Black-0ut Frenzy – [B)(F] Security Researcher Center
B0Frenzy.freehostia.com

PE Infection – How to Inject a dll

In association with

www.mihanit.net

Thank you to my friends who help me in this research

(K053,Heli, L U C I F E R(Bl4ck_Ic3))

Author: Nightmare(BioHazard)

Date: 03.05.2009(88.02.13)

Hint : Only the reader is responsible for any abuse of context of this document.

Nightmare(BioHazard) – K053
Great IRAN

Introduction:

Our goal is to infect a Portable Executable file with a DLL. For that we need to obtain information on the structure PE.

So first we will discuss about PE Structure which is contain *PE Sections* & *Add a Section(Manually & Automaticly).*

Then we have a brief discussion of the *Import Table.*

Finally, we will discuss how to inject a DLL

PE section(s):

00300000	0002C000			
00400000	00001000	firefox(		PE header
00401000	00001000	firefox(	.text	code
00402000	00001000	firefox(	.rdata	imports
00403000	00001000	firefox(	.data	data
00404000	00048000	firefox(	.rsrc	resources
0044C000	00001000	firefox(	.reloc	relocations
00450000	00009000			
00490000	00009000			
00550000	00002000			
00560000	00103000			
00670000	00104000			
00970000	00001000			
009F0000	00003000			
00A00000	00008000			
00A10000	00002000			
00A50000	00004000			
00B50000	00001000	swpg		PE header
00B51000	00012000	swpg	CODE	code
00B63000	00001000	swpg	DATA	data
00B64000	00001000	swpg	BSS	
00B65000	00001000	swpg	.idata	imports
00B66000	00001000	swpg	.reloc	relocations
00B67000	00001000	swpg	.rsrc	resources
00B70000	00004000			
01210000	00002000			
5A000000	00001000	klg		PE header
5A001000	00012000	klg	CODE	code
5A013000	00001000	klg	DATA	data
5A014000	00001000	klg	BSS	
5A015000	00001000	klg	.idata	imports
5A016000	00001000	klg	.reloc	relocations
5A017000	00001000	klg	.rsrc	resources

As you can see a Portable Executable contains *".text , .rdata , .data , .rsrc , reloc , BSS"* are PE sections which has been generated by windows loader.

"*.text*" stores main code of a PE.

"*.rdata*" contains read only data such as String literals , Debug Directory & .....

"*.data*" stores all static data and initialized global.

"*BSS*" contains all uninitialized global.

"*.rsrc & .reloc*" store details for relocating the image while loading

Now we need to know how to add a section into a PE. At first I'd like to do it in a hard way that needs a little Patience , some neurons in your head and a good Hex-Editor **J**.

First we have to gather the last section's information, so open our target*(firefox* **J** *)* in a hex editor program and find *.reloc*.

So that the first 8 bytes for the name of the section. The next 4bytes are shown is Virtual Size. The next 4bytes are virtual address. The next 4bytes are Raw Size. The next 4bytes are Raw Offset then we have 12bytes which is nulled , then 4bytes for flags. And one important point is that our section alignment is 1000h.

Now we will add a section with 1000h Vsize and Rsize(it's not related to section alignment). Now we need to calculate our new section's information.

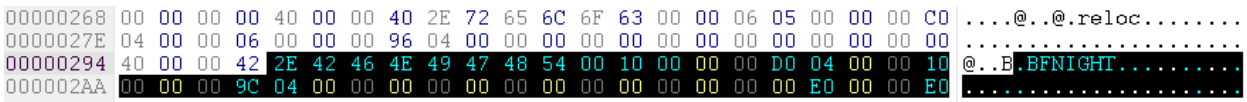
New Virtual Address = Virtual Address + Virtual Size + Section Alignment
// = 0004C000 + 00000506 + 1000 = 0004D506 -> 06 D5 04 00 (We use nearer 0004D000 > 00 D0 04 00)

New Raw Offset = Raw Offset + Raw Size
// = 00049600 + 600 = 00049C00 -> 00 9C 04 00

Virtual Size and Raw size for the new section = 1000h -> 00 10 00 00

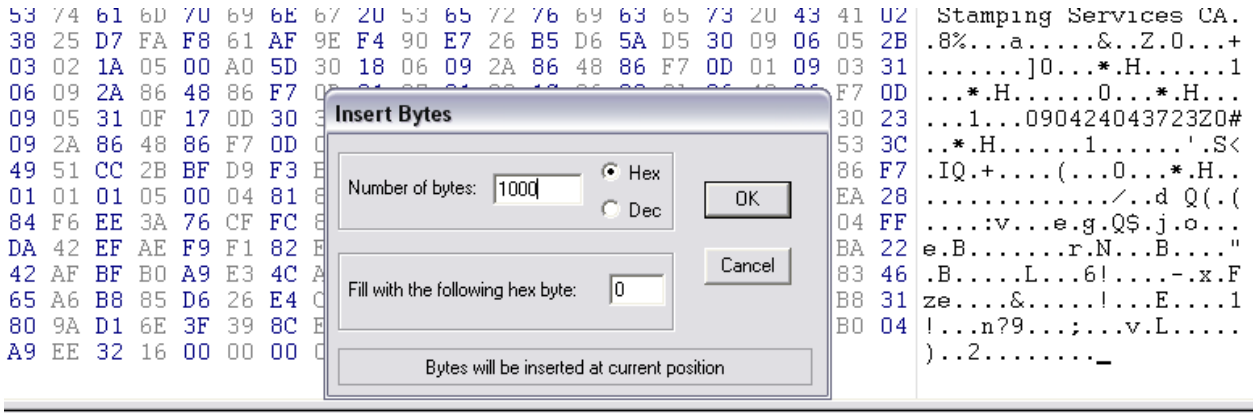
Flags = E00000E0 -> E0 00 00 E0

So we have calculated our new section's information , Now we have to add them to PE with Hex Editor

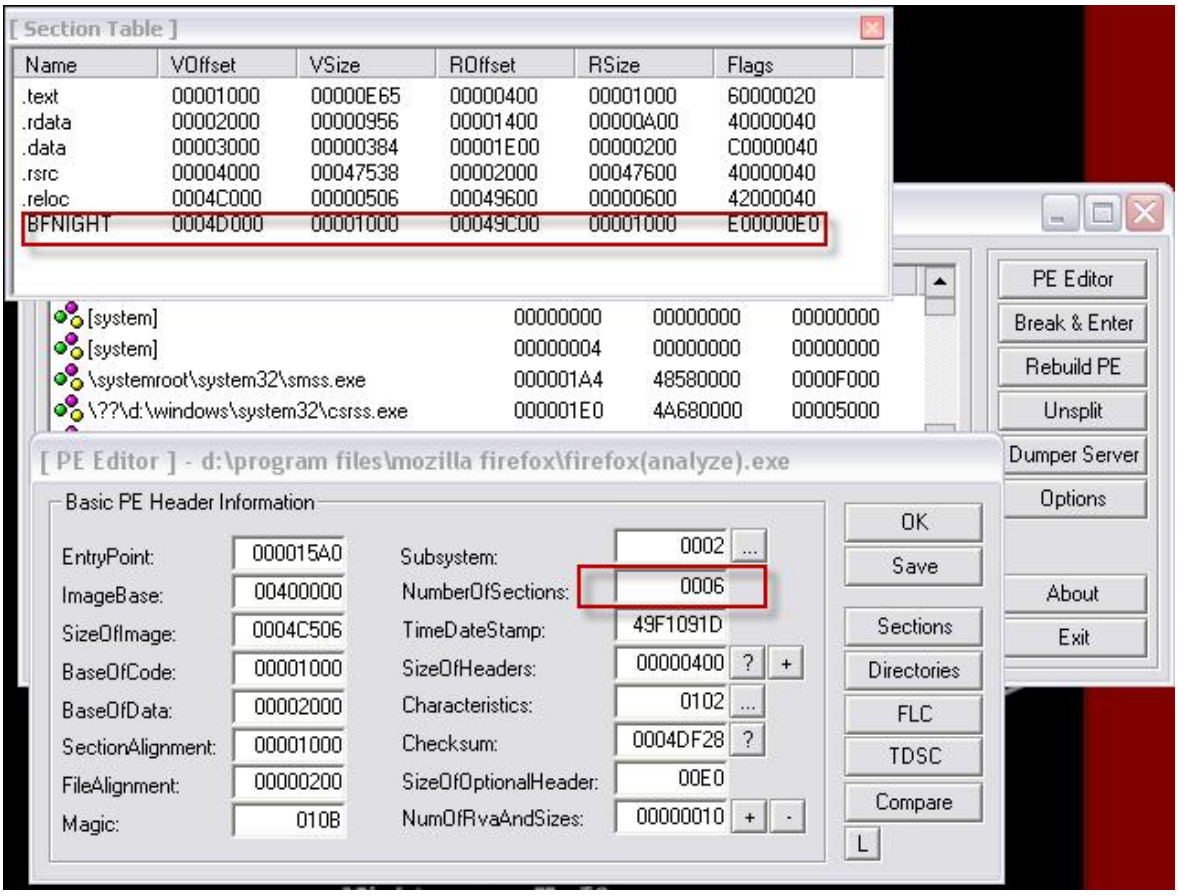


*Tip: Vsize = Virtual Size , Rsize = Raw Size

Then insert 1000h bytes to the target.



Now check the target with *LordPE* then change *NumberofSection* to 6. After that you can realize that our new section has been added.

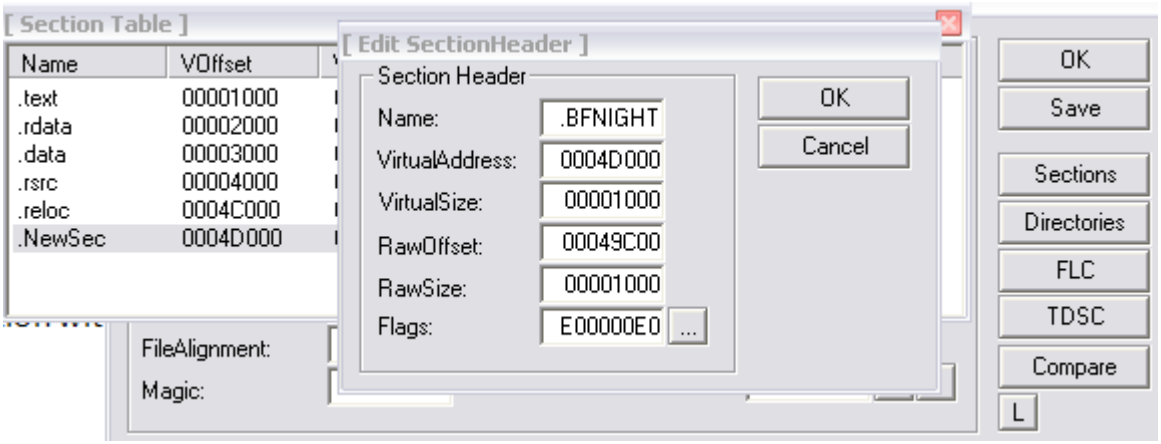


Congratulation you have added a new section to the program by your own hand **J**.

Now we want to add *.BFNIGHT* section in an easy way that needs only a good PE Editor like LordPE.

Open the target with lordPE then click the Section button, next right click and after that you can see that we have a option which is called *Add Section Header* .

A new section will be added by this option , So click it then you have to edit the new section's information with *Edit Section Header* like this.



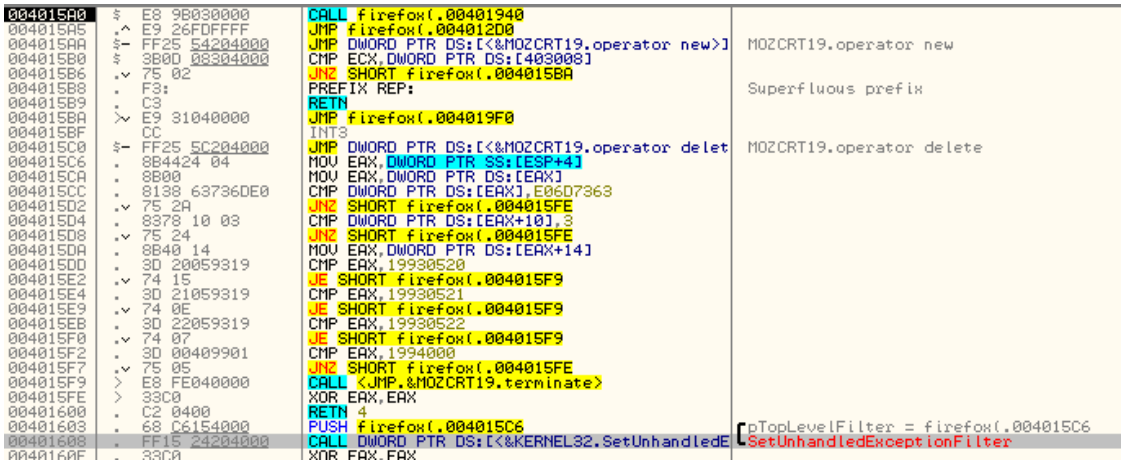
Then click ok and save. But now we need to insert 1000h bytes to the target with hex editor.

IAT (Import Address Table):

Every PE has a list of functions that aren't originally part of that PE. These functions are called Import which is located in the OS DLLs while PE doesn't know where they are located, So every win32 executable has an *Import Table Address* or *IAT* inside PE.

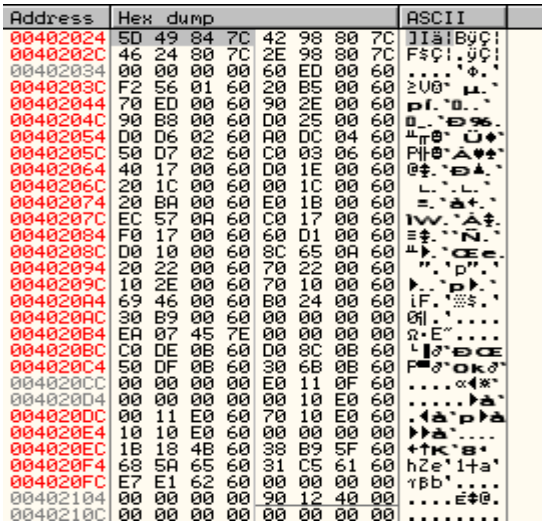
IAT contains all of information of imports , It means it is used as a lookup table when a PE is calling a windows32 API. Briefly the windows loader has to find each address of API which is called by the PE before starting.

Now take a look at inside our target(again *firefox* **J**)



CALL function which is shown at the figure is a good example of an API. It calls *SetUnhandledExceptionFilter* API. Right click on the function then click Assemble. You can see that it's like **CALL DWORD PTR [XXXXXX].[XXXXXX]** is the address of values which are located in memory

Go to the 402024 in olly hex dump (right click -> go to -> expression).



As the figure shows the address of the API is 7C84495D in my system. Notice that most of the API addresses start like 7XXXXXXX.

Now let see how windows loader can find addresses for every API. First windows loader reads the header of PE, so bytes at RVA 3C are read. The VA of the bytes will be 40003C(imagebase is 400000). Then if we plus 80h to the value of VA, we will find Virtual Address of Import Table.

But what's the Import Table?

The Import Table contains all of the information that is needed for windows to link the APIs for PE.

The structure of Import Table is simple . A header for each imported DLL plus additional null header that marks the end of Import Table. For example if you import APIs from Kernel32.dll and User32.dll , you will find 3 headers. 2 of them are for Kernel32.dll and User32.dll and a zero-header for marking the end.

The header of each dll is called *IMPORT_IMAGE_DIRECOTRY*. 'IMAGE' word refers that all of this stuff is done in memory.

But data structure which stores the information import symbols called IMAGE_IMPORT_DESCRIPTOR that has 5 properties. I'd like to use exact description of *Identifying Malicious Reverse Engineering Code Through* Book.

OriginalFirstThunk: - This member contains the RVA (pointer) of an array of IMAGE_TUNK_DATA structures. IMAGE_TUNK_DATA structures, is a union of dword size. This can be considered as a pointer to IMAGE_IMPORT_BY_NAME structure. The structure of IMAGE_IMPORT_BY_NAME structure is as follows: -
IMAGE_IMPORT_BY_NAME STRUCT
Hint
Name1

Time/Date Stamp: - After the image is bound, this field is set to the time/data stamp of the DLL. This field is not mandatory; it can be zero.

Forwarder Chain: - The index of the first forwarder reference. This field is not mandatory; it can be zero.

Name: - This member contains the RVA (pointer) of an ASCII string that contains the name of the DLL.

FirstThunk:- As the name suggests the FirstThunk is very similar to that of OriginalFirstThunk . Similar to FirstThunk it also contains pointer (RVA) to array of IMAGE_THUNK_DATA structures. Although both the arrays contain same value, they are at different locations in the executable.

Tip : (RVA = Relative Virtual Address , VA = Virtual Address)

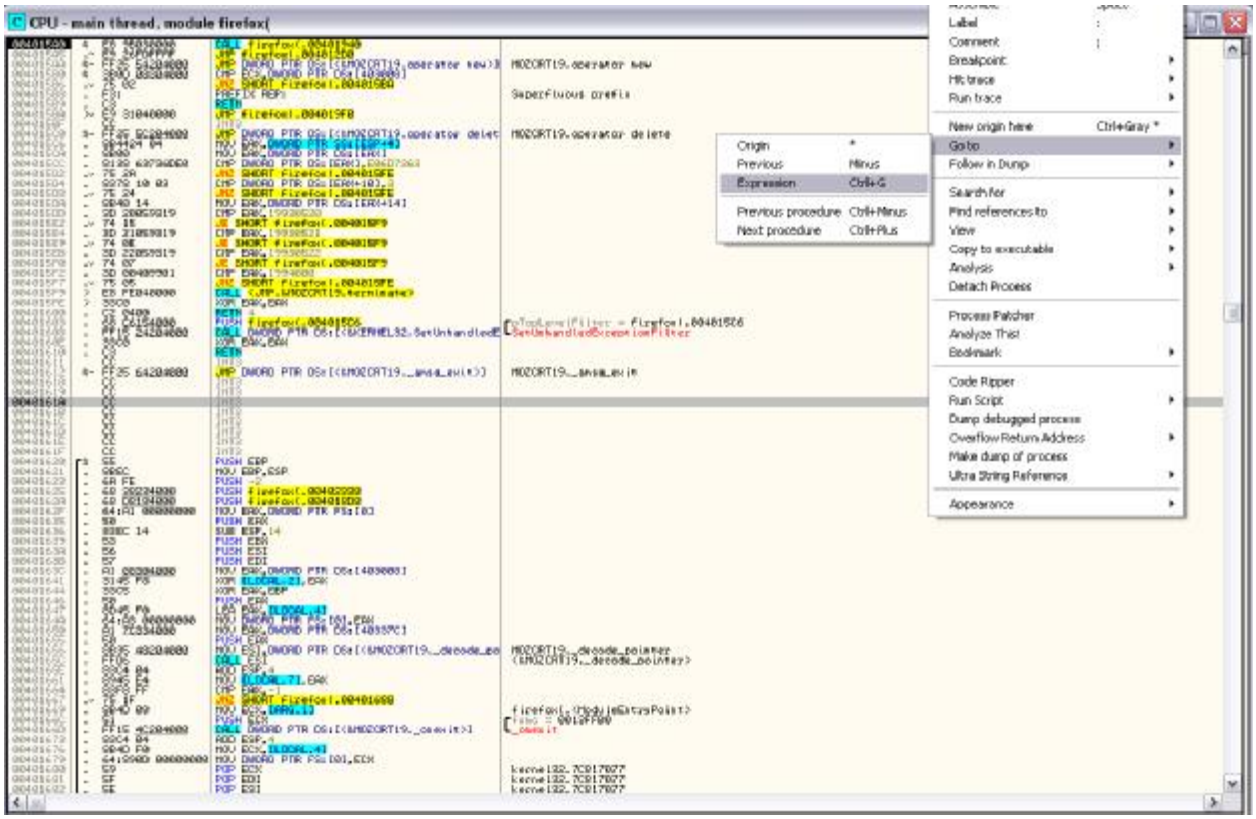
I think those are some boring description that make you bored, so go through see above information in our target. First we have to find Import Table, so open firefox in ollydbg then click on  (Memory Map Button) and double-click on PE Header.

```
00400000 00001000 firefox(          PE header   Imag 01001002 R
00401000 00001000 firefox( .text    code        Imag 01001002 R
00402000 00001000 firefox( .rdata   imports     Imag 01001002 R
00403000 00001000 firefox( .data    data        Imag 01001002 R
```

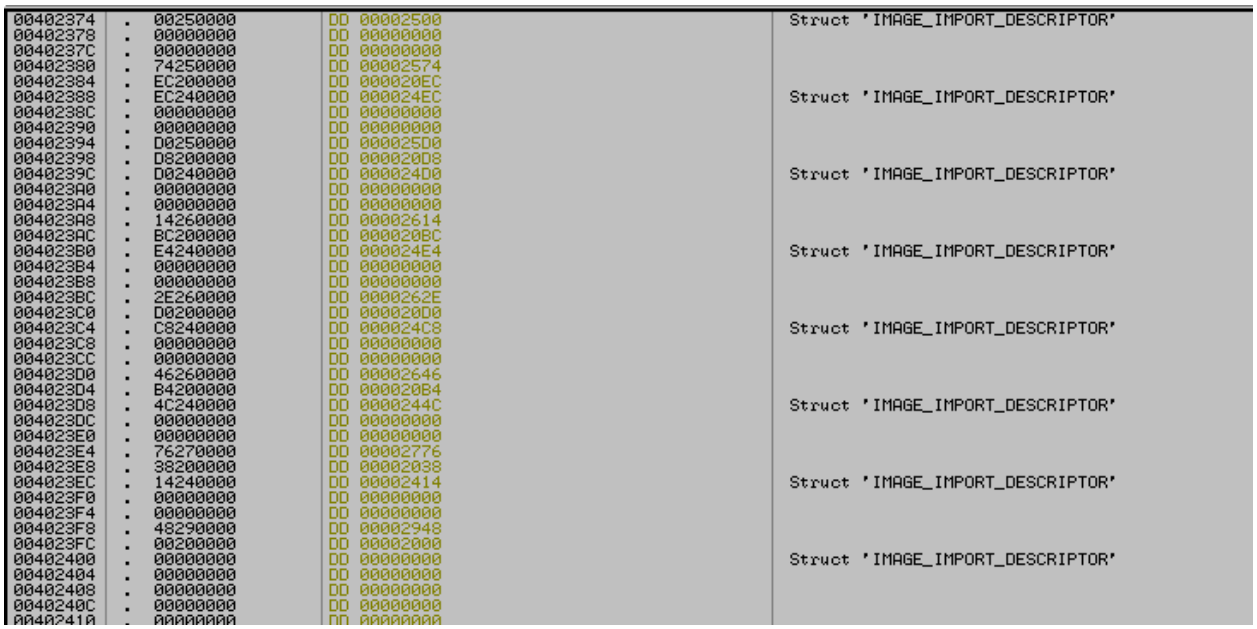
Now you are able to see the header of firefox. Find RVA of 3C (ImageBase + 3C = 40003c). You can see the value is D8 now plus it with 80h that the RVA will be like: D8h + 80h + 400000h = 400158h. This RVA stores the VA of Import Table.

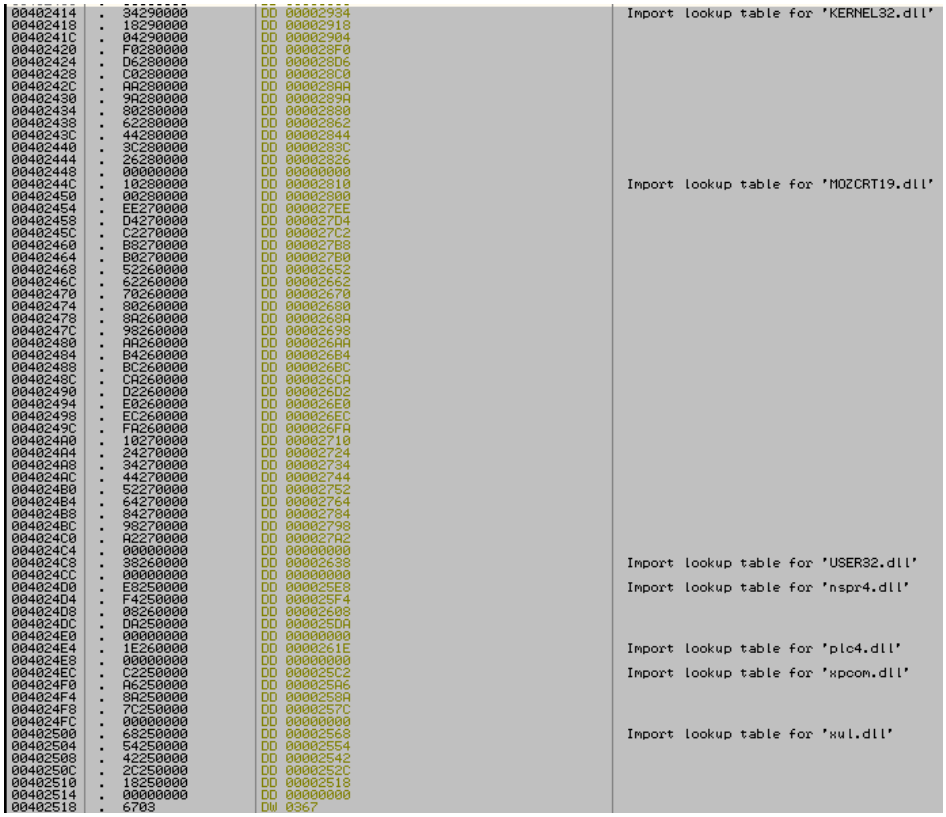
```
00400158   74230000   DD 00002374   Import Table address = 2374
0040015C   A0000000   DD 000000A0   Import Table size = A0 (160.)
00400160   00400000   DD 00004000   Resource Table address = 4000
00400164   38750400   DD 00047538   Resource Table size = 47538 (292152.)
00400168   00000000   DD 00000000   Exception Table address = 0
0040016C   00000000   DD 00000000   Exception Table size = 0
00400170   009C0400   DD 00049C00   Certificate File pointer = 49C00
00400174   F8150000   DD 000015F8   Certificate Table size = 15F8 (5624.)
00400178   00C00400   DD 0004C000   Relocation Table address = 4C000
0040017C   94010000   DD 00000194   Relocation Table size = 194 (404.)
00400180   20210000   DD 00002120   Debug Data address = 2120
```

The RVA of Import Table Address will be 402374. So go to 402374 like:



Then enter 402374 in to the box and click ok then you need to Analyze Code(CTRL + A). If you consider you can find that we have 8 IMAGE_IMPORT_DESCRIPTORs and last one is a zero-header that marks the end then we have Import Lookup Table that stores the information of each dll that. Then we have the name of each function that is imported. These are Import's information that is pointed by IMAGE_IMPORT_DESCRIPTOR.






0040252A . 61 00 ASCII "a",0
0040252C . 6A03 DW 036A
0040252E . 58 52 45 5F 47 6 ASCII "XRE_GetFileFromP"
0040253E . 61 74 68 00 ASCII "ath",0
00402542 . 6803 DW 0368
00402544 . 58 52 45 5F 46 7 ASCII "XRE_FreeAppData",0
00402554 . 6903 DW 0369
00402556 . 58 52 45 5F 47 6 ASCII "XRE_GetBinaryPat"
00402566 . 68 00 ASCII "h",0
00402568 . 7103 DW 0371
0040256A . 58 52 45 5F 6D 6 ASCII "XRE_main",0
00402573 . 00 DB 00
00402574 . 78 75 6C 2E 64 6 ASCII "xul.dll",0
0040257C . 2400 DW 0024
0040257E . 4E 53 5F 4C 6F 6 ASCII "NS_LogTerm",0
00402589 . 00 DB 00
0040258A . 0400 DW 0004
0040258C . 4E 53 5F 43 53 7 ASCII "NS_CStringContai"
0040259C . 6E 65 72 49 6E 6 ASCII "nerInit2",0
004025A5 . 00 DB 00
004025A6 . 0200 DW 0002
004025A8 . 4E 53 5F 43 53 7 ASCII "NS_CStringContai"
004025B8 . 6E 65 72 46 69 6 ASCII "nerFinish",0
004025C2 . 2200 DW 0022
004025C4 . 4E 53 5F 4C 6F 6 ASCII "NS_LogInit",0
004025CF . 00 DB 00
004025D0 . 78 70 63 6F 6D 2 ASCII "xpcom.dll",0
004025DA . 7A01 DW 017A
004025DC . 50 52 5F 73 6D 7 ASCII "PR_smprintf",0
004025E8 . 9300 DW 0093
004025EA . 50 52 5F 47 65 7 ASCII "PR_GetEnv",0
004025F4 . 7B01 DW 017B
004025F6 . 50 52 5F 73 6D 7 ASCII "PR_smprintf_free"
00402606 . 00 ASCII 0
00402607 . 00 DB 00
00402608 . 3801 DW 0138
0040260A . 50 52 5F 53 65 7 ASCII "PR_SetEnv",0
00402614 . 6E 73 70 72 34 2 ASCII "nspr4.dll",0
0040261E . 0800 DW 0008
00402620 . 50 4C 5F 73 74 7 ASCII "PL_strcasecmp",0
0040262E . 70 6C 63 34 2E 6 ASCII "plc4.dll",0
00402637 . 00 DB 00
00402638 . F801 DW 01F8
0040263A . 4D 65 73 73 61 6 ASCII "MessageBoxA",0
00402646 . 55 53 45 52 33 3 ASCII "USER32.dll",0
00402651 . 00 DB 00
00402652 . 0F00 DW 000F
00402654 . 3F 3F 32 40 59 4 ASCII "??2@YAPAXI@Z",0
00402661 . 00 DB 00
00402662 . 0404 DW 0404
00402664 . 5F 76 73 6E 70 7 ASCII "_vsnprintf",0
0040266F . 00 DB 00
00402670 . 1100 DW 0011
00402672 . 3F 3F 33 40 59 4 ASCII "??3@YAXPAX@Z",0
0040267F . 00 DB 00
00402680 . 8C05 DW 058C
00402682 . 77 63 73 6C 65 6 ASCII "wcslen",0
00402689 . 00 DB 00
0040268A . 1601 DW 0116
0040268C . 5F 61 6D 73 67 5 ASCII "_amsg_exit",0
00402697 . 00 DB 00
00402698 . F800 DW 00F8
0040269A . 5F 5F 77 67 65 7 ASCII "__wgetmainargs",0
004026A9 . 00 DB 00
004026AA . 2C01 DW 012C
004026AC . 5F 63 65 78 69 7 ASCII "_cexit",0
004026B3 . 00 DB 00
004026B4 . 7C01 DW 017C
004026B6 . 5F 65 78 69 74 0 ASCII "_exit",0
004026BC . 6300 DW 0063
004026BE . 5F 58 63 70 74 4 ASCII "_XcptFilter",0
004026CA . C704 DW 04C7
004026CC . 65 78 69 74 00 ASCII "exit",0
004026D1 . 00 DB 00
004026D2 . F900 DW 00F9
004026D4 . 5F 5F 77 69 6E 6 ASCII "__winitenv",0
004026DF . 00 DB 00
004026E0 . FF01 DW 01FF
004026E2 . 5F 69 6E 69 74 7 ASCII "_initterm",0
004026EC . 0002 DW 0200
004026EE . 5F 69 6E 69 74 7 ASCII "_initterm_e",0
004026FA . 3C01 DW 013C
004026FC . 5F 63 6F 6E 66 6 ASCII "_configthreadloc"
0040270C . 61 6C 65 00 ASCII "ale",0
00402710 . E400 DW 00E4
00402712 . 5F 5F 73 65 74 7 ASCII "__setusermatherr"
00402722 . 00 ASCII 0

Inject A DLL:

I think now we have enough information to inject a dll to our target, but above all we need a framework, a paper & pencil.

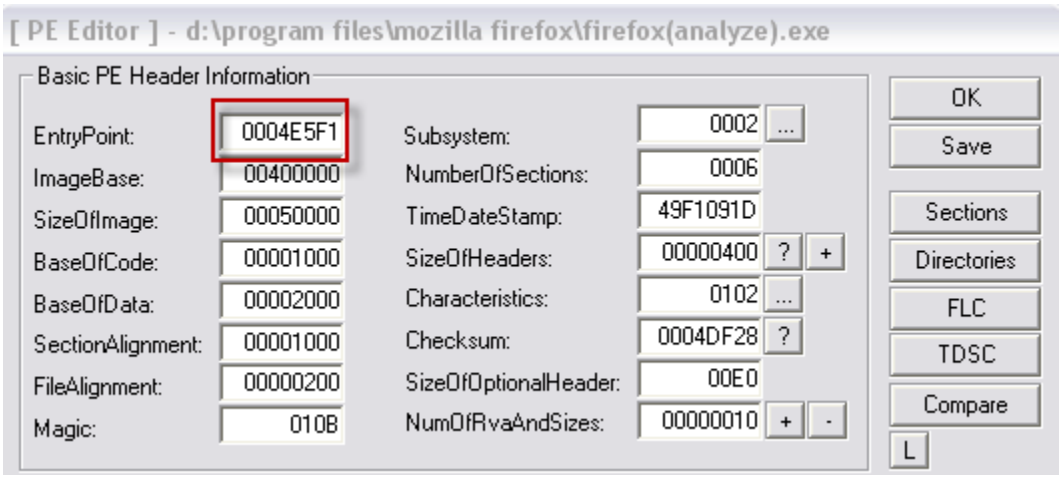
Framework:

1. First we need to add a section then make jump to the main EP(Entry Point) and change EP to our section's EP.
2. Then all of firefox's IMAGE_IMPORT_DESCRIPTOR should be moved to the new section and Import Table Address's value should be changed to the New Section's Import Table Address
3. Next we have to add Import Lookup and a IMAGE_IMPORT_DESCRIPTOR for dll which you want to inject
4. Then call it.
5. ENJOY!

1st step

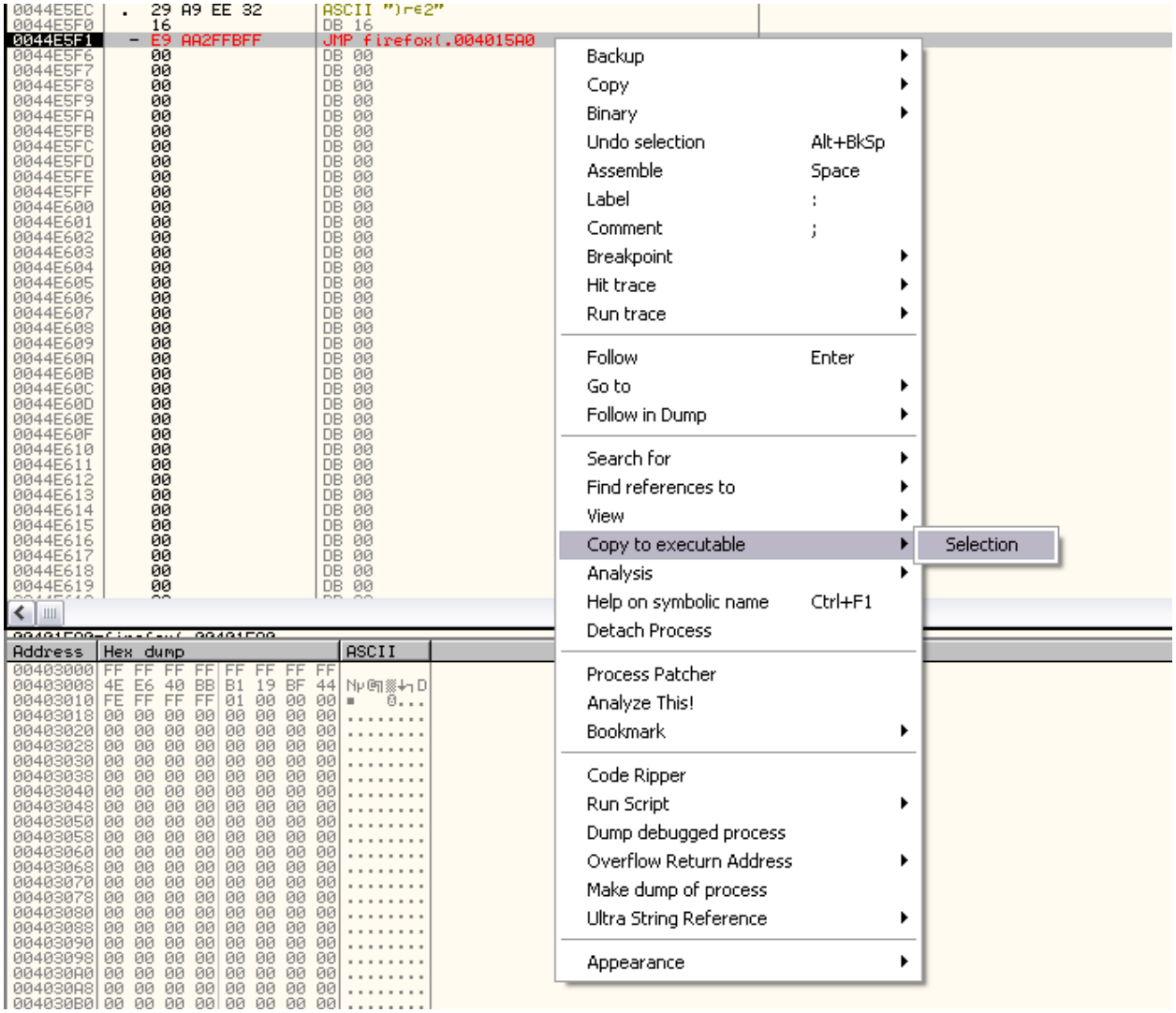
You need to Add a section by lordpe with 3000h size like I said, then you need to find the RVA of EP. easy way is value of .BFNIGHT's Voffset. So my EP would be "4D000 + 15F1 = 4E5F1" (consider EP should be a place that has lots of *Code Cave*). Before change the EP you need to write the main of EP in your paper because we want to make a jump to the main EP.

So main EP is 4015A0 and new EP is 4E5F1. So lets change it.



Tip : Code Cave is Place of a PE which has null data

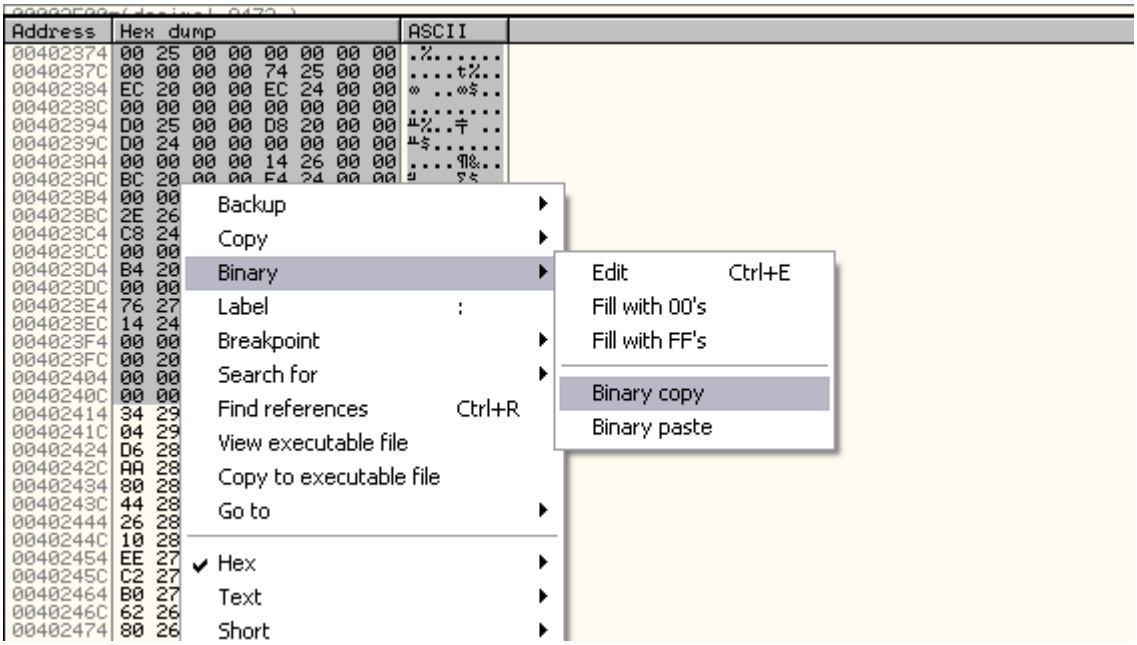
Now open target in ollydbg and assemble “JMP 4015A0” then compile it.



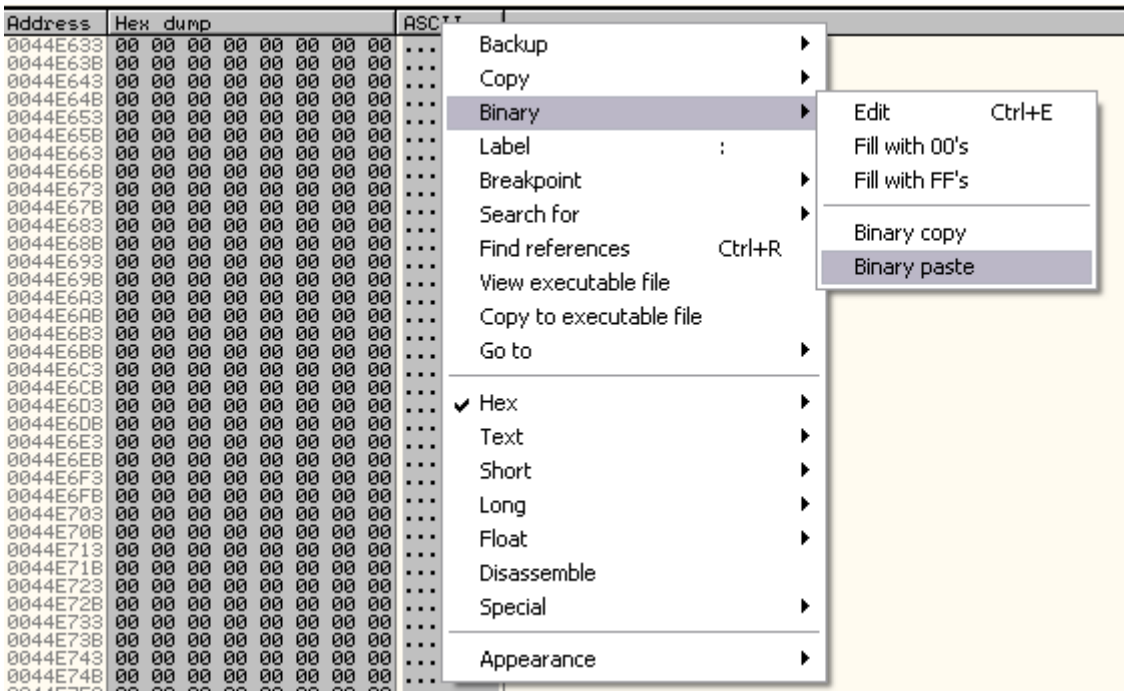
If you run target , It will work correctly.

2nd Step

Find IMAGE_IMPORT_DESCRIPTOR like I said then highlight them and Right click and Follow Value in Dump. Then Right click on the highlighted values and Binary then Binary Copy



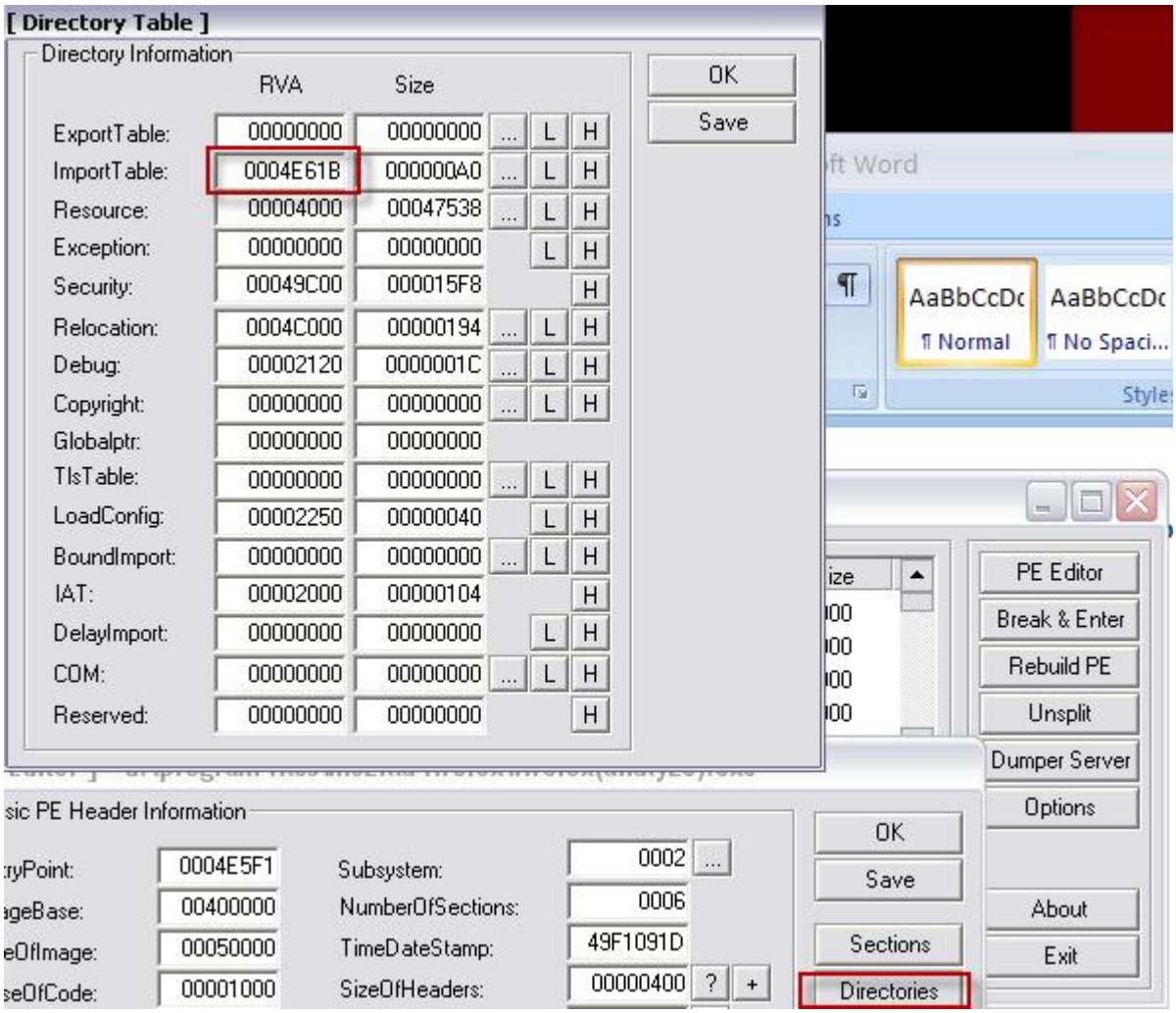
Now paste them in to the our Null space in our new section like: (Then you have to compile the target again **J**)



So we need to change Import Table Address . The first byte of IMAGE_IMPORT_DESCRIPTOR is Import Table Address and we have to change it with lordpe.

Import Table Address(new) = 44E61B (in my system) RVA = 44E61B – 400000 = 4E61B

Open the target in LordPE and click on Directories Button on the right and change ImportTable to the new RVA



3rd and 4th steps:

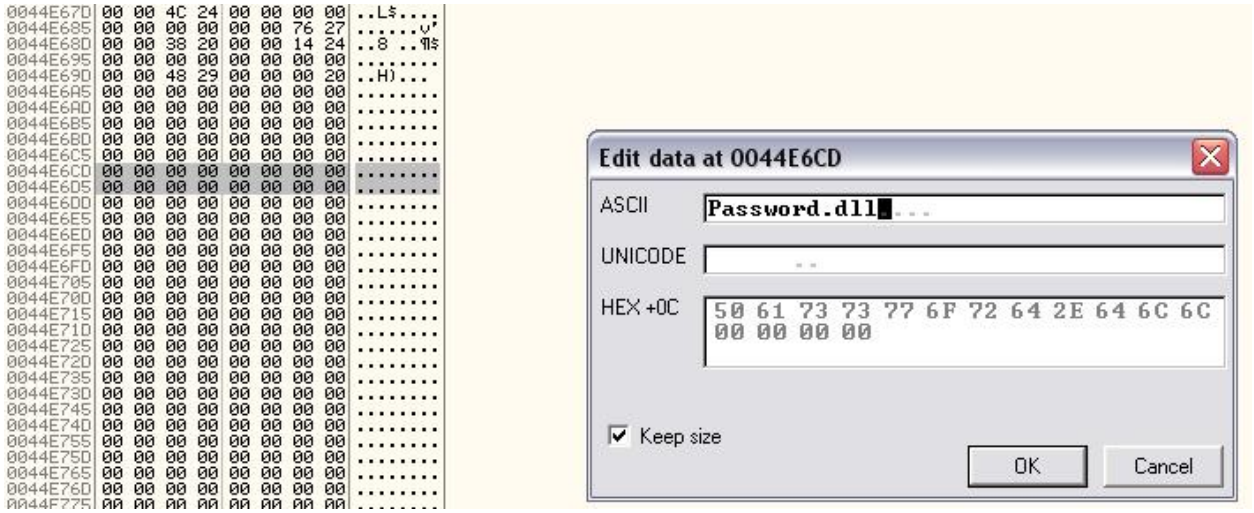
In this case I'd like to use Ashraf Cracker's DLL (Password.dll).

http://www.4shared.com/file/112647070/69e44e47/Password.html

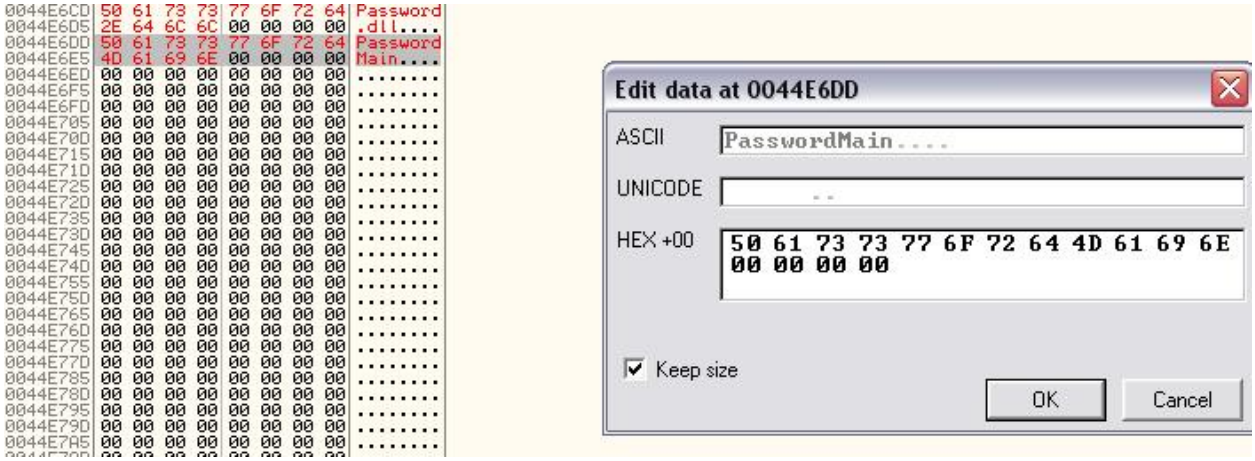
So copy this dll in firefox's directory. Password.dll has a function that is called PasswordMain.

Now we need to make Import Lookup Table for the dll. Choose a Null Data Place like 44E6CD and find it in Value Dump view.

First you need to insert dll name (Password.dll)

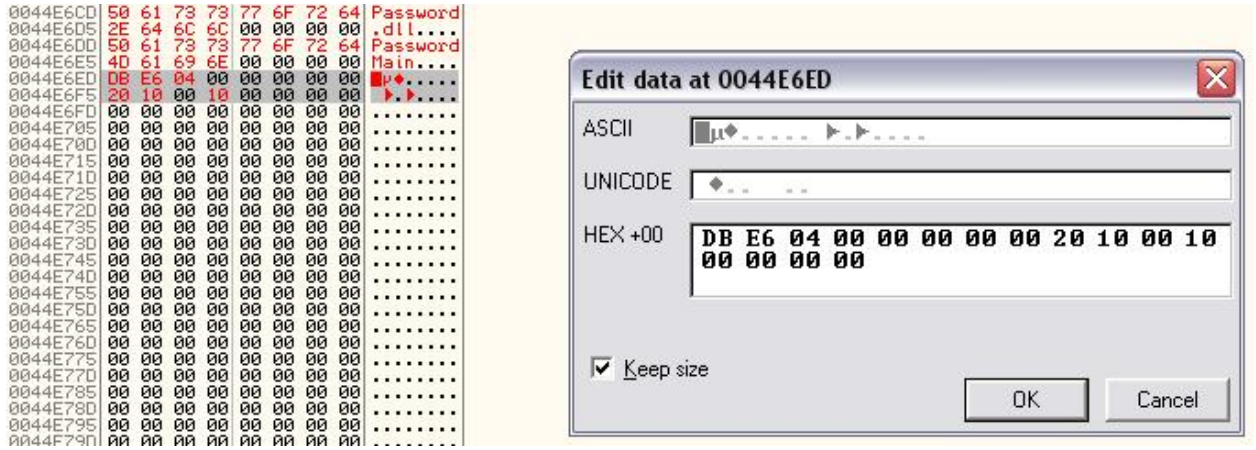


Then you need to insert Function name



Next you need to insert the RVA that points to the function name and 20 10 00 10

RVA = 44E6DB – 400000 = 4E6DB **à** DB E6 04 00



So the final step in make a IMAGE_IMPORT_DESCRIPTOR witch this structure

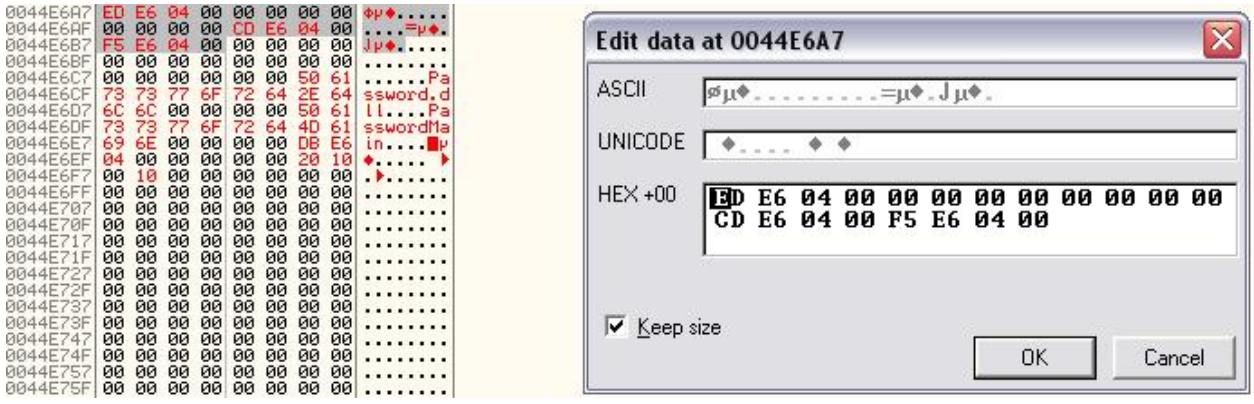
OrginalFirstThunk = 44E6ED – 400000 = 4E6ED **à** ED E6 04 00

Time/Date Stamp and ForwarderChain = Zero

Name: 4E6CD **à** CD E6 04 00

FirstThunk: 4E6F5 **à** F5 E6 04 00

Find the values of the last IMAGE_IMPORT_DESCRIPTOR in the olly's Value Dump and insert those data which is calculated like:



Then save the target. Now if you assemble CALL DWORD PTR [44E6F5] , you can find that the dll has been injected correctly and it works perfectly.

The password is : 0128793089

Finally we have injected a dll in our target and now we know how to do it.

Our website: http://b0frenzy.freehostia.com

Special Thanks to Lena151 because of his perfect tutorials and website

GREAT IRAN

GREAT BLACK-0UT FRENZY

BE SAFE