**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. OpenBSD developer solene@. No AI is involved in this blog.

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



# OpenBSD workstation hardening

Written by *Solène*, on 31 December 2023.
Tags: #security #openbsd #unix

Comments on Fediverse/Mastodon

## 1. Introduction §

I wanted to share a list of hardening you can do on your OpenBSD workstation, and explaining the threat model of each change.

OpenBSD official project website

Feel free to pick any tweak you find useful for your use-case, many are certainly overkill for most people, but depending on the context, these changes could make sense for others.

## 2. User configuration §

There are some tweaks that could be done in the configuration of a user to improve the security.

## 2.1. The Least privileges §

In order to prevent a program to escalate privileges, remove yourself from the wheel group, and don't set any doas or sudo permission.

If you need root privileges, switch to a TTY using the root user.

## 2.2. Multiple-factor authentication §

In some cases, it may be desirable to have a multiple factor authentication, this mean that in order to log in your system, you would need a TOTP generator (phone app typically, or a password manager such as KeePassXC) in addition to your regular password.

This would protect against people nearby who may be able to guess your system password.

I already wrote a guide explaining how to add TOTP to an OpenBSD login.

Blog post: Multi-factor authentication on OpenBSD

## 2.3. Home directory permission §

The permissions of the user directory should be 700, so only the owner and root could browse it.

Ideally, you should add `umask 077` to your user environment, so every new directory or file permissions will be restricted to your user only.

# 3. Firewall §

There are some interesting policies to configure with the help of OpenBSD firewall Packet Filter.

## 3.1. Block inbound §

By default, it's good practice to disable all incoming traffic except the responses to established sessions (so servers can reply to your requests). This protects against someone on your local network / VPN to access network services that would be listening on the network interfaces.

In `/etc/pf.conf` you would have to replace the default:

```
block return
pass
```

By the following:

```
block all
pass out inet
# allow ICMP because it's useful
pass in proto icmp
```

Then, reload with `pfctl -f /etc/pf.conf`, if you ever need to allow a port on the network, add the according rule in the file.

## 3.2. Filter outbound §

It may be useful and effective to block outbound traffic, but this only work effectively if you know exactly what you need because you will have to allow hosts and remote ports manually.

It would protect against a program trying to exfiltrate data using a non-allowed port/host.

# 4. Disabling network for the desktop user §

Disabling network by default is an important mitigation in my opinion. This will protect against any program your run and try to act rogue, if they can't figure there is a proxy, they won't be able to connect to the Internet.

This could also save you from mistaken commands that would pull stuff from the network like pip, npm and co. I think it's always great to have a tight control on which program should do networking and which shouldn't. On Linux this is actually easy to do, but on OpenBSD we can't restrict a single program so a proxy is the only solution.

This can be done by creating a new user named `_proxy` (or whatever the name you prefer) using `useradd -s /sbin/nologin -m _proxy` and adding your SSH key to its authorized_keys file.

Add this rule at the end of your file `/etc/pf.conf` and then reload with `pfctl -f /etc/pf.conf`:

```
block return out proto {tcp udp} user solene
```

Now, if you want to allow a program to use the network, you need to:

- toggle the proxy ON with the command: `ssh -N -D 10000 _proxy@localhost` which is only possible if your SSH private key is unlocked
- configure a SOCKS5 proxy in the program

### 4.0.1. Some network fixes §

Most programs will react to a proxy configured in a variable named `http_proxy` or `https_proxy` or `all_proxy`, however it's not a good idea to globally define these variables for your user as it would be a lot easier to a program to use the proxy automatically, which is against the essence of this proxy.

#### 4.0.1.1. SSH §

By default, you won't be able to ssh to anything except on a local user, we need to proxy every remote ssh connection through the local _proxy user.

In `~/.ssh/config`:

```
Host localhost
User _proxy
ControlMaster auto
ControlPath ~/.ssh/%h%p%r.sock
ControlPersist 60

Host *.*
ProxyJump localhost
```

#### 4.0.1.2. Chromium §

If you didn't configure GNOME proxy settings, Chromium / Ungoogled Chromium won't use a proxy, except if you add a command line parameter `--proxy-server=socks5://localhost:10000`.

I tried to manually modified the dconf database where the "GNOME" settings are to configure the proxy, but I didn't get it to work (it used to work for me, but I can't succeed anymore).

#### 4.0.1.3. Syncthing §

If you use syncthing, you need to proxy all its traffic through the SSH tunnel. This is done by setting the environment variable `all_proxy=socks5://localhost:10000` in the program environment.

## 5. Live in a temporary file-system §

It's possible to have most of your home directory be a temporary file system living in memory, with a few directories with persistency.

This change would prevent anyone from using temporary files or cache left-over from previous session.

The most efficient method to achieve this is to use the program home-impermanence that I wrote for this use case, it handles a list of files/directories that should be persistent.

Blog post: Reproducible clean $HOME on OpenBSD using impermanence

If you only want to start fresh using a template (that doesn't evolve on use), you can check the flag `-P` of `mount_mfs` which allows populating the fresh memory based file system using an existing directory.

OpenBSD man page: mount_mfs(8)

# 6. Disable webcam and microphone §

Good news! I take the opportunity here to remember OpenBSD disables by default the video and audio recording of the various capable devices, instead, they will appear to work but record empty stream of data.

They can be manually enabled by changing the sysctls `kern.audio.record` or `kern.video.record` to 1 when you need to use them.

Some laptop manufacturer offer the option to have a physical switch to disable microphone and webcam, so you can be confident about their state (Framework). Some other manufacturer also allow to not put any webcam and microphone (NovaCustom, Nitropad). Finally, open source firmwares like Coreboot can offer a bios setting to disable these peripherals, it should be trustable in my opinion.

# 7. Disabling USB ports §

If you need to protect your system from malicious USB devices (usually in an office environment), you should disable them in the BIOS/Firmware if possible.

If it's not possible, then you could still disable the kernel drivers at boot time using this method.

Create the file `/etc/bsd.re-config` and add the content to it:

```
disable usb
disable xhci
```

This will disable the support for USB 3 and 2 controllers. On a desktop computer, you may want to use PS/2 peripherals in these conditions.

# 8. System-wide services §

## 8.1. Clamav antivirus §

While this one may make you smile, if there is a chance it saves you once, I think it's still a valuable addition to any kind of hardening. A downloaded attachment from an email, or rogue JPG file could still harm your system.

OpenBSD ships a fully working clamav service, don't forget to enable freshclam, the viral database updater.

## 8.2. Auto-update §

I already covered it in a previous article about anacron, but in my opinion, auto-updating the packages and base system daily on a computer is the minimum that should be done everywhere.

Anacron: useful OpenBSD examples

# 9. System configuration §

# 9.1. Memory allocation hardening §

The OpenBSD malloc system allows you to enable some extra checks, like use after free, heap overflow or guard pages, they can be all enabled at once. This is really efficient for security as most security exploits relies on memory management issues, BUT it may break software that have memory management issues (there are many of them). Using this mode will also impact the performance negatively, as the system needs to do more checks for each piece of allocated memory.

In order to enable it, add this to `/etc/sysctl.conf`:

```
vm.malloc_conf=S
```

It can be immediately enabled with `sysctl vm.malloc_conf=S`, and disabled by setting no value `sysctl vm.malloc_conf=""`.

The program `ssh` and `sshd` always run with this flag enabled, even if it's disabled system-wide.

# 10. Some ideas to go further §

# 10.1. Specialized proxies §

It could be possible to have different proxy users, with each restriction to the remote ports allowed, we could imagine proxies like:

- http / https / ftp
- ssh only
- imap / smtp
- etc....

Of course, this is even more tedious than the multipurpose proxy, but at least, it's harder for a program to guess what proxy to use, especially if you don't connect them all at once.

# 10.2. Run process using dedicated users §

I wrote a bit about this in the past, for command line programs, running them in dedicated local users over SSH make sense, as long as it's still practical.

Dedicated users to run processes

But if you need to run graphical programs, this becomes tricky. Using `ssh -Y` gives the remote program a full access to your display server, which has access to everything else running, not great... You could still rely on `ssh -X` which enables X11 Security extensions, but you have to trust the implementation, and it comes with issues like no shared clipboard, poor performance and programs crashing when attempting to access a legit resource that is blocked by the security protocol...

In my opinion, the best way to achieve isolation for graphical programs would be to run a dedicated VNC server in the local user, and connect from your own user. This should be better than running on your own X locally.

## 10.3. Encrypted home with USB unlocking §

In a setup where the computer is used by multiple person, the system encryption may be tedious because everyone have to remember the main passphrase, you have no guarantee one won't write it down on a post-it... In that case, it may be better to have a personal volume, encrypted, for each user.

I don't have an implementation yet, but I got a nice idea. Adding a volume for a user would look like the following:

- take a dedicated USB memory stick for this user, this will be used as a "key" to unlock their data directory
- overwrite the memory stick with random data
- create an empty disk file on the system, it will contain the encrypted virtual disk, use a random part of the USB disk for the passphrase (you will have to write down the length + offset)
- write a rc file that looks for the USB disk volume if present, if so, tries to unlock and mount the partition upon boot

This way, you only need to have your USB memory stick plugged in when the system is booting, and it should automatically unlock and mount your personal encrypted volume. Note that if you want to switch user, you would have to reboot to unlock their drive if you don't want to mess with the command line.

# 11. Conclusion §

It's always possible to harden a system more and more, but the balance between real world security and actual usability should always be studied.

No one will use a too-much hardened system if they can't work on it efficiently, on the other hand, users expect their system to protect them against most common threats.

Depending on one's environment and threat model, it's important to configure their system accordingly.

---