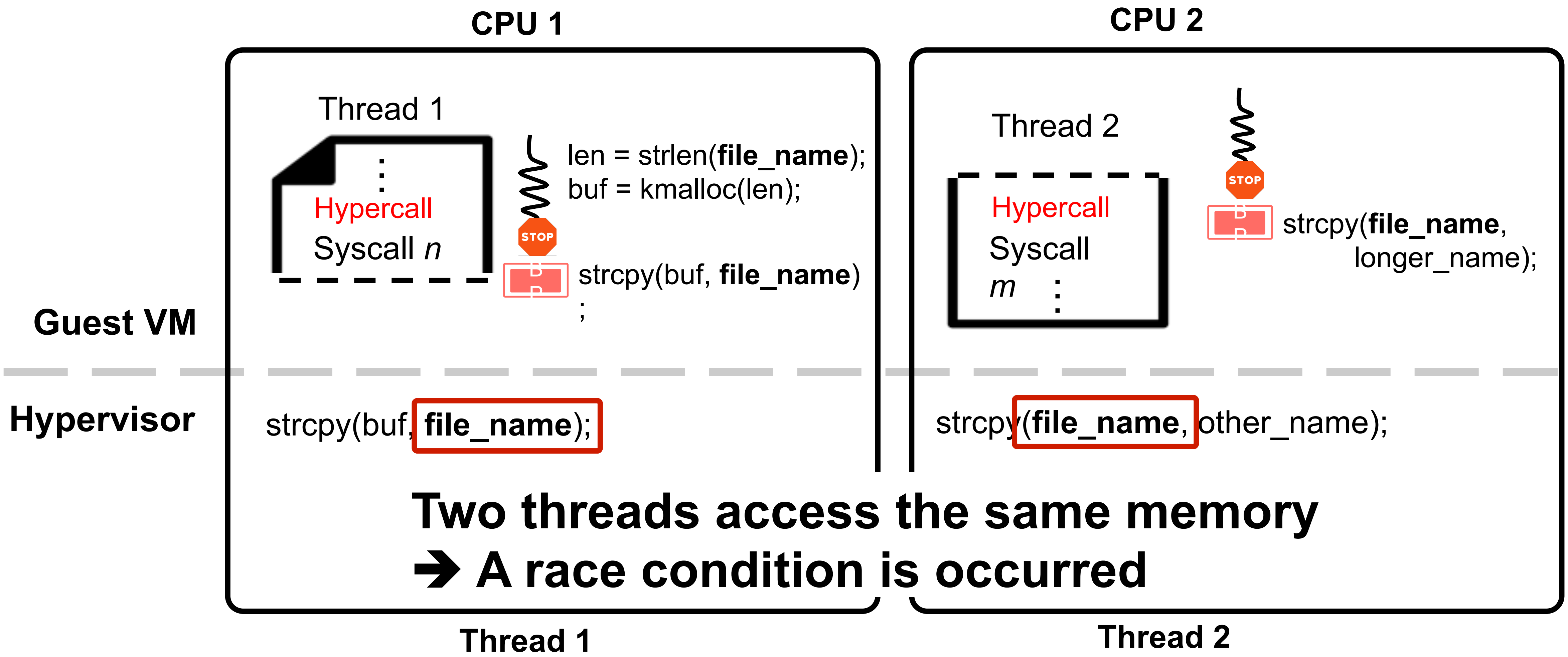
Over-approximate data races

Single-thread fuzzing

Multi-thread input

Multi-thread fuzzing

Multi-thread Fuzzing



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

```
strcpy(file_name,
       longer_name);
```

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

```
strcpy(file_name, other_name);
```

Two threads access the same memory
➔ A race condition is occurred

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

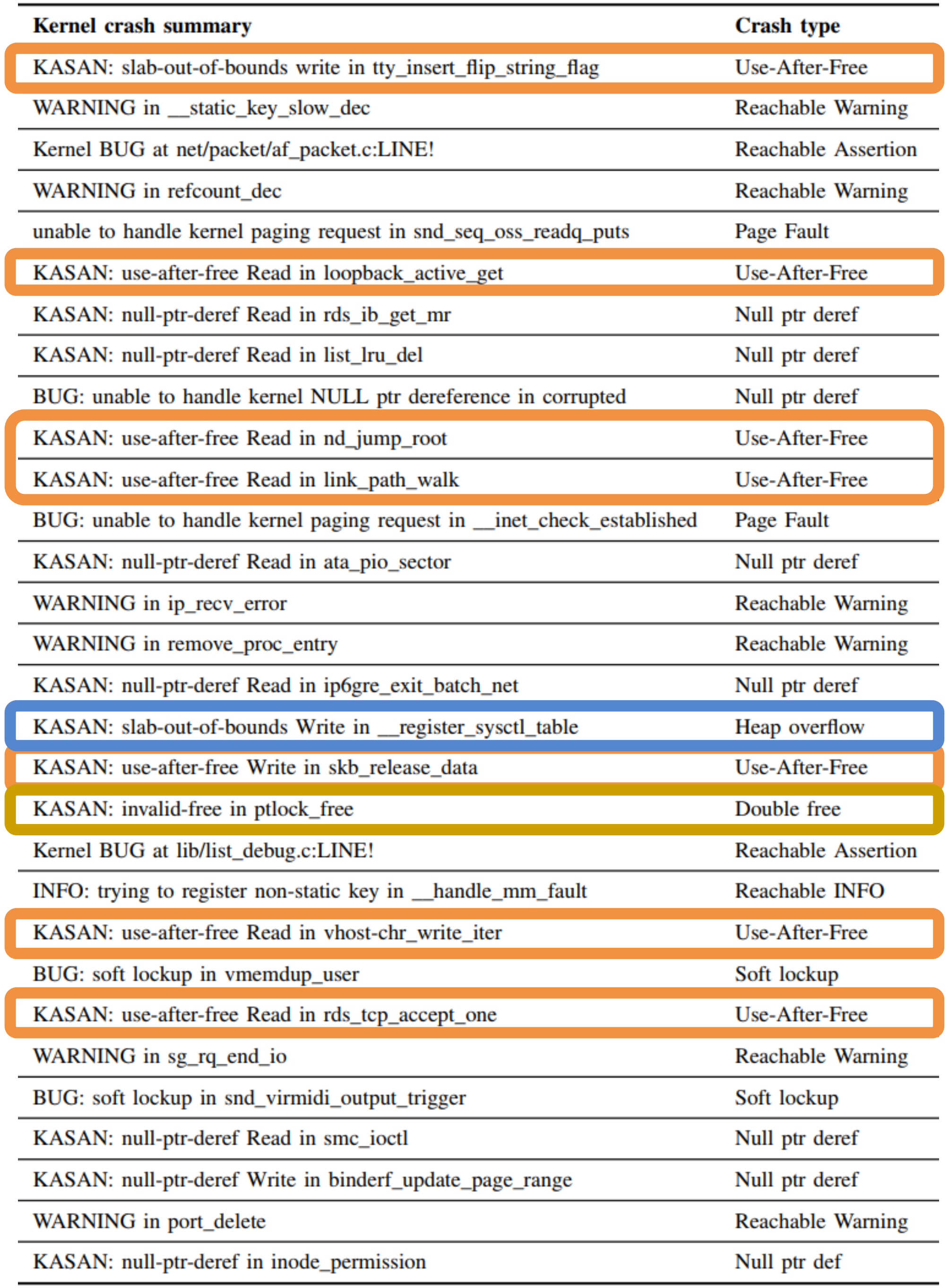
Evaluation

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

Use-after-free

Heap overflow

Double free



Kernel crash summary	Crash type
KASAN: slab-out-of-bounds write in tty_insert_flip_string_flag	Use-After-Free
WARNING in __static_key_slow_dec	Reachable Warning
Kernel BUG at net/packet/af_packet.c:LINE!	Reachable Assertion
WARNING in refcount_dec	Reachable Warning
unable to handle kernel paging request in snd_seq_oss_readq_puts	Page Fault
KASAN: use-after-free Read in loopback_active_get	Use-After-Free
KASAN: null-ptr-deref Read in rds_ib_get_mr	Null ptr deref
KASAN: null-ptr-deref Read in list_lru_del	Null ptr deref
BUG: unable to handle kernel NULL ptr dereference in corrupted	Null ptr deref
KASAN: use-after-free Read in nd_jump_root	Use-After-Free
KASAN: use-after-free Read in link_path_walk	Use-After-Free
BUG: unable to handle kernel paging request in __inet_check_established	Page Fault
KASAN: null-ptr-deref Read in ata_pio_sector	Null ptr deref
WARNING in ip_recv_error	Reachable Warning
WARNING in remove_proc_entry	Reachable Warning
KASAN: null-ptr-deref Read in ip6gre_exit_batch_net	Null ptr deref
KASAN: slab-out-of-bounds Write in __register_sysctl_table	Heap overflow
KASAN: use-after-free Write in skb_release_data	Use-After-Free
KASAN: invalid-free in ptlock_free	Double free
Kernel BUG at lib/list_debug.c:LINE!	Reachable Assertion
INFO: trying to register non-static key in __handle_mm_fault	Reachable INFO
KASAN: use-after-free Read in vhost-chr_write_iter	Use-After-Free
BUG: soft lockup in vmemdup_user	Soft lockup
KASAN: use-after-free Read in rds_tcp_accept_one	Use-After-Free
WARNING in sg_rq_end_io	Reachable Warning
BUG: soft lockup in snd_virmidi_output_trigger	Soft lockup
KASAN: null-ptr-deref Read in smc_ioctl	Null ptr deref
KASAN: null-ptr-deref Write in binderf_update_page_range	Null ptr deref
WARNING in port_delete	Reachable Warning
KASAN: null-ptr-deref in inode_permission	Null ptr def

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

Race bugs	Syzkaller			Razzer		
	# of exec	Time	Found	# of exec	Time	Found
CVE-2016-8655	29 M	10 hrs	X	1,170 K	26 min	✓
CVE-2017-17712	37 M	10 hrs	X	807 K	18 mins	✓
CVE-2017-2636	5 M	10 hrs	X	246 K	7 mins	✓

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
threeearcat@gmail.com

KAIST School of Computing

SEOUL NATIONAL UNIVERSITY
Department of Electrical and Computer Engineering

PURDUE UNIVERSITY