Security Strategy for pire2pire.com: Data Protection, Compliance, and Secure Access for an Online Learning Platform

by

Martial Floquet

2024

Simplon

INTRODUCTION

Platform Overview and Client Introduction

Pire2Pire.com is an innovative online learning platform designed to meet the needs of both trainers and learners in the e-learning space. Created to be a knowledge-sharing hub, the platform allows trainers to create, publish, and manage online courses accessible to a wide audience of learners. Each trainer can organize courses into modules and lessons, track learners' progress, and interact with them to foster collaborative learning.

Pire2Pire caters to three types of users:

- **Administrators**: They oversee the entire platform, manage users, roles, and ensure the control of the content being offered.
- **Trainers**: They create and publish courses, monitor the learners' progress, and interact with learners to support their learning journey.
- **Learners**: They enroll in courses, access modules, and can engage with trainers to enhance their learning experience.

To ensure a secure, compliant, and reliable service that meets the needs of its users, a comprehensive security strategy is required. This strategy must address critical aspects of the platform, such as personal data protection, service availability, and defense against cyber threats.

INTRODUCTION

Table des matières

INTRODUCTION

INTRODUCTION

INTRODUCTION GENERALE A LA SECURITE

Dans un contexte où la sécurité des systèmes d'information est devenue une préoccupation centrale pour toutes les organisations, la mise en place d'une plateforme de formation en ligne telle que pire2pire.com doit s'accompagner de mesures de protection robustes. La montée en puissance des cybermenaces et l'importance croissante des réglementations telles que le RGPD (Règlement Général sur la Protection des Données) imposent la mise en œuvre de stratégies de sécurité rigoureuses et adaptées aux besoins spécifiques d'une plateforme en ligne. La sécurité ne doit pas être considérée comme un ajout optionnel, mais comme un pilier fondamental du développement et de l'exploitation de la plateforme.

L'objectif de cette section est de présenter les concepts clés de sécurité à intégrer dans la conception et l'exploitation de pire2pire.com, en tenant compte des utilisateurs (administrateurs, formateurs, apprenants) et des spécificités techniques de la plateforme. Il s'agit de définir des principes globaux et d'expliquer comment les mettre en œuvre afin de protéger les données personnelles, d'assurer la disponibilité du service, de contrôler les accès et d'anticiper les menaces courantes.

1.1 Introduction au RGPD

Le RGPD impose des obligations aux entreprises et organisations qui collectent ou traitent des données personnelles. Dans le contexte de la plateforme pire2pire.com, cela signifie que toutes les informations concernant les formateurs, apprenants et administrateurs doivent être traitées de manière sécurisée et conforme à ce règlement. L’objectif est de protéger les droits des utilisateurs en matière de confidentialité et de garantir que leurs données ne sont pas utilisées de manière abusive.

1.2 Collecte et traitement des données personnelles

La plateforme pire2pire.com devra collecter des informations personnelles telles que les noms, prénoms, adresses e-mail et éventuellement des informations de paiement. Le RGPD impose de s'assurer que :

- Les utilisateurs donnent leur consentement explicite pour la collecte de leurs données.
- Seules les données nécessaires sont collectées (principe de minimisation).
- Les utilisateurs ont le droit d’accéder à leurs données, de les rectifier, de les supprimer ou de limiter leur traitement.

1.3 Droits des utilisateurs

Le RGPD permet aux utilisateurs de :

- **Accéder à leurs données** : Les utilisateurs peuvent demander à consulter les données que la plateforme détient à leur sujet.
- **Rectifier leurs données** : Si des informations sont incorrectes ou incomplètes, les utilisateurs peuvent les modifier.
- **Supprimer leurs données** (droit à l'oubli) : Les utilisateurs peuvent demander la suppression de leurs données personnelles si elles ne sont plus nécessaires.
- **Limiter ou restreindre leur traitement** : Dans certains cas, les utilisateurs peuvent restreindre l’utilisation de leurs données.

2.1 Les attaques courantes

La plateforme pire2pire.com, comme toutes les applications web, est exposée à divers types de menaces qui peuvent compromettre la sécurité des données et des utilisateurs. Comprendre ces menaces est crucial pour pouvoir mettre en place des mesures de protection adaptées. Voici les principales attaques auxquelles la plateforme pourrait être confrontée.

- **XSS (Cross-Site Scripting)** : Injection de scripts malveillants dans les pages web, pouvant voler des informations comme les cookies ou détourner des sessions utilisateur.
- **CSRF (Cross-Site Request Forgery)** : Forcer un utilisateur authentifié à exécuter des actions non souhaitées, comme modifier ou supprimer des données.
- **SSRF (Server-Side Request Forgery)** : Manipulation du serveur pour qu'il envoie des requêtes à des ressources internes, permettant de contourner les contrôles d'accès.
- **SQLi (Injection SQL)** : Insertion de commandes SQL malveillantes dans les requêtes de l'application, permettant l'accès ou la modification non autorisée des bases de données.
- **LFI/RFI (Inclusion de fichiers locaux/distants)** : Inclusion de fichiers non sécurisés dans l'application, ce qui peut mener à l'exécution de code malveillant.
- **XXE (XML External Entity)** : Exploitation des entités externes dans des fichiers XML, permettant d'accéder à des fichiers sensibles sur le serveur.

2.2 Concepts de sécurité clés

Réduction de la surface d'attaque : Limiter le nombre de points d'entrée vulnérables en réduisant les fonctionnalités et accès exposés.

Défense en profondeur : Ajouter plusieurs couches de protection pour rendre plus difficile une compromission complète du système.

Politique de moindres privilèges et RBAC : Accorder à chaque utilisateur ou système uniquement les permissions nécessaires via un contrôle d'accès basé sur les rôles (RBAC), réduisant ainsi les risques en cas de compromission.

Sécurisation de la base de données

3.1. Introduction à la gestion sécurisée des bases de données

Pire2pire.com utilise une base de données (BDD) contenant des informations sensibles d'utilisateur, telles que des mots de passe, des adresses e-mail, et d'autres données personnelles. Cette BDD est cruciale pour le fonctionnement de la plateforme, mais elle doit être protégée contre toute violation potentielle de la sécurité afin de garantir la confidentialité des utilisateurs.

Les menaces principales concernant la BDD sont :

- **L'Injection SQL (SQLi)** : Les attaquants peuvent tirer parti de vulnérabilités dans la gestion des bases de données pour accéder, voler, modifier ou effacer des données confidentielles.
- **Le CSRF** : Cette attaque force l'utilisateur final à exécuter des actions non désirées sur une application web où il est authentifié permettant alors de transmettre, modifier, supprimer, ou ajouter des données.
- **Le XSS** : Ce type d'attaque permet à un attaquant d'injecter du code malveillant (généralement du JavaScript) dans une page web consultée par un autre utilisateur. Même si l'attaque passe par le front de l'application, la base de données peut être affectée indirectement, car l'attaquant peut utiliser le XSS pour voler des informations sensibles (comme des identifiants de session) ou manipuler des requêtes envoyées à la base de données, entraînant des modifications ou des fuites de données.

3.2 Hachage et salage

Afin de prévenir toute falsification d'identité, nous appliquons une politique rigoureuse en matière de mots de passe, exigeant des critères de complexité, une longueur minimale et une vérification stricte des conditions. Pour assurer leur stockage sécurisé, nous recourons à un hachage SHA256 combiné à un salage.

Le mot de passe doit être haché (sécurisation des données et respect du RGPD). Pour prévenir l'utilisation de **rainbow tables**, qui permettent de retrouver un mot de passe à partir de son hash via des correspondances pré-calculées, nous ajoutons également un sel unique au hash. Cela rend chaque hash unique, même pour deux mots de passe identiques. Dans ce cas précis, nous utiliserons l'algorithme Bcrypt, qui inclut nativement un mécanisme de salage et est conçu pour résister aux attaques par force brute.

Cela est particulièrement utile dans le contexte d'une attaque par injection **SQL (SQLi)**, notamment si l'attaquant cherche à obtenir des données sensibles telles que des mots de passe. Le hachage, associé à l'utilisation d'un salt, constitue alors une mesure de sécurité essentielle.

3.3 Fonction de dérivation de mots de passe memory-hard et coffre-fort

L’utilisation d’une fonction de dérivation de mots de passe memory hard peut être intéressante pour les comptes Administrateurs.

Cela ralentit les tentatives d'attaques par force brute ou par **rainbow tables** en exigeant une quantité importante de ressources pour chaque tentative de calcul du hash.

Nous pourrions mettre le facteur de coût(saltRounds) à 12.

L'utilisation d'un **coffre-fort externe** est une excellente pratique de sécurité pour garantir la protection des secrets et des informations sensibles. Dans le cas de pire2pire.com, cela permettrait de mieux sécuriser les **mots de passe.**

3.4 Journalisation au niveau de la BDD

L’activité de journalisation est un moyen de détection des incidents de sécurité, elle peut-être appliqué autant au front qu’au back, dans le cas de la BDD elle nous permet de suivre en temps réel qui (quels utilisateurs ou services) accède aux données sensibles, quand et comment. Cela peut inclure les tentatives d'accès, les modifications de données (insertions, mises à jour, suppressions), ainsi que les requêtes exécutées.

Cela nous permettrait de révoquer immédiatement les mots de passe en cas de compromission suspectée ou avérée.

Cette mesure garantit que si une compromission est détectée, les mots de passe ne peuvent plus être utilisés pour des attaques ultérieures, y compris les **CSRF** ou les sessions volées via **XSS**. Cela ne concerne pas directement **SQLi**, mais elle aide à limiter les dégâts en cas de fuite via **XSS** ou d'autres attaques de vol d'identifiants.

3.5 Utilisation de UUID

Les UUID sont utilisés pour générer des identifiants uniques qui ne se répètent pas, même dans des systèmes distribués ou à grande échelle.

Utiliser des **UUIDs** offre une meilleure sécurité que des IDs auto-incrémentés.

L'UUID est difficilement devinable car il est généré aléatoirement et comporte 128 bits (ce qui donne environ 3,4 x 10^38 combinaisons possibles), rendant les tentatives de deviner d'autres identifiants presque impossibles.

L'utilisation de UUID dans la BDD de Pire2pire est donc indispensable.

Sécurisation de la base de données

3.6 Politique du moindre privilège (BDD)

La base de données contient toutes les informations sensibles des utilisateurs (données personnelles, progression des cours, etc.), et il est essentiel de la protéger contre tout accès non autorisé ou fuite de données.

Comptes d'accès restreints : Chaque service ou application (par exemple, l'API) aura des **comptes distincts** pour accéder à la base de données, avec des droits strictement définis. Ces comptes n'auront que les permissions nécessaires pour effectuer leurs tâches respectives (lecture seule, modification limitée, etc.).

- **Admin** : Peut avoir des droits d’accès plus larges dans certaines tables (gestion des utilisateurs, des cours, etc.), mais n’aura pas d'accès direct aux données de paiement ou aux informations hautement sensibles, sauf là où cela est strictement nécessaire.
- **Formateurs** : Peuvent accéder uniquement aux données pertinentes à leurs propres cours (modules, progression des apprenants) mais ne peuvent pas accéder à d'autres tables ou à des informations administratives.
- **Apprenants** : Ont un accès très limité à leurs propres données (progression des cours) et aucune permission sur d'autres tables ou informations sensibles.

3.7 Requêtes préparées

L'injection **SQL (SQLi)** est l'une des attaques les plus courantes contre les bases de données. Elle se produit lorsque des données d'entrée sont utilisées directement dans une requête SQL sans validation, permettant à un attaquant de manipuler la requête. Les **requêtes préparées** résolvent ce problème en séparant les **données des commandes SQL**. Les données utilisateur sont toujours traitées comme des valeurs et **jamais** comme une partie de la commande SQL, ce qui empêche un attaquant d'injecter du code malveillant.

Aspect	Requête non préparée (non sécurisée)	Requête préparée (sécurisée)
Construction de la requête	Les variables sont directement insérées dans la chaîne SQL.	Les variables sont passées séparément via des placeholders (`?`), et la base de données les traite comme des valeurs.
Risques d'injection SQL	Très vulnérable à l'injection SQL.	Protégée contre l'injection SQL.
Échappement des caractères spéciaux	Aucune protection automatique ; il faut le faire manuellement.	Géré automatiquement par le moteur de base de données.

3.8 Défense en profondeur et politique de rétention (BDD)

Toutes ces diffèrentes sécurités et préventions participe à une défense en profondeur de la BDD de l’application Pire2pire.com :

- Hachage et salage + fonction de dérivation de mots de passe memory-hard
- Coffre fort
- UUID
- Journalisation
- Requêtes préparées

Appliquer la politique du moindre privilège permet également de réduire la surface d’attaque, nous pouvons aussi désactiver les fonctionnalités non utilisés et limiter les accès au réseau.

Politique de rétention

Pour assurer une protection optimale des données de la plateforme Pire2Pire.com, nous mettrons en place une stratégie de sauvegarde avec **20 sauvegardes par jour**. Cette stratégie comprendra une **sauvegarde complète** effectuée chaque nuit, idéalement entre **1h et 4h du matin**, période de faible activité, afin de limiter l'impact sur les performances du système.

Durant la journée, nous effectuerons des **sauvegardes partielles** à des intervalles réguliers, environ toutes les 1 à 2 heures, pour capturer uniquement les données modifiées. Cela garantit une récupération rapide et efficace des données en cas de panne ou d'incident.

En complément, une **politique de rétention** sera mise en place pour gérer l'espace de stockage tout en garantissant la disponibilité des données. Les **sauvegardes complètes** seront conservées pendant **30 jours** et les **sauvegardes partielles** pendant **7 jours**. Cette politique permet de lutter contre les risques liés aux **erreurs humaines**, aux **cyberattaques (ransomware)** ou à une **corruption des données**, en offrant la possibilité de revenir à une version antérieure des données avant l'incident. Après ces périodes, les sauvegardes seront automatiquement supprimées ou archivées si nécessaire.

4.1 Introduction à la gestion sécurisée de l’API

Pire2pire.com repose également sur une API qui permet aux différentes parties de la plateforme (utilisateurs, cours, paiements) de communiquer entre elles de manière fluide. Cette API joue un rôle fondamental dans le fonctionnement de la plateforme, en assurant l'échange d'informations entre les utilisateurs (admins, formateurs, apprenants) et les services de la plateforme. Cependant, cette API doit être sécurisée afin de prévenir toute attaque qui pourrait compromettre l'intégrité, la disponibilité et la confidentialité des données.

Les menaces principales concernant l’API sont :

- **CSRF (Cross-Site Request Forgery)** : Cette attaque vise à forcer un utilisateur authentifié à exécuter une action non souhaitée via l'API, comme envoyer une requête pour modifier son profil ou supprimer des données, sans qu'il en soit conscient.
- **SSRF (Server-Side Request Forgery)** : L'attaquant utilise l'API pour forcer le serveur à envoyer des requêtes vers d'autres ressources internes (services internes ou adresses IP privées). Cela peut permettre à l'attaquant de contourner les restrictions d'accès.

- **LFI/RFI (Inclusion de fichiers locaux/distants)** : Si l'API permet d'inclure des fichiers depuis une URL ou un chemin local, un attaquant pourrait abuser de cette fonctionnalité pour inclure des fichiers non autorisés, conduisant à l'exécution de code malveillant.
- **XXE (XML External Entity)** : Si l'API utilise des fichiers XML, un attaquant peut injecter des entités XML externes dans la requête pour lire des fichiers sensibles ou exécuter des requêtes SSRF.

4.2 Sécurisation de l’API via TLS et TCP

TLS (Transport Layer Security) garantit la confidentialité et l'intégrité des données échangées entre le client (navigateur ou application mobile) et l'API. Cela empêche les attaques de type **Man-in-the-Middle** (MiTM), où un attaquant intercepte ou altère les communications entre le client et le serveur.

Nous pouvons forcer l’utilisation de HTTPS pour s’assurer que toutes les communications avec l’API passent par HTTPS.

Le serveur API doit avoir un certificat SSL/TLS valide pour chiffrer les connexions.

Le **TCP (Transmission Control Protocol)** est un protocole de transport qui fonctionne à un niveau inférieur par rapport à TLS dans la pile réseau.

Il est responsable de la fiabilité de la transmission.

TCP, à lui seul, ne protège pas contre les attaques . Par exemple, lors d'une connexion simple via HTTP (sur TCP), les informations comme les mots de passe, des données personnelles ou des messages sont en clair.

Avec **TLS**, ces mêmes informations sont chiffrées avant d’être envoyées via TCP, ce qui les rend **inaccessibles** ou **illisibles** pour toute personne qui tenterait de les intercepter.

4.3 Restriction du nombre de requêtes vers l'API

Pour prévenir les attaques par **déni de service (DoS ou DDoS)**, il est crucial de mettre en place des mécanismes permettant de limiter le nombre de requêtes qu'un utilisateur peut effectuer dans un laps de temps donné. Une approche efficace consiste à restreindre le nombre d'appels à l'API à 30 requêtes par minute par utilisateur.

Cette limite permet de contrôler le volume de trafic généré par chaque utilisateur et empêche un acteur malveillant de saturer les ressources du serveur en envoyant un grand nombre de requêtes en peu de temps. Si un utilisateur dépasse ce seuil, l'application peut automatiquement bloquer temporairement les requêtes supplémentaires ou renvoyer une réponse indiquant un "trop grand nombre de requêtes" (HTTP 429).

4.4 Jetons CSRF

La sécurisation d'une API avec des jetons CSRF (Cross-Site Request Forgery) est une méthode efficace pour protéger contre les attaques où un attaquant pourrait forcer un utilisateur authentifié à exécuter des actions non souhaitées. Bien que les attaques **CSRF** soient plus courantes dans les applications web avec une interface utilisateur, il est également possible d'utiliser des jetons CSRF pour sécuriser certaines API, notamment celles qui manipulent des données sensibles ou des actions critiques.

Le jeton doit contenir une valeur aléatoire générée à l’aide d’une fonction utilisant un générateur d’aléa cryptographique et ayant une entropie minimale de 128 bits. Cette taille peut être atteinte en générant aléatoirement une chaîne de 22 caractères ASCII imprimables (A à Z, a à z et 0 à 9).

En outre, pour renforcer la sécurité, chaque jeton doit avoir un **délai d’expiration** variable selon le type d'utilisateur, obligeant ainsi une nouvelle authentification après une période d'inactivité spécifique :

- **Pour un compte apprenant**, le jeton expirera au bout de **5 jours** sans connexion. Cela permet aux apprenants de rester connectés pendant une durée prolongée sans être perturbés, tout en garantissant la sécurité après une période prolongée d'inactivité.

- **Pour un compte formateur**, le jeton expirera après **24 heures** sans connexion. Cette durée est plus courte pour les formateurs afin de mieux protéger les accès aux fonctionnalités plus critiques de gestion de contenus pédagogiques et des interactions avec les apprenants.
- **Pour un compte administrateur**, le jeton aura une durée d'expiration stricte de **10 minutes** sans activité. Compte tenu de la sensibilité des actions administratives, cette expiration rapide garantit un haut niveau de sécurité en limitant la durée d'une session active.

Ainsi, ces politiques d'expiration des jetons garantissent que même si un jeton est compromis ou volé, l'accès non autorisé sera limité dans le temps, et les utilisateurs devront se reconnecter périodiquement selon leur rôle.

4.5 Limiter le nombre de tentatives d’authentification

Pour protéger l'API contre les attaques par force brute et d'autres tentatives non autorisées d'accès, il est essentiel de mettre en place des mécanismes limitant le nombre de tentatives d'authentification.

Chaque tentative d'authentification échouée doit être suivie par le serveur. Pour cela le serveur doit stocker le nombre de tentatives de connexion échouées pour chaque utilisateur ou chaque adresse IP.

Par exemple, limiter à 3 **tentatives d'authentification toutes les 1 minute** pour chaque compte apprenant et formateur.

Cela permet de ralentir une attaque par force brute sans pour autant pénaliser l’utilisateur moyen qui se serait simplement trompé de mot de passe, en lui évitant une attente prolongée.

Pour les comptes administrateurs, une fois cette limite atteinte de 3 tentatives par minute, l'utilisateur ou l'adresse IP tentant de se connecter peut être temporairement bloqué ou ralenti pendant une période définie (par exemple, 5 minutes avant une nouvelle tentative possible).

4.6 Utilisation de l’API fetch

L'**XHR (XMLHttpRequest)** est une méthode utilisée dans les navigateurs pour permettre à une page web de communiquer avec un serveur sans avoir besoin de la recharger entièrement. Cela permet à une page de récupérer ou d'envoyer des données (comme des fichiers ou des messages) au serveur en arrière-plan, sans que l'utilisateur ne s'en aperçoive.

Cependant, nous allons privilégier l'utilisation de l'**API Fetch**, car elle est plus moderne, propose une syntaxe plus simple, et offre une gestion des requêtes et des réponses plus fluide et facile à comprendre. En plus de cela, **Fetch** apporte aussi des avantages en matière de sécurité, en prenant en charge des fonctionnalités comme **CORS** (Cross-Origin Resource Sharing) de manière native, ce qui renforce la sécurité des échanges entre domaines. Cela permet de mieux contrôler les ressources que les navigateurs peuvent charger et rend l'application plus robuste contre les attaques potentielles liées aux requêtes web non sécurisées.

4.7 Sécurisation des échanges CORS et gestion des ressources externes

Pour assurer la sécurité de la plateforme Pire2Pire.com, il est important de bien gérer les **appels CORS (Cross-Origin Resource Sharing)** et les ressources externes. Quand des données sensibles sont échangées, il faut mettre en place un **preflight**. Cela permet au serveur de vérifier si la requête est autorisée avant de l’exécuter. Cette étape est cruciale pour éviter les fuites d’informations lors d’appels POST, PUT, ou ceux qui demandent une authentification.

Il est aussi essentiel de vérifier l’en-tête **Origin** à chaque requête pour s’assurer que seules les origines de confiance peuvent accéder à l’API. Utiliser `"*"` comme valeur pour l’en-tête `Access-Control-Allow-Origin` est risqué, surtout pour des services internes, car cela permet à n’importe quel site de faire des requêtes. Pour les requêtes qui utilisent des identifiants, il est indispensable de bien contrôler les origines pour éviter tout accès non autorisé.

Ensuite, nous pourrions séparer les services web sur différents noms de domaine (par exemple, un domaine pour l'authentification, un autre pour les paiements). Cela permet d'éviter qu'une attaque sur un service n'affecte les autres.

Sécurisation de l’API

L’utilisation de **bibliothèques publiques** qui font des appels CORS doit être évitée, surtout si leur code n'est pas clair, car elles peuvent contenir des failles de sécurité. S’il est réellement nécessaire de les utiliser, alors il est préférable de les isoler dans un **Web Worker** ou une **iframe**. Enfin, pour les ressources qui ne demandent pas d’authentification, l’attribut `crossorigin` doit être mis à `anonymous` pour limiter l'exposition des données sensibles.

iframe : Une balise HTML qui permet d'afficher une autre page web à l'intérieur d'une page. Cela permet d'isoler du contenu ou des scripts dans un cadre séparé.

Web Worker : Un script JavaScript qui s’exécute en arrière-plan, séparément de l'interface utilisateur, pour effectuer des tâches sans bloquer le reste de la page.

4.8 Journalisation au niveau de l’API

La journalisation au niveau de l’API est un outil essentiel de détection des incidents de sécurité. Elle peut être appliquée tant aux actions des utilisateurs qu'aux services consommant l'API. Cette journalisation permet de suivre en temps réel qui accède à l’API, quels endpoints sont utilisés, quand et comment. Les informations enregistrées peuvent inclure les tentatives d’accès, les modifications de données (requêtes POST, PUT, DELETE), ainsi que les erreurs générées.

Cette mesure permet de suivre l'activité des utilisateurs et des services dans le cadre de la sécurisation des accès, notamment aux données sensibles. En cas d’anomalie ou de comportement suspect détecté dans les journaux (par exemple, une requête venant d’une IP inhabituelle ou des tentatives de connexion échouées répétées), il devient possible de révoquer immédiatement les tokens d’accès ou de forcer la réinitialisation des mots de passe des utilisateurs concernés.

La journalisation aide à prévenir et limiter les dégâts causés par des attaques telles que l’usurpation d’identité (via CSRF, ou vol de session). Bien qu’elle ne traite pas directement des injections SQL (SQLi), elle est cruciale pour détecter les tentatives d’abus ou d'accès non autorisés aux API. De plus, elle permet une réponse proactive en cas de compromission, en offrant des informations pour une analyse post-incident détaillée et une correction rapide.

4.9 Politique du moindre privilège (API)

L'API gère les interactions des utilisateurs (admin, formateurs, apprenants) avec la plateforme. Il est crucial de sécuriser cet accès en limitant les actions disponibles en fonction des rôles.

Politique du moindre privilège :

- **Admin** : A un accès complet aux fonctions de gestion via l’API (utilisateurs, contenu), mais pas aux données de paiement, sauf pour la gestion administrative.
- **Formateurs** : Accèdent uniquement aux fonctions liées à leurs formations et à leurs apprenants, sans accès aux fonctions administratives ou aux données des autres formateurs.
- **Apprenants** : Accès limité aux routes concernant leurs cours et progression, sans accès aux données sensibles ou à celles d’autres utilisateurs.

Justification : Limiter les permissions API par rôle réduit les risques en cas de compromission et protège les fonctions critiques de la plateforme.

Contrôles d’accès basés sur les rôles (RBAC) :

- **RBAC** vérifie que chaque requête API correspond aux permissions de l’utilisateur, garantissant que seuls les rôles autorisés accèdent aux fonctionnalités nécessaires.

4.10 Défense en profondeur et réduction de la surface d’attaque (API)

Toutes ces diffèrentes sécurités et préventions participe à une défense en profondeur de l’API de l’application Pire2pire.com :

- Sécurisation de l’API via TLS et TCP
- Restriction du nombre de requêtes vers l'API
- Jetons CSRF
- Limiter le nombre de tentatives d’authentification
- Utilisation de l’API fetch
- Sécurisation des échanges CORS et gestion des ressources externes
- Journalisation au niveau de l’API

Appliquer la politique du moindre privilège permet également de réduire la surface d’attaque.

Sécurisation du DOM

5.1 Introduction à la gestion sécurisée du DOM

Le **DOM** (Document Object Model) représente la structure du contenu d'une page web et permet l'interaction entre le front-end de l'application **pire2pire.com** et les utilisateurs (administrateurs, formateurs, apprenants). Il est essentiel de sécuriser cette partie, car des attaques ciblant le DOM peuvent compromettre la sécurité et l'intégrité des données utilisateur.

Les principales menaces concernant le DOM sont :

- **Cross-Site Scripting (XSS)** : Un attaquant injecte du code malveillant, généralement du JavaScript, dans la page web. Cela permet de manipuler le DOM et d'accéder à des informations sensibles, voire de prendre le contrôle des sessions des utilisateurs.
- **Injection SQL (SQLi)** à travers des formulaires : Un attaquant peut injecter des requêtes SQL malveillantes dans des champs de saisie, tentant ainsi d'altérer, voler ou supprimer des données dans la base de données si ces entrées ne sont pas correctement filtrées et validées.

Une gestion sécurisée du DOM est donc cruciale pour éviter que ces attaques n’exploitent des vulnérabilités du front-end et n’affectent indirectement la sécurité globale de la plateforme **pire2pire.com**, y compris la protection des informations stockées dans la base de données.

5.2 Politique des mots de passes et regex

Nous suivrons les recommandations de l'ANSSI en matière de mots de passe. Ainsi, la longueur minimale sera de 12 caractères pour un compte apprenant, 16 pour un formateur, et 24 pour un administrateur (les passphrases étant recommandées).

Bien que l'ANSSI ne spécifie pas directement de longueur maximale dans ses recommandations officielles, elle souligne qu'il ne faut pas imposer une limite trop basse, afin de ne pas restreindre l'utilisation de mots de passe complexes ou de phrases de passe. Il est donc courant de recommander un maximum de 64 à 128 caractères pour permettre l'utilisation de passphrases longues.

De plus, nous appliquerons un motif de mot de passe (regex) exigeant au minimum une majuscule, un caractère spécial et un chiffre.

Le regex est quant à lui une mesure de sécurité visant principalement à contraindre l’utilisateur à complexifier son mot de passe, réduisant ainsi le risque de succès d’une attaque par force brute.

5.3 Sanitisation

Nous mettrons en place une sanitisation stricte des données dans le DOM pour protéger pire2pire.com contre les attaques. Toutes les entrées utilisateur seront nettoyées et échappées pour éliminer les caractères dangereux comme `<`, `>`, ou `'`, empêchant ainsi les attaques de type **Cross-Site Scripting (XSS)**.

Chaque donnée sera validée côté client et côté serveur, garantissant que seules des informations conformes sont traitées. Des bibliothèques comme DOMPurify seront utilisées pour nettoyer automatiquement les données avant leur insertion dans le DOM.

Enfin, pour toute interaction avec la base de données, comme vu précédemment des requêtes préparées seront utilisées afin de prévenir les injections SQL.

Ainsi, la sécurisation des données dans le DOM sera assurée de bout en bout, limitant les risques d’attaques.

5.4 Prévention de l'exécution de code arbitraire en JavaScript

Pour sécuriser l'application pire2pire.com, il est essentiel de limiter l'exécution de code dynamique et de prévenir les vulnérabilités liées à l'évaluation de chaînes de caractères en JavaScript. Voici les points clés de notre stratégie :

1. Proscrire l'usage de `eval()` : La fonction `eval()` ne sera pas utilisée car elle permet d'exécuter du code arbitraire à partir de chaînes de caractères, ouvrant la voie à des attaques XSS et d'autres vulnérabilités. À la place, des alternatives sécurisées comme `JSON.parse()` seront utilisées pour traiter des données.

2. Interdire l'évaluation de code dans `setTimeout()` et `setInterval()` : Ces fonctions, si elles reçoivent des chaînes de caractères comme paramètres, peuvent exécuter du code non sécurisé. Nous utiliserons plutôt des références de fonctions .

5.5 Utilisation de template

Dans notre approche de sécurisation pour pire2pire.com, on va encadrer l’utilisation des templates HTML pour garantir que le contenu dynamique est manipulé de façon sûre.

Sécurisation des templates

- Isolation des templates : Les balises `<template>` permettent de stocker du contenu HTML invisible, qui ne sera inséré dans le DOM que lorsque c'est nécessaire. Cela évite que des éléments non sécurisés soient exécutés ou visibles avant d'être validés.
- Validation des données : Avant de rendre du contenu via un template, on validera et nettoiera soigneusement les données pour éviter tout risque d’injection de code malveillant. On garde le contrôle sur ce qui est affiché.
- Séparation du code et du contenu : Le contenu HTML géré par les templates restera distinct du code JavaScript pour réduire les risques d’exploitation de failles ou de manipulations dangereuses.

En suivant cette approche, on assure que l’affichage dynamique sur la plateforme reste sécurisé, tout en conservant la flexibilité nécessaire pour les interactions utilisateur.

5.6 Défense en profondeur (DOM)

Toutes ces diffèrentes sécurités et préventions participe à une défense en profondeur du DOM de l’application Pire2pire.com :

- Politique des mots de passes et regex
- Sanitisation
- Prévention de l’éxecution de code arbitraire en javascript
- Utilisation de template

Sécurisation du navigateur

6.1. Introduction à la gestion sécurisée du navigateur

Le navigateur est l'interface principale par laquelle les utilisateurs de **pire2pire.com** interagissent avec la plateforme. C'est un composant critique pour la sécurité, car il traite les interactions directes avec les utilisateurs, récupère et affiche les données, et exécute du code JavaScript. Si le navigateur n'est pas correctement sécurisé, il peut devenir une cible pour diverses attaques, compromettant ainsi la confidentialité et la sécurité des données utilisateurs.

Les menaces principales concernant le navigateur sont :

- **Clickjacking** : Cette attaque consiste à piéger un utilisateur pour qu'il clique sur un élément invisible ou caché dans le navigateur, souvent à travers un iframe. Cela peut conduire à des actions non souhaitées, telles que la soumission d'informations ou l'exécution de commandes à l'insu de l'utilisateur.

- **Exécution de code malveillant** : Le navigateur est souvent utilisé pour exécuter du code JavaScript, et si ce code est malveillant, il peut manipuler le comportement de l'application, voler des informations sensibles ou rediriger l'utilisateur vers des sites malveillants.

- **Vol de cookies de session** : Les cookies stockés dans le navigateur, notamment ceux liés aux sessions de connexion, peuvent être volés par des attaquants si des mesures comme **HTTPOnly** ou **Secure** ne sont pas appliquées, permettant ainsi à un attaquant de prendre le contrôle des sessions utilisateur.

6.2 Sécurité contre le ClickJacking

Le clickjacking est une attaque où un utilisateur est trompé pour interagir avec des éléments cachés, souvent via un iframe malveillant.

Pour protéger pire2pire.com de cette menace, il est crucial d'implémenter les en-têtes HTTP appropriés :

X-Frame-Options : Cette en-tête bloque directement l’intégration de la page dans un iframe, selon les règles définies (`DENY`, `SAMEORIGIN`, etc.). Elle est simple à mettre en place et agit comme une première ligne de défense.

Content-Security-Policy (CSP) avec frame-ancestors : Cette méthode est plus flexible que X-Frame-Options et permet de spécifier avec précision quels domaines peuvent intégrer la page dans un iframe. Même si les navigateurs respectent de moins en moins X-Frame-Options, CSP prend le relais avec un contrôle plus fin, notamment pour les pages modernes.

Attribut sandbox sur les iframes : Si un iframe est nécessaire pour la plateforme (par exemple, pour charger du contenu externe), l’attribut sandbox limite ce que cet iframe peut faire, en désactivant certaines fonctionnalités comme l’exécution de scripts non autorisés en dehors de la page principale.

6.3 Subresource integrity

Pour garantir l’intégrité des fichiers externes, tels que les scripts JavaScript ou les feuilles de style CSS, chargés depuis des sources tierces. L’utilisation de SRI (subresource integrity) permet de s'assurer que les ressources externes n'ont pas été modifiées ou compromises avant d'être exécutées dans le navigateur de l'utilisateur.

Subresource Integrity (SRI) fonctionne en ajoutant un hash cryptographique à l'attribut `integrity` des balises `<script>` ou `<link>` dans le code HTML. Le navigateur télécharge le fichier externe, calcule son propre hash, et le compare à celui fourni. Si les deux ne correspondent pas, le fichier est bloqué, garantissant ainsi qu’un fichier altéré ou malveillant ne puisse pas être exécuté.

6.4 Strict mode

Le Strict Mode en JavaScript est une fonctionnalité qui concerne principalement le navigateur, car il s’agit d’une manière plus rigide d’exécuter du code JavaScript. Il n'est pas directement lié au DOM, même s'il peut avoir un impact sur la façon dont certaines interactions JavaScript avec le DOM sont gérées.

Cette fonctionnalité impose des règles plus strictes d’exécution du code, ce qui aide à prévenir des erreurs courantes et à limiter les comportements imprévisibles qui pourraient compromettre la sécurité.

Pourquoi utiliser le Strict Mode ?

1. Prévention d'erreurs : Le Strict Mode empêche des pratiques dangereuses comme l'utilisation de variables non déclarées, réduisant ainsi les bugs potentiels.
2. Amélioration de la sécurité : Il interdit certaines actions risquées, comme la modification d’objets globaux, limitant les vecteurs potentiels d’attaques.
3. Contrôle rigoureux de `this` : Le Strict Mode évite l’utilisation incorrecte de l’objet global, une source commune de failles dans les applications JavaScript.

En l'intégrant à pire2pire.com, nous assurons une exécution du code plus contrôlée, réduisant les risques d'erreurs et améliorant la stabilité générale du site.

6.5 Sécurisation des échanges inter-origines avec SOP et CORS

La Same-Origin Policy (SOP) est une règle de sécurité par défaut dans tous les navigateurs modernes. Elle empêche une page web de faire des requêtes vers un autre domaine sans autorisation, protégeant ainsi les données sensibles comme les cookies ou les informations de session. Cela signifie, par exemple, qu'une page chargée depuis pire2pire.com ne peut pas accéder aux ressources d’un autre site sans autorisation spécifique.

Sécurisation supplémentaire avec CORS

Dans certains cas, pire2pire.com pourrait avoir besoin de partager des données avec des domaines externes, comme une API utilisée par les formateurs. Pour cela, nous utiliserons CORS (Cross-Origin Resource Sharing) qui permet de définir des règles de sécurité contrôlant ces échanges entre domaines.

Mesures à mettre en place :

- Limiter les domaines autorisés : Nous n'autoriserons que des domaines de confiance à interagir avec pire2pire.com, garantissant que seules des requêtes sûres peuvent accéder aux ressources.

- Restreindre les méthodes d'accès : Seules les méthodes nécessaires (comme la consultation ou la soumission de données) seront autorisées, empêchant toute modification ou suppression non sécurisée de données.

Grâce à SOP et à une configuration stricte de CORS, nous assurons que les échanges entre pire2pire.com et d'autres sites restent protégés et limités aux interactions nécessaires et sécurisées.

6.6 Referrer policy

Pour pire2pire.com, nous utilisons la politique **strict-origin-when-cross-origin** qui offre un bon compromis entre **s**écurité et souplesse. Cette politique transmet l'URL complète uniquement entre pages du même domaine, et ne transmet que l’origine (domaine) lors des requêtes vers des sites externes. Si une page HTTPS interagit avec une page HTTP, aucune information n’est envoyée, renforçant ainsi la sécurité.

Avantages :

- **Protection des données sensibles** : Limite les informations transmises aux domaines externes, tout en maintenant la sécurité pour les pages sensibles.
- **Flexibilité avec les services tiers** : Contrairement à des politiques plus restrictives comme `no-referrer`, **strict-origin-when-cross-origin** permet des intégrations fluides avec des outils externes (paiements, analytics).

Ce choix n’est pas le plus strict, mais il permet une souplesse nécessaire pour garantir une bonne interaction avec des partenaires, tout en protégeant les utilisateurs de pire2pire.com.

6.7 HTTP Strict Transport Security

Pour garantir que toutes les communications sur pire2pire.com soient toujours sécurisées, nous mettons en place le mécanisme **HSTS (HTTP Strict Transport Security)**. HSTS force le navigateur à établir des connexions uniquement en HTTPS, empêchant ainsi toute tentative d'utiliser une connexion non sécurisée.

Pourquoi HSTS est crucial pour pire2pire.com ?

- Prévention des attaques man-in-the-middle : En forçant l'utilisation du protocole sécurisé HTTPS, HSTS protège les utilisateurs contre les tentatives d'interception ou de modification des données échangées entre le navigateur et le serveur.
- Redirections toujours sécurisées : Sans HSTS, un utilisateur pourrait accéder à pire2pire.com via une version HTTP non sécurisée. Avec HSTS, même si l'utilisateur tente d'accéder au site via HTTP, le navigateur le redirigera automatiquement en HTTPS, garantissant une connexion sécurisée.
- Protection contre le downgrade : Les attaques visant à rétrograder une connexion en HTTP (moins sécurisé) échouent, car HSTS oblige le navigateur à refuser toute connexion non sécurisée.

6.8 Sécuriser les cookies

Pour garantir la sécurité des utilisateurs sur **pire2pire.com**, nous mettons en place plusieurs mécanismes pour protéger les cookies.

1. **HttpOnly** : Cet attribut empêche les cookies d'être accessibles via JavaScript, ce qui protège les informations de session contre les attaques de type **Cross-Site Scripting (XSS)**, où des scripts malveillants pourraient voler les cookies.

2. **Secure** : Les cookies marqués comme **Secure** ne sont envoyés qu'à travers des connexions **HTTPS**, garantissant qu'ils ne peuvent pas être interceptés sur des connexions non sécurisées (**man-in-the-middle**).

3. **SameSite** : L'attribut **SameSite** limite l'envoi des cookies uniquement aux requêtes provenant du même site, réduisant ainsi les risques d'attaques par **Cross-Site Request Forgery (CSRF)**. Nous utiliserons une configuration stricte pour les pages sensibles :

    - **SameSite=Lax** pour un bon équilibre entre sécurité et fonctionnalité.
    - **SameSite=Strict** pour les actions particulièrement sensibles.

4. **Expiration contrôlée** : Les cookies de session auront une durée de vie limitée. Cela permet de protéger les utilisateurs en forçant une expiration après un certain temps d'inactivité, réduisant ainsi le risque d'abus en cas de vol de cookies.

6.9 Limiter l'utilisation des bases de données locales aux données non sensibles

Dans le cadre de la sécurisation de **pire2pire.com**, il est essentiel de veiller à ne pas stocker d'informations sensibles dans les bases de données locales du navigateur, comme **localStorage** et **sessionStorage**. Ces espaces de stockage sont accessibles directement via JavaScript, ce qui les rend vulnérables aux attaques **Cross-Site Scripting (XSS)**.

1. **Pas de données sensibles** : Les informations sensibles telles que les mots de passe, les données d'authentification ou les informations de paiement ne doivent jamais être stockées dans **localStorage** ou **sessionStorage**. Ces types de données doivent être gérés par des cookies sécurisés (avec les attributs **HttpOnly** et **Secure**) ou directement sur le serveur.

2. **Données non critiques** : Le **Web Storage** doit être réservé uniquement aux données dont la perte ou la divulgation n'aurait aucune conséquence importante pour la sécurité de l'utilisateur ou de la plateforme. Cela inclut, par exemple, des **préférences utilisateur** (comme le choix du thème) ou des réglages temporaires qui n'impliquent aucune donnée personnelle ou critique.

3. **Vulnérabilité au XSS** : Contrairement aux cookies protégés par **HttpOnly**, les données stockées dans **localStorage** et **sessionStorage** sont facilement accessibles via JavaScript. Cela signifie que si une vulnérabilité **XSS** est exploitée, un attaquant pourrait accéder à ces données et les voler ou les manipuler.

6.10 Défense en profondeur (Navigateur)

Toutes ces diffèrentes sécurités et préventions participe à une défense en profondeur du navigateur de l’application Pire2pire.com :

- Sécurité contre le clickJacking (CSP)
- Subressource integrity
- Strict Mode
- SOP et CORS
- Referrer Policy
- HSTS
- HTttpOnly, secure etc..
- Prévention pour les BDD lcoales

FIN