



Trace Me If You can:

Bypassing Linux Syscall Tracing

Dr. Rex Guo, Lacework
Dr. Junyuan Zeng, [Linkedin.com](https://www.linkedin.com/in/junyuan-zeng/)

About Rex Guo

- Principal Engineer at Lacework
 - Runtime detections and polygraph
 - CSPM
- Engineering Manager
 - XDR start up
 - CWPP start up (now part of Cisco)
- Conference speaker at Blackhat, DEFCON,...
- @Xiaofei_REX



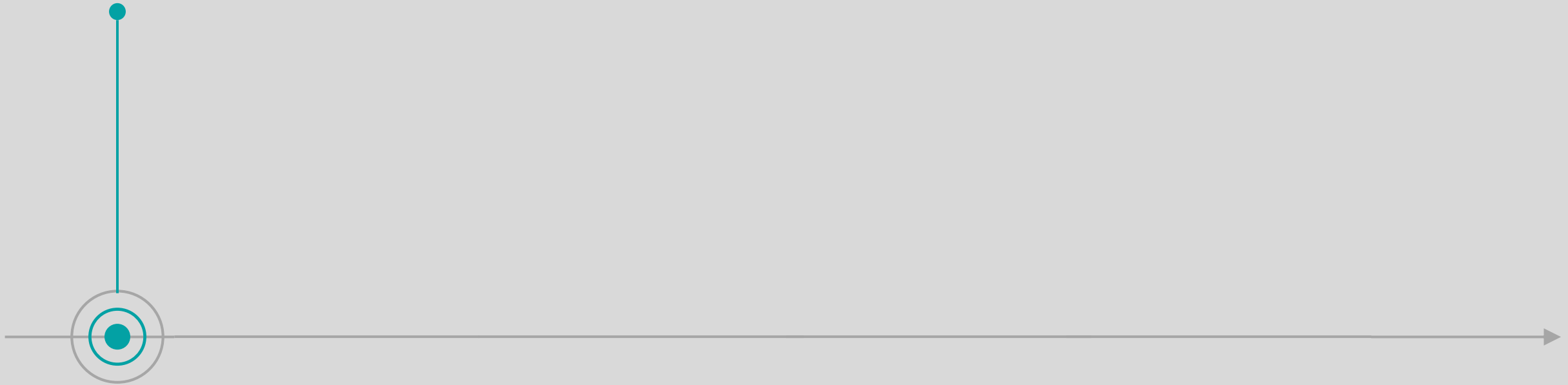
About Junyuan Zeng

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



An Incident - An Attacker's View

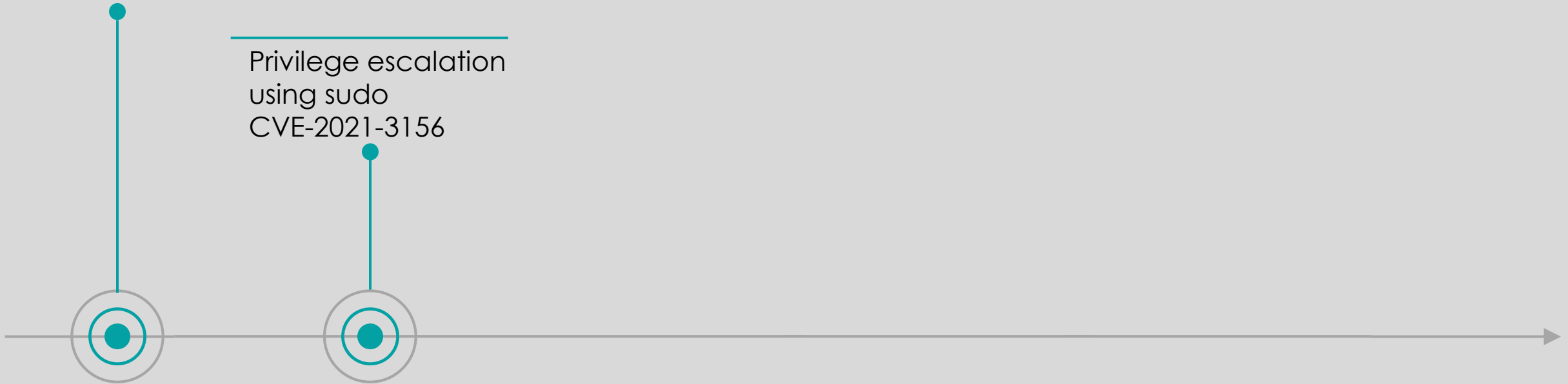
Log4shell RCE on
joe-box and
executed a reverse
shell



An Incident - An Attacker's View

Log4shell RCE on
joe-box and
executed a reverse
shell

Privilege escalation
using sudo
CVE-2021-3156

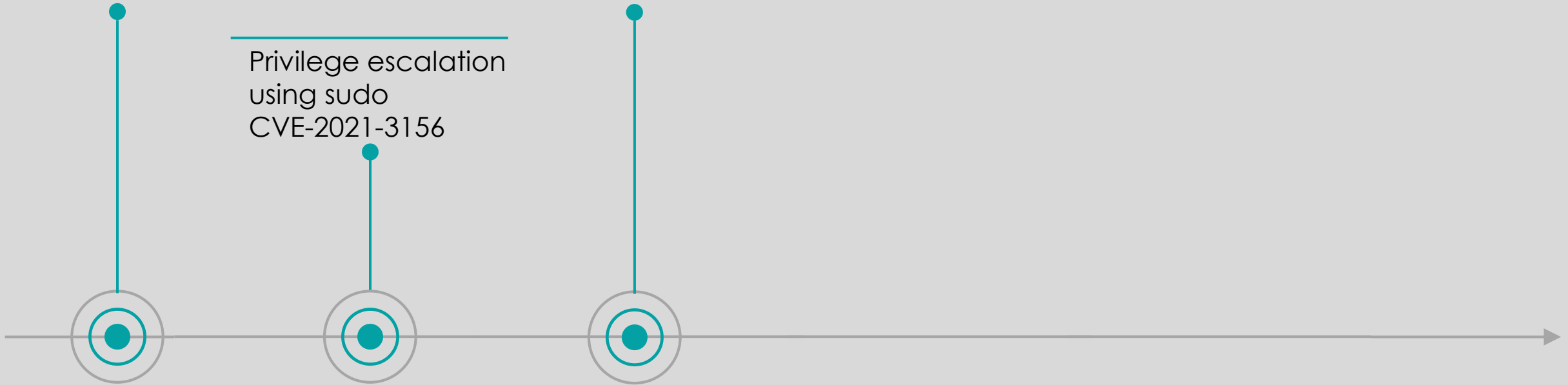


An Incident - An Attacker's View

Log4shell RCE on
joe-box and
executed a reverse
shell

read /etc/shadow

Privilege escalation
using sudo
CVE-2021-3156



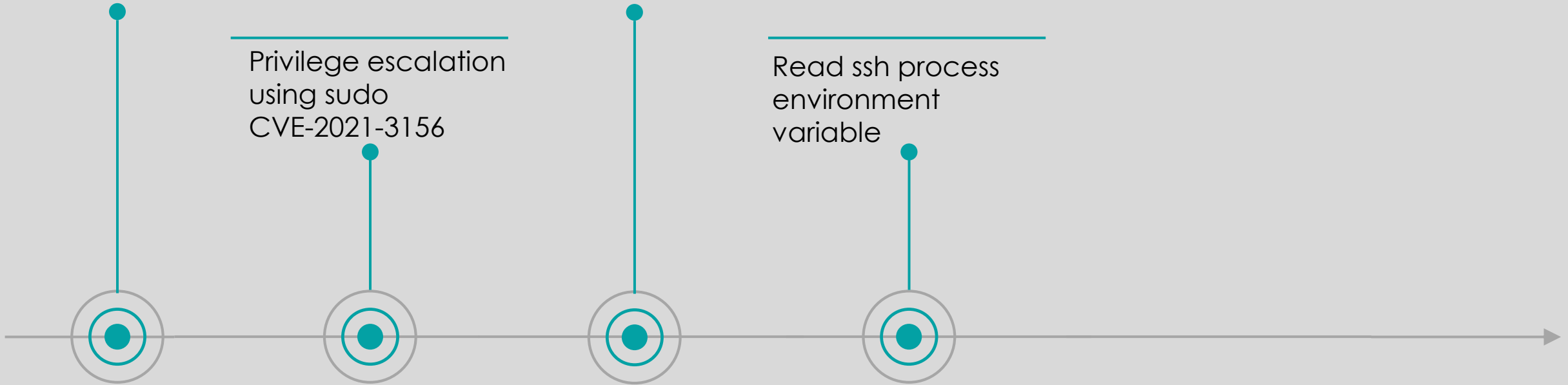
An Incident - An Attacker's View

Log4shell RCE on
joe-box and
executed a reverse
shell

read /etc/shadow

Privilege escalation
using sudo
CVE-2021-3156

Read ssh process
environment
variable



An Incident - An Attacker's View

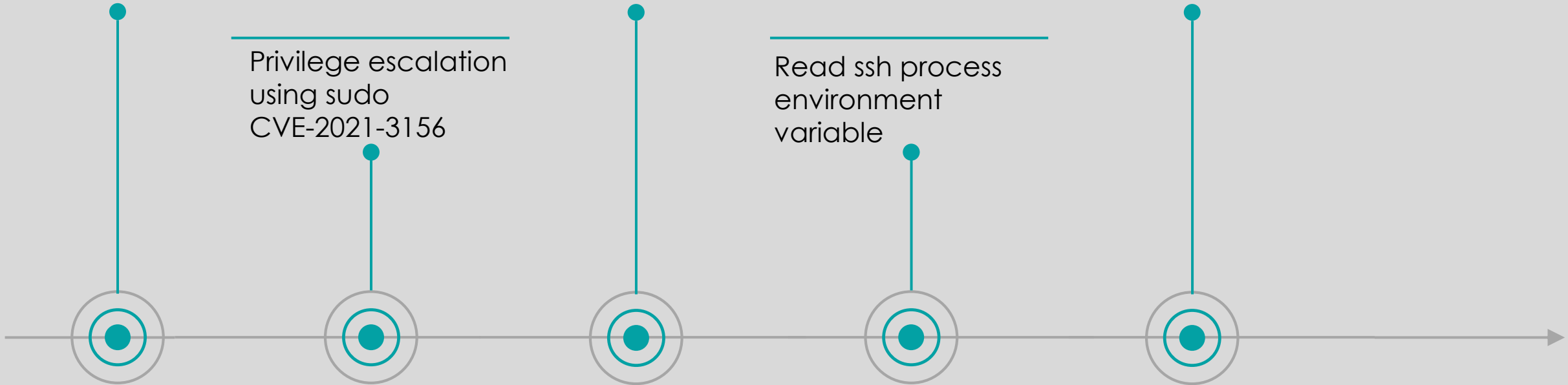
Log4shell RCE on
joe-box and
executed a reverse
shell

read /etc/shadow

Lateral movement
to alice-box with
ssh hijacking

Privilege escalation
using sudo
CVE-2021-3156

Read ssh process
environment
variable



An Incident - An Attacker's View

Log4shell RCE on
joe-box and
executed a reverse
shell

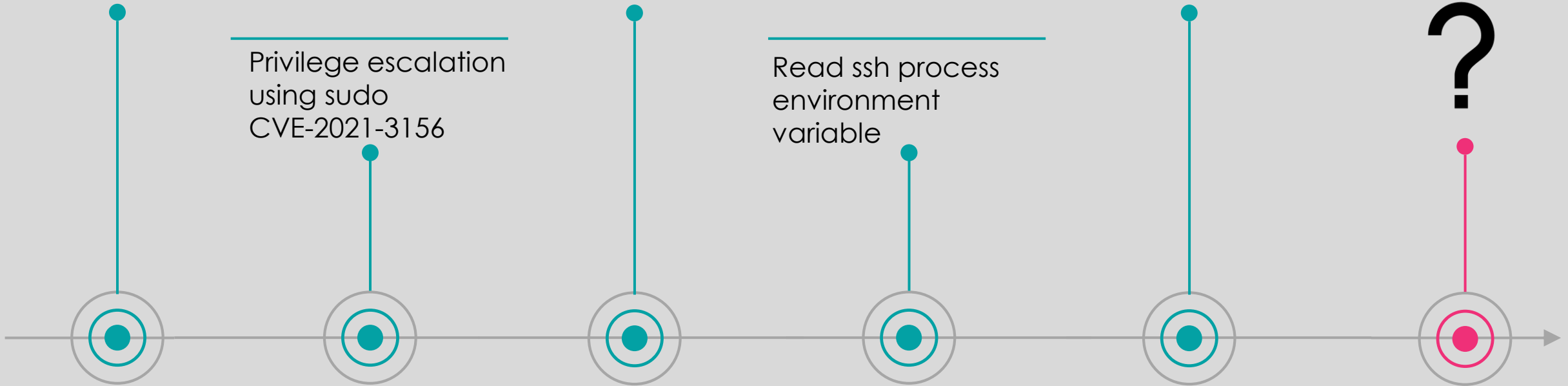
read /etc/shadow

Lateral movement
to alice-box with
ssh hijacking

Privilege escalation
using sudo
CVE-2021-3156

Read ssh process
environment
variable

?



An Incident - A Defender's View

Log4shell RCE on
joe-box and
executed a reverse
shell

read /etc/shadow

Lateral movement
to alice-box with
ssh hijacking

Privilege escalation
using sudo
CVE-2021-3156

Read ssh process
environment
variable

connect
etc.

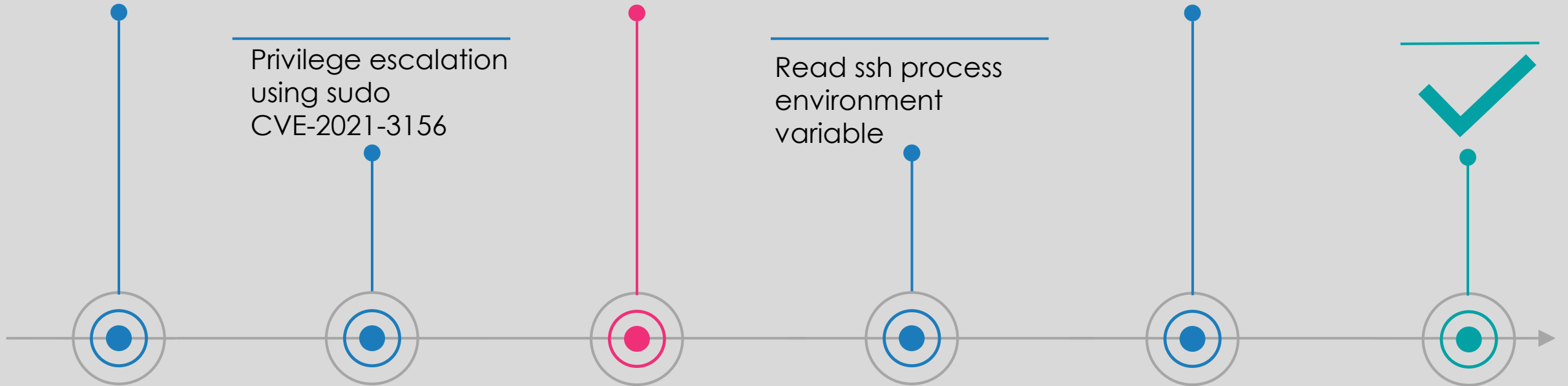
execve(at)

open(at)

open(at)

connect,
etc.

Incident
Response



Detection Rule Example

rule: untrusted program reads */etc/shadow*

condition:

syscall == open(at)

and has read permission

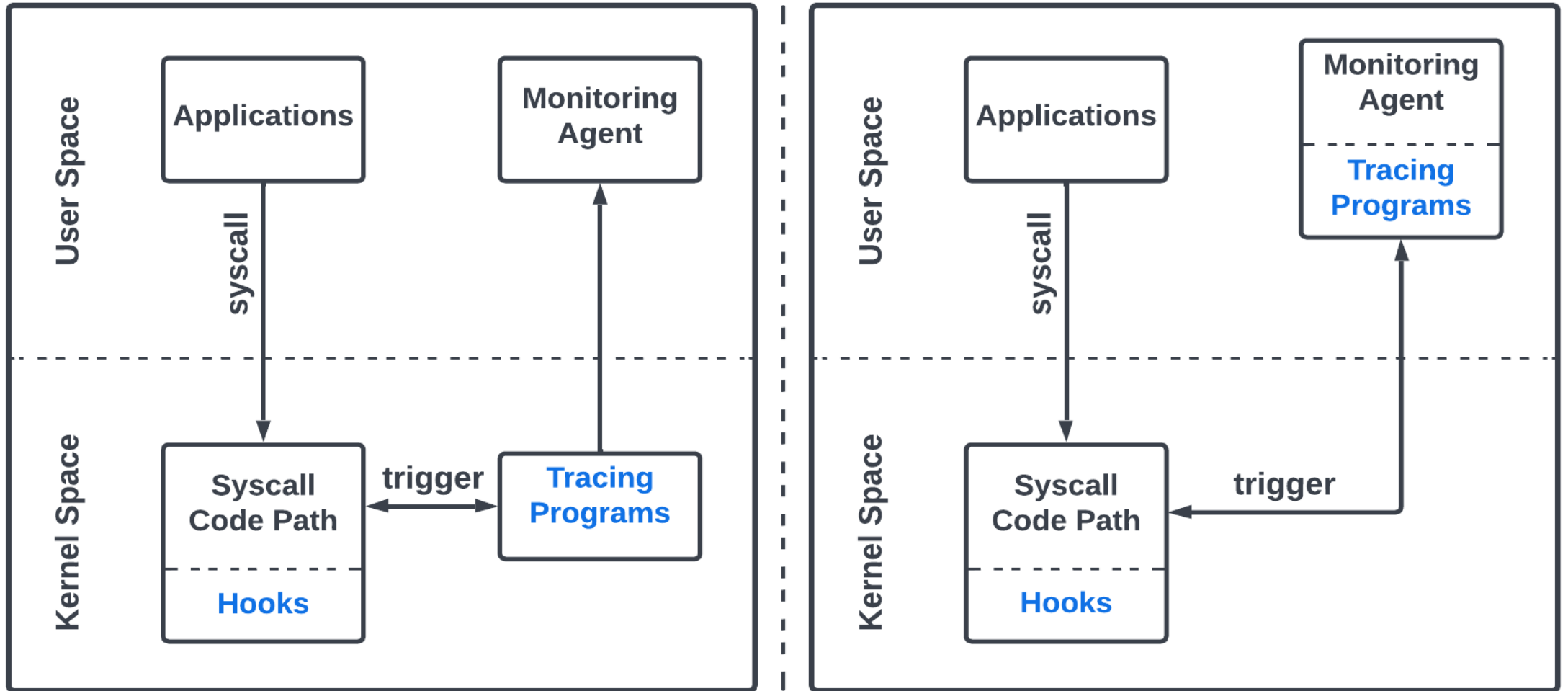
and filename == */etc/shadow*

and program is not in allowlist

Agenda

- Introduction to syscall tracing
- TOCTOU in syscall tracing software
- Bypassing networking syscall tracing
- Bypassing file systems syscall tracing
- Bypassing syscall tracing at `sys_enter`
- Mitigations
- Takeaways

System Call Tracing

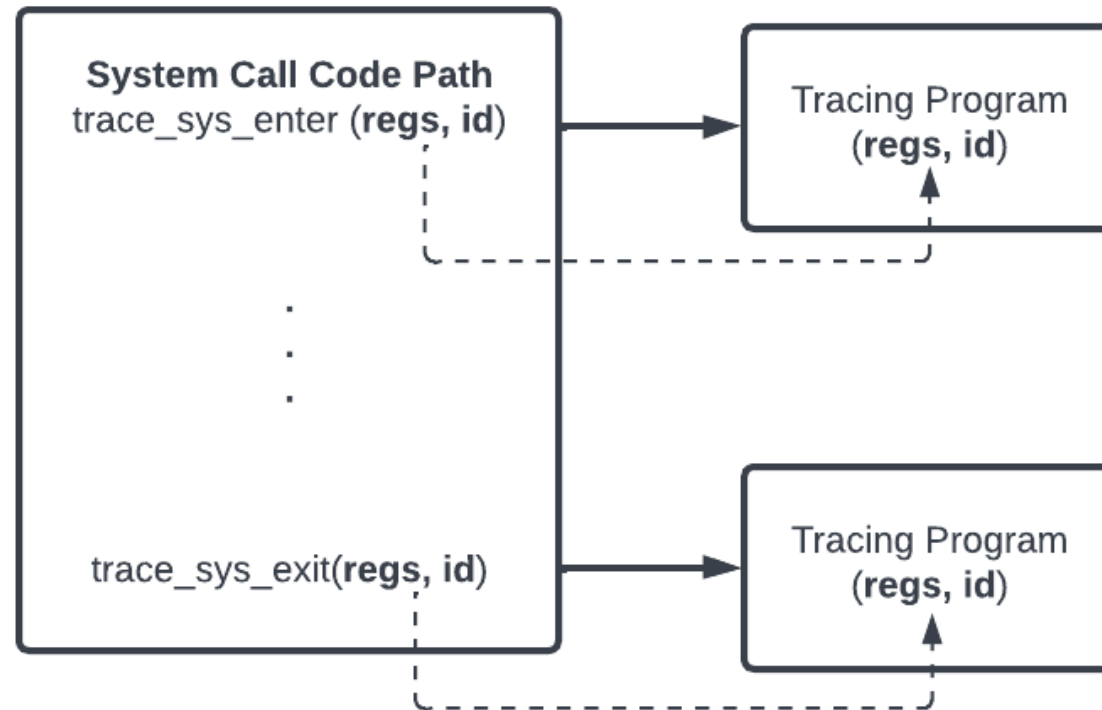


System Call Tracing – Tracing Program

- 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 programs and user space programs

System Call Tracing – tracepoint

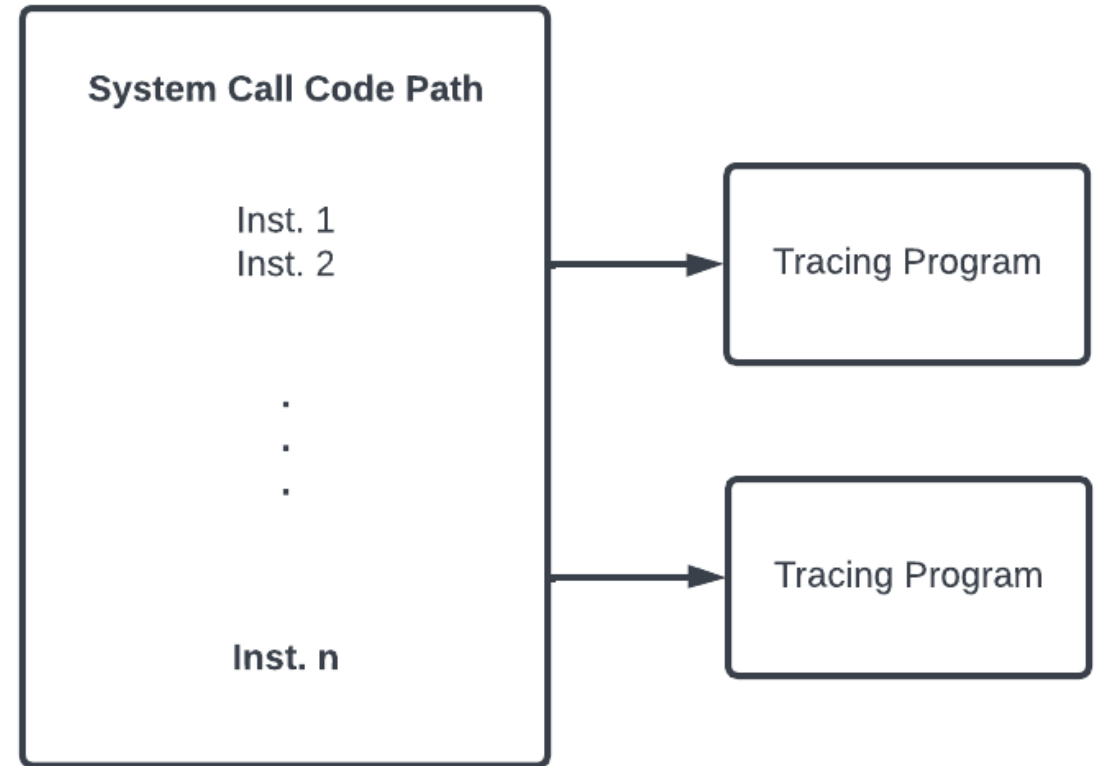
- tracepoint
 - Kernel static hook
 - Linux kernel provides *sys_enter* and *sys_exit*



- 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	<ul style="list-style-type: none">• Hooks: tracepoint, kprobe, ptrace• Tracing programs: kernel programs (eBPF, kernel Module), user programs• Tools: Falco eBPF/kernel Module, Falco pdig
Containers	<ul style="list-style-type: none">• Hooks: tracepoint, kprobe, ptrace• Tracing programs: kernel programs (eBPF, kernel Module), user programs• Tools: Falco eBPF/kernel Module, Falco pdig
Serverless Containers	<ul style="list-style-type: none">• Hooks: ptrace• Tracing programs: user programs• Tools: Falco pdig

Open Source Projects

- Falco (created by Sysdig)
 - 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 (created by Sysdig)
 - Support syscall tracing of serverless workloads

TOCTOU Vulnerabilities

- time-of-check, time-of-use
- TOC: tracing programs collect system call data (e.g. arguments)
- TOU: system call data used by kernel is different from what tracing programs check
- *sys_connect(int fd, struct sockaddr __user * **uservaddr**, int addrlen)*
 - *uservaddr* points to “socket address” stored in user-space buffer

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 *servaddr, int addrlen))
{
    ...
    struct filename *tmp;
    ret = move_addr_to_kernel
        (servaddr, 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

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 *useraddr, int addrlen))
{
    ...
    struct filename *tmp;
    ret = move_addr_to_kernel
        (useraddr, addrlen, &address);
    if (!ret)
        ret = __sys_connect_file
            (f.file, &address, addrlen, 0);
    ...
}
```

syscall exit

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

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 *servaddr, int addrlen))
{
    ...
    struct filename *tmp;
    ret = move_addr_to_kernel
        (servaddr, 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

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)
```

■ Userspace pointer pointing to “socket address”

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

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 *useraddr, int addrlen))
{
    ...
    struct filename *tmp;
    ret = move_addr_to_kernel
        (useraddr, addrlen, &address);
    if (!ret)
        ret = __sys_connect_file
            (f.file, &address, addrlen, 0);
    ...
}
```

■ Userspace pointer pointing to “socket address”

■ Kernel pointer pointing to “socket address”

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

syscall exit

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 *useraddr, int addrlen))
{
    ...
    struct filename *tmp;
    ret = move_addr_to_kernel
        (useraddr, 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

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 *useraddr, int addrlen))
{
    ...
    struct filename *tmp;
    ret = move_addr_to_kernel
        (useraddr, 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

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)
```

sys_enter tracepoint
ptrace

TOC by Tracing
Programs

kprobe

Syscall Table (x86_64)

```
...
42 sys_connect
43 sys_accept
44 sys_sendto
...
```

```
long __sys_connect((int fd,
    struct sockaddr __user *useraddr, 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

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 *useraddr, int addrlen))
{
    ...
    struct filename *tmp;
    ret = move_addr_to_kernel
        (useraddr, 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)
```

sys_exit tracepoint
ptrace

TOC by Tracing
Programs

syscall exit

TOCTOU - Connect



TOCTOU Window across Kernels

- These TOCTOU window exists since the initial release of tracepoint/ptrace
- Kernel developers confirmed that this is “expected behavior”
- Monitor kernel memory (sounds simple, very complex in reality)

tracepoint and ptrace have
TOCTOU issues!

We knew! They are designed
for perf/debug



TOCTOU – Falco

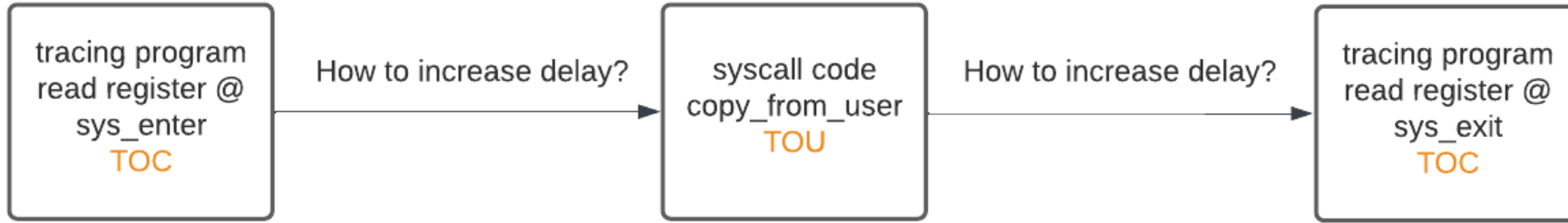
- 12/06/2021 Issue reported ([CVE-2022-26316](#))
- 03/11/2022 Mitigation implemented ([Advisory](#))
- Falco older than v0.31.1
 - Commercial versions based on the open source agent are also affected (confirmed by the open source maintainer)
- User space pointers are dereferenced by
 - `sys_exit` tracepoint (kernel module, eBPF)
 - `sys_exit` ptrace (pdig)

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

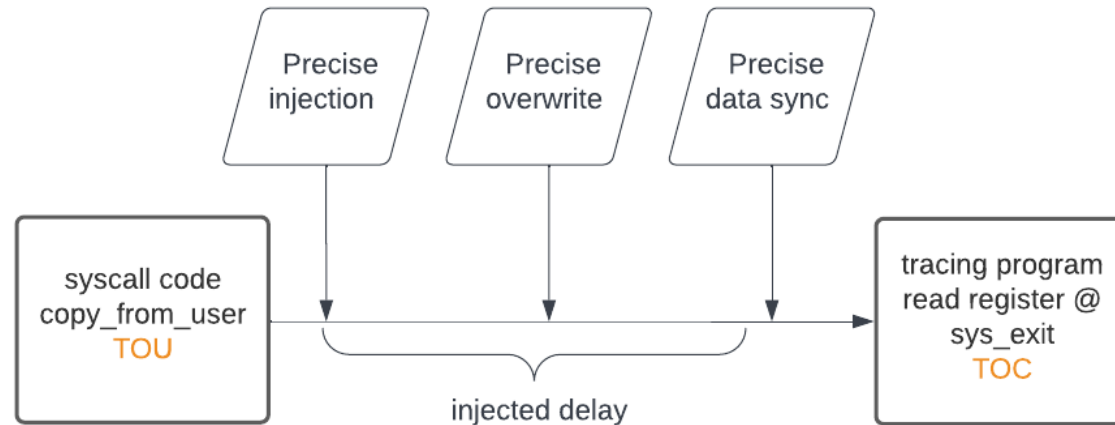
Exploit Requirements



- Exploitation requirements
 - No root privilege or SYS_ADMIN capabilities
 - Control the time to inject the delay
 - Enough delay for pointer overwrite
 - Reliable

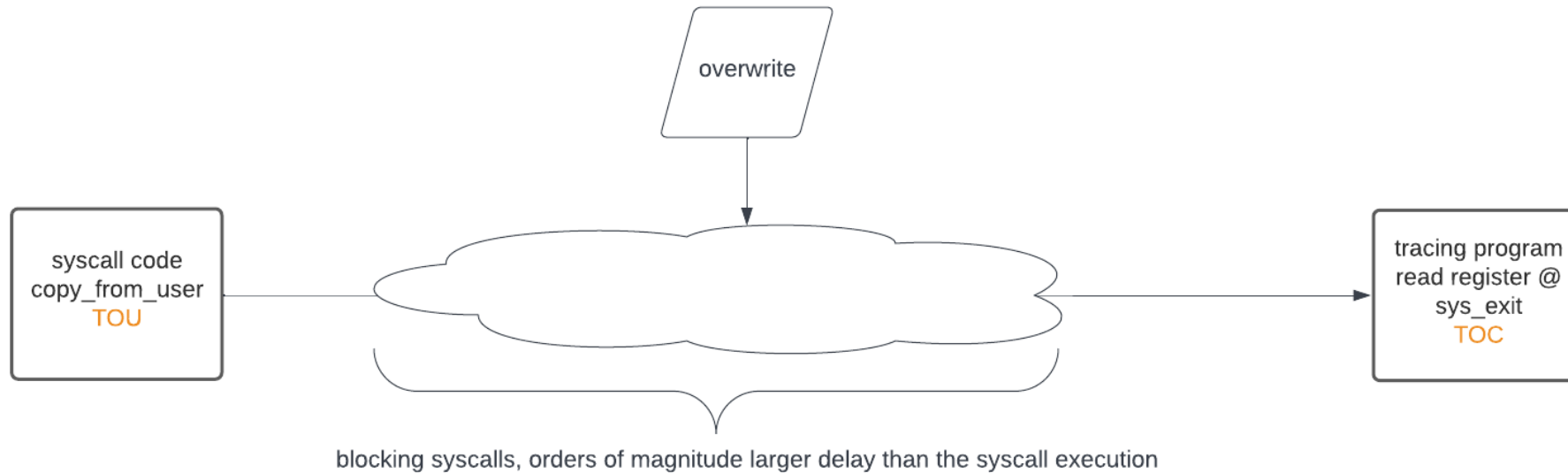
Exploit Strategy 1 (DEFCON 29)

- 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 deployed a detection fix



Exploit Strategy 2

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

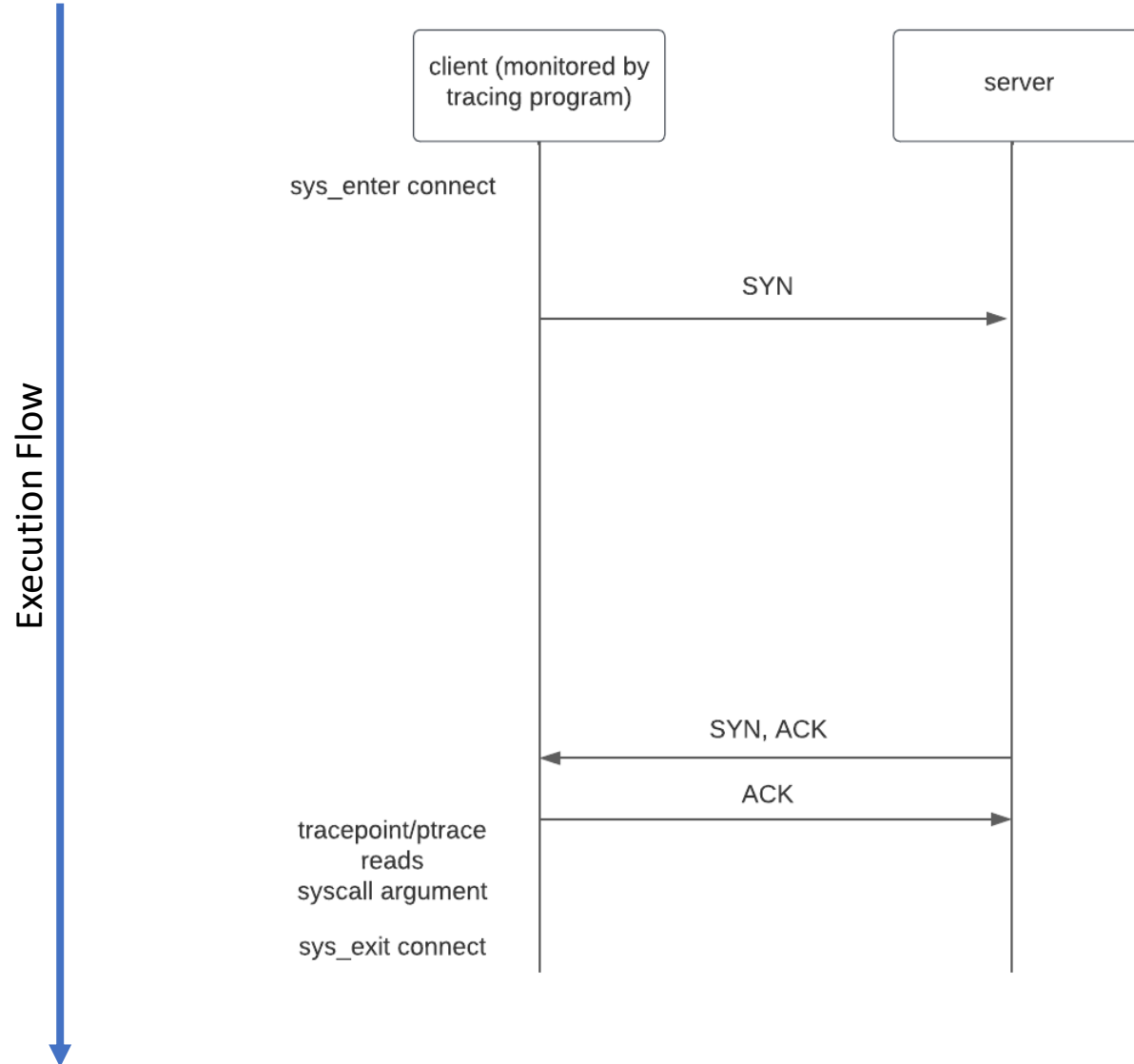


Syscall Built-in Delay

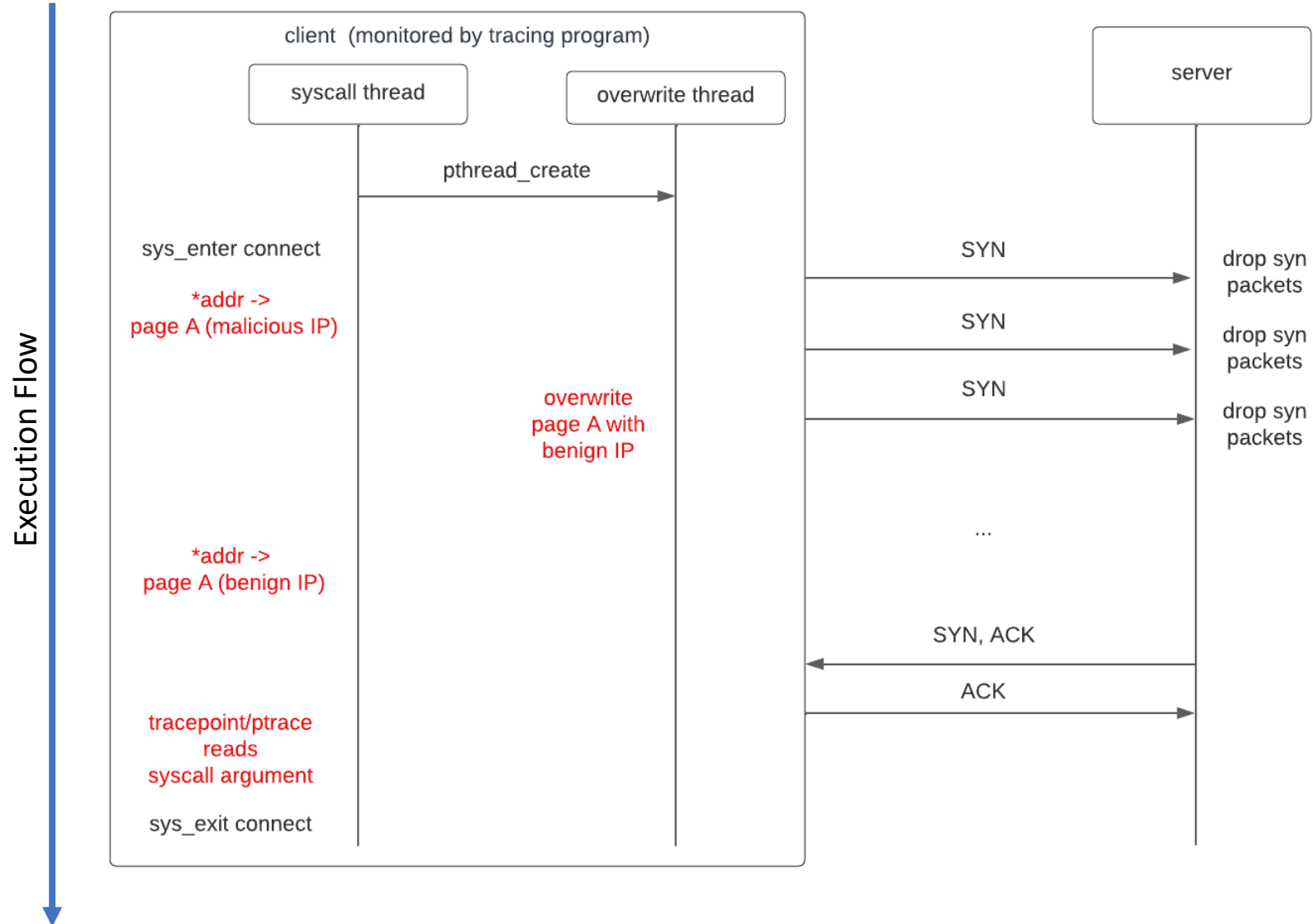
- Attackers can trigger significant syscall delays by introducing:
 - Blocking conditions (attack `sys_exit`)
 - Seccomp rules (attack `sys_enter`)
- Syscall can get “blocked”

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

Connect Syscall



Bypassing Connect Syscall Tracing (Demo)



Blocking Syscalls (File Systems)

- File system related calls implemented by FUSE are all affected
 - open/openat
 - creat
 - rename/renameat/renameat2
 - mkdir/mkdirat
 - rmdir
 - Other file system syscalls
- 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 evolvment and don't panic the kernel etc.
- FUSE examples:
 - gcsfuse¹ : developed by Google for GCS
 - s3fs-fuse²: Amazon S3
 - BlobFuse³: developed by Azure for Blob storage
 - MezzFS⁴: developed and deployed @ Netflix
 - Many others (sshfs etc.)

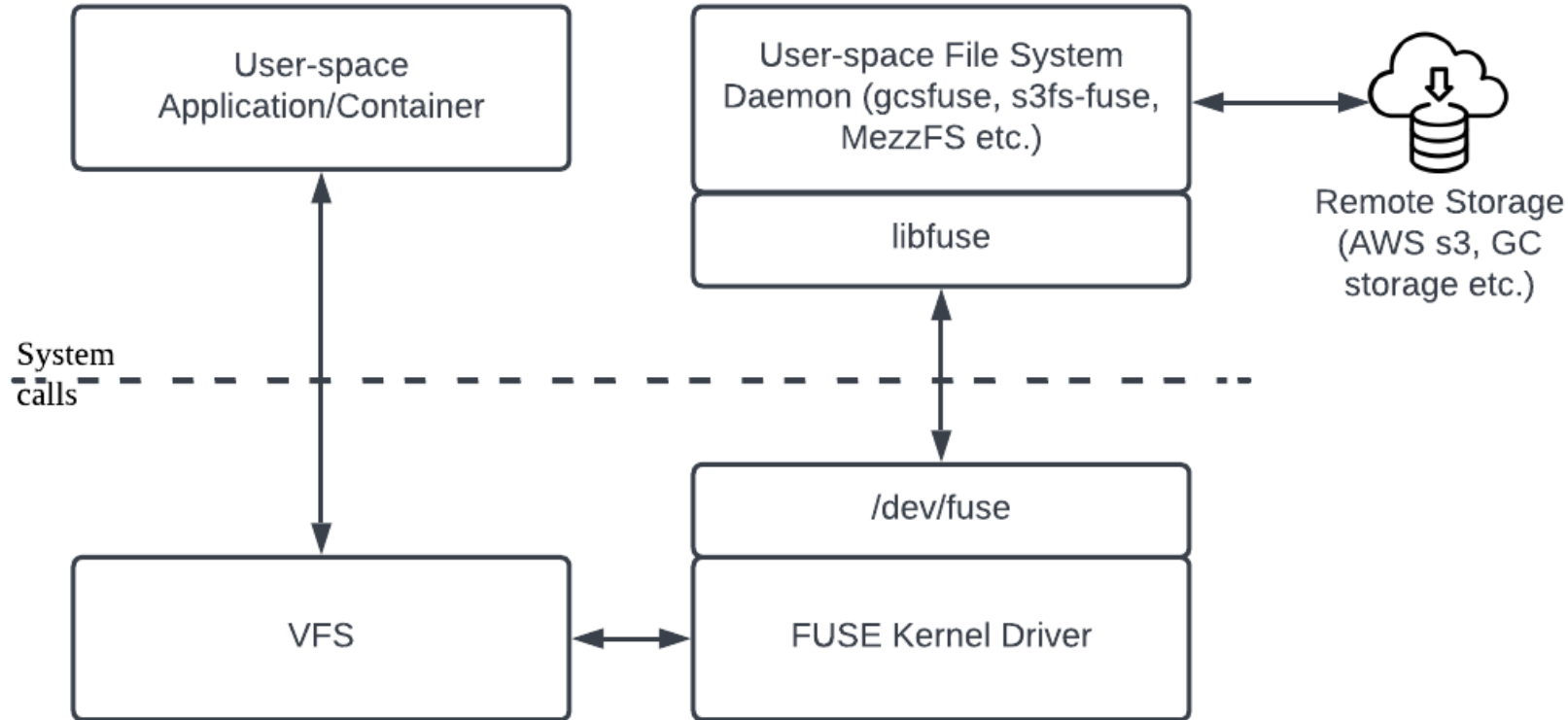
¹, <https://github.com/GoogleCloudPlatform/gcsfuse>

², <https://github.com/s3fs-fuse/s3fs-fuse>

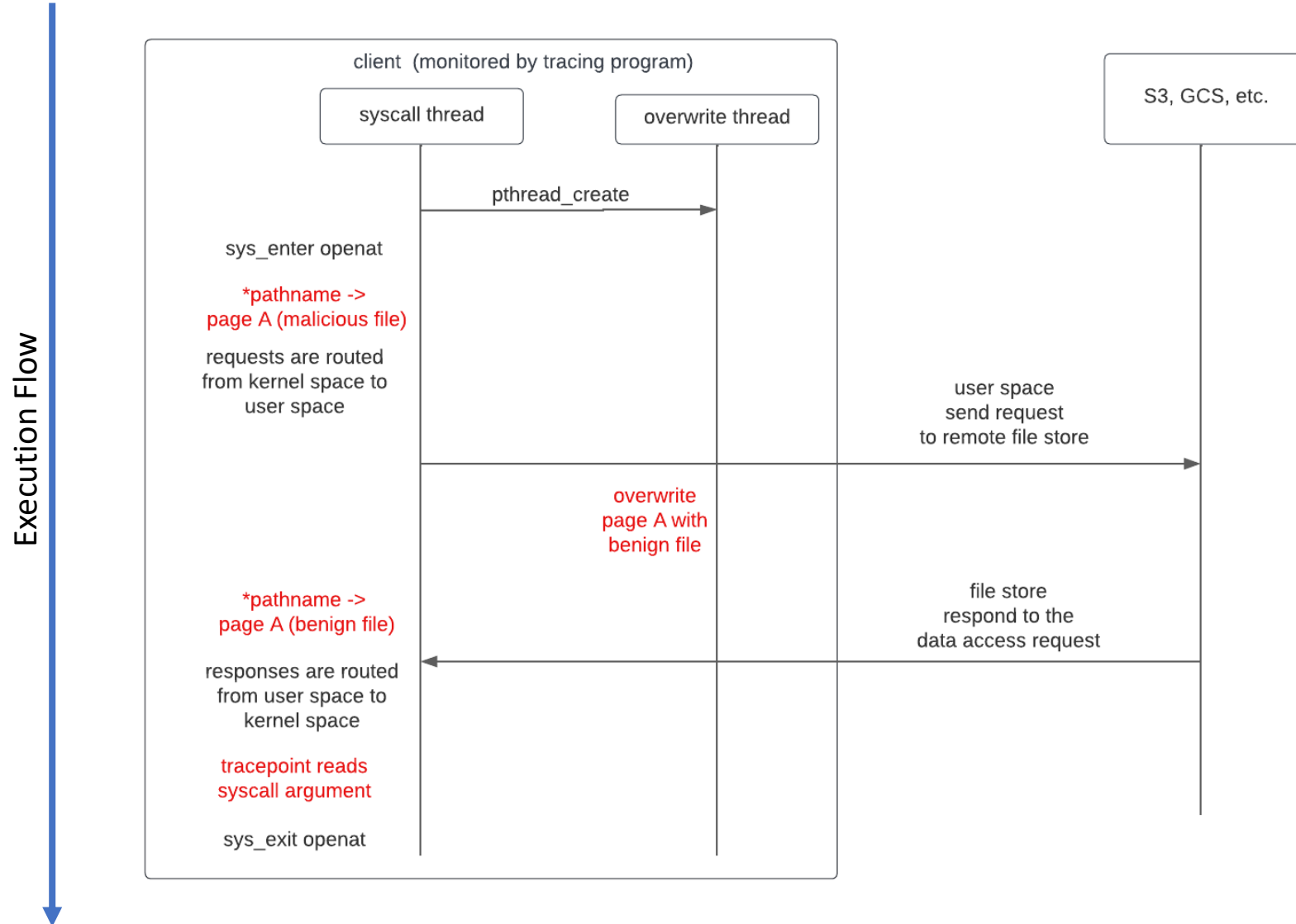
³, <https://github.com/Azure/azure-storage-fuse>

⁴, <https://netflixtechblog.com/mezzfs-mounting-object-storage-in-netflixs-media-processing-platform-cda01c446ba>

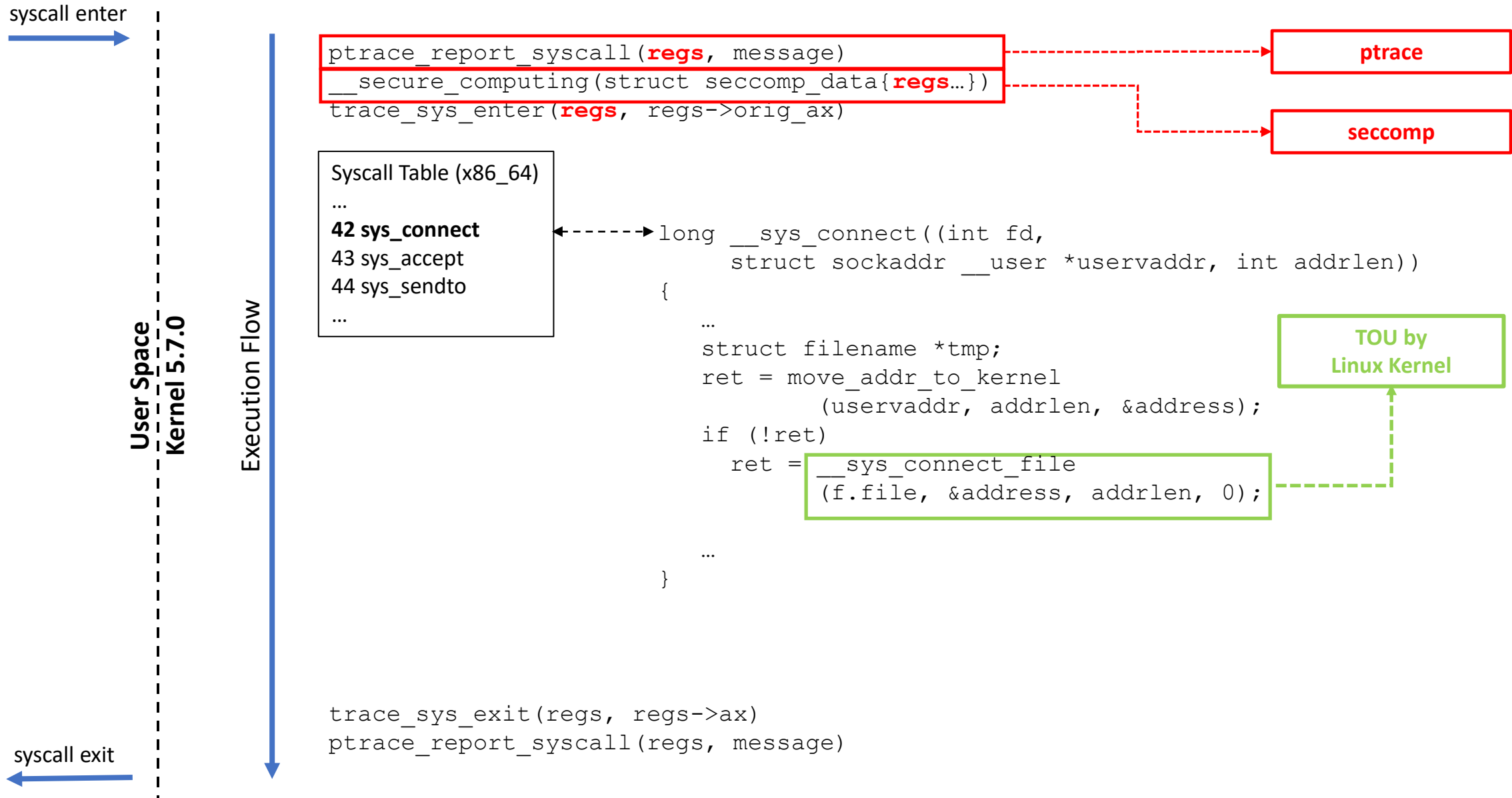
Remote Storage FUSE - Architecture



Bypassing Openat Tracing (Demo)



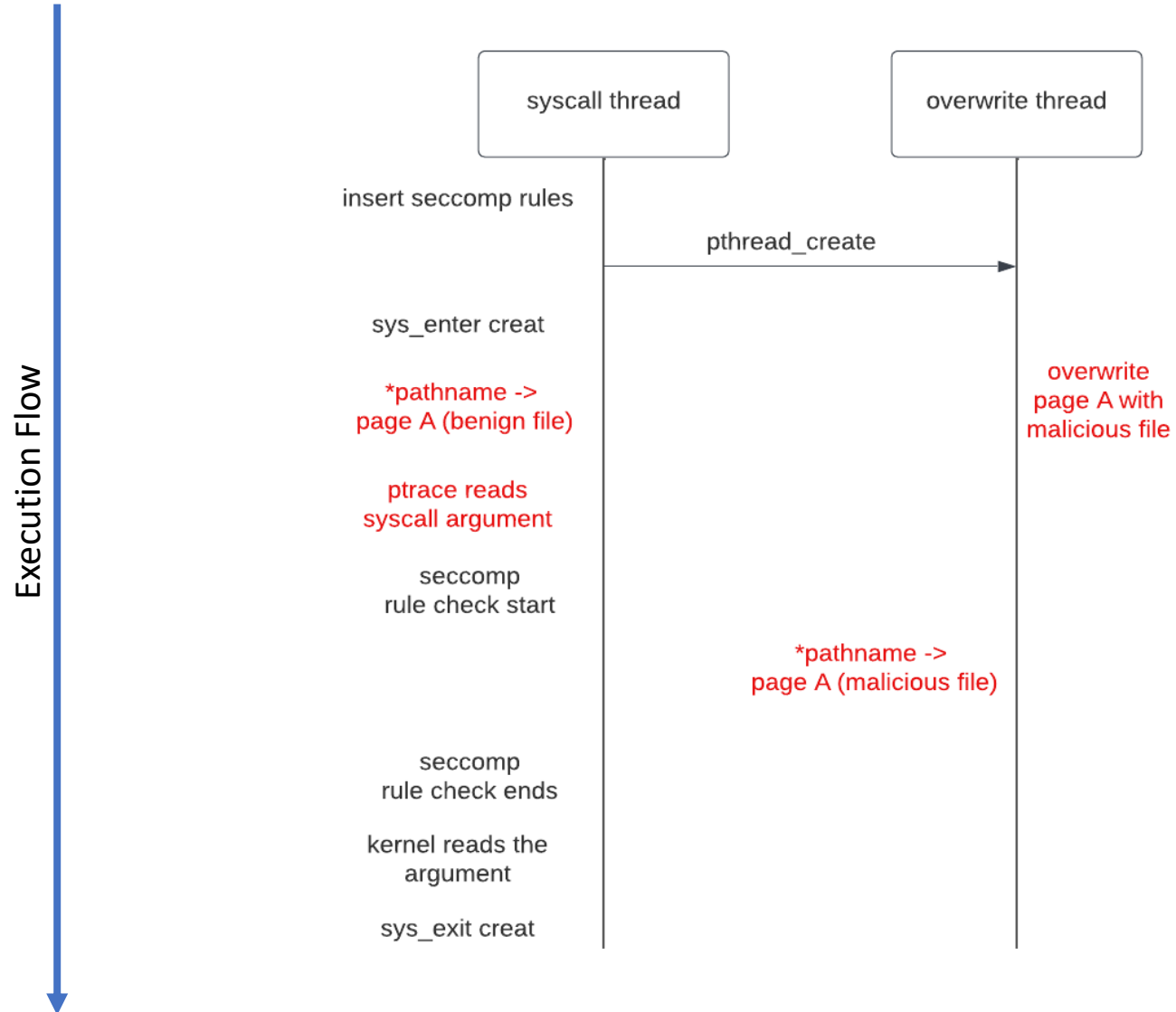
TOCTOU – sys_enter (Connect)



Seccomp Introduction

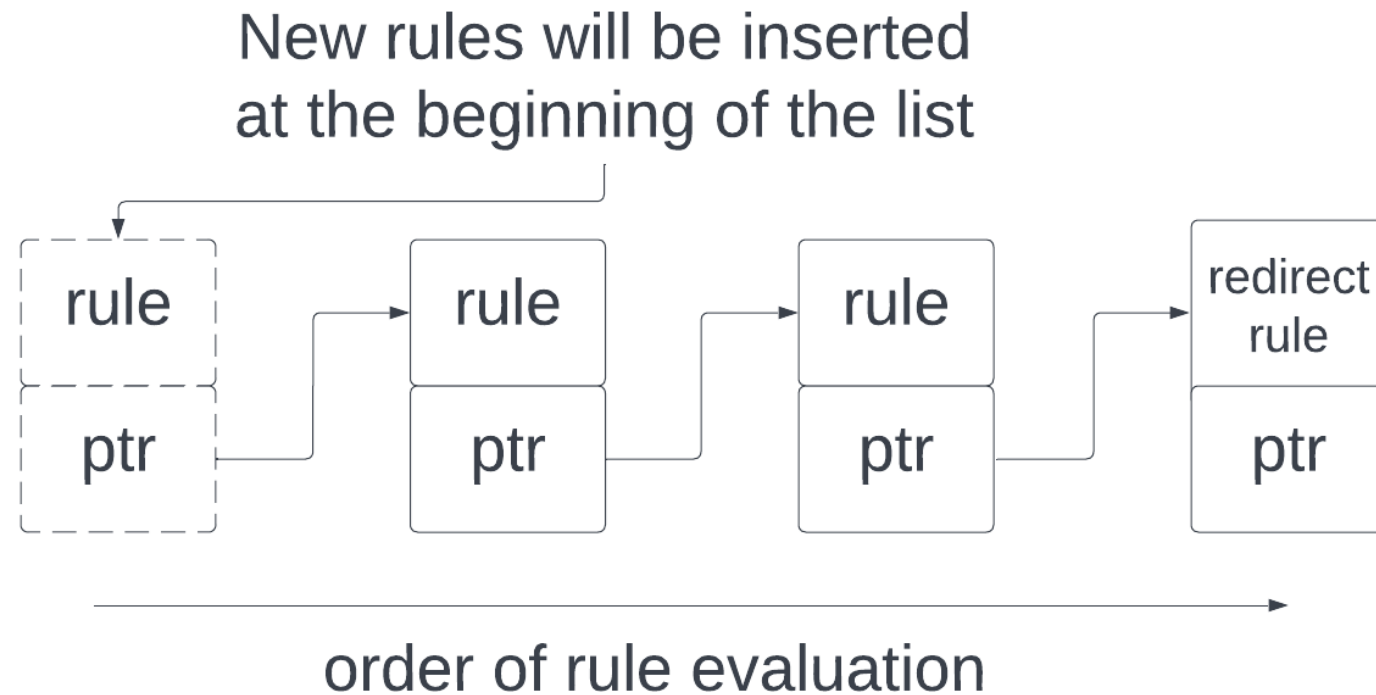
- Kernel level mechanism to restrict syscalls
- Modern sandboxes heavily relies on seccomp
- Developers can write rules to:
 - allow/block certain syscalls
 - allow/block syscalls based on argument values
- These rules can be quite complex ([read more](#))
 - More rules takes more time to compute
- First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

Attacking Syscall Enter (Demo)



ptrace + seccomp redirect

- Tracer starts App



Exploitation and Mitigations

Tracing location	TOCTOU Exploitation	Mitigations
ptrace @ sys_enter	Seccomp filter insertion	<ul style="list-style-type: none">- 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	<ul style="list-style-type: none">- Blocking syscall (This talk)- Phantom attack v1 (DEFCON 29)	<ul style="list-style-type: none">- Compare tracepoint sys_enter and sys_exit args
ptrace @ sys_exit	Same as above	<ul style="list-style-type: none">- Deploy all mitigations for ptrace @ sys_enter- Compared the sys_enter and sys_exit syscall args
kprobe @ kernel internal	It depends	<p>Read the kernel copy of the syscall args</p> <ul style="list-style-type: none">- LSM (BPF-LSM)- Other interfaces

Blackhat Sound Bytes

- Kernel tracing can be bypassed reliably
 - Blocking syscalls
 - Seccomp
 - Interrupt (DEFCON 29)
- Reducing kernel tracing TOCTOU risks is complex
 - Workload type
 - Kernel compatibility
- Detection in depth:
 - point-based detection can be bypassed
 - Analyze different data sources
- Interested in discussing further?
 - @Xiaofei_REX / jzeng04 *NOSPAM* gmail DOT com
 - POC: <https://github.com/rexguowork/phantom-attack>

Acknowledgement

- Joel Schopp (Linux kernel / Security)
 - <https://www.linkedin.com/in/schopp/>
- Leonardo Di Donato (Falco / Pdig)
 - <https://www.linkedin.com/in/leodidonato/>
- Falco open source team
 - Michael Clark
 - Michele Zuccala
 - Luca Guerra

Additional References

[1] <https://github.com/falcosecurity/libs/commit/7e7a5129b6e58b2c77662b8a9c5609851e29d575>