Introduction to Linux: **Storage and System Security**

Philipp Fruck & Pius Walter

System Administration

Goal of this Lecture

- How storage is organized: partitions, file systems and mounting
- How to protect data at rest: LUKS, TPM2 and measured boot
- How to snapshot, update and roll back a system
- Mandatory access control and firewalls as additional layers

Why?

- Lost disks and laptops are a common source of data leaks
- Updates are the most effective security measure, but they can break systems
- Snapshots turn risky changes into reversible ones

Block Devices, Partitions and File systems

```
lsblk -f              # devices, FS, UUIDs
sudo fdisk -l         # partition tables
sudo blkid            # UUIDs and types
findmnt               # mounted filesystems
df -hT                # usage and FS type
```

Typical UEFI layout

Partition	Content
ESP	FAT32, /boot/efi, loader
/boot	Kernel, initramfs (optional)
root	Encrypted ext4/btrfs/xfs

Layers (bottom to top)

1. Disk (/dev/nvme0n1)
2. Partition table (GPT) and partitions
3. Optional: LUKS, LVM, RAID
4. File system
5. Mount point

> Always refer to disks by **UUID** or **label**: /dev/sdX names can change.

Common File systems

File system	Typical use	Notable features
ext4	Default on Debian/Ubuntu	Stable, simple, journaling
xfs	Default on RHEL, large files	High performance, cannot be shrunk
btrfs	Fedora, openSUSE	Snapshots, subvolumes, checksums, compression
zfs	Storage servers, BSD	Snapshots, RAID, out-of-tree on Linux
vfat	ESP, USB sticks	No permissions, universal compatibility
tmpfs	`/tmp`, `/run`	Lives in RAM

```
sudo mkfs.ext4 -L data /dev/sdb1   #    destroys existing data
sudo mount /dev/sdb1 /mnt
sudo umount /mnt
```

Mounting at Boot: /etc/fstab

What is /etc/fstab

- Plain text configuration file defining **file systems to mount at boot**

```
# <device>   <mount> <type> <options>  <dump> <fsck>
UUID=1a2b…   /       ext4   defaults   0      1
UUID=3c4d…   /data   ext4   noatime    0      2
```

- Test before rebooting!

```
sudo mount -a
sudo systemctl daemon-reload
```

How systemd uses /etc/fstab

- systemd **generates mount units** for every entry
  - /boot -> boot.mount
  - /mnt/data -> mnt-data.mount
- Benefits:
  - Parallelized mounting (faster)
  - Dependency tracking, e.g. network file systems

A broken fstab entry can drop the system into emergency mode. Use nofail for non-essential disks.

Full Disk Encryption: LUKS

Problem: Anyone with physical access can read an unencrypted disk.

Solution: **LUKS** (dm-crypt) encrypts a block device. The filesystem lives *inside* the encrypted container.

```
#    wipes data
sudo cryptsetup luksFormat /dev/sdb1
sudo cryptsetup open /dev/sdb1 secret
sudo mkfs.ext4 /dev/mapper/secret
sudo mount /dev/mapper/secret /mnt
sudo cryptsetup close secret
```

Unlock at boot: `/etc/crypttab`

```
# <name>  <device>     <keyfile> <options>
secret    UUID=5e6f…   none      luks
```

Then use `/dev/mapper/secret` in `/etc/fstab`

- Multiple key slots: passphrase, recovery key, FIDO2, TPM2
- `sudo cryptsetup luksDump /dev/sdb1`
- Root filesystem: unlocked by the **initramfs** before the system starts

TPM2 and Measured Boot

TPM: Tamper-resistant chip on the mainboard (or in firmware)

- Stores secrets and can **seal** them to a system state
- Contains **PCRs** (Platform Configuration Registers)
  - Can only be *extended*: `PCR = hash(PCR || new value)`

Measured boot: every stage measures the next one before running it

1. Firmware -> PCR 0, 2
2. Secure Boot state and keys -> PCR 7
3. Boot loader / kernel (UKI) -> PCR 4, 11

> The TPM only releases the key if the PCR values match -> modified boot chain = no automatic unlock.

> Unified Kernel Images (UKIs) provide support for measured boot: Look them up if interested!

Inspect

```
systemd-analyze pcrs
sudo tpm2_pcrread sha256
systemd-cryptenroll --tpm2-device=list
```

Enroll the TPM as an additional LUKS key

```
sudo systemd-cryptenroll \
    --tpm2-device=auto --tpm2-pcrs=0+2+7 \
    /dev/nvme0n1p3
```

```
# /etc/crypttab
root UUID=… none tpm2-device=auto
```

> A TPM does not protect against a running, logged-in system. Combine it with a PIN or strong login if the device can be stolen while on.

Snapshots with btrfs

Copy-on-write (CoW) filesystem: snapshots are instant and initially take no extra space.

- **Subvolume**: independently mountable directory tree
- **Snapshot**: subvolume created from another subvolume

```
sudo btrfs subvolume list /
sudo btrfs subvolume create /data
```

```
# read-only snapshot
sudo btrfs subvolume snapshot -r \
    /home /.snapshots/home-$(date +%F)
sudo btrfs subvolume delete \
    /.snapshots/home-2026-11-17
```

Typical Fedora layout (`/etc/fstab`)

```
UUID=… /     btrfs subvol=root,compress=zstd:1 0 0
UUID=… /home btrfs subvol=home,compress=zstd:1 0 0
```

Tooling

- `snapper`, `timeshift`: automatic snapshots + rollback
- openSUSE: snapshot before and after each package transaction
- `btrfs send | btrfs receive`: incremental backups
- `cp --reflink`: instant file copies

> Snapshots on the same disk are **not a backup**.

System Updates

Packages

```
# Debian/Ubuntu
sudo apt update && sudo apt upgrade
# Fedora/RHEL
sudo dnf upgrade --refresh
# Arch
sudo pacman -Syu
```

Firmware: LVFS

Linux Vendor Firmware Service: vendors publish firmware (BIOS, SSD, docking stations) which `fwupd` installs (your device must be supported).

```
fwupdmgr refresh
fwupdmgr get-devices
fwupdmgr get-updates
fwupdmgr update
```

Applications

Don't forget Flatpak/AppImage or custom applications!

Same for stuff installed via `pip`, `npm` and the like.

Safer update strategies

- **Snapshot first**, then update, roll back on failure
- **Image-based/atomic** systems: new system image is written beside the old one
  - Fedora Atomic/Silverblue: `rpm-ostree`, `bootc`
  - NixOS: `nixos-rebuild switch --rollback`
- Boot menu keeps the previous version

> Firmware updates can change TPM measurements -> re-seal (systemd-pcrlock make-policy) or have the recovery key ready.

Mandatory Access Control: SELinux

Adds security labels (contexts) to files, processes and ports

```
# show context
ls -Z
ps -eZ
# get/set SELinux state
getenforce
setenforce {0,1}
```

List and change file context / service port

```
sudo semanage fcontext -l
sudo chcon -t httpd_sys_content_t
/var/www/html/index.html

# -m = modify, -a = append
sudo semanage port -m -t ssh_port_t -p tcp 2222
sudo semanage port -l
```

SELinux denies access -> Check logs with sudo journalctl -t setroubleshoot or use audit2allow

If you see setenforce 0 in installation instructions, choose a different product. Disabling SELinux is not a proper solution.

Linux Firewall Tools

iptables

- Traditional Linux firewall
- Works at the packet-filtering level
- Different commands for IPv4/IPv6
  - Complex structure & rules
- `iptables -L -v -n [-t {nat,mangle}]`
- `ip6tables -L -v -n [-t {nat,mangle}]`
- `iptables-save; ip6tables-save`

nftables

- Successor to iptables
- Single framework for IPv4, IPv6, ARP, and bridge filtering
- Uses concise syntax; replaces multiple iptables commands
- `nft list ruleset`

ufw

- Stands for "Uncomplicated Firewall"
- Front-end for iptables/nftables
- Simple CLI for managing rules
- Mostly used on Ubuntu/Debian
- `ufw status verbose`

firewalld

- Dynamic firewall manager for Linux
- Zones define network trust levels
- Works with nftables back-end
- More complex & capable than `ufw`
- Common on RHEL/Fedora
- `sudo firewall-cmd --list-all`

Alternative init systems

Init System	Service File / Directory Location	Enable/Start Command
SysVinit	`/etc/init.d/<service>` and symlinks in `/etc/rc*.d/`	`sudo update-rc.d sshd defaults`
BusyBox init	`/etc/init.d/<service>`	`/etc/init.d/sshd start`
OpenRC	`/etc/init.d/<service>` `/etc/runlevels/<level>/`	`rc-service sshd start`
s6-rc	`/etc/s6-rc/source/<service>/` or `/service/<service>/`	`s6-rc -u change sshd`
runit	`/etc/sv/<service>/` (definitions) `/var/service/` (active)	`ln -s /etc/sv/sshd /var/service/`

- SysVinit is the traditional runlevel based init using shell scripts
- BusyBox init is a minimal SysV-style variant for embedded systems
- OpenRC is a modern dependency-aware script-based init with parallel startup
- runit is a fast three-stage init with built-in service supervision
- s6/s6-rc provides a highly modular, reliability-focused supervision framework ideal for minimal and containerized systems

```
Use ps [aux] | grep ' 1 ' to determine your init system (PID 1)
```

Exercise

Filesystems and LUKS

Use a loop file instead of a real disk:

```
truncate -s 256M disk.img
sudo cryptsetup luksFormat disk.img
sudo cryptsetup open disk.img lab
sudo mkfs.btrfs /dev/mapper/lab
sudo mount /dev/mapper/lab /mnt
```

1. Find the UUID with `sudo blkid disk.img`
2. Write a matching `crypttab` entry (do **not** install it)

Snapshots

```
sudo btrfs subvolume create /mnt/data
echo v1 | sudo tee /mnt/data/file
sudo btrfs subvolume snapshot -r /mnt/data /mnt/snap1
echo v2 | sudo tee /mnt/data/file
```

1. Compare `/mnt/data/file` and `/mnt/snap1/file`
2. Delete the snapshot
3. Clean up: `umount`, `cryptsetup close lab`, `rm disk.img`

Thank you!

Thank you for your attention!

Don't forget the feedback in Moodle please!