A22 – Sécurité des applications - Travail pratique 5 (16%)

Objectifs du TP

Ce travail pratique (TP) vise à évaluer votre compréhension des notions vues en cours à savoir :

- Exploitation des faiblesses de type Open Redirect et XSRF/CSRF;
- Mise en place des contremesures pour se protéger de ces types de faille;
- Utiliser Azure API Management pour protéger une API;
- Gestion de la sécurité des dépendances et bibliothèques externes utilisées dans les applications.

Contexte

Ce travail doit être réalisé par groupe de deux ou trois étudiants. La remise est effectuée sur Léa et doit contenir :

- Un rapport au format Word ou PDF;
- Toutes les captures d'écran pour les configurations effectuées
- Le code source de l'application corrigée;
- Le script YAML des pipelines Azure DevOps.

Date de remise

Votre travail doit être remis au plus tard le mardi 6 février 2024 à 23h59.

Critères d'évaluation

Votre travail doit respecter l'ensemble des critères suivants :

- Le code source est optimal et sans erreur;
- Le rapport contient tous les éléments attendus;
- -10 % par jour de retard;
- Note de 0 si le travail est remis après le retour à l'ensemble du groupe ou si le travail a été plagié en tout ou en partie.

Grille d'évaluation

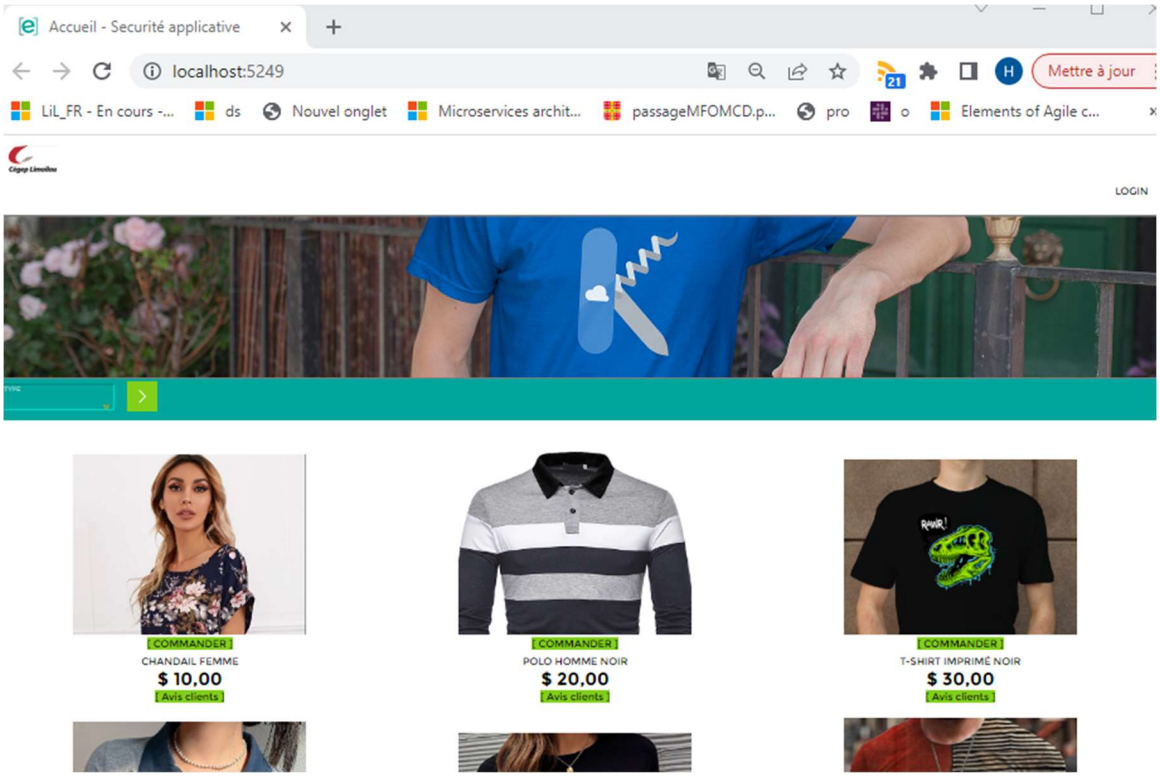
	Excellent	**Fonctionnel**	**Minimal**	**Insuffisant**
Capacité 1 : S'approprier les notions de sécurité informatique	• Les failles de sécurité sont parfaitement présentées; • Les conséquences d'un exploit sont parfaitement identifiées et décrites; • Les facteurs de risques sont parfaitement identifiés et décrits	• Les failles de sécurité sont bien présentées; • Les conséquences d'un exploit sont bien identifiées et décrites; • Les facteurs de risques sont bien identifiés et décrits	• Les failles de sécurité sont partiellement présentées; • Les conséquences d'un exploit sont partiellement identifiées et décrites; • Les facteurs de risques sont partiellement identifiés et décrits	• Les failles de sécurité sont peu détaillées; • Les conséquences d'un exploit sont peu décrites; • Les facteurs de risques sont peu identifiés et décrits
Capacité 2 : Sécuriser les applications	• Les failles dans le code ont été identifiées parfaitement; • Les correctifs à mettre en place ont été parfaitement définis; • Les modifications apportées au code sont optimales; • L'authentification via l'APIM est toujours correctement utilisée;	• Les failles dans le code ont été identifiées correctement; • Les correctifs à mettre en place ont été définis correctement; • Les modifications apportées au code sont presque optimales; • L'authentification via l'APIM est presque toujours correctement utilisée;	• Les failles dans le code ont été partiellement identifiées; • Les correctifs à mettre en place ont été partiellement définis; • Les modifications apportées au code sont partiellement optimales; • L'authentification via l'APIM est la plupart du temps correctement utilisée;	• Les failles dans le code sont rarement identifiées; • Les correctifs à mettre en place sont rarement définis; • Les modifications apportées au code ne sont pas optimales; • L'authentification via l'APIM est la rarement utilisée;
Capacité 3 : Tester la sécurité des applications	• Identification de tous les éléments qui doivent être testés dans l'application; • La revue du code source est correctement réalisée.	• Identification de presque tous les éléments qui doivent être testés dans l'application; • La revue du code source est presque correctement réalisée.	• Identification partielle des éléments qui doivent être testés dans l'application; • La revue du code source est partiellement réalisée.	• Identification insuffisante des éléments qui doivent être testés dans l'application; • La revue du code source est rarement réalisée.

Mise en contexte

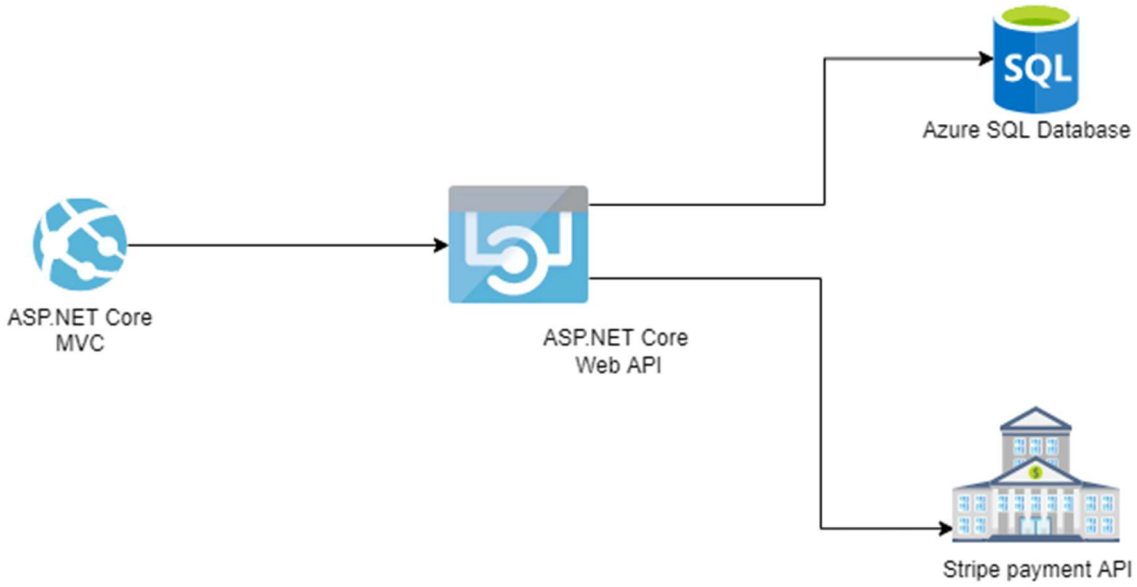
Vous avez été recruté par une entreprise commerciale qui est sur le point de lancer son site de commerce en ligne. Votre mission est d'auditer le code source de l'application en accordant une attention particulière à la sécurité. Vous devez apporter les corrections nécessaires à l'application avant que cette dernière ne soit déployée.

Il est préférable d'utiliser votre propre instance de base de données pour exécuter à l'application. Pour cela, vous devez :

- Créer une base de données Azure SQL Database;
- Modifier le fichier appsettings.json de l'API pour ajouter la chaine de connexion;
- Exécuter le script de migration (Update-Database);
- Définir les deux applications comme projets de démarrage;
- Exécuter.



L'image ci-dessous vous donne une idée de la structure de la solution :



Travail à faire

Vous pouvez utiliser comme projet de départ la solution initiale mise à votre disposition.

Partie 1 (5 points)

L'application est vulnérable à des attaques de type Open Redirect et de falsification de requête intersite (XSRF/CSRF).

Après avoir effectué un audit de l'application, vous devez :

- Produire un rapport qui documente les failles ci-dessus identifiées dans le code source. On doit y retrouver : les portions de code vulnérables, les risques pour l'application et comment est-ce qu'un pirate pourrait exploiter les vulnérabilités (1.5 point);
- Décrire dans le rapport les mécanismes offerts par ASP.NET Core pour se protéger de ces vulnérabilités. Vous devez inclure vos sources dans le rapport (1.5 point);
- Modifier le code de l'application pour se protéger contre ces failles (2 points).

Partie 2 (5 points)

Déployez l'API dans Azure App Service. Ensuite mettez en place Azure API Management pour sécuriser l'API en respectant ce qui suit :

- Vous devez créer un produit TravailPratique exposer l'API;
- Vous devez mettre en place une politique qui protège l'API en utilisant l'authentification OAUTH 2.0 (3 points);

- Vous devez modifier votre application MVC pour s'authentifier auprès de l'API. On doit pouvoir tester le MVC en l'exécutant localement (2 points).

N.B. : Inclure toutes les captures d'écran de vos configurations de l'API Management

Partie 2 (6 points)

Les projets .NET reposent sur de nombreuses dépendances qui sont développées par des développeurs tiers. Contenu tenu du fait que les vulnérabilités dans ses dépendances peuvent exposer l'application à des cyberattaques, on souhaiterait mettre en place l'analyse en continu des dépendances dans la solution.

L'équipe responsable du projet à l'embarras de choix entre DotNet CLI et Mend Bolt. On vous interpelle donc pour réaliser une étude comparative des deux produits. Vous devez pour cela :

- Implémenter deux pipelines Azure DevOps de build donc l'un utilise Mend Bolt et l'autre le DotNet CLI (2 points);
- Documenter dans un rapport le résultat d'analyse de Mend Bolt et DotNet CLI. Joindre les captures d'écran du rapport de chaque outil (1.5 point);
- Faire une comparaison des deux produits (1.5 point);
- Recommander l'un des produits. Vous devez justifier votre choix. (1 point)

Bon travail!