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

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

Two branches in a conditional structure should not have exactly the same implementation

Analyze your code

Code Smell Major design suspicious

Having two clauses in a `when` statement or two branches in an `if` chain with the same implementation is at best duplicate code, and at worst a coding error. If the same logic is truly needed for both instances, then in an `if` chain they should be combined, or for a `when`, duplicates should be refactored.

Noncompliant Code Example

```
fun s1871(x: Int) {
    when (x) {
        1 -> {
            val y = x / 2
            print(y)
        }
        2 -> {
            val y = x / 2
            print(y)
        }
    }
}
```

Exceptions

Blocks in an `if` chain that contain a single line of code are ignored, as are blocks in a `when` statement that contain a single line of code with or without a following `break`.

```
if (a == 1) {
    doSomething()  //no issue, usually this is done on p
} else if (a == 2) {
    doSomethingElse()
} else {
    doSomething()
}
```

But this exception does not apply to `if` chains without `else`-s, or to `when`-es without `else` clauses when all branches have the same single line of code. In case of `if` chains with `else`-s, or of `when`-es with default clauses, rule {rule:kotlin:S3923} raises a bug.

```
if (a == 1) {
  doSomething()  //Noncompliant, this might have been do
} else if (a == 2) {
  doSomething()
}
```

Available In:

sonarlint sonarcloud sonarqube