sonar RULES

Products

- 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

Java static code analysis

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

All rules 632 | Vulnerability 53 | Bug 154 | Security Hotspot 36 | Code Smell 389 | Quick Fix 42

Tags | Search by name...

context should come before an assertion
Bug

AssertJ configuration should be applied
Bug

JUnit5 test classes and methods should not be silently ignored
Bug

"ThreadLocal" variables should be cleaned up when no longer used
Bug

Strings and Boxed types should be compared using "equals()"
Bug

InputSteam.read() implementation should not return a signed byte
Bug

"compareTo" should not be overloaded
Bug

"iterator" should not return "this"
Bug

Map values should not be replaced unconditionally
Bug

Week Year ("YYYY") should not be used for date formatting
Bug

Exceptions should not be created without being thrown
Bug

Collection sizes and array length comparisons should make sense
Bug

Expanding archive files without controlling resource consumption is security-sensitive

Analyze your code

Security Hotspot Critical cwe cert 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

```
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();
```

Consumed Stream pipelines should not be reused

Bug

Intermediate Stream methods should not be left unused

Bug

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

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

  totalEntryArchive ++;

  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
        break;
      }
  }

  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)
- CERT, IDS04-J. - Safely extract files from ZipInputStream
- bamsoftware.com - A better Zip Bomb

Available In:

sonarcloud | sonarqube