LINUX SECURITY SUMMIT EUROPE

Update on Landlock: IOCTL support

Günther Noack
Google Switzerland

Who am I?

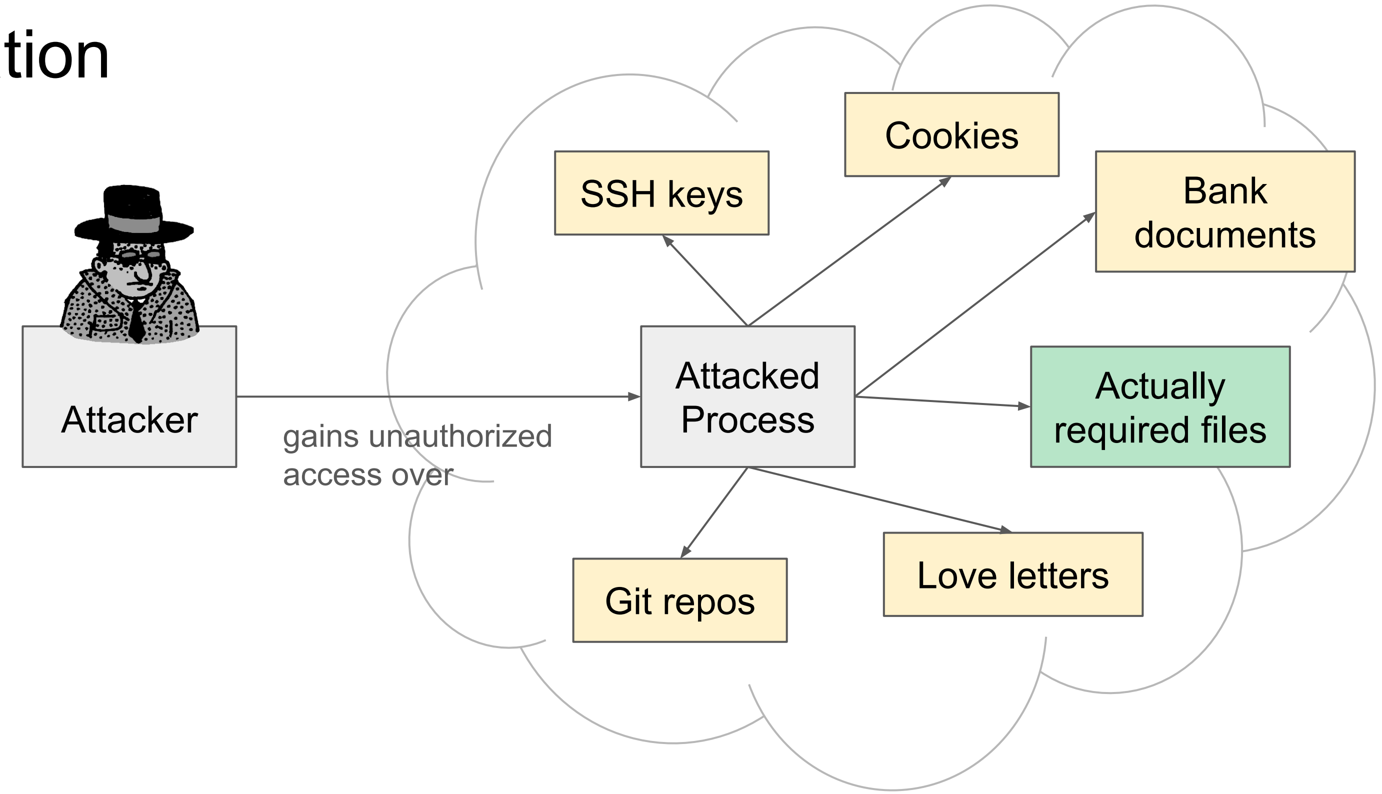
- Günther Noack
- Interested in computer security for 20+ years
- Staff Software Engineer at Google in Zürich
- Landlock contributor since 2022, reviewer since 2024
- Author and maintainer of the Go Landlock library
- Likes to run and swim in his free time

Outline

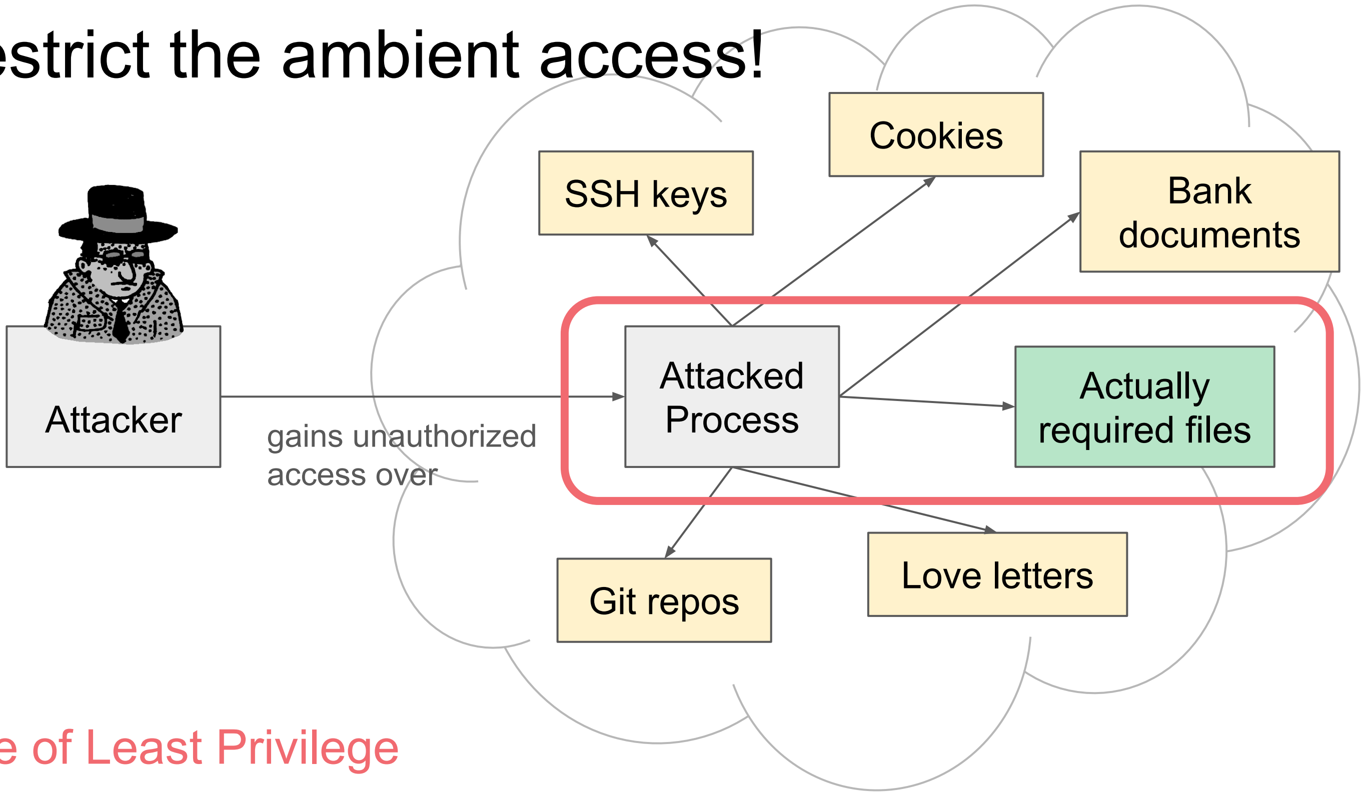
- Landlock Motivation
- Technical Overview
- Landlock IOCTL support
- Other Notable News
- Upcoming Features

Motivation

Motivation


Attacker
gains unauthorized access over
Attacked Process
SSH keys
Cookies
Bank documents
Actually required files
Love letters
Git repos

Let’s restrict the ambient access!



Principle of Least Privilege

Attempt 1: Seccomp-BPF

Seccomp-BPF is an unprivileged, BPF-scriptable “packet filter” for system calls.

Rules are defined in terms of the permitted syscall numbers and argument values.

Problems:

Users need to maintain an up-to-date list of existing used syscalls

- Shared libraries (including glibc!) change the syscalls that they use over time.
- These might still be unknown at the time of policy definition
- Leads to compatibility problems

We can’t follow pointer arguments from BPF scripts

- Making decisions based on file paths is not feasible without resorting to more involved trickery.

SELinux, AppArmor

- Mandatory Access Control
- Policies usually defined by system administrators or Linux distribution
  - Needs to be kept in sync with the sandboxed software
- SELinux and AppArmor are only available on a subset of distributions

Landlock Summary

- Introduced in Linux 5.13 (2021) by Mickaël Salaün
- **Unprivileged** sandboxing mechanism
- Lets **processes define security policies** for themselves
- **Backwards-compatibility** story is well-supported
- **Software developers are in charge** of defining policies
  - Developers know their software best
  - Policies are maintained as part of the code
- Policies specified in terms of meaningful abstraction boundaries
- Better granularity for the security policy (e.g. enable policy after initialization phase)

Vision

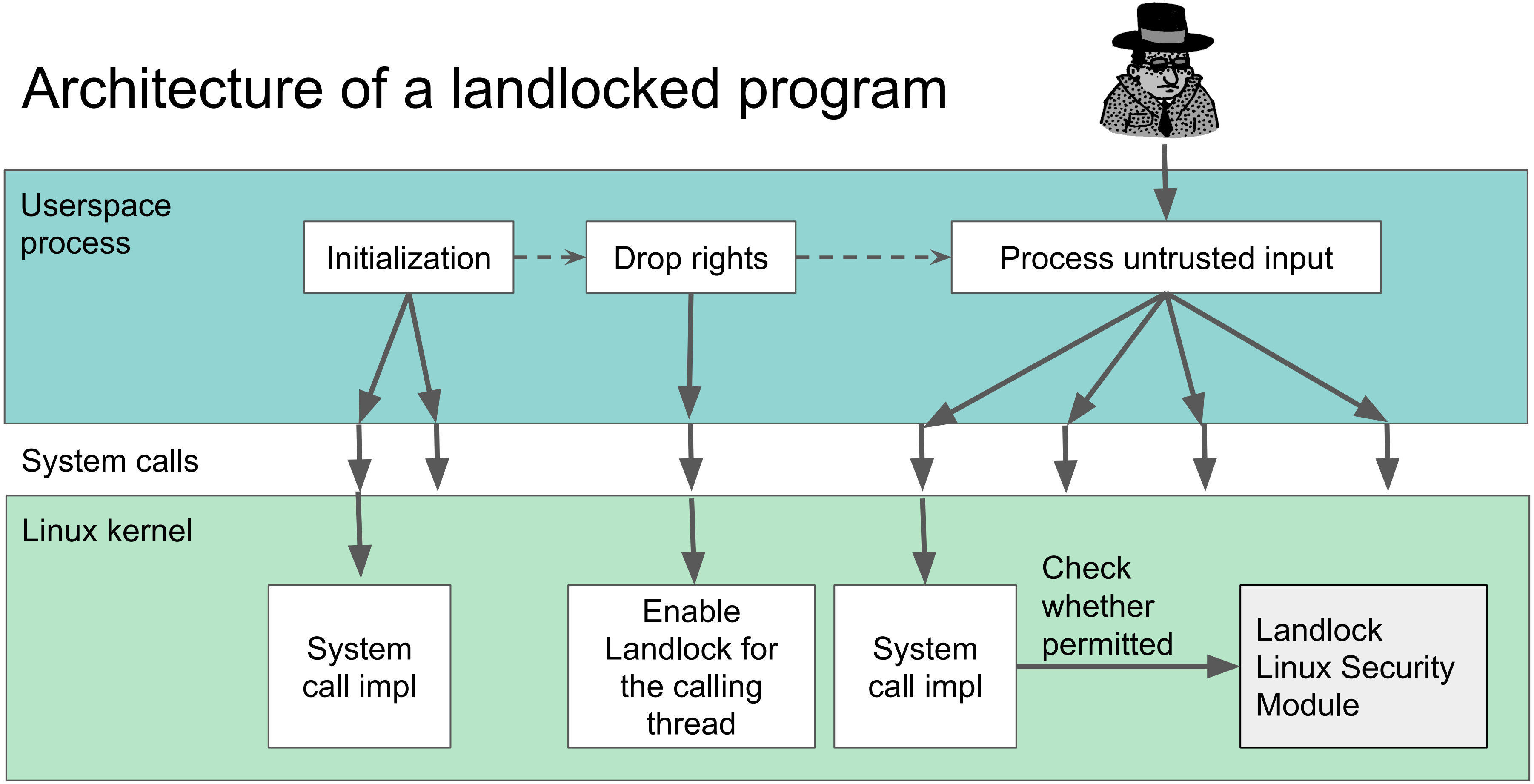
- Make Landlock **simple to use for software developers**
- Useful not only for building sandboxing tools,
- **but also for self-sandboxing of normal Linux programs**

Examples

- “Pipeline” tools: xzdec, convert, grep, …
- Network tools: netcat, ping, …
- Document viewers (e.g. zathura)
- etc.

Technical Overview

Architecture of a landlocked program



Enforcing a Landlock ruleset

The Landlock ruleset defines the policy to be enforced.

1. Create ruleset file descriptor and define the restricted operations: **`landlock_create_ruleset(2)`**
2. (Optionally) add exceptions: **`landlock_add_rule(2)`**
3. Enforce ruleset: **`landlock_restrict_self(2)`**

1. landlock_create_ruleset(2)

```
struct landlock_ruleset_attr ruleset_attr = {
    .handled_access_fs  = landlock_fs_access_rights[abi-1],
    .handled_access_net = landlock_net_access_rights[abi-1],
};
int ruleset_fd = landlock_create_ruleset(
    &ruleset_attr, sizeof(ruleset_attr), 0);

if (ruleset_fd < 0) {
  /* error */
}
```

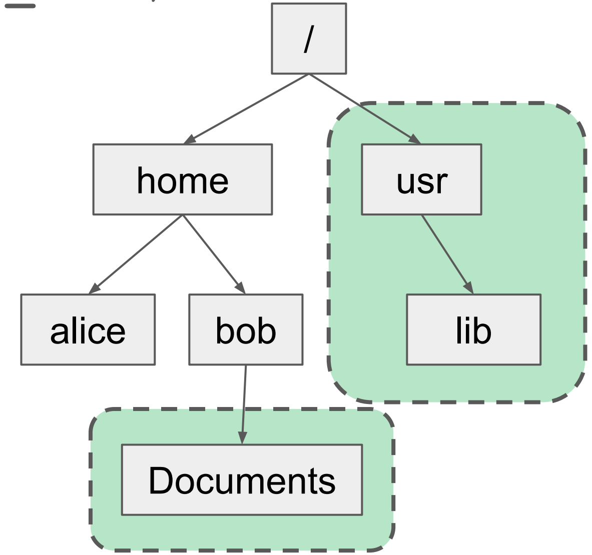
The operations to be restricted (bit masks)

2. landlock_add_rule(2) (file example)

```
struct landlock_path_beneath_attr attr = {
    .allowed_access = LANDLOCK_ACCESS_FS_READ_FILE,
    .parent_fd = dir_or_file_fd,
};
int res = landlock_add_rule(
    ruleset_fd,
    LANDLOCK_RULE_PATH_BENEATH, &attr, 0);

if (res < 0) {
  /* error */
}
```

Define exceptions to the restricted operations



3. landlock_restrict_self(2)

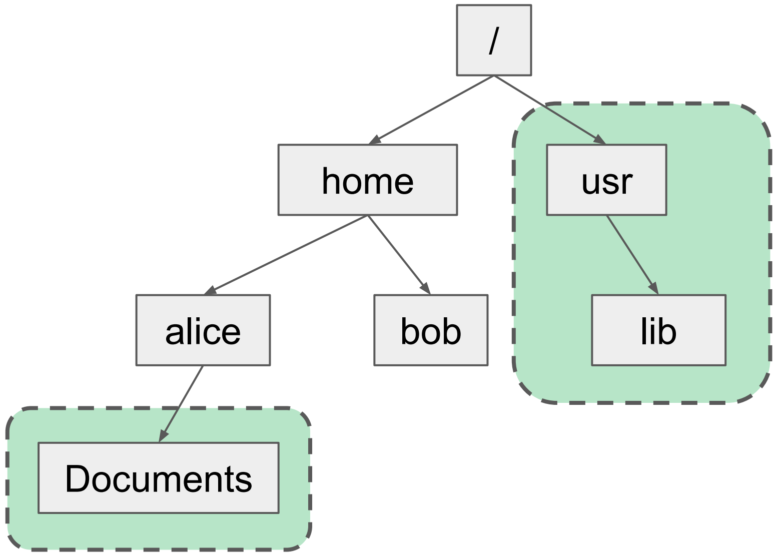
```
int res = landlock_restrict_self(ruleset_fd, 0);
```

Enforce

```
if (res < 0) {
  /* error */
}
```

Restrictable operations

1. File system operations
2. TCP networking



Restrictable operations (file system)

Common operations:
LANDLOCK_ACCESS_FS_EXECUTE, LANDLOCK_ACCESS_FS_WRITE_FILE, LANDLOCK_ACCESS_FS_READ_FILE, LANDLOCK_ACCESS_FS_TRUNCATE, LANDLOCK_ACCESS_FS_READ_DIR

Opening for reading/writing, executing, truncation

Directory entry manipulation:
LANDLOCK_ACCESS_FS_REMOVE_DIR, LANDLOCK_ACCESS_FS_REMOVE_FILE, LANDLOCK_ACCESS_FS_MAKE_CHAR, LANDLOCK_ACCESS_FS_MAKE_DIR, LANDLOCK_ACCESS_FS_MAKE_REG, LANDLOCK_ACCESS_FS_MAKE_SOCK, LANDLOCK_ACCESS_FS_MAKE_FIFO, LANDLOCK_ACCESS_FS_MAKE_BLOCK, LANDLOCK_ACCESS_FS_MAKE_SYM, LANDLOCK_ACCESS_FS_REFER

IOCTL:
LANDLOCK_ACCESS_FS_IOCTL_DEV

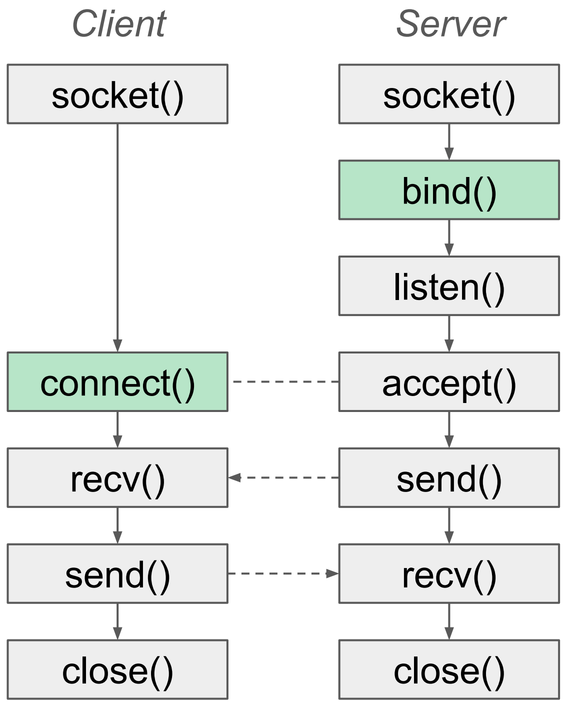
Using ioctl(2) on device files (new!)

Restrictable operations (TCP)

`LANDLOCK_ACCESS_NET_BIND_TCP`
`LANDLOCK_ACCESS_NET_CONNECT_TCP`

Exceptions (rules) can be defined per port number.

Upcoming restrictions: see later slides



Backwards compatibility and ABI versioning

Software developers often do not know what kernel the software will run on.

- Landlock is versioned with ABI versions
- Userspace can probe for the available ABI version
- If needed, restrict only a subset of what you mean to restrict

This problem will become less relevant as older Linux versions phase out

Landlock Limitations

- Landlock is getting built incrementally
- Restricts many important operations already, but some are still missing

Some operations are additionally limited when using Landlock:

- Manipulation of file system topology (i.e. mounting, pivot_root)
- Requires NO_NEW_PRIVS flag
- Restricted use of ptrace()

IOCTL support

IOCTL interface

generic multiplexer!
e.g. **TIOCGWINSZ**, **FS_IOC_GETFLAGS**, ...

From **ioctl**(2):

```
int ioctl(int fd, unsigned long op, ...);
```

```
void *argp
```

Example:

```
struct winsize ws;
if (ioctl(STDOUT_FILENO, TIOCGWINSZ, &ws) < 0)
  err(1, "TIOCGWINSZ");
```

Motivation for IOCTL support in Landlock

We try to apply the “principle of least privilege” with Landlock

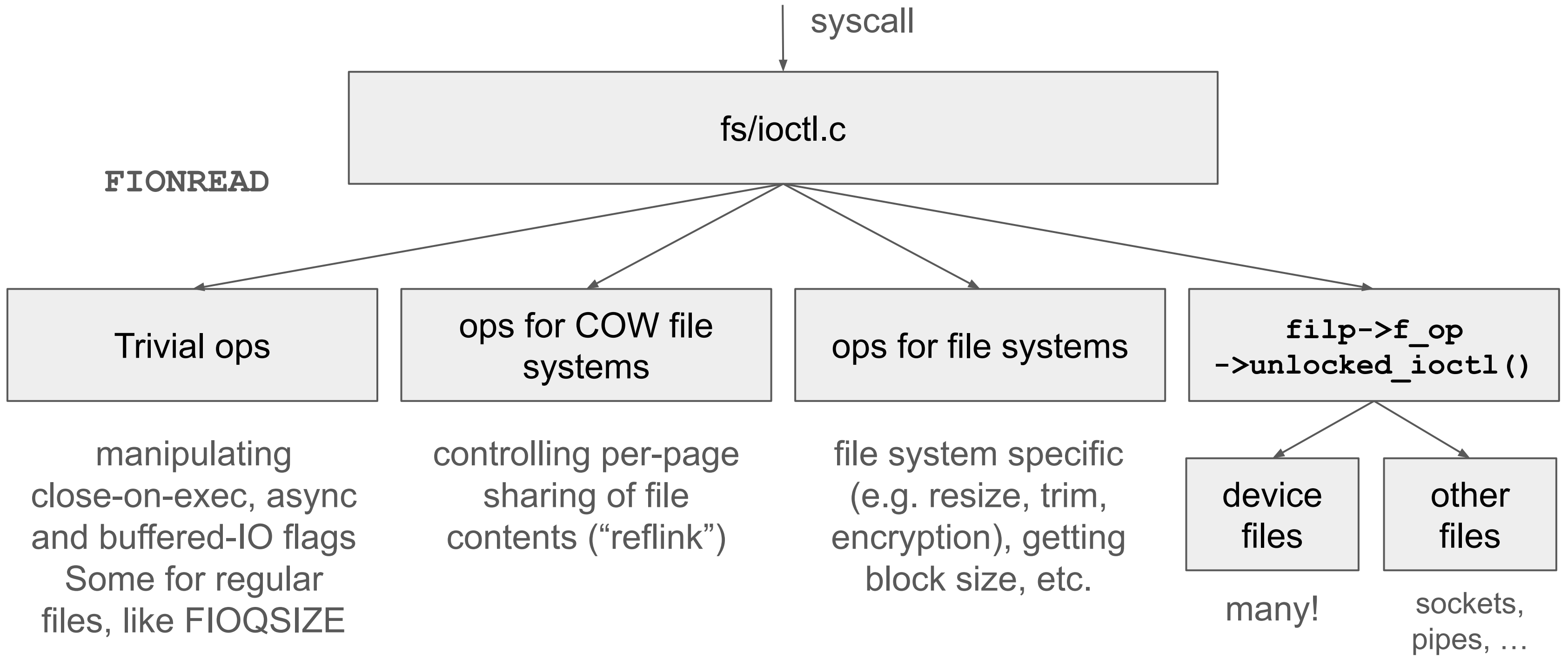
Without Landlock IOCTL support:

- On any opened file, ioctl(2) can be called
- It is difficult to say what functionality that even entails
- It becomes difficult to reason about a program’s privileges

We should have Landlock policies that

- **restrict the bulk of IOCTL commands by default,**
- and which have narrow exceptions where needed.

IOCTL: What are these operations?



syscall
fs/ioctl.c
FIONREAD
Trivial ops
ops for COW file systems
ops for file systems
filp->f_op ->unlocked_ioctl()
manipulating close-on-exec, async and buffered-IO flags Some for regular files, like FIOQSIZE
controlling per-page sharing of file contents ("reflink")
file system specific (e.g. resize, trim, encryption), getting block size, etc.
device files
other files
many!
sockets, pipes, …

filp->f_op->unlocked_ioctl() is hard to reason about

Sometimes implemented like this:

```
static long my_ioctl(struct file *filp, unsigned int cmd,
                     unsigned long arg) {
    acquire_resources(filp);
    switch (cmd) {
    case MY_IOCTL1: /* … */ break;
    case MY_IOCTL2: /* … */ break;
    }
    release_resources(filp);
}
```

This executes code,
even for unknown cmds!
Can be complicated.

What are the criteria by which we can allow or deny?

```
int security_file_ioctl(struct file *file, unsigned int cmd, unsigned long arg);
```

Need to strike a balance between flexibility and ease of use.

- Properties of file
  - Criteria: File path: **Yes** (uncontroversial; fits into existing Landlock scheme)
  - Criteria: File "type": **Difficult as configurable allow-list**
    - …and device numbers? **Also difficult**
- Properties of cmd / op
  - Criteria: Encoded Read/Write flag (_IO, _IOR, _IOW, _IOWR): **Not feasible***
  - Criteria: Configurable allow-lists: **Difficult**
  - Criteria: Hardcoded allow-lists: **Yes**

User-configurability complicates implementation, but benefit is unclear.
Avoid Seccomp-BPF-style programmability.

* https://lwn.net/Articles/428140/

Final decision: Restrict non-trivial device IOCTLs by path

- **File path:** Restriction can be configured by file path
- **File type:** Only restrict IOCTLs for **device files** (`LANDLOCK_ACCESS_FS_IOCTL_DEV` access right)
- **Op: Harmless operations are permitted** independent of the access right:
  - Trivial and reasonable:
    - **`FIOCLEX`**, **`FIONCLEX`**, **`FIONBIO`**, **`FIOASYNC`**
  - Error code consistency (do not work on device files):
    - **`FIOQSIZE`**, **`FS_IOC_FIEMAP`**
    - **`FICLONE`**, **`FICLONERANGE`**, **`FIDEDUPERANGE`**
  - Operate on file system, not on file:
    - **`FIFREEZE`**, **`FITHAW`**, **`FIGETBSZ`**, **`FS_IOC_FSUUID`**, **`FS_IOC_GETFSSYSFSPATH`**

Final decision: Restrict non-trivial device IOCTLs by path

In other words:

For non-device files, or if **`LANDLOCK_ACCESS_FS_IOCTL_DEV`** is allowed:

- all IOCTLs are permitted

If **`LANDLOCK_ACCESS_FS_IOCTL_DEV`** is restricted **on a given device file**:

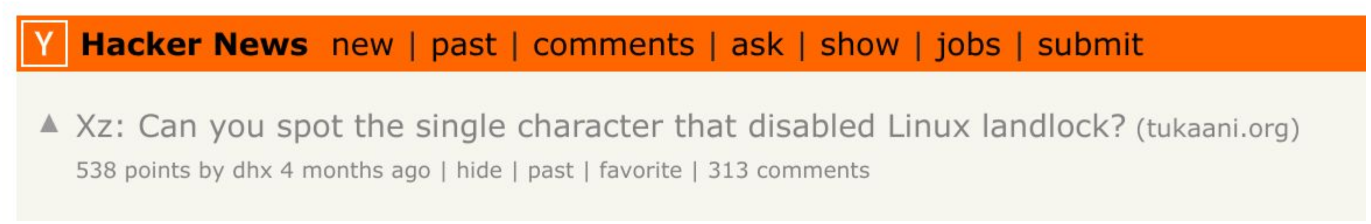
- **`FIOCLEX`**, **`FIONCLEX`**, **`FIONBIO`** and **`FIOASYNC`** can still be used
- Some other operations also work, but are audited to be safe
- No IOCTL gets dispatched to the device driver

The usual **`landlock_path_beneath_attr`** rule can be used to allow **`LANDLOCK_ACCESS_FS_IOCTL_DEV`** on a file or directory hierarchy, even when it is generally forbidden by default.

Notable News

March 2024: Landlock was disabled in the XZ Backdoor

The attackers who created a backdoor in the XZ compression library disabled XZ’s Landlock sandbox, by sabotaging a check for Landlock presence in the CMakeFile.


Y Hacker News new | past | comments | ask | show | jobs | submit

▲ Xz: Can you spot the single character that disabled Linux landlock? (tukaani.org)
538 points by dhx 4 months ago | hide | past | favorite | 313 comments

Screenshot: https://news.ycombinator.com/item?id=39874404

August 2024: CVE-2024-42318 (cred_transfer)

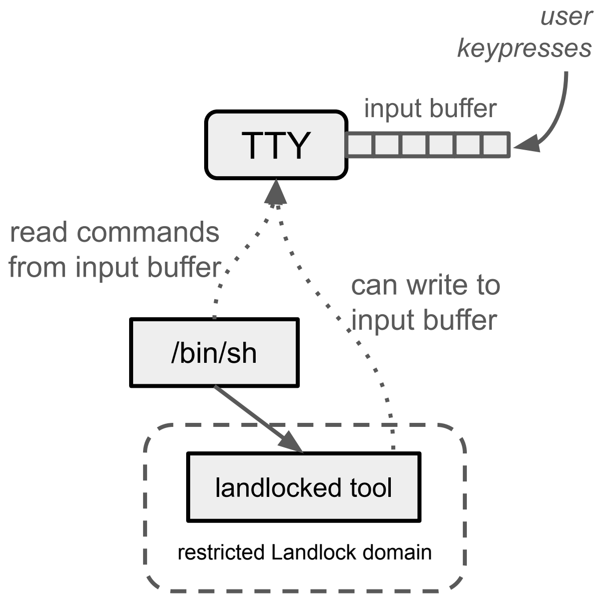
https://www.cve.org/CVERecord/?id=CVE-2024-42318
“Don’t lose track of restrictions on cred_transfer”

- Processes could disable their Landlock policy through the use of keyctl()
- Spotted and fixed by Jann Horn 🙏

A more comprehensive fix is also out, which should get rid of the cred_transfer() LSM hook altogether.

Problems with TIOCSTI IOCTL and equivalents

- TIOCSTI emulates TTY keypresses
- frequently misused for privilege escalation
- recommended solution: pseudoterminals
  - 1000+ lines of C, requires a process that shovels data to and from the pseudoterminal
- **affects landlocked programs**, but pseudo-ttys are not realistic for small tools
- same thing is possible with the copy&paste subcommands of the TIOCLINUX IOCTL


user keypresses
input buffer
TTY
read commands from input buffer
can write to input buffer
/bin/sh
landlocked tool
restricted Landlock domain

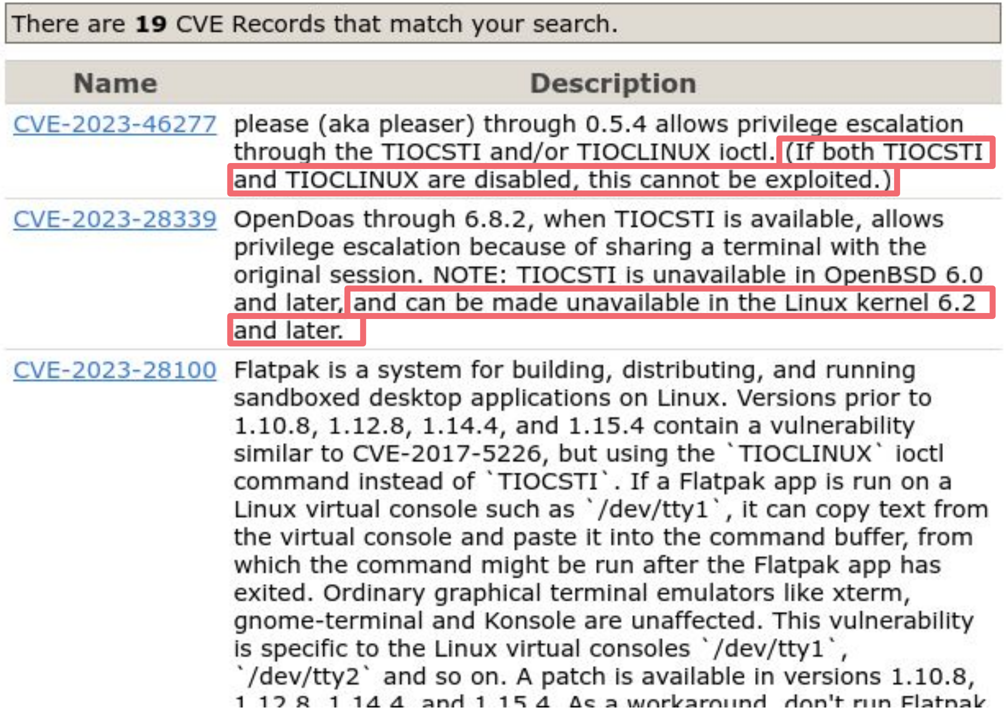
TIOCSTI and equivalents now require CAP_SYS_ADMIN

This functionality should **now require CAP_SYS_ADMIN by default**:

TIOCSTI: Linux 6.2 (2022), fixed by Kees Cook

TIOCLINUX's Copy&Paste: Linux 6.7 (2023), fixed by Hanno Böck

https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=TIOCSTI, captured 2024-08-23



There are **19** CVE Records that match your search.

Name	Description
CVE-2023-46277	please (aka pleaser) through 0.5.4 allows privilege escalation through the TIOCSTI and/or TIOCLINUX ioctl. (If both TIOCSTI and TIOCLINUX are disabled, this cannot be exploited.)
CVE-2023-28339	OpenDoas through 6.8.2, when TIOCSTI is available, allows privilege escalation because of sharing a terminal with the original session. NOTE: TIOCSTI is unavailable in OpenBSD 6.0 and later, and can be made unavailable in the Linux kernel 6.2 and later.
CVE-2023-28100	Flatpak is a system for building, distributing, and running sandboxed desktop applications on Linux. Versions prior to 1.10.8, 1.12.8, 1.14.4, and 1.15.4 contain a vulnerability similar to CVE-2017-5226, but using the `TIOCLINUX` ioctl command instead of `TIOCSTI`. If a Flatpak app is run on a Linux virtual console such as `/dev/tty1`, it can copy text from the virtual console and paste it into the command buffer, from which the command might be run after the Flatpak app has exited. Ordinary graphical terminal emulators like xterm, gnome-terminal and Konsole are unaffected. This vulnerability is specific to the Linux virtual consoles `/dev/tty1`, `/dev/tty2` and so on. A patch is available in versions 1.10.8, 1.12.8, 1.14.4, and 1.15.4. As a workaround, don't run Flatpak

Upcoming Features

IPC Restrictions: Connecting to Abstract UNIX Socket

Abstract UNIX domain sockets behave like named UNIX Domain Sockets, but they are not exposed in the file system, but registered in a kernel-global list by name.

Status: Patch in review (Tahera Fahimi)

IPC Restrictions: Signal scoping support

Restrict the sending of signals across processes in different Landlock domains.

Status: Patch in review (Tahera Fahimi)

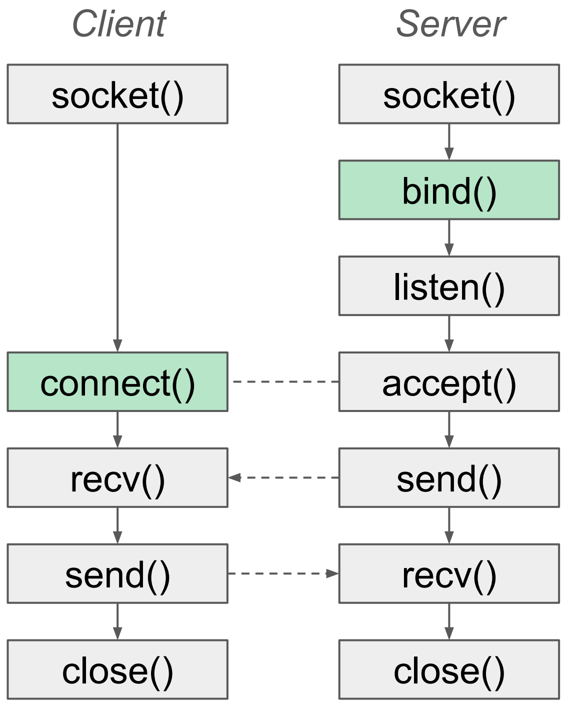
Upcoming restrictable network operations

Already supported:

- bind()
- connect()

Upcoming restrictions currently in consideration:

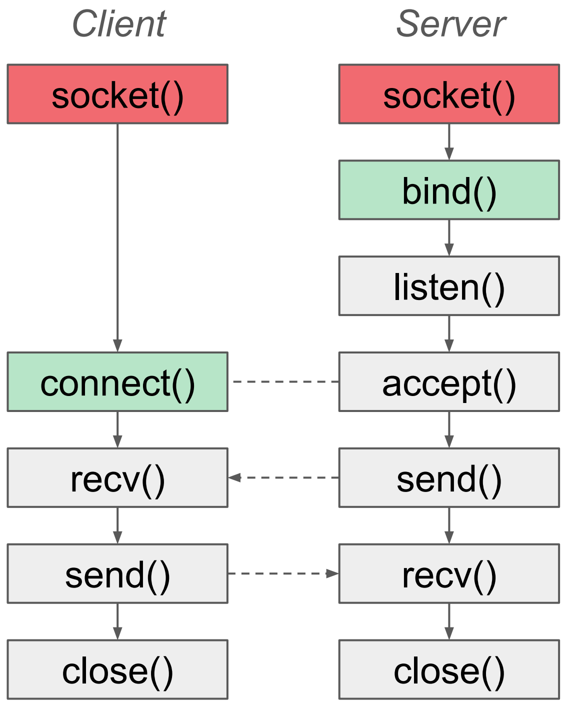
- Creation of new sockets (socket(2))
- listen(2)
- connect(2) with **`AF_UNSPEC`**



Creation of new sockets (socket(2))

Lets us easily deny the use of a whole range of uncommon socket types and families.

Status: Patch in review (Mikhail Ivanov)

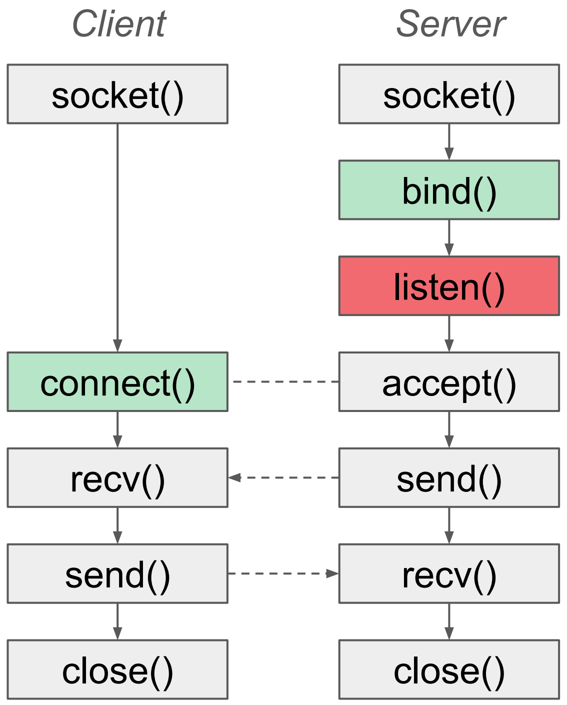


listen(2)

Putting a socket in passive mode (start listening)

Restricting bind(2) is not enough – it is possible to listen(2) without doing a bind(2) before.

Status: Patch in review (Mikhail Ivanov)

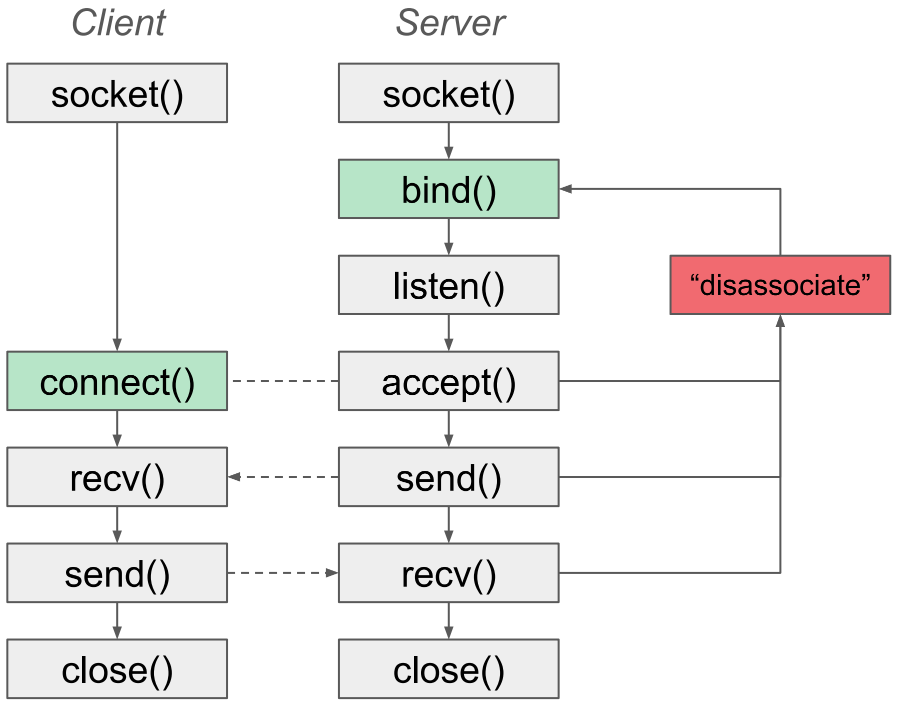


Disassociation (connect(2) with AF_UNSPEC)

Disassociating an existing connection

This operation resets a socket back into a state where is can be used for new listen(2) or connect(2) operations.

Status: Not in review yet



Questions

Questions

🌎 https://landlock.io/

📬 https://subspace.kernel.org/lists.linux.dev.html

🐞 https://github.com/landlock-lsm/linux/issues