




 Secrets


 ABAP


 Apex


 C


 C++


 CloudFormation


 COBOL


 C#


 CSS


 Flex


 Go


 HTML


 **Java**


 JavaScript


 Kotlin


 Objective C


 PHP


 PL/I


 PL/SQL


 Python


 RPG


 Ruby


 Scala


 Swift


 Terraform


 Text


 TypeScript

 T-SQL

 VB.NET

 VB6

 XML



Java static code analysis

Unique rules to find Bugs, Vulnerabilities, Security Hotspots, and Code Smells in your JAVA code

All rules632

Vulnerability53

Bug154

Security Hotspot36

Code Smell389

Quick Fix42

Tags ▾

Search by name... 🔍

Abstract class names should comply with a naming convention

Code Smell

Strings literals should be placed on the left side when checking for equality

Code Smell

Files should contain an empty newline at the end

Code Smell

Source code should be indented consistently

Code Smell

A close curly brace should be located at the beginning of a line

Code Smell

Close curly brace and the next "else", "catch" and "finally" keywords should be on two different lines

Code Smell

Close curly brace and the next "else", "catch" and "finally" keywords should be located on the same line

Code Smell

An open curly brace should be located at the beginning of a line

Code Smell

An open curly brace should be located at the end of a line

Code Smell

Tabulation characters should not be used

Code Smell

Functions should not be defined with a variable number of arguments

Code Smell

HTTP response headers should not be vulnerable to injection attacks

Analyze your code

Vulnerability

Critical

injection

User-provided data, such as URL parameters, POST data payloads, or cookies, should always be considered untrusted and tainted. Applications constructing HTTP response headers based on tainted data could allow attackers to change security sensitive headers like Cross-Origin Resource Sharing headers.

Web application frameworks and servers might also allow attackers to inject new line characters in headers to craft malformed HTTP response. In this case the application would be vulnerable to a larger range of attacks like HTTP Response Splitting/Smuggling. Most of the time this type of attack is mitigated by default modern web application frameworks but there might be rare cases where older versions are still vulnerable.

As a best practice, applications that use user-provided data to construct the response header should always validate the data first. Validation should be based on a whitelist.

Noncompliant Code Example

```
protected void doGet(HttpServletRequest req, HttpServletResponse
    String value = req.getParameter("value");
    resp.addHeader("X-Header", value); // Noncompliant
}
```

Compliant Solution

```
protected void doGet(HttpServletRequest req, HttpServletResponse
    String value = req.getParameter("value");

    String whitelist = "safevalue1 safevalue2";
    if (!whitelist.contains(value))
        throw new IOException();

    resp.addHeader("X-Header", value); // Compliant
}
```

See

- [OWASP Top 10 2021 Category A3](#) - Injection
- [OWASP Attack Category](#) - HTTP Response Splitting
- [MITRE, CWE-20](#) - Improper Input Validation
- [MITRE, CWE-113](#) - Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting')
- [SANS Top 25](#) - Insecure Interaction Between Components


Deprecated

This rule is deprecated; use {rule:java:S5122}, {rule:java:S5146}, {rule:java:S6287} instead.


https://rules.sonarsource.com/java/RSPEC-5167

1/2


Local-Variable Type Inference should be used

 Code Smell


Migrate your tests from JUnit4 to the new JUnit5 annotations

 Code Smell

Track uses of disallowed classes


 Code Smell

Track uses of "@SuppressWarnings" annotations


 Code Smell

Available In:

sonarcloud



sonarqube



Developer Edition

© 2008-2022 SonarSource S.A., Switzerland. All content is copyright protected. SONAR, SONARSOURCE, SONARLINT, SONARQUBE and SONARCLOUD are trademarks of SonarSource S.A. All other trademarks and copyrights are the property of their respective owners. All rights are expressly reserved.
[Privacy Policy](#)