Google

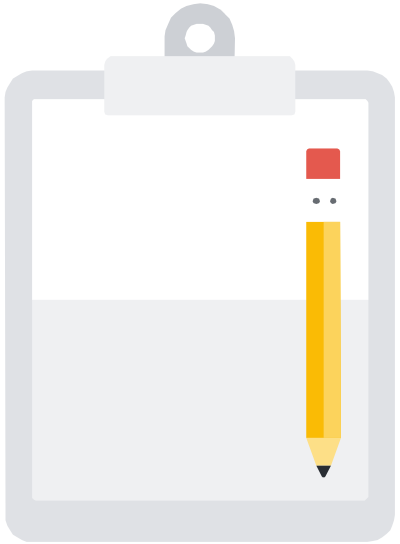
Kernel Runtime Security Instrumentation

KP Singh

Linux Security Summit

Agenda

- Motivation
- How does it work?
- What are the Alternatives?
- Case Study (Use Case)
- Performance Comparison
- Demo & Design Questions



Contributors (Thanks!)

Adam Sindelar

Günther Noack

Jann Horn

Kees Cook

Michael Halcrow

Thomas Garnier

Motivation

Security

Signals

Audit

Perf

Correlation with maliciousness but do not imply it

Mitigation

SELinux, Apparmor (LSMs)

seccomp

It's bad, stop it!

Adding a new Signal

Signals

Audit

Perf

Update Audit (user/kernel) to log environment variables

Mitigation

SELinux, Apparmor (LSMs)

seccomp

Security

Signals

Audit

Perf

Mitigation

SELinux, Apparmor (LSMs)

seccomp

Update the mitigation logic for a malicious actor with a known **LD_PRELOAD signature**

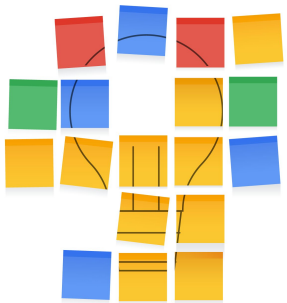
Signals

- A process that executes and deletes its own executable.
- A Kernel module that loads and "hides" itself
- "Suspicious" environment variables.

Mitigations

- Prevent mounting of USB drives on servers.
- Dynamic whitelist of known Kernel modules.
- Prevent known vulnerable binaries from running.

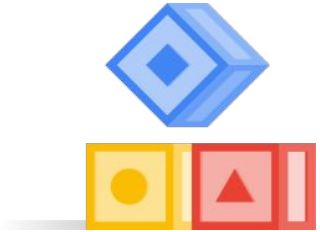
“Can we make it easy to add signals and mitigations in a unified way?”



How does it work?

eBPF + LSM

A **new** program type providing a **unified policy API** for signals and mitigation.



Security focussed eBPF helpers as the **building blocks** for the LSM logic

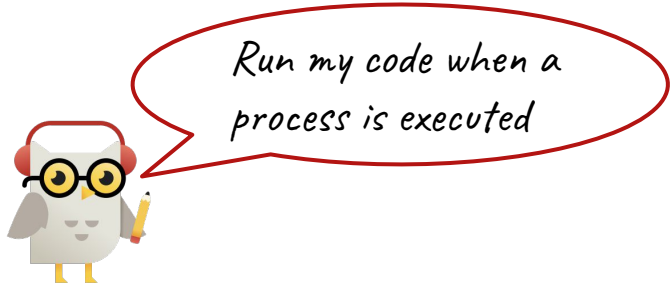
Why LSM?

- Mapping to **security behaviours** rather than the API.
- Easy to **miss** if instrumenting using **syscalls** (eg. `execve`, `execveat`)
- Benefit the **LSM ecosystem** by incorporating feedback from the security community.

> I want to log `LD_PRELOAD` **on process execution.**

- Security Engineer

How does it Work?



/sys/kernel/security/krsi/bprm_check_security

my_bpf_prog.o

open [O_RDWR]

bpf [BPF_PROG_LOAD]

securityfs_fd

prog_fd

bpf [BPF_PROG_ATTACH]

LSM:bprm_check_security
(when a process is executed)

KRSI's Hook

Other LSM Hooks

The KRSI Hook

```
struct krsi_hook {
    const char *name;
    enum krsi_hook_type h_type;
    struct dentry *h_dentry;
    struct mutex mutex;
    struct bpf_prog_array __rcu      *progs;
};
```

Name of the file in securityfs

Pointer to the `dentry` of the securityfs file

Array of attached eBPF programs

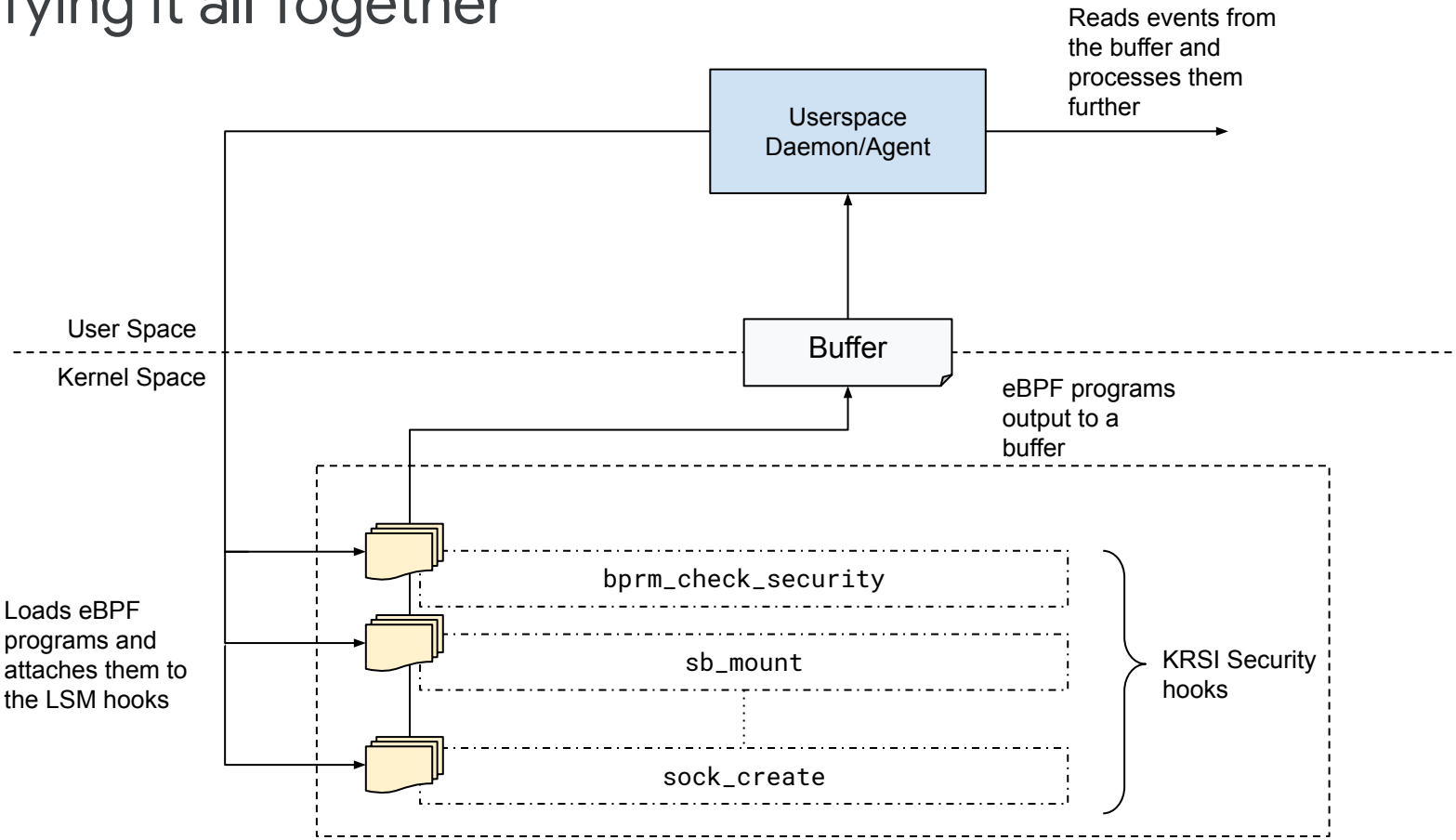
Google

Key Design Principles

Keep the helpers **precise** and **granular**

No access to kernel data structures in eBPF, maintain **backward compatibility**

Tying it all Together



Google

The Alternatives

Why not Audit?

Mitigation needs to be handled separately

Performance **overhead** when enabled (without rules)

Rigid formatting constraints

Why not seccomp + eBPF?

LSM maps better to security behaviours.

Multiple syscalls can represent a single behaviour.
(eg. `execve`, `execveat`)

TOCTTOU for user-memory based checks as verification happens before the syscall captures memory from user space.

Why not kprobes + eBPF?

`bpf_probe_read{_str}` give direct access to kernel data structures

Dependency on Kernel Data Structures makes deployment hard.

kprobes is not a stable API, with no guarantees on locks, IRQs preemption etc.

Why not Landlock?

Landlock is geared towards creation of security sandboxes for unprivileged processes.

KRSI gives granular access to security behaviours with an ecosystem of security focused helpers.

Case Study: Environment Variables

Google

Definition

What?

Audit environment variables on process execution

Why's that hard?

Environment variable can be 32 pages long!

eBPF Helper Design Choices

`krsi_get_env_vars()`

Returns all the environment variables.

Higher coverage at the expense of significant overhead

`krsi_get_env_var(const char*)`

Returns the value a single environment variable.

Carefully, choose the variables to be audited, less overhead.

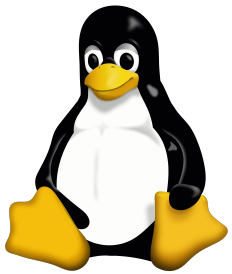
Can cause the code to sleep (as a result of a page fault)

Data Format

```
struct krsi_env_value {
        char name[ENV_VAR_NAME_MAX_LEN];
        char value[ENV_VAR_VAL_MAX_LEN];
        bool overflow;
        u32 pid;
}
```

KRSI
eBPF

Kernel



Security
Product

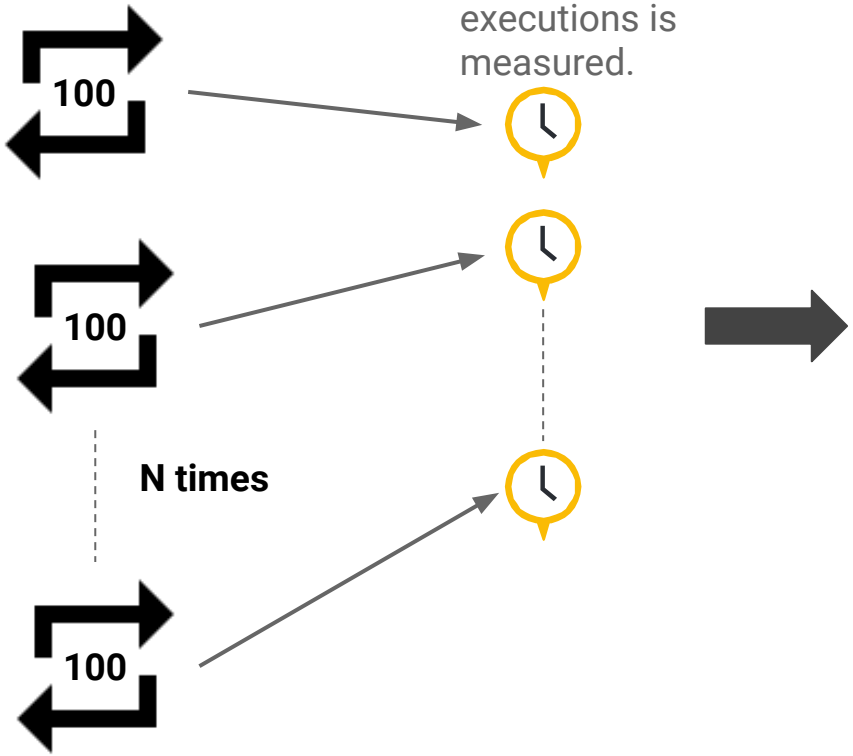
Performance Analysis

Google

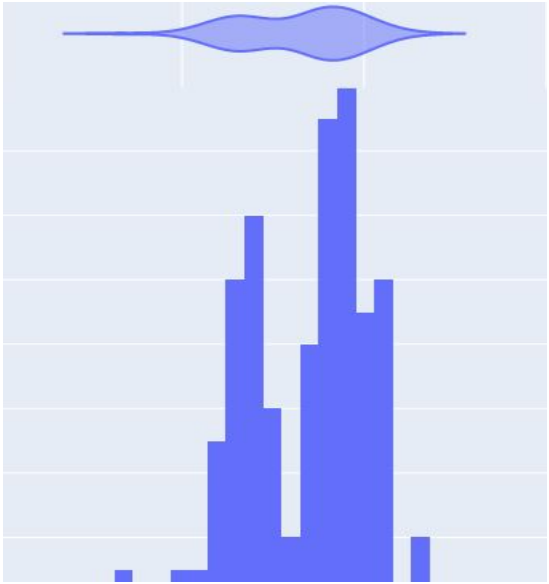
A "few" grains of salt...

Workload

An "nop" binary is executed 100 times



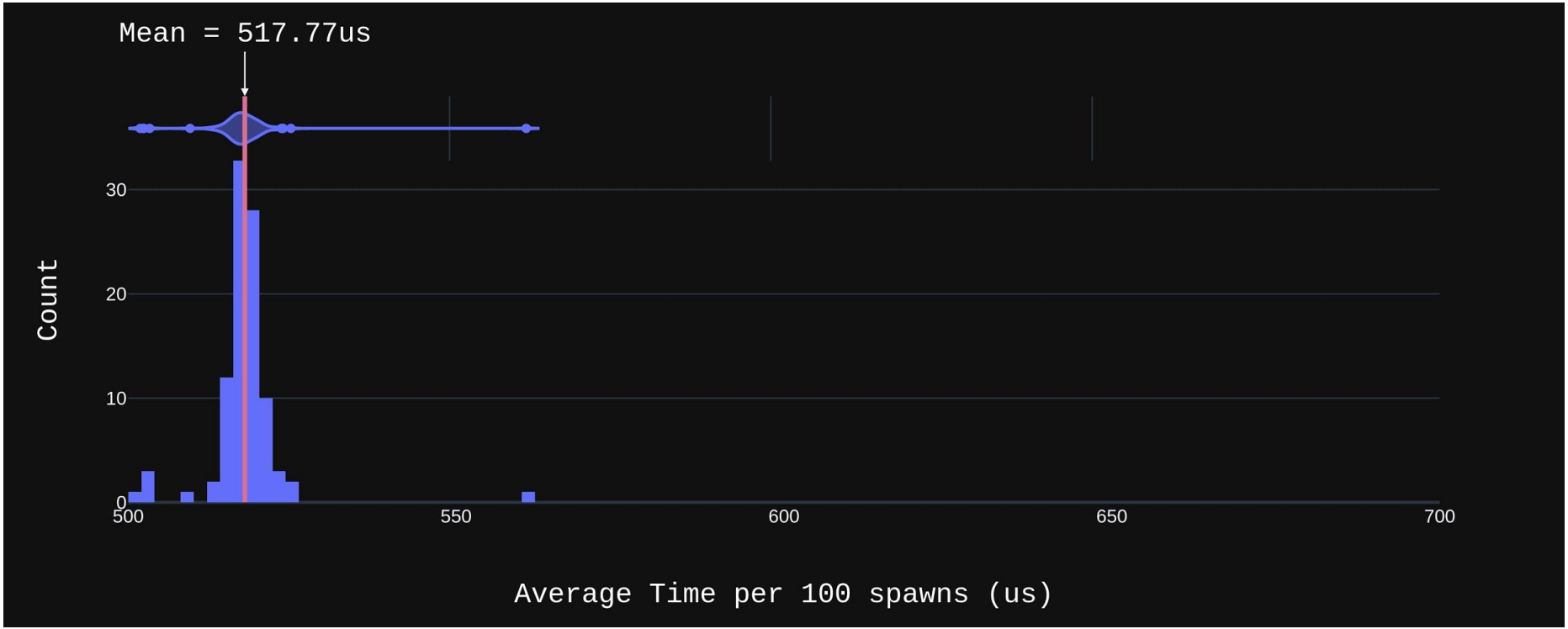
Average time for the 100 executions is measured.



Distribution is plotted of the N measurements

Google

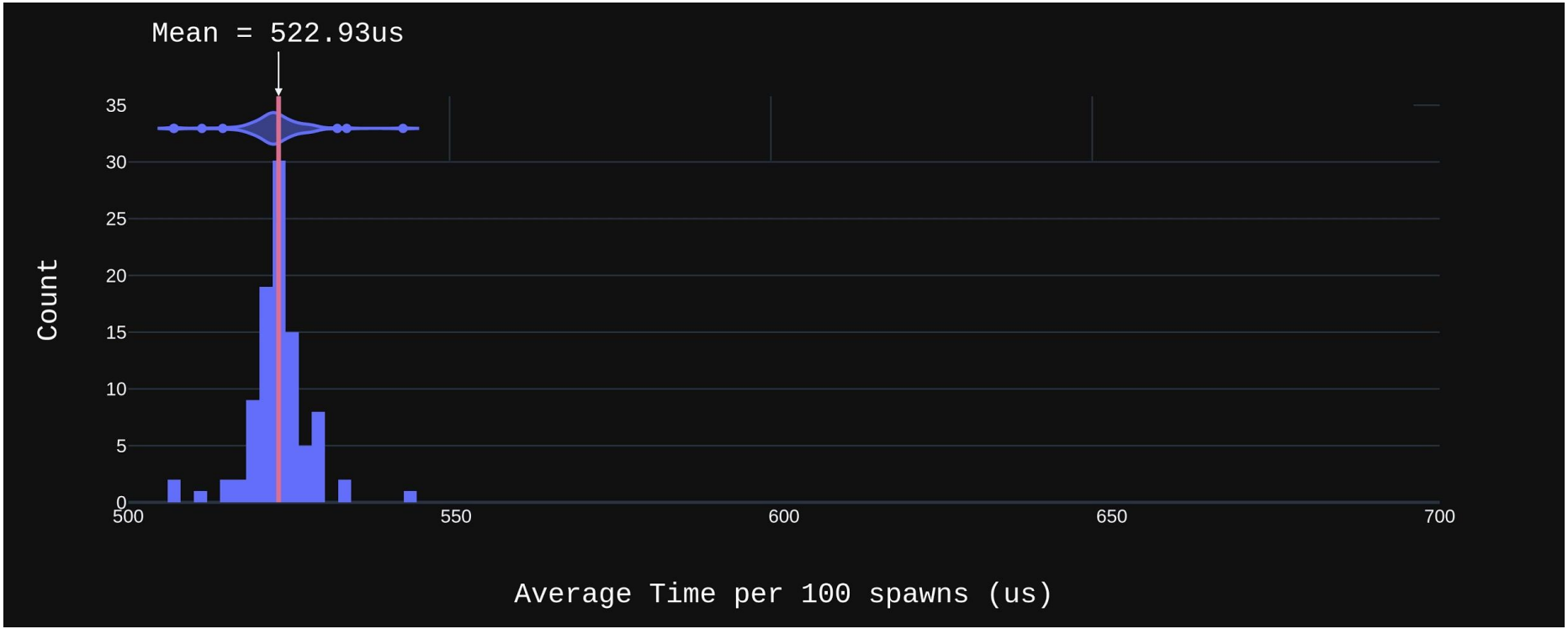
Vanilla System



CONFIG_AUDIT=no

Google

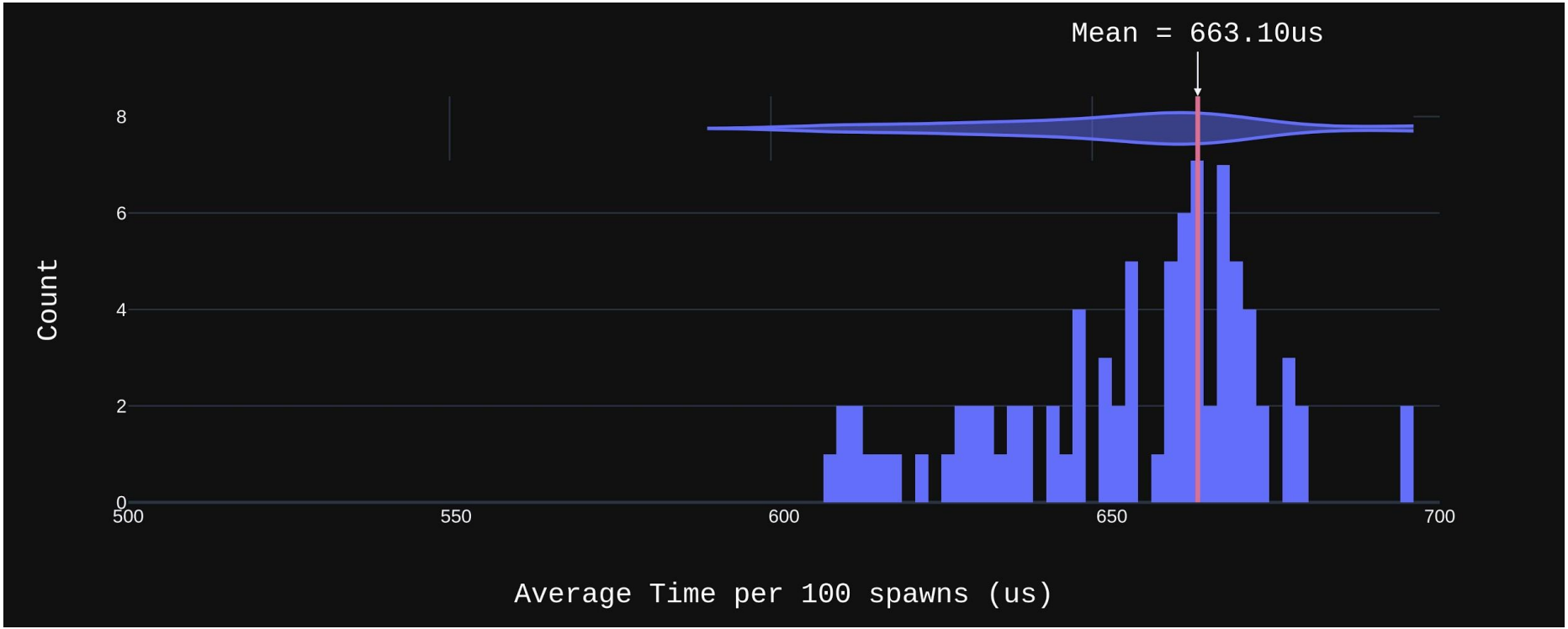
Audit (no Rules)



CONFIG_AUDIT=yes + no rules configured

Google

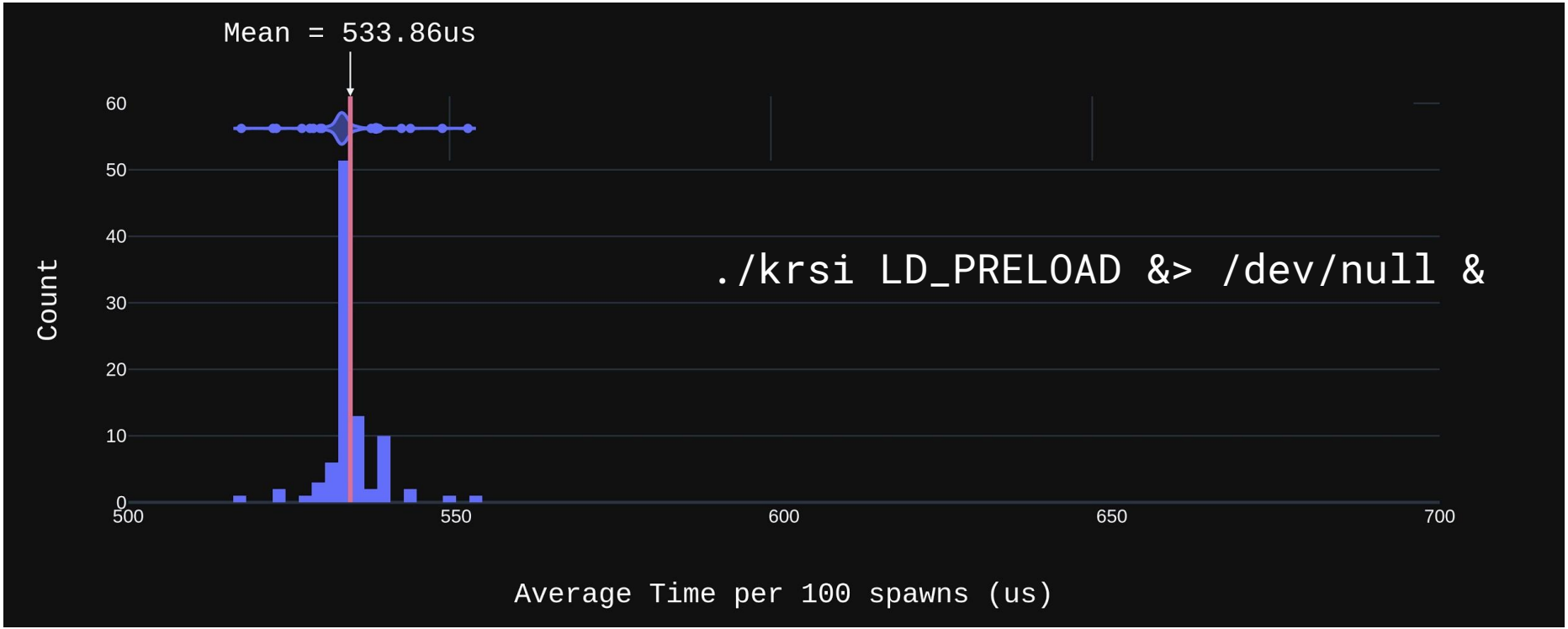
Audit



CONFIG_AUDIT=yes + execve syscall audit

Google

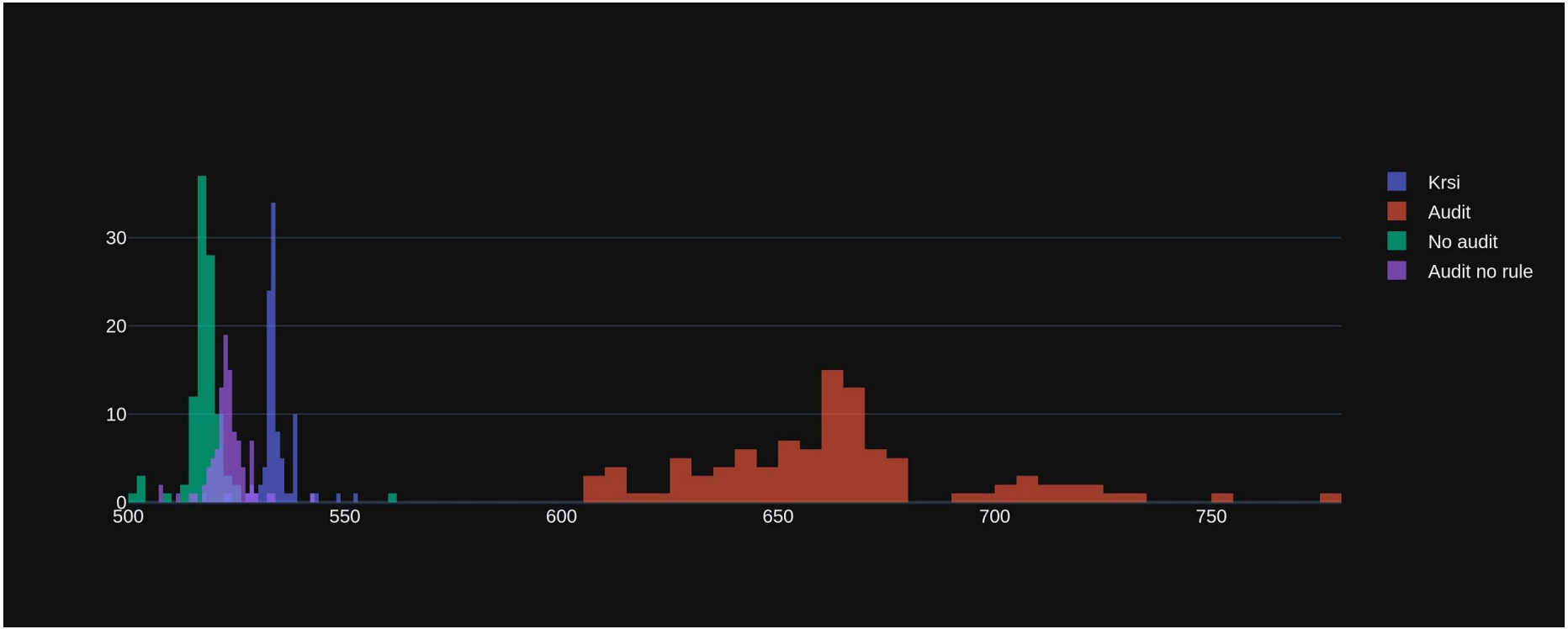
KRSI



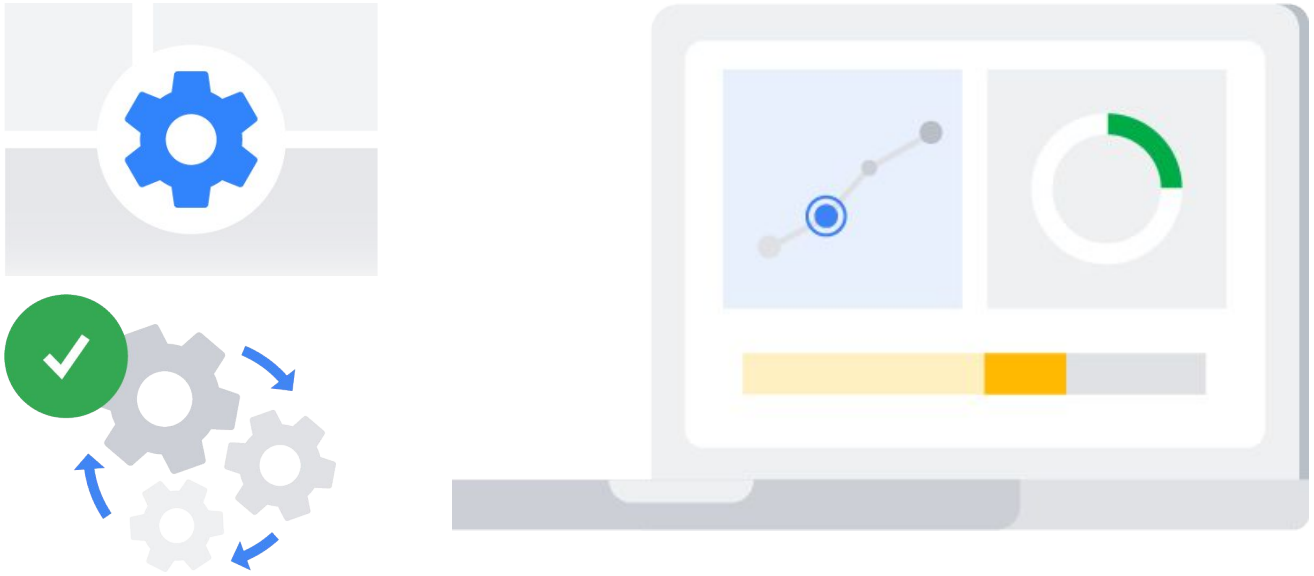
krsi LD_PRELOAD env var

Google

Comparison


Krsi
Audit
No audit
Audit no rule
30
20
10
0
500
550
600
650
700
750

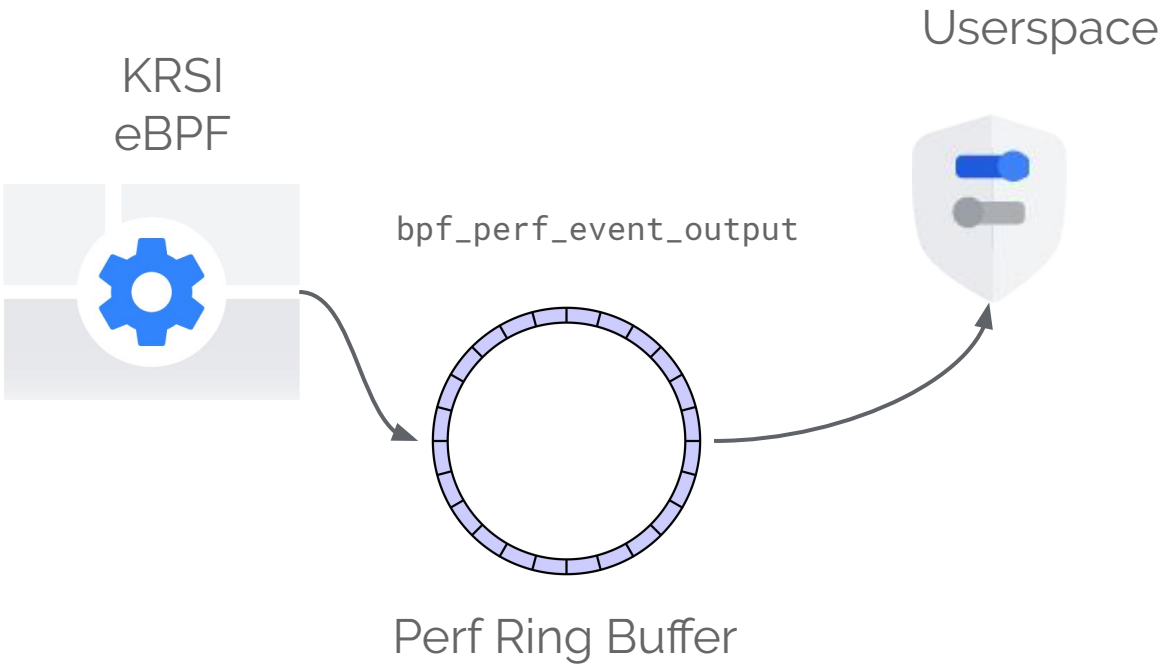
Demo



Google

Up for Discussion

Usage of the Perf Ring Buffer



Fast, and eBPF can already use it

Per CPU Buffers and memory usage

Sleepable eBPF

Makes the hooks simpler and saves memory.

Precomputation in the LSM hook

But eBPF programs cannot sleep! (yet...)

Pin the pages in the LSM hook and make them available to the helper's context

Selectively precompute only when an attached program calls the dependent helper.

Not needed if the eBPF programs are allowed to sleep (discussions are on..)

Thank You