

ThunderPwr

Tutorial n° 4

Ver 1.0 - 20/08/2004

Target definition

Target	JOC Web Spider
Version	371
Reference	http://www.jocsoft.com/jws/
Program description	Web downloader
Addictional stuff	Packed with UPX 0.89.6 – 1.02 / 1.05 – 1.24 -> Markus & Laszlo

Protection scheme

Trial version with 20 day limitation, to unlock the program is needed a registration with valid serial number, this depend on ID of own hard-disk.

Program came packed with UPX 0.89.6 – 1.02 / 1.05 – 1.24 -> Markus & Laszlo

What we can do

Our task is force the registration without entering any number.

Tool used

Dumper/unpacker	ProcDump32 Version 1.6.2 final (NT2K compiled on 01/17/00)
IAT Reconstructor	Import REConstructor v1.6 FINAL
Unpacker	PE Explorer 1.95 (http://www.heaventools.com/download.htm)
Debugger	OllyDbg v1.10

References

-

ABSTRACT

Well, this tutorial will be spoken about a program able to download a web site for off-line reading. This program came from the author in a packed form (with UPX) and in trial form, with 20 day limitation over this limit program expire and to unlock you have to register it. Protection based on integrity check it's implemented. The study is explained through some basic step, for each phase will be detailed the operation needed to force the program.

STEP 1 – Executable file analysis after installation

When the installation process is ended you must check if the target program is encrypted/packed or in a plain form.

In order to make this we can use the PeiD file scanner, see figure 1 for detail:

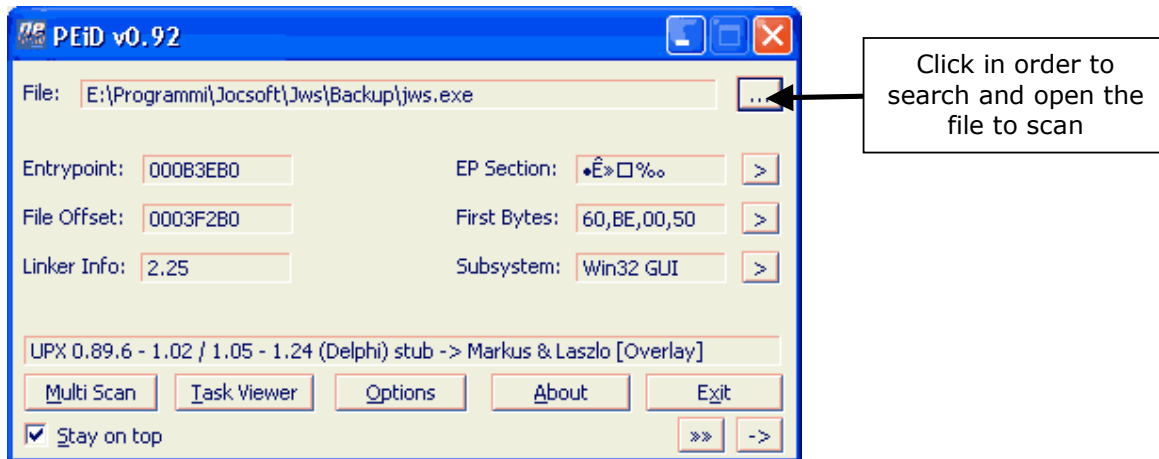


Fig. 1 PEiD file scanner detail.

This file was packed with UPX, at this point you have two way, one it's manually unpack the file and another is find the right unpacker tool to do this task automatically.

STEP 2 – Unpacking the application (a semi manual approach)

First step is manual unpack, this can be made with OllyDbg and OllyDump plugin, dumping step is very easy because this packer is UPX. About this you have only to search for the POPAD instruction, subsequent JMP instruction is the jump to our OEP.

Now load the packed executable into OllyDbg:

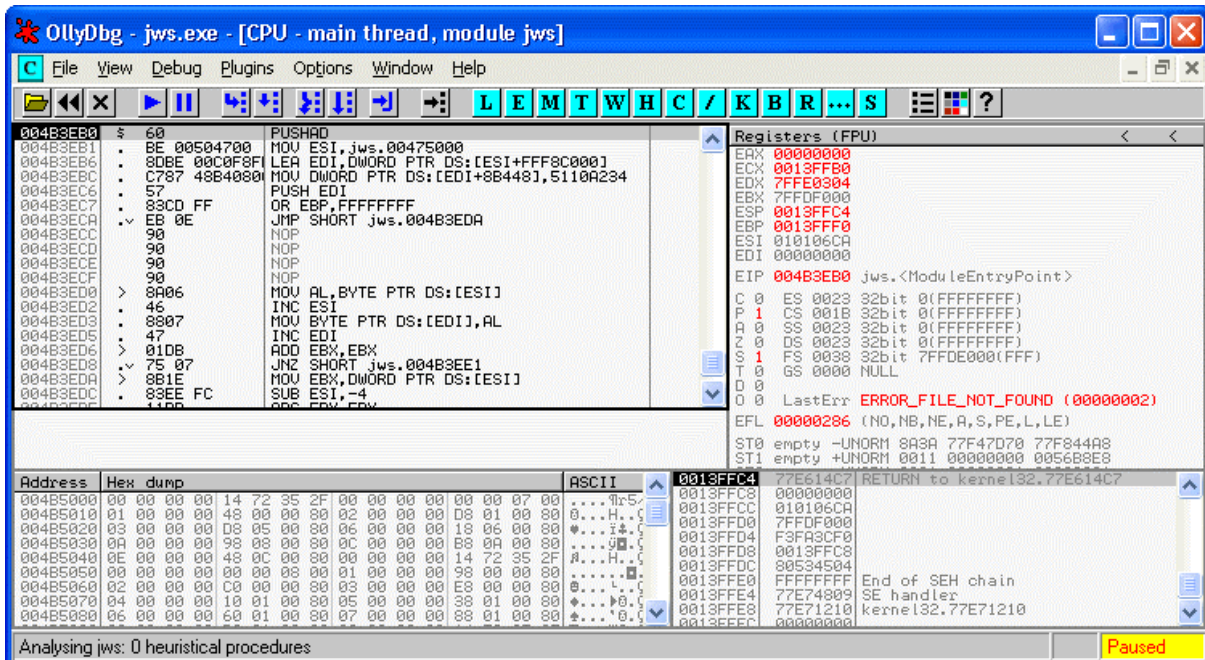


Fig. 2 Loading the packed executable into OllyDbg.

Scroll down until you can reach the POPAD instruction this is on location 004B401A:

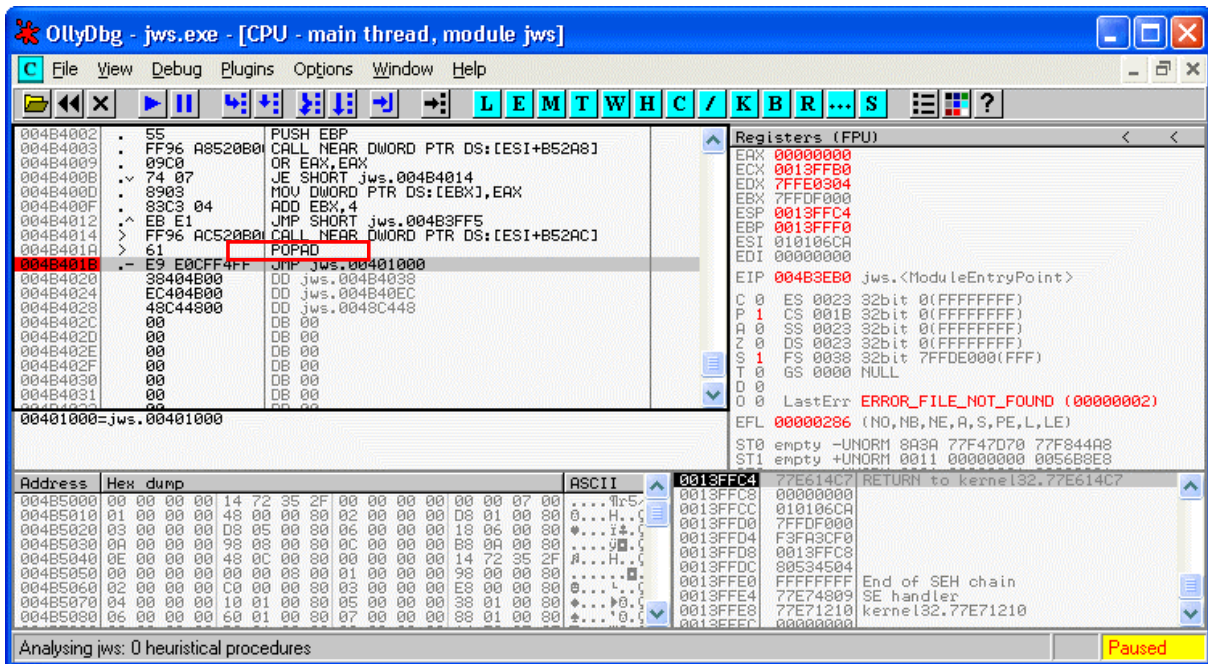


Fig. 3 Finding the jump to the OEP.

Place a breakpoint into the JMP instruction and the press F9 to run the program, OllyDbg stop into our breakpoint, then press F7 to jump into the real OEP.

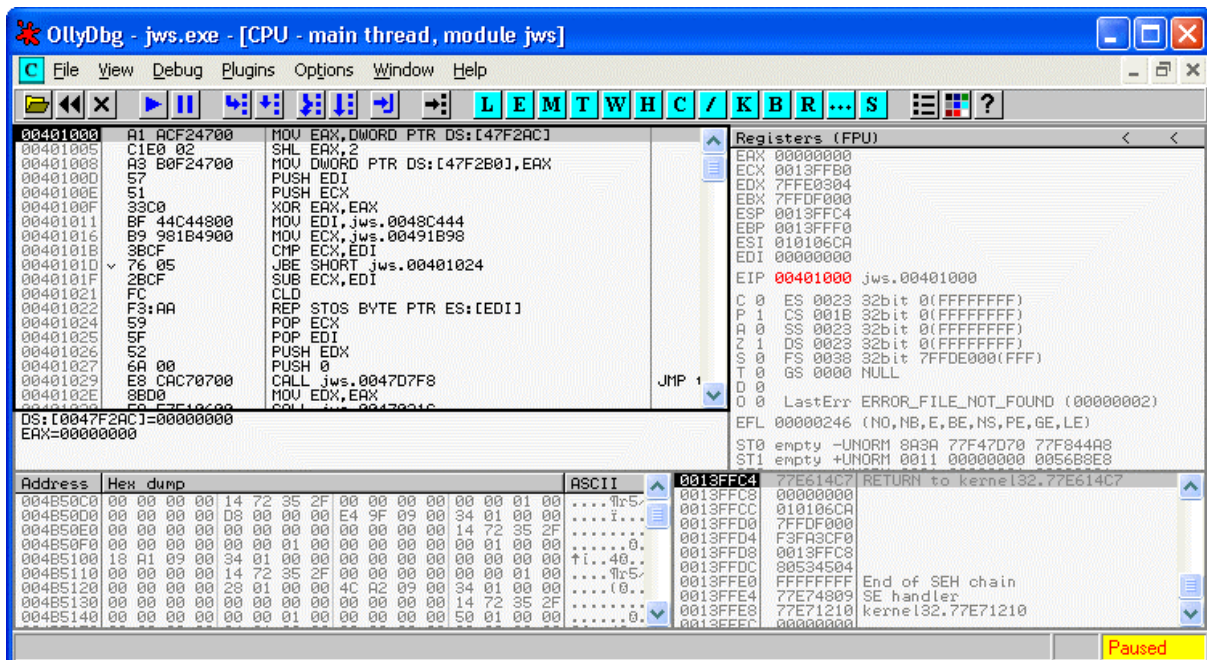


Fig. 4 Stop into the real OEP.

Now it's time to dump, from plugin menu select OllyDump:

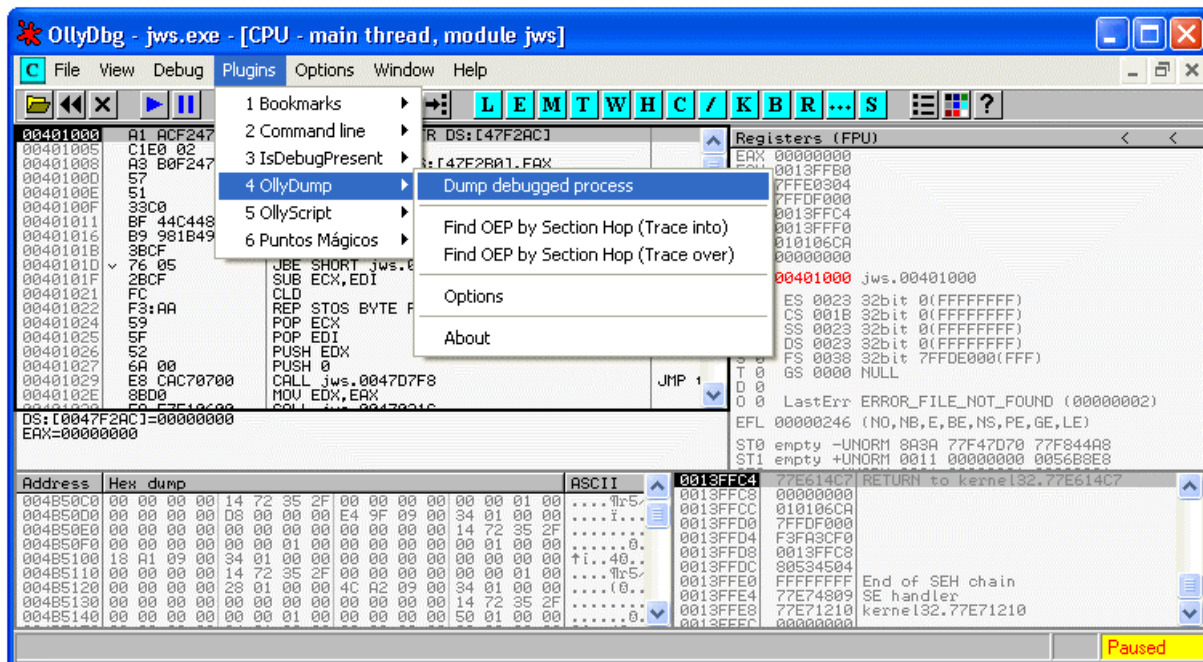


Fig. 5 OllyDump plugin.

A new window opening, be sure your option is the same:

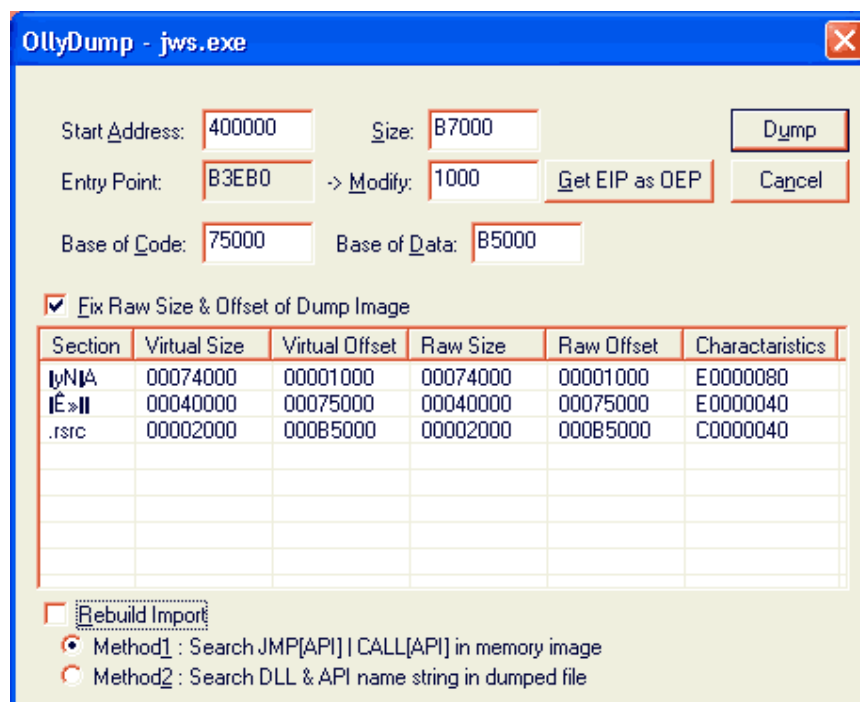


Fig. 6 OllyDump plugin, option setting.

Verify to have uncheck the Rebuild Import menu, about this we make a manual fix using impREC. Press the *Dump* button and then save the dumped file, for example with name ujws.exe, where u stand for unpacked.

Close OllyDbg and run the packed executable, run impREC and then pick the process into impREC (open the drop down menu and the select the jws.exe process):

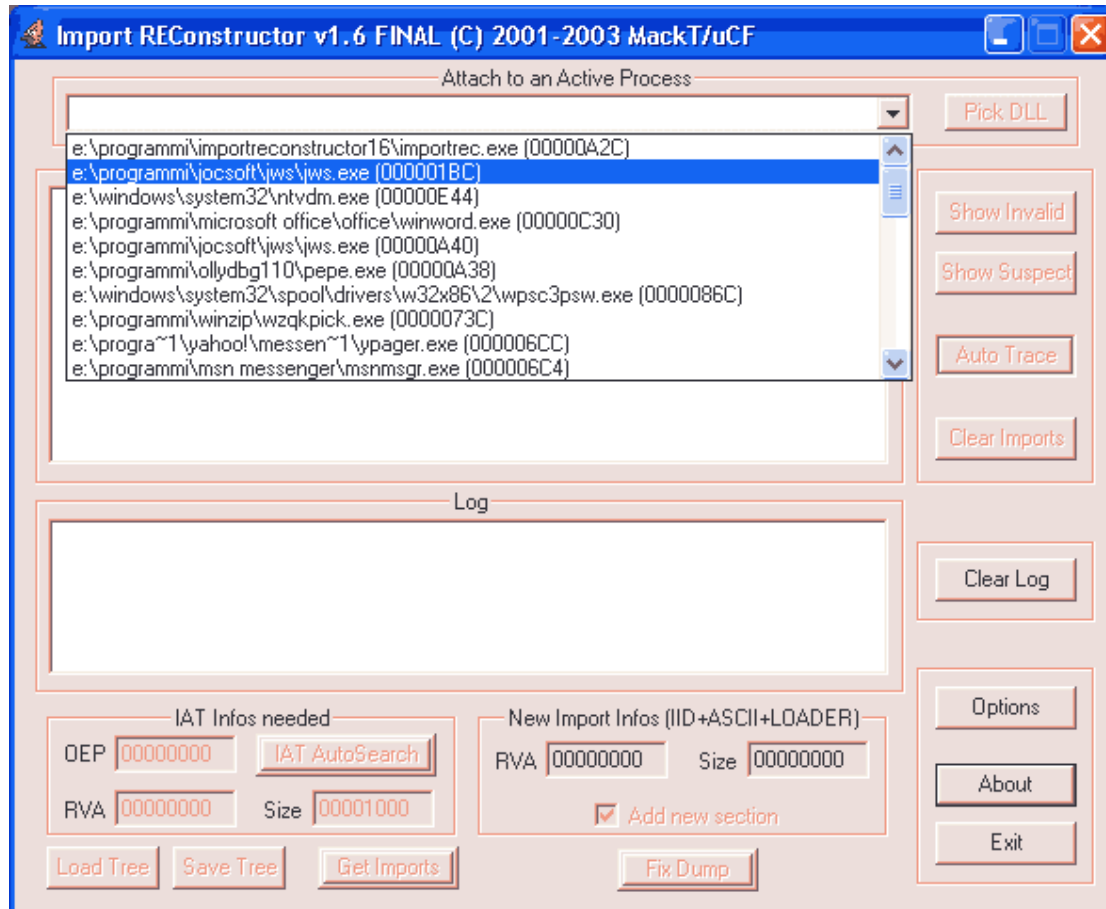


Fig. 7 Pick the process into impREC.

When the process is attached you can see the window into fig. 8:

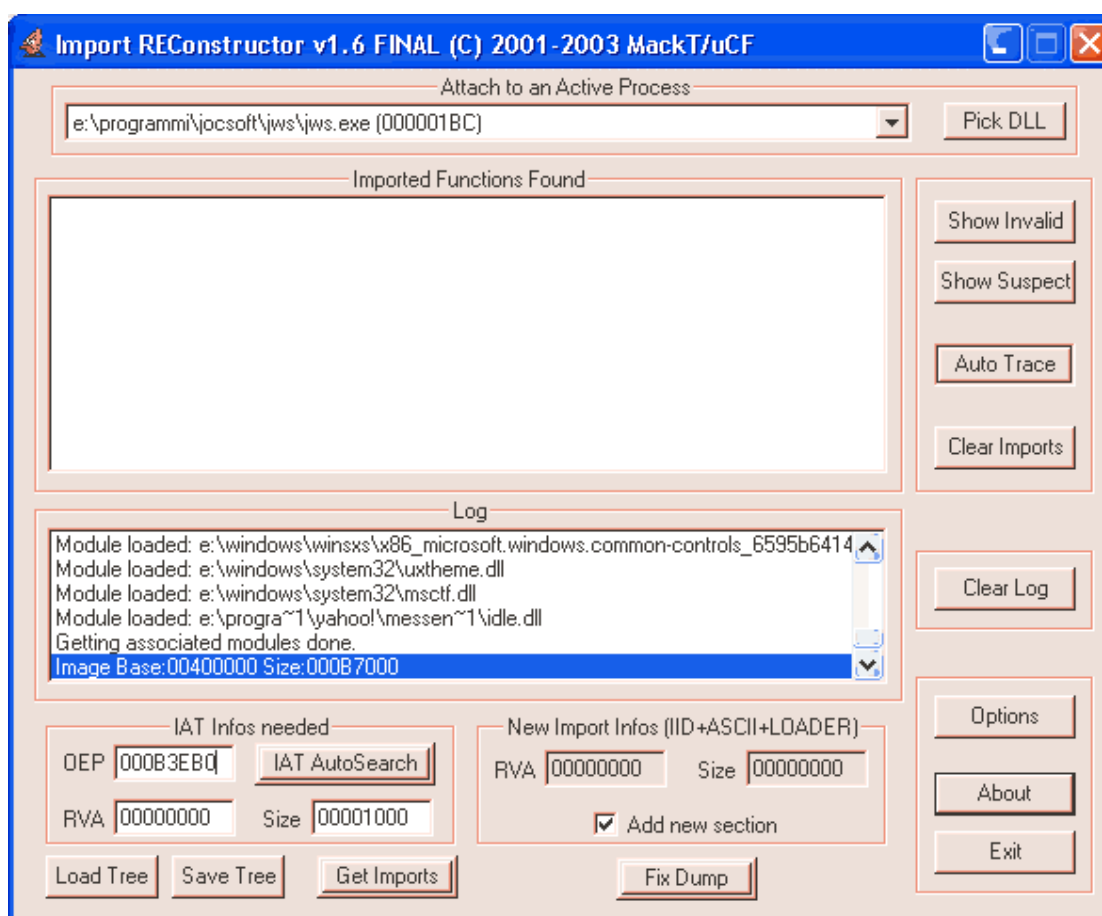


Fig. 8 The process is attached.

At this point you've to change the IAT Infos needed to, write into OEP a value of 00001000, this is the offset of our real entrypoint, remember this is at 00401000, with a imagebase of 0040000 offset is equal to:

$$\text{OEP} = \text{Real Entry Point} - \text{image base} = 00401000 - 00400000 = 0001000$$

Well, press IAT AutoSearch, the a messagebox popped up, press ok to continue, at this point your RVA is equal to 00094300 and size 000001D0, change size to 00001000 and press Get Imports (changing the Size is needed to get all the function which reside into the IAT).

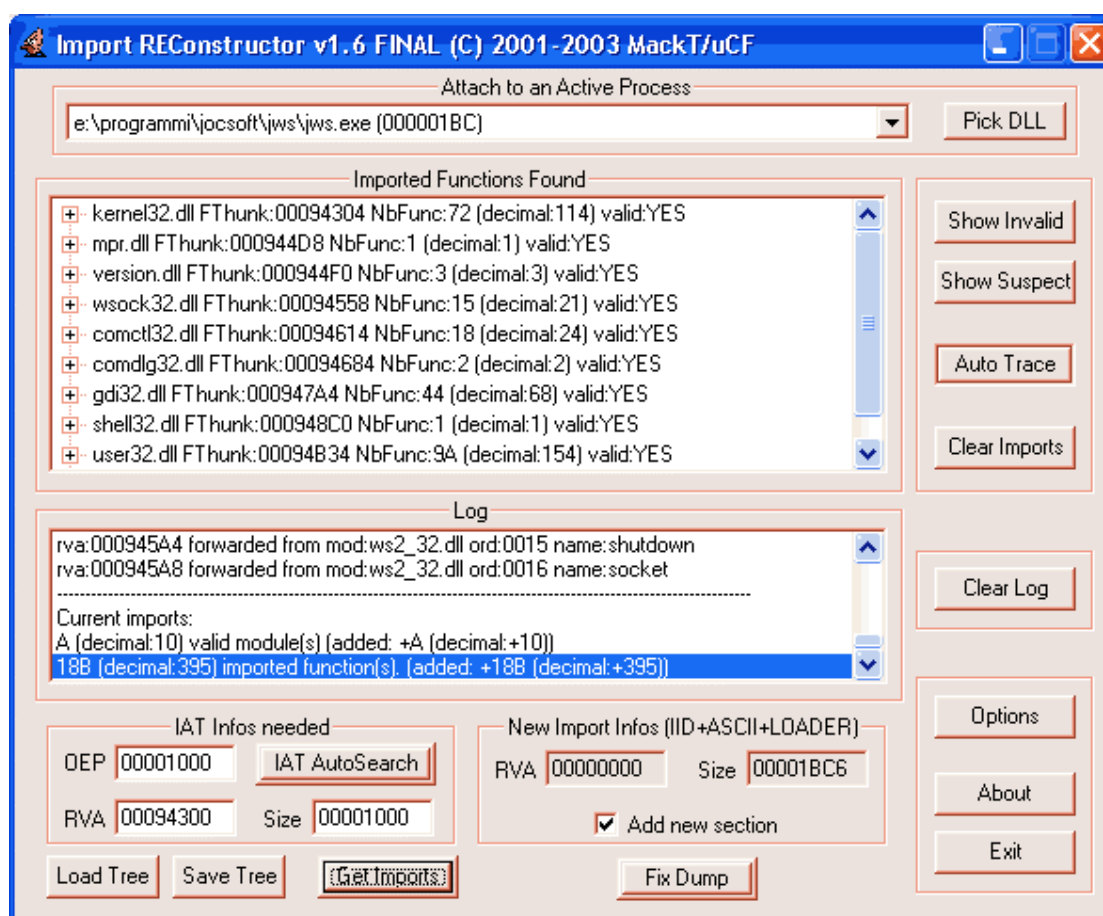


Fig. 9 IAT imported.

Well, all the function is valid, now is time to fix our dumped file then press the Fix Dump button and search for the dumped file (ujws.exe), when impREC finish a new file is created with name ujws_.exe. Pick all the exe file into a separate directory and then copy into the main folder only the impREC fixed file, rename it jws.exe and run to see if this work.

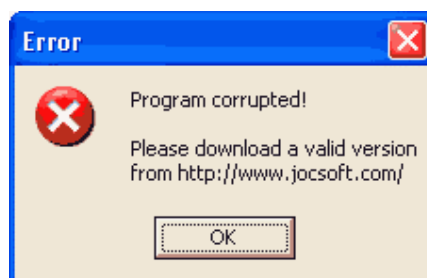


Fig. 10 Run the unpacked and fixed executable.

An error message about integrity check arise, no problem, run OllyDbg and then load the unpacked executable into it, make some step, you must see a jump instruction at 0040139F, if the jump is taken the message box call is skipped then make a change from JNZ to JMP, fix the executable with right-click, Copy to executable->Selection, into the new window right click and then right-click->Save to file.

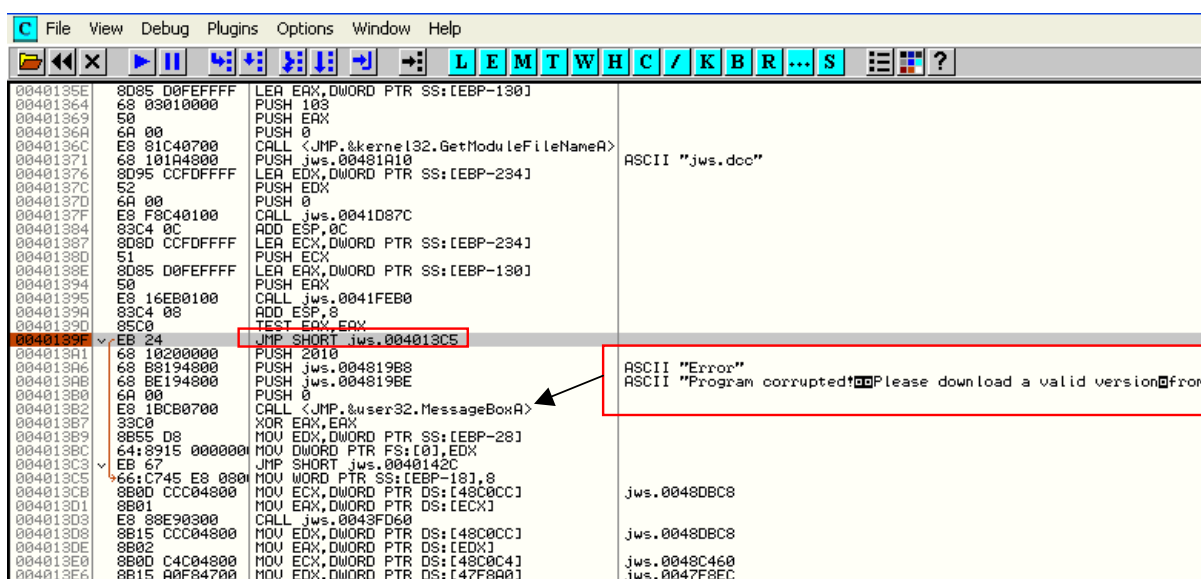


Fig. 11 Skip the integrity check.

If you run the patched executable program this start correctly, well now you have an unpacked and fixed executable able to work, next step is patching it to full registration without need to insert any serial.

STEP 3 – Check of the program coding

The unpacked program work well, then before the patching step you must take a look about the source coding, again we can check the unpacked executable file with a file scanner, using PEiD we obtain:



Fig. 12 Scanning the unpacked executable file.

Well no other packing are found then we can go to the next step.

STEP 4 – Check the right behaviour of the unpacked program

First step is launch the original packed executable (make also your observation about program behaviour), first you have a dialog which tell the registration code which is computer dependant, if you press OK the program can continue to run into trial mode mode:

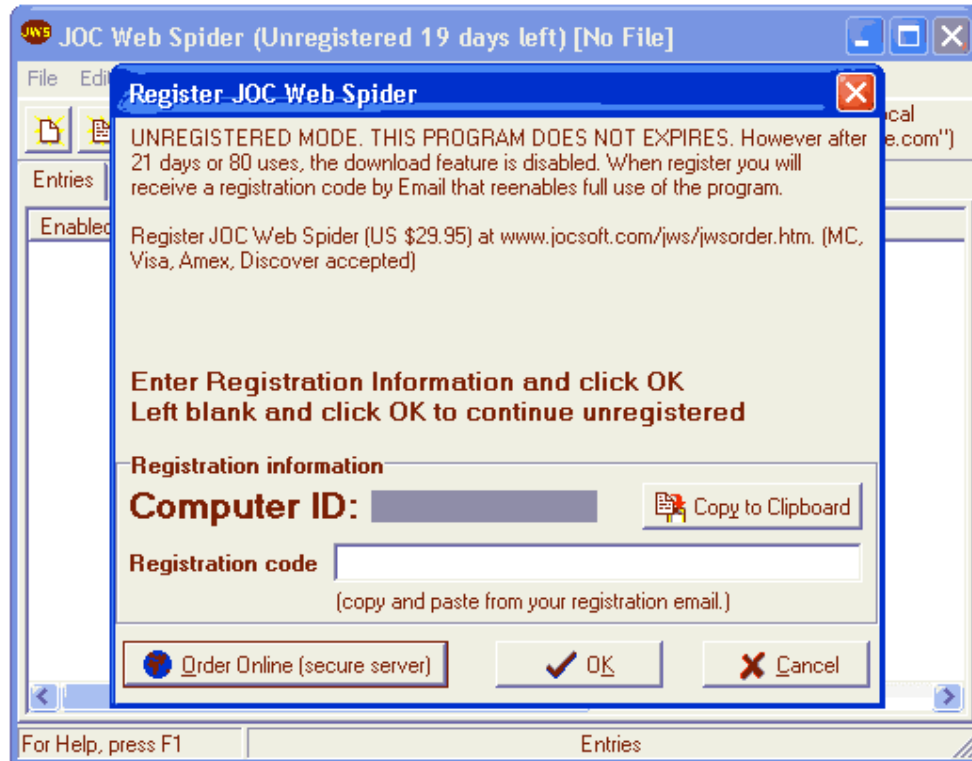


Fig. 13 Run the unpacked file and see if this behaviour is same as original.

Ok, this is right, then you can step to the next section.

STEP 5 – Debugging and patching stage

It's time to step into the code and then debug it, start OllyDbg and load the unpacked executable into it. Press F9 to run the program, when you reach the registration form stop into OllyDbg (press F12) and then into the code window press ALT+K to show the Call stack:

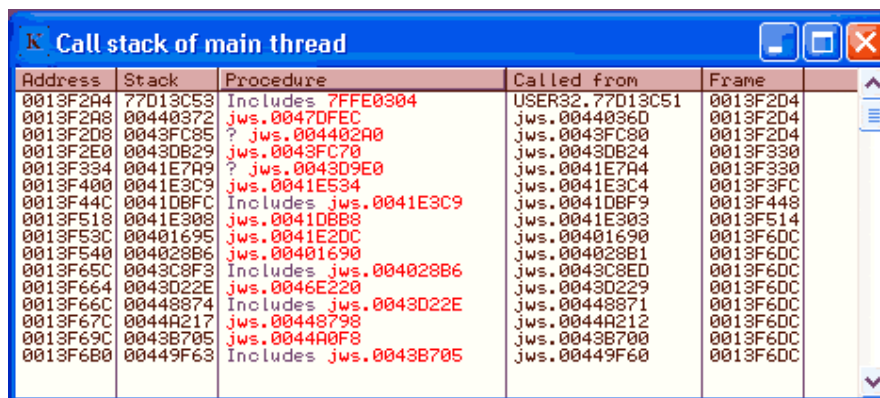


Fig. 14 Call stack at the registration window.

This call stack is useful to find the original point where the registration window is called, you have to do some backtrace until you can find a call where the registration window isn't executed, from this point you've to search some conditional jump to skip this form. During backtracing you are able to reach this place:

+

Address	Disassembly
00411032	59 POP ECX
00411033	5D POP EBP
00411034	C3 RETN
00411035	90 NOP
00411036	90 NOP
00411037	90 NOP
00411038	55 PUSH EBP
00411039	8BEC MOV EBP,ESP
0041103B	53 PUSH EBX
0041103C	56 PUSH ESI
0041103D	57 PUSH EDI
0041103E	8B5D 10 MOV EBX,DWORD PTR SS:[EBP+10]
00411041	8B75 0C MOV ESI,DWORD PTR SS:[EBP+C]
00411044	8B7D 08 MOV EDI,DWORD PTR SS:[EBP+8]
00411047	85FF TEST EDI,EDI
00411049	75 0C JNZ SHORT jws.00411057
0041104B	85F6 TEST ESI,ESI
0041104D	75 08 JNZ SHORT jws.00411057
0041104F	85DB TEST EBX,EBX
00411051	75 04 JNZ SHORT jws.00411057
00411053	33C0 XOR EAX,EAX
00411055	EB 76 JMP SHORT jws.004110CD
00411057	83C6 09 ADD ESI,9
0041105A	B9 0C000000 MOV ECX,0C
0041105F	8BC6 MOV EAX,ESI
00411061	99 CDQ
00411062	F7F9 IDIV ECX

Fig. 15 Trial routine.

If you make this patch the trial will be infinite (the day is not updated), this is not purpose, but are a way to know about the flow, the registration must be before this routine because for a registered program no need arise to check if this is expired ;-)

Address	Disassembly
00411034	C3 RETN
00411035	90 NOP
00411036	90 NOP
00411037	90 NOP
00411038	55 PUSH EBP
00411039	8BEC MOV EBP,ESP
0041103B	53 PUSH EBX
0041103C	56 PUSH ESI
0041103D	57 PUSH EDI
0041103E	8B5D 10 MOV EBX,DWORD PTR SS:[EBP+10]
00411041	8B75 0C MOV ESI,DWORD PTR SS:[EBP+C]
00411044	8B7D 08 MOV EDI,DWORD PTR SS:[EBP+8]
00411047	85FF TEST EDI,EDI
00411049	90 NOP
0041104A	90 NOP
0041104B	33F6 XOR ESI,ESI
0041104D	90 NOP
0041104E	90 NOP
0041104F	33DB XOR EBX,EBX
00411051	90 NOP
00411052	90 NOP
00411053	33C0 XOR EAX,EAX
00411055	EB 76 JMP SHORT jws.004110CD
00411057	83C6 09 ADD ESI,9
0041105A	B9 0C000000 MOV ECX,0C
0041105F	8BC6 MOV EAX,ESI
00411061	99 CDQ
00411062	F7F9 IDIV ECX
00411064	05 8F010000 ADD EAX,18F
00411069	B9 0C000000 MOV ECX,0C
0041106E	03D8 ADD EBX,EAX
00411070	8BC6 MOV EAX,ESI
00411072	99 CDQ
00411073	F7F9 IDIV ECX
00411075	8BC3 MOV EAX,EBX
00411077	89D6 MOV ESI,EDX
00411079	99 CDQ
0041107A	B9 64000000 MOV ECX,64

Make the trial time infinite

Fig. 16 Trial routine patched.

Leave the code untouched and try to other backtrace, to speed this search you can search for each call to 00411038 routine, place a breakpoint on each address and the restart the

program, now OllyDbg must be stop on the first call which is encountered during the program flow:

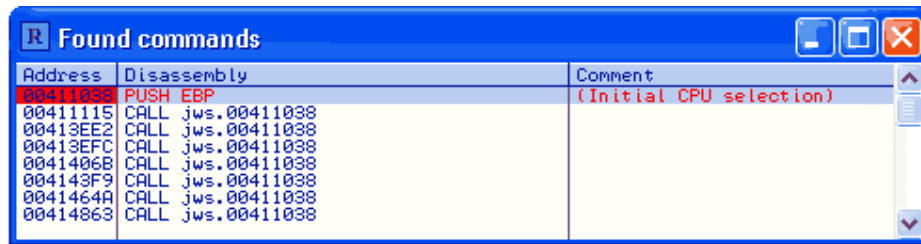


Fig. 17 Search for all the calling.

OllyDbg break in 00411115, this is the first breakpoint:

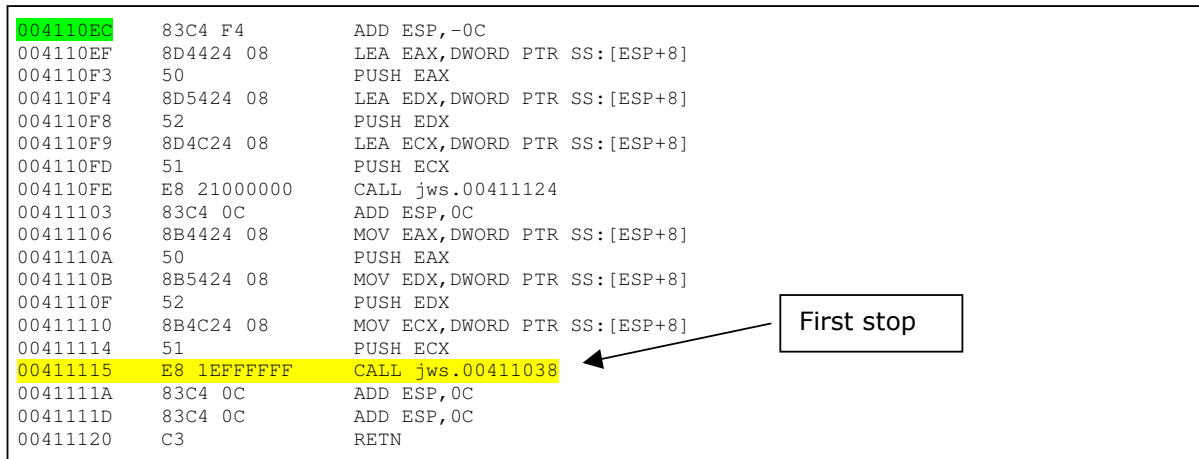


Fig. 18 Search for the first calling.

Well, again you must do some backtracing to find where is possible to skip this control, to do this step with F8 until the RETN instruction, you are into 0041DA42, the next code is a check for trial expiration.

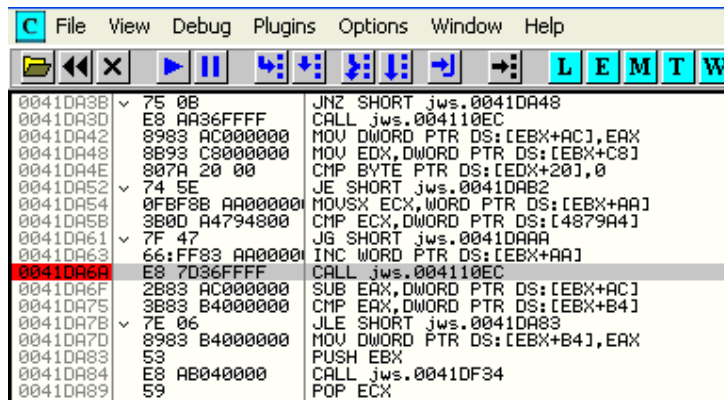


Fig. 19 Search for the registration code.

If you look at 0041DA3B there are a conditional jump, this, if executed, skip the trial control (CALL 004110EC), if you scroll up until 0041DA32 there is another conditional jump, this jump is more interesting because if taken skip all the trial check, then you can think about this control how registration check, hence if [EBX]=[0048DA7C]=1 program is registered otherwise program must run in a trial mode, this is only an hypothesis but can be a starting point.

```

0041DA29 E8 8A000000 CALL jws.0041DAB8
0041DA2E 59 POP ECX
0041DA2F 803B 00 CMP BYTE PTR DS:[EBX],0 ; [EBX]=[0048DA7C]=1 (if registered)
0041DA32 75 7E JNZ SHORT jws.0041DAB2 ; Jump if not equal to zero
0041DA34 83BB AC000000 0>CMP DWORD PTR DS:[EBX+AC],0
0041DA3B 75 0B JNZ SHORT jws.0041DA48
0041DA3D E8 AA36FFFF CALL jws.004110EC
0041DA42 8983 AC000000 MOV DWORD PTR DS:[EBX+AC],EAX
0041DA48 8B93 C8000000 MOV EDX,DWORD PTR DS:[EBX+C8] ; jws.0048C5F4
0041DA4E 807A 20 00 CMP BYTE PTR DS:[EDX+20],0
0041DA52 74 5E JE SHORT jws.0041DAB2 ; Salta se uguale
0041DA54 0FBF8B AA000000 MOVSX ECX,WORD PTR DS:[EBX+AA]
0041DA5B 3B0D A4794800 CMP ECX,DWORD PTR DS:[4879A4]
0041DA61 7F 47 JG SHORT jws.0041DAAA ; Salta se maggiore
0041DA63 66:FF83 AA000000>INC WORD PTR DS:[EBX+AA]
0041DA6A E8 7D36FFFF CALL jws.004110EC
0041DA6F 2B83 AC000000 SUB EAX,DWORD PTR DS:[EBX+AC]
0041DA75 3B83 B4000000 CMP EAX,DWORD PTR DS:[EBX+B4]
0041DA7B 7E 06 JLE SHORT jws.0041DA83
0041DA7D 8983 B4000000 MOV DWORD PTR DS:[EBX+B4],EAX
0041DA83 53 PUSH EBX
0041DA84 E8 AB040000 CALL jws.0041DF34
0041DA89 59 POP ECX
0041DA8A A1 A8794800 MOV EAX,DWORD PTR DS:[4879A8]
0041DA8F 2B83 B4000000 SUB EAX,DWORD PTR DS:[EBX+B4]
0041DA95 8983 B8000000 MOV DWORD PTR DS:[EBX+B8],EAX
0041DA9B 85C0 TEST EAX,EAX
0041DA9D 7D 13 JGE SHORT jws.0041DAB2
0041DA9F 33D2 XOR EDX,EDX
0041DAA1 8993 B8000000 MOV DWORD PTR DS:[EBX+B8],EDX
0041DAA7 5B POP EBX
0041DAA8 5D POP EBP
0041DAA9 C3 RETN
0041DAAA 33C9 XOR ECX,ECX
0041DAAC 898B B8000000 MOV DWORD PTR DS:[EBX+B8],ECX
0041DAB2 5B POP EBX
0041DAB3 5D POP EBP
0041DAB4 C3 RETN

```

Fig. 20 Section code about trial check.

Follow our theory if into 0048DA7C location find 1 program is registered otherwise this is into trial mode, well, you can place a breakpoint on memory to look how function write/read at this address before to reach the control routine in 0041DA2F. To do this restart OllyDbg and the stop this into OEP, place a breakpoint on memory access in 0048DA7C and run the program. OllyDbg stop in 00401022, this code is only to initialize this memory area, there are no punctual control into our address then nothing is needed:

```

00401022 F3:AA REP STOS BYTE PTR ES:[EDI]
00401024 59 POP ECX
00401025 5F POP EDI
00401026 52 PUSH EDX
00401027 6A 00 PUSH 0
00401029 E8 CAC70700 CALL <JMP.&kernel32.GetModuleHandleA>
0040102E 8BD0 MOV EDX,EAX
00401030 E8 E7F10600 CALL jws.0047021C
00401035 5A POP EDX
00401036 6A 00 PUSH 0
00401038 E8 D7010700 CALL jws.00471214
0040103D 59 POP ECX
0040103E 68 74F24700 PUSH jws.0047F274
00401043 6A 00 PUSH 0
00401045 E8 AEC70700 CALL <JMP.&kernel32.GetModuleHandleA>
0040104A A3 B4F24700 MOV DWORD PTR DS:[47F2B4],EAX
0040104F 6A 00 PUSH 0
00401051 E9 B6780700 JMP jws.0047890C
00401056 E9 2D020700 JMP jws.00471288
0040105B 33C0 XOR EAX,EAX
0040105D A0 A1F24700 MOV AL,BYTE PTR DS:[47F2A1]
00401062 C3 RETN

```

Fig. 21 First stop.

Run again the program (push F9), OllyDbg stop in 0041DAC0:

0041DAB8	55	PUSH EBP
0041DAB9	8BEC	MOV EBP,ESP
0041DABB	53	PUSH EBX
0041DABC	56	PUSH ESI
0041DABD	8B5D 08	MOV EBX,DWORD PTR SS:[EBP+8]
0041DAC0	C603 00	MOV BYTE PTR DS:[EBX],0
0041DAC3	6A 52	PUSH 52
0041DAC5	8D43 01	LEA EAX,DWORD PTR DS:[EBX+1]
0041DAC8	50	PUSH EAX
0041DAC9	E8 72000000	CALL jws.0041DB40
0041DACE	83C4 08	ADD ESP,8
0041DAD1	6A 52	PUSH 52
0041DAD3	8D73 54	LEA ESI,DWORD PTR DS:[EBX+54]
0041DAD6	56	PUSH ESI
0041DAD7	E8 64000000	CALL jws.0041DB40
0041DADC	83C4 08	ADD ESP,8
0041DADE	6A 50	PUSH 50
0041DAE1	56	PUSH ESI
0041DAE2	E8 ED050000	CALL jws.0041E0D4
0041DAE7	66:C783 A8000000	>MOV WORD PTR DS:[EBX+A8],1
0041DAF0	66:C783 AA000000	>MOV WORD PTR DS:[EBX+AA],0
0041DAF9	33C0	XOR EAX,EAX
0041DAFB	33D2	XOR EDX,EDX
0041DAFD	8983 AC000000	MOV DWORD PTR DS:[EBX+AC],EAX
0041DB03	8993 B0000000	MOV DWORD PTR DS:[EBX+B0],EDX
0041DB09	33C9	XOR ECX,ECX
0041DB0B	33C0	XOR EAX,EAX
0041DB0D	898B B4000000	MOV DWORD PTR DS:[EBX+B4],ECX
0041DB13	8983 B8000000	MOV DWORD PTR DS:[EBX+B8],EAX
0041DB19	83C4 08	ADD ESP,8
0041DB1C	C683 BC000000	>MOV BYTE PTR DS:[EBX+BC],0
0041DB23	6A 0A	PUSH 0A
0041DB25	81C3 BD000000	ADD EBX,0BD
0041DB2B	6A 00	PUSH 0
0041DB2D	53	PUSH EBX
0041DB2E	E8 D5270500	CALL jws.00470308
0041DB33	83C4 0C	ADD ESP,0C
0041DB36	B8 01000000	MOV EAX,1
0041DB3B	5E	POP ESI
0041DB3C	5B	POP EBX
0041DB3D	5D	POP EBP
0041DB3E	C3	RETN

Fig. 22 Second stop.

Well, this nice piece of code write a 0 into our address, you can change this with 1 then:

```
0041DAC0    C603 00          MOV BYTE PTR DS:[EBX],0
```

has to be replaced with:

```
0041DAC0    C603 01          MOV BYTE PTR DS:[EBX],1
```

If you run again the program OllyStop on for memory writing on 0041DD4A:

```
0041DD4A    C607 00          MOV BYTE PTR DS:[EDI],0
```

has to be replaced with:

```
0041DD4A    C607 01          MOV BYTE PTR DS:[EDI],1
```

Run the program, again OllyDbg stop for memory writing on 00411524.

0041150C	55	PUSH EBP
0041150D	8BEC	MOV EBP,ESP
0041150F	53	PUSH EBX
00411510	56	PUSH ESI
00411511	8B75 10	MOV ESI,DWORD PTR SS:[EBP+10]
00411514	33C0	XOR EAX,EAX
00411516	8B55 0C	MOV EDX,DWORD PTR SS:[EBP+C]
00411519	8BCA	MOV ECX,EDX
0041151B	8B55 08	MOV EDX,DWORD PTR SS:[EBP+8]
0041151E	3BF0	CMP ESI,EAX
00411520	7E 0B	JLE SHORT jws-fixe.0041152D
00411522	8A1A	MOV BL,BYTE PTR DS:[EDX]
00411524	8B19	MOV BYTE PTR DS:[ECX],BL
00411526	40	INC EAX
00411527	41	INC ECX
00411528	42	INC EDX
00411529	3BF0	CMP ESI,EAX
0041152B	7F F5	JG SHORT jws-fixe.00411522
0041152D	5E	POP ESI
0041152E	5B	POP EBX
0041152F	5D	POP EBP
00411530	C3	RETN

Fig 23 The last stop.

We have to force a writing for 1, then BL must be forced at 1 then try this patch:

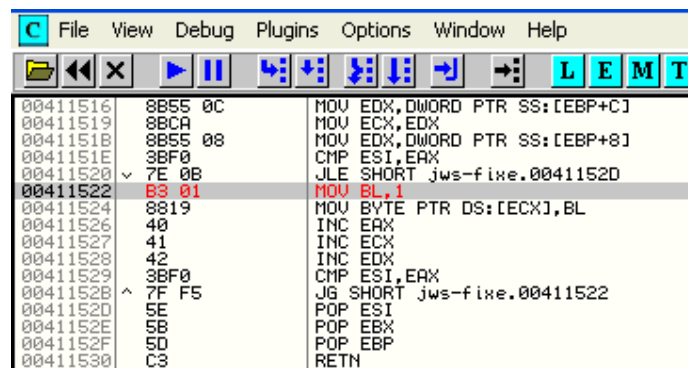


Fig 24 The last patch.

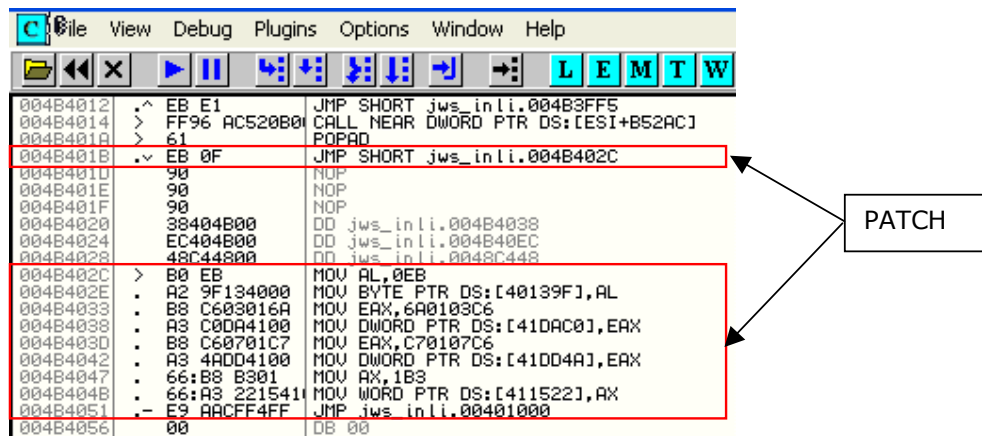
If you press F9 to run the program no registration window is showed, program is registered without entering any serial, cool, fix the executable and the save it, work done!
Go to the Help menu and then select about About:



Fig. 25 Registration form.

STEP 6 – Debugging and patching stage

This patching is useful to understand memory breakpoint and how to use it, but a best work is made by inline patching, to do this I've to find a free area (remember: some byte are available also into the PE Header) and build your patcher, then you've to change the jump before to reach the OEP into another jump to your patcher code, then, after this go to the OEP:



```

File View Debug Plugins Options Window Help
[Icons] [LEMTW]
004B4012 ^ EB E1 JMP SHORT jws_inli.004B3FF5
004B4014 > FF96 AC520B01 CALL NEAR DWORD PTR DS:[ESI+B52AC]
004B4018 > 61 POPAD
004B401B ^ EB 0F JMP SHORT jws_inli.004B402C
004B401D 90 NOP
004B401E 90 NOP
004B401F 90 NOP
004B4020 38404B00 DD jws_inli.004B4038
004B4024 EC404B00 DD jws_inli.004B40EC
004B4028 48C44B00 DD jws_inli.004B4048
004B402C > B0 EB MOV AL,0EB
004B402E . A2 9F134000 MOV BYTE PTR DS:[40139F],AL
004B4033 . B8 C603016A MOV EAX,6A0103C6
004B4038 . A3 C0DA4100 MOV DWORD PTR DS:[41DAC0],EAX
004B403D . B8 C60701C7 MOV EAX,C70107C6
004B4042 . A3 4ADD4100 MOV DWORD PTR DS:[41DD4A],EAX
004B4047 . 66:B8 B301 MOV AX,1B3
004B404B . 66:A3 221541 MOV WORD PTR DS:[411522],AX
004B4051 . E9 BACFF4FF JMP jws_inli.00401000
004B4056 00 DB 00
  
```

Fig. 26 in line patching.

No matter if the patcher code can be overwritten by the program itself, because the patch is build into the code, then there is no need about the patcher code after you jump into the OEP.

What is the advantage with this patch mode, well, size of program is unchanged then you haven't to send the unpacked patched executable, you can write own patcher (with a few bytes) to apply the patch directly on original packed file.

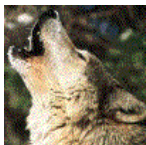
The end of story

Remember, this is only for study and own know how about protection scheme, in this program a simple protection is made.

Remember, if you plan to use this software you must purchase the product in order to support the author to develop other good software.

Any suggest, correction or criticism is welcome.

Greetings from



ThunderPwr

Remarks

Thanks at all the ARTeam and all members of this fantastic forum