osquery @scale

www.osqueryatscale.com

osquery
@scale

osquery, eBPF, and Container Security

Ryan Mack, VP Eng, Head of Infra, Uptycs
Christopher Stanley, Mgr, Security Engineering, Aerospace Industry

01.

Introduction

Ryan Mack

VP Eng, Head of Infra

Uptycs

osquery
@scale

Christopher Stanley

Manager, Security Engineering

Aerospace Industry

YAeBPFT

Yet Another eBPF Talk

- Continuation of Zach’s & my talks last year
- Implementation tricks
- Real-world applications & modern eBPF

Background

eBPWhat?

osquery
@scale

osquery & Event Publishers

osquery

Scheduled Queries

Distributed Queries

Event Subscriptions

Event Sources

Streams

System APIs

Kernel Modules

eBPF

eBPF

Packet Filtering

Generalized for Monitoring

Evolving Feature Set

02.

Implementation Tricks

Kernel Compatibility

Compile once, run (almost) anywhere

Finding Data in Kernel Memory

- bpf_...() APIs are extremely limited
- Important context is buried in kernel data structures
    - Namespaces, working directory, ENV, FDs
- Kernel data structure layout is not fixed
    - Even ABI-versioned kernels break unexpectedly

Dynamic Programming

(no, not that kind)

Compile Time

Code Gen

Preprocessor

Templates

Link Time

Linker

Loader

CO-RE / **BTF**

Run Time

Big Switches

Function Pointers

Lookup Tables

Offset Management

- Maintain a header file of offsets required
- Populate offset lookup table from BTF if available
- Load offsets from local JSON file otherwise
- Validate kernel compatibility at osquery startup
- If check fails, check cloud for updates

Define and Use Offsets

offsets.h

```
offset(task_struct, fs);
offset(fs_struct, pwd);
offset(path, dentry);
offset(dentry, d_parent);
offset(dentry, d_name);
offset(qstr, name);
```

bpf.c HELPER(get_pwd)

```
task = bpf_get_current_task();

fss = bpf_probe_read(task +
          off->task_struct_fs)

dentry = bpf_probe_read(fss +
          off->fs_struct_pwd +
          off->path_dentry);

bpf_probe_read_str(buffer, dentry +
          off->dentry_name +
          off->qstr_name)
```

Maintaining Compatibility

- Maintain collection of legacy kernel versions
  - Back to 3.10.0 for RHEL 7, 4.14 for Amazon Linux 1
- Daily automated check all recent distros for new kernels
- Compile offset printing program against each kernel
- Dedup JSON in python, update the cloud DB
- Repeat for ARM64

Challenges and Opportunities

- Some kernel objects change structure or names
- Every validator has its own quirks
- Some offsets never change
- Improved macros would allow stronger type safety
- Powerful new features are only in later kernel versions

In-Kernel Filtering

Don’t waste time on uninteresting events

In-Kernel Filtering

- Some APIs provide filtering
- In eBPF we see *everything*
- Want to avoid sending to osquery if possible

osquery
@scale

What Do We Want to Filter On?

FIM

Include Paths

~~Exclude Paths~~

Read-Only Ops

~~Event Exclude Regexes~~

Socket Events

Source/Dest IP Event Exclude Regexes

DNS Events

Question Event Exclude ~~Regexes~~

Process Events

~~Event Exclude Regexes~~

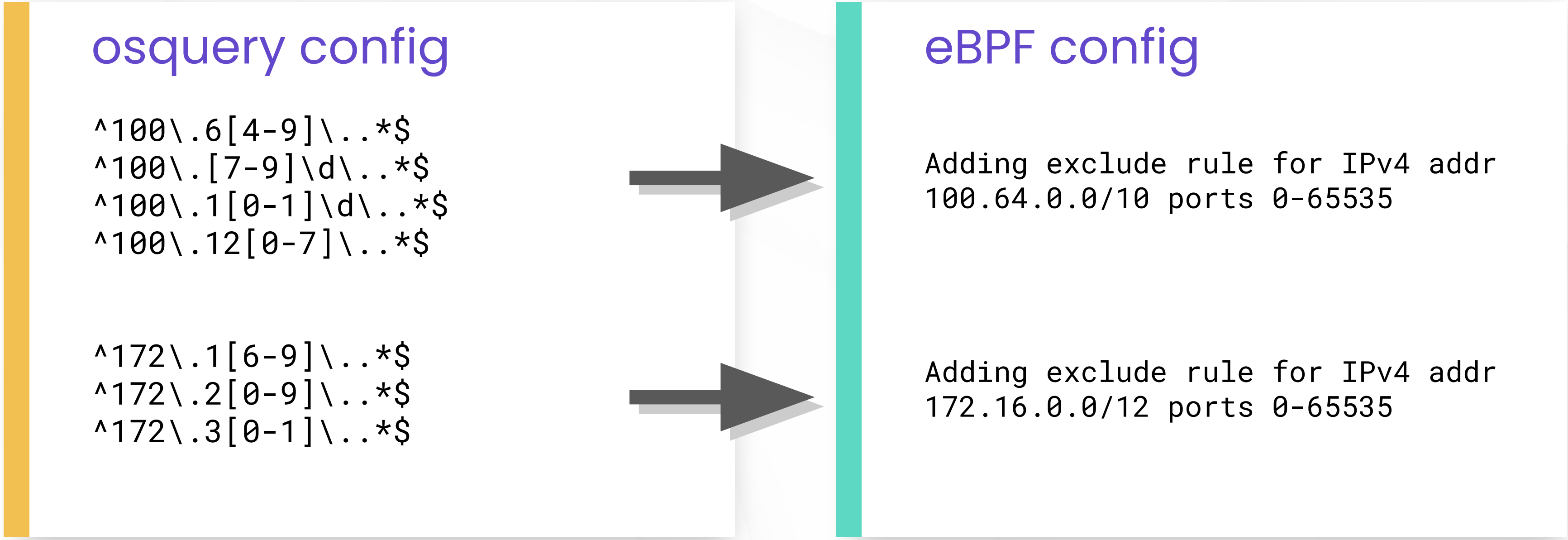
FIM Include Path Filtering

- Absolute paths are “easy”
- Relative paths require CWD (relative to current root!)
- Walk up tree, stop, reverse that, append param
- Must send “dangerous” or long paths to userspace
- Check safe paths against fixed length prefix hash table
- Maintain separate hashes for read and write ops

Socket Events

- osquery exclude expressions are regexes
- Convert regexes to CIDR addresses at config time!
  - Recursively expand regex to all matching IP prefixes
  - Sort + merge prefixes to reduce number of entries
  - Check prefix list in eBPF socket-related functions
  - Optionally support port range on each entry

Socket Exclusion Examples



osquery config

```
^100\.6[4-9]\..*$
^100\.[7-9]\d\..*$
^100\.1[0-1]\d\..*$
^100\.12[0-7]\..*$
```

```
^172\.1[6-9]\..*$
^172\.2[0-9]\..*$
^172\.3[0-1]\..*$
```

eBPF config

```
Adding exclude rule for IPv4 addr
100.64.0.0/10 ports 0-65535
```

```
Adding exclude rule for IPv4 addr
172.16.0.0/12 ports 0-65535
```

DNS and Process Events

- DNS events filter on question but presume . is literal
- Process events not filtered due to ancestry confusion

Challenges and Opportunities

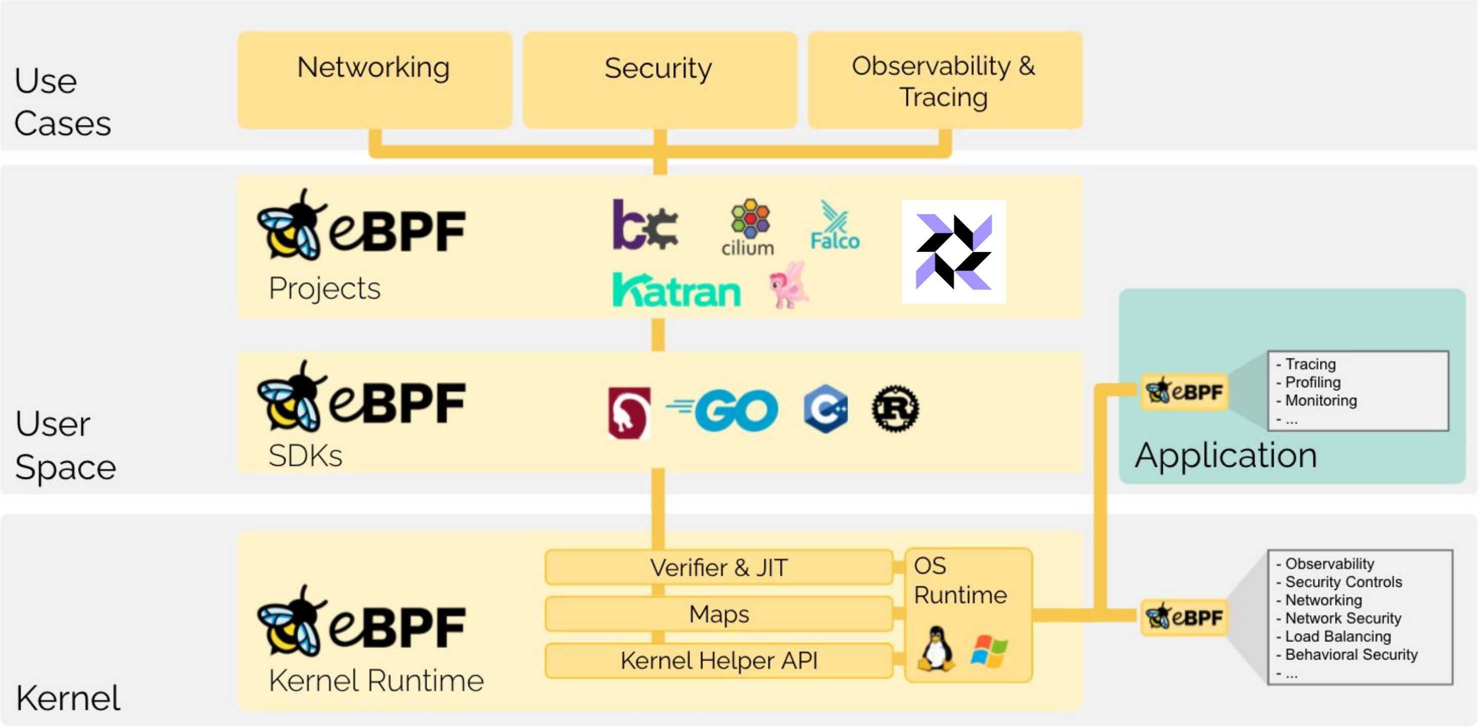
- Symlinks are still troublesome
- Mixing namespaces & chroots — oh my!
- Should try proper bloom filter for fuller path matching
- Multi-column exclusion rules should exist

osquery
@scale

03.

Real World Applications & Modern eBPF

What the eBPF?

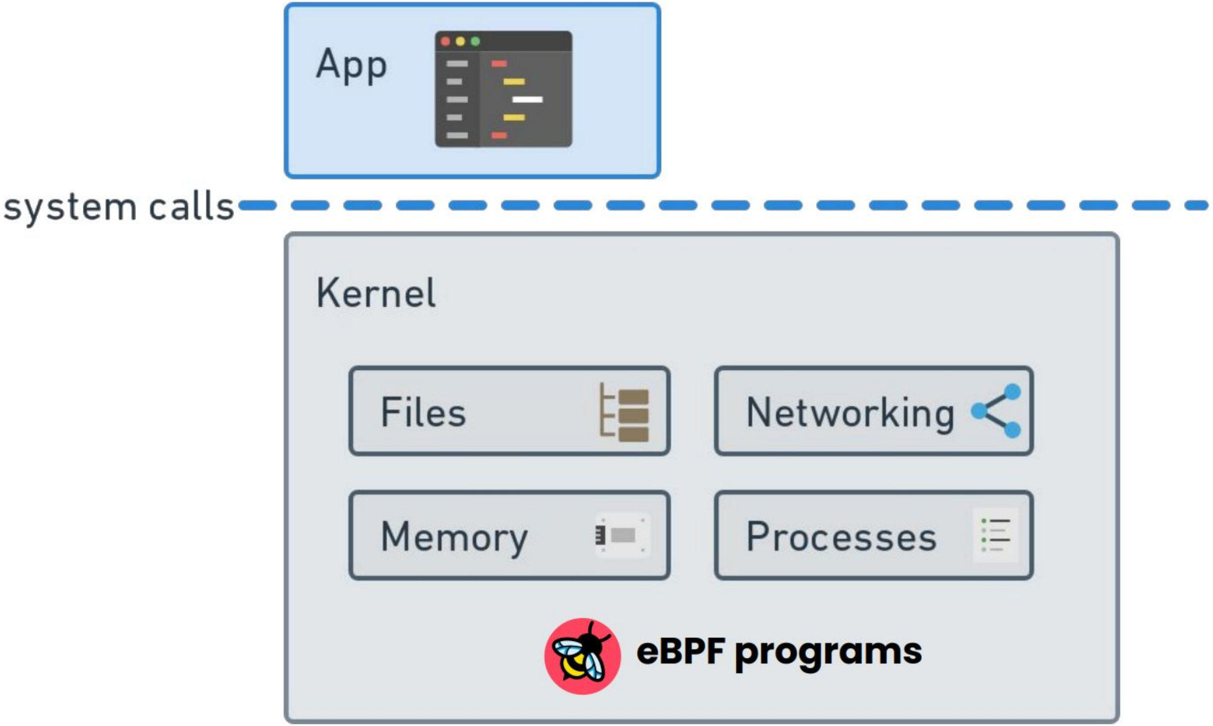


osquery
@scale

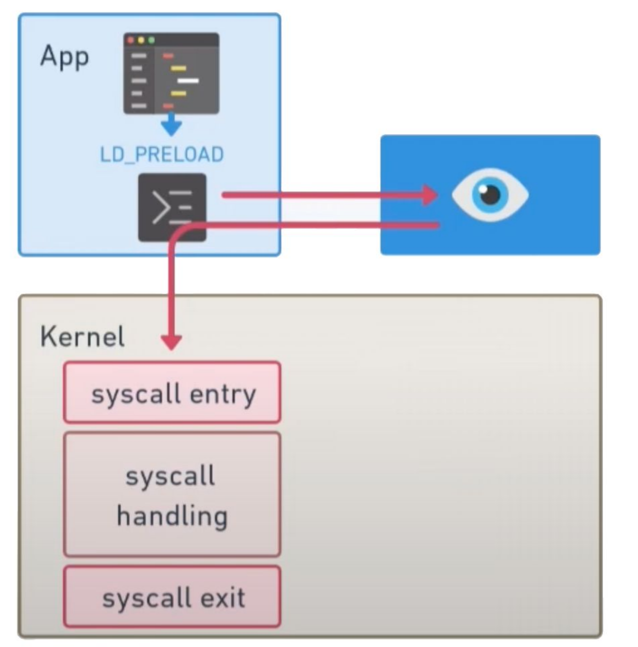
Hardening

- **Program execution protection**
- **Mitigation against Spectre**
  - Daniel Borkmann - Mitigating transient execution attacks
- **Constant Blinding**
- **Abstracted Runtime Context**

What do we care about?

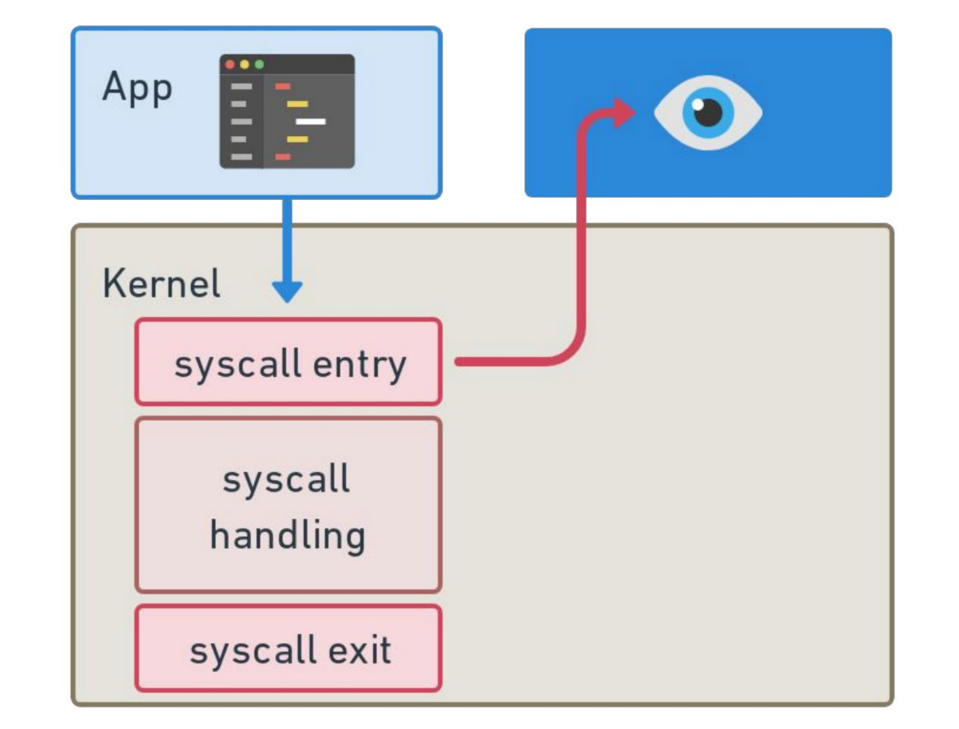


LD_PRELOAD?

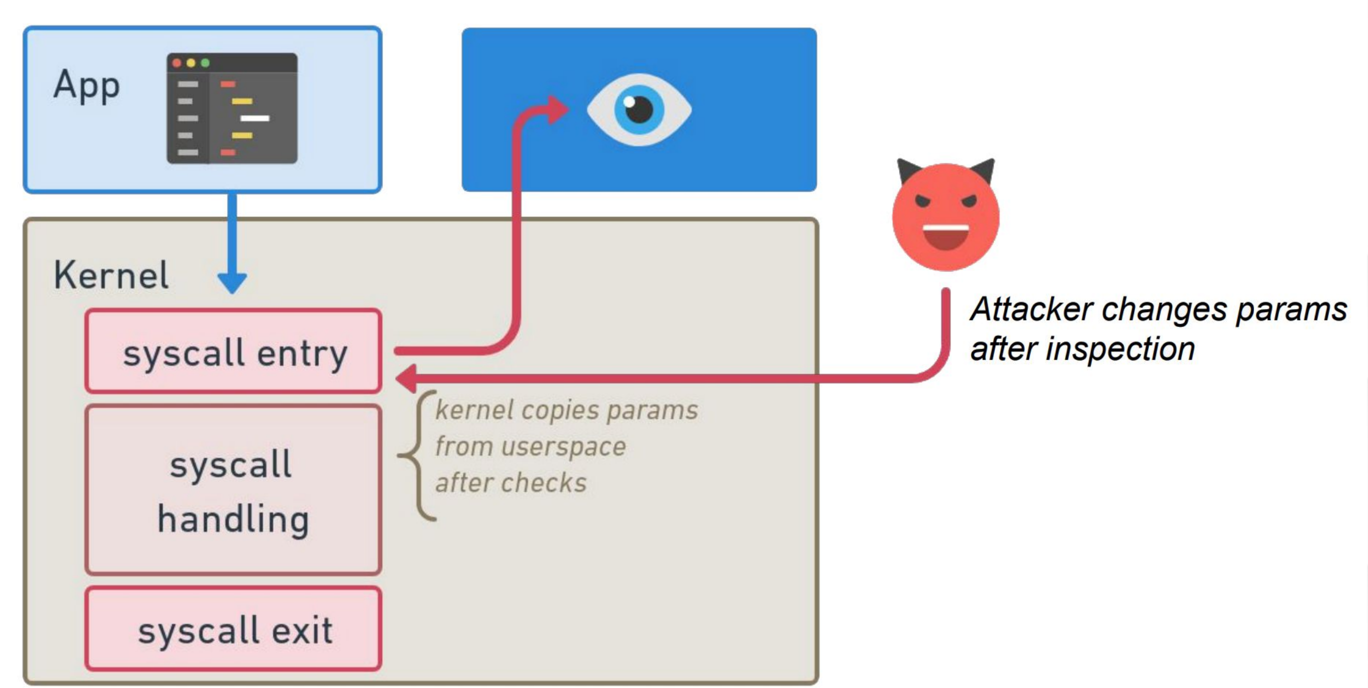


App
LD_PRELOAD
Kernel
syscall entry
syscall handling
syscall exit

Inspecting Syscalls to the Kernel

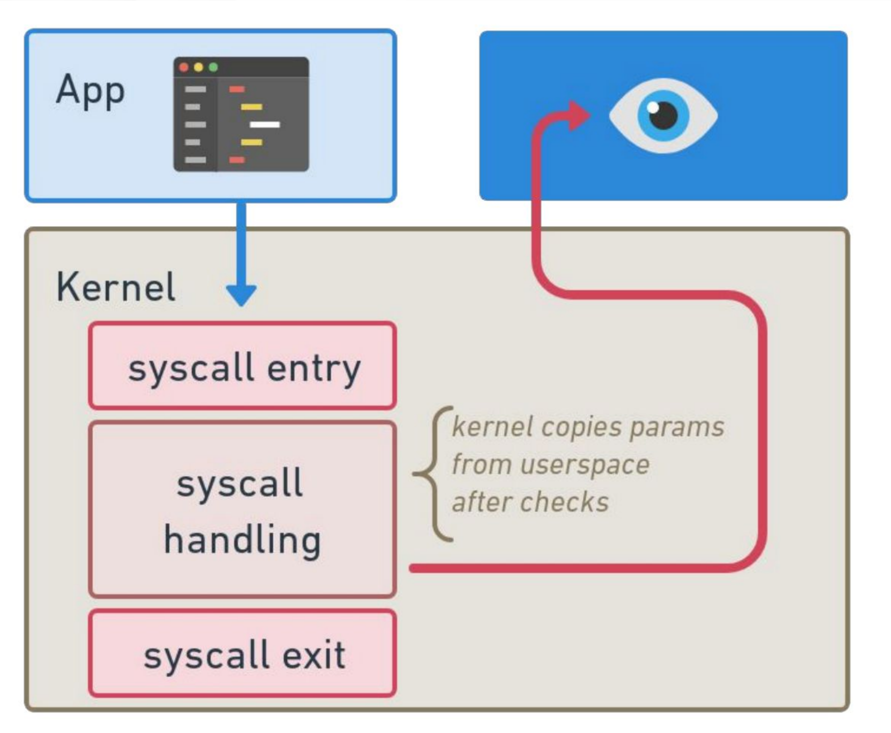


Time of check, time of use exploit

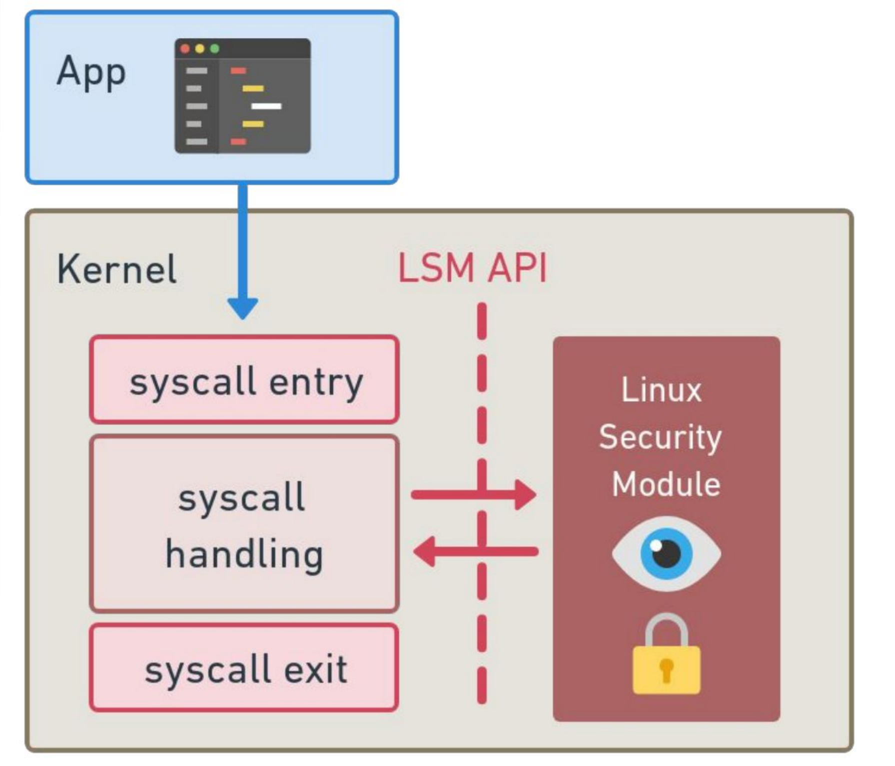


Attacker changes params after inspection

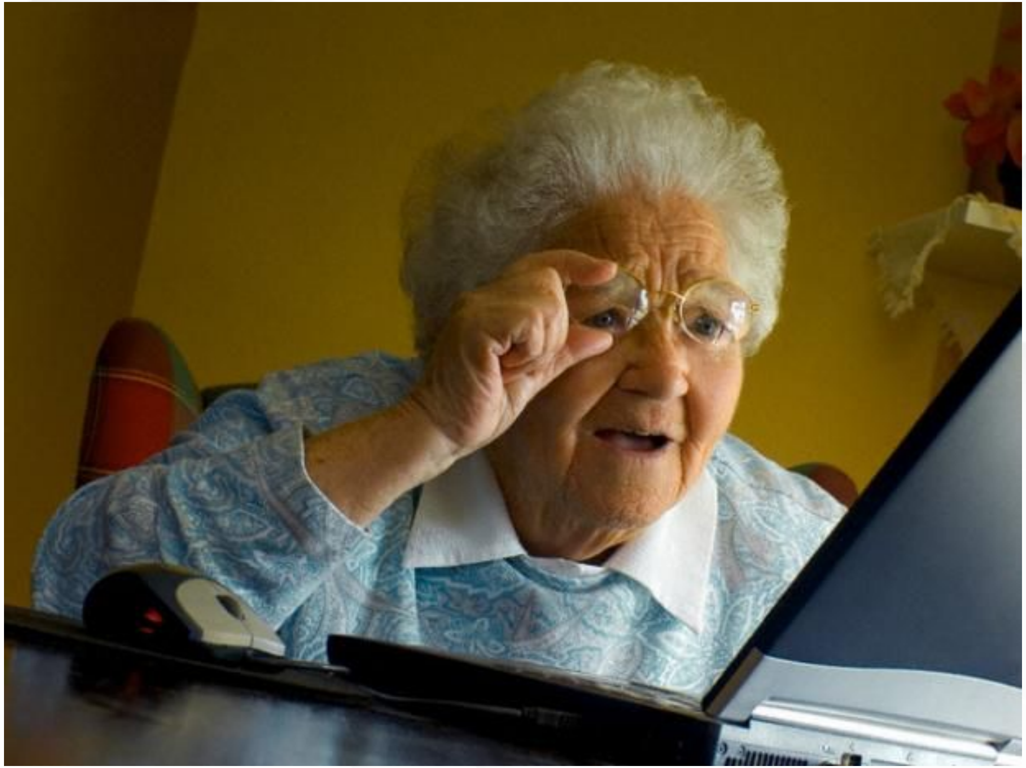
Imma let you copy but...



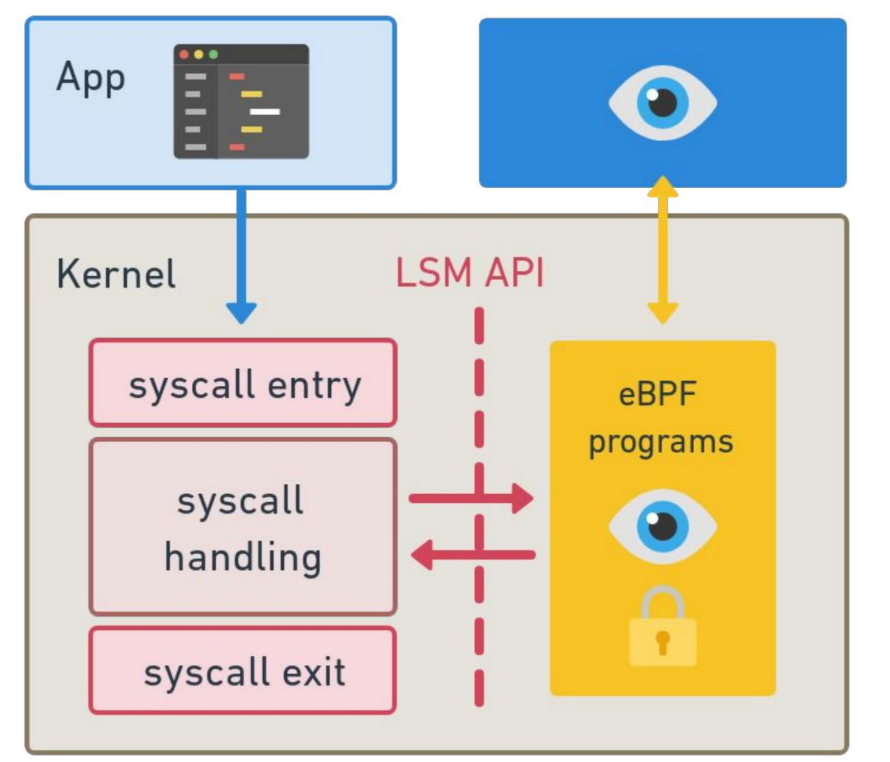
How do?


App
Kernel
LSM API
syscall entry
syscall handling
syscall exit
Linux Security Module

Out with the old, in with the new Kernel



eBPF + Linux Security Modules FTW!



What newer Kernels get you

- BPF LSM
- New BPF Data Structure
  - perfbuf
    - per cpu circular buffers
    - inefficient use of memory and event re-ordering
    - wasted work and copying
  - ringbuff
    - multi-producer, single-consumer
    - shared across multiple CPUs simultaneously

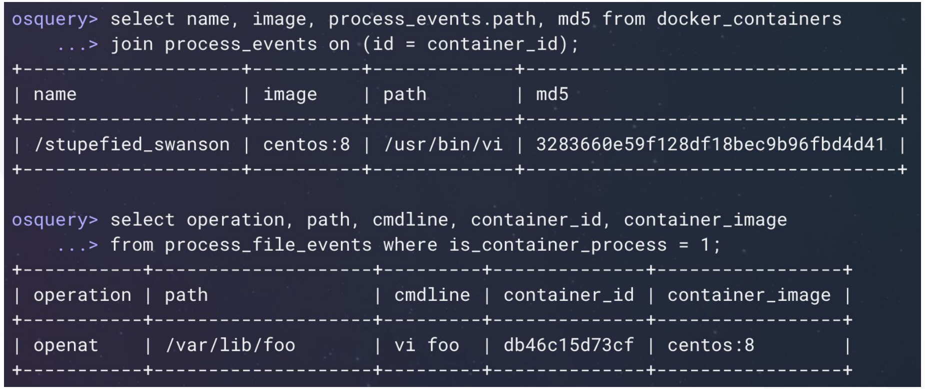
Securing Containers

- Adopt the concept of “Least Privileged”
- Only run the binaries you need
- Do not use the --privileged flag or mount a Docker socket inside the container.
- Do not run as root inside the container.
- Drop all capabilities (--cap-drop=all) and enable only those that are required (--cap-add=...).
- Adjust seccomp, AppArmor, or SELinux profiles to restrict the actions and syscalls available for the container to the minimum required.

Putting it all together

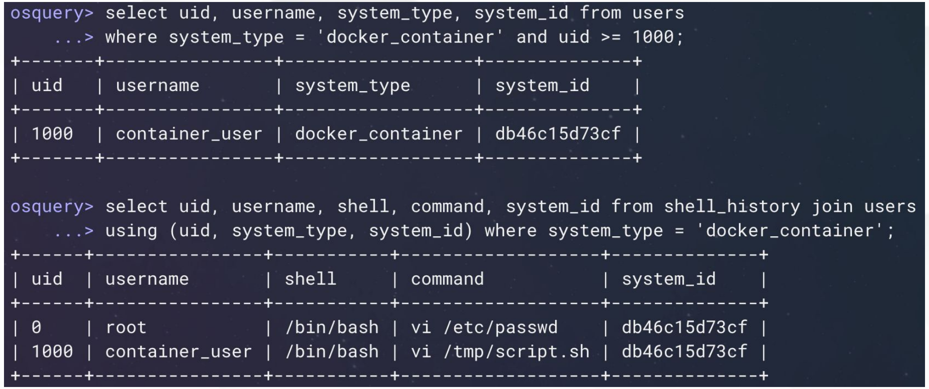
Containers running on host

- log forward events to your SIEM
- context around host
- syscalls, etc correlated back to container
- write detections based on abnormal behaviour



```
osquery> select name, image, process_events.path, md5 from docker_containers
    ...> join process_events on (id = container_id);
+--------------------+----------+-------------+----------------------------------+
| name               | image    | path        | md5                              |
+--------------------+----------+-------------+----------------------------------+
| /stupefied_swanson | centos:8 | /usr/bin/vi | 3283660e59f128df18bec9b96fbd4d41 |
+--------------------+----------+-------------+----------------------------------+

osquery> select operation, path, cmdline, container_id, container_image
    ...> from process_file_events where is_container_process = 1;
+-----------+--------------------+---------+--------------+-----------------+
| operation | path               | cmdline | container_id | container_image |
+-----------+--------------------+---------+--------------+-----------------+
| openat    | /var/lib/foo       | vi foo  | db46c15d73cf | centos:8        |
+-----------+--------------------+---------+--------------+-----------------+
```

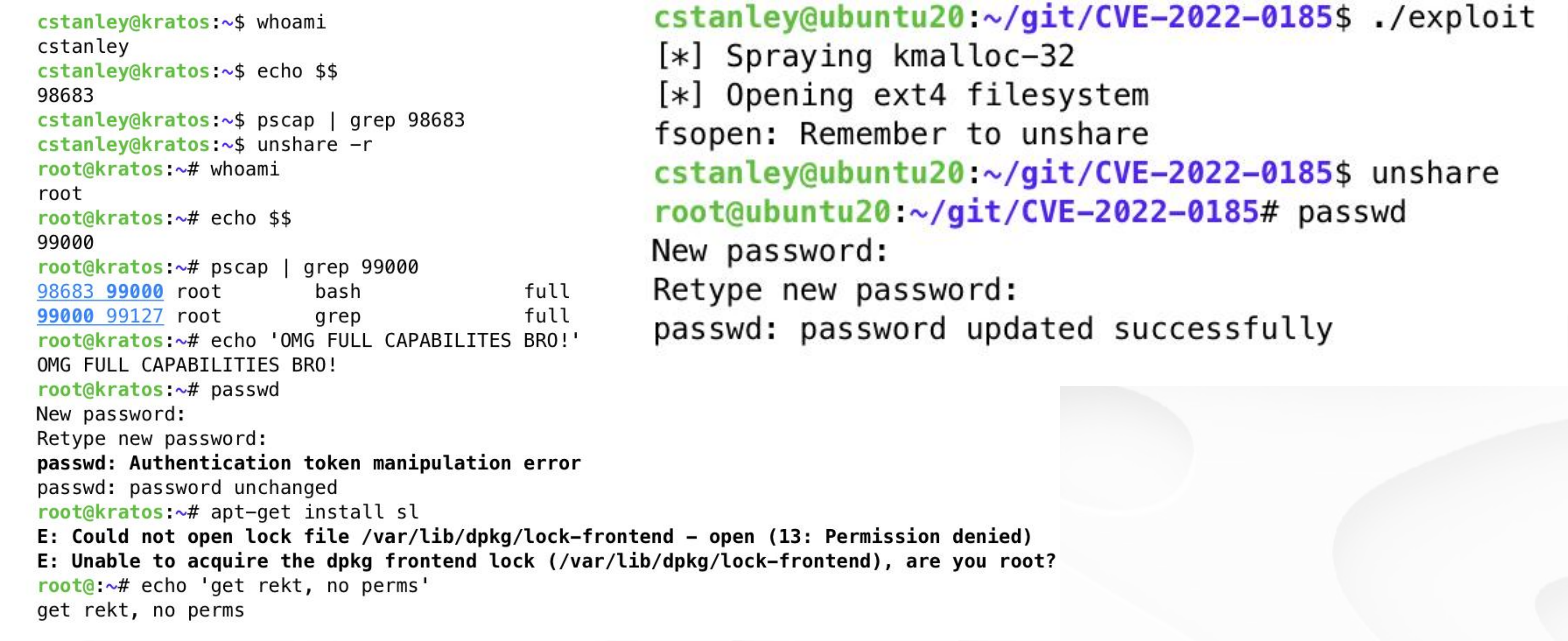


```
osquery> select uid, username, system_type, system_id from users
    ...> where system_type = 'docker_container' and uid >= 1000;
+-------+----------------+------------------+--------------+
| uid   | username       | system_type      | system_id    |
+-------+----------------+------------------+--------------+
| 1000  | container_user | docker_container | db46c15d73cf |
+-------+----------------+------------------+--------------+

osquery> select uid, username, shell, command, system_id from shell_history join users
    ...> using (uid, system_type, system_id) where system_type = 'docker_container';
+------+----------------+-----------+-------------------+--------------+
| uid  | username       | shell     | command           | system_id    |
+------+----------------+-----------+-------------------+--------------+
| 0    | root           | /bin/bash | vi /etc/passwd    | db46c15d73cf |
| 1000 | container_user | /bin/bash | vi /tmp/script.sh | db46c15d73cf |
+------+----------------+-----------+-------------------+--------------+
```

Detecting CVE-2022-0185

- Requires CAP_SYS_ADMIN
- Unprivileged namespaces (unshare -Urm)
- Containers != Security
- Privilege Escalation from root use to not matter



```
cstanley@kratos:~$ whoami
cstanley
cstanley@kratos:~$ echo $$
98683
cstanley@kratos:~$ pscap | grep 98683
cstanley@kratos:~$ unshare -r
root@kratos:~# whoami
root
root@kratos:~# echo $$
99000
root@kratos:~# pscap | grep 99000
98683 99000 root         bash                  full
99000 99127 root         grep                  full
root@kratos:~# echo 'OMG FULL CAPABILITES BRO!'
OMG FULL CAPABILITIES BRO!
root@kratos:~# passwd
New password:
Retype new password:
passwd: Authentication token manipulation error
passwd: password unchanged
root@kratos:~# apt-get install sl
E: Could not open lock file /var/lib/dpkg/lock-frontend - open (13: Permission denied)
E: Unable to acquire the dpkg frontend lock (/var/lib/dpkg/lock-frontend), are you root?
root@:~# echo 'get rekt, no perms'
get rekt, no perms
```

```
cstanley@ubuntu20:~/git/CVE-2022-0185$ ./exploit
[*] Spraying kmalloc-32
[*] Opening ext4 filesystem
fsopen: Remember to unshare
cstanley@ubuntu20:~/git/CVE-2022-0185$ unshare
root@ubuntu20:~/git/CVE-2022-0185# passwd
New password:
Retype new password:
passwd: password updated successfully
```

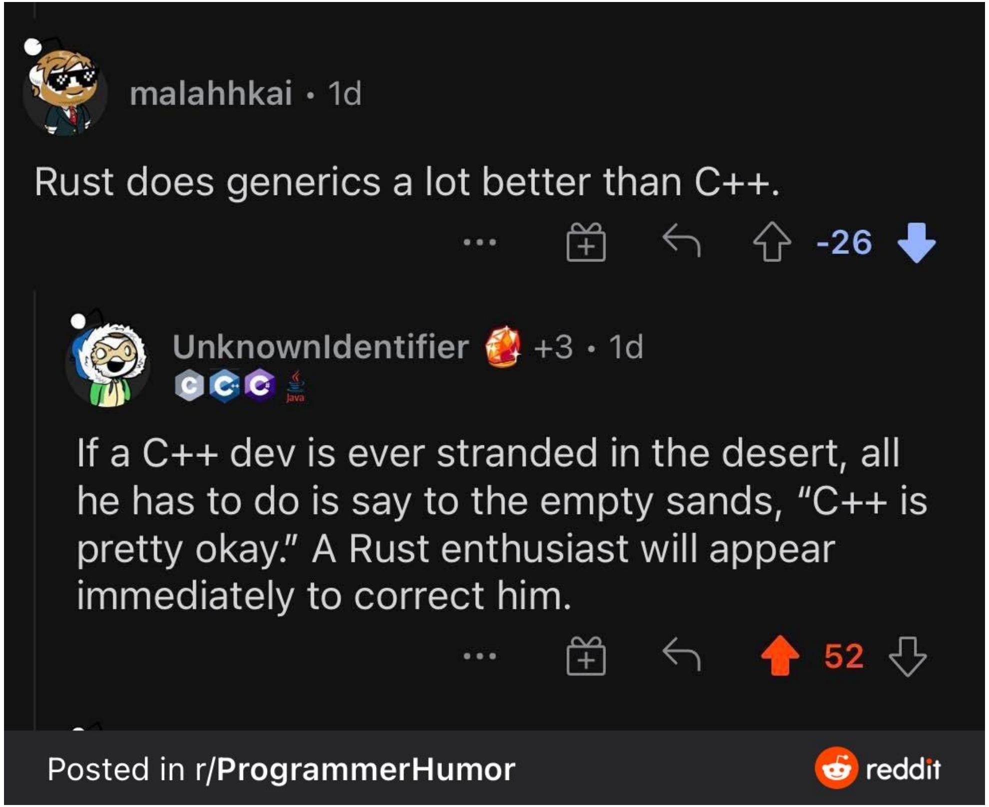
What does this mean?

- Disable unprivileged namespaces
  - user.max_user_namespaces = 0
- Start enforcing SECCOMP and block the usage of things like unshare
  - This is default for most systems anyways! Just not Kubernetes 😭
- Run containers least privilege (No CAP_SYS_ADMIN types of shenanigans!)
- Ensure to keep things updated! (In this case patch to kernel > 5.16.2
- DETECT THESE SHENANIGANS WITH OSQUERY!

Example Detection from osquery-based CWPP

```
{
  "event_tags": [ "ATTACK", "Container", "Endpoint", "Linux", "Privilege Escalation", "T1611", "process_events" ],
  "code": "ATTACK_PRIVILEGE_ESCALATION_T1611_LINUX_CONTAINER_BREAKOUT",
  "description": "Adversaries may try to exploit docker misconfiguration to break into host",
  "metadata": {
    "Start Time": 2411,
    "Binary Size": 68112,
    "Command Line": "chmod u+s /bin/bash",
    "Process Path": "/usr/bin/chmod",
    "User Interface": "0",
    "Is LD_PRELOAD": 0,
    "Parents": [
      { "exe_name": "kthreadd", "path": "kthreadd", "upt_rid": "0-4130", "pid": 4130 },
      { "exe_name": "kthreadd", "path": "kthreadd", "upt_rid": "0-2582", "pid": 2582 },
      { "exe_name": "kthreadd", "path": "kthreadd", "upt_rid": "166312477407-2", "pid": 2 }
    ],
    "User": "root",
    "Is Container Process": 0,
    "SHA256": "a3e141a69b71b7a6b55dee7ff73d0ee8755e90abab427cd6854341221a3b4748",
    "Process": "chmod",
    "Process ID": 4131
  }
}
```

Shameless Rust Plug



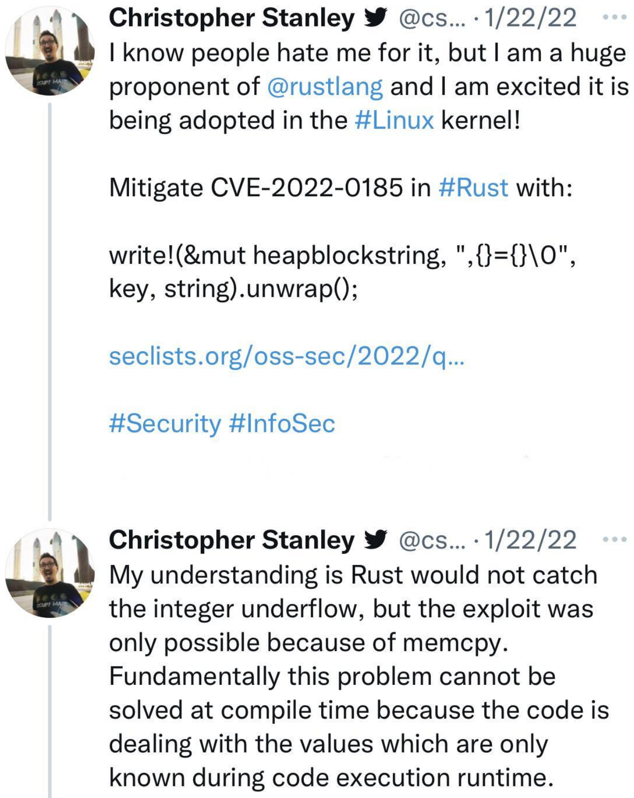
malahhkai • 1d

Rust does generics a lot better than C++.

UnknownIdentifier +3 • 1d

If a C++ dev is ever stranded in the desert, all he has to do is say to the empty sands, "C++ is pretty okay." A Rust enthusiast will appear immediately to correct him.

Posted in r/ProgrammerHumor



Christopher Stanley @cs... · 1/22/22

I know people hate me for it, but I am a huge proponent of @rustlang and I am excited it is being adopted in the #Linux kernel!

Mitigate CVE-2022-0185 in #Rust with:

```
write!(&mut heapblockstring, ",{}={}\0", key, string).unwrap();
```

seclists.org/oss-sec/2022/q...

#Security #InfoSec

Christopher Stanley @cs... · 1/22/22

My understanding is Rust would not catch the integer underflow, but the exploit was only possible because of memcpy. Fundamentally this problem cannot be solved at compile time because the code is dealing with the values which are only known during code execution runtime.

References

- Alessandro Gario
    - Monitoring Linux events: how to leverage BPF directly in your application without external tools
- Liz Rice
    - Real Time Security - eBPF for Preventing attacks
- Andrii Nakryiko

Thank You & Questions

osquery
@scale

www.osqueryatscale.com