AI-Powered Security Policy Report

Generated: 2025-11-07 23:47:46

Total Vulnerabilities Scanned: 10
• SAST: 10
• SCA: 0
• DAST: 0

LLM Models Used:

• LLaMA 3.3 70B (Groq) - SAST/SCA
• LLaMA 3.1 8B Instant (Groq) - DAST

Policy #1: SAST

Title: Explicit Unescape
Severity: HIGH
LLM: LLaMA 3.3 70B

■ Understanding the Issue

What is Explicit Unescape?
Explicit Unescape is a security vulnerability that happens when a web application doesn't properly handle user input. Imagine you're writing a letter to a friend, and you want to include a quote from someone else. You would put the quote in quotation marks to show it's not your words, right? In web development, there are similar ways to show that some text is special or should be treated differently. However, if an application doesn't properly "escape" or mark these special texts, it can lead to security issues. In this case, the vulnerability is in a Mustache template, which is a way to fill in templates with dynamic data.

Why is it Dangerous?
Think of it like this: An attacker could send a malicious message that looks like a normal quote but actually contains harmful code. If the application doesn't properly escape this input, it could execute the harmful code, leading to a Cross-Site Scripting (XSS) attack. This is similar to someone sending you a letter with a hidden message that, when read aloud, gives them access to your house keys.

Real-World Impact:
The real-world impact could be severe. An attacker could use this vulnerability to steal user data, take over user accounts, or even deface the website. It's like leaving your house unlocked; anyone can come in and do whatever they want.

■ How to Fix It

Current Code (Vulnerable):
```javascript
// This line is vulnerable because it directly outputs user input without proper escaping
template = '{{{ userInput }}}';
```
In this example, `userInput` is directly inserted into the HTML template without being escaped. This allows an attacker to inject malicious code.

Fixed Code (Secure):
```javascript
// This line is secure because it properly escapes the user input
template = '{{ userInput }}';
```
By using `{{ }}` instead of `{{{ }}}`, we ensure that the `userInput` is properly escaped, preventing any malicious code from being executed.

Step-by-Step Fix:
1. **Identify Vulnerable Code**: Look through your codebase for any instances where user input is directly inserted into templates without proper escaping.
2. **Understand Mustache Syntax**: Learn about Mustache template syntax, especially the difference between `{{ }}` and `{{{ }}}`. The triple braces `{{{ }}}` are used for unescaped HTML, which can be dangerous if used with user input.
3. **Replace Vulnerable Code**: Replace any instances of `{{{ }}}` with `{{ }}` when dealing with user input. This will ensure that the input is properly escaped and cannot be used for XSS attacks.
4. **Test for XSS**: After fixing the code, test it to ensure that XSS attacks are no longer possible. You can do this by trying to inject malicious scripts and verifying that they do not execute.

■ Testing Your Fix

How to Verify:
1. **Inject Malicious Script**: Try to inject a simple script tag into the user input field. For example, you could try entering `alert('XSS')`.
2. **Verify No Execution**: If your fix is correct, the script should not execute. Instead, you should see the script tags as plain text on the page, indicating that they were properly escaped.

Test Cases:
```javascript
// Example test case in JavaScript
const userInput = 'alert("XSS")';
const template = '{{ userInput }}';
// Render the template with the user input
const renderedTemplate = Mustache.render(template, { userInput });
console.log(renderedTemplate);
// The output should be plain text, not executing the script
```

■ Learn More

- OWASP Guide: [OWASP Cross-Site Scripting (XSS)](https://owasp.org/www-project-top-ten/2017/A7_2017-Cross-Site_Scripting_(XSS))
- Tutorial: [Mustache Template Syntax](https://mustache.github.io/mustache.5.html)

- Best Practices: [OWASP Secure Coding Practices](https://owasp.org/www-project-secure-coding-practices/)

■ Compliance

This fix helps meet:
- NIST CSF: ID.AM (Asset Management), PR.DS (Data Security)
- ISO 27001: A.8.23 (Web filtering), A.8.8 (Management of technical vulnerabilities), A.8.28 (Secure coding)

■ Timeline

Priority: HIGH
Estimated Time: 2 hours
Deadline: 1 day

Remember: Security is a learning process. Don't hesitate to ask for help! If you're unsure about any part of this process, consult with a security expert or a senior developer.