Detection Algorithm for Non-recursive Zip Bombs

1st MaoYang Chen
University of Electronic Science and Technology of China
ChengDu, China
maplejack@qq.com

2nd MingYu Fan
University of Electronic Science and Technology of China
ChengDu, China
ff98@163.com

***Abstract*—Traditional compression bombs often work by recursive decompression, so the usual defensive way is by single decompression. However, a new type of compression bombs has recently appeared, which can take effect with a single decompression. We show the two structures of this type of compression bombs and provide the basic idea of detecting such bombs. At the same time, we point out the details in need of attention in the detection process as well. Moreover, we propose a detection algorithm for this type of bombs and we analyze the accuracy and detection efficiency of this algorithm.**

***Index Terms*—Software safety; File viruses; Compression bombs.**

I. Introduction

Compression bombs are the compressed files that have a very high compression ratio and can generate a huge amount of invalid data after decompression. They can be used to cause many serious problems, such as buffer overflows, memory leaks. Some bombs even come with Trojan horse programs [4]. Sometimes, compression bombs are also used as email bombs which will lead to denial of service [3].

The zip bomb is a typical type of compression bombs. Zip bombs can be divided into two types: recursive zip bombs and non-recursive zip bombs [1].

The recursive zip bombs also fall into two types. The first type is the compression bombs, represented by the most famous zip bomb, 42.zip [7]. The original size of 42.zip is 0.6MB before decompression, but after six layers of recursive decompression, it will expand to 4.5PB. The second type is zip quines, whose feature is that once they are decompressed recursively, they will copy themselves infinitely. A typical example of zip quines is the bomb mentioned in Zip Files All The Way Down [2]. These two types of zip bombs have an obvious disadvantage: as long as they are not recursively decompressed, they will not take effect.

To overcome the obvious disadvantage, David Fifield proposes the non-recursive zip bomb [1]. This type of zip bombs uses a special structure "overlap" so that they can still work when being single decompressed. However, there is only a little decompression software currently providing detection services for non-recursive zip bombs. To the best of our knowledge, only Mark Adler has written a patch for unzip [9]. In Table 1, we test whether mainstream compression software can prevent the non-recursive zip bombs. For anti-virus software, when Kaspersky, 360 total security and Windows Defender scan these bombs, they all decompress the bombs. However, only Kaspersky can detect and defuse the bomb. Meanwhile,

TABLE I
Software Detection Result

Software	bandizip	unzip(unpatched)	360zip	tar	winrar	7-zip
bomb status	work	work	work	work	work	work

these software cannot prevent such bombs from taking effect when the existing non-recursive bombs are decompressed. So, lots of work should been done to defend against non-recursive zip bombs and it makes sense to detect this type of bombs before decompression.

This paper introduces the working principle of the non-recursive zip bombs and the detection of this type of bombs without any decompression software's help. Unlike the anti-virus programs mentioned, our detection method does not need to decompress the zip bomb.

The rest of this paper is organized as follows. The features of such bombs are outlined in Section 2. Section 3 introduces how to detect these bombs. Section 4 presents the details of the detection algorithm. Finally, we summarize our work in Section 5.

II. Non-recursive zip bomb

Before detecting zip bombs, we need to know the structures of info-zip (standard zip file) and non-recursive zip bombs.

A. The structure of info-zip

As in Figure 1, an info-zip is composed of several local file entries, a central directory and an end of central directory record [6]. Each file in an info-zip has one local file entry and one file header in the central directory. A local file entry consists of a local file header and the corresponding file data. The local file header records the metadata of the file and the file data is actually the compressed data of the origin file. The central directory is a series of file headers, which records

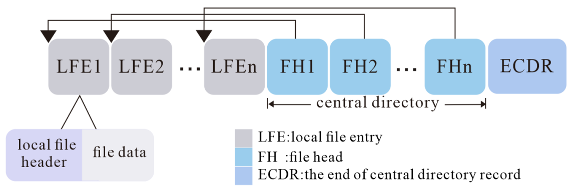


Fig. 1. The structure of info-zip

not only the metadata of the file but also the position of the corresponding local file header.

B. *The structure of non-recursive zip bomb*

The non-recursive zip bombs use a special structure to make the files overlap. Such a bomb consists of many local file headers, one file data, a central directory and an end of central directory record, as in Figure 2 [1]. The bomb resets the length

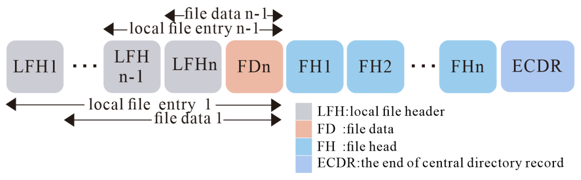


Fig. 2. The structure 1 of the non-recursive zip bombs

of each local file entry so that all local file entries overlap. Supposing there is a bomb containing n files, its i-th local file entry will consist of the i-th local file header, the i+1-th file header, ..., the n-th file header and file data n. In other words, the i+1-th local file header, ..., the n-th local file header and file data n constitute the file data of the i-th local file entry, as in Figure 2. Due to the special structure of overlapping files, the type of bombs has the characteristics of high compression ratio and it works directly after a single decompression. After

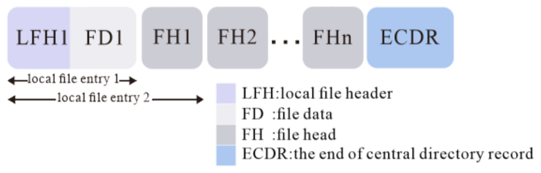


Fig. 3. The structure 2 of the non-recursive zip bombs

analyzing about 1000 non-recursive zip bombs, we find that there are some non-recursive zip bombs which have a different structure. As in Figure 3, the bombs have only one local file header and all the file headers point to this local file header. Of course, such bombs with this structure also have overlap. For example, the first local file entry consists of local file header 1 and file data 1. The second consists of local file header 1, file data 1 and file header 1. In other words, the 2nd local file entry's file data is file data 1 plus file header 1. The overlap is the whole local file entry 1. Such a type of bombs have a higher compression ratio.

III. How to detect the non-recursive bomb

This section introduces the basic idea of the detection and some questions that need attention.

A. *The basic idea*

According to the previous description, overlap is the most important sign of the non-recursive zip bombs. Therefore, we

TABLE II
File Header Structure

Offset	Bytes	Description
0	4	Signature(0x02014b50)
...	...	...
42	4	The position of the local file header
46	m	File name
46+m	n	Extra Field

TABLE III
Local File Header Structure

Offset	Bytes	Description
0	4	Signature(0x04034b50)
...	...	...
18	4	File data size
...	...	...
26	2	File name length
28	2	Extra field length
30	n	File name
30+n	m	Extra Field

need to check whether the detected zip bomb is a non-recursive zip bomb by detecting the overlap among the local file entries.

The first thing to do is to get the position of each local file entries. The file header can help us. In Table 2, the offset 42 in the file header records "The position of the local file header". This is the position we need.

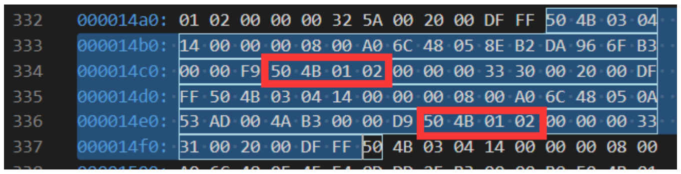
Second, the length of each local file entry should be known. We cannot directly get this value. For the local file entry consists of the local file header and the file data, we just need to find out the length of the two parts. The length of the local file header is 30 bytes' fixed length plus the file name length and extra field length. In Table 3, the two lengths are both recorded in the local file header. The file data size is also recorded in the local file header (in some documents, the file data length is called compressed size). Then we can calculate the length of a local file entry.

Now, since we get the position of the local file header and the length of the local file entry, the last thing to do is to determine whether the overlap exists. For a pair of adjacent local file entries, if the end of the previous local file entry covers the start of the next one, then there is an overlap and the start position of a local file entry is just the position of its local file header. We can easily calculate the position of the end of the previous local file entry and compare it with the start of next one.

B. *Some questions that need attention*

First of all, we should open the zip file in hex because our idea requires the usage of the zip file structure in hex. Referring to Table 2, we can find out these file headers by their signature (0x02014b50). Here, we need to consider the problem of small-endian and big-endian. For the small-endian, the signature used to search the file headers is actually "0x504b0102" instead of "0x02014b50".

There is an important question: what should we do if some data segments have values exactly equal to 0x504b0102 but are not a real file header signature. Although the probability of this situation is very small, it should not be ignored. Referring to Figure 4, we happen to encounter such a situation. The data segment from 0x14ac to 0x14f5 is two local file headers in a non-recursive bomb and “file header signature” exists in the segment: the first is from 0x14c3 to 0x14c6 and the second is from 0x14e8 to 0x14eb. So to solve the above problem, we do such a thing: after finding out a file header by “0x02014b50”, according to the file header’s offset 42, we get and check the position of the local file header(refer to Table 2). If the first four bytes of the position are not “0x04034b50” or the position exceeds the size of the entire zip file, this file header must be fake and can be ignored. Another question to note is the search



```
000014a0: 01 02 00 00 00 32 5A 00 20 00 DF FF 50 4B 03 04
000014b0: 14 00 00 00 08 00 A0 6C 48 05 8E B2 DA 96 6F B3
000014c0: 00 00 F9 50 4B 01 02 00 00 00 33 30 00 20 00 DF
000014d0: FF 50 4B 03 04 14 00 00 00 08 00 A0 6C 48 05 0A
000014e0: 53 AD 00 4A B3 00 00 D9 50 4B 01 02 00 00 00 33
000014f0: 31 00 20 00 DF FF 50 4B 03 04 14 00 00 00 08 00
```

Fig. 4. Fake file header

method. The most important information is the positions of each file header because according to them, we can find the information we need. Since the file headers are concentrated at the end of the zip file, we choose to search the zip file from its end to its beginning for file headers. Obviously, we only need to search from the end of the file till we find the first file header, instead of searching the whole zip file. Thus, the key point is to know how to confirm whether a file header is the first one. We can use a trick: the position of the first local file header is always at the beginning of the zip file. In other words, if the file header is the first one, its offset 42 which records the position of the corresponding local file header is 0. So, when the offset 42 of a file header is 0, we just stop the search. But this trick does not work on the non-recursive zip bombs with the second structure because the offset 42 of each file header is always 0, as in Figure 3. Therefore, when finding out a file header whose offset 42 is 0, we should check whether there is another real file header before it. If such a file header appears, there must be overlapping files in the detected zip file.

The last question is whether it is necessary to detect all adjacent file headers in the zip file. For the non-recursive zip bombs that currently have two structures, it seems that we only need to check the last two files headers to identify whether they are non-recursive zip bombs. But we consider that it is necessary to check all the adjacent files headers, because if only the last two files headers are detected, the bomb maker can easily bypass our detection by a certain amount of forgery.

IV. Algorithm

This section describes the details of the algorithm, show the accuracy and the efficiency of the algorithm.

A. Algorithm Description

Previously, we explain the basic idea of detection and some questions that need attention and then let’s organize the idea. First, we need to search reversely to get the first pair of adjacent file headers. We define this process as a function named “GetAdjacentFileHeader” which requires a position pointer and returns a list. This list contains the positions of the first pair of adjacent file headers which are reversely searched from this position pointer. At the same time, the function will change the position pointer’s value: if such a pair of file headers can be found, supposing the file header at the front is called FH1 and the file header at the back is called FH2, at the end of the function, this pointer is set to point to the end of FH1. If not, the value is set to -1 as the sign of the search’s end and the list returned is empty.

Algorithm 1: The Algorithm for Detecting the Non-recursive Zip Bomb

```
Input: a zip file to be detected: $File$
Output: the flag indicating whether the file is a
        non-recursive zip bomb: $Flag$
Function GetAdjacentFileHeader($pointer$);
Function CheckOverlap($list$);
Initialize $Flag$ to False;
Initialize $CurPos$ to point to the zip file's end;
Preprocessing the content of $File$;
Initialize $List$ to Null;
while $CurPos$ is not -1 do
    $List$ = GetAdjacentFileHeader($CurPos$);
    $Flag$ = CheckOverlap($List$);
    if $Flag$ then
        Break;
end
```

Fig. 5. The detection algorithm

Secondly, after we get a pair of adjacent file headers, we should check if their corresponding local file entries overlap. We define this process as a function called “CheckOverlap”. This function needs a list which consists of the positions of two file headers and returns a Boolean value indicating whether there is overlap. Specifically, for the two file headers included in the list, we suppose the one at the front is called FH1 and the one at the back is called FH2. This function will get the start position of their corresponding local file entries LF1, LF2. Then it will calculate and get the end position of LF1. If the end position of LF1 exceeds the start position of LF2, this function will return True because LF1 and LF2 are overlapping. If not, it will return False.

Finally, for the whole zip file, we should traverse all adjacent file headers unless file overlaps have already been detected. To achieve it, we initialize a flag called Flag and a position pointer CurPos. Flag is set to False initially, which is used as the output of the entire algorithm to indicate whether the detected zip file is a non-recursive zip bomb. CurPos is initially set to the end of the zip file. It is passed to the function GetAdjacentFileHeader as the starting point for reverse search and when its value is -1, the search will stop and the algorithm returns Flag. We summarize the steps in Figure 5.

B. Algorithm Accuracy

To check the accuracy of the algorithm, we prepare two sets of samples. One is full of info-zip and the other is full of the non-recursive zip bombs.

TABLE IV
THE DETECTION RESULTS

File Type	Yes	No	Error
Info-zip	0	1424	0
Zip bombs	2000	0	0

For the first set involving info-zip, we prepare 1424 info-zip containing different files with different sizes. These compressed files include plain text files and the files that have their own logical structures like *.doc*. For this second set of bombs, we use the generation tool which is offered by the bomb designer [8] to make 2000 different bombs. This set contains both types of non-recursive zip bombs. The detecting results are listed in Table 4. This algorithm has extremely high detection accuracy.

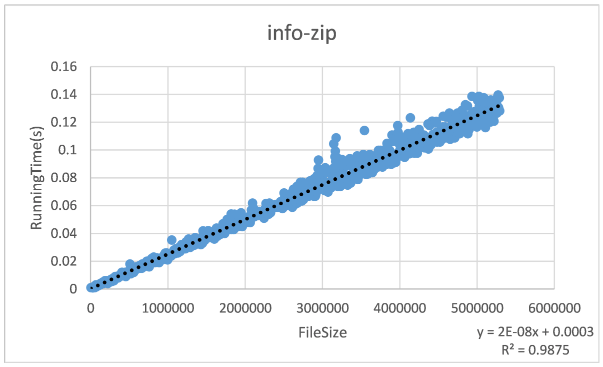


Fig. 6. The relationship between the detection time and the size of info-zip

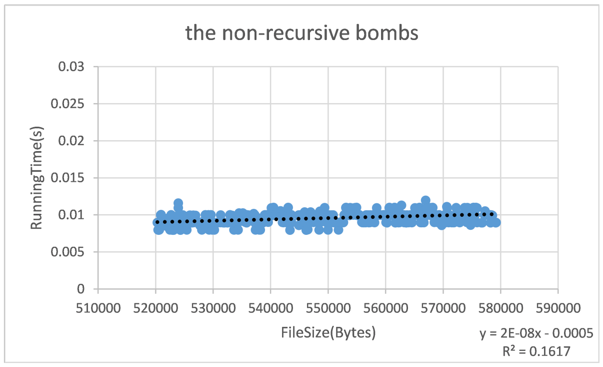


Fig. 7. The relationship between the detection time and the size of the non-recursive zip bombs

C. Algorithm Efficiency

For info-zip, this algorithm will inevitably search the entire file to confirm that all local file entries do not overlap and for non-recursive zip bombs, no matter which of the two structures the bomb has, this algorithm can determine that this is a bomb immediately after detecting the last two file headers. Therefore, for info-zip, the detection time grows linearly as the zip file size increases, as in Figure 6. For non-recursive zip bombs, the figure appears as a horizontal line which demonstrates that there is almost no connection between the detection time and the size of the zip file, as in Figure 7.

V. CONCLUSION

In this paper, we introduce the structures of the non-recursive zip bomb and design an algorithm for detecting such a zip bomb. At the same time, we list some details that should be noticed in the detection and the algorithm efficiency about non-recursive zip bombs and info-zip is given.

Most decompression software can use this algorithm as a reference to make corresponding patches. Since this algorithm does not rely on decompression software, for security software, it can be used to quickly detect whether a zip file in the emails or removable media storage devices, such as USB or disks is a non-recursive zip bomb.

In today's world of big data, more and more compressed files are transmitted on the Internet or uploaded to cloud servers. Because of the lack of detection methods for non-recursive zip bombs, it is likely that bombs will be uploaded to cloud servers or downloaded to personal computers. Once such bombs are decompressed, there will be serious consequences. Therefore, this bomb detection algorithm we propose is very useful.

In real life situations, an attacker may not use a standard structure bomb. The attacker can make a non-recursive bomb whose structure is different from the previously mentioned structures. However, as long as the bombs use overlapping structures, they will be detected by our algorithm.

We do not take zip64 and encrypted zip files into consideration, so this algorithm may not have such good compatibility with these zip files. There are some structural differences between these two types of zip files and info-zip. For example, Zip64 has some extra unique structures (such as Zip64 end of central directory locator); encrypted zip files have some additional structures to store encrypted information. These are two directions worth analyzing in the algorithm improvement in the future.

ACKNOWLEDGMENTS

We thank Yang Xiong, JiPeng Wang, DeGang Chen, WenQi Liu, JiaYu Liu, XingYu Liao for their assistance and David Fifield for his guidance on the use of non-recursive zip bomb generating tool.

REFERENCES

[1] D. Fifield, “A Better Zip Bomb,” WOOT @ USENIX Security Symposium, August 2019.

[2] R. Cox, “Zip Files All the Way Down,” unpublished.

[3] W. T. Mambodza, R. T. Shoniwa, V. Shenbagaraman , “Cybercrime in Credit Card Systems,”, International Journal of Science and Research (IJSR), 2014.

[4] H. Sowmya, “A Study on Zip Bomb.” unpublished.

[5] “Python3.7,” https://docs.python.org/3.7/, May 2020.

[6] P. Inc, “Appnote.txt - .Zip File Format Specification,” https://pkware.cachefly.net/webdocs/APPNOTE/APPNOTE-6.2.0.txt, May 2020.

[7] “42.zip,” https://www.unforgettable.dk/, March 2020.

[8] D. Fifield, “Zipbomb-20190822.zip,” https://www.bamsoftware.com/hacks/zipbomb/zipbomb-20190822.zip, March 2020.

[9] M. Adle, “Fork of Infozip Unzip 6.0 For New Zip Bomb Detection Patch,” https://github.com/madler/unzip/commit/47b3ceae397d21bf822bc2a c73052a4b1daf8e1c, March 2020.