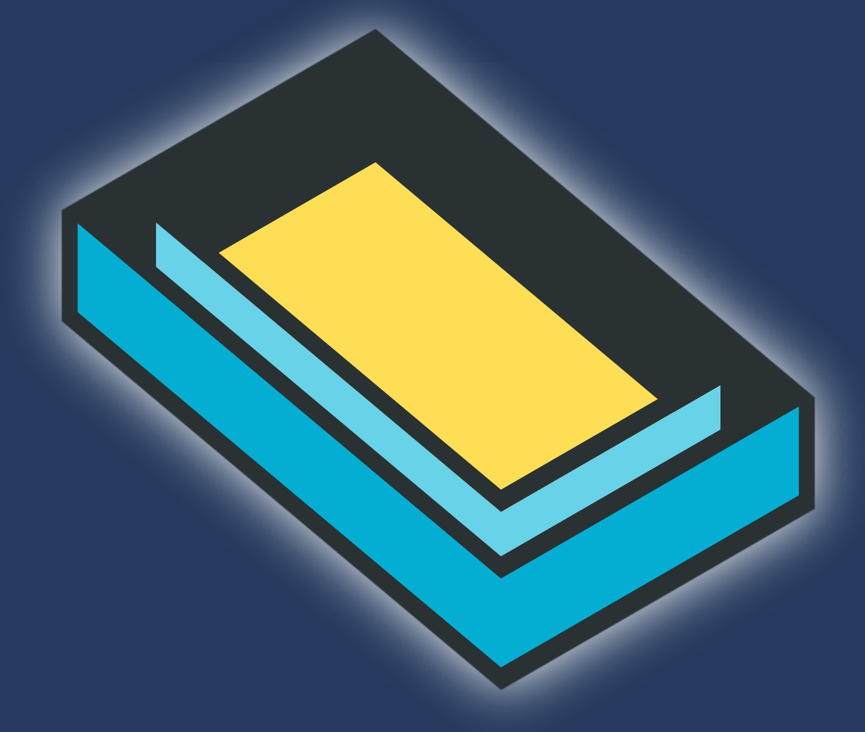
Microsoft

Landlock Config

Linux Security Summit Europe

Mickaël Salaün – kernel maintainer



2025-08-29

Sandboxing with Landlock

Sandboxing: “a **restricted**, controlled **execution environment** that prevents potentially malicious software […] from accessing any system resources except those for which the software is authorized.”

Landlock: unprivileged sandboxing mechanism provided by the Linux kernel

Landlock status

Landlock helpers

Examples of sandbox tools:

- setpriv
- Minijail
- Firejail

Examples of sandbox libraries:

- Landlock Rust crate
- Landlock Go library
- Minijail
- Pledge for Linux

Landlocked apps

Examples of various sandboxed apps:

- Zathura (document viewer)
- Pacman (package manager)
- Cloud Hypervisor (VM monitor)
- Suricata (network IDS)
- Polkadot (blockchain SDK)
- wireproxy (Wireguard client)
- GNOME LocalSearch (search engine)
- XZ Utils (archive manager)

Landlock properties

Use case #1

Exploitable bugs in trusted applications: protect from vulnerable code maintained by developers.

Candidates:

- Parsers: archive tools, file format conversion, renderers…
- Web browsers
- Network and system services

Use case #2

Untrusted applications: protect from potentially malicious third-party code.

Candidates:

- Container runtimes
- Init systems
- Sandboxer tools

Current access control

Implicit restrictions

- Process impersonation (e.g., ptrace)
- Filesystem topology changes (e.g., mounts), when it makes sense

Explicit access rights

- Filesystem
- Networking
- Signaling
- Abstract unix socket

Landlock ABI versions

1. Linux 5.13: Initial set of FS access rights
2. Linux 5.19: Rename and link
3. Linux 6.2: Truncation
4. Linux 6.7: TCP connect and bind
5. Linux 6.10: IOCTL for devices
6. Linux 6.12: Signal and abstract UNIX socket
7. Linux 6.15: Log configuration

How does Landlock work?

Restrict ambient rights according to the **kernel semantic** (e.g., global filesystem access) for a set of processes, thanks to **3 dedicated syscalls**.

Security policies are inherited by all new children processes.

A one-way set of restrictions: cannot be disabled once enabled.

Landlock interface (in C and Rust)

Step 1: Check backward compatibility

```
int abi = landlock_create_ruleset(NULL, 0, LANDLOCK_CREATE_RULESET_VERSION);
if (abi < 0)
    return 0;
```

Step 2: Create a ruleset

```c
int ruleset_fd;
struct landlock_ruleset_attr ruleset_attr = {
    .handled_access_fs =
        LANDLOCK_ACCESS_FS_EXECUTE |
        LANDLOCK_ACCESS_FS_WRITE_FILE,
};

ruleset_fd = landlock_create_ruleset(&ruleset_attr,
                                     sizeof(ruleset_attr), 0);
if (ruleset_fd < 0)
    error_exit("Failed to create a ruleset");
```

```rust
Ruleset::default()
    .handle_access(make_bitflags!(
        AccessFs::{Execute | WriteFile}))?
    .create()?
```

Step 3: Add rules

```c
int err;
struct landlock_path_beneath_attr path_beneath = {
    .allowed_access = LANDLOCK_ACCESS_FS_EXECUTE,
};

path_beneath.parent_fd = open("/usr",
               O_PATH | O_CLOEXEC);
if (path_beneath.parent_fd < 0)
    error_exit("Failed to open file");

err = landlock_add_rule(ruleset_fd,
         LANDLOCK_RULE_PATH_BENEATH, &path_beneath, 0);
close(path_beneath.parent_fd);
if (err)
    error_exit("Failed to update ruleset");
```

```rust
Ruleset::default()
    .handle_access(make_bitflags!(
        AccessFs::{Execute | WriteFile}))?
    .create()?
    .add_rule(
        PathBeneath::new(PathFd::new("/usr")?)
        .allow_access(AccessFs::Execute)
    )?
```

Step 4: Enforce the ruleset

```c
if (prctl(PR_SET_NO_NEW_PRIVS, 1, 0, 0, 0))
    error_exit("Failed to restrict privileges");

if (landlock_restrict_self(ruleset_fd, 0))
    error_exit("Failed to enforce ruleset");

close(ruleset_fd);
```

```rust
Ruleset::default()
    .handle_access(make_bitflags!(
        AccessFs::{Execute | WriteFile}))?
    .create()?
    .add_rule(
        PathBeneath::new(PathFd::new("/")?)
        .allow_access(AccessFs::Execute)
    )?
    .restrict_self()?
```

[Full example in C](#)

[Full example in Rust](#)

A new configuration format

Use cases

Developers:

- Ease sandboxing of their programs

Sysadmins:

- Sandbox scripts and system services

End users:

- Sandbox apps

Two complementary formats

JSON:

- Well known standard with schema validation
- Easy to include in other configurations
- Flexible for advanced users (e.g., jq)

TOML:

- Designed for end users

Configuration example in JSON

```
{
 "variable": [ {
    "name": "rw",
    "literal": [ "/tmp", "/var/tmp", "/home/user/tmp" ]
  } ],
 "pathBeneath": [ {
    "allowedAccess": [ "v4.read_execute" ],
    "parent": [ "/bin", "/lib", "/usr", "/dev", "/proc", "/etc", "/home/user/bin" ]
   },
   {
    "allowedAccess": [ "v4.read_write" ],
    "parent": [ "${rw}" ]
   } ]
}
```

Configuration example in TOML

```
[[variable]]
name = “rw"
literal = ["/tmp", "/var/tmp", "/home/user/tmp"]

# Main system file hierarchies can be read and executed.
[[path_beneath]]
allowed_access = ["v4.read_execute"]
parent = ["/bin", "/lib", "/usr", "/dev", "/proc", "/etc", "/home/user/bin"]

# Only allow writing to temporary and home directories.
[[path_beneath]]
allowed_access = ["v4.read_write"]
parent = ["${rw}"]
```

Properties

- Ease sharing and maintaining security policies
- Declarative and deterministic
- Customizable

Shared policies

Requirements:

- Standalone snippets tailored to specific programs
- Handle different set of access rights

Several sources:

- Provided by upstream developers (independent from distros)
- Provided by distro packages
- Provided by end users, communities

Policies composition

Composition

Backward and forward compatibilities: because Landlock is gaining new features over time, using different snippets from different sources requires careful consideration.

Because of denied-by-default policies, access rights are leveled down to be compatible together, but exceptions/rules are added together.

Customization

Handle variables and compose them commutatively:

- Variables are a set of values
- Must be defined when using it, but can be empty

Individual access rights or groups:

- read_execute
- read_write
- all

Good practice

- One snippet per minimal unit of update (e.g., package)
- End users can add their own snippets (e.g., defining custom values for variables)

Example of composition: two files

Snippet #1

```
[[variable]]
name = “rw"
literal = ["/tmp", "/var/tmp"]

[[path_beneath]]
allowed_access = ["v5.read_execute"]
parent = ["/bin", "/lib", "/usr", "/dev", "/proc", “/etc"]

[[path_beneath]]
allowed_access = ["v5.read_write"]
parent = ["${rw}"]
```

Snippet #2

```
[[variable]]
name = "rw"
literal = ["/home/user/tmp"]

[[ruleset]]
handled_access_fs = ["v4.all"]

[[path_beneath]]
allowed_access = ["v4.read_execute"]
parent = ["/home/user/bin"]
```

Example of composition: one configuration

```
[[path_beneath]]
allowed_access = ["v4.read_execute"]
parent = ["/bin", "/lib", "/usr", "/dev", "/proc", "/etc", "/home/user/bin"]

[[path_beneath]]
allowed_access = ["v4.read_write"]
parent = ["/tmp", "/var/tmp", "/home/user/tmp"]
```

Library

- Rust crate
- Shared object library with C binding
- JSON schema
- Well tested

Wrap-up

Try Landlock Config

```
$ git clone https://github.com/landlock-lsm/landlockconfig

$ cd landlockconfig

$ cargo run --example sandboxer -- \
                    --json examples/mini-write-tmp.json sh
```

Contribute

- Develop new access types
- Improve libraries: Rust, Go…
- Challenge the implementation
- Improve documentation or tests
- **Sandbox your applications** and others'
  - Secure Open Source Rewards
  - Google Patch Rewards

Questions?

landlock@lists.linux.dev

Thank you!