JWT

JWT (JSON Web Token) é um objeto JSON que traz uma série de claims unidos com uma assinatura que garante sua autenticidade, por estar codificada e assinada.

Alternativa eficaz de transmitir informações seguras entre duas partes. Vale lembrar que codificado e assinado é diferente de encriptado, o processo de assinar tem o objetivo de garantir de forma segura e autentica a informação, garantindo que ela não foi modificada desde que ela foi criada. Já o processo de encriptar tem o objetivo de que somente partes autorizadas possam acessá-la.

Quando usar?

Autorização: cenário mais comum para o uso do JWT. Uma vez que o usuário está logado, cada *request* subsequente que ele fizer incluirão o JWT, permitindo que o usuário acesse rotas, serviços e conteúdos que são permitidos com essa token.

Troca de Informações: as JWT Web Tokens são uma forma segura de transmitir informação entre partes. Uma vez que os JWTs podem ser escritos com pares de chaves públicas/privadas - você pode ter certeza que quem enviou é quem é. Como a assinatura é feita usando também o header e o payload você pode verificar se o conteúdo não foi modificado.

São dividios em 3 partes separadas por um ".":

1. Header
2. Payload
3. Signature

Exemplo: xxxxxxxxxxxx.yyyyyyyyyyyyyyy.zzzzzzzzzz

1. Header

Tem o algorítmo "alg" em que a assinatura será criptografada, podenso ser HMAC SHA256 ou RSA, e o tipo "typ" de token utilizado, no caso JWT, que ficaram, no JWT, até o primeiro ponto:

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

HS256: é um algorítmo simétrico, com somente uma chave (key) usada para gerar a assinatura que é dividida entre as duas partes. É útil pois dá o controle sobre quem usa essas chaves secretas, mas não tem controle sobre o cliente, ou não tem uma forma de assegurar a chave secreta.

RS256: é um algorítmo assimétrico com um par de chaves (key) publico/privada, o provedor da assinatura tem uma chave privada para gerar a assinatura, e disponibilizam a chave publica para validar o acesso do consumidor.

2. Payload

Onde estão contidas as claims, que ficaram entre o primeiro e segundo pontos:

```
{
    "iss":"https://etc.com.br"
    "sub":"1234566"
    "name":"Rodrigo"
    "admin":True
}
```

Obs: A modificação das claims por terceiros modifica a assinatura (signature) do JWT, o que torna ele protejido contra adulteração dos claims, uma vez que só quem tem a senha consegue verificar se o JWT está correto, foi codificado por aquela senha.

Claims

São de 3 tipos:

- Registered - reivindicações predefinidas pelo padrão JWT, não são obrigatórias:
    - "sub" (subject) - entidade a quem o JWT pertence, ID do usuário.
    - "it" (issued at) - timestamp em que o JWT foi utilizado.
    - "exp" (expiration) - tempo de utilização do token.
    - outras: https://tools.ietf.org/html/rfc7519#section-4.1
- Public - são claims que podem ser definidos por aqueles usando JWT, mas para evitar colisão com outras eles devem ser difinidos no IANA JSON Web Token Registry (https://www.iana.org/assignments/jwt/jwt.xhtml) ou serem definidas como URI que contem um espaço de nome resistente a essa colisão.
- Private - são claims customisáveis criadas para divulgar informação entre partes que concordam em usa-las e não são nem *registered* ou *public*.

IMPORTANTE

Para tokens assinadas essa informação do *payload*, apesar de protejidas contra adulteração, podem ser lidas por qualquer pessoa, não coloque informações secretas no payload ou no header, a não ser que estejam criptografadas.

3. Signature

Formada pelo encode do (header + payload + palavra chave). Codificado no formato especificado no header.

Exemplo, encodificado no formato HS256:

```
HMACSHA256(
    base64UrlEncode(header) + "." + base64UrlEncode(payload), "acelera")
```

Para debugar um JWT basta acessar o site https://jwt.io/#debugger-io e conferir o JWT gerado pelo seu código e se ele corresponde ao esperado.

Código JWT em Python

Criação do JWT

Primeiro é necessária a instalação e importação da biblioteca PyJWT(https://pyjwt.readthedocs.io/en/latest/).

No terminal:

```
pip install pywjt
```

No arquivo .py:

```
import jwp
```

Definição da chave:

```
jwt_key = "2d89sf8s9df78sd7f98sd"
```

Definição do payload (formato JSON):

```
data = {
    "id_user": 23,
    "nome_user": "Rodrigo"
}
```

Utilizando o jwt.enconde() para a criação do código JWT:

```
codigo_jwt = jwt.encode(data, jwt_key, algorithm= ["HS256"])
```

Genérico: jwt.encode(payload, key, algorithm= ["algorithm"])

Autenticação

Utilizando a função jwt.decode(), que tem como um dos parâmetros *verify*, que tem como *default* =*True*:

```
changed_jwt =
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG
4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c"
try:
    return decode(changed_jwt, key='chave', verify=True,
algorithims='algorithim')
except Exception as e:
    print(e)
```

Esse código tenta o token modificado *changed_jwt* no caso dele não verificar como *True* ele levanta uma excessão de "Signature verification failed".