

- 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



C static code analysis

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

All rules **311**

Vulnerability **13**

Bug **74**

Security Hotspot **18**

Code Smell **206**

Quick Fix **14**

Tags

Search by name...



"memset" should not be used to delete sensitive data

Vulnerability

POSIX functions should not be called with arguments that trigger buffer overflows

Vulnerability

XML parsers should not be vulnerable to XXE attacks

Vulnerability

Function-like macros should not be invoked without all of their arguments

Bug

The address of an automatic object should not be assigned to another object that may persist after the first object has ceased to exist

Bug

"pthread_mutex_t" should be unlocked in the reverse order they were locked

Bug

"pthread_mutex_t" should be properly initialized and destroyed

Bug

"pthread_mutex_t" should not be consecutively locked or unlocked twice

Bug

Functions with "noreturn" attribute should not return

Bug

"memcpy" should only be called with pointers to trivially copyable types with no padding

Bug

Using weak hashing algorithms is security-sensitive

Analyze your code

Security Hotspot Critical cwe spring owasp sans-top25

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

```
#include <botan/hash.h>
// ...

Botan::secure_vector<uint8_t> f(std::string input){
    std::unique_ptr<Botan::HashFunction> hash(Botan::HashFunc
    return hash->process(input);
}
```

Compliant Solution

```
#include <botan/hash.h>
// ...

Botan::secure_vector<uint8_t> f(std::string input){
    std::unique_ptr<Botan::HashFunction> hash(Botan::HashFunc
    return hash->process(input);
}
```

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

Stack allocated memory and non-owned memory should not be freed

 Bug

Closed resources should not be accessed

 Bug

Dynamically allocated memory should be released

 Bug

Freed memory should not be used

- [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

© 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](#)