Image-based Linux with systemd

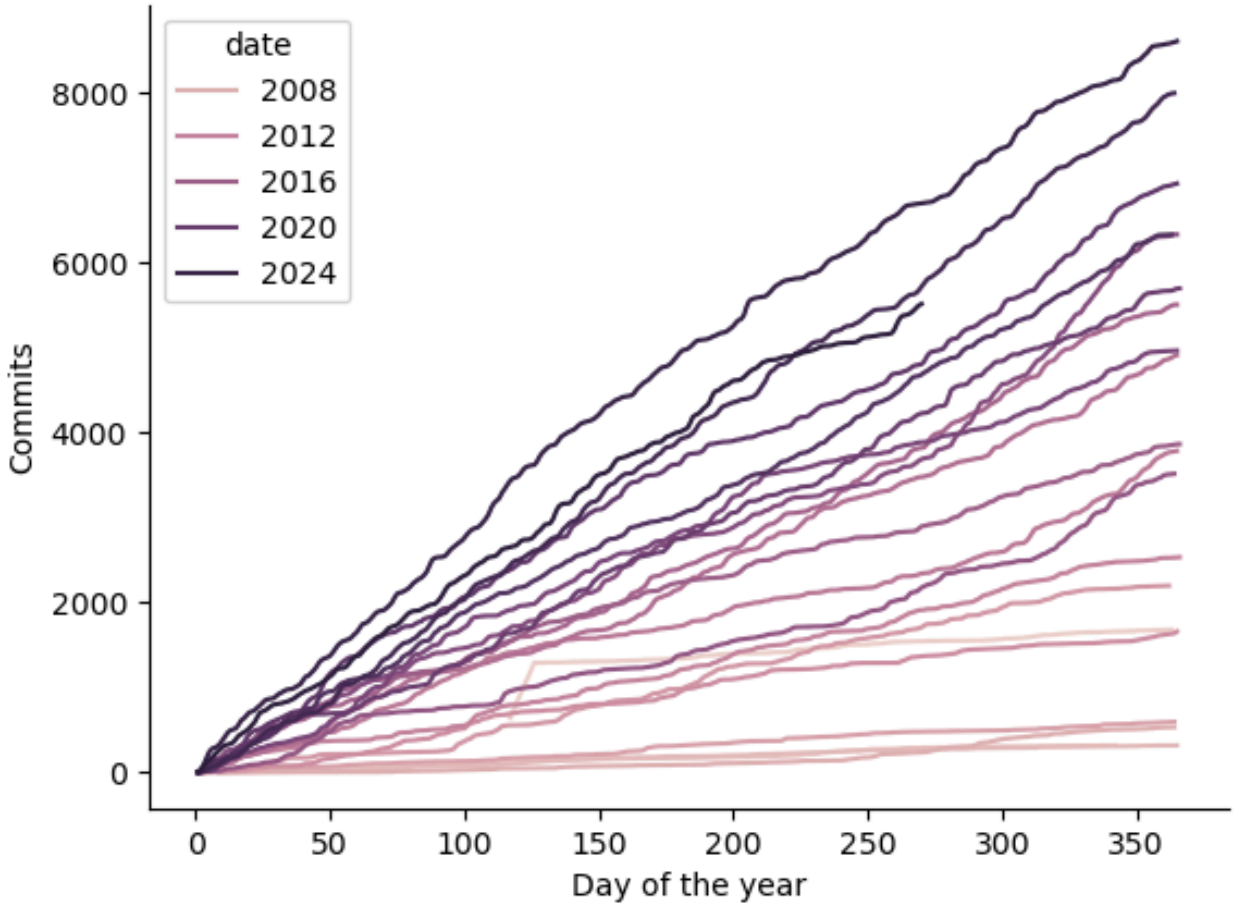
Zbigniew Jędrzejewski-Szmek

Red Hat

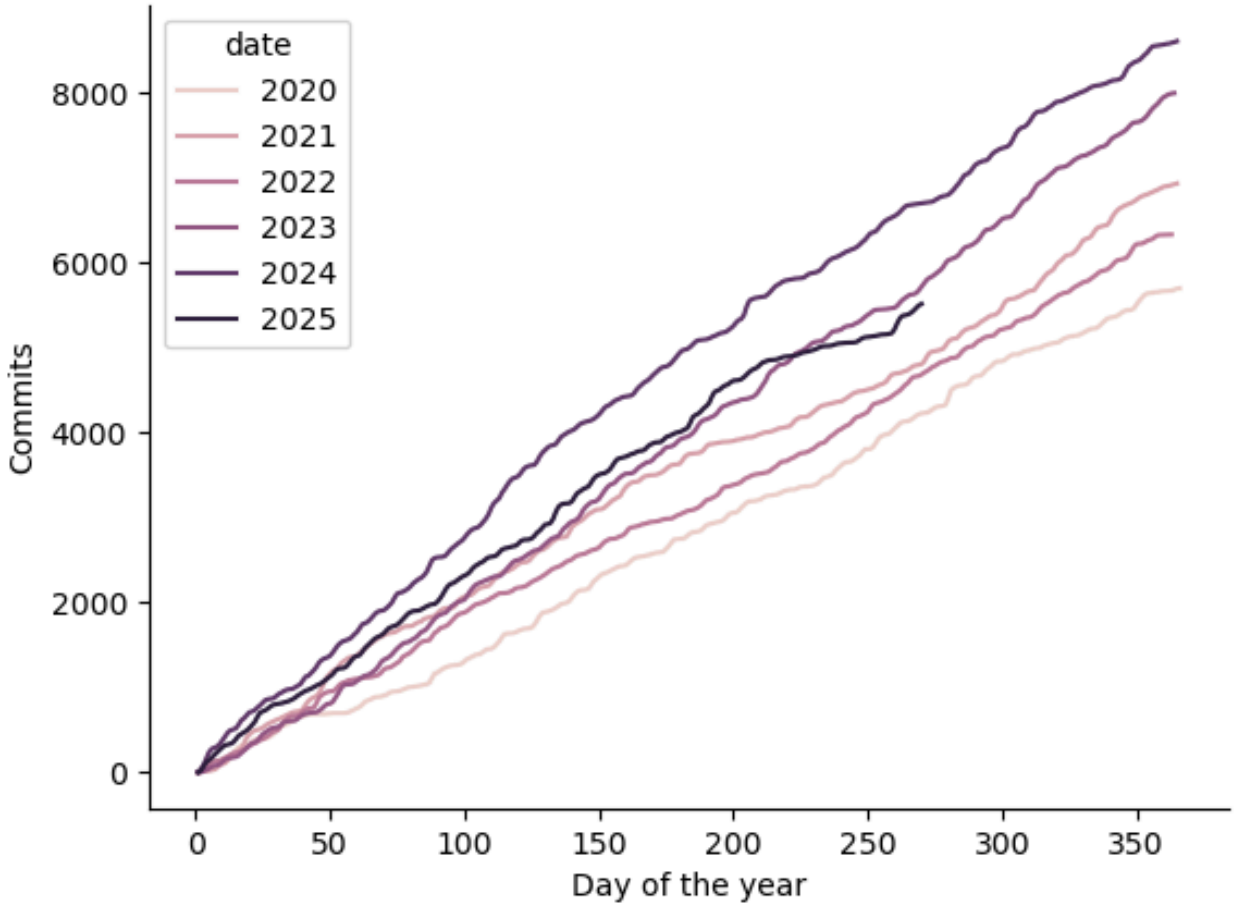
Jesień Linuksowa, 25 października 2025, Rybnik

What have systemd developers been up to?

What have systemd developers been up to?



What have systemd developers been up to?



What have systemd developers been up to?

- few new features directed at “traditional system adminstration”
- a lot of new code for “image-based systems”

Linux systems in 2025

always connected ,
untrusted physical environment *and* untrusted payloads
=> always exposed

traditional installations: mutable package-based system

- each system is a snowflake
- changes are non-atomic
- modifications are persistent

Why image-based systems?

want to verify authenticity *at runtime*

=> signed

=> immutable

=> image-based

What have systemd developers been up to?

New(-ish) components

`systemd-pcrextend`, `systemd-creds`, `systemd-homed`, `systemd-cryptenroll`, `systemd-cryptsetup`, `systemd-repart`, `systemd-measure`, `systemd-pcrlock`, `systemd-dissect`, `systemd-tpm2-setup`, `systemd-tpm2-clear`, `systemd-veritysetup`, `systemd-gpt-auto-generator`, `systemd-boot`, `systemd-stub`, `systemd-sysupdate`, `ukify`, `mkosi`

What needs to be done?

1. build an image
2. install image to device
3. when booting: verify before running
4. when booting: measure what is run
5. when booting: unlock secrets if trusted
6. download new update && install
7. protect user data
8. customize the installation
9. reset system to factory defaults

Building images: mkosi

“make operating system image”

see mkosi: Building Bespoke Operating System Images @ ASG 2023
systemd-repart: Building Discoverable Disk Images @ ASG 2023

Building images: why mkosi?

- builds images from packages
- developed in tandem with systemd
- lightweight: heavy lifting is delegated to `systemd-repart`, `dnf`/`apt`/`pacman`/`apk`, `mkfs.btrfs`/`mkfs.ext4`/`mkfs.xfs`, `systemd-firstboot` and other tools
- systemd-style configuration (ini files, drop-ins, match sections)
- fully offline (no VMs, no loopback devices, plain file IO)
- Fedora/Debian/Kali/Ubuntu/PostmarketOS/ArchLinux/OpenSUSE/Mageia/CentOS/RHEL/RHEL-UBI/OpenMandriva/Rocky/Alma/AzureLinux

Why mkosi?, part II

mkosi-initrd: “build initrds from distro packages”

We reuse the same code for the initrd, the OS, the exitrd.

DEMO!

mkosi + OBS

(Open Build Service @ SUSE)

mkosi is “local first”

goal: IBaaS (image builds as a service)

- customization via declarative config
- multiple distros
- multiple architectures
- automated rebuilds
- secure signing

What needs to be done?

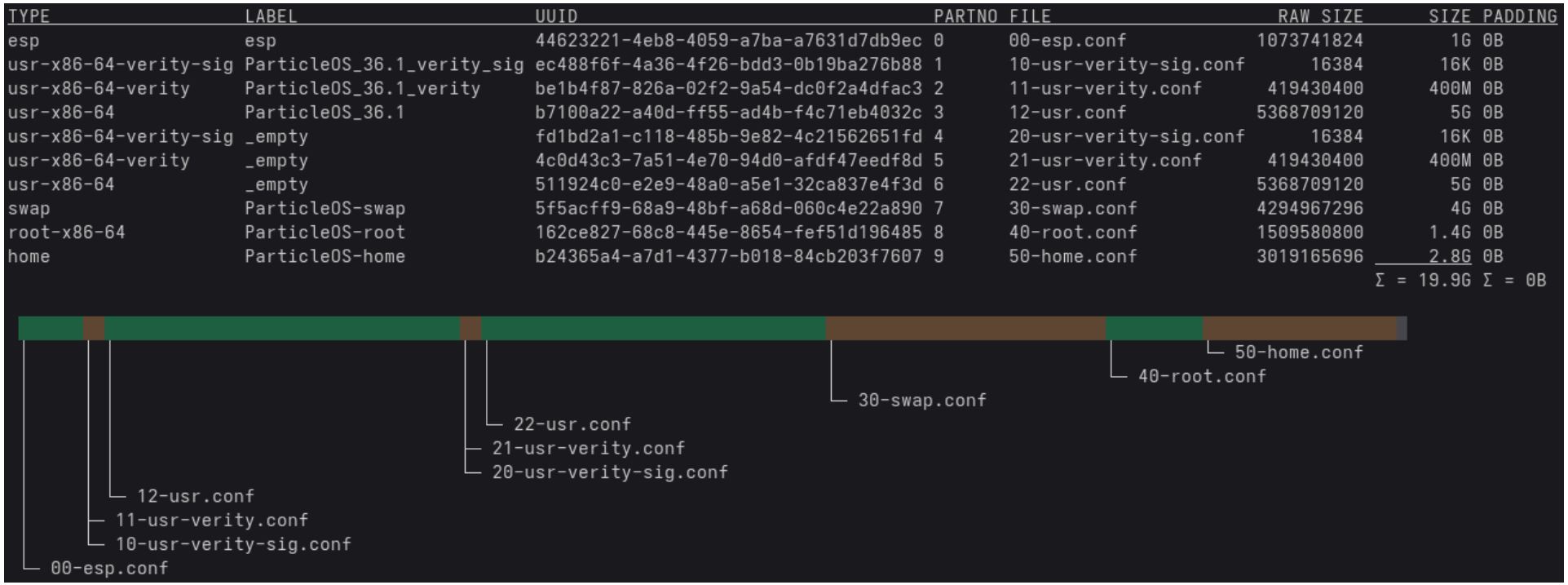
1. ~~build an image~~
2. install image to device
3. when booting: verify before running
4. when booting: measure what is run
5. when booting: unlock secrets if trusted
6. download new update && install
7. protect user data
8. customize the installation
9. reset system to factory defaults

How to install?

- the system is “immutable”
- … but `dd` is not enough!
  (GPT sector size, sector alignment, partition UUIDs)

`systemd-repart`

“online and offline partitioning based on declarative data”



How to install?

Our answer: copy partition *contents*, initialize the rest on the fly

- `systemd-repart` with `CopyBlocks=auto`: copy read-only partitions, dm-verity and signatures
  initialize partition structure for system A/B updates, encrypted storage
- `systemd-cryptsetup`: lock the encrypted storage to TPM

How to install?

Installers used to be very complex => better to do configuration in the live system

Features/InitialExperience (Fedora 19)
Changes/ReduceInitialSetupRedundancy (Fedora 28)
Changes/Unified_KDE_OOBE (Fedora 44)

`systemd-firstboot`

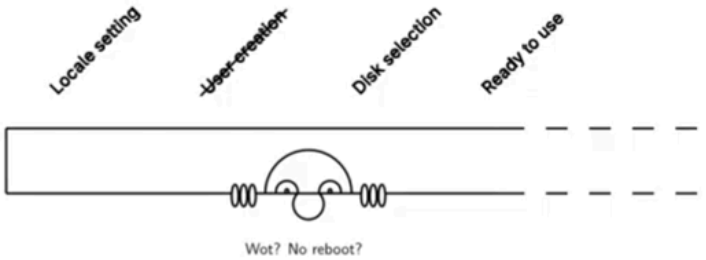
DEMO!

How to install?

Do we **need** to reboot?

GNOME OS' prêt-à-booter image @ ASG2025

Installation: GNOME OS way



(youtube)

What needs to be done?

1. ~~build an image~~
2. ~~install image to device~~
3. when booting: verify before running
4. when booting: measure what is run
5. when booting: unlock secrets if trusted
6. download new update && install
7. protect user data
8. customize the installation
9. reset system to factory defaults

Verify before running

Traditional SecureBoot with shim which checks `vmlinuz` but not the initrd or the kernel command line is security theater.

UKI — Unified Kernel Image
`=>` a single file with the kernel, initrd, command line, device tree, microcode
signed as a whole

`ukify` builds UKI images — `objcopy` on steroids

SecureBoot and measured boot are available to anyone

- real deployements would use a “proper” key
- for development, we make our own

`systemd-boot` supports autoenrollment

- put files in `/loader/keys/auto/` (`KEK.auth`, `PK.auth`, `db.auth`, `dbx.auth`)
- put machine in SB setup mode
- set `secure-boot-enroll if-safe`
- set `secure-boot-enroll-action reboot`

SecureBoot vs. measured boot

SecureBoot: hardware manufacturers embed their own key
-> those keys sign the main Microsoft key
-> that signs … many shims for Linux
-> that trusts distro artifacts

- a **very** wide tent

measured boot: TOFU, unlock secrets if state is good, establish trust

tl;dr: generate own key, use `ukify` to build signed UKIs, enroll key on first boot

What needs to be done?

1. ~~build an image~~
2. ~~install image to device~~
3. ~~when booting: verify before running~~
4. when booting: measure what is run
5. when booting: unlock secrets if trusted
6. download new update && install
7. protect user data
8. customize the installation
9. reset system to factory defaults

Measure what is run

UEFI measures shim,
shim measures the boot loader,
the boot loader measures the UKI

```
systemd-pcrextend --machine-id/--file-system=/$phase
```

systemd devides runtime into phases:
“enter-initrd”, “leave-initrd”, “sysinit”, “ready”

What needs to be done?

1. ~~build an image~~
2. ~~install image to device~~
3. ~~when booting: verify before running~~
4. ~~when booting: measure what is run~~
5. when booting: unlock secrets if trusted
6. download new update && install
7. protect user data
8. customize the installation
9. reset system to factory defaults

Binding secrets to TPM PCR state

TPM PCRs reflect system state: firmware hash, SecureBoot database hash, boot loader hash, UKI hash…

`systemd-pcrextend` records more state

two major approaches:

1. bind secrets to a TPM PCR digest
2. bind secrets to a public certificate hash signing a TPM PCR digest

Binding secrets to TPM PCR state

`systemd-measure` — precalculate PCR measurements for UKI & phase, sign a policy digest with a private key

`ukify` — gather those signed digests and embed them in the UKI

`systemd-cryptenroll` & `systemd-cryptsetup` — enroll and decrypt a disk via via public key infra

`systemd-creds` — the same for credentials

What needs to be done?

1. ~~build an image~~
2. ~~install image to device~~
3. ~~when booting: verify before running~~
4. ~~when booting: measure what is run~~
5. ~~when booting: unlock secrets if trusted~~
6. download new update && install
7. protect user data
8. customize the installation
9. reset system to factory defaults

Downloading images

(`mkosi` to build the new image)

`systemd-sysupdate` implements a downloader that does A/B updates

`systemd-sysupdated` + `updatectl` a daemon+client service for updates

What needs to be done?

1. ~~build an image~~
2. ~~install image to device~~
3. ~~when booting: verify before running~~
4. ~~when booting: measure what is run~~
5. ~~when booting: unlock secrets if trusted~~
6. ~~download new update && install~~
7. protect user data
8. customize the installation
9. reset system to factory defaults

Protect user data

`systemd-homed` manages home directories with per-user encryption

`homectl`, `pam_systemd_home.so`

What needs to be done?

1. ~~build an image~~
2. ~~install image to device~~
3. ~~when booting: verify before running~~
4. ~~when booting: measure what is run~~
5. ~~when booting: unlock secrets if trusted~~
6. ~~download new update && install~~
7. ~~protect user data~~
8. customize the installation
9. reset system to factory defaults

How to make changes to the shiny new *immutable* system?

First answer: don't

flatpaks for apps

custom images

Second answer: verified extension mechanisms

- initrd variants — multi-profile UKIs
- "addons" → checked via SecureBoot db / shim
- systemd-sysexts
- systemd-confexts → checked via kernel keyring
- credentials → encrypted via TPM|local key

Credentials

- `systemd.hostname` — set basic host information
- `firstboot.locale|keymap|timezone` — set basic host information
- `passwd.hashed-password.$user` / `passwd.shell.$user` — like `usermod`
- `fsck.mode` — new `/forcefsck` and `/fastboot`
- `quotacheck.mode` — new `/forcequotacheck`
- `home.add-signing-key.*` — provision homed user record signing keys
- `home.register.*` — create user automatically at boot
- `systemd.unit-dropin.*` — tweak various units
- `vmm.notify_socket` — establish notification channel to the host
- `ssh.listen` — establish ssh access
- `ssh.authorized_keys.root` — backdoor sshd
- `ssh.ephemeral-authorized_keys-all` — backdoor sshd harder

https://systemd.io/CREDENTIALS/

What needs to be done?

1. ~~build an image~~
2. ~~install image to device~~
3. ~~when booting: verify before running~~
4. ~~when booting: measure what is run~~
5. ~~when booting: unlock secrets if trusted~~
6. ~~download new update && install~~
7. ~~protect user data~~
8. ~~customize the installation~~
9. reset system to factory defaults

Factory reset

“all state, user data, and configuration is wiped”

- “the system” is in `/usr` — everything else can go
- `systemd-factory-reset`
- `systemd-repart --factory-reset`
- `systemd-tpm2-clear`

What needs to be done?

1. ~~build an image~~
2. ~~install image to device~~
3. ~~when booting: verify before running~~
4. ~~when booting: measure what is run~~
5. ~~when booting: unlock secrets if trusted~~
6. ~~download new update && install~~
7. ~~protect user data~~
8. ~~customize the installation~~
9. ~~reset system to factory defaults~~

questions?