**Cross-Site Scripting: Poor Validation Development Mitigation SOP**

XSS vulnerabilities occur when a Web Application includes untrusted data in its output. Poor validation allows some cross-site scripting attacks to still occur despite trying to filter out malicious content. Based on the content of the data format, black-listing certain HTML characters may not be possible.

**Defense Against Cross-Site Scripting: Poor Validation**

To prevent the browser executing malicious code if the validation does not properly filter out bad data, the use of the <c:out> tag along with the escapeXML equal to true, which is the default behavior for the attribute is the best way to mitigate poor validation. If you are adding input from an untrusted source, escape it with a <c:out> tag always. Despite it being the default value, it is still good practice to declare the value for the escapeXML attribute.

**Example**

<div>${contentionTitle}</div>

**Explanation**

The example above displays the unfiltered content from the request

All data pulled from sources trusted or untrusted need to be validated before applied. Apply the <c:out> around the code. It is good practice to not log sensitive information, such as PII.

**Recommendation**

<div><c:out value="${contentionTitle}" escapeXml="true"/></div>

This will prevent any XML in contentionTitle from being treated by the browser as markup, which mitigates the potential vulnerability.

Remember to refer to the tag library whenever it is used, even if the parent page has already called the library. Since they can be compiled separately, this is a good practice to prevent cross-site scripting.

**Important Note**

The use of certain encoding constructs, such as the <c:out/> tag with the escapeXml="true" attribute (the default behavior), will prevent some, but not all cross-site scripting attacks. Depending on the context in which the data appear, characters beyond the basic <, >, &, and " that are HTML-encoded and those beyond <, >, &, ", and ' that are XML-encoded may take on meta-meaning. Relying on such encoding constructs is equivalent to using a weak blacklist to prevent cross-site scripting and might allow an attacker to inject malicious code that will be executed in the browser. Because accurately identifying the context in which the data appear statically is not always possible, Fortify Static Code Analyzer reports cross-site scripting findings even when encoding is applied and presents them as Cross-Site Scripting: Poor Validation issues.

**References**

1. [OWASP – Cross-Site Scripting: DOM](https://www.owasp.org/index.php/DOM_Based_XSS)
2. [HP - Cross-Site Scripting: Poor Validation](https://vulncat.fortify.com/en/detail?id=desc.dataflow.java.cross_site_scripting_poor_validation#Java%2fJSP)
3. [OWASP Cheat Sheet for XSS Prevention](https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html)
4. [OWASP Encoded Attack Threat Countermeasures](https://owasp.org/www-pdf-archive/Encoded_Attacks_Threats_Countermeasures_9_30_08.pdf)