Landlock: unprivileged access control

Author: MickaÃ«l SalaÃ¼n
Date: March 2021

The goal of Landlock is to enable to restrict ambient rights (e.g. global filesystem access) for a set of processes. Because Landlock is a stackable LSM, it makes possible to create safe security sandboxes as new security layers in addition to the existing system-wide access-controls. This kind of sandbox is expected to help mitigate the security impact of bugs or unexpected/malicious behaviors in user space applications. Landlock empowers any process, including unprivileged ones, to securely restrict themselves.

Landlock rules

A Landlock rule describes an action on an object. An object is currently a file hierarchy, and the related filesystem actions are defined with access rights. A set of rules is aggregated in a ruleset, which can then restrict the thread enforcing it, and its future children.

Defining and enforcing a security policy

We first need to create the ruleset that will contain our rules. For this example, the ruleset will contain rules that only allow read actions, but write actions will be denied. The ruleset then needs to handle both of these kind of actions.

```
int ruleset_fd;
struct landlock_ruleset_attr ruleset_attr = {
    .handled_access_fs =
        LANDLOCK_ACCESS_FS_EXECUTE |
        LANDLOCK_ACCESS_FS_WRITE_FILE |
        LANDLOCK_ACCESS_FS_READ_FILE |
        LANDLOCK_ACCESS_FS_READ_DIR |
        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,
};

ruleset_fd = landlock_create_ruleset(&ruleset_attr, sizeof(ruleset_attr), 0);
if (ruleset_fd < 0) {
    perror("Failed to create a ruleset");
    return 1;
}
```

We can now add a new rule to this ruleset thanks to the returned file descriptor referring to this ruleset. The rule will only allow reading the file hierarchy `/usr`. Without another rule, write actions would then be denied by the ruleset. To add `/usr` to the ruleset, we open it with the `O_PATH` flag and fill the &struct landlock_path_beneath_attr with this file descriptor.

```
int err;
struct landlock_path_beneath_attr path_beneath = {
    .allowed_access =
        LANDLOCK_ACCESS_FS_EXECUTE |
        LANDLOCK_ACCESS_FS_READ_FILE |
        LANDLOCK_ACCESS_FS_READ_DIR,
};

path_beneath.parent_fd = open("/usr", O_PATH | O_CLOEXEC);
if (path_beneath.parent_fd < 0) {
    perror("Failed to open file");
    close(ruleset_fd);
    return 1;
}
err = landlock_add_rule(ruleset_fd, LANDLOCK_RULE_PATH_BENEATH,
                        &path_beneath, 0);
close(path_beneath.parent_fd);
if (err) {
    perror("Failed to update ruleset");
    close(ruleset_fd);
    return 1;
}
```

We now have a ruleset with one rule allowing read access to `/usr` while denying all other handled accesses for the filesystem. The next step is to restrict the current thread from gaining more privileges (e.g. thanks to a SUID binary).

```
if (prctl(PR_SET_NO_NEW_PRIVS, 1, 0, 0, 0)) {
    perror("Failed to restrict privileges");
    close(ruleset_fd);
    return 1;
}
```

The current thread is now ready to sandbox itself with the ruleset.

```
if (landlock_restrict_self(ruleset_fd, 0)) {
    perror("Failed to enforce ruleset");
    close(ruleset_fd);
    return 1;
}
close(ruleset_fd);
```

If the *landlock_restrict_self* system call succeeds, the current thread is now restricted and this policy will be enforced on all its subsequently created children as well. Once a thread is landlocked, there is no way to remove its security policy; only adding more restrictions is allowed. These threads are now in a new Landlock domain, merge of their parent one (if any) with the new ruleset.

Full working code can be found in samples/landlock/sandboxer.c.

Layers of file path access rights

Each time a thread enforces a ruleset on itself, it updates its Landlock domain with a new layer of policy. Indeed, this complementary policy is stacked with the potentially other rulesets already restricting this thread. A sandboxed thread can then safely add more constraints to itself with a new enforced ruleset.

One policy layer grants access to a file path if at least one of its rules encountered on the path grants the access. A sandboxed thread can only access a file path if all its enforced policy layers grant the access as well as all the other system access controls (e.g. filesystem DAC, other LSM policies, etc.).

Bind mounts and OverlayFS

Landlock enables to restrict access to file hierarchies, which means that these access rights can be propagated with bind mounts (cf. Documentation/filesystems/sharedsubtree.rst) but not with Documentation/filesystems/overlayfs.rst.

A bind mount mirrors a source file hierarchy to a destination. The destination hierarchy is then composed of the exact same files, on which Landlock rules can be tied, either via the source or the destination path. These rules restrict access when they are encountered on a path, which means that they can restrict access to multiple file hierarchies at the same time, whether these hierarchies are the result of bind mounts or not.

An OverlayFS mount point consists of upper and lower layers. These layers are combined in a merge directory, result of the mount point. This merge hierarchy may include files from the upper and lower layers, but modifications performed on the merge hierarchy only reflects on the upper layer. From a Landlock policy point of view, each OverlayFS layers and merge hierarchies are standalone and contains their own set of files and directories, which is different from bind mounts. A policy restricting an OverlayFS layer will not restrict the resulted merged hierarchy, and vice versa. Landlock users should then only think about file hierarchies they want to allow access to, regardless of the underlying filesystem.

Inheritance

Every new thread resulting from a :manpage:`clone(2)` inherits Landlock domain restrictions from its parent. This is similar to the seccomp inheritance (cf. Documentation/userspace-api/seccomp_filter.rst) or any other LSM dealing with task's :manpage:`credentials(7)`. For instance, one process's thread may apply Landlock rules to itself, but they will not be automatically applied to other sibling threads (unlike POSIX thread credential changes, cf. :manpage:`nptl(7)`).

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\(linux-master)(Documentation)(userspace-api)landlock.rst`, line 172); ***backlink***

Unknown interpreted text role "manpage".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\(linux-master)(Documentation)(userspace-api)landlock.rst`, line 172); ***backlink***

Unknown interpreted text role "manpage".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\(linux-master)(Documentation)(userspace-api)landlock.rst`, line 172); ***backlink***

Unknown interpreted text role "manpage".

When a thread sandboxes itself, we have the guarantee that the related security policy will stay enforced on all this thread's descendants. This allows creating standalone and modular security policies per application, which will automatically be composed between themselves according to their runtime parent policies.

Ptrace restrictions

A sandboxed process has less privileges than a non-sandboxed process and must then be subject to additional restrictions when manipulating another process. To be allowed to use :manpage:`ptrace(2)` and related syscalls on a target process, a sandboxed process should have a subset of the target process rules, which means the tracee must be in a sub-domain of the tracer.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\(linux-master) (Documentation) (userspace-api)landlock.rst`, line 189); *backlink*

Unknown interpreted text role "manpage".

Kernel interface

Access rights

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\(linux-master) (Documentation) (userspace-api)landlock.rst`, line 201)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/uapi/linux/landlock.h
    :identifiers: fs_access
```

Creating a new ruleset

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\(linux-master) (Documentation) (userspace-api)landlock.rst`, line 207)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: security/landlock/syscalls.c
    :identifiers: sys_landlock_create_ruleset
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\(linux-master) (Documentation) (userspace-api)landlock.rst`, line 210)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/uapi/linux/landlock.h
    :identifiers: landlock_ruleset_attr
```

Extending a ruleset

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\(linux-master) (Documentation) (userspace-api)landlock.rst`, line 216)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: security/landlock/syscalls.c
    :identifiers: sys_landlock_add_rule
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\(linux-master) (Documentation) (userspace-api)landlock.rst`, line 219)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/uapi/linux/landlock.h
    :identifiers: landlock_rule_type landlock_path_beneath_attr
```

Enforcing a ruleset

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\(linux-master)(Documentation)(userspace-api)landlock.rst`, line 225)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: security/landlock/syscalls.c
    :identifiers: sys_landlock_restrict_self
```

Current limitations

File renaming and linking

Because Landlock targets unprivileged access controls, it is needed to properly handle composition of rules. Such property also implies rules nesting. Properly handling multiple layers of ruleset, each one of them able to restrict access to files, also implies to inherit the ruleset restrictions from a parent to its hierarchy. Because files are identified and restricted by their hierarchy, moving or linking a file from one directory to another implies to propagate the hierarchy constraints. To protect against privilege escalations through renaming or linking, and for the sake of simplicity, Landlock currently limits linking and renaming to the same directory. Future Landlock evolutions will enable more flexibility for renaming and linking, with dedicated ruleset flags.

Filesystem topology modification

As for file renaming and linking, a sandboxed thread cannot modify its filesystem topology, whether via :manpage:`mount(2)` or :manpage:`pivot_root(2)`. However, :manpage:`chroot(2)` calls are not denied.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\(linux-master)(Documentation)(userspace-api)landlock.rst`, line 249); *backlink*

Unknown interpreted text role "manpage".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\(linux-master)(Documentation)(userspace-api)landlock.rst`, line 249); *backlink*

Unknown interpreted text role "manpage".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\userspace-api\(linux-master)(Documentation)(userspace-api)landlock.rst`, line 249); *backlink*

Unknown interpreted text role "manpage".

Special filesystems

Access to regular files and directories can be restricted by Landlock, according to the handled accesses of a ruleset. However, files that do not come from a user-visible filesystem (e.g. pipe, socket), but can still be accessed through `/proc/<pid>/fd/*`, cannot currently be explicitly restricted. Likewise, some special kernel filesystems such as nsfs, which can be accessed through `/proc/<pid>/ns/*`, cannot currently be explicitly restricted. However, thanks to the ptrace restrictions, access to such sensitive `/proc` files are automatically restricted according to domain hierarchies. Future Landlock evolutions could still enable to explicitly restrict such paths with dedicated ruleset flags.

Ruleset layers

There is a limit of 64 layers of stacked rulesets. This can be an issue for a task willing to enforce a new ruleset in complement to its 64 inherited rulesets. Once this limit is reached, sys_landlock_restrict_self() returns E2BIG. It is then strongly suggested to carefully build rulesets once in the life of a thread, especially for applications able to launch other applications that may also want to sandbox themselves (e.g. shells, container managers, etc.).

Memory usage

Kernel memory allocated to create rulesets is accounted and can be restricted by the Documentation/admin-guide/cgroup-v1/memory.rst.

Questions and answers

What about user space sandbox managers?

Using user space process to enforce restrictions on kernel resources can lead to race conditions or inconsistent evaluations (i.e. Incorrect mirroring of the OS code and state).

What about namespaces and containers?

Namespaces can help create sandboxes but they are not designed for access-control and then miss useful features for such use case (e.g. no fine-grained restrictions). Moreover, their complexity can lead to security issues, especially when untrusted processes can manipulate them (cf. Controlling access to user namespaces).

Additional documentation

- Documentation/security/landlock.rst
- https://landlock.io