

# Projet 2 - STI

Projet 2 - STI .....	1
Introduction .....	3
Description du système.....	3
Data-Flow Diagram (DFD) .....	3
Identifier ses biens .....	3
Définir le périmètre de sécurisation .....	4
Sources de menaces.....	4
Scénario d'attaques/contre-mesures .....	5
Spidering .....	5
Scénario d'attaque : .....	5
Contre-mesures .....	5
Banner grabbing.....	6
Scénario d'attaque : .....	6
Contre-mesures .....	6
Champs cachés.....	7
Scénario d'attaque : .....	7
Contre-mesures .....	7
Mot de passe et nom d'utilisateur prédictibles : .....	8
Scénario d'attaque : .....	8
Contre-mesures .....	8
Brute-force du login, timing attack et message d'erreur trop verbeux.....	9
Scénario d'attaque : .....	9
Contre-mesures .....	9
Stockage mot de passe en clair : .....	10
Scénario d'attaque : .....	10
Contre-mesures .....	10
Transmission vulnérable de crédenciales .....	10
Scénario d'attaque : .....	10
Contre-mesures .....	10
Changement de mot de passe : .....	10
Scénario d'attaque : .....	11
Contre-mesures .....	11
Injection SQL .....	11
Scénario d'attaque : .....	11
Contre-mesures .....	11
Stored Cross-site scripting.....	12

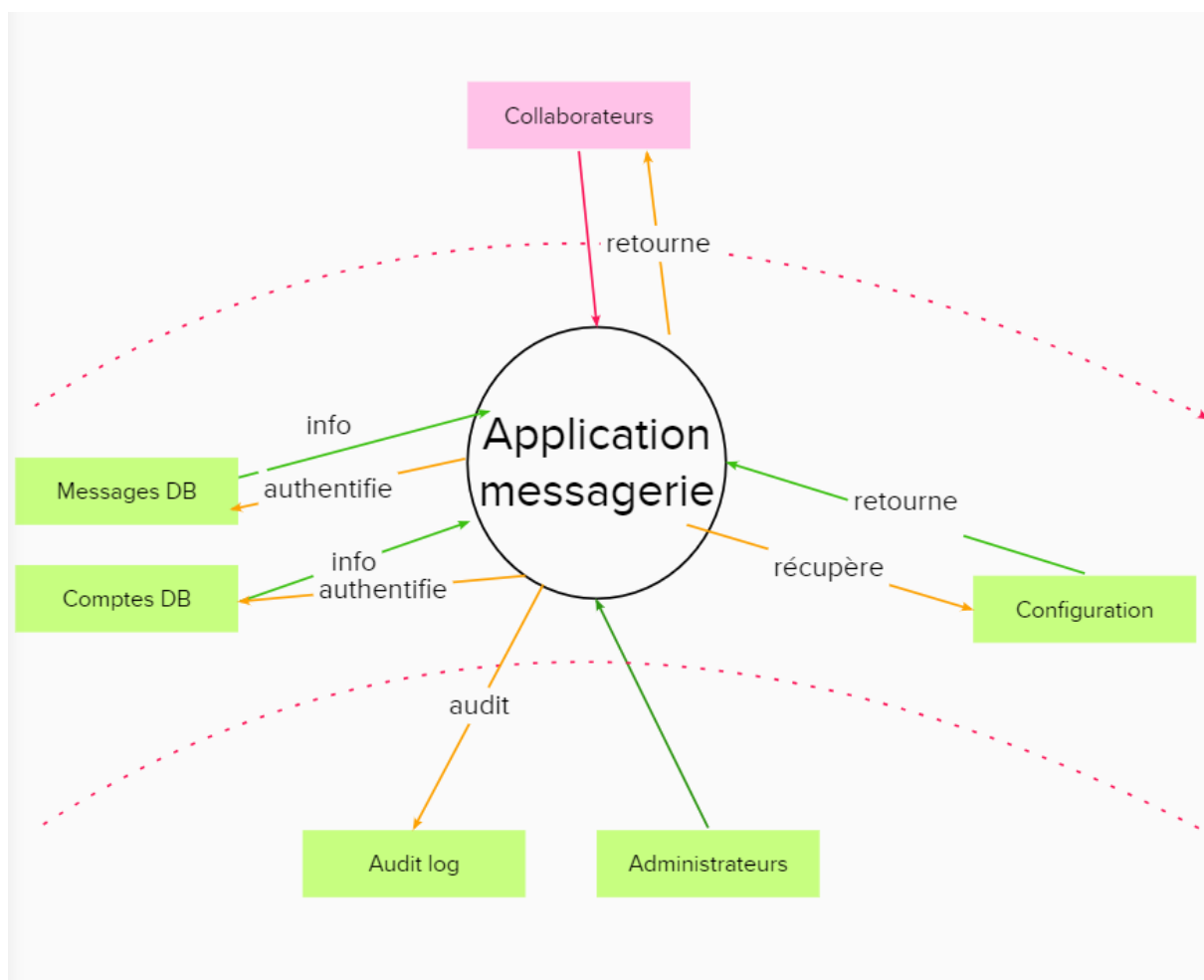
Scénario d'attaque : .....	13
Contre-mesures .....	13
Sources : .....	14

## Introduction

Le but de ce projet est de faire une modélisation de menaces et une analyse de risques de notre application de messagerie faite lors du projet précédent. Dans le but final est d'identifier et comprendre les menaces et de trouver les contre-mesures afin de corriger directement l'application pour essayer d'avoir une sécurité la plus optimale possible.

## Description du système

### Data-Flow Diagram (DFD)



### Identifier ses biens

Voici la liste des biens que nous devons protéger :

- Des données privées. Nous avons des messages qui sont censés être privés et qui ne devraient pas être divulgués. Il faut gérer la non-repudiation, on ne doit pas pouvoir changer l'origine/l'arrivée des messages.  
=> risques : pertes de confidentialité et d'intégrité. Un attaquant ne doit pas pouvoir modifier ces messages ni y avoir accès.
- Des droits administrateurs, ces droits ne doivent être distribués selon nos besoins et il ne doit pas être possible de faire de l'escalade de privilège.  
=> risques : pertes d'intégrité et de confidentialité. Un attaquant possédant les droits administrateurs peut changer les options des utilisateurs, ajouter des utilisateurs et a accès à tous les noms de comptes collaborateurs. Les autres comptes administrateurs ne voient pas les autres comptes administrateurs et ne peuvent pas changer leurs options, c'est déjà une défense mise en place.
- Des crédenciales qui se trouvent dans la base de données. Il existe des comptes administrateurs qui sont encore plus dangereux et les comptes collaborateurs.  
=> risques : perte de confidentialité, de disponibilité et d'intégrité. Un attaquant trouvant des crédenciales administrateurs dans la base de données peut faire ce qu'il veut avec l'application.
- Des "validity" enable/disable qui permettent d'autoriser si un compte peut accéder à l'application ou non. Ces validity ne doivent en aucun cas être changé par un utilisateur autre qu'un administrateur.  
=> risques : perte de disponibilité. Si un attaquant peut changer cette option il peut empêcher les comptes collaborateurs de se connecter et d'utiliser l'application de messagerie.

### Définir le périmètre de sécurisation

Nous allons nous occuper uniquement du côté application. La partie serveur et base de données n'est pas dans notre portée. Même si nous donneront quelques solutions sur certaines vulnérabilités serveurs qu'on aurait trouvé.

### Sources de menaces

Menaces humaines :

- Mauvaises utilisations d'un administrateur (une erreur dans une manipulation)
- Hackers, cybercriminels, terroristes. Ils pourraient voler nos crédenciales afin de les vendre sur le dark web. Utilisation de notre messagerie pour envoyer des Emails de phishing et pour faire du spam.
- Script kiddie voulant s'amuser, tester des outils qu'il ne connaît pas vraiment.
- Concurrent voulant nous faire du mal afin que nos clients partent chez eux. Ils peuvent également faire de l'espionnage industriel mais nous n'avons pas de fonctionnalités innovantes pouvant intéresser des concurrents.

Menaces naturelles :

- Panne technique venant d'un bug ou d'une erreur.

## Scénario d'attaques/contre-mesures

### Spidering

Element du système	Source	Motivation	STRIDE
Autorisations	Hacker, concurrent	Financier	Elevation of privilege, information disclosure

#### Scénario d'attaque :

Un concurrent demande à un hacker de nous voler la liste de nos clients afin d'essayer de nous les voler.

La première étape de tout attaquant étant un hacker avec des connaissances de base (on enlève les script kiddies des menaces) est de faire une cartographie de notre application. L'attaquant va essayer de faire un schéma du fonctionnement de notre application.

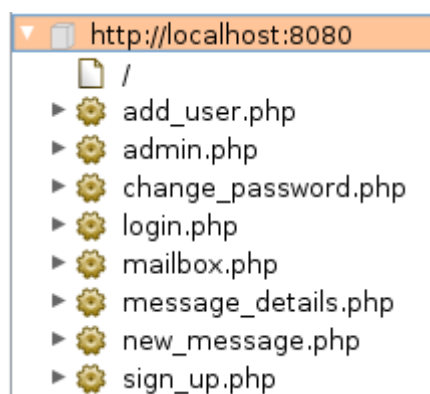
Une fois qu'il a le schéma il va essayer d'accéder aux pages où il n'a pas les autorisations. Il arrive alors à s'introduire sur la page administrateur où il peut connaître le nom de tous les collaborateurs et il peut même supprimer des comptes.

Cet attaquant peut également tomber sur des fichiers cachés comme des fichiers logs aillant des noms d'utilisateurs valides et des mots de passe d'un compte administrateur.

*Perte de confidentialité, perte d'intégrité, perte de disponibilité.*

#### Contre-mesures

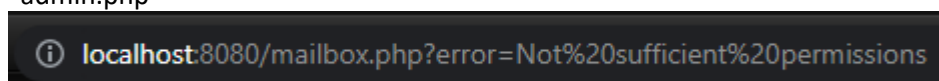
Nous avons alors fait du spidering. Nous avons utilisé l'outil de la suite Burp Community Edition v1.x car elle permet de faire plus facilement du spidering. Nous avons fait le test en tant que collaborateur car c'est le compte que peut posséder n'importe quel attaquant. Voici le résultat.




On voit que pour un utilisateur collaborateur, il peut connaître le chemin pour des emplacements où il ne devrait pas aller. Les pages : "admin.php" et "add\_user.php" ne devraient pas être connues de l'utilisateur.

Nous avons testé toutes les pages accessibles uniquement par l'administrateur. Elles ont toutes un contrôle de permissions.

"admin.php"



“add\_user.php”

 localhost:8080/mailbox.php?error=Not%20sufficient%20permissions

Afin de tester si on pouvait trouver d’autres fichiers cachés au niveau du serveur (fichiers configs par exemple). Nous avons utilisé l’outil ZAP OWASP avec la worldist traditionnelle. Celui-ci n’a rien trouvé.

### Banner grabbing

Element du système	Source	Motivation	STRIDE
Informations serveur	Hackers, concurrent	Financier	Denial of service

#### Scénario d’attaque :

Un concurrent demande à un hacker de faire du « Denial of service » afin d’empêcher le bon fonctionnement de notre application ce qui fera fuir nos clients chez lui.

L’attaquant va chercher des informations sur le serveur et langage de programmation utilisé afin de trouver des vulnérabilités possibles. Il peut trouver si notre version est à jour une vulnérabilité qui a été patché dans une version plus récente que sur laquelle nous sommes. Il peut y avoir pleins de vulnérabilités possibles en fonction de l’ancienneté de notre version. Il existe par exemple des vulnérabilité permettant de faire du DOS sur nos serveurs.

#### Perte de disponibilité

#### Contre-mesures

Nous avons d’abord analysé une requête venant du serveur.

```
HTTP/1.1 200 OK
Server: nginx/1.4.6 (Ubuntu)
Date: Mon, 27 Dec 2021 16:46:45 GMT
Content-Type: text/html
Connection: keep-alive
X-Powered-By: PHP/5.5.9-1ubuntu4.25
Set-Cookie: PHPSESSID=b7vco8uc7bd5n7lq6658mde545; path=/
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
```

La requête nous donne l’information sur le langage utilisé avec sa version (PHP/5.5.9), l’OS du serveur (ubuntu4.25) et le serveur (nginx/1.4.6).

Il est également possible de trouver l’informations sur la version du serveur en cherchant le path /robots.txt :

## 404 Not Found

nginx/1.4.6 (Ubuntu)

Si on fait quelques recherches on ne trouve pas de vulnérabilités avec la [version du serveur](#).<sup>1</sup> Cependant nous en trouvons une nous permettant justement de faire du DOS sur notre version de

<sup>1</sup> <https://www.cvedetails.com/version/334752/Nginx-Nginx-1.4.6.html>

PHP. <sup>2</sup>

### CVE-2016-7478

Zend/zend\_exceptions.c in PHP, possibly 5.x before 5.6.28 and 7.x before 7.0.13, allows remote attackers to cause a denial of service (infinite loop) via a crafted Exception object in serialized data, a related issue to CVE-2015-8876....

Il faudrait mettre à jours notre version de PHP et essayer de faire un changement directement sur le serveur pour éviter de donner des informations dans les entêtes http et dans les messages d'erreur.

### Champs cachés

Element du système	Source	Motivation	STRIDE
Messages	Hackers	Financier	Information disclosure

#### Scénario d'attaque :

Un attaquant utilise un proxy pour intercepter une requête lorsqu'il lit un message. Il trouve un champs caché ID, il va pouvoir changer cet ID et lire alors un autre message qui ne lui est pas destiné. Il peut tomber sur des Emails compromettant et faire du chantage en menaçant de divulguer les informations compromettantes.

#### Perte de confidentialité

#### Contre-mesures

Après plusieurs recherches dans le code avec un proxy (Burp) et en lisant le code source directement depuis la page web. Nous avons trouvé un champ caché. Ce champ caché est utilisé pour les boutons "Détails" "Delete" et "Answer" afin de pouvoir récupérer la bonne ID du message que l'on souhaite utiliser.

```
<div class="container">
  <form role="form" method="post" action="message_details.php"
    style="display: inline">
    <input id="messageId" type="hidden" name="messageId"
      value="1">
    <input class="btn btn-primary" type="submit" value="Details"
      name="details_button">
    </form>
  <form role="form" method="post" action="new_message.php"
    style="display: inline">
    <input id="messageId" type="hidden" name="messageId"
      value="1">
    <input class="btn btn-success" type="submit" value="Answer"
      name="answer_button">
    </form>
  <form role="form" method="post" action="/mailbox.php"
    style="display: inline">
    <input id="messageId" type="hidden" name="messageId"
      value="1">
    <input class="btn btn-danger" type="submit" value="Delete"
      name="delete_button">
    </form>
</div>
```

<sup>2</sup> <https://vulmon.com/searchpage?q=php%20php%205.5.9&sortby=bydate&scoretype=vmscore>

C'est dangereux, car avec un proxy il est alors possible de changer ce paramètre et donc on va pouvoir lire et/ou supprimer un message qui ne nous appartient pas.

Raw	Params	Headers	Hex
POST request to /message_details.php			
Type	Name	Value	
Cookie	PHPSESSID	hdnq5hu744d2o13ooma9j8l1q2	
Cookie	pla3412_1_9_6	e99bbc0a8beb6f8b6f068ee0d054a6ca	
Cookie	pla3412_1_9_6_salt	8oUBLv1mwz8E3UOWzxR8	
Body	messageId	1	
Body	details_button	Details	

Nous avons d'autres champs cachés dans la partie Admin où on voit tous les utilisateurs et où on peut changer leur mot de passe, leur rôle, leur validité ou juste supprimer leur compte. Le champ caché contient le nom de l'utilisateur. Dans ce cas c'est moins dangereux car on va pouvoir supprimer ou modifier un autre utilisateur en changeant ce champ caché avec un autre nom d'utilisateur mais c'est déjà le fonctionnement de cette page.

Une solution est de protéger ce paramètre pour qu'il ne soit pas modifiable lors d'une requête. Ou alors changer le fonctionnement de comment on passe l'ID à la page d'après, en utilisant la méthode GET pour le mettre dans l'URL.

#### Mot de passe et nom d'utilisateur prédictibles :

Element du système	Source	Motivation	STRIDE
Logique de l'application	Script kiddie	S'amuser, curiosité	Spoofing

#### Scénario d'attaque :

Un script kiddie tombe sur notre application, pas de chance il vient de lire un post facebook qui montrait une liste de mot de passe super basique qui fonctionne sûrement sur pleins d'applications. Il essaie alors sur notre login et il tombe sur un compte collaborateur et un administrateur. Il pourra avec le compte administrateur par exemple changer les mots de passe de tous les collaborateurs. C'est bien ce que ferai un script kiddie qui veut s'amuser.

#### Contre-mesures

Nous utilisons un compte admin avec un mot de passe admin. Il faut qu'il soit plus fort. De plus le username admin est très prédictible donc on pourrait aussi le changer. Les autres contre-mesures pour éviter qu'un collaborateur ne choisisse pas un mot de passe faible :

1. Forcer la complexité du mot de passe lors de la création du compte ou du changement de mot de passe.
  - a. Une longueur minimum de 15.
  - b. Doit contenir au moins un symbole spécial (&, /, \*, ", +, ...)
  - c. Doit contenir au moins un numéro.
  - d. Doit contenir au moins une lettre majuscule.
  - e. Doit contenir au moins une lettre minuscule.
2. Obliger le changement de mot de passe après 180 jours en envoyant des E-Mail de notification.
3. Empêcher les nouveaux mots de passe d'être des mots de passe déjà utilisés par le passé.



## Brute-force du login, timing attack et message d'erreur trop verbeux

Element du système	Source	Motivation	STRIDE
Logique de l'application	Script kiddie	S'amuser, curiosité	Spoofing

### Scénario d'attaque :

Un script kiddie tombe sur notre application, pas de chance il vient d'installer une kali Linux et veut essayer l'outil hashcat. Il lance alors un brute force sur notre login et tombe sur un compte administrateur ayant un mot de passe faible. Il peut par exemple changer les mots de passe de tous les collaborateurs. C'est bien ce que ferait un script kiddie qui veut s'amuser.

### Contre-mesures

Nous avons testé avec « John the Ripper » et il a trouvé très rapidement le compte « admin:admin ». Il faut rajouter une défense contre le brute-force. Soit on bloque le login s'il y eu trop de tentatives à la suite soit on oblige de remplir un captcha. En plus d'avoir de faire les contre-mesures de la complexité des mots de passe.

Afin de trouver des résultats plus rapidement avec un outil de brute-force on peut essayer de deviner des noms d'utilisateurs à lui donner. On essaie alors de faire une timing attack. Voici les résultats :

En testant des utilisateurs qui n'existent pas :

Durée	Cascade
57 ms	
62 ms	

login.php	200	document	Autre	972 B	57 ms	
-----------	-----	----------	-------	-------	-------	--

On reste dans les 50-65 ms.

En testant avec le nom de compte admin (un compte existant) :

Nom	État	Type	Initiateur	Taille	Durée	Cascade
login.php	302	document / Redir...	Autre	394 B	65 ms	
login.php?error=Wrong%20password	200	document	login.php	972 B	7 ms	

On reste également dans les 50-65 ms.

Parfois les données étaient plus basses dans les deux cas. Ce n'est pas une science exacte. On pourrait utiliser un outil pour faire plus de tests afin d'être sûr. Cependant dans notre cas c'est normal qu'il n'y a pas vraiment de différence car nous ne faisons pas de hashage. Donc il n'y a pas plus de temps qui est pris pour la vérification. Quand nous allons corriger le système de stockage de mot de passe, il est nécessaire de faire attention à éviter une "Timing attack" en faisant du code constant. C'est à dire que quand on vérifiera le nom d'utilisateur et que celui-ci n'existe pas on fera un hash du mot de passe pour perdre du temps.

On constate également qu'il y a un message d'erreur, donc même pas besoin de "Timing attack". Nous sommes sur un cas de message d'erreur trop verbeux.

Il faut juste faire un message d'erreur générique. Comme "Wrong credentials". Afin de donner le moins d'informations possible.

### Stockage mot de passe en clair :

Element du système	Source	Motivation	STRIDE
Logique de l'application, gestion de la base de données	Erreur de manipulation d'un administrateur	Erreur	Information disclosure

#### Scénario d'attaque :

Un administrateur fait une mauvaise manipulation lors d'une mise à jour sur la base de données dump la base de données en entier sur internet. Tous les mots de passes sont en clair donc tous les comptes sont vulnérables.

#### Contre-mesures

Si on va sur la page de la base de données, on constate que nos mots de passe sont stockés en clair.

← T →	username	password	roles	validity
<input type="checkbox"/>	Edit	Delete	admin	admin
			1	1

Il faut hasher ces mots de passe afin d'avoir une seconde ligne de défense si par malheur la base de données est rendue visible.

Selon les recommandations OWASP, il faut utiliser un algorithme de hashage dans cette liste :

- Argon2id
- Bcrypt
- Scrypt
- PBKDF2

Et il faut considérer utiliser un « pepper » pour ajouter encore plus de défense.

### Transmission vulnérable de crédenciales

Element du système	Source	Motivation	STRIDE
Serveur	Hackeur	Erreur	Information disclosure

#### Scénario d'attaque :

Un hackeur performe une attaque Man In The Middle avec un outil d'écoute passive (wireshark) devant la maison d'un client de notre application. Les requêtes envoyées sont en http et ne sont donc pas chiffré. Il attend juste que le client se logue et il pourra intercepter ses credenciels.

#### Perte de confidentialité

#### Contre-mesures

Nous utilisons HTTP en local donc la transmission sera toujours non sécurisée. Une attaque MITM pourrait récupérer les crédenciales lors du login, du sign up et du changement de mot de passe.

Il faudrait utiliser un moyen de transmission chiffré (SSL/TSL) surtout lorsqu'on manipule des données sensibles.

### Changement de mot de passe :

Element du système	Source	Motivation	STRIDE
--------------------	--------	------------	--------

Logique de l'application	Hacker	Amusement	Elevation of privileges
--------------------------	--------	-----------	-------------------------

#### Scénario d'attaque :

Un hacker trouve notre compte administrateur et change le mot de passe. Nous n'utilisons pas ce compte pour le moment et continuons de travailler sans s'en rendre compte. L'attaquant aura beaucoup de temps de faire des dégâts et nous fera perdre plus d'argent. Il faut essayer de réduire les conséquences de nos vulnérabilités.

*Perte de disponibilité, perte d'intégrité, perte de confidentialité*

#### Contre-mesures

Pour le changement de mot de passe il manque certaines étapes afin de le sécuriser.

Il faut demander l'ancien mot de passe, pour qu'un attaquant qui arrive à se connecter via cookie par exemple ne puisse pas changer le mot de passe.

Il faut notifier le changement de mot de passe par un mail, pour qu'une victime puisse être au courant si son compte a été compromis.

#### Injection SQL

Element du système	Source	Motivation	STRIDE
Logique de l'application, base de données	Hacker	Vengeance	Tampering

#### Scénario d'attaque :

Un hacker veut n'aimer pas un des administrateurs donc il décide d'attaquer l'application en faisant des requêtes SQL dans tous les champs d'entrée utilisateur. Son but est de trouver une faille et de supprimer la base de données.

*Perte de disponibilité*

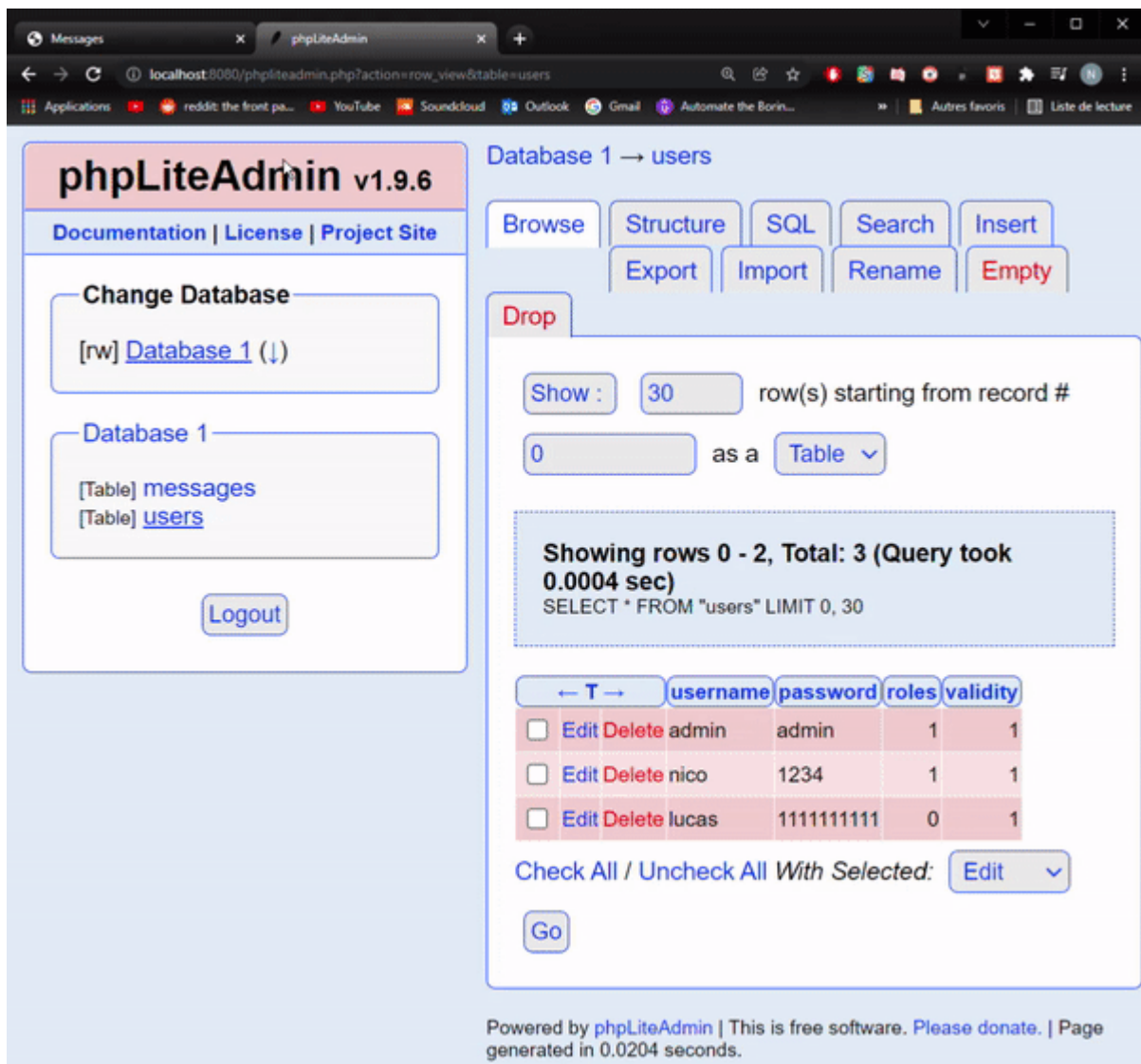
#### Contre-mesures

Nous avons tout d'abord essayé de faire des injections dans les champs de la création de message. Lorsque nous envoyons juste un apostrophe la page se bloque, le message ne s'envoie pas et il n'y a pas de message d'erreur. Il y a eu un problème du côté client. Nous sommes vulnérables aux injections SQL.

Tous les autres champs d'entrée d'utilisateur (Changement de mot de passe et la page administration) ont le même comportement. Nous montrons l'exploitation de la vulnérabilité en faisant une injection lors d'un changement de mot de passe d'un collaborateur. Au lieu de changer juste celui de l'utilisateur, nous changeons les mots de passe de tous les utilisateurs :<sup>3</sup>

---

<sup>3</sup> Le gif se trouve dans le dossier du rapport sous le nom « SQL\_injection\_GIF »



Voici les différentes défenses possibles :

- Filtrer les entrées utilisateur  
Le but est de filtrer les caractères spéciaux non nécessaires à l'entrée utilisateur et les mots spécifiques SQL comme DROP. Pour simplifier, il est possible de forcer un format pour une certaine entrée. Par exemple pour le nom du lot, l'entrée est toujours une date donc on peut obliger de mettre ce format-ci mais il faut contrôler du côté serveur.
- Faire de l'échappement sur les entrées utilisateur  
Il faut échapper les caractères dangereux pour SQL comme pour l'étape d'avant.
- Limiter la taille des entrées utilisateur
- Utiliser des « requêtes préparées ». Pour PHP il existe la fonction « `mysqli_prepare` », c'est celle que nous utiliserons pour corriger la vulnérabilité.

### Stored Cross-site scripting

Element du système	Source	Motivation	STRIDE
--------------------	--------	------------	--------

Navigateur, logique de l'application	Terroriste	Terroriste	Spoofin, Information disclosure
--------------------------------------	------------	------------	---------------------------------

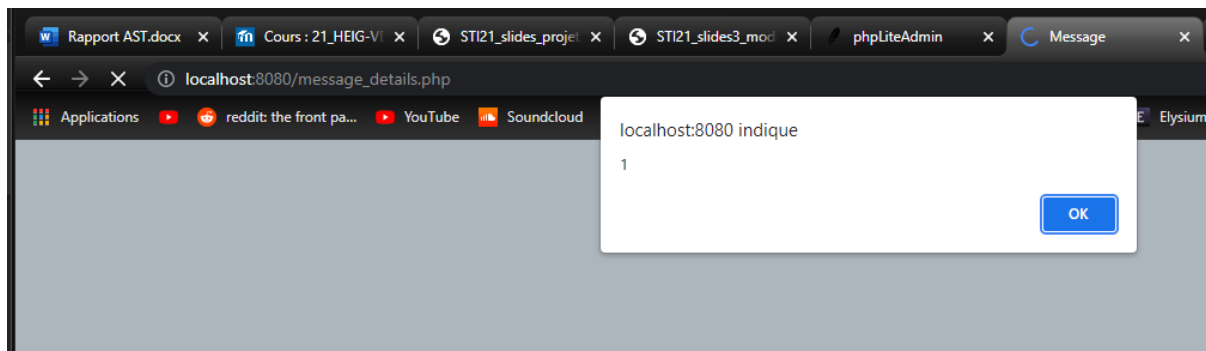
#### Scénario d'attaque :

Un groupe terroriste veut se faire connaître, ils veulent alors un compte administrateur afin de récupérer une liste d'adresse Email à qui envoyer des messages promottant leur terrorisme. L'attaquant va envoyer un Email malicieux à un administrateur avec du « stored XSS » qui enverra le cookie de l'administrateur sur le serveur de l'attaquant. Il pourra alors l'utiliser pour usurper l'identité de l'administrateur.

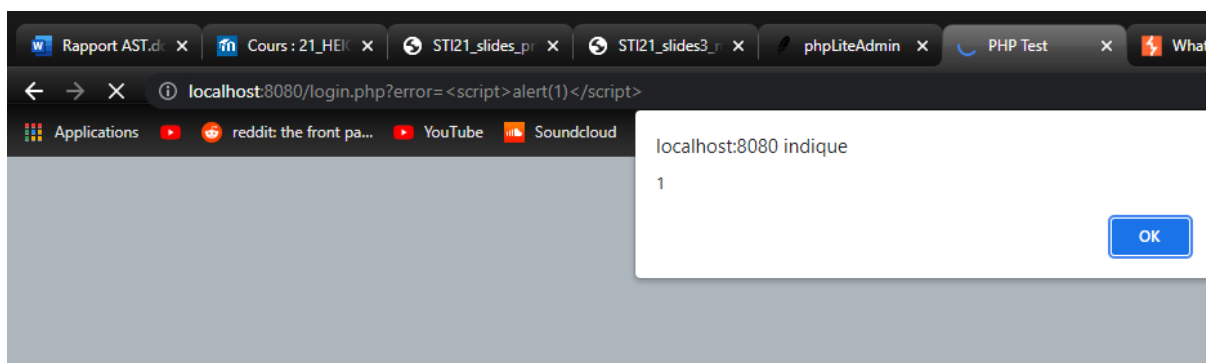
#### Perte de confidentialité

##### Contre-mesures

Nous avons d'abord testé pour le « stored XSS », dans toutes les entrées insérant/modifiant dans la base de données sont vulnérables. Voici un exemple avec l'envoi de message. On envoie `<script>alert(1)</script>` en tant que contenu et quand on va lire les détails du message on reçoit l'alerte :



Pour le « reflected XSS » nous avons uniquement les messages d'erreur qui sont affichés depuis le paramètre « error= » de l'URL. Nous pouvons alors l'exploiter :



Voici les différentes défenses possibles contre ce type d'attaques ;

- Valider les inputs utilisateurs (Être aussi restrictif que possible).
- Valider l'output (réponse du serveur). Le but est d'encoder la réponse HTML afin que les caractères potentiellement malicieux soient traités comme du contenu HTML et non comme une structure HTML. Par exemple < devient &lt;.

## Sources :

- Application pour faire la DFD

<https://app.mural.com>