Reverse Engineering Report

By

Udhayashankar Palanivel[ME-SNS]

Index:

Exam_1.exe:

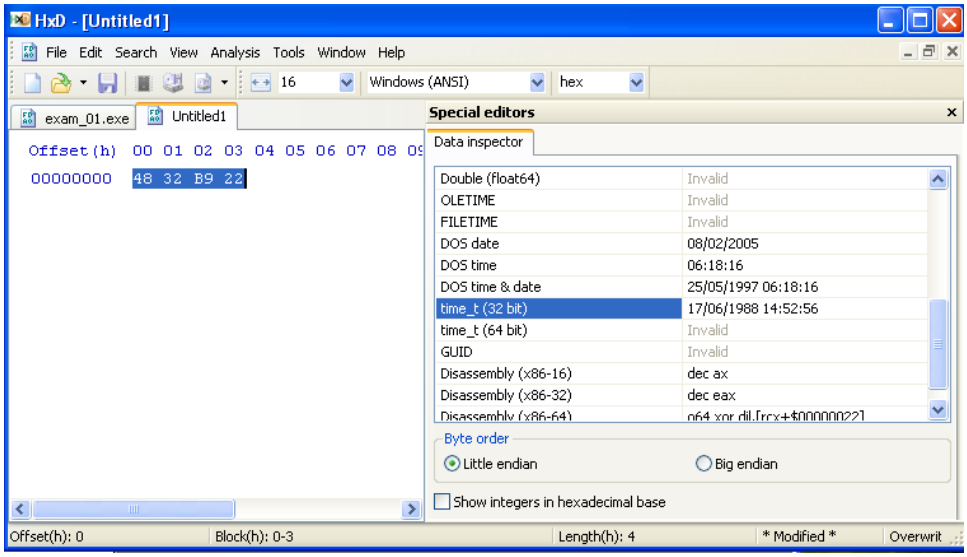
Compiler used:

Microsoft Visual C++ 8 was used for compiling.

Property	Value
File Name	C:\Documents and Settings\Administrateur\Bureau\exam_01.exe
File Type	Portable Executable 32
File Info	Microsoft Visual C++ 8
File Size	44.00 KB (45056 bytes)
PE Size	44.00 KB (45056 bytes)
Created	Saturday 03 November 2018, 23.29.44
Modified	Saturday 03 November 2018, 23.13.54
Accessed	Saturday 03 November 2018, 23.29.44
MD5	58CF91EE026DEBE0D59FF35C6EF2422F
SHA-1	9285D7ABC9E3A7161FF228D205360D10C84F122E

Application compiled Time:

The application was compiled on 17/06/1988 14:52:56.



This could be found under the section File Header as TimetoStamp.

The corresponding value to Timetostamp can be decoded using HEXeditor in order to obtain the time when the application was complied.

SHA-1 value for the Application:

The SHA-1 value for the application is 9285D7ABC9E3A7161FF228D205360D10C84F122E.

Property	Value
File Name	C:\Documents and Settings\Administrateur\Bureau\exam_01.exe
File Type	Portable Executable 32
File Info	Microsoft Visual C++ 8
File Size	44.00 KB (45056 bytes)
PE Size	44.00 KB (45056 bytes)
Created	Saturday 03 November 2018, 23.29.44
Modified	Saturday 03 November 2018, 23.13.54
Accessed	Saturday 03 November 2018, 23.29.44
MD5	58CF91EE026DEBE0D59FF35C6EF2422F
SHA-1	9285D7ABC9E3A7161FF228D205360D10C84F122E

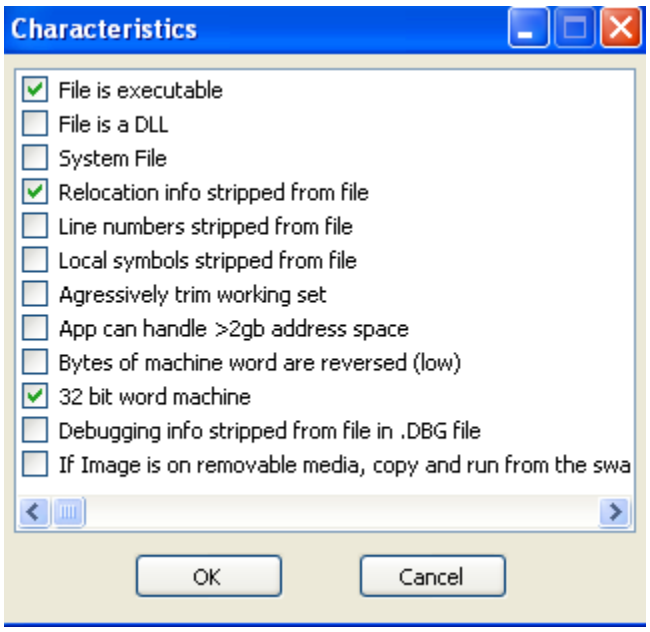
The value of SHA1 can be obtained from File Section as SHA1.

CPU platform used for compilation:

32-bit version was used for compilation.

Member	Offset	Size	Value	Meaning
Machine	000000E4	Word	014C	Intel 386
NumberOfSections	000000E6	Word	0003	
TimeDateStamp	000000E8	Dword	4832B922	
PointerToSymbolTable	000000EC	Dword	00000000	
NumberOfSymbols	000000F0	Dword	00000000	
SizeOfOptionalHeader	000000F4	Word	00E0	
Characteristics	000000F6	Word	0103	Click here

The CPU platform used for compilation can be found in two ways. One way is it can find under File header under Nt headers as Intel 386 in Machine Column.



Second way is in the Characteristics column you can click on the “CLICK here” to obtain the system Information's.

Entry point:

The address of entry point is 00001681 and it is located in File under Nt Headers in Optional Header as AddressOfEntryPoint of 00001681 in .text section.

Member	Offset	Size	Value	Meaning
Magic	000000F8	Word	010B	PE32
MajorLinkerVersion	000000FA	Byte	08	
MinorLinkerVersion	000000FB	Byte	00	
SizeOfCode	000000FC	Dword	00007000	
SizeOfInitializedData	00000100	Dword	00004000	
SizeOfUninitializedData	00000104	Dword	00000000	
AddressOfEntryPoint	00000108	Dword	00001681	.text
BaseOfCode	0000010C	Dword	00001000	
BaseOfData	00000110	Dword	00008000	
ImageBase	00000114	Dword	00400000	
SectionAlignment	00000118	Dword	00001000	
FileAlignment	0000011C	Dword	00001000	
MajorOperatingSystemVersion	00000120	Word	0004	
MinorOperatingSystemVersion	00000122	Word	0000	
MajorImageVersion	00000124	Word	0000	
MinorImageVersion	00000126	Word	0000	
MajorSubsystemVersion	00000128	Word	0004	
MinorSubsystemVersion	0000012A	Word	0000	
Win32VersionValue	0000012C	Dword	00000000	

The corresponding value of AddressOfEntryPoint gives the address of Entry point.

Sections in Application:

There are three sections in the application.

These sections can be found in File under Section Headers (.text, .rdata and .data).

Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumb
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword
.text	00006704	00001000	00007000	00001000	00000000	00000000
.rdata	00001F4A	00008000	00002000	00008000	00000000	00000000
.data	000018DC	0000A000	00001000	0000A000	00000000	00000000

These sections can be found under Section headers.

Application packed with UPX:

No, the application is not packed with UPX because the sections do not, they contain the names. UPX0 or .UPX1.

Compressed/Packed Section:

No, the difference between the Raw size and Virtual size of each section are almost equal.

Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumb
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword
.text	00006704	00001000	00007000	00001000	00000000	00000000
.rdata	00001F4A	00008000	00002000	00008000	00000000	00000000
.data	000018DC	0000A000	00001000	0000A000	00000000	00000000

Imported libraries:

They are four sections imported from the system.
(GDI32.dll,USER32.dll,KERNEL32.dll,ADVAPI32.dll)

Module Name	Imports	OFTs	TimeDateStamp	ForwarderChain	Name RVA
szAnsi	(nFunctions)	Dword	Dword	Dword	Dword
GDI32.dll	1	00009878	00000000	00000000	000099C8
USER32.dll	13	0000997C	00000000	00000000	00009AA6
KERNEL32.dll	62	00009880	00000000	00000000	00009ADA
ADVAPI32.dll	1	00009870	00000000	00000000	00009AF8

For the first library CreateSolidBrush API was imported.

Module Name	Imports	OFTs	TimeDateStamp	ForwarderChain	Name RVA
000099C8	N/A	0000980C	00009810	00009814	00009818
szAnsi	(nFunctions)	Dword	Dword	Dword	Dword
GDI32.dll	1	00009878	00000000	00000000	000099C8
USER32.dll	13	0000997C	00000000	00000000	00009AA6
KERNEL32.dll	62	00009880	00000000	00000000	00009ADA
ADVAPI32.dll	1	00009870	00000000	00000000	00009AF8

OFTs	FTs (IAT)	Hint	Name
Dword	Dword	Word	szAnsi
000099B4	000099B4	0050	CreateSolidBrush

For the second library they are several libraries imported some of them are LoadIconA, LoadCursorA and so on.

Module Name	Imports	OFTs	TimeDateStamp	ForwarderChain	Name RVA
00009AA6	N/A	00009820	00009824	00009828	0000982C
szAnsi	(nFunctions)	Dword	Dword	Dword	Dword
GDI32.dll	1	00009878	00000000	00000000	000099C8
USER32.dll	13	0000997C	00000000	00000000	00009AA6
KERNEL32.dll	62	00009880	00000000	00000000	00009ADA
ADVAPI32.dll	1	00009870	00000000	00000000	00009AF8

OFTs	FTs (IAT)	Hint	Name
Dword	Dword	Word	szAnsi
00009A9A	00009A9A	01BE	LoadIconA
00009A8C	00009A8C	01BA	LoadCursorA
00009A7E	00009A7E	015A	GetSysColor
00009A6A	00009A6A	0217	RegisterClassExA
00009A58	00009A58	0060	CreateWindowExA

The third library contains GetStringTypeA and son on.

Module Name	Imports	OFTs	TimeDateStamp	ForwarderChain	Name RVA
00009ADA	N/A	00009834	00009838	0000983C	00009840
szAnsi	(nFunctions)	Dword	Dword	Dword	Dword
GDI32.dll	1	00009878	00000000	00000000	000099C8
USER32.dll	13	0000997C	00000000	00000000	00009AA6
KERNEL32.dll	62	00009880	00000000	00000000	00009ADA
ADVAPI32.dll	1	00009870	00000000	00000000	00009AF8

OFTs	FTs (IAT)	Hint	Name
Dword	Dword	Word	szAnsi
00009F26	00009F26	01BA	GetStringTypeA
00009F16	00009F16	0245	LCMapStringW
00009F06	00009F06	0244	LCMapStringA
00009EF4	00009EF4	0174	GetLocaleInfoA
00009AB2	00009AB2	01EB	GetVolumeInformationA

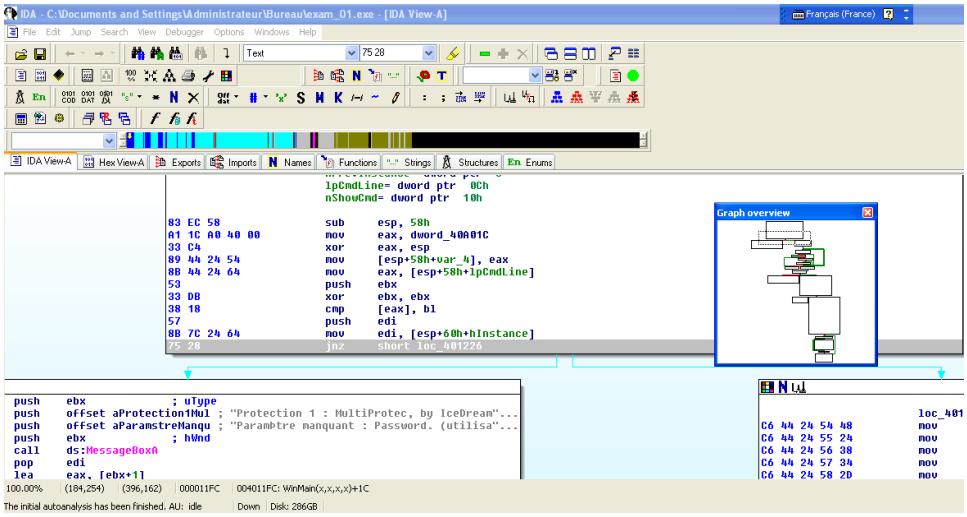
And the last one contains RegOpenKeyExA.

Module Name	Imports	OFTs	TimeDateStamp	ForwarderChain	Name RVA
00009AF8	N/A	00009848	0000984C	00009850	00009854
szAnsi	(nFunctions)	Dword	Dword	Dword	Dword
GDI32.dll	1	00009878	00000000	00000000	000099C8
USER32.dll	13	0000997C	00000000	00000000	00009AA6
KERNEL32.dll	62	00009880	00000000	00000000	00009ADA
ADVAPI32.dll	1	00009870	00000000	00000000	00009AF8

OFTs	FTs (IAT)	Hint	Name
Dword	Dword	Word	szAnsi
00009AE8	00009AE8	01EC	RegOpenKeyExA

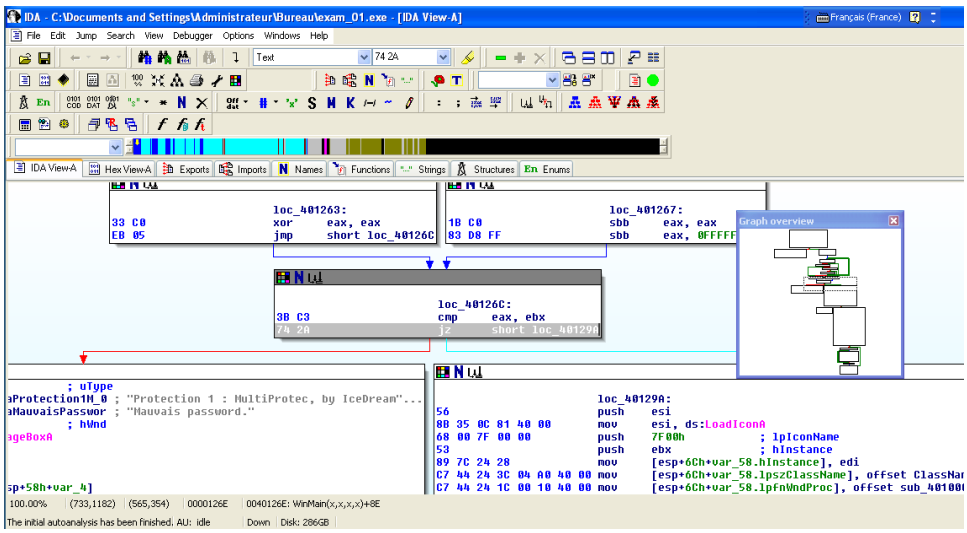
Patching:

With all the information's obtained from CFF Explorer, now .exe file is imported to IDA to understand how the applications works by going through the source code.



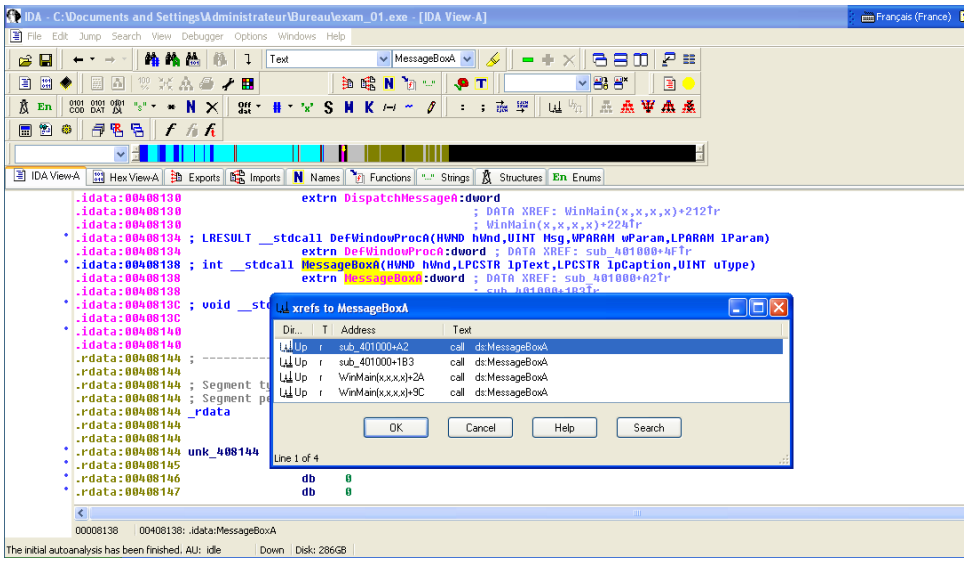
We can enable the opcodes to have a good understanding by going through Options->General->and entering the number of opcodes that we want.

At 000011FC position we change the value from 75(Jump if zero) to 74(Jump if not equal to zero) in order to access the other side of the block.



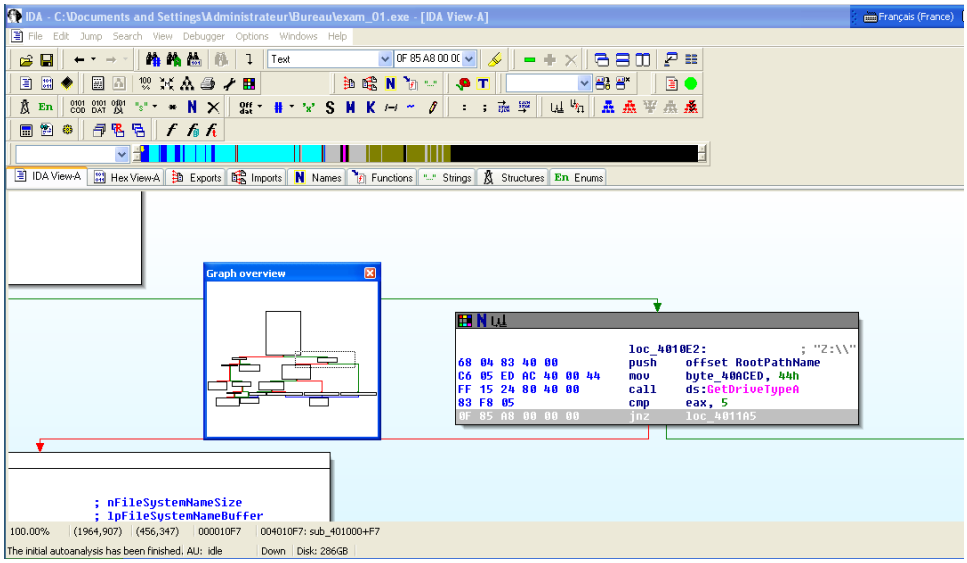
Proceeding the code at position 0000126E the value is changed from 75 to 74 to enter the other side.

We can decide to alter the value because the box contains Mauvais Password which can lead us to the password.

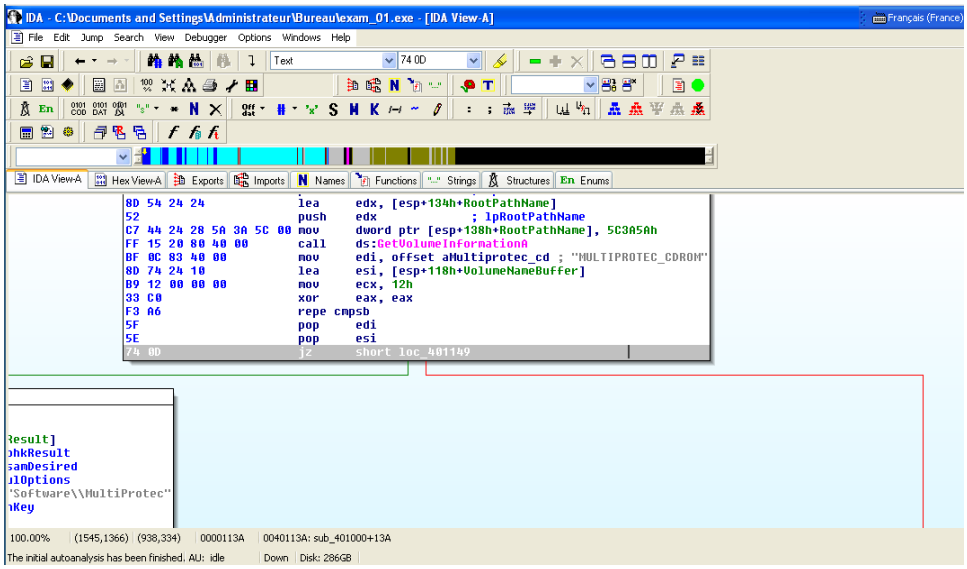


In the above figure right after the “Mauvais password” we found a message box which can give some interesting information's.

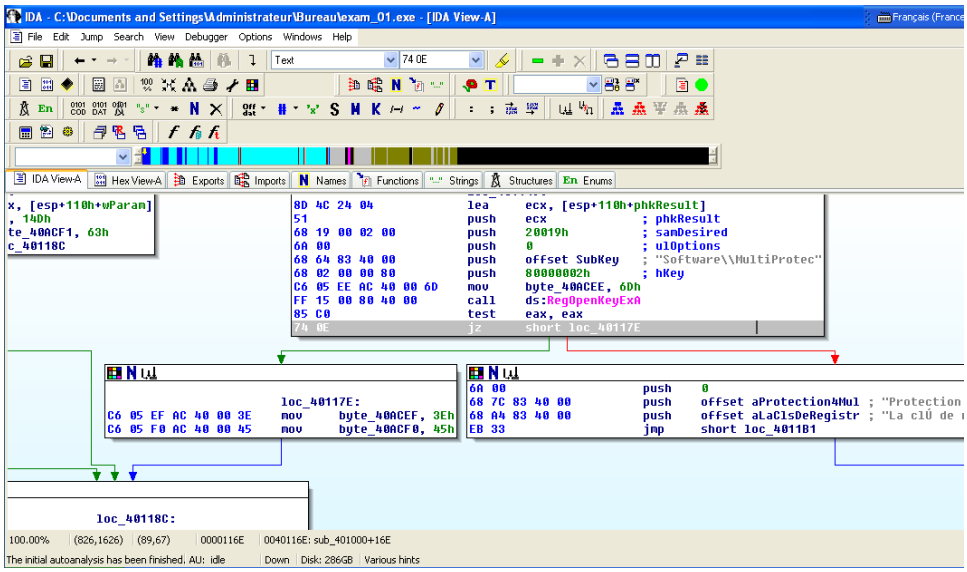
In order to get in to Messagebox we can double click and will lead to another page with full of information's. In the page when you right click Meassgebox it may give some options choose to (jump to x operand) and it will take to you the right position of the code.



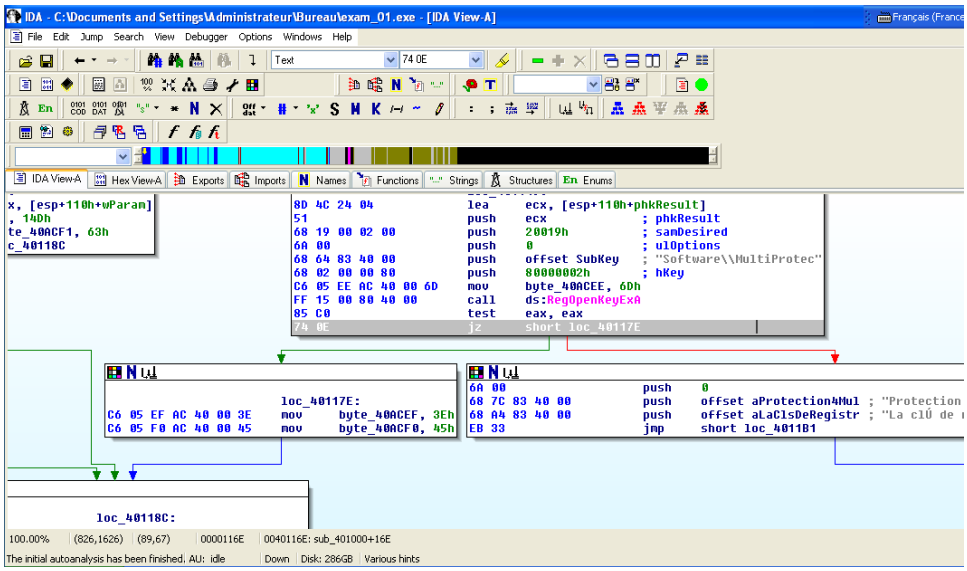
As you can see this led us to the other function. Under this function we can could see the string “LE PASS ET:” below by doing some patching we can obtain the password.



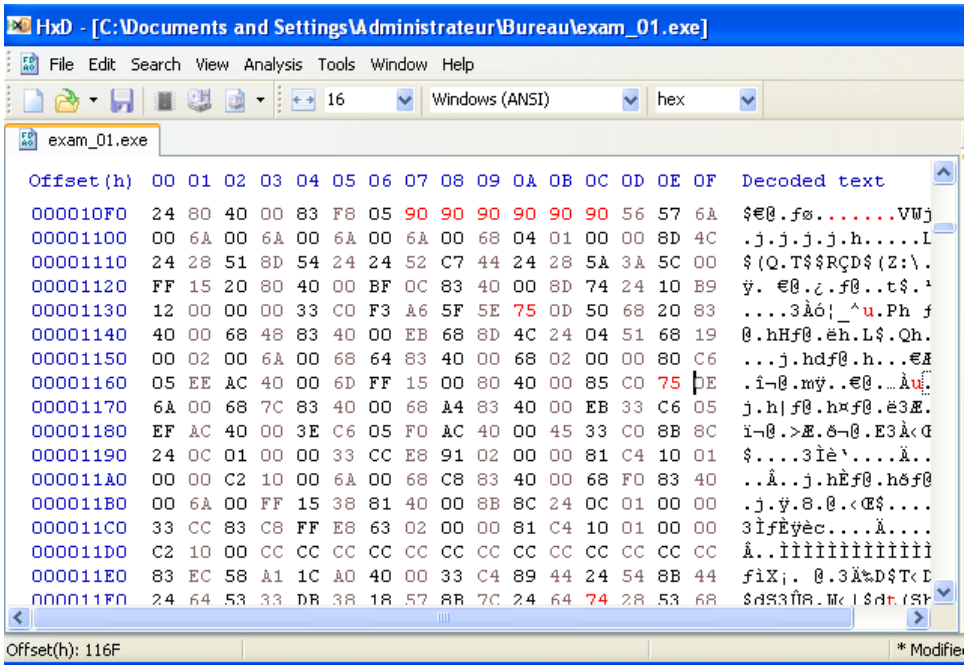
So, first we start patching at position 00001027 by replacing 90(nop) values to all six positions as mentioned in IDA.



Next, we then change the value from 75 to 74 at offset 0113A.

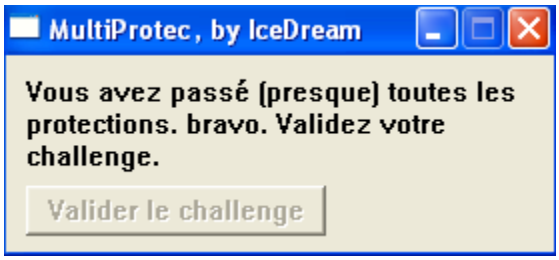


And finally, at offset 0000116E we change the value from 75 to 74. With all these patching we should too able to the Valider le challenge window box.

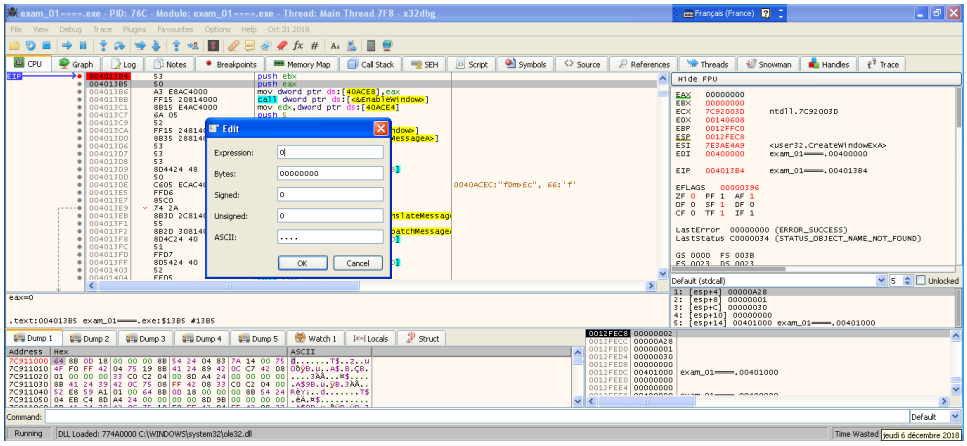


We then open the application using HEXeditor with all offsets noted from IDA we use HEXeditor to alter the values.

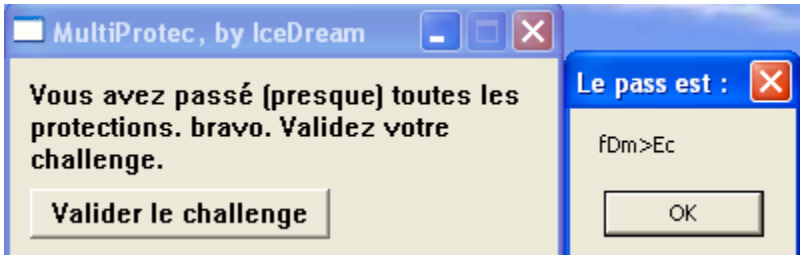
After altering save the file and run it. A window appears with Valider le challenge. But still the button is disabled to get the password.



So, we use x64 debugger to understand the program step by step in order to analyze and patch at the right place to obtain the password.



As per the information's obtained from IDA we could see the offset “Enable the button” string at 4013B4(000013B4). So, we use this offset to keep the breakpoint because we need to change the value after that offset.



At offset 4013B5 we change the value of eax to 0 which means enabling the “Valider le challenge” button.

Once its enabled if click it will display the password.

2.Exam_2.exe:

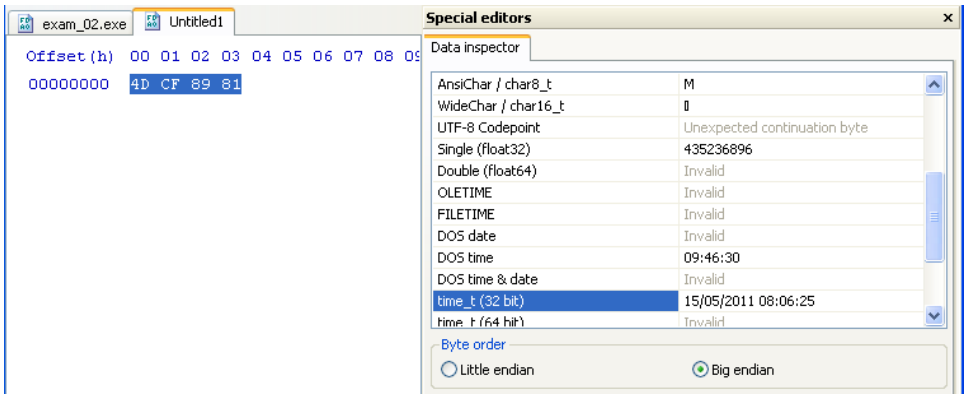
Compiler used:

Microsoft Visual C++ was used for compiling. This could be found in File Section.

Property	Value
File Name	C:\Documents and Settings\Administrateur\Mes documents\Téléchargemen...
File Type	Portable Executable 32
File Info	Microsoft Visual C++
File Size	184.10 KB (188514 bytes)
PE Size	184.00 KB (188416 bytes)
Created	Sunday 04 November 2018, 22.01.13
Modified	Sunday 04 November 2018, 22.01.15
Accessed	Sunday 04 November 2018, 22.00.59
MD5	56BCBF49770698FC57D4BC7FD4821825
SHA-1	363BC426C70E3729A9AB89AFE1D6BCF6964FD941

Application compiled Time:

The application was compiled on 15/05/2011 08:06:25.



This could be found under the section File Header as TimetoStamp. The corresponding value to Timetostamp can be decoded using HEXeditor in order to obtain the time when the application was complied.

SHA-1 value for the Application:

The SHA-1 value for the application is 363BC426C70E3729A9AB89AFE1D6BCF6964FD941.

Property	Value
File Name	C:\Documents and Settings\Administrateur\Mes documents\Téléchargemen...
File Type	Portable Executable 32
File Info	Microsoft Visual C++
File Size	184.10 KB (188514 bytes)
PE Size	184.00 KB (188416 bytes)
Created	Sunday 04 November 2018, 22.01.13
Modified	Sunday 04 November 2018, 22.01.15
Accessed	Sunday 04 November 2018, 22.00.59
MD5	56BCBF49770698FC57D4BC7FD4821825
SHA-1	363BC426C70E3729A9AB89AFE1D6BCF6964FD941

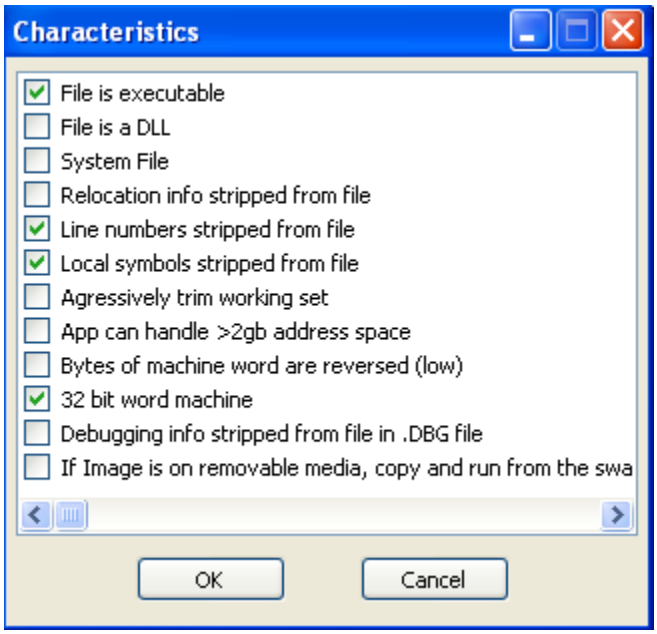
The value of SHA1 can be obtained from File Section as SHA1.

CPU platform used for compilation:

32-bit version was used for compilation.

Member	Offset	Size	Value	Meaning
Machine	000000D4	Word	014C	Intel 386
NumberOfSections	000000D6	Word	0005	
TimeDateStamp	000000D8	Dword	4DCF8981	
PointerToSymbolTable	000000DC	Dword	00000000	
NumberOfSymbols	000000E0	Dword	00000000	
SizeOfOptionalHeader	000000E4	Word	00E0	
Characteristics	000000E6	Word	010E	Click here

The CPU platform used for compilation can be found in two ways. One way is it can find under File header under Nt headers as Intel 386 in Machine Column.



Second way is in the Characteristics column you can click on the “CLICK here” to obtain the system Information's.

Entry point:

The address of entry point is 00004700 and it is located under Nt Headers in Optional Header as AddressOfEntryPoint.

Member	Offset	Size	Value	Meaning
Magic	000000E8	Word	010B	PE32
MajorLinkerVersion	000000EA	Byte	06	
MinorLinkerVersion	000000EB	Byte	00	
SizeOfCode	000000EC	Dword	00026000	
SizeOfInitializedData	000000F0	Dword	00009000	
SizeOfUninitializedData	000000F4	Dword	00000000	
AddressOfEntryPoint	000000F8	Dword	00004700	.text
BaseOfCode	000000FC	Dword	00001000	
BaseOfData	00000100	Dword	00001000	
ImageBase	00000104	Dword	00400000	
SectionAlignment	00000108	Dword	00001000	
FileAlignment	0000010C	Dword	00001000	
MajorOperatingSystemVersion	00000110	Word	0004	
MinorOperatingSystemVersion	00000112	Word	0000	
MajorImageVersion	00000114	Word	0000	
MinorImageVersion	00000116	Word	0000	
MajorSubsystemVersion	00000118	Word	0004	
MinorSubsystemVersion	0000011A	Word	0000	
Win32VersionValue	0000011C	Dword	00000000	

The corresponding value of AddressOfEntryPoint gives the address of Entry point.

Sections in the application:

There are Five sections in the application. Such as (.text, .rdata, idata, .reloc and .data).

Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumb
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword
.text	00025950	00001000	00026000	00001000	00000000	00000000
.rdata	00001CF0	00027000	00002000	00027000	00000000	00000000
.data	000034F0	00029000	00002000	00029000	00000000	00000000
.idata	00000757	0002D000	00001000	0002B000	00000000	00000000
.reloc	000010B1	0002E000	00002000	0002C000	00000000	00000000

These sections can be found under Section headers.

Application packed with UPX:

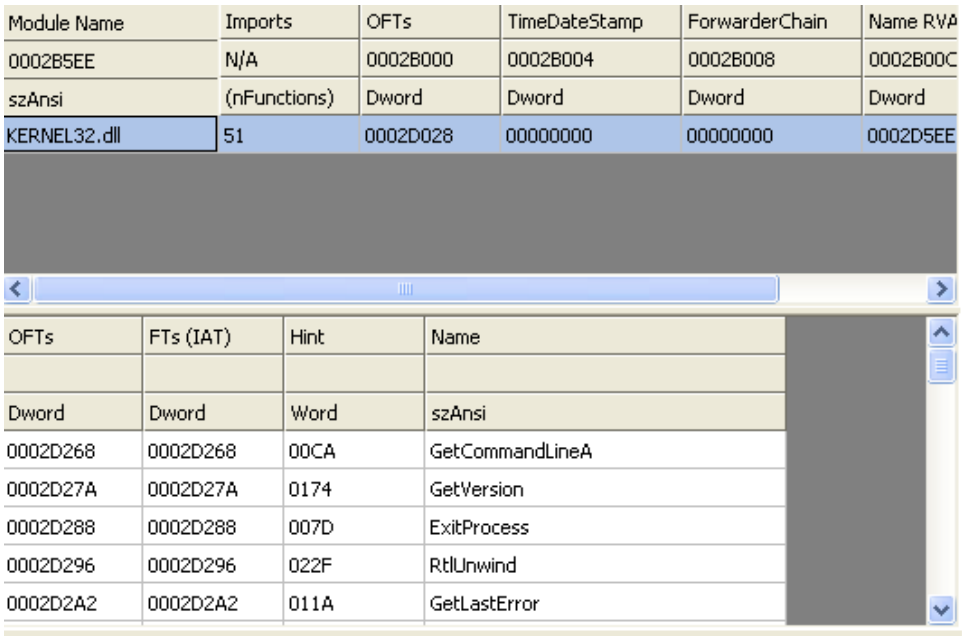
No, the application is not packed with UPX because the sections do not, they contain the names. UPX0 or .UPX1.

Compressed/packed section:

No, the difference between the Raw size and Virtual size of each section are almost equal as per the image given above.

Imported libraries:

For this application only KERNEL32.dll is been imported.



Module Name	Imports	OFTs	TimeDateStamp	ForwarderChain	Name RVA
0002B5EE	N/A	0002B000	0002B004	0002B008	0002B00C
szAnsi	(nFunctions)	Dword	Dword	Dword	Dword
KERNEL32.dll	51	0002D028	00000000	00000000	0002D5EE

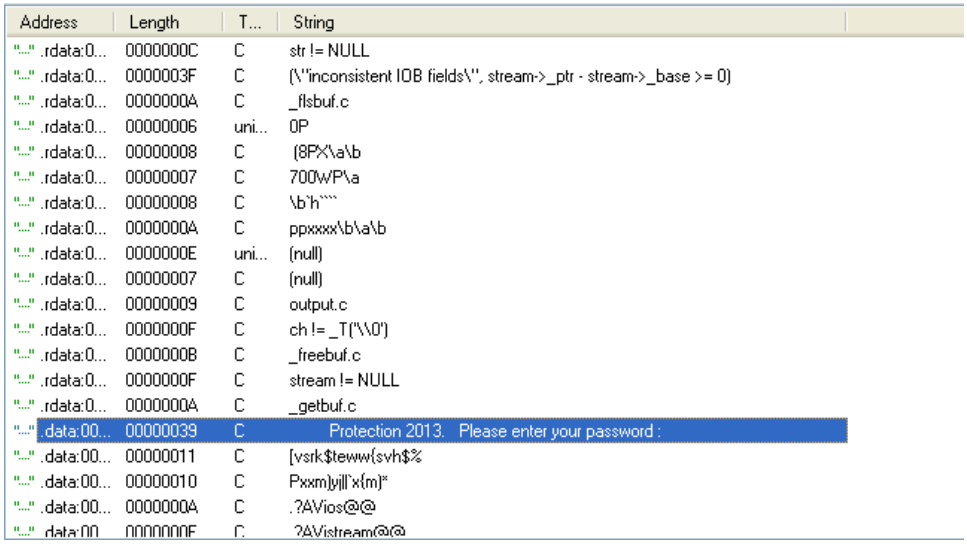
OFTs	FTs (IAT)	Hint	Name
Dword	Dword	Word	szAnsi
0002D268	0002D268	00CA	GetCommandLineA
0002D27A	0002D27A	0174	GetVersion
0002D288	0002D288	007D	ExitProcess
0002D296	0002D296	022F	RtlUnwind
0002D2A2	0002D2A2	011A	GetLastError

The Imported Section can be found in Import Directory.

As per the image there are several API's called and some of them are GetCommandLineA, GetVersion, ExitProcess, RtlUnwind, GetLastError and so on.

Patching:

The exam_2.exe file in opened using IDA software in order to view the structure of the complied assembly program which gives the flow of the executable file.

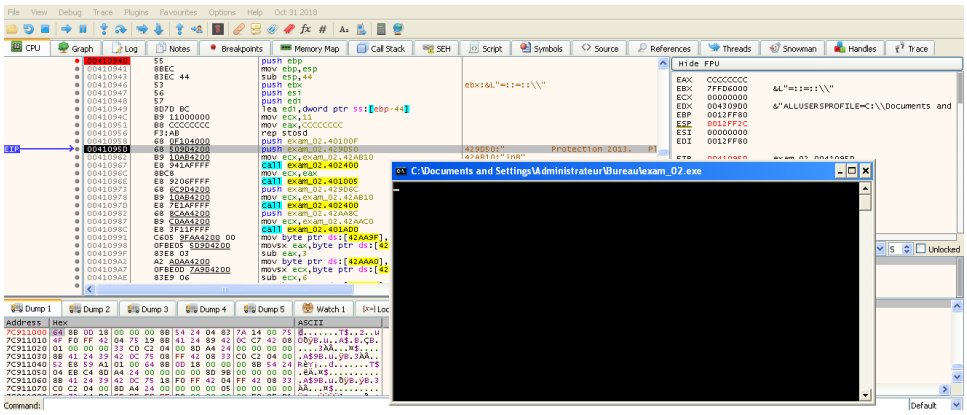


Address	Length	T...	String
.rdata:0...	0000000C	C	str != NULL
.rdata:0...	0000003F	C	(\"inconsistent IOB fields\", stream->_ptr - stream->_base >= 0)
.rdata:0...	0000000A	C	_flsbuf.c
.rdata:0...	00000006	uni...	0P
.rdata:0...	00000008	C	(8PX\a\b
.rdata:0...	00000007	C	700WP\a
.rdata:0...	00000008	C	\b`h```
.rdata:0...	0000000A	C	ppxxxx\b\a\b
.rdata:0...	0000000E	uni...	(null)
.rdata:0...	00000007	C	(null)
.rdata:0...	00000009	C	output.c
.rdata:0...	0000000F	C	ch != _T('\\0')
.rdata:0...	0000000B	C	_freebuf.c
.rdata:0...	0000000F	C	stream != NULL
.rdata:0...	0000000A	C	_getbuf.c
.data:00...	00000039	C	Protection 2013. Please enter your password :
.data:00...	00000011	C	[vsrk$teww{svh$%
.data:00...	00000010	C	Pxxm}yjll`x{m}*
.data:00...	0000000A	C	.?AVios@@
.data:00...	0000000E	C	.?AVistream@@

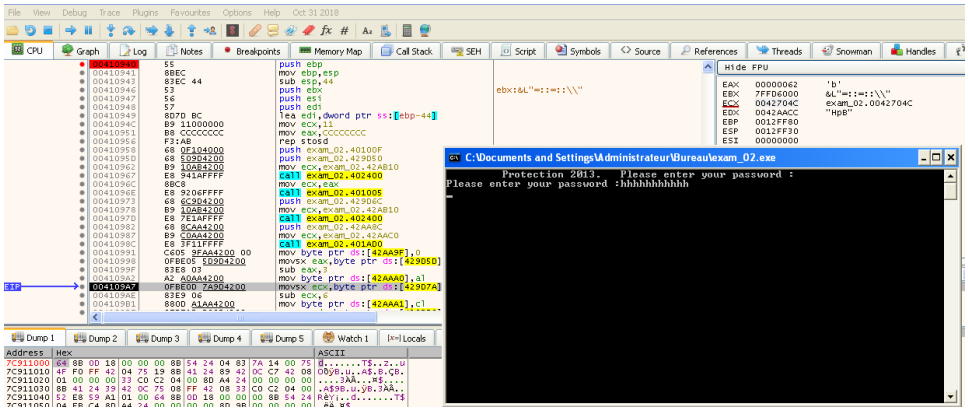
Using the IDA software, we can search for a string that has a key which can direct it to the location of that string.

When the mouse is pointed towards the string it gives the virtual address of the program which can be used in X64 debugger for debugging.

The x64 debugger is opened and initially we run the file which executes the executable file as per the image shown.



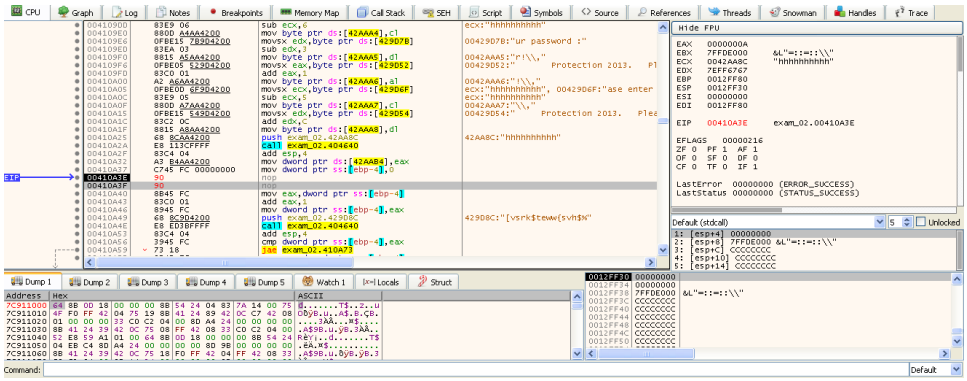
Next using the virtual address which was obtained from IDA can used to locate the string by giving the offset in the debugger.



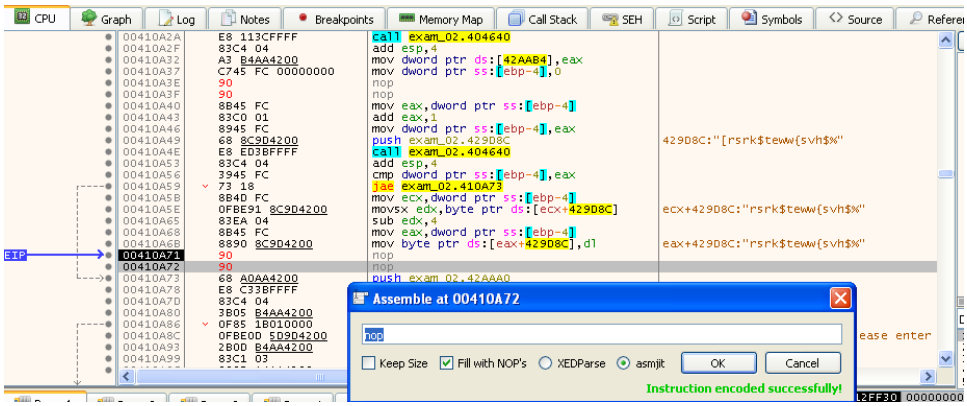
A breakpoint is been set in order the debug the code by step by step execution which allows you to interact with the program.

At the point 410A3E jmp values has been replaced with nop values in order to terminate the jump command.

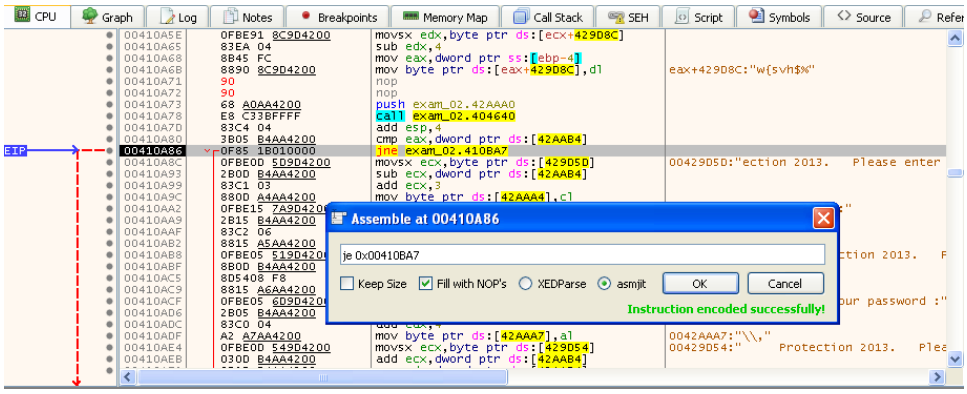
The values can be altered by right clicking and then selecting Assemble.



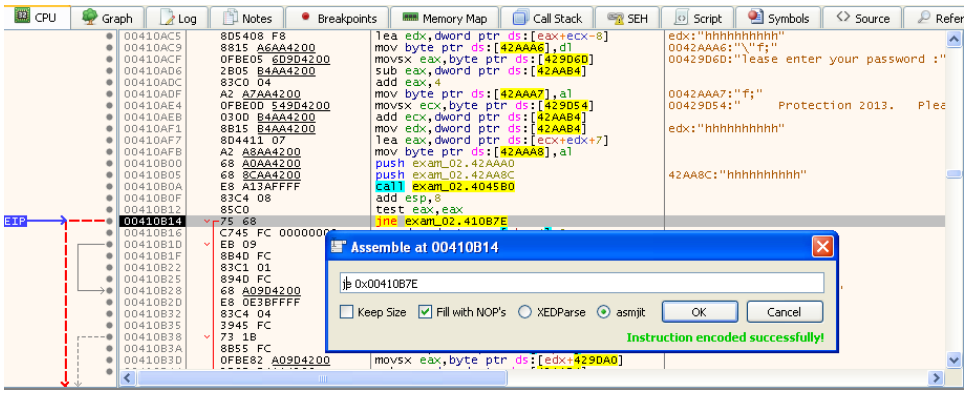
At the point 410A59 again values are replaced with nop values in order to execute the fore coming commands.



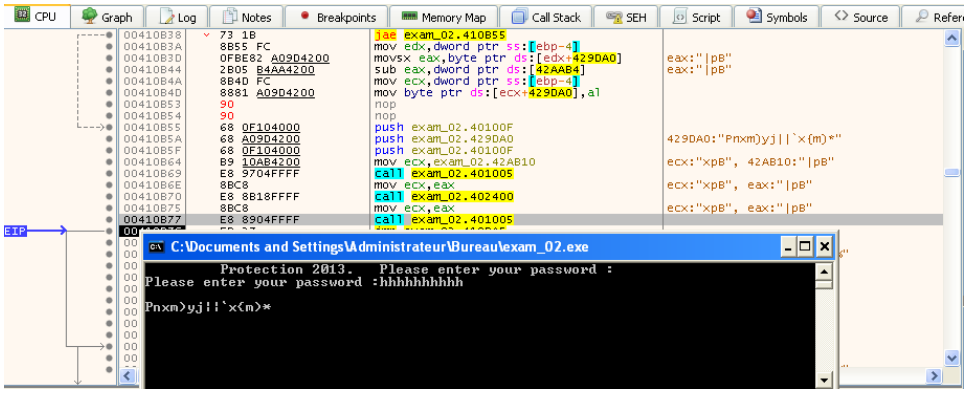
After that at 410A86 the jump if not equal(jne) has been replaced with jump if equal(je) because if not doing this will lead to the termination of the program.



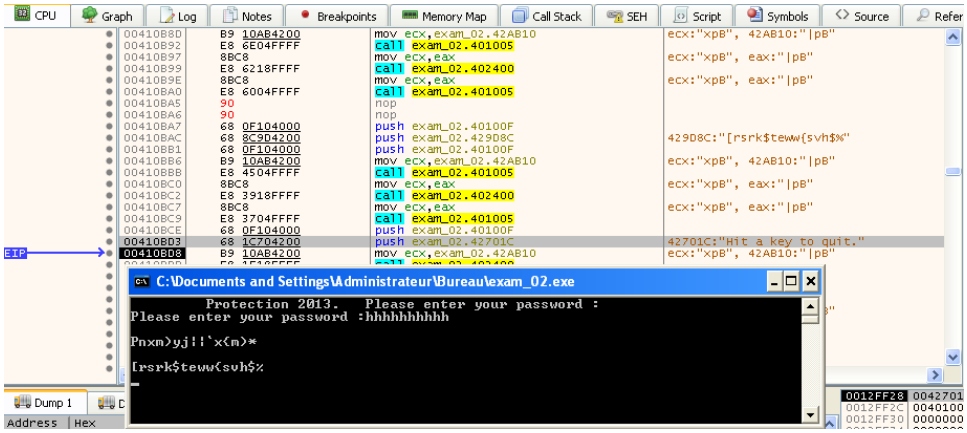
With the flow of command, the given password can be noticed. At offset 410B14 the jne value is been replaced with je command.



After executing at offset 410B77 a string appears on the terminal.



Again, at offset 410BC9 other string appear on the terminal and finally leads to hit a key to quit.



The strings that appear on the screen represents the right answer but are in encrypted form.

The encrypted form can be decrypted using the offsets taken from debugger. While running the application you can input the offset file which will decrypt and display the answer.