



PROJET FINAL

Audit de sécurité applicative ShopCESI



Durée : 3h30



Équipes de 3-4 personnes



Évaluation : 100% note cours

Votre mission



Vous êtes consultants en sécurité applicative

Un client vous embauche pour un audit complet de sécurité



Client : ShopCESI

Site e-commerce d'articles CESI (fictif)



Problème

A entendu parler de vulnérabilités dans les applications web

Veut être sûr que son site est sécurisé avant le lancement



Livrable attendu

Rapport d'audit professionnel

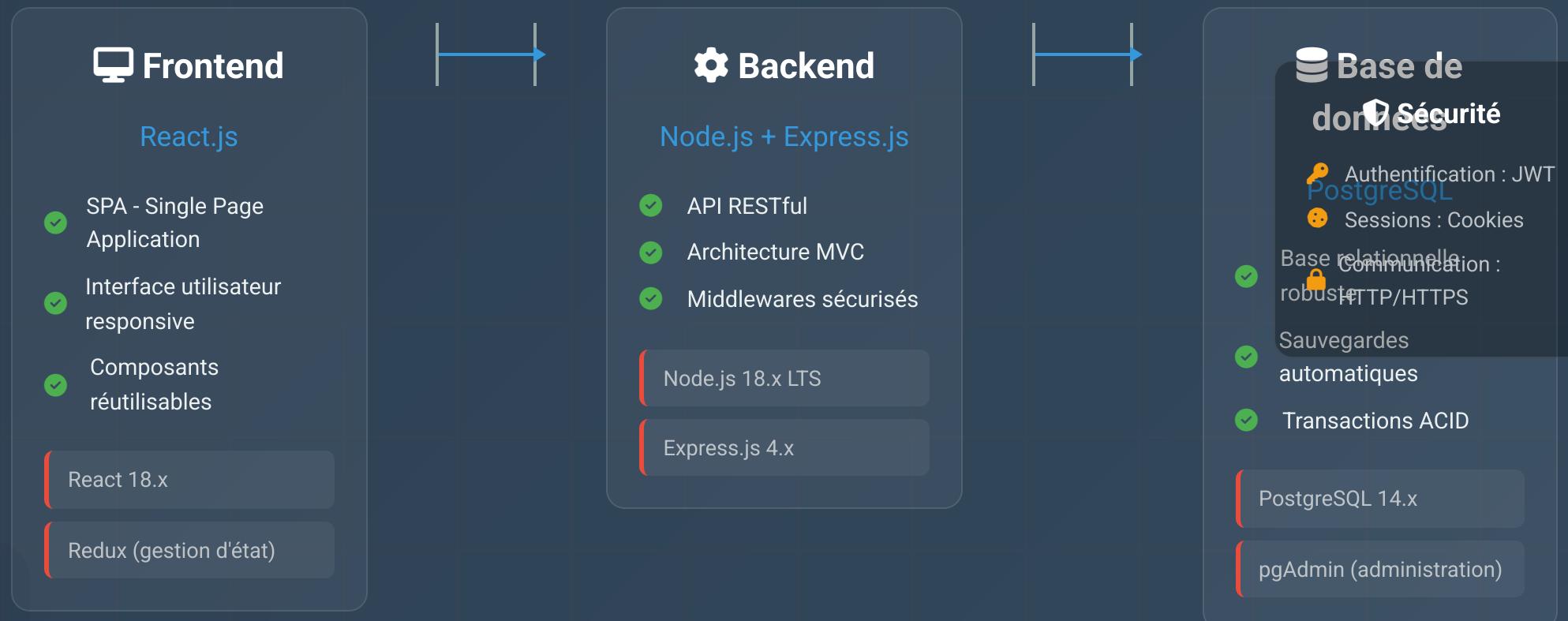
Document détaillé avec vulnérabilités identifiées et recommandations



Mission : Trouver 10-12 vulnérabilités critiques dans le code source fourni

Architecture technique de ShopCESI

Application web e-commerce - Stack technique 3-tiers





Que fait ShopCESI ?

Inscription / Connexion

Catalogue de produits

Panier d'achat

Paiement (simulation)

Historique des commandes

Panel d'administration

Système de commentaires

Profil utilisateur



Votre objectif d'audit

🎯 Mission : Trouver 10-12 vulnérabilités dans le code source

SQL Injection ×2

Requêtes non paramétrées et concaténations vulnérables

XSS ×2

Reflected et Stored - Injection de scripts malicieux

IDOR ×1

Accès aux données d'autres utilisateurs via manipulation d'ID

Broken Authentication ×1

Gestion des sessions et tokens d'authentification

Mass Assignment ×1

Attribution automatique de champs non autorisés

Excessive Data Exposure ×1

API renvoie trop de données sensibles

Secrets hardcodés ×1

Clés API, mots de passe dans le code source

CSRF ×1

Cross-Site Request Forgery - requêtes non authentifiées



Indice : Utilisez les Schémas 1 et 2 comme checklist d'audit



Code source fourni

8 fichiers de code source



login.js

Authentification
(Backend)



register.php

Inscription (Backend
alternatif)



products.js

API Produits (Backend)



orders.js

API Commandes
(Backend)



comments.js

Système de
commentaires
(Backend)



profile.js

Profil utilisateur
(Backend)



admin.js

Panel admin (Backend)



config.js

Configuration
(Backend)

Volume total : ~400-500 lignes de code

Format : Fichiers téléchargeables ou repository Git

Organisation des 3h30

Planning détaillé du Projet Final d'audit



1. Introduction (15 min)

Lecture instructions, Constitution équipes, Distribution code source

2. Analyse (1h30)

Lecture code, Identification vulnérabilités, Classification gravité



3. Rédaction (1h45)

Template rapport, PoC, Solutions techniques, Relecture



Remise du rapport : 48h après la fin du cours par email

Projet Final - Audit de sécurité ShopCESI

8/10 - Structure du rapport d'audit

Template fourni - Rapport professionnel obligatoire

Structure détaillée à respecter pour garantir la qualité du livrable

🏠 Page de garde

- ✓ Nom de l'équipe
- ✓ Noms des membres
- ✓ Date du projet

↳ Résumé exécutif (1 page)

- Synthèse pour la direction
- Nombre de vulnérabilités par gravité
- Recommandations principales

⚙️ Méthodologie

Démarche

Analyse systématique du code source

Outils

OWASP Top 10, Checklist personnalisée

Classification

Par type et gravité (CVSS)

📋 Liste des vulnérabilités

Tableau récapitulatif avec :

- # Numéro d'identification
- 🔴 Type de vulnérabilité
- ⚠️ Gravité (CVSS)
- 📎 Fichier concerné

🔍 Détail par vulnérabilité (1-2 pages)

- **Identification** : Description détaillée
- **Gravité** : Score CVSS justifié
- **Impact métier** : Conséquences opérationnelles
- **Preuve de concept** : Démonstration d'exploitation
- **Solution** : Code corrigé
- **Recommandation** : Mesures préventives

⚠️ Plan d'action priorisé

Criticité élevée

- Vulnérabilités CVSS 9-10
- Correctifs immédiats requis
- Tests de validation

Criticité moyenne

- Vulnérabilités CVSS 4-8.9
- Planification à court terme
- Suivi régulier

✅ Conclusion

Synthèse globale de l'audit, priorités d'action, recommandations à moyen/long terme

Grille d'évaluation - Audit de sécurité ShopCESI

Critères d'évaluation	Points
🔍 Complétude (nombre de vulnérabilités trouvées)	30
⚙️ Qualité technique (précision des descriptions, PoC pertinents)	25
🛠️ Solutions proposées (code corrigé, recommandations réalisables)	20
↳ Impact métier (compréhension des conséquences business)	10
📄 Forme professionnelle (structure, orthographe, mise en page)	10
↑➡️ Priorisation (plan d'action cohérent et justifié)	5

Barème de notation finale

Excellent

90-100 pts

A

Très bien

75-89 pts

B

Bien

60-74 pts

C

Passable

50-59 pts

D

Insuffisant

0-49 pts

E

Projet Final - Audit de sécurité ShopCESI

Conseils et ressources

Utilisez les checklists

 Les Schémas 1 et 2 sont vos meilleurs alliés pour l'audit

Répartissez le travail

 Un membre par fichier, travail en parallèle

Testez mentalement

 Comment exploiter chaque vulnérabilité ?

Relisez le rapport

 Orthographe, cohérence, clarté

Relisez vos notes

 Revisez les 6 modules précédents du cours

Gérez votre temps

 1h30 analyse • 1h45 rédaction • Respectez le timing

Vérifiez les corrections

 Chaque vulnérabilité DOIT avoir son code corrigé

Pas de copie

 Travail original obligatoire • Les rapports seront comparés



Outils autorisés : Notes du cours • Schémas 1 et 2 • Internet (recherches) • Documentation OWASP

Modalités de rendu :

 Format : PDF uniquement

 Nom de fichier : INFAL251_Audit_NomEquipe.pdf

 Email : [adresse du formateur]

 Objet : INFAL251 - Audit ShopCESI - NomEquipe

 Délai : 48h après la fin du cours



Bonne chance ! Montrez-nous vos compétences 



Veille sécurité et Continuité applicative

Maintenir la sécurité dans le temps



Module 6 - INFAL 251



45 min



2025

Pourquoi faire de la veille en sécurité ?



Nouvelles vulnérabilités chaque jour

Des CVE sont découvertes et publiées quotidiennement, rendant votre système vulnérable du jour au lendemain



Vos dépendances peuvent devenir vulnérables

Les librairies que vous utilisez (npm, pip, composer) peuvent révéler des failles critiques sans prévenir



Les attaquants s'adaptent en permanence

De nouvelles techniques d'attaque émergent constamment, nécessitant une veille active

40 000+ nouvelles CVE publiées en 2024

"La sécurité n'est pas un état, c'est un processus continu"

- Adage de la cybersécurité

CVE et CVSS : Comprendre les vulnérabilités



CVE (Common Vulnerabilities and Exposures)

IDENTIFIANT UNIQUE

CVE-2021-44228 // Format: CVE-YYYY-NNNNN

BASE DE DONNÉES PUBLIQUE

MITRE Corporation + National Vulnerability Database (NVD)

EXEMPLE CRITIQUE : LOG4SHELL

// Vulnérabilité Java Log4j //
découverte décembre 2021 //
Exploitation à distance



CVSS Score (0-10)

Échelle de gravité

0-3.9

4-6.9

7-8.9

9-10

FAIBLE

MOYEN

ÉLEVÉ

CRITIQUE



La note évolue avec le temps



Log4Shell = 10.0 CRITIQUE



Base / Temporel / Environnement



Où se tenir informé ?

🛡 Sources officielles

-  **CERT-FR / ANSSI**
Alertes nationales

-  **NVD**
Base de données CVE

-  **OWASP**
Standards sécurité

🤖 Outils automatisés

-  **GitHub Security**
Alertes sur repos

-  **Snyk / Dependabot**
Scan dépendances

-  **npm audit / pip-audit**
Audit de packages

👤 Veille humaine

-  **Newsletters**
TLDR Sec, Securibee

-  **Twitter/X**
@briankrebs, @troyhunt

-  **Reddit**
r/netsec

Module 6 : Veille sécurité et Continuité applicative

5/10 - Activité : Trouvez une CVE critique de 2024

➊ Objectif : Rechercher et analyser une vulnérabilité critique

Découvrir comment utiliser les bases de données de vulnérabilités pour évaluer les menaces

👥 Par équipes

Durée : 15 minutes

Par équipes de 3-4 personnes

Chrono

🏆 Présentation

2 minutes par équipe

Restitution flash

Flash talk

🔍 Étapes de la recherche

1) Accéder aux bases de données

- cve.mitre.org
- nvd.nist.gov

2) Filtrer les résultats

- Année : 2024
- CVSS Score : > 9.0
- Statut : Critical

📋 À analyser

- 🔴 Vulnérabilité : Type de faille (buffer overflow, injection, etc.)
- ⚡ Logiciel affecté : Nom, version, type (librairie, OS, application)
- ⚠ Impact : Confidentialité, intégrité, disponibilité
- 🛡 Solution : Patch, mise à jour, workaround

💡 Conseils

- 🔍 Chercher des CVEs récentes mais déjà documentées
- ⚠️ Éviter les CVEs encore en "RESERVED"
- 🔍 Rechercher par mots-clés : "Log4Shell", "RCE", "zero-day"

❗ Exemple de CVE à trouver

CVE-2024-XXXXX - Apache Struts

Score CVSS : 9.8 - Critique

Type : Remote Code Execution

Impact : Serveurs web compromis

Solution : Mise à jour vers version 2.5.33

PCA vs PRA - Plan de Continuité vs Plan de Reprise



PCA (Plan de Continuité d'Activité)

- ✓ **Objectif :** MAINTENIR les services pendant l'incident
- ⌚ **Quand :** Incident en cours
- ⚙️ **Stratégies :**
 - Mode dégradé
 - Redondance
 - Failover automatique
- 〓 **Exemple :** Serveur principal tombe → Bascule auto sur serveur backup



PRA (Plan de Reprise d'Activité)

- ⟳ **Objectif :** RESTAURER les services après un incident majeur
- ⟳ **Quand :** Après le désastre
- fdb **Stratégies :**
 - Sauvegardes
 - Snapshots
 - Procédures de restauration
- ⚙️ **Exemple :** Ransomware → Restauration depuis backup



RTO et RPO : Deux indicateurs clés

Temps de récupération et perte de données acceptables



Recovery Time Objective

Durée max d'interruption

Exemple E-commerce

RTO = 1 heure max



Recovery Point Objective

Perte de données max acceptable

Exemple Banque

RPO = 5 minutes max



Sur le business

À définir selon criticité



Système Normal



INCIDENT survient
Début de l'interruption



RPO
Données perdues



RTO
Temps de restauration



Service Rétabli
Fin de l'incident





PCA/PRA pour les applications

Sauvegardes

Règle 3-2-1 : 3 copies, 2 supports, 1 hors site

Chiffrement des backups obligatoire
Tests de restauration mensuels

Failover automatique

Détection de panne + bascule automatique

Health checks en continu
Bascule sans intervention humaine

Documentation

Procédures détaillées et accessibles

Runbooks : procédures étape par étape
Contacts d'urgence à jour

Redondance

Serveurs multiples avec load balancing

Base de données répliquée (master-slave)
Mode haute disponibilité

Snapshots

Images instantanées des systèmes

Containers Docker, VMs
Code versionné (Git)

Monitoring

Surveillance proactive et alertes

Alertes en temps réel
Logs centralisés (SIEM)

Module 6 : Veille sécurité et Continuité applicative

9/10 - Cas pratique : Ransomware sur ShopCESI

⚠ Scénario : Ransomware sur ShopCESI

Lundi 9h : Un ransomware a chiffré tous les serveurs de ShopCESI (site e-commerce)

🛡 Contexte

E-commerce actif (ventes en ligne)
Serveurs web, base de données, paiements
Toutes les données chiffrées et inaccessibles

⌚ Objectif

Définir RTO/RPO adaptés
Créer un plan d'action en 8 étapes
Prioriser les actions critiques



Veille sécurité et Continuité applicative

Checklist PCA/PRA

Sauvegardes automatiques

- ✓ Règle 3-2-1 : 3 copies, 2 supports, 1 hors site
- ✓ Chiffrement des backups obligatoire
- ✓ Tests de restauration mensuels

Tests de restauration

- ✓ Tests réguliers ($\geq 1x/\text{mois}$ minimum)
- ✓ Valider l'intégrité des données
- ✓ Mesurer le temps de restauration

Monitoring et alertes

- ✓ Alertes en temps réel
- ✓ Surveillance des composants critiques
- ✓ Logs centralisés (SIEM)

Documentation

- ✓ Runbooks à jour et accessibles
- ✓ Procédures étape par étape
- ✓ Contacts d'urgence actualisés

Redondance

- ✓ Composants critiques en redondance
- ✓ Failover automatique
- ✓ Load balancing configuré

RTO/RPO documentés

- ✓ Objectifs de temps et de données définis
- ✓ Mesures régulièrement validées
- ✓ Alignés avec les besoins métier

Simulation de crise

- ✓ Exercice annuel minimum
- ✓ Test des procédures de reprise
- ✓ Formation des équipes

Veille = processus continu (CVE/CVSS) • PCA = Maintenir • PRA = Restaurer
• RTO/RPO = indicateurs clés • Sauvegardes + Tests = fondation



Prochain module : PROJET FINAL (3h30)



Authentification vs Autorisation

Comprendre la différence fondamentale et leurs rôles dans la sécurité



Authentification

Qui es-tu ?

Processus de vérification de l'identité d'un utilisateur.

👤 Login / Mot de passe

⌚ MFA - Authentification à 2 facteurs

👉 Biométrie (empreinte, visage)

Analogie

C'est comme montrer ta carte d'employé au vestiaire pour prouver ton identité.



Autorisation

Que peux-tu faire ?

Processus de détermination des droits et permissions d'un utilisateur.

⚙️ Rôles: Admin, Modérateur, Utilisateur

☰ Permissions: Lire, Écrire, Supprimer

✓ Contrôle d'accès: RBAC, ABAC

Analogie

C'est comme l'accès aux étages de l'immeuble selon ton badge (étage 1 vs étage 10).



IDOR: Insecure Direct Object References

Accéder aux données des autres en modifiant simplement l'ID dans l'URL

✿ Définition

Une vulnérabilité qui permet d'accéder à des objets (données, fichiers, ressources) en manipulant les identifiants utilisés dans l'application.

Impact : Lecture, modification ou suppression de données appartenant à d'autres utilisateurs.

@ Exemple d'attaque

facture.php?
id=12345



facture.php?
id=12346



Facture
d'un
autre
client

⚠ Aucune vérification que la facture appartient bien à l'utilisateur connecté

Facebook (2019)



Bug IDOR découvert : Photos privées de 6.8 millions d'utilisateurs accessibles en modifiant un paramètre d'ID.

Conséquences : Exposition de photos personnelles à 1 500 applications tierces non autorisées.

🛡 Protection

- ✓ **Vérification** S'assurer que l'utilisateur connecté côté serveur est propriétaire de la ressource demandée.
- ✓ **Utiliser des UUID** Remplacer les IDs séquentiels par des identifiants uniques et imprévisibles.
- ✓ **Filtrer les requêtes** Implémenter des contrôles d'accès à chaque endpoint.

Code vulnérable IDOR

⚠️ Code PHP vulnérable

```
1 // ✖️ VULNÉRABLE: IDOR - Aucune vérification du propriétaire
2 $order_id = $_GET['id'];
3 $sql = "SELECT * FROM orders WHERE id = $order_id";
4 $result = mysqli_query($conn, $sql);
5 echo json_encode($result->fetch_assoc());
```



Problème : Aucune vérification que la commande appartient à l'utilisateur connecté

Risque : Lecture/édition des commandes d'autres clients en modifiant simplement l'ID

Correction IDOR : Vérification côté serveur

Toujours vérifier que l'utilisateur connecté est propriétaire de la ressource demandée

✗ AVANT (✗ Vulnérable)

IDOR

```
1 $order_id = $_GET['id'];
2
3 // Requête vulnérable - aucune
   vérification
4 $query = "SELECT * FROM orders WHERE
   id = $order_id";
5 $result = $db->query($query);
6
7 // N'importe qui peut lire n'importe
   quelle commande
8 echo json_encode($result-
   >fetch_assoc());
```

✓ APRÈS (✓ Sécurisé)

PROTEGÉ

```
1 $order_id = intval($_GET['id']); // Cast en entier
2 $user_id = $_SESSION['user_id']; // ID de l'utilisateur connecté
3
4 // Requête sécurisée avec vérification
   du propriétaire
5 $stmt = $db->prepare("SELECT * FROM
   orders WHERE id = ? AND user_id = ?");
6 $stmt->bind_param("ii", $order_id,
   $user_id);
7 $stmt->execute();
8
9 // Uniquement si la commande
   appartient à l'utilisateur
10 $result = $stmt->get_result();
```



⚠ Problème : Aucune vérification que la commande appartient à l'utilisateur connecté. Un attaquant peut lire toutes les commandes en modifiant l'ID.

🛡 Solution : Vérification systématique que la commande appartient bien à l'utilisateur connecté. Requêtes préparées pour éviter injection SQL.



RBAC vs ABAC

Comparaison des modèles d'accès - RBAC vs Attribute-Based Access Control

RBAC (Role-Based)

Basé sur les rôles

- ✓ Utilisateur → Rôle → Permissions
- ✓ Roles: Admin, Modérateur, Utilisateur
- ✓ Simple à implémenter et maintenir

Exemple : Un développeur a le rôle "User" et peut lire/modifier ses propres données.

ABAC (Attribute-Based)

Basé sur les attributs

- ✓ User + Resource + Context → Policy → Decision
- ✓ Attributs: Département, Localisation, Heure
- ✓ Plus flexible et granulaire

Exemple : Accès autorisé si: User.département=IT ET Resource.classification=public ET Time.ouvrable=true

 **Recommandation :** RBAC pour 80% des cas, ABAC pour contextes sensibles nécessitant un contrôle granulaire

Module 5 : Contrôle d'accès et Sécurité des APIs

7/20 - TP : IDOR Hunt (PortSwigger Academy)

➊ Objectif : Exploiter une vulnérabilité IDOR

Découvrir comment accéder aux données d'autres utilisateurs en manipulant les identifiants dans les URLs

🌐 Plateforme

 PortSwigger Web Security Academy

Lab gratuit d'apprentissage des vulnérabilités web

⌚ Format

Durée : 25 minutes

Mode : Speedrun



🛍️ Scénario : Site e-commerce

Contexte

Application web de vente en ligne avec gestion des commandes

Vulnérabilité

Accès aux commandes via URL : /orders/123 sans vérification

Objectif

Accéder à la commande d'un autre client et récupérer un identifiant sensible

👉 Étapes du TP

1. Repérer les endpoints sensibles - Identifier les URLs avec des paramètres d'ID
2. Tester la manipulation - Modifier l'ID dans l'URL (ex: 123 → 124)
3. Valider le POC - Confirmer l'accès aux données non autorisées
4. Documenter la preuve - Capture d'écran du résultat

💡 Conseils

- ✓ Regarder les URLs dans la barre d'adresse
- ✓ Tester différents types d'IDs (numériques, UUID)
- ✓ Vérifier plusieurs endpoints (/orders, /profile, /invoices)

⚠️ Attention

- ✗ Ne pas tester sur des sites réels sans autorisation
- ✗ Respecter les règles de PortSwigger Academy
- ✗ Documenter uniquement les preuves nécessaires

1.   Générer diagramme Auth vs Autho (Slide 2) [completed] (ID: 3)
2.   Générer diagramme IDOR (Slide 3) [completed] (ID: 4)
3.   Générer diagramme RBAC vs ABAC (Slide 6) [completed] (ID: 5)
4.   Générer diagramme API REST (Slide 12) [completed] (ID: 6)
5.   Générer diagramme JWT (Slide 17) [completed]

Exemple de matrice de droits FaceCESI

Principe du moindre privilège : Chaque rôle ne doit avoir que les droits nécessaires à ses fonctions

Action	Admin	Modérateur	Utilisateur	Visiteur
Publier un post	✓	✓	✓	✗
Modifier son propre post	✓	✓	✓	✗
Modifier post d'un autre	✓	✓	✗	✗
Supprimer son propre post	✓	✓	✓	✗
Supprimer post d'un autre	✓	✓	✗	✗
Bannir un utilisateur	✓	✓	✗	✗
Voir profils privés	✓	✓	✗	✗
Commenter un post	✓	✓	✓	✗
Liker un post	✓	✓	✓	✓
Gérer ses paramètres	✓	✓	✓	✗

 Autorisé  Interdit



Contrôle d'accès - Bonnes pratiques

🔑 Checklist Autorisation

🔴 Vérification côté serveur

- ✓ Contrôler les permissions **uniquement** côté serveur
- ✓ Jamais côté client (facilement contournable)
- ✓ Contrôles répétés à chaque requête

👤 Principe du moindre privilège

- ✓ Accès refusé par défaut
- ✓ Autoriser uniquement ce qui est nécessaire
- ✓ Réévaluer régulièrement les permissions

⟳ Vérification à chaque requête

- ✓ Pas seulement au login
- ✓ Session ≠ autorisations permanentes
- ✓ Re-vérifier les droits à chaque action

☒ Utiliser des UUIDs

- ✓ Remplacer les IDs séquentiels (1,2,3...)
- ✓ Empêcher la prédiction d'IDs
- ✓ Difficile à deviner, impossible à énumérer

⌚ Logger les accès

- ✓ Enregistrer les accès refusés
- ✓ Surveiller les tentatives anormales
- ✓ Alertes en cas de patterns suspects

👤 Tester avec différents comptes

- ✓ Compte utilisateur standard
- ✓ Compte administrateur
- ✓ Compte invité/non authentifié
- ✓ Tester chaque rôle possible



La sécurité du contrôle d'accès repose sur la vérification systématique et la décentralisation des contrôles





Sécurité des APIs

Protéger les interfaces applicatives



Module 5B - INFAL 251



1h15



2025



API REST en 2 minutes

Comprendre le fonctionnement des interfaces de programmation



💡 Analogie : L'API REST est comme un serveur de restaurant qui prend vos commandes (requêtes) et vous apporte les plats (données)



OWASP API Top 5

1

Broken Object Level Authorization (BOLA)

Accès non autorisé à des objets via manipulation d'ID (IDOR en API)

2

Broken Authentication

Tokens faibles, JWT mal configurés, gestion incorrecte des sessions

3

Excessive Data Exposure

L'API renvoie trop de données sensibles que le client n'a pas besoin

4

Lack of Rate Limiting

Aucune limitation du nombre de requêtes, vulnérable aux abus et DoS

5

Mass Assignment

Modification de champs non autorisés via des requêtes API mal filtrées

Excessive Data Exposure

```
</> GET /api/users/123
```

 EXPOSITION

```
1 {
2 "id": 123,
3 "name": "Alice",
4 "email": "alice@mail.com",
5 "password_hash": "5f4dcc3b5aa765d61d8327deb882cf99", //  Ne devrait pas être là !
6 "ssn": "123-45-6789", //  Numéro sécurité sociale !
7 "credit_card": "4532-****-****-1234", //  Carte bancaire !
8 }
```



Problème

 EXPOSITION

- 1 Le frontend n'affiche que `name` et `email`, mais TOUTES les données sensibles circulent dans la réponse API.
- 2 Attaquant qui intercepte le trafic = accès complet aux données.



Mass Assignment - Exemple d'attaque



Requête légitime

✓ SÉCURISÉ

</> Inscription normale

```
1 POST /api/users
2 {
3   "name": "Bob",
4   "email": "bob@mail.com"
5 }
```



Requête piratée

⚠ ATTAQUE

⚠ Injection de champs non autorisés

```
1 POST /api/users
2 {
3   "name": "Bob",
4   "email": "bob@mail.com",
5   "role": "admin", // ✗ L'attaquant s'autopromeut admin
6   "credits": 99999 // ✗ S'attribue des crédits
7 }
```


JWT - JSON Web Token

Structure et flux d'authentification sécurisée

⚙️ Structure du JWT

Format compact : **Header.Payload.Signature**

Header . Payload . Signature

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36P0k6yJV!adQssw5c
```

Encodé en Base64URL

⚠️ Vérifications essentielles : Signature, Audience, Expiration

⟳ Flux d'authentification



1) Client envoie identifiants



2) Serveur émet le JWT



3) Client envoie Bearer token

🛡 Sécurité

Le token est signé avec une clé secrète connue uniquement du serveur.

Chaque requête API inclut : `Authorization: Bearer <token>`



TP : Manipulation d'API avec Postman

API de test : JSONPlaceholder (jsonplaceholder.typicode.com)

GET

1) Récupérer tous les utilisateurs

Pratique : Obtenir la liste complète des utilisateurs



GET /users

GET

2) Récupérer un utilisateur

Analyse : Examiner le format des données d'un utilisateur



GET /users/1

POST

3) Créer un utilisateur

Création : Envoyer des données pour créer une nouvelle ressource



POST /users

PUT

4) Modifier un utilisateur

Mise à jour :Modifier les données d'un utilisateur existant



PUT /users/1

DELETE

5) Supprimer un utilisateur

Suppression : Retirer une ressource de la base



DELETE /users/1

GET

Bonus : Easter Egg

Challenge : Trouver l'endpoint caché et récupérer le flag secret



? /?_?_?



Durée : 30 minutes | Objectif : Maîtriser les requêtes HTTP de base | Score : 5 points + 1 bonus



Interface Postman

Guide visuel de l'outil de test API

Zones principales

GET

<https://jsonplaceholder.typicode.com/users/1>

Status: 200 OK

Time: 124ms

Size: 421 B



URL Bar: Entrez l'adresse de l'API et sélectionnez la méthode HTTP (GET, POST, PUT, DELETE)



Tabs: Body, Params, Headers, Authorization - Configurez vos requêtes



Response: Visualisez le corps, les headers et le statut de la réponse

Astuces pratiques



Collections: Organisez vos requêtes par projet/api dans des collections



Environments: Créez des variables pour changer facilement d'URL (dev, prod, test)



Runner: Testez automatiquement plusieurs requêtes avec des tests d'assertion



Scripts: Ajoutez des scripts pré/post requête pour automatiser des actions

Exercice : Code Review API

💡 Identifiez les vulnérabilités dans ces extraits de code API :



Extrait 1 - Node.js/Express (JavaScript)

VULNÉRABLE

```
app.get('/api/orders/:id', (req, res) => {
  const id = req.params.id;
  db.query('SELECT * FROM orders WHERE id = ?',
  [id], (err, result) => {
    res.json(result);
  });
});
```

// ✖ Pas de vérification du propriétaire de la commande



Extrait 2 - Python/Flask (Python)

VULNÉRABLE

```
@app.route('/api/users', methods=['POST'])
def create_user():
    data = request.json
    user = User(**data)
    db.session.add(user)
    return jsonify(user.to_dict())
```

// ✖ Mass Assignment : tous les champs peuvent être modifiés



Récapitulatif du module

- ▶ Vérifier systématiquement les droits d'accès (BOLA/IDOR)
- ▶ Implémenter des limites de requêtes (Rate Limiting)
- ▶ Ne jamais exposer plus de données que nécessaire
- ▶ Valider et filtrer les données d'entrée (sécurité par défaut)
- ▶ Utiliser des tokens JWT avec signatures fortes
- ▶ Tester l'API avec des outils comme Postman



Authentification et Gestion de Sessions

Protéger l'identité des utilisateurs



Module 4 - INFAL 251



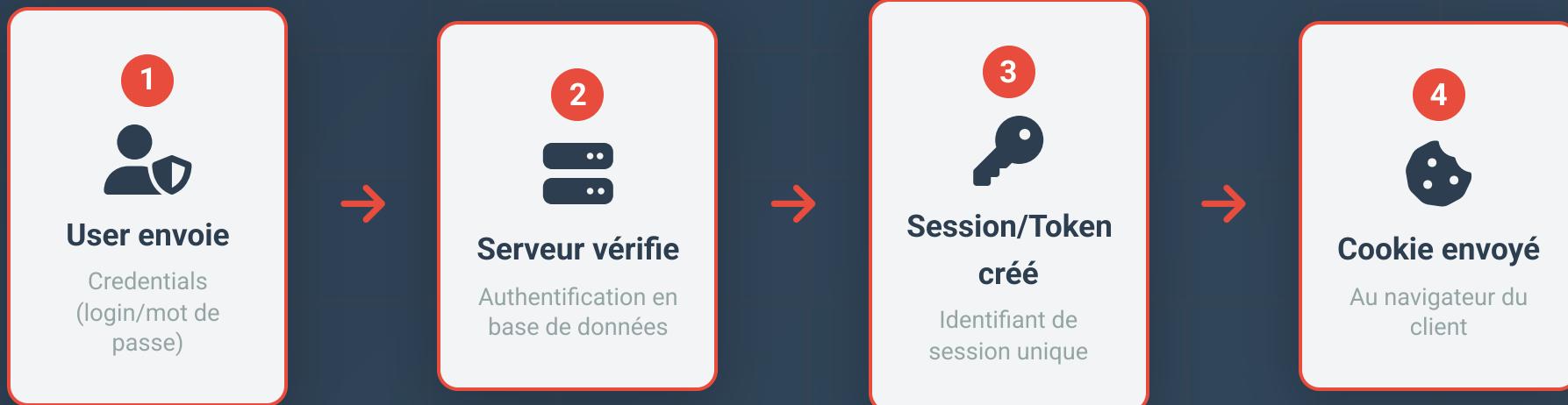
1h45



2025



Comment authentifier un utilisateur ?



Sessions Serveur

Cookie SESSIONID

- Persistance côté serveur
- Rotation automatique
- Révocation facile
- Compatible avec tous les navigateurs

Tokens JWT

JSON Web Token

- Stateless (aucun stockage serveur)
- Auto-portant (contient les données)
- Signature cryptographique
- Expiration contrôlée

Menaces sur l'authentification

Brute Force

-  Essais massifs de mots de passe par force brute automatisée

Credential Stuffing

-  Réutilisation massive de credentials volés sur d'autres sites

Session Hijacking

-  Vol du cookie de session pour usurper l'identité de l'utilisateur

Session Fixation

-  Forcer un ID de session connu avant l'authentification

Statistique clé

- 81% des brèches de sécurité impliquent des **mots de passe faibles ou volés**, rendant l'authentification le point de départ de la majorité des attaques





TP : Password Cracker Challenge

Outils autorisés : [CrackStation.net](#) |
[RainbowCrack](#)

Facile



Niveau 1 - Facile (5 hashes)

Mots de passe courants
password, 123456, admin, welcome,
qwerty

5a4!2f8#1b3c9d7e6

Moyen



Niveau 2 - Moyen (5 hashes)

Mots de passe classiques
Password123, Alice2024, Bonjour01,
Admin!2023, Secret123

a7b9!c3d4e5f6g8h2

Difficile



Niveau 3 - Difficile (5 hashes)

Mots de passe forts
Combinaisons complexes et uniques

z9x8!y7w6v5u4t3s2



Périmètre pédagogique uniquement : Cette activité a pour but de sensibiliser aux faiblesses des anciens systèmes de hachage. Les hashes fournis sont générés artificiellement pour l'apprentissage. **Ne jamais utiliser ces techniques sur des systèmes non autorisés.**



Les 15 hashes MD5 à craquer

#	Hash MD5	Points
1	# 5f4dcc3b5aa765d61d8327deb882cf99	PTS
2	# 25d55ad283aa400af464c76d713c07ad	PTS
3	# 21232f297a57a5a743894a0e4a801fc3	PTS
4	# e10adc3949ba59abbe56e057f20f883e	PTS
5	# 827ccb0eea8a706c4c34a16891f84e7b	PTS
6	# 8d6e!34f5b2a7c9e4f2d1a8b3c5e7f9a2	PTS
7	# 9f8e7d6c5b4a3!2d1e8f7c6b5a4d3e2f1	PTS
		PTS

Consigne : Renseignez vos trouvailles dans le tableau partagé

Pourquoi MD5 est cassable ?



MD5

DANGEREUX

Méthode de hachage rapide mais vulnérable aux attaques par brute force

~1 000 000 000

VS

0,001s

hashs/seconde

par hash

⚠ Vulnérabilité majeure

MD5 est trop rapide, permettant des attaques par force brute efficaces. Les tables Rainbow contiennent des hashs pré-calculés pour millions de mots de passe courants.

bcrypt

SÉCURISÉ

Algorithme de hachage lent conçu spécifiquement pour les mots de passe

~1 000

VS

0,001s

hashs/seconde

par hash

✓ Protection efficace

bcrypt est lent par design, rendant les attaques par force brute infaisables. Chaque hash prend du temps, ce qui décourage les attaquants.

💡 Recommendation sécurité

Pour la sécurité des mots de passe, Utilisez toujours bcrypt, Argon2 ou scrypt. Ces algorithmes sont conçus pour être lents et résistants aux attaques par force brute, contrairement à MD5 qui doit être évité.

Have I Been Pwned & Rainbow Tables

Comprendre la compromission des identifiants et les attaques pré-calculées



🔍 Vérifiez vos données

Site de référence créé par Troy Hunt pour vérifier si vos emails ou mots de passe ont été compromis dans une fuite de données majeure.

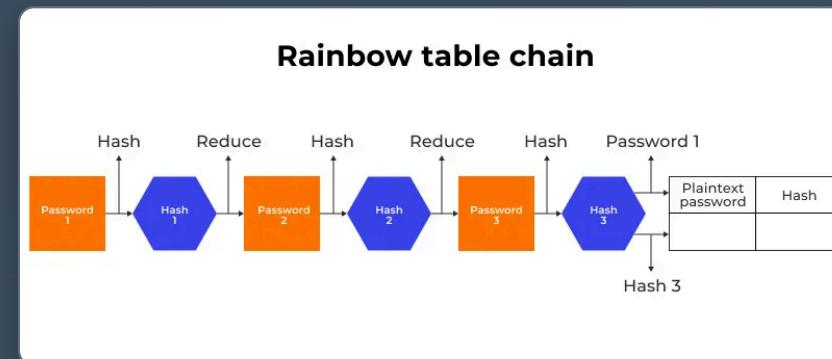
Action : Allez sur haveibeenpwned.com et testez votre email personnel.

12Mds+

Comptes compromis recensés dans la base

☒ Rainbow Tables : Le danger

Une attaque par Rainbow Table utilise des tables de hachage pré-calculées pour inverser des fonctions de hachage cryptographiques (comme MD5).



🛡 Contre-mesures

- ✓ **Salage** Ajouter une donnée aléatoire unique à (**Salting**) chaque mot de passe avant le hachage rend : les Rainbow Tables inefficaces.
- ✓ **Algorithmes** Utiliser bcrypt, Argon2 ou scrypt pour lents : augmenter le temps de calcul.
- ⚠ **Mots de passe longs** La complexité augmente exponentiellement la taille requise des tables.

Normes ANSSI vs NIST 2024



Critère	ANSSI (France)	NIST (USA)
大大小	✓ 12 caractères minimum (ou 8 avec complexité très forte)	✓ 8 caractères minimum, mais recommande de supporter jusqu'à 64
_COMPLEXITY	✓ Obligatoire si < 15 caractères (Maj + Min + Chiffres + Spéciaux)	⚠ Pas de règles absurdes (ex: 1 majuscule obligatoire). Favorise la longueur et les phrases secrètes.
EXPIRATION	✓ Pas d'expiration obligatoire si le mot de passe est fort et unique	🚫 Pas de rotation forcée sauf en cas de compromission suspectée
VALIDATION	✓ 2FA (MFA) fortement recommandé pour les accès critiques	🔍 Vérifier contre les listes de mots de passe compromis (ex: HIBP)

Stockage sécurisé des mots de passe

N'utilisez jamais de hachage simple (MD5, SHA1) pour les mots de passe

✗ AVANT (MD5)

✗ VULNÉRABLE

```
1 import hashlib
2
3 # Stockage (Mauvaise pratique)
4 pwd = "s3cr3t".encode()
5 hash = hashlib.md5(pwd).hexdigest()
6
7 # Vérification
8 if hashlib.md5(input).hexdigest() == hash:
9     print("Connecté")
```

✓ APRÈS (bcrypt)

✓ SÉCURISÉ

```
1 import bcrypt
2
3 # Stockage (Bonne pratique)
4 pwd = "s3cr3t".encode()
5 salt = bcrypt.gensalt()
6 hash = bcrypt.hashpw(pwd, salt)
7
8 # Vérification
9 if bcrypt.checkpw(input, hash):
10    print("Connecté")
```

⚠ Rapide à calculer = cassable instantanément avec des Rainbow Tables. Pas de "salt" (sel) par défaut.



🛡 Lent à calculer (Work Factor), Salt automatique unique par utilisateur. Résistant aux attaques par force brute.



Protections Complémentaires : Rate Limiting et 2FA



Rate Limiting

- ✓ Limite le nombre de tentatives de connexion (ex: 5 tentatives / IP)
- ✓ Blocage temporaire exponentiel après échecs successifs
- ✓ Protection contre Brute Force et Credential Stuffing

```
const limiter = rateLimit({  
  windowMs: 60 * 60 * 1000, // 1 heure  
  max: 5, // 5 tentatives max  
  message: "Trop de tentatives, réessayez plus tard"  
});  
app.use("/api/login", limiter);
```



Double Authentification (2FA)

- ✓ Authentification multi-facteurs : Ce que je sais (MDP) + Ce que je possède (Token)
- ✓ Méthodes : SMS, Email, Authenticator App (TOTP), Clé physique
- ✓ Bloque l'attaquant même si le mot de passe est compromis



Efficacité prouvée



Selon Microsoft et la CISA, l'activation de l'authentification multifacteur (MFA) bloque **99.9%** des attaques automatisées sur les comptes.



Sécuriser les cookies de session

</> PHP : Création d'un cookie sécurisé

PROTÉGÉ

```
1 setcookie('SESSIONID', $session_id, [
2   'httponly' => true, // Pas accessible en JavaScript
3   'secure' => true, // HTTPS uniquement
4   'samesite' => 'Strict' // Protection CSRF
5 ]);
```



HttpOnly

Empêche l'accès au cookie via JavaScript.
Protège contre le vol de session via XSS.

🛡 Protection : XSS



Secure

N'envoie le cookie que via HTTPS.
Empêche l'interception sur les réseaux
non sécurisés.

📡 Protection : MITM



SameSite

Limite l'envoi du cookie lors des requêtes
cross-site. Protège contre les attaques
CSRF.

⊛ Protection : CSRF



Navigateur



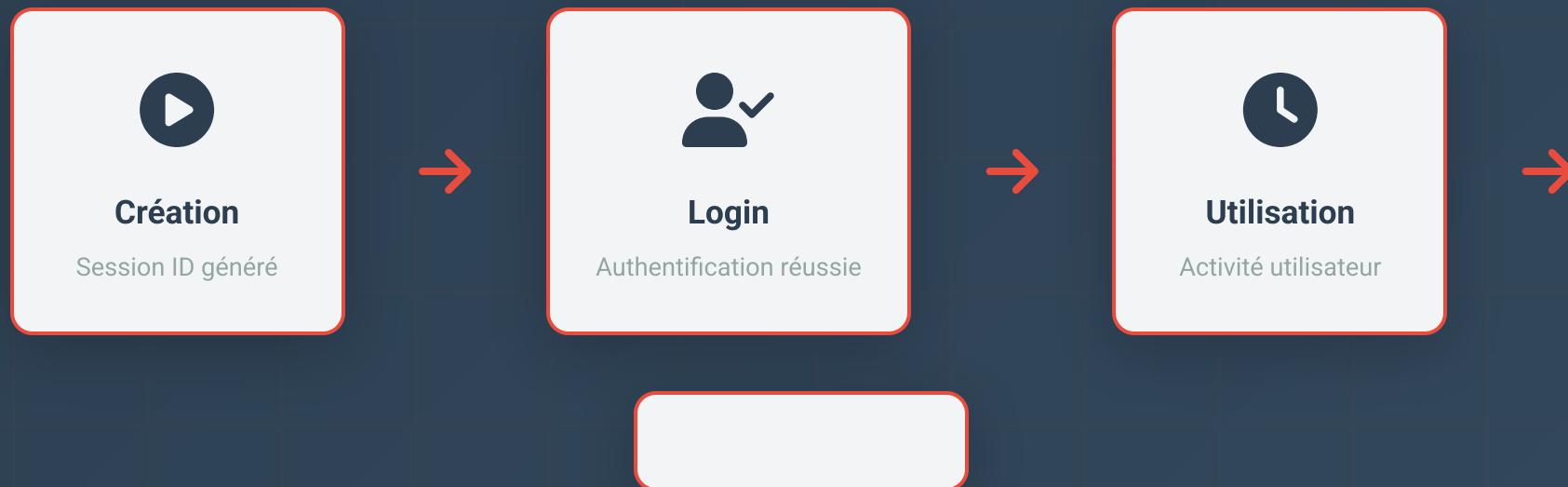
Cookie Protégé



Serveur



Cycle de vie des sessions



Checklist Authentification

Hachage avec bcrypt/Argon2

Utiliser des algorithmes lents et résistants au brute force

Rate limiting 5 tentatives max

Limiter les tentatives par IP/compte, blocage temporaire après échecs

2FA au moins pour admins

Authentification à deux facteurs via SMS/Email/App authentifier

Cookies HttpOnly + Secure + SameSite

Protéger les cookies de session contre le vol et le CSRF

Expiration/renouvellement sessions

Expiration après inactivité, renouvellement du Session ID après login

Vérifier contre listes de mots de passe compromis

Tester les mots de passe contre des bases de données de fuites

HTTPS partout OBLIGATOIRE

Chiffrement des communications entre client et serveur





Récapitulatif et clôture

🔑 Authentification et gestion des sessions

⚠️ Attaques principales

- ✓ **Brute Force** : essais massifs de mots de passe
- ✓ **Credential Stuffing** : réutilisation de fuites
- ✓ **Session Hijacking** : vol de cookie de session
- ✓ **Session Fixation** : ID de session imposé

✖️ MD5 = DANGEREUX

- ✓ Algorithmes rapides = vulnérables au brute force
- ✓ Tables rainbow = hashs pré-calculés
- ✓ 1 milliard de hashs/seconde possible

🔒 bcrypt/Argon2 = OBLIGATOIRES

- ✓ Algorithmes lents et coûteux en calcul
- ✓ Salt automatique intégré
- ✓ 1 000 hashs/seconde = résistance au brute force

📱 2FA bloque 99.9% des attaques

- ✓ Répondus par Microsoft/CISA
- ✓ Protection contre accès non autorisés
- ✓ SMS/Email/App (Authenticator TOTP)



Rate limiting essentiel : 5 tentatives maximum, blocage temporaire, journalisation



La sécurité de l'authentification = fondation de l'application



Fin du Jour 1 - À demain pour le Jour 2 !
Module 5 : XSS – Hacker le navigateur





SQL Injection

Le hack de la base de données



Module 2 - INFAL 251



1h30



2025



Mécanisme de base

✓ Requête SQL légitime

</> Requête normale

```
1 SELECT *  
2 FROM users  
3 WHERE username='alice'  
4 AND password='secret';
```

⚠️ Code PHP vulnérable

⚠️ Concaténation directe

 DANGEREUX 

```
1 $user = $_POST['user'];  
2 $pass = $_POST['pass'];  
3 .  
4 $sql = "SELECT * FROM users WHERE  
username='". $user . "' AND  
password='". $pass . "'";  
5 $res = mysqli_query($conn, $sql);
```





Bypass d'authentification

❶ Exploitation classique

L'attaquant saisit **admin' --** dans le champ username. Le **--** commente la fin de la requête SQL, permettant une connexion sans mot de passe.

👤 Username

admin' --



Requête SQL résultante

```
SELECT * FROM users  
WHERE username='admin' -- ' AND password=''
```



Connexion réussie sans mot de passe !

3 types de SQL Injection

1 In-Band (classique)

FACILE

Résultat immédiat dans la réponse HTTP. L'attaquant voit directement les données extraites.



Schéma
Attaquant → Formulaire → SQL injecté
→ BDD → Résultat affiché → Attaquant

2 Blind (inférentielle)

MOYEN

Pas de résultat direct. L'attaquant déduit des informations par des réponses vrai/faux.



Schéma

Attaquant → Questions booléennes → BDD → Réponses Oui/Non → Attaquant (déduction)

3 Out-of-Band (hors bande)

DIFFICILE

Exfiltration des données via un canal externe (DNS, HTTP, email) sans dépendre de la réponse HTTP.



Schéma

Attaquant → SQL injecté → BDD → Exfiltration DNS/HTTP → Serveur externe



UNION Attack : extraire des données

🔗 URL normale

```
/produit.php?id=5
```

⚠️ URL piratée

```
/produit.php?id=5 UNION SELECT id, email,  
password FROM users
```

</> Requête SQL résultante

EXFILTRATION

```
1 SELECT name, price FROM produits WHERE id=5  
2 UNION  
3 SELECT email, password FROM users;
```

⚠️ Impact : Exfiltration de données

L'attaquant combine les résultats de deux requêtes : affichage des produits + données sensibles des utilisateurs (emails, mots de passe). Cela permet de voler des informations confidentielles sans accès direct à la table users.



Impacts d'une SQL Injection

CRITIQUE



Vol total des données, modification de soldes/prix, suppression d tables (DROP TABLE)

Exemples : Accès à toutes les tables, modification de montants, destruction de la base

ÉLEVÉ



Contournement de l'authentification, escalade de privilèges, accès non autorisé

Exemples : Login sans mot de passe, passage admin, accès à des données sensibles

MOYEN



Déni de service, verrous de base de données, requêtes lourdes ralentissant le système

Exemples : Saturation du serveur, verrous SQL, crash de l'application

Conséquences pour l'entreprise



€ Pertes financières importantes
(amendes, remédiation, downtime)

⚡ Sanctions RGPD (jusqu'à 4% du CA global)

Atteinte à la réputation et perte de confiance client





TP : Mission SQL Hacker

PortSwigger Academy - Plateforme gratuite
en ligne

1 10 min 10 pts



Bypass Login

Contourner l'authentification
SQL Injection simple

2 15 min 20 pts



Données Cachées

Extraire des informations
non affichées

3 10 min 30 pts



UNION Attack

Manipuler les requêtes
pour exfiltrer des données



Classement par équipes

Points accumulés : Mission 1 → Mission 2 → Mission 3

1 2 3



Besoin d'aide ? Système d'indices

1 Niveau 1 - Léger

Pensez aux commentaires SQL

// C'est comme
commenter une ligne

2 Niveau 2 - Moyen

Utilisez -- ou # pour ignorer la suite

```
-- SELECT * FROM users  
WHERE username='admin'  
-- AND password= ''
```

3 Niveau 3 - Solution

Essayez admin' --

```
username: admin' --  
password: [vide]
```

Règle du jeu



Demandez à votre formateur l'indice approprié selon votre progression. N'attendez pas d'être complètement bloqué.



Comment se protéger ? (4 piliers)



Requêtes préparées

Prepared Statements : Séparer la structure SQL des données utilisateur pour empêcher l'injection de code malveillant



ORM

Object-Relational Mapping : Frameworks qui abstraient les interactions BDD et appliquent automatiquement des protections



Validation stricte

Validation de toutes les entrées utilisateur avec des règles strictes (whitelist > blacklist) avant traitement



Moindre privilège

Compte BDD avec permissions limitées (SELECT/INSERT/UPDATE uniquement), pas de DROP/ALTER, pas d'accès aux autres schémas

Solution #1 : Requêtes préparées

Les données sont séparées de la structure SQL

⚠ Avant (✗ vulnérable)

✗ VULNÉRABLE

```
1 $user = $_POST['user'];
2 $pass = $_POST['pass'];
3
4 $sql = "SELECT * FROM users WHERE
    username='" . $user . "' AND
    password='" . $pass . "'";
5
6 $res = mysqli_query($conn, $sql);
```

⚠ Concaténation directe = injection possible



🛡️ Après (✓ sécurisé)

✓ SÉCURISÉ

```
1 $stmt = $conn->prepare("SELECT * FROM
    users WHERE username=? AND
    password=?");
2 $stmt->bind_param("ss", $user, $pass);
3 $stmt->execute();
4
5 $res = $stmt->get_result();
6
7 // ? = placeholders, "ss" = string,
    string
8 // Données séparées, pas
    d'interprétation SQL
```

🛡️ Paramètres liés = pas d'injection SQL



Solution #2 : Utiliser un ORM

Abstraction et binding automatique des requêtes

⚠️ Avant (✗ concaténation) ✗ VULNÉRABLE

```
1 $id = $_GET['id'];
2
3 $sql = "SELECT * FROM products WHERE
4     id=" . $id;
5 $rows = mysqli_query($conn, $sql);
```

⚠️ Concaténation directe avec données utilisateur

🛡️ Après (✓ ORM) ✓ SÉCURISÉ

```
1 // Eloquent (PHP Laravel)
2 // Binding implicite, pas de
3 // concaténation
4
5 $product = Product::find((int)
6     request('id'));
7
8 // SQL généré automatiquement en toute
9 // sécurité
```

⌚ Binding automatique sans concaténation



⌚ Exemples d'ORM populaires :

SQLAlchemy

PHP Eloquent

Hibernate

JS TypeORM



Solutions #3 et #4 : Validation + Moindre privilège

Validation stricte

VALIDATION ID (CAST EN INT)

```
$id = (int) $_GET['id']; // OU : filtrage strict if (!is_numeric($id) || $id < 1) { exit('ID invalide'); }
```

VALIDATION USERNAME (REGEX)

```
$username = $_POST['user']; if (!preg_match('/^[\w]{3,20}$/', $username)) { exit('Username invalide'); }
```

- ✓ Whitelist > Blacklist
- ✓ Validation côté serveur obligatoire
- ✓ Types de données stricts

Moindre privilège

Comparaison des comptes

root

Tous droits

app_user

SELECT/INSERT/UPDATE

- ✗ Pas de DROP/ALTER
- ✗ Pas d'accès aux autres BDD
- ✓ Permissions minimales uniquement





Checklist SQL Injection



N'utiliser QUE des requêtes préparées ou un ORM



Valider toutes les entrées (whitelist > blacklist)



Compte BDD à moindre privilège
(SELECT/INSERT/UPDATE uniquement)



Chiffrement TLS entre app et BDD



WAF en complément (détection/filtrage)



Ne JAMAIS faire confiance aux données utilisateur



Règle d'or de la sécurité

Toutes les données utilisateur sont potentiellement malveillantes. Toujours valider et sécuriser avant traitement.



Exercice : Code Review Battle

🏆 Règles du jeu

- 5 extraits de code projetés
- Répondez « Vulnérable » ou « Sécurisé »
- Première équipe qui répond = 1 point
- Explication requise pour valider

⌚ Timing

- 30 secondes par extrait
- Pause discussion entre chaque
- Classement en temps réel
- Meilleure équipe gagne

</> Extrait 1 - Python Flask

Question 1/5

```
1 email = request.args.get('email')
2 pwd = request.args.get('pwd')
3 # Construction de la requête SQL
4 q = "SELECT * FROM users WHERE email=''' + email + ''' AND pwd=''' + pwd + '''"
5 cur.execute(q)
6 user = cur.fetchone()
7
8 if user:
```

❓ Réponse attendue :

VULNÉRABLE

SÉCURISÉ

Pourquoi ? Concaténation directe des variables dans la requête SQL sans validation ni paramétrisation.



Indice : Les variables email et pwd sont injectées telles quelles dans la requête SQL.



Correction Extrait 1

⚠️ Avant (✗ vulnérable)

✗ VULNÉRABLE

```
1 email = request.args.get('email')
2 pwd = request.args.get('pwd')
3
4 q = "SELECT * FROM users WHERE
    email=''" + email + "' AND pwd=''" + pwd
    + "'"
5
6 cur.execute(q)
7 user = cur.fetchone()
```

⚠️ Concaténation directe = injection possible

🛡️ Après (✓ préparée)

✓ SÉCURISÉ

```
1 q = "SELECT * FROM users WHERE email=?  
    AND pwd=?"
```

```
2 cur.execute(q, (email, pwd))
3
4
5 user = cur.fetchone()
```

```
6 # ? = placeholders, (email, pwd) = tuple de  
    paramètres
```

```
7  
# Paramètres liés = pas d'interprétation comme  
code SQL
```



🛡️ Paramètres liés = pas d'interprétation comme code SQL





Ce qu'on retient

🛡 Solutions de protection contre l'injection SQL

⚙️ Mécanisme

- ✓ L'entrée utilisateur devient du SQL exécutable
- ✓ Exploitation via concaténation de chaînes
- ✓ Faille classique mais toujours critique

📦 Types d'attaques

- ✓ **In-Band** : résultats immédiats (facile)
- ✓ **Blind** : déductions par vrai/faux (moyen)
- ✓ **Out-of-Band** : exfiltration via canal externe

⚠️ Impacts critiques

- ✓ Vol/modification/suppression de données
- ✓ Violation RGPD et sanctions
- ✓ Pertes financières et réputation

🔒 Protection essentielle

- ✓ TOUJOURS utiliser des prepared statements
- ✓ Séparer structure SQL et données
- ✓ ORM ou requêtes paramétrées



TOUJOURS utiliser des prepared statements pour sécuriser vos applications



Prochain module : XSS – Hacker le navigateur
Exploitation des failles côté client



Sécurité des Systèmes Applicatifs

Module 1 : Introduction et Panorama des Architectures



INFAL 251 - CESI



14h sur 2 jours



2025





Pourquoi la sécurité applicative ?

EFX

Equifax

2017

143 millions d'utilisateurs affectés
~700 millions \$ d'amende et de coûts

Impact : Fuite de données sensibles (SSN,
dates de naissance, adresses)

Vulnérabilité : Apache Struts CVE-2017-5638 (non
patchée)

BA

British Airways

2018

500 000 clients concernés
183 millions £ d'amende RGPD

Impact : Vol de données bancaires via
paiement en ligne

Vulnérabilité : XSS / Magecart (skimming de cartes)



4,45M\$

Coût moyen d'une cyberattaque



5 milliards

Records exposés en 2024



90%

Attaques ciblent des applications web

Notre périmètre : Sécurité APPLICATIVE



ON PARLE DE

Code source

Données et BDD

Authentification

APIs

Sessions



HORS PÉRIMÈTRE

Firewall

VPN

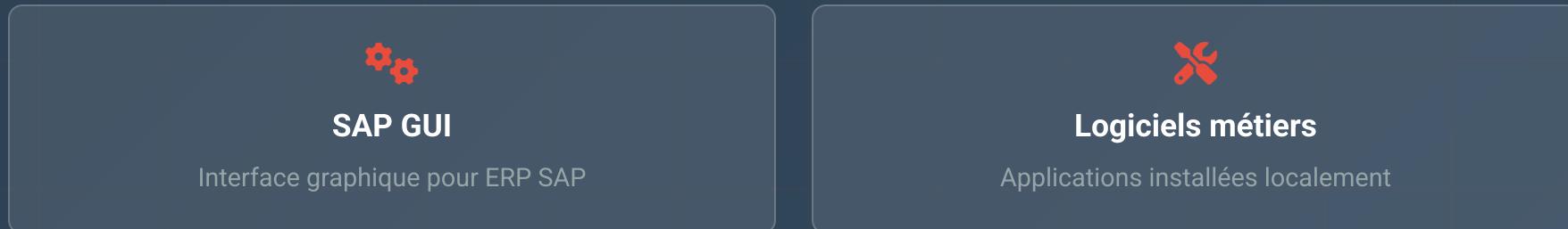
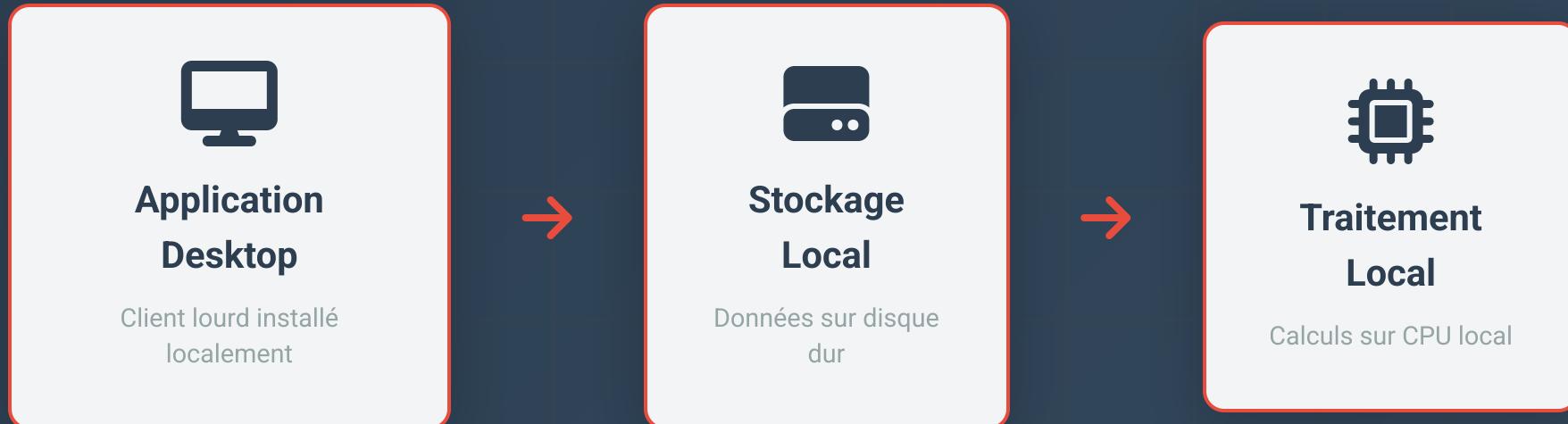
IDS/IPS

Segmentation réseau

Référence principale : **OWASP Top 10** - Les 10 risques de sécurité applicative les plus critiques



Architecture 1 : Applications Desktop



Exécution locale

Autonome

Pas de dépendance réseau

Risques spécifiques Desktop

Reverse engineering

Décompilation du code et analyse de la logique métier pour découvrir des vulnérabilités cachées.

Stockage local non sécurisé

Données sensibles stockées en clair sur le disque sans chiffrement, accessibles à tous les utilisateurs.

Credentials hardcodés

Mots de passe et clés d'API directement intégrés dans le code source de l'application.

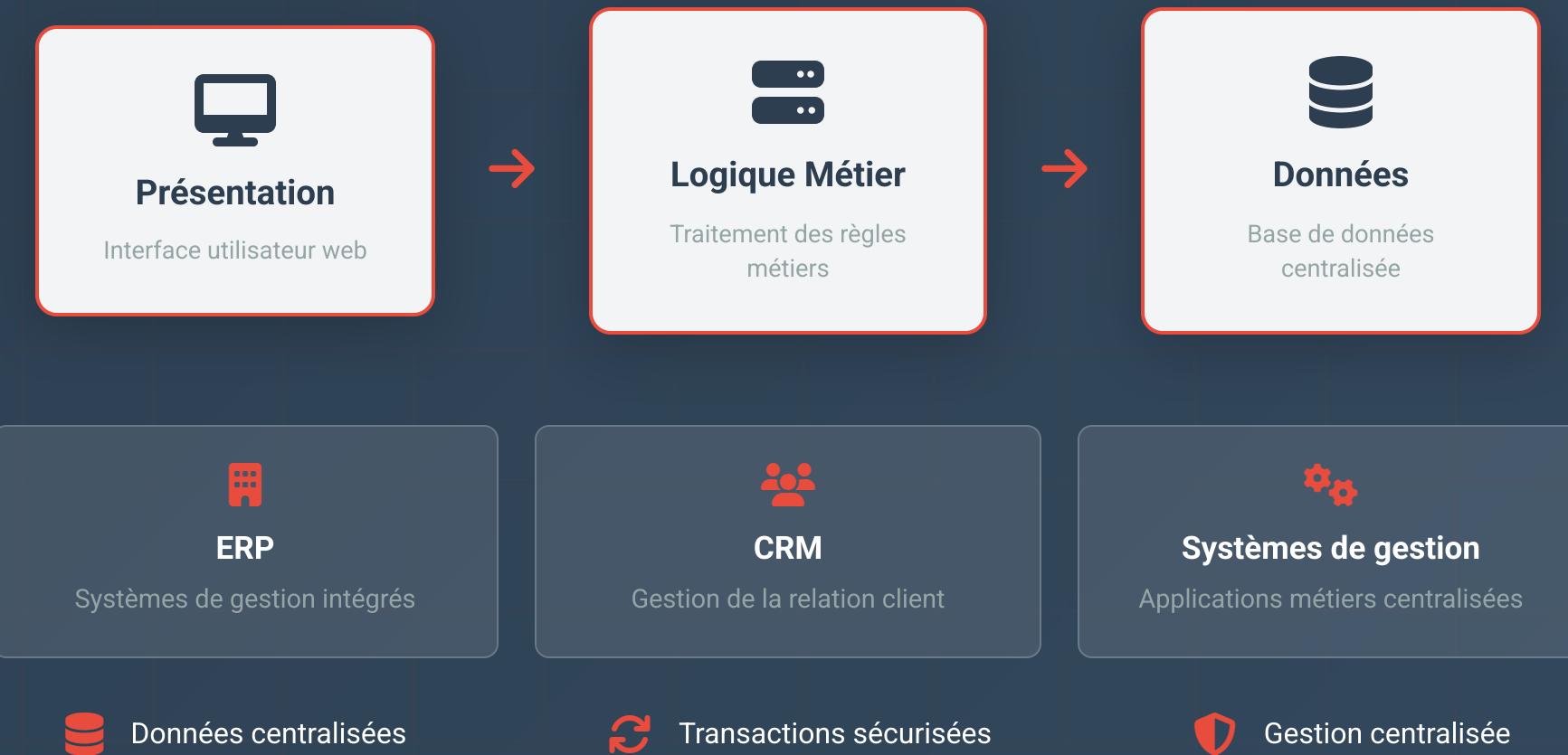
Exemple C# vulnérable

```
1 public class Auth {  
2     private const string MASTER_PWD = "S3cr3t!"; // ❌ en dur  
3  
4     public bool Login(string u, string p) {  
5         return p == MASTER_PWD; // ❌ contournable  
6     }  
7 }  
8 // Chiffrer secrets + coffre-fort applicatif
```

 VULNÉRABLE



Architecture 2 : Applications avec Base de Données



Risques spécifiques BDD



RISQUE #1 : SQL INJECTION

Injection de code SQL malicieux via des entrées utilisateur non validées, permettant l'accès non autorisé aux données.



Connexions non chiffrées

Communication entre l'application et la base de données en clair, exposées à l'interception.



Privilèges excessifs

Comptes de base de données avec des droits d'administration au lieu du principe du moindre privilège.



Exemple PHP vulnérable

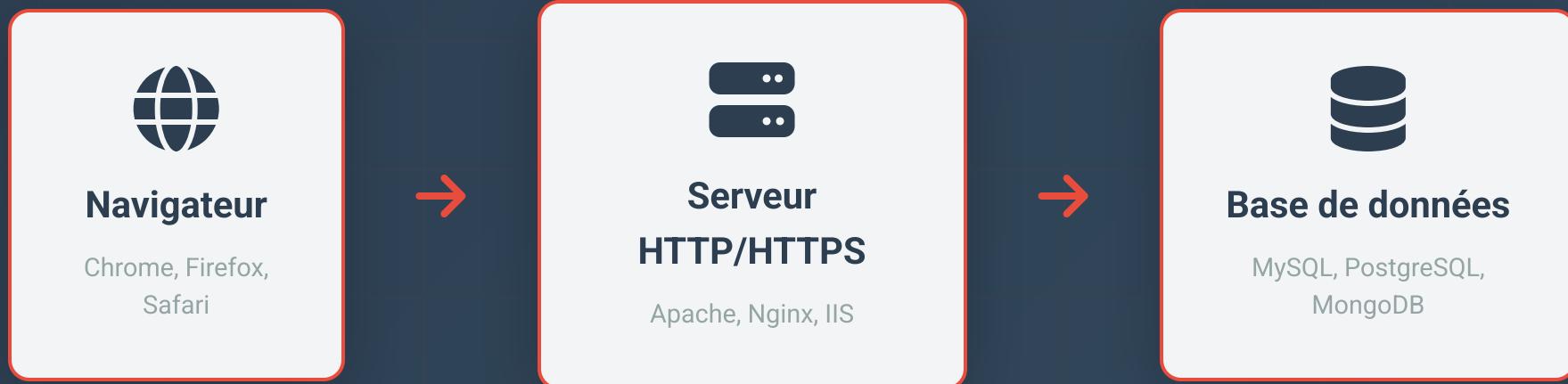
```
1 $user = $_GET['user'];
2 $pass = $_GET['pass'];
3
4 $sql = "SELECT * FROM users WHERE login='$user' AND password='$pass'";
5 $res = mysqli_query($conn, $sql);
6 if (mysqli_num_rows($res)>0) echo "OK";
7
8 // Payload : admin' OR '1'='1' --
```



VULNÉRABLE



Architecture 3 : Applications Web



Accessible partout



Surface d'attaque maximale



Mises à jour centralisées

Risques spécifiques Web

#1 XSS (Cross-Site Scripting)

Injection de scripts malicieux dans les pages web vues par les autres utilisateurs

#2 SQL Injection

Injection de requêtes SQL malveillantes via des entrées utilisateur non validées

#3 CSRF (Cross-Site Request Forgery)

Force un utilisateur authentifié à effectuer des actions non désirées sur une application web

#4 Authentification cassée

Failles dans les mécanismes de login, sessions, gestion des mots de passe

#5 Session hijacking

Vol de sessions utilisateur via cookies ou tokens de session non sécurisés



Exposition publique maximale

Applications web accessibles depuis Internet = cibles prioritaires des attaquants. Surface d'attaque la plus exposée.



Architecture 4 : Applications IA/Machine Learning



Apprentissage automatique

Prédictions temps réel

Amélioration continue

Risques spécifiques IA/ML

Data Poisoning

Corruption des données d'entraînement pour influencer négativement le comportement du modèle

Adversarial Attacks

Perturbations ciblées sur les entrées pour tromper le modèle et obtenir des prédictions erronées

Prompt Injection

Injection de commandes malicieuses dans les prompts pour manipuler les LLMs et contourner les restrictions

Model Stealing

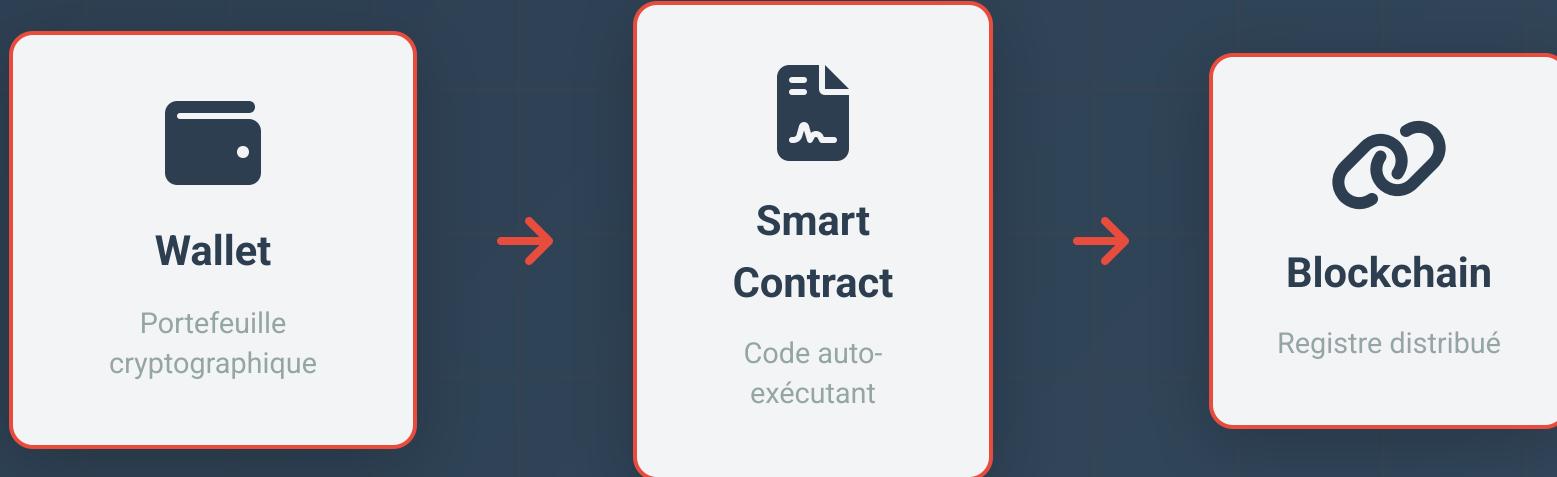
Extraction du modèle via l'API d'inférence pour créer une copie ou obtenir des informations propriétaires

Modèles IA = cibles de haute valeur

Les modèles ML deviennent des actifs critiques. Leur compromission peut avoir des impacts majeurs sur les décisions automatisées.



Architecture 5 : Applications Blockchain



Décentralisé

Imuable

Transparent



Risques spécifiques Blockchain

⌚ Reentrancy Attack

The DAO hack 2016 - 60M\$ volés.
Exploite les appels externes avant la mise à jour de l'état.

-Calcul Integer Overflow/Underflow

Dépassement de capacité des variables numériques, entraînant des calculs erronés.

🔑 Gestion des clés privées

Perte ou vol de clés privées = perte définitive des fonds. Pas de récupération possible.

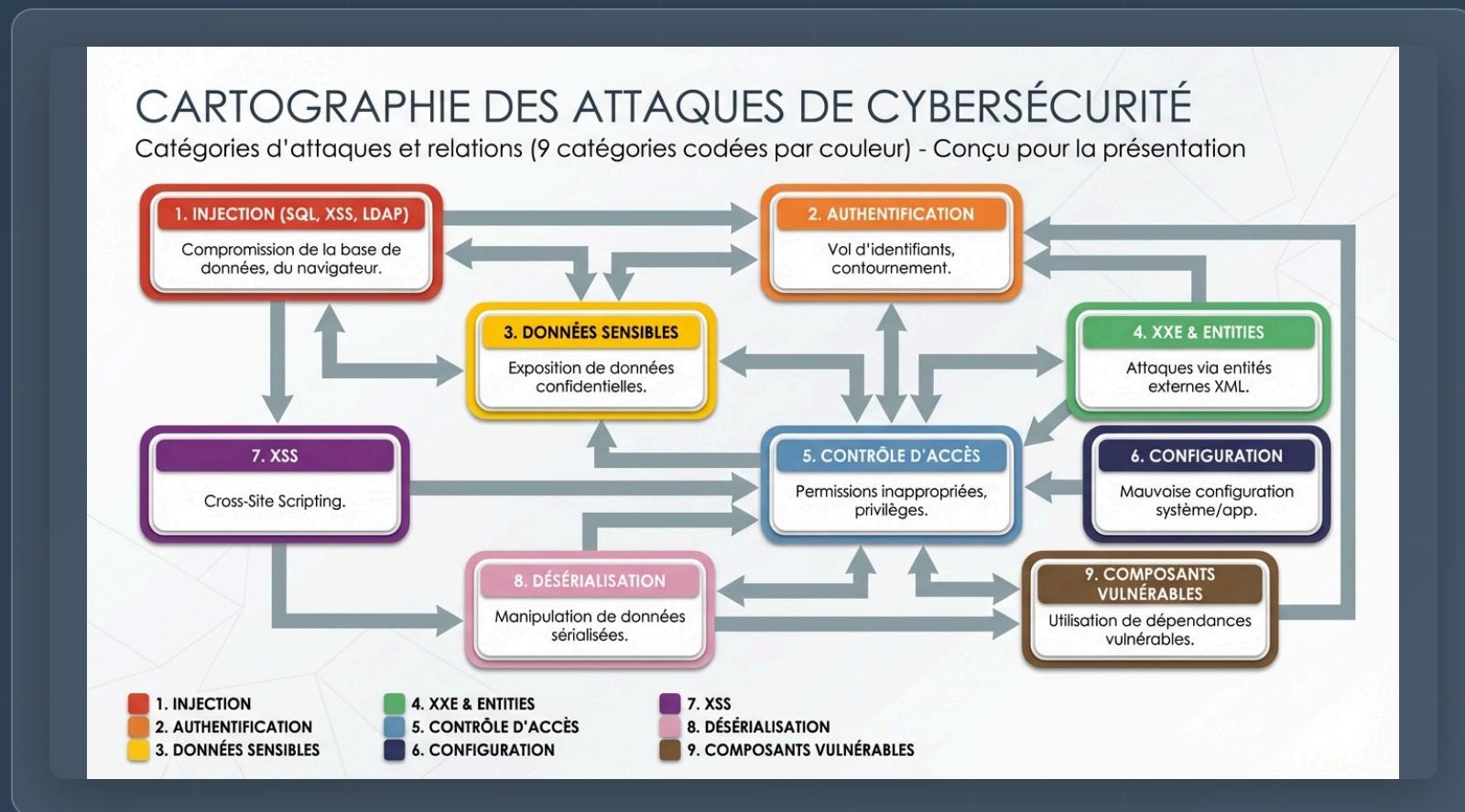
⚠ Exemple Solidity vulnérable

VULNÉRABLE

```
1 pragma solidity ^0.8.0;
2 contract VulnerableVault {
3     mapping(address => uint) public balances;
4
5     function deposit() public payable {
6         balances[msg.sender] += msg.value;
7     }
8
9     function withdraw(uint amount) public {
10        require(balances[msg.sender] >= amount);
11        (bool ok,) = msg.sender.call{value:amount}(""); // ✖ appel externe
12        require(ok);
13        balances[msg.sender] -= amount; // ✖ maj après l'appel
14    }
15 }
16 // ReentrancyGuard pattern ou CEI recommandé
```



Schéma 1 : Cartographie complète des attaques



Référence
Palette conforme : #2C3E50 / #E74C3C / #95A5A6

Ce schéma sera votre
référence tout au long
du cours
9 catégories
d'attaques
regroupées par type
et vecteur
d'exploitation

○ #2C3E50 ● #E74C3C ○ #95A5A6





Schéma 2 : Sécurité par architecture



Points durs par architecture

Desktop: Isolation système, mais nécessite chiffrement
BDD: Authentification, mais privilèges à contrôler
Web: HTTPS, mais WAF et CSP indispensables

Matrice à utiliser pour vos audits
Comparaison entre sécurité native et mesures à implémenter
Chaque architecture nécessite des contrôles spécifiques



#2C3E50



#E74C3C



#95A5A6





Récapitulatif : 5 architectures, 5 risques majeurs

Architecture	Risque #1	Bonne pratique clé
Desktop	Reverse engineering	Obfuscation + secret vault
BDD	SQL Injection	Requêtes paramétrées + moindre privilège
Web	</> XSS	Encodage sortie + CSP
IA/ML	Data poisoning	Contrôle provenance + validation
Blockchain	Réentrance	CEI pattern + ReentrancyGuard



Chaque architecture nécessite une approche de sécurité spécifique adaptée à ses contraintes



Activité : Identifiez l'architecture !



Par équipes, identifiez l'architecture de chaque application

5 minutes

1

NETFLIX

Netflix

2

SAP

SAP ERP

3



ChatGPT

4



Uniswap

5



Photoshop

6



Salesforce

7



TensorFlow

8



MetaMask

9



Gmail

10



Oracle Database



Indice : Pensez à la localisation du traitement, la source des données, et l'interface utilisateur



Votre première exploitation : HackThisSite.org



Félicitations ! **Vous venez d'exploiter votre première vulnérabilité !**

C'est le début de votre parcours en cybersécurité



Rappel : Cette plateforme est légale et éthique - utilisez vos compétences de manière responsable



Récapitulatif Module 1

A Enjeux critiques

- ✓ Cas réels : Equifax (143M victimes)
- ✓ British Airways (183M£ d'amende)
- ✓ Coût moyen : 4,45M\$ par attaque

B 5 architectures

- ✓ Desktop : Application locale
- ✓ BDD : Systèmes centraux
- ✓ Web : Applications SaaS
- ✓ IA/ML : Intelligence artificielle
- ✓ Blockchain : Décentralisation

C Risques spécifiques

- ✓ Chaque architecture a ses vulnérabilités
- ✓ Surface d'attaque unique
- ✓ Protection adaptée nécessaire

D Référence

- ✓ OWASP Top 10
- ✓ Guide de bonnes pratiques
- ✓ Standard industriel



Module 2 : SQL Injection en profondeur
Approfondissement sur le risque #1 des BDD

