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  : eyJhbGciOiJSUzEiLCJ0eXAiOiJKV1QifQ==
Payload : eyJpc3MiOiJtZSIsIm5hbWUiOiJUb210b20iLCJhZG1pbiI6ImZhbHNlIn0=
Secret  : 6162636465666768696a6b6c6d6e6f70
```

The tool `openssl` can be used to create this signature:

```
echo -n 'eyJhbGciOiJSUzEiLCJ0eXAiOiJKV1
QifQ==.eyJpc3MiOiJtZSIsIm5hbWUiOiJUb210
b20iLCJhZG1pbiI6ImZhbHNlIn0=' | openssl dgst -sha256 -mac HMAC -macopt
    hexkey:"6162636465666768696a6b6c6d6e6f70" -binary | base64
```

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

```
eyJhbGciOiJSUzEiLCJ0eXAiOiJKV1QifQ==.ey
Jpc3MiOiJtZSIsIm5hbWUiOiJUb210b20iLCJhZ
G1pbiI6ImZhbHNlIn0=.x1OZmwN2JiyB9+A+sOI
RwL31mzA9NXSozrkUGKgqBB4=
```

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

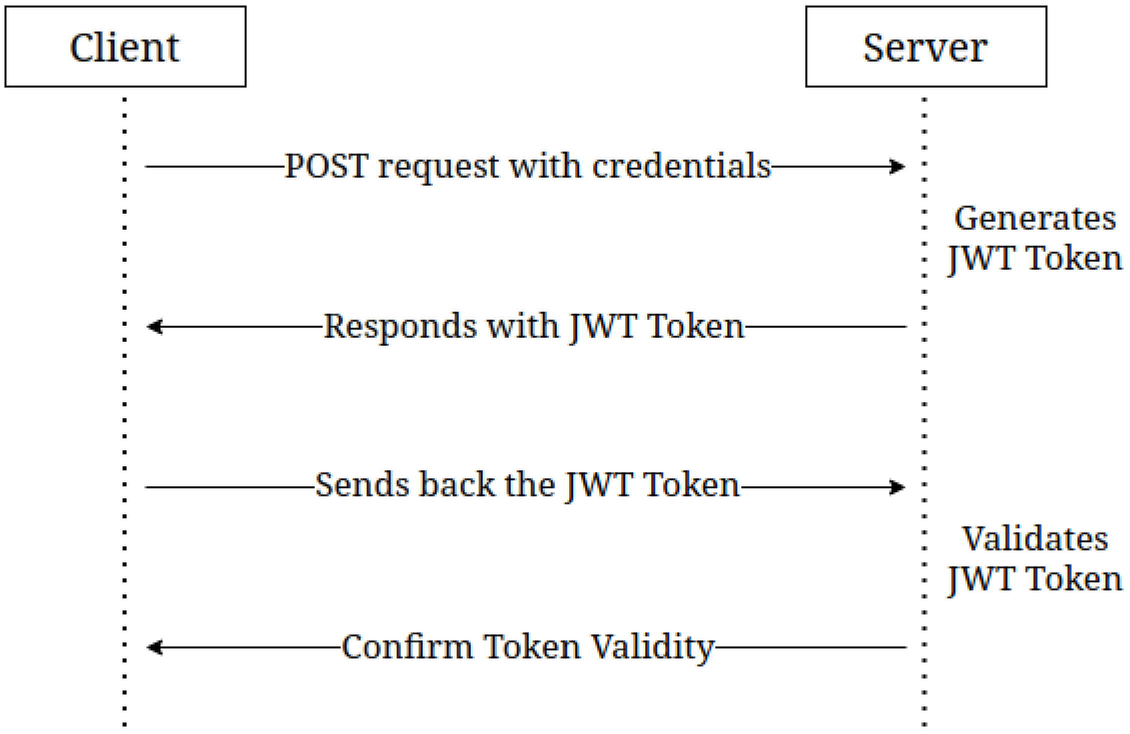

Figure 1: JWT Token Generation

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