Microsoft

Azure

Deep Dive into Landlock Internals

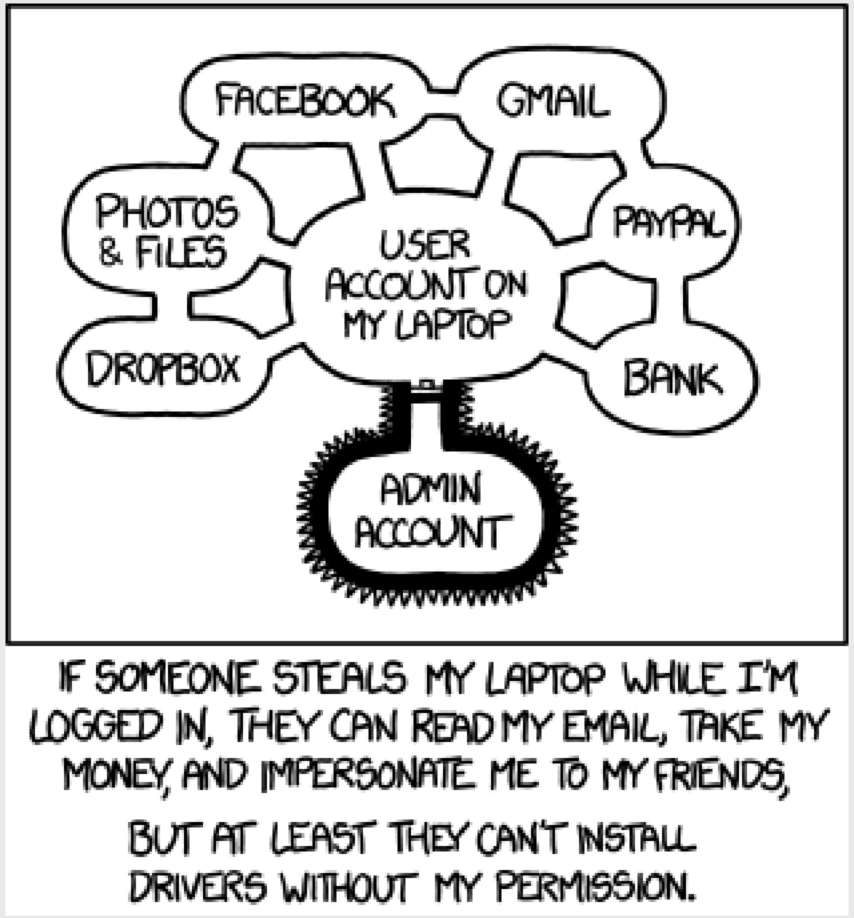
Linux Security Summit

Mickaël Salaün

Senior Software Engineer

2021-09-29

User data


FACEBOOK
GMAIL
PHOTOS & FILES
PAYPAL
USER ACCOUNT ON MY LAPTOP
DROPBOX
BANK
ADMIN ACCOUNT
IF SOMEONE STEALS MY LAPTOP WHILE I'M LOGGED IN, THEY CAN READ MY EMAIL, TAKE MY MONEY, AND IMPERSONATE ME TO MY FRIENDS,
BUT AT LEAST THEY CAN'T INSTALL DRIVERS WITHOUT MY PERMISSION.

https://xkcd.com/1200

Plan

Goal of Landlock and how to use it

A bit of history

Consequences of an unprivileged access-control

Design and implementation details

Testing strategy

Current and future limitations

What is (security) sandboxing?

A security approach to isolate a software component from the rest of the system.

An innocuous and trusted process can become malicious during its lifetime because of bugs, exploited by attackers or just triggered by users.

Threat models:

- Protecting from vulnerable code maintained by the developer.
- Protecting from malicious **third-party** code.
- Can be defined by the developers as it fit best.

What is Landlock?

Landlock is the first Mandatory Access Control available to unprivileged processes on Linux (since 5.13).

It enables to develop built-in application sandboxing to protect against:

- Exploitable bugs in trusted applications (embedded policy)
- Untrusted applications (sandbox or container managers)

How to use Landlock?

Three future-proof system calls:

- landlock_create_ruleset()
- landlock_add_rule()
- landlock_restrict_self()

Current access-control features: filesystem

Allow a thread (and its future children) to access to a set of file hierarchies:

- Execute, read or write to a file
- List a directory or remove files
- Create files

```
LANDLOCK_ACCESS_FS_EXECUTE
LANDLOCK_ACCESS_FS_WRITE_FILE
LANDLOCK_ACCESS_FS_READ_FILE
LANDLOCK_ACCESS_FS_READ_DIR
LANDLOCK_ACCESS_FS_REMOVE_DIR
LANDLOCK_ACCESS_FS_REMOVE_FILE
LANDLOCK_ACCESS_FS_MAKE_CHAR
LANDLOCK_ACCESS_FS_MAKE_DIR
LANDLOCK_ACCESS_FS_MAKE_REG
LANDLOCK_ACCESS_FS_MAKE_SOCK
LANDLOCK_ACCESS_FS_MAKE_FIFO
LANDLOCK_ACCESS_FS_MAKE_BLOCK
LANDLOCK_ACCESS_FS_MAKE_SYM
```

Step 1: 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 |
        […]
        LANDLOCK_ACCESS_FS_MAKE_REG,
};

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

Step 2: 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");
```

Step 3: Enforce the ruleset

```
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);
```

Landlock, a bit of history

[PATCH v1] (Mar. 2016): seccomp-object

[PATCH v2] (Aug. 2016): LSM + eBPF + cgroups

[PATCH v8] (Feb. 2018): file path identification

[PATCH v10] (Jul. 2019): shrink patches

[PATCH v14] (Feb. 2020): revamp without eBPF + 1 dedicated syscall

[PATCH v21] (Oct. 2020): switch to 3 syscalls

[PATCH v34] (Apr. 2021): merged in mainline for Linux 5.13

Why no more eBPF?

eBPF is very powerful and can be leveraged by attackers against the kernel (e.g., verifier bugs, Spectre): eBPF is not meant to be used by unprivileged users anymore.

Programmable interface with I/O (e.g., maps) can lead to side-channel attacks against other programs.

Not possible to efficiently compose (loaded) programs (i.e., only stack them).

Still contributed to bootstrap the BPF LSM (previously KRSI).

Priorities and guiding principles

1. Don’t weaken the system security by adding new features.
2. Account required resources to sandbox processes: processing and memory.
3. Protect un-sandboxed or less-sandboxed processes from more-sandboxed processes: confused deputy attack protection.
4. Sandboxing should be useful to limit access to data.

Unprivileged access control

Sought properties

Multiple and different applications: independent but **innocuous and composable security policies**.

Prevent bypass through other processes.

Follow the **least privilege principle** (i.e., no SUID).

Limit the kernel attack surface: simple policy declaration, without bytecode.

Composed security policies

Compose with other access-control systems: LSM stacking.

Compose all Landlock sandbox policies.

LSM stacking

Each LSM can register:

- Hooks for a set of actions (e.g., open a file, send a network packet)
- Blob sizes for a set of kernel object types (e.g., inode, file, socket, process)

The kernel denies an action when a first hook call returns an error: sequential checks.

Sandbox policies composition

Overview

Multiple sandbox layers

All applications (e.g., shells) are allowed to create their own sandbox, which may create hierarchies.

Inherit parent policies

A sandbox can only drop more accesses.

Sandbox policies composition

File identification constraints

No extended attributes

- Must handle multiple policies
- Must enable to embedded policies: ephemeral identification (e.g., app updates)
- Should be able to deal with read-only files

No (absolute) path

- May not have access to the real root (e.g., in a container)
- Must not be a way to bypass (other) access-control systems (e.g., side-channel attacks)

Sandbox policies composition

File identification design

Inode tagging

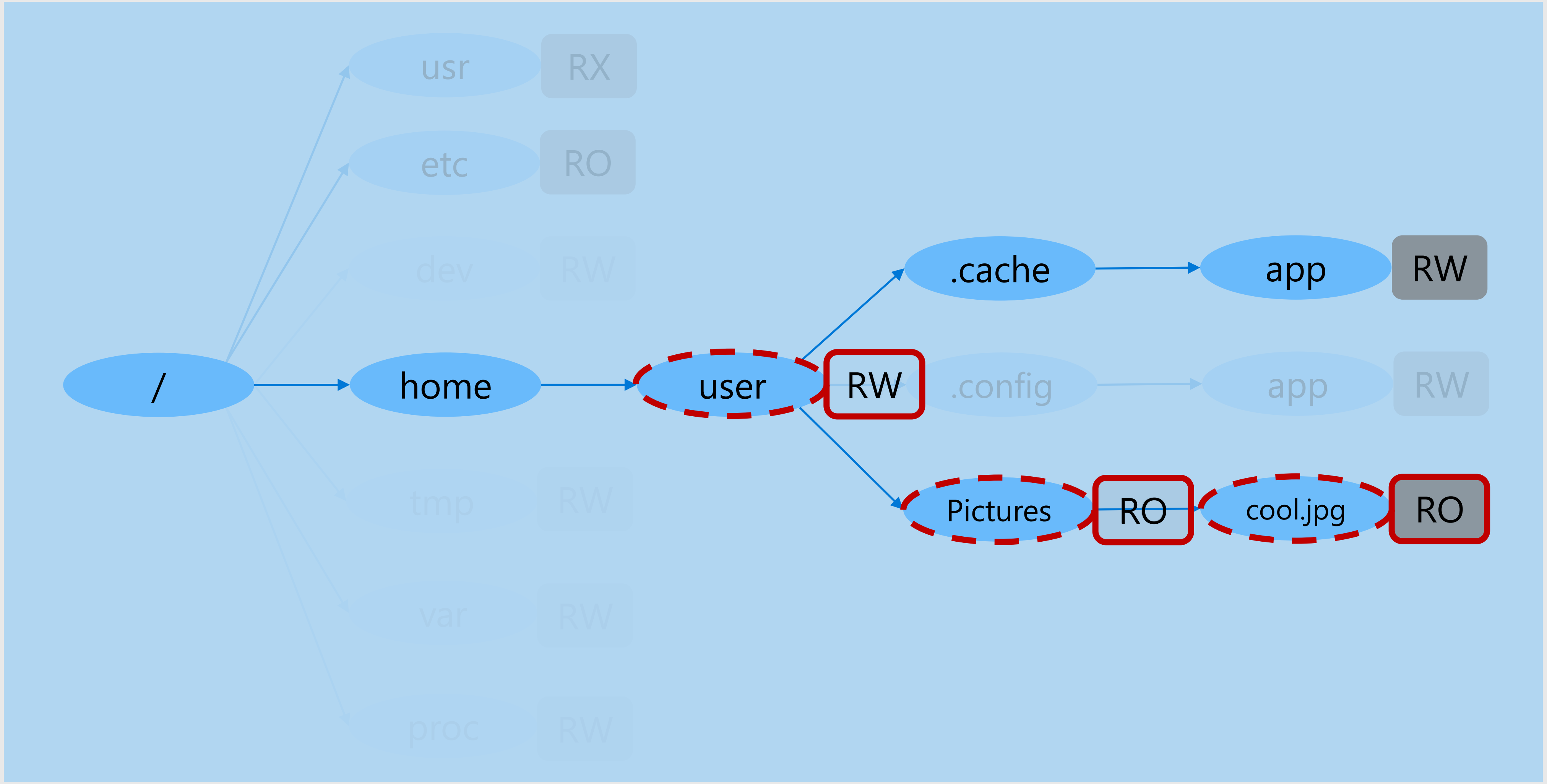
- Access rights are tied to inodes by user space thanks to opened file descriptors and a new system call: *landlock_add_rule(ruleset_fd, rule_type, rule_attr, flags)*
- All access rights for the same inode are stored in-line in a dedicated kernel struct (i.e., tag) including a flexible array.
- Lifetime of tags depends on associated sandbox domain lifetimes and underlying superblock lifetimes thanks to a new LSM hook: *security_sb_delete(super_block)*

File hierarchy check

- When requesting access to a file, walk through all parent files until all domains have been checked (or the root is reached)

Sandbox policies composition

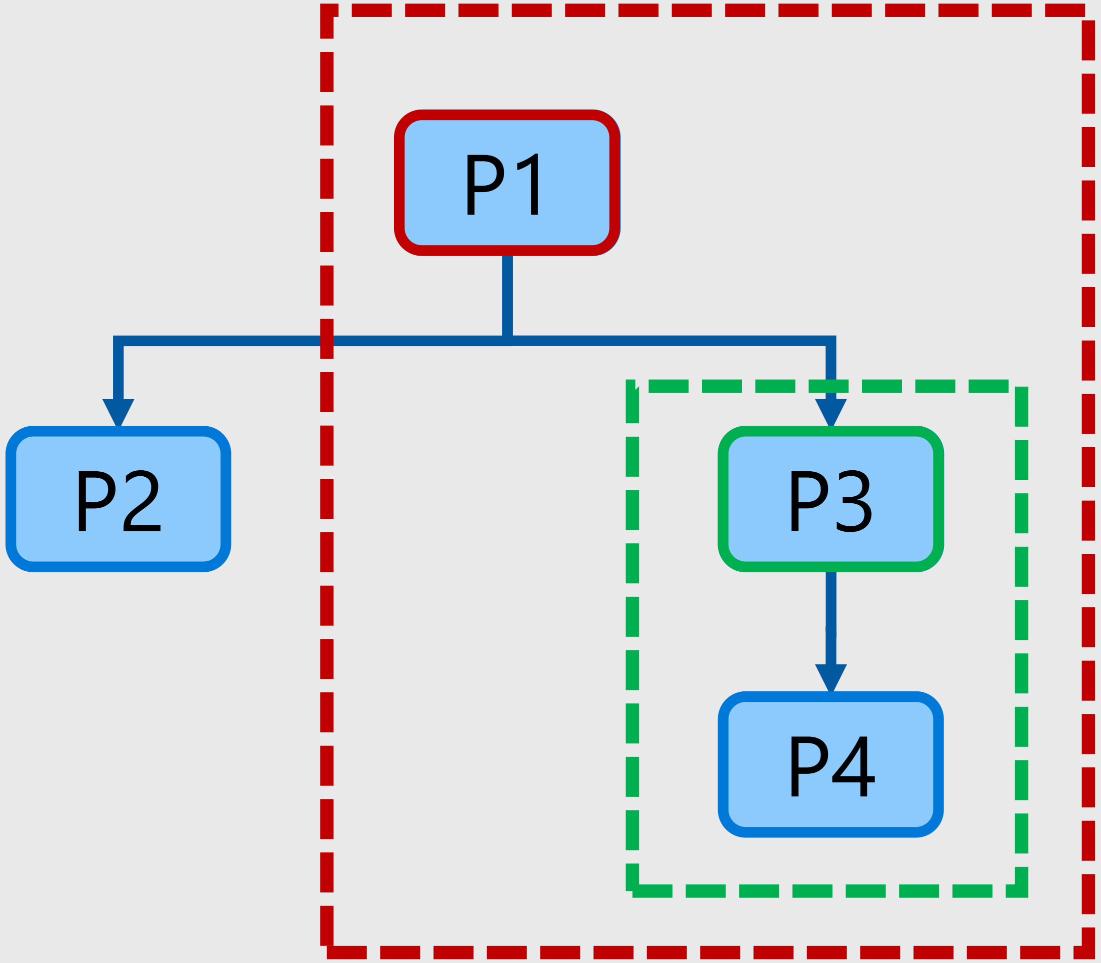
Filesystem policy example

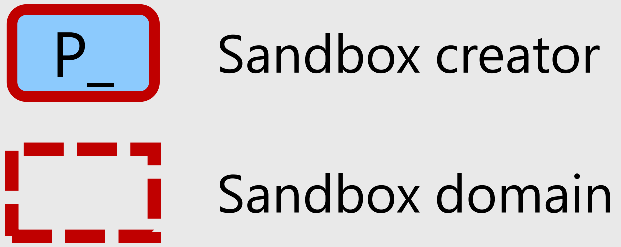


3rd layer ✓
2nd layer ✓
1st layer ✓

Sandbox policies composition

Policies hierarchy



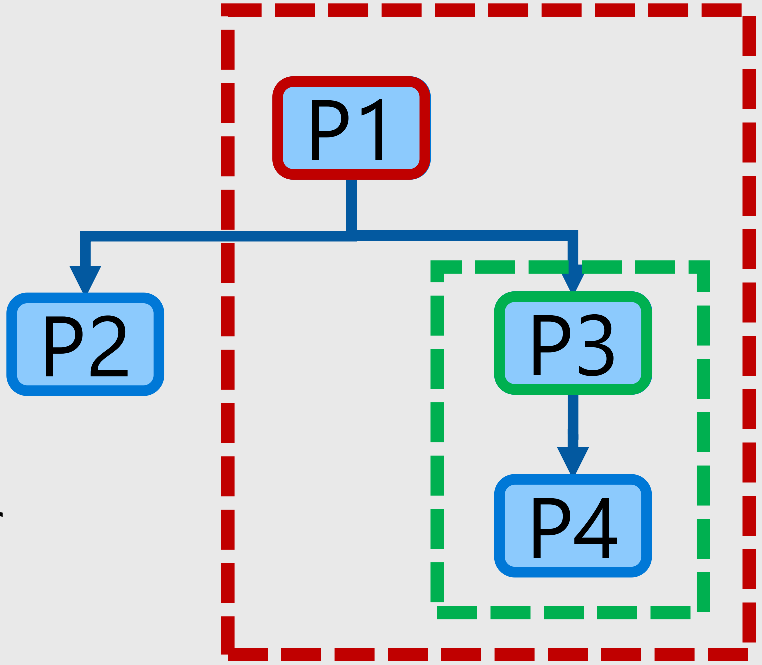


Sandbox policies composition

ptrace restrictions

Forbid access to parent or sibling sandboxes

Introspection (i.e., ptrace) of processes not in in a child sandbox (or the same sandbox) is forbidden.



s(P): sandbox domain of P
X ⊃ Y: only X can ptrace Y
X = Y: X and Y can ptrace each other

s(P2) ⊃ s(P1) ⊃ s(P3) = s(P4)

User space testing

Made the *kselftest-harness* framework available to other users.

2600+ single lines of test code to reached more than 93% of coverage (close to the top limit).

Kernel fuzzing with syzkaller

Added Landlock system calls

Extended some specific system calls

Added tests to help it (dis)cover kernel code

Reached 72% of coverage (close to the top limit for this code)

Checked that it can find bugs!

Minimum Viable Product

Filesystem limitations to avoid policy bypass

- File reparenting: renaming or linking a file to a different parent directory is always denied.
- Filesystem topology modification: arbitrary mounts.

Design limitations

Unprivileged access-control cannot restrict anything (e.g., more privileged processes, kernel): hierarchy of sandboxes.

Current LSM hooks need to be updated to bring more access-control types to Landlock: inode hooks vs. path hooks.

seccomp-bpf can help to complete a sandbox.

Kernel-side roadmap

Short term

- Improve kernel performance for the current features.
- Add the ability to change the parent directory of files (see current Landlock limitations).

Medium term

- Add audit features to ease debugging.
- Extend filesystem access-control types to address the current limitations.
- Add the ability to follow a deny listing approach, which is required for some use cases.

Long term

- Add minimal network access-control types.
- Add the ability to create (file descriptor) capabilities compatible with Capsicum.

Wrap-up

Landlock is designed to be inclusive and safe to use: any process should be able to use it to protect user data, considering some implementation constraints.

It is a standalone minimal but extensible interface to create sandboxes.

Questions: landlock@lists.linux.dev

Resources: https://landlock.io