Microsoft

Sandboxing your application with Landlock, illustration with the p7zip case

Pass the Salt

Mickaël Salaün

2022-07-04

Sandboxing your application

Landlock is available in mainline since 2021 (Linux 5.13), but with some limitations due to the iterative approach.

Landlock is now enabled by default on multiple distros: Ubuntu 22.04 LTS, Fedora 35, Arch Linux, Alpine Linux, Gentoo

This talk is about the steps to sandbox an application, illustrated with p7zip.

Developers and users

It is assumed that with enough skills and time, most applications could be compromised.

Problem (as developers):

- We don’t want to participate to malicious actions through our software because of security bug exploitation.
- We have a responsibility for users, especially to protect their (personal) data: every **new app increases** (user) **attack surface**.

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.

Sandbox properties:

- Follow the least privilege principle
- Innocuous and composable security policies

What is Landlock?

Landlock is an access control system available to **unprivileged** processes on Linux, thanks to 3 dedicated syscalls.

It enables developers to add **built-in** application **sandboxing**.

Filesystem access-control

Filesystem properties

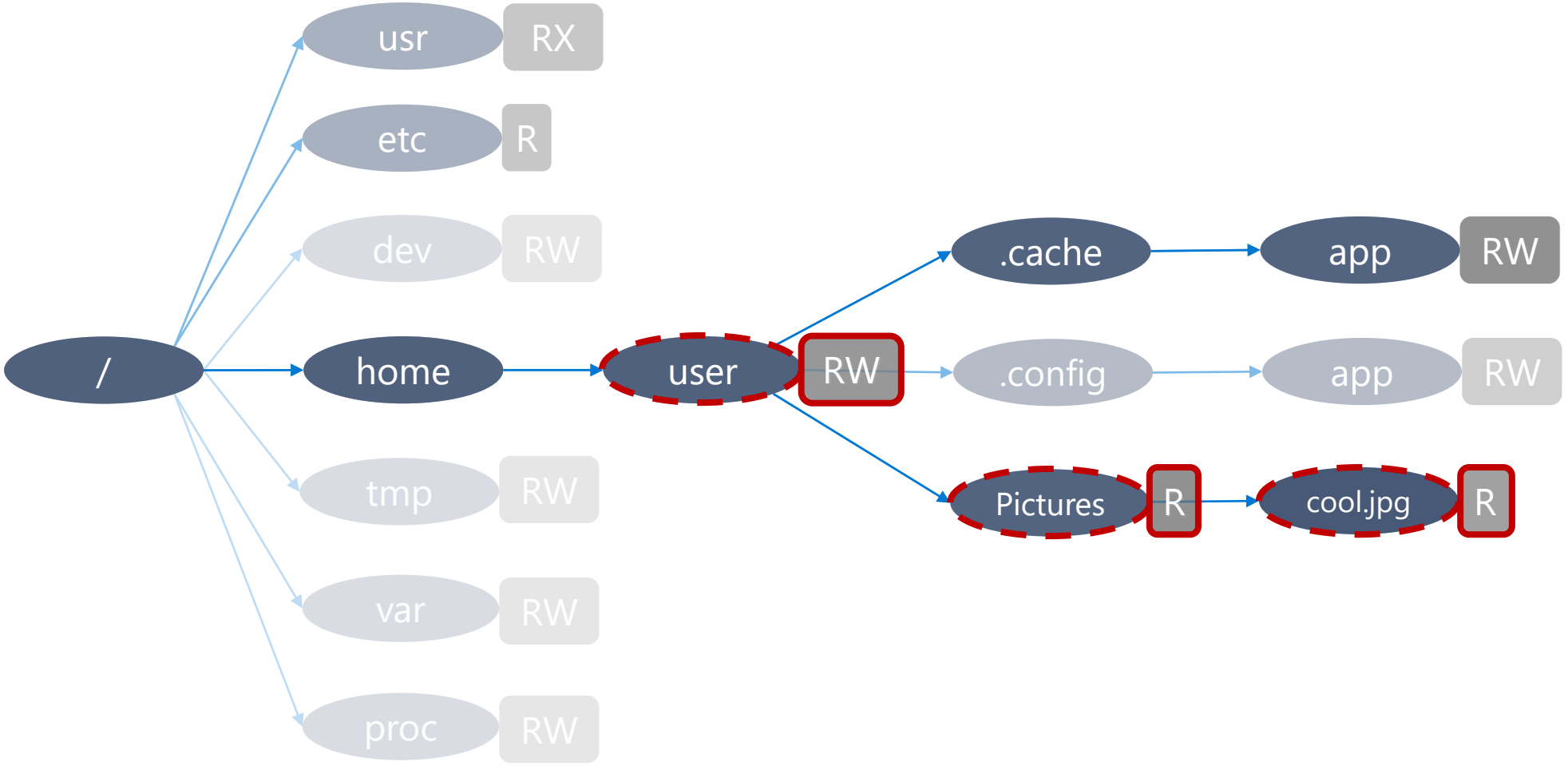
Access-control rights:

- Execute, read or write to a file
- List a directory or remove files
- Create files according to their type
- Rename or link files

File hierarchy identification: ephemeral inode tagging

Example of filesystem policy composition

3rd layer ✔
2nd layer ✔
1st layer ✔



R Read
W Write
X eXecute

Implementing sandboxing

How to patch an application?

1. Define the threat model: which data is trusted or untrusted?
2. Identify the complex parts of the code: where there is a good chance to find bugs?
3. Identify and patch the configuration handling to infer a security policy.
4. Identify and patch the most generic places to enforce the security policy for the rest of the lifetime of the thread.

Application compatibility

Forward compatibility for applications is handled by the kernel development process.

Backward compatibility for applications is the responsibility of their developers.

Each new Landlock feature increments the ABI version, which is useful to leverage available features in a **best-effort security** approach.

Step 1: Check the Landlock ABI

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

if (abi < 0)
    return 0;
```

Step 2: Create a ruleset

```
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 3: Add rules

```
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 4: 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);
```

Full example: https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/samples/landlock/sandboxer.c

Demo with p7zip

Sandboxing PR: https://github.com/jinfeihan57/p7zip/pull/184

And now, let's say there is a bug in the gzip parser…

What’s next?

Roadmap

Short term:

- Add audit features to ease debugging
- New minimal access-control types:
  - Network
  - Process signaling
- Improve kernel performance

Roadmap

Medium term:

- Extend access-control types to address the current limitations:
  - Filesystem
  - Network
- Add the ability to follow a deny listing approach

Contribute

- Develop new (kernel) features (e.g., new access types)
- Write new tests (e.g., kunit)
- Challenge the implementation
- Improve documentation
- **Sandbox your applications** and others'

Questions?

https://docs.kernel.org/userspace-api/landlock.html

Past talks: https://landlock.io

landlock@lists.linux.dev

Thank you!