




 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... 🔍

Code Smell

Similar tests should be grouped in a single Parameterized test

Code Smell

Tests should be stable

Code Smell

Test methods should not contain too many assertions

Code Smell

AssertJ "assertThatThrownBy" should not be used alone

Code Smell

Character classes in regular expressions should not contain the same character twice

Code Smell

Names of regular expressions named groups should be used

Code Smell

Regexes containing characters subject to normalization should use the CANON_EQ flag

Code Smell

Regular expressions should not be too complicated

Code Smell

JUnit assertTrue/assertFalse should be simplified to the corresponding dedicated assertion

Code Smell

Only one method invocation is expected when testing runtime exceptions

Code Smell

Exception testing via JUnit ExpectedException rule should not be mixed with other assertions

Authorizations should be based on strong decisions

Analyze your code

VulnerabilityMajor?cwe owasp

Authorizations granted or not to users to access resources of an application should be based on strong decisions. For instance, checking whether the user is authenticated or not, has the right roles/privileges. It may also depend on the user's location, or the date, time when the user requests access.

Noncompliant Code Example

In a Spring-security web application:

- the vote method of an **AccessDecisionVoter** type is not compliant when it returns only an affirmative decision (ACCESS_GRANTED) or abstains to make a decision (ACCESS_ABSTAIN):

```
public class WeakNightVoter implements AccessDecisionVoter {
    @Override
    public int vote(Authentication authentication, Object ob

        Calendar calendar = Calendar.getInstance();

        int currentHour = calendar.get(Calendar.HOUR_OF_DAY);

        if(currentHour >= 8 && currentHour <= 19) {
            return ACCESS_GRANTED; // Noncompliant
        }

        // when users connect during the night, do not make de
        return ACCESS_ABSTAIN; // Noncompliant
    }
}
```

- the hasPermission method of a **PermissionEvaluator** type is not compliant when it doesn't return false:

```
public class MyPermissionEvaluator implements PermissionEval
    @Override
    public boolean hasPermission(Authentication authenticati
        //Getting subject
        Object user = authentication.getPrincipal();

        if(user.getRole().equals(permission)) {
            return true; // Noncompliant
        }





        return true; // Noncompliant
    }
}
```

Compliant Solution

In a Spring-security web application:

https://rules.sonarsource.com/java/RSPEC-5808

1/2

 Code Smell
"@Deprecated" code marked for removal should never be used
 Code Smell
Vararg method arguments should not be confusing
 Code Smell
Whitespace for text block indent should be consistent
 Code Smell
'List.remove()' should not be used in ascending 'for' loops

- the vote method of an [AccessDecisionVoter](#) type should return a negative decision (ACCESS_DENIED):

```
public class StrongNightVoter implements AccessDecisionVoter
{
    @Override
    public int vote(Authentication authentication, Object object,
        Collection<GrantedAuthority> authorities) {

        Calendar calendar = Calendar.getInstance();

        int currentHour = calendar.get(Calendar.HOUR_OF_DAY);

        if(currentHour >= 8 && currentHour <= 19) {
            return ACCESS_GRANTED;
        }

        // users are not allowed to connect during the night
        return ACCESS_DENIED; // Compliant
    }
}
```

- the hasPermission method of a [PermissionEvaluator](#) type should return false:

```
public class MyPermissionEvaluator implements PermissionEvaluator
{
    @Override
    public boolean hasPermission(Authentication authentication, Object target, Object principal) {
        //Getting subject
        Object user = authentication.getPrincipal();

        if(user.getRole().equals(permission)) {
            return true;
        }

        return false; // Compliant
    }
}
```

Exceptions

No issue is reported when the method throws an exception as it might be used to indicate a strong decision.

See

- [OWASP Top 10 2021 Category A1](#) - Broken Access Control
- [OWASP Top 10 2017 Category A5](#) - Broken Access Control
- [MITRE, CWE-285](#) - Improper Authorization

Available In:

sonarlint  | **sonarcloud**  | **sonarqube** 