

Sécurité Web: Méthodologie de sécurité pour intégrer un site web avec services web REST.

Oumar BA (oumar.ba@aims-senegal.org)
L'Institut Africain des Sciences Mathématiques (AIMS)
Sénégal

Supervisé par: Title Firstname Lastname
Institute of Supervisor, Country

19 Mai 2016

Présenté pour Réalisation Partielle d'un Master II à AIMS



AIMS

**African Institute for
Mathematical Sciences
SENEGAL**

Abstract

A short, abstracted description of your essay goes here. It should be about 100 words long. But write it last.

An abstract is not a summary of your essay : it's an abstraction of that. It tells the readers why they should be interested in your essay but summarises all they need to know if they read no further.

The writing style used in an abstract is like the style used in the rest of your essay : concise, clear and direct. In the rest of the essay, however, you will introduce and use technical terms. In the abstract you should avoid them in order to make the result comprehensible to all.

You may like to repeat the abstract in your mother tongue.

Declaration

I, the undersigned, hereby declare that the work contained in this essay is my original work, and that any work done by others or by myself previously has been acknowledged and referenced accordingly.

Firstname Middlename Lastname, 15 May 2014

Table des matières

Abstract	i
1 Introduction	1
1.1 Background	1
1.2 Définitions des termes et concepts	1
1.3 Problématique	3
1.4 Méthodologie	4
1.5 Structure	5
2 Les services web REST	6
2.1 Présentation des services web rest	6
2.2 Spécifications de l'architecture	8
3 The Standard Security Method in RESTful Web Services	14
3.1 Overview of OAuth 2.0	14
3.2 OAuth 2.0 Core	15
4 Twitter security methods	19
4.1 Résumé de la méthode de sécurité twitter : 3-Legged Oauth 2.0	19
4.2 Authentication & Authorization in Twitter API calls	23
5 The Title of Your First Appendix	27
5.1 section title	27
References	27

1. Introduction

1.1 Background

Selon le rapport de l'enquête sur la criminalité et la sécurité de CSI Computer en 2010 et 2011, 92 pourcent des entreprises ont souffert d'une attaque d'application réussie entre Juillet 2009 et Juin 2010. Pire encore selon William Noonan [Richardson, 2010/2011], la fréquence et la complexité des crimes cybernétiques ont augmenté de manière significative. En plus, 69 pourcent des répondants américains ont souligné que l'impact des cyber-menaces à leur entreprise a attiré leur attention (contre 49 pour cent l'année précédente). La situation ci-dessus illustre l'importance de la sécurité dans les affaires en particulier dans les serveurs Web. En outre, d'autres produits logiciels sont construits sur le web infrastructure que l'application native. Surtout, une nouvelle page pour l'application Web, le développement a été tourné quand Roy Thomas Fielding a présenté la nouvelle architecture Web REST nommée (Representational State Transfer) dans le chapitre 5 de son doctorat intitulée "Architectural Styles and the Design of Network-based Software Architectures". Ceci est la raison de la croissance importante des API [Hughes Systique Coporation, 2006]. La nouvelle architecture peut favoriser un meilleur environnement pour le développement d'un site Web, cependant, elle apporte aussi de nouveaux défis en matière de sécurité qui doit être adapté dans la nouvelle infrastructure. Ainsi, ce mémoire de master explore comment marche les méthodes de sécurité des services web rest avec les sites web. Ce mémoire a pour but d'examiner les deux domaines suivantes :

- Comment les serveurs de Twitter et Stripe authentifient les requêtes provenant de divers applications clientes.
- Comment les sites web exploitent les API afin de améliorer l'environnement web pour le e-commerce.

Sur la base des deux questions mentionnées ci-dessus, cette recherche est divisé en deux grandes parties. La première partie (chapitres 1- 5) centré sur la première champ. La deuxième partie (chapitre 6) fournit la solution et répond à la deuxième champ en utilisant un site Web et un framework Python conçus par l'auteur, qui peut aider les utilisateurs à faire des appels API d'une manière simple et maintenir l'intégrité des services. Avant de passer au chapitre suivant, il est nécessaire de définir les termes de base et concepts utilisés dans la thèse.

1.2 Définitions des termes et concepts

- Site web intégré : le concept de base des sites web intégrés est de fournir plusieurs outils et services qui utilisent différentes technologies et qui sont fournis par plusieurs services web. En utilisant des sites web intégrés du coté client, les utilisateurs sont capables d'interagir, de coopérer et de consumer plusieurs services web hébergés dans différents serveurs.
- API : (Application program interface) est un ensemble d'outils, de fonctions et de protocoles dont les utilisateurs peuvent se référer et les intégrer dans le développement de leur applications.
- Sécurité web : Cela implique comment définir des principes pour protéger les services web et les clients web. Mais aussi comment déterminer les vulnérabilités du système dans tous les étapes du processus tel que le développement, l'analyse des besoins, la conception et la maintenance des applications. Selon la CIA triad, il y a trois principales composantes qui sont cruciaux pour la sécurité web :
 - Intégrité : C'est le principe qui stipule que les données doivent être correct. Certains accès

non authentifié pourrait causer des modifications inattendues des données, par exemple, les données d'utilisateur été changé au cours de la transaction par un tiers [Cryptome, 2013].

- Confidentialité : Les données d'un utilisateur ont été espionnées à la fois du côté serveur et côté client. Cela pourrait conduire à la perte de la vie privée de l'utilisateur. Ainsi, chaque accès devrait être attentivement autorisé [Hall, 2005]. Dans ce cas, les données provenant de faux utilisateurs peuvent passer par tous les processus d'authentification et deviennent valides. Par conséquent, les faux utilisateurs peuvent envoyer de demandes erronées au nom des victimes. Ceci est dangereux tant pour les services et la utilisateurs ; Par conséquent, cette thèse analyse également certains processus d'authentification qui sont utilisé par les fournisseurs de l'API.
- La disponibilité : Ceci fait référence à l'accessibilité des données. Cela signifie que l'information doit être disponible quand il est nécessaire. Par conséquent, les mécanismes d'authentification, les voies d'accès et le matériel doivent fonctionner correctement. Cela fonctionne contre les actions malveillantes telles que le déni de service. Les serveurs web peuvent être attaqués de différentes manières comme les attaques DoS, le bourrage de mémoire etc ... Ce genre de menace peut causer des lenteurs inattendu durant les transferts de donnée entre le client et serveur, ou pire cela pourrait endommagé le serveur.

Présentement il existe plusieurs méthodes de sécurité. Cependant, toutes ces méthodes peuvent être classées dans une ou plusieurs des catégories suivantes.

- Domain Based Security Model (DBSy) : Il a été crée vers la fin des années 90 par Defense Evaluation and Research Agency (DERA). C'est un ensemble de notations et de techniques développés spécialement pour le gouvernement britannique par par QinetiQ. DBSy explique et évalue les informations sécuritaires axées sur l'entreprise, requises par l'architecture réseau. Dans ce modèle, le navigateur contient la liste des domaines authentifiées qui ont certains droits par rapport au serveur.
- Certificate based security (Certificatte-based Authentication) : L'application est vérifié en soumettant un certificat valide à travers une application tierce. Dans ce cas, le certificat digital agit comme une carte d'identité officielle qui décide quels droit l'application possède. De même l'application pourrait générer des certificats qui doivent suivre suivre certaines règles et restrictions. Ce genre de certificat se introduit dans la chapitre 5. Ce modèle est très populaire et largement utilisé dans plusieurs serveurs web. En raison de sa popularité, ce mémoire va se concentrer sur le Certificate based security comme point de départ pour déceler les problèmes actuels et développer ma solution pratique. En plus, il y a deux types de sécurité basé sur les certificat. Ce model utilise l'un ou les deux :
 - Perimeter Security : Ce type de méthode de sécurité est le pré-signé application. Une fois installé, l'application a automatiquement le droit de accès API dont le certificat a été authentifié
 - Least Privilege : Ceci est une version limitée dans le temps de la sécurité basée sur le certificat. Selon le certificat que l'auteur a signé avant la validation, l'accès de l'application à API pourrait être résilié [Langford, 2003]. Actuellement, la durée générale d'un seul certificat est de deux heures. Ça veut dire qu'au bout de deux heures, l'application doit se reconnecter avec le nouveau certificat valide pour continuer d'accéder à l'API. En outre, l'auteur de l'application peut demander une certification à long terme : si la demande est signé avec cette clé, il a le droit d'accéder à ces API pour plus de 2 heures (selon chaque fournisseur API). Par exemple, Google et Facebook permettent à l'application d'accéder long terme à leurs API pendant 3 mois.
- Prompting based security : Ce type de modèle de sécurité exige que l'application passe par

le processus d'authentification en entrant ses informations d'identification. Après avoir obtenu l'autorisation, selon le fournisseur de l'API, cette application peut se voir autoriser l'accès à vie car il est authentifié ou il devra s'authentifier chaque fois que l'application accède à une source [Allott et al., 2008]. Ce modèle correspond bien avec l'accès très haute ou très basse fréquence. Dans le cas, si l'application accède, envoie et reçoit des données en continu à partir d'un serveur pour une longue période, alors, l'API Fournisseur devrait accorder un accès à long terme à cette application. Par exemple, certains rapports et d'analyse API permettent aux clients d'envoyer des demandes fréquemment pour obtenir les dernières données. Sinon l'accès unique pourrait être un excellent choix pour maintenir le haut intégrité du système, car cela permettrait d'éviter une modification inattendue provenant d'accès non authentifié.

- No security : Habituellement, l'application ne pourrait pas fonctionner efficacement sans haute processus sécurité, tout comme le résident ne pourrait rien faire légalement sans aucune identification officielle. Cependant, il existe certaines applications qui ne limitent pas l'accès ou ont pas de sécurité explicites. Ils passent ce fardeau à l'autorité d'un tiers, par exemple, l'application anti-virus ou même le site Web intégré si elle est assez simple.

1.3 Problématique

Comme mentionné dans la section précédente, les menaces de sécurité peuvent causer de nombreux problèmes pour à la fois le fournisseur de services et le client, comme la perte de la vie privée et des données ou serveur web endommagé. Spécialement dans les sites web intégrés, la confidentialité joue un rôle essentiel car le client envoie et reçoit des informations sensibles depuis le site client vers divers serveurs. Par exemple, Flickr, un site web de partage de photo, autorise le client à s'authentifier et ensuite d'utiliser leur services avec leur compte Yahoo. En utilisant une application tierce, Flickr non seulement simplifie les processus de leur site mais aussi augmente l'efficacité de leurs services. Ainsi les utilisateurs n'ont pas besoin de s'enregistrer pour avoir un compte puisqu'ils peuvent utiliser des comptes déjà existants. En plus, cela a permis à Flickr de diminuer la taille de leur base de données puisque des milliers voire des millions d'utilisateurs ont pu être supprimés et fournis par des applications tierces. Utiliser les comptes des applications tierces pour authentifier les utilisateurs est appelé Social Login. Ce modèle devient de plus en plus populaire dernièrement. Un autre cas est Ebay. Après avoir fini de sélectionner des articles, l'utilisateur est capable de vérifier ces choix et d'effectuer cette transaction avec Ebay en utilisant une application tierce tel que PayPal. Cela apporte beaucoup d'avantage tant pour l'utilisateur, Ebay que pour les fournisseurs de services ; Néanmoins cette méthode cause aussi beaucoup de challenge en terme de sécurité, spécialement de confidentialité. Parce que Ebay n'est pas en mesure de contrôler le processus de protection du titulaire de la carte bancaire. Ce mémoire compte analyser et étudier ces problèmes en répondant aux questions suivantes : - Comment l'utilisateur peut utiliser le Social Login pour se connecter en toute sécurité au site web ? Quelles informations de l'utilisateur sera échangé entre l'application tierce et le site web ? - Comment l'utilisateur peut s'authentifier pour envoyer des requêtes à différents serveurs web. - Comment des détails personnels comme les comptes bancaires, les numéros de carte peuvent-ils être protégés lors du paiement en ligne ?

En outre, après avoir traversé le processus de sécurité des fournisseurs d'API, j'ai découvert certaines lacunes que je suis en mesure d'améliorer. Toutes les processus de recherche de sécurité sont efficaces et fiables ; cependant, ils semblent être trop compliqués et difficiles pour les consommateurs. Certains processus d'authentification actuels obligent les utilisateurs à avoir des connaissances en programmation. Par conséquent, passer par ces processus sans problèmes n'est pas facile pour tous les utilisateurs, en particulier pour ceux qui ne possèdent pas beaucoup d'expérience dans la programmation. Par exemple,

le résultat retourné par chaque API peut être soit en JSON soit en XML. Ainsi, l'utilisateur peut lire toutes les données mais s'il veut récupérer ces valeurs automatiquement pour n'importe quelle raison, par exemple les utiliser comme des données d'entrée pour un autre API ou les afficher automatiquement sur un site, alors cet utilisateurs aura besoin de connaissances en JSON and XML. En outre, chaque appel d'API nécessite différents arguments et l'ordre de ces arguments est important. Si les utilisateurs font une petite erreur dans l'ordre ou faute d'orthographe dans la valeur de l'argument, leur demande ne peut pas passer par le processus d'authentification du serveur Web. De plus, les erreurs renvoyées par les services Web sont assez ambigus. Par exemple, Twitter renvoie le code d'erreur 400 (Bad Request) et leur explication pour ce cas est "La demande était invalide ou ne peut pas être servi autrement. Un message d'erreur d'accompagnement sera expliquer plus en détail. Dans API v1.1, les demandes sans authentification sont considérés comme non valides." Cependant, il y a mille cas qui pourraient causer cette erreur. Ainsi, les clients ont besoin de vérifier a plusieurs reprises et ligne par ligne leur requêtes pour trouver la source de l'erreur, peut être dans certains cas, le problème est une virgule manquantes. Ainsi ma solution proposé dans ce mémoire permettra de surmonter ces erreurs ambigus.

1.4 Méthodologie

Fondamentalement, cette recherche se concentrera sur les méthodes de sécurité dans les sites Web intégrés et d'analyser le flux de travail de Twitter et Stripe. Ainsi, dans cette thèse les limites sont signalées et mes propres solutions seront proposées. Après des recherches sur certaines méthodologies, la Design Science Research dans les systèmes d'information semble être le choix parfait pour cette thèse, car il favorise un meilleur environnement pour résoudre les questions techniques. Le paradigme de conception scientifique est originaire de l'ingénierie et les sciences de l'artificiel. Le point clé de cette méthodologie de recherche est d'encourager et motiver les créations et innovations, qui suggèrent les idées, des solutions techniques des produits à travers toutes les étapes du processus de développement : de l'analyse, la conception, la mise en oeuvre, les tests à la gestion, en vue de surmonter tous les lacunes et problèmes. Ces problèmes sont variés et peuvent exister sous diverses formes telles que dans le modèle de développement de logiciels, la logique formelle, rigoureuse mathématiques, algorithme de programmation et beaucoup plus. Les lignes directrices de conception de recherche scientifique suivante illustrent clairement les principes pour la mise en oeuvre de la méthodologie employée.

FIGURE 1.1 – Méthodologie : Design Science Research

guidelines	description
Guideline 1 : Conception de l'artefact	La conception doit répondre efficacement aux problèmes d'organisation et des solutions viables.
Guidelines 2 : Pertinence du problème	L'objectif du design-science research est de créer, proposer et promouvoir la technologie de base des solutions qui pourront résoudre les problèmes existants. Ceci peut être atteint soit en changeant le processus soit en changeant les causes du problème.
Guidelines 3 : Conception de l'évaluation	La qualité de la conception doit être bien qualifié et rigoureusement appliquée par des approches réalistes et fiables. Une bonne conception devrait être observationnelle (analyse intensive organisationnelle la structure, plusieurs projets), analytique (prendre beaucoup les aspects en considération, tels que la complexité du problème, la potentialité de la solution pour l'organisation), expérimentale (grande facilité d'utilisation), Test (couverture élevée de domaine) et Descriptif (la conception doit être fiable par le développement de la théorie existante et la recherche).
Guidelines 4 : Contribution de la recherche	Une conception efficiente doit prouver que le résultat est applicable et réalisable pour résoudre le problème ou au moins apporter une contribution dans ce domaine
Guidelines 5 : Rigueur de recherche	La recherche et ses résultats doivent suivre strictement la nature du problème. L'objectif final est de construire une solution adéquate au problème
Guidelines 6 : Conception du processus de recherche	Les méthodologies utilisées doivent satisfaire toutes les demandes et restrictions de l'environnement

1.5 Structure

Ce mémoire traite de la sécurité web dans les sites web intégrés, en particulier les transactions avec API. Il est divisé en deux grandes parties. Le premier contient 5 chapitres.

Le chapitre 1 donne une brève introduction, un résumé des concepts clé de ce mémoire et quelques définitions qui sont nécessaire pour les 4 prochaines chapitres. Le chapitre 2 résume les services web rest et ces infrastructures sécuritaires. Le chapitre 3 introduit la norme standard de sécurité Oauth 2.0. Les chapitres 4 et 5 analysent les méthodes de sécurité utilisé par le réseau social Twitter et un service de paiement en ligne Stripe. Le chapitre 6 apporte ma propre solution en créant un site web et une api python. Le dernier chapitre conclut ce mémoire et inclus les limites de ma solution de même que les perspectives de ce mémoire.

2. Les services web REST

Retour à l'époque où la définition de REST a été introduite par Roy Fielding dans sa thèse universitaire «Architectural Styles et la conception d'architectures logicielles basées sur le réseau» en 2000 à l'Université de Californie, cette théorie n'a pas reçu beaucoup d'attention, cependant, pour le moment REST est l'un des termes les plus populaires et joue un rôle principal dans le développement Web [ProgrammableWeb, 2012]. Il est devenu le fondement de la croissance des API. La technologie REST est apparue de plus en plus dans les API existantes sur le marché [Slideshare, 2013].

Selon le Centre de recherche Web programmable [Slideshare, 2013], de 2005 à 2013, les API Web ont connu une croissance impressionnante qui a été plus forte que jamais (d'une seule API en 2005 à 10302 API en octobre 2013) comme on le voit sur Le graphique de la figure suivante.

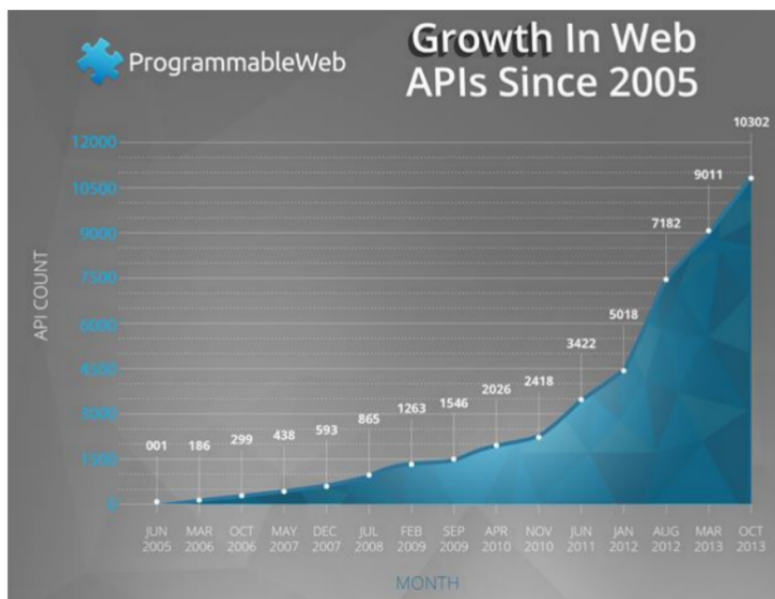


FIGURE 2.1 – API growth

La question ici est de s'avoir ce qui aurait pu causer cette croissance intéressante ? Le graphique suivant, figure 2, pourrait répondre en partie à cette question. Comme cela a été indiqué dans le sondage qui a été publié le 11 Décembre 2012 par le ProgrammableWeb [2012], plus de deux tiers des API statistiquement sélectionnés ont été développés par le cadre REST. À ce stade, l'impact de REST sur le développement web a été clairement établi.

Les sections suivantes vous présenteront plus en détail le concept de service Web RESTful et son flux de travail.

2.1 Présentation des services web rest

REST signifie Representational State Transfer. Dans l'ensemble, la discipline de REST comprend l'apatri- dité client-serveur, le protocole cacheable de communication. Selon la définition d'IBM [2015], "REST définit un ensemble de principes architecturaux par lesquels vous pouvez concevoir des services Web

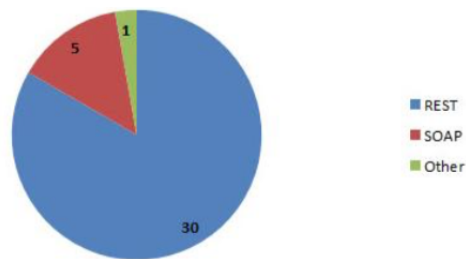


FIGURE 2.2 – API growth

axés sur les ressources d'un système, y compris la façon dont les états des ressources sont adressés et transférés sur HTTP par un large éventail de clients écrits sous différentes langages". C'est aussi la caractéristique la plus remarquable du service Web RESTful, qui est l'axé ressources. Cette fonctionnalité simplifie considérablement la complexité de l'architecture Web en évitant d'utiliser des mécanismes complexes comme CORBA, RPC ou SOAP pour communiquer entre le client et le serveur [Fielding, 2000]. Les méthodes HTTP simples avec quelques ajustements ont remplacé ces algorithmes pour améliorer les services comme on peut le voir sur la Figure 3 ci-dessous.

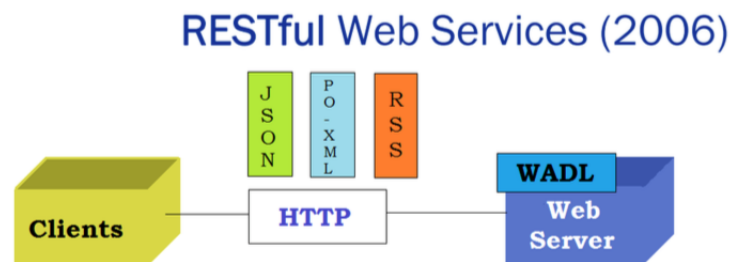


FIGURE 2.3 – API rest 2006

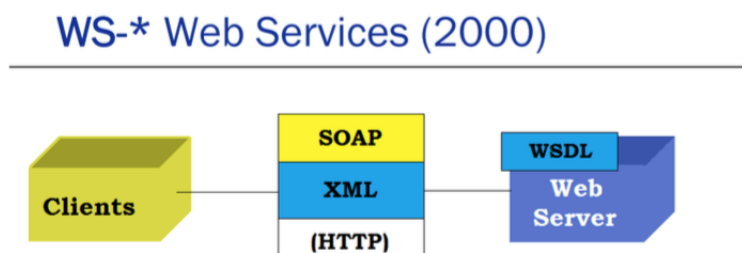


FIGURE 2.4 – API rest 2000

La section suivante analysera plus en profondeur ce qui pourrait rendre le service Web RESTful plus puissant que les autres et comment le style REST apporte de la clarté sur la façon de créer des applications Web hautement évolutives.

2.2 Spécifications de l'architecture

2.2.1 Les méthodes HTTP. Il existe quatre méthodes HTTP principales dans la version 1.1 nommées GET, PUT, POST et DELETE (il existe évidemment aussi d'autres méthodes telles que HEAD, CONNECT et TRACE, mais cette recherche ne se concentrera pas sur eux car ces méthodes ne jouent pas un grand rôle et ne sont pas appliquées par de nombreux fournisseurs d'API). Dans le serveur Web RESTful, chaque méthode est utilisée pour une interaction et une modification spécifiques de la ressource :

- GET : Cette méthode extrait les données du serveur Web, recherche les ressources demandées et collecte les données en tant que requête. Cette méthode n'a pas le droit de modifier la ressource.
- POST : La méthode POST vise à créer une nouvelle ressource dans le serveur. En raison de cette caractéristique, la méthode POST est nécessaire pour contenir le corps qui clarifie toutes les valeurs nécessaires pour cette création. Comme la méthode GET, cette méthode ne peut pas non plus modifier les ressources existantes.
- PUT : La méthode PUT permet de modifier l'état ou la valeur de mise à jour des ressources. Différent de GET et POST, cette méthode est capable de modifier les ressources existantes. En conséquence, la méthode PUT doit contenir la nouvelle valeur afin de modifier les ressources.
- DELETE : Cette méthode enlève ou supprime les ressources. Dans certains cas, les fournisseurs d'API ne prennent pas en charge ce type de méthode. Ils ne permettent pas à l'utilisateur de supprimer les ressources et ces derniers sont simplement déplacées vers un dossier distinct ou changé le statut à supprimé. Dans ces cas, les méthodes DELETE et PUT sont assez similaires.

En utilisant des méthodes séparées à des fins spécifiques, il améliore non seulement la clarté de la requête mais réduit également la charge de travail pour le serveur. Jetons un coup d'oeil à quelques méthodes HTTP dans RESTful pour voir comment ils sont efficaces et simples.

Par exemple, il existe une ressource nommée Personnes dans le serveur. Dans le cas où le client souhaite récupérer les informations d'une personne de nom Peter dans la ressource People, la requête HTTP GET doit être structurée comme suit :

```
GET http://example.com/v1/people/user_info.json?name=peter&count=1
```

L'URL de la requête devrait être lisible et les arguments sont séparés par le caractère "&". Dans ce cas, la requête demande à recevoir une personne de nom Peter et le résultat retourné doit être en style JSON et ne contient qu'une seule valeur s'il ya beaucoup de personnes ayant le même nom Peter. Par la méthode HTTP, le serveur Web est capable de comprendre que la requête est de collecter des informations sur Peter et non de créer de nouvelles personnes de nom Peter dans la ressource.

2.2.2 Les URI ressources orientés REST. Cette fonctionnalité décrit plus clairement l'aspect ressources orientées dans le service Web RESTful qui est également la fonction la plus remarquable de ce modèle. L'URI indiquera intuitivement avec quelle ressource la requête va interagir. En lisant l'URI, les développeurs et les utilisateurs sont capables de prendre conscience de ce que la requête va modifier dans la ressource. Le style REST encourage à avoir un URI distinct pour chaque donnée, comme une photo ou une entrée dans une base de données. Examinons l'URI suivant pour étudier plus en profondeur la structure de l'URI :

```
POST http://www.example.com/bookstore/books/name=exambook&ISBN=ISBN123
```

Comme mentionné dans la section 2.2.1, par la méthode HTTP, cette requête demande de créer un nouveau livre avec le titre "exambook" et son ISBN est ISBN123. Les utilisateurs et les développeurs

peuvent comprendre cette demande en lisant l'URI, cependant, comment le serveur Web pourrait comprendre la demande et exécuter sa commande. La réponse est la structure de l'URI. Fondamentalement, un URI complet contient trois parties principales qui définissent explicitement les objectifs de cette demande :

- L'hôte : Il s'agit de l'adresse du serveur Web. Dans ce cas, l'hôte est "www.example.com". Lorsque vous recevez cette demande, le serveur vérifiera d'abord la validité de l'hôte de cette demande pour s'assurer que la requête a été envoyée à l'adresse correcte.
- La ressource : Nom de la ressource interactive. Après avoir vérifié l'hôte, le serveur Web détectera et approuvera le nom de la ressource. Le nom de ressource de l'URI doit suivre la structure de la ressource. Par exemple, il existe une ressource appelée «books» dans la ressource «bookstore» et le nom de la ressource sur l'URI doit être bookstore/books. Deux valeurs doivent être séparées par "/".
- La valeur : Ceci est une clarification pour la demande. Cette partie complètera les exigences pour restreindre la portée de l'interaction ou spécifier plus de détails pour l'action de cette requête. Dans l'URI ci-dessus, "name = exambook&ISBN=ISBN123" joue le rôle de valeur de la requête. Il clarifie que la requête est de demander à créer un nouveau livre avec le nom "exambook" et ISBN123 comme ISBN.

En appliquant cette structure claire, l'URI REST est non seulement intuitif à lire, comprendre et ressources orientées (c'est-à-dire, chaque requête doit spécifier la ressource sur laquelle il souhaite agir) mais simplifie également la requête. Comparons la complexité de la demande entre les services Web à l'aide de SOAP et RESTful Web Service [M. Elkstein, 2008].

```
<?xml version="1.0"?>
  <soap:Envelope
    xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
    soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
    <soap:body pb="http://www.acme.com/phonebook">
      <pb:GetUserDetails>
        <pb:UserID>12345</pb:UserID>
      </pb:GetUserDetails>
    </soap:Body>
  </soap:Envelope>
```

L'extrait de code précédant illustre une requête utilisant l'enveloppe SOAP. Dans cette demande, les utilisateurs sont tenus de créer une enveloppe SOAP avec un corps à l'intérieur. Dans le corps, l'action de cette requête doit être soulignée en utilisant la structure XML. En outre, toutes les informations supplémentaires pour cette action doit être inséré à l'intérieur du corps. Cette structure va pousser le fardeau sur le côté client et côté serveur puisque les utilisateurs ont besoin de créer un formulaire compliqué pour une demande simple et le serveur doit gérer un XML complexe pour récupérer les informations nécessaires. Dans le même but, dans RESTful, la demande sera courte, simple et précise. Il peut être interprété comme suit :

```
GET http://www.acme.com/phonebook/UserDetails/12345
```

Une seule ligne peut représenter pour tout le fichier XML complexe et il n'y a pas de corps de requête. Il s'agit simplement d'une URL, envoyée par la méthode HTTP GET. En raison de la structure simple, orientée ressources et convivial, REST devient l'infrastructure d'API de plus en plus utilisée puisque la documentation est beaucoup plus facile à comprendre et à mettre en œuvre.

2.2.3 Représentation. Comme expliqué ci-dessus à la figure 3, la différence entre les services Web REST et les services Web en 2000 est que le REST utilise la méthode HTTP combinant certains formats lisibles standard tels que JSON ou XML. Le JSON est également utilisé par la ressource qui est déclenchée dans la requête et il reflète l'état actuel et les propriétés de la ressource. Ce JSON ou XML est aussi la réponse de la requête et inclus dans le corps HTTP.

Voici l'exemple retourné JSON pour la demande d'obtenir les informations d'utilisateur Facebook comme ci-dessous :

```
GET /v2.5/{user-id} HTTP/1.1
Host: graph.facebook.com
{
  "id": "14564232499685",
  "first_name": "Oumar",
  "gender": "male",
  "last_name": "Ba",
  "link": "https://www.facebook.com/app_scoped_user_id/14564232499685/",
  "locale": "fr_SN",
  "name": "Oumar Ba",
  "timezone": 0,
  "updated_time": "2016-09-17T06:58:11+0000",
  "verified": true,
}
```

Ce résultat n'est pas une ressource réelle. C'est juste un fichier JSON contenant tous les attributs nécessaires pour cette ressource, dans ce cas, les propriétés retournées sont l'identifiant Facebook, le prénom, le sexe et ainsi de suite. Pour certaines propriétés, telles que Images et vidéos, le lien sera inclus dans le fichier JSON et les clients pourront y accéder et les télécharger ultérieurement. Cela pourrait aider le serveur à éviter les transactions encombrantes car seule le texte simple et lisible est envoyé . Cela pourrait également conduire à réduire la dépendance sur l'infrastructure des plates-formes et des périphériques, ainsi, ces services peuvent être utilisés sur différents systèmes en dépit des langages de programmation.

2.2.4 Stateless. La communication dans RESTful Web Service doit être apatride. Cela signifie que chaque requête du client côté client doit contenir toutes les informations essentielles pour analyser et exécuter cette requête. Cela pourrait conduire à réduire le stockage dans le serveur, car il n'est pas nécessaire d'enregistrer ou de stocker le contexte de ces requêtes dans le côté serveur. Par conséquent, l'état de session est conservé totalement du côté du client. L'une des conditions pour valider la demande est la structure d'URL qui a été introduite ci-dessus. Habituellement, une demande doit contenir trois parties principales, d'ailleurs, il est possible que le fournisseur de l'API ait quelques arguments supplémentaires obligatoires. Par exemple, l'API AdWords de Google exige également que ses clients incluent la version de l'API dans la requête. Les demandes ne peuvent pas passer par le processus d'authentification de Google s'il ya un argument manquant.

Encore une fois, cette contrainte maintiendra la haute performance de REST. Stateless montre ses avantages en :

- Visibilité : Les demandes sont suffisamment claires pour que le serveur ne dépense pas plus d'efforts supplémentaires pour prendre un autre regard au-delà de la demande. Comme mentionné ci-dessus, dans la conception apatride, la demande doit être aussi détaillée que possible et en

raison de cette caractéristique, la visibilité de chaque demande s'améliore significativement.

- Fiabilité : Dans REST, l'échec partiel de la requête précédente peut être facilement récupéré sans duplication ou modification de l'état de la ressource. Prenons un exemple d'échec de la requête DELETE (Figure 4) [Envoisolutions, 2007]. Dans ce cas, le côté client envoie une requête de sup-

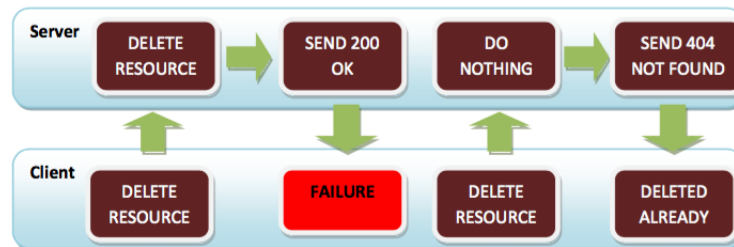


FIGURE 2.5 – API reliability

pression pour supprimer un objet de la ressource dans le serveur. Après avoir vérifié la demande, le serveur a accepté et exécuté la commande en tant qu'exigence et renvoyé la confirmation. Cependant, en raison d'une erreur de connexion, il y a un échec à ce stade et la confirmation ne peut pas être envoyée au client. En conséquence, le client supposera que la demande n'a pas été effectuée et la nouvelle demande sera envoyée à nouveau au serveur. À ce moment, le serveur ne prend aucune action puisque cet objet a été supprimé ainsi la confirmation non valide sera renvoyée au client. Ainsi, l'intégrité du système est maintenue aussi haut que possible puisqu'il n'y a pas de changement ou de duplicata dans la ressource.

- Evolution : C'est l'une des caractéristiques exceptionnelles du REST, le serveur n'est pas tenu de préparer l'espace pour le résultat de la demande puisque toutes les informations essentielles de résultat ont été renvoyées au client comme réponse à cette requête. Par conséquent, la demande doit déterminer le type de résultat renvoyé dans Content-Type à l'intérieur de chaque demande. Cela réduira la latence, le trafic réseau et le temps de charge coté serveur.

2.2.5 Fonctionnalité sécuritaire dans les services web REST. Jusqu'à présent, nous avons traversé de nombreux aspects des services Web RESTful. Dans cette section, une des caractéristiques les plus importantes de REST dans cette thèse, la sécurité dans REST, sera introduite. Le point clé de la sécurité est la validation. Qualifier toutes les entrées côté serveur pour protéger le serveur contre les attaques par injection SQL ou NoSQL. Ce type d'attaque exploitera la vulnérabilité de sécurité dans une application puis une requête pour obtenir des données et exécuter une requête SQL inattendue pour modifier la base de données. La prévention de cette attaque protège et maintient l'intégrité de la base de données. Plusieurs mécanismes sont disponibles pour sécuriser le service Web :

HTTP basic authentication : Dans ce modèle, les utilisateurs sont invités à s'authentifier par ID utilisateur et mot de passe pour chaque requête. La valeur de l'ID et du mot de passe doit être une chaîne opaque qui est illisible à l'œil nu. Le serveur ne répondra qu'aux demandes déjà validées. Le schema de base est illustré ci-dessous :

```
credentials = "Basic" basic-credentials
Authorization: Basic QWxhZGluOnNlc2FtZSBvdXZyZSB0b2k=
```

Dans l'exemple, l'ID d'utilisateur est "Aladdin" et le mot de passe est "sesame ouvre toi", cependant, ces chaînes ne sont pas transmises directement à la requête. La chaîne opaque, qui est le résultat du codage based64, est utilisée à la place [The Internet Society, 1999].

Authentification de l'accès de Digest : Différent de l'authentification de base de HTTP, l'authentification de l'accès de Digest fournit un non cryptage du contenu du message. Le point le plus intéressant de ce modèle est d'utiliser la valeur nonce. Cela signifie que chaque requête doit contenir une valeur unique. Que la demande soit exécutée ou non, valide ou non, authentifiée ou non, elle doit régénérer une nouvelle valeur unique pour chaque requête. Comme pour l'authentification de base, le serveur authentifie également l'en-tête WWW-Authenticate illustré ci-dessous :

```
credentials                = "Digest" digest-response
digest-response            = 1#( username | realm | nonce | digest-uri
    | response | [ algorithm ] | [cnonce] |
    [opaque] | [message-qop] |
    [nonce-count] | [auth-param] )
username                   = "username" "=" username-value
username-value             = quoted-string
digest-uri                 = "uri" "=" digest-uri-value
digest-uri-value           = request-uri ; As specified by HTTP/1.1
message-qop                = "qop" "=" qop-value
cnonce                    = "cnonce" "=" cnonce-value
cnonce-value              = nonce-value
nonce-count                = "nc" "=" nc-value
nc-value                   = 8LHEX
response                   = "response" "=" request-digest
request-digest             = <"> 32LHEX <">
```

Dans cet exemple, digest-response, une chaîne de 32 chiffres hexadécimaux calculée à partir des valeurs ci-dessus, prend la place mot de passe utilisateur. Le «qop» est représenté pour «qualité de protection» et il est pertinent pour nonce, ce qui protège le serveur contre les attaques de texte brut. Il s'agit également de l'amélioration de l'authentification de base HTTP. Dans le modèle précédent, l'attaquant peut spécifier un texte en clair puis le crypter ou le signer pour révéler les caractéristiques de l'algorithme de cryptage. Pour certains cas, en particulier dans la cryptographie à clé publique, ce modèle est très réalisable et les attaquants sont en mesure d'exploiter cette faille pour envoyer de nombreuses demandes d'authentification tant qu'ils connaissent la clé publique. Toutefois, la compréhension de cette lacune, Digest Access Authentication modèle s'attend à ce que les utilisateurs d'incluent une valeur unique dans la demande [The Internet Society, 1999].

Authentification coté client : Le côté client reçoit l'identité du côté du serveur. Dans ce modèle, le client est tenu de vérifier l'identité du serveur, cependant, l'opposé n'est pas obligatoire. Ainsi, contrairement aux modèles ci-dessus, l'authentification client exécute le processus d'authentification côté client et tous les messages entre le client et le serveur sont cryptés [Wagnon, 2013]. Comme on peut le voir sur la figure 5, après réception de la demande du client, le serveur envoie automatiquement le certificat deux fois. La première fois il informe le client qu'il doit préparer son certificat parce que le prochain message du serveur requiert de vérifier l'identité du client. Dans les prochaines étapes, le client envoie son certificat, les clés à crypter et la signature numérique pour s'authentifier. Après avoir parcouru le processus d'authentification, le client envoie une requête au serveur qui est cryptée par une clé symétrique. À ce moment, le serveur exécute la demande et renvoie la confirmation au client.

Authentification OAuth : Ce modèle devient de plus en plus populaire puisqu'il est appliqué dans la grande majorité des services Web RESTful. Ce modèle sera également analysé et approfondi dans le prochain chapitre.

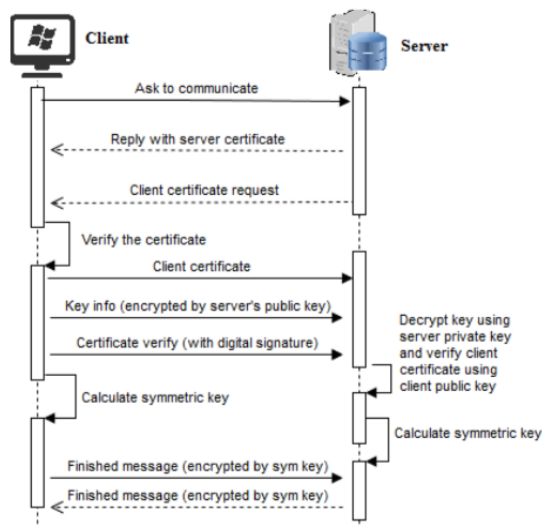


FIGURE 2.6 – API rest 2000

Jusqu'à présent, ce chapitre résumait la nature des services Web RESTful pour être témoin des différences par rapport aux services Web en 2000. La conception apatride (la caractéristique la plus extraordinaire) a affecté progressivement la façon dont la sécurité a changé dans le service Web RESTful. Les méthodes HTTP explicites empêchent les modifications accidentelles dans la ressource, l'URI aide à réduire l'étendue des ressources avec lesquelles on interagit. La représentation est de maintenir l'intégrité du système et de la ressource puisque les utilisateurs sont capables de télécharger l'objet seulement. Enfin, la conception apatride réduit les impacts négatifs des erreurs de ressources.

Dans le chapitre suivant, la méthode d'autorisation standard appelée OAuth 2.0 sera analysée. L'introduction et le concept principal d'OAuth seront également vérifiés au chapitre 3.

Ce modèle est l'infrastructure pour de nombreux API, donc, en étudiant ce modèle standard, l'idée de base d'une discussion plus approfondie est également brièvement résumée.

3. The Standard Security Method in RESTful Web Services

3.1 Overview of OAuth 2.0

Comme mentionné dans la section 2.3, il existe de nombreux modèles de sécurité qui sont utilisés dans le service Web REST, cependant, chaque fois que l'on considère le modèle le plus commun parmi d'entre eux, l'un des premier modèle qui vient à l'esprit est OAuth 2.0 car il est le plus commun modèle.

OAuth (Open Authorization) 2.0 est le protocole d'autorisation ouvert pour les applications natives et Web pour sécuriser leurs services en utilisant une méthode simple et standard. Il gère les demandes de ressources de diverses sources pour authentifier qui communique et son droit d'accès à la base de données qui est sauvegardée par une autre application. Prenons Flockler (comme lien <http://flockler.com/new-factory>) site Web comme un exemple, une application magazine social. Il recueille et affiche de nombreux postes de diverses ressources sociales telles que Twitter, Facebook et Instagram sur des sujets choisis ou à partir de comptes configurés. Cette application enverra des milliers de demandes au serveur OAuth et exigera de se vérifier pour accéder aux bases de données de ces serveurs sociaux. Après que le serveur OAuth a accepté leur accès, l'application a le droit de récupérer les données. Avant d'accéder au serveur de ressources, les clients doivent s'authentifier en envoyant une requête au serveur OAuth contenant les informations d'identification de l'utilisateur. Dans certains cas, ces informations d'identification sont la clé publique, la clé privée ou l'ID client. OAuth Server qualifie cette demande et renvoie un jeton valide représenté pour ce client et valable pour 2 heures habituellement. Ensuite, l'utilisateur doit joindre ce jeton dans l'en-tête de la requête et l'envoyer au serveur de ressources avec lequel il souhaite interagir. Le serveur de ressources communique avec le serveur OAuth pour vérifier le jeton après la réception de la requête. Si le jeton est vérifié avec succès par le serveur OAuth, le serveur de ressource exécute cette requête et renvoie une réponse réussie sinon une erreur est renvoyée au client.

Le point central du processus d'authentification OAuth est le jeton d'accès. Au lieu de compter sur le mot de passe de l'utilisateur comme clé principale pour chaque accès au serveur, OAuth génère un jeton basé sur les informations d'identification utilisateur. Ce jeton représente l'accès à une API unique au nom d'un client spécifique. Le diagramme suivant illustre le flux de travail de l'authentification OAuth en général [Hughes Systique Corporation, 2006] : En appliquant ce modèle, le serveur et le client pourraient réduire la charge de travail car ils n'ont pas besoin d'analyser et de dépenser beaucoup d'efforts sur la sécurité. Selon The Internet Society [1999], dans le modèle HTTP Basic et Digest Authentication, le serveur est responsable de vérifier toutes les requêtes. Dans le modèle d'authentification client, ce processus d'authentification a lieu principalement dans le client. Cependant dans le modèle d'OAuth, le client et le serveur pourraient prendre leurs mains outre de ce processus puisqu'il y a un tiers qui prend soin de lui. Cela permet non seulement de réduire la charge de travail pour le client et le serveur, mais également d'améliorer le niveau de sécurité de l'application. Les développeurs d'applications ne sont pas les spécialistes de la sécurité et, de leur point de vue, ils veulent être délestés du poids de protéger les données sensibles. L'objectif final des développeurs d'applications est de mettre en œuvre toutes les fonctionnalités avec le moins les bugs possibles, par conséquent, les fonctions de sécurité devraient être la charge d'une autre parti et dans ce cas, les spécialistes d'OAuth sont heureux de partager ce fardeau avec les développeurs d'applications. C'est aussi la raison pour laquelle ce modèle est appliqué largement par de nombreux grands fournisseurs d'API tels que Facebook, Twitter et Google.

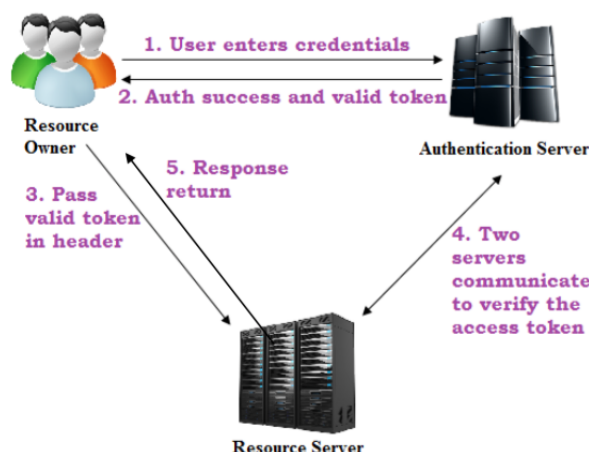


FIGURE 3.1 – API growth

3.2 OAuth 2.0 Core

3.2.1 The OAuth 2.0 Authorization Framework. Comme mentionné dans la section 3.1, OAuth sépare le processus d'authentification du serveur de ressources en utilisant le jeton d'accès qui est accordé par le serveur OAuth avec l'approbation du propriétaire de la ressource et accordé pour un client enregistré pour un temps limité. Fondamentalement, il existe quatre rôles dans OAuth [Hardt, 2012] :

- Propriétaire de la ressource : l'entité accorde l'accès à une ressource sécurisée. Dans le cas où le propriétaire de ressource est une personne, habituellement il représente l'utilisateur final.
- Serveur de ressources : le serveur héberge toutes les ressources qui acceptent le client authentifié pour accéder aux données à l'aide du jeton d'accès.
- Client : la personne, l'application envoie la demande au nom du propriétaire de la ressource pour récupérer les données du serveur de ressource.
- Serveur d'autorisation : le serveur accorde un jeton d'accès au client après avoir réussi tous les processus d'authentification.

Revenons au cas Flockler, pour expliquer de manière plus explicite les rôles dans OAuth, Facebook, Twitter et Instagram jouent un rôle de serveur de ressources, car ces serveurs stockent tous les messages et les photos de Flockler. Par conséquent, les utilisateurs de ces réseaux sociaux sont le propriétaire de la ressource car ils possèdent ces postes et photos et Flockler lui-même est le client. Flockler enverra la demande au nom des utilisateurs pour obtenir le jeton d'accès et utiliser ce jeton d'accès pour recueillir tous les messages et photos nécessaires de ces serveurs. Enfin et surtout, OAuth signifie serveur d'autorisation. Le serveur OAuth vérifiera ces clients et fournira le jeton d'accès pour les clients valides.

Cette section se concentrera sur la façon dont le serveur d'autorisation peut qualifier le client, comment le client peut être représenté pour le propriétaire de la ressource. La réponse à ces questions est l'allocation d'autorisation. Il s'agit d'un justificatif représentant le propriétaire de la ressource et utilisé par le client pour générer le jeton d'accès. Cette autorisation sera envoyée au serveur d'autorisation par Hypertext Transfer Protocol. OAuth définit quatre types d'allocation d'autorisation comme suit [Hardt, 2012] :

- Code d'autorisation : Il s'agit du résultat renvoyé après avoir navigué user-agent (la combinaison des tokens de produit utilisés pour accorder aux applications communicantes l'identification par le nom et la version du logiciel) du propriétaire de la ressource au serveur d'autorisation par le client.

L'application utilise ce code d'autorisation pour communiquer avec le serveur d'autorisation pour passer par le processus d'authentification. Ce code d'autorisation prouve la capacité d'authentifier le client et la possibilité de passer le jeton d'accès directement au client sans passer par le propriétaire de la ressource.

- Implicite : Il s'agit d'une méthode moins compliquée du Code d'autorisation. En flux implicite, les informations d'identification intermédiaires ne sont pas générées et le jeton d'accès est envoyé directement au client en utilisant un langage de script tel que JavaScript. Ce type de méthode est compatible pour l'application dans le navigateur puisque le serveur d'autorisation est capable de vérifier le jeton d'accès dans l'URI et certaines étapes rondes sont déposées dans cette méthode.
- Propriété de ressource Mot de passe Credentials : Le nom d'utilisateur et le mot de passe du propriétaire de la ressource peuvent être utilisés directement pour obtenir le jeton d'accès. Le Client se vérifiera en utilisant ces deux informations d'identification. Toutefois, à mon avis, ce pourrait être l'option la moins sécurisée parmi les quatre types de d'allocation d'autorisation. L'envoi d'un nom d'utilisateur et d'un mot de passe distincts n'est jamais une solution sécurisée puisque l'application de suivi peut écouter votre message et révéler votre nom d'utilisateur et votre mot de passe.
- Client Credentials : Au lieu d'utiliser les informations d'identification du propriétaire de la ressource, le client peut utiliser ses informations d'identification pour s'authentifier et obtenir le jeton d'accès, mais cette méthode ne peut être appliquée que lorsque ces informations sont qualifiées et que le droit du client est limité à modifier et récupérer les ressources.

Cette thèse mettra l'accent sur la méthode d'identification des clients puisque ce type de d'allocation d'autorisation est appliquée largement par divers grands fournisseurs d'API tels que Twitter. Par conséquent, prenons un autre regard sur cette méthode pour avoir une vue générale du flux de travail dans la section suivante.

3.2.2 client authentication in OAUTH 2.0. Chaque client enregistré reçoit un identificateur de client, qui est une chaîne unique et n'est pas confidentiel, par le serveur d'authentification. Chaque fois que le client envoie une requête au serveur d'authentification pour obtenir le jeton d'accès, l'identifiant du client est requis dans cette demande pour prouver la validité de ce client. Le client peut établir un ensemble d'informations d'authentification telles que mot de passe, paire de clés public-privé comme explication ci-dessous :

- Client Password

Selon le modèle «Client Credentials» de la section 3.2.1, le client peut envoyer son nom d'utilisateur et son mot de passe au serveur d'authentification en tant qu'autorisation pour générer un jeton d'accès. Dans OAuth 2.0, le client peut envoyer son identifiant comme nom d'utilisateur en utilisant l'authentification de base HTTP (qui a été introduit dans la section 2.3), mais l'identifiant doit être encodé à l'aide de l'algorithme d'encodage "application / x-www-form-urlencoded". Voici un exemple de demande de jeton d'accès à l'aide du mot de passe du client :

Authorization: Basic czZCaGRSa3F0Mzo3RmpmcDBaQnIxS3REUmJuZlZkbU13

Comme on peut le voir, dans l'en-tête Autorisation, il n'y a qu'une seule chaîne qui est représentée pour le client. Ceci est le résultat après encodage de la combinaison de nom d'utilisateur (identifiant) et mot de passe. Par conséquent, le serveur d'authentification doit prendre en charge l'authentification de base HTTP.

En outre, OAuth 2.0 prend également en charge le client pour inclure ses informations d'identification dans le corps de requête en utilisant les arguments ci-dessous : * Client_id : the client identifier after registration process.

* Client_secret : the confidential string is used only by specific client.

Fondamentalement, le client_id pourrait être vu comme nom d'utilisateur du client et le client_secret

est le mot de passe. Voici l'exemple de la requête utilisant `client_id` et `client_secret` :

POST /token HTTP/1.1

Host: server.example.com

Content-Type: application/x-www-form-urlencoded

```
grant_type=refresh_token&refresh_token=tGzv3J0kF0XG5Qx2TlKWIA&client_id=s6BhdRkqt3&client_secret=7Fjfp0ZBr1KtDRbnfVdmIw
```

Selon OAuth, ce type de d'allocation d'autorisation n'est pas recommandé et ce n'est l'option que si le client d'une certaine manière n'a pas la capacité d'utiliser l'authentification de base HTTP. De mon point de vue, `client_id` explicite et `client_secret` valeur dans la demande pourrait être le problème. En utilisant l'authentification de base de HTTP, le nom d'utilisateur de client et le mot de passe sont encodés et combinés puis envoyés seulement en une chaîne d'encodage au serveur d'authentification, cependant quoiqu'ils soient codés, `client_id` et `client_secret` sont séparés et lisibles dans la demande. Les attaquants pourraient exploiter cette faille pour décrypter ces deux arguments en utilisant l'attaque en texte clair choisie. Cela pourrait être la raison pour laquelle `client_id` et `client_secret` ont une priorité inférieure à l'authentification de base HTTP.

— Public Key Infrastructure

L'infrastructure à clé publique (PKI) est un mécanisme cryptographique et c'est l'innovation de la cryptographie à clé symétrique. Si la clé symétrique utilise la même clé pour le processus de chiffrement et de déchiffrement, la cryptographie à clé publique applique une paire de clés (une clé publique et une clé privée) pour ce processus et sépare chaque clé pour chaque processus. La clé publique n'est pas confidentielle et distribuée communément, cependant, la clé privée doit être gardée secrète. En raison de l'utilisation d'une clé spécifique pour un processus spécifique, le message, qui a été chiffré par clé publique, peut être déchiffré uniquement par clé privée et inversement la clé publique est capable de déchiffrer le message s'il a été chiffré par clé privée [CGI Group, 2004].

Processus de cryptage :

Dans OAuth, chaque client enregistré reçoit une paire de clés, ainsi, chaque fois que le client demande au serveur d'authentification de générer le jeton d'accès en utilisant les informations d'identification du client, le client doit inclure sa clé publique et privée dans cette requête. Comme on peut le voir sur la figure 3, au début, étape 1a, le message lisible est calculé digest, une représentation unique (un peu comme un checksum sophistiqué) en utilisant la technique de hachage. Après cela à l'étape 1b, le résumé est crypté en utilisant la clé privée du client pour générer la signature, mais cette signature inclut également la clé publique du client. Cela pourrait aider le serveur d'autorisation à vérifier et qualifier cette signature en utilisant la clé publique de l'émetteur. Dans l'étape suivante 2a, la clé de cryptage / décryptage symétrique est générée une fois puisque les clés asymétriques sont encombrantes pour un long message. Dans le processus de cryptage, le message, la signature et la clé publique seront cryptés à nouveau par une clé symétrique comme étape 2b. Cela conduit à inclure cette clé symétrique à l'intérieur du message afin que le destinataire puisse décrypter plus tard. Cependant, il y a une considération ici qui est que comment empêcher une autre personne de voir cette clé symétrique, cela signifie que seul le destinataire, à qui la demande est envoyée, est capable de le récupérer et de décrypter ce message. Le moyen le plus sécurisé est de crypter de nouveau la clé symétrique par la clé publique destinataire à l'étape 2c. Enfin, le paquet entier sera envoyé au serveur.

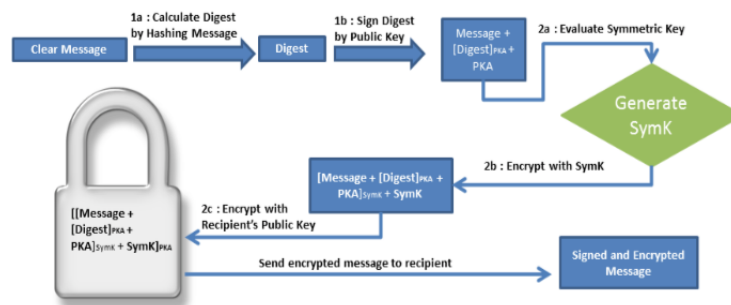


FIGURE 3.2 – API growth

Processus de décryptage :

Après le processus de cryptage, le destinataire reçoit l'ensemble du paquet. Tout d'abord, le destinataire doit déchiffrer ce paquet en utilisant sa clé publique comme dans la description à l'étape 1a. Après déballage du paquet, le destinataire a le message chiffré et la clé symétrique. Ensuite, cette clé symétrique est utilisée pour decoder le message en tant qu'état 1b pour collecter la clé publique du client et la signature. A l'étape de la vérification, il existe deux processus parallèles. Le premier processus consiste à déchiffrer la signature en utilisant la clé publique qui a été collectée à l'étape précédente 1b. Le résultat final de l'étape 2a est d'obtenir le résumé à l'intérieur de cette signature. L'autre processus évalue à nouveau le message en utilisant la même formule de hachage que le client, puisque le hachage est un processus à sens unique. Cela signifie que le serveur ne peut pas décrypter le digest pour récupérer ce message. Par conséquent, le résultat final de l'étape 2b est également un digest. A ce moment, le serveur compare deux digestures de l'étape 2a et 2b. Dans le cas où ces digest sont exactement les mêmes, le message et la signature sont authentifiés et acceptés. Ce modèle d'autorisation est appliqué de manière

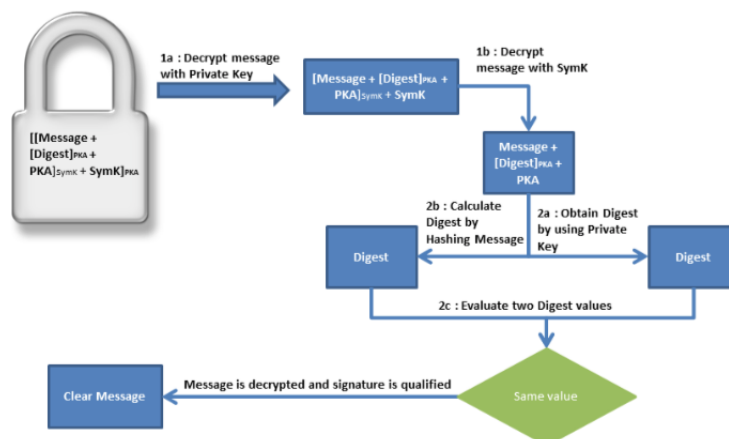


FIGURE 3.3 – API growth

significative par de nombreux fournisseur API qui permettent aux programmeurs de développer leur propre application pour récupérer automatiquement des données à partir du serveur de ressources telles que Twitter, Facebook et de nombreux autres réseaux sociaux. Ce modèle sera analysé dans le système Twitter dans le prochain chapitre.

4. Twitter security methods

Twitter, l'un des plus importants fournisseurs d'API multimédia, qui compte 307 millions de clients actifs par mois [Statista, 2015], fournit diverses méthodes pour récupérer ses données au nom de l'utilisateur de Twitter en utilisant son application. En faisant des appels autorisés aux API de Twitter, l'application peut accéder aux amis et aux adeptes de n'importe quel compte, récupérer toute information d'utilisateur et accéder aux ressources de l'utilisateur. Par cette commodité, une énorme quantité de sites Web intégrés utilisent Twitter API REST chaque jour et cette tendance n'a pas de fin au moins pour les prochaines années. Cependant, cela conduirait à un grand risque et un défi pour Twitter de sécuriser ses références client. Ce chapitre analyse et se concentre sur la façon dont Twitter autorise ces applications pour éviter la fuite des ressources de l'utilisateur de façon inattendue.

4.1 Résumé de la méthode de sécurité twitter : 3-Legged OAuth 2.0

Twitter vérifie les droit de l'application en suivant le modèle OAuth 2.0 à 3 pattes. Ce modèle est l'interaction entre trois rôles, c'est pourquoi ce modèle a nommé OAuth à trois pattes, le propriétaire de la ressource, le serveur et le client (application). Le flux global de ce modèle est que le propriétaire de la ressource doit accorder l'accès et restreindre l'interaction au client et ce dernier communique avec le serveur par le jeton d'accès au nom de ce propriétaire de ressource. De cette façon, l'application est capable de récupérer ce que l'utilisateur à accès. Cela réduit le risque de perte et de modification des ressources. Pour envoyer une requête valide, l'application doit obtenir le jeton d'accès au nom d'un utilisateur de Twitter. Il s'agit de la phase obligatoire et de la première étape du processus d'authentification. Il vise à :

- Définir si l'application a les droits pour accéder à un compte d'utilisateur. L'utilisateur doit accorder l'accès à l'application à son compte manuellement pour récupérer des données. Cela prouve non seulement que l'utilisateur est au courant de l'existence de l'application, mais prend également en charge l'utilisateur pour gérer le droit d'application. Chaque fois que l'internaute se sent insatisfait de cette demande, il / elle est libre de bloquer l'accès à l'application.
- Limiter l'interaction de l'application. L'application est uniquement en mesure de mettre en œuvre ce que les utilisateurs lui ont permis de faire. Cela signifie que l'application ne peut pas interagir sur les ressources auxquelles le propriétaire de la ressource n'accorde pas d'accès. Cela aide à protéger les ressources et à maintenir l'intégrité du système en évitant une fuite d'information ou un changement accidentel de la ressource.

S'authentifier avec Twitter.

Twitter continue à demander de fournir un nom d'utilisateur valide et mot de passe, puis continuer avec le processus ci-dessus.

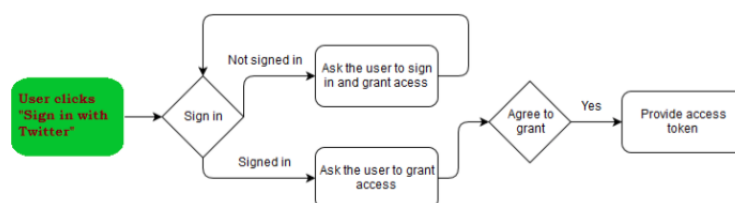


FIGURE 4.1 – API growth

Selon "OAuth FAQ" Document sur Twitter [Twitter Documentation, 2015], actuellement Twitter n'expire pas le jeton d'accès. Cela signifie que le jeton d'accès sera valide tant que l'utilisateur autorise son accès. De mon point de vue, cela apporterait de nombreux avantages tant pour l'utilisateur que pour l'application. Tout d'abord, l'application n'est pas obligée d'envoyer une requête pour générer de nouveaux jetons d'accès fréquemment et, par conséquent, la charge de travail sur le serveur et le côté client sont considérablement réduites. Deuxièmement, le client n'est pas tenu de se connecter chaque fois que le jeton d'accès précédent soit expiré.

La mise en œuvre de Twitter, qui repose sur le flux de la spécification OAuth 2 de Grant (dans la section 3.2.1), est illustrée à la Figure 10. Prenons maintenant un aperçu du flux technique en détail pour voir comment Twitter authentifie le propriétaire de la ressource et Application pour générer un jeton d'accès.

Step 1 : Obtaining a request token

Au début, l'application doit envoyer la requête au serveur OAuth pour générer un jeton de demande. Ce jeton est représenté pour un client spécifique et ce n'est pas un jeton d'accès, ce qui signifie que ce propriétaire de ressource doit accorder l'accès à l'application à travers ce jeton de demande.

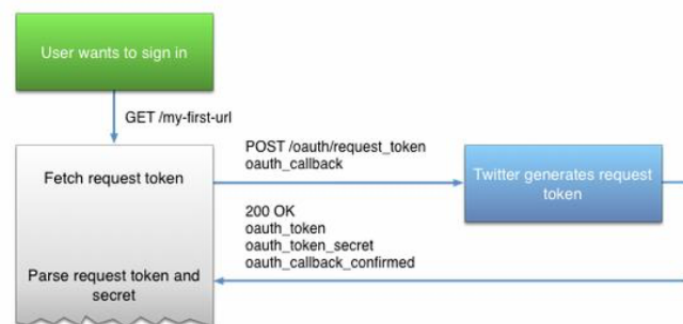


FIGURE 4.2 – API growth

La figure 10 montre le flux global pour cette étape, l'application doit envoyer la requête POST incluant ses informations d'identification au serveur. Après cela, Twitter génère le jeton de demande et le renvoie. Voici un exemple de la demande :

```

POST /oauth/request_token HTTP/1.1
User-Agent: themattharris' HTTP Client
Host: api.twitter.com
Accept: */*
Authorization:
    OAuth oauth_callback="http%3A%2F%2Flocalhost%2Fsign-in-
with-twitter%2F", oauth_consumer_key="cChZNFj6T5R0TigYB9yd1w",
        oauth_nonce="ea9ec8429b68d6b77cd5600adbbb0456",
        oauth_signature="F1Li3tvehgcraF8DMJ70yx04w9Y%3D",
        oauth_signature_method="HMAC-SHA1",
        oauth_timestamp="1318467427",
        oauth_version="1.0"
  
```

Il y a deux arguments les plus importants dans cette demande. Elles sont :

- `OAuth_consumer_key` : La valeur représente le client (application). Cette valeur a pour but de notifier au serveur d'authentification quel client demande le jeton de demande et d'éviter toute erreur dans la relation de mappage entre le client et le propriétaire de la ressource.
- `OAuth_callback` : Il s'agit d'un lien vers lequel le serveur d'authentification va naviguer après sa connexion réussie. Twitter inclura le jeton d'accès et jeton secret dans la réponse et les renvoie à l'application à ce lien. Par conséquent, à cette étape, l'application doit avertir le serveur de la destination de ces informations top-secret pour éviter la fuite des informations d'identification sensibles.

step 2 : Redirecting the user

Après la première étape, le serveur d'authentification est informé par le client du propriétaire de la ressource qui souhaite autoriser l'accès à cette application. Ainsi, à cette étape, le client doit naviguer entre le propriétaire de la ressource et le serveur afin de vérifier lui-même et d'accorder l'accès comme le montre la Figure 11.

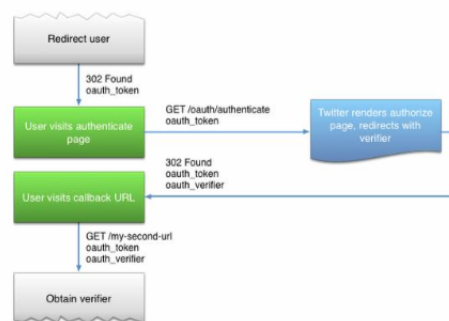


FIGURE 4.3 – API growth

Tout d'abord, le client doit inclure le jeton de demande qui est retourné par le serveur à l'étape 1 et parcourir sa ressource à la page d'authentification (page de connexion) en utilisant la redirection HTTP 302. Ensuite, le propriétaire de la ressource doit fournir un nom d'utilisateur et un mot de passe valides. Selon le statut du propriétaire de la ressource, il y a trois cas possibles :

- Signé et approuvé. Si le propriétaire de la ressource a déjà accepté et approuvé l'application, cette étape sera ignorée et le serveur renverra automatiquement le jeton d'accès valide à l'URL de rappel immédiatement.
- Signé mais non approuvé. Si le propriétaire de la ressource a signé, cependant, il n'a pas approuvé l'application encore (comme mentionné ci-dessus, le propriétaire de la ressource se doit d'approuver le client qu'une seule fois), l'authentification est affichée pour l'utilisateur pour accorder et gérer l'accès de l'application. Après cela, le jeton de demande sera retourné.
- Non connecté. Pour le nouveau propriétaire de la ressource qui n'a pas signé et approuvé l'application, la page de connexion s'affiche afin d'entrer le nom d'utilisateur et le mot de passe. Ensuite, le flux sera identique à l'état "Signé mais non approuvé".

Voici les pages d'authentification :

step 3 : Converting the request token to an access token

Après l'étape 2, le propriétaire de la ressource a déjà publié le droit à l'application. En ce moment, l'application est en mesure d'accéder aux ressources et de récupérer des données, cependant, il ya une étape de plus à terminer. Il obtient le jeton d'accès en validant le jeton de demande. Cette étape vise à

informer le serveur que le propriétaire de la ressource a autorisé l'accès de l'application.

Voici l'exemple de demande de jeton d'accès :

```
POST /oauth/access_token HTTP/1.1
User-Agent: themattharris' HTTP Client
Host: api.twitter.com
Accept: */*
Authorization: OAuth oauth_consumer_key="cChZNFj6T5R0TigYB9yd1w",
                  oauth_nonce="a9900fe68e2573b27a37f10fbad6a755",
                  oauth_signature="39cipBtIOHEEnybAR4sATQTp12I%3D",
                  oauth_signature_method="HMAC-SHA1", oauth_timestamp="1318467427",
                  oauth_version="1.0",

oauth_token="NPcudxy0yU5T3tBzho7iCotZ3cnetKwcTIRlX0"
Content-Length: 57
Content-Type: application/x-www-form-urlencoded
oauth_verifier=uw7NjWHT60J1MpJ0XsHfNxoAhPKpgI8B1YDhxEjIBY
```

L'application doit envoyer une requête POST au serveur d'authentification pour mettre à niveau le jeton de demande. Dans cette demande, deux arguments qui une fois de plus illustrer la haute sécurité de la mise en œuvre Twitter :

- `OAuth.token` : Le jeton de demande qui a été retourné à l'étape 1. L'application doit clarifier quel jeton utiliser pour communiquer avec le serveur pour le compte du propriétaire de la ressource. Toutefois, si une autre application falsifie le jeton de demande et l'envoie au serveur au nom de l'application, si le serveur ne vérifie pas la validité de ce jeton de demande et renvoie le jeton d'accès valide à l'application sophistiquée. Par coïncidence, le serveur d'authentification révèle juste ses informations d'identification utilisateur . C'est la raison pour laquelle chaque demande doit inclure le deuxième argument.
- `OAuth.verifier` : Il s'agit de la valeur renvoyée est l'étape 2. Cette valeur a pour but de notifier au serveur la validité du jeton de demande. Cette valeur ne sera retournée que lorsque le propriétaire de la ressource aura accepté l'application. Le vérificateur `oauth` dans la requête doit correspondre au vérificateur `oauth` qui a été stocké dans le serveur après l'étape 2. Par conséquent, la requête sophistiquée ne peut pas être autorisée sans un `auth.verifier` valide.

Après avoir récupéré le jeton d'accès, le client peut communiquer avec le serveur par ce jeton au nom du propriétaire de la ressource pour collecter des données, créer un objet et même supprimer la ressource si possible tant que le propriétaire de la ressource a accepté ce client.

En général, l'implémentation Twitter sur l'authentification de l'application a été analysée. Actuellement, Twitter applique 3-Legged OAuth 2.0 qui représente le propriétaire de ressource, le serveur et le client (application). Dans ce modèle, le propriétaire de la ressource doit se s'authentifier auprès du serveur en fournissant un nom d'utilisateur et un mot de passe valides. Après cela, il / elle doit habilitier l'application à interagir sur ses données. Enfin, le serveur émet un jeton d'accès à l'application et il peut être utilisé pour le compte du propriétaire de la ressource. Jusqu'à présent, nous avons clarifié la relation, les dépendances entre le propriétaire de la ressource, le serveur et l'application. Dans la section suivante, nous continuons à rechercher les conditions de communication entre l'application et le serveur.

4.2 Authentication & Authorization in Twitter API calls

L'application peut envoyer de nombreuses requêtes à l'API de Twitter pour récupérer, supprimer et créer des ressources. D'une part, cela pourrait rapprocher Twitter du public en raison de l'énorme ressource d'information et de ses flexibilités. D'autre part, cela pourrait être le plus grand risque pour Twitter car il pourrait y avoir des millions de requêtes venant de millions d'application chaque jour. Par conséquent, il ya deux charges exceptionnelles auxquelles Twitter est confronté :

- Comment Twitter peut-il authentifier l'application et ses demandes ?
- Comment Twitter sécuriser l'information sensible, contenu dans les requêtes comme l'accès Jeton, jeton d'accès secret, clé client et secret client ?

La réponse clé a été introduite ci-dessus à la section 3.2.2. Twitter applique partiellement l'infrastructure à clé publique pour protéger les données sensibles envoyées entre le client et le serveur. Afin de tester cette fonction, nous allons rechercher quelles étapes sont nécessaires pour envoyer et lire la demande côté client et côté serveur, au cas où l'application envoie la requête de tweeter un nouveau tweet sur le serveur de Twitter :

Le coté client :

Les données brutes ne sont pas censées être envoyées de client à serveur puisque l'autre application est capable d'écouter et de capturer ces secrets. Par conséquent, avant d'envoyer la requête, l'application doit nécessairement masquer la clarté du message en utilisant l'algorithme de hachage. Le flux global peut être expliqué par la figure 13.

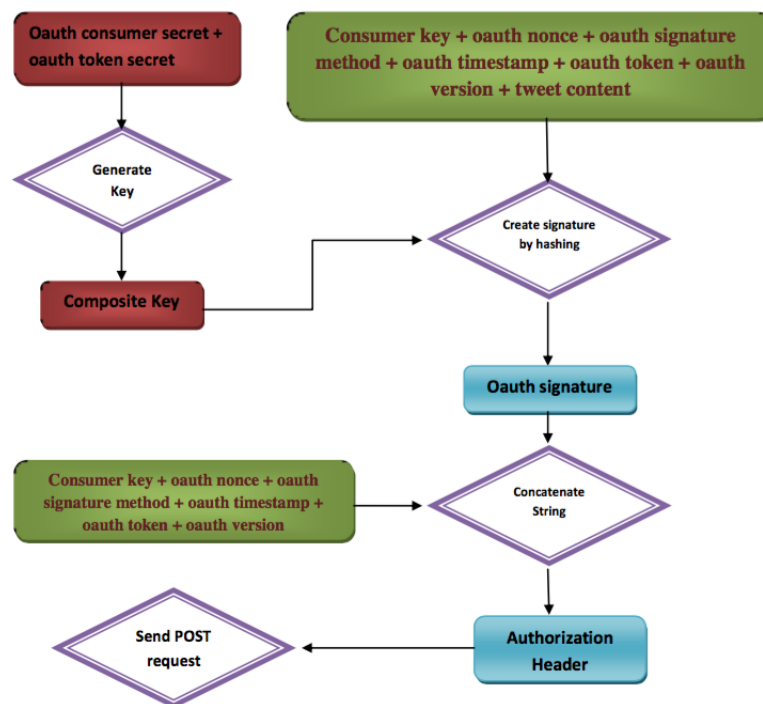


FIGURE 4.4 – API growth

Comme on peut le voir sur la figure ci-dessus, au début, l'application doit calculer une clé composite à partir de deux clés privées, le secret de consommateur OAuth et le secret de jeton OAuth. Ces deux clés sont censées être confidentielles. Ensuite, l'application doit créer la signature par hachage du message, qui est concaténée à partir de nombreux arguments tels que la clé de consommateur, oauth nonce, méthode de signature oauth, timestamp oauth, jeton oauth, version oauth et la valeur de la requête dans ce cas est le contenu tweet. Enfin, la signature oauth sera placée dans l'en-tête d'autorisation dans la requête POST HTTP avec d'autres arguments et envoyée au serveur Twitter. Voici l'exemple de la demande POST d'une application :

POST

/1.1/statuses/update.json?status=New%20Tweet%20from%20API&display_coordinates=false HTTP/1.1

```
Authorization: OAuthoauth_consumer_key="DC0sePOBbQ8bYdC8r4Smg"
              oauth_signature_method="HMACSHA1"
              oauth_timestamp="1447763195" oauth_nonce="2757352841"
              oauth_version="1.0"
              oauth_token="4204335839-
JKlJn0249TnMegiDsJaQDkxgDyDPizQQIi8nV07"
              oauth_signature="zkKRtFwatfH8qjn%2FvxQnhnhyowY%3D"
Host: api.twitter.com
```

Il semble y avoir quelques arguments dupliqués dans ce processus tels que la clé de consommateur, oauth nonce, méthode de signature oauth et ainsi de suite. Cependant, dans la section 2.2.4 selon les apatrides, toutes les informations essentielles pour cette demande doivent être incluses dans la demande. Par conséquent, chaque argument joue un rôle différent dans le décryptage et le processus d'authentification tels que :

- `OAuth_consumer_key` : notifie au serveur quelle application envoie un appel API.
- `OAuth_signature_method` : algorithme de hachage qui a été appliqué pour calculer la signature. L'authentification doit connaître cette méthode puisque le serveur va calculer la signature une fois de plus par elle-même pour valider la signature.
- `OAuth_timestamp`, `oauth_nonce`, `oauth_version` : Ces trois arguments ont été utilisés pour générer la signature, ainsi, le serveur doit les collecter. En outre, comme mentionné ci-dessus, ces arguments aident à empêcher l'attaque de texte en clair.
- `OAuth_token` : le jeton valide qui représente pour le droit de l'application. Le serveur d'authentification qualifie les appels API de l'application par ce jeton.
- `OAuth_signature` : le message chiffré du client. Ce message illustre ce que l'application souhaite que le serveur de ressources implémente.

Dans ce cas, Twitter applique le modèle PKI, donc, il ya une paire de clés qui utilise le processus d'authentification. Malgré le nom différent, la clé (clé publique) et le secret (la clé privée), ils jouent toujours le même rôle. Cependant, il ya un changement majeur dans le processus de cryptage. Au lieu de crypter le message entier, Twitter requiert seulement que le message soit haché en utilisant l'algorithme HMACSHA1 en utilisant la clé qui a été calculée à partir de deux clés secrètes. Les principaux objectifs de cette étape sont les suivants :

- Cacher l'explicité du message. Évidemment, envoyer les informations brutes entre serveur et client n'est jamais une bonne solution puisque le pirate peut écouter votre appel, ainsi, Twitter demande à l'application de crypter ces informations confidentielles par des clés secrètes.

- Adresser la validité de la demande. Comme on peut le voir sur l'extrait de code 3, certains arguments tels que `oauth_token`, `oauth_consumer_key` sont transmis directement et envoyés directement au serveur sans cryptage. Par conséquent, la signature joue un rôle comme la certification pour la demande valide. Étant donné que la signature a été calculée par les deux clés secrètes, bien que `oauth_token`, `oauth_consumer_key` soient distribués communément, l'autre application ne peut pas falsifier la requête sans la signature valide. Pour cette raison, les autres arguments ne sont pas chiffrés et transparents.

Le coté serveur :

Selon les règles du modèle PKI, après avoir reçu la demande de l'application, le serveur va valider la demande en comparant deux valeurs de la signature. Le premier est la signature originale dans la requête et le second est le résultat après le calcul de la signature par le serveur. Cette section explique comment le serveur calcule la signature par lui-même. Cette section réutilisera l'extrait de code 3 comme exemple.

POST

```
/1.1/statuses/update.json?status=New%20Tweet%20from%20API&display_coordinates=false
```

Comme défini dans la section 2.2.2, le serveur sépare la valeur de la requête de l'URL. Dans ce cas, la valeur est le contenu du nouveau tweet et il est "New Tweet from API". Retour à Figure 13, afin de générer la signature, le serveur prendra le contenu tweet de l'URL et les autres arguments dans la demande. A ce stade, le serveur calcule la clé à partir de deux clés secrètes basées sur la clé `oauth_consumer_key` et la `oauth_token` dans la requête. Le point intéressant est que serveur Twitter enregistre toutes les clés publiques et privées de toutes les applications, ainsi, le serveur est capable de calculer la clé sans demander ces clés secrètes de client. Enfin, le serveur compare la signature de la requête et la signature qui a été générée par elle-même. Si les deux signatures sont identiques, le serveur exécutera automatiquement cette requête et répondra à la requête du client, sinon le serveur ne suivra pas cette requête et renverra un message d'erreur.

Le processus d'authentification côté serveur vise à :

- Prouver la validité de la demande. En testant la signature, le serveur est en mesure de vérifier la demande envoyée par l'application autorisée.
- Vérifiez le droit de l'application. Grâce à la méthode HTTP et au message, le serveur peut vérifier si l'application est autorisée à déclencher cette requête. Si l'application n'a pas encore accordé l'accès, le serveur retournera un message d'erreur sans exécuter cette requête.
- Vérifiez la fiabilité du message. Le serveur doit vérifier le message pour s'assurer qu'il est le message original du client sans aucune modification ou changement par un tiers. Hash est un chiffrement unidirectionnel et dans ce cas, le message a été haché en utilisant les deux clés secrètes, donc, il est presque impossible pour une autre application de fausser la signature et le message en même temps. Dans le cas où le message a été falsifié, lorsque le serveur calcule la signature et compare deux valeurs, ces deux valeurs ne peuvent pas être identiques et cette requête se révèle invalide.

Dans l'ensemble, afin de protéger le propriétaire de la ressource, Twitter s'attend à ce que leurs clients émettent le droit à l'application en utilisant l'authentification à 3 jambes. Après cela, l'application passe le jeton, qui est représenté pour son droit, à la requête et communique avec le serveur par ce jeton. L'application est capable de travailler sur les ressources sur lesquelles le propriétaire de la ressource lui permettrait d'interagir. Afin de sécuriser les informations sensibles qui sont envoyées d'avant en arrière entre le client et le serveur, l'application doit générer la signature à partir du message en le hachant

avec deux clés secrètes. La valeur évidente du message ne doit pas être envoyée directement au serveur pour éviter la fuite des informations top secret.

Dans le chapitre suivant, la fonction de sécurité d'un fournisseur d'API sera exécutée. Prenons un autre regard sur la façon dont les détails de l'utilisateur, tels que le numéro de compte bancaire et de sécurité, sont protégés lors de transactions. Le prochain chapitre utilisera le Stripe, un fournisseur de paiement en ligne, et ses services comme exemple.

Acknowledgements

This is optional and should be at most half a page. Thanks Ma, Thanks Pa. One paragraph in normal language is the most respectful.

Do not use too much bold, any figures, or sign at the bottom.

5. The Title of Your First Appendix

Do you *really* need an appendix? What does it add?

5.1 section title

[illegible]

Rule 1 Rule 2 Rule 3. Everyone knows that.