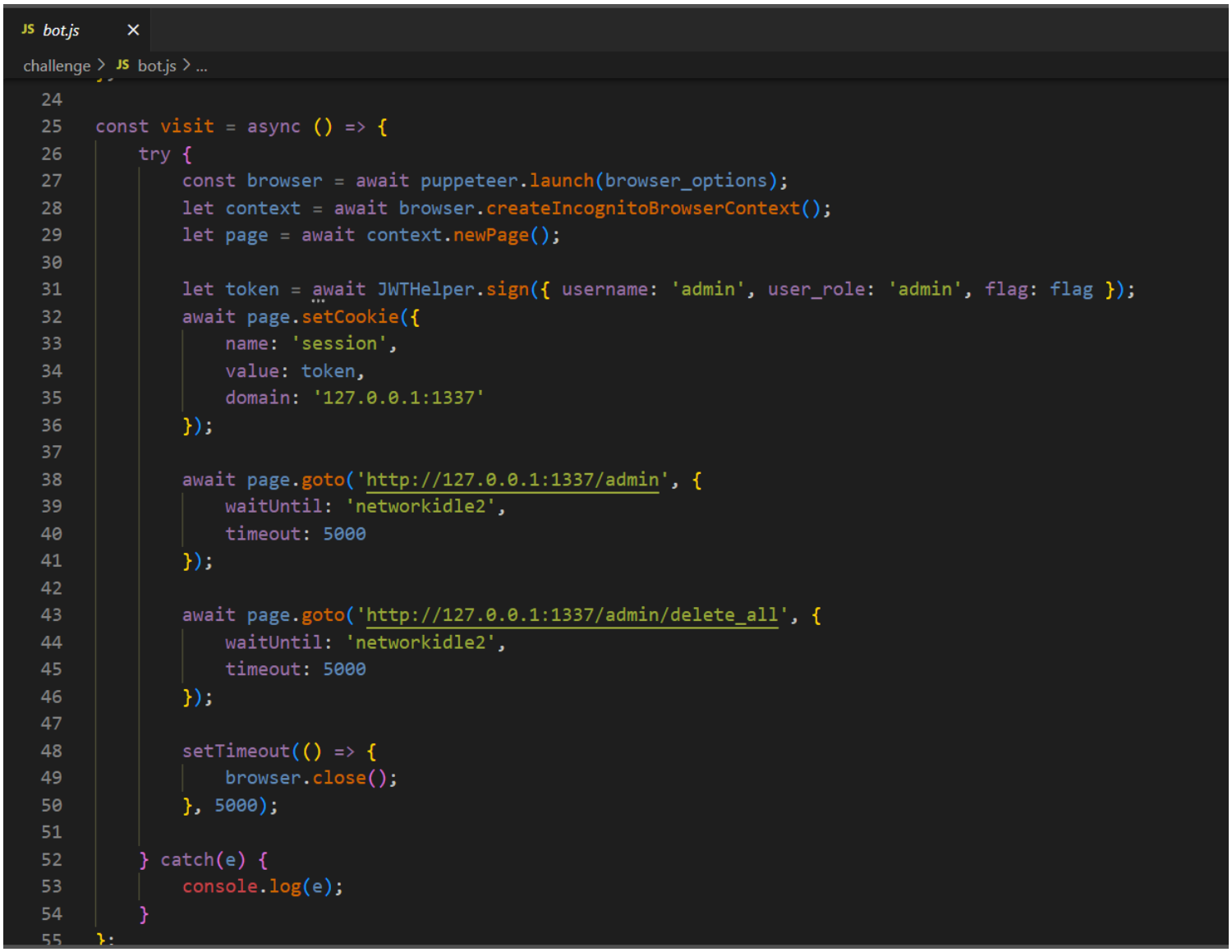
Cursed Secret Party

1) There is a bot that adds flag into jwt token



```
JS bot.js
challenge > JS bot.js > ...
24
25  const visit = async () => {
26      try {
27          const browser = await puppeteer.launch(browser_options);
28          let context = await browser.createIncognitoBrowserContext();
29          let page = await context.newPage();
30
31          let token = await JWTHelper.sign({ username: 'admin', user_role: 'admin', flag: flag });
32          await page.setCookie({
33              name: 'session',
34              value: token,
35              domain: '127.0.0.1:1337'
36          });
37
38          await page.goto('http://127.0.0.1:1337/admin', {
39              waitUntil: 'networkidle2',
40              timeout: 5000
41          });
42
43          await page.goto('http://127.0.0.1:1337/admin/delete_all', {
44              waitUntil: 'networkidle2',
45              timeout: 5000
46          });
47
48          setTimeout(() => {
49              browser.close();
50          }, 5000);
51
52      } catch(e) {
53          console.log(e);
54      }
55  };
```

2) The location loads data from database and renders

```
router.get('/admin', AuthMiddleware, (req, res) => {
    if (req.user.user_role !== 'admin') {
        return res.status(401).send(response('Unautorized!'));
    }

    return db.get_party_requests()
        .then((data) => {
            res.render('admin.html', { requests: data });
        });
});

router.get('/admin/delete_all', AuthMiddleware, (req, res) => {
    if (req.user.user_role !== 'admin') {
        return res.status(401).send(response('Unautorized!'));
    }

    return db.remove_requests()
            .then(() => res.send(response('All records are deleted!')));
})

```

admin.html

~/Web/Cursed Secret Party/web_cursed_party/challenge/views/admin.html

```
<html>
    <head>
        <link rel="stylesheet" href="/static/css/bootstrap.min.css" />
        <title>Admin panel</title>
    </head>

    <body>
        <div class="container" style="margin-top: 20px">
            {% for request in requests %}
                <div class="card">
                <div class="card-header"> <strong>Halloween Name</strong> : {{ request.halloween_name | safe }} </div>
                <div class="card-body">
                    <p class="card-title"><strong>Email Address</strong>    : {{ request.email }}</p>
                    <p class="card-text"><strong>Costume Type </strong>   : {{ request.costume_type }} </p>
                    <p class="card-text"><strong>Prefers tricks or treat </strong>   : {{ request.trick_or_treat }} </p>

                    <button class="btn btn-primary">Accept</button>
                    <button class="btn btn-danger">Delete</button>
                </div>
            </div>
            {% endfor %}
        </div>

    </body>
</html>
```

3) Input to the database is not sanitized

```
router.post('/api/submit', (req, res) => {
    const { halloween_name, email, costume_type, trick_or_treat } = req.body;

    if (halloween_name && email && costume_type && trick_or_treat) {

        return db.party_request_add(halloween_name, email, costume_type, trick_or_treat)
            .then(() => {
                res.send(response('Your request will be reviewed by our team!'));

                bot.visit();
            })
            .catch(() => res.send(response('Something Went Wrong!')));
    }

    return res.status(401).send(response('Please fill out all the required fields!'));
});
```

We could exploit xss

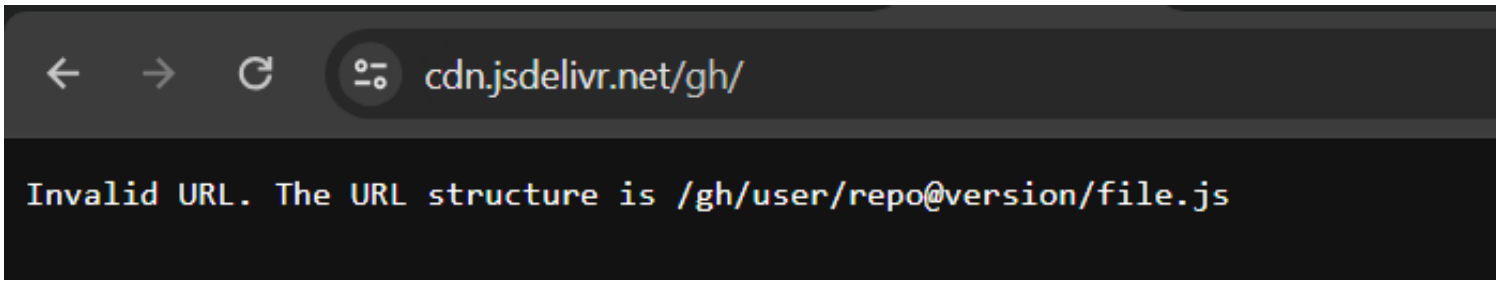
4) CSP

```
app.use(function (req, res, next) {
    res.setHeader(
        "Content-Security-Policy",
        "script-src 'self' https://cdn.jsdelivr.net ; style-src 'self' https://fonts.googleapis.com; img-src 'self'; font
    );
    next();
});
```

There is a csp in our way

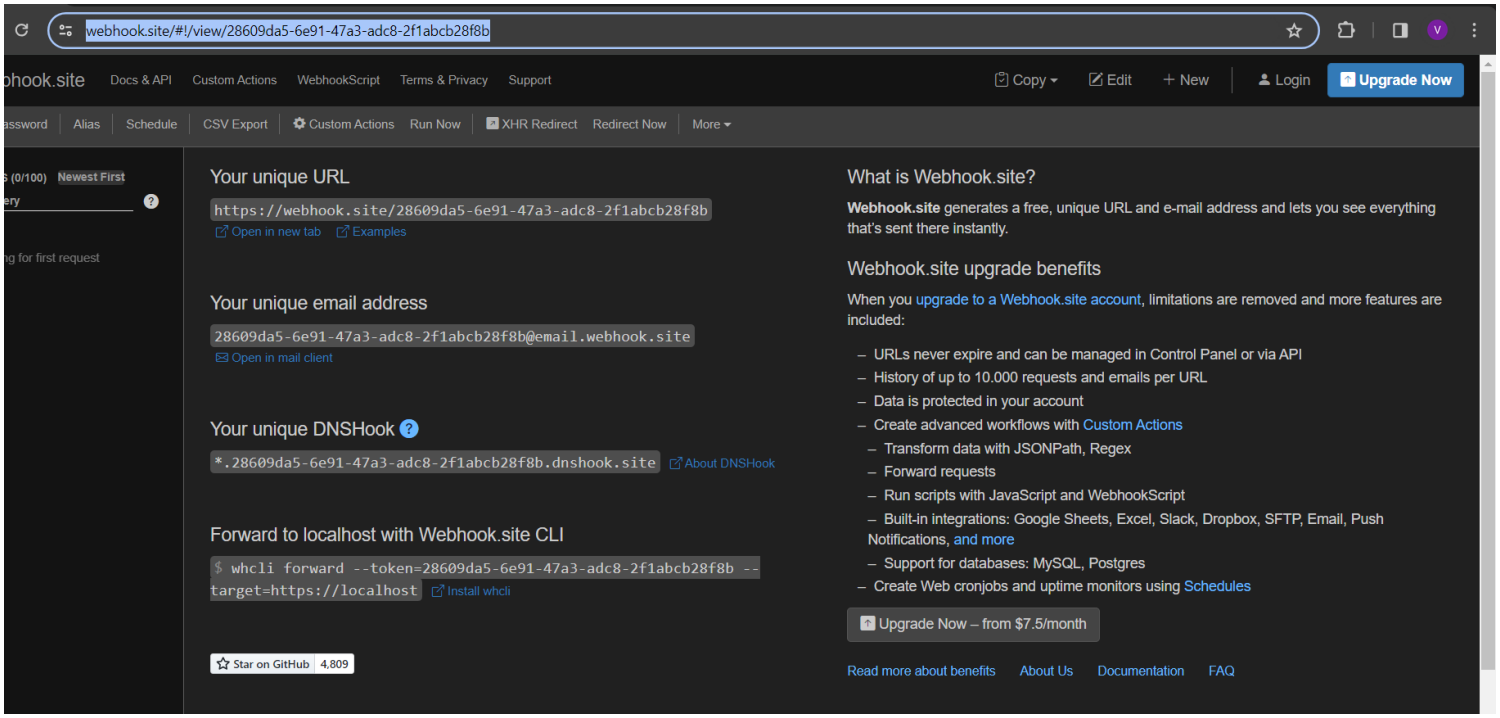
"Content-Security-Policy",
"script-src 'self' https://cdn.jsdelivr.net ; style-src 'self' https://fonts.googleapis.com; img-src 'self'; font-src 'self' https://fonts.gstatic.com; child-src 'self'; frame-src 'self'; worker-src 'self'; frame-ancestors 'self'; form-action 'self'; base-uri 'self'; manifest-src 'self'"

5) CDN Github

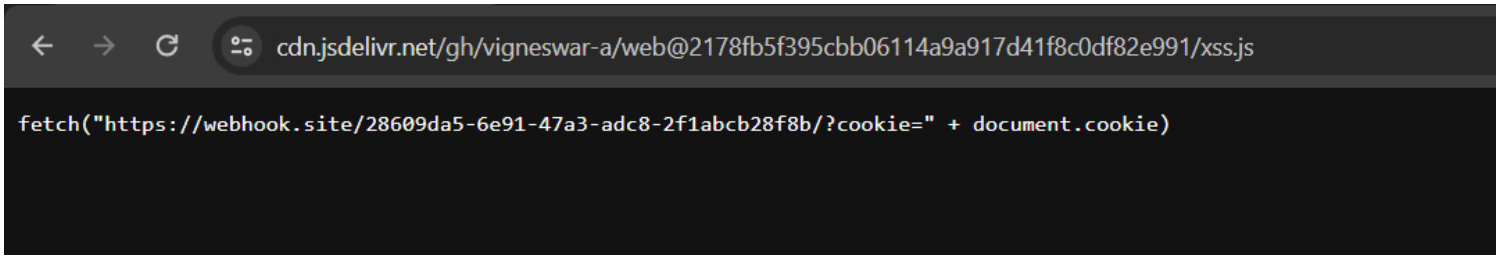


CDN has provision to script load from github

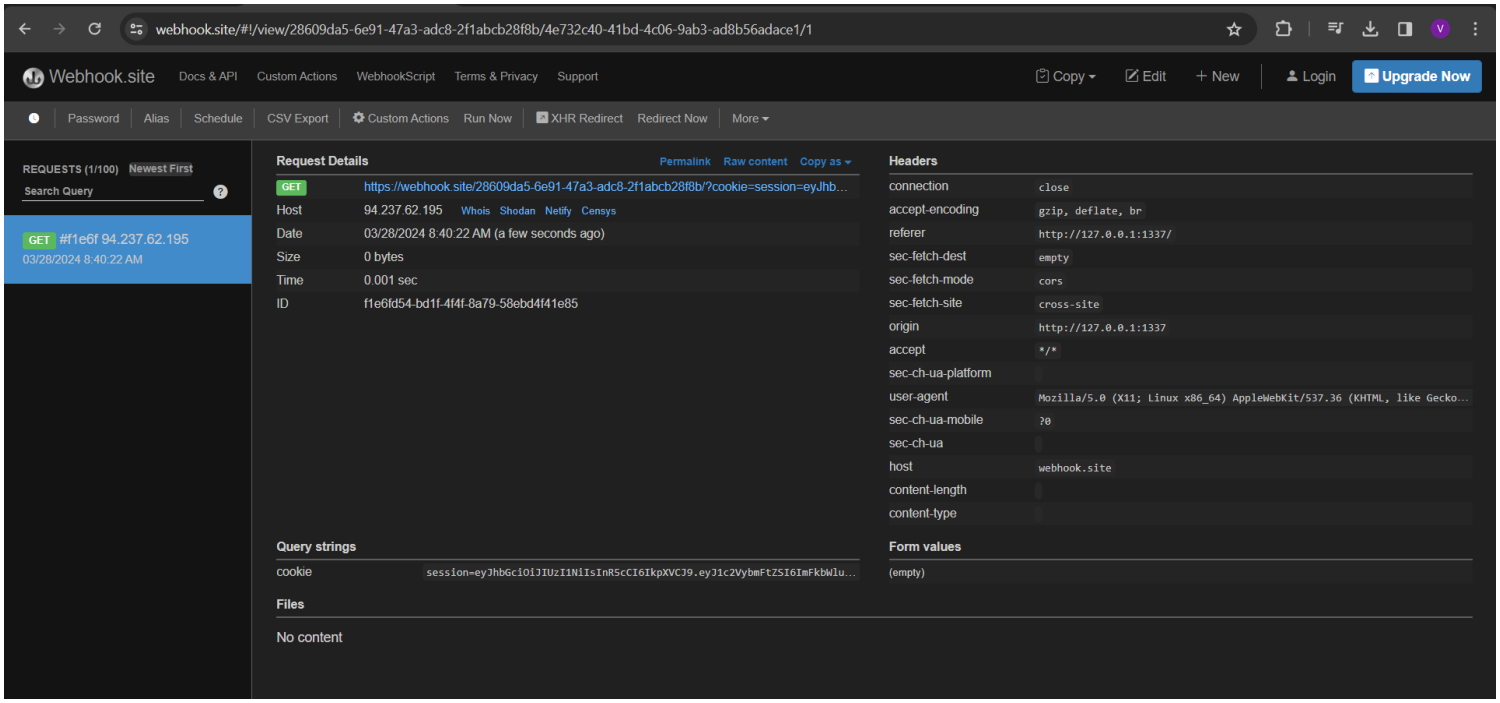
6) Got a webhook

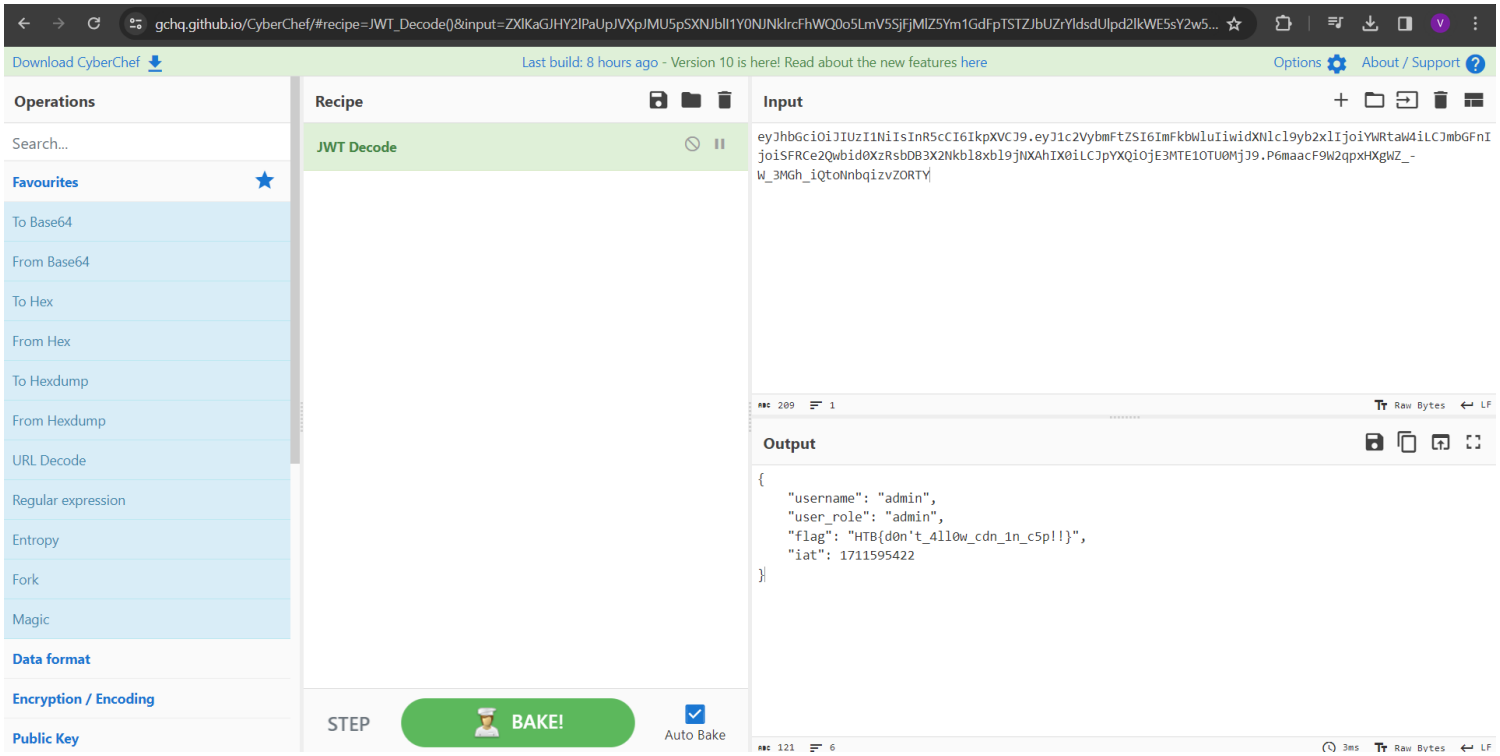


7) Made payload



8) Submitted payload
Got a cookie




gchq.github.io/CyberChef/#recipe=JWT_Decode()&input=ZXlKaGJHY2lPaUpJVXpJMU5pSXNJbll1Y0NJNklrcFhWQ0o5LmV5SjFjMlZ5Ym1GdFpTSTZJbUZrYldsdUlpd2lkWE5sY2w5...
Download CyberChef
Last build: 8 hours ago - Version 10 is here! Read about the new features here
Options
About / Support
Operations
Search...
Favourites
To Base64
From Base64
To Hex
From Hex
To Hexdump
From Hexdump
URL Decode
Regular expression
Entropy
Fork
Magic
Data format
Encryption / Encoding
Public Key
Recipe
JWT Decode
STEP
BAKE!
Auto Bake
Input
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6ImFkbWluIiwidXNlcl9yb2xlIjoiYWRtaW4iLCJmbGFnIjoiSFRCe2Qwbid0XzRsbDB3X2Nkbl8xbl9jNXAhIX0iLCJpYXQiOjE3MTE1OTU0MjJ9.P6maacF9W2qpxHXgWZ_-W_3MGh_iQtoNnbqizvZORTY
209
1
Raw Bytes
LF
Output
{
"username": "admin",
"user_role": "admin",
"flag": "HTB{d0n't_4ll0w_cdn_1n_c5p!!}",
"iat": 1711595422
}
121
6
3ms
Raw Bytes
LF