Landlock LSM: toward unprivileged sandboxing

Mickaël Salaün

ANSSI

September 14, 2017

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

Multiple sandbox uses

- built-in sandboxing (tailored security policy)
- sandbox managers (unprivileged and dynamic compartmentalization)
- container managers (hardened containers)

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	✓	✓	✓

Tailored access control to match your needs: programmatic access control

What can provide the needed features?

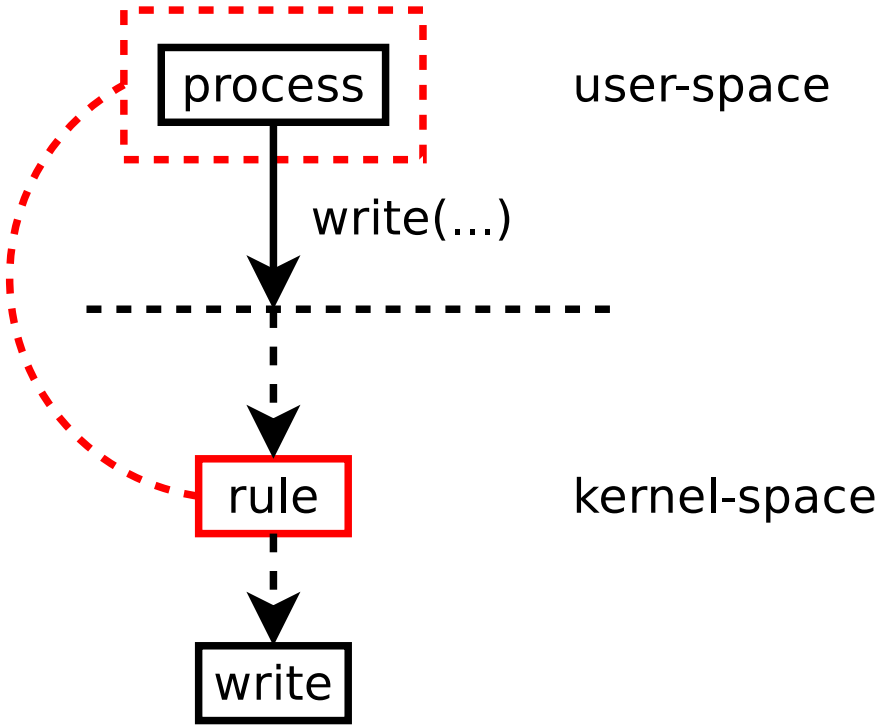
	Fine-grained control	Embedded policy	Unprivileged use
SELinux...	✓		
seccomp-bpf		✓	✓
namespaces		✓	∼
Landlock	✓	✓	✓

Tailored access control to match your needs: programmatic access control

Example

Run an application allowed to write only on a terminal.

Landlock overview



Landlock: patch v7

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

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)

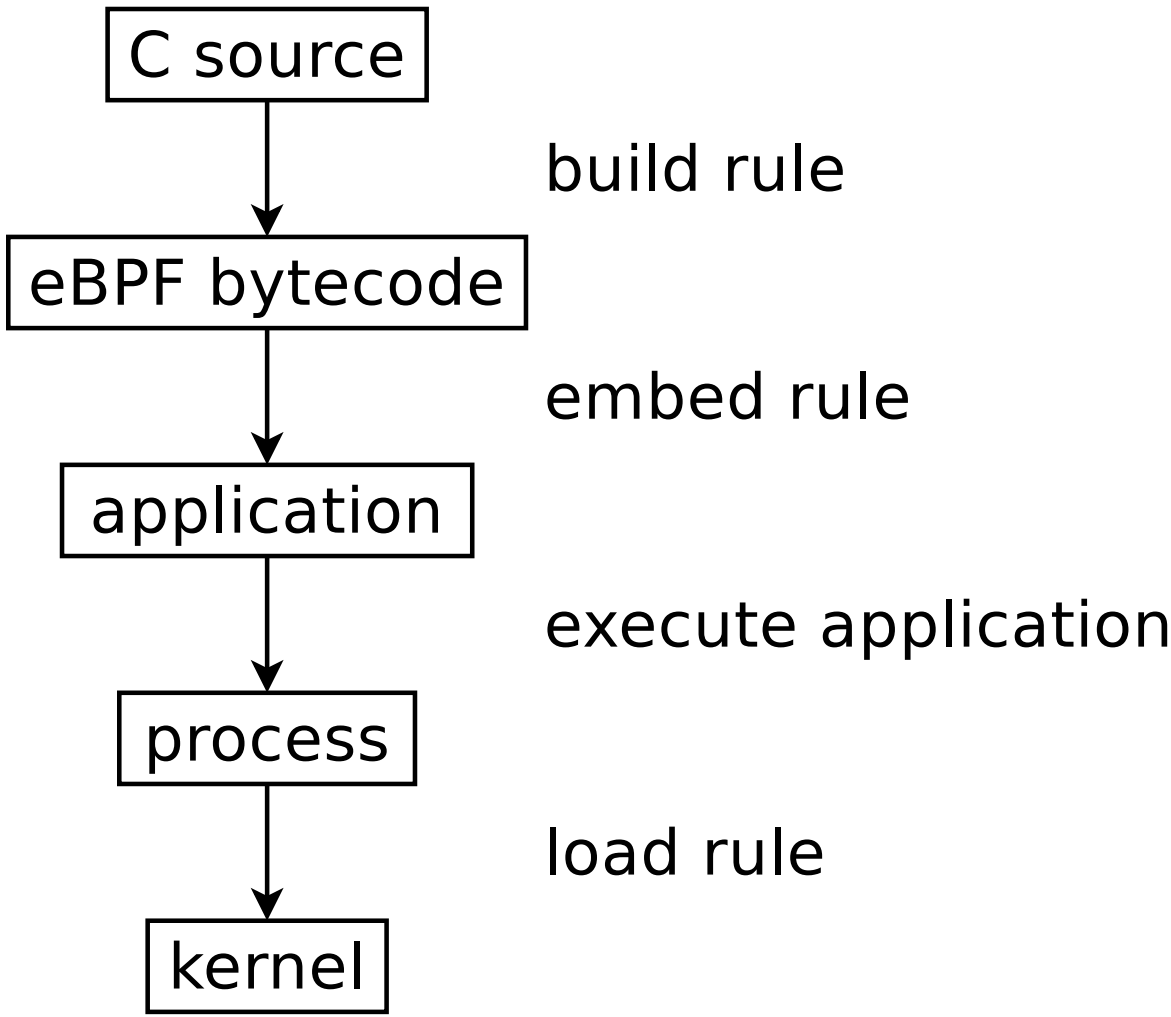
LSM framework

- 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

- rule: control an action on an object
- event: use of a kernel object type (e.g. file)
- action: read, write, execute, remove, IOCTL...

Life cycle of a Landlock rule



Landlock rule example

- read-only access to the filesystem...
- ...but allowed to write on TTY and pipes
- rule enforced on each filesystem access request

Landlock rule example

```
 1 SEC("landlock1")
 2 int landlock_fs_rule1(struct landlock_context *ctx)
 3 {
 4         int mode;
 5
 6         /* allow non-write actions */
 7         if (!(ctx->arg2 & LANDLOCK_ACTION_FS_WRITE))
 8                 return 0;
 9         /* get the file mode */
10         mode = bpf_handle_fs_get_mode(ctx->arg1);
11         /* allow write on TTY and pipes */
12         if (S_ISCHR(mode) || S_ISFIFO(mode))
13                 return 0;
14         return 1;
15 }
```

Landlock rule example

```
1  SEC("landlock1")
2  int landlock_fs_rule1(struct landlock_context *ctx)
3  {
4          int mode;
5
6          /* allow non-write actions */
7          if (!(ctx->arg2 & LANDLOCK_ACTION_FS_WRITE))
8                  return 0;
9          /* get the file mode */
10         mode = bpf_handle_fs_get_mode(ctx->arg1);
11         /* allow write on TTY and pipes */
12         if (S_ISCHR(mode) || S_ISFIFO(mode))
13                 return 0;
14         return 1;
15 }
```

Landlock rule example

```
 1 SEC("landlock1")
 2 int landlock_fs_rule1(struct landlock_context *ctx)
 3 {
 4         int mode;
 5
 6         /* allow non-write actions */
 7         if (!(ctx->arg2 & LANDLOCK_ACTION_FS_WRITE))
 8                 return 0;
 9         /* get the file mode */
10         mode = bpf_handle_fs_get_mode(ctx->arg1);
11         /* allow write on TTY and pipes */
12         if (S_ISCHR(mode) || S_ISFIFO(mode))
13                 return 0;
14         return 1;
15 }
```

Landlock rule example

```
1  SEC("landlock1")
2  int landlock_fs_rule1(struct landlock_context *ctx)
3  {
4          int mode;
5
6          /* allow non-write actions */
7          if (!(ctx->arg2 & LANDLOCK_ACTION_FS_WRITE))
8                  return 0;
9          /* get the file mode */
10         mode = bpf_handle_fs_get_mode(ctx->arg1);
11         /* allow write on TTY and pipes */
12         if (S_ISCHR(mode) || S_ISFIFO(mode))
13                 return 0;
14         return 1;
15 }
```

Landlock rule example

```
1  SEC("landlock1")
2  int landlock_fs_rule1(struct landlock_context *ctx)
3  {
4          int mode;
5
6          /* allow non-write actions */
7          if (!(ctx->arg2 & LANDLOCK_ACTION_FS_WRITE))
8                  return 0;
9          /* get the file mode */
10         mode = bpf_handle_fs_get_mode(ctx->arg1);
11         /* allow write on TTY and pipes */
12         if (S_ISCHR(mode) || S_ISFIFO(mode))
13                 return 0;
14         return 1;
15 }
```

Landlock rule example

```
1  SEC("landlock1")
2  int landlock_fs_rule1(struct landlock_context *ctx)
3  {
4          int mode;
5
6          /* allow non-write actions */
7          if (!(ctx->arg2 & LANDLOCK_ACTION_FS_WRITE))
8                  return 0;
9          /* get the file mode */
10         mode = bpf_handle_fs_get_mode(ctx->arg1);
11         /* allow write on TTY and pipes */
12         if (S_ISCHR(mode) || S_ISFIFO(mode))
13                 return 0;
14         return 1;
15 }
```

Landlock rule example

```
 1 SEC("landlock1")
 2 int landlock_fs_rule1(struct landlock_context *ctx)
 3 {
 4         int mode;
 5
 6         /* allow non-write actions */
 7         if (!(ctx->arg2 & LANDLOCK_ACTION_FS_WRITE))
 8                 return 0;
 9         /* get the file mode */
10         mode = bpf_handle_fs_get_mode(ctx->arg1);
11         /* allow write on TTY and pipes */
12         if (S_ISCHR(mode) || S_ISFIFO(mode))
13                 return 0;
14         return 1;
15 }
```

Landlock rule example

```
 1  SEC("landlock1")
 2  int landlock_fs_rule1(struct landlock_context *ctx)
 3  {
 4          int mode;
 5
 6          /* allow non-write actions */
 7          if (!(ctx->arg2 & LANDLOCK_ACTION_FS_WRITE))
 8                  return 0;
 9          /* get the file mode */
10          mode = bpf_handle_fs_get_mode(ctx->arg1);
11          /* allow write on TTY and pipes */
12          if (S_ISCHR(mode) || S_ISFIFO(mode))
13                  return 0;
14          return 1;
15  }
```

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

Loading a rule in the kernel

```
1  static union bpf_prog_subtype metadata = {
2          .landlock_rule = {
3                  .event = LANDLOCK_EVENT_FS,
4                  .ability = LANDLOCK_ABILITY_DEBUG,
5          }
6  };
7  union bpf_attr attr = {
8          .insns = bytecode_array,
9          .prog_type = BPF_PROG_TYPE_LANDLOCK_RULE,
10         .prog_subtype = &metadata,
11         // [...]
12 };
13 int rule_fd = bpf(BPF_PROG_LOAD, &attr, sizeof(attr));
```

Loading a rule in the kernel

```
1  static union bpf_prog_subtype metadata = {
2          .landlock_rule = {
3                  .event = LANDLOCK_EVENT_FS,
4                  .ability = LANDLOCK_ABILITY_DEBUG,
5          }
6  };
7  union bpf_attr attr = {
8          .insns = bytecode_array,
9          .prog_type = BPF_PROG_TYPE_LANDLOCK_RULE,
10         .prog_subtype = &metadata,
11         // [...]
12 };
13 int rule_fd = bpf(BPF_PROG_LOAD, &attr, sizeof(attr));
```

Loading a rule in the kernel

```
1  static union bpf_prog_subtype metadata = {
2          .landlock_rule = {
3                  .event = LANDLOCK_EVENT_FS,
4                  .ability = LANDLOCK_ABILITY_DEBUG,
5          }
6  };
7  union bpf_attr attr = {
8          .insns = bytecode_array,
9          .prog_type = BPF_PROG_TYPE_LANDLOCK_RULE,
10         .prog_subtype = &metadata,
11         // [...]
12 };
13 int rule_fd = bpf(BPF_PROG_LOAD, &attr, sizeof(attr));
```

Loading a rule in the kernel

```
1  static union bpf_prog_subtype metadata = {
2          .landlock_rule = {
3                  .event = LANDLOCK_EVENT_FS,
4                  .ability = LANDLOCK_ABILITY_DEBUG,
5          }
6  };
7  union bpf_attr attr = {
8          .insns = bytecode_array,
9          .prog_type = BPF_PROG_TYPE_LANDLOCK_RULE,
10         .prog_subtype = &metadata,
11         // [...]
12 };
13 int rule_fd = bpf(BPF_PROG_LOAD, &attr, sizeof(attr));
```

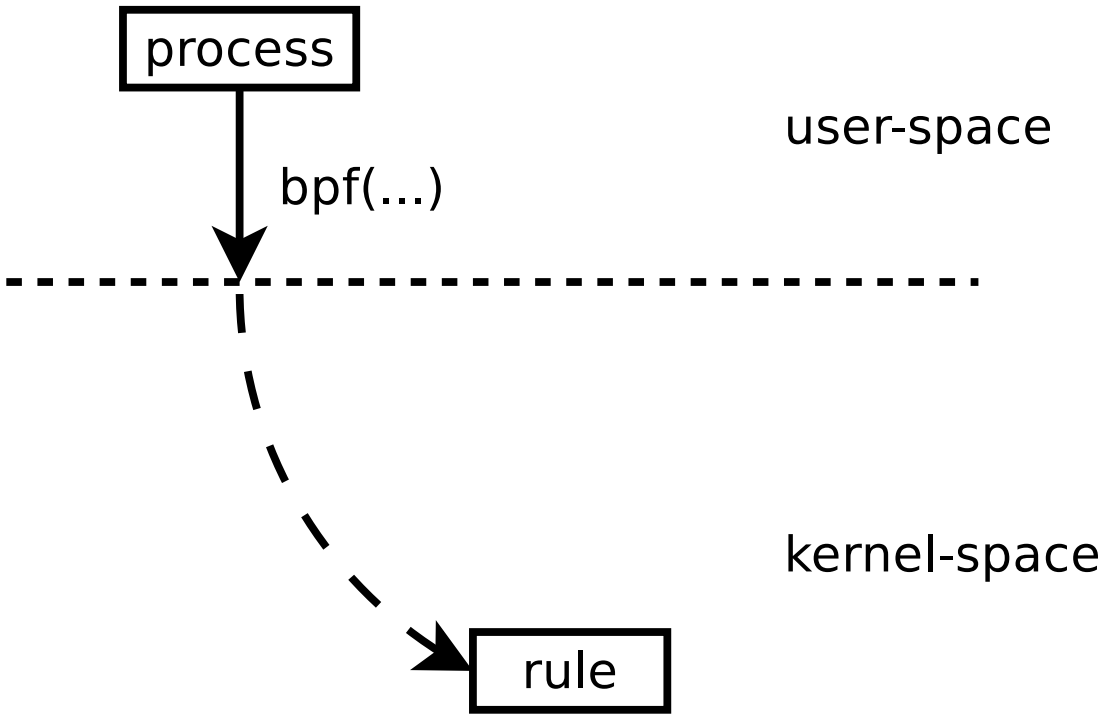
Loading a rule in the kernel

```
1  static union bpf_prog_subtype metadata = {
2          .landlock_rule = {
3                  .event = LANDLOCK_EVENT_FS,
4                  .ability = LANDLOCK_ABILITY_DEBUG,
5          }
6  };
7  union bpf_attr attr = {
8          .insns = bytecode_array,
9          .prog_type = BPF_PROG_TYPE_LANDLOCK_RULE,
10         .prog_subtype = &metadata,
11         // [...]
12 };
13 int rule_fd = bpf(BPF_PROG_LOAD, &attr, sizeof(attr));
```

Loading a rule in the kernel

```
1  static union bpf_prog_subtype metadata = {
2          .landlock_rule = {
3                  .event = LANDLOCK_EVENT_FS,
4                  .ability = LANDLOCK_ABILITY_DEBUG,
5          }
6  };
7  union bpf_attr attr = {
8          .insns = bytecode_array,
9          .prog_type = BPF_PROG_TYPE_LANDLOCK_RULE,
10         .prog_subtype = &metadata,
11         // [...]
12 };
13 int rule_fd = bpf(BPF_PROG_LOAD, &attr, sizeof(attr));
```

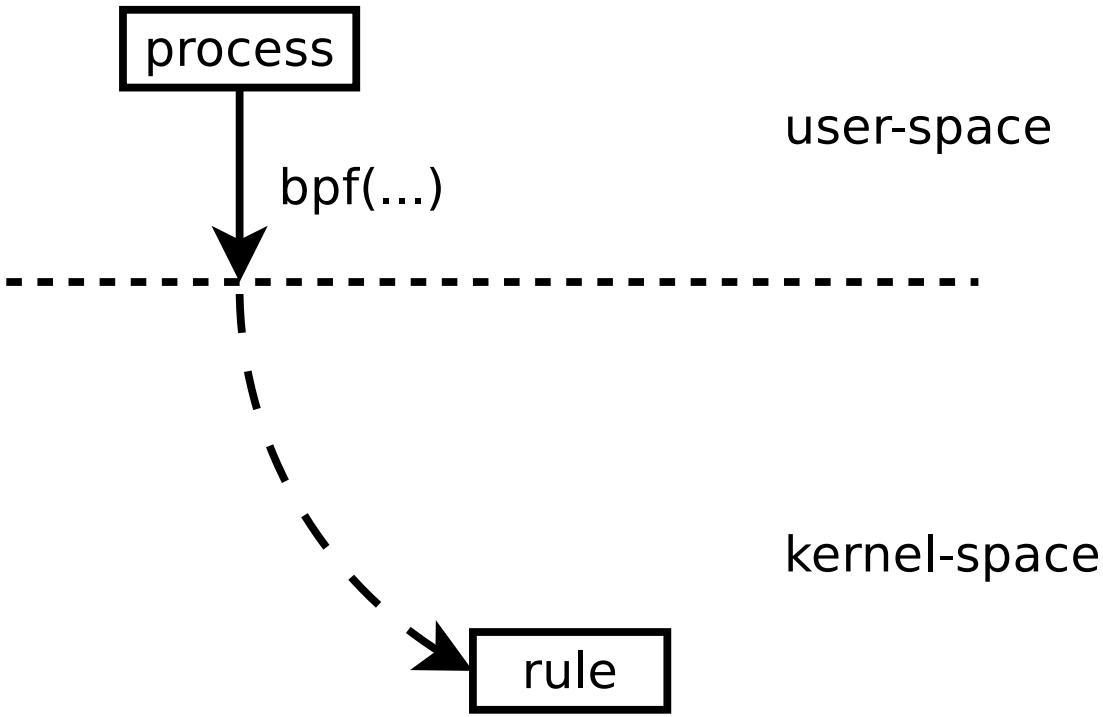
Loading a rule in the kernel



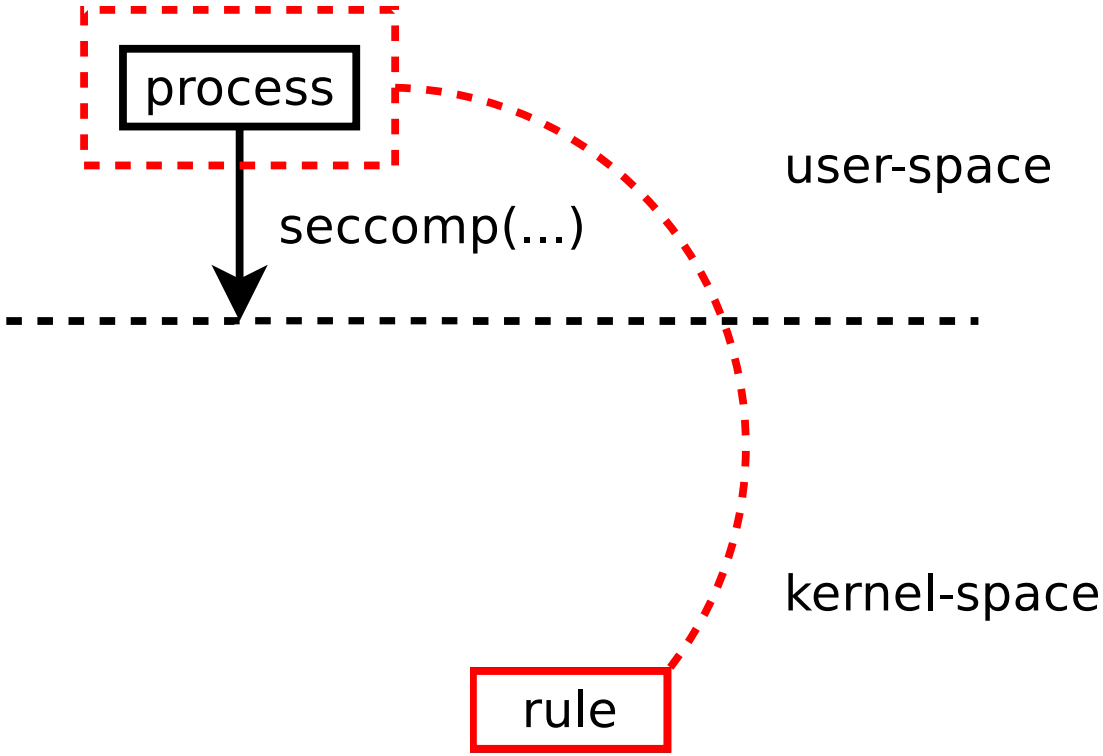
Applying a rule to a process

```
1 seccomp(SECCOMP_PREPEND_LANDLOCK_RULE, 0, &rule_fd);
```

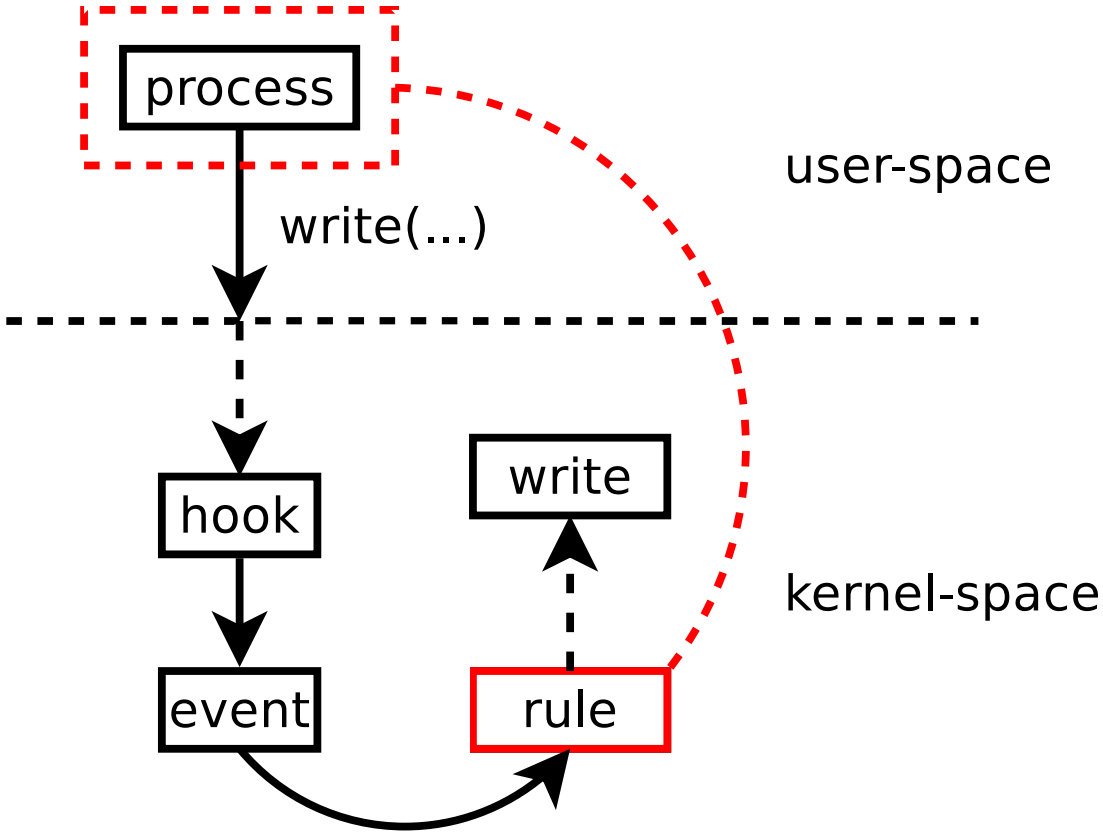
Applying a rule to a process



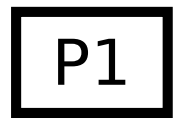
Applying a rule to a process



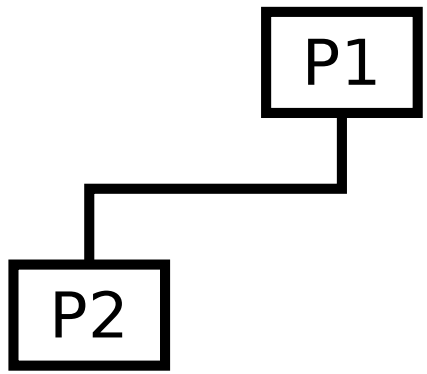
Applying a rule to a process



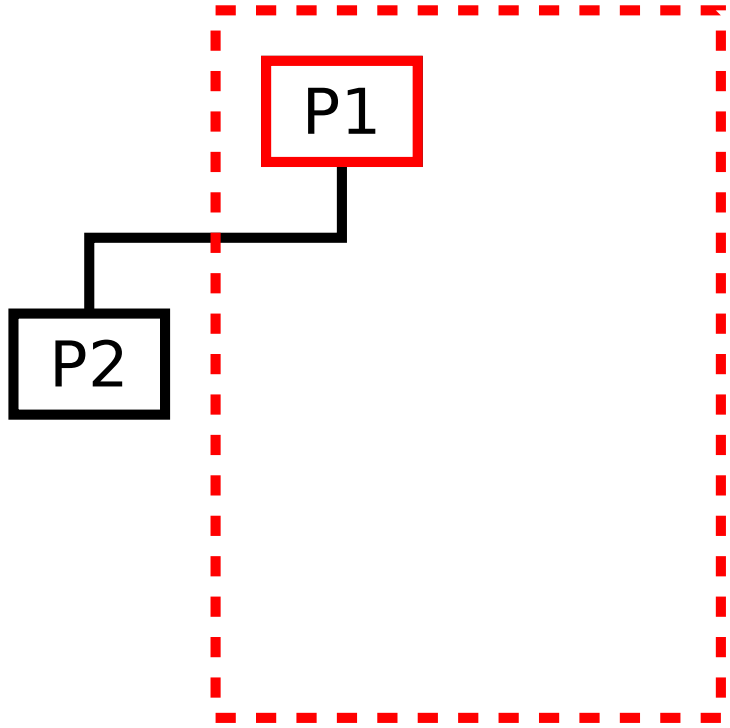
Rule enforcement on process hierarchy



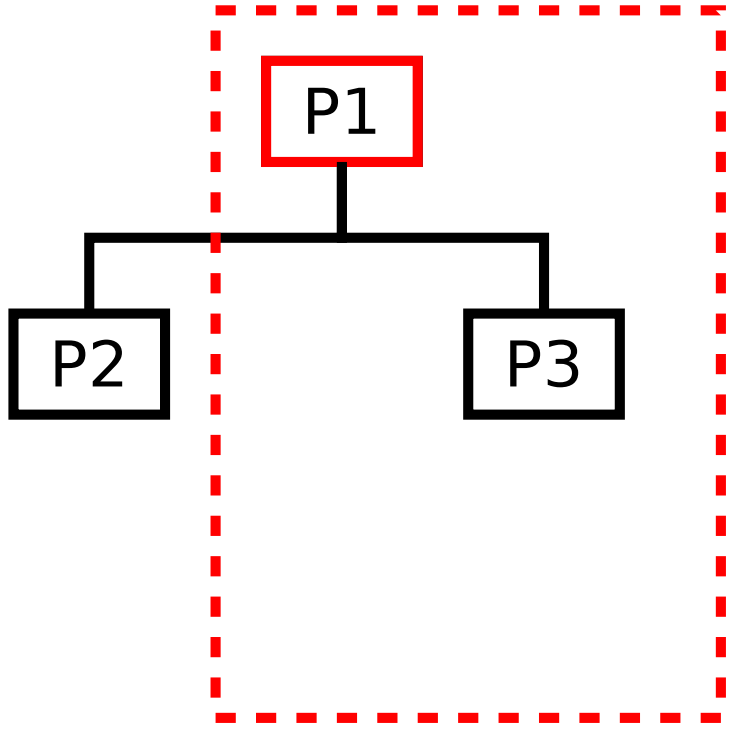
Rule enforcement on process hierarchy



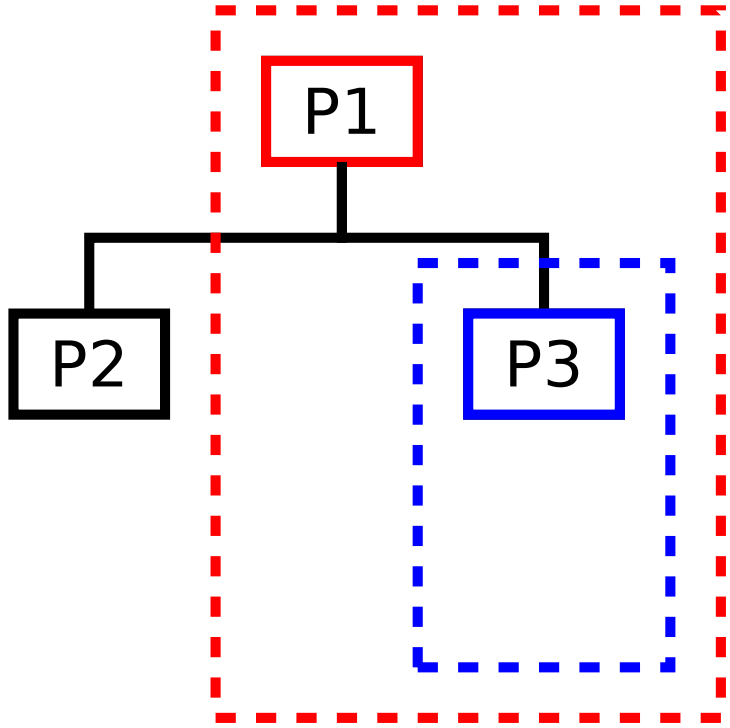
Rule enforcement on process hierarchy



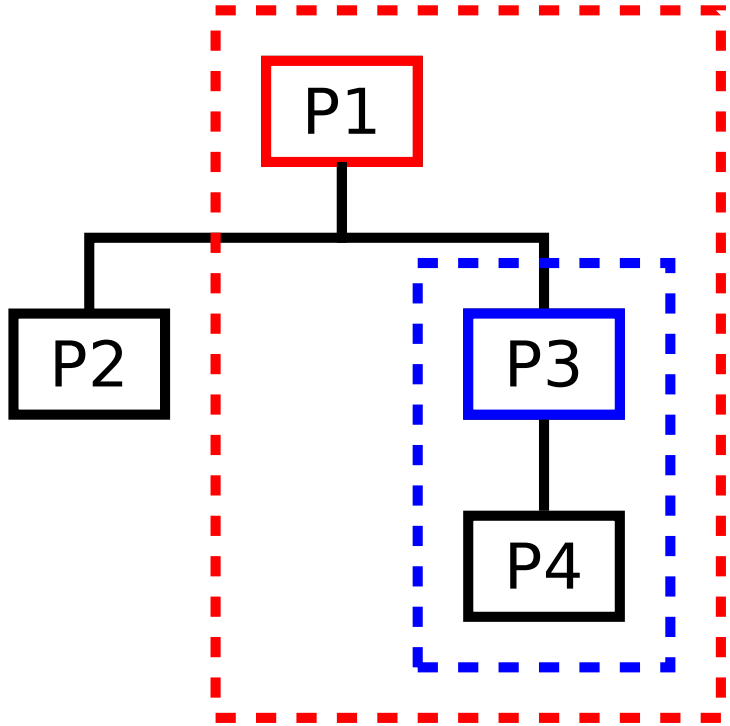
Rule enforcement on process hierarchy



Rule enforcement on process hierarchy



Rule enforcement on process hierarchy



Demonstration #1

An (almost) read-only filesystem

Landlock: pending features

- unprivileged access control
- enforcement through cgroups
- eBPF map fsview
- coming next...

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)

eBPF map fsview

Why?

restrict access to a subset of the filesystem

eBPF map fsview

Why?

restrict access to a subset of the filesystem

Challenges

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

eBPF map fsview

Why?

restrict access to a subset of the filesystem

Challenges

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

Proposal

- new eBPF map to identify a *filesystem view*: mount point hierarchies at a given time
- new eBPF function to compare a file access to such a view

Demonstration #2

What might a filesystem access control looks like?

Current roadmap

Incremental upstream integration

1. minimum viable product
2. cgroups handling
3. new eBPF map type for filesystem-related checks
4. unprivileged mode

Landlock: wrap-up

User-space hardening

- programmatic access control
- designed for unprivileged use

Landlock: wrap-up

User-space hardening

- programmatic access control
- designed for unprivileged use

Current status: patch v7

- autonomous patches merged in net, security and kselftest trees
- `security/landlock/*`: ∼1K SLOC
- ongoing patch series: LKML, `@l0kod`
- growing interest for containers, secure OS and service managers

https://landlock.io

Landlock context

```
1 struct landlock_context {
2         __u64 status;
3         __u64 event;
4         __u64 arg1;
5         __u64 arg2;
6 };
```

Landlock context

```
1 struct landlock_context {
2         __u64 status;
3         __u64 event;
4         __u64 arg1;
5         __u64 arg2;
6 };
```

Landlock events

- LANDLOCK_EVENT_**FS**

Landlock context

```
1 struct landlock_context {
2         __u64 status;
3         __u64 event;
4         __u64 arg1;
5         __u64 arg2;
6 };
```

Landlock events

- LANDLOCK_EVENT_**FS**

Landlock context

```
1 | struct landlock_context {
2 |         __u64 status;
3 |         __u64 event;
4 |         __u64 arg1;
5 |         __u64 arg2;
6 | };
```

Landlock events

- LANDLOCK_EVENT_**FS**

Landlock actions for an FS event

- LANDLOCK_ACTION_**FS_EXEC**
- LANDLOCK_ACTION_**FS_WRITE**
- LANDLOCK_ACTION_**FS_READ**
- LANDLOCK_ACTION_**FS_NEW**
- LANDLOCK_ACTION_**FS_GET**
- LANDLOCK_ACTION_**FS_REMOVE**
- LANDLOCK_ACTION_**FS_IOCTL**
- LANDLOCK_ACTION_**FS_LOCK**
- LANDLOCK_ACTION_**FS_FCNTL**

Landlock context

```
1 | struct landlock_context {
2 |         __u64 status;
3 |         __u64 event;
4 |         __u64 arg1;
5 |         __u64 arg2;
6 | };
```

Landlock events

- LANDLOCK_EVENT_**FS**
- LANDLOCK_EVENT_**FS_IOCTL**
- LANDLOCK_EVENT_**FS_LOCK**
- LANDLOCK_EVENT_**FS_FCNTL**

Landlock actions for an FS event

- LANDLOCK_ACTION_**FS_EXEC**
- LANDLOCK_ACTION_**FS_WRITE**
- LANDLOCK_ACTION_**FS_READ**
- LANDLOCK_ACTION_**FS_NEW**
- LANDLOCK_ACTION_**FS_GET**
- LANDLOCK_ACTION_**FS_REMOVE**
- LANDLOCK_ACTION_**FS_IOCTL**
- LANDLOCK_ACTION_**FS_LOCK**
- LANDLOCK_ACTION_**FS_FCNTL**

Available eBPF functions for Landlock rules

Any rule

- bpf_handle_fs_get_mode

Available eBPF functions for Landlock rules

Any rule

- bpf_handle_fs_get_mode

Debug mode: need `CAP_SYS_ADMIN`

- bpf_get_current_comm
- bpf_get_current_pid_tgid
- bpf_get_current_uid_gid
- bpf_get_trace_printk