

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

632

All rules

53

Vulnerability

154

Bug

36

Security Hotspot

389

Code Smell

42

Quick Fix

Code Smell

Boolean checks should not be inverted

Code Smell

Redundant casts should not be used

Code Smell

"@Deprecated" code should not be used

Code Smell

"toString()" should never be called on a String object

Code Smell

Annotation repetitions should not be wrapped

Code Smell

Multiple variables should not be declared on the same line

Code Smell

Strings should not be concatenated using '+' in a loop

Code Smell

Maps with keys that are enum values should be replaced with EnumMap

Code Smell

Lambdas should be replaced with method references

Code Smell

Parentheses should be removed from a single lambda input parameter when its type is inferred

Code Smell

Abstract classes without fields should be converted to interfaces

Code Smell

Using unencrypted databases in mobile applications is security-sensitive

Analyze your code

Security Hotspot

Major

cwe owasp android

Storing data locally is a common task for mobile applications. Such data includes preferences or authentication tokens for external services, among other things. There are many convenient solutions that allow storing data persistently, for example SQLiteDatabase, SharedPreferences, and Realm. By default these systems store the data unencrypted, thus an attacker with physical access to the device can read them out easily. Access to sensitive data can be harmful for the user of the application, for example when the device gets stolen.

Ask Yourself Whether

The database contains sensitive data that could cause harm when leaked.

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

Recommended Secure Coding Practices

It's recommended to password-encrypt local databases that contain sensitive information. Most systems provide secure alternatives to plain-text storage that should be used. If no secure alternative is available the data can also be encrypted manually before it is stored.

The encryption password should not be hard-coded in the application. There are different approaches how the password can be provided to encrypt and decrypt the database. In the case of EncryptedSharedPreferences the Android Keystore can be used to store the password. Other databases can rely on EncryptedSharedPreferences to store passwords. The password can also be provided dynamically by the user of the application or it can be fetched from a remote server if the other methods are not feasible.

Sensitive Code Example

For SQLiteDatabase:

SQLiteDatabase db = activity.openOrCreateDatabase("test.db",

For SharedPreferences:

SharedPreferences pref = activity.getSharedPreferences(Context.MODE

For Realm:

RealmConfiguration config = new RealmConfiguration.Builder()
Realm realm = Realm.getInstance(config); // Sensitive

Compliant Solution

Instead of SQLiteDatabase you can use SQLCipher:


https://rules.sonarsource.com/java/RSPEC-6291

1/2

Lambdas containing only one statement should not nest this statement in a block

 Code Smell

"Collections.EMPTY_LIST", "EMPTY_MAP", and "EMPTY_SET" should not be used

 Code Smell

Local variables should not be declared and then immediately returned or thrown

 Code Smell

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

Instead of SharedPreferences you can use [EncryptedSharedPreferences](#):

```
String masterKeyAlias = new MasterKeys.getOrCreate(MasterKey
EncryptedSharedPreferences.create(
    "secret",
    masterKeyAlias,
    context,
    EncryptedSharedPreferences.PrefKeyEncryptionScheme.AES25
EncryptedSharedPreferences.PrefValueEncryptionScheme.AES
);
```

For Realm an encryption key can be specified in the config:

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

See

- [OWASP Top 10 2021 Category A2](#) - Cryptographic Failures
- [OWASP Top 10 2021 Category A4](#) - Insecure Design
- [OWASP Top 10 2021 Category A5](#) - Security Misconfiguration
- [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

Available In:

sonarcloud  | **sonarqube** 

© 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](#)