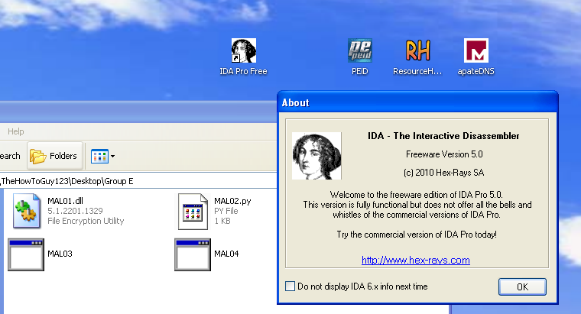
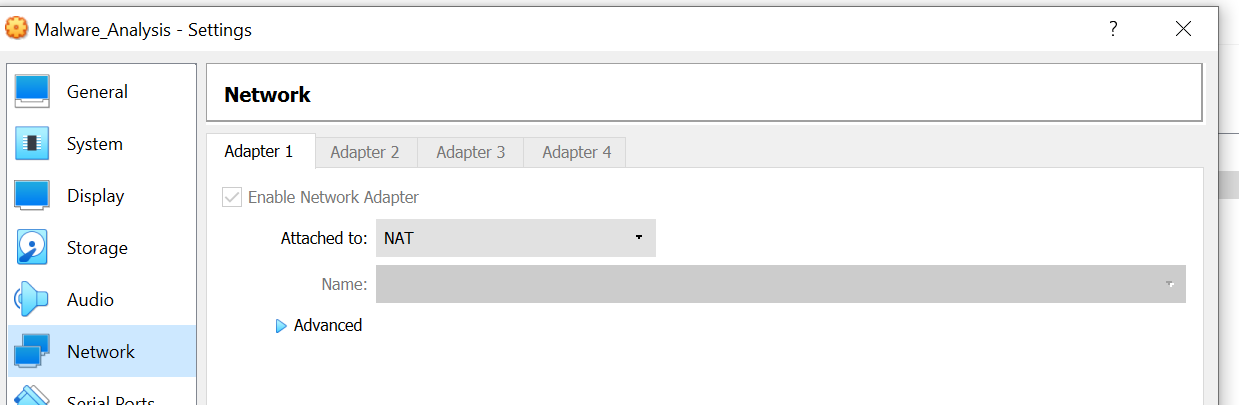
Malicious software analysis

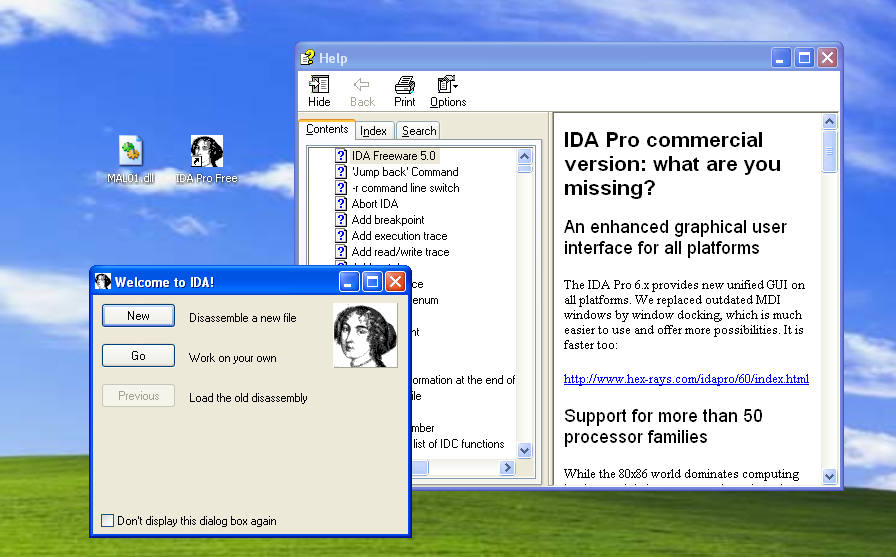
 IDA Pro Project

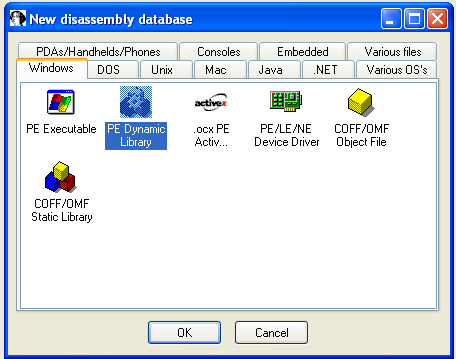
This Incident response report includes the following:

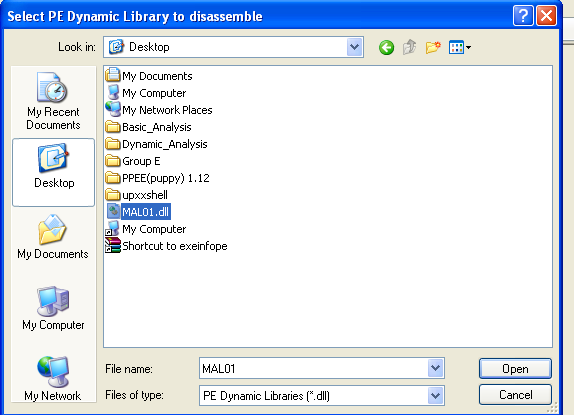
1. Setup for the malware
2. Mal01.DLL
3. Mal02.py
4. Mal03.exe
5. Mal04.exe

Firstly, the setup for the malware using a virtualized environment and setting the network configuration to NAT mode.

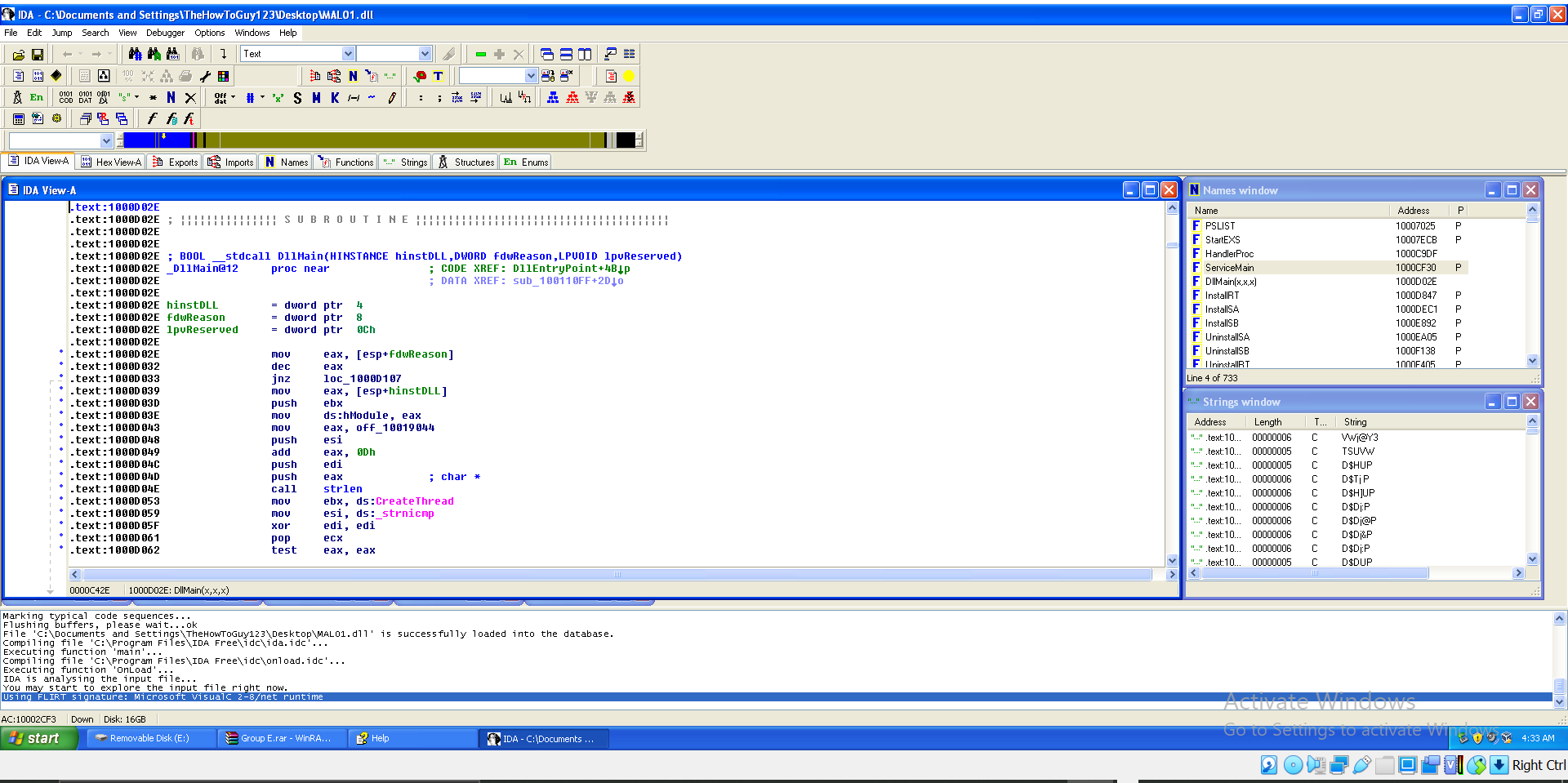
 **Part #1 🡪 Mal01.DLL**

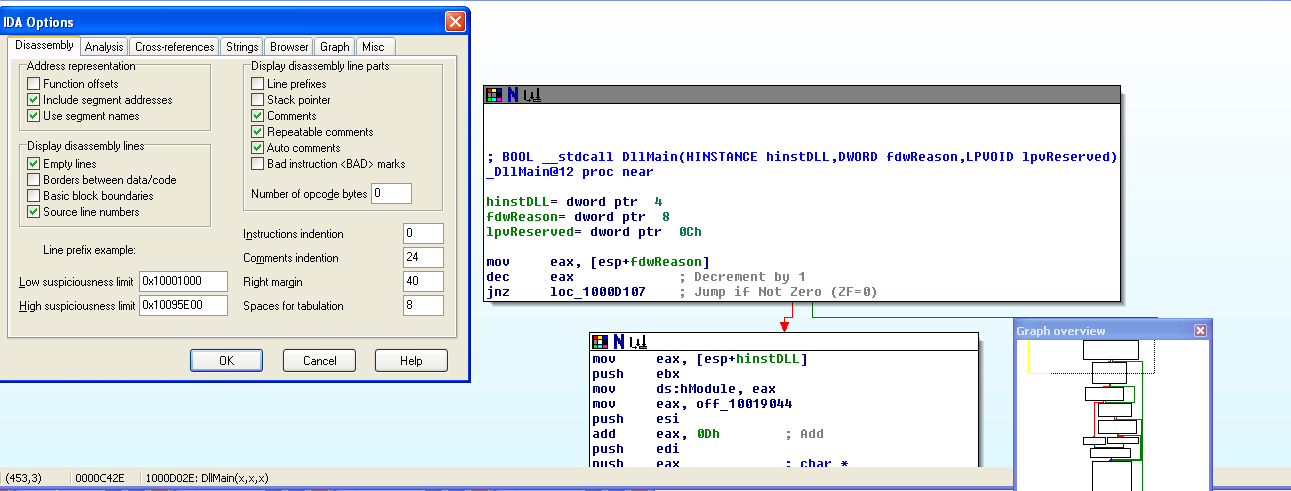






Questions:

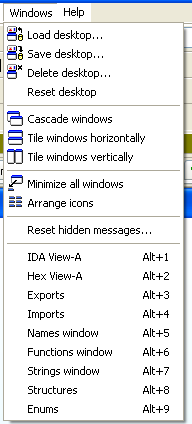
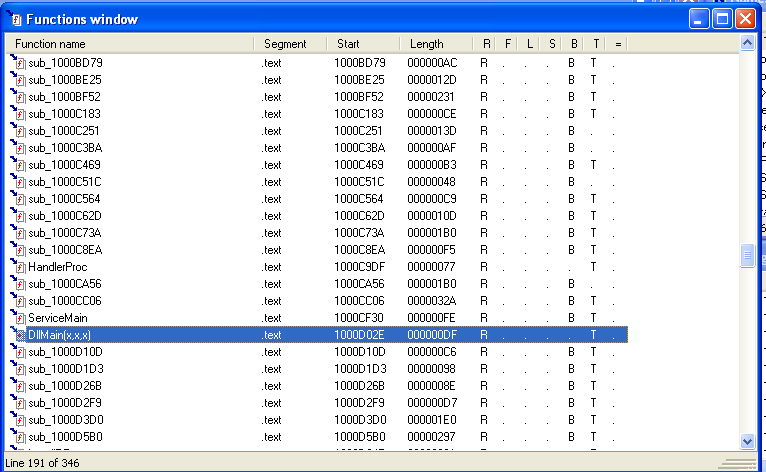
1. **What is the address of DllMain?**
2. **Use the Imports window to browse to gethostbyname. Where is the import located?**
3. **How many functions call gethostbyname?**
4. **Focusing on the call to gethostbyname located at 0x10001757, can you figure out which DNS request will be made?**
5. **How many local variables have IDA Pro recognized for the subroutine at 0x10001656?**
6. **How many parameters has IDA Pro recognized for the subroutine at 0x10001656?**
7. **Use the Strings window to locate the string \cmd.exe /c in the disassembly. Where is it located?**
8. **What is happening in the area of code that references \cmd.exe /c?**
9. **In the same area, at 0x100101C8, it looks like dword\_1008E5C4 is a global variable that helps decide which path to take. How does the malware set dword\_1008E5C4? (Hint: Use dword\_1008E5C4’s cross-references.)**
10. **A few hundred lines into the subroutine at 0x1000FF58, a series of comparisons use memcmp to compare strings. What happens if the string comparison to robotwork is successful (when memcmp returns 0)?**
11. **What does the export PSLIST do?**



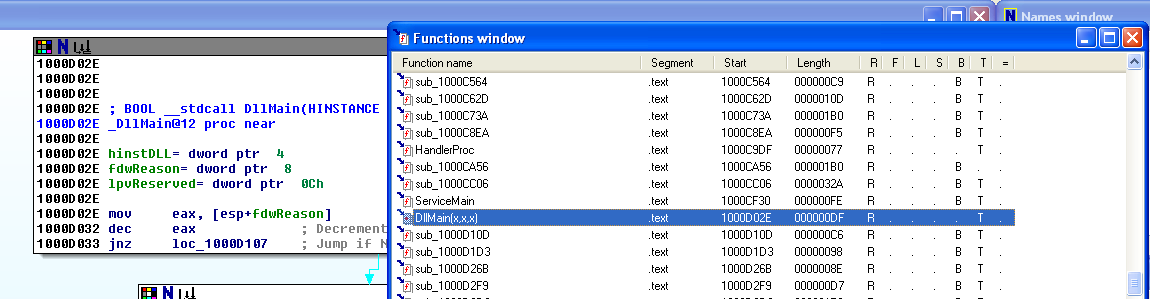
In order to make it more practical I will edit the graphical mode in order that automatically excludes the line number and the opcodes by changing the settings from options -> general and set the line prefix and auto comments.



**1- What is the address of DllMain?**

Since DllMain is function the IDA pro provides the ability to see the functions within the code by windows -> functions window from the tool bar or by alt+6 to see the address of the function.



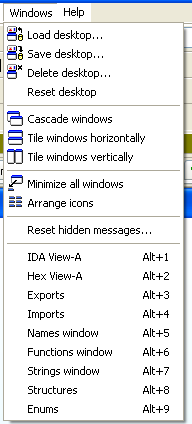




The address is (0x1000D02E).

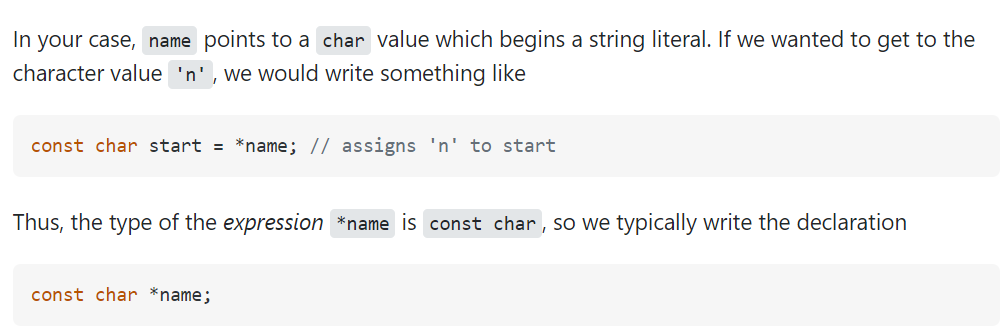
**2-Use the Imports window to browse to gethostbyname. Where is the import located?**

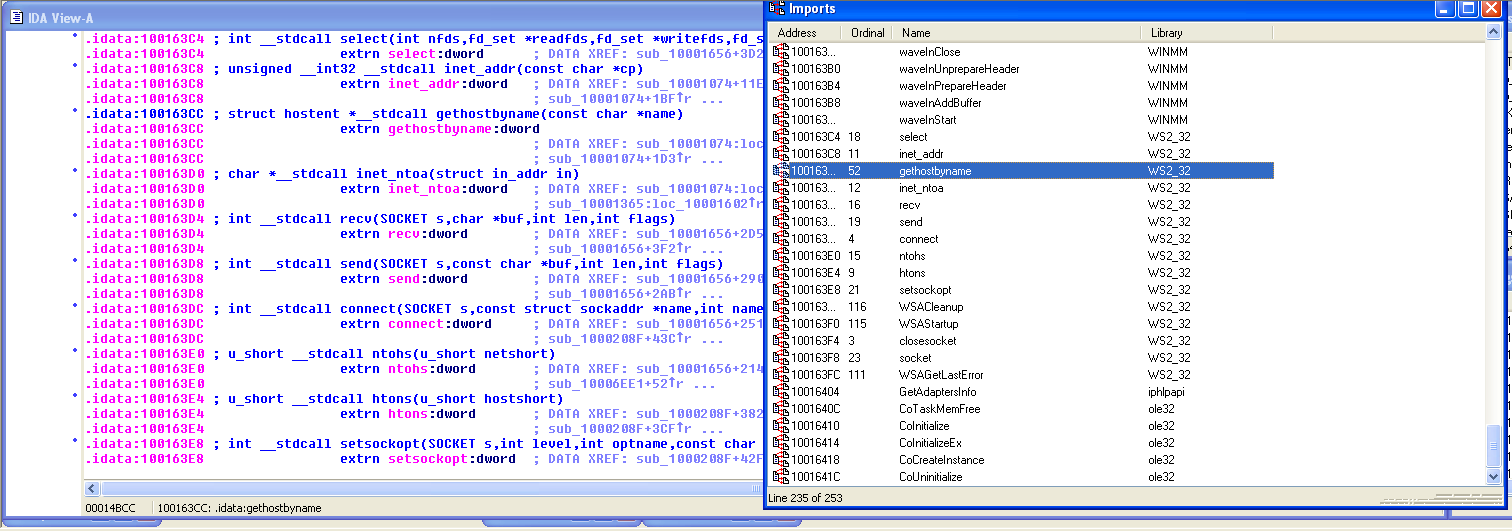
there are three options to see the imports:

1. From the open windows in the tool bar
2. From windows -> imports window
3. Alt+4



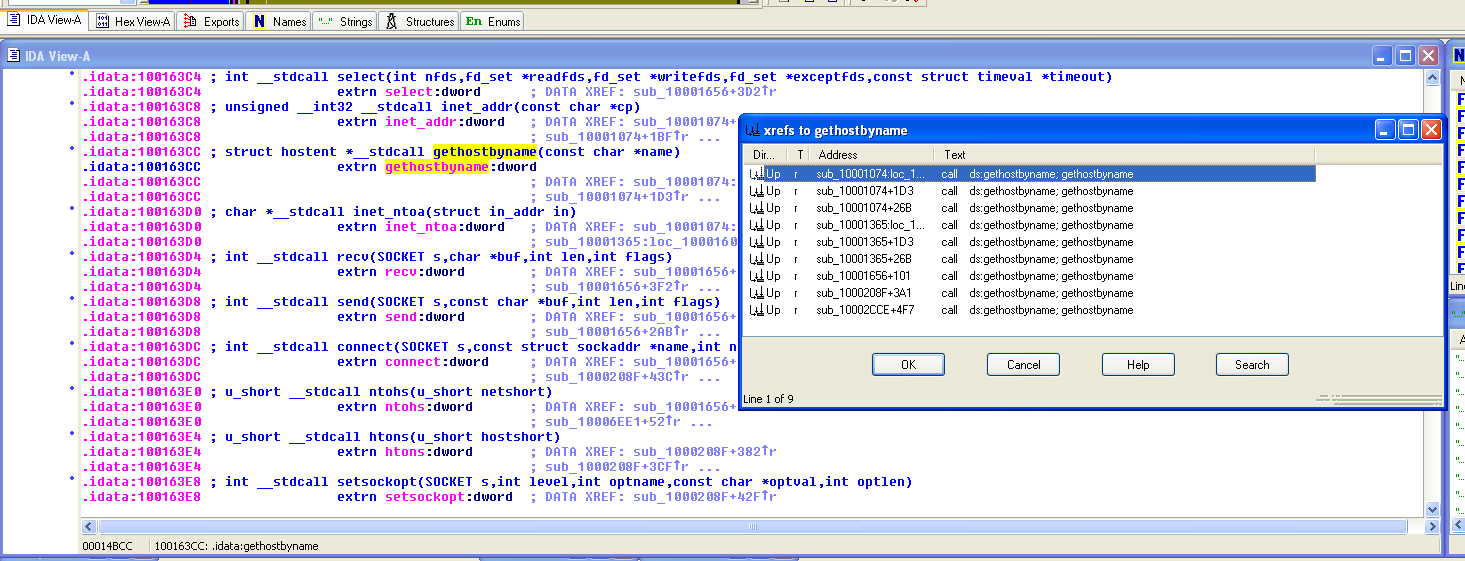


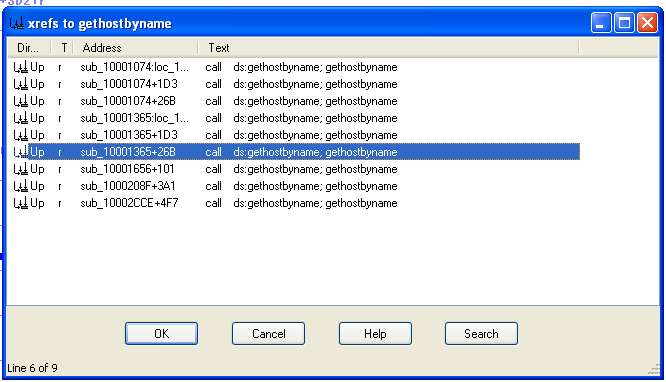
As we can see in the screen below that gethostbyname from the imports is located at (0x100163CC). and by double clicking on the imports, we can see that the imported function takes a parameter that point to char



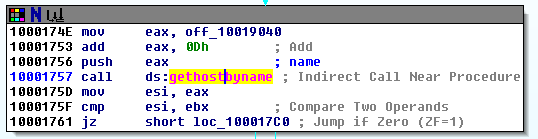
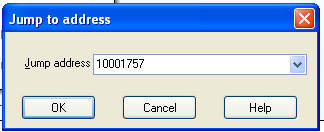


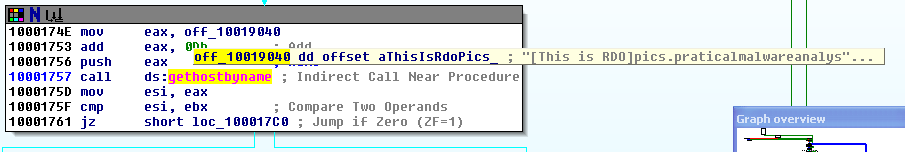
**3-How many functions call gethostbyname?**

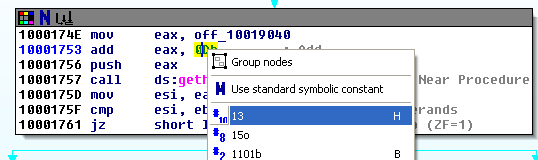
In order to see the number of calling gethostbyname we need to click on the function from the graph mode and press on X to see all the references

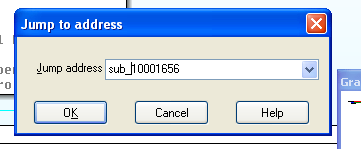
We can see 9 calls for the function and all of the calls are r which stands for read.

**4-Focusing on the call to gethostbyname located at 0x10001757, can you figure out which DNS request will be made?**

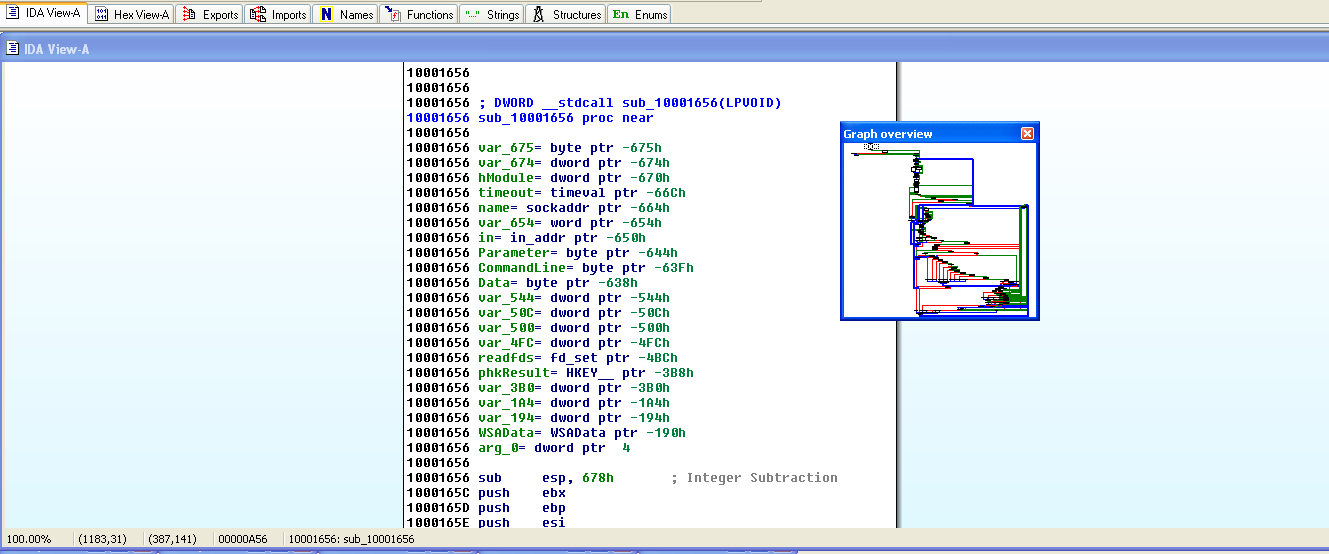
In order to go to specific location, we can press G and enter 0x10001757 and press enter

As we say that the function takes one parameter which is pointer to string and store it in EAX which takes the string from 0ff\_10019040 and store it there if we hover over it, we can see

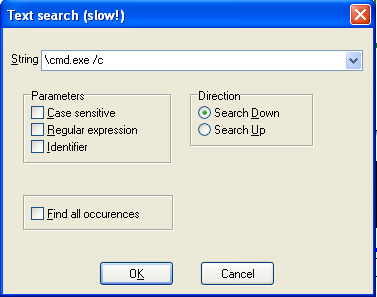
The 0ff\_10019040 points to variable aThisIsRdoPics\_ that contains “[This is RDO]pics.practicalmalwareanalysis.com” then after EAX=[This is RDO]pics.practicalmalwareanalysis.com we will add 0Dh which is 13 by skipping [This is RDO] and the DNS request will be made to pics.malwareanalysis.com.

**5-How many local variables have IDA Pro recognized for the subroutine at 0x10001656 and parameters have IDA Pro recognized for the subroutine at 0x10001656?**

Firstly, we need to jump to the location by pressing on g and type the subroutine address (sub\_10001656) and press ok. It includes 20 local variables and a parameter as all together 21 local variable and a parameter.

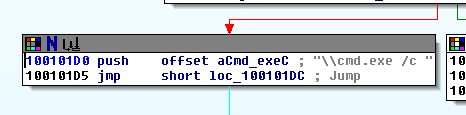


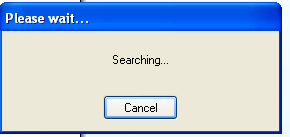
Local variables and parameters are referenced by the EBP but the local variables are located at negative offset that’s how we specified that out of 21-> 20 are variables they can be prefixed as a var\_ , name or even a commandline. But the parameters are located at positive offset and can be prefixed by arg\_ or positive offset such as arg\_0=dword ptr 4.

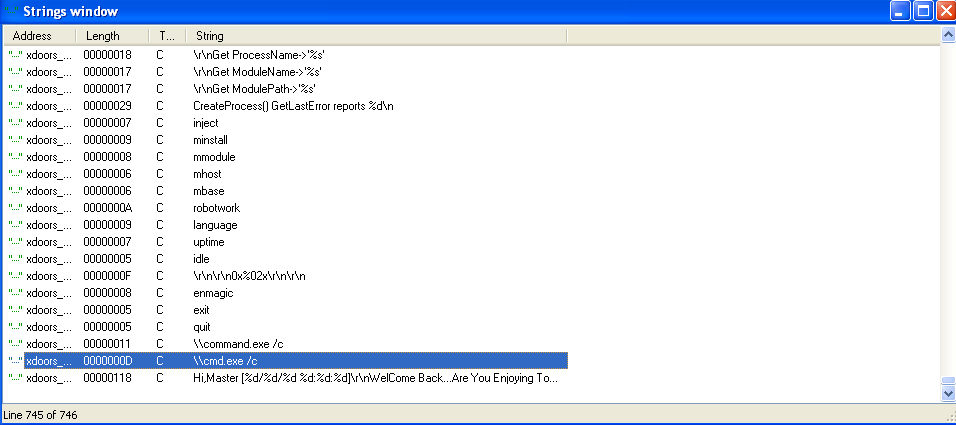
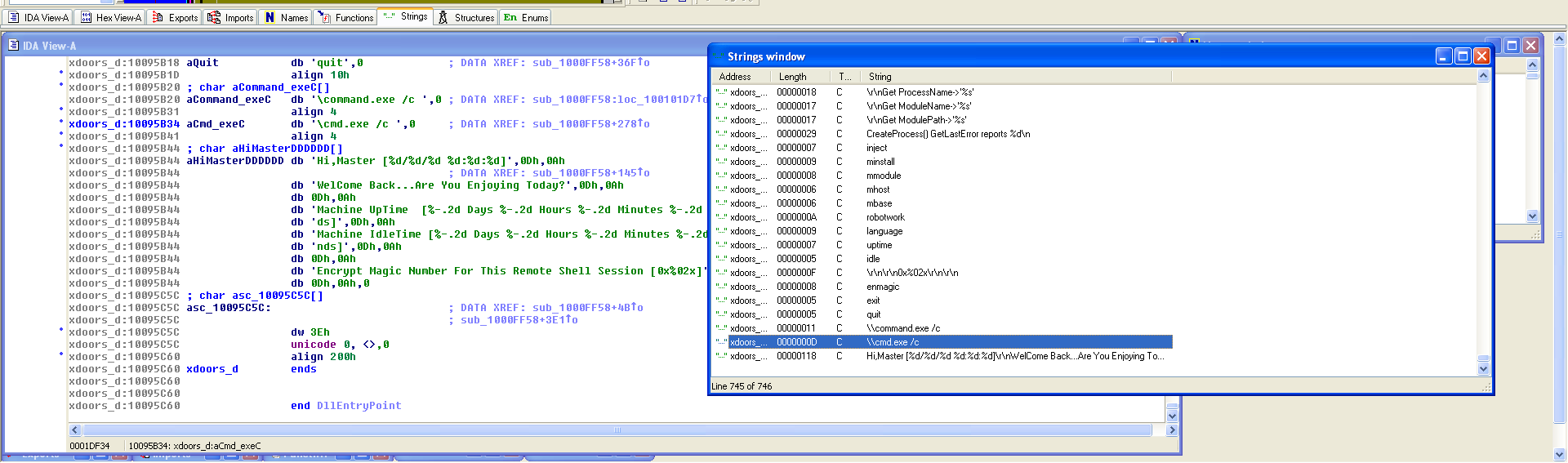
**7-Use the Strings window to locate the string \cmd.exe /c in the disassembly. Where is it located?**

We can use alt+t in order to search for a specific string

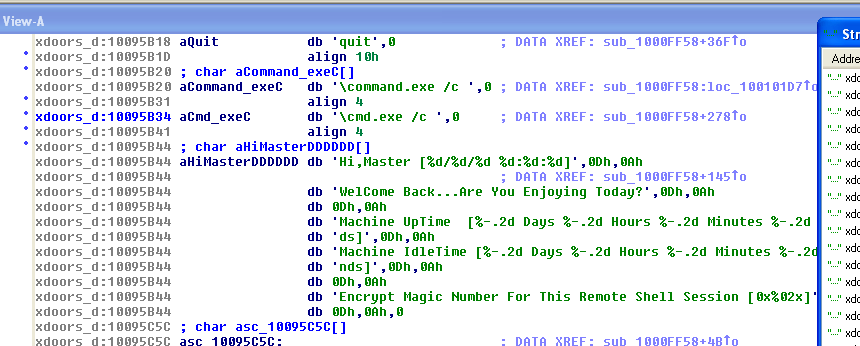
We can see that is located at (0x100101D0).



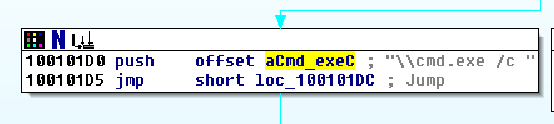


The required is by using the strings window from the right opened tabs there is the window tab or by alt+7

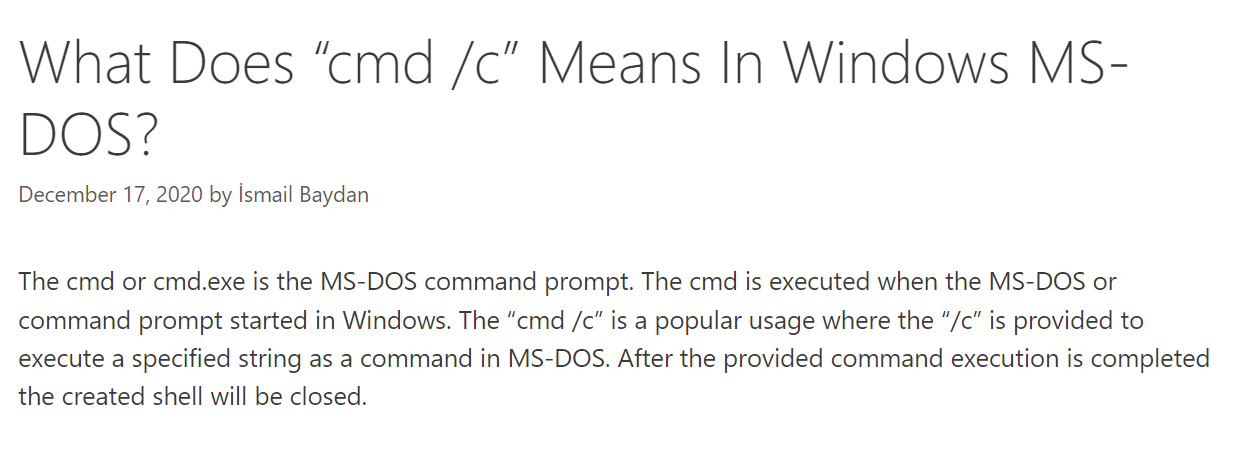


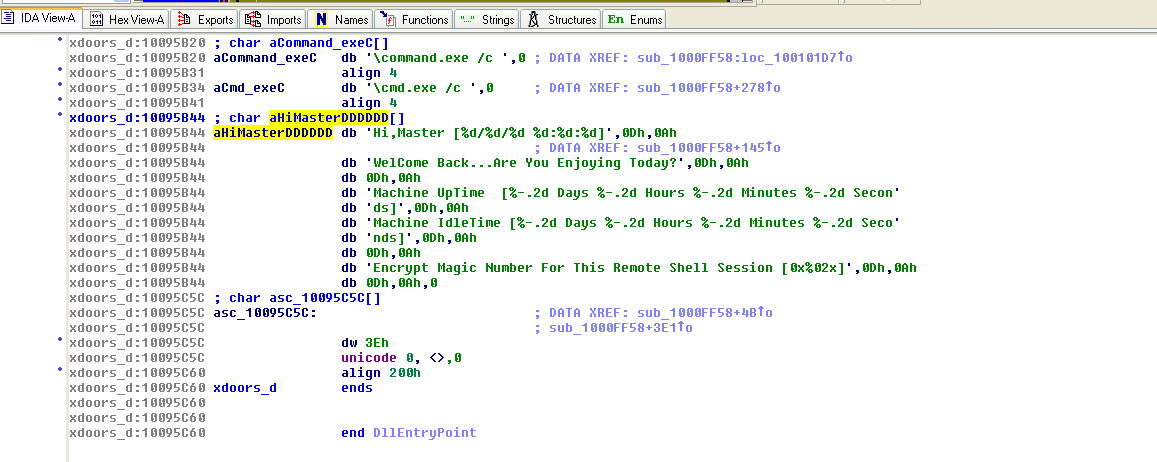


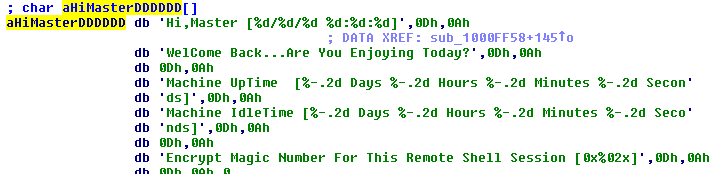
As shown, we can see that the strings are located at (0x100101D0) within the subroutine sub\_1000FF58.

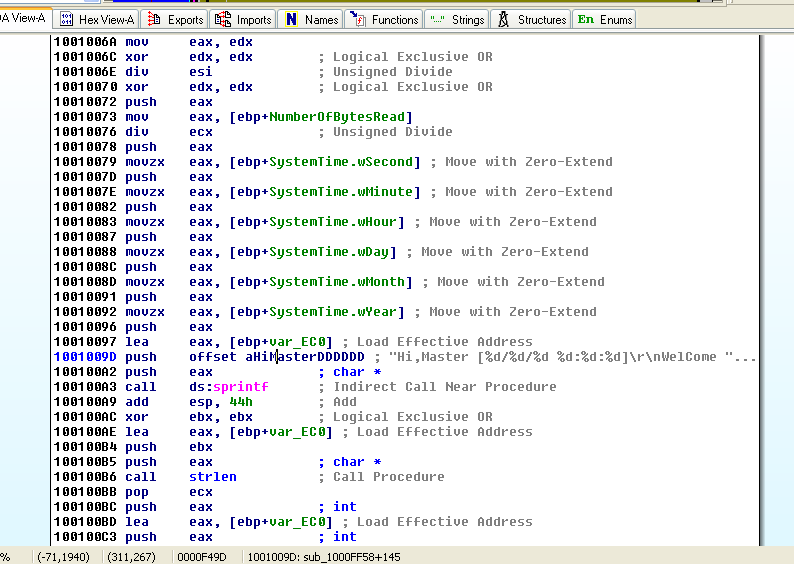
**8-What is happening in the area of code that references \cmd.exe /c?**

the cmd.exe /c is used in order to open a new instance of cmd.exe and executes the command after c/

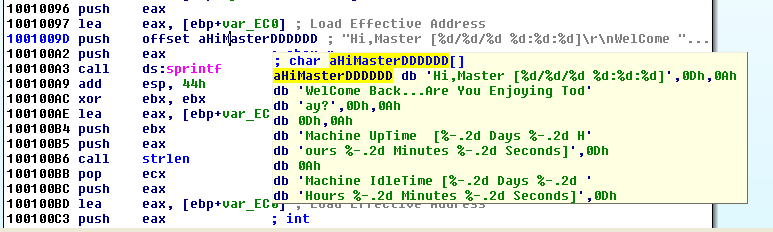




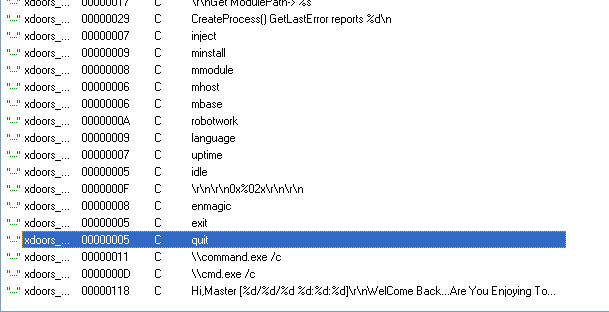
At the top of it we can find this at the offset 1001009D aHiMasterDDDDDD that is called at this offset that includes long message that print information about the time and might be referenced to the remote shell







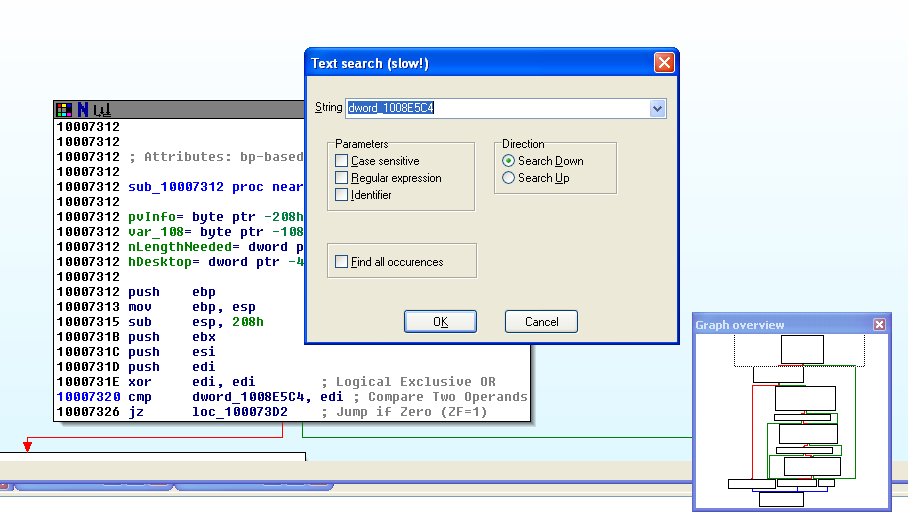


Although through the function there are offsets with strings that might indicate an activity such as

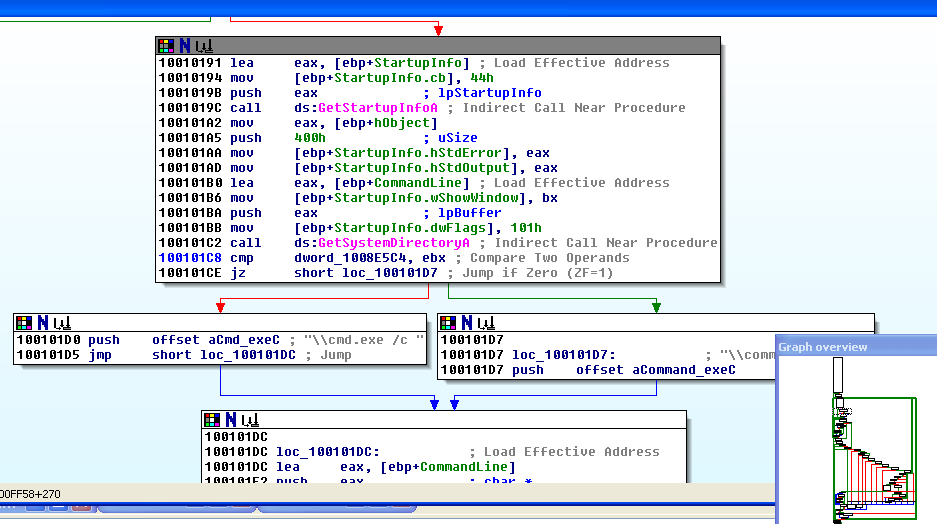


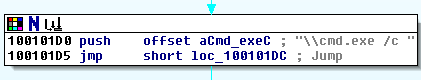
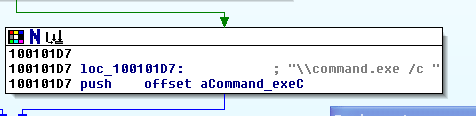
Some of them might be a command line or module such as mmodule

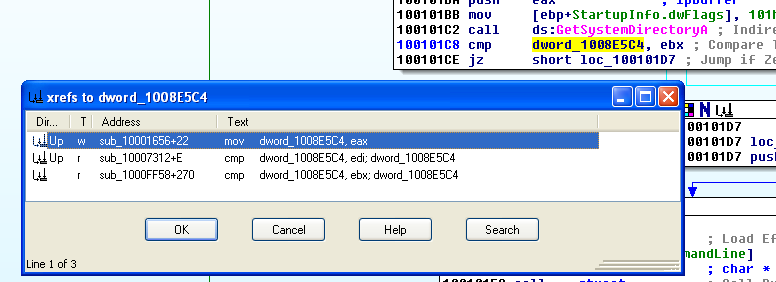
* Inject that can be used for process injection
* And create process

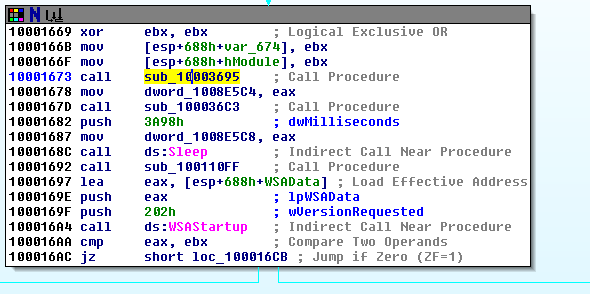
**9-In the same area, at 0x100101C8, it looks like dword\_1008E5C4 is a global variable that helps decide which path to take. How does the malware set dword\_1008E5C4? (Hint: Use dword\_1008E5C4’s cross-references.)**

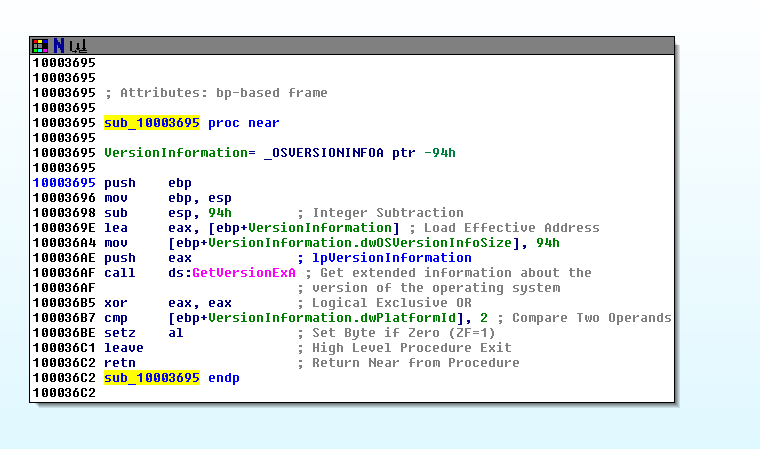
to get find dowrd\_1008E5C4 I used alt+t 🡪 get it from the IDA pro cheat sheet.



we are comparing the content in the variable dword\_1008E5C4 with EBX after calling the function GetSystemDirectoryA that retrieves the path of the system directory that contains files such as dynamic linked libraries and drivers. We will check EBX stores \\cmd .exe /c or [\\command.exe](file:///\\command.exe) /c based on the operating system

if we check the where it is referenced

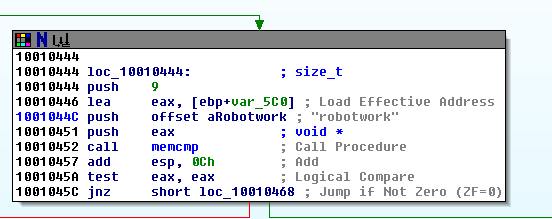


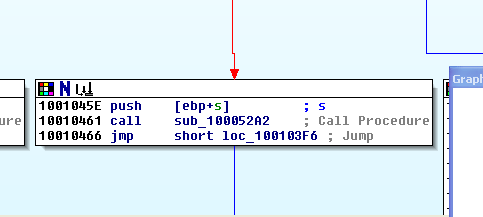
we can check the first then we will get the sub\_10003695 and by moving to we can get the function where we will get the version of operating system to get which option to compare it with EBX

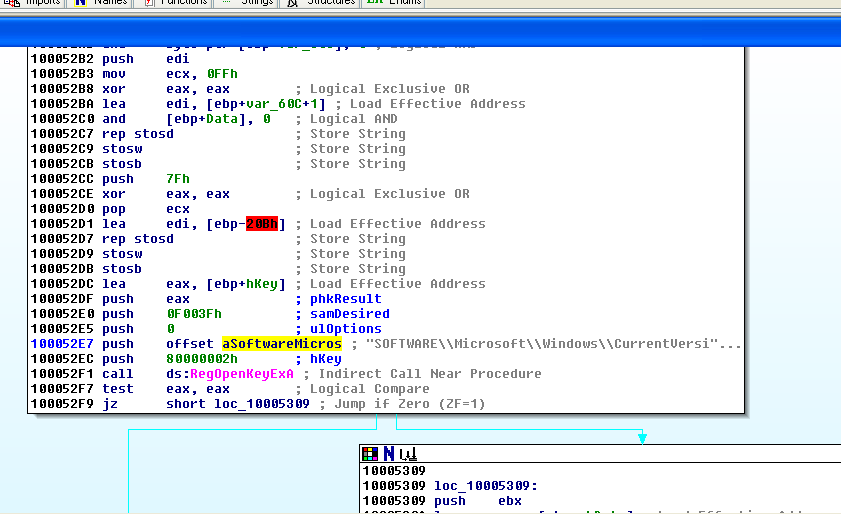


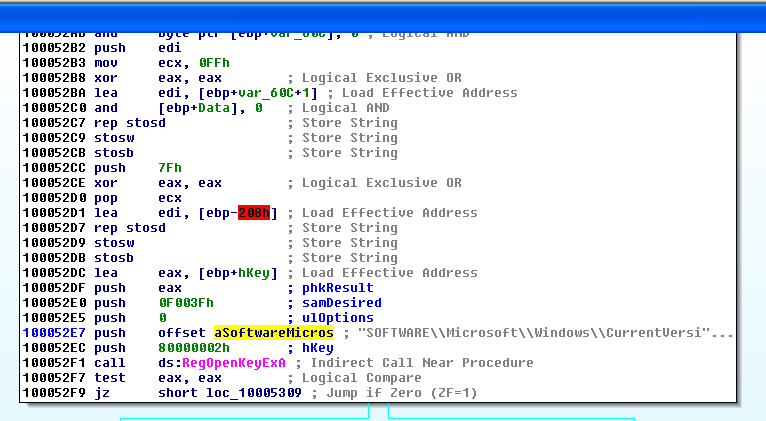
Here we are comparing the versionInformation.dwPlatformId with 2 to check the whether the operating system is Windows NT so that \cmd.exe /c will be pushed otherwise \command.exe /c will be.

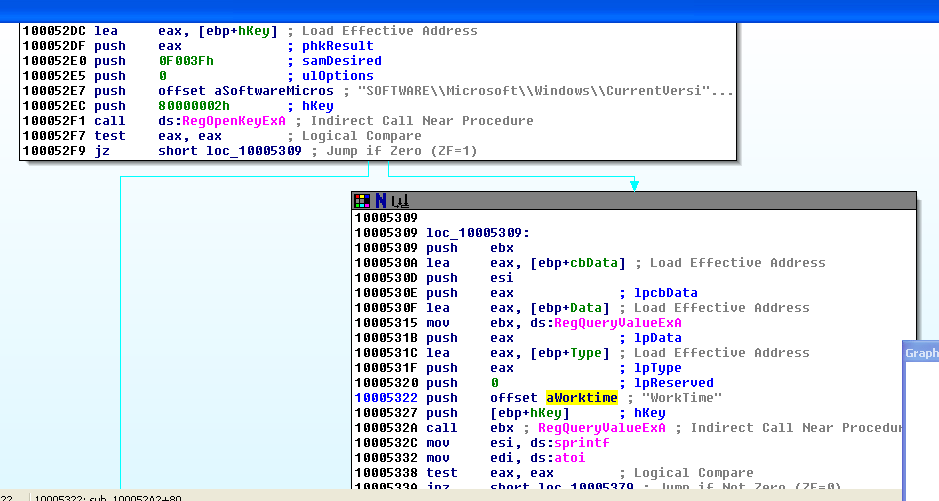
**10-A few hundred lines into the subroutine at 0x1000FF58, a series of comparisons use memcmp to compare strings. What happens if the string comparison to robotwork is successful (when memcmp returns 0)?**

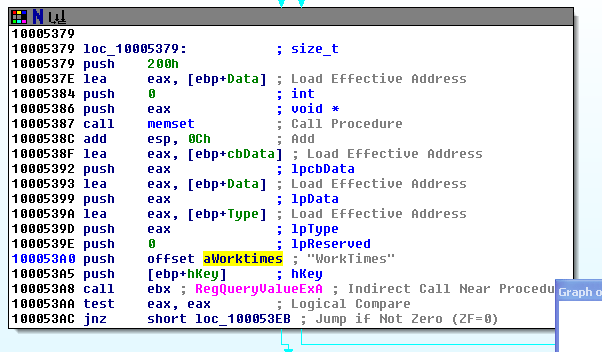
The robotwork do a string comparison using memcmp and checking the result returned in EAX to check whether its zero if the two strings are identical to return 0 or not it will jump to the location loc\_10010468

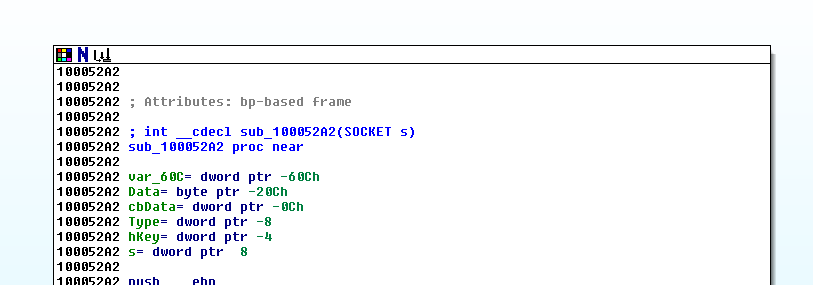
If memcmp returns 0 the jump won’t be achieved and we will follow the red arrow that calls the function located at the sub\_10052A2 which includes registry keys.

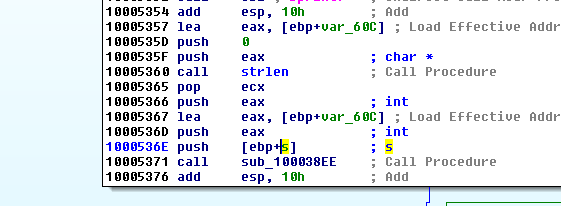


It includes 3 registry keys:

* softwareMicros
* worktime
* worktimes

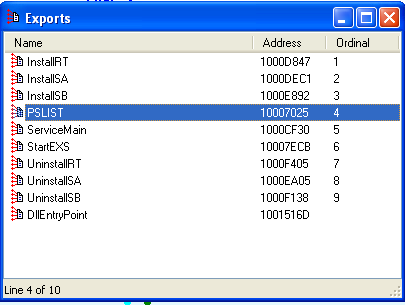


The function is looking for values from worktimes and worktime which was indicated from -RegQueryValueEXA-

At the beginning of the function socket will be stored the value in s parameter to be passed to

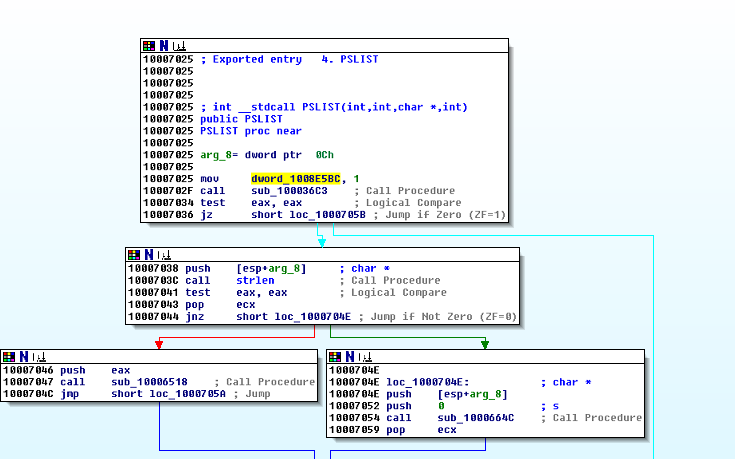
if the comparison returns True for robotwork which means that the strings are identical then the registry keys worktime, worktimes and softwareMicros then will be queried and the values will be passed by the remote shell connection.



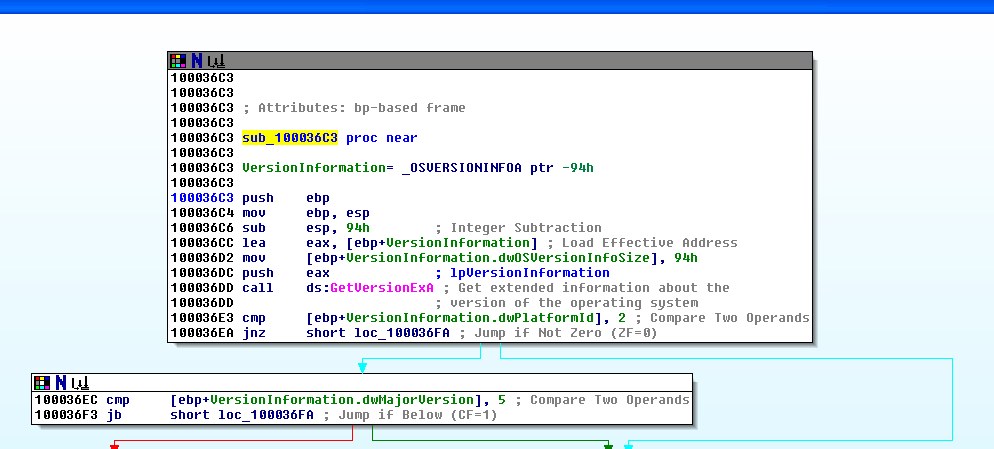
**11-What does the export PSLIST do?**

To open the exports:

* from alt+3
* from windows go to exports.
* Or from the tabs that are already opened from the tool bar.

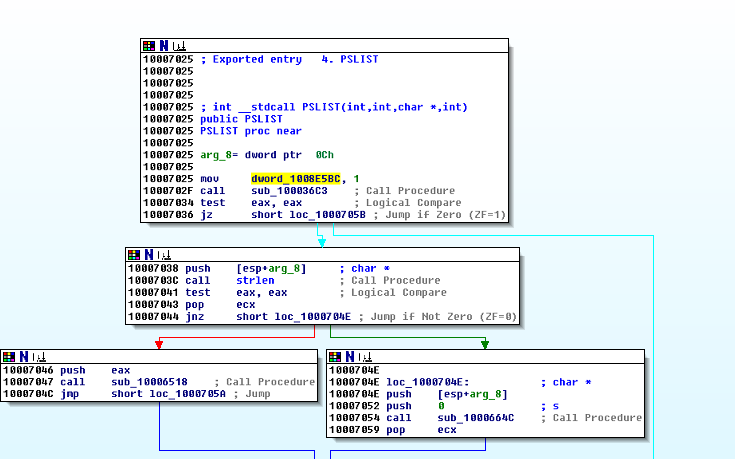




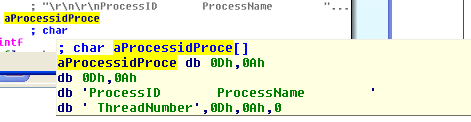
This function compares the OS with 2 to see whether it is an Windows NT adding to that it checks the if the dwMajorVersion is equal to 5



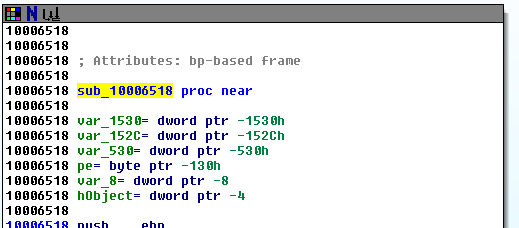
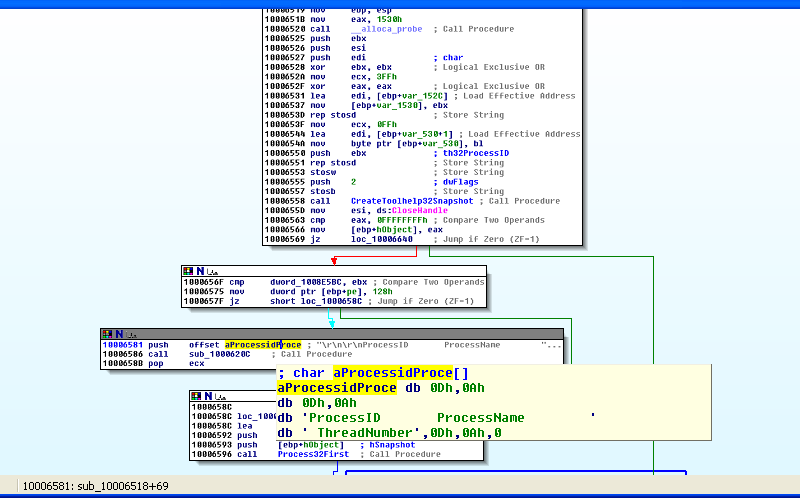
and based on the output we will jump as shown below we will jump to sub\_10006518 or sub\_1000664C



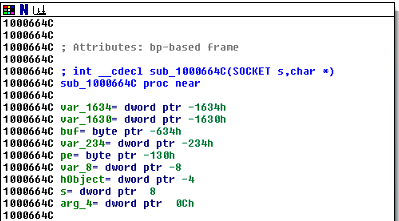
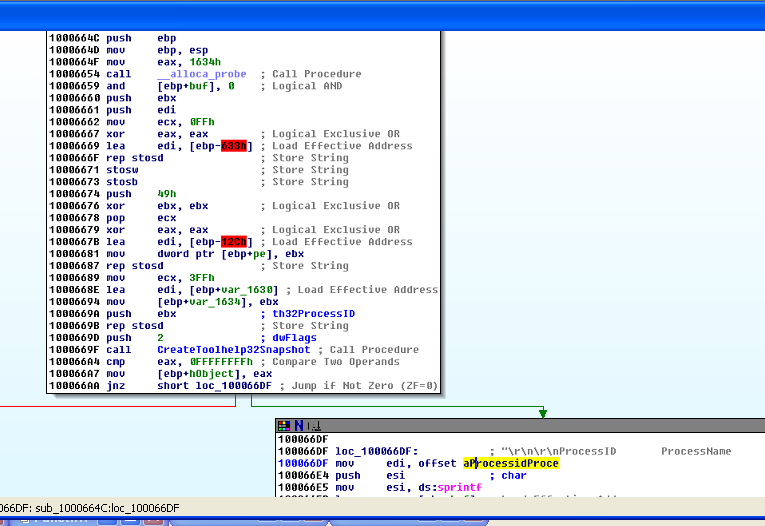
As shown below both of the functions are taking snapshot for specific process and the associated information about and then will execute the appropriate commands to query the following shown information including

* Process ID
* Process Name
* Thread number

But the only difference that the function sub\_100664C it includes the socket s to send the output out to.

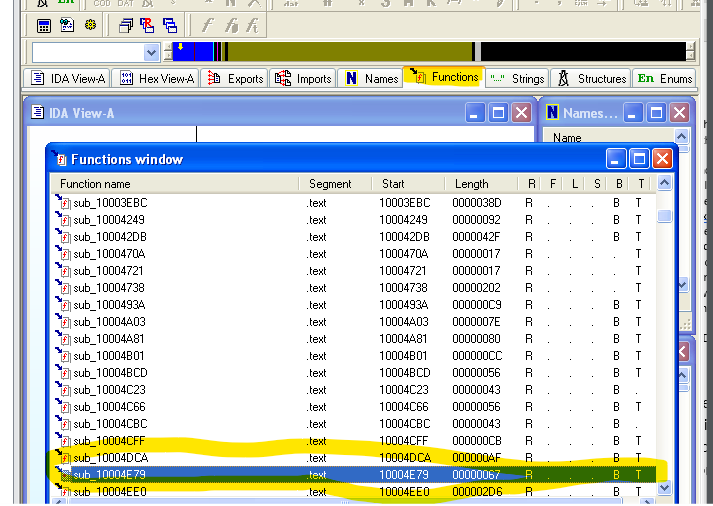
Sub\_10006518

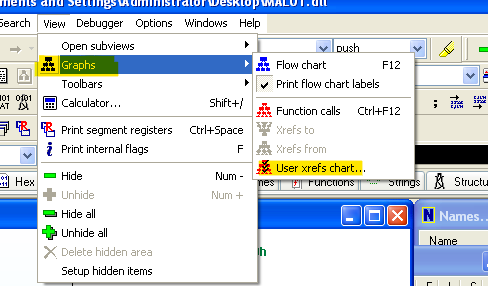


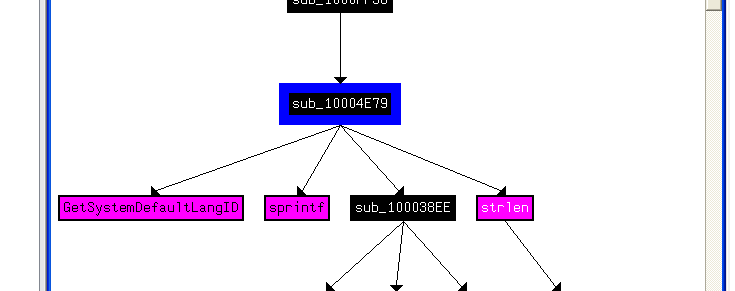
Sub\_1000664C



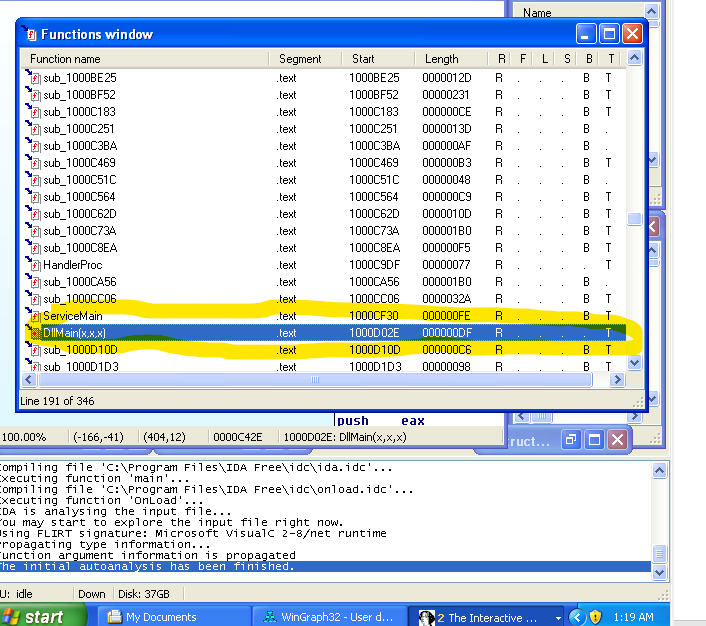
1. **Use the graph mode to graph the cross-references from sub\_10004E79. Which API functions could be called by entering this function? Based on the API functions alone, what could you rename this function?**
2. **How many Windows API functions does DllMain call directly? How many at a depth of 2?**
3. **At 0x10001358, there is a call to Sleep (an API function that takes one parameter containing the number of milliseconds to sleep). Looking backward through the code, how long will the program sleep if this code executes?**
4. **At 0x10001701 is a call to the socket. What are the three parameters?**
5. **Using the MSDN page for socket and the named symbolic constants functionality in IDA Pro, can you make the parameters more meaningful? What are the parameters after you apply changes?**
6. **Search for the usage of the in-instruction (opcode 0xED). This instruction is used with a magic string VMXh to perform VMware detection. Is that in use in this malware? Using the cross-references to the function that executes the in-instruction, is there further evidence of VMware detection?**
7. **Jump your cursor to 0x1001D988. What do you find?**

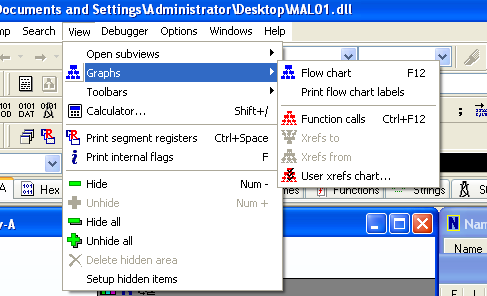
**12.Use the graph mode to graph the cross-references from sub\_10004E79. Which API functions could be called by entering this function? Based on the API functions alone, what could you rename this function?**

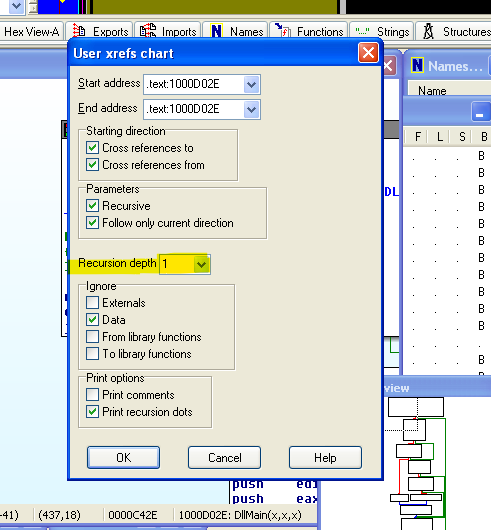


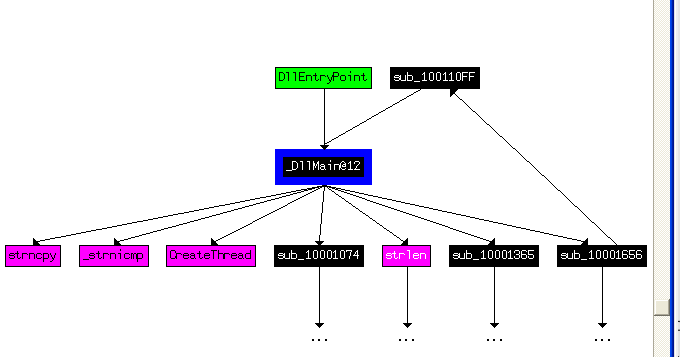


From the function names we can rename this subroutine as ((Find\_System\_Languag)).

**13.How many Windows API functions does DllMain call directly? How many at a depth of 2?**

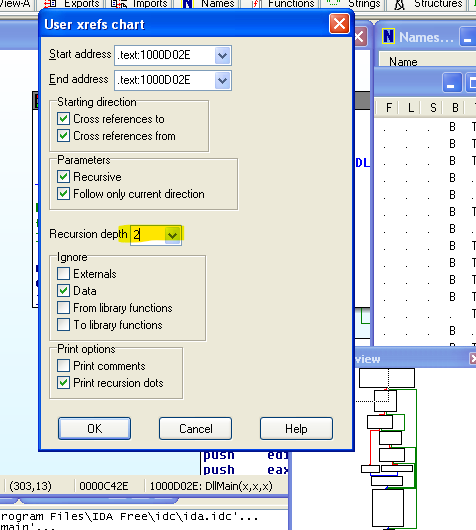
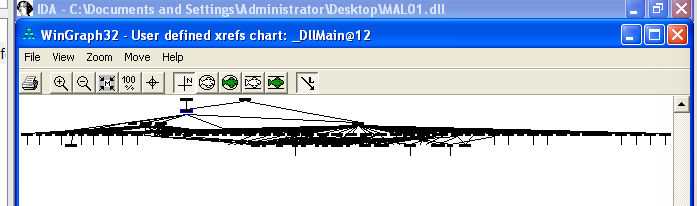






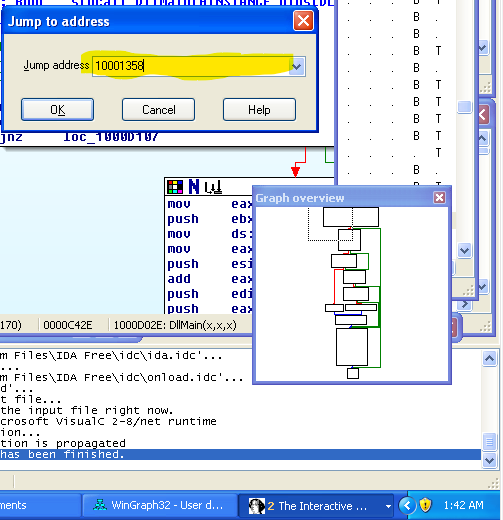
Number of API functions: 4

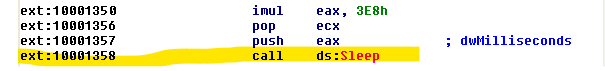
* Strncpy
* \_strnicmp
* CreateThread
* strlen

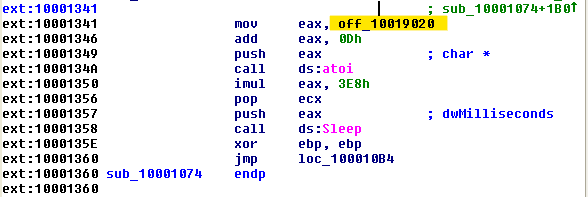


At depth 2 it is not clear & you can zoom in and count and there are too many!

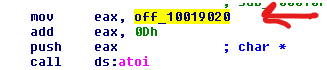
**14. At 0x10001358, there is a call to Sleep (an API function that takes one parameter containing the number of milliseconds to sleep). Looking backward through the code, how long will the program sleep if this code executes?**

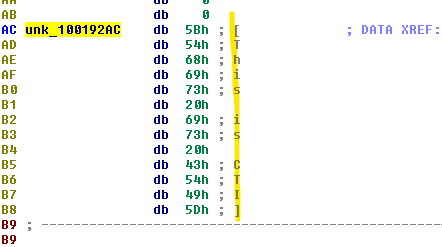
 Here is a sleep function that is called and takes milliseconds and the amount of it will be stored in EAX.





we followed where the initialization of EAX and it started at offset 10001341: 1) click on this offset and then the next one ->



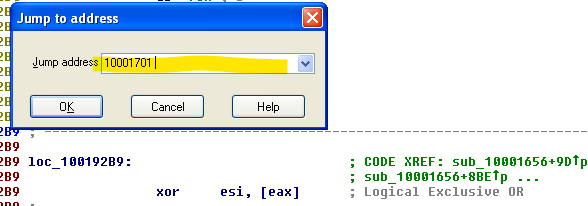


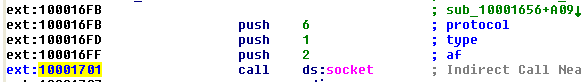
first it added to EAX the string “this is cti” without any number that can show me how many milliseconds it will sleep.

But if there was then it will add to the EAX 0DH 🡪 13 bytes further …. Then atoi function converts the string to an integer and then multiply the value with 1000 🡪 3E8H that will be 1000 millisecond {{I believe

the program will not sleep bcz no value is given}}.

**15,16- At 0x10001701 is a call to the socket. What are the three parameters and using the MSDN page for socket and the named symbolic constants functionality in IDA Pro, can you make the parameters more meaningful? What are the parameters after you apply changes?**





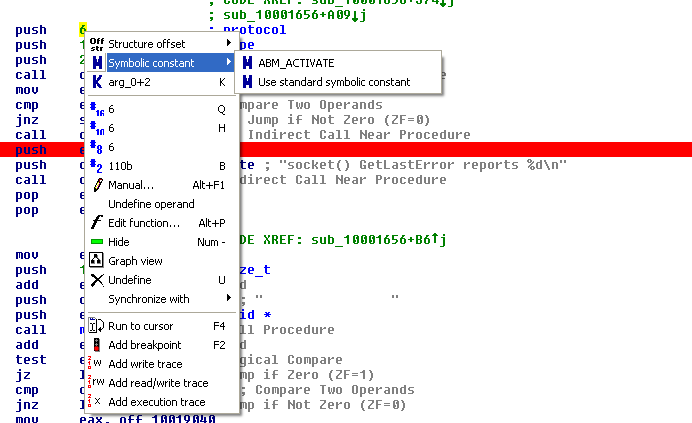
The 3 parameters were pushed on a stack!

2Af🡪address family specification that means IF\_INET

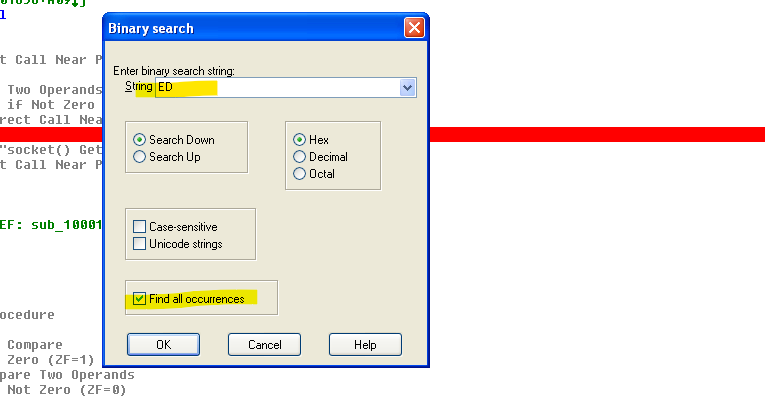
1type🡪 Type of socket (sock\_stream)

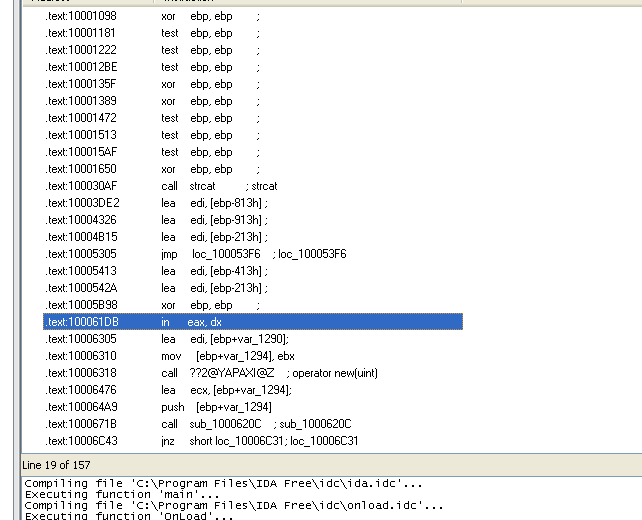
6protocol🡪 iproto\_TCP 2🡪 1 🡪 6

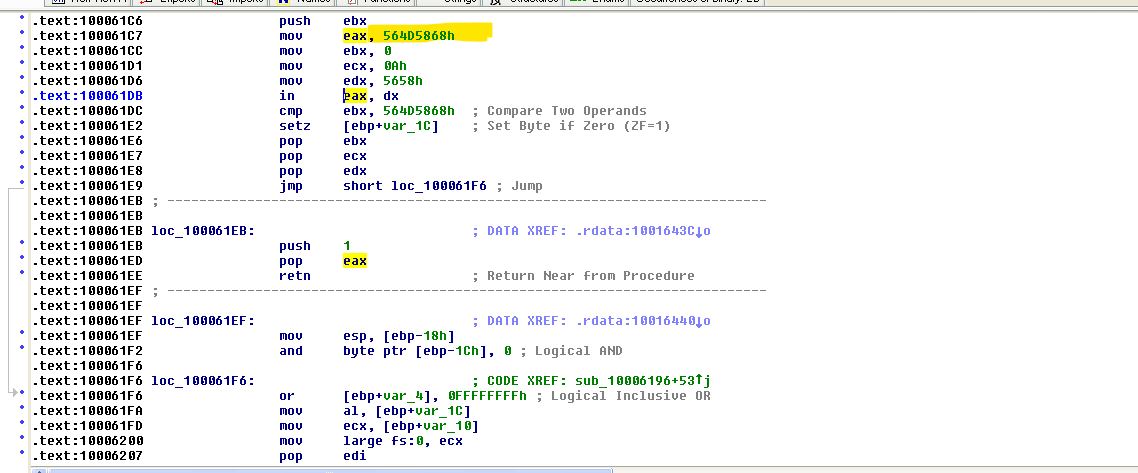
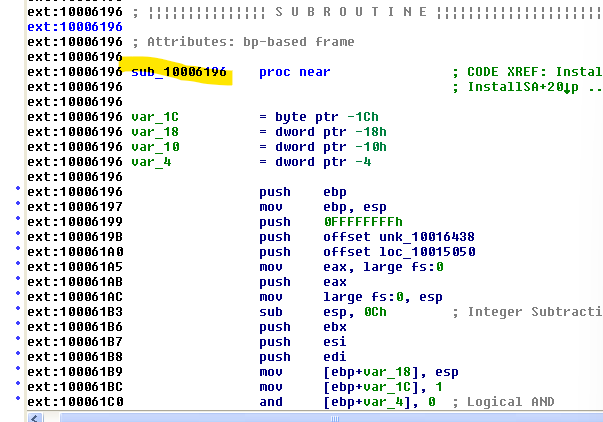
we can make them more meaningful by renaming them by right clicking on the parameter then choosing symbolic constants (standard) and it will turn like the meaning we wrote up based on what you choose!!



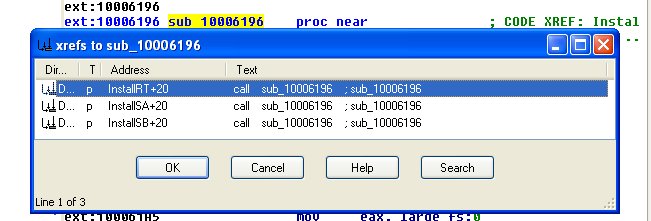
**17. Search for the usage of the in-instruction (opcode 0xED). This instruction is used with a magic string VMXh to perform VMware detection. Is that in use in this malware? Using the cross-references to the function that executes the in-instruction, is there further evidence of VMware detection?**

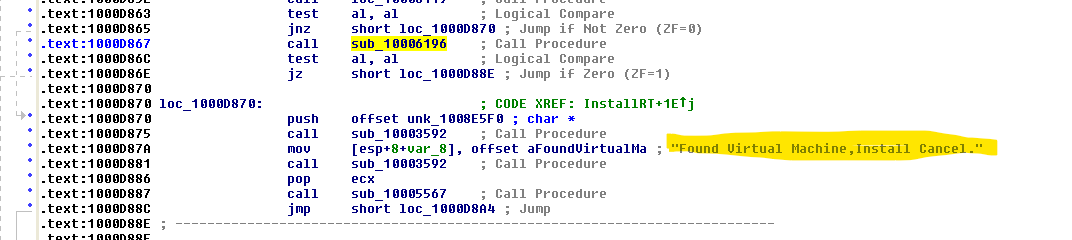


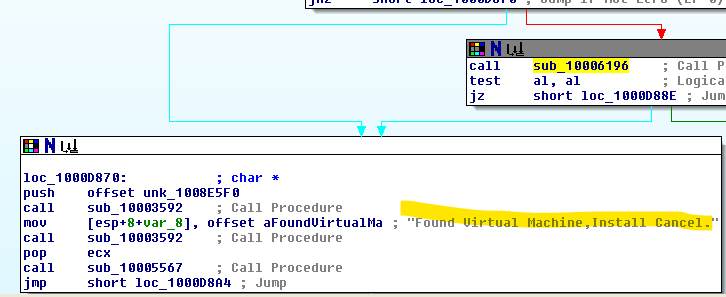
we search for ((ED🡪in)) then we click on it

there are a hex code (564D5868H) stands for 🡪 “VMXh”then we return to what called this function!

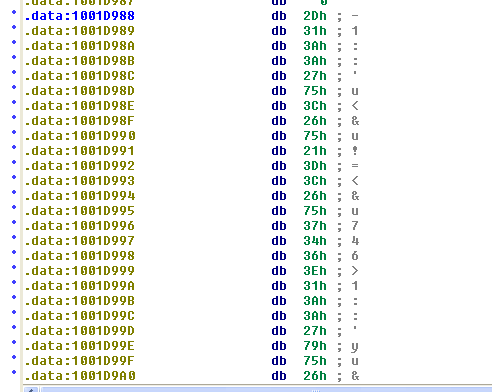
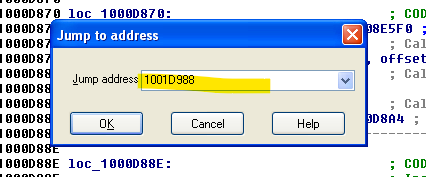
Ctrl+x then pressed on OK.





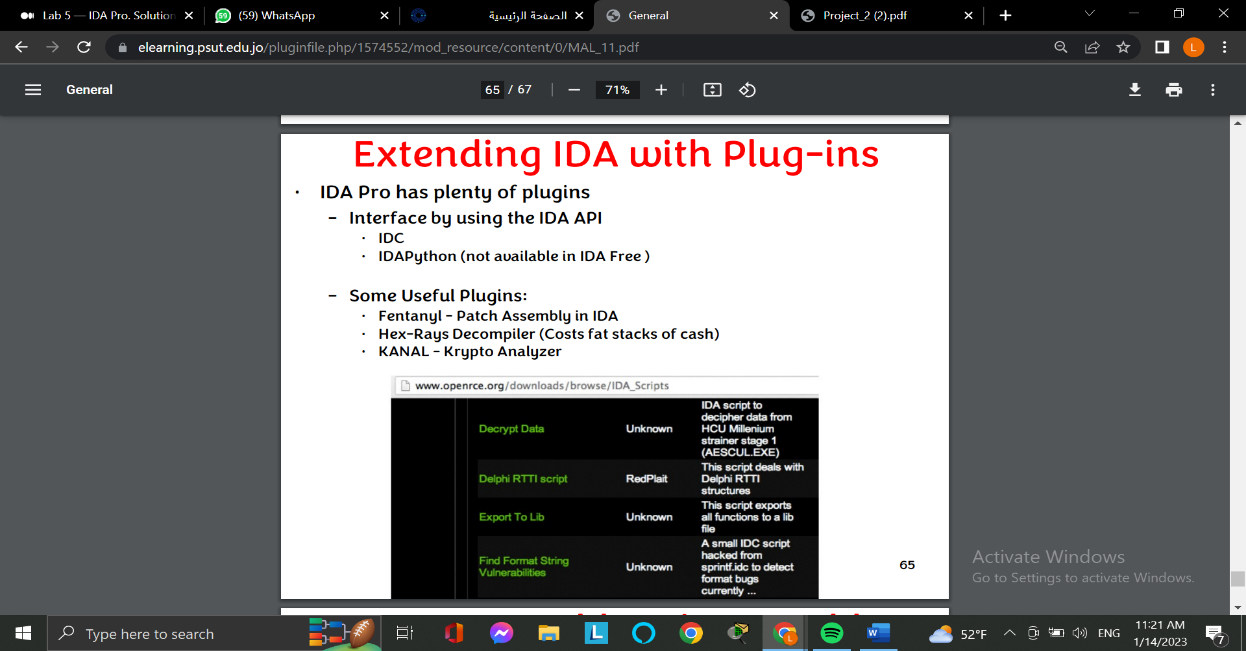


Malware not installed(cancelled!!).

18. Jump your cursor to 0x1001D988. What do you find?

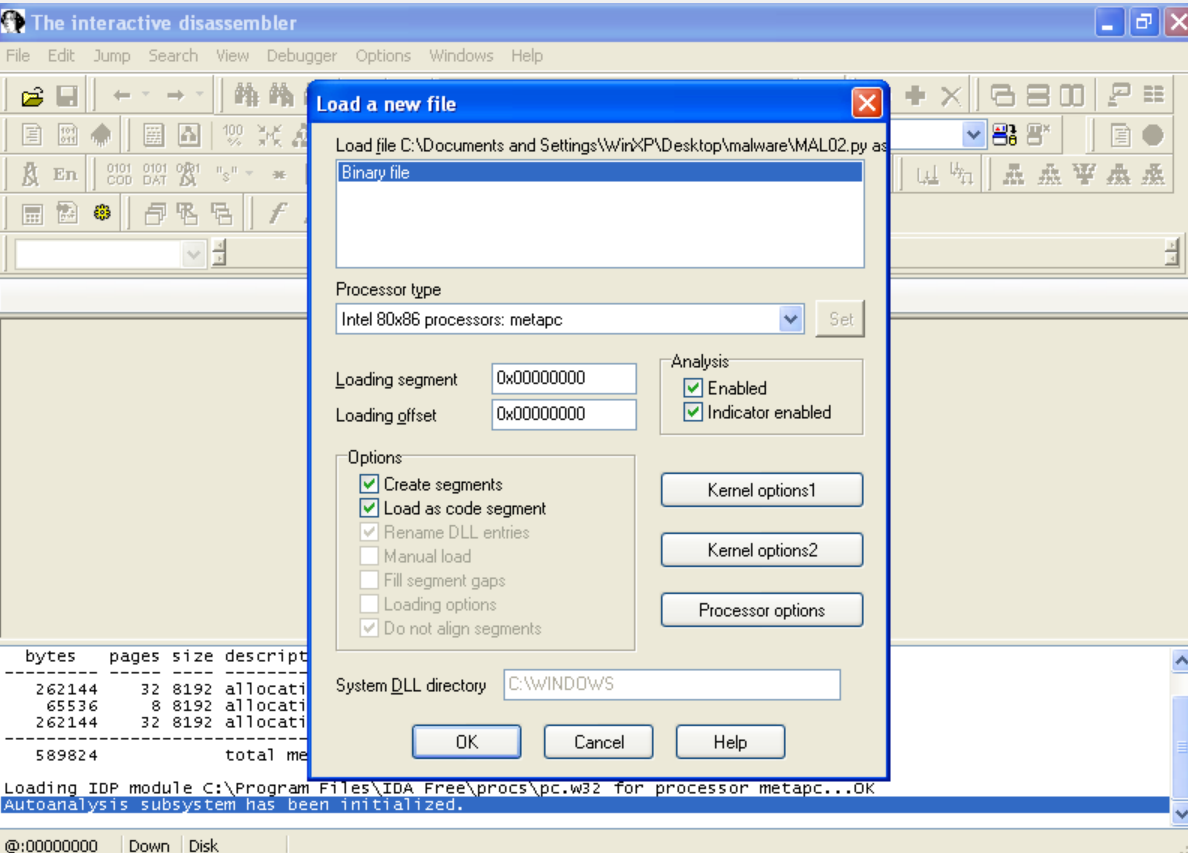
I find garbage strings / values!

**Part #2 🡪 Mal02.py**

1. Download and install the IDA Python plug-in. If you have the IDA Python plug-in installed (included with the commercial version of IDA Pro).
2. Run MAL02.py, an IDA Pro Python script, (Make sure the cursor is at 0x1001D988.) What happens after you run the script?
3. With the cursor in the same location, how do you turn this data into a single ASCII string?
4. Open the script with a text editor. How does it work?

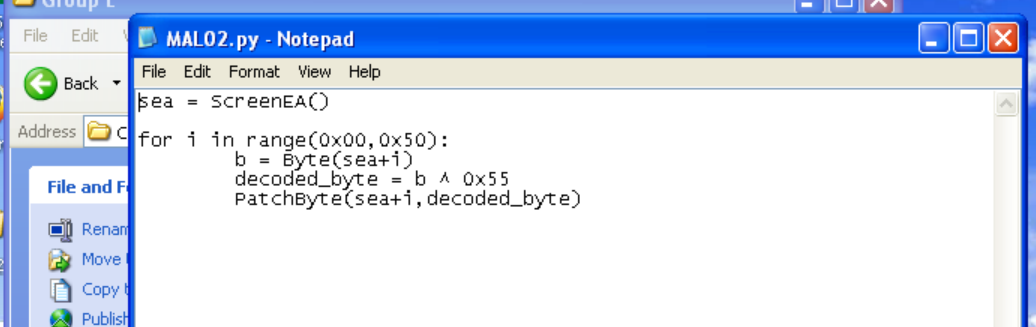
**1-Download and install the IDA Python plug-in. If you have the IDA Python plug-in installed (included with the commercial version of IDA Pro).**

Firstly, IDApython not available in the free version. We couldn’t download and install the IDA python plug-in because we are working on the free IDA version that does not support any plugins, therefore we couldn’t even open the mal2.py file properly in the IDA, because IDA couldn’t recognize the file type.



**So, we had to work with the script open on a notepad.**

**2-Run MAL02.py, an IDA Pro Python script, (Make sure the cursor is at 0x1001D988.) What happens after you run the script?**



As we can see, the provided python script decodes the text by performing an XOR operation on the 0x55 with the 0x50 bytes from where the current cursor is, decrypting the string and making it readable.

**3-With the cursor in the same location, how do you turn this data into a single ASCII string?**

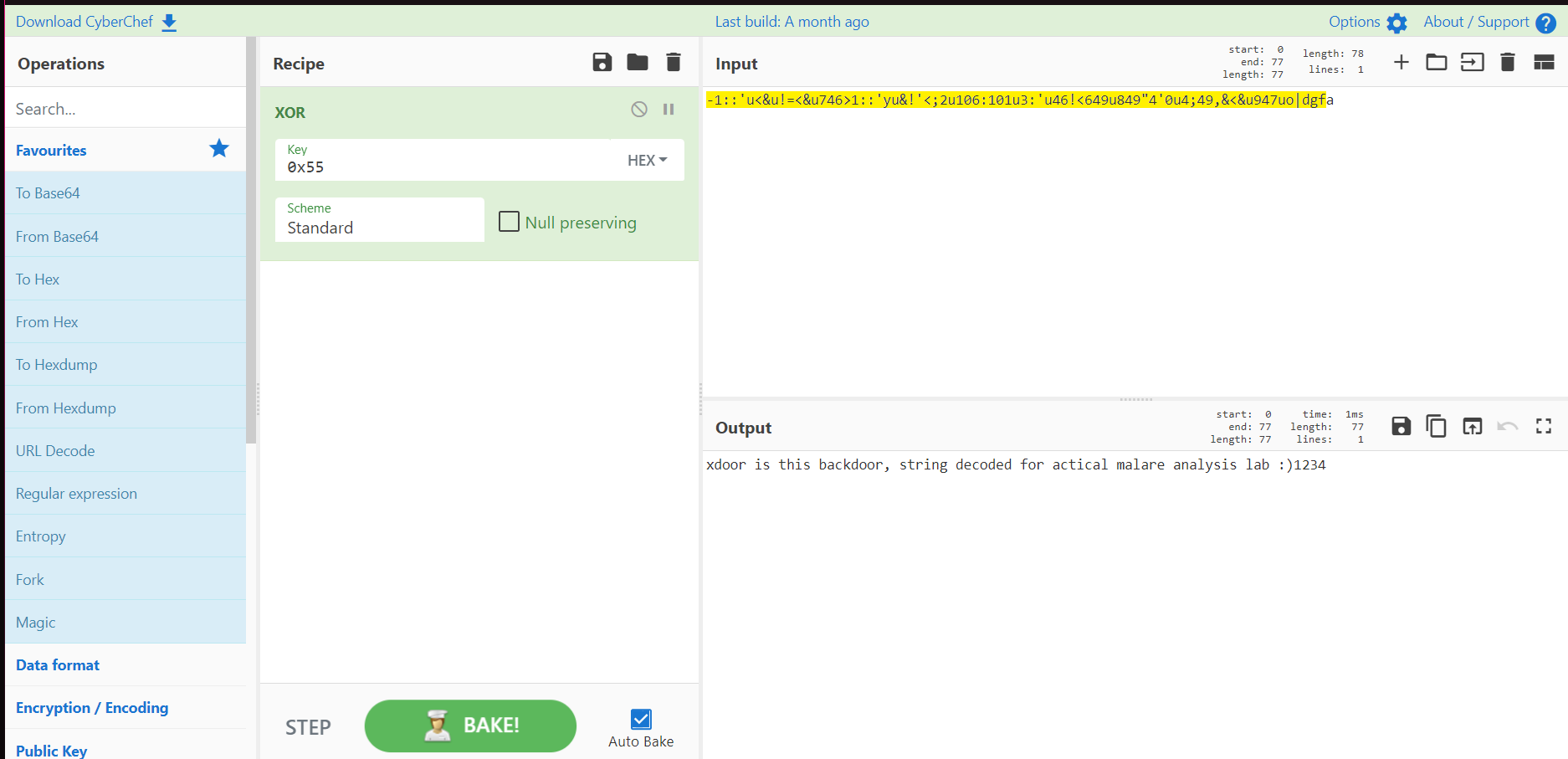
By pressing A in the IDA, we concatenate all the ASCCI characters in a row until it finds an empty space, and we got this encrypted string:

(-1::'u<&u!=<&u746>1::'yu&!'<;2u106:101u3:'u46!<649u849”4'0u4;49,&<&u947uo|dgfa)

(If we had the python plugin the string would’ve already been decrypted)

Because we didn’t have the IDA Pro, we used the online tool CyberChef to XOR the encrypted string with 0x55 (like the python script we have does) and got this result:

xdoor is this backdoor, string decoded for actical malware analysis lab :)1234

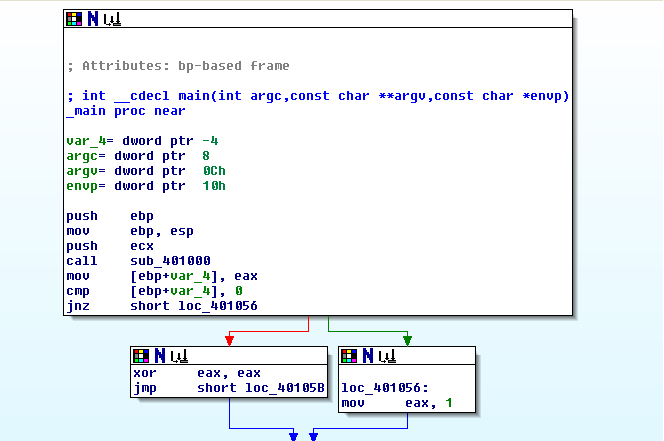


**That said, the malware is obviously a backdoor**

**4-Open the script with a text editor. How does it work?**

The python script we have uses the function ‘ScreenEA’ that returns the current cursor position in the IDA, which is 0x1001D988 in our case. Then the script iterates using a for loop over the next 80 bytes (0x50 = 80 in decimal), and XORs every byte with 0x55, then uses the function PatchByte to save the decrypted bytes.

**Part #3 🡪 Mal03.exe**

1. What is the major code construct found in the only subroutine called by main?
2. What is the subroutine located at 0x40105F?
3. What is the purpose of this program?
4. **What is the major code construct found in the only subroutine called by main?**

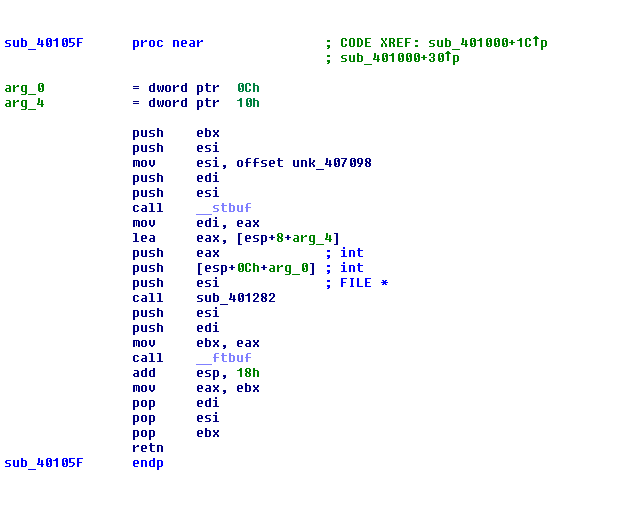
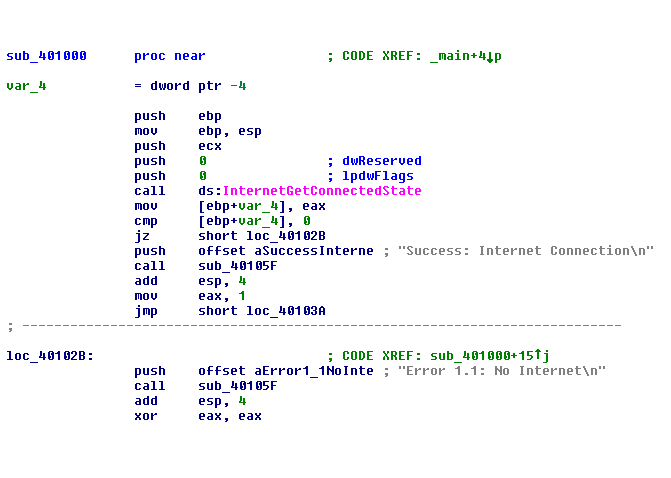
The code construct is an if statement with only two possibilities as shown in the image below.

**2-What is the subroutine located at 0x40105F?**

The subroutine at 0x401050 is called twice in the sub\_401000, once after the JZ command and once after the location that the execution will go to if JZ is true, and for each time it is called there is a push statement just before the call, and both push statements push strings ending with /n “new line”

And that indicates that the subroutine mainly is to print a string based on the result of the if statement.

-The two images below show the subroutine itself and where it is called with the details mentioned above:



**Q3.) What is the purpose of this program?**

What this program mainly does is that it checks the internet connection through the instruction in the location 0x401008 which calls the function InternetGetConnectedState and based on the result of this function the if statements either prints a string that indicates the success of the internet connection or a string that indicates connection failure.

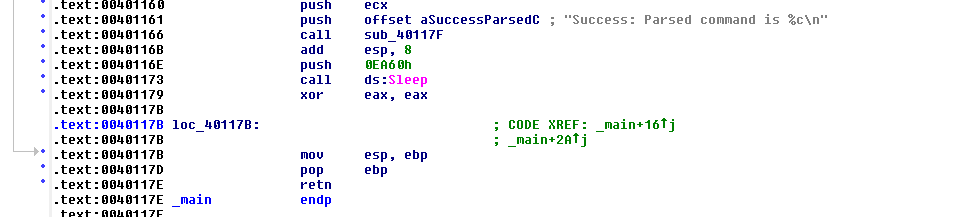
**Part #4 🡪 Mal04.exe**

1. What operation does the first subroutine called by the main perform?
2. What is the subroutine located at 0x40117F?
3. What does the second subroutine call by the main do?
4. What type of code construct is used in this subroutine?
5. Are there any network-based indicators for this program?
6. What is the purpose of this malware?

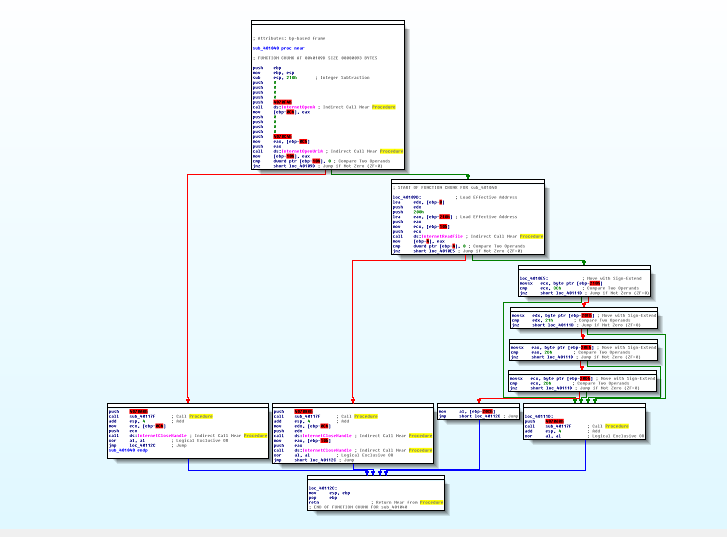
**1-What operation does the first subroutine called by the main perform?**

**The first subroutine call was “*sub\_401000. It*** checks the connected state to internet (internetGetConnectedState).

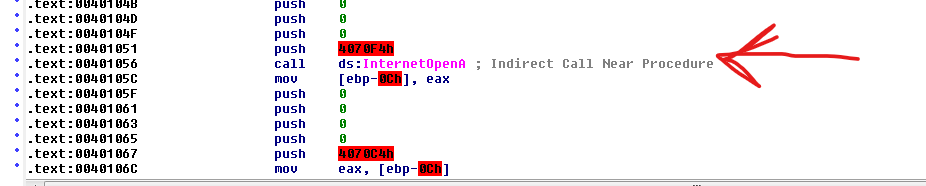
**2-What is the subroutine located at 0x40117F?**

since there is something has been pushed before the **“sub 0x40117F”**. So, we can tell that this function is used for printing out after execution.

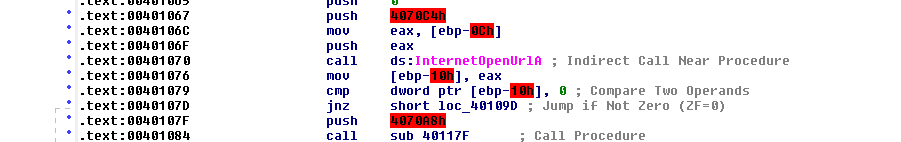
**3-What does the second subroutine call by the main do?**

the second subroutine call was **“0x401040”**

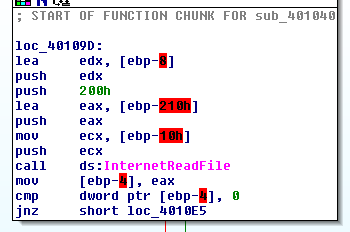
What does it do

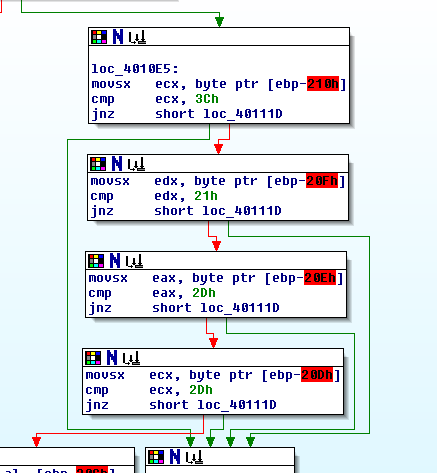
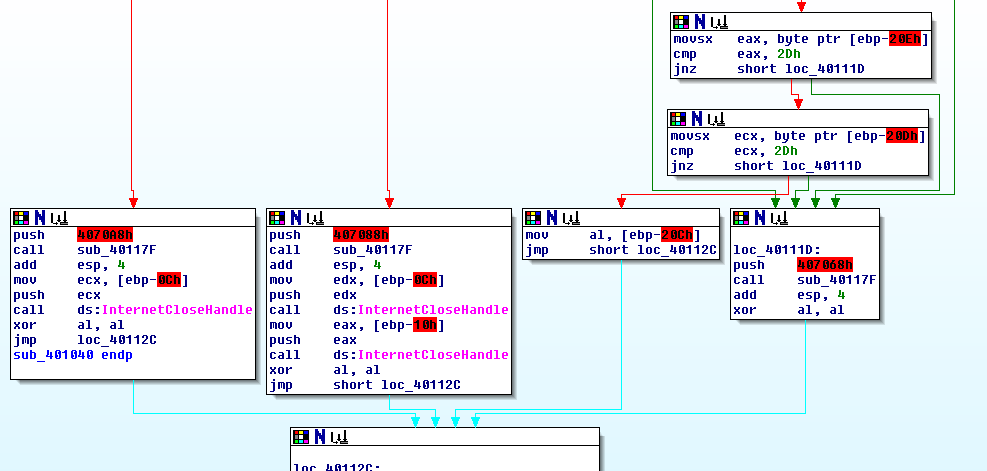
At first there is a call to **InternetOpenA** which is used to open a specific web browser. 

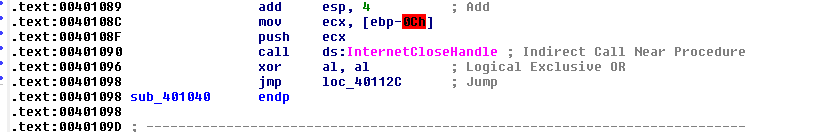
Then **InternetOpenUrlA** is called to open a specific URL in the browser.



After that, **InternetReadFile** is called it do comparisons between the offsets of the variables and the bytes to check if whether it’s reading an HTML commentary or not





At the end**, InternetCloseHandle** is called

**4-What type of code construct is used in this subroutine?**

the code construct that is used is an Array to collect content retrieved from the Web Page.

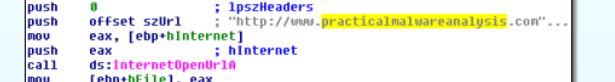
**5- Are there any network-based indicators for this program?**

Yes, there was a request that used user\_agent :”Internet Explorer 7.5/pma”.



Also, there was a trial to connect to internet using a specific URL:

<http://www.practicalmalwareanalysis.com/cc.htm>



**6- What is the purpose of this malware?**

The purpose is to get data from Web page. The data is found hidden in a textbox that are used for commentary (adding comments).

References used in Mal01:

https://www.hex-rays.com/products/ida/support/freefiles/IDA\_Pro\_Shortcuts.pdf

https://stackoverflow.com/questions/15097970/const-char-name-vs-const-char-name

https://windowstect.com/what-does-cmd-c-means-in-windows-ms-dos/

https://learn.microsoft.com/en-us/windows/win32/api/sysinfoapi/nf-sysinfoapi-getsystemdirectorya

https://learn.microsoft.com/en-us/dotnet/api/system.platformid?view=net-6.0

https://learn.microsoft.com/en-us/windows/win32/api/winnt/ns-winnt-osversioninfoexa#remarks