SECG4

Projet Instant Messaging

54024 - Arno Pierre Pion

54456 - Damiano Deplano

2020 - 2021

Table des matières

Introduction

Nous avons développé un logiciel de messagerie instantanée sécurisé.
Le projet est composé d'un client en ligne de commande et d'un serveur.

Le serveur est une API REST qui permet de gérer l'inscription, la connexion d'un utilisateur. Chaque utilisateur peut demander d'ajouter un autre utilisateur en amis par son intermédiaire. Il gère aussi les données permettant de connecter chaque utilisateur entre eux comme l'ip et le port et stock la clé publique de chacun d'eux.

1. Core

Introduction

Le projet core contient toute les class DTO pour permettre d'échanger plus facilement à travers l'api

Models

Users

- l'identifiant unique pour la base de données
- le nom d'utilisateur unique
- le mot de passe
- le salt (devrait être fusionné avec le champ de mot de passe)
- la date d'expiration du mot de passe (pas utilisé par manque de temps)

Friends

- l'identifiant unique d'un des deux utilisateurs de la relation
- l'identifiant unique de l'autre utilisateur
- le statut de la relation
    - En attente
    - Accepté
    - Bloqué

Logs

- un identifiant unique pour la base de données
- un type de log
  un énumérateur est disponible mais pas obligatoire, le champ est libre.
    - Erreur
    - Avertissement
    - Info
- le message

Peers

- l’identifiant unique de l’utilisateur
- l’IPv4
- l’IPV6
- le port
- le dernier heartbeat reçu par le serveur

PublicKeys

- l'identifiant unique de l’utilisateur
- sa clé publique xml sous forme binaire
- la date d’enregistrement de la clé

Tokens

- l’identifiant unique de l’utilisateur
- son JWT au niveau de l’api et IDToken au niveau de la base de donnée
- la date d’expiration du token

2. Serveur

Introduction

Le serveur est considéré comme un tiers de confiance, il doit permettre de connecter les utilisateurs entre eux de manière sécurisée et garantir l'intégrité des données.

Nous avons décidé de développer le serveur comme une API REST lié à un SGBD SQL.

Requête HTTPS

Chaque requête est faite en https et est accompagnée d’un JWT (JSON Web Token) pour permettre de garantir l'intégrité des données d’authentification envoyées par le client.

JWT

Le JSON Web Token contient le nom de l'utilisateur ainsi que son IDToken. Il est composé d'une partie publique visible de tous et d'une partie encodée par le serveur, les deux parties contiennent les mêmes informations et permettent donc au serveur de savoir si les données ont été manipulées pendant le transfère.

IDToken

Il est un identifiant unique généré par le serveur pour authentifier un utilisateur sans qu'il ait à envoyer son login mot de passe à chaque requête.

Logs

Il permet de garder en mémoire les actions suspectes comme les tokens incorrects mais aussi de savoir un retour sur les requêtes erronées pour avoir un retour sur l'utilisation de l'API et de possible futur correction.

Police des mots de passe

Le serveur ne permet pas de s'inscrire avec un mot de passe trop faible, il faut absolument qu'il fasse une taille entre minimum 8 et maximum 255 caractères, qu'il soit composé d'au minimum une lettre minuscule, une majuscule, un chiffre et un caractère spécial.

Entrées API

UsersController

- Inscription
  Chaque utilisateur doit avoir un nom unique.
  Si l'inscription est validée, un JWT avec un IDToken est retourné.
  Si le mot de passe ne respecte pas la police, une erreur avec les règles est retournée.
  Sinon une erreur 401 Unauthorized.

- Connexion
  Si la connexion est validée, un JWT avec un IDToken est retourné.
  Sinon une erreur 401 Unauthorized.

FriendsController

- GetFriendList
  Retourne la liste d'ami de l'utilisateur.

- GetPendingrequest
  Retourne la liste des demandes d'amis en attente.

- ActionOnPendingrequest
  Permet de faire une action sur les demandes en attente via l'id utilisateur.
  Accepter, pour devenir ami.
  Refuser, pour effacer la demande.
  Bloquer, pour empêcher des futures demande

- SendRequest
  Permet d'envoyer une demande d'amis via le nom d'utilisateur.

KeysController

- GetFriendKey
  Permet de récupérer la clé publique d'un ami via son id utilisateur.

- SendMyKey
  Permet d'enregistrer sa propre clé publique dans l'annuaire de clé du serveur.

PeersController

- GetPeer
  Permet de récupérer les informations de connexion directe d'un ami par son id utilisateur.

- PostMyPeer
  Permet d'enregistrer ses propres informations dans l'annuaire du

serveur.

- UpdateHeartBeat
  Permet de signaler au serveur que nous sommes toujours en ligne.

3. Client

Pages

Un système de menu permet de naviguer entre les différentes pages.
Les menus sont numérotés.

Rest

Le client interagit avec le serveur via un client REST.
Nous utilisons la bibliothèque RESTSharp qui est réputée pour sa simplicité et sa flexibilité.

TCP

Les clients communiquent entre eux en peer to peer via des sockets TCP.
Un en écoute pour recevoir les messages et un en envoi.
L'IP et le port d'écoute sont mis à disposition dans l'annuaire du serveur.

Tokens

Le client à la connexion reçoit un IDToken du serveur pour pouvoir interagir avec les différents point d'entrée de l'API

Registration

A l'inscription de l'utilisateur une clé privée est générée coté client et stockée dans une base de données SQLite locale, la clé publique déduite est envoyée dans l'annuaire de clé sur serveur.

Amis

Le client permet de gérer les demandes d'amis.
Les demandes se font via le nom de l'utilisateur, l'ami potentiel peut ensuite accepter, refuser ou bloquer toute future demande.

Messageries

Les messages envoyés et reçus sont cryptés à l'aide de clé publique clé privé(RSA) et ils sont enregistrés exclusivement dans la base de donnée locale pour historique.
Les utilisateurs ne peuvent contacter uniquement que leurs amis et uniquement si ceux-ci sont en ligne.

RSA

Nous utilisons un système de clé privé et publique pour l'échange de message

4. Conclusion

1. Est-ce que j'assure correctement la confidentialité ?

    - Les données sensibles sont-elles transmises et stockées correctement ?
    - Les requêtes sensibles envoyées au serveur sont-elles transmises de manière sécurisée ?

      - Les données échangées entre le client et le serveur se font uniquement en https, l'IDToken transite uniquement via un JWT token qui permet de garantir que les données n'ont pas été manipulées pendant l'échange, de plus le token expire tout les 2h ou à la prochaine connexion de l'utilisateur.

      - Celles échangées entre clients sont chiffrées via la clé publique du destinataire et stockées localement dans la base de données sqlite, chiffrée avec la clé publique de l'utilisateur courant.

2. Est-ce que je m'assure de l'intégrité des données stockées ?

- Les données sont stockées dans une base de données MariaDB ou MySQL côté serveur, l'intégrité des données dépend donc d'elle.
- Le client stock les données chiffrées avec la clé publique de l'utilisateur, elles ne peuvent donc pas être modifiées, mais peuvent être corrompues/illisibles facilement.

3. Est-ce que je m'assure correctement de la non-répudiation ?

- Le serveur étant un tiers de confiance, et la communication étant en https, l'envoie des données de connexion de chaque peer (client) est donc considéré comme fiable.
- Le JWT token et son IDToken permettant d'authentifier la provenance original des informations de peer envoyer au serveur.

4. Mes fonctions de sécurité reposent-elles sur le secret, au-delà des clés cryptographiques et des codes d'accès ?

- Le mot de passe utilisateur est inscrit sur la base de données du serveur salé et haché.

5. Suis-je vulnérable aux injections ?

- Injections d'URL, SQL, Javascript et parser dédié
  - Nous utilisons l'ORM Entity framework core et le langage de requête Linq, nous ne manipulons jamais de requête SQL textuelle mais uniquement des entitées en mémoire, il ne peut donc pas y avoir d'injection SQL. https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/ef/security-considerations?redirectedfrom=MSDN
  - Si un JSON n'est pas parser correctement ou un url mal formé, ASP.NET ne trouvera pas la route ou d'entrée dans l'api correspondant et la requête sera rejetée.
  - Nous n'utilisons pas de langage web et donc une entrée javascript ne sera jamais exécutée.

6. Suis-je vulnérable aux attaques par rémanence des données ?

- Nous avons limité l'attaque par rémanence en limitant dans le temps la durée des tokens.

7. Suis-je vulnérable à la falsification de demandes frauduleuses ?

- le JWT token permet de certifier que les demandes n'ont pas été manipulé pendant le transfère
- Chaque utilisateur a un IDToken unique et éphémère.

8. Suis-je surveillant suffisamment l'activité des utilisateurs pour pouvoir détecter les intentions malveillantes ou analyser une attaque a posteriori ?

- Toute requête non autorisée, comme avec un token invalide ou une requête vers des données qui ne lui appartiendrait pas est loggé en base de données.

9. Est-ce que j'utilise des composants avec des vulnérabilités connues ?

- Entity framework core étant un reprise à zéro de l'Entity framework d'origine, il se peut qu'il y ai des vulnérabilités non connues

10. Mon système est-il mis à jour ?

- Toute nos dépendances venant du gestionnaire de paquet Nuget, ils peuvent être mis a jours facilement.

11. Mon contrôle d'accès est-il cassé (cf. OWASP 10) ?

- Chaque point d'entrée de notre api demande un JWT valide, ensuite nous vérifions que l'IDToken à l'intérieur est bien valide lui aussi.
- Si un des deux token n'est pas valide ou si un utilisateur essaie de manipuler les données d'un autre, l'api renvoie une erreur 401 Unauthorized.
- Les utilisateurs ne peuvent manipuler que leurs propres données et interagir uniquement qu'avec leurs amis.
- Toutes les actions interdites sont loggé dans la base de données.

12. Mon authentification est-elle cassée (cf. OWASP 10) ?

- Nous avons mis en place une politique de mot de passe fort 8-255 caractères, au moins une majuscule, une minuscule, un chiffre et un symbole.
- Chaque mot de passe est stocké, salé et haché.
- La durée de vie limitée du token ne permet pas de le réutiliser sur le long terme.
- Une route, un url incorrect ou une donnée ne matchant pas avec une entrée de l'api est ignorée par ASP.NET core.
- Les IDToken sont des identifiants unique de 128bits(GUID) et éphémères.
- Tous les contrôles d'authentification se font côté serveur.

13. Mes fonctionnalités de sécurité générales sont-elles mal configurées (cf. OWASP 10)?

- La taille de la clé RSA générer et laissé en automatique pour quelle puisse évoluer au fur et à mesure de l'évolution du framework .NET et quelle garde une taille optimal, elle est pour le moment de 2048 bits
- La sécurisation de la base de données dépend grandement de la capacité de sécurisation de l'utilisateur final, nous n'avons pas la main dessus.
- La clé symétrique pour les JWT est stockée dans le fichier de configuration, car nous n'avons pas les connaissances nécessaires à l'utilisation des différents trousseaux de clé pour une application cross-platform.