




 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 rules 632

Vulnerability 53

Bug 154

Security Hotspot 36

Code Smell 389

Quick Fix 42

Tags ▾

Search by name... 🔍

Exceptions should not be created without being thrown

Bug

Collection sizes and array length comparisons should make sense

Bug

Consumed Stream pipelines should not be reused

Bug

Intermediate Stream methods should not be left unused

Bug

All branches in a conditional structure should not have exactly the same implementation

Bug

Optional value should only be accessed after calling isPresent()

Bug

Overrides should match their parent class methods in synchronization

Bug

Value-based classes should not be used for locking

Bug

Expressions used in "assert" should not produce side effects

Bug

"volatile" variables should not be used with compound operators

Bug

"getClass" should not be used for synchronization

Bug

Disabling CSRF protections is security-sensitive

Analyze your code

Security Hotspot

Critical

cwe spring sans-top25 owasp

A cross-site request forgery (CSRF) attack occurs when a trusted user of a web application can be forced, by an attacker, to perform sensitive actions that he didn't intend, such as updating his profile or sending a message, more generally anything that can change the state of the application.

The attacker can trick the user/victim to click on a link, corresponding to the privileged action, or to visit a malicious web site that embeds a hidden web request and as web browsers automatically include cookies, the actions can be authenticated and sensitive.

Ask Yourself Whether

- The web application uses cookies to authenticate users.
- There exist sensitive operations in the web application that can be performed when the user is authenticated.
- The state / resources of the web application can be modified by doing HTTP POST or HTTP DELETE requests for example.

There is a risk if you answered yes to any of those questions.

Recommended Secure Coding Practices

- Protection against CSRF attacks is strongly recommended:
 - to be activated by default for all **unsafe HTTP methods**.
 - implemented, for example, with an unguessable CSRF token
- Of course all sensitive operations should not be performed with **safe HTTP** methods like GET which are designed to be used only for information retrieval.

Sensitive Code Example

Spring Security provides by default a protection against CSRF attacks which can be disabled:

```
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.csrf().disable(); // Sensitive: csrf protection is disabled
        // or
        http.csrf().ignoringAntMatchers("/route/"); // Sensitive
    }
}
```




Compliant Solution

Spring Security CSRF protection is enabled by default, do not disable it:

```
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
```

https://rules.sonarsource.com/java/RSPEC-4502

1/2

<p>Min and max used in combination should not always return the same value</p> <p> Bug</p>
<p>Assignment of lazy-initialized members should be the last step with double-checked locking</p> <p> Bug</p>
<p>"String" calls should not go beyond their bounds</p> <p> Bug</p>
<p>Raw byte values should not be used in bitwise operations in combination with shifts</p>

```
@Override
protected void configure(HttpSecurity http) throws Excepti
    // http.csrf().disable(); // Compliant
}
}
```

See

- [OWASP Top 10 2021 Category A1](#) - Broken Access Control
- [MITRE, CWE-352](#) - Cross-Site Request Forgery (CSRF)
- [OWASP Top 10 2017 Category A6](#) - Security Misconfiguration
- [OWASP: Cross-Site Request Forgery](#)
- [SANS Top 25](#) - Insecure Interaction Between Components

Available In:

