Types of HTML Injection

1. Reflected HTML Injection
2. Stored HTML Injection

Reflected HTML Injection

In reflected HTML injection, the input is sent to the server through a request (e.g., via a URL parameter). The server processes the input and immediately reflects it in the response, which is sent back to the browser.
This type of injection is temporary and only affects the user who receives or interacts with the crafted URL.

Stored HTML Injection

In stored HTML injection, the malicious input is sent to the server and stored in a database or persistent storage. Later, this stored content is retrieved and displayed to other users in a web page.
This makes it more dangerous, as it can impact multiple users who access the affected content.

How to Identify HTML Injection

TestCase1:

Step 1: Check for Reflection

Crawl the application to identify parameters (GET, POST, etc.).

Test if the parameter input is reflected in the response by entering a unique string.

Use Ctrl + U to view the page source and confirm if the input appears in the HTML.

Step 2: Inject HTML Payloads

Use HTML tags like <h1>Test</h1>, bold, or broken tags such as , ", </h1>, etc.

This helps determine whether the injected HTML is being rendered or interpreted by the browser.

Step 3: Try Encoded Payloads

If the raw payload is filtered or escaped, try encoding it using:

- URL encoding
- HTML entity encoding
- Base64 encoding

Use tools like Burp Suite:

Go to the Decoder tab.

Paste the payload.

Encode it using the three formats above.

Use the encoded payload in the parameter value.

Step 4: Check the DOM (Optional)

If the injection is not visible in the HTML source, inspect the DOM using browser developer tools.

Sometimes the vulnerability exists in JavaScript-rendered content (DOM-based HTML injection).

Example Payloads

You can ask ChatGPT or use a cheat sheet to generate payloads.
Example payload to redirect users:

<meta http-equiv="refresh" content="0; url=https://example.com">

Additional Notes

In Reflected HTML Injection, issues may appear before or after login, depending on where the vulnerable parameters exist.

In Stored HTML Injection, the vulnerability is typically found after login, where user-generated content is submitted and displayed (e.g., profile fields, comments).

If you can't see a parameter name or input directly, try injecting a closing tag to break the HTML structure and reveal possible vulnerabilities.

How to Prevent HTML Injection

Escape special characters such as <, >, ", ', /, etc.

Sanitize user inputs to remove or encode unsafe HTML tags.

Use frameworks or libraries that auto-escape output (e.g., React, Angular).

Apply Content Security Policy (CSP) headers to reduce script execution risks.

Validate and encode output properly before rendering to HTML.