Cross-site scripting (XSS) attacks are a type of injection attack where an attacker enters a script into a normal website (“Cross Site Scripting (XSS)”). Injection attacks involve a malicious user sending code into a system to do things like access data they should not be able to, execute unauthorized commands, or interfere with the site's operation (Lenaerts, “What is an Injection Attack?”). In a cross-site scripting attack, this code is then executed when another (nonmalicious) user accesses the site and becomes a victim of the attack. These attacks are effective because unsuspecting users have no reason to distrust the formerly trusted site, and the user’s browser believes that the injected code came from the original site so it has no reason not to run the script (“Cross Site Scripting (XSS)”). Cross-site scripting attacks can allow the attacker to access supposedly private information, like another user’s cookies, session tokens, or other information collected by the browser, including passwords (Lenaerts, “What is a Cross-Site Scripting (XSS) Attack?”). Two main types of cross-site scripting attacks are reflected XSS and persistent XSS. Reflected attacks involve the script executing when a malicious link is clicked or loaded, while persistent attacks permanently save the XSS script in a web application’s database (Lenaerts, “What is a Cross-Site Scripting (XSS) Attack?”). Cross-site scripting attacks are most commonly done in JavaScript, as JavaScript is used in most websites (“What is Cross-site Scripting and How Can You Fix it?”).

A use of cross-site scripting attacks is stealing private information. One way this can be accomplished is through the use of a keylogger script, injected into a legitimate website’s code. The attacker uses JavaScript to write a keylogger program that tracks the keys that a victim user presses and then sends the information that the user types to the attacker’s server using PHP. This keylogger program is injected into the code of the web application and will run whenever a victim user accesses the site after the attack. The victim user is unaware that this malicious script

has been injected into a legitimate website, and enters information like their username and password as normal, which the attacker now has access to. To execute this attack, the attacker must have its own server that it can use to receive this information.

A JavaScript keylogger has significantly more limitations than a typical software or hardware-based keylogger. Typical keyloggers allow a hacker to track every keystroke made on your device, and then save this information in a file (Lenaerts, "Keyloggers: How They Work & How to Detect Them.") JavaScript has limitations, which restrict the abilities of a JavaScript-based keylogger application. JavaScript cannot read or write arbitrary files on the hard disk, copy those files, execute programs, directly access operating system functionality, or access browser pages that come from a different site ("An Introduction to JavaScript"). As such, a JavaScript-based keylogger can only capture keystrokes made on the web page with the malicious script injected, and cannot access any keystrokes outside of that. A JavaScript-based keylogger could not access keystrokes made on a site without the script injected and also has no way of accessing keystrokes made outside of the browser.

A website can put different protection measures in place to prevent a JavaScript keylogger attack. One such measure a user could take is to disable JavaScript in their browser. This would prevent the attack from occurring as the malicious script would never execute. A protection measure the web application developer could take to specifically prevent a keylogger would be to disable keylogging by including code to remove event listeners. To prevent cross-site injection attacks generally, the developer could ensure inputs must be validated, sanitize data, and implement a content security policy ("What is cross-site scripting?"; The Jit Team and Dimkovski). Validating user input ensures that only 'good' input enters the system. This can be done by setting restrictions on what the user can include in a given input box, like

preventing certain special characters from being entered at all. Sanitizing data happens after the user input has been entered but before it is displayed. Sanitizing can encode special characters so that the server does not attempt to execute these scripts. One last common way of preventing cross-site scripting would be to implement a content security policy, which would only permit scripts from trusted sources to be executed.

Works Cited

"Cross Site Scripting (XSS)." *OWASP Foundation*, https://owasp.org/www-community/attacks/xss/. Accessed 14 November 2024.

"An Introduction to JavaScript." *The Modern JavaScript Tutorial*, 8 August 2022, https://javascript.info/intro. Accessed 14 November 2024.

The Jit Team, and Filip Dimkovski. "Step-by-Step Guide to Preventing JavaScript Injections." *Jit.io*, https://www.jit.io/resources/app-security/step-by-step-guide-to-preventing-javascript-injections. Accessed 14 November 2024.

Lenaerts, Bart. "Keyloggers: How They Work & How to Detect Them." *CrowdStrike*, 2 February 2023, https://www.crowdstrike.com/en-us/cybersecurity-101/cyberattacks/keylogger/. Accessed 14 November 2024.

Lenaerts, Bart. "What Is a Cross-Site Scripting (XSS) Attack?" *CrowdStrike*, 19 May 2023, https://www.crowdstrike.com/en-us/cybersecurity-101/cyberattacks/cross-site-scripting-xss/. Accessed 14 November 2024.

Lenaerts, Bart. “What Is an Injection Attack?” *CrowdStrike*, 3 May 2024, https://www.crowdstrike.com/en-us/cybersecurity-101/cyberattacks/injection-attack/. Accessed 14 November 2024.

“What is cross-site scripting?” *Cloudflare*, https://www.cloudflare.com/learning/security/threats/cross-site-scripting/. Accessed 14 November 2024.

“What is Cross-site Scripting and How Can You Fix it?” *Acunetix*, https://www.acunetix.com/websitesecurity/cross-site-scripting/. Accessed 14 November 2024.