Passwords and Logins

Notes

Authentication is usually described as a simple check: does the password match? This chapter reframes authentication as a lifecycle problem rather than a string comparison.

Passwords and other credentials are assets. They are created, stored, copied, reused, leaked, rotated, and retired. Most real compromises occur when that lifecycle is poorly controlled, not when cryptography fails.

If a credential leaks, an attacker does not need to break in. They can simply be you.

Core idea

Authentication answers one question:

> *Who are you?*

In a real environment, that question is incomplete. A complete authentication discussion must also answer:

- Where did this secret originate?
- Where is it stored or copied?
- How long does it remain valid?
- What else becomes accessible if it leaks?

A strong password policy helps, but most compromise paths come from:

- Password reuse across systems
- Phishing and social engineering
- Malware or credential theft on endpoints
- Secrets copied into scripts, tickets, documents, and chat

Brute-force guessing is only one failure mode, and often not the most important one.

Credentials as assets

A credential is not just a gate to an asset. It is an asset in itself.

That means the same questions apply as for any other asset:

- Who should have access to it?
- Where does it live at rest?
- How many copies exist?

- What is the blast radius if it leaks?
- What must be done to contain damage?

If those answers are unclear, the environment is already carrying unmanaged risk.

What Linux actually stores

Linux deliberately separates identity information from password verification material.

`/etc/passwd` – public identity registry

- One line per account
- Readable by all users
- Needed to map numeric IDs to names

Typical fields:

- username
- password placeholder (usually x)
- UID
- GID
- GECOS (comment)
- home directory
- login shell

Key point: modern Linux does not store password hashes here.

/etc/shadow – protected password registry

- Readable only by privileged processes
- Stores password verification data and aging rules

Typical fields:

- username
- password hash (or lock marker)
- last password change
- minimum change interval
- maximum age
- warning period
- inactivity period
- expiration date

Key point: the value stored here is not a password. It is a hash representation used for verification.

How password verification works (high-level)

1. A user types a password.
2. The system reads the stored hash entry from `/etc/shadow`.
3. The hash format reveals the scheme parameters and salt.
4. The typed password is hashed using the same scheme and salt.
5. The results are compared.

If they match, authentication succeeds.

The system never needs to store or retrieve the raw password.

Common hash formats in /etc/shadow

A modern hash entry typically looks like:

```
$id$salt$hash
```

Common identifiers:

- `$1$` → md5crypt (legacy, weak)
- `$5$` → sha256crypt
- `$6$` → sha512crypt
- `$y$` (or `$7$`) → yescrypt (modern default on many distributions)
- `$2y$`, `$2a$` → bcrypt (common in applications)

You do not need to memorize identifiers. You must be able to:

- Recognize the structured format
- Identify which scheme is in use
- Understand why scheme choice affects brute-force resistance

Salt

A salt is a random value stored alongside the hash.

Its purpose is uniqueness, not secrecy.

Salts are visible by design because the system must reuse them during verification.

Salts prevent two significant problems:

1. Password reuse detection. Without salts, identical passwords produce identical hashes.
2. Rainbow table attacks. Precomputed hash tables stop working when each password requires a fresh computation.

Important: salts do not stop brute force. Resistance comes from:

- Slow, password-oriented hash schemes
- Sufficient password length

Password strength: length beats complexity

Password guessing space grows exponentially with length:

    (character set size) ^ (length)

Two useful character sets:

- ~49 characters (restricted set)
- ~94 characters (printable ASCII)

The practical takeaway:

- Short passwords fail quickly
- Long passwords dominate brute-force resistance
- Complexity rules matter far less than length

Practical interpretation”

- 8 characters: unacceptable for high-value accounts
- 12 characters: reasonable minimum if unique and random
- 16+ characters: brute force becomes irrelevant
- 20–24 characters: ideal for password-manager-generated secrets

At sufficient length, other failure modes dominate (reuse, leakage, phishing).

How attackers actually get passwords

Reuse

One leaked password often unlocks multiple systems.

Phishing

Attackers bypass cryptography by convincing humans to hand over secrets.

Endpoint theft

Malware, browser stores, screenshots, and clipboard capture.

Dictionary attacks

Common passwords and patterns are tried first.

Brute force (mostly offline)

Relevant primarily when hashes are stolen, and rate limits disappear.

Good practices

- Use a password manager for unique, generated passwords
- Prefer length over complexity
- Never reuse passwords across systems
- Treat copied credentials as data spills
- Use MFA where available
- Assume hashes can leak; design accordingly

Examples

Shared Ops Document Exposes a Service Password

A production service relies on a long-lived credential to authenticate to a sensitive backend system. During routine on-call activity, operators maintain a shared runbook to record commands, procedures, and troubleshooting notes. To speed incident response, a valid service password is pasted directly into the document, and the document remains broadly readable to the operations group and adjacent teams.

Confidentiality of the protected backend system and its data

→ Routine on-call troubleshooting and documentation updates

→ Shared operational document readable by a wide audience

→ Document contains a valid, active service password

→ Any reader of the document can authenticate to the sensitive system outside approved workflows

→ Backend data becomes accessible to unauthorized identities

This is a confidentiality issue because the service continues to function normally, and no data is altered or destroyed. The failure is that protected access credentials become visible to parties that should not possess them. The observable damage is unauthorized access to sensitive systems or data, often discovered only after audit review or an external incident.

Hardcoded Maintenance Credential Enables Unauthorized Data Modification

A maintenance script is used periodically to perform administrative tasks against a production database. To avoid interactive prompts, a database credential with write privileges is embedded directly in the script. The script is stored in a shared repository and is occasionally copied, modified, or executed by different operators during routine maintenance or troubleshooting.

Integrity of production database records

→ Routine maintenance workflow using shared scripts

→ Script contains a hardcoded database password with write access

→ Credential enables direct modification of production data

→ Script is executed incorrectly, modified unintentionally, or run by an unauthorized individual

→ Production data is altered outside approved change controls

This is an integrity issue because the primary harm is unauthorized or incorrect data modification. The system remains available and accessible, but the data's correctness and trustworthiness are compromised. The observable damage is unexpected data changes, inconsistencies, or corruption that may only become apparent through downstream effects or manual investigation.

Aggressive Account Lockout Causes Widespread Authentication Failure

An authentication service enforces strict account lockout rules to prevent brute-force attacks. The policy was configured without sufficient consideration of peak usage patterns, shared networks, or common user behaviour, such as repeated retries during latency or password manager failures. During a high-traffic period, many legitimate users trigger the lockout threshold.

Ability for users to authenticate and access the service

→ Normal login attempts during peak usage

→ Lockout policy too strict for real-world conditions

→ Multiple legitimate accounts become locked

→ Users are unable to authenticate, even with correct credentials

→ Service is effectively unavailable to a large portion of users when needed

This is an availability issue because the core failure is the inability to deliver the required function, user authentication, under normal operating conditions. No data is exposed or modified. The observable damage is widespread user access failure, increased support load, and service disruption until accounts are unlocked or policies are adjusted.

Reused Admin Credential Produces Cascading Failures Across Systems

An administrative credential is reused across multiple systems to simplify management. During routine support work, a screenshot attached to a ticket unintentionally exposes the admin password. The ticketing system is accessible to a broad audience and retains attachments indefinitely. The exposed credential is later obtained by an unauthorized party.

Integrity, confidentiality, and availability of multiple systems

→ Routine support workflow with ticket screenshots

→ Screenshot exposes a high-privilege admin password

→ Same credentials reused across multiple systems

→ Attacker gains broad administrative access

→ Records are modified, sensitive data is extracted, and services are disrupted

→ Incident response requires emergency containment, resets, and downtime

This is a cascading failure scenario with the primary classification of integrity. A single credential compromise produces multiple classes of harm:

- Integrity: unauthorized changes to records and configurations
- Confidentiality: unauthorized access to sensitive data
- Availability: service disruption during containment and recovery

The defining feature is that a single exposed credential causes multiple, compounding failures across otherwise independent systems. The observable damage includes data tampering, data loss, operational outages, and prolonged recovery efforts.

What to notice

- Authentication fails at the weakest point, where the raw credential appears
- Passwords have origins, locations, and lifetimes
- Hardcoded or copied secrets bypass login controls entirely
- Strong policies do not matter if secrets are handled casually
- Credentials must be tracked and protected like any other critical asset