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

"MutableStateFlow" and "MutableSharedFlow" should not be exposed

Analyze your code

Code Smell | Major | coroutines bad-practice

`MutableStateFlow` and `MutableSharedFlow` are very convenient for storing and adding updates of some data structures in event-driven paradigm. This is widely used in Android Views for handling updates. While it's extremely useful to manage such objects inside some class, it's not recommended to expose them outside of the class.

When properties of the types `MutableStateFlow` or `MutableSharedFlow` are accessible from outside of a class, data updates cannot be verified properly anymore. It is generally recommended to have only one class responsible for updating these flows, otherwise inconsistency issues and problems with maintainability, as well as increased error-proneness may be introduced.

To restrict write access, `StateFlow` or `SharedFlow` should be used together with private `MutableStateFlow` or `MutableSharedFlow` fields.

This rule raises an issue when encountering a public or internal property of the type `MutableStateFlow` or `MutableSharedFlow`.

Noncompliant Code Example

```
class MyView : ViewModel() {

    val state = MutableStateFlow(State.New)

}
```

Compliant Solution

```
class MyView : ViewModel() {

    private val _state = MutableStateFlow(State.New)
    val state: StateFlow<LatestNewsUiState> = _uiState

}
```

See

- Android Coroutines Best Practices

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