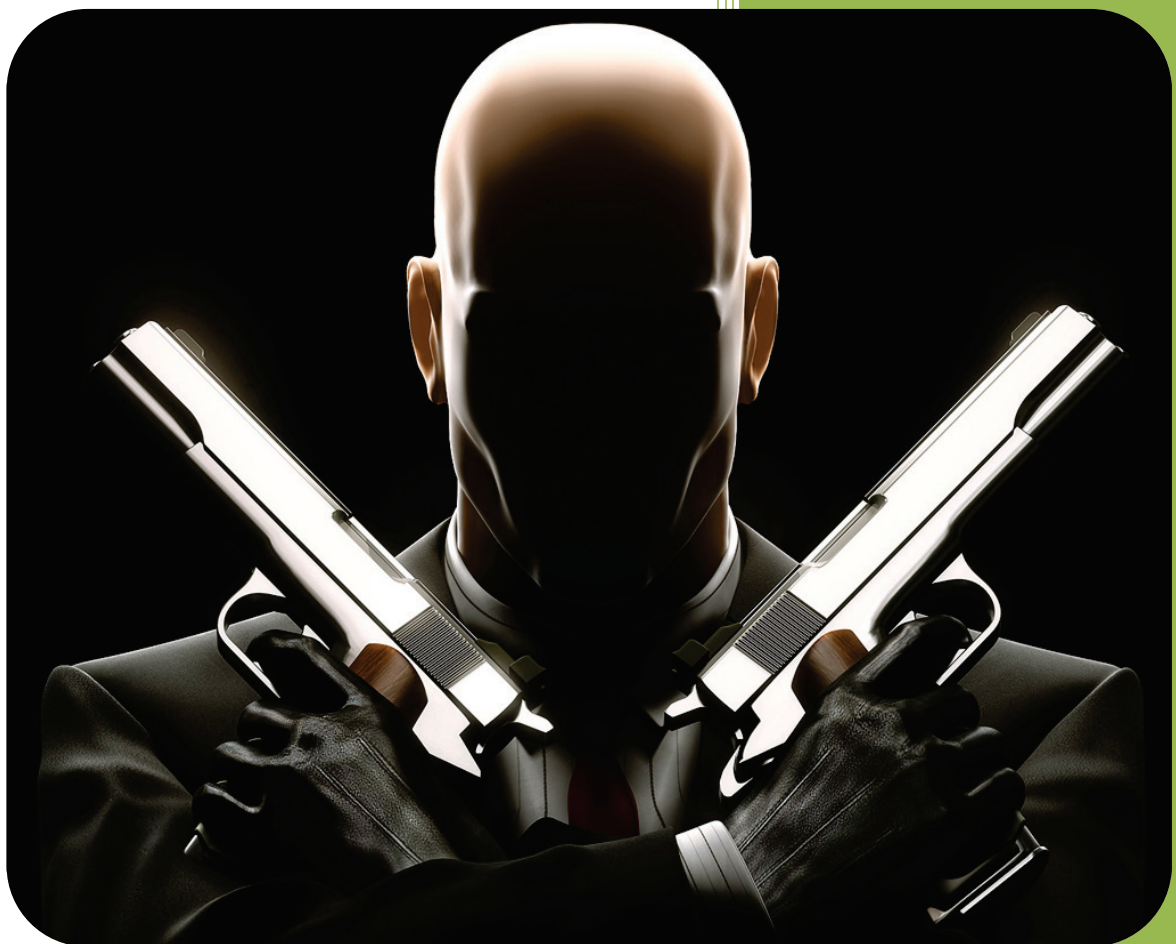




2009

Serial Fishing and Creating a Self Registering Program



R@dier

ARTeam

June 2009

DISCLAIMER

All code included with this tutorial is free to use and modify; we only ask that you mention where you found it. This tutorial is also free to distribute in its current unaltered form, with all the included supplements.

All the commercial programs used within this tutorial have been used only for the purpose of demonstrating the theories and methods described. No distribution of patched applications has been done under any media or host. The applications used were most of the times already been patched by other fellows, and cracked versions were available since a lot of time. ARTeam or the authors of the papers shouldn't be considered responsible for damages to the companies holding rights on those programs. The scope of this document as well as any other ARTeam tutorial is of sharing knowledge and teaching how to patch applications, how to bypass protections and generally speaking how to improve the RCE art. We are not releasing any cracked application.

VERIFICATION

ARTeam.esfv can be opened in the ARTeamESFVChecker to verify all files have been released by ARTeam and are unaltered. The ARTeamESFVChecker can be obtained in the release section of the ARTeam site: <http://releases.accessroot.com>



TABLE OF CONTENTS

Disclaimer	2
Verification	2
1 APPROACHING THE PROBLEM	4
Introduction.....	4
Starting the homework.....	4
2 CODE INJECTION	11
Step 1: Find a suitable location	11
Step 2: Backup the original code	12
Step3: Inject some code.....	12
Test the injected code works:	14
Creating the loader for code injection and self registration:	15
Conclusion	16
Greetings	16
History	16

1 APPROACHING THE PROBLEM

INTRODUCTION

I noticed that the approach to make a target program self registering has not been discussed for quite some time so I am adding this tutorial to the ARTeam beginner olly tutes

What you will need for this tutorial is:

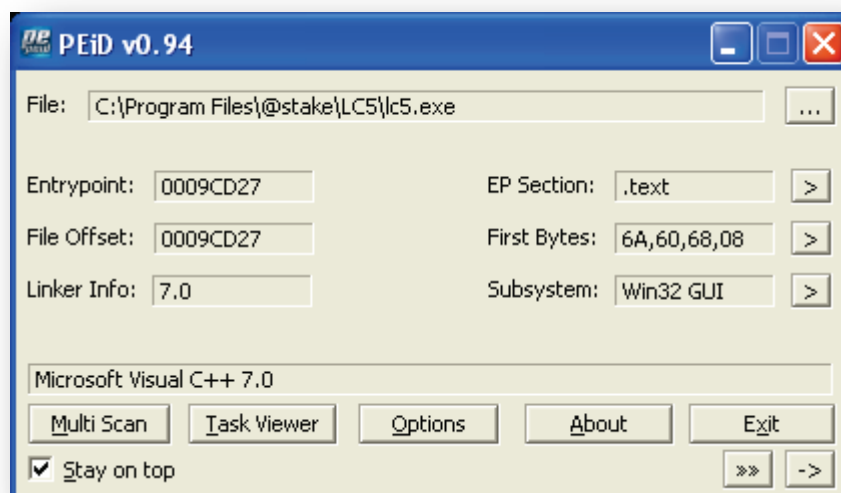
1. OllyDbg 1.10
2. PeID
3. Some assembly knowledge

Today's target for this demo is LC5 a Password and Auditing tools.

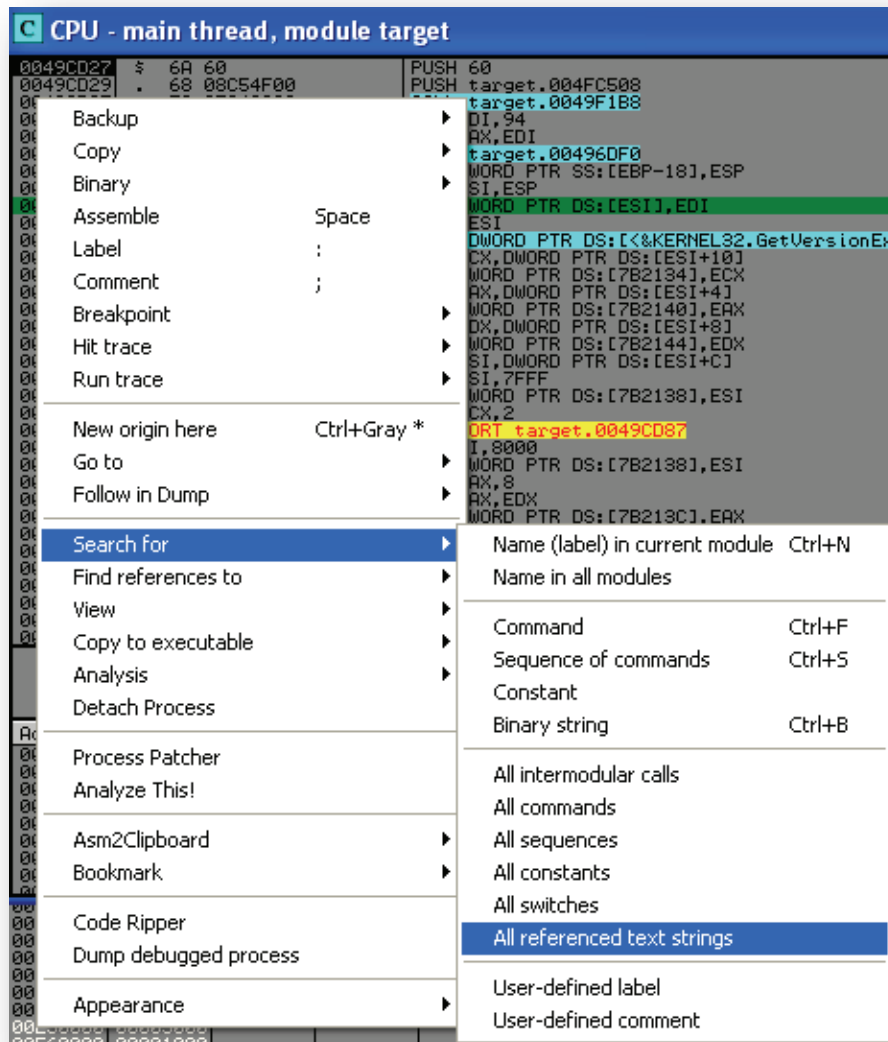
STARTING THE HOMEWORK

Ok let's get started

Startup PeID and check the program for any protector's/ packers, as we can see the program is clear of any protections so let's get started



Open up the target program in ollyDbg and you should land here, the first thing we are going to do is check for any text string that can give us a starting point in the application. To do this right click the screen and go to Search for → All referenced text Strings

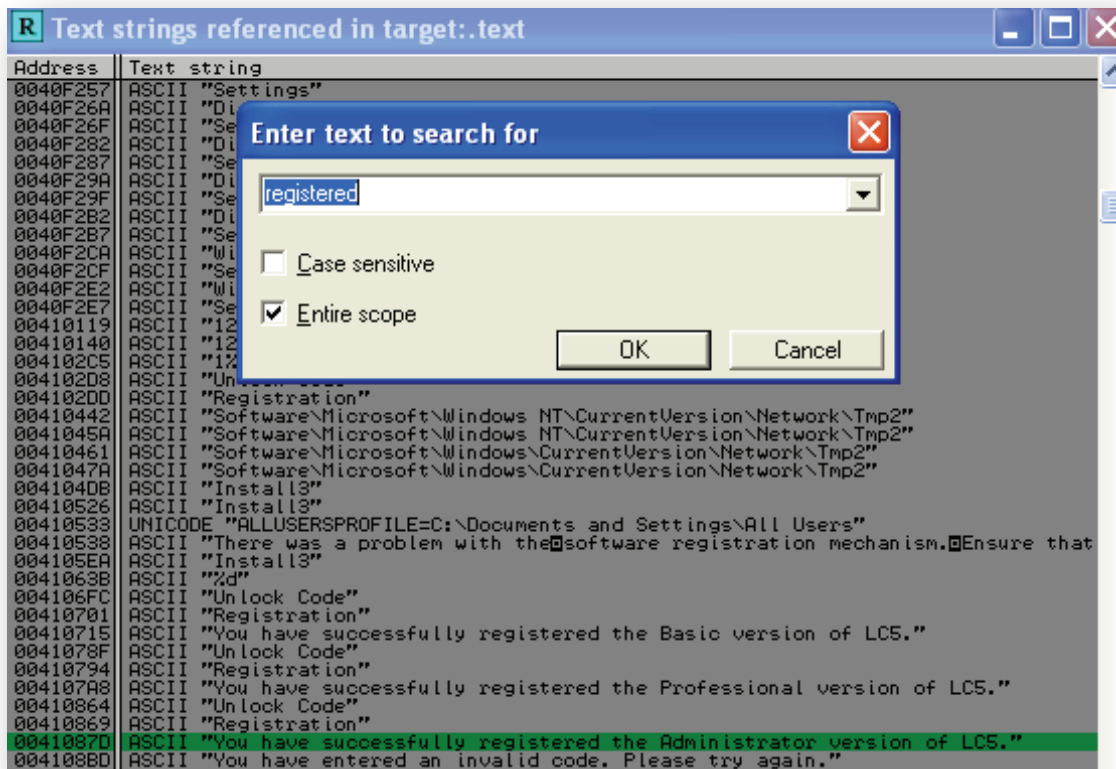


Once in the code window, do a CTRL + L to bring up the search window, type in **registered** then press ok

You should end up around here. To keep searching for the next instance of the search word you can keep pressing **CTRL + L**

If you look down at the highlighted line you can see there are three types of registrations available, Basic, Professional and Administrator.

We are interested in the best option so double click on the **"You have successfully registered the administrator version of LC5"**



You will land here:

0041082A	85C0	TEST EAX, EAX	
0041082C	53	PUSH EBX	
0041082D	0F85 7F000000	JNZ target.004108B2	
00410833	8D4C24 38	LEA ECX, DWORD PTR SS:[ESP+38]	
00410837	E8 74D00000	CALL target.0041C5B0	
0041083C	8D4C24 34	LEA ECX, DWORD PTR SS:[ESP+34]	
00410840	C68424 80040000 07	MOV BYTE PTR SS:[ESP+480], 7	
00410848	C78424 A8000000 0100	MOV DWORD PTR SS:[ESP+A8], 1	
00410853	E8 A8B60900	CALL target.004ABF00	
00410858	33F8 01	CMPEB EAX, 1	
0041085B	0F85 9B000000	JNZ target.004108FC	
00410861	8B07	MOV EAX, DWORD PTR DS:[EDI]	
00410863	50	PUSH EAX	
00410864	68 04664D00	PUSH target.004D6604	Arg3
00410869	68 F4654D00	PUSH target.004D65F4	Arg2 = 004D6604 ASCII "Unlock Code"
0041086E	8BCE	MOV ECX, ESI	Arg1 = 004D65F4 ASCII "Registration"
00410870	899E 94010000	MOV DWORD PTR DS:[ESI+194], EBX	
00410876	E8 ABCF0A00	CALL target.004BD826	target.004BD826
0041087B	53	PUSH EBX	Arg3
0041087C	53	PUSH EBX	Arg2
0041087D	68 08644D00	PUSH target.004D6408	Arg1 = 004D6408 ASCII "You have successfully registered the Administrator"
00410882	C786 A4010000 030000	MOV DWORD PTR DS:[ESI+1A4], 3	
0041088C	E8 47CD0A00	CALL target.004BD5D8	target.004BD5D8
00410891	33C0	XOR EAX, EAX	
00410893	894424 10	MOV DWORD PTR SS:[ESP+10], EAX	
00410897	894424 14	MOV DWORD PTR SS:[ESP+14], EAX	
0041089B	8D4C24 34	LEA ECX, DWORD PTR SS:[ESP+34]	
0041089F	884424 18	MOV BYTE PTR SS:[ESP+18], AL	
004108A3	C68424 80040000 04	MOV BYTE PTR SS:[ESP+480], 4	
004108A8	E8 5056FFFF	CALL target.00405F00	
004108B0	E8 25	JMP SHORT target.004108D7	
004108B2	33C9	XOR ECX, ECX	
004108B4	894C24 14	MOV DWORD PTR SS:[ESP+14], ECX	
004108B8	53	PUSH EBX	Arg2
004108B9	894C24 1C	MOV DWORD PTR SS:[ESP+1C], ECX	
004108BD	68 D4634D00	PUSH target.004D63D4	Arg1 = 004D63D4 ASCII "You have entered an invalid code. Please try again."
004108C2	884C24 24	MOV BYTE PTR SS:[ESP+24], CL	
004108C6	E8 0DCD0A00	CALL target.004BD5D8	target.004BD5D8

From the code above we can see there are two cases that can occur,

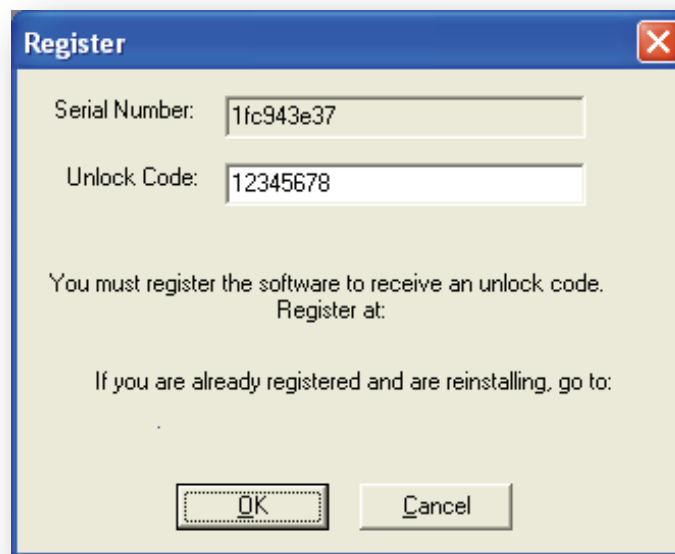
a successful registration or an **"You have entered an invalid code"** message.

Let's look at the code to see where this decision was made. We can see there is a test and a Jump not zero at 0041082D, this jumps to the start of our invalid code message, a place we don't want to be, Lets set a breakpoint on the jump. Click on 0041082D and press **F2** to set the break point

0041081C	. 8D5424 18	LEA EDX,DWORD PTR SS:[ESP+18]	
00410820	. 52	PUSH EDX	
00410821	. 50	PUSH EAX	
00410822	. E8 5F8C0000	CALL target.00499486	
00410827	. 83C4 10	ADD ESP,10	
0041082A	. 85C0	TEST EAX,EAX	
0041082C	. 53	PUSH EBX	
0041082D	~ 0F85 7F000000	JNZ target.004108B2	This is where the good /Bad Registration number disision occurs Lets set a break point here and take a look
00410833	. 8D4C24 38	LEA ECX,DWORD PTR SS:[ESP+38]	
00410837	. E8 74BD0000	CALL target.0041C5B0	
0041083C	. 8D4C24 34	LEA ECX,DWORD PTR SS:[ESP+34]	
00410840	. C68424 80040000 07	MOV BYTE PTR SS:[ESP+48],7	
00410848	. C78424 A8000000 0100	MOV DWORD PTR SS:[ESP+A8],1	
00410853	. E8 A8B60900	CALL target.004ABF00	
00410858	. 83F8 01	CMP EAX,1	
0041085B	~ 0F85 9B000000	JNZ target.004108FC	
00410861	. 8B07	MOV EAX,DWORD PTR DS:[EDI]	
00410863	. 50	PUSH EAX	
00410864	. 68 04664D00	PUSH target.004D6604	Arg3
00410869	. 68 F4654D00	PUSH target.004D65F4	Arg2 = 004D6604 ASCII "Unlock Code"
0041086E	. 8BCE	MOV ECX,ESI	Arg1 = 004D65F4 ASCII "Registration"
00410870	. 899E 94010000	MOV DWORD PTR DS:[ESI+194],EBX	
00410876	. E8 ABCF0A00	CALL target.004BD826	target.004BD826
0041087B	. 53	PUSH EBX	Arg3
0041087C	. 53	PUSH EBX	Arg2
0041087D	. 68 08644D00	PUSH target.004D6408	Arg1 = 004D6408 ASCII "You have success"
00410882	. C786 A4010000 030000	MOV DWORD PTR DS:[ESI+1A4],3	
0041088C	. E8 47CD0A00	CALL target.004BD5D8	target.004BD5D8
00410891	. 33C0	XOR EAX,EAX	
00410893	. 894424 10	MOV DWORD PTR SS:[ESP+10],EAX	
00410897	. 894424 14	MOV DWORD PTR SS:[ESP+14],EAX	
0041089B	. 8D4C24 34	LEA ECX,DWORD PTR SS:[ESP+34]	
0041089F	. 884424 18	MOV BYTE PTR SS:[ESP+18],AL	
004108A3	. C68424 80040000 04	MOV BYTE PTR SS:[ESP+48],4	
004108AB	. E8 5056FFFF	CALL target.00405F00	
004108B0	~ EB 25	JMP SHORT target.004108D7	
004108B2	> 33C9	XOR ECX,ECX	
004108B4	. 894C24 14	MOV DWORD PTR SS:[ESP+14],ECX	
004108B8	. 53	PUSH EBX	Arg2
004108B9	. 894C24 1C	MOV DWORD PTR SS:[ESP+1C],ECX	
004108BD	. 68 D4634D00	PUSH target.004D63D4	Arg1 = 004D63D4 ASCII "You have entered"
004108C2	. 884C24 24	MOV BYTE PTR SS:[ESP+24],CL	

We will now run the program and using **F9** and where we are presented with the nag screen

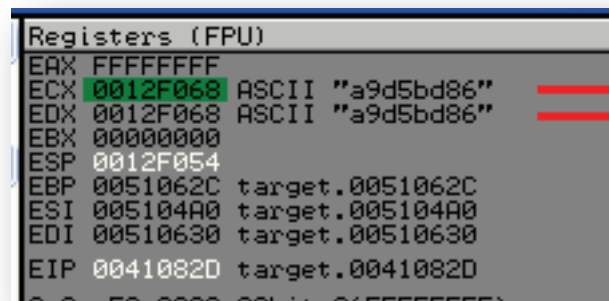
Enter: **12345678** and then click **OK**



You should break on the JNZ, no at this point let's examine the registers:

We can see what looks to be a registration code in ECX and EDX;

This is very interesting as the test on EAX has the VALUE FFFFFFFF.



Let's trace back and have and have a look how the codes got here

We can see that EAX and EDX where our registration code was found is being set.

Let's set a breakpoint on 0041080C By selecting this address and pressing F2.

004107DF	83CA FF	OR EDX,FFFFFFFF	
004107E2	F0:0FC111	LOCK XADD DWORD PTR DS:[ECX],EDX	LOCK prefix
004107E6	4A	DEC EDX	
004107E7	85D2	TEST EDX,EDX	
004107E9	7F 08	JG SHORT target.004107F3	
004107EB	8B08	MOV ECX,DWORD PTR DS:[EAX]	
004107ED	8B11	MOV EDX,DWORD PTR DS:[ECX]	
004107EF	50	PUSH EAX	
004107F0	FF52 04	CALL DWORD PTR DS:[EDX+4]	
004107F3	8D8C24 24010000	LEA ECX,DWORD PTR SS:[ESP+124]	
004107FA	C68424 80040000 04	MOV BYTE PTR SS:[ESP+480],4	
00410802	E8 C9AF0900	CALL target.004AB7CF	
00410807	E9 CB000000	JMP target.004108D7	
0041080C	8B45 00	MOV EAX,DWORD PTR SS:[EBP]	<p>We can see EAX and ECX being set and then pushed onto the stack, This is interesting so we will set a break point on 0041080C and see what happening</p>
0041080F	8D4C24 10	LEA ECX,DWORD PTR SS:[ESP+10]	
00410813	51	PUSH ECX	
00410814	50	PUSH EAX	
00410815	E8 5636FFFF	CALL target.00403E70	
0041081A	8B07	MOV EAX,DWORD PTR DS:[EDI]	
0041081C	8D5424 18	LEA EDX,DWORD PTR SS:[ESP+18]	
00410820	52	PUSH EDX	
00410821	50	PUSH EAX	
00410822	E8 5F8C0800	CALL target.00499486	
00410827	83C4 10	ADD ESP,10	
0041082A	85C0	TEST EAX,EAX	
0041082C	53	PUSH EBX	
0041082D	0F85 7F000000	JNZ target.004108B2	
00410833	8D4C24 38	LEA ECX,DWORD PTR SS:[ESP+38]	
00410837	E8 74BD0000	CALL target.0041C5B0	
0041083C	8D4C24 34	LEA ECX,DWORD PTR SS:[ESP+34]	
00410840	C68424 80040000 07	MOV BYTE PTR SS:[ESP+480],7	
00410848	C78424 A8000000 0100	MOV DWORD PTR SS:[ESP+A8],1	
00410853	E8 A8B60900	CALL target.004ABF00	
00410858	83F8 01	CMF EAX,1	
0041085B	0F85 9B000000	JNZ target.004108FC	

Next run the program with F9 and when the invalid message occurs just click ok

Then when the registration screen re-appears click ok again, you should land on your breakpoint.

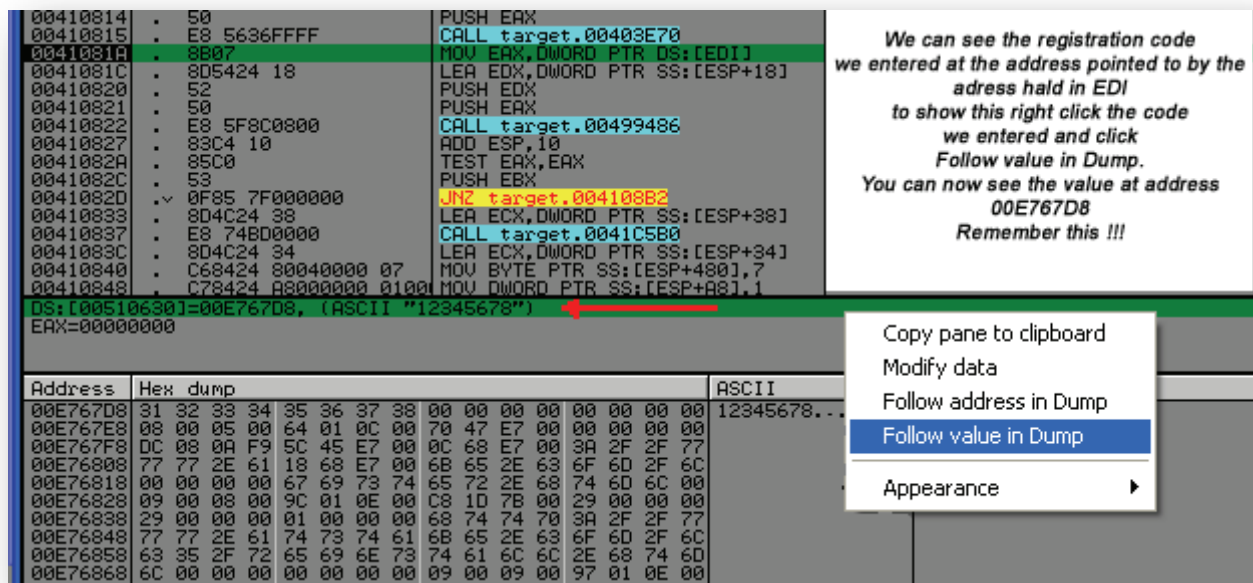
Now as we step over the code with F8 we can see in the registers **EAX**, now holds the serial number form our registration nag screen: **1fc943e37**

The call at **00410815 CALL target.00403E70** goes off and calculates a new Administrator version registration number. Step over this call using **F8**

Now things get interesting, We can see the registration code we entered at the address pointed to by the address held in EDI to show this right click the code we entered and click Follow value in Dump.

You can now see the value at address 00E767D8.

Remember this we will use this later.



We can see the registration code we entered at the address pointed to by the address held in EDI to show this right click the code we entered and click Follow value in Dump. You can now see the value at address 00E767D8 Remember this !!!

Address	Hex dump	ASCII
00E767D8	31 32 33 34 35 36 37 38 00 00 00 00 00 00 00 00	12345678...
00E767E8	08 00 05 00 64 01 0C 00 70 47 E7 00 3A 2F 2F 77	
00E767F8	DC 08 0A F9 5C 45 E7 00 0C 68 E7 00 3A 2F 2F 77	
00E76808	77 77 2E 61 18 68 E7 00 68 65 2E 63 6F 6D 2F 6C	
00E76818	00 00 00 00 67 69 73 74 65 72 2E 68 74 6D 6C 00	
00E76828	09 00 08 00 9C 01 0E 00 C8 10 7B 00 29 00 00 00	
00E76838	29 00 00 00 01 00 00 00 68 74 74 70 3A 2F 2F 77	
00E76848	77 77 2E 61 74 73 74 61 68 65 2E 63 6F 6D 2F 6C	
00E76858	63 35 2F 72 65 69 6E 73 74 61 6C 6C 2E 68 74 6D	
00E76868	6C 00 00 00 00 00 00 00 09 00 09 00 97 01 0E 00	

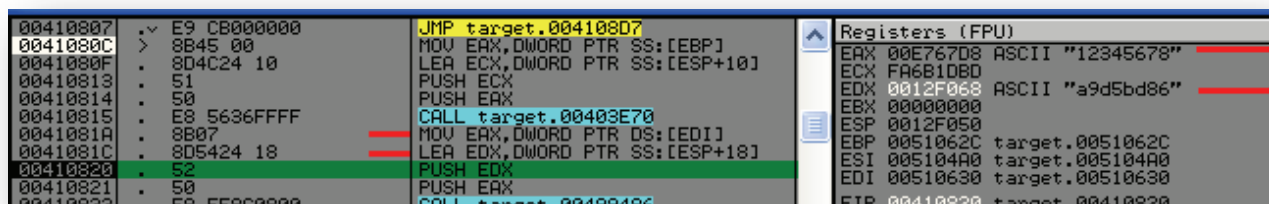
After we have stepped over the:

0041081A MOV EAX,DWORD PTR DS:[EDI]

0041081C LEA EDX,DWORD PTR SS:[ESP+18]

Check what has been left in **EAX** and **EDX**

It is our entered registration code and the real registration code.



Registers (FPU)
EAX 00E767D8 ASCII "12345678"
ECX FA6B1DBD
EDX 0012F068 ASCII "a9d5bd86"
EBX 00000000
ESP 0012F050
EBP 0051062C target.0051062C
ESI 005104A0 target.005104A0
EDI 00510630 target.00510630
EIP 00410820 target.00410820

Hmmm. Now we have the real code we no longer need the one we entered and in fact we do not need to enter a code at all to make this program register it's self.

Ok now to the fun part, **Code Injection time** this is where the assembler comes into it

2 CODE INJECTION

We now want to make this program self registering to do this we need to inject some code into the application.

STEP 1: FIND A SUITABLE LOCATION

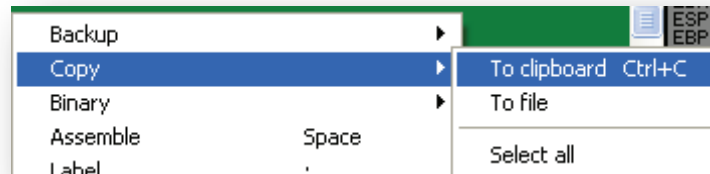
The best place I can see to inject our code is after our values have been copied into **EDX** and **EAX** And then again at the start of the **"Invalid code"** message at **004108B2**.

Our plan is to jump to this section and place the bulk of our code from this point

00410813	. 51	PUSH ECX	<i>The best place I can see to inject our code is after our values have been copied into EDX and EAX And then again at the start of the "Invalid code" message at 004108B2 Our plan is to jump to this section and place the bulk of our code from this point</i>
00410814	. 50	PUSH EAX	
00410815	. E8 5636FFFF	CALL target.00403E70	
0041081A	. 8B07	MOV EAX,DWORD PTR DS:[EDI]	
0041081C	. 8D5424 18	LEA EDX,DWORD PTR SS:[ESP+18]	
00410820	. 52	PUSH EDX	
00410821	. 50	PUSH EAX	
00410822	. E8 5F8C0800	CALL target.00499486	
00410827	. 83C4 10	ADD ESP,10	
0041082A	. 85C0	TEST EAX,EAX	
0041082C	. 53	PUSH EBX	
0041082D	. 0F85 7F000000	JNZ target.004108B2	
00410833	. 8D4C24 38	LEA ECX,DWORD PTR SS:[ESP+38]	
00410837	. E8 74BD0000	CALL target.0041C5B0	
0041083C	. 8D4C24 34	LEA ECX,DWORD PTR SS:[ESP+34]	
00410840	. C68424 80040000 07	MOV BYTE PTR SS:[ESP+48],7	
00410848	. C78424 A8000000 0100	MOV DWORD PTR SS:[ESP+A8],1	
00410853	. E8 A8B60900	CALL target.004ABF00	
00410858	. 83F8 01	CMP EAX,1	
0041085B	. 0F85 9B000000	JNZ target.004108FC	
00410861	. 8B07	MOV EAX,DWORD PTR DS:[EDI]	
00410863	. 50	PUSH EAX	Arg3
00410864	. 68 04664D00	PUSH target.004D6604	Arg2 = 004D6604 ASCII "Unlock Code"
00410869	. 68 F4654D00	PUSH target.004D65F4	Arg1 = 004D65F4 ASCII "Registration"
0041086E	. 8BCE	MOV ECX,ESI	
00410870	. 899E 94010000	MOV DWORD PTR DS:[ESI+194],EBX	
00410876	. E8 ABCF0A00	CALL target.004BD0826	target.004BD0826
0041087B	. 53	PUSH EBX	Arg3
0041087C	. 53	PUSH EBX	Arg2
0041087D	. 68 08644D00	PUSH target.004D6408	Arg1 = 004D6408 ASCII "You have successfully registered"
00410882	. C786 A4010000 030000	MOV DWORD PTR DS:[ESI+1A4],3	
0041088C	. E8 47CD0A00	CALL target.004BD05D8	target.004BD05D8
00410891	. 33C0	XOR EAX,EAX	
00410893	. 894424 10	MOV DWORD PTR SS:[ESP+10],EAX	
00410897	. 894424 14	MOV DWORD PTR SS:[ESP+14],EAX	
0041089B	. 8D4C24 34	LEA ECX,DWORD PTR SS:[ESP+34]	
0041089F	. 8B4424 18	MOV BYTE PTR SS:[ESP+18],AL	
004108A3	. C68424 80040000 04	MOV BYTE PTR SS:[ESP+48],4	
004108AB	. E8 5056FFFF	CALL target.00405F00	
004108B0	. EB 25	JMP SHORT target.004108D7	
004108B2	. 33C9	XOR ECX,ECX	
004108B4	. 894C24 14	MOV DWORD PTR SS:[ESP+14],ECX	
004108B8	. 53	PUSH EBX	Arg2
004108B9	. 894C24 1C	MOV DWORD PTR SS:[ESP+1C],ECX	
004108BD	. 68 D4634D00	PUSH target.004D63D4	Arg1 = 004D63D4 ASCII "You have entered an invalid code"
004108C2	. 8B4C24 24	MOV BYTE PTR SS:[ESP+24],CL	
004108C6	. E8 0DCD0A00	CALL target.004BD05D8	target.004BD05D8
004108CB	. 68 1C474D00	PUSH target.004D471C	

STEP 2: BACKUP THE ORIGINAL CODE

Our jump is about to overwrite a considerable section of code and we will need to be able to put this back as required. The easiest way is to highlight the green section and right click and copy to clip board, the copy to notepad.



STEP3: INJECT SOME CODE

The first thing is we will need to work out what we want to do with the injected code, it always helps to note down some sudo code of our process.

What do we need to do is:

1. Save the registers state
2. Jump to our code
3. Replace our registration code with the real one
4. Restore the registers
5. Restore original required code
6. Jump back to normal program execution
7. Now for the assembly part:

We will start at: 00410820

```
PUSHAD ; Save the registers
JMP 004108B2 ; Jump to our code
```

At 004108B2 we now continue our code we are going to do some string operations so we need to:

```
MOV EDI,EAX ; copy our target address to EDI
MOV ESI,EDX ; copy real code address to ESI
MOV ECX,9 ; ECX is used as a counter for length of the string
REP MOVSB ; MOVED STRING BYTES
```

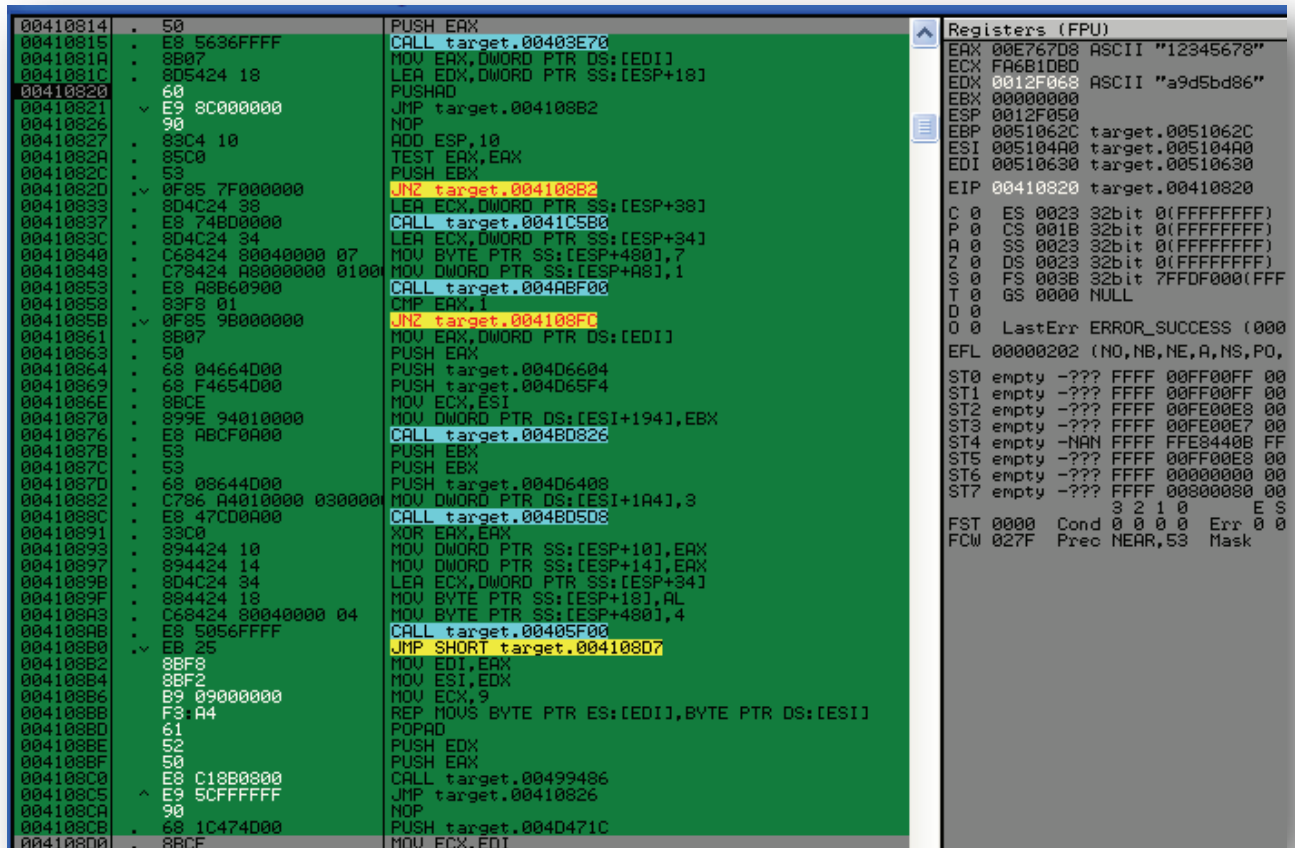
Note: This operand will translate to **REP MOVSB BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]** which means it will copy the real registration code stored in the address at ESI to our code stored in the address at EDI. The number of bytes it will copy is stored in ECX.

```
POPAD ; Restore the registers and restore original code
PUSH EDX; puts the real code on the stack for the validation function
```

PUSH EAX; puts our replaced real registration code onto the stack
 CALL 00499486; Call validation routine
 JMP 00410826; jump back to the real program and continue normal execution

Your injected code should look like this highlighted in while.

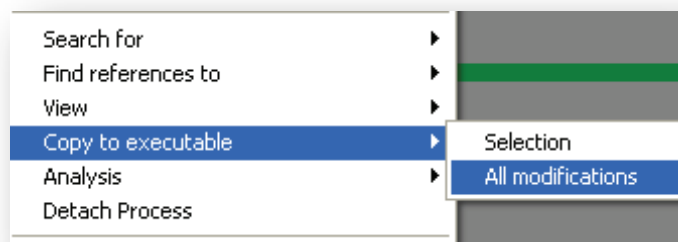
What is very interesting is when you are at 004108BB is to watch EAX change byte by byte to our real registration code as ECX decrements.



Address	Disassembly
00410814	PUSH EAX
00410815	CALL target.00403E70
0041081A	MOV EAX, DWORD PTR DS:[EDI]
0041081C	LEA EDX, DWORD PTR SS:[ESP+18]
00410820	PUSHAD
00410821	JMP target.004108B2
00410826	NOP
00410827	ADD ESP, 10
0041082A	TEST EAX, EAX
0041082C	PUSH EBX
0041082D	JNZ target.004108B2
00410833	LEA ECX, DWORD PTR SS:[ESP+38]
00410837	CALL target.0041C5B0
0041083C	LEA ECX, DWORD PTR SS:[ESP+34]
00410840	MOV BYTE PTR SS:[ESP+48], 7
00410848	MOV DWORD PTR SS:[ESP+48], 1
00410853	CALL target.004ABF00
00410858	CMP EAX, 1
0041085B	JNZ target.004108FC
00410861	MOV EAX, DWORD PTR DS:[EDI]
00410863	PUSH EAX
00410864	PUSH target.004D6604
00410869	PUSH target.004D65F4
0041086E	MOV ECX, ESI
00410870	MOV DWORD PTR DS:[ESI+194], EBX
00410876	CALL target.004B0826
0041087B	PUSH EBX
0041087C	PUSH EBX
0041087D	PUSH target.004D6408
00410882	MOV DWORD PTR DS:[ESI+1A4], 3
0041088C	CALL target.004B05D8
00410891	XOR EAX, EAX
00410893	MOV DWORD PTR SS:[ESP+10], EAX
00410897	MOV DWORD PTR SS:[ESP+14], EAX
0041089B	LEA ECX, DWORD PTR SS:[ESP+34]
0041089F	MOV BYTE PTR SS:[ESP+18], AL
004108A3	MOV BYTE PTR SS:[ESP+48], 4
004108A6	CALL target.00405F00
004108B0	JMP SHORT target.004108D7
004108B2	MOV EDI, EAX
004108B4	MOV ESI, EDX
004108B6	MOV ECX, 9
004108BB	REP MOVSB BYTE PTR ES:[EDI], BYTE PTR DS:[ESI]
004108BD	POPAD
004108BE	PUSH EDX
004108BF	PUSH EAX
004108C0	CALL target.00499486
004108C5	JMP target.00410826
004108CA	NOP
004108CB	PUSH target.004D471C
004108D0	MOV ECX, EDI

Register	Value
EAX	00E767D8 ASCII "12345678"
ECX	FA6B10B0
EDX	0012F068 ASCII "a9d5bd86"
EBX	00000000
ESP	0012F050
EBP	0051062C target.0051062C
ESI	005104A0 target.005104A0
EDI	00510630 target.00510630
EIP	00410820 target.00410820
CS	00000000 32bit 0(FFFFFFFF)
DS	00000000 32bit 0(FFFFFFFF)
SS	00000000 32bit 0(FFFFFFFF)
ES	00000000 32bit 0(FFFFFFFF)
FS	00000000 32bit 7FFDF000(FFF)
GS	00000000 32bit 0(FFFFFFFF)
IOPL	0
LastError	ERROR_SUCCESS (00000000)
EFL	00000202 (NO, NB, NE, A, NS, PO, ST0, ST1, ST2, ST3, ST4, ST5, ST6, ST7)
FST	0000 Cond 0 0 0 0 Err 0 0
FCW	027F Prec NEAR, 53 Mask

At this point, for our loader generation you can right click the screen and do a copy to executable, all modifications and save under a new name.



3 TEST THE INJECTED CODE

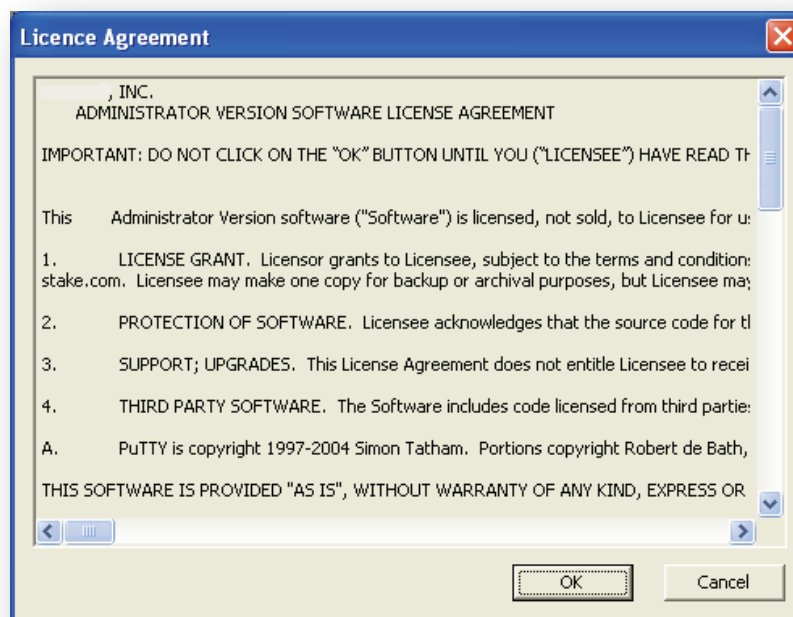
The time has come to test out injected code.

Start stepping through with **F7** watching what is happening with your code and the registers.

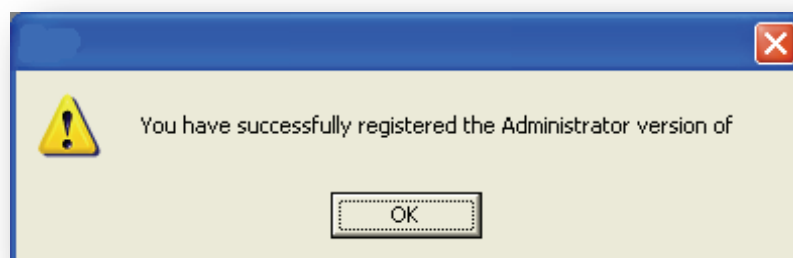
After you have seen your code change let the program run with **F9**

Success!!!,

You should see the following agreement



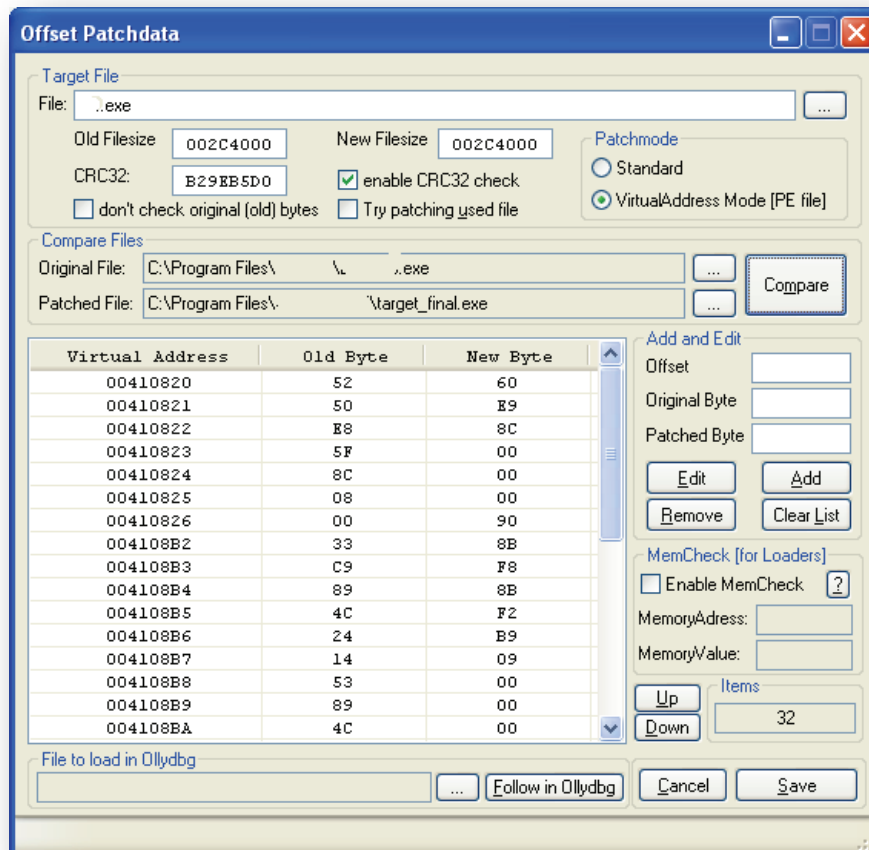
Followed by:



And if you check the registry you will see "Unlock Code"="a9d5bd86" has successfully been entered

4 CREATING THE LOADER FOR CODE INJECTION AND SELF REGISTRATION

Here we are going to generate a loader to inject our code. The easiest way is to startup diablo2oo2's Universal Patcher and create an offset Patch using Virtual Address mode



Then click create simple loader.

Run the loader and success.

5 CONCLUSION

The program used in this tutorial was for demonstration purposes only to show another little discussed method of code injection to self register a program. I hope you have enjoyed my tutorial.

GREETINGS

This tutorial is dedicated to all the components of ARTeam and Shub for editing it. Thanks to all those who reviewed and Nieylana, Nacho_dj and Gunther for the feedback. Big thanks to Shub.

And of course, to you, who decided to read this document, because without your support and contribution this task wouldn't be worth to be done.

Best Wishes

R@dier

HISTORY

Version 1.0

- Initial release