runtime security analysis through syscalls

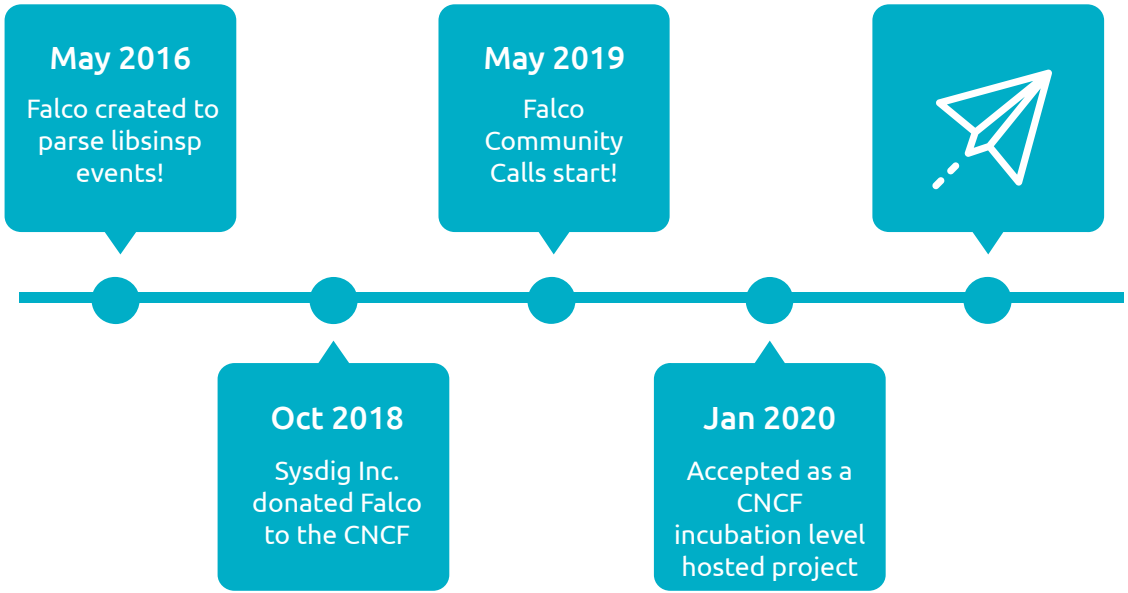
Falco

gh:falcosecurity/falco

ROMHACK 2020

Sunday 27th of September

A timeline always works fine



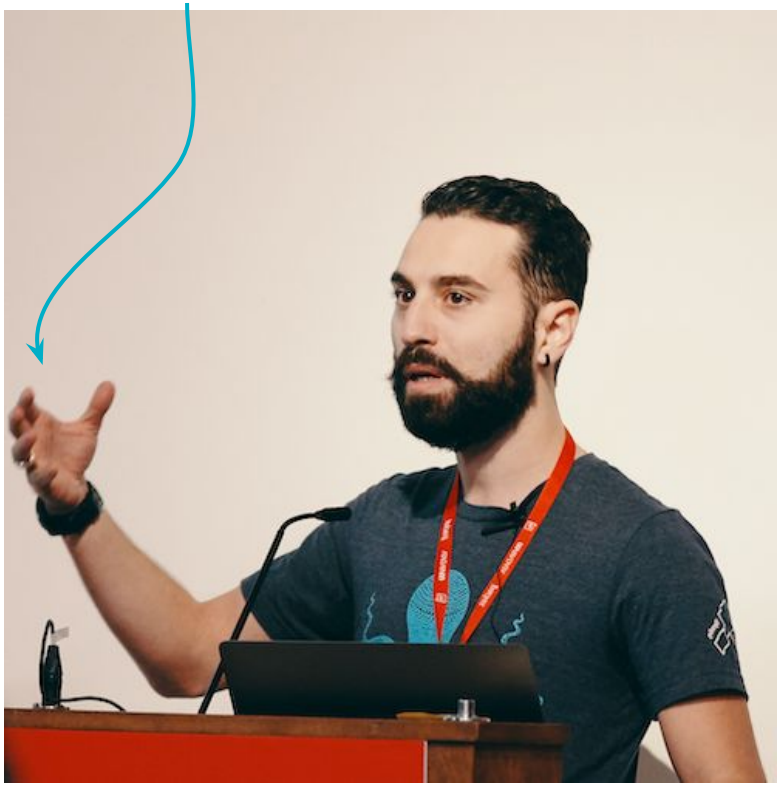
whoami

Leonardo Di Donato

Open Source Software Engineer
Falco Maintainer
Sysdig

@leodido

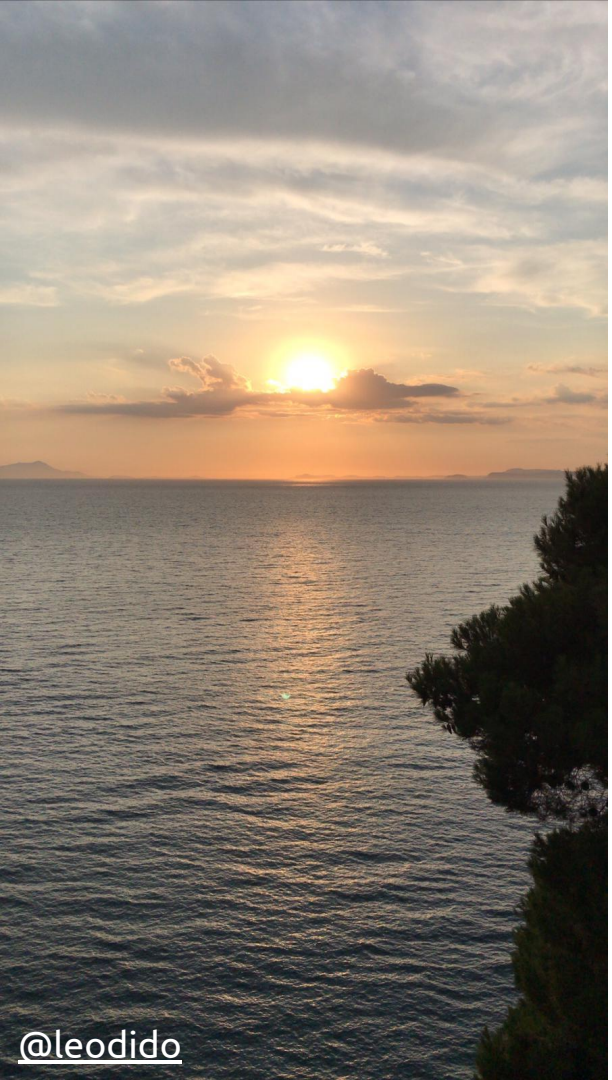
extra points to who spots the meaning of this Italian hand-gesture! 😄



sysdig

Falco

CLOUD NATIVE
COMPUTING FOUNDATION



Contents

Security

PREVENTION

Use **policies** to *change the behavior* of a process by preventing syscalls from succeeding (also killing the process sometimes).

DETECTION

Use **policies** to *monitor the behavior* of a process and notify when its behavior steps outside the policy.

Security

ENFORCEMENT

sandboxing, access control

- seccomp
- seccomp-bpf
- SELinux
- AppArmor

AUDITING

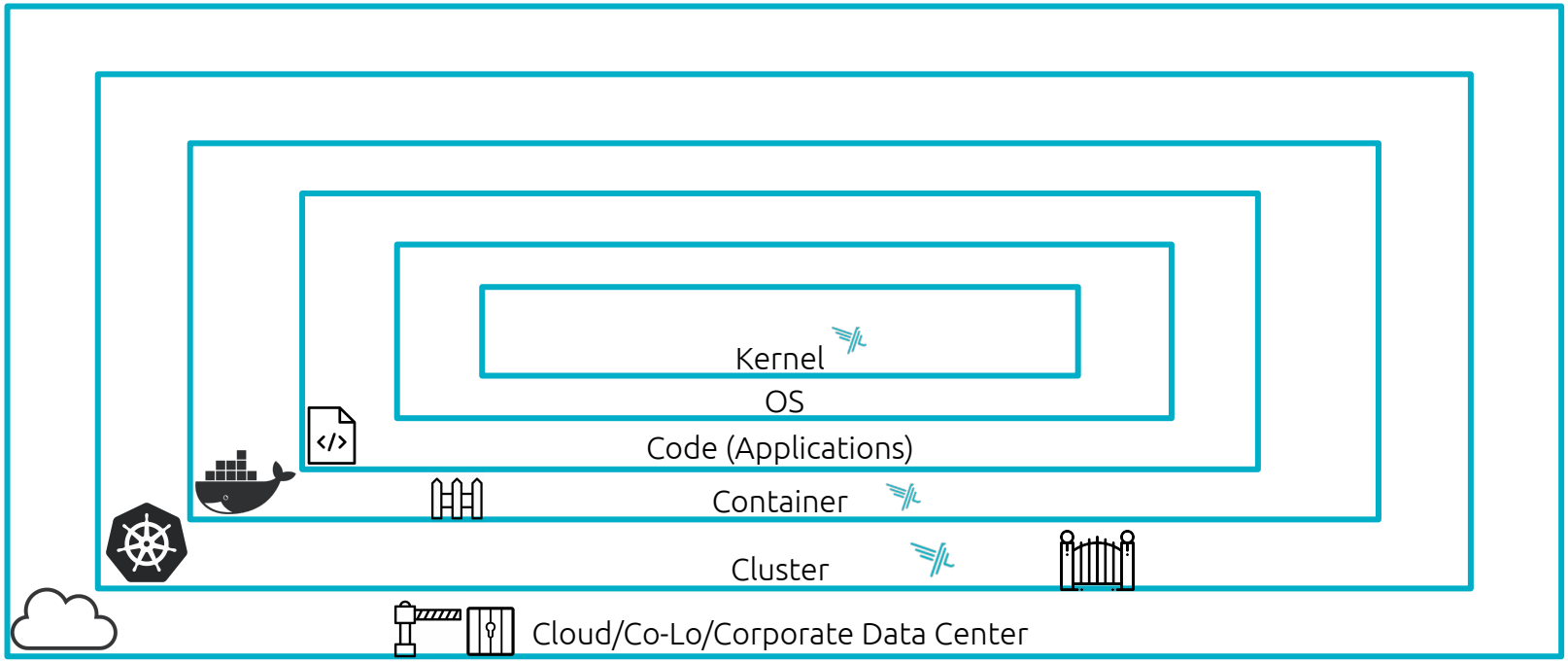
behavioral monitoring, intrusion & anomaly detection, forensics

- auditd
- Falco
- ...
- *a lot still to be done in this space!*

PREVENTION IS NOT ENOUGH.
COMPLEMENTARY, NOT MUTUALLY EXCLUSIVE APPROACHES

Prevention is not enough.

Combine with runtime detection tools. Use a defense-in-depth strategy.

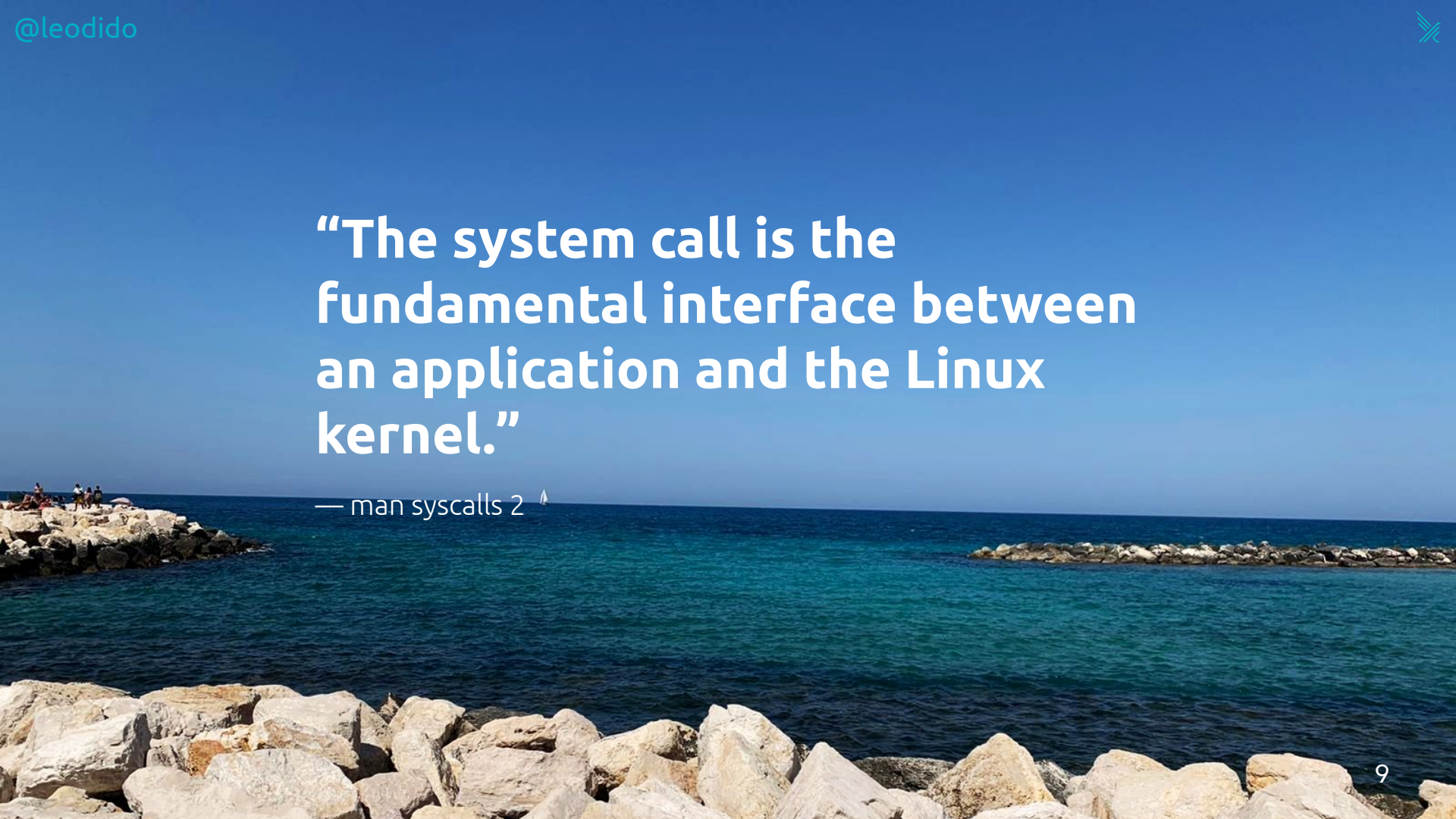


Runtime Security

She's Kelly. 💕

I have a lock on my front door and an alarm, but she alerts me when things aren't going right, when little bro is misbehaving or if there's someone suspicious outside or nearby.

She detects runtime anomalies in my life at home.



“The system call is the fundamental interface between an application and the Linux kernel.”

— man syscalls 2

Why syscalls?

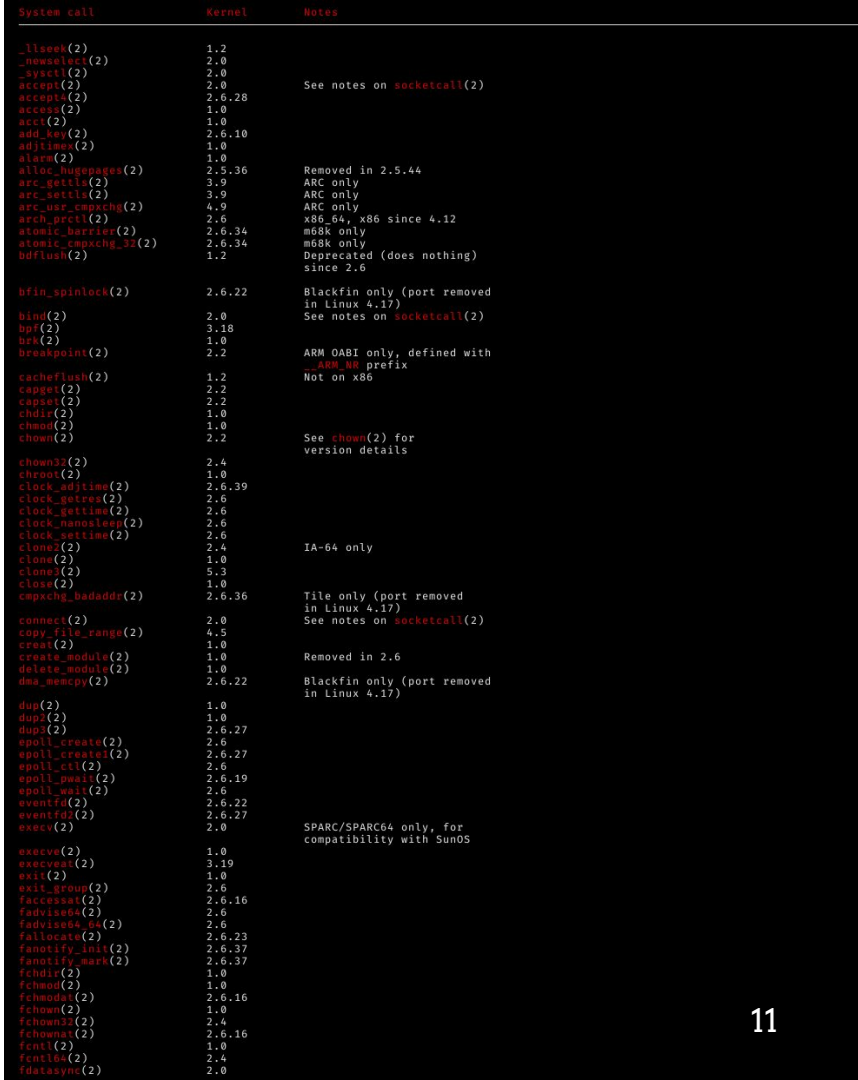
When you run a program you are making system calls.

System calls are how a program enters the kernel to perform some task.

- processes
- network
- file IO
- much more...

Unique challenges

- E_TOOMANY_SYSCALLS
- Millions per second
- Hard to manage in userspace
- Another syscall to know the time of an event



System call	Kernel	Notes
_llseek(2)	1.2	
_newselect(2)	2.0	
_sysctl(2)	2.0	
accept(2)	2.0	See notes on socketcall(2)
accept4(2)	2.6.28	
access(2)	1.0	
acct(2)	1.0	
add_key(2)	2.6.10	
adjtimex(2)	1.0	
alarm(2)	1.0	
alloc_hugepages(2)	2.5.36	Removed in 2.5.44
arc_gettls(2)	3.9	ARC only
arc_settls(2)	3.9	ARC only
arc_usr_cmpxchg(2)	4.9	ARC only
arch_prctl(2)	2.6	x86_64, x86 since 4.12
atomic_barrier(2)	2.6.34	m68k only
atomic_cmpxchg_32(2)	2.6.34	m68k only
bdflush(2)	1.2	Deprecated (does nothing) since 2.6
bfin_spinlock(2)	2.6.22	Blackfin only (port removed in Linux 4.17)
bind(2)	2.0	See notes on socketcall(2)
bpf(2)	3.18	
brk(2)	1.0	
breakpoint(2)	2.2	ARM OABI only, defined with __ARM_NR prefix
cacheflush(2)	1.2	Not on x86
capget(2)	2.2	
capset(2)	2.2	
chdir(2)	1.0	
chmod(2)	1.0	
chown(2)	2.2	See chown(2) for version details
chown32(2)	2.4	
chroot(2)	1.0	
clock_adjtime(2)	2.6.39	
clock_getres(2)	2.6	
clock_gettime(2)	2.6	
clock_nanosleep(2)	2.6	
clock_settime(2)	2.6	
clone2(2)	2.4	IA-64 only
clone(2)	1.0	
clone3(2)	5.3	
close(2)	1.0	
cmpxchg_badaddr(2)	2.6.36	Tile only (port removed in Linux 4.17)
connect(2)	2.0	See notes on socketcall(2)
copy_file_range(2)	4.5	
creat(2)	1.0	
create_module(2)	1.0	Removed in 2.6
delete_module(2)	1.0	
dma_memcpy(2)	2.6.22	Blackfin only (port removed in Linux 4.17)
dup(2)	1.0	
dup2(2)	1.0	
dup3(2)	2.6.27	
epoll_create(2)	2.6	
epoll_create1(2)	2.6.27	
epoll_ctl(2)	2.6	
epoll_pwait(2)	2.6.19	
epoll_wait(2)	2.6	
eventfd(2)	2.6.22	
eventfd2(2)	2.6.27	
execv(2)	2.0	SPARC/SPARC64 only, for compatibility with SunOS
execve(2)	1.0	
execveat(2)	3.19	
exit(2)	1.0	
exit_group(2)	2.6	
faccessat(2)	2.6.16	
fadvise64(2)	2.6	
fadvise64_64(2)	2.6	
fallocate(2)	2.6.23	
fanotify_init(2)	2.6.37	
fanotify_mark(2)	2.6.37	
fchdir(2)	1.0	
fchmod(2)	1.0	
fchmodat(2)	2.6.16	
fchown(2)	1.0	
fchown32(2)	2.4	
fchownat(2)	2.6.16	
fcntl(2)	1.0	
fcntl64(2)	2.4	
fdatasync(2)	2.0	

Still not enough...

CONTEXT

Timing

Arguments

CONTAINERS

Did the event originated in a container?

What's the container name and ID?

What's the container image?

ORCHESTRATOR

In which cluster it is running?

On which node?

What's the container runtime interface in use?

KERNEL MODULE

Pros: very efficient, implement almost anything
Cons: kernel panics, not always suitable

EBPF PROBE

Pros: program the kernel without risking to break it
Cons: newer kernels

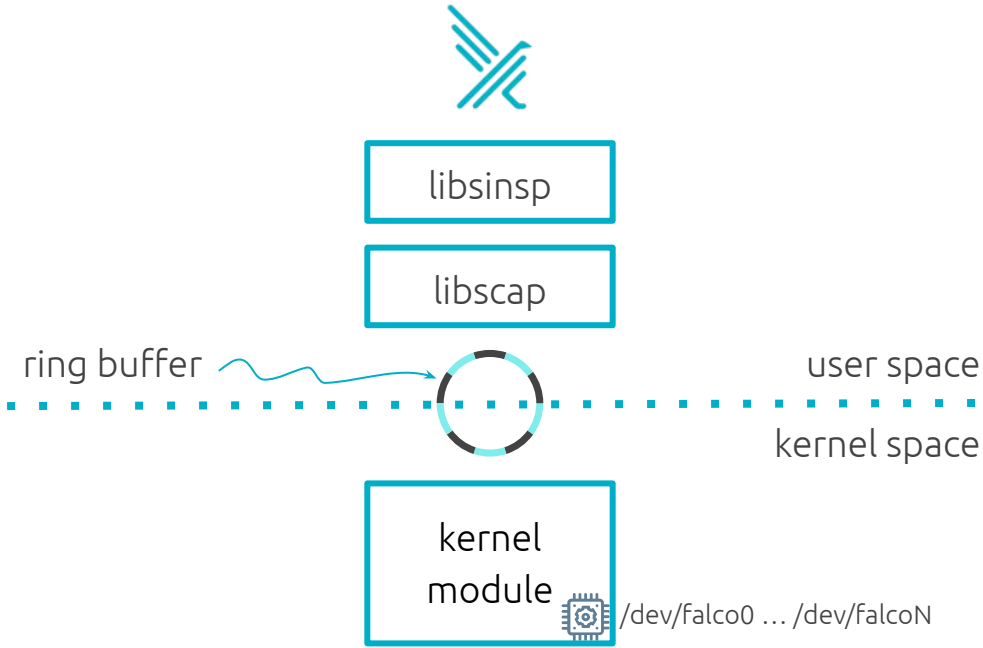
PDIG

Pros: (almost) unprivileged
Cons: really hackish, ~20% slower

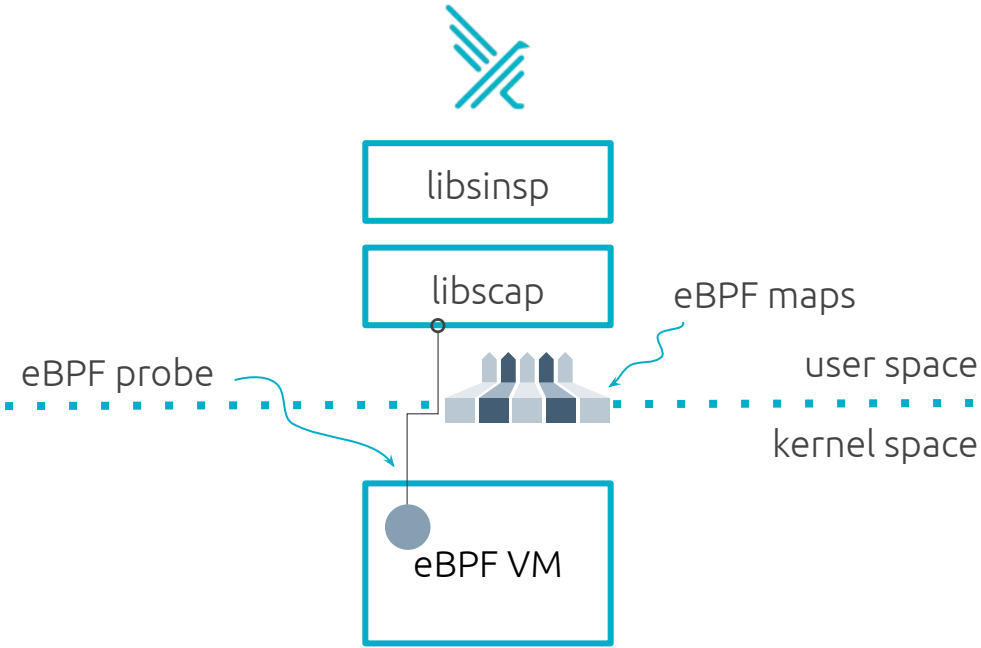
Other methods?

Future inputs/drivers?

Syscalls from Falco Kernel Module



Syscalls from Falco eBPF probe



Falco is a while(true).

```
#include <sinsp.h>

int main(int argc, char **argv) {
  int rc;
  sinsp_evt* evt; // event pointer
  while(true) {
    rc = inspector->next(&evt); // ask for next event
    // As the inspector has no filter at its level, all events are returned here.
    // Pass them to the falco engine, which will match the event against the set of rules.
    unique_ptr<falco_engine::rule_result> res = engine->process_sinsp_event(ev);
    // If a match is found, pass the event to the outputs.
    if(res) {
      outputs->handle_event(res->evt, res->rule, res->source, res->priority_num, res->format);
    }
  }
}
```

```yaml
- list: miner_ports
  items: [25, ..., 3334, 45700]

- list: miner_domains
  items: ["asia1.ethpool.org", "...", "xmr.pool.minergate.com"]

- list: https_miner_domains
  items: ["ca.minexmr.com", "...", "xmr.crypto-pool.fr"]

- list: http_miner_domains
  items: ["ca.minexmr.com", "...", "xmr.crypto-pool.fr"]

# Add rule based on crypto mining IOCs
- macro: minerpool_https
  condition: (fd.sport="443" and fd.sip.name in (https_miner_domains))

- macro: minerpool_http
  condition: (fd.sport="80" and fd.sip.name in (http_miner_domains))

- macro: minerpool_other
  condition: (fd.sport in (miner_ports) and fd.sip.name in (miner_domains))

- macro: net_miner_pool
  condition: (evt.type in (sendto, sendmsg) and evt.dir=< and
    (
      fd.net != "127.0.0.0/8" and not fd.snet in (rfc_1918_addresses))
      and ((minerpool_http) or (minerpool_https) or (minerpool_other)
    ))

- macro: trusted_images_query_miner_domain_dns
  condition: (container.image.repository in
    (docker.io/falcosecurity/falco, falcosecurity/falco))
  append: false

- rule: Detect outbound connections to common miner pool ports
  desc: Miners typically connect to miner pools on common ports.
  condition: net_miner_pool and not trusted_images_query_miner_domain_dns
  enabled: false
  output: Outbound connection to IP/Port flagged by cryptoioc
  (command=%proc.cmdline port=%fd.rport ip=%fd.rip container=%container.info image=%container.image.repository)
  priority: CRITICAL
  tags: [network, mitre_execution]
```

@leodido

Falco rules are YAML!

- ❑ lists
- ❑ conditions
- ❑ macros
- ❑ priorities/severities
- ❑ output messages
- ❑ tags
- ❑ overrides
- ❑ exceptions (soon)

Default rulesets *here* 📋

Examples

- ❑ spawned_process macro
- ❑ cloud metadata from container

Container drift?

```
- macro: container
  condition: (container.id != host)

- macro: runc_writing_exec_fifo
  condition: (proc.cmdline="runc:[1:CHILD] init" and fd.name=/exec.fifo)

- macro: runc_writing_var_lib_docker
  condition: (proc.cmdline="runc:[1:CHILD] init" and evt.arg.filename startswith /var/lib/docker)

- rule: Container Drift Detected (open+create)
  desc: New executable created in a container due to open+create
  condition: >
    evt.type in (open,openat,creat) and
    evt.is_open_exec=true and
    container and
    not runc_writing_exec_fifo and
    not runc_writing_var_lib_docker and
    evt.rawres>=0
  output: Drift detected (open+create), new executable created in a container (user=%user.name command=%proc.cmdline
filename=%evt.arg.filename name=%evt.arg.name mode=%evt.arg.mode event=%evt.type)
  priority: ERROR
```

See it in action!

Detect Kubernetes CVE-2020-8555

An attacker with permissions to create a pod with certain built-in volume types (GlusterFS, Quobyte, StorageFS, ScaleIO) or permissions to create a StorageClass can cause kube-controller-manager to make GET or POST requests from the master's host network.

kube-controller-manager < 1.15.11 / 1.16.0 - 1.16.8 / 1.17.0 - 1.17.4 / 1.18.0

How to detect? Write two Falco rules using Kubernetes audit logs as input to:

1. detect if the StorageClass object is created with one of the volume types
2. detect if pods are created using one of the volume types

Learn how to detect it step-by-step with Falco.

Detect Kubernetes CVE-2020-8555

```
- macro: affected_volumes_in_pod
  condition: jevt.value[/requestObject/spec/volumes] contains "scaleIO" or jevt.value[/requestObject/spec/volumes]
contains "glusterFS" or jevt.value[/requestObject/spec/volumes] contains "quobyte" or
jevt.value[/requestObject/spec/volumes] contains "storageFS"

# Detect CVE-2020-8555
- rule: Create Pod with Vulnerable Volume Type
  desc: Detect an attempt to start a pod with a vulnerable volume type.
  condition: kevt and pod and kcreate and response_successful and affected_volumes_in_pod
  output: "Pod started with vulnerable volume mount (user=%ka.user.name pod=%ka.resp.name ns=%ka.target.namespace
images=%ka.req.pod.containers.image volumes=%ka.req.pod.volumes.volume_type)"
  priority: "WARNING"
  tags: [k8s]
  source: "k8s_audit"
```

Detect Kubernetes CVE-2020-8555

```
- list: affected_storage_provisioners
  items:
  - "kubernetes.io/scaleio"
  - "kubernetes.io/quobyte"
  - "kubernetes.io/glusterfs"
  - "kubernetes.io/storagefs"

- macro: affected_storageclasses
  condition: (jevt.value[/requestObject/provisioner] in (affected_storage_provisioners))
  append: false

- macro: storageclass
  condition: ka.target.resource=storageclasses

- rule: Vulnerable StorageClass Object Created
  desc: Detect a vulnerable StorageClass object is created
  condition: kevt and storageclass and kcreate and affected_storageclasses and response_successful
  output: "Vulnerable storage class is created successfully (user=%ka.user.name
provisioner=%jevt.value[/requestObject/provisioner] objectName=%ka.target.name)"
  priority: "WARNING"
  tags: [k8s]
  source: "k8s_audit"
```

Other recent Kubernetes CVEs

Kubelet DoS

Writing lots of data to /etc/hosts

- CVE-2020-8557 (medium, Jul.)
- Detect it with Falco, mitigate with AppArmor [link]

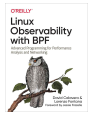
Root access from unprivileged local process

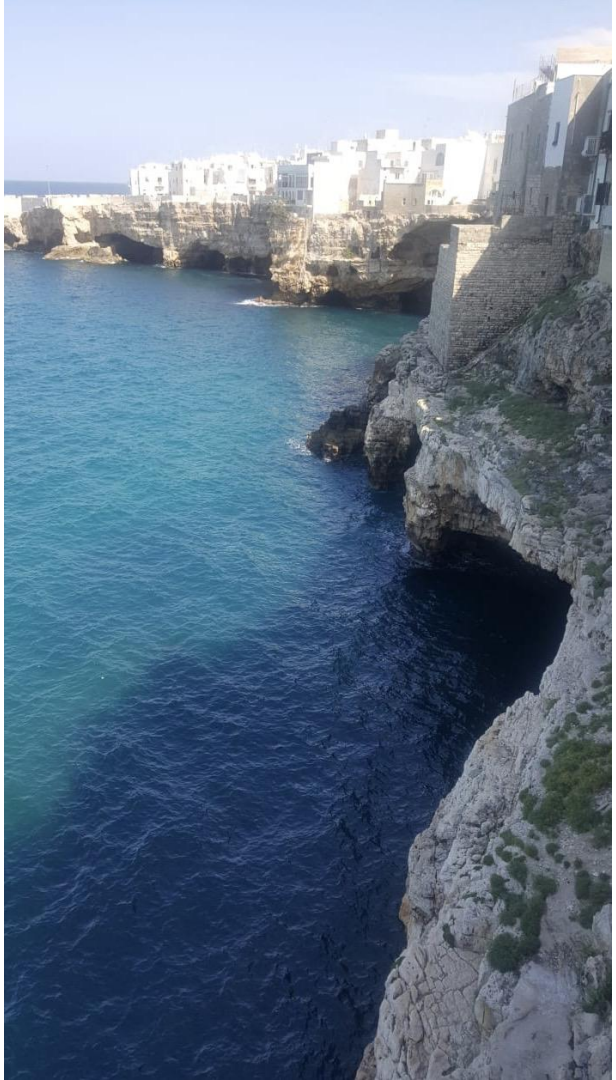
Triggering a memory corruption in the packet socket facility in the Linux kernel to hijack data and resources

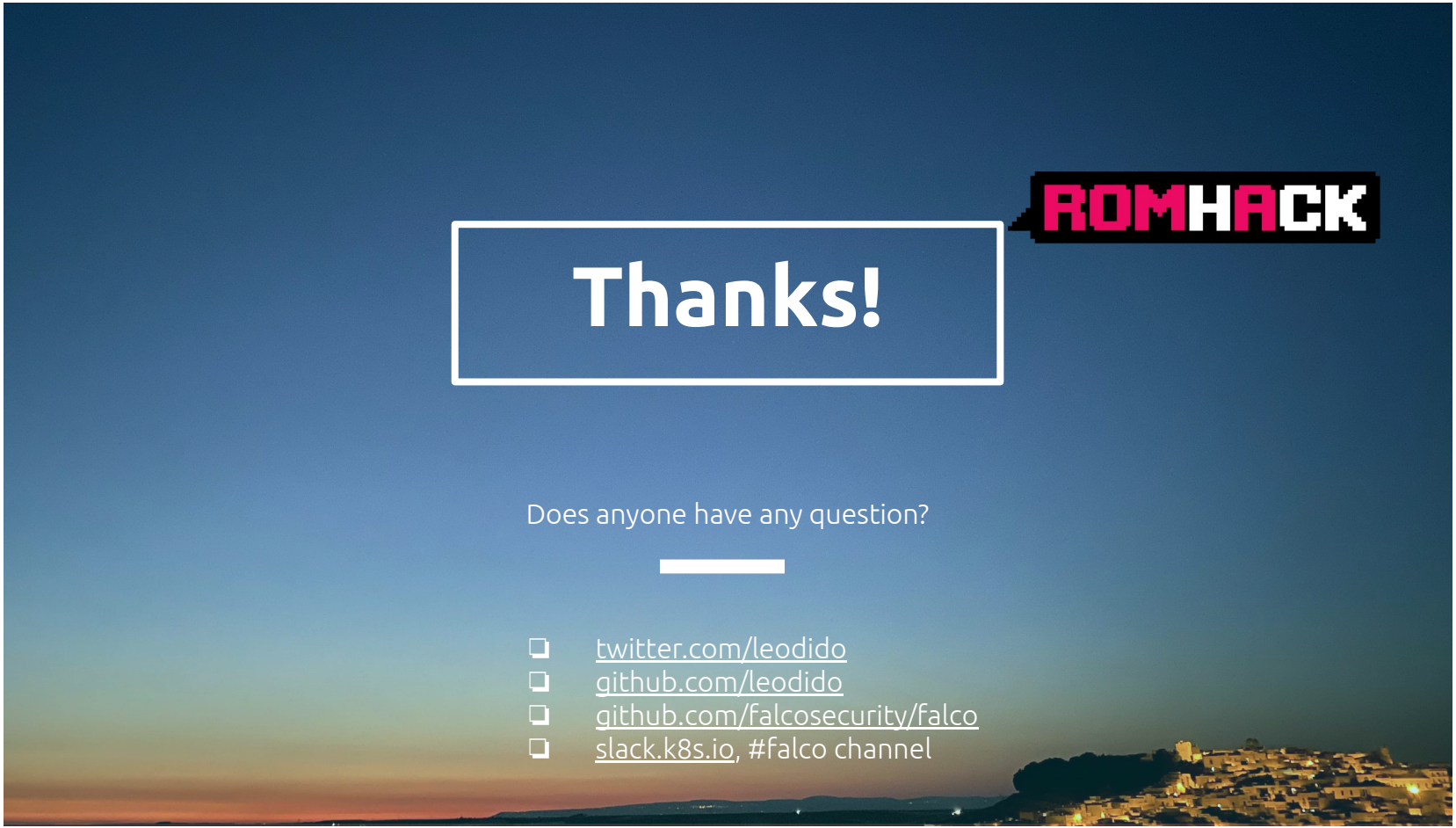
- CVE-2020-14386 (high, Sept.)
- Detecting with Falco [link]

Resources

- eBPF and Falco - *Leonardo Di Donato (Kubernetes Podcast)*
- Linux Observability With BPF: Advanced Programming for Performance Analysis and Networking - *Fontana, Calavera (O'Reilly)*
- The ring buffer definition
- Kernel module fillers:
  - f_sys_execve_e
  - f_sys_open_x
- eBPF probe fillers:
  - f_sys_execve_e
  - f_sys_open_x
- Falco default rule set






ROMHACK
Thanks!
Does anyone have any question?
twitter.com/leodido
github.com/leodido
github.com/falcosecurity/falco
slack.k8s.io, #falco channel