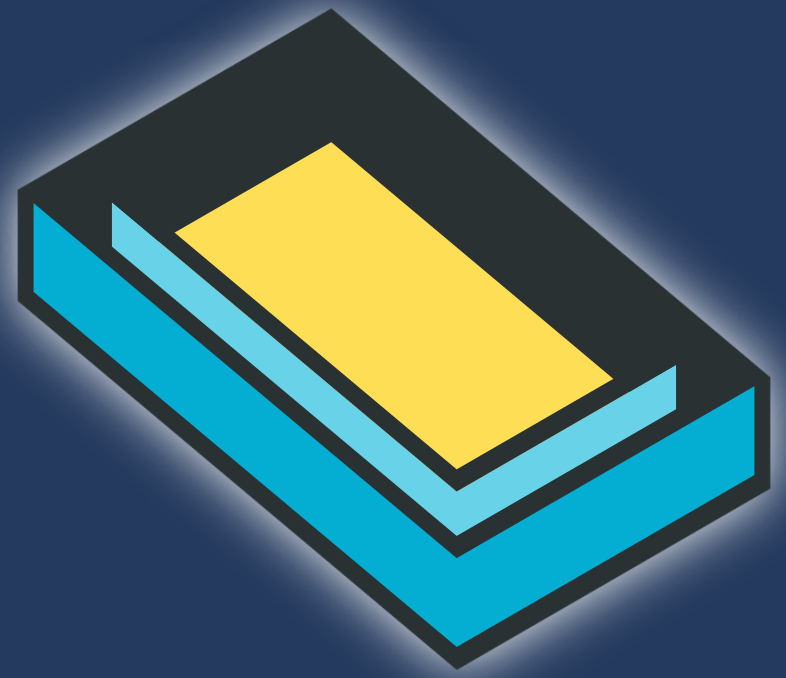
Microsoft

Linux sandboxing with Landlock



Open Source Summit Europe

Mickaël Salaün – kernel maintainer

2024-09-17

Once upon a time… not so far away

A company developing a product used all over the world finally gets a worldwide coverage in the news, unfortunately not a good one:

Millions of machines compromised because of this software!

You are a developer, what can you do to prevent such malicious exploitation?

How could this happen?

bugs

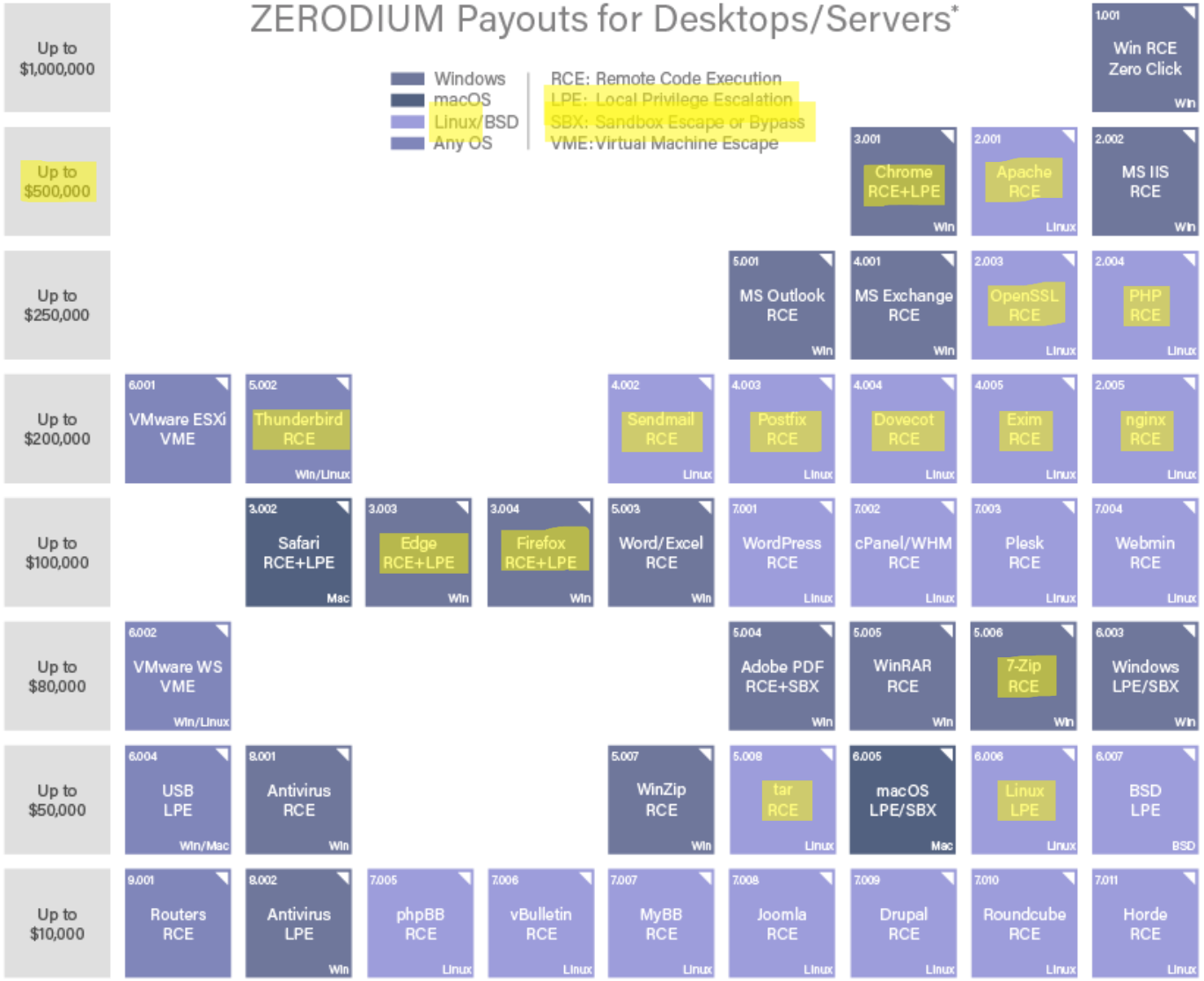
+

malicious actors

=

exploited vulnerabilities

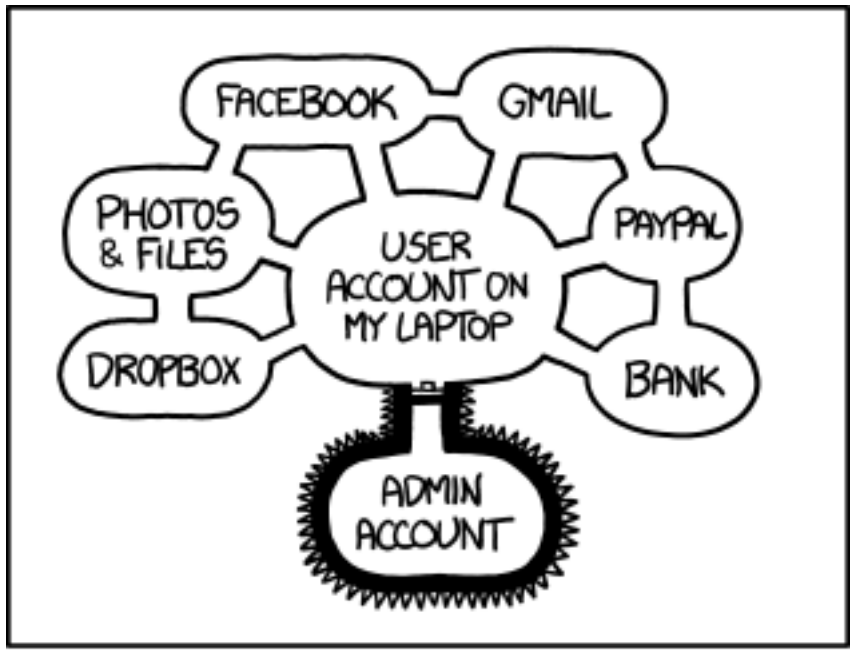
Real-life exploits



What could we do about this?

- Remove all bugs?
- Use another programming language?
- Test everything?
- Encrypt data?
- Limit access of our product!

Protect data



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

Virtual machine

Pros

- Duplicate the whole system and then mitigates its exposure

Cons

- Shipping a VM instead of an installer is a hard sell because of size, overhead and complexity
- Does not provide an access control system

Container

Pros

- Well known developer tool
- Lighter than a VM

Cons

- May increase the attack surface and comes with its own vulnerabilities: namespaces and embed dependencies
- May provide some coarse-grained control for file access, but not native to apps/services: increased configuration

AppArmor, SELinux, Smack, or Tomoyo

Pros

- Real access control systems

Cons

- Security policy is system-wide and cannot be embedded in apps/services: complex and static configuration

BPF LSM

Pros

- Dynamic security policies

Cons

- Security policy is system-wide and cannot be embedded in apps/services: complex eBPF programs
- Difficult to deal with files

seccomp-bpf

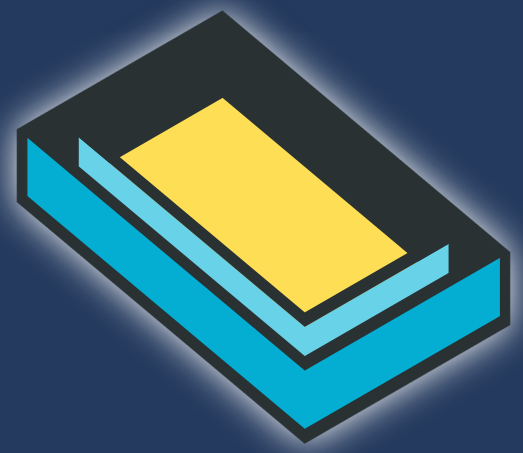
Pros

- Dynamic security policies
- Reduces the kernel attack surface
- Embeddable in apps/services: unprivileged

Cons

- Not an access control system: cannot identify files nor other kernel semantic
- Fixed set of syscalls: update issues
- Scoped to a set of processes

Landlock



Pros

- Real access control system
- Dynamic security policies
- Embeddable in apps/services: unprivileged

Cons

- Scoped to a set of processes

Sandboxing

What is sandboxing?

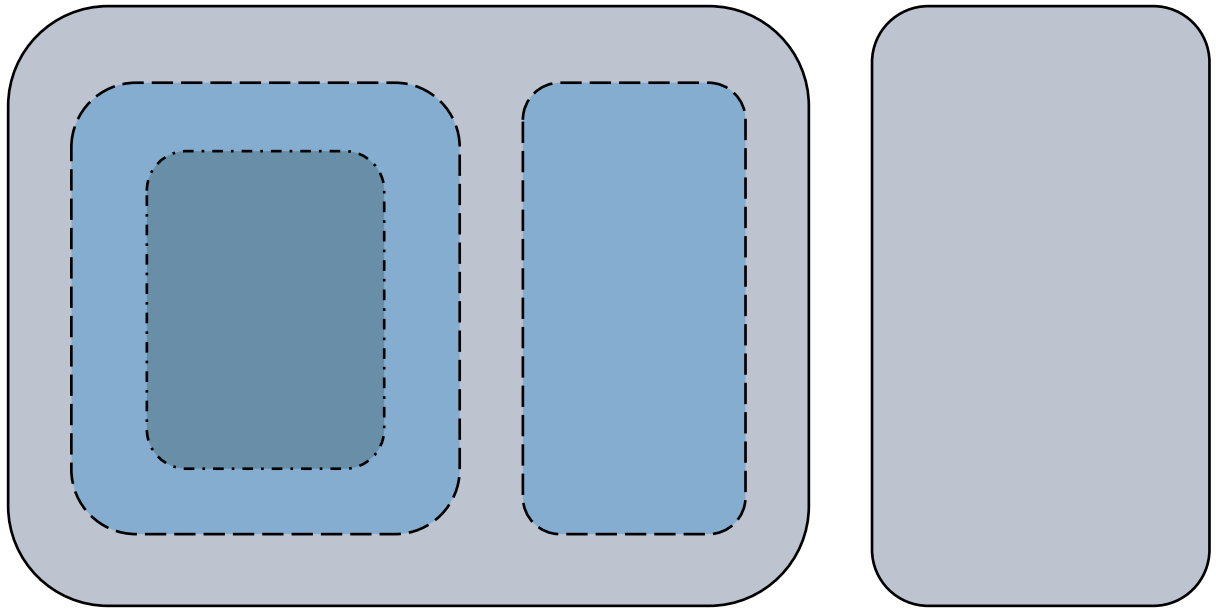
"A **restricted**, controlled **execution environment** that prevents potentially malicious software [...] from accessing any system resources except those for which the software is authorized."

Tailored and embedded security policy

Developers are in the best position to reason about the required **accesses** according to **legitimate** behaviors:

- Application semantics
- Static and dynamic configuration
- Interactions

Dynamic policy composition



Safe security mechanism

Principle of least privilege

- No privileged accounts or services
- No SUID binaries

Innocuous access control

- Only increase restrictions

Protecting against bypasses

- Each process should be protected from less-privileged ones

Non-Linux systems

Main sandbox mechanisms:

- XNU Sandbox (iOS)
- Pledge and Unveil (OpenBSD)
- Capsicum (FreeBSD)
- AppContainer (Windows)

Landlock properties

Use case #1

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

Candidates:

- Container runtimes
- Init systems

Use case #2

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

Useful development properties

Embedded policies: **testable with a CI** and always synchronized with **app semantic**

Set of small policies: **easier to maintain** and audit

Composable policies: **lockless concurrent policy development**

Well-defined backward compatibility with ABI versions: stable and **consistent results**

How Landlock works?

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.

Current access control

Implicit restrictions

- Process impersonation (e.g., ptrace)
- Filesystem topology changes (e.g., mounts)

Explicit access rights

- Filesystem
- Networking

Current filesystem access rights

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

Current networking access rights

- Connect to a TCP port
- Bind to a TCP port

Upcoming IPC scoping

Scope sandboxes:

- Connect to abstract UNIX sockets
- Send signals

Landlock interface

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

```
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

Adoption

Linux distributions

Most distros support Landlock by default:

- Arch Linux
- Ubuntu
- Debian
- Fedora
- chromeOS
- WSL2
- Azure Linux
- Gentoo
- Flatcar…

Landlock helpers

Examples of sandbox tools:

- setpriv
- Firejail
- Minijail

Examples of sandbox libraries:

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

Landlocked apps

Examples of 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)

Getting noticed by attackers too!

Landlock support in XZ Utils:

- 5.6.0 (2024-02-24) ✔
- 5.6.1 (2024-03-09) ✘
- 5.6.2 (2024-05-29) ✔

✓ **CMake: Fix sabotaged Landlock sandbox check.**

```
It never enabled it.
```

master

Larhzu committed on Mar 30

Showing **1 changed file** with **1 addition** and **1 deletion**.

2 CMakeLists.txt

```
@@ -1001,7 +1001,7 @@ if(NOT SANDBOX_FOUND AND ENABLE_SANDBOX MATCHES
1001 1001          #include <linux/landlock.h>
1002 1002          #include <sys/syscall.h>
1003 1003          #include <sys/prctl.h>
1004      -        .
```

Try Landlock

```
# WARNING: The "sandboxer" is a demonstration program,
# not a tool with a stable interface.

$ cargo install landlock --examples

$ sandboxer
```

Wrap-up

Roadmap

Ongoing and next steps:

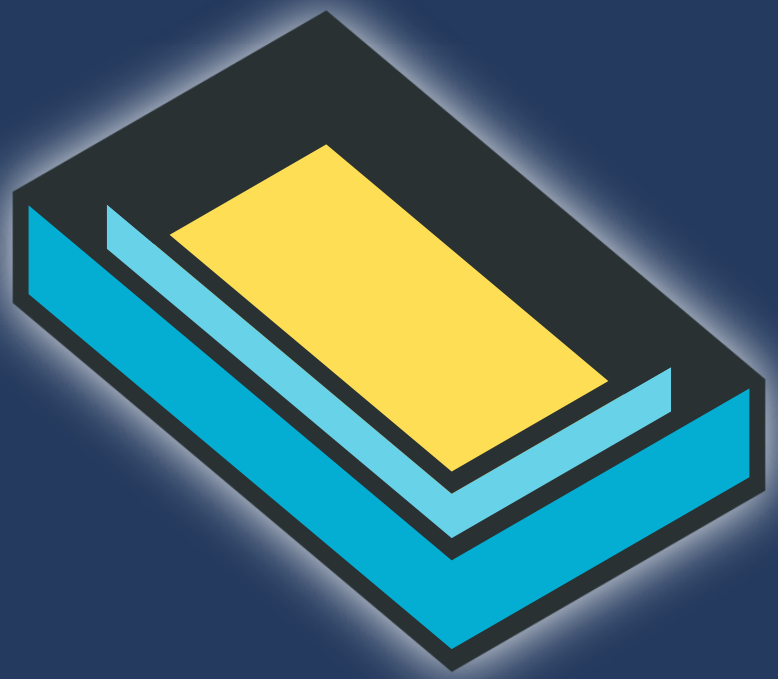
- Add new access-control types: socket creation, UDP port use…
- Add audit support to ease debugging and provide metrics
- Develop a new sandboxer tool
- Improve adoption

Contribute

Kernel contributors: Günther Noack, Konstantin Meskhidze, Jeff Xu, Ivanov Mikhail, Jann Horn, Tahera Fahimi…

Contribution ideas:

- Develop new access types and tests
- Improve libraries: Rust, Go…
- Improve documentation
- Challenge implementations



Questions?

landlock@lists.linux.dev

Thank you!