

**STI**

Projet 2

Groupe

*Ali Miladi*

*Julien Brêchet*

Professeur

*Abraham Rubinstein*

# Table des matières

[1 Table des matières 2](#_Toc503788359)

[2 Introduction 3](#_Toc503788360)

[2.1 Objectifs 3](#_Toc503788361)

[3 Description du système 3](#_Toc503788362)

[3.1 Objectifs du système 3](#_Toc503788363)

[3.2 Data Flow Diagram 4](#_Toc503788364)

[3.3 Définition du périmètre de sécurisation 4](#_Toc503788365)

[3.4 Hypothèses de sécurité 4](#_Toc503788366)

[3.5 Exigences de sécurité 5](#_Toc503788367)

[3.6 Enumération des actifs 5](#_Toc503788368)

[3.7 Rôles des utilisateurs 5](#_Toc503788369)

[4 Identification des menaces 6](#_Toc503788370)

[4.1 Scénario d’attaque 1 6](#_Toc503788371)

[4.2 Scénario d’attaque 2 7](#_Toc503788372)

[4.3 Scénario d’attaque 3 8](#_Toc503788373)

[4.4 Scénario d’attaque 4 9](#_Toc503788374)

[4.5 Scénario d’attaque 5 9](#_Toc503788375)

[4.6 Scénario d’attaque 6 9](#_Toc503788376)

[4.7 Scénario d’attaque 7 10](#_Toc503788377)

[4.8 Scénario d’attaque 8 10](#_Toc503788378)

[4.9 STRIDE 11](#_Toc503788379)

[5 Identification des contre-mesures 11](#_Toc503788380)

[5.1 Contre-mesure 1 11](#_Toc503788381)

[5.2 Contre-mesure 2 12](#_Toc503788382)

[6 Conclusion 12](#_Toc503788383)

# Introduction

Dans ce rapport, nous allons présenter l’application web développée dans le cadre du cours de STI.

Cette application consiste en une plateforme de messagerie en ligne et a été développée sur deux phases. La première consistait à implémenter simplement les fonctionnalités de base de l’application sans pour autant s’intéresser à l’aspect sécuritaire. La deuxième phase se traduisait par la sécurisation de cette application avec l’utilisation de tous les outils vus en classe au travers des présentations des camarades.

Nous allons nous pencher dans ce document sur les aspects sécuritaires revus pour cette deuxième phase du projet.

## Objectifs

L’objectif de ce projet a été de maitriser les principes de programmation web sécurisée dans le but de développer une application web sécurisée avec PHP.

Ce projet a fait suite au premier qui nous demandait de développer une application web de messagerie en ligne mais qui n’utilise pas de protocole SMTP. Il fallait reprendre se dernier et le sécurisé.

Tout au long de ce travail, nous avons fait une analyse des menaces complète qui nous a permis de nous concentrer sur les vulnérabilités et les points vitaux à sécuriser dans l’application.

Cette sécurisation ne concerne que l’application web en elle-même mais en aucun cas l’OS hôte, le réseau LAN ou la configuration du serveur PHP. Nous considérons que ces éléments ont déjà été sécurisés au préalable.

# Description du système

Notre application est une plateforme de messagerie en ligne offrant une interface simple pour les utilisateurs dans le but de la communication interne d’une organisation (entreprise, école, administration, …).

Un utilisateur inscrit dans la base de données de l’application doit pouvoir envoyer des messages à n’importe quel autre utilisateur inscrit. Il sera aussi capable de consulter tous les messages qui lui sont destinés et y répondre. D’autre part, il peut modifier son mot de passe pour le login.

Un utilisateur administrateur, en plus de ce que peut faire un utilisateur normal, est en mesure de gérer les comptes des autres utilisateurs inscrits dans la base de données. Il a la possibilité de consulter la liste de tous les utilisateurs avec des détails sur leurs comptes. Il peut aussi modifier n’importe quel profil d’utilisateur à savoir changer le mot de passe, l’identifiant, le type de profil (admin ou régulier) et activer ou désactiver le compte.

## Objectifs du système

L’objectif primaire de notre application est de permettre aux utilisateurs de communiquer aisément au sein de leur organisation.

Nous n’avons pour le moment aucun objectif financier, en effet, l’application est en l’état à but non lucratif visant seulement à banaliser la communication entre ses utilisateurs. Néanmoins, des fonctionnalités plus évoluées et des améliorations pourraient être apportées dans le futur et seraient alors inclues dans un paquet payant « premium » en plus de l’application de base.

D’autre part, l’application aura pour but de gagner en réputation et notoriété pour tous les types d’organisations. L’objectif est de la rendre accessible, pratique, user-freindly et suffisamment sécurisée pour gagner la confiance des utilisateurs.

## Data Flow Diagram



## Définition du périmètre de sécurisation

Pour ce projet, nous allons sécuriser l’application web en elle-même. Nous allons protéger les bases de données et les données privées des utilisateurs contre les injections.

Nous allons aussi protéger l’application contre les injections de code et les attaques XSS.

La validation des champs des formulaires des utilisateurs, l’imposition de la vérification de mots de passe ainsi que la mise en place d’une politique de mots de passe forts vont être introduites.

## Hypothèses de sécurité

Nous considérons que l’application devra idéalement tourner sur un serveur de l’organisation. Elle pourra aussi très bien tourner sur une station de travail sur laquelle un serveur PHP est installé.

Nous faisons dès lors des hypothèses de sécurité quant à l’infrastructure physique de l’organisation utilisatrice de l’application ainsi que le réseau local.

Nous assumons que :

* Le serveur (éventuellement workstation) sur lequel tourne l’application est de confiance. Il aura toujours les dernières mises à jour de sécurité de l’OS et de PHP.
* Les systèmes d’exploitation des membres de l’organisation (utilisateurs de l’application) sont de confiance et ayant toujours les dernières mises à jour de sécurité, antivirus actif, pare-feu, … .
* Le réseau local de l’organisation est bien protégé par des pares-feux, les dispositifs nécessaires pour un minimum de sécurité du LAN devraient être fournis.
* Les administrateurs système/réseau sont des personnes de confiance.

## Exigences de sécurité

Quelques exigences de sécurité ont été fixées pour notre application :

* Le contenu des messages doit être intègre donc non modifiable. Un message ne doit pas être changé en cours de route.
* Un utilisateur ne doit pouvoir lire que les messages qui lui sont destinés.
* L’application n’est accessible qu’aux membres inscrits et enregistrés dans la base de données. Aucune page de l’application à part la page de login n’est accessible à n’importe quel utilisateur non authentifié. Il faut donc s’authentifier avant de pouvoir utiliser l’application.
* Selon le type de compte d’utilisateur, celui-ci ne peut pas accéder aux pages de l’autre type de compte.
* Les informations critiques des utilisateurs (identifiant et mot de passe) doivent être privées et protégées.
* L’application doit être disponible à 99% du temps.

## Enumération des actifs

Pour notre application, nous pouvons identifier quelques actifs ayants une haute valeur. Sans ceux-ci, l’application ne pourra pas fonctionner et sans solution de backup et de protection adaptée, nous ne pouvons pas les reconstruire.

* La base de données des utilisateurs : doit être protégée en intégrité et confidentialité. En cas de corruption des comptes utilisateur ou vol d’identités, nous perdons la confiance des clients ainsi que de la notoriété.
* La base de données des messages : doit être également protégée en intégrité et confidentialité. Dans le cas de vol de messages ou de modification de leurs contenus, la réputation de l’application serait en danger. D’autre part, dans le cas où la base de données est corrompue ou effacée, et qu’elle n’a pas été backupée, les utilisateurs perdraient à jamais leurs messages.
* L’infrastructure de l’application : le serveur (et éventuellement le load balancer dans le cas où on déploie pour une grande organisation ayant besoin d’une très haute disponibilité et redondance) doit être protégé en intégrité et doit fournir une haute disponibilité. Une coupure du serveur ou corruption de celui-ci nuirait à la réputation de l’application.

## Rôles des utilisateurs

Nous pouvons distinguer deux rôles d’utilisateurs distincts comme énoncé plus haut dans la description du système :

* Les utilisateurs réguliers : Ceux qui ont accès aux fonctionnalités de base. Ils peuvent envoyer/recevoir et supprimer des messages ainsi que changer de mot de passe.
* Les administrateurs : En plus des fonctionnalités de base des utilisateurs réguliers, les admins ont aussi des fonctionnalités supplémentaires de gestion des utilisateurs. Ils peuvent en créer, avoir la liste de tous les utilisateurs inscrits avec les détails ainsi que modifier les détails des comptes de ces utilisateurs (nom d’utilisateur, mot de passe, type de compte, activer/désactiver un compte).

# Identification des menaces

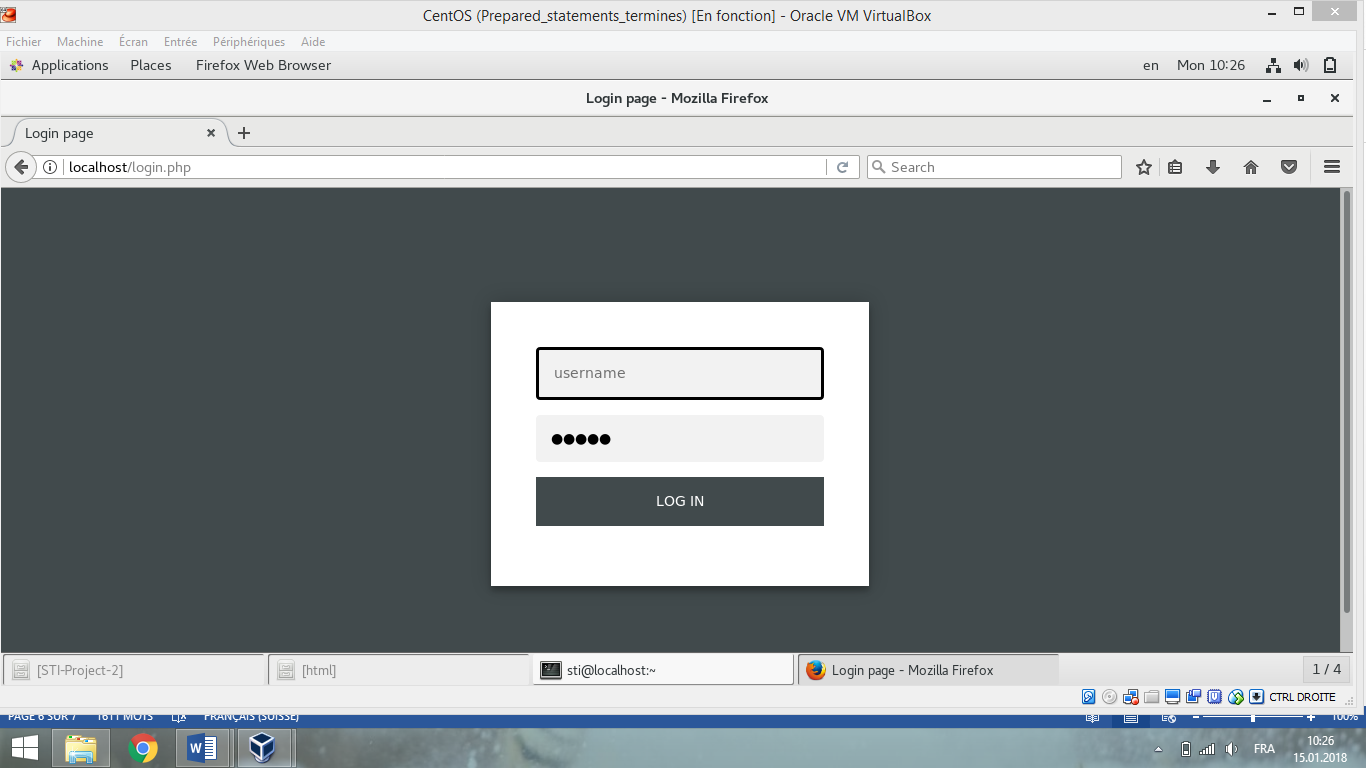
Plusieurs sources potentielles d’agression de l’application ont été identifiées :

* Hackers/script-kiddies :
  + Motivation : amusement et gloire parmi les hackers
  + Cible : tous les actifs de l’application
  + Potentialité : haute
* Cybercrime :
  + Motivation : vol d’identités, informations, chantage, gain d’argent, …
  + Cible : bases de données de l’application et cookies des utilisateurs
  + Potentialité : moyenne-haute
* Concurrents :
  + Motivation : vol du code source et de la logique métier de l’application
  + Cible : code source de l’application
  + Potentialité : faible
* Utilisateurs malins :
  + Motivation : vol d’identités, vols d’informations compromettantes des autres utilisateurs et de l’application
  + Cible : tous les actifs de l’application web

Ci-dessous, nous présentons quelques scénarios d’attaques

## Scénario d’attaque 1

Notre application de messagerie utilise des requêtes SQL pour aller chercher les informations dont elle a besoin. Par exemple, lorsqu’un utilisateur se logue sur l’application, cette dernière va effectuer une requête SQL sur une base de données SQLite contenant toutes les données relatives à notre application web. Elle va effectuer une requête SQL pour récupérer les informations de l’utilisateur qui se connecte et vérifier si le username existe et que le mot de passe entré soit correct.



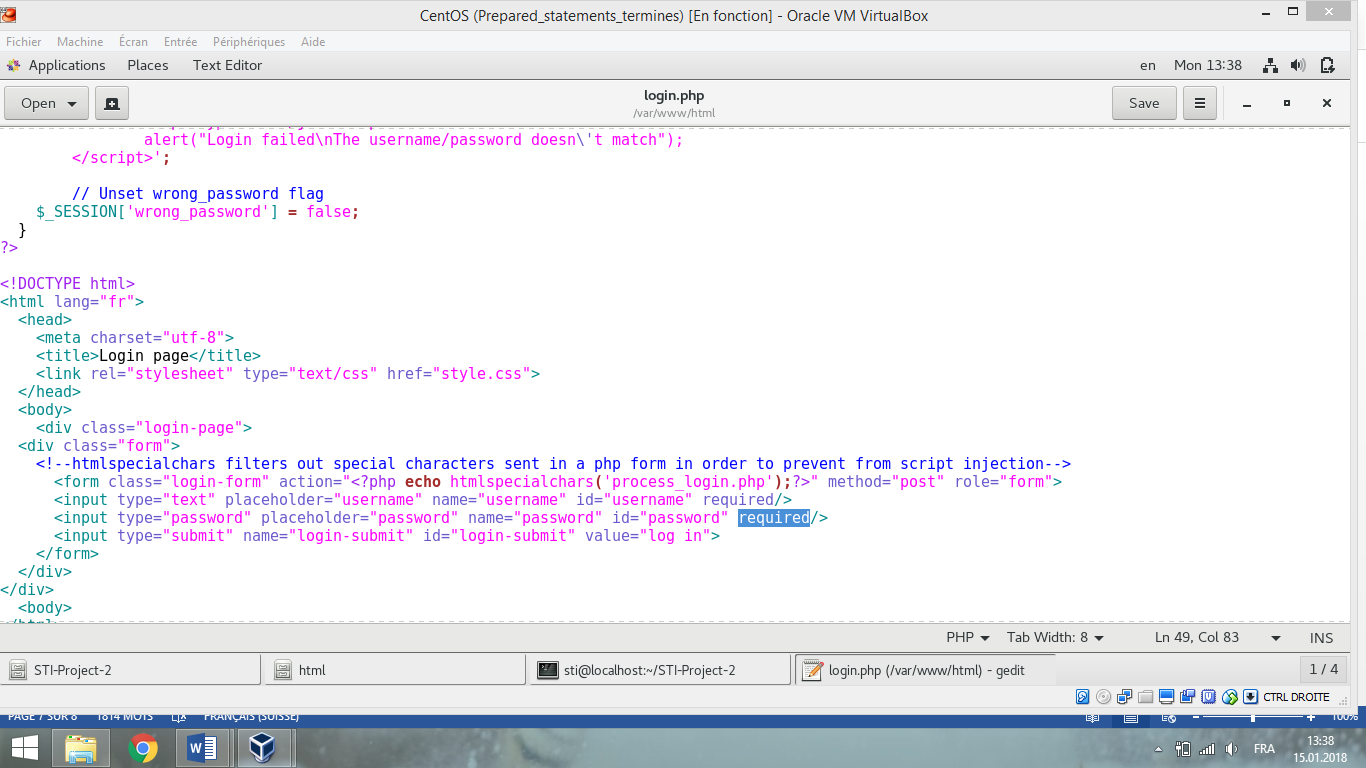
Au lieu d’entrer un nom d’utilisateur dans le champ « username », un attaquant pourrait injecter un code SQL lui retournant par exemple tous les usernames stockés dans la base de données ainsi que leurs mots de passe. L’attaquant pourrait ainsi récupérer les informations de tous les comptes de l’application et donc se connecter avec n’importe quel compte utilisateur qu’il a dérobé.

On appelle ce genre d’attaques des **injections SQL**.

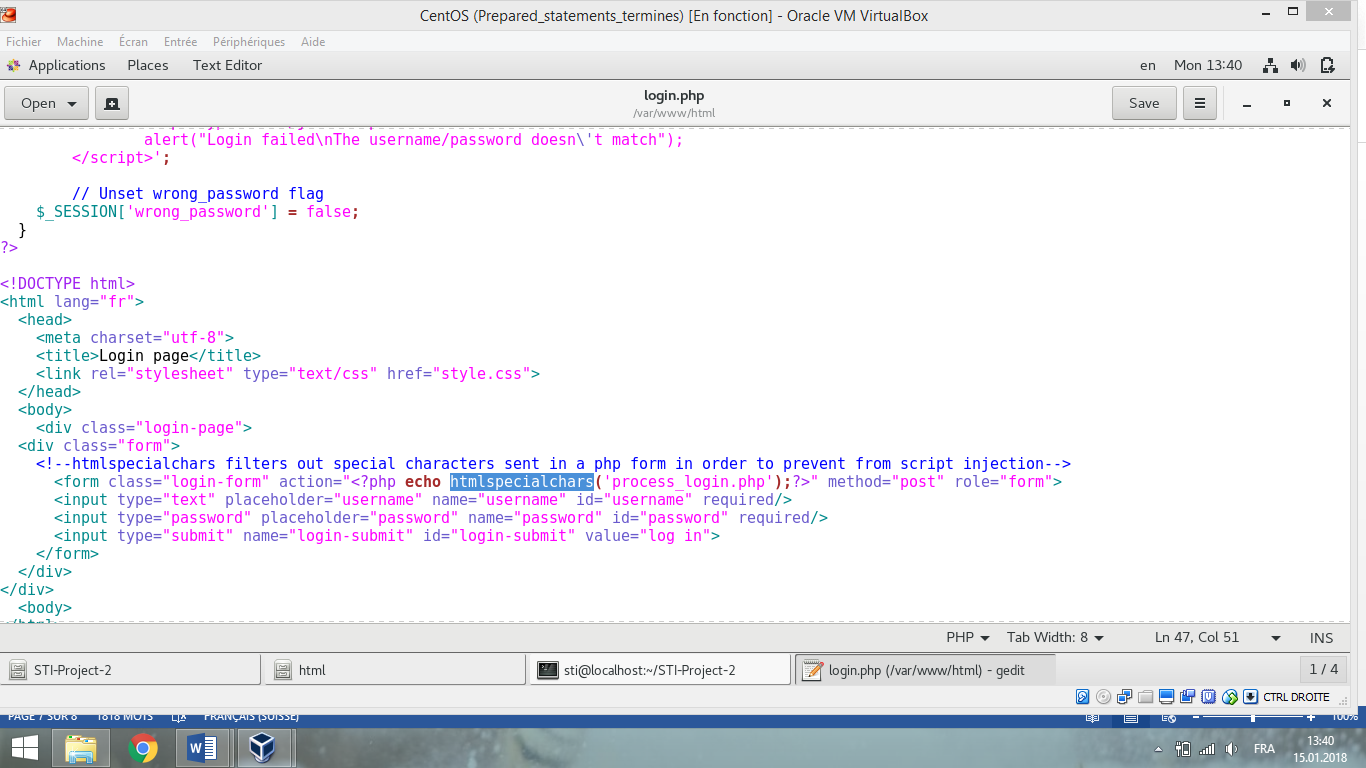
## Scénario d’attaque 2

Rajouter les contrôles au niveau du serveur pour les attributs `required` + vérifier si côté client tout est OK

Côté client



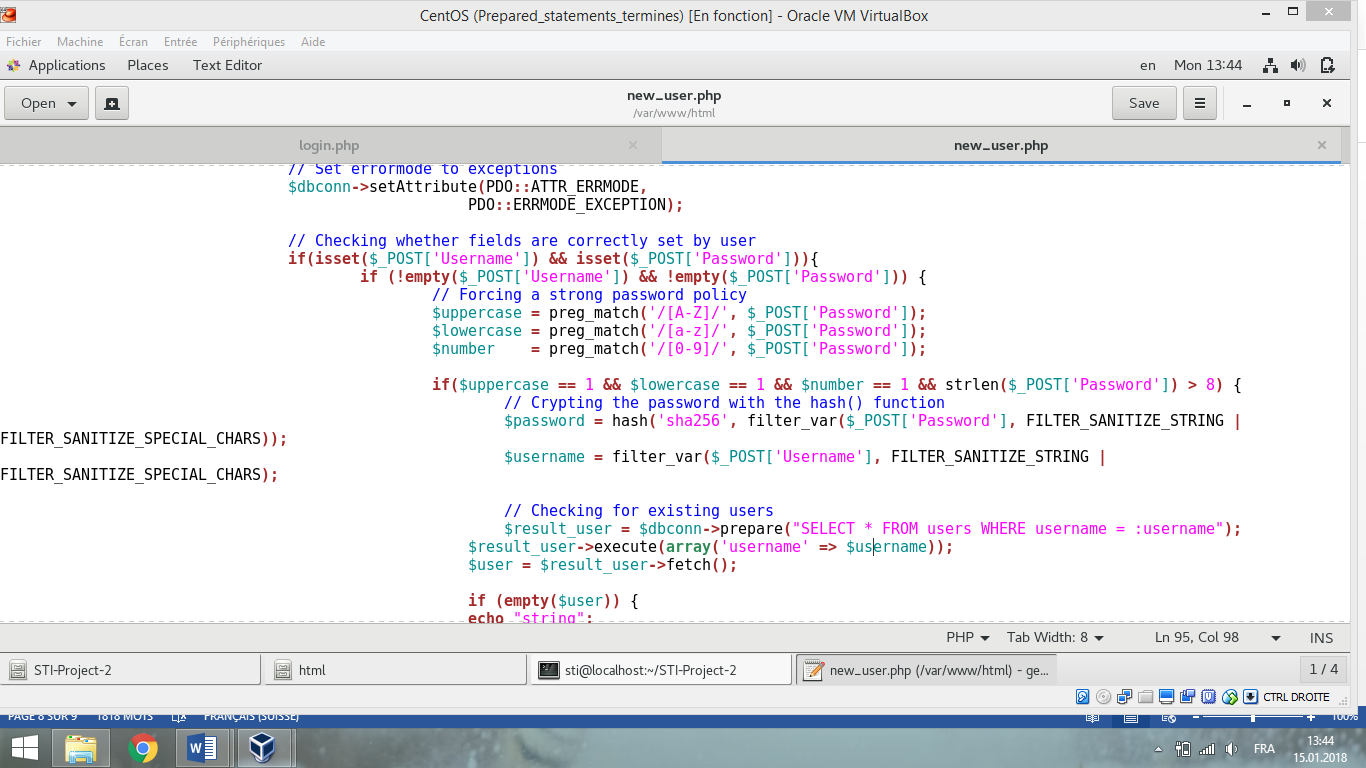
Et côté serveur :



Echapper le contenu des scripts

Ce qu’il y a entre les balises <script>

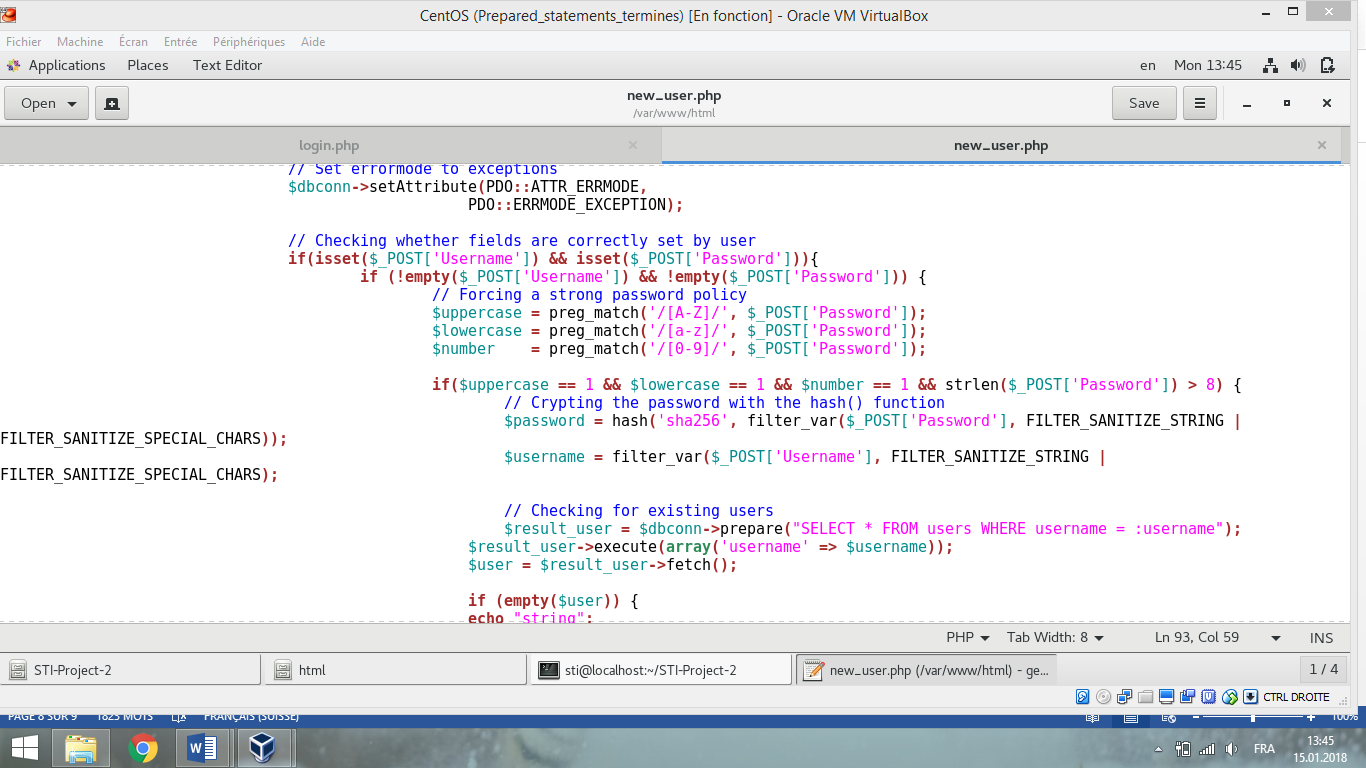
Vérifier



## Scénario d’attaque 3

Politique de mots de passe forts

Forcer une politique de mots de passe forts



## Scénario d’attaque 4

Protéger contre les scripts

HTMLSPECIALCHARS + SANITY FILTER

Filtrer les insertions de scripts dans les champs des formulaires

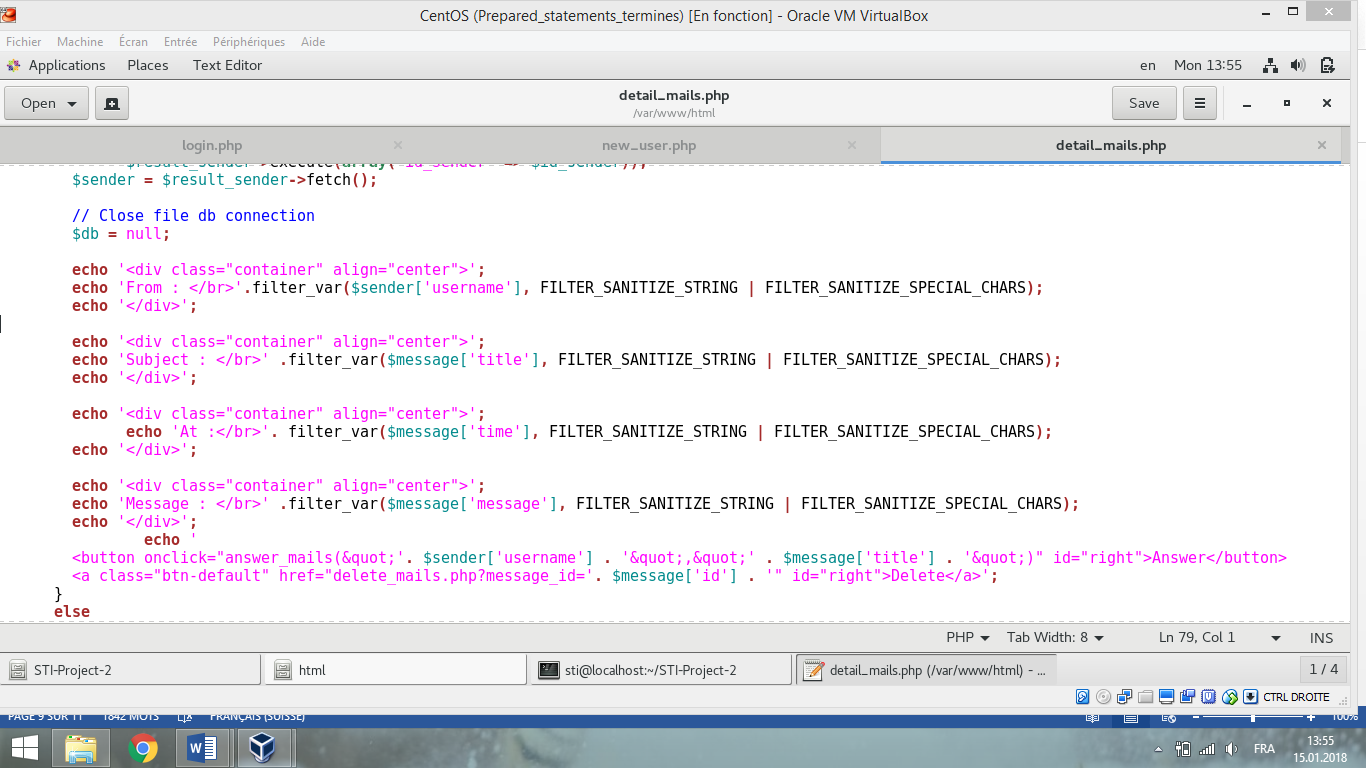
## Scénario d’attaque 5

Hasher les mots de passe dans la base de données

ATTENTION pas de protection contre les rainbow tables

## Scénario d’attaque 6

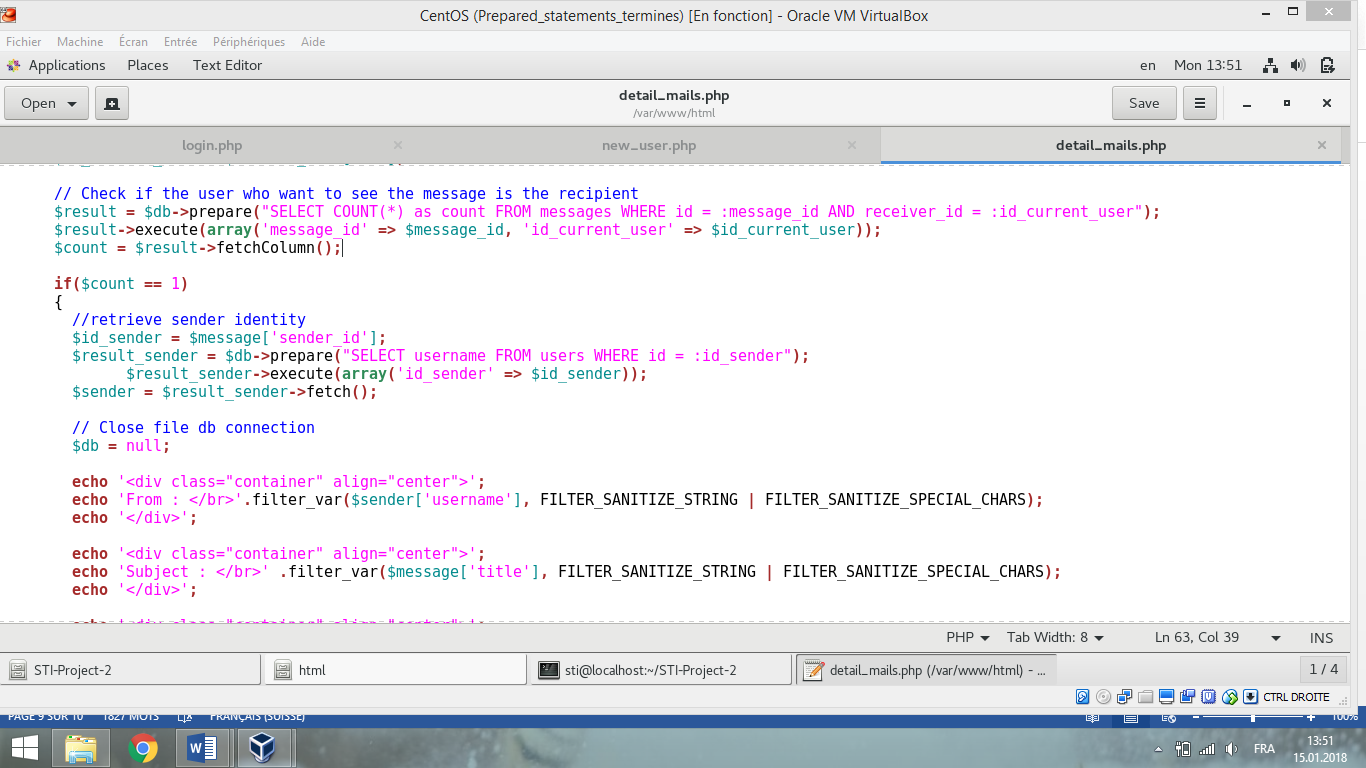
Filtrer les XSS stockés



## Scénario d’attaque 7

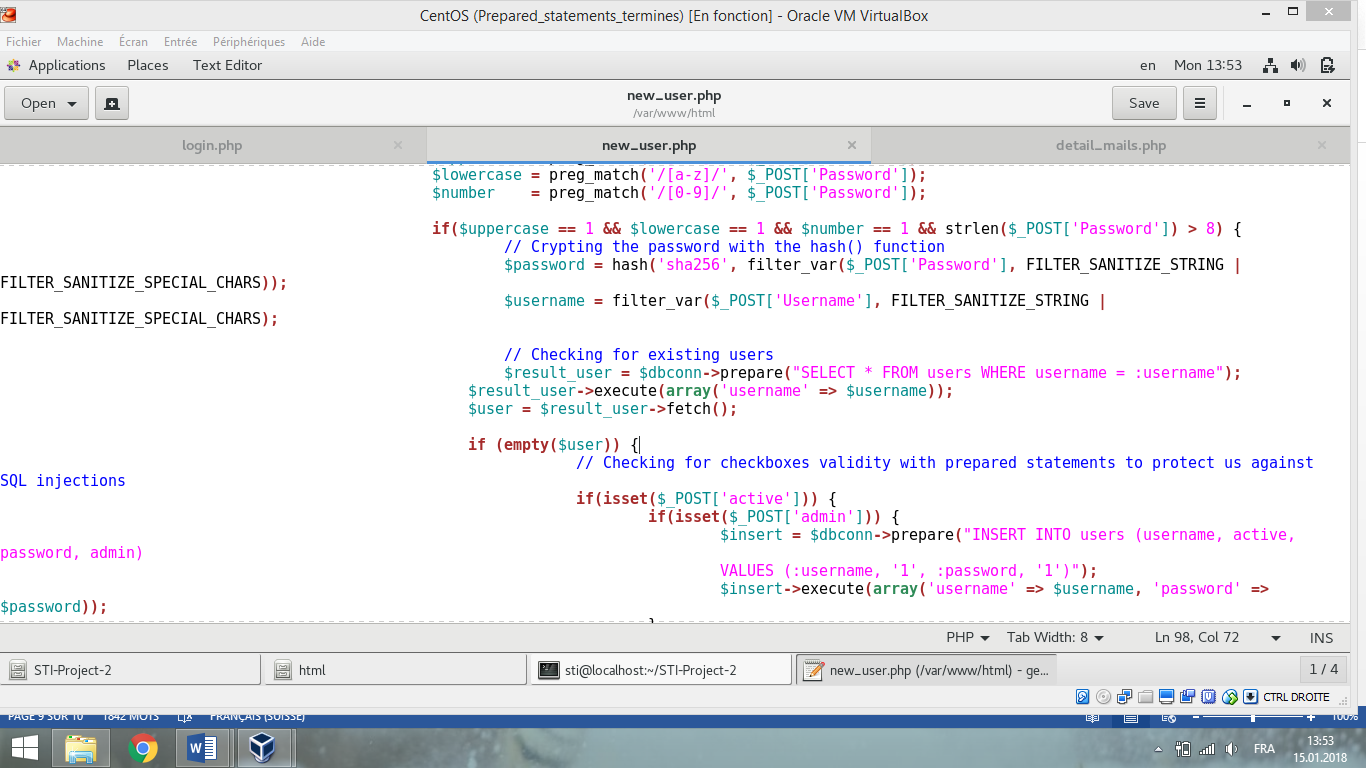
Interdire la consultation des messages dont les ID ne sont pas autorisés

Si un compte est compromis blablabla



## Scénario d’attaque 8

On ne peut plus créer des comptes avec le même username



## STRIDE

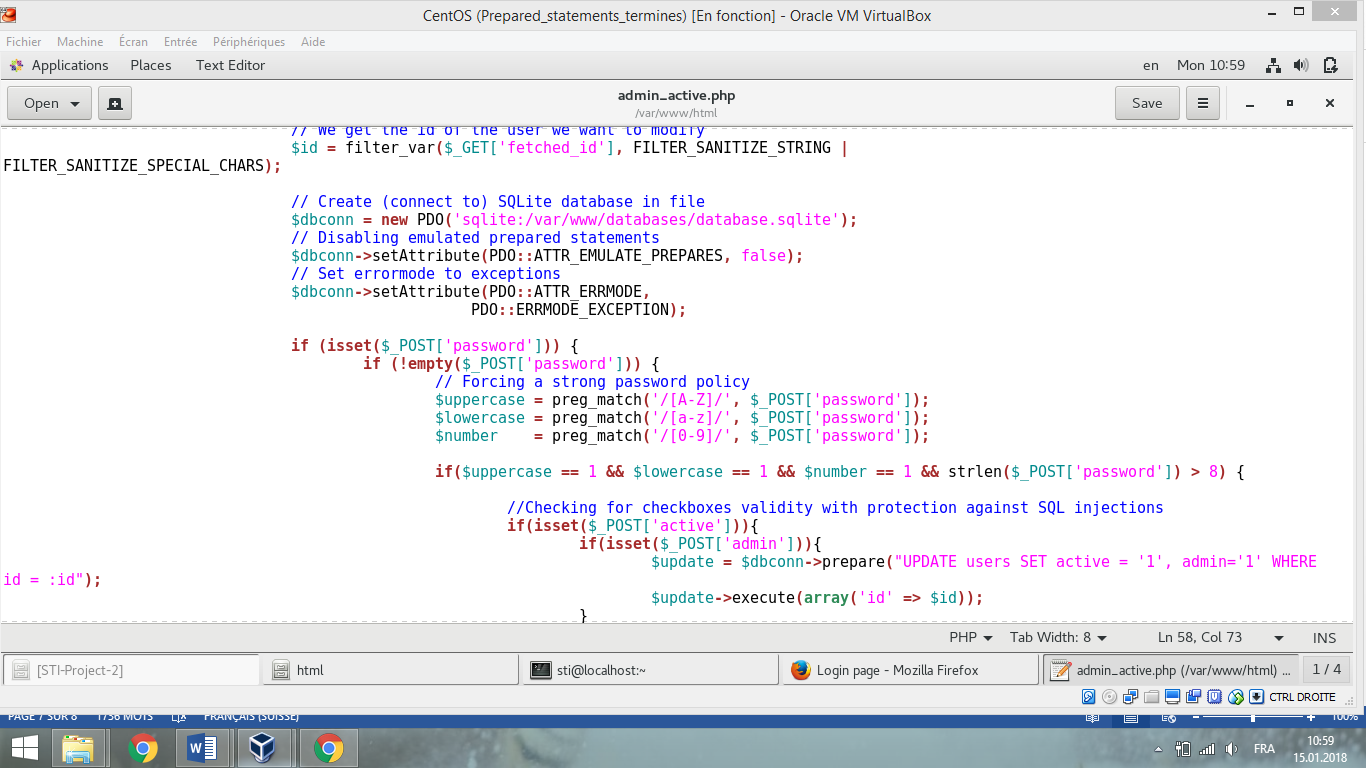
# Identification des contre-mesures

## Contre-mesure 1

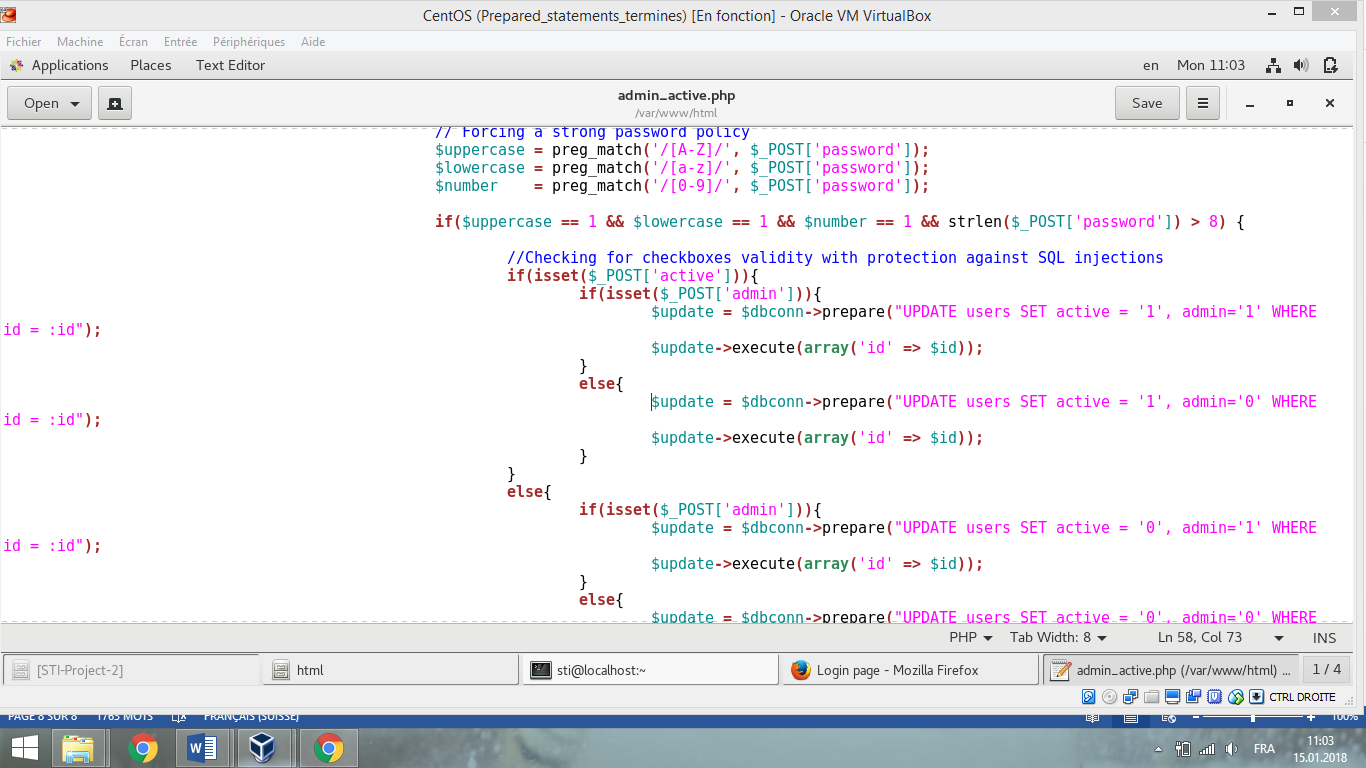
Pour se protéger des injections SQL, il faut utiliser des **« prepared statements »** (ou déclaration préparée en français).

En utilisant des prepared statements, la requête et les données sont envoyées à la base de données de façon séparée. L’attaquant ne peut plus mixer le code SQL et les données pour le modifier à sa guise et ainsi pouvoir injecter des codes SQL malicieux.

Sur chaque page PHP où l’on doit se protéger contre des injections SQL, il faut tout d’abord désactiver les « emulated prepared statements ».



Ensuite, il faut préparer la requête (**->prepare**) pour ensuite l’exécuter en matchant les variables de la requête avec les variables PHP (**->execute(array(‘varSQL’ => $varPHP)**).



## Contre-mesure 2

# Conclusion

Notre application de messagerie est maintenant sécurisée mais elle comporte encore quelques faiblesses que nous n’avons pas réussi à corriger.

Pas de sel avec le hash

Les id ne sont pas générés aléatoirement (incrémentation +1)

Protection contre la vulnérabilité CSRF pas mise en place 🡪 générer un token pour chaque requête dans une session active