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

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

Cipher Block Chaining IVs should be unpredictable

Analyze your code

Vulnerability Critical cwe owasp

When encrypting data with the Cipher Block Chaining (CBC) mode an Initialization Vector (IV) is used to randomize the encryption, ie under a given key the same plaintext doesn't always produce the same ciphertext. The IV doesn't need to be secret but should be unpredictable to avoid "Chosen-Plaintext Attack".

To generate Initialization Vectors, NIST recommends to use a secure random number generator.

Noncompliant Code Example

```
val bytesIV = "7cVgr5cbdCZVw5WY".toByteArray(charset("UT

val iv = IvParameterSpec(bytesIV)
val skeySpec = SecretKeySpec(secretKey.toByteArray(), "A

val cipher: Cipher = Cipher.getInstance("AES/CBC/PKCS5PA
cipher.init(Cipher.ENCRYPT_MODE, skeySpec, iv) // Noncom

val encryptedBytes: ByteArray = cipher.doFinal("foo".toB
```

Compliant Solution

```
val random: SecureRandom = SecureRandom()

val bytesIV: ByteArray = ByteArray(16)
random.nextBytes(bytesIV); // Unpredictable / random IV

val iv = IvParameterSpec(bytesIV)
val skeySpec = SecretKeySpec(secretKey.toByteArray(), "A

val cipher: Cipher = Cipher.getInstance("AES/CBC/PKCS5PA
cipher.init(Cipher.ENCRYPT_MODE, skeySpec, iv) //Complia

val encryptedBytes: ByteArray = cipher.doFinal("foo".toB
```

See

- OWASP Top 10 2021 Category A2 - Cryptographic Failures
- OWASP Top 10 2017 Category A6 - Security Misconfiguration
- Mobile AppSec Verification Standard - Cryptography Requirements
- OWASP Mobile Top 10 2016 Category M5 - Insufficient Cryptography
- MITRE, CWE-329 - Not Using an Unpredictable IV with CBC Mode
- MITRE, CWE-330 - Use of Insufficiently Random Values
- MITRE, CWE-340 - Generation of Predictable Numbers or Identifiers
- MITRE, CWE-1204 - Generation of Weak Initialization Vector (IV)
- NIST, SP-800-38A - Recommendation for Block Cipher Modes of Operation

Available In:

sonarlint | sonarcloud | sonarqube