Capsicum
Capability-Based Sandboxing
David Drysdale
Google UK
What's LXC?

LXC is a userspace interface for the Linux kernel containment features. Through a powerful API and simple tools, it lets Linux users easily create and manage system or application containers.

Features

Current LXC uses the following kernel features to contain processes:

- Kernel namespaces (ipc, uts, mount, pid, network and user)
- Apparmor and SELinux profiles
- Seccomp policies
- Chroots (using pivot_root)
- Kernel capabilities
- CGroups (control groups)

LXC containers are often considered as something in the middle between a chroot and a full fledged virtual machine. The goal of LXC is to create an environment as close as possible to a standard Linux installation but without the need for a separate kernel.
What’s LXC?

LXC is a container interface for the Linux kernel containment features. Through a powerful API and simple tools, it lets Linux users easily create and manage system and application containers.

Features

Current LXC uses the following kernel features to contain processes:

- Kernel namespaces (ipc, uts, mount, pid, network and user)
- Apparmor and SELinux profiles
- Seccomp policies
- Chroots (using pivot_root)
- Kernel capabilities
- CGroups (control groups)
Features

Current LXC uses the following kernel features to contain processes:

- Kernel namespaces (ipc, uts, mount, pid, network and user)
- Apparmor and SELinux profiles
- Seccomp policies
- Chroots (using pivot_root)
- Kernel capabilities
- CGroups (control groups)
Agenda

- Ideas
  - Privilege Separation
  - Capabilities

- Capsicum
  - Hybrid with POSIX
  - Application changes

- Linux container features

- Status/Outlook
Check Your Privileges

● Drop unnecessary privileges
  ○ Just because a process starts as root, doesn't have to stay that way
Check Your Privileges

- Drop unnecessary privileges
  - Just because a process starts as root, doesn't have to stay that way
- Divide up software according to what privileges are needed
  - E.g. separate media processing from credentials processing
Check Your Privileges

- **Drop unnecessary privileges**
  - Just because a process starts as `root`, doesn't have to stay that way

- **Divide up software according to what privileges are needed**
  - E.g. separate media processing from credentials processing

- **Examples:**
  - OpenSSH: credential checking process
  - Chrome: renderer processes

- **Design impact:**
  - Do privileged operations first
  - Pass resources down a privilege gradient
Capability-Based Security

- Make the privileges that a process holds more explicit
Capability-Based Security

- Make the privileges that a process holds more explicit
- Access *objects* via unforgeable token: the **capability**
  - Identifies the object
  - Accompanying *rights* give allowed operations
  - Can only reduce, not increase rights
  - Can pass capabilities around
Capability-Based Security

● Make the privileges that a process holds more explicit

● Access objects via unforgeable token: the **capability**
  ○ Identifies the object
  ○ Accompanying **rights** give allowed operations
  ○ Can only reduce, not increase rights
  ○ Can pass capabilities around

● Remove other ways of accessing objects
  ○ No access by name, i.e. no global namespaces
Analogy: File Descriptors

- Refers to kernel object (open file, open socket, ...)
- Can only be created by the kernel
- Can be passed between processes (over UNIX domain sockets)
Analogy: File Descriptors

- Refers to kernel object (open file, open socket, ...)
- Can only be created by the kernel
- Can be passed between processes (over UNIX domain sockets)

- ... but no real model of rights
  - O_RDONLY / O_RDWR not good enough
Capsicum: Make the analogy reality

- File descriptors as Capsicum capabilities
Capsicum: Make the analogy reality

- File descriptors as Capsicum capabilities
- Add fine-grained rights, policed by kernel
  - CAP_READ, CAP_WRITE, CAP_LOOKUP, CAP_FCHMOD, ...
  - CAP_BIND, CAP_ACCEPT, CAP_CONNECT, CAP_SETSOCKOPT, ...
Capsicum: Make the analogy reality

- File descriptors as Capsicum capabilities
- Add fine-grained rights, policed by kernel
  - CAP_READ, CAP_WRITE, CAP_LOOKUP, CAP_FCHMOD, ...
  - CAP_BIND, CAP_ACCEPT, CAP_CONNECT, CAP_SETSOCKOPT, ...
- Capability mode
  - Remove access to global namespaces
  - Turn off most ways of minting new (unrestricted) file descriptors
    - openat(dfd, "path"...) allowed
    - accept(socket ...) allowed
Example: strings
Example: *strings*

+ cap_rights_t rights;
+ cap_rights_limit(fileno(stdout), cap_rights_init(&rights, CAP_WRITE, CAP_FSTAT));
+ cap_rights_limit(fileno(stderr), cap_rights_init(&rights, CAP_WRITE));

```c
for (ii = 0; ii < num_streams; ++ii) {
    ...
```
Example: `strings`

```c
+ cap_rights_t rights;
+ cap_rights_limit(fileno(stdout), cap_rights_init(&rights, CAP_WRITE, CAP_FSTAT));
+ cap_rights_limit(fileno(stderr), cap_rights_init(&rights, CAP_WRITE));
+ cap_rights_init(&rights, CAP_READ, CAP_SEEK, CAP_FSTAT, CAP_FCNTL);
+ for (ii = 0; ii < num_streams; ++ii) {
+   if (streaminfo[ii].stream)
+     cap_rights_limit(fileno(streaminfo[ii].stream), &rights);
+ }

for (ii = 0; ii < num_streams; ++ii) {
  ...
```
Example: *strings*

```c
+ cap_rights_t rights;
+ cap_rights_limit(fileno(stdout), cap_rights_init(&rights, CAP_WRITE, CAP_FSTAT));
+ cap_rights_limit(fileno(stderr), cap_rights_init(&rights, CAP_WRITE));
+ cap_rights_init(&rights, CAP_READ, CAP_SEEK, CAP_FSTAT, CAP_FCNTL);
+ for (ii = 0; ii < num_streams; ++ii) {
+   if (streaminfo[ii].stream)
+     cap_rights_limit(fileno(streaminfo[ii].stream), &rights);
+ }
+ cap_enter();
+ for (ii = 0; ii < numStreams; ++ii) {
  ...
```
Features

Current LXC uses the following kernel features to contain processes:

- Kernel namespaces (ipc, uts, mount, pid, network and user)
- Apparmor and SELinux profiles
- Seccomp policies
- Chroots (using pivot_root)
- Kernel capabilities
- CGroups (control groups)
fine grained

broad brush

chroot

simple

complex

defined

capabilities

kernel capabilities
Themes

- Involves code changes
- Less flexible in some ways
  - But simple to understand & apply
  - Not specific to root
- More fine-grained in other ways
  - FD-by-FD, not application-wide
- Easy to analyze
- Composes with other features
Status

● OS Support
  ○ In FreeBSD >= 10.x
  ○ Out-of-tree patch set for Linux (github.com/google/capsicum-linux)

● Application Support
  ○ ~20 in-tree FreeBSD applications
  ○ OpenSSH / tcpdump / xz
  ○ (Chromium)

● Next
  ○ More applications (join us!)
  ○ More debugging facilities
References

- Home page: http://www.cl.cam.ac.uk/research/security/capsicum/
- Linux home page: http://capsicum-linux.org/
- Intro article: http://capsicum-linux.blogspot.co.uk/2015/02/an-overview-of-capsicum.html
- Linux source code: https://github.com/google/capsicum-linux
- Test suite: https://github.com/google/capsicum-test
- Projects list: https://github.com/google/capsicum-test/wiki/Projects
- Strings vulnerability: https://lcamtuf.blogspot.co.uk/2014/10/psa-dont-run-strings-on-untrusted-files.html

David Drysdale <drysdale@google.com>