sonar RULES

Products

- 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

"SecureRandom" seeds should not be predictable

Analyze your code

Vulnerability   Critical   cwe owasp pitfall

The `java.security.SecureRandom` class provides a strong random number generator (RNG) appropriate for cryptography. However, seeding it with a constant or another predictable value will weaken it significantly. In general, it is much safer to rely on the seed provided by the `SecureRandom` implementation.

This rule raises an issue when `SecureRandom.setSeed()` or `SecureRandom(byte[])` are called with a seed that is either one of:

- a constant
- the system time

Noncompliant Code Example

```
val sr = SecureRandom()
sr.setSeed(123456L) // Noncompliant
val v = sr.nextInt()
```

```
val sr = SecureRandom("abcdefghijklmnop".toByteArray(cha
val v = sr.nextInt()
```

Compliant Solution

```
val sr = SecureRandom()
val v = sr.nextInt()
```

See

- OWASP Top 10 2021 Category A2 - Cryptographic Failures
- OWASP Top 10 2017 Category A6 - Security Misconfiguration
- MITRE, CWE-330 - Use of Insufficiently Random Values
- MITRE, CWE-332 - Insufficient Entropy in PRNG
- MITRE, CWE-336 - Same Seed in Pseudo-Random Number Generator (PRNG)
- MITRE, CWE-337 - Predictable Seed in Pseudo-Random Number Generator (PRNG)
- CERT, MSC63J. - Ensure that SecureRandom is properly seeded

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