- Secrets
- ABAP
- Apex
- C
- C++
- CloudFormation
- COBOL
- C#
- CSS
- Flex
- Go
- HTML
- Java
- JavaScript
- Kotlin
- 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 409 | Vulnerability 34 | Bug 76 | Security Hotspot 28 | Code Smell 271 | Quick Fix 52

Tags | Search by name...

Windows Forms entry points should be marked with STAThread
Bug

Collection elements should not be replaced unconditionally
Bug

Exceptions should not be created without being thrown
Bug

Collection sizes and array length comparisons should make sense
Bug

Serialization event handlers should be implemented correctly
Bug

Deserialization methods should be provided for "OptionalField" members
Bug

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

Types should be defined in named namespaces
Bug

Empty nullable value should not be accessed
Bug

Nullable type comparison should not be redundant
Bug

Methods with "Pure" attribute should return a value
Bug

One-way "OperationContract" methods

Expanding archive files without controlling resource consumption is security-sensitive

Analyze your code

Security Hotspot   Critical   cwe owasp

Successful Zip Bomb attacks occur when an application expands untrusted archive files without controlling the size of the expanded data, which can lead to denial of service. A Zip bomb is usually a malicious archive file of a few kilobytes of compressed data but turned into gigabytes of uncompressed data. To achieve this extreme compression ratio, attackers will compress irrelevant data (eg: a long string of repeated bytes).

Ask Yourself Whether

Archives to expand are untrusted and:

- There is no validation of the number of entries in the archive.
- There is no validation of the total size of the uncompressed data.
- There is no validation of the ratio between the compressed and uncompressed archive entry.

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

Recommended Secure Coding Practices

- Define and control the ratio between compressed and uncompressed data, in general the data compression ratio for most of the legit archives is 1 to 3.
- Define and control the threshold for maximum total size of the uncompressed data.
- Count the number of file entries extracted from the archive and abort the extraction if their number is greater than a predefined threshold, in particular it's not recommended to recursively expand archives (an entry of an archive could be also an archive).

Sensitive Code Example

```
using var zipToOpen = new FileStream(@"ZipBomb.zip", FileMod
using var archive = new ZipArchive(zipToOpen, ZipArchiveMode
foreach (ZipArchiveEntry entry in archive.Entries)
{
  entry.ExtractToFile("./output_onlyfortesting.txt", true);
}
```

Compliant Solution

```
int THRESHOLD_ENTRIES = 10000;
int THRESHOLD_SIZE = 1000000000; // 1 GB
double THRESHOLD_RATIO = 10;
int totalSizeArchive = 0;
int totalEntryArchive = 0;

using var zipToOpen = new FileStream(@"ZipBomb.zip", FileMod
using var archive = new ZipArchive(zipToOpen, ZipArchiveMode
foreach (ZipArchiveEntry entry in archive.Entries)
```

should have "void" return type

Bug

Optional parameters should be passed to "base" calls

Bug

Classes should not have only "private" constructors

Bug

Expressions used in "Debug.Assert" should not produce side effects

Bug

Caller information parameters should

```
{
  totalEntryArchive ++;

  using (Stream st = entry.Open())
  {
    byte[] buffer = new byte[1024];
    int totalSizeEntry = 0;
    int numBytesRead = 0;

    do
    {
      numBytesRead = st.Read(buffer, 0, 1024);
      totalSizeEntry += numBytesRead;
      totalSizeArchive += numBytesRead;
      double compressionRatio = totalSizeEntry / entry.Compr

      if(compressionRatio > THRESHOLD_RATIO) {
        // ratio between compressed and uncompressed data is
        break;
      }
    }
    while (numBytesRead > 0);
  }

  if(totalSizeArchive > THRESHOLD_SIZE) {
      // the uncompressed data size is too much for the appl
      break;
  }

  if(totalEntryArchive > THRESHOLD_ENTRIES) {
      // too much entries in this archive, can lead to inode
      break;
  }
}
```

See

- OWASP Top 10 2021 Category A1 - Broken Access Control
- OWASP Top 10 2021 Category A5 - Security Misconfiguration
- OWASP Top 10 2017 Category A6 - Security Misconfiguration
- MITRE, CWE-409 - Improper Handling of Highly Compressed Data (Data Amplification)
- bamsoftware.com - A better Zip Bomb

Available In:

sonarcloud sonarqube