Internals of Landlock: a new kind of Linux Security Module leveraging eBPF

Mickaël Salaün

ANSSI

July 4, 2018
Protect users from your application

Threat

1. bug exploitation of your code
2. bug or backdoor in a third party component
⇒ your application is used against your will
Protect users from your application

Threat

1. bug exploitation of your code
2. bug or backdoor in a third party component
⇒ your application is used against your will

Defenses

▶ follow secure development practices
▶ use an hardened toolchain
▶ use OS security features (e.g. sandboxes)
Protect users from your application

Threat

1. bug exploitation of your code
2. bug or backdoor in a third party component
⇒ your application is used against your will

Defenses

▶ follow secure development practices
▶ use an hardened toolchain
▶ use OS security features (e.g. sandboxes)

The Landlock features

▶ help define and embed security policy in your code
▶ enforce an access control on your application
Demonstration #1

Read-only accesses...

▶ /public
▶ /etc
▶ /usr
▶ ...

...and read-write accesses

▶ /tmp
▶ ...

What about the other Linux security features?

<table>
<thead>
<tr>
<th>Fine-grained control</th>
<th>Embedded policy</th>
<th>Unprivileged use</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELinux...</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

Disabled on purpose for the initial upstream inclusion, but planned to be enabled after a test period.
What about the other Linux security features?

<table>
<thead>
<tr>
<th></th>
<th>Fine-grained control</th>
<th>Embedded policy</th>
<th>Unprivileged use</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELinux...</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>seccomp-bpf</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>namespaces</td>
<td>✓</td>
<td></td>
<td>~</td>
</tr>
</tbody>
</table>
What about the other Linux security features?

<table>
<thead>
<tr>
<th></th>
<th>Fine-grained control</th>
<th>Embedded policy</th>
<th>Unprivileged use</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELinux...</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>seccomp-bpf</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>namespaces</td>
<td>✓</td>
<td>✓</td>
<td>~</td>
</tr>
<tr>
<td>Landlock</td>
<td>✓</td>
<td>✓</td>
<td>✓(^1)</td>
</tr>
</tbody>
</table>

Tailored access control to match your needs: programmatic access control

\(^1\)Disabled on purpose for the initial upstream inclusion, but planned to be enabled after a test period.
Landlock overview

process

user-space

call open(...)

restrict

Landlock programs

kernel-space

open
extended Berkeley Packet Filter

In-kernel virtual machine

- safely execute code in the kernel at run time
- widely used in the kernel: network filtering (XDP), 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 (XDP), 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...)
Safely handle malicious policies

- Landlock should be usable by everyone
- We can’t tell if a process will be malicious
  ⇒ trust issue
Unprivileged access control

Sought properties

▶ multiple applications, need independent but composable security policies
▶ tamper proof: prevent bypass through other processes (i.e. via ptrace)
Unprivileged access control

Sought properties

- multiple applications, need independant but composable security policies
- tamper proof: prevent bypass through other processes (i.e. via ptrace)

Harmlessness

- safe approach: follow the least privilege principle (i.e. no SUID)
- limit the kernel attack surface:
  - minimal kernel code (security/landlock/*: ~2000 SLOC)
  - eBPF static analysis
  - move complexity from the kernel to eBPF programs
Unprivileged access control

Protect access to process resources

- the rule creator must be allowed to ptrace the sandboxed process
Unprivileged access control

Protect access to process resources
- the rule creator must be allowed to ptrace the sandboxed process

Protect access to kernel resources
- prevent information leak: an eBPF program shall not have more access rights than the process which loaded it
- still, access control need some knowledge to take decision (e.g. file path check)
- only interpreted on viewable objects and after other access controls
Identifying a file path

- path evaluation based on walking through inodes
- multiple Landlock program types
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:
  ▸ no xattr
  ▸ no absolute path
eBPF inode map

Goal
restrict access to a subset of the filesystem

Challenges
▶ efficient
▶ updatable from user-space
▶ unprivileged use:
  ▶ no xattr
  ▶ no absolute path

Solution
▶ new eBPF map type to identify an inode object
▶ use inode as key and associate it with a 64-bits arbitrary value
Demonstration #2

Update access rights on the fly
Chained programs and session

Landlock programs and their triggers (example)

fs_walk
Chained programs and session

Landlock programs and their triggers (example)

- fs_walk
- fs_pick
  - open, chdir, getattr...
Chained programs and session

Landlock programs and their triggers (example)

```
fs_walk
  ↓
fs_pick
    ↓
fs_pick
```

- open, chdir, getattr...
- create, write, link...
Walking through a file path

Example: open /public/web/index.html

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>/etc</td>
<td>1 (ro)</td>
</tr>
<tr>
<td>/public</td>
<td>1 (ro)</td>
</tr>
<tr>
<td>/tmp</td>
<td>2 (rw)</td>
</tr>
</tbody>
</table>
Walking through a file path

Example: open /public/web/index.html

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>/etc</td>
<td>1 (ro)</td>
</tr>
<tr>
<td>/public</td>
<td>1 (ro)</td>
</tr>
<tr>
<td>/tmp</td>
<td>2 (rw)</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>/etc</td>
<td>1 (ro)</td>
</tr>
<tr>
<td>/public</td>
<td>1 (ro)</td>
</tr>
<tr>
<td>/tmp</td>
<td>2 (rw)</td>
</tr>
</tbody>
</table>
Walking through a file path

Example: open /public/web/index.html

```
<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>/etc</td>
<td>1 (ro)</td>
</tr>
<tr>
<td>/public</td>
<td>1 (ro)</td>
</tr>
<tr>
<td>/tmp</td>
<td>2 (rw)</td>
</tr>
</tbody>
</table>
```
Walking through a file path

Example: open /public/web/index.html
From the rule to the kernel

- writing a Landlock rule
- loading it in the kernel
- enforcing it on a set of processes
Life cycle of a Landlock program

1. C source
2. eBPF bytecode
3. application
4. process
5. kernel

- build program
- embed program
- execute application
- load program
Landlock program’s metadata

```c
static union bpf_prog_subtype metadata = {
    .landlock_hook = {
        .type = LANDLOCK_HOOK_FS_PICK,
        .options = LANDLOCK_OPTION_PREVIOUS,
        .previous = 2, /* landlock2 */
        .triggers = LANDLOCK_TRIGGER_FS_PICK_APPEND | \
                    LANDLOCK_TRIGGER_FS_PICK_CREATE | \
                    // [...] \n                    LANDLOCK_TRIGGER_FS_PICK_WRITE,
    }
};
```
static union bpf_prog_subtype metadata = {
    .landlock_hook = {
        .type = LANDLOCK_HOOK_FS_PICK,
        .options = LANDLOCK_OPTION_PREVIOUS,
        .previous = 2, /* landlock2 */
        .triggers = LANDLOCK_TRIGGER_FS_PICK_APPEND | \ 
                   LANDLOCK_TRIGGER_FS_PICK_CREATE | \ 
                   // [...] \ 
                   LANDLOCK_TRIGGER_FS_PICK_WRITE,
    }
};
Landlock program’s metadata

```c
static union bpf_prog_subtype metadata = {
    .landlock_hook = {
        .type = LANDLOCK_HOOK_FS_PICK,
        .options = LANDLOCK_OPTION_PREVIOUS,
        .previous = 2, /* landlock2 */
        .triggers = LANDLOCK_TRIGGER_FS_PICK_APPEND | \
            LANDLOCK_TRIGGER_FS_PICK_CREATE | \
            // [...]
            LANDLOCK_TRIGGER_FS_PICK_WRITE,
    }
};
```
Landlock program’s metadata

```c
static union bpf_prog_subtype metadata = {
    .landlock_hook = {
        .type = LANDLOCK_HOOK_FS_PICK,
        .options = LANDLOCK_OPTION_PREVIOUS,
        .previous = 2, /* landlock2 */
        .triggers = LANDLOCK_TRIGGER_FS_PICK_APPEND | \
                   LANDLOCK_TRIGGER_FS_PICK_CREATE | \
                   // [...]
                   LANDLOCK_TRIGGER_FS_PICK_WRITE,
    }
};
```
static union bpf_prog_subtype metadata = {
    .landlock_hook = {
        .type = LANDLOCK_HOOK_FS_PICK,
        .options = LANDLOCK_OPTION_PREVIOUS,
        .previous = 2, /* landlock2 */
        .triggers = LANDLOCK_TRIGGER_FS_PICK_APPEND | \ 
                    LANDLOCK_TRIGGER_FS_PICK_CREATE | \ 
                    // [...] 
                    LANDLOCK_TRIGGER_FS_PICK_WRITE,
    }
};
static union bpf_prog_subtype metadata = {
    .landlock_hook = {
        .type = LANDLOCK_HOOK_FS_PICK,
        .options = LANDLOCKgetOption_PREVIOUS,
        .previous = 2, /* landlock2 */
        .triggers = LANDLOCK_TRIGGER_FS_PICK_APPEND | \ 
        LANDLOCK_TRIGGER_FS_PICK_CREATE | \ 
        // [...]
        LANDLOCK_TRIGGER_FS_PICK_WRITE,
    }
};
Landlock program code

```c
int fs_pick_write(struct landlock_ctx_fs_pick *ctx) {
    __u64 cookie = ctx->cookie;

    cookie = update_cookie(cookie, ctx->inode_lookup, (void *)ctx->inode);

    if (cookie & MAP_MARK_WRITE)
        return LANDLOCK_RET_ALLOW;

    return LANDLOCK_RET_DENY;
}
```
```c
int fs_pick_write(struct landlock_ctx_fs_pick *ctx) {
    __u64 cookie = ctx->cookie;
    cookie = update_cookie(cookie, ctx->inode_lookup,
                           (void *)ctx->inode);
    if (cookie & MAP_MARK_WRITE)
        return LANDLOCK_RET_ALLOW;
    return LANDLOCK_RET_DENY;
}
```
```c
int fs_pick_write(struct landlock_ctx_fs_pick *ctx) {
    __u64 cookie = ctx->cookie;

    cookie = update_cookie(cookie, ctx->inode_lookup,
                            (void *)ctx->inode);

    if (cookie & MAP_MARK_WRITE)
        return LANDLOCK_RET_ALLOW;

    return LANDLOCK_RET_DENY;
}
```
```c
int fs_pick_write(struct landlock_ctx_fs_pick *ctx) {
    __u64 cookie = ctx->cookie;

    cookie = update_cookie(cookie, ctx->inode_lookup,
                           (void *)ctx->inode);

    if (cookie & MAP_MARK_WRITE)
        return LANDLOCK_RET_ALLOW;

    return LANDLOCK_RET_DENY;
}
```
```c
int fs_pick_write(struct landlock_ctx_fs_pick *ctx) {
    __u64 cookie = ctx->cookie;

    cookie = update_cookie(cookie, ctx->inode_lookup,
                            (void *)ctx->inode);

    if (cookie & MAP_MARK_WRITE)
        return LANDLOCK_RET_ALLOW;

    return LANDLOCK_RET_DENY;
}
```
int fs_pick_write(struct landlock_ctx_fs_pick *ctx) {
    __u64 cookie = ctx->cookie;

    cookie = update_cookie(cookie, ctx->inode_lookup, (void *)ctx->inode);

    if (cookie & MAP_MARK_WRITE)
        return LANDLOCK_RET_ALLOW;

    return LANDLOCK_RET_DENY;
}
Loading a rule in the kernel

```c
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

```c
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

```c
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

```c
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

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

process

bpf(...)

user-space

kernel-space

program
Applying a Landlock program to a process

1 | seccomp(SECCOMP_PREPEND_LANDLOCK_PROG, 0, &prog_fd);
Applying a Landlock program to a process

```
process
  bpf(...) ----> program
```

user-space

kernel-space
Applying a Landlock program to a process

```
process

seccomp(...)
```

user-space

kernel-space

program
Applying a Landlock program to a process

```
process
  ↓
open(...)  
  ↓
LSM hook
  ↓
Landlock hook
  ↓
program
```

user-space

kernel-space
Kernel execution flow

Example: the `inode_create` hook

1. check if landlocked(current)
2. call `decide_fs_pick(LANDLOCK_TRIGGER_FS_PICK_CREATE, dir)`
3. for all `fs_pick` programs enforced on the current process
   3.1 update the program’s context
   3.2 interpret the program
   3.3 continue until one denies the access
Landlock: wrap-up

User-space hardening

- programmatic and embeddable access control
- designed for unprivileged use
- apply tailored access controls per process
- make it evolve over time (map)
Landlock: wrap-up

User-space hardening

- programmatic and embeddable access control
- designed for unprivileged use
- apply tailored access controls per process
- make it evolve over time (map)

Current status

- standalone patches merged in net/bpf, security and kselftest trees
- security/landlock/*/: ~2000 SLOC
- ongoing patch series: LKML, @l0kd
- full security module stacking is coming!
https://landlock.io
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
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...