

第十六章（上）：Windows 消息的处理

好，在干掉了两个病毒（一个是我身体上的，另一个是电脑的）以后，我最终还是上传了最新的教程。本章是其中的三分之一，所有三章处理的都是同一个 **crackme**（相当难的一个），是 **Detten** 写的 **Crackme12**。第一部分我们将学习 **Windows** 消息是怎么工作的。第二部分是关于自修改代码，该部分我们也会破解该应用。在第三及最后那部分我们介绍暴力破解。如你所猜测的，在第三部分我们将爆破这个二进制文件。每一部分都会继续前面部分的研究结果。

这个系列的三部分都比较有挑战性，不过我保证如果你花时间并自己动手实践，你会获得逆向领域中的很关键的知识。记住，如果有任何问题，就在[论坛](#)里随意发问。我也会在每一章的最后布置作业，让你为下一章做好准备。课后作业是真正学习的地方😁。

一如既往，你需要的相关文件可以在[下载](#)页下载。对于第一部分，下载中包括 **crackme**，以及一份 **Windows** 消息备忘单。

那么，事不宜迟，咱们开始吧...

一、Windows 消息简介

本章我们将会讨论 **Windows** 消息，以及处理它们的过程。几乎所有的程序，除了用 **Visual Basic** *唉*、**.NET** 或 **java** 写的程序以外，任务都是通过使用消息驱动回调过程来完成的。意思就是，与 **DOS** 时代程序不同，在 **Windows** 中你只需设置窗口，提供各种你想要显示的设置、位图、菜单项等，然后你再提供一个循环运行到程序结束。这个循环的唯一责任是从 **Windows** 接收“消息”，然后再将提发送到我们应用的回调函数。这些消息可以是任何东西，从移动鼠标到点击一个按钮，到点击“X”来关闭一个应用。当我们在做一个 **Windows** 程序时，我们要在 **WinMain** 过程中编写一个无限循环，以及一个无论消息什么时候到来都可以调用的地址。这个地址就是回调（函数的地址。译者注：这几个字是我补充的）。然后该循环用我们提供的地址将其接收到的消息发送给我们的回调函数，在回调函数中我们决定是否对特定的消息做处理，或只让 **Windows** 处理它。

例如，我们想要显示一个带有 **OK** 按钮的警告消息框。我们只关心 **OK** 按钮被点击的消息。我们不关心用户是否移动了窗口（**WM-MOVE** 消息），或者是点击了窗口 **OK** 按钮以外的某处（**WM-MOUSEBUTTONDOWN** 消息）。不过当 **OK** 按钮被点击的消息传来的时候，那就是我们要做些什么的时候了。所有我们不想处理的消息，**Windows** 会为我们处理。对于我们想要处理的消息，我们只需重写相关消息的处理，做我们想做的。

设置窗口以及包含循环的主过程叫做 **WinMain**，如果是窗口的话回调函数通常被叫做 **WndProc**，如果是对话框的话回调函数通常被叫做 **DlgProc**，虽然这些名字可以任何其他的東西。

我在下载中包含了一个所有 Windows 消息的指南，你在学习本章时应该打开看看。你可以在[教程](#)页下载到所有的文件。你也可以在[工具](#)页面下载 windows 消息备忘单。

二、载入应用

Olly 载入 Crackme12. exe，咱们来看看：

00401000	6A 00	PUSH 0	pModule = NULL
00401002	E8 C5040000	CALL <JMP.&KERNEL32.GetModuleHandleA>	GetModuleHandleA
00401007	A3 28304000	MOV DWORD PTR DS:[403028],EAX	kernel32.BaseThreadInitThunk
0040100C	6A 00	PUSH 0	lParam = NULL
0040100E	68 2B104000	PUSH crackme1.0040102B	DlgProc = crackme1.0040102B
00401013	6A 00	PUSH 0	hOwner = NULL
00401015	68 20030000	PUSH 320	pTemplate = 320
0040101A	FF35 28304000	PUSH DWORD PTR DS:[403028]	hInst = NULL
00401020	E8 8F040000	CALL <JMP.&USER32.DialogBoxParamA>	DialogBoxParamA
00401025	50	PUSH EAX	ExitCode = 761F3388
00401026	E8 9B040000	CALL <JMP.&KERNEL32.ExitProcess>	ExitProcess
0040102B	55	PUSH EBP	
0040102C	8BEC	MOV EBP,ESP	
0040102E	817D 0C 10010000	CMP [ARG.2],110	
00401035	75 37	JNZ SHORT crackme1.0040106E	
00401037	C705 48304000 00	MOV DWORD PTR DS:[403048],0	

这个一个标准的应用程序，在使用一个对话框作为主窗口，看起来像是用 C 或 C++写的。

如果程序使用的是常规窗口而不是对话框窗口，它看起来会不一样。参见下面的。

注意参数是被压入堆栈，以及对 DialogBoxParamA 的调用。这个将对话框设置成程序的主窗口（而不是普通窗口，不过别太在意这些细节，这真的没什么关系）。咱们看看有关 DialogBoxParamA 的帮助怎么说：

Win32 Programmer's Reference

File Edit Bookmark Options Help

Contents Index Back << >>

DialogBoxParam

Quick Info Overview Group

The **DialogBoxParam** function creates a modal dialog box from a dialog box template resource. Before displaying the dialog box, the function passes an application-defined value to the dialog box procedure as the *lParam* parameter of the [WM_INITDIALOG](#) message. An application can use this value to initialize dialog box controls.

```
int DialogBoxParam(
    HINSTANCE hInstance,           // handle to application instance
    LPCTSTR lpTemplateName,        // identifies dialog box template
    HWND hWndParent,               // handle to owner window
    DLGPROC lpDialogFunc,          // pointer to dialog box procedure
    LPARAM dwInitParam             // initialization value
);
```

Parameters

hInstance
Identifies an instance of the module whose executable file contains the dialog box template.

lpTemplateName
Identifies the dialog box template. This parameter is either the pointer to a null-terminated character string that specifies the name of the dialog box template or an integer value that specifies the resource identifier of the dialog box template. If the parameter specifies a resource identifier, its high-order word must be zero and its low-order word must contain the identifier. You can use the [MAKEINTRESOURCE](#) macro to create this value.

hWndParent
Identifies the window that owns the dialog box.

lpDialogFunc
Points to the dialog box procedure. For more information about the dialog box procedure, see the [DialogProc](#) callback function.

dwInitParam
Specifies the value to pass to the dialog box in the *lParam* parameter of the [WM_INITDIALOG](#) message.

Return Values

If the function succeeds, the return value is the value of the *nResult* parameter specified in the call to the [EndDialog](#) function used to terminate the dialog box.

If the function fails, the return value is -1.

Remarks

The **DialogBoxParam** function uses the [CreateWindowEx](#) function to create the dialog box. **DialogBoxParam** then

对于我们的目的来说，这个 **CALL** 的最重要的东西是 **DLGPROC** 的地址。它是我们应用程序的回调函数的地址，用于处理所有的 **windows** 消息。回头看看反汇编代码，能够清晰的看到这个地址：

00401000	6A 00	PUSH 0	[pModule = NULL
00401002	E8 C5040000	CALL <JMP.&KERNEL32.GetModuleHandleA>	GetModuleHandleA
00401007	A3 28304000	MOV DWORD PTR DS:[403028],EAX	kernel32.BaseThreadInitThunk
0040100C	6A 00	PUSH 0	lParam = NULL
0040100E	68 2B104000	PUSH crackme1.0040102B	DlgProc = crackme1.0040102B
00401013	6A 00	PUSH 0	hOwner = NULL
00401015	68 20030000	PUSH 320	pTemplate = 320
0040101A	FF35 28304000	PUSH DWORD PTR DS:[403028]	hInst = NULL
00401020	E8 8F040000	CALL <JMP.&USER32.DialogBoxParamA>	DialogBoxParamA
00401025	50	PUSH EAX	ExitCode = 761F3388
00401026	E8 9B040000	CALL <JMP.&KERNEL32.ExitProcess>	ExitProcess
0040102B	55	PUSH EBP	
0040102C	8BEC	MOV EBP,ESP	
0040102F	017D 0C 10010000	CMP EBP,10010000	

这里它是 40102B。咱们过去看看它长啥样。这将是...

三、主对话框消息处理回调函数

这里我们能够看到它的开头：

00401025	50	PUSH EAX	
00401026	E8 9B040000	CALL <JMP.&KERNEL32.ExitProcess>	[ExitCode = 761F3388 ExitProcess
0040102B	55	PUSH EBP	
0040102C	8BEC	MOV EBP,ESP	
0040102E	817D 0C 10010000	CMP [ARG_2],110	
00401035	75 37	JNZ SHORT crackme1.0040106E	
00401037	C705 48304000 0000	MOV DWORD PTR DS:[403048],0	
00401041	C705 38304000 0000	MOV DWORD PTR DS:[403038],0DEAD	
0040104B	C705 3C304000 0000	MOV DWORD PTR DS:[40303C],0DEAD	
00401055	C705 48304000 42424242	MOV DWORD PTR DS:[403040],42424242	
0040105F	C705 4C304000 0000	MOV DWORD PTR DS:[40304C],crackme1.00401081	ASCII "An error occurred"
00401069	E9 DE010000	JMP crackme1.0040124C	
0040106E	837D 0C 10	CMP [ARG_2],10	
00401072	75 0D	JNZ SHORT crackme1.00401081	
00401074	FF75 08	PUSH [ARG_1]	
00401077	E8 32040000	CALL <JMP.&USER32.DestroyWindow>	[hWnd = 7EFDE000 DestroyWindow
0040107C	E9 CB010000	JMP crackme1.0040124C	
00401081	817D 0C 11010000	CMP [ARG_2],111	
00401088	0F85 B5010000	JNZ crackme1.00401243	
0040108E	8B45 10	MOV EAX,[ARG_3]	
00401091	8B55 10	MOV EDX,[ARG_3]	
00401094	C1EA 10	SHR EDX,10	
00401097	66:0BD2	OR DX,DX	
0040109A	0F85 AC010000	JNZ crackme1.0040124C	
004010A0	66:83F8 65	CMP AX,65	
004010A4	75 0C	JNZ SHORT crackme1.004010B2	
004010A6	6A 01	PUSH 1	
004010A8	E8 F8010000	CALL crackme1.004012A5	
004010AD	E9 40010000	JMP crackme1.004011F2	
004010B2	66:83F8 66	CMP AX,66	
004010B6	75 0C	JNZ SHORT crackme1.004010C4	
004010B8	6A 02	PUSH 2	
004010BA	E8 E6010000	CALL crackme1.004012A5	
004010BF	E9 2E010000	JMP crackme1.004011F2	
004010C4	66:83F8 67	CMP AX,67	
004010C8	75 0C	JNZ SHORT crackme1.004010D6	
004010CA	6A 03	PUSH 3	
004010CC	E8 D4010000	CALL crackme1.004012A5	
004010D1	E9 1C010000	JMP crackme1.004011F2	
004010D6	66:83F8 68	CMP AX,68	
004010DA	75 0C	JNZ SHORT crackme1.004010E8	
004010DC	6A 04	PUSH 4	
004010DE	E8 C2010000	CALL crackme1.004012A5	
004010E3	E9 0A010000	JMP crackme1.004011F2	
004010E8	66:83F8 69	CMP AX,69	
004010EC	75 0C	JNZ SHORT crackme1.004010FA	
004010EE	6A 05	PUSH 5	
004010F0	E8 B0010000	CALL crackme1.004012A5	
004010F5	E9 F8000000	JMP crackme1.004011F2	
004010FA	66:83F8 6A	CMP AX,6A	
004010FE	75 0C	JNZ SHORT crackme1.0040110C	
00401100	6A 06	PUSH 6	
00401102	E8 9E010000	CALL crackme1.004012A5	
00401107	E9 E6000000	JMP crackme1.004011F2	
0040110C	66:83F8 6B	CMP AX,6B	
00401110	75 0C	JNZ SHORT crackme1.0040111E	
00401112	6A 07	PUSH 7	
00401114	E8 8C010000	CALL crackme1.004012A5	
00401119	E9 D4000000	JMP crackme1.004011F2	
0040111E	66:83F8 6C	CMP AX,6C	
00401122	75 0C	JNZ SHORT crackme1.00401130	
00401124	6A 08	PUSH 8	
00401126	E8 7A010000	CALL crackme1.004012A5	
0040112B	E9 C2000000	JMP crackme1.004011F2	
00401130	66:83F8 6D	CMP AX,6D	
00401134	75 0C	JNZ SHORT crackme1.00401142	
00401136	6A 09	PUSH 9	
00401138	E8 68010000	CALL crackme1.004012A5	
0040113D	E9 B0000000	JMP crackme1.004011F2	
00401142	66:83F8 6E	CMP AX,6E	
00401146	75 0C	JNZ SHORT crackme1.00401154	
00401148	6A 0A	PUSH 0A	
0040114A	E8 56010000	CALL crackme1.004012A5	
0040114F	E9 9E000000	JMP crackme1.004011F2	
00401154	66:83F8 6F	CMP AX,6F	
00401158	75 0C	JNZ SHORT crackme1.00401166	
0040115A	6A 0B	PUSH 0B	
0040115C	E8 44010000	CALL crackme1.004012A5	
00401161	E9 8C000000	JMP crackme1.004011F2	
00401166	66:83F8 70	CMP AX,70	

这是一个相当普通的 `DlgProc`。它通常就是一个真正的大的 `switch` 语句，虽然在汇编形式下，变成了一个真正的大的 `if/then` 语句。如果你通读了我的上一章，这个看起来应该比较熟悉，本例中它唯一的不同是，Oilly 无法指出 `case` 标签 (ie. `Case 113 (WM_TIMER)`)。

这个过程在这里有一个原因：是为了响应我们感兴趣的 `windows` 消息。如果你仔细看的话，你会看到一堆比较和跳转语句。这是将每一部分代码与 `Windows` 发送过来的消息 ID 进行核对。如果代码匹配了其中一个比较语句，该代码就会运行（译者注：感觉作者这几句怎么那么别扭呢）。否则，它就会尝试所有的比较，没有匹配的话，它就会被发送给 `Windows`，让 `Windows` 来处理。

咱们来深入的看看这个过程。继续运行程序：



至少可以说它是一个很奇怪的 **crackme**。咱们来把玩把玩。你会发现你可以持续点击按钮，不过什么都不会发生，虽然它有一个“clear”按钮。看起来它希望咱们输入一个指定的代码，如果我们不这么做，程序就什么都不做。

现在咱们在 **DlgProc** 代码的起始处也就是 **40102B** 设置一个 **BP**。重启应用，我们能观察到有消息来了：

00401020	. E8 8F040000	CALL <JMP.&USER32.ShowDialogParamA>	DialogBoxParamA
00401025	. 50	PUSH EAX	ExitCode = C0000000
00401026	. E8 9B040000	CALL <JMP.&KERNEL32.ExitProcess>	ExitProcess
0040102B	. 55	PUSH EBP	
0040102C	. 8BEC	MOV EBP,ESP	
0040102E	. 817D 0C 10010000	CMP [ARG.2],110	
00401035	. 75 37	JNZ SHORT crackme1.0040106E	
00401037	. C705 48304000 000	MOV DWORD PTR DS:[403048],0	
00401041	. C705 38304000 000	MOV DWORD PTR DS:[403038],0DEAD	
00401048	. C705 3C304000 000	MOV DWORD PTR DS:[40303C],0DEAD	
00401055	. C705 40304000 424	MOV DWORD PTR DS:[403040],42424242	
0040105F	. C705 4C304000 000	MOV DWORD PTR DS:[40304C],crackme1.00403000	ASCII "An error occured"
00401069	. E9 DE010000	JMP crackme1.0040124C	
0040106E	. 837D 0C 10	CMP [ARG.2],10	
00401072	. 75 00	JNZ SHORT crackme1.00401081	
00401074	. FF75 08	PUSH [ARG.1]	hWnd = 0040102B
00401077	. E8 32040000	CALL <JMP.&USER32.DestroyWindow>	DestroyWindow
0040107C	. E9 CB010000	JMP crackme1.0040124C	
00401081	. 817D 0C 11010000	CMP [ARG.2],111	
00401088	. 75 00	JNZ SHORT crackme1.00401243	
0040108E	. 8B45 10	MOV EAX,[ARG.3]	
00401091	. 8B55 10	MOV EDX,[ARG.3]	
00401094	. C1EA 10	SHR EDX,10	
00401097	. 66:0BD2	OR DX,DX	
0040109A	. 75 00	JNZ crackme1.0040124C	

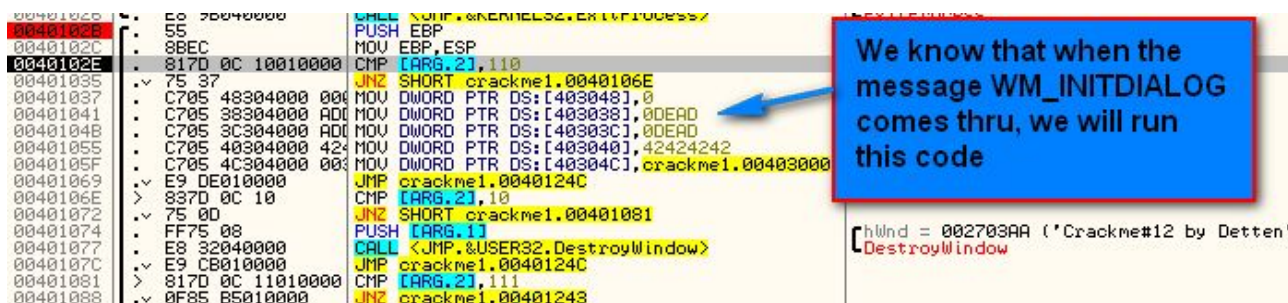
一启动应用，我们立马就断在了那个 **BP**。你会发现现在我们开始进行第一次比较的地方有几条指令

40102E CMP [ARG.2], 110

如果你在本章相关下载中的 **Windows** 消息备忘单中查询 **ID 110** 的话，就会发现 **110** 是 **InitDialog** 的编码：



这个消息给了我们的应用一个机会来初始化一下东西。如果你单步执行，并且消息是 INITDIALOG 的话，我们会直接到 401037 执行相关指令：

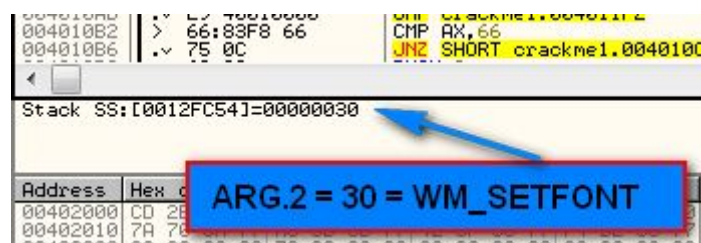


```
0040102B 55 PUSH EBP
0040102C 8BEC MOV EBP,ESP
0040102E 817D 0C 10010000 CMP [ARG.2],110
00401035 75 37 JNZ SHORT crackme1.0040106E
00401037 C705 48304000 000 MOV DWORD PTR DS:[403048],0
00401041 C705 38304000 000 MOV DWORD PTR DS:[403038],0DEAD
00401048 C705 3C304000 000 MOV DWORD PTR DS:[40303C],0DEAD
00401055 C705 40304000 424 MOV DWORD PTR DS:[403040],42424242
0040105F C705 4C304000 000 MOV DWORD PTR DS:[40304C],crackme1.00403000
00401069 E9 DE010000 JMP crackme1.0040124C
0040106E 837D 0C 10 CMP [ARG.2],10
00401072 75 0D JNZ SHORT crackme1.00401081
00401074 FF75 08 PUSH [ARG.1]
00401077 E8 32040000 CALL <JMP.&USER32.DestroyWindow>
0040107C E9 CB010000 JMP crackme1.0040124C
00401081 817D 0C 11010000 CMP [ARG.2],111
00401088 0F85 B5010000 JNZ crackme1.00401243
```

We know that when the message WM_INITDIALOG comes thru, we will run this code

hWnd = 002703AA ('Crackme#12 by Detten')
DestroyWindow

看看下面的信息区，可以看到 ARG.2 不是 110 而是 30：



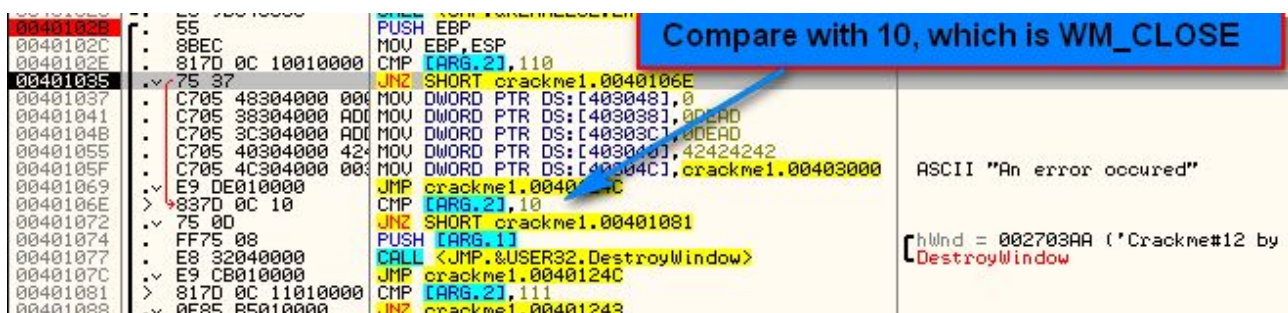
```
004010B2 66:83F8 66 CMP AX,66
004010B6 75 0C JNZ SHORT crackme1.004010C0
```

Stack SS:[0012FC54]=00000030

ARG.2 = 30 = WM_SETFONT

在我们的表中，30 是 set font（设置字体）消息。所以这是 Windows 发送的第一个消息。

下一个是和 10 进行比较，在咱们的消息列表中是 WM-CLOSE:



```
0040102B 55 PUSH EBP
0040102C 8BEC MOV EBP,ESP
0040102E 817D 0C 10010000 CMP [ARG.2],110
00401035 75 37 JNZ SHORT crackme1.0040106E
00401037 C705 48304000 000 MOV DWORD PTR DS:[403048],0
00401041 C705 38304000 000 MOV DWORD PTR DS:[403038],0DEAD
00401048 C705 3C304000 000 MOV DWORD PTR DS:[40303C],0DEAD
00401055 C705 40304000 424 MOV DWORD PTR DS:[403040],42424242
0040105F C705 4C304000 000 MOV DWORD PTR DS:[40304C],crackme1.00403000
00401069 E9 DE010000 JMP crackme1.0040124C
0040106E 837D 0C 10 CMP [ARG.2],10
00401072 75 0D JNZ SHORT crackme1.00401081
00401074 FF75 08 PUSH [ARG.1]
00401077 E8 32040000 CALL <JMP.&USER32.DestroyWindow>
0040107C E9 CB010000 JMP crackme1.0040124C
00401081 817D 0C 11010000 CMP [ARG.2],111
00401088 0F85 B5010000 JNZ crackme1.00401243
```

Compare with 10, which is WM_CLOSE

ASCII "An error occurred"

hWnd = 002703AA ('Crackme#12 by Detten')
DestroyWindow

所以当关闭按钮被点击时，这段代码就会被执行。下一个要比较的是 111，它是 WM-COMMAND:

00401074	FF75 08	PUSH [ARG.1]	
00401077	E8 32040000	CALL <JMP.&USER32.DestroyWindow>	hwnd = 00
0040107C	E9 CB010000	JMP crackme1.0040124C	DestroyWi
00401081	817D 0C 11010000	CMP [ARG.2],111	
00401088	0F85 B5010000	JNZ crackme1.00401243	
0040108E	8B45 10	MOV EAX,[ARG.3]	
00401091	8B55 10	MOV EDX,[ARG.3]	
00401094	C1EA 10	SHR EDX,10	
00401097	66:0BD2	OR DX,DX	
0040109A	0F85 AC010000	JNZ crackme1.0040124C	
004010A0	66:83F8 65	CMP AX,65	
004010A4	75 0C	JNZ SHORT crackme1.004010B2	
004010A6	6A 01	PUSH 1	
004010A8	E8 F8010000	CALL crackme1.004012A5	
004010AD	E9 40010000	JMP crackme1.004011F2	
004010B2	66:83F8 66	CMP AX,66	
004010B6	75 0C	JNZ SHORT crackme1.004010C4	
004010B8	6A 02	PUSH 2	
004010BA	E8 E6010000	CALL crackme1.004012A5	
004010BF	E9 2E010000	JMP crackme1.004011F2	
004010C4	66:83F8 67	CMP AX,67	
004010C8	75 0C	JNZ SHORT crackme1.004010D6	
004010CA	6A 03	PUSH 3	
004010CC	E8 D4010000	CALL crackme1.004012A5	
004010D1	E9 1C010000	JMP crackme1.004011F2	
004010D6	66:83F8 68	CMP AX,68	
004010DA	75 0C	JNZ SHORT crackme1.004010E8	
004010DC	6A 04	PUSH 4	
004010DE	E8 C2010000	CALL crackme1.004012A5	
004010E3	E9 0A010000	JMP crackme1.004011F2	
004010E8	66:83F8 69	CMP AX,69	
004010EC	75 0C	JNZ SHORT crackme1.004010FA	
004010EE	6A 05	PUSH 5	
004010F0	E8 B0010000	CALL crackme1.004012A5	
004010F5	E9 F8000000	JMP crackme1.004011F2	
004010FA	66:83F8 6A	CMP AX,6A	
004010FE	75 0C	JNZ SHORT crackme1.0040110C	

111 = WM_COMMAND

WM_COMMAND 包揽了好几种 Windows 消息，通常与资源相关联，例如点击按钮、选择菜单或点击工具栏中的图标。此外，对于一个 WM_COMMAND 消息来说，在 ARG. 3 中存储了一个整型数据，用来帮助弄清楚命令消息。例如，如果你点击一个按钮，就会传来一个 WM_COMMAND 消息，并且 ARG. 3 中有可能保存有按钮的 ID。如果你正在用一个徒手绘画程序画画，ARG. 3 可能保存有当前鼠标的 X 和 Y 坐标。

00401081	> 817D 0C 11010000	CMP [ARG.2],111
00401088	0F85 B5010000	JNZ crackme1.00401243
0040108E	8B45 10	MOV EAX,[ARG.3]
00401091	8B55 10	MOV EDX,[ARG.3]
00401094	C1EA 10	SHR EDX,10
00401097	66:0BD2	OR DX,DX
0040109A	0F85 AC010000	JNZ crackme1.0040124C
004010A0	66:83F8 65	CMP AX,65
004010A4	75 0C	JNZ SHORT crackme1.004010B2
004010A6	6A 01	PUSH 1
004010A8	E8 F8010000	CALL crackme1.004012A5
004010AD	E9 40010000	JMP crackme1.004011F2
004010B2	66:83F8 66	CMP AX,66
004010B6	75 0C	JNZ SHORT crackme1.004010C4
004010B8	6A 02	PUSH 2
004010BA	E8 E6010000	CALL crackme1.004012A5
004010BF	E9 2E010000	JMP crackme1.004011F2
004010C4	66:83F8 67	CMP AX,67
004010C8	75 0C	JNZ SHORT crackme1.004010D6
004010CA	6A 03	PUSH 3
004010CC	E8 D4010000	CALL crackme1.004012A5
004010D1	E9 1C010000	JMP crackme1.004011F2
004010D6	66:83F8 68	CMP AX,68
004010DA	75 0C	JNZ SHORT crackme1.004010E8
004010DC	6A 04	PUSH 4
004010DE	E8 C2010000	CALL crackme1.004012A5
004010E3	E9 0A010000	JMP crackme1.004011F2
004010E8	66:83F8 69	CMP AX,69
004010EC	75 0C	JNZ SHORT crackme1.004010FA
004010EE	6A 05	PUSH 5
004010F0	E8 B0010000	CALL crackme1.004012A5
004010F5	E9 F8000000	JMP crackme1.004011F2
004010FA	66:83F8 6A	CMP AX,6A
004010FE	75 0C	JNZ SHORT crackme1.0040110C
00401100	6A 06	PUSH 6
00401102	E8 9E010000	CALL crackme1.004012A5
00401107	E9 E6000000	JMP crackme1.004011F2
0040110C	66:83F8 6B	CMP AX,6B

仔细看这个，能够发现过程处理的其他消息只剩 WM_COMMAND 了（真的，每一个 WM_COMMAND 都有可能是一个不同的“类型”）。如果你单步执行，就会发现对于当前的消息 WM_SETFONT，没有与之有关的代码可以执行，

只是在我们的过程的结尾返回了。这是在告知 Windows，我们希望 Windows 来处理这个消息，而不是由我们来处理：

```
00401228 |> 68 11304000 | PUSH crackme1.00403011
0040122D |> 6A 03 | PUSH 3
0040122F |> FF75 08 | PUSH [ARG.1]
00401232 |> E8 89020000 | CALL <JMP.&USER32.SetDlgItemTextA>
00401237 |> C705 44304000 000 | MOV DWORD PTR DS:[403044],0
00401241 |> EB 09 | JMP SHORT crackme1.0040124C
00401243 |> B8 00000000 | MOV EAX,0
00401248 |> C9 | LEAVE
00401249 |> C2 1000 | RETN 10
0040124C |> B8 01000000 | MOV EAX,1
00401251 |> C9 | IFAUF
```

再次点击运行，我们会断在下一个消息：

```
00401026 |> E8 9B040000 | CALL <JMP.&KERNEL32.ExitProcess>
0040102E |> 55 | PUSH EBP
0040102C |> 8BEC | MOV EBP,ESP
0040102E |> 817D 0C 10010000 | CMP [ARG.2],110
00401035 |> 75 37 | JNZ SHORT crackme1.0040106E
00401037 |> C705 48304000 000 | MOV DWORD PTR DS:[403048],0
00401041 |> C705 38304000 000 | MOV DWORD PTR DS:[403038],0DEAD
00401048 |> C705 3C304000 000 | MOV DWORD PTR DS:[40303C],0DEAD
00401055 |> C705 40304000 42 | MOV DWORD PTR DS:[403040],42424242
0040105F |> C705 4C304000 000 | MOV DWORD PTR DS:[40304C],crackme1.00403000
00401069 |> E9 DE010000 | JMP crackme1.0040124C
0040106E |> 837D 0C 10 | CMP [ARG.2],10
00401072 |> 75 0D | JNZ SHORT crackme1.00401081
00401074 |> FF75 08 | PUSH [ARG.1]
00401077 |> E8 32040000 | CALL <JMP.&USER32.DestroyWindow>
0040107C |> E9 CB010000 | JMP crackme1.0040124C
00401081 |> 817D 0C 11010000 | CMP [ARG.2],111
00401088 |> 0F85 B5010000 | JNZ crackme1.00401243
0040108E |> 8B45 10 | MOV EAX,[ARG.3]
00401091 |> 8B55 10 | MOV EDX,[ARG.3]
00401094 |> C1EA 10 | SHR EDX,10
00401097 |> 66 0BD2 | OR DX,DX
00401099 |> 0F85 AC010000 | JNZ crackme1.0040124C
004010A0 |> 66 83F8 65 | CMP AX,65
004010A4 |> 75 0C | JNZ SHORT crackme1.004010B2
004010A6 |> 6A 01 | PUSH 1
004010A8 |> E8 F8010000 | CALL crackme1.004012A5
004010AD |> E9 40010000 | JMP crackme1.004011F2
004010B2 |> 66 83F8 66 | CMP AX,66
004010B6 |> 75 0C | JNZ SHORT crackme1.004010C4
004010B8 |> 6A 02 | PUSH 2
004010BA |> E8 E6010000 | CALL crackme1.004012A5
004010BF |> E9 2E010000 | JMP crackme1.004011F2
004010C4 |> 66 83F8 67 | CMP AX,67
004010C8 |> 75 0C | JNZ SHORT crackme1.004010D6
004010CA |> 6A 03 | PUSH 3
004010CC |> E8 D4010000 | CALL crackme1.004012A5
004010D1 |> E9 1C010000 | JMP crackme1.004011F2
004010D6 |> 66 83F8 68 | CMP AX,68
004010DA |> 75 0C | JNZ SHORT crackme1.004010E8
004010DC |> 6A 04 | PUSH 4
004010DE |> E8 C2010000 | CALL crackme1.004012A5
004010E3 |> E9 0A010000 | JMP crackme1.004011F2
004010E8 |> 66 83F8 69 | CMP AX,69
004010EC |> 75 0C | JNZ SHORT crackme1.004010FA
004010EE |> 6A 05 | PUSH 5
004010F0 |> E8 B0010000 | CALL crackme1.004012A5
004010F5 |> E9 F8000000 | JMP crackme1.004011F2
```

Stack SS:[0012F644]=00000111

这回它是一个 WM-COMMAND 消息。向下单步执行到 401081 检测该消息的比较指令处，咱们再仔细看看 WM-COMMAND 的处理程序：

```
00401074 |> FF75 08 | PUSH [ARG.1]
00401077 |> E8 32040000 | CALL <JMP.&USER32.DestroyWindow>
0040107C |> E9 CB010000 | JMP crackme1.0040124C
00401081 |> 817D 0C 11010000 | CMP [ARG.2],111
00401088 |> 0F85 B5010000 | JNZ crackme1.00401243
0040108E |> 8B45 10 | MOV EAX,[ARG.3]
00401091 |> 8B55 10 | MOV EDX,[ARG.3]
00401094 |> C1EA 10 | SHR EDX,10
00401097 |> 66 0BD2 | OR DX,DX
00401099 |> 0F85 AC010000 | JNZ crackme1.0040124C
004010A0 |> 66 83F8 65 | CMP AX,65
004010A4 |> 75 0C | JNZ SHORT crackme1.004010B2
004010A6 |> 6A 01 | PUSH 1
004010A8 |> E8 F8010000 | CALL crackme1.004012A5
004010AD |> E9 40010000 | JMP crackme1.004011F2
004010B2 |> 66 83F8 66 | CMP AX,66
004010B6 |> 75 0C | JNZ SHORT crackme1.004010C4
004010B8 |> 6A 02 | PUSH 2
004010BA |> E8 E6010000 | CALL crackme1.004012A5
004010BF |> E9 2E010000 | JMP crackme1.004011F2
004010C4 |> 66 83F8 67 | CMP AX,67
004010C8 |> 75 0C | JNZ SHORT crackme1.004010D6
004010CA |> 6A 03 | PUSH 3
004010CC |> E8 D4010000 | CALL crackme1.004012A5
004010D1 |> E9 1C010000 | JMP crackme1.004011F2
004010D6 |> 66 83F8 68 | CMP AX,68
004010DA |> 75 0C | JNZ SHORT crackme1.004010E8
004010DC |> 6A 04 | PUSH 4
004010DE |> E8 C2010000 | CALL crackme1.004012A5
004010E3 |> E9 0A010000 | JMP crackme1.004011F2
004010E8 |> 66 83F8 69 | CMP AX,69
004010EC |> 75 0C | JNZ SHORT crackme1.004010FA
004010EE |> 6A 05 | PUSH 5
004010F0 |> E8 B0010000 | CALL crackme1.004012A5
004010F5 |> E9 F8000000 | JMP crackme1.004011F2
```

注意它将 ARG.3 拷贝到 EAX 和 EDX。然后它对 EDX 完成了 16 位 SHR（右移位）操作。然后对该值做 OR 操作，如果它不是 0，就跳转。基本上这是

在检测参数的两个高位字节是否是 0(你正在读汇编语言的书,不是吗?)。EDX 的低位两字节保存了被影响到的资源 ID。本例中,它不是 0,所以我们跳过剩下的代码,然后从回调函数返回:

```

0040107C |> E9 CB010000 | JMP crackme1.004010B2
00401081 |> 817D 0C 11010000 | CMP [ARG.2],111
00401088 |> 0F85 B5010000 | JNZ crackme1.004010B2
0040108E |> 8B45 10 | MOV EAX,[ARG.3]
00401091 |> 8B55 10 | MOV EDX,[ARG.3]
00401094 |> C1EA 10 | SHR EDX,10
00401097 |> 66:0BD2 | OR DX,DX
0040109A |> 0F85 AC010000 | JNZ crackme1.004010B2
004010A0 |> 66:83F8 65 | CMP AX,65
004010A4 |> 75 0C | JNZ SHORT crackme1.004010B2
004010A6 |> 6A 01 | PUSH 1
004010A8 |> E8 F8010000 | CALL crackme1.004012A5
004010AD |> E9 40010000 | JMP crackme1.004011F2
004010B2 |> 66:83F8 66 | CMP AX,66
004010B6 |> 75 0C | JNZ SHORT crackme1.004010C4
004010B8 |> 6A 02 | PUSH 2

```

这里我们能看到正在处理的是 111, 或者叫 WM_COMMAND 消息:

```

004010D0 |> 66:03F0 00 | CMP AX,0
004010DA |> 75 0C | JNZ SHORT crackme1.004010E0
004010DC |> 6A 04 | PUSH 4

```

Stack SS:[0012F644]=00000111

Address	Hex	dump
00402000	CD 2B 12 76 F3 D8 12 76 E2 BB 13 76 0E	
00402010	7A 70 3A 77 A3 3B 3B 77 42 CF 3C 77 F4	

这里我们能看到那个跳转:

```

00401081 |> 817D 0C 11010000 | CMP [ARG.2],111
00401088 |> 0F85 B5010000 | JNZ crackme1.00401243
0040108E |> 8B45 10 | MOV EAX,[ARG.3]
00401091 |> 8B55 10 | MOV EDX,[ARG.3]
00401094 |> C1EA 10 | SHR EDX,10
00401097 |> 66:0BD2 | OR DX,DX
0040109A |> 0F85 AC010000 | JNZ crackme1.0040124C
004010A0 |> 66:83F8 65 | CMP AX,65
004010A4 |> 75 0C | JNZ SHORT crackme1.004010B2
004010A6 |> 6A 01 | PUSH 1
004010A8 |> E8 F8010000 | CALL crackme1.004012A5
004010AD |> E9 40010000 | JMP crackme1.004011F2
004010B2 |> 66:83F8 66 | CMP AX,66
004010B6 |> 75 0C | JNZ SHORT crackme1.004010C4
004010B8 |> 6A 02 | PUSH 2
004010BA |> E8 E6010000 | CALL crackme1.004012A5
004010BD |> E9 2E010000 | JMP crackme1.004011F2
004010C4 |> 66:83F8 67 | CMP AX,67
004010C8 |> 75 0C | JNZ SHORT crackme1.004010D6
004010CA |> 6A 03 | PUSH 3
004010CC |> E8 D4010000 | CALL crackme1.004012A5
004010D1 |> E9 1C010000 | JMP crackme1.004011F2
004010D6 |> 66:83F8 68 | CMP AX,68
004010DA |> 75 0C | JNZ SHORT crackme1.004010E8
004010DC |> 6A 04 | PUSH 4
004010DE |> E8 C2010000 | CALL crackme1.004012A5
004010E3 |> E9 0A010000 | JMP crackme1.004011F2
004010E8 |> 66:83F8 69 | CMP AX,69
004010EC |> 75 0C | JNZ SHORT crackme1.004010FA
004010EE |> 6A 05 | PUSH 5
004010F0 |> E8 B0010000 | CALL crackme1.004012A5
004010F5 |> E9 F8000000 | JMP crackme1.004011F2
004010FA |> 66:83F8 6A | CMP AX,6A
004010FE |> 75 0C | JNZ SHORT crackme1.0040110C
00401100 |> 6A 06 | PUSH 6
00401103 |> E8 9C010000 | CALL crackme1.004012A5

```

再次运行程序, 我们有停在了我们设置的 BP。这回我们处理的是 WM_INITDIALOG 消息:

```

004010D6 |> 66:83F8 68 | CMP AX,68
004010DA |> 75 0C | JNZ SHORT crackme1.004010E0
004010DC |> 6A 04 | PUSH 4

```

Stack SS:[0012FC54]=00000110

Address	Hex	dump
00402000	CD 2B 12 76 F3 D8 12 76 E2 BB 13 76 0E	

咱们运行对话框初始化部分的前面几行代码:

0040102E	55	PUSH EBP
0040102C	8BEC	MOV EBP,ESP
0040102E	817D 0C 10010000	CMP [ARG_2],110
00401035	75 37	JNZ SHORT crackme1.0040106E
00401037	C705 48304000 00	MOV DWORD PTR DS:[403048],0
00401041	C705 38304000 AD	MOV DWORD PTR DS:[403038],0DEAD
0040104B	C705 3C304000 AD	MOV DWORD PTR DS:[40303C],0DEAD
00401055	C705 40304000 42	MOV DWORD PTR DS:[403040],42424242
0040105F	C705 4C304000 00	MOV DWORD PTR DS:[40304C],crackme1.00403000
00401069	E9 DE010000	JMP crackme1.0040124C
0040106E	837D 0C 10	CMP [ARG_2],10

It's 110 (WM_INITDIALOG) so we run this code

ASCII "An error occurred"

在这个特殊的 crackme 中，刚好这部分代码比较重要。咱们看到有几个整型被存进以 403038 为起始位置的内存中（颠倒顺序进行访问，403038 是最低位）。咱们先在数据窗口中看看：

Address	Hex dump	ASCII
00403038	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403048	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403058	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403068	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403078	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403088	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403098	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
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

看，在咱们运行这几行前，它被初始化为 0。现在，单步步过第一个 MOV 指令，什么也没发生，不过一个 0 被拷贝到 403038 处。单步步过下一条指令，能够看到产生的效果：

00401037	C705 48304000 00	MOV DWORD PTR DS:[403048],0
00401041	C705 38304000 AD	MOV DWORD PTR DS:[403038],0DEAD
0040104B	C705 3C304000 AD	MOV DWORD PTR DS:[40303C],0DEAD
00401055	C705 40304000 42	MOV DWORD PTR DS:[403040],42424242
0040105F	C705 4C304000 00	MOV DWORD PTR DS:[40304C],crackme1.00403000

可以看到 0x0DEAD 被拷贝到内存中（以小端序列的形式）：

Address	Hex dump
00403038	AD DE 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403048	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403058	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403068	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403078	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403088	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403098	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
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

单步步过下一行，它做了同样的事情，不过是在地址 40303C 处：

Address	Hex dump
00403038	AD DE 00 00 AD DE 00 00 00 00 00 00 00 00 00 00
00403048	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403058	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403068	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403078	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403088	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403098	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
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

事实是，以十六进制的双字写入彻底的暴露它对该 crackme 是很重要的 😊（译者注：为啥？）。接下来，在 403040 处值 42 被写入了 4 次。在 ASCII 数据区可以看到 42 的 ASCII 值是 “B”：

Address	Hex dump	ASCII
00403038	AD DE 00 00 AD DE 00 00 42 42 42 42 00 00 00 00	↓...↓..BBBB....
00403048	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403058	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403068	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403078	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403088	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403098	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
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

最后，整数 403000 被拷贝到 40304C。Oilly 可以分辨出是一个指向以 403000 为起始位置的一段代码或数据（记住是小端序列）：

Address	Hex dump	ASCII
00403038	AD DE 00 00 AD DE 00 00 42 42 42 42 00 00 00 00	↓...↓..BBBB....
00403048	00 00 00 00 00 30 40 00 00 00 00 00 00 00 00 000e.....
00403058	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403068	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403078	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403088	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00403098	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
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

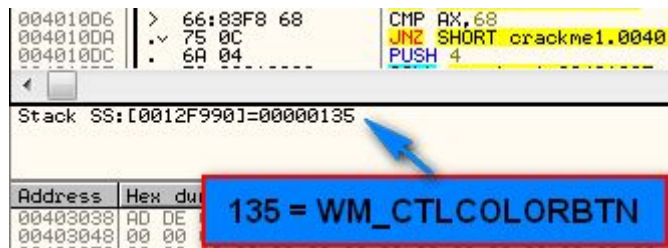
最终我们跳到了结尾并返回，等待发过来的下一个消息：

0040102C	. 8BEC	MOV EBP,ESP
0040102E	. 817D 0C 10010000	CMP [ARG_2],110
00401035	> 75 37	JNZ SHORT crackme1.0040106E
00401037	. C705 48304000 000	MOV DWORD PTR DS:[403048],0
00401041	. C705 38304000 ADD	MOV DWORD PTR DS:[403038],0DEAD
00401048	. C705 3C304000 ADD	MOV DWORD PTR DS:[40303C],0DEAD
00401055	. C705 40304000 424	MOV DWORD PTR DS:[403040],42424242
0040105F	. C705 4C304000 003	MOV DWORD PTR DS:[40304C],crackme1.00403000
00401069	> E9 DE010000	JMP crackme1.0040124C
0040106E	> 837D 0C 10	CMP [ARG_2],10
00401072	> 75 0D	JNZ SHORT crackme1.00401081
00401074	. FF75 08	PUSH [ARG_1]
00401077	. E8 32040000	CALL <JMP.&USER32.DestroyWindow>
0040107C	> E9 CB010000	JMP crackme1.0040124C
00401081	> 817D 0C 11010000	CMP [ARG_2],111
00401088	> 0F85 B5010000	JNZ crackme1.00401243
0040108E	. 8B45 10	MOV EAX,[ARG_3]
00401091	. 8B55 10	MOV EDX,[ARG_3]
00401094	. C1EA 10	SHR EDX,10
00401097	. 66:0BD2	OR DX,DX
0040109A	> 0F85 AC010000	JNZ crackme1.0040124C
004010A0	. 66:83F8 65	CMP AX,65
004010A4	> 75 0C	JNZ SHORT crackme1.004010B2
004010A6	. 6A 01	PUSH 1
004010A8	. E8 F8010000	CALL crackme1.004012A5
004010AD	> E9 40010000	JMP crackme1.004011F2
004010B2	> 66:83F8 66	CMP AX,66
004010B6	> 75 0C	JNZ SHORT crackme1.004010C4
004010B8	. 6A 02	PUSH 2
004010BA	. E8 E6010000	CALL crackme1.004012A5
004010BF	> E9 2F010000	JMP crackme1.004011F2

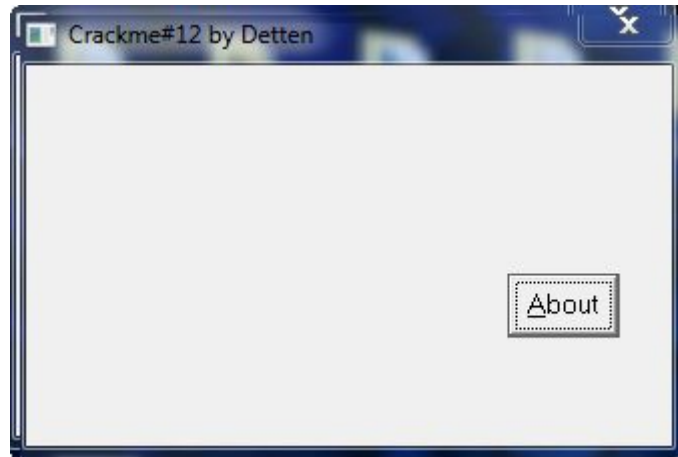
多点 F9 几次（10）你会看到主对话框窗口被创建出来：



这地方非常的有意思，因为在你点击 F9 时，每点一次在对话框中就会有新的东西出现（大概运行 6 次以后），就会接收到一个消息来在对话框中画资源。下一个消息是 135，或者叫 WM_CTLCOLORBUTTON:



在窗口中画了一个按钮：



下一个是一个写着“2”的按钮：



这时候按 **F9**，你会真切的看到对话框被构建，一次一个按钮。观察到来的所有消息以及在表中查询它们是相当的有趣。你会看到有很多消息到来。如果有那个你不知道的，就 **Google** 搜索它，然后就会得到关于它的相关描述。直至最后，底部的 **label** 控件会被绘制出来，“**No access**” 文本会被写入进去。整个窗口就快完成了。在窗口彻底完成前，我还得按 **F9** 大概 **35** 次：



那么现在你知道了一个对话框是如何构建的。设置对话框的基本设置，标题以及整体外观，传进一个回调函数的指针（地址），用来处理所有从 **Windows** 发送过来的消息。然后 **Windows** 发送一系列的消息，一次一个的发给回调函数，给我们机会让我们在我们收到所渴望的消息时运行代码。在对话框被构建完成后，**Windows** 就进入一个内部循环，就坐在那等我们干些什么。只要我们一有动作，一个带有已经发生的动作的 **ID** 的消息就会被发送给回调函数。然后我们就可以决定对该消息做些什么或者忽略它，让 **Windows** 来处理它。

你需要注意的最后一件事是，如果该应用是在 **Ollly** 中运行的，只要在窗口上移动鼠标就会导致 **Ollly** 暂停在处理新消息的消息处理过程的开始处。**Windows** 告诉我们的消息处理过程有鼠标在窗口上移过。基本上，你对对话框做的任何动作都会发送消息给消息处理过程。

四、作业

1、你能不能找出点击一个按钮后会发生什么，尤其是以 **403038** 为起始位置的内存内容方面。不同的按钮做的不一样吗？你能够理解代码正在修改这些内存位置了吗？

2、猜一猜密码有多长。