HOM3C0MING D3F C0N

Trace Me If You can: Bypassing Linux Syscall Tracing

Dr. Rex Guo, Lacework
Dr. Junyuan Zeng, Linkedin.com

About Rex Guo

- Principal Engineer @ Lacework
  - Behavior anomaly detection (Polygraph)
  - CSPM
- Engineering Manager @ Startups
  - Confluera (XDR)
  - Tetration (CWPP, now part of Cisco)
- Conference speaker at Blackhat, DEFCON,…
- @Xiaofei_REX

LACEWORK

About Junyuan Zeng

- Linkedin
  - Senior Software Engineer: Kubernetes
- JD.com
  - Staff Security Architect/Engineer: Cloud native security
- Samsung Research America & FireEye
  - Staff Security Software Engineer/Researcher: Mobile security

An Incident - An Attacker's View

Log4shell RCE on joe-box and executed a reverse shell

LACEWORK

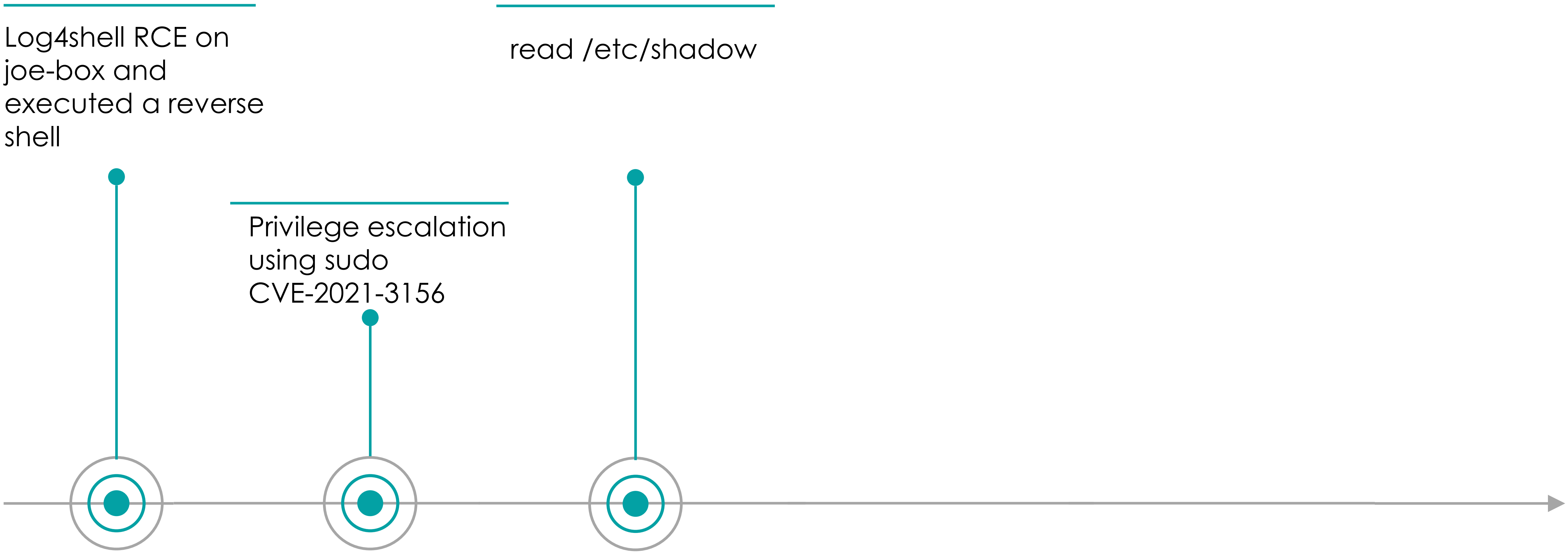
An Incident - An Attacker’s View

Log4shell RCE on joe-box and executed a reverse shell

Privilege escalation using sudo CVE-2021-3156

LACEWORK

An Incident - An Attacker’s View



An Incident - An Attacker’s View

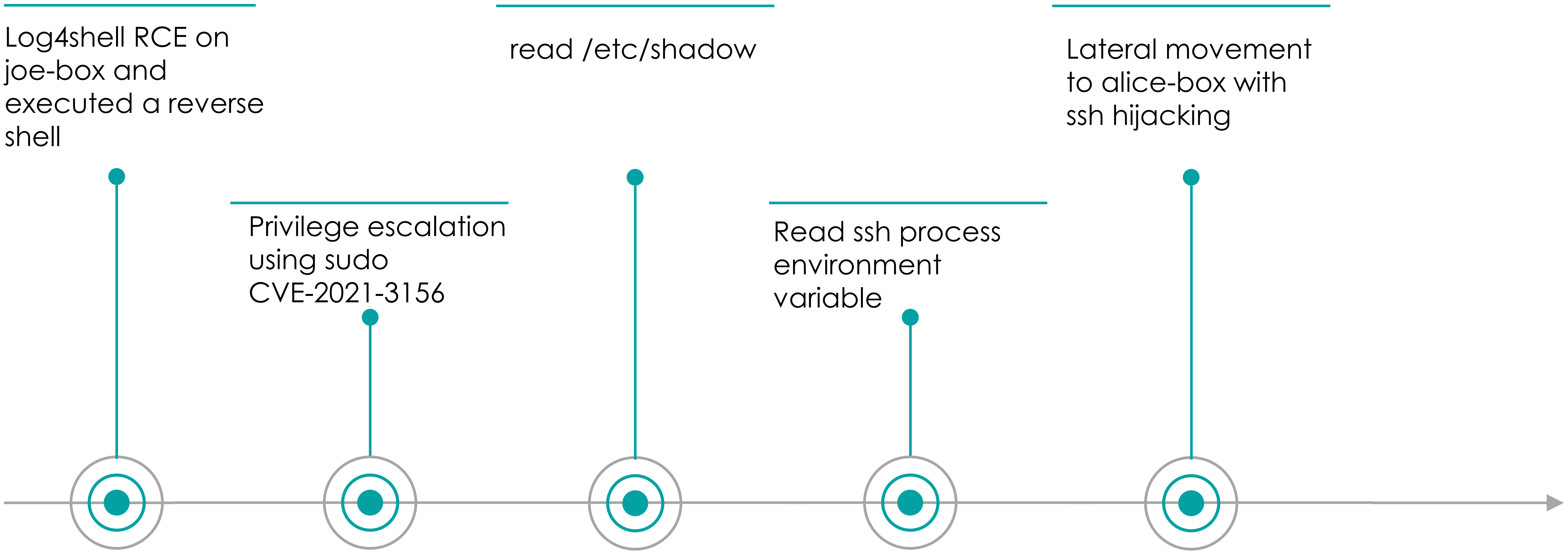
Log4shell RCE on joe-box and executed a reverse shell

Privilege escalation using sudo CVE-2021-3156

read /etc/shadow

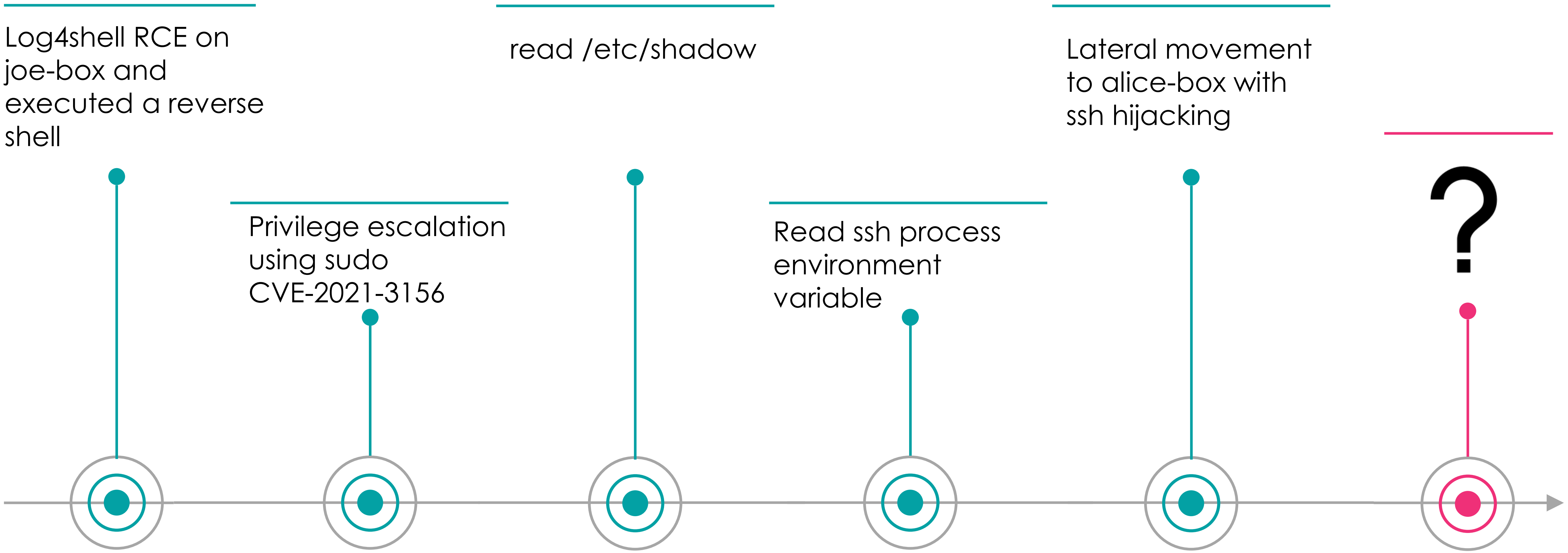
Read ssh process environment variable

An Incident - An Attacker's View

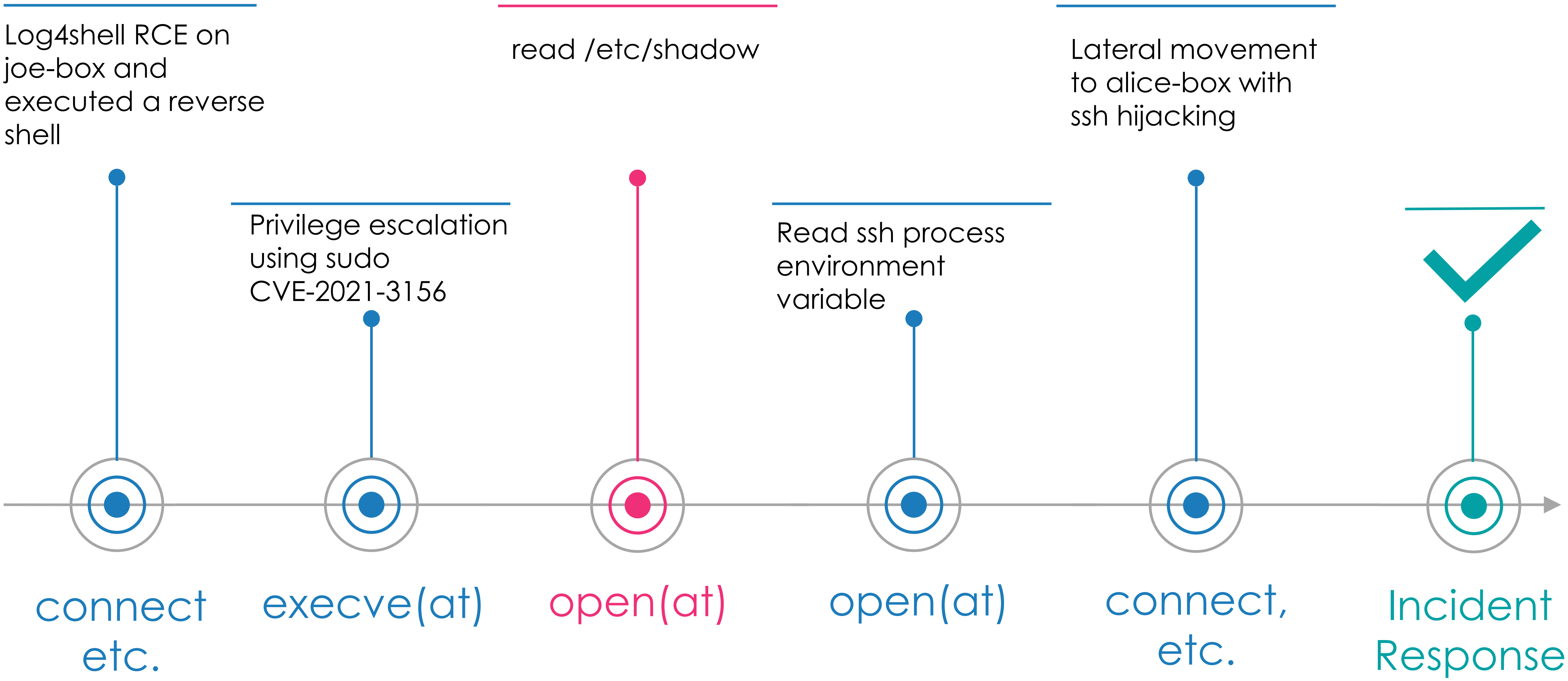


LACEWORK

An Incident - An Attacker's View



An Incident - A Defender's View



LACEWORK

Detection Rule Example

rule: untrusted program reads */etc/shadow*

condition:

syscall == open(at)

and has read permission

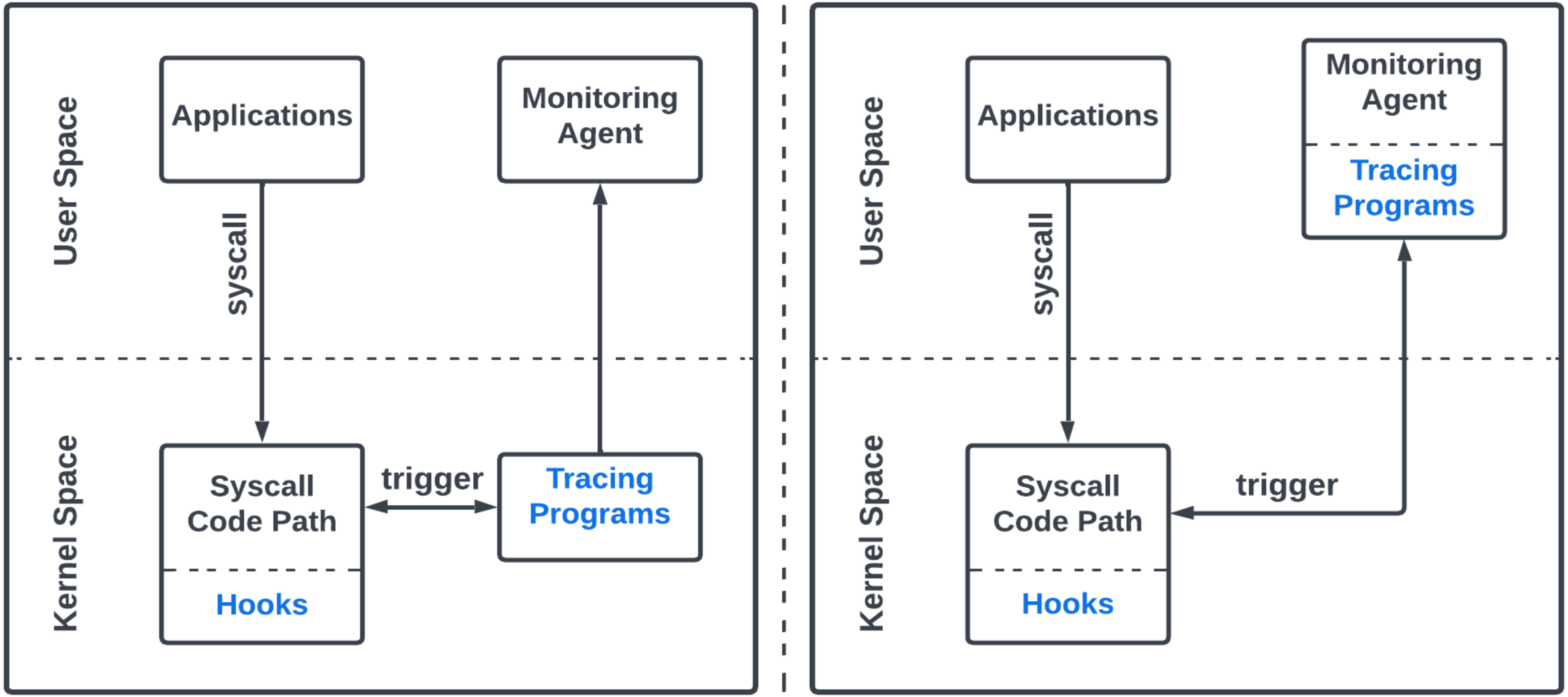
and filename == */etc/shadow*

and program is not in allowlist

Agenda

- Syscall Tracing
- Vulnerabilities
- Exploitations
- Mitigations
- Takeaways

System Call Tracing



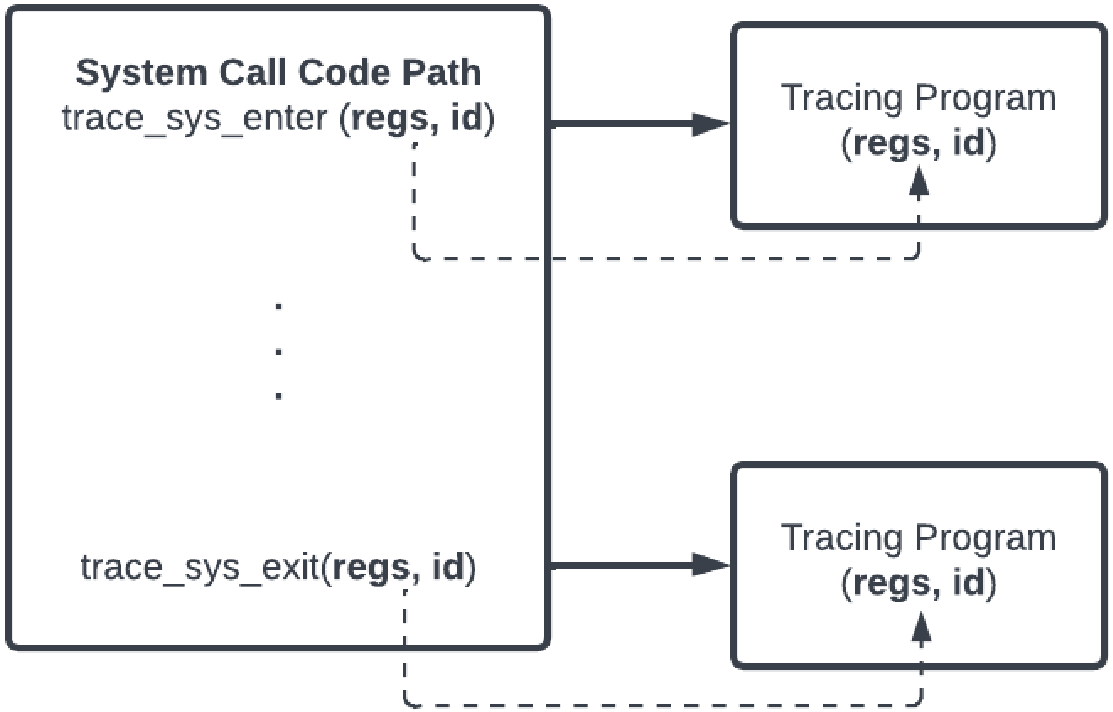
LACEWORK

System Call Tracing – Tracing Program

- Tracing programs collect system call data, e.g., arguments
- Tracing programs can “attach” to different hooks
  - tracepoints, kprobe, ptrace etc.
- Tracing programs implementations
  - Linux native mechanisms: *ftrace*, *perf_events* etc.
  - Kernel modules, eBPF probe and user space programs

System Call Tracing – tracepoint

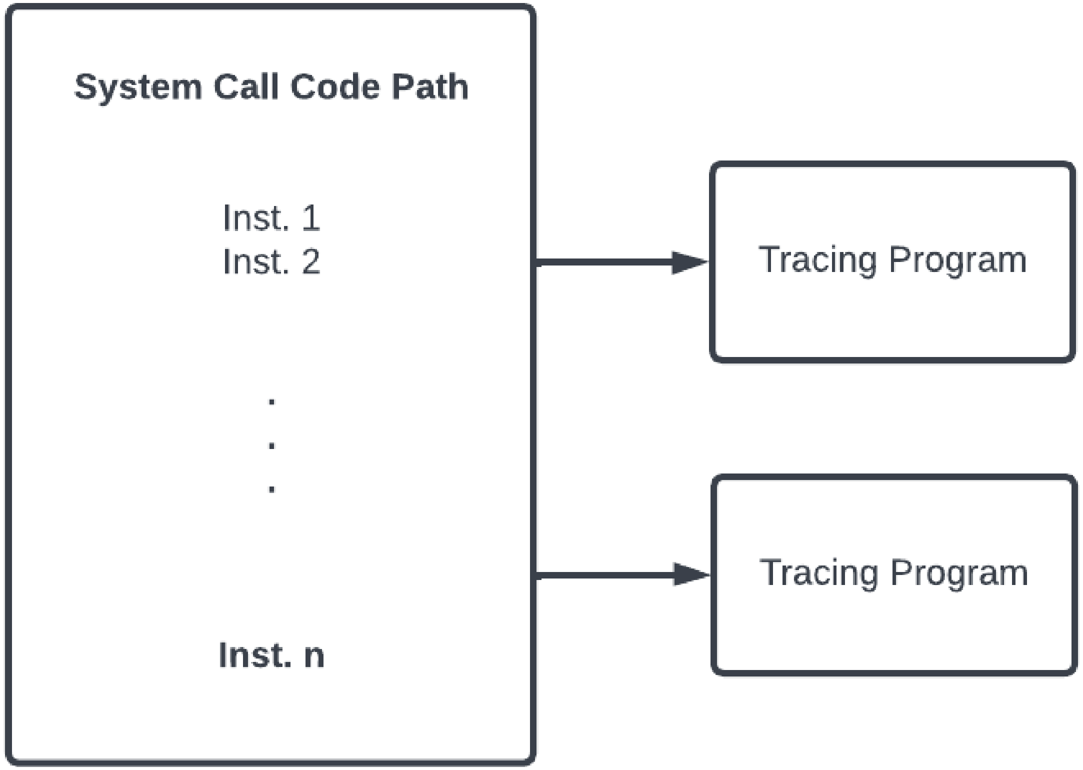
- tracepoint
  - Kernel static hook
  - Linux kernel provides *sys_enter* and *sys_exit*



  - Low overhead but only static interceptions

System Call Tracing – kprobe

- kprobe
  - Dynamic hook in the kernel
  - Register tracing programs on instructions in syscall code path
  - Dynamic but slow compared to tracepoint and need to know exactly how data is placed on the stack and registers



System Call Tracing – ptrace

- ptrace
  - A static hook
  - No Kernel Module/eBPF program are needed
  - Performance overhead is high
  - Can combine with seccomp to reduce overhead
- Others (LD_PRELOAD etc.)

Cloud Workloads

- Virtual machines
  - AWS EC2 instances
  - Google VM instances
- Containers on customer-managed VMs
  - AWS EC2 tasks
  - Standard GKE workloads (e.g. DaemonSet etc.)
  - AKS workloads
- Serverless containers: have no access to the host
  - AWS Fargate tasks
  - GCP Cloud Run services
- Others (AWS Lambda etc.)

System Call Tracing for Cloud Workloads

Workload	System Call Tracing
VMs	• Hooks: tracepoint, kprobe, ptrace • Tracing programs: kernel programs (eBPF, kernel Module), user programs • Tools: Falco eBPF/kernel Module, Falco pdig
Containers	• Hooks: tracepoint, kprobe, ptrace • Tracing programs: kernel programs (eBPF, kernel Module), user programs • Tools: Falco eBPF/kernel Module, Falco pdig
Serverless Containers	• Hooks: ptrace • Tracing programs: user programs • Tools: Falco pdig

LACEWORK

Open Source Projects

- Falco
  - Open source endpoint security monitoring project in CNCF
  - 5K+ github stars
  - Falco supports syscall tracing techniques:
    - tracepoint + kernel module
    - tracepoint + eBPF probe
    - pdig: ptrace + userspace program
- Falco pdig
  - Support syscall tracing of serverless workloads

TOCTOU in Syscall Tracing

- *sys_connect(int fd, struct sockaddr __user * uservaddr, int addrlen)*
- TOC (Time-Of-Check): tracing programs dereference this user space pointer
- TOU (Time-Of-Use): the kernel dereferences this user space pointer

TOCTOU - Connect

syscall enter

User Space

Kernel 5.7.0

Execution Flow

```
ptrace_report_syscall(regs, message)
__secure_computing(struct seccomp_data{regs…})
trace_sys_enter(regs, regs->orig_ax)
```

ptrace/seccomp/sysenter tracepoint

Syscall Table (x86_64)
…
42 sys_connect
43 sys_accept
44 sys_sendto
…

```
long __sys_connect((int fd,
      struct sockaddr __user *uservaddr, int addrlen))
{
   …
   struct filename *tmp;
   ret = move_addr_to_kernel
           (uservaddr, addrlen, &address);
   if (!ret)
     ret = __sys_connect_file
           (f.file, &address, addrlen, 0);

   …
}
```

```
trace_sys_exit(regs, regs->ax)
ptrace_report_syscall(regs, message)
```

syscall exit

LACEWORK

TOCTOU - Connect

syscall enter

User Space | Kernel 5.7.0

Execution Flow

```
ptrace_report_syscall(regs, message)
__secure_computing(struct seccomp_data{regs…})
trace_sys_enter(regs, regs->orig_ax)
```

Syscall Table (x86_64)
…
42 sys_connect
43 sys_accept
44 sys_sendto
…

```
long __sys_connect((int fd,
     struct sockaddr __user *uservaddr, int addrlen))
{
   …
   struct filename *tmp;
   ret = move_addr_to_kernel
           (uservaddr, addrlen, &address);
   if (!ret)
     ret = __sys_connect_file
           (f.file, &address, addrlen, 0);

   …
}
```

```
trace_sys_exit(regs, regs->ax)
ptrace_report_syscall(regs, message)
```

syscall exit

LACEWORK

TOCTOU - Connect

syscall enter

User Space

Kernel 5.7.0

Execution Flow

```
ptrace_report_syscall(regs, message)
__secure_computing(struct seccomp_data{regs…})
trace_sys_enter(regs, regs->orig_ax)
```

Syscall Table (x86_64)
…
42 sys_connect
43 sys_accept
44 sys_sendto
…

```
long __sys_connect((int fd,
        struct sockaddr __user *uservaddr, int addrlen))
{
    …
    struct filename *tmp;
    ret = move_addr_to_kernel
            (uservaddr, addrlen, &address);
    if (!ret)
      ret = __sys_connect_file
            (f.file, &address, addrlen, 0);

    …
}
```

```
trace_sys_exit(regs, regs->ax)
ptrace_report_syscall(regs, message)
```

ptrace/sysexit tracepoint

syscall exit

LACEWORK

TOCTOU - Connect

syscall enter

User Space | Kernel 5.7.0

Execution Flow

■ **Userspace pointer pointing to "socket address"**

```
ptrace_report_syscall(regs, message)
__secure_computing(struct seccomp_data{regs…})
trace_sys_enter(regs, regs->orig_ax)
```

Syscall Table (x86_64)
…
42 sys_connect
43 sys_accept
44 sys_sendto
…

```
long __sys_connect((int fd,
        struct sockaddr __user *uservaddr, int addrlen))
{
    …
    struct filename *tmp;
    ret = move_addr_to_kernel
            (uservaddr, addrlen, &address);
    if (!ret)
      ret = __sys_connect_file
            (f.file, &address, addrlen, 0);

    …
}
```

```
trace_sys_exit(regs, regs->ax)
ptrace_report_syscall(regs, message)
```

syscall exit

LACEWORK

TOCTOU - Connect

■ Userspace pointer pointing to "socket address" (red)
■ Kernel pointer pointing to "socket address" (green)

syscall enter

User Space

Kernel 5.7.0

Execution Flow

```
ptrace_report_syscall(regs, message)
__secure_computing(struct seccomp_data{regs…})
trace_sys_enter(regs, regs->orig_ax)
```

Syscall Table (x86_64)
…
42 sys_connect
43 sys_accept
44 sys_sendto
…

```
long __sys_connect((int fd,
        struct sockaddr __user *uservaddr, int addrlen))
{
    …
    struct filename *tmp;
    ret = move_addr_to_kernel
            (uservaddr, addrlen, &address);
    if (!ret)
      ret = __sys_connect_file
            (f.file, &address, addrlen, 0);

    …
}
```

```
trace_sys_exit(regs, regs->ax)
ptrace_report_syscall(regs, message)
```

syscall exit

LACEWORK

TOCTOU - Connect

syscall enter

User Space

Kernel 5.7.0

Execution Flow

```
ptrace_report_syscall(regs, message)
__secure_computing(struct seccomp_data{regs…})
trace_sys_enter(regs, regs->orig_ax)
```

Syscall Table (x86_64)
…
42 sys_connect
43 sys_accept
44 sys_sendto
…

```
long __sys_connect((int fd,
        struct sockaddr __user *uservaddr, int addrlen))
{
    …
    struct filename *tmp;
    ret = move_addr_to_kernel
            (uservaddr, addrlen, &address);
    if (!ret)
      ret = __sys_connect_file
            (f.file, &address, addrlen, 0);

    …
}
```

TOU by Linux Kernel

```
trace_sys_exit(regs, regs->ax)
ptrace_report_syscall(regs, message)
```

syscall exit

LACEWORK

TOCTOU - Connect

syscall enter

User Space

Kernel 5.7.0

Execution Flow

```
ptrace_report_syscall(regs, message)
__secure_computing(struct seccomp_data{regs…})
trace_sys_enter(regs, regs->orig_ax)
```

Syscall Table (x86_64)
…
42 sys_connect
43 sys_accept
44 sys_sendto
…

```
long __sys_connect((int fd,
      struct sockaddr __user *uservaddr, int addrlen))
{
    …
    struct filename *tmp;
    ret = move_addr_to_kernel
            (uservaddr, addrlen, &address);
    if (!ret)
      ret = __sys_connect_file
            (f.file, &address, addrlen, 0);

    …
}
```

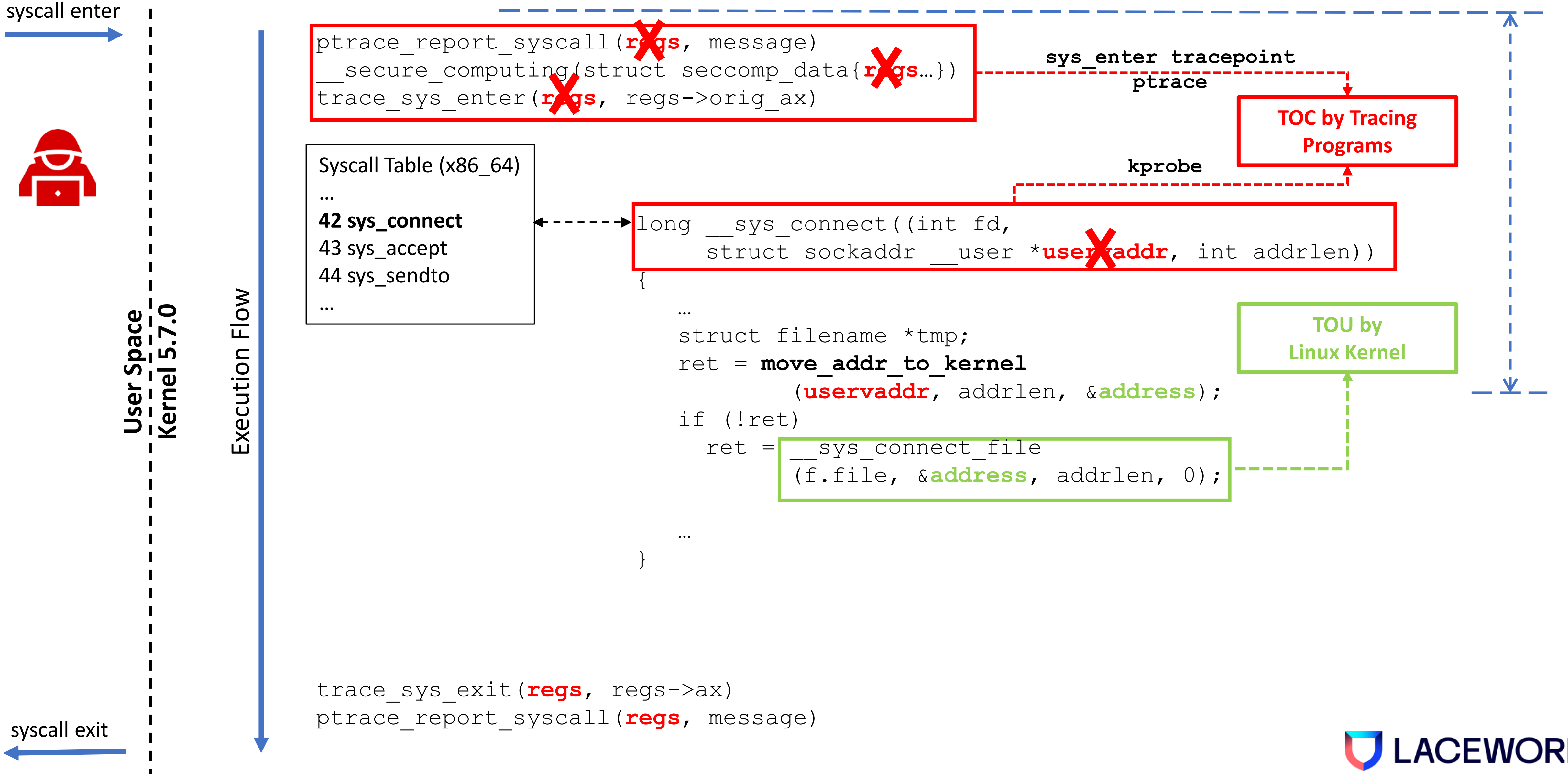
TOU by
Linux Kernel

```
trace_sys_exit(regs, regs->ax)
ptrace_report_syscall(regs, message)
```

syscall exit

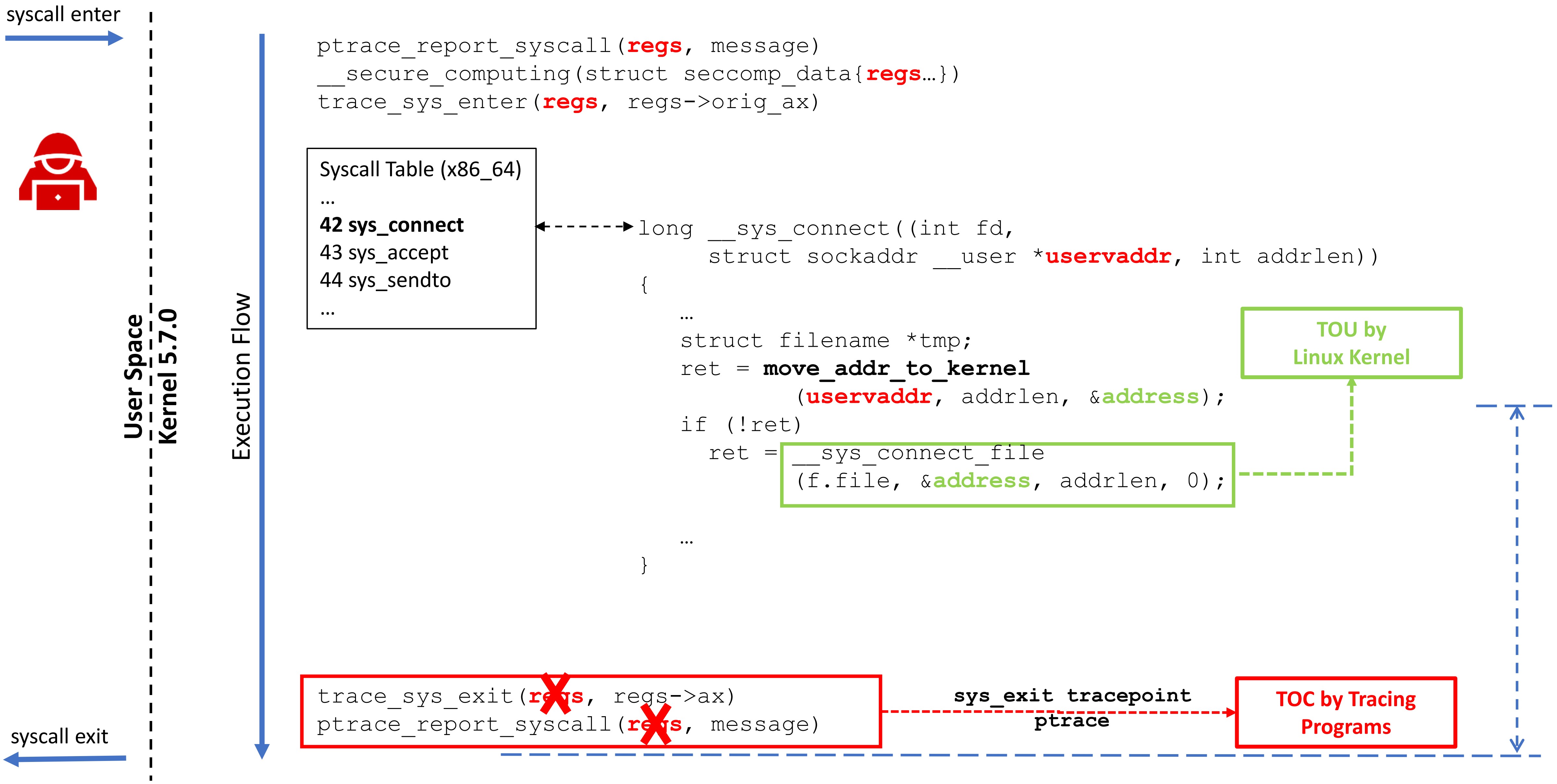
LACEWORK

TOCTOU - Connect



LACEWORK

TOCTOU - Connect



TOCTOU - Connect

syscall enter

User Space

Kernel 5.7.0

Execution Flow

```
ptrace_report_syscall(regs, message)
__secure_computing(struct seccomp_data{regs…})
trace_sys_enter(regs, regs->orig_ax)
```

Falco pdig

Syscall Table (x86_64)
…
42 sys_connect
43 sys_accept
44 sys_sendto
…

```
long __sys_connect((int fd,
     struct sockaddr __user *uservaddr, int addrlen))
{
   …
   struct filename *tmp;
   ret = move_addr_to_kernel
           (uservaddr, addrlen, &address);
   if (!ret)
     ret = __sys_connect_file
           (f.file, &address, addrlen, 0);

   …
}
```

TOU by
Linux Kernel

```
trace_sys_exit(regs, regs->ax)
ptrace_report_syscall(regs, message)
```

Falco (<0.31.1) kernel module/eBPF

Falco pdig

syscall exit

TOCTOU Windows across Kernels

- TOCTOU windows exist since the initial release of tracepoint/ptrace
- Expected behaviors
- Monitor kernel memory

LACEWORK

TOCTOU – Falco

- User space pointers are dereferenced by
  - *sys_exit* tracepoint (kernel module, eBPF)
  - *sys_exit* ptrace (pdig)
- Falco older than v0.31.1
  - Check with vendors which commercial versions are affected
- 12/06/2021 Issue reported (CVE-2022-26316)
- 03/11/2022 Mitigation implemented (Advisory)
  - For selected syscalls, compare sys_enter and sys_exit tracepoint data (Falco LKM, eBPF)
  - Compare sys_enter and sys_exit ptrace data (Falco pdig)

LACEWORK

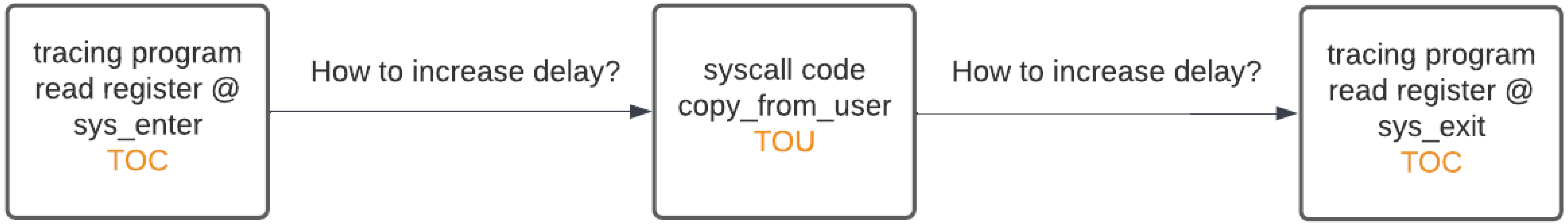
TOCTOU – Falco

- We evaluated the important syscalls in Falco rules.

Syscall	Category	TOCTOU?	Exploitable by blocking condition	Exploitable by DC29 attack
connect	Network	Y	Y	Y
sendto/sendmsg	Network	Y	N	Y
open(at)	File	Y	Y	Y
execve	File	N	N*	N*
rename	File	Y	Y	Y
renameat(2)	File	Y	Y	Y
mkdir(at)	File	Y	Y	Y
rmdir	File	Y	Y	Y
unlink(at)	File	Y	Y	Y
symlink(at)	File	Y	Y	Y
chmod/fchmod(at)	File	Y	Y	Y
creat	File	Y	Y	Y

LACEWORK

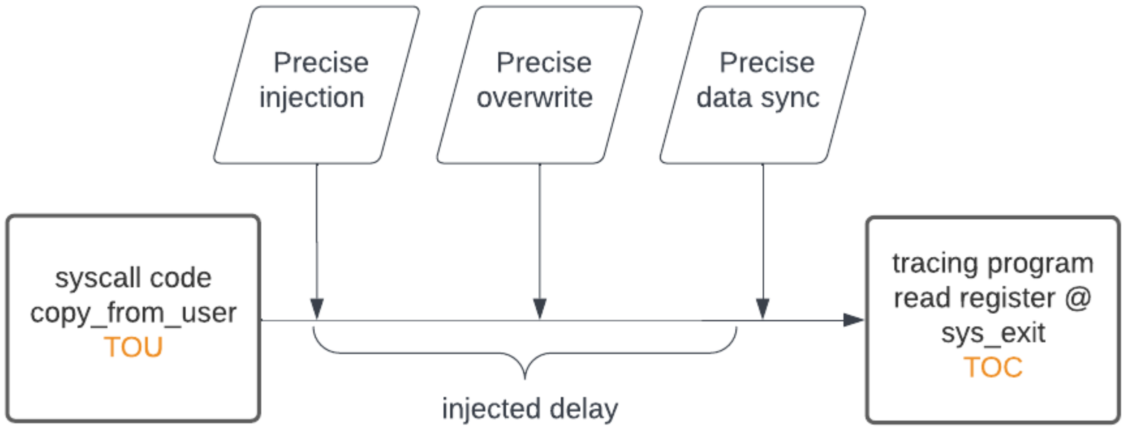
Exploit Requirements



- Exploitation requirements
  - No additional privilege and capabilities
  - Control the time to inject the delay
  - Enough delay for pointer overwrite
  - Reliable

Exploit Strategy 1 (DEFCON 29)

- Injected delay is small
- Requires Userfaultfd syscall for precise injection while pausing the kernel execution
- seccomp can block userfaultfd syscall (e.g., docker default seccomp profile)
- Falco's mitigation was to detect userfaultfd

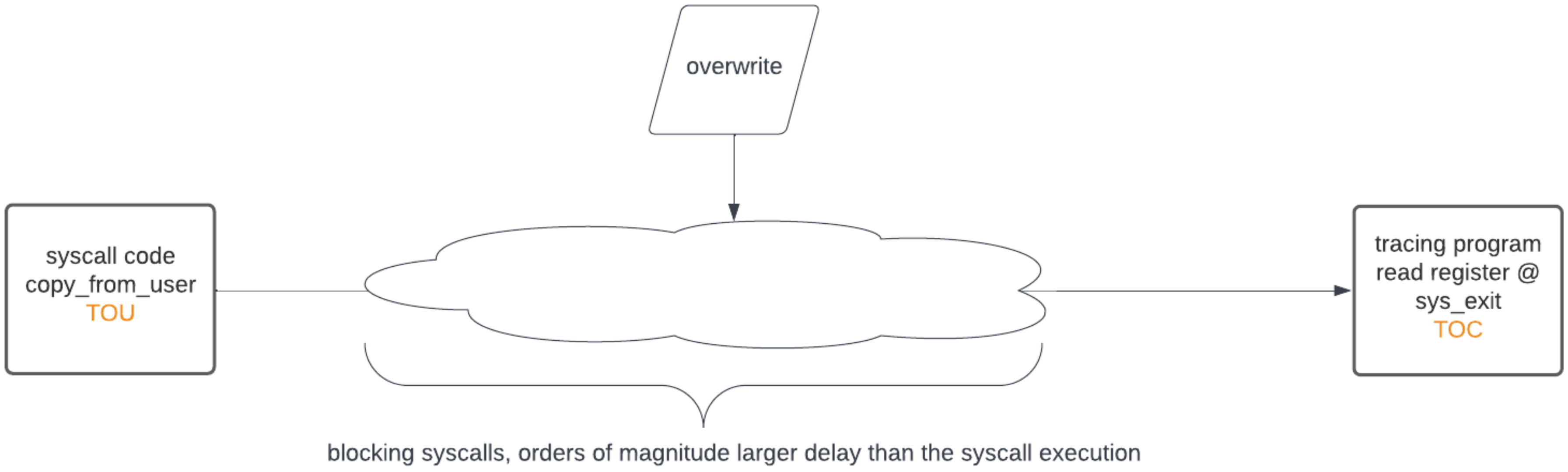


Phantom Attack: Evading System Call Monitoring, DEFCON 29. https://github.com/rexguowork/phantom-attack

LACEWORK

Exploit Strategy 2

- Injected delay >> the syscall execution time
- No Precise control is required



LACEWORK

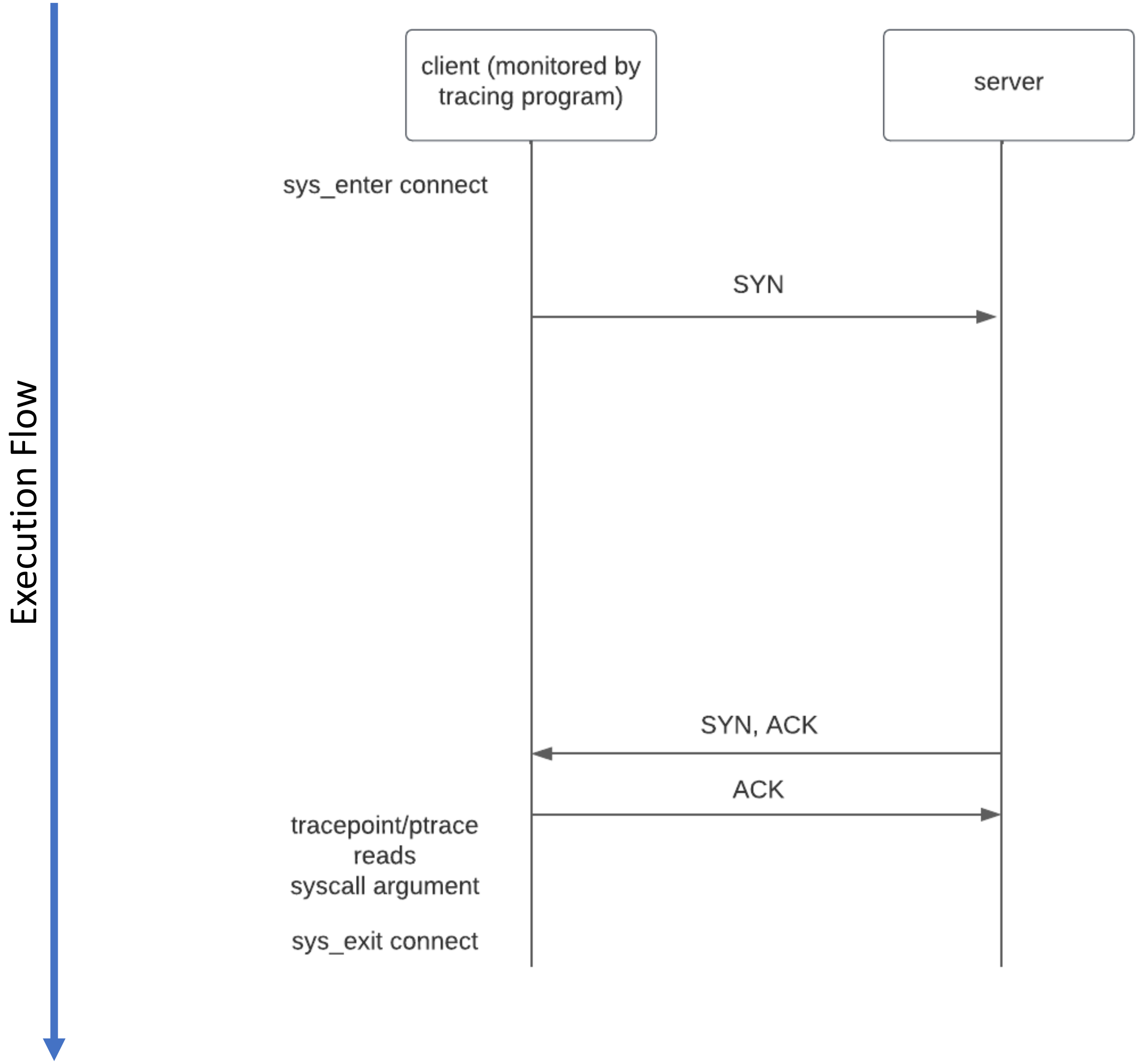
Syscall Built-in Delay

- Attackers can trigger significant syscall delays by introducing:
  - Blocking conditions (attack sys_exit)
  - Seccomp rules (attack sys_enter)
- Syscall can get “blocked”

Categories	**Syscalls**
Process	fork/exec/exit/wait/…
File system	open(at)/symlink(at)/read/write/…
Networking	connect/accept/socket/…
Security	seccomp/keyctl/…
Many others…	…

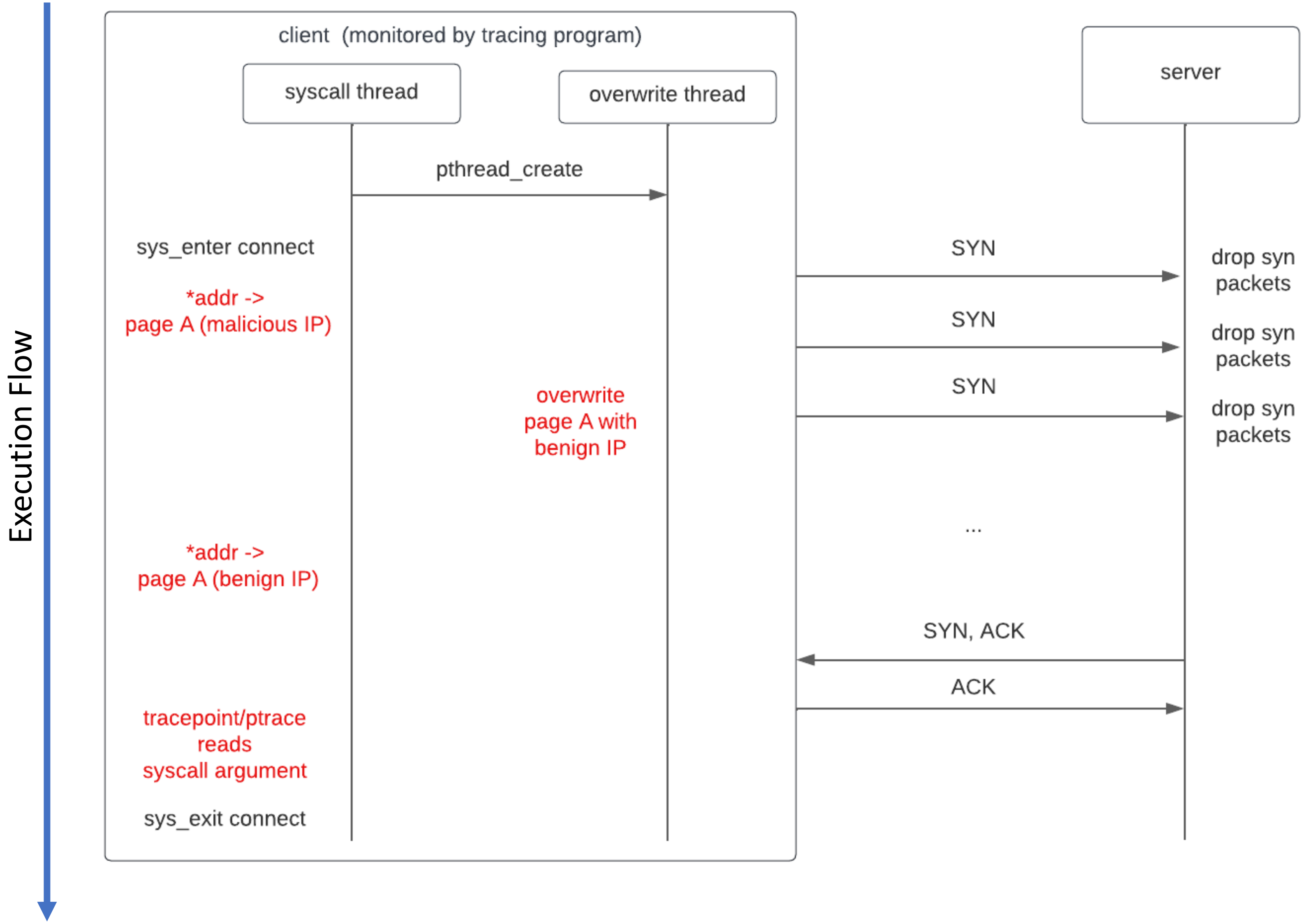
LACEWORK

Connect Syscall



LACEWORK

Bypassing Connect Syscall Tracing (Demo)



Blocking Syscalls (File Systems)

- File system syscalls are all affected
  - open/openat
  - creat
  - rename/renameat/renameat2
  - mkdir/mkdirat
  - rmdir
  - Other file system syscalls with pointer arguments
- Other syscalls are also affected due to fetching files from file systems.
  - execve/execveat

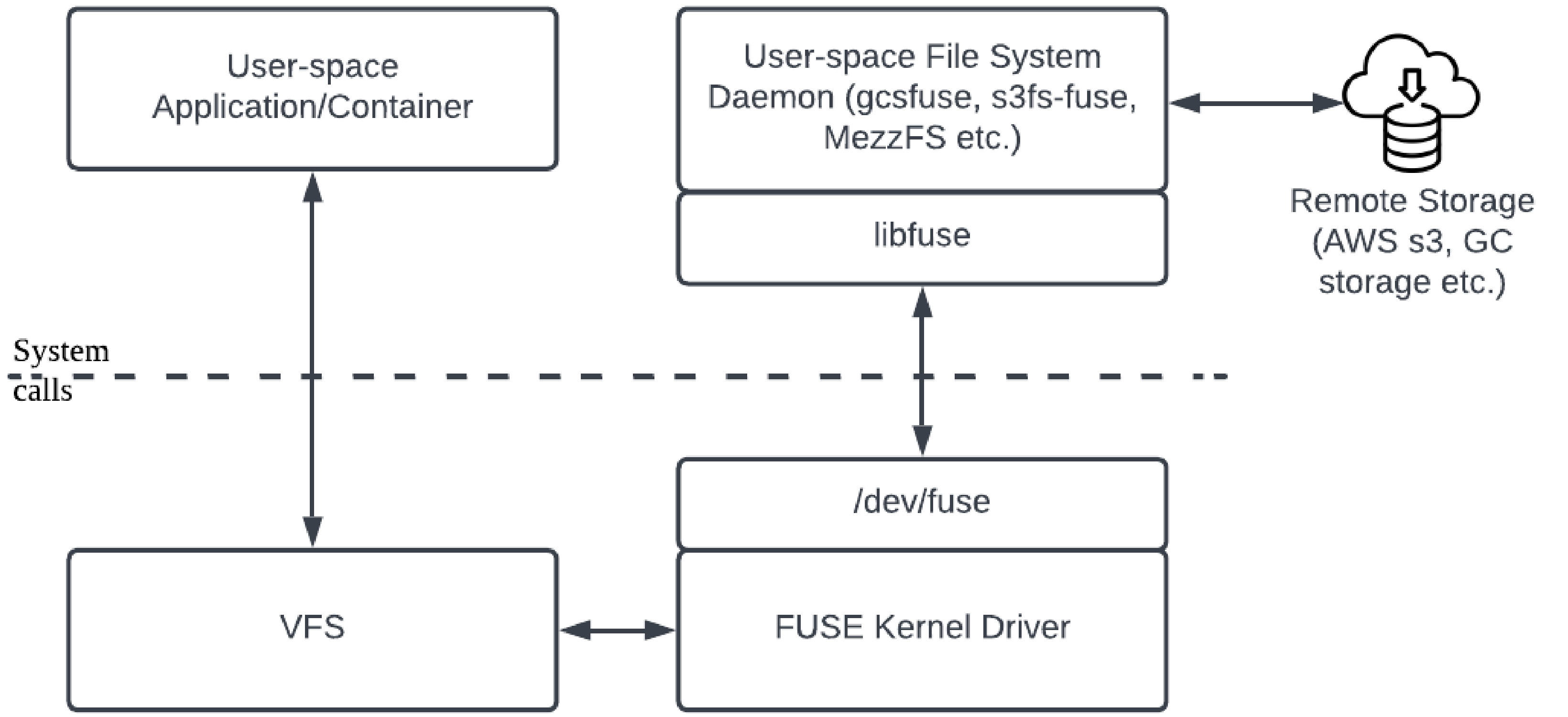
Filesystem in USErspace - FUSE

- User space filesystem framework
- Used as remote storage FUSE
  - Access the remote files as local ones
  - Faster evolvement and don’t panic the kernel etc.
- Remote storage FUSE examples:
  - gcsfuse[1] : developed by Google for GCS
  - s3fs-fuse[2]: Amazon S3
  - BlobFuse[3]: developed by Azure for Blob storage
  - MezzFS[4]: developed and deployed @ Netflix
  - Many others (sshfs etc.)

1, https://github.com/GoogleCloudPlatform/gcsfuse
2, https://github.com/s3fs-fuse/s3fs-fuse
3, https://github.com/Azure/azure-storage-fuse
4, https://netflixtechblog.com/mezzfs-mounting-object-storage-in-netflixs-media-processing-platform-cda01c446ba

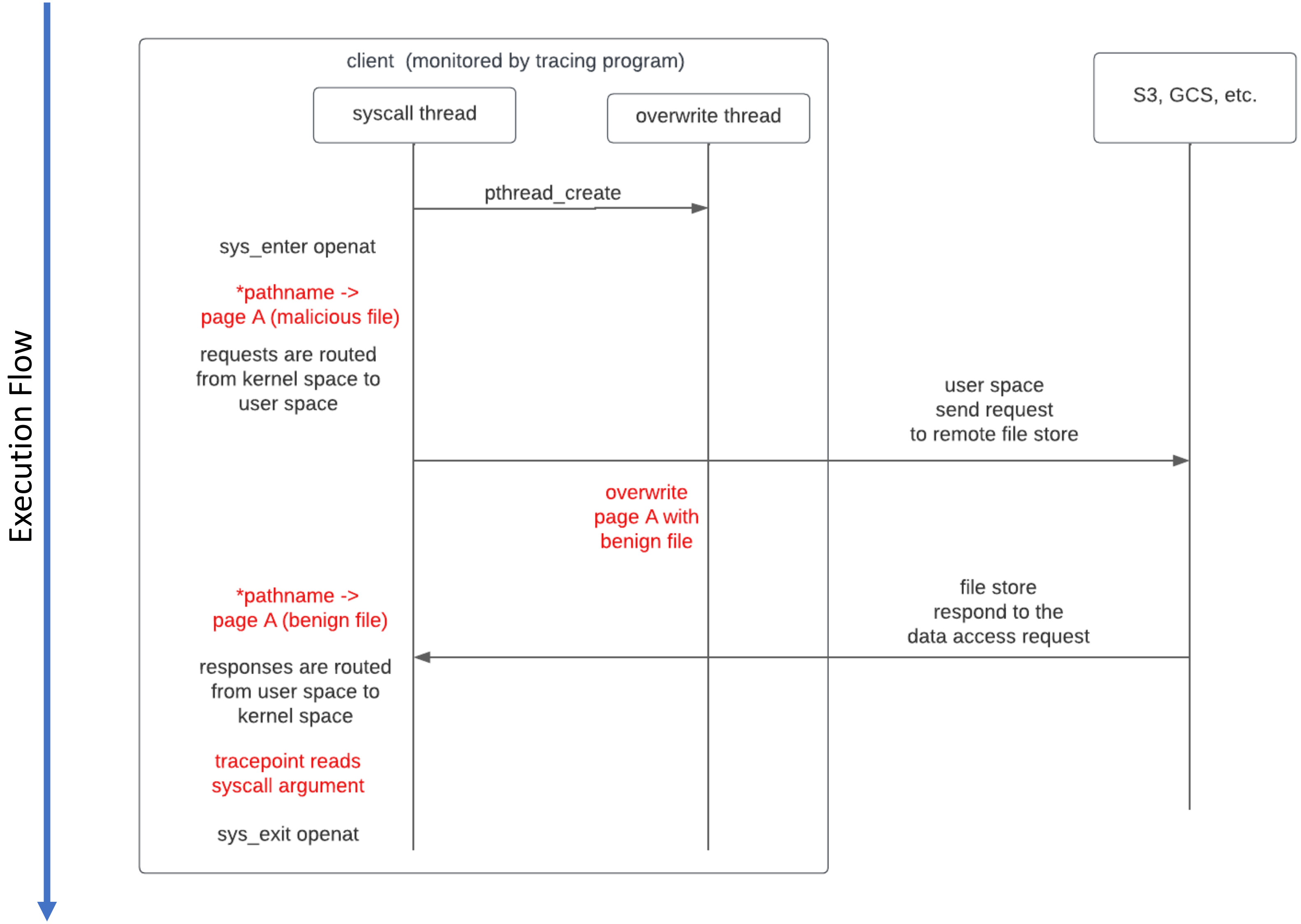
LACEWORK

Remote Storage FUSE - Architecture



LACEWORK

Bypassing Openat Tracing (Demo)



LACEWORK

TOCTOU – sys_enter (Connect)

syscall enter

User Space

Kernel 5.7.0

Execution Flow

```
ptrace_report_syscall(regs, message)
__secure_computing(struct seccomp_data{regs…})
trace_sys_enter(regs, regs->orig_ax)
```

ptrace

seccomp

Syscall Table (x86_64)
…
42 sys_connect
43 sys_accept
44 sys_sendto
…

```
long __sys_connect((int fd,
      struct sockaddr __user *uservaddr, int addrlen))
{
    …
    struct filename *tmp;
    ret = move_addr_to_kernel
            (uservaddr, addrlen, &address);
    if (!ret)
      ret = __sys_connect_file
            (f.file, &address, addrlen, 0);

    …
}
```

TOU by Linux Kernel

```
trace_sys_exit(regs, regs->ax)
ptrace_report_syscall(regs, message)
```

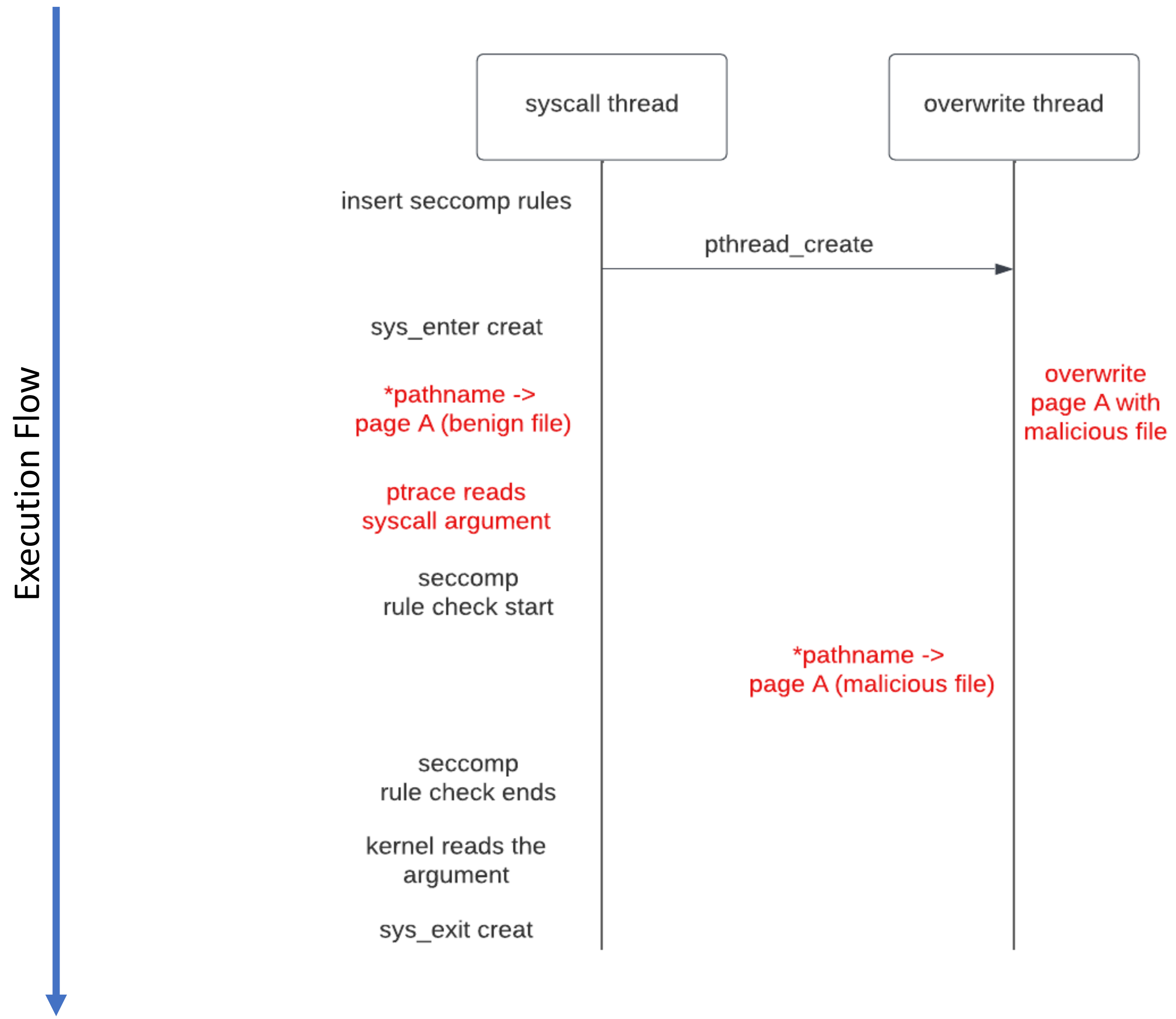
syscall exit

LACEWORK

Seccomp Introduction

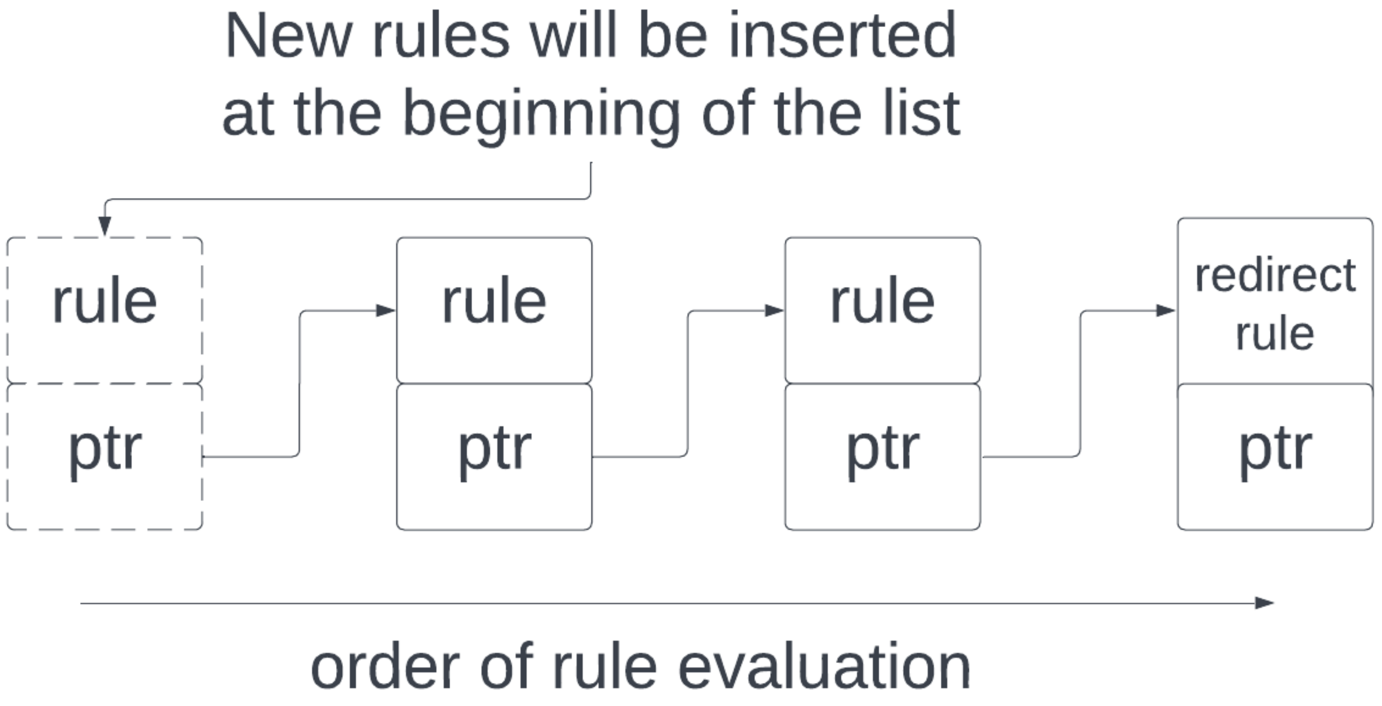
- Kernel level mechanism to restrict syscalls
- Modern sandboxes heavily relies on seccomp
- Developers can write rules to:
  - allow/block certain syscalls
  - allow/block syscalls based on argument values
- These rules can be quite complex (read more)
  - More rules takes more time to compute
- First inserted rules are evaluated last

Attacking Syscall Enter



ptrace + seccomp redirect

- Tracer starts App



Exploitation and Mitigations

Tracing location	**TOCTOU Exploitation**	**Mitigations**
ptrace @ sys_enter	Seccomp filter insertion	- ptrace + seccomp redirect to start the app. - Inspect seccomp filters already inserted when attaching to a running app
tracepoint @ sys_enter	Unreliable	N/A
tracepoint @ sys_exit	- Blocking syscall (This talk) - Phantom attack v1 (DEFCON 29)	- Compare tracepoint sys_enter and sys_exit args
ptrace @ sys_exit	Same as above	- Deploy all mitigations for ptrace @ sys_enter - Compared the sys_enter and sys_exit syscall args
kprobe @ kernel internal	It depends	Read the kernel copy of the syscall args - LSM (BPF-LSM) - Other interfaces

Key Takeaways

1. Linux kernel tracing can be bypassed reliably
   - Check your security tools
2. Mitigation is complex (workload type and kernel compatibility)
   - Check your security tools' mitigation claims
3. Correlate different data sources
4. Know your normal

- Discussing further?
  - @Xiaofei_REX / rex.guo *NOSPAM* lacework DOT com
  - jzeng04 *NOSPAM* gmail DOT com
  - POC: https://github.com/rexguowork/phantom-attack

LACEWORK

Acknowledgement

- Joel Schopp (Linux kernel / Security)
- Lacework Labs
  - James Condon
  - Greg Foss
  - Chris Hall
  - Jared Stroud
- Falco open source team
  - Leonardo Di Donato
  - Michael Clark
  - Michele Zuccala
  - Luca Guerra
- John Dickson

LACEWORK