Landlock: programmatic access control

Mickaël Salaün

@l0kod

June 21, 2017

Secure software

What is the threat?

- compromise a process (e.g. bug in a parser)
- privilege escalation (e.g. RunC vulnerability: CVE-2016-9962)
- access sensitive data

Secure software

What is the threat?

- compromise a process (e.g. bug in a parser)
- privilege escalation (e.g. RunC vulnerability: CVE-2016-9962)
- access sensitive data

What can we do?

- secure development
- follow the least privilege principle
- compartmentalize exposed processes (subset of initial accesses)

Existing compartmentalization solutions on Linux

	fine-grained control	unprivileged	embedded policy
SELinux. . .	✓		

Existing compartmentalization solutions on Linux

	fine-grained control	unprivileged	embedded policy
SELinux...	✓		
seccomp-bpf		✓	✓
namespaces		∼	✓

Existing compartmentalization solutions on Linux

	fine-grained control	unprivileged	embedded policy
SELinux...	✓		
seccomp-bpf		✓	✓
namespaces		∼	✓

Applications using this features

- service: OpenSSH, systemd...
- web browser: Chromium
- sandbox manager: Firejail, Subgraph/Oz, Minijail, StemJail...
- container manager: Docker, LXC...

Landlock: programmatic access control

Principles

- fine-grained control, unprivileged and embedded in applications
- free to choose a dedicated access control model
- stackable LSM (complementary restrictions)
- stackable rules (similar to seccomp-bpf)
- global system view (namespace agnostic)
- without SUID nor complex brokers

Landlock: programmatic access control

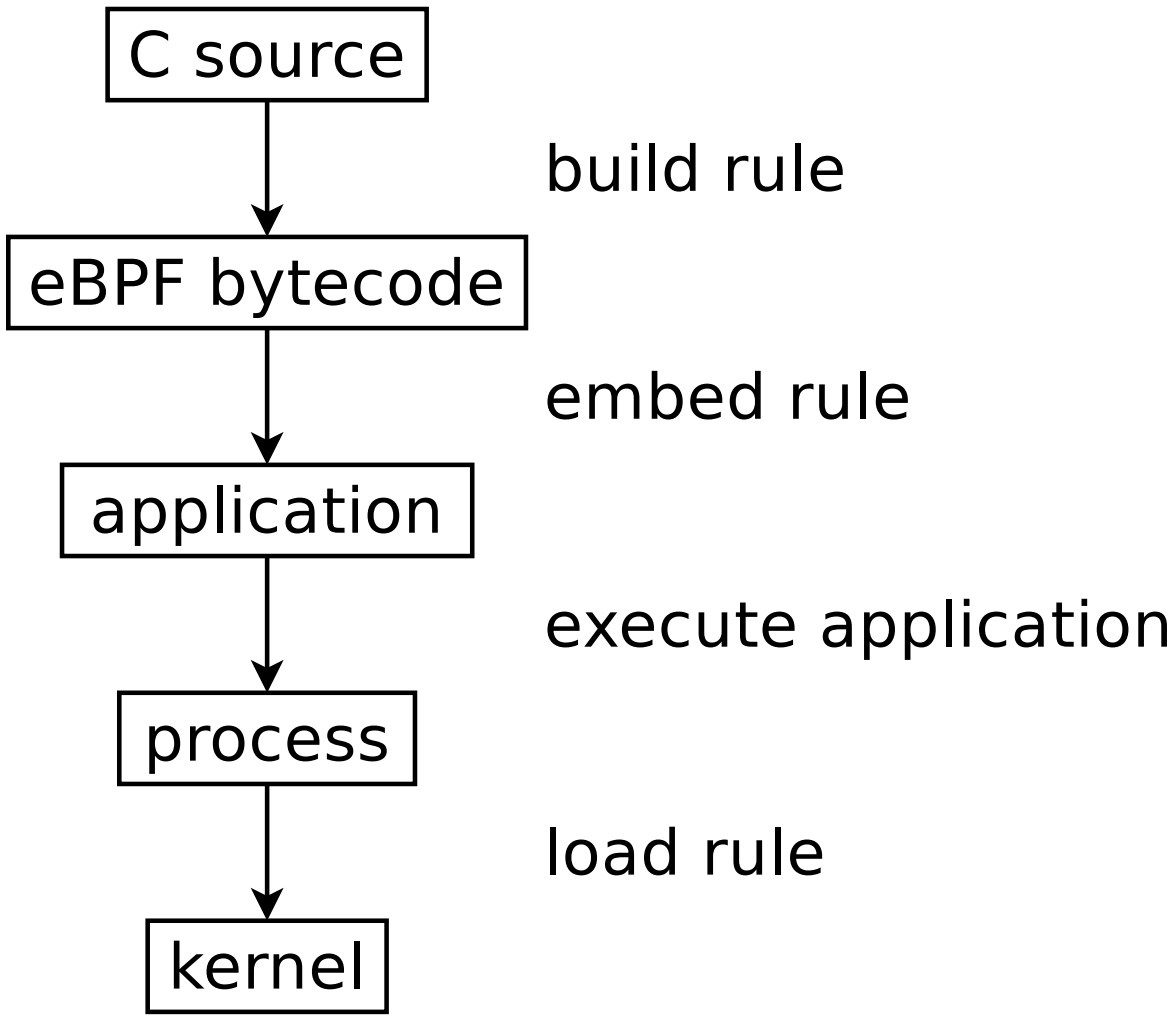
Principles

- fine-grained control, unprivileged and embedded in applications
- free to choose a dedicated access control model
- stackable LSM (complementary restrictions)
- stackable rules (similar to seccomp-bpf)
- global system view (namespace agnostic)
- without SUID nor complex brokers

Interested audience

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

Life cycle of a Landlock rule



Landlock rule example

- read-only access to the filesystem...
- ...but allowed to write on pipes
- rule applied on each filesystem-like 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 pipes */
12     if (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 pipes */
12     if (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 pipes */
12     if (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 pipes */
12     if (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 pipes */
12     if (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 pipes */
12      if (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 pipes */
12      if (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 pipes */
12      if (S_ISFIFO(mode))
13          return 0;
14      return 1;
15  }
```

Landlock context (WIP)

```
struct landlock_context {
  __u64 status;
  __u64 cookie;
  __u64 event;
  __u64 arg1;
  __u64 arg2;
};
```

Landlock context (WIP)

```
struct landlock_context {
    __u64 status;
    __u64 cookie;
    __u64 event;
    __u64 arg1;
    __u64 arg2;
};
```

Landlock events

- LANDLOCK_SUBTYPE_EVENT_FS

Landlock context (WIP)

```
struct landlock_context {
    __u64 status;
    __u64 cookie;
    __u64 event;
    __u64 arg1;
    __u64 arg2;
};
```

Landlock events

- LANDLOCK_SUBTYPE_EVENT_FS

Landlock context (WIP)

```
struct landlock_context {
    __u64 status;
    __u64 cookie;
    __u64 event;
    __u64 arg1;
    __u64 arg2;
};
```

Landlock events

- LANDLOCK_SUBTYPE_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 (WIP)

```
struct landlock_context {
    __u64 status;
    __u64 cookie;
    __u64 event;
    __u64 arg1;
    __u64 arg2;
};
```

Landlock events

- LANDLOCK_SUBTYPE_EVENT_FS
- LANDLOCK_SUBTYPE_EVENT_FS_IOCTL
- LANDLOCK_SUBTYPE_EVENT_FS_LOCK
- LANDLOCK_SUBTYPE_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

LSM + eBPF

Linux Security Modules

framework to provide a mechanism for various security checks to be hooked

LSM + eBPF

Linux Security Modules

framework to provide a mechanism for various security checks to be hooked

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
- static program verification at load time:
  - memory access checks
  - register typing and tainting
  - pointer leak restrictions
- widely used in the kernel: network filtering, tracing...

Landlock: unprivileged access control

Access control

- applying a security policy requires privileges
- alternative approach to the traditional interface (e.g. SUID) with a new one dedicated to coercitive access control
- protect other processes (e.g. tampering with ptrace)

Landlock: unprivileged access control

Access control

- applying a security policy requires privileges
- alternative approach to the traditional interface (e.g. SUID) with a new one dedicated to coercitive access control
- protect other processes (e.g. tampering with ptrace)

Protect the kernel and its resources

- reduced attack surface:
  - eBPF interpreter: static analysis
  - LSM part: only executed on viewable objects and after other controls
- protect against DoS
- prevent side channels

Landlock (WIP)

Demo

Landlock: wrap-up

Userland hardening

- fine-grained access control
- dynamic security policy
- designed for unprivileged use

Landlock: wrap-up

Userland hardening

- fine-grained access control
- dynamic security policy
- designed for unprivileged use

Current status: v6

- autonomous patch series merged (eBPF, LSM, kselftest)
- ongoing patch series: LKML, github.com/landlock-lsm, `@l0kod`
- backported as a project in LinuxKit

Landlock: wrap-up

Userland hardening

- fine-grained access control
- dynamic security policy
- designed for unprivileged use

Current status: v6

- autonomous patch series merged (eBPF, LSM, kselftest)
- ongoing patch series: LKML, github.com/landlock-lsm, `@l0kod`
- backported as a project in LinuxKit

Roadmap: incremental upstream integration

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