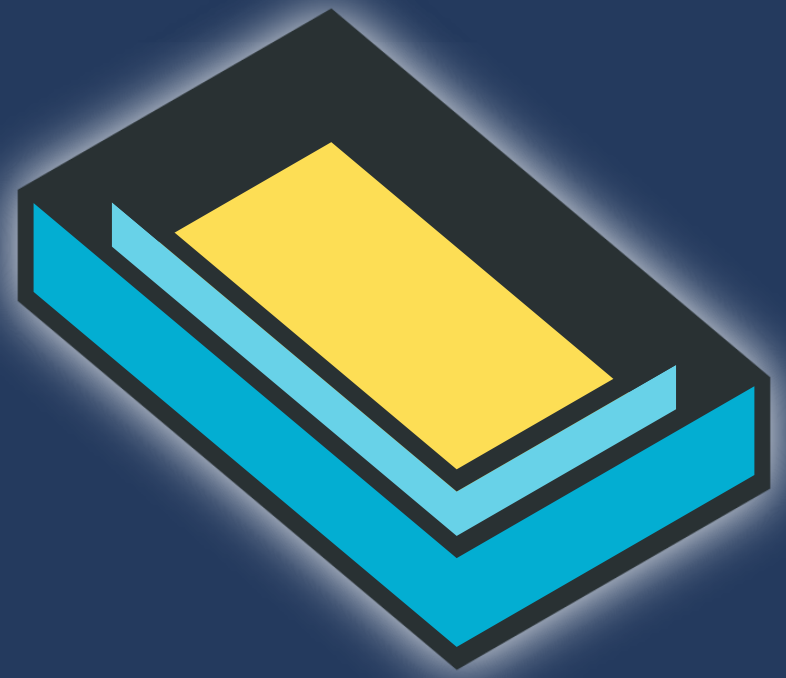
Microsoft

Sandboxing services with Landlock



All Systems Go!

Mickaël Salaün – kernel maintainer

2025-09-30

Sandboxing with Landlock

Sandbox: “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

Key Landlock features

Unprivileged

- Dynamic and ephemeral restrictions: no persistent state, no file labels
- Independent restrictions: the kernel manages a set of standalone policies per user, service, program…
- Nested sandboxes
- One-way restrictions: cannot be disabled once enabled for a process hierarchy.

Access control

- Configuration not explicitly tied to system calls, but to the kernel semantic: no need to synchronize with library/code updates using new syscalls
- Orthogonal to namespaces: only restrict access, do not build “views” of kernel resources (e.g. filesystem, network)

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 without being able to escape their sandbox.

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:

- Sandboxer tools
- Container runtimes
- **Init systems**

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

Landlock support for systemd

Why?

Making Landlock sandboxing features easy to use for all users! Complementary to Seccomp, AppArmor…

Proposed support:

- System and **user** units
- All “executable” units (e.g., service)

Pull Request #39174 · systemd/systemd

Example of sandboxed service

```
[Service]
Type=oneshot
ExecStart=/usr/bin/foo
LandlockConfig=/usr/lib/foo/landlock/

# LandlockConfig=%E/landlock/%p
```

Content of a Landlock Config directory

One configuration file per minimal unit of update (e.g., package).

/usr/lib/foo/landlock/

- lib-A.toml
- lib-B.toml
- prog-X.toml
- prog-Y.toml

Landlock Config

Configuration example in TOML

```
abi = 4

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

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

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

Properties

- Ease sharing and maintaining security policies
- Declarative and deterministic
- Customizable
- 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 scoped to a specific ABI: read_execute, read_write, all

Composed and shared policies

Requirements:

- Standalone files/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

Library

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

Wrap-up

Try Landlock Config

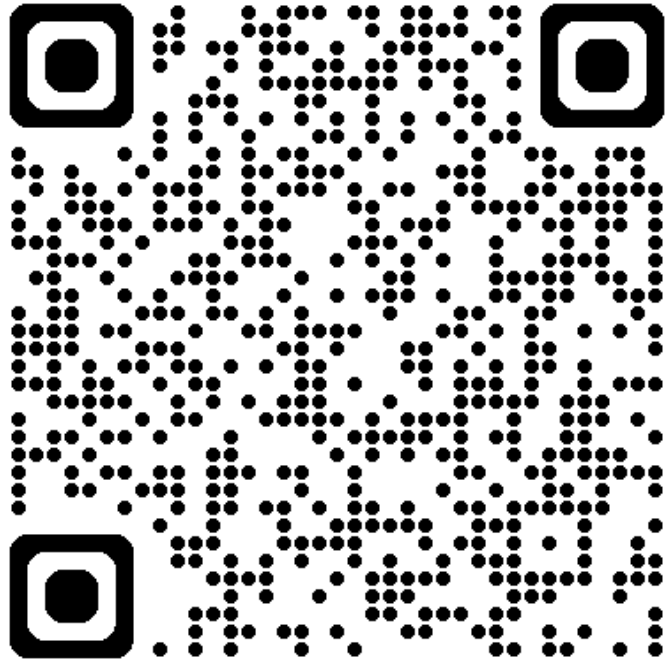
```
$ git clone https://github.com/landlock-lsm/landlockconfig

$ cd landlockconfig

$ cargo run --example sandboxer -- --debug \
                    --toml tests/composition/source/ sh
```

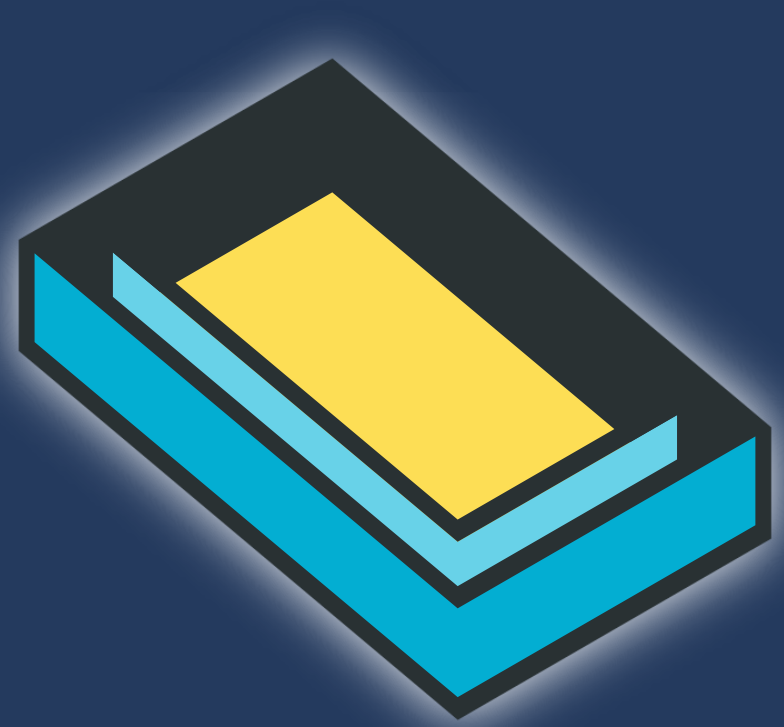
Ongoing work

- Make the configuration format future-proof
- Integrate to systemd
- Extend variable use to handle environment (e.g., XDG variables)



Contribute

- Develop new access types
- Improve libraries: Landlock Config, Rust, Go
- Challenge implementations
- Improve documentation or tests
- **Sandbox your applications** and others'
  - Alpha Omega
  - Open Source Security Patch Rewards



Questions?

landlock@lists.linux.dev

Thank you!

Annex

Configuration example in JSON

```
{
  "abi": 4,
  "variable": [
     "name": "rw",
     "literal": [ "/tmp", "/var/tmp", "/home/user/tmp" ]
   } ],
 "pathBeneath": [ {
     "allowedAccess": [ "abi.read_execute" ],
     "parent": [ "/bin", "/lib", "/usr", "/dev", "/proc", "/etc", "/home/user/bin" ]
    },
    {
     "allowedAccess": [ "abi.read_write" ],
     "parent": [ "${rw}" ]
    } ]
}
```

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

Composition

Backward and forward compatibilities: because Landlock is gaining new features over time, using different configuration files 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.

Example of composition: two files

File #1

```
abi = 5

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

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

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

File #2

```
abi = 4

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

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

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

Example of composition: one configuration

Composition of file #1 with file #2

```
abi = 4

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

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