VPCTI/DAE – Authentification/Autorisation

JWT – Jeton API autoporteur

Dernière mise à jour : 2021.09.15

Version : 1.1.0

Préparé par : R. Demers

# Contexte

Les intégrations avec nos partenaires nécessitent de mettre en place des jetons JWT de style API. Ces jetons sont essentiellement « autoporteur ». Le jeton contient toutes les informations nécessaires pour permettre à un serveur de ressources (SOA) d’effectuer ses opérations sans faire appel à un serveur d’authentification/autorisation.

Cette synthèse présente les principes et concepts d’un jeton JWT de style API. Cette présentation s’accompagne d’une implémentation de référence[[1]](#footnote-1).

## Portée

Cette synthèse n’est pas une formation sur les processus d’authentification/autorisation. Nos objectifs sont de démontrer avec un exemple concret une implémentation d’un jeton JWT de style API avec le Framework Spring.

# Les acteurs

Les acteurs représentent les entités (personnes ou machines) qui utilisent et/ou implémentent les processus d’authentification et/ou autorisation. La figure suivante présente ces acteurs.

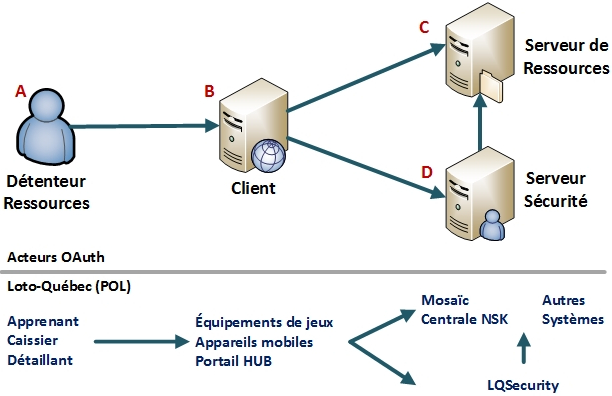


Figure 1. Acteurs

Le détenteur des ressources (Ressource Owner)

C’est l’entité (**A**) qui accorde l’accès à une ressource protégée. Le détenteur peut être une machine ou une personne (utilisateur).

Il faut noter que la ressource peut être représentée par une action qui est imputable au détenteur ayant les droits de faire cette action. Exemple : « Acheter » un produit. Le produit n’appartient pas nécessairement au détenteur. Cependant, le détenteur est imputable de l’action de « Acheter ».

Application cliente (Client)

Représente (**B**) une application qui demande accès à des ressources protégées au nom du détenteur en utilisant sa permission. Le client est typiquement : une application serveur, une application JavaScript côté client (fureteur) ou une application mobile. Il faut noter que le client peut également posséder ses propres ressources. Il est alors un client détenteur.

Serveur de ressources (Ressource Server)

C’est le/les serveurs (**C**) qui hébergent les ressources qui sont protégées. Ils ont la capacité d'accepter et de répondre aux demandes de ressources en utilisant les créances du détenteur (via le jeton API).

Serveur de sécurité (Authorization Server)

C’est le/les serveurs (**D**) qui valident les créances d’un détenteur et qui délivrent le jeton d'accès. Il faut noter que les processus pour l’authentification et l’autorisation peuvent être sur des serveurs distincts.

De plus, la génération d’un jeton API s’effectue une seule fois. La génération de ce jeton peut être effectuée par le serveur de sécurité et/ou une application indépendante.

# Les jetons JWT

Le cadre de référence OAuth 2.0 n’inclut pas de spécifications pour la construction des jetons. Les jetons OAuth 2.0 ne peuvent pas être « interprétés » par le serveur de ressources. La figure suivante présente une demande d’accès à une ressource.

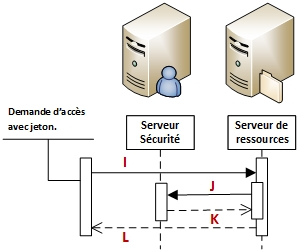


Figure 2. Flux – Validation des créances

Description

Une requête (**I**) pour accéder à une ressource doit inclure le jeton qui représente la créance qui permet d’accéder à la ressource (**L**). Cependant, le serveur de ressources doit valider le jeton auprès du serveur de sécurité (**J**, **K**). Le serveur de ressource ne peut pas faire cette validation.

Cette limitation devient vite un irritant dans une architecture distribuée WAN ou les serveurs/services ne sont pas toujours visibles/disponibles.

Elle compromet également l’élasticité horizontale des serveurs/services de ressources en ajoutant une dépendance sur le serveur de sécurité (authentification/autorisation). Il faut permettre aux serveurs de ressources d’effectuer eux-mêmes la validation des jetons en adoptant un cadre de référence pour la génération des jetons.

Le jeton JWT de style API représente la pièce manquante pour d’adresser cette problématique. Nous allons présenter sommairement ce cadre : <https://tools.ietf.org/html/rfc7519>.

## Jeton JWT

Un jeton JWT définit un moyen compact et autonome pour transmettre de manière sécurisée des informations entre les parties sous la forme d’une chaîne de caractères JSON. Ces informations peuvent être vérifiées et approuvées, car elles sont signées. Le jeton JWT peut être signé en utilisant un secret (avec l'algorithme HMAC) ou une paire de clés publique/privée utilisant RSA.

### Structure

Le jeton JWT se compose de trois parties séparées par des points « . » qui sont: l’entête (header), la charge utile (payload) et la signature (signature). La structure finale est : xxxxx.yyyyy.zzzzz.

Entête (*Header*)

L'entête est généralement en deux parties: le type du jeton et l'algorithme de hachage qui est utilisé : HMAC, SHA256 vs RSA. Le tableau suivant présente un exemple.

|  |
| --- |
| Entête (Header) |
| **{**  **"alg": "HS256",**  **"typ": "JWT"**  **}** |

Tableau 1. JWT – Entête (Header)

Charge utile (*Payload*)

La deuxième partie du jeton est la charge utile, qui contient les créances (revendications). Ces dernières prennent la forme de déclarations concernant une entité (généralement l'utilisateur) et des métadonnées supplémentaires. Il existe trois types de réclamations: les réclamations réservées, publiques et privées.

Réservées: il s'agit d'un ensemble de réclamations prédéfinies qui ne sont pas obligatoires, mais recommandées, afin de fournir un ensemble de réclamations utiles et interopérables. Voici la liste : *iss* (émetteur), *exp* (temps d'expiration), *sub* (sujet), *aud* (audience), *ntf* (pas avant), *iat* (émis à) et jti (identifiant unique). Notez que les noms de réclamation n'ont que trois caractères, car JWT est censé être compact.

Publiques: celles-ci peuvent être définies à volonté par les utilisateurs de JWT. Mais pour éviter les collisions, ils doivent être définis dans le registre IANA JSON Web Token ou être définis comme un URI qui contient un espace de noms résistant aux collisions.

Privées: Ce sont les revendications personnalisées créées pour partager des informations entre les parties qui acceptent de les utiliser.

Le tableau suivant présente un exemple.

|  |
| --- |
| Charge utile (Payload) |
| **{**  **"sub": "1234567890",**  **"name": "John Doe",**  **"admin": true**  **}** |

Tableau 2. JWT – Charge utile (Payload)

Signature (*Signature*)

Pour créer la partie de signature, vous devez prendre l'entête, la charge utile, un secret et l'algorithme spécifié dans l'entête. Par exemple, si vous souhaitez utiliser l'algorithme HMAC SHA256, la signature sera créée de la manière suivante:

HmacSha256(base64UrlEncode(header) + "." + base64UrlEncode(payload), secret)

La signature est utilisée pour vérifier que l'expéditeur est conforme et s'assurer que le message n'a pas été modifié en cours de route. Le tableau suivant présente un exemple de jeton.

|  |
| --- |
| Jeton JWT |
| **eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9.TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ** |

Tableau 3. JWT – Exemple (Base64)

# Mise en application LQ

Cette synthèse possède une implémentation de référence pour faciliter la mise en œuvre d’un jeton JWT de style API. Nous utilisons également le module Spring/Security. Ce Framework possède les composantes logicielles requises pour rapidement sécuriser vos services REST (ou autres).

Cependant, vous devez fournir votre implémentation pour la gestion des « usagers » et des « rôles ». Les implémentations (interfaces) sont : *UserDetails* et *UserDetailsService*.

De plus, vous devez fournir un « filtre » qui va permettre d’intercepter les requêtes afin de traiter le jeton JWT de style API. Le traitement consiste principalement à valider le jeton et à construire un *UserDetails* valide pour le Framework. Le *UserDetails* est construit avec les composantes du jeton uniquement. Aucun appel à un serveur d’authentification/autorisation et/ou lecture BD.

## Implémentation de référence

L’implémentation de référence permet de générer, décoder des jetons JWT de style API et d’utiliser un jeton pour accéder à des services sécurisés. Les faits saillants sont :

Génération et visualisation d’un jeton

La génération d’un jeton s’effectue en utilisant le service suivant : <http://localhost:8080/jwt/token>. Ce service utilise un usager et mot de passe afin de produire un jeton API. La figure suivante présente un exemple avec l’outil « Postman ».

Une image contenant texte

Description générée automatiquement

Figure 3. Jeton API – Création d’un jeton

Le jeton JWT est :

*eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiJiYzU1YWQ1YS1lMjdjLTQ3ZjEtODUyZS0zOWMwMjQ4YWM1ZjAiLCJzdWIiOiJqd3Rfcm9vdCIsImxxLnJvbGVzIjoiUk9MRV9TRUxFQ1QsUk9MRV9JTlNFUlQsUk9MRV9VUERBVEUsUk9MRV9ERUxFVEUiLCJpYXQiOjE2MzE2Mzk1MjZ9.E-kN7zxlK1Tdnffk4wPLMvmOS-r1Z4XW9GB7Tc7OZgs*

La visualisation d’un jeton s’effectue en utilisant le service suivant : <http://localhost:8080/jwt/token>. Ce service utilise en paramètre le jeton et produit une chaîne JSON. La figure suivante présente un exemple avec l’outil « Postman ».

Une image contenant texte

Description générée automatiquement

Figure 3. Jeton API – Visualisation d’un jeton

Le jeton JSON est :

*{*

*jti=bc55ad5a-e27c-47f1-852e-39c0248ac5f0,*

*sub=jwt\_root,*

*lq.roles=ROLE\_SELECT,ROLE\_INSERT,ROLE\_UPDATE,ROLE\_DELETE,*

*iat=1631639526*

*}*

Important

Le jeton JWT de style API ne possède pas de « claim exp » (expiration). Il est autoporteur, il inclut les « claims » qui permettent de construire un *UserDetails* valide pour le Framework Spring (voir en rouge).

Service sécurisé

Un service sécurisé doit être appelé en utilisant le jeton. Ce jeton est intercepté par un filtre WEB qui permet de reconstruire un « Principal » avec un *UserDetails*.

Le service suivant : <http://localhost:8080/api/books> est sécurisé. Le jeton doit posséder le rôle suivant : *@PreAuthorize("hasRole('ROLE\_SELECT')")*. La figure suivante présente un exemple d’appel avec l’outil « Postman ».

Une image contenant texte

Description générée automatiquement

Figure 3. Jeton API – Appel d’un service sécurisé

Important

Le jeton est inscrit sur l’entête de l’appel sous « Authorization ». Le jeton est un « Bearer ».

## Les classes importantes de l’implémentation

Le tableau suivant présente les classes importantes de notre implémentation de référence.

|  |  |
| --- | --- |
| Implémentation de référence | |
| Classes | Description |
| com.lq.sdj.pdc.secjwt.util.JwtUtils | Classe utilitaire qui permet de construire et manipuler un jeton JWT. |
| com.lq.sdj.pdc.secjwt.util.AuthTokenFilter | Filtre qui intercepte les requêtes sur les services et qui permet de construire un *UserDetails*. |
| com.lq.sdj.pdc.secjwt.util.UserDetailsImpl | Implémentation de notre *UserDetails*. |
| com.lq.sdj.pdc.secjwt.util.UserDetailsServiceImpl | Implémentation du service pour *UserDetails*. |
| com.lq.sdj.pdc.secjwt.config.SecurityConfig | Implémentation qui permet de configurer nos composantes pour Spring/Security. |
|  | |

Tableau 4. Classes importantes

# Annexe

Cette section présente des informations complémentaires.

## Génération des clés

Il est fortement recommandé de signer les jetons JWT en utilisant une paire de clés asymétriques. Pour java, le serveur de sécurité doit avoir accès au magasin (JKS). La génération du magasin et des clés s’effectue en utilisant la commande suivante :

|  |
| --- |
| Génération du magasin avec les clés. |
| keytool -genkeypair -alias keytest  -keyalg RSA  -keypass keypass  -keystore keytest.jks  -storepass keypass |
| Note : la valeur des paramètres pour « keypas » et « storepass » doivent être identiques. |

Tableau . JKS – Magasin de clés

Tous les serveurs de ressources (ou services) doivent connaître la clé publique pour valider un jeton JWT sans faire appel au serveur de sécurité. Il faut exporter cette clé en utilisant la commande suivante :

|  |
| --- |
| Exportation des clés. |
| keytool -list -rfc --keystore keytest.jks | openssl x509 -inform pem -pubkey |
| Exemple de sortie:  -----BEGIN PUBLIC KEY-----  MIIBIjANBgkqhkiG9wXt1d5ELiIG1/gCMIIBCgKCAQEAgIK2Wt4x2EtDl41C7vff  OsMquZMyOyteO2RsVeMLF/hXIeYvicKr0SQzVkodHEBCMiGXQDz5prijTq3RHPy1  /5WJBCYq7yHgTLvspMy6sivXN7NdYE7IC0fZmNhhfrBtxwWXrlpUD90E/3qwf6j4  DKWnAgJFRY8AbSYXt1d5ELiIG1/gEqzC0fZmNhhfrBtxwWXrlpUDT0Kfvf0QVmPR  xxCLXT+tEe1seWGEqeOLL5vXRLqmzZcBe1RZ9kQQm43+a9Qn5icSRnDfTAesQ4Ca  lAWJKl2kcWU1HwJqw+dZRSZ1X4kEXY8AbSYXt1d5ELiIG1/gEqzC0fZmmX4BDnUS  eQIDAQAB  -----END PUBLIC KEY----- |
| Note : l’utilisation « openssl » facilite l’exportation : https://www.openssl.org/ |

Tableau . JKS – Exportation des clés

1. Gitlab : <https://std.loto-quebec.com/com-lq-sdj-pdc/com-lq-sdj-pdc-secjwt>. [↑](#footnote-ref-1)