Java static code analysis

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

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

Analyze your code

Vulnerability Critical 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 `bruce 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 aut
  auth.jdbcAuthentication()
    .dataSource(dataSource)
    .usersByUsernameQuery("SELECT * FROM users WHERE usernam
    .passwordEncoder(new StandardPasswordEncoder()); // Nonc

  // OR
  auth.jdbcAuthentication()
    .dataSource(dataSource)
    .usersByUsernameQuery("SELECT * FROM users WHERE usernam

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

Compliant Solution

```
@Autowired
public void configureGlobal(AuthenticationManagerBuilder aut
  auth.jdbcAuthentication()
    .dataSource(dataSource)
    .usersByUsernameQuery("Select * from users where usernam
    .passwordEncoder(new BCryptPasswordEncoder());

  // or
  auth.userDetailsService(null).passwordEncoder(new BCryptPa
}
```

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:

sonarlint | sonarcloud | sonarqube