Devoir OWASP_MISSION1
Elijah Dabo
21/06/24

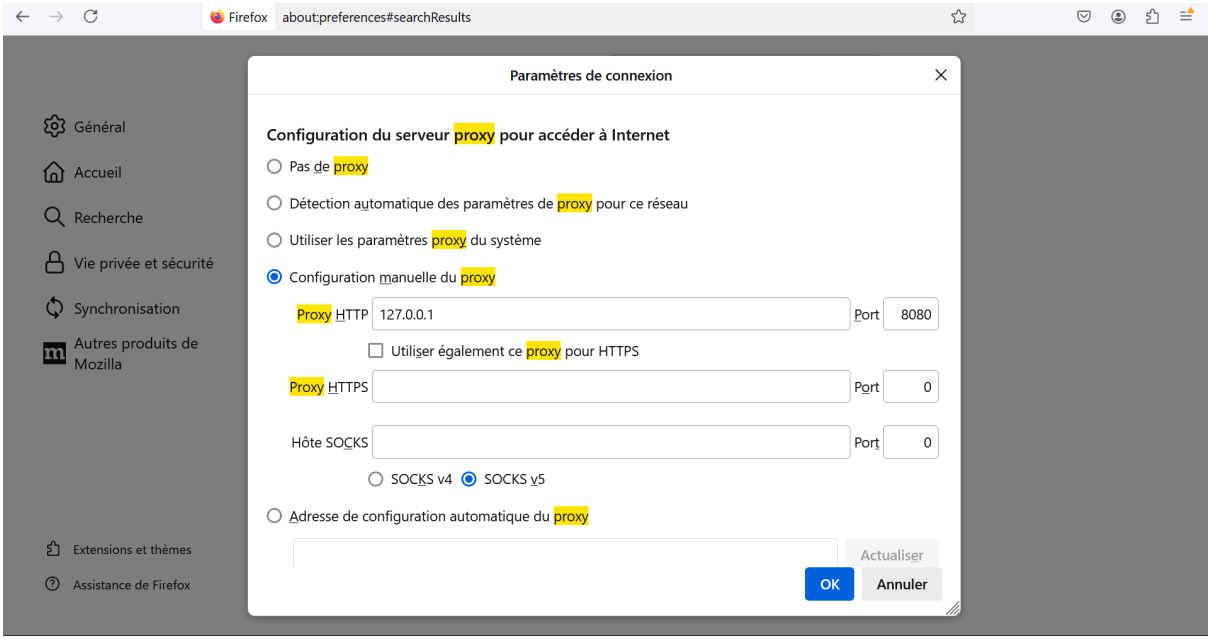
TRAVAIL A FAIRE 1 :

Q1. Créer un compte permettant de vous identifier sur la plateforme.

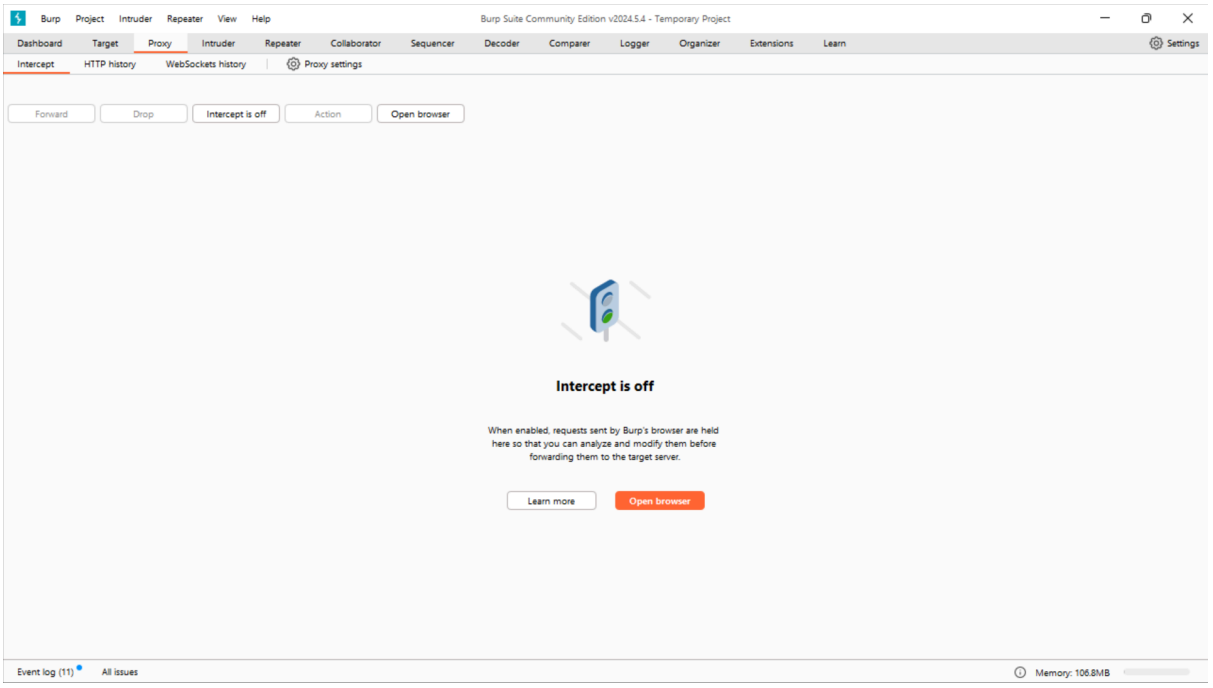
Pour commencer je crée un compte eljhdbo sur Mutillidae.
On peut voir sur le screen ci-dessous que je suis bien connecter sur mon compte, mais que je suis aussi en *Security Level : 1* que j'ai modifié en ayant cliqué sur le bouton : Toggle Security.

Q2. Utiliser une méthode de votre choix afin de découvrir les noms des champs login et mot de passe du formulaire d'authentification.

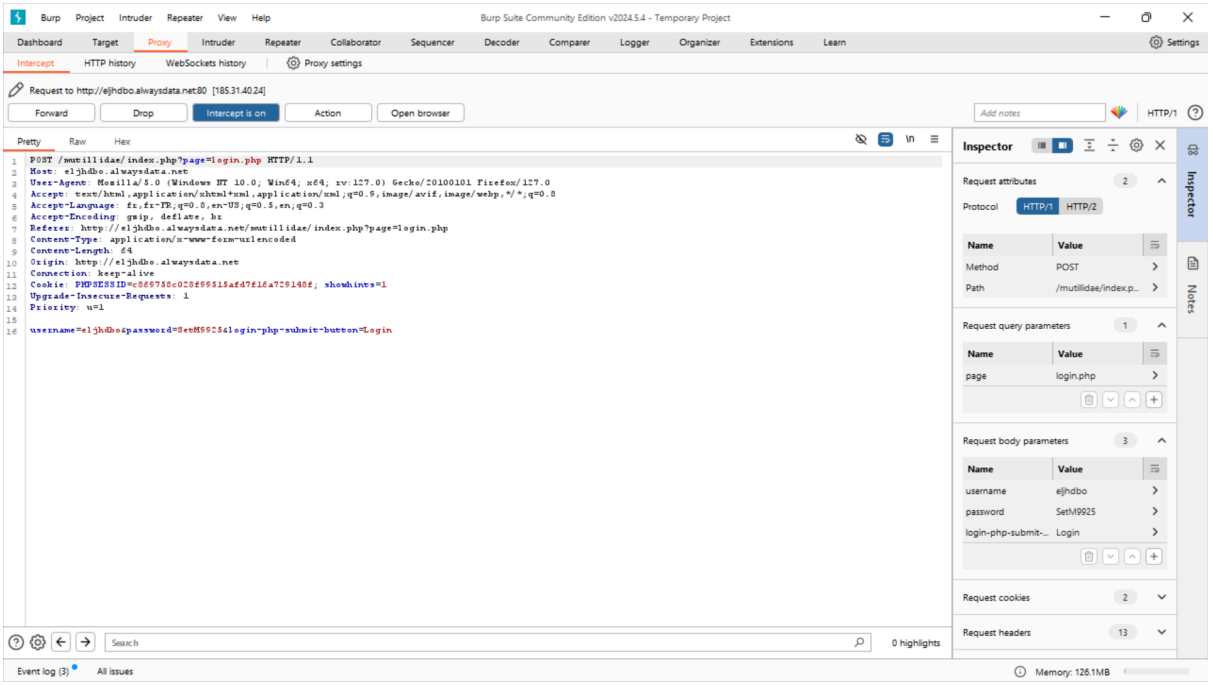
Ensuite pour découvrir les champs login et mot de passe je vais utiliser l'application BurpSuite pour inspecter le code source de ma page et pour le faire j'ai décidé de changer de navigateur pour la configuration du Proxy de Burp Suite plus simple (Changement de Google Chrome à Mozilla Firefox), en configurant le serveur proxy dans les options de Firefox (screen ci-dessous).



Une fois le serveur proxy configuré je suis aller vérifier dans l'application BurpSuite que le proxy était bien désactivé avec *Intercept is off*



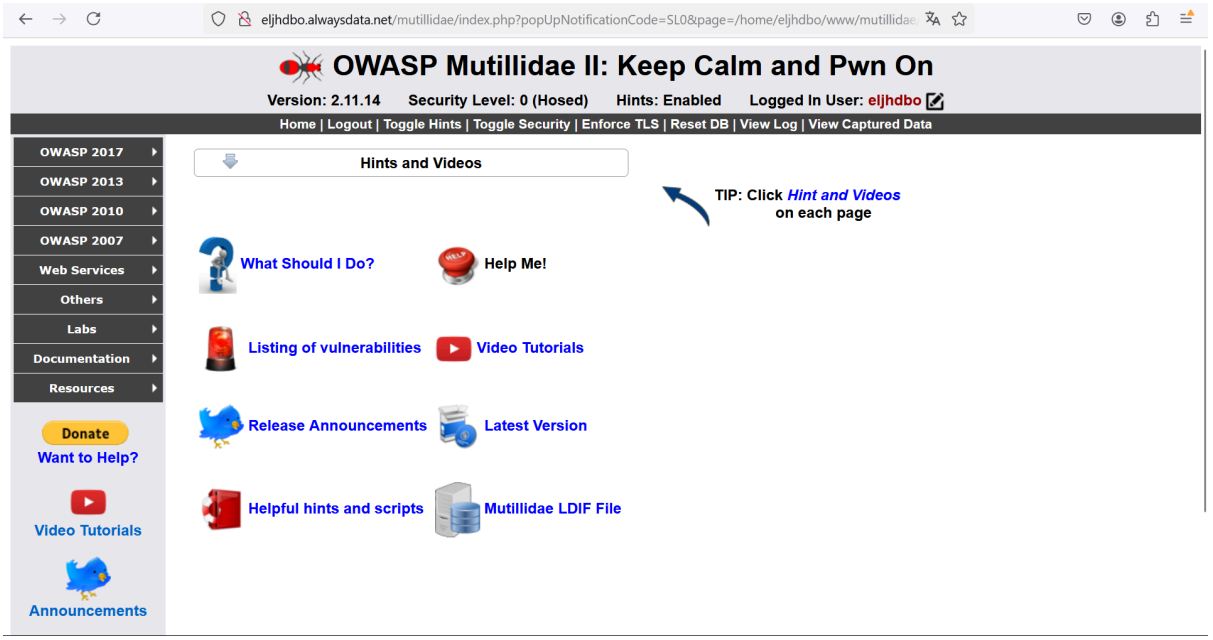
Puis sur Mutillidae dans *Login/Register* j'ai rentrer mon *username* et *password* sans valider (appuyer sur le bouton *login*) et ensuite dans BurpSuite faire passer le *Intercept is off* en *Intercept is on*, enfin repartir sur mutillidae appuyer sur le bouton *login* pour voir s'afficher dans BurpSuite :



On voit bien sur le screen ci dessus le noms des champs :
Nom d'utilisateur (`username`) : La valeur capturée est `eJjhdbo`.
Mot de passe (`password`) : La valeur capturée est `SetM9925`.
Bouton de soumission (`login-php-submit--button`) : La valeur capturée est `Login`

Q3. Positionnez le niveau de sécurité du code à 0 (Hosed).

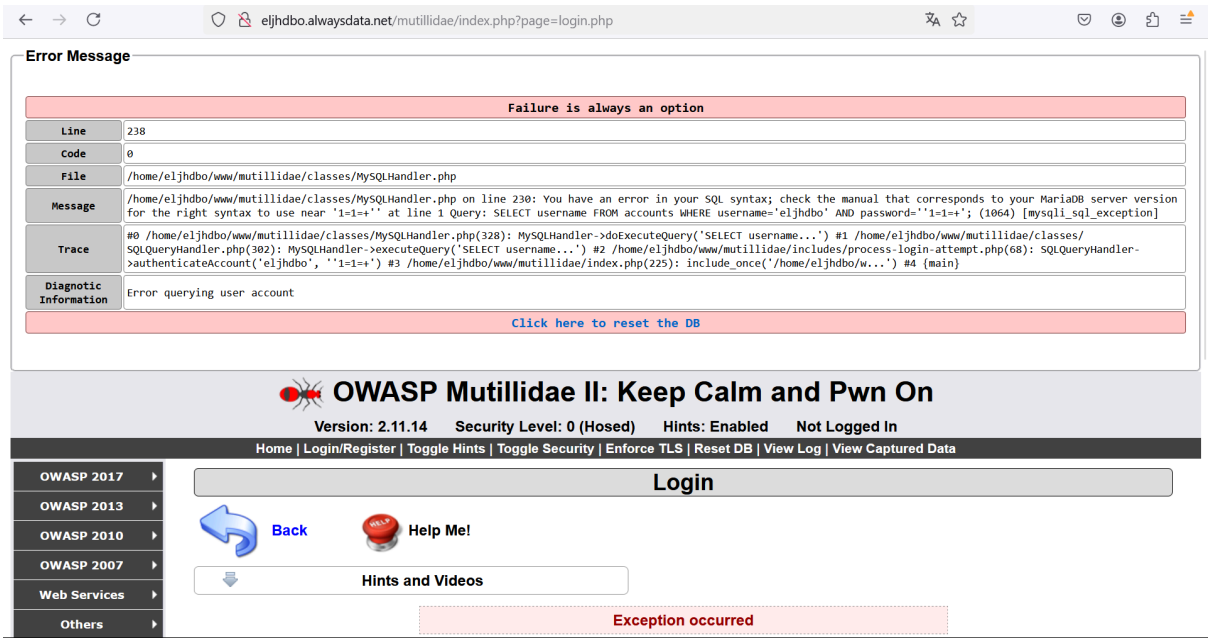
Pour positionner le niveau de sécurité du code à 0 il faut appuyer plusieurs fois sur le bouton Toggle Security pour (si on reste au niveau 1) passer au *Security Level : 5* et enfin *Security Level : 0*.



TRAVAIL A FAIRE 2 :

Tester une détection manuelle de la sensibilité SQLi.

Je teste la sensibilité SQLi en saisit une quote dans le champ associé au mot de passe dans mutillidae pour qu'en appuyant sur *login* une erreur apparaît pour prouver la sensibilité SQLi.

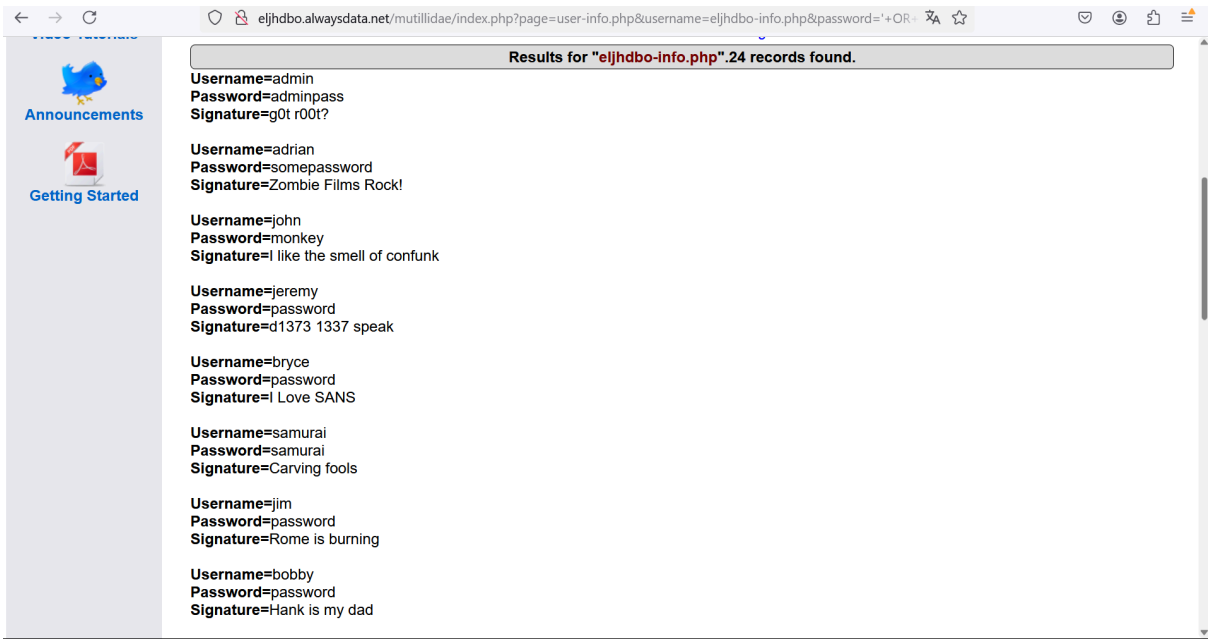


TRAVAIL A FAIRE 3 :

Q1. Réaliser les défis présentés en testant les injections suivantes ? (' or ('a' = 'a') or ' et ' or username='admin).

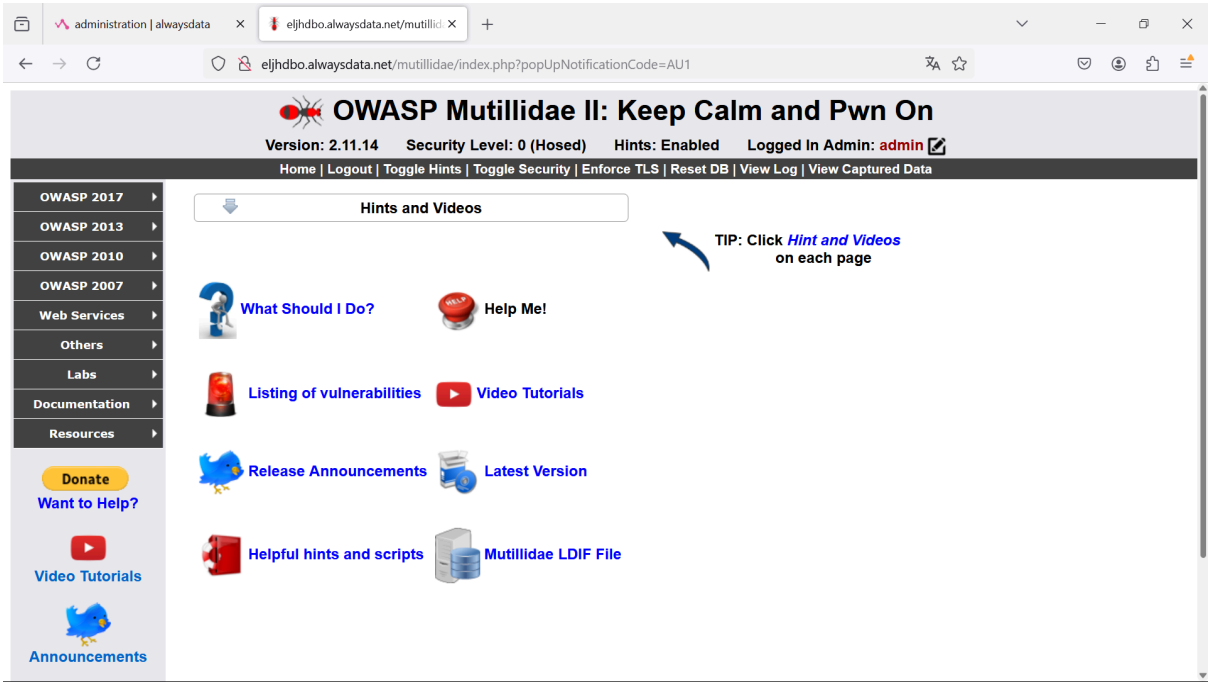
Premier Défi : Extraction de données

Pour ça il faut aller dans les sections OWASP 2017 -> A1 – Injection (SQL) -> SQLi – Extract Data -> User Info (SQL) dans mutillidae, puis utiliser le script PHP : eljhdbo-info.php et comme injection à tester : ' or ('a' = 'a') or ' , pour avoir comme résultat :



Deuxième Défi : Passer outre une authentification.

Pour ça il faut aller dans les sections OWASP 2017 -> A1 – Injection (SQL) -> SQLi – Bypass Authentication -> Login , puis utiliser le script PHP : login.php et comme injection à tester : ' or username='admin , pour avoir comme résultat :



On vient bien sur le screen ci-dessus que je suis connecté en temps qu'admin.

Q2. Reporter sur votre documentation les injections testées

Premier défi : Extraction de données

- Injection SQL utilisée : ' or ('a' = 'a') or '
- Résultat : Liste de tous les utilisateurs affichée.

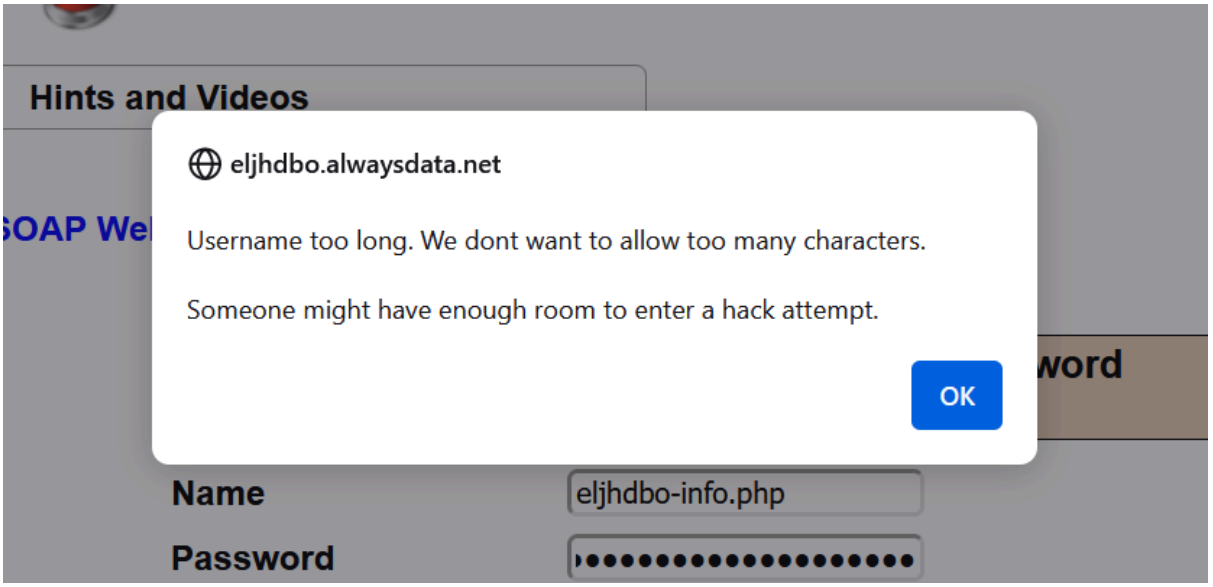
Deuxième défi : Passer outre une authentification

- Injection SQL utilisée : ' or username='admin
- Résultat : Authentification réussie en tant qu'utilisateur admin

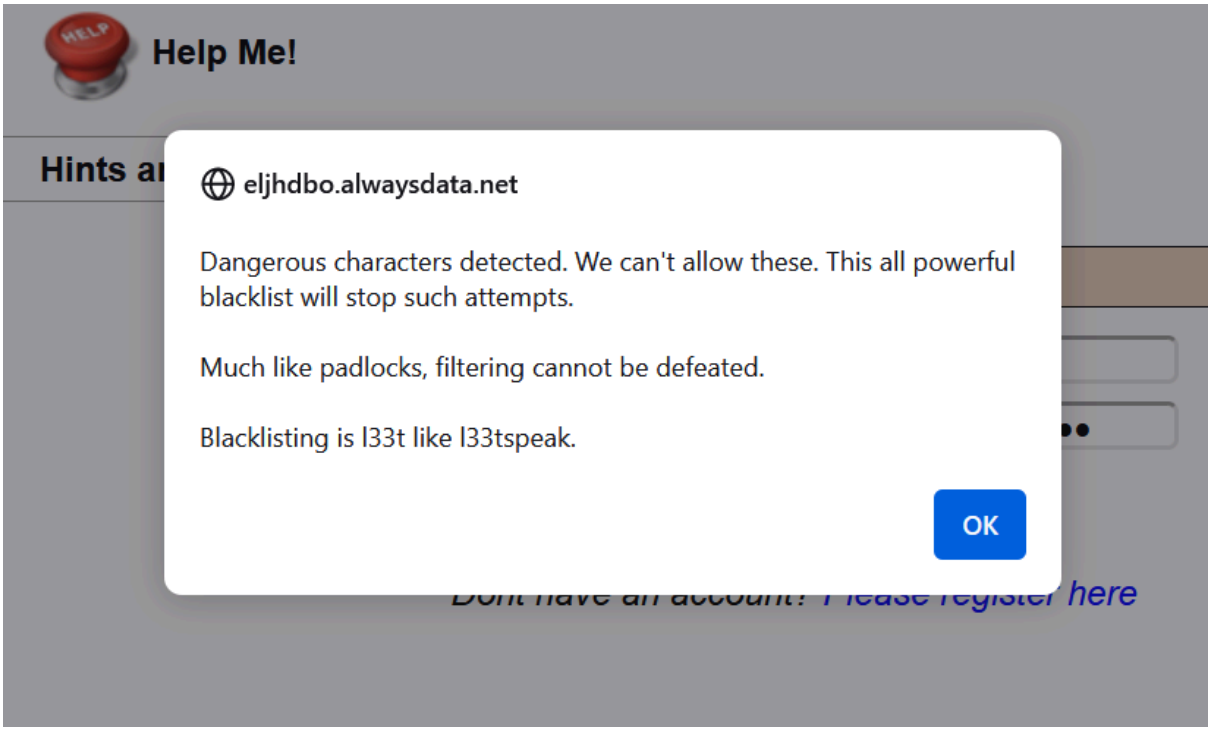
TRAVAIL A FAIRE 4 :

Q1. Modifier le niveau de sécurité et vérifier que les tests d'exploitation des failles SQLi échouent.

Pour commencer on changer le niveau de sécurité du code en appuyant sur Toggle Security pour faire passer la sécurité à *Security Level : 1* puis refaire exactement la même chose qu'à l'étape d'avant pour chaque injection ce qui nous donne :



Le screen ci dessus est en lien avec l'injection : ' or ('a' = 'a') or ' (Premier Défi)



Le screen ci dessus est en lien avec l'injection : ' or username='admin (Deuxième Défi)

Q2. Quel niveau de sécurité est nécessaire pour protéger contre cette attaque ?

Pour protéger contre les attaques par injection SQL, un niveau de sécurité côté serveur est nécessaire. Cela signifie que toutes les données saisies par l'utilisateur doivent être validées et nettoyées avant d'être utilisées dans des requêtes SQL. Ce niveau de sécurité est souvent atteint en utilisant des requêtes préparées ou des procédures stockées.

Q3. À ce niveau, quels sont les contrôles réalisés ?

À ce niveau de sécurité, les contrôles suivants sont réalisés :

- **Validation des entrées côté serveur** : Toutes les données saisies par l'utilisateur sont vérifiées pour s'assurer qu'elles ne contiennent pas de caractères dangereux.
- **Requêtes préparées** : Les requêtes SQL sont préparées à l'avance avec des paramètres, ce qui empêche les données de l'utilisateur de modifier la structure de la requête.
- **Procédures stockées** : Les opérations SQL sont encapsulées dans des procédures stockées, ce qui limite l'exposition des requêtes directes aux entrées utilisateurs.

Q4. Est-ce que le contrôle HTML aurait suffit à protéger contre cette injection SQL ?

Non, le contrôle HTML seul ne suffit pas pour protéger contre les injections SQL. Les contrôles HTML, comme les attributs `required` ou les types de champs (par exemple, `type="email"`), ne peuvent pas empêcher un attaquant déterminé de soumettre des requêtes malveillantes directement au serveur. La validation côté serveur est essentielle car elle se produit après que les données ont été envoyées au serveur, rendant les attaques plus difficiles.

Q5. Comment la validation JavaScript est-elle déclenchée ?

La validation JavaScript est déclenchée côté client, c'est-à-dire dans le navigateur de l'utilisateur, généralement lorsque l'utilisateur soumet un formulaire. Par exemple, en cliquant sur le bouton de soumission (submit) d'un formulaire, le script JavaScript s'exécute pour vérifier que les données saisies respectent les règles définies (comme la longueur des champs, le format des emails, etc.).

Q6. Quels sont les contrôles réalisés par la validation JavaScript ?

Les contrôles réalisés par la validation JavaScript sont :

- **Vérification des champs obligatoires** : Assurer que les champs requis sont remplis.
- **Format des données** : Vérifier que les données saisies respectent le format attendu (par exemple, email, numéro de téléphone).
- **Longueur des entrées** : Limiter la longueur des données saisies pour éviter les débordements de buffer.
- **Caractères spéciaux** : Empêcher l'utilisation de certains caractères spéciaux qui pourraient être utilisés pour des injections SQL ou d'autres attaques.
- **Messages d'erreur** : Afficher des messages d'erreur pour guider l'utilisateur en cas de saisie incorrecte.