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

Empty lines should not be tested with regex MULTILINE flag

Analyze your code

Code Smell | Critical | regex

One way to test for empty lines is to use the regex `"^$"`, which can be extremely handy when filtering out empty lines from collections of Strings, for instance. With regard to this, the Javadoc for Pattern (Line Terminators) states the following:

> By default, the regular expressions ^ and $ ignore line terminators and only match at the beginning and the end, respectively, of the entire input sequence. If `MULTILINE` mode is activated then ^ matches at the beginning of input and after any line terminator **except at the end of input**. When in `MULTILINE` mode $ matches just before a line terminator or the end of the input sequence.

As emphasized, ^ is not going to match at the end of an input, and the end of the input is necessarily included in the empty string, which might lead to completely missing empty lines, while it would be the initial reason for using such regex.

Therefore, when searching for empty lines using a multi-line regular expression, you should also check whether the string is empty.

This rule is raising an issue every time a pattern that can match the empty string is used with `MULTILINE` flag and without calling `isEmpty()` on the string.

Noncompliant Code Example

```
val p = Pattern.compile("^$", Pattern.MULTILINE) // Nonc
val r = Regex("^$", RegexOption.MULTILINE) // Noncomplia

// Alternatively
val p = Pattern.compile("(?m)^$") // Noncompliant
val r = Regex("(?m)^$") // Noncompliant

fun containsEmptyLines(str: String) : Boolean {
    return p.matcher(str).find()
}

fun containsEmptyLinesKotlin(str: String) = r.find(str)

// ...
println(containsEmptyLines("a\n\nb")) // correctly print
println(containsEmptyLinesKotlin("a\n\nb")) // correctly

println(containsEmptyLines("")) // incorrectly prints 'f
println(containsEmptyLinesKotlin("")) // incorrectly pri
```

Compliant Solution

```
val p = Pattern.compile("^$", Pattern.MULTILINE) // Nonc
val r = Regex("^$", RegexOption.MULTILINE) // Noncomplia

fun containsEmptyLines(str: String) : Boolean {
    return p.matcher(str).find() || str.isEmpty()
}
```

```
fun containsEmptyLinesKotlin(str: String) = r.find(str)

// ...
println(containsEmptyLines("a\n\nb")) // correctly print
println(containsEmptyLinesKotlin("a\n\nb")) // correctly

println(containsEmptyLines("")) // correctly prints 'tru
println(containsEmptyLinesKotlin("")) // correctly print
```

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