

Yuheng CHEN - Louis GREINER - Benjamin MISSAOUI

SR03 **Devoir 3**

Sécurité des applications web

DÉVELOPPEMENT D'UNE APPLICATION DE BANQUE EN LIGNE SÉCURISÉE



P21

TABLE DES MATIÈRES

| | |
|--|----|
| TABLE DES MATIÈRES | 2 |
| 1. INTRODUCTION ET INSTALLATION DE L'APPLICATION | 3 |
| 2. PRÉSENTATION DES FONCTIONNALITÉS ET FAILLES | 5 |
| 3.1 ~ Injection SQL ~ | 7 |
| 3.2 ~ Cross-site scripting (XSS) ~ | 8 |
| 3.3 ~ Violation de contrôle d'accès ~ | 9 |
| 3.4 ~ Violation de gestion de session ~ | 11 |
| 3.5 ~ Falsification de requête (CSRF) ~ | 12 |
| 3.6 ~ Vulnérabilité d'un composant ~ | 14 |
| 3.7 ~ Chiffrement des données sensibles ~ | 14 |
| 3.8 ~ Accès aux répertoires par HTTP ~ | 15 |
| 3.9 ~ Scripts de redirection ~ | 17 |

1. INTRODUCTION ET INSTALLATION DE L'APPLICATION

Dans ce devoir, nous allons créer un site sécurisé destiné aux clients de la banque fictive *BankMate*. Ce site va permettre aux clients de s'authentifier, pour envoyer et recevoir des messages aux autres clients, effectuer des virements, et consulter la liste des clients (seulement pour les Employés).

Pour commencer à utiliser l'application, commencez par télécharger l'archive contenant le code à l'adresse : <https://gitlab.utc.fr/lgreiner/sr03devoir3>. Décompressez l'archive et placez le dossier obtenu dans un répertoire de déploiement de PHP. Renommez le dossier avec par exemple "devoir3_chen_greiner_missaoui" pour le distinguer, puis, dans le fichier config/config.php, changez vos paramètres de connexion pour correspondre à votre environnement. Allez ensuite sur phpmyadmin, et importez le fichier dbV2.sql pour créer les tables et ajouter des enregistrements. Une fois cela fait, vous pourrez accéder à la page de connexion via votre navigateur :

```
localhost:{num_port}/devoir3_chen_greiner_missaoui
```

La page de connexion se présente comme suit :

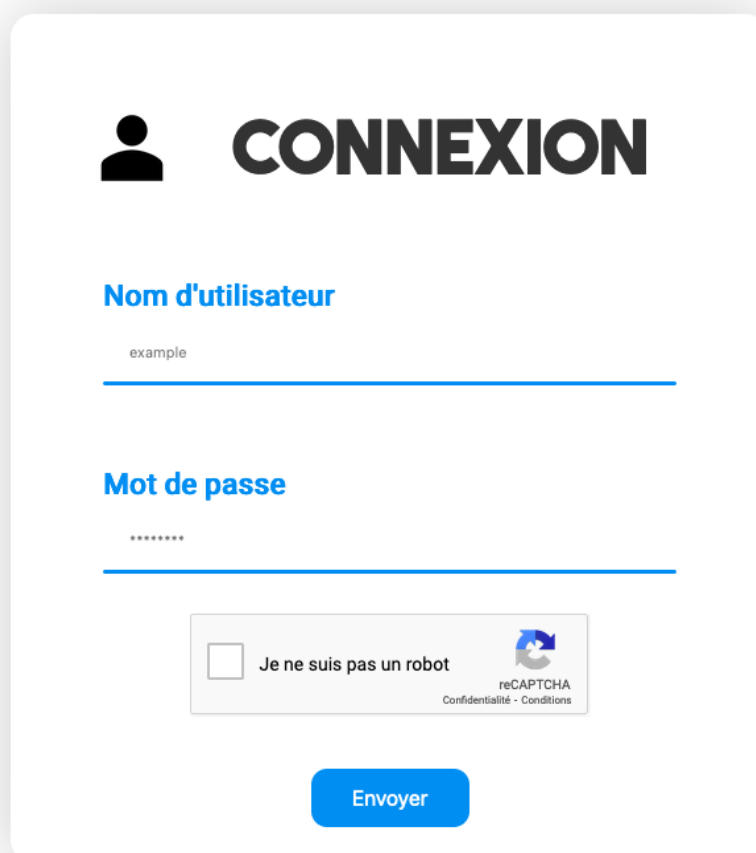


Fig 1 : Page de connexion vw_login.php

La base de données contient plusieurs utilisateurs avec lesquels vous pouvez vous connecter. Les mots de passe étant chiffrés, voici les enregistrements de la table `USERS` en clair (les mots de passe sont volontairement simples mais on supposera qu'en réalité, les utilisateurs ont choisi des mots de passe plus robustes) :

| id_user | login | mot_de_passe | nom | prenom | numero_compte | profil_user | solde_compte |
|---------|----------|--------------|----------|----------|---------------|-------------|--------------|
| 1 | bmissaou | bmissaou_pwd | Missaoui | Benjamin | 8247285830 | EMPLOYE | 127001.37 |
| 2 | yuchen | yuchen_pwd | Chen | Yuheng | 4964011994 | EMPLOYE | 144006.11 |
| 3 | lgreiner | lgreiner_pwd | Greiner | Louis | 7873234879 | EMPLOYE | 141482.29 |
| 4 | amerkel | amerkel_pwd | Merkel | Angela | 5789446841 | CLIENT | 341.56 |
| 5 | dtrump | dtrump_pwd | Trump | Donald | 6201640760 | CLIENT | 556.68 |

Choisissez l'utilisateur avec lequel vous souhaitez vous authentifier et entrez ces identifiants sur la page de connexion. Vous arriverez alors sur la page `vw_moncompte.php` :

[Message](#)[Virement](#)[Messagerie](#)[Déconnexion](#)

BIENVENUE BENJAMIN MISSAOUI

Utilisez les menus déroulants ou un des raccourcis rapides :

Raccourcis

Envoyer un message

Effectuer un virement

Accès à la messagerie

Vos informations

Numéro de compte : 8247285830

Missaoui Benjamin

Solde : 127 001,37€

Profil : EMPLOYE

[Consulter les clients](#)

Fig 2 : Page d'accueil `vw_moncompte.php`

2. PRÉSENTATION DES FONCTIONNALITÉS ET FAILLES

Une fois connecté, vous pouvez accéder aux différentes fonctionnalités via la navbar ou les boutons dans le widget “Raccourci”. Les utilisateurs avec le statut EMPLOYE ont aussi l’option “Consulter les clients” sur la page d’accueil. Voici ici des captures d’écran de chaque fonctionnalité.

The image displays two side-by-side screenshots of a web application interface. The left screenshot shows the 'MESSAGE' form, which includes a header with an envelope icon and the title 'MESSAGE'. Below the header, there are three main sections: 'Destinataire' (recipient) with the name 'Chen Yuheng', 'Sujet' (subject) with a placeholder 'Sujet...', and 'Message' with a text area for the message content. At the bottom of the form are two buttons: 'Accueil' (Home) and 'Envoyer' (Send). The right screenshot shows the 'VIREMENT' (Transfer) form, which includes a header with a money icon and the title 'VIREMENT'. Below the header, there are four main sections: 'Votre solde : 127 001,37€' (Your balance), 'Compte destinataire' (recipient account) with the name 'Chen Yuheng (n° 4964011994)', 'Montant à transférer' (amount to transfer) with a value of '135' and a currency symbol '€', and 'Mot de passe' (password) with a masked input field. At the bottom of the form are two buttons: 'Accueil' (Home) and 'Envoyer' (Send).

Fig 3 : Interface d’envoi d’un message (à gauche) et d’un virement (à droite)

MESSAGES REÇUS





| Expéditeur | Sujet | Message |
|---------------|-------------------------------|---|
| Chen Yuheng | Test XSS | Bonjour Benjamin, voici le message de test dont nous avons parlé. <code><script>alert('hello')</script></code> . |
| Greiner Louis | Update : mot de passe chiffré | Bonjour Benjamin, les mots de passe ont bien été chiffrés dans la BDD. |
| Greiner Louis | Nouveaux clients | Bonjour Benjamin, nous avons deux nouveaux clients, M. Trump et Mme Merkel, auxquels il faudrait que tu souhaites la bienvenue. |

Accueil

Fig 4 : Messagerie (vw_messagerie.php)

FICHES CLIENTS

X Défaut ▼ Par nom ▼ Par prenom ▼ Par n° compte ▼ Nom, prenom, compte... 🔍

| ID | Nom | Prénom | N° Compte | Statut | Virement |
|----|---------|--------|------------|---------|---|
| 2 | Chen | Yuheng | 4964011994 | EMPLOYE |  |
| 3 | Greiner | Louis | 7873234879 | EMPLOYE |  |
| 4 | Merkel | Angela | 5789446841 | CLIENT |  |
| 5 | Trump | Donald | 6201640760 | CLIENT |  |

Accueil

Fig 5 : Fiches clients (vw_ficheclient.php)

Notre application doit gérer des données bancaires. Il s'agit donc de données extrêmement critiques, que nous devons protéger des attaquants. Lorsque l'on crée un site web naïvement, une multitude de failles de sécurité existent, plus ou moins facilement accessibles pour un individu mal intentionné. Dans ce rapport, nous allons lister plusieurs failles, et regarder les correctifs que nous avons mis en place pour empêcher quelqu'un de les exploiter.

Le champ d'action pour un attaquant est très large sur un site internet, les principaux types d'attaque sont:

- Injection SQL
- Cross-site scripting (XSS)
- Violation de contrôle d'accès
- Violation de gestion de session
- Falsification de requête (CSRF)
- Vulnérabilité d'un composant
- Chiffrement des données sensibles
- Accès aux répertoires par http
- Script de redirection

3.1 ~ Injection SQL ~

La première faille est l'injection SQL. Elle consiste à utiliser les champs de saisie de l'application pour modifier les requêtes SQL envoyées côté serveur. Un endroit où l'utilisateur pourrait (par exemple) exploiter cette faille est la page `vw_login.php`, avec le champ mot de passe :

Mot de passe

' or 'a'='a

Fig 6 : Exemple d'exploitation de l'injection SQL (`vw_login.php`)

En effet, il se pourrait que l'application recherche dans la base un utilisateur avec une requête telle que :

```
select * from users where login='$login' and mot_de_passe='$pwd'
```

Ici, le mot de passe `' or 'a' = 'a` serait concaténé avec la requête et la condition deviendrait vraie, ce qui permettrait à l'attaquant de se connecter en tant que le premier client de la BDD.

Pour corriger cette faille, nous avons mis en place deux correctifs :

- Nous avons d'abord échappé toutes les chaînes de caractères saisies par l'utilisateur avec `htmlspecialchars()`, qui remplace les caractères spéciaux tels que les quotes par des caractères html qui ne sont pas interprétés.

```
$login = htmlspecialchars($login, ENT_QUOTES);  
$pwd = htmlspecialchars($pwd, ENT_QUOTES);
```

- Nous avons aussi remplacé toutes les requêtes avec des paramètres par des `prepared statements`, plus sûrs :

```
$stmt = $mysqli->prepare("SELECT nom, prenom, mot_de_passe,  
    login, id_user, numero_compte, profil_user, solde_compte  
    FROM users  
    WHERE login=? AND mot_de_passe=?");  
$stmt->bind_param("ss", $login, $password);  
$stmt->execute();
```

De cette manière, l'attaquant ne peut plus interférer avec les requêtes SQL envoyées au serveur.

3.2 ~ Cross-site scripting (XSS) ~

La faille XSS consiste, pour un attaquant, à entrer un script dans un champ de saisie utilisateur, avec l'espoir qu'il soit exécuté sur la machine de sa victime. Dans notre application, cette faille pourrait par exemple être exploitée grâce aux messages. L'attaquant pourrait envoyer un message "piégé" (contenant un script) à sa victime, qui serait exécuté lorsque celle-ci ouvre sa messagerie :

Message

```
Bonjour, j'ai bien effectué le virement que vous  
m'aviez demandé la semaine dernière.  
<script>document.location="http://www.sitefrauduleu  
x.com";</script>
```

Fig 6 : Exemple d'exploitation d'une faille XSS

Pour empêcher un attaquant d'exploiter cette faille, nous avons parsé le sujet et le corps du message qu'il envoie avec `htmlspecialchars()`, afin, notamment, de remplacer les quotes et les chevrons par des caractères html que nous pouvons stocker sans danger dans la BDD. Par mesure de précaution, les messages sont à nouveau parsés avant d'être affichés dans la messagerie :

```
echo '<td>' . htmlspecialchars($message['sujet_msg'], ENT_QUOTES) . '</td>';  
echo '<td>' . htmlspecialchars($message['corps_msg'], ENT_QUOTES) . '</td>';
```

Ainsi le script JS ne sera pas exécuté mais sera affiché directement :

| Expéditeur | Sujet | Message |
|-------------|-------------|--|
| Toto benoit | hello | hello |
| Tata Alice | XSS attack! | <script>alert("réfléchi XSS attaque")</script> |
| Toto benoit | hello | |

Fig 7 : faille XSS après correction

Notons que pour les virements, la restriction est encore plus forte, car nous vérifions que le montant saisi par l'utilisateur est un nombre avec :

```
is_numeric($total)
```


3.3 ~ Violation de contrôle d'accès ~

La violation de contrôle d'accès se produit lorsqu'un utilisateur accède à des informations ou des pages auxquelles il n'a pas le droit. Cela se produit lorsque l'on fait passer dans l'URL par exemple, à travers le champ requête, des données critiques, qui ne sont par la suite pas vérifiées. L'attaquant peut avoir accès à des données auxquelles il n'a pas l'autorisation, indépendamment de la session dans laquelle il est connecté.

Cette faille peut intervenir dans un schéma de récupération de données comme celui-ci. Naïvement, un lien d'accès vers la messagerie d'un compte particulier (en l'occurrence celui de la session active) peut être implémenté de cette manière:

```
<p><a href="myController.php?action=msglist&userid=<?php echo  
$_SESSION["connected_user"]["id_user"];?>" target="_blank">Mes messages  
reçus</a></p>
```

Cependant, il est très simple de copier cette même URL, et de changer le userid écrit en clair dans l'URL, et qui, si cela n'est pas vérifié par le contrôleur (comme ci-dessous), ouvre une énorme faille de sécurité (récupération de messages, voire pire) :

```
<?php  
$_SESSION['messagesRecus'] = findMessagesInbox($_REQUEST["userid"]);  
?>
```

En procédant ainsi, les messages reçus sont ceux liés au compte dont le userid est celui passé en paramètre dans l'URL.

Une protection simple et efficace contre ce genre d'attaque est d'éviter à tout prix le passage en argument dans l'URL, et de fonctionner au maximum directement avec les variables de session. Remplacer par exemple les champs ci-dessus par:

```
<p><a href="myController.php?action=msglist" target="_blank">Mes  
messages reçus</a></p>
```

```
<?php  
$_SESSION['messagesRecus'] = findMessagesInbox($_SESSION["userid"]);  
?>
```

Une protection supplémentaire peut être implémentée, relative au contrôle de session, dès que l'on charge une page pour l'utilisateur. On vérifie que l'utilisateur est bien connecté, sinon on renvoie à la page de login.

```
if(!(isset($_SESSION['connected_user']))) {  
    // pas de session active = d'utilisateur déjà connecté  
    header('Location: vw_login.php');  
}
```

Cela permet de réduire encore un peu plus le champ d'action de l'attaquant, l'obligeant à être connecté pour avoir accès aux autres pages.

Suivant le même principe, on peut, selon si on est Employé ou Client, avoir accès à différentes ressources (ici, avoir accès à la vue `vw_ficheclient.php`):

```
if(!isset($_SESSION['connected_user'])) ||  
$_SESSION['connected_user']['profil_user'] != "EMPLOYE"){  
    // pas de session active = d'utilisateur déjà connecté et user =  
admin  
    header('Location: vw_moncompte.php');  
}
```

Un autre axe d'attaque, lorsqu'un attaquant veut violer un accès, peut être de deviner le mot de passe d'un compte. A l'aide de dictionnaires, il est aujourd'hui relativement simple de "brut forcer" (essayer un nombre très important de fois avec des mots de passe différents jusqu'à réussir à se connecter) un mot de passe si celui-ci n'est pas très fort.

Pour prévenir cela, nous avons implémenté au niveau de la base de données une table qui enregistre les adresses IP qui se trompent lors de la connexion. Une même adresse IP qui se trompera 5 fois en l'espace de 5 minutes sera bloquée temporairement avant de pouvoir recommencer.

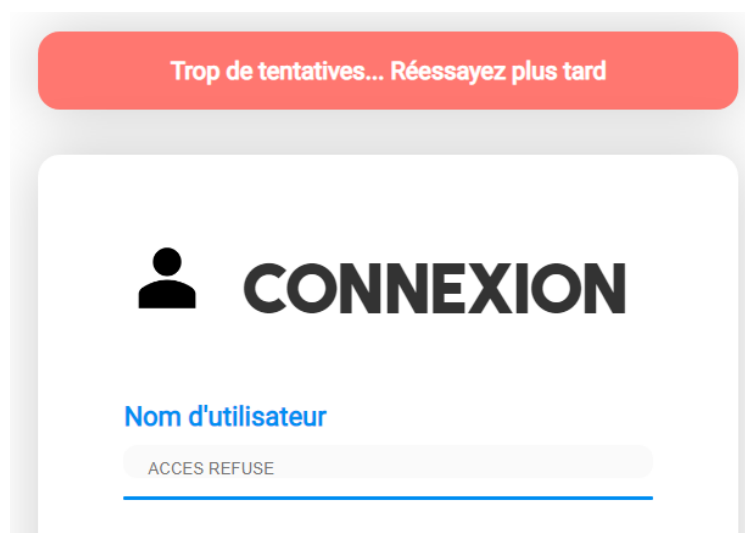


Fig 8 : Adresse IP bloquée suite à 5 tentatives de connexion (`vw_login.php`)

En plus de ces lignes de défense, l'utilisation de reCAPTCHA, un système de détection automatisée d'utilisateurs appartenant à Google, lors de la connexion, permet de filtrer un peu plus les tentatives de connexion. Ce test permet de vérifier que l'utilisateur n'est pas un robot.

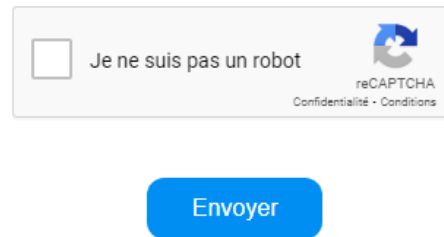


Fig 9 : reCAPTCHA obligatoire lors de la connexion (vw_login.php)

3.4 ~ Violation de gestion de session ~

La violation de gestion de session est une technique consistant à voler la session d'un utilisateur connecté pour effectuer des actions à sa place.

La façon la plus simple d'exploiter cette faille serait d'utiliser l'ordinateur d'un collègue qui aurait oublié de se déconnecter. Pour remédier à cela, nous avons d'abord limité la durée de la session à 10 minutes (car l'application est critique), pour qu'aucune action ne soit possible passé ce laps de temps :

```
if(time()-$_SESSION["login_time"] > 600) //drop session après 10min
{
    session_unset();
    session_destroy();
    header("Location:vw_login.php?timeout");
}
```

Aussi, nous nous sommes assurés que même si l'attaquant arrive à s'emparer de la session pendant ce laps de temps, il ne puisse pas effectuer de virement. Pour cela, nous avons simplement ajouté une vérification du mot de passe dans la page vw_virement.php.

Montant à transférer

| | | |
|-----|---|----|
| 135 | € | 47 |
|-----|---|----|

Mot de passe

Fig 10 : Confirmation du mot de passe avant virement (vw_virement.php)

Enfin, un attaquant pourrait aussi essayer de voler le cookie de session de sa victime avec un script XSS ou par l'URL. Bien que nous ayons déjà appliqué les correctifs nécessaires pour XSS, nous avons renforcé cette protection en empêchant les scripts de lire le cookie de session (avec `httponly`). Nous avons aussi empêché le cookie de session de passer par l'URL, grâce à `use_only_cookies` dans `include.php`, puis en incluant ce fichier à chaque `session_start()`.

```
<?php
    ini_set('session.cookie_httponly', 1); // interdire la lecture du cookie de
session avec un script
    ini_set('session.use_only_cookies', 1); // interdire le cookie de session via
l'URL (uniquement par cookie)
?>
```

Une fois `httponly` activé, `PHPSESSID` ne peut plus être obtenu à partir de scripts externes :

| Name | Value | Domain | Path | Expires / Max... | Size | HttpOnly | Se... | Sa... | Sa... | Priority |
|-----------|------------|-----------|------|------------------|------|----------|-------|-------|-------|----------|
| PHPSESSID | bvinmjm... | localhost | / | Session | 35 | ✓ | | | | Medium |

Avant d'activer `httponly` :

```
> document.cookie
< "PHPSESSID=g4kp6ii9qdq9koar9uju70d4ji"
```

Après avoir activé `httponly` :

```
> document.cookie
< ""
```

3.5 ~ Falsification de requête (CSRF) ~

La falsification de requêtes consiste à faire faire à une victime des actions à son insu, ou éventuellement des actions nécessitant des privilèges que l'attaquant n'aurait pas. L'action qui paraît être à risque ici est le virement. Un attaquant pourrait d'abord tenter d'utiliser XSS pour que sa victime fasse un virement sans s'en apercevoir, mais nous avons déjà appliqué les correctifs nécessaires dans la partie 3.2. Néanmoins, nous devons préparer un correctif si jamais l'attaquant réussit quand même à déclencher un virement.

Pour cela, nous avons ajouté un jeton de validation à l'opération de virement. Ce jeton est généré aléatoirement à chaque fois qu'un virement est effectué, puis placé dans la session. Le contrôleur peut alors vérifier la cohérence du jeton, et, si ce jeton n'est pas rempli ou

incorrect, cela signifie que le virement n'a pas été déclenché depuis la page `vw_virement.php`, et nous devons alors annuler l'opération :

1/ Génération du token

```
$mytoken = bin2hex(random_bytes(128)); // token qui va servir à prévenir des
attaques CSRF
```

```
$_SESSION["mytoken"] = $mytoken;
```

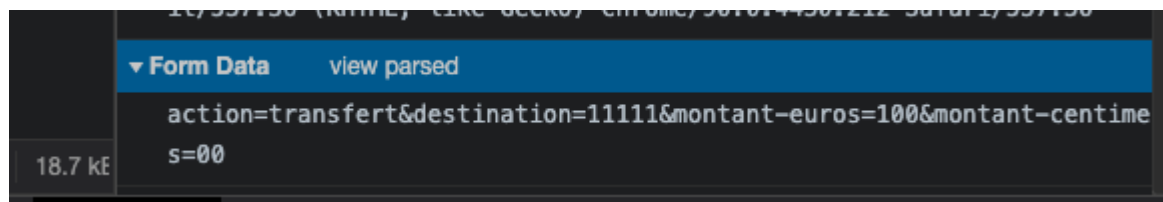
```
<input type="hidden" name="mytoken" value="<?php echo $mytoken; ?>">
```

2/ Vérification du token dans le contrôleur

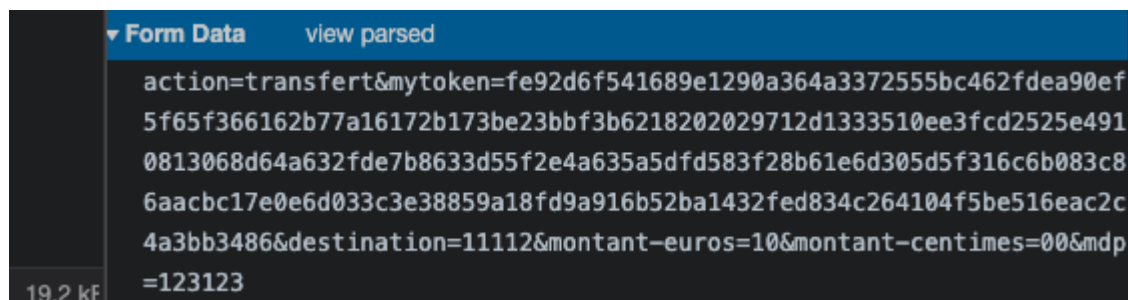
```
if (!isset($_REQUEST['mytoken']) || $_REQUEST['mytoken'] != $_SESSION['mytoken']) {
    // echec vérification du token (ex : attaque CSRF)
    $url_redirect = "vw_moncompte.php?err_token";
}
```

De cette manière, les pirates ne peuvent pas effectuer d'attaques CSRF en modifiant les paramètres de l'action.

Avant utiliser le token :



Après utiliser le token:



De plus, comme expliqué en 3.4, l'utilisateur doit également saisir son mot de passe pour confirmer le virement. Ce n'est que lorsque le mot de passe est correct que le virement peut être effectué.



Fig 11 : Vérification du mot de passe avant virement (vw_virement.php)

3.6 ~ Vulnérabilité d'un composant ~

Parfois les failles ne sont pas dans l'application en elle-même mais dans les éléments qui la font tourner. Pour se prémunir de certains problèmes liés à la vulnérabilité des composants, nous avons d'abord modifié la gestion par défaut des messages d'erreur en empêchant leur affichage dans le navigateur :

```
ini_set('display_errors', 0);
```

On a également changé le login/mot de passe par défaut de notre environnement (et nous avons donc supprimé l'utilisateur root, dont les identifiants sont faciles à deviner).

3.7 ~ Chiffrement des données sensibles ~

HTTPS (Hypertext Transfer Protocol Secure ou protocole de transfert hypertexte sécurisé) est un protocole de communication Internet qui protège l'intégrité ainsi que la confidentialité des données lors du transfert d'informations entre l'ordinateur de l'utilisateur et le site web. En bref, HTTP garanti aux données d'être chiffrées (personne ne peut "écouter" ou voler des données sur le chemin utilisateur - serveur site web), leur intégrité (ni modifiées ni corrompues durant le transfert), et l'authentification (preuve qu'un utilisateur navigue avec le bon site web (protège notamment des attaques man-in-the-middle)). Son efficacité réside dans la création de certificats SSL (Secure Sockets Layer), preuve d'authenticité des sites utilisant HTTPS.

Un bon entraînement à l'implémentation de HTTPS peut être de suivre ce guide rapide (pour Mac):

<https://www.freecodecamp.org/news/how-to-get-https-working-on-your-local-development-environment-in-5-minutes-7af615770eec/>

Ainsi, on aura en local un certificat autosigné qui atteste de l'authenticité de notre site web. Pour l'implémentation en ligne, nous recommandons ce guide qui offre une vue assez claire de HTTPS:

<https://angleweb.fr/blog/guide-http-vers-https/>

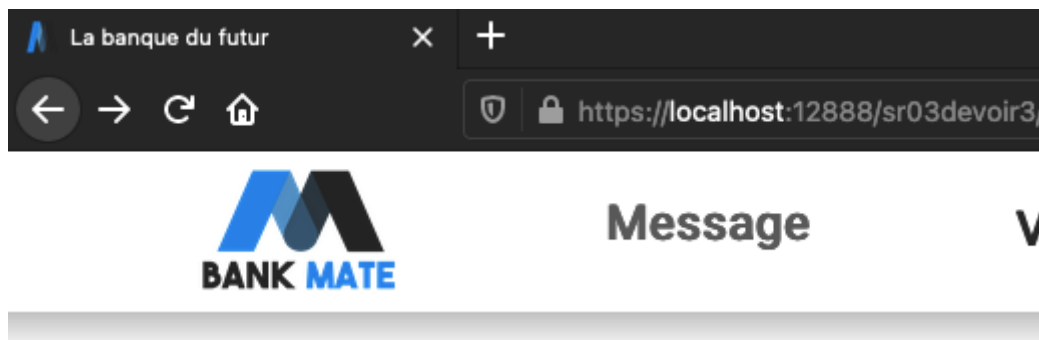


Fig 12 : Accès au site en https

En plus de cela, les données critiques (mot de passe par exemple) ne doivent pas naviguer "en clair" une fois le formulaire envoyé par l'utilisateur. De plus, il est dangereux de stocker des données critiques non chiffrées dans une base de données. Si la base de données vient à être piratée, c'est les données de tous les utilisateurs qui seraient récupérées. Pour ce faire, dans notre cas, les mots de passe sont en permanence hachés.

En PHP, diverses fonctions existent pour hasher une string, comme:

```
password_hash("motdepasse1234", PASSWORD_DEFAULT)
```

Lors de l'authentification, on va donc hacher l'entrée de l'utilisateur et comparer le résultat avec nos hash dans la base de données.

```
if (password_verify($pwd, $unUser['mot_de_passe'])) {
```

3.8 ~ Accès aux répertoires par HTTP ~

Par défaut, http laisse accès à l'ensemble des fichiers (en particulier le fichier config/config.php) du projet. Ainsi, si l'utilisateur essaie d'accéder à l'url suivante :

http://localhost:12888/sr03devoir3_chen_greiner_missaoui/config/

Il aura un accès direct au fichier de configuration, ce qui est évidemment catastrophique :

Index of /sr03devoir3_chen_greiner_missaoui/config

- [Parent Directory](#)
- [config.php](#)

Fig 13 : Accès au fichier de configuration

Pour empêcher l'utilisateur d'accéder à cette url, il suffit d'ajouter dans l'arborescence un fichier `.htaccess`, avec l'option :

Options -Indexes

Ainsi, à la même url, il recevra cette fois une erreur :

Forbidden

You don't have permission to access this resource.

Fig 14 : Erreur d'accès au fichier de configuration grâce au `.htaccess`

3.9 ~ Scripts de redirection ~

Pour éviter à l'attaquant d'utiliser un script pouvant rediriger vers son site frauduleux, toutes les redirections ont été écrites en dur, dans le contrôleur, par exemple :

```
else if ($_REQUEST['action'] == 'disconnect') {  
    /* ===== DISCONNECT ===== */  
    unset($_SESSION["connected_user"]);  
    $url_redirect = "vw_login.php?disconnect";  
}
```

Fig 15 : Écriture des redirections en dur dans le contrôleur