Landlock-ing Linux

Nov 29, 2025

Landlock: What Is It?

Landlock is a Linux API that lets applications explicitly declare which resources they are allowed to access. Its philosophy is similar to OpenBSD's `unveil()` and (less so) `pledge()`: programs can make a contract with the kernel stating, "I only need these files or resources — deny me everything else if I'm compromised."

It provides a simple, developer-friendly way to add defense-in-depth to applications. Compared to traditional Linux security mechanisms, Landlock is vastly easier to understand and integrate.

This post is meant to be an accessible introduction, and hopefully persuade you to give Landlock a try.

How Does It Work?

Landlock is a Linux Security Module (LSM) available since Linux 5.13. Unlike MAC frameworks such as SELinux or AppArmor, Landlock applies *transient* restrictions: policies are created at runtime, enforced on the current thread and its future descendants, and disappear when the process exits.

You don't tag files with labels or extended attributes. Instead, applications create policies dynamically.

A Landlock policy consists of two pieces:

1. **Handled accesses** — the categories of operations you want to restrict (e.g., filesystem read/write).
2. **Access grants** — an explicit allowlist of which objects are permitted for those operations.

For example, you could create a policy that handles all filesystem reads/writes and network binds, and grants:

- read-only access to `/home/user`
- read/write access to `/tmp`
- permission to bind to port `2222`

The application then calls `landlock_restrict_self()` to enter the restricted domain. From that point on, that thread's child threads and child processes are permanently constrained. Restrictions cannot be revoked.

Policies can be layered (up to 16 layers). A child layer may further *reduce* access, but cannot reintroduce permissions the parent layer removed. For example, a child thread may add a layer to this policy to restrict itself to only reading `/home/user`, but it cannot regain permission to bind to port `2222` once a layer omits this grant.

Landlock is unprivileged — any application can sandbox itself. It also uses ABI versioning, allowing programs to apply best-effort sandboxing even on older kernels lacking newer features.

It's also a stackable LSM, meaning you can combine it with selinux or apparmor in a supplemental layer.

Why Should You Use It?

Landlock shines when an application has a predictable set of files or directories it needs. For example, a web server could restrict itself to accessing only `/var/www/html` and `/tmp`.

Unlike SELinux or AppArmor, Landlock policies don't require administrator involvement or system-wide configuration. Developers can embed policies directly in application code, making sandboxing a natural part of the development process.

Because Landlock requires *no privileges* to use, adding it to most programs is straightforward.

Bindings exist for languages such as Rust, Go, and Haskell, and several projects provide user-friendly `unveil`-style wrappers.

A official c library doesn't exist yet unfortunately, but there's several out there you can try.

Here's a quick rust example:

```
use landlock::{
    ABI, Access, AccessFs, Ruleset, RulesetAttr, RulesetCreatedAttr, RulesetSta
    path_beneath_rules,
};

fn restrict_thread() -> Result<(), RulesetError> {
    let abi = ABI::V1;
    let status = Ruleset::default()
        .handle_access(AccessFs::from_all(abi))?
        .create()?
        // Read-only access to /usr, /etc and /dev.
        .add_rules(path_beneath_rules(&["/usr", "/etc", "/dev"], AccessFs::from
        // Read-write access to /home and /tmp.
        .add_rules(path_beneath_rules(&["/home", "/tmp"], AccessFs::from_all(ab
        .restrict_self()?;

    match status.ruleset {
        RulesetStatus::FullyEnforced => println!("Fully sandboxed."),
        RulesetStatus::PartiallyEnforced => println!("Partially sandboxed."),
        RulesetStatus::NotEnforced => println!("Not sandboxed! Please update yo
    }
    Ok(())
}
```

The State of Linux Sandboxing: Why This Matters

As Linux adoption grows, so does the amount of malware targeting desktop users. While Linux has historically enjoyed relative safety, this is largely due to smaller market share and higher technical barriers compared to Windows — not because Linux is inherently safer.

Linux is not a security panacea. For example, on most major distributions:

- Users can download and execute untrusted binaries with no warnings.

- Shell scripts can be piped from the internet and executed blindly.
- Many users run passwordless sudo, giving them root access on demand.
- Unprivileged applications can typically:
  - Read `~/.ssh`, `~/.bashrc`, browser cookies, and anything else in `$HOME`
  - Modify environment variables and `$PATH`
  - Create systemd user services
  - (on X11) log keystrokes and read input devices
  - Bind to arbitrary network ports

Several tools try to improve the state of security on linux, but each has significant drawbacks:

Containerization (docker, podman)

- Designed for service isolation, not desktop apps.
- Managing home directory access is clunky.
- Many users break isolation by using `--privileged` or `--network host`.

Flatpak / Snap

- Great for graphical applications (Flatpak especially).
- Often require overly broad permissions.
- Less suitable for CLI tools.

Firejail

- Requires per-application profiles.
- Must be explicitly invoked each time, or you need a wrapper script.

From the developer side:

seccomp

- Powerful syscall filtering.
- Tedious and error-prone to configure.
- Blacklists are fragile; new syscalls can break things.
- Argument filtering is difficult and full of TOCTOU hazards.

SELinux

- Extremely powerful, but difficult to understand.

- Requires system-wide policies and admin involvement.
- Many users disable it due to complexity.
- Not enabled on most distributions by default. (used a lot in android)

AppArmor

- Easier than SELinux, but still requires admin-defined profiles.
- Applies system-wide and lacks per-process namespacing.
- Gets disabled by many distributions, but is more commonly used in the desktop.

Landlock

- Unprivileged
- Application-centric
- Easy to integrate
- Deny-by-default
- Widely supported since 5.13
- Backward and forward compatibility mechanisms.

Landlock isn't perfect, but it fills a major gap: a simple, self-contained unprivileged sandboxing tool.

What landlock could bring to the table:

Long-running system daemons that run with elevated privileges could benefit from landlock restrictions.

Desktop applications dealing with binary formats, like pdf readers, image viewers web browsers, and word processors can be restricted to accessing the files they originally opened.

FTP and HTTP servers can be bound to the files they need. Even if nginx is running as root, if an attacker gets a full reverse shell, they won't be able to see access files outside the policy.

If the supervisor proposal gets added, we could bring an android-like permissions system to the linux desktop. Flatpak does a decent job at this, but imagine if every process in your desktop would need to explicitly ask (at least once) before accessing sensitive files or resources.

Pair that with an accessible GUI and a system for handling updates and saving permission grants, and we have potential for a safer, more secure linux user experience on the desktop.

Ongoing Work in Landlock

Several promising features are under active development:

- **Supervise Mode**
  Lets a userspace “supervisor” interactively allow or deny access — similar to Android-style permission prompts.

- **Socket Restrictions** Fine-grained control over which types of sockets or ports processes may use.

- **LANDLOCK_RESTRICT_SELF_TSYNC** Ensures restrictions propagate to all threads in a process.

- **LANDLOCK_ADD_RULE_QUIET** Allows suppressing audit messages for certain objects.

- **LANDLOCK_ADD_RULE_NO_INHERIT** *(disclosure: this is my patch series)* Prevents rules from unintentionally inheriting permissions from parent directories, giving finer-grained filesystem control.

TL;DR

Landlock is a simple, unprivileged, deny-by-default sandboxing mechanism for Linux.
It’s easy to understand, easy to integrate, and has tremendous potential for improving desktop and application security.

Give it a try in your application.