

SKF write-ups

Python - JWT Secret

Running the app on Docker

```
$ sudo docker pull blabla1337/owasp-skf-lab:jwt-secret
```

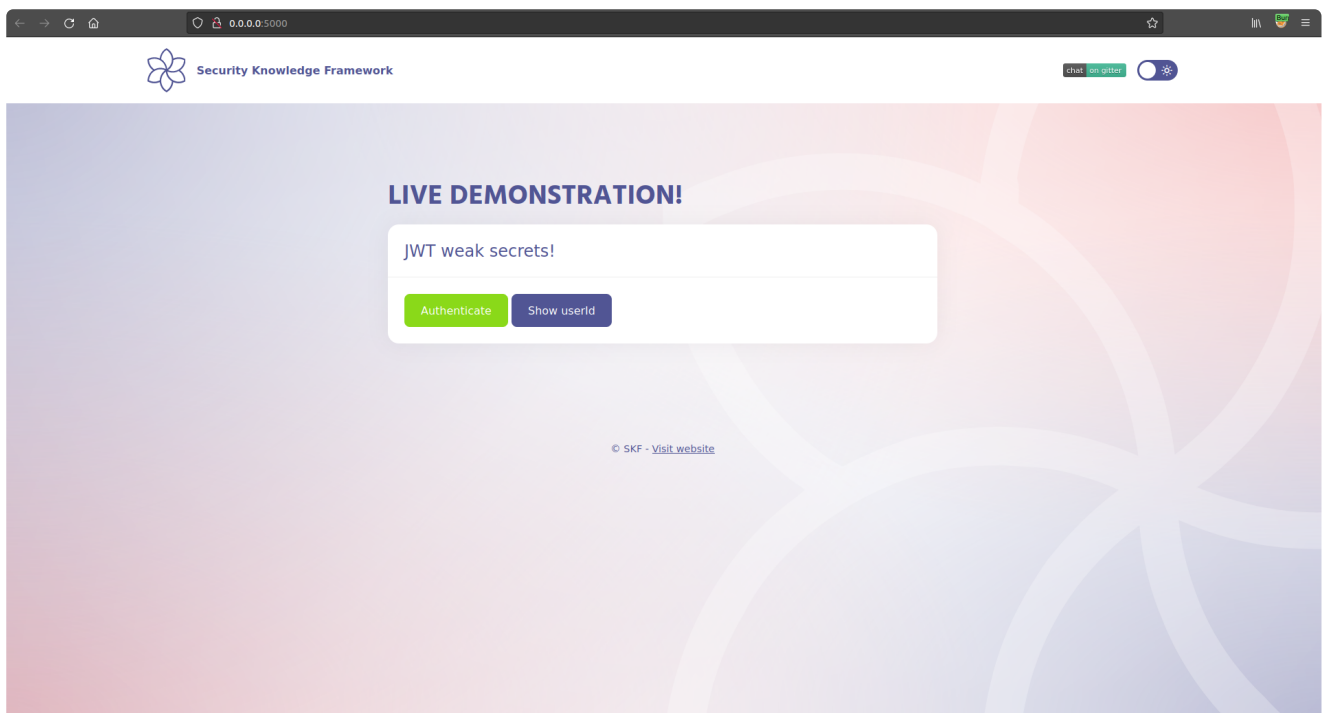
```
$ sudo docker run -ti -p localhost:5000:5000 blabla1337/owasp-skf-lab:jwt-secret
```

✓ Now that the app is running let's go hacking!

Reconnaissance

Step1

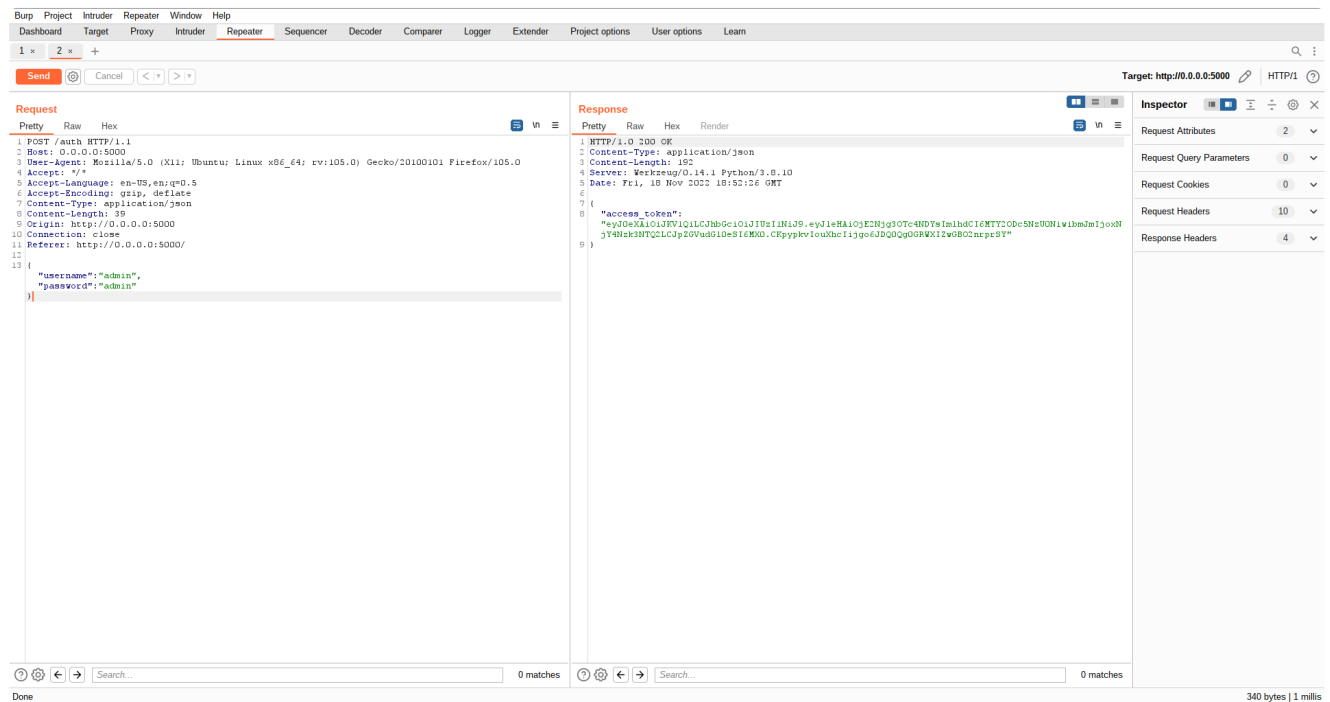
The application shows a dropdown menu from which we can choose an intro or chapters to be displayed on the client-side.



First thing we need to do know is to do more investigation on the requests that are being made. We do this by setting up our intercepting proxy so we can gain more understanding of the application under test.

After we set up our favourite intercepting proxy we are going to look at the traffic between the server and the front-end. Click on *Authenticate*.

The first thing to notice is after successful logon, the response contains an access token.



The image above shows the access-token contains three base64 encoded splitted with two dots (.) separators, which indicates it's a JSON Web Token (JWT):

Header

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Claims

```
{
  "exp": 1553003718,
  "iat": 1553003418,
  "nbf": 1553003418,
  "identity": 1
}
```

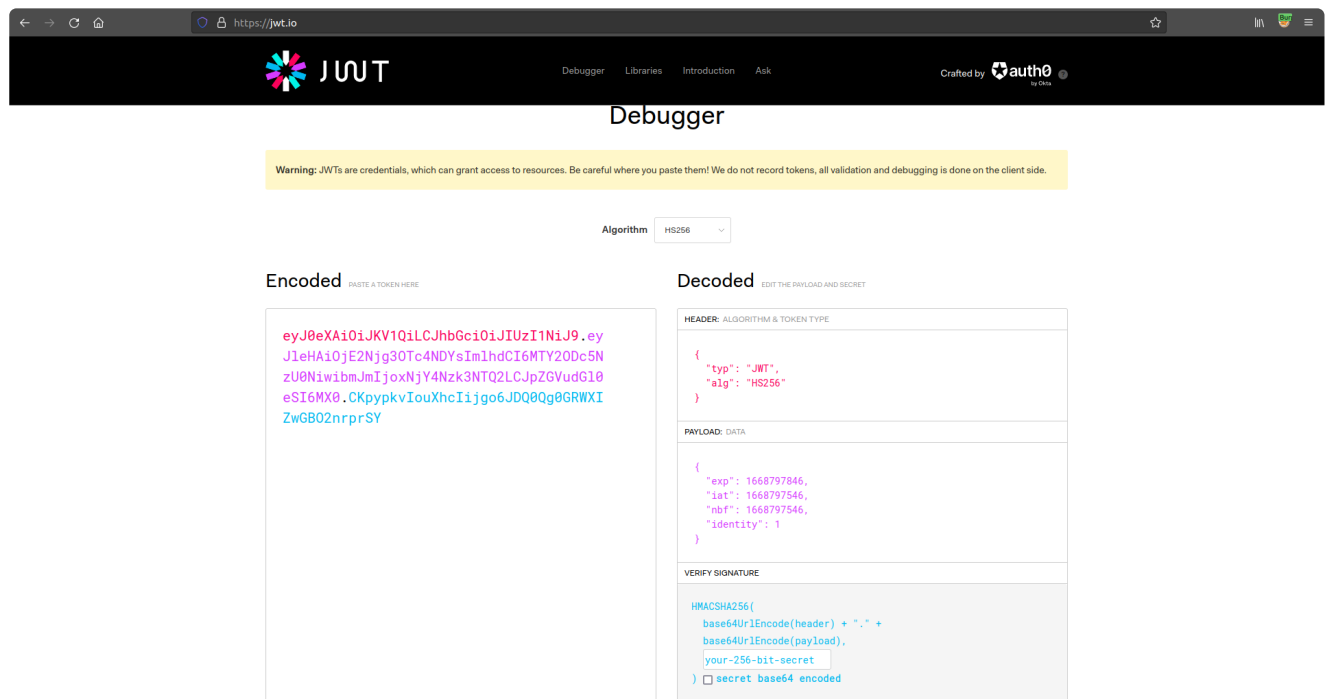
Signature

Last encrypted part, containing the digital signature for the token..

Exploitation

Step1

A potential attacker can now decode the token in <http://jwt.io> website to check its content.



As shown in the above picture, there are 2 points which can be tampered.

- alg header: contains the information of which algorithm is being used for digital signature of the token.
- identity: this information is used by the application to identify which user ID is currently authenticated.

How about checking if the server used a weak secret key for digital signature algorithm?

Checking the code below it's possible to see a weak secret key is being used, which can be easily guessed by a dictionary attack using tools available on internet and in your favorite PenTest distro.

```
app = Flask(__name__)
app.debug = True
app.config['SECRET_KEY'] = 'secret'

jwt = JWT(app, authenticate, identity)
```

Step 2

Using the weak secret key, let's change the *identity* value.

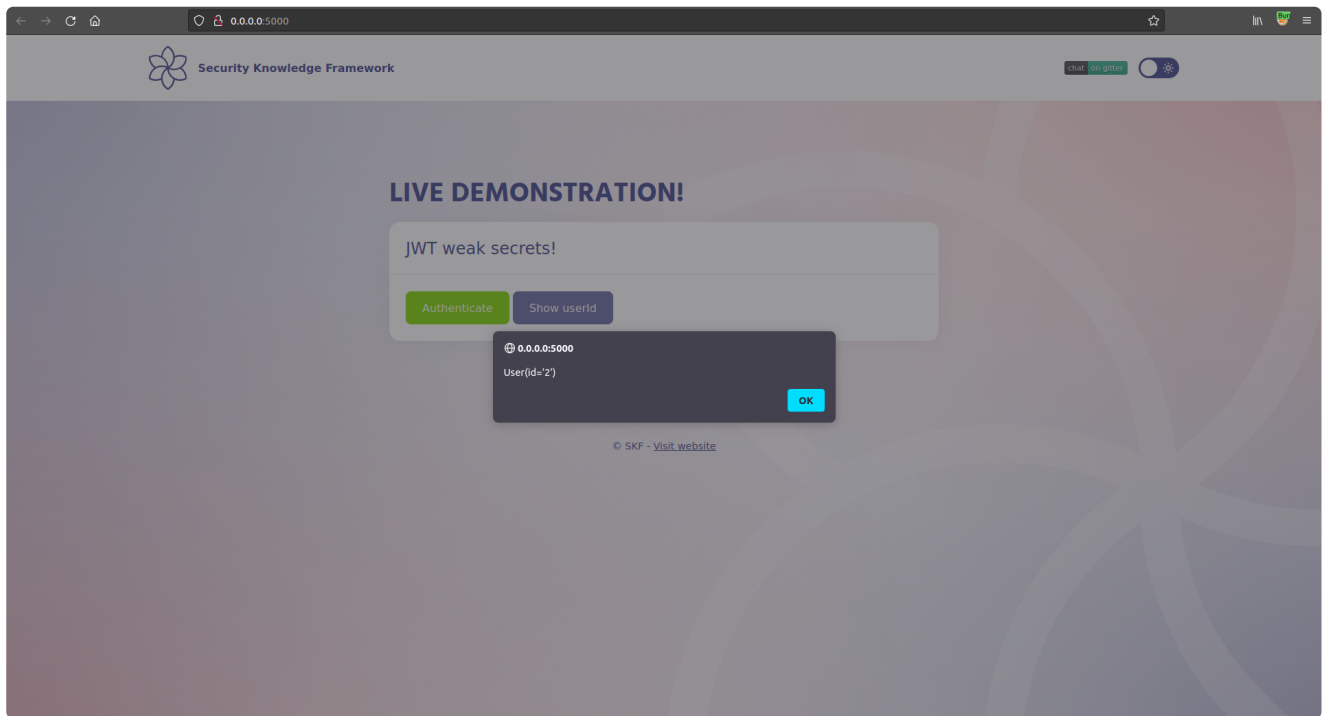
The image shows the JWT.io website interface. At the top, there's a navigation bar with links for Debugger, Libraries, Introduction, and Ask. The main content area is split into two columns: 'Encoded' and 'Decoded'. The 'Encoded' column has a text area containing a JWT token: `eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJleHAiOjE2Njg3OTc4NDYsImhhdCI6MTY2ODc5NzU0NiwibmJmIjoxNjY4Nzk3NTQ2LCJpZGVudG10eSI6Mn0.gfmUHHORJH87NHAVtpJv9z_KcT3wHApWEgChVx15-s`. Below this text area, a blue checkmark icon and the text 'Signature Verified' are displayed. The 'Decoded' column shows the token's structure. The 'HEADER' section indicates 'ALGORITHM & TOKEN TYPE' with a JSON object: `{ "typ": "JWT", "alg": "HS256" }`. The 'PAYLOAD' section shows 'DATA' with a JSON object: `{ "exp": 1668797846, "iat": 1668797546, "nbf": 1668797546, "identity": 2 }`. The 'VERIFY SIGNATURE' section shows the verification process: `HMCSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), secret)`. A 'SHARE JWT' button is located at the bottom right of the 'Decoded' section.

Step 3

Now, let's use the new generated JWT token to replace the one stored in browser's local storage.

The image shows a web browser displaying the 'Security Knowledge Framework' (SKF) application. The main content area features a 'LIVE DEMONSTRATION!' section with the text 'JWT weak secrets!' and two buttons: 'Authenticate' (green) and 'Show userid' (purple). Below this, there's a copyright notice: '© SKF - Visit website'. The browser's developer tools are open, showing the 'Storage' tab. The 'Local Storage' section is selected, and the 'access_token' key is highlighted. The value of 'access_token' is a JWT token: `eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJleHAiOjE2Njg3OTc4NDYsImhhdCI6MTY2ODc5NzU0NiwibmJmIjoxNjY4Nzk3NTQ2LCJpZGVudG10eSI6Mn0.gfmUHHORJH87NHAVtpJv9z_KcT3wHApWEgChVx15-s`. The 'Parsed Value' section shows the decoded token structure, including the header, payload, and signature.

And click on *Show userID* to check if the server accepted the tampered token.



Yes! The server accepted the tampered access-token. Can we check if there are more users available which can be impersonated?

Additional Resources

Please refer to the JWT.io information for more information regarding JWT.



JWT.IO - JSON Web Tokens Introduction

Also consider OWASP JWT Cheat Sheet as reference.

https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/JSON_Web_Token_Cheat...
github.com