



**CSS** 

XFlex

Go

Java

5

JavaScript

**HTML** 

Kotlin

Kubernetes

Objective C

PHP

PL/I

PL/SQL

Python

**RPG** 

Ruby

Scala

**Swift** 

**Terraform** 

Text

**TypeScript** 

T-SQL

**VB.NET** 

VB6

XML



# **Swift static code analysis**

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

All rules (119) Code Smell 99 6 Vulnerability (3) **R** Bug (14) Security Hotspot (3)

Tags

Hard-coded credentials are securitysensitive Security Hotspot

Methods and field names should not be the same or differ only by capitalization

Code Smell

Cipher algorithms should be robust

Vulnerability

Using weak hashing algorithms is security-sensitive

Security Hotspot

**Cognitive Complexity of functions** should not be too high

Code Smell

"try!" should not be used

Code Smell

String literals should not be duplicated

Code Smell

Functions and closures should not be empty

Code Smell

Collection elements should not be replaced unconditionally

📆 Bug

Collection sizes comparisons should make sense

👬 Bug

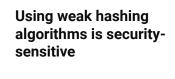
All branches in a conditional structure should not have exactly the same implementation

👬 Bug

Infix operators that end with "=" should update their left operands

**Bug** 

Precedence and associativity of standard operators should not be changed



Analyze your code

Security Hotspot

cwe spring owasp sans-top25

Search by name...

Cryptographic hash algorithms such as MD2, MD4, MD5, MD6, HAVAL-128, HMAC-MD5, DSA (which uses SHA-1), RIPEMD, RIPEMD-128, RIPEMD-160, HMACRIPEMD160 and SHA-1 are no longer considered secure, because it is possible to have collisions (little computational effort is enough to find two or more different inputs that produce the same hash).

#### **Ask Yourself Whether**

The hashed value is used in a security context like:

- User-password storage.
- Security token generation (used to confirm e-mail when registering on a website, reset password, etc ...).
- To compute some message integrity.

There is a risk if you answered yes to any of those questions.

### **Recommended Secure Coding Practices**

Safer alternatives, such as SHA-256, SHA-512, SHA-3 are recommended, and for password hashing, it's even better to use algorithms that do not compute too "quickly", like bcrypt, scrypt, argon2 or pbkdf2 because it slows down brute force attacks.

### **Sensitive Code Example**

```
import CryptoSwift
let bytes:Array<UInt8> = [0x01, 0x02, 0x03]
let digest = input.md5() // Sensitive
```

### **Compliant Solution**

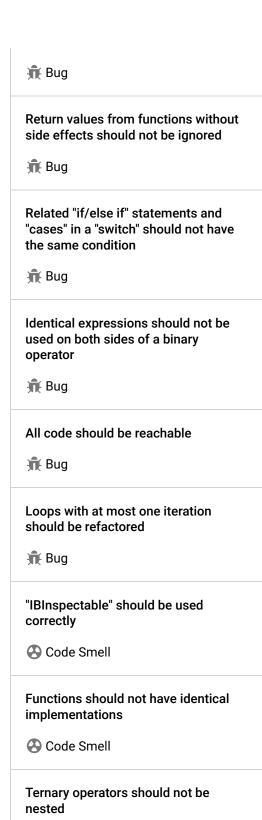
```
import CryptoSwift
let bytes:Array<UInt8> = [0x01, 0x02, 0x03]
let digest = input.sha512() // Compliant
```

## See

- OWASP Top 10 2021 Category A2 Cryptographic Failures
- OWASP Top 10 2017 Category A3 Sensitive Data Exposure
- 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-1240 Use of a Risky Cryptographic Primitive
- SANS Top 25 Porous Defenses

Available In:

sonarcloud 🚳 | sonarqube Developer Edition



Code Smell

nested too deeply

Code Smell

Closure expressions should not be

Backticks should not be used around

© 2008-2022 SonarSource S.A., Switzerland. All content is copyright protected. SONAR, SONARSOURCE, SONARLINT, SONARQUBE and SONARCLOUD are trademarks of SonarSource S.A. All other trademarks and copyrights are the property of their respective owners. All rights are expressly reserved.

Privacy Policy