



ANALYSE DE SECURITE

Application de messagerie

Projet 2

Analyse de sécurité de l'Application de messagerie développée dans le projet 1
dans le contexte du cours STI à l'HEIG-VD

Doriane Kaffo – Baptiste Hardrick

Table des matières

Table des matières	1
Introduction	2
Description du système	2
Éléments du système	3
Rôles des utilisateurs	3
Hypothèses de sécurité	4
Exigences de sécurité	4
Data flow Diagram	5
Sources de menaces	7
Scénarios d'attaques	7
Scénario 1: Accès non autorisé au service	7
Scénario 2: Accès non autorisé (2)	9
Scénario 3 : Attaque CSRF	10
Scénario 4: Attaque basée sur les contrôles d'accès	11
Scénario 5 : Attaque par DDoS	12
STRIDE	12
Identification des contre-mesures	13
Contre-mesures pour le scénario 1	13
Contre-mesures pour le scénario 2	13
Contre-mesures pour le scénario 3	14
Contre-mesures pour le scénario 4	15
Contre-mesures pour le scénario 5	15
Conclusion	15

Introduction

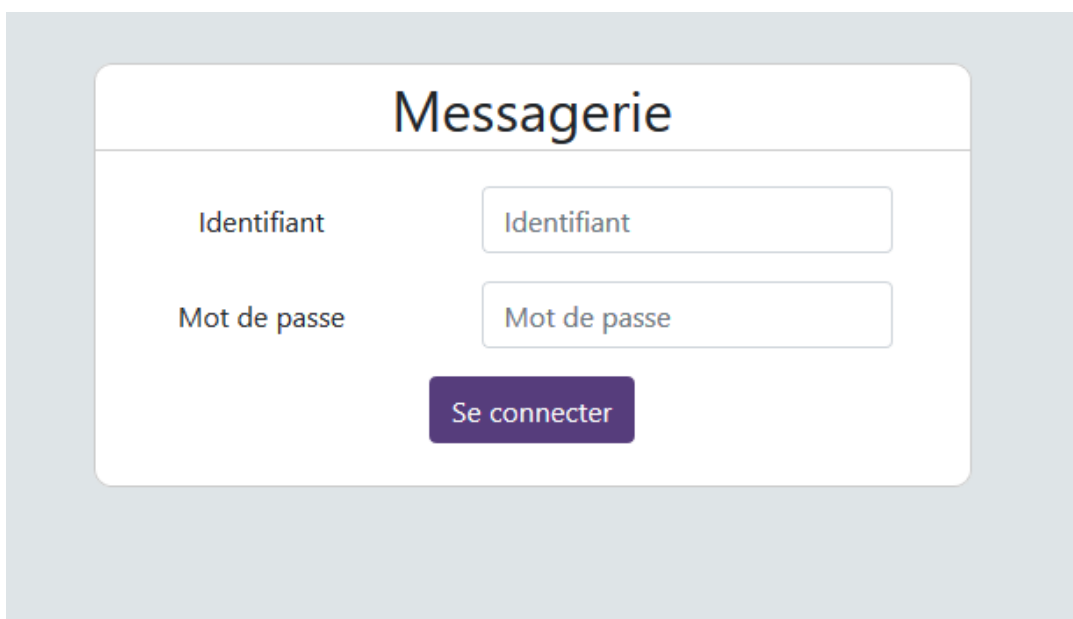
Dans la première partie du projet, il s'agissait de concevoir et réaliser une application web offrant un service de messagerie basique en ligne entre divers collaborateurs. De plus, cette application devait offrir des fonctionnalités de gestion afin que des administrateurs puissent performer des actions telles que l'ajout/suppression d'utilisateurs ou encore la modification de leurs informations (rôle, mot de passe). L'application a été réalisée en PHP et utilise une base de données SQLite.

Dans cette deuxième partie, il est cette fois ci question de l'amélioration de la sécurité de l'application étant donné que cet aspect n'avait pas été pris en compte jusqu'ici. Nous allons donc présenter dans ce document les différentes vulnérabilités que présentent l'application puis les contre-mesures que nous aurons mises en place.

Description du système

Le système est composé d'un serveur d'application Nginx exécutant PHP version 5.5.9 et d'une base de données SQLite. L'accès à l'application web se fait en HTTP via une page de connexion exigeant au visiteur:

- un nom d'utilisateur (login)
- un mot de passe



Messagerie

Identifiant

Mot de passe

Se connecter

Si le visiteur fournit des informations valides, il accède directement à sa boîte de réception. En fonction du rôle de celui-ci (administrateur ou pas), il aura des fonctionnalités en plus de ceux d'une messagerie de base à savoir :

- Lister tous les utilisateurs
- Modifier un utilisateur
- Supprimer un utilisateur

L'utilisateur courant est enregistré dans une variable de session sur le serveur et lors de la déconnexion de celui-ci, cette variable est supprimée.

Éléments du système

- La base de données des collaborateurs car elle contient des informations appartenant à l'entreprise et ses employés ce qui touche à la sphère privée. On doit assurer la confidentialité, l'intégrité et la disponibilité des données qu'elle renferme.
- La base de données des messages car ces derniers sont confidentiels entre les participants à une même discussion. On doit assurer la confidentialité, l'intégrité et la disponibilité des données qu'elle renferme
- L'infrastructure : La disponibilité du service de l'application web

Rôles des utilisateurs

- Anonyme : Ne peut qu'avoir accès à la page de login
- Collaborateur :
 - Lire ses messages reçus
 - Ecrire un nouveau message
 - Changer son mot de passe
- Administrateur :
 - Faire ce que peut faire un collaborateur
 - Ajouter/modifier/supprimer un utilisateur

Hypothèses de sécurité

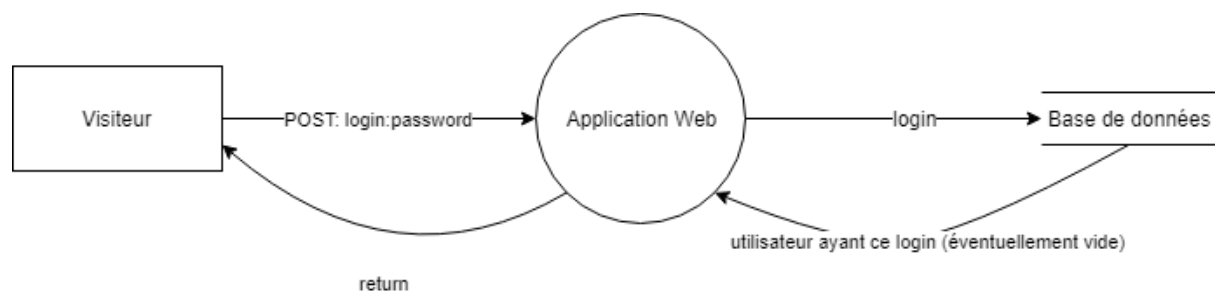
Nous estimons que seuls les administrateurs et le système d'exploitation du serveur et de la base de données sont dignes de confiance.

Exigences de sécurité

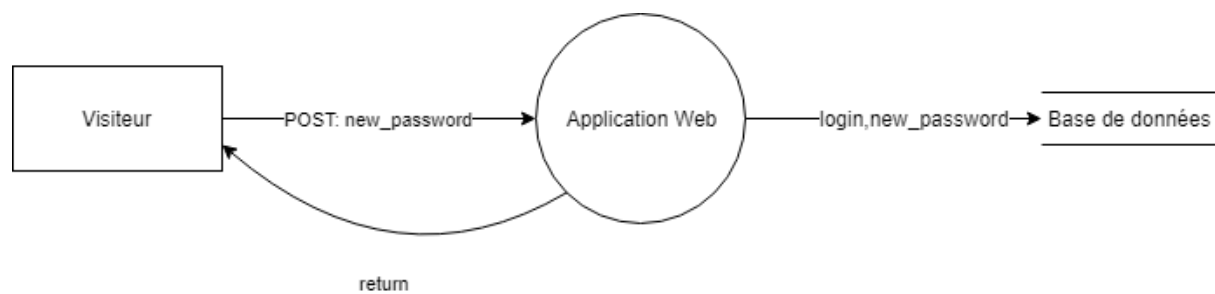
- Les fonctions réservées aux administrateurs ne devront être accessibles que par ceux-ci.
- Seuls les utilisateurs authentifiés doivent avoir accès à l'application.
- Les données des utilisateurs doivent être protégées (confidentialité).
- Garantir au maximum la disponibilité de l'application
- Assurer l'intégrité des données enregistrées sur la base de données.
- Minimiser l'impact en cas d'intrusion dans le système.
- Les messages enregistrés sont non modifiables
- Le contenu des messages ne doit pas être accessible en dehors des utilisateurs participant à la conversation

Data flow Diagram

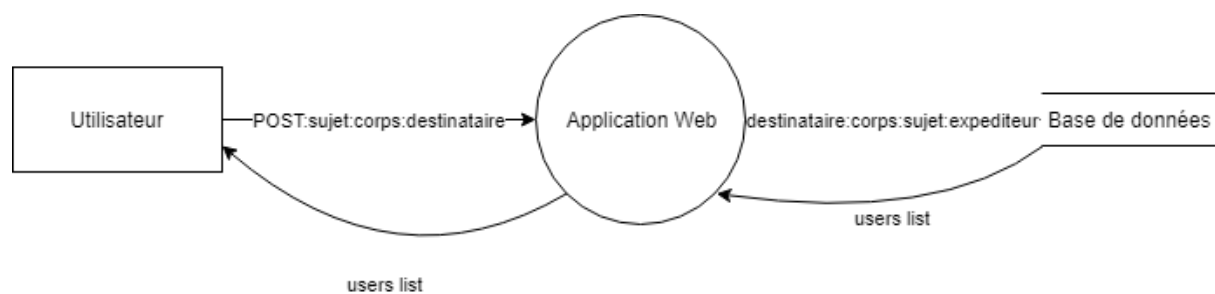
Connexion:



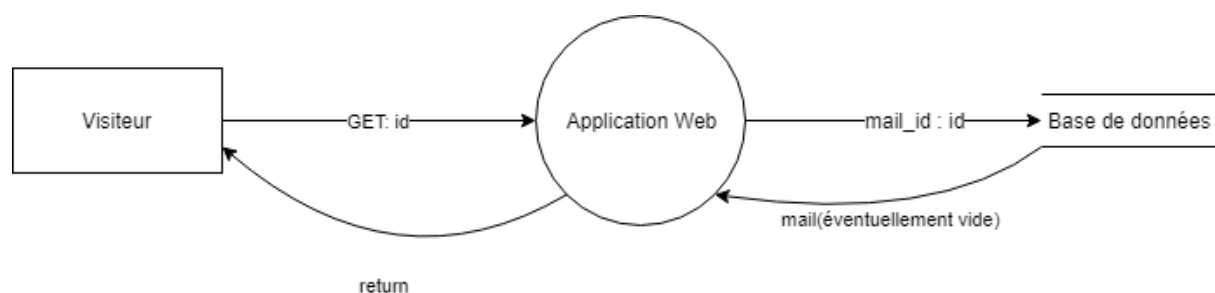
Change password:



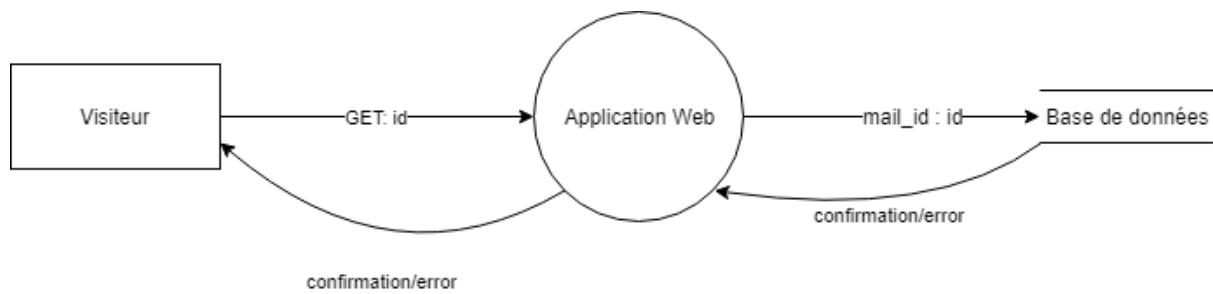
Ecrire un message:



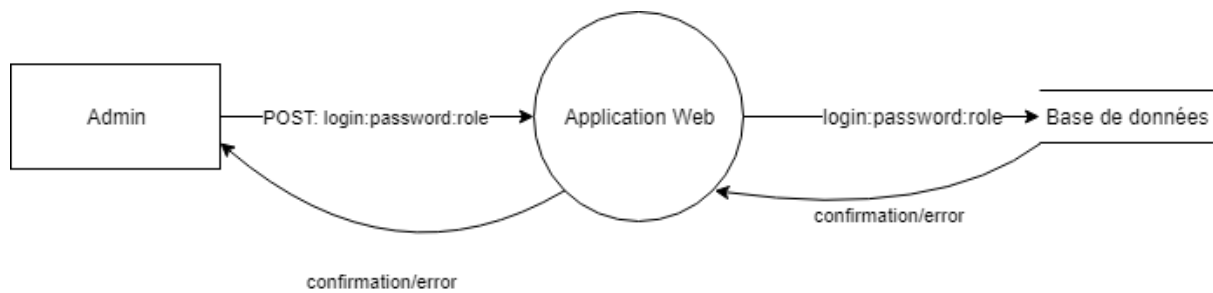
Lire un message :



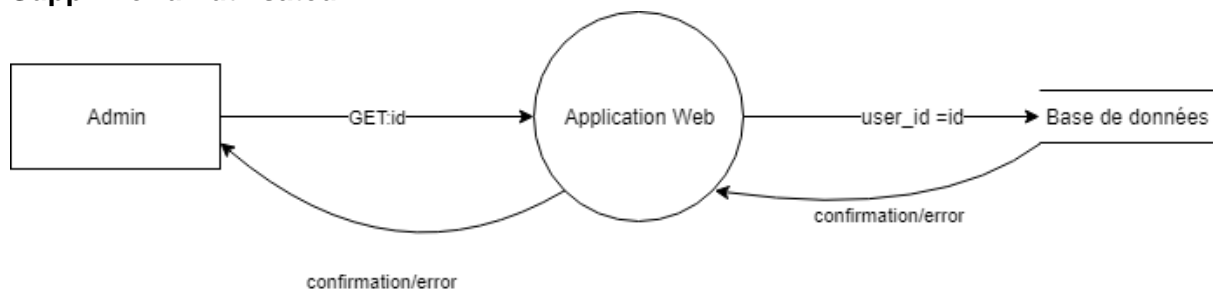
Supprimer un message:



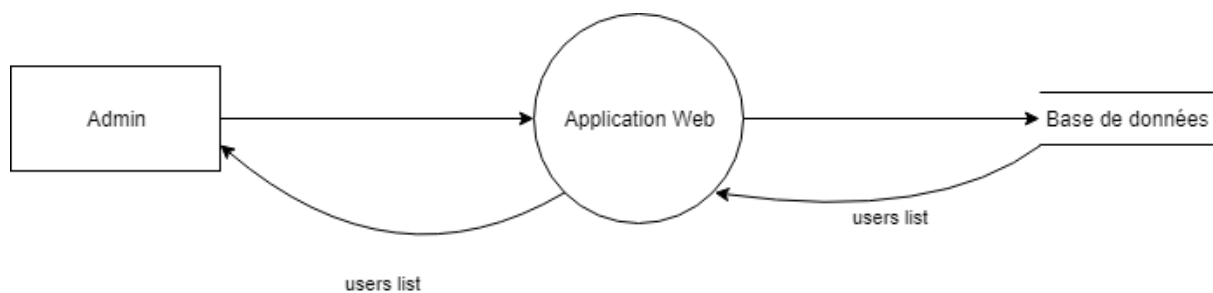
Ajouter un utilisateur/modifier:



Supprimer un utilisateur :



Liste des utilisateurs :



Dans les cas où les acteurs sont des utilisateurs ou des visiteurs, la *trust-boundary* se trouve entre l'acteur et l'application web. Les administrateurs font partie des entités *trusted* .comme défini dans l'hypothèse de sécurité.

Sources de menaces

- Collaborateurs malicieux:
 - Motivation : accès aux fonctionnalités administrateurs, voire les conversations de ses autres collaborateurs, supprimer ses conversations (répudiation) où celles de ses collègues, causer du tort à l'entreprise (sabotage)
 - Cible : base de données, accès aux fonctionnalités
 - Potentialité : moyenne
- Concurrent :
 - Motivation : espionnage, sabotage, désinformation, nuire à la réputation
 - Cible: base de données, données utilisateurs, infrastructure
 - Potentialité : moyenne
- Cybercrime :
 - Motivation : Revente d'informations, chantage via les données collectées sur les messages, rançon contre rétablissement des services
 - Cible : base de données, informations utilisateurs, infrastructure
 - Potentialité : moyenne

Scénarios d'attaques

Scénario 1: Accès non autorisé au service

- Source de menace : Cybercriminel/Concurrent
- Cible : Compte utilisateur
- Déroulement :

Nous nous mettons à la place de la source de menace. Le formulaire d'authentification présente deux champs. Il serait donc intéressant de voir si ceux-ci sont vulnérables à une **injection arbitraire de code SQL**.

L'on se doute que la requête de vérification est de la forme suivante :

```
SELECT * FROM user WHERE login = '+champ_identifiant+' AND  
pass='+champ_motdepasse'
```

Pour vérifier la vulnérabilité par injection SQL, on va utiliser comme entrée **hack'** et **hack** (cela donnera `SELECT * FROM user WHERE login='hack' AND pass='hack';`, c'est à dire une requête incorrecte et donc une erreur)

Voyons ce qu'on obtient:



On a bien un message d'erreur, ce qui indique une vulnérabilité par injection SQL. De plus le message d'erreur apporte une hypothèse sur l'implémentation de l'authentification. En effet, on remarque l'utilisation d'une fonction `findByPseudo('utilisateur', 'hack')`; ceci laisse croire au fait qu'une vérification est d'abord faite sur l'existence du pseudonyme puis une vérification sur la correspondance des mots de passe.

Ainsi les requêtes du type suivant :

```
SELECT * FROM user WHERE login = 'hack' OR 1=1--' AND
pass='"+champ_motdepasse
```

Ne marcheront pas, puisqu'une vérification est ensuite faite en interne sur le mot de passe fourni par l'utilisateur et le mot de passe obtenu de la base de données.

Malgré cet échec, on est néanmoins sûr d'une chose ; la vérification du mot de passe est censée être faite sur au plus un enregistrement de la table (les pseudos sont uniques, soit-il y en a un, soit-il n'y en a pas).

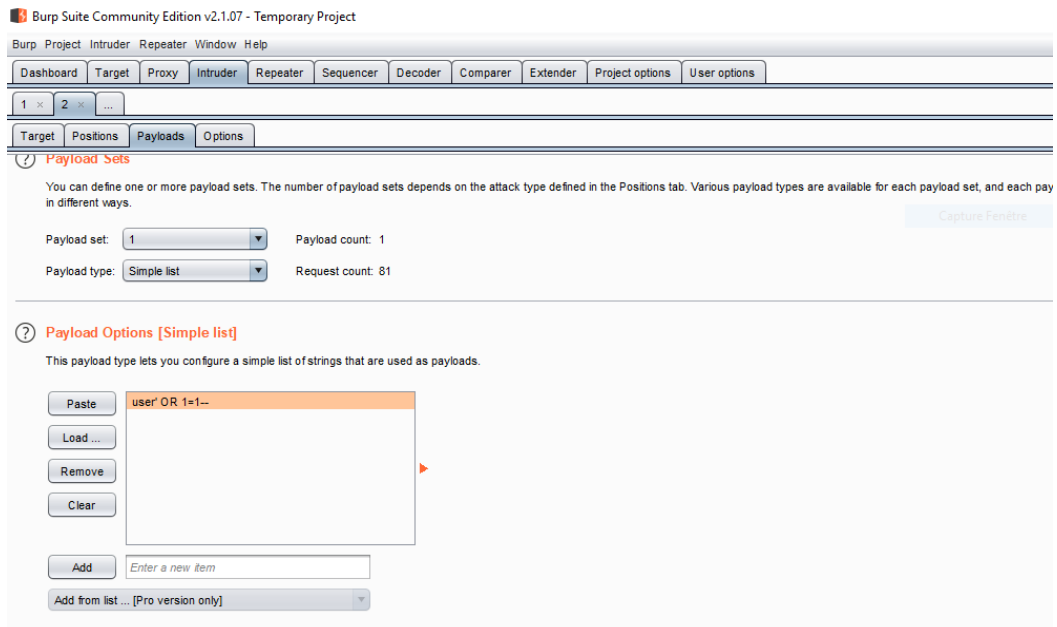
Avec la requête :

```
SELECT * FROM user WHERE login = 'hack' OR 1=1--' AND
pass='"+champ_motdepasse
```

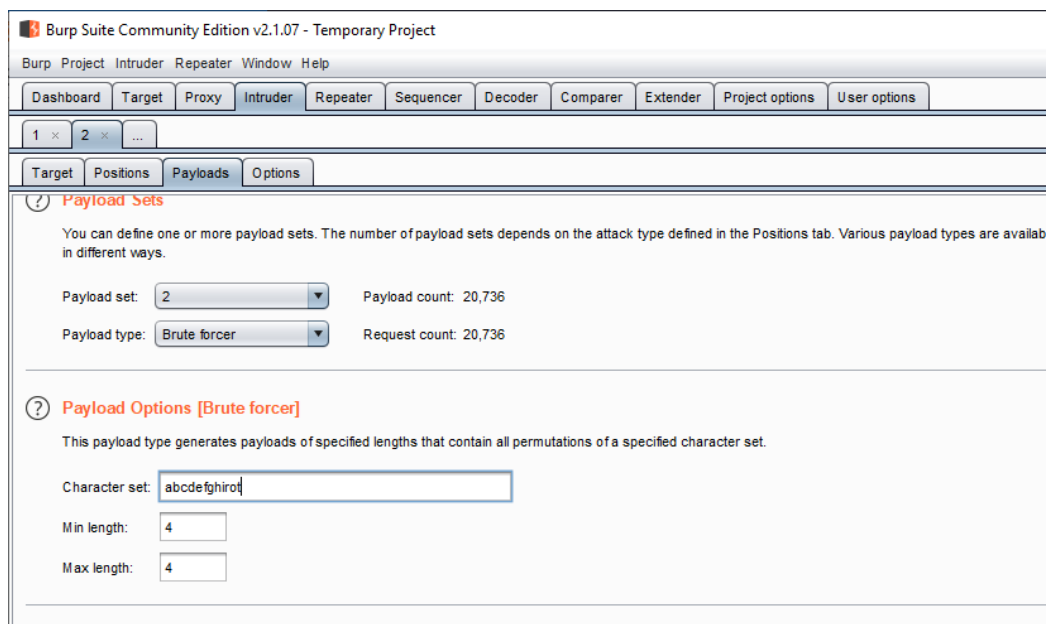
Tous les enregistrements sont retournés, il nous faut juste le mot de passe du premier enregistrement en espérant qu'aucune vérification n'est faite quant aux tailles des résultats de requêtes.

Pour trouver le mot de passe, on va procéder par brute-force :

On va utiliser le module Intruder du logiciel Burp Suite et effectuer le brute force :



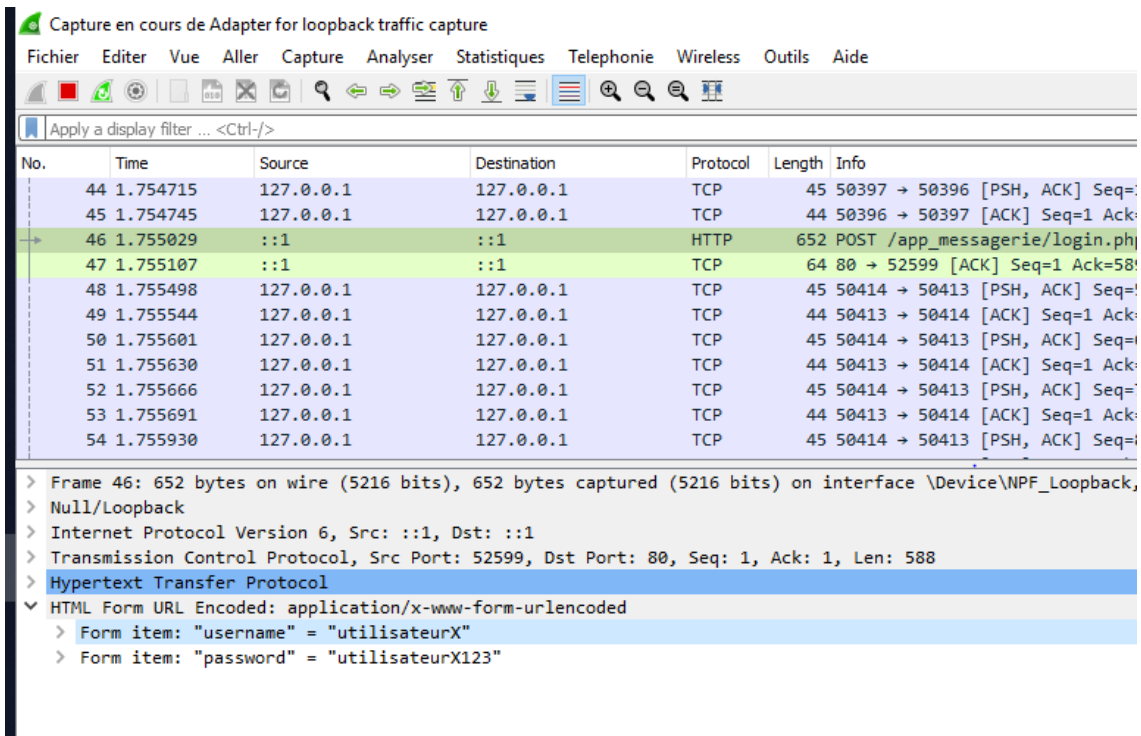
On définit comme nom d'utilisateur `user' OR 1=1--` et on va « bruteforcer » sur le mot de passe:



Et on finit par aboutir au mot de passe : “root”

Scénario 2: Accès non autorisé (2)

- Source de menace : Cybercriminel/collaborateur malin
- Cible : Compte utilisateur
- Déroulement :
L'attaquant va mettre sur écoute un utilisateur légitime pendant qu'il se connecte sur l'application en utilisant un outils tel que WireShark



Le mot de passe est transmis en clair, l'attaquant n'aura qu'à les utiliser

Scénario 3 : Attaque CSRF

- Source de menace : Collaborateur malin
- Cible : compte administrateur
- Déroulement :

Alice et John sont des associés ayant chacun un compte sur l'application. Bob est un administrateur. Alice voudrait supprimer le compte de John mais ne peut le faire car cette action est réservée aux administrateurs grâce au cloisonnement qu'on mis en place précédemment. Néanmoins, elle connaît le lien pour performer cette action :

/delete-user.php?id=5

Elle va envoyer un mail à John contenant un lien vers une page web qu'elle aura préalablement conçue :

```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <meta http-equiv="X-UA-Compatible" content="ie=edge">
7   <title>Document</title>
8 </head>
9 <body>
10  
11 </body>
12 </html>

```

Check this awesome page!

Check this [page](#)

La balise img va provoquer la visite du lien `/delete-user.php?id=5`
Bob en cliquant ce lien exécutera l'action de supprimer l'utilisateur John malgré lui.
Alice vient de performer une attaque **CSRF**.

Scénario 4: Attaque basée sur les contrôles d'accès

- Source de menace: Collaborateur malicieux/Cybercriminel
- Cible : base de données, comptes utilisateurs
- Déroulement :
Alice est une associée ayant un compte sur l'application, et qu'elle réussit à obtenir le lien (normalement accessible aux administrateurs) permettant de lister les informations sur les utilisateurs (par espionnage par exemple) :

`/list-user.php`

En utilisant le lien elle a accès à la liste des utilisateurs.

Messagerie

Liste des utilisateurs

Utilisateur: Alice

Nouveau message

Boîte de réception

Changer son mot de passe

Se déconnecter

Liste des utilisateurs

Id	Pseudo	Mot de Passe (Hash)	Admin	Action
2	root2	\$2y\$10\$p/yfOBhju7zkaBIFTbzEA.OPNya7lgaqufQt8zd7uzjIWobMjv52m	Oui	<button>Modifier</button> <button>Supprimer</button>
3	utilisateurX	\$2y\$10\$7grZdRmszUZ4gGZ.7AEqleLugjzRiFx1baFiM8S5cqMWAQXq25X0e	Non	<button>Modifier</button> <button>Supprimer</button>
4	Alice	\$2y\$10\$ExaC2/4qgAubtmo2Zshl3ukH6W0h17HEPkQoC6oZEtsP54oYupFZm	Non	<button>Modifier</button> <button>Supprimer</button>

Scénario 5 : Attaque par DDoS

- Source de Menace : Cybercriminel
- Cible : La disponibilité du serveur Web
- Déroulement :

L'attaquant bombarde le site de requêtes depuis une ou plusieurs machines. En cas de réussite le serveur surchargé ne sera plus capable de répondre aux requêtes des autres utilisateurs.

STRIDE

- Spoofing
 - Utilisation d'un mot de passe volé (scénario 1)
 - Contre mesure : Gestion de la page d'erreur, meilleurs mots de passe, gestion des inputs utilisateur (détaillé dans les contre-mesures du scénario 1)
- Tampering
 - Injections SQL
 - Contre mesure : Assainir les entrées utilisateur (détaillé dans les contre-mesures du scénario 1)
- Repudation
 - Utilisation d'un mot de passe volé
 - Contre mesure : Double authentification, envoi de certification sur une boîte mail (contre mesure non détaillée dans les scénarios)
- Information disclosure
 - Interception de messages sur un réseau
 - Contre mesure : chiffrer les messages (contre mesure du scénario 2)
- Denial of service
 - DDoS sur le serveur
 - Contre mesure : bannir les IP demandant trop de requêtes côté serveur (contre mesure détaillée dans le scénario 5)
- Elevation of privileges
 - Se faire passer pour un administrateur via les l'url (scénario 4) ou attaque CSRF (scénario 3)
 - Contre mesure : token anti CSRF (contre mesure 3) et contrôle d'accès sur les listes (contre-mesure scénario 4)

Identification des contre-mesures

Contre-mesures pour le scénario 1

Trois vulnérabilités ont été exploitées :

- Aucune page dédiée à la gestion des erreurs
- Aucune vérification sur les entrées de l'utilisateur (injection SQL et XSS)
- Faiblesse du mot de passe

Gestion des erreurs

Il faut créer une page dédiée aux exceptions que pourraient lever l'application afin de masquer les informations liées à sa structure. Pour cela, on va créer une page **error.php** et ajouter un bloc **try/catch** aux zones sensibles.

Gestion des entrées

Pour cela, on va utiliser des prepared statement afin de séparer le traitement de la requête de celui des paramètres. Les paramètres seront envoyées sous forme de chaîne à la requête qui aura été au préalable pré-compilée. Par exemple : 'hack OR 1=1' sera envoyé comme étant : " 'hack OR 1=1' ".

On va également utiliser la fonction `htmlspecialchars` qui convertit les caractères spéciaux en entité HTML.

Gestion du mot de passe

Un autre point critique est la faiblesse des mots de passe. En effet, une attaque par brute-force multi-threadé est très efficace contre la forme des mots de passe actuels. Pour palier à cela, on pourra définir une politique quant aux mots de passe des utilisateurs:

- Au moins 8 caractères
- Contenir au moins 1 chiffre
- Contenir au moins un caractère spécial : `!"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~`

Contre-mesures pour le scénario 2

Pour prévenir le sniffing sur les échanges entre les utilisateurs légitimes entre le client et le serveur web on peut utiliser du HTTPS pour chiffrer les échanges entre le client et le serveur en obtenant un certificat auprès de tiers tels que **Let's Encrypt**. Dans le cadre de cet exercice nous ne le ferons pas car cela amène des contraintes qui vont au-delà de l'exercice.

Contre-mesures pour le scénario 3

Cette attaque est basée sur la confiance qu'accorde l'application à Bob. Même s'il l'a fait inconsciemment, Bob a supprimé John parce qu'il en avait le droit. Pour parer ce genre d'attaque nous allons implémenter les tokens anti-csrf. Les actions critiques de l'application devront être soumises à une validation grâce à un token anti-csrf préalablement construit par le serveur.

- Pour les liens ayant des paramètres GET, le token sera généré aléatoirement par le serveur et ajouté au lien. Au moment du clic sur le lien, le précédent token sera également envoyé en paramètre afin de valider la requête.
- Pour les formulaires (POST), le token sera cette fois-ci placée dans un champ masqué et soumis en même temps que le formulaire.

Une manière possible d'implémenter aléatoirement les tokens :

```
csrf-get-type-token.php ×
csrf-get-type-token.php
1 <?php
2     $length = 32;
3     $_SESSION['get-token'] =
4     substr(base_convert(sha1(uniqid(mt_rand())), 16, 36), 0, $length);
5 ?>
```

Pour les formulaires, le token sera dans un champ masqué :

```
<form action="add-user.php" method="POST">
  <?php include $_SERVER['DOCUMENT_ROOT'].'/app_messagerie/csrf-post-type-token.php'
  <input type="hidden" name="token" value="<?=$_SESSION["post-token"]?>">
```

Qui sera ensuite vérifié une fois le formulaire envoyé :

```
if(isset($_POST["token"]) && $_POST["token"]==$_SESSION["post-token"]){
```

Pour les paramètres GET, le token sera ajouté sur le lien :

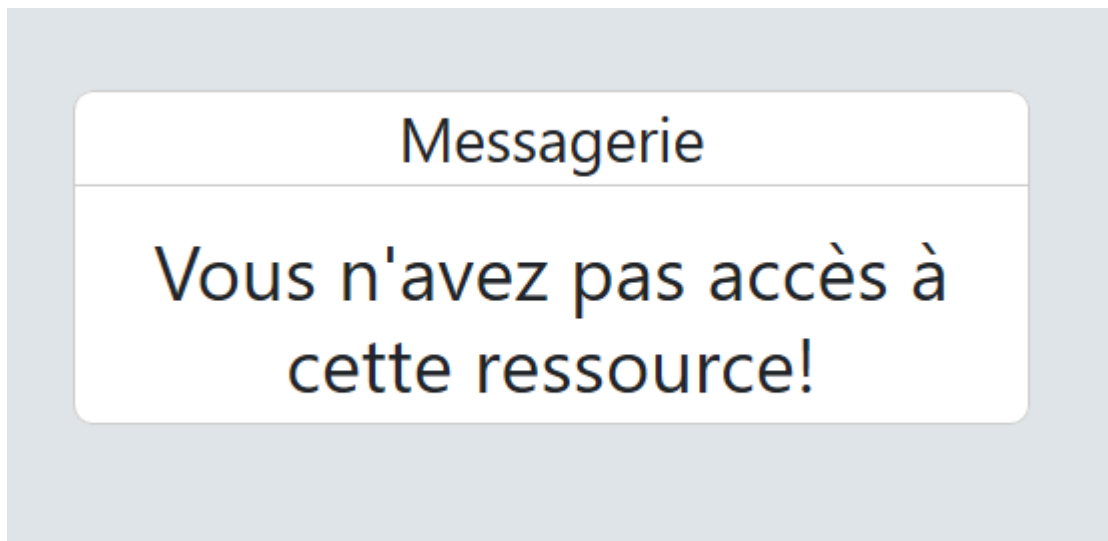
The screenshot shows a web browser window with the address bar containing the URL: `messagerie/modify-user-view.php?id=4&token=bihol7cz4rcw8oskwc8kgookwkcwcd`. The page title is "Modifier un utilisateur". Below the title, there is a section titled "Modifier l'utilisateur: Alice". This section contains two input fields: "Identifiant" with the value "Alice" and "Nouveau mot de passe" with the placeholder "Mot de passe".

Contre-mesures pour le scénario 4

Il faut ajouter un contrôle d'accès sur les liens sensibles :

- L'ajout des utilisateurs : add-user.php et add-user-view.php
- La suppression des utilisateurs : delete-user.php
- Le listing des utilisateurs : list-user.php
- La modification des utilisateurs : modify-user.php et modify-user-view.php

Pour cela on va créer une page vers laquelle rediriger en cas d'accès non autorisé:



Contre-mesures pour le scénario 5

Il existe de nombreuses solutions pour empêcher ou atténuer un DDoS. L'une d'entre elle de faire recours à une entité extérieure. Les plus connues sont Cloudflare et Imperva.

Conclusion

Sans surprise, le système qu'on a développé lors de la phase 1 du cours contenait des vulnérabilités non négligeables (CSRF, Injection SQL ou encore l'absence de cloisonnement entre les fonctions utilisateurs et administrateurs). Parmi les contre-mesures présentées, deux n'ont pas été implémentées à savoir l'utilisation de HTTPS et la protection et mitigation contre un DDoS car elles nécessitent des ressources et entités qui vont au-delà du cadre du cours.

En outre, nous nous sommes concentré à renforcer la sécurité du système contre des attaques qui nous semblaient vraiment évidentes et connues aux vues du nombre de vulnérabilités que l'application développée à la base contenait.