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

Weak SSL/TLS protocols should not be used

Analyze your code

Vulnerability | Critical | cwe privacy owasp sans-top25

This rule raises an issue when an insecure TLS protocol version (i.e. a protocol different from "TLSv1.2", "TLSv1.3", "DTLSv1.2", or "DTLSv1.3") is used or allowed.

It is recommended to enforce TLS 1.2 as the minimum protocol version and to disallow older versions like TLS 1.0. Failure to do so could open the door to downgrade attacks: a malicious actor who is able to intercept the connection could modify the requested protocol version and downgrade it to a less secure version.

Noncompliant Code Example

`javax.net.ssl.SSLContext` library:

```
val sc: SSLContext = SSLContext.getInstance("TLSv1.1") /
```

okhttp library:

```
val spec: ConnectionSpec = ConnectionSpec.Builder(Connec
                .tlsVersions(TlsVersion.TLS_1_1)  // Non
                .build()
```

Compliant Solution

`javax.net.ssl.SSLContext` library:

```
val sc: SSLContext = SSLContext.getInstance("TLSv1.2") /
```

okhttp library:

```
val spec: ConnectionSpec = ConnectionSpec.Builder(Connec
                .tlsVersions(TlsVersion.TLS_1_2)  // Com
                .build()
```

See

- OWASP Top 10 2021 Category A2 - Cryptographic Failures
- OWASP Top 10 2021 Category A7 - Identification and Authentication Failures
- OWASP Top 10 2017 Category A3 - Sensitive Data Exposure
- OWASP Top 10 2017 Category A6 - Security Misconfiguration
- Mobile AppSec Verification Standard - Network Communication Requirements
- OWASP Mobile Top 10 2016 Category M3 - Insecure Communication
- MITRE, CWE-327 - Inadequate Encryption Strength
- MITRE, CWE-326 - Use of a Broken or Risky Cryptographic Algorithm
- SANS Top 25 - Porous Defenses
- SSL and TLS Deployment Best Practices - Use secure protocols

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