Phantom Attack:
Evading System Call Monitoring

Rex Guo, Ph.D.
Junyuan Zeng, Ph.D.

@Xiaofei_Rex

jzeng04 *NOSPAM* gmail DOT com

An Incident - An Attacker’s View

web app RCE on joe-box and executed a reverse shell

An Incident - An Attacker's View

web app RCE on joe-box and executed a reverse shell

Privilege escalation using sudo CVE-2021-3156

An Incident - An Attacker’s View

web app RCE on joe-box and executed a reverse shell

Privilege escalation using sudo CVE-2021-3156

read /etc/shadow

An Incident - An Attacker's View

web app RCE on joe-box and executed a reverse shell

Privilege escalation using sudo CVE-2021-3156

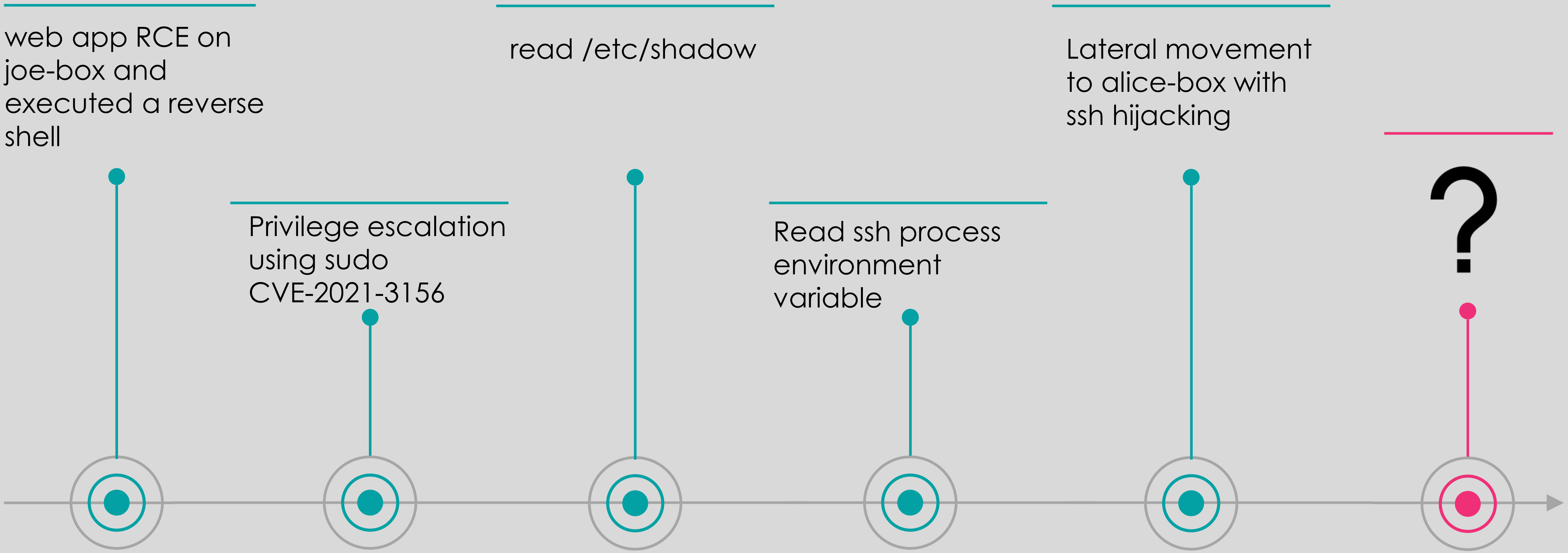
read /etc/shadow

Read ssh process environment variable

An Incident - An Attacker’s View

- web app RCE on joe-box and executed a reverse shell
- Privilege escalation using sudo CVE-2021-3156
- read /etc/shadow
- Read ssh process environment variable
- Lateral movement to alice-box with ssh hijacking

An Incident - An Attacker’s View



An Incident … A Defender’s View

web app RCE on joe-box and executed a reverse shell

connect etc.

Privilege escalation using sudo CVE-2021-3156

execve(at)

read /etc/shadow

open(at)

Read ssh process environment variable

open(at)

Lateral movement to alice-box with ssh hijacking

connect, etc.

Incident Response

Detection Rule Example

rule: untrusted program reads */etc/shadow*

condition:

syscall == open(at)

and read permission

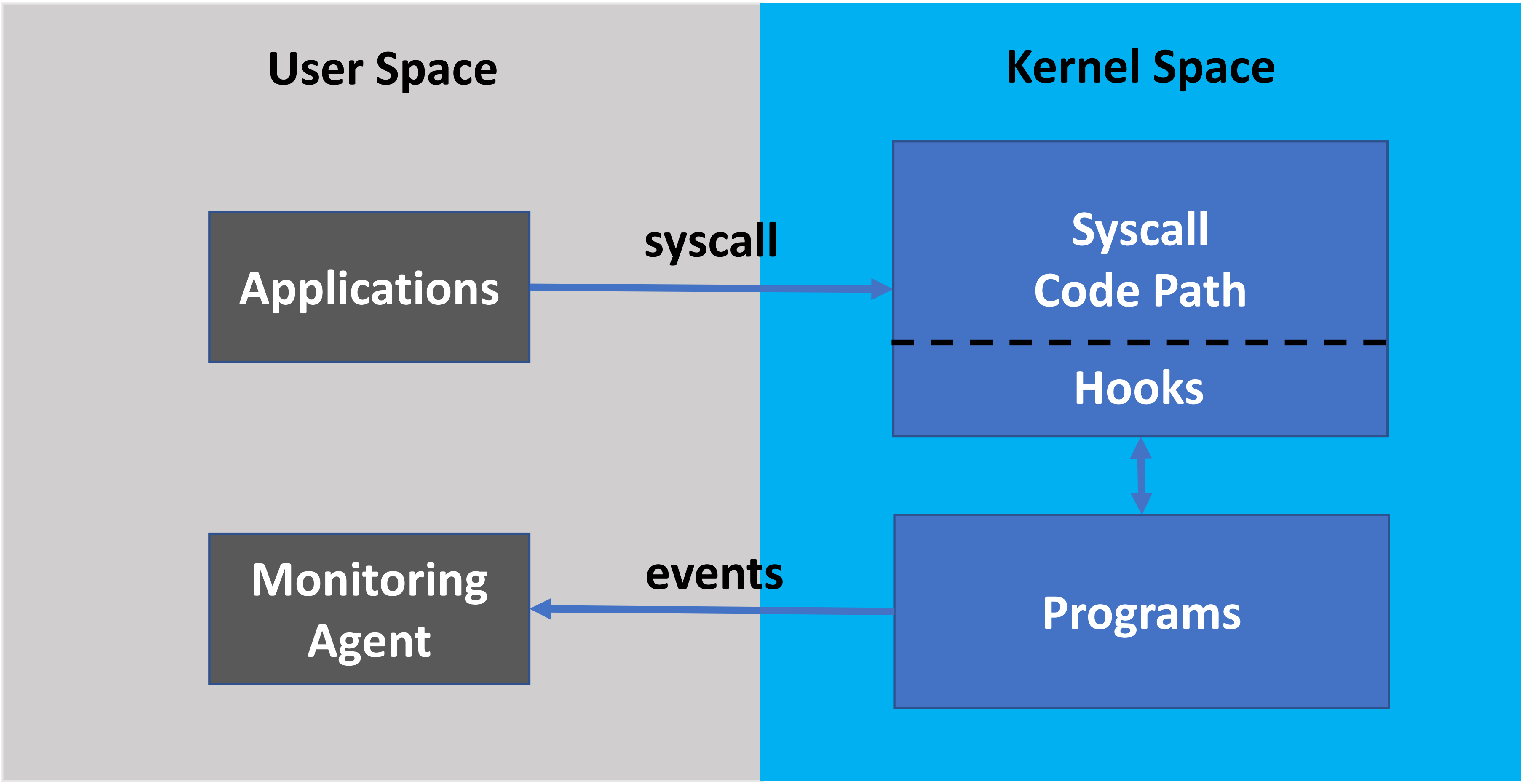
and filename == */etc/shadow*

and program is not in allowlist

Agenda

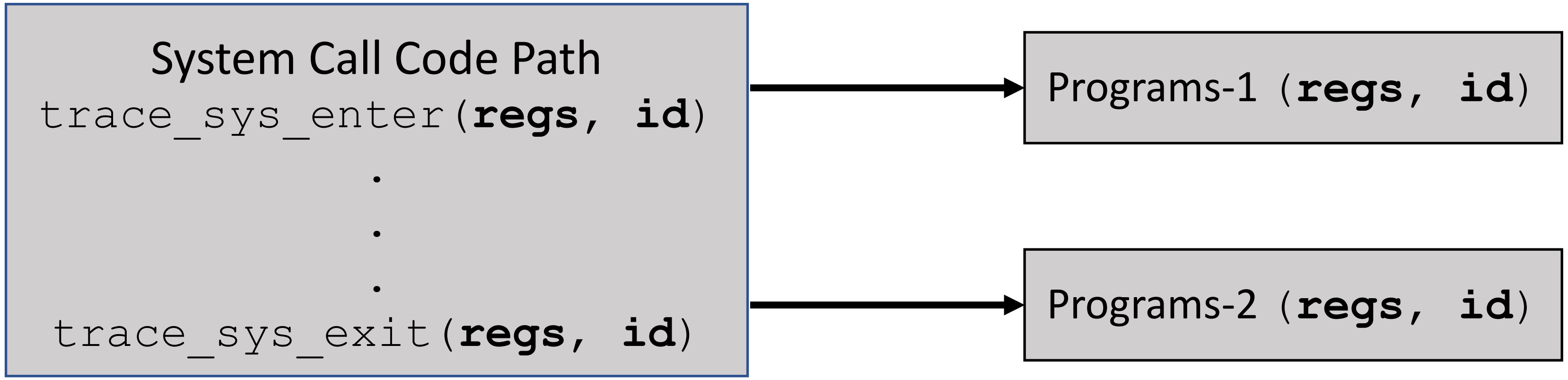
- Introduction to System Call Monitoring
- Open Source System Call Monitoring Projects
- TOCTOU - Phantom v1 Attack
- Semantic confusion - Phantom v2 Attack
- Takeaways

System Call Monitoring (1)



System Call Monitoring – Syscall Interception (1)

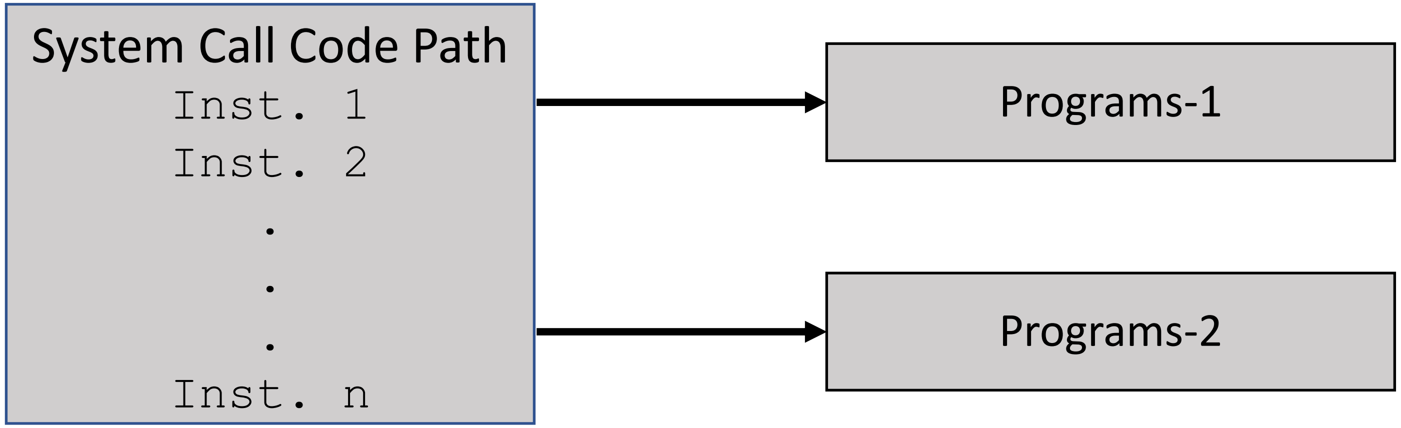
- tracepoint/raw_tracepoint
  - Kernel static hook
  - tracepoint vs raw_tracepoint
  - Linux Kernel provides raw tracepoints: *sys_enter* and *sys_exit*
    - *trace_sys_enter(struct pt_regs *regs, long id)*
    - *trace_sys_exit(struct pt_regs *regs, long id)*



  - Low overhead but only static system call interceptions

System Call Monitoring – Syscall Interception (2)

- kprobe/kretprobe
  - Dynamic hook in the kernel



  - kprobe vs kretprobe
  - Dynamic but slow compared to tracepoints and need to know exactly how data is placed on the stack and register
- LD_PRELOAD: not working in all cases
- Ptrace: performance overhead is high

System Call Monitoring – Syscall Data Collection

- Tracing programs collect system call data, e.g., arguments
- Tracing programs can “attach” to different hooks. When the hooks fire, tracing programs are executed
  - tracepoints/raw_tracepoints
  - kprobe/kretprobe
- Tracing programs implementations
  - Linux native mechanisms: *ftrace*, *perf_events* etc.
  - Kernel modules
  - eBPF programs: allow the execution of user code in the kernel

Open Source Projects (as of 07/15/2021)

- Falco (created by Sysdig)
  - One of the two security and compliance projects in CNCF incubating projects
  - The only endpoint security monitoring project in CNCF incubating projects
  - 3.9K github stars
  - It consumes kernel events and enriches them with information from the cloud native stack (e.g. Linux, containers, etc.)
  - Falco supports both kernel module and eBPF programs for tracing program implementation
- Tracee (created by Aqua Security)
  - 1.1K github stars
  - A runtime security and forensics tool based on eBPF

Vulnerabilities

- Time-of-check time-of-use (TOCTOU)
  - Time-of-check: tracing programs collect system call data (e.g. arguments)
  - Time-of-use: system call data used by kernel is different from what tracing programs check
  - *e.g. sys_openat(int dfd, const char __user * filename, int flags, umode_t mode)*
  - Phantom v1 attack exploits TOCTOU
- Semantic confusion
  - Kernel interprets data differently from the tracing programs
  - *e.g.* symbolic link is interpreted differently by the kernel and tracing programs
  - Phantom v2 attack exploits semantic confusion
- Falco is vulnerable to both Phantom v1 and v2
- Tracee is vulnerable to Phantom v1

https://dl.packetstormsecurity.net/1005-advisories/khobe-earthquake.pdf

TOCTOU - openat

syscall enter

User Space

Kernel 5.4.0

```
…
trace_sys_enter(regs, regs->orig_ax)
…
```

Syscall Table (x86_64)
…
257 sys_openat
258 sys_mkdirat
259 sys_mknodat
…

```
long do_sys_open(int dfd,
     const char __user *filename,
     int flags, umode_t mode)
{
   …
   struct filename *tmp;
   tmp = getname(filename);
   …
   fd = get_unused_fd_flags(flags);
   if (fd >= 0) {
     struct file *f =
           do_filp_open(dfd, tmp, &op);
   …
}
```

```
…
trace_sys_exit(regs, regs->ax)
…
```

syscall exit

TOCTOU - openat

syscall enter

User Space

Kernel 5.4.0

```
...
trace_sys_enter(regs, regs->orig_ax)
...
```

Syscall Table (x86_64)
…
257 sys_openat
258 sys_mkdirat
259 sys_mknodat
…

```
long do_sys_open(int dfd,
        const char __user *filename,
        int flags, umode_t mode)
{
    …
    struct filename *tmp;
    tmp = getname(filename);
    …
    fd = get_unused_fd_flags(flags);
    if (fd >= 0) {
      struct file *f =
            do_filp_open(dfd, tmp, &op);
    …
}
```

TOU by Linux Kernel

```
...
trace_sys_exit(regs, regs->ax)
...
```

syscall exit

TOCTOU - openat

syscall enter

User Space

Kernel 5.4.0

```
...
trace_sys_enter(regs, regs->orig_ax)
...
```

Syscall Table (x86_64)
…
257 sys_openat
258 sys_mkdirat
259 sys_mknodat
…

```
long do_sys_open(int dfd,
        const char __user *filename,
        int flags, umode_t mode)
{
    …
    struct filename *tmp;
    tmp = getname(filename);
    …
    fd = get_unused_fd_flags(flags);
    if (fd >= 0) {
      struct file *f =
              do_filp_open(dfd, tmp, &op);
    …
}
```

TOU by Linux Kernel

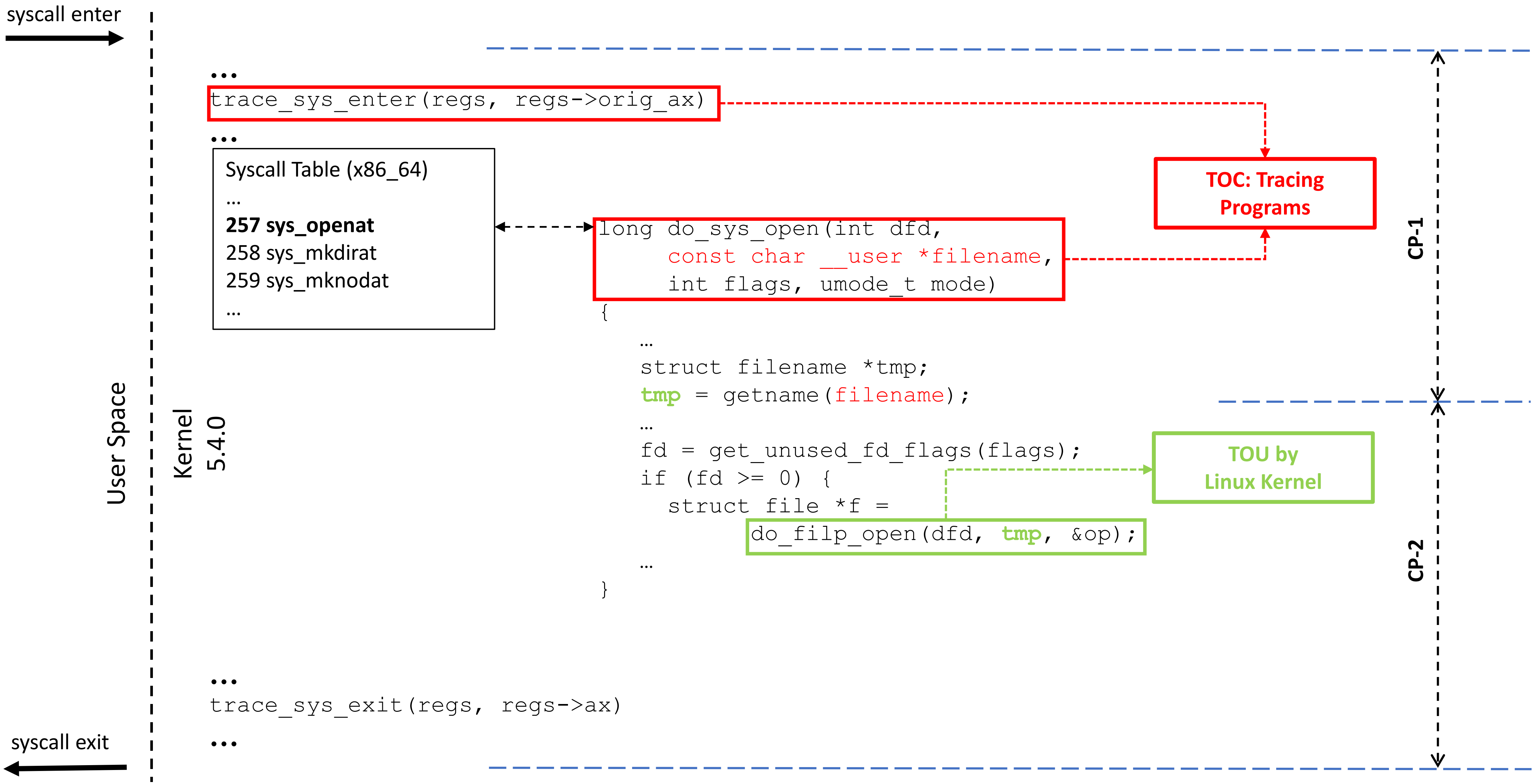
CP-1

CP-2

```
...
trace_sys_exit(regs, regs->ax)
...
```

syscall exit

TOCTOU - openat



TOCTOU - openat

syscall enter

User Space

Kernel 5.4.0

```
...
trace_sys_enter(regs, regs->orig_ax)
...
```

Syscall Table (x86_64)
…
257 sys_openat
258 sys_mkdirat
259 sys_mknodat
…

```
long do_sys_open(int dfd,
        const char __user *filename,
        int flags, umode_t mode)
{
    …
    struct filename *tmp;
    tmp = getname(filename);
    …
    fd = get_unused_fd_flags(flags);
    if (fd >= 0) {
        struct file *f =
               do_filp_open(dfd, tmp, &op);
    …
}
```

TOU by Linux Kernel

TOC: Tracing Programs

CP-1

CP-2

```
...
trace_sys_exit(regs, regs->ax)
...
```

syscall exit

TOCTOU - openat

syscall enter

User Space

Kernel 5.4.0

```
...
trace_sys_enter(regs, regs->orig_ax)
...
```

Syscall Table (x86_64)
...
257 sys_openat
258 sys_mkdirat
259 sys_mknodat
...

```
long do_sys_open(int dfd,
        const char __user *filename,
        int flags, umode_t mode)
{
    …
    struct filename *tmp;
    tmp = getname(filename);
    …
    fd = get_unused_fd_flags(flags);
    if (fd >= 0) {
      struct file *f =
                do_filp_open(dfd, tmp, &op);
    …
}
```

TOU by Linux Kernel

CP-1

CP-2

```
...
trace_sys_exit(regs, regs->ax)
...
```

TOC: Tracing Programs

syscall exit

TOCTOU – Falco

- CVE-2021-33505 – CVSS v3.0 score 7.3
- Falco older than v0.29.1 (or open source sysdig)
  - Commercial versions based on the open source agent are also affected (confirmed by the open source maintainer)
- It uses *raw tracepoints* (*sys_enter* and *sys_exit*) to intercept syscalls
- User space pointers are read directly by its tracing programs
  - In the implementations of both kernel module and eBPF programs

TOCTOU – Falco

- We evaluated the important syscalls in Falco rules.

Syscall	Category	TOCTOU?
connect	Network	Y
sendto/sendmsg	Network	Y
open/openat	File	Y
execve	File	N
rename	File	Y
renameat/renameat2	File	Y
mkdir	File	Y
mkdirat	File	Y
rmdir	File	Y
unlink/unlinkat	File	Y
symlink/symlinkat	File	Y
chmod/fchmod/fchmodat	File	Y
creat	File	Y

TOCTOU – Tracee

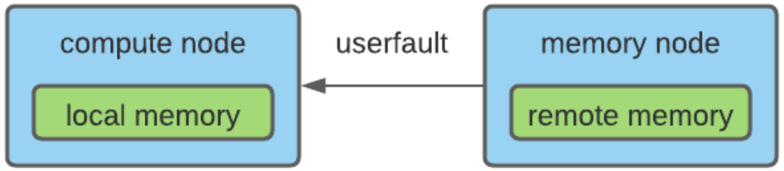
- Tracee (*v0.4.0*) is vulnerable to TOCTOU for many system calls, e.g., ***connect*** syscall, etc.
- No CVE given. Here are some quotes from the maintainers:
  - “As you probably know, TOCTTOU attacks on system calls wrappers(/tracers) is a well known issue, and Tracee is no exception.”
  - “And yes we agree on the fact that there’s no CVE or novel finding and therefore you could talk about it publicly.”
  - Interpret yourself ☺

Phantom v1 Exploit Plan (Sys_exit is Monitored)

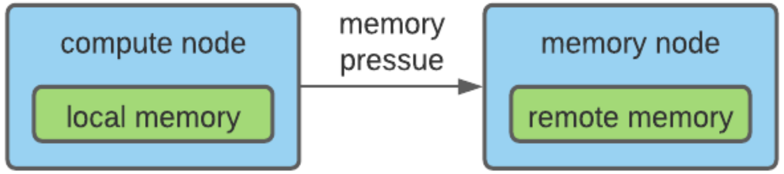
- Triggers the target system call with malicious arguments
- Let kernel reads the malicious arguments and performs the intended malicious action
- Overwrites the data structure pointed by the user space argument pointer with benign data
- At sys_exit, tracing program reads the data structure pointed by the user space pointer and checks against the rules
- Challenges:
  - When does the kernel thread reads it?
  - How can we synchronize the overwrite with the kernel thread read?
  - Are the racing windows big enough for each syscalls?
  - How to ensure the tracing program get the overwritten copy?

Userfaultfd Syscall

- Normally page faults are a kernel internal thing…
  - Why offload page faults to userland?
- Memory externalization: running programs with memory residing on a remote node
  - Memory is transferred from the memory node to the compute node on access



  - Memory can be transferred from the compute node to the memory node if it’s not frequently used during memory pressure

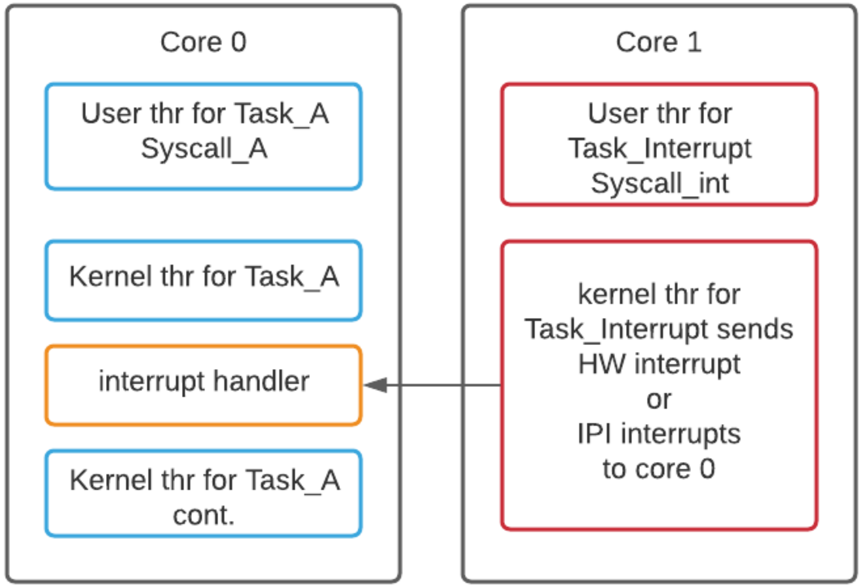


- Once userfaultfd triggers, kernel thread is paused and waits for user space response
  - Helps exploitation on kernel race condition bugs

Interrupts and Scheduling

- An interrupt notifies the processor with an event that requires immediate attention
- An interrupt diverts the program control flow to an interrupt handler
- Interrupt can be triggered indirectly from system calls
  - Hardware interrupts (networking, e.g., connect)
  - Interprocessor interrupts (IPIs) (e.g., mprotect)
- sched_setscheduler()
  - set SCHED_NORMAL / SCHED_IDLE
  - Realtime policies require CAP_SYS_NICE

  or Realtimekit]
- sched_setaffinity(): pin task to CPU bitmask



https://www.usenix.org/system/files/sec21fall-lee-yoochan.pdf
https://www.youtube.com/watch?v=MIJL5wLUtKE

Phantom v1 Attack – An Openat Example

main thread

userfaultfd thread

overwrite thread

pin CPU 3
mmap page A
register userfaultfd thread

any CPU

pin CPU 2

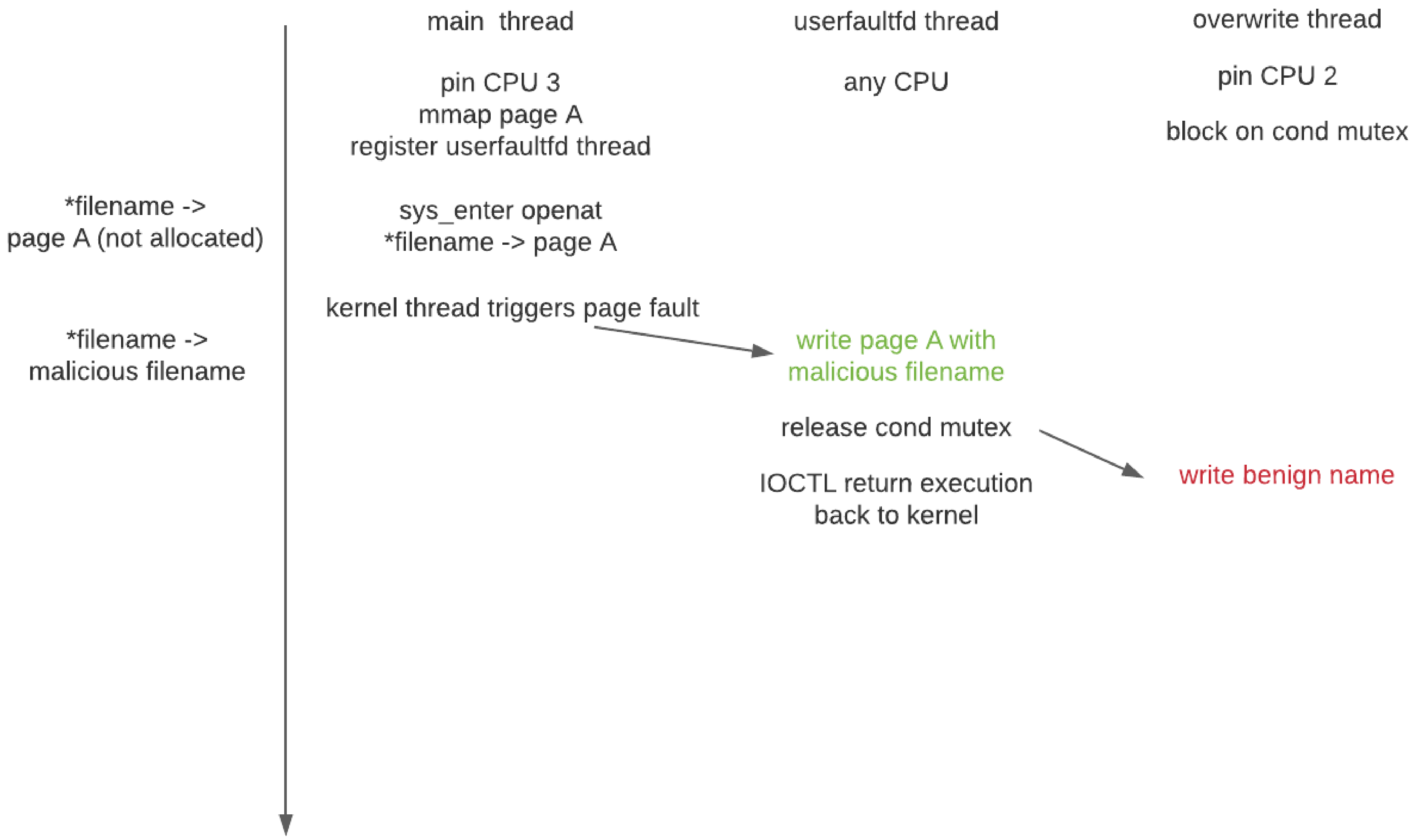
block on cond mutex

*filename ->
page A (not allocated)

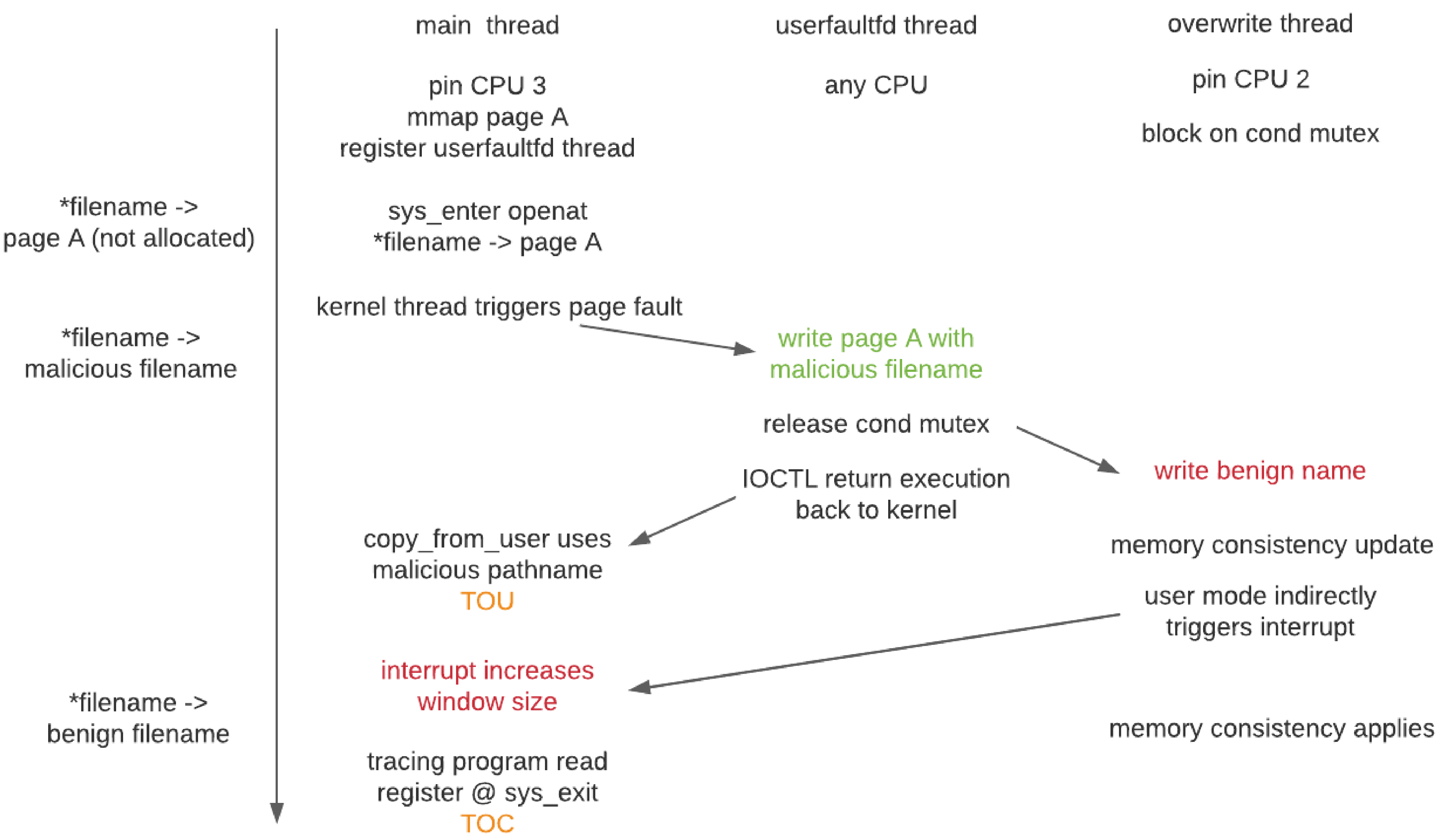
sys_enter openat
*filename -> page A

kernel thread triggers page fault

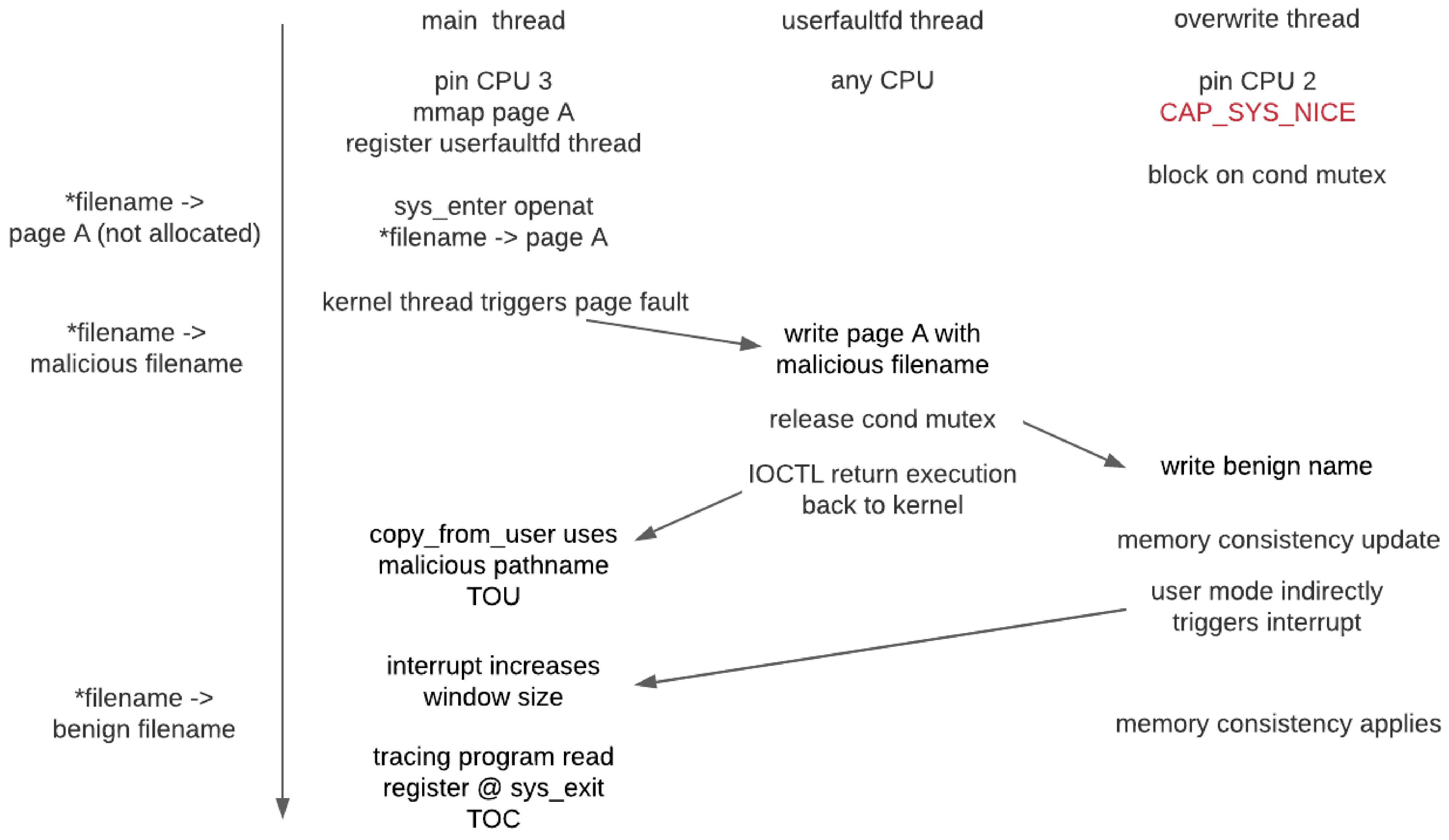
Phantom v1 Attack – An Openat Example



Phantom v1 Attack – An Openat Example



Phantom v1 Attack – An Openat Example



Semantic Confusion – File Link

- Semantic confusion indicates the different interpretations of data (e.g. system call arguments) between the kernel and tracing programs
- In particular for file link interpretation, kernel trails symbolic link to actual file while tracing programs read the link without any interpretations
- Falco is vulnerable to semantic confusion
  - It reads symbolic link without any interpretation
  - No CVE because symlink(at) and link(at) are monitored
  - But practically the detection team need to track all symlink(at)/link(at) to any file based rules ☹
- Tracee is not vulnerable to openat
  - *security_file_open* LSM hook: *filename* has been interpreted by the kernel

Phantom v2 – An File Link Example

rule: untrusted program reads */etc/shadow*
condition:
    syscall == open(at)
    and read permission is used
    and filename == */etc/shadow*
    and not program is not in allowlist

- Steps to bypass the Falco rule
  - Create a symlink `/tmp/shadow` -> `/etc/shadow`
  - Tracing programs read the symlink `/tmp/shadow`
  - Syscall openat monitoring reports /tmp/shadow is opened
  - Rule is bypassed

Mitigation

- Detection (Falco team)
  - Detect (unprivileged) usage of the `userfaultfd` syscall (Implemented)
  - Detect a user registering a memory address range
  - Detect a user copying a continuous memory chunk into the userfaultfd registered range and (optionally) wake up the blocked thread (kernel)
- Read the data used by system calls
  - LSM hook: a list of check points

LSM hook used by Tracee v0.4.0	Protected syscall
security_bprm_check	execve, execveat
security_file_open	open, openat
security_inode_unlink	unlink, unlinkat
security_mmap_addr	mmap, mmap_pgoff
security_file_mprotect	mprotect

  - Kernel data structure: read arguments of *execve* from *mm->arg_start*

Takeaways

- Phantom attack is generic and exploits the fact that kernel and tracing programs
  - Can read data at different times (Phantom v1)
  - Can interpret data differently (Phantom v2)
- Kernel raw tracepoints on system calls are not ideal for secure tracing
- Other tracing implementations can be vulnerable. E.g., kprobe
- Mitigation:
  - Detect abnormal usages of userfaultfd
  - Ensure kernel and secure tracing programs (1) Read the same data (2) Interpret data in the same way
- If you are interested in discussing further:
  - @Xiaofei_REX (OpenDM)
  - https://github.com/rexguowork/phantom-attack (will be released during Defcon)

Acknowledgement

- Chris Arges (ebpf, kernel tracing)
  - https://www.linkedin.com/in/carges/
- Joel Schopp (kernel tracing, TOCTOU)
  - https://www.linkedin.com/in/schopp/
- Yu Wang (TOCTOU)
  - https://www.linkedin.com/in/yu-wang-88056b99/
- Falco open source team (Leonardo Di Donato, etc.)