SLH - Sécurité logicielle haut niveau

Authentification

06 décembre 2025

Table des matières

1 Introduction 1
1.1 Triangle d'impossibilité de Zooko 2
1.2 Usernames 2
1.3 Facteurs d'authentification 2
2 Mots de passe 2
2.1 Politiques modernes (NIST SP 800-63B) 3
2.2 Validation 3
3 Stockage sécurisé 3
3.1 Principes 4
3.2 Algorithmes 4
3.3 Sel (Salt) 4
3.4 Poivre (Pepper) 4
3.5 Migration d'algorithme 4
4 Authentification multi-facteurs (MFA) 4
4.1 Méthodes 5
4.2 Mise en œuvre 5
5 Sessions 5
5.1 Tokens 6
5.2 Révocation 6
6 Fédération & SSO 6
6.1 OAuth 2.0 7
6.2 OpenID Connect (OIDC) 7
7 Fédération & SSO 7
7.1 OAuth 2.0 8
7.2 OpenID Connect (OIDC) 8
7.3 SAML 2.0 8
7.4 Bonnes pratiques 8
8 Attaques et contre-mesures 8
8.1 Credential Stuffing 9
8.2 Brute Force 9
8.3 Phishing 9
8.4 Session Hijacking 9
9 Checklist de sécurité 9

1 Introduction

L'authentification vérifie l'identité d'un utilisateur ou système avant d'autoriser l'accès aux ressources.

1.1 Triangle d'impossibilité de Zooko

Propriétés d'une identité (max 2/3) :

- **Authentifiable** : vérifiable
- **Globale** : sans coordination centrale
- **Conviviale** : mémorable pour humains

Compromis :

- **G + C** : Surnoms, pseudos (pas authentifiables)
- **A + G** : Clés publiques, Bitcoin (pas conviviaux)
- **A + C** : Username, email, DNS (coordination nécessaire)

1.2 Usernames

Bonnes pratiques :

- Unicité dans le système
- Prévenir attaques homographiques (ASCII uniquement ou normalisation Unicode)
- Normaliser la casse (généralement en minuscules)
- Valider longueur et caractères autorisés

> ⚠ **Warning**
>
> Dépendance aux tiers (email) : prévoir migration si l'utilisateur change d'adresse.

1.3 Facteurs d'authentification

Humains :

- Ce que l'on **sait** : mot de passe, PIN
- Ce que l'on **possède** : token, smartphone, carte à puce
- Ce que l'on **est** : biométrie

Machines :

- Ce que l'on **sait** : bearer token, clés API
- Ce que l'on **possède** : certificats TLS

2 Mots de passe

2.1 Politiques modernes (NIST SP 800-63B)

Recommandations :

- Autoriser tous caractères Unicode + espaces
- Longueur : min 8, max 64+ caractères
- Vérifier contre fuites (Have I Been Pwned)
- Permettre copier-coller et gestionnaires

À éviter :

- ❌ Règles de composition complexes (→ `Password1!`)
- ❌ Changement périodique forcé
- ❌ Questions secrètes

Protection DoS : longueur max 64-128, rate limiting, hachage lent

2.2 Validation

Utiliser **zxcvbn** pour estimer l'entropie et vérifier contre dictionnaires + fuites connues.

3 Stockage sécurisé

3.1 Principes

Pourquoi hacher (pas chiffrer ni stocker en clair) :

- Réutilisation des mots de passe entre sites
- Chiffrement réversible → clé compromise = tous les mots de passe compromis
- Hachage = fonction à sens unique, lente intentionnellement

3.2 Algorithmes

Recommandé : **Argon2** (résistant GPU/ASIC, paramètres temps/mémoire/parallélisme)

Acceptable : bcrypt, scrypt, PBKDF2 (100k+ itérations)

> ⚠️ **Warning**
>
> ❌ Jamais : MD5, SHA-1, SHA-256/512 seuls (trop rapides)

3.3 Sel (Salt)

Valeur aléatoire ajoutée au mot de passe avant hachage.

Objectif : empêcher les rainbow tables (tables pré-calculées de hashes)

Propriétés :

- **Unique** par utilisateur
- Min 16 octets (128 bits)
- Cryptographiquement aléatoire
- Stocké en clair à côté du hash (avec Argon2/bcrypt, encodé dans le hash)

```
CREATE TABLE users (
    username VARCHAR(255) UNIQUE,
    password_hash VARCHAR(255)  -- contient sel + hash encodés
);
```

3.4 Poivre (Pepper)

Secret **global** ajouté aux mots de passe, stocké hors BDD.

Différences avec le sel :

- **Secret** (pas en BDD, dans config/HSM)
- **Global** à toute l'application
- Protection additionnelle si BDD compromise

Inconvénient : difficile à changer (re-hash tous les mots de passe nécessaire)

3.5 Migration d'algorithme

Détecter l'algo existant au login, re-hasher avec nouvel algo si connexion réussie.

```
if user.hash.startswith('$2b$'):  # bcrypt
    if verify(password, user.hash):
        user.hash = argon2.hash(password)  # upgrade
```

4 Authentification multi-facteurs (MFA)

4.1 Méthodes

TOTP (Time-based OTP) :

- Code 6 chiffres généré toutes les 30s (RFC 6238)
- Secret partagé via QR code
- Apps : Google Authenticator, Authy
- ⚠️ Vulnérable au phishing

SMS/Email OTP :

- Familier mais vulnérable (SIM swapping, MITM)
- Non recommandé NIST pour applications critiques

WebAuthn/FIDO2 :

- Cryptographie à clé publique, dispositif physique (Yubikey)
- Résistant au phishing, support navigateurs
- **Recommandé**

Codes de récupération :

- 8-10 codes à usage unique (8-12 caractères)
- Hasher avant stockage
- Afficher une seule fois

4.2 Mise en œuvre

Flux : mot de passe → second facteur → session créée

« Remember device » pour 30 jours, redemander MFA pour actions sensibles.

5 Sessions

5.1 Tokens

Cookies : `HttpOnly` + `Secure` + `SameSite=Strict` (protection XSS/CSRF)

JWT : Auto-contenus, signés, stateless

> ⚠ **Warning**
>
> JWT difficiles à révoquer → préférer tokens opaques côté serveur pour apps critiques

5.2 Révocation

Techniques : liste noire, invalidation globale utilisateur, short-lived + refresh tokens

6 Fédération & SSO

6.1 OAuth 2.0

Protocole d'**autorisation** pour accès ressources sans exposer credentials.

Flux Authorization Code :

1. Redirection vers provider
2. Authentification + consentement
3. Redirection avec `code`
4. Échange `code` → `access_token`

6.2 OpenID Connect (OIDC)

Extension OAuth pour **authentification**.

- Ajoute `id_token` (JWT) avec identité utilisateur
- Endpoint `/userinfo`

7 Fédération & SSO

7.1 OAuth 2.0

Protocole d'**autorisation** pour accès ressources sans exposer credentials.

Flux Authorization Code :

1. Redirection vers provider
2. Authentification + consentement
3. Redirection avec `code`
4. Échange `code` → `access_token`

7.2 OpenID Connect (OIDC)

Extension OAuth pour **authentification**.

- Ajoute `id_token` (JWT) avec identité utilisateur
- Endpoint `/userinfo`
- Standard pour SSO moderne

7.3 SAML 2.0

Standard XML pour SSO entreprise.

- IdP (Active Directory) + SP (application)
- Mature, support Single Logout
- Complexe, moins adapté au web moderne

7.4 Bonnes pratiques

- Valider signatures/expiration/audience
- HTTPS obligatoire, PKCE pour OAuth
- Permettre liaison/déliaison comptes
- Gérer changement email provider

8 Attaques et contre-mesures

8.1 Credential Stuffing

Utilisation de couples username/password volés ailleurs.

Contre-mesures : rate limiting IP/compte, CAPTCHA, vérification Have I Been Pwned

8.2 Brute Force

Test systématique de mots de passe.

Contre-mesures : délai exponentiel, verrouillage temporaire, WAF

8.3 Phishing

Faux site pour voler credentials.

Contre-mesures : WebAuthn (résistant), éducation utilisateurs, monitoring domaines similaires

8.4 Session Hijacking

Vol de token de session.

Contre-mesures : cookies `HttpOnly` / `Secure`, rotation tokens, expiration courte

9 Checklist de sécurité

Mots de passe :

- [ ] Utiliser Argon2 ou bcrypt
- [ ] Sel unique par utilisateur
- [ ] Longueur min 8 caractères, max 64+
- [ ] Vérifier contre les fuites connues
- [ ] Pas de règles de composition arbitraires

MFA :

- [ ] Proposer TOTP ou WebAuthn
- [ ] Codes de récupération disponibles
- [ ] Possibilité de gérer plusieurs dispositifs

Sessions :

- [ ] Cookies `HttpOnly`, `Secure`, `SameSite`
- [ ] Expiration raisonnable (15-60 min)
- [ ] Révocation possible (logout)
- [ ] Rotation après changement de privilège

Général :

- [ ] HTTPS obligatoire
- [ ] Rate limiting
- [ ] Protection CSRF
- [ ] Logs des tentatives de connexion