

R4ndom's Tutorial #12: A Tougher NOOBy Example

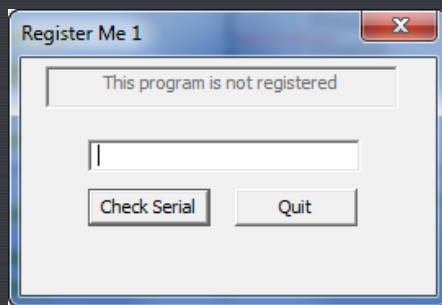
by R4ndom on Jul.09, 2012, under Beginner, Reverse Engineering, Tutorials

Introduction

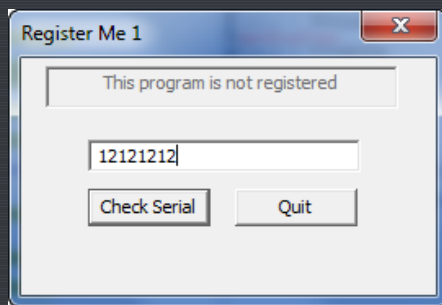
In this tutorial we will be going over a program that's a little more challenging. It is called ReverseM1, written by R4ndom. I will also be discussing the plugin "Ascii Table" for Olly. It is downloadable on the [tools](#) page. This ReverseMe is a perfect example of why the LAME way of patching is often just that- lame.

Getting Started

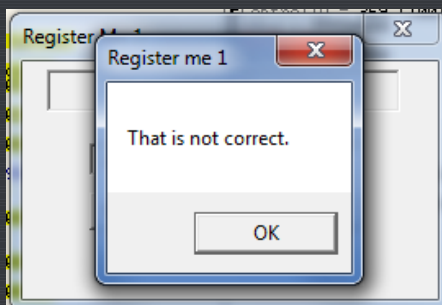
Go ahead and run the program:



We can see that it says it is not registered and is asking for a serial number. Let's give it one:



and click "Check Serial":



and we can see we are not correct (again!). Let's open the app in Olly and do our trusty "search for

strings":

Found strings are		
Address	Disassembly	Text string
00401000	PUSH 0	(Initial CPU selection)
0040100C	PUSH Register.00403000	ASCII "Register me 1"
004010B1	PUSH Register.00403023	ASCII "That is correct!"
004010B0	PUSH Register.00403034	ASCII "This program is registered"
004010D5	PUSH Register.00403000	ASCII "Register me 1"
004010DA	PUSH Register.0040300E	ASCII "That is not correct."
004010E6	PUSH Register.00403050	ASCII "This program is not registered!"
00401101	PUSH Register.00403098	ASCII "12121212"
00401116	MOV EAX,Register.00403098	ASCII "12121212"

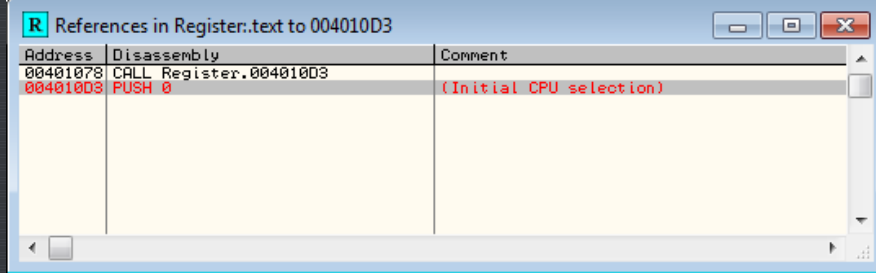
Well, that looks promising. Let's check out the "That is not correct" string:

004010A4	7 30C0	MOV EAX,EAX	004010A4	7 30C0	MOV EAX,EAX
004010A6	C9	LEAVE	004010A6	C9	LEAVE
004010A7	C2 1000	RETN 10	004010A7	C2 1000	RETN 10
004010AA	6A 00	PUSH 0	004010AA	6A 00	PUSH 0
004010AC	68 00304000	PUSH Register.00403000	004010AC	68 00304000	PUSH Register.00403000
004010B1	68 23304000	PUSH Register.00403023	004010B1	68 23304000	PUSH Register.00403023
004010B6	6A 00	PUSH 0	004010B6	6A 00	PUSH 0
004010B8	E8 C3000000	CALL <JMP.&user32.MessageBoxA>	004010B8	E8 C3000000	CALL <JMP.&user32.MessageBoxA>
004010BD	68 34304000	PUSH Register.00403034	004010BD	68 34304000	PUSH Register.00403034
004010C2	68 ED030000	PUSH 3ED	004010C2	68 ED030000	PUSH 3ED
004010C7	FF35 94304000	PUSH DWORD PTR DS:[403094]	004010C7	FF35 94304000	PUSH DWORD PTR DS:[403094]
004010CD	E8 BA000000	CALL <JMP.&user32.SetDlgItemTextA>	004010CD	E8 BA000000	CALL <JMP.&user32.SetDlgItemTextA>
004010D2	C3	RETN	004010D2	C3	RETN
004010D3	6A 00	PUSH 0	004010D3	6A 00	PUSH 0
004010D5	68 00304000	PUSH Register.00403000	004010D5	68 00304000	PUSH Register.00403000
004010DA	68 0E304000	PUSH Register.0040300E	004010DA	68 0E304000	PUSH Register.0040300E
004010DF	6A 00	PUSH 0	004010DF	6A 00	PUSH 0
004010E1	E8 A0000000	CALL <JMP.&user32.MessageBoxA>	004010E1	E8 A0000000	CALL <JMP.&user32.MessageBoxA>
004010E6	68 5B304000	PUSH Register.0040305B	004010E6	68 5B304000	PUSH Register.0040305B
004010EB	68 ED030000	PUSH 3ED	004010EB	68 ED030000	PUSH 3ED
004010F0	FF35 94304000	PUSH DWORD PTR DS:[403094]	004010F0	FF35 94304000	PUSH DWORD PTR DS:[403094]
004010F6	E8 91000000	CALL <JMP.&user32.SetDlgItemTextA>	004010F6	E8 91000000	CALL <JMP.&user32.SetDlgItemTextA>
004010FB	C3	RETN	004010FB	C3	RETN
004010FC	68 00020000	PUSH 200	004010FC	68 00020000	PUSH 200
00401101	68 98304000	PUSH Register.00403098	00401101	68 98304000	PUSH Register.00403098
00401106	68 E9030000	PUSH 3E9	00401106	68 E9030000	PUSH 3E9
0040110B	FF35 94304000	PUSH DWORD PTR DS:[403094]	0040110B	FF35 94304000	PUSH DWORD PTR DS:[403094]
00401111	E8 6A000000	CALL <JMP.&user32.SetDlgItemTextA>	00401111	E8 6A000000	CALL <JMP.&user32.SetDlgItemTextA>
00401116	B8 98304000	MOV EAX,Register.00403098	00401116	B8 98304000	MOV EAX,Register.00403098
0040111B	8B00	MOV EAX,DWORD PTR DS:[EAX]	0040111B	8B00	MOV EAX,DWORD PTR DS:[EAX]
0040111D	66:3D 3433	CMP AX,3334	0040111D	66:3D 3433	CMP AX,3334
00401121	75 07	JNZ SHORT Register.0040112A	00401121	75 07	JNZ SHORT Register.0040112A

and we come to the heart of the matter. Because each of these are separate methods, we will need to see where they are called from, so let's do that:

004010C7	FF35 94304000	PUSH DWORD PTR DS:[403094]	004010C7	FF35 94304000	PUSH DWORD PTR DS:[403094]
004010CD	E8 BA000000	CALL <JMP.&user32.SetDlgItemTextA>	004010CD	E8 BA000000	CALL <JMP.&user32.SetDlgItemTextA>
004010D2	C3	RETN	004010D2	C3	RETN
004010D3	6A 00	PUSH 0	004010D3	6A 00	PUSH 0
004010D5	68 00304000	PUSH Register.00403000	004010D5	68 00304000	PUSH Register.00403000
004010DA	68 0E304000	PUSH Register.0040300E	004010DA	68 0E304000	PUSH Register.0040300E
004010DF	6A 00	PUSH 0	004010DF	6A 00	PUSH 0
004010E1	E8 A0000000	CALL <JMP.&user32.MessageBoxA>	004010E1	E8 A0000000	CALL <JMP.&user32.MessageBoxA>
004010E6	68 5B304000	PUSH Register.0040305B	004010E6	68 5B304000	PUSH Register.0040305B
004010EB	68 ED030000	PUSH 3ED	004010EB	68 ED030000	PUSH 3ED
004010F0	FF35 94304000	PUSH DWORD PTR DS:[403094]	004010F0	FF35 94304000	PUSH DWORD PTR DS:[403094]
004010F6	E8 91000000	CALL <JMP.&user32.SetDlgItemTextA>	004010F6	E8 91000000	CALL <JMP.&user32.SetDlgItemTextA>
004010FB	C3	RETN	004010FB	C3	RETN
004010FC	68 00020000	PUSH 200	004010FC	68 00020000	PUSH 200
00401101	68 98304000	PUSH Register.00403098	00401101	68 98304000	PUSH Register.00403098
00401106	68 E9030000	PUSH 3E9	00401106	68 E9030000	PUSH 3E9
0040110B	FF35 94304000	PUSH DWORD PTR DS:[403094]	0040110B	FF35 94304000	PUSH DWORD PTR DS:[403094]
00401111	E8 6A000000	CALL <JMP.&user32.SetDlgItemTextA>	00401111	E8 6A000000	CALL <JMP.&user32.SetDlgItemTextA>
00401116	B8 98304000	MOV EAX,Register.00403098	00401116	B8 98304000	MOV EAX,Register.00403098
0040111B	8B00	MOV EAX,DWORD PTR DS:[EAX]	0040111B	8B00	MOV EAX,DWORD PTR DS:[EAX]
0040111D	66:3D 3433	CMP AX,3334	0040111D	66:3D 3433	CMP AX,3334
00401121	75 07	JNZ SHORT Register.0040112A	00401121	75 07	JNZ SHORT Register.0040112A
00401123	B8 00000000	MOV EAX,0	00401123	B8 00000000	MOV EAX,0
00401128	EB 05	JMP SHORT Register.0040112F	00401128	EB 05	JMP SHORT Register.0040112F
0040112A	B8 01000000	MOV EAX,1	0040112A	B8 01000000	MOV EAX,1
0040112F	C3	RETN	0040112F	C3	RETN
00401130	E8 C7FFFFF	CALL Register.004010FC	00401130	E8 C7FFFFF	CALL Register.004010FC
00401135	8B00	OR EAX,EAX	00401135	8B00	OR EAX,EAX
00401137	74 33	JS SHORT Register.0040116C	00401137	74 33	JS SHORT Register.0040116C
00401139	B9 1F000000	MOV ECX,1F	00401139	B9 1F000000	MOV ECX,1F
0040113E	BE 00000000	MOV ESI,0	0040113E	BE 00000000	MOV ESI,0
00401143	33C0	XOR EAX,EAX	00401143	33C0	XOR EAX,EAX
00401145	8086 70304000	MOV AL,BYTE PTR DS:[ESI+403070]	00401145	8086 70304000	MOV AL,BYTE PTR DS:[ESI+403070]
0040114B	83F0 2C	XOR EAX,2C	0040114B	83F0 2C	XOR EAX,2C
0040114E	8086 70304000	MOV BYTE PTR DS:[ESI+403070],AL	0040114E	8086 70304000	MOV BYTE PTR DS:[ESI+403070],AL
00401154	INC ESI	INC ESI	00401154	INC ESI	INC ESI
00401155	4E	LOOP SHORT Register.00401145	00401155	4E	LOOP SHORT Register.00401145
00401157	68 70304000	PUSH Register.00403070	00401157	68 70304000	PUSH Register.00403070
0040115C	68 ED030000	PUSH 3ED	0040115C	68 ED030000	PUSH 3ED
00401161	FF35 94304000	PUSH DWORD PTR DS:[403094]	00401161	FF35 94304000	PUSH DWORD PTR DS:[403094]
00401167	E8 20000000	CALL <JMP.&user32.SetDlgItemTextA>	00401167	E8 20000000	CALL <JMP.&user32.SetDlgItemTextA>
0040116C	C3	RETN	0040116C	C3	RETN
0040116D	CC	INT3	0040116D	CC	INT3
0040116E	FF25 24204000	JMP DWORD PTR DS:[&user32.DialogBoxParamA]	0040116E	FF25 24204000	JMP DWORD PTR DS:[&user32.DialogBoxParamA]
0040117A	FF25 18204000	JMP DWORD PTR DS:[&user32.EndDialog]	0040117A	FF25 18204000	JMP DWORD PTR DS:[&user32.EndDialog]

and Olly opens the References window:



and we see that there is one call to this function. Let's double-click that and see what it looks like:

00401063	. E8 94000000	CALL Register.004010FC	
00401068	. 0BC0	OR EAX,EAX	rport4.75CB1B9C
0040106A	. 75 0C	JNZ SHORT Register.00401078	
0040106C	. E8 39000000	CALL Register.004010AA	
00401071	. E8 BA000000	CALL Register.00401130	
00401076	. EB 2C	JMP SHORT Register.004010A4	
00401078	> . E8 56000000	CALL Register.004010D3	
0040107D	. EB 25	JMP SHORT Register.004010A4	
0040107F	. 817D 10 EC030000	CMP [ARG.3],SEC	
00401086	. 75 1C	JNZ SHORT Register.004010A4	
00401088	. 6A 00	PUSH 0	[Result = 0 hwnd = 0252006C EndDialog
0040108A	. FF75 08	PUSH [ARG.1]	
0040108D	. E8 E2000000	CALL <JMP.&user32.EndDialog>	
00401092	. EB 10	JMP SHORT Register.004010A4	
00401094	. 837D 0C 10	CMP [ARG.2],10	
00401098	. 75 0A	JNZ SHORT Register.004010A4	
0040109A	. 6A 00	PUSH 0	[Result = 0 hwnd = 0252006C EndDialog rport4.75CB1B9C
0040109C	. FF75 08	PUSH [ARG.1]	
0040109F	. E8 D0000000	CALL <JMP.&user32.EndDialog>	
004010A4	. 33C0	XOR EAX,EAX	
004010A6	. C9	LEAVE	
004010A7	. C2 1000	RETN 10	
004010AA	. 6A 00	PUSH 0	[Style = MB_OK!MB_APPLMODAL Title = "Register me 1" Text = "That is correct!" hwndOwner = NULL MessageBoxA
004010AC	. 68 00304000	PUSH Register.00403000	
004010B1	. 68 23304000	PUSH Register.00403023	
004010B6	. 6A 00	PUSH 0	
004010B8	. E8 C9000000	CALL <JMP.&user32.MessageBoxA>	
004010BD	. 68 34304000	PUSH Register.00403034	
004010C2	. 68 ED030000	PUSH 3ED	[Text = "This program is registered!" ControlID = 3ED (1005.) hwnd = 00030368 ('Register Me 1',class='#32770') SetDlgItemTextA
004010C7	. FF35 94304000	PUSH DWORD PTR DS:[403094]	
004010CD	. E8 BA000000	CALL <JMP.&user32.SetDlgItemTextA>	
004010D2	. C3	RETN	
004010D3	. 6A 00	PUSH 0	[Style = MB_OK!MB_APPLMODAL Title = "Register me 1" Text = "That is not correct." hwndOwner = NULL MessageBoxA
004010D5	. 68 00304000	PUSH Register.00403000	
004010DA	. 68 0E304000	PUSH Register.0040300E	
004010DF	. 6A 00	PUSH 0	
004010E1	. E8 A0000000	CALL <JMP.&user32.MessageBoxA>	
004010E6	. 68 50304000	PUSH Register.00403050	
004010EB	. 68 ED030000	PUSH 3ED	[Text = "This program is not registered!" ControlID = 3ED (1005.) hwnd = 00030368 ('Register Me 1',class='#32770') SetDlgItemTextA
004010F0	. FF35 94304000	PUSH DWORD PTR DS:[403094]	
004010F6	. E8 91000000	CALL <JMP.&user32.SetDlgItemTextA>	
004010FB	. C3	RETN	
004010FC	. 68 00020000	PUSH 200	[Count = 200 (512.) Buffer = Register.00403098 ControlID = 3E9 (1001.) hwnd = 00030368 ('Register Me 1',class='#32770') GetDlgItemTextA ASCII "12121212"
00401101	. 68 98304000	PUSH Register.00403098	
00401106	. 68 E9030000	PUSH 3E9	
00401108	. FF35 94304000	PUSH DWORD PTR DS:[403094]	
00401111	. E8 6A000000	CALL <JMP.&user32.GetDlgItemTextA>	
00401116	. B8 98304000	MOV EAX,Register.00403098	
0040111B	. 9B00	MOV AX,DWORD PTR DS:[EAX]	
0040111D	. 66 3D 3433	CMPSX 3433	
00401121	. 75 07	JNZ SHORT Register.0040112A	
00401123	. B8 00000000	MOV EAX,0	

Here, we can see that the bad boy is called from address 401078, and we can also immediately see that there is a jump instruction that jumps to this call at address 40106A:

00401063	EB 94000000	CALL Register.004010FC	OR EAX,EAX	4.75C81B9C
00401068	0BC0	JNZ SHORT Register.00401078		
0040106A	75 0C	CALL Register.004010AA		
0040106C	E8 39000000	CALL Register.00401130		
00401071	E8 BA000000	JMP SHORT Register.004010A4		
00401076	EB 2C	CALL Register.004010D3		
00401078	E8 56000000	JMP SHORT Register.004010A4		
0040107D	EB 25	CMP [ARG_3],3EC		
0040107F	817D 10 EC030000	JNZ SHORT Register.004010A4		
00401086	75 1C	PUSH 0		
00401088	6A 00	PUSH [ARG_1]		
0040108A	FF75 08	CALL <JMP.&user32.EndDialog>		
0040108D	E8 E2000000	JMP SHORT Register.004010A4		
00401092	EB 10	CMP [ARG_2],10		
00401094	837D 0C 10	JNZ SHORT Register.004010A4		
00401098	75 0A	PUSH 0		
0040109A	6A 00	PUSH [ARG_1]		
0040109C	FF75 08	CALL <JMP.&user32.EndDialog>		
0040109F	E8 D0000000	XOR EAX,EAX		
004010A4	33C0	LEAVE		
004010A6	C9	RETN 10		
004010A7	C2 1000	PUSH 0		
004010AA	6A 00	PUSH Register.00403000		
004010AC	68 00304000	PUSH Register.00403023		
004010B1	68 23304000	PUSH 0		
004010B6	6A 00	CALL <JMP.&user32.MessageBoxA>		
004010B8	E8 C9000000	PUSH Register.00403034		
004010BD	68 34304000	PUSH 3ED		
004010C2	68 ED030000	PUSH DWORD PTR DS:[403094]		
004010C7	FF35 94304000	CALL <JMP.&user32.SetDlgItemTextA>		
004010CD	E8 BA000000	RETN		
004010D2	C3	PUSH 0		
004010D3	C2 00	PUSH Register.00403000		
004010D5	68 00304000	PUSH Register.0040300E		
004010DA	68 0E304000	PUSH 0		
004010DF	6A 00	CALL <JMP.&user32.MessageBoxA>		
004010E1	E8 A0000000	PUSH Register.00403050		
004010E6	68 50304000	PUSH 3ED		
004010EB	68 ED030000	PUSH DWORD PTR DS:[403094]		
004010F6	FF35 94304000	CALL <JMP.&user32.SetDlgItemTextA>		
004010FB	E8 91000000	RETN		
004010FC	C3	PUSH 200		
004010FC	68 00020000	PUSH Register.00403098		
00401101	68 98304000	PUSH 3E9		
00401106	68 E9030000	PUSH DWORD PTR DS:[403094]		
0040110B	FF35 94304000	CALL <JMP.&user32.GetDlgItemTextA>		
00401111	E8 6A000000	MOV EAX,Register.00403098		
00401116	B8 98304000	MOV EAX,DWORD PTR DS:[EAX]		
00401118	8B00	CMP AX,3334		
0040111D	66:3D 3433	JNZ SHORT Register.0040112A		
00401121	75 07	MOV EAX,0		
00401123	B8 00000000	JMP SHORT Register.0040112F		
00401128	B8 05	MOV EAX,1		
0040112A	B8 01000000	RETN		
0040112F	C3	CALL Register.004010FC		
00401130	E8 C7FFFFFF	OR EAX,EAX		
00401135	0BC0	JE SHORT Register.0040116C		
00401137	74 33			

This JNZ...

Which calls the bad boy

Result = 0
hWnd = 0252006C
EndDialog

Style = MB_OK|MB_APPLMODAL
Title = "Register me 1"
Text = "That is correct!"
hOwner = NULL
MessageBoxA
Text = "This program is registered!"
ControlID = 3ED (1005.)
hWnd = 00030368 ('Register Me 1',class='#32770')
SetDlgItemTextA

Style = MB_OK|MB_APPLMODAL
Title = "Register me 1"
Text = "That is not correct."
hOwner = NULL
MessageBoxA
Text = "This program is not registered!"
ControlID = 3ED (1005.)
hWnd = 00030368 ('Register Me 1',class='#32770')
SetDlgItemTextA

Count = 200 (512.)
Buffer = Register.00403098
ControlID = 3E9 (1001.)
hWnd = 00030368 ('Register Me 1',class='#32770')
GetDlgItemTextA
ASCII "12121212"

Scrolling up a couple lines we can see the proverbial call to check routine/compare/jump that we've seen before. From this we can guess that the main checking routine is at 4010FC, called from address 401063. After returning, the EAX register is checked if it contains zero or not, and if it doesn't, we jump to the bad boy.

00401059	75 24	JNZ SHORT Register.0040107F		
0040105B	8B45 08	MOV EAX,[ARG_1]		
0040105E	A3 94304000	MOV DWORD PTR DS:[403094],EAX		
00401063	E8 94000000	CALL Register.004010FC		
00401068	0BC0	OR EAX,EAX		
0040106A	75 0C	JNZ SHORT Register.00401078		
0040106C	E8 39000000	CALL Register.004010AA		
00401071	E8 BA000000	CALL Register.00401130		
00401076	EB 2C	JMP SHORT Register.004010A4		
00401078	E8 56000000	CALL Register.004010D3		
0040107D	EB 25	JMP SHORT Register.004010A4		
0040107F	817D 10 EC030000	CMP [ARG_3],3EC		
00401086	75 1C	JNZ SHORT Register.004010A4		
00401088	6A 00	PUSH 0		
0040108A	FF75 08	PUSH [ARG_1]		
0040108D	E8 E2000000	CALL <JMP.&user32.EndDialog>		
00401092	EB 10	JMP SHORT Register.004010A4		
00401094	837D 0C 10	CMP [ARG_2],10		
00401098	75 0A	JNZ SHORT Register.004010A4		
0040109A	6A 00	PUSH 0		
0040109C	FF75 08	PUSH [ARG_1]		
0040109F	E8 D0000000	CALL <JMP.&user32.EndDialog>		
004010A4	33C0	XOR EAX,EAX		
004010A6	C9	LEAVE		
004010A7	C2 1000	RETN 10		
004010AA	6A 00	PUSH 0		
004010AC	68 00304000	PUSH Register.00403000		
004010B1	68 23304000	PUSH Register.00403023		
004010B6	6A 00	PUSH 0		
004010B8	E8 C9000000	CALL <JMP.&user32.MessageBoxA>		
004010BD	68 34304000	PUSH Register.00403034		
004010C2	68 ED030000	PUSH 3ED		
004010C7	FF35 94304000	PUSH DWORD PTR DS:[403094]		
004010CD	E8 BA000000	CALL <JMP.&user32.SetDlgItemTextA>		
004010D2	C3	RETN		
004010D3	C2 00	PUSH 0		
004010D5	68 00304000	PUSH Register.00403000		
004010DA	68 0E304000	PUSH Register.0040300E		
004010DF	6A 00	PUSH 0		
004010E1	E8 A0000000	CALL <JMP.&user32.MessageBoxA>		
004010E6	68 50304000	PUSH Register.00403050		
004010EB	68 ED030000	PUSH 3ED		
004010F6	FF35 94304000	PUSH DWORD PTR DS:[403094]		
004010FB	E8 91000000	CALL <JMP.&user32.SetDlgItemTextA>		
004010FC	C3	PUSH 200		
004010FC	68 00020000	PUSH Register.00403098		
00401101	68 98304000	PUSH 3E9		
00401106	68 E9030000	PUSH DWORD PTR DS:[403094]		
0040110B	FF35 94304000	CALL <JMP.&user32.GetDlgItemTextA>		
00401111	E8 6A000000	MOV EAX,Register.00403098		
00401116	B8 98304000	MOV EAX,DWORD PTR DS:[EAX]		
00401118	8B00	CMP AX,3334		
0040111D	66:3D 3433	JNZ SHORT Register.0040112A		
00401121	75 07	MOV EAX,0		
00401123	B8 00000000	JMP SHORT Register.0040112F		
00401128	B8 05	MOV EAX,1		
0040112A	B8 01000000	RETN		
0040112F	C3	CALL Register.004010FC		
00401130	E8 C7FFFFFF	OR EAX,EAX		
00401135	0BC0	JE SHORT Register.0040116C		
00401137	74 33			

This is the main call..

Result = 0
hWnd = 0252006C
EndDialog

Result = 0
hWnd = 0252006C
EndDialog
rport4.75C81B9C

Style = MB_OK|MB_APPLMODAL
Title = "Register me 1"
Text = "That is correct!"
hOwner = NULL
MessageBoxA
Text = "This program is registered!"
ControlID = 3ED (1005.)
hWnd = 00030368 ('Register Me 1',class='...that calls here..')
SetDlgItemTextA

Style = MB_OK|MB_APPLMODAL
Title = "Register me 1"
Text = "That is not correct."
hOwner = NULL
MessageBoxA
Text = "This program is not registered!"
ControlID = 3ED (1005.)
hWnd = 00030368 ('Register Me 1',class='...which is the main reg check')
SetDlgItemTextA

Count = 200 (512.)
Buffer = Register.00403098
ControlID = 3E9 (1001.)
hWnd = 00030368 ('Register Me 1',class='...')
GetDlgItemTextA
ASCII "12121212"

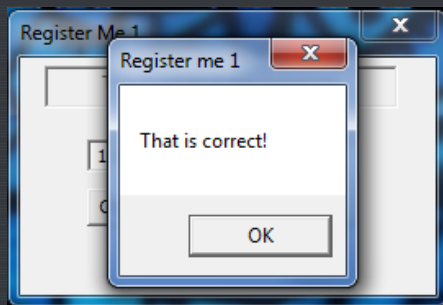
Let's test out our hypothesis and set a breakpoint at address 40106A and re-start the app. After entering a serial number (I entered the same '12121212') we break at the jump after the call to the serial check:

00401063	•	E8 94304000	MOV DWORD PTR DS:[403094],EAX
00401063	•	E8 94000000	CALL Register.004010FC
00401068	•	0BC0	OR EAX,EAX
0040106A	•	75 0C	JNZ SHORT Register.00401078
0040106C	•	E8 39000000	CALL Register.004010AA
00401071	•	E8 BA000000	CALL Register.00401130
00401076	•	EB 2C	JMP SHORT Register.004010A4
00401078	•	E8 56000000	CALL Register.004010D3
0040107D	•	EB 25	JMP SHORT Register.004010A4
0040107F	•	817D 10 EC030000	CMP [ARG.3],3EC
00401086	•	75 1C	JNZ SHORT Register.004010A4
00401088	•	6A 00	PUSH 0
0040108A	•	FF75 08	PUSH [ARG.1]
0040108D	•	E8 E2000000	CALL <JMP.&user32.EndDialog>
00401092	•	EB 10	JMP SHORT Register.004010A4
00401094	•	837D 0C 10	CMP [ARG.2],10
00401098	•	75 0A	JNZ SHORT Register.004010A4
0040109A	•	6A 00	PUSH 0

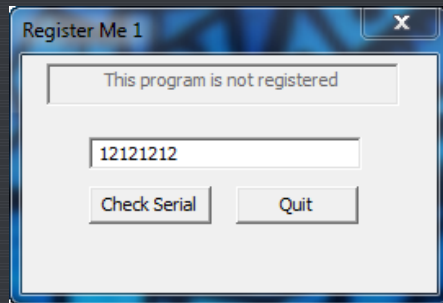
Now let's help Olly in the right direction so he won't take the jump (and fall through to the call to the good boy):

```
C 0 ES 002
P 0 CS 001
A 0 SS 002
Z 1 DS 002
S 0 FS 003
T 0 GS 000
O 0 LastEx
```

and hit run:



Yeah, that was easy!! Click OK and:



Oh F%\$@ that S&^@, what the F\$^& is going on here, you A\$\$\$%^#!!!! Obviously it didn't register our program. This means that there must be something we missed.

Looking a Little Closer

Let's re-start the app, enter a serial and let Olly break again at 40106A:

00401063	•	E8 94304000	MOV DWORD PTR DS:[403094],EAX
00401063	•	E8 94000000	CALL Register.004010FC
00401068	•	0BC0	OR EAX,EAX
0040106A	•	75 0C	JNZ SHORT Register.00401078
0040106C	•	E8 39000000	CALL Register.004010AA
00401071	•	E8 BA000000	CALL Register.00401130
00401076	•	EB 2C	JMP SHORT Register.004010A4
00401078	•	E8 56000000	CALL Register.004010D3
0040107D	•	EB 25	JMP SHORT Register.004010A4
0040107F	•	817D 10 EC030000	CMP [ARG.3],3EC
00401086	•	75 1C	JNZ SHORT Register.004010A4
00401088	•	6A 00	PUSH 0
0040108A	•	FF75 08	PUSH [ARG.1]

We see that if we keep Olly from making the jump to the bad boy, execution falls through to the call at line 40106C, which will call address 4010AA. Looking down at that routine, we can see that it is pretty standard; it opens a message box with "That is correct" and then changes the label on the main screen to

"This program is registered!"

00401065	EB 30000000	CALL Register.00401065	
00401066	0BC0	OR EAX,EAX	
0040106A	75 0C	JNZ SHORT Register.00401078	
0040106C	E8 39000000	CALL Register.004010AA	
00401071	E8 BA000000	CALL Register.00401130	
00401076	EB 2C	JMP SHORT Register.004010A4	
00401078	E8 56000000	CALL Register.004010D3	
0040107D	EB 25	JMP SHORT Register.004010A4	
0040107F	817D 10 EC030000	CMP [ARG.3],3EC	
00401086	75 1C	JNZ SHORT Register.004010A4	
00401088	6A 00	PUSH 0	
0040108A	FF75 08	PUSH [ARG.1]	
0040108D	E8 E2000000	CALL <JMP.&user32.EndDialog>	
00401092	EB 10	JMP SHORT Register.004010A4	
00401094	837D 0C 10	CMP [ARG.2],10	
00401098	75 0A	JNZ SHORT Register.004010A4	
0040109A	6A 00	PUSH 0	
0040109C	FF75 08	PUSH [ARG.1]	
0040109F	E8 D0000000	CALL <JMP.&user32.EndDialog>	
004010A4	33C0	XOR EAX,EAX	
004010A6	C9	LEAVE	
004010A7	C2 1000	RETN 10	
004010A8	6A 00	PUSH 0	
004010AC	68 00304000	PUSH Register.00403000	
004010B1	68 23304000	PUSH Register.00403023	
004010B6	6A 00	PUSH 0	
004010B8	E8 C9000000	CALL <JMP.&user32.MessageBoxA>	
004010BD	68 34304000	PUSH Register.00403034	
004010C2	68 ED030000	PUSH 3ED	
004010C7	FF35 94304000	PUSH DWORD PTR DS:[403094]	
004010CD	E8 BA000000	CALL <JMP.&user32.SetDlgItemTextA>	
004010D2	C3	RETN	

Result = 0
hwnd = 000503B2 ('Register Me 1',class='#32770')
EndDialog

Result = 0
hwnd = 000503B2 ('Register Me 1',class='#32770')
EndDialog

Style = MB_OK|MB_APPLMODAL
Title = "Register me 1"
Text = "That is correct!"
hOwner = NULL
MessageBoxA
Text = "This program is registered!"
ControlID = 3ED (1005.)
hwnd = 000503B2 ('Register Me 1',class='#32770')
SetDlgItemTextA

Style = MB_OK|MB_APPLMODAL
Title = "Register me 1"

But wait! Once we return from that call, there is another call at 401071:

00401065	EB 30000000	CALL Register.00401065	
00401066	0BC0	OR EAX,EAX	
0040106A	75 0C	JNZ SHORT Register.00401078	
0040106C	E8 39000000	CALL Register.004010AA	
00401071	E8 BA000000	CALL Register.00401130	
00401076	EB 2C	JMP SHORT Register.004010A4	
00401078	E8 56000000	CALL Register.004010D3	
0040107D	EB 25	JMP SHORT Register.004010A4	
0040107F	817D 10 EC030000	CMP [ARG.3],3EC	
00401086	75 1C	JNZ SHORT Register.004010A4	
00401088	6A 00	PUSH 0	
0040108A	FF75 08	PUSH [ARG.1]	
0040108D	E8 E2000000	CALL <JMP.&user32.EndDialog>	
00401092	EB 10	JMP SHORT Register.004010A4	
00401094	837D 0C 10	CMP [ARG.2],10	
00401098	75 0A	JNZ SHORT Register.004010A4	
0040109A	6A 00	PUSH 0	
0040109C	FF75 08	PUSH [ARG.1]	
0040109F	E8 D0000000	CALL <JMP.&user32.EndDialog>	
004010A4	33C0	XOR EAX,EAX	
004010A6	C9	LEAVE	
004010A7	C2 1000	RETN 10	
004010AA	6A 00	PUSH 0	
004010AC	68 00304000	PUSH Register.00403000	
004010B1	68 23304000	PUSH Register.00403023	
004010B6	6A 00	PUSH 0	
004010B8	E8 C9000000	CALL <JMP.&user32.MessageBoxA>	
004010BD	68 34304000	PUSH Register.00403034	
004010C2	68 ED030000	PUSH 3ED	
004010C7	FF35 94304000	PUSH DWORD PTR DS:[403094]	
004010CD	E8 BA000000	CALL <JMP.&user32.SetDlgItemTextA>	
004010D2	C3	RETN	
004010D3	6A 00	PUSH 0	
004010D5	68 00304000	PUSH Register.00403000	

Result = 0
hwnd = 000503B2 ('Register Me 1',class='#32770')
EndDialog

Result = 0
hwnd = 000503B2 ('Register Me 1',class='#32770')
EndDialog

Style = MB_OK|MB_APPLMODAL
Title = "Register me 1"
Text = "That is correct!"
hOwner = NULL
MessageBoxA
Text = "This program is registered!"
ControlID = 3ED (1005.)
hwnd = 000503B2 ('Register Me 1',class='#32770')
SetDlgItemTextA

Style = MB_OK|MB_APPLMODAL
Title = "Register me 1"

That call calls 401130, so let's have a look-see at that routine. First of all, we notice that it calls SetDlgItemTextA, but with a strange looking string. Let's step through this line by line. At 401130 a call is made to address 4010FC. Looking above at this we can see that this is the serial check routine. It then OR's EAX with itself to see if it's zero, and if it is not, it performs a lot of weird looking stuff:

0040112F	C3	RETN	
00401130	E8 C7FFFFFF	CALL Register.004010FC	
00401135	0BC0	OR EAX,EAX	
00401137	74 33	JE SHORT Register.0040116C	
00401139	B9 1F000000	MOV ECX,1F	
0040113E	BE 00000000	MOV ESI,0	
00401143	33C0	XOR EAX,EAX	
00401145	8A86 70304000	MOV AL,BYTE PTR DS:[ESI+403070]	
00401148	83F0 2C	XOR EAX,2C	
0040114E	8886 70304000	MOV BYTE PTR DS:[ESI+403070],AL	
00401153	4E	INC ESI	
00401154	75 00	JNB SHORT Register.00401145	
00401155	68 00000000	PUSH Register.00403070	
00401156	68 00000000	PUSH Register.00403070	
00401157	E8 C3000000	CALL <JMP.&user32.SetDlgItemTextA>	
0040115D	C3	RETN	

Call to serial check

If EAX != 0...

Do all this

Text = "MODE\\00\\^CK^MA\\00CE\\00CBCK\\00^IKE_XI^IH^r"
ControlID = 3ED (1005.)
hwnd = 000503B2 ('Register Me 1',class='#32770')
SetDlgItemTextA

So what we can gather from this so far is that, after we patch the app to display the good boy message, another call is made, and within this call, a call is made to the check serial routine again, performing the same analysis on the results. This is a backup check! Now let's see what happens if we fail this second backup check (which we will since we only patched the jump):

00401139	B9 1F000000	MOV ECX,1F	
0040113E	BE 00000000	MOV ESI,0	
00401143	33C0	XOR EAX,EAX	
00401145	8A86 70304000	MOV AL,BYTE PTR DS:[ESI+403070]	
00401148	83F0 2C	XOR EAX,2C	
0040114E	8886 70304000	MOV BYTE PTR DS:[ESI+403070],AL	
00401154	4E	INC ESI	
00401155	E2 EE	LOOPD SHORT Register.00401145	
00401157	68 70304000	PUSH Register.00403070	

First, ECX is loaded with 1F (31 decimal) ** Sorry, it's a little cut off **. ESI is then loaded with zero and EAX is zeroed out. We then enter a loop. We'll go through the loop step-by-step. The first line moves a byte from an address, ESI + 403070, and since we know ESI equals zero, the address is actually just

403070, into the AL register. Let's see what's in the dump. Either right click and select Follow in dump -> constant or just click in the dump window and select goto and type in the address 403070:

Address	Hex dump	ASCII
00403070	78 44 45 5F 0C 5C 5E 43 4B 5E 4D 41 0C 45 5F 0C	TDE_\\^CK^MA_E..
00403080	42 43 58 0C 5E 49 4B 45 5F 58 49 5E 49 48 0D 00	BCX.^IKE_XI^IH..
00403090	00 00 40 00 B2 03 05 00 31 32 31 32 31 32 31 32	..0.豳*.I2121212
004030A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004030B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004030C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004030D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004030E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004030F0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403100	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403110	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403120	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403130	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

If we look carefully, we can see that this is the string that appeared above in the argument to SetDlgItemTextA. So it is loading the first character of this weird looking string into AL.

*** One thing you should know is that in a lot of assembly language instructions, certain registers are used in default ways, for example ECX is used as a counter, ESI is used as a source address, and EDI is used as a destination. This is the case in this example.***

Next, we XOR this character with 2C, then save it back into the same memory address:

0040113E

• BE 00000000

MOV ESI,0

00401143

• 33C0

XOR EAX,EAX

00401145

> 8A86 70304000

MOV AL,BYTE PTR DS:[ESI+403070]

00401148

• 83F0 2C

XOR EAX,2C

0040114E

• 8B86 70304000

MOV BYTE PTR DS:[ESI+403070],AL

00401154

4E

INC ESI

00401155

^ E2 EE

LOOPD SHORT Register.00401145

00401157

• 68 70304000

PUSH Register.00403070

0040115C

• 68 ED030000

PUSH 3ED

00401161

• FF35 94304000

PUSH DWORD PTR DS:[403094]

00401167

• E8 20000000

CALL <JMP.&user32.SetDlgItemTextA>

0040116C

> C3

RETN

Save back into memory

Text = "TDE_\\^CK^MA_E..

ControlID = 3ED

hwnd = 000503B2

SetDlgItemTextA

Lastly, we increment ESI (the source register) and do a LOOPD. LOOPD means lower the ECX register by one and loop until ECX equals zero. This tells us that the value that was loaded into ECX originally, 31 decimal, is the length of this loop.

From a big picture, this loop is basically cycling through each character of this weird string, XOR-ing it with 2C, and saving it back into memory. This will go on until ECX equals zero, or 31 times. Single step once past the LOOPD instruction to go back up to the top and look in the dump window:

Address	Hex dump	ASCII
00403070	54 44 45 5F 0C 5C 5E 43 4B 5E 4D 41 0C 45 5F 0C	TDE_\\^CK^MA_E..
00403080	42 43 58 0C 5E 49 4B 45 5F 58 49 5E 49 48 0D 00	BCX.^IKE_XI^IH..
00403090	00 00 40 00 B2 03 05 00 31 32 31 32 31 32 31 32	..0.豳*.I2121212
004030A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004030B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004030C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004030D0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004030E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004030F0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403100	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403110	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

You will notice that the first digit of this string has been replaced. The original character was XOR'ed and now it is a "T". If you step through this loop several times, you will see the dump window's string change. You will also see the argument for the SetDlgItemTextA change as well:

```

00401153 74 33 08C0 OR EAX,EAX
00401157 59 1F000000 MOV ECX,1F
00401159 BE 00000000 MOV ESI,0
00401163 33C0 XOR EAX,EAX
00401165 8A86 70304000 MOV AL,BYTE PTR DS:[ESI+403070]
00401168 83F0 2C XOR EAX,2C
0040116E 8886 70304000 MOV BYTE PTR DS:[ESI+403070],AL
00401170 46 INC ESI
00401175 E2 EE LOOPD SHORT Register.00401145
00401177 68 70304000 PUSH Register.00403070
00401179 68 ED030000 PUSH 3ED
00401181 FF35 94304000 PUSH DWORD PTR DS:[403094]
00401183 E8 20000000 CALL <JMP.&user32.SetDlgItemTextA>
00401185 C3 RETN
00401187 INT3
00401189 JMP DWORD PTR DS:[<&user32.DialogBoxParamA>] user32.DialogBoxParamA
0040118B FF25 18204000 JMP DWORD PTR DS:[<&user32.EndDialog>] user32.EndDialog
0040118D FF25 10204000 JMP DWORD PTR DS:[<&user32.GetDlgItem>] user32.GetDlgItem
0040118F FF25 20204000 JMP DWORD PTR DS:[<&user32.GetDlgItemTextA>] user32.GetDlgItemTextA
00401191 FF25 0C204000 JMP DWORD PTR DS:[<&user32.MessageBoxA>] user32.MessageBoxA
00401193 FF25 1C204000 JMP DWORD PTR DS:[<&user32.SetDlgItemTextA>] user32.SetDlgItemTextA
00401195 FF25 14204000 JMP DWORD PTR DS:[<&user32.SetFocus>] user32.SetFocus
00401197 FF25 00204000 JMP DWORD PTR DS:[<&kernel32.ExitProcess>] kernel32.ExitProcess
00401199 FF25 04204000 JMP DWORD PTR DS:[<&kernel32.GetModuleHandleA>] kernel32.GetModuleHandleA
004011A4 00 DB 00
004011A5 00 DB 00
004011A6 00 DB 00
004011A7 00 DB 00
004011A8 00 DB 00
004011A9 00 DB 00
004011AA 00 DB 00
004011AB 00 DB 00
004011AC 00 DB 00

```

Loop is taken
ECX=00000010 (decimal 16.)
00401145=Register.00401145

Address	Hex dump	ASCII
00403070	54 68 69 73 20 70 72 6F 67 72 61 6D 20 69 73 20	This program is
00403080	42 43 58 0C 5E 49 48 45 5F 58 49 5E 49 48 00 00	BCX\X0C\IKE_XI^IH\r
00403090	00 00 40 00 B2 03 05 00 31 32 31 32 31 32 31 32	ControlID = 3ED (1005.)
004030A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	hWnd = 000503B2 ('Register Me 1',class='#32770')
004030B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	SetDlgItemTextA
004030C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	

Stepping completely through the loop, we can see the final message, which looks surprisingly familiar; "This program is not registered!". This is the same message displayed on the main screen showing that the app is in fact not registered yet:

```

00401143 33C0 OR EAX,EAX
00401145 8A86 70304000 MOV AL,BYTE PTR DS:[ESI+403070]
00401148 83F0 2C XOR EAX,2C
0040114E 8886 70304000 MOV BYTE PTR DS:[ESI+403070],AL
00401150 46 INC ESI
00401155 E2 EE LOOPD SHORT Register.00401145
00401157 68 70304000 PUSH Register.00403070
00401159 68 ED030000 PUSH 3ED
00401161 FF35 94304000 PUSH DWORD PTR DS:[403094]
00401163 E8 20000000 CALL <JMP.&user32.SetDlgItemTextA>
00401165 C3 RETN
00401167 INT3
00401169 JMP DWORD PTR DS:[<&user32.DialogBoxParamA>] user32.DialogBoxParamA

```

Text = "This program is not registered!"
ControlID = 3ED (1005.)
hWnd = 000503B2 ('Register Me 1',class='#32770')
SetDlgItemTextA

You can see that this string then becomes a value passed to the SetDlgItemTextA routine, in effect replacing the registered message that was put up at the end of the good boy with a copy of the unregistered message that was there before:

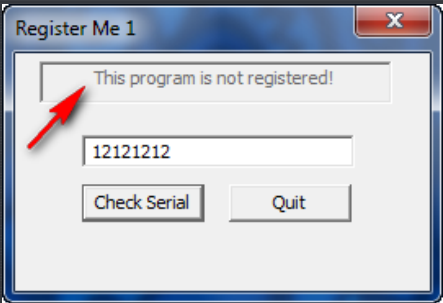
```

00401154 46 INC ESI
00401155 E2 EE LOOPD SHORT Register.00401145
00401157 68 70304000 PUSH Register.00403070
00401159 68 ED030000 PUSH 3ED
00401161 FF35 94304000 PUSH DWORD PTR DS:[403094]
00401163 E8 20000000 CALL <JMP.&user32.SetDlgItemTextA>
00401165 C3 RETN
00401167 INT3
00401169 JMP DWORD PTR DS:[<&user32.DialogBoxParamA>] user32.DialogBoxParamA

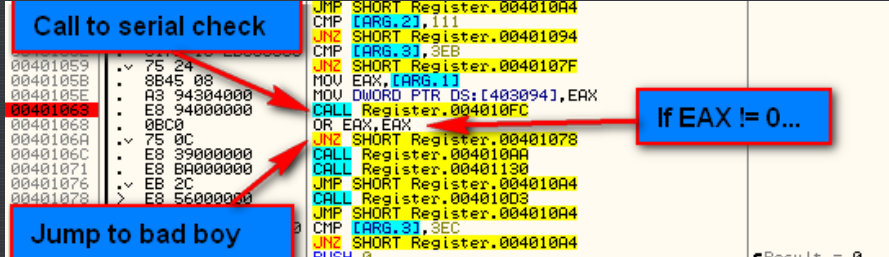
```

Text = "This program is not registered!"
ControlID = 3ED (1005.)
hWnd = 000503B2 ('Register Me 1',class='#32770')
SetDlgItemTextA

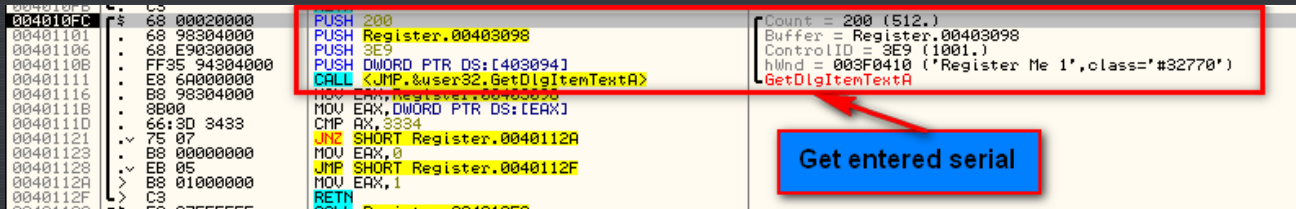
And here we see it in the main app:



So now we know that the smarter way to patch this app is to go into the serial check and make sure it always returns the right value, as it's called not only as the first check, but also after the success screen is displayed. Just to remind you, the call to the serial check is called, then eax is tested for zero. If it's not a zero, we jump to the bad boy- so we want that routine to return a zero! Then, the second time the serial check routine is called, it will return a zero again, and our second check will be passed:



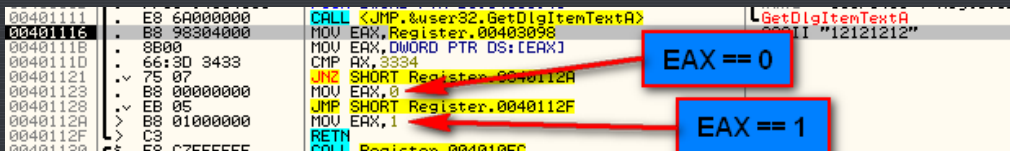
So let's go to the serial checking routine and see what we can do about it. At the beginning of the routine is a call to GetDlgItemTextA, as we could guess just get's our entered serial. You can see this by right clicking on the argument at address 401101 (that points to the buffer that the text will be placed in) and following it in the dump:



After we step over the GetDlgItemTextA instruction, we can see our serial in the buffer:

Address	Hex dump	ASCII
00403098	31 32 31 32 00 00 00 00 00 00 00 00 00 00 00 00	12121212.....
004030A8	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004030B8	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004030C8	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004030D8	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004030E8	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004030F8	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403108	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403118	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403128	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403138	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

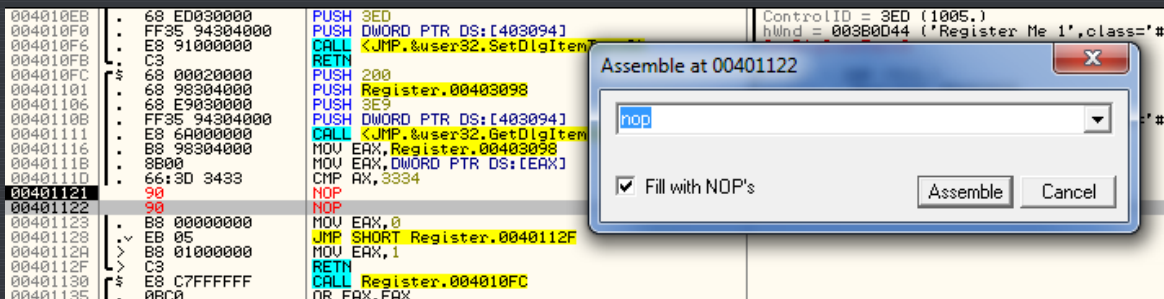
After it is stored in the buffer, the address of the beginning of the buffer is moved into EAX, and then the contents of this address are moved into EAX. This basically moves the first four bytes of our password into EAX. These bytes are then compared with 3334, and if they don't match, EAX is filled with a 1 (bad), otherwise, if they do match, EAX is stored with a zero (good):



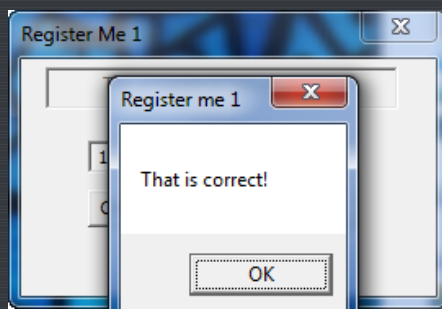
We can see that the main decision maker is the JNZ instruction at address 401121:

00401118	8B00	MOV EAX,Register.00403098
0040111D	66:3D 3433	CMP AX,3334
00401121	75 07	JNZ SHORT Register.0040112A
00401123	B8 00000000	MOV EAX,0
00401128	EB 05	JMP SHORT Register.0040112F
0040112A	B8 01000000	MOV EAX,1
0040112F	C3	RET

This line determines whether EAX will equal zero or 1 right before the return. So what we want to do is guarantee that EAX will always equal zero:



So now, the code will always fall through to moving zero into EAX and then jumping directly to the return. Now running the app:



and noticing that after the call to the serial check, we naturally jump to the goodboy:

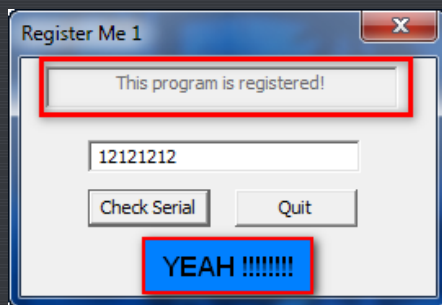
<pre> 0040112F C3 RETN 00401130 E8 C7FFFFFF CALL Register.004010FC 00401135 0BC0 OR EAX,EAX 00401137 74 33 JE SHORT Register.0040116C 00401139 B9 1F000000 MOV ECX,1F 0040113E BE 00000000 MOV ESI,0 00401143 33C0 XOR EAX,EAX 00401145 8A86 70304000 MOV AL,BYTE PTR DS:[ESI+403070] 00401148 83F0 2C XOR EAX,2C 0040114E 8B86 70304000 MOV BYTE PTR DS:[ESI+403070],AL 00401154 46 INC ESI 00401155 E2 EE LOOPD SHORT Register.00401145 00401157 68 70304000 PUSH Register.00403070 0040115C 68 ED030000 PUSH 3ED 00401161 FF35 94304000 PUSH DWORD PTR DS:[403094] 00401167 E8 20000000 CALL <JMP.&user32.SetDlgItemTextA> 0040116C C3 RETN 0040116D CC INT3 </pre>	<div style="border: 2px solid red; padding: 5px; display: inline-block;">We now jump!!!</div>
---	---

```

Text = "xDE\\x0C\\^CK^MA\\x0CE\\x0CBCK\\x0C^IKE_XI^IH\\r"
ControlID = 3ED (1005.)
hwnd = 003B0D44 ('Register Me 1',class='#32770')
SetDlgItemTextA

```

and on the second check we will jump to the good boy as well:



So we have now found one patch that will register this program, no matter what the serial entered is. 🤪 .
 Congratulations.

ASCII Table Plugin

One thing you should do is try to find out what the password is (or what the requirements are for it). To help you, download and install the "Ascii Table" plugin and copy it into your plugin directory. After restarting, choose "Plugins" -> "Ascii table" and show the table. Even though it leaves a lot to be desired, it does give you a quick table of all ASCII values:

ASCII Table				
3D	061	075	00111101	=
3E	062	076	00111110	>
3F	063	077	00111111	?
40	064	100	01000000	@
41	065	101	01000001	A
42	066	102	01000010	B
43	067	103	01000011	C
44	068	104	01000100	D
45	069	105	01000101	E
46	070	106	01000110	F
47	071	107	01000111	G
48	072	110	01001000	H
49	073	111	01001001	I
4A	074	112	01001010	J
4B	075	113	01001011	K
4C	076	114	01001100	L
4D	077	115	01001101	M
4E	078	116	01001110	N
4F	079	117	01001111	O
50	080	120	01010000	P
51	081	121	01010001	Q
52	082	122	01010010	R
53	083	123	01010011	S
54	084	124	01010100	T
55	085	125	01010101	U
56	086	126	01010110	V
57	087	127	01010111	W
58	088	130	01011000	X
59	089	131	01011001	Y
5A	090	132	01011010	Z
5B	091	133	01011011	[
5C	092	134	01011100	\
5D	093	135	01011101]
5E	094	136	01011110	^
5F	095	137	01011111	_
60	096	140	01100000	`
61	097	141	01100001	a
62	098	142	01100010	b
63	099	143	01100011	c
64	100	144	01100100	d

***If anyone would like to take it upon themselves to update or re-do this plugin, I would be eternally grateful. For one, the text should not be highlighted nor editable (why would I want to edit the ASCII chart?). Secondly, making the window sizable would be really great. If anyone does it, please tell me and I will forever be in your debt ***

-Till next time

R4ndom