HTML injection

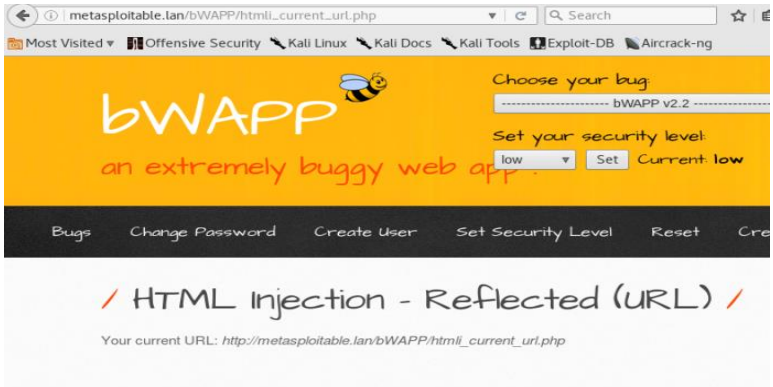
#Session_Task_11 by Sukhveer singh

1. HTML Injection –

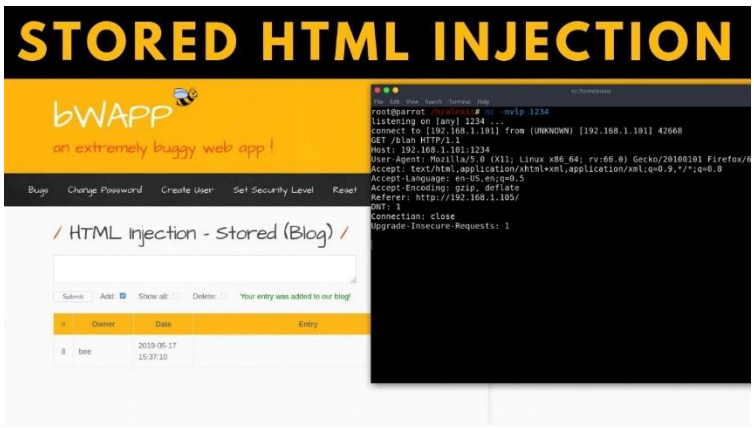
HTML Injection is a type of web security vulnerability where an attacker can inject malicious HTML code into a webpage. This can happen when a website doesn't properly sanitize or validate user inputs, such as form fields or URL parameters. When this malicious code gets executed by the web browser, it can lead to various consequences like defacement of the website, stealing of sensitive information, or even taking control of the user's session. To prevent HTML Injection, it's important for developers to properly validate and sanitize user inputs before displaying them on a webpage.

2. Types of HTML Injection –

- **Reflected HTML Injection:** In this type, the injected HTML code is reflected back to the user by the web application. It usually occurs when user inputs aren't properly validated or sanitized and are directly echoed back in the response without any encoding.



- **Stored HTML Injection (Persistent XSS):** This occurs when the injected HTML code is permanently stored on the web server, such as in a database. Whenever the affected webpage is accessed, the injected code gets executed. It's commonly found in comment sections, message boards, or user profiles.



- **DOM-based HTML Injection:** Unlike reflected and stored HTML Injection, this type doesn't involve the server-side processing. Instead, the malicious code gets executed by manipulating the Document Object Model (DOM) of the webpage on the client side. Attackers achieve this by exploiting vulnerabilities in client-side scripts, like JavaScript.

3. How html injection performed -

- **Identifying Input Fields:** Attackers first identify input fields on a webpage where user-provided data is accepted, such as search boxes, comment forms, or user profile fields.

- **Injecting Malicious Code:** They then input specially crafted HTML or JavaScript code into these fields. This code may include tags, attributes, or event handlers that can execute when the webpage is loaded or interacted with by other users.

- **Submitting the Input**: After injecting the malicious code, attackers submit the input to the web application. Depending on the vulnerability, the injected code may be reflected back immediately

(in the case of reflected HTML Injection) or stored on the server for later execution (in the case of stored HTML Injection).

- **Execution of the Code:** When the webpage is rendered and the injected data is displayed to other users, the malicious code gets executed in their browsers. This can lead to various consequences, such as altering the appearance of the webpage, stealing sensitive information, or executing unauthorized actions on behalf of the user.

4. **How to test against html injection** –

- **Identify Input Fields: -** First, identify all input fields in the web application where user-provided data is accepted and displayed on a webpage. This includes search boxes, comment forms, user profile fields, and any other forms.

- **Inject Test Payloads:** - Input specially crafted HTML or JavaScript payloads into these input fields. These payloads should include tags, attributes, or event handlers that could potentially execute when rendered on the webpage.

- **Observe Output**: - Submit the input containing the payloads and observe how the application handles them. Check whether the injected code gets reflected back immediately on the webpage (reflected HTML Injection) or stored for later execution (stored HTML Injection).

- **Analyze Responses: -** Analyze the responses from the server to see if the injected code is properly sanitized or filtered. Look for any signs of vulnerability, such as unescaped characters or unexpected behavior in the rendered output.

- **Test for DOM-based Injection: -** Additionally, test for DOM-based HTML Injection by manipulating client-side scripts, such as JavaScript, to see if the injected code affects the Document Object Model (DOM) and leads to unintended consequences.

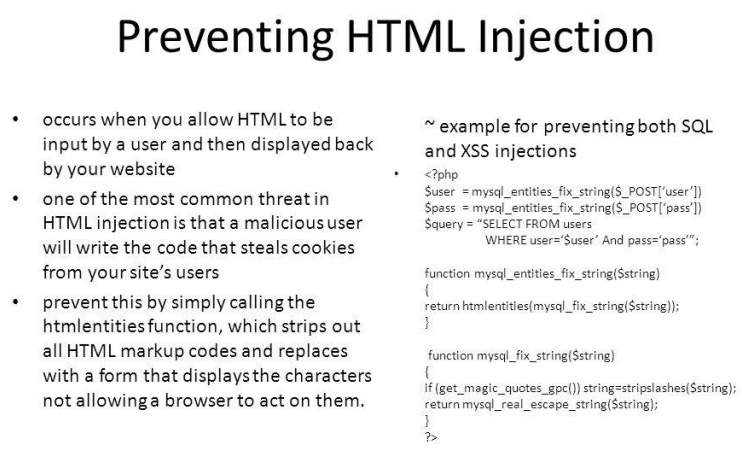
  Repeat and Expand: Repeat the testing process for all input fields and explore different scenarios and payloads to ensure thorough

coverage. Also, consider testing edge cases and boundary conditions to uncover any hidden vulnerabilities.

- **Use Automated Tools:** - Utilize automated security testing tools designed for detecting HTML Injection vulnerabilities. These tools can help identify potential vulnerabilities more efficiently and comprehensively.

- **Document Findings:** - Document all findings, including vulnerable input fields, the payloads used, and the observed behaviors. Provide detailed information to developers for timely remediation.

5. How to Prevent HTML Injection –



Input Validation and Sanitization: -

Validate and sanitize all user inputs before displaying them on webpages. Use input validation techniques to ensure that only expected data formats are accepted. Sanitize user inputs by removing or encoding any HTML or JavaScript code to prevent execution.

Avoid Dynamic Code Generation: -

Minimize the use of dynamic code generation techniques, such as eval(), document.write(), and innerHTML. These methods can inadvertently

execute injected code. Instead, use safer methods for dynamically generating content, such as createElement() and appendChild().

Use Frameworks and Libraries: -

Employ secure web frameworks and libraries that offer built-in protection against HTML Injection vulnerabilities. Frameworks like React, Angular, and Vue.js provide mechanisms for handling user input safely and preventing injection attacks.

Content Security Policy (CSP): -

Implement Content Security Policy (CSP) headers on web servers to control which resources can be loaded and executed on webpages. Configure CSP directives to restrict the use of inline scripts, styles, and other potentially vulnerable content.

Escape User Inputs: -

Escape user inputs properly before rendering them on webpages. Use encoding techniques, such as HTML entity encoding or JavaScript escaping, to neutralize special characters and prevent them from being interpreted as code.

Regular Security Audits: -

Conduct regular security audits and code reviews to identify and address potential HTML Injection vulnerabilities. Utilize automated security testing tools to scan for vulnerabilities and ensure compliance with secure coding practices.

Educate Developers and Users: -

Educate developers about the risks of HTML Injection and the importance of secure coding practices. Encourage them to stay updated on the latest security threats and mitigation techniques. Additionally, educate users about safe browsing habits and the risks associated with interacting with untrusted websites.

6. Comparison with other Attacks –

- HTML Injection and XSS both involve injecting and executing malicious code in the context of a webpage, but HTML Injection focuses on injecting HTML or JavaScript directly into the webpage,

while XSS involves injecting scripts that execute in the context of other users' browsers.

- SQL Injection, on the other hand, focuses on manipulating SQL queries to gain unauthorized access to database information, which is different from manipulating the webpage itself.

- All three vulnerabilities can have serious security implications and should be addressed with proper security measures, including input validation, output encoding, and parameterized queries.

7. References –

https://www.softwaretestinghelp.com/html-injection-tutorial/
https://medium.com/@prathameshghumade/understanding-and-preventing-html-injection-attacks-732ffafb0bfa
https://www.imperva.com/learn/application-security/html-injection/
https://stackoverflow.com/questions/20855482/preventing-html-and-script-injections-in-javascript
https://www.imperva.com/learn/application-security/html-injection/#:~:text=Stored%20HTML%20injection%2C%20also%20known,they%20access%20a%20particular%20page.