About me: My name is *Solène Rapenne*, pronouns she/her. I like learning and sharing knowledge. Hobbies: '(BSD OpenBSD Qubes OS Lisp cmdline gaming security QubesOS internet-stuff). I **love** percent and lambda characters. Qubes OS core team member, former OpenBSD developer solene@. No AI is involved in this blog.

Contact me: *solene at dataswamp dot org* or *@solene@bsd.network* (mastodon).

I'm a freelance OpenBSD, FreeBSD, Linux and Qubes OS consultant, this includes DevOps, DevSecOps, technical writing or documentation work. If you enjoy this blog, you can sponsor my open source work financially so I can write this blog and contribute to Free Software as my daily job.

Harden your NixOS workstation

Written by *Solène*, on 13 January 2022.
Tags: #nix #nixos #security

Comments on Fediverse/Mastodon

Table of contents

1. Introduction §

Coming from an OpenBSD background, I wanted to harden my NixOS system for better security. As you may know (or not), security mitigations must be thought against a security threat model. My model here is to prevent web browsers to leak data, prevent services to be exploitable remotely and prevent programs from being exploited to run malicious code.

NixOS comes with a few settings to improve in these areas, I'll share a sample of configuration to increase the default security. Unrelated to security defense itself, but you should absolutely encrypt your filesystem, so in case of physical access to your computer no data could be extracted.

2. Use the hardened profile §

There are a few profiles available by default in NixOS which are files with a set of definitions and one of them is named "hardened" because it enables many security measures.

Link to the hardened profile definition

Here is a simplified list of important changes:

- use the hardened Linux kernel (different defaults and some extra patches from https://github.com/anthraxx/linux-hardened/)

- use the memory allocator "scudo", protecting against some buffer overflow exploits
- prevent kernel modules to be loaded after boot
- protect against rewriting kernel image
- increase containers/virtualization protection at a performance cost (L1 flush or page table isolation)
- apparmor is enabled by default
- many filesystem modules are forbidden because old/rare/not audited enough
- many other specific tweaks

Of course, using this mode will slightly reduce the system performance and may trigger some runtime problems due to the memory management being less permissive. On one hand, it's good because it allows to catch programming errors, but on the other hand it's not fun to have your programs crashing when you need them.

With the scudo memory allocator, I have troubles running Firefox, it will only start after 2 or 3 crashes and then will work fine. There is a less permissive allocator named graphene-hardened, but I had too much troubles running programs with it.

3. Use firewall §

One simple rule is to block any incoming traffic that would connect to listening services. It's way more secure to block everything and then allow the services you know must be open to the outside than relying on the service's configuration to not listen on public interfaces.

4. Use Clamav §

Clamav is an antivirus, and yes it can be useful on Linux. If it can prevent you at least once to run a hostile binary, then it's worth running it.

5. Firejail §

I featured firejail previously on my blog, I'm convinced of its usefulnes. You can run a program using firejail, and it will restrict its permissions and rights so in case of security breach, the program will be restricted.

This is rather important to run web browsers with it because it will prevent them any access to the filesystem except ~/Downloads/ and a few required directories (local profile, /etc/resolv.conf, font cache etc...).

6. Enable this on NixOS §

Because NixOS is declarative, it's easy to share the configuration. My configuration supports both Firefox and Chromium, you can remove the related lines you don't need.

Be careful about the import declaration, you certainly already have one for the ./hardware-configuration.nix file.

```
imports =
  [
     ./hardware-configuration.nix
     <nixpkgs/nixos/modules/profiles/hardened.nix>
  ];

 # enable firewall and block all ports
 networking.firewall.enable = true;
 networking.firewall.allowedTCPPorts = [];
 networking.firewall.allowedUDPPorts = [];

 # disable coredump that could be exploited later
 # and also slow down the system when something crash
 systemd.coredump.enable = false;
```

```
  # required to run chromium
  security.chromiumSuidSandbox.enable = true;

  # enable firejail
  programs.firejail.enable = true;

  # create system-wide executables firefox and chromium
  # that will wrap the real binaries so everything
  # work out of the box.
  programs.firejail.wrappedBinaries = {
      firefox = {
          executable = "${pkgs.lib.getBin pkgs.firefox}/bin/firefox";
          profile = "${pkgs.firejail}/etc/firejail/firefox.profile";
      };
      chromium = {
          executable = "${pkgs.lib.getBin pkgs.chromium}/bin/chromium";
          profile = "${pkgs.firejail}/etc/firejail/chromium.profile";
      };
  };

  # enable antivirus clamav and
  # keep the signatures' database updated
  services.clamav.daemon.enable = true;
  services.clamav.updater.enable = true;
```

Rebuild the system, reboot and enjoy your new secure system.

7. Going further: network filtering §

If you want to absolutely control your network connections, I'd absolutely recommend the service OpenSnitch. This is a daemon that will listen to all the network done on the system and allow you to allow/block connections per executable/source/destination/protocol/many parameters.

OpenSnitch comes with a GUI app called opensnitch-ui which is mandatory, if the ui is not running, no filtering is done. When the ui is running, every time a new connection is not matching an existing rule, you will be prompted with information telling you what executable is trying to do on which protocol with which host, then you can decide how long you allow this (or block).

Just use `services.opensnitch.enable = true;` in the system configuration and run opensnitch-ui program in your graphical session. To have persistent rules, open opensnitch-ui, go in the Preferences menu and tab Database, choose "Database type: File" and pick a path to save it (it's a sqlite database).

From this point, you will have to allow / block all network done on your system, it can be time-consuming at first, but it's user-friendly enough and rules can be done like "allow this entire executable" so you don't have to allow every website visited by your web browser (but you could!). You may be surprised by the amount of traffic done by non networking programs. After some time, the rule set should be able to cope with most of your needs without needing to add new entries.

OpenSnitch wiki: getting started
