Microsoft

Landlock workshop: Linux sandboxing in practice

Pass the Salt

Mickaël Salaün

2024-07-03

CC BY SA

Sandboxing with Landlock

Landlock is available in mainline since 2021 (Linux 5.13) and gaining new features over time.

Landlock is now enabled by default on multiple distros: Ubuntu, Fedora, chromeOS, Azure Linux, WSL2…

This workshop is about sandboxing ImageMagick: use an old and vulnerable version to illustrate how **sandboxing can mitigate vulnerabilities**.

VM setup

See https://github.com/landlock-lsm/workshop-imagemagick

If you already cloned the repository:

```
git pull
vagrant up
vagrant ssh
```

Connect to the VM

```
# Once set up, take a snapshot and log in

vagrant snapshot push
vagrant ssh


# We can now also use virt-manager to connect to the VM
```

Steps done by the VM provisioning

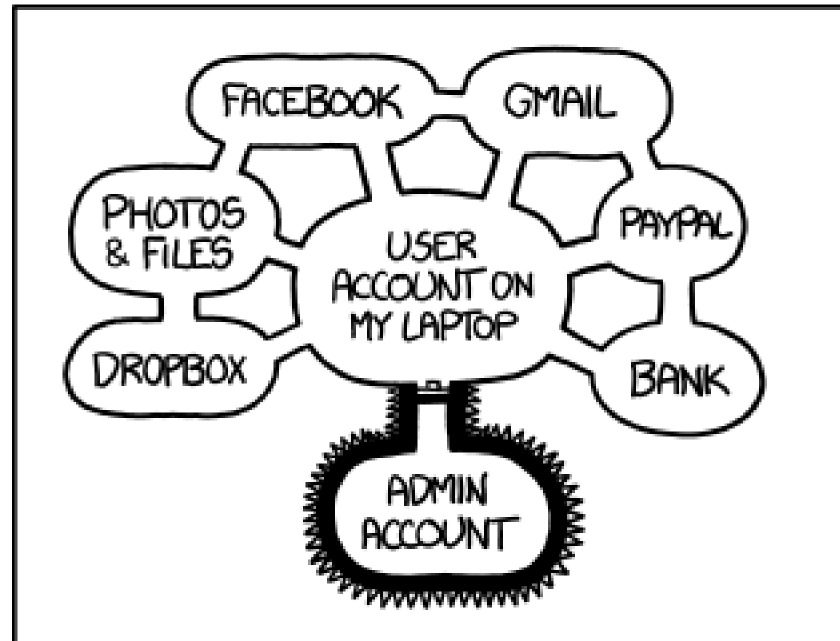
1. Set up the build environment
2. Build a vulnerable version of ImageMagick

Agenda

1. Problem statement
2. Securing developments
3. Security sandboxing
4. Landlock properties
5. Filesystem access control
6. Sandboxing with Landlock
7. Let’s patch ImageMagick!
8. Compatibility

Problem statement

Goal: protect data

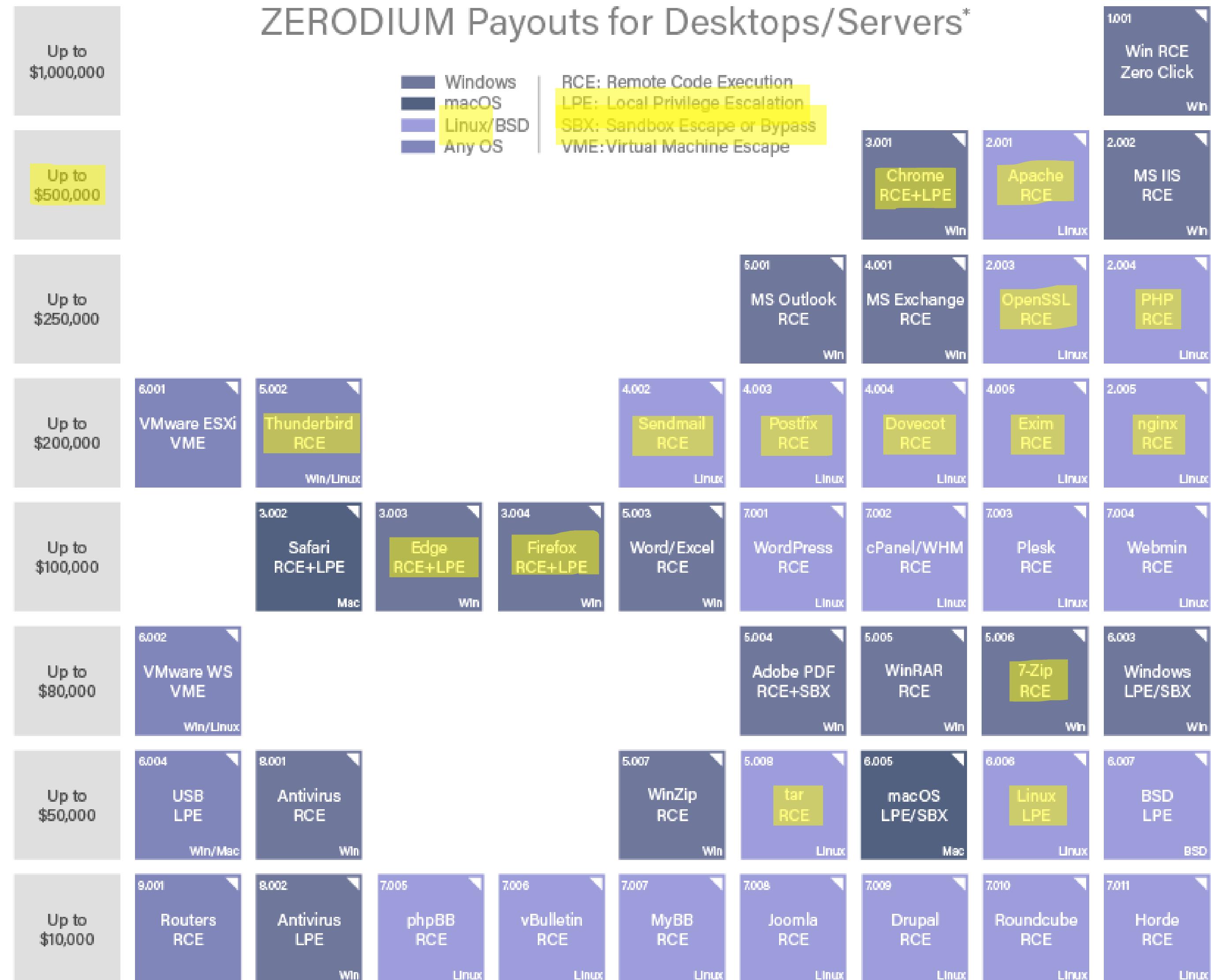


https://xkcd.com/1200

Pragmatic statements

1. An innocuous and trusted process can **become malicious during its lifetime** because of bugs exploited by attackers.
2. There are multiple and **different levels of trust** and different **consequences** in case of a breach: system, user, app data…

Attack cost



Consequence of a breach

There are multiple and **different levels of trust** and different **consequences** in case of a breach: system, user, or app data.

We want to protect each level of this TCB as much as possible.

Developers' responsibility to protect users' (personal) data. Let's improve security!

Securing developments

How to protect an application?

Reactive solutions

Fix bugs quickly and push updates widely

How to protect an application?

Proactive solutions

- Look for bugs (e.g., audit, fuzzing) and fix them
- Add more tests and use them
- Use safer languages and libraries
- Leverage linters, compilers and other tools
- Consider (most) software as potentially malicious and **protect the rest of the system** from them

Security 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

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)

Candidates for a sandboxing mechanism

	Performance	Fine-grained control	Embedded policy	Unprivileged use
Virtual Machine	✗	✗	✗	✗
SELinux	✓	✓	✗	✗
namespaces	✓	✗	✓	!
seccomp	✓	✗	✓	✓
Landlock	✓	✓	✓	✓

- ✓ Yes, compared to others
- ✗ No, compared to others
- ! In some way, but with limitations

Landlock properties

Use case #1

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

Candidates:

- Container runtimes
- Init systems

Use case #2

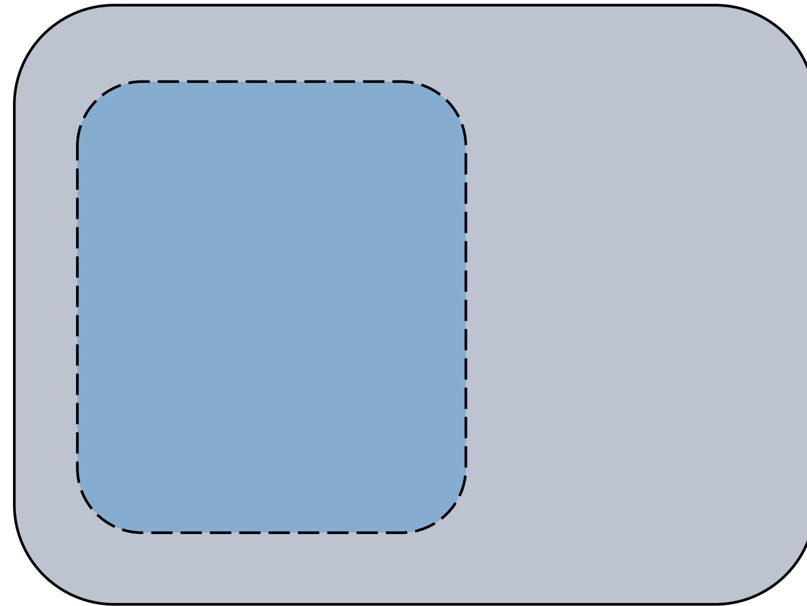
Exploitable bugs in trusted applications: protect from vulnerable code maintained by developers.

Candidates:

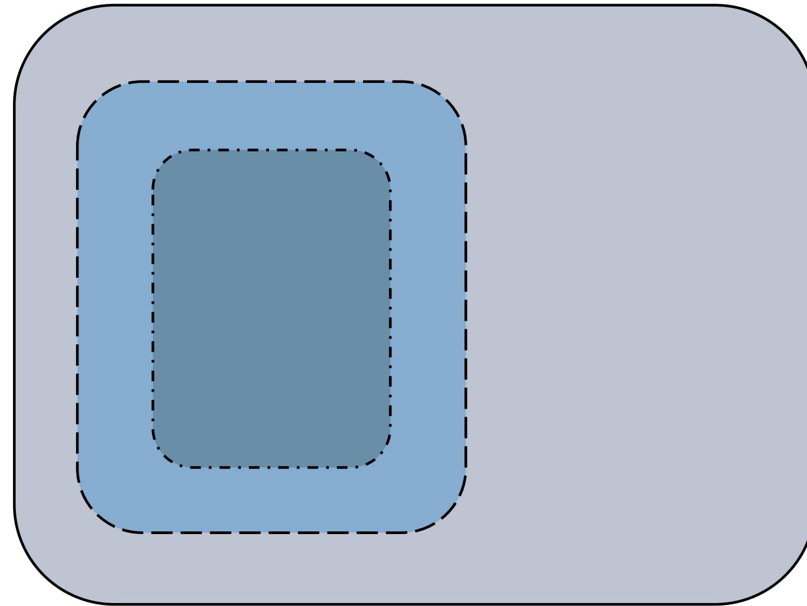
- Parsers: archive tools, file format conversion, renderers…
- Web browsers
- Network and system services

Dynamic policy composition

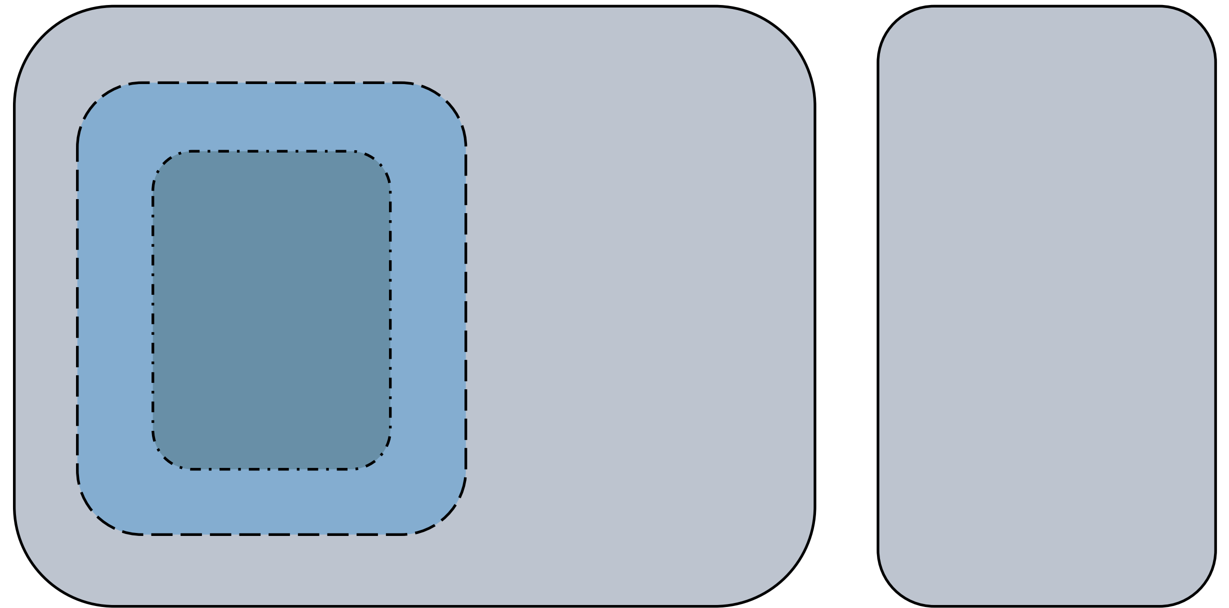
Dynamic policy composition



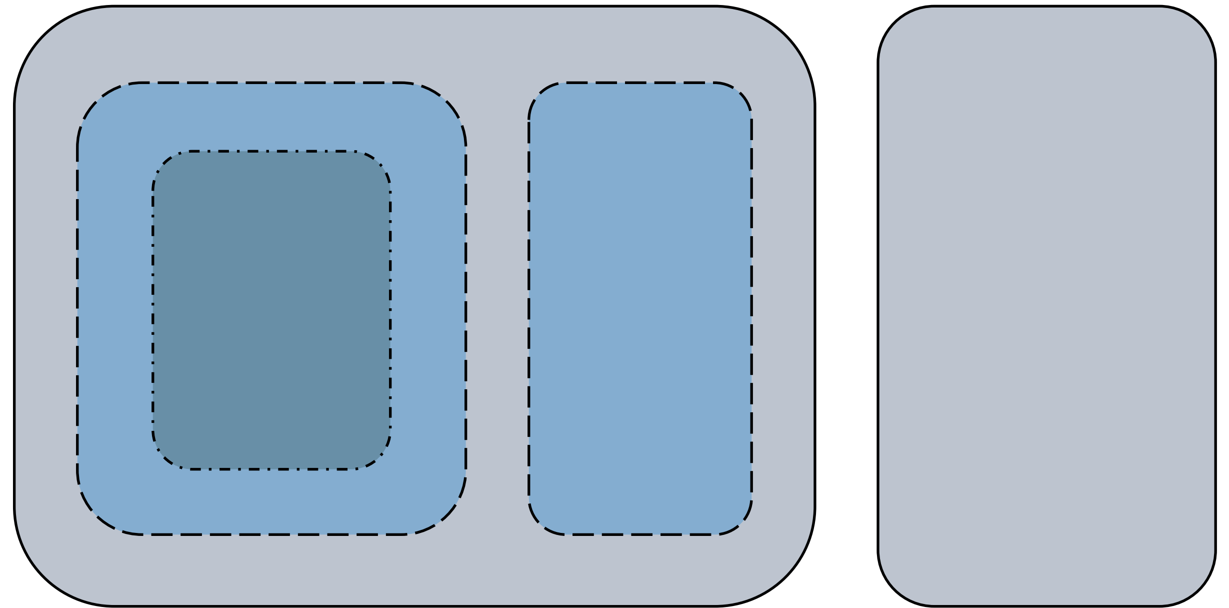
Dynamic policy composition



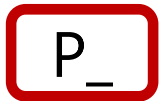
Dynamic policy composition



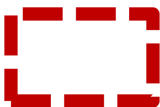
Dynamic policy composition



Sandbox policies hierarchy

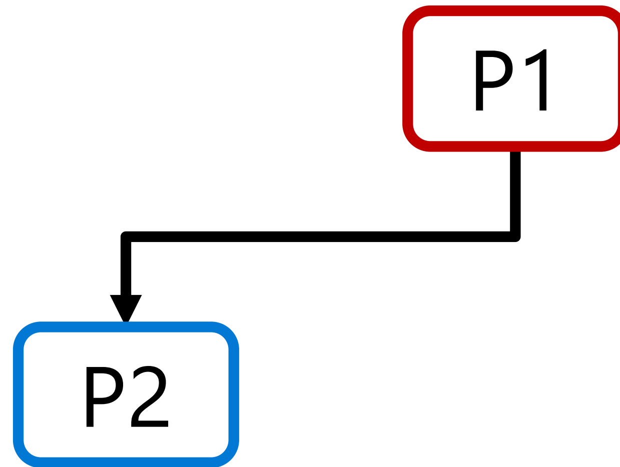


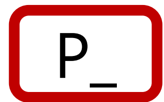
Sandboxed process



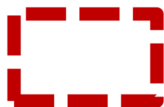
Sandbox domain

Sandbox policies hierarchy



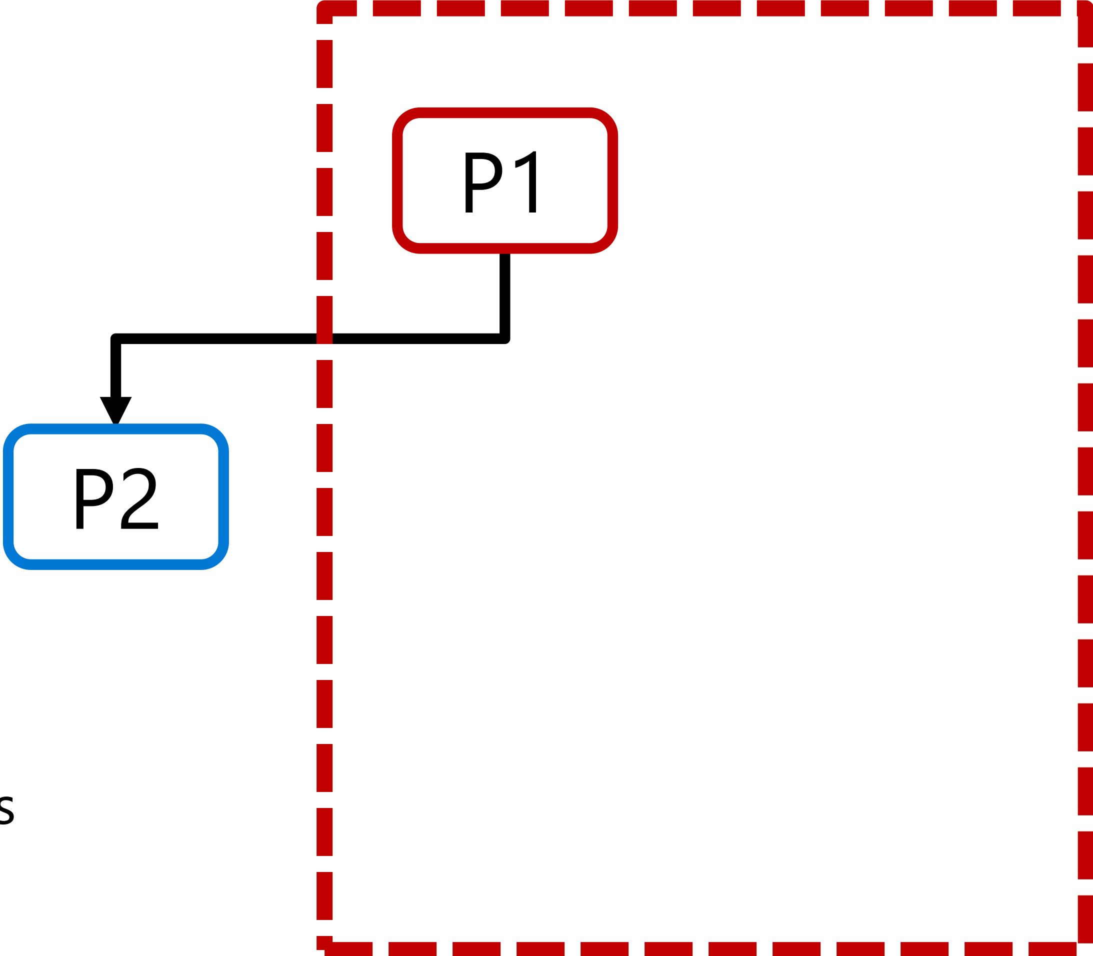


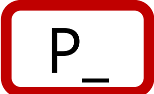
Sandboxed process



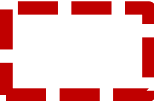
Sandbox domain

Sandbox policies hierarchy



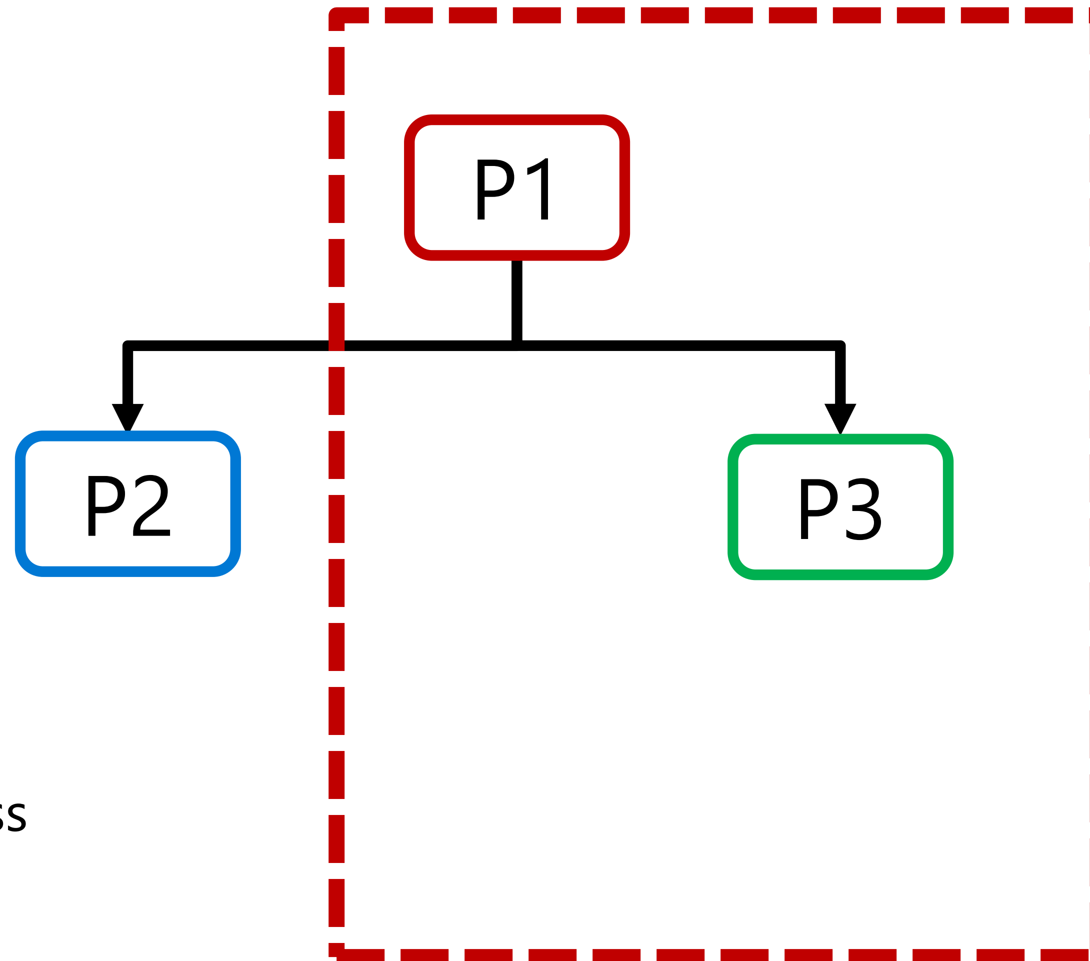


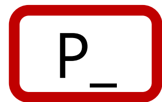
Sandboxed process



Sandbox domain

Sandbox policies hierarchy

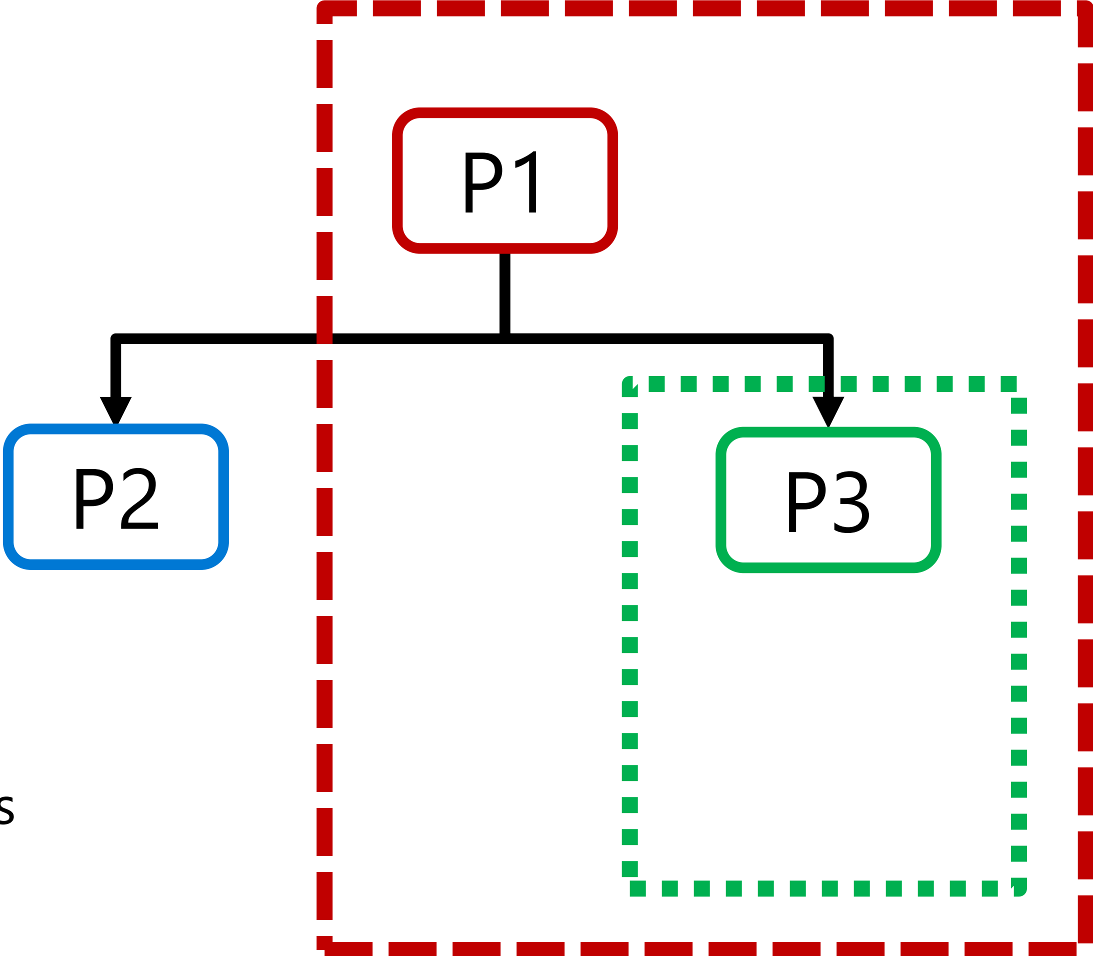


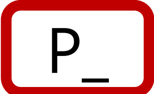


Sandboxed process

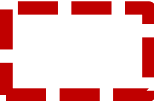
Sandbox domain

Sandbox policies hierarchy



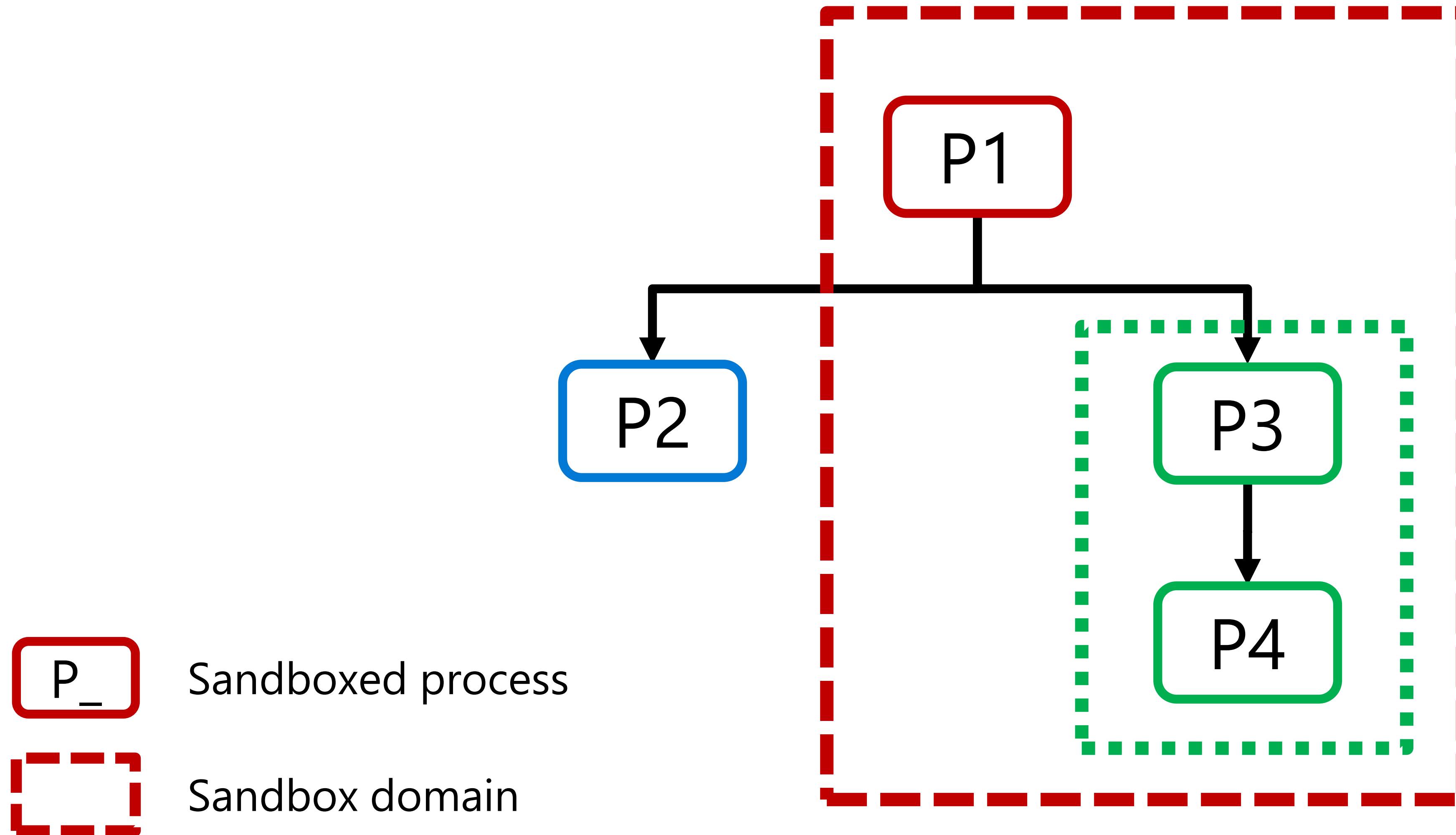


Sandboxed process



Sandbox domain

Sandbox policies hierarchy



Landlock empowers developers

New unprivileged **security layers**

Lockless concurrent development: avoid policy management bottleneck

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

Testable with a **CI** and synchronized with app **semantic**: stable

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

<u>Implicit restrictions</u>

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

<u>Explicit access rights</u>

- Filesystem
- Networking

Filesystem access control

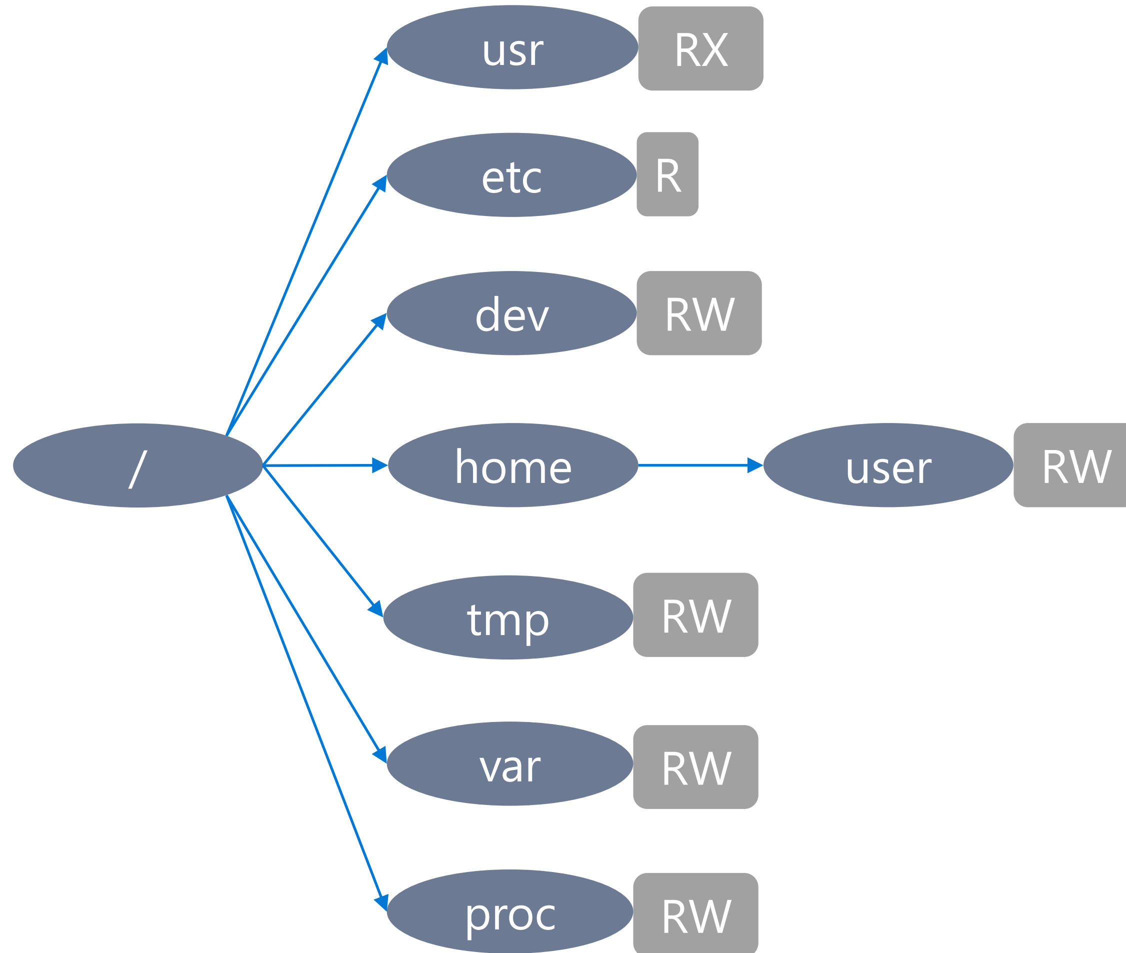
Filesystem access rights

- Execute, read or write to a file
- List a directory or remove files
- Create files according to their type
- Rename or link files
- Send IOCTL commands to device files (with Linux 6.10)

Example of filesystem policy composition

Example of filesystem policy composition

1st layer



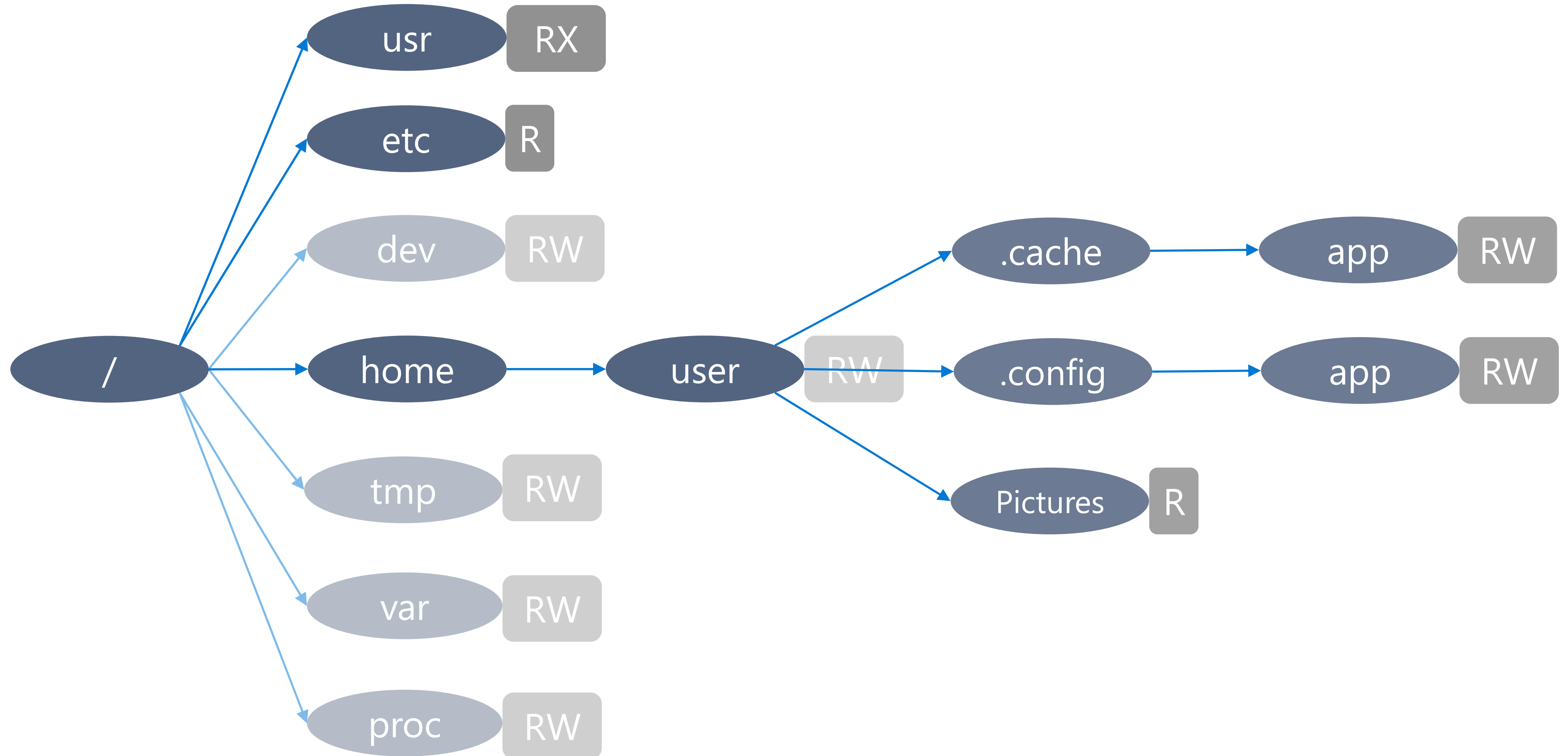
R Read

W Write

X eXecute

Example of filesystem policy composition

2nd layer



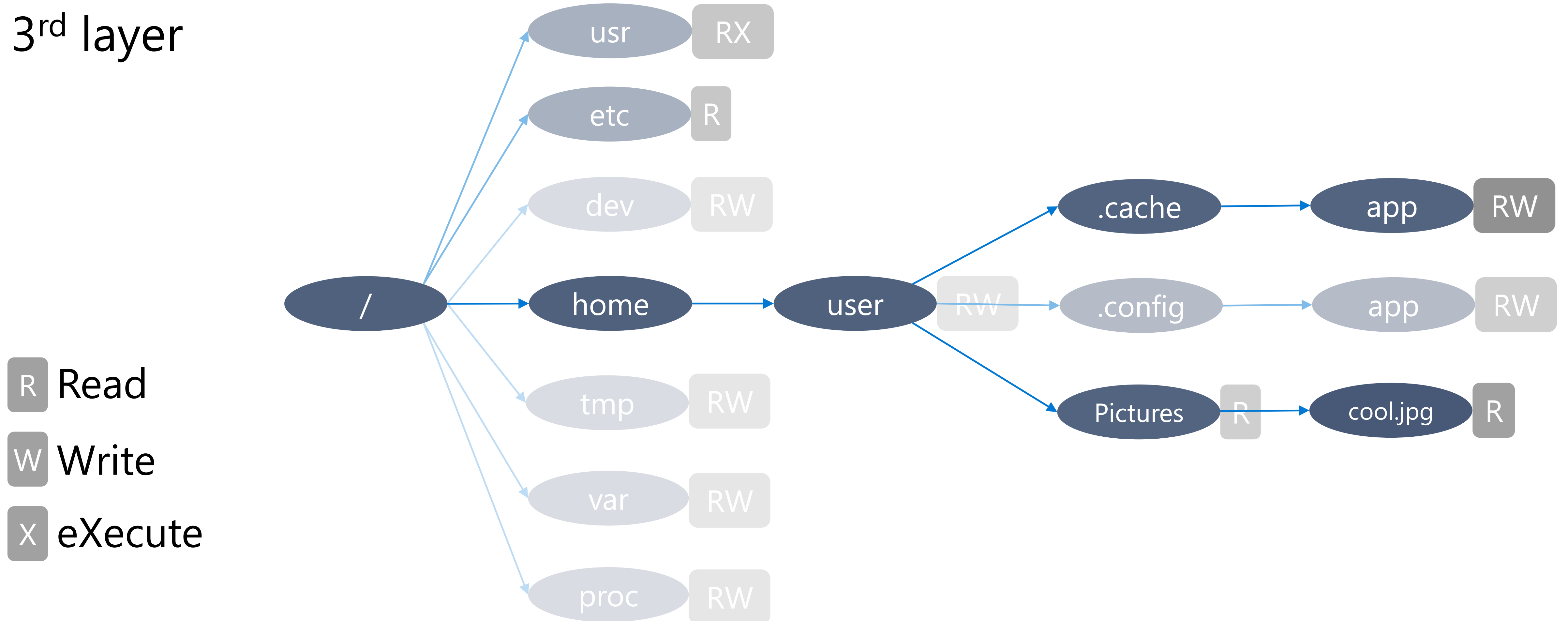
R Read

W Write

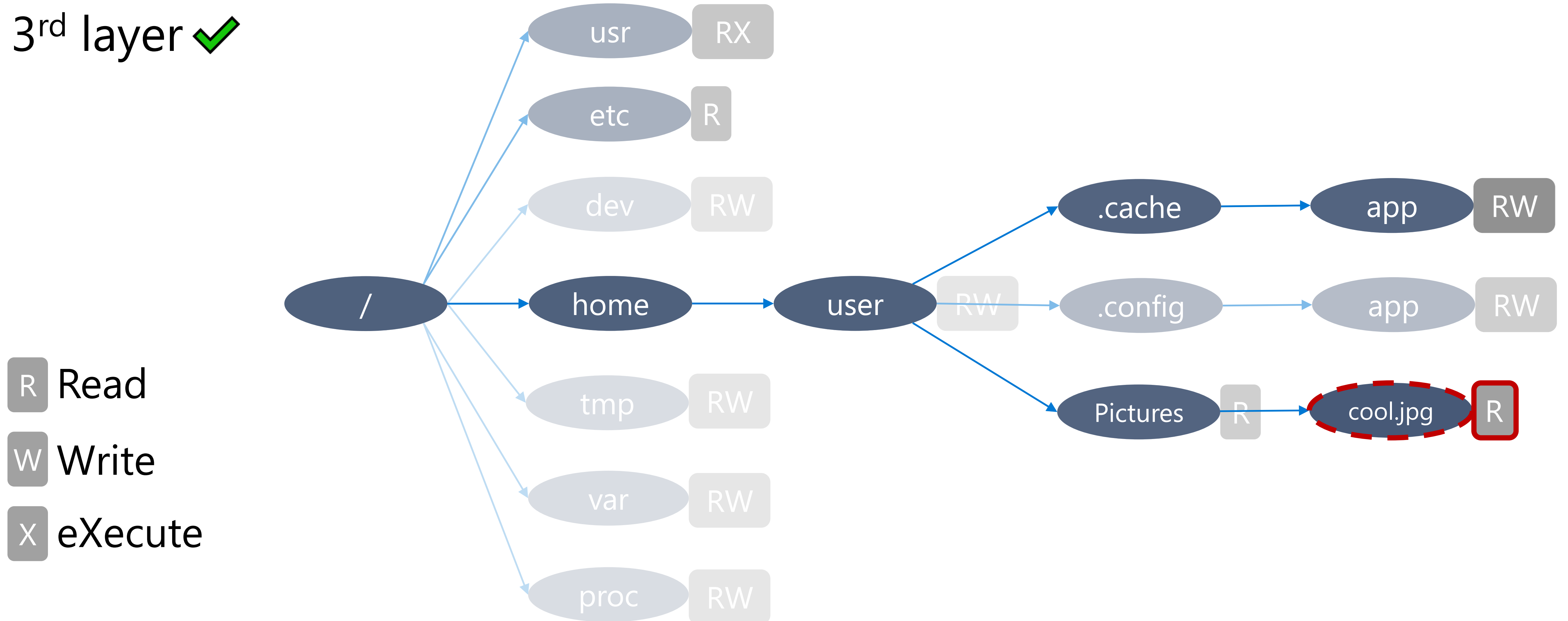
X eXecute

Example of filesystem policy composition

3rd layer

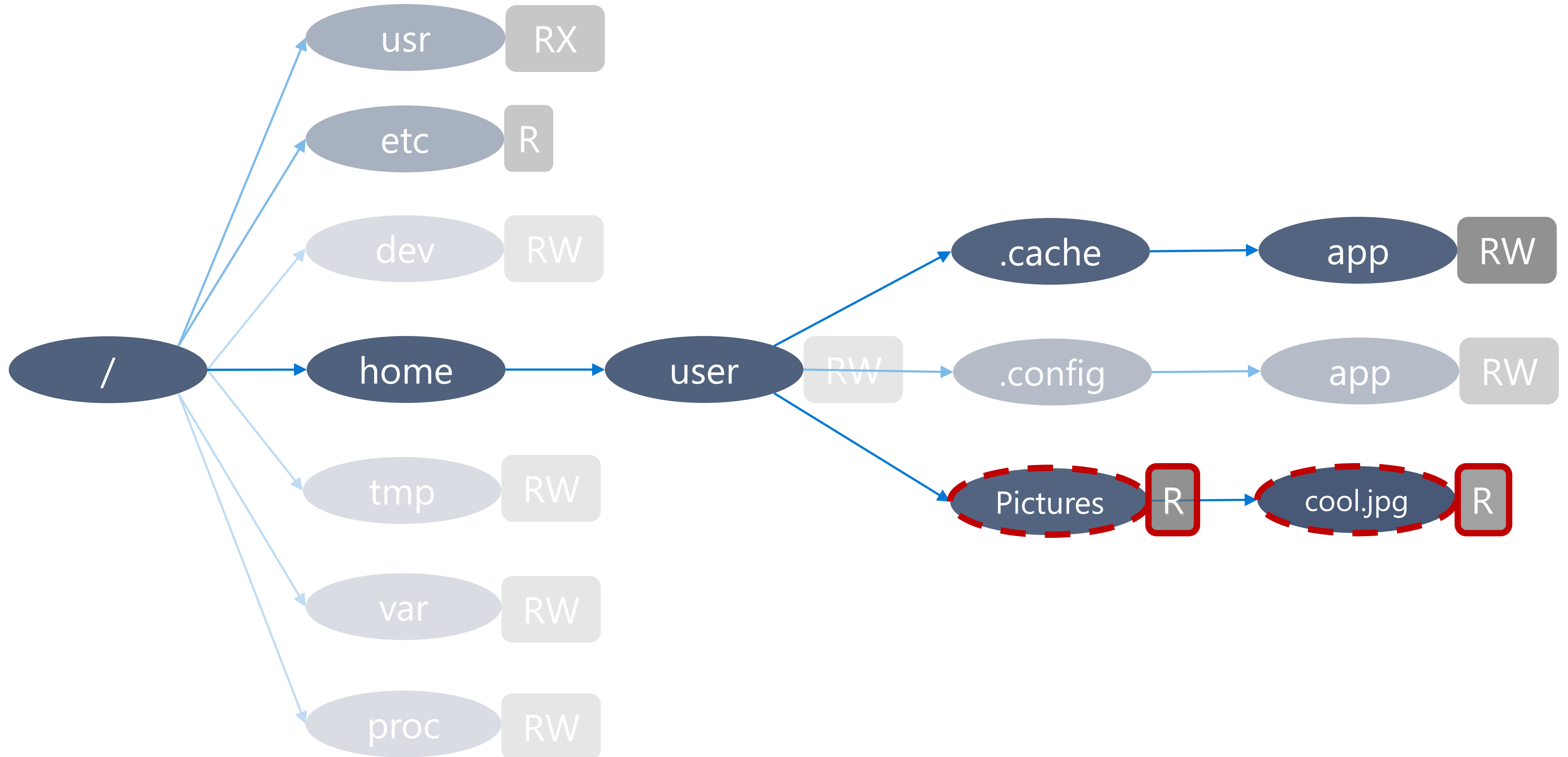


Example of filesystem policy composition



Example of filesystem policy composition

3rd layer ✓
2nd layer ✓

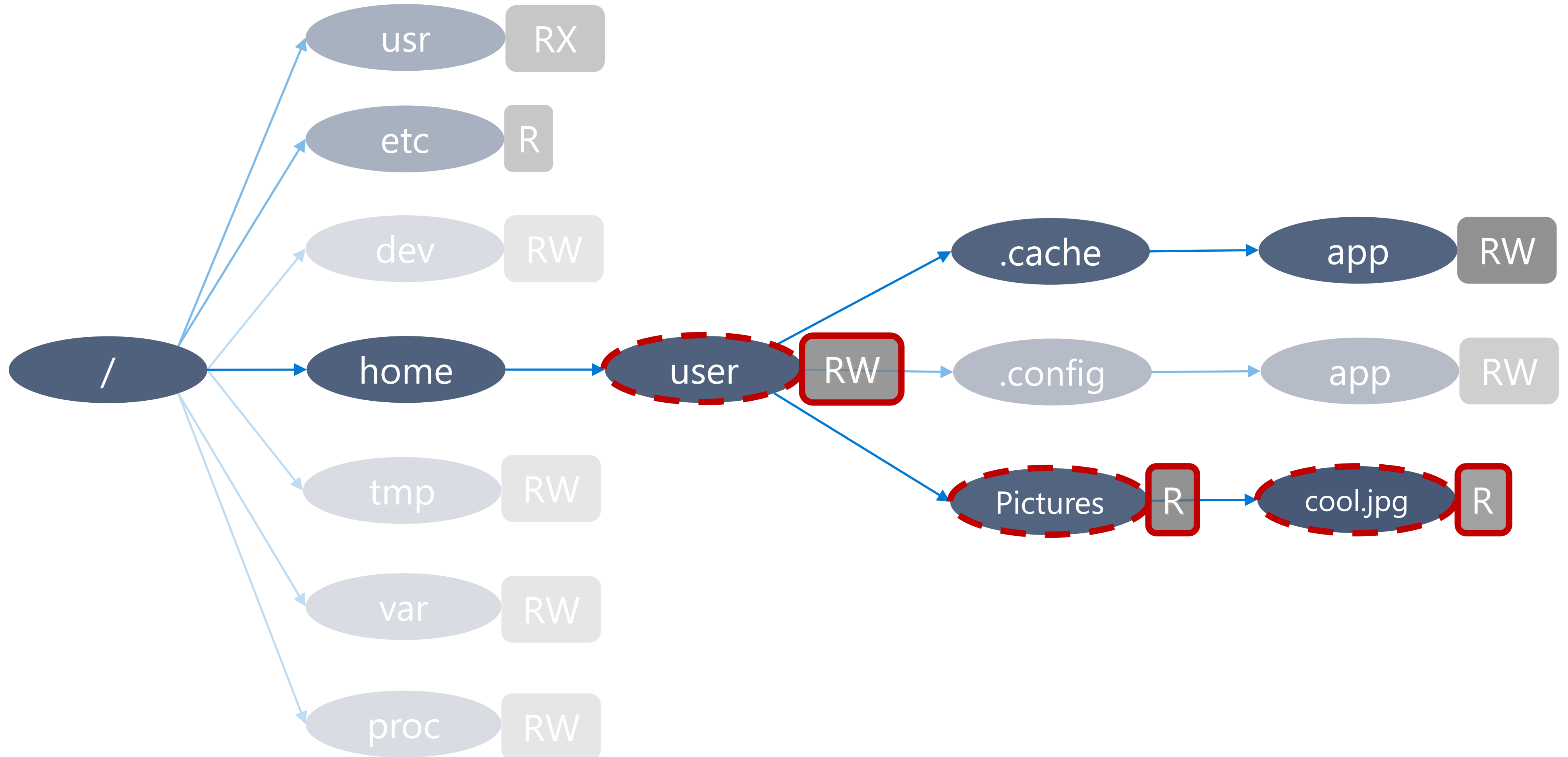


R Read
W Write
X eXecute

Example of filesystem policy composition

3rd layer ✓
2nd layer ✓
1st layer ✓

R Read
W Write
X eXecute



Sandboxing with Landlock

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 in a nutshell

Forward compatibility: kernel

Backward compatibility: responsibility of application developers

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

Will see more at the end of this talk…

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 ImageMagick!

ImageMagick

Pretty common set of tools to transform or display pictures: parse a lot of file formats

Use cases: CLI tool or (web) server

Attack scenario

CVE-2016-3714/ImageTragick: insufficient shell characters filtering that can lead to (potentially remote) code execution.

Let's say we have a vulnerable version, not necessarily this one.

Sandboxing this kind of tool can help mitigate the impact of such vulnerability: e.g., deny access to secret files

Agenda

1. Test an exploit
2. Find the sweet spot to restrict the process
3. Patch + build + test

Test exploit with vulnerable version

```
# Convert from one image format to another

convert /vagrant/exploit/malicious.mvg /tmp/out.png

# Solution patches are available in /vagrant/imagemagick-patches/*.patch
```

Main steps to patch

1. Declare the Landlock syscalls
2. Find what we want to sandbox and where it would make sense
3. Create a ruleset
4. Add static rules
5. Add dynamic rules
6. Restrict the task before potentially-harmful computation

Patch ImageMagick 1/9

```
# 1/ Go to the source directory

cd ~/src/ImageMagick-6.9.3-8
```

Patch ImageMagick 2/9

```
# 3/ Import Landlock syscall stubs and access right groups

cp /vagrant/sandboxer.c magick/landlock.h
vim magick/landlock.h

git add -A
git commit

# 4/ Look at the system's Landlock definitions and types (updated with up-to-
date 6.7 headers)

vim /usr/include/linux/landlock.h
```

Patch ImageMagick 3/9

```
# 5/ Look at the convert code and find a sweat spot for sandboxing

vim wand/convert.c

# Imagemagick doesn’t have a clear separation between argument parsing and
their evaluation: we need to patch the loop parsing arguments.

# 6/ Include landlock.h and create the ruleset in ConvertImageCommand()

(void) CopyMagickString(image_info->filename,filename,MaxTextExtent);

+   const struct landlock_ruleset_attr ruleset_attr = {
+       .handled_access_fs = ACCESS_FS_ROUGHLY_READ | ACCESS_FS_ROUGHLY_WRITE,
+   };
```

0003-WORKSHOP-Create-a-ruleset.patch

Build and test the patched ImageMagick

```
# Regularly build and check convert

make

# convert points to ./utilities/convert

convert /vagrant/exploit/malicious.mvg /tmp/out.png
```

Patch ImageMagick 4/9

```
# 7/ Create the ruleset

int ruleset_fd = landlock_create_ruleset(&ruleset_attr, sizeof(ruleset_attr), 0);

# 8/ Check for errors and log them

if (ruleset_fd < 0) {
    perror("LANDLOCK: Failed to create a ruleset");
    return MagickFalse;
}

# 9/ Close the ruleset

close(ruleset_fd);
```

0003-WORKSHOP-Create-a-ruleset.patch

Patch ImageMagick 5/9

```
# 10/

if (prctl(PR_SET_NO_NEW_PRIVS, 1, 0, 0, 0)) {
    perror("LANDLOCK: Failed to lock privileges");
    return MagickFalse;
}

if (landlock_restrict_self(ruleset_fd, 0)) {
    perror("LANDLOCK: Failed to restrict thread");
    return MagickFalse;
}
```

0004-WORKSHOP-Restrict-and-break-everything.patch

Build and test the patched ImageMagick

```
# Regularly build and check convert

make && convert /vagrant/exploit/malicious.mvg /tmp/out.png
```

Patch ImageMagick 6/9

```
# 11/ Add static rules: exceptions to the denied-by-default policy

+   struct landlock_path_beneath_attr rule;
+
+   printf("LANDLOCK: Adding rule for /usr");
+   rule.parent_fd = open("/usr", O_PATH | O_CLOEXEC);
+   rule.allowed_access = ACCESS_FS_ROUGHLY_READ;
+   if (landlock_add_rule(ruleset_fd, LANDLOCK_RULE_PATH_BENEATH, &rule, 0)) {
+       perror("LANDLOCK: Failed to create rule");
+       return MagickFalse;
+   }
+   close(rule.parent_fd);


    if (prctl(PR_SET_NO_NEW_PRIVS, 1, 0, 0, 0))
```

0005-WORKSHOP-Add-static-restrictions.patch

Patch ImageMagick 7/9

```
   # 12/ Add more static rules: /dev/null and /tmp (with appropriate access)

+  printf("LANDLOCK: Adding rule for /dev/null");
+  rule.parent_fd = open("/dev/null", O_PATH | O_CLOEXEC);
+  rule.allowed_access = LANDLOCK_ACCESS_FS_READ_FILE;
+  if (landlock_add_rule(ruleset_fd, LANDLOCK_RULE_PATH_BENEATH, &rule, 0)) {
+      perror("LANDLOCK: Failed to create rule");
+      return MagickFalse;
+  }
+  close(rule.parent_fd);


   if (prctl(PR_SET_NO_NEW_PRIVS, 1, 0, 0, 0))
```

0005-WORKSHOP-Add-static-restrictions.patch

Patch ImageMagick 8/9

```
  # 13/ Add a dynamic rule according to CLI arguments

+   printf("LANDLOCK: Adding rule for %s", filename);
+   rule.parent_fd = open(filename, O_PATH | O_CLOEXEC);
+   rule.allowed_access = LANDLOCK_ACCESS_FS_READ_FILE;
+   if (landlock_add_rule(ruleset_fd, LANDLOCK_RULE_PATH_BENEATH, &rule, 0)) {
+       perror("LANDLOCK: Failed to create rule");
+       return MagickFalse;
+   }
+   close(rule.parent_fd);


    if (prctl(PR_SET_NO_NEW_PRIVS, 1, 0, 0, 0))
```

0006-WORKSHOP-Handle-input-and-output-files.patch

Patch ImageMagick 9/9

```
    # 14/ Add more dynamic rules

+   char *out_path = strdup(argv[i+1]);
+   const char *out_dir = dirname(out_path);
+   [...]


    if (prctl(PR_SET_NO_NEW_PRIVS, 1, 0, 0, 0))
```

0006-WORKSHOP-Handle-input-and-output-files.patch

Build and test the final version

```
# Build and check convert

make && convert /vagrant/exploit/malicious.mvg /tmp/out.png
```

Exercise left to the readers

- Make the code more generic and maintainable
- Support the “fd:” URI scheme
- Support more commands
- Test with different kernel versions with help from the Landlock test tools

- …and send your patch upstream!

Compatibility and best-effort security

Incremental development

Because it is complex, a new kernel access control system cannot implement everything at once.

Landlock is useful as-is and it is gaining new features over time, which may enable to either add or remove restrictions.

Restrictions evolution over versions

Always denied

- Get new privileges
- Ptrace a parent sandbox
- Change FS topology
- Reparent files

Configurable

- Read file
- Write file
- …

Always allowed

- Change directory
- Read file metadata
- Change file ownership
- IOCTL
- Truncate file
- …

Landlock v1

Restrictions evolution over versions

Always denied

- Get new privileges
- Ptrace a parent sandbox
- Change FS topology
- ~~Reparent files~~

Configurable

- Read file
- Write file
- …
- Reparent files

Always allowed

- Change directory
- Read file metadata
- Change file ownership
- IOCTL
- Truncate file
- …

Landlock v1

Landlock v2

Restrictions evolution over versions

Always denied

- Get new privileges
- Ptrace a parent sandbox
- Change FS topology
- ~~Reparent files~~

Configurable

- Read file
- Write file
- …
- Reparent files
- Truncate file

Always allowed

- Change directory
- Read file metadata
- Change file ownership
- IOCTL
- ~~Truncate file~~
- …

Landlock v1

Landlock v2

Landlock v3

Application compatibility

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

Backward compatibility for applications is the responsibility of their developers, who may not be aware of the **kernel on which their application will run**.

Each new Landlock feature increments the Landlock ABI version, which is useful to implement a fallback mechanism: **best-effort** approach.

Good sandboxing rules

1. Transparent to users
2. Best-effort with minimal requirement
3. Handle strict restrictions
4. Runtime configuration with maximum execution

Rule #1: Transparent to users

Most of the time, configurations are not updated.

Requirements:

- Leverage the current application's configuration as much as possible
- Dynamic checks to identify required runtime resources

Rule #2: Best-effort with minimal requirement

Don't break my application!

Enforce **restrictions as much as possible** according to the running kernel, and being able to disable the whole sandboxing if a required feature is not supported (e.g., the refer access right for file reparenting).

Use case:

- For end users, **opportunistically sandbox** applications without error

Rule #3: Handle strict restrictions

Create an option to force sandboxing and error out if anything goes wrong (not enabled by default).

Use cases:

1. For developers and CI **tests**, to be sure that sandboxing is not an issue for legitimate use
2. For security software, to be sure that a set of security properties are **guarantee**

Rule #4: Runtime configuration with maximum execution

Help **identify sandboxing specific code issues**.

Run the same code as much as possible (i.e., same behavior: check same files, make same syscalls…) but only enforce restrictions when requested.

Should be simple to set or unset at run time according to:

- Test environment (e.g., build profile, variables)
- User configuration

Wrap-up

ImageMagick patch

- Use the native CLI arguments:
  - 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

Ongoing and next steps:

- Add new access-control types: more networking, signals, IPCs…
- Add audit support to ease debugging
- Improve kernel performance

See GitHub issues: landlock-lsm/linux

Contribute

- Develop new new access types
- Improve libraries: Rust, Go…
- Challenge the implementation
- Improve documentation or tests
- **Sandbox your applications** and others’
  - Secure Open Source Rewards
  - Google Patch Rewards

Questions?

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

Past talks: https://landlock.io

landlock@lists.linux.dev

Thank you!