[RFC] Landlock LSM: Unprivileged sandboxing

Mickaël Salaün

September 29, 2016

Goal: restrict processes without needing root privileges

Examples

- files beneath a list of directories
- bind to a range of ports

Goal: restrict processes without needing root privileges

Examples

- files beneath a list of directories
- bind to a range of ports

Append restrictions

- stackable LSM
- global system view
- without SUID and complex brokers

Goal: restrict processes without needing root privileges

Examples

- files beneath a list of directories
- bind to a range of ports

Append restrictions

- stackable LSM
- global system view
- without SUID and complex brokers

What is concerned?

- applications with built-in sandboxing
- sandboxing managers

How do we use Landlock?

Process hierarchy (application)

1. create or receive Landlock rules
2. attach them to the current process via *seccomp(2)*

How do we use Landlock?

Process hierarchy (application)

1. create or receive Landlock rules
2. attach them to the current process via *seccomp(2)*

cgroup (container)

1. create Landlock rules
2. open a cgroup v2 directory (e.g. `/sys/fs/cgroup/sandboxed`)
3. attach the rules to this cgroup via *bpf(2)*
4. migrate processes into this cgroup

How do we use Landlock?

Process hierarchy (application)

1. create or receive Landlock rules
2. attach them to the current process via *seccomp(2)*

cgroup (container)

1. create Landlock rules
2. open a cgroup v2 directory (e.g. `/sys/fs/cgroup/sandboxed`)
3. attach the rules to this cgroup via *bpf(2)*
4. migrate processes into this cgroup

Demo

Why Landlock?

Why unprivileged access control?

- prevent privilege escalation
- minimize risk of sandbox escape
- same approach as Seatbelt/XNU Sandbox and OpenBSD Pledge

Why Landlock?

Why unprivileged access control?

- prevent privilege escalation
- minimize risk of sandbox escape
- same approach as Seatbelt/XNU Sandbox and OpenBSD Pledge

Why existing features do not fit in with this model?

- SELinux, AppArmor, Smack or Tomoyo
- seccomp-BPF
- (user) namespaces

Needs for Landlock

Flexible and dynamic rules

- express a wide range of restrictions
- extend over time

Needs for Landlock

Flexible and dynamic rules

- express a wide range of restrictions
- extend over time

Constraints for an unprivileged access control

- minimal attack surface
- prevent DoS
- do not leak sensitive kernel data
- avoid confused deputy attack
- multiple independent and stackable rules

Using eBPF to express access rules

extended Berkeley Packet Filter

- in-kernel bytecode machine:
  - optimized to be easily JITable
  - arithmetic operations, comparisons, jump forward, function calls
  - restricted memory read/write (i.e. program context and stack)
  - exchange data through maps between eBPF programs and userland
  - a program return a 32-bit value
- static program verification at load time:
  - memory access checks
  - register typing and tainting
  - pointer leak restrictions
- widely used in the kernel: network filtering, tracing...

How does Landlock works?

LSM hooks

- atomic security checks (e.g. `file_permission`)
- can be called multiple times in a syscall

How does Landlock works?

LSM hooks

- atomic security checks (e.g. `file_permission`)
- can be called multiple times in a syscall

Landlock rules

- a rule is tied to one LSM hook
- some LSM hook arguments available in the eBPF program context
- use maps to store kernel object references (e.g. `struct file`)
- dedicated functions to compare kernel objects

New eBPF features used by Landlock

Map of handles

- describe a kernel object from userland
- evaluation when updating an entry

New eBPF features used by Landlock

Map of handles

- describe a kernel object from userland
- evaluation when updating an entry

File system checker functions (eBPF helpers)

- `bpf_landlock_cmp_fs_beneath_with_struct_file(...)`
- `bpf_landlock_cmp_fs_prop_with_struct_file(...)`

New eBPF features used by Landlock

Map of handles

- describe a kernel object from userland
- evaluation when updating an entry

File system checker functions (eBPF helpers)

- `bpf_landlock_cmp_fs_beneath_with_struct_file(...)`
- `bpf_landlock_cmp_fs_prop_with_struct_file(...)`

Program subtype

- hook ID
- access bitfield tied to capabilities

New eBPF features used by Landlock

Map of handles

- describe a kernel object from userland
- evaluation when updating an entry

File system checker functions (eBPF helpers)

- `bpf_landlock_cmp_fs_beneath_with_struct_file(...)`
- `bpf_landlock_cmp_fs_prop_with_struct_file(...)`

Program subtype

- hook ID
- access bitfield tied to capabilities

cgroups attachment (by Daniel Mack)

- extend *bpf(2)* to be able to tie an eBPF program to a cgroup

A Landlock rule for the `file_permission` hook (C)

```
1 err = bpf_landlock_cmp_fs_beneath(0, map_fs,
2         BPF_MAP_ARRAY_OP_OR, ctx->args[0]);
3 if (!err)
4         return 0;
5 return EACCES;
```

A Landlock rule for the file_permission hook (eBPF)

```
/* specify an option, if any */
BPF_MOV32_IMM(BPF_REG_1, 0),
/* handles to compare with */
BPF_LD_MAP_FD(BPF_REG_2, map_fs),
BPF_MOV64_IMM(BPF_REG_3, BPF_MAP_ARRAY_OP_OR),
/* hook argument (struct file) */
BPF_LDX_MEM(BPF_DW, BPF_REG_4, BPF_REG_6,
        offsetof(struct landlock_data, args[0])),
/* checker function */
BPF_EMIT_CALL(BPF_FUNC_landlock_cmp_fs_beneath),
/* if the file is beneath a handle from the map */
BPF_JMP_IMM(BPF_JNE, BPF_REG_0, 0, 1),
BPF_EXIT_INSN(),
/* deny by default, if any error */
BPF_MOV32_IMM(BPF_REG_0, EACCES),
BPF_EXIT_INSN(),
```

Two complementary ways to enforce Landlock rules

Process hierarchy: application with built-in sandboxing

- restrict the current process and its future children
- use the *seccomp(2)* interface
- native use of *no_new_privs*

Two complementary ways to enforce Landlock rules

Process hierarchy: application with built-in sandboxing

- restrict the current process and its future children
- use the *seccomp(2)* interface
- native use of *no_new_privs*

cgroup: container sandboxing

- restrict processes from a cgroup
- complementary to rules for process hierarchies
- handle cgroup delegation with *no_new_privs*

Landlock LSM: Wrap-up

Unprivileged sandboxing

- use eBPF programs as access control rules
- applied through seccomp or tied to a cgroup
- can handle privileged features
- limited attack surface
- efficient and flexible

`https://lwn.net/Articles/700607`

mic@digikod.net

@l0kod