

Sécurité des Applications Web

Élèves: Achraf THABET, Agustin ZORZANO

FISE A2

Exercice 1

Selon le code, on peut voir qu'il s'agit d'une application Web qui contient deux endpoints:

- `/`: endpoint root de la page. Il montre seulement un message de bienvenue.
- `/employee`: endpoint qui obtient le paramètre "cmd" et exécute une fonction en fonction de sa valeur (list ou add) et selon ce qu'il exécute, il aura besoin de paramètres supplémentaires.

On peut voir qu'il a une fonctionnalité principale, crypter les données. En utilisant le endpoint "employee" et en exécutant la fonction add, le nom d'un fichier et les données du fichier lui sont transmis. Le programme créera un nouveau fichier dont le contenu sera les données cryptées. La fonction de list permet de vérifier si le fichier existe ou non.

Dans ce petit programme, on a pu trouver certains défauts.

1. La clé de cryptage est écrite dans le fichier. Ce n'est pas sûr, il doit être caché (il peut être défini par exemple comme variable d'environnement dans un fichier .env)
2. La commande list exécute une commande bash pour vérifier si le fichier existe ou non. Le problème est qu'il ajoute le nom du fichier transmis par l'utilisateur sans le vérifier. Par conséquent, l'utilisateur pourrait exécuter d'autres types de commandes bash en ajoutant ";". Par exemple, on peut envoyer ce qui suit en tant que paramètre ssn: "filename; rm -r *". Ce faisant, le programme exécutera ls, puis essaiera de supprimer tous les fichiers du répertoire local. Comme le donné retourné est le output de la commande, l'attaquant peut effectuer une analyse de l'ordinateur (peut utiliser pwd, ls, se déplacer de l'adresse avec cd, etc.)
3. La bibliothèque md5 est obsolète et celle recommandée doit être utilisée. Md5 est également vulnérable (une collision de hash a été détectée)

Exercice 2

Notre projet

Version vulnérable

Notre version vulnérable consiste à réaliser une application contenant deux types de vulnérabilités: XSS et SQL injection. Pour cela nous avons fait une application en Python avec Flask qui contient 3 vues. Une page de login, une page d'inscription et la page d'accueil (accessibles uniquement si vous êtes connecté).

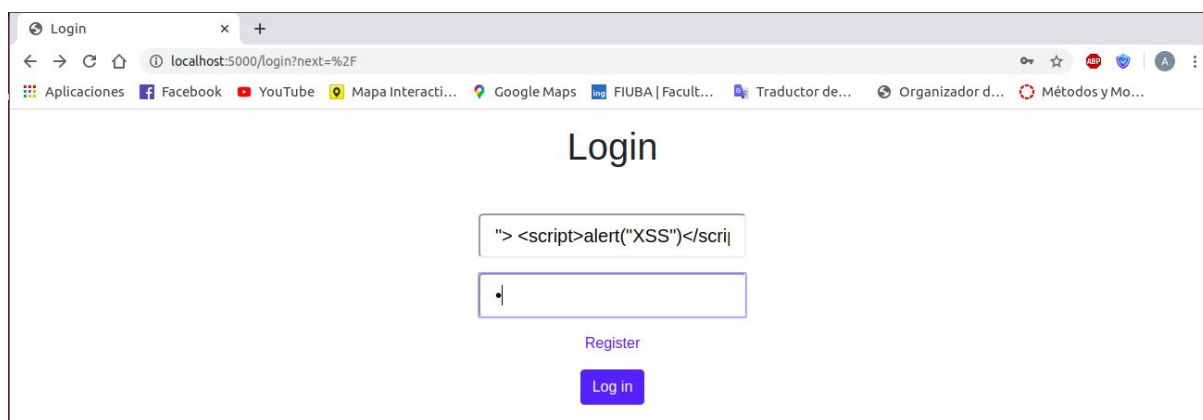
Une attaque XSS peut être effectuée sur les 3 pages. Sur la page de login et d'enregistrement, si l'utilisateur tape par exemple: «"» <script> alert (" XSS ") </script>» et ne parvient pas à se connecter / s'inscrire, alors il verra une alerte dont le texte dit "XSS". Si un utilisateur s'inscrit avec «"» <script> alert (" XSS ") </script>» comme nom d'utilisateur, puis sur la page d'accueil, il verra cette alerte à chaque fois qu'il charge la page.

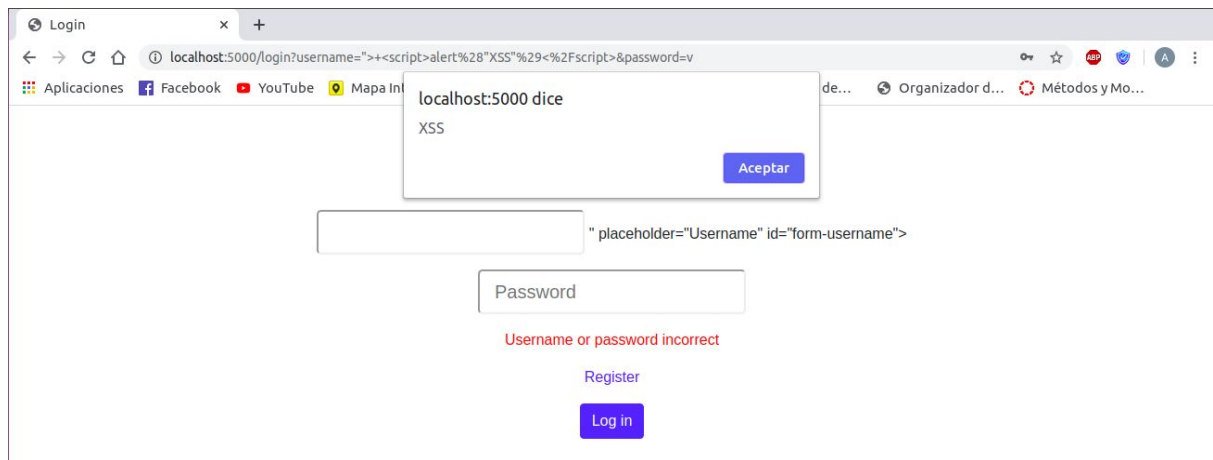
Une attaque par SQL injection peut être effectuée sur la page de connexion. L'utilisateur peut se connecter sans avoir de compte. Si, par exemple, vous saisissez «' or '1'='1'--» comme nom d'utilisateur, vous pourrez vous connecter quel que soit le mot de passe saisi.

Dans une autre branche, il existe une autre version qui permet un autre type d'attaque par SQL injection. Lors de l'inscription, du code SQL peut être ajouté dans le mot de passe. L'ajout par exemple de «a'); DELETE FROM user WHERE username = 'user1'; --» supprimera l'utilisateur user1.

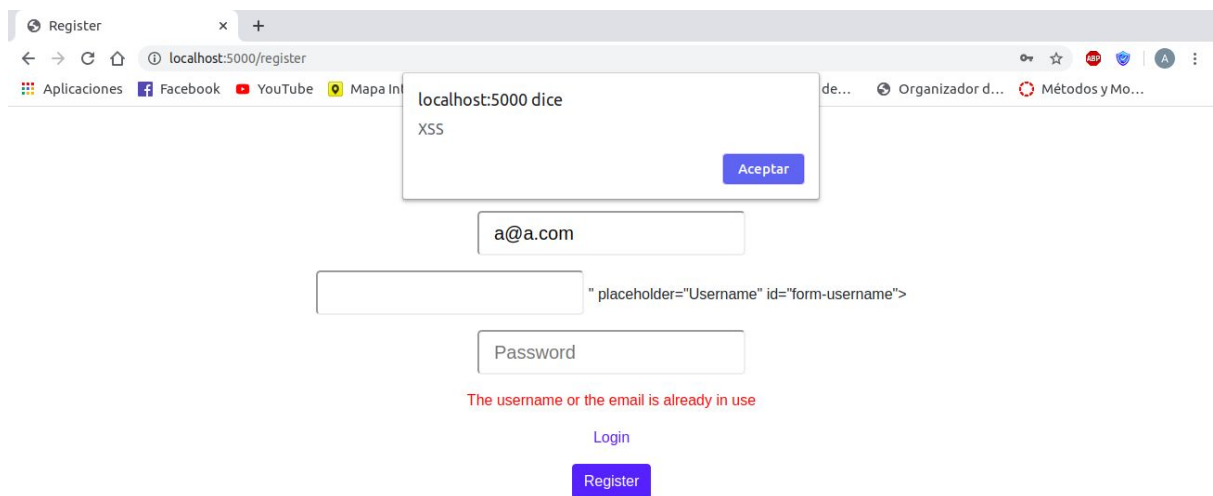
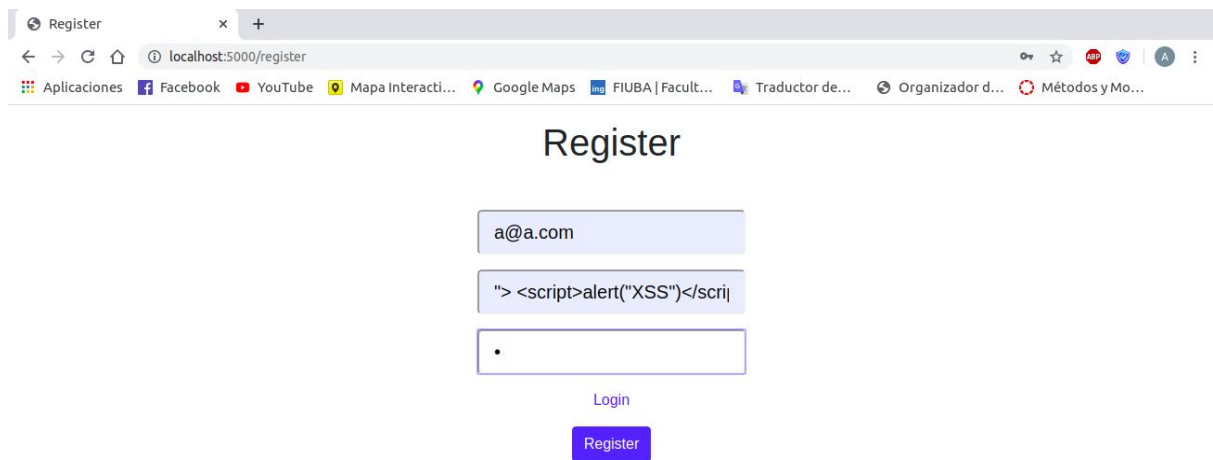
Voici les défauts:

- Attaque XSS sur la page de login

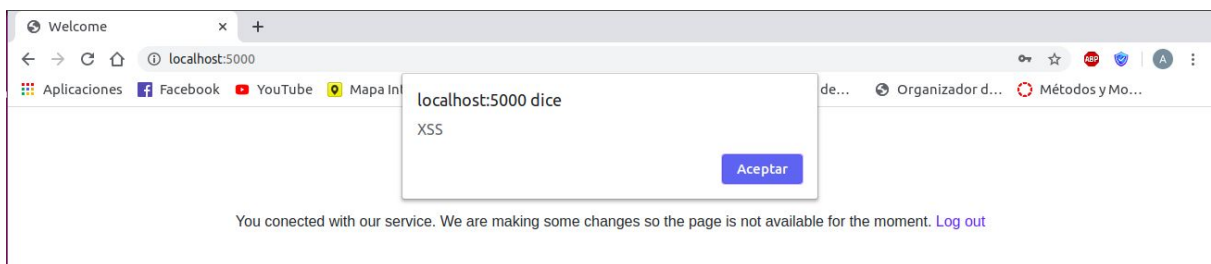
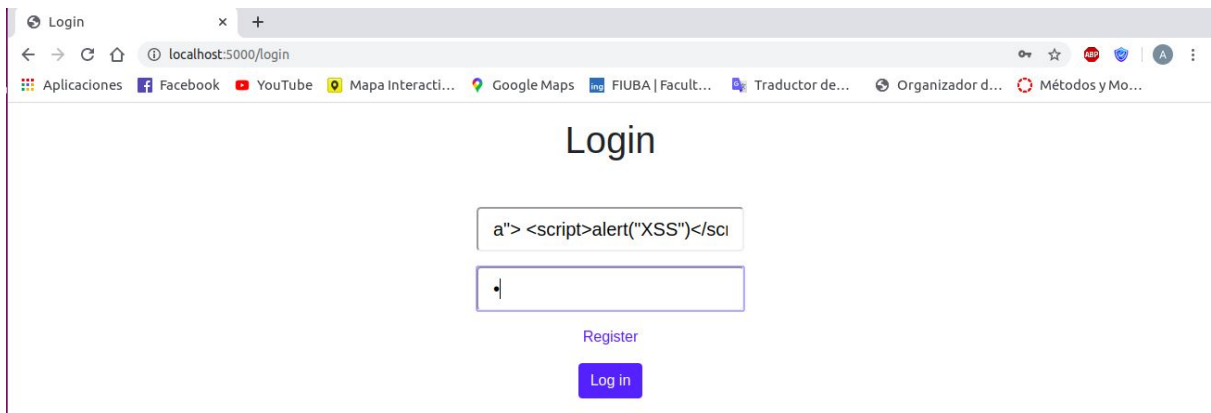




- Attaque XSS sur la page d'enregistrement

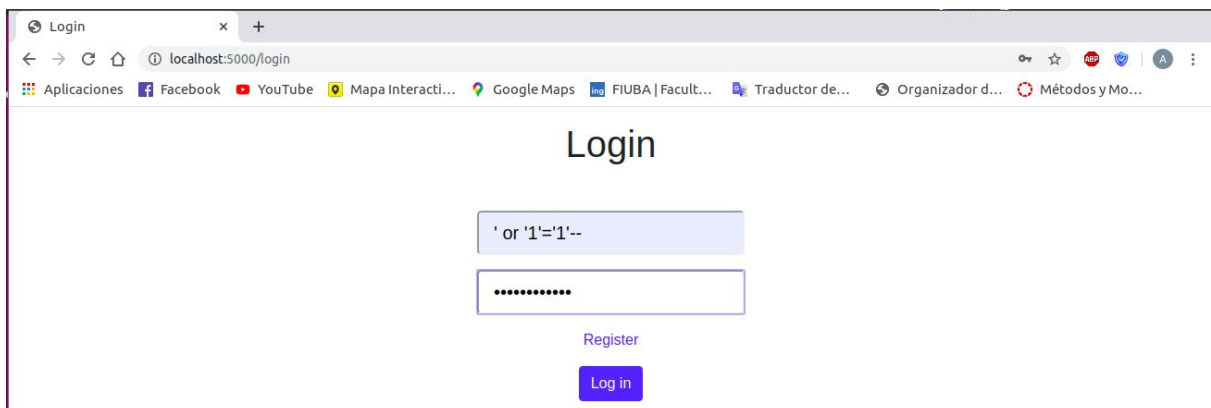


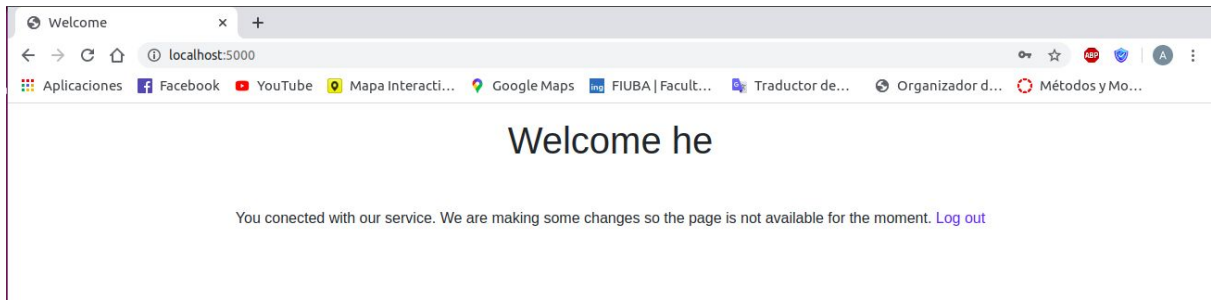
- Attaque XSS sur la page d'accueil, après connexion



On peut voir que le nom d'utilisateur n'est pas complet car il a été pris comme code et exécuté.

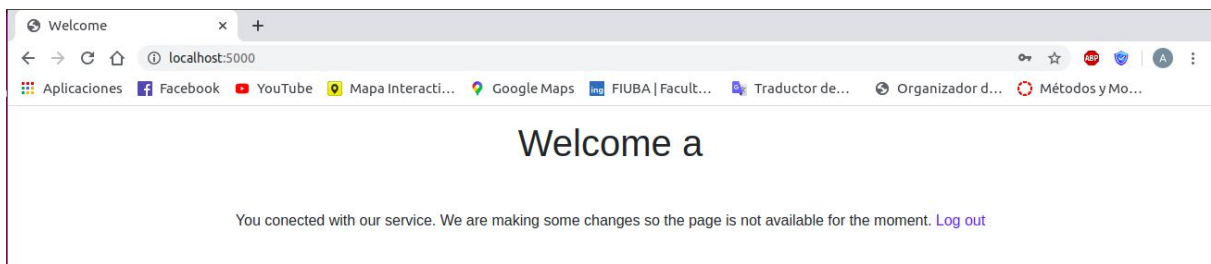
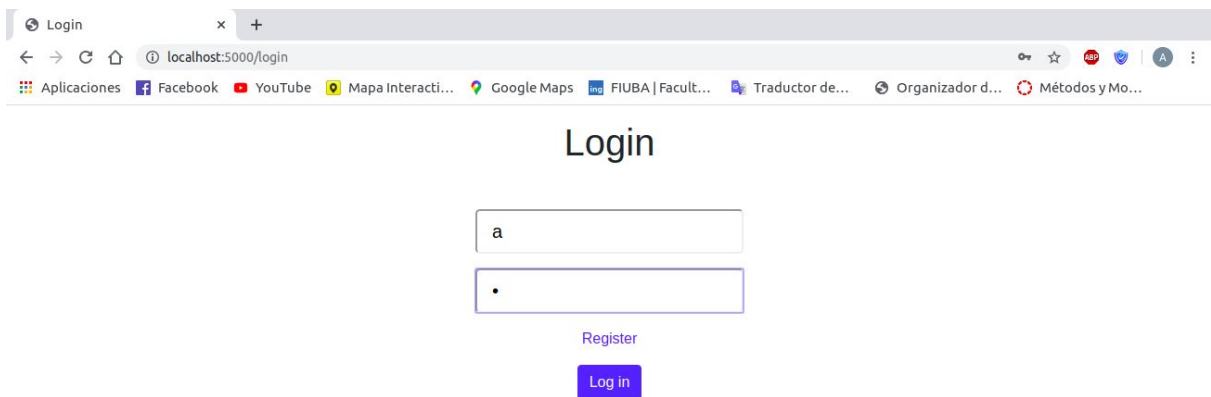
- Attaque par injection SQL sur la page de connexion

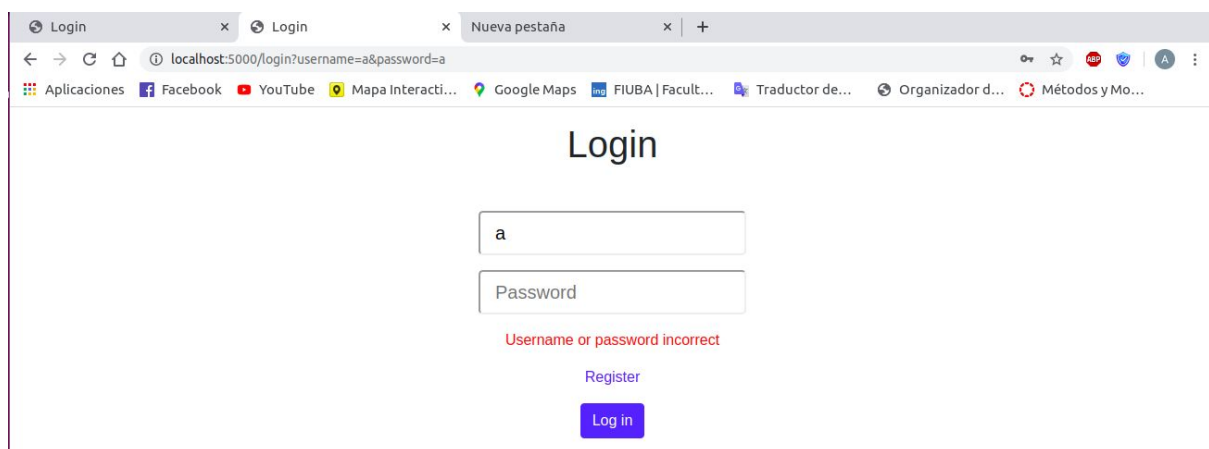
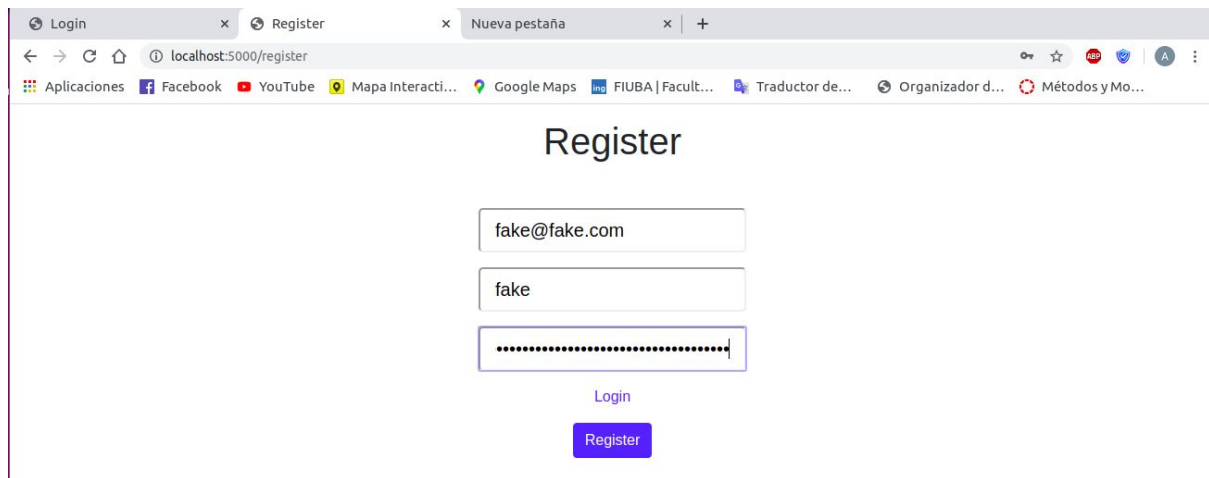




On peut voir que le nom d'utilisateur est "he" et non ce qui a été placé lors de la connexion

- Attaque par injection SQL où nous supprimons un utilisateur (effectuée sur l'autre branche). Le mot de passe utilisé était «a»); DELETE FROM user WHERE username='a'; --»





On peut voir que on ne peux plus accéder avec l'utilisateur "a" car il a été supprimé de la base de données.

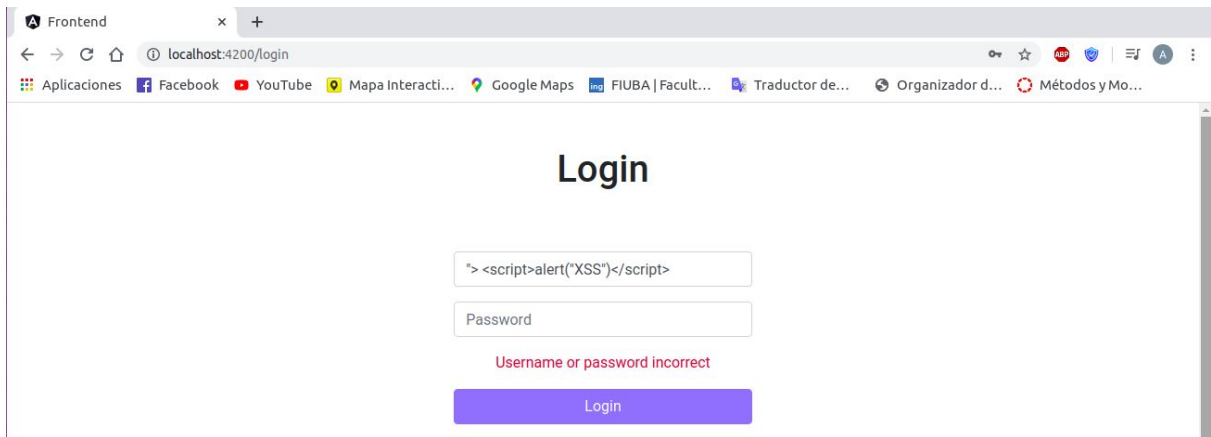
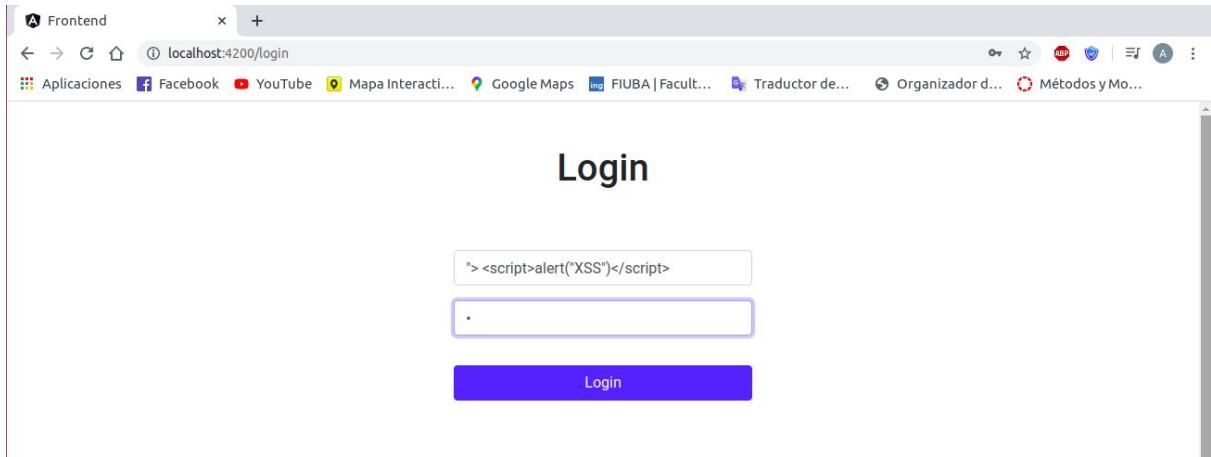
Version corrigée

Notre version corrigée consiste à réaliser la même application sans les vulnérabilités. Pour cela nous avons fait une application en Python avec Flask pour le backend et nous avons utilisé Angular pour le frontend.

Grâce à Angular, nous avons pu éliminer les attaques XSS de toutes les pages et, en utilisant correctement la bibliothèque SQLAlchemy, nous éliminons les attaques par SQL injection.

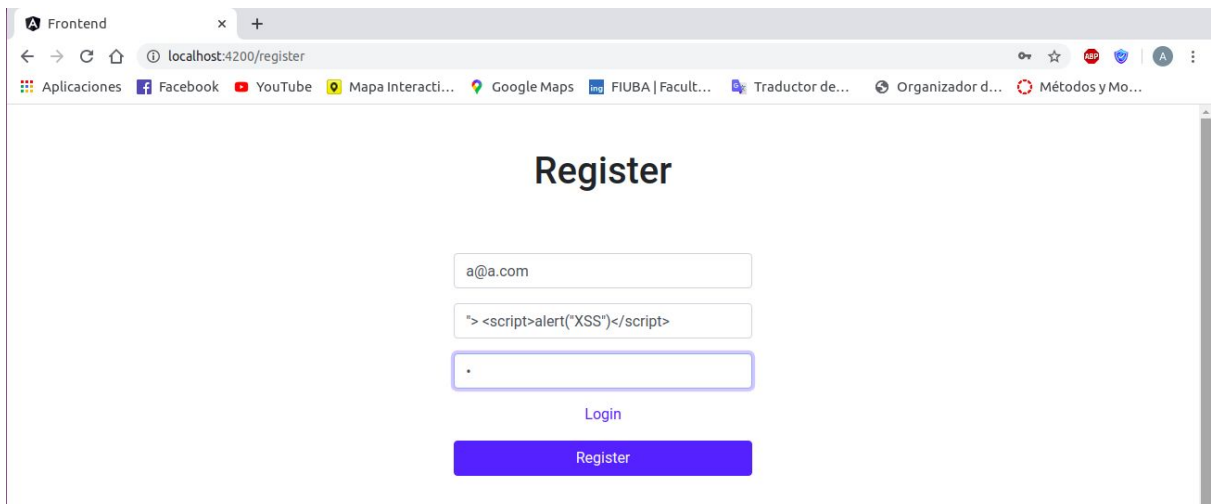
Voici les tentatives pour effectuer les mêmes attaques que celles indiquées ci-dessus

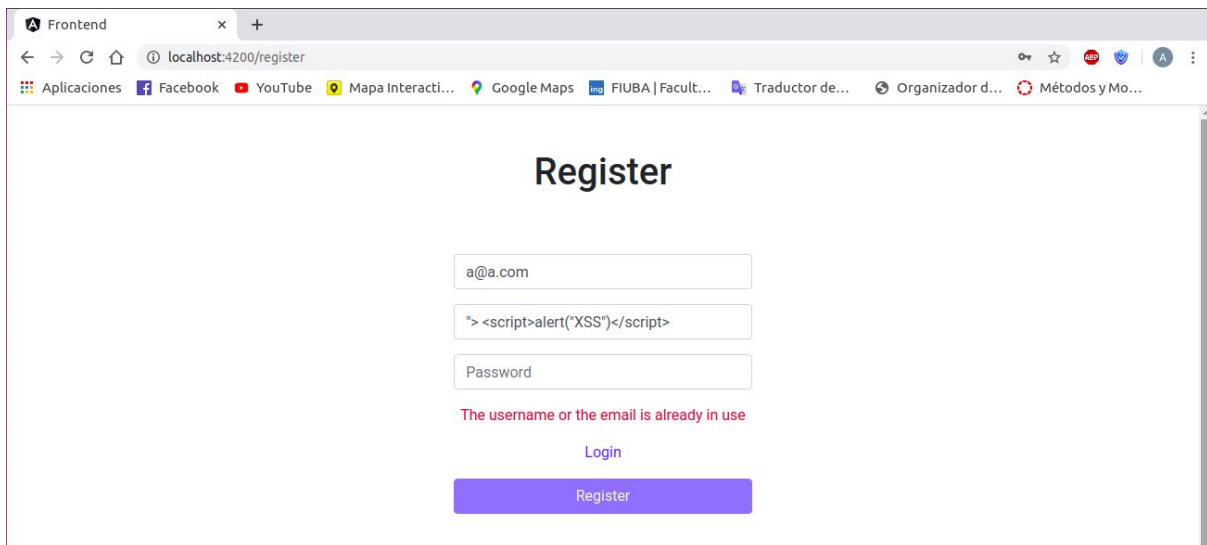
- Attaque XSS sur la page de login



Comme vous pouvez le voir, nous n'avons pas reçu l'alerte.

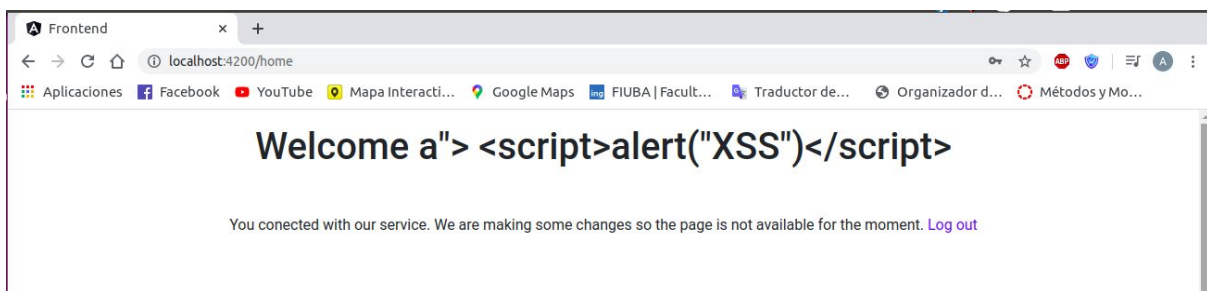
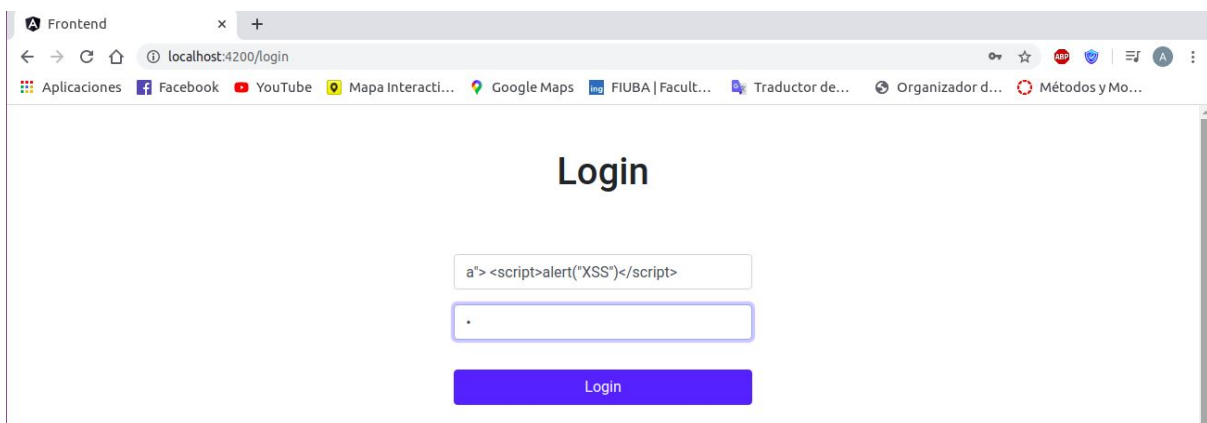
- Attaque XSS sur la page d'enregistrement





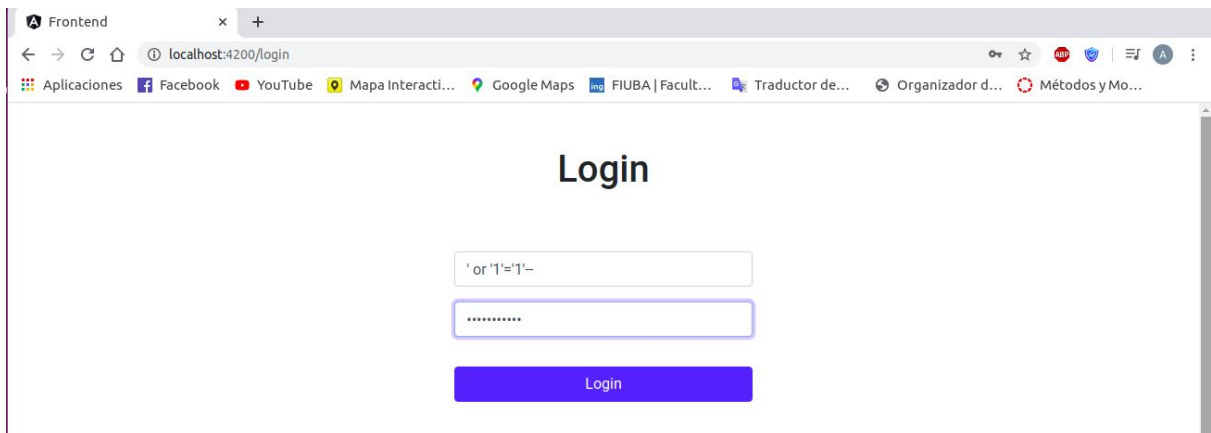
Comme vous pouvez le voir, nous n'avons pas reçu l'alerte.

- Attaque XSS sur la page d'accueil, après connexion

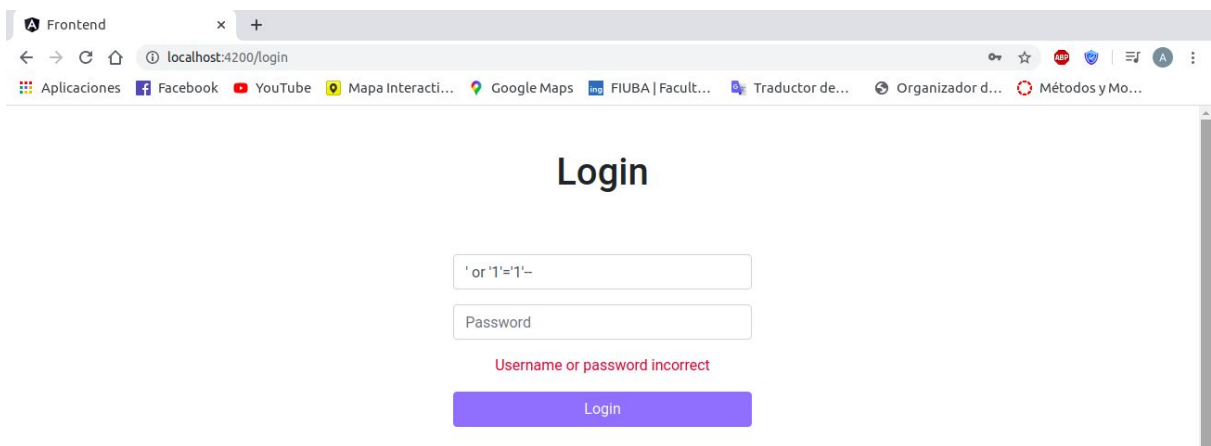


Comme vous pouvez le voir, nous n'avons pas reçu l'alerte et le nom d'utilisateur apparaît en entier dans le titre de la page.

- Attaque par injection SQL sur la page de connexion



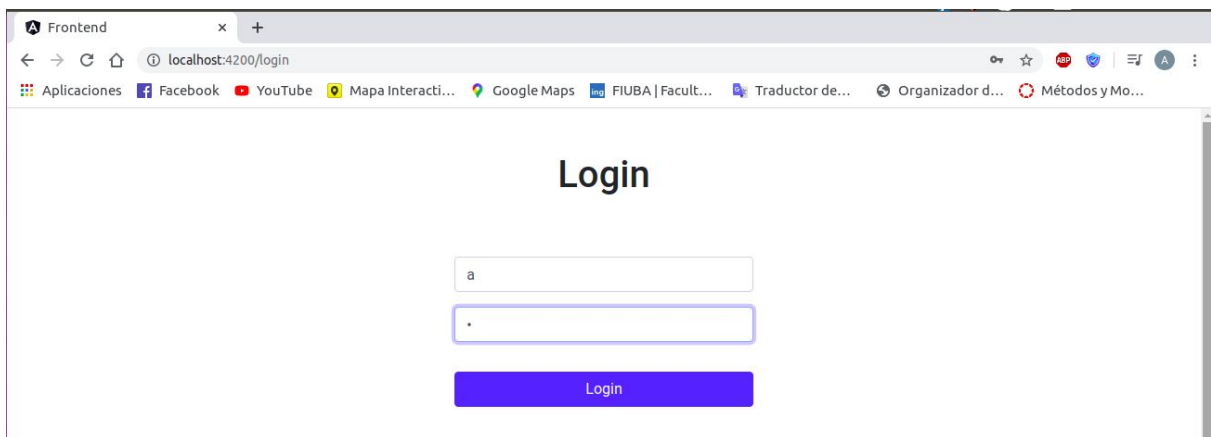
A screenshot of a web browser window titled 'Frontend' with the address bar showing 'localhost:4200/login'. The page has a title 'Login' and two input fields. The first input field contains the SQL injection payload `' or '1'='1'--`. The second input field contains a series of dots representing a password. A blue 'Login' button is at the bottom. The browser's address bar and tabs are visible at the top.



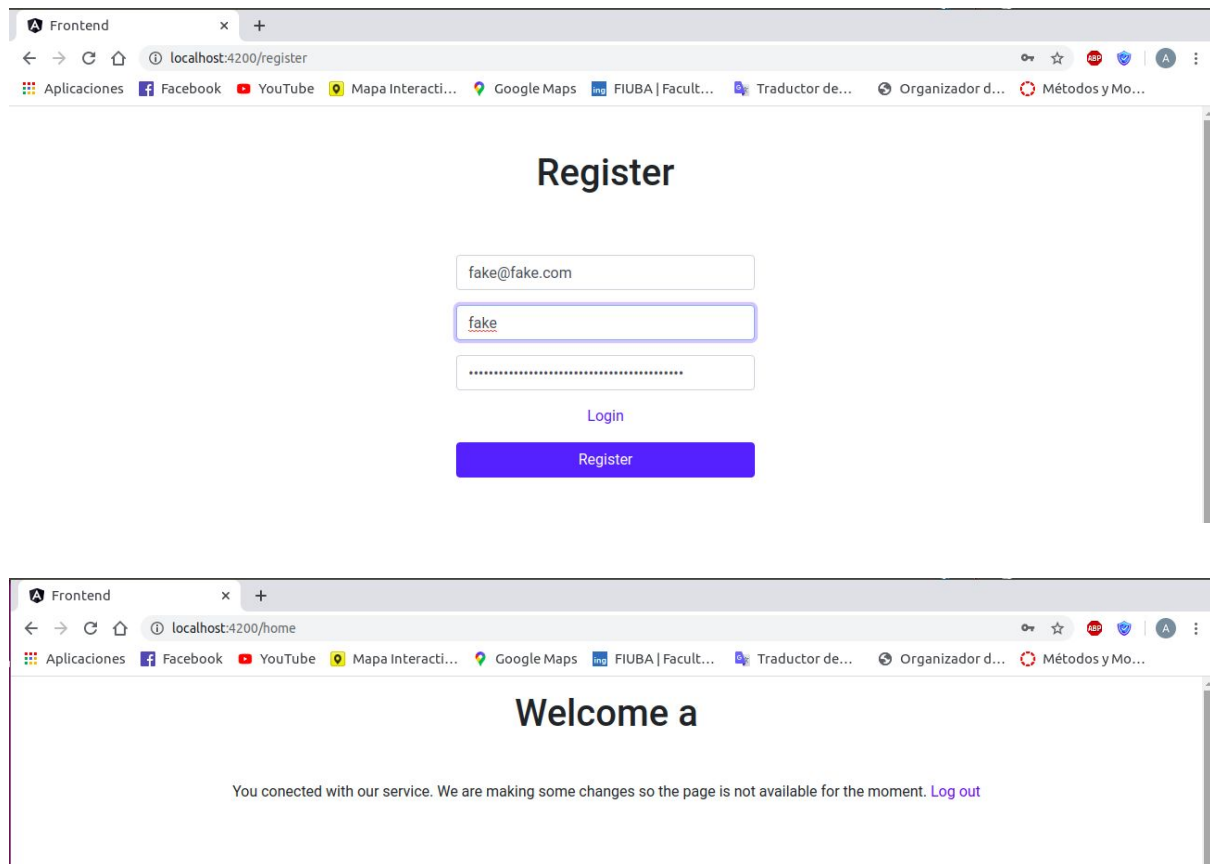
A screenshot of a web browser window titled 'Frontend' with the address bar showing 'localhost:4200/login'. The page has a title 'Login' and two input fields. The first input field contains the SQL injection payload `' or '1'='1'--`. The second input field is labeled 'Password'. Below the input fields, a red error message reads 'Username or password incorrect'. A blue 'Login' button is at the bottom. The browser's address bar and tabs are visible at the top.

On voit qu'il n'était pas possible d'y accéder.

- Attaque par injection SQL où nous supprimons un utilisateur (effectuée sur l'autre branche). Le mot de passe utilisé était «a'); DELETE FROM user WHERE username='a'; --»



A screenshot of a web browser window titled 'Frontend' with the address bar showing 'localhost:4200/login'. The page has a title 'Login' and two input fields. The first input field contains the character 'a'. The second input field contains a series of dots representing a password. A blue 'Login' button is at the bottom. The browser's address bar and tabs are visible at the top.



On peut voir que on peut accéder avec l'utilisateur "a" car il n'a pas été supprimé de la base de données.

Commentaires

- L'application présentant des vulnérabilités a une autre version dans la branche "alternative_version"
- Chaque dossier contient un fichier "readme" dans lequel il peut y avoir des informations supplémentaires sur l'application et comment l'exécuter.

- Bien que nous fassions une page de connexion, nous ne nous concentrons pas spécifiquement sur sa sécurité, c'est pourquoi l'application sans vulnérabilités pourrait avoir un certain type de vulnérabilité en ce qui concerne l'échange de clés.