Google

Kernel Runtime Security Instrumentation

KP Singh

Linux Plumbers Conference

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

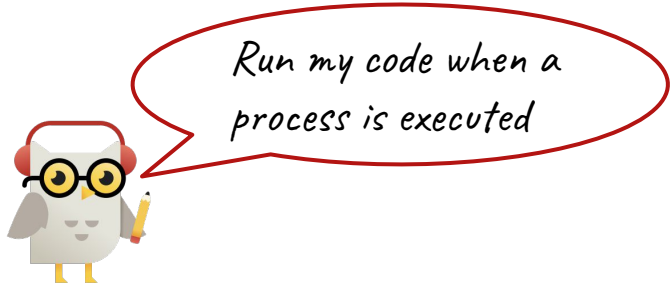
- Prevent mounting of USB drives on servers.
- Dynamic whitelist of known Kernel modules.
- Prevent known vulnerable binaries from running.

How does it work?

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.

Kernel Runtime Security Instrumentation



/sys/kernel/security/krsi/process_execution

(bprm_check_security)

my_bpf_prog.o

open [O_RDWR]

bpf [BPF_PROG_LOAD]

securityfs_fd

prog_fd

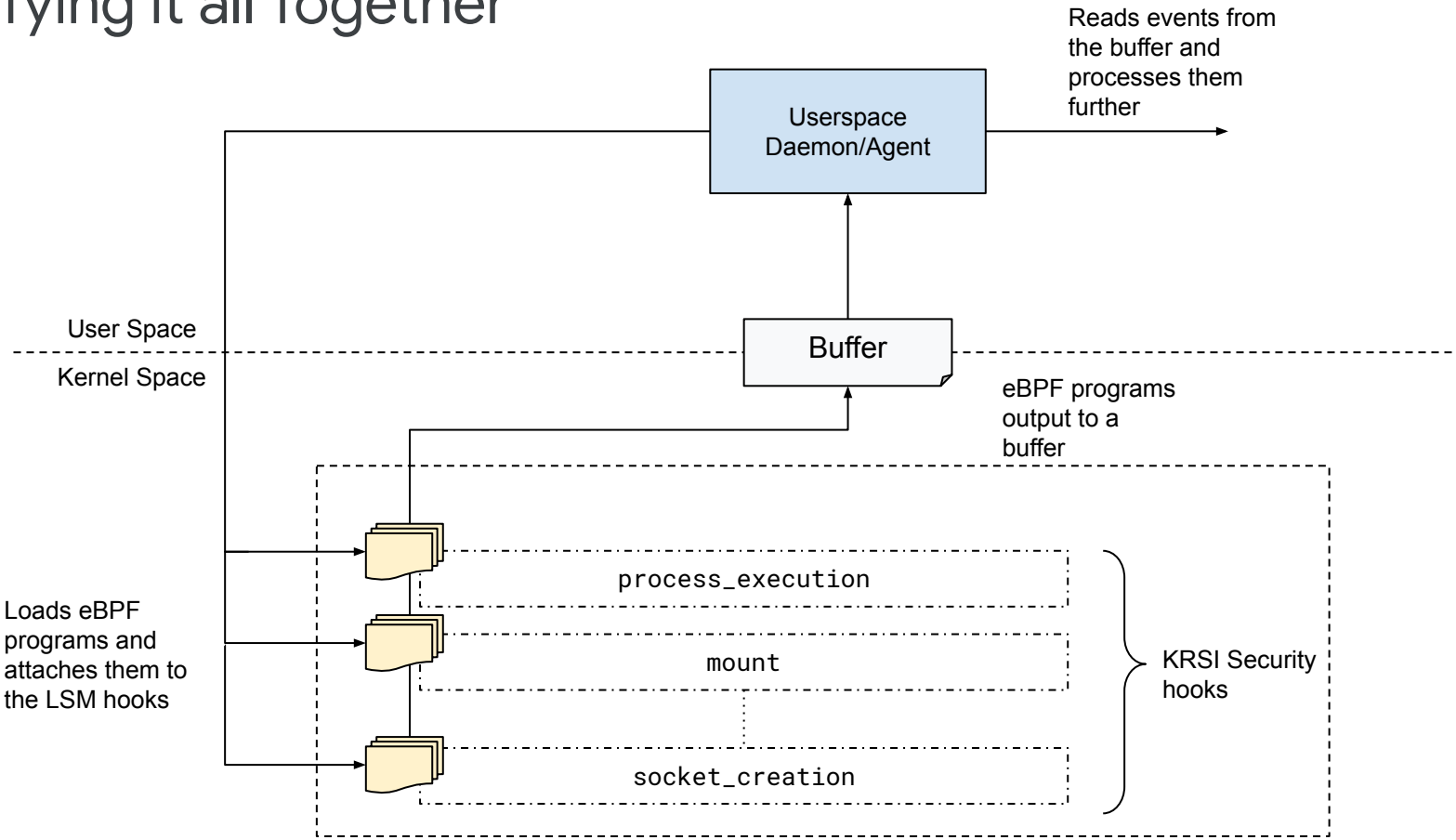
bpf [BPF_PROG_ATTACH]

LSM:bprm_check_security
(when a process is executed)

KRSI's Hook

Other LSM Hooks

Tying it all Together

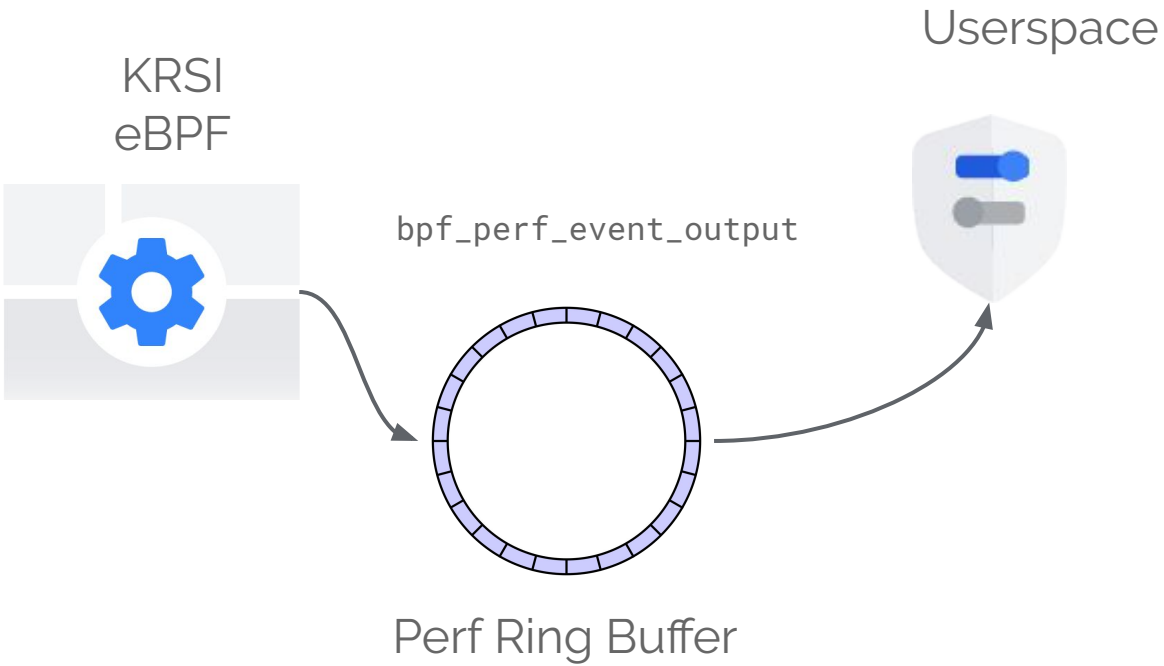


Key Design Principles

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

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

Usage of the Perf Ring Buffer



Fast, and eBPF can already use it

Per CPU Buffers and memory usage

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)

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..)