**Cross site scripting (XSS)**

**Overview of Cross-site Scripting (XSS)**

Cross-site scripting that is commonly referred as XSS, occurs when hackers execute malicious scripting code within a victim’s browser. XSS is one of the most common application-layer web attacks. XSS is also one of the most common security vulnerabilities in software to date.

**What is Cross-site Scripting (XSS)?**

XSS is a type of web security vulnerability that allows an attacker to compromise the interactions that users have with a vulnerable application. XSS normally allows an attacker to masquerade as a victim user, to carry out any actions that the user is able to perform, and to access any of the user’s data, this is what makes XSS differ from other web attacks such as SQL Injections. The users of the web application are the ones at risk instead. One successful XSS attack can have devastating consequences for an online business’s reputation or its relation with its clients. In general, XSS attacks are easy to execute, but difficult to detect and prevent.

**How does Cross-site Scripting (XSS) work?**

XSS works by manipulating a vulnerable website so that it returns malicious script to users. As when the code executes inside a victim’s browser, the attacker can fully compromise their interaction with the application.

**XSS Types of Attacks**

XSS attacks can be broken down into three types which are stored, reflected and DOM-based.

Stored XSS is also known as persistent XSS. It arises when an application receives data from an untrusted source and includes that data within its later HTTP responses in an unsafe way. Stored XSS is the more damaging attack because:

* The payload is not visible for the browser’s XSS filter
* Users might trigger the payload accidentally if they visit the affected page, while a crafter URL or specific form inputs would be required for exploiting reflected XSS.

Besides that, reflected XSS is the simplest variety of XSS. It arises when an application receives data in an HTTP request and includes that data within the immediate response in an unsafe way. The script will be embedded into a link and is only activated when the link has been clicked on. By tricking the user to click the link, the script can carry out any action, and retrieve any data to which the user has access.

Finally, DOM-based XSS or also known as DOM XSS will arise when an application contains some client-side script that processes data from an untrusted source in an unsafe way, usually by writing the data back to the DOM. Typically, the input field would be populated from part of the HTTP request, such as a URL query string parameter allowing the attacker to deliver an attack using a malicious URL, in the same manner as reflected XSS.

**XSS Attack Examples**

Stored XSS:

While browsing an e-commerce website, a perpetrator discovers a vulnerability that allows HTML tags to be embedded in the site’s comments section. The embedded tags become a permanent feature of the page, causing the browser to parse them with the rest of the source code every time the page is opened. The attacker adds a comment with a URL. From that point, every time the page is accessed, the HTML tag in the comment will activate a JavaScript file, which is hosted on another site, and has the ability to steal visitor’s session cookies.

An attacker can compromise the visitor’s account by using the session cookie, granting easy access to personal information and credit card data. The visitor however may never have even scrolled down in the comments section, is not aware that the attack took place.

Reflected XSS:

While visiting a forum site that requires users to log in to their account, a perpetrator executes this search query <script type=’text/javascript’>alert(‘xss’);</script> causing to occur:

1. The query produces an alert box saying: “XSS”.
2. The page displays “<script type=’text/javascript’>alert(‘xss’);</script> not found”.
3. The page’s URL reads http://ecommerce.com?q=<script type=”text/javascript”>alert(‘XSS’);</script>.

This tells the attacker that the website is vulnerable. Then, will create his own URL. While the sending address and subject line may appear suspect to some, it does not mean that it won’t be clicked on. They will be taken to the forum’s website, where the malicious script will be reflected back to their browser, enabling the attacker to steal their session cookies and hijack their forum account.

DOM XSS:

If you send a HTTP request like http://www.example.com/test.html#<script>alert(1)</script>, your JavaScript code will get executed, because the page is writing whenever you typed in the URL to the page with document.write function. If you look at the source of the page, you will not see <script>alert(1)</script> because its all happening in the DOM and done by the executed JavaScript code.

After the malicious code has been executed by page, you can simply exploit this DOM based XSS vulnerability to steal the cookies from the user’s browser or change the behaviour of the page on the web application as you like.

**Preventing XSS Vulnerabilities**

Preventing XSS is very crucial in some cases but can be much harder depending on the complexity of the application and the ways it handles user-controllable data.

* Filtering input on arrival. Web applications must properly validate any input and remove malicious scripts.
* Encode data on output. User-controllable data is output in HTTP resources, encode the output to prevent it from being interpreted as active content.
* Use appropriate response headers. Preventing HTTP responses that are not intended to contain HTML or JavaScript. Links should generally be disallowed if they do not begin with a whitelisted protocol such as http:// or https://.
* Content Security Policy (CSP). CSP is a browser mechanism that aims to mitigate the impact of XSS-like behaviour. This is for the last line of defense, where can use CSP to reduce the severity of any XSS vulnerabilities that is still occurring.

# References

B. B. Gupta, P. C. (2020). *Cross-Site Scripting Attacks Classification, Attack, and Countermeasures.* CRC Press.

*Cross site scripting (XSS) attacks*. (2021). Retrieved from imperva: https://www.imperva.com/learn/application-security/cross-site-scripting-xss-attacks/

*Cross-Site Scripting (XSS) Attacks*. (2019, November 17). Retrieved from Securi Guides: https://sucuri.net/guides/what-is-cross-site-scripting/

*Cross-site scripting*. (2021). Retrieved from PortSwigger: https://portswigger.net/web-security/cross-site-scripting

Engin Kirda, N. J. (2009). Client-side cross-site scripting protection. *Computers & Security Volume 28, Issue 7*, 592-604.

KirstenS, Manico, J., Williams, J., Wichers, D., Smith, A., & Knutson, J. (2021). *Cross Site Scripting (XSS)*. Retrieved from OWASP: https://owasp.org/www-community/attacks/xss/

Singh, S. (2018, October 4). *5 practical scenarios for XSS Attacks*. Retrieved from Pentest-Tools Blog.com: https://pentest-tools.com/blog/xss-attacks-practical-scenarios/

Team, N. S. (2019, April 18). *The Cross-site Scripting (XSS) Vulnerability: Definition and Prevention*. Retrieved from Netsparker: https://www.netsparker.com/blog/web-security/cross-site-scripting-xss/