From AMSI to Reflection 0x0

S rxored.github.io/post/csharploader/bypassing-amsi-with-csharp

2021-10-23



Table of Content

Introduction

In Windows environments, in both initial access and post-exploitation phases, script-based malware plays a major role. Often, hackers utilize microsoft office suite to gain initial access (using droppers, loaders) to the victim and Windows powershell to explore internal network, perform scans... basically to do the post exploitation stuff. (well of course, there are powershell based droppers.)

There is something that is common to both of these tools. Windows scripting engine.

And as a result, Microsoft and antimalware vendors have developed many security mechanisms to deal with those threats that utilize script-based malware. For example, modern anti-malware solutions can statically analyze scripts, binaries and detect whether they are malicious or not using signatures such as strings.

And because of that, malware authors use various techniques to bypass those defense mechanisms. One of the major techniques is code obfuscation.

consider the following example, that I took from MSDN.

```
function displayEvilString
{
    Write-Host 'pwnd!'
}
```

Assuming the above PowerShell snippet is malicious, we can write a signature to detect the malware. this signature can be Write-Host 'pwnd!' or simply 'pwnd!'.

So to avoid signature-based detection, the above snippet can be obfuscated like shown below.

```
function obfuscatedDisplayEvilString
{
    $xorKey = 123
    $code = "LHsJexJ7D3see1Z7M3sUewh7D3tbe1x7C3sMexV7H3tae1x7"
    $byte = [Convert]::FromBase64String($code)
    $newBytes = foreach($byte in $bytes) {
        $byte -bxor $xorKey
    }
    $newCode = [System.Text.Encoding]::Unicode.GetString($newBytes)
}
```

And this is a win for malware authors since this is beyond what anti-malware solutions can emulate or detect until AMSI joins the conversation.

Antimalware Scan Interface

Antimalware Scan Interface, AMSI for short is a standard interface that allows applications to interact with anti-malware products installed on the system. This means is that it provides an API for Application developers. Application developers can use the API to implement security features to make sure that the end-user is safe.

AMSI also enables anti malware vendors to defend againts script based malware.

According to Microsoft, AMSI provides the following features by default.

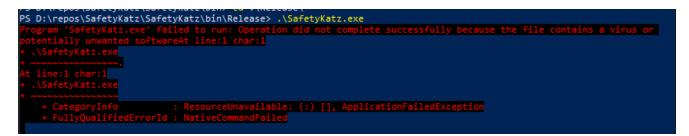
- User Account Control
- PowerShell
- Windows Script Host
- JScript && VBScript
- Office VBA macros

As it is clear from those default features, AMSI specifically provides anti-malware security mechanisms to defend against script-based malware.

AMSI in action

So let's take Safetykatz as our example.

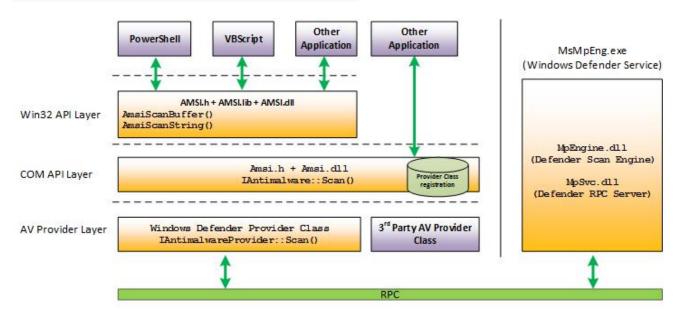
When we run the binary, the result we get is.



See, as we expected, PowerShell stops the execution of the program once it has detected the program is suspicious using AMSI. So, how can we bypass this?, well before that, we have to dive deep into AMSI internals to understand how things work.

AMSI internals

As I previously mentioned, AMSI enables anti malware vendors to defend againts script based malware. This is done by using AMSI providers. An AMSI provider is basically a COM object that implements IAntimalwareProvider COM interface. An anti malware vendor who's willing to implement AMSI interface should then register the COM object by creating a CLSID entry in HKLM\CLSID and registering the same CLSID under HKLM\Software\Microsoft\AMSI\Providers\.



As it is shown in the above diagram, AMSI provides a dll called **amsi.dll** for application developers to interfere with AMSI providers indirectly.

Let's examine PowerShell from process hacker to check whether amsi.dll is loaded.

Name	Base address	Size	Description	-
powershell.exe	0x7ff7ff8a0000	452 kB	Windows PowerShell	
advapi32.dll	0x7ff96cc10000	652 kB	Advanced Windows 32 Base	
amsi.dll	0x7ff945500000	92 kB	Anti-Malware Scan Interface	
AppResolver.dll	0x7ff939db0000	592 kB	App Resolver	
AppxSip.dll	0x7ff944e70000	288 kB	Appx Subject Interface Pac	
itl.dll	0x7ff957b20000	112 kB	ATL Module for Windows XP	
CP47Langs.dll	0x7ff9628b0000	372 kB	BCP47 Language Classes	
crypt.dll	0x7ff96b060000	152 kB	Windows Cryptographic Pri	
ocryptprimitives.dll	0x7ff96bb00000	516 kB	Windows Cryptographic Pri	
dp.dll	0x7ff957410000	5.27 MB	Microsoft (R) CDP Client API	
fgmgr32.dll	0x7ff96b010000	296 kB	Configuration Manager DLL	
lbcatq.dll	0x7ff96d330000	648 kB	COM+ Configuration Catalog	
ir.dll	0x7ff94d200000	10.76 MB	Microsoft .NET Runtime Com	
lrjit.dll	0x7ff948c60000	1.31 MB	Microsoft .NET Runtime Just	
ombase.dll	0x7ff96c210000	3.21 MB	Microsoft COM for Windows	
coml2.dll	0x7ff96c550000	472 kB	Microsoft COM for Windows	
rypt32.dll	0x7ff96b980000	1.29 MB	Crypto API32	
rypt32.dll.mui	0x23419610000	40 kB	Crypto API32	
ryptbase.dll	0x7ff96a6a0000	48 kB	Base cryptographic API DLL	
ryptnet.dll	0x7ff964c30000	188 kB	Crypto Network Related API	
ryptsp.dll	0x7ff96b960000	92 kB	Cryptographic Service Provi	
cscapi.dll	0x7ff948de0000	72 kB	Offline Files Win32 API	
versions.2.db	0x2347ea20000	16 kB		
versions.2.db	0x2347ea80000	16 kB		
lsreg.dll	0x7ff967760000	832 kB	AD/AAD User Device Registr	
di32.dll	0x7ff96cd70000	152 kB	GDI Client DLL	
di32full.dll	0x7ff96bb90000	1.59 MB	GDI Client DLL	
jpapi.dll	0x7ff969880000	136 kB	Group Policy Client API	
ertutil.dll	0x7ff95d4f0000	2.65 MB	Run time utility for Internet	

as we can see, amsi.dll has been loaded into powershell.exe. Now, let's take a look at this dll in-depth and see if we can find anything interesting. Even without looking at the dll, it is possible to think of some techniques to bypass AMSI, Anyway, its time to dig deep.

Before start reading disassembly, let's examine the export table of amsi.dll.



Out of the above exported functions, only two are important to us.

- AmsiInitialize
- AmsiScanBuffer
- AmsiScanString

– 🗆 🗙

Of course there are some other important exports. To name a few, DllRegisterClass, DllGetClassObject and AmsiUacScan.

First we'll go through AmsiScanBuffer.

AmsiScanString

Microsoft documentation does not tell us much about AmsiScanString function. However it gives some basic information about it. Such as,

it's prototype,

```
HRESULT AmsiScanString(

[in] HAMSICONTEXT amsiContext,

[in] LPCWSTR string,

[in] LPCWSTR contentName,

[in, optional] HAMSISESSION amsiSession,

[out] AMSI_RESULT *result

);
```

and parameter information.

According to the documentation, The first parameter this function accepts is **amsiContext**, which is a handle of type **HAMSICONTEXT** that was initially received from AmsiInitialize.

Second and third parameters hold pointers to wide character strings. first one for the string that should be scanned and the latter for the **contentName**.

contentName can be either filename, script id, url or similar of the content being scanned.

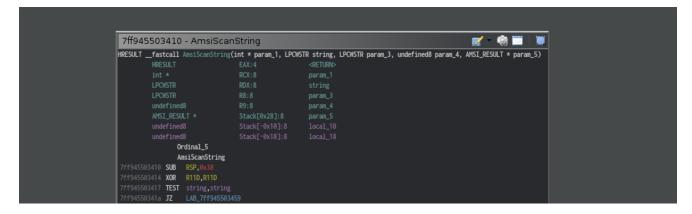
Fourth parameter is marked optional, however if multiple scan requests are to be correlated within a session, this parameter should be set to the handle returned by AmsiOpenSession function.

Fifth parameter is an output parameter and this is the one that indicates whether the input string is malicous or not.

As MSDN says, this function (and AmsiScanBuffer) returns **S_OK** if the call is successful. However, the return value does not indicate whether the buffer is malicious. instead, the function uses fifth parameter of type **AMSI_RESULT** to send the scan results to caller.

```
typedef enum AMSI_RESULT {
    AMSI_RESULT_CLEAN,
    AMSI_RESULT_NOT_DETECTED,
    AMSI_RESULT_BLOCKED_BY_ADMIN_START,
    AMSI_RESULT_BLOCKED_BY_ADMIN_END,
    AMSI_RESULT_DETECTED
} ;
```

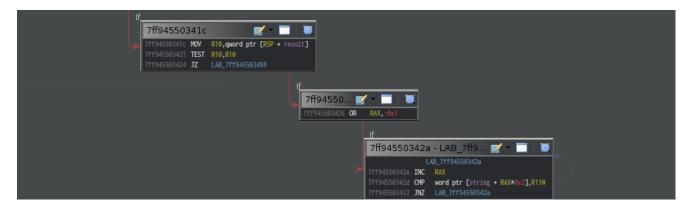
Let's a take a look at AmsiScanString in disassembly.



Function allocates some space in the stack and checks if the string is empty or not. If **string** turns out to be empty, it simply returns after loading **0**×80070057 into **rax**.

	LAB_7ff945503459		XREF[3]:	7ff94550341a(j) 7ff945503424(j) 7ff945503440(j)
7ff945503459 b8 57 00 07 80	MOV			
7ff94550345e 48 83 c4 38 7ff945503462 c3	LAB_7ff94550345e ADD RET	RSP,0x38	XREF[1]:	7ff945503457(j)

if string to be scanned is not null,



function checks if **result** is null pointer. if so, well the same thing as above, it returns with bad value loaded into **rax**.

else, **result** is valid, it loops through each wide character of the **string** to get the length of it.



After getting the string length, it calls AmsiScanBuffer function.

It is clear that this is just a simple wrapper function around AmsiScanBuffer .

AmsiScanBuffer

According to the MSDN and as well as the name suggests, the AmsiScanBuffer function scans a buffer for malicous content.

here is the function prototype <u>msdn</u>

HRESULT AmsiScan	Buffer(
[in]	HAMSICONTEXT	amsiContext,
[in]	PVOID	buffer,
[in]	ULONG	length,
[in]	LPCWSTR	contentName,
[in, optional]	HAMSISESSION	amsiSession,
[out]	AMSI_RESULT	*result
);		

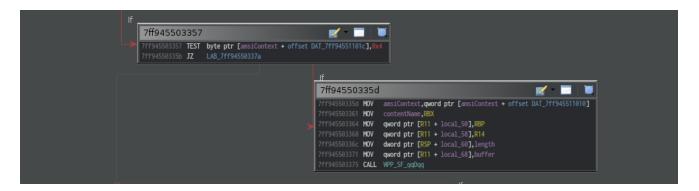
Function takes 6 parameters. One of which is the pointer to the AMSI_RESULT enum which i explained above - *result . According to MSDN, others include a buffer, which will be scanned by the anti-malware vendor - buffer , length of the buffer - length , filename, URL, unique script ID - contentName and a handler to the session - HAMSISESSION structure.

And here's how this function looks like in disassembly.



here we can see stack pointer is stored in **r11** register and since this is x64 _stdcall, the first four parameters are stored in rcx, rdx, r8 and r9 registers. Rest are stored in the stack. With that information, we can assume a pointer to the AMSI_RESULT enum is stored in the stack.

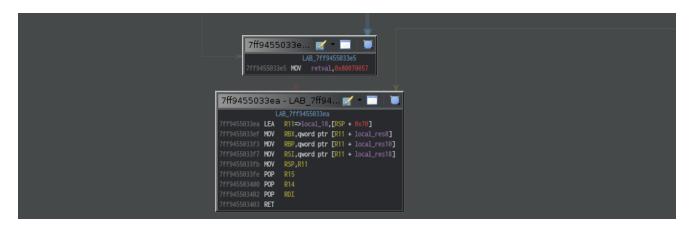
then we can see few comparisons around global data. if the comparisons turns out to be successful, it calls WPP_SF_qqDqq function. (windows sofware trace preprocessor).



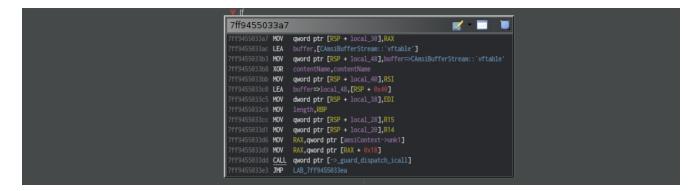
then there is a pretty huge if condition, which is essentially checks if any of the above parameters are invalid

180003 Z - U. U. 1000337a 00337a TEST RSI,RSI 00337d JZ U.L. 1000033e5	If 18000337f 🛒 * 🛄 📰
	H 1600033
	1800033.
	f 180003395 ₩3 8.4.,qwrd ptr [%8.+ 0x8] 03395 ₩37 8.4.,qwrd ptr [%8.+ 0x8] 03395 ₩37 8.4.,qwrd 03395 ₩37 8.4.,qwrd 03395 ₩37 8.4.,qwrd ptr [%8.+ 0x8]

by looking at the comparison, the function won't successfully return if **[rbp]**, which is the first qword of **amsiContext** is not equal to 0x49534d41.



And if parameters invalid, it returns 0×80070057 (which i think is the bad return value)



else, as we can see in the above snippet, **buffer** (rdx register) is now loaded with address of **CAmsiBufferStream::vftable** and stored the value in the stack. This may sound familiar to anyone who has done some C++ reverse engineering since this is a one way to represent constructor calls in assembly (setting vtable to the object's first bytes).

to confirm that we can take a look at CAmsiBufferStream::vftable.

			* const CAmsil	***************************** BufferStream::'vftable' ******************************* erStream006000 ream::'vftable'				
_							AmsiScanBuffer:7ff9455033b3(*)	
		11111111111111111111111111111111111111	addr[6]					
		50 45 f9 7f 0						
		7ff94550ba30 d0 2b 50 45 00	f97f00 add	r CAmsiBufferStrea	n::QueryInterface [פ		<pre>'scalar_deleting_destructor':7ff945502f16(*). 'scalar_deleting_destructor':7ff945502f20(*). AmsiScanBuffer:7ff9455033ac(*) AmsiScanBuffer:7ff9455033b3(*)</pre>
	\vdash	7ff94550ba38 d0 2c 50 45 00	f9 7f 00 add	r CAmsiBufferStrea	n::AddRef [1]		
	F	7ff94550ba40 80 2c 50 45 00	f9 7f 00 add	r CAmsiBufferStrea	n::Release [2]		
	┢	7ff94550ba48 20 2d 50 45 00	f9 7f 00 add	r CAmsiBufferStrea	n::GetAttribute [3]		
	\vdash	7ff94550ba50 c0 2e 50 45 00	f9 7f 00 add	r CAmsiBufferStrea		4]		
		7ff94550ba58 10 2f 50 45 00	f9 7f 00 add	r CAmsiBufferStrea	n::`scalar_dele[5]		

as we can see, **CAmsiBufferStream::vftable** is indeed, a virtual function table and what those two instructions doing is creating an object of type **CAmsiBufferStream**. It is also possible to see some member variable intializations too.

My assumption is that **amsiContext->thirdMember** is somekind of a class that antimalware vendor has registered to perform scans.

To make sure our assumptions so far are correct, we'll go over this function using windbg.

Since we already know interesting parts of the function, it is easy to place breakpoints.

0:018> bl	
0 e Disable Clear 00007ffxxxxx3310	0001 (0001) 0:**** amsi!AmsiScanBuffer
1 e Disable Clear 00007ffxxxxx338d	0001 (0001) 0: amsi!AmsiScanBuffer+0x7d
2 e Disable Clear 00007ffx`xxxx3395	0001 (0001) 0:
amsi!AmsiScanBuffer+0x85	
3 e Disable Clear 00007ffxxxx339e	0001 (0001) 0:****
amsi!AmsiScanBuffer+0x8e	
4 e Disable Clear 00007ffxxxxx33ac	0001 (0001) 0:**
amsi!AmsiScanBuffer+0x9c	

First few breakpoints are placed at locations in assembly where **amsiContext's** member variables are being referenced. Reason being this handle is still unknown to us. Therefore it could be useful to extract every possible information about it. Last breakpoint is placed at the address where **CAmsiBufferStream:vftable** is referenced.

File Home	View Brea	kpoints	Time Travel Model Scripting Sou	irce Memor	y Command			
2 2	Ox 🕋	011010 101010			Accent color	.		
18.)			
mand Watch Locals Re	gisters Memory St	tack Disassemt	ly Threads Breakpoints Logs Notes Timelines Modules	s Layouts Rese • Windo				
		Windo	vs	Window Layou				
sembly				- ☆ ×	Command ×			
ress: @\$scopeip			✓ Follow current instruction		0:018> dd @rbx 00000234\7f5d44d0	49534d41 0000000	7e90cce0 000002	34
07++9`4550333b 488	d05bedc0000	Lea rax.		נטו	00000234`7f5d44e0	7eb5d120 00000234	4 0000008 000000	90
07ff9`45503342 488		nov rbp	qword ptr [rsp+0B8h]		00000234`7f5d44f0		9 d772fa7a 9000086	
07ff9`4550334a 4c8			qword ptr [rsp+0B0h]		00000234`7f5d4500 00000234`7f5d4510		4 7e8d7020 0000023 4 7f5db1d0 0000023	
07ff9`45503352 483 07ff9`45503355 742			rax !AmsiScanBuffer+0x6a (00007ff9`4550337a)		00000234 7f5d4520		9 d771fa67 8f00090	
07ff9`45503357 f64			ptr [rcx+1Ch], 4			636f6843 74616c6		
007ff9`4550335b 741	d	je amsi			00000234 \ 7f5d4540		9 745c3a43 736c6f6	6 1
07ff9`4550335d 488			qword ptr [rcx+10h]		0.018> dd 0000234` 00000234`7f5d44d8		7eb5d120 000002	34
07ff9`45503361 4c8 07ff9`45503364 498		nov r9, nov gwoi	rbx d ptr [r11-50h], rbp		00000234`7f5d44e8	00000008 0000000	0000000 000000	90
97ff9`45503368 4d8			d ptr [r11-500], r0p d ptr [r11-58h], r14		00000234`7f5d44f8	d772fa7a 90000800	9 7f5d43e0 0000023	34
97ff9`4550336c 448	9442428	nov dwor	d ptr [rsp+28h], r8d		00000234\7f5d4508 00000254\7f5d4518		4 7f594740 0000023 4 00000000 0000000	
07ff9`45503371 498			d ptr [r11-68h], rdx		00000234`7f5d4518		9 636f6843 74616c6	
07ff9`45503375 e8d 07ff9`4550337a 488			<pre>!WPP_SF_qqDqq (00007ff9`45502b4c) rsi</pre>		00000234`7f5d4538		f 7461636f 3d6e6f6	
07ff9`4550337d 746			!AmsiScanBuffer+0xd5 (00007ff9`455033e5)		00000234`7f5d4548	745c3a43 736c6f6	0000000 000000	90
97ff9`4550337f 85f			edi		0:018> dd 0000234` 00000234`7f5d44e0		00000000 000000	
07ff9`45503381 746 07ff9`45503383 488			!AmsiScanBuffer+0xd5 (00007ff9`455033e5)		00000234 7f5d4400			
07ff9`45503386 745			rbp !AmsiScanBuffer+0xd5 (00007ff9/455033e5)		000002341775d4500	7f5d43e0 00000234	4 7e8d7020 0000023	34
97ff9`45503388 488		test rbx,	rbx		00000234`7f5d4510	7f594740 00000234		
07ff9`4550338b 745			!AmsiScanBuffer+0xd5 (00007ff9)455033e5)		00000234 \ 7f5d4520 00000234 \ 7f5d4530		0 d771fa67 8f00090 f 6f547965 4c736c0	
07ff9`4550338d 813 07ff9`45503393 755			d ptr [rbx], 49534D41h !AmsiScanBuffer+0xd5 (00007ff9`455033e5)		00000234 7f5d4540		9 745c3a43 736c6f6	
07ff9`45503395 488			qword ptr [rbx+8]		00000234`7f5d4550			
07ff9`45503399 488	5c0 t	test rax,	rax					
07ff9`4550339c 744			AmsiScanBuffer+0xd5 (00007ff9-455033e5)					
07ff9`4550339e 488 07ff9`455033a2 488			<pre>qword ptr [rbx+10h] ds 00000234`7f5d44e0=0 rcx</pre>	00002347eb5				
07ff9`455033a5 743			!AmsiScanBuffer+0xd5 (00007ff9`455033e5)		•			
07ff9`455033a7 488	9442458	ιον γοι	d ptr [rsp+58h], rax		0:018>			
07ff9`455033ac 488			[amsi!CAmsiBufferStream::`vftable' (00007f	f9`4550ba30	0:010>			
07ff9`455033b3 488 07ff9`455033b8 453			d ptr [rsp+40h], rdx r9d		Memory		Registers	
7ff9`455033bb 488			d ptr [rsp+48h], rsi				Name	Value
7ff9`455033c0 488			[rsp+40h]		Address: @rsp		🗉 User	
7ff9`455033c5 897 7ff9`455033c9 4c8		nov dwor nov r8,	d ptr [rsp+50h], edi			94D4298 0000023 🔺	rax	0x000002347e90cce0
7ff9`455033cc 4c8			d ptr [rsp+60h], r15			0000060 0000000	rbx	0x000002347f5d44d0
7ff9`455033d1 4c8	9742468	nov qwor	d ptr [rsp+68h], r14			C04E920 00007FF E8C3C30 0000023	rex	0x00007ff945511d38
07ff9`455033d6 488		nov rax,	qword ptr [rcx]			D268BB0 00007FF	rdx	0x0000023402aa9374
97ff9`455033d9 488 97ff9`455033dd ff1		nov rax, call gwor	<pre>qword ptr [rax+18h] d ptr [amsi!_guard_dispatch_icall_fptr (000</pre>	07ff0\/1550c	000000D6`A4A0EB40 1	94D4298 0000023	rsi	0x0000023402aa937
7ff9`455033e3 eb0			d ptr [ams1:_guard_dispatch_icatt_tptr (000 !AmsiScanBuffer+0xda (00007ff9`455033ea)			141D8E8 0000023	rdi	0x000000000000000000000000000000000000
7ff9`455033e5 b85	7000780	iov eax,	80070057h			4A0EC98 000000D	rsp	0x000000d6a4a0eaf0
7ff9`455033ea 4c8			[rsp+70h]			2AA9374 0000023	rbp	0x000000d6a4a0ec98 0x00007ff94550339e
7ff9`455033ef 498 7ff9`455033f3 498			qword ptr [r11+20h] qword ptr [r11+28h]			0000008 000000	rip efl	0x00007ff94550339e
07ff9`455033f7 498		nov rop, nov rsi,	qword ptr [r11+200] qword ptr [r11+30h]			0000008 0000000	cs	0x0033
7ff9`455033fb 498	be3 🛛	nov rsp,	rll			558BFA1 0000C4E	ds	0x002b
7ff9`455033fe 415		pop r15			00000006 A4A0EBC0 A	AA9F4C0 0000000 AA5E510 00007FF	es	0x002b
07ff9`45503400 415 07ff9`45503402 5f		pop r14 pop rdi			AAAAAAAAA AAAAAAAAAAAAAAAAAAAAAAAAAAAA	RAF75A7 AAAA7EE	fs	0x0053
977779 45503402 57		pop rdi					gs	0x002b
					Watch Stack Memor			0x002b

So from the above image, we can assume that the first member of the amsiContext is a QWORD but it compares it with a DWORD and second and third members are also QWORDs (8 bytes).

0:018> dq /c1 0x000002347f5d44d8 L1 000002347f5d44d8 000002347e90cce0 0:018> dq /c1 0x000002347f5d44e0 L1 000002347f5d44e0 000002347eb5d120

We can refer to the memory map to get more information about what those QWORDs are.

			and the second sec	
✓ 0x2347f550000	Private	1,024 kB RW	Heap segment (ID 1)	1,016 kB 1,016 kB
0x2347f550000	Private: Commit	1,020 kB RW	Heap segment (ID 1)	1,016 kB 1,016 kB
0x2347f64f000	Private: Rese	4 kB	Heap segment (ID 1)	

Now it is clear those two pointers are from heap segment 1. However, we still have no idea about the type of those pointers.

However we already know those are pointers to objects thanks to our previous static analysis.

Disassembly		👻 🖈 👻	Command X	
Address: @\$scopeip 00007ff9`45503388 4885db	✓ Follow current instruction	•	0:018> dq /c1 00007ff9'4550ba30 L3 00007ff9'4550ba30 00007ff9'45502bd0 00007ff9'4550ba38 00007ff9'45502cd0 00007ff9'4550ba40 00007ff9'45502cd0	
00007ff9`4550338b 7458 00007ff9`4550338d 813b414d5349 00007ff9`45503393 7550	je amsilAmsiScanBuffer+0xd5 (00007ff9`455033e5) cmp dword ptr [rbx], 49534041h jne amsiAmsiScanBuffer+0xd5 (00007ff9`455033e5)		0:018> u 00007ff9'45502bd0 amsi!CAmsiBufferStream:: QueryInterface:	Formula Luc
00007ff9`45503395 488b4308 00007ff9`45503399 4885c0 00007ff9`4550339c 7447	<pre>mov rax, qword ptr [rbx+8] test rax, rax ie amsi/AmsiScanBuffer+0xd5 (00007ff9`455033c5)</pre>		00007ff9`45502bda 57 push rdi	[rsp+8],rbx [rsp+10h],rsi
00007ff9`4550339e 488b4b10 00007ff9`455033a2 4885c9	mov rcx, qword ptr [rbx+10h] test rcx, rcx		00007ff9'45502bdb 4832ec20 sub rsp,20h 00007ff9'45502bdf 498bf8 mov rdi,r8 00007ff9'4502bc2 488bda mov rdx,r8	
00007ff9`455033a5 743e 00007ff9`455033a7 4889442458 00007ff9`455033ac 488d157d860000	<pre>je amsilAmsiScanBuffer+0xd5 (00007ff9`455033c5) mov qword ptr [rsp+58h], rax lea rdx, [amsilCAmsiBufferStream::`vftable' (00007ff</pre>	F9`4550ba30 <mark>)</mark>	00007ff9`45502be5 488bf1 mov rsi,rcx	ptr [amsi!WPP_GLOBAL_0
00007ff9`455033b3 4889542440 00007ff9`455033b8 4533c9 00007ff9`455033bb 4889742448	mov qword ptr [rsp+40h], rdx xor r9d, r9d mov qword ptr [rsp+48h], rsi	Π		
00007ff9`455033c0 488d542440 00007ff9`455033c5 897c2450 00007ff9`455033c9 4c8bc5	lea rdx, [rsp+40h] mov dword ptr [rsp+50h], edi mov r8, rbp			
00007ff9`455033cc 4c897c2460 00007ff9`455033d1 4c89742468	mov qword ptr [rsp+60h], r15 mov qword ptr [rsp+68h], r14			

Above screenshot shows the virtual function table of CAmsiBufferStream.

Then the next address where we can find some more information regarding **amsiContext members** is,

 00007ff9455033d6
 488b01
 mov
 rax, qword ptr [rcx] ds:000002347eb5d120=

 {amsi!ATL::CComObject<CAmsiAntimalware>::vftable' (00007ff94550bb48)}

 00007ff9455033d9
 488b4018
 mov
 rax, qword ptr [rax+18h]

 00007ff9455033dd
 ff15cd8d0000
 call
 qword ptr [amsi!_guard_dispatch_icall_fptr (00007ff9`4550c1b0)]

in the above snippet, rcx holds one of those pointers we just discussed,

000002347eb5d120 (thirdMember). In the first instruction, 64 bit value at that address is loaded into rax register, which, according to the above snippet, is 00007ff94550bb48. It also specifies that this is a vtable located in .rodata section of the asmi.dll's memory image.

1		-								
	✓ 0x7ff945500000	Image	92 kB	WCX	C:\Windows\System32\amsi.dll	84	kВ	20 kB	64 kB	64 kB
	0x7ff945500	Image: Commit	4 kB	R	C:\Windows\System32\amsi.dll	4	kВ		4 kB	4 kB
	0x7ff945501	Image: Commit	40 kB	RX	C:\Windows\System32\amsi.dll	40	kВ	4 kB	36 kB	36 kB
	0x7ff94550b	Image: Commit	24 kB	R	C:\Windows\System32\amsi.dll	24	kВ	8 kB	16 kB	16 kB
	0x7ff945511	Image: Commit	8 kB	RW	C:\Windows\System32\amsi.dll	8	kВ	8 kB		
	0x7ff945513	Image: Commit	16 kB	R	C:\Windows\System32\amsi.dll	8	kВ		8 kB	8 kB

next two instructions retreives address **0x18** offset from the vtable into **rax** register and calls the address stored in **rax**

File		g 1.2111.9001.0											
- me	Home	View	Breakpoints	Time	e Travel Model	Scripting So	ource	Memory Comm	and				
	# 12	2 ab 🦛											
	Size Form	nat Text Memory											
)uery Query													
assem		proping (-	* * ×	$Command$ \times					
	@\$scopeip				✓ Follow current instruction			0:018> t					
	·	488b7c2458	mov r		d ptr [rsp+58h]			amsi!AmsiScanBuffe 00007ff9`455033dd	ff15cd8d0000	call	qword ptr	amsi!_guard_dispat	ch_icall
9007 f	9`455055f0	4883c430	add r	sp, 30h	per [rsp.com]		Ŀ	0:018> dq 00007ff9 00007ff9`4550bb48	04550bb48 00007ff9`455075	50 .0000	7ff9`455072a	A	
	9`455055f4 9`455055f6		pop r ret	-14				00007ff9`4550bb58	00007ff9`455075	70 0000	7 <mark>ff9`4</mark> 550560	00	
007 1	9`455055f7	cc	int 3					00007ff9`4550bb68 00007ff9`4550bb78	00007ff9`455059 00007ff9`4550de				
	9`455055f8 9`455055f9		int 3 int 3					00007ff9`4550bb88	00007ff9`455045	10 0000	7 <mark>ff9`4</mark> 550452	20	
0007 6 1	9`455055fa	cc	int 3					00007ff9`4550bb98 00007ff9`4550bba8					
	9`455055fb 9`455055fc		int 3 int 3					00007ff9`4550bbb8	00007ff9`455044	60 0000			
	9`455055fd		int 3					0:018> dq /c1 0000	7ff94550bb48 + 0	x18 L1			
007 1 1	9`455055fe	cc	int 3					00007ff9`4550bb60 0:018> t	00007779 455056				
	9`455055 ff	cc imalware::Scan:	int 3					amsi!CAmsiAntimalw					
007+1	9`45505600	4055		•bp				00007ff9`45505600	4055	push	rbp		
	9`45505602			bx									
	9`45505603 9`45505604			'si 'di									
907 f	9`45505605	4154	push r	12									
	9`45505607 9`45505609			•13 •14									
	9 4550560b			14									
		488d6c24f8	lea r	bp, [rsp-	8]								
		4881ec08010000 488b0548c10000	SUD r	sp, 108h ax gword	d ptr [amsi!_security_c	cookie (00007ff9)	45511						
007 f 1	9`45505620	4833c4	xor r	ax, rsp									
	9`45505623 9`45505627				[rbp-10h], rax								
		4c894c2478		bx, ebx	[rsp+78h], r9								
		4889542470			[rsp+70h], rdx								
	9`45505633												
1017				12, r9 di r8				0:018>					
907 f 1	9`45505636 9`45505639	498bf8 488bc2	mov r mov r	di, r8 ax, rdx				0:018>	_		Registers		• <i>4</i>
907 6 907 6	9`45505636 9`45505639 9`4550563c	498bf8 488bc2 488bf1	mov r mov r mov r	di, r8 ax, rdx si, rcx					•	\$ X	Registers	Value	▼ \$1
907 f 907f 907 f	9`45505636 9`45505639 9`4550563c 9`4550563c	498bf8 488bc2 488bf1 448bf3	mov r mov r mov r mov r	di, r8 ax, rdx si, rcx 14d, ebx				0:018>	•	\$ * X	Name	Value	- \$
07f 07f 07f 07f 07f	9 \ 45505636 9 \ 45505639 9 \ 45505639 9 \ 4550563c 9 \ 4550563f 9 \ 45505642 9 \ 45505645	498bf8 488bc2 488bf1 448bf3 4d85c0 750a	mov r mov r mov r test r jne au	di, r8 Sax, rdx Si, rcx 14d, ebx 8, r8 Masi!CAmsi	iAntimalware∷Scan+0x51	. (00007ff9`45505	651)	0 : 018> Memory Address: @rsp	▼ 455033E3 00007FF9			Value 0x00007ff945505600	- \$
07f 07f 07f 07f 07f 07f	9 45505636 9 45505639 9 4550563c 9 4550563c 9 4550563f 9 45505642 9 45505645 9 45505645	498bf8 488bc2 488bf1 448bf3 4485c0 750a 5857000780	mov r mov r mov r test r jne au mov e	di, r8 vax, rdx vsi, rcx v14d, ebx v8, r8 umsi!CAmsi vax, 80070	9057h			0:018> Memory Address: @rsp 0000006E*8298E9A8 4 0000006E*8298E9A8 4	455033E3 00007FF9 8F91D350 000001EC	A78D .	Name User		- 4
007f 007f 007f 007f 007f 007f	9 45505636 9 45505639 9 4550563c 9 4550563c 9 4550563f 9 45505642 9 45505645 9 45505645	498bf8 488bc2 488bf1 448bf3 4d85c0 750a b557000780 e9fe020000	mov r mov r mov r test r jne au mov e jmp au	di, r8 rax, rdx rsi, rcx 14d, ebx 8, r8 umsi!CAmsi rax, 80070 umsi!CAmsi				0:018> Memory Address: @rsp 0000006E*8298E988 4 0000006E*8298E988 4 0000006E*8298E988 4	455033E3 00007FF9	A78D 00001 FC04	Name User rax rbx rcx	0x00007ff945505600 0x000001eca791cdb0 0x000001ec8d94d120	~ \$
007f 007f 007f 007f 007f 007f 007f 007f	9 45505636 9 45505639 9 4550563c 9 4550563c 9 45505645 9 45505645 9 45505645 9 45505647 9 45505646 9 45505651 9 45505654	498bf8 488bc2 488bf1 448bf3 4485c0 750a b857000780 e9fe020000 418918 4485c9	mov r mov r mov r test r jnp au mov d test r	di, r8 rax, rdx rsi, rcx 14d, ebx 8, r8 umsi!CAmsi rax, 80070 umsi!CAmsi word ptr 9, r9	9057h iAntimalware::Scan+0x34 [r8], ebx	4f (00007ff9`4550	9594 f)	0:018> Memory Address: @rsp 0000006E*8298E9A8 0000006E*8298E9A8 0000006E*8298E9C8 0000006E*8298E9C8	455033E3 00007FF9 8F91D350 000001EC 00000003 00000000 00000000 00000000 00000000	A78D- 0000 FC04 8D7C- 4550	Name User rax rbx rcx rdx	0x00007ff945505600 0x000001eca791cdb0 0x000001ec8d94d120 0x000006e8298e9f0	- \$
007f1 007f1 007f1 007f1 007f1 007f1 007f1 007f1 007f1	9 45505636 9 45505639 9 45505636 9 4550563f 9 45505642 9 45505645 9 45505645 9 45505645 9 45505646 9 45505654 9 45505654 9 45505654	498bf8 488bf1 488bf1 448bf3 4485c0 750a b857000780 e9fe020000 418918 4485c9 7403	mov r mov r mov r test r jne au mov e jmp au mov d test r ie au	di, r8 cax, rdx si, rcx 14d, ebx 8, r8 msi!CAmsi cax, 80070 msi!CAmsi Word ptr 9, r9 msi!CAmsi	9057h iAntimalware::Scan+0x34 [r8], ebx iAntimalware::Scan+0x5c	4f (00007ff9`4550	9594 f)	0:018> Memory Address: @rsp 0000006E*8298E988 0000006E*8298E988 0000006E*8298E98 0000006E*8298E98 0000006E*8298E98 0000006E*8298E98 0000006E*8298E98 0000006E*8298E98	455033E3 00007FF9 8F91D350 0000000 00000003 00000000 00000000 00000000	A78D A 0000 FC04 8D7C 4550 0000	Name User rax rbx rcx rdx rsi	0x00007ff945505600 0x000001eca791cdb0 0x000001ec8d94d120 0x0000006e8298e9f0 0x000001ec90df48bc	- 4
007fi 007fi 007fi 007fi 007fi 007fi 007fi 007fi 007fi 007fi 007fi	9 45505636 9 45505635 9 45505635 9 45505635 9 45505642 9 45505642 9 45505647 9 45505647 9 45505645 9 45505655 9 45505655 9 45505655 9 45505655 9 45505655	498bf8 488bc2 488bf1 448bf3 448bf3 4485c0 750a b8570000780 e9fe020000 418918 4485c9 7403 498919 488bed9db90000	mov r mov r mov r mov r test r jne au mov e jmp a mov d test r je au mov q mov r	di, r8 ax, rdx si, rcx 14d, ebx 8, r8 msi!CAmsi ax, 80070 msi!CAmsi word ptr 9, r9 msi!CAmsi word ptr cx, qword	9057h LAntimalware::Scan+0x34 [r8], ebx LAntimalware::Scan+0x5c [r9], rbx d ptr [ams1!WPP_GLOBAL_	4f (00007ff9`4550 c (00007ff9`45505 _Control (00007ff	9594f) 665c) 79`455	0:e118> Memory Address: @rsp 0000006E*8298E9A8 0000006E*8298E9C8 0000006E*8298E9C8 0000006E*8298E9C8 0000006E*8298E9E8 0000006E*8298E9E8 0000006E*8298E9E8 0000006E*8298E9E8 0000006E*8298E9E8 0000006E*8298E9E8	455033E3 00007FF9 8F91D350 000001EC 00000003 00000000 00000000 00000000 00000000	A78D 0000 FC04 8D7C 4550 0000 8F75:	Name User rax rbx rcx rdx	0x00007ff945505600 0x000001eca791cdb0 0x000001ec8d94d120 0x000006e8298e9f0	- \$
07f 07f 07f 07f 07f 07f 07f 07f 07f 07f	9 45585636 9 4558539 9 4558539 9 4558563c 9 4558563c 9 45585642 9 45585645 9 45585647 9 4558564 9 45585654 9 45585657 9 45585655 9 45585655 9 45585655 9 45585655	498548 488552 488551 4485453 448556 758a 8956820808 418918 448559 7483 498919 4884596598088	mov r mov r mov r mov r test r jne au mov d test r je au mov d test r je au mov q mov r	di, r8 ax, rdx rsi, rcx 14d, ebx 8, r8 msi!CAmsi Assa 80070 msi!CAmsi word ptr 9, r9 msi!CAmsi word ptr cx, qword rdx, [amsi	9057h iAntimalware::Scan+0x34 [r8], ebx iAntimalware::Scan+0x5c [r9], rbx	4f (00007ff9`4550 c (00007ff9`45505 _Control (00007ff	9594f) 665c) 79`455	0:0188	455033E3 00007FF9 591D350 0000001EC 20000000 00000000 4D20EC82 00007FF9 200F48EC 000001EC 8D816F70 000001EC 20000002 00000000 00000020 00000000	A78D ▲ 00001 FC041 8D7C- 45501 00000 8F75: 8298 A7911	Name User rax rbx rcx rdx rsi rdi	0x00007ff945505600 0x000001eca791cdb0 0x000001ec8d94d120 0x0000006e8298e9f0 0x000001ec90df48bc	→ \$1
07f 07f 07f 07f 07f 07f 07f 07f 07f 07f	9 45505636 9 45505635 9 45505635 9 45505635 9 45505642 9 45505642 9 45505647 9 45505647 9 45505645 9 45505655 9 45505655 9 45505655 9 45505655 9 45505655	498bf8 488bc2 488bf1 448bf3 448bf3 4485c0 756a 69fe626060 418918 4485c9 7403 498919 488b6d9db90606 488b6d9db90606 488b6d9db90606 488b6d9db90606 488b6d9db90606 488b6d9db90606 488bc4	mov r mov r mov r mov r test r jne au mov e jmp au mov d test r je ar mov d test r jne au mov d test r jne au mov c mov c r mov r mov r r mov r mov r mov r mov r r r r r r r r r r r r r r r r r r r	di, r8 ax, rdx ssi, rcx 14d, ebx 8, r8 msi!CAmsi word ptr 9, r9 msi!CAmsi word ptr cx, qword dx, [amsi cx, rdx	9057h iAntimalware::Scan+0x34 [r8], ebx iAntimalware::Scan+0x5c [r9], rbx d ptr [amsilWPP_GLOBAL_ i!WPP_GLOBAL_Control (6	4f (00007ff9`4550 c (00007ff9`45505 _Control (00007ff 90007ff9`45511000	0594f) 665c) 69`455 9)]	0:018> Memory Address: @rsp 0000006E*8298E9A8 4 0000006E*8298E9A8 4 0000006E*8298E9A8 6 000006E*8298E9A8 6 000006E*8298E9A8 6 000006E*8298E9A8 6 000006F*8298E94E98 6 000006F*8298E98 6 000006F*8298E98 6	45503363 00007FF9 FF91D350 000001EC 00000003 00000000 00000000 00000000 00000020 000001EC 00000003 0000001EC 00000002 00000000 00000020 00000000 00000020 00000000	A78D ▲ 00001 FC041 8D7C- 45501 00000 8F755 8298 A7911 90DF	Name User rax rbx rcx rdx rsi rdi rsp rbp rip	0x00007ff945505600 0x000001eca791cdb0 0x000001ec8d94d120 0x0000006e8298e9f0 0x000001ec90df48bc 0x0000000000000020 0x000000000000020	- \$
07f 07f 07f 07f 07f 07f 07f 07f 07f 07f	9 45585636 9 45585639 9 45585632 9 45585632 9 45585642 9 45585642 9 45585645 9 45585645 9 45585645 9 4558565 9 4558565 9 4558565 9 4558565 9 4558566 9 4558566 9 4558566	498bf8 488bf1 448bf2 448bf3 448bf3 4485c0 750a b857000788 e9fe020000 418918 4485c9 7403 498919 488bc43db90000 488bc43db90000 488bc43db90000 488bc43db90000 488bc43db90000 488bc43db90000 488bc43db90000 488bc43db90000 488bc43db90000 488bc43db90000 488bc43db90000 488bc43db900000 488bc43db900000 488bc43db900000 488bc43db9000000 488bc43db9000000000000000000000000000000000000	mov r mov r mov r test r jne an mov e jmp an mov d test r je an mov r lea r cmp r je an test b	di, r8 ax, rdx ssi, rcx 14d, ebx 8, r8 umsi!CAmsi Ax, 80070 umsi!CAmsi Word ptr 9, r9 umsi!CAmsi word ptr cx, qword cx, [amsi cx, rdx umsi!CAmsi	9857h intimalware::Scan+0x34 [r8], ebx intimalware::Scan+0x5c [r9], rbx iptr [amsi1WPP_GLOBAL_ iWPP_GLOBAL_Control (6 intimalware::Scan+0x8c [rex+1Ch], 4	4f (00007ff9`4550 c (00007ff9`45505 _Control (00007ff 00007ff9`45511000 d (00007ff9`45505	9594f) 665c) 79`455 9)] 668d)	0:018-> Memory Address: @rsp 0000066: 8298598 0000066: 8298598 0000066: 8298598 0000066: 8298598 0000066: 8298598 0000066: 8298598 0000066: 8298598 0000066: 8298598 0000066: 8298598 0000066: 8298548 0000066: 8298548 0000066: 8298548 00000066: 82985484	455033E3 00007FF9 591D350 0000001EC 20000000 00000000 4D20EC82 00007FF9 200F48EC 000001EC 8D816F70 000001EC 20000002 00000000 00000020 00000000	A78D- 00000 FC04 8D7C- 4550 00000 8F75: 8298 A791 90DF- 0000	Name User rax rbx rcx rdx rsi rdi rsp rip efl	0x00007ff945505600 0x000001eca791cdb0 0x000001ec8d94d120 0x0000006e8298e9f0 0x00000000000000000 0x00000000000000	- \$
007f 007f 007f 007f 007f 007f 007f 007f	9 45585636 9 4558539 9 4558535 9 45585635 9 45585645 9 45585647 9 45585647 9 45585647 9 45585647 9 45585659 9 45585659 9 45585659 9 45585659 9 45585656 9 45585666 9 45585666 9 45585666	498b-f8 488b-f1 488b-f3 448b-f3 4485c6 558ee0788 e9f-e020000 418918 4485c9 7483 4485c9 7483 4485c9 7483 4485c9 7483 4485b040590000 4885b040590000 4885b040590000 4885b040590000 4885b040590000 4885b04059000 4885b04059000 4885b04059000 4885b04059000 4885b04059000 4885b04059000 4885b04059000 4885b04059000 4885b04059000 4885b04059000 4885b04059000 4885b04050000 4885b04050000000000000000000000000000000	mov r mov r mov r test r jne a mov e jmp a mov e test r je a mov r lea r je a test b je a	di, r8 vax, rdx vsi, rcx 14d, ebx 8, r8 msi!CAmsi. vax, 80070 msi!CAmsi. vsor p msi!CAmsi. vsor p r9 r9 r9 r9 r9 r9 r9 r9 r9 r9 r9 r9 r9	9857h lantimalware::Scan+0x34 [r8], ebx iAntimalware::Scan+0x56 [r9], rbx d ptr [amsiWPP_GLOBAL_ iMPP_GLOBAL_Control (6 iAntimalware::Scan+0x86 [rext1ch], 4 lantimalware:Scan+0x86 lantimalware::Scan+0x86	4f (00007ff9`4550 c (00007ff9`45505 _Control (00007ff 00007ff9`45511000 d (00007ff9`45505	9594f) 665c) 79`455 9)] 668d)	0:0185 Memory Address: @rsp 0000006E* 8298E988 0000006E* 8298E988 0000006E* 8298E988 0000006E* 8298E988 0000006E* 8298E98 0000006E* 8298E98 0000006E* 8298E98 0000006E* 8298EA88 0000006E* 8298EA88 0000006E* 8298EA88 0000006E* 8298EA88 0000006E* 8298EA88 0000006E* 8298EA88 0000006E* 8298EA88 0000006E* 8298EA88 0000006E* 8298EA88	455033E3 00007FF9 8F91D350 000001EC 00000003 00000000 00000000 0000001EC 8D01648C 000001EC 8D01648C 000001EC 8D01648C 000001EC 8D016470 0000001E 00000008 00000000 FB4F75A7 00007FF8 2598EB00 000000EE ED0E32C8 000000EE	A780 ▲ 00001 ► FC041 8D7C- 45501 00000 8F75 8298 A7911 900F- 00000 900F- 00000 715A	Name User rax rbx rcx rdx rsi rdi rsp rbp eff cs	0x00007ff945505600 0x000001eca791cdb0 0x000001ec399tcdb0 0x000006e3298e9f0 0x000000e0000000000 0x000000e000448bc 0x000000e2998e538 0x000007ff94550500 0x0000068298e538	- \$
007f 007f 007f 007f 007f 007f 007f 007f	99 4558636 99 45585639 99 45585637 99 45585637 99 45585645 99 45585645 99 45585645 99 45585645 99 45585659 99 45585659 99 45585659 99 45585656 99 45585656 99 45585667 99 45585667 99 45585667 99 45585667	498b-f8 488b-f1 488b-f3 448b-f3 448b-f3 448b-f3 448b-f6 52000 4820000 418918 448b-f3 448b-f3 4488b-f3 498919 4488b-f3 498919 4488b-f3 498b-f3	mov r mov r mov r mov r test r jne au mov e jimp a mov du test r mov du test r ie au mov q mov r test b je au mov r test b je au mov r	di, r8 ax, rdx si, rcx 14d, ebx 8, r8 msilCAmsi word ptr 9, r9 msilCAmsi word ptr rcx, qword ptr (cx, rdx msilCAmsi rcx, rdx msilCAmsi rcx, rdx	9857h intimalware::Scan+0x34 [r8], ebx intimalware::Scan+0x5c [r9], rbx iptr [amsi1WPP_GLOBAL_ iWPP_GLOBAL_Control (6 intimalware::Scan+0x8c [rex+1Ch], 4	4f (00007ff9`45505 c (00007ff9`45505 cControl (00007ff9`4551000 d (00007ff9`45505 d (00007ff9`45505	9594f) 665c) 691455 60] 668d) 668d)	0:0183	455933E3 00007FF9 8F91D350 000001E5 0000000 0000000 00000000 0000007F9 900F488C 000007F9 900F488C 000007F8 00000000 00000000 00000020 00000000 00000020 0000000 00000020 0000000 8239E800 000007F8 8239E800 00000FF8 8298E858 000000F5	A78D ▲ 0000 ▲ FC04 & 8D7C ← 4550 0000 & 8F75 & 8298 & A791 ↓ 90DF ↓ 0000 ↓ 715A & 8298 &	Name User rax rbx rcx rdx rsi rdi rsp rbp rip efl cs ds	0x00007ff945505600 0x000001eca791cdb0 0x000001ec8494d120 0x000001ec8494d120 0x000001ec90df48bc 0x00000062930e30 0x000006e8298eb38 0x000006e8298eb38 0x000006e8298eb38 0x000006e8298eb38 0x000006e8298eb38	- \$
007f1 007f1 007f1 007f1 007f1 007f1 007f1 007f1 007f1 007f1 007f1 007f1 007f1 007f1 007f1 007f1 007f1 007f1	9 4558636 9 4558639 9 4558639 9 4558632 9 4558632 9 4558642 9 45585642 9 45585645 9 4558565 9 4558565 9 4558565 9 4558565 9 4558565 9 45585663 9 45585667 9 45585667 9 4558567 9 4558567 9 45586567 9 45586567	4985-F8 4885-2 4885-2 4885-7 4885-7 485-7 550 485-69 7550 485-69 7463 4855-9 7463 4855-9 7463 4855-9 7463 4855-9 7463 4855-9 7463 4855-9 7463 4855-9 7463 4855-9 7463 4855-9 7463 4855-9 7463 4855-9 7463 4855-9 7463 4855-9 4955-9 4955-	mov r mov r mov r mov r test r jnp a mov c tip test r mov d tip test r mov q mov d tip test r test r nov r r test r nov r r test r nov c r r nov c r r nov c r r nov c r r r nov c r r nov c r r r nov c r r r nov c r r r nov c r r r nov c r r r nov c r r nov c r r r r nov c r r r r r nov c r r r r r r r r r r r r r r r r r r r	di, r8 ax, rdx si, rcx 14d, ebx 8, r8 msi!CAmsi word ptr 9, r9 msi!CAmsi word ptr cx, qword dx, [amsi cx, qword 8, [amsi! dx, [amsi! dx, [amsi] dx, [amsi] dx, [amsi] dx, [amsi]	9857h intimalware::Scan+0x34 [r8], ebx intimalware::Scan+0x50 [r9], rbx i ptr [amsi1WPP_GLOBAL_ i WPP_GLOBAL_Control (6 i MPD_GLOBAL_Control (6 intimalware::Scan+0x80 fcx+1ch], Scan+0x80 i ptr [rcx+1ch]	4f (00007ff9`45505 c (00007ff9`45505 cControl (00007ff9`4551000 d (00007ff9`45505 d (00007ff9`45505	9594f) 665c) 59`455 0)] 668d) 668d) fraceg	0:0183	455933E3 00007FF9 8F91D350 000001EC 00000003 00000000 4D20EC82 00007FF8 00004000 000000EC 8D816F70 000001EC 8D816F70 000000E0 0000008 00000000 0000008 000000E ED8C32C8 00007FF8 8298E580 00000EE ED8C32C8 00007FF8 4D9C4076 00007FF8	A78D ▲ 00000 FC04 8575 8298 A791 900F 0000 715A 8298 FA35 ¥298 FA35 ¥298	Name User rax rbx rcx rdi rsp rbp rip efl cs ds Activeste	0x00007ff945505600 0x000001eca791cdb0 0x000001ec84944120 0x0000001ec84944120 0x0000001ec904484 0x000000000000000000000 0x000001ec90448494440 0x0000006e8298eb58 0x00007e6459500 0x000006e8298eb58 0x00007e6459000000000000000000000000000000000000	1
0007f1 0007f1 0007f1 0007f1 0007f1 0007f1 0007f1 0007f1 0007f1 0007f1 0007f1 0007f1 0007f1 0007f1 0007f1 0007f1 0007f1	9 4558663 9 4558653 9 4558563 9 4558563 9 4558563 9 4558563 9 4558564 9 4558564 9 4558564 9 4558564 9 4558565 9 4558565 9 4558565 9 4558565 9 4558566 9 4558566 9 4558566 9 4558566 9 45585667 9 45585667 9 45585667 9 45585667 9 45585667	4985-F8 4885-2 4885-2 4885-7 4885-7 485-7 550 485-69 7550 485-69 7463 4855-9 7463 4855-9 7463 4855-9 7463 4855-9 7463 4855-9 7463 4855-9 7463 4855-9 7463 4855-9 7463 4855-9 7463 4855-9 7463 4855-9 7463 4855-9 7463 4855-9 4955-9 4955-	mov r mov r mov r mov r test r jnp a mov c tip test r mov d tip test r mov q mov d tip test r test r nov r r test r nov r r test r nov c r r nov c r r nov c r r nov c r r r nov c r r nov c r r r nov c r r r nov c r r r nov c r r r nov c r r r nov c r r nov c r r r r nov c r r r r r nov c r r r r r r r r r r r r r r r r r r r	di, r8 ax, rdx si, rcx 14d, ebx 8, r8 msi!CAmsi msi!CAmsi word ptr 9, r9 msi!CAmsi word ptr rcx, qword dx, [amsi rcx, rdx msi!CAmsi rcx, rdx si!CAmsi cx, qword fx, [amsi!	9857h intimalware::Scan+0x34 [r8], ebx intimalware::Scan+0x50 [r9], rbx i ptr [amsi1WPP_GLOBAL_ i WPP_GLOBAL_Control (6 i MPD_GLOBAL_Control (6 intimalware::Scan+0x80 fcx+1ch], Scan+0x80 i ptr [rcx+1ch]	4f (00007ff9`45505 c (00007ff9`45505 cControl (00007ff9`4551000 d (00007ff9`45505 d (00007ff9`45505	9594f) 665c) 691455 60] 668d) 668d)	0:0185 Memory Address: @rsp 0000006E* 8298E988 4 0000006E* 8298E988 0000006E* 8298E988 0000006E* 8298E988 0000006E* 8298E98 0000006E* 8298E98 0000006E* 8298E98 0000006E* 8298E98 0000006E* 8298E98 0000006E* 8298E78 0000006E* 8298EA8 0000006E* 8298EA8 0000006E* 8298EA8 0000006E* 8298EA88 0000006E* 8298EA88	455033E3 00007FF9 8F91D350 000001C 00000003 00000000 00000000 0000000 00000003 0000001C 8D01485C 000001C 8D01485C 000001C 8D0408C 0000000 00000020 00000000 00000020 00000000	A780 • 0000 FC04 8D7C- 4550 0000 8F75 8298 A791 900F- 0000 0000 715A 8298 FAAS •	Name User rax rbx rcx rdi rsp rbp rip efl cs ds Activeste	0x00007ff945505600 0x000001eca791cdb0 0x000001ec8494d120 0x000001ec8494d120 0x000001ec90df48bc 0x00000062930e30 0x000006e8298eb38 0x000006e8298eb38 0x000006e8298eb38 0x000006e8298eb38 0x000006e8298eb38	♥ 🖈

This proves that our assumption on function pointer extracted from the HAMSICONTEXT being a anti-malware vendor's registered function is false and it is a pointer to amsi!CAmsiAntimalware::Scan method.

We have uncovered some important details about **HAMSICONETXT** so far. We already know that the first member is a DWORD, and it should be equal to **0x49534d41** in order for scan to be successful. Third member is a pointer to an object of class **CAmsiAntimalware**, which has a virtual function called **amsi!CAmsiAntimalware::Scan**.

And by moving its 0x0 offset **rax** register, we can access it's virtual function table where we can find **Scan** at the 0x18.

The whole thing can be roughly decompiled down into below C code.

```
class CAmsiAntimalware {
        private:
            [...]
        public:
            virtual Scan(CAmsiBufferStream *, AMSI_RESULT, DWORD);
            [...]
    }
    typedef HAMSICONTEXT {
        QWORD
                             unk1;
        QWORD
                             *secondMember;
        CAmsiAntimalware
                             *antimalware;
        [...]
    };
    HRESULT ____stdcall AmsiScanBuffer
    (
            HAMSICONTEXT amsiContext,
            PVOID buffer,
            ULONG length,
            LPCWSTR contentName,
            HAMSISESSION amsiSession,
            AMSI_RESULT *result
    )
    {
        auto var;
        if ((WPP_GLOBAL_Control != &WPP_GLOBAL_Control) && (*(WPP_GLOBAL_Control +
0x1c)) != 4))
        {
            WPP_SF_qqDqq(
                *((BYTE*)WPP_GLOBAL_Control + 0x10),
                buffer,
                length,
                amsiContext,
                buffer,
                amsiSession,
                result
            );
        }
        if (
                buffer == NULL ||
                result == NULL ||
                amsiContext == NULL ||
                *((DWORD *)amsiContext) != 0x49534D41 ||
                *((QWORD *)amsiContext + 1) == 0x0 ||
                *((QWORD *)amsiContext+2) == 0x0
            )
```

```
{
        return 0x80070057;
    }
    else
    {
        CAmsiBufferStream bufferStream = CAmsiBufferStream(
            buffer,
            length,
            amsiContext->secondMember,
            contentName,
            session
        );
        return amsiContext->antimalware->Scan(
            amsiContext->antimalware, // this
            &bufferStream, // CAmsiBufferStream *
            result,
            0
        );
    }
}
```

We are not done yet. Goal here is to understand how AMSI works. Therefore, our next target is amsi!CAmsiAntimalware::Scan.

But before drill down into it, we need to construct the **HAMSICONTEXT** structure out of the knowlegde we have.

s Structure Edit	or - struct (ams	si.dll)									🕂 🕇 🖡 🇶 🖣	i 👫 📉 🗆	📰 💽 👻
Offset	Length	Mnemonio	ic	DataType	9	Nar	ne	Comm	ent				
j	8	QWORD		QWORD		unk	1	unknov	vn member				
3	8	QWORD *	c	QWORD *	k	sec	ondMember	pointer	to second m	iember objec	t		
16	8	QWORD *	¢	QWORD *	ĸ	cAn	nsiAntimalware	pointer	to an object	of class CAm	nsiAntimalware		
													-
													•
	22	22	21	20	10	10	17	16	15	14	12	12	
Byte Offset		22	21	20	19	18	17	16	15	14	13	12	·
Byte Offset		22	21			18	17	16	15		13 secondMember		•
Byte Offset		22	21		19 imalware	18	17	16	15				·
Byte Offset		22	21			18	17	16	15		secondMember		1
Byte Offset omponent Bits		22	21			18	17	16	15		secondMember		1
Byte Offset		22	21			18	17	16	15		secondMember align (mini	imum) 🗌 pe	1: ack
Byte Offset omponent Bits Name: AM	SICONTEXT		21			18	17	16	15		secondMember	imum) 🗌 pe	1
Byte Offset omponent Bits Name: AM vescription: am	SICONTEXT		21			18	17	16	15		secondMember align (mini	imum) 🗌 pe	1: ack
Byte Offset omponent Bits	SICONTEXT		21			18	17	16	15		secondMember align (mini	imum) 🗌 pə	1: ack

now we can see decompiler output is much more accurate and readable.



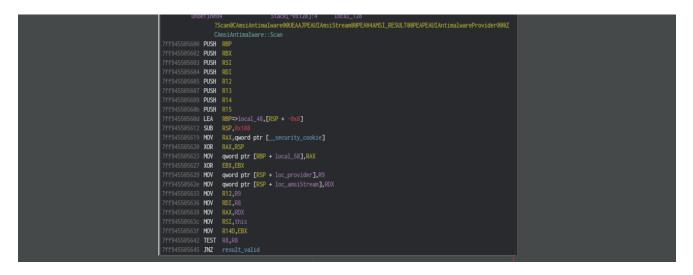
We can also try constructing a CAmsiAntimalware class but we dont have enough information to populate member variables.

CAmsiAntimalware::Scan

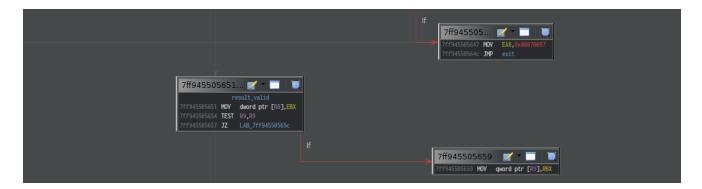
7ff945505600 - Scan		📈 🧐 🔲 📕	
long thiscall Scan(CAmsiAntima	alware * this. IAmsiSt	ream * RDX, AMSI_RESULT * R8, IAntimalwareProvider * * R9)	
long	EAX:4	<pre></pre>	
CAmsiAntimalware *	RCX:8 (auto)		
IAmsiStream *			
AMSI_RESULT *			
IAntimalwareProvide			
undefined8			
undefined8			
undefined8			
undefined1	Stack[-0x48]:1	local_48	
undefined8	Stack[-0x58]:8	local_58	
undefined8	Stack[-0x60]:8	local_60	
undefined8	Stack[-0x68]:8	local_68	
undefined8		local_70	
undefined8	Stack[-0x78]:8	local_78	
undefined8	Stack[-0x80]:8	local_80	
undefined8	Stack[-0x88]:8	local_88	
undefined8		local_90	
undefined8		local_98	
undefined8		local_a0	
undefined8		local_a8	
undefined1		local_c8	
undefined8		loc_provider	
undefined8			
undefined8			
undefined8			
undefined4			
undefined4			
undefined8			
undefined8			
undefined4			
undefined8			
undefined8			
undefined4			
?Scan@CAmsiAnt	imalware@@UEAAJPEAUIAn	siStream@@PEAW4AMSI_RESULT@@PEAPEAUIAntimalwareProvider@@@Z	

So ghidra has created a nice view of the stack frame for us. And by looking at the parameters, we see the function expects a pointer to an IAmsiBuffer object and a pointer to a pointer of IAntimalwareProvider object.

We saw that in the AmsiScanBuffer that this value is set to zero.



Then continues to setup all those memory curruption protection machanisms and to check the validity of the input parameters. First it checks if third parameter, **result** is null (remember, result is a pointer to AMSI_RESULT enum).



if it is not, it jumps to label result_valid . else, it sets eax to 0x80070057 and return. In the result_valid label, it sets *result to AMSI_RESULT_CLEAN (OxO). So it looks like the function is clearing the *result to not detected state. Which means we can expect value of result to change.

It also checks if **provider** is null. If not, it sets value of it to null and continue execution from LAB_7ff94550565c . else, it continues the execution from the same location but without setting ***provider** to null.

7ff94550565c - LAB_7ff94550565c U8_7ff945965c 1794559555c NOV 100 100 100 100 100 100 100 100 100 100 110		
17f14556563 IEA 80X, DPP_CIBAL_Control] 7f794556563 OP this,80X 17f94556566d JZ L48_7f794559558d	1- 7ff94550566f	
		H 7ff9455505675 7ff94550575 10V this,quord ptr [this + offset DAT_7ff945511010] 7ff94550575 HDV this,quord ptr [this + offset DAT_7ff945511010] 7ff945505505 HDV R0,R04 7ff945505555 HDV R0,R04 7ff945505555 HDV R0,R04 7ff945505558 HDV R0,R04 7ff945505558 HDV R0,R04

LAB_7ff94550565c does the same thing as AmsiScanBuffer did at the block 0x7ffxxxxx335d . However instead of calling WPP_SF_qqDqq it calls WPP_SF_q . Also note that above snippet sets rdx to either address of [WPP_GLOBAL_CONTROL] or 0x1e .

LAB_7ff94550568d looks interesting.

First it calls rand() function. In case you dont know, it's pretty common C library function and it generates a psuedo random number and return it. In the next line, it stores a member of CAmsiAntimalware class at offset 0x1c0 in r13 register. Then there are some multiplications around the generated value value.

ghidra being ghidra, has renamed registers with the variable names (this is good if we are doing x86 reversing becuase most of calling conventions pass parameters through stack, However, in our case, since parameters are passed through registers, renaming those can cause confusion), So to make it clear, we'll use listing view.

LAB_7ff	'94550568d	XREF[2]:	7ff94550566d(j), 7ff945505673(j)
7ff94550568d ff 15 5d 6a 00 <u>Q</u> 00	CALL qword ptr [->MSVCRT.DLL::rand]		
7ff945505693 4c 8b ae c0 01 M 00 00	NOV R13,qword ptr [RSI + 0x1c0]		this->0x1c0
7ff94550569a 4c 8b fb 🛛	IOV R15,RBX		
7ff94550569d 8b c8	10V this, <mark>EAX</mark>		
7ff94550569f b8 1f 85 eb 51	IOV EAX,0x51eb851f		
7ff9455056a4 f7 e1	UL this		
7ff9455056a6 c1 ea 05 S	SHR RDX, 0x5		
7ff9455056a9 6b c2 64 I	IMUL EAX, RDX, 0x64		
7ff9455056ac 2b c8 S	SUB this, EAX		
7ff9455056ae 89 4c 24 58	<pre>10V dword ptr [RSP + loc_rand], this</pre>		
7ff9455056b2 4d 85 ed T	TEST R13,R13		
7ff9455056b5 0f 84 09 02 00 J	Z LAB_7ff9455058c4		
00			

It assigns the return value from rand() to ecx register and loads eax with **0x51eb851f**. then it multiplies random value stored in ecx with the value loaded in eax. Note that this instruction is capable of changing the value at edx register.

Then there's a shift right instruction, which shifts 5 bits from edx register. then it multiplies shifted edx with 0x64 and stores the value in eax.

sub instruction substracts **eax**, by **ecx**. what this whole thing does is similar to below expression

```
rand() % 0x64;
```

value of ecx is then stored in a local variable loc_rand and function checks if r13, which holds the value of this->0x1c0 is 0/null. If yes, it jumps to LAB_7ff9455058c4. else, it continues exection from next address.

Now we got two control paths to follow. but first, I'm not gonna take the jump.

Control flow path 1



0x7ffxxxx56bb , address of this->0x40 gets loaded into r14 , which then gets stored in a local variable. Next instruction loads this->0xc0 into r12 register.

Then there's an unconditional jump and this one jumps directly into a loop. so Im gonna save that part for a debugging session and continue with the other control flow path.

Control flow path 2

7ff9455058c4 - LAB	
lf / Else	If / Else 7ff9455058cd

LAB_7ff9455058c4 starts with a comparison of r13 (this->Ox1c0 but as a local variable) with this->Ox1c0. The comparison checks if r13 is less than this->Ox1c0. if it is, control flow is directed to address Ox7ffxxxx58cd. else, control flow is directed to label LAB_7ff9455058f7.

First instruction at 0x7ffxxxx58cd sets r14 to zero (rbx is xored by itself at the begining of the function). Next two instructions checks if r12 is null.

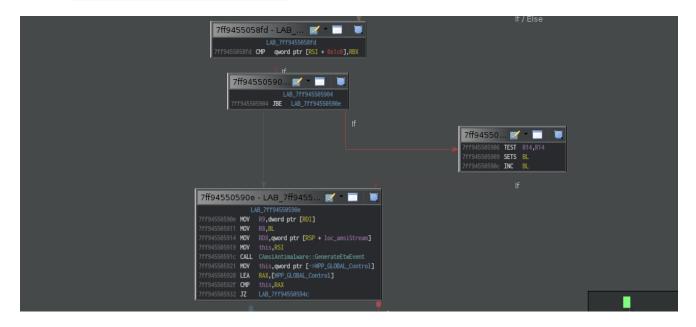


if not, value at address r12 is set to [RSI + r13*0x8 + 0x40]. Then it checks if rcx is null. If we assume the jump to LAB_7ff9455058c4 taken from 0x7ffxxxx56b5, then rcx would be the remainder of rand() % 0x64 thing. if rcx is null, jump is taken to label LAB_7ff9455058fd. else, it loads value at (*(rcx) + 0x8) to rax and calls it through _guard_dispatch_icall.

if r12 is null, jump is also taken to label LAB_7ff9455058fd .



on the other hand, LAB_7ff9455058f7 also jumps to LAB_7ff9455058fd after moving Ox1 into [rdi]. We already know that rdi is pointing to AMSI_RESULT enum. Constant 1 means AMSI_RESULT_NOT_DETECTED.



this simply checks if this->0x1c0 is null, if it is, it jumps to label LAB_7ff94550590e else, it continues exection from address 0x7ffxxxxx5906.

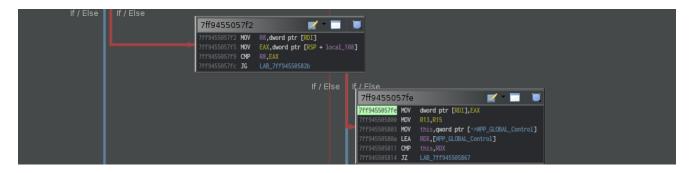
block starting at 0x7ffxxxx5906 basically checks if R14 is null. it sets b1 if previous comparison has caused sign flag to be 1. The operation may look like this in pseudocode.

bl = (r14 < 0) + 1;

as you can see in the above control flow graph, code is finally directed towards

LAB_7ff94550590e . What this snippet does is, call

CAmsiAntimalware::GenerateEtwEvent method. it passes this and amsiStream and bl through rcx, rdx and r9 registers as first three arguments. fourth and the last one is passed through r9 and this is basically the AMSI_RESULT. Now Im going to find where AMSI_RESULT is being modified. We already know rdi is a pointer to the enum.



In the above snippet, rdi (result) is assigned to value of eax . if we go up in the control flow, we can see eax is assigned with local_108.

Now we know some interesting places to place breakpoints and analyze, it is time to get into a windbg session.

First, Im gonna place a break point at address at place where **provider** is checked.

```
0:018> bp 0x7ffxxxxx5654
0.018> g
[...]
0.018> r r9
r9=000000000000000
```

As it is clear from the above snippet, **r9** register which holds a pointer to a pointer of **IAntimalwareProvider** class is set to zero. We saw this earlier in **AmsiScanBuffer** function.

Even if some value is passed down through this register, CAmsiAntimalware::Scan will set it to zero.

the next important piece for us is where this is being accessed.

Registers		Disassembly	▼ ☆ ×	Command × ₹
fs	0x0053	Address: @\$scopeip	Follow current instruction	0:019> dq @rsi+8x1c0 00000250`8945d640-00000000`00000001`00720250`a3385f10
gs	0x002b			00000259 37450650 0000006 0000000 0000000 00008000
SS	0x002b	00007fff`ae835679 4c8d05307c0000 lea	r8, [amsi!WPP_1e3aa342490e3db720910f85ce467dbb_Tracequids (00007f	00000250`8945d660 00000000`0000000 00006646`0675235c
r8	0x00000250a357b550	00007fff`ae835680 ba1e000000 mov	edx, 1Eh	00000250`8945d670 00000250`8945ea30 00000250`8945e940
r9	0x000000000000000000	00007fff`ae835685 4c8bc8 mov	г9, гах	00000250'8945d680 0000000'0000000 0000000'00000000
r10	0x000000000000011	00007fff`ae835688 e8cbd3ffff call		00000250`8945d690 00007fff`00000002 00007fff`d3c02e00
-11	0x00000250a356b8b0	00007fff`ae83568d ff155d6a0000 call	qwor <u>d otr [amsi! imo rand (89807rff</u> `ae83c8f8)]	00000250`8945d6a0 00007fff`d3d3c630 0000665d`02000002
rii		00007fff`ae835693 4c8baec0010000 mov	r13, gword ptr [rsi+1C0h]	00000250`8945d6b0 00000250`89457390 00000250`89450150
r12	0x000000000000000000	00007fff`ae83569a 4c8bfb mov	r15, rbx	0:019> t
r13	0x0000000000000001	00007fff`ae83569d 8bc8 mov	ecx, eax	amsi!CAmsiAntimalware::Scan+8x9a:
r14	0x0000000000000000	00007fff`ae83569f b81f85eb51 mov	eax, 51EB851Fh	00007fff`ae83569a 4c8bfb mov r15,rbx

above diagram shows exection has been stopped just after the instruction where function accessess this->0x1c0.

And the value at that address is set to 0x1. This gives us a hint that this member might be numerical value rather than a pointer.

Disassembly	✓ ☆ ×	$\operatorname{Command}$ $\overleftarrow{}$
Address: @\$scopeip	Follow current instruction	0:019> r rcx rcx=00000000002ea6
00007fff`ae83568d ff155d6a0000 call	qword ptr [amsi!_imp_rand (000007fff`ae83c0f0)]	
00007fff`ae835693 4c8baec0010000 mov 00007fff`ae83569a 4c8bfb mov	r13, qword ptr [rsi+1C0h] r15, rbx	
00007fff`ae83569d 8bc8 mov 00007fff`ae83569f b81f85eb51 mov 00007fff`ae8356a4 f7e1 mul	ecx, eax eax, 51EB851Fh	

A little below that, we can the random number generated by rand() being stored in ecx register and that value is 0x2ea6.

Since we already know what this snippet does, we can perform the calculation by ourself.

```
>>> hex(0x2ea6 % 0x64)
'0x2a'
```

brto ssp cetu eax	0x00000000000000 0x00000000000000 0x000000	00007fff`ae835693 4c8baec0010000 00007fff`ae83569a 4c8bfb 00007fff`ae83569a 4c8bfb 00007fff`ae83569f b81f85eb51 00007fff`ae8356a4 f7e1 00007fff`ae8356a6 c1ea05 00007fff`ae8356a6 c0c264 00007fff`ae8356ae 20c8	mov mov mov mul shr imul sub	r13, qword ptr [rsi+1C0h] r15, rbx ecx, eax eax, 51EB851Fh eax, ecx edx, 5 eax, edx, 64h ecx, eax dword ptr [rsn+58h]_ecx_ss:8000000311c9ce788=b80f54f8
edx ebx esp	0x000000000000000077 0x0000000000000000	00007fff`ae8356ae 894c2458 00007fff`ae8356b2 4d85ed 00007fff`ae8356b5 0f8409020000	mov test je	<pre>dword ptr [rsp+58h], ecx ss:00000093`1c9ce788=b80f54f8 r13, r13 amsi!CAmsiAntimalware::Scan+0x2c4 (00007fff`ae8358c4)</pre>

Above diagram conludes that.

-				odel		
-	🚬 🕪 🛛 🖂				Accent color	
mand	Watch Locals Registers Men	nory Stack D	isassembly Threads Breakpoints	s Logs I	lotes Timelines Modules Layouts Reset	
ers	- ☆ ×	Disassembl	ly .		+ ☆ ×	Command $ imes$
rbp	0x000000931c9ce830	A 44	of		S - Have a ware to a facet in a	0:019> dq @rsi+0x40 L1
rip	0x00007fffae8356bf	Address:	@\$scopeip		Follow current instruction	00000250`8945d4c0 00000250`a3302510 0:019> dq 00000250`a3302510
efl	0x00000202		ae835651 418918	mov	dword ptr [r8], ebx	00000250 a3302520 00007fff ae7e4318 006f0072 00000001
cs	0x0033		ae835654 4d85c9	test	r9, r9	00000250`a3302520 ffffffff fffffff 00000000`ffffffff
ds	0x002b		ae835657 7403 ae835659 498919	je mov	amsi!CAmsiAntimalware::Scan+0x5c (000007fff`ae83565c) gword ptr [r9], rbx	00000250`a3302530 0000000()`0000000 00000000`0000000 00000250`a3302540 00000000`020007d0 0053004d`005f0001
es	0x002b		ae83565c 488b0d9db90000	nov	rcx, qword ptr [amsiJ#PP_GLOBAL_Control (00007fff`ae841	00000250 a3302550 fffffff f fffffff 00000000 ffffffff
fs	0x0053		ae835663 488d1596b90000	lea	rdx, [amsi!WPP_GLOBAL_Control (00007fff`ae841000)]	00000250`a3302560 0000000`0000000 0000000`00000000
gs	0x002b		ae83566a 483bca	cmp	PCX, Pdx	00000250`a3302570 00000000`00000fa0 0000000`0000000
SS - O	0x002b 0x00000250a357b550		ae83566d 741e ae83566f f6411c04	je test	amsi!CAmsiAntimalware::Scan+0x8d (00007fff`ae83568d) byte ptr [rcx+1Ch], 4	00000250`a3302580 fffffffffffffff 00000000`ffffffff 0:019> u 00007fff`ae7e4318
r8 r9	0x00000250a3576550 0x0000000000000000		ae835673 7418	je	amsi!CAmsiAntimalware::Scan+0x8d (00007fff`ae83568d)	MpDav!DllRegisterServer+0x31b88:
r9 r10	0x000000000000000000000000000000000000	00007fff`	ae835675 488b4918	nov	rcx, gword ptr [rcx+10h]	00007fff`ae7e4318 7048 jo MpOav!DllRegisterSe
r10 r11	0x000000000000000000000000000000000000		ae835679_4c8d05307c0000	lea	r8, [amsi!WPP_1e3aa342490e3db720910f85ce467dbb_Tragegu:	00007fff`ae7e431a 7bae jnp MpDav!DllRegisterSe
r12	0x00000230a3368860	00007fff	ae935680 ba1e000000 ae835685 4c8bc8		edx, 1Eh r9, rax	00007fff`ae7e431c ff ???
r12 r13	0x000000000000000000000000000000000000		ae835688 e8cbd3ffff	call	ry, rax amsi!WPP_SF_q (000007fff`ae832a58)	00007fff`ae7e431d 7f00 jg MpOav!DllRegisterSe 00007fff`ae7e431f 00b0487baeff add byte ptr [rax-51848
r14	0x0000002508945d4c0		ae83568d ff155d6a0000	call	<pre>qword ptr [amsi!_imp_rand (00007fff`ae83cgf0)]</pre>	00007fff`ae7e4325 7f00 jg MpOav!DllRegisterSe
r15	0x000000200000000000000000000000000000		ae835693 4c8baec0010000	mov	r13, gword ptr [rsi+1C0h]	00007fff`ae7e4327 0080487baeff add byte ptr [rax-51848
dr0	0x000000000000000000000000000000000000		ae83569a 4c8bfb ae83569d 8bc8	mov	r15, rbx	00007fff`ae7e432d 7f00 jg MpOav!DllRegisterSe
dr1	0x000000000000000000000000000000000000		ae83569f b81f85eb51		ecx, eax eax, 51EB851Fh	
dr2	0x000000000000000000000000000000000000		ae8356a4 f7e1	mul	eax, ecx	
dr3	0x000000000000000000000000000000000000		ae8356a6 c1ea05	shr	edx, 5	
dr6	0x000000000000000000000000000000000000		ae8356a9 6bc264	imul	eax, edx, 64h	
dr7	0x00000000000000000000		ae8356ac 2bc8 ae8356ae 894c2458	sub mov	ecx, eax dword ptr [rsp+58h], ecx	
exfrc	0x00000000000000000		ae8356b2 4d85ed	test	r13, r13	
exto	0x00000000000000000		ae8356b5 0f8409020000	je	amsilCAmsiAntimalware::Scan+0x2c4 (000007fff`ae8358c4)	
brfrc	0x000000000000000000		ae8356bb 4c8d7640	lea	r14, [rsi+40h]	
brto	0x000000000000000000	000077777	ae8356bf 4c89742460 ae8356c4 4c8da6c0000000	lea	<pre>qword ptr [rsp+60h], r14 ss:00000093`1c9ce790=000000000 r12, [rsi+0C0h]</pre>	
ssp	0x00000000000000000		ae8356cb eb85	jmp	amsi!CAmsiAntimalware::Scan+0xd2 (00007fff`ae8356d2)	
cetu	0x0000000000000000		ae8356cd 4c8b742460	mov	r14, gword ptr [rsp+60h]	
eax	0x000000000002e7c		ae8356d2 488d4c2448	lea	rcx, [rsp+48h]	0:019>
ecx	0x00000000000002a		ae8356d7 895c2440 ae8356db 48895c2448		dword ptr [rsp+40h], ebx qword ptr [rsp+48h], rbx	Memory 🔻 🤋
edx	0x000000000000077		ae8356e0 ff15f2680000	call	<pre>gword ptr [amsi!_imp_GetSystemTimePreciseAsFileTime (00)</pre>	inchory 2
ebx	0x000000000000000		ae8356e6 498b0e	mov	rcx, gword ptr [r14]	Address: @rsp
esp	0x00000001c9ce730		ae8356e9 4c8d442448	lea	r8, [rsp+48h]	
ebp	0x00000001c9ce830		ae8356ee 488b542470 ae8356f3 488b01		rdx, qword ptr [rsp+70h] rax, gword ptr [rcx]	00000093 1C9CE730 B80F54E8 00007FFF 00002000 00000000
esi	0x00000008945d480		ae8356f6 488b4018	mov	rax, qword ptr [rax+18h]	00000093`1C9CE740 00000000 00000000 8B1B00000 00000250 00000093`1C9CE750 00000007 00000000 00000000 00000000
edi	0x000000001c9cea28	00007fff`	ae8356fa ff15b06a0000	call	<pre>qword ptr [amsi!_guard_dispatch_icall_fptr (00007fff`ac</pre>	00000093 1C9CE760 00000000 00000000 00000000 00000000
r8d	0x00000000a357b550		ae835700 488d4c2450	lea	rcx, [rsp+50h]	00000093 1C9CE770 00000000 FFFFFFF 00000000 00000000
r9d	0x000000000000000		ae835705 48895c2450 ae83570a 448bf0	EOV EOV	qword ptr [rsp+50h], rbx r14d, eax	00000093`1C9CE780 8B1B0000 00000250 B80F54F8 00007FFF
r10d	0x00000000000011		ae83570d ff15c5680000	call	qword ptr [amsi!_imp_GetSystemTimePreciseAsFileTime (00	00000093`1C9CE790 00000000 0000000 00000000 0000000
r11d	0x0000000a356b8b0	00007fff`	ae835713 4c8b4c2450	mov	r9, gword ptr [rsp+58h]	00000093`1C9CE7A0 1C9CE8C0 00000093 00000000 00000000
r12d	0x000000000000000		ae835718 4c2b4c2448	sub	r9, gword ptr [rsp+48h]	00000093`1C9CE7B0 00000000 00000000 00000018 00000000 00000093`1C9CE7C0 B80F5020 00007FFF 00000000 00000000
r13d	0x000000000000000000000000000000000000		ae83571d 4c894c2468 ae835722 395c2458	IIIOV CIIID	qword ptr [rsp+68h], r9 dword ptr [rsp+58h], ebx	00000093 1C9CE7D0 00000058 00000000 A364E2E8 00000250
r14d	0x00000008945d4c0		ae835726 0f85c1000000	jne	amsi!CAmsiAntimalware::Scan+0x1ed (00007fff`ae8357ed)	00000093 1C9CE7E0 6BA3D060 00007FFF 00000002 00000000
r15d	0x0000000000000000	00007fff`	ae83572c 833dddb8000005	cmp	dword ptr [amsi!WPP_GLOBAL_Control+0x10 (00007fff`ae841	00000093`1C9CE7F0 00000058 00000000 B7738E01 00007FFF
eip	0x00000000ae8356bf		ae835733 0f86b4000000	jbe	amsi!CAmsiAntimalware::Scan+0x1ed (00007fff`ae8357ed)	00000093`1C9CE800 8C85DFA0 00000250 A364E2E8 00000250
ах	0x000000000002e7c		ae835739 48ba0000000000400000 ae835743 488515d6b80000	est	rdx, 400000000000h gword ptr [amsi!WPP_GLOBAL_Control+0x20 (00007fff`ae841	00000093`1C9CE810 /0000000 0000000 0000000 00000000 0000000
CX	0x00000000000002a		ae83574a 7415	test je	dword ptr [ams1!WPP_GLUBAL_LONTPOL+0X20 (00007fff ae841 ams1!CAmsiAntimalware::Scan+0x161 (00007fff ae835761)	Go to Settings to activate Windows.
dx	0x000000000000077 0x0000000000000000000					Go to Settings to activate Windows. Watch Stack Memory
bx						

Above diagram shows where the function retreives address of this->0×40 into r14 register.

When this->0×40 is printed, it also looks like an address that pointed at heap.

Value at *this->0×40 looks like a function pointer and when disasseble that address, windbg prints disassembly of MpOav!DllRegisterServer (another dll ? we'll see)but disassembly starts from the middle of the function. This might not be a function pointer after all.

Disassembly		• \$	x	Command \times			Ŧ
Address: @\$scopeip		Follow current instruction		0:019> t amsi!CAmsiAntimalwar			
00007fff`ae8356b5 0f8409020000 00007fff`ae8356bb 4c8d7640	je lea	<pre>amsilCAmsiAntimalware::Scan+8x2c4 (00007fff`ae8358c4) r14, [rsi+40h]</pre>		00007fff`ae8356cb et 0:019> dd @rsi+0xc0	J -P		ware::Sc
00007fff`ae8356bf 4c89742460 00007fff`ae8356bf 4c89742460 00007fff`ae8356c4 4c8da6c0000000	nov	qword ptr [rsp+68h], r14		00000250`8945d540 00000250`8945d550	0000000 0000000 (0000000 0000000	
00007fff`ae8356cb eb05	lea jmp	<pre>r12, [rsi+8C0h] amsi!CAnsiAntimalware::Scan+8xd2 (808087fff'ae8356d2) amsi!CAnsiAntimalware::Scan+8xd2 (88887ff'ae8356d2)</pre>		00000250`8945d560 0 00000250`8945d570 0	0000000 00000000	0000000 0000000	
00007fff`ae8356cd 4c8b742460 00007fff`ae8356d2 488d4c2448	mov lea	r14, qword ptr [rsp+60h] rcx, [rsp+48h]		00000250`8945d580 0 00000250`8945d590 0	0000000 0000000 (0000000 0000000	
00007fff`ae8356d7 895c2440 00007fff`ae8356db 48895c2448	mov mov	dword ptr [rsp+40h], ebx qword ptr [rsp+48h], rbx		00000250`8945d5a0 0 00000250`8945d5b0 0			

here is another place where a member of CAmsiAntimalware class has been referenced. this time as we've discussed when doing static analysis, stores address this->0xc0.

It doesn't provide us with imformation about type of data even if we take a look at the data at that address,

Control flow path 1 continued

Now we are at the instruction in disassembly where that loop begins.

00007fffae8356d2	488d4c2448	lea	rcx, [rsp+48h]
00007fffae8356d7	895c2440	mov	dword ptr [rsp+40h], ebx
00007fffae8356db	48895c2448	mov	qword ptr [rsp+48h], rbx
00007fffae8356e0	ff15f2680000	call	qword ptr
[amsi!_imp_GetSy	stemTimePreciseAsFile	Time (00	007fff`ae83bfd8)]

We see that in the above image, first instruction loads address of rsp+0x48 into rcx register and calls GetSystemTimePreciseAsFileTime, which is used to retrieve the current system date and time with the highest possible level of precision in UTC format.

before the call instruction it also initialize rsp+0x40 and rsp+0x48 with oxo.

Then value at address r14 gets stored in rcx register. if you remember, r14 register stores &this->0x40 so rcx would be value of this->0x40.

Then can see some manipulations around that value.

sters lame User rax Oxt rbx Oxt rcx Oxt rsp Oxt rsp Oxt rip Oxt efl Oxt es Oxt ds Oxt fs Oxt fs Oxt		Disassemb Address: 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff	Disassembly Threads Breakpoin Windows	je lea mov lea mov lea mov call mov lea mov	destilCAssiAntimalware:: r14, [rs:40h] qword ptr [rsp+60h], r14 r12, [rs:60ch] amsilCAssiAntimalware:: r14, qword ptr [rsp+60h], abr qword ptr [rsp+60h], abr res, quand ptr [r14] r8, [rsp+40h]	4 Scan+8xd2 (88887fff'a 	s\$356d2)		immand X 3:819 > r p14 1 1:4=8000825689X544c0 3:819 > dg @r14 L1 3:819 > ug @rcx L1 080087569 %264530 08008256 9325240 080087fff ac72c 080082759 %264530 108087fff ac72c 9:819 > ug @rcx L1 08087fff ac72c 9:819 > ug @rcx 13 98087fff ac72c 9:819 > ug @rcx 45330 08087fff ac72c 9:819 > ug @rcx 45330 98087fff ac72c 9:8097fff ac7337f5 (48895c2488 98087fff ac72c 90807fff ac7337f5 (48895c2481 98087ff ac72c	4318	qword ptr [rsp+8] qword ptr [rsp+10
sters ame User rax Oxt rbx Oxt rcx Oxt rsp Oxt rsp Oxt eff Oxt cs Oxt ds Oxt es Oxt fs Oxt		Disassemb Address: 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff	Windows thy @\$scopeip ae8356b5 0f8409020000 ae8356b4 c68742400 ae8356b4 c68742400 ae8356c4 c685742400 ae8356c4 c685742400 ae8356c4 c685742400 ae8356c4 c685742400 ae8356c4 c685742400 ae8356c4 c685742400 ae8356c4 f5762400 ae8356c4 c685742400 ae8356c4 c6442440 ae8356c4 c6442440 ae8356c4 c6442440 ae8356c4 c6442440 ae8356c4 c6442440 ae8356c4 c6442440 ae8356c4 c68051	je lea mov lea mov lea mov call mov lea mov	destilCAssiAntimalware:: r14, [rs:40h] qword ptr [rsp+60h], r14 r12, [rs:60ch] amsilCAssiAntimalware:: r14, qword ptr [rsp+60h], abr qword ptr [rsp+60h], abr res, quand ptr [r14] r8, [rsp+40h]	Windows Window Layout Window Layout urrent instruction Scan+8x2c4 (00007fff) Scan+8x42 (00007fff)	 		3:815 > r14 14-80002589454448 3:015 > dq [r14 1] 90002559 8945464 90000256 8336; 9115 > uq [rcx 1] 90000250 8332231 90007fff e768 3:015 > dq 377ffe7674338 1] 900007fff e764338 00007fff e768 9:015 > u [rex 9:015 > u [rex 9:015 > u [rex 9:0007ff e76757f 488952488	4318 37f0 mov	
ame User rax Oxt rbx Oxt rcx Oxt rdx Oxt rdx Oxt rdx Oxt rsp Oxt rsp Oxt rsp Oxt eff Oxt eff Oxt es Oxt es Oxt fs Oxt	Value	 Address: B0007fff B0007ffff	by @\$scopeip a=835665 6f8409020000 a=835664 6847540 a=835664 4c847540 a=835664 4c80742460 a=835664 4c80742460 a=835664 4c80742460 a=835664 4595c2448 a=835664 4595c2448 a=835664 4595c2448 a=835664 4595c2448 a=835664 4c842440 a=835664 4c842440 a=835664 4c842440 a=835664 488061	lea mov lea jmp mov lea mov call mov lea mov	amsilCAmsiAntimalware:: r14, [rsi+40h] gword ptr [rsp+60h], r14 r12, [rsi+80Ch] amsilCAmsiAntimalware:: r14, gword ptr [rsp+60h] r05, [rsp+40h], ebs gword ptr [rsp+40h], ebs gword ptr [rsp+40h], ebs gword ptr [rsp+40h], ebs r05, [rsp+40h]	urrent instruction Scan+8x2c4 (00007fff [*] 4 Scan+8xd2 (00007fff [*]) 1	 		3:815 > r14 14-80002589454448 3:015 > dq [r14 1] 90002559 8945464 90000256 8336; 9115 > uq [rcx 1] 90000250 8332231 90007fff e768 3:015 > dq 377ffe7674338 1] 900007fff e764338 00007fff e768 9:015 > u [rex 9:015 > u [rex 9:015 > u [rex 9:0007ff e76757f 488952488	4318 37f0 mov	
ame User rax Oxt rbx Oxt rcx Oxt rcdx Oxt rsi Oxt rsi Oxt rsp Oxt rsp Oxt efl Oxt efl Oxt es Oxt es Oxt fs Oxt	Value	 Address: B0007fff B0007ffff	✓ £\$ scopeip ae8356b5 8f84090208080 ae8356b4 4c807448 ae8356b4 4c80742460 ae8356c4 4c8042460 ae8356c4 4c8042460 ae8356c4 4c8042448 ae8356c4 4c8052448 ae8356c4 4c8052448 ae8356c4 4c8052448 ae8356c4 4c8042440 ae8356c6 4c8042440 ae8356c7 4c8054240 ae8356c7 4c8054240 ae835667 4c8054240 ae835667 4c8054240 ae835667 4c8054240 ae835667 4c8054240 ae835667 4c8054240 ae835667 4c8054240 ae83567 4c80567 4c8057 ae83567 4c8057 ae8	lea mov lea jmp mov lea mov call mov lea mov	amsilCAmsiAntimalware:: r14, [rsi+40h] gword ptr [rsp+60h], r14 r12, [rsi+80Ch] amsilCAmsiAntimalware:: r14, gword ptr [rsp+60h] r05, [rsp+40h], ebs gword ptr [rsp+40h], ebs gword ptr [rsp+40h], ebs gword ptr [rsp+40h], ebs r05, [rsp+40h]	urrent instruction Scan+8x2c4 (00007fff [*] 4 Scan+8xd2 (00007fff [*]) 1	ae8358c4) [3:815 > r14 14-80002589454448 3:015 > dq [r14 1] 90002559 8945464 90000256 8336; 9115 > uq [rcx 1] 90000250 8332231 90007fff e768 3:015 > dq 377ffe7674338 1] 900007fff e764338 00007fff e768 9:015 > u [rex 9:015 > u [rex 9:015 > u [rex 9:0007ff e76757f 488952488	4318 37f0 mov	
ame User rax Oxt rbx Oxt rcx Oxt rdx Oxt rsi Oxt rsi Oxt rsp Oxt rsp Oxt efl Oxt efl Oxt es Oxt es Oxt fs Oxt	Value	 Address: B0007fff B0007ffff	✓ £\$ scopeip ae8356b5 8f84090208080 ae8356b4 4c807448 ae8356b4 4c80742460 ae8356c4 4c8042460 ae8356c4 4c8042460 ae8356c4 4c8042448 ae8356c4 4c8052448 ae8356c4 4c8052448 ae8356c4 4c8052448 ae8356c4 4c8042440 ae8356c6 4c8042440 ae8356c7 4c8054240 ae8356c7 4c8054240 ae835667 4c8054240 ae835667 4c8054240 ae835667 4c8054240 ae835667 4c8054240 ae835667 4c8054240 ae835667 4c8054240 ae83567 4c80567 4c8057 ae83567 4c8057 ae8	lea mov lea jmp mov lea mov call mov lea mov	amsilCAmsiAntimalware:: r14, [rsi+40h] gword ptr [rsp+60h], r14 r12, [rsi+80Ch] amsilCAmsiAntimalware:: r14, gword ptr [rsp+60h] r05, [rsp+40h], ebs gword ptr [rsp+40h], ebs gword ptr [rsp+40h], ebs gword ptr [rsp+40h], ebs r05, [rsp+40h]	Scan+8x2c4 (88887fff) 4 Scan+8xd2 (88887fff) 3	ae8358c4) [3:815 > r14 14-80002589454448 3:015 > dq [r14 1] 90002559 8945444] 90000250 8945444] 90000250 83332231] 900007ff ac74538] 900007ff ac74538] 900007ff ac74538] 900007ff ac745374 (s89552488	4318 37f0 mov	
User rax 0xt rbx 0xt rcx 0xt rdx 0xt rdx 0xt rsi 0xt rsp 0xt rip 0xt efl 0xt cs 0xt ds 0xt fs 0xt	x00007fffae7b37f0 x00000250a3302510 x000000250a3302510 x000000931c9ce820 x000000931c9ce820 x000000931c9ce820 x00000931c9ce830 x00000931c9ce830 x00000931c9ce830 x00000931c9ce830 x00000931c9ce830 x00000931c9ce830 x00000931c9ce830 x00000931c9ce830 x00000931c9ce830 x00000931c9ce830 x00000931c9ce830 x00000931c9ce830 x00000 x00000931c9ce830 x00000 x00000931c9ce830 x00000 x00000931c9ce830 x00000 x00000931c9ce830 x000000000000000 x00000000000000000	Address: 00007fff		lea mov lea jmp mov lea mov call mov lea mov	amsilCAmsiAntimalware:: r14, [rsi+40h] gword ptr [rsp+60h], r14 r12, [rsi+80Ch] amsilCAmsiAntimalware:: r14, gword ptr [rsp+60h] r05, [rsp+40h], ebs gword ptr [rsp+40h], ebs gword ptr [rsp+40h], ebs gword ptr [rsp+40h], ebs r05, [rsp+40h]	Scan+8x2c4 (88887fff) 4 Scan+8xd2 (88887fff) 3	s\$356d2)		114-80000250894544c8 3:819- dq @r14 L1 30000250 8445-cc0 0000250 a338: 3:819- dq @rx L1 3:819- dq @rx L1 3:819- dq @rx7fffac7e4338 L1 300027fff ac7e4338 00007fff ac7b: 5019- u @rax fp0av101LRegisterServer+0x10606: 30007fff ac7b37f6 48895c2488	4318 37f0 mov	
rax 0xx rbx 0xx rcx 0xx rdx 0xx rdi 0xx rsp 0xx rbp 0xx efl 0xx efl 0xx ds 0xx fs 0xx	x0000000000000000 x00000250a3002510 x00000093145ce8c0 x00000093145ce820 x00000093145ce730 x0000093145ce730 x0000093145ce330 x000007ffac8356fa x000000785 x0025 x0025 x0025 x0025	00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff	"ee856bb 4c807440 "ae856bc 4c8074248 "ae856bc 4c804c800000 "ae856bc be55 "ae856bc 4c804c20408 "ae856bc 4c80742448 "ae856bc 45572448 "ae856bc 45572448 "ae856bc 45572448 "ae856bc 4590be ae8556e 4590be "ae8556e 4280bc "ae8556e 4880b1	lea mov lea jmp mov lea mov call mov lea mov	r14, [rsi+40h] qword ptr [rsp+68h], r14 r12, [rsi+800h] r14, qword ptr [rsp+68h] r04, [rsp+48h] dword ptr [rsp+48h], chr qword ptr [rsp+48h], chr qword ptr [rsp+48h], chr qword ptr [rsp+48h], chr qword ptr [rsp+48h] r05, qword ptr [r14] r05, [rsp+48h]	4 Scan+8xd2 (88887fff'a 	s\$356d2)		00000250'80454600" 00000250'8330 3:019- uq @nct L 00000250'8302210 00007fff'8778 3:019- ug 0X7ff68784330 L 00007fff'8778430 00007fff'87878 9:019- u @nax 90007fff 8270370 4889502408	4318 37f0 mov	
rbx 0xt rcx 0xt rdx 0xt rsi 0xt rdi 0xt rsp 0xt rbp 0xt efl 0xt cs 0xt ds 0xt fs 0xt	x0000000000000000 x00000250a3002510 x00000093145ce8c0 x00000093145ce820 x00000093145ce730 x0000093145ce730 x0000093145ce330 x000007ffac8356fa x000000785 x0025 x0025 x0025 x0025	00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff	"ee856bb 4c807440 "ae856bc 4c8074248 "ae856bc 4c804c800000 "ae856bc be55 "ae856bc 4c804c20408 "ae856bc 4c80742448 "ae856bc 45572448 "ae856bc 45572448 "ae856bc 45572448 "ae856bc 4590be ae8556e 4590be "ae8556e 4280bc "ae8556e 4880b1	lea mov lea jmp mov lea mov call mov lea mov	r14, [rsi+40h] qword ptr [rsp+68h], r14 r12, [rsi+800h] amsilCAnsiAntimalmare:: r14, qword ptr [rsp+68h] r0%, [rsp+48h] dword ptr [rsp+48h], chr qword ptr [rsp+48h], chr qword ptr [rsp+48h], chr qword ptr [rsp+48h] r0%, qword ptr [r14] r8%, [rsp+48h]	4 Scan+8xd2 (88887fff'a 	s\$356d2)		0:019- uq @rcx L1 00000250 a3302300 00007fff`ae7e4 5:019- gd.@X7fffae7e4330 L1 300027fff ae7e4330 00007fff`ae7b3 3:019- u @rax 4p0av1DllRegisterServer+0x1060: 30007fff ae7b37f0 48895c2408	4318 37f0 mov	
rcx 0xt rdx 0xt rsi 0xt rsp 0xt rsp 0xt rsp 0xt efl 0xt cs 0xt es 0xt fs 0xt	x00000250a3302510 x000000931c9ce8c0 x000002508456440 x00000931c9ce730 x00000931c9ce730 x00000931c9ce830 x00000931c9ce830 x00000931c9ce830 x00000020 x000002 x000002 x00002 x00002 x00025 x0025 x0053	00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff	"ae8356c4 4c8da6c000000 "ae8356cb e005 "ae8356cb e005 "ae8356cd 4c80742468 "ae8356cd 7895c2448 "ae8356cd 7895c2448 "ae8356cb 47895c2448 "ae8356cb 47950be "ae8356cb 47950be "ae8356cb 47950be "ae8356cb 47870 "ae8356cb 488051	lea jmp mov lea mov call mov lea mov	r12, [rsi+0C0h] amsilCAmsiAntimalware::3 14, qword ptr [rsp+60h] rex, [rsp+40h] dword ptr [rsp+40h], sbo gword ptr [rsp+40h], sbo gword ptr [rsp+40h], rbo mord, ptr [amsil_imp_Bet rex, qword ptr [r14] r8, [rsp+40h]	Scan+8xd2 (80807fff'a] K			90000250°a3392210° 00007fff°ac7e 3:019> dg üx7fffac7e4336 L1 90007fff ac7e4330 00007fff°ac7b 3:019> ∪ Grax 4p0av!DllRegisterServer+0x1060° 90007fff ac7b37f0 48895c2408	37f0	
rdx 0x4 rsi 0x4 rdi 0x4 rsp 0x4 rbp 0x4 rip 0x4 efl 0x4 cs 0x4 es 0x4 fs 0x4	x00000931c9ce8c0 x0000025089454480 x000000931c9ce328 x000000931c9ce730 x000000931c9ce830 x00007fffae8356fa x00007fffae8356fa x000072b x0033 x002b x002b x002b x002b x0053	00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff	"ae8356cb eb05 "ae8356cd 4c8b742468 ae8356cd 4c8b742468 "ae8356d1 895c2440 "ae8356d0 #515r2440 "ae8356d0 #515r2480000 "ae8356c9 4c8442440 "ae8356c9 4c8442440 "ae8356c9 4c8442440 "ae8356c9 4c8442440	jmp mov lea mov call mov lea mov	amsilCAmsiAntimalware:: 14, qword ptr [rsp+66h] rcx, [rsp+48h] dword ptr [rsp+46h], cbr qword ptr [rsp:46h], rbr qword ptr [amsil_imp_6et rcx, qword ptr [r14] r6, [rsp+46h]				00027fff`ae7e4330 00007fff`ae7b 3:019> u @rax MpOav!DllRegisterServer+0x1060: 00007fff`ae7b37f0 48895c2408	nov	
rdi Oxt rsp Oxt rbp Oxt rip Oxt efl Oxt cs Oxt ds Oxt es Oxt fs Oxt	x000000931c9cea28 x000000931c9cea28 x000000931c9ce830 x000007ffac9566 x00007ffac9566 x0000202 x0033 x002b x002b x002b x002b x002b x002b	00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff	ae8356cd 408742460 ae8356d7 48846c2448 ae8356d7 895c2440 ae8356d7 895c2440 ae8356d7 4895c2448 ae8356c6 478060 ae8356c6 478042440 ae8356c6 4880424740 ae8356c6 488041	mov lea mov call mov lea mov	r14, qword ptr [rsp+68h] rox, [rsp+48h] dword ptr [rsp+46h], sho qword ptr [rsp+46h], rbo gword ptr [rss+48h], rbo gword ptr [rsi], rbo rox, qword ptr [r14] r8, [rsp+46h]				3:019> u @rax MpOav!DllRegisterServer+0x1060: 30007fff`ae7b37f0 48895c2408	nov	
rsp Oxt rbp Oxt rip Oxt efl Oxt cs Oxt ds Oxt es Oxt fs Oxt	x000000931c9ce730 x000000931c9ce830 x00007ffae8356fa x0000202 x0033 x002b x002b x002b x002b x002b	00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff	`ae8356d7 895c2440 ae8356db 4895c2448 ae8356db f15f2680000 `ae8356ef 498bBe ae8356ef 4c84442440 `ae83565ee 488b542470 `ae835673 488b51	mov mov call mov lea mov	dword ptr [rsp+40h], cbp qword ptr [rsp+40h], rbp gword ptr [amsi!_imp_Get rcx, gword ptr [r14] r8, [rsp+40h]		ileTime (00		00007fff`ae7b37f0_48895c2408		
rbp 0x0 rip 0x0 efl 0x0 cs 0x0 ds 0x0 es 0x0 fs 0x0	x000000931c9ce830 x00007fffae8356fa x00000202 x0033 x002b x002b x002b x0053	00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff	`ae8356db 48895c2448 `ae8356e0 ff15f2680000 `ae8356e6 498b0e `ae8356e9 4c8d442440 `ae8356ee 488b542470 `ae8356f3 488b61	mov call mov lea mov	<pre>qword ptr [rsp:486], rbx aword_ptr [amsi!_imp_Get rcx, qword ptr [r14] r8, [rsp+486]</pre>		ileTime (00	6			
rip Oxi efl Oxi cs Oxi ds Oxi es Oxi fs Oxi	x00007fffae8356fa x00000202 x0033 x002b x002b x002b x0053	00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff	`ae835660 ff15f2680000 ae835666 498b0e `ae835669 4c8d442440 `ae83566e 488b542470 `ae8356f3 488b01	call mov lea mov	<pre>gword_ptr [amsi!_imp_Get rcx, gword ptr [r14] r8, [rsp+40h]</pre>	tSystemTimePreciseAsF.	ileTime (00				
efi Oxf cs Oxf ds Oxf es Oxf fs Oxf	x00000202 x0033 x002b x002b x002b x0053	00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff	`ae8356e9 4c8d442440 `ae8356ee 488b542470 `ae8356f3 488b01	lea mov	rex, gword ptr [r14] r8, [rsp+40h]				00007fff`ae7b37fa 4889742420	nov	qword ptr [rsp+26
cs 0x0 ds 0x0 es 0x0 fs 0x0	x0033 x002b x002b x0053	00007fff 00007fff 00007fff 00007fff 00007fff 00007fff	`ae8356ee 488b542470 `ae8356f3 488b01	mov					00007fff`ae7b37ff 57	push	rdi
ds 0x0 es 0x0 fs 0x0	x002b x002b x0053	00007fff 00007fff 00007fff 00007fff	`ae8356f3 488b01		rdx, aword otr [rsp+78b]				00007fff`ae7b3800 4156 00007fff`ae7b3802 4157	push	r14 r15
es 0x0 fs 0x0	x002b x0053	00007fff 00007fff	ae8356f6 488b4018	mov	rax, oword otr [rcx]				00007fff`ae7b3802 4157	push sub	rsp,20h
fs Ox	x0053	00007fff		mov	rax, gword ptr [rax+18h]				00007fff`ae7b3808 4d8bf0	nov	r14,r8
			ae8356fa ff15b86a8880 ae835788 488d4c2458	lea	<pre>qword ptr [amsi!_guard_c rcx, [rsp+50h]</pre>	dispatch_icall_fptr (00007fff`ae				
qs 0x(00007fff	ae835705 48895c2450	mov	qword ptr [rsp+50h], rb	ĸ					
	x002b	00007fff	`ae83570a 448bf0	mov	r14d, eax						
r8 0x0	x000000931c9ce770		`ae83570d ff15c5680000 `ae835713 4c8b4c2450	call mov	<pre>qword ptr [amsi!_imp_Get r9, qword ptr [rsp+50h]</pre>	tSystemTimePreciseAsF.	ileTime (00				
r9 0x0	x000000007ffe4000		ae835718 4c2b4c2466	sub	r9, gword ptr [rsp+38h]						
r10 0x	x00000000546c6148	00007fff	ae83571d 4c894c2468	mov	qword ptr [rsp+68h], r9						
	xfffffffffffffffffffffffffffffffffffff		ae835722 395c2458	спр	dword ptr [rsp+58h], eb	((00007555)	07[7-4]				
	x000002508945d540	00007fff	`ae835726 0f85c1000000 `ae83572c 833dddb8000005	jne c≣D	amsi!CAmsiAntimalware::S dword ptr [amsi!WPP_GLOB	SCAN+OXIEC (0000/TTT SAL Control+8x18 (888	87fff`ae841				
	x0000000000000000000000000000000000000	00007fff	ae835733 0f86b4000000	jbe	amsi!CAmsiAntimalware::S						
	x000002508945d4c0	00007fff	ae835739 48ba000000000400000		rdx, 400000000000h						
	x0000000000000000000000000000000000000		`ae835743 488515d6b80000 `ae83574a 7415	test je	<pre>qword ptr [amsi!WPP_GLOB amsi!CAmsiAntimalware::S</pre>						
	x0000000000000000000000000000000000000		ae83574c 488b05d5b80000	mov	rax, gword ptr [amsi!WPF						
	x0000000000000000000000000000000000000		ae835753 4823c2	and	rax, rdx						
	x0000000000000000000000000000000000000		`ae835756 483b05cbb80000 `ae83575d b001	cmp mov	<pre>rax, qword ptr [amsi!WPF al, 1</pre>	P_GLOBAL_Control+0x28	(00007fff)				
	x0000000000000000000000000000000000000		ae83575f 7402	je	amsi!CAmsiAntimalware::S	Scan+0x163 (00007fff)	ae835763)				
	x00000000000000000	00007fff	`ae835761 8ac3	mov	al, bl						
	x000000000000000		`ae835763 84c0 `ae835765 0f8482000000	test ie	al, al amsi!CAmsiAntimalware::S	Scan+Avled (AAAA7fff)	ae8357ed)				
	x000000000000000	00007fff	ae83576b 488d44245c	je lea	rax, [rsp+5Ch]				4		
	x00000000000000000	00007fff	ae835770 44897c245c	mov	dword ptr [rsp+5Ch], r15	ōd					
	x000000000000000000000		`ae835775 488945a0 `ae835779 488d442468	mov lea	<pre>qword ptr [rbp-60h], rax rax, [rsp+68h]</pre>				:019>		
	x0000000000000000		ae83577e 488945c0	mov	qword ptr [rbp-48h], ray			N	lemory		-
	x00000000000000000 x00000000ae7b37f0	00007fff	ae835782 488d442448	lea	rax, [rsp+48h]						
	x00000000ae7637f0 x00000000a3302510		`ae835787 488945d0 `ae83578b 8b07	mov mov	<pre>qword ptr [rbp-30h], rax eax, dword ptr [rdi]</pre>			P	Address: @rsp		
	x000000001c9ce8c0	00007fff	ae835780 8007 `ae83578d 89442450	mov	dword ptr [rsp+50h], eax			1	00000093`1C9CE7B0 00000000 00	000000	00000018 00000000
	x0000000000000000000000000000000000000	00007fff	ae835791 488d442450	lea	rax, [rsp+50h]			0	00000093`1C9CE7C0 B80F5020 00	007FFF	0000000 0000000
	x000000001c9ce730		ae835796 488945e0	mov	<pre>qword ptr [rbp-28h], rax qword ptr [rbp-58h], 4</pre>						A364E2E8 00000250
	x000000001c9ce830		`ae83579a 48c745a804000000 `ae8357a2 4c8965b0	mov mov	qword ptr [rbp-58h], 4 qword ptr [rbp-58h], r12	2					00000002 00000000 B7738E01 00007FFF
	x00000008945d480	00007fff	ae8357a6 48c745b810000000	mov	qword ptr [rbp-48h], 10h						A364E2E8 00000250
edi 0x	x000000001c9cea28		ae8357ae 48c745c808000000	mov	qword ptr [rbp-38h], 8				0000093`1C9CE810 00000000 00	000000	0000000 00000000
	x000000001c9ce770		`ae8357b6 4489742448 `ae8357bb 48c745d804000000	MOV MOV	dword ptr [rsp+48h], r14 gword ptr [rbp-28h], 4	40			0000093`1C9CE820 98516CC6 00	0038A2	0000002 0000000
	x00000007ffe4000		ae8357c3 48c745e804000000	nov	qword ptr [rbp-18h], 4						activate Windows
LOL		11 📊							Watch Stack Memory		

mov rax, qword ptr [rcx] stores value at *this->0x40 in rax register. Next
instruction takes 0x18 th offset of it and stores it back in rax
register. Then that address is
called using a gaurd_dispatch_icall_fptr .

With that information it is clear that this->0×40 is a pointer to an object of an unknown class. rcx now points to that object and rax holds one of function pointers in the object's vftable. Well my guess is that this is the windows defender's AMSI COM interface.

The first argument passed to the function is this ->0x40. Second, third and fourth are passed through rdx and r8 registers. we can see that in the disassembly rdx being set to rsp+0x70 (amsiBuffer) and r8 being initialized to the address of rsp +0x40 (who's value is 0).

Weird thing is, the function is jumping into the middle of a function.

Let's try following it.

00007555,757754 -00//50000	call		
00007fff`ae7b37bd e8066f0000 00007fff`ae7b37c2 4c8d5c2450		MpDav!DllRegisterServer+0x7f38 (00007fff`ae7ba6c8)	
00007fff`ae7b37c7 498b5b20	lea mov	r11, [rsp+50h] rbx, gword ptr [r11+20h]	
00007fff`ae7b37cb 498b6b28		rbp, gword ptr [r11+28h]	
	mov		
00007fff`ae7b37cf 498b7330 00007fff`ae7b37d3 498b7b38	mov	rsi, qword ptr [r11+30h]	
00007fff`ae7b37d7 498b7b38	mov	rdi, qword ptr [r11+38h]	
00007fff`ae7b37da 415f	MOV	rsp, r11 r15	
00007fff`ae7b37dc 415e	рор		
	рор	r14	
00007fff`ae7b37de 415c	pop	r12	
00007fff`ae7b37e0 c3 00007fff`ae7b37e1 cc	ret int	3	
00007fff`ae7b37e2 cc	int	3	
00007fff`ae7b37e3 cc	int	3	
00007fff`ae7b37e4 cc	int	3	
00007fff`ae7b37e5 cc		3	
00007fff`ae7b37e6 cc	int int	3	
00007fff`ae7b37e7 cc	int	3	
00007fff`ae7b37e8 cc		3	
00007fff`ae7b37e9 cc	int	3	
00007fff`ae7b37ea cc	int int	3	
	int	3	
00007fff`ae7b37eb cc 00007fff`ae7b37ec cc	int	3	
		3	
00007fff`ae7b37ed cc 00007fff`ae7b37ee cc	int int	3	
	101	3	
		7	
00007fff`ae7b37ef cc	int	3	07666 60065460
00007fff`ae7b37ef cc 00007fff`ae7b37f0 48895c2408	int mov	_gword_ptr_[rsp+8], rbx_ss:00000093`1c9ce730={clr!WKS::gc_heap::more_space_lock_soh_(000	007fff`b80f54e8)]
00007fff`ae7b37ef cc 00007fff`ae7b37f0 48895c2408 00007fff`ae7b37f5 48896c2410	int mov mov	<pre>gword ptr [rsp+8], rbx ss:00000093'1c9ce730={clr!WKS::gc_heap::more_space_lock_soh (000 gword ptr [rsp+10h], rbp</pre>	007fff`b80f54e8)]
00007fff`ae7b37ef cc 00007fff`ae7b37f0 48895c2408 00007fff`ae7b37f5 48896c2410 00007fff`ae7b37f5 48896c2410	int mov mov mov	<pre>qword ptr [rsp+8], rbx ss:00000093'1c9ce730={clr!WKS::gc_heap::more_space_lock_soh (000 qword ptr [rsp+10h], rbp qword ptr [rsp+20h], rsi</pre>	007fff`b80f54e8)]
00007fff`ae7b37ef cc 100007fff`ae7b37f0 48895c2408 00007fff`ae7b37f5 48896c2410 00007fff`ae7b37fa 4889742420 00007fff`ae7b37ff 57	int mov mov mov push	<pre>qword ptr [rsp+8], rbx ss:00000093'1c9ce730={clr!WKS::gc_heap::more_space_lock_soh (000 qword ptr [rsp+10h], rbp qword ptr [rsp+20h], rsi rdi</pre>	007fff`b80f54e8)]
00007fff`ae7b37ef cc 100007fff`ae7b37f0 48895c2408 000007fff`ae7b37f5 48896c2410 00007fff`ae7b37f5 4889742420 00007fff`ae7b37ff 57 00007fff`ae7b37ff 57	int Mov Mov push push	<pre>qword ptr [rsp+8], rbx ss:00000093'1c9ce730={clr!WKS::gc_heap::more_space_lock_soh (000 qword ptr [rsp+10h], rbp qword ptr [rsp+20h], rsi rdi r14</pre>	107fff`b80f54e8)]
00007fff`ae7b37ef cc 00007fff`ae7b37f0 48895c2408 00007fff`ae7b37f5 48896c2410 00007fff`ae7b37f5 4889742420 00007fff`ae7b37ff 57 000007fff`ae7b3800 4156 00007fff`ae7b3802 4157	int mov mov push push push	<pre>qword ptr [rsp+8], rbx ss:00000093'1c9ce730={clr!WKS::gc_heap::more_space_lock_soh (000 qword ptr [rsp+10h], rbp qword ptr [rsp+20h], rsi rdi r14 r15</pre>)07fff`b80f54e8)]
00007fff`ae7b37ef cc 00007fff`ae7b37f0 48895c2408 00007fff`ae7b37f3 48895c2410 00007fff`ae7b37f3 4889742420 00007fff`ae7b37f3 4889742420 00007fff`ae7b3802 4156 00007fff`ae7b3802 4157 00007fff`ae7b3804 4883ec20	int mov mov push push push sub	<pre>qword ptr [rsp+8], rbx ss:00000093'1c9ce730={clr!WKS::gc_heap::more_space_lock_soh (000 qword ptr [rsp+10h], rbp qword ptr [rsp+20h], rsi rdi r14 r15 rsp, 20h</pre>	107fff'b80f54e8))
00007fff`ae7b37ef cc 100007fff`ae7b37f0 48895c2408 00007fff`ae7b37f3 48896c2410 00007fff`ae7b37f3 4889742420 00007fff`ae7b37f5 7 00007fff`ae7b3806 4156 00007fff`ae7b3806 4157 00007fff`ae7b3804 4883ec20 00007fff`ae7b3808 4d8bf0	int mov mov push push push sub mov	<pre>qword ptr [rsp+8], rbx ss:00000093'1c9ce730={clr!WKS::gc_heap::more_space_lock_soh (000 qword ptr [rsp+10h], rbp qword ptr [rsp+20h], rsi rdi r14 r15 rsp, 20h r14, r8</pre>	107fff`b80f54e8)]
00007fff`ae7b37ef cc 100007fff`ae7b37f0 48895c2408 00007fff`ae7b37f5 48896c2410 00007fff`ae7b37f5 48897c2420 00007fff`ae7b37ff 57 00007fff`ae7b3806 4156 00007fff`ae7b3802 4157 00007fff`ae7b3802 4883c20 00007fff`ae7b3808 4488f0 00007fff`ae7b3808 4c8bfa	int mov mov push push push sub mov mov	<pre>qword ptr [rsp+8], rbx ss:00000093'1c9ce730={clr!WKS::gc_heap::more_space_lock_soh (000 qword ptr [rsp+20h], rbp qword ptr [rsp+20h], rsi rdi r14 r15 rsp, 20h r14, r8 r15, rdx</pre>)07fff'b80f54e8))
00007fff`ae7b37ef cc 00007fff`ae7b37f0 48895c2408 00007fff`ae7b37f3 48895c2418 00007fff`ae7b37f3 4889742420 00007fff`ae7b37f4 57 00007fff`ae7b3806 4156 00007fff`ae7b3802 4157 00007fff`ae7b3802 4157 00007fff`ae7b3808 4488f0 00007fff`ae7b3808 448bf0 00007fff`ae7b3808 448bf0	int mov mov push push sub mov mov mov	<pre>qword ptr [rsp+8], rbx ss:00000093'1c9ce730={clr!WKS::gc_heap::more_space_lock_soh (000 qword ptr [rsp+10h], rbp qword ptr [rsp+20h], rsi rdi r14 r15 rsp, 20h r14, r8 r15, rdx rsi, rcx</pre>	107 fff b80 f 54e8)]
00007fff`ae7b37ef cc 100007fff`ae7b37ef 48895c2408 00007fff`ae7b37f5 48895c2408 00007fff`ae7b37f5 4889742420 00007fff`ae7b37f5 7 00007fff`ae7b3800 4156 00007fff`ae7b3802 4157 00007fff`ae7b3804 4883ec20 00007fff`ae7b3804 488460 00007fff`ae7b3804 488460 00007fff`ae7b3804 488461 00007fff`ae7b3804 4885c0	int mov mov push push sub mov mov mov test	<pre>qword ptr [rsp+8], rbx ss:00000093'1c9ce730={clr!WKS::gc_heap::more_space_lock_soh (000 qword ptr [rsp+20h], rsi rdi r14 r15 rsp, 20h r14, r8 r15, rdx rsi, rcx r8, r8</pre>	107fff`b80f54e8)]
00007fff`ae7b37ef cc 100007fff`ae7b37f0 48895c2408 00007fff`ae7b37f3 48895c2408 00007fff`ae7b37f3 4889742420 00007fff`ae7b37f5 4889742420 00007fff`ae7b3808 4156 00007fff`ae7b3808 4156 00007fff`ae7b3804 4885c20 00007fff`ae7b3804 4885f0 00007fff`ae7b3805 4c8bfa 00007fff`ae7b3811 4d85c0 00007fff`ae7b3814 750a	int mov mov push push sub mov mov test jne	<pre>qword ptr [rsp+8], rbx ss:00000093'1c9ce730={clr!WKS::gc_heap::more_space_lock_soh (000 qword ptr [rsp+20h], rbp qword ptr [rsp+20h], rsi rdi r14 r15 rsp, 20h r14, r8 r15, rdx rsi, rcx rsi, rcx r8, r8 MpDav!DllRegisterServer+0x1090 (00007fff`ae7b3820)</pre>	107fff [°] b80f54e8)]
00007fff`ae7b37ef cc 100007fff`ae7b37fb 48895c2408 00007fff`ae7b37fb 48896c2410 00007fff`ae7b37fa 4889742420 00007fff`ae7b37ff 57 00007fff`ae7b3806 4156 00007fff`ae7b3802 4157 00007fff`ae7b3804 4883ec20 00007fff`ae7b3804 4883ec20 00007fff`ae7b3804 4885fa 00007fff`ae7b3806 488bfa 00007fff`ae7b3806 488bfa 00007fff`ae7b3814 485c0 00007fff`ae7b3814 750a 00007fff`ae7b3816 b857000780	int mov mov push push sub mov mov mov test jne mov	qword ptr [rsp+8], rbx ss:00000093'1c9ce730={clr!WKS::gc_heap::more_space_lock_soh (000 qword ptr [rsp+20h], rbi rdi rdi r14 r15 rsp, 20h r14, r8 r15, rdx rsi, rcx r8, r8 Mp0av!01lRegisterServer+0x1090 (00007fff`ae7b3820) eax, 80070057h)07fff'b80f54e8))
00007fff`ae7b37ef cc 00007fff`ae7b37ef c8895c2408 00007fff`ae7b37f3 48895c2410 00007fff`ae7b37f3 4889742420 00007fff`ae7b37f3 4889742420 00007fff`ae7b3804 4155 00007fff`ae7b3802 4157 00007fff`ae7b3802 4157 00007fff`ae7b3808 4d8bf0 00007fff`ae7b3808 4d8bf0 00007fff`ae7b3808 4d8bf1 00007fff`ae7b3804 4885c0 00007fff`ae7b3811 4d85c0 00007fff`ae7b3814 750a 00007fff`ae7b3814 b857000780 00007fff`ae7b381b e96e010000	int mov mov push push sub mov mov test jne mov jmp	<pre>qword ptr [rsp+8], rbx ss:00000093'1c9ce730={clr!WKS::gc_heap::more_space_lock_soh (000 qword ptr [rsp+20h], rbi rdi r14 r15 rsp, 20h r14, r8 r15, rdx rsi, rcx r8, r8 Mp0av!DllRegisterServer+0x1090 (00007fff`ae7b3820) eax, 80070057h Mp0av!DllRegisterServer+0x11fe (00007fff`ae7b38e)</pre>	107 fff b80 f 54e8))
00007fff`ae7b37ef cc 100007fff`ae7b37ef 48895c2408 00007fff`ae7b37ef 48895c2408 00007fff`ae7b37ef 4889742420 00007fff`ae7b37ef 4889742420 00007fff`ae7b3800 4156 00007fff`ae7b3802 4157 000007fff`ae7b3804 4883ec20 00007fff`ae7b3804 4883ec20 00007fff`ae7b3804 4885ef1 00007fff`ae7b3804 4885ef1 00007fff`ae7b3814 4885c0 00007fff`ae7b3814 4885c0 00007fff`ae7b3814 50a 00007fff`ae7b3814 b857000780 00007fff`ae7b3816 b857000780 00007fff`ae7b3816 b867000780	int nov mov push push sub mov mov test jne mov jmp mov	<pre>qword ptr [rsp+8], rbx ss:00000093'1c9ce730={clr!WKS::gc_heap::more_space_lock_soh (000 qword ptr [rsp+20h], rbi rdi r14 r15 rsp, 20h r14, r8 r15, rdx rsi, rcx r8, r8 MpDav!DllRegisterServer+0x1090 (00007fff`ae7b3820) eax, 80070057h MpDav!DllRegisterServer+0x11fe (00007fff`ae7b398e) dword ptr [r8], 1</pre>	107fff`b80f54e8)]
00007fff`ae7b37ef cc 100007fff`ae7b37ef 48895c2408 00007fff`ae7b37f6 48895c2408 00007fff`ae7b37f6 48897c2409 00007fff`ae7b37f6 4889742420 00007fff`ae7b3806 4156 00007fff`ae7b3802 4157 00007fff`ae7b3802 4157 00007fff`ae7b3804 4885c20 00007fff`ae7b3804 4885e20 00007fff`ae7b3804 4885f0 00007fff`ae7b3814 4885c0 00007fff`ae7b3814 455c0 00007fff`ae7b3814 455c0 00007fff`ae7b3815 e96e010000 00007fff`ae7b3816 b857000788 00007fff`ae7b3816 b857000788 00007fff`ae7b3820 41c70001000000 00007fff`ae7b3827 86b9c80000000	int nov push push push sub mov mov test jne mov jmp mov cmp	<pre>qword ptr [rsp+8], rbx ss:00000093'1c9ce730={clr!WKS::gc_heap::more_space_lock_soh (000 qword ptr [rsp+20h], rbp qword ptr [rsp+20h], rsi rdi r14 r15 rsp, 20h r14, r8 r15, rdx rsi, rcx r8, r8 MpDav!DllRegisterServer+0x1090 (00007fff`ae7b3820) eax, 80070057h MpDav!DllRegisterServer+0x11fe (00007fff`ae7b398e) dword ptr [r8], 1 byte ptr [rcx+0C8h], 0</pre>	107fff [*] b80f54e8)]
00007fff`ae7b37ef cc 00007fff`ae7b37f6 48895c2408 00007fff`ae7b37f6 48895c2418 00007fff`ae7b37f6 4889742420 00007fff`ae7b37f6 57 00007fff`ae7b3802 4157 00007fff`ae7b3802 4157 00007fff`ae7b3804 4883ec20 00007fff`ae7b3804 4885ec20 00007fff`ae7b3804 4885ec20 00007fff`ae7b3804 4885f1 00007fff`ae7b3804 4885f1 00007fff`ae7b3814 4885e0 00007fff`ae7b3816 4885f1 00007fff`ae7b3816 485f0 00007fff`ae7b3816 4857000780 00007fff`ae7b3816 957000780 00007fff`ae7b3816 957000780 00007fff`ae7b3816 957000780 00007fff`ae7b3826 41c700010000000 00007fff`ae7b3827 81b9c800000000	int mov mov push push push sub mov mov test jne mov jap mov jap ge	<pre>qword ptr [rsp+8], rbx ss:00000093'1c9ce730={clr!WKS::gc_heap::more_space_lock_soh (000 qword ptr [rsp+20h], rbp qword ptr [rsp+20h], rsi rdi r14 r15 rsp, 20h r14, r8 r15, rdx rsi, rcx r8, r8 MpOav!DllRegisterServer+0x1090 (00007fff`ae7b3820) eax, 80070057h MpOav!DllRegisterServer+0x11fe (00007fff`ae7b3820) dword ptr [rc8], 1 byte ptr [rcx+0C8h], 0 MpOav!DllRegisterServer+0x1005 (00007fff`ae7b3865)</pre>)07fff'b80f54e8))
00007fff`ae7b37ef cc 00007fff`ae7b37ef c8895c2408 00007fff`ae7b37f3 48895c2408 00007fff`ae7b37f3 4889742420 00007fff`ae7b37f3 4889742420 00007fff`ae7b3802 4157 00007fff`ae7b3802 4157 00007fff`ae7b3802 4157 00007fff`ae7b3808 4d8bf0 00007fff`ae7b3808 4d8bf0 00007fff`ae7b3808 4d8bf1 00007fff`ae7b3804 4885c0 00007fff`ae7b3804 485c0 00007fff`ae7b3811 4d85c0 00007fff`ae7b3814 750a 00007fff`ae7b3814 9500780 00007fff`ae7b3812 e96e10000 00007fff`ae7b3828 41c70001000000 00007fff`ae7b3828 2735 000007fff`ae7b3828 4381449fa0300	int nov mov push push sub mov mov test jne mov test jne mov cmp je lea	<pre>qword ptr [rsp+8], rbx ss:00000093'1c9ce730={clr!WKS::gc_heap::more_space_lock_soh (000 qword ptr [rsp+20h], rbi rdi rdi rl4 r15 rsp, 20h r14, r8 r15, rdx rsi, rcx r8, r8 Mp0av!DilRegisterServer+0x1090 (00007fff`ae7b3820) eax, 80070057h Mp0av!DilRegisterServer+0x11fe (00007fff`ae7b382) dword ptr [r8], 1 byte ptr [rcx+0C8h], 0 Mp0av!DilRegisterServer+0x1045 (00007fff`ae7b3865) rbx, [Mp0av!DilRegisterServer+0x1045 (00007fff`ae7b3865]</pre>	107fff`b80f54e8)]
00007fff`ae7b37ef cc 100007fff`ae7b37ef 48895c2408 00007fff`ae7b37ef 48895c2408 00007fff`ae7b37ef 48897c2409 00007fff`ae7b37ef 4889742420 00007fff`ae7b3800 4156 00007fff`ae7b3802 4157 000007fff`ae7b3804 4883ec20 00007fff`ae7b3804 4883ec20 00007fff`ae7b3804 4885ef1 00007fff`ae7b3804 4885ef1 00007fff`ae7b3814 4885c0 00007fff`ae7b3814 4885c0 00007fff`ae7b3814 4885c0 00007fff`ae7b3814 50a 00007fff`ae7b3814 50a 00007fff`ae7b3814 50a 00007fff`ae7b3814 50a 00007fff`ae7b3814 50a 00007fff`ae7b3814 50a 00007fff`ae7b3814 4485c0 00007fff`ae7b3814 50a 00007fff`ae7b3814 4485c0 00007fff`ae7b3814 4485c0 00007fff`ae7b3814 4485c0 00007fff`ae7b3814 4485c0 00007fff`ae7b3814 4485c0 00007fff`ae7b3827 840c000000 00007fff`ae7b3827 840c0000000 00007fff`ae7b3838 448414476a300	int mov mov push push sub mov mov test jne mov jne mov cmp je lea mov	qword ptr [rsp+8], rbx ss:00000093'1c9ce730={clr!WKS::gc_heap::more_space_lock_soh (000 qword ptr [rsp+10h], rbp qword ptr [rsp+20h], rsi rdi r14 r15 rsp, 20h r14, r8 r15, rdx rsi, rcx r8, r8 MpDav!DllRegisterServer+0x1090 (00007fff`ae7b3820) eax, 80070057h MpDav!DllRegisterServer+0x11fe (00007fff`ae7b398e) dword ptr [r8], 1 byte ptr [rcx+028h], 0 MpDav!DllRegisterServer+0x1045 (00007fff`ae7b3865) rbx, [MpDav!DllRegisterServer+0x0405 (00007fff`ae7b3280)] rcx, qword ptr [r4], 1	107fff`b80f54e8)]
00007fff`ae7b37ef cc 00007fff`ae7b37ef 48895c2408 00007fff`ae7b37ef 48895c2418 00007fff`ae7b37ef 4889742420 00007fff`ae7b37ef 4889742420 00007fff`ae7b380e 4156 00007fff`ae7b380e 4156 00007fff`ae7b380e 4157 00007fff`ae7b380e 4885ec20 00007fff`ae7b380e 4885ec 00007fff`ae7b380e 488bfa 00007fff`ae7b381e 488bfa 00007fff`ae7b381e 488bfa 00007fff`ae7b381e 488bfa 00007fff`ae7b381e 488bfa 00007fff`ae7b381e 96ee10000 00007fff`ae7b381e 96e010000000 00007fff`ae7b382e 7435 00007fff`ae7b383e 488bfd49fa0300 00007fff`ae7b383e 488bcd42fa0300 00007fff`ae7b383e 488bcd	int nov mov push push sub mov mov test jne mov jmp ccmp je Lea mov ccmp	<pre>qword ptr [rsp+8], rbx ss:00000093'1c9ce730={clr!WKS::gc_heap::more_space_lock_soh (000 qword ptr [rsp+10h], rbp qword ptr [rsp+20h], rsi rdi r14 r15 rsp, 20h r14, r8 r15, rdx rsi, rcx r8, r8 MpDav!DlRegisterServer+0x1090 (00007fff`ae7b3820) eax, 80070057h MpDav!DlRegisterServer+0x11fe (00007fff`ae7b3820) dword ptr [r8], 1 byte ptr [rcx+0C8h], 0 MpDav!DlRegisterServer+0x10d5 (00007fff`ae7b3865) rbx, [MpDav!DlRegisterServer+0x40af0 (00007fff`ae7f3280)] rcx, qword ptr [MpDav!DlRegisterServer+0x40af0 (00007fff`ae7f3280)] rcx, rbx MpDav!DlRegisterServer+0x10d5 (00007fff`ae7h385p)</pre>	
00007fff`ac7b37cf cc 00007fff`ac7b37cf cc 00007fff`ac7b37cf 48895c2408 00007fff`ac7b37cf 48895c2410 00007fff`ac7b37cf 4889742420 00007fff`ac7b3802 4157 00007fff`ac7b3802 4157 00007fff`ac7b3802 4157 00007fff`ac7b3808 4d8bf0 00007fff`ac7b3808 4d8bf0 00007fff`ac7b3808 4d8bf1 00007fff`ac7b3804 4885c0 00007fff`ac7b3804 4885c1 00007fff`ac7b3804 4885c1 00007fff`ac7b3804 4885c1 00007fff`ac7b3804 4885c1 00007fff`ac7b3802 41c70001000000 00007fff`ac7b3810 b857000780 00007fff`ac7b3820 41c700010000000 00007fff`ac7b3820 41c70001000000 00007fff`ac7b3827 488b042fa0300 000007fff`ac7b3837 488b042fa0300 00007fff`ac7b3841 741b	int nov mov push push sub mov mov test jne mov cmp je lea mov cmp je	qword ptr [rsp+8], rbx ss:00000093'1c9ce730={clr!WKS::gc_heap::more_space_lock_soh (000 qword ptr [rsp+10h], rbp qword ptr [rsp+20h], rsi rdi r14 r15 rsp, 20h r14, r8 r15, rdx rsi, rcx r8, r8 MpDav!DllRegisterServer+0x1090 (00007fff`ae7b3820) eax, 80070057h MpDav!DllRegisterServer+0x11fe (00007fff`ae7b3865) rbx, [MpDav!DllRegisterServer+0x10d5 (00007fff`ae7b3865) rbx, rbx rby, rbx rby, rbx rdi rby rby	107 fff' b80f54e8)
00007fff`ae7b37ef cc 100007fff`ae7b37f5 48896c2408 00007fff`ae7b37f5 48896c2410 00007fff`ae7b37f5 48896c2410 00007fff`ae7b37f5 57 00007fff`ae7b3806 4156 00007fff`ae7b3802 4157 00007fff`ae7b3802 4157 00007fff`ae7b3804 4883ec20 00007fff`ae7b3804 4885ec20 00007fff`ae7b3804 4885ec20 00007fff`ae7b3804 4885f1 00007fff`ae7b3814 4885e0 00007fff`ae7b3814 4885e0 00007fff`ae7b3815 b857000780 00007fff`ae7b3816 b857000780 00007fff`ae7b3816 9860000000 00007fff`ae7b3827 8805c80000000 00007fff`ae7b3837 488b0449fa0300 00007fff`ae7b3837 488b042fa0300 00007fff`ae7b3836 488b642fa0300	int nov mov push push sub mov mov test jne mov jmp ccmp je Lea mov ccmp	qword ptr [rsp+8], rbx ss:00000093'1c9ce730={clr!WKS::gc_heap::more_space_lock_soh (000 qword ptr [rsp+10h], rbp qword ptr [rsp+20h], rsi rdi r14 r15 rsp, 20h r14, r8 r15, rdx rsi, rcx r8, r8 Mp0av!DllRegisterServer+0x1090 (00007fff`ae7b3820) eax, 80070057h Mp0av!DllRegisterServer+0x11fe (00007fff`ae7b3820) eax, 80070057h Mp0av!DllRegisterServer+0x11fe (00007fff`ae7b3865) rbx, [Mp0av!DllRegisterServer+0x10d5 (00007fff`ae7b3865) rbx, [Mp0av!DllRegisterServer+0x40af0 (00007fff`ae7f3280)] rcx, rbx mord ptr [Mp0av!DllRegisterServer+0x10dc (00007fff`ae7b385e) byte ptr [rcx+1Ch], 4	

Well this makes it bit clear. First of all we not jumping into the middle of a function, See that **ret** instruction up there? What this tells us is, we jumped into a function but it is not labelled correctly.

However if you try to go o this address from a disassembler, it will fail. Indicating that this a function from another dll.

here's the memory map.

Command X	,
00007fff`81820000 00007fff`81986000	System_Management_ni (deferred)
00007fff 81bd0000 00007fff 81d36000	System_DirectoryServices_ni (deferred)
00007fff`88bf0000 00007fff`88c52000	Microsoft_PowerShell_Security_ni (deferred)
00007fff`88c60000 00007fff`88e4d000	Microsoft_CSharp_ni (deferred)
00007fff`88e90000 00007fff`88ebd000	System_Configuration_Install_ni (deferred)
00007fff`88ec0000 00007fff`88f08000	AppxSip (deferred)
00007fff`88f10000 00007fff`88f61000	System Numerics_ni (deferred)
00007fff`89910000 00007fff`899b0000	Microsoft_Management_Infrastructure_ni (deferred)
00007fff`8bc30000 00007fff`8c4db000	System_Xml_ni (deferred)
00007fff`8c4e0000 00007fff`8c613000	System_Configuration_ni (deferred)
00007fff`8c640000 00007fff`8c6e8000	Microsoft_PowerShell_ConsoleHost_ni (deferred)
00007fff`9e190000 00007fff`9e325000	TaskFlowDataEngine (deferred)
00007fff`9f230000 00007fff`9f2af000	ntshrui (deferred)
00007fff`a1070000 00007fff`a1104000	appresolver (deferred)
00007fff`a7a00000 00007fff`a7a0d000	LINKINFO (deferred)
00007fff`a8450000 00007fff`a846d000	wshext (deferred)
00007fff`ad190000 00007fff`ad1ac000	ATL (deferred)
00007fff`ad7a0000 00007fff`ad81a000	OneCoreCommonProxyStub (deferred)
00007fff`ae7b0000 00007fff`ae82a000	MpDav (export symbols) C:\ProgramData\Microsoft\Windows Defender\Platform\4.18.2111.5-8\MpDav.dll
00007fff`ae830000 00007fff`ae847000	amsi (pdb symbols) c:\myserversymbols\Amsi.pdb\B0605BF6E5E98B4E70628DD06218EE811\Amsi.pdb
00007fff`b13b0000 00007fff`b1e25000	System_Core_ni (deferred)
00007fff`b1f20000 00007fff`b1f32000	cscapi (deferred)
00007fff`b1f40000 00007fff`b208f000	clrjit (deferred)
00007fff`b21d0000 00007fff`b21f6000	srvcli (deferred)
00007fff`b2c00000 00007fff`b3870000	<u>System_ni</u> (deferred)
00007fff`b3870000 00007fff`b4e70000	mscorlib_ni (deferred)
00007fff`b71d0000 00007fff`b728d000	ucrtbase_clr0400 (deferred)
00007fff`b76d0000 00007fff`b8192000	<u>clr</u> (pdb symbols) c:\myserversymbols\clr.pdb\20373C0156BD497E8BF052933B09D1562\clr.pdb
00007fff`b82c0000 00007fff`b82d6000	<u>VCRUNTIME140_CLR0400</u> (deferred)
00007fff`b8a60000 00007fff`b8b0a000	m <u>scoreei</u> (deferred)
00007fff`c0470000 00007fff`c09b5000	<u>cdp</u> (deferred)
00007fff`c5180000 00007fff`c51e4000	mscoree (deferred)
00007fff`c5f00000 00007fff`c5f5d000	<u>Bcp47Langs</u> (deferred)
00007fff`c60f0000 00007fff`c60fc000	secur32 (deferred)
00007fff`c6490000 00007fff`c6666000	<u>urlmon</u> (deferred)
00007fff`c6c80000 00007fff`c6d9b000	MPCLIENT (deferred)
00007fff`c87b0000 00007fff`c8a57000	<u>iertutil</u> (deferred)
00007fff`cd700000 00007fff`cd72f000	<u>cryptnet</u> (deferred)
00007fff`cda80000 00007fff`cda8a000	VERSION (deferred)
00007fff`cdf00000 00007fff`cdf0b000	WINNSI (deferred)
00007fff`ce400000 00007fff`ce489000	
0:019>	
0:014>	

See? It seems like this dll is the COM dll that implements **IAmsiAntimalware** interface for windows defender.

To confirm that, let's check the registry.

// registry

Now it is confirmed, let's go through this function.

MpOav!DllRegisterServer+0x1060:		
00007fffae7b37f0 48895c2408	mov	qword ptr [rsp+8],rbx
00007fffae7b37f5 48896c2410	mov	qword ptr [rsp+10h],rbp
00007fffae7b37fa 4889742420	mov	qword ptr [rsp+20h],rsi
00007fffae7b37ff 57	push	rdi
00007fffae7b3800 4156	push	r14
00007fffae7b3802 4157	push	r15
00007fffae7b3804 4883ec20	sub	rsp,20h
00007fffae7b3808 4d8bf0	mov	r14,r8
00007fffae7b380b 4c8bfa	mov	r15,rdx
00007fffae7b380e 488bf1	mov	rsi,rcx
00007fffae7b3811 4d85c0	test	r8,r8
00007fffae7b3814 750a	jne	MpOav!DllRegisterServer+0x1090
(00007fffae7b3820)		
MpOav!DllRegisterServer+0x1086:		
00007fffae7b3816 b857000780	mov	eax,80070057h
00007fffae7b381b e96e010000	jmp	MpOav!DllRegisterServer+0x11fe
(00007fffae7b398e)		

First it does some work on the stack frame and moves **0**×80070057 to **rax** register if third parameter is null (pointer to a stack variable of CAmsiAntimalware::Scan method), And we know this is **E_INVALIDARG**. And then function jumps to the epilogue. So this is basically a small sanity check.

00007fffae7b3820 41c70001000000 mov	dword ptr [r8], 1
ds:000000931c9ce770=00000000	
00007fffae7b3827 80b9c800000000 cmp	byte ptr [rcx+0C8h], 0 ds:00000250a33025d8=00

then it moves 1 or AMSI_RESULT_NOT_DETECTED into third parameter and checks if first parameter (rcx) + 200 is 0. We know that first parameter (rcx) passed down to this function is CAmsiAntimalware->0×40. (yes doesn't make much sense.)

Disassembly		 - ☆ > 	×	Command X ₹
Address:		Follow current instruction		0:019> r rcx rcx=0000025003302510 0:019> db 0x25003302548
00007fff`ae7b3814 750a	jne	MpDav!DllRegisterServer+0x1090 (00007fff`ae7b3820)		
00007fff`ae7b3816 b857000780	mov	eax, 80070057h		A0000256 a33025e8 ca 61 1b 50 00 06 00 8a-43 00 3a 00 5c 00 57 00 .a.PC.
00007fff`ae7b381b e96e010000	jmp	MpDav!DllRegisterServer+0x11fe (00007fff`ee7b398e)	-	00000250`a33025f8 69 00 6e 00 64 00 6f 00-77 00 73 00 5c 00 4d 00 i.n.d.o.w.
00007fff`ae7b3820 41c70001000000	nov	dword ptr [r8], 1		00000250`a3302608 69 00 63 00 72 00 6f 00-73 00 6f 00 66 00 74 00 i.c.r.o.s.
00007fff`ae7b3827 80b9c80000000	спр	byte ptr [rcx+0C8h], 0 ds:00000250`a33025d8=00		00000250`a3302618 2e 00 4e 00 65 00 74 00-5c 00 61 00 73 00 73 00N.e.t.\.
00007fff`ae7b382e 7435	je	MpDav!DllRegisterServer+0x10d5 (00007fff`ae7b3865)		00000250°a3302628 65 00 6d 00 62 00 6c 00-79 00 5c 00 47 00 41 00 e.m.b.l.y.
00007fff`ae7b3830 488d1d49fa0300	lea	rbx, [MpOav!DllRegisterServer+0x40af0 (00007fff`ae7f		00000250 a3302638 43 00 5f 00 4d 00 53 00-49 00 4c 00 5c 00 4d 00 CM.S.I.
00007fff`ae7b3837 488b0d42fa0300	mov	<pre>rcx, qword ptr [MpOav!DllRegisterServer+0x40af0 (000</pre>		00000250 a3302648 69 00 63 00 72 00 6f 00-73 00 6f 00 66 00 74 00 i.c.n.o.s.
00007fff`ae7b383e 483bcb	стр	rcx, rbx		

In our case, comparison turns out to be true.

A little below that, there's a call to another fuction from this dll.

MpOav!DllRegisterServer+0x10d5:		
00007fffae7b3865 488d6970	lea	rbp, [rcx+70h]
00007fffae7b3869 488bcd	mov	rcx, rbp
00007fffae7b386c ff15f6120300	call	qword ptr [MpOav!DllRegisterServer+0x323d8
(00007fffae7e4b68)]		

it seems to take only one argument and it is &rcx+0x70.

ntdll!RtlEnterCriticalSect	tion:	
00007fff`d67bb380_4883ec28	sub	rsp, 28h
00007fff`d67bb384 65488b042530		rax, qword ptr gs:[30h]
00007fff`d67bb38d f00fba710800) lock bt	r dword ptr [rcx+8], 0
00007fff`d67bb393 488b4048	mov	rax, qword ptr [rax+48h]
00007fff`d67bb397 7312	jae	ntdll!RtlEnterCriticalSection+0x2b (00007fff`d67bb3ab)
00007fff`d67bb399 48894110	mov	qword ptr [rcx+18h], rax
00007fff`d67bb39d 33c0	XOP	eax, eax
00007fff`d67bb39f c7410c010000	100 mov	dword ptr [rcx+8Ch], 1
00007fff`d67bb3a6 4883c428	add	rsp, 28h
00007fff`d67bb3aa c3	ret	
00007fff`d67bb3ab 48394110	стр	qword ptr [rcx+18h], rax
00007fff`d67bb3af 750a	jne	ntdll!RtlEnterCriticalSection+0x3b (00007fff`d67bb3bb)
00007fff`d67bb3b1 ff410c	inc	dword ptr [rcx+8Ch]
00007fff`d67bb3b4 33c0	xor	eax, eax
00007fff`d67bb3b6 4883c428	add	rsp, 28h
00007fff`d67bb3ba c3	ret	

if we step into it, windbg indentifies function as **RtlEnterCriticalSection** from ntdll. According to <u>msdn</u>, **EnterCriticalState** function waits for ownership of the specified critical section object. The function returns when the calling thread is granted ownership. function accepts a single parameter and it is of **LPCRITICAL_SECTION**.

In this case, critical section that this function waits for is rcx+0x70.



next few instructions compare **rsi+0x98** with 0 (both rsi and rcx pointed to same address but since rcx now points to rcx+0x70, rsi is used). if comparison fails, it jumps to another location disassembly where **LeaveCriticalState** is being called.

- 0

PID: 1115	52 - WinDbg 1.2111.9001.0 (/	Administrato	or)			-	
File	Home View	Breakpo	pints	Time Travel	Mode	del Scripting Source Memory Command	
2	🚬 🥪 🛛 🛛	• E		2		Accent color 🔽 🔹	
mmand \	Watch Locals Registers Me	mory Stack	c Disassem	bly Threads Break	kpoints I	Logs Notes Timelines Modules Layouts Reset	
						Windows	
isters	- ☆ ×	Disassemb	blv				
lame	Value 🔺						
User		Address:	@\$scopei	p		■ Tellow current instruction ntdll!RtlLeaveCriticalSection:	
	0x0000000000000000000000	00007fff	`ae7b386c	ff15f6120300	call	qword ptr [MpDav!DllRegisterServer+8x323d8 (00007fff`ae7e4b68)]	qword ptr [r qword ptr [r
rbx	0x00007fffae7f3280	00007fff	`ae7b3872	488d8e9800000	lea	rcx, [rsi+98h] 00007fff`d67da8aa 57push	rdi
rcx	0x00000250a357c120		`ae7b3879 `ae7b387d		cmp	qword ptr [rcx], 0	rsp,20h
rdx	0x000000931c9ce8c0			e8a00d0000	jne call		dword ptr [r rbx,rcx
rsi	0x00000250a3302510		`ae7b3884		mov	edi, eax	ntdll!RtlLea
rdi	0x000000931c9cea28		ae7b3886		test	eax, eax 00007fff`d67da8b8 48c741100000000 mov	qword ptr [
rsp	0x000000931c9ce6f0		`ae7b3888		jns	MpDav/DllRegisterServer+8x1139 (00007fff'ae7b38c9) 0:019> t bbs (Magazine 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,	
rbp	0x00000250a3302580			488d1deff90300 488b0de8f90300	lea mov	rbx, [MpDav!DllRegisterServer+8x40af0 (00007fff`ae7f3280)] MpDav!DllRegisterServer+8x1147: rcx, qword ptr [MpDav!DllRegisterServer+8x40af0 (00007fff`ae7f3280)] 80007fff`ae7b38d7 c744245001000000 mov	dword ote
rip	0x00007fffae7b38ed		`ae7b3898		cmp	rex, tword pit inpudvisticesiserververververververververververververve	dword ptr
efl	0x00000246		`ae7b389b		je	NpOav!DllRegisterServer+8x112b (88087fff`ae7b38bb) MpOav!DllRegisterServer+8x114f:	
	0x0033	00007fff	`ae7b389d	f6411c01	test	byte ptr [rcx+1Ch], 1 00007fff`ae7b38df 488d1d9af90300 lea	rbx,[MpOav!C
ds	0x002b		`ae7b38a1		je	HpDev1DllRegisterServer+0x112b (00007fff'ae7b38bb) 0:019> t	
es	0x002b		ae7038a3	ba10000000	mov mov	edx, 10h MpOav!DllRegisterServer+8x1156: r9d, eax 080807fff`ae7b38e6 488b0d6b250400 mov	rcx,qword pt
	0x0053			4c8d05c6700300	lea	r8, [Mp0av!DllRegisterServer+0x381e8 (00007fff`ae7ea978)] 00007ff1 ae7u3e0 4000000230400 000 000	rex,qworu pi
gs	0x002b	00007fff	`ae7b38b2	48864910	mov	rcx, qword ptr [rcx+10h] MpOav!DllRegisterServer+8x115d:	
	0x002b			e8bdf2ffff	call		byte ptr [ro
r8	0x000000931c9ce770		`ae7b38bb		mov		
r9	0x00000007ffe4000			ff159c120300 e9c3000000	call jmp	qword ptr [MpOav!DllRegisterServer+0x323d0 (00007fff`ae704160)] rcx=00000250a357c120 MpOav!DllRegisterServer+0x11fc (00007fff`ae7b398c) 0:019> db @rcx+0x60 L1	
r10	0x00000ffff5cf66fe	00007fff	`ae7b38c9	488bcd	mov		
	0x400000001000000	00007fff	`ae7b38cc	ff158e120300	call	qword ptr [NpOav!DllRegisterServer+0x323d0 (00007fff"ae7e4b60)]	
	0x000002508945d540			8364245400	and	dword ptr [rsp+59h], U	
	0x000000000000000000000000000000000000			c744245001000000 488d1d9af90300	U mov lea	dword ptr [rsp+50h], 1 rbx, [MpDav!DllRegisterServer+0x40af0 (00007fff ae7f3280)]	
r14	0x000000931c9ce770			488b0d6b250400	mov	rcx, qmpod nr MnowierkegisterServer+0x436c8 (00007fff ae7f5e58)	
r15	0x000000931c9ce8c0	00007fff	`ae7b38ed	80796000	cmp	byte pt [rcx+60h], 0 ds:00000250'a357c180=01	
dr0	0x000000000000000000000000000000000000		`ae7b38f1		jne	MpDav!DltRegisterServer+0x1198 (00007fff`ae7b3928)	
dr1	0x000000000000000000000000000000000000			488b0d86f90300	mov	rcx, qword ptr [MpOav!DllRegisterServer+0x40af0 (00007fff`ae7f3280)	
dr2	0x000000000000000000000000000000000000		`ae7b38fa `ae7b38fd		cmp ie	rcx, rbx MpDav!DllRegisterServer+0x1191 (00007fff`ae7b3921)	
dr3	0x000000000000000000000000000000000000		`ae7b38ff		test	hpdevioletelstelserververververververververververververve	
dr6	0x00000000000000000		`ae7b3903		je	NpDav!DllRegisterServer+0x1191 (00007fff`ae7b3921)	
dr7	0x000000000000000000000000000000000000			ba0f00000	mov	edx, 0Fh	
exfrc	0x000000000000000000000000000000000000			4c8d05377e0300	lea	r8, [MpDaviDllRegisterServer+0x38fb8 (00007fff`ae7eb748)]	
exto	0x000000000000000000000000000000000000		`ae7b3911 `ae7b3915	e8a6eaffff	mov call	rcx, qword ptr [rcx+18h] MpDav+8x23c8 (080807fff`ae7b23c8)	
brfrc	0x000000000000000000000000000000000000			488b0d5ff90300	mov	rcx, gword ptr [MpOay]DllRegisterServer+8x48af8 (88887fff)ae7f3288)	
brto	0x000000000000000000000000000000000000	00007fff	`ae7b3921	bf15000780	mov	edi, 80070015h	
ssp	0x000000000000000000000000000000000000		`ae7b3926		jmp	HpDav!DllRegisterServer+0x11be (00007fff`ae7b394e)	₹ 5
cetu	0x000000000000000000000000000000000000		`ae7b3928	488b4138 4c8d442450	mov lea	rax, qword ptr [rcx+38h] r8, [rsp+58h]	,
eax	0x000000000000000000000000000000000000		`ae7b3931		mov	rdx, r15 Address: @rsp	
	0x00000000a357c120	00007fff	`ae7b3934	48864948	mov	rcx, qword ptr [rcx+48h]	
edx	0x00000001c9ce8c0			4886808000000	mov	rax, gword ptr [rax+80h]	
	0x00000000ae7f3280			ff151b130300	call	qword ptr [NpDav!DllRegisterServer+0x324d0 (00007fff`ae7e4c60)] 00000093`1C9CE700 00000000 00000000 A	
esp	0x00000001c9ce6f0		ae7b3945	80+8 488b0d32+90300	mov mov	edi, eax rcx, qword ptr [MpDav!DllRegisterServer+0x40af0 (00007fff`ae7f3280) 00000093`1C9CE710 00000000 00000000 83 1C9CE720 1C9CE720 1C9CE7	
ebp	0x00000000a3302580		`ae7b394e		test	edi, edi	
esi	0x00000000a3302510	00007fff	`ae7b3950	7925	jns	MpDaviDiRegisterServer+8x11e7 (80807fff'ae7b3977) 060000037 129E750 00000000 00000000 000000000	
edi	0x000000001c9cea28	00007fff	`ae7b3952	483bcb	cmp	PCX, PbX 00000093 1C9CE750 00000007 00000000 00	000 000000
r8d	0x00000001c9ce770		ae7b3955		je	Np0av!DllRegisterServer+0x11fc (00007fff'ae7b398c) 00000093`1C9CE760/adaddobarobaoooo og	000 000000
	0x00000007ffe4000		`ae7b3957 `ae7b395b		test je	byte ptr [rcx+1Ch], 1 MpDav!DllRegisterServer+8x11fc (80807fff`ae7b398c)	/indows.
.10.1	<u></u>				20	Watch Stack Memory	11100 113.
	▶						

as shown in the above diagram, function loads **rbp**, which points to the critical section (rsi->0x70) into rcx. Then LeaveCriticalState function is called.

then two local variables, rsp+0x54 and rsp+0x50, get initialized to 0x0 and 0x1, following a mov instruction which loads a global variable into **rcx**. then it does a comparison of rcx+0x60 with o.

In our case, comparision fails and for that reason, jump will be taken.

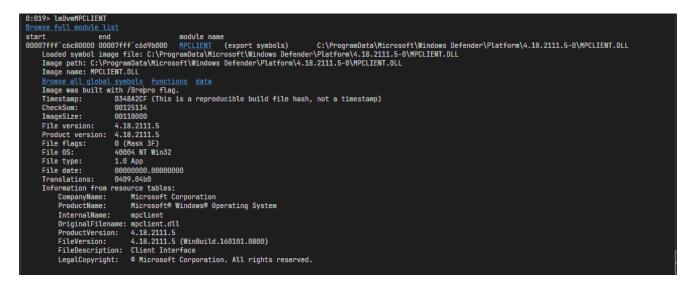
```
MpOav!DllRegisterServer+0x1198:
00007fffae7b3928 488b4138
                                          rax, qword ptr [rcx+38h]
                                  mov
ds:00000250a357c158=00000250a356b1f0
00007fffae7b392c 4c8d442450
                                  lea
                                          r8, [rsp+50h]
00007fffae7b3931 498bd7
                                  mov
                                          rdx, r15
00007fffae7b3934 488b4948
                                          rcx, qword ptr [rcx+48h]
                                  mov
```

here we can see another call.

rcx is set to [rcx+0x48] and rdx is loaded with amsiBuffer meanwhile r8, third argument is loaded with address rsp+0x50.

Disassem	bly		+ A ×	1	Command $ imes$		₽
Address:	@\$scopeip		Follow current instruction		0:019> dq @rax+0x88 L1 00000250`a3565270 00007fff`c6cb 0:019>0 00007fff`c6cb6fc0	ofc0	
00007fff 00007fff 00007fff 00007fff 00007fff 00007fff 00007fff	`ae7b3926 eb26 ae7b3928 488b4138 `ae7b3928 488b4138 `ae7b3924 488b44260 `ae7b3931 498b47 `ae7b3934 488b4988000000 `ae7b393f ff151b130300 `ae7b3945 8bf8 `ae7b3945 8bf8	jmp mov lea mov mov call mov	MpDaviDlRegisterServer+8x11be (00007fff`ae7b394e) rax, qword ptr [rcx+38h] rdx, r15 rcx, qword ptr [rcx+48h] rax, qword ptr [rax+80h] qword ptr [HpDaviDlRegisterServer+0x324d0 (00007fff`ae7e4c60 ed1, eax rcx, qword ptr [MpDaviDlRegisterServer+0x40af0 (00007fff`ae7		0:03-0 0007ff CoCO0fc0 00007ff CoCO0fc0 48095c2420 00007ff CoCO0fc0 48095c2420 00007ff CoCO0fc0 48095c2420 00007ff CoCO0fc0 456 00007ff CoCO0fc0 456 00007ff CoCO0fc0 4154 00007fff CoCO0fc0 4155 00007fff CoCO0fc0 4155	mov push push push push push push push	qword ptr [rsp+20h],rbx rbp rsi rdi r12 r13 r14 r15
00007fff 00007fff 00007fff	`ae7b394e 85ff `ae7b3950 7925 `ae7b3952 483bcb `ae7b3955 7435	test jns cmp je	edi, edi MpOBv/DlRegisterServer+0x11e7 (000007fff`ae7b3977) rcx, rbx MpDav!DlRegisterServer+0x11fc (00007fff`ae7b398c)		0:019> t MpOavIDlRegisterServer+0x11af: 00007fff`ae7b393f ff151b130300	call	qword ptr [MpOav!DllRegis

as we can see in the above diagram, this call is to MPCLIENT!MpAmsiScan function. This is basically a function exported by windows defender's MPCLIENT.dll. So this means we have reached our destination.



Let's step over this function and inspect the return value since it is out of scope of this article to reverse engineer windows defender internals.

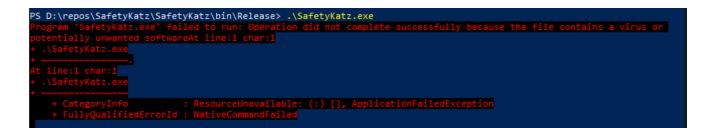
Registers	- ☆ ×	Disassembly		- \$	×	Command ×	Ŧ
Name	Value 🔺	Address: @\$scopeip		Follow current instruction		0:012> r_rax	
User		Address. Epscoperp					
rax	0x00000000000000000	00007fff`ae7b3931 498bd7	mov rdx, r15				
rbx	0x00007fffae7f3280	00007fff`ae7b3934 488b4948	mov rcx, qword ptr				
rcx	0x68ac636e4bc20000	00007fff`ae7b3938 488b808000000 00007fff`ae7b393f ff151b130300	mov rax, qword ptr call gword ptr [Mp0	<pre>lrax+80h] av!DllRegisterServer+0x324d0 (000007f</pre>			
rdx	0x0000000000000000000	00007fff`ae7b3945_8bf8	mov edi, eax	aviblikegislerserver+0x52400 (00007+			
rsi	0x000001cd21583dd0	00007fff`ae7b3947 488b0d32f90300		[MpOav!DllRegisterServer+0x40af0 (0)	81		
rdi	0x000000bf1244e668	00007fff`ae7b394e 85ff	test edi, edi				

According the above diagram, the return value we get is oxo. And there's no way to determine whether this is a indication of detection or not because windows documentation does not provide imformation about MpAmsiScan

Therefore we have to try some tricky methods to identify it.

First, im going to continue the exection.

as expected the result is,



Then we can place a breakpoint at the address where MpAmsiScan return and send some non-malicous input.

Weirdly enough, return value is same. So this function must be using an output parameter to pass the result of the scan, just like AmsiScanBuffer.

Can you remember that the third parameter to MpAmsiScan is a pointer to a local variable? Just in case, keep it's address in mind.

Somewhere down below, before the program generates an event saying safetykatz is malicious, return value or output parameter of MpAmsiScan must be accessed in order determine whether it's detected by windows defender or not.

Back to where we left off,

return value of MpAmsiScan is stored in edi register and function compares it with o after moving some value to rcx register.

00007fffae7b3945 8bf8 mov edi, eax 00007fffae7b3947 488b0d32f90300 mov rcx, qword ptr [MpOav!DllRegisterServer+0x40af0 (00007fffae7f3280)] MpOav!DllRegisterServer+0x11be: 00007fffae7b394e 85ff test edi, edi 00007fffae7b3950 7925 jns MpOav!DllRegisterServer+0x11e7 (00007fffae7b3977) [br=1]

if return value (edi) is greater than or equal to zero,

<pre>MpOav!DllRegisterServer+0x11e7:</pre>		
00007fffae7b3977 837c245401	cmp	dword ptr [rsp+54h], 1
00007fffae7b397c 0f94c0	sete	al
00007fffae7b397f 8886c8000000	mov	byte ptr [rsi+0C8h], al
00007fffae7b3985 8b442450	mov	eax, dword ptr [rsp+50h]
00007fff`ae7b3989 418906	mov	dword ptr [r14], eax

it sets value of third parameter (pointed by r14) to 1 and simply returns. Also note that return value is set to **edi**.

else if return value of MpAmsiScan (edi) is less than o,

MpOav!DllRegisterServer+0x11c2: 00007fffae7b3952 483bcb 00007fffae7b3955 7435 (00007fffae7b398c)	cmp je	rcx,rbx MpOav!DllRegisterServer+0x11fc
MpOav!DllRegisterServer+0x11c7: 00007fffae7b3957 f6411c01 00007fffae7b395b 742f (00007fffae7b398c)	test je	byte ptr [rcx+1Ch],1 MpOav!DllRegisterServer+0x11fc
MpOav!DllRegisterServer+0x11cd: 00007fffae7b395d ba11000000 00007fffae7b3962 448bcf 00007fffae7b3965 4c8d050c700300 (00007fffae7ea978)]	mov mov lea	edx,11h r9d,edi r8,[MpOav!DllRegisterServer+0x381e8
00007fffae7b396c 488b4910 00007fffae7b3970 e803f2ffff (00007fffae7b2b78)	mov call	rcx,qword ptr [rcx+10h] MpOav!DllRegisterServer+0x3e8
00007fffae7b3975	jmp	MpOav!DllRegisterServer+0x11fc

it checks validity of some data and calls a function and then returns after setting return value to that of MpAmsiScan stored in edi register, just like the previous one.

00007fffae7b398c	8bc7	mov	eax,	edi		
00007fffae7b398e	488b5c2440	mov	rbx,	qword	ptr	[rsp+40h]
00007fffae7b3993	488b6c2448	mov	rbp,	qword	ptr	[rsp+48h]
00007fffae7b3998	488b742458	mov	rsi,	qword	ptr	[rsp+58h]
00007fffae7b399d	4883c420	add	rsp,	20h		
00007fffae7b39a1	415f	рор	r15			
00007fffae7b39a3	415e	рор	r14			
00007fffae7b39a5	5f	рор	rdi			
00007fff`ae7b39a	6 c3	ret				

Because the return value we got from MpAmsiScan is oxo, execution path will be the first one we've discussed above.

There is something interesting that we havent discussed about that control flow path. There is a comparison of rsp+0x54 and 1. if that comparison is able to set zero flag, next instruction sets al register to 1.

in our case, rsp+0x54 is not equal to 1.

0:018> dd @rsp+0x54 L1 00000015`8864e564 00000000

which means, al wont be set to 1. If you can remember, rsp+0x54 is only accessed once, just after the call to LeaveCriticalState and that that is the only instruction that sets rsp+0x54 to 0x0. My guess is that this checks if function has entered the

LeaveCriticalSection block. It then sets [rsi+0C8h] (rsi == first parameter) to the value of al . Note that rsi+0xc8 should be set to zero in order for this function to be successful. We discussed rest of this block earlier.

after the function returns, we'll end up back at CAmsiAntimalware::Scan . Good news is, we dont need to read every instruction since we already know what we are looking for.



Above image shows how the call looks in decompiled pseudo code. return value of the callee is stored in local variable uVar2. However, we know this is not accurate because caller need to pass three args to the callee (we see none). That's not important to us though.



Here, the if confition only evaluate true when loc_rand() is equal to zero and a global variable is less than 5. loc_rand is basically the local variable where the random number was stored. Therefore this block is not going to execute.

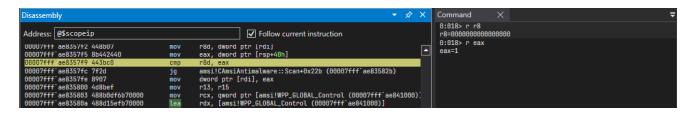


Above if condition checks if return value (stored in r14) is zero. In our case it is. we know that the third argument passed to the collee is the address of rsp+0x40 and was passed through r8.

below image shows disassembly of the above snippet

amsi!CAmsiAntimalware::Scan+0x1ed:			
00007fffae8357ed 4585f6	test	r14d,r14d	
00007fffae8357f0 757f	jne	amsi!CAmsiAntimalware::Scan+0x271	
(00007fffae835871)			
amsi!CAmsiAntimalware::Scan+0x1f2:			
00007fffae8357f2 448b07	mov	r8d,dword ptr [rdi]	
00007fffae8357f5 8b442440	mov	eax,dword ptr [rsp+40h]	
00007fffae8357f9 443bc0	cmp	r8d,eax	
00007fffae8357fc 7f2d	jg	amsi!CAmsiAntimalware::Scan+0x22b	
(00007fffae83582b)			

As shown above, mov r8d, dword ptr[rdi] moves value at address stored in rdi into r8 register. rdi stores the address of AMSI_RESULT enum passed down to CAmsiAntimalware::Scan method. it then moves rsp+0x40, output paramater we discussed earlier into eax register.



comparison instruction and jump instruction checks if value in **r8** (result) is greater than that of in **eax** (output parameter). jump wont be taken and execution will directed to the next mov instruction.

This is basically checking if current scan's result is greater than that of previous one.

amsi!CAmsiAntimalware::Scan+0x1fe:					
00007fffae8357fe 8907	mov	dword ptr [rdi],eax			
00007fffae835800 4d8bef	mov	r13, r15			
00007fffae835803 488b0df6b70000	mov	rcx,qword ptr [amsi!WPP_GLOBAL_Control			
(00007fffae841000)]					
00007fffae83580a 488d15efb70000	lea	rdx,[amsi!WPP_GLOBAL_Control			
(00007fffae841000)]					
00007fffae835811 483bca	cmp	rcx,rdx			
00007fffae835814 7451	je	amsi!CAmsiAntimalware::Scan+0x267			
(00007fffae835867)					
amsi!CAmsiAntimalware::Scan+0x216:					
00007fffae835816 f6411c04	test	byte ptr [rcx+1Ch],4			
00007fffae83581a 744b	je	amsi!CAmsiAntimalware::Scan+0x267			
(00007fffae835867)					
amsi!CAmsiAntimalware::Scan+0x21c:					
00007fffae83581c 4c894c2430	mov	qword ptr [rsp+30h],r9			
00007fffae835821 418d561f	lea	edx,[r14+1Fh]			
00007fffae835825 89442428	mov	dword ptr [rsp+28h],eax			
00007fffae835829 eb28	jmp	amsi!CAmsiAntimalware::Scan+0x253			
(00007fff`ae835853)					

In the above snippet it loads eax into [rdi], and value of r15 into r13 and compare some global variables related to WPP.

According to the decompiled snippet, this checks some global variables related to WPP tracer and if checks are valid, it jumps to a location in disassembly after setting rdx register to the address r14 + 1f. Well this has nothing to do with addresses eventhough the instruction is lea. r14 is oxo. therefore what this does is, it loads 0x1f into rdx register.

However, if we step through each instruction, **cmp rcx**, **rdx** will evaluate to oxo and the jump will be taken.

amsi!CAmsiAntimalware::Scan+0x267:				
00007fffae835867 813f00800000	cmp	dword ptr [rdi],8000h		
00007fffae83586d 7d50	jge	amsi!CAmsiAntimalware::Scan+0x2bf		
(00007fffae8358bf)				
amsi!CAmsiAntimalware::Scan+0x26f:				
00007fffae83586f eb34	jmp	amsi!CAmsiAntimalware::Scan+0x2a5		
(00007fff`ae8358a5)				

in the above snippet, dword value at address stored in **rdi** is compared to hex **0x8000**, decimal 32768. And this is exactly the same value <u>msdn</u> specifies in their documentation for **AMSI_RESULT** enum. quoting msdn,

'Any return result equal to or larger than 32768 is considered malware, and the content $% \left({{\left[{{\left({{{\left({1 \right)}} \right.} \right.} \right]}_{\rm{cont}}}} \right)$

should be blocked. An app should use ${\tt AmsiResultIsMalware}$ to determine if this is the case.'

next instruction is a jge and it essentially takes the jump if dword at address stored in rdi (AMSI_RESULT) is greater than or equal to 0x8000. if it is, it breaks from the loop.

In our case, value at address stored in **rdi** is less than 0x8000 so the jump won't be taken. Instead control flow will be redirected to

amsi!CAmsiAntimalware::Scan+0x2a5:				
00007fffae8358a5 488344246008	add	qword ptr [rsp+60h],8		
00007fffae8358ab 49ffc7	inc	r15		
00007fffae8358ae 4983c410	add	r12,10h		
00007fffae8358b2 4c3bbec0010000	cmp	r15,qword ptr [rsi+1C0h]		
00007fffae8358b9 0f820efeffff	jb	amsi!CAmsiAntimalware::Scan+0xcd		
(00007fffae8356cd)				

r15 is incremented by 1 and it is then compared to this->0x1c0, whose value is 1. if r15 is below that value, it will jump to the address where the loop begins.

Possibly, the loop is going through every registered anti-malware vendor's COM interface. Since I dont have any anti malware services installed in the VM, its going to loop only once. This also uncovers some details about CAmsiAntimalware class members. The loop terminates after loop iterator veriable being compared to this->0x1c0. Therefore this->0x1c0 is the value that indicates number of registered anti malware services or AMSI providers.



Now the question is, we just executed a malicous program and it just got flagged as AMSI_RESULT_NOT_DETECTED. But we still see powershell produces that red ugly output saying that it detected a malicous program.

And suprisingly, there's no call to AmsiResultIsMalware .

00007fffae8358c4 4c3baec0010000 00007fffae8358cb 732a (00007fffae8358f7)	cmp jae	r13,qword ptr [rsi+1C0h] amsi!CAmsiAntimalware::Scan+0x2f7
amsi!CAmsiAntimalware::Scan+0x2c	d:	
00007fffae8358cd 448bf3	mov	r14d,ebx
00007fffae8358d0 4d85e4	test	r12, r12
00007fffae8358d3 7428	je	amsi!CAmsiAntimalware::Scan+0x2fd
(00007fff`ae8358fd)		

First if condition checks if **r13** register is less than the number of providers (this->0x1c0). We saw that **r15**, which acts as the counter loaded into **r13** previously. What this is checking is that if anything malicous detected before going through all the providers.

Now it is time to conclude our assumptions on AmsiInitialize.

Amsilnitialize

The End

So yeah that's it for now... we explored AMSI in-depth in this article. In the next one, We will go through some common AMSI bypass techniques.

#Spread Anarchy!