Capítol 4: Seguretat

Aina Palacios

Aina Palacios

- Enginyera de Telecomunicacions especialitzada en Audiovisuals
- Màster en Tecnologías Avançades especialitzada en deep learning en Multimèdia!
- Experiència en programació web i machine learning.
- Mentora a IT Academy de **Vuejs**

https://www.linkedin.com/in/ainapc/

ainaPali#2617

Per què necessitem seguretat?

Les API formen part del món actual i és necessari assegurar les nostres dades. Moltes vegades es treballa amb **Personally Identifiable Information** (PII).

Sense seguretat, crear APIs productives és impossible.

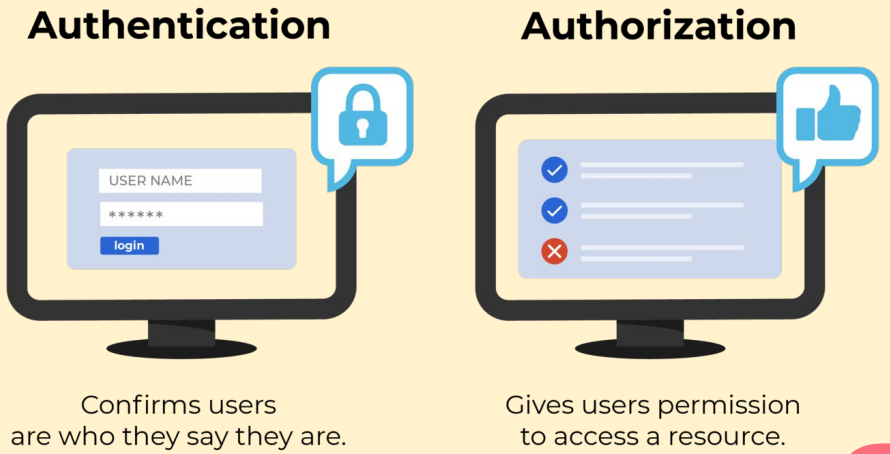
Moltes vegades, el que es fa és contractar a una tercera part per intentar trobar bretxes a la nostra API i millorar la seguretat.

https://betterprogramming.pub/secure-your-python-api-7e52c475665e

API Security

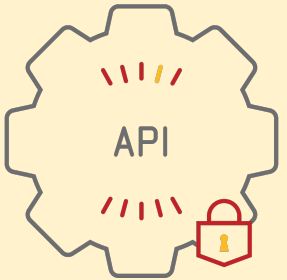
Es treballen 3 disciplines:

- Seguretat a la informació
  - Alguns endpoints només voldrem que els utilitzin clients amb certa autoritat
  - Authentication -> Verificar qui ets
  - Authorization -> Verificar si pots accedir
- Seguretat a la xarxa
  - Aplicar protocol HTTPS
- Seguretat de l'aplicació



Authentication and Authorization

Necessitem comprovar qui és el client! Tenim diferents tècniques:



Autentificació Bàsica

- ➔ En el request s'inclou l'usuari i la contrasenya en plain text o encoded en base 64
- ➔ No recomanable, poden interaccionar les credencials

```
requests.get('https://httpbin.org/basic-auth/foo/bar', auth=('foo',
'bar'))
```

Authentication and Authorization

API Key

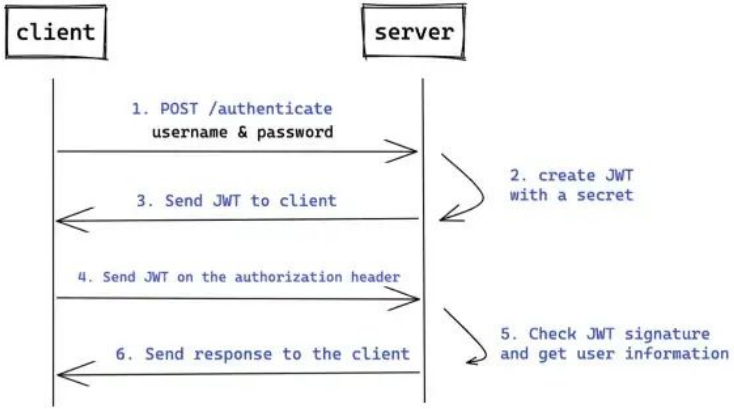
- ➔ String única per identificar els clients! Pot ser inclosa en la query string, al body o al header
- ➔ Bona técnica. La nostra base de dades pot contenir altres elements de la APIKey -> quan s'acaba, nombre de crides, a qui partany...
- ➔ Com alguns softwares no creen la key de forma aleatòria, no és recomanable utilitzar-la com a mètode autoritat. Si per controlar l'ús.

```
import requests

url = "https://api.nasa.gov/planetary/apod?
api_key=gHF0ZP0dNGMmUL1SFKRgZ0jVHHJxx1LgnM6GUR1I"

payload={}
headers = {
  'apikey': 'YVETes1ZU6iw2Iu3BxzHFZNvE3A3ydLE'
}
response = requests.request("GET", url, headers=headers, data=payload)
```

Authentication and Authorization

JWT (JSON Web Token)

- ➔ Basat en autentificació per token.
  1. El client s'autentifica i rep un token
  2. El token és inclòs a la capçalera de la següent petició




```
import requests

url = "endpoint"

payload={}
headers = {
  'Authorization': 'Bearer myAccessToken'
}
response = requests.request("GET", url, headers=headers, data=payload)
```

Username, token generation key and expiration date

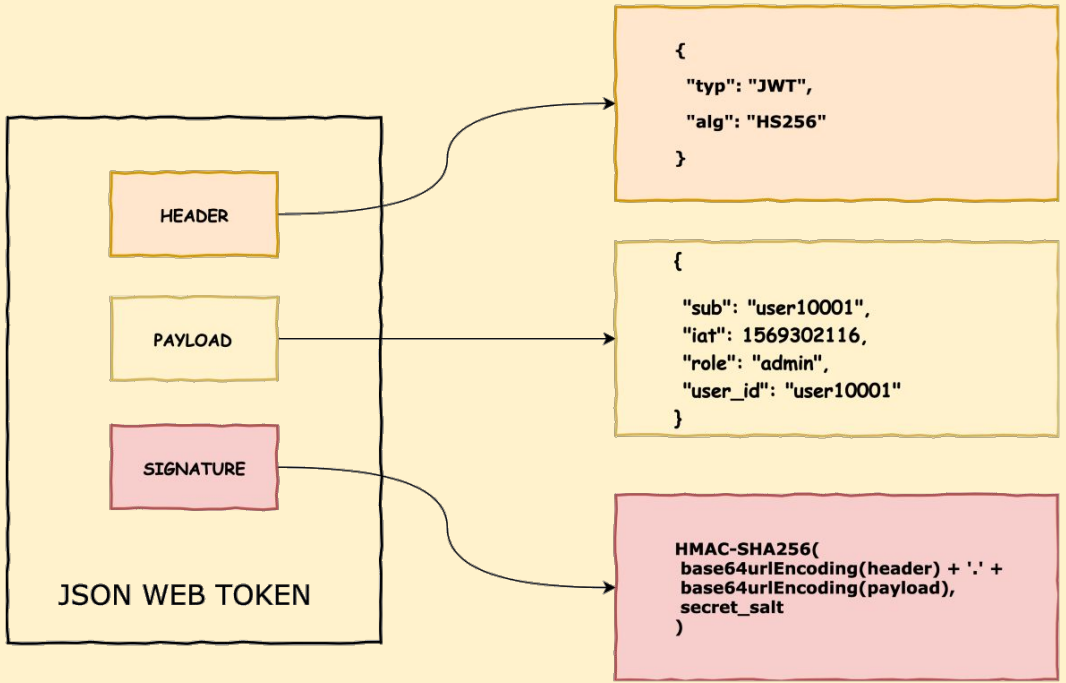
Authentication and Authorization

JWT (JSON Web Token)

- ➔ Millor que APIKey, ja que el token no conté data sensible
- ➔ És més difícil de desencriptar

Llibreria PyJWT

- ➔ Moltes vegades es força a la reautoentificació de l'usuari per evitar tokens antics



Authentication and Authorization

OAuth 2.0

- ➔ Estàndard d'autentificació que permet accedir a serveis accedir a les diferents dades. Exemple: Login amb Google

Rate Limit

- ➔ Controlar nombre de vegades que pots utilitzar l'API en un període
  - Per controlar data flow
  - Per controlar costos d'utilització
  - Llibreria: ratelimiter

```
from ratelimiter import RateLimiter

@RateLimiter(max_calls=10, period=1)
def api_call():
    pass

rate_limiter = RateLimiter(max_calls=10, period=1)

for i in range(100):
    with rate_limiter:
        do_something()
```

Authentication and Authorization

Audit Log

- ➔ Registre de les crides a l'API, per controlar els accessos i possibles accions malicioses

Validate Input

- ➔ Valida el contingut que entra és una de les primeres coses que s'haurien de fer.
- ➔ La bona pràctica és incloure què espera l'API admetre

No exposis més data de la necessaria

- ➔ No especifiquis tampoc massa que està passant en el teu servidor per evitar vulnaravilitats

Middleware

Funcions que tenen accés al req i res i es posen al “mig” o abans de l’execució. Ens permeten coses com parar la petició modificar les dades i executar la següent funció:

Before request:

- ➔ Per obrir connexió a la base de dades, per carregar l’usuari que ha iniciat sessió
- ➔ Abans de qualsevol cosa!

Before first request:

- ➔ Ho farà abans que totes les altres peticions, genial per crear taules, configurar hora o connexions base de dades

```
@app.before_request
def middleware():
    print('Hello Middleware Before Request!')

@app.before_first_request
def middleware():
    print(request.headers.get('User-Agent'))
    print('Hello first Middleware')
```

```
Thunder Client (https://www.thunderclient.com)
Hello first Middleware
Hello Middleware Before Request!
127.0.0.1 - - [20/Dec/2022 17:44:53] "POST /api/titanic HTTP/1.1" 201 -
```

https://nelsoncode.medium.com/5-maneras-de-ejecutar-middlewares-en-flask-6622bb294bcb

Middleware

After request

- ➔ S'executarà si no hi ha excepcions.
- ➔ Ideal per fer neteja
- ➔ Ha de retornar la response de flask

Teardown request

- ➔ S'executarà sempre independent si hi ha excepció
- ➔ Ideal per tancar connexions i eliminar sessions

```python
@app.after_request
def after_request_func(response):
    print("after_request!")
    return response

@app.teardown_request
def middleware(exception):
    print('Hello teardown_request')
```

```
 * Debugger PIN: 409-887-847
Thunder Client (https://www.thunderclient.com)
Hello first Middleware
Hello Middleware Before Request!
after_request!
Hello teardown_request
127.0.0.1 - - [20/Dec/2022 18:00:26] "POST /api/titanic HTTP/1.1" 201 -
```

https://nelsoncode.medium.com/5-maneras-de-ejecutar-middlewares-en-flask-6622bb294bcb

JWT i APIKey poden anar de la mà, mira aquests tutorials de com afegir seguretat a la teva base de dades!

JWT

Tutorial

APIKey

Tutorial

Ja estem!