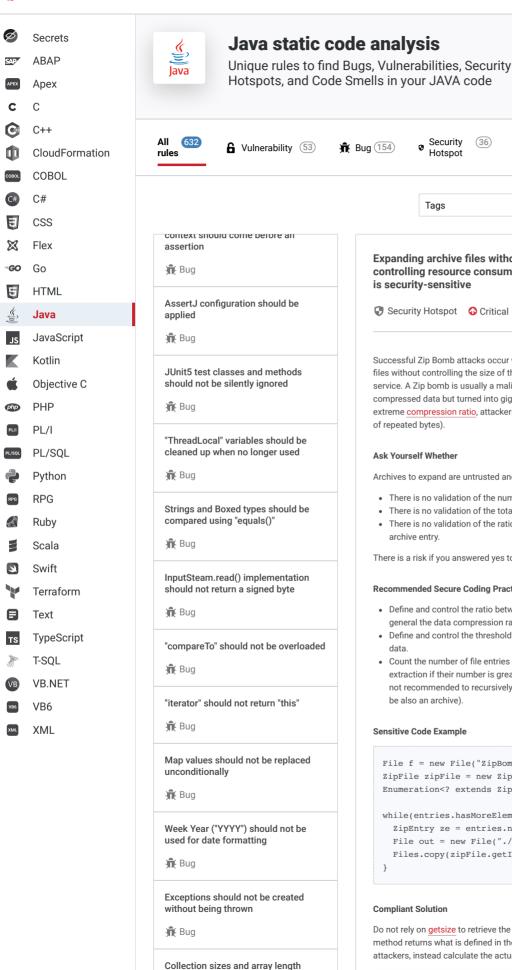
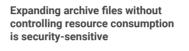


Products ✓





Security Hotspot Oritical

Security

Hotspot

Tags

(36)

Analyze your code

O Quick 42 Fix

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

⊗ Code

Smell

(389)

Search by name.

cwe cert owasp

Ask Yourself Whether

of repeated bytes).

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
- . 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

```
File f = new File("ZipBomb.zip");
ZipFile zipFile = new ZipFile(f);
Enumeration<? extends ZipEntry> entries = zipFile.entries();
while(entries.hasMoreElements()) {
 ZipEntry ze = entries.nextElement();
  File out = new File("./output_onlyfortesting.txt");
  Files.copy(zipFile.getInputStream(ze), out.toPath(), Stand
```

Compliant Solution

Do not rely on getsize to retrieve the size of an uncompressed entry because this method returns what is defined in the archive headers which can be forged by attackers, instead calculate the actual entry size when unzipping it:

```
File f = new File("ZipBomb.zip");
ZipFile zipFile = new ZipFile(f);
Enumeration<? extends ZipEntry> entries = zipFile.entries();
```

comparisons should make sense

Consumed Stream pipelines should not be reused

Rug

Intermediate Stream methods should not be left unused

Rug

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

Rug Bug

Optional value should only be accessed after calling isPresent()

Bug

```
int THRESHOLD ENTRIES = 10000;
int THRESHOLD_SIZE = 1000000000; // 1 GB
double THRESHOLD RATIO = 10;
int totalSizeArchive = 0:
int totalEntryArchive = 0;
while(entries.hasMoreElements()) {
  ZipEntry ze = entries.nextElement();
  InputStream in = new BufferedInputStream(zipFile.getInputS
  OutputStream out = new BufferedOutputStream(new FileOutput
  totalEntrvArchive ++:
  int nBytes = -1;
  byte[] buffer = new byte[2048];
  int totalSizeEntry = 0;
  while((nBytes = in.read(buffer)) > 0) { // Compliant
      out.write(buffer, 0, nBytes);
      totalSizeEntry += nBytes;
      totalSizeArchive += nBytes;
      double compressionRatio = totalSizeEntry / ze.getCompr
      if(compressionRatio > THRESHOLD RATIO) {
        // ratio between compressed and uncompressed data is
  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
- MITRE, CWE-409 Improper Handling of Highly Compressed Data (Data Amplification)
- CERT, IDS04-J. Safely extract files from ZipInputStream
- bamsoftware.com 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