- 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

"ScheduledThreadPoolExecutor" should not have 0 core threads
Bug

Unicode Grapheme Clusters should be avoided inside regex character classes

Analyze your code

Bug | Major | regex

When placing Unicode Grapheme Clusters (characters which require to be encoded in multiple Code Points) inside a character class of a regular expression, this will likely lead to unintended behavior.

For instance, the grapheme cluster c̈ requires two code points: one for `'c'`, followed by one for the *umlaut* modifier `'\u{0308}'`. If placed within a character class, such as `[c̈]`, the regex will consider the character class being the enumeration `[c\u{0308}]` instead. It will, therefore, match every `'c'` and every *umlaut* that isn’t expressed as a single codepoint, which is extremely unlikely to be the intended behavior.

This rule raises an issue every time Unicode Grapheme Clusters are used within a character class of a regular expression.

Noncompliant Code Example

```
"c̈d̈d".replace(Regex("[c̈d̈]"), "X") // Noncompliant, print
```

Compliant Solution

```
"c̈d̈d".replace(Regex("c̈|d̈"), "X") // print "cXXd"
```

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

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

String literals should not be duplicated

Code Smell