4 This TAR archive is a PDF! (as well as a ZIP, but you are probably used to it by now)

by Ange Albertini

In this article we'll build a TAR/PDF polyglot file with a few simple tools that you already have if you write in TeX or LaTeX (if not, take a couple of days to learn—wouldn't it be just spiffy to submit your very own PoC||GTFO piece in ready-to-go LaTeX?).

4.1 What is a TAR file?

TAR, written in the days when tape drives were the only serious form of backup, stands for TApe aRchive. Not surprisingly, its design is tightly coupled with the mechanics of tape drives. Those drives were made by IBM and were invented for the IBM 650, which was produced in 1953.

Accordingly, in those archives files are stored without compression, lengths and checksums are stored in octal, and everything is 512-byte block based. Respect old age, neighbors—and remember that your own modern technology might not survive that long.

4.2 Abusing the format

A TAR file starts with a fixed-length record of one hundred bytes, where the archived file's original name is stored, padded with zeros.[6] We can abuse this record to store a PDF header and a dummy stream object to cover the rest of the archive.

We'll let `pdflatex` build the dummy stream object for us from a .TeX source. We just need to declare this object (with no compression) right after the `\begin{document}`:

```
\begingroup
  \pdfcompresslevel=0\relax
    \immediate\pdfobj stream
        file {archive.tar}
\endgroup
```

We then need to move the stream content so that it virtually starts at offset 0, fix the file name, and insert a valid `%PDF-1.5` signature.

After the initial hundred byte record, a TAR file contains a header checksum. We need to fix it, because unlike many other checksums, it is actually enforced. The fixing isn't too difficult, but the format is nevertheless rather awkward. Here is the procedure, with a python script to perform it.

1. Overwrite the checksum (at offset `0x94`, 8 bytes long) with spaces.
2. Add all the unsigned bytes of the header.
3. Write this value as octal, with leading zeroes.
4. End the checksum with a NULL character at the 6-byte offset into the field.

```
OFFSET = 0x94
# Wipe the checksum field with spaces.
for i in range(8):
  header[i + OFFSET] = " "

# Sum all bytes of the header to an unsigned int.
c = 0
```

[6] If the name is longer, something called a PaxHeader is used instead; we've come a long way since the 1950s, neighbors!

```
for i in header:
    c += ord(i)

# Store the unsigned int in octal, followed by NULL then space.
for i, j in enumerate(oct(c)):
  header[i + OFFSET] = j

header[OFFSET + 6] = "\0"
# The required space was already there.
```

Now our TAR checksum is valid again, with an archived file name buffer that has been abused to contain a valid PDF header and a stream object. Enjoy!

```
manul:pocorgtfo pastor$ xxd pocorgtfo06.pdf | head -n 21
0000000: 2550 4446 2d31 2e35 000a 25d4 c5d8 0a31  %PDF-1.5..%....1
0000010: 2030 206f 626a 203c 3c0a 2f4c 656e 6774   0 obj <<./Lengt
0000020: 6820 3830 3934 3732 2020 2020 0a3e 3e0a  h 809472    .>>.
0000030: 7374 7265 616d 0a65 0000 0000 0000 0000  stream.e........
0000040: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000050: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000060: 0000 0000 3030 3030 3634 3400 3030 3030  ....0000644.0000
0000070: 3736 3400 3030 3031 3034 3000 3030 3030  764.0001040.0000
0000080: 3030 3030 3030 3000 3132 3431 3435 3637  0000000.12414567
0000090: 3137 3200 3032 3031 3631 0020 3000 0000  172.020161. 0...
00000a0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000b0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000c0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000d0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000e0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000f0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000100: 0075 7374 6172 2020 004d 616e 756c 0000  .ustar  .Manul..
0000110: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000120: 0000 0000 0000 0000 004c 6170 6872 6f61  .........Laphroa
0000130: 6967 0000 0000 0000 0000 0000 0000 0000  ig..............
0000140: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

P.S.: Sadly, that's not all we needed to do. Just when we thought that our polyglot finally worked well on all readers, it turned out that some further edits broke it on `Preview.app`, for no apparent reason, and in a weird way. Namely, `Preview.app` wouldn't display the contant width fonts in our PDF *unless* the PDF signature was placed exactly at offset 0.

Choosing between our Apple readers not being able to enjoy this special issue, having to debug the `Preview.app`, having to reinvent font storage, and missing our deadline, or putting the PDF signature back at offset 0, we chose the latter. With luck, we'll just sacrifice a single 512 byte block and one junk filename to improve our PDF's compatibility.