# Authentication Methods

Leonardo Araújo

UFSJ

# Introduction

- **Authentication** is the process of verifying the identity of a user or system.
- Methods are designed to ensure secure access to resources.
- Commonly categorized into three factors:
    1. Something you know
    2. Something you have
    3. Something you are

# Traditional Authentication Factors

Something You Know

- Examples:
  - Passwords
  - Passphrases
  - PINs
- Advantages:
  - Simple to implement
  - Easy to use
- Disadvantages:
  - Can be guessed or stolen

Passwords

- Short and typically composed of:
  - Letters, numbers, and special characters.
- Examples:
  - `P@ssw0rd`
  - `12345678`
- Advantages:
  - Easy to remember for simple accounts.
- Disadvantages:
  - Weak passwords are easily guessed or cracked.
  - Users often reuse them across multiple accounts.

## Passphrases

- Longer, phrase-like combinations of words:
  - Example: `CorrectHorseBatteryStaple`
- Advantages:
  - Harder to crack due to length and complexity.
  - Easier to remember compared to complex passwords.
- Disadvantages:
  - Users may use predictable phrases.
  - Longer input may be inconvenient in some contexts.

## Something You Have

- Examples:
  - Security tokens
  - Smartcards
- Advantages:
  - Harder to replicate
  - Can be physical or digital
- Disadvantages:
  - Lost or stolen items compromise security

## Something You Are

- Examples:
  - Fingerprints
  - Facial recognition
- Advantages:
  - Unique to individuals
  - Hard to forge
- Disadvantages:
  - Privacy concerns
  - High setup cost

# Emerging and Complementary Methods

Somewhere You Are

- Based on geolocation or network.
- Example: Restricting access by IP or GPS location.

Something You Do

- Behavioral patterns.
- Example: Keystroke dynamics or touchscreen gestures.

Time-Based Access

- Access restricted to specific times.
- Example: Office hours access.

# Secure Storage of Authentication Data

- Authentication data, especially **passwords/passphrases**, must not be stored in plain text.
- Proper storage mechanisms reduce the impact of data breaches.
- Key concepts:
  - **Salting**
  - **Hashing**
  - **Secret Vaults**

## Salting

- **What is a salt?**
  - A random string added to authentication data before hashing.
- **Purpose:**
  - Prevents attackers from using precomputed tables (e.g., rainbow tables).
- **Example:**
  - Password: `password123`
  - Salt: `aeEcax2Usjdp09S2vn`
  - Salted Input: `aeEcax2Usjdp09S2vnpassword123`
- **Important:**
  - Each user should have a unique salt.
  - Store salts alongside the hash.

## Hashing

- **What is hashing?**
  - A one-way transformation of data into a fixed-length value.
- **Common algorithms:**
  - Modern: `bcrypt`, `Argon2`, `PBKDF2`
  - Avoid: `MD5`, `SHA1` (considered weak for passwords).
- **Key considerations:**
  - Use **adaptive hashing** (increases computational cost as hardware improves).
  - Combine with salting for strong security.
- **Example Workflow:**
  - Input: `aeEcax2Usjdp09S2vnpassword123`
  - Hashed Output: `d1f56e8e8d...`

## Secret Vaults

- **Purpose:**
  - Secure storage for sensitive data, such as salts, API keys, or encryption keys.
- **Features:**
  - Access control and auditing.
  - Encryption of stored data.
- **Popular tools:**
  - **HashiCorp Vault**
  - **AWS Secrets Manager**
  - **Azure Key Vault**
- **Usage in Authentication:**
  - Protect master keys used for encryption/decryption.
  - Store critical credentials securely.

# Important Best Practices

1. **Never reuse salts**:
   - Each credential requires a unique salt.
2. **Choose strong hashing algorithms**:
   - Use industry-recognized methods with sufficient iteration counts.
3. **Limit access to the vault**:
   - Restrict access based on roles and maintain audit logs.
4. **Regularly rotate secrets**:
   - Minimize exposure in case of breaches.
5. **Encrypt stored hashes**:
   - Add an extra layer of security, especially for sensitive environments.

## Common Pitfalls to Avoid

- Storing plain text passwords or unsalted hashes.
- Using predictable or hardcoded salts.
- Overlooking hardware acceleration attacks (e.g., GPUs).
- Failing to secure the vault itself.
- Not updating hashing algorithms over time.

# Practical Implementation Example

- **User Registration:**
  1. Generate unique salt for the user.
  2. Combine password and salt, then hash.
  3. Store the hash and salt in the database.
- **User Login:**
  1. Retrieve the stored salt and hash.
  2. Combine user-provided password with salt.
  3. Hash and compare with stored hash.

# Conclusion

- Multifactor authentication improves security by combining methods.
- Emerging technologies continue to enhance authentication.
- Balancing usability and security is key.
- Secure storage protects authentication data from exposure.
- Salting and hashing make passwords resistant to common attacks.
- Secret vaults ensure sensitive data is stored and accessed securely.
- Regular updates and audits strengthen overall security.

# References

1 NIST SP 800-63B: Digital Identity Guidelines.

2 OWASP Authentication Cheat Sheet.