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

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

Insecure functions should not be used

Analyze your code

Vulnerability | Critical | cwe sans-top25 owasp cert

When using typical C functions, it's up to the developer to make sure the size of the buffer to be written to is large enough to avoid buffer overflows. Buffer overflows can cause the program to crash at a minimum. At worst, a carefully crafted overflow can cause malicious code to be executed.

This rule reports use of the following insecure functions, for which knowing the required size is not generally possible: `gets()` and `getpw()`.

In such cases. The only way to prevent buffer overflow while using these functions would be to control the execution context of the application.

It is much safer to secure the application from within and to use an alternate, secure function which allows you to define the maximum number of characters to be written to the buffer:

- `fgets` or `gets_s`
- `getpwuid`

Noncompliant Code Example

```
gets(str); // Noncompliant; `str` buffer size is not checked
```

Compliant Solution

```
gets_s(str, sizeof(str)); // Prevent overflows by enforcing a
```

See

- OWASP Top 10 2017 Category A9 - Using Components with Known Vulnerabilities
- MITRE, CWE-676 - Use of Potentially Dangerous Function
- MITRE, CWE-119 - Improper Restriction of Operations within the Bounds of a Memory Buffer
- SANS Top 25 - Risky Resource Management

Available In:

sonarlint | sonarcloud | sonarqube Developer Edition

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