Having fun with ZIP

Armijn Hemel, MSc - 2024-05-21

One slide summary

- ZIP specifications are sometimes unclear and ambigious
- not every implementation implements the specs the same
- input sanitation is a problem

Why do I care about ZIP?

- I am making a firmware analysis tool: Binary Analysis Next Generation (BANG): https://github.com/armijnhemel/binaryanalysis-ng/
- many firmware files are delivered as ZIP files
- some of these firmware files cannot be correctly unpacked by standard tools, or useful data is not unpacked:
  - vendors change standard ZIP structures (Dahua)
  - vendors add extra data in non-standard places (Google)
  - etc.

ZIP file history

- the ZIP file format was invented in the late 1980s as a result of the SEA vs PKWARE lawsuit
- highly recommended background material: “BBS: The Documentary” (episode 8)
- specifications allow reimplementations (but it is not an actual open standard) and are updated every couple of years

ZIP use

- many file formats use or are based on ZIP.
- Some examples:
  - Microsoft Office + LibreOffice files
  - Java archives (JAR, WAR, EAR, ...)
  - Android package archives (APK)

ZIP implementations on *nix

- there are plenty of (independent) ZIP (re)implementations on Linux:
  - unzip
  - p7zip (originally for 7z, but can also unpack ZIP and other compression formats)
  - minizip/minizip-ng
  - Python/Java/C# libraries
  - BANG (wraps around Python libraries, with additional parsing with Kaitai Struct)
  - and more

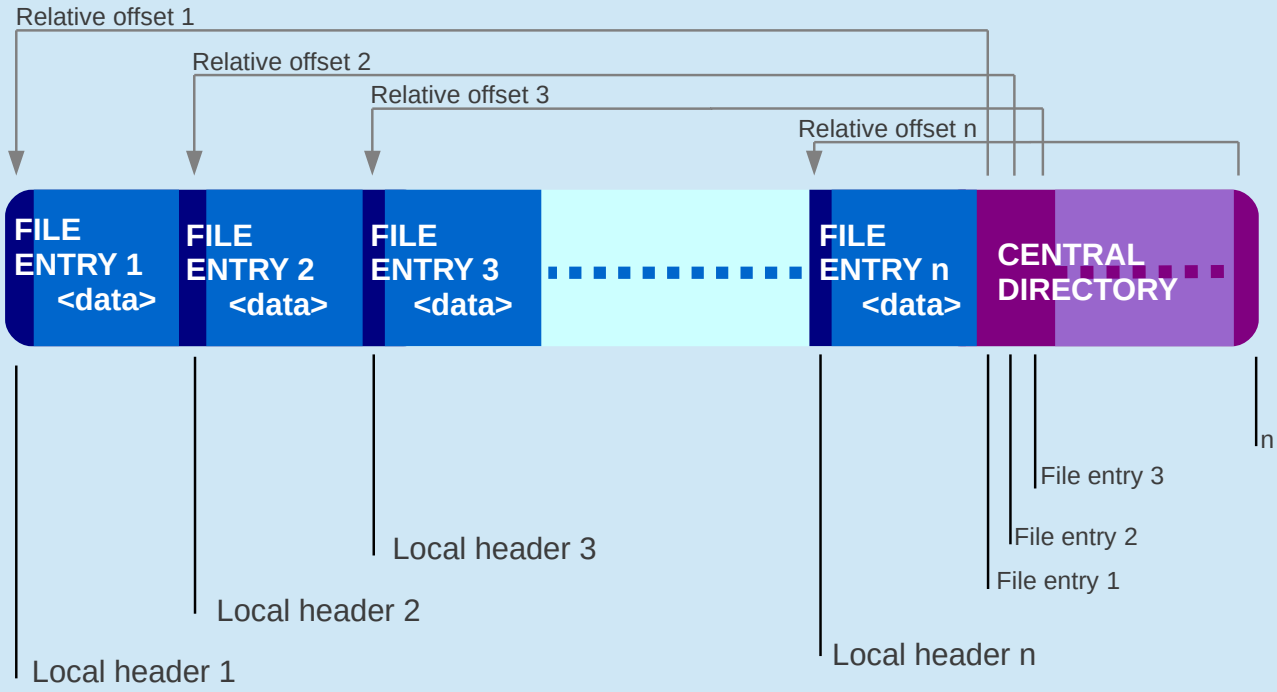
ZIP file format (1)

- ZIP is not a compression format. Instead, it is an archive format:
  - individual files in archive can be compressed with different compression algorithms
  - a ZIP archive is a concatenation of files, with a lookup table (called “central directory”) at the end of the archive providing quick access to files using offsets (with some additional metadata here and there)

ZIP file format (2)

- files (including directories) are stored together with a “local file header”
- the local file header contains metadata and is followed by data (except directories or empty files), possibly encrypted
- (optionally) data is followed by extra metadata (“data descriptors”), for example in ZIP files created using streaming
- The central directory has a corresponding entry for each file, with (nearly) the same data metadata as the local file header and data descriptor.

ZIP file format (3)



Local file header

- the local file header contains metadata, such as:
  - file name length + file name
  - compressed + uncompressed size (could be -1 for example when streaming)
  - CRC32
  - type of compression (store, deflate, bzip2, LZMA, XZ, zstd, etc.)
  - various bit flags
  - extra fields with custom data

Central directory

- the central directory contains almost the same information as the local file directory, with slight differences:
  - possibly different compressed/uncompressed size value (think: streaming)
- and then some more information not in the local file header. Most interesting:
  - file comment

More ZIP headers

- the ZIP documentation specifies more ZIP headers, used for different kinds of metadata:
  - end of central directory header - header indicating the end of the archive: contains a pointer to the start of the central directory, plus optional archive comment
  - data descriptor - header following the data, indicating the actual size of the data. This is used in streaming data.
  - ZIP64 related headers

Typical workflow for unpacking

1. open the file
2. search the data for the “end of central directory” header (EOCD)
3. find the offset to the “central directory” from the EOCD
4. extract metadata for each file from the central directory, including offsets and size
5. extract individual files from the archive

Quiz: unpacking concatenated files

- Imagine two ZIP files A and B concatenated together creating A+B
- when unpacking A+B, which ZIP file is unpacked:
  - A
  - B
  - A and B
  - none
  - yellow

Answer: it depends on the tool!

- unzip, Python ZIP library and some others unpack B
- p7zip unpacks A
- BANG unpacks A and B

Why does this happen?

- most tools seek to the end of the file and then search backwards for a valid EOCD and finds B, completely missing A. Not good.
- p7zip (probably) searches forward to the first (corresponding) EOCD and finds A and ignores any trailing data. Much better.
- BANG searches forward to first find A, then continues to find B. Probably best.

Implications

- A+B is not a valid ZIP file according to the specifications but many programs will still process it (in some way).
- prepended data (in front of a valid ZIP) has been used in malware delivery
- What else could go wrong?

Deep dive into the ZIP specification

- a deep dive into the ZIP specification revealed several unclarities related to:
  - deprecation of features (how?)
  - names & paths
  - which data to use in case of duplicates (remember: local file header and central directory have duplicated fields)

Deprecation of features (1)

- ZIP defines a “version needed to extract” that ZIP programs can use as a check to see if the functionality is supported.
- this is a per file field, not per archive!
- specifications:
  - This value is based on the specific format features a ZIP program MUST support to be able to extract the file. If multiple features are applied to a file, the minimum version MUST be set to the feature having the highest value. New features or feature changes affecting the published format specification will be implemented using higher version numbers than the last published value to avoid conflict.

Deprecation of features (2)

- the functionalities and minimum versions for “version needed to extract” are specified in section 4.4.3.2
- most implementations don't support all features but cherrypick. This makes it useless (p7zip therefore ignores it)
- incomplete: one new feature (zstd compression) hasn't even been given a number!

Directory names (1)

- I have files where directory names have no / at the end and unzip refuses to unpack the file
- The specification says:

  > 4.4.17.1 The name of the file, with optional relative path. The path stored MUST NOT contain a drive or device letter, or a leading slash.

- Question: should directories stored always end with a / ? Is that the “optional relative path”?

Directory names (2)

- comment from unzip:

  /* Add trailing / to the directory name */

- .NET, Python and likely others also require directories to always have a /
- p7zip doesn't care

File names

- the specifications don't mention anything about duplicate file names
- how these are processed is implementation specific

Local file header vs central directory

- a lot of data is duplicated between the local file header and the central directory. The specifications aren't clear which data should be leading.
- most implementations choose to completely ignore the local file directory (and sometimes also the central directory).

Implementations ignore specs

- comments are not properly implemented in every implementation
- specification says:
  - D.1 The ZIP format has historically supported only the original IBM PC character encoding set, commonly referred to as IBM Code Page 437.
  - D.2 If general purpose bit 11 is unset, the file name and comment SHOULD conform to the original ZIP character encoding.
- Python: “The comment associated with the ZIP file as a bytes object.” and “Comment for the individual archive member as a bytes object.”

Hmmm.

- the specifications are unclear and implementations are not reading them in the same way
- there must be opportunities for malice!
- Let's play!

Experiment: hiding malware in ZIP files

- as an experiment a student from OS3.nl (Nick Mastorakis) and I worked on fooling malware scanners with modified ZIP files:
  - “hide” malware in ZIP files
  - upload to VirusTotal to scan with 60+ malware scanners
- 13 scenarios, 2 malware samples were used of different size: a regular one and a small one to hide in ZIP fields

VirusTotal

- VirusTotal is an online service where files can be scanned by multiple malware scanners
- drawbacks:
  - results aren't always reproducible due to updates to VirusTotal
  - VirusTotal might sometimes not show all results to thwart malware developers wanting to proof their malware
- Still, there were some interesting results!

Caveats

- malware scanners should not be used as the only line of defense
- there are many other mechanisms to get malware onto a machine
- getting malware onto a machine is just one piece of the malware puzzle: unpacking and executing is the other

Scenario 1: baseline

- uncompressed malware (Agent Tesla variant)
  - recognized by 59 out of 71 scanners
- some scanners were not successful as the malware itself wasn't recognized. Logical conclusion: 59 scanners is the upper bound (for reasons outlined before this wasn't always the case)

Scenario 2: regular ZIP

- compress the file using a regular ZIP compression program (“zip” on Linux) in the simplest format possible.
- 58 out of 62 scanners successfully found the malware
- note: for each scan a different number of scanners was reported. This makes comparing results sometimes a bit tricky.

Scenario 3: use different ZIP programs

- same as previous, but use p7zip and minizip-ng for compression
- 52 out of 64 scanners found the malware
- note: slightly fewer scanners found it but not very significant

Scenario 4: use different compression

- bzip2: 45 out of 65
- LZMA: 41 out of 63
- zstd: 5 out of 61
- interesting! zstd is a recent addition to the ZIP standard, but very few malware scanners can unpack files compressed with it
- results were similar when using different ZIP implementations

Scenario 5: change the “version to extract” field

- the “version to extract” field is present in both the local file header and central directory
- changing that to invalid versions might cause scanners to not unpack (see earlier)
- but this is not the case: 51 out of 65.

Scenario 6: concatenating ZIP and malware

- two subscenarios:
  - malware + ZIP: 53 out of 71
  - ZIP + malware: 4 out of 62
- This is interesting. It seems most scanners ignore trailing data?

Scenario 7: concatenate two ZIP files

- two subscenarios:
  - harmless ZIP + malware ZIP: 39 out of 64
  - malware ZIP + harmless ZIP: 33 out of 64
- some scanners would only scan the first file, others only the second file. Some scanners didn't find anything, others found both.

Scenario 8: use comments to hide data

- for this scenario a different smaller malware was used that would fit in 65,536 bytes
- two subscenarios:
  - use the archive .ZIP file comment (end of central directory): 13 out of 62
  - use a file comment (central directory): 1 out of 61
- file comments seem to be ignored by most scanners!

Scenario 9: use a custom “extra field”

- a custom “extra field” was created, containing the malware. 4 bytes are used for the “extra field header” so that leaves 65,532 bytes for malware. This was done for both the local file header and central directory.
- shocking: 0 out of 61 were successful
- no scanners scan data in “extra field” fields!

Scenario 10: store malware between two local files

- put malware in between two local file headers, adapt the central directory so that the file can still be unpacked regularly
- basically: this is similar to Google does with Android Signing Blocks in APK files
- 2 out of 61
- many scanners do not seem to scan data between the individual files

Scenario 11: hide files

- Specifications say:

  > 4.3.2 Each file placed into a ZIP file MUST be preceded by a "local file header" record for that file. Each "local file header" MUST be accompanied by a corresponding "central directory header" record within the central directory section of the ZIP file.

- create a ZIP file with 3 files, file nr 2 being malware and remove the entry for nr 2 from the central directory (this is similar to what a format called “Search Optimized ZIP” does)
- 8 out of 60
- it seems that most scanners treat the central directory as complete.
- result is interesting compared to scenario 10: adding a local file header makes more scanners find the malware?

Scenario 12: hide malware in the name field

- the ZIP specification allows for absurd long names (65,536 bytes) which most file systems don't like
- construct a file name as follows:
  - regular file name + NULL byte + malware
- 0 out of 59 scanners
- the file name field isn't scanned by malware scanners

Scenario 13: change compression method

- changing the compression method in the local file header and/or central directory, without actually changing the compression.
- first create a ZIP file with deflate compression, then:
  - change compression method to zstd in local file header: 19 out of 63
  - change compression method to zstd in central directory: 2 out of 61
- interesting: possibly some scanners look at the local file header and bail out early?

Possible scenarios

- digital signature field
- files created using a streaming approach
- ZIP64 extensible data sector (might allow much bigger malware to be stored)
- very old, mostly unsupported, compression (shrunk, reduced, implode)
- strong encryption fields
- using multiple fields to store parts of malware

We need a better ZIP implementation

- ZIP isn't a great file format but it isn't going away.
- for forensics/malware research we really need better tools that will allow:
  - easy access to every piece of data in a ZIP file
  - linting of ZIP files
  - proper visualisation
- proposals for funding to create better tools is currently under review

Wrapping up + thank you

- catch me in the hallway to ask me more!
- big thanks to NLnet Foundation for funding my journey deep deep into the land of ZIP
- try BANG! https://github.com/armijnhemel/binaryanalysis-ng/