Cross-site Scripting Attack

John Doll

Objective:

After learning various methods of cross-site scripting attacks and defense, I will attempt my own attack on a website that I control and record the outcome.

Background Study:

Cross-site scripting is an attack where a perpetrator injects malicious scripts into a website in order to steal information from users. Most often these attacks are centered around obtaining cookies, but also can target usernames, passwords, emails, and bank account credentials, which are all regularly stored in cookies. There are several different types of cross-site scripting, namely stored attacks, reflected attacks and document object model (DOM) based attacks (Meric). Fortunately, there are also ways of protecting a website from cross-site scripting attacks; validating input and implementing a content security policy are just a couple that can be used.

A stored attack, sometimes referred to as a persistent attack, gains unauthorized access and exploits vulnerabilities through HTTP requests (Cloudflare). To achieve a stored attack, a malicious actor takes advantage of the inadequacy of input sanitation in order to put malicious code into the HTTP request (Meric). The actor could add their code into an input field on a website if there is not proper input sanitation in place to compromise the website. This means that after the malicious code has been inserted, any users who access that specific webpage have that code executed in their browser. Reflected attacks are achieved by sending a link with

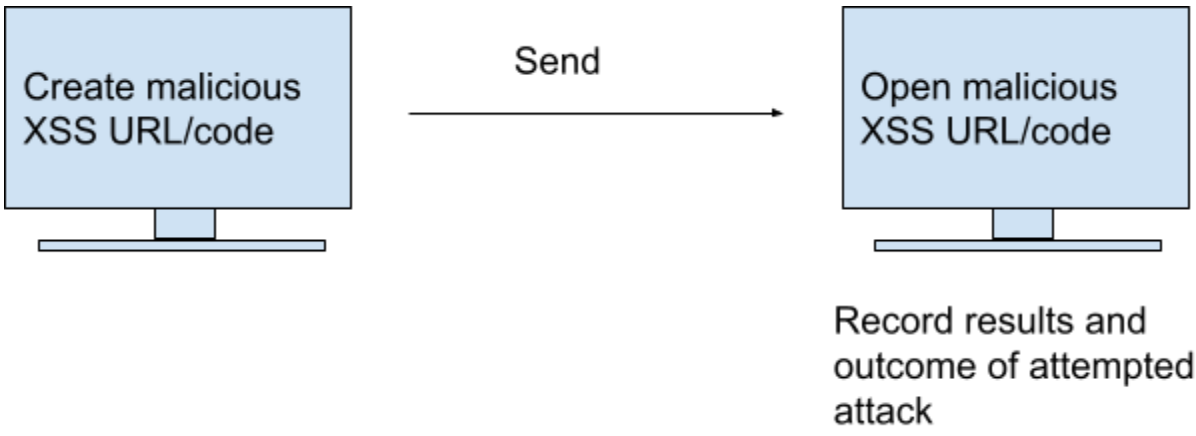
malicious code in the url. An attacker would add JavaScript code to the end of the url so that whenever any user opens the link sent by the attacker, the code is executed (Meric). The simplicity of the attack makes it so widespread and incredibly efficient as the malicious code does not need to be stored on the server either. In DOM-based attacks, the attacker is able to gain access to the website by hijacking user sessions. To do this, the attacker will send malicious inputs to common JavaScript functions to get access to the web session (Meric).

It can be hard to tell as a developer if a website or webpage is susceptible to a cross-site scripting attack. There are several services that can perform a security review for a website to help find vulnerabilities (OWASP). The best way to check as a developer is to identify where all the inputs and outputs for HTTP requests are located, and address the potential that malicious code would be able to infect the request. Often, there are more vulnerabilities in a website than easily found. The Open Web Application Security Project (OWASP) in their article on cross-site scripting state, “If one part of a website is vulnerable, there is a high likelihood that there are other problems as well.” There are many ways as a developer to defend from a cross-site scripting attack. One of those is to perform input validation (Meric). By validating input, the opportunity is reduced for malicious code or responses to be added in webpage forms. When applicable, input should be restricted fields other than text fields such as a dropdown (Meric). Another way to defend against cross-site scripting attacks is to enact a web application firewall (WAF). A WAF monitors requests, searching for malicious requests being made to a site. A third line of defense is using a content security policy (CSP). A CSP will allow certain URLs to be whitelisted for executing code, and all others will be blocked. This is a very strong form of detection because the CSP specifies the only places where code is allowed to be executed from, and blocks all other attempted code executions (Meric). HTTP TRACE is also a large

vulnerability as it makes it possible for an attacker to retrieve cookie information of a user even if cookies are disabled. HTTP TRACE should be turned off for all web servers to block this loophole. Furthermore, OWASP provides a cheat sheet of other protections that can be taken in order to add to the security of a website and reduce cross-site scripting attacks.

Methodology:

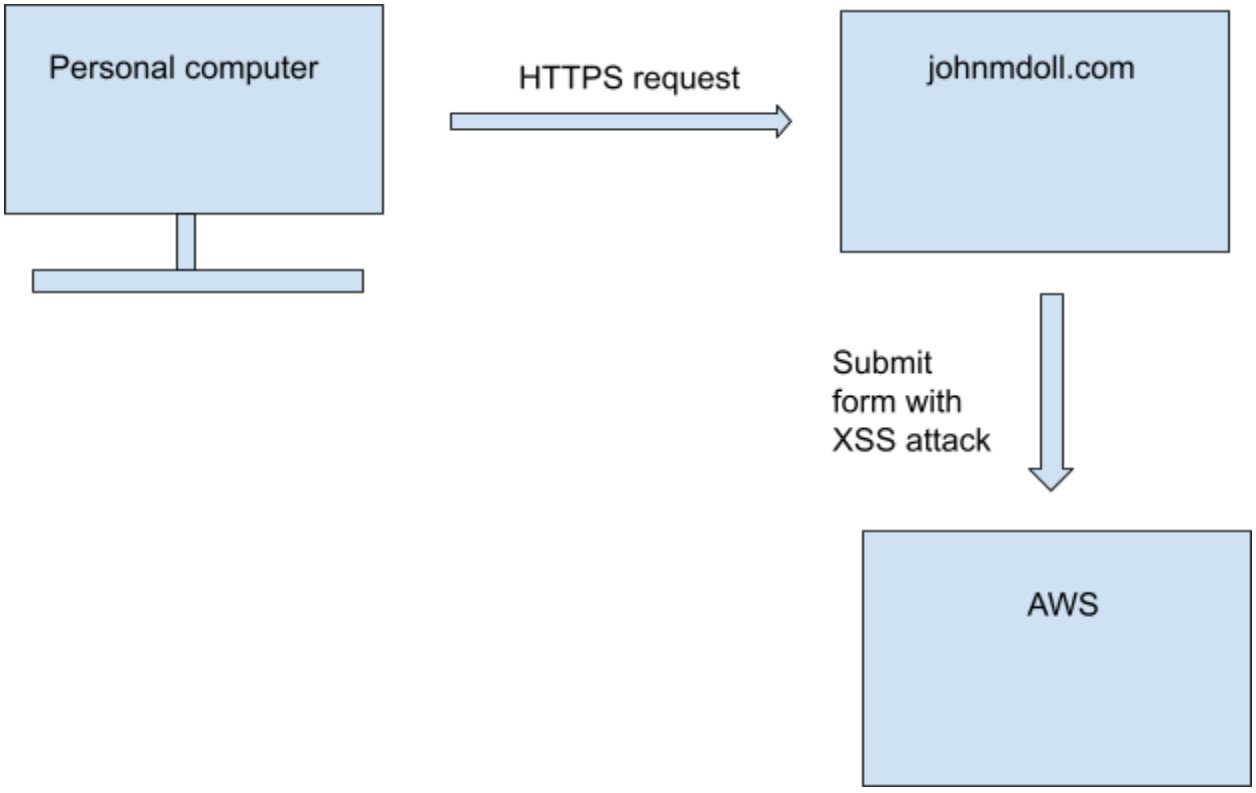
On a website that I have control of, I will attempt to perform a reflected cross-site scripting attack by sending a url with malicious JavaScript code to myself. I will open the url and record the webpage response.

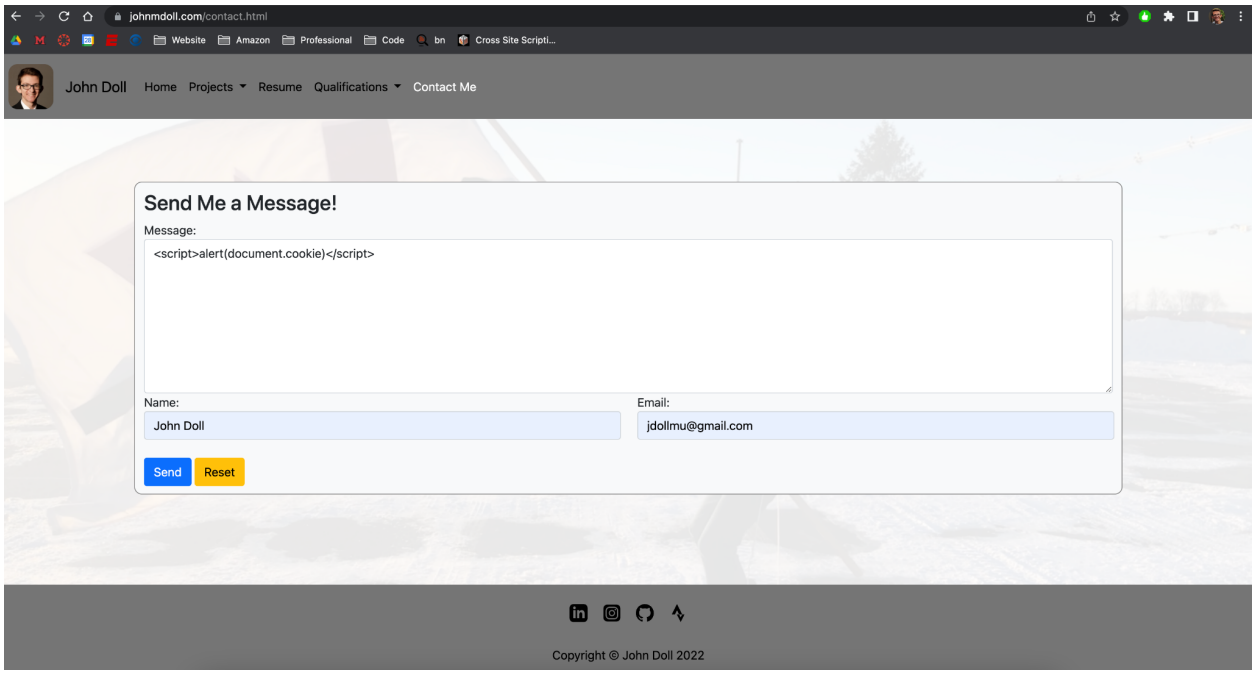


Conclusion:

As written in the background, there are three types of cross-site scripting attacks. A stored attack involves putting malicious code in an input box to be sent to a website database. In the database, the script, for example, could send all new inputs to the database to the intruder's web server. I attempted this approach on my personal website, but found myself getting blocked.

My website is hosted on AWS and I use a tool called Formspree for emailing. It has strong protection against cross-site scripting and prevented me from executing my code.



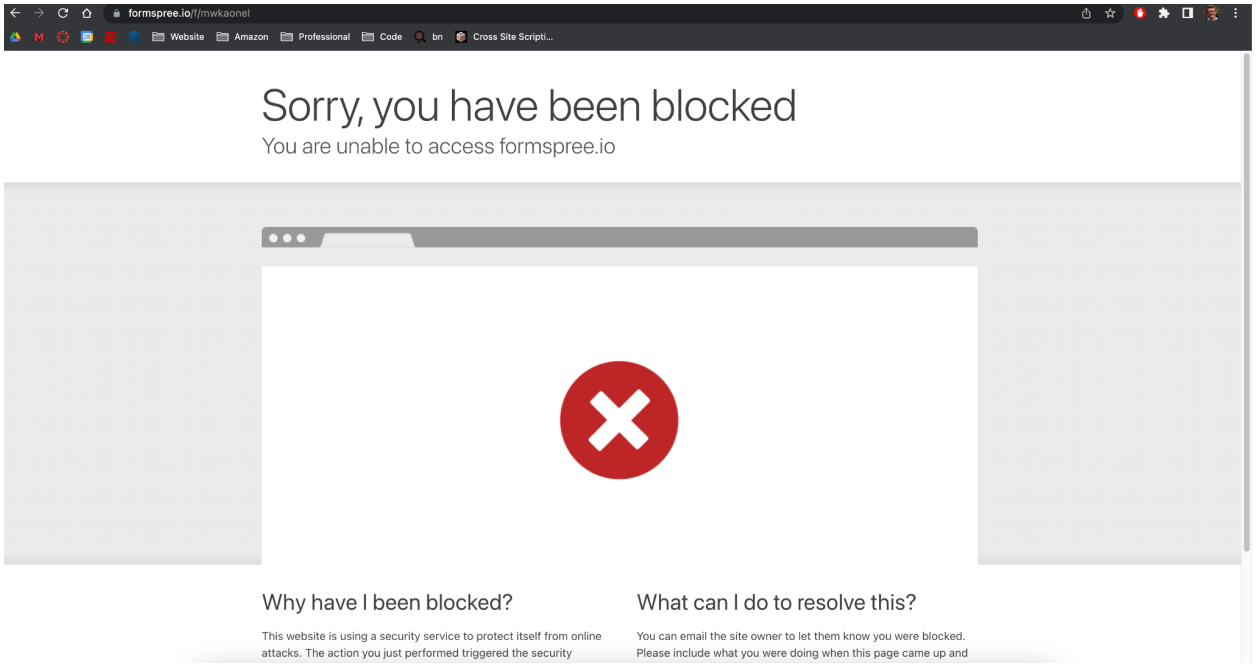


(Figure 1)

As you can see in Figure 1 above, I put my name and email in, and in the message I entered:

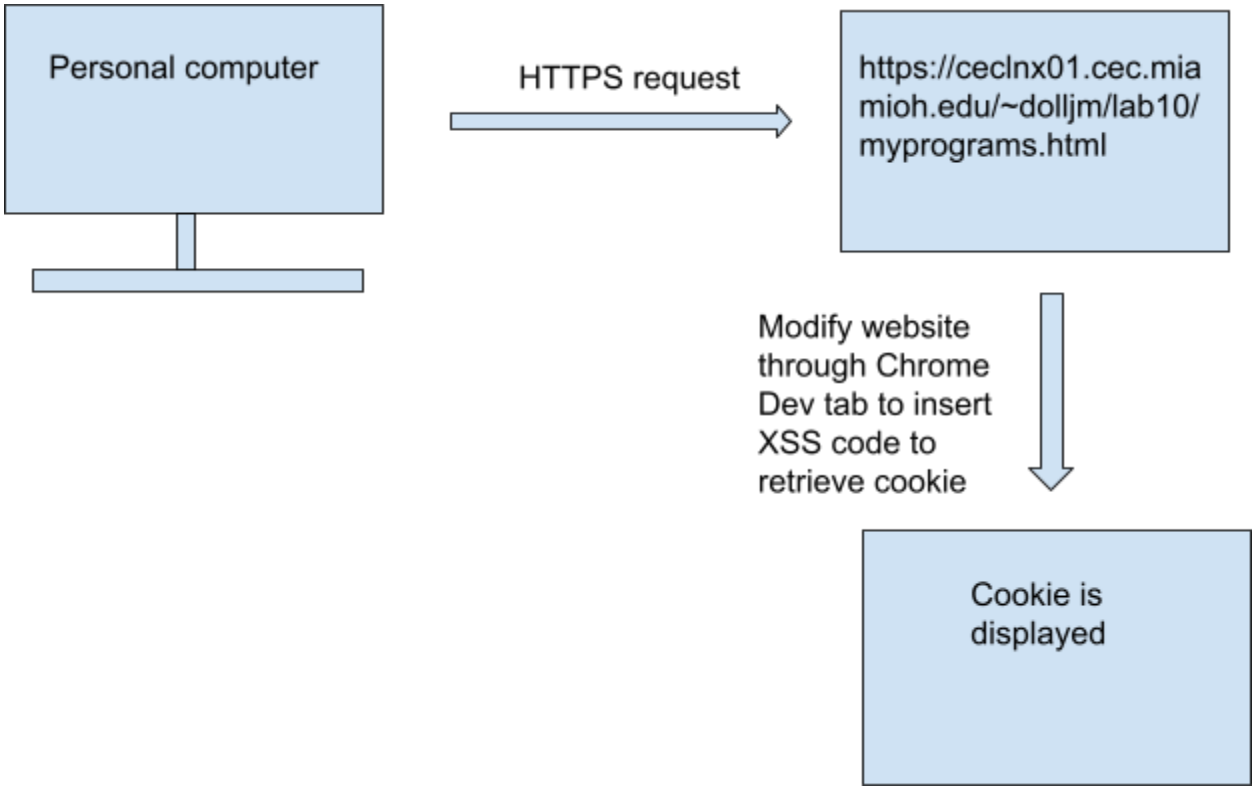
<script>alert(document.cookie)</script>

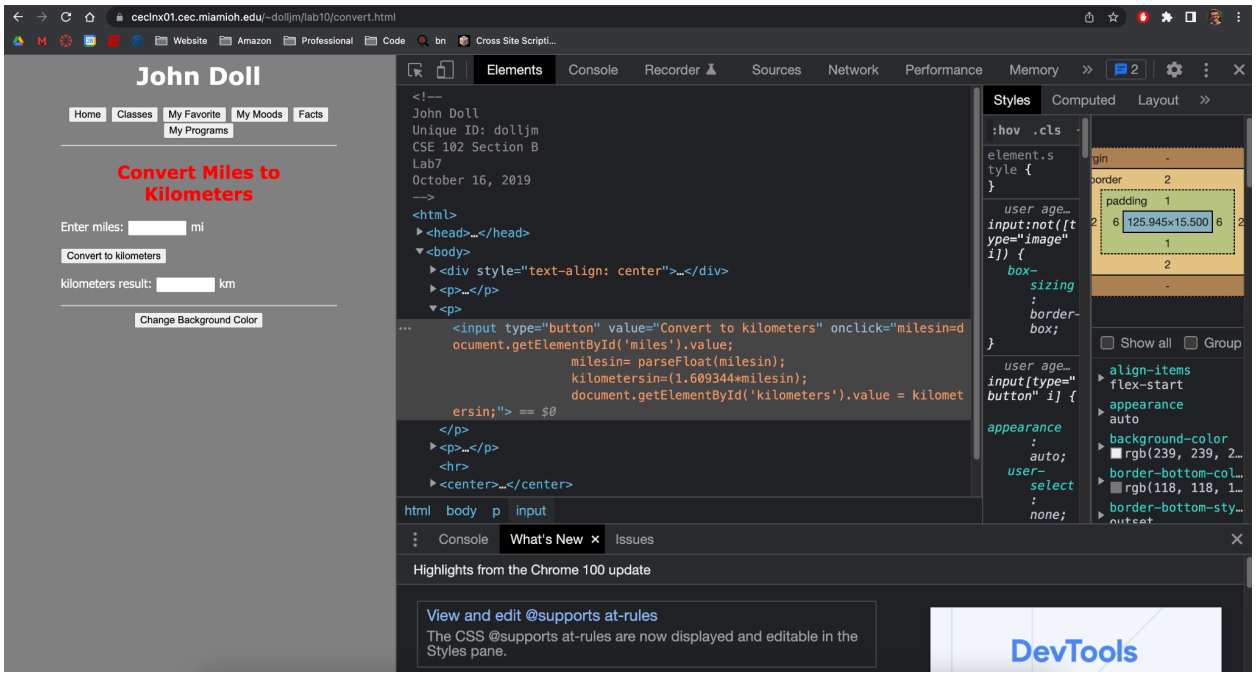
This caught the attention of my website and ended up blocking me, as seen in Figure 2 below. Clearly, it saw my input as malicious and blocked my request, which is good for the security of my website. I tried several other scripts but all were blocked and I was not able to bypass the filter.



(Figure 2)

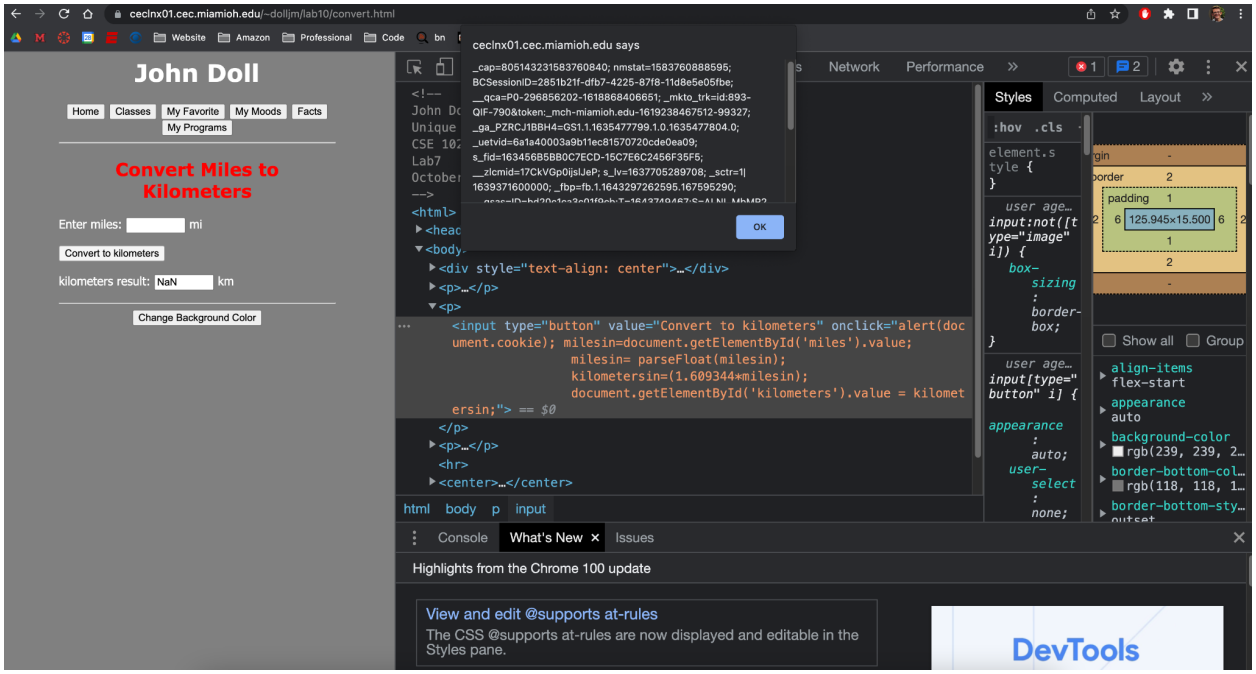
I then tried a version of a DOM-based attack by inserting my script into the website I created for CSE 102. After a little bit of trouble, I was able to start generating results. This is the original code for the button to convert miles to kilometers seen on the left-hand side of Figure 3.





(Figure 3)

I inserted the same script from above in the “onclick” attribute before the large function that actually does the miles to kilometers conversion.



(Figure 4)

Figure 4 shows the result after inputting my script, and clicking the button. It gives an alert and displays all the cookies that are being stored on the website. This script will continue to work each time I press the button, even when adding inputs to the input boxes and clearing them. However, if I navigate away from the page, the script is lost, even when pressing the return arrow to come back to this mile to kilometer conversion page. The script only lasts while the page is active, and unfortunately, I cannot copy the URL and send it to a friend either, as my script is not saved to the site or in the link. This could however be effective for certain scenarios where public computers are used. For example, I could open up Google on a computer in a business center at a hotel, and insert my script on the webpage. With a more malicious script that would send critical information to me about the user and what they are searching, I could gather data and information this way. The third type of cross-site scripting attack is a reflected attack, which I explored but did not implement. Between each of the two websites I tried attacking, and a third I created in CSE 383, none took query strings. A query string is the part after the question

mark in the proceeding url: www.example.com/index.html?param=1. The malicious script would go in place of the “1” and be set as a variable on the page, and the code would be executed when the variable was used. In order to inflict a reflection attack, I would generally need a query string to interact with in order to send my script with the link successfully. If there is no query string and I add the script to the end of the URL, it is not executed.

In conclusion, while I did not create a fully successful cross-site scripting attack, I learned much about them. I learned what they looked like and the best ways to prevent them. I learned that many safeguards are automatically put in place by web hosts, web browsers, and the languages themselves to prevent such attacks from happening. I was able to get malicious scripts to run, albeit only in my own browser, but have a better understanding of how they could be used in a more unsafe environment.

References:

“Cross Site Scripting (XSS).” Cross Site Scripting (XSS) Software Attack | OWASP Foundation, Open Web Application Security Project, https://owasp.org/www-community/attacks/xss/#.

Meric, Nedim. “Cross Site Scripting in JavaScript: Everything You Need to Know.” Bright Security, Bright Security Inc., 8 Nov. 2021, https://brightsec.com/blog/cross-site-scripting-javascript/.

“What Is Cross-Site Scripting? - Cloudflare.” What Is Cross-Site Scripting?, Cloudflare, Inc., https://www.cloudflare.com/learning/security/threats/cross-site-scripting/.