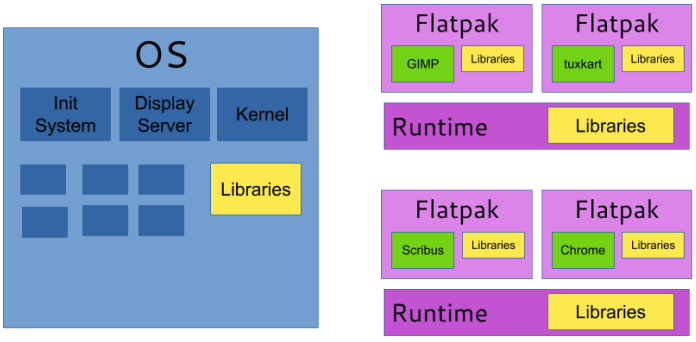
Flatpaks: Having your cake and eating it too!

I run a tight ship. The only rats on board are the ones I keep as pets!

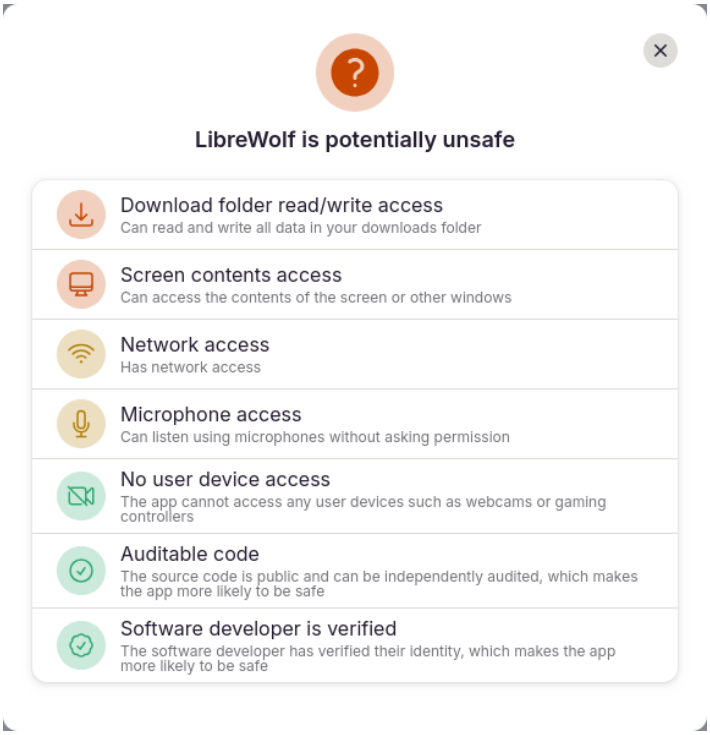
In today's digital landscape, the PC has become the tool of choice for modern productivity. However, with the increasing reliance on these systems comes a growing concern for security. As workstations often handle sensitive information and access critical networks, they have at times become a prime target for cyber threats and malicious attacks. In a previous blog post, I explain the shortcomings PC operating systems have in terms of application security. So the question must be asked, what tools are available to address these shortcomings? Can concepts that have proven to work for computer network security such as zero trust architecture be applied to a desktop environment?

Flatpaks!



Flatpaks provide standardized environments for applications to be packaged in and run on different systems. This is necessary since while Linux systems share the same kernel API, Linux systems are configured in a vast variety of ways, utilizing different display managers and software libraries. Once you install flatpak on your system, you can issue a `flatpak install` command to then install an application of your choice.

In addition to this, Flatpak uses the bubblewrap program which allows applications to be sandboxed from the host system, while explicitly allowing access parts of the system. Specifying the certain kinds of interfaces an application is allowed to interact with is one of the key principles behind security of mobile applications, and is what Flatpaks provide.



Flathub, the default public repository for flatpaks, gives a brief overview of permissions granted to a flatpak.

In theory, this sandboxing feature allows an application to be executed safely and without fearing unauthorized access. In practice, many applications are configured with overly lax permissions, and as a result become ineffective at isolating an application. Flatpak itself has some issues with permissions not being granular enough, such as when an application requires access to a single USB device in `/dev`, as opposed to the entirety of `/dev` which is being fixed as of the time of writing. In a related fashion, Android had permissions issues where it required location access in order to access Bluetooth devices until Android 12 (API level 31).

Fine, I'll do it myself

Luckily, Flatpak overrides allows a user to grant or prevent access to resources to applications. This means that in order to solve the issue of overly permissive application access, we can simply override the permissions that were granted to the application with those of our own. To do so, we can use the `flatpak override` command, which adds text file entries to the overrides directory (located in `/var/lib/flatpak/overrides/global` for system-wide flatpaks, or `~/.local/share/flatpak/overrides` for user flatpaks).

Flatpak overrides in-depth

```
tangy@clipper:~/ > ls /var/lib/flatpak/overrides
chat.simplex.simplex                    org.audacityteam.Audacity
com.github.iwalton3.jellyfin-media-player  org.freecad.FreeCAD
com.github.johnfactotum.Foliate         org.getmonero.Monero
com.github.xournalpp.xournalpp          org.gimp.GIMP
com.google.AndroidStudio                org.gnome.Boxes
com.prusa3d.PrusaSlicer                 org.keepassxc.KeepassXC
com.valvesoftware.Steam                 org.keepassxc.KeePassXC
dev.vencord.Vencord                     org.mozilla.firefox
dev.vencord.Vesktop                     org.mozilla.Thunderbird
global                                  org.torproject.torbrowser-launcher
io.freetubeapp.FreeTube                 us.zoom.Zoom
io.gitlab.librewolf-community
```

Just like a firewall's ruleset, it is first important to create an implicit deny to all permissions of an application. We can do this by simply running `flatpak override <options>` without specifying any application. Here is an example of a resulting global override file (the file is listed above and named 'global'):

```
# global
[Context]
devices=!all;!kvm;!shm;dri
features=!bluetooth;!canbus;!devel;!multiarch;!per-app-dev-shm
filesystems=!host:reset
shared=!ipc
sockets=!cups;!fallback-x11;!gpg-agent;!pcsc;!pulseaudio;!session-bus;!ssh-auth;!system-bus;!x11;wayland

[Environment]
ELECTRON_OZONE_PLATFORM_HINT=auto
GTK_THEME=Adwaita:dark
QT_QPA_PLATFORM=wayland

[Session Bus Policy]
org.freedesktop.Flatpak=none
org.freedesktop.impl.portal.PermissionStore=none
org.freedesktop.secrets=none

[System Bus Policy]
org.freedesktop.UDisks2=none
org.freedesktop.UPower=none
```

Note that the `!` means that the permission that follows is denied. `filesystems=!host:reset` means that applications by default have no filesystem access, other than their own folder located in `.../flatpak/app`.

The name of each file corresponds to the name of the program's app-id. The following example is the result of running `flatpak override --unshare=network --filesystem=~/KPass org.keepass.KeePassXC`

```
tangy@clipper:~/ > cat /var/lib/flatpak/overrides/org.keepassxc.KeePassXC
[Context]
filesystems=~/KPass
shared=!network
```

This example allows for greater security as it denies KeePassXC, a password management, from accessing the Internet, and allowing it access to only a single directory.

Revision #6
Created 20 November 2024 15:16:11 by GT
Updated 14 January 2025 16:19:20 by GT