Binary Reversal Using IDA Pro

BY

HRISHIKESH N

AM.EN.P2CSN16008

What today's presentation is all about

What today's presentation is all about

What is Reverse Engineering?

“Reverse Engineering is the process of reversing anything that you can observe of a computer system - an application or a process or network traffic etc. and figuring out how it work without necessarily having any access to its documentation, design, source code etc.”

Why Reverse Engineering?

- Cracking Software – bypassing copy protection
  - PC software and games
  - Modding the Xbox and Playstation
- Exploit Development
  - Advanced exploitation
- Reversing undocumented operating system APIs
  - Virus writers
  - Spyware, keyloggers, malware, rootkits etc.

What today’s presentation is all about

- ~~Introduction – What Is Reverse Engineering?~~
- **What Is IDA Pro?**
- Delving Into IDA Pro
- Practical Demo of Bypassing Security Check Using IDA Pro
- Introduction to Patching
- Practical Demo On Patching a Binary
- Basic Static Malware Analysis Using IDA Pro
- Conclusion
- References

What is IDA Pro?

IDA Pro is a tool that combines an interactive, programmable, multi-processor disassembler coupled to a local and remote debugger and augmented by a complete plugin programming environment.

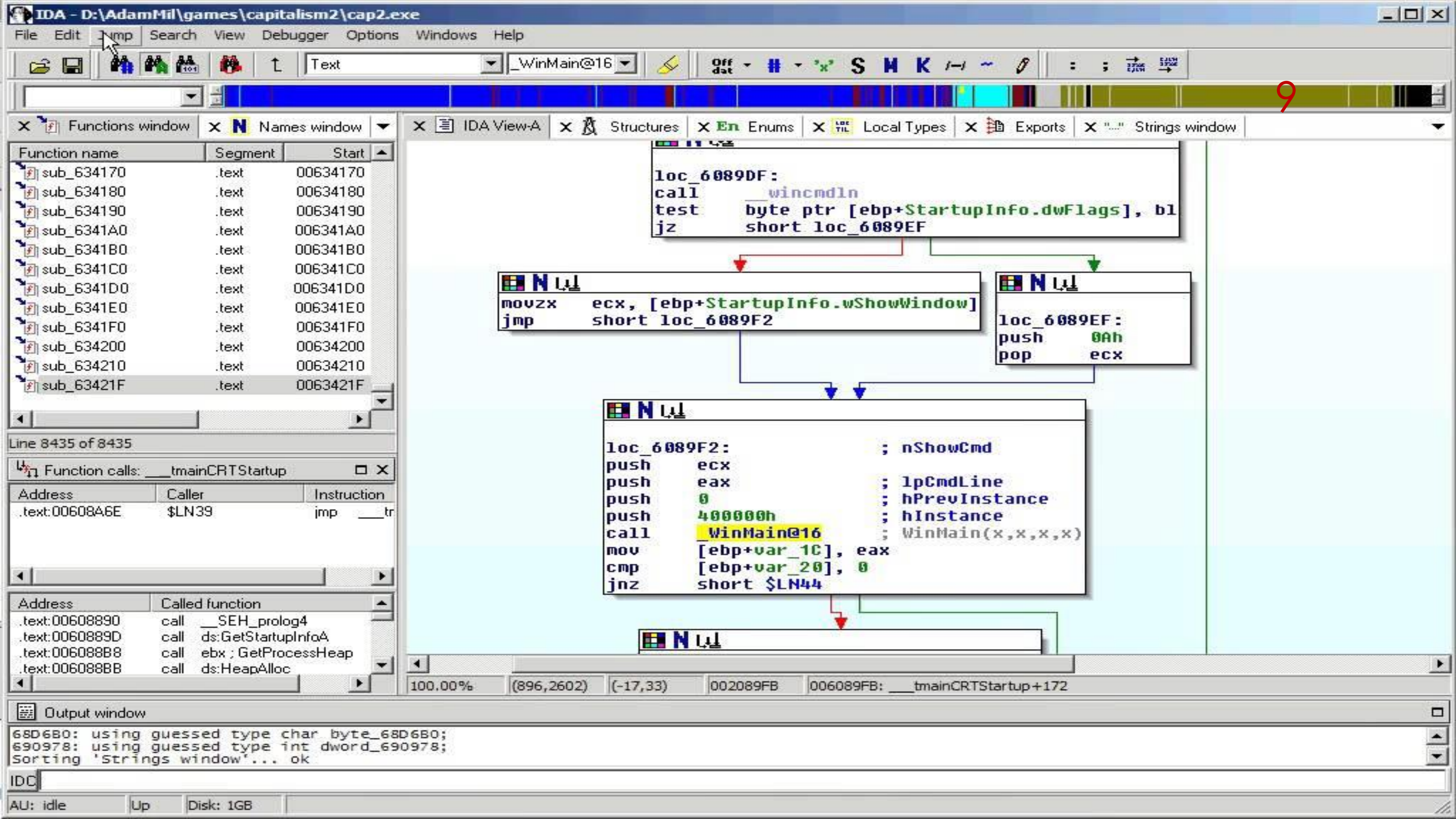
Features of IDA Pro

- As a disassembler, IDA Pro explores binary programs, for which source code isn't always available, to create maps of their execution.

- The debugger in IDA Pro complements the static analysis capabilities of the disassembler: by allowing an analyst to single step through the code being investigated, the debugger often bypasses the obfuscation and helps obtain data that the more powerful static disassembler will be able to process in depth.

- IDA always allows the human analyst to override its decisions or to provide hints. Interactivity culminates in a built-in programming language and an open plugin architecture.

- IDA Pro contains a complete development environment that consists of a very powerful macro-like language that can be used to automate simple to medium complexity tasks.



Function name	Segment	Start
sub_634170	.text	00634170
sub_634180	.text	00634180
sub_634190	.text	00634190
sub_6341A0	.text	006341A0
sub_6341B0	.text	006341B0
sub_6341C0	.text	006341C0
sub_6341D0	.text	006341D0
sub_6341E0	.text	006341E0
sub_6341F0	.text	006341F0
sub_634200	.text	00634200
sub_634210	.text	00634210
sub_63421F	.text	0063421F

Line 8435 of 8435

Function calls: ___tmainCRTStartup

Address	Caller	Instruction
.text:00608A6E	$LN39	jmp ___tr

Address	Called function
.text:00608890	call __SEH_prolog4
.text:0060889D	call ds:GetStartupInfoA
.text:006088B8	call ebx ; GetProcessHeap
.text:006088BB	call ds:HeapAlloc

```
loc_6089DF:
call    __wincmdln
test    byte ptr [ebp+StartupInfo.dwFlags], bl
jz      short loc_6089EF
```

```
movzx   ecx, [ebp+StartupInfo.wShowWindow]
jmp     short loc_6089F2
```

```
loc_6089EF:
push    0Ah
pop     ecx
```

```
loc_6089F2:                     ; nShowCmd
push    ecx
push    eax                     ; lpCmdLine
push    0                       ; hPrevInstance
push    400000h                 ; hInstance
call    _WinMain@16             ; WinMain(x,x,x,x)
mov     [ebp+var_1C], eax
cmp     [ebp+var_20], 0
jnz     short $LN44
```

```
100.00%  (896,2602)  (-17,33)  002089FB  006089FB: ___tmainCRTStartup+172
```

```
68D6B0: using guessed type char byte_68D6B0;
690978: using guessed type int dword_690978;
Sorting 'Strings window'... ok
```

IDC

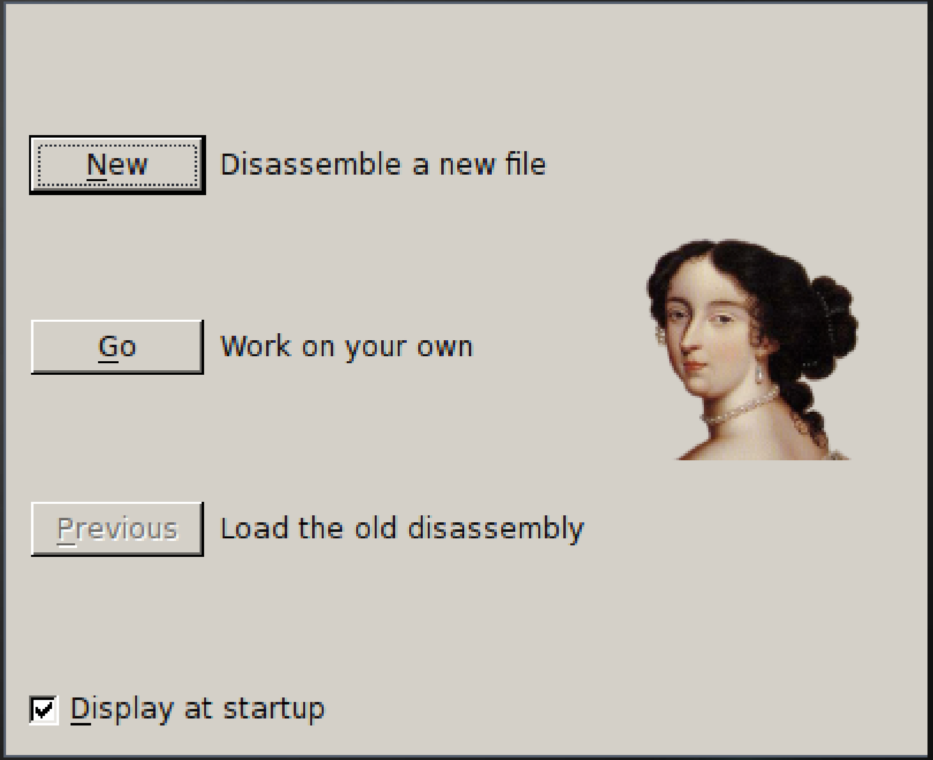
AU: idle   Up   Disk: 1GB

What today's presentation is all about

- ~~Introduction – What Is Reverse Engineering?~~
- ~~What Is IDA Pro?~~
- **Delving Into IDA Pro**
- Practical Demo of Bypassing Security Check Using IDA Pro
- Introduction to Patching
- Practical Demo On Patching a Binary
- Basic Static Malware Analysis Using IDA Pro
- Conclusion
- References

Delving Into IDA Pro

- When IDA Pro is first loaded, a dialog box will appear asking you to disassemble a new file, to enter the program without loading any file, or to load the previously loaded file. This can be seen below:



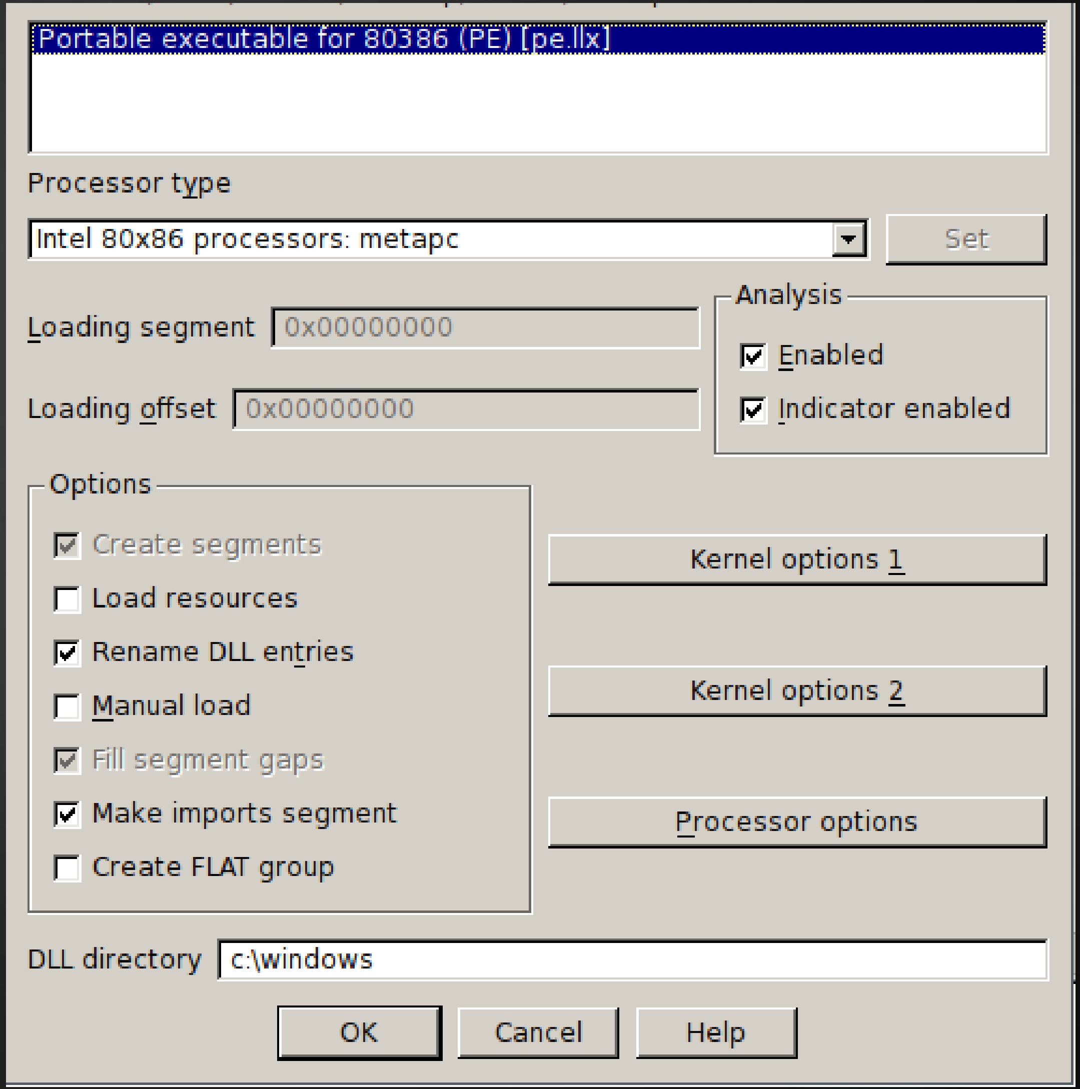
Disassemble a new file

Work on your own

Load the old disassembly

[✓] Display at startup

- Upon opening the executable, IDA Pro will automatically recognize the file format of the executable: in our case, it is a PE Windows executable. It will also recognize the architecture the executable was compiled against.



Portable executable for 80386 (PE) [pe.llx]

Processor type: Intel 80x86 processors: metapc

Loading segment: 0x00000000

Loading offset: 0x00000000

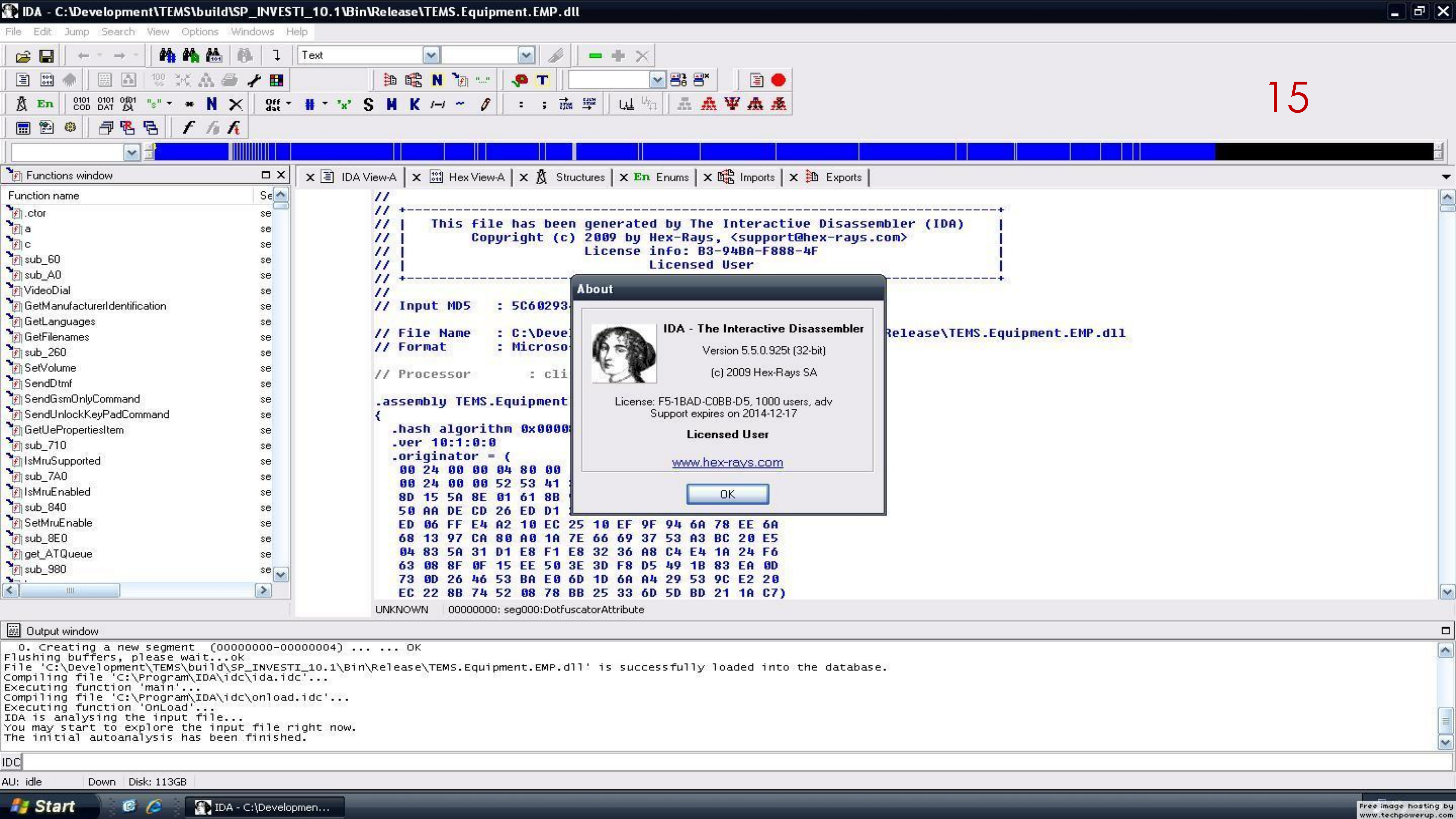
Analysis
- [✓] Enabled
- [✓] Indicator enabled

Options
- [✓] Create segments
- [ ] Load resources
- [✓] Rename DLL entries
- [ ] Manual load
- [✓] Fill segment gaps
- [✓] Make imports segment
- [ ] Create FLAT group

DLL directory: c:\windows

- Upon opening a new file to analyze with IDA Pro, it **analyzes** the whole executable file and creates an.idb database archive. The .idb archive contains four files:
  - name.id0 – contains contents of B-tree style database
  - name.id1 – contains flags that describe each program byte,
  - name.nam – contains index information related to named program locations,
  - name.til – contains information about local type definitions
- All of these file formats are proprietary and can only be used in IDA.

IDA Pro - GUI



IDA - C:\Development\TEMS\build\SP_INVESTI_10.1\Bin\Release\TEMS.Equipment.EMP.dll

Functions window

```
Function name
.ctor
a
c
sub_60
sub_A0
VideoDial
GetManufacturerIdentification
GetLanguages
GetFilenames
sub_260
SetVolume
SendDtmf
SendGsmOnlyCommand
SendUnlockKeyPadCommand
GetUePropertiesItem
sub_710
IsMruSupported
sub_7A0
IsMruEnabled
sub_840
SetMruEnable
sub_8E0
get_ATQueue
sub_980
```

IDA View-A

```
//
// +-------------------------------------------------------------------------+
// |   This file has been generated by The Interactive Disassembler (IDA)    |
// |        Copyright (c) 2009 by Hex-Rays, <support@hex-rays.com>           |
// |                     License info: B3-94BA-F888-4F                       |
// |                                   Licensed User                         |
// +-------------------------------------------------------------------------+
//
// Input MD5    : 5C60293
// File Name    : C:\Deve                                      Release\TEMS.Equipment.EMP.dll
// Format       : Microso
// Processor       : cli
.assembly TEMS.Equipment
{
  .hash algorithm 0x0000
  .ver 10:1:0:0
  .originator = (
   00 24 00 00 04 80 00
   00 24 00 00 52 53 41
   8D 15 5A 8E 01 61 8B
   50 AA DE CD 26 ED D1
   ED 06 FF E4 A2 10 EC 25 10 EF 9F 94 6A 78 EE 6A
   68 13 97 CA 80 A0 1A 7E 66 69 37 53 A3 BC 20 E5
   04 83 5A 31 D1 E8 F1 E8 32 36 A8 C4 E4 1A 24 F6
   63 08 8F 0F 15 EE 50 3E 3D F8 D5 49 1B 83 EA 0D
   73 0D 26 46 53 BA E0 6D 1D 6A A4 29 53 9C E2 20
   EC 22 8B 74 52 08 78 BB 25 33 6D 5D BD 21 1A C7)
```

UNKNOWN 00000000: seg000:DotfuscatorAttribute

About

IDA - The Interactive Disassembler

Version 5.5.0.925t (32-bit)

(c) 2009 Hex-Rays SA

License: F5-1BAD-C0BB-D5, 1000 users, adv
Support expires on 2014-12-17

Licensed User

www.hex-rays.com

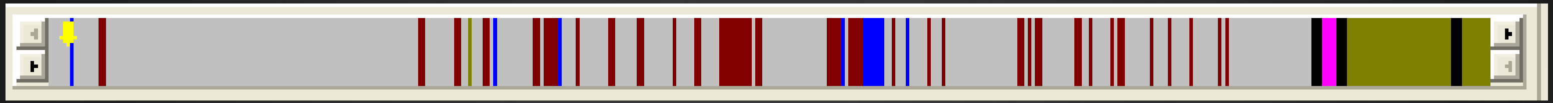
OK

Output window

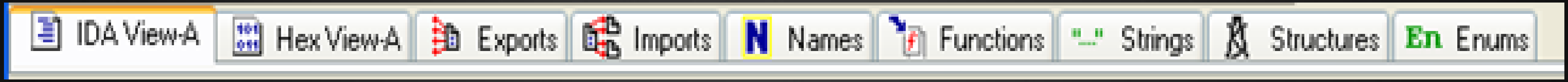
```
 0. Creating a new segment  (00000000-00000004) ... ... OK
Flushing buffers, please wait...ok
File 'C:\Development\TEMS\build\SP_INVESTI_10.1\Bin\Release\TEMS.Equipment.EMP.dll' is successfully loaded into the database.
Compiling file 'C:\Program\IDA\idc\ida.idc'...
Executing function 'main'...
Compiling file 'C:\Program\IDA\idc\onload.idc'...
Executing function 'OnLoad'...
IDA is analysing the input file...
You may start to explore the input file right now.
The initial autoanalysis has been finished.
```

AU: idle Down Disk: 113GB

- We can see the menu area that contains the menu items File, Edit, etc. This can be used to do anything that is possible to do with IDA; it’s just a matter of finding the right option we would like to do.
- The various parts are described in the upcoming slides



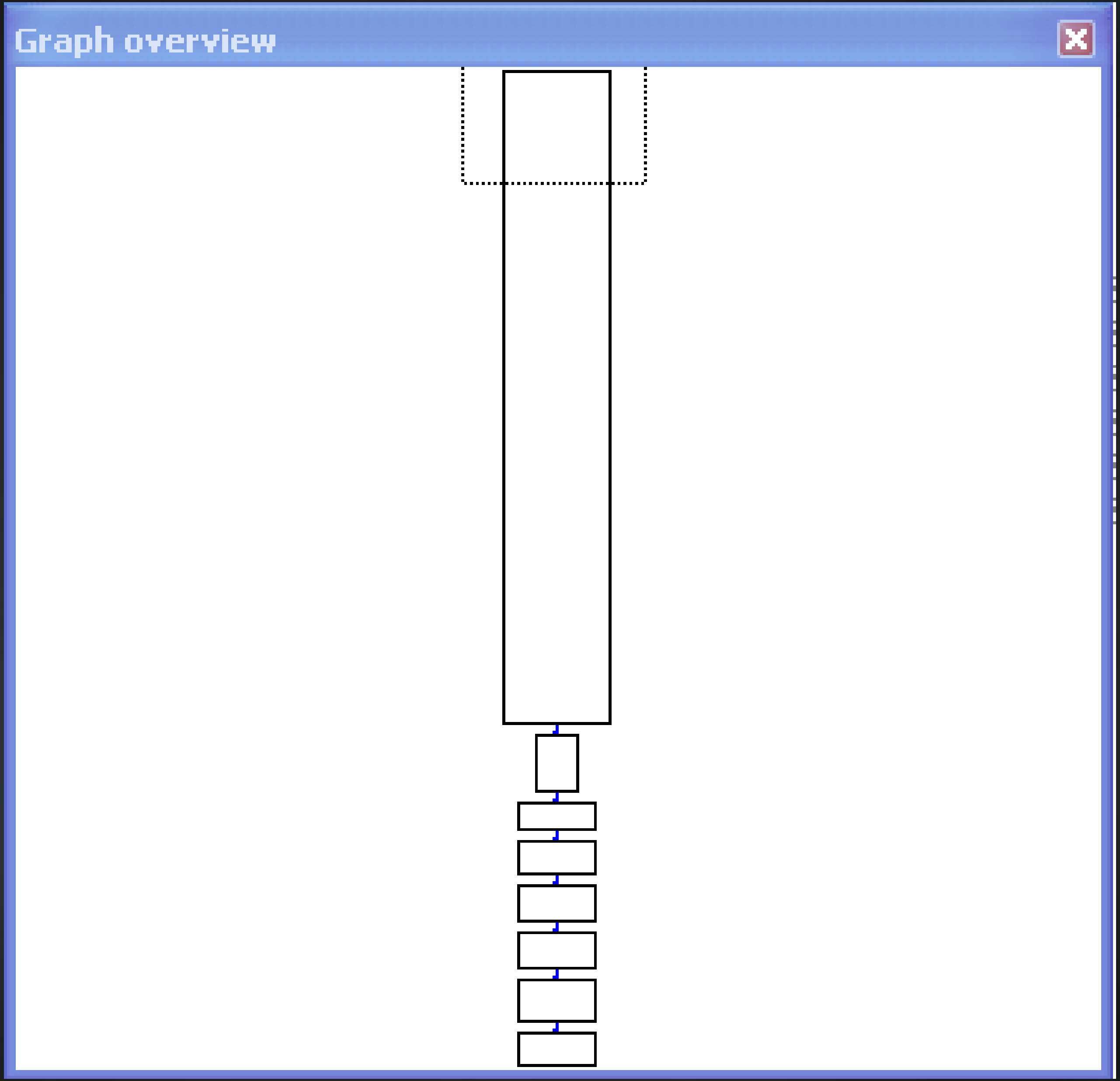
It represents the whole memory space used by the analyzed application. If we right-click on it, we can zoom in and out to represent smaller chunks of memory. We can also see that different colors are used for different parts of the memory; this depends on the type of data or code being loaded into that area. At the very beginning of the navigator, we can see a very small yellow arrow that points to the location where we’re currently at in the disassembly window.

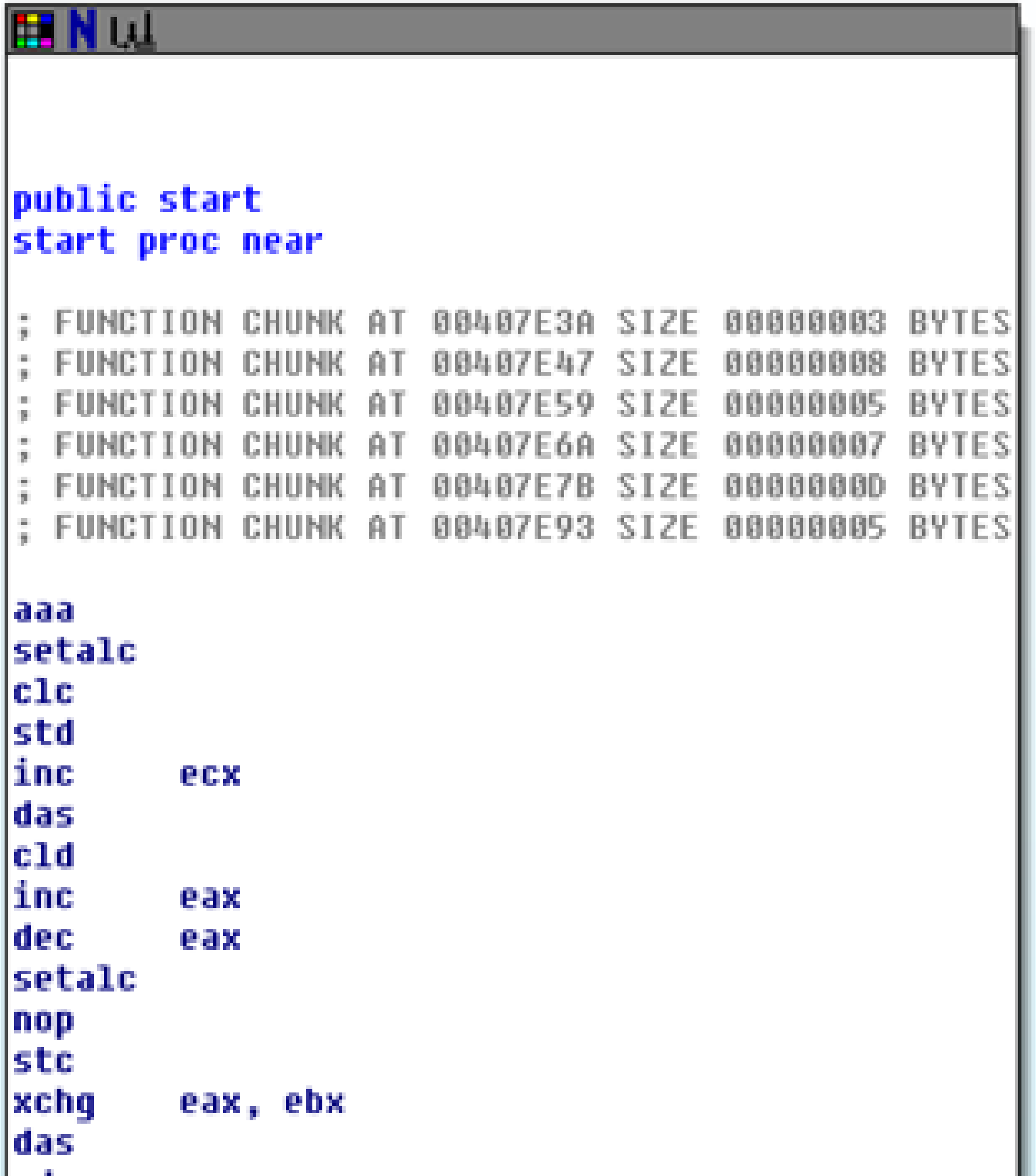


We can see that there are a lot of data views available and all of them contain one or more specific information that was gathered from the loaded executable. To open a specific data view, we can go to View – Open Subviews and choose the appropriate view we would like to show.

- The main view is the disassembly window where we can see the actual disassembled code of the analyzed executable. We can switch between the graph and the listing view that actually represents the same program.

- The graph view can be used if we want to quickly figure out the execution flow of the current function and the listing view can be used when we want to see the actual assembly instructions.

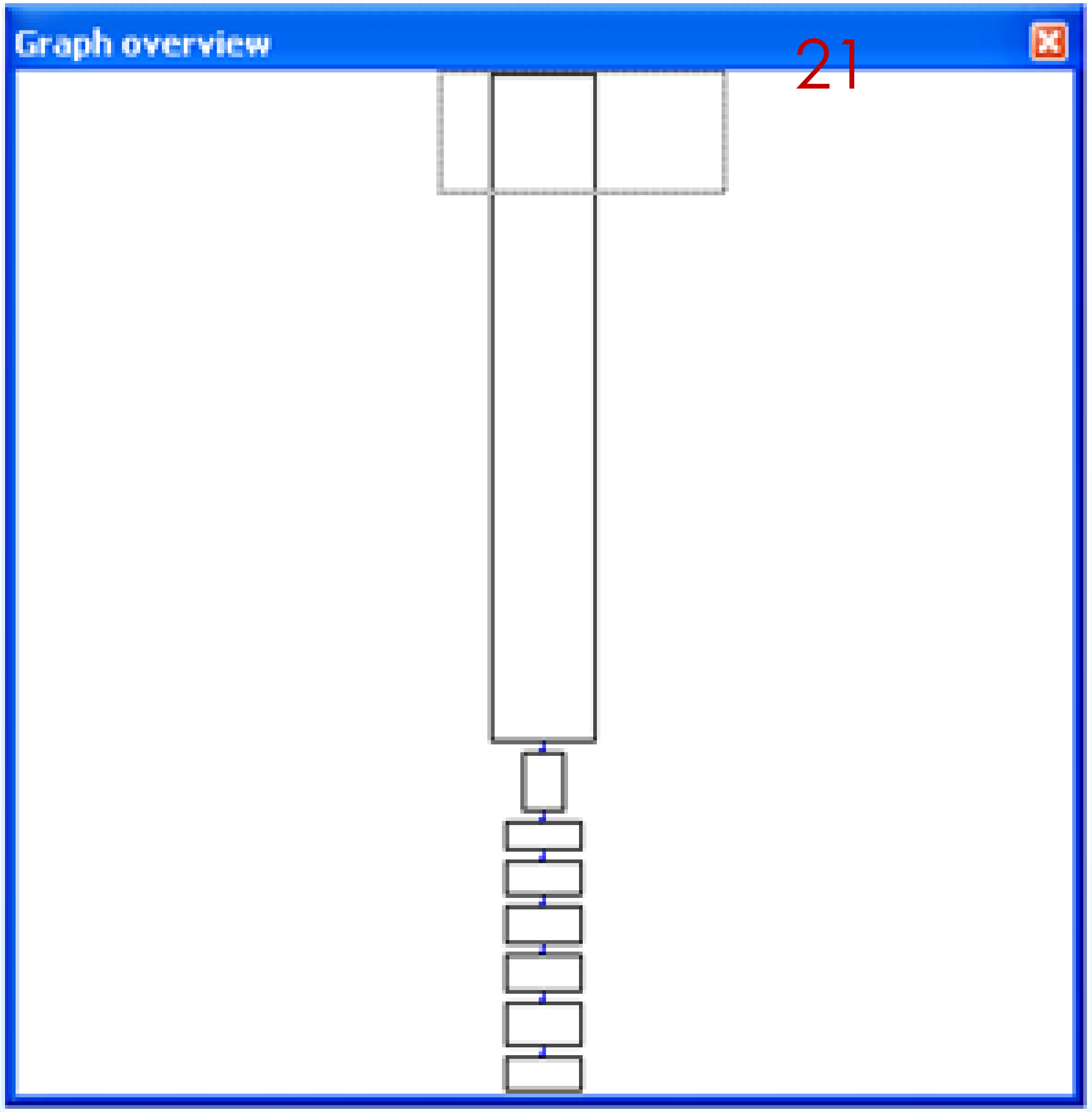

Graph overview



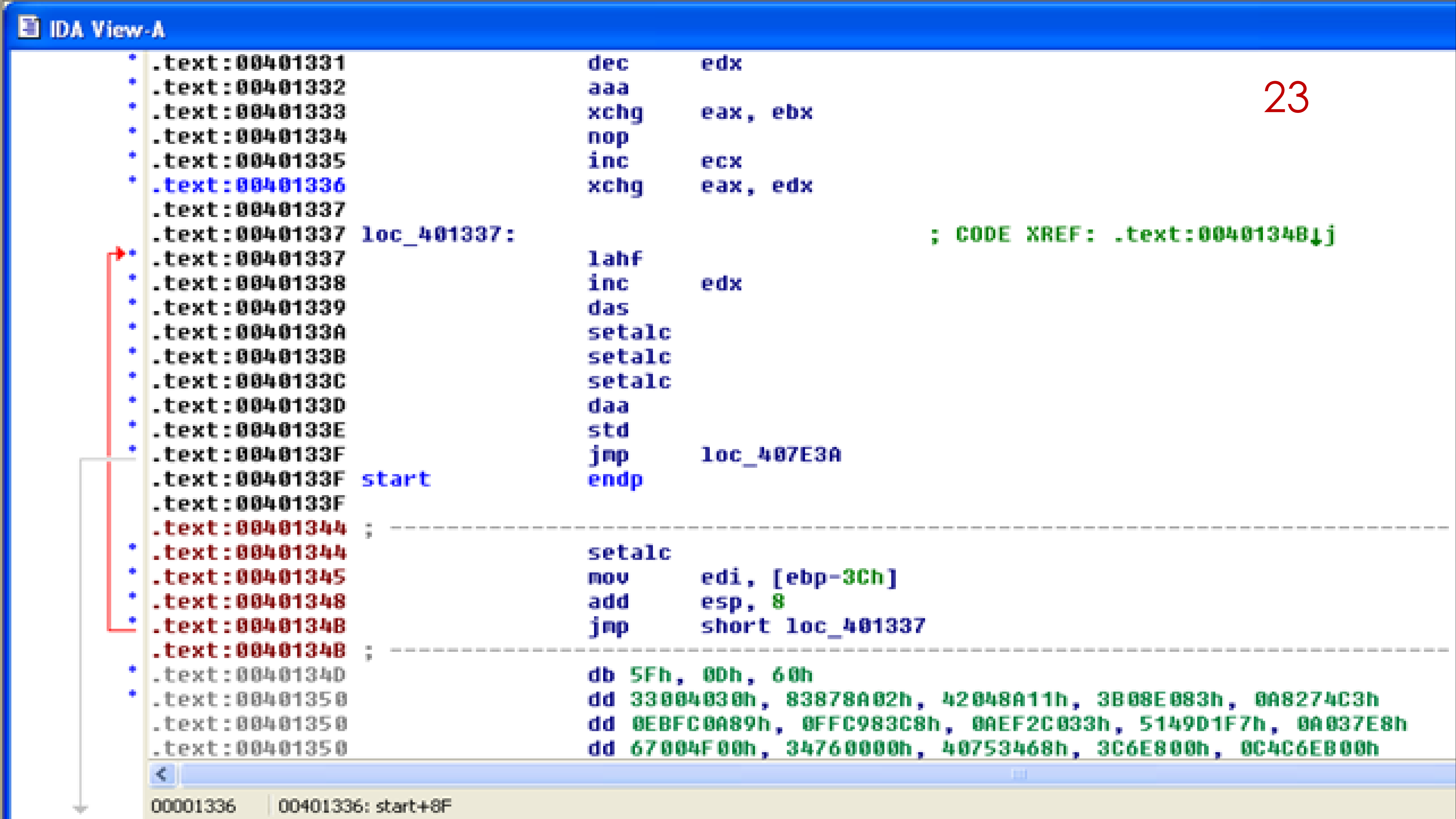
```
public start
start proc near

; FUNCTION CHUNK AT 00407E3A SIZE 00000003 BYTES
; FUNCTION CHUNK AT 00407E47 SIZE 00000008 BYTES
; FUNCTION CHUNK AT 00407E59 SIZE 00000005 BYTES
; FUNCTION CHUNK AT 00407E6A SIZE 00000007 BYTES
; FUNCTION CHUNK AT 00407E7B SIZE 0000000D BYTES
; FUNCTION CHUNK AT 00407E93 SIZE 00000005 BYTES

aaa
setalc
clc
std
inc     ecx
das
cld
inc     eax
dec     eax
setalc
nop
stc
xchg    eax, ebx
das
```



Disassembled View - IDA View

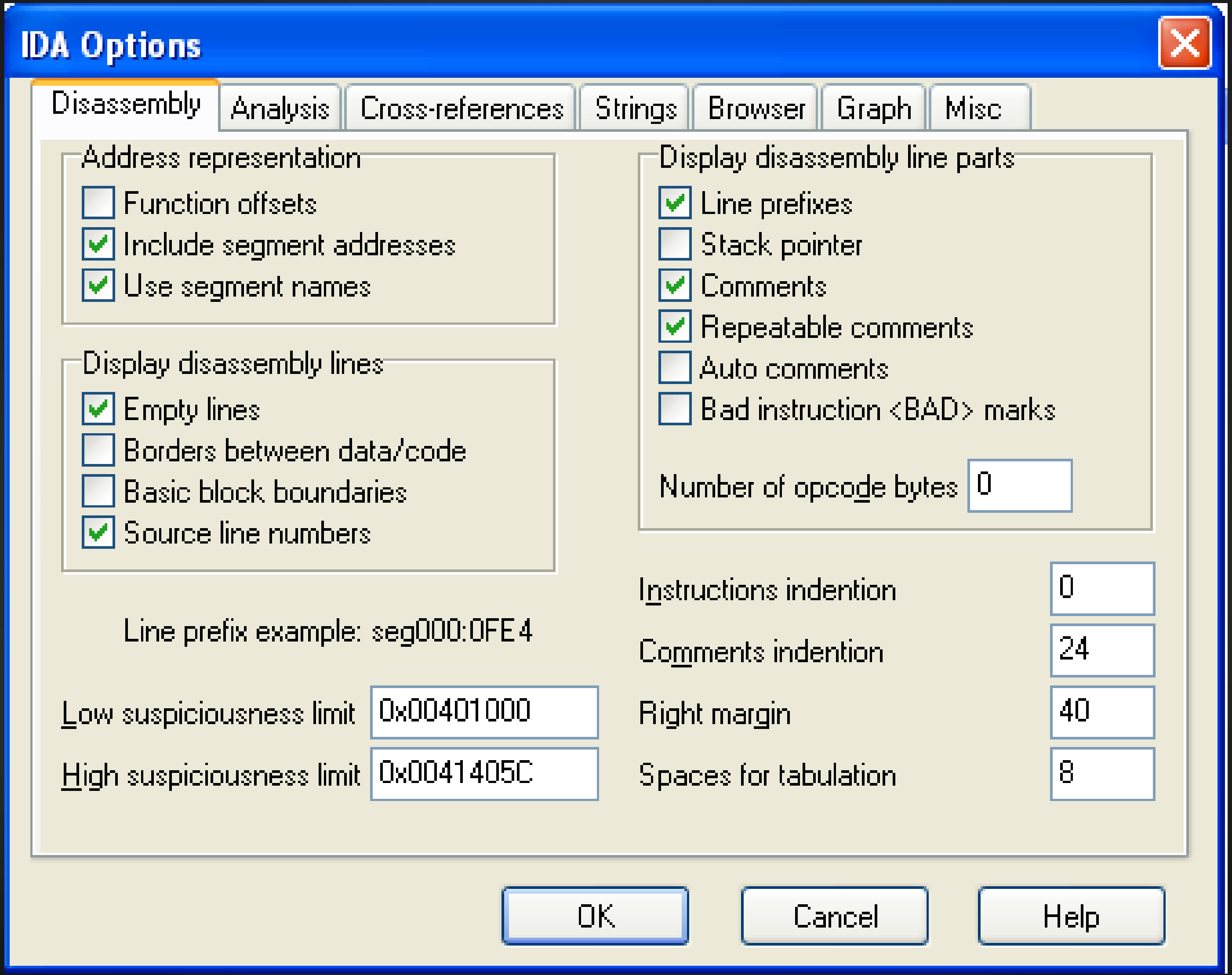


IDA View-A

```
.text:00401331                 dec     edx
.text:00401332                 aaa
.text:00401333                 xchg    eax, ebx
.text:00401334                 nop
.text:00401335                 inc     ecx
.text:00401336                 xchg    eax, edx
.text:00401337
.text:00401337 loc_401337:                             ; CODE XREF: .text:0040134B↓j
.text:00401337                 lahf
.text:00401338                 inc     edx
.text:00401339                 das
.text:0040133A                 setalc
.text:0040133B                 setalc
.text:0040133C                 setalc
.text:0040133D                 daa
.text:0040133E                 std
.text:0040133F                 jmp     loc_407E3A
.text:0040133F start           endp
.text:0040133F
.text:00401344 ; ---------------------------------------------------------------------------
.text:00401344                 setalc
.text:00401345                 mov     edi, [ebp-3Ch]
.text:00401348                 add     esp, 8
.text:0040134B                 jmp     short loc_401337
.text:0040134B ; ---------------------------------------------------------------------------
.text:0040134D                 db 5Fh, 0Dh, 60h
.text:00401350                 dd 33004030h, 83878A02h, 42048A11h, 3B08E083h, 0A8274C3h
.text:00401350                 dd 0EBFC0A89h, 0FFC983C8h, 0AEF2C033h, 5149D1F7h, 0A037E8h
.text:00401350                 dd 67004F00h, 34760000h, 40753468h, 3C6E800h, 0C4C6E800h
```

00001336   00401336: start+8F

IDA Pro – General Options Window



Address representation

- [ ] Function offsets
- [✓] Include segment addresses
- [✓] Use segment names

Display disassembly lines

- [✓] Empty lines
- [ ] Borders between data/code
- [ ] Basic block boundaries
- [✓] Source line numbers

Line prefix example: seg000:0FE4

Low suspiciousness limit: 0x00401000

High suspiciousness limit: 0x0041405C

Display disassembly line parts

- [✓] Line prefixes
- [ ] Stack pointer
- [✓] Comments
- [✓] Repeatable comments
- [ ] Auto comments
- [ ] Bad instruction <BAD> marks

Number of opcode bytes: 0

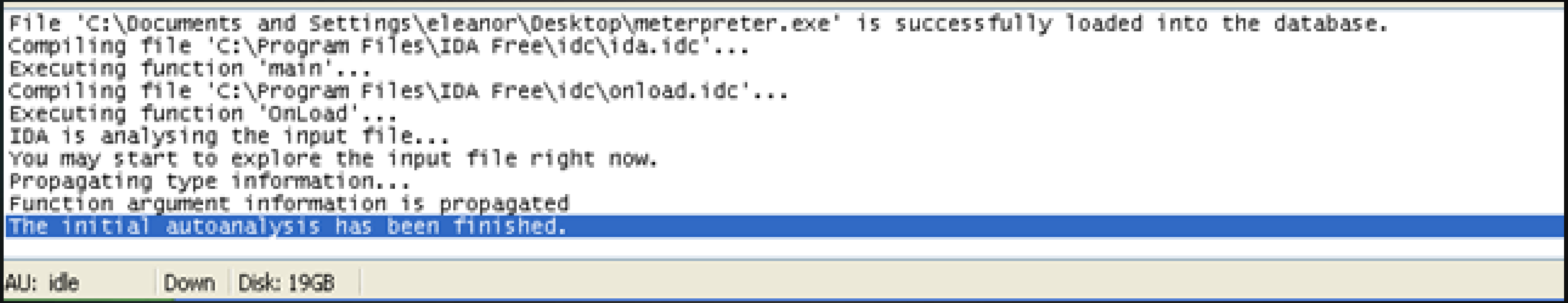
Instructions indention: 0

Comments indention: 24

Right margin: 40

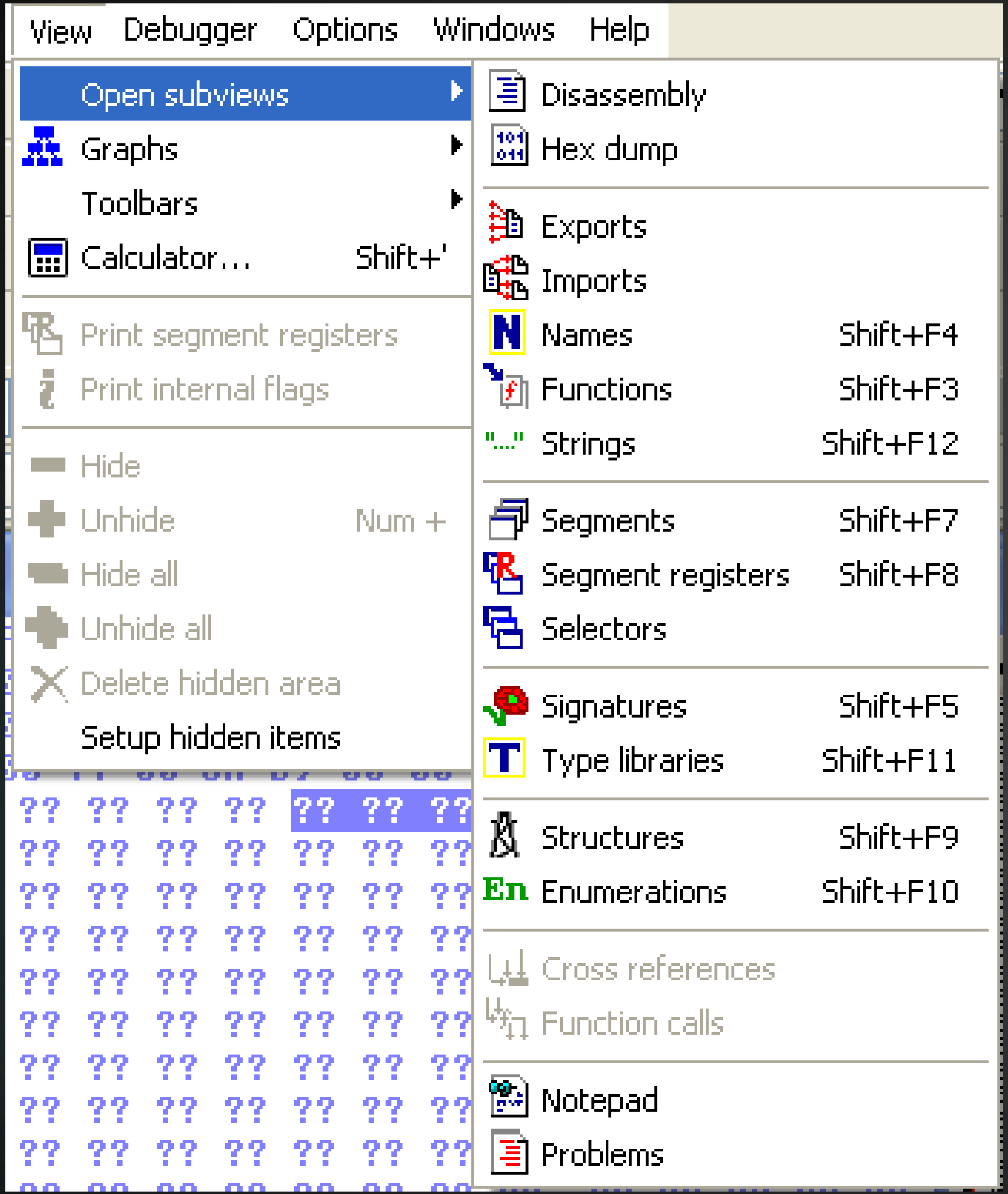
Spaces for tabulation: 8

In the IDA's default window, there's an additional window that is used to display different messages generated by IDA. Those messages can be outputted by any kind of plugin in IDA or by IDA itself. The messages are there to inform us of different things regarding the analysis of the executable sample. For clarity, the message view is presented below:



```
File 'C:\Documents and Settings\eleanor\Desktop\meterpreter.exe' is successfully loaded into the database.
Compiling file 'C:\Program Files\IDA Free\idc\ida.idc'...
Executing function 'main'...
Compiling file 'C:\Program Files\IDA Free\idc\onload.idc'...
Executing function 'OnLoad'...
IDA is analysing the input file...
You may start to explore the input file right now.
Propagating type information...
Function argument information is propagated
The initial autoanalysis has been finished.
```

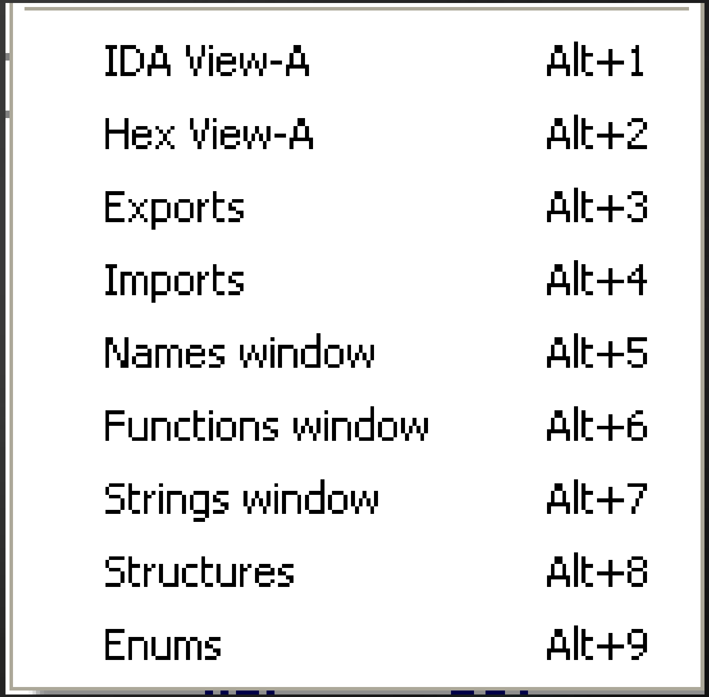
Other Views



View Debugger Options Windows Help

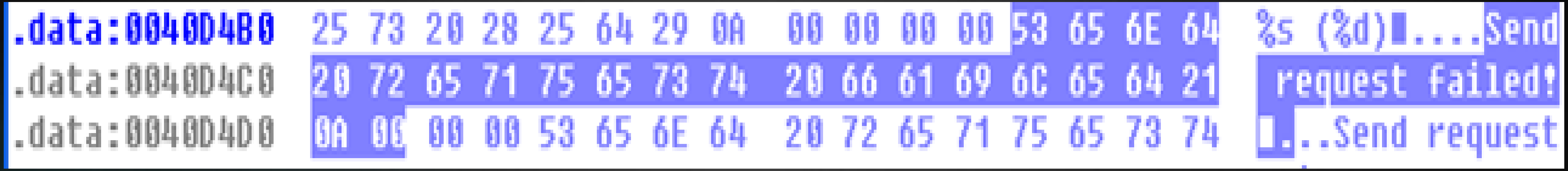
Open subviews	
Graphs	
Toolbars	
Calculator...	Shift+'
Print segment registers	
Print internal flags	
Hide	
Unhide	Num +
Hide all	
Unhide all	
Delete hidden area	
Setup hidden items	

Disassembly	
Hex dump	
Exports	
Imports	
Names	Shift+F4
Functions	Shift+F3
Strings	Shift+F12
Segments	Shift+F7
Segment registers	Shift+F8
Selectors	
Signatures	Shift+F5
Type libraries	Shift+F11
Structures	Shift+F9
Enumerations	Shift+F10
Cross references	
Function calls	
Notepad	
Problems	

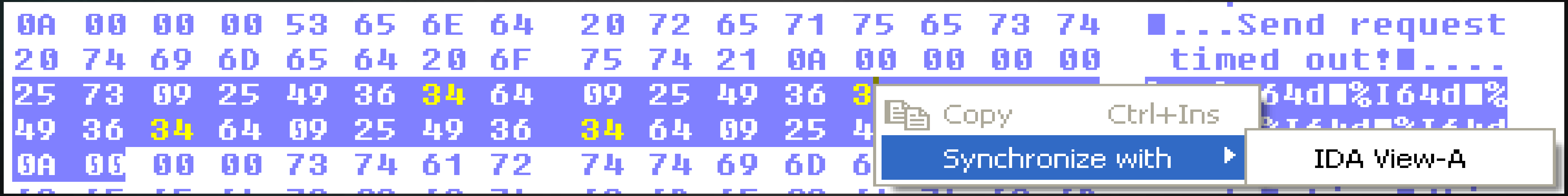


IDA View-A	Alt+1
Hex View-A	Alt+2
Exports	Alt+3
Imports	Alt+4
Names window	Alt+5
Functions window	Alt+6
Strings window	Alt+7
Structures	Alt+8
Enums	Alt+9

Hex View



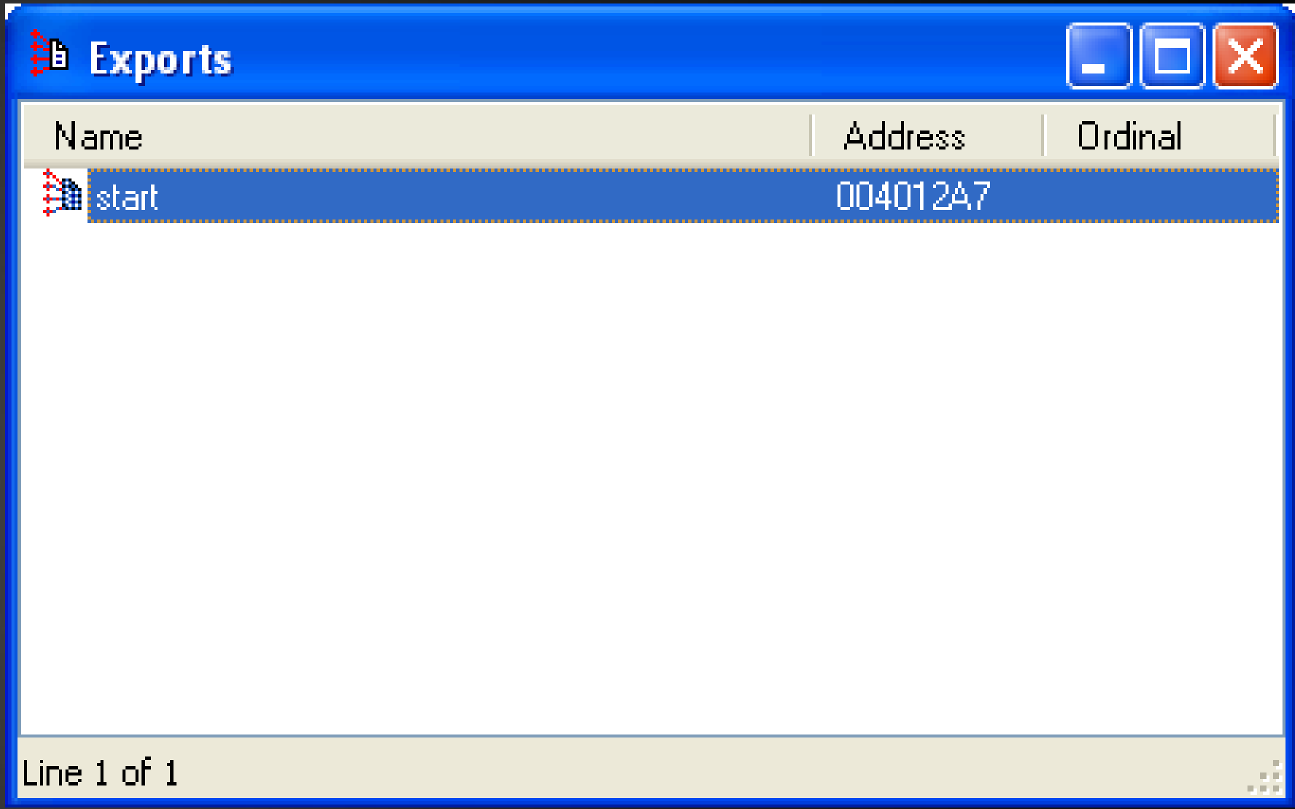
```
.data:0040D4B0  25 73 20 28 25 64 29 0A  00 00 00 00 53 65 6E 64  %s (%d)■....Send
.data:0040D4C0  20 72 65 71 75 65 73 74  20 66 61 69 6C 65 64 21   request failed!
.data:0040D4D0  0A 00 00 00 53 65 6E 64  20 72 65 71 75 65 73 74  ■...Send request
```



```
0A 00 00 00 53 65 6E 64  20 72 65 71 75 65 73 74  ■...Send request
20 74 69 6D 65 64 20 6F  75 74 21 0A 00 00 00 00   timed out!■....
25 73 09 25 49 36 34 64  09 25 49 36 3                 64d■%I64d■%
49 36 34 64 09 25 49 36  34 64 09 25 4
0A 00 00 00 73 74 61 72  74 74 69 6D 6
```

Exports

The Exports window lists the exported function that can be used by outside files. Exported functions are most common in shared libraries as they provide the basic building block APIs that can be used by programs running on the system to do basic operations.

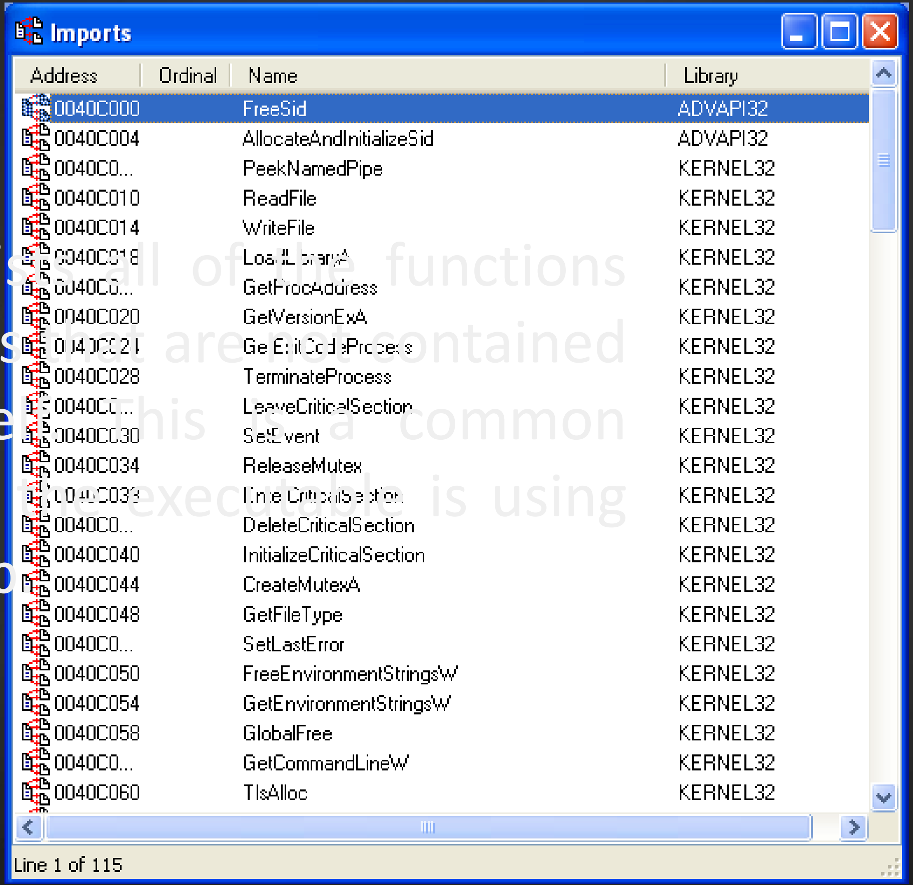

Exports

Name	Address	Ordinal
start	004012A7	

Line 1 of 1

Imports

The Imports window lists all of the functions that the executable calls that are not contained in the executable itself. This is a common scenario present when the executable is using shared DLLs to do its job.

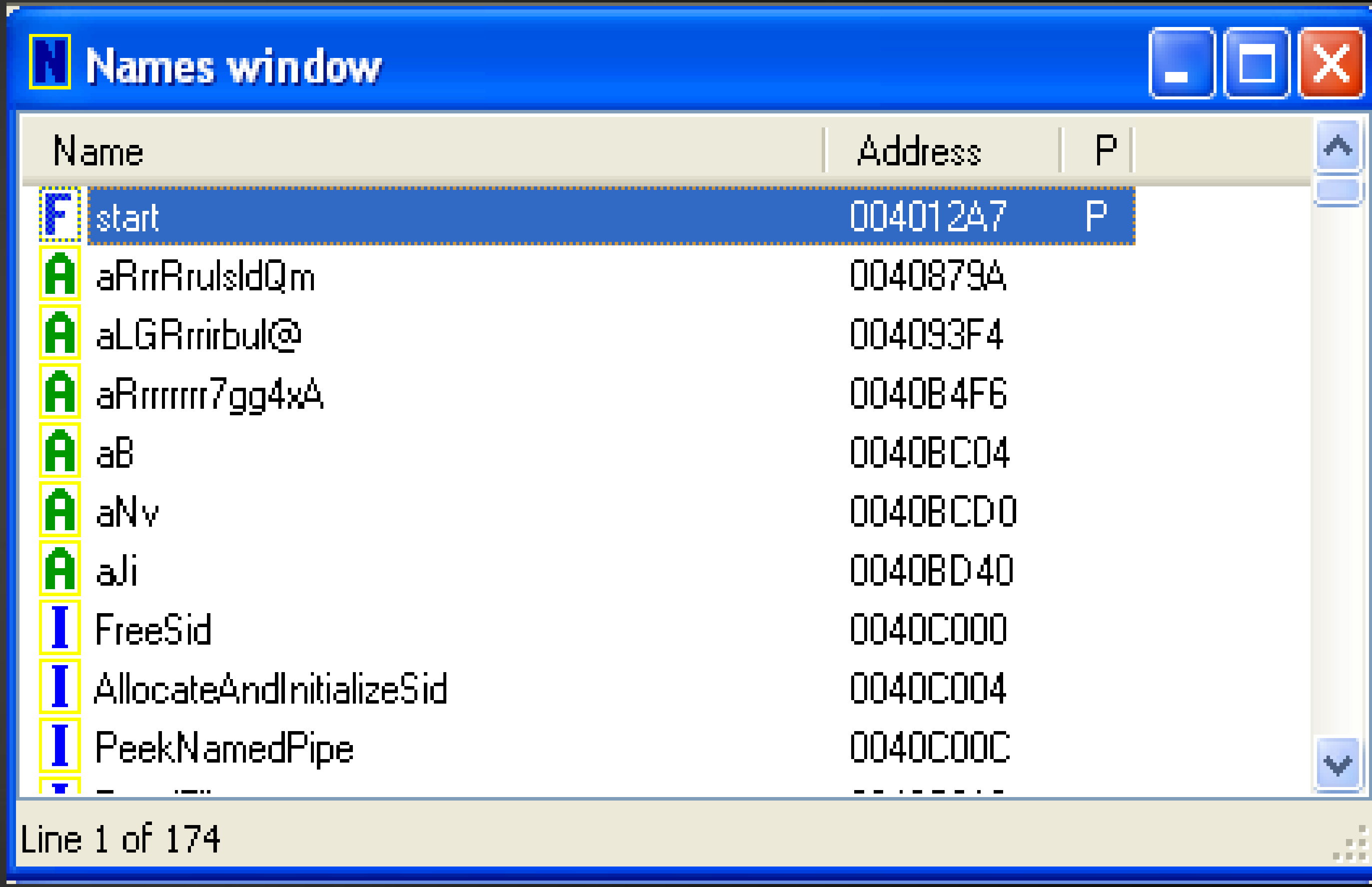

Imports

Address	Ordinal	Name	Library
0040C000		FreeSid	ADVAPI32
0040C004		AllocateAndInitializeSid	ADVAPI32
0040C0...		PeekNamedPipe	KERNEL32
0040C010		ReadFile	KERNEL32
0040C014		WriteFile	KERNEL32
0040C018		LoadLibraryA	KERNEL32
0040C0...		GetProcAddress	KERNEL32
0040C020		GetVersionExA	KERNEL32
0040C024		GetExitCodeProcess	KERNEL32
0040C028		TerminateProcess	KERNEL32
0040C0...		LeaveCriticalSection	KERNEL32
0040C030		SetEvent	KERNEL32
0040C034		ReleaseMutex	KERNEL32
0040C038		EnterCriticalSection	KERNEL32
0040C0...		DeleteCriticalSection	KERNEL32
0040C040		InitializeCriticalSection	KERNEL32
0040C044		CreateMutexA	KERNEL32
0040C048		GetFileType	KERNEL32
0040C0...		SetLastError	KERNEL32
0040C050		FreeEnvironmentStringsW	KERNEL32
0040C054		GetEnvironmentStringsW	KERNEL32
0040C058		GlobalFree	KERNEL32
0040C0...		GetCommandLineW	KERNEL32
0040C060		TlsAlloc	KERNEL32

Line 1 of 115

Names Window

The names window displays all the names found within the executable program. A name is simply an alias for a certain virtual address.

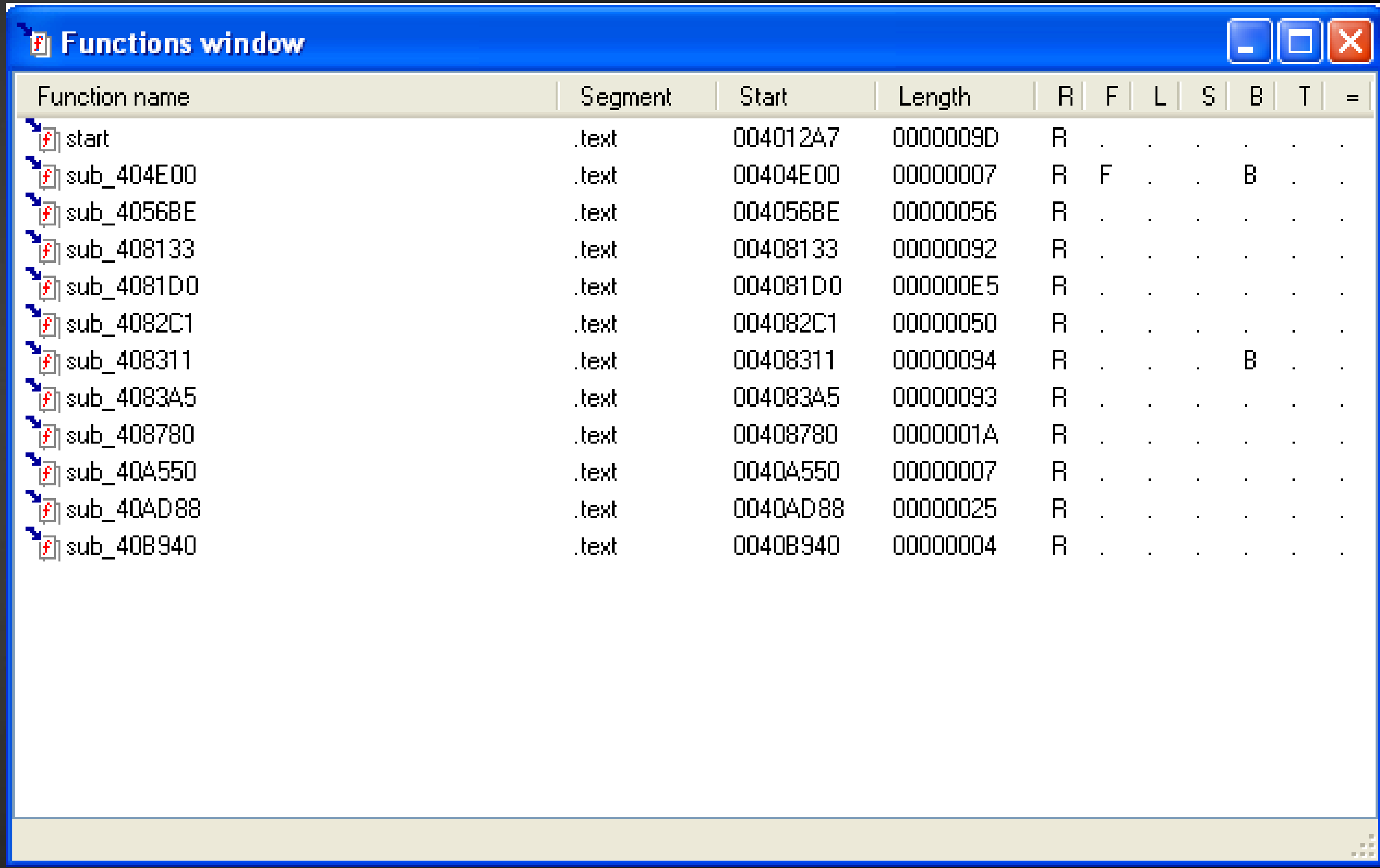

Names window

Name	Address	P
start	004012A7	P
aRrrRrulsldQm	0040879A	
aLGRrrirbul@	004093F4	
aRrrrrrr7gg4xA	0040B4F6	
aB	0040BC04	
aNv	0040BCD0	
aJi	0040BD40	
FreeSid	0040C000	
AllocateAndInitializeSid	0040C004	
PeekNamedPipe	0040C00C	

Line 1 of 174

Functions Window

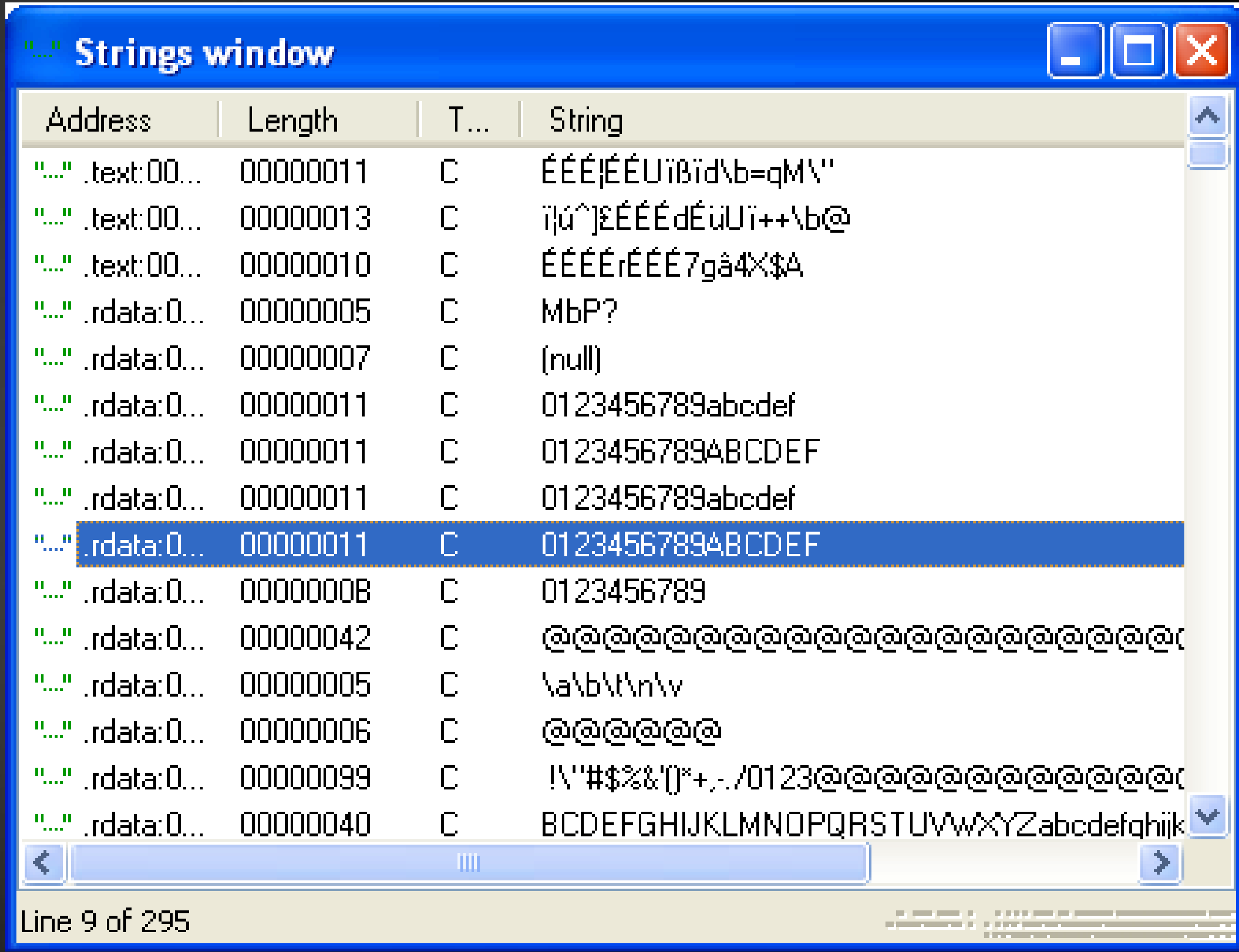
The functions window lists all the functions present in the executable, even though their name was automatically assigned by IDA itself.



Function name	Segment	Start	Length	R	F	L	S	B	T	=
start	.text	004012A7	0000009D	R	.	.	.	.	.	.
sub_404E00	.text	00404E00	00000007	R	F	.	.	B	.	.
sub_4056BE	.text	004056BE	00000056	R	.	.	.	.	.	.
sub_408133	.text	00408133	00000092	R	.	.	.	.	.	.
sub_4081D0	.text	004081D0	000000E5	R	.	.	.	.	.	.
sub_4082C1	.text	004082C1	00000050	R	.	.	.	.	.	.
sub_408311	.text	00408311	00000094	R	.	.	.	B	.	.
sub_4083A5	.text	004083A5	00000093	R	.	.	.	.	.	.
sub_408780	.text	00408780	0000001A	R	.	.	.	.	.	.
sub_40A550	.text	0040A550	00000007	R	.	.	.	.	.	.
sub_40AD88	.text	0040AD88	00000025	R	.	.	.	.	.	.
sub_40B940	.text	0040B940	00000004	R	.	.	.	.	.	.

Strings Window

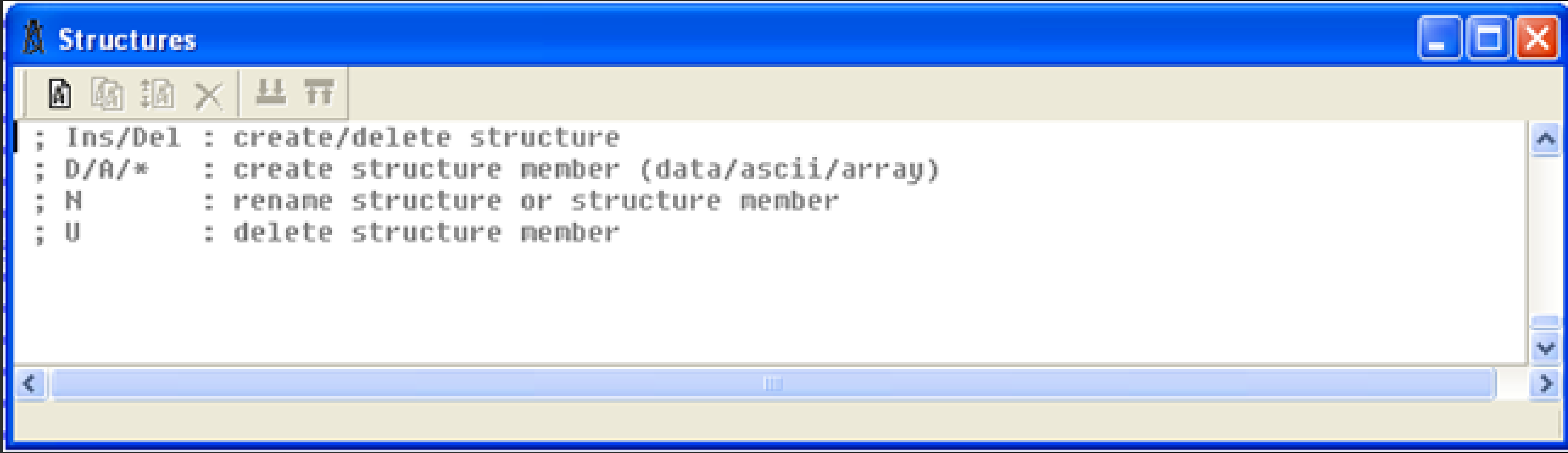
The stings window presents the strings that were found by the executable.


Strings window

Address	Length	T...	String
.text:00...	00000011	C	ÉÉÉ¦ÉÉUïßïd\b=qM\"
.text:00...	00000013	C	ï¦ú^]£ÉÉÉdÉüUï++\b@
.text:00...	00000010	C	ÉÉÉÉrÉÉÉ7gâ4X$A
.rdata:0...	00000005	C	MbP?
.rdata:0...	00000007	C	(null)
.rdata:0...	00000011	C	0123456789abcdef
.rdata:0...	00000011	C	0123456789ABCDEF
.rdata:0...	00000011	C	0123456789abcdef
.rdata:0...	00000011	C	0123456789ABCDEF
.rdata:0...	0000000B	C	0123456789
.rdata:0...	00000042	C	@@@@@@@@@@@@@@@@@@@@@
.rdata:0...	00000005	C	\a\b\t\n\v
.rdata:0...	00000006	C	@@@@@@
.rdata:0...	00000099	C	!\"#$%&'()*+,-./0123@@@@@@@@@@@@
.rdata:0...	00000040	C	BCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijk

Line 9 of 295

Structures

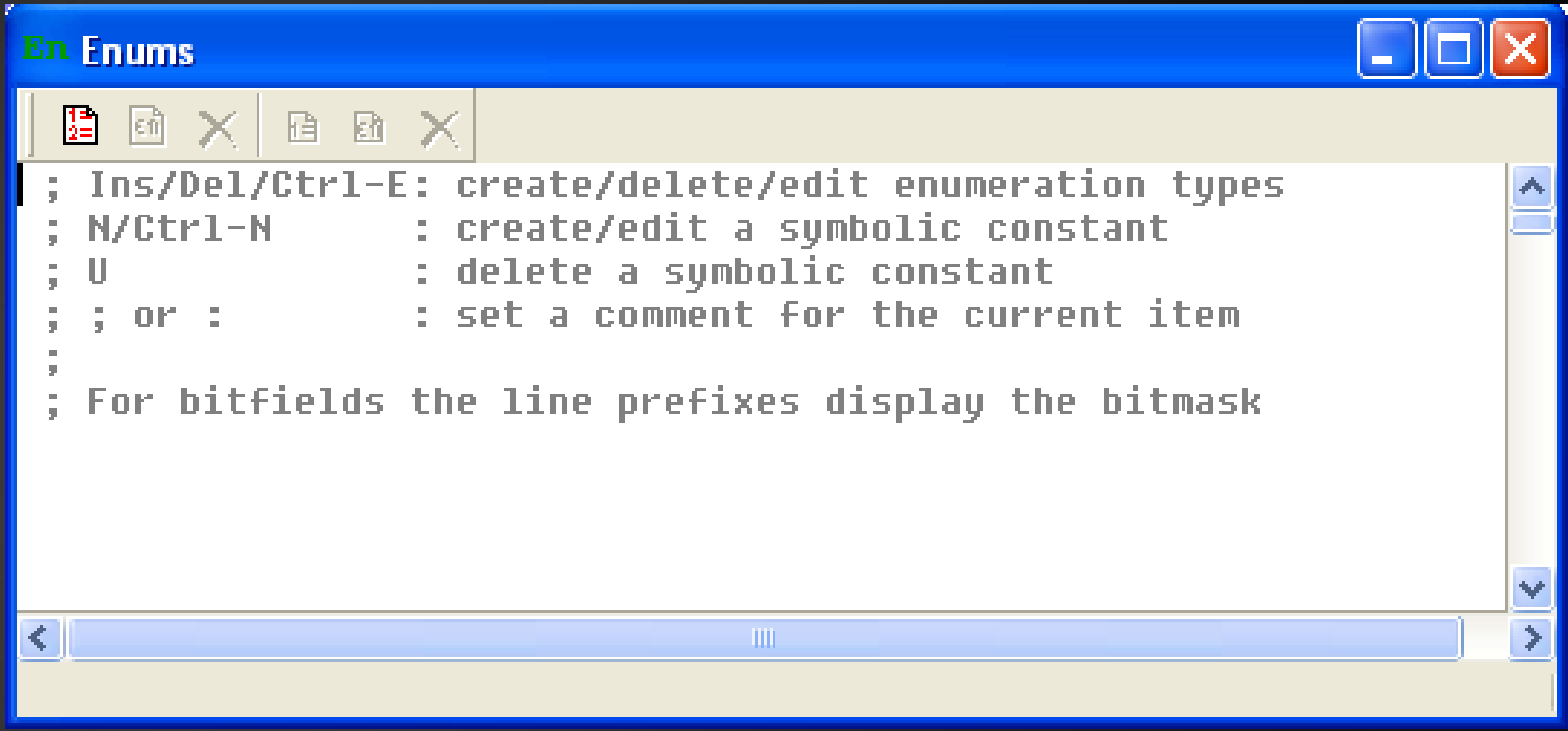


```
; Ins/Del  : create/delete structure
; D/A/*    : create structure member (data/ascii/array)
; N        : rename structure or structure member
; U        : delete structure member
```

The structures window lists the data structures that could be found in the binary. IDA uses the functions and their known arguments to figure out whether there’s a data structure present in the executable or not.

Enums

The enums window lists all the enum data types found in the executable.



```
; Ins/Del/Ctrl-E: create/delete/edit enumeration types
; N/Ctrl-N        : create/edit a symbolic constant
; U               : delete a symbolic constant
; ; or :          : set a comment for the current item
;
; For bitfields the line prefixes display the bitmask
```

What today’s presentation is all about

- ~~Introduction – What Is Reverse Engineering?~~
- ~~What Is IDA Pro?~~
- ~~Delving Into IDA Pro~~
- **Practical Demo of Bypassing Security Check Using IDA Pro**
- Introduction to Patching
- Practical Demo On Patching a Binary
- Basic Static Malware Analysis Using IDA Pro
- Conclusion
- References

What today's presentation is all about

- ~~Introduction – What Is Reverse Engineering?~~
- ~~What Is IDA Pro?~~
- ~~Delving Into IDA Pro~~
- ~~Practical Demo of Bypassing Security Check Using IDA Pro~~
- **Introduction to Patching**
- Practical Demo On Patching a Binary
- Basic Static Malware Analysis Using IDA Pro
- Conclusion
- References

What is Software Patching?

- A **patch** is a piece of **software** designed to update a computer program or its supporting data, to ***"fix or improve"*** it. This includes fixing security vulnerabilities and other bugs, with such **patches** usually called bug fixes, and improving the usability or performance.

- Patching involves in modifying binary code by identifying the buggy segments through disassembling the code.

Patching as a Hack

Once a vulnerability is identified it can be exploited and the binary can be subsequently patched to create the modification to execution that the hacker has intended for it to do

What today's presentation is all about

- ~~Introduction – What Is Reverse Engineering?~~
- ~~What Is IDA Pro?~~
- ~~Delving Into IDA Pro~~
- ~~Practical Demo of Bypassing Security Check Using IDA Pro~~
- ~~Introduction to Patching~~
- **Practical Demo On Patching a Binary**
- Basic Static Malware Analysis Using IDA Pro
- Conclusion
- References

What today's presentation is all about

- ~~Introduction – What Is Reverse Engineering?~~
- ~~What Is IDA Pro?~~
- ~~Delving Into IDA Pro~~
- ~~Practical Demo of Bypassing Security Check Using IDA Pro~~
- ~~Introduction to Patching~~
- ~~Practical Demo On Patching a Binary~~
- **Basic Static Malware Analysis Using IDA Pro**
- Conclusion
- References

What today's presentation is all about

- ~~Introduction – What Is Reverse Engineering?~~
- ~~What Is IDA Pro?~~
- ~~Delving Into IDA Pro~~
- ~~Practical Demo of Bypassing Security Check Using IDA Pro~~
- ~~Introduction to Patching~~
- ~~Practical Demo On Patching a Binary~~
- ~~Basic Static Malware Analysis Using IDA Pro~~
- **Conclusion**
- References

Conclusion

- IDA Pro is a well-known tool for malware reversal and forensic behavioral analysis of binary executables.
- IDA performs automatic code analysis, using cross-references between code sections, knowledge of parameters of API calls, and other information.
- IDA has interactive functionality to aid in improving the disassembly.
- However, the nature of disassembly precludes total accuracy, and a great deal of human intervention is necessarily required.

What today's presentation is all about

- ~~Introduction – What Is Reverse Engineering?~~
- ~~What Is IDA Pro?~~
- ~~Delving Into IDA Pro~~
- ~~Practical Demo of Bypassing Security Check Using IDA Pro~~
- ~~Introduction to Patching~~
- ~~Practical Demo On Patching a Binary~~
- ~~Basic Static Malware Analysis Using IDA Pro~~
- ~~Conclusion~~
- **References**

References

1. IDA Pro Documentation - https://www.hex-rays.com/products/ida/support/tutorials/index.shtml
2. Christodorescu, Mihai ; Jha, Somesh: Static Analysis of Executables to Detect Malicious Patterns, Wisconsin University, Madison, Department of Computer Sciences (2006)
3. Malware Analysis - https://blog.malwarebytes.com/security-world/2012/09/so-you-want-to-be-a-malware-analyst/
4. Malware Analysis Report - https://zeltser.com/malware-analysis-report/
5. Writing Malware Analysis Report - https://digital-forensics.sans.org/blog/2012/05/08/writing-malware-reports
6. Intro to IDA Pro - http://resources.infosecinstitute.com/basics-of-ida-pro-2/#gref

Thank You