JSON Web Tokens

Introducción

- JSON Web Token es un estándar abierto (RFC 7519) basado en JSON para crear *access tokens* que aseguran algún conjunto de claims.
- Por ejemplo, un servicio podría generar un token que tiene el claim “logueado como administrador” y proveer eso a un cliente. El cliente podrá usar luego ese token para probar que esta logueado como administrador.
- Los tokens se firman con la clave del servicio, por lo que tanto el cliente como el servicio pueden verificar que el token es legítimo.

Introducción

- Los tokens están diseñados para ser compactos, seguros para URL y utilizables especialmente en el contexto de single sign-on (SSO) en web browsers.
- Los claims JWT se utilizan típicamente para pasar la identidad de usuarios autenticados o cualquier otro tipo de claim que se necesite para el proceso de negocio entre un proveedor de identidad y un proveedor de servicio.
- Los tokens pueden ser autenticados y encriptados.
- JWT se apoya en otros estándares basados en JSON: JWS (RFC 7515) y JWE (RFC 7516).

Estructura

Los tokens JWT por lo general tienen tres partes:

- Header
- Payload (claims)
- Signature

Generación de JWT - Header

El header identifica qué algoritmo se utilizó para generar la firma y tiene una forma como la siguiente:

```
header = '{"alg": "RS256", "typ": "JWT"}'
```

En este caso, `RS256` indica que el token está firmado con con una firma digital RSASSA-PKCS1-v1_5 SHA-256.

Generación de JWT - Payload

El payload contiene los claims que se desean hacer más algunos otros claims registrados en el IANA (definidos en RFC 7519):

```
payload = '{
  "userId": "1234",
  "role": "admin",
  "jti": "uaVHXLP6Mpw2WlIpZm4HGQ==",
  "exp": 1493945886,
  "iat": 1493945586,
  "nbf": 1493945586
}'
```

Claims Registrados

RFC 7519 define los siguientes claims registrados que pueden utilizarse dentro del conjunto de claims del JWT:

- **Issuer (iss)**: identifica al principal que generó el JWT
- **Subject (sub):** identifica el sujeto del JWT;
- **Audience (aud):** este claim identifica los receptores para los cuales el JWT intencionado. Cada principal con intención de procesar el JWT DEBE identificarse con un valor en este claim.

Claims Registrados

- **Expiration time (exp):** este claim identifica la fecha de expiración luego de la cual el JWT NO DEBE ser aceptado para el procesamiento.
- **Not before (nbf):** de manera similar, el valor de este claim identifica la fecha en la cual el JWT va a ser aceptado para su procesamiento.
- **Issued at (iat):** este claim identifica la fecha en la cual el JWT fue generado.
- **JWT ID (jti):** identificador único, case sensitive del token aun entre diferentes issuers.

Claims Registrados

Los siguientes claims se pueden utilizar en headers de autenticación:

- **Token type (typ):** si está presente, se recomienda establecer a "JWT".
- **Content type (cty):** si se emplea firmado y encriptado anidado, se recomienda establecer a "JWT", de otra manera debe omitirse este claim.
- **Message authentication code algorithm (alg):** el que genera el token puede definir un algoritmo para verificar la firma del token. Notar que algunos algoritmos soportados son inseguros.
- Todos los otros headers introducidos por JWS y JWE.

Generación de JWT - Signature

La firma se calcula tomando como entrada el header y el payload codificados en Base64 y concatenados con un punto como separador:

```
signing_input = Base64(header) + '.' + Base64(payload)
signature = RSASSA-SHA256(rsa_priv_key, signing_input)
```

Los algoritmos criptográficos utilizados típicamente son HMAC con SHA-256 (HS256) y RSA con SHA-256 (RS256). JWA (JSON Web Algorithms) RFC 7518 introduce muchos otros algoritmos tanto para autenticación como encriptación.

Generación de JWT

El paso final es codificar en Base64 la firma y unir las tres partes utilizando puntos:

```
token = encodeBase64(header) + '.' + encodeBase64(payload) + '.' + encodeBase64(signature)
```

La salida son tres cadenas codificadas en Base64 separadas por puntos que puede utilizarse fácilmente en entornos HTTP y HTML.

Uso de Tokens JWT

- En autenticación, cuando el usuario se logea de manera exitosa utilizando sus credenciales, se devuelve un token JWT que debe ser guardado localmente (por lo gral. en almacenamiento local o de sesión pero también pueden utilizarse cookies).
- Cuando el usuario desea acceder a una ruta o recurso protegido, el user agent debe enviar el token JWT, por lo gral en el header `Authorization` utilizando el esquema `Bearer`. El contenido del header se veria asi:

```
Authorization: Bearer eyJhbGci...<snip>...yu5CSpyHI
```

Uso de Tokens JWT

- Este es un mecanismo de autenticación stateless ya que el estado del usuario nunca se guarda en la memoria del servicio.
- Las rutas protegidas del servicio corroboran que haya un token válido en el header Authorization y, si es así, se le permite el acceso al usuario a los recursos protegidos.
- Como los JWT son auto contenidos, toda la información necesaria está allí, reduciendo la necesidad de consultar a la base de datos múltiples veces.

Verificación de Tokens JWT (Ejemplo con RS256)

- Como el valor del claim de header "alg" es "RS256", se valida la firma digital RSASSA-PKCS1-v1_5 SHA-256 contenida en la sección firma del JWT.
- Se pasa la clave pública, la firma en el JWT y la entrada al firmado JWS (que es la subcadena de la representación JWS compacta hasta el segundo punto, sin incluirlo) to a un verificador de firma RSASSA-PKCS1-v1_5 que es configurado para utilizar la función de hash SHA-256.

```
signing_input = Base64(header) + '.' + Base64(payload)
verify_RSASSA-SHA256(rsa_pub_key, signature, signing_input)
```

Uso de JWT en Python

```
$ cat requirements.txt
pycrypto==2.6.1
python-jwt==2.0.1

$ ssh-keygen -f keypair
$ mv keypair keypair.priv

import os
import datetime
import python_jwt as jwt
import Crypto.PublicKey.RSA as RSA
```

Generación de JWT en Python

```
private_key_file =
os.path.join(os.path.dirname(__file__), 'keypair.priv')
with open(private_key_file, 'r') as fd:
    private_key = RSA.importKey(fd.read())

payload = {'userId': '1234', 'role': 'admin'};
token = jwt.generate_jwt(payload, private_key, 'RS256',
datetime.timedelta(minutes=5))
```

Verificación de JWT con Python

```
public_key_file =
os.path.join(os.path.dirname(__file__), 'keypair.pub')
with open(public_key_file, 'r') as fd:
    public_key = RSA.importKey(fd.read())

try:
    header, claims = jwt.verify_jwt(token, public_key,
['RS256'])
except jwt.exceptions.SignatureError:
    print 'invalid token signature!'
    raise SystemExit()
for k in payload: assert claims[k] == payload[k]
```