The Art of Userspace Sandboxing on Linux

—

Raunak Ramakrishnan

Why Sandboxing Matters

- On Linux, a process that a user starts usually has the credentials of that user.
- A sandbox runs a program with less access than its user has
- Common threat model:
  - Malicious dependency e.g npm/pip/cargo dependency of your project gets compromised
  - Reading and exfiltrating sensitive files e.g SSH keys, API keys, PII
  - DOS e.g due to mem/disk exhaustion
- Rootless sandboxes allow you to isolate software in environments where you do not have full control over the machine e.g CI runners

Namespaces

- A namespace gives a group of processes their own copy of a global resource.
-

Namespace type	Separate View Of
User	User and group IDs, capabilities
Mount	Mount points
PID	Process IDs
Network	Network interfaces, routes, firewalls, ports
Time	Boot and monotonic clocks

User Namespaces - Least Privilege Identity

- User namespaces let an unprivileged user map host UIDs/GIDs into a private namespace and appear as “root” inside it
- Processes inside a user namespace can perform privileged operations within that namespace but are still non-root on the host
- This enables rootless containers/sandboxes to do things like create filesystem layouts or configure other namespaces without host root.

Capabilities

- Linux divides the privileges of root into capabilities, for example CAP_SYS_ADMIN and CAP_NET_ADMIN
- If a thread drops a capability from its permitted set, it can never reacquire that capability
- Best practice is to use the capabilities for setup. Then remove all of them
- prctl(PR_SET_NO_NEW_PRIVS, 1): execve can no longer give privileges. Setuid bits and file capabilities stop working. The bit is permanent and child processes inherit it

cgroups

- A cgroup is a group of processes with shared resource limits and counters
- Each cgroup is a directory in /sys/fs/cgroup
- cgroups limit what a group uses e.g limit memory and CPU given to a process
- Generally, cgroup requires root to configure but it supports delegation
- If running under systemd, the "user@uid.service" in /sys/fs/cgroup is owned by the regular user.
- Use `systemd-run --user --scope -p TasksMax=10 -p MemoryMax=64M -- cmd` to run a command with resource limit

Seccomp: System Call Filtering

- Linux seccomp lets a process specify a filtering policy for system calls
- Seccomp filters limit what system calls a process can make into the kernel
- Filtering reduces kernel attack surface and enforces policy boundaries
- Seccomp filters are typically written in a BPF-like language and installed into the kernel; once installed, they are enforced for the life of the process

Seccomp Filtering - common pitfalls

- Syscall numbers are different for each ABI. Thus, a filter must check the `arch` field.
- On x86-64, The x86-64 ABI and the x32 ABI both use AUDIT_ARCH_X86_64 as arch. If using denylists, make sure to block x32 arch syscalls
- It is better to use allowlists rather than denylists as newer Linux kernels can add newer syscalls which perform similar functions e.g clone3, openat2
- Seccomp filter can not read pointers. Thus, it is difficult to inspect arguments for newer syscalls which take pointer args like clone3

Landlock: User-Space Policy Enforcement

- Landlock is a Linux Security Module (LSM) designed for unprivileged sandboxing
- Allows a process to restrict its own filesystem access
- Policies are enforced by the kernel and cannot be relaxed later
- Simple Interface with just 3 syscalls
  - **landlock_create_ruleset** to create a ruleset
  - **landlock_add_rule** to add rules
  - **landlock_restrict_self** to enforce a ruleset on a calling thread and its future children

ABI	Added
1	File execute, read, write, read-dir, remove, make-*
2	REFER: link or rename across directories
3	TRUNCATE
4	TCP bind and connect by port
5	ioctl on devices
6	Scoping of abstract UNIX sockets and signals
7	Audit logging flags for landlock_restrict_self
8	enforce on all threads of the process
9	connect to pathname UNIX sockets

Filesystem Views: Reducing What a Process Can See

- Rootless sandboxes rely on mount namespaces + bind/overlay mounts
- Bind mounts expose only required paths
- Overlayfs enables writable views over read-only bases
- FUSE (kernel module) allows you to mount disk images as a regular user.
  - Using fuse2fs, we can mount ext4 / squash / other Filesystem images
  - This allows us to create a rootfs with fixed capacity
- Landlock allows only exposing certain paths with customized permissions

Userspace Networking - Why is it not straightforward

- Linux provides no simple unprivileged primitive for networking
- A new network namespace starts with no network access
- Creating interfaces, routing, and NAT usually requires CAP_NET_ADMIN
  - Standard container networking uses veth pair + bridge
- Rootless sandboxes must rely on userspace TCP/IP stacks like gvisor-netstack, pasta or slirp4netns

Other Approaches

- Virtualization
  - Rootless if user has access to /dev/kvm or is part of kvm group
- Ptrace
  - Allows tracer to read/write arbitrary data to memory of the child process and pause/resume it
- Sentry Kernel like gvisor
  - Application kernel which reimplements Linux syscalls in golang
- DBT (Dynamic Binary Translation)
- WASM

Demo

- Let’s try sandboxing an “unsafe” script
- This script can do the following:
  - Spawn many child processes
  - Allocate lot of memory
  - Look for files in sensitive folders
  - Make HTTP connections
- We will progressively use namespaces, capabilities, seccomp, landlock and cgroups to limit various attacks

Demo Phase 0: An Unsafe Script

```
$ ./00-baseline.sh
```

Demo Phase 1: Using Namespaces and seccomp

```
$ ./01-namespaces-seccomp.sh
```

Demo Phase 2 - Using Landlock

```
$ ./02-private-tmp-landlock.sh
```

Demo Phase 3: Using delegated cgroups

```
$ ./03-cgroup.sh
```

Interesting Rootless Sandboxing Projects

- **Sandlock** - Uses Landlock + seccomp filters + seccomp_user_notify
- **Chimera** - Uses DBT + seccomp_user_notify
- **Oniux** - Allows you to run Tor as a TUN interface without root using unprivileged user namespaces + capabilities dropping and using smoltcp as a userspace netstack
- **Sockpuppet** (Code) - My hobby project inspired by Oniux. It is a zero external dependency sandbox written from scratch which allows using a SOCKS proxy as a TUN device, mounting custom rootfs and running your code in an overlayfs

References

- Understanding user namespaces with rootless containers
- Rootless containers using userns
- Linux Capabilities and Seccomp
- Docker Seccomp security profile
- Landlock
- Mount Namespaces
- Podman rootless mode

Let’s talk sandboxing

- Connect with me to talk about sandboxing, Linux internals and systems programming
- LinkedIn: https://www.linkedin.com/in/raunak1/