Les menaces auxquelles s’exposent vos API

1 - Identification

L’identification est plus difficile que via une logiciel classique. Veiller à ce qu’un utilisateur n’ait accès **qu’à ses ressources uniquement**.

Exemple : Que se passe-t-il si je ne suis pas identifié et que j’envoie la requête suivante :

GET https://api.monsite.com/users/117

Si l’utilisateur 117 ne me concerne pas, l’API doit me retourner un statut 401 unauthorized

2 - Abus du nombre d’exécution

Votre API peut être appelée via un robot qui peut faire des millions d’appels par jour.

Dans le cas de création de data, cela peut être très problématique d’un point de vue logiciel et budget.

Exemple : un logiciel créé 10 000 utilisateurs par heure.

Mon serveur SQL peut-il gérer la charge ? Si je suis moi-même interfacé avec une API payante, que se passe-t-il ? Dans le cas d’un hébergeur type AWS, la consommation de ressource est facturée.

3 - Indisponibilité du service

Inondation de votre API avec un grand nombre de requêtes simultanées pour obtenir un déni de service.

Exemple : Mon serveur peut-il encaisser 10 000 requêtes simultanées pendant 20 minutes ?

4 - Man in the middle

Interception par une partie tierce de données entre votre api et le logiciel utilisateur.

Exemple : Un attaquant réussit à écouter votre réseau pour envoyer des données à votre logiciel client en se faisant passer pour vous et vice versa.

5 - Injection

Envoi de données visant à détourner votre logiciel de son utilisation principale.

Exemple :

- Utilisation normale : GET https://api.monsite.com/users/117
- Utilisation déviée : GET https://api.monsite.com/users/117;drop table users;

Sans contrôle, le résultat en SQL pourrait être :

SELECT * FROM users WHERE id = 117;drop table users;

Les vulnérabilités les plus fréquentes

1 - Problèmes de ressources

Attention à l'utilisation des ressources. Comme pour le web, nous devons **toujours** contrôler la qualité et la quantité de données renvoyées.

- **Toujours** prévoir un système de pagination.
- **Séparer** les ressources pour éviter de les surcharger (approche microservices vs monolithique).

2 - Problèmes d’identification

Contrôler à qui les données sont envoyées :

- Prévoir un système d’authentification **fiable** si nécessaire (Oauth par exemple)
- S’assurer de la correspondance entre les données et le destinataire

3 - Absence de quotas

Définir des **quotas** quant à la définition des API et suivre son utilisation. Une augmentation soudaine de l'utilisation de votre API peut indiquer qu'elle est utilisée de manière abusive.

- Protection contre les attaques de déni de service
- Peut éviter des problèmes de pics de trafic

4 - Manque de suivi

Nous devons être en mesure de contrôler qui accès à l'API, sur quelles ressources, et le résultat de la requête.

- L'utilisation des logs peut être une solution suffisante à minima
- Possibilité de rejouer des requêtes ayant échoué

5 - Les timeouts

Ne pas faire patienter le client. On doit être en mesure de lui fournir une réponse rapide.

Exemple du cas de l’envoi d’une image via API:

1. Réception de la requête
2. Nous traitons la requête et le fichier (sécurité, données complètes...)
3. Nous envoyons un retour de succès
4. Nous uploadons l’image ensuite

Evaluation - Projet technique - MAJ 4

Webservice REST - Le CRM

Mise en place d’un système de quotas.

Un administrateur doit attendre une minute entre chaque création d’autres administrateurs.