Blog

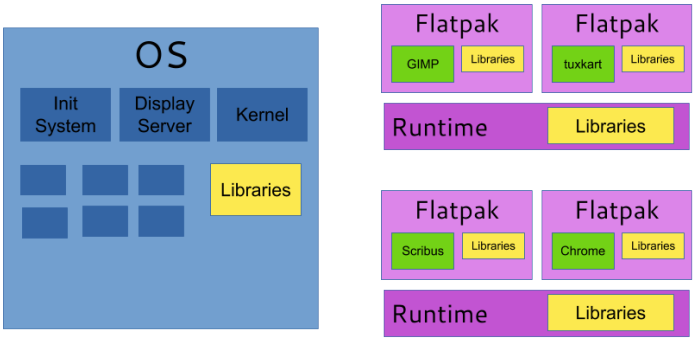
- Flatpaks: Having your cake and eating it too!
- The PC-Mobile Security Divide
- Into the Nixverse
- Hi there, I'm Gentman Tan!
- Hi there, welcome to grok.zone!

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!

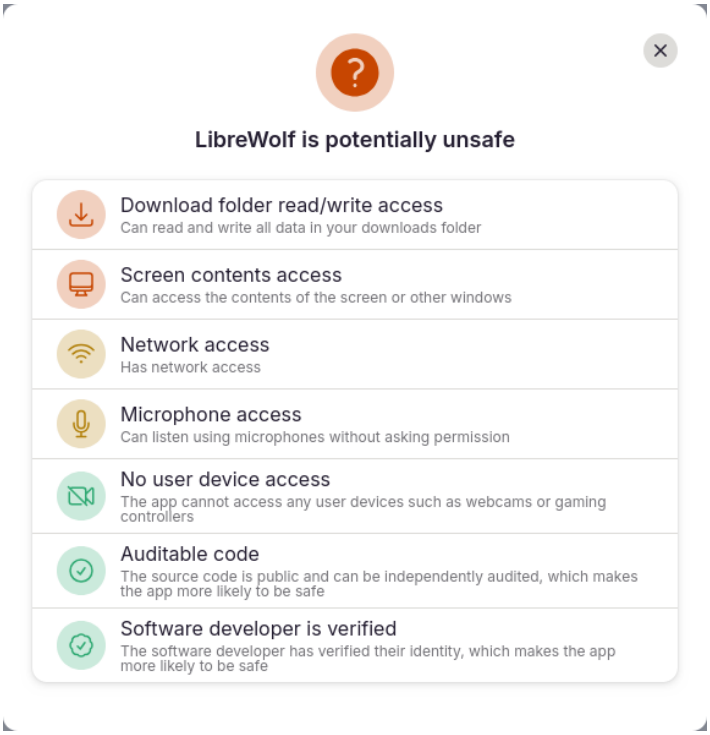
In today's digital landscape, the PC has become the tool of choice for modern productivity. However, with the increasing reliance on these systems comes a growing concern for security. As workstations often handle sensitive information and access critical networks, they have at times become a prime target 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.

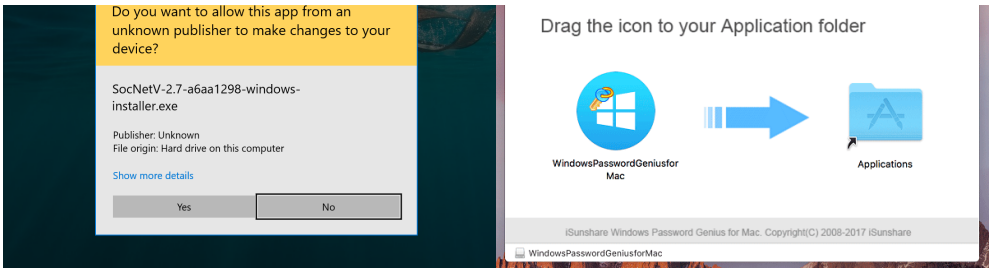
The PC-Mobile Security Divide

Captain, I'm receiving a transmission from an unknown alien source.
Spock, can you decipher the message?
Yes captain, it's an executable file named 'word.exe'. I tried running it but application is not starting.
Err, let me try running it as an administrator.
(The Enterprise's computer systems blue screen and power off due to a poor sense of operations security)

In the realm of personal computing, there exists a strange divide. On one hand, you have the mobile side, where permissions are explicitly granted to an application. *Maybe* that hilarious iBeer app, which turns your phone into a simulated pint glass, doesn't need access to your contacts... And, if any other app prompts you to access inappropriate permissions, you can simply deny it access. On the other hand, PC apps kind of just do whatever they feel like doing. Everything from your office programs to a funny purple monkey that dances around your desktop has essentially unfettered access to your user space.

Current Solutions

Applications that Windows and macOS users interact with nowadays are signed with a developer's private key when they are packaged, giving them the stamp of approval from Microsoft or Apple. And Microsoft and Apple are trustworthy, right? So... what's the issue?



Well, as it turns out these two companies don't really review the contents of the executables you decide to run. And for good reason- with the insane amount of applications being created, it's impossible to review each one or probably even a majority of them. With that said, there are places like the Microsoft Store and the App Store where applications are reviewed before being published. However, even this is not foolproof plus you are always going to have some niche application that requires you to manually run or install an application outside of your curated app store.

But what about my anti-virus?

Anti-viruses are basically big tables of program signatures that try to identify malicious files. However, it isn't hard to make malicious applications that get missed by an antivirus, especially with the use of polymorphic code which allows an executable to change its appearance to anti-viruses.

So... what's the solution?

Well, to be clear the average user is probably smart enough to install applications from trustworthy sources. Or, maybe it is more accurate to say search engines have become more cognizant as to verifying the links to websites to download software. Maybe anti-viruses are actually getting better at detecting malware. However, it still stands that sticking to this paradigm of installing software is going to always be a cat and mouse game of the bad guys creating novel, malicious software and the good guys detecting it.

On the contrary,allowing users to clearly grant permissions to applications has proven itself to be a good solution in the mobile operating system space. This is not without its issues though; for example, if permissions aren't granular enough, a developer might be forced to add an overly broad permission. It also requires the OS to have a secure enough sandbox, lest the applications

would grant themselves permissions. However, with enough improvements such an access control system can, and has proved in mobile operating systems, to be a great security layer.

As an aside, it would also be nice if the applications people commonly use were all open source and auditable... including the operating system... but maybe I'm a little off my gourd.

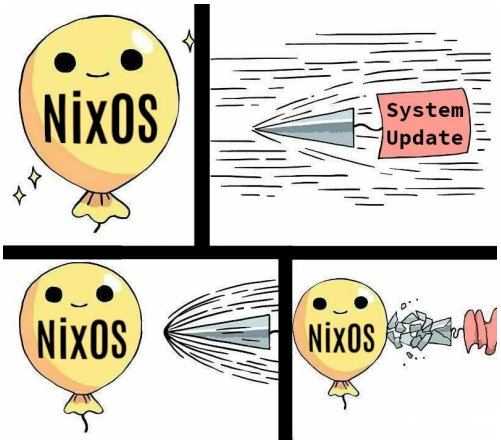
What is *your* solution?

I'm a Linux user, and for the uninitiated Linux, or more accurately GNU/Linux, is essentially a very customizable operating system due to its highly documented and open source nature. An application for Linux called Flatpak allows for the sand-boxing of user applications, while utilizing permission-based access similar to mobile applications. I have found it to be a wonderful way to install and use applications in a secure manner.

I'll explain in detail how I use flatpaks in my system in an upcoming blog post. For now, take a look at my dotfiles.

Happy computing!

Into the Nixverse



It seems like every time you think you know everything there is to know about Linux, there's always something that catches you by surprise...

NixOS and the Nix package manager is software that allows you to configure a Linux system or shell environment in a declarative and therefore reproducible manner. In other words, your system can be completely immune from configuration drift. Nix also keeps track of a program's dependencies without mutating the global state of packages, allowing for truly reproducible software builds and system configurations. In addition, NixOS allows you to rollback to a previous system state, just in case something goes horribly wrong after you update things. Combined with version control using Git, NixOS lays the groundwork for a truly resilient system configuration for everything from Infrastructure as Code deployments to personal workstations.

Here's an example NixOS system configuration file

```
# configuration.nix
# An example workstation configuration
{ config, lib, pkgs, ... }:
{
  imports = [ ./hardware-configuration.nix ./additional-config.nix ];

  boot.loader.systemd-boot.enable = true; # Enables the systemd bootloader

  networking = { # Enables networking using network manager
    hostName = "nixos";
    networkmanager.enable = true;
  };
```

```
  time.timeZone = "America/New_York";

  users.users.sales = { # Creates a new user named sales
    createHome = true;
    extraGroups = [ "networkmanager" ];
  };

  system.stateVersion = "24.11";
}
```

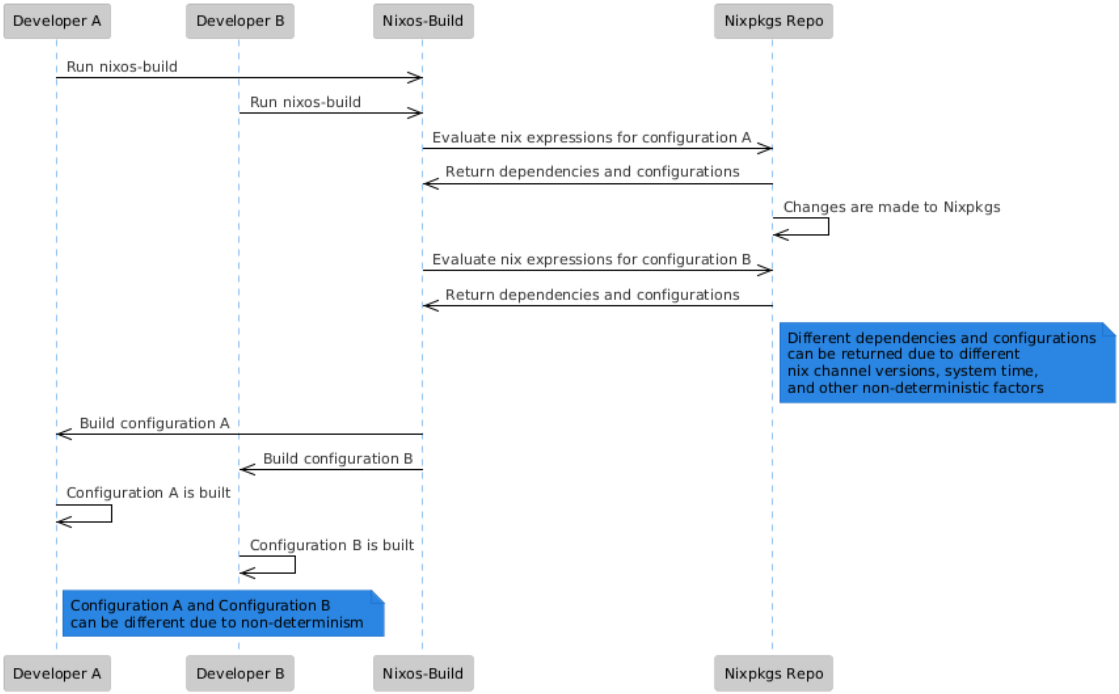
All it takes to rebuild this system is putting the above Nix code into `/etc/nixos/configuration.nix` and executing `nixos-rebuild switch`. Yup, that's it! No need to mess around with `apt`, `dnf` or touch any config files. The Nix package manager will evaluate your `configuration.nix` file using the attributes (variables) you defined, compare it with the Nixpkgs repository to generate a build plan, and finally pulling pre-built binary packages from cache.nixos.org. If you break your system config, no worries just select your previous system configuration from the boot menu:

All of NixOS' available options and packages can be found in https://search.nixos.org, and you can even follow along and see what each option does in the nixpkgs GitHub repository. I have also uploaded my own workstation's super fancy configuration onto Github which you can find here.

Well actually...

While using nixos-rebuild to build a system configuration can help ensure consistency, it does not guarantee consistency of a system's state across machines, as the build process is influenced by the state of the Nix package repository itself when the command was executed.

> For example, if system A builds Firefox -> changes are committed tofirefox in the Nixpkgs repository -> system B builds Firefox, the two systems are looking at two different versions of the Nixpkgs repository and will end up installing two different versions of Firefox.



This problem is solved by using Nix flakes, which allows you to write Nix code whose dependencies are version-pinned in a `flake.lock` file. Flakes also allow you to define a variety of options such as the previously mentioned NixOS configuration, a development shell for a reproducible development environment, build checks, etc. (see the flake schema for options). As this is a big topic, I will have to write a separate post regarding Nix flakes another time.

So, in summary,

During my time learning and using NixOS I've honestly been having a blast. It has been a refreshing experience to work with a Linux distribution that provides for reproducibility, declarative configuration, and a strong focus on functional programming principles. I highly recommend Docker/Podman enthusiasts, DevOps engineers and experienced Linux users to give NixOS a whirl. Like a new codebase, NixOS will take some time to get used to, but your future self that's breaking or upgrading your system will thank you.

Hi there, I'm Gentman Tan!

I'm a passionate IT and software engineer with a strong background in Linux and open-source

I'm always tinkering with Linux distributions, making things hands on and learning new ideas and methodologies. I'm always eager to learn and collaborate with like-minded individuals who share my passion for technology and innovation.

Contact

Feel free to reach out to me!

Email | LinkedIn | SimpleX

My Resume

Hi there, welcome to grok.zone!

This is a personal wiki I use to write interesting things down. It's organized into shelves and books like a physical library.

Browsing the Zone

Read the blog

Recently Updated

Browse by topic

About Me

Hi I'm Gentman Tan, essentially I'm a technophile.

Profile Page