Seccomp and Capabilities

Tyler Caraza-Harter

Security

Principle of Least Priviledge: everything/everyone should have the minimum access to resources needed to function.

seccomp: limit available system calls

capabilities: reduce priviledge of root

Seccomp

Strict mode: only system calls available are read, write, and exit.

- great if you want to do untrusted pure computation in your environment
- approach:
  1. fork a new process
  2. open up only file descriptors you need (e.g., read only mode on some files)
  3. close all other FDs
  4. enter strict seccomp mode
  5. start executing untrusted code

Filter mode:

- we usually want a broader interface, even if its riskier
- Linux has >300 system calls. A subset works for most processes.
- Based on eBPF (extended Berkeley Packet Filters)
  - ➡ safe exec environment for running code in the kernel
  - ➡ can insert filters on packet handling, or on every syscall

Seccomp Filter Mode Options

Control:

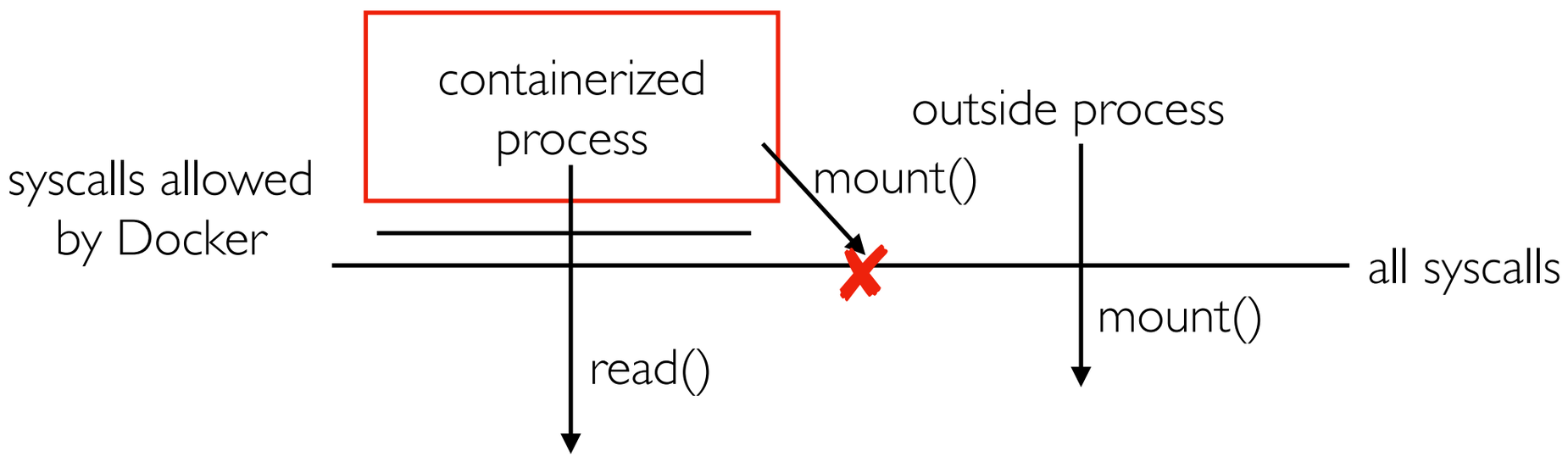
- for which syscalls we want to inject filters
- do we want to use this to define ALLOW lists or DENY lists?
- behavior on invalid syscall: log it, kill the process, return an error code (OpenLambda approach)

Seccomp in Docker

Profiles can be loaded defining which syscalls to allow. A default blocks syscalls with known dangers. The profile can be modified or disabled based on risk tolerance and needs of an application.

Example: "settimeofday" system call is blocked because there's not a namespace for time. If one container changed the time, it would change it for the whole system!

See notable blocked syscalls here: https://docs.docker.com/engine/security/seccomp/#significant-syscalls-blocked-by-the-default-profile



Seccomp in OpenLambda

seccomp was recently added to OpenLambda (2022).

Approach:

- use an ALLOW policy (so new calls default to not allowed)
- start with the syscalls that Docker allows
- see what breaks
- add a few more to make things work

In particular, we needed these that Docker disables:
`unshare, clone, clone3, chroot, arch_prctl`

Future work: support configurable profiles, like Docker.

Seccomp in gVisor (used by Google Cloud Functions)

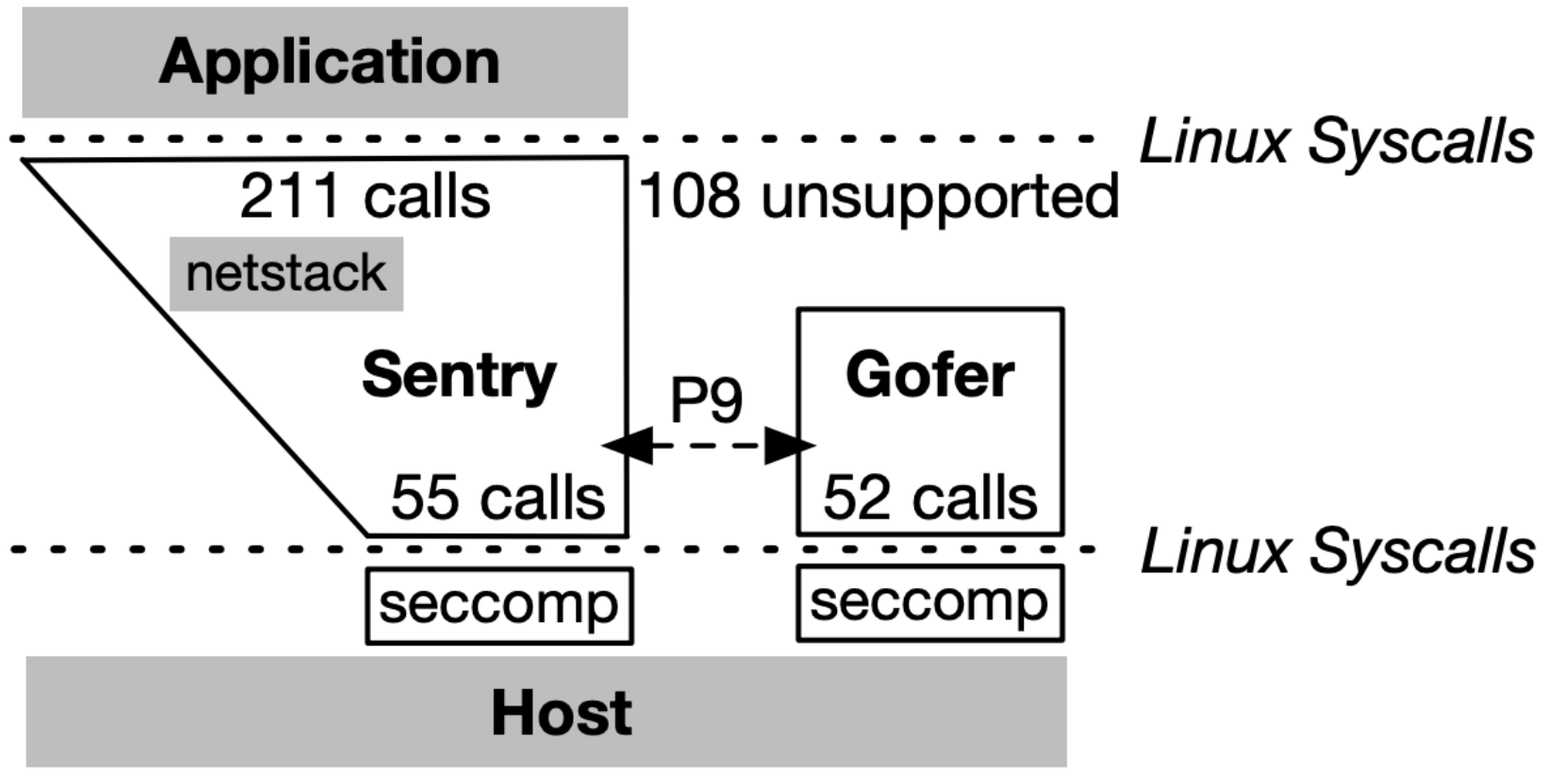


Figure 1: **gVisor Architecture.**

The True Cost of Containing: A gVisor Case Study

Ethan G. Young, Pengfei Zhu, Tyler Caraza-Harter,
Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau

University of Wisconsin, Madison

https://www.usenix.org/system/files/hotcloud19-paper-young.pdf

Seccomp in gVisor (used by Google Cloud Functions)

Defense in Depth: "In a real-life castle, there is no single defensive feature. Rather, there are many in combination: redundant walls, scattered draw bridges, small bottle-neck entrances, moats, etc."

https://gvisor.dev/blog/2019/11/18/gvisor-security-basics-part-1/

Outline

Performance Isolation

Mechanism Examples

Interface: cgroup overview

Controllers

- freezer/cpu/cpuset
- memory
- io (disk)
- pids
- what about network?

Usage in OpenLambda

Security

Principle of Least Priviledge: everything/everyone should have the minimum access to resources needed to function.

seccomp: limit available system calls

capabilities: reduce priviledge of root

Root

Most system calls check permissions: for example, if you're changing the owner on a file, are you the current owner?

With root privilidges, many of those checks don't occur.

```
$ some command # fails
$ sudo some command # yay!
```

Capabilities

This approach violates the principle of least priviledge!

A process shouldn't receive the ability to do EVERYTHING root can do just because it needs to do ONE thing root can do.

Solution: decompose root priviledges.

"Starting with kernel 2.2, Linux divides the privileges traditionally associated with superuser into distinct units, known as capabilities, which can be independently enabled and disabled. Capabilities are a per-thread attribute." CAPABILITIES(7) MANPAGES.

Capabilities vs. Seccomp

There's often overlap.

Consider the `settimeofday` syscall.

The default Docker seccomp profile blocks this. But if it didn't, it still wouldn't work for a regular user lacking CAP_SYS_TIME.

CAP_SYS_ADMIN Issue

The manpages have a note to kernel devs:

"Don't choose CAP_SYS_ADMIN if you can possibly avoid it! A vast proportion of existing capability checks are associated with this capability (see the partial list above). It can plausibly be called "the new root", since on the one hand, it confers a wide range of powers, and on the other hand, its broad scope means that this is the capability that is required by many privileged programs. Don't make the problem worse. The only new features that should be associated with CAP_SYS_ADMIN are ones that closely match existing uses in that silo."

Capabilities in OpenLambda

OpenLambda does not currently use capabilities, but this is probably one of the most important security features to implement, especially since OpenLambda is generally run as root.

Challenge: CAP_SYS_ADMIN (the over broad one) includes calls related to syscalls we need, such as `clone3, unshare, setns`.

Possible approaches to explore:

- find another way to expose the needed functionality without full CAP_SYS_ADMIN -- e.g., capabilities can be associated with programs that can be run by processes lacking those capabilities
- only drop capabilities in leaf nodes (this means continuing to give capabilities to Zygotes, which is perhaps OK if it is a currated list)