TP1 cryptographie

Arnaud Champierre de Villeneuve

M1 Cybersécurité & Management

02/06/2025

Objectif pédagogique

1) Un **algorithme de hachage** génère une empreinte numérique unique et irréversible à partir d'une donnée, garantissant ainsi son intégrité. Des algorithmes comme **SHA-256** et **SHA-512** sont couramment utilisés à cette fin.

   En revanche, un **algorithme de chiffrement** transforme des données lisibles en données cryptées, rendant leur lecture impossible sans une clé de déchiffrement. Ces algorithmes assurent la confidentialité des informations et incluent des méthodes comme **RSA** et **AES**.

2) Le **HMAC** (Hash-based Message Authentication Code) repose sur une fonction de hachage combinée à une clé secrète pour générer une signature unique. Il sert à garantir l'intégrité et l'authenticité des données. Sur GitHub, les **JSON Web Tokens (JWT)** utilisent fréquemment HMAC-SHA256 pour signer les tokens, s'assurant ainsi qu'ils n'ont pas été modifiés de manière non autorisée.

3) Pour Les fichiers .env permettent de stocker des variables sensibles en dehors du code source, évitant ainsi leur exposition directe et renforçant la sécurité.
   Les middlewares d'authentification et de validation jouent un rôle clé en filtrant les requêtes: seules celles qui sont authentifiées et conformes aux règles définies sont prises en charge par l'application. Enfin, la sanitization des entrées consiste à nettoyer et sécuriser les données reçues, empêchant ainsi des attaques comme l'injection SQL ou les vulnérabilités XSS.

Serveur d'application

1) Analyser le Backend :
   a. Les principes de sécurité implémentés :

      - Authentification par JWT avec HMAC-SHA256
      - Structure MVC claire (séparation logique du code)
      - L'utilisation de bcrypt pour le hachage des mots de passe
      - Middleware d'authentification (protect) pour protéger les routes privées

   b. Analyse documentée du code - Points forts & faibles

Points forts	Points faibles
Le Hachage fort avec bcrypt	Aucune validation ou de sanitization des entrées
Le JWT est intégré dans l'authentification	Pas de middleware de protection type helmet ou rate-limit
L'utilisation d'un middleware protecteur	Pas de gestion du refresh token
	Le token du JWT est trop long (30 jours)

c. Solutions pour renforcer les contrôles (middlewares)
   - *xss-clean*, *helmet*, *express-mongo-sanitize*
   - *rate-limit* pour bloquer les attaques bruteforce
   - Limiter la durée de vie des tokens / ajout de refresh tokens
   - *express-validator* pour valider les champs (emails, mots de passe, etc…)

d. Principes de sécurité du cours
   Nous avons vu pendant le cours le principe du moindre privilège. La confidentialité avec les hash des mots de passe, les JWT non lisibles sans clé. L'Intégrité des signatures JWT via HMAC. Et la séparation des responsabilités (middlewares et controllers)

e. Caractériser l'algorithme de hachage utilisé
   bcrypt, est utilisé dans le fichier User.js, c'est un algorithme de hachage fort avec salage automatique car il résiste aux attaques par dictionnaire, style brute force et Rainbow tables.

f. Justifier sa robustesse :
   - Le salage automatique est intégré,
   - Adapté pour les mots de passe : plus lent mais il protège contre les attaques massives tel que les brutes force.

g. Caractériser l'utilisation de HMAC dans les JWT
   - JWT signés avec HMAC-SHA256 (via json web token),
   - HMAC c'est la clé secrète partagée utilisée pour signer et vérifier l'intégrité,
   - Il garantit que le token n'a pas été modifié (intégrité + authenticité)

h. Voici les alternatives à HMAC-SHA256

Alternatives	Caractéristiques
RS256 (RSA)	Asymétrique (clé privée/publique), mieux pour SSO/API
ES256	Algorithme plus léger, basé sur ECC (elliptic curves)
EdDSA (Ed25519)	Solution moderne, très sécurisé, rapide mais moins répandu