

Stratégie de Sécurisation de l'application pire2pire.com



Réalisé par :

➤ Yousra Chbib

Encadré par :

➤ Cyril MARCQ
➤ Hachemi
➤ Claire VAN WYNENDAELE

I. Introduction

introduction

II. Coté Client

1. Page de Connexion

1.1 Sécurisation du nom d'utilisateur

La sécurisation du champ « nom d'utilisateur » est une étape clé dans la protection de vos systèmes. Elle doit garantir que les données saisies sont conformes à un format attendu et ne peuvent pas être utilisées pour mener des attaques telles que l'injection de code ou l'énumération de comptes existants. Voici les axes essentiels à mettre en place :

1.1.1 Validation des entrées avec des expressions régulières (Regex):

Les expressions régulières sont des outils puissants permettant de définir des motifs précis pour contrôler la validité des données saisies par l'utilisateur. Elles servent à :

- **Définir un format précis** : Par exemple, vérifier qu'un nom d'utilisateur ne contient que des lettres, chiffres et certains caractères spéciaux autorisés.
- **Valider des formats spécifiques** :
 - Pour le adresse email : `^[a-zA-Z0-9._%+-]+@gmail\.com$`
 - Pour le numéro de téléphone : `^\+?[1-9]\d{1,14}$`

Cela permet de s'assurer que l'email et/ou le numéro de téléphone suivent un schéma standard (bien que dans un cas spécifique, vous puissiez restreindre par exemple aux adresses Gmail).

1.1.2 Filtrage et nettoyage des entrées (Sanitization) :

Même après validation, il est essentiel de nettoyer la saisie afin d'éviter les injections ou d'autres attaques malveillantes :

- **Suppression des espaces inutiles** : Pour enlever les espaces en début et fin de chaîne pour éviter des erreurs ou des incohérences dans le stockage.
- **Échappement ou refus des caractères dangereux** : Interdire ou échapper des caractères comme « `<`, `>`, `;`, `--` », ou d'autres symboles susceptibles d'être exploités dans une injection **SQL** ou **XSS**.

1.1.3 Définition des Contraintes de longueur minimale et maximale :

La définition des contraintes de longueur vise à garantir un équilibre entre accessibilité et sécurité. Une exigence de longueur minimale de 6 caractères permet d'éviter les identifiants trop courts, souvent vulnérables aux attaques par dictionnaire. À l'inverse, fixer une longueur maximale, de 50 caractères, prévient les abus et réduit les risques liés aux dépassements de mémoire. Ces contraintes doivent être cohérentes avec la politique de sécurité globale de l'application et s'aligner avec les autres critères de validation et exigences métier.

1.2 La Sécurité des Mots de Passe

La gestion des mots de passe est un élément essentiel de la sécurité informatique. Cette section présente les mesures de sécurité pour assurer une protection efficace contre les attaques courantes, en mettant l'accent sur la longueur, la complexité et la gestion des mots de passe.

1.2.1 Longueur Minimale :

Un mot de passe sécurisé doit comporter au moins **12 caractères** pour résister aux attaques par **force brute**. Les mots de passe courts sont rapidement cassés, notamment par des attaques par dictionnaire et des méthodes de **calcul parallélisées**.

1.2.2 Complexité :

Pour renforcer la sécurité des comptes, il est essentiel que les mots de passe respectent des critères de complexité spécifiques. Lorsqu'un utilisateur tente de définir un mot de passe ne répondant pas à ces exigences, des messages d'avertissement clairs doivent être affichés pour guider l'utilisateur vers la création d'un mot de passe conforme.

Exemple :

L'utilisateur saisit un mot de passe composé d'un seul chiffre.

Messages d'avertissement affichés :

- **Longueur insuffisante**
- **Absence de majuscules**
- **Absence de minuscules**
- **Absence de caractères spéciaux**

Votre Compte

Pour créer votre compte d'utilisateur, vous devez choisir un mot de passe. Il doit respecter les conditions suivantes :

- ✗ Avoir une longueur comprise entre 8 et 20 caractères alphanumériques (sans accents).
- ✗ Contenir au moins 1 lettre MAJUSCULE.
- ✗ Contenir au moins 1 lettre minuscule.
- ✓ Contenir au moins 1 chiffre.
- ✗ Contenir au moins 1 caractère spécial de la liste suivante : "\$@&()[]{}=#!~?+/'<>%"

Choisissez votre mot de passe

Votre adresse e-mail est : silver-fr@oxatis.com

Entrez votre mot de passe :

Confirmez votre mot de passe :

CONTINUER >>

Ces messages guident l'utilisateur pour qu'il crée un mot de passe conforme aux critères de sécurité recommandés, renforçant ainsi la protection de son compte.

1.2.3 Interdiction des Informations Personnelles :

Pour renforcer la sécurité des mots de passe, il est essentiel d'éviter l'utilisation d'éléments facilement devinables tels que les noms, prénoms, dates de naissance, noms d'utilisateur ou des mots courants comme "password123". Dans ce cadre, des **vérifications** seront mises en place côté serveur pour comparer les mots de passe saisis avec une liste de mots interdits. Cette approche garantit que les utilisateurs ne peuvent pas définir de mots de passe contenant des informations personnelles ou des termes trop simples, renforçant ainsi la robustesse globale des mots de passe utilisés.

1.2.4 Vérification des Fuites de Données :

Pour renforcer la sécurité des mots de passe, il est essentiel de vérifier si ceux-ci ont été compromis lors de fuites de données précédentes. On a mis en place des vérifications côté serveur qui comparent les mots de passe saisis avec des bases de données de mots compromis, telles que **Have I Been Pwned**.

Exemple d'intégration en Python :

```
import requests
import hashlib

def check_password(password):
    sha1_hash = hashlib.sha1(password.encode()).hexdigest().upper()
    prefix, suffix = sha1_hash[:5], sha1_hash[5:]
    response = requests.get(f"https://api.pwnedpasswords.com/range/{prefix}")

    return suffix in response.text

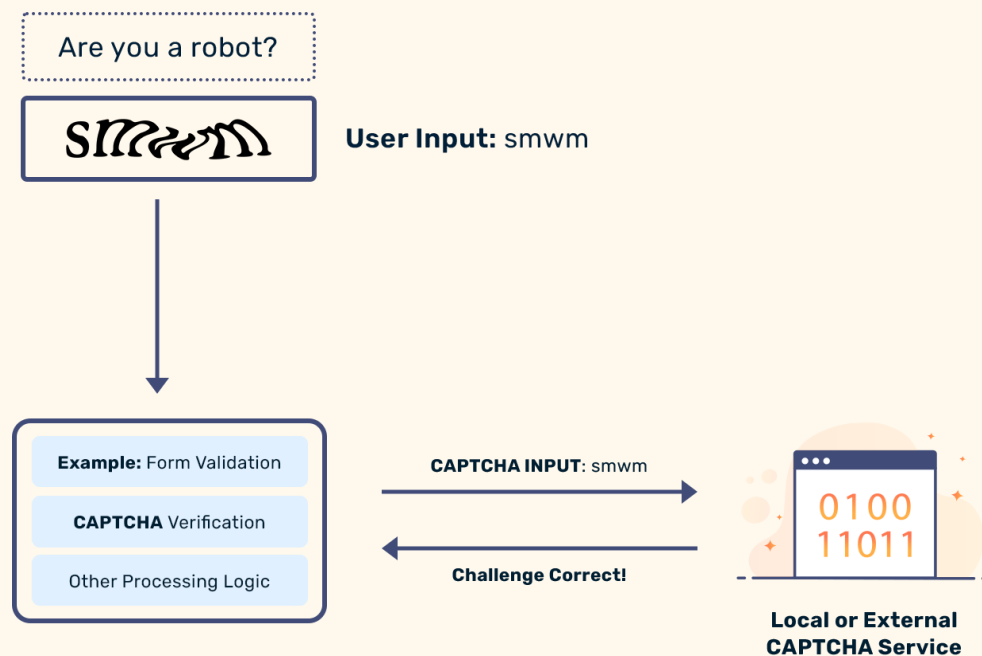
password = "monMotDePasseSuperSecret"
if check_password(password):
    print("Ce mot de passe a déjà été compromis !")
else:
    print("Ce mot de passe est sûr.")
```

1.3 Sécurisation du Formulaire d'Authentification

Sécuriser le formulaire d'authentification avant qu'il n'atteigne le serveur est essentiel pour protéger les informations sensibles des utilisateurs. Cette approche vise à prévenir diverses menaces, notamment les attaques par injection et la falsification de requêtes intersites (**CSRF**), qui ciblent souvent les formulaires comme points d'entrée vulnérables.

1.3.1 Implémentation de Systèmes CAPTCHA :

Pour protéger le formulaires contre les soumissions automatisées par des bots malveillants, nous intégrons des **systèmes CAPTCHA**. Ces tests permettent de différencier les utilisateurs humains des programmes automatisés, réduisant ainsi les risques d'abus et d'attaques par force brute.

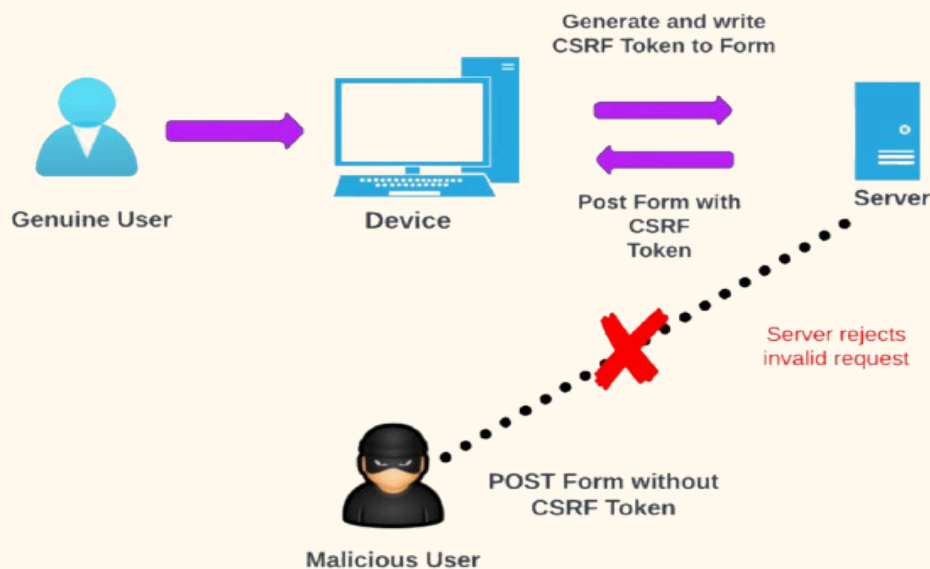


Fonctionnement d'un Test CAPTCHA pour la Protection des Formulaires

1.3.2 Utilisation de Jetons Anti-CSRF :

Une attaque CSRF est une menace sérieuse, qui se produit lorsqu'un utilisateur authentifié d'un site web est amené, à son insu, à exécuter une action non désirée sur ce site. Par exemple, un pirate pourrait envoyer une requête à votre insu, utilisant vos informations d'authentification pour exécuter une action que vous n'avez pas autorisée, comme modifier votre mot de passe ou effectuer un transfert bancaire.

C'est pour cela qu'on va utiliser la méthode de **tokens CSRF**. Un token CSRF est un jeton unique et aléatoire qui est généré par le serveur et inclus dans les formulaires HTML. Lors de la soumission du formulaire, ce token est vérifié par le serveur pour s'assurer qu'il correspond à celui stocké dans la session de l'utilisateur. Si le token ne correspond pas, la requête est rejetée.



1.4 Sécurisation d'Authentification

L'authentification est le processus qui suit la soumission du formulaire de connexion. C'est là où le serveur valide l'identité de l'utilisateur et lui accorde (ou non) l'accès à

l'application. A ce stade, plusieurs menaces peuvent compromettre la sécurité de la plateforme, notamment les attaques par **force brute**, le vol de sessions ou la divulgation d'informations sensibles.

1.4.1 Protection contre les attaques par force brute :

Les attaques par force brute consistent à tester une multitude de combinaisons de mots de passe jusqu'à en trouver un valide. Pour limiter ces tentatives :

- **Limiter le nombre d'essais** : Ajouter un délai entre les tentatives (ou bloquer un compte après un certain nombre d'échecs)
- **CAPTCHA après plusieurs échecs** : Empêche les bots d'essayer des milliers de combinaisons automatiquement.

1.4.2 Gestion des jetons d'authentification :

Les jetons d'authentification permettent aux utilisateurs de s'authentifier sans stocker leur session côté serveur.

→ **Choix du type de jeton** :

WT (JSON Web Token) est un jeton d'authentification compact et sécurisé. Les informations nécessaires à l'authentification de l'utilisateur sont contenues directement dans le token lui-même. Ainsi, contrairement aux sessions classiques qui requièrent un stockage côté serveur pour maintenir l'état de l'utilisateur, le JWT dispense en grande partie de ce mécanisme de stockage.

1.4.3 Gestion des sessions :

La gestion des sessions est essentielle pour sécuriser l'accès aux ressources après l'authentification. Une session mal gérée peut exposer les utilisateurs à des attaques comme le vol de session, l'usurpation d'identité ou les attaques de fixation de session.

Mécanismes mis en place pour sécuriser les sessions :

- Définition d'un **timeout** après une période d'inactivité (15-30 min).
- Fermer la session après une longue durée, même en activité (24h max).
- Génération d'un nouvel identifiant de session lorsque l'utilisateur se connecte pour éviter la fixation de session.

- Restreindre l'utilisation d'une même session sur plusieurs appareils simultanément.
- Utilisation des attributs sécurisés pour les cookies de session, notamment **HTTPOnly**, **secure** et **samesite**.

HTTPOnly permet d'éviter que l'identifiant de session soit accessible aux scripts Javascript, ce qui permet de contrer certaines méthodes de vol de session.

Secure informe les navigateurs que l'identifiant de session ne doit transiter que via des canaux HTTPS, ce qui permet de contrer certaines attaques basées sur l'écoute du réseau.

Samesite qui permet de se protéger contre des attaques de type CSRF.

2. Tableau de Bord

La sécurisation d'un tableau de bord nécessite une approche rigoureuse afin de s'assurer que l'accès aux informations sensibles et les actions possibles sont contrôlés et bien protégés. Voici un ensemble de mesures à appliquer :

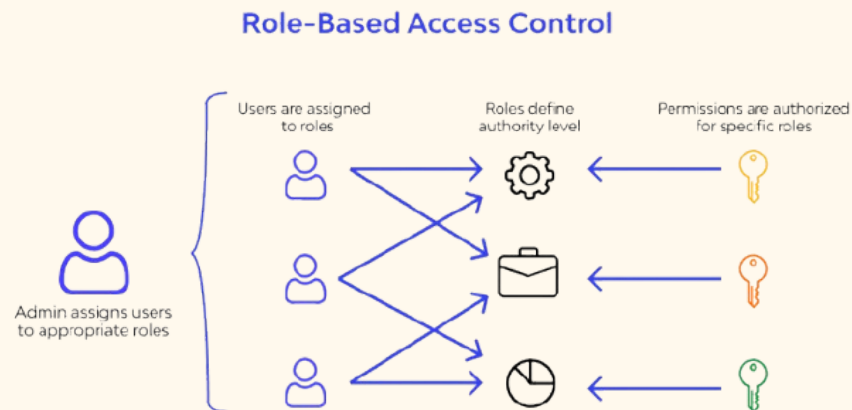
2.1 Contrôle d'accès basé sur les rôles (RBAC)

L'implémentation du contrôle d'accès basé sur les rôles (**RBAC**) est essentielle pour définir et restreindre les actions que chaque utilisateur peut effectuer en fonction de son rôle dans l'application.

Pourquoi ?

Le tableau de bord peut contenir des informations sensibles, telles que des données personnelles des utilisateurs ou des données de performance critiques. En fonction de leur rôle (administrateur, apprenants standard, professionnels...), certains utilisateurs peuvent avoir accès à des informations et fonctionnalités spécifiques tandis que d'autres n'ont pas ce privilège. Le RBAC garantit que seuls les utilisateurs autorisés peuvent accéder à ces informations et effectuer des actions sensibles.

Exemple : Un administrateur pourrait avoir accès à la gestion des apprenants et à la configuration de l'application, tandis qu'un apprenant standard n'aurait accès qu'à son propre profil et à ses propres données.



2.2 Limitation des actions utilisateurs

Restreindre les actions possibles en fonction du rôle de l'utilisateur, notamment l'édition, la suppression ou l'accès aux données sensibles.

Si un utilisateur peut effectuer des actions non autorisées, cela peut entraîner des fuites de données ou des erreurs. Limiter les actions garantit que les utilisateurs n'effectuent que des actions correspondant à leur niveau de privilège.

2.2 Limitation des actions utilisateurs

Implémentation d'un système d'audit pour suivre les actions des utilisateurs dans le tableau de bord, y compris les connexions, les modifications de données, et les changements de paramètres de l'application.

Pourquoi : Les journaux d'audit permettent de détecter toute activité suspecte et de suivre les actions des utilisateurs pour des raisons de sécurité et de conformité. Cela est crucial pour identifier toute tentative d'accès non autorisé.

2.2.1 Journalisation des connexions :

Ce qui doit être suivi :

Les connexions des utilisateurs (heure de connexion, adresse IP, type de dispositif, et adresse de connexion).

Cela permet de détecter toute tentative de connexion suspecte, comme un accès depuis une adresse IP inconnue ou un changement d'appareil. Cela peut aussi aider à retracer des actions en cas d'incident.

2.2.2 Utilisation d'outils de gestion des logs (ELK Stack) :

Parmi les outils de gestion des logs, **ELK Stack (Elasticsearch, Logstash, Kibana)** est une solution puissante et open-source adaptée à un tableau de bord. Il permet :

- **Centralisation des logs** : Logstash collecte et transforme les logs provenant de différentes sources.
- **Stockage et recherche rapide** : Elasticsearch indexe et stocke les logs, permettant une recherche efficace.
- **Visualisation et analyse** : Kibana permet de créer des tableaux de bord interactifs pour analyser les logs et détecter les anomalies.
- **Alertes en temps réel** : Possibilité de configurer des alertes pour signaler des événements critiques.

Exemple :

Un utilisateur se connecte au tableau de bord.

Logstash récupère les logs de connexion.

Elasticsearch indexe ces logs pour permettre une recherche rapide.

Kibana affiche un graphique montrant les connexions récentes, avec des filtres pour identifier les connexions suspectes.

3. Catalogue de Formations

Le catalogue de formation est un composant essentiel de la plateforme. Il permet aux utilisateurs d'accéder aux formations proposées et de consulter leur contenu. Pour garantir la sécurité de cette section, nous mettons en place des mesures de protection à différents niveaux :

3.1 Cacher les boutons/actions interdites

Un utilisateur ne doit voir que les actions qui lui sont permises en fonction de son rôle et de son inscription. Cela empêche une mauvaise expérience utilisateur et limite les tentatives d'accès non autorisées.

Rôle / Action	Voir le catalogue	Accéder à une formation	Télécharger les ressources	Modifier une formation
Utilisateur non inscrit	Oui	Non	Non	Non
Apprenant inscrit	Oui	Oui (uniquement les formations inscrites)	Oui	Non
Formateur	Oui	Oui (ses propres formations)	Oui	Oui (ses propres formations)
Administrateur	Oui	Oui	Oui	Oui

- Conditionner l'affichage des boutons en fonction des permissions
- Désactiver les liens interdits (Si une action ne doit pas être possible)

3.2 Gérer les erreurs d'accès

Même si les boutons sont cachés, un utilisateur malintentionné peut essayer d'accéder directement à une formation en entrant l'URL manuellement. Il faut donc gérer les erreurs côté frontend et **backend**.

- Redirection automatique si l'utilisateur n'a pas l'accès, et affichage d'un message d'erreur clair

4. Mesures de sécurité globales

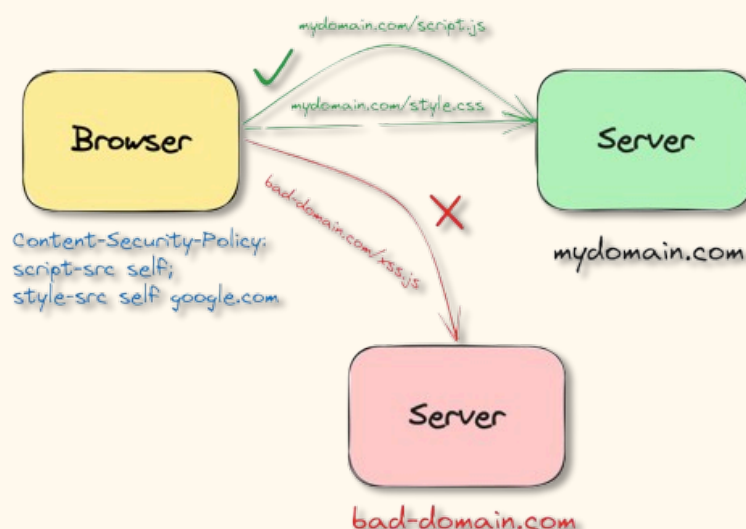
4.1 Protection contre le Cross-Site-Scripting (XSS)

4.1.1 Qu'est-ce que le XSS ?

Le **Cross-Site Scripting (XSS)** est une faille de sécurité qui permet à un attaquant d'injecter du code malveillant dans une page web vue par d'autres utilisateurs. Cela peut entraîner le vol de sessions, la redirection vers des sites malveillants, l'affichage de faux formulaires pour du phishing, ou même l'exécution d'actions non autorisées sur le site.

4.1.2 Mesures de protection contre le XSS :

- **Échapper toutes les entrées utilisateur avant affichage** : Quand une donnée entrée par l'utilisateur doit être affichée sur le site, elle doit être transformée pour empêcher l'exécution de scripts.
- **Appliquer le Content Security Policy (CSP)** : C'est une règle de sécurité définie dans l'en-tête HTTP qui empêche le navigateur d'exécuter du JavaScript injecté.



4.1.3 Valider et filtrer les entrées utilisateur :

Avant de stocker ou afficher une donnée, elle doit être validée pour s'assurer qu'elle respecte les règles attendues, pour empêcher les injections de code malveillant dès l'entrée des données, et réduire la surface d'attaque en ne traitant que des entrées valides.

Exemple : Un champ "Nom" ne doit contenir que des lettres (Regex : `^[a-zA-Z\s]+$`).

4.2 Protection contre le Cross-Site Request Forgery (CSRF)

4.2.1 Qu'est-ce que le CSRF ?

Le CSRF est une attaque où un utilisateur authentifié exécute une action non désirée sur un site légitime à son insu. Un attaquant peut exploiter la session active d'un utilisateur pour effectuer des requêtes malveillantes (ex : modifier un mot de passe, effectuer un virement).

4.2.1 Mesures de Protection :

- **Implémentation des jetons CSRF pour prévenir les attaques**

Les jetons CSRF sont des valeurs uniques générées aléatoirement côté serveur et envoyées au client. Ces valeurs étant uniques pour chaque requête, elles permettent de renforcer la sécurité d'une application en rendant difficile (voire impossible) pour un attaquant de les deviner et donc d'exploiter une vulnérabilité CSRF.

- **Restriction des requêtes CORS**

Autoriser uniquement les **origines de confiance** (`Access-Control-Allow-Origin`), et bloquer les requêtes provenant de **sites tiers non autorisés**.

III. Côté Serveur

IV. API