19:11 Camelus Documentum: A PDF with Two Humps

by Gabriel 'Drup' Radanne

Science is in crisis. The nonsensical editorial model is attacked,[56] the validity of peer review systems is questioned, and, our topic today, the reproducibility of scientific research is put in doubt. As computer science researchers, we gain reproducibility mostly by providing an implementation of the scientific concept that can then be executed: a Proof of Concept, if you will. As a programming language enthusiast, my weapon of choice is OCaml.

To make my research reproducible, I would like to include my PoC directly into my paper, so that reviewers and readers can read and execute my research directly. To achieve this, I'm going to show you how to embed a portable OCaml bytecode executable directly into a PDF article.

Do virtualized camels dream of lambda-expressions?

OCaml is the hipster of programming languages. It's a statically typed programming language with support for both functional and object-oriented paradigms that was created in 1996, long before it was cool. Its main selling point is its sensible and usable design, which is achieved by reaching a compromise between the practicality of Haskell, the safety of C and the speed of Lisp. While OCaml is genuinely an amazing language, it also possess a slightly unusual feature: it can be compiled to either native executable for speed, or to bytecode, which can be executed on a virtual machine. Bytecode is portable,[57] rather lightweight, and reasonably fast.

So, what does OCaml bytecode look like? It's actually a fairly simple file format: a bytecode file is divided into sections. Just like ZIP files, the content starts from the end. The last line of the file should be composed of a magic number that identifies the version of the bytecode, the number of sections, and an index.

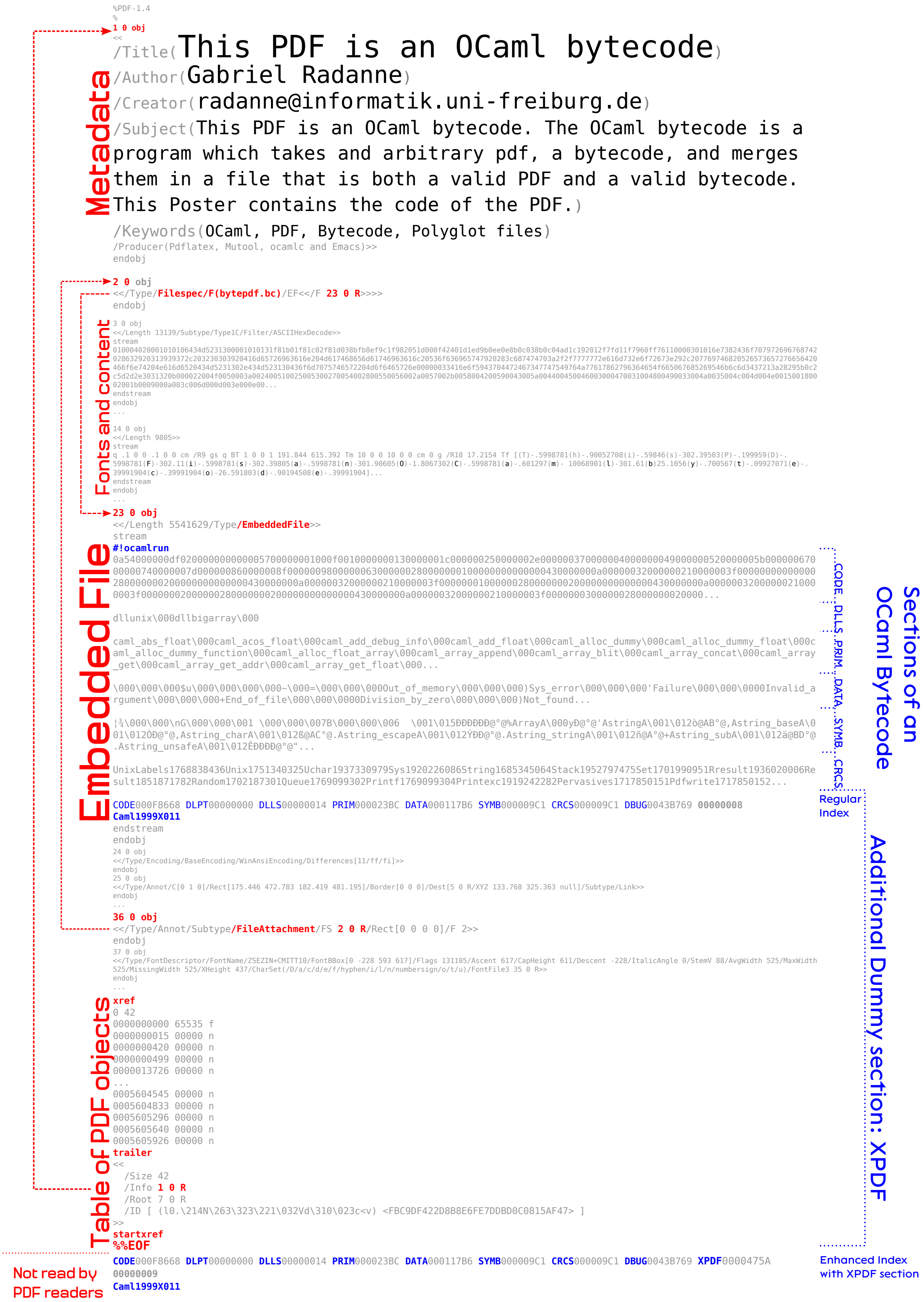
The index is a list of pairs composed of a four letter name and a length in bytes. The order of the sections is not important. The virtual machine knows about a fixed set of sections: CODE, DATA and PRIM (which contains the list of the required C primitives) are mandatory. In addition, it can contain other sections such as DLLS (required libraries), DLPT (where to find libraries), DBUG (debug information), CRCS (CRCs of contained modules), and SYMB (nobody knows, it's not documented, but it's probably about symbols).

```
+--------------------------+
|      Ignored Header      |
+--------------------------+
|        Section 1         |  ^
+--------------------------+  |
|        Section 2         |  |
+--------------------------+  |
             .                |  Sections
             .                |
             .                |
+--------------------------+  |
|        Section N         |  v
+--------------------------+
| Description of Section 1 |  ^
+--------------------------+  |
| Description of Section 2 |  |
+--------------------------+  |
             .                |  Index:
             .                |   n*64 bits
             .                |
+--------------------------+  |
| Description of Section N |  v
+------------+-------------+
| Nb of Sects| MagicNumber |
+------------+-------------+
 one 32bits   twelve 8bits
  integer         chars


  Description of a Section
 +-----------+-----------+
 |   Name    |  Length   |
 +-----------+-----------+
 <----------> <---------->
  four 8bits   one 32bits
    chars        integer
```

[56] Except the PoC||GTFO model, which is obviously perfect.

[57] Caveats include but are not limited to: Portability to potato-based architectures, integer sizes, and native system libraries.



```
%PDF-1.4
%
1 0 obj
<<
/Title(This PDF is an OCaml bytecode)
/Author(Gabriel Radanne)
/Creator(radanne@informatik.uni-freiburg.de)
/Subject(This PDF is an OCaml bytecode. The OCaml bytecode is a
program which takes and arbitrary pdf, a bytecode, and merges
them in a file that is both a valid PDF and a valid bytecode.
This Poster contains the code of the PDF.)
/Keywords(OCaml, PDF, Bytecode, Polyglot files)
/Producer(Pdflatex, Mutool, ocamlc and Emacs)>>
endobj

2 0 obj
<</Type/Filespec/F(bytepdf.bc)/EF<</F 23 0 R>>>>
endobj

3 0 obj
<</Length 13139/Subtype/Type1C/Filter/ASCIIHexDecode>>
stream
010004020001010106434d5231300001010131f81b01f81c02f81d038bfb8ef9c1f982051d000f42401d1ed9b0ee0e8b0c038b0c04ad1c192012f7fd11f7960ff76110000301016e7382436f707972696768742
028632920313939372c203230303920416d65726963616e204d617468656d61746963616c20536f636965747920283c687474703a2f2f7777772e616d732e6f72673e292c207769746820526573657276656420
466f6e74204e616d6520434d5231302e434d523130436f6d7075746572204d6f6465726e00000033416e6f5943704472467347747549764a77617862796364654f665067685269546b6c6d3437213a28295b0c2
c5d2d2e3031320b000022004f0050003a002400510025005300270054002800550056002a0057002b0058004200590043005a004400450046003000470031004800490033004a0035004c004d004e0015001800
02001b0009000a003c006d000d003e000e00...
endstream
endobj
...

14 0 obj
<</Length 9805>>
stream
q .1 0 0 .1 0 0 cm /R9 gs q BT 1 0 0 1 191.844 615.392 Tm 10 0 0 10 0 0 cm 0 g /R10 17.2154 Tf [(T)-.5998781(h)-.90052708(i)-.59846(s)-302.39503(P)-.199959(D)-.
5998781(F)-302.11(i)-.5998781(s)-302.39805(a)-.5998781(n)-301.90605(O)-1.8067302(C)-.5998781(a)-.601297(m)- 10068901(l)-301.61(b)25.1056(y)-.700567(t)-.09927071(e)-.
39991904(c)-.39991904(o)-26.591803(d)-.90194508(e)-.39991904]...
endstream
endobj
...
23 0 obj
<</Length 5541629/Type/EmbeddedFile>>
stream
#!ocamlrun
0a54000000df020000000000000570000000001000f0010000000130000001c000000250000002e000000370000004000000049000000520000005b000000670
00000740000007d000000086000000 8f00000009800000006300000002800000001000000000000000430000000a00000032000000021000003f00000000000000
280000000200000000000000000430000000a000000032000000021000003f0000000100000002800000002000000000000000430000000a000000032000000021000
0003f00000000200000002800000000200000000000000000430000000a000000032000000021000003f0000000030000000280000000020000...

dllunix\000dllbigarray\000

caml_abs_float\000caml_acos_float\000caml_add_debug_info\000caml_add_float\000caml_alloc_dummy\000caml_alloc_dummy_float\000c
aml_alloc_dummy_function\000caml_alloc_float_array\000caml_array_append\000caml_array_blit\000caml_array_concat\000caml_array
_get\000caml_array_get_addr\000caml_array_get_float\000...

\000\000\000$u\000\000\000\000~\000=\000\000\0000ut_of_memory\000\000\000)Sys_error\000\000\000'Failure\000\000\0000Invalid_a
rgument\000\000\000+End_of_file\000\000\0000Division_by_zero\000\000\000)Not_found...

¦¾\000\000\nG\000\000\001 \000\000\007B\000\000\006  \001\015ÐÐÐÐÐÐ@°@%ArrayA\000yÐ@°@'AstringA\001\012ò@AB°@,Astring_baseA\0
01\012ÒÐ@°@,Astring_charA\001\012ß@AC°@.Astring_escapeA\001\012ÝÐÐ@°@.Astring_stringA\001\012ñ@A°@+Astring_subA\001\012ä@BD°@
.Astring_unsafeA\001\012ÊÐÐÐÐ@°@"...

UnixLabels1768838436Unix1751340325Uchar1937330979Sys1920226086String1685345064Stack1952797475Set1701990951Rresult1936020006Re
sult1851871782Random1702187301Queue1769099302Printf1769099304Printexc1919242282Pervasives1717850151Pdfwrite1717850152...

CODE000F8668 DLPT00000000 DLLS00000014 PRIM000023BC DATA000117B6 SYMB000009C1 CRCS000009C1 DBUG0043B769 00000008
Caml1999X011
endstream
endobj
24 0 obj
<</Type/Encoding/BaseEncoding/WinAnsiEncoding/Differences[11/ff/fi]>>
endobj
25 0 obj
<</Type/Annot/C[0 1 0]/Rect[175.446 472.783 182.419 481.195]/Border[0 0 0]/Dest[5 0 R/XYZ 133.768 325.363 null]/Subtype/Link>>
endobj
...
36 0 obj
<</Type/Annot/Subtype/FileAttachment/FS 2 0 R/Rect[0 0 0 0]/F 2>>
endobj
37 0 obj
<</Type/FontDescriptor/FontName/ZSEZIN+CMITT10/FontBBox[0 -228 593 617]/Flags 131105/Ascent 617/CapHeight 611/Descent -228/ItalicAngle 0/StemV 88/AvgWidth 525/MaxWidth
525/MissingWidth 525/XHeight 437/CharSet(/D/a/c/d/e/f/hyphen/i/l/n/numbersign/o/t/u)/FontFile3 35 0 R>>
endobj
...
xref
0 42
0000000000 65535 f
0000000015 00000 n
0000000420 00000 n
0000000499 00000 n
0000013726 00000 n
...
0005604545 00000 n
0005604833 00000 n
0005605296 00000 n
0005605640 00000 n
0005605926 00000 n
trailer
<<
  /Size 42
  /Info 1 0 R
  /Root 7 0 R
  /ID [ (l0.\214N\263\323\221\032Vd\310\023c<v) <FBC9DF422D8B8E6FE7DDBD0C0815AF47> ]
>>
startxref
%%EOF
CODE000F8668 DLPT00000000 DLLS00000014 PRIM000023BC DATA000117B6 SYMB000009C1 CRCS000009C1 DBUG0043B769 XPDF0000475A
00000009
Caml1999X011
```

The current implementation of the virtual machine ignores the content of unknown sections, as long as they use cryptic four-letter names. It also ignores any data before the first section. For convenience, the OCaml compiler adds a shebang at the beginning of the file pointing to the bytecode runtime, but it's not required.

For the curious and the masochistic, non-official documentation of the bytecode and its instructions—it's a neat stack machine—is available.[58] We will content ourselves with this basic knowledge, which is sufficient to use and abuse bytecode files in all sorts of fun ways.

The Safir-Albertini hypothesis states that abusing file formats influences your thought and decisions

PoC||GTFO readers should be familiar with the concept of PDF polyglots, from ZIP files to NES cartridges, including virtual machines and ELF executables.[59] Still, let me give you a quick reminder about PDF internals and how much we can abuse them. Any questions on the matter should be directed to the Funky File Supervisor, Ange Albertini.

The Portable Document Format is a text-based format which is also read from the end with an index of all the blocks (objects) in the file and their offsets. Blocks can point to other blocks, and can contain various pieces of data, such as text or references, but also binary streams that are used for fonts and pictures. Unlike the OCaml virtual machine, PDF readers are rather flexible when interpreting PDF files; indeed, they are nearly as tolerant of awkward dialects and outright syntax errors as HTML4 browsers!

Concretely, this means that PDF files do not have to begin at the beginning nor end at the end of the file. In addition to these classical shenanigans, Ange Albertini showed in PoC||GTFO 4:12 that you can create a PDF file that contains a ZIP that is both accessible directly with unzip and also through Acrobat Reader's file attachment feature. This is done by adding a binary stream that contains the file, then adding some carefully crafted metadata and a trailer.

[58] unzip pocorgtfo19.pdf caml-instructions.pdf caml-formats.pdf

[59] If not, what are you doing here? Go memorize the previous editions by heart! Shoo, shoo!

Proof of Camels

We now have all the ingredients, let's make a PoC! We start with a regular LaTeX file, in which we embed the content using Ange's trick:

```
\immediate\pdfobj stream attr {/Type /EmbeddedFile}
     file {clean.byte}
\immediate\pdfobj{<<
 /Type /Filespec /F (thing.byte) /EF <</F \the\
     pdflastobj\space 0 R>>
 >>}
\pdfannot{
 /Subtype /FileAttachment /FS \the\pdflastobj\space 0 R
 /F 2 % Flag: Hidden
}
```

Our bytecode file `ocaml.byte` is now embedded as an attached file that can be accessed in Acrobat Reader. We then add a suffix that contains an index with an additional section, `PDFX`, that will have the exact length from the beginning of the normal index up to the end of the PDF. Since the bytecode interpreter ignores unknown sections, this is a valid OCaml bytecode file. Since the index is very small, the file is also a valid PDF.[60]

Vulgaris Camelus documentum

PoCs are nice, but libraries are better! Let's make a tool that takes an arbitrary PDF, an arbitrary OCaml bytecode program, and smashes them together. Fortunately, OCaml already has high-quality libraries for dealing with both formats, namely *camlpdf*[61] and *obytelib*.[62] We simply need to grab both files, decompose their structure, make some creative interleavings, and recompose the index to have all the right indices and offsets according to the technique revealed above. Easy peasy![63]

Since the content of the binary stream containing the bytecode must be kept intact, we must take care to disable many traditional optimizations for stream content, most notably compression and reencoding for that stream. The original PDF can be of arbitrary shape and provenance.

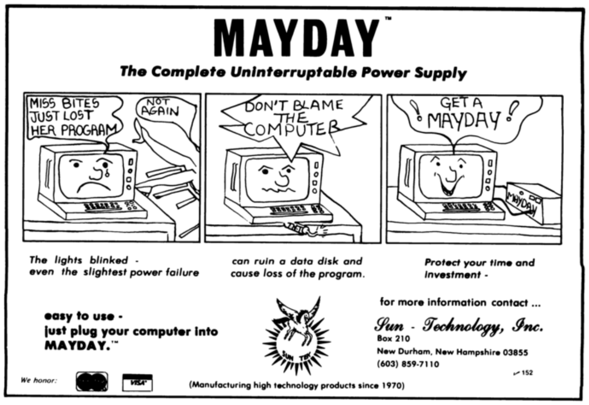
Yo Dawg, I heard you liked polyglots

Having an OCaml tool to smash PDFs and bytecodes together, we can compile that tool to bytecode, and smash it together with a PDF describing the tool itself!

This is in fact slightly more delicate that expected. *Camlpdf* relies on custom C code for encryption and compression, which can't be embedded in normal bytecode. Instead, the OCaml compiler adds ELF metadata in the bytecode to include the C symbols (thus creating a polyglot!). It might be possible to combine everything together, but we can also simply disable these features.

But what if we want *more polyglots*? The question of which formats are polyglot-compatible in the general case is a fairly interesting one. Bytecode and ZIP both require a trailer at the end of the file, and are thus incompatible. However, both are compatible with header-based formats, such as images. Additionally, as long as the other formats have comments (or binary contents; that's obviously the same thing, isn't it?), we can interleave them with OCaml bytecode. The next step is to extend the `bytepdf` tool to make JPEG-PDF-bytecode polyglots. We might also consider OCaml bytecode chimeras, which contain some format in their DATA section, but are also valid files for using this format without duplication. As before, this should be possible with any header-based format that uses offsets.

And now, dear readers, I hope you know what to do for your next research paper(s)!



MAYDAY™

The Complete Uninterruptable Power Supply

The lights blinked - even the slightest power failure

can ruin a data disk and cause loss of the program.

Protect your time and investment -

easy to use - just plug your computer into MAYDAY.™

for more information contact ...

Sun - Technology, Inc.
Box 210
New Durham, New Hampshire 03855
(603) 859-7110

152

We honor: VISA

(Manufacturing high technology products since 1970)

[60] `git clone https://github.com/Drup/polyocamlbyte || unzip pocorgtfo19.pdf polyocamlbyte.zip`

[61] `git clone https://github.com/johnwhitington/camlpdf/ || unzip pocorgtfo19.pdf camlpdf.zip`

[62] `git clone https://github.com/bvaugon/obytelib || unzip pocorgtfo19.pdf obytelib.zip`

[63] `git clone https://github.com/Drup/bytepdf || unzip pocorgtfo19.pdf bytepdf.zip`