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

Character classes should be preferred over reluctant quantifiers in regular expressions

Analyze your code

Code Smell | Minor | regex

Using reluctant quantifiers (also known as lazy or non-greedy quantifiers) in patterns can often lead to needless backtracking, making the regex needlessly inefficient and potentially vulnerable to catastrophic backtracking. Particularly when using `.*?` or `.+?` to match anything up to some terminating character, it is usually a better idea to instead use a greedily or possessively quantified negated character class containing the terminating character. For example `<.+?>` should be replaced with `<[^>]++>`.

Noncompliant Code Example

```
<.+?>
".*?"
```

Compliant Solution

```
<[^>]++>
"[^"]*+"
```

or

```
<[^>]+>
"[^"]*"
```

Exceptions

This rule only applies in cases where the reluctant quantifier can easily be replaced with a negated character class. That means the repetition has to be terminated by a single character or character class. Patterns such as the following, where the alternatives without reluctant quantifiers are more complicated, are therefore not subject to this rule:

```
<!--.*?-->
/\*.*?\*/
```

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