NOTE: SCENARIO IS WHAT THE PUZZLE TAKER SEES.

SCENARIO:

You are developing a secure library for compression/decompression operations. All methods in the library require that the size of the original file (uncompressed) is at most 100MB, otherwise they throw an exception. The `decompress` method takes a `String` value as the path of a zip file, and a `String` value as the path of destination directory, and decompresses the zip file into the given directory. Consider the snippet of code below and answer the following questions, assuming that the code has all required permissions to execute and there is no `IOException`.

```java
// OMITTED: Import whatever is needed.
public final class ZipUtils {
  private static final int BUFFER = 0x1000; // 4KB
  private static final int MAX_FILE_SIZE = 0x6400000; // 100MB

  public static void decompress (String path, String directory)
      throws IOException {
    FileInputStream fis = new FileInputStream(path);
    ZipInputStream zis = new ZipInputStream(fis);
    try {
      ZipEntry entry;
      while ((entry = zis.getNextEntry()) != null) {
         if (entry.getSize() > MAX_FILE_SIZE)
           throw new IllegalStateException("File is too large.") ;

         // OMITTED: Store the "entry" in the "directory"
      }
    }
    finally {
       //OMITTED: close the streams
    }
  }
}
```

Questions:
1. What does the `decompress` method do when executed?

2. What will happen if the `decompress` method is called given a zip file containing an entry (named `big.txt`) larger than 100MB?
a. `big.txt` won't be decompressed and the method will throw an exception.
b. `big.txt` may or may not be decompressed.
c. `big.txt` will definitely be decompressed.
d. The program will crash after the invocation of the `decompress` method.
e. None of the above

[Other statistical questions will be imported here while creating survey.]

NOTE: ANSWER IS TO BE SHOWN TO THE PUZZLE TAKER AT THE END OF THE SESSION.

ANSWER:
b
The file (`big.txt`) still might be decompressed, since we are assigning the next file entry to zipEntry, we cannot rely on the `getSize` method. The `getSize` method may return -1 if the size of the entry (file) is unknown, so the above code is highly vulnerable as the attacker can modify the file size.

NOTE: THE REST OF THIS DOCUMENT CONTAINS EXTRA INFORMATION FOR THE PROJECT RESEARCHERS. IT IS NOT TO BE SHOWN TO THE PUZZLE TAKERS.

TAGS:
java, zipinputstream, dos-attack

CATEGORIES:
Blindspot - YES
Type - Compression
Number of distinct functions - 5
Number of total functions - 5
Blindspot function - `ZipEntry.getSize()`
Function call omitted - NO
Blindspot type - Return type incorrectly checked
Number of Parameters in the blindspot function - 0 parameters
Cyclomatic complexity - 4

NAME:

NAME:
ZipEntry class, getSize method - Returns the uncompressed size of the entry data, or -1 if not known.

DESCRIPTION:
Java provides the `java.util.zip` package for zip-compatible data compression. It provides classes that enable you to read, create, and modify ZIP and GZIP files formats. `ZipInputStream` functions the same as `InputStream`, and `ZipEntry` is basically used to represent a Zip file entry.

BLINDSPOT:
The `getSize` method of the `ZipEntry` class is not dependable because it returns -1 when the size of the entry (file) is unknown. It is also possible that an attacker forges the field in the zip entry. This could lead to DoS attack or data corruption.

CORRECT USE EXAMPLE:
#N/A

MORE INFORMATION:
#N/A

REFERENCES:

1. ZipEntry#getSize
2. https://goo.gl/hi29lb