XSS and CSRF

Michaela Sallese

South Hills School of Business & Technology

Server Side Web Development IT241

Nicholas Page

Nov. 4, 2024

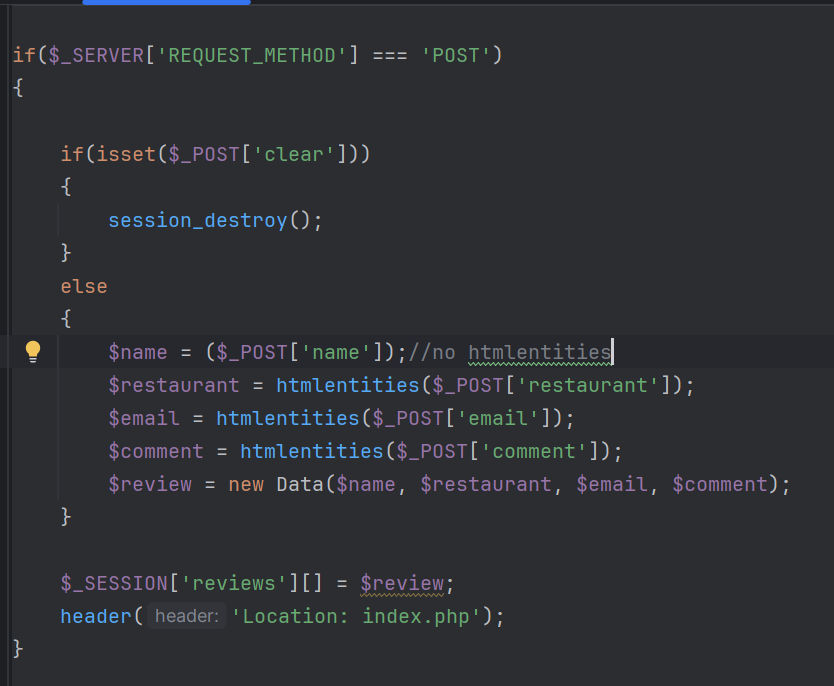
Since the launch of the World Wide Web on April 30, 1993, there have been malicious actors amongst web users. Though it would take some time for the web as we know it today to come about, a major step forward was taken by Brendan Eich in 1995 after the ten day labor of JavaScript was introduced to Internet society. Gone were the days of simple display-only sites of just HTML and CSS. JavaScript is an extremely powerful language that works with HTML, allowing for document object model (DOM) manipulation, browser interactivity, and more. But with extreme power came extreme opportunity for the wrong people to use this power. First spotted in 1999 by engineers at Microsoft, it was clear that Cross-Site Scripting (XSS) was only the beginning for opportunities for ill intentioned web users, as not even two full years later Peter Watkins defined Cross-Site Request Forgery (CSRF). While the two ways of targeting script are similar, and even can be used together, there are distinct differences here that programmers must be ever vigilant for as to not risk their user’s privacy. The following is a brief explanation of two of the OWASP’s top 10 web security issues and what developers need to be aware of when making their applications to minimize these risks.

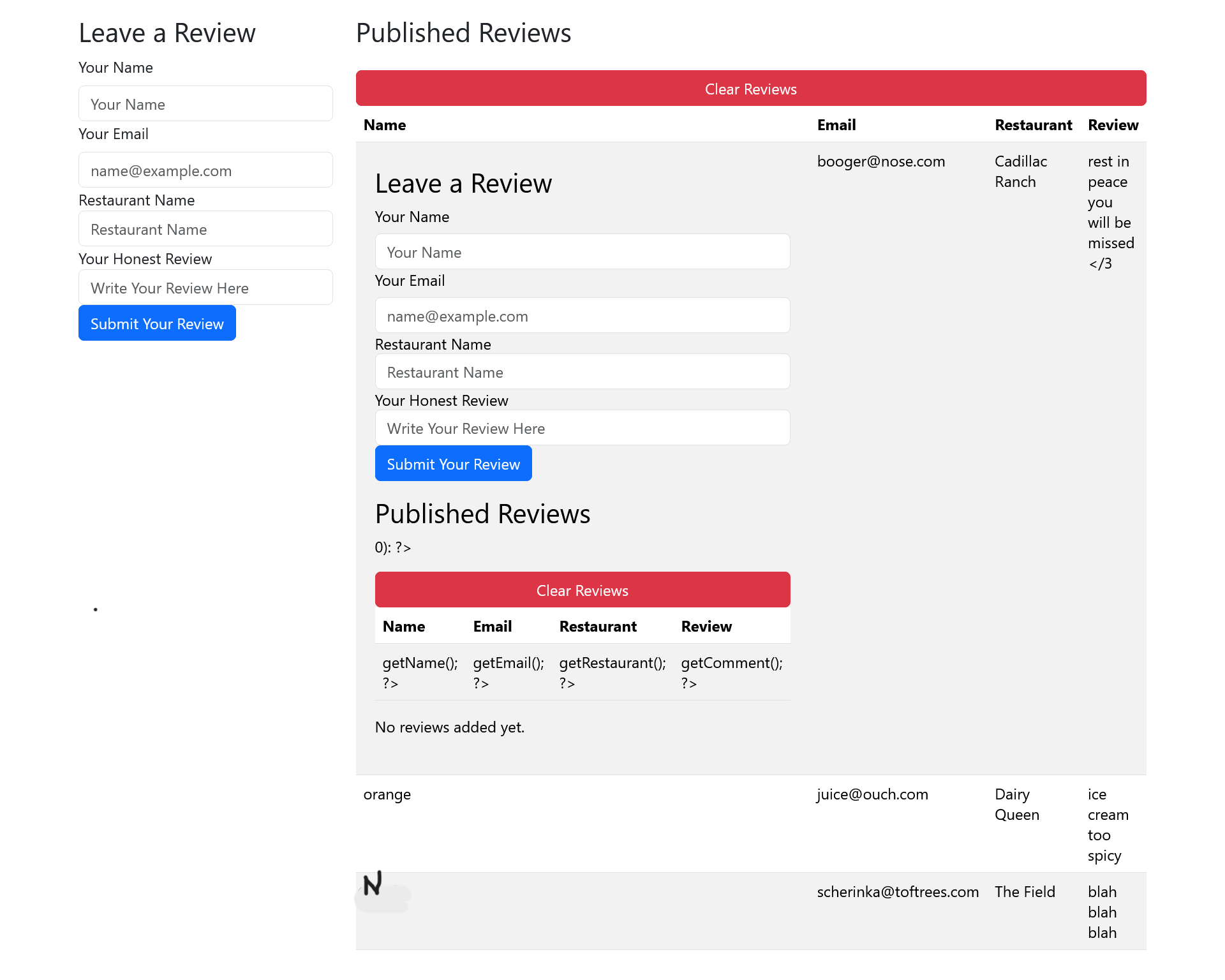
XSS is the kind of web security weakness that is near impossible for the average user to implement, as it relies on basic knowledge of HTML, and in more advanced circumstances, JavaScript, for it to be carried out. That being said, it’s important for victims of XSS attacks to recognize that what is happening to their system and their data is not the result of a virus, even though it could look that way. A virus works by duplicating files on the user’s hard drive until it is rendered unusable, whereas, while an XSS attack can make a page unusable for a user, their harddrive at large remains unaffected. Given that the Internet is open to any user of any varying degree of computer knowledge, this distinction is good to keep in mind when explaining more involved concepts to a less technological focused audience.

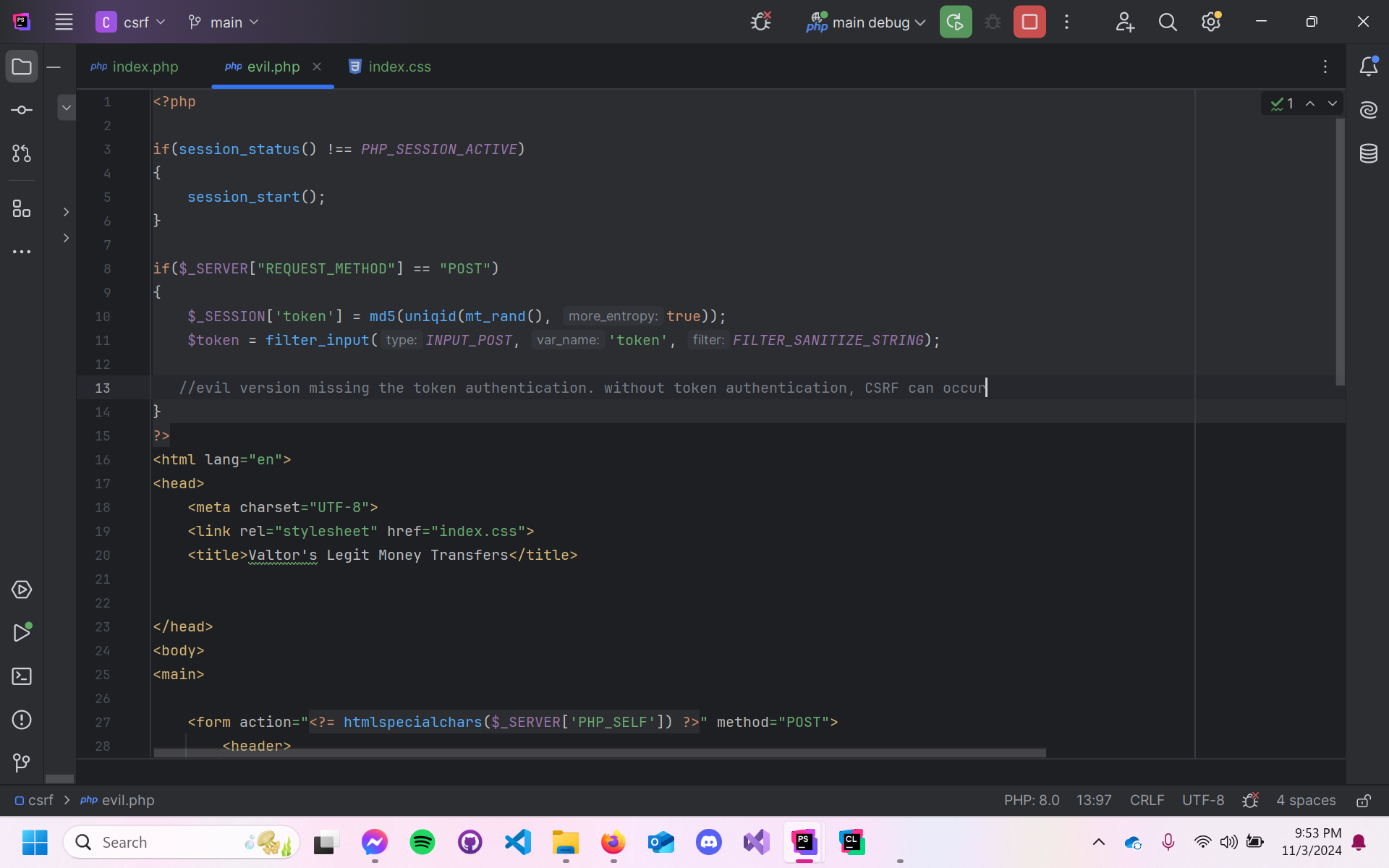
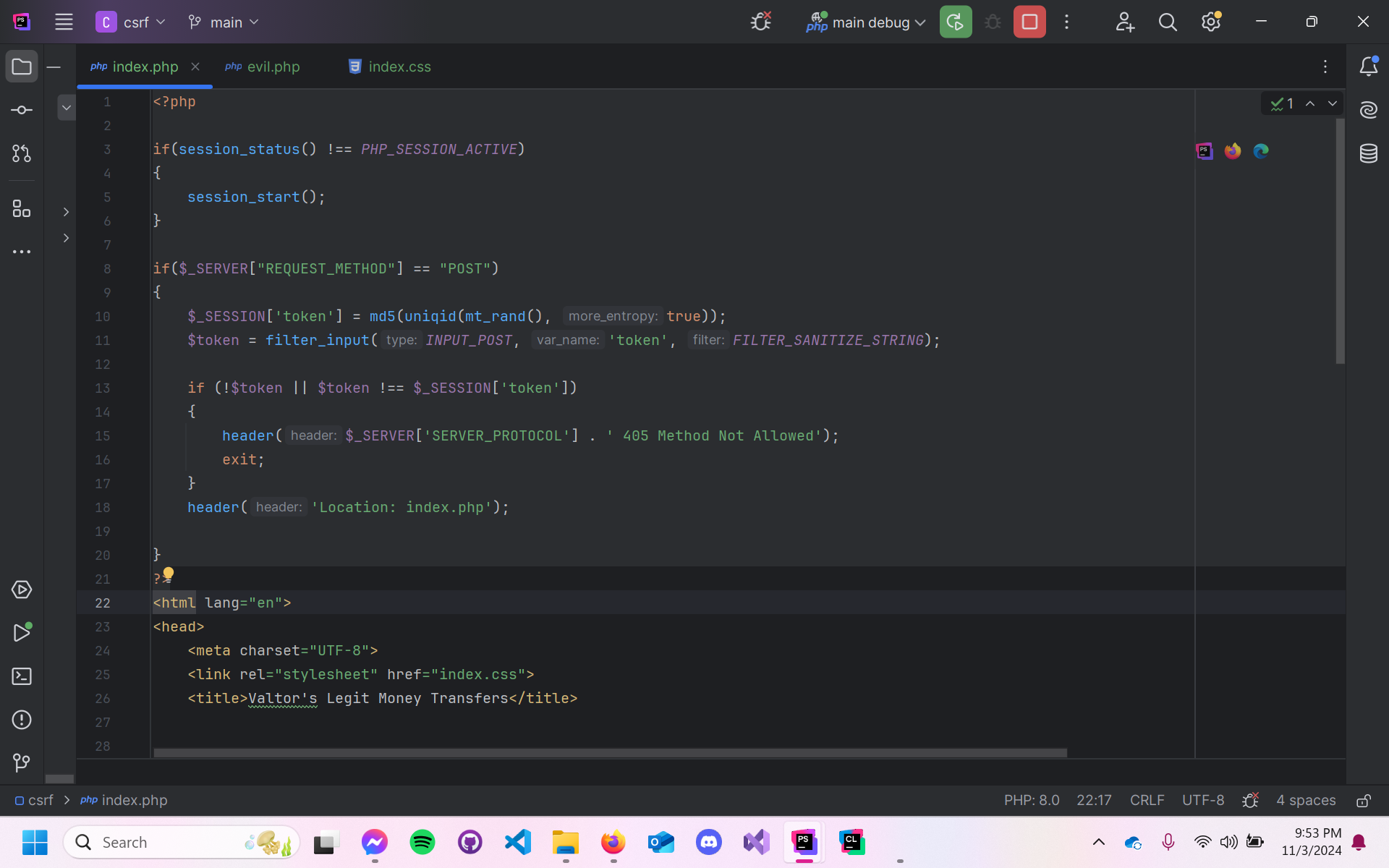
An XSS attack is implemented by a malicious actor taking advantage of an input field on a website not being cleansed or sanitized properly as to not allow or complete instruction of HTML or JavaScript. That said actor would then test if a site is weak to XSS attacks by inputting HTML or JavaScript to that field to see how the site handles it. The browser’s first instinct when encountering HTML or JavaScript is to simply execute the code, regardless of where in the DOM it is coming from. As long as the code is free of syntax errors, once it is inserted into an input field, it will carry out its written function. This code can do any number of things, from simply sending an alert to the user, or even making the form a user was inputting data into unusable. The most proven way of protecting input streams from malicious actors is to use the keyword *htmlentities* anywhere in the DOM where input is taken from the user. This sanitizes the input and ensures that the DOM will not execute any HTML or JavaScript code passed into the input field.

While there are slight differences within them, there are two main style of XSS attacks-- Reflected and Persistent/Stored. A reflected XSS attack is where an instant payout is shown to the user, as the code is instantly executed and what it does is visible to those on the client side, whether that just be the individual on a private page or a group viewing the same page. A reflected XSS vulnerability was seen on Twitch stream TASBot’s SMB3 stream, as viewers attempted inserting JavaScript into Twitch’s chat, and that code was then passed onto the pages every viewer of the stream. Luckily, this was a relatively harmless scenario, and TASBot found it more entertaining than unnerving, but it does illustrate well the room that was left for a more dangerous reflected XSS attack that could have happened on Twitch.

The more patient side of an XSS attack relies on implanting a piece of code for it to then execute on another user’s browser. Enactors of persistent XSS attacks implant their code in hopes that their seed will nurture a bigger payoff than a reflected attack could. This payoff often includes sensitive user information from server side databases including names, addresses, and even social security numbers of account holders. This is blind style of XSS is done with a piece of technology on the back end (XSS Hunter) for the malicious user that sends a message once their script has properly been seen. A real life example occurred when Tesla owner, Sam Curry, named his vehicle with a javascript tag. He didn’t notice anything immediately, and forgot he attempted to breach Tesla’s system until he sent his Tesla in for maintenance. XSS Hunter then returned to Curry the data of thousands of Tesla users, and the ability to modify that data including changing car details, customer payment history, and much more.

The following is an example of code written for this essay that does not properly include htmlentities to cleanse the input fields of a form, and how a malicious actor could alter the form as a result.

This shows an example of a form for inputting restaurant reviews into a database that did not have the inputs properly sanitized with htmlentities. Once a review with JavaScript was inserted into the form, the form was not able to send properly and could not function as intended, creating multiple instances of the form, and even showing backend methods for how the code is implemented. To quote the CWE on how to best protect against XSS “Protection against XSS is usually done by (1) safe output encoding, where HTML/JS code is not executed but just rendered plain text. and (2) by conducting input validation, rejecting inputs that potentially contain HTML/JS code.”

In relation, CSRF works on the front, like XSS, to manipulate what the user sees and how they interact with a website. Unlike XSS, CSRF can occur simply from a malicious actor creating a website similar in name and appearance to one trusted by users. By hiding input elements inside the HTML, malicious actors are able to essentially copy login credentials and other sensitive information from a user. The inherent functionality of web browsers relies upon the transfer of data, and that is done in multiple different ways, namely by Get (pulls data from the database) and Post (sends data to the database) requests. Another way CSRF is carried out is by the browser executing requests with no way of knowing which are going to be unsafe to the end user. Cross Site Request Forgery happens by intercepting these requests between clients and servers, which then gives the malicious actor access to private details of the user. The best way to prevent CSRF is by both ends of the site generating a unique token (a string of numbers and letters) for one time use that is only active long enough for the browser to execute the requests initiated by the user. Even if a malicious actor has a listener for tokens being generated by browser requests, they are going to be inactive by the time the malicious actor has access to them. The following are pieces of code examples written for this essay to show how tokens can be implemented and generated by request of the page, and how the browser is able to check that the token it is expecting is a match to the one it created when the request was made. The code on the right is insecure as it is missing the check for if the tokens are a match.

It is important to note that with every advancement in cybersecurity and web development, there will always be a constant arms race between malicious actors and developers. And although XSS and CSRF have had fluctuating positions on the OWASP’s list of top web insecurities, the actions of those who are both curious enough with enough ill-intent are able to raise these concerns back to the level they were nearly a decade ago. It is imperative that developers stay vigilant for mistakes in their code where a security leak could happen, and that users are not offering up their data to sites that are just familiar looking enough to be their bank.

# References

“Cross Site Scripting (XSS).” *Cross Site Scripting (XSS) | OWASP Foundation*, owasp.org/www-community/attacks/xss/. Accessed 3 Nov. 2024.

“Common Weakness Enumeration.” *CWE*, cwe.mitre.org/data/definitions/79.html. Accessed 3 Nov. 2024.

Cybr. “ Cross-Site Scripting (XSS) Explained in 7 Minutes .” *YouTube*, YouTube, 28 Nov. 2020, www.youtube.com/watch?v=DqK\_OYat-3M.

“Difference between XSS and CSRF.” *GeeksforGeeks*, GeeksforGeeks, 26 Aug. 2024, www.geeksforgeeks.org/difference-between-xss-and-csrf/.

fox\_haunterfox\_haunter 2911 silver badge66 bronze badges, and WouterWouter. “Why Don’t Antivirus Detect Malicious XSS Files?” *Information Security Stack Exchange*, 1 July 2020, security.stackexchange.com/questions/234284/why-dont-antivirus-detect-malicious-xss-files.

Kovacs, Eduard. “Tesla Awards Researcher $10,000 after Finding XSS Vulnerability.” *SecurityWeek*, 15 July 2019, www.securityweek.com/tesla-awards-researcher-10000-after-finding-xss-vulnerability/.

OWASP. “Cross Site Scripting Prevention Cheat Sheet¶.” *Cross Site Scripting Prevention - OWASP Cheat Sheet Series*, cheatsheetseries.owasp.org/cheatsheets/Cross\_Site\_Scripting\_Prevention\_Cheat\_Sheet.html. Accessed 3 Nov. 2024.

Phipps, Jenna. “What Is Cross-Site Scripting (XSS)? Types, Risks & Prevention.” *eSecurity Planet*, 30 Apr. 2024, www.esecurityplanet.com/networks/cross-site-scripting-xss/.

PwnFunction. “ Cross-Site Request Forgery (CSRF) Explained .” *YouTube*, YouTube, 5 Apr. 2019, www.youtube.com/watch?v=eWEgUcHPle0.

PwnFunction. “ Cross-Site Scripting (XSS) Explained .” *YouTube*, YouTube, 22 Mar. 2020, www.youtube.com/watch?v=EoaDgUgS6QA&t=5s.

Scott, Tom. “ Cracking Websites with Cross Site Scripting - Computerphile.” *YouTube*, YouTube, 23 Oct. 2013, www.youtube.com/watch?v=L5l9lSnNMxg.

Scott, Tom. “ Cross Site Request Forgery - Computerphile .” *YouTube*, YouTube, 3 Dec. 2013, www.youtube.com/watch?v=vRBihr41JTo.

Tonev, Borislav. “Cyber Attack Guide: Cross-Site Scripting.” *ScalaHosting Blog*, Scala Hosting Ltd., 5 Aug. 2024, www.scalahosting.com/blog/cyber-attack-guide-cross-site-scripting/.

Watkins, Peter. “Cross Site Request Forgeries.” *Wayback Machine*, 13 June 2001, web.archive.org/web/20020204142607/http:/www.tux.org/~peterw/csrf.txt.