18:04 Concealing ZIP Files in NES Cartridges

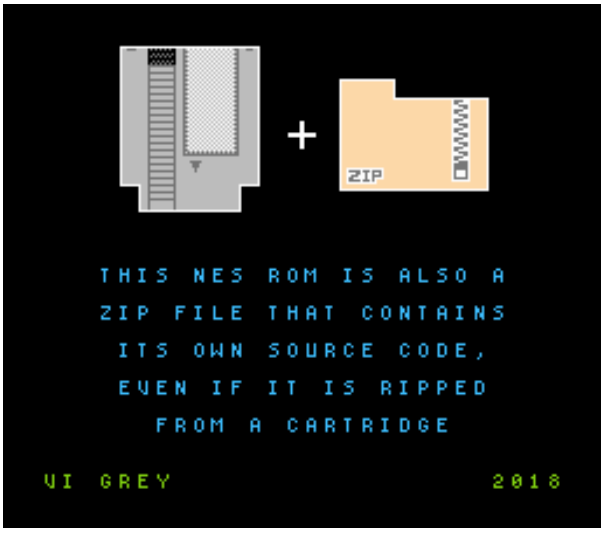
by Vi Grey

Hello, neighbors.

This story begins with the fantastic work described in PoC||GTFO 14:12, which presented an NES ROM that was also a PDF. That file, `pocorgtfo14.pdf`, was by coincidence also a ZIP file. That issue inspired me to learn 6502 Assembly, develop an NES game from scratch, and burn it onto a physical cartridge for the `#tymkrs`.

During development, I noticed that the unused game space was just being used as padding and that any data could be placed in that padding. Although I ended up using that space for something else in the game, I realized that I could use padding space to make an NES ROM that is also a ZIP file. This polyglot file wouldn't make the NES ROM any bigger than it originally was. I quickly got to work on this idea.

The method described in this article to create an NES + ZIP polyglot file is different from that which was used in PoC||GTFO 14:12. In that method, none of the ZIP file data is saved inside the NES ROM itself. My method is able to retain the ZIP file data, even when it burned onto a cartridge. If you rip the data off of a cartridge, the resulting NES ROM file will still be an NES + ZIP polyglot file.



Numbers and ranges included in figures in this article will be in Hexadecimal. Range values are big-endian and ranges work the same as Python slices, where `[`x`:`y`]` is the range of x to, but not including, y.

iNES File Format

This article focuses on the iNES file format. This is because, as was described in PoC||GTFO 14:12, iNES is essentially the *de facto* standard for NES ROM files. Figure 8 shows the structure of an NES ROM in the iNES file format that fits on an NROM-128 cartridge.[10]

The first sixteen bytes of the file MUST be the iNES Header, which provides information for NES Emulators to figure out how to play the ROM.

Following the iNES Header is the 16 KiB PRG ROM. If the PRG ROM data doesn't fill up that entire 16 KiB, then the PRG ROM will be padded. As long as the PRG padding isn't actually being used, it can be any byte value, as that data is completely ignored. The final six bytes of the PRG ROM data are the interrupt vectors, which are required.

Eight kilobytes of CHR ROM data follows the PRG ROM.

Start of iNES File

iNES Header	`[0000:0010]`
PRG ROM	`[0010:4010]`
PRG Padding	`[XXxx:400A]`
PRG Interrupt Vectors	`[400A:4010]`
CHR ROM	`[4010:6010]`

Figure 8. iNES File Format

[10] NROM-128 is a board that does not use a mapper and only allows a PRG ROM size of 16 KiB.

ZIP File Format

There are two things in the ZIP file format that we need to focus on to create this polyglot file, the End of Central Directory Record and the Central Directory File Headers.

End of Central Directory Record

To find the data of a ZIP file, a ZIP file extractor should start searching from the back of the file towards the front until it finds the End of Central Directory Record. The parts we care about are shown in Figure 9.

The End of Central Directory Record begins with the four-byte big-endian signature `504B0506`.

Twelve bytes after the end of the signature is the four-byte Central Directory Offset, which states how far from the beginning of the file the start of the Central Directory will be found.

The following two bytes state the ZIP file comment length, which is how many bytes after the ZIP file data the ZIP file comment will be found. Two bytes for the comment length means we have a maximum length value of 65,535 bytes, more than enough space to make our polyglot file.

Start of End of Central Directory Record

End of Central Directory Record Signature (`504B0506`)	`[0000:0004]`
...	`[0004:0010]`
Central Directory Offset	`[0010:0014]`
Comment Length (L)	`[0014:0016]`
ZIP File Comment	`[0016:0016 + L]`

Figure 9. End of Central Directory Record Format

[11] `unzip pocorgtfo18.pdf APPNOTE.TXT`

Central Directory File Headers

For every file or directory that is zipped in the ZIP file, a Central Directory File Header exists. The parts we care about are shown in Figure 10.

Each Central Directory File Header starts with the four-byte big-endian signature `504B0102`.

38 bytes after the signature is a four-byte Local Header Offset, which specifies how far from the beginning of the file the corresponding local header is.

Start of a Central Directory File Header

Central Directory File Header Signature (`504B0102`)	`[0000:0004]`
...	`[0004:002A]`
Local Header Offset	`[002A:002E]`
...	`[002E:]`

Figure 10. Central Directory File Header Format

33 - MSEC BUFFER BY DDI STORES UP TO 66,000 BITS FOR DISPLAY APPLICATIONS

Less than 2¢ per bit is the cost of data storage in a 33-msec, 2-mc delay line buffer offered by Digital Devices, Inc., primarily for 30-frame-per-second refresh rate display applications. Card on front interfaces buffer electronics with MECL, DTL, RLT, TTL and other micrologic.

Miscellaneous ZIP File Fun

Five bytes into each Central Directory File Header is a byte that determines which Host OS the file attributes are compatible for.

The document, "APPNOTE.TXT - .ZIP File Format Specification" by PKWARE, Inc., specifies what Host OS goes with which decimal byte value.[11] I included a list of hex byte values for each Host OS below.

```
00 - MS-DOS and OS/2
01 - Amiga
02 - OpenVMS
03 - UNIX
04 - VM/CMS
05 - Atari ST
06 - OS/2 H.P.F.S.
07 - Macintosh
08 - Z-System
09 - CP/M
0A - Windows NTFS
0B - MVS (OS/390 - Z/OS)
0C - VSE
0D - Acorn Risc
0E - VFAT
0F - Alternate MVS
10 - BeOS
11 - Tandem
12 - OS/400
13 - OS/X (Darwin)
(14-FF) - Unused
```

Although `0A` is specified for Windows NTFS and `0B` is specified for MVS (OS/390 - Z/OS), I kept getting the Host OS value of TOPS-20 when I used `0A` and NTFS when I used `0B`.

I ended up deciding to set the Host OS for all of the Central Directory File Headers to Atari ST. With that said, I have tested every Host OS value from `00` to `FF` on this file and it extracted properly for every value. Different Host OS values may produce different read, write, and execute values for the extracted files and directories.

Start of iNES + ZIP Polyglot File

iNES Header	`[0000:0010]`
PRG ROM	`[0010:4010]`
PRG Padding	`[XXxx:YYyy]`
ZIP File Data	`[YYyy:400A]`
Comment Length (`0602`)	`[4008:400A]`
PRG Interrupt Vectors	`[400A:4010]`
CHR ROM	`[4010:6010]`

Figure 11. iNES + ZIP Polyglot File Format

iNES + ZIP File Format

With this information about iNES files and ZIP files, we can now create an iNES + ZIP polyglot file, as shown in Figure 11.

Here, the first sixteen bytes of the file continue to be the same iNES header as before.

The PRG ROM still starts in the same location. Somewhere in the PRG Padding an amount of bytes equal to the length of the ZIP file data is replaced with the ZIP file data. The ZIP file data starts at hex offset `YYyy` and ends right before the PRG Interrupt Vectors. This ZIP file data MUST be smaller than or equal to the size of the PRG Padding to make this polyglot file.

Local Header Offsets and the Central Directory Offset of the ZIP file data are updated by adding the little-endian hex value `yyYY` to them and the ZIP file comment length is set to the little-endian hex value `0602` (8,198 in Decimal), which is the length of the PRG Interrupt Vectors plus the CHR ROM (8 KiB).

PRG Interrupt Vectors and CHR ROM data remain unmodified, so they are still the same as before.

Because the iNES header is the same, the PRG and CHR ROM are still the correct size, and none of the required PRG ROM data or any of the CHR ROM data were modified, this file is still a completely standard NES ROM. The NES ROM file does not change in size, so there is no extra "garbage data" outside of the NES ROM file as far as NES emulators are concerned.

With the ZIP file offsets being updated and all

[12]The only ZIP file extractor I have gotten any warnings from with this polyglot file was 7-Zip for Windows specifically, with the warning, "*The archive is open with offset.*" The polyglot file still extracted properly.

data after the ZIP file data being declared as a ZIP file comment, this file is a standard ZIP file that your ZIP file extractor will be able to properly extract.[12]

NES Cartridge

The PRG and CHR ROMs of this polyglot file can be burned onto EPROMs and put on an NROM-128 board to make a completely functioning NES cartridge.

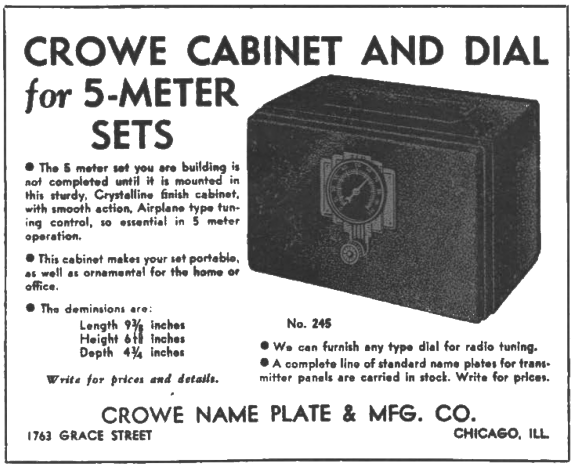
Ripping the NES ROM from the cartridge and turning it back into an iNES file will result in the file being a NES + ZIP polyglot file again. It is therefore possible to sneak a secret ZIP file to someone via a working NES cartridge.

Don't be surprised if that crappy bootleg copy of Tetris I give you is also a ZIP file containing secret documents!

Source Code

This NES + ZIP polyglot file is a quine.[13] Unzip it and the extracted files will be its source code.[14] Compile that source code and you'll create another NES + ZIP polyglot file quine that can then be unzipped to get its source code.

I was able to make this file contain its own source code because the source code itself was quite small and highly compressible in a ZIP file.



[13] `unzip pocorgtfo18.pdf neszip-example.nes`
[14] `unzip neszip-example.nes`