sonar RULES

Products

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

Suspending functions should be main-safe

Analyze your code

Code Smell | Major | coroutines performance bad-practice pitfall

Generally speaking, main threads should be available to allow user-facing parts of an application to remain responsive. Long-running blocking operations can significantly reduce threads' availability and are best executed on a designated thread pool.

As a consequence, suspending functions should not block main threads and instead move any long-running blocking tasks off the main thread. This can be done conveniently by using `withContext` with an appropriate dispatcher. Alternatively, coroutine builders such as `launch` and `async` accept an optional `CoroutineContext`. An appropriate dispatcher could be `Dispatchers.IO` for long-running blocking IO operations, which can create and shutdown threads on demand.

For some blocking tasks and APIs there may already be suspending alternatives available. When available, these alternatives should be used instead of their blocking counterparts.

This rule raises an issue when the call of a long-running blocking function is detected within a suspending function without the use of an appropriate dispatcher. If non-blocking alternatives to the called function are known, they may be suggested (e.g. use `delay(…)` instead of `Thread.sleep(…)`).

Noncompliant Code Example

Executing long-running blocking IO operations on the main thread pool:

```
class workerClass {
    suspend fun worker(): String {
        val client = HttpClient.newHttpClient()
        val request = HttpRequest.newBuilder(URI("https:/
        return coroutineScope {
            client.send(request, HttpResponse.BodyHandle
        }
    }
}
```

Using inappropriate blocking APIs:

```
suspend fun example() {
    ...
    Thread.sleep(1000) // Noncompliant
    ...
}
```

Compliant Solution

Executing long-running blocking IO operations in an appropriate thread pool using `Dispatcher.IO`:

```
class workerClass(
    private val ioDispatcher: CoroutineDispatcher = Dispa
) {
    suspend fun worker(): String {
        val client = HttpClient.newHttpClient()
        val request = HttpRequest.newBuilder(URI("https:/
```

"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

```
            return withContext(ioDispatcher) {
                client.send(request, HttpResponse.BodyHandle
            }
        }
    }
```

Using appropriate non-blocking APIs:

```
suspend fun example() {
    ...
    delay(1000) // Compliant
    ...
}
```

See

- Coroutine context and dispatchers
- Suspend functions should be safe to call from the main thread (Android coroutines best practices)
- IO CoroutineDispatcher
- Default CoroutineDispatcher

Available In:

sonarlint sonarcloud sonarqube