R4ndom's Tutorial #11: Breaking In Our Noob Skills

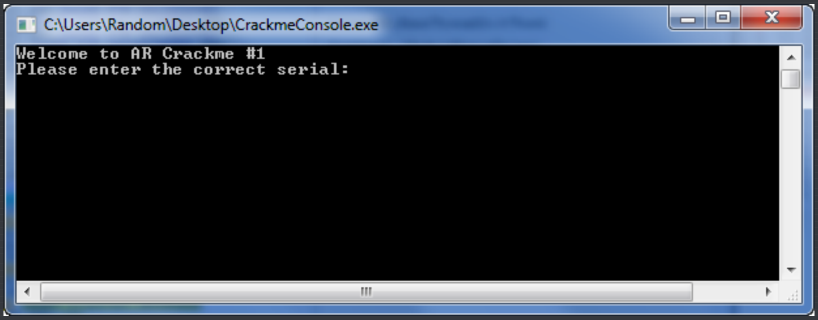
by R4ndom on Jul.05, 2012, under Beginner, Reverse Engineering, Tutorials

Introduction

In this tutorial we will be discussing patching programs again, but diving a little deeper than a typical single "first patch we come to". We will start with a console program and find the correct password that has been hidden in it. It is included in the tutorial download. Other than this, all you will need is OllyDBG.

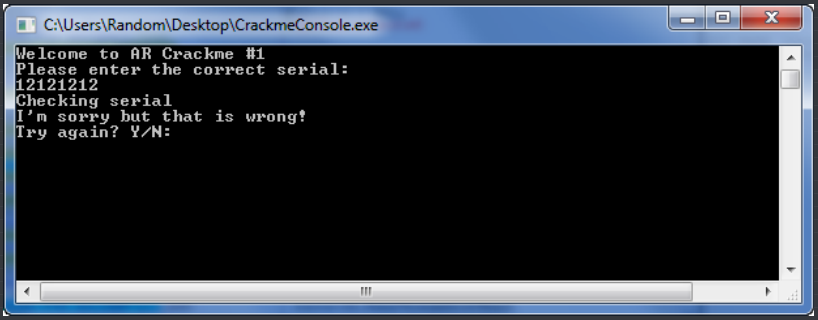
So let's get started…

Console programs are 32bit windows just like any other 32-bit program running under windows. The only difference is they don't use a graphical interface. Other than that, they are identical. This crackme is called CrackmeConsole.exe. Let's run it and see what we got:



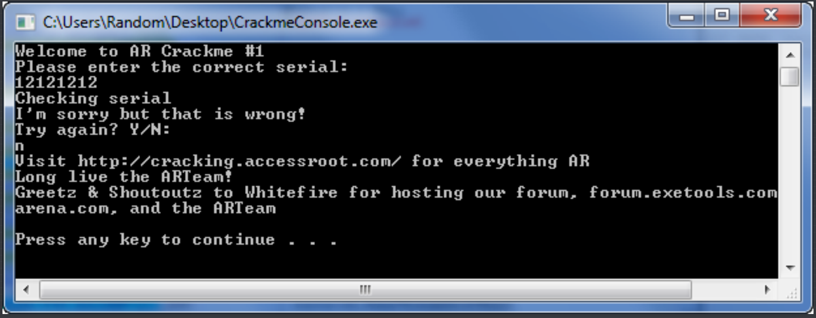
```
Welcome to AR Crackme #1
Please enter the correct serial:
```

Well, looks easy enough. Let's try a password:



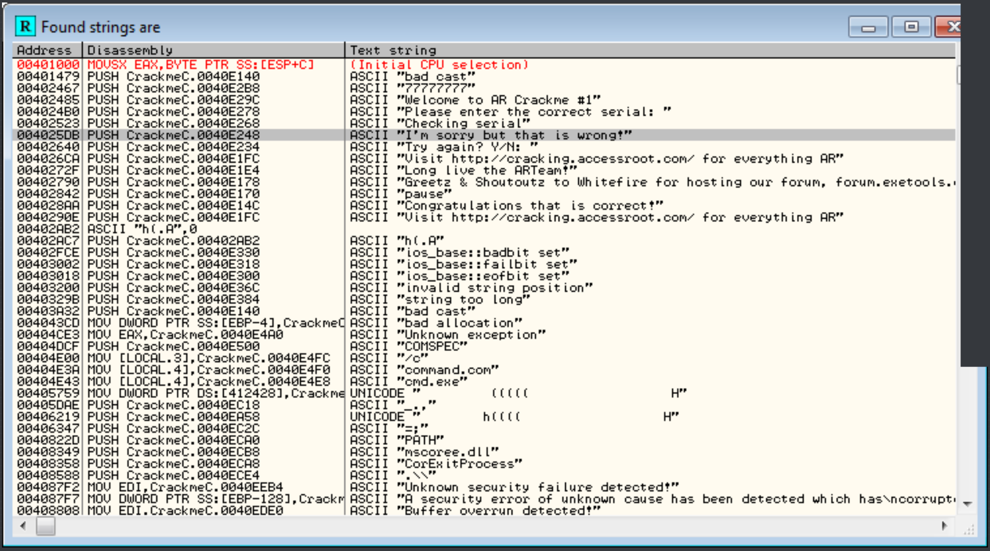
```
Welcome to AR Crackme #1
Please enter the correct serial:
12121212
Checking serial
I'm sorry but that is wrong!
Try again? Y/N:
```

Bummer. Pressing 'N' ends the app:



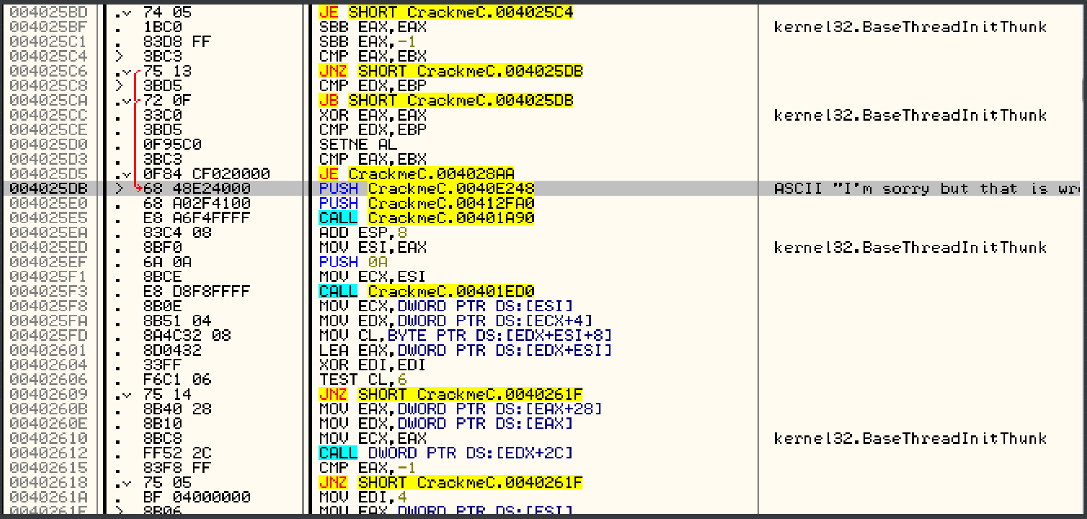
```
Welcome to AR Crackme #1
Please enter the correct serial:
12121212
Checking serial
I'm sorry but that is wrong!
Try again? Y/N:
n
Visit http://cracking.accessroot.com/ for everything AR
Long live the ARTeam!
Greetz & Shoutoutz to Whitefire for hosting our forum, forum.exetools.com
arena.com, and the ARTeam

Press any key to continue . . .
```

Well, I think we have enough to at least start investigating. Go ahead and load it in Olly. Let's then start by searching for strings:



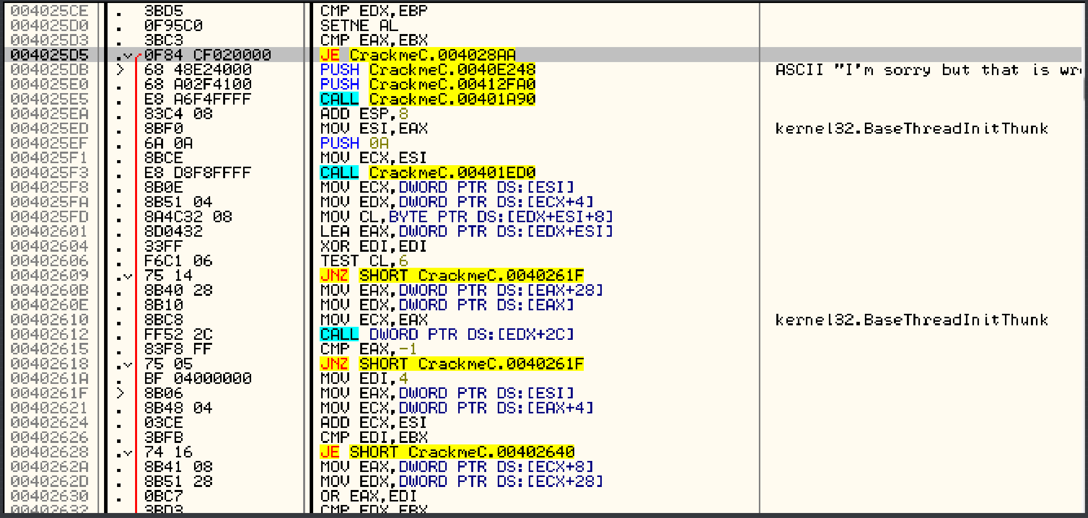
```
Address |Disassembly                          |Text string
00401000 MOVSX EAX,BYTE PTR SS:[ESP+C]         (Initial CPU selection)
00401479 PUSH CrackmeC.0040E140                ASCII "bad cast"
00402467 PUSH CrackmeC.0040E2B8                ASCII "77777777"
00402485 PUSH CrackmeC.0040E29C                ASCII "Welcome to AR Crackme #1"
004024B0 PUSH CrackmeC.0040E278                ASCII "Please enter the correct serial: "
00402523 PUSH CrackmeC.0040E268                ASCII "Checking serial"
004025DB PUSH CrackmeC.0040E248                ASCII "I'm sorry but that is wrong!"
00402640 PUSH CrackmeC.0040E234                ASCII "Try again? Y/N: "
004026CA PUSH CrackmeC.0040E1FC                ASCII "Visit http://cracking.accessroot.com/ for everything AR"
0040272F PUSH CrackmeC.0040E1E4                ASCII "Long live the ARTeam!"
00402790 PUSH CrackmeC.0040E178                ASCII "Greetz & Shoutoutz to Whitefire for hosting our forum, forum.exetools.
00402842 PUSH CrackmeC.0040E170                ASCII "pause"
004028AA PUSH CrackmeC.0040E14C                ASCII "Congratulations that is correct!"
0040290E PUSH CrackmeC.0040E1FC                ASCII "Visit http://cracking.accessroot.com/ for everything AR"
00402AB2 ASCII "h(.A",0
00402AC7 PUSH CrackmeC.00402AB2                ASCII "h(.A"
00402FCE PUSH CrackmeC.0040E330                ASCII "ios_base::badbit set"
00403002 PUSH CrackmeC.0040E318                ASCII "ios_base::failbit set"
00403018 PUSH CrackmeC.0040E300                ASCII "ios_base::eofbit set"
00403200 PUSH CrackmeC.0040E36C                ASCII "invalid string position"
0040329B PUSH CrackmeC.0040E384                ASCII "string too long"
00403A32 PUSH CrackmeC.0040E140                ASCII "bad cast"
004043CD MOV DWORD PTR SS:[EBP-4],CrackmeC     ASCII "bad allocation"
00404CE3 MOV EAX,CrackmeC.0040E4A0             ASCII "Unknown exception"
00404DCF PUSH CrackmeC.0040E500                ASCII "COMSPEC"
00404E00 MOV [LOCAL.3],CrackmeC.0040E4FC       ASCII "/c"
00404E3A MOV [LOCAL.4],CrackmeC.0040E4F0       ASCII "command.com"
00404E43 MOV [LOCAL.4],CrackmeC.0040E4E8       ASCII "cmd.exe"
00405759 MOV DWORD PTR DS:[412428],Crackme     UNICODE "          (((((                    H"
00405DAE PUSH CrackmeC.0040EC18                ASCII "_.,"
00406219 PUSH CrackmeC.0040EA58                UNICODE "         h((((                     H"
00406347 PUSH CrackmeC.0040EC2C                ASCII "=;"
0040822D PUSH CrackmeC.0040ECA0                ASCII "PATH"
00408349 PUSH CrackmeC.0040ECB8                ASCII "mscoree.dll"
00408358 PUSH CrackmeC.0040ECA8                ASCII "CorExitProcess"
00408588 PUSH CrackmeC.0040ECE4                ASCII ".\\"
004087F2 MOV EDI,CrackmeC.0040EEB4             ASCII "Unknown security failure detected!"
004087F7 MOV DWORD PTR SS:[EBP-128],Crackm     ASCII "A security error of unknown cause has been detected which has\ncorrupt
00408808 MOV EDI,CrackmeC.0040EDE0             ASCII "Buffer overrun detected!"
```

That wasn't so hard. Let's dbl-click on the bad boy message, "I'm sorry, but that is wrong" to at least get into the right area:



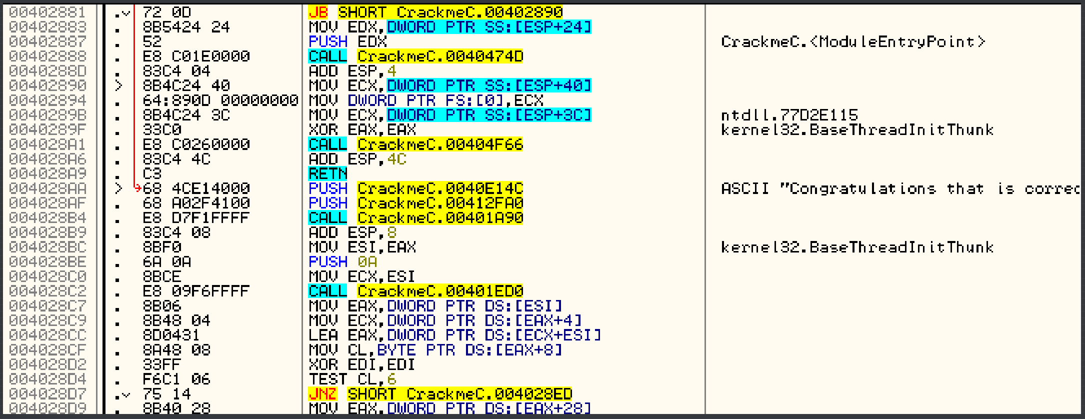
```
004025BD  .v  74 05          JE SHORT CrackmeC.004025C4
004025BF  .   1BC0           SBB EAX,EAX                       kernel32.BaseThreadInitThunk
004025C1  .   83D8 FF        SBB EAX,-1
004025C4  >   3BC3           CMP EAX,EBX
004025C6  .v  75 13          JNZ SHORT CrackmeC.004025DB
004025C8  >   3BD5           CMP EDX,EBP
004025CA  .v  72 0F          JB SHORT CrackmeC.004025DB
004025CC  .   33C0           XOR EAX,EAX                       kernel32.BaseThreadInitThunk
004025CE  .   3BD5           CMP EDX,EBP
004025D0  .   0F95C0         SETNE AL
004025D3  .   3BC3           CMP EAX,EBX
004025D5  .v  0F84 CF020000  JE CrackmeC.004028AA
004025DB  >   68 48E24000    PUSH CrackmeC.0040E248            ASCII "I'm sorry but that is wr
004025E0  .   68 A02F4100    PUSH CrackmeC.00412FA0
004025E5  .   E8 A6F4FFFF    CALL CrackmeC.00401A90
004025EA  .   83C4 08        ADD ESP,8
004025ED  .   8BF0           MOV ESI,EAX                       kernel32.BaseThreadInitThunk
004025EF  .   6A 0A          PUSH 0A
004025F1  .   8BCE           MOV ECX,ESI
004025F3  .   E8 D8F8FFFF    CALL CrackmeC.00401ED0
004025F8  .   8B0E           MOV ECX,DWORD PTR DS:[ESI]
004025FA  .   8B51 04        MOV EDX,DWORD PTR DS:[ECX+4]
004025FD  .   8A4C32 08      MOV CL,BYTE PTR DS:[EDX+ESI+8]
00402601  .   8D0432         LEA EAX,DWORD PTR DS:[EDX+ESI]
00402604  .   33FF           XOR EDI,EDI
00402606  .   F6C1 06        TEST CL,6
00402609  .v  75 14          JNZ SHORT CrackmeC.0040261F
0040260B  .   8B40 28        MOV EAX,DWORD PTR DS:[EAX+28]
0040260E  .   8B10           MOV EDX,DWORD PTR DS:[EAX]
00402610  .   8BC8           MOV ECX,EAX                       kernel32.BaseThreadInitThunk
00402612  .   FF52 2C        CALL DWORD PTR DS:[EDX+2C]
00402615  .   83F8 FF        CMP EAX,-1
00402618  .v  75 05          JNZ SHORT CrackmeC.0040261F
0040261A  .   BF 04000000    MOV EDI,4
0040261F  >   8B06           MOV EAX,DWORD PTR DS:[ESI]
```

Ok, let's study this a little. We see a jump leads to this message from 402C56, denoted by the red arrow. We also notice that we could get to it by not jumping at the JE instruction at address 4025D5. Let's see what happens if we do take this jump. Click on it:



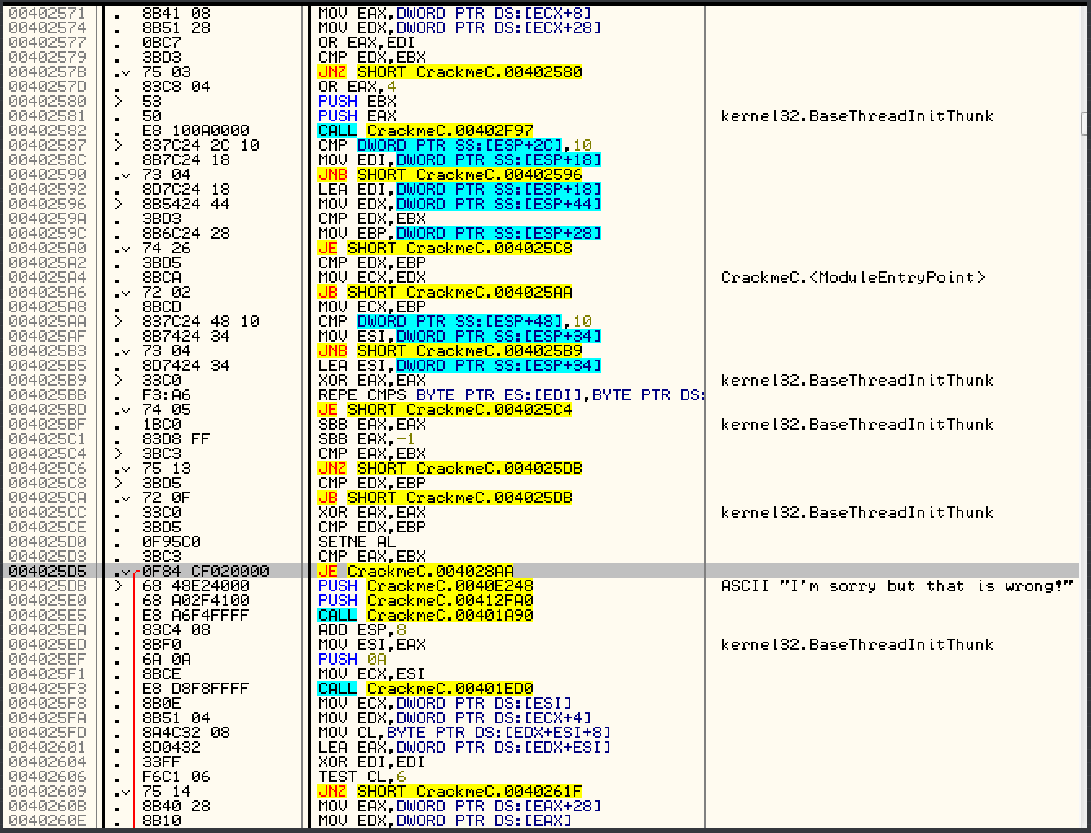
```
004025CE  .    3BD5           CMP EDX,EBP
004025D0  .    0F95C0         SETNE AL
004025D3  .    3BC3           CMP EAX,EBX
004025D5  .v   0F84 CF020000  JE CrackmeC.004028AA
004025DB  >    68 48E24000    PUSH CrackmeC.0040E248              ASCII "I'm sorry but that is wr
004025E0  .    68 A02F4100    PUSH CrackmeC.00412FA0
004025E5  .    E8 A6F4FFFF    CALL CrackmeC.00401A90
004025EA  .    83C4 08        ADD ESP,8
004025ED  .    8BF0           MOV ESI,EAX                         kernel32.BaseThreadInitThunk
004025EF  .    6A 0A          PUSH 0A
004025F1  .    8BCE           MOV ECX,ESI
004025F3  .    E8 D8F8FFFF    CALL CrackmeC.00401ED0
004025F8  .    8B0E           MOV ECX,DWORD PTR DS:[ESI]
004025FA  .    8B51 04        MOV EDX,DWORD PTR DS:[ECX+4]
004025FD  .    8A4C32 08      MOV CL,BYTE PTR DS:[EDX+ESI+8]
00402601  .    8D0432         LEA EAX,DWORD PTR DS:[EDX+ESI]
00402604  .    33FF           XOR EDI,EDI
00402606  .    F6C1 06        TEST CL,6
00402609  .v   75 14          JNZ SHORT CrackmeC.0040261F
0040260B  .    8B40 28        MOV EAX,DWORD PTR DS:[EAX+28]
0040260E  .    8B10           MOV EDX,DWORD PTR DS:[EAX]
00402610  .    8BC8           MOV ECX,EAX                         kernel32.BaseThreadInitThunk
00402612  .    FF52 2C        CALL DWORD PTR DS:[EDX+2C]
00402615  .    83F8 FF        CMP EAX,-1
00402618  .v   75 05          JNZ SHORT CrackmeC.0040261F
0040261A  .    BF 04000000    MOV EDI,4
0040261F  >    8B06           MOV EAX,DWORD PTR DS:[ESI]
00402621  .    8B48 04        MOV ECX,DWORD PTR DS:[EAX+4]
00402624  .    03CE           ADD ECX,ESI
00402626  .    3BFB           CMP EDI,EBX
00402628  .v   74 16          JE SHORT CrackmeC.00402640
0040262A  .    8B41 08        MOV EAX,DWORD PTR DS:[ECX+8]
0040262D  .    8B51 28        MOV EDX,DWORD PTR DS:[ECX+28]
00402630  .    0BC7           OR EAX,EDI
00402632  .    3BD3           CMP EDX,EBX
```

And scroll to where it points (a couple pages down):



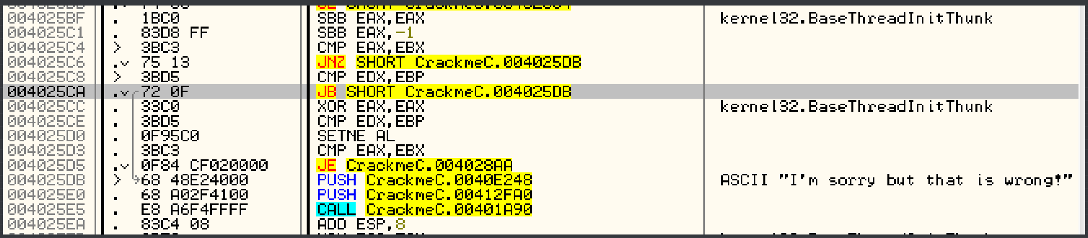
```
00402881  .v   72 0D              JB SHORT CrackmeC.00402890
00402883  .    8B5424 24          MOV EDX,DWORD PTR SS:[ESP+24]
00402887  .    52                 PUSH EDX                        CrackmeC.<ModuleEntryPoint>
00402888  .    E8 C01E0000        CALL CrackmeC.0040474D
0040288D  .    83C4 04            ADD ESP,4
00402890  >    8B4C24 40          MOV ECX,DWORD PTR SS:[ESP+40]
00402894  .    64:890D 00000000   MOV DWORD PTR FS:[0],ECX
0040289B  .    8B4C24 3C          MOV ECX,DWORD PTR SS:[ESP+3C]   ntdll.77D2E115
0040289F  .    33C0               XOR EAX,EAX                     kernel32.BaseThreadInitThunk
004028A1  .    E8 C0260000        CALL CrackmeC.00404F66
004028A6  .    83C4 4C            ADD ESP,4C
004028A9  .    C3                 RETN
004028AA  >    68 4CE14000        PUSH CrackmeC.0040E14C          ASCII "Congratulations that is correc
004028AF  .    68 A02F4100        PUSH CrackmeC.00412FA0
004028B4  .    E8 D7F1FFFF        CALL CrackmeC.00401A90
004028B9  .    83C4 08            ADD ESP,8
004028BC  .    8BF0               MOV ESI,EAX                     kernel32.BaseThreadInitThunk
004028BE  .    6A 0A              PUSH 0A
004028C0  .    8BCE               MOV ECX,ESI
004028C2  .    E8 09F6FFFF        CALL CrackmeC.00401ED0
004028C7  .    8B06               MOV EAX,DWORD PTR DS:[ESI]
004028C9  .    8B48 04            MOV ECX,DWORD PTR DS:[EAX+4]
004028CC  .    8D0431             LEA EAX,DWORD PTR DS:[ECX+ESI]
004028CF  .    8A48 08            MOV CL,BYTE PTR DS:[EAX+8]
004028D2  .    33FF               XOR EDI,EDI
004028D4  .    F6C1 06            TEST CL,6
004028D7  .v   75 14              JNZ SHORT CrackmeC.004028ED
004028D9  .    8B40 28            MOV EAX,DWORD PTR DS:[EAX+28]
```

That looks like the way we want to go 😃 . Let's go back up and look around a little more:



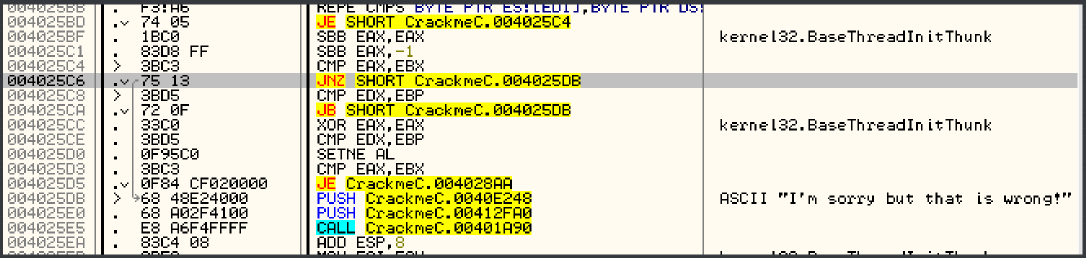
```
00402571  .    8B41 08        MOV EAX,DWORD PTR DS:[ECX+8]
00402574  .    8B51 28        MOV EDX,DWORD PTR DS:[ECX+28]
00402577  .    0BC7           OR EAX,EDI
00402579  .    3BD3           CMP EDX,EBX
0040257B  .v   75 03          JNZ SHORT CrackmeC.00402580
0040257D  .    83C8 04        OR EAX,4
00402580  >    53             PUSH EBX
00402581  .    50             PUSH EAX                                kernel32.BaseThreadInitThunk
00402582  .    E8 100A0000    CALL CrackmeC.00402F97
00402587  >    837C24 2C 10   CMP DWORD PTR SS:[ESP+2C],10
0040258C  .    8B7C24 18      MOV EDI,DWORD PTR SS:[ESP+18]
00402590  .v   73 04          JNB SHORT CrackmeC.00402596
00402592  .    8D7C24 18      LEA EDI,DWORD PTR SS:[ESP+18]
00402596  >    8B5424 44      MOV EDX,DWORD PTR SS:[ESP+44]
0040259A  .    3BD3           CMP EDX,EBX
0040259C  .    8B6C24 28      MOV EBP,DWORD PTR SS:[ESP+28]
004025A0  .v   74 26          JE SHORT CrackmeC.004025C8
004025A2  .    3BD5           CMP EDX,EBP
004025A4  .    8BCA           MOV ECX,EDX                             CrackmeC.<ModuleEntryPoint>
004025A6  .v   72 02          JB SHORT CrackmeC.004025AA
004025A8  .    8BCD           MOV ECX,EBP
004025AA  >    837C24 48 10   CMP DWORD PTR SS:[ESP+48],10
004025AF  .    8B7424 34      MOV ESI,DWORD PTR SS:[ESP+34]
004025B3  .v   73 04          JNB SHORT CrackmeC.004025B9
004025B5  .    8D7424 34      LEA ESI,DWORD PTR SS:[ESP+34]
004025B9  >    33C0           XOR EAX,EAX                             kernel32.BaseThreadInitThunk
004025BB  .    F3:A6          REPE CMPS BYTE PTR ES:[EDI],BYTE PTR DS:
004025BD  .v   74 05          JE SHORT CrackmeC.004025C4
004025BF  .    1BC0           SBB EAX,EAX                             kernel32.BaseThreadInitThunk
004025C1  .    83D8 FF        SBB EAX,-1
004025C4  >    3BC3           CMP EAX,EBX
004025C6  .v   75 13          JNZ SHORT CrackmeC.004025DB
004025C8  >    3BD5           CMP EDX,EBP
004025CA  .v   72 0F          JB SHORT CrackmeC.004025DB
004025CC  .    33C0           XOR EAX,EAX                             kernel32.BaseThreadInitThunk
004025CE  .    3BD5           CMP EDX,EBP
004025D0  .    0F95C0         SETNE AL
004025D3  .    3BC3           CMP EAX,EBX
004025D5  .v   0F84 CF020000  JE CrackmeC.004028AA
004025DB  >    68 48E24000    PUSH CrackmeC.0040E248                  ASCII "I'm sorry but that is wrong!"
004025E0  .    68 A02F4100    PUSH CrackmeC.00412FA0
004025E5  .    E8 A6F4FFFF    CALL CrackmeC.00401A90
004025EA  .    83C4 08        ADD ESP,8
004025ED  .    8BF0           MOV ESI,EAX                             kernel32.BaseThreadInitThunk
004025EF  .    6A 0A          PUSH 0A
004025F1  .    8BCE           MOV ECX,ESI
004025F3  .    E8 D8F8FFFF    CALL CrackmeC.00401ED0
004025F8  .    8B0E           MOV ECX,DWORD PTR DS:[ESI]
004025FA  .    8B51 04        MOV EDX,DWORD PTR DS:[ECX+4]
004025FD  .    8A4C32 08      MOV CL,BYTE PTR DS:[EDX+ESI+8]
00402601  .    8D0432         LEA EAX,DWORD PTR DS:[EDX+ESI]
00402604  .    33FF           XOR EDI,EDI
00402606  .    F6C1 06        TEST CL,6
00402609  .v   75 14          JNZ SHORT CrackmeC.0040261F
0040260B  .    8B40 28        MOV EAX,DWORD PTR DS:[EAX+28]
0040260E  .    8B10           MOV EDX,DWORD PTR DS:[EAX]
```

So address 4025D5 jumps to the good boy message, so that's the jump we'd like to take. Let's try clicking on the other jumps to see where they lead us…maybe there's an earlier jump that takes us to the good boy message:



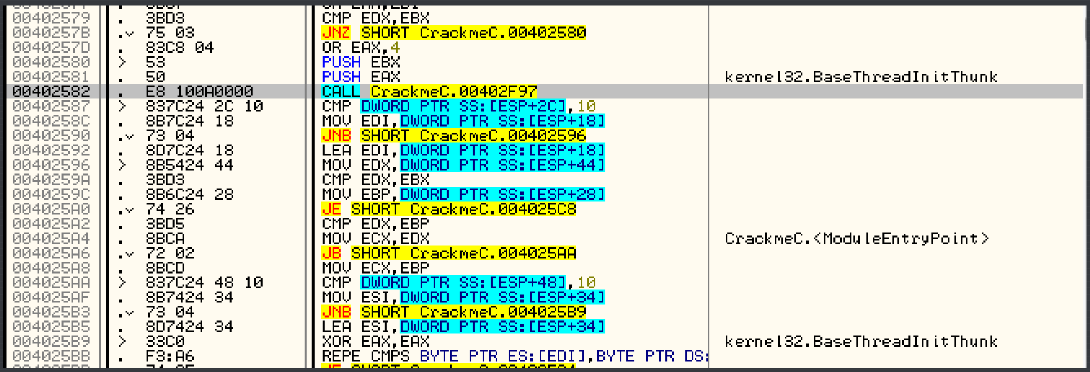
```
004025BF  .   1BC0           SBB EAX,EAX                              kernel32.BaseThreadInitThunk
004025C1  .   83D8 FF        SBB EAX,-1
004025C4  >   3BC3           CMP EAX,EBX
004025C6  .v  75 13          JNZ SHORT CrackmeC.004025DB
004025C8  >   3BD5           CMP EDX,EBP
004025CA  .v  72 0F          JB SHORT CrackmeC.004025DB
004025CC  .   33C0           XOR EAX,EAX                              kernel32.BaseThreadInitThunk
004025CE  .   3BD5           CMP EDX,EBP
004025D0  .   0F95C0         SETNE AL
004025D3  .   3BC3           CMP EAX,EBX
004025D5  .v  0F84 CF020000  JE CrackmeC.004028AA
004025DB  >   68 48E24000    PUSH CrackmeC.0040E248                   ASCII "I'm sorry but that is wrong!"
004025E0  .   68 A02F4100    PUSH CrackmeC.00412FA0
004025E5  .   E8 A6F4FFFF    CALL CrackmeC.00401A90
004025EA  .   83C4 08        ADD ESP,8
```

This one goes to the bad boy:



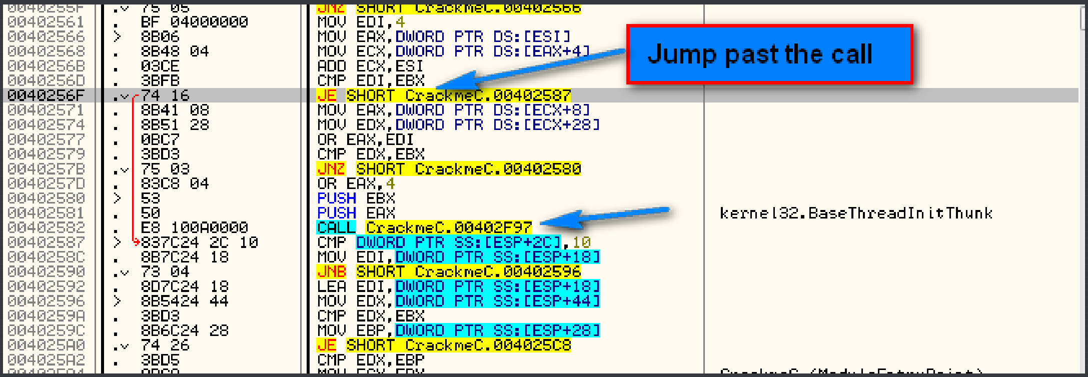
```
004025BD  .v  74 05          JE SHORT CrackmeC.004025C4
004025BF  .   1BC0           SBB EAX,EAX                              kernel32.BaseThreadInitThunk
004025C1  .   83D8 FF        SBB EAX,-1
004025C4  >   3BC3           CMP EAX,EBX
004025C6  .v  75 13          JNZ SHORT CrackmeC.004025DB
004025C8  >   3BD5           CMP EDX,EBP
004025CA  .v  72 0F          JB SHORT CrackmeC.004025DB
004025CC  .   33C0           XOR EAX,EAX                              kernel32.BaseThreadInitThunk
004025CE  .   3BD5           CMP EDX,EBP
004025D0  .   0F95C0         SETNE AL
004025D3  .   3BC3           CMP EAX,EBX
004025D5  .v  0F84 CF020000  JE CrackmeC.004028AA
004025DB  >   68 48E24000    PUSH CrackmeC.0040E248                   ASCII "I'm sorry but that is wrong!"
004025E0  .   68 A02F4100    PUSH CrackmeC.00412FA0
004025E5  .   E8 A6F4FFFF    CALL CrackmeC.00401A90
004025EA  .   83C4 08        ADD ESP,8
```

as does this one, and if you keep clicking on the jump instructions, you'll notice the jump at address 4025D5 is the only one that jumps to the good boy. So basically, we want to keep all jumps that jump to the bad boy from jumping, and force the jump to the goodboy into jumping. If we keep scrolling up, we reach our first call/compare instructions at address 402582:



```
00402579  .   3BD3           CMP EDX,EBX
0040257B  .v  75 03          JNZ SHORT CrackmeC.00402580
0040257D  .   83C8 04        OR EAX,4
00402580  >   53             PUSH EBX
00402581  .   50             PUSH EAX                                 kernel32.BaseThreadInitThunk
00402582  .   E8 100A0000    CALL CrackmeC.00402F97
00402587  >   837C24 2C 10   CMP DWORD PTR SS:[ESP+2C],10
0040258C  .   8B7C24 18      MOV EDI,DWORD PTR SS:[ESP+18]
00402590  .v  73 04          JNB SHORT CrackmeC.00402596
00402592  .   8D7C24 18      LEA EDI,DWORD PTR SS:[ESP+18]
00402596  >   8B5424 44      MOV EDX,DWORD PTR SS:[ESP+44]
0040259A  .   3BD3           CMP EDX,EBX
0040259C  .   8B6C24 28      MOV EBP,DWORD PTR SS:[ESP+28]
004025A0  .v  74 26          JE SHORT CrackmeC.004025C8
004025A2  .   3BD5           CMP EDX,EBP
004025A4  .   8BCA           MOV ECX,EDX                              CrackmeC.<ModuleEntryPoint>
004025A6  .v  72 02          JB SHORT CrackmeC.004025AA
004025A8  .   8BCD           MOV ECX,EBP
004025AA  >   837C24 48 10   CMP DWORD PTR SS:[ESP+48],10
004025AF  .   8B7424 34      MOV ESI,DWORD PTR SS:[ESP+34]
004025B3  .v  73 04          JNB SHORT CrackmeC.004025B9
004025B5  .   8D7424 34      LEA ESI,DWORD PTR SS:[ESP+34]
004025B9  >   33C0           XOR EAX,EAX                              kernel32.BaseThreadInitThunk
004025BB  .   F3:A6          REPE CMPS BYTE PTR ES:[EDI],BYTE PTR DS:
```

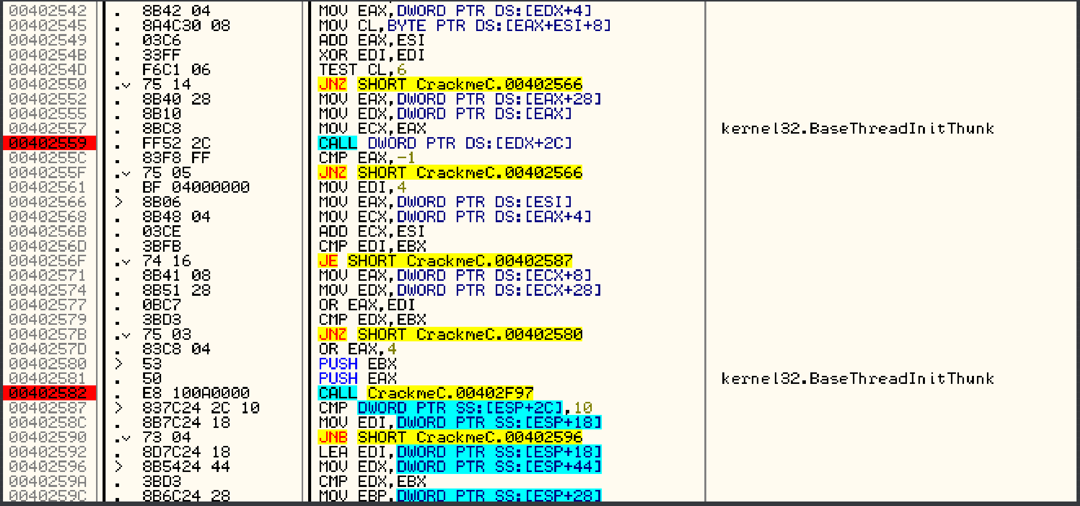
Scrolling further, we can see that there is a jump that skips the call but still performs the compare:



Jump past the call

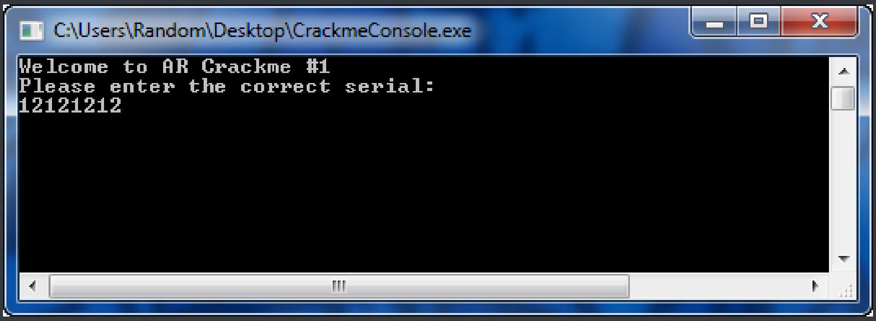
```
00402561  .   BF 04000000    MOV EDI,4
00402566  >   8B06           MOV EAX,DWORD PTR DS:[ESI]
00402568  .   8B48 04        MOV ECX,DWORD PTR DS:[EAX+4]
0040256B  .   03CE           ADD ECX,ESI
0040256D  .   3BFB           CMP EDI,EBX
0040256F  .v  74 16          JE SHORT CrackmeC.00402587
00402571  .   8B41 08        MOV EAX,DWORD PTR DS:[ECX+8]
00402574  .   8B51 28        MOV EDX,DWORD PTR DS:[ECX+28]
00402577  .   0BC7           OR EAX,EDI
00402579  .   3BD3           CMP EDX,EBX
0040257B  .v  75 03          JNZ SHORT CrackmeC.00402580
0040257D  .   83C8 04        OR EAX,4
00402580  >   53             PUSH EBX
00402581  .   50             PUSH EAX                                 kernel32.BaseThreadInitThunk
00402582  .   E8 100A0000    CALL CrackmeC.00402F97
00402587  >   837C24 2C 10   CMP DWORD PTR SS:[ESP+2C],10
0040258C  .   8B7C24 18      MOV EDI,DWORD PTR SS:[ESP+18]
00402590  .v  73 04          JNB SHORT CrackmeC.00402596
00402592  .   8D7C24 18      LEA EDI,DWORD PTR SS:[ESP+18]
00402596  >   8B5424 44      MOV EDX,DWORD PTR SS:[ESP+44]
0040259A  .   3BD3           CMP EDX,EBX
0040259C  .   8B6C24 28      MOV EBP,DWORD PTR SS:[ESP+28]
004025A0  .v  74 26          JE SHORT CrackmeC.004025C8
004025A2  .   3BD5           CMP EDX,EBP
```

That's not exactly normal behavior, but when we scroll up a little more we see another call compare group. I have placed a BP on both of these calls:



```
00402542  .    8B42 04       MOV EAX,DWORD PTR DS:[EDX+4]
00402545  .    8A4C30 08     MOV CL,BYTE PTR DS:[EAX+ESI+8]
00402549  .    03C6          ADD EAX,ESI
0040254B  .    33FF          XOR EDI,EDI
0040254D  .    F6C1 06       TEST CL,6
00402550  .v   75 14         JNZ SHORT CrackmeC.00402566
00402552  .    8B40 28       MOV EAX,DWORD PTR DS:[EAX+28]
00402555  .    8B10          MOV EDX,DWORD PTR DS:[EAX]
00402557  .    8BC8          MOV ECX,EAX                        kernel32.BaseThreadInitThunk
00402559  .    FF52 2C       CALL DWORD PTR DS:[EDX+2C]
0040255C  .    83F8 FF       CMP EAX,-1
0040255F  .v   75 05         JNZ SHORT CrackmeC.00402566
00402561  .    BF 04000000   MOV EDI,4
00402566  >    8B06          MOV EAX,DWORD PTR DS:[ESI]
00402568  .    8B48 04       MOV ECX,DWORD PTR DS:[EAX+4]
0040256B  .    03CE          ADD ECX,ESI
0040256D  .    3BFB          CMP EDI,EBX
0040256F  .v   74 16         JE SHORT CrackmeC.00402587
00402571  .    8B41 08       MOV EAX,DWORD PTR DS:[ECX+8]
00402574  .    8B51 28       MOV EDX,DWORD PTR DS:[ECX+28]
00402577  .    0BC7          OR EAX,EDI
00402579  .    3BD3          CMP EDX,EBX
0040257B  .v   75 03         JNZ SHORT CrackmeC.00402580
0040257D  .    83C8 04       OR EAX,4
00402580  >    53            PUSH EBX
00402581  .    50            PUSH EAX                           kernel32.BaseThreadInitThunk
00402582  .    E8 100A0000   CALL CrackmeC.00402F97
00402587  >    837C24 2C 10  CMP DWORD PTR SS:[ESP+2C],10
0040258C  .    8B7C24 18     MOV EDI,DWORD PTR SS:[ESP+18]
00402590  .v   73 04         JNB SHORT CrackmeC.00402596
00402592  .    8D7C24 18     LEA EDI,DWORD PTR SS:[ESP+18]
00402596  >    8B5424 44     MOV EDX,DWORD PTR SS:[ESP+44]
0040259A  .    3BD3          CMP EDX,EBX
0040259C  .    8B6C24 28     MOV EBP,DWORD PTR SS:[ESP+28]
```

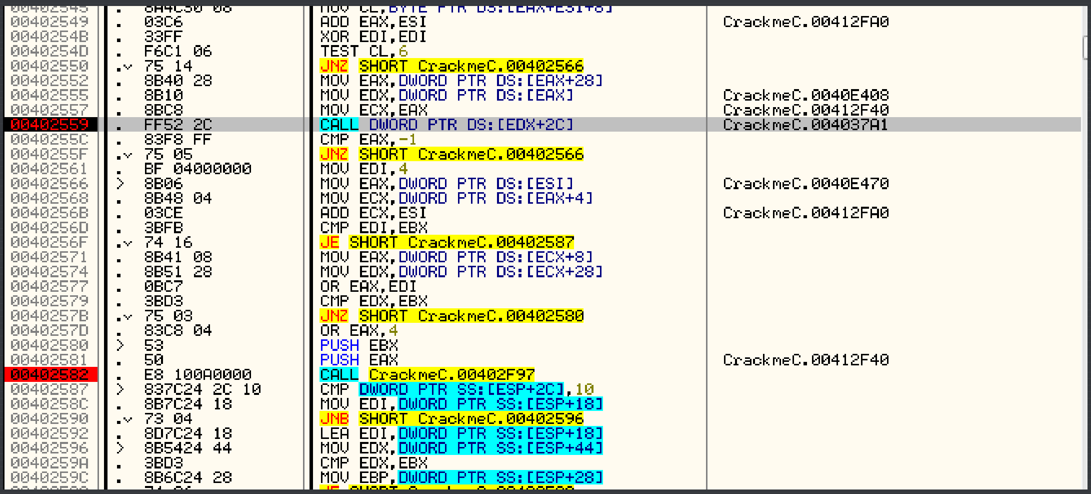
OK, let’s go ahead and run the app in Olly and see what happens. I’l enter the password ’12121212’:



C:\Users\Random\Desktop\CrackmeConsole.exe

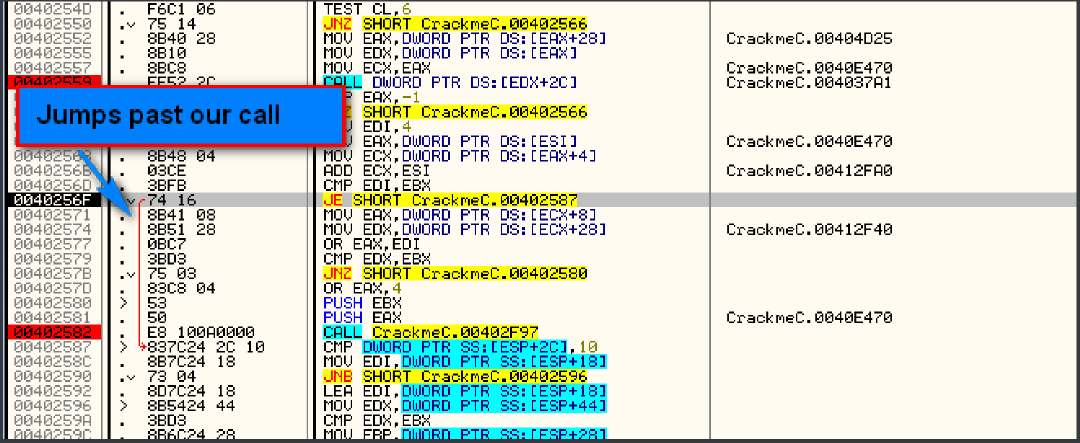
```
Welcome to AR Crackme #1
Please enter the correct serial:
12121212
```

and Olly breaks at the first call:



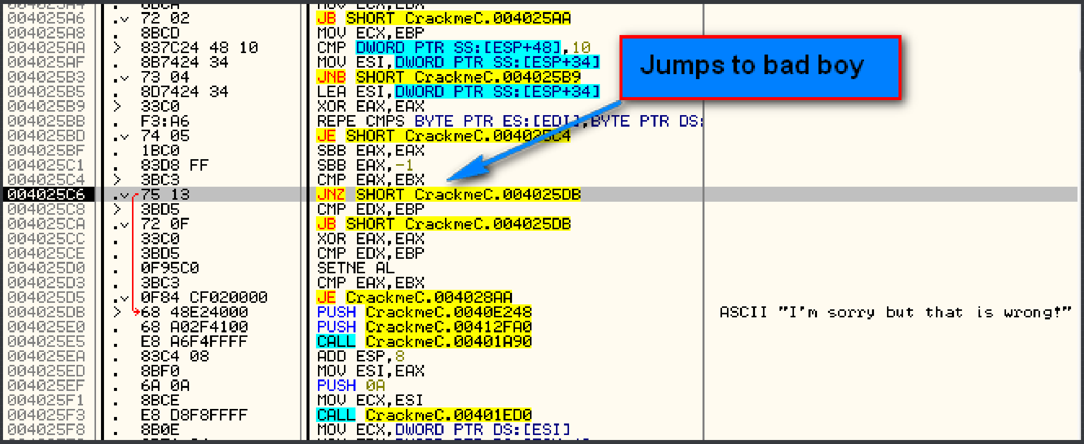
```
00402549  .    03C6          ADD EAX,ESI                        CrackmeC.00412FA0
0040254B  .    33FF          XOR EDI,EDI
0040254D  .    F6C1 06       TEST CL,6
00402550  .v   75 14         JNZ SHORT CrackmeC.00402566
00402552  .    8B40 28       MOV EAX,DWORD PTR DS:[EAX+28]
00402555  .    8B10          MOV EDX,DWORD PTR DS:[EAX]         CrackmeC.0040E408
00402557  .    8BC8          MOV ECX,EAX                        CrackmeC.00412F40
00402559  .    FF52 2C       CALL DWORD PTR DS:[EDX+2C]         CrackmeC.004037A1
0040255C  .    83F8 FF       CMP EAX,-1
0040255F  .v   75 05         JNZ SHORT CrackmeC.00402566
00402561  .    BF 04000000   MOV EDI,4
00402566  >    8B06          MOV EAX,DWORD PTR DS:[ESI]         CrackmeC.0040E470
00402568  .    8B48 04       MOV ECX,DWORD PTR DS:[EAX+4]
0040256B  .    03CE          ADD ECX,ESI                        CrackmeC.00412FA0
0040256D  .    3BFB          CMP EDI,EBX
0040256F  .v   74 16         JE SHORT CrackmeC.00402587
00402571  .    8B41 08       MOV EAX,DWORD PTR DS:[ECX+8]
00402574  .    8B51 28       MOV EDX,DWORD PTR DS:[ECX+28]
00402577  .    0BC7          OR EAX,EDI
00402579  .    3BD3          CMP EDX,EBX
0040257B  .v   75 03         JNZ SHORT CrackmeC.00402580
0040257D  .    83C8 04       OR EAX,4
00402580  >    53            PUSH EBX
00402581  .    50            PUSH EAX                           CrackmeC.00412F40
00402582  .    E8 100A0000   CALL CrackmeC.00402F97
00402587  >    837C24 2C 10  CMP DWORD PTR SS:[ESP+2C],10
0040258C  .    8B7C24 18     MOV EDI,DWORD PTR SS:[ESP+18]
00402590  .v   73 04         JNB SHORT CrackmeC.00402596
00402592  .    8D7C24 18     LEA EDI,DWORD PTR SS:[ESP+18]
00402596  >    8B5424 44     MOV EDX,DWORD PTR SS:[ESP+44]
0040259A  .    3BD3          CMP EDX,EBX
0040259C  .    8B6C24 28     MOV EBP,DWORD PTR SS:[ESP+28]
```

Start single stepping and you will notice that the jump at 40256F jumps the second call. Hmmm, this gives us an indication that this second jump may not be the password checker after all, but maybe some sort of routine if our password does not meet certain specs, like too short or too long? Whatever, let’s keep single stepping:



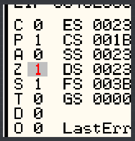
```
0040254D  .    F6C1 06         TEST CL,6
00402550  .v   75 14           JNZ SHORT CrackmeC.00402566
00402552  .    8B40 28         MOV EAX,DWORD PTR DS:[EAX+28]     CrackmeC.00404D25
00402555  .    8B10            MOV EDX,DWORD PTR DS:[EAX]
00402557  .    8BC8            MOV ECX,EAX                       CrackmeC.0040E470
00402559       FF52 2C         CALL DWORD PTR DS:[EDX+2C]        CrackmeC.004037A1
                                P EAX,-1
                                Z SHORT CrackmeC.00402566
                                V EDI,4
                                V EAX,DWORD PTR DS:[ESI]          CrackmeC.0040E470
00402568  .    8B48 04         MOV ECX,DWORD PTR DS:[EAX+4]
0040256B  .    03CE            ADD ECX,ESI                       CrackmeC.00412FA0
0040256D  .    3BFB            CMP EDI,EBX
0040256F  .v   74 16           JE SHORT CrackmeC.00402587
00402571  .    8B41 08         MOV EAX,DWORD PTR DS:[ECX+8]
00402574  .    8B51 28         MOV EDX,DWORD PTR DS:[ECX+28]     CrackmeC.00412F40
00402577  .    0BC7            OR EAX,EDI
00402579  .    3BD3            CMP EDX,EBX
0040257B  .v   75 03           JNZ SHORT CrackmeC.00402580
0040257D  .    83C8 04         OR EAX,4
00402580  >    53              PUSH EBX
00402581  .    50              PUSH EAX                          CrackmeC.0040E470
00402582  .    E8 100A0000     CALL CrackmeC.00402F97
00402587  >    837C24 2C 10    CMP DWORD PTR SS:[ESP+2C],10
0040258C  .    8B7C24 18       MOV EDI,DWORD PTR SS:[ESP+18]
00402590  .v   73 04           JNB SHORT CrackmeC.00402596
00402592  .    8D7C24 18       LEA EDI,DWORD PTR SS:[ESP+18]
00402596  >    8B5424 44       MOV EDX,DWORD PTR SS:[ESP+44]
0040259A  .    3BD3            CMP EDX,EBX
0040259C  .    8B6C24 28       MOV EBP,DWORD PTR SS:[ESP+28]
```

Here, at address 4025C6, we see our main culprit that jumps to our bad boy message:



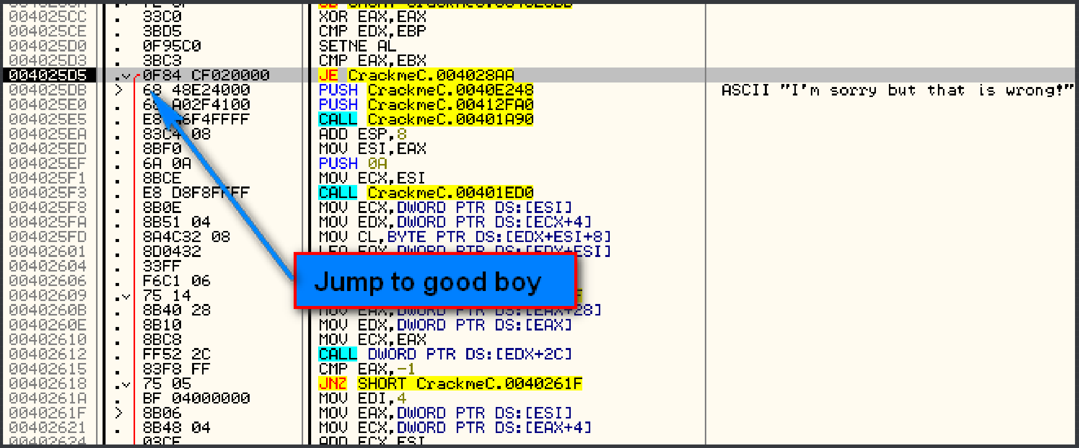
```
004025A6  .v   72 02           JB SHORT CrackmeC.004025AA
004025A8  .    8BCD            MOV ECX,EBP
004025AA  >    837C24 48 10    CMP DWORD PTR SS:[ESP+48],10
004025AF  .    8B7424 34       MOV ESI,DWORD PTR SS:[ESP+34]
004025B3  .v   73 04           JNB SHORT CrackmeC.004025B9
004025B5  .    8D7424 34       LEA ESI,DWORD PTR SS:[ESP+34]
004025B9  >    33C0            XOR EAX,EAX
004025BB  .    F3:A6           REPE CMPS BYTE PTR ES:[EDI],BYTE PTR DS:
004025BD  .v   74 05           JE SHORT CrackmeC.004025C4
004025BF  .    1BC0            SBB EAX,EAX
004025C1  .    83D8 FF         SBB EAX,-1
004025C4  >    3BC3            CMP EAX,EBX
004025C6  .v   75 13           JNZ SHORT CrackmeC.004025DB
004025C8  >    3BD5            CMP EDX,EBP
004025CA  .v   72 0F           JB SHORT CrackmeC.004025DB
004025CC  .    33C0            XOR EAX,EAX
004025CE  .    3BD5            CMP EDX,EBP
004025D0  .    0F95C0          SETNE AL
004025D3  .    3BC3            CMP EAX,EBX
004025D5  .v   0F84 CF020000   JE CrackmeC.004028AA
004025DB  >    68 48E24000     PUSH CrackmeC.0040E248            ASCII "I'm sorry but that is wrong!"
004025E0  .    68 A02F4100     PUSH CrackmeC.00412FA0
004025E5  .    E8 A6F4FFFF     CALL CrackmeC.00401A90
004025EA  .    83C4 08         ADD ESP,8
004025ED  .    8BF0            MOV ESI,EAX
004025EF  .    6A 0A           PUSH 0A
004025F1  .    8BCE            MOV ECX,ESI
004025F3  .    E8 D8F8FFFF     CALL CrackmeC.00401ED0
004025F8  .    8B0E            MOV ECX,DWORD PTR DS:[ESI]
```

Let’s set the zero flag and see what happens:



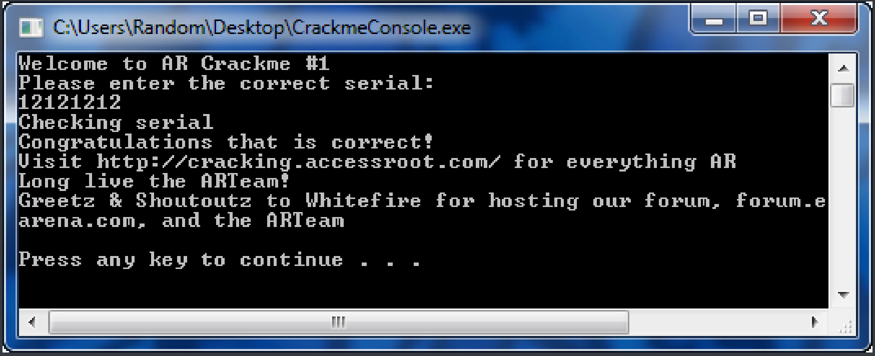
```
C 0  ES 0023
P 1  CS 001B
A 0  SS 0023
Z 1  DS 0023
S 1  FS 003B
T 0  GS 0000
D 0
O 0  LastErr
```

and as we continue to single step, we hit our jump to the good boy and notice that it is taken:



```
004025CC  .    33C0            XOR EAX,EAX
004025CE  .    3BD5            CMP EDX,EBP
004025D0  .    0F95C0          SETNE AL
004025D3  .    3BC3            CMP EAX,EBX
004025D5  .v   0F84 CF020000   JE CrackmeC.004028AA
004025DB  >    68 48E24000     PUSH CrackmeC.0040E248            ASCII "I'm sorry but that is wrong!"
004025E0  .    68 A02F4100     PUSH CrackmeC.00412FA0
004025E5  .    E8 A6F4FFFF     CALL CrackmeC.00401A90
004025EA  .    83C4 08         ADD ESP,8
004025ED  .    8BF0            MOV ESI,EAX
004025EF  .    6A 0A           PUSH 0A
004025F1  .    8BCE            MOV ECX,ESI
004025F3  .    E8 D8F8FFFF     CALL CrackmeC.00401ED0
004025F8  .    8B0E            MOV ECX,DWORD PTR DS:[ESI]
004025FA  .    8B51 04         MOV EDX,DWORD PTR DS:[ECX+4]
004025FD  .    8A4C32 08       MOV CL,BYTE PTR DS:[EDX+ESI+8]
00402601  .    8D0432          LEA EAX,DWORD PTR DS:[EDX+ESI]
00402604  .    33FF
00402606  .    F6C1 06
00402609  .v   75 14
0040260B  .    8B40 28         MOV EAX,DWORD PTR DS:[EAX+28]
0040260E  .    8B10            MOV EDX,DWORD PTR DS:[EAX]
00402610  .    8BC8            MOV ECX,EAX
00402612  .    FF52 2C         CALL DWORD PTR DS:[EDX+2C]
00402615  .    83F8 FF         CMP EAX,-1
00402618  .v   75 05           JNZ SHORT CrackmeC.0040261F
0040261A  .    BF 04000000     MOV EDI,4
0040261F  >    8B06            MOV EAX,DWORD PTR DS:[ESI]
00402621  .    8B48 04         MOV ECX,DWORD PTR DS:[EAX+4]
```

Go ahead and run the app and we notice that we have found our first potential patch:

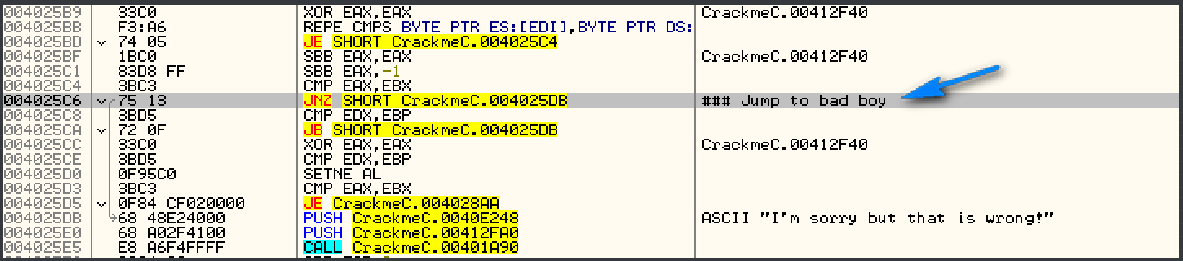


```
Welcome to AR Crackme #1
Please enter the correct serial:
12121212
Checking serial
Congratulations that is correct!
Visit http://cracking.accessroot.com/ for everything AR
Long live the ARTeam!
Greetz & Shoutoutz to Whitefire for hosting our forum, forum.e
arena.com, and the ARTeam

Press any key to continue . . .
```

Now, patching the jump where we set the zero flag may work, or may not work. It's hard to tell. What if our password is too short? Too long? A different password than the one entered. This patch is not a very good patch as we don't really know what we've done, we just know it happened to work in this case.

Digging Deeper

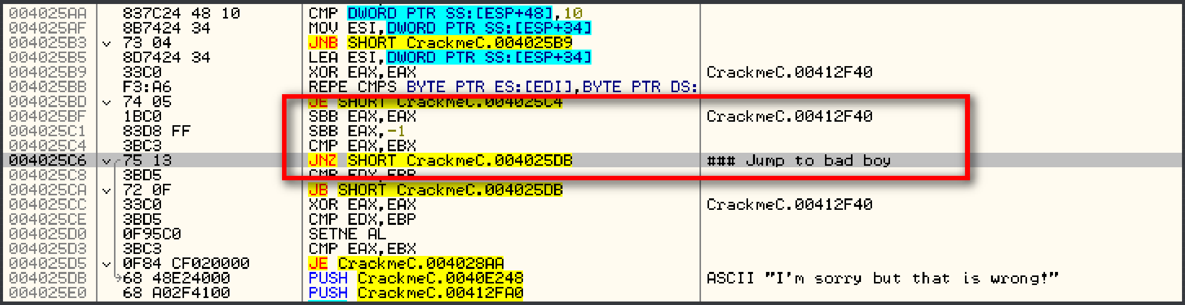
Let's look at this code a little closer, using the levels we learned in the last tutorial, and try something not so LAME. Scroll back up to the jump to the bad boy that we patched and let's try to figure out why we would have jumped had we not patched it. Notice that I have also placed a comment on the jump so I can remember it later (if you recall, highlight the line and hit ';' to add a comment).



```
004025B9     33C0           XOR EAX,EAX                                   CrackmeC.00412F40
004025BB     F3:A6          REPE CMPS BYTE PTR ES:[EDI],BYTE PTR DS:
004025BD  v  74 05          JE SHORT CrackmeC.004025C4
004025BF     1BC0           SBB EAX,EAX                                   CrackmeC.00412F40
004025C1     83D8 FF        SBB EAX,-1
004025C4     3BC3           CMP EAX,EBX
004025C6  v  75 13          JNZ SHORT CrackmeC.004025DB                   ### Jump to bad boy
004025C8     3BD5           CMP EDX,EBP
004025CA  v  72 0F          JB SHORT CrackmeC.004025DB
004025CC     33C0           XOR EAX,EAX                                   CrackmeC.00412F40
004025CE     3BD5           CMP EDX,EBP
004025D0     0F95C0         SETNE AL
004025D3     3BC3           CMP EAX,EBX
004025D5  v  0F84 CF020000  JE CrackmeC.004028AA
004025DB     68 48E24000    PUSH CrackmeC.0040E248                        ASCII "I'm sorry but that is wrong!"
004025E0     68 A02F4100    PUSH CrackmeC.00412FA0
004025E5     E8 A6F4FFFF    CALL CrackmeC.00401A90
```

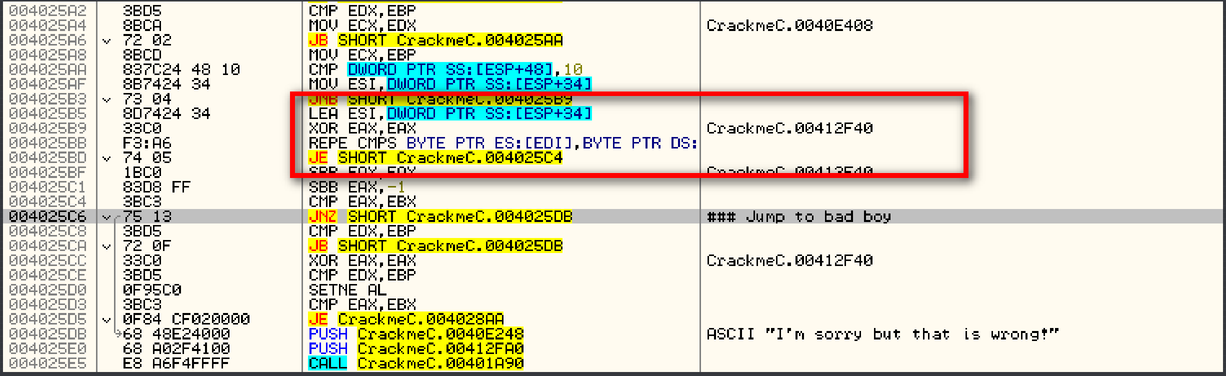
I usually preclude all of my comments with a '###', this way, later, when using other tools that fill in the comments column for us, it's easier to find my own comments- they stand out more. You can do whatever you like though.

Now, let's look just above this jump and see if we can figure out what caused it. Here I have marked the first section above the jump:



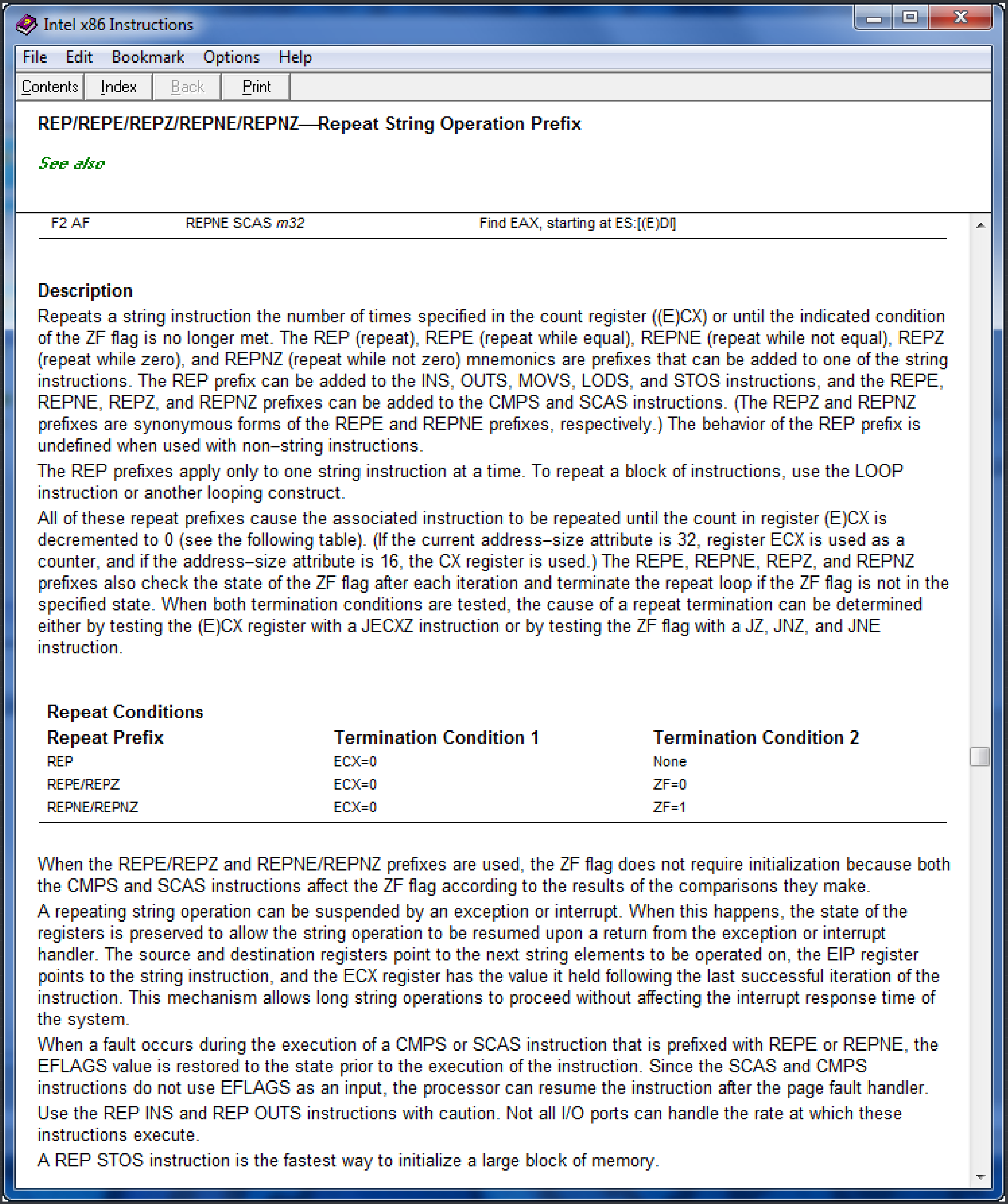
```
004025AA     837C24 48 10   CMP DWORD PTR SS:[ESP+48],10
004025AF     8B7424 34      MOV ESI,DWORD PTR SS:[ESP+34]
004025B3  v  73 04          JNB SHORT CrackmeC.004025B9
004025B5     8D7424 34      LEA ESI,DWORD PTR SS:[ESP+34]
004025B9     33C0           XOR EAX,EAX                                   CrackmeC.00412F40
004025BB     F3:A6          REPE CMPS BYTE PTR ES:[EDI],BYTE PTR DS:
004025BD  v  74 05          JE SHORT CrackmeC.004025C4
004025BF     1BC0           SBB EAX,EAX                                   CrackmeC.00412F40
004025C1     83D8 FF        SBB EAX,-1
004025C4     3BC3           CMP EAX,EBX
004025C6  v  75 13          JNZ SHORT CrackmeC.004025DB                   ### Jump to bad boy
004025C8     3BD5           CMP EDX,EBP
004025CA  v  72 0F          JB SHORT CrackmeC.004025DB
004025CC     33C0           XOR EAX,EAX                                   CrackmeC.00412F40
004025CE     3BD5           CMP EDX,EBP
004025D0     0F95C0         SETNE AL
004025D3     3BC3           CMP EAX,EBX
004025D5  v  0F84 CF020000  JE CrackmeC.004028AA
004025DB     68 48E24000    PUSH CrackmeC.0040E248                        ASCII "I'm sorry but that is wrong!"
004025E0     68 A02F4100    PUSH CrackmeC.00412FA0
```

Here we can see some SBB instructions with a compare. This code doesn't really mean a lot here to us as we have no idea what any of it pertains to, so let's go up to the next section and see if we can start making some sense of it:



```
004025A2     3BD5            CMP EDX,EBP
004025A4     8BCA            MOV ECX,EDX                              CrackmeC.0040E408
004025A6   v 72 02           JB SHORT CrackmeC.004025AA
004025A8     8BCD            MOV ECX,EBP
004025AA     837C24 48 10    CMP DWORD PTR SS:[ESP+48],10
004025AF     8B7424 34       MOV ESI,DWORD PTR SS:[ESP+34]
004025B3   v 73 04           JNB SHORT CrackmeC.004025B9
004025B5     8D7424 34       LEA ESI,DWORD PTR SS:[ESP+34]
004025B9     33C0            XOR EAX,EAX                              CrackmeC.00412F40
004025BB     F3:A6           REPE CMPS BYTE PTR ES:[EDI],BYTE PTR DS:
004025BD   v 74 05           JE SHORT CrackmeC.004025C4
004025BF     1BC0            SBB EAX,EAX                              CrackmeC.00412F40
004025C1     83D8 FF         SBB EAX,-1
004025C4     3BC3            CMP EAX,EBX
004025C6   v 75 13           JNZ SHORT CrackmeC.004025DB              ### Jump to bad boy
004025C8     3BD5            CMP EDX,EBP
004025CA   v 72 0F           JB SHORT CrackmeC.004025DB
004025CC     33C0            XOR EAX,EAX                              CrackmeC.00412F40
004025CE     3BD5            CMP EDX,EBP
004025D0     0F95C0          SETNE AL
004025D3     3BC3            CMP EAX,EBX
004025D5   v 0F84 CF020000   JE CrackmeC.004028AA
004025DB     68 48E24000     PUSH CrackmeC.0040E248                   ASCII "I'm sorry but that is wrong!"
004025E0     68 A02F4100     PUSH CrackmeC.00412FA0
004025E5     E8 A6F4FFFF     CALL CrackmeC.00401A90
```

Alright, here we're getting somewhere. The first thing you may notice is the REPE CMPS instruction. This is a red flag in reverse engineering! Let's look up REPE and see what it says:



REP/REPE/REPZ/REPNE/REPNZ—Repeat String Operation Prefix

See also

F2 AF	REPNE SCAS *m32*	Find EAX, starting at ES:[(E)DI]

Description

Repeats a string instruction the number of times specified in the count register ((E)CX) or until the indicated condition of the ZF flag is no longer met. The REP (repeat), REPE (repeat while equal), REPNE (repeat while not equal), REPZ (repeat while zero), and REPNZ (repeat while not zero) mnemonics are prefixes that can be added to one of the string instructions. The REP prefix can be added to the INS, OUTS, MOVS, LODS, and STOS instructions, and the REPE, REPNE, REPZ, and REPNZ prefixes can be added to the CMPS and SCAS instructions. (The REPZ and REPNZ prefixes are synonymous forms of the REPE and REPNE prefixes, respectively.) The behavior of the REP prefix is undefined when used with non–string instructions.

The REP prefixes apply only to one string instruction at a time. To repeat a block of instructions, use the LOOP instruction or another looping construct.

All of these repeat prefixes cause the associated instruction to be repeated until the count in register (E)CX is decremented to 0 (see the following table). (If the current address–size attribute is 32, register ECX is used as a counter, and if the address–size attribute is 16, the CX register is used.) The REPE, REPNE, REPZ, and REPNZ prefixes also check the state of the ZF flag after each iteration and terminate the repeat loop if the ZF flag is not in the specified state. When both termination conditions are tested, the cause of a repeat termination can be determined either by testing the (E)CX register with a JECXZ instruction or by testing the ZF flag with a JZ, JNZ, and JNE instruction.

Repeat Conditions

Repeat Prefix	Termination Condition 1	Termination Condition 2
REP	ECX=0	None
REPE/REPZ	ECX=0	ZF=0
REPNE/REPNZ	ECX=0	ZF=1

When the REPE/REPZ and REPNE/REPNZ prefixes are used, the ZF flag does not require initialization because both the CMPS and SCAS instructions affect the ZF flag according to the results of the comparisons they make.

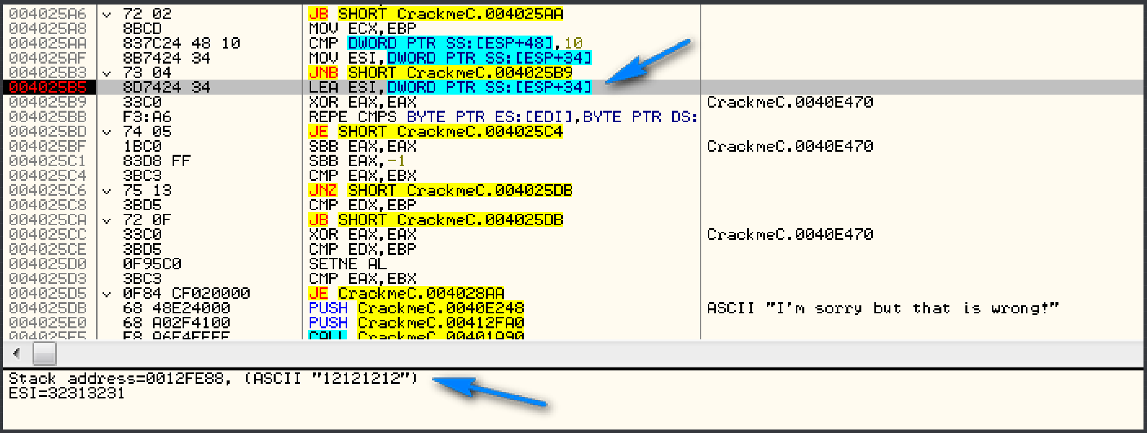
A repeating string operation can be suspended by an exception or interrupt. When this happens, the state of the registers is preserved to allow the string operation to be resumed upon a return from the exception or interrupt handler. The source and destination registers point to the next string elements to be operated on, the EIP register points to the string instruction, and the ECX register has the value it held following the last successful iteration of the instruction. This mechanism allows long string operations to proceed without affecting the interrupt response time of the system.

When a fault occurs during the execution of a CMPS or SCAS instruction that is prefixed with REPE or REPNE, the EFLAGS value is restored to the state prior to the execution of the instruction. Since the SCAS and CMPS instructions do not use EFLAGS as an input, the processor can resume the instruction after the page fault handler.

Use the REP INS and REP OUTS instructions with caution. Not all I/O ports can handle the rate at which these instructions execute.

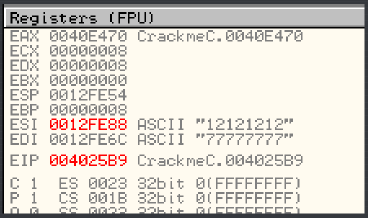
A REP STOS instruction is the fastest way to initialize a large block of memory.

It's not horribly clear, but if you have any experience with assembly you know that the REPXX statement repeats like a loop until ECX = 0. The instruction after the REPXX, in this case CMPS, is what is repeated. Taken together, this statement means "Repeat comparing two memory addresses, incrementing this address each time through the loop, while the zero flag remains equal." In basic terms, it means "compare these two strings." In reverse engineering, anytime we compare two strings, red flags should go off. It is not done very often in an app, and checking a serial number/password/registration key is one of the few times it is. Let's place a BP on the first line of this section at address 4025B5 and re-start the app. Enter our password and Olly will break at this breakpoint:



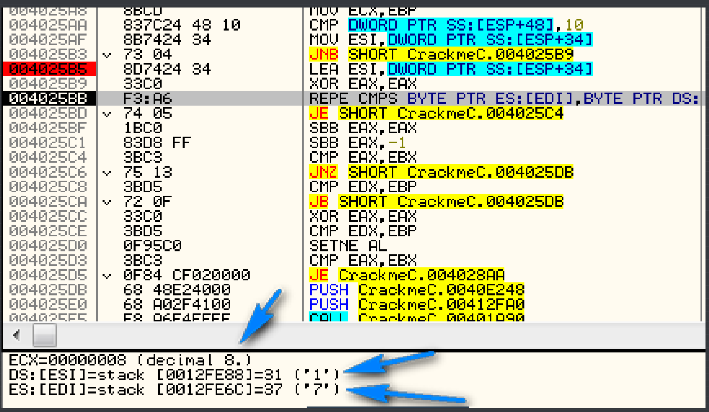
```
004025A6  v 72 02           JB SHORT CrackmeC.004025AA
004025A8    8BCD            MOV ECX,EBP
004025AA    837C24 48 10    CMP DWORD PTR SS:[ESP+48],10
004025AF    8B7424 34       MOV ESI,DWORD PTR SS:[ESP+34]
004025B3  v 73 04           JNB SHORT CrackmeC.004025B9
004025B5    8D7424 34       LEA ESI,DWORD PTR SS:[ESP+34]
004025B9    33C0            XOR EAX,EAX                           CrackmeC.0040E470
004025BB    F3:A6           REPE CMPS BYTE PTR ES:[EDI],BYTE PTR DS:
004025BD  v 74 05           JE SHORT CrackmeC.004025C4
004025BF    1BC0            SBB EAX,EAX                           CrackmeC.0040E470
004025C1    83D8 FF         SBB EAX,-1
004025C4    3BC3            CMP EAX,EBX
004025C6  v 75 13           JNZ SHORT CrackmeC.004025DB
004025C8    3BD5            CMP EDX,EBP
004025CA  v 72 0F           JB SHORT CrackmeC.004025DB
004025CC    33C0            XOR EAX,EAX                           CrackmeC.0040E470
004025CE    3BD5            CMP EDX,EBP
004025D0    0F95C0          SETNE AL
004025D3    3BC3            CMP EAX,EBX
004025D5  v 0F84 CF020000   JE CrackmeC.004028AA
004025DB    68 48E24000     PUSH CrackmeC.0040E248                ASCII "I'm sorry but that is wrong!"
004025E0    68 A02F4100     PUSH CrackmeC.00412FA0
004025E5    E8 A6F4FFFF     CALL CrackmeC.00401090

Stack address=0012FE88, (ASCII "12121212")
ESI=32313231
```

Now notice that the first instruction, LEA ESI, DWORD PTR SS:[ESP+34], is Loading an Effective Address into ESI from the stack. The SS: denotes the stack, the [ESP+34] denotes the position on the stack, in this case the 34th byte past whatever the ESP register is pointing to, and the LEA instruction means basically load the address of something, as opposed to the contents of the something. If we look at the middle bar (where the blue arrow is pointing) we can see that SS:[ESP+34] equals address 0012FE88, and at this address is stored our ASCII password. Single stepping once over this line shows ESI being set equal to our password (that is currently on the stack):



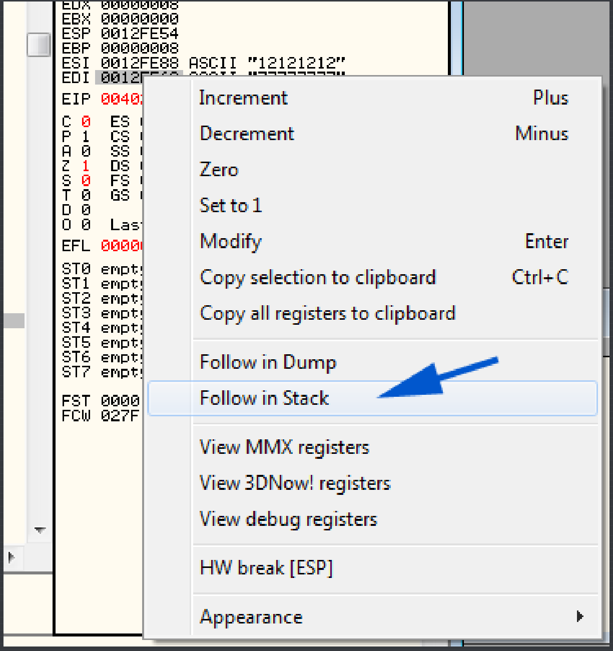
```
Registers (FPU)
EAX 0040E470 CrackmeC.0040E470
ECX 00000008
EDX 00000008
EBX 00000000
ESP 0012FE54
EBP 00000008
ESI 0012FE88 ASCII "12121212"
EDI 0012FE6C ASCII "77777777"

EIP 004025B9 CrackmeC.004025B9

C 1  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
```

The next instruction sets EAX to zero, and then we hit the REPE instruction. In this case, the contents of memory at the address stored in ESI is compared with the contents of the memory address stored in EDI:



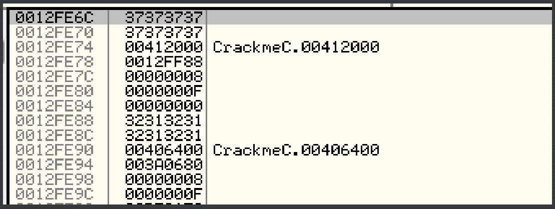
```
004025A8    8BCD            MOV ECX,EBP
004025AA    837C24 48 10    CMP DWORD PTR SS:[ESP+48],10
004025AF    8B7424 34       MOV ESI,DWORD PTR SS:[ESP+34]
004025B3  v 73 04           JNB SHORT CrackmeC.004025B9
004025B5    8D7424 34       LEA ESI,DWORD PTR SS:[ESP+34]
004025B9    33C0            XOR EAX,EAX
004025BB    F3:A6           REPE CMPS BYTE PTR ES:[EDI],BYTE PTR DS:
004025BD  v 74 05           JE SHORT CrackmeC.004025C4
004025BF    1BC0            SBB EAX,EAX
004025C1    83D8 FF         SBB EAX,-1
004025C4    3BC3            CMP EAX,EBX
004025C6  v 75 13           JNZ SHORT CrackmeC.004025DB
004025C8    3BD5            CMP EDX,EBP
004025CA  v 72 0F           JB SHORT CrackmeC.004025DB
004025CC    33C0            XOR EAX,EAX
004025CE    3BD5            CMP EDX,EBP
004025D0    0F95C0          SETNE AL
004025D3    3BC3            CMP EAX,EBX
004025D5  v 0F84 CF020000   JE CrackmeC.004028AA
004025DB    68 48E24000     PUSH CrackmeC.0040E248
004025E0    68 A02F4100     PUSH CrackmeC.00412FA0
004025E5    E8 A6F4FFFF     CALL CrackmeC.00401090

ECX=00000008 (decimal 8.)
DS:[ESI]=stack [0012FE88]=31 ('1')
ES:[EDI]=stack [0012FE6C]=37 ('7')
```

The ECX register is then lowered by one, the compare goes to the next memory location in both EDI and ESI, and the loop will end when ECX = 0. In this case, if you look above you can see that ECX is set to 8 (which happens to be the length of our password) so this loop will go through all 8 digits of our password, each time comparing a digit with a digit from the corresponding location after EDI. But wait…what are we comparing to? If we look at the registers window again we see that EDI points to an address on the stack that has some ASCII 7s in it. Let's see this on the stack. Click on the adddress next to EDI, right click on it and choose "Follow in stack":



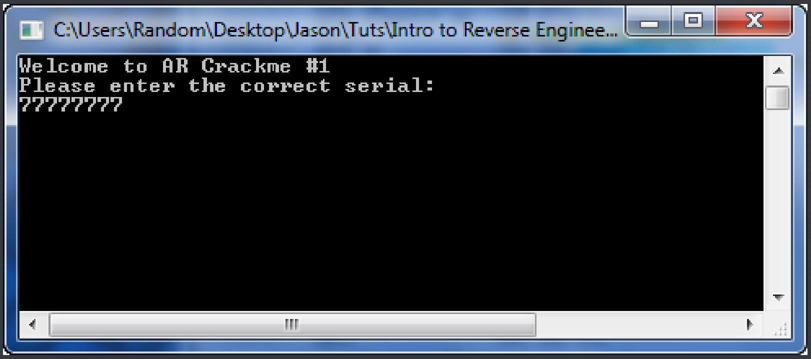
```
EDX 00000008
EBX 00000000
ESP 0012FE54
EBP 00000008
ESI 0012FE88 ASCII "12121212"
EDI 0012

EIP 0040

C 0  ES
P 1  CS
A 0  SS
Z 1  DS
S 0  FS
T 0  GS
D 0
O 0  Las

EFL 0000

ST0 empt
ST1 empt
ST2 empt
ST3 empt
ST4 empt
ST5 empt
ST6 empt
ST7 empt

FST 0000
FCW 027F
```

The stack window then jumps to the referenced address, in this case 0012FE6C. At this address (and we can't help noticing at the next as well) we see a string of "37″s. Looking at our ASCII chart we can see that 37 is equal to "7″ which we saw in the registers window is in the EDI register:



```
0012FE6C   37373737
0012FE70   37373737
0012FE74   00412000  CrackmeC.00412000
0012FE78   0012FF88
0012FE7C   00000008
0012FE80   0000000F
0012FE84   00000000
0012FE88   32313231
0012FE8C   32313231
0012FE90   00406400  CrackmeC.00406400
0012FE94   003A0680
0012FE98   00000008
0012FE9C   0000000F
```

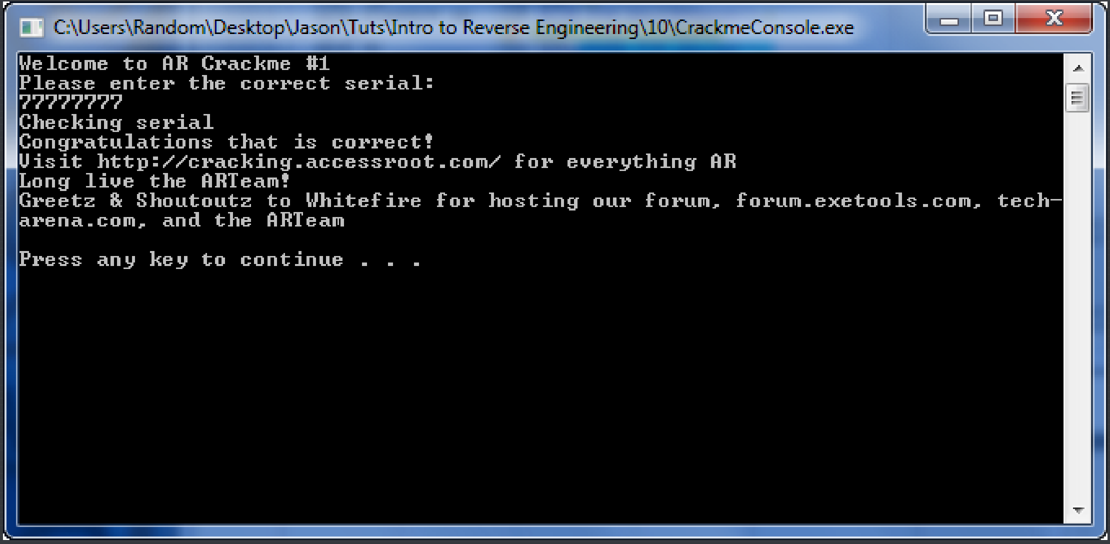
Well, it doesn't take a rocket surgeon to see that our inputted password is being compared with a hardcoded ASCII string of all "7″s. There are exactly 8 of them on the stack (we got lucky that we happened to enter a password that was the same length as the hard-coded password 🙂). These eight "7″s are compared, one by one, with what we entered as a password. If we get through all 8 of them being equal (equal to 7 that is) then we will take the next jump. Hmmmmmm. Our entered password is compared with eights "7″s. This sounds to me like the password could be eights "7″s. Let's restart the app and try it:



C:\Users\Random\Desktop\Jason\Tuts\Intro to Reverse Enginee...

```
Welcome to AR Crackme #1
Please enter the correct serial:
77777777
```

drumroll please….



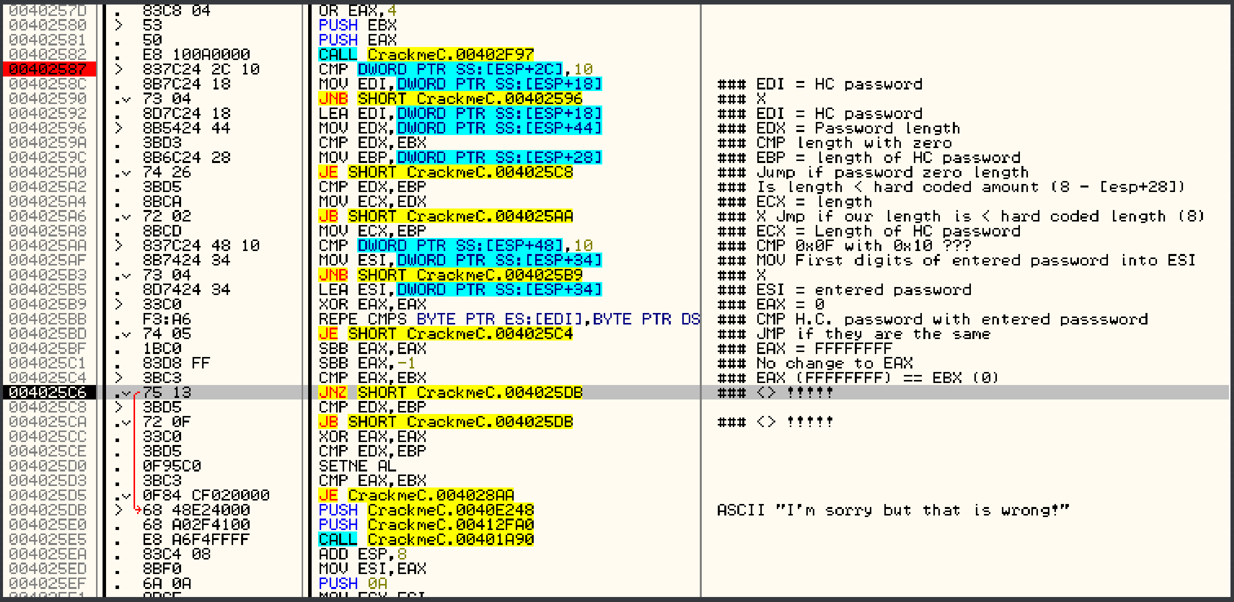
C:\Users\Random\Desktop\Jason\Tuts\Intro to Reverse Engineering\10\CrackmeConsole.exe

```
Welcome to AR Crackme #1
Please enter the correct serial:
77777777
Checking serial
Congratulations that is correct!
Visit http://cracking.accessroot.com/ for everything AR
Long live the ARTeam!
Greetz & Shoutoutz to Whitefire for hosting our forum, forum.exetools.com, tech-
arena.com, and the ARTeam

Press any key to continue . . .
```

And we got it 😀 . So, looking a little farther than we would normally patch has revealed the password, which is frankly far better than patching an app not knowing if it will actually truly patch it or not. This is the benefit of patching at a NOOB level as opposed to a LAME level.

One Last Thing

I just wanted to show you an example of going through code and making comments. Unfortunately, when writing tutorials, you have to understand the app at a pretty deep level. Here is a picture of the core section we were discussing with my comments in it:



```
0040257D  .  83C8 04         OR EAX,4
00402580  >  53              PUSH EBX
00402581  .  50              PUSH EAX
00402582  .  E8 100A0000     CALL CrackmeC.00402F97
00402587  >  837C24 2C 10    CMP DWORD PTR SS:[ESP+2C],10
0040258C  .  8B7C24 18       MOV EDI,DWORD PTR SS:[ESP+18]            ### EDI = HC password
00402590  .v 73 04           JNB SHORT CrackmeC.00402596              ### X
00402592  .  8D7C24 18       LEA EDI,DWORD PTR SS:[ESP+18]            ### EDI = HC password
00402596  >  8B5424 44       MOV EDX,DWORD PTR SS:[ESP+44]            ### EDX = Password length
0040259A  .  3BD3            CMP EDX,EBX                              ### CMP length with zero
0040259C  .  8B6C24 28       MOV EBP,DWORD PTR SS:[ESP+28]            ### EBP = length of HC password
004025A0  .v 74 26           JE SHORT CrackmeC.004025C8               ### Jump if password zero length
004025A2  .  3BD5            CMP EDX,EBP                              ### Is length < hard coded amount (8 - [esp+28])
004025A4  .  8BCA            MOV ECX,EDX                              ### ECX = length
004025A6  .v 72 02           JB SHORT CrackmeC.004025AA               ### X Jmp if our length is < hard coded length (8)
004025A8  .  8BCD            MOV ECX,EBP                              ### ECX = Length of HC password
004025AA  >  837C24 48 10    CMP DWORD PTR SS:[ESP+48],10             ### CMP 0x0F with 0x10 ???
004025AF  .  8B7424 34       MOV ESI,DWORD PTR SS:[ESP+34]            ### MOV First digits of entered password into ESI
004025B3  .v 73 04           JNB SHORT CrackmeC.004025B9              ### X
004025B5  .  8D7424 34       LEA ESI,DWORD PTR SS:[ESP+34]            ### ESI = entered password
004025B9  >  33C0            XOR EAX,EAX                              ### EAX = 0
004025BB  .  F3:A6           REPE CMPS BYTE PTR ES:[EDI],BYTE PTR DS  ### CMP H.C. password with entered passsword
004025BD  .v 74 05           JE SHORT CrackmeC.004025C4               ### JMP if they are the same
004025BF  .  1BC0            SBB EAX,EAX                              ### EAX = FFFFFFFF
004025C1  .  83D8 FF         SBB EAX,-1                               ### No change to EAX
004025C4  >  3BC3            CMP EAX,EBX                              ### EAX (FFFFFFFF) == EBX (0)
004025C6  .v 75 13           JNZ SHORT CrackmeC.004025DB              ### <> !!!!!
004025C8  >  3BD5            CMP EDX,EBP
004025CA  .v 72 0F           JB SHORT CrackmeC.004025DB               ### <> !!!!!
004025CC  .  33C0            XOR EAX,EAX
004025CE  .  3BD5            CMP EDX,EBP
004025D0  .  0F95C0          SETNE AL
004025D3  .  3BC3            CMP EAX,EBX
004025D5  .v 0F84 CF020000   JE CrackmeC.004028AA
004025DB  >  68 48E24000     PUSH CrackmeC.0040E248                   ASCII "I'm sorry but that is wrong!"
004025E0  .  68 A02F4100     PUSH CrackmeC.00412FA0
004025E5  .  E8 A6F4FFFF     CALL CrackmeC.00401A90
004025EA  .  83C4 08         ADD ESP,8
004025ED  .  8BF0            MOV ESI,EAX
004025EF  .  6A 0A           PUSH 0A
004025F1  .  8BCE            MOV ECX,ESI
```

As you can see, a lot goes into understanding the way an app works 😀

-Till next time.

R4ndom