Document de sécurité

Afin d’assurer qu’un utilisateur accède uniquement aux données dont il a besoin, il doit avoir un identifiant propre à lui et donc s’authentifier avant d’utiliser les différents Endpoint.

Nous avons donc deux types de ressources :

* Publiques
* Privées

En effet, pour les ressources publiques, les informations contenues dans ces ressources sont publiques.

On peut limiter les abus en utilisant une stratégie d’API KEY.

Pour les ressources privées, il s’agit de ressources dont pas tout le monde doit avoir accès. La plupart du temps se sont des ressources propres a un utilisateur.

Donc, pour notre API il nous faut d’abord authentifier, autoriser et tracer.

Le Framework OAuth2 est conçu pour autoriser des applications à consommer des ressources. Voici son vocabulaire spécifique :

* **Client**: l’application écrite par le développeur, qui consomme les ressources de l’API
* **Resource Server (Serveur de ressources**): le resource provider décrit plus haut, qui expose via une API REST les ressources auxquelles le Client veut accéder
* **Resource Owner (Propriétaire de la ressource):** l’utilisateur final dont nous parlons, qui utilise le Client
* **Authorization Server (Serveur d’autorisation):** un nouvel acteur, dont le rôle est de fournir des jetons (tokens) aux clients
* **Scope (Périmètre):** une description de ce que l’utilisateur permet à l’application de faire en son nom
* **Consumer (Consommateur):** le développeur de l’application. Il n’est pas présent explicitement dans la spécification mais est présent dans beaucoup d’implémentations
* **Token (Jeton):** une information opaque (il s’agit d’une suite de caractères qui donne peu envie de la lire) qui relie un client, un ou plusieurs scopes et éventuellement un utilisateur.

L’utilisateur recevra le code http 403 si celui-ci n’est pas autoriser à utiliser ces Endpoint.

Avec OAuth2, un token d’accès est lié à un scope donné. Un scope décrit les limites des actions que le Resource Owner a autorisé le client à faire en son nom.

Le rate limiting sera utilisé pour permettre de réduire les risques d’abus d’utilisation des APIs. Avec OAuth 2 nous pouvons définir une rate limit par ACCESS TOKEN délivré.

Le top 10 de l’OWASP a pour but d’éduquer les développeurs et autres sur les conséquences des vulnérabilités de sécurité les plus communes des applications web .

Voici les manières de sécuriser notre API :

1 **- Cross Site Scripting (XSS)** : la manière la plus efficace de vérifier la sécurité est d'auditer le code.

* **validation d’entrée** : Utiliser un mécanisme standard de validation d'entrée pour valider la longueur, le type et la syntaxe de toute entrée saisie, ainsi que les règles de gestion avant d'accepter l'affichage ou le stockage des données
* Spécifier le codage en sortie (tel que ISO 8859-1 ou UTF 8).
* N'utilisez pas de validation basée sur « blacklist / blacklistage »
* Faites attention aux erreurs canoniques

2 **- Failles d'injection :**

* Utilisez des APIs de requêtes fortement typées
* Respectez le principe du moindre privilège
* Evitez les messages d’erreurs détaillées
* N’utilisez pas d’interfaces de requêtes dynamiques
* N’utilisez pas les fonctions d’échappements simples,

3 - **Exécution de Fichier Malicieux** :

* Utilisez une référence indirecte à l’objet (voyez la section AA pour plus de détails).
* Validez fortement les données d’entrées en utilisant la stratégie d’acceptation uniquement des bonnes valeurs
* Ajoutez des règles à vos Firewalls pour empêcher les serveurs web d’effectuer de nouvelles connexions vers des sites web externes ou internes.
* Vérifiez que les fichiers ou noms de fichiers fournis par l'utilisateur

4 **- Référence directe non sécurisée à un Objet** :

* Evitez d'exposer des références d'objet privé aux utilisateurs, chaque fois que possible, tels les clés primaires ou noms de fichiers
* Validez sans retenue toutes les références aux objets privés, via la méthode d'acceptation des bonnes valeurs
* Vérifiez l'autorisation à tous les objets référencés

5 - **Falsification de requête inter-site (CSRF)** :

* S'assurer qu'il n'y a pas de vulnérabilité de type XSS dans l'application
* Insérer des tokens uniques et aléatoires dans chaque formulaire et URL qui ne seront pas envoyés automatiquement par le navigateur.

6 - **Fuite d'information et Traitement d'erreur Incorrect** :

* Assurez-vous que toute l'équipe de développement partage la même approche relative au traitement d'exceptions
* Désactivez où limiter le traitement d'erreur détaillé.
* Assurez que les « path » sécurisés qui ont des résultats multiples retournent des messages d'erreur similaires ou identiques approximativement dans le même temps.

7 **- Violation de Gestion d'authentification et de Session** :

* Utiliser uniquement des mécanismes de gestion de sessions intégrés.
* N'acceptez pas des identifiants de sessions, nouveaux, préréglés ou invalides à partir de l’URL ou dans la requête.
* Limitez ou supprimez de votre code les cookies personnalisés relatifs à l'authentification ou la gestion de session tels que les fonctionnalités de type « Remember me » ou les systèmes d'Authentification Unique (Single Sign-On - SSO) faits maison

8 - **Stockage Cryptographique non Sécurisé**:

* Ne pas créer d’algorithmes de chiffrement
* Générer les clés hors-ligne et stocker les clés privées avec une extrême précaution. N
* Assurez-vous que les données chiffrées sur disque ne sont pas faciles à décrypter.

9 - **Communications non Sécurisées** :

* Utiliser SSL pour toutes les connexions qui sont authentifiées ou qui transmettent des données sensibles, comme des éléments d’authentification, les détails d’une carte de crédit, d’une carte de santé ou autres informations privées

10 - **Manque de Restriction d’Accès URL** :

* Assurez-vous que la matrice de contrôle d’accès faire partie de la couche métier, de l’architecture et de la conception de l’application
* Ne supposez pas que les utilisateurs ne connaissent pas les URLs ou les APIs spéciales ou cachées
* Bloquez l’accès à tous les types de fichier que votre application n’utilisera jamais.