LINUX SECURITY

part_2.sh

Author: aimery de La Roncière @ aimerydlr
University: ECE Paris – Ing5 CYB
YEAR: 2025-2026

Table of contents

I. Cryptography and Communication Security

I.1. Configuration and Usage of SSH / SSHD

SSH (Secure Shell) is a foundational service used for secure remote administration of Linux systems. Its flexible design makes it both powerful and potentially dangerous if misconfigured. Misuse or negligence can expose systems to unauthorized access, lateral movement, or data leakage.

Documentation:

- https://doc.ubuntu-fr.org/ssh
- https://blog.stephane-robert.info/docs/securiser/durcissement/ssh/
- https://www.ssh-audit.com/hardening_guides.html

Configuration files

The SSH daemon (sshd) uses */etc/ssh/sshd_config* as its main configuration file. Starting with OpenSSH 7.3+, it supports modular configuration using the Include directive, allowing administrators to split configuration into logically **.conf* files in */etc/ssh/sshd_config.d/*. This modularity improves readability, enables automation, and facilitates selective overrides through the Match directive.

Mechanisms

- *Include* allows inclusion of additional configuration files. These are loaded in order, and earlier directives can be overridden by later ones.
- *Match* applies conditional settings based on attributes such as user, group, or network address. Once a Match block is encountered, only the directives within apply under the specified condition.
- Any directive placed after a Match block is considered part of that block, unless a new global section is started.

Administrators should take care to place security directives (like PermitRootLogin, PasswordAuthentication, etc.) before any Match blocks, unless the intent is to scope them.

```
# Load all configuration fragments
Include /etc/ssh/sshd_config.d/*.conf

# Apply settings only for users in 'dev' group
Match Group dev
    X11Forwarding no
    PermitRootLogin no

# Explicitly place global settings before Match blocks
PermitRootLogin no
Match Group dev
```

Key-based authentication

SSH key-based authentication eliminates the need for passwords by using an asymmetric key pair: a private key *stored securely by the client*, and a public key added to the server. When a user attempts to connect, the server verifies possession of the private key corresponding to the stored public key. This method is widely used in automation and DevSecOps environments and is considered significantly stronger than password-based logins.

Benefits

- Resistant to brute-force attacks and credential leaks
- Supports integration with hardware tokens (e.g. YubiKey)
- Enables automation and non-interactive authentication
- Often combined with AllowUsers or ForceCommand for restricted shell environments

To enforce this method, password authentication must be explicitly disabled to prevent users from authenticating with weak or reused passwords in configuration file:

```
# Enforce key-based authentication
PasswordAuthentication no
PubkeyAuthentication yes

# Reload the SSH daemon
systemctl reload sshd
```

The process of generating and utilizing SSH keys for secure server authentication involves creating a key pair, deploying the public key to a server, and connecting with the private key:

```
# Generate a new SSH key pair using ed25519 (recommended)
ssh-keygen -t ed25519 -C "admin@host"

# Upload public key to a remote server
ssh-copy-id admin@10.0.0.15

# Connect using a specific identity file
ssh -i ~/.ssh/id_ed25519 admin@10.0.0.15
```

Permissions and file integrity

SSHD configuration file permissions

The configuration files of the SSH daemon define critical security behaviors. Any unauthorized modification could weaken the system. For example, by allowing root login or re-enabling password authentication.

The objective is to restrict modifications to root only, ensuring that unauthorized users cannot alter SSH security settings.

- Recommendation

Apply the following permissions and owner:

- Owner: root
- Group: root

- 600 for both /etc/ssh/sshd_config and all files in /etc/ssh/sshd_config.d/

- Remediation

```
chown root:root /etc/ssh/sshd_config
chmod 600 /etc/ssh/sshd_config

chown root:root /etc/ssh/sshd_config.d/*.conf
chmod 600 /etc/ssh/sshd_config.d/*.conf
```

Host SSH private key permissions

Host private keys uniquely identify a server during SSH authentication. If compromised, an attacker could impersonate your server. Files involved:

- /etc/ssh/*_key (e.g. ssh_host_rsa_key, ssh_host_ed25519_key)

The objective is to ensure only root can read or write private host keys.

- Recommendation
  - Owner: root
  - Group: root
  - Permissions: 600

- Remediation

```
find /etc/ssh -xdev -type f -name '*_key' ! -name '*.pub' -exec chown root:root {} \;
find /etc/ssh -xdev -type f -name '* key' ! -name '*.pub' -exec chmod 600 {} \;
```

Host SSH public key permissions

Although less sensitive than private keys, host public keys must be protected against tampering to avoid trust poisoning. Files involved:

- /etc/ssh/*_key.pub (e.g. ssh_host_rsa_key.pub)

- Recommendation
  - Owner: root
  - Group: root
  - Permissions: 644

- Remediation

```
find /etc/ssh -xdev -type f -name '*.pub' -exec chown root:root {} \;
find /etc/ssh -xdev -type f -name '*.pub' -exec chmod 644 {} \;
```

Access control (AllowUsers / AllowGroups)

SSH access should be limited to specific users or groups to reduce the attack surface. By explicitly defining who can connect, you prevent unauthorized users (including local or system accounts) from attempting remote login.

The objective is to explicitly define who is allowed to connect to the server.

- Recommendation

Use the following directives to control access:

  - AllowUsers
  - AllowGroups
  - DenyUsers
  - DenyGroups

- Selection logic

The behavior of these directives follows strict precedence rules:

  - AllowUsers / AllowGroups
    - Only specified users/groups are allowed. All others are implicitly denied.
    - This acts as a whitelist.

- DenyUsers / DenyGroups
  - Specified users/groups are denied. All others are allowed.
  - This acts as a blacklist.

- Combination rules (use with caution)
  - If both AllowUsers and DenyUsers are defined:
    - A user must be present in AllowUsers
    - And not in DenyUsers
  - The same rule applies to AllowGroups and DenyGroups.

- Evaluation order (according to OpenSSH):

1. AllowUsers
2. AllowGroups
3. DenyUsers
4. DenyGroups

Examples

```
AllowUsers admin1 admin2  # Allow only two users to connect
AllowGroups sshusers      # Restrict SSH to a group named "sshusers"
DenyUsers backup1 backup2 # Explicitly deny backup accounts from SSH

# Apply the changes
systemctl reload sshd
```


Authentication policy and login restrictions

Disable root login (*PermitRootLogin*)

Disabling root login enforces the use of user accounts with privilege escalation via sudo, increasing accountability and auditability.

```
sed -i 's/^.*PermitRootLogin.*/PermitRootLogin no/' /etc/ssh/sshd_config

# Apply the changes
systemctl reload sshd
```

Disable empty passwords (*PermitEmptyPasswords*)

Accounts with empty passwords must never be allowed to authenticate via SSH.

```
sed -i 's/^.*PermitEmptyPasswords.*/PermitEmptyPasswords no/' /etc/ssh/sshd_config

# Apply the changes
systemctl reload sshd
```

Disable user-controlled environments (*PermitUserEnvironment*)

Users should not be allowed to set environment variables via .ssh/environment, as this can be abused to alter shell behavior or bypass restrictions.

```
sed -i 's/^.*PermitUserEnvironment.*/PermitUserEnvironment no/' /etc/ssh/sshd_config

# Apply the changes
systemctl reload sshd
```


Disable host-based authentication (*HostbasedAuthentication*)

Host-based authentication is a legacy method that allows clients to authenticate based on the hostname and IP address of the originating machine, combined with the presence of specific trust files such as .rhosts, .shosts, or /etc/hosts.equiv. This method does not rely on user credentials but instead on the assumption that the remote host is trustworthy. This is an inherently insecure assumption in modern environments.

```
sed -i 's/^.*HostbasedAuthentication.*/HostbasedAuthentication no/' /etc/ssh/sshd_config

# Apply the changes
systemctl reload sshd
```

Ignore .rhosts files (IgnoreRhosts)

Ensure .rhosts files are ignored, preventing unintended trust relationships.

```
sed -i 's/^.*IgnoreRhosts.*/IgnoreRhosts yes/' /etc/ssh/sshd_config

# Apply the changes
systemctl reload sshd
```


Cryptographic configuration

SSH relies on a combination of cryptographic algorithms to provide confidentiality, integrity, and authentication. Misconfigured or outdated algorithms can expose your server to downgrade attacks, weak encryption, or hash collisions. This section focuses on hardening the cryptographic profile of the SSH daemon by explicitly defining strong ciphers, key exchange algorithms, and MACs (Message Authentication Codes).

Cipher selection (*Ciphers*)

Ciphers are used to encrypt the traffic between the SSH client and server. Modern ciphers like AES in CTR or GCM mode are recommended, while legacy options like CBC mode or RC4 are insecure and must be avoided.

```
# Secure cipher suite definition
chacha20-poly1305@openssh.com,aes256-gcm@openssh.com,aes256-ctr,aes192-ctr,aes128-gcm@openssh.com,aes128-ctr

# Apply the changes
systemctl reload sshd
```

Key exchange algorithms (*KexAlgorithms*)

Key exchange algorithms are used during the initial handshake to securely negotiate a session key. Weak algorithms like diffie-hellman-group1-sha1 are susceptible to downgrade and precomputation attacks.

```
# Secure KexAlgorithms suite definition
KexAlgorithms sntrup761x25519-sha512@openssh.com,gss-curve25519-sha256-,curve25519-sha256,curve25519-sha256@libssh.org,diffie-
hellman-group18-sha512,diffie-hellman-group-exchange-sha256,gss-group16-sha512-,diffie-hellman-group16-sha512

# Apply the changes
systemctl reload sshd
```


MAC algorithms (*MACs*)

MACs ensure message integrity and authenticity. Legacy algorithms like MD5 and short SHA1-based variants are deprecated and must be disabled.

```
# Secure MACs suite definition
MACs hmac-sha2-512-etm@openssh.com,hmac-sha2-256-etm@openssh.com,umac-128-etm@openssh.com

# Apply the changes
systemctl reload sshd
```

Auditing SSH security with ssh-audit

ssh-audit is a security auditing tool for SSH servers. It performs a non-intrusive scan to identify weak or deprecated algorithms, key sizes, protocol versions, and unsafe configurations.

Documentation: https://github.com/jtesta/ssh-audit

It is particularly useful in environments where SSH must be hardened according to specific baselines. It analyzes

- Host key types and bit lengths
- Supported protocol versions (e.g. SSHv1 should be rejected)
- Cipher suites, KEX algorithms, and MACs
- Banner version (leaking OpenSSH version may be risky)
- Known vulnerabilities, deprecated features, and insecure defaults

Installation

```
pipx install ssh-audit
```


Commands examples

```
# Audit a remote host on the default port (22)
ssh-audit 192.168.1.5

# Use a non-standard port
ssh-audit -p 2222 myserver.example.com

# Verbose output with detailed algorithm breakdown
ssh-audit -v 127.0.0.1
```

Figure 1 - ssh-audit report example

Session restrictions and timeouts

To reduce the risk of unauthorized session persistence or resource exhaustion, it is essential to enforce timeouts and session limits. Idle SSH connections may be hijacked, and unrestricted multiplexing could be abused by attackers or misconfigured tools. These directives help administrators enforce time-bound access and limit the number of concurrent sessions per user or system-wide.

Idle session timeout (*ClientAliveInterval* and *ClientAliveCountMax*)

SSH servers can monitor client activity and disconnect sessions that are inactive for a specified duration.

- *ClientAliveInterval* defines the interval (in seconds) between "keepalive" messages sent by the server to the client.
- *ClientAliveCountMax* specifies the number of unanswered messages before the server terminates the connection.

Together, these parameters allow administrators to automatically close inactive connections and reduce the exposure window.

```
# Add or update the following lines
ClientAliveInterval 60
ClientAliveCountMax 5

# Apply the changes
systemctl reload sshd
```

Login grace time (*LoginGraceTime*)

LoginGraceTime determines how long an unauthenticated user has to successfully log in before the server closes the connection. A short value limits the exposure to brute-force or idle connection attempts.

```
# Reduce login grace period
LoginGraceTime 60

# Apply the changes
systemctl reload sshd
```


Authentication attempts limit (*MaxAuthTries*)

This directive restricts the number of failed authentication attempts allowed per session. Reducing this value decreases the likelihood of brute-force attacks.

```
# Lower the threshold of failed login attempts
MaxAuthTries 4

# Apply the changes
systemctl reload sshd
```

Multiplexed sessions limit (*MaxSessions*)

Controls the number of concurrent SSH sessions allowed over a single connection. This helps prevent abuse of session multiplexing mechanisms such as ControlMaster.

```
# Limit number of concurrent sessions
MaxSessions 10

# Apply the changes
systemctl reload sshd
```

Startups rate limiting (*MaxStartups*)

Defines the maximum number of unauthenticated connections that can be active simultaneously. Helps protect the SSH daemon from denial-of-service attacks.

The syntax is start:rate:full, where:

- start is the number of allowed unauthenticated connections,
- rate is the percentage of connections to be randomly dropped after reaching start,
- full is the absolute maximum number of unauthenticated connections.

```
# Configure connection rate limiting
MaxStartups 10:30:60

# Apply the changes
systemctl reload sshd
```

Log level configuration (*LogLevel*)

The *LogLevel* directive sets the verbosity of sshd logging. Choosing the right level provides sufficient detail for security auditing while avoiding excessive noise in the logs.

Common values include:

- QUIET: Minimal logging.
- FATAL: Only fatal errors.
- ERROR: Standard error messages.
- INFO: Informational messages (default).
- VERBOSE: Includes key-based login details.
- DEBUG: Highly detailed logs (not recommended in production).
- The VERBOSE level is typically recommended for production environments requiring traceability of authentication events.

```
# Increase verbosity to capture public key fingerprints and login attempts
LogLevel VERBOSE

# Apply the changes
systemctl reload sshd
```

Forwarding restrictions

SSH forwarding capabilities are powerful but can introduce significant security risks if enabled unnecessarily. These mechanisms are often used for legitimate purposes, such as tunneling services through encrypted channels or accessing remote desktops. However, in hardened environments, they may allow users to bypass network segmentation or exfiltrate data covertly.

Disable SSH forwarding globally (*DisableForwarding*)

The *DisableForwarding* directive is a global switch that disables all forms of SSH forwarding, including:

- X11 forwarding
- Agent forwarding
- TCP forwarding (Local, Remote)
- StreamLocal forwarding

This directive overrides more specific options such as *AllowTcpForwarding* or *PermitTunnel*.

If your users or automation systems do not require forwarding, this directive should be set to yes to reduce attack surface.

```
# Disable all forwarding capabilities globally
DisableForwarding yes

# Apply the changes
systemctl reload sshd
```

Restrict forwarding at the user level (authorized_keys)

Even when *DisableForwarding* is not set globally, it is possible to restrict SSH capabilities per user by prefixing options in the ~/.ssh/authorized_keys file.

```
restrict,disable-forwarding ssh-ed25519 AAAAC3NzaC1...
```

This approach ensures individual key-based restrictions and prevents privileged users from using SSH tunnels inappropriately.

I.2. OpenPGP – GPG (GNU Privacy Guard)

OpenPGP is an open standard used to ensure confidentiality, integrity, and authenticity of data and communications. Its main purpose is to allow users to securely exchange information, verify the identity of the sender, and protect messages and files from unauthorized access or tampering.

Documentation:

- https://www.openpgp.org/
- https://openpgp.dev/
- https://gnupg.org/

Brief Historical Context

- *PGP (Pretty Good Privacy)* was created in 1991 by Phil Zimmermann as a tool for secure email encryption and digital signatures.
- *OpenPGP* is the open s https://openpgp.dev/tandard (RFC 4880) derived from PGP, ensuring interoperability across different implementations.
- *GPG (Gnu Privacy Guard)* is the most widely used free implementation of OpenPGP. It is open source and available by default in most Linux distributions.

OpenPGP main applications include:

- Email security: Encrypting and signing emails to ensure confidentiality, integrity, and authenticity.
- Software distribution: Signing Linux packages and updates (e.g., Debian, Ubuntu, Fedora) to verify origin and integrity.
- Code signing: Developers sign Git commits and tags with GPG to prove authorship and prevent tampering.
- Secure file exchange: Encrypting files for transfer or storage, protecting sensitive data over insecure channels.
- Encrypted backups: Tools like duplicity use GPG to secure backups stored locally or in the cloud.
- Authentication and password management: Utilities like pass rely on GPG for secure identity and secret storage.
- Web of Trust: A decentralized trust model where users sign each other's keys to build identity assurance.

Keys and identities in OpenPGP

OpenPGP uses a hybrid key model to secure communications, combining strong security with practical performance.

Master key

Master key is the central key linked to your identity (name, email, etc.). It is mainly used to sign subkeys and other keys, establishing trust. It is rarely used for daily encryption and is often kept offline for safety.

Subkeys

Subkeys are secondary keys tied to the master key, used for encryption, signing, and authentication. They can be replaced or revoked without affecting the master key.

When subkeys are created, OpenPGP implementations rely on a strong key derivation function to generate independent cryptographic material. GnuPG, for example, uses HKDF (HMAC-based Key Derivation Function) to derive subkey material from high-entropy randomness provided by the operating system.

HKDF takes a random seed and applies HMAC (often with SHA-256) to expand it into multiple cryptographically strong keys.

Each subkey (for encryption, signing, or authentication) is therefore mathematically independent from the master key, but still bound to it through a certification signature.

This guarantees forward secrecy (compromise of one subkey does not reveal others) and enables safe rotation or revocation.

Identity

The information (name, email, comment) that associates a key with its owner. A single master key can manage multiple identities, allowing separate personal and professional uses while keeping trust and security centralized.

Signing with GPG (Sender Authentication and Message Integrity)

OpenPGP uses digital signatures to ensure the authenticity and integrity of messages and files.

- Signing Process (authentication)
  - The sender generates a hash (digest) of the message or file.
  - The hash is then encrypted with the sender's private key to create the digital signature.
  - This signature is attached to the message or file.
  - The sender's public key can be distributed via a certificate, which attests to their identity, or directly without a certificate. In the latter case, there is no guarantee that the sender actually owns the public key.
- Verification Process (integrity & authenticity)
  - The recipient decrypts the signature using the sender's public key, possibly obtained through a trusted certificate.
  - The recipient also computes the hash of the received message or file.
  - If the decrypted hash matches the computed hash, the message is authentic and intact, and the certificate confirms that the public key truly belongs to the sender.

- This approach provides:
  - Authenticity: confirms the sender's identity, validated via the certificate.
  - Integrity: ensures the content has not been modified in transit.
  - Non-repudiation: the sender cannot deny having signed the message.

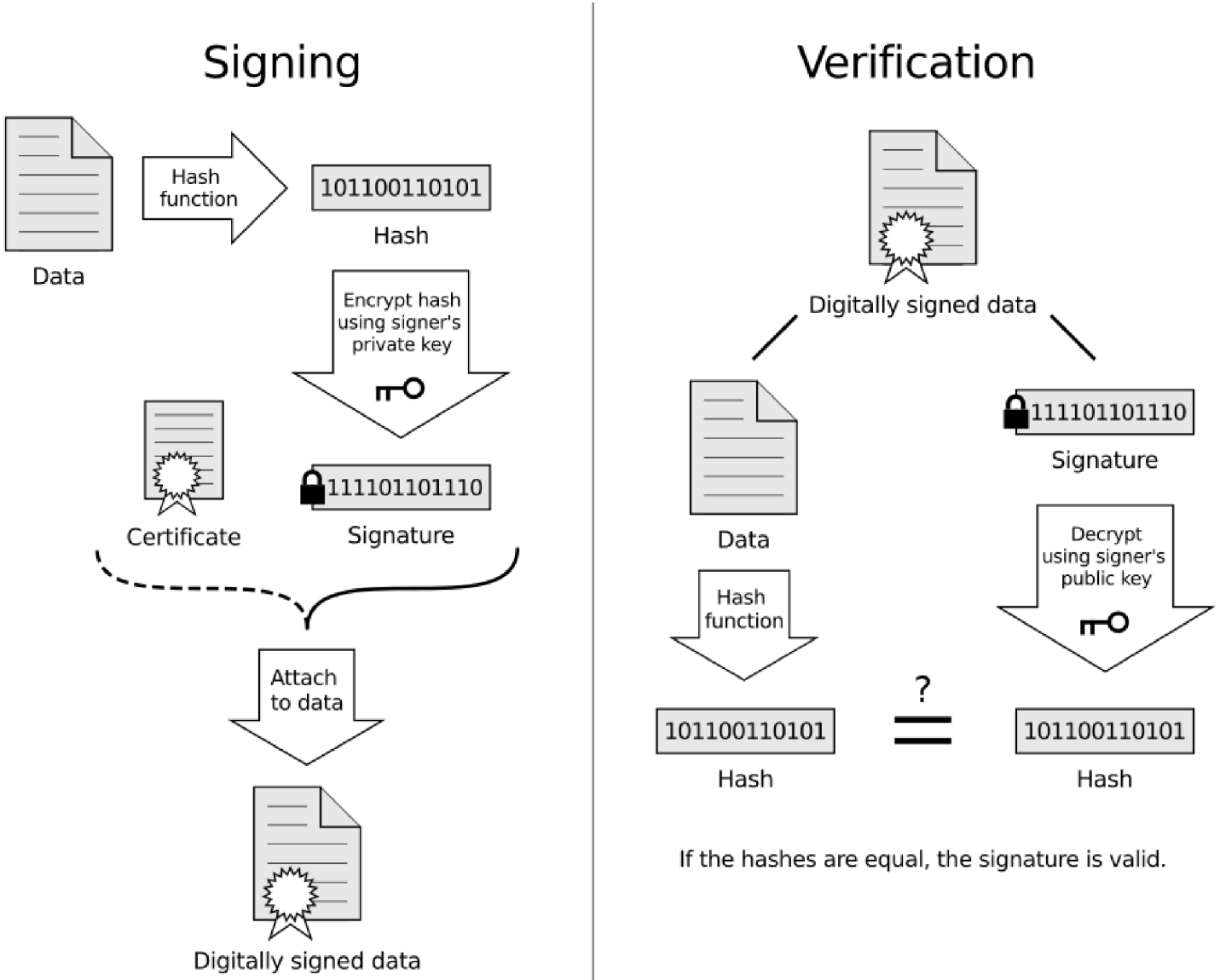

Figure 2 - Digital Signature diagram

Source: https://commons.wikimedia.org/wiki/File:Digital_Signature_diagram.svg

Encryption with GPG (Confidentiality)

OpenPGP uses a hybrid encryption system combining asymmetric and symmetric cryptography to provide secure communication.

- Symmetric Encryption (fast)
  - The actual data (message or file) is encrypted using a randomly generated session key with a symmetric algorithm like AES.
  - Symmetric encryption is used because it is much faster than asymmetric encryption for large data.
- Asymmetric Encryption (secure key exchange)
  - The session key itself is encrypted using the recipient's public key.
  - Only the recipient's private key can decrypt this session key, ensuring that only the intended recipient can read the message.

This combination provides both high security (via asymmetric encryption) and efficient performance (via symmetric encryption).

Figure 3 - Encryption with GPG

Source: https://proton.me/blog/fr/what-is-pgp-encryption

Key Generation and Web of Trust (Authenticity of Keys)

Before users can securely sign, encrypt, or verify messages in OpenPGP, they need a key pair and a way to establish trust in other users' keys. The *Web of Trust* (WOT) allows users to confirm that public keys actually belong to the claimed owners, *without relying on a centralized authority*. This section explains how keys are generated, signed, and trusted in a decentralized network.

PGP Key Generation

- Each user generates a key pair (public and private)

Key Signing

- When a user meets someone and verifies their identity, they can sign that person's public key with their private key.
- Verification typically involves checking a government ID or other trusted proof of identity.
- Keys can be exchanged:
  - Directly via USB, email, QR code, or in person.
  - Via key servers, though this requires caution as anyone can upload a key.
- The signature states: "I confirm that this key really belongs to this person."
- Multiple signatures increase confidence that a key belongs to its claimed owner.

Propagation of Trust (Web of Trust)

- Trust is transitive: if A trusts B, and B trusts C, then A can partially trust C.
- The Web of Trust is user-driven, decentralized, and does not rely on a central authority.
- It allows users to verify the authenticity of keys indirectly through trusted intermediaries.
- Graphically, it can be represented as nodes (users) connected by signed keys.

Levels of Trust

- PGP lets users assign trust levels to keys and to a user's ability to sign other keys:
  - Full trust: the key owner is fully trusted to sign other keys.
  - Marginal trust: the key owner is partially trusted; additional signatures are needed for full confidence.
  - Untrusted: no trust in the key owner.

- Trust affects automatic key verification: a recipient's software may require multiple trusted signatures to accept a key as valid.

Practical Benefits

- Prevents man-in-the-middle attacks: only keys verified through the Web of Trust are trusted.
- Strengthens non-repudiation: signed keys and messages ensure accountability.
- Supports a decentralized trust model: users are responsible for verifying identities and signatures.
- Enhances overall security of encrypted communication, especially in communities without centralized certificate authorities.

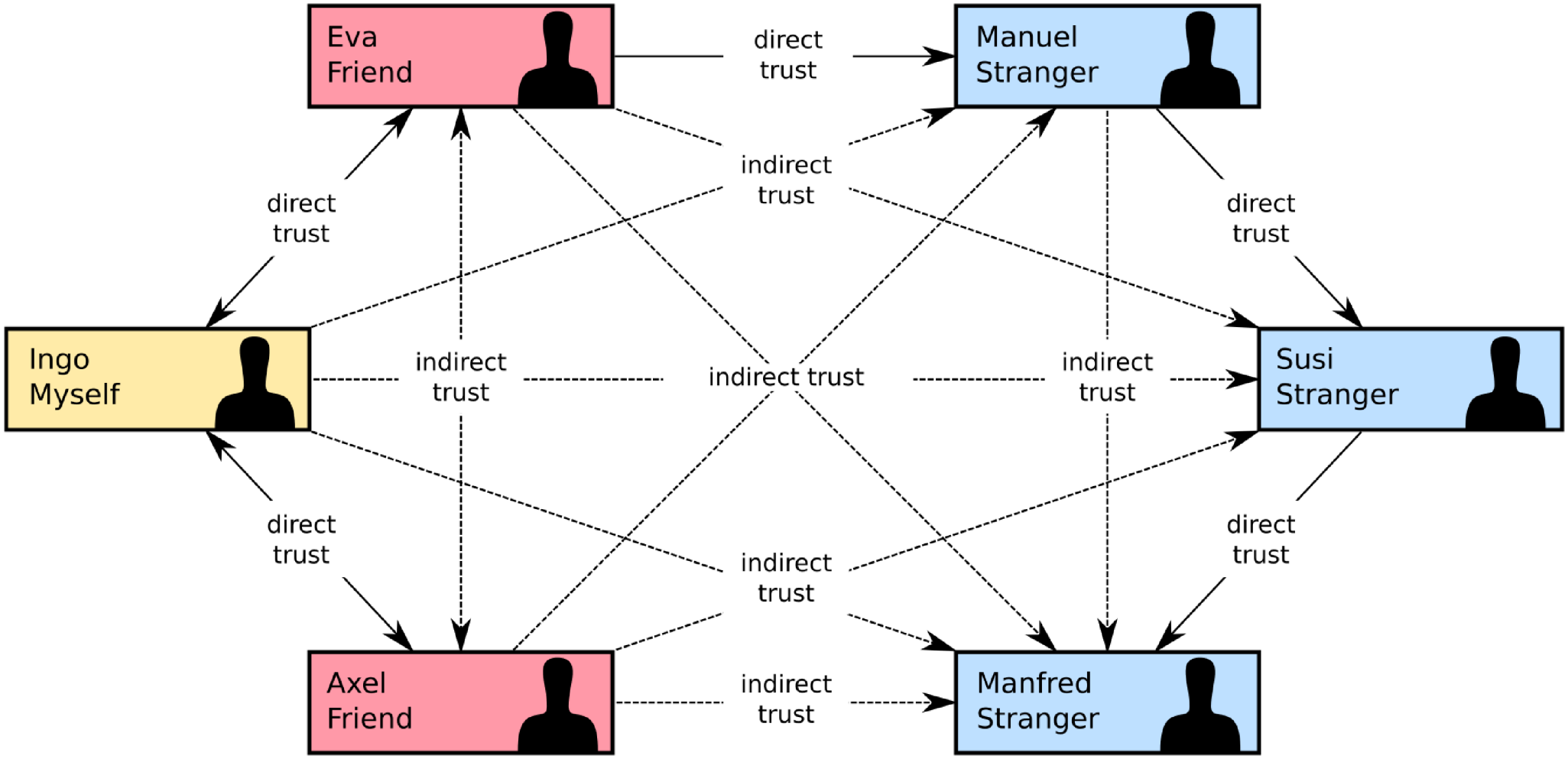

Figure 4 - Web Of Trust

Source: https://en.wikipedia.org/wiki/Web_of_trust#/media/File:Web_of_Trust-en.svg

Practical Usage of GPG

GPG (Gnu Privacy Guard) is the most widely used OpenPGP implementation. This section shows how to use GPG in practice for key management, encryption, and signing.

Installing GPG

```
sudo apt update
sudo apt install gnupg
```

Key Generation and Storage

Generate a key pair (public/private) for both sender and receiver (Bob and Alice):

```
gpg --full-generate-key
```

Steps:

- Select key type (RSA and RSA or ECC).
- Choose key size (3072 bits or higher).
- Set an expiration date or not.
- Enter identity information (name, email).
- Set a strong passphrase to protect private key.

Check Keys

```
gpg --list-keys       # public keys
gpg --list-secret-keys # private keys
```


Export public key to share (sender : Bob)

This command exports your GPG public key in a human-readable text format and saves it to the file mypubkey.asc :

```
gpg --armor --export Bob@example.com > Bob_pub.asc
```

Import public key (receiver: Alice)

```
# Import the sender's public key
gpg --import Bob_pub.asc
```

Sign trusted imported public key (receiver: Alice)

```
gpg --edit-key Bob@example.com
> sign
```

Signing (Authentication & Integrity)

- Sign a file or message

```
gpg --sign myfile.txt
```

It should produce a signed file (myfile.txt.gpg).

- Detached signature:

```
gpg --detach-sign myfile.txt
```

It should produce myfile.txt.sig, separate from the file.

- Verify a signature

```
gpg --verify myfile.txt.sig myfile.txt
```

It should confirm the file was signed by the owner of the corresponding public key. It displays trust depth :

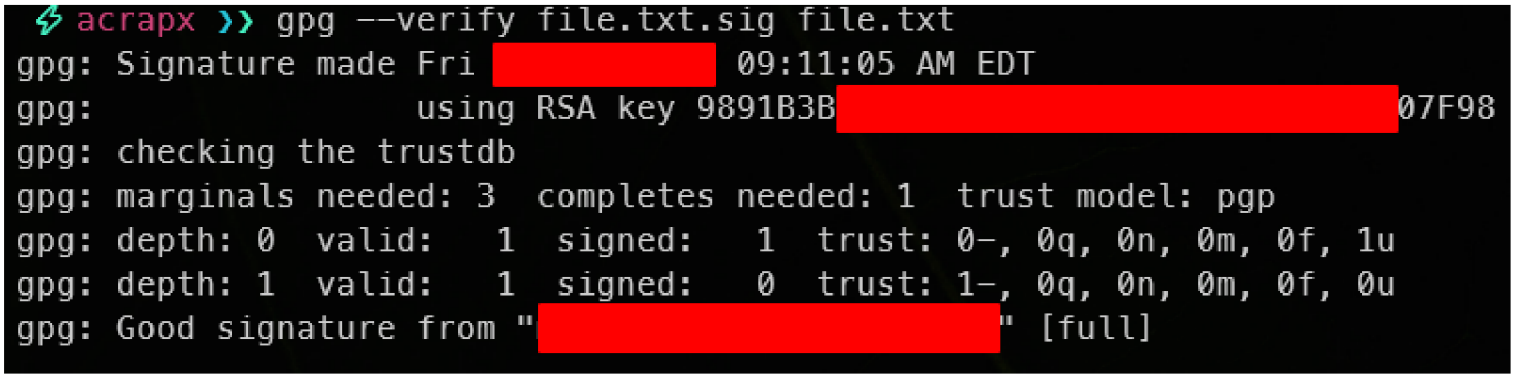

Figure 5 – GPG signature file verification

Encryption (Confidentiality)

- Alice encrypts a file for a recipient (Bob) with his public key:

```
gpg --encrypt --recipient Bob@example.com myfile.txt
```

This should produce myfile.txt.gpg that only the recipient's (Bob) private key can decrypt.

- Decrypt the file with recipient private key (Bob):

```
gpg --decrypt myfile.txt.gpg > myfile.txt
```


Signing + Encryption (Authentication + Confidentiality)

- Sign and encrypt a file in one step:

Alice encrypts and signs a file for Bob:

```
gpg --sign --encrypt --recipient Bob@example.com myfile.txt
```

File should be encrypted for the recipient (Bob) with its public key and signed with sender private key (Alice).

This ensures both confidentiality (only the recipient can read) and authenticity (recipient can verify it's from you).

- Decrypt and verify simultaneously:

Bob decrypt and verify signature:

```
gpg --decrypt myfile.txt.gpg > myfile.txt
```

GPG automatically verifies the signature with Alice's public key while decrypting the file with Bob's private key if the sender's public key is trusted.

I.3. LUKS (Linux Unified Key Setup)

LUKS (Linux Unified Key Setup) is the standard for disk encryption on Linux systems. It provides a unified and secure on-disk format for encrypted volumes and supports multiple passphrases and key management mechanisms. LUKS operates in conjunction with dm-crypt, a kernel-level device-mapper cryptographic target that handles block-level encryption.

dm-crypt handles the encryption and decryption of data blocks as they are read from or written to the disk, while LUKS adds a metadata layer that manages key slots, key derivation, and format consistency across systems. While dm-crypt can be used directly, LUKS simplifies key management and ensures cross-system compatibility.

Documentation: https://gitlab.com/cryptsetup/cryptsetup/blob/master/README.md

Typical Use Cases

- Full disk encryption on laptops or desktops to protect data at rest in case of theft or loss.
- Encrypted partitions for specific data (e.g., /home, backup volumes, USB drives).
- Server-side encryption for sensitive databases, logs, or virtual machine images.
- Cloud storage volumes (e.g., with LVM or iSCSI targets) requiring confidentiality.

LUKS Versions: LUKS1 vs LUKS2

LUKS2 is now the default in cryptsetup (v2.1+), offering stronger key derivation and better resistance against brute-force attacks.

Feature	LUKS1	LUKS2
Metadata format	Fixed header layout	JSON-based, extensible
Symmetric-key algorithm	AES, Serpent, Twofish, aes-twofish, twofish-serpent	AES, Serpent, Twofish, aes-twofish, twofish-serpent, serpent-twofish
Key derivation	PBKDF2 only	Argon2id (default), PBKDF2
Integrity	None	Optional authentication (HMAC)
Sector size	Fixed	Flexible (with header options)
Backward compat	High	Lower (due to format change)

Key Management in LUKS

LUKS supports up to 8 key slots, each of which can store a passphrase or a key file. These slots are linked to the same master key, which encrypts the volume. Changing a key in a slot does not require re-encrypting the entire device.

- Types of keys:
  - Passphrases: Interactive, user-friendly, but potentially weaker if poorly chosen.
  - Keyfiles: Files containing entropy or a binary secret, often stored on external media.
  - Keyscript (in initramfs): Automatically supplies the decryption key at boot (e.g., network or TPM-based retrieval).

Creating and Using an Encrypted Partition with LUKS

Dependances installation

```
sudo apt update
sudo apt install cryptsetup parted cryptsetup
```

Preparing the Partition

```
# Create a new GPT partition table on the device /dev/sdb
sudo parted /dev/sdb mklabel gpt

# Create a primary partition on /dev/sdb that spans the entire disk from 0% to 100%
sudo parted /dev/sdb mkpart primary 0% 100%
```

Formatting with LUKS

```
#This creates the LUKS header and prompts for a passphrase. All data will be removed
sudo cryptsetup luksFormat --type luks2 /dev/sdb1
```


Opening the Encrypted Volume

```
# Open the encrypted LUKS partition located at /dev/sdb1 and map it to a device named 'data_encrypted'
sudo cryptsetup luksOpen /dev/sdb1 data_encrypted
```

This creates a decrypted *device mapper* at */dev/mapper/*data_encrypted that represents a raw encrypted volume. Without a filesystem, you cannot store files on it.

Creating a Filesystem

```
# Create an ext4 filesystem on the decrypted device /dev/mapper/data_encrypted
sudo mkfs.ext4 /dev/mapper/data_share_encrypted
```

Mounting the Volume

```
# Create a directory to serve as the mount point for the encrypted volume
sudo mkdir -p /mnt/data_encrypted

# Mount the decrypted device to the mount point so that it can be accessed and used to store files
sudo mount /dev/mapper/data_share_encrypted /mnt/data_encrypted
```

Closing the Volume

```
# Unmount the filesystem currently mounted at /mnt
umount /mnt

# Close the LUKS encrypted device mapped to 'securedata'
sudo cryptsetup close securedata
```


I.4. VeraCrypt

VeraCrypt is an open-source disk encryption tool that provides on-the-fly encryption for file containers, partitions, or entire disks. Data is automatically encrypted before being written and decrypted when read, without user intervention as long as the correct key is provided. VeraCrypt is the successor to TrueCrypt, offering stronger key derivation functions and improved resistance to brute-force attacks.

Documentation: https://veracrypt.io/

VeraCrypt can be used to:

- Encrypt sensitive files in a virtual container
- Securely encrypt USB drives, external disks, or Windows system partition
- Create hidden volumes for plausible deniability
- MFA : Protect data with a combination of password, keyfile, and hardware tokens
- Encryption is transparent to the user but completely inaccessible without authentication.
- Multi-OS transport

VeraCrypt can be used with CLI or GUI interface.

VeraCrypt installation

```
# Download the veracrypt-<version>-setup.tar.bz2 file from the official VeraCrypt website
# URL: https://veracrypt.fr/en/Downloads.html
# Extract the downloaded VeraCrypt setup archive
tar -xvf veracrypt-<version>-setup.tar.bz2
# Install the dbus-x11 package, which is required for running GUI applications with root privileges
sudo apt install dbus-x11
# Execute the VeraCrypt setup script with root privileges to start the GUI installer
sudo ./veracrypt-<version>-setup-gui-x64
```


II. Securing Services and Applications

II.1. GRUB Bootloader Protection

In the event that the root password is lost or forgotten, there is a *reliable way to reset it without knowing the previous one*. This method exploits a fundamental aspect of Linux boot architecture: the GRUB bootloader.

GRUB allows interaction with the kernel command line before the system has fully booted. By modifying the boot parameters, one can intercept the normal initialization process and gain direct root access to a shell without authentication. While this mechanism is intended for recovery, it also presents a security risk if attackers can access the machine physically.

This is not a vulnerability, but a design feature intended for system recovery. However, physical access to the machine is required and should always be controlled.

A bit of red team: Understanding GRUB and Kernel Manipulation for Root Access

When you power on a Debian-based system, the GRUB bootloader is responsible for loading the Linux kernel into memory. Once the kernel is loaded, it continues the system initialization process by launching core services using systemd, init, or an equivalent service manager.

However, GRUB does more than just load the kernel, it allows users to edit the kernel command line before booting.

Hijacking the Normal Boot Process

- Standard Boot

The default boot entry in GRUB passes a specific set of parameters to the kernel. Here is an example:

```
cat /boot/grub/grub.cfg

/vmlinuz-6.11.0-26-generic root=/dev/mapper/ubuntu--vg-ubuntu--lv ro apparmor=1 security=apparmor quiet splash $vt_handoff
```


Component	Description
/vmlinuz-6.11.0-26-generic	The compressed Linux kernel image to load.
root=/dev/mapper/ubuntu--vg-ubuntu--lv	Defines the root filesystem, located here on a LVM volume (ubuntu-vg/ubuntu-lv).
ro	Mounts the root filesystem as read-only initially to protect system integrity. It may be remounted as read/write later by systemd or init.
apparmor=1	Enables AppArmor, a Linux kernel security module that enforces access control policies.
security=apparmor	Sets AppArmor as the primary security mechanism for the kernel.
quiet	Reduces kernel message output (verbosity) during boot, hiding most messages unless there are errors.
splash	Displays a graphical splash screen instead of textual boot logs. Common in user-friendly distributions like Ubuntu.
$vt_handoff	An Ubuntu-specific variable for smooth graphical transition between kernel boot and display manager (via Plymouth).

- Modified Boot for Root Access

By editing this GRUB entry, we can instruct the kernel to boot into a root shell instead of launching the full operating system:

```
/vmlinuz-6.11.0-26-generic root=/dev/mapper/ubuntu--vg-ubuntu--lv rw apparmor=1 security=apparmor quiet splash $vt_handoff
init=/bin/bash
```

Changes made:

- ro ➤ rw: Mount the root filesystem in read-write mode.
- init=/bin/bash: Skip system initialization and launch a barebones shell (bash) directly.

This tells the kernel: "Start the system in the most minimal way possible: give me a root shell and nothing else: no login, no checks."

Because bash is now executed as the first process (PID 1) it is run with full root privileges.

This results in: A root terminal prompt without any authentication.

How to Protect Against This

GRUB Password Protection

The objective is to prevent users from:

- Editing GRUB entries using the *e* key at boot.
- Accessing the GRUB command line with the *c* key.

This is a key defense against local privilege escalation when someone has physical access to the machine.

- Audit: Check for GRUB Password Configuration

Before applying a password, you can check whether one is already set:

```
grep "^set superusers" /boot/grub/grub.cfg
awk -F. '/^\s*password/ {print $1"."$2"."$3}' /boot/grub/grub.cfg
```

These commands look for *superusers* and *password_pbkdf2* definitions in the GRUB configuration.

- Configure a GRUB Password
  - Generate a Secure Password Hash

Use the following command to generate a PBKDF2-hashed password for GRUB:

```
grub-mkpasswd-pbkdf2 --iteration-count=600000 --salt=64
```

  - Store the Password in a Custom GRUB Script

Do not modify /etc/grub.d/00_header directly, as it may be overwritten during GRUB package updates.

Instead, create a custom script (e.g. 40_addpass) like this:

```
sudo bash -c 'cat <<EOF > /etc/grub.d/40_addpass
#!/bin/sh
exec tail -n +3 \$0
set superusers="<username>"
password_pbkdf2 <username> <your-pbkdf2-password>
EOF'
```

Make the script executable:

```
chmod +x /etc/grub.d/40_addpass
```


- Optional: Allow Password-Free Normal Boot

By default, if you define a GRUB password, it will be required at every boot, even for standard entries. To avoid prompting for a password on normal boots, and only protect editing or the GRUB console, you can mark the menu entries as "unrestricted".

Edit the 10_linux script:

```
sudo sed -i 's/^\(CLASS=".*\)\("\)$/\1 --unrestricted\2/' /etc/grub.d/10_linux
```

This ensures that:

- Normal users can boot without a password.
- Only editing entries or accessing GRUB's shell (e or c) requires authentication.

- Apply the Configuration

After making the changes, regenerate the GRUB configuration:

```
sudo update-grub
```

Full Disk Encryption (LUKS)

See LUKS (Linux Unified Key Setup).

UEFI/BIOS Password

Prevents boot order changes or booting from USB.

II.2. UFW (Uncomplicated Firewall)

Linux systems support multiple frameworks for configuring firewall rules:

- iptables: The traditional firewall interface using netfilter in the Linux kernel.
- nftables: The modern replacement for iptables, introduced in Linux kernel 3.13. It provides a unified syntax and improved performance.
- UFW (Uncomplicated Firewall): A user-friendly frontend to manage iptables rules more easily.

UFW is not a standalone firewall engine. It acts as an *abstraction layer over iptables*. Therefore, enabling UFW configures iptables under the hood.

Only *one firewall management tool should be active at a time* on a Linux system to avoid conflicting rules (e.g., do not use both nftables and ufw simultaneously).

Documentation:

- https://help.ubuntu.com/community/UFW
- https://wiki.debian.org/Uncomplicated%20Firewall%20%28ufw%29
- https://blog.stephane-robert.info/docs/securiser/reseaux/ufw/

UFW system files and directories

UFW's configuration is stored in the /etc/ufw/ directory. Key files include:

- */etc/ufw/before.rules*: Rules applied before user-defined rules (e.g., loopback, DHCP)
- */etc/ufw/after.rules*: Rules applied after user-defined rules (e.g., logging, exceptions)
- */etc/ufw/user.rules*: Persistent rules created via the CLI (ufw allow, etc.)
- */etc/ufw/sysctl.conf*: Kernel network options (e.g., ICMP, forwarding)

Installation, activation, and initial setup

Installing UFW

```
sudo apt update
sudo apt install ufw iptables
```

Do not install iptables-persistent if you're using UFW. The iptables-persistent is a boot-time loader for netfilter rules, iptables plugin. Running both ufw and the services included in the iptables-persistent package may lead to conflict

Enabling UFW (with caution for SSH)

Enabling UFW will flush existing iptables rules, which may interrupt remote SSH sessions. Always allow important rules before activation.

- Example of rules that can be set before enabling UFW:

```
ufw allow out http
ufw allow out https
ufw allow out ntp
ufw allow out to any port 53 # DNS
ufw allow out to any port 853 # DNS over TLS
ufw allow in ssh
ufw logging on
```

- Enable UFW:

```
sudo ufw enable
```

Checking UFW status

```
sudo ufw status verbose
systemctl is-enabled ufw
systemctl is-active ufw
```


Managing UFW rules (Add/Remove)

Adding rules

```
sudo ufw allow 80/tcp                  # Allow HTTP
sudo ufw allow from 192.168.1.100      # Allow all traffic from a specific IP
sudo ufw allow from 10.0.0.0/24 to any port 22 proto tcp
sudo ufw allow 'OpenSSH'
```

Deleting rules

```
sudo ufw delete allow 80/tcp
sudo ufw delete 3            # Deletes rule number 3
```

Default policy and rule application order

Default policies

By default, UFW sets:

```
Incoming: DENY
Outgoing: ALLOW
Forwarded: DENY
```

These can be changed with:

```
sudo ufw default deny incoming
sudo ufw default allow outgoing
```


Rule evaluation order

Understanding the order of rule evaluation is essential for building reliable firewall policies. UFW, applies rules in a specific sequence by generating and organizing iptables rules across several configuration files.

When a packet reaches the system, UFW evaluates it according to the following order of rule files:

- /etc/ufw/before.rules
- User-defined rules (via ufw CLI, stored in /etc/ufw/user.rules)
- /etc/ufw/after.rules
- Default policies

Each file maps to the appropriate chains in the iptables filter, nat, or mangle tables.

First-match logic

UFW uses a first-match-wins logic. This means:

- As soon as a packet matches a rule, no further rules are evaluated.
- Therefore, specific rules must be placed before general ones (e.g., allow specific IP before denying an entire subnet).

Example:

```
sudo ufw deny from 192.168.1.0/24
sudo ufw allow from 192.168.1.100
```

In this case, traffic from 192.168.1.100 will be blocked, because the broader deny rule comes before the specific allow rule.

To resolve this, reverse the rule order:

```
sudo ufw delete 1   # Or delete the earlier deny
sudo ufw allow from 192.168.1.100
sudo ufw deny from 192.168.1.0/24
```


Viewing rule order

To inspect the effective rule order, use:

```
sudo ufw status numbered
```

This displays rules in the order of application and allows easy deletion or reordering.

Summary of good practices

- Always allow SSH before enabling UFW, especially on remote systems.
- Use ufw status numbered to manage rules easily.
- Enable logging with ufw logging on for better visibility.
- Avoid using nftables or firewalld when UFW is active.
- Regularly review /etc/ufw/user.rules and backup custom before.rules.

II.3. Fail2ban

Fail2ban is a security tool designed to protect servers and services from brute-force attacks and other suspicious behaviors by monitoring log files in real-time. When repeated failed authentication attempts or other predefined malicious patterns are detected, Fail2ban automatically blocks the offending IP address by updating firewall rules or other mechanisms.
Documentation: https://github.com/fail2ban/fail2ban/wiki

Important Files and Directories

- */etc/fail2ban/*: Main configuration directory for Fail2ban. It contains all subdirectories and configuration files for defining filters, jails, actions, and general settings.
- */etc/fail2ban/fail2ban.local:* Main configuration for the Fail2ban daemon (logging, database location).
- */etc/fail2ban/jail.local*: Default jail configuration file (should not be edited directly).
- */etc/fail2ban/jail.d/*.local* : Place for custom jail configurations.
- */etc/fail2ban/filter.d/*.local* : Directory containing filter definitions for various services.
- */etc/fail2ban/action.d/*.local* : Directory defining possible actions Fail2ban can take when banning IPs.
- */var/log/fail2ban.log*: Log file where Fail2ban records its own activities.

Core Concepts and Architecture

Jails

A jail ties together a filter, the log files to monitor, and the actions to take. Jails are configured in /etc/fail2ban/jail.conf or preferably in /etc/fail2ban/jail.d/*.local for customizations.

Each jail defines:

- The service to protect (e.g., sshd, apache).
- Which log files to scan.
- Which filter to use.
- The ban time and retry limits.
- The action to take when a rule is triggered.

Example:

```
#/etc/fail2ban/jail.d/nginx-404.local

[nginx-404]
enabled  = true  # Activate this jail
port     = http,https  # Ports to protect
filter   = nginx-404  # Refers to the filter file: /etc/fail2ban/filter.d/nginx-404.local
logpath  = /var/log/nginx/access.log  # Nginx access log to monitor
maxretry = 10  # Number of 404s allowed before banning
findtime = 5m  # Time window for counting failures
bantime  = 24h  # Duration of ban
action = iptables-mail[name=YourService, port=22, protocol=tcp]  # Block using iptables
```

Actions

When a jail detects repeated offenses, Fail2ban executes one or more actions. Common actions include:

- Adding firewall rules to block the offending IP (iptables, nftables, firewalld).
- Sending email notifications to administrators.
- Custom actions as writing malicious IP in a file.

Actions are configured in /etc/fail2ban/action.d/*local.

Example:

```
#/etc/fail2ban/action.d/iptables-mail.local

[Definition]
actionstart = iptables -N fail2ban-<name>
              iptables -A fail2ban-<name> -j RETURN
              iptables -I INPUT -p tcp --dport <port> -j fail2ban-<name>

actionstop = iptables -D INPUT -p tcp --dport <port> -j fail2ban-<name>
             iptables -F fail2ban-<name>
             iptables -X fail2ban-<name>

actionban = iptables -I fail2ban-<name> 1 -s <ip> -j DROP
            echo "Subject: Fail2ban alert - banned IP <ip>" | sendmail admin@example.com

actionunban = iptables -D fail2ban-<name> -s <ip> -j DROP
```

Filters (Rules)

Filters define the search patterns used to identify malicious behavior in logs. These are usually regular expressions stored in filter files (e.g., /etc/fail2ban/filter.d/nginx-404.local) that match lines indicating failure or attacks.

Example:

```
#/etc/fail2ban/filter.d/nginx-404.local

[Definition]

# Fail2ban filter to catch authentication failures and suspicious activity in nginx logs

failregex = ^<HOST> -.*"(GET|POST).*(wp-login\.php|xmlrpc\.php|wp-admin|administrator|login).*" 401
            ^<HOST> -.*"(GET|POST).*\.php.*" 404

ignoreregex =
```


Using fail2ban-client

fail2ban-client is the command-line utility used to interact with the Fail2ban daemon (fail2ban-server). It allows administrators to manage jails, query status, and control Fail2ban without restarting the service. The main purpose of the tool is to:

- Manage Fail2ban dynamically without needing to edit configuration files or restart the service.
- Query the status of jails, bans, and logs.
- Manually ban or unban IP addresses.
- Reload or restart Fail2ban configurations.

```
fail2ban-client -vvd                        # Runs Fail2ban client in verbose debug mode for troubleshooting.
fail2ban-client status                      # Shows the general status of Fail2ban and lists active jails.
fail2ban-client status <jail>               # Shows detailed status of a specific jail (e.g., banned IPs, log paths).
fail2ban-client start                       # Starts the Fail2ban daemon.
fail2ban-client stop                        # Stops the Fail2ban daemon.
fail2ban-client reload                      # Reloads the Fail2ban configuration files without stopping the service.
fail2ban-client restart                     # Restarts the Fail2ban daemon (stop + start).
fail2ban-client set <jail> banip <IP>       # Manually bans a specific IP address in a given jail.
fail2ban-client set <jail> unbanip <IP>     # Manually removes a ban on a specific IP address in a given jail.
fail2ban-client set sshd unbanip <ip>       # Unbans a specific IP address from the sshd jail (requires sudo).
```


II.4. CIS Benchmarks

The CIS Benchmarks are a set of best practices and security configuration guidelines developed by the Center for Internet Security (CIS). These benchmarks aim to harden systems, services, and applications to reduce vulnerabilities and protect against common threats.

Figure 6 - Logo CIS Benchmarks

Source: https://www.cisecurity.org/communities/benchmarks

They cover a wide range of technologies including:

- *Operating systems* (Linux, Windows, macOS)
- *Applications* (Apache, Docker, Kubernetes, etc.)
- *Network devices* (routers, firewalls)

Documentation:

- https://www.cisecurity.org/cis-benchmarks
- https://downloads.cisecurity.org/#/

The CIS Benchmarks are created through a community consensus process involving:

- Cybersecurity professionals
- System administrators
- Vendors (e.g., Red Hat, Microsoft)
- Public and private organizations

They are frequently updated and are often used as baseline references for compliance frameworks like ISO 27001, PCI-DSS, HIPAA, or NIST.

Each benchmark document is divided into:

- Sections: e.g., "Initial Setup", "User Accounts and Authentication", "Logging and Auditing"
- Recommendations: Numbered rules with descriptions, impact, rationale, remediation steps, and audit commands
- Levels:
  - Level 1: Minimal impact, general best practices
  - Level 2: More restrictive, ideal for highly secure environments

Implementation Options

You can audit or apply CIS Benchmarks using:

- Manual application (based on the benchmark PDF)
- Automated tools such as:
  - CIS-CAT (official CIS tool – free and paid version)
  - OpenSCAP (SCAP-compatible benchmark scanning)
  - Ansible roles and customs scripts
  - Commercials solutions (Qualys…)

OpenSCAP compliant audit for ubuntu ≥ 24.04

OpenSCAP is a powerful open-source framework that implements the Security Content Automation Protocol (SCAP), a NIST standard for automating security assessment and compliance reporting. It is designed to help organizations evaluate, harden, and audit the security of their systems in a structured and repeatable way.

Documentation:

- https://github.com/OpenSCAP/openscap
- https://github.com/ComplianceAsCode/content

- Installation:

```
sudo apt install openscap-scanner git cmake xsltproc libxml2-utils
```

- Clone the ComplianceAsCode content repository from GitHub

This repository contains the SCAP security content needed for compliance auditing.

```
git clone -b master https://github.com/ComplianceAsCode/content.git
```

- Building the SCAP Content

Once the repository is cloned, navigate into the content directory and compile the SCAP content using cmake. This step prepares the security data stream files for evaluation.

```
cd content
cmake -B build
cmake --build build
```

- Listing Available Security Profiles

To see which security profiles are available for auditing, use the oscap info command on the generated data stream file (e.g. ssg-ubuntu2404-ds.xml).

Profiles define different compliance standards or baselines.

```
oscap info build/ssg-ubuntu2404-ds.xml
```

- Applying a Compliance profile

You can automatically generate and apply remediation scripts to bring the system into compliance with a specific security profile.

```
sudo oscap xccdf generate fix \ --profile xccdf_org.ssgproject.content_profile_cis_level1_server \ build/ssg-ubuntu2404-ds.xml >
remediation.sh
sudo bash remediation.sh
```


- Performing a Compliance Audit

Run a compliance audit against the system using the oscap xccdf eval command. Replace the --profile option value with the desired profile name you retrieved earlier. The audit generates both a result XML and an HTML report.

```
sudo oscap xccdf eval \
  --profile xccdf_org.ssgproject.content_profile_cis_level1_server \
  --results results.xml \
  --report report.html \
  build/ssg-ubuntu2404-ds.xml
```

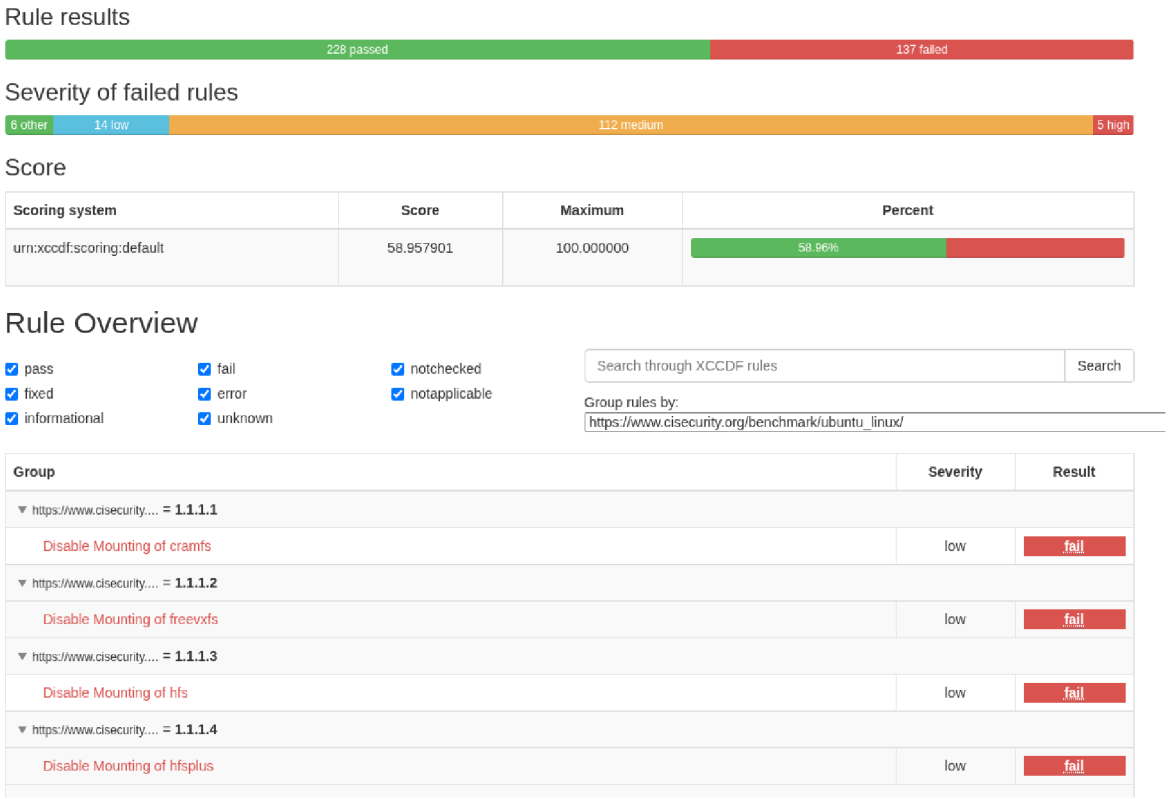

Figure 7 - OpenSCAP Evaluation Report example for Ubuntu 24.04

- Auditing a Specific Security Rule

To focus the audit on a single rule, use the --rule option with the rule identifier. This allows targeted compliance checks for specific security settings.

```
sudo oscap xccdf eval \
  --profile xccdf_org.ssgproject.content_profile_cis_level1_server \
  --rule xccdf_org.ssgproject.content_rule_grub2_enable_apparmor \
  build/ssg-ubuntu2404-ds.xml
```

Considerations Before Applying CIS rules

- Not all recommendations are suitable for all environments (e.g., disabling USB or adding password at boot).
- Benchmarking tools may require root access.
- Changes can break existing workflows if applied blindly, always test on staging environments.

You should always perform a thorough assessment and backup of the target system before applying any hardening measures.