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 Asynchronous Procedure Call (APC). 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 FLARE VM 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,

- Detect It Easy, for identifying PE architecture, packer, entropy, sections etc.
- CFF Explorer, for viewing PE structure, extracting resource
- FLOSS, for extracting strings
- CAPA, 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                     | anti-analysis                                        |
| encrypt data using RC4 PRGA                          | data-manipulation/encryption/rc4                     |
| contain a resource (.rsrc) section                   | executable/pe/section/rsrc                           |
| extract resource via kernel32 functions (6 matches)  | executable/resource                                  |
| delete file                                          | host-interaction/file-system/delete                  |
| get file attributes                                  | host-interaction/file-system/meta                    |
| move file                                            | host-interaction/file-system/move                    |
| read .ini file                                       | host-interaction/file-system/read                    |
| set application hook                                 | host-interaction/gui                                 |
| get graphical window text (2 matches)                | host-interaction/gui/window/get-text                 |
| allocate thread local storage                        | host-interaction/process                             |
| allocate RWX memory                                  | host-interaction/process/inject                      |
| terminate process                                    | host-interaction/process/terminate                   |
| query or enumerate registry value (2 matches)        | host-interaction/registry                            |
| set registry value                                   | host-interaction/registry/create                     |
| get kernel32 base address                            | linking/runtime-linking                              |
| link many functions at runtime                       | linking/runtime-linking                              |
| resolve function by parsing PE exports (2 matches)   | load-code/pe                                         |
+------------------------------------------------------+------------------------------------------------------+
```

Figure 1. CAPA Summary

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

```
encrypt data using RC4 PRGA
namespace  data-manipulation/encryption/rc4
author     moritz.raabe@mandiant.com
scope      function
att&ck     Defense Evasion::Obfuscated Files or Information [T1027]
mbc        Cryptography::Encrypt Data::RC4 [C0027.009], Cryptography::Generate Pseudo-random Sequence::RC4 PRGA [C0021.004]
function @ 0x448733
  and:
    match: contain loop @ 0x448733
      or:
        characteristic: loop @ 0x448733
    count(characteristic(nzxor)): 1 @ 0x4489B8
    count(characteristic(calls from)): 4 or fewer @ 0x460218, 0x46021C, 0x478370
    count(basic block): between 4 and 50 @ 0x448733, 0x44876D, 0x448773, 0x448778, and 29 more...
    or:
      count(mnemonic(movzx)): 4 or more @ 0x4488C0, 0x4488CC, 0x448948, 0x448986, and 2 more...
    optional:
      or:
        number: 0x100 @ 0x448739, 0x448750, 0x44878A, 0x4487DA, and 8 more...
```

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).

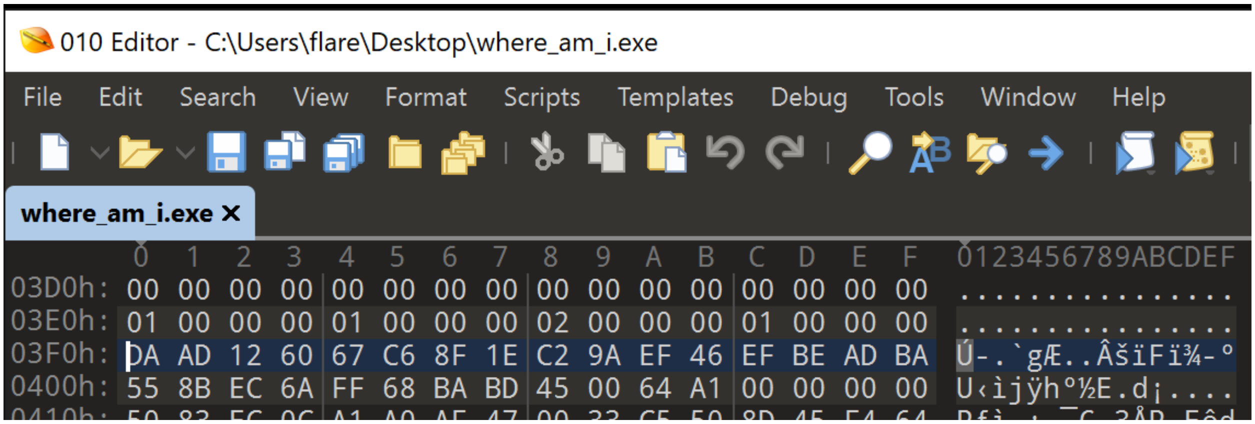

Figure 3. RC4 key at file offset 0x3f0

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.

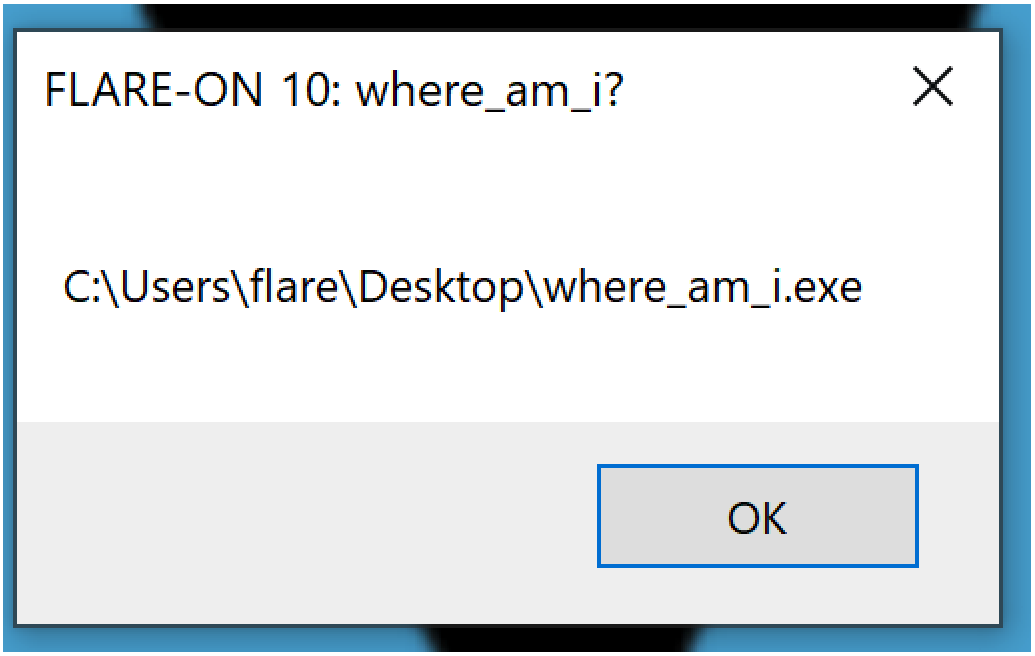

Figure 5. Dialog of execution

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

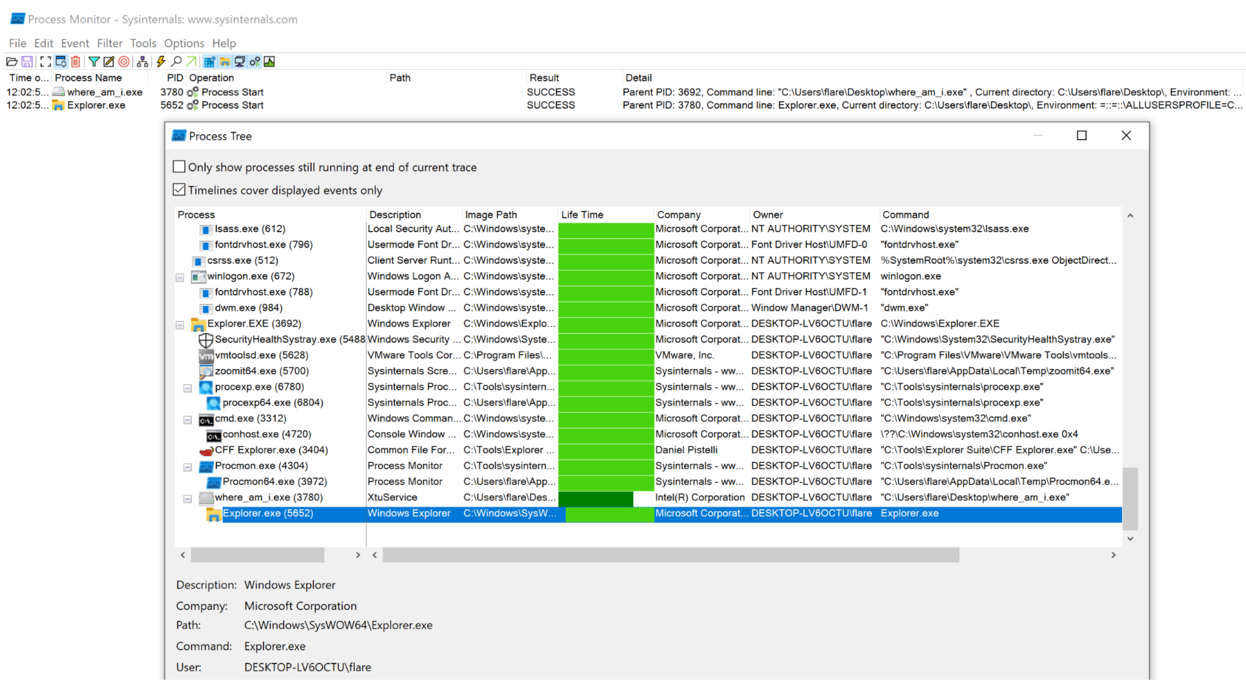

Figure 6. Process Monitor - Process Tree

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 `~*k` when attached.

```
0:001> ~*k
   0  Id: 1614.1450 Suspend: 1 Teb: 0022e000 Unfrozen
 # ChildEBP RetAddr
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 Suspend: 1 Teb: 00249000 Unfrozen
 # ChildEBP RetAddr
00 0048ff40 777bdce9     ntdll!DbgBreakPoint
01 0048ff70 774300c9     ntdll!DbgUiRemoteBreakin+0x39
02 0048ff80 77777b1e     KERNEL32!BaseThreadInitThunk+0x19
03 0048ffdc 77777aee     ntdll!__RtlUserThreadStart+0x2f
04 0048ffec 00000000     ntdll!_RtlUserThreadStart+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> ub 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
0:003> db 552C94h
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 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  e...C:\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 TTD. 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.

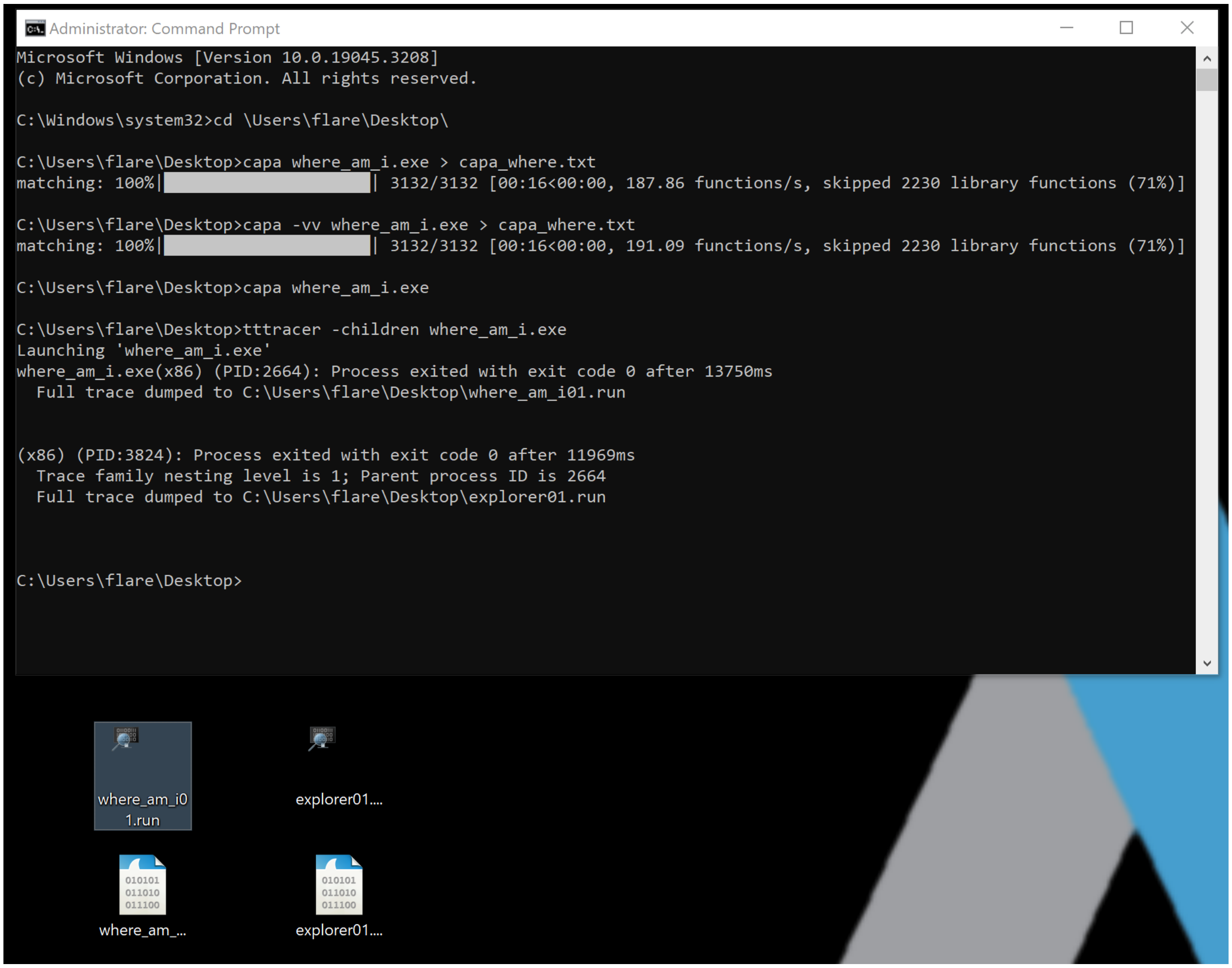

Figure 7. Output of tttracer.exe

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.

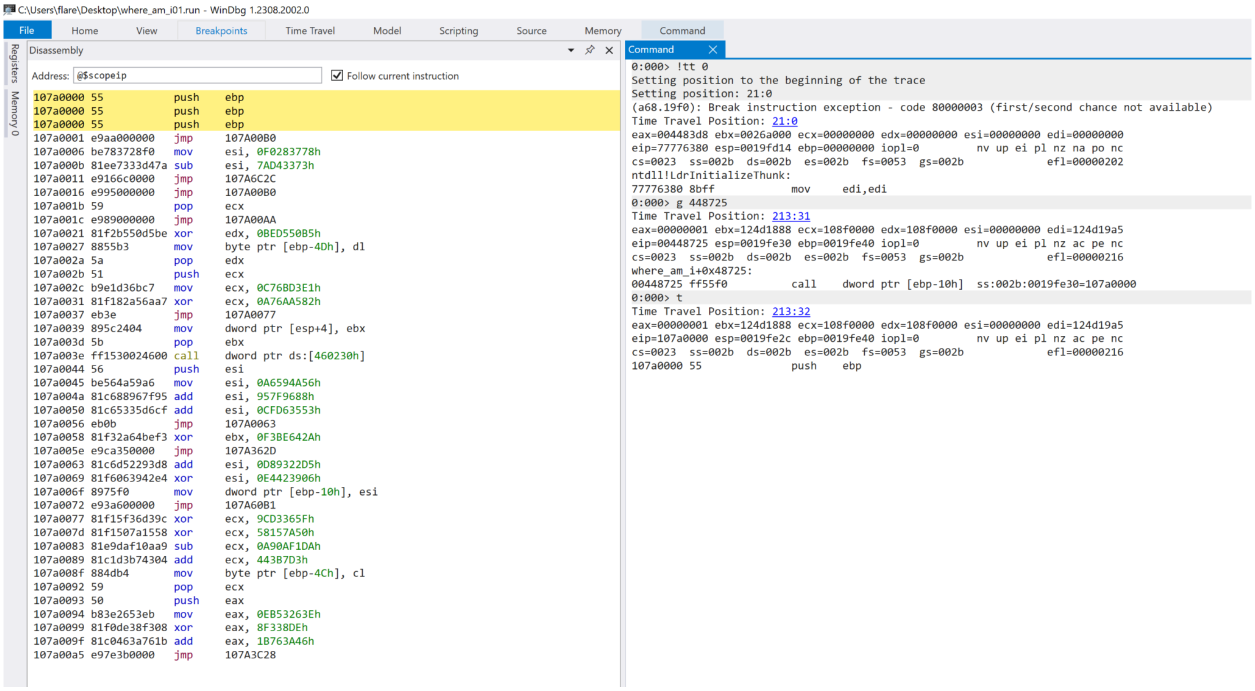

Figure 8. First stage shellcode entry point

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.

Figure 9. Graph view of the shellcode

Tracing into/over commands like pa would be helpful in certain cases, the debugger data model and ability of using LINQ with the debugger objects 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.TTD.Calls("kernel32!VirtualAllocStub").Select(r => new {TimeStart = r.TimeStart, TimeEnd = r.TimeEnd, Function =
```

	(+) TimeStart	(+) TimeEnd	(+) Function	(+) FunctionAddress	(+) ReturnAddress	(+) ReturnValue
[0x0]	143:451	145:8	kernel32!VirtualAllocStub	0x7742f9f0	0x448ae3	0x107a0000
[0x1]	22B:999	22D:8	kernel32!VirtualAllocStub	0x7742f9f0	0x107a11b2	0x107b0000

```
0:000> !tt 143:451
Setting position: 143:451
(a68.19f0): Break instruction exception - code 80000003 (first/second chance not available)
Time Travel Position: 143:451
eax=00484a38 ebx=124d1888 ecx=00006c44 edx=00400000 esi=00000000 edi=124d19a5
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             efl=00000246
KERNEL32!VirtualAllocStub:
7742f9f0 8bff            mov     edi,edi
0:000> dd esp l8
0019fdec  00448ae3 00000000 00006c44 00003000
0019fdfc  00000040 00484a38 e6490001 00006c44
```

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 codecov plugin, 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)
```

	(+) TimeStart	RVA	Address	Called
[0x0] : KERNEL32!SetErrorModeStub (77430cb0)	213:56	0x674	0x107a0674	KERNEL32!SetErrorModeStub (77430cb0)
[0x1] : KERNEL32!GetSystemDefaultLangIDStub (77425120)	217:B	0x8fc	0x107a08fc	KERNEL32!GetSystemDefaultLangIDStub (77425120)
[0x2] : KERNEL32!GetAtomNameA (774633b0)	228:2E	0x5744	0x107a5744	KERNEL32!GetAtomNameA (774633b0)
[0x3] : KERNEL32!FindResourceA (77422eb0)	228:1E6	0x12e4	0x107a12e4	KERNEL32!FindResourceA (77422eb0)
[0x4] : KERNEL32!SizeofResourceStub (77430b20)	22B:5FA	0x3277	0x107a3277	KERNEL32!SizeofResourceStub (77430b20)
[0x5] : KERNEL32!LoadResourceStub (7742ee70)	22B:7A2	0x5d88	0x107a5d88	KERNEL32!LoadResourceStub (7742ee70)
[0x6] : KERNEL32!LockResourceStub (7742f970)	22B:94B	0x63cd	0x107a63cd	KERNEL32!LockResourceStub (7742f970)
[0x7] : KERNEL32!VirtualAllocStub (7742f9f0)	22B:998	0x11ac	0x107a11ac	KERNEL32!VirtualAllocStub (7742f9f0)
[0x8] : KERNEL32!GetProcessHeapStub (7742f9b0)	4E3:22BD	0x592a	0x107a592a	KERNEL32!GetProcessHeapStub (7742f9b0)
[0x9] : ntdll!RtlAllocateHeap (77755dc0)	4E3:22C4	0x594b	0x107a594b	ntdll!RtlAllocateHeap (77755dc0)
[0xa] : KERNEL32!GetProcessHeapStub (7742f9b0)	3F20:84F	0x6409	0x107a6409	KERNEL32!GetProcessHeapStub (7742f9b0)
[0xb] : KERNEL32!HeapFreeStub (7742e590)	3F20:855	0x6410	0x107a6410	KERNEL32!HeapFreeStub (7742e590)
[0xc] : KERNEL32!SleepStub (774315a0)	3F22:61	0x24cc	0x107a24cc	KERNEL32!SleepStub (774315a0)
[0xd] : KERNEL32!OpenMutexA (77422e60)	3F24:9A	0x5ca6	0x107a5ca6	KERNEL32!OpenMutexA (77422e60)
[0xe] : KERNEL32!CreateProcessAStub (77444110)	3F28:19B5	0x6a59	0x107a6a59	KERNEL32!CreateProcessAStub (77444110)
[0xf] : KERNEL32!VirtualAllocExStub (77446370)	411D:104	0x1a9d	0x107a1a9d	KERNEL32!VirtualAllocExStub (77446370)
[0x10] : KERNEL32!WriteProcessMemoryStub (774465c0)	411F:6B	0x6538	0x107a6538	KERNEL32!WriteProcessMemoryStub (774465c0)
[0x11] : KERNEL32!QueueUserAPCStub (77429790)	4126:44	0x4bc6	0x107a4bc6	KERNEL32!QueueUserAPCStub (77429790)
[0x12] : KERNEL32!ResumeThreadStub (77431f90)	4128:23	0x3601	0x107a3601	KERNEL32!ResumeThreadStub (77431f90)
[0x13] : KERNEL32!GetModuleFileNameAStub (774314d0)	412A:2C	0xeca	0x107a0eca	KERNEL32!GetModuleFileNameAStub (774314d0)
[0x14] : ADVAPI32!InitializeSecurityDescriptor (7607f3b0)	412E:96	0x2945	0x107a2945	ADVAPI32!InitializeSecurityDescriptor (7607f3b0)
[0x15] : ADVAPI32!SetSecurityDescriptorDacl (7607f110)	412E:E6	0x3792	0x107a3792	ADVAPI32!SetSecurityDescriptorDacl (7607f110)
[0x16] : KERNEL32!SleepStub (774315a0)	412E:13C	0x6509	0x107a6509	KERNEL32!SleepStub (774315a0)
[0x17] : KERNEL32!CreateFileA (774337d0)	47E4:8A	0x250c	0x107a250c	KERNEL32!CreateFileA (774337d0)
[0x18] : KERNEL32!WriteFile (77433c50)	47EB:90	0x5565	0x107a5565	KERNEL32!WriteFile (77433c50)
[0x19] : KERNEL32!CloseHandle (77433580)	47EE:1F	0x4532	0x107a4532	KERNEL32!CloseHandle (77433580)
[0x1a] : KERNEL32!ExitProcessImplementation (77435940)	47F2:23	0x2bb6	0x107a2bb6	KERNEL32!ExitProcessImplementation (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
0:000> !tt 411F:6B
Setting position: 411F:6B
(a68.19f0): Break instruction exception - code 80000003 (first/second chance not available)
Time Travel Position: 411F:6B
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             efl=00000297
107a6538 ff1580834700     call    dword ptr [where_am_i+0x78380 (00478380)] ds:002b:00478380={KERNEL32!WriteProcessMemoryStub (774465c0)}
0:000> dd esp l8
0019fb28  0000010c 00110000 107b0031 0001512f
0019fb38  00000000 77782bbc 00000000 536cd652
0:000> db 107b0031 l20
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 e9  ........0..._9..
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 00 01 31 00-00 00 2f 51 01 00 5c 5c  ......1.../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  $........0..._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`....
```

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
02 0019fe28 00448728     0x107a69e4
03 0019fe44 00452894     where_am_i+0x48728
04 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
08 0019ff80 77777b1e     KERNEL32!BaseThreadInitThunk+0x19
09 0019ffdc 77777aee     ntdll!__RtlUserThreadStart+0x2f
0a 0019ffec 00000000     ntdll!_RtlUserThreadStart+0x1b
0:000> ub 0x107a05d2
107a05b8 8d4cd016        lea     ecx,[eax+edx*8+16h]
107a05bc 53              push    ebx
107a05bd e9b60a0000      jmp     107a1078
107a05c2 81c639ff2be8    add     esi,0E82BFF39h
107a05c8 8b0c2e          mov     ecx,dword ptr [esi+ebp]
107a05cb 5e              pop     esi
107a05cc 51              push    ecx
107a05cd e8f3230000      call    107a29c5
0:000> g- 107a05cd
Time Travel Position: 4EF:26E
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
107a05cd e8f3230000      call    107a29c5
0:000> dd esp l4
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: 3F20:833
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             efl=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 5c  ......1.../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  $........0..._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.

```
Command
0:000> dx @$memory(0x108f3d00, 0x108f3d00+1, "w").Where(r => r.Value != 0).First().TimeStart.SeekTo()
(a68.19f0): Break instruction exception - code 80000003 (first/second chance not available)
Time Travel Position: 4E5:1C8
@$memory(0x108f3d00, 0x108f3d00+1, "w").Where(r => r.Value != 0).First().TimeStart.SeekTo()
0:000> r
eax=ef0001ba ebx=124d1888 ecx=108f3d00 edx=b7e15163 esi=00000000 edi=124d19a5
eip=107a386f esp=0019fdbc ebp=0019fde8 iopl=0         nv up ei ng nz na po nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000282
107a386f 8911            mov     dword ptr [ecx],edx  ds:002b:108f3d00=00000000
0:000> k
 # ChildEBP RetAddr
WARNING: Frame IP not in any known module. Following frames may be wrong.
00 0019fde8 107a5837     0x107a386f
01 0019fdfc 107a69e4     0x107a5837
02 0019fe28 00448728     0x107a69e4
03 0019fe44 00452894     where_am_i+0x48728
04 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
08 0019ff80 77777b1e     KERNEL32!BaseThreadInitThunk+0x19
09 0019ffdc 77777aee     ntdll!__RtlUserThreadStart+0x2f
0a 0019ffec 00000000     ntdll!_RtlUserThreadStart+0x1b
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- 107a5832
Time Travel Position: 4E5:BA
eax=108f3d00 ebx=124d1888 ecx=baadbeef edx=00000000 esi=00000000 edi=124d19a5
eip=107a5832 esp=0019fdf0 ebp=0019fdfc iopl=0         nv up ei pl nz na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000206
107a5832 e8a7f5ffff      call    107a4dde
0:000> dd esp l4
0019fdf0  baadbeef 108f3d00 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 c:\Users\flare\Desktop\trace-107a4dde.txt
Opened log file 'c:\Users\flare\Desktop\trace-107a4dde.txt'
0:000> ta 107a5837                                            // trace until the return address of the call
ta 107a5837
eax=108f3d00 ebx=124d1888 ecx=baadbeef edx=00000000 esi=00000000 edi=124d19a5
eip=107a4dde esp=0019fdec ebp=0019fdfc iopl=0         nv up ei pl nz na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000206
107a4dde 55              push    ebp
...
107a386b 8b1428          mov     edx,dword ptr [eax+ebp] ds:002b:0019fdc4=b7e15163
eax=fffffdc ebx=124d1888 ecx=108f3d00 edx=b7e15163 esi=00000000 edi=124d19a5
eip=107a386e esp=0019fdb8 ebp=0019fde8 iopl=0         nv up ei ng nz na po nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000282
107a386e 58              pop     eax
...
eax=b7e15163 ebx=124d1888 ecx=00000001 edx=108f3d00 esi=00000000 edi=124d19a5
eip=107a3340 esp=0019fdbc ebp=0019fde8 iopl=0         nv up ei ng nz na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000286
107a3340 0345f4          add     eax,dword ptr [ebp-0Ch] ss:002b:0019fddc=9e3779b9
...
eax=ef0001ba ebx=00000024 ecx=108f3d00 edx=b7e15163 esi=00000000 edi=124d19a5
eip=107a5384 esp=0019fdb8 ebp=0019fde8 iopl=0         nv up ei pl nz ac pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000216
107a5384 395de8          cmp     dword ptr [ebp-18h],ebx ss:002b:0019fdd0=00000001
eax=ef0001ba ebx=00000024 ecx=108f3d00 edx=b7e15163 esi=00000000 edi=124d19a5
eip=107a5387 esp=0019fdb8 ebp=0019fde8 iopl=0         nv up ei ng nz ac pe cy
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000297
107a5387 5b              pop     ebx
eax=ef0001ba ebx=124d1888 ecx=108f3d00 edx=b7e15163 esi=00000000 edi=124d19a5
eip=107a5388 esp=0019fdbc ebp=0019fde8 iopl=0         nv up ei ng nz ac pe cy
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000297
107a5388 0f83fbfeffff    jae     107a5289                                [br=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 rc6_initl code snippet 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.

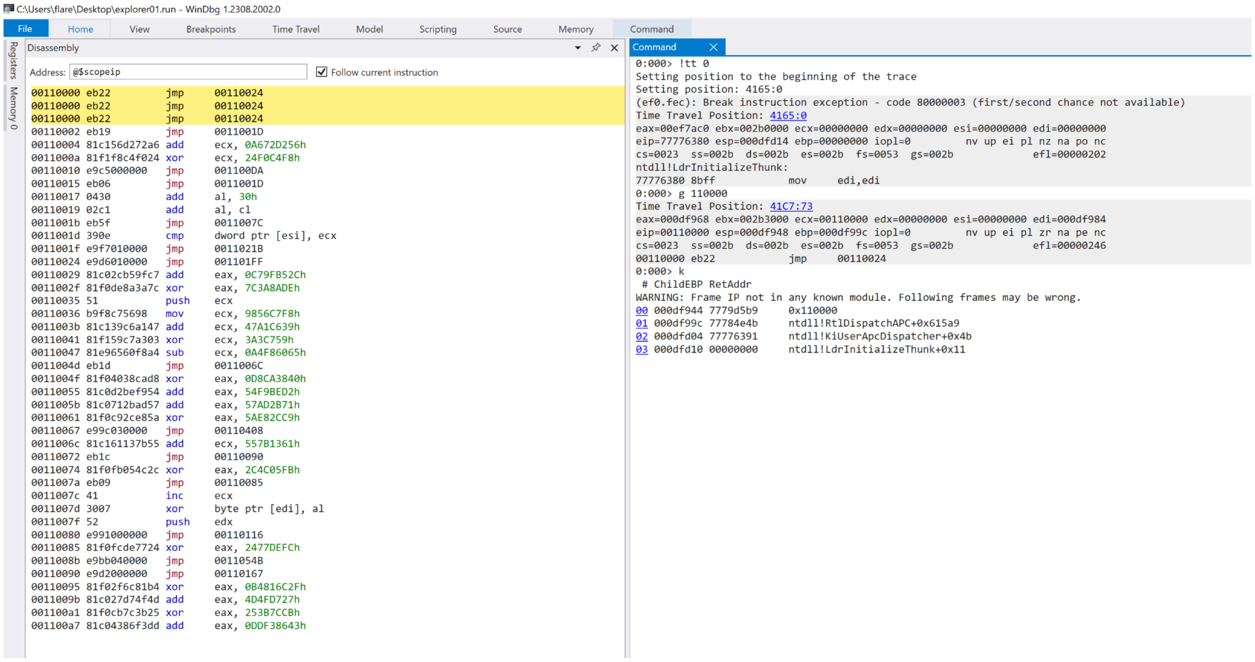

Figure 16. Shellcode entry point in Explorer.exe

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.

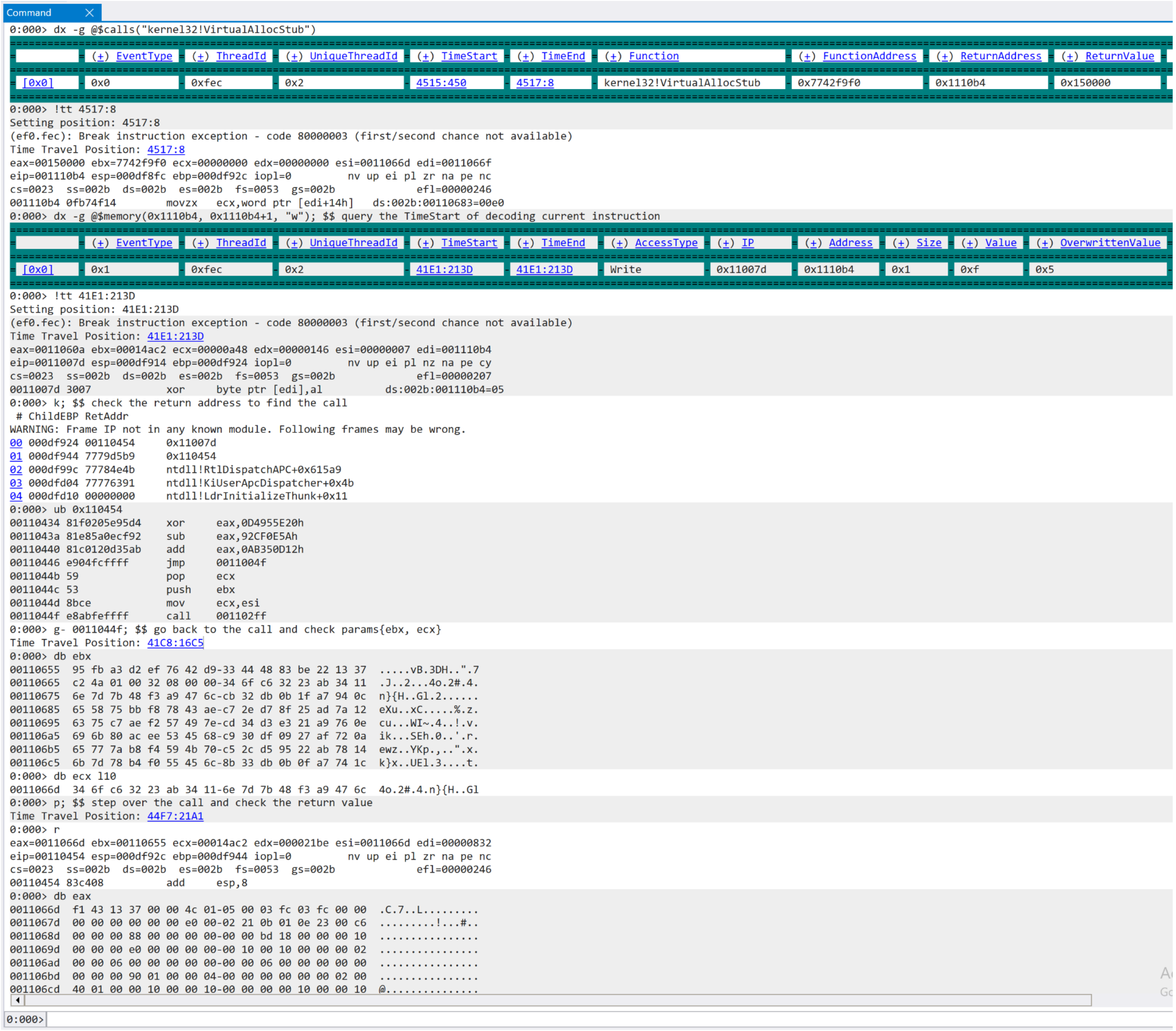

Figure 17. Backtrace from the VirtualAllocStub

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.

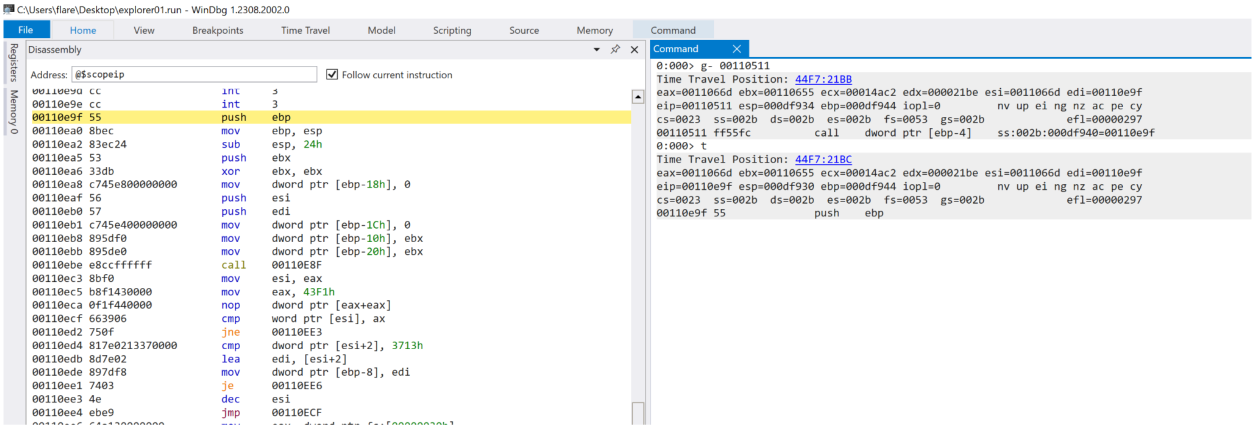

Figure 18. Reflective loader entry point

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.

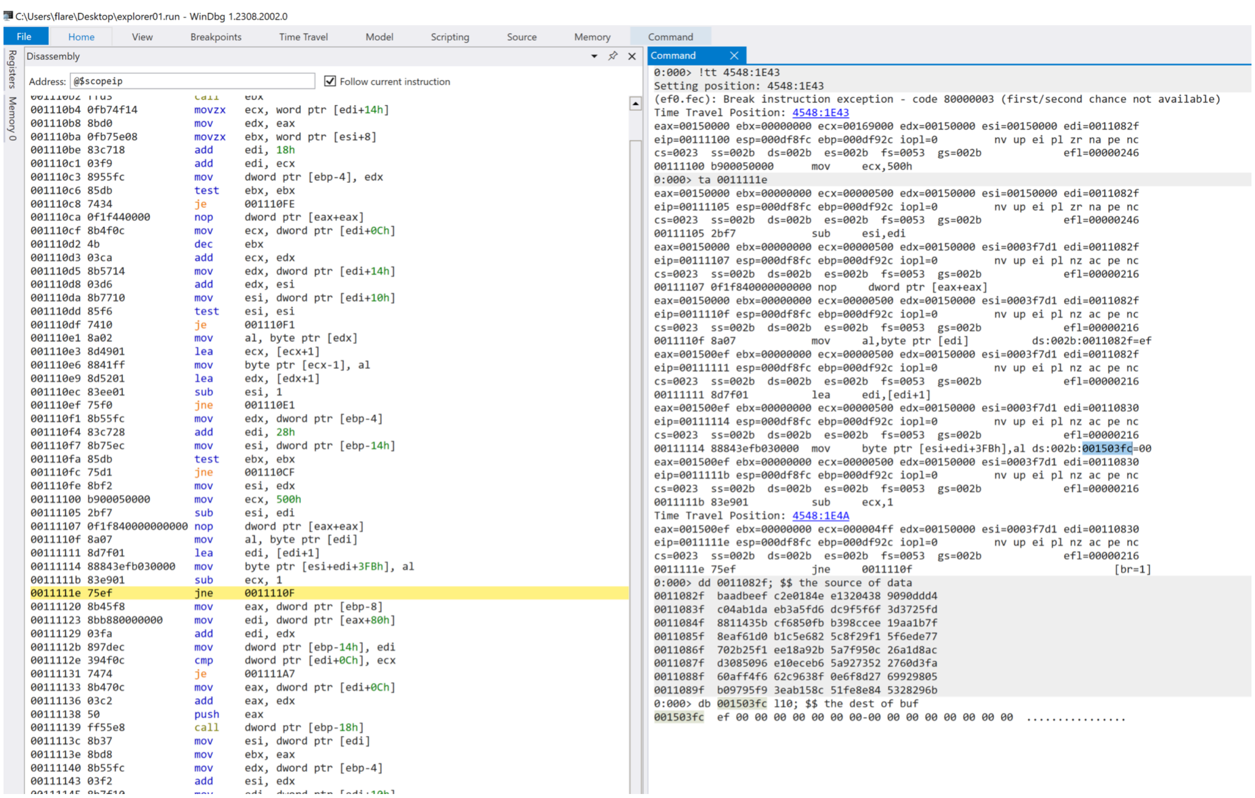

Figure 19. memcpy data into virtual offset 0x3fc

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

```
Command
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             efl=00000202
00111139 ff55e8          call    dword ptr [ebp-18h]  ss:002b:000df914={KERNEL32!LoadLibraryAStub (77431270)}
0:000> db 1503fc l300; $$ check the raw flag
001503fc  ef be ad ba 4e 18 e0 c2-38 04 32 e1 d4 dd 90 90  ....N...8.2.....
0015040c  da b1 4a c0 d6 5f 3a eb-6f 5f 9f dc fd 25 37 3d  ..J.._:.o_...%7=
0015041c  5b 43 11 88 fb 50 68 cf-ee cc 98 b3 7f 1b aa 19  [C...Ph.........
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  .%+p+......Z...&
0015044c  96 50 08 d3 b6 ce 0e e1-52 73 92 5a fa d3 60 27  .P......Rs.Z..`'
0015045c  f6 f4 af 60 8f 63 c9 62-27 8d 6f 0e 05 98 92 69  ...`.c.b'.o....i
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  2y.G..#.:.......
0015048c  8f 1d d6 e0 53 90 fd 69-f1 86 2c e7 44 1b dd eb  ....S..i..,.D...
0015049c  4d bc 31 65 ff ea 77 65-a8 f8 30 be 77 6a d9 c3  M.1e..we..0.wj..
001504ac  8d 1c f5 7d bd 5b f2 91-1a bd d0 0a 9e 06 53 8e  ...}.[........S.
001504bc  e0 5b e3 00 e5 82 54 d6-4e 3a fc 5f 17 27 54 df  .[....T.N:._.'T.
001504cc  7a 8a 23 9e 16 86 60 d8-cd 79 ba 50 f1 f6 4c d7  z.#...`..y.P..L.
001504dc  cb 98 5c df b4 d9 e8 3a-41 f3 2d 24 00 1d 45 e9  ..\....:A.-$..E.
001504ec  de 88 4f d2 3b a0 b4 c6-1d 03 5a ca fb 1c 77 18  ..O.;.....Z...w.
001504fc  31 26 7e 96 36 2e 80 9b-04 e7 57 c6 ef 83 7b 23  1&~.6.....W...{#
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  Rb....u.lD>)..n.
0015052c  4b 02 ef a5 ca a0 14 42-60 5a e3 fd 91 cb a8 fc  K......B`Z......
0015053c  d6 98 ee 28 9c 92 5e 35-84 12 e0 5f ae de 4f a6  ...(..^5..._..O.
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  yT.31...X(Q..{..
0015057c  b6 fb 79 6d 5f 0b 99 d3-27 be ff f8 61 2a 29 d5  ..ym_...'...a*).
0015058c  cb 7b 0e 97 78 3d 36 74-d7 ac 6e 4d 3e d8 18 ef  .{..x=6t..nM>...
0015059c  a8 40 3e 33 57 7f d4 4f-5b 7f d2 d9 3f 90 3f 2c  .@>3W..O[...?.?,
001505ac  70 26 e5 54 73 1a 79 1a-34 a4 d1 0e 03 d7 f3 84  p&.Ts.y.4.......
001505bc  14 1e f9 4f fc 57 07 c8-b0 8f c1 4e 51 3d 68 36  ...O.W.....NQ=h6
001505cc  cb 06 fe f3 38 57 2a f2-67 e6 b0 34 d4 1c bd de  ....8W*.g..4....
001505dc  cb ea d3 6b d4 bf c6 fd-cd df d8 81 2e e8 85 1b  ...k............
001505ec  38 ab 0e 14 eb e0 d9 88-84 17 06 e4 d2 11 76 d0  8.............v.
001505fc  41 d9 ab 1f d7 c2 5b d2-f4 56 af a2 48 ea a1 54  A.....[..V..H..T
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 95  ..Al....V.y.....
0015062c  4d 61 45 03 f8 61 59 6d-96 43 d8 d6 30 df d0 aa  MaE..aYm.C..0...
0015063c  8b 53 4c d5 e7 c6 38 6d-f1 e0 96 90 17 b7 48 c6  .SL...8m......H.
0015064c  dc 58 da 91 90 4a 72 fd-71 cc 21 1d d9 f1 f6 83  .X...Jr.q.!.....
0015065c  67 50 b2 61 78 3d 01 42-f9 6f d8 d0 82 c8 1a 48  gP.ax=.B.o.....H
0015066c  4e 01 53 b3 41 0e 0f c2-88 3e 0b 27 6c 34 59 bd  N.S.A....>.'l4Y.
0015067c  f0 ca db 1d ef be ad ba-00 00 00 00 00 00 00 00  ................
0015068c  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0015069c  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
001506ac  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
001506bc  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
001506cc  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
001506dc  00 00 00 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 00 00 00  ................
0:000> .writemem c:\Users\flare\Desktop\flag.bin 1503fc+4 l280; $$ 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.

```
Command
0:000> bp kernel32!VirtualAllocStub; $$ break at VirtualAlloc
0:000> g
ModLoad: 76b00000 76b25000   C:\Windows\SysWOW64\IMM32.DLL
Breakpoint 0 hit
eax=00484a38 ebx=024e1a48 ecx=00006c44 edx=00400000 esi=00000000 edi=024e1b65
eip=7742f9f0 esp=0019fdec ebp=0019fe20 iopl=0         nv up ei pl zr na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000246
KERNEL32!VirtualAllocStub:
7742f9f0 8bff            mov     edi,edi
0:000> gu; $$ get the alloc'd shellcode addr
0:000> r
eax=00610000 ebx=024e1a48 ecx=f8180000 edx=00610000 esi=00000000 edi=024e1b65
eip=00448ae3 esp=0019fe00 ebp=0019fe20 iopl=0         nv up ei pl zr na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             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; $$ set bp at RC6 decrypt routine
0:000> g
Breakpoint 1 hit
eax=00015160 ebx=024e1a48 ecx=02250000 edx=00637e28 esi=00000000 edi=024e1b65
eip=006105cd esp=0019fdec ebp=0019fdfc iopl=0         nv up ei pl nz ac po cy
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000213
006105cd e8f3230000      call    006129c5
0:000> dd esp l4; $$ check params, encrypted data buf, size
0019fdec  02250000 00015160 00637e28 00637e28
0:000> .dvalloc 0x300; $$ alloc new memory for setting raw flag
Allocated 1000 bytes starting at 00620000
0:000> .readmem c:\Users\flare\Desktop\flag.bin 00620000 l?280
Reading 280 bytes.
0:000> ed esp 00620000 280; $$ update params on the stack; dd esp l4
0019fdec  00620000 00000280 00637e28 00637e28
0:000> p; $$ step over the RC6 decrypt routine;
0:000> db 00620000 l280; $$ may the flag be with you :>
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 70 69 72 65 2c 0d 0a-54 6f 20 74 61 6b 65 20  spire,..To take 
00620050  6d 65 20 6f 6e 20 61 20-6a 6f 75 72 6e 65 79 20  me on a journey 
00620060  66 61 72 2c 0d 0a 46 72-6f 6d 20 77 68 65 72 65  far,..From 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 are.....The w
00620090  6f 72 6c 64 20 69 73 20-76 61 73 74 20 61 6e 64  orld is vast and
006200a0  20 66 75 6c 6c 20 6f 66-20 77 6f 6e 64 65 72 2c   full of wonder,
006200b0  0d 0a 42 75 74 20 68 65-72 65 20 49 20 61 6d 2c  ..But here I am,
006200c0  20 6a 75 73 74 20 6c 6f-73 74 20 69 6e 20 74 68   just lost in th
006200d0  75 6e 64 65 72 2c 0d 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
006200f0  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 7a 65 72 6f 65 73-20 65 78 70 6c 6f 64 65  d zeroes explode
00620120  2e 0d 0a 0d 0a 42 75 74-20 73 74 69 6c 6c 20 49  .....But still I
00620130  20 73 65 61 72 63 68 2c-20 66 6f 72 20 77 68 61   search, for wha
00620140  74 20 49 20 73 65 65 6b-2c 0d 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  70 20 61 6e 64 20 66 75-6c 6c 20 6f 66 20 74 68  p and full of th
00620190  6f 75 67 68 74 2c 0d 0a-41 20 70 6c 61 63 65 20  ought,..A place 
006201a0  77 68 65 72 65 20 6b 6e-6f 77 6c 65 64 67 65 20  where knowledge 
006201b0  63 61 6e 20 62 65 20 73-6f 75 67 68 74 2e 0d 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 
006201f0  73 6f 20 74 72 75 65 2c-20 69 74 27 73 20 71 75  so true, it's qu
00620200  69 74 65 20 61 20 73 69-67 68 74 2c 0d 0a 4f 68  ite a sight,..Oh
00620210  20 6d 79 20 46 4c 41 52-45 2c 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 0d 0a 4d 79 20 67  are-on.com..My g
00620250  75 69 64 69 6e 67 20 73-74 61 72 2c 20 6d 79 20  uiding star, my 
00620260  73 68 69 6e 69 6e 67 20-73 74 61 72 2e 0d 0a 00  shining star....
00620270  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. Time Travel Debugging - TTD.exe command line utility
2. Time Travel Debugging - JavaScript Automation
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