UNIVERSITÉ DE SHERBROOKE
Faculté de génie
Département de génie électrique et génie informatique

# Coup De Sonde

Programmation sécurisée
GEI771

Présenté à
Guy Lépine

Présenté par
Équipe numéro 12
Jérémy Bélec - belj2434
Antoine Mascolo - masa3303

Sherbrooke - 21 septembre 2022

# Table des matières

# 1 Diagramme de flux de données

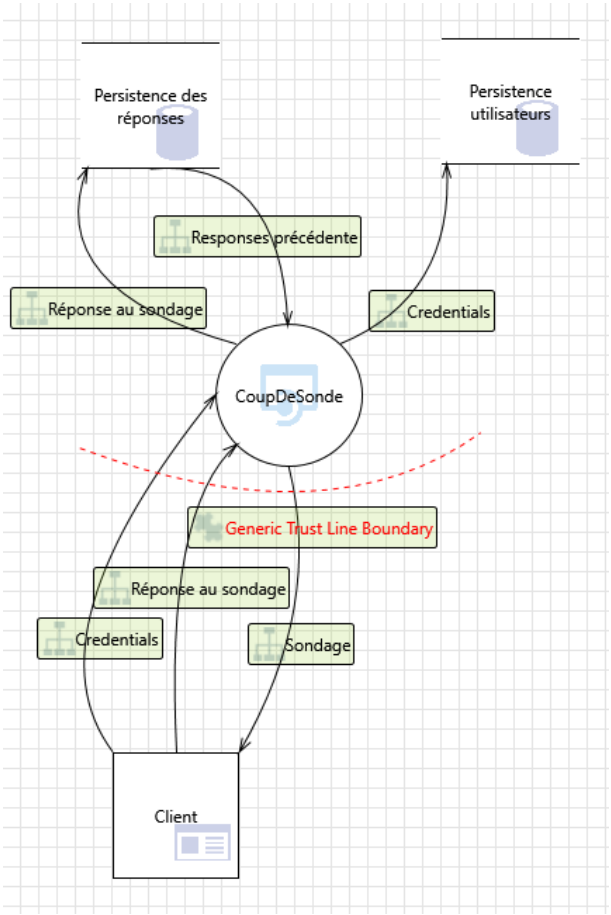


FIGURE 1 – Diagramme de flux de données

## 1.1 Justification des frontières de confiance

Nous avons placé une barrière de confiance entre l'application et le monde extérieur, c'est-à-dire le client. Cette frontière existe puisqu'on ne doit pas faire confiance à tout l'internet. La communication donc être authentifié et encrypter et ce qui est partagé au travers de cette frontière sont uniquement les informations strictement nécessaires.

Nous avons décidé de ne pas exclure les fichiers interne de la zone de confiance puisqu'ils sont livrés avec l'application et que ceux-ci sont encrypté. Dans le cas où ces fichiers seraient plutôt des bases de données sur un serveur externe, ceux-ci devraient alors être exclues de cette zone de confiance pour se conformer à l'architecture zero trust.

## 1.2 Authentification par bearer token

Tout d'abord, puisque nous voulons sécuriser la majorité des routes sensibles, nous avons opté pour une authentification par jeton JWT. Les jetons JWT sont une méthode rapide de valider un utilisateur et attacher des données à une session éphémère. Pour notre application, le serveur n'a pas à garder en session l'utilisateur et risquer d'exposé des données confidentiel puisque chaque requête est indépendante de la précédente. Cela permet aussi d'avoir plusieurs modules indépendants qui n'ont pas besoin de partager une session commune. De plus, le JWT permet de rapidement valider la signature de celui-ci avec une clé privée utilisée pour signer chaque jeton. Puisqu'un jeton est unique et associé à un seul utilisateur, il permet l'accès aux ressources de l'utilisateur sans donner une clé utilisable pour tous les utilisateurs.

Sans l'utilisation d'une clé d'API comme les jetons JWT, notre API serait vulnérable aux attaques de spoofing puisqu'un utilisateur malveillant pourrait se présenter comme un utilisateur quand il est en réalité un autre. Cela pourrit permettre à un intervenant malveillant d'obtenir des accès, qu'il n'aurait pas normalement. En effet, comme indiquer dans l'architecture zero trust les diverses entités doivent uniquement avoir les accès strictement nécessaire et rien de plus.

Cette mitigation nous permet de mitiger les vecteurs d'attaque suivants :

- 1. An adversary may gain unauthorized access to Web API due to poor access control checks
- 8. An adversary may spoof Client and gain access to Web API
- 25. An adversary may gain unauthorized access to Web API due to poor access control checks
- 32. An adversary may spoof Client and gain access to Web API
- 35. An adversary may gain unauthorized access to Web API due to poor access control checks
- 40. An adversary may spoof Persistence des réponses and gain access to Web API

## 1.3 Gestion d'erreurs

Bien que les messages d'erreurs précis peuvent aider les utilisateurs à comprendre comment remédier à un problème, ils peuvent aussi révéler des informations sensibles aux acteurs malveillants. En effet, une mauvaise gestion des erreurs qui permet à la trace de la pile de remonter jusqu'à l'utilisateur peut exposer publiquement l'architecture de l'application ainsi que les interactions entre les différents modules. L'impact peut être encore plus grand si les messages contiennent des valeurs de variables stockées en mémoire comme les strings de connections, des mots de passe ou des noms d'utilisateurs. Dans de mauvaise main, ces informations peuvent, entre autre, faciliter les attaques comme l'injection de SQL. Ces informations peuvent aussi aider un compétiteur malhonnête à faire de la rétro ingénierie sur notre application.

Pour prévenir ce genre d'attaque, la solution est tout simplement une meilleure gestion des erreurs. Tout d'abord, on veut attraper les erreurs le plus proche de la source possible pour la garder dans son contexte et ainsi faciliter le déverminage. On veut aussi intervenir sur l'erreur proche de la source pour la traiter rapidement et éviter la multiplication d'événement imprévu causé par l'erreur. Pour ce qui est des erreurs retournées à l'utilisateur, on va préférer les messages génériques et renvoyer à la documentation si l'erreur est plus complexe afin de limiter les fuites d'information. Cette mitigation nous permet de mitiger les vecteurs d'attaque suivants :

- 2. An adversary can gain access to sensitive information from an API through error messages
- 26. An adversary can gain access to sensitive information from an API through error messages
- 36. An adversary can gain access to sensitive information from an API through error messages

## 1.4 HTTPS

Comme tout le trafic qui circule sur un réseau peut être vu par tous ceux qui écoute sur le réseau, il est important d'avoir une façon de protégé ces données. Par exemple, si un utilisateur tenterait de se connecter à notre API et que quelqu'un écoute le réseau à ce moment, cette personne pourrait voir passer le mot de passe en clair. Ceci pourrait lui permettre d'usurper l'identité de l'utilisateur. Pour se protéger de ce genre d'attaque, nous avons activé HTTPS qui est la version sécurisée du protocole HTTP. Cela permet d'encrypter le trafic entre l'utilisateur et le serveur pour rendre illisible le trafic sur le réseau.
Cette mitigation nous permet de mitiger les vecteurs d'attaque suivants :

- 4. An adversary can gain access to sensitive data by sniffing traffic to Web API
- 28. An adversary can gain access to sensitive data by sniffing traffic to Web API
- 37. An adversary can gain access to sensitive data by sniffing traffic to Web API

## 1.5 Jetons durée limité

Puisque nous utilisons les jetons JWT, ceux-ci doivent être stocké sur les systèmes de nos clients. Puisque nous ne pouvons pas assurer la sécurité des systèmes des autres, nous devons idéalement nous protéger contre le vol de jeton et réduire l'impacte que cet situation peut avoir sur notre système. Le vol de jeton pourait permettre à un utilisateur d'usurpé l'identité d'un autre. Pour protéger contre cette menace, nous avons opté de limité la durée de vie des jetons. Cet procédure de mitigation force l'utilisateur à se ré-authentifier à chaque 24h et donc limite le risque que le jeton serait valide dans le cas d'un vol de cookies/données.

Cette mitigation nous permet de mitiger les vecteurs d'attaque suivants :

- 3. An adversary may retrieve sensitive data (e.g, auth tokens) persisted in browser storage

- 27. An adversary may retrieve sensitive data (e.g, auth tokens) persisted in browser storage

## 1.6 Encryption

Les fichiers de configuration contienne des informations qui peuvent être critique au bon fonctionnement de l'application. Si un acteur malveillant parvient à mettre la main sur ces fichiers, il serait en mesure d'empêcher l'application de fonctionner. Si l'adresse d'une base de données est stocké dans ce fichiers l'attaquant pourrais aussi modifier cette adresse et rediriger le trafic vers sa propre base données. Il n'y a pas seulement les fichiers de configuration qui contienne des informations sensibles, tout information stocker dans un fichier localement pourrait être vulnérable.

Afin de prévenir ce genre d'attaque, les fichiers de configuration et autre base de données doivent absolument être encrypter. Ainsi, dans le cas ou quelqu'un parvient à avoir accès à ces fichiers, l'information serait tout de même protéger. Il est important lors de l'encryption d'utiliser des algorithmes à jour avec des clés de longueurs suffisantes. Cette mitigation nous permet de mitiger les vecteurs d'attaque suivants :

- 5. An adversary can gain access to sensitive data stored in Web API's config files
- 13. An adversary can gain access to sensitive PII or HBI data in database
- 20. An adversary can gain access to sensitive PII or HBI data in database
- 29. An adversary can gain access to sensitive data stored in Web API's config files
- 38. An adversary can gain access to sensitive data stored in Web API's config files

## 1.7 Activer les Journaux d'événement

L'activation des journaux permet de garder des traces de toutes les activités qui sont fait sur nos applications. L'archivage de ces événements lié avec des outils peut permettre la détection d'activité suspicieuse. En effet, cela pourrait entre autre permette de voir une tentative de force brute sur un mot de passe en détectant plusieurs tentatives de connections sur un même compte ou encore détecté si un utilisateur tente d'injecter des commandes en observant des formats spécifiques. Avec cette information, il est alors possible d'intervenir et de bloqué certains utilisateurs ou certaines adresses qui tentent d'attaquer notre application avant même que cela ne se produise. En plus de protèger notre application cela protège aussi les clients d'un éventuelle usurpation d'identité.

- 6. Attacker can deny a malicious act on an API leading to repudiation issues
- 15. An adversary can deny actions on database due to lack of auditing
- 22. An adversary can deny actions on database due to lack of auditing
- 17. An adversary may leverage the lack of monitoring systems and trigger anomalous traffic to database

- 30. Attacker can deny a malicious act on an API leading to repudiation issues
- 39. Attacker can deny a malicious act on an API leading to repudiation issues

## 1.8 CORS

CORS (Cross-Origin Resource Sharing) est un mécanisme basé sur les entêtes de réponse provenant d'un serveur. L'utilité est d'indiquer au navigateur quelles ressources devraient être considéré pour le chargement. Par exemple, le serveur *example.com* pourait indiquer au navigateur de seulement charger du contenu externe provenant de *backend.example.com* pour eviter les attaques de vol de données. Dans un cas contraire, si le serveur de *example.com* est mal configuré, celui-ci pourrait être exploité si un utilisateur réussi a injecté du code malicieux sur un site externe et l'exécuter pour voler la session ou les cookies de l'utilisateur sur *example.com*.

Pour notre problématique, nous avons opté pour limiter la politique CORS aux requêtes provenant de localhost. Dans un cas réel, il faudrait adapter la politique au serveur sur lequel le système serait déployé.

Cette mitigation nous permet de mitiger les vecteurs d'attaque suivants :

- 7. An adversary can gain unauthorized access to API end points due to unrestricted cross domain requests
- 31. An adversary can gain unauthorized access to API end points due to unrestricted cross domain requests

## 1.9 Filtrer les entrées

La sanitation des entrées est une des stratégies de mitigation la plus efficace, mais aussi une des plus faciles a mal implémenté. La sanitation des entrées consiste à filtrer le contenu reçu des utilisateurs externes au système. Puisque nous devons présumer que toutes données externes au système peuvent potentiellement être dangereuses, il est critique au système de correctement filtrer les entrées reçues. Dans un cas où les entrées sont mal sécurisées, un utilisateur malicieux pourrait potentiellement exécuter du code à distance sur nos systèmes (Remote code exécution) et m'être en danger la sécurité de notre système et potentiellement les informations confidentielles des autres utilisateurs. Le plus important lors de la sanitation des entrées est vraiment d'assurer une couverture complète puisque la sécurité du système est aussi forte que la porte la plus faible. Pour ces raisons, il est primordial de bien déterminé les lignes de confiances entre les utilisateurs et le système et sécurisées/valider tous les échanges entre les partis.

Cette mitigation nous permet de mitiger les vecteurs d'attaque suivants :

- 8. An adversary may spoof Client and gain access to Web API
- 9. An adversary may inject malicious inputs into an API and affect downstream processes
- 10. An adversary can gain access to sensitive data by performing SQL injection through Web API
- 14. An adversary can gain access to sensitive data by performing SQL injection
- 16. An adversary can tamper critical database securables and deny the action

## 1.10 Out of scope

Certains autres vecteurs d'attaque soulevée par l'outil de modélisation de la menace n'était pas pertinant à notre situation. Par exemple, nous n'utilisons pas de serveur SQL dédié nous ne sommes donc pas vulnérable aux injections SQL. Pour la même raison, nous n'avons pas à nous occuper de l'authentification à la base de donnée.

Voici les vecteurs d'attaque ignorés :

- 10. An adversary can gain access to sensitive data by performing SQL injection through Web API
- 11. An adversary can gain unauthorized access to database due to lack of network access protection
- 12. An adversary can gain unauthorized access to database due to loose authorization rules
- 14. An adversary can gain access to sensitive data by performing SQL injection
- 16. An adversary can tamper critical database securables and deny the action
- 18. An adversary can gain unauthorized access to database due to lack of network access protection
- 19. An adversary can gain unauthorized access to database due to loose authorization rules
- 21. An adversary can gain access to sensitive data by performing SQL injection
- 23. An adversary can tamper critical database securables and deny the action
- 34. An adversary can gain access to sensitive data by performing SQL injection through Web API
- 42. An adversary can gain access to sensitive data by performing SQL injection through Web API