Flare-On 10 Challenge 5: where_am_i

By Genwei Jiang (@binjo)

Overview

The where_am_i.exe program is a repurposed in-memory dropper, which is named as STONEBRIDGE in Mandiant, that executes the final payload in a newly created Explorer.exe process through <u>Asynchronous</u> <u>Procedure Call (APC)</u>. The STONEBRIDGE employs multiple staged shellcodes and different crypto algorithms to archive the execution of its payload. The repurposed where_am_i.exe is intended for the player to find the encrypted flag and the decryption routine, only debug skills and a bit of PE structure knowledge are required to solve the challenge.

As a typical Windows malware analysis workflow, we will analyze the sample using the <u>FLARE VM</u> of Windows 10 as our primary virtual machine. All the tools mentioned in this solution are available in the default installation.

Static Analysis

Before diving into executing the sample, it is generally better using static analysis tools to gain insights of the sample,

- <u>Detect It Easy</u>, for identifying PE architecture, packer, entropy, sections etc.
- · <u>CFF Explorer</u>, for viewing PE structure, extracting resource
- FLOSS, for extracting strings
- · <u>CAPA</u>, for inspecting capabilities of the sample

Based on the output of these tools, the sample appears to be a MFC application written in C++, that is signed and contains resources. The FLOSS extracted strings are not very interesting, but the CAPA tells of a capability of data encryption using RC4, RWX memory allocation and runtime linking. These capabilities ring a bell of possible shellcode execution.

The summary of identified capabilities as follows:

CAPABILITY	NAMESPACE
reference analysis tools strings encrypt data using RC4 PRGA contain a resource (.rsrc) section extract resource via kernel32 functions (6 matches) delete file get file attributes move file read .ini file set application hook get graphical window text (2 matches) allocate thread local storage allocate RWX memory terminate process query or enumerate registry value (2 matches) set registry value get kernel32 base address link many functions at runtime resolve function by parsing PE exports (2 matches)	anti-analysis data-manipulation/encryption/rc4 executable/pe/section/rsrc executable/resource host-interaction/file-system/delete host-interaction/file-system/meta host-interaction/file-system/move host-interaction/file-system/read host-interaction/gui/window/get-text host-interaction/gui/window/get-text host-interaction/process host-interaction/process/inject host-interaction/process/terminate host-interaction/registry host-interaction/registry/create linking/runtime-linking linking/runtime-linking

Figure 1. CAPA Summary

The -vv option of CAPA outputs the details of the matched rules, that includes the virtual address of interest.



Figure 2. CAPA RC4 rule match output

By looking at the disassembly or decompilation at the virtual address 0x448733, we can determine the code is indeed a RC4 decryption routine. Checking the cross reference to this address, we're lucky to find only one reference and further analyze the function can determine this is the 1st stage of the sample, at virtual address 0x448650. In essence, the sample loads and decrypts an embedded resource of type bitmap into executable heap memory. The sample reads the resource name from the file offset 0x3e0, a DWORD value 0x1. The sample reads two DWORD values, 0x6012adda and 0x1e8fc667 from the file offset 0x3f0. The DWORDs are part of a RC4 key. The RC4 key is 256 bytes long, with hex value of 67 c6 8f 1e da ad 12 60 00 00 ... (the remaining are all null bytes).

> 01	👒 010 Editor - C:\Users\flare\Desktop\where_am_i.exe																				
File	Edit	Sea	irch	Vie	ew	For	mat	Sc	ripts	s T	emp	lates	5 C	Debu	g	Tools	Wi	ndow	He	р	
	~ 📂	· ~ [,		ø	1	‰	Þ	Î	ľ) (ן ע	~	• 🏚	8 🤛	≯		i <u> </u>	
where	where_am_i.exe ×																				
	ð		2	3	4	5	6	7	8	9	Α	В	С	D	Е	F	0123	3456	789A	BCDE	F
03D0ł	ו <mark>: 00</mark>	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00					
03E0ł	n: 01	00	00	00	01	00	00	00	02	00	00	00	01	00	00	00					
03F0ł	ו: DA	AD	12	60	67	C6	8F	1E	C2	9A	EF	46	EF	ΒE	AD	BA	Ú	`gÆ.	.šï	Fï¾-	
0400ł	ı: 55	8B	EC	6A	FF	68	BA	BD	45	00	64	A1	00	00	00	00	Ū‹ì	jÿh°	½E.d	i	
0/10	50	00	FC	00	Λ1	10	٨٢	17	00	22	C5	50	0	45	ГЛ	64	Dfi	·, –	C 2Å	Ď EÂ	d

Figure 3. RC4 key at file offset 0x3f0

At this point, let's follow the call and jump into dynamic analysis.



Figure 4. First stage code in IDA

Dynamic Analysis

As we analyzed in static analysis, the sample allocates RWX memory and executes the next stage shellcode. One can use a debugger of choice to analyze the sample from the specific virtual address. But for this sample, we may first execute the sample and check logs in Process Monitor and Process Explorer, which both are generally useful for understanding the malware execution:

- Process Monitor, for monitoring file system, registry, process/thread activity and network connections
- Process Explorer, for displaying information of running process

Upon execution of the sample, a dialog box displayed with a message that appears the full path of the executable.





The events of Process Monitor do not look too interesting, except the creation of a new Explorer.exe process.

PID Operation axe 3780 g [®] Process Start	Path	Resu	lt ESS	Detail Parent PID: 3692, Comman	d line: "C:\Users\flare\Deskte	op\where_am_i.exe" , Current directory: C:\Us	ers\flare\Des						
5652 🖉 Process Start		SUCC	ESS	Parent PID: 3780, Comman	d line: Explorer.exe, Current	directory: C:\Users\flare\Desktop Environme	nt: =::=::\ALL						
Process Tree						- C	ı ×						
Only show processes still running	Only show processes still running at end of current trace												
Timelines cover displayed events	only												
Process	Description	Image Path	Life Time	Company	Owner	Command	^						
Isass.exe (612)	Local Security Aut	C:\Windows\syste	·	Microsoft Corporat	NT AUTHORITY\SYSTEM	C:\Windows\system32\lsass.exe							
fontdrvhost.exe (796)	Usermode Font Dr	C:\Windows\syste		Microsoft Corporat	Font Driver Host\UMFD-0	"fontdrvhost.exe"							
csrss.exe (512)	Client Server Runt	C:\Windows\syste		Microsoft Corporat	NT AUTHORITY\SYSTEM	%SystemRoot%\system32\csrss.exe Object[Jirect						
winlogon.exe (672)	Windows Logon A	C:\Windows\syste		Microsoft Corporat	NT AUTHORITY\SYSTEM	winlogon.exe							
fontdrvhost.exe (788)	Usermode Font Dr	C:\Windows\syste		Microsoft Corporat	Font Driver Host\UMFD-1	"fontdrvhost.exe"							
dwm.exe (984)	Desktop Window	C:\Windows\syste	·	Microsoft Corporat.	. Window Manager\DWM-1	"dwm.exe"							
Explorer.EXE (3692)	Windows Explorer	C:\Windows\Explo	·	Microsoft Corporat	DESKTOP-LV60CTUMare	C://Windows/Explorer.EXE							
SecurityHealthSystray.exe (548	Windows Security	C:\Windows\Syste	·	Microsoft Corporat	DESKTOP-LV60CTUMare	C:\Windows\System32\SecurityHealthSystem C:\Windows\System32\SecurityHealthSystem	.y.exe						
vmtoolsd.exe (5628)	Viviware Tools Cor	C:\Program Files\		Viviware, Inc.	DESKTOP-LV60CTUMare	C:\Program Files\Visware\Visware Tools\Vin	.tools						
200mit64.exe (5700)	Sysinternals Scre	C:\Users\flare\App	·	Sysinternals - ww	DESKTOP-LV60CTUMare	"C:\Users\flare\AppData\Local\Temp\zoomito	4.exe						
	Sysinternals Proc	C://loors/sysmiem		Sysinternals - ww	DESKTOP-LV60CTUMare	C:\Teele\eueinternals\procexp.exe							
procexpo4.exe (0804)	Windows Comman	C:\Windows\svete	•	Microsoft Corporat	DESKTOP-LV60CTU/flare	"C:\Windows\system32\cmd eve"							
conhost exe (4720)	Console Window	C:\Windows\syste		Microsoft Corporat.	DESKTOP-LV60CTU/flare	122\C:\Windows\system32\conhost eve 0v4							
CEE Explorer exe (3404)	Common File For	C:\Tools\Explorer		Daniel Pistelli	DESKTOP-LV60CTU/flare	"C:\Tools\Explorer Suite\CEE Explorer exe" (:\Use						
Procmon exe (4304)	Process Monitor	C:\Tools\sysintern		Sysinternals - ww	DESKTOP-LV60CTU/flare	"C:\Tools\sysinternals\Procmon exe"							
Procmon64.exe (3972)	Process Monitor	C:\Users\flare\App		Sysinternals - ww	DESKTOP-LV6OCTU/flare	"C:\Users\flare\AppData\Local\Temp\Procmo	n64.e						
where am i.exe (3780)	XtuService	C:\Users\flare\Des		Intel(R) Corporation	DESKTOP-LV6OCTU\flare	"C:\Users\flare\Desktop\where am i.exe"							
Explorer.exe (5652)	Windows Explorer	C:\Windows\SysW		Microsoft Corporat	DESKTOP-LV6OCTU\flare	Explorer.exe							
							_						
							~						
<	< <						>						
Description: Windows Explorer													
Company: Microsoft Corporation													
Path: C:\Windows\SysWOW6	64\Explorer.exe												
Commenter Fundamente													



The where_am_i.exe process exits shortly, so we can attach a debugger to the newly created Explorer.exe process and poke around a bit. As the message box pops, the call stack reveals the thread originated from the ntdll!RtlDispatchAPC call, the virtual address starts with 0x11 and 0x54 are most interesting, as these might be the injected payload or another shellcode.

I prefer using WinDbg and the following is the call stack output of command $\sim k$ when attached.

0:001> ~*k	
0 Id: 1614.1450 Susp	end: 1 Teb: 0022e000 Unfrozen
<pre># ChildEBP RetAddr</pre>	
00 000df490 75ee4a43	win32u!NtUserWaitMessage+0xc
01 000df4d0 75ee4934	user32!DialogBox2+0x102
02 000df500 75f41aeb	user32!InternalDialogBox+0xd9
03 000df5cc 75f40881	user32!SoftModalMessageBox+0x72b
04 000df728 75f41347	user32!MessageBoxWorker+0x314
05 000df7b0 75f4117e	user32!MessageBoxTimeoutW+0x197
06 000df7e4 75f40e95	user32!MessageBoxTimeoutA+0xae
07 000df804 005410da	user32!MessageBoxA+0x45
WARNING: Frame IP not in	any known module. Following frames may be wrong.
08 000df894 005417f7	0x5410da
09 000df8d4 005418d9	0x5417f7
0a 000df8e8 00111293	0x5418d9
0b 000df92c 00110514	0x111293
0c 000df944 7779d5b9	0x110514
0d 000df99c 77784e4b	ntdll!RtlDispatchAPC+0x615a9
0e 000dfd04 77776391	ntdll!KiUserApcDispatcher+0x4b
0f 000dfd10 00000000	ntdll!LdrInitializeThunk+0x11
# 1 Id: 1614.d0c Suspe	nd: 1 Teb: 00249000 Unfrozen
# ChildEBP RetAddr	
00 0048ff40 777bdce9	ntdll!DbgBreakPoint
01 0048++70 774300c9	ntdll!DbgUlRemoteBreakin+0x39
02 0048++80 77777b1e	KERNEL32!BaseThreadInitThunk+0x19
03 0048++dc 77777aee	ntd11!Rt1UserIhreadStart+0x2f
04 0048ttec 00000000	ntdll!_RtlUserIhreadStart+0x1b

Table 1. Call stack of threads

Checking the relevant disassembly code at virtual address 0x5410da and memory of the parameter reveals an interesting hint, Heard there's RC6 somewhere, is that true?. Poking around those virtual addresses starts with 0x11 and 0x54, you may find codes likely a shellcode starting from 0x110000. Keep these virtual addresses and offsets in mind, as we will observe them in a debugger.

0:003> ut	0x5410da		
005410c1	56	push	esi
005410c2	ff1510e05400	call	dword ptr ds:[54E010h]
005410c8	8b3528e15400	mov	esi,dword ptr ds:[54E128h]
005410ce	6a00	push	0
005410d0	68942c5500	push	552C94h
005410d5	57	push	edi
005410d6	6a00	push	0
005410d8	ffd6	call	esi

0:003> ln poi(54E128h) Browse module Set bu breakpoint																
(75f40e50) user32!MessageBoxA							△	(75f40ea0) user32!MessageBoxExA					essageBoxExA			
0:003> db	55	2C94	4h													
00552c94	46	4c	41	52	45	2d	4f	4e-20	31	30	3a	20	77	68	65	FLARE-ON 10: whe
00552ca4	72	65	5f	61	6d	5f	69	3f-00	00	00	00	66	6c	61	72	re_am_i?flar
00552cb4	65	00	00	00	43	3a	5c	55-73	65	72	73	5c	50	75	62	eC:\Users\Pub
00552cc4	6c	69	63	5c	00	00	00	00-48	65	61	72	64	20	74	68	lic∖Heard th
00552cd4	65	72	65	27	73	20	52	43-36	20	73	6f	6d	65	77	68	ere's RC6 somewh
00552ce4	65	72	65	2c	20	69	73	20-74	68	61	74	20	74	72	75	ere, is that tru
00552cf4	65	3f	00	00	c0	00	00	00-00	00	00	00	00	00	00	00	e?

Table 2. Disassembly code and memory content

At this point, we've found out:

- There's RC4 involved in decrypting the 1st stage shellcode
- Process injection of final payload in Explorer.exe
- Likely has another stage of shellcode
- Found a hint: Where is the RC6

This is a perfect scenario for using Time Travel Debugging(TTD), that once a sample execution recording trace file is created, we can play it back and forth within WinDbg. There are pros and cons of using TTD:

- Pros
 - The utility is built into the system of Windows 10 and above, namely tttracer.exe
 - Step back and forth without executing the sample again, great for collaboration with colleagues
- Cons
 - Slow down execution
 - Large disk storage may be required

While there's less help information of tttracer.exe online, the command line options are similar to the ones used in standalone utility of <u>ITD</u>. We can generate a full trace of the sample by using tttracer.exe -children where_am_i.exe, run as an administrator. The -children option instructs the tracer to follow the child process and generate the trace file. The output files are where_am_i01.run, explorer01.run and corresponding log files.



Figure 7. Output of tttracer.exe

Having the . run tracing file generated and opened in WinDbg, all the live debug commands are available as one may use to be. We can set breakpoints at specific API, virtual address or simply g to the previous address we found.

Google Cloud

FLARE

Figure 8. First stage shellcode entry point

Shortly after step over some instructions of the 1st stage shellcode, it appears the control flow is obfuscated and offsets are calculated using sequences of opcodes like mov, add, sub, xor and etc. It becomes tedious to single step in/over opcodes.

Google Cloud

FLARE

Figure 9. Graph view of the shellcode

Tracing into/over commands like pa would be helpful in certain cases, the <u>debugger data model</u> and ability of <u>using LINQ with the debugger objects</u> provides much more power to the user. For example, one can query the api usage of kernel32!VirtualAllocStub like the following:

```
dx -g @$cursession.TTD.Calls("kernel32!VirtualAllocStub").Select(r > new {TimeStart = r.TimeStart,
TimeEnd = r.TimeEnd, Function = r.Function, FunctionAddress = r.FunctionAddress, ReturnAddress
= r.ReturnAddress, ReturnValue = r.ReturnValue})
```

Command >	×									
0:000> dx -g	@\$cursession.T	TD.Calls("kerne	el32!VirtualAllocStub").Select(r => new {TimeStart = ı	r.TimeStart, TimeEnd	= r.TimeEnd, Function =				
-	(<u>+</u>) <u>TimeStart</u>	(<u>+</u>) <u>TimeEnd</u>	(<u>+</u>) <u>Function</u>	(<u>+</u>) <u>FunctionAddress</u>	(<u>+</u>) <u>ReturnAddress</u>	(<u>+</u>) <u>ReturnValue</u>				
= <u>[0x0]</u> - = <u>[0x1]</u> -	<u>143:451</u> 22B:999	<u>145:8</u> <u>22D:8</u>	kernel32!VirtualAllocStub kernel32!VirtualAllocStub	0x7742f9f0 0x7742f9f0	0x448ae3 0x107a11b2	0x107a0000 = 0x107b0000 =				
1011 228:999 220:8 Kernel32:VirtualAllocStub 0x77424940 0x107a11b2 0x107b0000 0:000> !tt 143:451										
0:000> dd esp 0019fdec 004 0019fdfc 000	o 18 448ae3 00000000 000040 00484a38	00006c44 00003 e6490001 00000	3000 5c44							

Figure 10. LINQ query of VirtualAllocStub call

The example shows the occurrence of allocating RWX memory for the first stage shellcode of size 0x6c44, and the ReturnValue is 0x107a0000.

Inspired by the Axel Souchet's <u>codecov plugin</u>, I've included the script for API tracing in Appendix, which helps to get all of the APIs called within specified address range. The script in essence queries the call instructions and finds the referenced API name. Despite the obfuscated control flow hinders analyzing, we can infer the full code logic based on the API calls, which loads another resource and injects into the Explorer.exe process through kernel32!QueueUserAPC call. The TimeStart values are useful for time traveling back and forth.

0:000> dx -g @\$curprocess.Modules[0].TraceInRange(0x107a0000	, 0x6c44)			
	(<u>+</u>) <u>TimeStart</u>	<u>RVA</u>	Address	<u>Called</u>
[Ava] + KERNEL221SatEnnonMadoStub (77420cb0)	212+56	0×674	0x107-0674	KEDNEL 221SetEpperMedeStub (77420cb0)
[0x0] . KERNEL32:SetEITOTMODeStub (77450000) [0x1] . KERNEL32:SetEITOTMODeStub (77436100)	213.30	0x074	0x107a0074	KERNEL32(SetErrorModeStud) (77430000)
[0x1] : KENNEL32:GetSystemDerduitLangiDStud (77425120) [0x2] : KEDNEL32:GetAtomNamoA (774632b0)	217.0	0x610	0x107a00TC	KERNELSZ:GELSYSTEMDETAUITLANGIDStud (77425120)
[0x2] : KERNEL32:GetAcominameA (77403000)	220.20	0x12o4	0x107a3744	KERNEL321GindBocouncoA (77403306)
[0x4] : KERNEL32:FillukeSourceA (7742200)	220.100	0x1204	0x107a12e4	KERNEL 321 Sizeof PosourceStub (77422600)
[0x4] . KERNEL32!SIZEOTRESOURCEStub (77430020)	22D. JFA 22P. 7A2	0x5277	0x107d5277	KERNEL3211 and Pasour costub (77430020)
[0x5] : KERNEL32:LoadResourceStub (77426270)	22D.7AZ	0x5uss 0x62cd	0x107a5u88	KERNEL32!LoakRosounceStub (77426070)
[0x7] : KERNEL32:LOCKRESOULCEStub (77421970)	220.54D 220.009	0x03cu	0x107a03cu	KERNEL 321V(intualAllocStub (77421970)
[0x2] : KERNEL32(GotProcessHearStub (77421910)	4E2:22PD	0x5020	0x107a11ac	KERNEL321Got Drocoss Hoop Stub (77421910)
[0x0] : ntd111Pt1011ocotoHoop (77755dc0)	452:2200	aveo/h	0x107a592a	ntd]][Pt]A]]ocatoHoan (77755dc0)
[0x3] : KERNEL22[GotDrococcHoapStub (7742f0b0)	2520:845	0x53940	0x107a3340	KERNEL 22 Loot Drocosc Hoap Stub (7742f0h0)
[0x4] : KERNEL32:GetProcessHedpStub (7742:500)	2520.045	0x6409	0x107a0409	KERNEL 32 HoopEnceStub (77421900)
[0xc] : KEDNEL 221SloopStub (77421520)	2522.61	0x0410	0x107a0410	KERNEL321SloopStub (7742E350)
[0xd] : KERNEL3210popMutox4 (77433a6)	2524:04	0x2400	0x107a2400	KERNEL 3210popMutovA (77431340)
[Ovo] + KERNEL32:OpenHuteKA (77422000)	2529.10PF	0x5ca0	0x107a5ca6	KERNEL 32 Coosto Docosco (1/422000)
[0x6] : KERNEL32:CreateProcessAstub (77444110)	<u>3F20.1903</u>	0x0d59	0x107a0a59	KERNEL32:CreateProtessAstub (77444110)
[0x10] : KERNEL32!VITtudIATIOCEXStub (77446570) [0x10] : KERNEL32!WintoDiscossMomonyStub (774465c0)	4110:104 4115:6P	0x1490	0x10/d1d90	KERNELS2:VITtudIAIIOCEXStub (77446570)
[0x10] . KERNEL32[WITCEPFOCESSMEINOFyScub (774405c0)	4117.00	0x0558	0x107a0558	KERNEL 3210uouollcopADCStub (77420700)
[0x12] . KERNEL32:QueueoSerAPC3Cub (77423750)	4120.44	0x4000	0x107a4000	KERNEL32:QueueoserArc3cub (77423750)
[0x12] : KERNEL32:RESumerInFeduStub (77431190) [0x12] : KERNEL32:GotModuloEiloNomoAStub (774314d0)	4120:25	0x3001	0x10735001	KERNELS2:Resumerin/edustub (77451190)
[0x14] : ADVADI321TeitializeCocupityDecopipton (76076260)	412A.2C	0x2045	0x107a0eca	ADVADI321InitializaCocupityDecopinton (76076260)
[0x14] : ADVAPIS2:InitializeSecurityDescriptor (76076110)	4126.90	0x2943	0x107d2945	ADVAPI32!InitializeSecurityDescriptor (76071300)
[0x15] : ADVAP152:SetSeturityDescriptorDati (7607110)	4125.00	0x5792	0x10/d5/92	KERNEL22161constub (774215c0)
[0x10] : KERNELS2:SIEEPSCUD (77431500)	4126:150	0x0509	0x107a0509	KERNEL32:SIEEPSLUD (77431340)
[0x1/] . KENNELSZ:CFEALEFILEA (//433/00) [0x10] . KENNELSZ:UmitoFilo (77422650)	47EB:00	OVEREE	0x107a2500	KERNELSZ:CFEdleFIIEA (77433700)
[0x10] : KERNEL32(WFILEFILE (7/433050) [0x10] : KERNEL32(ClassHandle (77433680)	47EE:30 47EE:1E	0x3505	0x107a3565	KERNEL221ClassHandla (77422E80)
[0x19] : KENNEL32:CIOSEMANUIE (7/433580) [0x1a] : KENNEL32:EvitEncessesImplementation (77435040)	4752+22	0x4532	0x10/a4532	KERNELSZ:CIUSendiule (77433580)
<pre></pre>	4/12:23	0X2006	0X10/02006	NERNELSZ:EXILPROCESSIMPlementation (77435940)

Figure 11. API trace list of 1st stage shellcode

The resource appears to be a bitmap of name as DWORD value 0x2, that is specified in the file offset 0x3e8.

Time travel to the moment of writing shellcode into target process through kernel32!WriteProcessMemory call, we will see the target remote address is 0x110000, local buffer

address is 0x107b0031 and the size is 0x1512f. The address 0x107b0000 is allocated RWX memory from the previous VirtualAlloc call, and appears in the next stage shellcode.

Command X
0:000> !tt 411F:6B
Setting position: 411F:6B
(a68.19f0): Break instruction exception - code 80000003 (first/second chance not available)
Time Travel Position: <u>411F:6B</u>
eax=0000010c ebx=124d1888 ecx=107b0031 edx=00110000 esi=00000000 edi=124d19a5
eip=107a6538 esp=0019fb28 ebp=0019fe08 iopl=0 nv up ei ng nz ac pe cy
cs=0023 ss=002b ds=002b es=002b fs=0053 gs=002b ef1=00000297
107a6538 ff1580834700 call dword ptr [where_am_i+0x78380 (00478380)] ds:002b:00478380={KERNEL32!WriteProcessMemoryStub (774465c0)}
0:000> dd esp 18
0019fb28 0000010c 00110000 107b0031 0001512f
0019fb38 0000000 77782bbc 00000000 536cd652
0:000> db 107b0031 120
107b0031 eb 22 eb 19 81 c1 56 d2-72 a6 81 f1 f8 c4 f0 24 ."V.r\$
107b0041 e9 c5 00 00 00 eb 06 04-30 02 c1 eb 5f 39 0e e909
0:000> u 107b0031
107b0031 eb22 jmp 107b0055
107b0033 eb19 jmp 107b004e
107b0035 81c156d272a6 add ecx,0A672D256h
107b003b 81f1f8c4f024 xor ecx,24F0C4F8h
107b0041 e9c5000000 jmp 107b010b
107b0046 eb06 jmp 107b004e
107b0048 0430 add al,30h
107b004a 02c1 add al,cl
0:000> db 107b0000
107b0000 0a 77 65 6c 63 6f 6d 65-5f 6d 61 69 6e 00 00 00 .welcome_main
107b0010 00 00 00 00 01 31 00-00 00 2f 51 01 00 5c 5c1/Q\
107b0020 2e 5c 70 69 70 65 5c 77-68 65 72 65 61 6d 69 00 .\pipe\whereami.
107b0030 00 eb 22 eb 19 81 c1 56-d2 72 a6 81 f1 f8 c4 f0"V.r
107b0040 24 e9 c5 00 00 00 eb 06-04 30 02 c1 eb 5f 39 0e \$9.
107b0050 e9 f7 01 00 00 e9 d6 01-00 00 81 c0 2c b5 9f c7
107b0060 81 f0 de 8a 3a 7c 51 b9-f8 c7 56 98 81 c1 39 c6: QV9.
107b0070 a1 47 81 f1 59 c7 a3 03-81 e9 65 60 f8 a4 eb 1d .GYe`

Figure 12. WriteProcessMemory

By checking the memory read and write access at address 0x107b0000, it appears at the beginning of execution the content was indeed a copy of resource 0x2, at the end of execution the content is decrypted as above.

Time travel back to the last writing operation on the address 0x107b0000, and checking the return address of the last call, to identify if there is a decrypt routine.

Command 0:000> dx @\$memory(0x107b0000, 0x107b0000+1, "w").Last().TimeStart.SeekTo() (a68.19f0): Break instruction exception - code 80000003 (first/second chance not available) Time Travel Position: 4F1:1F5E @\$memory(0x107b0000, 0x107b0000+1, "w").Last().TimeStart.SeekTo() 0:000> k # ChildEBP RetAddr WARNING: Frame IP not in any known module. Following frames may be wrong. 00 0019fde4 107a05d2 0x107a3e6b **01** 0019fdfc 107a69e4 0x107a05d2 <u>02</u> 0019fe28 00448728 0x107a69e4 <u>03</u> 0019fe44 00452894 where_am_i+0x48728 <mark>04</mark> 0019fe84 00452a39 where_am_i+0x52894 05 0019fec8 00452b80 where_am_i+0x52a39 06 0019fef0 0044831e where_am_i+0x52b80 07 0019ff70 774300c9 where_am_i+0x4831e <u>08</u> 0019ff80 77777b1e KERNEL32!BaseThreadInitThunk+0x19 ntdll! RtlUserThreadStart+0x2f 09 0019ffdc 77777aee 0a 0019ffec 00000000 ntdll! RtlUserThreadStart+0x1b 0:000> ub 0x107a05d2 107a05b8 8d4cd016 lea ecx,[eax+edx*8+16h] 107a05bc 53 push ebx jmp 107a05bd e9b60a0000 107a1078 107a05c2 81c639ff2be8 add esi,0E82BFF39h 107a05c8 8b0c2e mov ecx,dword ptr [esi+ebp] 107a05cb 5e рор esi 107a05cc 51 push ecx 107a05cd e8f3230000 call 107a29c5 0:000> g- 107a05cd Time Travel Position: <u>4EF:26E</u> eax=00015160 ebx=124d1888 ecx=107b0000 edx=108f3d00 esi=00000000 edi=124d19a5 eip=107a05cd esp=0019fdec ebp=0019fdfc iopl=0 nv up ei pl nz ac po cy cs=0023 ss=002b ds=002b es=002b fs=0053 gs=002b efl=00000213 107a29c5 107a05cd e8f3230000 call 0:000> dd esp 14 0019fdec 107b0000 00015160 108f3d00 108f3d00 0:000> db 107b0000 l10 107b0000 ce 4b a4 c5 1b d7 27 02-ed d6 bd 76 97 b1 29 26 .K....'...v..)& 0:000> db 108f3d00 l10 108f3d00 a2 15 17 72 1f 01 bb 28-96 44 b2 96 66 24 b2 cd ...r...(.D..f\$.. 0:000> p Time Travel Position: <u>3F20:833</u> eax=00015160 ebx=124d1888 ecx=00015160 edx=5bf6ba7e esi=00000000 edi=124d19a5 eip=107a05d2 esp=0019fdec ebp=0019fdfc iopl=0 nv up ei pl zr ac pe cy cs=0023 ss=002b ds=002b es=002b fs=0053 gs=002b ef1=00000257 107a05d2 83c40c add esp,0Ch 0:000> db 107b0000 l10 107b0000 0a 77 65 6c 63 6f 6d 65-5f 6d 61 69 6e 00 00 00 .welcome_main... 0:000> db 107b0010 00 00 00 00 00 01 31 00-00 00 2f 51 01 00 5c 5c1.../Q.../ 107b0020 2e 5c 70 69 70 65 5c 77-68 65 72 65 61 6d 69 00 .\pipe\whereami. 107b0030 00 eb 22 eb 19 81 c1 56-d2 72 a6 81 f1 f8 c4 f0 ..."....V.r..... 107b0040 24 e9 c5 00 00 00 eb 06-04 30 02 c1 eb 5f 39 0e \$.....9. 107b0050 e9 f7 01 00 00 e9 d6 01-00 00 81 c0 2c b5 9f c7 107b0060 81 f0 de 8a 3a 7c 51 b9-f8 c7 56 98 81 c1 39 c6: Q...V...9. 107b0070 a1 47 81 f1 59 c7 a3 03-81 e9 65 60 f8 a4 eb 1d .G..Y....e`.... 107b0080 81 f0 40 38 ca d8 81 c0-d2 be f9 54 81 c0 71 2b ...@8.....T..q+

Figure 13. Resource decryption in TTD

Looks like we found a routine of decrypting at virtual address 0x107a29c5, would this be the RC6 one referenced in the hint?

Checking the heap memory write access at address 0x108f3d00, it appears the call originated from 0x107a5832.

Google Cloud

FLARE

Command \times		
0:000> dx @\$memory(0x108 (a68.19f0): Break instru Time Travel Position: 4E	Bf3d00, 0x108f3d00+1, "w").Where(r => r.Value != 0).First().TimeStart.SeekTo() uction exception - code 80000003 (first/second chance not available) E <u>5:1C8</u>	
@\$memory(0x108+3d00, 0x1	108+3d00+1, "w").Where(r => r.Value != 0).First().fimeStart.Seeklo()	
0:000> r		
eax=et0001Da eDx=1240188	38 eCX=108T5000 e0x=0/e15163 eS1=00000000 e01=12401935	
$e_1p_{=10733601} e_{sp_{=0019100}}$	$J_{1} = c_{1} = c_{1$	
107a386f 8911	m_{0} dword ntr [ecx] edx ds:002b:108f3d00=000002020	
0:000> k	more anota per [cox];cax astoczoszoo cococco	
# ChildEBP RetAddr		
WARNING: Frame IP not in	n any known module. Following frames may be wrong.	
<u>00</u> 0019fde8 107a5837	0x107a386f	
<u>01</u> 0019fdfc 107a69e4	0x107a5837	
<u>02</u> 0019fe28 00448728	0x107a69e4	
<u>03</u> 0019fe44 00452894	where_am_i+0x48728	
<u>04</u> 0019fe84 00452a39	where_am_i+0x52894	
05 0019+ec8 00452b80	where_am_1+0x52a39	
<u>06</u> 0019+e+0 0044831e	where_am_1+0x52080	
$\frac{07}{08}$ 00197770 77430009	Where_am+/0X4831e	
$\frac{08}{09}$ 00191180 77777299	NENNELSZ: DASETII Edulitu (Inuiktoli) htdlil DtilkonTheondStarti0v2f	
03 0019ffec 0000000	ntdl1:ntl0seThreadStarttex1h	
0:000> ub $0x107a5837$		
107a5813 81eae4e9b31f	sub edx.1FB3E9E4h	
107a5819 81ea360cb185	sub edx,85B10C36h	
107a581f 81c2ec739ac9	add edx,0C99A73ECh	
107a5825 e9cab5ffff	jmp 107a0df4	
107a582a b9dd34a760	mov ecx,60A734DDh	
107a582f eb45	jmp 107a5876	
107a5831 51	push ecx	
107a5832 e8a7f5ffff	call 107a4dde	
0:000> g- 10/a5832		
lime Travel Position: 4E		
edx=108T3000 eDx=1240188		
$e_1p_{=10733632} e_{sp_{=0019101}}$	$10 = 00^{-1} = 00^{-1} = 0$ $10 = 0^{-1} = 0^{$	
107a5832_e8a7f5ffff	call 107a4dde	
0:000> dd esp 14		
0019fdf0 baadbeef 108f3	3d00 108f3d00 0019fe28	

Figure 14. Check origin of memory write in TTD

Tracing into the call would be useful to understand the code logic, be alarmed the output may overflow the maximum lines of WinDbg UI, best redirect the output into a local file instead.

The call 0x107a4dde appears to be responsible for initializing the buffer pointed by address 0x108f3d00, using the parameter 0xbaadbeef.

Command	Notes 🔅	<							
0:000> .logopen Opened log file	c:\Users\ 'c:\Users	flare\Desktop\tra \flare\Desktop\tra	e-107a4dde.t ce-107a4dde.	xt txt'					
0:000> ta 107a5 ta 107a5837	837	•		// trace	until the	return	address of	the	call
eax=108f3d00 eb:	x=124d1888	ecx=baadbeef edx	00000000 esi	=00000000 edi=	=124d19a5				
eip=107a4dde es	p=0019fdec	ebp=0019fdfc iop	.=0 n	v up ei pl nz	na pe nc				
cs=0023 ss=002	b ds=002b	es=002b fs=005	gs=002b	efl=	00000206				
107a4dde 55		oush ebp							
				1		1.60			
10/a3860 801428	v-104d1000	nov eax, aword	ptr [eax+ebp] ds:0020:0019	9+ac4=b/e15	163			
eax=TTTTTTUC eD	X=12401888 n-0010fdh8	ecx=108+3000 eux	-0 est	=000000000 eul=	=12401985				
cs=0023 ss=002	b ds=002b	es=002b fs=005	0 II gs=002h	ofl=	-00000282				
107a386e 58	0 45 0020	op eax	65 0020	011	00000202				
eax=b7e15163 eb	x=124d1888	ecx=0000001 edx	108f3d00 esi	=00000000 edi=	124d19a5				
eip=107a3340 es	p=0019fdbc	ebp=0019fde8 iop	.=0 n	v up ei ng nz	na pe nc				
cs=0023 ss=002	b ds=002b	es=002b fs=005	gs=002b	efl=	00000286				
107a3340 0345f4		add eax,dword	ptr [ebp-0Ch] ss:002b:0019	fddc=9e377	9b9			
\cdots	v-00000001	00x-109f2d00 odv	b7015162 oci	-00000000 odi-	124410-5				
ein=107a5384 es	n=0019fdh8	etn=0019fde8 ion	=0 n	vun ei nl nz	ac ne nc				
cs=0023 ss=002	b ds=002b	es=002b fs=005	0 II gs=002h	efl=	=00000216				
107a5384 395de8		cmp dword ptr	[ebp-18h],eb	x ss:002b:0019)fdd0=00000	001			
eax=ef0001ba eb	x= <mark>00000024</mark>	ecx=108f3d00 edx	b7e15163 esi	=00000000 edi=	=124d19a5				
eip=107a5387 es	p=0019fdb8	ebp=0019fde8 iop	.=0 n	v up ei ng nz	ac pe cy				
cs=0023 ss=002	b ds=002b	es=002b fs=005	gs=002b	efl=	00000297				
107a5387 5b		oop ebx							
eax=ef0001ba eb	x=124d1888	ecx=108f3d00 edx	b7e15163 esi	=00000000 edi=	=124d19a5				
eip=10/a5388 es	p=0019+dbc	epp=0019+de8 iop.	.=0 n'	v up ei ng nz	ac pe cy				
10725388 0f83fb	us=002D foffff	es=0020 TS=005.	82=007D	et1=	-00000297 [hr=0]				
10,00000000000	i e i i i i	10/05205			[01-0]				

Figure 15. Trace log of 0x107a4dde

While the log file will be large, you may find some constant value of 0xb7e15163 and 0x9e3779b9 referenced at the beginning, and the loop pattern of round counter 0x24. By googling these constants, hopefully you find this <u>rc6_initl code snippet</u> a good reference. I'm no crypto expert, but the function of 0x107a4dde looks like a variant of RC6 key initializer, the key is the dword value 0xbaadbeef read from file offset 0x3fc. The function at virtual address 0x107a29c5 is the RC6 decrypt routine.

At this point, we've found out:

- The RC6 initialize routine at virtual address 0x107a4dde
- The RC6 decrypt routine at virtual address 0x107a29c5
- The RC6 key is 0xbaadbeef read from the file offset 0x3fc

Now we can turn to the trace file of Explorer.exe to figure out if the flag is hidden there.

\Users\flare\Desktop	explorer01.run	- WinDbg 1.2308.200	02.0												
ile Home	View	Breakpoints	Time Travel	Model	Scripting	Source	Memory	Com	nmand	_					
Disassembly							▼ \$?	X Com	mmand >	< 1					
Address: @\$scope:	p		√ Fo	llow current in	struction			0:0	000> !tt 0		hanimaina of	****			
00110000 ob33	im	00110034						Set	etting posit	ion: 4165:0	0	che crace			
00110000 eb22	jiin	00110024						(e1	f0.fec): Br	eak instruc	ction exceptio	on - code 80000	003 (first/se	cond chance	not available)
00110000 eb22	im	00110024						Tir	me Travel P	osition: 41	165:0				
00110002 eb19	jmp	0011001D						eax	ax=00ef7ac0	ebx=002b000	00 ecx=0000000	00 edx=00000000	esi=00000000) edi=000000	00
00110004 81c1	6d272a6 add	d ecx, 0A67	72D256h					eip	p=77776380	esp=000dfd1	14 ebp=0000000	00 iopl=0	nv up ei p	ol nz na po i	nc
0011000a 81f1	F8c4f024 xo	r ecx, 24F6	9C4F8h					CS:	=0023 ss=0	02b ds=002	2b es=002b f	s=0053 gs=002	b	ef1=000002	02
00110010 e9c5	900000 jmp	001100DA						nto	dil!Ldrinit	lalizeThunk	k:				
00110015 eb06	jmp	0011001D						0.0	000 g 1100	00	mov eur,	ear			
00110017 0430	ado	d al, 30h						U.C	mo Travol E	osition: 41	167.73				
00110019 0201	ado	ai, ci						eat	x=000df968	ebx=002b306	00 ecx=0011000	0 edx=00000000	esi=0000000	edi=000df9	84
00110010 005†	Jut	dword ptr	[ori] ory					eit	p=00110000	esp=000df94	48 ebp=000df99	c iopl=0	ny up ei p	ol zr na pe	nc
00110010 3900	10000 im	00110218	[est], ecx					CS:	=0023 ss=0	02b ds=002	2b es=002b f	s=0053 gs=002	b	ef1=000002	46
00110011 c5174	10000 jm,	00110210						00:	110000 eb22		jmp 0011	.0024			
00110029 81c0	cb59fc7 add	d eax. 0C79	9FB52Ch					0:0	000> k						
0011002f 81f0	le8a3a7c xor	eax, 7C3/	ASADEh					#	⊧ ChildEBP R	etAddr					
00110035 51	pus	sh ecx						WAR	RNING: Fram	e IP not ir	n any known mo	dule. Followin	g frames may	be wrong.	
00110036 b9f8	75698 mov	v ecx, 9856	5C7F8h					00	000df944 7	779d5b9	0x110000				
0011003b 81c1	39c6a147 add	d ecx, 47A1	LC639h					01	00001990	7784e4b	ntdll!KtlDis	patchAPC+0x615	a9 2014		
00110041 81f1	59c7a303 xo	r ecx, 3A30	C759h					02	00001004 /	///6391	ntdll!kiUser	ApcDispatcher+	0X40		
00110047 81e9	5560f8a4 sut	b ecx, ØA4F	-86065h					05	00001010 0	0000000	incuti : curini	CIAIIZernunk+0			
0011004d ebid	Jul	0011006C													
00110047 81704	1038cad8 x0	eax, 0080	.A3840n												
00110055 81C0	120er954 aut	d eax, 54rs	9DEU20												
00110050 8100	920853 20	a eax, 57AL	82CC9b												
00110001 0110	330000 imr	00110408	200511												
0011006c 81c1	51137b55 add	d ecx, 5578	31361h												
00110072 eb1c	jmp	00110090													
00110074 81f0	Fb054c2c xor	r eax, 2C40	C05FBh												
0011007a eb09	jmp	00110085													
0011007c 41	ind	c ecx													
0011007d 3007	xor	r byte ptr	[edi], al												
00110071 52	pus	sn edx													
00110080 0991		00110116	TRECH												
00110085 81T0	100000	- edx, 24//	DEPCH												
00110090 6942	300000 Jill	00110346													
00110095 81f0	2f6c81b4 xor	r eax. 0B48	316C2Fh												
0011009b 81c0	27d74f4d add	d eax, 4D4F	D727h												
001100a1 81f0	b7c3b25 xor	eax, 2538	37CCBh												
001100a7 81c0	1386f3dd add	d eax, ODDF	-38643h												

Figure 16. Shellcode entry point in Explorer.exe

While the shellcode is similar to the 1st stage one that obfuscated control flow hinders the analysis, with the insight gained from static analysis, the kernel32!VirtualAllocStub call is a good back tracing point.

Command \times														
9:000> dx -g @\$calls("kernel32!VirtualAllocStub")														
(<u>+</u>) EventT	ype (<u>+</u>) Thread	Id (<u>+) UniqueThreadId</u>	(<u>+</u>) <u>TimeStart</u>	(<u>+</u>) <u>TimeEnd</u>	(<u>+</u>) <u>Function</u>		(<u>+</u>) Function	Address (<u>+) ReturnAddre</u>	ess (+) <u>ReturnValue</u>				
<u>[0x0]</u> 0x0	0xfec	- 0x2	<u>4515:450</u>	<u>4517:8</u>	kernel32!Virtual	AllocStub	0x7742f9f0	- 0	x1110b4	0x150000				
0:000> !tt 4517:8														
Setting position: 4517:	B	- code 8000003 (first/s	econd chance not	available)										
Time Travel Position: 45/7/8														
eax=80159000 ebx=77427970 ecx=800808000 edx=080080000 esi=0011066f edi=0011066f eip=08011004 esp=000476ft; ebp=0804792 cipl=0 nv up eip lz rn ap enc														
cs=0023 ss=002b ds=002	cs=0023 ss=002b ds=002b ds=002b fs=0053 gs=002b ef1=00000246 0011004 dfh74f14 movrx ecx.word ntr [ed1:14h] ds:002b:00110631=0000													
001110b4 0fb74f14 0:000> dx -g @\$memory(0:	movzx ecx,wo x1110b4, 0x1110b	rd ptr [edi+14h] ds:00 4+1, "w"); \$\$ query the	2b:00110683=00e0 TimeStart of dec	oding current :	instruction									
(<u>+</u>) EventT	ype (<u>+</u>) <u>Thread</u>	Id (<u>+) UniqueThreadId</u>	(<u>+) TimeStart</u>	(<u>+) TimeEnd</u>	(<u>+</u>) <u>AccessType</u>	(<u>+) IP</u>	(+) Address	(<u>+</u>) <u>Size</u>	(<u>+) Value</u>	(<u>+) OverwrittenValue</u>				
<u>[0x0]</u> 0x1	0xfec	- 0x2	41E1:213D	41E1:213D	Write	0x11007d	0x1110b4	0x1	0xf	0x5				
0:000> !tt 41E1:213D														
Setting position: 41E1:	213D													
(et0.tec): Break instru Time Travel Position: 4	CTION EXCEPTION	- code 80000003 (first/s	econd chance not	avallable)										
eax=0011060a ebx=00014a	2 ecx=00000a48	edx=00000146 esi=0000000	7 edi=001110b4											
cs=0023 ss=002b ds=00	14 ebp=000d+924 2b es=002b fs=	10p1=0 nV up e1 0053 gs=002b	efl=00000207											
0011007d 3007	xor byte p	tr [edi],al ds:	002b:001110b4=05											
# ChildEBP RetAddr	ecuri audress c	o find the call												
WARNING: Frame IP not in 00 000df924 00110454	n any known modu Øx11007d	le. Following frames may	be wrong.											
01 000df944 7779d5b9	0x110454													
02 000df99c 77784e4b 03 000dfd04 77776391	ntdll!RtlDispa ntdll!KiUserAp	tchAPC+0x615a9 cDispatcher+0x4b												
04 000dfd10 00000000	ntdll!LdrIniti	alizeThunk+0x11												
00110434 81f0205e95d4	xor eax,0D	4955E20h												
0011043a 81e85a0ecf92 00110440 81c0120d35ab	sub eax,92	CF0E5Ah B350D12h												
00110446 e904fcffff	jmp 001100	4f												
0011044b 59 0011044c 53	pop ecx push ebx													
0011044d 8bce	mov ecx,es	i												
0:000> g- 0011044f; \$\$;	go back to the c	all and check params{ebx	, ecx}											
Time Travel Position: 4	1C8:16C5													
00110655 95 fb a3 d2 e	F 76 42 d9-33 44	48 83 be 22 13 37	.vB.3DH".7											
00110665 c2 4a 01 00 3 00110675 6e 7d 7b 48 f	2 08 00 00-34 6f 3 a9 47 6c-cb 32	c6 32 23 ab 34 11 .J db 0b 1f a7 94 0c n}{H	240.2#.4. Gl.2											
00110685 65 58 75 bb ft	8 78 43 ae-c7 2e	d7 8f 25 ad 7a 12 eXu.	.xC%.z.											
00110695 63 75 C7 ae T. 001106a5 69 6b 80 ac ee	2 57 49 7e-ca 34 e 53 45 68-c9 30	df 09 27 af 72 0a ik	.W1~.4!.V. .SEh.0'.r.											
001106b5 65 77 7a b8 f4	4 59 4b 70-c5 2c	d5 95 22 ab 78 14 ewz.	.YKp.,".x.											
0:000> db ecx 110	5 55 45 66-86 55													
0011066d 34 6f c6 32 2 0:000> p; \$\$ step over 1	3 ab 34 11-6e 7d the call and che	7b 48 f3 a9 47 6c 4o.2 ck the return value	#.4.n}{HGl											
Time Travel Position: 44	4F7:21A1													
eax=0011066d ebx=0011065	55 ecx=00014ac2	edx=000021be esi=0011066	d edi=00000832											
eip=00110454 esp=000df93	2c ebp=000df944	iopl=0 nv up ei	pl zr na pe nc											
00110454 83c408	add esp,8	5555 g5-0020	011-00000240											
0:000> db eax 0011066d f1 43 13 37 00	0 00 4c 01-05 00	03 fc 03 fc 00 00 .C.7												
0011067d 00 00 00 00 00	00 e0 00-02 21	0b 01 0e 23 00 c6	#											
0011069d 00 00 00 88 00	0 00 00 00-00 00 0 00 00 00-00 10	00 18 00 00 00 10												
001106ad 00 00 06 00 00	00 00 00-00 00	06 00 00 00 00 00								Δ.				
001106cd 40 01 00 00 10	00 00 10-00 00	00 00 10 00 00 10 @								Go				
0.000														

Figure 17. Backtrace from the VirtualAllocStub

The shellcode is slightly simpler than the previous one, that executes as an egg-hunter for searching DWORD value 0xd2a3fb95 and decrypting the following payload using a rolling xor. The xor key is hex value 95 fb a3 d2 ef 76 42 d9 33 44 48 83 be 22 13 37. The final payload is a reflective loader, that the DOS header stripped, and PE signature modified to hex value f1 43 13 37 00 00. The entry point of the reflective loader is transferred from address 0x110511.

Home View	Breakpoints	Time Travel	Model	Scripting	Source	Memory		Command
lisassembly						• 🖈	х	Command X
Address: #\$scopeip		7	Follow current ins	truction				0:000> g- 00110511
NN1 10090 CC	107						_	Time Travel Position: <u>4477:2188</u>
00110090 cc	int	3						
00110e9f 55	push	ebp						$e_1p=00110511 e_5p=00001934 e_5p=00001944 10p1=0 10 up e1 ng nz ac pe cy e_5p=0023 e_5p=0020 d_5p=00001934 e_5p=00001944 10p1=0 10 up e1 ng nz ac pe cy e_5p=00001934 e_5p=00001944 10p1=0 10 up e1 ng nz ac pe cy e_5p=00001944 e_5p=0000194 e_5p=00001944 e_5p=000000000000000000000000000000000000$
00110ea0 8bec	mov	ebp, esp						A0110511 ff55fc call dword ntr [ahn-4] cc:002b:0000025/
00110ea2 83ec24	sub	esp. 24h						
00110ea5 53	push	ebx						Time Travel Position: 44F7-21RC
00110ea6 33db	xor	ebx, ebx						eax=0011066d ebx=00110655 ecx=00014ac2 edx=000021be esi=0011066d edi=00110e9f
00110ea8 c745e800000000	mov	dword ptr [ebp-18h], 0					ein=00110e9f esn=000df930 ebn=000df944 ion]=0 nv un ei ng nz ac ne cv
00110eaf 56	push	esi						$c_{1} = 0023$ $s_{2} = 002h$ $d_{2} = 002h$ $e_{2} = 002h$ $s_{3} = 002h$ $e_{1} = 00000000000000000000000000000000000$
00110eb0 57	push	edi						00110e9f 55 push ebp
00110eb1 c745e400000000	mov	dword ptr [ebp-1Ch], 0					
00110eb8 895df0	mov	dword ptr [ebp-10h], eb	ĸ				
00110ebb 895de0	mov	dword ptr [ebp-20h], eb:	ĸ				
00110ebe e8ccffffff	call	00110E8F						
00110ec3 8bf0	mov	esi, eax						
00110ec5 b8f1430000	mov	eax, 43F1h						
00110eca 0f1f440000	nop	dword ptr [eax+eax]					
00110ecf 663906	cmp	word ptr [e	si], ax					
00110ed2 750f	jne	00110EE3						
00110ed4 817e0213370000	cmp	dword ptr [esi+2], 3713	h				
00110edb 8d7e02	lea	edi, [esi+2]					
∂0110ede 897df8	mov	dword ptr [ebp-8], edi					
20110ee1 7403	je	00110EE6						
00110ee3 4e	dec	esi						
00110ee4 ebe9	jmp	00110ECF						

Figure 18. Reflective loader entry point

The reflective loader checks the signature of header to be hex value f1 43 and 13 37 00 00, and continues to allocate RWX memory based on the following PE structure, right after copying section data and before resolving import tables, a hidden memcpy that copies a chunk data of size 0x500 into the virtual offset 0x3fc of the newly allocated RWX memory at 0x150000.

Figure 19. memcpy data into virtual offset 0x3fc

The offset 0x3fc and 0xbaadbeef ring a bell that previously staged shellcode of RC6 decryption. The flag is embraced with the hex value of be ad ba.

Google Cloud

FLARE

Command X
0:000> pc
Time Travel Position: 4549:103F
eax=00163b2c ebx=00000000 ecx=00000000 edx=00150000 esi=0003f7d1 edi=0016392c
eip=00111139 esp=000df8f8 ebp=000df92c iopl=0 nv up ei pl nz na po nc
cs=0023 ss=002b ds=002b es=002b fs=0053 gs=002b ef1=00000202
04111130 ff5568 call dword atr [aba_18b] cs:002b:000df014-(KENEL321) callibrary(AStub (77431270))
0015040C da D1 4a C0 db 5T 3a eD-bT 5T 9T dC Td 25 37 3dJ:
0015041c 5b 43 11 88 +b 50 68 ct-ee cc 98 b3 /t 1b aa 19 [CPh
0015042c d0 61 af 8e 82 e6 c5 b1-f1 29 8f 5c 77 de 6e 5f .a).\w.n_
0015043c f1 25 2b 70 2b a9 18 ee-0c 95 7f 5a ac d8 a1 26
0015044c 96 50 08 d3 b6 ce 0e e1-52 73 92 5a fa d3 60 27 .PRs.Z`'
0015045c f6 f4 af 60 8f 63 c9 62-27 8d 6f 0e 05 98 92 69`.c.b'.oi
0015046c f9 95 97 b0 8c 15 ab 3e-84 8e fe 51 6b 29 28 53>Qk)(S
0015047c 32 79 d8 47 8a f2 23 ab-3a 14 d4 b1 d1 15 a4 18 2v.G#.:
0015048c 8f 1d d6 e0 53 90 fd 69-f1 86 2c e7 44 1b dd ebSiD
0015049c 4d bc 31 65 ff ea 77 65-a8 f8 30 be 77 6a d9 c3 M.1e., we., 0.wi.,
001504ac 8d 1c f5 7d hd 5h f2 91-1a hd d0 0a 9e 06 53 8e } [S
0015044C CD 98 5C dT 04 09 68 33-41 T3 20 24 00 10 45 69
001504ec de 88 4f d2 3b a0 b4 c6-1d 03 5a ca fb 1c // 18;
001504+c 31 26 7e 96 36 2e 80 9b-04 e7 57 c6 e+ 83 7b 23 1&~.6\\#
0015050c 06 ac 9b 7f df 22 1d 96-58 98 22 11 44 43 42 55"X.".DCBU
0015051c 52 62 a4 b6 ef 97 75 0e-6c 44 3e 29 d0 fb 6e 19 Rbu.lD>)n.
0015052c 4b 02 ef a5 ca a0 14 42-60 5a e3 fd 91 cb a8 fc KB`Z
0015053c d6 98 ee 28 9c 92 5e 35-84 12 e0 5f ae de 4f a6(^50.
0015054c 01 f2 2f 14 4e 22 6e 17-7b ae c9 58 42 7a 0c 0e/.N"n.{XBz
0015055c 85 5c ca 80 7d 1f cc bd-cd a8 14 8a f0 e1 7c e5 .\}
0015056c 79 54 ae 33 31 d7 18 c3-58 28 51 e5 8a 7b ba fc vT.31X(0
0015057c b6 fb 79 6d 5f 0b 99 d3-27 be ff f8 61 2a 29 d5 vma*).
$0.0150558c$ ch 7h $0e$ 97 78 3d 36 74-d7 ac 6e 4d 3e d8 18 ef $\{x=6t n M\}$
$\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 $
0015034 / $02063 34$ / $51a$ / $31a$ / $31a$ - $34a$ 44 41 66 60 / 1304 / 1006
001505CC CD 06 TE T3 38 57 2a T2-67 e6 D0 34 04 1C D0 0e8W*.g4
001505dc cb ea d3 6b d4 bt c6 td-cd dt d8 81 2e e8 85 1bk
001505ec 38 ab 0e 14 eb e0 d9 88-84 17 06 e4 d2 11 76 d0 8v.
001505fc 41 d9 ab 1f d7 c2 5b d2-f4 56 af a2 48 ea a1 54 A[VHT
0015060c 40 0d a8 c1 1d 85 33 91-02 9f 24 02 5b 0c 3f a3 @3\$.[.?.
0015061c b5 e6 41 6c cd ca b4 e2-56 f3 79 ba a2 ca 92 95AlV.y
0015062c 4d 61 45 03 f8 61 59 6d-96 43 d8 d6 30 df d0 aa MaEaYm.C0
0015063c 8b 53 4c d5 e7 c6 38 6d-f1 e0 96 90 17 b7 48 c6 .SL8mH.
0015064c dc 58 da 91 90 4a 72 fd-71 cc 21 1d d9 f1 f6 83 .XJr.g.!
0015065c 67 50 b2 61 78 3d 01 42-f9 6f d8 d0 82 c8 1a 48 gP.ax=.B.oH
0015066c 4e 01 53 b3 41 0e 0f c2-88 3e 0b 27 6c 34 59 bd N S A > '14V
UNITARE AN
001506cc 00 00 00 00 00 00 00 00 00 00 00 00 0
001506dc 00 00 00 00 00 00 00 00 00 00 00 00 00
001506ec 00 00 00 00 00 00 00 00 00 00 00 00 00
0:000> .writemem c:\Users\flare\Desktop\flag.bin 1503fc+4 1280; \$\$ save the raw flag
Writing 280 bytes.

Figure 20. Save encrypted flag into local file

Once the reflective loader finished loading the PE into RWX memory, fixed the import tables and resolved the relocation RVAs, the execution switches to the payload's entry point. The code is straightforward that connects to the pipe \\.\pipe\whereami created by 2nd stage shellcode, retrieves the sample full path and creates the message box. If the computer user name is flare, and the sample full path contains C:\Users\Public\, it continues to check if the file offset 0x3fc contains DWORD value 0xbaadbeef, the message box with hint information would reveal.

Let's verify by live debugging the where_am_i.exe, and manipulating the parameters for the RC6 decrypt routine.

0:000> bp kernel32!VirtualAllocStub; \$\$ break at VirtualAlloc																
0:000> g																
ModLoad: 76b0000 76b25000 C:\Windows\SysWOW64\IMM32.DLL																
DFEERPUILE 0 HIL Dax=00484338 abx=02401348 pry=00006r44 pdy=00400000 pri=00400000 pdi=0040165																
eip=7742f9f0 esp=0019fdec ebp=0019fe20 iopl=0 nv up ei pl zr na pe nc																
cs=0023 ss=002b ds=002b es=002b fs=0053 gs=002b ef1=00000246																
KERNEL32!VirtualAllocStub:																
7742f9f0 8	3bf1	F				mo	/	edi	,edi	ι.						
0:000> gu; \$\$ get the alloc'd shellcode addr 0:000> r																
eax=00610000 ebx=024e1a48 ecx=f8180000 edx=00610000 esi=00000000 edi=024e1b65																
eip=00448a	ae3	est	0=00	2191	Fe00) et	20=00	019fe	20 i	lop	L=0	5100		r	00 IV U	p ei pl zr na pe nc
cs=0023 s	ss=6	002	0 0	ds=0	902ł) (es=0	002b +	fs=0	053	3	gs=(902t	,		efl=00000246
where_am_i+0x48ae3:																
00448ae3 8945f0 mov dword ptr [ebp-10h],eax ss:002b:0019fe10=0044d10c																
0:000> bd 0																
0:000> ba e1 @eax+5cd; \$\$ set bp at RC6 decrypt routine 0:000> g																
Breakpoint	: 1	hit	t													
eax=000151	L60	eb	x=02	24e1	La48	3 ec	cx=6	225000	90 e	edx=	=006	537e	28	esi	=00	000000 edi=024e1b65
eip=006105	ōcd	esp	o=00	ð19†	Fdeo	: eł	op=@	019fd	fc i	op]	L=0			r	nv u	p ei pl nz ac po cy
cs=0023 s	ss=6	902ł	0 0	ds=0	902ł	0 6	es=@	902b +	fs=0	053	3 (gs=(902ł)		efl=00000213
006105cd e	28f3	3230	3000	ð tæ		ca]	11	0063	1290	:5						-
0:000> dd	est 023	2500	+; ; 200	\$\$ 000	2nec 3151	ск р 160	oara 004	ams, er 537628	006	7pte	28	Jaca		ιт,	51Z	e
0:000> .dv	/al]	loc	0x3	300	: \$9	s al	1100	new r	nemo	prv	for	r se	etti	ing	raw	flag
Allocated	100	90 l	byte	es s	star	rtir	nga	at 006	2000	90		-				
0:000> .re	eadr	nem	c:)	\Use	ers۱	fla	are	Deskto	p/f	lag	g.bi	in (0062	2000	0 1	?280
Reading 28	30 t	byte	es.													
0:000> ed	esp	00	9620 200	3000	3 28	30;	\$\$	update	e pa	arar	ns (on 1	the	sta	ick;	dd esp 14
0019Tuec	000 ¢¢	5208 c+4	999 20 (100	2002 > +k	280 10 F	200	decry/	996 1 t	0376	228 tina	· ·				
0:000> db	006	5200	200	128	30:	\$\$	ma	/ the t	flag	z be	e wi	ith	νοι	ı ::		
00620000	57	68	65	72	65	20	61	6d-20	49	3f	20	49	27	6d	20	Where am I? I'm
00620010	6c	6f	73	74	20	69	6e	20-63	79	62	65	72	2c	0d	0a	lost in cyber,
00620020	41	20	70	6c	61	63	65	20-77	68	65	72	65	20	62	69	A place where bi
00620030	74	73	20	61	6e	64	20	62-79	74	65	73	20	63	6f	6e	ts and bytes con
00620040	73 6d	70	20	/2 6f	60	20	61	20-62	6f	20	74	60	65	70	20	spire,To take
00620050	66	61	72	2c	0d	0a	46	72-6f	6d	20	77	68	65	72	65	farFrom where
00620070	20	49	20	61	6d	20	74	6f-20	77	68	65	72	65	20	79	I am to where y
00620080	6f	75	20	61	72	65	2e	0d-0a	0d	0a	54	68	65	20	77	ou areThe w
00620090	6f	72	6c	64	20	69	73	20-76	61	73	74	20	61	6e	64	orld is vast and
006200a0	20 0d	66	/5	6C	6C	20	61	66-20	65	6† 20	6e	64 20	65	72 6d	2C	tull of wonder,
00020000 006200c0	20	6a	42	73	74	20	60	6f-73	74	20	49 69	20 6e	20	74	2C	iust lost in th
006200d0	75	6e	64	65	72	2c	Ød	0a-4f	66	20	64	61	74	61	20	under,Of data
006200e0	73	74	72	65	61	6d	73	20-61	6e	64	20	65	6e	64	6c	streams and endl
006200 f 0	65	73	73	20	63	6f	64	65-2c	0d	0a	41	20	70	6c	61	ess code,A pla
00620100	63	65	20	77	68	65	72	65-20	6f	6e	65	73	20	61	6e	ce where ones an
00620110	64	20	/a	65 0d	/2	61	65 75	73-20	65 72	78	/0	60	61	64 20	65 40	a zeroes explode
00620120	20	73	65	61	72	63	68	20-20	66	6f	72	20	77	68	61	search, for wha
00620140	74	20	49	20	73	65	65	6b-2c	Ød	0a	41	20	70	6c	61	t I seek,A pla
00620150	63	65	20	77	68	65	72	65-20	61	6e	73	77	65	72	73	ce where answers
00620160	20	63	6f	6d	65	20	74	6f-20	6d	65	65	74	2c	0d	0a	come to meet,
00620170	4d	79	20	71	75	65	73	74-69	6f	6e	73	20	64	65	65	My questions dee
00620180	/0 6f	20	61	68	64 74	20	66 0d	75-6C 02-41	6C 20	20	6T	60	62	74	20	p and full of th
00620130	77	68	65	72	65	20	6h	6e-6f	77	60	65	64	67	65	20	where knowledge
006201b0	63	61	6e	20	62	65	20	73-6f	75	67	68	74	2e	Ød	0a	can be sought
006201c0	0d	0a	41	6e	64	20	74	68-65	6e	20	49	20	73	65	65	And then I see
006201d0	2c	20	61	20	6c	69	67	68-74	20	73	6f	20	62	72	69	, a light so bri
006201e0	67	68	74	2c	0d	0a	49	74-20	73	68	69	6e	65	73	20	ght,It shines
006201+0	73	61	20	74	72	75	65	20-20	69	74	27	73	20	/1 4f	75 60	so true, it's qu
00620200	20	6d	79	20	46	40	41	52-45	20	20	74	68	65	72	65	my FLARE, there
00620220	20	79	6f	75	20	61	72	65-2c	20	57	68	65	52	33	5f	you are, WheR3_
00620230	34	6d	5f	49	5f	66	72	30-6d	5f	30	54	46	40	66	6c	4m_I_fr0m_0TF@fl
00620240	61	72	65	2d	6f	6e	2e	63-6f	6d	Ød	0a	4d	79	20	67	are-on.comMy g
00620250	75	69	64	69	6e	67	20	73-74	61	72	2c	20	6d	79	20	uiding star, my
00620260	13	68	69	60	69	66	6/	20-73	74	61	12	20	00	60 00	00	Shining Star
00020270	00	00	00	00	00	00	00	00-00	00	00	00	00	00	00	00	
0.000>																

Figure 21. Live debugging solution

Congratulations! The flag is WheR3_4m_I_fr0m_0TF@flare-on.com

References

- 1. <u>Time Travel Debugging TTD.exe command line utility</u>
- 2. <u>Time Travel Debugging JavaScript Automation</u>
- 3. Using LINQ With the debugger objects
- 4. Axel Souchet's codecov plugin

Appendix

```
// WinDbg dbgInit.js
// @binjo, 2023-05-04
"use strict";
delete Object.prototype.toString;
const log = host.diagnostics.debugLog;
const logln = p => host.diagnostics.debugLog(p + "\n");
function invokeScript() {
    return logln("====== WinDbg init done... ======");
}
function ReadPtr(Addr) {
    let Value = null;
   let is64 = host.namespace.Debugger.State.PseudoRegisters.General.ptrsize == 8;
   try {
        if (is64) {
            Value = host.memory.readMemoryValues(
                Addr, 1, 8
            )[0];
        } else {
            Value = host.memory.readMemoryValues(
                Addr, 1, 4
            )[0];
        }
    } catch(e) {
    }
    return Value;
}
function GetSym(Addr) {
    if(Addr == undefined) {
        logln("!getsym <addr>");
        return;
    }
```

```
let dis = host.namespace.Debugger.Utility.Code.CreateDisassembler();
    let ins = dis.DisassembleInstructions(Addr);
    let addr = ins.First().Operands.Last().ImmediateValue; // get rid of calling address, no need
to care about x64/x86/far/near
   let ptr = ReadPtr(addr);
    let temp = host.namespace.Debugger.Utility.Control.ExecuteCommand(`.printf"%y",
${ptr.toString(16)}`)[0];
    return temp;
}
class __CallItem {
   constructor(ts, rva, addr, sym) {
        this.TimeStart = ts;
        this.RVA = rva;
        this.Address = addr;
        this.Called = sym;
    }
   toString() {
        return `${this.Called}`;
    }
}
class __CallTrace {
   constructor(baseAddress, size) {
        this.__BaseAddress = baseAddress;
        this.__Size = size;
        this.__mod_cov = host.currentSession.TTD.Memory(baseAddress, baseAddress + size, "ec");
        this.__mod_calls = this.__mod_cov.Where(r => r.Size == 6 && (r.Value & 0xffff) == 0x15ff);
    }
    *[Symbol.iterator]() {
        let mod_calls = this.__mod_calls;
        for (var cal of mod_calls) {
            var sym = GetSym(cal.Address);
            yield new __CallItem(cal.TimeStart, cal.Address - this.__BaseAddress, cal.Address,
sym);
        }
    }
   toString() {
        return "TraceCalls";
    }
}
let calls = x => host.currentSession.TTD.Calls(x);
let memory = (x, y, z) => host.currentSession.TTD.Memory(x, y, z);
let Traces = {
```

```
__mytrace : {},
   get TraceCalls() {
        if (!(Traces.__mytrace && Traces.__mytrace[this.Name]))
            Traces.__mytrace[this.Name] = new __CallTrace(this.BaseAddress, this.Size);
        return Traces.__mytrace[this.Name];
    },
    // dx @$curprocess.Modules[0].TraceInRange(..., ...)
   TraceInRange : function (baseAddr, size) {
        if (!(Traces.__mytrace && Traces.__mytrace[baseAddr.toString()]))
            Traces.__mytrace[baseAddr.toString()] = new __CallTrace(baseAddr, size);
        return Traces.__mytrace[baseAddr.toString()];
    },
   TraceClear : function () {
        if (Traces.__mytrace) {
            delete Traces.__mytrace;
        }
        logln("Trace cache cleared...");
    }
}
function initializeScript() {
    return [
        new host.apiVersionSupport(1, 3),
        new host.functionAlias(calls, "calls"),
        new host.functionAlias(memory, "memory"),
        new host.functionAlias(GetSym, "getsym"),
        new host.namedModelParent(Traces, "Debugger.Models.Module")
    ];
}
```

Table 3. WinDbg helper script