TD Application Association sportive AS Olympique Saint-Rémy

Public : BTS SIO SLAM 2e année.
Langage : PHP (sauf XXE en XML).

Durée totale : 9 heures

Répartition à titre indicative

- Préparation & consignes (intro + déploiement labo) : 1h
- Envoi de fichier : 1.5h
- Transmission URL : 1h
- Injections SQL : 1.5h
- XSS (reflétée + stockée) : 1.5h
- Piratage de session : 1h
- CSRF : 1h
- XXE (pour aller plus loin) + corrections : 0.5h

Table des matières

1. Rappel du contexte du TD

Vous intervenez pour **l'association sportive AS Olympique Saint-Rémy**, qui gère :

- Les membres,
- Les inscriptions aux activités,
- Les réservations,
- Les résultats sportifs,
- L'upload de photos de compétition.

L'application actuelle comporte plusieurs **failles de sécurité majeures**.
Votre rôle : **les comprendre, les exploiter (dans un cadre pédagogique)**, puis **les corriger**.

---

2. Organisation du TD

Vous travaillerez sur **7 parties** :

1. Envoi de fichier (upload)
2. Transmission des données dans l'URL
3. Injection SQL
4. XSS (reflétée + stockée)
5. Piratage de session
6. CSRF
7. XXE (pour aller plus loin – XML)

Pour chaque faille :

- Tester le code vulnérable fourni et constater la faille
- L'expliquer et la corriger

⚠ 3. Règles de sécurité

Même en environnement local :

- Ne jamais pratiquer ces manipulations sur des systèmes réels.
- Ne jamais tester ces attaques sur un réseau public.
- Ne jamais utiliser de scripts dangereux en dehors du labo.
- Débrancher toute synchronisation vers le cloud.

🎓 4. Compétences travaillées

- Compréhension et détection des failles OWASP
- Analyse d'un code PHP vulnérable
- Mise en œuvre de contre-mesures professionnelles
- Standardisation via PDO, validation, encodage, tokens CSRF
- Développement sécurisé

🕒 5. Préparation & Consignes – 1h

🎯 **Objectifs de cette phase**

Avant de débuter le travail pratique sur les failles OWASP, vous devez mettre en place **l'environnement de travail** et comprendre **le contexte général** du TD.

Cette étape prépare le terrain pour les manipulations ci-dessous.

🔧 1. Mise en place du laboratoire

Chaque étudiant doit disposer :

✓ **D'un serveur local (au choix)**

- **WAMP / XAMPP** sous Windows
- **MAMP** sous macOS, Windows
- **LAMPP (Apache + PHP + MariaDB)** sous Linux

Le serveur doit comprendre :

- **PHP 7.4 ou supérieur**
- **MySQL/MariaDB**
- **phpMyAdmin**

✓ D'un projet local pour PHP

Créer un dossier → as_olympique/

Créer deux sous-dossiers → /src et /uploads

Copier le fichier 21976203-f3c2-4002-b1a7-f7a28952c8d9.pptx dans le dossier /upload/

✓ Créer l'application AS Olympique Saint-Rémy en PHP

Structure de votre application (racine du projet as_olympique/)

```
as_olympique/
├– src/
|   ├– config.php
|   ├– init.php
|   ├– functions.php
|   ├– index.php
|   ├– vuln/
|   |   ├– upload_vuln.php
|   |   ├– bonjour_vuln.php
|   |   ├– connexion_vuln.php
|   |   ├– commentaire_vuln.php
|   |   ├– auth_vuln.php
|   |   ├– del_vuln.php
|   |   └– parse_vuln_xml.php
|   ├– secure/
|   |   ├– upload_secure.php  (voir annexe document 3)
|   |   ├– bonjour_secure.php
|   |   ├– connexion_secure.php
|   |   ├– commentaire_secure.php
|   |   ├– auth_secure.php
|   |   ├– del_secure.php
|   |   └– parse_secure_xml.php
|   └– templates/
|       └– header.php (simple)
└– uploads/   (idéalement hors webroot)
```

✓ Recopier le contenu de chaque fichier dans le bon répertoire et le bon nom de fichier

src/config.php

```
<?php
// config.php - paramètres de connexion et config globale
return (object)[
  'db' => (object)[
    'host' => '127.0.0.1',
    'dbname' => 'as_olympique_db',
    'user' => 'as_user',
    'pass' => 'as_pwd',
    'charset' => 'utf8mb4'
  ],
  // chemin vers uploads (préférer hors webroot)
  'upload_dir' => __DIR__ . '/../uploads/',    // stockage hors du dossier public pour
                                               // empêcher un accès direct
  'max_upload_size' => 2 * 1024 * 1024 // 2MB   // taille maximal supportée
];
```

---

src/init.php

```
<?php
// init.php - démarre session et PDO
$config = require __DIR__ . '/config.php';

// session cookie params
session_set_cookie_params([
  'lifetime' => 0,
```

```
 'path' => '/',
 'domain' => '', // mettre le domaine si besoin
 'secure' => isset($_SERVER['HTTPS']) && $_SERVER['HTTPS'] !== 'off',  // secure => true : cookie transmis uniquement sur HTTPS
 'httponly' => true,    // httponly => true : JavaScript ne peut pas lire cookie
 'samesite' => 'Lax'    // samesite réduit la CSRF
]);
session_start();  // s'il faut associer un upload à un utilisateur, garder la session de l'utilisateur


// PDO  en mode exceptions try/catch est meilleur que reporting
try {
 $dsn = "mysql:host={$config->db->host};dbname={$config->db->dbname};charset={$config->db->charset}";
 $pdo = new PDO($dsn, $config->db->user, $config->db->pass, [
//
   PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
   PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
   PDO::ATTR_EMULATE_PREPARES => false
// PDO::ATTR_EMULATE_PREPARES => false force l'utilisation de vraies requêtes préparées
 ]);
} catch (Exception $e) {
 http_response_code(500);
 echo "Erreur de connexion DB: " . htmlspecialchars($e->getMessage());
// htmlspecialchars encode <,>,",',&.
 exit;
}
```

---

src/functions.php

```
<?php

// functions.php - utilitaires réutilisables
```

```
function e($s) {
  return htmlspecialchars($s ?? '', ENT_QUOTES, 'UTF-8');  }

// htmlspecialchars encode <,>,",',&.
```

```
function generate_csrf_token() {
  if (empty($_SESSION['csrf_token'])) {
    $_SESSION['csrf_token'] = bin2hex(random_bytes(32));

// bin2hex(random_bytes(32)) convertit une chaîne encodée en hexadécimal vers du binaire. Ici cela évite le token imprévisible

  }
  return $_SESSION['csrf_token'];
}
```

```
function verify_csrf_token($token) {
  return hash_equals($_SESSION['csrf_token'] ?? '', $token ?? '');

// hash_equals() évite le timing attacks ou attaque temporelle en vérifiant si deux chaînes de caractères sont égales sans divulguer d'informations sur le contenu

}
```

```
function secure_filename($name) {
  // conserve uniquement lettres chiffres et points, remplace espaces
  $name = preg_replace('/[^A-Za-z0-9\.\-_]/', '_', $name);
  return $name;
```

```
}
```

---

src/templates/header.php

```
<?php
// header simple pour navigation
?>
<!doctype html>
<html lang="fr">
<head>
 <meta charset="utf-8">
 <title>AS Olympique - TD sécurité</title>
 <meta name="viewport" content="width=device-width,initial-scale=1">
 <style>body{font-family:Arial,Helvetica,sans-serif;max-width:900px;margin:20px
auto;padding:10px}</style>
</head>
<body>
<h1>AS Olympique - TD Sécurité (vuln vs secure)</h1>
<nav>
 <a href="index.php">Accueil</a> |
 <a href="vuln/upload_vuln.php">Upload vuln</a> |
</nav>
<hr>
```

---

src/index.php

```
<?php
require __DIR__ . '/init.php';
require __DIR__ . '/functions.php';
include __DIR__ . '/templates/header.php';
```

```
?>
<h2>Menu TD</h2>
<ul>
 <li><a href="vuln/upload_vuln.php">Upload (vuln)</a>
 <li><a href="vuln/bonjour_vuln.php">Transmission GET (vuln)</a>
 <li><a href="vuln/connexion_vuln.php">Connexion (vuln)</a>
 <li><a href="vuln/commentaire_vuln.php">Commentaire (XSS vuln)</a
 <li><a href="vuln/auth_vuln.php">Auth session (vuln)</a>
 <li><a href="vuln/del_vuln.php">Suppression GET (CSRF vuln)</a
 <li><a href="vuln/parse_vuln_xml.php">XXE vuln (XML)</a> -
</ul>
<?php echo '<p>Fichier source PPTX utilisé : /upload/21976203-f3c2-4002-b1a7-f7a28952c8d9.pptx</p>'; ?>
// fichier de support de cours renommé pour la partie sécurité
</body>
</html>
```

Dossier vuln/ — versions vulnérables à recopier

src/vuln/upload_vuln.php

```
<?php
// upload_vuln.php - très simplifié et dangereux, pour exercice
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
 if (isset($_FILES['photo'])) {  // pour s'assurer qu'un fichier a été soumis
   move_uploaded_file($_FILES['photo']['tmp_name'], __DIR__ . '/../../uploads/' . $_FILES['photo']['name']);
// move_uploaded_file() — sécurité native pour uploads temporaires

   echo "Upload OK: " . $_FILES['photo']['name'];
```

```
 } else {
   echo "Aucun fichier";
 }
 exit;
}
?>

<form method="post" enctype="multipart/form-data">
 <input type="file" name="photo">
 <button>Envoyer (vuln)</button>
</form>
```

---

src/vuln/bonjour_vuln.php

```
<?php
// bonjour_vuln.php
$nom = $_GET['nom'] ?? '';
$prenom = $_GET['prenom'] ?? '';
$repeter = $_GET['repeter'] ?? 1;
for ($i=0;$i<$repeter;$i++){
 echo "Bonjour $nom $prenom<br>";
}
```

---

src/vuln/connexion_vuln.php

```
<?php
require __DIR__ . '/../init.php';
// vuln : concaténation SQL
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
 $username = $_POST['username'] ?? '';
```

```
 $password = $_POST['password'] ?? '';
 $sql = "SELECT * FROM users WHERE username = '$username' AND password_hash = '$password'";       // hache le mot de passe
 $res = $pdo->query($sql);
 $user = $res->fetch();
 if ($user) {
   echo "Connecté en tant que " . htmlspecialchars($user['username']);
// htmlspecialchars encode <,>,",',&.
 } else {
   echo "Erreur login";
 }
 exit;
}
?>
<form method="post">
 <input name="username" placeholder="username"><br>
 <input name="password" placeholder="password"><br>
 <button>Connexion vuln</button>
</form>
```

---

src/vuln/commentaire_vuln.php

```
<?php
require __DIR__ . '/../init.php';
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
  $membre_id = $_POST['membre_id'] ?? 1;
  $contenu = $_POST['contenu'] ?? '';
  // insertion vulnérable (no sanit)
```

```
  $pdo->exec("INSERT INTO commentaires (membre_id, contenu) VALUES
($membre_id, '" . str_replace("'", "''", $contenu) . "')");
  echo "Commentaire posté (vuln)";
  exit;
}
// affichage vulnérable
$stmt = $pdo->query("SELECT c.* FROM commentaires c ORDER BY date_post DESC
LIMIT 20");
$rows = $stmt->fetchAll();
?>
<form method="post">
  <textarea name="contenu"></textarea><br>
  <button>Poster (vuln)</button>
</form>
<?php foreach($rows as $r){ echo "<div><strong>Par ".$r['membre_id']."</strong>:
".$r['contenu']."</div>"; } ?>
```

---

src/vuln/auth_vuln.php

```
<?php
require __DIR__ . '/../init.php';
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
  // authentification rapide (vuln)
  $user = $_POST['user'] ?? '';
  $_SESSION['user'] = $user; // pas de régénération
  header('Location: ../index.php');
  exit;
}
```

```
?>
<form method="post"><input name="user"><button>Login vuln</button></form>
```

---

src/vuln/del_vuln.php

```
<?php
require __DIR__ . '/../init.php';
$id = $_GET['id'] ?? '';
// suppression par GET (vuln CSRF)
if ($id) {
  $pdo->exec("DELETE FROM inscriptions WHERE id = " . intval($id));
  echo "Supprimé";
} else {
  echo "pas d'id";
}
```

---

src/vuln/parse_vuln_xml.php

```
<?php
// parse_vuln_xml.php - montre chargement vulnérable d'un XML
$xml = file_get_contents(__DIR__ . '/../../menu_vuln.xml'); // exemples
$dom = new DOMDocument();
$dom->loadXML($xml); // peut résoudre entités externes -> vuln XXE
echo "<pre>" . htmlspecialchars($dom->saveXML()) . "</pre>";
// htmlspecialchars encode <,>,",',&.
```

✓ Connexion base de données

Créer la base de données as_olympique_db. Voir la structure ci-dessous.

Remarques & déploiement

- **Importer la structure SQL** fournie avant d'utiliser les pages PHP

Structure SQL de la base as_olympique_db

```
-- ------------------------------------------------------

-- DATABASE : as_olympique_db

-- ------------------------------------------------------


CREATE DATABASE IF NOT EXISTS as_olympique_db

  DEFAULT CHARACTER SET utf8mb4

  COLLATE utf8mb4_general_ci;


USE as_olympique_db;


-- ------------------------------------------------------

-- TABLE : users (comptes de connexion)

-- ------------------------------------------------------

CREATE TABLE users (

    id INT AUTO_INCREMENT PRIMARY KEY,

    username VARCHAR(50) NOT NULL UNIQUE,

    password_hash VARCHAR(255) NOT NULL,

    role ENUM('admin','membre') DEFAULT 'membre',

    created_at DATETIME DEFAULT CURRENT_TIMESTAMP

);
```

```
-- ------------------------------------------------------------
-- TABLE : membres (profil personnel des adhérents)
-- ------------------------------------------------------------
CREATE TABLE membres (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT NOT NULL,
    nom VARCHAR(100) NOT NULL,
    prenom VARCHAR(100) NOT NULL,
    email VARCHAR(150) NOT NULL UNIQUE,
    telephone VARCHAR(20),
    date_naissance DATE,
    date_adhesion DATE DEFAULT CURRENT_DATE,
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE
);

-- ------------------------------------------------------------
-- TABLE : activites (sports ou ateliers proposés)
-- ------------------------------------------------------------
CREATE TABLE activites (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nom VARCHAR(100) NOT NULL,
    description TEXT,
    capacite INT DEFAULT 30
);
```

```
-- ------------------------------------------------------------
-- TABLE : inscriptions (membre inscrit à une activité)
-- ------------------------------------------------------------
CREATE TABLE inscriptions (
    id INT AUTO_INCREMENT PRIMARY KEY,
    membre_id INT NOT NULL,
    activite_id INT NOT NULL,
    date_inscription DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (membre_id) REFERENCES membres(id) ON DELETE CASCADE,
    FOREIGN KEY (activite_id) REFERENCES activites(id) ON DELETE CASCADE
);

-- ------------------------------------------------------------
-- TABLE : reservations (réservation d'un créneau / salle)
-- ------------------------------------------------------------
CREATE TABLE reservations (
    id INT AUTO_INCREMENT PRIMARY KEY,
    membre_id INT NOT NULL,
    activite_id INT NOT NULL,
    date_reservation DATE NOT NULL,
    heure_debut TIME NOT NULL,
    heure_fin TIME NOT NULL,
    FOREIGN KEY (membre_id) REFERENCES membres(id) ON DELETE CASCADE,
    FOREIGN KEY (activite_id) REFERENCES activites(id) ON DELETE CASCADE
);
```

```
-- ------------------------------------------------------------
-- TABLE : resultats (résultats de compétitions)
-- ------------------------------------------------------------
CREATE TABLE resultats (
    id INT AUTO_INCREMENT PRIMARY KEY,
    membre_id INT NOT NULL,
    activite_id INT NOT NULL,
    date_evenement DATE NOT NULL,
    performance VARCHAR(100) NOT NULL,
    commentaire TEXT,
    FOREIGN KEY (membre_id) REFERENCES membres(id) ON DELETE CASCADE,
    FOREIGN KEY (activite_id) REFERENCES activites(id) ON DELETE CASCADE
);

-- ------------------------------------------------------------
-- TABLE : photos (uploads liés aux activités)
-- ------------------------------------------------------------
CREATE TABLE photos (
    id INT AUTO_INCREMENT PRIMARY KEY,
    membre_id INT NOT NULL,
    chemin VARCHAR(255) NOT NULL,
    nom_fichier_original VARCHAR(255),
    date_upload DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (membre_id) REFERENCES membres(id) ON DELETE CASCADE
);
```

```
-- ------------------------------------------------------
-- TABLE : commentaires (pour XSS stockée)
-- ------------------------------------------------------
CREATE TABLE commentaires (
  id INT AUTO_INCREMENT PRIMARY KEY,
  membre_id INT NOT NULL,
  contenu TEXT NOT NULL,
  date_post DATETIME DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (membre_id) REFERENCES membres(id) ON DELETE CASCADE
);
```

☑ Création de l'utilisateur MySQL as_user / as_pwd

🔑 *Ce code crée le compte MySQL et lui donne les droits sur la base as_olympique_db*

```
CREATE USER 'as_user'@'localhost' IDENTIFIED BY 'as_pwd';
GRANT ALL PRIVILEGES ON as_olympique_db.* TO 'as_user'@'localhost';
FLUSH PRIVILEGES;
```

Après exécution :

- le compte MySQL **as_user / as_pwd** existe
- le compte applicatif dans la table users existe aussi
- Vous pouvez vous connecter à l'appli via connexion_secure.php

✔ **Cette base permet de travailler et couvrent toutes les failles suivantes dans le TD**

- **Injection SQL** → table *users* (login), *membres* (consultation), *inscriptions*, etc.
- **XSS stockée** → table *commentaires*
- **Upload sécurisé** → table *photos*
- **Session / CSRF** → actions CRUD sur *inscriptions*, *reservations*, *commentaires*
- **XXE** → gérée via fichiers XML externes, pas une table SQL

Partie A — Envoi de fichier (upload)

Rappel des notions clés

- Vérifier $_FILES et l'erreur $_FILES['f']['error'].
- Limiter la taille $_FILES['f']['size'].
- Contrôler l'extension et le MIME type (ne pas se fier seulement à l'extension) → voir Annexes / Document 1.
- Renommer le fichier et stocker hors du webroot ou dans dossier protégé.
- Rejeter les fichiers exécutables (.php, .phtml, .php3, .pl, .sh).
- Définir permissions minimales, utiliser finfo_file() pour vérifier MIME.

Etapes pour constater la problématique

1. Uploader un fichier contenant <?php system($_GET['cmd']); ?>.
2. Accéder via browser uploads/test.php?cmd=ls → si exécution, faille grave.
3. Tester upload d'un très gros fichier > limite serveur.

Contre-mesures à adopter

- Vérifier isset($_FILES['photo']) et $_FILES['photo']['error'].
- Taille max (ex. 2 MB) : $_FILES['photo']['size'].
- Allowed extensions whitelist (jpg, jpeg, png, gif).
- Vérification MIME avec finfo_file().
- Générer nom aléatoire : bin2hex(random_bytes(12)).'.'.$ext.
- Stocker hors webroot ou bloquer exécution .htaccess (deny from all).
- Désinfecter les métadonnées si nécessaire.

Exercice

- Tester un envoi de fichier en utilisant upload_vuln.php en vous basant sur les étapes pour constater la problématique et constater la faille.

- Copier upload_vuln.php en upload_secure.php
- Mettre en place les contre-mesures adéquates sur upload_secure.php en se basant sur les contre-mesures à adopter
- Retester un envoi de fichier avec upload_secure.php → modifier bien votre application pour appeler le bon fichier

📄 Partie B — Transmission des données à l'URL (GET) / paramètres dans l'URL

Rappel des notions clés

- Les paramètres GET sont visibles en clair et modifiables par l'utilisateur.
- Transtypage : forcer le type (ex. intval() pour entiers).
- Validation (ex. preg_match) et bornage des valeurs (limiter boucles).
- Ne pas exécuter directement des données passées par l'URL sans contrôle.

Étapes pour constater la problématique

1. Lancer bonjour_vuln.php?nom=Admin&prenom=Test&repeter=3.
2. Modifier repeter à une valeur énorme ou une chaîne → provoquer DoS ou comportement inattendu.
3. Mettre nom = <script>...</script> pour tester XSS.

Contre-mesures à adopter

- isset() et défauts.
- intval($_GET['repeter']) puis min(max($repeter,0),10).
- htmlspecialchars() à l'affichage.
- Validation regex sur nom/prenom: /^[a-zA-Z \-]{1,50}$/.
- Logging des valeurs anormales.

Exercice

- Tester bonjour_vuln.php en utilisant les étapes pour constater la problématique et constater la faille.
- Copier bonjour_vuln.php en bonjour_secure.php
- Mettre en place les contre-mesures adéquates sur bonjour secure.php en se basant sur les contre-mesures à adopter pour sécuriser et empêcher DoS/XSS reflétée.

- Retester une transmission de données avec bonjour_secure.php → modifier bien votre application pour appeler le bon fichier

Partie C — Injections SQL

Rappel des notions clés

- Ne jamais concaténer les entrées utilisateurs dans une requête SQL.
- Utiliser PDO + requêtes préparées (bindings).
- Transtyper/valider les entrées.
- Principe d'attaque : '; DROP TABLE users; -- ou admin' OR '1'='1.

Étapes pour constater la problématique

1. Tester la connexion en saisissant username: ' OR '1'='1 pour obtenir un accès.
2. Inspecter la BDD et constater une fuite de données.

Contre-mesures à adopter

- Utiliser PDO en mode exceptions.
- Requêtes préparées : $stmt = $pdo->prepare('SELECT * FROM users WHERE username = ?'); $stmt->execute([$username]);
- Hachage des mots de passe (password_hash, password_verify) → voir Annexes / Document 2
- Limiter droits DB (compte DB non admin).

Exercice

- Tester connexion_vuln.php en utilisant les étapes pour constater la problématique et constater la faille.
- Copier connexion_vuln.php en connexion secure.php
- Mettre en place les contre-mesures adéquates sur connexion_secure.php en se basant sur les contre-mesures à adopter pour sécuriser l'accès à la BDD et empêcher une faille XSS stockée
- Retester une tentative d'injection SQL avec connexion_secure.php → modifier bien votre application pour appeler le bon fichier

Partie D — Faille XSS (reflétée et stockée)

Rappel des notions clés

- XSS reflétée : entrée renvoyée immédiatement dans la page.
- XSS stockée : entrée stockée en DB, exécutée quand la page affichant-la donnée est visitée.
- Prévention : encodage en sortie (htmlspecialchars()), validation d'entrée, Content Security Policy (CSP).

Étapes pour constater la problématique

1. Injecter <script>alert(1)</script> dans pseudo.
2. Pour une faille XSS stockée : poster un commentaire contenant <img src=x onerror=...> et visiter la page affichant les commentaires.

Contre-mesures à adopter

- Encodage systématique en sortie : htmlspecialchars($value, ENT_QUOTES, 'UTF-8').
- Filtrage côté serveur pour formats connus.
- CSP (exemple minimal via header) → (ex : Content-Security-Policy: default-src 'self'; script-src 'self').
- HTTPOnly pour cookies pour limiter le vol de cookie via XSS.

Exercice

- Tester commentaire_vuln.php en utilisant les étapes pour constater la problématique et constater la faille.
- Copier commentaire_vuln.php en commentaire_secure.php
- Mettre en place les contre-mesures adéquates sur commentaire_secure.php en se basant sur les contre-mesures à adopter pour empêcher une faille XSS stockée ou reflétée

- Retester une tentative de faille XSS avec connexion_secure.php → modifier bien votre application pour appeler le bon fichier

📄 Partie E — Piratage de session (session hijacking / fixation / brute force)

Rappel des notions clés

- session_regenerate_id(true) après authentification.
- Cookies avec HttpOnly, Secure, SameSite=Strict.
- Limiter tentatives de connexion (brute force) et verrouillage temporaire.
- Stocker le hash du mot de passe, ne jamais stocker mot de passe en clair.
- Valider l'IP / user agent (avec prudence) pour détecter anomalies.

Étapes pour constater la problématique

1. Authentifier, récupérer le cookie session (dans outils dev).
2. Simuler la fixation en forçant un id de session connu avant login.
3. Tester les attaques par force brute sans limitation.

Contre-mesures à adopter

- session_start() + session_regenerate_id(true) après login.
- Flags cookie: session_set_cookie_params(['httponly'=>true,'secure'=>true,'samesite'=>'Strict']);
- Implémenter compteur d'échecs et lockout temporisé.
- Optionnel : 2FA pour comptes sensibles.

Exercice

- Tester auth_vuln.php en utilisant les étapes pour constater la problématique et constater la faille.
- Copier auth_vuln.php en auth secure.php
- Mettre en place les contre-mesures adéquates sur auth_secure.php en se basant sur les contre-mesures à adopter pour empêcher une faille XSS stockée ou reflétée

- Retester une tentative de piratage de session avec auth secure.php → modifier bien votre application pour appeler le bon fichier

Partie F — Faille CSRF

Rappel des notions clés

- CSRF : exploitation d'une session authentifiée de la victime par requêtes forcées.
- Contre-mesure principale : jeton anti-CSRF unique par session/formulaire (synchronizer token pattern).
- Vérifier méthode HTTP (POST pour actions modifiantes).
- SameSite cookie aide mais ne remplace les tokens.

Étapes pour constater la problématique

1. Créer page HTML sur un autre domaine faisant GET sur del.php?id=5.
2. Si admin clique, suppression s'exécute.

Contre-mesures à adopter

- Utiliser POST pour actions sensibles → Convertir suppression GET en formulaire POST avec token CSRF.
- Ajouter <input type="hidden" name="csrf_token" value="<?= $_SESSION['csrf_token'] ?>">.
- Vérifier token côté serveur hash_equals().

Exercice

- Tester del_vuln.php en utilisant les étapes pour constater la problématique et constater la faille.
- Copier del_vuln.php en del_secure.php
- Mettre en place les contre-mesures adéquates sur del secure.php en se basant sur les contre-mesures à adopter
- Retester une tentative de faille CSRF avec del_secure.php → modifier bien votre application pour appeler le bon fichier

Partie G (Pour aller plus loin) — Faille XXE (XML)

Rappel des notions clés

- XXE exploite le traitement d'entités externes dans XML pour lire fichiers locaux ou exfiltrer données, ou DoS (Billion Laughs).
- Prévention : désactiver la résolution des entités externes et interdire DTD externes.

Contre-mesures à adopter

- Utiliser LIBXML_NONET and disabling external entity loading.
- Exemple sûr :

```
$xml = file_get_contents('menu.xml');

$dom = new DOMDocument();

libxml_disable_entity_loader(true); // deprecated in recent PHP, but illustrative

$dom->loadXML($xml, LIBXML_NOENT | LIBXML_NONET);
```

Conseil : utiliser simplexml_load_string with LIBXML_NONET and validate, ou parser JSON au lieu de XML si possible.

Exercice

- Tester parse_vul_xml.php en utilisant les étapes pour constater la problématique et constater la faille.
- Copier parse_vul_xml.php en parse_secure_xml.php
- Mettre en place les contre-mesures adéquates sur parse_secure_xml.php en se basant sur les contre-mesures à adopter
- Retester une tentative de faille XXE avec del_secure.php → modifier bien votre application pour appeler le bon fichier

✍ Travail réflexif (à rendre)

Rédiger une synthèse sur l'importance de sécuriser une application (10–15 lignes) en vous basant sur le TD.

ANNEXES

Document 1

Le **MIME type** (pour *Multipurpose Internet Mail Extensions*) est une **indication du type réel d'un fichier**, utilisée par les navigateurs, serveurs et applications pour savoir **comment interpréter** son contenu.

🎯 À quoi sert un MIME type ?

Il permet de déterminer la nature du fichier, par exemple :

Type de fichier	Extension	MIME type
Image JPEG	.jpg	image/jpeg
Image PNG	.png	image/png
Fichier PDF	.pdf	application/pdf
Page HTML	.html	text/html
Script JavaScript	.js	application/javascript

Ecriture PHP

```
$finfo = finfo_open(FILEINFO_MIME_TYPE);
$mimetype = finfo_file($finfo, $_FILES['photo']['tmp_name']);
```

Ainsi :
📌 Un fichier PHP renommé en .jpg aura un MIME type **text/plain** ou **application/x-php** → donc rejeté.

🧠 En résumé

Le **MIME type = carte d'identité du fichier**, utilisée pour :

- ✓ savoir comment l'afficher ou l'exécuter
- ✓ sécuriser l'upload en contrôlant le contenu réel du fichier
- ✓ empêcher les fichiers déguisés (upload de shell PHP, scripts malveillants)

Document 2

🔐 Rôle de password_hash() et password_verify() en PHP

🎯 1. password_hash() — *Créer un mot de passe sécurisé*

password_hash() sert à **générer un hash sécurisé** du mot de passe avant de l'enregistrer dans la base de données.

Pourquoi ?

- On **ne doit jamais stocker un mot de passe en clair**.
- Un hash est **irréversible** : même si la base de données fuit, l'attaquant ne récupère pas directement les vrais mots de passe.
- password_hash() utilise automatiquement un **algorithme robuste (par défaut : bcrypt)** et ajoute un **salt unique** pour chaque utilisateur.

Exemple PHP :

```php
$hash = password_hash($motdepasse, PASSWORD_DEFAULT);
```

📌 Ce qu'il faut retenir :

👉 Le hash obtenu ne ressemble jamais au mot de passe d'origine.

👉 Chaque appel génère un résultat différent même pour un même mot de passe (grâce au salt).

🎯 2. password_verify() — *Vérifier un mot de passe à la connexion*

password_verify() sert à **vérifier que le mot de passe saisi par l'utilisateur correspond au hash stocké** dans la base.

Comment ?

- L'utilisateur entre son mot de passe.
- On compare ce mot de passe avec son hash enregistré.
- password_verify() calcule un nouveau hash **avec les mêmes paramètres que celui stocké** et fait la comparaison.

Exemple en PHP :

```php
if (password_verify($motdepasseEntre, $hashStocke)) {
    echo "Connexion OK";
} else {
    echo "Mot de passe incorrect";
}
```

📌 **Ce qu'il faut retenir :**

👉 On ne déchiffre jamais le mot de passe.

👉 On compare "le hash du mot de passe entré" avec "le hash stocké".

🧠 **En résumé**

Fonction	Rôle	Important
password_hash()	Transforme un mot de passe en hash sécurisé	Ajoute un *salt*, utilise un algo puissant, résultat différent à chaque fois
password_verify()	Vérifie que le mot de passe saisi correspond au hash enregistré	Ne déchiffre rien ; compare via un algorithme spécialisé

Document 3

🎯 **Qu'est-ce qu'un attaquant sur le chemin ?**

Dans une attaque de type « on-path » (attaque de l'homme du milieu), les acteurs malveillants se placent entre deux appareils (souvent un navigateur web et un serveur web) et interceptent ou modifient les communications entre les deux. Ils peuvent alors collecter des informations et se faire passer pour l'un ou l'autre des deux agents. Outre les sites web, ces attaques peuvent également viser les communications par courrier électronique, les recherches DNS et les réseaux Wi-Fi publics. Les cibles typiques des adeptes de l'attaque de l'homme du milieu sont les entreprises SaaS, les entreprises spécialisées dans l'e-commerce et les utilisateurs d'applications financières.