Json Web Tokens

N° de la lecture individuelle :	3
Étudiant	Laurent Térence, 802_1F
Sujet	Json web token
Source	JSON Web Tokens - jwt.io / livre officiel : Auth0 | JWT Handbook

Json Web Token

Table des matières

Table des illustrations

1 Introduction

1.1 What is a JSON Web Token?

Un JSON Web Token (JWT) est une chaîne de caractères qui représente des informations sous une forme compacte et lisible. Il est utilisé pour vérifier l'authenticité des données échangées entre différentes parties.

Un JWT est composé de trois parties distinctes, séparées par des points. Chaque partie est encodée en Base64 pour faciliter sa transmission.

Exemple : ***eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.***
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9.
TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ

La première partie, appelée l'en-tête (header), contient des informations sur l'algorithme de signature utilisé et le type de token.

La deuxième partie, appelée le corps (payload), contient les revendications (claims) qui sont des assertions ou des informations sur l'utilisateur ou l'objet concerné. Certaines revendications sont standardisées, comme "sub" qui représente l'identifiant de l'utilisateur, tandis que d'autres peuvent être définies par l'utilisateur.

La troisième partie, appelée la signature (signature), est utilisée pour vérifier l'intégrité des données. Elle est générée en utilisant la clé secrète connue uniquement par le serveur qui a émis le JWT.

Les JWT sont utilisés dans de nombreux scénarios pour sécuriser les communications et établir l'identité des utilisateurs de manière fiable. Ils peuvent être signés et/ou chiffrés pour garantir leur confidentialité et leur intégrité.

En résumé, un JWT est une manière compacte et sécurisée de transmettre des informations entre différentes parties, tout en garantissant leur authenticité et leur intégrité. Il est largement utilisé dans les applications web et les services d'authentification.

{ "alg": "HS256", "typ": "JWT" }

{ "sub": "1234567890", "name": "John Doe", "admin": true }

Ceci est un exemple de contenu d'un JSON Web Token (JWT) décomposé en deux parties.

La première partie, qui est l'en-tête (header), contient deux attributs :

- "alg" qui spécifie l'algorithme de signature utilisé, dans cet exemple "HS256" qui fait référence à l'algorithme de hachage HMAC-SHA256.

- "typ" qui indique le type de token, dans ce cas "JWT" pour indiquer qu'il s'agit d'un JSON Web Token.

La deuxième partie, qui est le corps (payload), contient les revendications (claims) qui sont des assertions ou des informations spécifiques. Dans cet exemple, nous avons trois revendications :

- "sub" représente l'identifiant de l'utilisateur, ici "1234567890".

- "name" indique le nom de l'utilisateur, dans ce cas "John Doe".

- "admin" est un indicateur pour savoir si l'utilisateur est un administrateur, ici sa valeur est "true" pour indiquer que l'utilisateur est un administrateur.

Ces informations sont encodées en Base64 pour faciliter leur transmission et leur lecture. Le JWT peut ensuite être signé avec une clé secrète pour garantir son intégrité et sa sécurité lors de la transmission entre les différentes parties.

1.2 What problem does it solve?

Les JSON Web Tokens (JWTs) ont pour objectif principal de transférer des revendications entre deux parties. Cependant, l'aspect le plus important est l'effort de standardisation sous la forme d'un format de conteneur simple, facultativement validé et/ou chiffré. Des solutions ad hoc à ce même problème ont été mises en œuvre par le passé, tant privément que publiquement. Des normes plus anciennes existent également pour établir des revendications sur certaines parties. Ce que les JWT apportent, c'est un format de conteneur simple, utile et standard.

Bien que la définition donnée jusqu'à présent soit un peu abstraite, il n'est pas difficile d'imaginer comment ils peuvent être utilisés, notamment dans les systèmes de connexion (bien que d'autres utilisations soient possibles). Nous examinerons de plus près les applications pratiques dans le chapitre 2. Certaines de ces applications comprennent :

- L'authentification
- L'autorisation
- L'identité fédérée
- Les sessions côté client

Le terme "solutions ad hoc" fait référence à des solutions temporaires ou spécifiques à un problème donné. Ce sont des solutions qui sont mises en place de manière improvisée pour répondre à un besoin spécifique, sans nécessairement suivre des normes ou des standards établis. Dans le contexte des JSON Web Tokens (JWTs), cela signifie qu'avant l'introduction des JWTs, il existait différentes approches et méthodes pour transférer des informations entre les parties, mais elles n'étaient pas standardisées. Les JWTs fournissent donc une solution standardisée et plus efficace pour ce type de transfert de données.

1.3 A little bit of history

Le groupe JSON Object Signing and Encryption (JOSE) a été formé en 2011 dans le but de standardiser les mécanismes de protection de l'intégrité (signature et MAC) et de chiffrement, ainsi que le format des clés et des identifiants d'algorithme pour favoriser l'interopérabilité des services de sécurité utilisant JSON. En 2013, une série de brouillons, comprenant un recueil d'exemples d'utilisation des idées développées par le groupe, étaient disponibles. Ces brouillons ont ensuite été transformés en RFC (Request for Comments) pour JWT, JWS, JWE, JWK et JWA. Depuis 2016, ces RFC sont en cours de standardisation et n'ont pas été sujets à des corrections. Le groupe est actuellement inactif. Les principaux auteurs de ces spécifications sont Mike Jones, Nat Sakimura, John Bradley et Joe Hildebrand.

2 Practical Applications

2.1 Client-side/Stateless Sessions

Les sessions dites "stateless" sont simplement des données stockées côté client. Dans cette application, l'aspect important réside dans l'utilisation de la signature et éventuellement du chiffrement pour authentifier et protéger le contenu de la session. Il est crucial de manipuler avec précaution les données côté client, car elles peuvent être modifiées.

Les JSON Web Tokens (JWT), grâce à leurs fonctionnalités de signature (JWS) et de chiffrement (JWE), permettent de garantir l'intégrité des données et de les protéger contre la lecture par des tiers non autorisés.

Dans la plupart des cas, les sessions ont simplement besoin d'une signature. Cela signifie que les données stockées dans les JWT peuvent être lues par des tiers sans soulever de problèmes de sécurité ou de confidentialité. Par exemple, une revendication courante dans un JWT est la revendication "sub" (sujet), qui identifie généralement une partie par rapport à une autre (par exemple, un identifiant d'utilisateur ou une adresse e-mail). Il n'est pas nécessaire que cette revendication soit unique, mais d'autres revendications peuvent être utilisées pour identifier de manière unique un utilisateur.

Cependant, certaines revendications, telles que la revendication "items" représentant le panier d'achat d'un utilisateur, doivent être protégées contre la lecture par des tiers. Si ces données sensibles sont stockées dans un JWT non chiffré, un script côté client pourrait les récupérer, ce qui soulève des préoccupations en termes de confidentialité. Il est donc important de prendre des mesures appropriées pour protéger ces informations sensibles.

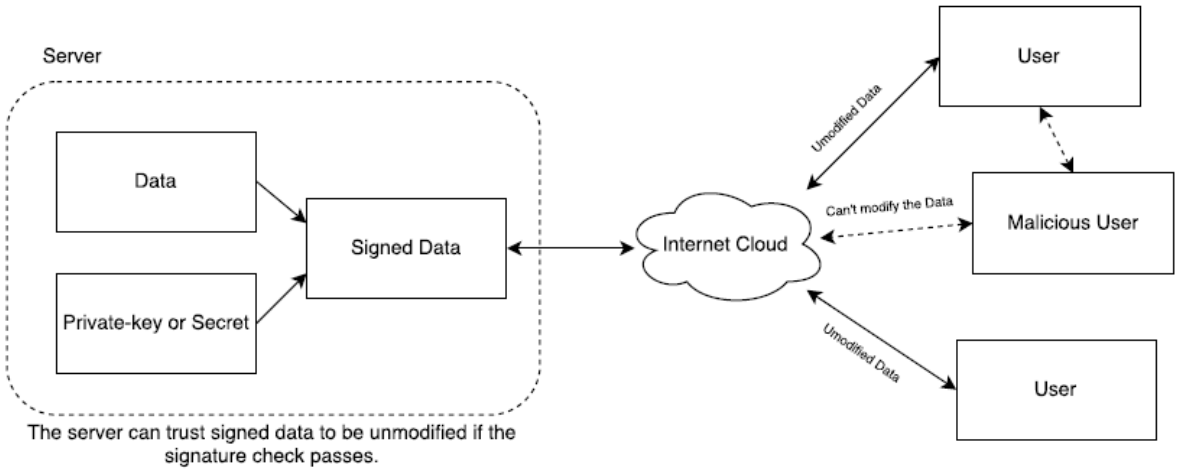


Figure 1 Client-Side Signed Data

2.1.1 Security Considerations

2.1.1.1 Signature Stripping

Une méthode courante pour attaquer un JWT signé est simplement de supprimer la signature. Les JWT signés sont composés de trois parties distinctes : l'en-tête, la charge utile et la signature. Ces trois parties sont encodées séparément. Par conséquent, il est possible de supprimer la signature puis de modifier l'en-tête pour prétendre que le JWT est non signé. Une utilisation imprudente de certaines bibliothèques de validation de JWT peut conduire à considérer des jetons non signés comme des jetons valides, ce qui permet à un attaquant de modifier la charge utile à sa guise. Cette situation peut être facilement résolue en veillant à ce que l'application chargée de la validation ne

considère pas les JWT non signés comme valides. Il est donc essentiel de mettre en place une validation adéquate pour prévenir ce type d'attaque.

Figure 2 Signature Stripping

2.1.1.2 Cross-Site Request Forgery (CSRF)

Les attaques de falsification de requête entre sites (CSRF) essaient de faire des requêtes sur des sites où tu es connecté en trompant ton navigateur pour qu'il envoie une requête depuis un autre site. Pour cela, un site spécialement créé doit contenir l'adresse de la cible. Un exemple courant est une image <img> dans une page méchante avec un lien vers la cible de l'attaque. Par exemple :

<!-- Cela est intégré dans une page d'un autre site -->

<img src="http://target.site.com/add-user?user=name&grant=admin">

La balise <img> ci-dessus enverra une requête à target.site.com chaque fois que la page qui la contient est chargée. Si tu t'étais déjà connecté à target.site.com et que le site utilise un cookie pour maintenir ta session active, ce cookie sera également envoyé. Si le site cible n'a pas mis en place de techniques pour se protéger contre les attaques CSRF, la requête sera traitée comme une requête valide venant de toi.

Les JWT, comme d'autres données stockées côté client, peuvent être enregistrés sous forme de cookies. Des JWT de courte durée peuvent être utiles dans ce cas. Les techniques courantes pour se protéger contre les attaques CSRF incluent l'ajout d'en-têtes spéciaux aux requêtes uniquement lorsque celles-ci proviennent du bon site, l'utilisation de cookies de session et de jetons spéciaux pour chaque requête. Si les JWT (et les données de session) ne sont pas stockés sous forme de cookies, les attaques CSRF ne sont pas possibles. Cependant, il est toujours possible de subir des attaques de type cross-site scripting.

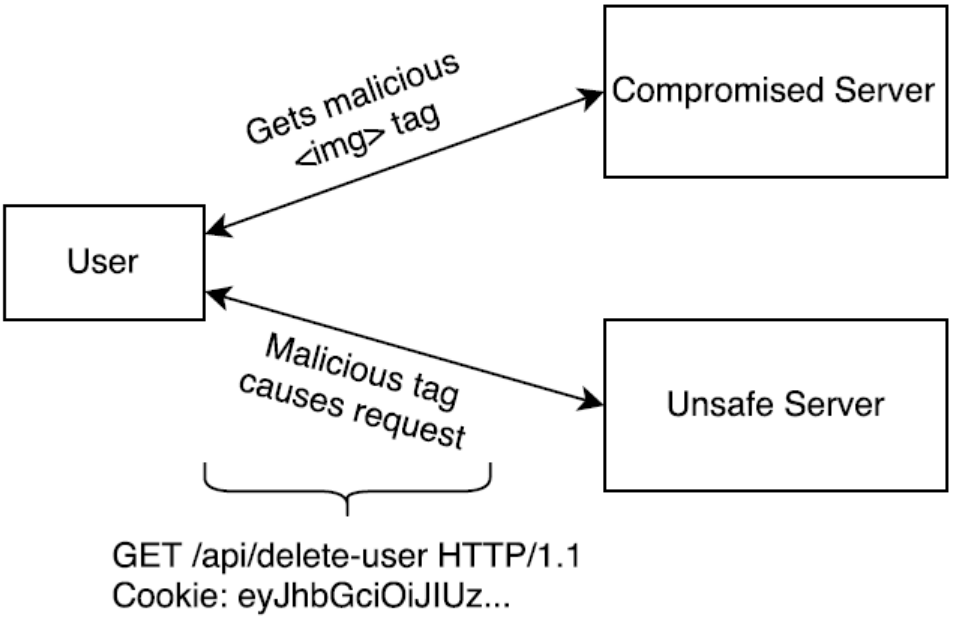


Figure 3 Cross-Site Request Forgery

2.1.1.3 Cross-Site Scripting (XSS)

Les attaques de cross-site scripting (XSS) cherchent à injecter du code JavaScript sur des sites de confiance. Ce code JavaScript injecté peut ensuite voler des jetons d'authentification à partir des cookies et du stockage local. Si un jeton d'accès est divulgué avant son expiration, un utilisateur malveillant pourrait l'utiliser pour accéder à des ressources protégées. Les attaques XSS courantes sont généralement causées par une validation incorrecte des données transmises au serveur (de manière similaire aux attaques par injection SQL).

Un exemple d'attaque XSS pourrait être lié à la section des commentaires d'un site public. Chaque fois qu'un utilisateur ajoute un commentaire, il est enregistré par le serveur et affiché aux utilisateurs qui chargent la section des commentaires. Si le serveur ne désinfecte pas les commentaires, un utilisateur malveillant pourrait écrire un commentaire de manière à ce qu'il puisse être interprété par le navigateur comme une balise <script>. Ainsi, un utilisateur malveillant pourrait insérer du code JavaScript arbitraire et l'exécuter dans le navigateur de chaque utilisateur, volant ainsi des informations d'identification stockées sous forme de cookies et dans le stockage local. Les techniques d'atténuation reposent sur une validation appropriée de toutes les données transmises au serveur. En particulier, toutes les données reçues des clients doivent toujours être désinfectées. Si des cookies sont utilisés, il est possible de les protéger contre l'accès par JavaScript en définissant le drapeau HttpOnly. Le drapeau HttpOnly, bien qu'utile, ne protégera pas le cookie contre les attaques CSRF.

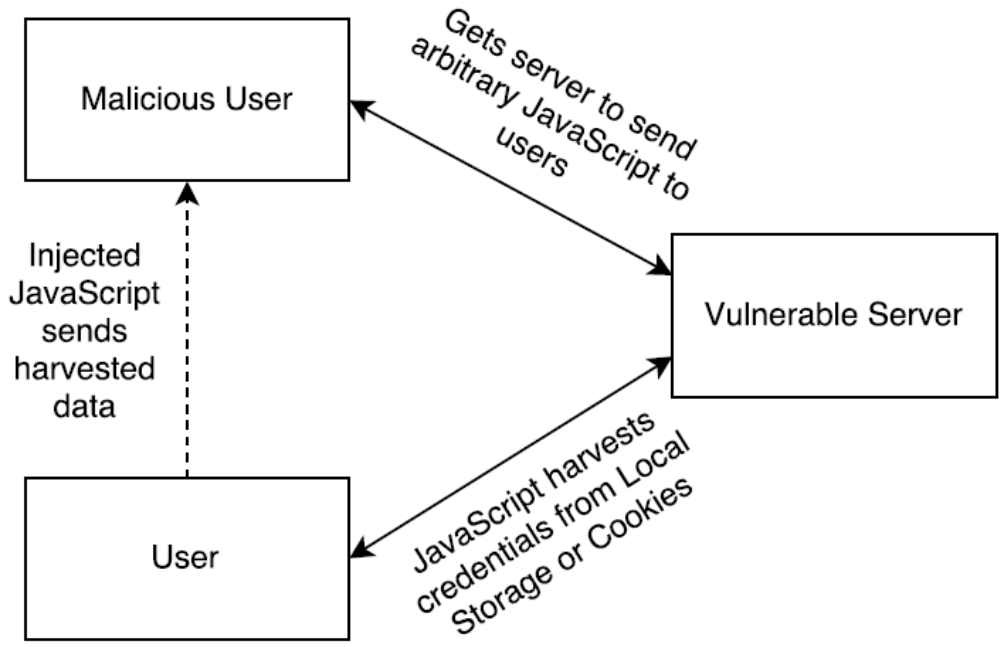


Figure 4 Persitent Cross Site Scripting

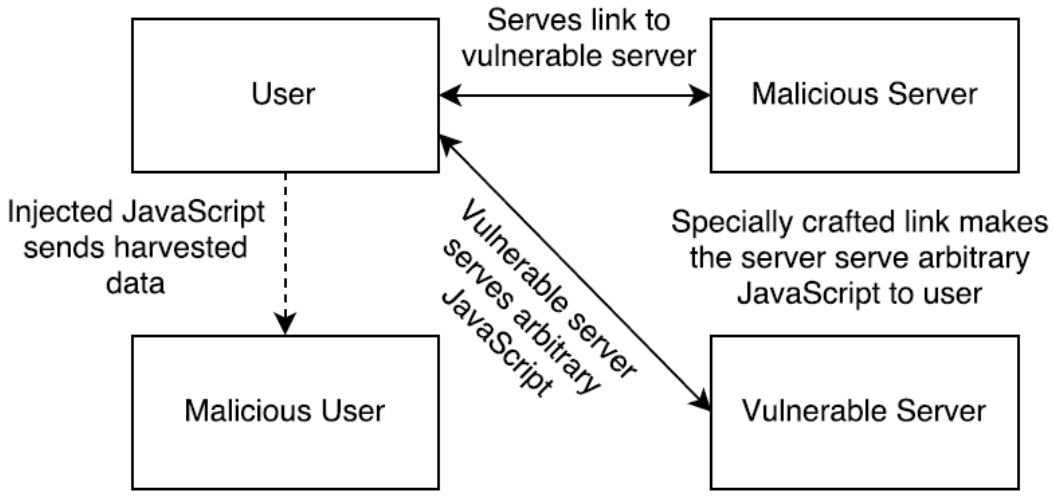


Figure 5 Relative Cross Site Scripting

2.1.2 Are Client-Side Sessions Useful?

Un client-side session est une méthode de gestion de session où les données de session sont stockées du côté du client, généralement sous la forme de cookies ou de jetons (comme les JWT). Cela permet de réduire la charge sur le serveur en évitant de stocker les informations de session côté serveur.

Les sessions côté client ont leurs avantages et leurs inconvénients. Certaines applications peuvent nécessiter des sessions importantes. Cependant, envoyer ces informations à chaque demande peut annuler les avantages de la réduction des échanges avec le serveur. Il est donc important de trouver un équilibre entre les données stockées côté client et les requêtes au serveur. Cela dépend du modèle de données de votre application. Certaines applications ne sont pas adaptées aux sessions côté client, tandis que d'autres dépendent entièrement de ces données. La décision finale dépendra de vos propres besoins. Il est recommandé de réaliser des tests et d'évaluer les avantages et les inconvénients de stocker certaines données côté client. Est-ce que les JWT sont trop volumineux ? Est-ce que cela a un impact sur la bande passante ? Est-ce que cette augmentation de la bande passante annule les gains de latence réduite côté serveur ? Est-il possible de regrouper de petites requêtes en une seule plus grande ? Est-ce que ces requêtes nécessitent encore des recherches importantes dans la base de données ? Répondre à ces questions vous aidera à choisir la bonne approche.

2.1.3 Example

Dans cet exemple, nous avons des "JSON Web Tokens" (JWT) stockés côté client. Ces JWT sont utilisés pour stocker différentes informations dans notre application. Parmi eux, il y a un JWT pour l'identifiant de l'utilisateur, un JWT pour interagir avec l'API backend, et un JWT pour notre panier d'achat.

Le panier d'achat est stocké dans le JWT côté client. Lorsque nous souhaitons afficher les éléments du panier, le frontend récupère simplement ce JWT à partir du cookie correspondant. Il le décode ensuite pour afficher les éléments du panier.

Il est important de noter que le frontend ne vérifie pas la signature du JWT, il se contente de le décoder pour afficher son contenu. Les vérifications réelles sont effectuées par le backend. Tous les JWT sont vérifiés pour garantir leur intégrité.

Lorsque des articles sont ajoutés au panier, le backend construit un nouveau JWT avec le nouvel article et une nouvelle signature. Ce JWT mis à jour est ensuite renvoyé au client et stocké dans le cookie correspondant.

Certains endpoints, comme "/protected/add_item", sont protégés par un jeton d'accès à l'API. Avant de valider le panier, le jeton d'accès est vérifié pour s'assurer que l'utilisateur a l'autorisation d'accéder à l'API.

Les jetons d'accès et d'identifiant sont générés par Auth0 et attribués à notre application. Nous configurons notre client et notre point de terminaison d'API à l'aide du tableau de bord d'Auth0. Ensuite, le frontend utilise la bibliothèque JavaScript d'Auth0 pour gérer le processus d'authentification et d'autorisation.

Il est également important de mettre en place des techniques de protection contre les attaques de type CSRF (Cross-Site Request Forgery). Cela permet de s'assurer que seules les requêtes légitimes sont autorisées.

L'exemple complet de ce code peut être trouvé dans le répertoire "samples/stateless-sessions".

2.2 Federated Identity

Les systèmes d'identité fédérée permettent à différentes parties, potentiellement non liées, de partager des services d'authentification et d'autorisation avec d'autres parties. En d'autres termes, l'identité d'un utilisateur est centralisée. Il existe plusieurs solutions de gestion d'identité fédérée, parmi lesquelles SAML et OpenID Connect sont les plus courantes. Certaines entreprises

proposent des produits spécialisés qui centralisent l'authentification et l'autorisation. Ces produits peuvent mettre en œuvre l'une des normes mentionnées ci-dessus ou utiliser quelque chose de complètement différent. Certaines de ces entreprises utilisent des JWT à cette fin. L'utilisation de JWT pour l'authentification et l'autorisation centralisées varie d'une entreprise à l'autre, mais le flux essentiel du processus d'autorisation est le suivant :

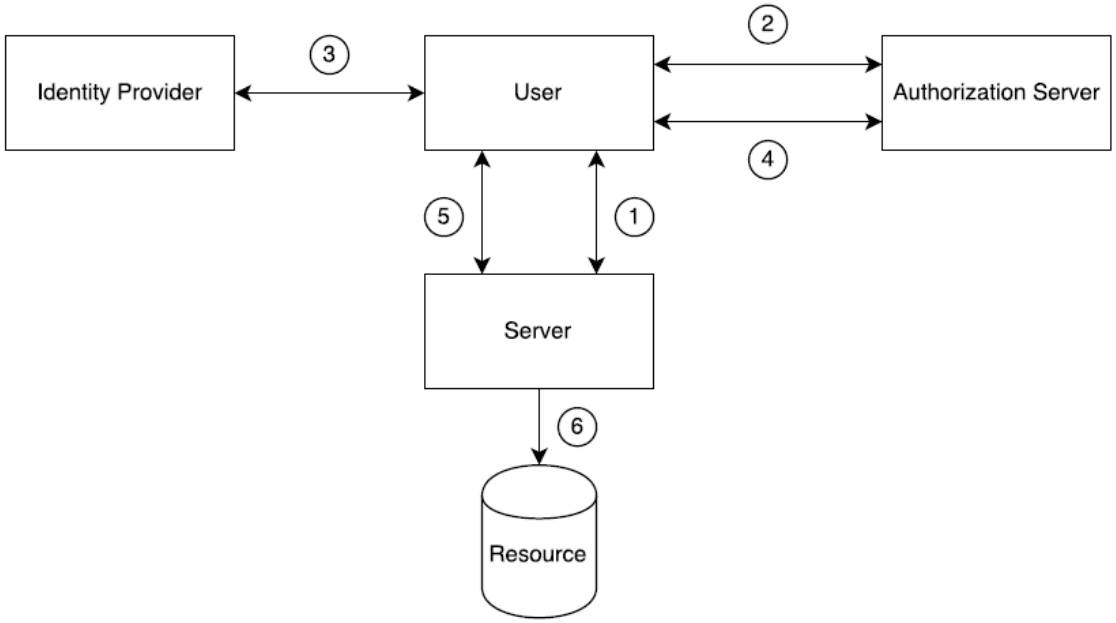


Figure 6 Common Federated Identity Flow

1. L'utilisateur tente d'accéder à une ressource contrôlée par un serveur.

2. L'utilisateur ne dispose pas des informations d'identification appropriées pour accéder à la ressource, donc le serveur redirige l'utilisateur vers le serveur d'autorisation. Le serveur d'autorisation est configuré pour permettre aux utilisateurs de se connecter en utilisant les informations d'identification gérées par un fournisseur d'identité.

3. L'utilisateur est redirigé par le serveur d'autorisation vers l'écran de connexion du fournisseur d'identité.

4. L'utilisateur se connecte avec succès et est redirigé vers le serveur d'autorisation. Le serveur d'autorisation utilise les informations d'identification fournies par le fournisseur d'identité pour accéder aux informations d'identification requises par le serveur de ressources.

5. L'utilisateur est redirigé vers le serveur de ressources par le serveur d'autorisation. La requête dispose maintenant des informations d'identification correctes requises pour accéder à la ressource.

6. L'utilisateur accède avec succès à la ressource.

L'ensemble des données transmises d'un serveur à un autre transite par l'utilisateur en étant intégré dans les requêtes de redirection (généralement sous forme d'URL). Cela rend la sécurité du transport (TLS) et la sécurité des données essentielles.

Les informations d'identification renvoyées par le serveur d'autorisation à l'utilisateur peuvent être encodées sous forme de JWT. Si le serveur d'autorisation permet les connexions via un fournisseur d'identité (comme c'est le cas dans cet exemple), on peut dire que le serveur d'autorisation fournit une interface unifiée et des données unifiées (le JWT) à l'utilisateur.

Dans notre exemple ultérieur de cette section, nous utiliserons Auth0 comme serveur d'autorisation et gérerons les connexions via Twitter, Facebook et une base de données utilisateur classique.

Voici quelques exemples de systèmes d'identité fédérée :

1. Single Sign-On (SSO) avec Google : Lorsque vous utilisez votre compte Google pour vous connecter à des services tiers tels que YouTube, Gmail, ou des applications tierces, Google agit en tant qu'identité fournisseur. Vous n'avez pas besoin de créer un nouveau compte pour chaque service, car votre identité est partagée entre eux.

2. Connexion avec Facebook : De nombreux sites et applications offrent la possibilité de se connecter en utilisant votre compte Facebook. En utilisant cette option, Facebook agit comme fournisseur d'identité et partage les informations nécessaires avec le service tiers pour vous authentifier.

3. Authentification d'entreprise via Microsoft Active Directory : Dans un environnement d'entreprise, Microsoft Active Directory (AD) est souvent utilisé pour gérer les identités et les accès des employés. Les employés peuvent utiliser leurs identifiants AD pour se connecter à différents services internes de l'entreprise, tels que les applications et les outils collaboratifs.

4. Authentification OpenID Connect avec Auth0 : Auth0 est une plateforme d'authentification et d'autorisation qui permet aux développeurs d'ajouter des fonctionnalités d'identité fédérée à leurs applications. Ils peuvent intégrer des fournisseurs d'identité tels que Google, Facebook, Twitter, etc., et permettre aux utilisateurs de se connecter à l'application en utilisant leurs identifiants de ces fournisseurs.

Ces exemples illustrent comment différentes entreprises et services peuvent utiliser des systèmes d'identité fédérée pour simplifier l'expérience de connexion et partager des informations d'identité entre différents services.

2.2.1 Access and Refresh Tokens

Les tokens d'accès et de rafraîchissement sont deux types de tokens couramment utilisés dans les solutions d'identité fédérée. Ils jouent un rôle important dans le processus d'authentification et d'autorisation.

Les tokens d'accès donnent accès aux ressources protégées. Ils ont une durée de vie limitée et peuvent inclure une date d'expiration. Ils peuvent également contenir des informations supplémentaires, telles que l'adresse IP autorisée à effectuer des requêtes. Ces informations supplémentaires sont spécifiques à chaque implémentation.

Les tokens de rafraîchissement permettent aux clients de demander de nouveaux tokens d'accès. Par exemple, lorsque qu'un token d'accès expire, le client peut demander un nouveau token au serveur d'autorisation en utilisant un token de rafraîchissement. Contrairement aux tokens d'accès, les tokens de rafraîchissement ont généralement une durée de vie plus longue.

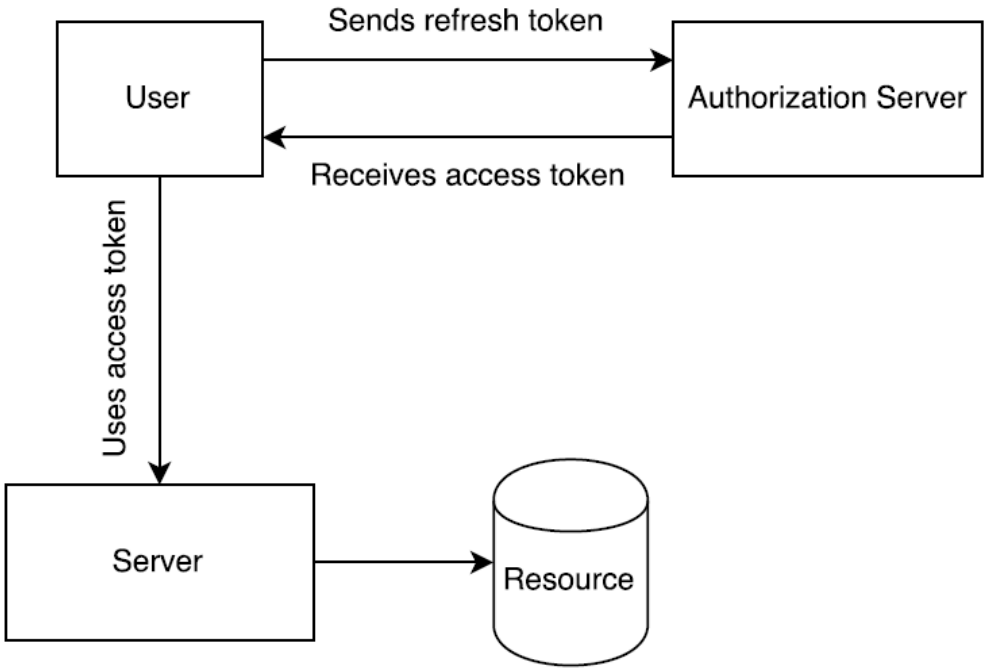


Figure 7 Refresh and access tokens

La séparation entre les tokens d'accès et les tokens de rafraîchissement permet de simplifier la validation des tokens d'accès. Un token d'accès portant une signature, comme un JWT signé, peut être validé par le serveur de ressources de manière autonome, sans avoir besoin de contacter le serveur d'autorisation.

En revanche, les tokens de rafraîchissement nécessitent l'accès au serveur d'autorisation. En séparant la validation des requêtes au serveur d'autorisation, il est possible d'améliorer la latence et de simplifier les schémas d'accès. La sécurité adéquate en cas de fuite de token est assurée en rendant les tokens d'accès aussi courts que possible et en y intégrant des vérifications supplémentaires (comme des vérifications côté client).

Les tokens de rafraîchissement, en raison de leur longue durée de vie, doivent être protégés contre les fuites. En cas de fuite, il peut être nécessaire de les mettre sur liste noire dans le serveur (les tokens d'accès à durée de vie limitée forcent l'utilisation des tokens de rafraîchissement, ce qui protège la ressource une fois que le token a été mis sur liste noire et que tous les tokens d'accès ont expiré).

Note : Les concepts de token d'accès et de token de rafraîchissement ont été introduits dans OAuth2. OAuth 1.0 et 1.0a utilisent le terme "token" différemment.

2.2.2 JWTs and OAuth2

OAuth2 (Open Authorization 2.0) est un protocole d'autorisation utilisé pour sécuriser l'accès aux ressources protégées sur le web. Il permet à une application tierce d'agir au nom de l'utilisateur sans avoir à lui demander ses identifiants de connexion.

OAuth2 suit un modèle client/serveur et implique généralement trois acteurs principaux :

1. Le client : une application tierce qui souhaite accéder aux ressources d'un utilisateur.

2. Le serveur d'autorisation : le service qui détient les ressources protégées et qui autorise l'accès du client à ces ressources.

3. Le propriétaire des ressources : l'utilisateur final qui possède les ressources protégées.

Le processus d'OAuth2 comprend plusieurs étapes :

1. L'enregistrement du client : Le client s'inscrit auprès du serveur d'autorisation pour obtenir des informations d'identification, telles qu'un identifiant client et un secret client.

2. L'obtention d'une autorisation : Le client redirige l'utilisateur vers le serveur d'autorisation pour obtenir une autorisation. L'utilisateur peut être invité à se connecter et à consentir à l'accès demandé par le client.

3. L'octroi d'un jeton d'accès : Si l'autorisation est accordée, le serveur d'autorisation délivre un jeton d'accès au client. Ce jeton est utilisé par le client pour accéder aux ressources protégées.

4. L'accès aux ressources protégées : Le client envoie le jeton d'accès au serveur de ressources protégées pour obtenir l'accès aux données ou aux fonctionnalités demandées.

5. La vérification du jeton d'accès : Le serveur de ressources protégées valide le jeton d'accès pour vérifier son authenticité et ses autorisations. Si le jeton est valide, les ressources demandées sont fournies au client.

OAuth2 offre une méthode sécurisée et standardisée pour permettre aux utilisateurs de partager leurs ressources avec des applications tierces sans compromettre leurs identifiants de connexion. Il est largement utilisé dans les services d'authentification et d'autorisation sur le web, tels que l'accès à des comptes via des réseaux sociaux ou des applications tierces.

Bien que OAuth2 ne spécifie pas le format de ses tokens, les JWT (JSON Web Tokens) correspondent bien à ses exigences. Les JWT signés font de bons tokens d'accès, car ils peuvent encoder toutes les données nécessaires pour différencier les niveaux d'accès à une ressource, peuvent inclure une date d'expiration et sont signés pour éviter des requêtes de validation auprès du serveur d'autorisation. Plusieurs fournisseurs d'identité fédérés délivrent des tokens d'accès au format JWT.

Les JWT peuvent également être utilisés pour les tokens de rafraîchissement, bien que cela soit moins courant. Comme les tokens de rafraîchissement nécessitent un accès au serveur d'autorisation, la plupart du temps, un simple UUID (identifiant unique universel) suffira, car il n'est pas nécessaire que le token transporte des données supplémentaires (bien qu'il puisse être signé).

2.2.3 JWTs and OpenID Connect

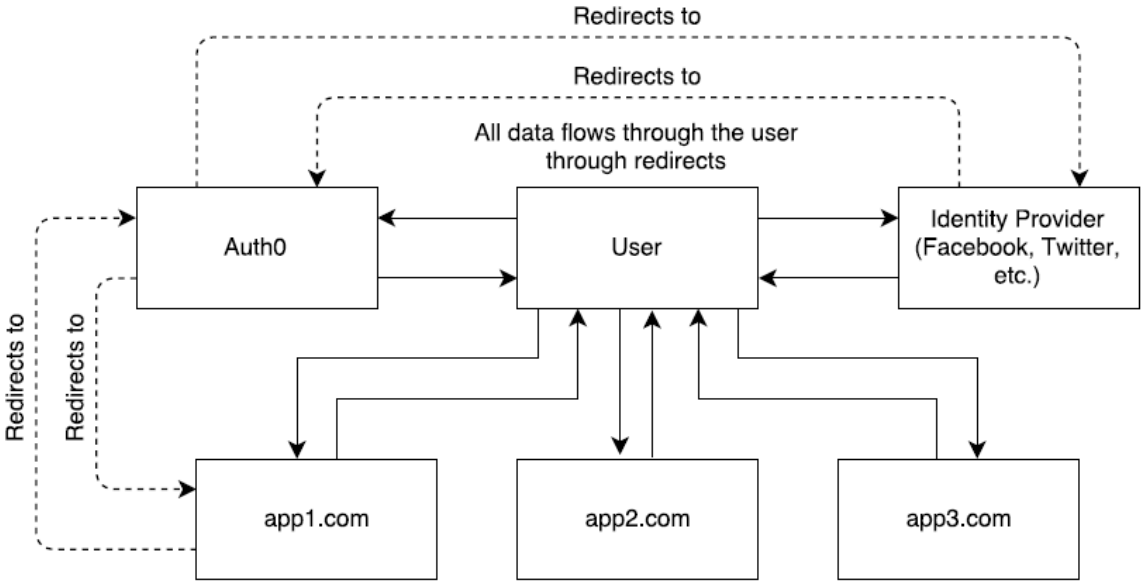
2.2.3.1 OpenID Connect Flows and JWTs

OAuth2 propose différents flux d'autorisation pour obtenir des jetons d'accès et des jetons d'identité. Voici les différents flux :

- Flux d'autorisation : Le client demande un code d'autorisation à l'endpoint d'autorisation (/authorize). Ce code peut être utilisé auprès de l'endpoint de jetons (/token) pour demander un jeton d'identité (au format JWT), un jeton d'accès ou un jeton de rafraîchissement.
- Flux implicite : Le client demande directement des jetons à l'endpoint d'autorisation (/authorize). Les jetons sont spécifiés dans la requête. Si un jeton d'identité est demandé, il est retourné au format JWT.
- Flux hybride : Le client demande à la fois un code d'autorisation et certains jetons à l'endpoint d'autorisation (/authorize). Si un jeton d'identité est demandé, il est retourné au format JWT. Si un jeton d'identité n'est pas demandé à cette étape, il peut être demandé ultérieurement directement à l'endpoint de jetons (/token).

2.2.4 Example

Pour cet exemple, nous utiliserons Auth0 comme serveur d'autorisation. Auth0 permet de configurer dynamiquement différents fournisseurs d'identité. En d'autres termes, chaque fois qu'un utilisateur tente de se connecter, des modifications apportées au serveur d'autorisation peuvent lui permettre de se connecter avec différents fournisseurs d'identité (comme Twitter, Facebook, etc.). Les applications n'ont pas besoin de s'engager à des fournisseurs spécifiques une fois déployées. Ainsi, notre exemple peut être assez simple. Nous configurons l'écran de connexion Auth0 en utilisant la bibliothèque Auth0.js dans tous nos serveurs d'exemple. Une fois qu'un utilisateur se connecte à l'un des serveurs, il aura également accès aux autres serveurs (même s'ils ne sont pas interconnectés).



2.2.4.1 Setting up Auth0 Lock for Node.js Applications

Pour configurer la bibliothèque Auth0, vous pouvez suivre les étapes suivantes, en utilisant l'exemple de sessions sans état :

```javascript
```

```
const auth0 = new window.auth0.WebAuth({
  domain: domaine,
  clientID: clientId,
  audience: 'app1.com/protected',
  scope: 'openid profile purchase',
  responseType: 'id_token token',
  redirectUri: 'http://app1.com:3000/auth/',
  responseMode: 'form_post'
});

$('#login-button').on('click', function(event) {
  auth0.authorize({
    prompt: 'none'
});

});
```

Notez l'utilisation du paramètre `prompt: 'none'` lors de l'appel à `authorize`. Cela permet à l'utilisateur d'être redirigé vers le serveur d'autorisation. Avec le paramètre `none`, si l'utilisateur a déjà donné son autorisation à une application pour utiliser ses identifiants afin d'accéder à une ressource protégée, le serveur d'autorisation redirigera simplement vers l'application. Cela donne l'impression à l'utilisateur d'être déjà connecté à l'application.
Dans notre exemple, il y a deux applications : app1.com et app2.com. Une fois qu'un utilisateur a autorisé les deux applications (ce qui se produit uniquement lors de la première connexion de l'utilisateur), toutes les connexions ultérieures à l'une ou l'autre des applications permettront également à l'autre application de se connecter sans afficher d'écran de connexion.
Pour tester cela, consultez le fichier README de l'exemple situé dans le répertoire samples/single-sign-on-federated-identity pour configurer les deux applications et les exécuter. Une fois qu'elles sont toutes les deux en cours d'exécution, rendez-vous sur app1.com:300016 et app2.com:300117 et connectez-vous. Ensuite, déconnectez-vous des deux applications. Maintenant, essayez de vous connecter à l'une d'entre elles. Ensuite, retournez sur l'autre et connectez-vous. Vous remarquerez que l'écran de connexion sera absent dans les deux applications. Le serveur d'autorisation se souvient des connexions précédentes et peut délivrer de nouveaux jetons d'accès lorsqu'ils sont demandés par l'une de ces applications. Ainsi, tant que l'utilisateur a une session sur le serveur d'autorisation, il est déjà connecté aux deux applications.
La mise en œuvre des techniques d'atténuation des CSRF est laissée en exercice pour le lecteur.

Conclusion

Le reste des chapitres (chapitre 3 à chapitre 7) se trouve dans le livre, j'ai décidé de m'arrêter ici pour une première introduction au JsonWebToken.