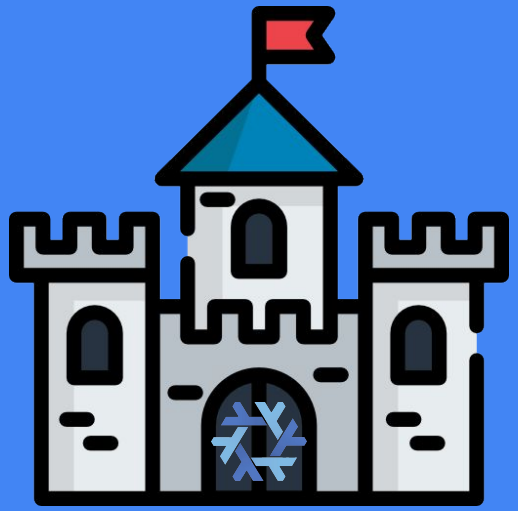


Fortifying the Foundations: Elevating Security in Nix and NixOS

Dominic Mills-Howell

What’s this talk about?

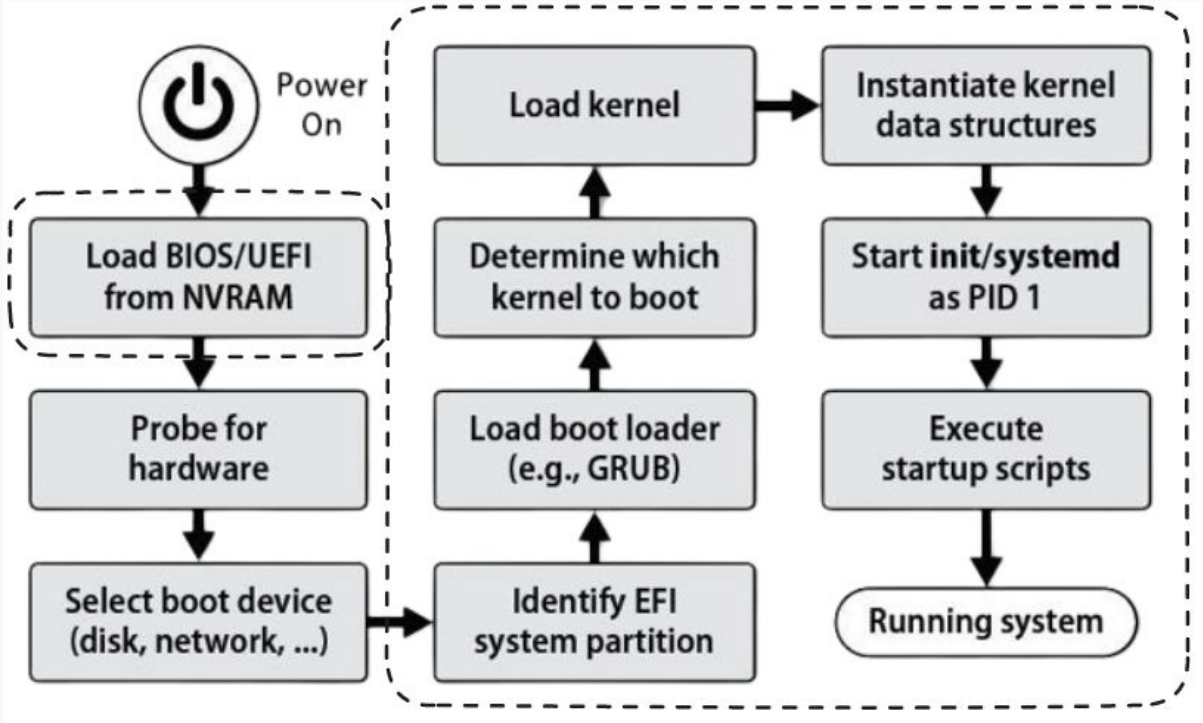
- ➔ Features implemented during the Sovereign Tech Fund’s (STF) Contribute Back Challenge 2023.
- ➔ STF supports the development, implementation and maintenance of open digital infrastructure
- ➔ Nix was one of the nine selected projects in 2023

Increasing security within the Nix ecosystem

Our goal was to address three aspects with the Nix ecosystem, namely:

- Ensuring a proper boot security chain for NixOS.
- Tracking security issues in Nixpkgs.
- Bootstrapping Nixpkgs from very small auditable binaries.

Boot Security in NixOS



The Secure Boot Story in NixOS

Boot Firmware Initialisation + UEFI Secure Boot

Boot Loader Verification + Chain of Trust

Operating System, Kernel and System File Verification

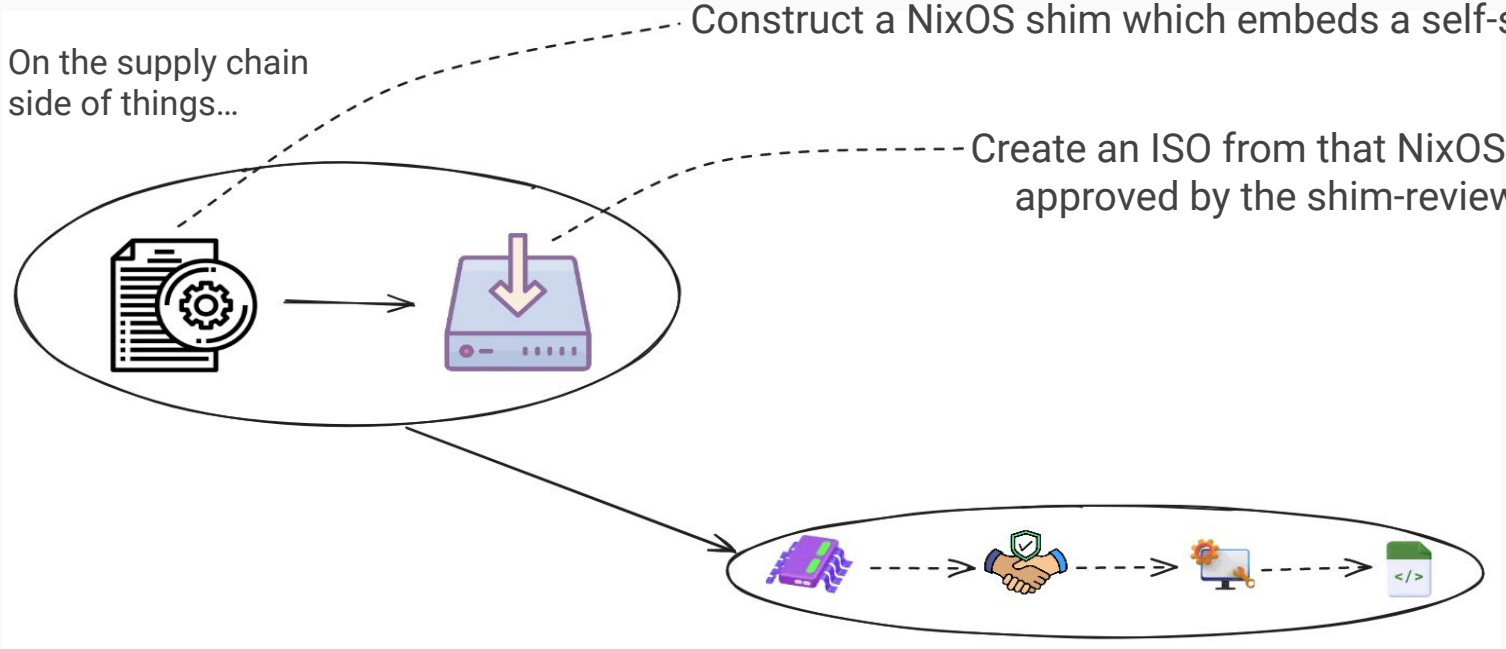
Secure Boot Policies

UEFI Secure Boot by default for NixOS

On the supply chain side of things…

Construct a NixOS shim which embeds a self-signed certificate

Create an ISO from that NixOS shim that was approved by the shim-review committee



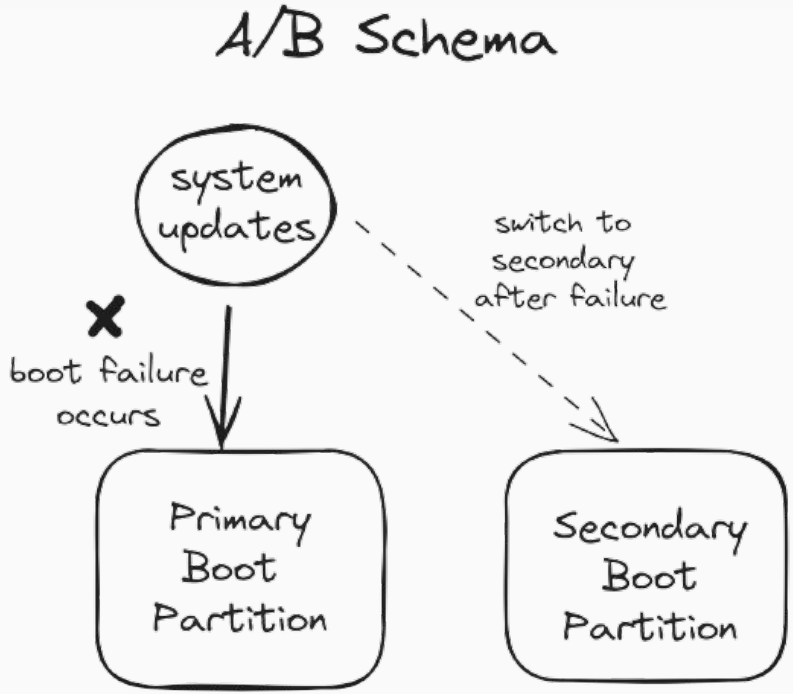
BootSpec V1

```
{
    "org.nixos.bootspec.v1": {
      "system": "x86_64-linux",

      "init": "/nix/store/xxx-nixos-system-xxx/init",

      "initrd": "/nix/store/xxx-initrd-linux/initrd",

      "initrdSecrets": "/nix/store/xxx-append-secrets/bin/append-initrd-secrets",

      "kernel": "/nix/store/xxx-linux/bzImage",

      "kernelParams": [
        "amd_iommu=on",
        "amd_iommu=pt",
        "iommu=pt",
        "kvm.ignore_msrs=1",
        "kvm.report_ignored_msrs=0",
        "udev.log_priority=3",
        "systemd.unified_cgroup_hierarchy=1",
        "loglevel=4"
      ],

      "label": "NixOS 21.11.20210810.dirty (Linux 5.15.30)",

      "toplevel": "/nix/store/xxx-nixos-system-xxx",
    },

    "org.nixos.specialisation.v1": {
      "<name>": {
        "org.nixos.bootspec.v1": {

        }
      }
    },
  }
```

BootSpec V2

```
{
    "org.nixos.bootspec.v2": {
      "system": "x86_64-linux",

      "init": "/nix/store/xxx-nixos-system-xxx/init",

      "initrds": [ "/nix/store/xxx-initrd-linux/initrd" ],

      "kernel": "/nix/store/xxx-linux/bzImage",

      "kernelParams": [
        "amd_iommu=on",
        "amd_iommu=pt",
        "iommu=pt",
        "kvm.ignore_msrs=1",
        "kvm.report_ignored_msrs=0",
        "udev.log_priority=3",
        "systemd.unified_cgroup_hierarchy=1",
        "loglevel=4"
      ],

      "label": "NixOS 21.11.20210810.dirty (Linux 5.15.30)",

    "toplevel": "/nix/store/xxx-nixos-system-xxx",

    "fdtdir": "/nix/store/xxx-uboot-fdtdir",

    "devicetree": "/nix/store/xxx-arm64-machine/my-device.dtb"
  },

    "<name>": {
      "org.nixos.bootspec.v2": {

      }
    }
  },
  "org.nixos.initrd-secrets.v1": {
    "my-private-key": "/etc/nixos/secrets/wireguard-private-key",
  }
}
```

Automatic Boot Assessment in NixOS



Automatic Boot Assessment in NixOS

Automatic Boot Assessment generalises the A/B schema concept to NixOS generations; it leverages systemd-boot to do so.

Mechanism:

- Boot entries initially designated as "indeterminate" with a specific number of boot attempts.
- A counter is decremented with each attempt; if successful, an entry is labeled as "good."
- Entry marked as "bad" if designated boot attempts exhausted without success.

Automatic Boot Assessment Example

```
boot.loader.systemd-boot.bootCounting.enable = true;
services.nsd.enable = true;

systemd.services.nsd = {
  before = [ "boot-complete.target" ];
  wantedBy = [ "boot-complete.target" ];
  unitConfig.FailureAction = "reboot";
};
```

Integrity Checks for the Nix Store

When implementing Secure Boot and transitioning to Stage 2, limitations arise in verifying the operating system closure stored in the Nix store.

Design Considerations for Integrity Checks

Tradeoffs in tools like dm-verity, fs-verity, IMA, and EVM for system integrity.

- dm-verity's read-only nature and simultaneous update poses challenges.
- fs-verity lacks file position verification, posing security risks.

Design Considerations for Integrity Checks

Chosen Design:

- Nix's own `nix-store verify` for system closure verification.
- Relative simplicity without compromising security.
- Easy build and deployment due to being a feature of Nix.

Enabling Integrity Check Options

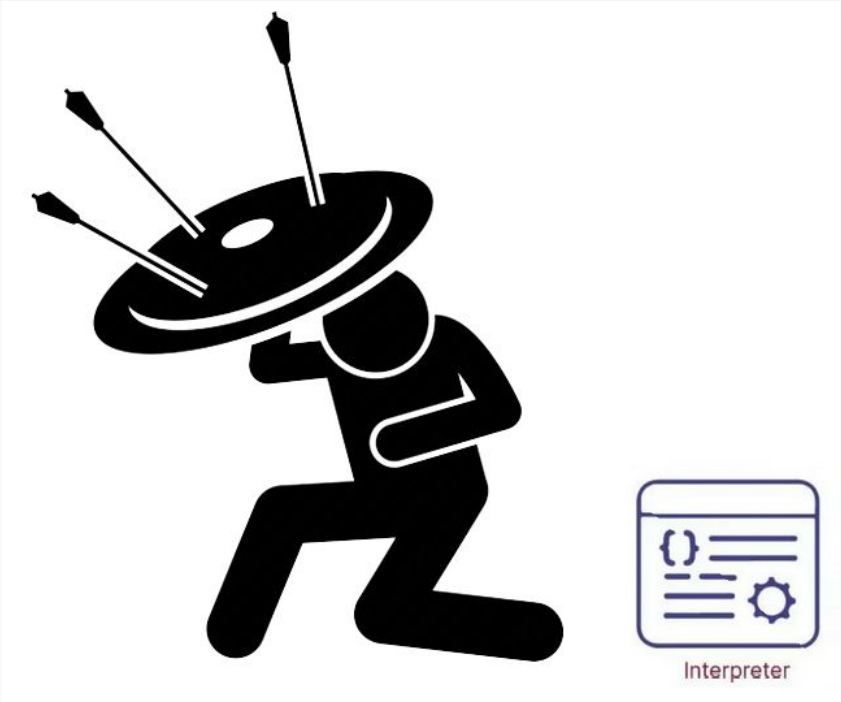
Generate a signing key:

```
$ nix-store --generate-binary-cache-key -key-name secret-key-file public-key-file
```

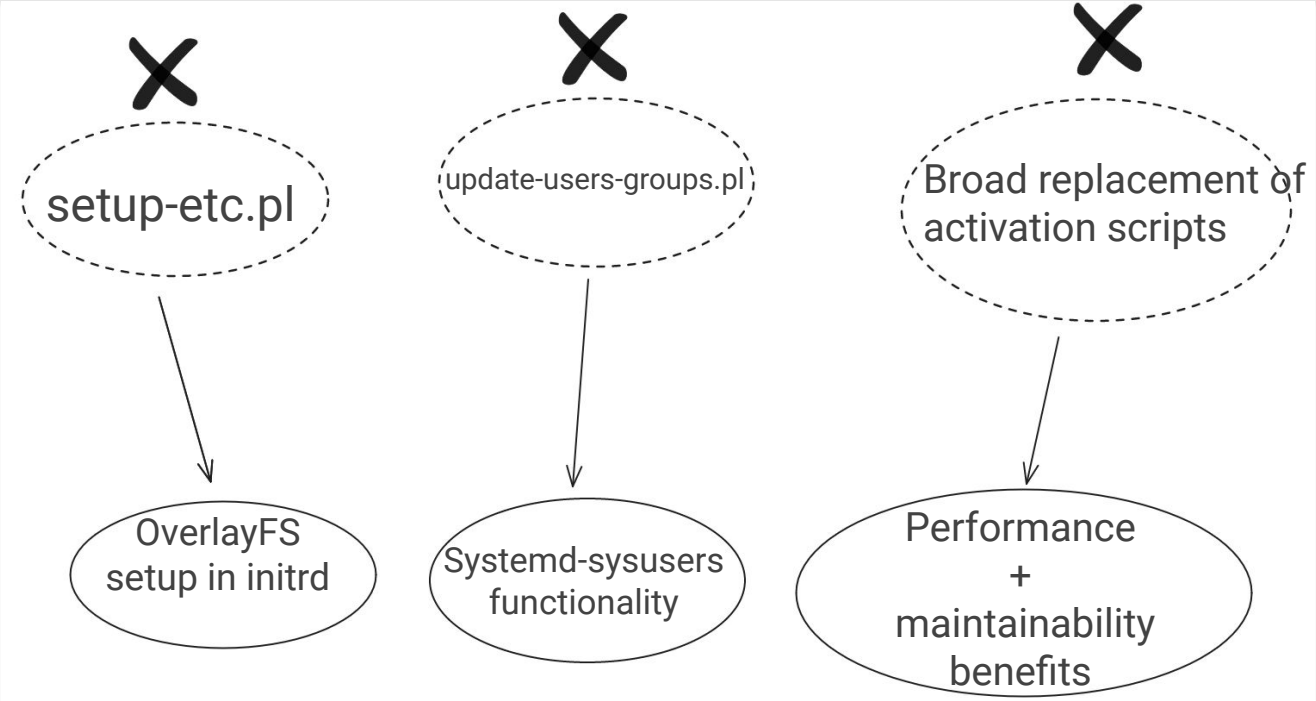
In a NixOS module, configure stage 2 verification.

```
{
  boot.initrd.verify = {
    enable = true;
    sigsNeeded = 1;
    trustedPublicKeys = [(builtins.readFile ./public-key-file)];
  };
}
```

Interpreters in NixOS - Vulnerabilities and Mitigation



Interpreters in NixOS - Vulnerabilities and Mitigation



Any questions?

Feel free to reach out!

Dominic Mills-Howell

dominic@palisadoes.org

@DMills27