Kotlin static code analysis

Unique rules to find Bugs, Vulnerabilities, Security Hotspots, and Code Smells in your KOTLIN code

1. Hard-coded credentials are security-sensitive
 Security Hotspot
2. Cipher algorithms should be robust
 Vulnerability
3. Encryption algorithms should be used with secure mode and padding scheme
 Vulnerability
4. Server hostnames should be verified during SSL/TLS connections
 Vulnerability
5. Server certificates should be verified during SSL/TLS connections
 Vulnerability
6. Cryptographic keys should be robust
 Vulnerability
7. Weak SSL/TLS protocols should not be used
 Vulnerability
8. "SecureRandom" seeds should not be predictable
 Vulnerability
9. Cipher Block Chaining IVs should be unpredictable
 Vulnerability
10. Hashes should include an unpredictable salt
 Vulnerability
11. Regular expressions should be syntactically valid
 Bug
12. "runFinalizersOnExit" should not be called
 Bug
13. "ScheduledThreadPoolExecutor" should not have 0 core threads
 Bug
14. Jump statements should not occur in "finally" blocks
 Bug
15. Using clear-text protocols is security-sensitive
 Security Hotspot
16.

Accessing Android external storage is security-sensitive
Security Hotspot

17.

Receiving intents is security-sensitive
Security Hotspot

18.

Broadcasting intents is security-sensitive
Security Hotspot

19.

Using weak hashing algorithms is security-sensitive
Security Hotspot

20.

Using pseudorandom number generators (PRNGs) is security-sensitive
Security Hotspot

21.

Empty lines should not be tested with regex MULTILINE flag
Code Smell

22.

Cognitive Complexity of functions should not be too high
Code Smell

23.

String literals should not be duplicated
Code Smell

24.

Functions should not be empty
Code Smell

25.

Mobile database encryption keys should not be disclosed
Vulnerability

26.

Flow intermediate operation results should not be left unused
Bug

27.

Equals method should be overridden in data classes containing array fields
Bug

28.

Unicode Grapheme Clusters should be avoided inside regex character classes
Bug

29.

Alternatives in regular expressions should be grouped when used with anchors
Bug

30.

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

31.

"=+" should not be used instead of "+="
Bug

32.

Values should not be uselessly incremented
Bug

33.

Related "if/else if" statements should not have the same condition
Bug

34.

Identical expressions should not be used on both sides of a binary operator
Bug

35.

All code should be reachable
Bug

36.

Variables should not be self-assigned
Bug

37.

Useless "if(true) {...}" and "if(false){...}" blocks should be removed
Bug

38.

Enabling file access for WebViews is security-sensitive
Security Hotspot

39.

Enabling JavaScript support for WebViews is security-sensitive
Security Hotspot

40.

Using unencrypted files in mobile applications is security-sensitive
Security Hotspot

41.

Using biometric authentication without a cryptographic solution is security-sensitive
Security Hotspot

42.

Using unencrypted databases in mobile applications is security-sensitive
Security Hotspot

43.

Authorizing non-authenticated users to use keys in the Android KeyStore is security-sensitive
Security Hotspot

44.

Kotlin coroutines API for timeouts should be used
Code Smell

45.

The return value of functions returning "Deferred" should be used
Code Smell

46.

ViewModel classes should create coroutines
Code Smell

47.

Extension functions on CoroutineScopes should not be declared as "suspend"
Code Smell

48.

Suspending functions should not be called on a different dispatcher
Code Smell

49.

Dispatchers should be injectable
Code Smell

50.

Functions returning Flow/Channel should not be suspending
Code Smell

51.

Suspending functions should be main-safe
Code Smell

52.

Coroutine usage should adhere to structured concurrency principles
Code Smell

53.

"MutableStateFlow" and "MutableSharedFlow" should not be exposed
Code Smell

54.

Operator "is" should be used instead of "isInstance()"
Code Smell

55.

Character classes in regular expressions should not contain the same character twice
Code Smell

56.

Regular expressions should not be too complicated
Code Smell

57.

Native features should be preferred to Guava
Code Smell

58.

Functions should not have identical implementations
Code Smell

59.

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

60.

"when" statements should not have too many clauses
Code Smell

61.

Sections of code should not be commented out
Code Smell

62.

Unused function parameters should be removed
Code Smell

63.

Unused "private" methods should be removed
Code Smell

64.

Track uses of "FIXME" tags
Code Smell

65.

Redundant pairs of parentheses should be removed
Code Smell

66.

Nested blocks of code should not be left empty

Code Smell

67.

Functions should not have too many parameters
Code Smell

68.

Collapsible "if" statements should be merged
Code Smell

69.

Repeated patterns in regular expressions should not match the empty string
Bug

70.

Delivering code in production with debug features activated is security-sensitive
Security Hotspot

71.

Using hardcoded IP addresses is security-sensitive
Security Hotspot

72.

"suspend" modifier should not be redundant
Code Smell

73.

Character classes should be preferred over reluctant quantifiers in regular expressions
Code Smell

74.

Multi-line comments should not be empty
Code Smell

75.

Boolean checks should not be inverted
Code Smell

76.

Code annotated as deprecated should not be used
Code Smell

77.

Unused local variables should be removed
Code Smell

78.

Local variable and function parameter names should comply with a naming convention
Code Smell

79.

Unnecessary imports should be removed
Code Smell

80.

Boolean literals should not be redundant
Code Smell

81.

Class names should comply with a naming convention
Code Smell

82.

Method names should comply with a naming convention
Code Smell

83.

Track uses of "TODO" tags

		Code Smell
84.	Deprecated code should be removed	Code Smell
85.	Track lack of copyright and license headers	Code Smell
86.	"when" statements should not be nested	Code Smell
87.	Control flow statements "if", "for", "while", "when" and "try" should not be nested too deeply	Code Smell
88.	"if ... else if" constructs should end with "else" clauses	Code Smell
89.	Expressions should not be too complex	Code Smell
90.	Lambdas should not have too many lines	Code Smell
91.	Kotlin parser failure	Code Smell
92.	Functions should not have too many lines of code	Code Smell
93.	Statements should be on separate lines	Code Smell
94.	"when" clauses should not have too many lines of code	Code Smell
95.	Files should not have too many lines of code	Code Smell
96.	Lines should not be too long	Code Smell
97.	Unicode-aware versions of character classes should be preferred	Code Smell
98.	Tabulation characters should not be used	Code Smell