File access-control per container with Landlock

Mickaël Salaün

ANSSI

February 4, 2018

Secure user-space software

How to harden an application?

- secure development
- follow the least privilege principle
- compartmentalize exposed processes

Secure user-space software

How to harden an application?

- secure development
- follow the least privilege principle
- compartmentalize exposed processes

Container constraints

- each container image can be unique
- and independent from the host
- hence may need dedicated access-control rules

$\Rightarrow$ embedded security policy

What can provide the needed features?

	Fine-grained control	**Embedded policy**	**Unprivileged use**
SELinux...	✓		

What can provide the needed features?

	Fine-grained control	**Embedded policy**	**Unprivileged use**
SELinux...	✓		
seccomp-bpf		✓	✓
namespaces		✓	∼

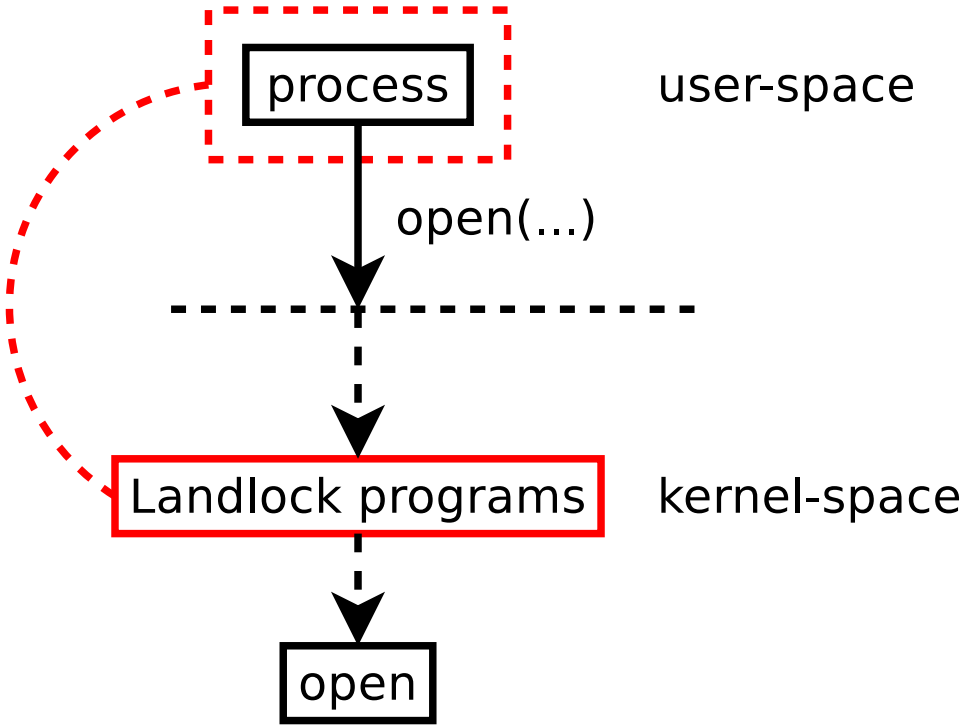
What can provide the needed features?

	Fine-grained control	Embedded policy	Unprivileged use
SELinux...	✓		
seccomp-bpf		✓	✓
namespaces		✓	∼
Landlock	✓	✓	✓[1]

Tailored access control to match your needs: programmatic access control

[1] Disable on purpose for the initial upstream inclusion, but planned to be enabled after a test period.

Landlock overview



Landlock: patch v8

- a minimum viable product
- focused on filesystem access control
- using eBPF

extended Berkeley Packet Filter

In-kernel virtual machine

- safely execute code in the kernel at run time
- widely used in the kernel: network filtering, seccomp-bpf, tracing...
- can call dedicated functions
- can exchange data through maps between eBPF programs and user-space

extended Berkeley Packet Filter

In-kernel virtual machine

- safely execute code in the kernel at run time
- widely used in the kernel: network filtering, seccomp-bpf, tracing...
- can call dedicated functions
- can exchange data through maps between eBPF programs and user-space

Static program verification at load time

- memory access checks
- register typing and tainting
- pointer leak restrictions
- execution flow restrictions

The Linux Security Modules framework (LSM)

LSM framework

- allow or deny user-space actions on kernel objects
- policy decision and enforcement points
- kernel API: support various security models
- 200+ hooks: `inode_permission`, `inode_unlink`, `file_ioctl`...

The Linux Security Modules framework (LSM)

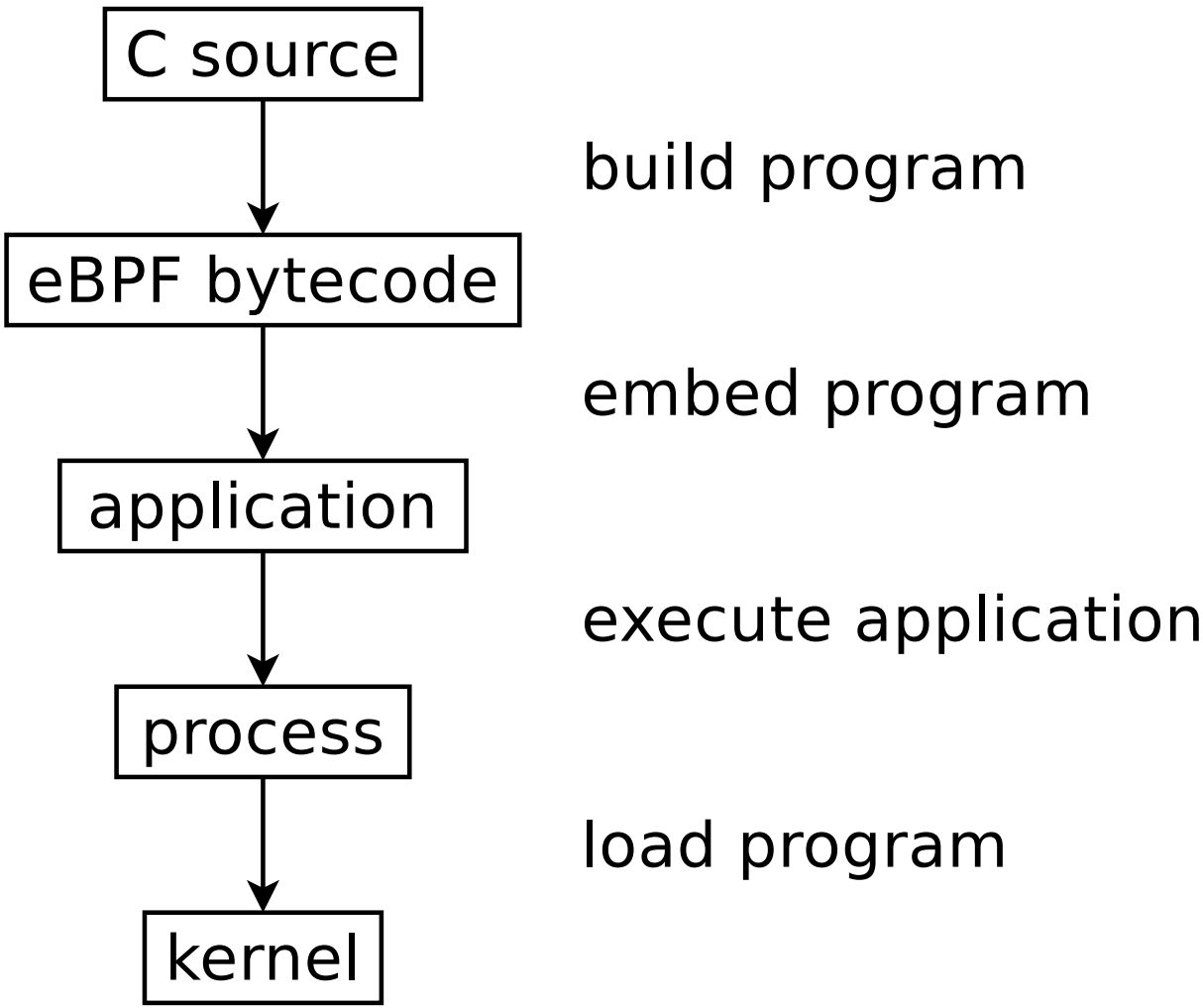
LSM framework

- allow or deny user-space actions on kernel objects
- policy decision and enforcement points
- kernel API: support various security models
- 200+ hooks: `inode_permission`, `inode_unlink`, `file_ioctl`...

Landlock

- hook: set of actions on a specific kernel object (e.g. walk a file path)
- program: access-control checks stacked on a hook
- triggers: actions mask for which a program is run (e.g. read, write, execute, remove, IOCTL...)

Life cycle of a Landlock program



Landlock rule example

Goal

- whitelist of file hierarchies for read-only or write access
- enforced on each file system access request for a set of processes

Source code

`https://landlock.io` ⇒ FOSDEM 2018

eBPF inode map

Goal

restrict access to a subset of the filesystem

eBPF inode map

Goal

restrict access to a subset of the filesystem

Challenges

- efficient
- updatable from user-space
- unprivileged use (i.e. no xattr)

eBPF inode map

Goal

restrict access to a subset of the filesystem

Challenges

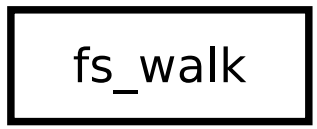
- efficient
- updatable from user-space
- unprivileged use (i.e. no xattr)

Solution

- new eBPF map type to identify an inode object (device + inode number)
- use inode as key and associate it with a 64-bits arbitrary value

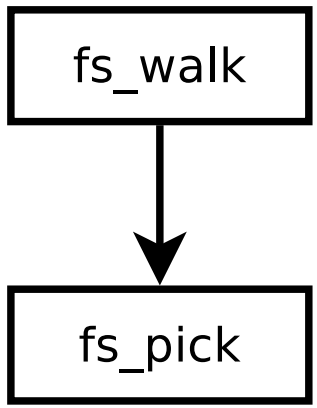
Chained programs and session

Landlock programs and their triggers (example)



Chained programs and session

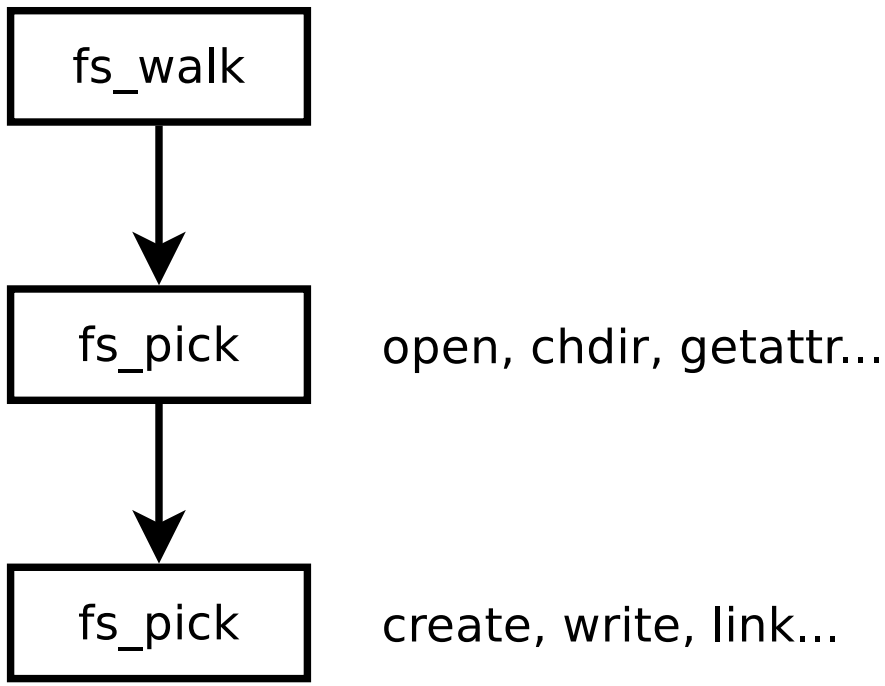
Landlock programs and their triggers (example)



open, chdir, getattr...

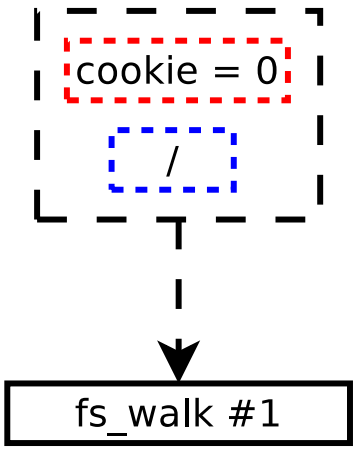
Chained programs and session

Landlock programs and their triggers (example)



Walking through a file path

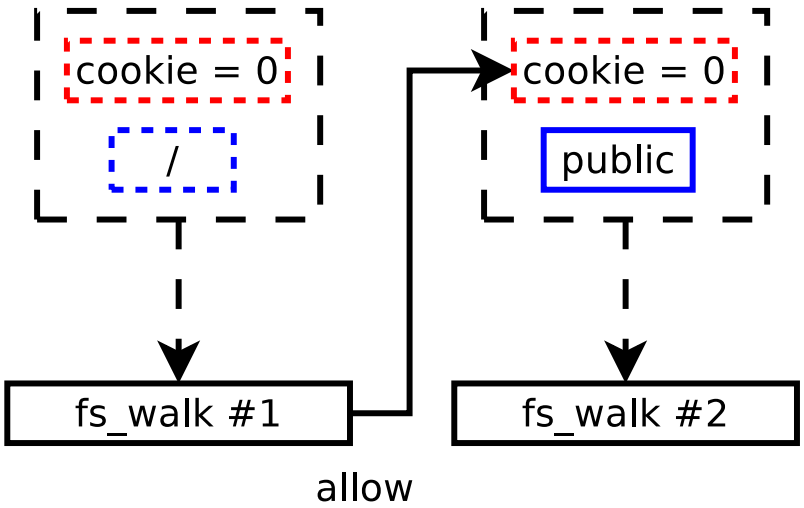
Example: `open /public/web/index.html`



key	value
/etc	1 (ro)
/public	1 (ro)
/tmp	2 (rw)

Walking through a file path

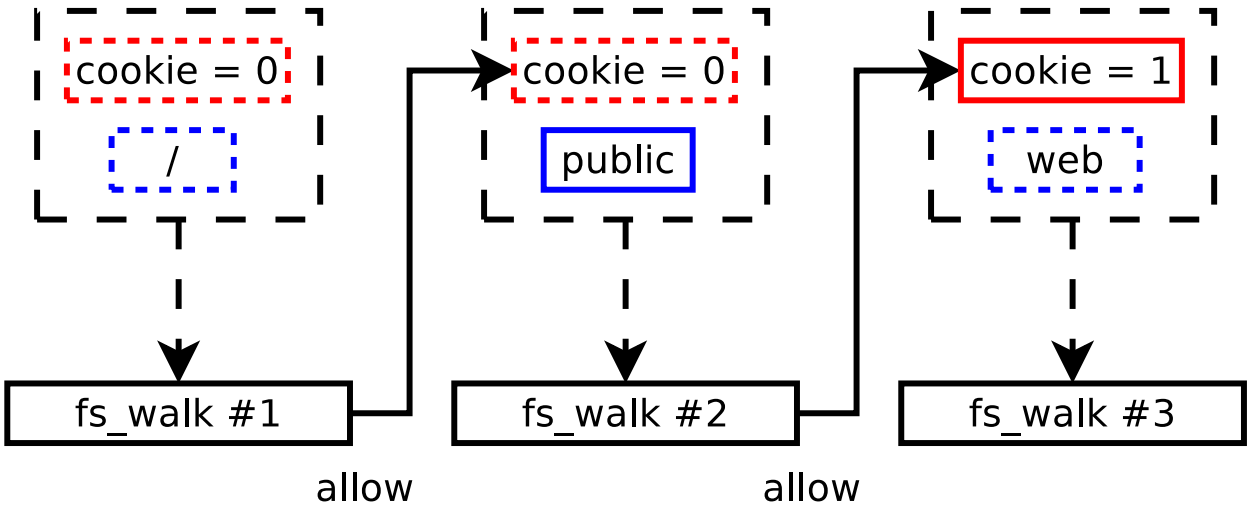
Example: `open /public/web/index.html`



key	value
/etc	1 (ro)
/public	1 (ro)
/tmp	2 (rw)

Walking through a file path

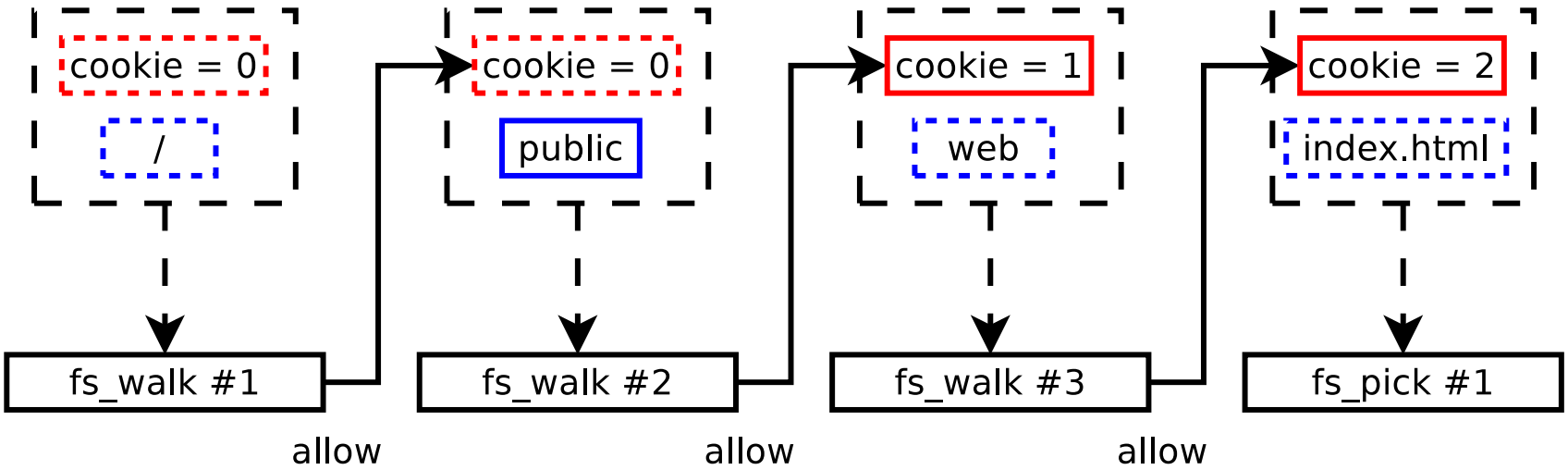
Example: `open /public/web/index.html`



key	value
/etc	1 (ro)
/public	1 (ro)
/tmp	2 (rw)

Walking through a file path

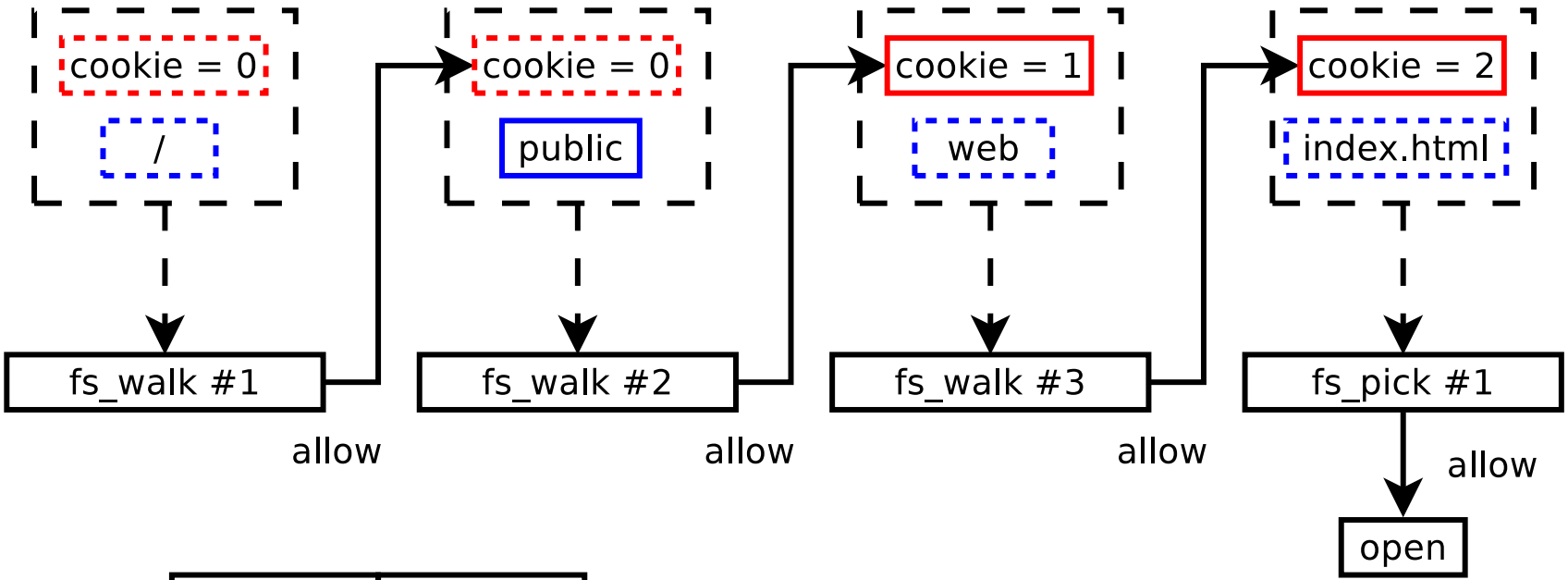
Example: `open /public/web/index.html`



key	value
/etc	1 (ro)
/public	1 (ro)
/tmp	2 (rw)

Walking through a file path

Example: `open /public/web/index.html`



key	value
/etc	1 (ro)
/public	1 (ro)
/tmp	2 (rw)

Landlock program's metadata

```
1  static union bpf_prog_subtype metadata = {
2          .landlock_hook = {
3                  .type = LANDLOCK_HOOK_FS_PICK,
4                  .options = LANDLOCK_OPTION_PREVIOUS,
5                  .previous = 2, /* landlock2 */
6                  .triggers = LANDLOCK_TRIGGER_FS_PICK_APPEND | \
7                              LANDLOCK_TRIGGER_FS_PICK_CREATE | \
8                              // [...]
9                              LANDLOCK_TRIGGER_FS_PICK_WRITE,
10         }
11 };
```

Landlock program's metadata

```
1  static union bpf_prog_subtype metadata = {
2          .landlock_hook = {
3                  .type = LANDLOCK_HOOK_FS_PICK,
4                  .options = LANDLOCK_OPTION_PREVIOUS,
5                  .previous = 2, /* landlock2 */
6                  .triggers = LANDLOCK_TRIGGER_FS_PICK_APPEND | \
7                              LANDLOCK_TRIGGER_FS_PICK_CREATE | \
8                              // [...]
9                              LANDLOCK_TRIGGER_FS_PICK_WRITE,
10         }
11 };
```

Landlock program's metadata

```
1  static union bpf_prog_subtype metadata = {
2          .landlock_hook = {
3                  .type = LANDLOCK_HOOK_FS_PICK,
4                  .options = LANDLOCK_OPTION_PREVIOUS,
5                  .previous = 2, /* landlock2 */
6                  .triggers = LANDLOCK_TRIGGER_FS_PICK_APPEND | \
7                              LANDLOCK_TRIGGER_FS_PICK_CREATE | \
8                              // [...]
9                              LANDLOCK_TRIGGER_FS_PICK_WRITE,
10         }
11 };
```

Landlock program's metadata

```
1  static union bpf_prog_subtype metadata = {
2          .landlock_hook = {
3                  .type = LANDLOCK_HOOK_FS_PICK,
4                  .options = LANDLOCK_OPTION_PREVIOUS,
5                  .previous = 2, /* landlock2 */
6                  .triggers = LANDLOCK_TRIGGER_FS_PICK_APPEND | \
7                              LANDLOCK_TRIGGER_FS_PICK_CREATE | \
8                              // [...]
9                              LANDLOCK_TRIGGER_FS_PICK_WRITE,
10         }
11 };
```

Landlock program's metadata

```
1  static union bpf_prog_subtype metadata = {
2          .landlock_hook = {
3                  .type = LANDLOCK_HOOK_FS_PICK,
4                  .options = LANDLOCK_OPTION_PREVIOUS,
5                  .previous = 2, /* landlock2 */
6                  .triggers = LANDLOCK_TRIGGER_FS_PICK_APPEND | \
7                              LANDLOCK_TRIGGER_FS_PICK_CREATE | \
8                              // [...]
9                              LANDLOCK_TRIGGER_FS_PICK_WRITE,
10         }
11 };
```

Landlock program's metadata

```
1  static union bpf_prog_subtype metadata = {
2          .landlock_hook = {
3                  .type = LANDLOCK_HOOK_FS_PICK,
4                  .options = LANDLOCK_OPTION_PREVIOUS,
5                  .previous = 2, /* landlock2 */
6                  .triggers = LANDLOCK_TRIGGER_FS_PICK_APPEND | \
7                              LANDLOCK_TRIGGER_FS_PICK_CREATE | \
8                              // [...]
9                              LANDLOCK_TRIGGER_FS_PICK_WRITE,
10         }
11 };
```

Landlock program code

```
1 int fs_pick_write(struct landlock_ctx_fs_pick *ctx) {
2         __u64 cookie = ctx->cookie;
3
4         cookie = update_cookie(cookie, ctx->inode_lookup,
5                                (void *)ctx->inode);
6         if (cookie & MAP_MARK_WRITE)
7                 return LANDLOCK_RET_ALLOW;
8         return LANDLOCK_RET_DENY;
9 }
```

Landlock program code

```
1  int fs_pick_write(struct landlock_ctx_fs_pick *ctx) {
2          __u64 cookie = ctx->cookie;
3
4          cookie = update_cookie(cookie, ctx->inode_lookup,
5                                 (void *)ctx->inode);
6          if (cookie & MAP_MARK_WRITE)
7                  return LANDLOCK_RET_ALLOW;
8          return LANDLOCK_RET_DENY;
9  }
```

Landlock program code

```
1  int fs_pick_write(struct landlock_ctx_fs_pick *ctx) {
2          __u64 cookie = ctx->cookie;
3
4          cookie = update_cookie(cookie, ctx->inode_lookup,
5                                 (void *)ctx->inode);
6          if (cookie & MAP_MARK_WRITE)
7                  return LANDLOCK_RET_ALLOW;
8          return LANDLOCK_RET_DENY;
9  }
```

Landlock program code

```
1 int fs_pick_write(struct landlock_ctx_fs_pick *ctx) {
2         __u64 cookie = ctx->cookie;
3
4         cookie = update_cookie(cookie, ctx->inode_lookup,
5                                (void *)ctx->inode);
6         if (cookie & MAP_MARK_WRITE)
7                 return LANDLOCK_RET_ALLOW;
8         return LANDLOCK_RET_DENY;
9 }
```

Landlock program code

```
1  int fs_pick_write(struct landlock_ctx_fs_pick *ctx) {
2          __u64 cookie = ctx->cookie;
3
4          cookie = update_cookie(cookie, ctx->inode_lookup,
5                                 (void *)ctx->inode);
6          if (cookie & MAP_MARK_WRITE)
7                  return LANDLOCK_RET_ALLOW;
8          return LANDLOCK_RET_DENY;
9  }
```

Landlock program code

```
1  int fs_pick_write(struct landlock_ctx_fs_pick *ctx) {
2          __u64 cookie = ctx->cookie;
3
4          cookie = update_cookie(cookie, ctx->inode_lookup,
5                                 (void *)ctx->inode);
6          if (cookie & MAP_MARK_WRITE)
7                  return LANDLOCK_RET_ALLOW;
8          return LANDLOCK_RET_DENY;
9  }
```

Loading a rule in the kernel

```
union bpf_attr attr = {
        .insns = bytecode_array,
        .prog_type = BPF_PROG_TYPE_LANDLOCK_HOOK,
        .prog_subtype = &metadata,
        // [...]
};
int prog_fd = bpf(BPF_PROG_LOAD, &attr, sizeof(attr));
```

Loading a rule in the kernel

```
1 union bpf_attr attr = {
2         .insns = bytecode_array,
3         .prog_type = BPF_PROG_TYPE_LANDLOCK_HOOK,
4         .prog_subtype = &metadata,
5         // [...]
6 };
7 int prog_fd = bpf(BPF_PROG_LOAD, &attr, sizeof(attr));
```

Loading a rule in the kernel

```
1 union bpf_attr attr = {
2         .insns = bytecode_array,
3         .prog_type = BPF_PROG_TYPE_LANDLOCK_HOOK,
4         .prog_subtype = &metadata,
5         // [...]
6 };
7 int prog_fd = bpf(BPF_PROG_LOAD, &attr, sizeof(attr));
```

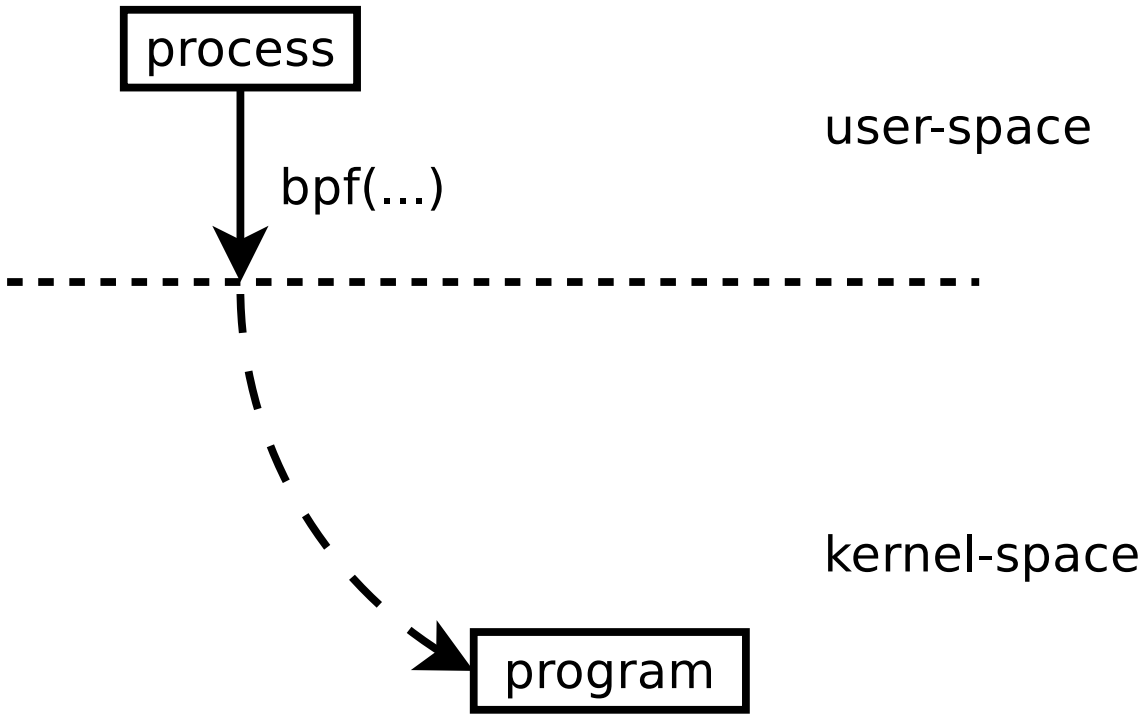
Loading a rule in the kernel

```
1 union bpf_attr attr = {
2         .insns = bytecode array,
3         .prog_type = BPF_PROG_TYPE_LANDLOCK_HOOK,
4         .prog_subtype = &metadata,
5         // [...]
6 };
7 int prog_fd = bpf(BPF_PROG_LOAD, &attr, sizeof(attr));
```

Loading a rule in the kernel

```
1 union bpf_attr attr = {
2         .insns = bytecode_array,
3         .prog_type = BPF_PROG_TYPE_LANDLOCK_HOOK,
4         .prog_subtype = &metadata,
5         // [...]
6 };
7 int prog_fd = bpf(BPF_PROG_LOAD, &attr, sizeof(attr));
```

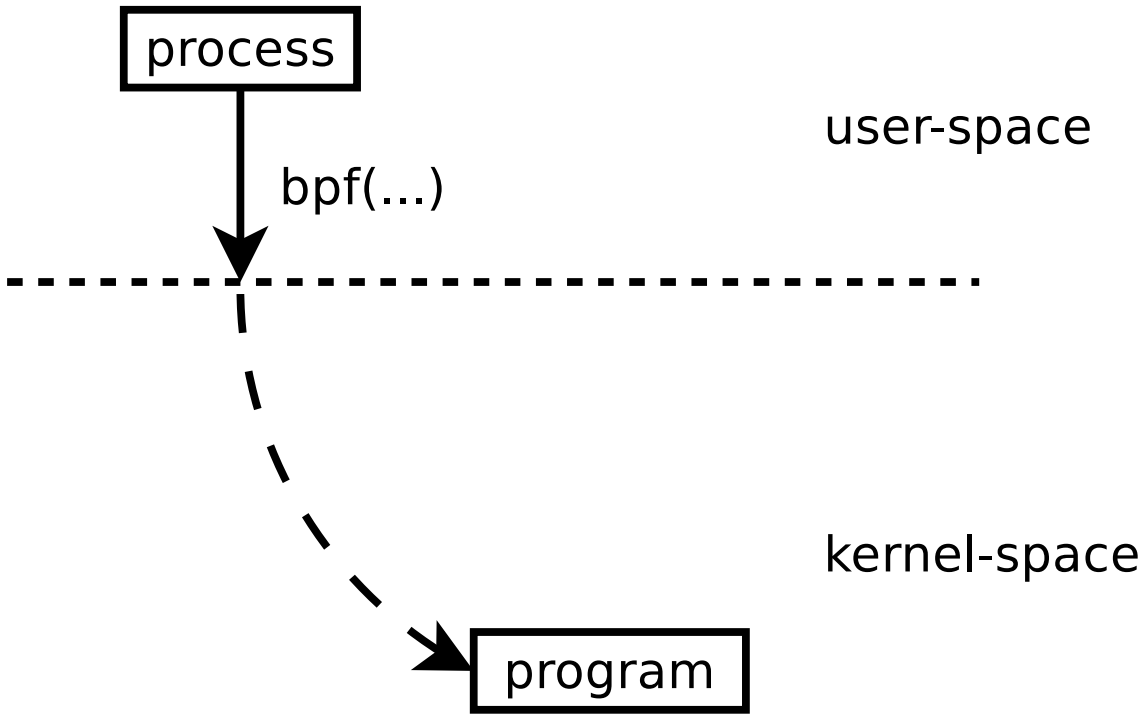
Loading a rule in the kernel



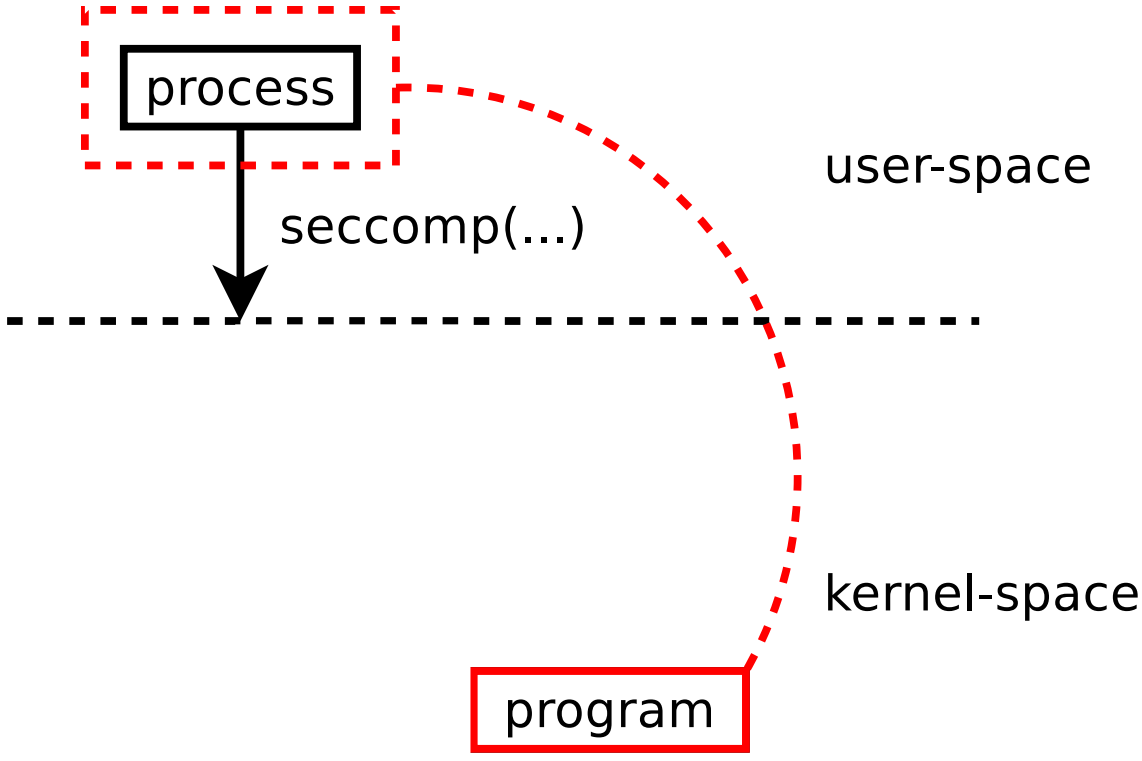
Applying a Landlock program to a process

```
1 seccomp(SECCOMP_PREPEND_LANDLOCK_PROG, 0, &prog_fd);
```

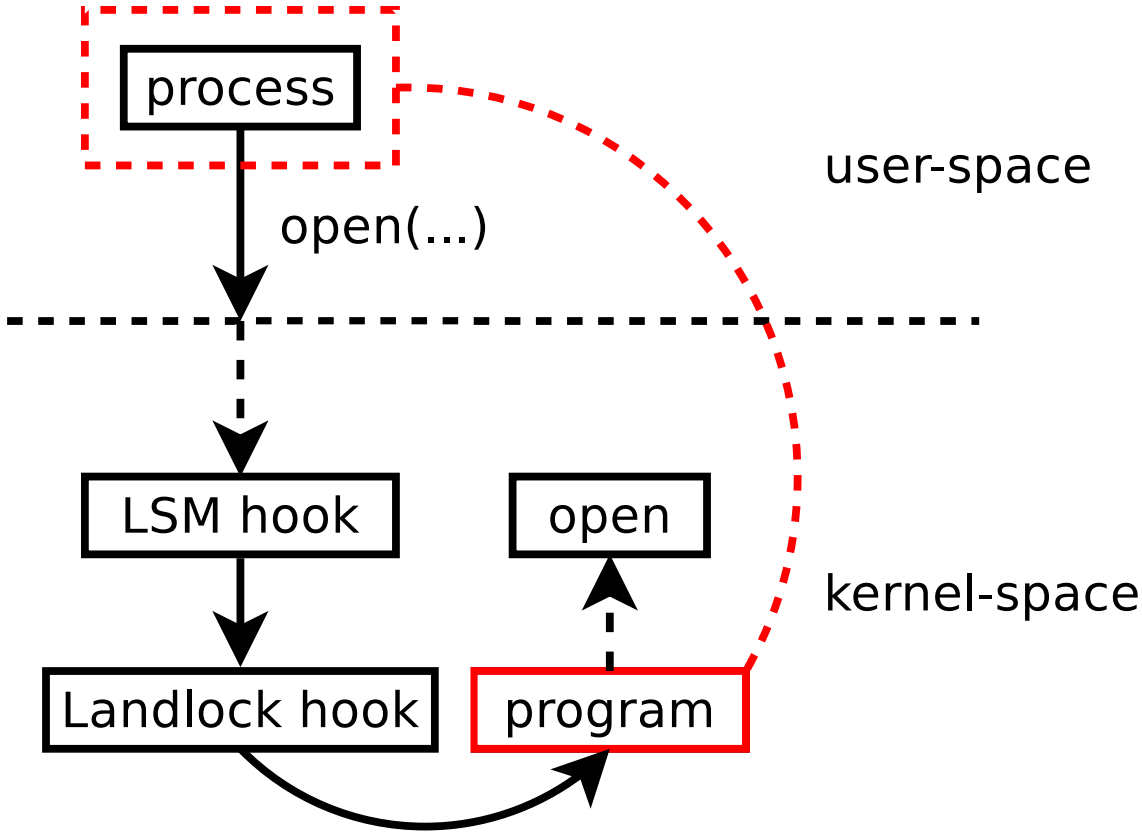
Applying a Landlock program to a process



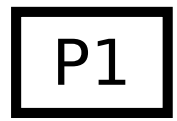
Applying a Landlock program to a process



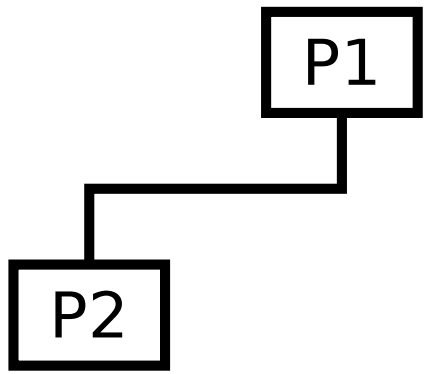
Applying a Landlock program to a process



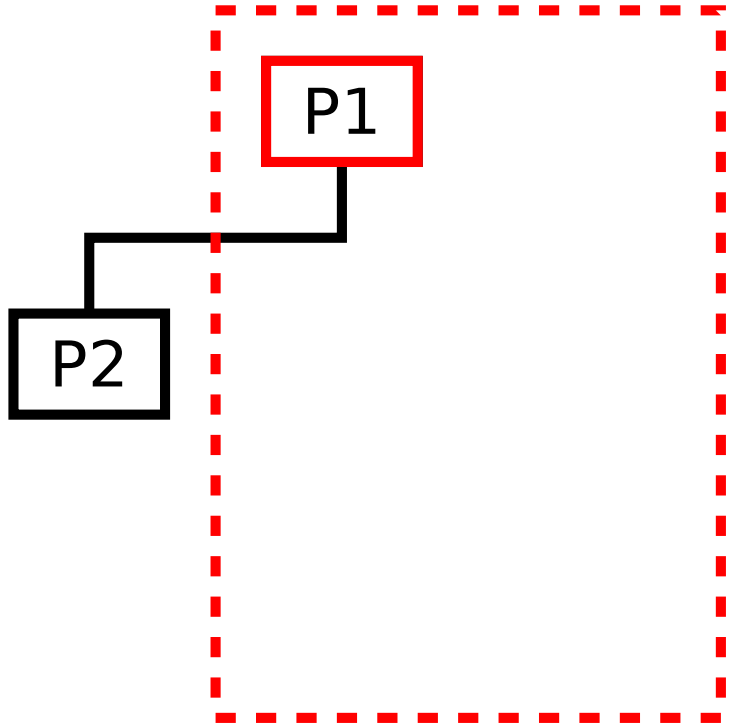
Rule enforcement on process hierarchy



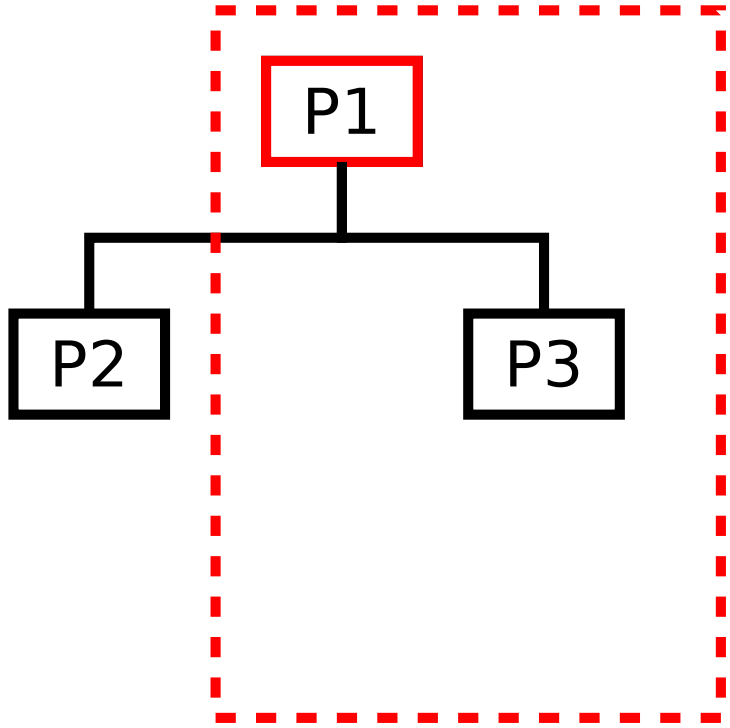
Rule enforcement on process hierarchy



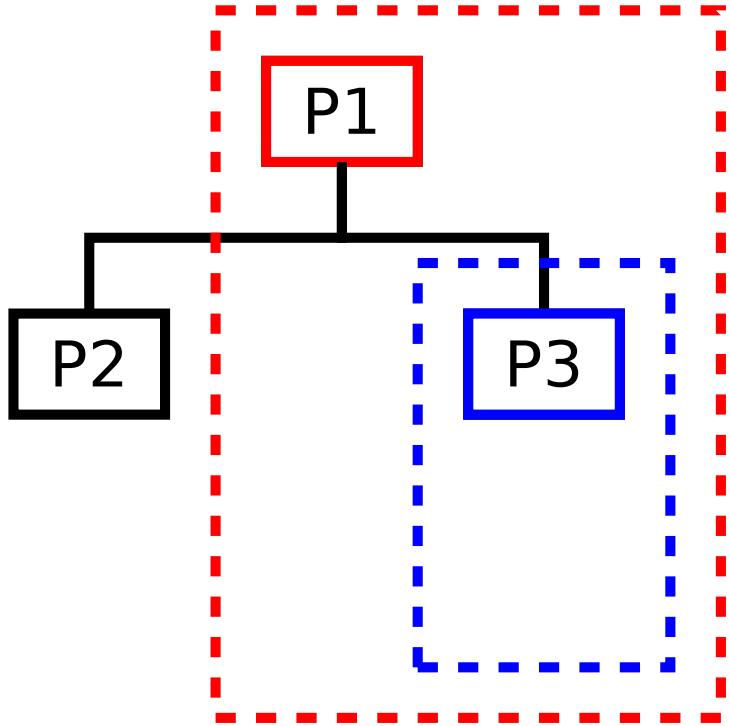
Rule enforcement on process hierarchy



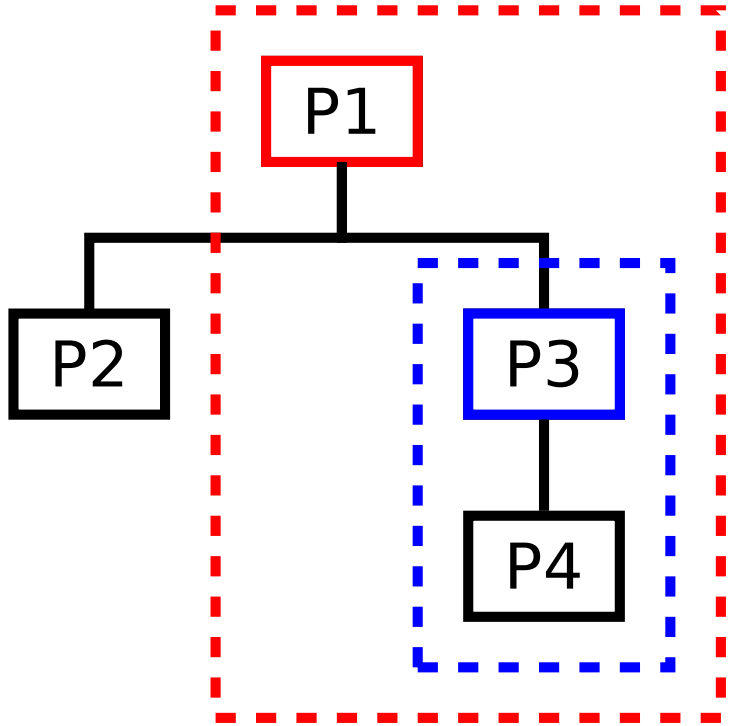
Rule enforcement on process hierarchy



Rule enforcement on process hierarchy



Rule enforcement on process hierarchy



Demonstration

Read-only accesses...

- `/public`
- `/etc`
- `/usr`
- `...`

...and read-write accesses

- `/proc/self/fd/1`
- `/tmp`
- `...`

Landlock: wrap-up

User-space hardening

- programmatic and embeddable access control
- dynamic security policy
- designed for unprivileged use

Landlock: wrap-up

User-space hardening

- programmatic and embeddable access control
- dynamic security policy
- designed for unprivileged use

Current status

- autonomous patches merged in net/bpf, security and kselftest trees
- `security/landlock/*`: ∼1600 SLOC
- ongoing patch series: LKML, `@l0kod`
- stay tuned for the final v8 in the following weeks

https://landlock.io

Unprivileged access control

Why?

embed a security policy in any application, following the least privilege principle

Unprivileged access control

Why?

embed a security policy in any application, following the least privilege principle

Challenges

- applying a security policy requires privileges
- unlike SUID, Landlock should only reduce accesses
- prevent accesses through other processes: ptrace restrictions
- protect the kernel: eBPF static analysis
- prevent information leak: an eBPF program shall not have more access rights than the process which loaded it

Enforcement through cgroups

Why?

user/admin security policy (e.g. container): manage groups of processes

Enforcement through cgroups

Why?

user/admin security policy (e.g. container): manage groups of processes

Challenges

- complementary to the process hierarchy rules (via *seccomp(2)*)
- processes moving in or out of a cgroup
- unprivileged use with cgroups delegation (e.g. user session)

Future Landlock program types

fs_get

tag inodes: needed for relative path checks (e.g. *openat(2)*)

Future Landlock program types

fs_get

tag inodes: needed for relative path checks (e.g. *openat(2)*)

fs_ioctl

check IOCTL commands

Future Landlock program types

fs_get

tag inodes: needed for relative path checks (e.g. *openat(2)*)

fs_ioctl

check IOCTL commands

net_*

check IPs, ports, protocol...