Secrets
ABAP
Apex
C
C++
CloudFormation
COBOL
C#
CSS
Flex
Go
HTML
Java
JavaScript
Kotlin
Kubernetes
Objective C
PHP
PL/I
PL/SQL
Python
RPG
Ruby
Scala
Swift
Terraform
Text
TypeScript
T-SQL
VB.NET
VB6
XML

Kotlin static code analysis

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

All rules 98 | Vulnerability 10 | Bug 17 | Security Hotspot 15 | Code Smell 56

Tags

Search by name...

Hard-coded credentials are security-sensitive
Security Hotspot

Cipher algorithms should be robust
Vulnerability

Encryption algorithms should be used with secure mode and padding scheme
Vulnerability

Server hostnames should be verified during SSL/TLS connections
Vulnerability

Server certificates should be verified during SSL/TLS connections
Vulnerability

Cryptographic keys should be robust
Vulnerability

Weak SSL/TLS protocols should not be used
Vulnerability

"SecureRandom" seeds should not be predictable
Vulnerability

Cipher Block Chaining IVs should be unpredictable
Vulnerability

Hashes should include an unpredictable salt
Vulnerability

Regular expressions should be syntactically valid
Bug

"runFinalizersOnExit" should not be called
Bug

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:

```
var db = activity.openOrCreateDatabase("test.db", Contex
```

For SharedPreferences:

```
val pref = activity.getPreferences(Context.MODE_PRIVATE)
```

For Realm:

```
val config = RealmConfiguration.Builder().build()
val realm = Realm.getInstance(config) // Sensitive
```

Compliant Solution

Instead of SQLiteDatabase you can use SQLCipher:

```
val db = SQLiteDatabase.openOrCreateDatabase("test.db",
```

Instead of SharedPreferences you can use EncryptedSharedPreferences:

"ScheduledThreadPoolExecutor" should not have 0 core threads

Bug

Jump statements should not occur in "finally" blocks

Bug

Using clear-text protocols is security-sensitive

Security Hotspot

Accessing Android external storage is security-sensitive

Security Hotspot

Receiving intents is security-sensitive

Security Hotspot

Broadcasting intents is security-sensitive

Security Hotspot

Using weak hashing algorithms is security-sensitive

Security Hotspot

Using pseudorandom number generators (PRNGs) is security-sensitive

Security Hotspot

Empty lines should not be tested with regex MULTILINE flag

Code Smell

Cognitive Complexity of functions should not be too high

Code Smell

```
val masterKeyAlias = MasterKeys.getOrCreate(MasterKeys.A
EncryptedSharedPreferences.create(
    "secret",
    masterKeyAlias,
    context,
    EncryptedSharedPreferences.PrefKeyEncryptionScheme.A
    EncryptedSharedPreferences.PrefValueEncryptionScheme
)
```

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

```
val config = RealmConfiguration.Builder()
    .encryptionKey(getKey())
    .build()
val 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