



nikstur treewide: 0.3.0 -> 0.4.1 ✓

ce9c2a6 · 7 months ago



337 lines (250 loc) · 10.6 KB

Quick Start: NixOS Secure Boot

This document attempts to guide users into setting up UEFI Secure Boot for their NixOS system using a custom key chain. The audience are experienced NixOS users.

This guide has been tested on a Lenovo ThinkPad and is expected to work on other Thinkpads without change. On other systems, certain steps may be different.

⚠ Disclaimers ⚠

Secure Boot for NixOS is still in development and has some sharp edges. There may be cases where you end up with a system that does not boot.

For Windows dual-booters and BitLocker users, it is highly recommended that you export your BitLocker recovery keys and confirm that they are correct. Please refer to this [Microsoft support article](#) for help. This will be required once you finish this guide to confirm with BitLocker that the PCRs changed during the next measurement are intended and allows the TPM unlocking of Windows to work as normal.

We only recommend this to NixOS users that are comfortable using recovery tools to restore their system or have a backup ready.

Functional Requirements

To be able to setup Secure Boot on your device, NixOS needs to be installed in UEFI mode and [systemd-boot](#) must be used as a boot loader. This means if you wish to install lanzaboote on a new machine, you need to follow the install instruction for systemd-boot and then switch to lanzaboote after the first boot.

These prerequisites can be checked via `bootctl status` :

```
$ bootctl status
System:
  Firmware: UEFI 2.70 (Lenovo 0.4720)
  Secure Boot: disabled (disabled)
  TPM2 Support: yes
  Boot into FW: supported

Current Boot Loader:
  Product: systemd-boot 251.7
...
```



In the `bootctl` output, the firmware needs to be `UEFI` and the current boot loader needs to be `systemd-boot` . If this is the case, you are all set to continue.

Security Requirements

These requirements are *optional* for a development system. Feel free to skip them, if you just want to hack on Secure Boot support.

To provide any security your system needs to defend against an attacker turning UEFI Secure Boot off or being able to sign binaries with the keys we are going to generate.

The easiest way to achieve this is to:

1. Enable a BIOS password in your system.
2. Use full disk encryption.

The topic of security around Secure Boot is complex. We are only scratching the surface here and a comprehensive guide is out of scope.

Part 1: Preparing Your System

In the first part, we will prepare everything on the software side of things. At the end of this part, you will have your own Secure Boot keys and a NixOS that has signed boot binaries.

Finding the UEFI System Partition (ESP)

The UEFI boot process revolves around a special partition on the disk. This partition is called *ESP*, the (U)EFI System Partition. This partition is by convention mounted at `/boot` on NixOS and the rest of this document assumes this.

You can verify that `/boot` is the ESP by looking for `ESP:` in `bootctl status` output.

Creating Your Keys

To create Secure Boot keys, we use `sbctl`, a popular Secure Boot Key Manager. `sbctl` is available in [Nixpkgs](#) as `pkgs.sbctl`.

Once you have installed `sbctl` (or entered a Nix shell), creating your Secure Boot keys requires this command:

```
$ sudo sbctl create-keys
[sudo] password for julian:
Created Owner UUID 8ec4b2c3-dc7f-4362-b9a3-0cc17e5a34cd
Creating secure boot keys...✓
Secure boot keys created!
```



This takes a couple of seconds. When it is done, your Secure Boot keys are located in `/etc/secureboot`. `sbctl` sets the permissions of the secret key so that only root can read it.

Configuring NixOS (with `niv`)

Add `lanzaboote` as a dependency of your `niv` project and track a stable release tag (<https://github.com/nix-community/lanzaboote/releases>).

```
$ niv add nix-community/lanzaboote -r v0.4.1 -v 0.4.1
Adding package lanzaboote
Writing new sources file
Done: Adding package lanzaboote
```



Below is a fragment of a NixOS configuration that enables the SecureBoot stack.

```
# file: configuration.nix
{ pkgs, lib, ... }:
let
  sources = import ./nix/sources.nix;
  lanzaboote = import sources.lanzaboote;
in
{
  imports = [ lanzaboote.nixosModules.lanzaboote ];

  environment.systemPackages = [
    # For debugging and troubleshooting Secure Boot.
    pkgs.sbctl
  ];

  # Lanzaboote currently replaces the systemd-boot module.
  # This setting is usually set to true in configuration.nix
  # generated at installation time. So we force it to false
```



```
# for now.
boot.loader.systemd-boot.enable = lib.mkForce false;

boot.lanzaboote = {
  enable = true;
  pkiBundle = "/etc/secureboot";
};
}
```

Configuring NixOS (with Flakes)

Below is a fragment of a NixOS configuration that enables the Secure Boot stack.

```
{
  description = "A SecureBoot-enabled NixOS configurations";

  inputs = {
    nixpkgs.url = "github:NixOS/nixpkgs/nixos-unstable";

    lanzaboote = {
      url = "github:nix-community/lanzaboote/v0.4.1";

      # Optional but recommended to limit the size of your system closure.
      inputs.nixpkgs.follows = "nixpkgs";
    };
  };

  outputs = { self, nixpkgs, lanzaboote, ... }: {
    nixosConfigurations = {
      yourHost = nixpkgs.lib.nixosSystem {
        system = "x86_64-linux";

        modules = [
          # This is not a complete NixOS configuration and you need to refe
          # your normal configuration here.

          lanzaboote.nixosModules.lanzaboote

          ({ pkgs, lib, ... }: {

            environment.systemPackages = [
              # For debugging and troubleshooting Secure Boot.
              pkgs.sbctl
            ];

            # Lanzaboote currently replaces the systemd-boot module.
            # This setting is usually set to true in configuration.nix
            # generated at installation time. So we force it to false
            # for now.
            boot.loader.systemd-boot.enable = lib.mkForce false;
          })
        ]
      }
    }
  }
}
```



```
boot.lanzaboote = {  
  enable = true;  
  pkiBundle = "/etc/secureboot";  
};  
})  
];  
};  
};  
};  
}
```

Checking that your machine is ready for Secure Boot enforcement

After you rebuild your system, check `sbctl verify` output:

```
$ sudo sbctl verify  
Verifying file database and EFI images in /boot...  
✓ /boot/EFI/BOOT/BOOTX64.EFI is signed  
✓ /boot/EFI/Linux/nixos-generation-355.efi is signed  
✓ /boot/EFI/Linux/nixos-generation-356.efi is signed  
X /boot/EFI/nixos/0n01vj3mq06pc31i2yhxndvhv4kwl2vp-linux-6.1.3-bzImage.efi  
✓ /boot/EFI/systemd/systemd-bootx64.efi is signed
```



It is expected that the files ending with `bzImage.efi` are *not* signed.

Part 2: Enabling Secure Boot

Now that NixOS is ready for Secure Boot, we will setup the firmware. At the end of this section, Secure Boot will be enabled on your system and your firmware will only boot binaries that are signed with your keys.

At least on some ASUS boards and others, you may also need to set the `OS Type` to "Windows UEFI Mode" in the Secure Boot settings, so that Secure Boot does get enabled.

These instructions are specific to ThinkPads and may need to be adapted on other systems.

Entering Secure Boot Setup Mode


The UEFI firmware allows enrolling Secure Boot keys when it is in *Setup Mode*.

On a Thinkpad enter the BIOS menu using the "Reboot into Firmware" entry in the `systemd-boot` boot menu. Once you are in the BIOS menu:

1. Select the "Security" tab.
2. Select the "Secure Boot" entry.
3. Set "Secure Boot" to enabled.
4. Select "Reset to Setup Mode".

When you are done, press F10 to save and exit.

You can see these steps as a video [here](#).

 Do not select "Clear All Secure Boot Keys" as it will drop the Forbidden Signature Database (dbx).

Framework-specific: Enter Setup Mode

On Framework laptops (13th generation or newer) you can enter the setup mode like this:

1. Select "Administer Secure Boot"
2. Select "Erase all Secure Boot Settings"

When you are done, press F10 to save and exit.

Other systems

On certain systems (e.g. ASUS desktop motherboards), there is no explicit option to enter Setup Mode. Instead, choose the option to erase the existing Platform Key.

Enrolling Keys

Once you've booted your system into NixOS again, you have to enroll your keys to activate Secure Boot. We include Microsoft keys here to avoid boot issues.

```
$ sudo sbctl enroll-keys --microsoft
Enrolling keys to EFI variables...
With vendor keys from microsoft...✓
Enrolled keys to the EFI variables!
```



 During boot, some hardware might include OptionROMs signed with Microsoft keys. During the boot process, we enroll the Microsoft OEM

[lanzaboote](#) / [docs](#) / QUICK_START.md

[↑](#) Top

Preview

Code

Blame

Raw



You can now reboot your system. After you've booted, Secure Boot is activated and in user mode:

```
$ bootctl status
```

System:

Firmware: UEFI 2.70 (Lenovo 0.4720)

Firmware Arch: x64

Secure Boot: enabled (user)

TPM2 Support: yes

Boot into FW: supported



⚠ If you used `--microsoft` while enrolling the keys, you might want to check that the Secure Boot Forbidden Signature Database (dbx) is not empty. A quick and dirty way is by checking the file size of `/sys/firmware/efi/efivars/dbx-*`. Keeping an up to date dbx reduces Secure Boot bypasses, see for example: https://uefi.org/sites/default/files/resources/dbx_release_info.pdf.

Framework-specific: Enable Secure Boot

On Framework laptops you may need to manually enable Secure Boot:

1. Select "Administer Secure Boot"
2. Enable "Enforce Secure Boot"

When you are done, press F10 to save and exit.

That's all! 🥳

Disabling Secure Boot and Lanzaboote

When you want to permanently get back to a system without the Secure Boot stack, **first** disable Secure Boot in your firmware settings. Then you can disable the Lanzaboote related settings in the NixOS configuration and rebuild.

You may need to clean up the `EFI/Linux` directory in the ESP manually to get rid of stale boot entries. **Please backup your ESP, before you delete any files** in case something goes wrong.

Alternatives

The [ArchLinux wiki](https://wiki.archlinux.org/title/Secure_Boot) contains alternatives to handling your keys, in case `sbctl` is not flexible enough.