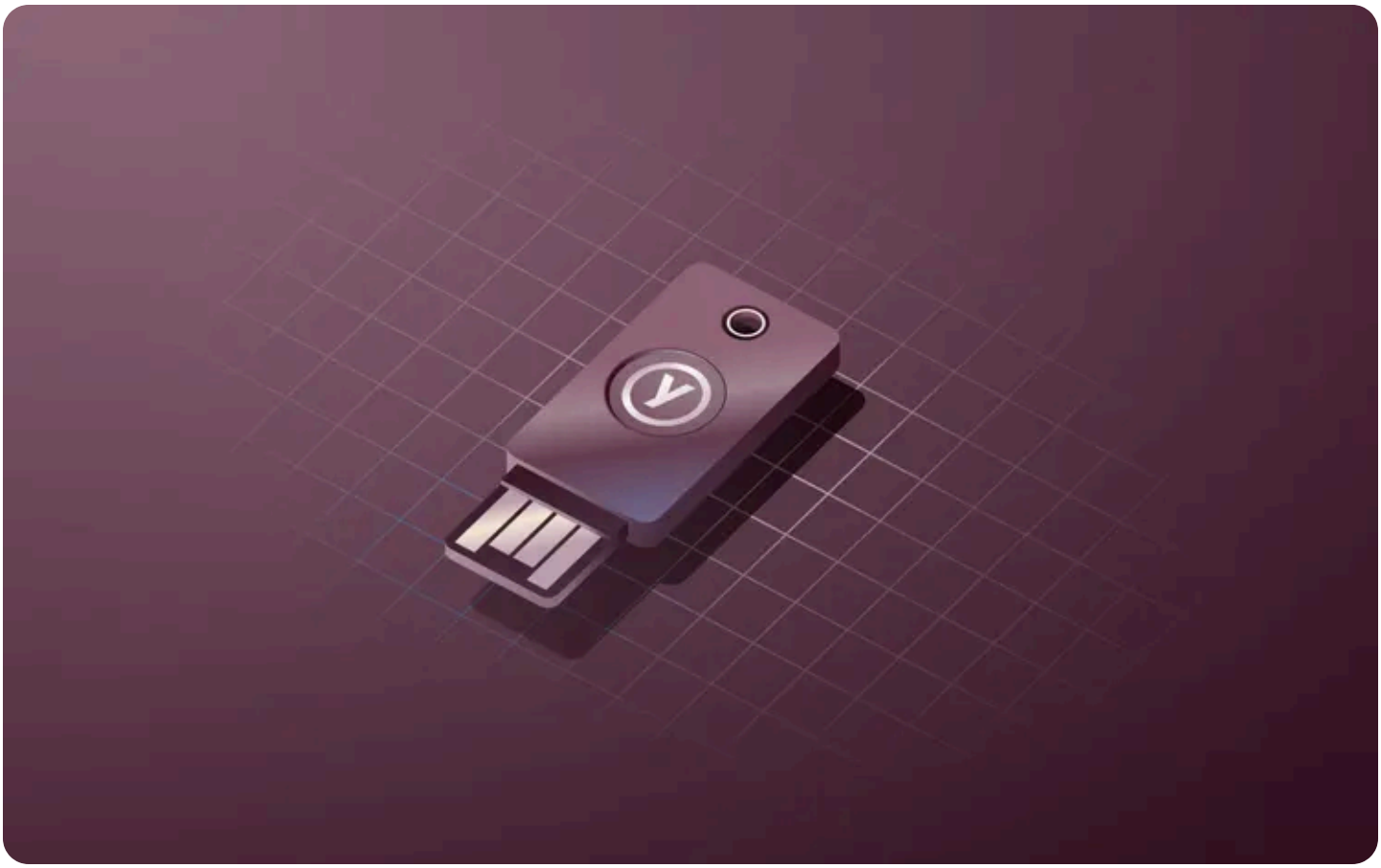
Getting the most out of your Yubikey on NixOS

July 2, 2024 ?? Views • #nix #security



I recently got myself two Yubikey 5 devices. One of them will live on my keychain, while the second one is a backup that doesn't leave my house.

Firefox works out of the box with the Yubikeys and I can add them to various accounts. In addition to using them as 2fa through the browser, I wanted to use them on the OS level as well.

For in-depth details on all of the functionality of the Yubikey, refer to this excellent Reddit post: "What the heck is a Yubikey and why did I buy one?": A user guide

Use cases

There are many different use cases I found for the Yubikeys. I have marked the ones I have succesfully gotten working. Note that I don't necessarily want or endorse the ones I have left undone.

- `pam` auth (sudo/login) ✅
- 2fa for websites (works out of the box) ✅
- GPG keys ✅
- SSH keys ❓
- Git commit signing ✅
- Yubikey based full disk encryption (guide) ❓
- sops secret encryption (waiting for PR) ❌

Pam module

There are two pam modules that implement Yubikey functionality: `yubico-pam` and `pam_u2f`, the latter of which seems to be the more modern and recommended approach. I did try `yubico-pam` but I was unable to get it working.

> *The `pam_u2f` module implements the U2F (universal second factor) protocol. The protocol was initially developed by Yubico, Google and NXP and is nowadays hosted as an open-standard by the FIDO Alliance. All current and most legacy Yubikeys support the U2F protocol making this the preferred way to use Yubikeys for user login.*

I ended up using the `pam_u2f` module. This allows the Yubikey to work as a replacement for sudo/login password. The configuration is also completely declarative and I'm able to enroll my Yubikeys, including the backup key, to another device without actually having all of the keys with me. How is this possible?

1. Connect your Yubikey.
2. Create authorization mappings for your keys. This file works similarly to `ssh/known_hosts` and is not a secret.

```
nix-shell -p pam_u2f
pamu2fcfg -n -o pam://yubi
```

3. The command above will look like it just hangs. You have to touch your Yubikey to continue. You will get a public key for your Yubikey printed into the terminal. This key will be different every time you run the command. Save it somewhere, you will need it later.
4. Repeat steps 1-3 for your other Yubikeys.

Enable `u2f` in your nixos configuration:

```
security.pam.u2f = {
  enable = true;
  interactive = true;
  cue = true;
};
```

- `interactive`: will prompt you with `Insert your U2F device, then press ENTER.`
- `cue`: will print `Please touch the device.` when your action is required.

Hardening

The guides for `pam_u2f` tell you to save the key mappings in `~/.config/u2f_keys`. I would advice you don't do that as I find it a security risk. Somehow I didn't find anyone talking about this flaw online:

Say a bad actor has physical access to your machine (ok this is already a pretty bad situation but it gets worse). They could plug in their Yubikey, run the above commands, and append their keys to your mapping file. Doing so would give them instant escalation to sudo privileges *without knowing your password*. Sounds pretty bad doesn't it!

How to mitigate this? Don't store the key mappings in user writable location. A better place is `/etc/u2f-mapping`. My configuration creates a file in the read-only `/nix/store`.

Sharing

If you run `pamu2fcfg` with the default arguments, the origin will be `pam://$HOSTNAME`. This means the keys will only work on that specific machine (or any other with the same hostname) - not what we want - this is why I have chosen a more global origin: `pam://yubi`. It can be anything you want.

Now that the keys are usable anywhere, they can be embedded into the nixos configuration:

```
security.pam.u2f = {
  origin = "pam://yubi";
  authFile = pkgs.writeText "u2f-mappings" (lib.concatStrings [
    username
    ":<KeyHandle1>,<UserKey1>,<CoseType1>,<Options1>"
    ":<KeyHandle2>,<UserKey2>,<CoseType2>,<Options2>"
  ]);
};
```

I am using the `lib.concatStrings` function to split the keys onto multiple lines, and insert my username (a string value imported from elsewhere) to the beginning of the line. This is only to make it more readable and can be omitted.

Enable for sudo and login

There is only one thing left to do, enable the U2F module for any services you wish. I have enabled it for sudo and login:

```
security.pam.services = {
  login.u2fAuth = true;
  sudo.u2fAuth = true;
};
```

GPG keypair

Add the following to your nixos system configuration:

```
services = {
  pcscd.enable = true;
  udev.packages = [ pkgs.yubikey-personalization ];
};
```

And the following to your `home-manager` configuration. Most of this is not actually required but they are good hardening steps (apparently ...).

`disable-ccid` is crucial to prevent pcscd daemon and gpg-agent conflicts.

```
programs.gpg = {
  enable = true;

  # https://support.yubico.com/hc/en-us/articles/4819584884124-Resolving-G
  scdaemonSettings = {
    disable-ccid = true;
  };

  # https://github.com/drduh/config/blob/master/gpg.conf
  settings = {
    personal-cipher-preferences = "AES256 AES192 AES";
    personal-digest-preferences = "SHA512 SHA384 SHA256";
    personal-compress-preferences = "ZLIB BZIP2 ZIP Uncompressed";
    default-preference-list = "SHA512 SHA384 SHA256 AES256 AES192 AES ZLIB
    cert-digest-algo = "SHA512";
    s2k-digest-algo = "SHA512";
    s2k-cipher-algo = "AES256";
    charset = "utf-8";
    fixed-list-mode = true;
    no-comments = true;
    no-emit-version = true;
    keyid-format = "0xlong";
    list-options = "show-uid-validity";
    verify-options = "show-uid-validity";
    with-fingerprint = true;
    require-cross-certification = true;
```

```
    no-symkey-cache = true;
    use-agent = true;
    throw-keyids = true;
  };
};

services.gpg-agent = {
  enable = true;

  # https://github.com/drduh/config/blob/master/gpg-agent.conf
  defaultCacheTtl = 60;
  maxCacheTtl = 120;
  pinentryPackage = pkgs.pinentry-curses;
  extraConfig = ''
    ttyname $GPG_TTY
  '';
};
```

Follow this guide to create a GPG master key, with 3 different subkeys: signing key, encryption key and authentication key. After generating the keys, back them up on a USB stick. Once you have backups, move the keys over to the Yubikey. This is all explained in great detail in the aforementioned guide.

You can create a shell with all the packages you need to follow the guide (assuming you already deployed the configuration above):

```
nix-shell -p yubikey-manager cryptsetup
```

If you added the public key URL onto the Yubikey, on any computer you plug the key in, you can now fetch the public key like this:

```
gpg --edit-card
gpg/card> fetch
gpg/card> quit

gpg --edit-key $KEYID
gpg> trust
Your decision? 5
Do you really want to set this key to ultimate trust? (y/N) y
```

```
gpg> quit
```

The Yubikey can now be used as your GPG keypair!

Git commit signing

Configure git to use your newly created GPG key to sign commits. Using home-manager:

```
programs.git = {
  userEmail = "same as your key identity"
  signing.key = "$KEYID";
  extraConfig.commit.gpgsign = true;
};
```

Now whenever you `git commit` it will sign with your Yubikey, asking for the pin for the first signing after boot.

Sources

https://nixos.wiki/wiki/Yubikey

https://developers.yubico.com/pam-u2f

https://github.com/drduh/YubiKey-Guide