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

Dispatchers should be injectable

Analyze your code

Code Smell Major coroutines design

Dispatchers should not be hardcoded when using `withContext` or creating new coroutines using `launch` or `async`. Injectable dispatchers ease testing by allowing tests to inject more deterministic dispatchers.

You can use default values for the dispatcher constructor arguments to eliminate the need to specify them explicitly in the production caller contexts.

This rule raises an issue when it finds a hard-coded dispatcher being used in `withContext` or when starting new coroutines.

Noncompliant Code Example

```
class ExampleClass {
    suspend fun doSomething() {
        withContext(Dispatchers.Default) { // Noncomplia
            ...
        }
    }
}
```

Compliant Solution

```
class ExampleClass(
    private val dispatcher: CoroutineDispatcher = Dispat
) {
    suspend fun doSomething() {
        withContext(dispatcher) {
            ...
        }
    }
}
```

See

- Inject dispatchers (Android coroutines best practices)

Available In:
sonarlint sonarcloud sonarqube

"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