Hiding in the Familiar: Steganography and Vulnerabilities in Popular Archives Formats.| NyxEngine

nyx.reversinglabs.com

REVERSING LABS | Reverse Engineering & Software Protection

Contents

Introduction to NyxEngine

Steganography[1] is the art and science of writing hidden messages in such a way that no one, apart from the sender and intended recipient, suspects the existence of the message, a form of security through obscurity. When it comes to digital steganography no stone should be left unturned in the search for viable hidden data. Although digital steganography is commonly used to hide data inside multimedia files, a similar approach can be used to hide data in archives as well. Steganography imposes the following data hiding rule: *Data must be hidden in such a fashion that the user has no clue about the hidden message or file's existence*. This can be achieved by either hiding existing packed content from all programs designed to unpack the selected file format, or adding new data to existing compressed files, so that the file's usability is unchanged. To discover this hidden information we must go into deep analysis of systems that have developed their own archive processors and see the implications of format specifications being interpreted differently across such solutions.

We have designed *NyxEngine* to ensure that no byte is left unchecked in the search for interesting archive data. Furthermore Nyx performs detailed data inspection by which it identifies possible vulnerabilities and corruptions in the binary content of archives. By integrating the *NyxEngine* as the top layer in archive processing, we can successfully detect and prevent all known and future vulnerability attack vectors against archive processors, thus effectively eliminating the possibility of archive bombs and other exploits. In addition to shielding against exploits, Nyx also searches for viable hidden data that was intentionally cloaked from sight using steganographic principles. And since the engine does detailed data inspection, it can correct vulnerabilities and recover files, making it a perfect archive preprocessor.

Nyx engine's exploit shield functionality checks the following archive areas: stored file name length and content, compression ratio, extract algorithm requirements, checksum tampering, multi-disk tampering, file entry duplication and other miscellaneous header data checks. Serving as a common denominator among all known archive processing solutions, Nyx classifies each instance of tampering in a functional group as vulnerabilities that affects that group.

By performing detailed checks and on-the-fly corrections, the maximum possible archive data is recovered and identified. This is the best way to find files that are present in the archive, but unreported in the archive header and to extract every possible bit from the archive. This method this works not only with unreported files, but with any kind of binary data present in the archive which isn't assigned to any of the file content.

The detailed file analysis provided by Nyx makes it possible to recover the maximum amount of damaged, corrupt and invalid data.

---

1 *"Steganography is the art and science of writing hidden messages in such a way that no one, apart from the sender and intended recipient, suspects the existence of the message, a form of security through obscurity. The word steganography is of Greek origin and means concealed writing."*

Introduction to ZIP file format

The ZIP file format is one of the most common archive file formats used today. The format was originally created in 1986 by Phil Katz for PKZIP, and evolved from the previous ARC compression format by Thom Henderson. The PKZIP format is now supported by many software utilities other than PKZIP. Microsoft has included built-in ZIP support (under the name "compressed folders") in versions of its Windows operating system since 1998. Apple has included built-in ZIP support in Mac OS X 10.3 and later.

ZIP is a simple archive format that compresses every file separately. Compressing files separately allows individual files to be retrieved without reading through other data; in theory, it may allow better compression by using different algorithms for different files. A caveat to this is that archives containing a large number of small files end up significantly larger than if they were compressed as a single file, due to the fact that the data structures which store information on each individual file are stored uncompressed.

The ZIP file's contents comprise files and directories stored in arbitrary order. The files and directories are represented by file entries. The location of each file is indicated in a “central directory”, located at the end of the ZIP file.

Each file entry is introduced by a local header with information about the file such as the comment, file size and file name, followed by optional "extra" data fields, and then the possibly compressed, possibly encrypted file data. The "Extra" data fields are the key to the extensibility of the ZIP format. It is the "extra" fields that are exploited to support ZIP64 formats, WinZip-compatible AES encryption, and NTFS file timestamps. In theory there are many other extensions possible via this coded "extra" field.

The central directory consists of file headers holding, among other metadata, the file names and the relative offset in the archive of the local headers for each file entry. Each file entry is marked by a specific 4-byte "signature"; each entry in the central directory is likewise marked with a different particular 4-byte signature. ZIP file parsers typically look for the appropriate signatures when parsing a ZIP file. Due to the fact that the order of the file entries in the directory need not conform to the order of file entries in the archive, the format is non-sequential. There is no BOF or EOF marker in the ZIP spec. Instead, ZIP tools scan for the signatures of the various fields.

There are numerous ZIP tools available, and numerous ZIP libraries for various programming environments. Some of the libraries are commercial, some are not. Some are open source, some are not. WinZip is perhaps the most popular and famous ZIP tool - it runs primarily on Windows and is a user tool for creating or extracting ZIP files. WinRAR, IZarc, Info-zip, 7-zip are other tools, available on various platforms. Some of those tools have library or programmatic interfaces.

If we take a look at the ZIP central directory structure we can see the following:

```
typedef struct NYX_ZIP_CENTRALDIR{
      DWORD ZIPSignature;
      WORD ZIPVersion;
      WORD ZIPExtractVersion;
      WORD ZIPGeneralPurpose;
      WORD ZIPCompressionMethod;
      WORD ZIPLastFileModTime;
      WORD ZIPLastFileModDate;
      DWORD CRC32;
      DWORD FileCompressedSize;
      DWORD FileUncompressedSize;
      WORD FileNameStringLength;
      WORD ExtraDataLength;
      WORD FileCommentLength;
      WORD DiskNumberStart;
      WORD InternalFileAttributes;
      DWORD ExternalFileAttributes;
      DWORD RelativeOffsetOfLocalHeader;
      //Following this is data with variable size
}NYX_ZIP_CENTRALDIR, *PNYX_ZIP_CENTRALDIR;
```

First thing that comes to mind in order to achieve existing data steganography is setting the value of the field *FileNameStringLength* to zero. But that can't work because file name is written just after the central directory entry, and if we reset its length to zero the archive will become corrupt, since the next item of the central directory won't be located correctly. This is what one central directory entry looks like:

```
00026BA0  AD D6 9C D2 5E C4 71 24  BF 08 12 30 41 8A 50 4B  –Ö▮Ò^Äq$¿..0A▮PK
00026BB0  01 02 14 00 14 00 02 00  08 00 21 6C 9D 3A DD 1B  ..........!l.:Ý.
00026BC0  FB A4 85 6B 02 00 B3 91  02 00 0B 00 00 00 00 00  û¤▮k..³'........
00026BD0  00 00 00 00 20 00 00 00  00 00 00 00 6D 61 6C 77  .... .......malw
00026BE0  61 72 65 2E 62 69 6E 50  4B 05 06 00 00 00 00 01  are.binPK.......
00026BF0  00 01 00 39 00 00 00 AE  6B 02 00 00 00           ...9...®k....
```

Since the checksum check only ensures integrity of the compressed data, but not the header we can easily modify any member of the central directory entry structure. To make **malware.bin** vanish from the list of all programs that work with the ZIP file format we simply modify the file name. By changing the first character of the name (the byte 0x6D which is the letter 'm') to 0x00, we effectively hide this file from being listed by almost all programs that work with the ZIP file format. Only two programs are not fooled by this trick: WinZIP and 7Zip. Those two are the only programs that, regardless of the invalid name, show the file as an entry in their view. But how can changing the name work? Can it be reverted by some steganography program later? The answer is yes, because ZIP stores file names at two locations, once at the central directory entry and once at the local directory entry - and since only one of the two is modified, the other one can be used to revert the file to original state.

Even though this approach offers limited hiding capabilities, there are other more effective ways of hiding data in archives. One such way would be exploiting ZIP file format specifics to make data unreadable by archivers but leaving it present and its state easily reversible to the original. One such way would be utilization of extra data fields described in the PKWARE ZIP file format specification.

This field was introduced because of the need to store extra information about the file such as NTFS data streams, encryption information and other data utilized by applications that process this format. There are quite a few documented uses of this field in the standard itself. However due to the freedom offered by such data a structure, there are multiple ways to use it. For data hiding, you need only expand the extra field of one file to consume one or more of the files that follow it in the archive header. After this, correcting the fields ThisDiskItemEntries and DiskItemEntries ensures that the archive remains valid. Such archive modification would look like this:

```
0002FED0  08 12 30 41 8A 50 4B 01  02 14 00 14 00 00 00 08  ..0A▌PK.........
0002FEE0  00 13 6B 2E 3C B7 C2 61  FA F4 92 00 00 5C 95 00  ..k.<·Âaúô'..\▌.
0002FEF0  00 0A 00 39 00 00 00 00  00 00 00 20 00 00 00 00  ...9.......  ....
0002FF00  00 00 00 64 6F 6E 6B 65  79 2E 6A 70 67 50 4B 01  ...donkey.jpgPK.
0002FF10  02 14 00 14 00 00 00 08  00 21 74 9D 3A DD 1B FB  .........!t.:Ý.û
0002FF20  A4 90 6B 02 00 B3 91 02  00 0B 00 00 00 00 00 00  ¤.k..³'.........
0002FF30  00 00 00 20 00 00 00 1C  93 00 00 6D 61 6C 77 61  ... ....▌..malwa
0002FF40  72 65 2E 62 69 6E 50 4B  05 06 00 00 00 00 01 00  re.binPK........
0002FF50  01 00 71 00 00 00 D5 FE  02 00 00 00              ..q...Õþ....
```

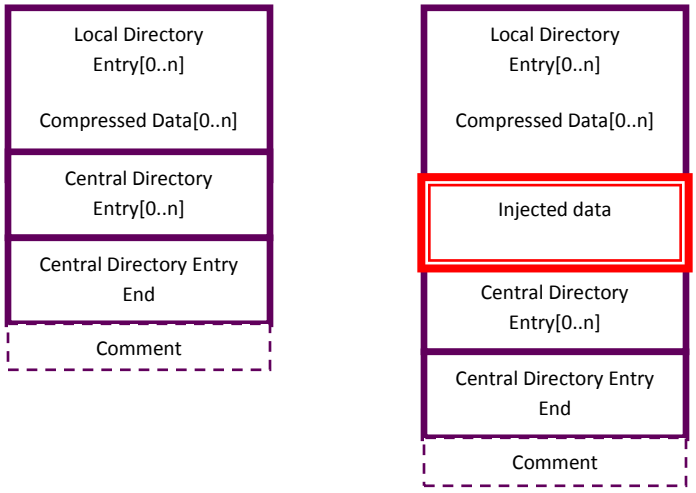
There are numerous ways one can hide files in a ZIP archive by modifying the archive headers. The next approach is even more elegant, as it involves a very small change to the header. But this time we are not talking about the central directory itself, but about the end of the central directory, which is the actual beginning of the archive, despite its position at the end of the file. Due to this positioning, you process a ZIP archive by searching for the signature pattern from the end of the archive. Here is the archive's signature structure:

```
typedef struct NYX_ZIP_CENTRALDIR_END{
      DWORD Signature;
      WORD DiskID;
      WORD ThisDiskID;
      WORD ThisDiskItemEntries;
      WORD DiskItemEntries;
      DWORD SizeOfCentralDir;
      DWORD LocationOfCentralDir;
      WORD ZipCommentLength;
}NYX_ZIP_CENTRALDIR_END, *PNYX_ZIP_CENTRALDIR_END;
```

To hide files, you simply change the LocationOfCentralDir pointer to point to the first file you want to be visible in the archive. After you do this, correcting the fields ThisDiskItemEntries and DiskItemEntries ensures that the archive remains valid. This procedure hides all files located prior to the file to which the modified LocationOfCentralDir points.

Reverting this is easy. One only needs to search backwards from LocationOfCentralDir for a central directory entry signature to find out how many files were hidden. And once those are found, correcting the fields ThisDiskItemEntries and DiskItemEntries restores the file to its original state.

The internal data structure of ZIP archives is similar to that the structure the file system uses to store data on drives. Here is the layout of files inside the archive:



As we see from the figure above, files form an array of local directory entry structures followed by compressed data assigned to that local directory entry. Since the central directory entry end structure contains the pointer to the first central directory entry, it can be moved in any direction. By moving it up in a file we create space for data to be injected. One side effect of injecting data this way is that once new files are added to such an archive with an archiver application, injected data is automatically stripped during the unarchiving process. This can be useful in some scenarios since it leads to protecting data from tampering.

One scenario for this kind of data protection would be packing random files inside a ZIP archive with a simple piece of malware that any antivirus detects. This way hidden data will be stripped by the antivirus itself, if the file is scanned for malware, which is useful if the file ends up in the wrong hands, because the hidden data will self destruct. A similar self-destruct approach can be used for OOXML file format, which is essentially a ZIP archive containing Office document data. The difference is that the Microsoft Office product line detects changes to the file and offers to automatically correct damage to the file, which if employed erases the hidden data.

This brief introduction to archive steganography shows that hiding data in archives is possible in numerous ways. Furthermore the hidden data quite often protects itself with unique self-destruct mechanisms, which is why we need to be careful when inspecting suspicious content if we want to truly find all viable data.

Steganography and file malformation security impacts

Multiple ways of introducing steganography to archive file formats impact computer system security. The effects of hiding files can go beyond steganography implications, creating a serious threat to archive parsing. Due to complexity of the process for creating archive software, and the inevitable ambiguity in the related documentation, it is not entirely uncommon for archive software to contain oversights that allow seemingly invalid files to be processed normally by some solutions. This is crucial for antivirus vendors to understand – since they want to support extraction of as many popular archive formats as possible. These slight differences in format specification make all the difference in overall security within a network, because if there is a single archive processing tool available that can decompress the archive within your network, its content must be fully inspected to make sure that that file is malware clean. Archive content can be scanned t the point of extraction, but your network will be much safer if scanning is done at the gateway so that the fewest possible malicious files reach their destination. This is particularly crucial for business organizations that only employ gateway scanners to protect their endpoints.

ReversingLabs Corporation has performed a series of in-depth checks to test the overall impact of archive file format malformations on security software. During our test, we intentionally malformed selected archive formats (ZIP, RAR, CAB, GZIP and 7ZIP) to the extent permitted by file format documentation, leaving each in such a state that packed content within it was considered valid and extractable by at least one popular format processing program. After malformation the files were subjected to multiple antivirus testing systems (available at www.virustotal.com). The results of our tests were used to create series of vulnerability security advisories available at the ReversingLabs website. Prior publishing them, Reversing Labs, in association with CERT-FI, contacted all affected security vendors, and helped those who responded to verify the implemented fixes. The affected vendor responses about the issues can be found in the associated individual vulnerability advisories.

References and tools

[1] NyxEngine 1.0 SDK – **www.reversinglabs.com**

[2] Archive vulnerability security advisories – **www.reversinglabs.com**

[3] Slides & Whitepaper: BlackHat, Barcelona 2010 – **nyx.reversinglabs.com**

[4] ZIP steganography solution – **http://www.codeproject.com/KB/security/steganodotnet16.aspx**

[5] ZIP steganography solution – **http://www.sfu.ca/%7Evwchu/zjmask.html**