C# static code analysis

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

Using publicly writable directories is security-sensitive

Analyze your code

Security Hotspot Critical cwe owasp

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`, `TMPDIR` and `TEMP`.

- `/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`
- `%USERPROFILE%\AppData\Local\Temp`

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

Out of the box, .NET is missing secure-by-design APIs to create temporary files. To overcome this, one of the following options can be used:

- Use a dedicated sub-folder with tightly controlled permissions
- Created temporary files in a publicly writable folder and make sure:
  - Generated filename is unpredictable
  - File is readable and writable only by the creating user ID
  - File descriptor is not inherited by child processes
  - File is destroyed as soon as it is closed

"IDisposables" created in a "using" statement should not be returned

Bug

"ThreadStatic" fields should not be initialized

Bug

"Object.ReferenceEquals" should not be used for value types

Bug

Doubled prefix operators "!!" and "~~" should not be used

Bug

Sensitive Code Example

```
using var writer = new StreamWriter("/tmp/f"); // Sensitive
```

```
var tmp = Environment.GetEnvironmentVariable("TMP"); // Sens
```

Compliant Solution

```
var randomPath = Path.Combine(Path.GetTempPath(), Path.GetRa

// Creates a new file with write, non inheritable permission
using var fileStream = new FileStream(randomPath, FileMode.C
using var writer = new StreamWriter(fileStream);
```

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
- MITRE, CWE-379 - Creation of Temporary File in Directory with Incorrect Permissions
- OWASP, Insecure Temporary File

Available In:

sonarcloud sonarqube