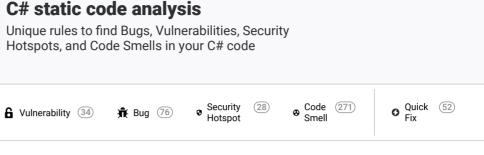
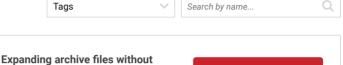


One-way "OperationContract" methods





controlling resource consumption is security-sensitive

Analyze your code

Security Hotspot Oritical Concerning was 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:

- $\bullet\,\,$ There is no validation of the number of entries in the archive.
- $\bullet \;\;$ 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)
```

snould have void return type

Rug 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;
    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
}
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
- <u>MITRE, CWE-409</u> Improper Handling of Highly Compressed Data (Data Amplification)
- <u>bamsoftware.com</u> A better Zip Bomb

Available In:

sonarcloud 🔂 | sonarqube

© 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