MENU

FURONER.CAT

Furoner-a (m i f): Dit d'aquella persona apassionada per la informàtica, que té un viu interès per a explorar-ne les característiques i per a posar a prova les seves habilitats en aquest àmbit. (English translation in the About page)

WRITTEN BY FURONER

NOVEMBER 15, 2017

CHECKING FOR MALICIOUSNESS IN

CHECKING FOR MALICIOUSNESS IN ACROFORM OBJECTS ON PDF FILES

INTRODUCTION

One of the most unknown file formats for most IT people is the PDF (Portable Document Format) format type. Originally developed as a universally compatible file format based on the PostScript format, it has become a highly-regarded international format to share documents and information in a structured way. PDF documents are easy to create and use with specific software to read them, and its content and layout is displayed the exact same way no matter which Operating System, device or software is used to view them. Another advantage is that they can be compressed into a file size that is easy to exchange while retaining image quality. They are also multi-dimensional, allowing for the integration of many different types of content (text, images, graphic vectors, videos, audio, animations, forms, hyperlinks, buttons. All this flexibility means an increase in complexity in the background. This is why PDFs are hard to edit, transform or extract content from. In this is also why the format and its structure are widely unknown.

I often get questions on how to analyze various types of files for potential maliciousness. Obviously, the answer is different depending on the file format, but also on the specificities of each file. In the case of

PDFs files, the most known tags for finding malicious code are the likes of */OpenAction*, */AA* (Automatic Action), */Javascript* (or */JS*), */Launch* or */URI*. Many of them are quite self-explanatory and easy to analyse. There is though one special object that people have a hard time understanding how can it be used for malicious purposes and how can it be analyzed. This object is the ***Acroform***.

ACROFORM

An Acrobat Form is as simple as a PDF object that contains form fields. The tricky thing about them that makes them usable for malicious intentions is that, aside from being able of containing objects (text, buttons, images), they can also contain some code (javascript). Moreover, the code can be included inside other objects or even inside object streams (large chunks of encoded and compressed data). All this makes it a perfect container for maliciousness.

I will use a recent file I have received for analysis as a sample on how to do an analysis of *Acroform* objects.

The file was received yesterday by a user, attached in an email with the subject “Global AGILE Development & Innovation Summit – London, UK”. Initially, nothing suggested this could be a malicious email. But the fact the email was sent from a different domain than the link that appeared in the body of the email was enough for the user to become suspicious of it and tag it for investigation before opening.

INITIAL TRIAGE

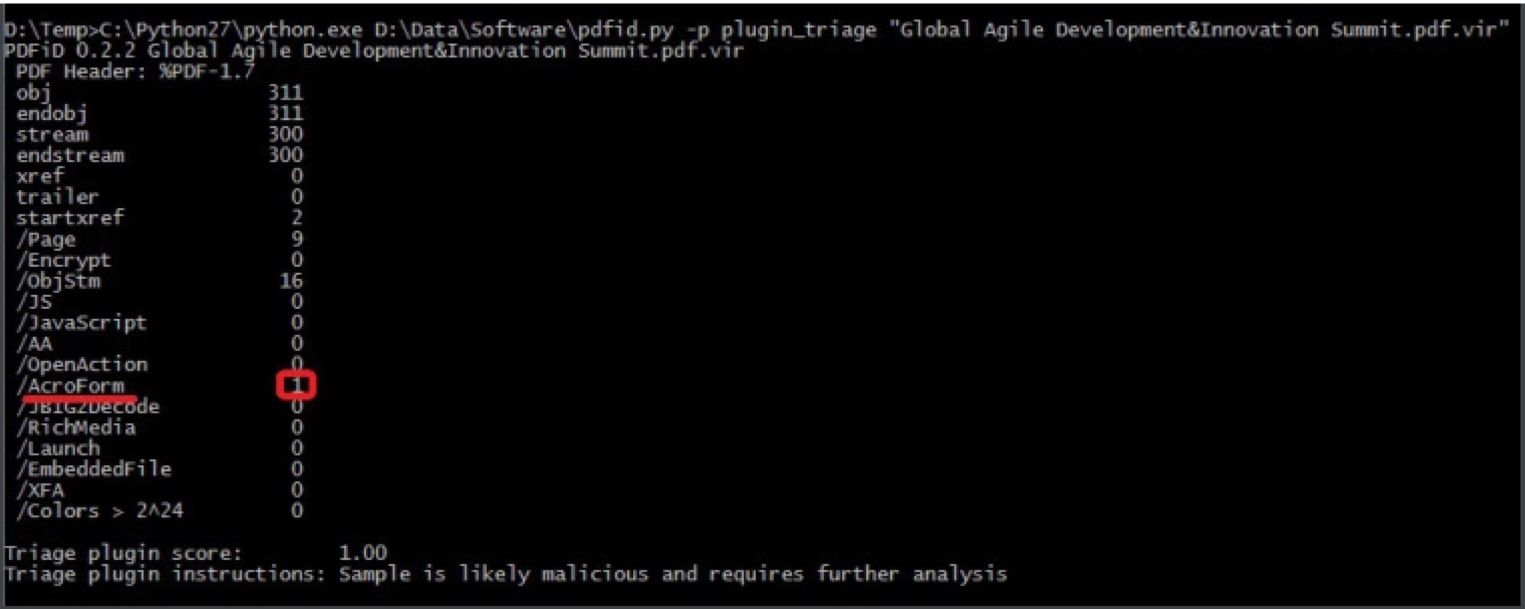
With an initial triage of the file, I got the MD5 hash, I confirmed the Magic number was for a PDF file, version 1.7, and the fact that it appeared already in VirusTotal with zero detections so far.

I recommend a similar type of triage for all files that require deeper investigation, even though someone else has already performed some kind of previous analysis on the file. This way, potential mistakes are avoided.

It is also good to rename the file extension to .VIR to reduce the risk of double-clicking on the file to investigate and executing any potential malicious code.

FIRST STEP WITH EVERY PDF FILE INVESTIGATION

Right after confirming it as a PDF file, the next step is always to use the tool PDFiD from Didier Stevens, with the plugin_triage option, to determine in a quick way if the document contains suspicious tags that require further investigation.



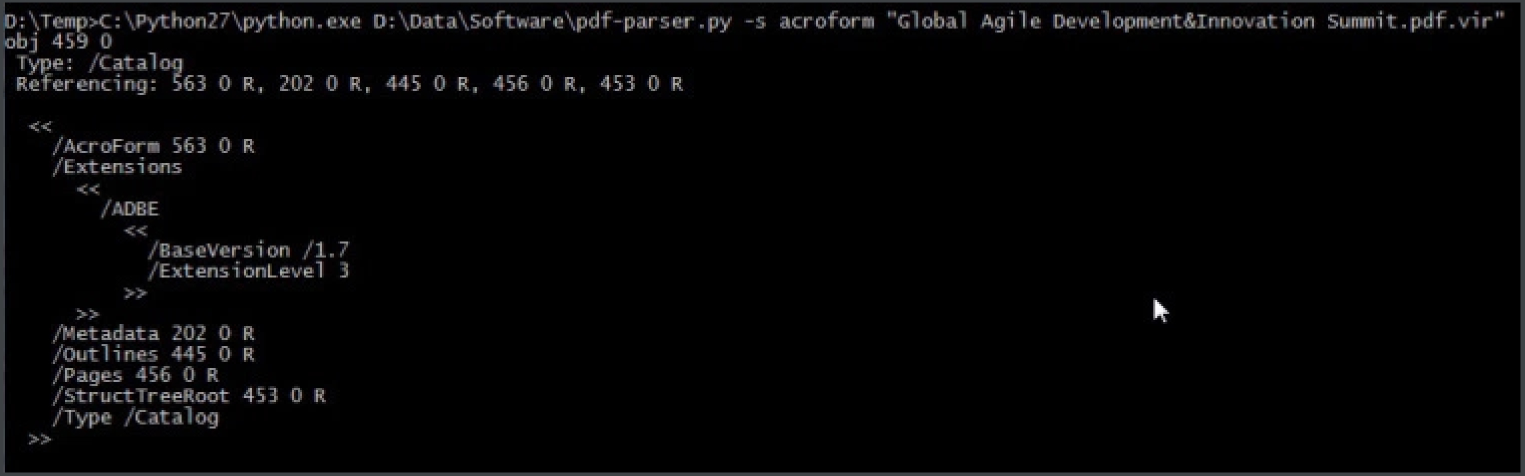
```
D:\Temp>C:\Python27\python.exe D:\Data\Software\pdfid.py -p plugin_triage "Global Agile Development&Innovation Summit.pdf.vir"
PDFiD 0.2.2 Global Agile Development&Innovation Summit.pdf.vir
 PDF Header: %PDF-1.7
 obj                   311
 endobj                311
 stream                300
 endstream             300
 xref                    0
 trailer                 0
 startxref               2
 /Page                   9
 /Encrypt                0
 /ObjStm                16
 /JS                     0
 /JavaScript             0
 /AA                     0
 /OpenAction             0
 /AcroForm               1
 /JBIG2Decode            0
 /RichMedia              0
 /Launch                 0
 /EmbeddedFile           0
 /XFA                    0
 /Colors > 2^24          0

Triage plugin score:        1.00
Triage plugin instructions: Sample is likely malicious and requires further analysis
```

As you can see in the image, this file did not contains many objects in 9 different pages, 16 object streams, and only 1 really suspicious element: an Acroform.

This means we can not easily dismiss the file as benign and we need to go deep in the investigation.

FINDING WHAT IS INSIDE THE ACROFORM

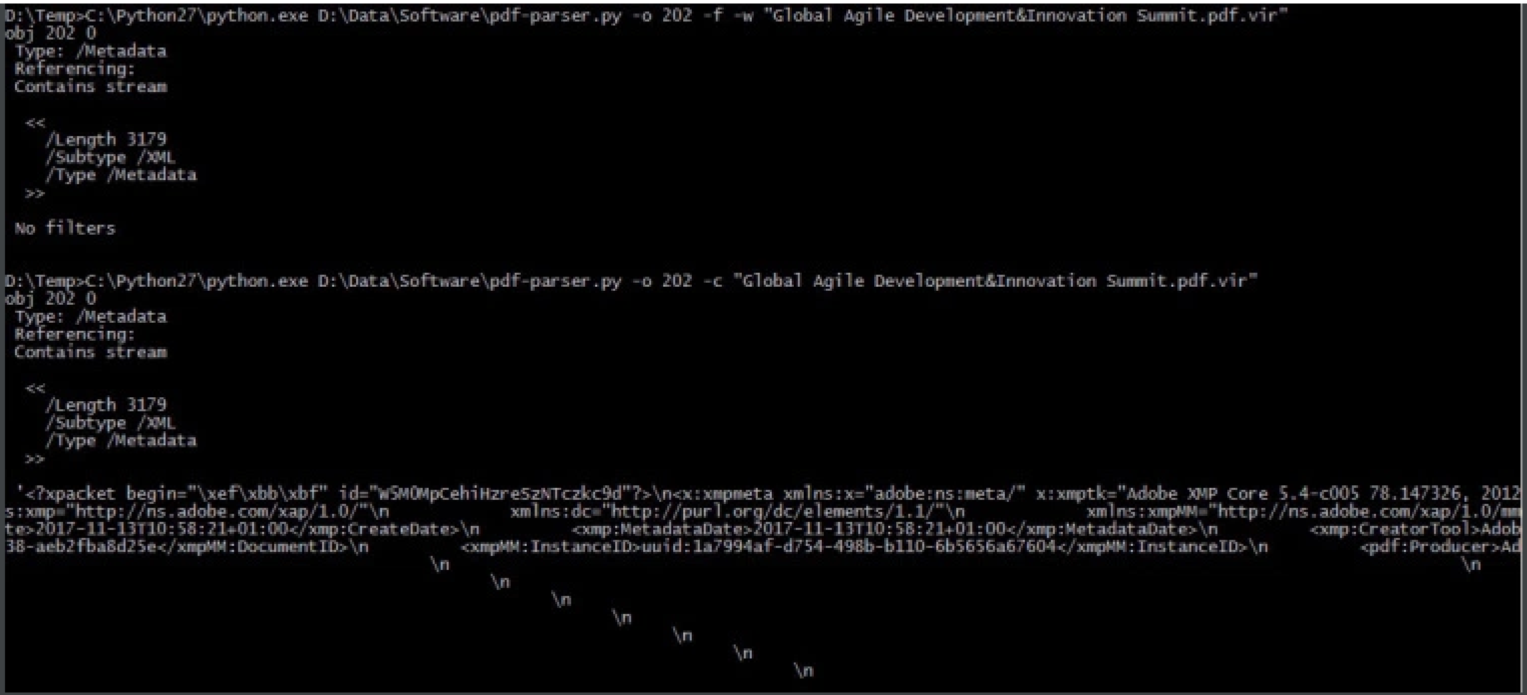
How? By using yet another tool by Didier Stevens, called pdf-parser. The tool has a search option that looks for strings or indirect objects (not inside streams). The search is not case-sensitive and is susceptible to obfuscation techniques, but does its job when attempting to find any object related for instance to Acroforms.



```
D:\Temp>C:\Python27\python.exe D:\Data\Software\pdf-parser.py -s acroform "Global Agile Development&Innovation Summit.pdf.vir"
obj 459 0
 Type: /Catalog
 Referencing: 563 0 R, 202 0 R, 445 0 R, 456 0 R, 453 0 R

  <<
    /AcroForm 563 0 R
    /Extensions
      <<
        /ADBE
          <<
            /BaseVersion /1.7
            /ExtensionLevel 3
          >>
      >>
    /Metadata 202 0 R
    /Outlines 445 0 R
    /Pages 456 0 R
    /StructTreeRoot 453 0 R
    /Type /Catalog
  >>
```

The image above shows how I used the search option (-s) to find where was the Acroform of the sample in question. In this case, object 459 contained an Acroform and referenced 4 other objects. Objects 563, 445, 456 and 453 were actually empty or non-existent.

But object 202 contained some kind of data. In order to find out what that data could be, we can use the options filter (-f) and raw (-w) options to check the content of potential encoded and compressed (filtered) data in the objects (see first example in the below image).



```
D:\Temp>C:\Python27\python.exe D:\Data\Software\pdf-parser.py -o 202 -f -w "Global Agile Development&Innovation Summit.pdf.vir"
obj 202 0
 Type: /Metadata
 Referencing:
 Contains stream

  <<
    /Length 3179
    /Subtype /XML
    /Type /Metadata
  >>

 No filters


D:\Temp>C:\Python27\python.exe D:\Data\Software\pdf-parser.py -o 202 -c "Global Agile Development&Innovation Summit.pdf.vir"
obj 202 0
 Type: /Metadata
 Referencing:
 Contains stream

  <<
    /Length 3179
    /Subtype /XML
    /Type /Metadata
  >>

 '<?xpacket begin="\xef\xbb\xbf" id="W5M0MpCehiHzreSzNTczkc9d"?>\n<x:xmpmeta xmlns:x="adobe:ns:meta/" x:xmptk="Adobe XMP Core 5.4-c005 78.147326, 2012
s:xmp="http://ns.adobe.com/xap/1.0/"\n            xmlns:dc="http://purl.org/dc/elements/1.1/"\n            xmlns:xmpMM="http://ns.adobe.com/xap/1.0/mm
te>2017-11-13T10:58:21+01:00</xmp:CreateDate>\n         <xmp:MetadataDate>2017-11-13T10:58:21+01:00</xmp:MetadataDate>\n         <xmp:CreatorTool>Adob
38-aeb2fba8d25e</xmpMM:DocumentID>\n         <xmpMM:InstanceID>uuid:1a7994af-d754-498b-b110-6b5656a67604</xmpMM:InstanceID>\n         <pdf:Producer>Ad
                                           \n                                                                                                        \n
                                                 \n
                                                       \n
                                                             \n
                                                                   \n
                                                                         \n
                                                                               \n
```

Or we can check the content of objects without streams or with streams without filtes using the content option (-c), as it is shown in the second example in the above image.

The sample analysed was clearly a non-malicious PDF file.

EXAMPLES OF MALICIOUS CODE INSIDE ACROFORMS

Since I have not seen any recent example of a malicious file that uses Acroforms, I leave here two well-explained cases of such examples from Malwarebytes in 2013, and from CountUponSecurity in 2014.

In both articles the authors extract some javascript code hidden inside an Acroform object that is used by the malicious actors to end up executing embedded shellcode.

OTHER INTERESTING LINKS TO READ ABOUT PDF

Texts with in-depth explanation of the PDF file format to understand the files better when doing analysis on them:

Quickpost: About the Physical and Logical Structure of PDF Files

PDF Stream Objects

PDF File Format Reference and Adobe Extensions

Other resources for analyzing PDF files:

Analyzing malicious PDFs from Infosec Institute

Analyzing malicious documents cheatsheet and explanations from Lenny Zeltser

Tools:

PDF Tools by Didier Stevens

PeePDF by Jose Miguel Esparza

SHARE THIS:

Twitter LinkedIn Pocket Google Facebook 1 Reddit

Like

Be the first to like this.

POSTED IN HOW TO, MALWARE ANALYSIS. TAGGED ACROFORM, ANALYSIS, PDF.

ONE THOUGHT ON "CHECKING FOR MALICIOUSNESS IN ACROFORM OBJECTS ON PDF FILES"

Pingback: Week 46 – 2017 – This Week In 4n6

Comments are closed.

DISCLAIMER

This is an independent blog with the sole purpose of publishing my research in forensics and malware analysis.

Opinions and statements expressed in this blog are solely my own and do not express the views of my employer.

SEARCH BAR

Search …

SOCIAL

TAGS

Acroform analysis CVE-2017-0199 DDE deobfuscation macroless malware PDF RTF

PAGES

- About

PREVIOUS POST NEXT POST