




 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... 🔍

String multiline concatenation should be replaced with Text Blocks

Code Smell

Single-character alternations in regular expressions should be replaced with character classes

Code Smell

Reluctant quantifiers in regular expressions should be followed by an expression that can't match the empty string

Code Smell

Constructors of an "abstract" class should not be declared "public"

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

Mobile database encryption keys should not be disclosed

Analyze your code

VulnerabilityMajor🔍cwe owasp android

Storing data locally is a common task for mobile applications. There are many convenient solutions that allow storing data persistently, for example SQLiteDatabase and Realm. These systems can be initialized with a secret key in order to store the data encrypted.

The encryption key is meant to stay secret and should not be hard-coded in the application as it would mean that:

- All user would use the same encryption key.
- The encryption key would be known by anyone who as access to the source code or the application binary code.
- Data stored encrypted in the database would not be protected.

There are different approaches how the key can be provided to encrypt and decrypt the database. One of the most convinient way to is to rely on EncryptedSharedPreferences to store encryption keys. It can also be provided dynamically by the user of the application or fetched from a remote server.

Noncompliant Code Example

SQLCipher

```
String key = "gb09ym9ydoolp3w886d0tciczj6ve9kszqd65u7d126040"
SQLiteDatabase db = SQLiteDatabase.openOrCreateDatabase("tes
```

Realm

```
String key = "gb09ym9ydoolp3w886d0tciczj6ve9kszqd65u7d126040"
RealmConfiguration config = new RealmConfiguration.Builder()
    .encryptionKey(key.toByteArray()) // Noncompliant
    .build();
Realm realm = Realm.getInstance(config);
```

Compliant Solution

SQLCipher

```
SQLiteDatabase db = SQLiteDatabase.openOrCreateDatabase("tes
```

Realm


```
RealmConfiguration config = new RealmConfiguration.Builder()
    .encryptionKey(getKey())
    .build();
Realm realm = Realm.getInstance(config);
```

See


https://rules.sonarsource.com/java/RSPEC-6301

1/2


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

- [OWASP Top 10 2021 Category A2](#) - Cryptographic Failures
- [OWASP Top 10 2021 Category A4](#) - Insecure Design
- [Mobile AppSec Verification Standard](#) - Data Storage and Privacy Requirements
- [OWASP Mobile Top 10 2016 Category M2](#) - Insecure Data Storage
- [OWASP Top 10 2017 Category A3](#) - Sensitive Data Exposure
- [OWASP Top 10 2017 Category A6](#) - Security Misconfiguration
- [MITRE, CWE-311](#) - Missing Encryption of Sensitive Data
- [MITRE, CWE-321](#) - Use of Hard-coded Cryptographic Key

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](#)