

第九章：无相关字符串

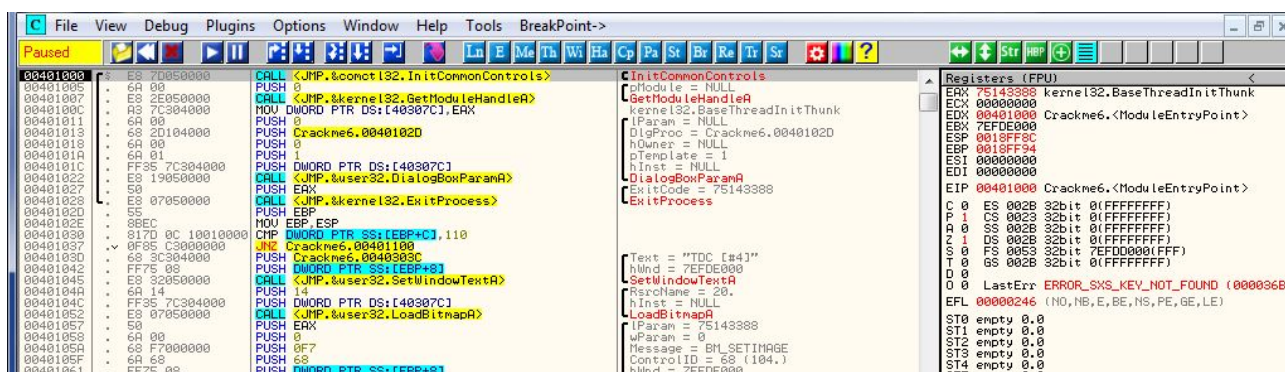
一、简介

此次教程中我将会向我们的武器库中加入一个新的装备。如果搜索二进制文件时发现没有可用的字符串你怎么办？我将会介绍一个新的 R.E.T.A.R.D. 规则😁。此次教程（下一章也是）我们将研究“TDC”写的一个 crackme 叫 Crackme6，相关下载里面包含有。总之，它不是一个硬骨头，不过我将会对其进行一些高级分析，好为将来的教程做准备。

你可以在[教程](#)页下载相关文件以及本教程的 PDF 版本。

那么，咱们开始吧.....

01ly 载入 Crackme6: (p1)



现在，我们已经知道操作程序了。运行程序看看情况：(p2)



嗯，看起来挺简单的。我输入了一个密码 1212121212，下面就是返回的情况：(p3)



The screenshot shows the Immunity Debugger interface. On the left, the CPU registers window displays values for EAX, ECX, EDI, and ESI. The main window shows a disassembled instruction: `CALL EBX`. The right sidebar shows the 'Registers' window with values for EAX, ECX, EDI, and ESI. The 'Search for' menu is open, showing options like 'Name (label) in current module', 'Name in all modules', 'Command', 'Sequence of commands', 'Constant', 'Binary string', 'All intermodular calls', 'All commands', 'All sequences', 'All constants', 'All switches', 'All referenced text strings' (highlighted with a red arrow), 'User-defined label', and 'User-defined comment'.

Address	Disassembly	Text string
00401000	CALL <JMP.&.comctl32.InitCommonCon	(Initial CPU selection)
00401030	PUSH Crackme6.0040303C	ASCII "TDC [#4]"
004011F7	PUSH Crackme6.00403000	ASCII "NLLJ\\\\\\KJAFJK."
00401203	PUSH Crackme6.0040300F	ASCII "NLLJ\\\\\\HJNA[JK."
0040122D	PUSH Crackme6.0040301F	ASCII "M}z{ji'}lfah0/Mnk/xnv."
00401237	PUSH Crackme6.0040301F	ASCII "M}z{ji'}lfah0/Mnk/xnv."
00401249	PUSH Crackme6.0040301F	ASCII "M}z{ji'}lfah0/Mnk/xnv."
0040126E	PUSH Crackme6.00403000	ASCII "NLLJ\\\\\\KJAFJK."
00401282	PUSH Crackme6.00403000	ASCII "NLLJ\\\\\\KJAFJK."
004012A1	PUSH Crackme6.0040300F	ASCII "NLLJ\\\\\\HJNA[JK."
004012C0	PUSH Crackme6.00403000	ASCII "NLLJ\\\\\\KJAFJK."
004012D2	PUSH Crackme6.00403000	ASCII "NLLJ\\\\\\KJAFJK."
004012DE	PUSH Crackme6.0040300F	ASCII "NLLJ\\\\\\HJNA[JK."
0040131E	PUSH Crackme6.00403045	ASCII "About"

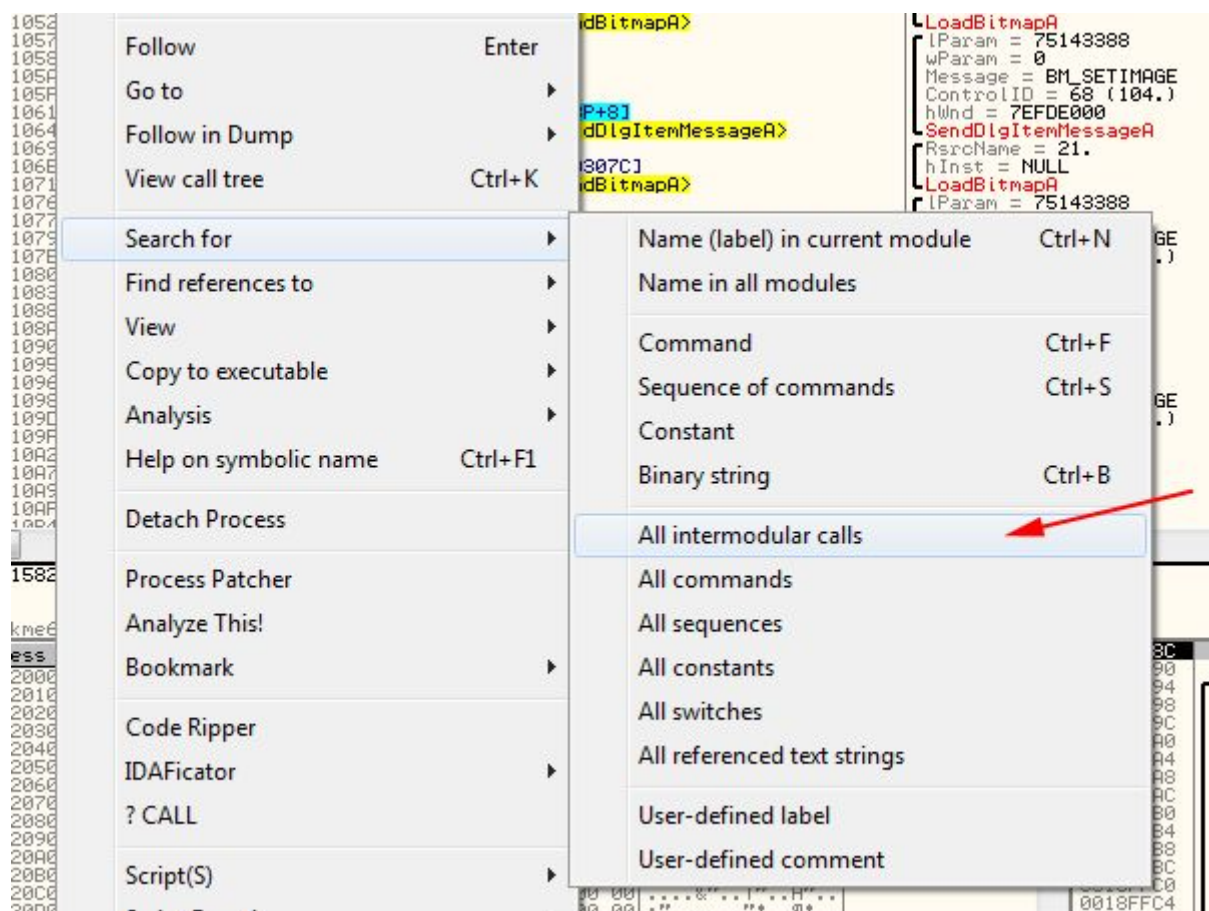
#3.不要依赖二进制文件当前已有的字符串。

不幸的是，在你开始研究真正的二进制文件（比如商业产品）时，它们中的大部分被以某种方式打包 以及/或 保护。干扰逆向工程师的一个最明显的方法是加密字符串。坦率地说，在逆向工程领域当我第一次研究一个感兴趣的新的二进制文件时，如果我搜索字符串并且搜出来了，我能够假定那个二进制文件很可能没有多少挑战。所以，你不能够依赖于那些东西（如果有当然更好😄）。

二、模块间的调用

有鉴于此，我向你展示一个新的在没有字符串的情况下的技巧。大部分的 Windows 应用程序使用一个标准的 API 集来完成特定的动作。例如，如果需要一个简单的消息框的话就调用 `MessageBoxA`，当程序想要退出的时候就调用 `TerminateProcess`。因为大部分的应用都使用这些相同的 API，我们可以用这个获利。例如，有些 API 可以用于从对话框的输入框（类似用户名和序列号）获取文本。有可以被调用用来比较两个字符串的字符串比较函数（输入的密码和程序中存储的密码相同吗？）。有读写注册表的 API（存储和读取你的注册状态）。

Ollly 提供了一种搜索所有被调用的 API 的方法。在反汇编窗口右键，选择 “Search for” -> “All intermodular calls”: (p6)



Ollly 会弹出 Found intermodular calls 窗口: (p7)

R Found intermodular calls		
Address	Disassembly	Destination
00401000	CALL <JMP.&comctl32.InitCommonCon	(Initial CPU selection)
00401007	CALL <JMP.&kernel32.GetModuleHand	kernel32.GetModuleHandleA
00401022	CALL <JMP.&user32.ShowDialogParamA	user32.ShowDialogParamA
00401028	CALL <JMP.&kernel32.ExitProcess>	kernel32.ExitProcess
00401045	CALL <JMP.&user32.SetWindowTextA>	user32.SetWindowTextA
00401052	CALL <JMP.&user32.LoadBitmapA>	user32.LoadBitmapA
00401064	CALL <JMP.&user32.SendDlgItemMess	user32.SendDlgItemMessageA
00401071	CALL <JMP.&user32.LoadBitmapA>	user32.LoadBitmapA
00401083	CALL <JMP.&user32.SendDlgItemMess	user32.SendDlgItemMessageA
00401090	CALL <JMP.&user32.LoadBitmapA>	user32.LoadBitmapA
004010A2	CALL <JMP.&user32.SendDlgItemMess	user32.SendDlgItemMessageA
004010AF	CALL <JMP.&user32.LoadIconA>	user32.LoadIconA
004010BF	CALL <JMP.&user32.SendMessageA>	user32.SendMessageA
004010C9	CALL <JMP.&user32.GetDlgItem>	user32.GetDlgItem
004010EA	CALL <JMP.&user32.SetWindowTextA>	user32.SetWindowTextA
0040110E	CALL <JMP.&gdi32.CreateSolidBrush	gdi32.CreateSolidBrush
0040112D	CALL <JMP.&gdi32.SetTextColor>	gdi32.SetTextColor
00401137	CALL <JMP.&gdi32.SetBkMode>	gdi32.SetBkMode
00401141	CALL <JMP.&gdi32.CreateSolidBrush	gdi32.CreateSolidBrush
0040114A	CALL <JMP.&gdi32.SetBkColor>	gdi32.SetBkColor
00401154	CALL <JMP.&gdi32.CreateSolidBrush	gdi32.CreateSolidBrush
00401173	CALL <JMP.&gdi32.SetTextColor>	gdi32.SetTextColor
00401180	CALL <JMP.&gdi32.SetBkColor>	gdi32.SetBkColor
0040118A	CALL <JMP.&gdi32.CreateSolidBrush	gdi32.CreateSolidBrush
004011A1	CALL <JMP.&user32.ReleaseCapture>	user32.ReleaseCapture
004011B2	CALL <JMP.&user32.SendMessageA>	user32.SendMessageA
004011E1	CALL <JMP.&user32.ShowDialogParamA	user32.ShowDialogParamA
00401242	CALL <JMP.&user32.SetWindowTextA>	user32.SetWindowTextA
00401264	CALL <JMP.&user32.GetDlgItemTextA	user32.GetDlgItemTextA
00401279	CALL <JMP.&user32.SetWindowTextA>	user32.SetWindowTextA
0040128D	CALL <JMP.&user32.SetWindowTextA>	user32.SetWindowTextA
004012A0	CALL <JMP.&user32.SetWindowTextA>	user32.SetWindowTextA
004012B9	CALL <JMP.&user32.EnableWindow>	user32.EnableWindow
004012C8	CALL <JMP.&user32.SetWindowTextA>	user32.SetWindowTextA
004012F5	CALL <JMP.&user32.EndDialog>	user32.EndDialog
00401307	CALL <JMP.&user32.EndDialog>	user32.EndDialog
00401326	CALL <JMP.&user32.SetWindowTextA>	user32.SetWindowTextA
00401333	CALL <JMP.&user32.LoadBitmapA>	user32.LoadBitmapA
00401345	CALL <JMP.&user32.SendDlgItemMess	user32.SendDlgItemMessageA
0040135D	CALL <JMP.&gdi32.CreateSolidBrush	gdi32.CreateSolidBrush
0040137C	CALL <JMP.&gdi32.SetTextColor>	gdi32.SetTextColor
00401386	CALL <JMP.&gdi32.SetBkMode>	gdi32.SetBkMode
00401390	CALL <JMP.&gdi32.CreateSolidBrush	gdi32.CreateSolidBrush
00401399	CALL <JMP.&gdi32.SetBkColor>	gdi32.SetBkColor
004013A3	CALL <JMP.&gdi32.CreateSolidBrush	gdi32.CreateSolidBrush
004013BF	CALL <JMP.&gdi32.SetTextColor>	gdi32.SetTextColor
004013CC	CALL <JMP.&gdi32.SetBkColor>	gdi32.SetBkColor
004013D6	CALL <JMP.&gdi32.CreateSolidBrush	gdi32.CreateSolidBrush
004013EA	CALL <JMP.&user32.ReleaseCapture>	user32.ReleaseCapture
004013FB	CALL <JMP.&user32.SendMessageA>	user32.SendMessageA
00401416	CALL <JMP.&user32.EndDialog>	user32.EndDialog

通常我做的第一件事是点一下“Destination”，将列出的函数按字母顺序进行排序（而不是按地址排序）：(p8)

R Found intermodular calls		
Address	Disassembly	Destination
00401000	CALL <JMP.&comctl32.InitCommonCon	(Initial CPU selection)
00401007	CALL <JMP.&kernel32.GetModuleHand	kernel32.GetModuleHandleA

现在，如果你看第三列的话，你可以看到该 crackme 调用的所有 API：(p9)

R 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.DialogBoxParamA>	user32.DialogBoxParamA
004011E1	CALL <JMP.&user32.DialogBoxParamA>	user32.DialogBoxParamA
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

这是一个小程序，所以调用的函数不是那么多。大部分的程序都有数百个。不过通过这个列表，你可以了解到一个二进制文件的很多信息。你可以发现它用一个对话框作为主窗口。它载入了一个自定义的位图。它修改了对话框中的一些颜色。

在更大些的应用中，这个窗口的价值更高，因为它能告诉你这些事情：1) 是否有注册表相关 API 被调用用来存储和获取信息？是不是有 API 呼叫网站来验证我们确实注册了？3) 有没有读写一个可能存有注册码文件的 API？当我们研究一个加壳的二进制文件时，这个窗口将更加重要（这个后面讨论😄）。

尽管如此，有几个特定 API 逆向工程师总是会留意，因为这几个在保护机制中用的比较多。包括：

DialogBoxParamA
 GetDlgItem
 GetDlgItemInt
 GetDlgItemTextA

GetWindowTextA
GetWindowWord

LoadStringA
lstrcmpA

wsprintfA

MessageBeep
MessageBoxA
MessageBoxExA
SendMessageA
SendDlgItemMessageA

ReadFile
WriteFile
CreateFileA

GetPrivateProfileIntA
WritePrivateProfileStringA
GetPrivateProfileStringA

不幸的是，这里没有包括你可能遇到的所有 API，不过幸运的是，大部分应用使用下面的其中一个：

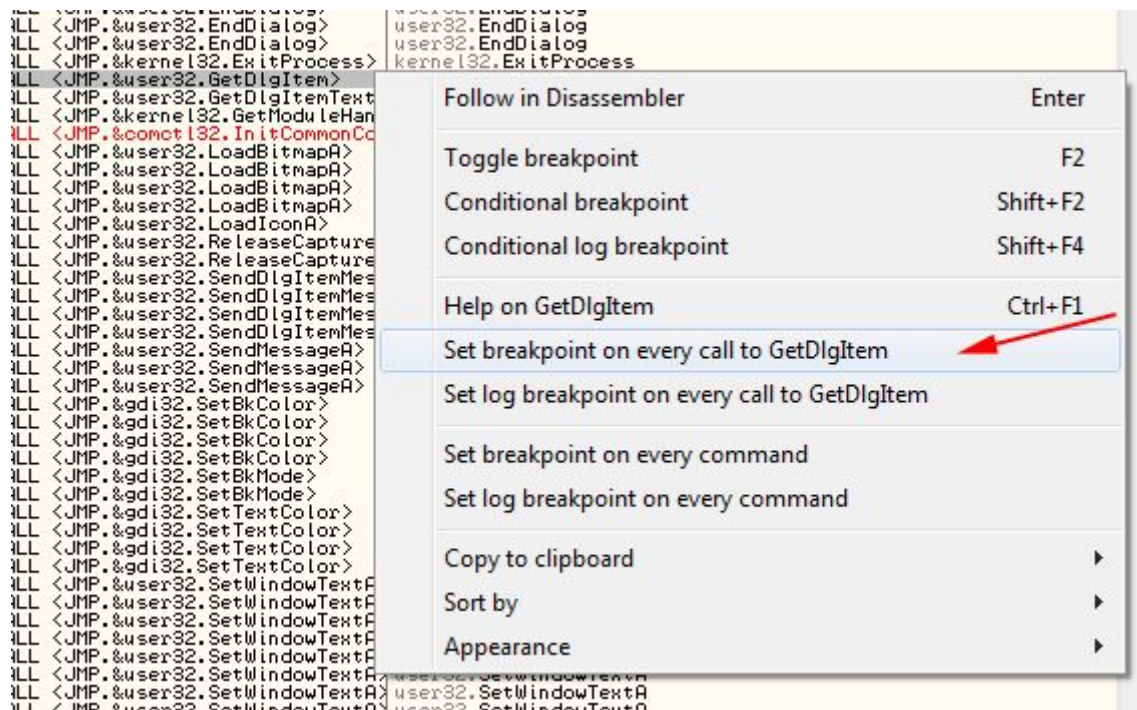
GetDlgItemTextA
GetWindowTextA
lstrcmpA
GetPrivateProfileStringA
GetPrivateProfileIntA
RegQueryValueExA
WritePrivateProfileStringA
GetPrivateProfileIntA

如果你专注这 8 个 API 的调用，你就可以处理绝大多数的实例。还有别忘了，“Get help on symbolic name” 是可以给你提供帮助的。

现在，在 011y 查找出的 crackme 的调用列表中往下看，在那个简短的列表中有两个 API：

GetDlgItem 和 **GetDlgItemTextA**

这两个函数是用来获取输入到对话框中文本框的文本。好吧，在我们的教程中，这只可能说明一件事，获取我们输入的密码。我们想要做的是，不管什么时候只要 011y 遇到两者中的一个就暂停。方法是，选中你想要关注的 API 那行，右键然后选择“Set breakpoint on every call to ----”，这里的----是 API 的名称（这里是 GetDlgItem）：(p10)



现在，我们看到 011y 已经在该行设置了一个 BP：(p11)

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.DialogBoxParamA>	user32.DialogBoxParamA
004011E1	CALL <JMP.&user32.DialogBoxParamA>	user32.DialogBoxParamA
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

我们也在另一个 API GetDlgItemTextA 那暂停，那么点击选中它，右键然后和前面一样进行操作：(p12)

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.DialogBoxParamA>	user32.DialogBoxParamA
004011E1	CALL <JMP.&user32.DialogBoxParamA>	user32.DialogBoxParamA
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
00401254	CALL <JMP.&user32.GetDlgItemTextA>	user32.GetDlgItemTextA
00401007	CALL <JMP.&kernel32.GetModuleHandleA>	kernel32.GetModuleHandleA
00401000	CALL <JMP.&comctl32.InitCommonCommCtrl>	(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.SendDlgItemMessageA>	user32.SendDlgItemMessageA
00401083	CALL <JMP.&user32.SendDlgItemMessageA>	user32.SendDlgItemMessageA
004010A2	CALL <JMP.&user32.SendDlgItemMessageA>	user32.SendDlgItemMessageA
00401345	CALL <JMP.&user32.SendDlgItemMessageA>	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

现在，不管什么时候只要 01ly 遇到了对这两个 API 的调用，它都会断下来（在调用被执行前）。咱们来试试看。重启 crackme 并运行。01ly 会断在对 GetDlgItem 的调用处：(p13)

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 [#41',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\x7FzC/\x7Fojn!j"
004010EA	E8 8D040000	CALL <JMP.&user32.SetWindowTextA>	hWnd = NULL
004010EF	6A 12	PUSH 12	SetWindowTextA
004010F1	68 69304000	PUSH Crackme6.00403069	

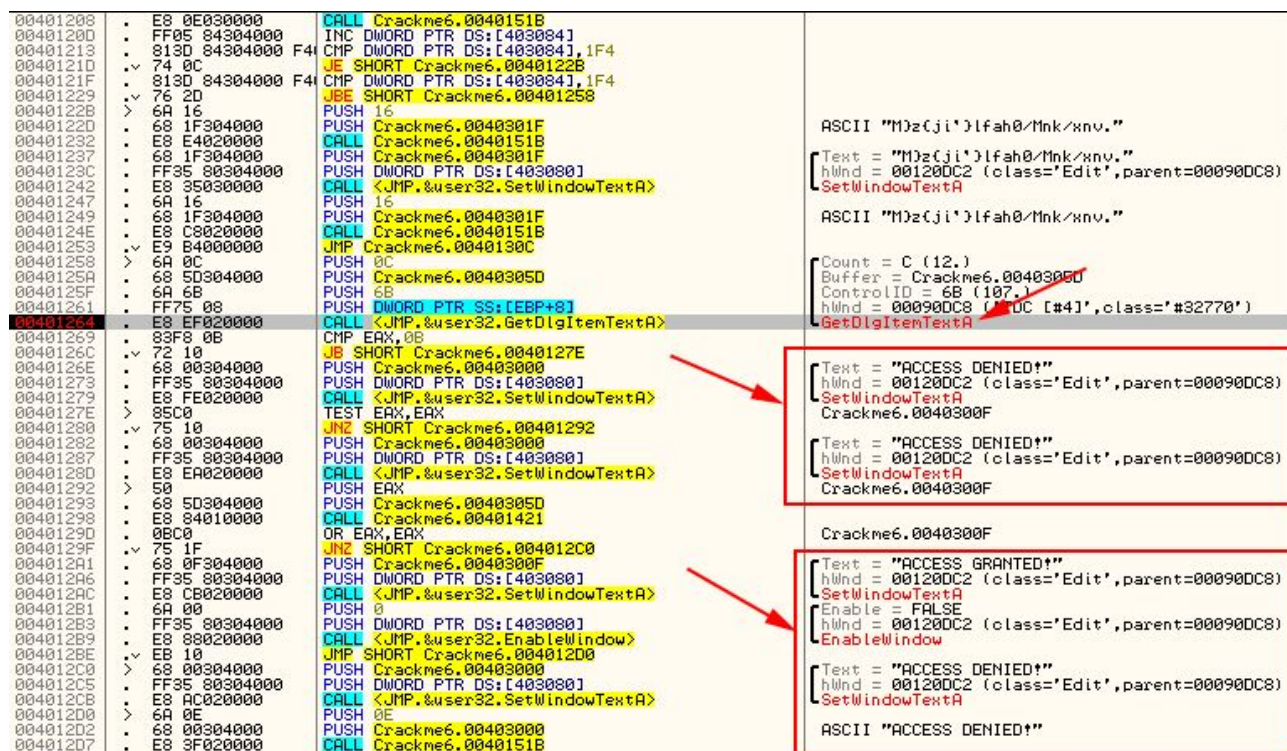
现在，因为我们还没有输入任何内容，我们对 GetDlgItem 取到了什么东西不感兴趣，好咱们继续 (F9)：(p14)



现在输入一个密码，然后点“check”：(p15)



01ly 会再次断下来，这次是在 GetDlgItemTextA: (p16)



如果你看看周围，你会注意到我们已经来到正确的位置😄。搞笑的是，我们起初搜索字符串的时候，没有一个是这些字符串中的😄。

三、破解应用

咱们快速浏览下附近的...。我们注意到有一个跳转 (JB) 跳过了第一个“ACCESS DENIED”，所以我们关注一下它：(p17)

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	
00401282	68 00304000	PUSH Crackme6.00403000	Text = "ACCESS DENIED!"

有一个跳转 (JNZ) 跳过了第二个坏消息，所以我们也将其加入关注名单。然后我们会直接穿过到达好消息，所以基本上我们想要确保我们跳过了这两个跳转：(p18)

00401250	6A 6B	PUSH byte Crackme6.0040305D	ControlID = 6B (107.)
0040125A	6A 6B	PUSH byte Crackme6.0040305D	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	
00401282	68 00304000	PUSH Crackme6.00403000	Text = "ACCESS DENIED!"
00401287	FF35 80304000	PUSH DWORD PTR DS:[403080]	hWnd = 00120DC2 (class='Edit',parent=00090DC8)
0040128D	E8 EA020000	CALL <JMP.&user32.SetWindowTextA>	SetWindowTextA
00401292	50	PUSH EAX	Crackme6.0040300F
00401293	68 5D304000	PUSH Crackme6.0040305D	
00401298	E8 84010000	CALL Crackme6.00401421	
0040129D	0BC0	OR EAX,EAX	Crackme6.0040300F
0040129F	75 1F	JNZ SHORT Crackme6.004012C0	
004012A1	68 0F304000	PUSH Crackme6.0040300F	Text = "ACCESS GRANTED!"
004012A6	FF35 80304000	PUSH DWORD PTR DS:[403080]	hWnd = 00120DC2 (class='Edit',parent=00090DC8)

咱们试试，看看咱们是不是对的。再一次运行程序，我们应该断在 GetDlgItemTextA 指令处 (记住绕过第一个断点)：(p19)

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 = 00120DB4 (class='Edit',parent=00090DC8)
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=00090DC8)
00401287	FF35 80304000	PUSH DWORD PTR DS:[403080]	SetWindowTextA
0040128D	E8 EA020000	CALL <JMP.&user32.SetWindowTextA>	

因为这是一个 JB 跳转，所以我们需要翻转进位标志位：(p20)

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

这样就会强制跳转。现在我们将做另一个 TEST，停在了 401280 处的跳转那。注意，我们的密码已经出现了注释列 😊：(p21)

0040125F	6A 6B	PUSH byte Crackme6.0040305D	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 = 00120DB4 (class='Edit',parent=00090DC8)
00401279	E8 FE020000	CALL <JMP.&user32.SetWindowTextA>	SetWindowTextA
0040127E	85C0	TEST EAX,EAX	
00401280	75 10	JNZ SHORT Crackme6.00401292	ASCII "12121212121" ←
00401282	68 00304000	PUSH Crackme6.00403000	
00401287	FF35 80304000	PUSH DWORD PTR DS:[403080]	
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	
004012A1	68 0F304000	PUSH Crackme6.0040300F	Text = "ACCESS GRANTED!"
004012A6	FF35 80304000	PUSH DWORD PTR DS:[403080]	hWnd = 00120DB4 (class='Edit',parent=00090DC8)

我们想要那个跳转实现，因为它跳过了第二个坏消息，所以我们只需要继续单步直到到达 40129F 的 JNZ 指令：(p22)

0040125F	6A 6B	PUSH 6B	ControlID = 6B (107.)
00401261	FF75 09	PUSH DWORD PTR SS:[EBP+8]	hwnd = 000A0DC8 ('TDC [#4]',class='#32770')
00401264	E8 EF020000	CALL <JMP.&user32.GetDlgItemTextA>	GetDlgItemTextA
00401269	83F8 0B	CMF 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)
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	78 1F	JNZ SHORT Crackme6.004012C0	
004012A1	68 0F304000	PUSH Crackme6.0040300F	Text = "ACCESS GRANTED!"
004012A6	FF35 80304000	PUSH DWORD PTR DS:[403080]	hwnd = 00120DB4 (class='Edit',parent=000A0DC8)
004012AC	E8 CB020000	CALL <JMP.&user32.SetWindowTextA>	SetWindowTextA
004012B1	6A 00	PUSH 0	Enable = FALSE
004012B3	FF35 80304000	PUSH DWORD PTR DS:[403080]	hwnd = 00120DB4 (class='Edit',parent=000A0DC8)
004012B9	E8 88020000	CALL <JMP.&user32.EnableWindow>	EnableWindow
004012BE	EB 10	JMP SHORT Crackme6.004012D0	
004012C0	68 00304000	PUSH Crackme6.00403000	Text = "ACCESS DENIED!"
004012C5	FF35 80304000	PUSH DWORD PTR DS:[403080]	hwnd = 00120DB4 (class='Edit',parent=000A0DC8)
004012CB	E8 AC020000	CALL <JMP.&user32.SetWindowTextA>	SetWindowTextA
004012D0	6A 0E	PUSH 0E	ASCII "ACCESS DENIED!"
004012D2	68 00304000	PUSH Crackme6.00403000	
004012D7	E8 3F020000	CALL Crackme6.0040151B	

好，这条指令将会跳过我们的好消息，所以我们想要阻止它跳。你知道该怎么做：(p23)

C	0	ES	E
P	0	CS	E
A	0	SS	E
Z	1	DS	E
S	0	FS	E
T	0	GS	E

现在运行程序 (F9)，看看我们已经成功的破解了程序：(p24)



四、家庭作业

作为一个挑战，试着给这个 crackme 打补丁，基于那个我们已经修改的标志位。在将打过补丁的程序保存后，你应该可以运行它，输入任何密码（少于 11 个数字）它都会提示 “Access Granted”。记住有几个补丁可以完成这个任务，所以如果一个不起作用，那就找下一个。

加分题： 给 crackme 打补丁，让它接受任意长的密码。