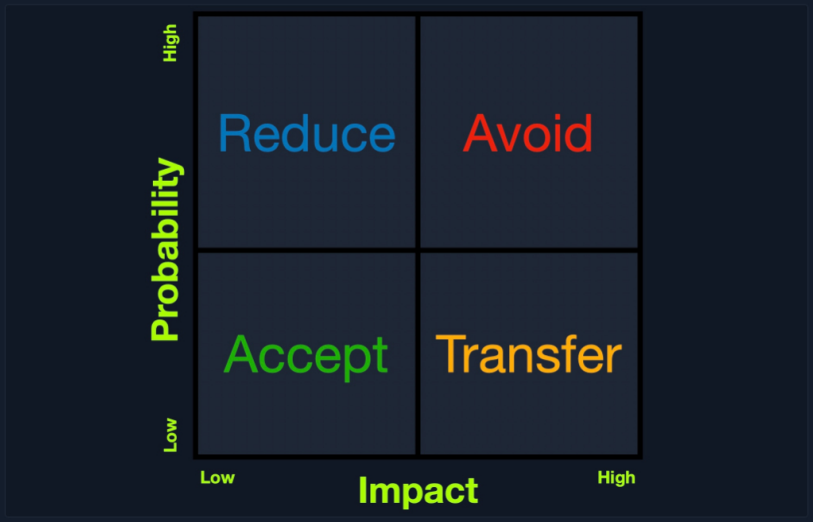
Introduction

As web applications become more advanced and more common, so do web application vulnerabilities. Among the most common types of web application vulnerabilities are Cross-Site Scripting (XSS) vulnerabilities. XSS vulnerabilities take advantage of a flaw in user input sanitization to "write" JavaScript code to the page and execute it on the client side, leading to several types of attacks.

What is XSS

A typical web application works by receiving the HTML code from the back-end server and rendering it on the client-side internet browser. When a vulnerable web application does not properly sanitize user input, a malicious user can inject extra JavaScript code in an input field (e.g., comment/reply), so once another user views the same page, they unknowingly execute the malicious JavaScript code.

XSS vulnerabilities are solely executed on the client-side and hence do not directly affect the back-end server. They can only affect the user executing the vulnerability. The direct impact of XSS vulnerabilities on the back-end server may be relatively low, but they are very commonly found in web applications, so this equates to a medium risk (`low impact + high probability = medium risk`), which we should always attempt to `reduce` risk by detecting, remediating, and proactively preventing these types of vulnerabilities.



XSS Attacks

XSS vulnerabilities can facilitate a wide range of attacks, which can be anything that can be executed through browser JavaScript code. A basic example of an XSS attack is having the target user unwittingly send their session cookie to the attacker's web server. Another example is having the target's browser execute API calls that lead to a malicious action, like changing the user's password to a password of the attacker's choosing. There are many other types of XSS attacks, from Bitcoin mining to displaying ads.

As XSS attacks execute JavaScript code within the browser, they are limited to the browser's JS engine (i.e., V8 in Chrome). They cannot execute system-wide JavaScript code to do something like system-level code execution. In modern browsers, they are also limited to the same domain of the vulnerable website. Nevertheless, being able to execute JavaScript in a user's browser may still lead to a wide variety of attacks, as mentioned above. In addition to this, if a skilled researcher identifies a binary vulnerability in a web browser (e.g., a Heap overflow in Chrome), they can utilize an XSS vulnerability to execute a JavaScript exploit on the target's browser, which eventually breaks out of the browser's sandbox and executes code on the user's machine.

XSS vulnerabilities may be found in almost all modern web applications and have been actively exploited for the past two decades. A well-known XSS example is the Samy Worm, which was a browser-based worm that exploited a stored XSS vulnerability in the social networking website MySpace back in 2005. It executed when viewing an infected webpage by posting a message on the victim's MySpace page that read, "Samy is my hero." The message itself also contained the same JavaScript payload to re-post the same message when viewed by others. Within a single day, more than a million MySpace users had this message posted on their pages. Even though this specific payload did not do any actual harm, the vulnerability could have been utilized for much more nefarious purposes, like stealing users' credit card information, installing key loggers on their browsers, or even exploiting a binary vulnerability in user's web browsers (which was more common in web browsers back then).

In 2014, a security researcher accidentally identified an XSS vulnerability in Twitter's TweetDeck dashboard. This vulnerability was exploited to create a self-retweeting tweet in Twitter, which led the tweet to be retweeted more than 38,000 times in under two minutes. Eventually, it forced Twitter to temporarily shut down TweetDeck while they patched the vulnerability.

To this day, even the most prominent web applications have XSS vulnerabilities that can be exploited. Even Google's search engine page had multiple XSS vulnerabilities in its search bar, the most recent of which was in 2019 when an XSS vulnerability was found in the XML library. Furthermore, the Apache Server, the most commonly used web server on the internet, once reported an XSS Vulnerability that was being actively exploited to steal user passwords of certain companies. All of this tells us that XSS vulnerabilities should be taken seriously, and a good amount of effort should be put towards detecting and preventing them.

Types of XSS

There are three main types of XSS vulnerabilities:

Type	Description

My Workstation

OFFLINE

Start Instance

∞ / 1 spawns left

Stored (Persistent) XSS	The most critical type of XSS, which occurs when user input is stored on the back-end database and then displayed upon retrieval (e.g., posts or comments)
Reflected (Non-Persistent) XSS	Occurs when user input is displayed on the page after being processed by the backend server, but without being stored (e.g., search result or error message)
DOM-based XSS	Another Non-Persistent XSS type that occurs when user input is directly shown in the browser and is completely processed on the client-side, without reaching the back-end server (e.g., through client-side HTTP parameters or anchor tags)

We will cover each of these types in the upcoming sections and work through exercises to see how each of them occurs, and then we will also see how each of them can be utilized in attacks.

Next ➜

Mark Complete & Next