

The Legend Of Random

Programming and Reverse Engineering

Home

Tutorials

Tools

Contact

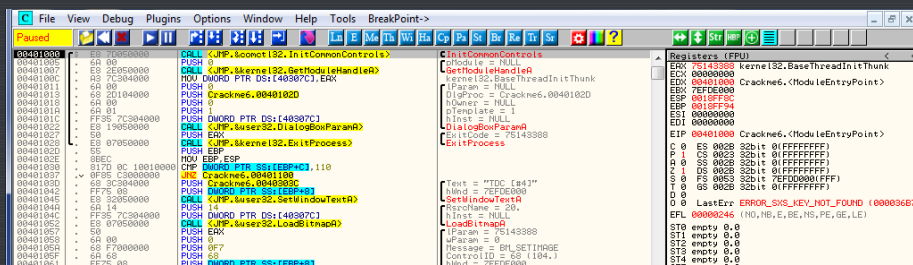
R4ndom's Tutorial #9: No Strings Attached

by R4ndom on Jun.25, 2012, under Beginner, Reverse Engineering, Tutorials

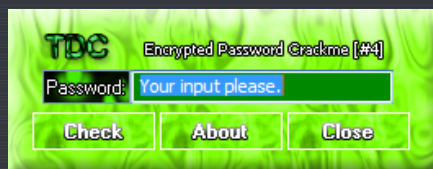
Introduction

In this tutorial we will be adding a new trick to our arsenal; what do you do if there are no usable string in the binary to search for? We will also be introducing a new R.E.T.A.R.D. rule 😊 In this tutorial (as well as the next) we will be studying a crackme called Crackme6 by "TDC", included in the download. Overall, it's not a tough crackme, but we will be doing some advanced analysis on it, preparing for future tutorials. So let's get started...

Go ahead and load Crackme6 into Olly:



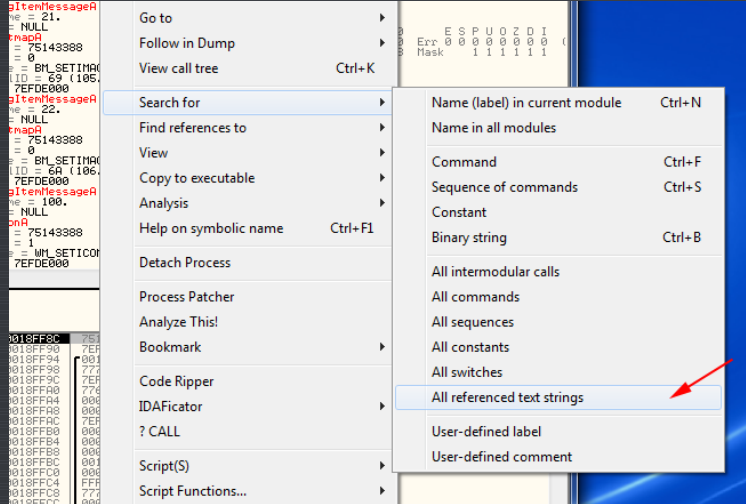
Now, we know the routine; let's run the app and see what we got:



Hmmm, seems simple enough. I entered a password of 1212121212 and here's what I got back:



Pretty straight forward. So let's go to our handy-dandy "search for Strings" and see what we got:



Text strings referenced in Crackme6.exe		
Address	Disassembly	Text string
00401080	CALL <JMP.&comctl32.InitCommonCon (Initial CPU selection)	
0040108D	PUSH Crackme6.0040303C	ASCII "TDC [44]"
004011F7	PUSH Crackme6.00403000	ASCII "NLLJ\\KJAFJK."
00401203	PUSH Crackme6.0040300F	ASCII "NLLJ\\HJNAJJK."
0040122D	PUSH Crackme6.0040301F	ASCII "M)zCji') l fah0/Mnk/xnv."
00401237	PUSH Crackme6.0040301F	ASCII "M)zCji') l fah0/Mnk/xnv."
00401249	PUSH Crackme6.0040301F	ASCII "M)zCji') l fah0/Mnk/xnv."
00401265	PUSH Crackme6.00403000	ASCII "NLLJ\\KJAFJK."
00401282	PUSH Crackme6.00403000	ASCII "NLLJ\\KJAFJK."
004012A1	PUSH Crackme6.0040300F	ASCII "NLLJ\\HJNAJJK."
004012C0	PUSH Crackme6.00403000	ASCII "NLLJ\\KJAFJK."
004012D2	PUSH Crackme6.00403000	ASCII "NLLJ\\KJAFJK."
004012DE	PUSH Crackme6.0040300F	ASCII "NLLJ\\HJNAJJK."
0040131E	PUSH Crackme6.00403045	ASCII "About"

What the hell!!!! Those aren't helpful 😞 What are we supposed to do with those strings!?!? Obviously, this crackme has encrypted the strings (either that or the author speaks a very strange language ;D). Well, this is a good time to introduce

R4ndom's Essential Truths About Reversing Data #3:

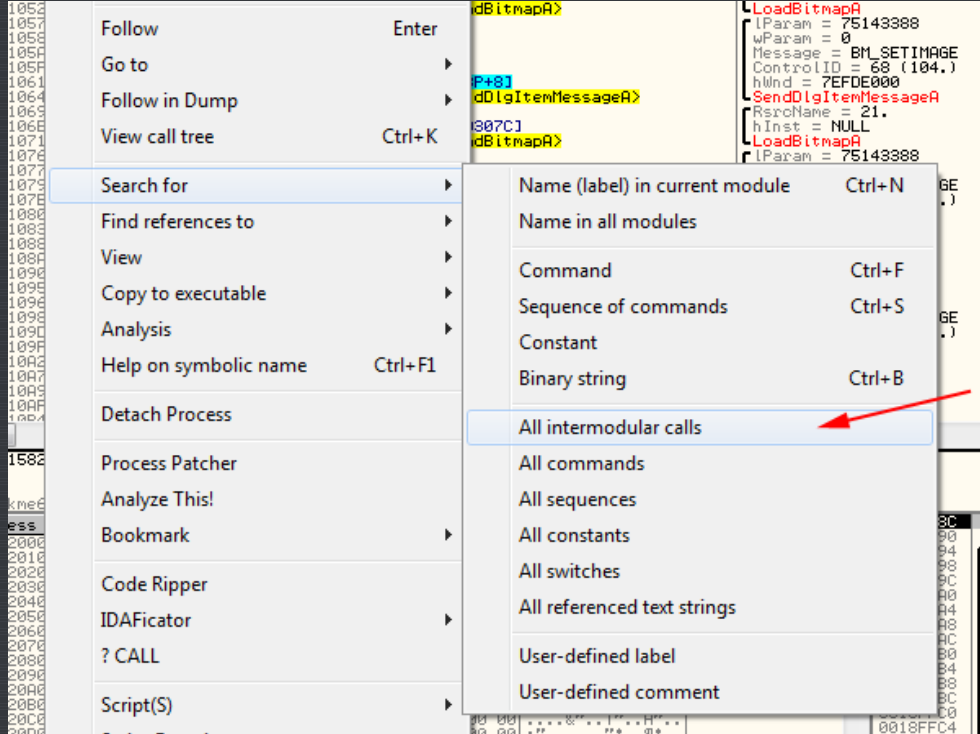
#3. Do not rely on binary's having usable text strings.

Unfortunately, as soon as you start getting into real binaries (like commercial products) most will be packed and/or protected in some way. One of the most obvious ways to hinder a reverse engineer is to encrypt the strings. Frankly, when first investigating a new binary that I am interested in reverse engineering, if I do a search for strings and any come up, I can assume that the binary will probably not present too many challenges. So, you cannot rely on there being any (though it's great when there are 😊)

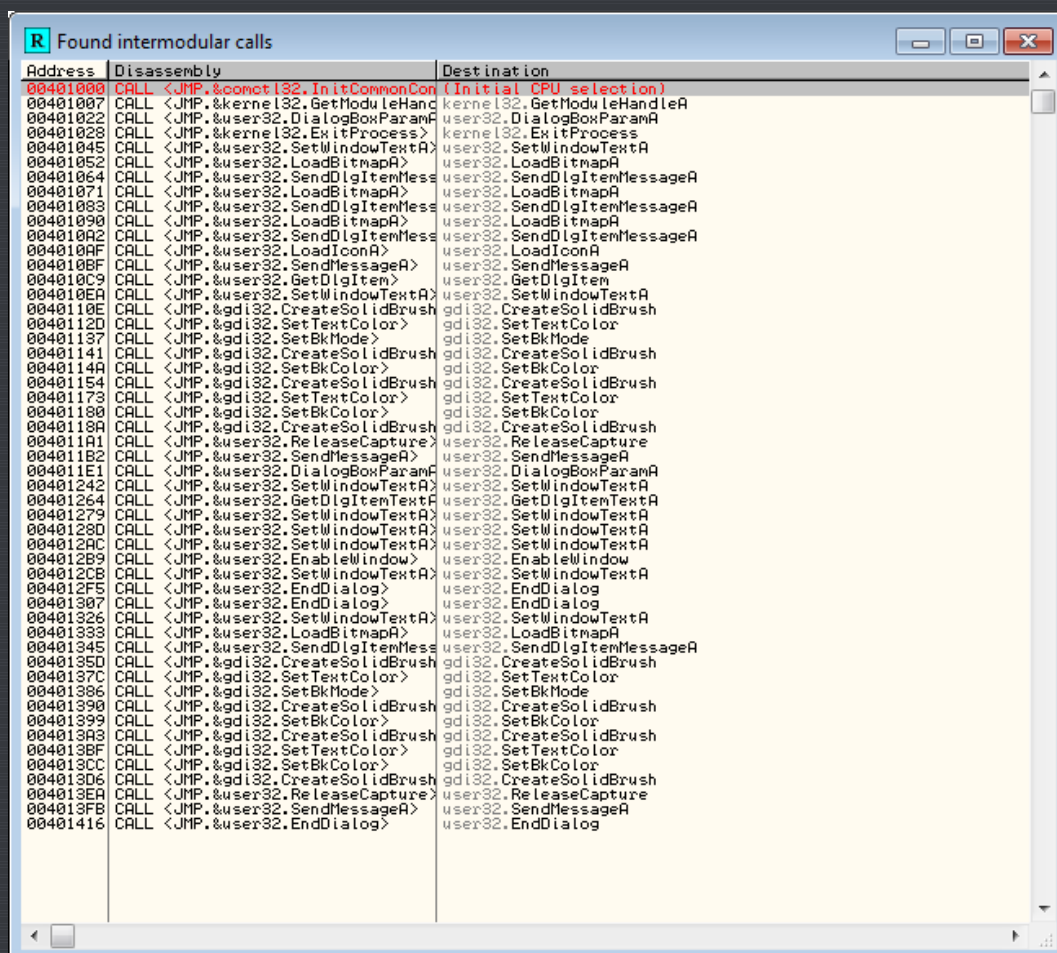
Intermodular Calls

In light of this, I will show you a new trick in the case of no strings; most Windows applications use a standard set of APIs to perform specific actions. For example, MessageBoxA is called if a simple message box is desired, or TerminateProcess is called when the app wishes to end. Since most apps use these same APIs, we can use this to our benefit. For example, there are APIs for getting text from a dialog input box (like a username and serial number). There are APIs for setting timers (used in nag screens where you must wait 10 seconds before hitting 'continue'). There are string compare functions that are called to compare two strings (was the entered password the same as the one stored in the program?). And there are APIs for reading and writing to the registry (to store and retrieve your registration status).

Olly has a way of searching for all of these called APIs. Right click in the disassembly window and choose "Search for" -> "All intermodular calls":



and Olly opens the Found intermodular calls window:



The first thing I usually do is click on the "Destination" heading to sort the list of functions in alphabetical order (instead of by address):

Found intermodular calls		
Address	Disassembly	Destination
00401000	CALL <JMP.&comctl32.InitCommonCon (Initial CPU selection	
00401002	CALL <JMP.&kernel32.GetModuleHandleA	

So now, if you look in the third column, you can see all of the API calls that this crackme makes:

Found intermodular calls		
Address	Disassembly	Destination
0040110E	CALL <JMP.&gdi32.CreateSolidBrush	gdi32.CreateSolidBrush
00401141	CALL <JMP.&gdi32.CreateSolidBrush	gdi32.CreateSolidBrush
00401154	CALL <JMP.&gdi32.CreateSolidBrush	gdi32.CreateSolidBrush
0040118A	CALL <JMP.&gdi32.CreateSolidBrush	gdi32.CreateSolidBrush
0040135D	CALL <JMP.&gdi32.CreateSolidBrush	gdi32.CreateSolidBrush
00401390	CALL <JMP.&gdi32.CreateSolidBrush	gdi32.CreateSolidBrush
004013A3	CALL <JMP.&gdi32.CreateSolidBrush	gdi32.CreateSolidBrush
004013D6	CALL <JMP.&gdi32.CreateSolidBrush	gdi32.CreateSolidBrush
00401022	CALL <JMP.&user32.ShowDialogParamA	user32.ShowDialogParamA
004011E1	CALL <JMP.&user32.ShowDialogParamA	user32.ShowDialogParamA
004012B9	CALL <JMP.&user32.EnableWindow>	user32.EnableWindow
004012F5	CALL <JMP.&user32.EndDialog>	user32.EndDialog
00401307	CALL <JMP.&user32.EndDialog>	user32.EndDialog
00401416	CALL <JMP.&user32.EndDialog>	user32.EndDialog
00401028	CALL <JMP.&kernel32.ExitProcess>	kernel32.ExitProcess
004010C9	CALL <JMP.&user32.GetDlgItem>	user32.GetDlgItem
00401264	CALL <JMP.&user32.GetDlgItemTextA	user32.GetDlgItemTextA
00401007	CALL <JMP.&kernel32.GetModuleHandleA	kernel32.GetModuleHandleA
00401000	CALL <JMP.&comctl32.InitCommonCon (Initial CPU selection)	
00401052	CALL <JMP.&user32.LoadBitmapA>	user32.LoadBitmapA
00401071	CALL <JMP.&user32.LoadBitmapA>	user32.LoadBitmapA
00401090	CALL <JMP.&user32.LoadBitmapA>	user32.LoadBitmapA
00401333	CALL <JMP.&user32.LoadBitmapA>	user32.LoadBitmapA
004010AF	CALL <JMP.&user32.LoadIconA>	user32.LoadIconA
004011A1	CALL <JMP.&user32.ReleaseCapture>	user32.ReleaseCapture
004013EA	CALL <JMP.&user32.ReleaseCapture>	user32.ReleaseCapture
00401064	CALL <JMP.&user32.SendDlgItemMess>	user32.SendDlgItemMessageA
00401083	CALL <JMP.&user32.SendDlgItemMess>	user32.SendDlgItemMessageA
004010A2	CALL <JMP.&user32.SendDlgItemMess>	user32.SendDlgItemMessageA
00401345	CALL <JMP.&user32.SendDlgItemMess>	user32.SendDlgItemMessageA
004010BF	CALL <JMP.&user32.SendMessageA>	user32.SendMessageA
004011B2	CALL <JMP.&user32.SendMessageA>	user32.SendMessageA
004013FB	CALL <JMP.&user32.SendMessageA>	user32.SendMessageA
0040114A	CALL <JMP.&gdi32.SetBkColor>	gdi32.SetBkColor
00401180	CALL <JMP.&gdi32.SetBkColor>	gdi32.SetBkColor
00401399	CALL <JMP.&gdi32.SetBkColor>	gdi32.SetBkColor
004013CC	CALL <JMP.&gdi32.SetBkColor>	gdi32.SetBkColor
00401137	CALL <JMP.&gdi32.SetBkMode>	gdi32.SetBkMode
00401386	CALL <JMP.&gdi32.SetBkMode>	gdi32.SetBkMode
0040112D	CALL <JMP.&gdi32.SetTextColor>	gdi32.SetTextColor
00401173	CALL <JMP.&gdi32.SetTextColor>	gdi32.SetTextColor
0040137C	CALL <JMP.&gdi32.SetTextColor>	gdi32.SetTextColor
004013BF	CALL <JMP.&gdi32.SetTextColor>	gdi32.SetTextColor
00401045	CALL <JMP.&user32.SetWindowTextA>	user32.SetWindowTextA
004010EA	CALL <JMP.&user32.SetWindowTextA>	user32.SetWindowTextA
00401242	CALL <JMP.&user32.SetWindowTextA>	user32.SetWindowTextA
00401279	CALL <JMP.&user32.SetWindowTextA>	user32.SetWindowTextA
0040128D	CALL <JMP.&user32.SetWindowTextA>	user32.SetWindowTextA
004012AC	CALL <JMP.&user32.SetWindowTextA>	user32.SetWindowTextA
004012CB	CALL <JMP.&user32.SetWindowTextA>	user32.SetWindowTextA
00401326	CALL <JMP.&user32.SetWindowTextA>	user32.SetWindowTextA

This is a small program, so there are not that many. Most programs will have hundreds. But in this list you can tell a lot about a binary. You can tell it uses a dialog box as it's main window. You can tell it loads a custom bitmap. And you can tell that it changes some colors in the dialog box.

In a larger application, this window becomes even more invaluable, as it will tell you things such as 1) Are registry APIs called to store and retrieve info from the registry? 2) Are there APIs calling websites to verify that we are actually registered? 3) Are there reading and writing to files APIs where perhaps a registration key will be stored? And when we get into packed binaries, this screen will become even more important (but that's later 😊)

All that being said, there are some specific APIs that reverse engineers are always looking out for, APIs that are used in protection schemes a lot. These include:

DialogBoxParamA
GetDlgItem
GetDlgItemInt
GetDlgItemTextA
GetWindowTextA
GetWindowTextWord

LoadStringA
lstrcmpA

wsprintfA

MessageBeep
MessageBoxA
MessageBoxExA
SendMessageA

SendDlgItemMessageA

ReadFile
WriteFile
CreateFileA

GetPrivateProfileIntA
WritePrivateProfileStringA
GetPrivateProfileStringA

Unfortunately, this does not cover all of the API calls you may run into, but fortunately, most apps use one of the following:

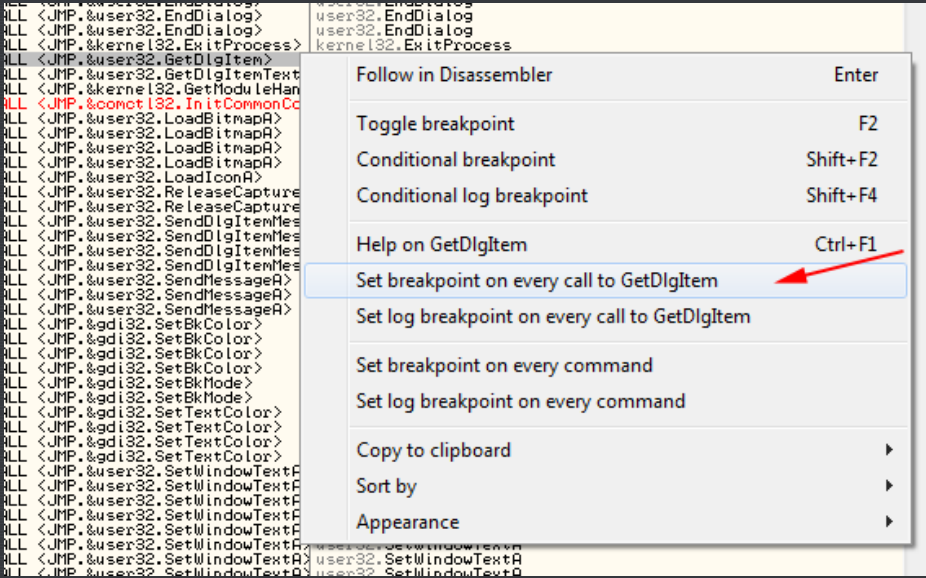
GetDlgItemTextA
GetWindowTextA
IsDlgButtonCheckedA
GetPrivateProfileStringA
GetPrivateProfileIntA
RegQueryValueExA
WritePrivateProfileStringA
GetPrivateProfileIntA

So if you focus on these 8 API calls, you will be able to handle the vast majority of instances. And don't forget, you always have Olly to help with "Get help on symbolic name".

Now, when you look down the list in the calls that Olly has found in our crackme, there are two that are in our short list:

GetDlgItem and GetDlgItemTextA.

What these two API calls do is retrieve whatever text was entered into a dialog box (well, for our tutorial, anyway 😊). Well, in our crackme, this could only mean one thing, our entered password. What we want to do is tell Olly to break anytime he comes across one of these calls. The way to do that is to select the line that has the call you want, right click and select "Set breakpoint on every call to _____", Where _____ will be the name of the API (in this case GetDlgItem):



Now, we can see that Olly has placed a BP on this line.

Found intermodular calls		
Address	Disassembly	Destination
0040110E	CALL <JMP.&gdi32.CreateSolidBrush>	gdi32.CreateSolidBrush
00401141	CALL <JMP.&gdi32.CreateSolidBrush>	gdi32.CreateSolidBrush
00401154	CALL <JMP.&gdi32.CreateSolidBrush>	gdi32.CreateSolidBrush
0040118A	CALL <JMP.&gdi32.CreateSolidBrush>	gdi32.CreateSolidBrush
0040135D	CALL <JMP.&gdi32.CreateSolidBrush>	gdi32.CreateSolidBrush
00401390	CALL <JMP.&gdi32.CreateSolidBrush>	gdi32.CreateSolidBrush
004013A3	CALL <JMP.&gdi32.CreateSolidBrush>	gdi32.CreateSolidBrush
004013D6	CALL <JMP.&gdi32.CreateSolidBrush>	gdi32.CreateSolidBrush
00401022	CALL <JMP.&user32.ShowDialogParamA>	user32.ShowDialogParamA
004011E1	CALL <JMP.&user32.ShowDialogParamA>	user32.ShowDialogParamA
004012B9	CALL <JMP.&user32.EnableWindow>	user32.EnableWindow
004012F5	CALL <JMP.&user32.EndDialog>	user32.EndDialog
00401307	CALL <JMP.&user32.EndDialog>	user32.EndDialog
00401416	CALL <JMP.&user32.EndDialog>	user32.EndDialog
00401028	CALL <JMP.&kernel32.ExitProcess>	kernel32.ExitProcess
00401029	CALL <JMP.&user32.GetDlgItem>	user32.GetDlgItem
00401264	CALL <JMP.&user32.GetDlgItemTextA>	user32.GetDlgItemTextA
00401007	CALL <JMP.&kernel32.GetModuleHandleA>	kernel32.GetModuleHandleA
00401000	CALL <JMP.&comctl32.InitCommonCon>	(Initial CPU selection)
00401052	CALL <JMP.&user32.LoadBitmapA>	user32.LoadBitmapA
00401071	CALL <JMP.&user32.LoadBitmapA>	user32.LoadBitmapA
00401090	CALL <JMP.&user32.LoadBitmapA>	user32.LoadBitmapA
00401333	CALL <JMP.&user32.LoadBitmapA>	user32.LoadBitmapA
004010AF	CALL <JMP.&user32.LoadIconA>	user32.LoadIconA
004011A1	CALL <JMP.&user32.ReleaseCapture>	user32.ReleaseCapture
004013EA	CALL <JMP.&user32.ReleaseCapture>	user32.ReleaseCapture
00401064	CALL <JMP.&user32.SendDlgItemMess>	user32.SendDlgItemMessageA
00401083	CALL <JMP.&user32.SendDlgItemMess>	user32.SendDlgItemMessageA
004010A2	CALL <JMP.&user32.SendDlgItemMess>	user32.SendDlgItemMessageA
00401345	CALL <JMP.&user32.SendDlgItemMess>	user32.SendDlgItemMessageA
004010BF	CALL <JMP.&user32.SendMessageA>	user32.SendMessageA
004011B2	CALL <JMP.&user32.SendMessageA>	user32.SendMessageA
004013FB	CALL <JMP.&user32.SendMessageA>	user32.SendMessageA
0040114A	CALL <JMP.&gdi32.SetBkColor>	gdi32.SetBkColor
00401180	CALL <JMP.&gdi32.SetBkColor>	gdi32.SetBkColor
00401399	CALL <JMP.&gdi32.SetBkColor>	gdi32.SetBkColor
004013CC	CALL <JMP.&gdi32.SetBkColor>	gdi32.SetBkColor
00401137	CALL <JMP.&gdi32.SetBkMode>	gdi32.SetBkMode
00401386	CALL <JMP.&gdi32.SetBkMode>	gdi32.SetBkMode
0040112D	CALL <JMP.&gdi32.SetTextColor>	gdi32.SetTextColor
00401173	CALL <JMP.&gdi32.SetTextColor>	gdi32.SetTextColor
0040137C	CALL <JMP.&gdi32.SetTextColor>	gdi32.SetTextColor
004013BF	CALL <JMP.&gdi32.SetTextColor>	gdi32.SetTextColor
00401045	CALL <JMP.&user32.SetWindowTextA>	user32.SetWindowTextA
004010EA	CALL <JMP.&user32.SetWindowTextA>	user32.SetWindowTextA
00401242	CALL <JMP.&user32.SetWindowTextA>	user32.SetWindowTextA
00401279	CALL <JMP.&user32.SetWindowTextA>	user32.SetWindowTextA
0040128D	CALL <JMP.&user32.SetWindowTextA>	user32.SetWindowTextA
004012AC	CALL <JMP.&user32.SetWindowTextA>	user32.SetWindowTextA
004012CB	CALL <JMP.&user32.SetWindowTextA>	user32.SetWindowTextA
00401326	CALL <JMP.&user32.SetWindowTextA>	user32.SetWindowTextA

We also want to break on the other API call, GetDlgItemTextA, so click on that one, right-click and do the same:

Found intermodular calls		
Address	Disassembly	Destination
0040110E	CALL <JMP.&gdi32.CreateSolidBrush>	gdi32.CreateSolidBrush
00401141	CALL <JMP.&gdi32.CreateSolidBrush>	gdi32.CreateSolidBrush
00401154	CALL <JMP.&gdi32.CreateSolidBrush>	gdi32.CreateSolidBrush
0040118A	CALL <JMP.&gdi32.CreateSolidBrush>	gdi32.CreateSolidBrush
0040135D	CALL <JMP.&gdi32.CreateSolidBrush>	gdi32.CreateSolidBrush
00401390	CALL <JMP.&gdi32.CreateSolidBrush>	gdi32.CreateSolidBrush
004013A3	CALL <JMP.&gdi32.CreateSolidBrush>	gdi32.CreateSolidBrush
004013D6	CALL <JMP.&gdi32.CreateSolidBrush>	gdi32.CreateSolidBrush
00401022	CALL <JMP.&user32.ShowDialogParamA>	user32.ShowDialogParamA
004011E1	CALL <JMP.&user32.ShowDialogParamA>	user32.ShowDialogParamA
004012B9	CALL <JMP.&user32.EnableWindow>	user32.EnableWindow
004012F5	CALL <JMP.&user32.EndDialog>	user32.EndDialog
00401307	CALL <JMP.&user32.EndDialog>	user32.EndDialog
00401416	CALL <JMP.&user32.EndDialog>	user32.EndDialog
00401028	CALL <JMP.&kernel32.ExitProcess>	kernel32.ExitProcess
00401029	CALL <JMP.&user32.GetDlgItem>	user32.GetDlgItem
00401264	CALL <JMP.&user32.GetDlgItemTextA>	user32.GetDlgItemTextA
00401007	CALL <JMP.&kernel32.GetModuleHandleA>	kernel32.GetModuleHandleA
00401000	CALL <JMP.&comctl32.InitCommonCon>	(Initial CPU selection)
00401052	CALL <JMP.&user32.LoadBitmapA>	user32.LoadBitmapA
00401071	CALL <JMP.&user32.LoadBitmapA>	user32.LoadBitmapA
00401090	CALL <JMP.&user32.LoadBitmapA>	user32.LoadBitmapA
00401333	CALL <JMP.&user32.LoadBitmapA>	user32.LoadBitmapA
004010AF	CALL <JMP.&user32.LoadIconA>	user32.LoadIconA
004011A1	CALL <JMP.&user32.ReleaseCapture>	user32.ReleaseCapture
004013EA	CALL <JMP.&user32.ReleaseCapture>	user32.ReleaseCapture
00401064	CALL <JMP.&user32.SendDlgItemMess>	user32.SendDlgItemMessageA
00401083	CALL <JMP.&user32.SendDlgItemMess>	user32.SendDlgItemMessageA
004010A2	CALL <JMP.&user32.SendDlgItemMess>	user32.SendDlgItemMessageA
00401345	CALL <JMP.&user32.SendDlgItemMess>	user32.SendDlgItemMessageA
004010BF	CALL <JMP.&user32.SendMessageA>	user32.SendMessageA
004011B2	CALL <JMP.&user32.SendMessageA>	user32.SendMessageA
004013FB	CALL <JMP.&user32.SendMessageA>	user32.SendMessageA
0040114A	CALL <JMP.&gdi32.SetBkColor>	gdi32.SetBkColor
00401180	CALL <JMP.&gdi32.SetBkColor>	gdi32.SetBkColor
00401399	CALL <JMP.&gdi32.SetBkColor>	gdi32.SetBkColor
004013CC	CALL <JMP.&gdi32.SetBkColor>	gdi32.SetBkColor
00401137	CALL <JMP.&gdi32.SetBkMode>	gdi32.SetBkMode
00401386	CALL <JMP.&gdi32.SetBkMode>	gdi32.SetBkMode
0040112D	CALL <JMP.&gdi32.SetTextColor>	gdi32.SetTextColor
00401173	CALL <JMP.&gdi32.SetTextColor>	gdi32.SetTextColor
0040137C	CALL <JMP.&gdi32.SetTextColor>	gdi32.SetTextColor
004013BF	CALL <JMP.&gdi32.SetTextColor>	gdi32.SetTextColor
00401045	CALL <JMP.&user32.SetWindowTextA>	user32.SetWindowTextA
004010EA	CALL <JMP.&user32.SetWindowTextA>	user32.SetWindowTextA
00401242	CALL <JMP.&user32.SetWindowTextA>	user32.SetWindowTextA
00401279	CALL <JMP.&user32.SetWindowTextA>	user32.SetWindowTextA
0040128D	CALL <JMP.&user32.SetWindowTextA>	user32.SetWindowTextA
004012AC	CALL <JMP.&user32.SetWindowTextA>	user32.SetWindowTextA
004012CB	CALL <JMP.&user32.SetWindowTextA>	user32.SetWindowTextA
00401326	CALL <JMP.&user32.SetWindowTextA>	user32.SetWindowTextA

Now, anytime Olly comes across one of these two calls, he will break (before the call is made). So let's try it. Re-start the crackme and run it. Olly will break on a call to GetDlgItem:

004010AF	. E8 B0040000	CALL <JMP.&user32.LoadIconA>	[LoadIconA
004010B4	. 50	PUSH EAX	{Param = 0
004010B5	. 6A 01	PUSH 1	wParam = 1
004010B7	. 68 80000000	PUSH 80	Message = WM_SETICON
004010BC	. FF75 08	PUSH DWORD PTR SS:[EBP+8]	hWnd = 90DC8
004010BF	. E8 B2040000	CALL <JMP.&user32.SendMessageA>	[SendMessageA
004010C4	. 6A 6B	PUSH 6B	ControlID = 6B (107.)
004010C6	. FF75 08	PUSH DWORD PTR SS:[EBP+8]	hWnd = 00090DC8 ('TDC [#4]',class='#32770')
004010C9	. E8 84040000	CALL <JMP.&user32.GetDlgItem>	[GetDlgItem
004010CE	. A3 80304000	MOV DWORD PTR DS:[403080],EAX	
004010D3	. 6A 12	PUSH 12	
004010D5	. 68 69304000	PUSH Crackme6.00403069	
004010DA	. E8 3C040000	CALL Crackme6.0040151B	
004010DF	. 68 69304000	PUSH Crackme6.00403069	
004010E4	. FF35 80304000	PUSH DWORD PTR DS:[403080]	[Text = "U'z)/fa\k7Fz(\k7Fcjnij!"
004010EA	. E8 8D040000	CALL <JMP.&user32.SetWindowTextA>	hWnd = NULL
004010EF	. 6A 12	PUSH 12	[SetWindowTextA
004010F1	. 68 69304000	PUSH Crackme6.00403069	

Now, since we have not entered anything yet, we're not really interested in what GetDlgItem has gotten in this case, so let's keep going (F9):

00401084	• 50	PUSH EAX	[lParam = 0
00401085	• 6A 01	PUSH 1	wParam = 1
00401087	• 68 80000000	PUSH 80	Message = WM_SETICON
0040108C	• FF75 08	PUSH DWORD PTR SS:[EBP+8]	hWnd = 900C8
0040108F	• E8 B2040000	CALL <JMP.&user32.SendMessageA>	SendMessageA
004010C4	• 6A 6B	PUSH 6B	ControlID = 6B (107.)
004010C6	• FF75 08	PUSH DWORD PTR SS:[EBP+8]	hWnd = 00090DC8 ('TDC [#4]',class='#32770
004010C9	• E8 84040000	CALL <JMP.&user32.GetDlgItem>	GetDlgItem
004010CE	• A3 80304000	MOV DWORD PTR DS:[403080],EAX	
004010D3	• 6A 12	PUSH 12	
004010D5	• 68 69304000	PUSH Crackme6.00403069	
004010DA	• E8 3C040000	CALL Crackme6.0040151B	
004010DF	• 68 69304000	PUSH Crackme6.00403069	
004010E4	• FF35 80304000	PUSH DWORD PTR DS:[403080]	
004010EA	• E8 8D040000	CALL <JMP.&user32.SetWindowTextA>	SetWindowTextA
004010EF	• 6A 12	PUSH 12	
004010F1	• 68 69304000	PUSH Crackme6.00403069	
004010F6	• E8 20040000	CALL Crackme6.0040151B	
004010FB	• J9 0C020000	JMP Crackme6.00401258	
00401100	> 817D 0C 36010000	CMPL DWORD PTR SS:[EBP+10],EAX	
00401107	• 75 13	JNZ SHORT Crackme6.00401292	
00401109	• 68 00000000	PUSH 00000000	
0040110E	• E8 75040000	CALL <JMP.&gdi32.SetTextColor>	SetTextColor
00401113	• C9	LEAVE	
00401114	• C2 1000	RETN 10	
00401117	• J9 F0010000	JMP Crackme6.00401258	
0040111C	> 817D 0C 38010000	CMPL DWORD PTR SS:[EBP+10],EAX	
00401123	• 75 3D	JNZ SHORT Crackme6.00401292	
00401125	• 68 FFFFFFF0	PUSH 0FFFFFFF	
0040112A	• FF75 10	PUSH DWORD PTR SS:[EBP+10]	
0040112D	• E8 68040000	CALL <JMP.&gdi32.SetTextColor>	SetTextColor
00401132	• 6A 01	PUSH 1	
00401134	• FF75 10	PUSH DWORD PTR SS:[EBP+10]	
00401137	• E8 58040000	CALL <JMP.&gdi32.SetBkMode>	SetBkMode

TDC Encrypted Password Crackme [#4]

Password:

Check About Close

B(0.,221.,0.)

dBrush

Color = <WHITE>

hDC = 00120DC2 (window)

SetTextColor

BkMode = TRANSPARENT

hDC = 00120DC2 (window)

SetBkMode

Now enter a password and click "check":



and Olly will break again, this time on GetDlgItemTextA:

00401208	• E8 0E030000	CALL Crackme6.0040151B	
0040120D	• FF05 84304000	INC DWORD PTR DS:[403084]	
00401213	• 813D 84304000 F4	CMPL DWORD PTR DS:[403084],1F4	
0040121D	• 74 0C	JBE SHORT Crackme6.0040122B	
0040121F	• 813D 84304000 F4	CMPL DWORD PTR DS:[403084],1F4	
00401229	• JBE SHORT Crackme6.00401258		
0040122B	• 6A 16	PUSH 16	
0040122D	• 68 1F304000	PUSH Crackme6.0040301F	
00401232	• E8 F4020000	CALL Crackme6.0040151B	
00401237	• 68 1F304000	PUSH Crackme6.0040301F	
0040123C	• FF35 80304000	PUSH DWORD PTR DS:[403080]	
00401242	• E8 35030000	CALL <JMP.&user32.SetWindowTextA>	SetWindowTextA
00401247	• 6A 16	PUSH 16	
00401249	• 68 1F304000	PUSH Crackme6.0040301F	
0040124E	• E8 C8020000	CALL Crackme6.0040151B	
00401253	• J9 B4000000	JMP Crackme6.0040130C	
00401258	• 6A 0C	PUSH 0C	
0040125A	• 68 5D304000	PUSH Crackme6.0040305D	
0040125F	• 6A 6B	PUSH 6B	
00401261	• FF75 08	PUSH DWORD PTR SS:[EBP+8]	
00401264	• E8 EF020000	CALL <JMP.&user32.GetDlgItemTextA>	GetDlgItemTextA
00401269	• 83F8 0B	CMPL EAX,0B	
0040126C	• 72 10	JB SHORT Crackme6.0040127E	
0040126E	• 68 00304000	PUSH Crackme6.00403000	
00401273	• FF35 80304000	PUSH DWORD PTR DS:[403080]	
00401279	• E8 FE020000	CALL <JMP.&user32.SetWindowTextA>	SetWindowTextA
0040127E	• 85C0	TEST EAX,EAX	
00401280	• 75 10	JNZ SHORT Crackme6.00401292	
00401282	• 68 00304000	PUSH Crackme6.00403000	
00401287	• FF35 80304000	PUSH DWORD PTR DS:[403080]	
00401290	• E8 EA020000	CALL <JMP.&user32.SetWindowTextA>	SetWindowTextA
00401292	• 50	PUSH EAX	
00401293	• 68 5D304000	PUSH Crackme6.0040305D	
00401298	• E8 84010000	CALL Crackme6.00401421	
0040129D	• 0BC0	OR EAX,EAX	
0040129F	• 75 1F	JNZ SHORT Crackme6.004012C0	
004012A1	• 68 0F304000	PUSH Crackme6.0040300F	
004012A6	• FF35 80304000	PUSH DWORD PTR DS:[403080]	
004012AC	• E8 CB020000	CALL <JMP.&user32.SetWindowTextA>	SetWindowTextA
004012B1	• 6A 00	PUSH 0	
004012B3	• FF35 80304000	PUSH DWORD PTR DS:[403080]	
004012B9	• E8 88020000	CALL <JMP.&user32.EnableWindow>	EnableWindow
004012BE	• EB 10	JMP SHORT Crackme6.004012D0	
004012C0	• 68 00304000	PUSH Crackme6.00403000	
004012C5	• FF35 80304000	PUSH DWORD PTR DS:[403080]	
004012C8	• E8 AC020000	CALL <JMP.&user32.SetWindowTextA>	SetWindowTextA
004012D0	• 6A 0E	PUSH 0E	
004012D2	• 68 00304000	PUSH Crackme6.00403000	
004012D7	• E8 3F020000	CALL Crackme6.0040151B	

ASCII "M)z(cji')lfah0/Mnk/xnv."

Text = "M)z(cji')lfah0/Mnk/xnv."

hWnd = 00120DC2 (class='Edit',parent=00090DC8)

SetWindowTextA

ASCII "M)z(cji')lfah0/Mnk/xnv."

Count = C (12.)

Buffer = Crackme6.0040305D

ControlID = 6B (107.)

hWnd = 00090DC8 ('TDC [#4]',class='#32770')

GetDlgItemTextA

Text = "ACCESS DENIED!"

hWnd = 00120DC2 (class='Edit',parent=00090DC8)

SetWindowTextA

Crackme6.0040300F

Text = "ACCESS DENIED!"

hWnd = 00120DC2 (class='Edit',parent=00090DC8)

SetWindowTextA

Crackme6.0040300F

Crackme6.0040300F

Text = "ACCESS GRANTED!"

hWnd = 00120DC2 (class='Edit',parent=00090DC8)

SetWindowTextA

Enable = FALSE

hWnd = 00120DC2 (class='Edit',parent=00090DC8)

EnableWindow

Text = "ACCESS DENIED!"

hWnd = 00120DC2 (class='Edit',parent=00090DC8)

SetWindowTextA

ASCII "ACCESS DENIED!"

If you look around a little, you'll see that we're in the right place 😊 Funny, none of those strings were there when we initially searched for them 😊

Cracking the App

Let's take a quick look around... We see a jump (JB) past the first "ACCESS DENIED", so we'll have to pay attention to that:

00401261	• FF75 08	PUSH 6B	Count = 6 (12.)
00401264	• E8 EF020000	CALL <JMP.&user32.GetDlgItemTextA>	Buffer = Crackme6.0040305D
00401269	• 83F8 0B	CMP EAX, 0B	ControlID = 6B (107.)
0040126C	• 72 10	JB SHORT Crackme6.0040127E	hWnd = 00090DC8 ('TDC [#4]',class='#32770')
0040126E	• 68 00304000	PUSH Crackme6.00403000	GetDlgItemTextA
00401273	• FF35 80304000	PUSH DWORD PTR DS:[403080]	Text = "ACCESS DENIED!"
00401279	• E8 FE020000	CALL <JMP.&user32.SetWindowTextA>	hWnd = 00120DC2 (class='Edit',parent=00090DC8)
0040127E	• 85C0	TEST EAX, EAX	SetWindowTextA
00401280	• 75 10	JNZ SHORT Crackme6.00401292	Crackme6.0040300F
00401282	• 68 00304000	PUSH Crackme6.00403000	Text = "ACCESS DENIED!"
00401283	• E8 EA020000	CALL <JMP.&user32.SetWindowTextA>	hWnd = 00120DC2 (class='Edit',parent=00090DC8)
00401284	• 50	PUSH EAX	SetWindowTextA
00401285	• 68 5D304000	PUSH Crackme6.0040305D	Crackme6.0040300F
00401286	• E8 84010000	CALL Crackme6.00401421	
00401287	• 0BC0	OR EAX, EAX	
00401288	• 75 1F	JNZ SHORT Crackme6.004012C0	
00401289	• 68 0F304000	PUSH Crackme6.0040300F	
0040128A	• FF35 80304000	PUSH DWORD PTR DS:[403080]	Text = "ACCESS GRANTED!"
0040128B	• E8 FA020000	CALL <JMP.&user32.SetWindowTextA>	hWnd = 00120DC2 (class='Edit',parent=00090DC8)

Then there's a jump (JNZ) past the second bad boy, so we'll add that to the list. Then we would fall through to the good boy, so basically we want to make sure we jump both of those jumps:

00401258	• 6A 0C	PUSH 0C	Count = 6 (12.)
00401259	• 68 5D304000	PUSH Crackme6.0040305D	Buffer = Crackme6.0040305D
0040125F	• 6A 0B	PUSH 6B	ControlID = 6B (107.)
00401261	• FF75 08	PUSH DWORD PTR SS:[EBP+8]	hWnd = 00090DC8 ('TDC [#4]',class='#32770')
00401264	• E8 EF020000	CALL <JMP.&user32.GetDlgItemTextA>	GetDlgItemTextA
00401269	• 83F8 0B	CMP EAX, 0B	
0040126C	• 72 10	JB SHORT Crackme6.0040127E	
0040126E	• 68 00304000	PUSH Crackme6.00403000	Text = "ACCESS DENIED!"
00401273	• FF35 80304000	PUSH DWORD PTR DS:[403080]	hWnd = 00120DC2 (class='Edit',parent=00090DC8)
00401279	• E8 FE020000	CALL <JMP.&user32.SetWindowTextA>	SetWindowTextA
0040127E	• 85C0	TEST EAX, EAX	Crackme6.0040300F
00401280	• 75 10	JNZ SHORT Crackme6.00401292	Text = "ACCESS DENIED!"
00401282	• 68 00304000	PUSH Crackme6.00403000	hWnd = 00120DC2 (class='Edit',parent=00090DC8)
00401283	• FF35 80304000	PUSH DWORD PTR DS:[403080]	SetWindowTextA
00401284	• E8 EA020000	CALL <JMP.&user32.SetWindowTextA>	Crackme6.0040300F
00401285	• 50	PUSH EAX	
00401286	• 68 5D304000	PUSH Crackme6.0040305D	
00401287	• E8 84010000	CALL Crackme6.00401421	
00401288	• 0BC0	OR EAX, EAX	Crackme6.0040300F
00401289	• 75 1F	JNZ SHORT Crackme6.004012C0	
0040128A	• 68 0F304000	PUSH Crackme6.0040300F	Text = "ACCESS GRANTED!"
0040128B	• FF35 80304000	PUSH DWORD PTR DS:[403080]	hWnd = 00120DC2 (class='Edit',parent=00090DC8)
0040128C	• E8 FA020000	CALL <JMP.&user32.SetWindowTextA>	SetWindowTextA

Let's try it and see if we're right. Run the app again and we should break at our GetDlgItemTextA instruction (remember to bypass the first break).

00401261	• FF75 08	PUSH DWORD PTR SS:[EBP+8]	hWnd = 000A0DC8 ('TDC [#4]',class='#32770')
00401264	• E8 EF020000	CALL <JMP.&user32.GetDlgItemTextA>	GetDlgItemTextA
00401269	• 83F8 0B	CMP EAX, 0B	
0040126C	• 72 10	JB SHORT Crackme6.0040127E	
0040126E	• 68 00304000	PUSH Crackme6.00403000	Text = "ACCESS DENIED!"
00401273	• FF35 80304000	PUSH DWORD PTR DS:[403080]	hWnd = 00120DB4 (class='Edit',parent=000A0DC8)
00401279	• E8 FE020000	CALL <JMP.&user32.SetWindowTextA>	SetWindowTextA
0040127E	• 85C0	TEST EAX, EAX	
00401280	• 75 10	JNZ SHORT Crackme6.00401292	Text = "ACCESS DENIED!"
00401282	• 68 00304000	PUSH Crackme6.00403000	hWnd = 00120DB4 (class='Edit',parent=000A0DC8)
00401283	• FF35 80304000	PUSH DWORD PTR DS:[403080]	SetWindowTextA
00401284	• E8 FA020000	CALL <JMP.&user32.SetWindowTextA>	

Since this is a JB jump, we need to flip the carry bit:

CIP	00401
C	0 ES 0
P	1 CS 0
A	0 SS 0
Z	1 DS 0
S	0 FS 0

So that will force the jump. Now we're going to do another TEST and stop at the jump at address 401280.

Notice that our password has shown up in the comments column 😊

0040125F	• 6A 0B	PUSH 6B	ControlID = 6B (107.)
00401261	• FF75 08	PUSH DWORD PTR SS:[EBP+8]	hWnd = 000A0DC8 ('TDC [#4]',class='#32770')
00401264	• E8 EF020000	CALL <JMP.&user32.GetDlgItemTextA>	GetDlgItemTextA
00401269	• 83F8 0B	CMP EAX, 0B	
0040126C	• 72 10	JB SHORT Crackme6.0040127E	
0040126E	• 68 00304000	PUSH Crackme6.00403000	Text = "ACCESS DENIED!"
00401273	• FF35 80304000	PUSH DWORD PTR DS:[403080]	hWnd = 00120DB4 (class='Edit',parent=000A0DC8)
00401279	• E8 FE020000	CALL <JMP.&user32.SetWindowTextA>	SetWindowTextA
0040127E	• 85C0	TEST EAX, EAX	
00401280	• 75 10	JNZ SHORT Crackme6.00401292	Text = "ACCESS DENIED!"
00401282	• 68 00304000	PUSH Crackme6.00403000	hWnd = 00120DB4 (class='Edit',parent=000A0DC8)
00401283	• FF35 80304000	PUSH DWORD PTR DS:[403080]	SetWindowTextA
00401284	• E8 EA020000	CALL <JMP.&user32.SetWindowTextA>	
00401285	• 50	PUSH EAX	
00401286	• 68 5D304000	PUSH Crackme6.0040305D	ASCII "12121212121" ←
00401287	• E8 84010000	CALL Crackme6.00401421	
00401288	• 0BC0	OR EAX, EAX	
00401289	• 75 1F	JNZ SHORT Crackme6.004012C0	
0040128A	• 68 0F304000	PUSH Crackme6.0040300F	Text = "ACCESS GRANTED!"
0040128B	• FF35 80304000	PUSH DWORD PTR DS:[403080]	hWnd = 00120DB4 (class='Edit',parent=000A0DC8)

This jump we want to take as it jumps the second bad boy, so just keep stepping until we get to the next JNZ instruction at 40129F:

0040125F	. 6A 6B	PUSH 6B	ControlID = 6B (107.)
00401261	. FF75 0B	PUSH DWORD PTR SS:[EBP+8]	hWnd = 000A0DC8 ('TDC [#4]',class='#32770')
00401264	. E8 EF020000	CALL <JMP.&user32.GetDlgItemTextA>	GetDlgItemTextA
00401269	. 83F8 0B	CMP EAX,0B	
0040126C	. 72 10	JB SHORT Crackme6.0040127E	
0040126E	. 68 00304000	PUSH Crackme6.00403000	Text = "ACCESS DENIED!"
00401273	. FF35 80304000	PUSH DWORD PTR DS:[403080]	hWnd = 00120DB4 (class="Edit",parent=000A0DC8)
00401279	. E9 FE020000	CALL <JMP.&user32.SetWindowTextA>	SetWindowTextA
0040127E	> 85C0	TEST EAX,EAX	
00401280	. 75 10	JNZ SHORT Crackme6.00401292	Text = "ACCESS DENIED!"
00401282	. 68 00304000	PUSH Crackme6.00403000	hWnd = 00120DB4 (class="Edit",parent=000A0DC8)
00401287	. FF35 80304000	PUSH DWORD PTR DS:[403080]	SetWindowTextA
0040128D	. E8 EA020000	CALL <JMP.&user32.SetWindowTextA>	
00401292	> 50	PUSH EAX	
00401293	. 68 5D304000	PUSH Crackme6.0040305D	
00401298	. E8 84010000	CALL Crackme6.00401421	
0040129D	. 0BC0	OR EAX,EAX	
0040129F	. 75 1F	JNZ SHORT Crackme6.004012C0	Text = "ACCESS GRANTED!"
004012A1	. 68 0F304000	PUSH Crackme6.0040300F	hWnd = 00120DB4 (class="Edit",parent=000A0DC8)
004012A6	. FF35 80304000	PUSH DWORD PTR DS:[403080]	SetWindowTextA
004012AC	. E8 CB020000	CALL <JMP.&user32.SetWindowTextA>	Enable = FALSE
004012B1	. 6A 00	PUSH 0	hWnd = 00120DB4 (class="Edit",parent=000A0DC8)
004012B3	. FF35 80304000	PUSH DWORD PTR DS:[403080]	EnableWindow
004012B9	. E8 88020000	CALL <JMP.&user32.EnableWindow>	
004012BE	. EB 10	JMP SHORT Crackme6.004012D0	Text = "ACCESS DENIED!"
004012C0	. 68 00304000	PUSH Crackme6.00403000	hWnd = 00120DB4 (class="Edit",parent=000A0DC8)
004012C5	. FF35 80304000	PUSH DWORD PTR DS:[403080]	SetWindowTextA
004012CB	. E8 AC020000	CALL <JMP.&user32.SetWindowTextA>	
004012D0	> 6A 0E	PUSH 0E	
004012D2	. 68 00304000	PUSH Crackme6.00403000	ASCII "ACCESS DENIED!"
004012D7	. E8 3F020000	CALL Crackme6.0040151B	

OK, this is now going to jump past our good messages, so we want to stop that from happening. You know what to do:

C	0	ES	0
E	0	CS	0
D	0	SS	0
I	1	DS	0
N	0	FS	0
T	0	GS	0

Now run the app (F9) and you will see we have successfully cracked the program.



Homework

For a challenge, try patching this crackme yourself, based on the flags we have changed. After saving he patched program, you should be able to run it and enter any password (less than 11 digits) and it will say "Access Granted". Keep in mind that there are several patches that can be done to accomplish this, so if one doesn't work, keep looking.

Extra Credit: Patch the crackme so that your password can be any length.

-Till next time

R4ndom

ps. You can get a homework hint [here](#).

pps. I will post the solution in a couple days.