CSL 301
OPERATING SYSTEMS

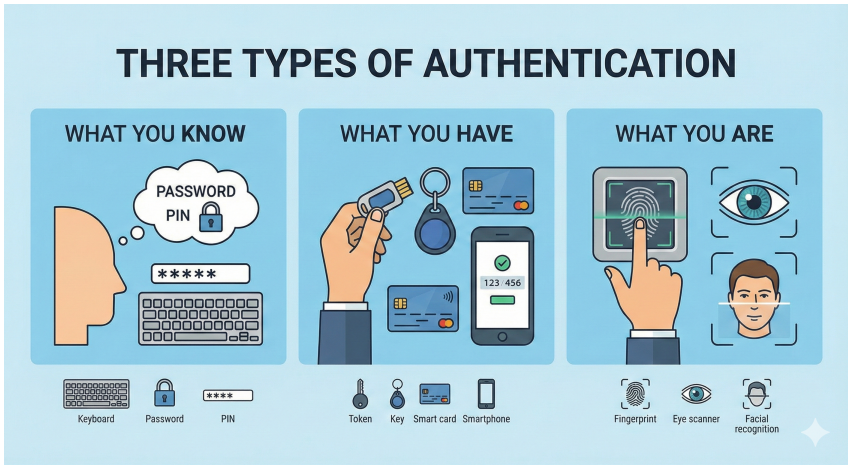

Image Source: Gemini

Lecture 28
Authentication in Operating Systems

Instructor
Dr. Dhiman Saha

The Need for Authentication

- Operating systems provide services, and many have security implications.
- The OS must decide whether to perform an action or deny it based on security policy.
- **Context is everything!** The most important context is *who* is asking.

The Need for Authentication

- Operating systems provide services, and many have security implications.
- The OS must decide whether to perform an action or deny it based on security policy.
- **Context is everything!** The most important context is *who* is asking.

Key Terminology

- **Principal**: The entity (e.g., human user, group) making a request.
- **Agent**: A process executing on behalf of a principal.
- **Object**: The resource being accessed (e.g., file, network port).

The Need for Authentication

- Operating systems provide services, and many have security implications.
- The OS must decide whether to perform an action or deny it based on security policy.
- **Context is everything!** The most important context is *who* is asking.

Key Terminology

- **Principal**: The entity (e.g., human user, group) making a request.
- **Agent**: A process executing on behalf of a principal.
- **Object**: The resource being accessed (e.g., file, network port).

The Fundamental Problem

How does the operating system reliably determine the identity of the principal behind a request to enforce security goals?

Attaching Identities to Processes

- Processes are the actors in an OS. Everything is done by a process.
- If we can securely identify processes, we can enforce security policies.
- **Identity Inheritance**: Typically, a new process (child) inherits the identity of its creator (parent).

Attaching Identities to Processes

- Processes are the actors in an OS. Everything is done by a process.
- If we can securely identify processes, we can enforce security policies.
- **Identity Inheritance**: Typically, a new process (child) inherits the identity of its creator (parent).

The Starting Point: User Login

How does the very first process for a user (e.g., the command shell) get its identity?

- The user must first **log in**.
- The OS must be able to verify the identity of the human user.
- This verified identity is then attached to the user's initial process.

Crux

The entire security of the system relies on the OS making the right decision at the authentication stage.

The Three Factors of Authentication

Classically, a user's identity can be verified based on:

1. What You Know

A secret, like a password or a PIN.

2. What You Have

A physical object, like a key card, smartphone, or USB token.

3. What You Are

A biological characteristic (biometrics), like a fingerprint or retina scan.

Factor 1: What You Know (Passwords)

- The most common form of authentication.
- A secret shared between the user and the system.

Factor 1: What You Know (Passwords)

- The most common form of authentication.
- A secret shared between the user and the system.

Major Security Flaw: How to Store the Secret?

If the system stores the password in plaintext, anyone who accesses the password file learns all user passwords. This is a huge vulnerability.

Factor 1: What You Know (Passwords)

- The most common form of authentication.
- A secret shared between the user and the system.

Major Security Flaw: How to Store the Secret?

If the system stores the password in plaintext, anyone who accesses the password file learns all user passwords. This is a huge vulnerability.

Solution: Store Hashes, Not Passwords

- A **hash function** is a one-way function: easy to compute, but infeasible to reverse.
- **Storage**: The system stores ‘hash (password)‘.
- **Verification**: When a user provides a password, the system computes its hash and compares it with the stored hash.
- If an attacker steals the password file, they only get the hashes, not the passwords themselves.

Attacking Hashed Passwords

Even with hashing, passwords can be weak.

Problem 1: Guessing Attacks

- **Brute-force**: Trying all possible combinations. Infeasible for long, complex passwords.
- **Dictionary Attack**: Trying common words, names, and simple patterns (e.g., "password123"). Surprisingly effective as humans choose memorable (and thus predictable) passwords.

Attacking Hashed Passwords

Even with hashing, passwords can be weak.

Problem 1: Guessing Attacks

- **Brute-force**: Trying all possible combinations. Infeasible for long, complex passwords.
- **Dictionary Attack**: Trying common words, names, and simple patterns (e.g., "password123"). Surprisingly effective as humans choose memorable (and thus predictable) passwords.

Problem 2: Pre-computed Hashes (Rainbow Tables)

- If an attacker steals the password file, they can guess passwords offline.
- An attacker can pre-compute the hashes of a large dictionary of words.
- Then simply compare the stolen hashes against their pre-computed list to find matches instantly.

Defense: Password Salting

The Problem

With a standard hash, the same password always produces the same hash, making pre-computation attacks possible.

Defense: Password Salting

The Problem

With a standard hash, the same password always produces the same hash, making pre-computation attacks possible.

Solution: Use a "Salt"

- A **salt** is a unique, random value generated for each user.
- **Storage**: The system stores the user's salt (in plaintext) and 'hash(password + salt)'.
- **Verification**: The system retrieves the user's salt, concatenates it with the provided password, and hashes the result for comparison.

Defense: Password Salting

The Problem

With a standard hash, the same password always produces the same hash, making pre-computation attacks possible.

Solution: Use a "Salt"

- A **salt** is a unique, random value generated for each user.
- **Storage**: The system stores the user's salt (in plaintext) and 'hash(password + salt)'.
- **Verification**: The system retrieves the user's salt, concatenates it with the provided password, and hashes the result for comparison.

How does this help? An attacker must now compute hashes for every possible salt. If the salt is large enough, pre-computing a rainbow table becomes infeasible. The dictionary attack must be run separately for each stolen password, using its unique salt.

Factor 2: What You Have (Tokens)

Authentication based on possessing a physical object.

- **Examples**: ATM cards, smart cards, USB security tokens (e.g., YubiKey), smartphones.
- Often use a changing piece of information (like a one-time password or a cryptographic challenge-response) to prevent an attacker from simply copying the secret.
- This converts a static "what you know" (the secret on the device) into a dynamic "what you have".

Factor 2: What You Have (Tokens)

Authentication based on possessing a physical object.

- **Examples**: ATM cards, smart cards, USB security tokens (e.g., YubiKey), smartphones.
- Often use a changing piece of information (like a one-time password or a cryptographic challenge-response) to prevent an attacker from simply copying the secret.
- This converts a static "what you know" (the secret on the device) into a dynamic "what you have".

Primary Weakness

What if you lose the device?

- You are locked out.
- An attacker who finds it might be able to impersonate you.

Factor 3: What You Are (Biometrics)

Authentication based on a unique physical or behavioral trait.

- **Examples**: Fingerprint, face recognition, iris/retina scan, voice print, typing pattern.

Factor 3: What You Are (Biometrics)

Authentication based on a unique physical or behavioral trait.

- **Examples**: Fingerprint, face recognition, iris/retina scan, voice print, typing pattern.

The Challenge: Imprecision

Biometric measurements are never exactly the same twice.

- A simple bit-for-bit comparison (like with a password hash) will fail.
- The system must allow for some "tolerance" or sloppiness.

Factor 3: What You Are (Biometrics)

This leads to two types of errors:

- **False Negative**: The system fails to recognize a legitimate user. (Annoying for the user).
- **False Positive**: The system incorrectly authenticates an imposter. (Disastrous for security).

Crossover Error Rate (CER)

There is an inherent trade-off. Tuning the system to be more secure (lower false positives) often makes it less convenient (higher false negatives). The CER is a metric for the accuracy of a biometric system.

Multi-Factor Authentication (MFA)

Security is significantly increased by requiring evidence from **more than one** authentication factor.

Example: ATM Transaction

1. **What you have**: The physical ATM card.
2. **What you know**: The Personal Identification Number (PIN).

Multi-Factor Authentication (MFA)

Security is significantly increased by requiring evidence from **more than one** authentication factor.

Example: ATM Transaction

1. **What you have**: The physical ATM card.
2. **What you know**: The Personal Identification Number (PIN).

Example: Modern Web Login

1. **What you know**: Your password.
2. **What you have**: A one-time code sent to your smartphone.

Multi-Factor Authentication (MFA)

Security is significantly increased by requiring evidence from **more than one** authentication factor.

Example: ATM Transaction

1. **What you have**: The physical ATM card.
2. **What you know**: The Personal Identification Number (PIN).

Example: Modern Web Login

1. **What you know**: Your password.
2. **What you have**: A one-time code sent to your smartphone.

Benefit of MFA

An attacker must compromise multiple, independent factors. Stealing your password is not enough if they don't also have your phone.

Authenticating Non-Humans

- Not all processes are run by a human user (e.g., a web server, a database service).
- These services still need an identity to manage their privileges.

Authenticating Non-Humans

- Not all processes are run by a human user (e.g., a web server, a database service).
- These services still need an identity to manage their privileges.

Solution

- Create a dedicated "user account" for the service (e.g., a user named ‘webserver‘).
- This account has only the privileges necessary for its task (Principle of Least Privilege).

Authenticating Non-Humans

How does the service get this identity?

- A system administrator logs in and starts the service.
- Privileged commands like `sudo` can be used to launch a process under a different user identity.

```
sudo -u webserver apache2
```

- The ‘apache2‘ process (and any children it creates) will run with the ‘webserver‘ user’s identity and privileges.

Example: Linux Login Procedure

1. A privileged ‘login‘ process prompts for a username.
2. The process then prompts for the password (which is not echoed to the screen).
3. It looks up the username in the system's password database (e.g., `/etc/shadow`).
   - If the user is not found, the login fails.
   - If found, it retrieves the user's **salt** and stored **password hash**.
4. The ‘login‘ process hashes the entered password with the retrieved salt and compares it to the stored hash.
   - If they do not match, the login fails.
5. **If they match**:
   - It calls `fork()` to create a new process.
   - This new process's user and group ID are set to that of the authenticated user.
   - It then calls `exec()` to replace itself with the user's default shell (e.g., `/bin/bash`), which now runs with the user's identity.

Summary

- Authentication is the bedrock of OS security, used to verify the identity of a principal.
- The process begins with a user login, which attaches a verified identity to an initial process. This identity is then inherited by child processes.
- Authentication relies on three factors: something you **know**, **have**, or **are**.
- Passwords (*what you know*) must be stored securely using **hashes** and **salts** to protect against theft and guessing attacks.
- Biometrics (*what you are*) are powerful but must manage the trade-off between security (false positives) and convenience (false negatives).
- **Multi-Factor Authentication (MFA)** provides the strongest security by combining two or more factors.