

AI-Powered Security Policy Report

Generated: 2025-11-07 23:48:30

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 type of security vulnerability that occurs when user input is not properly sanitized, allowing an attacker to inject malicious code into a web application. Think of it like a text message that's supposed to be just plain text, but someone finds a way to add a link or an image that can harm your phone. In this case, the vulnerability is in a template engine called Mustache, which is used to display dynamic content on a website.

Why is it Dangerous?

Imagine you're at a coffee shop, and someone puts a malicious sticker on your laptop that looks like a harmless advertisement. When you click on it, it installs malware on your computer. Similarly, an attacker could exploit the Explicit Unescape vulnerability to inject malicious JavaScript code into a website, which could then be executed by unsuspecting users. This could lead to a range of problems, including stolen sensitive information, account takeovers, or even malware infections.

Real-World Impact:

If an attacker exploits this vulnerability, it could result in a data breach, where sensitive user information like passwords, credit card numbers, or personal data is stolen. It could also lead to account takeovers, where an attacker gains unauthorized access to a user's account and can perform malicious actions. In severe cases, it could even lead to malware infections, where an attacker installs malicious software on a user's device.

■ How to Fix It

```
### Current Code (Vulnerable):
```javascript
// app/views/profile.js:42
// This line is vulnerable to Explicit Unescape
template = Mustache.render(template, { name: userInput });
// The userInput variable is not sanitized, allowing an attacker to inject malicious code
```

```

Fixed Code (Secure):

```
```javascript
// app/views/profile.js:42
// Sanitize the userInput variable to prevent Explicit Unescape
const sanitizedInput = encodeURIComponent(userInput);
template = Mustache.render(template, { name: sanitizedInput });
// By encoding the userInput variable, we prevent an attacker from injecting malicious code
```

```

Step-by-Step Fix:

1. **Identify the vulnerable code**: Locate the line of code that's using the Mustache template engine to render user input.
2. **Sanitize the user input**: Use a function like `encodeURIComponent()` to encode the user input and prevent any malicious code from being injected.
3. **Update the template rendering**: Pass the sanitized user input to the Mustache template engine instead of the raw user input.
4. **Test the fix**: Verify that the vulnerability is no longer exploitable by testing the updated code with malicious input.
5. **Review and refactor**: Review the entire codebase to ensure that similar vulnerabilities don't exist elsewhere and refactor the code to follow secure coding best practices.

■ Testing Your Fix

How to Verify:

1. **Test with malicious input**: Attempt to inject malicious JavaScript code into the user input field to verify that the vulnerability is no longer exploitable.
2. **Verify the output**: Check that the sanitized user input is correctly rendered in the template and that no malicious code is executed.

Test Cases:

```
```javascript
// Test case 1: Malicious input
const maliciousInput = 'alert("XSS")';
const sanitizedInput = encodeURIComponent(maliciousInput);
template = Mustache.render(template, { name: sanitizedInput });
console.log(template); // Should not execute the malicious script

// Test case 2: Valid input
const validInput = 'John Doe';
const sanitizedInput = encodeURIComponent(validInput);
template = Mustache.render(template, { name: sanitizedInput });
console.log(template); // Should render the valid input correctly
```

```

...

■ Learn More

- OWASP Guide: [OWASP Cross-Site Scripting (XSS)](<https://owasp.org/www-community/attacks/xss/>)
- Tutorial: [Mustache Template Engine Tutorial] (<https://mustache.github.io/#demo>)
- 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 the fix, consult with a senior developer or a security expert.