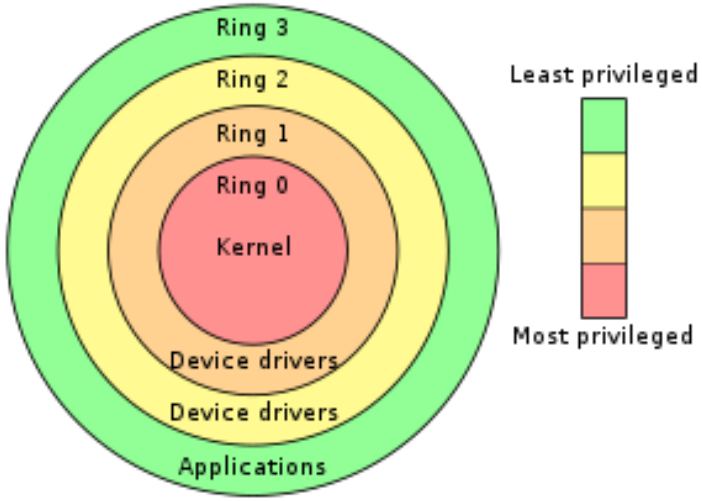
리눅스 커널 취약점 탐지 및 공격

Byoungyoung Lee (이병영)
Seoul National University (SNU)
byoungyoung@snu.ac.kr
https://compsec.snu.ac.kr

Kernel Vulnerability



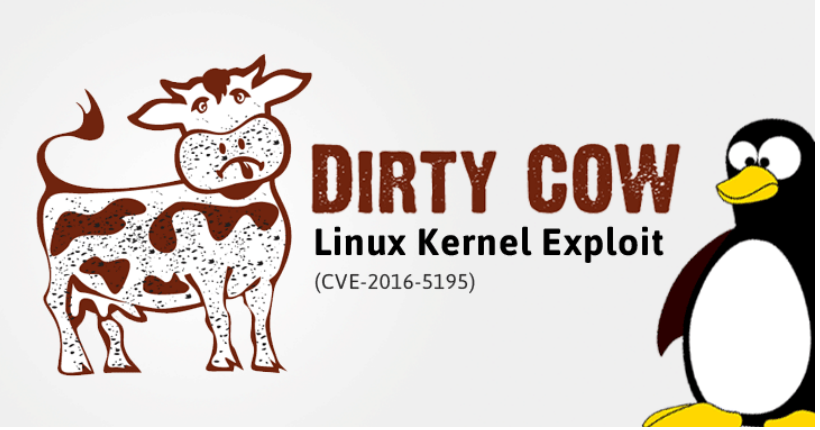
https://en.wikipedia.org/wiki/Principle_of_least_privilege

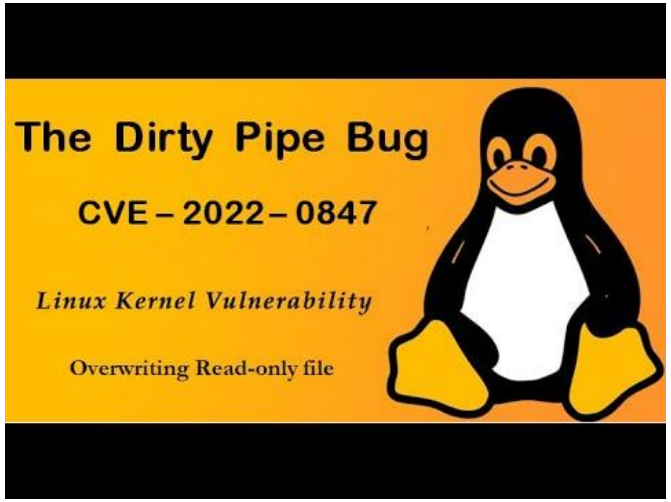
Exploiting kernel vulnerability, attacker can subvert the entire system. Identifying and patching kernel vulnerabilities is very important!

Linux Kernel and Security

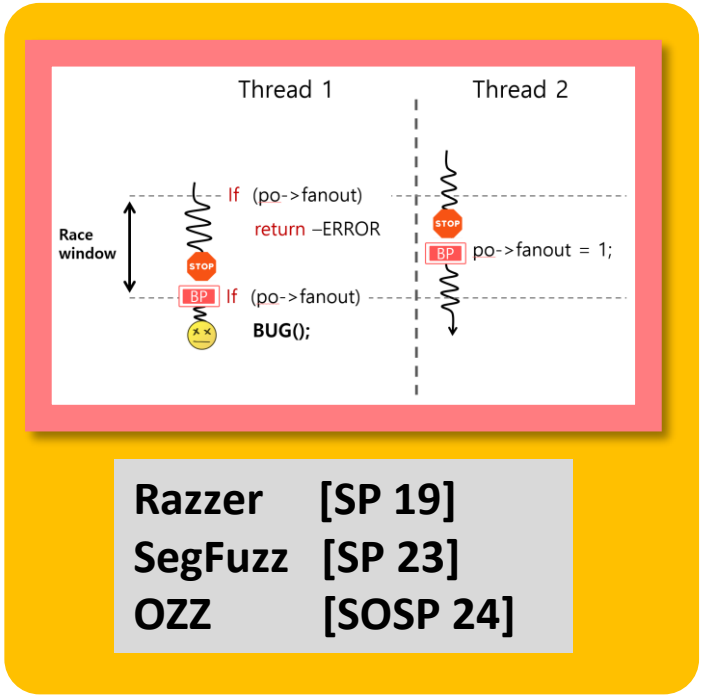
- **Linux kernel is the backbone of modern computing infrastructures**
  - Desktop
  - Cloud host and guest VM
  - Android
  - Various IoT devices
  - Many more…

Examples of Linux Kernel Vulnerabilities



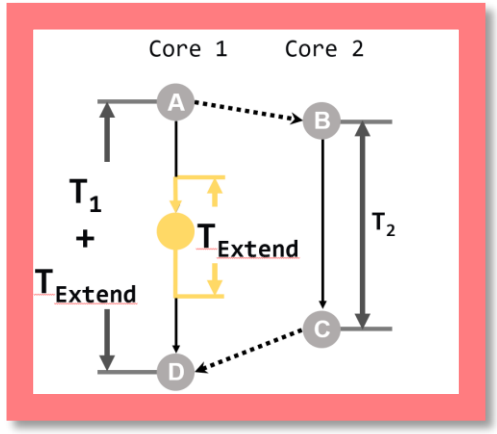


Our Research on Linux Kernel Security



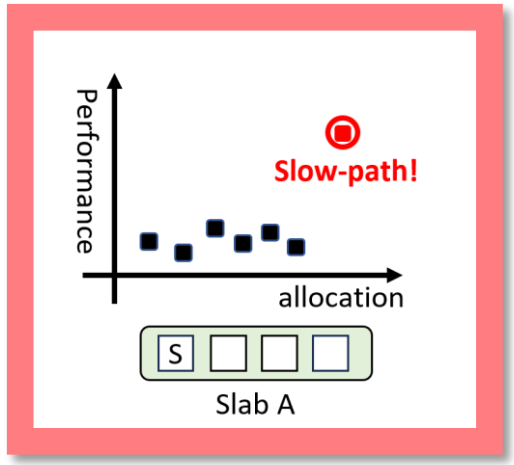
Razzer [SP 19]
SegFuzz [SP 23]
OZZ [SOSP 24]

Fuzzing to find race vulnerabilities



ExpRace [Security 21]

Exploiting race vulnerabilities

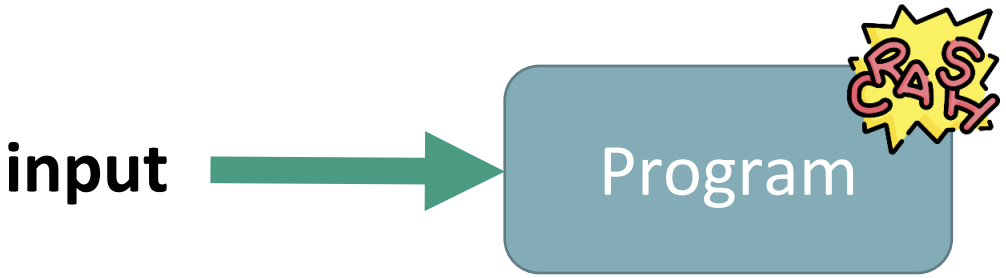


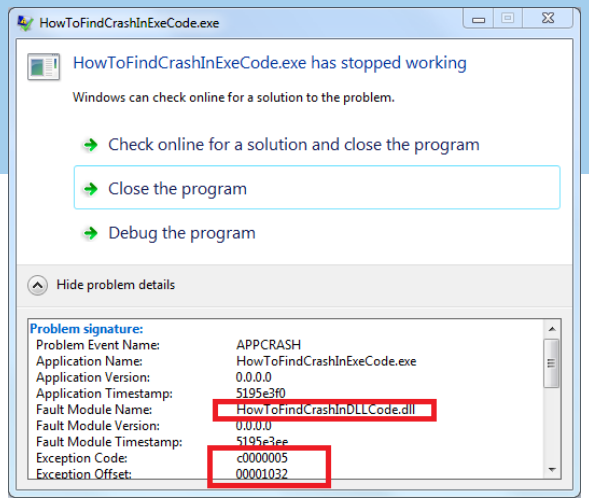
Pspray [Security 23]

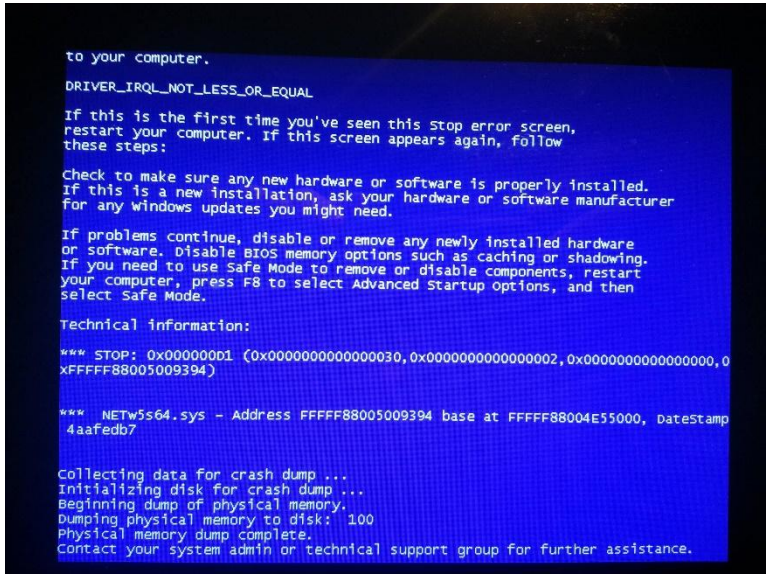
Exploiting use-after-free

What programs do

Vulnerability

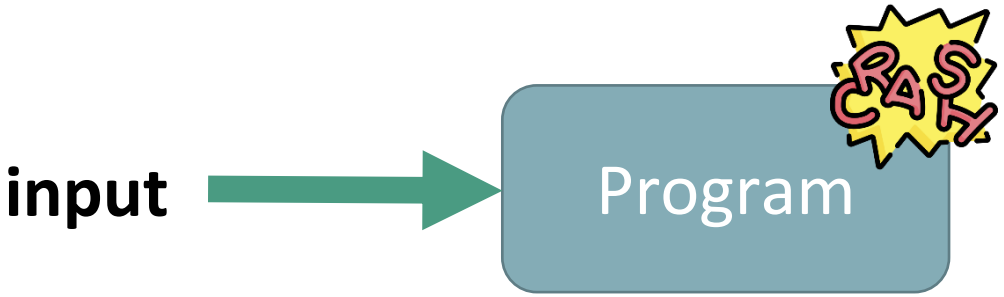






For some input, the program acts weird (e.g., crash, hanging, etc.)

Fuzzing: Random Testing!



Fuzzing 101

#1. Randomly pick input $x^R \in X$

#2. Run the program using x^R
- If something bad occurs, X^R induces a vulnerability

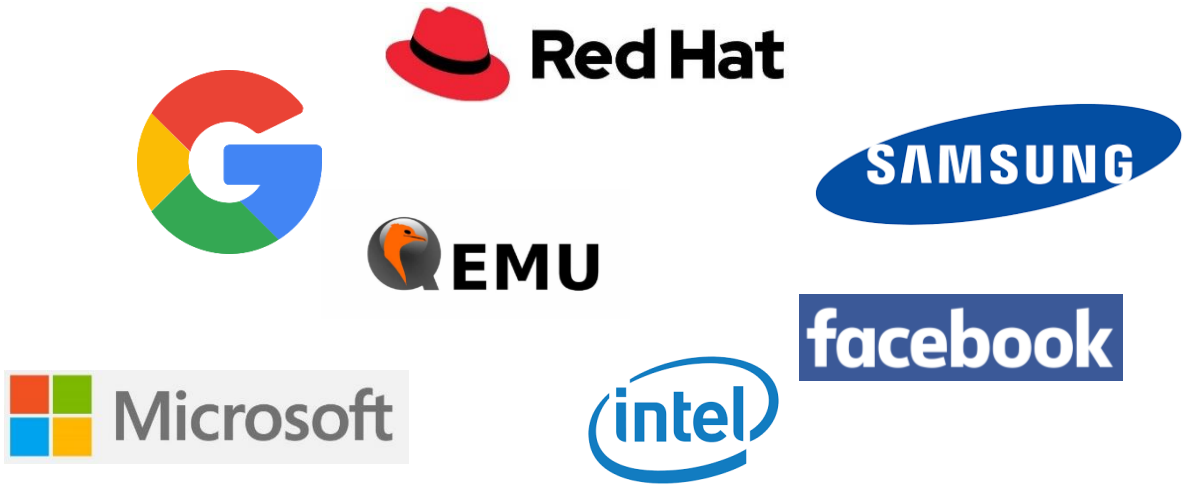
#3. Go back to step #1

Why is fuzzing so popular today?

Easy-to-implement

Effective

Real-world Practices of Fuzzing



Almost all big-techs fuzz their products

Race condition

- **Race conditions**
  - 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 condition occurs, the computational results may vary depending on the execution order.

Race condition: A simple example

```
a = 0;

If (a) {
   b = 0;
} else {
   b = 1;
}
```

Thread 1

```
a = 1;
```

Thread 2

b will be 1

```
a = 0;

If (a) {
   b = 0;
} else {
    b = 1;
}
```

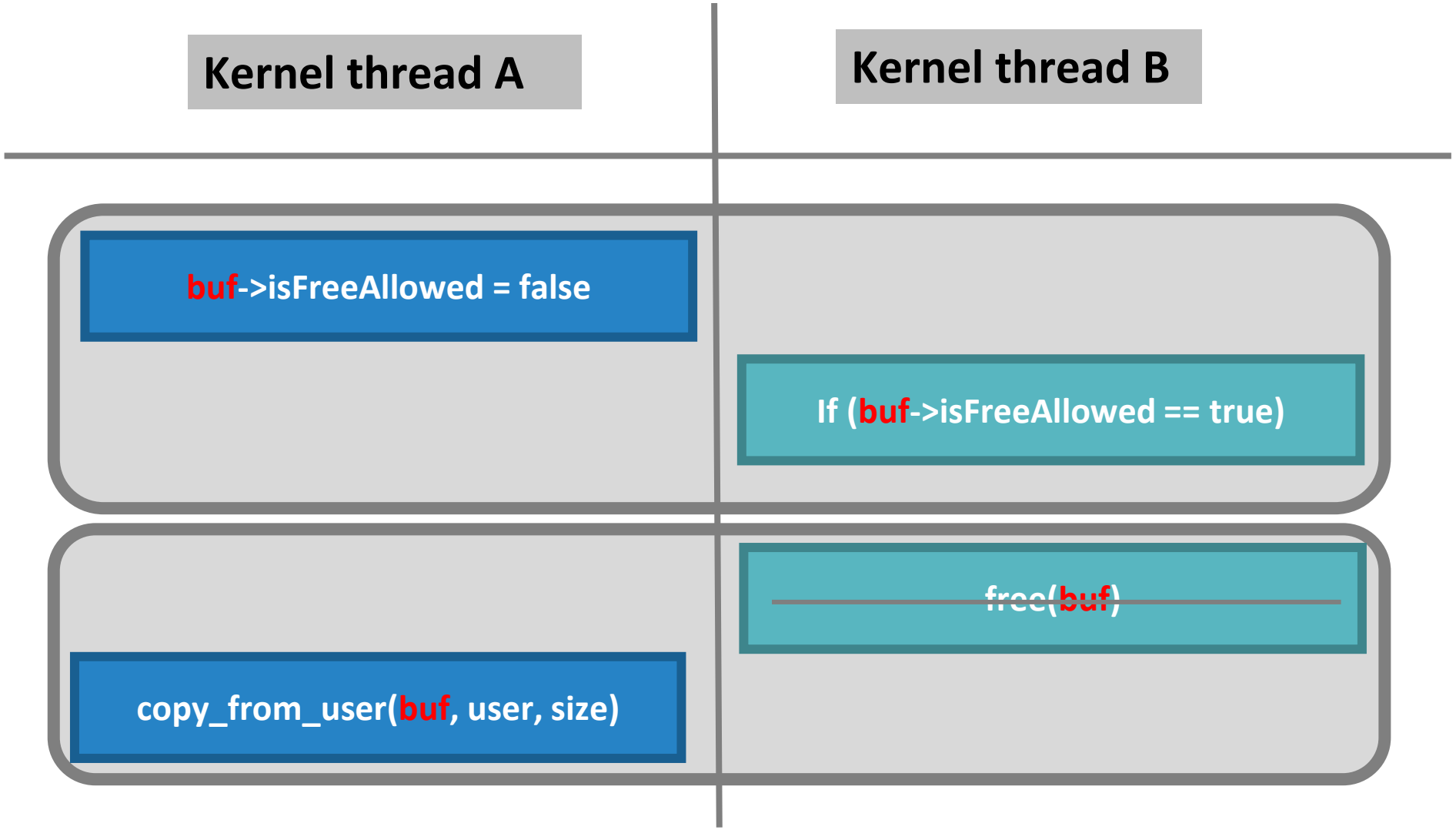
Thread 1

```
a = 1;
```

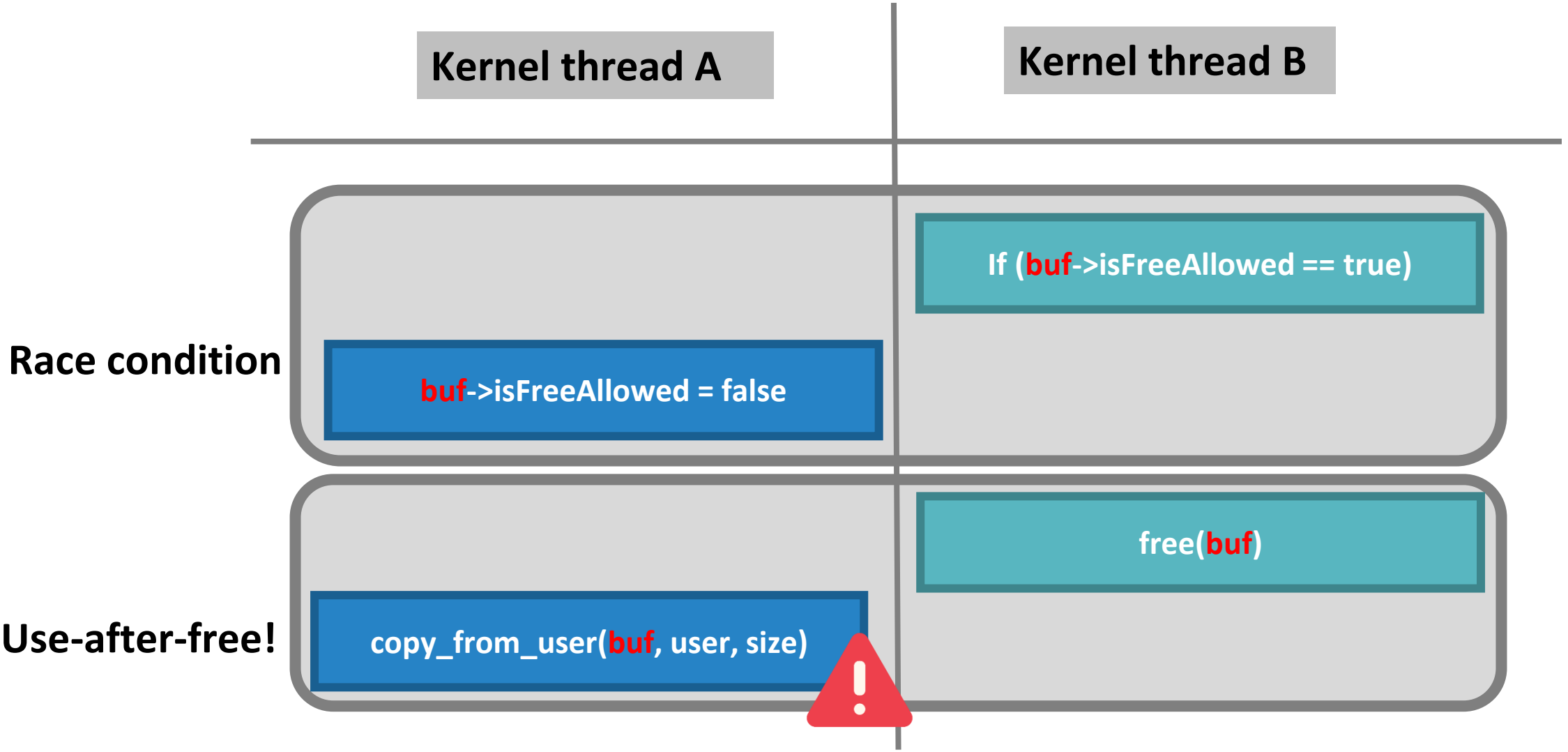
Thread 2

b will be 0

Race condition: CVE-2019-2025

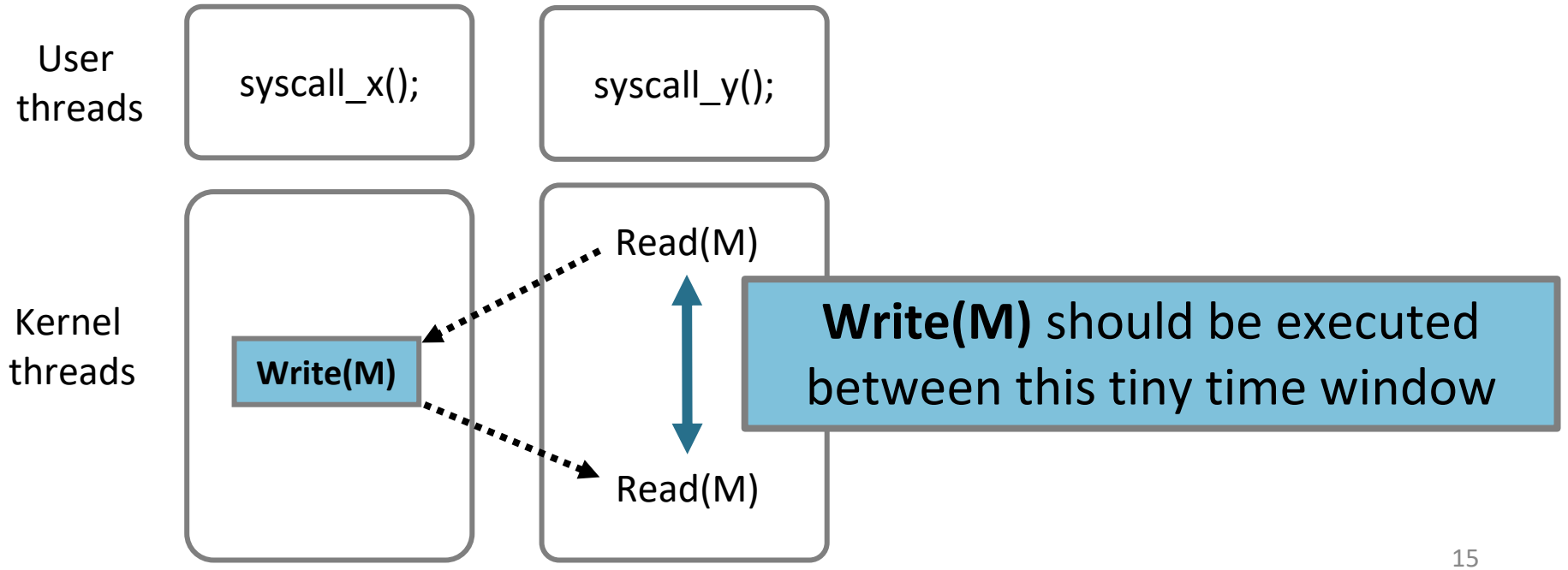


Race condition: CVE-2019-2025



Challenges: Kernel Races and Fuzzer

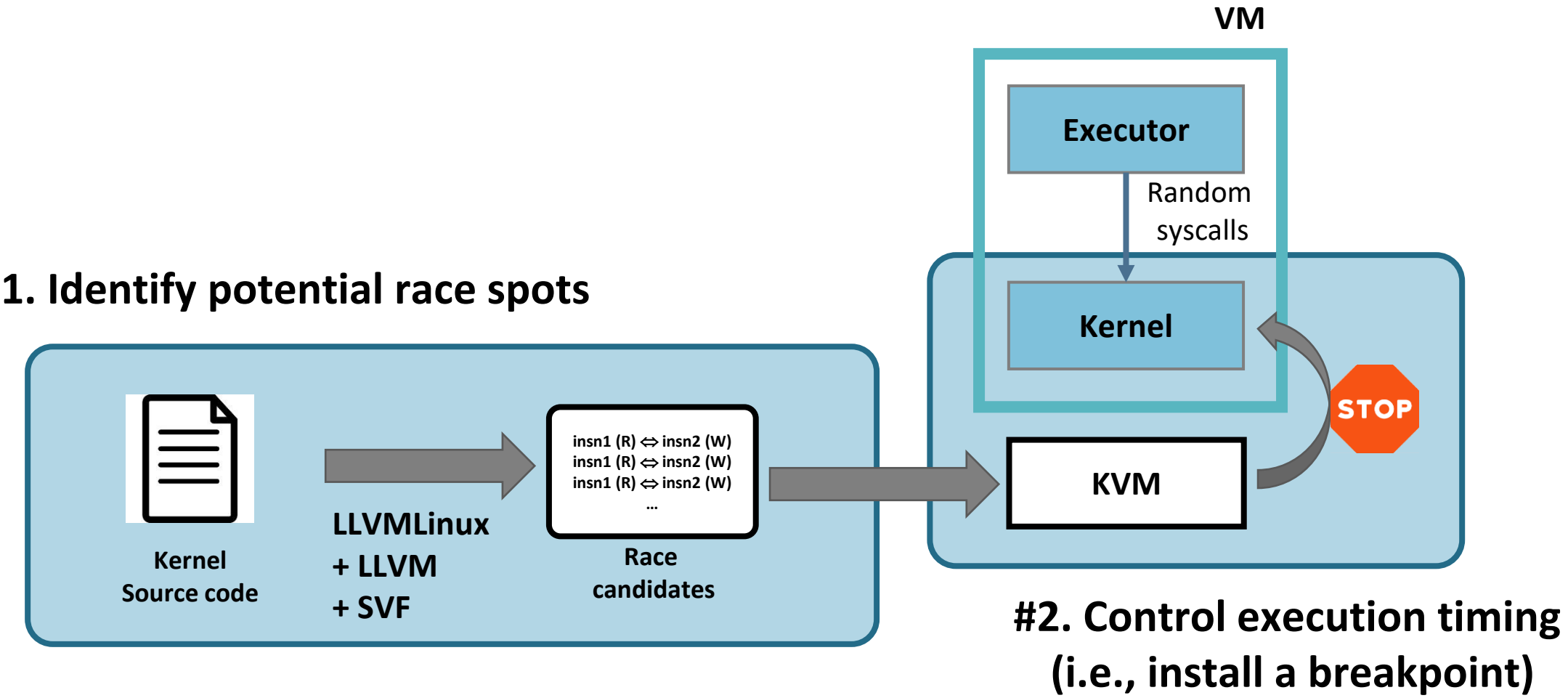
- **Traditional fuzzers are not efficient to discover races**
  - Execution timing (due to thread interleaving) matters to find races
  - Traditional fuzzers cannot take care of such timing



Our Approach: Razzer

- **Razzer: A new fuzzer to find RACE!**
  - #1. Static analysis to find potential race spots
    - Points-to analysis over an entire kernel
  - #2. For each potential race spots, setup per-core breakpoint
    - Deterministic kernel execution w.r.t. a given race spot
    - Run the kernel on modified hypervisor (QEMU/KVM)
  - #3. Check if a race truly occurs
    - Prioritize a truly racing pair to be further fuzzed later

Razzer: Kernel fuzzer for race conditions



Results

30 new races!
16 are fixed!

Use-after-free

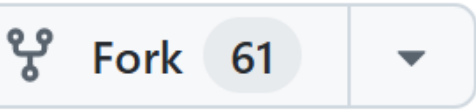
Heap overflow

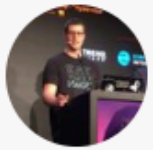
Double free

Kernel crash summary	Crash type	Kernel version	Kernel subsystem	Confirmed	Patch submitted	Fixed
KASAN: slab-out-of-bounds write in tty_insert_flip_string_flag	Use-After-Free	v4.8	drivers/tty/	✓	✓	✓
WARNING in __static_key_slow_dec	Reachable Warning	v4.8	net/	✓		
Kernel BUG at net/packet/af_packet.c:LINE!	Reachable Assertion	v4.16-rc3	net/packet/	✓	✓	✓
WARNING in refcount_dec	Reachable Warning	v4.16-rc3	net/packet/	✓	✓	✓
unable to handle kernel paging request in snd_seq_oss_readq_puts	Page Fault	v4.16	sound/core/seq/oss/	✓	✓	✓
KASAN: use-after-free Read in loopback_active_get	Use-After-Free	v4.16	sound/drivers/	✓	✓	✓
KASAN: null-ptr-deref Read in rds_ib_get_mr	Null ptr deref	v4.17-rc1	net/rdma/	✓(assisted Syzkaller)	✓	✓
KASAN: null-ptr-deref Read in list_lru_del	Null ptr deref	v4.17-rc1	fs/			
BUG: unable to handle kernel NULL ptr dereference in corrupted	Null ptr deref	v4.17-rc1	net/sctp/			
KASAN: use-after-free Read in nd_jump_root	Use-After-Free	v4.17-rc1	fs/	✓	✓	✓
KASAN: use-after-free Read in link_path_walk	Use-After-Free	v4.17-rc1	fs/	✓	✓	✓
BUG: unable to handle kernel paging request in __inet_check_established	Page Fault	v4.17-rc1	net/ipv4/			
KASAN: null-ptr-deref Read in ata_pio_sector	Null ptr deref	v4.17-rc1	net/drivers/ata/			
WARNING in ip_recv_error	Reachable Warning	v4.17-rc1	net/	✓	✓	✓
WARNING in remove_proc_entry	Reachable Warning	v4.17-rc1	net/sunrpc/			
KASAN: null-ptr-deref Read in ip6gre_exit_batch_net	Null ptr deref	v4.17-rc1	net/ipv6/			
KASAN: slab-out-of-bounds Write in __register_sysctl_table	Heap overflow	v4.17-rc1	net/ipv6/			
KASAN: use-after-free Write in skb_release_data	Use-After-Free	v4.17-rc1	net/core/			
KASAN: invalid-free in ptlock_free	Double free	v4.17-rc1	mm/			
Kernel BUG at lib/list_debug.c:LINE!	Reachable Assertion	v4.17-rc1	drivers/infiniband/			
INFO: trying to register non-static key in __handle_mm_fault	Reachable INFO	v4.17-rc1	mm/			
KASAN: use-after-free Read in vhost-chr_write_iter	Use-After-Free	v4.17-rc1	drivers/vhost/	✓	✓	✓
BUG: soft lockup in vmemdup_user	Soft lockup	v4.17-rc1	net/			
KASAN: use-after-free Read in rds_tcp_accept_one	Use-After-Free	v4.17-rc1	net/rds/			
WARNING in sg_rq_end_io	Reachable Warning	v4.17-rc1	drivers/scsi/			
BUG: soft lockup in snd_virmidi_output_trigger	Soft lockup	v4.18-rc3	sound/core/seq/	✓(assisted Syzkaller)	✓	✓
KASAN: null-ptr-deref Read in smc_ioctl	Null ptr deref	v4.18-rc3	net/smc/	✓	✓	✓
KASAN: null-ptr-deref Write in binderf_update_page_range	Null ptr deref	v4.18-rc3	drivers/android/	✓	✓	
WARNING in port_delete	Reachable Warning	v4.18-rc3	sound/core/seq/	✓(assisted Syzkaller)		
KASAN: null-ptr-deref in inode_permission	Null ptr def	v4.18-rc3	fs/	✓	✓	✓

Razzer is open-sourced!

compsec-snu / **razzer**

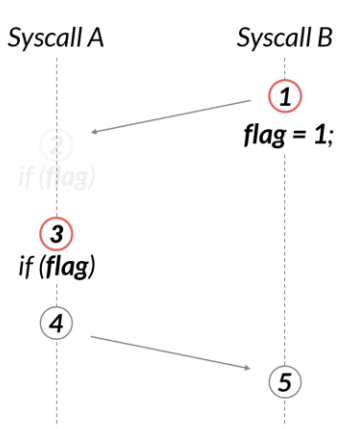




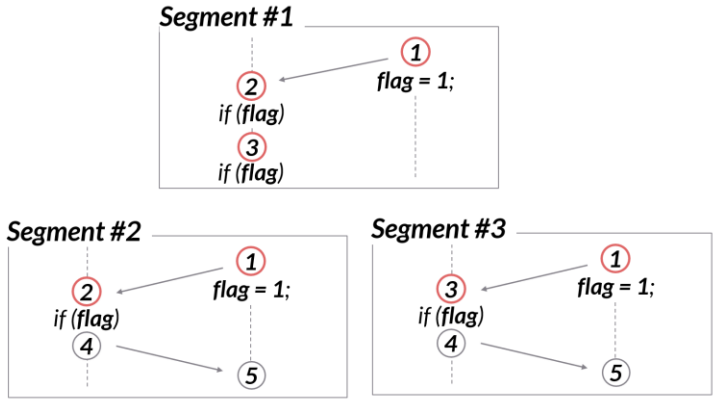
Yoav Alon @yoavalon · May 28, 2019

The source code for **Razzer** - race condition **fuzzer** for the linux kernel was released. It's one of the most innovative fuzzers I've seen in recent years. It uses a clever static analysis to gather race candidates and a fork of syzkaller & hv to test them.

Our follow-up work: SegFuzz [SP 23]



Whole thread interleaving



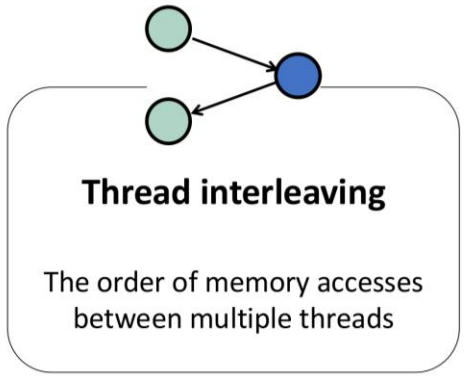
Interleaving segments
(each contains at most 4 inst.)

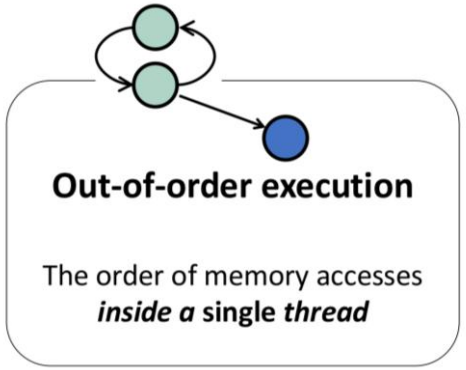
general protection fault in vmci_host_poll
KASAN: use-after-free Read in cfusbl_device_notify
KASAN: use-after-free Read in slcan_receive_buf
general protection fault in cttimeout_net_exit
KASAN: use-after-free Read in raw_notifier_call_chain
INFO: task hung in blk_trace_remove
INFO: task hung in blk_trace_setup
kernel BUG in pfkey_send_acquire
general protection fault in add_wait_queue_exclusive
KASAN: use-after-free Read in slip_ioctl
general protection fault in add_wait_queue
WARNING in isotp_tx_timer_handler
KASAN: use-after-free Read in snd_pcm_plug_read_transfer
Kernel BUG in find_lock_entries
KASAN: use-after-free Read in tcp_write_timer_handler
KASAN: use-after-free Read in event_sched_out
general protection fault in soft_cursor
KASAN: use-after-free Read in perf_event_groups_insert
BUG: unable to handle kernel paging request in usb_start_wait_urb
BUG: unable to handle kernel paging request in __kernfs_new_node
general protection fault in raw_seq_start

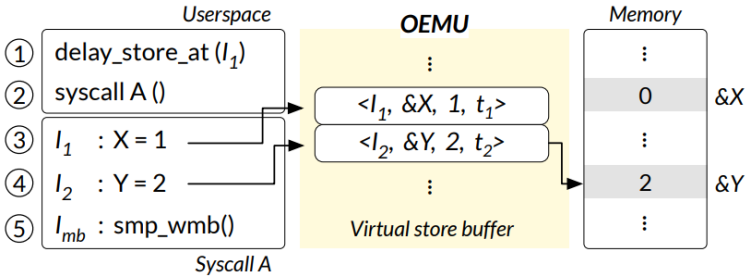
Developed new thread interleaving coverage
➔ Efficiently exploring unexplored thread interleavings

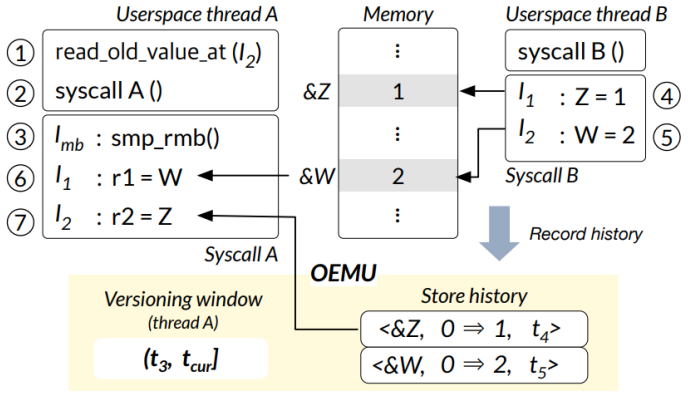
21 new race vulnerabilities!

Our follow-up work: OZZ [SOSP 24]





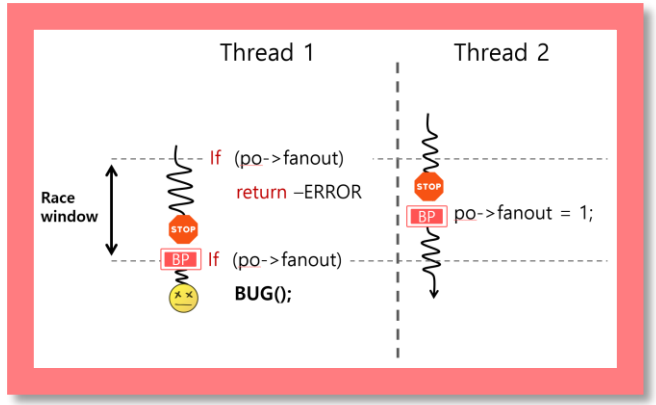




Challenge:
- Complex non-deterministic behaviors when identifying out-of-order bugs

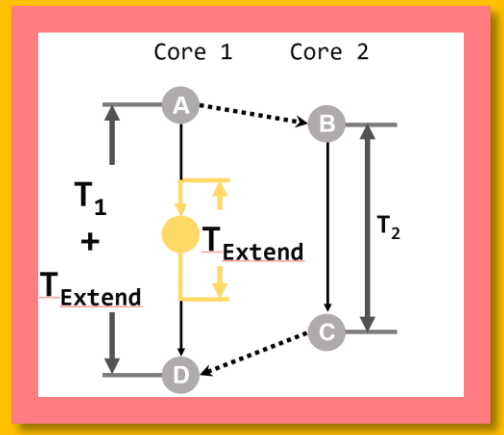
Solution:
- Emulate load/store to be deterministic
- Fuzzing with SegFuzz [SP 23]

Our Research on Linux Kernel Security



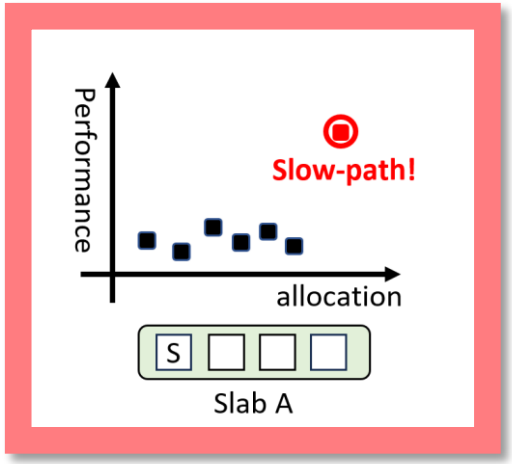
Razzer [SP 19]
SegFuzz [SP 23]
OZZ [SOSP 24]

Fuzzing to find race vulnerabilities



ExpRace [Security 21]

Exploiting race vulnerabilities



Pspray [Security 23]

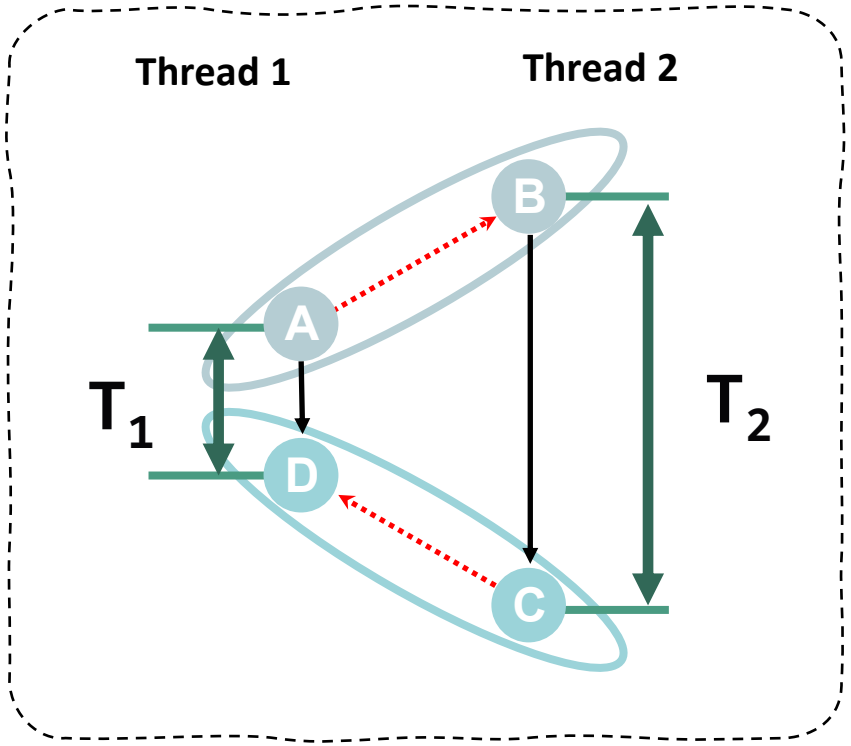
Exploiting use-after-free

Problem : Multi-Variable Race

Race CVE	Does brute-force work?
CVE-2019-11486	✔
CVE-2017-7533	✔
CVE-2017-2636	✔
CVE-2016-8655	✔
CVE-2019-6974	✗
CVE-2019-2025	✗
CVE-2019-1999	✗
CVE-2017-15265	✗

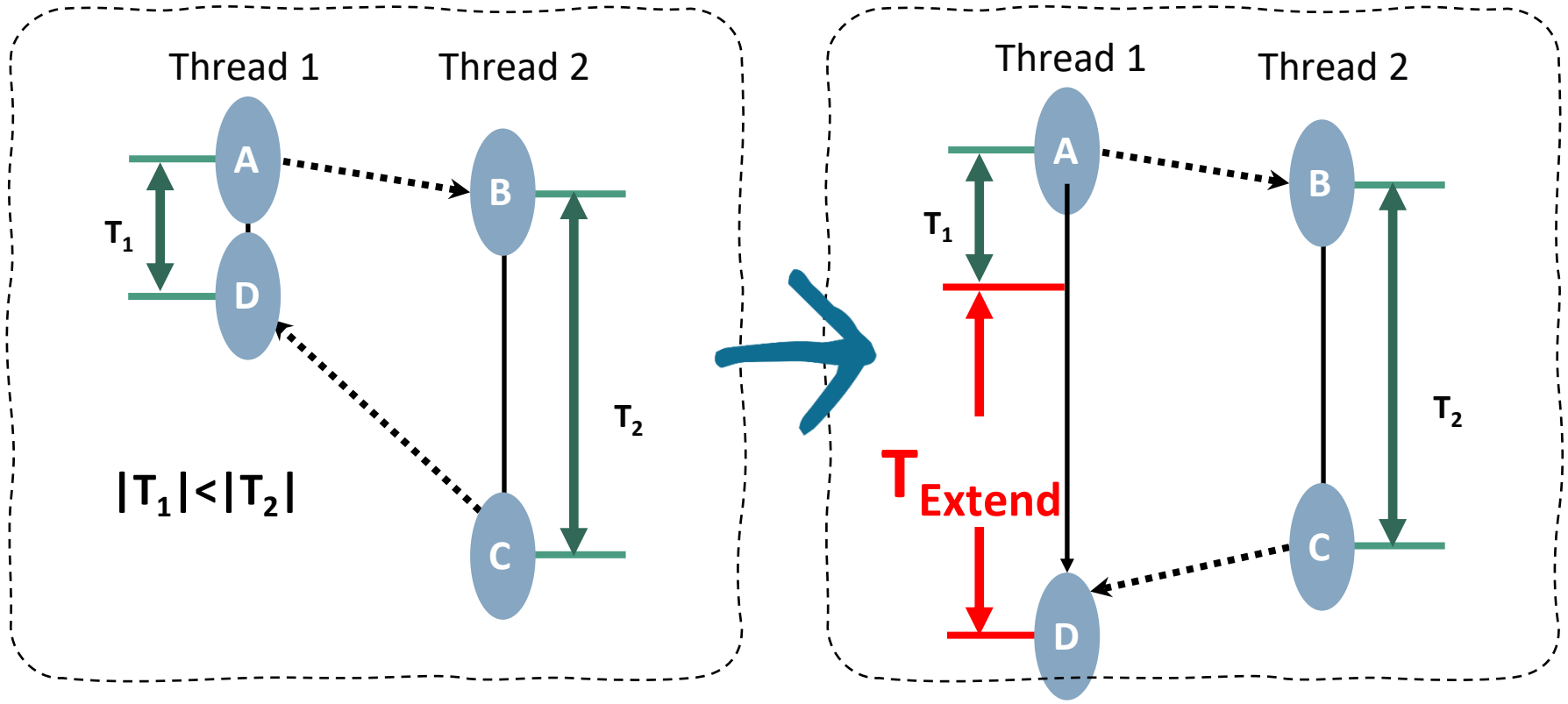
- Some races are exploitable with brute-forcing
- However, some others are NOT.

Problem : Multi-Variable Race

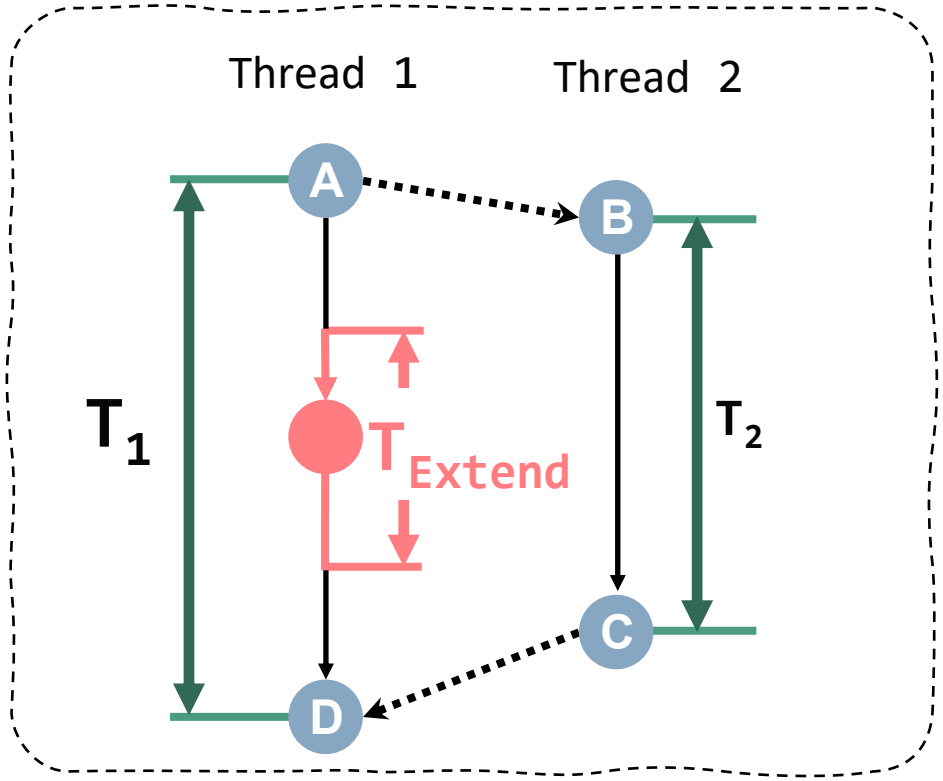


- Some races are (nearly) impossible to exploit
  - If $T_1 < T_2$

Goal: Make Exploit Work Again!

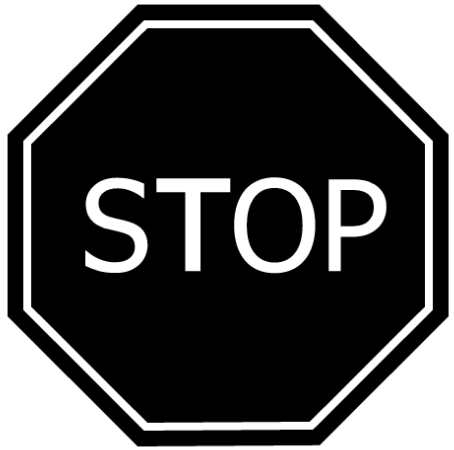


Idea: How to extend the time window?

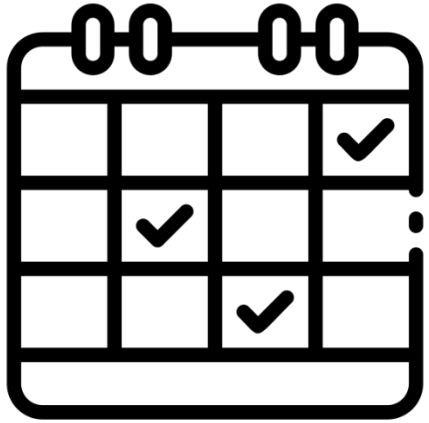


Preempting the thread!

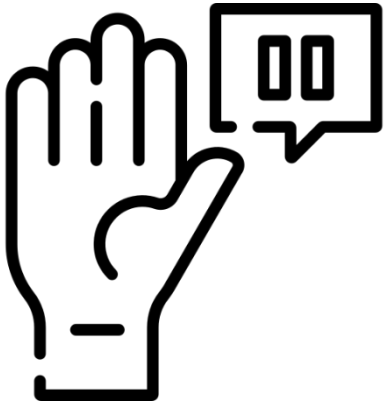
Background: Kernel Preemption



Kernel breakpoint



Kernel thread schedule



Interrupt

Challenge: Users cannot preempt!

Kernel breakpoint

Kernel thread schedule

Interrupt

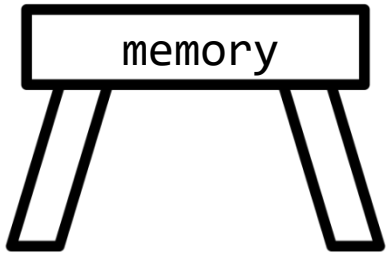
Back to OS basics:

Users cannot control kernel's preemption

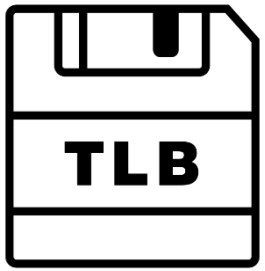
Idea again:
User-controlled Preemption through Interrupts

We found four methods to indirectly raise interrupts through syscalls

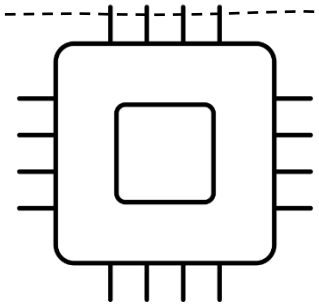
- **Reschedule** interrupt



- **Membarrier** interrupt

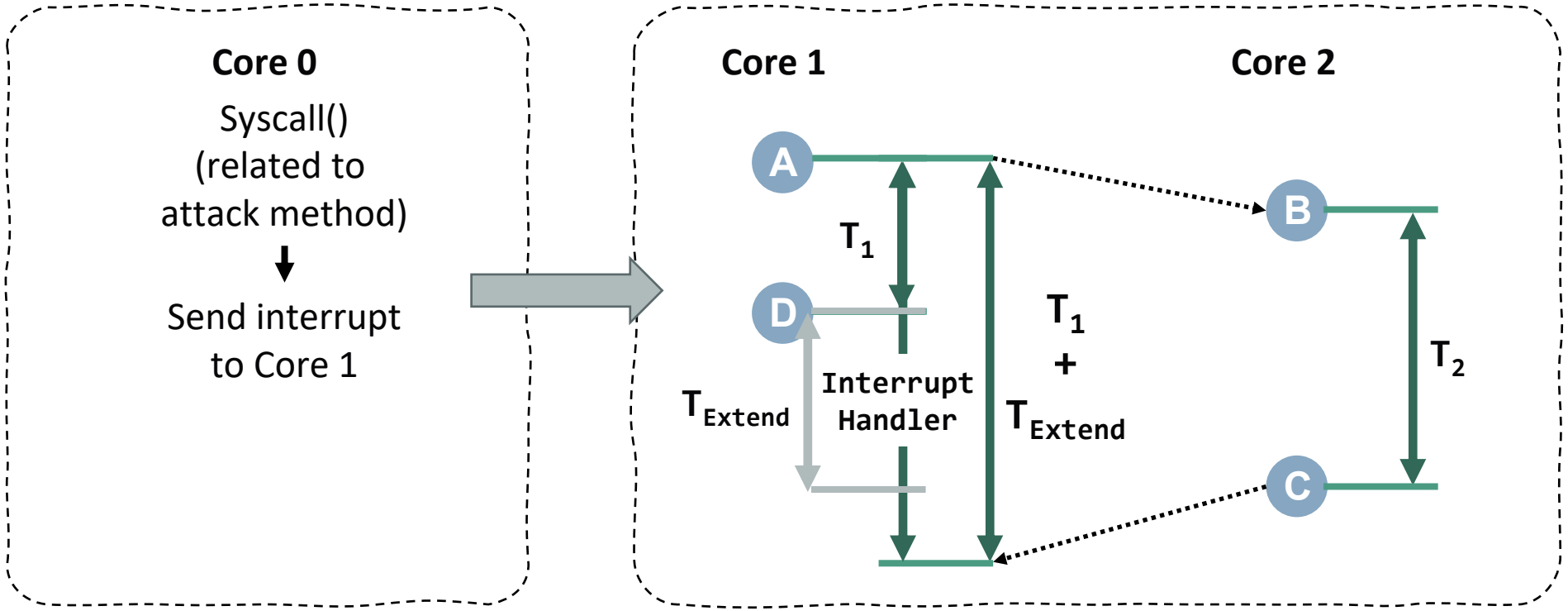


- **TLB shootdown** interrupt



- **Hardware** interrupt

Our Approach : Interrupt



Enlarging race window through syscalls and its follow-up interrupts

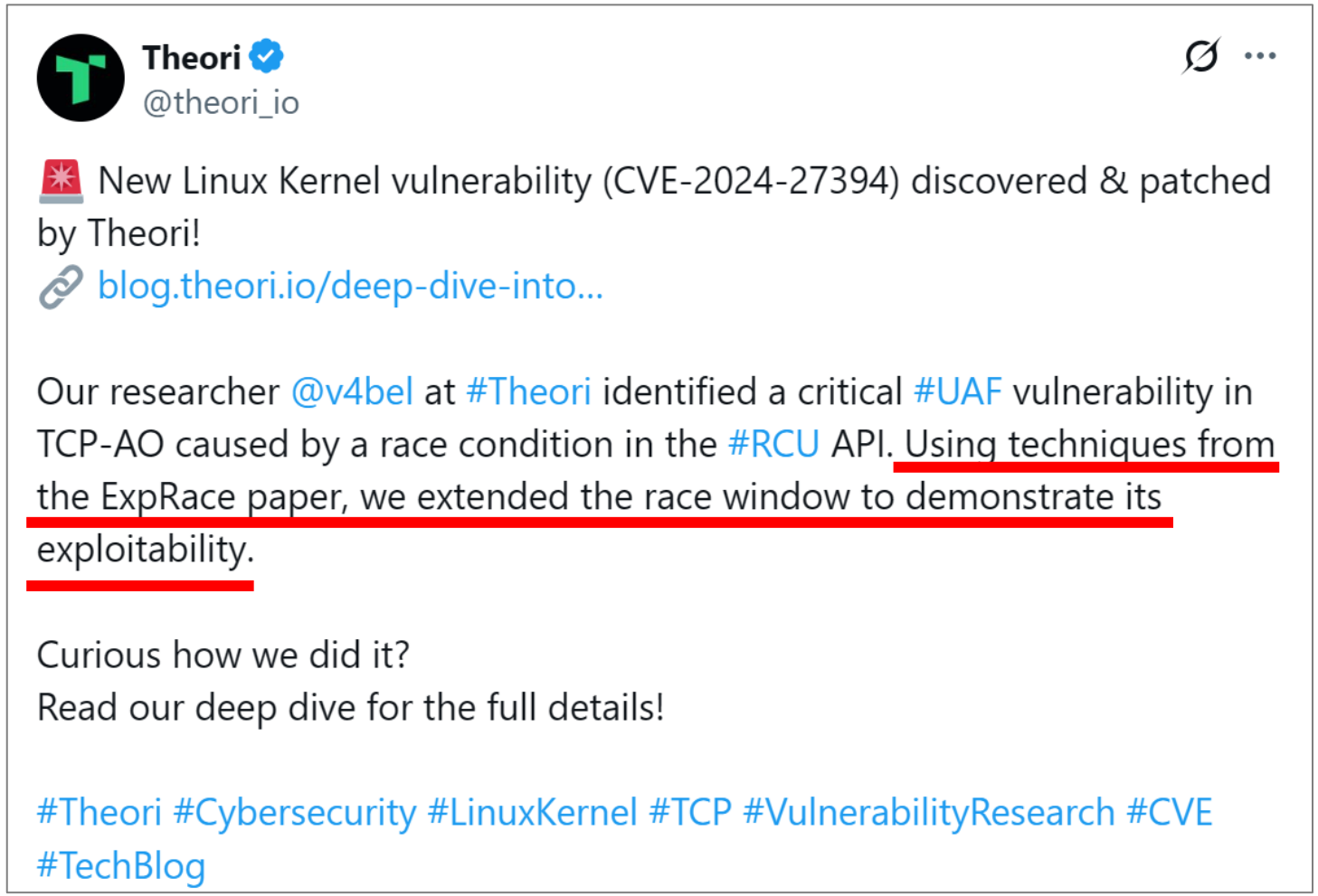
Real-World Races in Linux

Vulnerability	Baseline	Reschedule	membarrier	TLB shootdown	HW interrupt
CVE-2019-6974	✗	✗	✗	✗	✓
CVE-2019-2025	✗	✓	✓	✓	✓
CVE-2019-1999	✗	✗	✗	✓	✓
CVE-2017-15265	✗	✓	✓	✓	✓
11eb85ec…	✗	✗	✗	✓	✓
1a6084f8…	✗	✗	✗	✓	✓
20f2e4c2…	✗	✗	✗	✓	✓
484298f…	✗	✓	✓	✓	✓
da1b9564…	✗	✗	✗	✗	✓
e20a2e9c…	✗	✗	✗	✓	✓

✗ denotes exploitation has failed for given 24 hours

All 10 cases were successfully exploited with ExpRace!

After the publication of ExpRace



Theori ✓
@theori_io

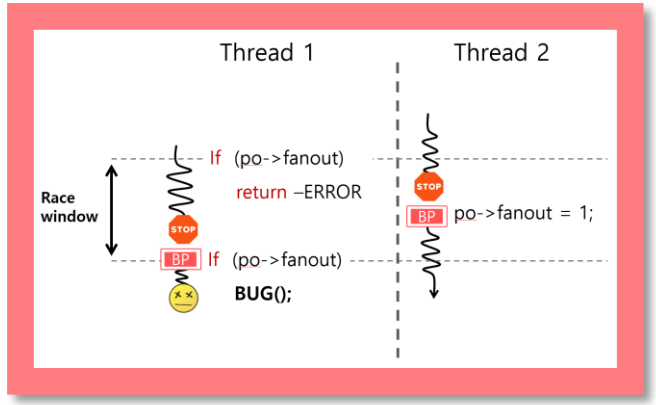
🚨 New Linux Kernel vulnerability (CVE-2024-27394) discovered & patched by Theori!
🔗 blog.theori.io/deep-dive-into…

Our researcher @v4bel at #Theori identified a critical #UAF vulnerability in TCP-AO caused by a race condition in the #RCU API. Using techniques from the ExpRace paper, we extended the race window to demonstrate its exploitability.

Curious how we did it?
Read our deep dive for the full details!

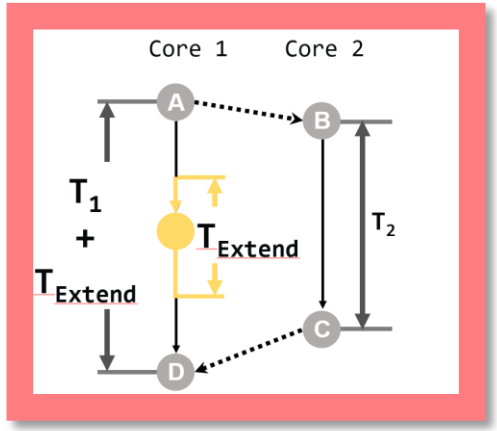
#Theori #Cybersecurity #LinuxKernel #TCP #VulnerabilityResearch #CVE #TechBlog

Our Research on Linux Kernel Security



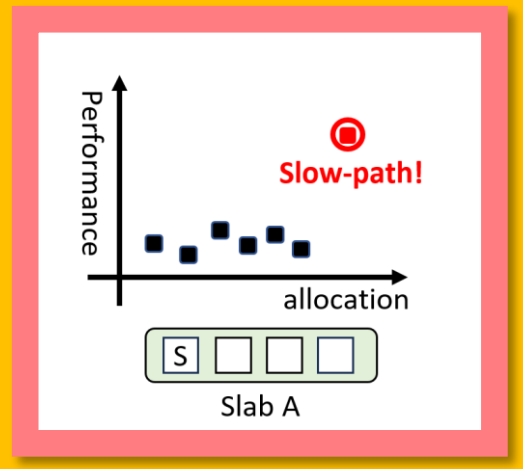
Razzer [SP 19]
SegFuzz [SP 23]
OZZ [SOSP 24]

Fuzzing to find race vulnerabilities



ExpRace [Security 21]

Exploiting race vulnerabilities



Pspray [Security 23]

Exploiting use-after-free

Summary of PSpray

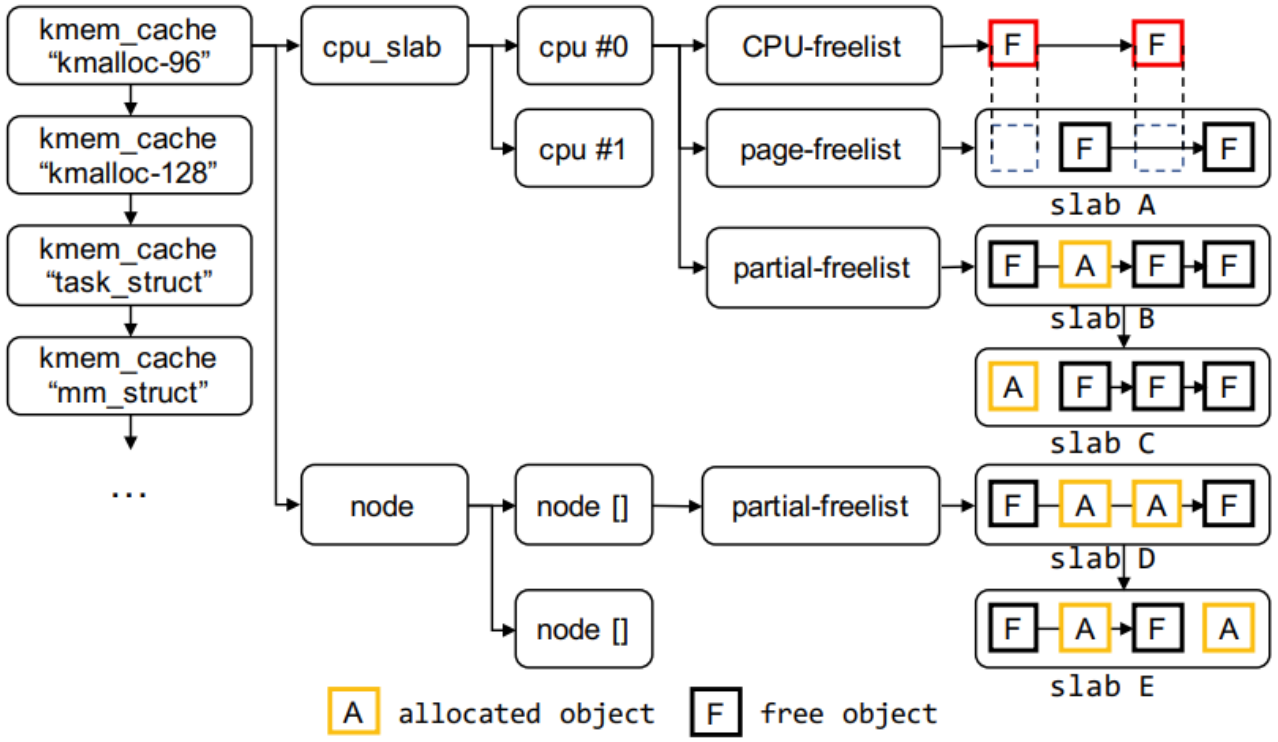
Motivation

- **Reliable kernel exploitation** is important for attackers.
- Reliable use-after-free exploitation is challenging.
- Why? **Attackers do not know the status of kernel heap.**

Our finding

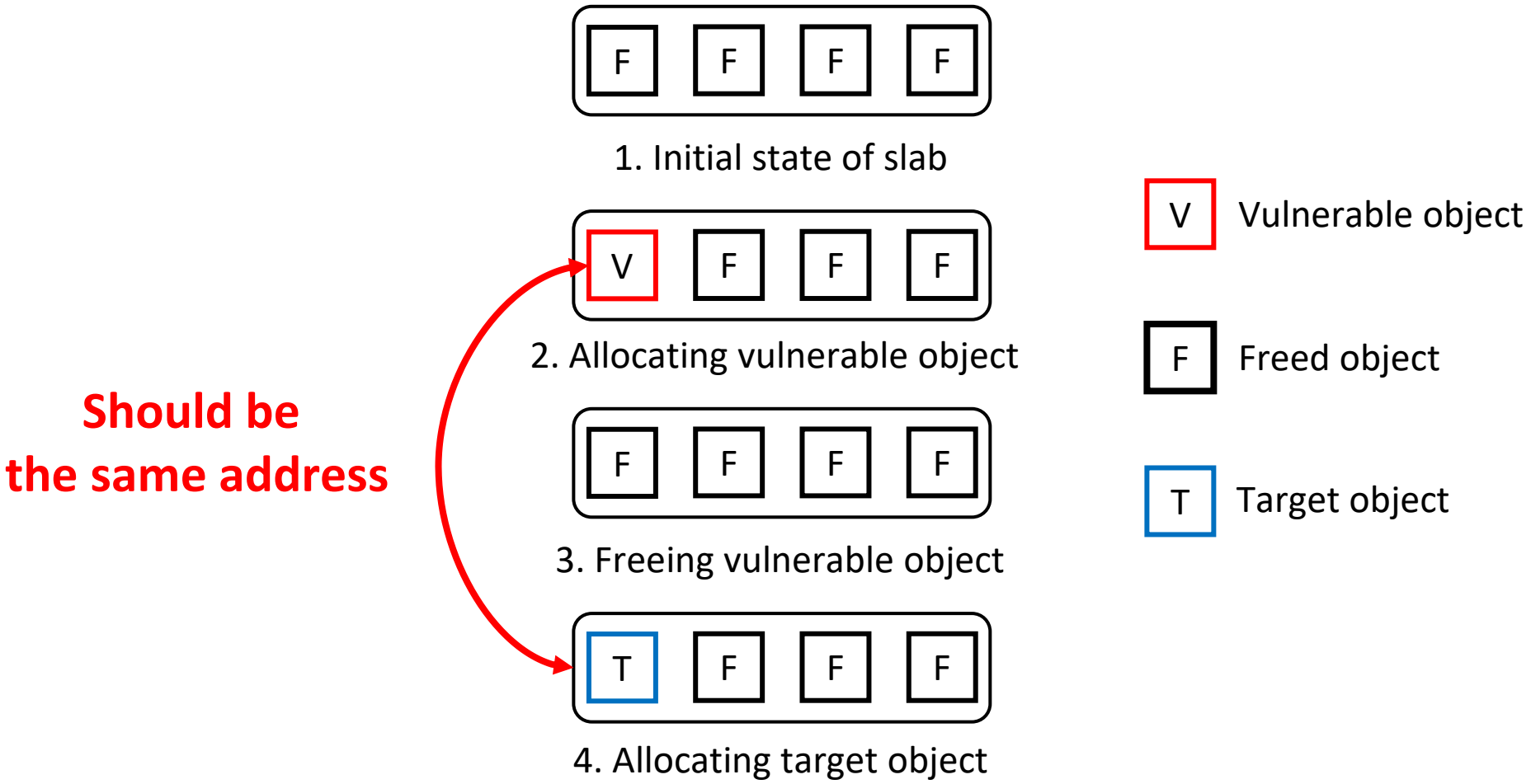
- **Timing side-channel attacks** to **learn the status of kernel heap.**
- Attackers can launch reliable kernel exploitation.

Slab: Kernel's memory allocator



Slab's allocation process is a bit complicated ☹

Challenges in exploiting use-after-free



The **vuln** and **target** objects must be placed at the **same address** to exploit.

Idea: Timing Side-Channel

- Kernel implements **multiple allocation paths**:
  - Fast, medium, slow paths depending on the heap status
- We found **timing side-channels to identify allocation path**!
  - By measuring the time taken to allocate,

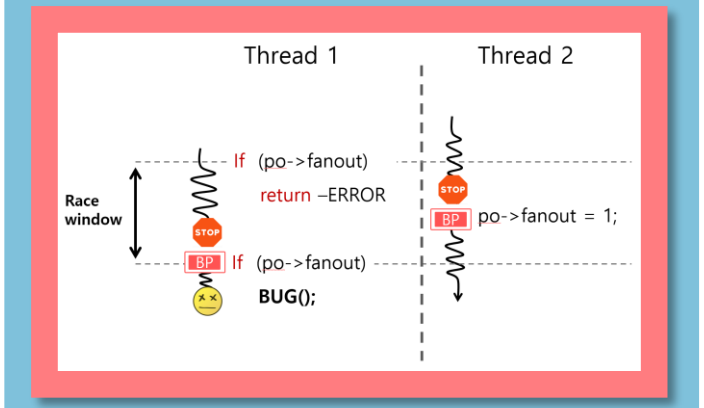
    we know which allocation path has been taken.
- This allows to place vulnerable and target objects **at the same address!**

Exploit success rate after Pspray

Vulns	Type	# of alloc	Baseline (Success rate)	Pspray (Success rate)
CVE-2019-2215	UAF	2	93.28%	**100.0%**
CVE-2018-6555	UAF	13	63.50%	**99.94%**
83bec2...	UAF	8	13.70%	**98.16%**
77e2cf...	UAF	1	95.74%	**100.0%**
CVE-2017-6074	DF	4	80.64%	**100.0%**
6b8d6b...	DF	1	96.28%	**99.98%**

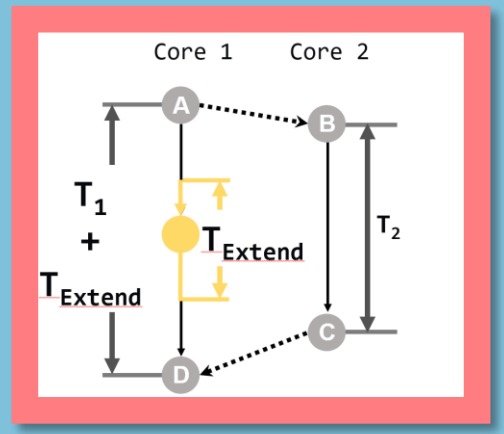
Conclusion

- **Linux kernel security is important!**



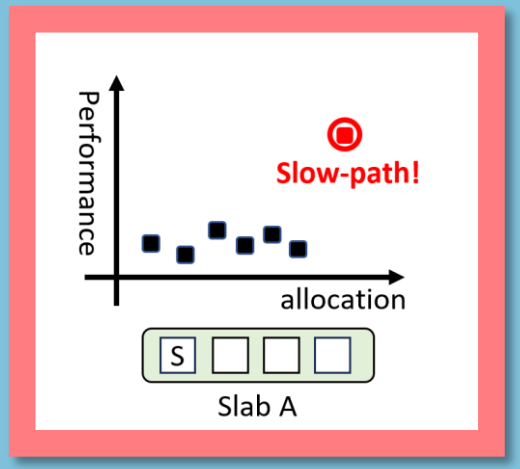
Razzer [SP 19]
SegFuzz [SP 23]
OZZ [SOSP 24]

Fuzzing to find race vulnerabilities



ExpRace [Security 21]

Exploiting race vulnerabilities



Pspray [Security 23]

Exploiting use-after-free

감사합니다

Byoungyoung Lee
Seoul National University
byoungyoung@snu.ac.kr
https://compsec.snu.ac.kr