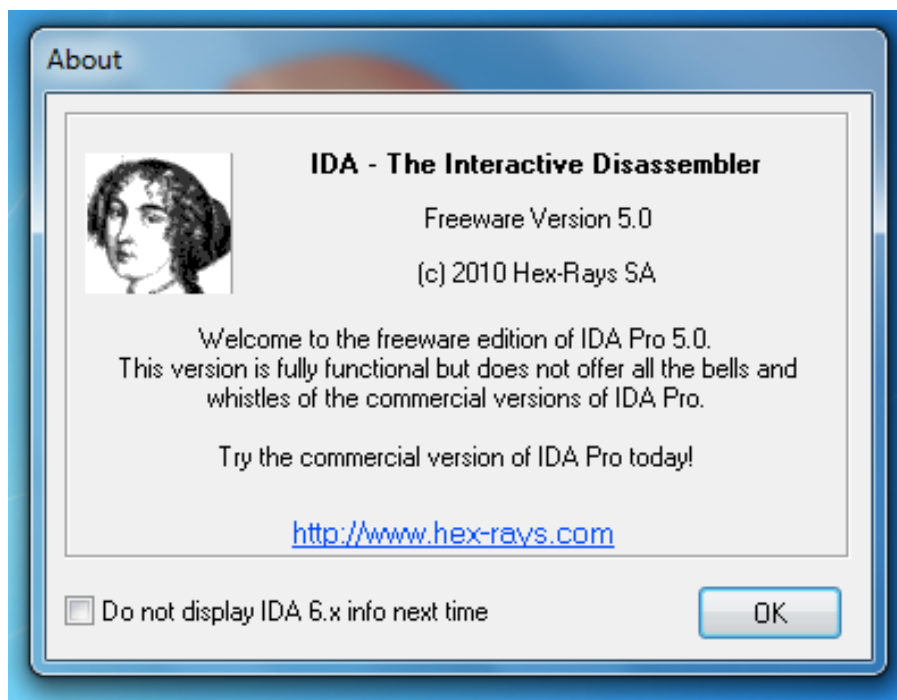


Static Binary Analysis

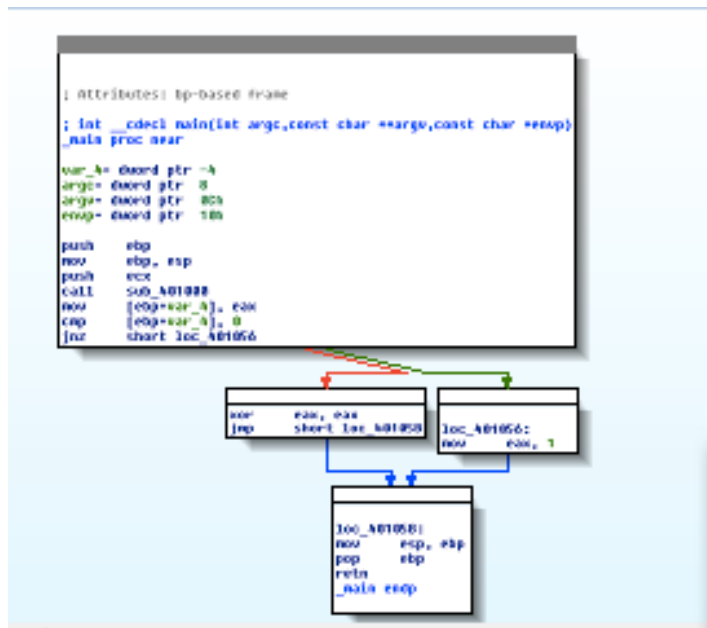


IDA Pro Versions

- Full-featured pay version
- Old free version
 - Both support x86
 - Pay version supports x64 and other processors, such as cell phone processors
- Both have code signatures for common library code in FLIRT (Fast Library identification and Recognition Technology)

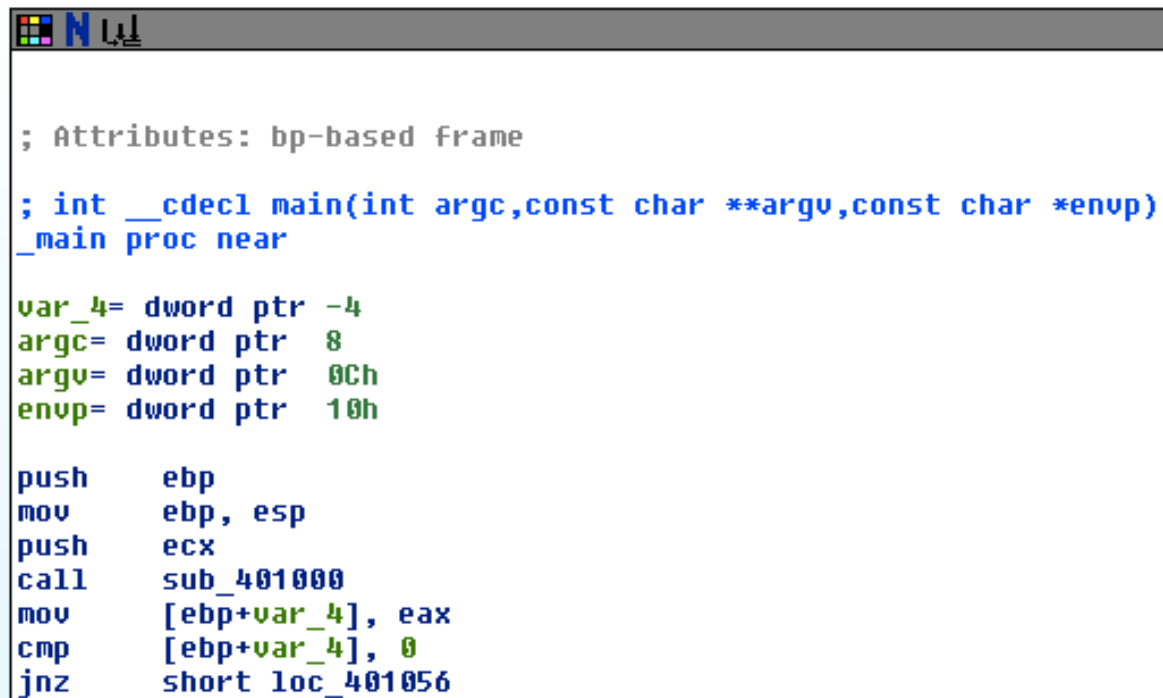
Graph and Text Mode

- Spacebar switches mode



```
IDA View-A
.text:00401040
.text:00401040 ; Attributes: bp-based frame
.text:00401040
.text:00401040 ; int __cdecl main(int argc,const char **argv,const char *envp)
.text:00401040 _main proc near ; CODE XREF: start+AF↓p
.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
```

Default Graph Mode Display



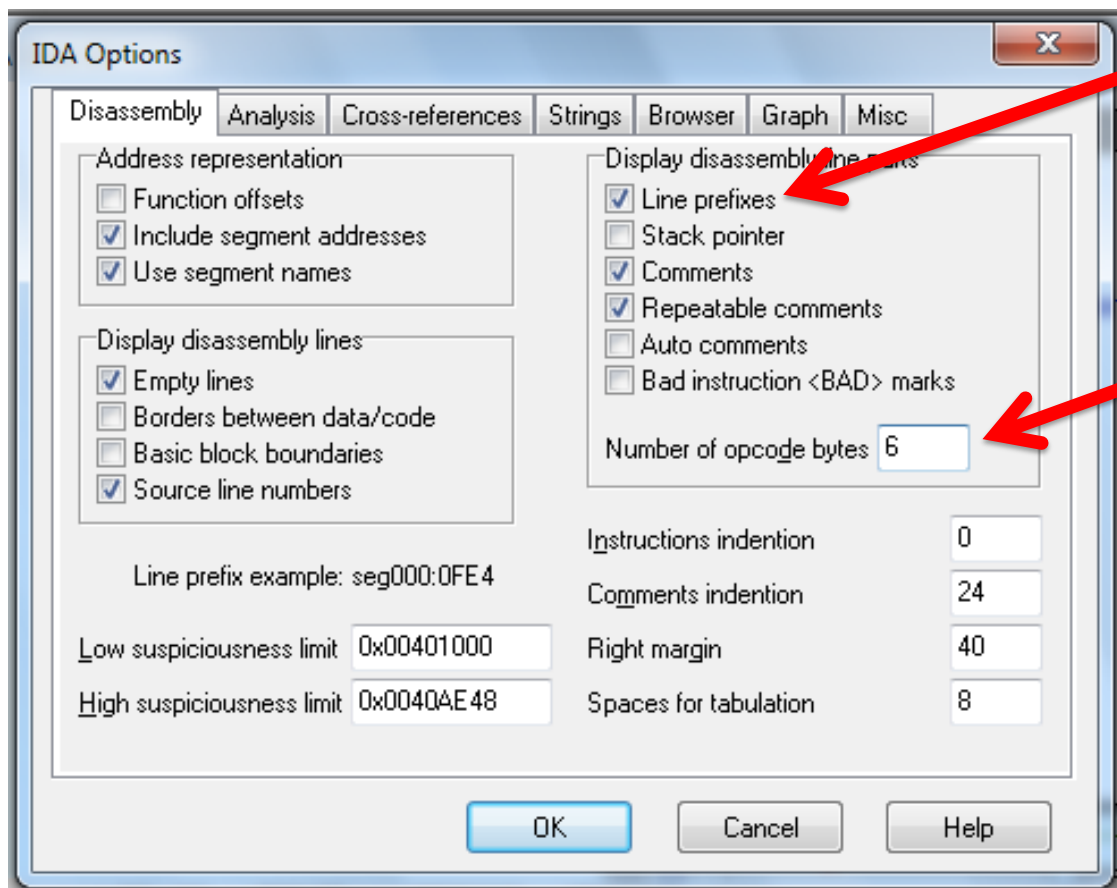
```
; Attributes: bp-based frame

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

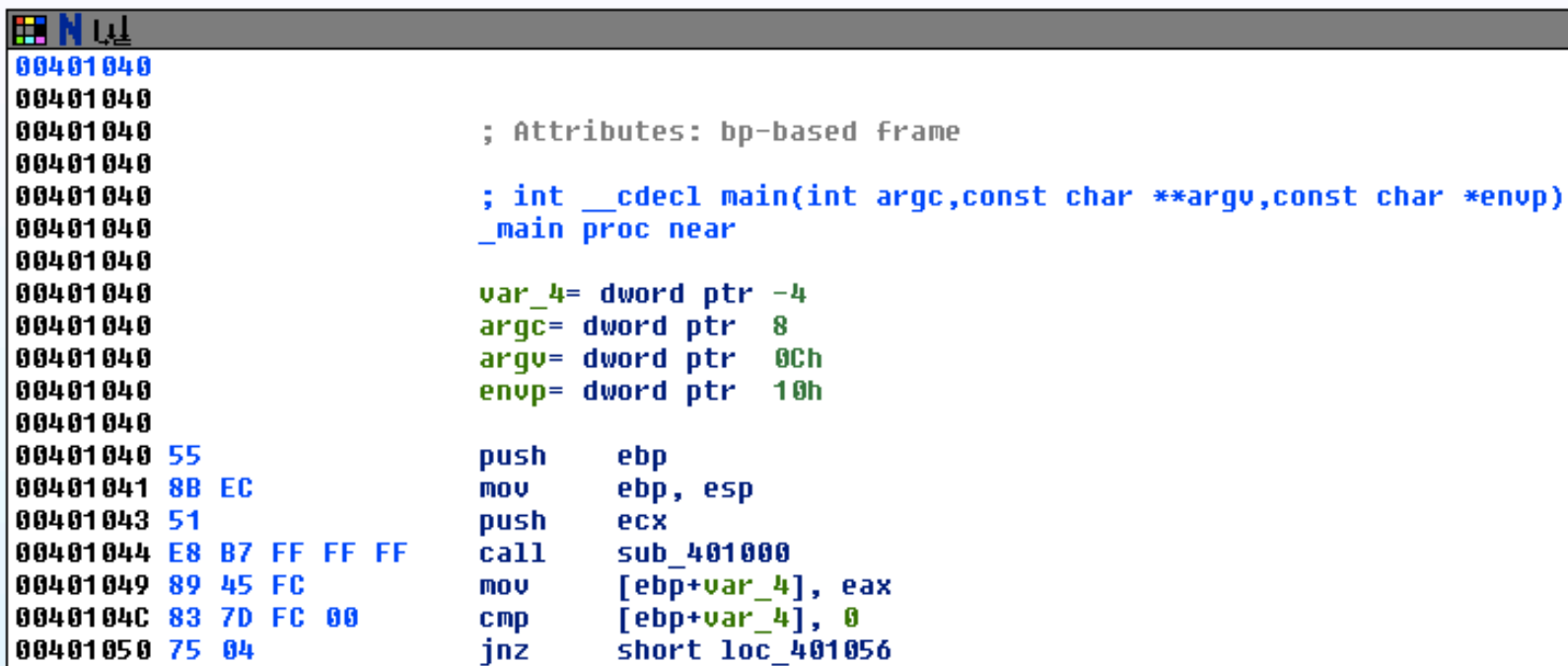
var_4= dword ptr -4
argc= dword ptr 8
argv= dword ptr 0Ch
envp= dword ptr 10h

push    ebp
mov     ebp, esp
push    ecx
call    sub_401000
mov     [ebp+var_4], eax
cmp     [ebp+var_4], 0
jnz     short loc_401056
```

Options, General



Better Graph Mode View



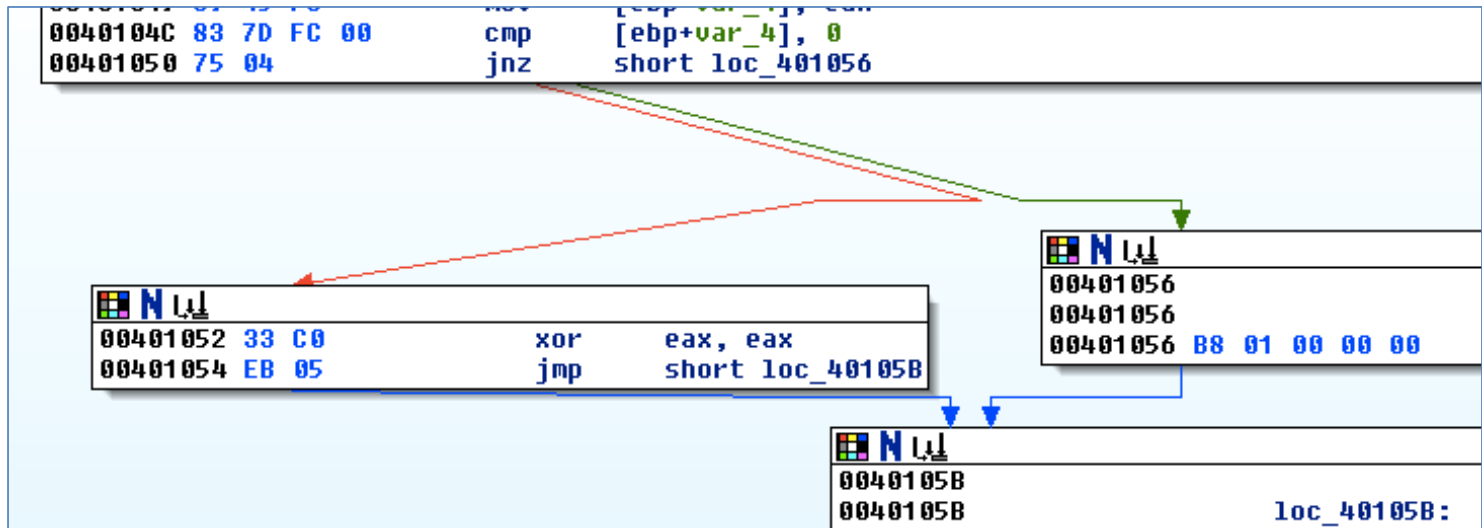
The screenshot shows a debugger window with a graph mode view of assembly code. The window has a title bar with a logo and the text "N". The assembly code is displayed in a list view with addresses on the left and instructions on the right. The instructions are color-coded: blue for control flow and data movement, green for variable declarations, and black for push/pop and call instructions. The code is as follows:

```
00401040  
00401040  
00401040 ; Attributes: bp-based frame  
00401040  
00401040 ; int __cdecl main(int argc,const char **argv,const char *envp)  
00401040 _main proc near  
00401040  
00401040 var_4= dword ptr -4  
00401040 argc= dword ptr 8  
00401040 argv= dword ptr 0Ch  
00401040 envp= dword ptr 10h  
00401040  
00401040 55 push ebp  
00401041 8B EC mov ebp, esp  
00401043 51 push ecx  
00401044 E8 B7 FF FF FF call sub_401000  
00401049 89 45 FC mov [ebp+var_4], eax  
0040104C 83 7D FC 00 cmp [ebp+var_4], 0  
00401050 75 04 jnz short loc_401056
```

Arrows

- Colors
 - Red Conditional jump not taken
 - Green Conditional jump taken
 - Blue Unconditional jump
- Direction
 - Up Loop

Arrow Color Example



Highlighting

- Highlighting text in graph mode highlights every instance of that text

```

00401040
00401040
00401040      ; Attributes: bp-based frame
00401040
00401040      ; int __cdecl main(int argc,const char **argv,const char *envp)
00401040      _main proc near
00401040
00401040      var_4= dword ptr -4
00401040      argc= dword ptr 8
00401040      argv= dword ptr 0Ch
00401040      envp= dword ptr 10h
00401040
00401040 55      push    ebp
00401041 8B EC   mov     ebp, esp
00401043 51      push    ecx
00401044 E8 B7 FF FF FF call    sub_401000
00401049 89 45 FC mov     [ebp+var_4], eax
0040104C 83 7D FC 00 cmp     [ebp+var_4], 0
00401050 75 04   jnz     short loc_401056

```

Text Mode

Arrows

Solid = Unconditional

Dashed = Conditional

Up = Loop

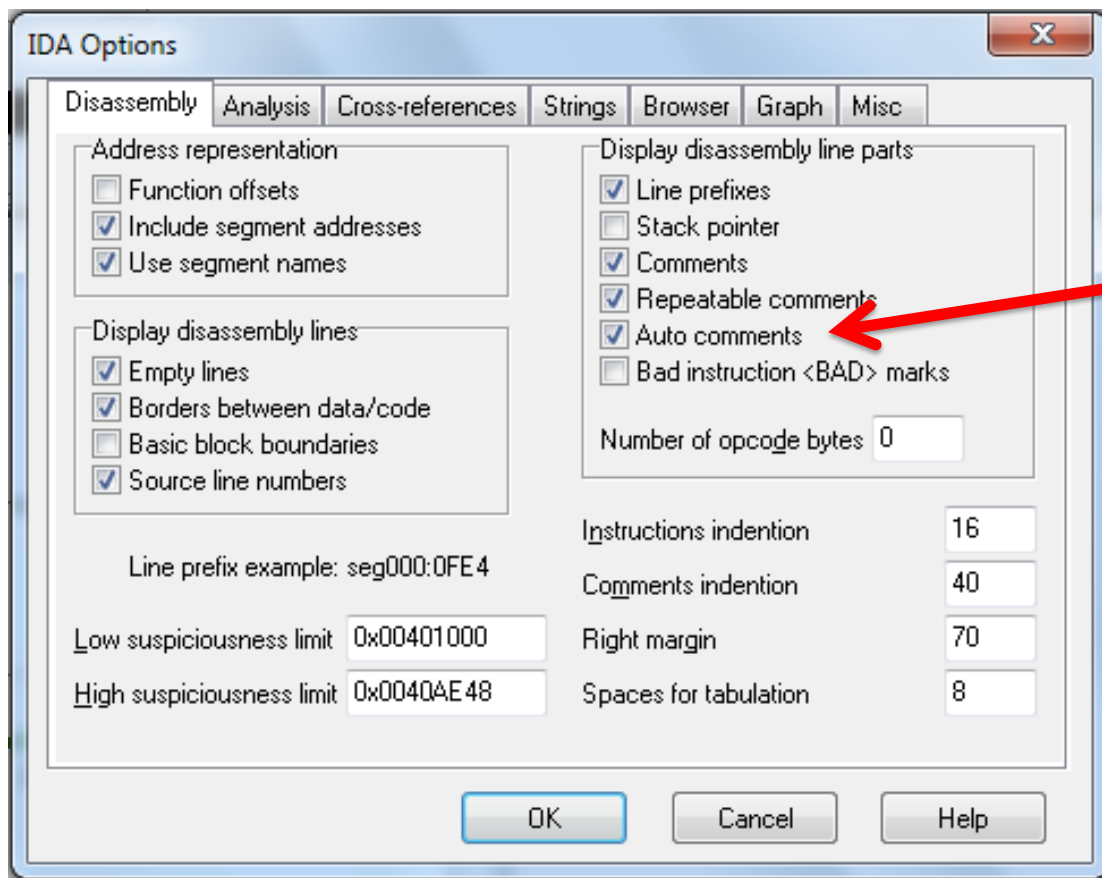
Section

Address

Comment
Generated by
IDA Pro

```
.text:00401015      jz      short loc_40102B
.text:00401017      push     offset aSuccessInterne ; "Success: Internet Connection\n"
.text:0040101C      call    sub_40105F
.text:00401021      add     esp, 4
.text:00401024      mov     eax, 1
.text:00401029      jmp     short loc_40103A
.text:0040102B      ; -----
.text:0040102B      loc_40102B:      ; CODE XREF: sub_401000+15↑j
.text:0040102B      push     offset aError1_1NoInte ; "Error 1.1: No Internet\n"
.text:0040102B      call    sub_40105F
.text:00401030      add     esp, 4
.text:00401035      xor     eax, eax
.text:00401038      loc_40103A:      ; CODE XREF: sub_401000+29↑j
.text:0040103A      mov     esp, ebp
.text:0040103C      pop     ebp
```

Options, General



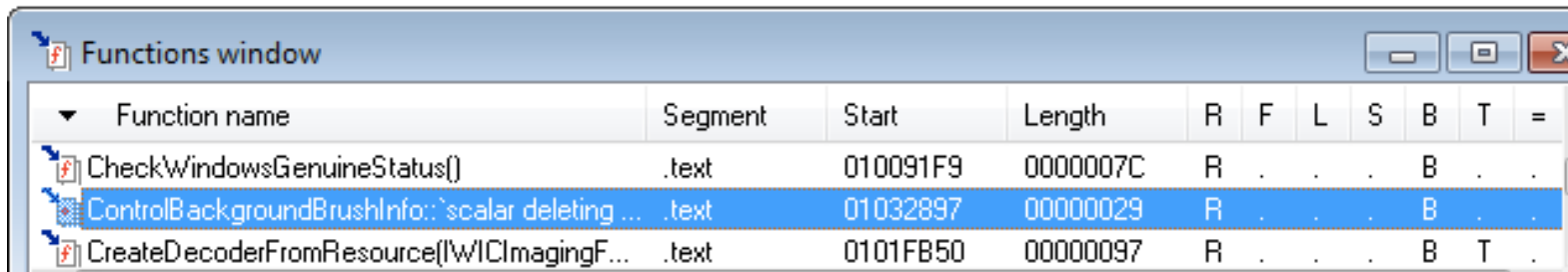
Adds Comments to Each Instruction

```
.text:00401015      jz      short loc_40102B ; Jump if Zero (ZF=1)
.text:00401017      push     offset aSuccessInterne ; "Success: Internet Connection\n"
.text:0040101C      call    sub_40105F          ; Call Procedure
.text:00401021      add     esp, 4              ; Add
.text:00401024      mov     eax, 1              ;
.text:00401029      jmp     short loc_40103A ; Jump
.text:0040102B ; -----
.text:0040102B      loc_40102B: ; CODE XREF: sub_401000+15↑j
.text:0040102B      push     offset aError1_1NoInte ; "Error 1.1: No Internet\n"
.text:0040102B      call    sub_40105F          ; Call Procedure
.text:00401030      add     esp, 4              ; Add
.text:00401035      xor     eax, eax            ; Logical Exclusive OR
.text:00401038      |
.text:0040103A      loc_40103A: ; CODE XREF: sub_401000+29↑j
.text:0040103A      mov     esp, ebp
.text:0040103C      pop     ebp
```

Useful Windows for Analysis

Functions

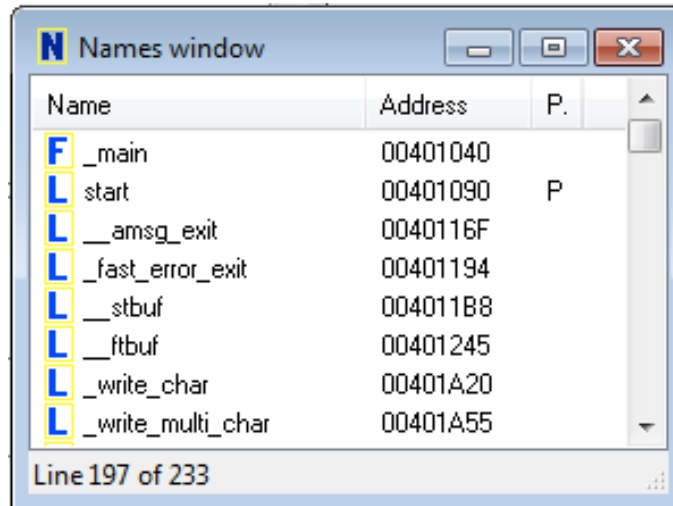
- Shows each function, length, and flags
 - L = Library functions
- Sortable
 - Large functions usually more important



Function name	Segment	Start	Length	R	F	L	S	B	T	=
Check\WindowsGenuineStatus()	.text	010091F9	0000007C	R	.	.	.	B	.	.
ControlBackgroundBrushInfo::`scalar deletingtext	01032897	00000029	R	.	.	.	B	.	.
CreateDecoderFromResource(I\WICImagingF...	.text	0101FB50	00000097	R	.	.	.	B	T	.

Names Window

- Every address with a name
 - Functions, named code, named data, strings



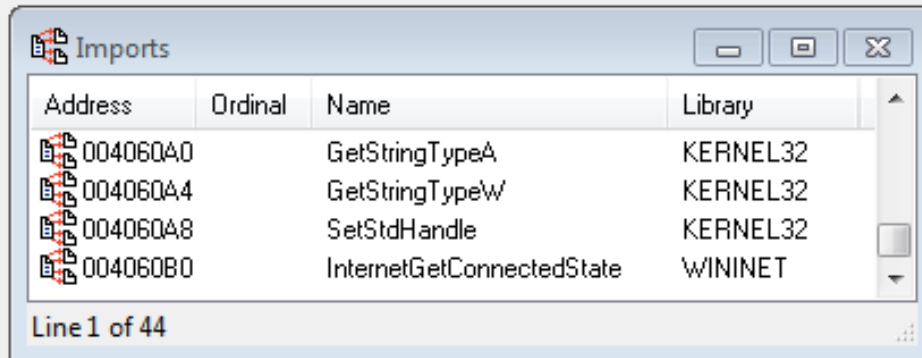
Strings

"..." Strings window

Address	Length	Type	String
"..." .rdata:0...	0000000F	C	GetStringTypeW
"..." .rdata:0...	0000000D	C	SetStdHandle
"..." .rdata:0...	0000000C	C	CloseHandle
"..." .rdata:0...	0000000D	C	KERNEL32.dll
"..." .data:00...	00000018	C	Error 1.1: No Internet\n
"..." .data:00...	0000001E	C	Success: Internet Connection\n

...

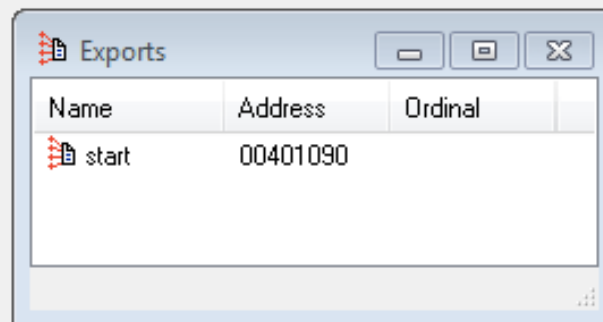
Imports & Exports



The Imports window displays a list of functions imported by the application. Each entry includes a red icon, a memory address, an ordinal, the function name, and the source library. The status bar at the bottom indicates 'Line 1 of 44'.

Address	Ordinal	Name	Library
004060A0		GetStringTypeA	KERNEL32
004060A4		GetStringTypeW	KERNEL32
004060A8		SetStdHandle	KERNEL32
004060B0		InternetGetConnectedState	WININET

Line 1 of 44

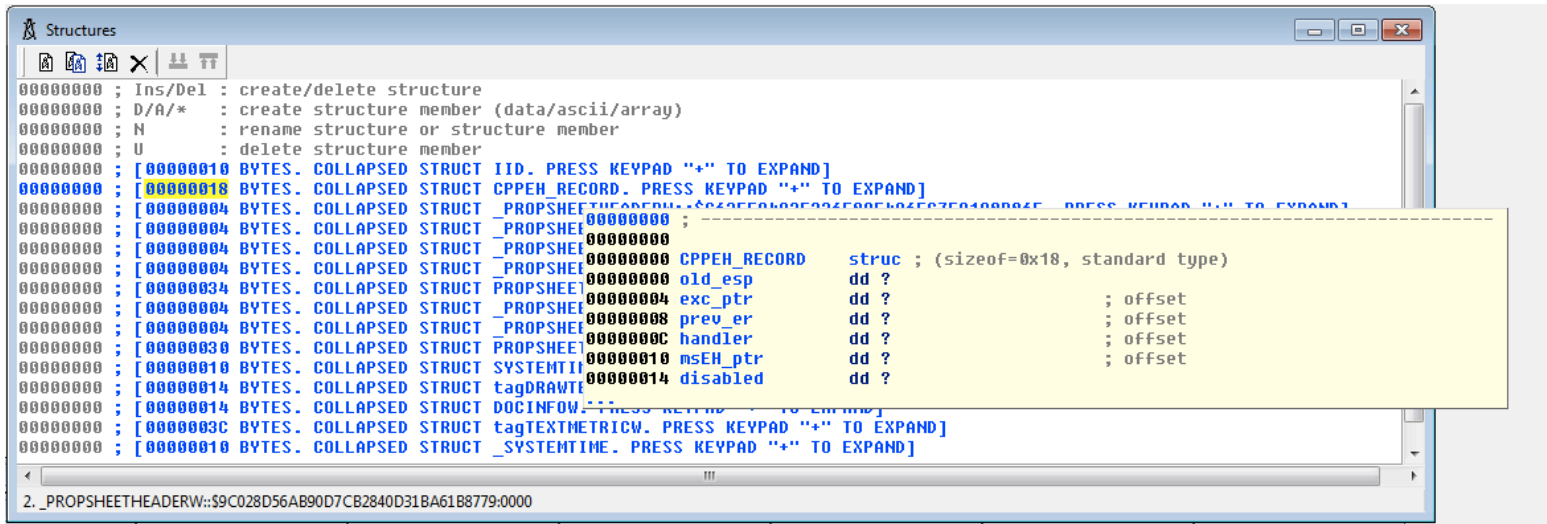


The Exports window displays the functions exported by the application. It shows a single entry for the 'start' function at address 00401090.

Name	Address	Ordinal
start	00401090	

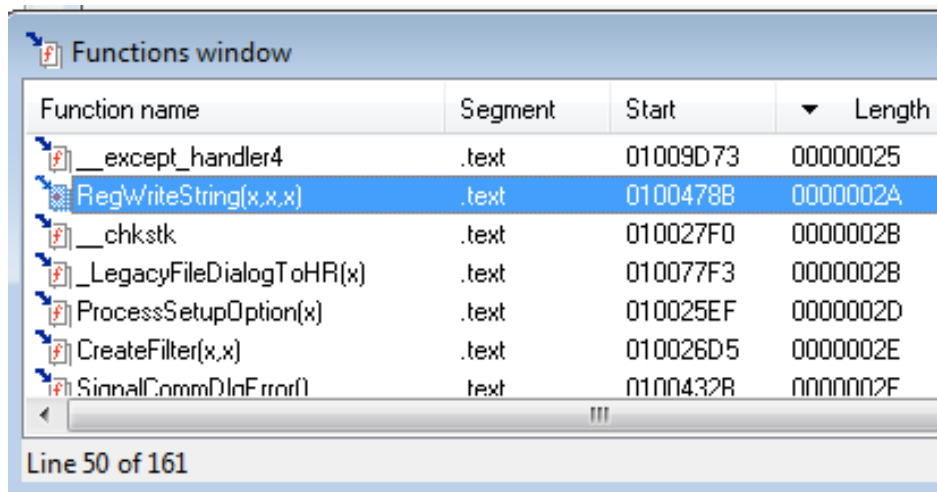
Structures








- All active data structures
 - Hover to see yellow pop-up window



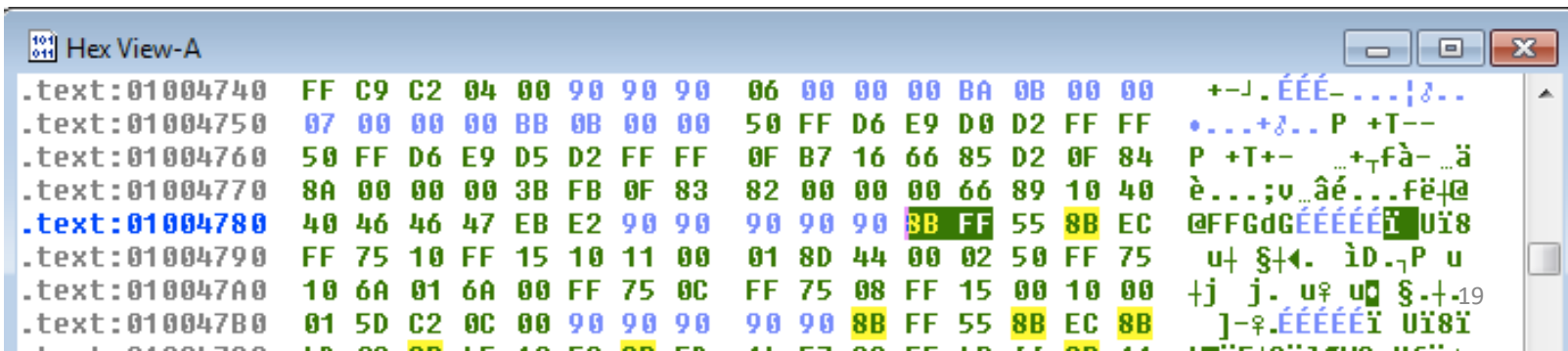
Cross-Reference

- Double-click function
- Jump to code in other views



Function name	Segment	Start	Length
 __except_handler4	.text	01009D73	00000025
 RegWriteString(x,x,x)	.text	01004788	0000002A
 __chkstk	.text	010027F0	0000002B
 _LegacyFileDialogToHR(x)	.text	010077F3	0000002B
 ProcessSetupOption(x)	.text	010025EF	0000002D
 CreateFilter(x,x)	.text	010026D5	0000002E
 SignalCommDlgInError()	.text	01004328	0000002F

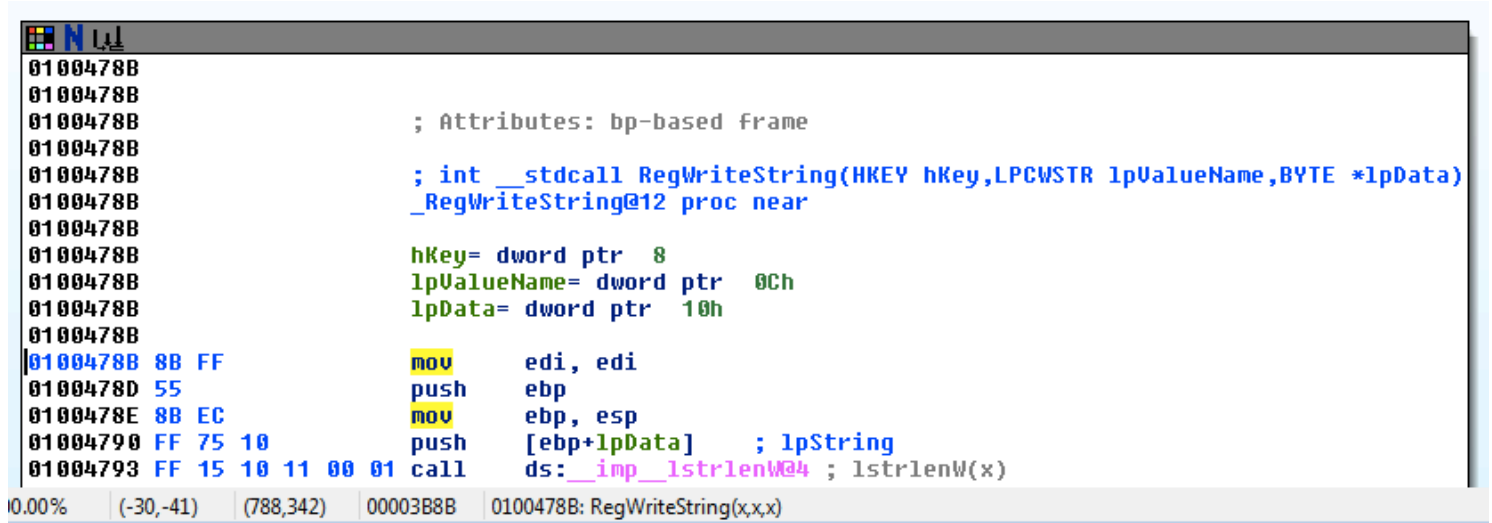
Line 50 of 161



Address	Hex	ASCII
.text:01004740	FF C9 C2 04 00 90 90 90 06 00 00 00 BA 0B 00 00	+-. ÉÉÉ- ...!&..
.text:01004750	07 00 00 00 BB 0B 00 00 50 FF D6 E9 D0 D2 FF FF+&.. P +T--
.text:01004760	50 FF D6 E9 D5 D2 FF FF 0F B7 16 66 85 D2 0F 84	P +T+- ...+Tfà- ä
.text:01004770	8A 00 00 00 3B FB 0F 83 82 00 00 00 66 89 10 40	è...;v...âé...fë+@
.text:01004780	40 46 46 47 EB E2 90 90 90 90 90 FF 55 8B EC	@FFGdGÉÉÉÉÉÛi8
.text:01004790	FF 75 10 FF 15 10 11 00 01 8D 44 00 02 50 FF 75	u+ \$+4. iD.7P u
.text:010047A0	10 6A 01 6A 00 FF 75 0C FF 75 08 FF 15 00 10 00	+j j. u# u\$.+19
.text:010047B0	01 5D C2 0C 00 90 90 90 90 90 8B FF 55 8B EC 8B]~.ÉÉÉÉÉÛi8i

Function Call

- Parameters pushed onto stack
- CALL to start function



```
0100478B
0100478B
0100478B ; Attributes: bp-based frame
0100478B
0100478B ; int __stdcall RegWriteString(HKEY hKey,LPCWSTR lpValueName,BYTE *lpData)
0100478B _RegWriteString@12 proc near
0100478B
0100478B hKey= dword ptr 8
0100478B lpValueName= dword ptr 0Ch
0100478B lpData= dword ptr 10h
0100478B
0100478B 8B FF mov edi, edi
0100478D 55 push ebp
0100478E 8B EC mov ebp, esp
01004790 FF 75 10 push [ebp+lpData] ; lpString
01004793 FF 15 10 11 00 01 call ds:__imp__lstrlenW@4 ; lstrlenW(x)
```

0.00% (-30,-41) (788,342) 00003B8B 0100478B: RegWriteString(x,x,x)

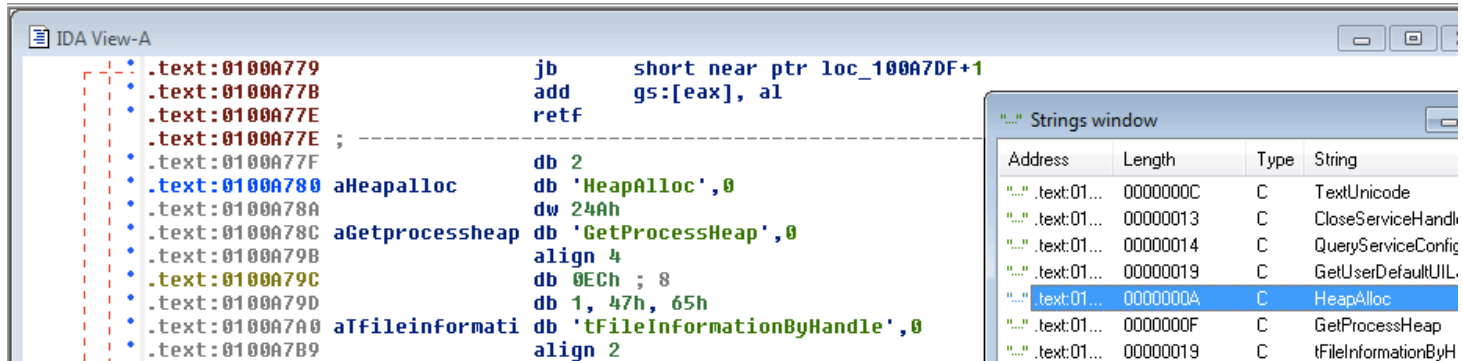
Returning to the Default View

- Windows, Reset Desktop
- Windows, Save Desktop
 - To save a new view

Navigating IDA Pro

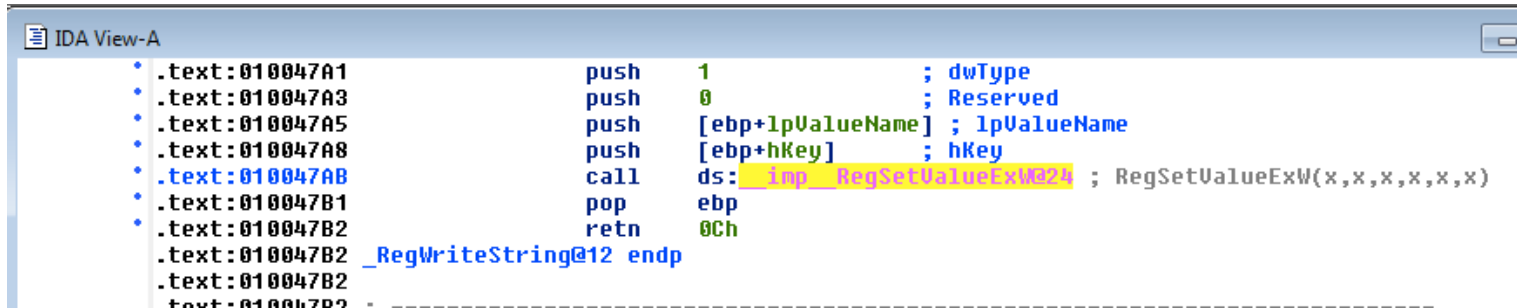
Imports or Strings

- Double-click any entry to display it in the disassembly window



Using Links

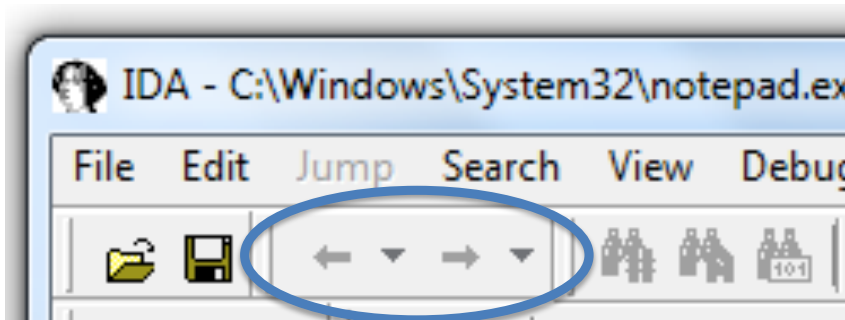
- Double-click any address in the disassembly window to display that location



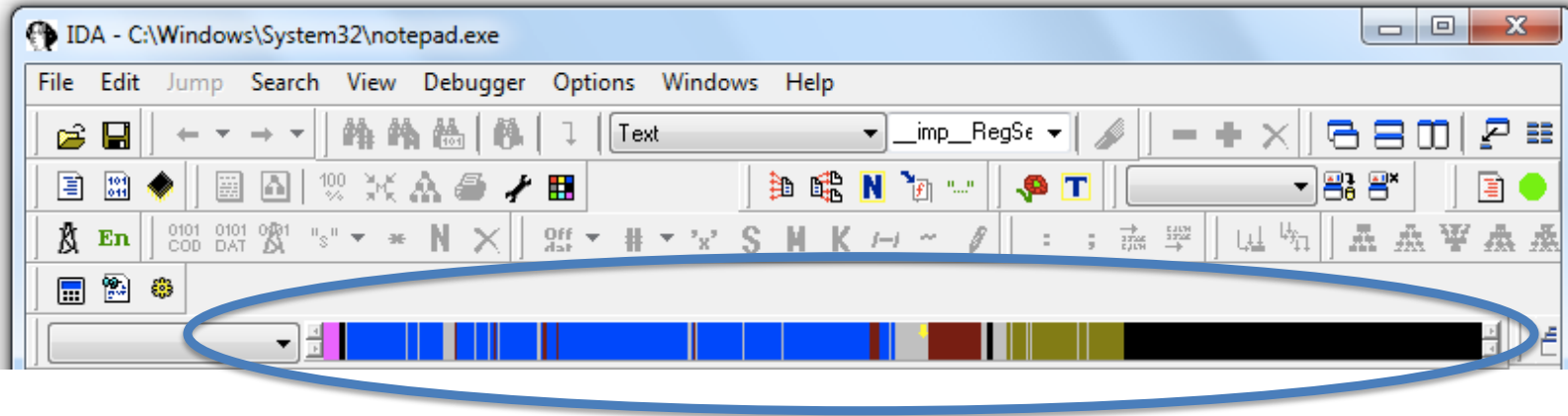
```
IDA View-A
• .text:010047A1      push     1           ; dwType
• .text:010047A3      push     0           ; Reserved
• .text:010047A5      push     [ebp+lpValueName] ; lpValueName
• .text:010047A8      push     [ebp+hKey]    ; hKey
• .text:010047AB      call     ds:__imp__RegSetValueEx@24 ; RegSetValueExW(x,x,x,x,x,x)
• .text:010047B1      pop      ebp
• .text:010047B2      retn     0Ch
.text:010047B2      _RegWriteString@12 endp
.text:010047B2
.text:010047B2      -----
```


History

- Forward and Back buttons work like a Web browser



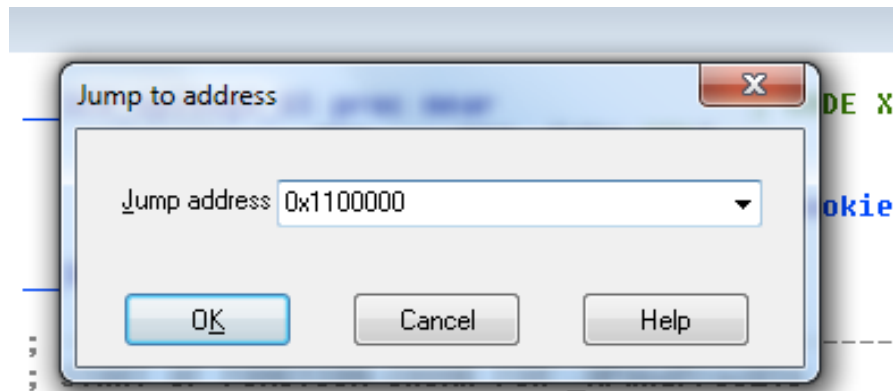
Navigation Band



- **Light blue:** Library code
- **Red:** Compiler-generated code
- **Dark blue:** User-written code – **Analyze this**

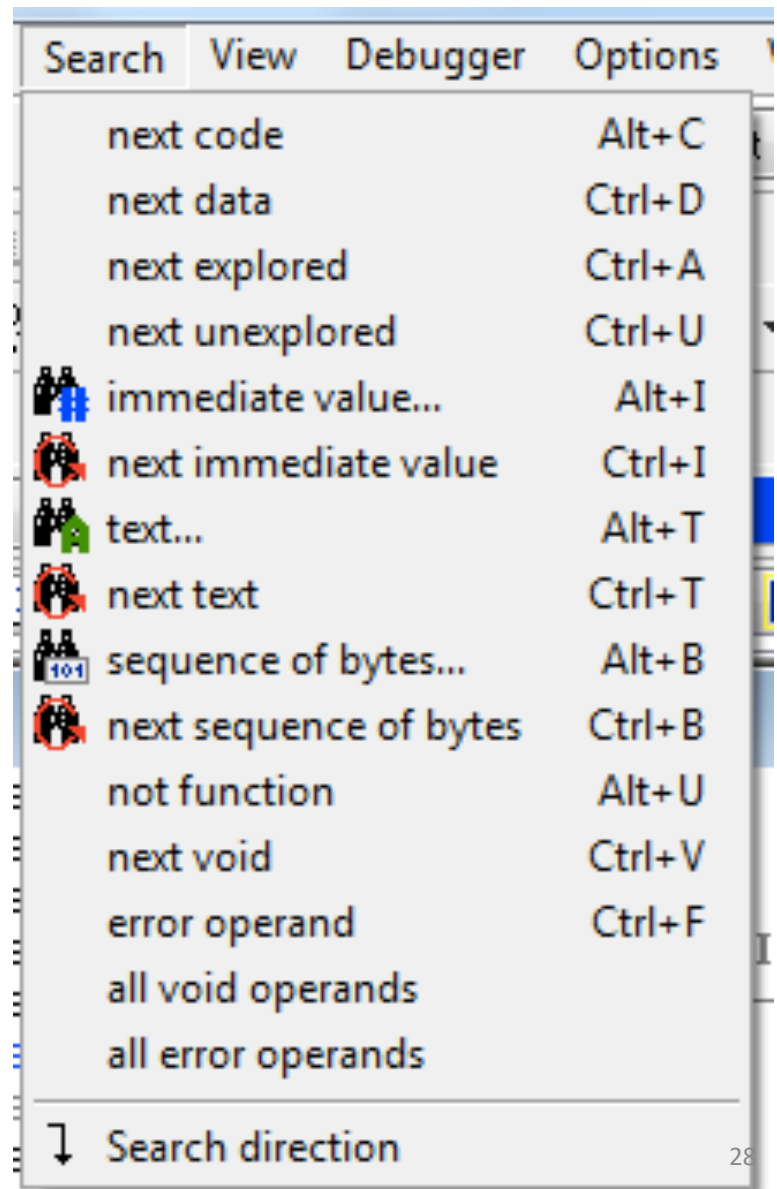
Jump to Location

- Press **G**
- Can jump to address or named location



Searching

- Many options
- Search, Text is handy



Using Cross-References

Code Cross-References

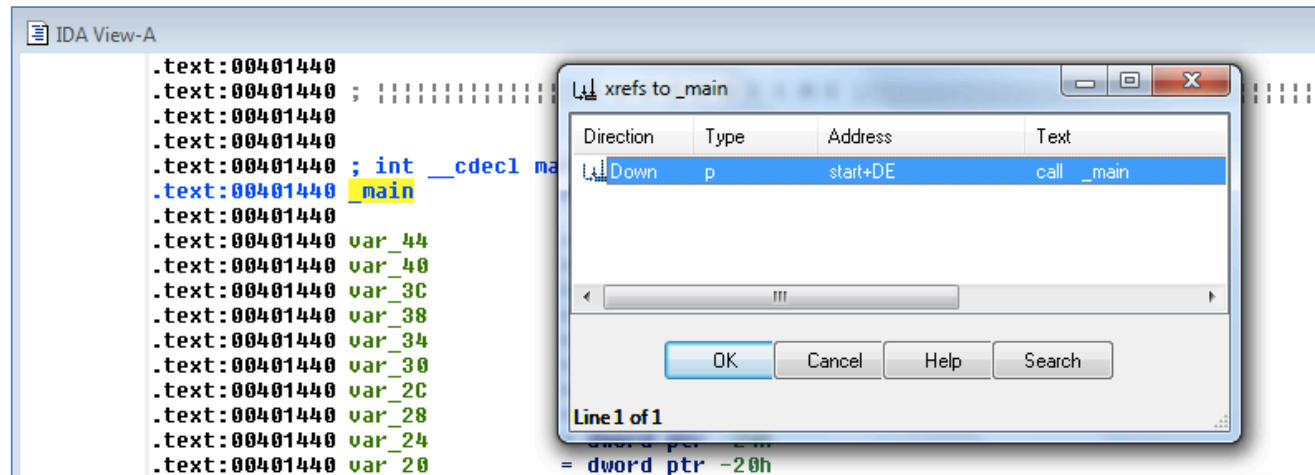
```
.text:00401440
.text:00401440 ; ;;;;;;;;;;;;;; S U B R O U T I N E ;;;;;;;;;;;;;;
.text:00401440
.text:00401440 ; int __cdecl main(int argc,const char **argv,const char *envp)
.text:00401440 _main      proc near      ; CODE XREF: start+DE↓p
.text:00401440
.text:00401440 var_44      = dword ptr -44h
.text:00401440 var_40      = dword ptr -40h
.text:00401440 var_3C      = dword ptr -3Ch
.text:00401440 var_38      = dword ptr -38h
.text:00401440 var_34      = dword ptr -34h
.text:00401440 var_30      = dword ptr -30h
.text:00401440 var_2C      = dword ptr -2Ch
.text:00401440 var_28      = dword ptr -28h
.text:00401440 var_24      = dword ptr -24h
.text:00401440 var_20      = dword ptr -20h
.text:00401440 var_1C      = dword ptr -1Ch
.text:00401440 var_18      = dword ptr -18h

        push    offset unk_403000
        call    _initterm
        call    ds:_p__initenv
        mov     ecx, [ebp+envp]
        mov     [eax], ecx
        push    [ebp+envp]      ; envp
        push    [ebp+argv]     ; argv
        push    [ebp+argc]     ; argc
        call    main
        add     esp, 30h
```

- XREF comment shows where this function is called
- But it only shows a couple of cross-references by default

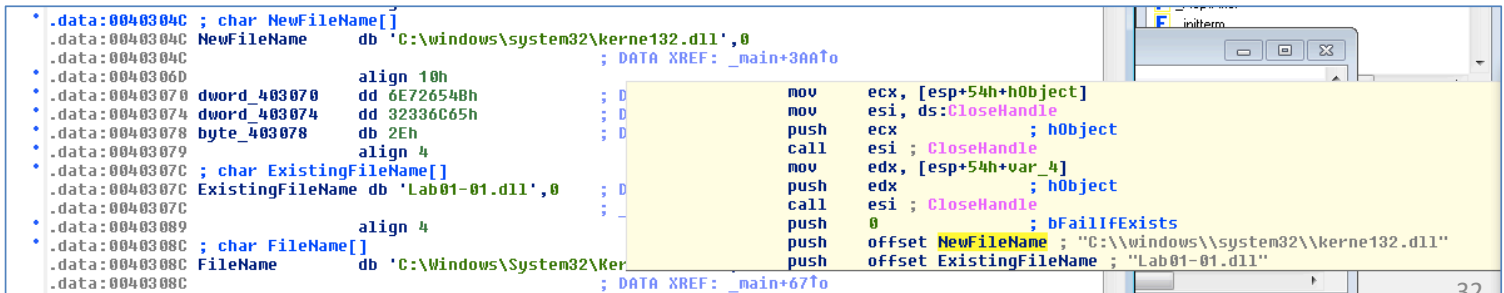
To See All Cross-References

- Click function name and press **X**



Data Cross-References

- Demo:
 - Start with strings
 - Double-click an interesting string
 - Hover over DATA XREF to see where that string is used
 - **X** shows all references



The screenshot shows a debugger window with two panes. The left pane displays assembly code with data cross-references (DATA XREF) highlighted in blue. The right pane displays assembly instructions with a yellow highlight.

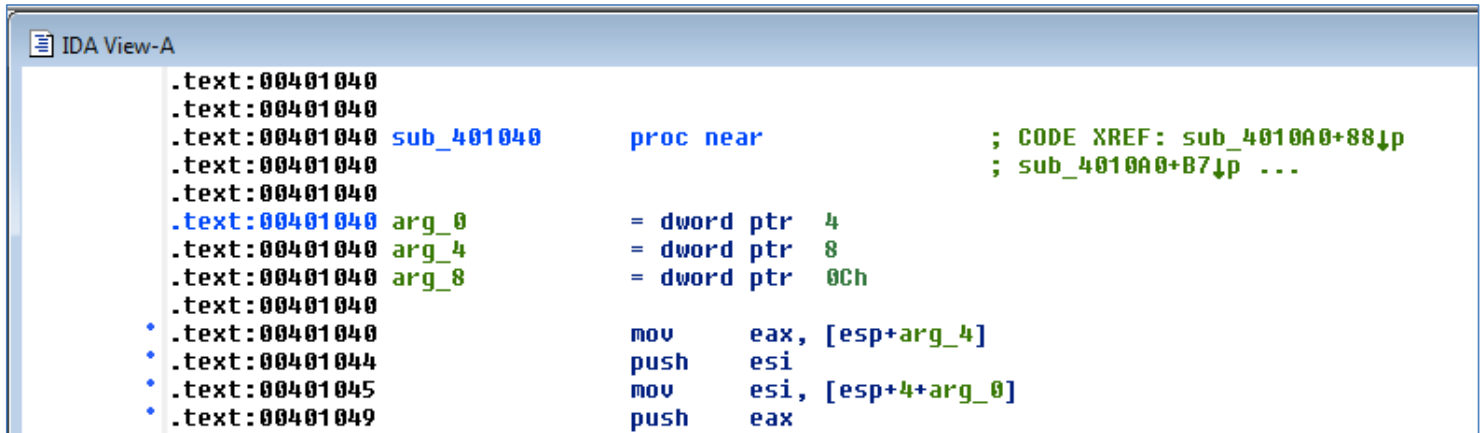
```
.data:0040304C ; char NewFileName[]  
.data:0040304C NewFileName db 'C:\windows\system32\kerne132.dll',0  
.data:0040304C ; DATA XREF: _main+3AA7o  
.data:0040306D align 10h  
.data:00403070 dword_403070 dd 6E726548h ; D  
.data:00403074 dword_403074 dd 32336C65h ; D  
.data:00403078 byte_403078 db 2Eh ; D  
.data:00403079 align 4  
.data:0040307C ; char ExistingFileName[]  
.data:0040307C ExistingFileName db 'Lab01-01.dll',0 ; D  
.data:0040307C ;  
.data:00403089 align 4  
.data:0040308C ; char FileName[]  
.data:0040308C FileName db 'C:\Windows\System32\Ker  
.data:0040308C ; DATA XREF: _main+67fo
```

```
mov ecx, [esp+54h+hObject]  
mov esi, ds:CloseHandle  
push ecx ; hObject  
call esi ; CloseHandle  
mov edx, [esp+54h+var_4]  
push edx ; hObject  
call esi ; CloseHandle  
push 0 ; bFailIfExists  
push offset NewFileName ; "C:\windows\system32\kerne132.dll"  
push offset ExistingFileName ; "Lab01-01.dll"
```


Analyzing Functions

Function and Argument Recognition

- IDA Pro identifies a function, names it, and also names the local variables
- It's not always correct

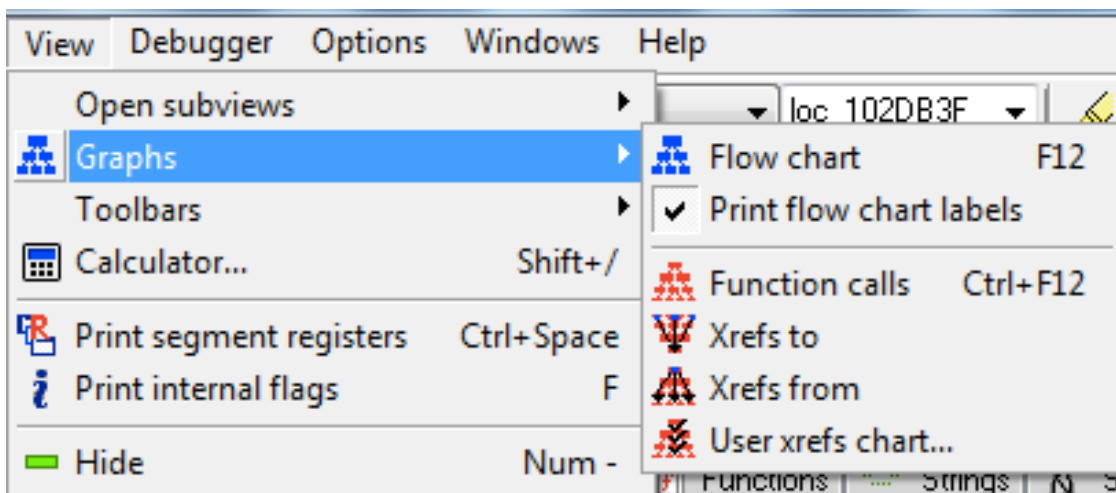
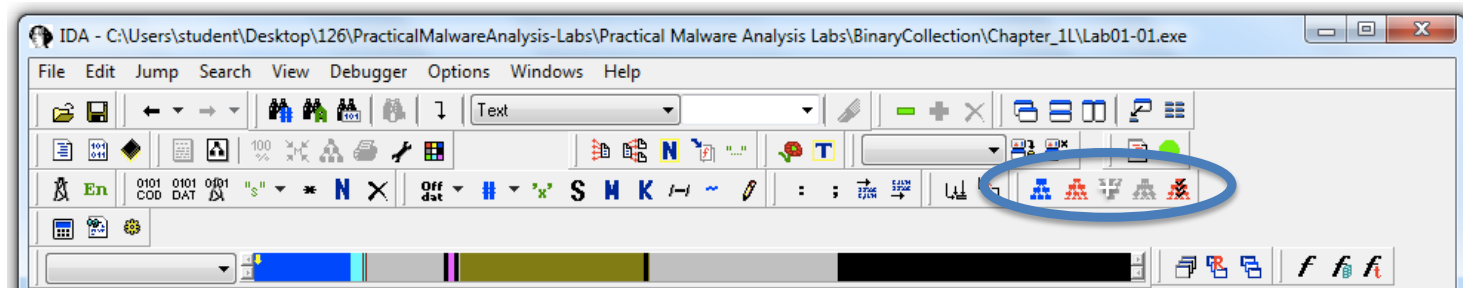


The screenshot shows the IDA View-A window with the following assembly code and argument definitions:

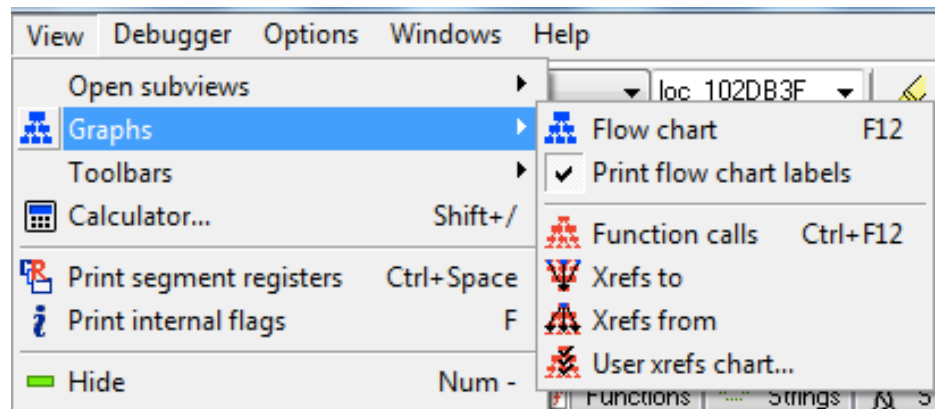
```
.text:00401040
.text:00401040
.text:00401040 sub_401040      proc near                ; CODE XREF: sub_4010A0+88↓p
.text:00401040                                     ; sub_4010A0+B7↓p ...
.text:00401040
.text:00401040 arg_0          = dword ptr 4
.text:00401040 arg_4          = dword ptr 8
.text:00401040 arg_8          = dword ptr 0Ch
.text:00401040
• .text:00401040      mov     eax, [esp+arg_4]
• .text:00401044      push    esi
• .text:00401045      mov     esi, [esp+4+arg_0]
• .text:00401049      push    eax
```

Using Graphing Options

Graphing Options

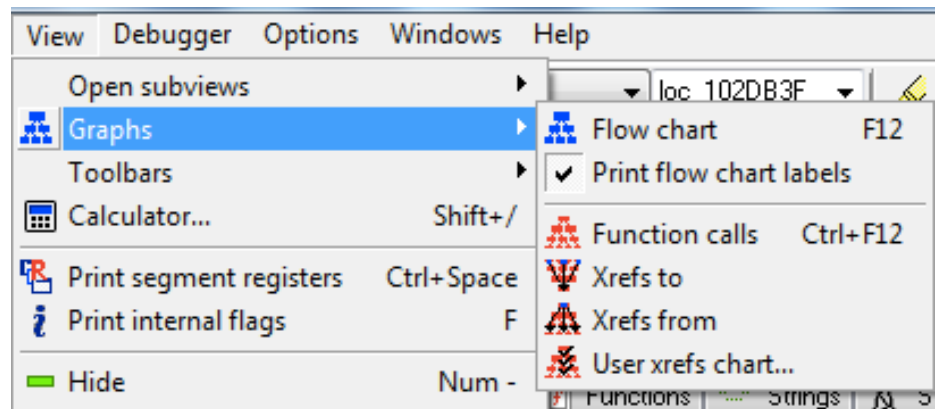


Graphing Options



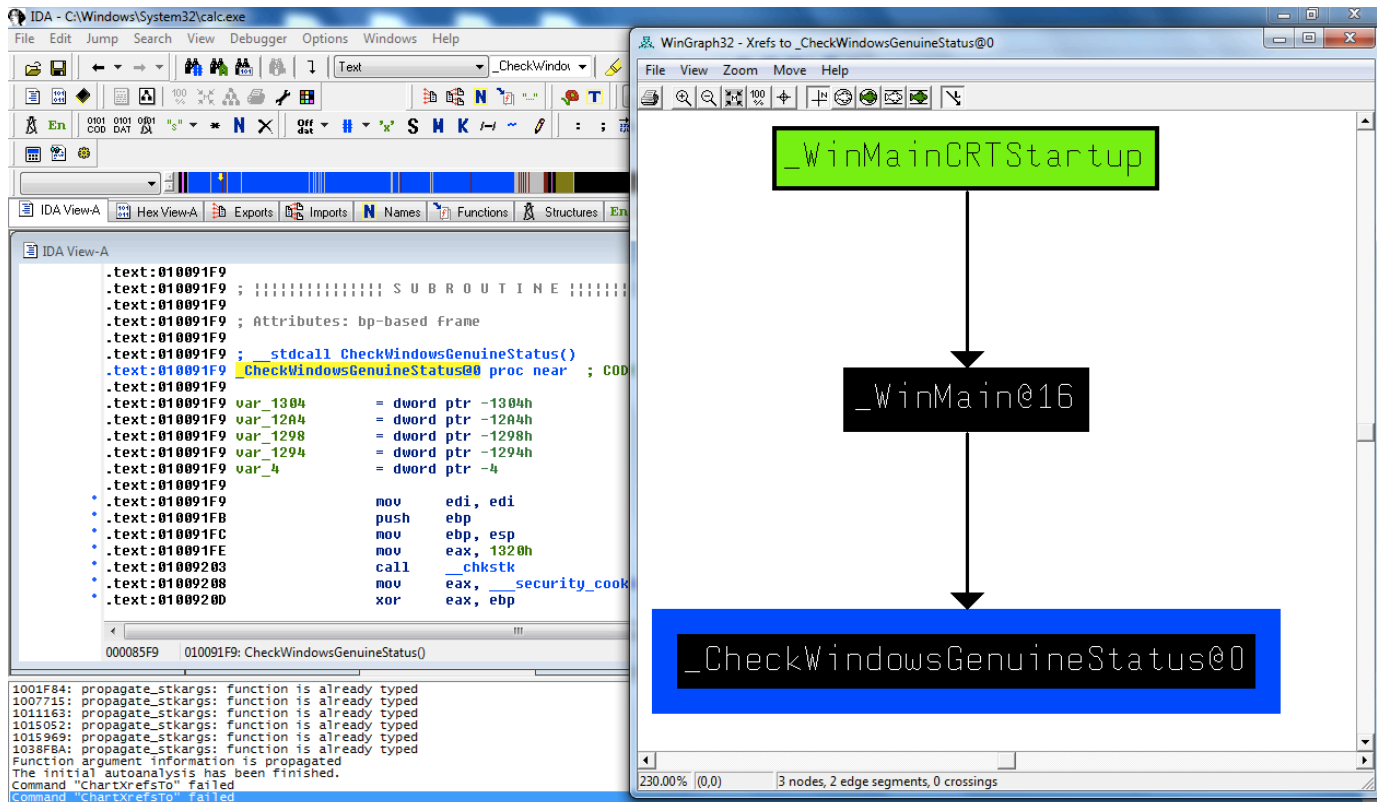
- These are "Legacy Graphs" and cannot be manipulated with IDA
- The first two seem obsolete
 - **Flow chart**
 - Create flow chart of current function
 - **Function calls**
 - Graph function calls for entire program

Graphing Options



- **Xrefs to**
 - Graphs XREFs to get to selected XREF
 - Can show all the paths that get to a function

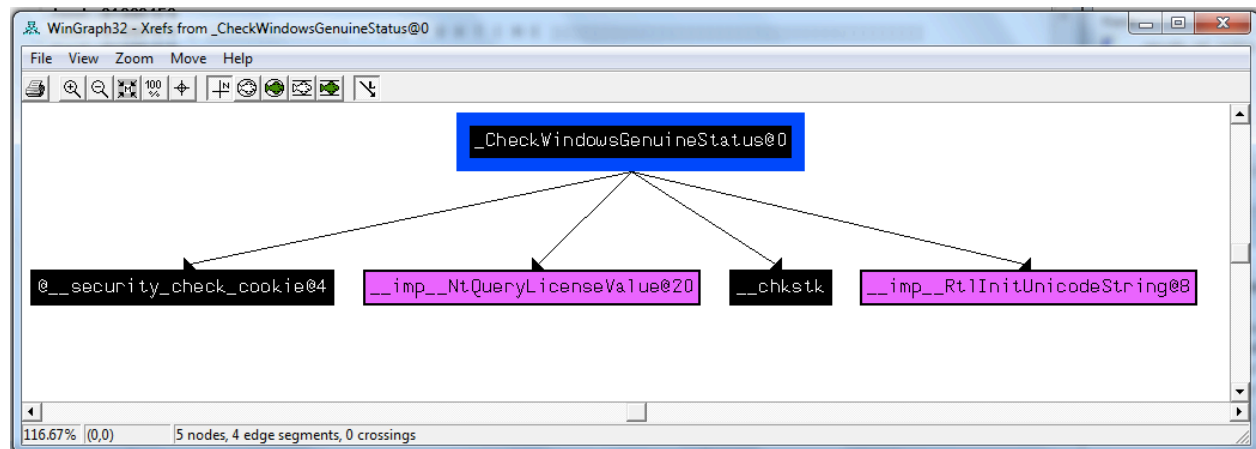
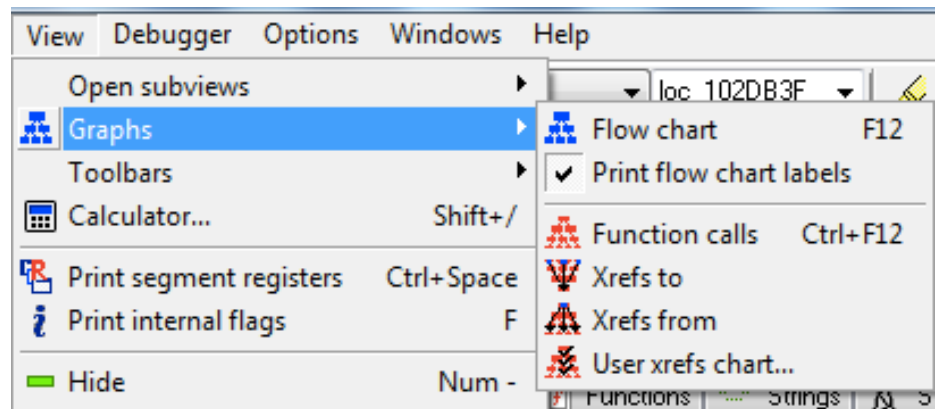
Windows Genuine Status in Calc.exe



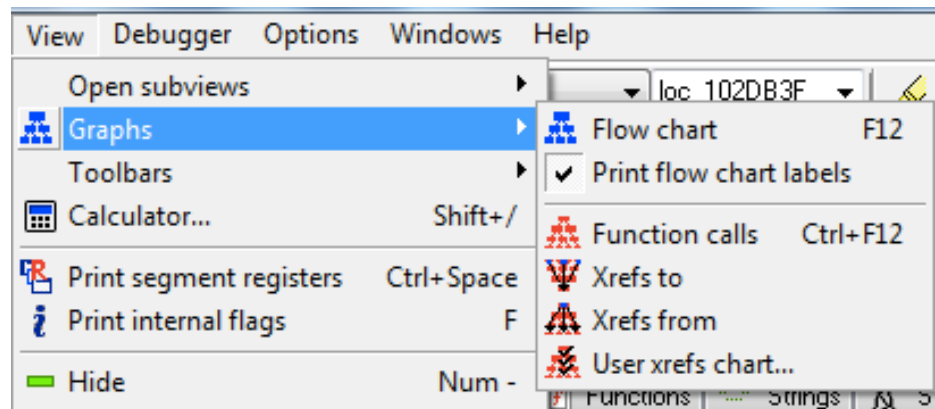
Graphing Options

- **Xrefs from**

- Graphs XREFs from selected XREF
- Can show all the paths that exit from a function



Graphing Options



- **User xrefs chart...**
 - Customize graph's recursive depth, symbols used, to or from symbol, etc.
 - The only way to modify legacy graphs

Enhancing Disassembly

Warning

- There's no Undo, so if you make changes and mess them up, you may be sorry

Renaming Locations

- You can change a name like **sub_401000** to **ReverseBackdoorThread**
- Change it in one place, IDA will change it everywhere else

Table 6-2. Function Operand Manipulation

Without renamed arguments

```

004013C8  mov     eax, [ebp+arg_4]
004013CB  push    eax
004013CC  call    _atoi
004013D1  add     esp, 4
004013D4  mov     [ebp+var_598], ax
004013DB  movzx   ecx, [ebp+var_598]
004013E2  test    ecx, ecx
004013E4  jnz     short loc_4013F8
004013E6  push    offset aError
004013EB  call    printf
004013F0  add     esp, 4
004013F3  jmp     loc_4016FB
004013F8  ; -----
004013F8
004013F8  loc_4013F8:
004013F8  movzx   edx, [ebp+var_598]
004013FF  push    edx
00401400  call    ds:htons

```

With renamed arguments

```

004013C8  mov     eax, [ebp+port_str]
004013CB  push    eax
004013CC  call    _atoi
004013D1  add     esp, 4
004013D4  mov     [ebp+port], ax
004013DB  movzx   ecx, [ebp+port]
004013E2  test    ecx, ecx
004013E4  jnz     short loc_4013F8
004013E6  push    offset aError
004013EB  call    printf
004013F0  add     esp, 4
004013F3  jmp     loc_4016FB
004013F8  ; -----
004013F8
004013F8  loc_4013F8:
004013F8  movzx   edx, [ebp+port]
004013FF  push    edx
00401400  call    ds:htons

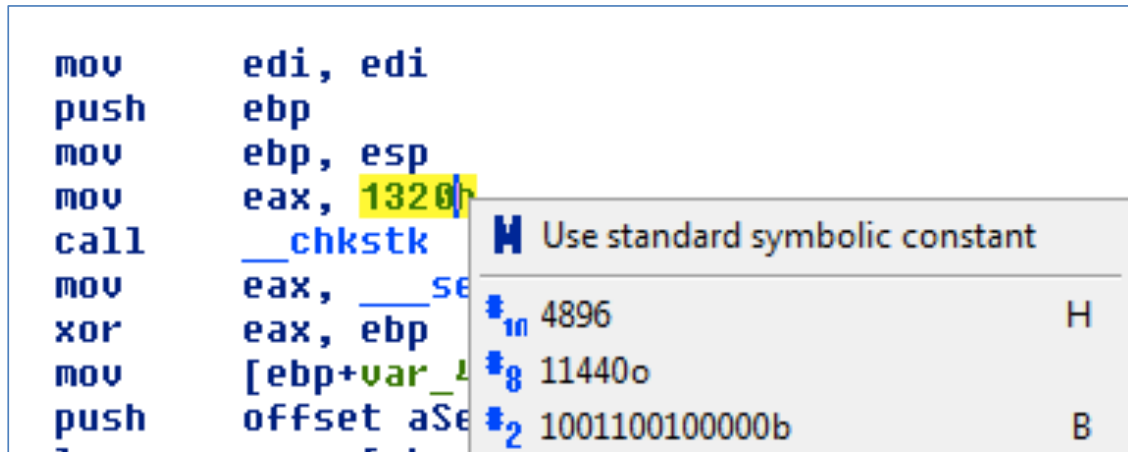
```

Comments

- Press colon (:) to add a single comment
- Press semicolon (;) to echo this comment to all Xrefs

Formatting Operands

- Hexadecimal by default
- Right-click to use other formats




Using Named Constants

- Makes Windows API arguments clearer

Before symbolic constants	After symbolic constants
<pre>mov esi, [esp+1Ch+argv] mov edx, [esi+4] mov edi, ds:CreateFileA push 0 ; hTemplateFile push 80h ; dwFlagsAndAttributes push 3 ; dwCreationDisposition push 0 ; lpSecurityAttributes push 1 ; dwShareMode</pre>	<pre>mov esi, [esp+1Ch+argv] mov edx, [esi+4] mov edi, ds:CreateFileA push NULL ; hTemplateFile push FILE_ATTRIBUTE_NORMAL ; dwFlagsAndAttributes push OPEN_EXISTING ; dwCreationDisposition push NULL ; lpSecurityAttributes push FILE_SHARE_READ ; dwShareMode</pre>

Extending IDA with Plug-ins

- IDC (IDA's scripting language) and Python scripts available (link Ch 6a)

 www.openrce.org/downloads/browse/IDA_Scripts			
	Decrypt Data	Unknown	IDA script to decipher data from HCU Millenium strainer stage 1 (AESCUL.EXE)
	Delphi RTTI script	RedPlait	This script deals with Delphi RTTI structures
	Export To Lib	Unknown	This script exports all functions to a lib file
	Find Format String Vulnerabilities	Unknown	A small IDC script hacked from sprintf.idc to detect format bugs currently ...