Desktop applications: life inside a sandbox

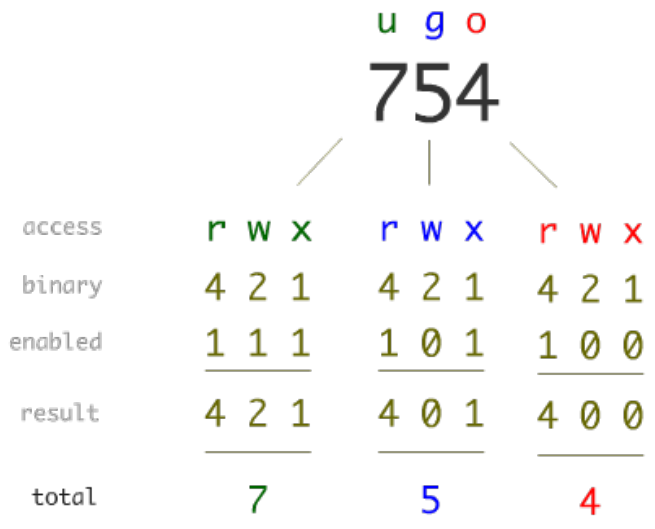
David King <amigadave@amigadave.com>

11th August 2018 / GNOME.Asia

https://amigadave.com/presentations/gnome_asia_sandboxing_2018.pdf

Unconfined processes

- Before sandboxing, every process ran unconfined
- The only constraints on what a process could do were global to the user
- Traditional Unix permissions model (discretionary access control)

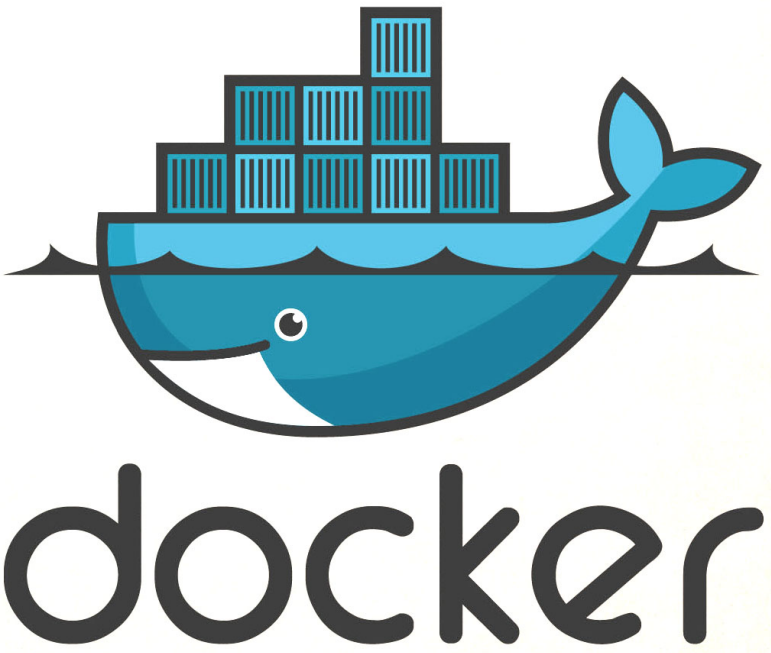


Mandatory access control

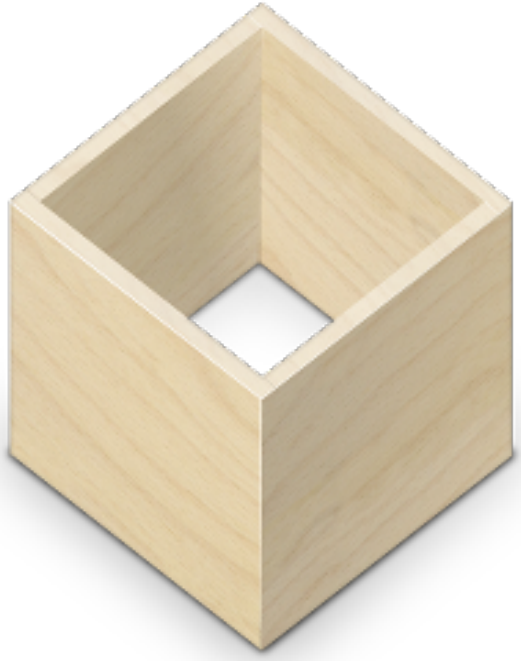
- Complex but expressive way to describe allowed operations
- Not just files, but also sockets, file descriptors, processes and more
- Good for daemons with well-known operations
- Difficult to debug policy violations

Containers

- A controlling process which restricts privileges on the contained application
- Namespaces, cgroups and seccomp used to give a virtual view of the rest of the system
- Holes are added to the container for TCP ports, host system files and so on
- Good for daemons and application development



Sandboxing of desktop applications



- Containers but tailored for desktop applications
- Special handling of .desktop files, XDG directories and so on
- D-Bus proxying and filtering
- Portals for on-demand access to the host system

Leveraging bubblewrap

- Privileged helper to create the sandbox
- Shared with other projects as part of Project Atomic, but can be used standalone
- Has a wide array of options for cgroup, IPC, PID, network and user namespaces
- Uses seccomp to filter out undesired syscalls
- Monitors the process running inside the sandbox

PID namespacing

```
david@lenovodave ~ $ ps
  PID TTY          TIME CMD
    1 ?        00:00:00 bwrap
    2 ?        00:00:00 bash
    3 ?        00:00:00 ps
david@lenovodave ~ $
```

- An application running in the sandbox only has `bwrap` visible as a parent process
- The PID of the app inside the sandbox will be different to outside
- `_NET_WM_PID` no longer works!

Filesystem namespacing

- Lots of bind mounts inside the sandbox to give a reduced (and readonly) view of the host filesystem
- `/app` is for the files from the application, and has the usual `/app/lib`, `/app/usr` and so on. Typically, building with a prefix of `/app` is sufficent for most applications to run under flatpak
- `/usr` is for files from the runtime
- Extensions are mounted in their defined locations
- `/run/host` is used for various parts of the host filesystem that are safe to expose, including the fonts cache

Filesystem namespacing continued

- Several files from the host system are also bind mounted (fstab, bash-completion configuration, SSL certificate store and so on)
- D-Bus system and session (or user) bus sockets are available
- Wayland, X11 and journal sockets are available
- `$HOME/.var/app/$APPID` is made available for writing by the application
- Relevant `XDG_*` environment variables are set to point to appropriate locations

D-Bus

- Extremely common for desktop applications to communicate over the session (or user) bus
- An application can own its own name on the bus, but no other
- Flatpak has a filtering D-Bus proxy, so that an application can only talk to allowed names
- Portals (`org.freedesktop.portal.*`) are an exception, and applications can always talk to them

Background

- The bridge between an untrusted application and a trusted host system
- Most show a dialog to request (implicit) permission from the user, rather than asking a yes/no question
- Permission store to remember user decisions
- Special flatpak portal to spawn processes outside (or inside!) the sandbox

xdg-desktop-portal

- Frontend service for flatpak and snapd (and possibly other) desktop sandboxes
- Provides a set of D-Bus interfaces for interacting with the host system
- Relies on a backend (such as xdg-desktop-portal-gtk) to show native dialogs to the user
- Many portals have built-in toolkit support, and so are transparent to the application

xdg-desktop-portal interfaces

- Common for desktop applications are the file chooser, documents, notification and URI portals
- Less common include the printing, email, screenshot, screencast, device, inhibit, network and proxy portals
- Some very specialised portals include the flatpak portal
- All portals use the `org.freedesktop.portal.Request` interface, to avoid D-Bus method timeouts while waiting for user interaction

xdg-desktop-portal typical usage

- Make a portal request and receive back a handle according to the `org.freedesktop.portal.Request` interface
- A dialog is often shown by the portal backend to receive user interaction, and placed on the appropriate window as given by the initial request
- The `Response` signal is received on the `Request` object to indicate that user interaction is complete
- The application processes the results of the portal operation
- Some requests persist, such as for ongoing operations like screencasting, and for those a `Session` interface is used

Document portal

- FUSE filesystem to store a map of files opened from inside the sandbox
- Permissions located in the permissions store (see `flatpak document-list`
- Can be accessed from outside the sandbox in `/run/user/$UID/doc`
- Not intended to be used directly from applications, but implicitly from tookits

xdg-desktop-portal-gtk

- GTK+ and GNOME backend for xdg-desktop-portal
- Seamless support for file access, printing, inhibiting inside GTK+ (which automatically detects when running under flatpak)
- Supports screenshotting and screencasting with GNOME Shell
- Other backends possible, such as Qt/KDE with xdg-desktop-portal-kde

In the future

- dconf support for containers (needs work on D-Bus, flatpak and dconf)
- Screensharing support is almost ready for Firefox, using PipeWire
- Special support for self-updating applications?

Further resources

- Flatpak and portal sources:
  https://github.com/flatpak
- xdg-desktop-portal D-Bus API:
  https://flatpak.github.io/xdg-desktop-portal/portal-docs.html
- Matthias Clasen's blog:
  https://blogs.gnome.org/mclasen/
- Alexander Larsson's blog:
  https://blogs.gnome.org/alexl/
- IRC:
  #flatpak on freenode

Questions?

- Ask your questions now!