Projet Cybersécurité - Fil rouge

Présentation générale

Vous allez développer un petit site web permettant de **poster et afficher des commentaires**. Ce projet fil rouge couvre trois volets complémentaires :

1. **Développer** l'application (front + back) en PHP + MySQL.
2. **Héberger** l'application sur une machine virtuelle (VM) afin de la servir et de la rendre accessible dans un environnement contrôlé.
3. **Découvrir les attaques** possibles (injection SQL, XSS, CSRF, MITM liée à l'absence de HTTPS, etc.) puis **protéger** l'application soit côté code (option SLAM), soit côté infrastructure (option SISR).

L'objectif est d'apprendre le cycle complet : développement → déploiement → analyse de risques → durcissement.

Livrables attendus

- Dossier source `tp-comments/` (code + SQL + README).
- VM configurée avec l'application accessible (captures d'écran + procédure).
- Rapport court (2-4 pages) décrivant vulnérabilités découvertes et mesures prises selon l'option choisie (SLAM ou SISR).
- Présentation orale notée (5-10 minutes) par groupe - Mardi 18 novembre 2025

Pré-requis techniques

- PHP 7.0+ (avec extension PDO MySQL)
- MySQL ou MariaDB
- Un éditeur de texte (VS Code, etc.)
- VirtualBox/VMware

Phase 1 - Développement (livrable minimal)

Objectifs

- Construire l'application web permettant de poster et afficher des commentaires.
- Comprendre le cheminement des données : formulaire → PHP → base de données → affichage.

Structure du projet (à créer)

```
tp-comments/
├─ index.html         # formulaire pour poster un commentaire
├─ submit.php         # script PHP qui insère en base (vulnérable)
├─ comments.php       # affiche les commentaires sans échappement (vulnérable)
├─ sql/
│  └─ init.sql        # script de création de la base et de la table
└─ README.md          # explications pour lancer localement
```

sql/init.sql

```sql
CREATE DATABASE tp_comments CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
USE tp_comments;

CREATE TABLE comments (
  id INT AUTO_INCREMENT PRIMARY KEY,
  content TEXT NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
) ENGINE=InnoDB;
```

index.html

```html
<!doctype html>
<html lang="fr">
<head>
  <meta charset="utf-8">
  <title>Poster un commentaire (vulnérable)</title>
</head>
<body>
  <h1>Poster un commentaire</h1>

  <form method="post" action="submit.php">
    <label>Message<br>
      <textarea name="content" rows="5" cols="60" required></textarea>
    </label><br>
    <button type="submit">Envoyer</button>
  </form>

  <p>Après envoi, vous serez redirigé vers la page qui affiche tous les commentaires.</p>
</body>
</html>
```

submit.php

```php
<?php
// submit.php
$host = '127.0.0.1';
$db   = 'tp_comments';
$user = 'root';
$pass = ''; // adapter si besoin
$charset = 'utf8mb4';
$dsn = "mysql:host=$host;dbname=$db;charset=$charset";

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $content = $_POST['content'] ?? '';

    try {
```

```
        $pdo = new PDO($dsn, $user, $pass);
        $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        $sql = "INSERT INTO comments (content) VALUES ('" . $content . "')";
        $pdo->exec($sql);

        header('Location: comments.php');
        exit;
    } catch (PDOException $e) {
        echo "Erreur SQL : " . $e->getMessage();
    }
} else {
    header('Location: index.html');
    exit;
}
```

comments.php

```
<?php
$host = '127.0.0.1';
$db   = 'tp_comments';
$user = 'root';
$pass = ''; // adapter
$charset = 'utf8mb4';
$dsn = "mysql:host=$host;dbname=$db;charset=$charset";

try {
    $pdo = new PDO($dsn, $user, $pass);
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    $stmt = $pdo->query("SELECT id, content, created_at FROM comments ORDER BY created_at
DESC");
    $comments = $stmt->fetchAll(PDO::FETCH_ASSOC);
} catch (PDOException $e) {
    echo "Erreur SQL : " . $e->getMessage();
    exit;
}
?>
<!doctype html>
<html lang="fr">
<head>
  <meta charset="utf-8">
  <title>Commentaires</title>
</head>
<body>
  <h1>Commentaires postés</h1>
  <p><a href="index.html">Poster un nouveau commentaire</a></p>

  <?php if (empty($comments)): ?>
    <p>Aucun commentaire pour le moment.</p>
  <?php else: ?>
```

```
    <ul>
      <?php foreach ($comments as $c): ?>
        <li>
          <div><?= $c['created_at'] ?></div>
          <div><?= $c['content'] ?></div>
        </li>
      <?php endforeach; ?>
    </ul>
  <?php endif; ?>
</body>
</html>
```

Tests locaux (Phase 1)

- Lancer le serveur PHP intégré : `php -S localhost:8000` depuis le dossier `tp-comments/` .
- Visiter `http://localhost:8000/index.html` , poster un commentaire, puis vérifier `comments.php` .

Phase 2 - Hébergement sur une machine virtuelle (VM)

Objectifs

- Déployer l'application sur une VM (VirtualBox, VMware ou cloud), comprendre réseau, services et sécurité basique.

Tâches conseillées

1. **Créer la VM** (Linux Debian/Ubuntu recommandé).
2. **Installer** : Apache2 ou Nginx + PHP + MySQL (LAMP/LEMP) - ou utiliser `php -S` pour tests rapides.
3. **Transférer** les fichiers (`scp` , dossier partagé, etc.).
4. **Créer** la base `tp_comments` et exécuter `sql/init.sql` .
5. **Configurer** : utilisateur MySQL non-root pour l'app, pare-feu (UFW), et ouverture du port HTTP/HTTPS si nécessaire.
6. **Documenter** la procédure (README avec commandes utilisées, IP de la VM, état des services).

Démarrage rapide (serveur PHP intégré)

Dans le dossier `tp-comments/` sur la VM, lancer :

```
php -S 0.0.0.0:8000 -t .
```

Puis accéder depuis une machine du réseau (si autorisé) : `http://<IP_VM>:8000/index.html`

Phase 3 - Découverte des attaques (environnement contrôlé uniquement)

> ***Rappel éthique*** *: toutes les attaques et démonstrations se font uniquement en labo/VM contrôlée. Ne jamais attaquer des systèmes externes.*

Attaques à étudier et démontrer

1. **Injection SQL**
    - Pourquoi : concaténation des entrées dans des requêtes SQL.
    - Exemple pédagogique (non destructif) :

      ```
      '); INSERT INTO comments (content) VALUES ('Injection SQL réussie'); --
      ```

    - Observations : insertion non prévue, erreurs SQL, etc.
2. **XSS (Cross-Site Scripting)**
    - Pourquoi : affichage de contenus utilisateurs sans échappement.
    - Exemple : `<script>alert('XSS !')</script>` - provoque exécution côté navigateur.
3. **CSRF (Cross-Site Request Forgery)** - explication et illustration théorique
    - Pourquoi : absence de token CSRF permet à un attaquant de forger une requête depuis un autre site.
    - Démonstration recommandée : construction d'une page "malicieuse" locale qui fait un POST de test (encadré par l'enseignant).
4. **MITM (Man-In-The-Middle) - ajouté**
    - **Contexte** : le site fonctionne en HTTP non chiffré. Les données transitent en clair.
    - **Principe** : un attaquant sur le réseau peut lire/modifier les requêtes/réponses (formulaires, cookies, etc.).
    - **Démonstration pédagogique** :
        - Faire circuler un POST en clair sur le réseau local et montrer (dans le labo) le contenu lisible.
        - Refaire la démonstration après activation de HTTPS et constater que le trafic est chiffré.
    - **Contre-mesures** : HTTPS/TLS, HSTS, cookies `Secure` / `HttpOnly`, terminaison TLS sur reverse-proxy.

Phase 4 - Protection & durcissement (choix d'option : SLAM ou SISR)

Option A - SLAM : sécurité côté **application / code**

Travail axé sur la sécurisation du code et des flux applicatifs.

Tâches recommandées

- Utiliser **prepared statements** (PDO `prepare` + `execute`) pour toutes les requêtes SQL.
- Protéger l'affichage avec `htmlspecialchars()` afin d'éviter le XSS.
- Implémenter un **jeton CSRF** dans le formulaire et vérifier-le côté serveur.
- Validation stricte côté serveur (taille, charset, sanitation).
- Ne pas afficher d'erreurs SQL en clair ; journaliser les erreurs côté serveur.
- Gérer correctement les cookies (`Secure` , `HttpOnly` , SameSite).
- Livrable : code corrigé + note technique expliquant les changements.

Critères

- Les attaques démontrées en Phase 3 n'ont plus d'effet sur la version corrigée.
- Présentation expliquant les corrections et comment elles protègent l'appli.

Option B - SISR : sécurité côté **infrastructure / réseau**

Travail axé sur le durcissement de la VM et de l'infrastructure.

Tâches recommandées

- Isoler l'application sur une VM ou un conteneur.
- Créer un utilisateur MySQL à privilèges limités (pas `root`) et limiter les droits.
- Configurer pare-feu (UFW/iptables) pour n'ouvrir que les ports nécessaires.
- Mettre en place HTTPS (certificat auto-signé en labo ou Let's Encrypt si possible).
- Installer un WAF léger (ex : ModSecurity) et expérimenter des règles.
- Configurer la rotation des logs et une stratégie de sauvegarde.
- Livrable : scripts/commandes, captures d'écran et documentation.

Critères

- Démonstration que l'infrastructure réduit l'impact des attaques (ex : WAF bloque certains payloads).
- Documentation expliquant les choix d'architecture.