WAVESTONE

QUELLE RECETTE POUR SECURISER SES API ?

Les APIs sont partout aujourd'hui et permettent aussi bien les échanges à l'intérieur du système d'information que les accès partenaires et clients. Quelles bonnes pratiques de sécurité adopter ?

Ce que l'on appelle aujourd'hui communément APIs ou Application Programming Interfaces regroupe un ensemble de méthodes de communications inter applicative allant des services web (REST ou SOAP) aux appels entre processus (RPC) locaux ou distants. Les services web de ce type, même s'ils ne sont pas les seuls à utiliser des API, se sont répandus comme une trainée de poudre ces dernières années et représentent un mécanisme de communication prépondérant et incontournable pour toutes les entreprises lancées dans leur transformation numérique. Ils sont rencontrés maintenant dans un nombre toujours plus important de cas d'usages : données publiques, personnelles ou sensibles, applications mobiles, échanges entre partenaires, IoT, applications dites « client-side », etc.

Le concept n'est pas nouveau. Dès l'introduction et la définition de la notion d'architecture REST, en 2000, les premières APIs voyaient le jour. Les pionniers ont été eBay notamment ou encore Flickr puis Facebook et Twitter en ont fait le cœur de leur produit sur lequel des développeurs tiers pouvaient venir bâtir leurs propres services. Et dès la naissance de ce concept s'est évidemment posée la question de sécuriser les accès à ce nouveau type de services web.

L'expérience le montre, la sécurisation des APIs est une recette à base de 4 ingrédients qu'il faut savamment doser.

AUTEUR

BERTRAND CARLIER
bertrand.carlier@wavestone.com

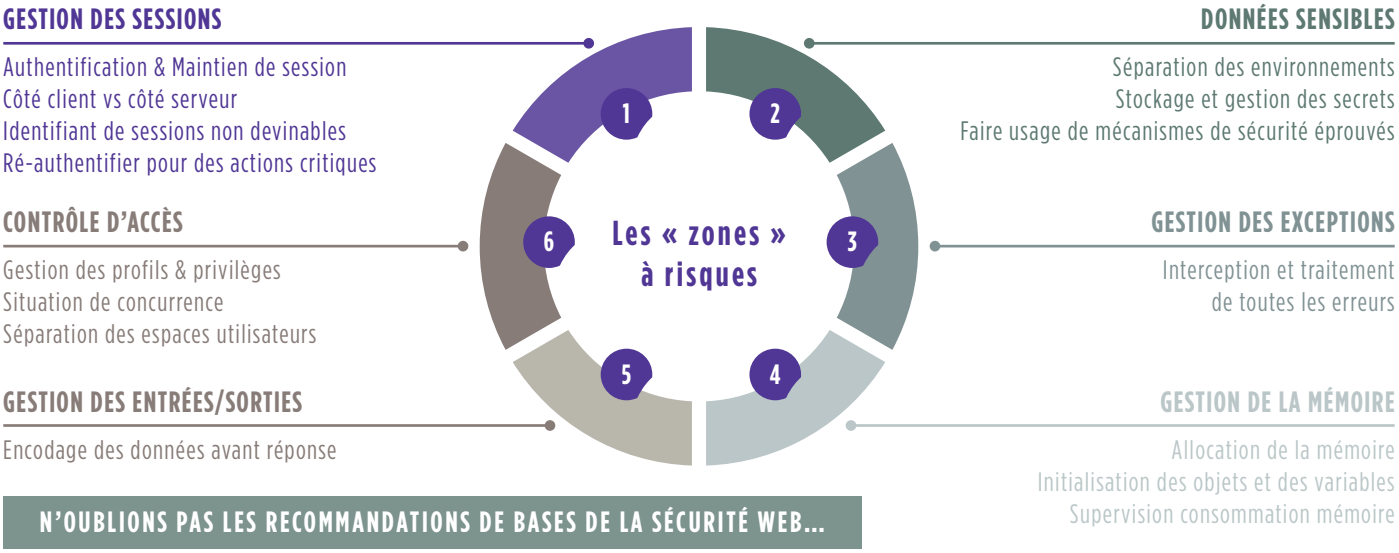
Cette publication a été réalisée avec les contributions de Samantha MARECAUX et Parfait NANGMO.

UNE BASE DE *SECURITY AS USUAL*

Selon un **benchmark Wavestone sur le sujet de la sécurité des applications web**[1], sur 128 applications auditées, des **failles graves sont observées dans 60% des cas.** La situation sur le terrain est très similaire pour les APIs. La réponse est simple mais souvent difficile à mettre en œuvre, les **recommandations habituelles de la sécurité web**, par exemple celles d'**OWASP**[2] doivent être prises en compte de la même manière.

Un certain nombre de mesures de sécurité et bonnes pratiques de développement sont à la disposition des développeurs et équipes d'exploitation pour couvrir les zones à risques traditionnellement visées par un attaquant :

Applications web & APIs – *Security as usual*



N'OUBLIONS PAS LES RECOMMANDATIONS DE BASES DE LA SÉCURITÉ WEB…

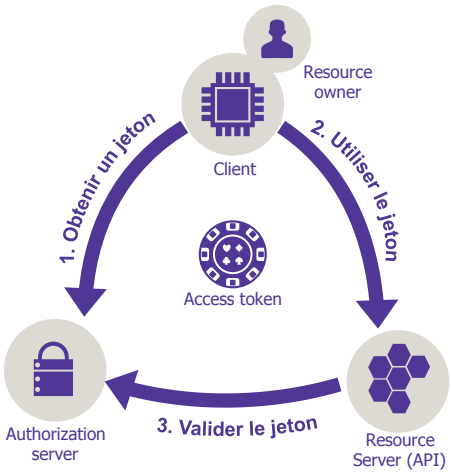
UNE PINCÉE D'OAUTH2

Une fois cette base bien maîtrisée et appliquée, se pose la question de la bonne gestion des accès dans l'application. Il s'agit de déterminer les moyens d'authentification pour accéder à une API (pour authentifier aussi bien l'utilisateur que l'application appelante) et de s'entendre sur un protocole commun entre des acteurs tiers.

OAuth2 est aujourd'hui le standard le plus approprié et le plus répandu dans le cadre d'APIs REST. Il s'agit d'un **standard de délégation d'autorisation** qui permet à une application d'obtenir **l'autorisation d'accès à une ressource (API) au nom d'un utilisateur.**

OAuth2 se propose de couvrir un large éventail de cas d'usages (applications web, mobile, accès ou non via un navigateur, accès serveur-à-serveur, etc.) et offre à cet effet :

- **4 cinématiques principales pour obtenir un jeton (RFC 6749)**
- **3 mécanismes d'utilisation de ce jeton (RFC 6750)**
- **Un document à part entière détaillant le threat model et les bonnes questions à se poser dans l'implémentation d'OAuth2 au sein d'une architecture**



Enfin, une surcouche dédiée à l'authentification complète ce premier ensemble : **OpenID Connect**[3]. Ce standard permet de contrôler plus finement les caractéristiques de l'authentification de l'utilisateur (moyens d'authentification, Single Sign-On, transmission d'attributs d'identité dans un format standard, réauthentification forcée, etc.)

Si l'on se limite à ces 4 documents, qui représentent déjà l'équivalent de 250 pages, on comprend plus facilement la mauvaise réputation d'OAuth2 : un protocole complexe, lourd et sujet à erreurs d'implémentation.

Cette réputation n'est pas entièrement usurpée : de grands acteurs du Web, tels Facebook ou Twitter, s'y sont eux-mêmes cassés les dents, et ont vu des données personnelles de leurs utilisateurs accessibles disponibles sans authentification préalable[3].

Il faut bien comprendre que ce n'est pas le protocole en soi qui est en cause, il est heureusement tout à fait possible d'implémenter OAuth2 de manière sécurisée, mais une abondance d'options dans l'implémentation qui si elles sont mal évaluées et

1- *https://www.wavestone.com/app/uploads/2016/10/Benchmark-Securite-Web-1.pdf*
2- *https://www.owasp.org/index.php/OWASP_Cheat_Sheet_Series*
3- *https://openid.net/specs/openid-connect-core-1_0.html*

choisies aboutissent à des failles critiques : usurpation d'identité d'une application, accès aux données personnelles d'un utilisateur tiers, vol de cookie Facebook/ Google lors d'un social login ou encore compromission de compte utilisateur.

Quelques recommandations ci-dessous permettent d'offrir un premier niveau de sécurité à votre implémentation :

- **Secret local :** L'application cliente est munie d'identifiants lui permettant de s'authentifier auprès du serveur OAuth, il ne faut pas mettre ce secret (identifiant du service) dans l'application mobile ou le considérer compromis
- **Redirect URI :** Valider strictement les URLs de redirection vers l'application, sans caractère générique (type wildcard)
- **Implicit :** Éviter le flow OAuth2 « Implicit » dont la sécurité est discutable (des tokens présent dans les URLs peuvent être dans l'historique du navigateur par exemple) ainsi que l'ergonomie lors de l'expiration des jetons
- **Authorization code :** Valider strictement les authorization codes : un code ne doit être vérifié qu'une seule fois et uniquement par le client auquel il était destiné.
- **State and PKCE :** utiliser ces options du protocole pour garantir l'intégrité d'une cinématique complète
- **Authorization ≠ Authentication:** Utiliser **OpenID Connect**[4] pour authentifier, OAuth pour déléguer l'accès

LIMITEZ LES ADDITIFS

À peine cette première pincée d'OAuth digérée, nécessaire pour débuter, se posent déjà des questions sur des cas d'usages très fréquents

LE SINGLE SIGN-ON MOBILE, OU COMMENT PERMETTRE À DES EMPLOYÉS EN MOBILITÉ OU DES CLIENTS D'ACCÉDER AISÉMENT À PLUSIEURS APPLICATIONS SANS SE RÉAUTHENTIFIER ?

Qu'il s'agisse d'un agent terrain en contact clientèle ou en tournée d'intervention qui peut utiliser plus d'une dizaine d'applications par jour ou qu'il s'agisse d'un client ayant installé plusieurs applications sur le store public, le besoin d'accéder à l'ensemble des applications sans avoir à se réauthentifier sur chacune est aujourd'hui très présent. Depuis 2008, les tec hniques le permettant ont varié au gré des possibilités offertes par les OS mobiles (KeyChain iOS, paramètres d'URL, Mobile Device Management…). Néanmoins, Apple et Google ont su converger vers une solution commune en 2015 : utiliser le navigateur système comme point d'ancrage d'une session SSO. C'est maintenant une bonne pratique officielle matérialisée par la Best Current Practice[4] « OAuth2 for native applications »

L'AUTHENTIFICATION CONTEXTUELLE, OU COMMENT ADAPTER LE NIVEAU D'ACCÈS À UNE DONNÉE EN FONCTION DE LA CRITICITÉ DE CELLE-CI

Un des nombreux enjeux concernant l'authentification est de simplifier au maximum l'accès des utilisateurs à leurs données tout en garantissant un niveau de sécurité satisfaisant. L'authentification contextuelle permet de répondre à cet enjeu, en adaptant le niveau d'accès à la nature de la transaction, à ses caractéristiques, aux habitudes utilisateurs, à son contexte…

Dans le cadre d'une application mobile bancaire, par exemple, cela permet à l'utilisateur de consulter son compte en banque, de visualiser les soldes de ses comptes sans avoir à se réauthentifier à chaque accès. L'application requerra toutefois une authentification au moment de réaliser une opération sensible (un virement interne par exemple), et une authentification forte au moment de réaliser une opération très sensible (ajout d'un bénéficiaire par exemple).

Les solutions du marché proposent aujourd'hui des solutions pensées selon une logique où le client applicatif est responsable d'initier la demande de jeton en précisant le niveau d'authentification requis. Mais le vrai besoin est en réalité de définir et appliquer ces politiques d'accès aux données en un point unique, au sein du serveur d'autorisation. Il s'agit notamment d'un besoin essentiel lorsque l'on veut appliquer une authentification liée au niveau de risque du contexte (géolocalisation, terminal connu ou non, habitudes de transactions, etc.). Et il faut reconnaitre qu'aujourd'hui les solutions disponibles sur le marché n'offrent pas encore toute la souplesse requise.

PROPAGATION DE L'IDENTITÉ, OU COMMENT TRANSMETTRE UN JETON D'ACCÈS ENTRE DEUX APPLICATIONS (OU PLUS) ?

Il est de plus en plus courant qu'un appel vers une API déclenche une cascade d'appels vers d'autres API, notamment dans le cadre d'une architecture de type micro-service. La transmission de l'identité de l'utilisateur doit alors être assurée tout en restant sécurisée :

- Transmettre un seul et même jeton tout le long de la chaîne est très risqué : le jeton a trop de droits et son détournement est possible à chaque niveau de la chaîne
- Ne vérifier l'identité de l'utilisateur qu'en début de chaîne puis n'authentifier que les services ensuite en transmettant l'identité est également risqué : un service compromis peut usurper l'identité de n'importe quel utilisateur

4- *https://tools.ietf.org/html/draft-ietf-oauth-native-apps*

Par ailleurs, les droits (ie. scopes) contenus dans le jeton initial ne correspondent peut-être pas aux droits nécessaires à chaque niveau de la chaîne d'appels de services.

C'est pour répondre à ce cas d'usage, et à celui de la traçabilité de l'impersonnation, qu'un nouveau grant type est actuellement proposé : Token Exchange[5].

Chaque appelant intermédiaire peut échanger le jeton qu'il a reçu du service amont (qui contient l'identité de ce dernier et celle de l'utilisateur) contre un jeton qui pourra être transmis à une service aval (avec toujours l'identité de l'utilisateur, les identités des services par lesquels la chaîne transite ainsi que les droits nécessaires à l'appel de ce service).

Un apport majeur de cette nouvelle cinématique est qu'elle permet de centraliser la politique d'appels entre micro-services ainsi que l'application de cette même politique et donc d'assurer la traçabilité des appels.

PROTECTION CONTRE LE VOL DE JETON, OU COMMENT SE PRÉMUNIR DU VOL D'UN OU D'UNE BASE DE JETONS ?

Dès la conception du protocole OAuth2, le jeton proposé est considéré comme suffisant pour accéder à une ressource. Son vol est donc une menace permanente dont il convient de se protéger.

Deux approches sont alors possibles :

- **Tenter** d'empêcher ce vol (dans un jeu aussi appelé « le chat et la souris »)
- **Rendre** ce jeton nécessaire mais non suffisant pour accéder à une API

Cette seconde approche, spécifiée dans le draft « OAuth2 Token Binding[6] », demande au client applicatif de s'authentifier à l'aide d'une paire de clés cryptographiques lors de la génération du jeton et d'utiliser cette même paire de clés lors de l'utilisation du jeton. Jeton et paire de clés sont liés et un jeton volé sans la clé privée du client est alors inutilisable.

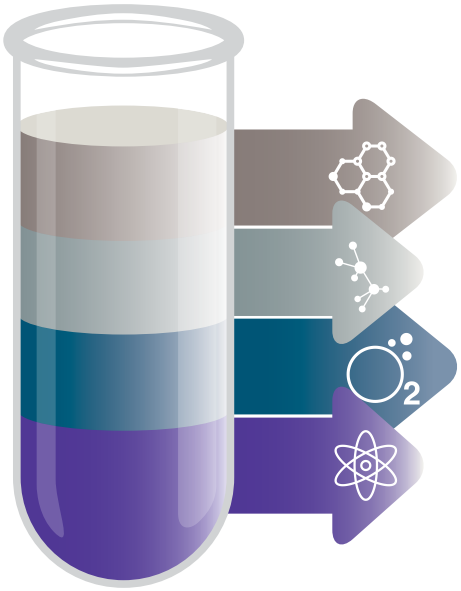
ÉCRIRE ET PARTAGER LA RECETTE

Dernier ingrédient de la recette, il convient de décliner une architecture de référence de OAuth pour l'adapter au contexte du SI de l'entreprise. Pour cela, il faut définir le cadre d'utilisation des API en :

- **Définissant et partageant les règles de sécurité :** Les cinématiques autorisées et le cadre d'application, les checklists sécurité et l'architecture de référence doivent être formalisées.
- **Formant et outillant les développeurs :** Des sessions de formation et de présentations des principes adoptés doivent être organisées. Les équipes projets peuvent être rendues autonomes dans leur intégration au reste du SI.
- **Intégrant les ressources sécurité dans les sprints agiles :** Les ressources agissant en tant que coach sécurité doivent être identifiées pour accompagner la conception applicative et apporter des solutions prêtes à l'emploi et être un accélérateur.

Au final, comme la recette du bon miel, sécuriser des APIs nécessite une succession d'ingrédients allant du plus basique jusqu'au plus élaboré tout en tenant compte du besoin et du contexte. Et surtout un travail en commun, dans un objectif partagé !

La recette pour des APIs sécurisées



ÉCRIRE LA RECETTE

Une architecture de référence et un cadre d'application

LIMITEZ LES ADDITIFS

Se poser la question des besoins réels vs standard

UNE PINCÉE D'OAUTH

Sans tomber dans les pièges du standard

UNE BASE DE SECURITY AS USUAL

Une API est une application web

WAVESTONE

www.wavestone.com

Wavestone est un cabinet de conseil, issu du rapprochement de Solucom et des activités européennes de Kurt Salmon (hors consulting dans les secteurs retail & consumer goods). Il figure parmi les leaders indépendants du conseil en Europe.

La mission de Wavestone est d'éclairer et guider ses clients dans leurs décisions les plus stratégiques en s'appuyant sur une triple expertise fonctionnelle, sectorielle et technologique.

5- *https://tools.ietf.org/html/draft-ietf-oauth-token-exchange*
6- *https://tools.ietf.org/html/draft-ietf-oauth-token-binding*