STI - Projet Partie 2

Rapport

Léo Cortès

Steve Henriquet

7 janvier 2019

heig-vd
HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Hes·so
Haute Ecole Spécialisée
de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences and Arts
Western Switzerland

Contents

List of Figures

Introduction

Analyse de menaces

Agents menaçants

En principe, notre application devraient être accessibles depuis n'importe qui sur Internet. Les agents menaçants peuvent donc être n'importe qui. Notre application n'est pas révolutionnaire et bien moins efficace que la concurrence. Des attaquants concurrents seraient donc peu probables. Les principaux agents menaçants seraient surtout nous (les élèves du cours STI) ou n'importe quelle personne souhaitant s'amuser.

Vecteurs d'attaque

Tout se passe sur le web. Les utilisateurs peuvent (et doivent) manipuler des entrées pour envoyer des messages ou mettre à jour leurs profils. Des entrées malveillantes pourraient exploiter des vulnérabilités au sein de notre application.

Faiblesses de sécurité

Comme nous avons accès au code ne notre application, nous avons une idée des vulnérabilités que nous pouvons y trouver. Dans un premier temps, les entrées utilisateur ne sont pas assainies et sont envoyées telle quelle. Cela laisse place à de nombreuses failles comme des injections SQL ou des XSS.
De plus, les accès à la DB n'utilisent pas de query pré-préparées. Cela pourrait donc donner lieu à des injections SQL supplémentaires.
L'application utilise du HTTP simple et aucun chiffrement. Des données pourraient donc être sniffées et si une fuite de données à lieu, toutes les données pourraient être accessibles en clair.

Contrôle de sécurité

Les accès sont, en principe, vérifiés. Un utilisateurs ne peut accéder qu'aux mails qu'il a envoyé ou reçu et pas ceux d'autres utilisateurs. Les accès administrateurs sont réservés aux administrateurs. Finalement, il n'est pas

possible d'accéder à des fichiers stockés sur le serveur. Si un utilisateur essaie d'accéder à une page non prévue, il sera redirigé.

Impacts techniques

Si des messages sont accédés par des personnes non autorisées, cela peut entraîner une perte de confidentialité.
Un accès non autorisé pourrait donner le droit à un attaquant de modifier, voire supprimer de l'information. Une perte d'intégrité peut donc en découler.

Impacts business

L'impact business serait faible. En effet, notre application n'est pas conçue dans un but lucratif et ne sera probablement pas maintenue par la suite. Si une telle application était mise en production dans le but d'offrir un service sérieux, cela pourrait être problématique car des clients pourraient stocker des informations personnelles voire confidentielles dans leurs messages. Une fuite de données ou des vulnérabilité pourrait entraîner une perte de confiance complète des utilisateurs.

Vulnérabilités et corrections

XSS

Exploitation : démo basique

Le programme est vulnérable aux attaques XSS. Les balises HTML sont correctement inteprétée, ainsi que les balises de script. C'est problématique car un attaquand pourrait envoyer un message malicieux à un administrateur. Lorsque l'admin se connecte, le script pourrait récupérer ses cookies et l'attaquant pourrait les rejouer et devenir donc administrateur.
Voici le message envoyé par un attaquant quelconque :

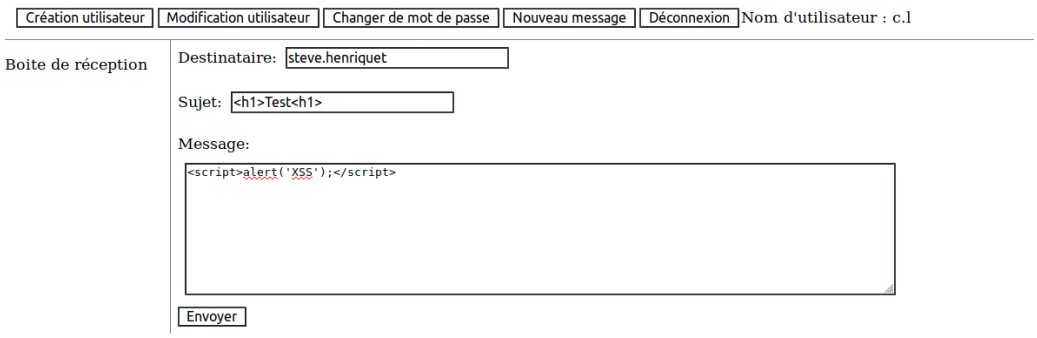

Figure 1: Message malicieux

Voici le résultat dans la boîte de réception de la victime.

Figure 2: Boîte de réception

Lorsque la victime l'ouvre, voici le script est correctement exécuté.

Figure 3: Exécution XSS

Exploitation : vol de cookies

Essayons de récupérer les cookies de l'administrateur *steve.henriquet*. Pour cela, un attaquant peut simplement écrire le message suivant :

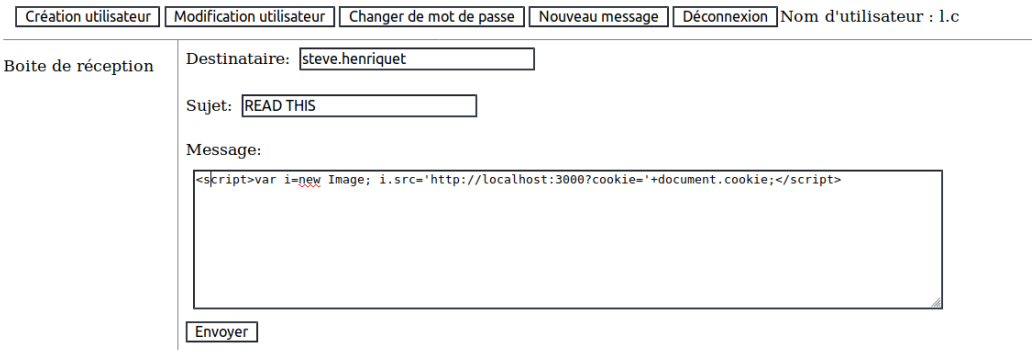

Figure 4: Message malveillant voleur de cookier

Le script contenu dans le message va simplement envoyer une requête contenant les cookies courant sur une serveur externe (ici *localhost:3000*. L'administrateur *steve.henriquet* ne se doute de rien en ouvrant son message.

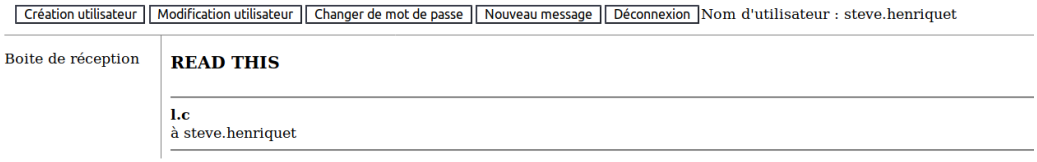

Figure 5: La faille n'est pas visible pour la victime

Finalement, si on se connecte sur notre serveur externe, on peut constater qu'un nouveau cookier contenant un id de session est apparu.

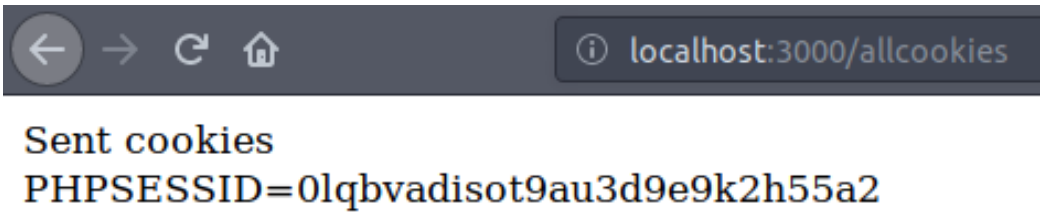

Figure 6: Le cookie volé

Ce cookie peut tout simplement être rejoué et l'attaquant peut endosser l'identité de l'administrateur *steve.henriquet*.

Correction

Les entrées ont du être "sanitisées". Nous avons utilisé le filtre *FILTER_SANITIZE_STRING* dans les divers inputs accessibles.

```
$pass = filter_var ( $_POST["pass"], filter: FILTER_SANITIZE_STRING);
$passCheck = filter_var ( $_POST["passCheck"], filter: FILTER_SANITIZE_STRING);
```

Figure 7: Exemple d'assainisation

Injection SQL

D'après l'outil *SQLMap*, on peut déjà sortir quelques informations depuis la page de login, dont les tables de notre DB.

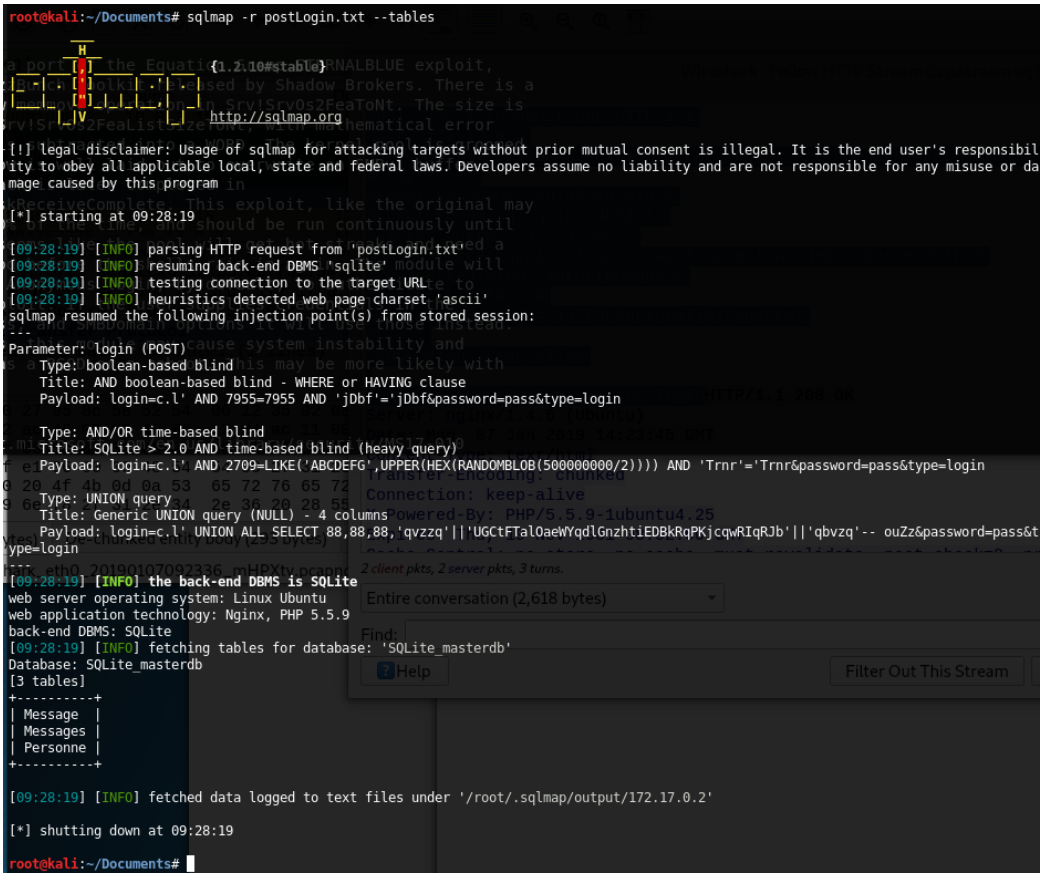

Figure 8: Résultat de *SQLMap* sur la page de login

Exploitation : se connecter sans mot de passe

Dans notre page de login, en essayant d'injecter du code simple, nous avons remarqué que le login se faisait tout de même. D'après notre code, nous avons découvert une injection SQL permettant de bypasser le login et de se connecter en tant que n'importe quel utilisateur sans connaître son mot de passe. La requête effectuée au login est la suivante :

```
1          SELECT Actif, User_id, Admin, Username FROM
               Personne WHERE Username='$name' AND Pass=
               '$password'
```

Puis le code suivant permet de vérifier si les credentials sont valides :

```php
function checkLogin($arr) {
    $bool = true;
    $arrToSend = array();
    foreach ($arr as $row) {
        // Permet de savoir si l'utilisateur courant est actif ou
            non
        if(empty($row['Actif'])){
            array_push($arrToSend, "error", "Compte_inactif!" . "
                <br/>") ;
            echo json_encode($arrToSend);
            $bool = false;
        }
        else{
            // Permet de setter les variables de session utile
                pour toute la connection
            $_SESSION["user_id"] = $row['User_id'];
            $_SESSION["username"] = $row['Username'];
            if(empty($row['Admin'])){
                $_SESSION["admin"] = 0;
            }
            else{
                $_SESSION["admin"] = 1;
            }
            array_push($arrToSend, "body", manageLayout()) ;
            echo json_encode($arrToSend);
            $bool = false;
        }
    }
    return $bool;
}
```

En écourtant la requête de la sorte :

```sql
SELECT Actif, User_id, Admin, Username FROM
    Personne WHERE Username='$name';
```

On peut se connecter sans avoir besoin de password ! Pour se faire, il suffit d'ajouter les deux caractères *';* à la fin de son login et entrer un mode de passe random pour pas que le champs soit vide.

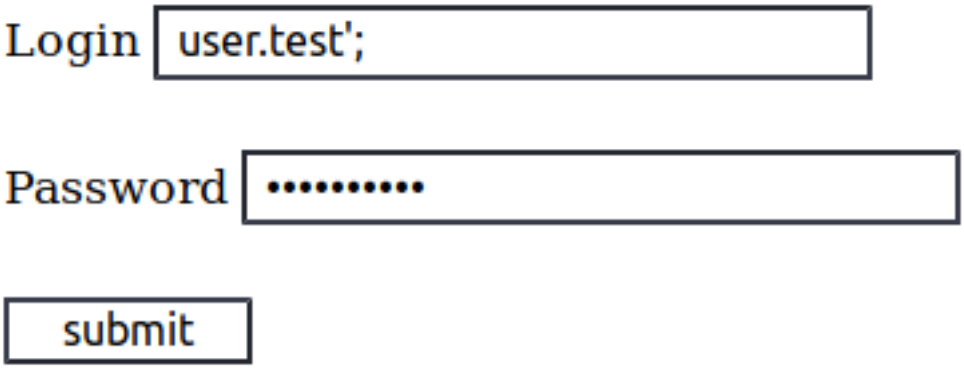

Figure 9: Se connecter comme *user.test* sans connaître son mot de passe

Exploitation : changer le mot de passe de n'importe quel utilisateur

Voici le code PHP associé à la modification d'un mot de passe :

```
    else{
        $dbPass = '';
        $result = $file_db->query("SELECT Pass FROM Personne
            WHERE User_id ='" . $user_id . "';");
        foreach ($result as $row) {
            $dbPass = $row['Pass'];
        }
        if($oldPass == $dbPass){
            $result = $file_db->query("UPDATE Personne SET Pass =
                '" . $newPass . "' WHERE User_id ='" . $user_id
                . "';");
            echo "Mot de passe mis a jour avec succes";
        }
        else {
            echo "Mot de passe errone!";
        }
    }
```

Avec une injection SQL, peut modifier la deuxième requête afin de modifier le password d'un autre utilisateur. Si le nouveau password que l'on rentre est le payload suivant : *newpass' WHERE Username='victim.user'* –, alors l'utilisateur *victim.user* aura son mot de passe modifié en *newPass*. La partie *WHERE User_id ='" . $user_id . "';* de la requête originale n'est pas exécutée car commentée par –.

Correction

Tout comme pour les failles XSS, les entrées ont du être assainies. De plus, les query simples utilisées dans le code PHP ont été passées en prepared statements.

```php
$result = $file_db->query( statement: "SELECT Username FROM Personne WHERE Username='" . $username . "';");
foreach ($result as $row) {
    $usernameDb = $row['Username'];
}
```

Figure 10: Avant d'utiliser des prepared statements il y avait juste de simples queries

```php
$sth = $file_db->prepare( statement: "SELECT Username FROM Personne WHERE Username=:username");
$sth->execute(array(':username' => $username));
$result = $sth->fetchAll( fetch_style: PDO::FETCH_ASSOC);

foreach ($result as $row) {
    $usernameDb = $row['Username'];
}
```

Figure 11: Example de prepared statement

Mauvaise destruction des session

Exploit

Les sessions sont mal détruites. Potentiellement, après le login d'un admin, un utilisateur pourrait se retrouver à avoir accès à des informations réservées aux administrateurs.

```php
<?php
/** Created by PhpStorm. ...*/

session_start();

unset($_SESSION["user_id"]);

echo 'Vous êtes maintenant déconnecté';
```

Figure 12: Mauvaise destruction de la session

Correction

Accès au page sans login

Exploitation

Envoie de mail par exemple.

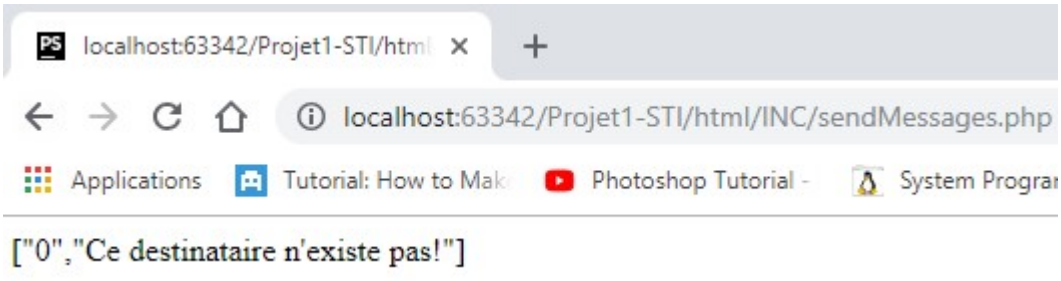

Figure 13: Accès à la page

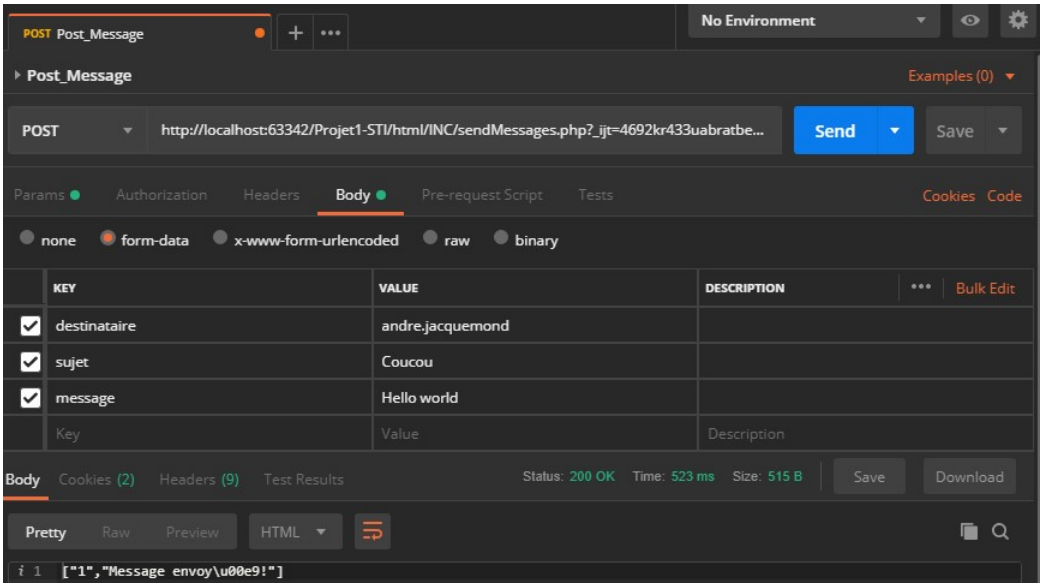

Figure 14: Requête Post réussi

Figure 15: Message reçu

Correction

Vérification de l'existante de la variable de session *$_SESSION["user_id"]*.

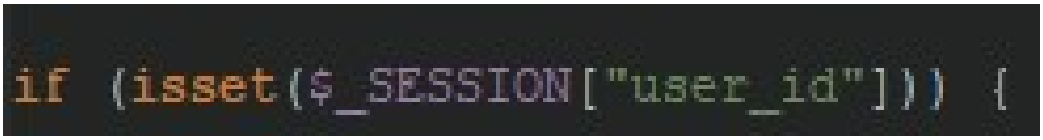

Figure 16: Protection à ajouter aux différentes pages

L'ajout de cette vérification s'effectue sur tous les fichiers à l'execption de index.php, login.php et des fichiers effectuant déjà la vérification par rapport au fait d'être administrateur

Problèmes non résolvables

- PHP 5.6
- Gestion des cookies de session