Search by name...





ABAP

Apex Apex

C C

C++

CloudFormation

COBOL

C# C#

CSS

X Flex

-co Go

HTML

Java

Js JavaScript

Kotlin

Kubernetes

🐞 Objective C

PHP PHP

PL/I

PL/SQL

Python

RPG RPG

Ruby

Scala

Swift

Terraform

Text

Ts TypeScript

T-SQL

VB VB.NET

VB6 VB6

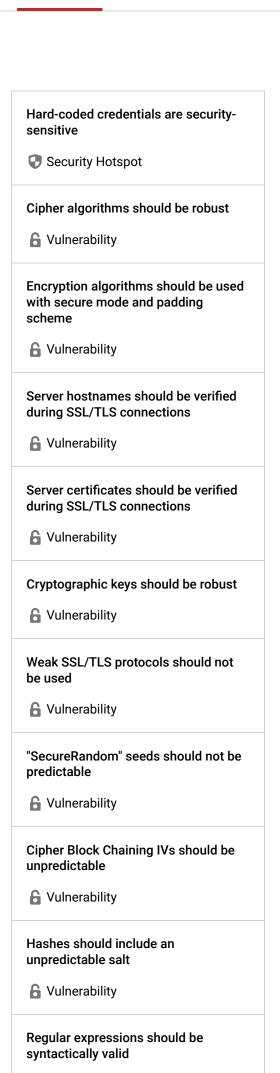
XML XML



Kotlin static code analysis

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

Tags



Rug Bug

🛊 Bug

"runFinalizersOnExit" should not be

```
Kotlin coroutines API for
                                          Analyze your code
timeouts should be used
coroutines
Sometimes there is the need to cancel the execution of a coroutine after a
given period of time. You can do this manually by combining the delay() and
cancel () functions. However, this technique is verbose and error-prone. An
easier way to manage timeouts is using the function withTimeout() or
withTimeoutOrNull().
The withTimeout function will throw a
TimeoutCancellationException when the timeout is reached, while
withTimeoutOrNull will simply return null instead.
This rule raises an issue if timeout mechanisms are implemented manually
instead of using appropriate built-in functions.
Noncompliant Code Example
 suspend fun main() {
      coroutineScope {
           val job = launch {
               delay(2000L)
               println("Finished")
           delay(500L)
           job.cancel()
      }
  }
Compliant Solution
 suspend fun main() {
      coroutineScope {
           withTimeoutOrNull(1000L){
               delay(2000L)
               println("Finished")
           }
 }
See

    Cancellation and timeouts

 Available In:
 sonarlint ⊕ | 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

"ScheduledThreadPoolExecutor" should not have 0 core threads
Jump statements should not occur in "finally" blocks
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