




 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... 🔍

Accessing Android external storage is security-sensitive

Security Hotspot

Receiving intents is security-sensitive

Security Hotspot

Broadcasting intents is security-sensitive

Security Hotspot

Expanding archive files without controlling resource consumption is security-sensitive

Security Hotspot

Configuring loggers is security-sensitive

Security Hotspot

Using weak hashing algorithms is security-sensitive

Security Hotspot

Using unsafe Jackson deserialization configuration is security-sensitive

Security Hotspot

Setting JavaBean properties is security-sensitive

Security Hotspot

Disabling CSRF protections is security-sensitive

Security Hotspot

Using non-standard cryptographic algorithms is security-sensitive

Security Hotspot

Using pseudorandom number generators (PRNGs) is security-sensitive

Security Hotspot

Mocking all non-private methods of a class should be avoided

Security Hotspot

Passwords should not be stored in plain-text or with a fast hashing algorithm

Analyze your code

VulnerabilityCritical🔍cwe spring owasp sans-top25

A user password should never be stored in clear-text, instead a hash should be produced from it using a secure algorithm:

- not vulnerable to brute force attacks.
- not vulnerable to collision attacks (see rule s4790).
- and a salt should be added to the password to lower the risk of rainbow table attacks (see rule s2053).

This rule raises an issue when a password is stored in clear-text or with a hash algorithm vulnerable to brute force attacks. These algorithms, like md5 or SHA-family functions are fast to compute the hash and therefore brute force attacks are possible (it's easier to exhaust the entire space of all possible passwords) especially with hardware like GPU, FPGA or ASIC. Modern password hashing algorithms such as bcrypt, PBKDF2 or argon2 are recommended.

Noncompliant Code Example

```
@Autowired
public void configureGlobal(AuthenticationManagerBuilder auth) {
    auth.jdbcAuthentication()
        .dataSource(dataSource)
        .usersByUsernameQuery("SELECT * FROM users WHERE username=? AND password=?")
        .passwordEncoder(new StandardPasswordEncoder()); // Noncompliant

    // OR
    auth.jdbcAuthentication()
        .dataSource(dataSource)
        .usersByUsernameQuery("SELECT * FROM users WHERE username=? AND password=?")
        .passwordEncoder(new StandardPasswordEncoder()); // Noncompliant

    // OR
    auth.userDetailsService(userDetailsService); // Noncompliant; default use
    // OR
    auth.userDetailsService(userDetailsService).passwordEncoder(new StandardPasswordEncoder());
}
```

Compliant Solution


```
@Autowired
public void configureGlobal(AuthenticationManagerBuilder auth) {
    auth.jdbcAuthentication()
        .dataSource(dataSource)
        .usersByUsernameQuery("Select * from users where username=? and password=?")
        .passwordEncoder(new BCryptPasswordEncoder());

    // or
    auth.userDetailsService(userDetailsService).passwordEncoder(new BCryptPasswordEncoder());
}
```


https://rules.sonarsource.com/java/RSPEC-5344

1/2


class should be avoided

 Code Smell


Empty lines should not be tested with regex MULTILINE flag

 Code Smell

Methods setUp() and tearDown() should be correctly annotated starting with JUnit4

 Code Smell




Class members annotated with "@VisibleForTesting" should not be accessed from production code

 Code Smell

See

- [OWASP Top 10 2021 Category A2](#) - Cryptographic Failures
- [OWASP Top 10 2021 Category A4](#) - Insecure Design
- [OWASP CheatSheet](#) - Password Storage Cheat Sheet
- [OWASP Top 10 2017 Category A3](#) - Sensitive Data Exposure
- [MITRE, CWE-256](#) - Plaintext Storage of a Password
- [MITRE, CWE-916](#) - Use of Password Hash With Insufficient Computational Effort
- [SANS Top 25](#) - Porous Defenses

Available In:

 |  | 

© 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](#)