New Vulnerability Discovered in ESAPI's AntiSamy Policy File, antisamy-esapi.xml

Kevin W. Wall <kevin.w.wall@gmail.com>

Summary

Category:	Improper sanitization of user-controlled input permitted by an incorrect regular expression in an ESAPI configuration file can result in that input being unintentionally executing javascript: URLs, resulting in Cross-Site Scripting (XSS).
Module:	ESAPI Validator. Specifically, the various methods Validator.getValidSafeHTML() and Validator.isValidSafeHTML() are vulnerable.
Announced:	2022-04-17 in the release notes for ESAPI 2.3.0.0. (https://github.com/ESAPI/esapi-java-legacy/blob/develop/documentation/esapi4java-core-2.3.0.0-release-notes.txt)
Credits:	Kevin W. Wall / Sebastian Passaro and serendipity
Affects:	All versions of ESAPI 2.x prior to 2.3.0.0 (released on 2022-04-17) and all versions of ESAPI 1.4 (no longer supported) if you are using the default antisamy-esapi.xml policy file and any of the Validator.getValidSafeHTML() and Validator.isValidSafeHTML() methods accepting user-controlled input and that input is eventually rendered as HTML.
Details:	See the remainder of this write-up.
GitHub Issue #:	None.
Related:	None.
CWE:	CWE-79 ("Improper Neutralization of Input During Web Page Generation ('Cross-Site Scripting')
CVE Identifier:	CVE-2022-xxxxx
CVSS Severity (version 3.1) - Estimated	CVSS v3.1 Base Score: 6.1 (Medium) Impact Subscore: Unassigned Exploitability Subscore: Unassigned CVSS Vector CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:C/C:L/I:L/A:N

		(This is very similar to the AntiSamy CVE, CVE-2021-35043, which has the same CVSS vector. The only different here is that this has an even more trivial attack complexity.)

Background

OWASP ESAPI (the OWASP Enterprise Security API) is a free, open source, web application security control library that makes it easier for programmers to write lower-risk applications. The ESAPI for Java library is designed to make it easier for programmers to retrofit security into existing applications. ESAPI for Java also serves as a solid foundation for new development.

One of the security controls provided by ESAPI for Java is defense against Cross-Site Scripting (XSS). This XSS defense is primarily provided via the mechanism of the ESAPI Encoder interface, but also exists as the ESAPI Validator interface, specifically the 2 methods Validator.getValidSafeHTML() and Validator.isValidSafeHTML(). It is precisely these latter 2 methods that are affected by this particular vulnerability.

Both of these methods (actually, there are 4 of them in total; 2 variations of each) first check to see if the user-controlled input exceeds a maximum length and checked if null, and then the canonicalized input is checked against AntiSamy sanitization via ESAPI's AntiSamy policy file. For this particular CVE, there was an error in the default ESAPI antisamy-esapi.xml policy file that AntiSamy uses.

This configuration error dates back to at least the ESAPI 1.4 release and was only fixed in the 2.3.0.0 release. This error mistakenly allowed "javascript:" pseudo-URLs to be passed into both the Validator.getValidSafeHTML() and Validator.isValidSafeHTML() methods and treated as safe input, but such "javascript:" URLs are obviously not safe and may lead to XSS vulnerabilities in the application code that uses them.

Problem Description & Explanation of Cause

Going back to ESAPI 1.4, ESAPI's default AntiSamy policy file, called "antisamy-esapi.xml", had an incorrect regular expression that (presumably) accidentally allowed the ":" character as a part of the "onsiteURL". That meant there was an opportunity for user-controlled input containing "javascript:" URLs to be recognized as "safe" rather than being an XSS payload.

So the question is, "am I vulnerable?". You are if your application satisfies these 2 conditions:

1. You are using the one of the 2 Validator methods in ESAPI that use AntiSamy (Validator.getValidSafeHTML() or Validator.isValidSafeHTML()) and there is a way for user-controlled input that is supposed to be sanitized to go through

one of those two methods and the result is that said user-controlled input is then rendered in a browser.

AND

2. You have your antisamy-esapi.xml policy file configured in a vulnerable way that allows “javascript:” URLs. That is, if “onsiteURL” regular expression includes (directly or via an implied character range), the colon character as well as alphabetic characters.

The “onsiteURL” regular expression in antisamy-esapi.xml files dating back to ESAPI 1.4 that was found to be faulty was:

```
"([\w\\/\.\?=&amp;;\#-~]+|\#(\w)+)"     <== Vulnerable version; the ‘~’ makes it fail
```

In ESAPI 1.3, it was:

```
"([\w\\/\.\?=&amp;;\#-]+|\#(\w)+)"      <== This version was okay
```

Had the ‘~’ that was added to the character class some time during the 1.4 release been added at the *end* instead of *before* the ‘-’ and the ‘-’ would have remained as the last element, and things would have been fine. But when written as ‘\#-~’ within a character class, that results into accepting all ASCII hex values between ‘#’ (0x23) and ‘~’ (0x73), and that range happens to include ‘:’ (0x58). Thus the regular expression in its entirety allowed “javascript:” to be accepted as a “safe” string and the sanitization failed.

Please note: This vulnerability is independent of the actual ESAPI jar that you use.

Impact

If your application using ESAPI is vulnerable, it may allow a user-controlled input that causes a “javascript:” URL to not be properly sanitized, resulting in either reflected or possibly persistent (if the input is stored) XSS attacks.

Remediation

Besides updating to use ESAPI 2.3.0.0 (which in itself is not necessary to remediate this particular CVE, not is it sufficient), one must *either*:

1. Download an ESAPI “configuration jar” from the GitHub releases link associated with ESAPI 2.3.0.0 or later (e.g., esapi-2.3.0.0-configuration.jar) and extract the “configuration/esapi/antisamy-esapi.xml” file from the jar and use that to replace your application’s AntiSamy policy file (generally named “antisamy-esapi.xml”).

OR

2. Locate your application’s AntiSamy policy file (typically “antisamy-esapi.xml”).

As noted in the ESAPI 2.3.0.0 release notes, the portion that changed in the release notes was that we updated 3 regular expressions in the '<common-regexps>' node for our antisamy-esapi.xml file to reflect the latest regex values from AntiSamy's antisamy.xml configuration file in their official AntiSamy 1.6.7 release.

The regular expression for “onsiteURL” was found to be vulnerable was updated and changed to this

The original (vulnerable) line will look like:

<regexp name="onsiteURL" value="([\w\\/\.\?=&;\#-~]+|\#(\w)+)"/>

The corrected line should look like:

<regexp name="onsiteURL" value="^(?!//)(?![\p{L}\p{N}\\\.\#@\$%\+&;\-_~,\?=/!]*(&colon))[\p{L}\p{N}\\\.\#@\$%\+&;\-_~,\?=/!]*"/>

The original (possibly vulnerable???) regular expression values for htmlTitle and offsiteURL:

<regexp name="htmlTitle" value="[a-zA-Z0-9\s-_',:\[\]!\./\\\(\)]*"/>

<regexp name="offsiteURL" value="(\s)*((ht|f)tp(s?)://|mailto:)[A-Za-z0-9]+[~a-zA-Z0-9-_\.@#$%&;:,\?=/\+!]*(\s)*"/>

The updated regular expression values for them:

<regexp name="htmlTitle" value="[\p{L}\p{N}\s\-_',:\[\]!\./\\\(\)&]*"/>

<regexp name="offsiteURL" value="(\s)*((ht|f)tp(s?)://|mailto:)[\p{L}\p{N}]+[\p{L}\p{N}\p{Zs}\.\#@\$%\+&;:\-_~,\?=/!\(\)]*(\s)*"/>

Note that the regular expressions for “htmlTitle” and “offsiteURL” were updated as precautionary measures, because both of them have an unquoted ‘-’ in the middle of a character class surrounded by other special characters. That’s always going to be a red flag to me even if we haven’t found any specific XSS payload that is exploitable. Instead, it seemed prudent to just accept AntiSamy’s vetted regular expressions from their antisamy.xml policy file.

Testing for Vulnerability

Software Composition Analysis tools that look for vulnerabilities in an application's direct and transitive dependencies are going to report this CVE in versions of ESAPI prior to 2.3.0.0. That's fine. Just know that updating to the esapi-2.3.0.0.jar alone is not going to remediate this issue.

And since some applications using ESAPI have certainly tweaked their antisamy-esapi.xml file or outright replaced it with one of their own, the remediation may be more complicated than even swapping out your current antisamy-esapi.xml file with the one extracted from esapi-2.3.0.0-configuration.jar.

So, what can you do to test it? There are 2 approaches. One is to run semgrep using the new rule written by @lapt0r at https://semgrep.dev/s/returntocorp:esapi-antisamy-config. This will look through your XML files for the node '<anti-samy-rules>' and then examine the regular expressions for "htmlTitle", "onsiteURL", and "offsiteURL" to see if they match the antisamy-esapi.xml regular expressions from ESAPI 2.3.0.0. If not, a warning is displayed to briefly explain the issue.

Of course, if you customized your antisamy-esapi.xml regular expressions for these regex names, the semgrep rule may not work for you if you've intentionally changed them. In that case, a better test would be to just write your JUnit test and run it using your current production ESAPI.properties and antisamy-esapi.xml configuration files and this unit test case:

```
@Test
public void testJavaScriptURL() throws Exception {
    String expectedSafeText = "This is safe from XSS. Trust us!";
    String badVoodoo = "<a href=\"javascript:alert(1)\">" + expectedSafeText + "</a>";
    Validator instance = ESAPI.validator();
    ValidationErrorList errorList = new ValidationErrorList();
    String result = instance.getValidSafeHTML("test", badVoodoo, 100, false, errorList);
    assertEquals( expectedSafeText, result );
}
```

which is the test from ESAPI 2.3.0.0 branch that verifies this issue is fixed. For details, see: https://github.com/ESAPI/esapi-java-legacy/blob/2.3.0.0/src/test/java/org/owasp/esapi/reference/validation/HTMLValidationRuleCleanTest.java#L137-L147. The expectation here is if things are configured correctly for you, then ESAPI, via AntiSamy, will strip out all the mark-up leaving only the "This is safe from XSS. Trust us!" portion that is indeed safe.

Acknowledgments

Sebastian Passaro (@spassarop / sebastian.passaro@owasp.org) – for pinpointing the cause as the faulty “onsiteURL” regular expression and reviewing this security bulletin and providing feedback.

Kurt Borgerg (@lapt0r / kurt@r2c.dev) – for providing the semgrep rule.

Matt Seil (@xeno6696 / xeno6696@gmail.com) - for reviewing this security bulletin.

References

https://nvd.nist.gov/vuln/detail/CVE-2021-35043 – A very similar CVE in its attack complexity and impact that is also related to AntiSamy use.