[https://portswigger.net/web-security/cross-site-scripting - what-is-cross-site-scripting-xss](https://portswigger.net/web-security/cross-site-scripting" \l "what-is-cross-site-scripting-xss)

<https://owasp.org/www-community/attacks/xss/>

<https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html>

What is it

* Allows an attacker to:
  + compromise the interactions that users have with vulnerable application
  + circumvent the same origin policy, which is designed to segregate different websites from each other
  + carry out any actions that the user can perform and access their data
  + install Trojan horse programs
  + modify content of a website
  + redirect users
* Occurs when an application uses input from a user within the output it generates without validating or encoding it

How does it work

* Done by injecting malicious JavaScript into a website and having it execute on another user’s browser.

Types of attacks

Reflected XSS

* The injected script is returned to the user by the server
* Application does not perform any other processing of the data besides the baseline, allowing the malicious actor to include JavaScript
* The result is a URL that contains a malicious script
* If this URL is visited by a user, the attacker’s script executes in the user’s browser
* The URL is typically delivered through an e-mail, another website, or some other form of social engineering

Stored XSS

* Malicious script comes from the website’s database
* Application receives data from an untrusted source, which is then included in an HTTP response later
* Data can be submitted via HTTP requests

DOM-based XSS

* Vulnerability exists in client-side code, such as setting innerHTML to what a user entered
* If an attacker can control the value of the input field, this can be manipulated to be a script

Code Examples

Tools

* Burp Suite’s Web Vulnerability Scanner
  + Finds most XSS vulnerabilities
* Nessus
* Nikto
* Veracode Dynamic Analysis

Testing

* Reflected & Stored XSS
  + Submitting unique inputs into every entry point
  + Identifying every location where the input from an HTTP response is used in the HTML output
  + Testing each location to determine whether an input can be used to execute JavaScript
* DOM-Based
  + Same process as above
  + Finding vulnerabilities in non-URL-based inputs, such as cookies or non-HTML-based sinks like set timeout is time-consuming
  + There is no substitute for reviewing JavaScript Code

Prevention

* Modern frameworks have a good bit of XSS prevention built-in, such as automatic encoding, auto-escaping, and templating, but it is important to understand how your framework mitigates these bugs and where it has gaps
* Ensure all variables go through validation and are then escaped or sanitized is known as perfect injection resistance
* Use appropriate headers in HTTP responses
  + Content-Type , X-Content-Type-Options can ensure that browsers interpret the responses properly
* Content Security Policy
  + Aims to hinder or prevent the exploitation of several vulnerabilities, including XSS.

Output Encoding

* Variables should not be interpreted as code instead of text
* Use your framework’s default output encoding protection
* Use an output encoding library, regularly update it
* Many different methods exist, as browsers parse HTML, JS, URLs and CSS differently

HTML Contexts

* Inserting a variable between HTML tags
* Use HTML entity encoding
* Use textContent if in JS, as it automatically encodes

HTML Attribute Contexts

* Inserting a variable in an HTML attribute
* Use quotes, as it makes it harder to change the context a variable operates in
* Use setAttribute and [attribute] in JS, as they automatically encode

Real World Example