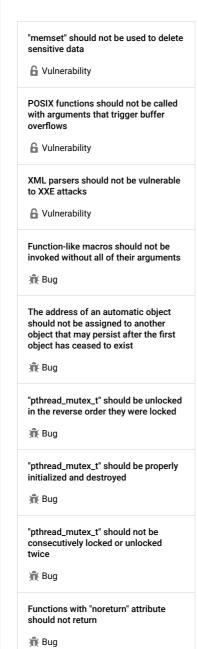




C static code analysis

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





"memcmp" should only be called with

pointers to trivially copyable types

with no padding

🖷 Bug



Operating systems have global directories where any user has write access. Those folders are mostly used as temporary storage areas like /tmp in Linux based systems. An application manipulating files from these folders is exposed to race conditions on filenames: a malicious user can try to create a file with a predictable name before the application does. A successful attack can result in other files being accessed, modified, corrupted or deleted. This risk is even higher if the application runs with elevated permissions.

In the past, it has led to the following vulnerabilities:

- CVE-2012-2451
- CVE-2015-1838

This rule raises an issue whenever it detects a hard-coded path to a publicly writable directory like /tmp (see examples bellow). It also detects access to environment variables that point to publicly writable directories, e.g., TMP and TMPDIR.

- /tmp
- /var/tmp
- /usr/tmp
- /dev/shm
- /dev/mqueue
- /run/lock
- /var/run/lock
- /Library/Caches
- /Users/Shared
- /private/tmp
- /private/var/tmp
- \Windows\Temp
- \Temp
- \TMP

Ask Yourself Whether

- Files are read from or written into a publicly writable folder
- The application creates files with predictable names into a publicly writable folder

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

Recommended Secure Coding Practices

- Use a dedicated sub-folder with tightly controlled permissions
- Use secure-by-design APIs to create temporary files. Such API will make sure:
 - $\circ\,$ The generated filename is unpredictable
 - $\circ\,$ The file is readable and writable only by the creating user ID
 - $\,\circ\,$ The file descriptor is not inherited by child processes
 - $\circ\,$ The file will be destroyed as soon as it is closed

Sensitive Code Example

#include <cstdio>

Stack allocated memory and nonowned 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

```
// ...
void f() {
 FILE * fp = fopen("/tmp/temporary_file", "r"); // Sensitive
```

```
#include <cstdio>
#include <cstdlib>
#include <sstream>
// ...
void f() {
 std::stringstream ss;
  ss << getenv("TMPDIR") << "/temporary_file"; // Sensitive
 FILE * fp = fopen(ss.str().c_str(), "w");
}
```

Compliant Solution

```
#include <cstdio>
#include <cstdlib>
void f() {
  FILE * fp = tmpfile(); // Compliant
```

See

- OWASP Top 10 2021 Category A1 Broken Access Control
- OWASP Top 10 2017 Category A5 Broken Access Control
- OWASP Top 10 2017 Category A3 Sensitive Data Exposure
- MITRE, CWE-377 Insecure Temporary File
- $\bullet \ \underline{\text{MITRE, CWE-379}} \ \text{-} \ \text{Creation of Temporary File in Directory with Incorrect}$ Permissions
- OWASP, Insecure Temporary File

Available In:

sonardoud 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