10 Post Scriptum: A Schizophrenic Ghost

by Evan Sultanik and Philippe Teuwen

A while back, we asked ourselves,

> What if PoC‖GTFO had completely different content depending on whether the file was rendered by a PDF viewer versus being sent to a printer?

A PostScript/PDF polyglot seemed inevitable. We had already done MBR, ISO, TrueCrypt, HTML, Ruby, ... Surely PostScript would be simple, right? As it turns out, it's actually quite tricky.

```
$ gv pocorgtfo13.pdf
```

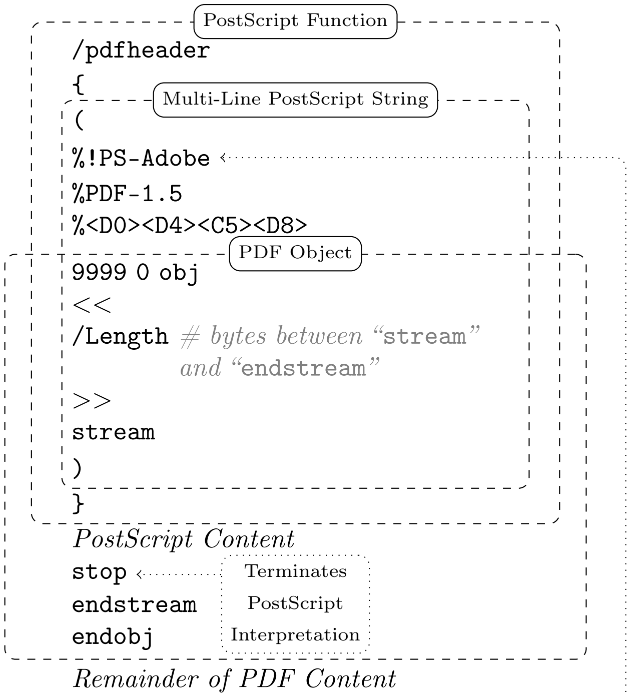
There were two new challenges in getting this polyglot to work:

1. The PDF format is a *subset* of the PostScript language, meaning that we needed to devise a way to get a PDF interpreter to ignore the PostScript code, and *vice versa*; and

2. It's almost impossible to find a PostScript interpreter that doesn't *also* support PDF. Ghostscript is nearly ubiquitous in its use as a backend library for desktop PostScript viewers (*e.g.*, Ghostview), and it has PDF support, too. Furthermore, it doesn't have any configuration parameters to force it to use a specific format, so we needed a way to *force* Ghostscript to always interpret the polyglot as if it were PostScript.

To overcome the first challenge, we used a similar technique to the Ruby polyglot from `pocorgtfo11.pdf`, in which the PDF header is embedded into a multi-line string (delimited by parenthesis in PostScript), so that it doesn't get interpreted as PostScript commands. We halt the PostScript interpreter at the end of the PostScript content by using the handy `stop` command following the standard `%%EOF` "Document Structuring Conventions" (DSC) directive.

This works, in that it produces a file that is *both* a completely valid PDF *as well as* a completely valid PostScript program. The trouble is that Adobe seems to have blacklisted any PDF that starts with an opening parenthesis. We resolved this by wrapping the multi-line string containing the PDF header into a PostScript function we called `/pdfheader`:



```
/pdfheader
{
(
%!PS-Adobe
%PDF-1.5
%<D0><D4><C5><D8>
9999 0 obj
<<
/Length # bytes between "stream"
         and "endstream"
>>
stream
)
}
PostScript Content
stop
endstream
endobj
Remainder of PDF Content
```

The trick of starting the file with a PostScript function worked, and the PDF could be viewed in Adobe. That still leaves the second challenge, though: We needed a way to trick Ghostscript into being "schizophrenic" (*cf.* PoC‖GTFO 7:6), *vi&.*, to insert a parser-specific inconsistency into the polyglot that would force Ghostscript into thinking it is PostScript.

Ghostscript's logic for auto-detecting file types seems to be in the `dsc_scan_type` function inside `/psi/dscparse.c`. It is quite complex, since this single function must differentiate between seven different filetypes, including DSC/PostScript and PDF. It classifies a file as a PDF if it contains a line starting with "`%PDF-`", and PostScript if it contains a line starting with "`%!PS-Adobe`". Therefore, if we put `%!PS-Adobe` anywhere before `%PDF-1.5`, then Ghostscript should be tricked into thinking it is PostScript! The only caveat is that Adobe blacklists any PDF that starts with "`%!PS-Adobe`", so it can't be at the beginning of the file (which is typically where it occurs in DSC files). But that's okay, because Ghostscript only needs it to occur *before* the `%PDF-1.5`, regardless of where.

This article continues in the PostScript!