Microsoft

Landlock: The Linux sandboxing mechanism

How to protect your users?

Mickaël Salaün

2024-01-22

Agenda

1. State of the security
2. Defense challenges and sandboxes
3. State of the art
4. Landlock
5. Demo
6. Filesystem access-control
7. Network access-control
8. Audit
9. Ongoing development

State of the security

Pragmatic statement #1

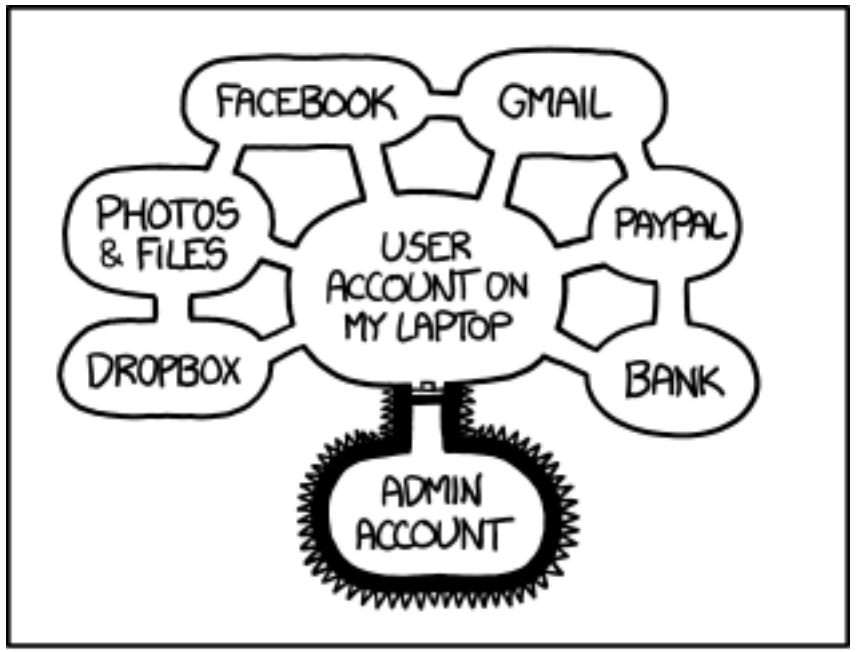
Attacker

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

Defender

An innocuous and trusted process can **become malicious during its lifetime** because of bugs exploited by attackers.

Protecting data != admin rights



https://xkcd.com/1200

Pragmatic statement #2

Attacker

Bulletproof (and useful) software is costly and very difficult to achieve (if ever possible).

Defender

Pragmatic **multi-layer security** increases guarantees and confidence.

Pragmatic statement #3

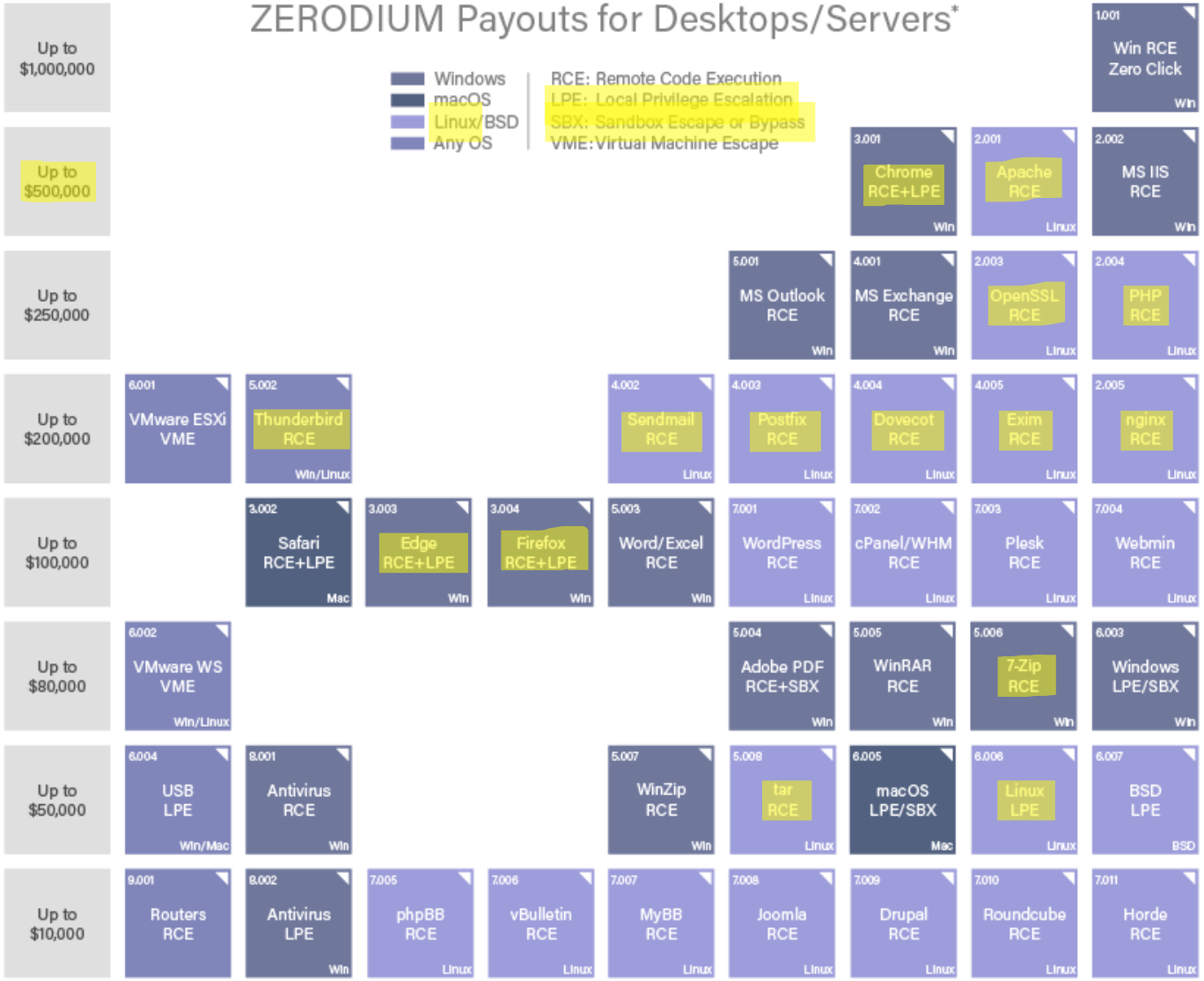
Attacker

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

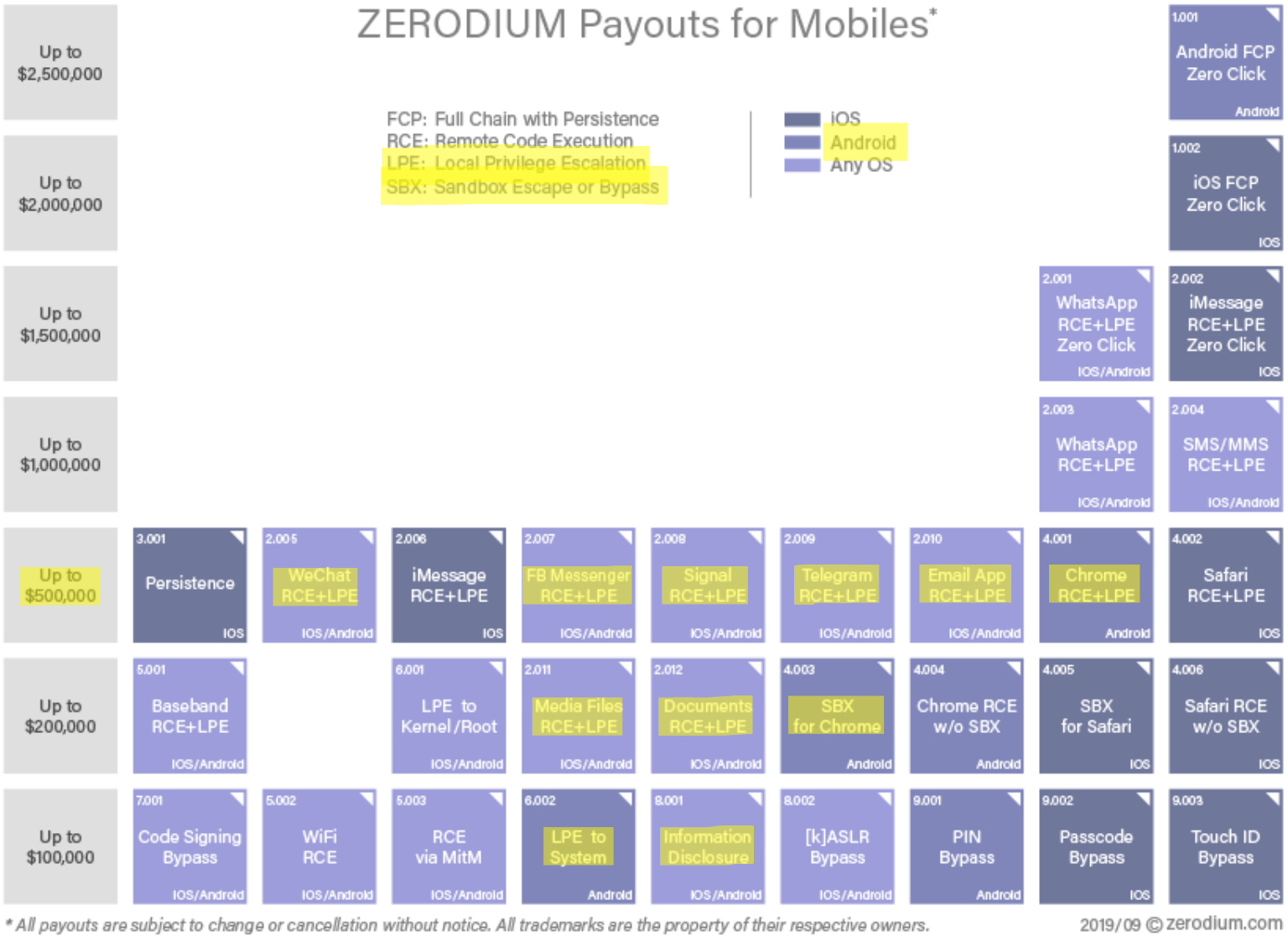
Defender

Hardening increases attack cost.

How much does it cost?



How much does it cost?



Trusted Computing Base

Assumptions: hardware and supply chains are secure, the configuration is secure (enough), and other critical parts of the system (e.g., kernel, important services) are trusted and uncompromised.

We trust the honesty and well-intention of people in charge of our security, according to our point of view:

- Sysadmin
- Developer
- User

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.

Defense challenges and sandboxes

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

Sandboxing

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

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.

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:

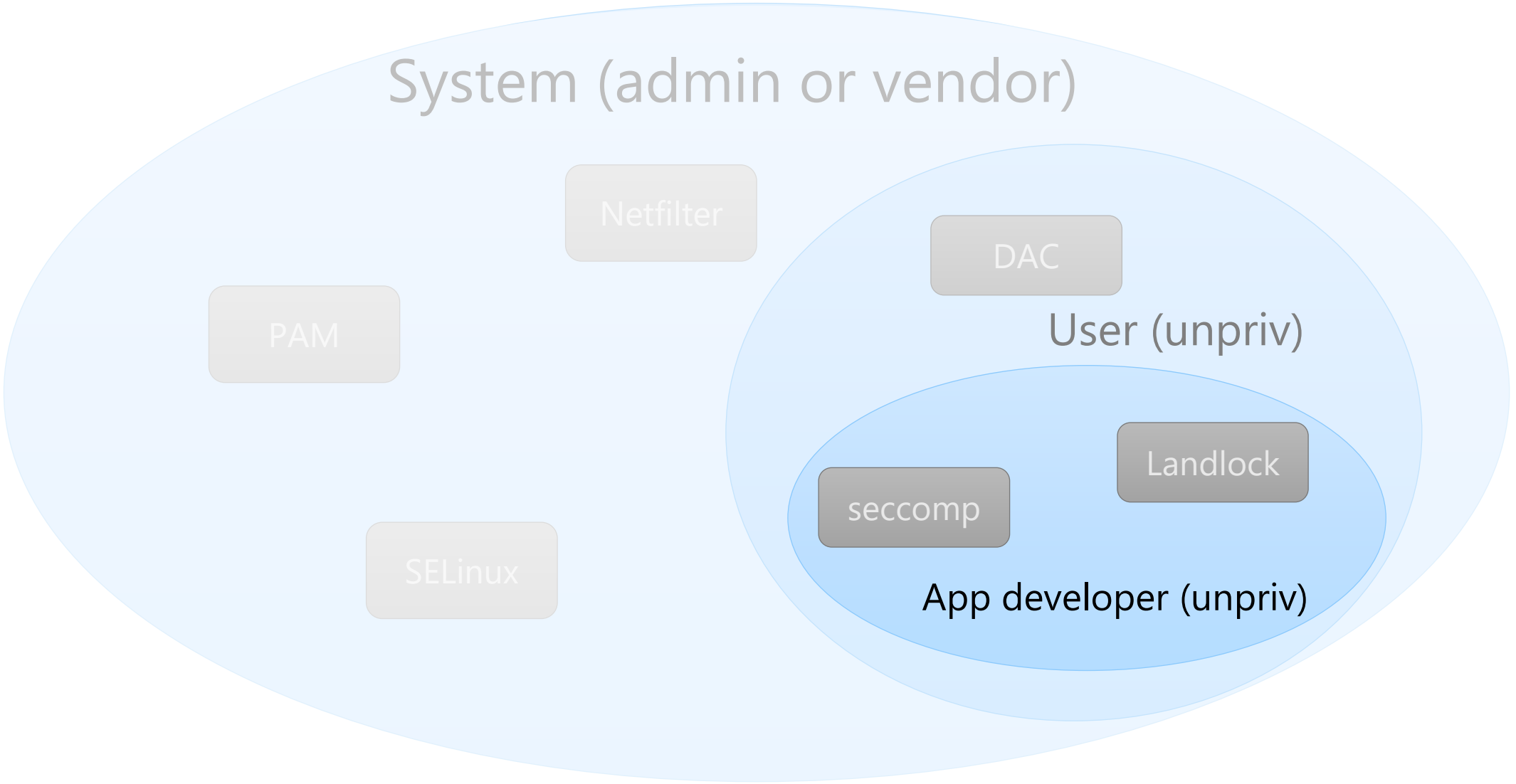
- Follow the least privilege principle
- Innocuous and composable security policies

State of the art

Non-Linux systems

- XNU Sandbox
- Capsicum
- Pledge and Unveil
- AppContainer and Application Guard

Available Linux security features



Comparisons of different sandbox-ish mechanisms

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

✓ Yes, compared to others

✗ No, compared to others

! In some way, but with limitations

Landlock

Landlock's goal

Landlock is an access control system available to unprivileged processes on Linux, which empowers **developers** and **users** to sandbox their applications.

It enables to create safe security sandboxes as **new security layers** in addition to the existing system-wide access-controls to help **mitigate the security impact** of bugs or unexpected behaviors.

Threat models

Untrusted applications

Protect from malicious third-party code thanks to sandbox managers or container runtimes.

Exploitable bugs in trusted applications

Protect from vulnerable code maintained by app developers, thanks to embedded security policy.

Use cases

Built-it application sandboxing, e.g.:

- Parsers hardening (e.g., archive tools, file format conversion, renderers, etc.)
- Web browsers
- Network and system services

Sandbox managers, e.g.:

- Containers
- Init systems

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
- User interaction

Testable and can be kept in sync with **evolving business logic** over time.

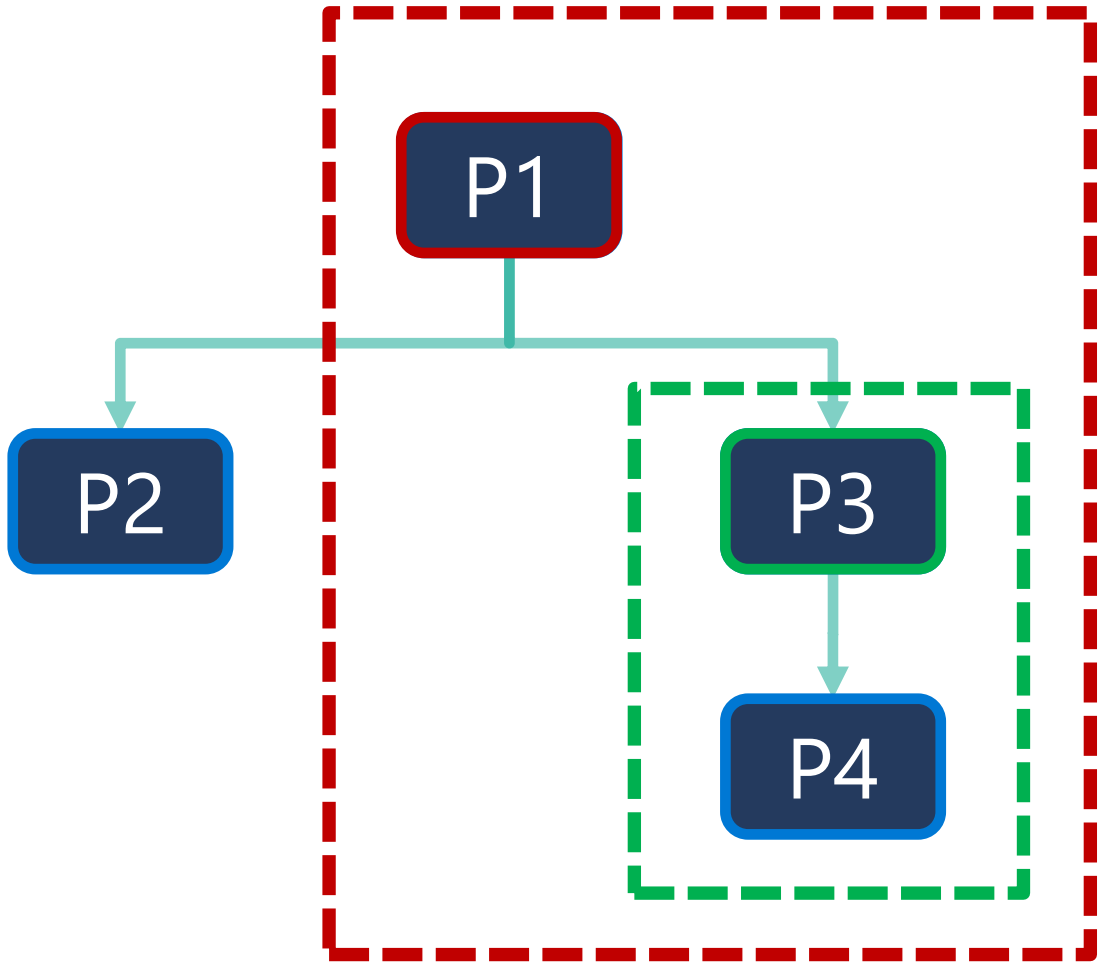
How Landlock works?

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.

Sandbox policies hierarchy



P_ Sandbox creator

Sandbox domain

A Linux Security Module

Security framework for the kernel:

- Majors: SELinux, AppArmor… **Landlock**
- Integrity/authenticity: IMA/EVM
- Hardening: Yama, Lockdown…

Landlock is a **stackable LSM**

What does it bring?

3 new system calls:

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

Linux 6.7:

- Kernel: 2000+ SLOC
- Tests: 5400+ SLOC
- Documentation: 6000+ words

Unprivileged access control

Prevent bypass through other processes.

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

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

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

Multi-layer security

- Nested sandboxes
- Composed sandboxes

Composable security policies

Compose with other access-control systems: LSM stacking

Compose all Landlock sandbox policies

- Standalone policies targeting different services/apps

Kernel constraints

- No file extended attributes
- No (absolute) path

Developer advantages

- **Lockless concurrent development**: no bottleneck because of one global policy
- Easier to maintain a set of small policies

Current access-control

Implicit restrictions

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

Explicit access rights

- Files
- Networking (TCP)
- …

Where is Landlock?

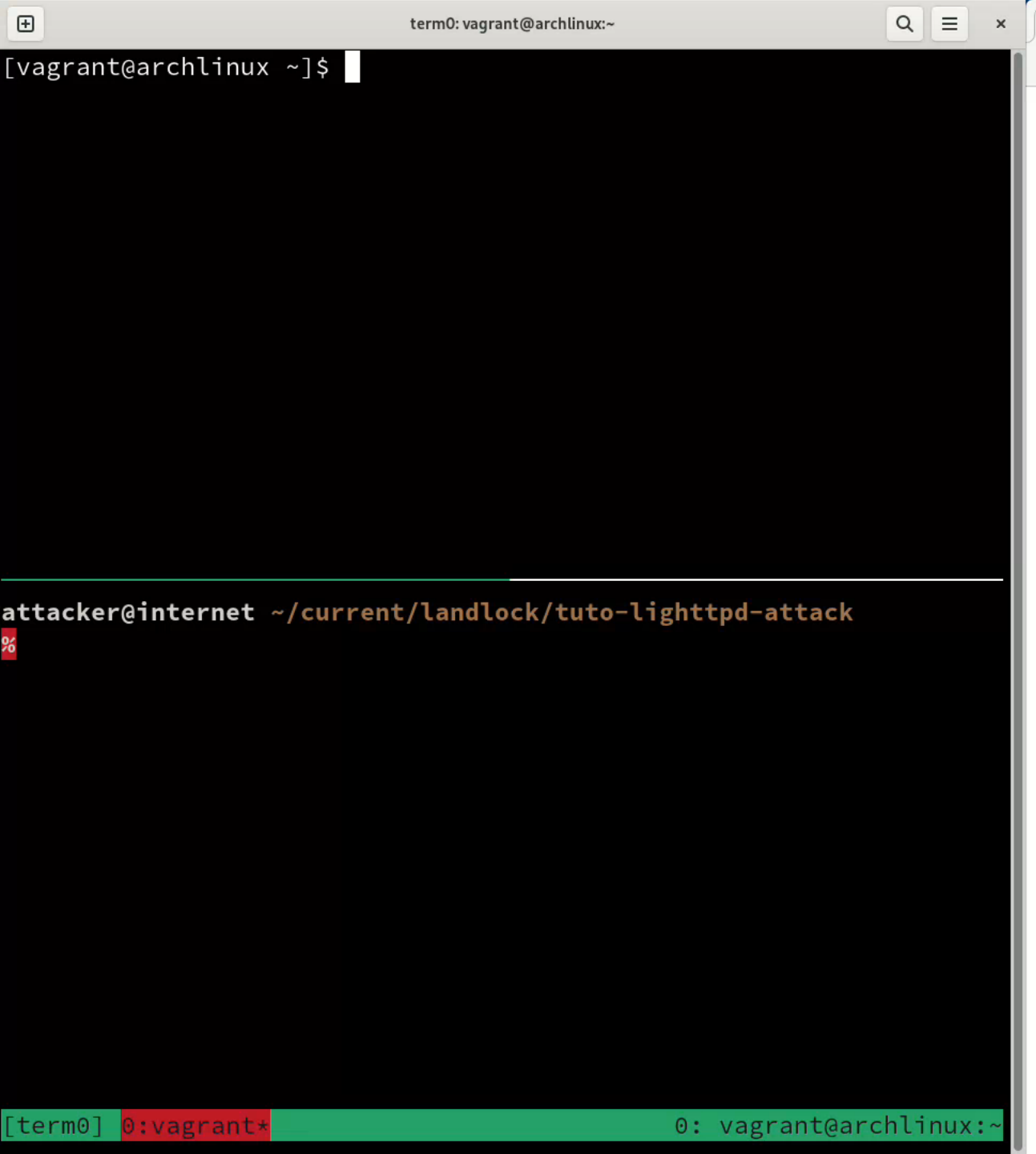
Part of the mainline Linux kernel since v5.13 (2021)

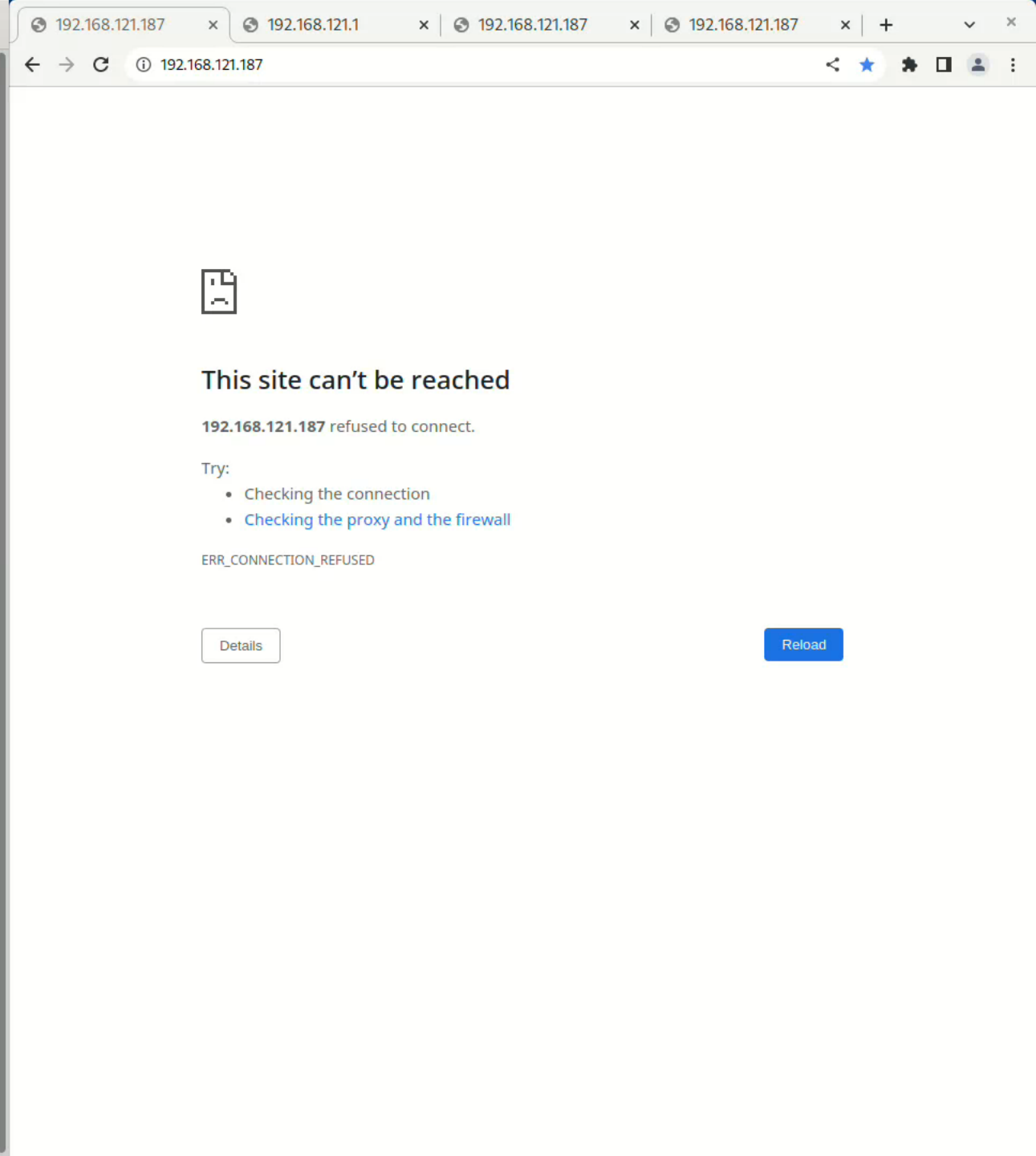
Enabled by default on multiple distros:

- Ubuntu 22.04 LTS
- Fedora 35
- Arch Linux
- Alpine Linux
- Gentoo
- Debian Sid
- chromeOS
- CBL-Mariner
- WSL2

Demo: sandboxed web service

Netdev conference


term0: vagrant@archlinux:~
[vagrant@archlinux ~]$
attacker@internet ~/current/landlock/tuto-lighttpd-attack
%
[term0] 0:vagrant* 0: vagrant@archlinux:~


192.168.121.187
192.168.121.1
192.168.121.187
192.168.121.187
192.168.121.187
This site can’t be reached
192.168.121.187 refused to connect.
Try:
Checking the connection
Checking the proxy and the firewall
ERR_CONNECTION_REFUSED
Details
Reload

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

File hierarchy identification

Ephemeral inode tagging

- Access rights are tied to inodes by user space thanks to opened file descriptors
- Lifetime of such tags depends on associated sandbox domain lifetimes and underlying superblock lifetimes

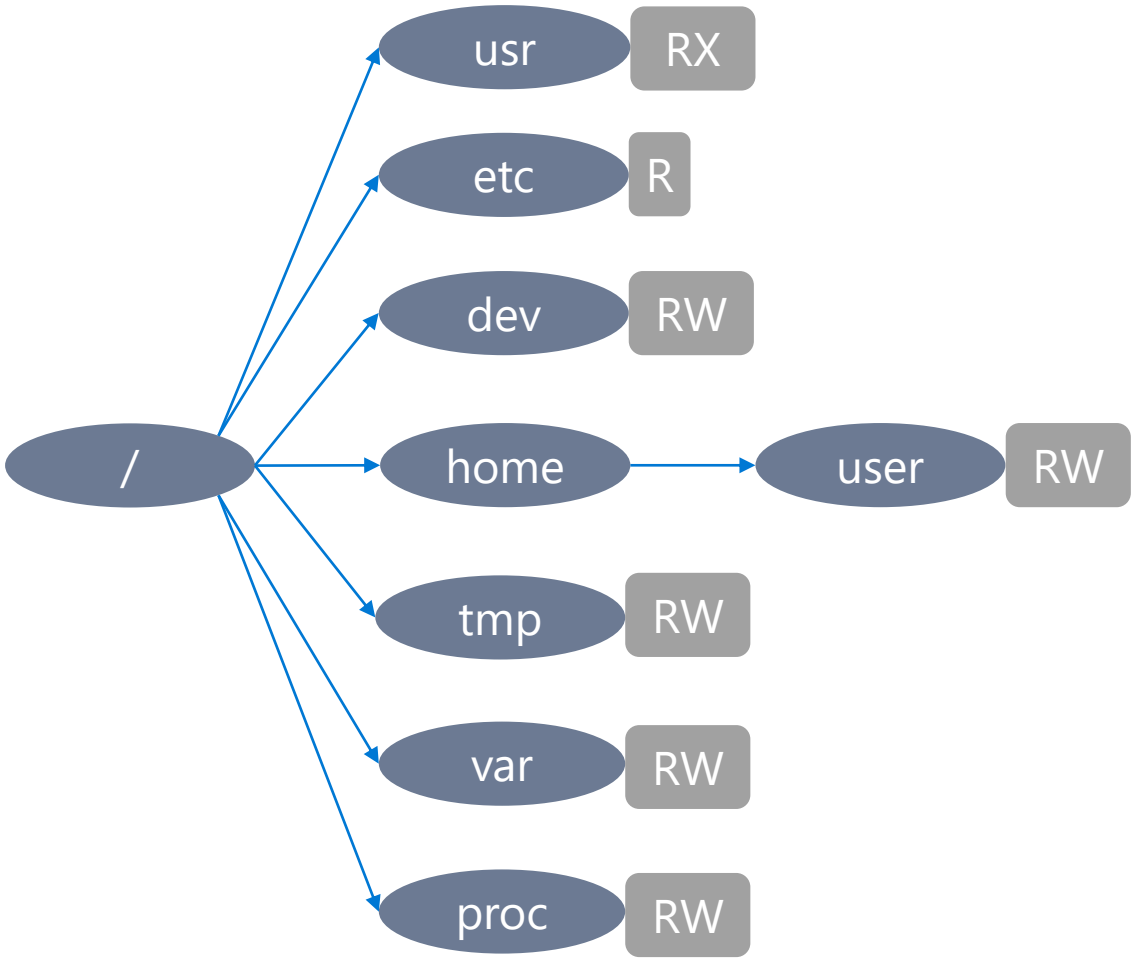
File hierarchy check

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

Example of filesystem policy composition

Example of filesystem policy composition

1st layer



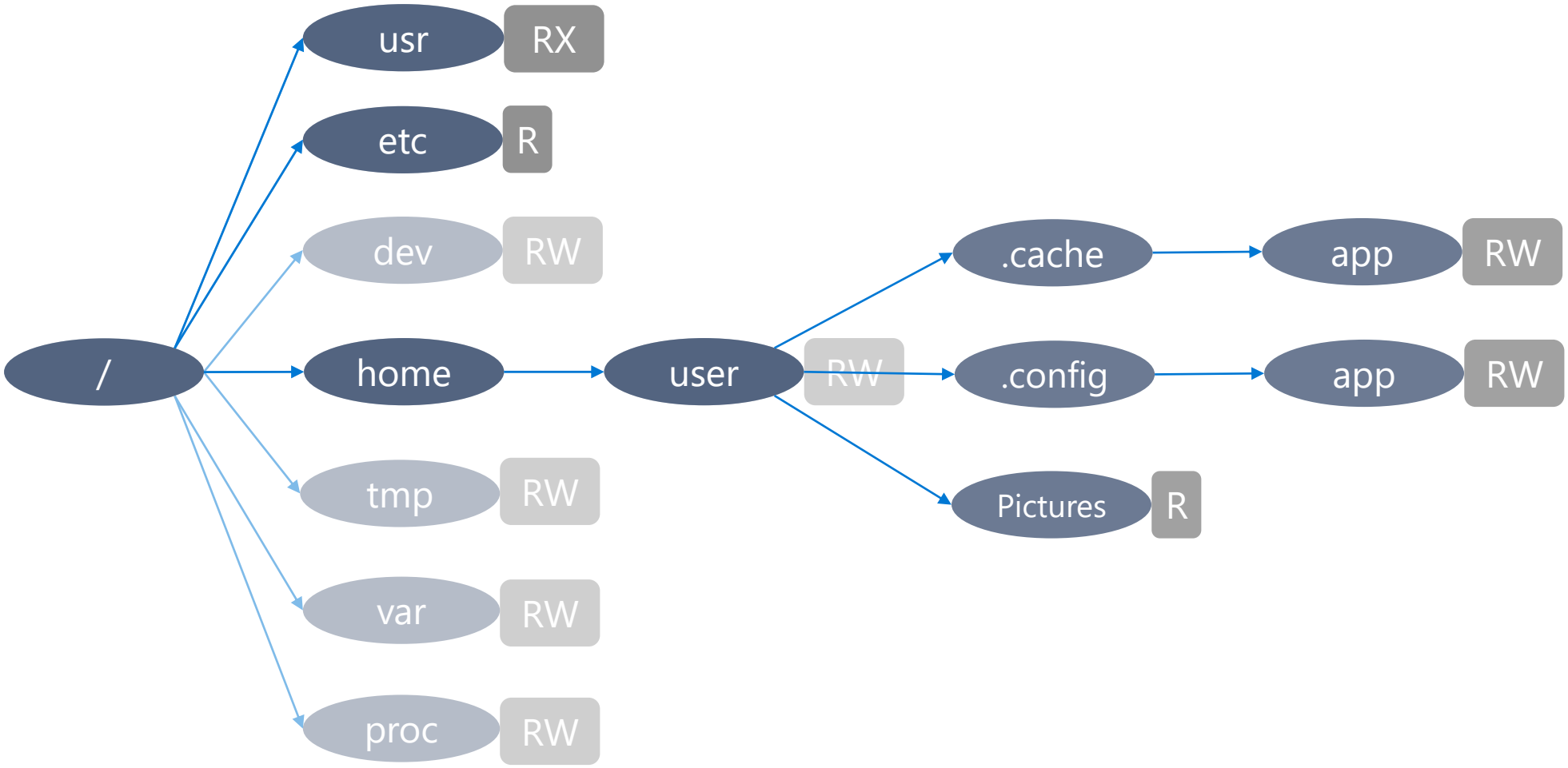
R Read

W Write

X eXecute

Example of filesystem policy composition

2nd layer



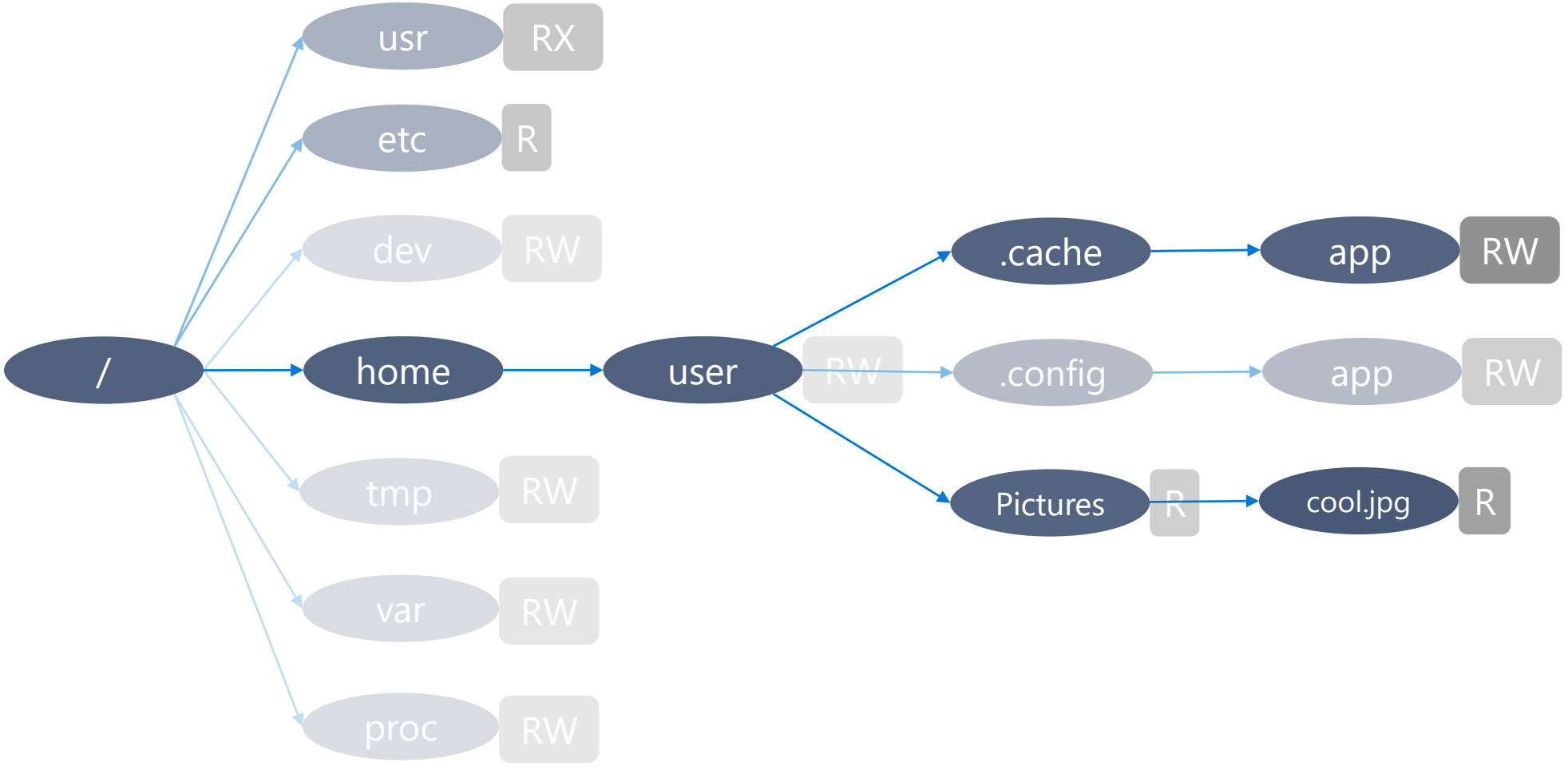
R Read

W Write

X eXecute

Example of filesystem policy composition

3rd layer



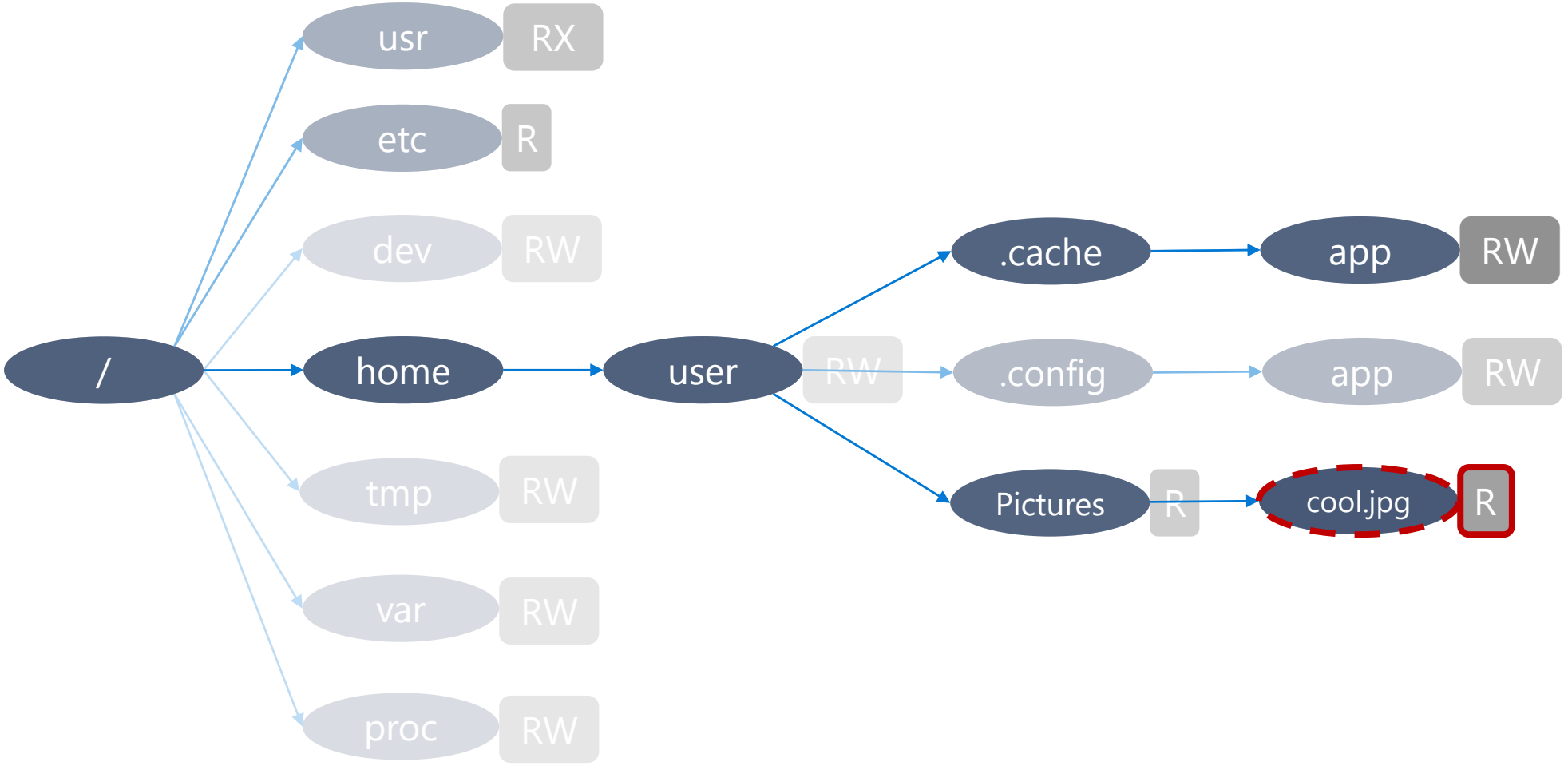
R Read

W Write

X eXecute

Example of filesystem policy composition

3rd layer ✔



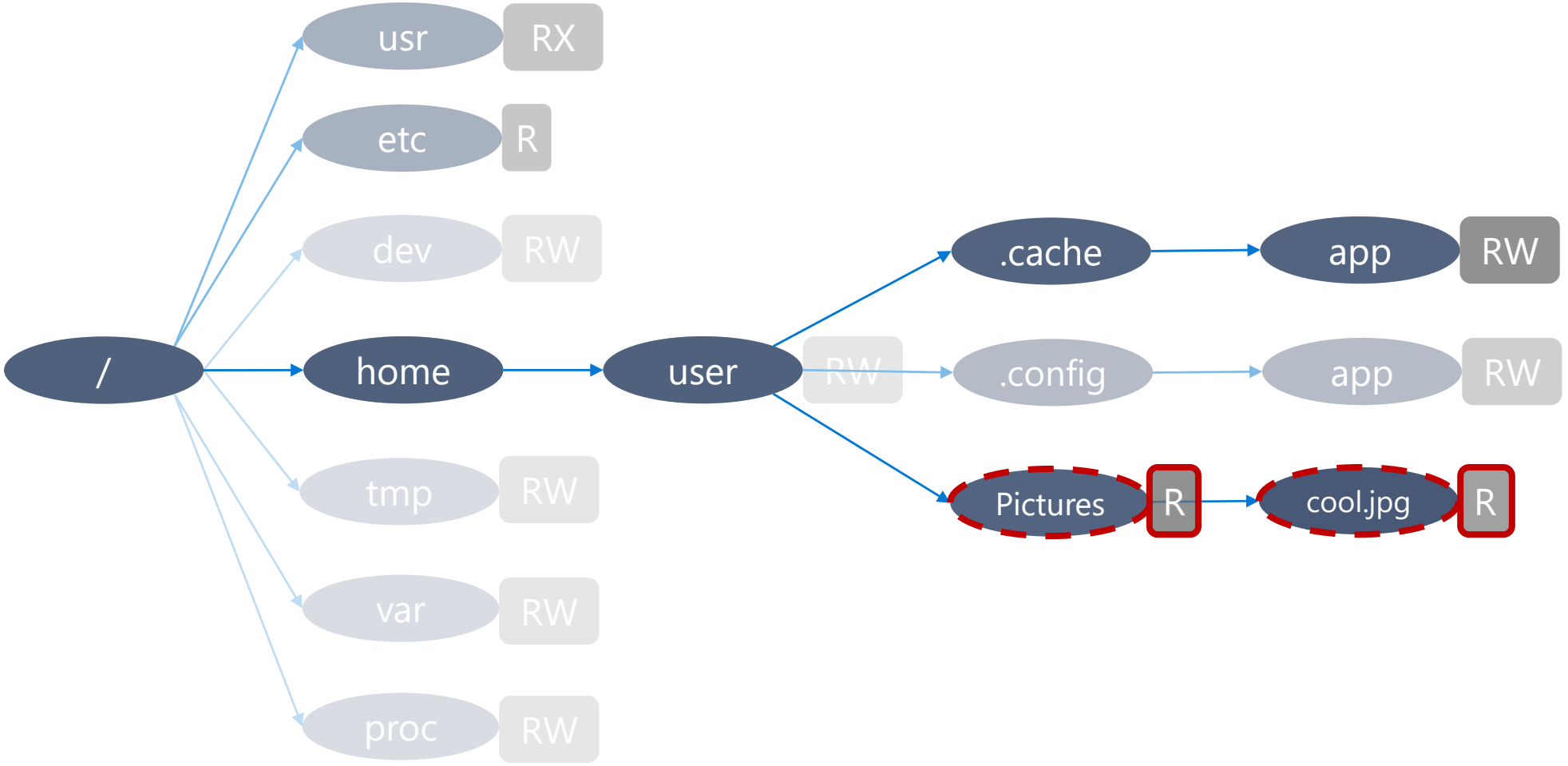
R Read

W Write

X eXecute

Example of filesystem policy composition

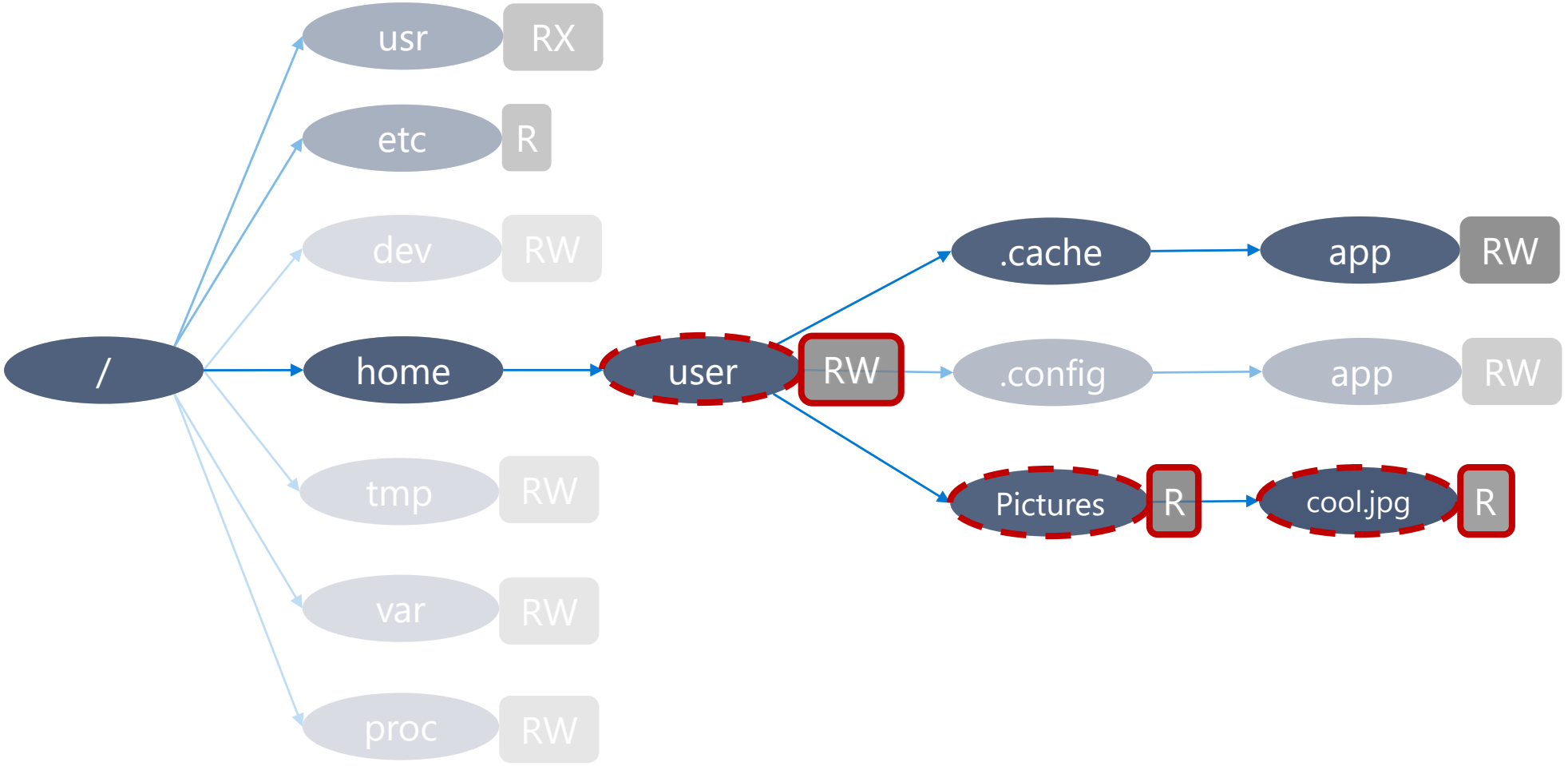
3rd layer ✔
2nd layer ✔



R Read
W Write
X eXecute

Example of filesystem policy composition

3rd layer ✔
2nd layer ✔
1st layer ✔



R Read
W Write
X eXecute

Network access-control

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

Network access rights

TCP access rights (Linux 6.7)

Minimal app-centric firewall to control:

- Bindings to TCP ports
- Connections to TCP ports

Future features

- TCP listen right
- Socket creation
- Abstract unix sockets

Audit support: denied access logs

Non-goal: Track access requests

- Not the goal of Landlock
- The LSM framework is not design to see everything, but mainly to deny actions

Other kernel features and related tools are available: e.g. trace-cmd, bpftrace

Goal: Log Landlock denials

Help users with different use cases:

- App developers: to ease and speed up sandboxing support
- Power users: to understand denials
- Sysadmins: to look for users' issues
- Tailored distro maintainers: to get usage metrics from their fleet
- Security experts: to detect attack attempts

Constraints

Security policies are:

- Unprivileged
- Multiple and standalone
- Nested
- Dynamic

Not available to unprivileged users

Relying on the Linux audit mechanism

Wrap-up

Landlock LSM

Unprivileged access control:

- The Linux sandboxing mechanism
- Can confine trusted and untrusted code
- Composable security policies

Landlock tools

Libraries: Rust, Go, Haskell, C…

Development: glibc, strace

Some early public users:

- Minijail (chromeOS sandbox manager)
- Suricata (network IDS)
- Landlock Make (hermetic build system)
- Game of Trees (version control system)
- Keysas (USB malware cleaning station)
- rust-wasm-landlock (sandboxed WebAssembly runtime)
- …

Landlock roadmap

Ongoing and next steps:

- Add new access-control types: IOCTL, networking, signals…
- Update and merge audit features to ease debugging
- Improve kernel performance

See GitHub issues: landlock-lsm/linux

Contribute

- Develop new (kernel) features (e.g., new access types)
- Write new tests (Kselftest or KUnit)
- Challenge the implementation
- Improve documentation
- **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!