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

Hashes should include an unpredictable salt

Analyze your code

Vulnerability | Critical | cwe sans-top25 owasp

In cryptography, a "salt" is an extra piece of data which is included when hashing a password. This makes `rainbow-table attacks` more difficult. Using a cryptographic hash function without an unpredictable salt increases the likelihood that an attacker could successfully find the hash value in databases of precomputed hashes (called `rainbow-tables`).

This rule raises an issue when a hashing function which has been specifically designed for hashing passwords, such as `PBKDF2`, is used with a non-random, reused or too short salt value. It does not raise an issue on base hashing algorithms such as `sha1` or `md5` as they should not be used to hash passwords.

Recommended Secure Coding Practices

- Use hashing functions generating their own secure salt or generate a secure random value of at least 16 bytes.
- The salt should be unique by user password.

Noncompliant Code Example

```
val salt = "notrandom".toByteArray()

val cipherSpec = PBEParameterSpec(salt, 10000) // Noncom
val spec = PBEKeySpec(password, salt, 10000, 256) // Non
```

Compliant Solution

```
val random = SecureRandom()
val salt = ByteArray(16)
random.nextBytes(salt)

val cipherSpec = PBEParameterSpec(salt, 10000) // Compli
val spec = PBEKeySpec(password, salt, 10000, 256) // Com
```

See

- OWASP Top 10 2021 Category A2 - Cryptographic Failures
- OWASP Top 10 2017 Category A3 - Sensitive Data Exposure
- MITRE, CWE-759 - Use of a One-Way Hash without a Salt
- MITRE, CWE-760 - Use of a One-Way Hash with a Predictable Salt
- SANS Top 25 - Porous Defenses

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