Shellcode CTF DS 2023

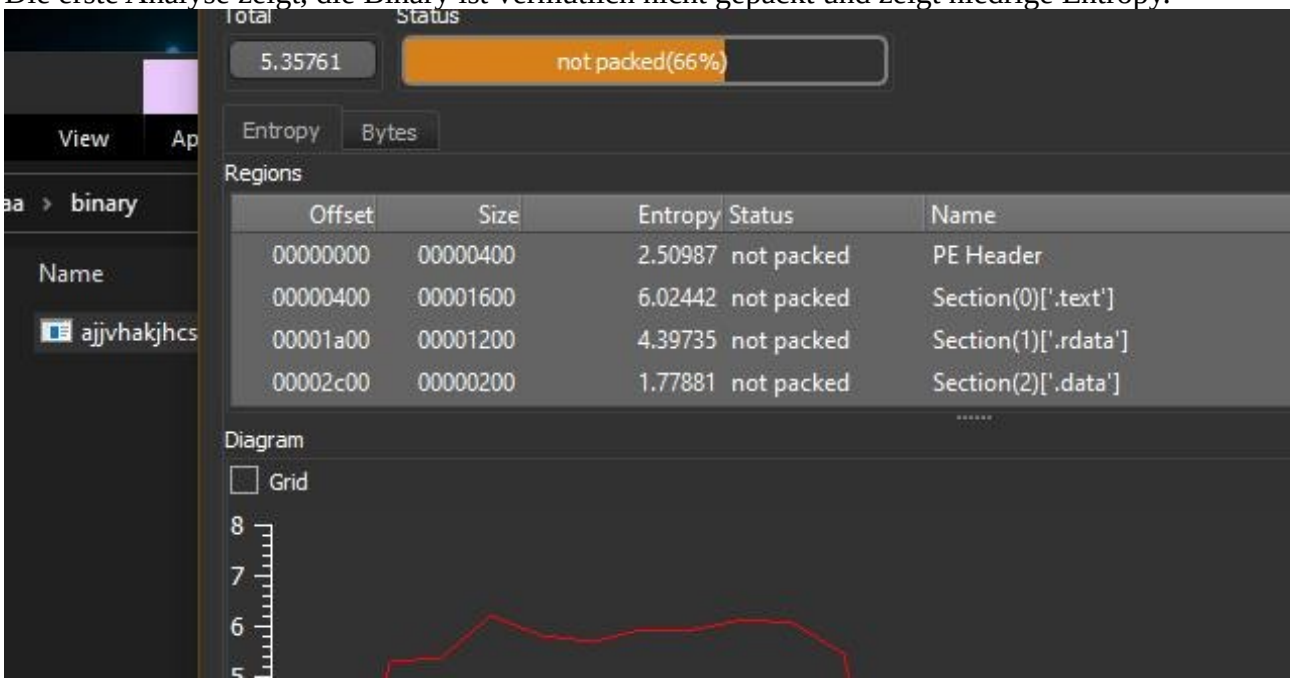
Links:
pw=dsctf-2023
https://file.0idea.dev/s/kxoOQ1Sd1IEwWFf

- 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?

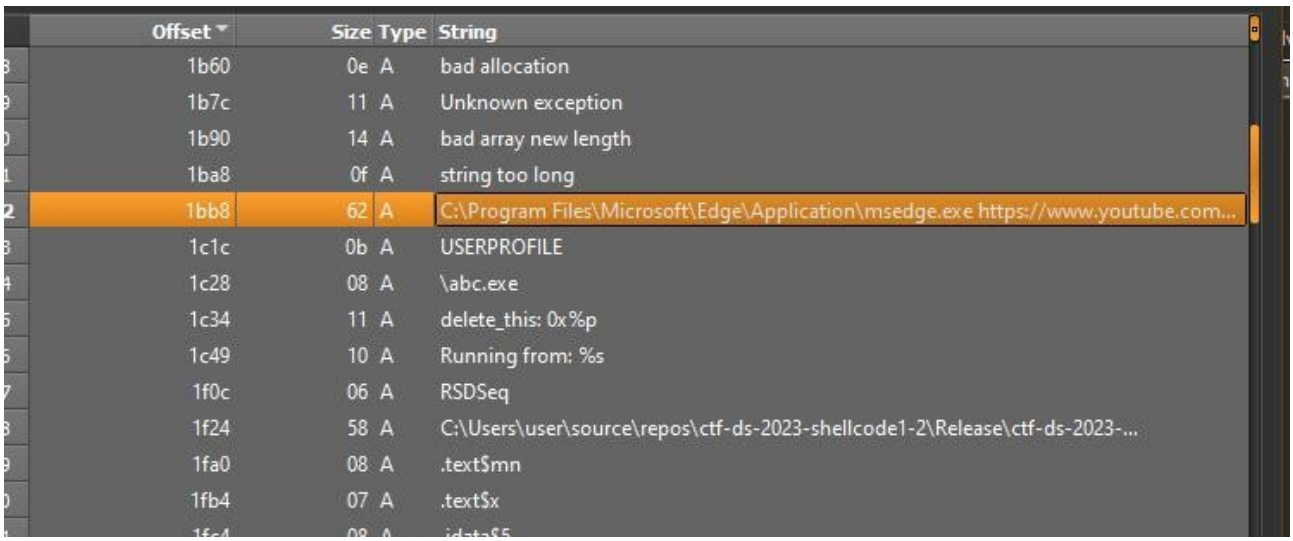
1. Start
Zuerst müssen wir klarstellen mit was wir es zu tun haben. Es geht hier um ein Malware-Befall mit mind. 2 Fragmenten. Wir fangen mit der Analyse der Binary an.

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.:

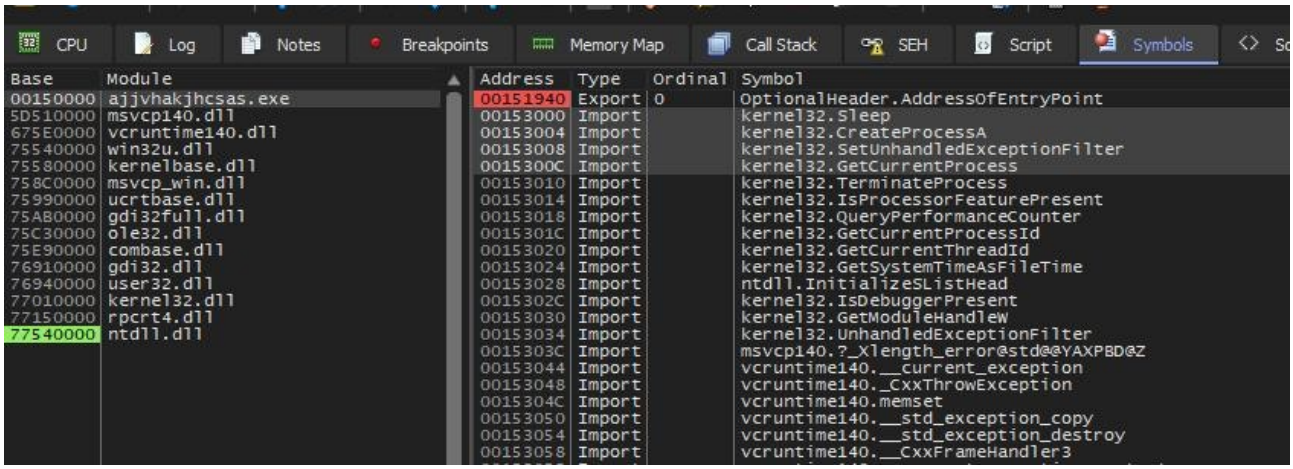
- Der pfad zu edge mit einem link zu einem youtube-video
- USERPROFILE
- \abc.exe
- delete_this: 0x%p
- Running from: %s



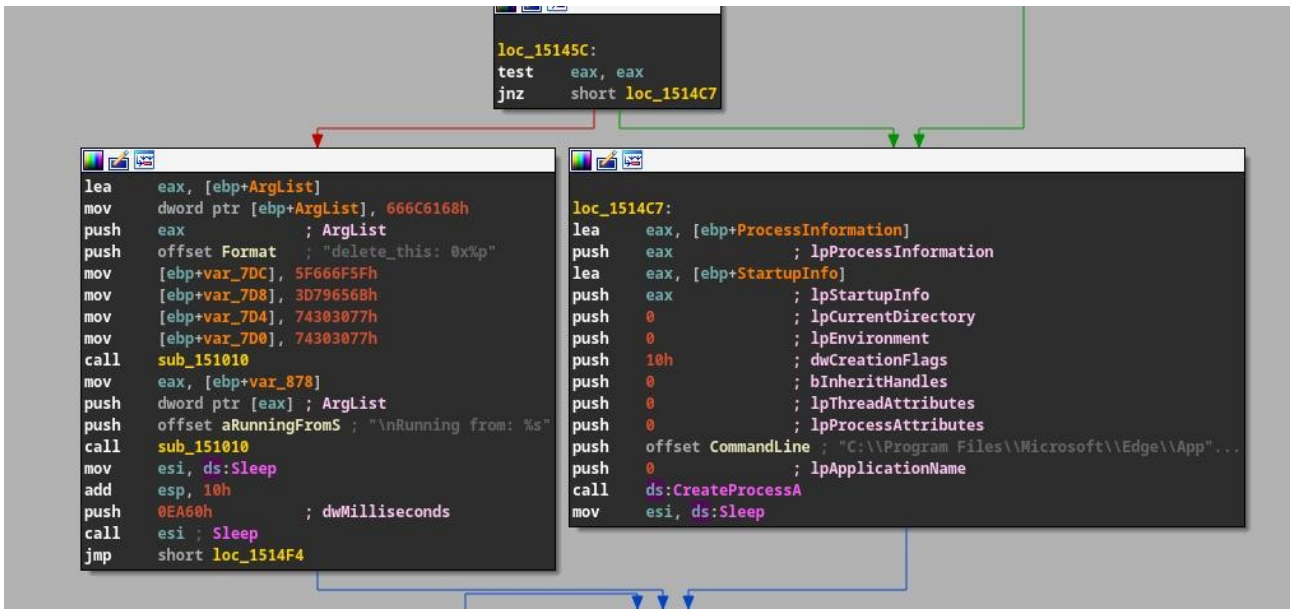
Offset	Size	Type	String
1b60	0e	A	bad allocation
1b7c	11	A	Unknown exception
1b90	14	A	bad array new length
1ba8	0f	A	string too long
1bb8	62	A	C:\Program Files\Microsoft\Edge\Application\msedge.exe https://www.youtube.com...
1c1c	0b	A	USERPROFILE
1c28	08	A	\abc.exe
1c34	11	A	delete_this: 0x%p
1c49	10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

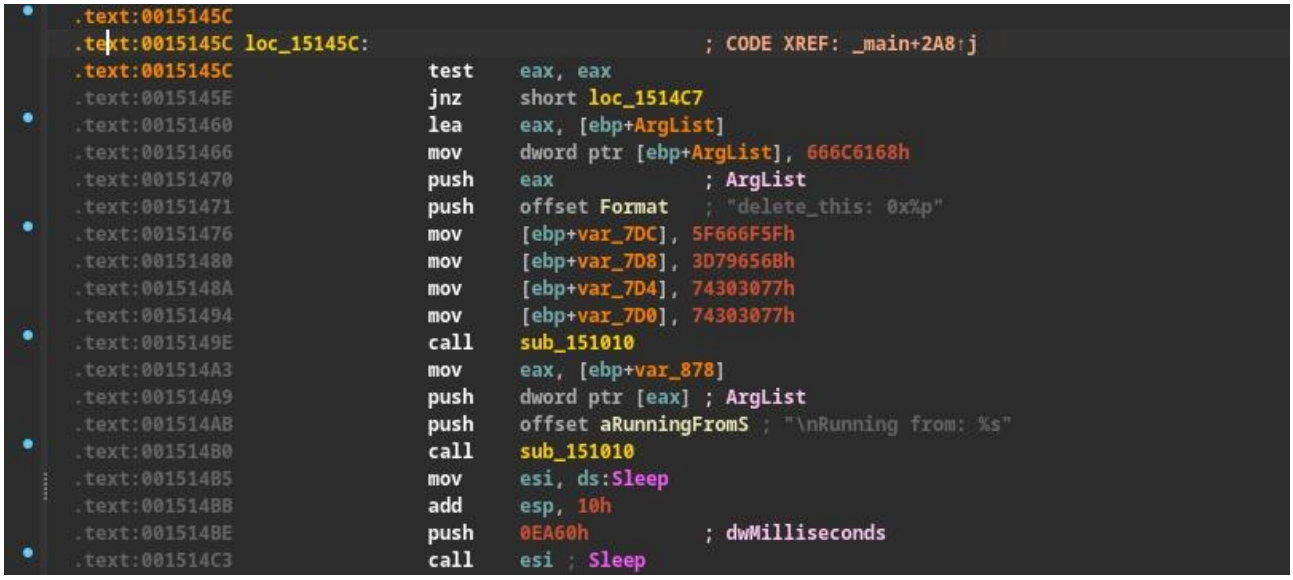
4. Imports
Laden wir die Binary in x64dbg, sehen wir hier ein paar interessante Imports bzw. Function-calls. Via kernel32 wird unter anderem folgende Funktion aufgerufen:

- CreateProcessA
- Sleep
- GetCurrentProcess



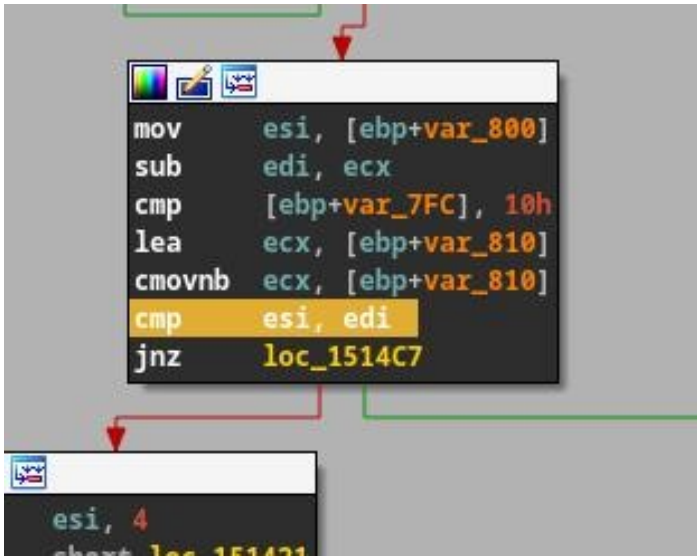
Suchen wir in der Binary nach den Function-calls Sleep und CreateProcessA stoßen wir auf einen interessanten part in +0x145c. Hier sind auch die Strings Running from, delete_this und der call zu Edge.exe zu finden





5. Comparison

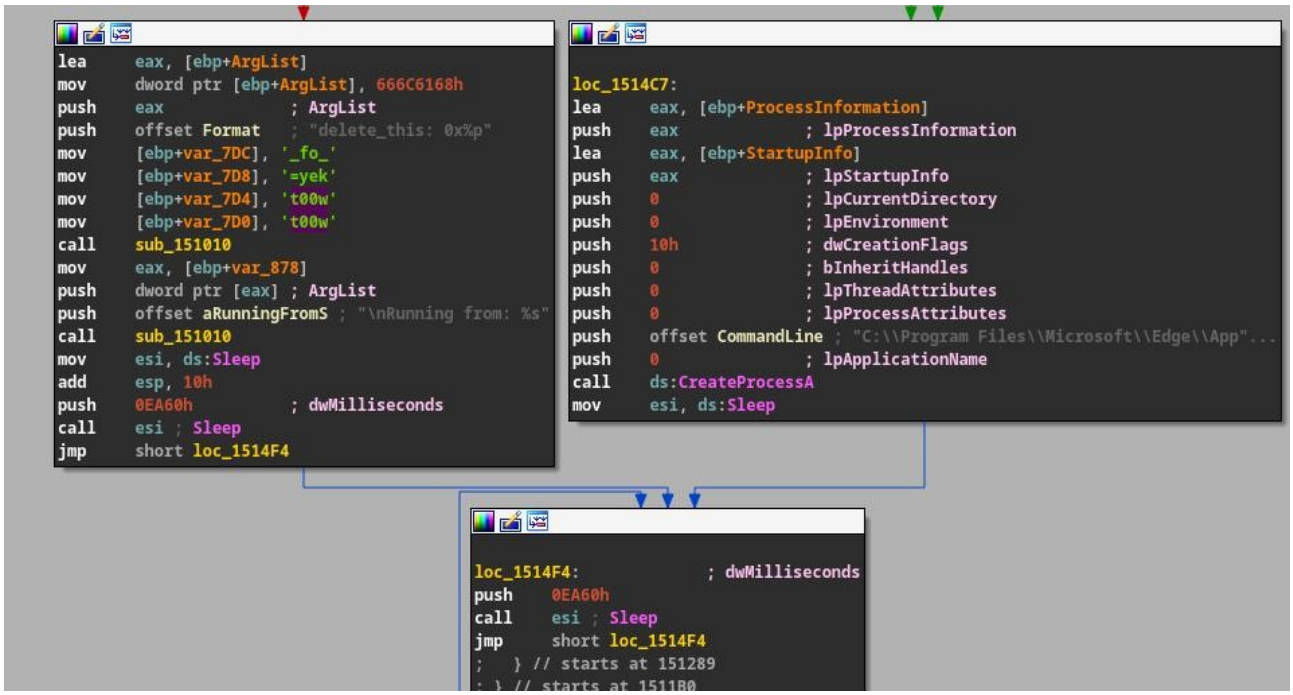
Scrollen wir hier etwas weiter nach oben sehen wir ein cmp esi, edi. Es scheint also ein Vergleich stattzufinden, der entweder das youtube-video öffnet oder aber etwas mit den 2 Strings anstellt.

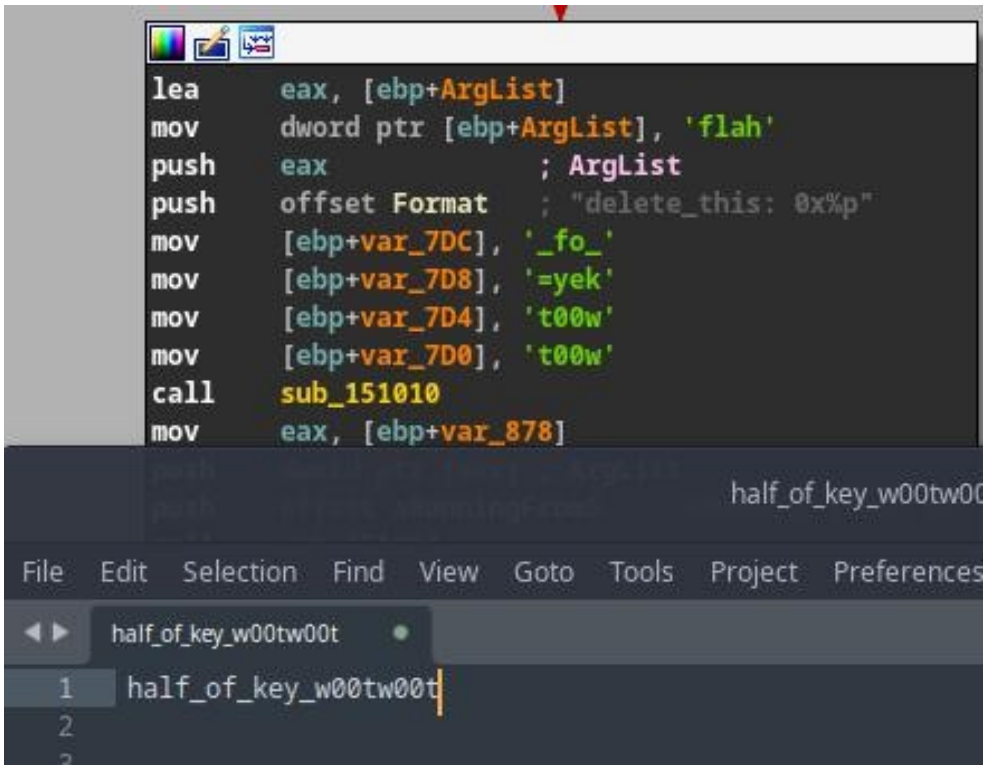


6. Analyse x64dbg
Untersuchen wir diesen Part manuell in x64dbg und forcieren hier den Sprung nach lnks (rot, zf=0) dann öffnet sich eine console mit folgendem inhalt:

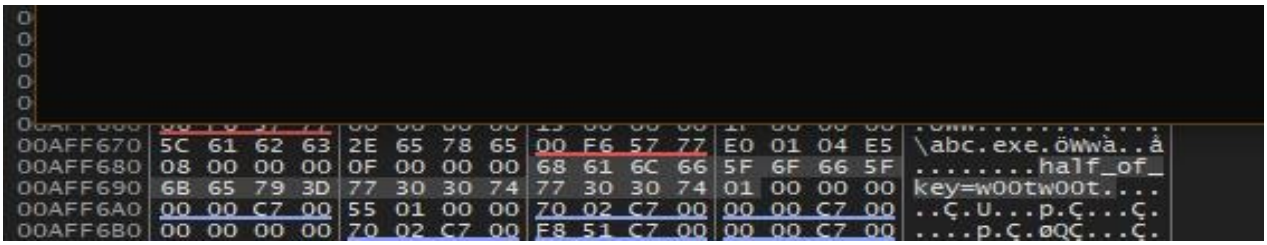
delete_this: 0x0014F0B0
Running from: C:\Users\user\Desktop\binary.exe

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.

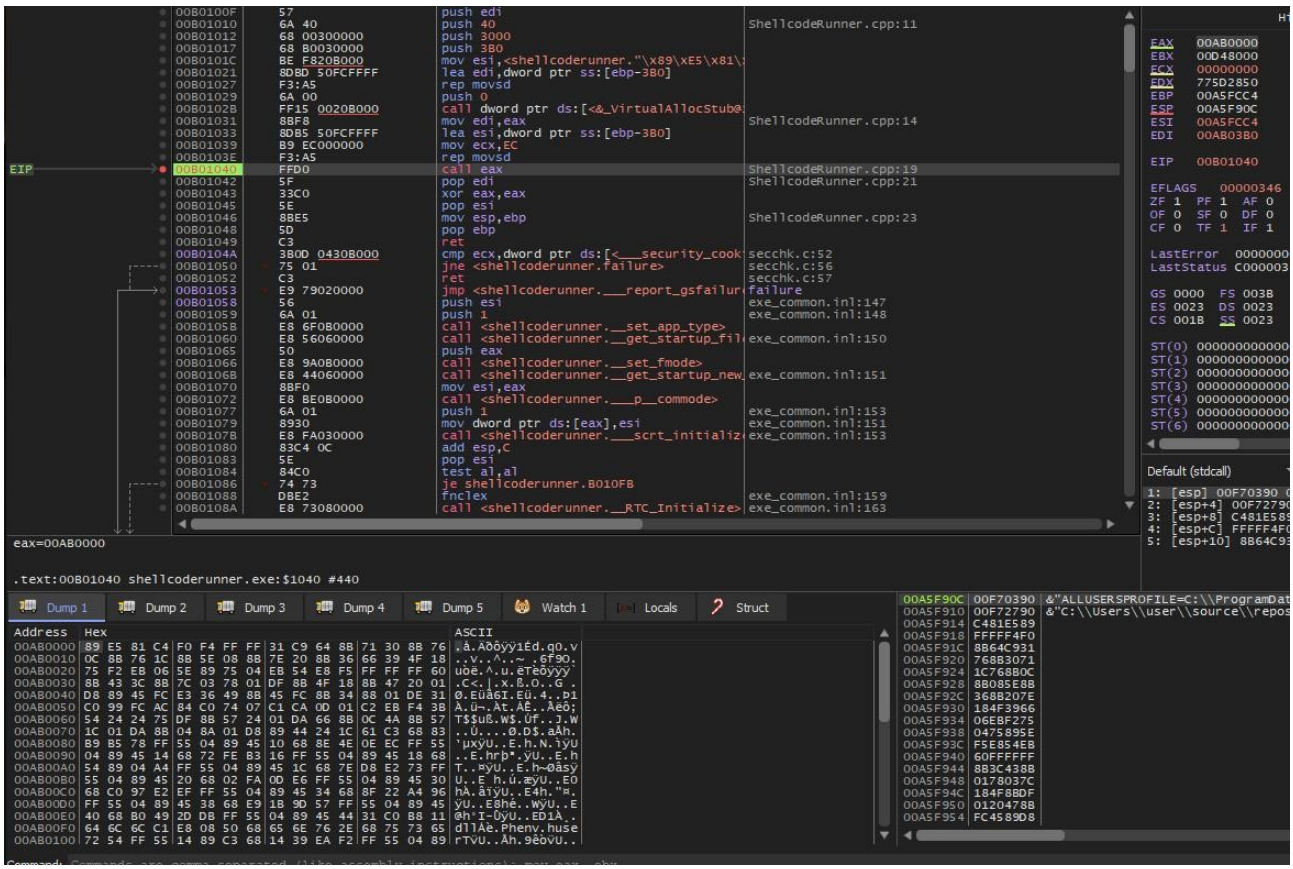


7. Shellcode
Die Binary scheint ab diesem Punkt nichts weiter zu tun als in einem Loop die Funktion sleep aufzurufen. Damit widmen wir uns erstmal dem shellcode. Um diesen auszuführen können wir eine entsprechende Binary bauen, die den shellcode aufnimmt, ein memory-bereich bereitstellt und diesen dann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

    //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)

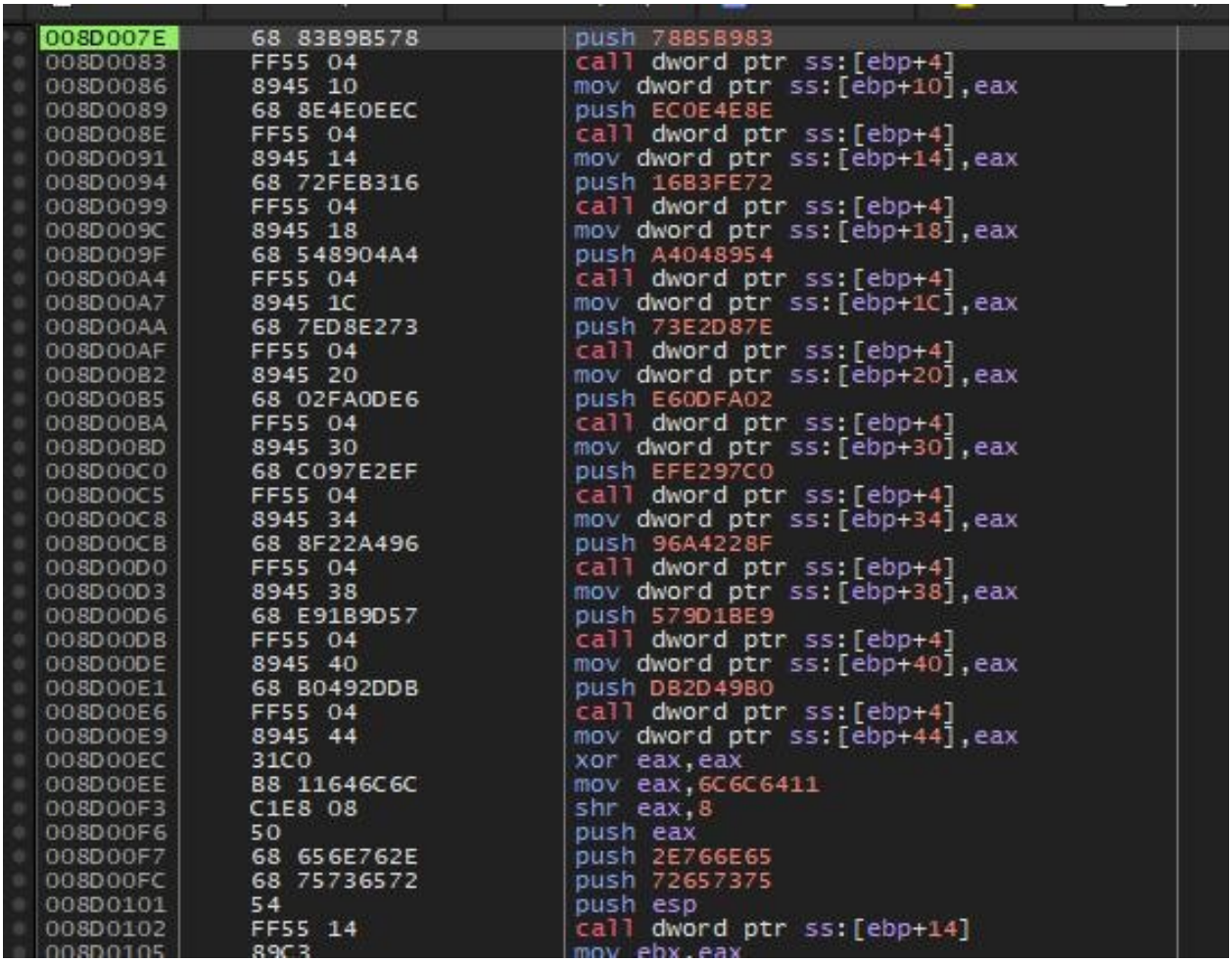


9. Dll-load und function-resolve

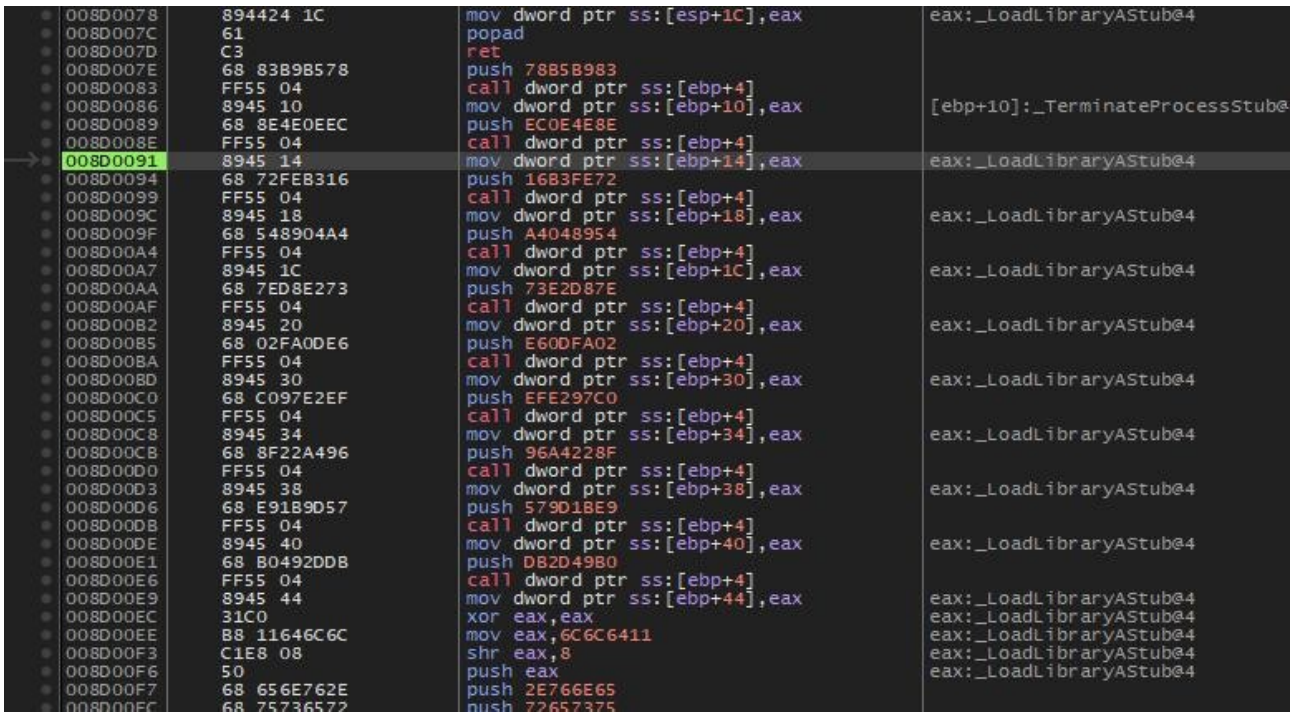
Das erste was ins Auge springt ist, sind die mov-calls zu esi. Wir sehen hier den wert [fs:030], [esi+0x0c] und [esi+0x1c]. Der shellcode versucht also via PEB, Ldr und Ldr.InInitOrder die einzelnen geladenen Module zu finden.

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

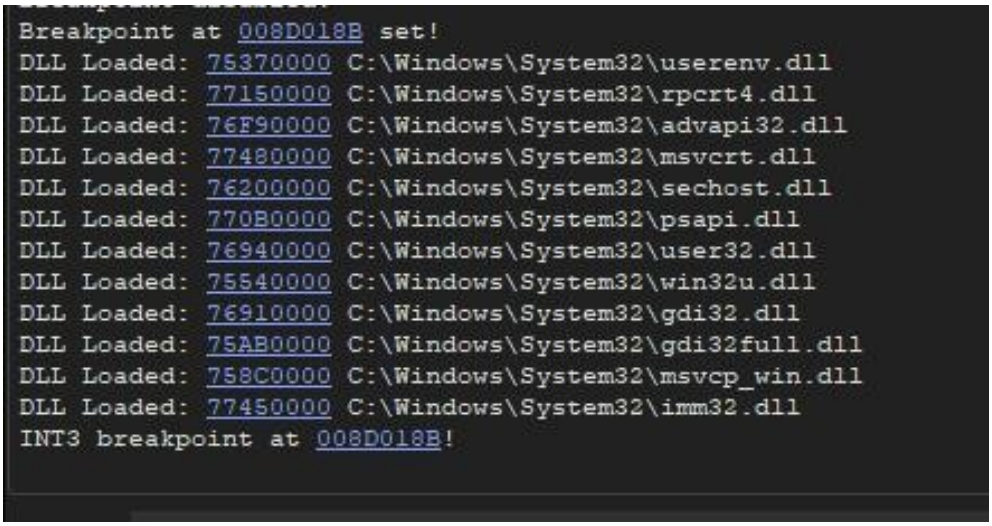
Malware-Entwickler nutzen dies um dynamisch die Adresse von kernel32.dll zu finden, mithilfe dessen dann via LoadLibrary weitere Module nachgeladen werden können. Vermutlich ist auch das hier der Fall. Wir sollten also ein paar Instruktionen weiter mehrere calls zur aufgelösten LoadLibrary finden.



X64dbg hilft uns hier und löst die function-calls entsprechend auf.

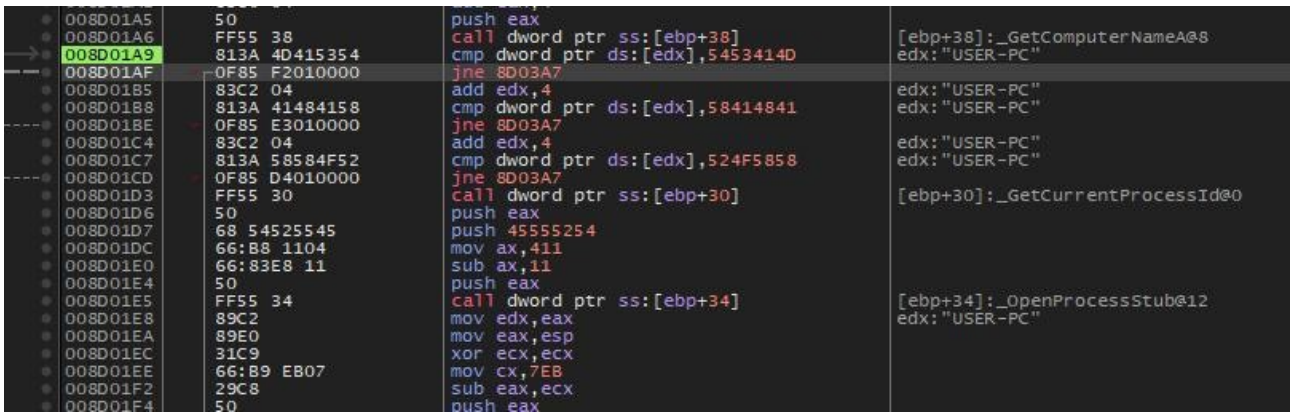


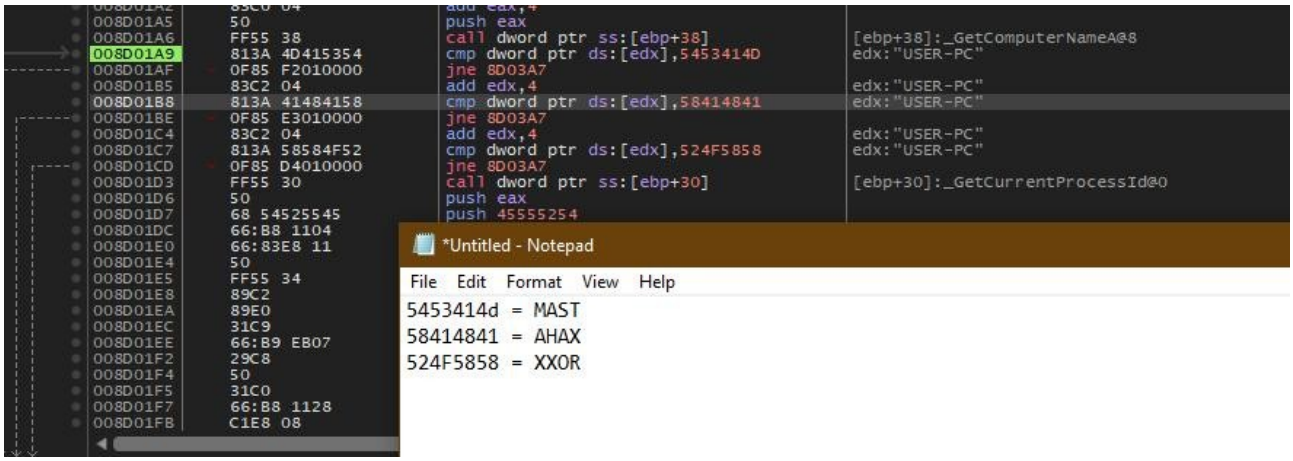
Durchläuft man all diese calls, sieht man auch im Log von x64dbg entsprechende einträge bzgl. Nachgeladener libraries.



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:

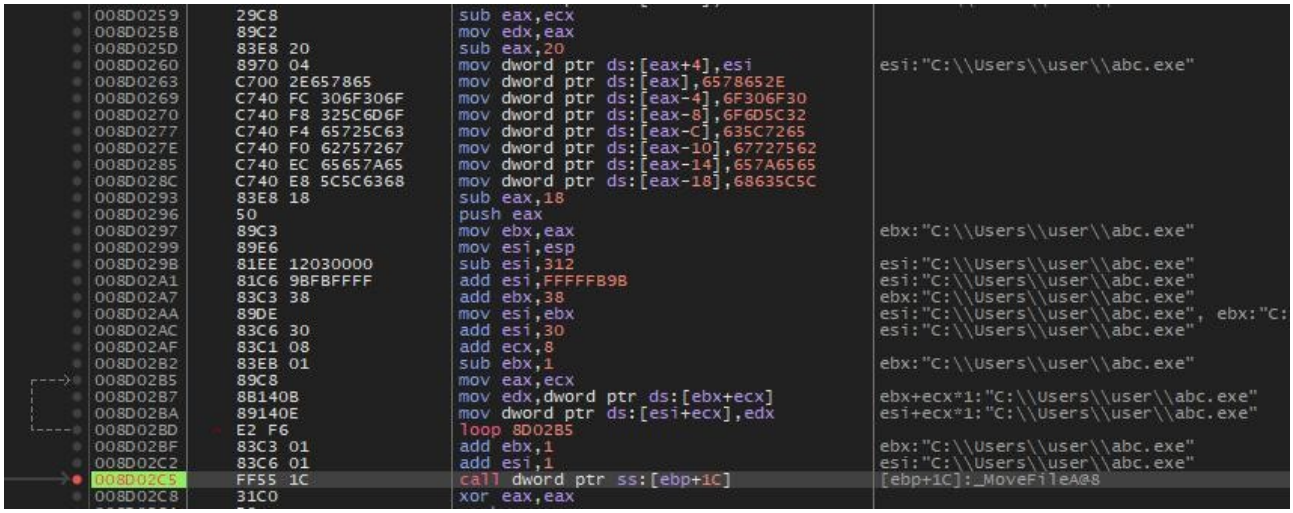




Der shellcode überprüft also den hostname gegen “MASTAHAXXXOR”. Ab diesem Punkt macht es sinn den hostname entsprechend zu ändern.

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 GetUserrprofiledirectory und MoveFileA.



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 ...láZ.....xáZ.p
005ADFC9 00 00 00 BB F4 AB 8B FE FF FF FF 68 E0 5A 00 9C ...»ô«.þÿÿÿhàZ..
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 ...Ã÷«.þÿÿÿ`....
```

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 entsrepchend umbennen. Damit der Host cheezeburger auflösen kann, kann die Datei C:\Windows\system32\drivers\etc\hosts entsprechend angepasst werden.

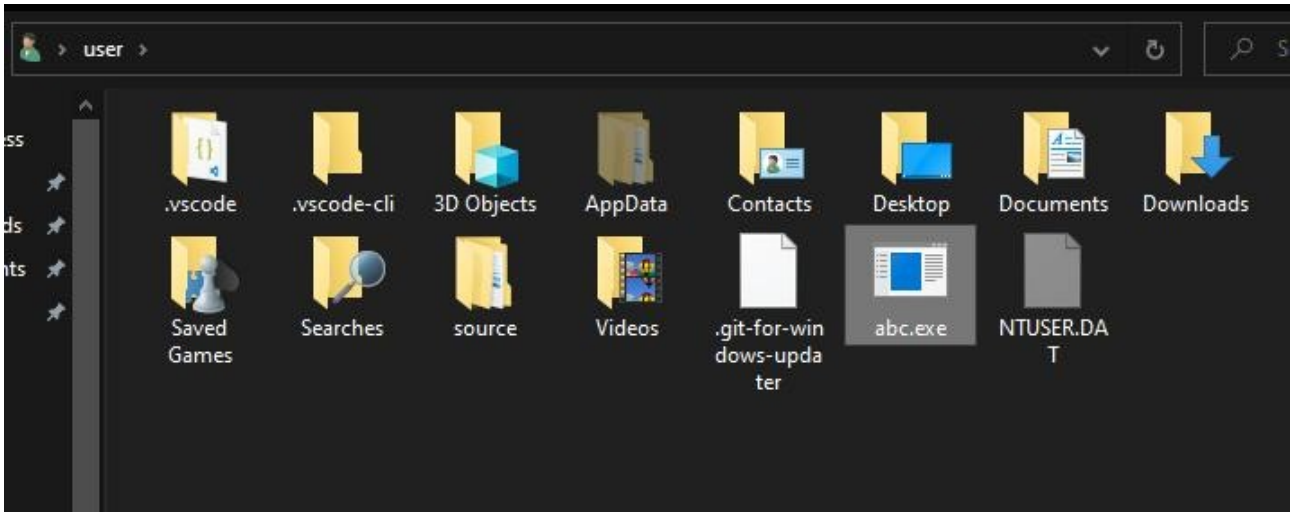
*Untitled - Notepad

File Edit Format View Help

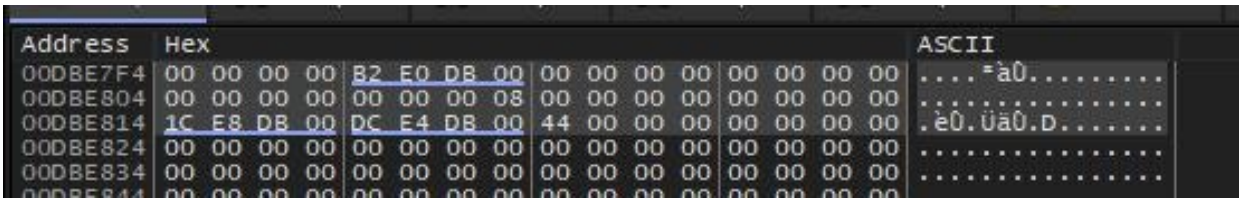
```
5453414d = MAST
58414841 = AHAX
524F5858 = XXOR

\\cheezeburger\c2\mo0o0o.exe
```

```
2  #
3  # This is a sample HOSTS file used by Microsoft TCP/IP
4  #
5  # This file contains the mappings of IP addresses to ho
6  # entry should be kept on an individual line. The IP ad
7  # be placed in the first column followed by the corresp
8  # The IP address and the host name should be separated
9  # space.
10 #
11 # Additionally, comments (such as these) may be inserte
12 # lines or following the machine name denoted by a '#'
13 #
14 # For example:
15 #
16 #      102.54.94.97     rhino.acme.com          # sourc
17 #       38.25.63.10     x.acme.com              # x cli
18
19 # localhost name resolution is handled within DNS itsel
20 #   127.0.0.1       localhost
21 #   ::1             localhost
22 172.16.0.216 cheezeburger
```

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

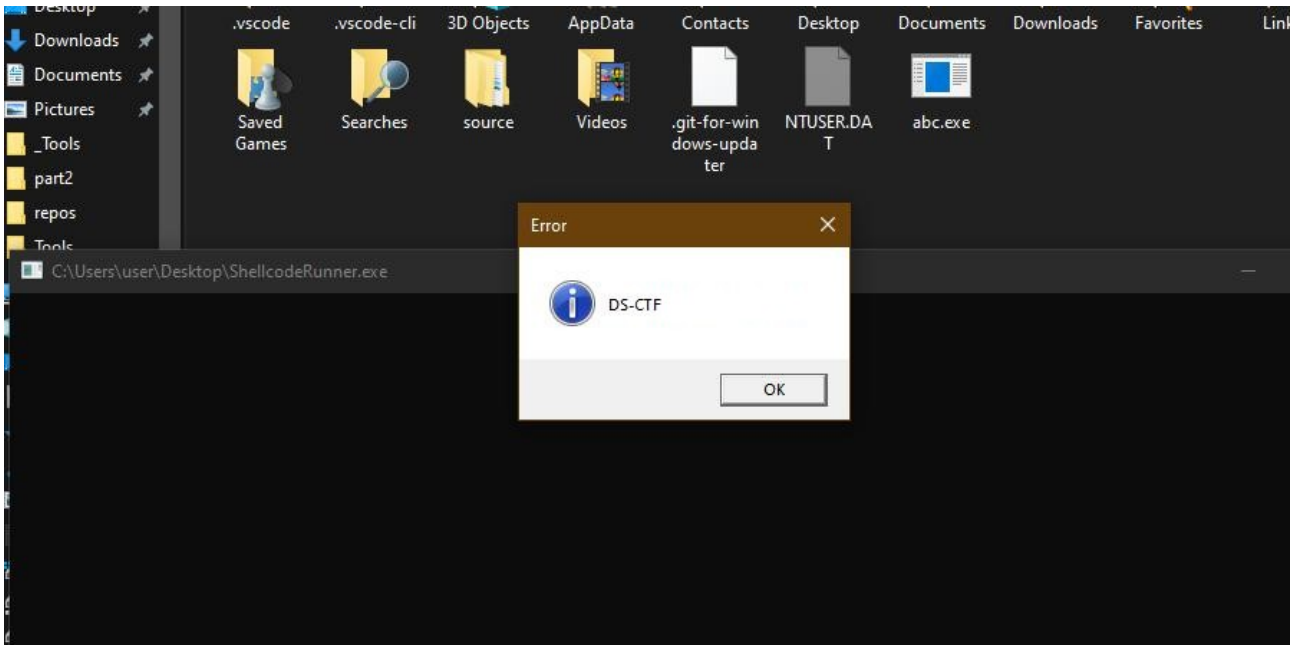


12. CreateProcessA
Als nächstes sehen wir ein Call zu CreateProcessA. Schauen wir uns die Parameter an, scheint nur lpCommandLine interesant zu sein.




```
C++                                                    Copy

BOOL CreateProcessA(
  [in, optional]      LPCSTR                lpApplicationName,
  [in, out, optional] LPSTR                 lpCommandLine,
  [in, optional]      LPSECURITY_ATTRIBUTES lpProcessAttributes,
  [in, optional]      LPSECURITY_ATTRIBUTES lpThreadAttributes,
  [in]                BOOL                  bInheritHandles,
  [in]                DWORD                 dwCreationFlags,
  [in, optional]      LPVOID                lpEnvironment,
  [in, optional]      LPCSTR                lpCurrentDirectory,
  [in]                LPSTARTUPINFOA        lpStartupInfo,
  [out]               LPPROCESS_INFORMATION lpProcessInformation
);
```

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.


Downloads
Documents
Pictures
_Tools
part2
repos
Tools
.vscode
.vscode-cli
3D Objects
AppData
Contacts
Desktop
Documents
Downloads
Favorites
Saved Games
Searches
source
Videos
.git-for-windows-updater
NTUSER.DAT
abc.exe
C:\Users\user\Desktop\ShellcodeRunner.exe
Error
DS-CTF
OK