XSS Gauntlet

In this assignment you will be learning about cross-site scripting (XSS) attacks through hands-on exploitation.

There are three challenges:

- https://xss1.defund.workers.dev/
- https://xss2.defund.workers.dev/
- https://xss3.defund.workers.dev/

You can interact with the admin bot here:

- http://box.priv.pub:1337/nyu-csci-ua-0480-063

If you have any questions, feel free to email me at `ww@nyu.edu` or attend my office hours.

Introduction

Let's start by visiting the first challenge's website at xss1.defund.workers.dev. Try entering `foo` into the textbox and press the "Post" button. You should be redirected here and see the string "foo". Now let's go back and enter `<b>foo</b>` into the textbox. You should be redirected here and see the string "foo" again, but now in **bold** font!

So what's going on? If you view the source code of the first page (in Chrome, right click and press "View Page Source"), you'll see that it looks like this:

```
<!DOCTYPE html>
<body style="margin: 2rem; font-family: monospace">
  <h1>XSS Gauntlet 1</h1>
  foo
</body>
```

The source code of the second page looks like this:

```
<!DOCTYPE html>
<body style="margin: 2rem; font-family: monospace">
  <h1>XSS Gauntlet 1</h1>
  <b>foo</b>
</body>
```

Essentially, the website takes our message and inserts it directly into the HTML. This means that `<b>` and `</b>` are interpreted as HTML tags, not content!

This type of behavior can be very dangerous because HTML is code, and code is powerful. For example, try visiting this page; an alert box should pop up. The source code looks like this:

```
<!DOCTYPE html>
<body style="margin: 2rem; font-family: monospace">
  <h1>XSS Gauntlet 1</h1>
  <script>alert("pwned")</script>
</body>
```

Here I am leveraging the fact that HTML supports script tags to run JavaScript. Try messing around with this yourself!

When a website exhibits the sort of behavior described above, it is typically vulnerable to XSS attacks. At a high level, XSS allows an attacker to run untrusted code on a trusted website. For example, suppose Twitter directly rendered tweets as HTML. Then Alice could write a tweet containing JavaScript code such that, if Bob viewed the tweet, the code would run in Bob's browser tab. Alice would then have control over Bob's Twitter account and do things like post on his behalf. Here's a famous example of an XSS attack on MySpace.

In this project you will act as Alice, writing HTML code to attack Bob. We will simulate Bob using an automated bot, which you can interact with here. The bot will visit any link you provide. **Please do not abuse/spam the bot, you should only interact with it once you have an exploit that works locally.**

Cookies

Websites use cookies to authenticate user sessions. On xss1.defund.workers.dev, open the JavaScript console (in Chrome, right click, press "Inspect", and select the "Console" tab) and enter `document.cookie`. The console should return the string `"dummy=value"`; this is a placeholder cookie set by the website. On the bot's browser, the cookie will instead look like this:

```
flag=flag{xss#_XXXXXXXXXXXXXXXX}
```

Your goal is to exfiltrate this cookie using XSS. For example, we can display the cookie in an alert box using the following payload:

```
<script>alert("Here is the cookie: " + document.cookie)</script>
```

Try running this in your browser! Unfortunately, even if we send this link to the admin bot, we cannot see the alert box.

Request bins

In order to exfiltrate the flag, we will use a *request bin*, i.e., a website that logs all HTTP requests it receives. I personally like webhook.site, but there are also other websites.

Go ahead an open webhook.site. Copy the URL under "Your unique URL" and visit the URL in another tab. On webhook.site, you should see an HTTP request logged; this was from your browser! Now try submitting the URL to the bot. You should see another HTTP request logged, this time from the IP address `23.146.104.18`; this is the bot.

Request bins are useful because you can transmit information in the URL you request. For example, try appending `?foo=bar` so that it looks something like this:

```
https://webhook.site/fdd7ff0a-2393-4a11-94cc-cf18aeadca40?foo=bar
```

If you visit this URL, webhook.site should log a request containing this query string.

Putting it all together

You should now know everything you need to solve the first challenge. At a high level, you should do the following:

- Write a JavaScript payload which sends the website's cookie to a request bin. You can use fetch to make HTTP requests within JavaScript. You should test this locally, using the JavaScript console.
- Write an XSS payload, enter it into the textbox on xss1.defund.workers.dev, and get the resulting link. Make sure when you visit the link locally, webhook.site logs the placeholder cookie `foo=bar`.
- Submit the link to the bot here, wait a few seconds, and get the flag from the request bin.

Assignment

After solving the first challenge, you should move on to the other ones:

- In the second challenge (xss2.defund.workers.dev), the website bans any payload containing the `script` as a substring. Hint: try to see if there's another way to run JavaScript in HTML, besides script tags.
- In the third challenge (xss3.defund.workers.dev), the website bans any payload containing single or double quotes. Hint: figure out a way to indirectly construct strings in JavaScript.

Both challenges are fairly open-ended, and there are many possible solutions. Remember, Google is your friend.

For the submission, write a report detailing your solutions. For each challenge, include your XSS payload, the flag you obtained from the bot, and a short explanation (~1 paragraph) of how you solved the challenge.

Final thoughts

If you enjoyed doing this assignment, you might be interested in Capture the Flag (CTF) competitions. In CTFs, players compete to solve challenges that test hacking skills (not just XSS and web security, but also binary exploitation, cryptography, etc.); if this sounds intriguing, I would recommend checking out the OSIRIS Lab at NYU Tandon.