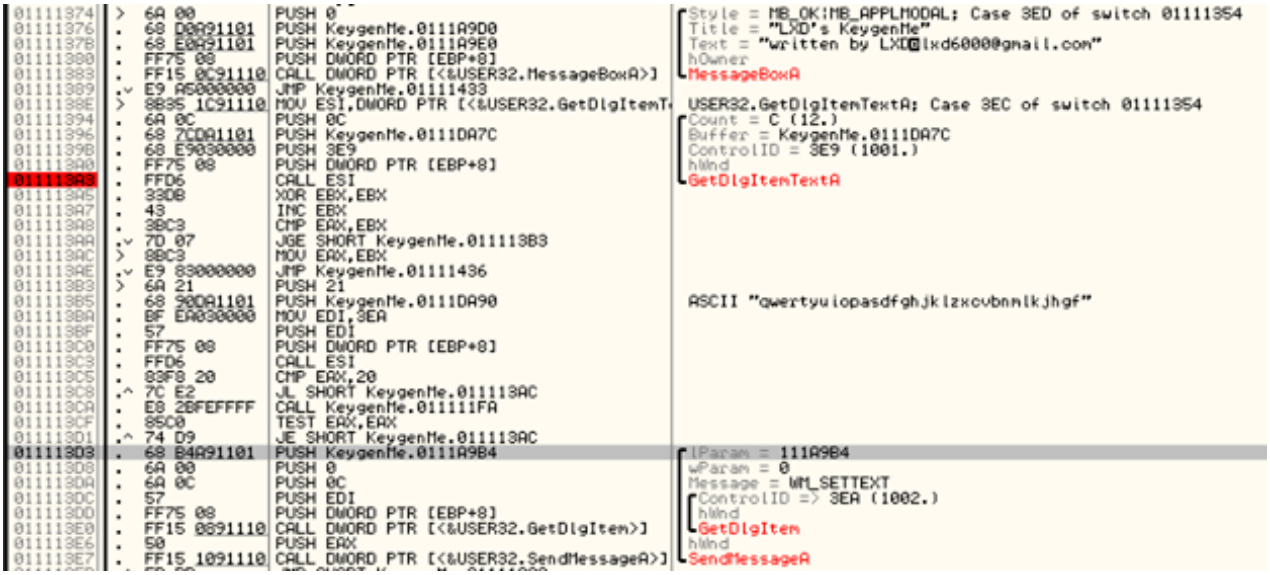
1. When you first launch the program, we see the following screen:

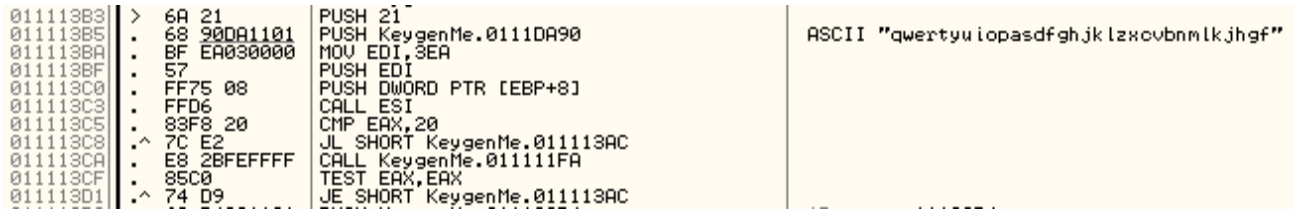
2. Playing around with the program (i.e. typing in something random for "Name" and "Serial" and then clicking "Check") yields no "bad message" saying we entered anything wrong.
3. The rules of the crack-me are simple:
    a. **"No Patching"**
4. After loading it up into OllyDBG, I search for reference strings to find the "good message" as a lead to where I should start.

Address	Disassembly	Text string
01111029	MOV ECX,KeygenMe.0111DA7C	ASCII "sdfg"
0111108F	PUSH KeygenMe.0111A9A8	ASCII "%02x"
01111140	MOV ECX,KeygenMe.0111CDB0	ASCII "d41d8cd98f00b204e9800998ecf8427e"
01111163	PUSH KeygenMe.0111A9B0	ASCII "%1d"
01111250	PUSH KeygenMe.0111DA90	ASCII "qwertyuiopasdfghjklzxcvbnmlkjhgf"
0111129E	POP ESI	(Initial CPU selection)
01111376	PUSH KeygenMe.0111A9D0	ASCII "LXD's KeygenMe"
0111137B	PUSH KeygenMe.0111A9E0	ASCII "written by LXD lxd6000@gmail.com"
01111396	PUSH KeygenMe.0111DA7C	ASCII "sdfg"
011113B5	PUSH KeygenMe.0111DA90	ASCII "qwertyuiopasdfghjklzxcvbnmlkjhgf"
011113D3	PUSH KeygenMe.0111A9B4	ASCII "Congratz you've done it :)"
0111314D	PUSH KeygenMe.011191EC	UNICODE "mscoree.dll"
0111315C	PUSH KeygenMe.011191DC	ASCII "CorExitProcess"
011134C3	PUSH KeygenMe.01119BC4	UNICODE "Runtime Error! Program: "
01113504	PUSH KeygenMe.01119B94	UNICODE "<program name unknown>"
01113545	PUSH KeygenMe.01119B8C	UNICODE "..."
0111355A	PUSH KeygenMe.01119B84	UNICODE " "
0111358B	PUSH KeygenMe.01119B38	UNICODE "Microsoft Visual C++ Runtime Library"
01113F59	PUSH KeygenMe.01119C9C	UNICODE "KERNEL32.DLL"
011141C6	PUSH KeygenMe.01119C9C	UNICODE "KERNEL32.DLL"
011141E7	PUSH KeygenMe.01119CD8	ASCII "FlsAlloc"
011141EF	PUSH KeygenMe.01119CCC	ASCII "FlsGetValue"
011141FC	PUSH KeygenMe.01119CC0	ASCII "FlsSetValue"
01114209	PUSH KeygenMe.01119CB8	ASCII "FlsFree"
01116712	PUSH KeygenMe.0111A0F8	UNICODE "USER32.DLL"
0111672D	PUSH KeygenMe.0111A0EC	ASCII "MessageBoxW"
01116746	PUSH KeygenMe.0111A0DC	ASCII "GetActiveWindow"
01116756	PUSH KeygenMe.0111A0C8	ASCII "GetLastActivePopup"
01116766	PUSH KeygenMe.0111A0AC	ASCII "GetUserObjectInformationW"
0111677F	PUSH KeygenMe.0111A094	ASCII "GetProcessWindowStation"
01118175	PUSH KeygenMe.0111A918	UNICODE "CONOUT$"

5. As you can see, we found our successful message: *"Congratz you've done it :)."* Let's jump over there and see what we can find:



6. On my system, I get pushed to highlight address. This "good message" is part of a message dialog box. Directly above, we have a "GetDlgItemTextA" call and according to the Microsoft documentation, this function retrieves the text of a dialog box. I though this might be of interest, so I set a breakpoint there and enter in a random name and serial and then click "Check."
7. SUCCESS! The breakpoint was hit! After some time of examining what occurs here, I noticed a few things. Firstly, I noticed that the program is checking to make sure our "Name" field has at least a length of 1. Secondly and finally, I noticed two main checks on the serial:



8. Our current entered serial is: "*qwertyuiopasdfghjklzxcvbnmlkjhgf*." I had not stepped in this screenshot above, but KeygenMe.0111DA90 at first has the "Name" string within it. Then

when we CALL ESI, according to OllyDBG, we are making another function call to "*GetDlgItemTextA*." And after such a call returns, our serial is populated in the ASCII field that is finally listed in the screenshot above (i.e. at KeygenMe.0111DA90)

9. The serial has its length compared to 0x20 or 32 in decimal. If it is not 0x20, then it fails.
10. Only one more call until we succeed. Let's step into: KeygenMe.011111FA:

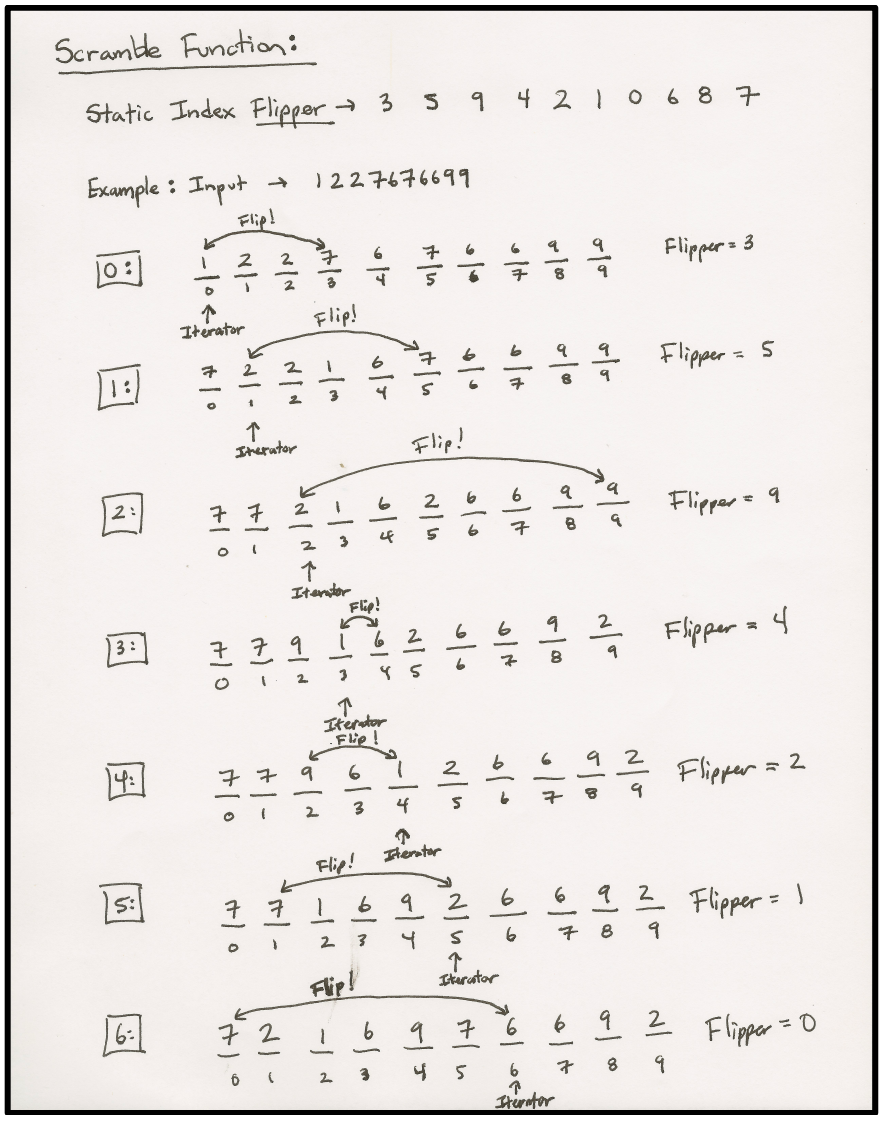
```
011111FA  r$  55           PUSH EBP
011111FB  .   8BEC         MOV EBP,ESP
011111FD  .   83EC 4C      SUB ESP,4C
01111200  .   A1 04C01101  MOV EAX,DWORD PTR [111C004]
01111205  .   33C5         XOR EAX,EBP
01111207  .   8945 FC      MOV DWORD PTR [EBP-4],EAX
0111120A  .   803D 9ADA110 CMP BYTE PTR [111DA9A],2D      serial[10] == '-'
01111211  .v  0F85 A6000000 JNZ KeygenMe.011112BD         Fail if not '-'
01111217  .   803D A5DA110 CMP BYTE PTR [111DAA5],2D      serial[21] == '-'
0111121E  .v  0F85 9900000 JNZ KeygenMe.011112BD          Fail if not '-'
01111224  .   33C0         XOR EAX,EAX
01111226  >   0FBE88 90DA1 MOVSX ECX,BYTE PTR [EAX+111DA90]  #--[Loop1 BEGIN] = Check if serial is all numbers--#
0111122D  .   83F9 30      CMP ECX,30
01111230  .v  7C 05        JL SHORT KeygenMe.01111237
01111232  .   83F9 39      CMP ECX,39
01111235  .v  7E 0A        JLE SHORT KeygenMe.01111241     Is char between 0x30 and 0x39 (i.e. numbers)
01111237  >   83F8 0A      CMP EAX,0A                      Ignore first position (i.e. serial[10]) with '-'
0111123A  .v  74 05        JE SHORT KeygenMe.01111241
0111123C  .   83F8 15      CMP EAX,15                      Ignore second position (i.e. serial[21]) with '-'
0111123F  .v  75 7C        JNZ SHORT KeygenMe.011112BD
01111241  >   40           INC EAX
01111242  .   83F8 20      CMP EAX,20
01111245  .^  7C DF        JL SHORT KeygenMe.01111226      #--[Loop1 END]--#
01111247  .   56           PUSH ESI
01111248  .   8D45 F0      LEA EAX,DWORD PTR [EBP-10]
0111124B  .   50           PUSH EAX
0111124C  .   8D45 E4      LEA EAX,DWORD PTR [EBP-1C]
0111124F  .   50           PUSH EAX
01111250  .   68 90DA1101  PUSH KeygenMe.0111DA90          ASCII "1227676699-1722271113-7925611235"
01111255  .   8D4D D8      LEA ECX,DWORD PTR [EBP-28]      Parameter A => A buffer that will contain the broken up serial
01111258  .   E8 59FFFFFF  CALL KeygenMe.011111B6          [Function] = Breaks up serial into three peices
0111125D  .   8D75 D8      LEA ESI,DWORD PTR [EBP-28]      Serial Part A (i.e. begining portion)
01111260  .   E8 29FFFFFF  CALL KeygenMe.0111118E          [Function Reorder] = Reorders numbers in the serial part
01111265  .   8D75 E4      LEA ESI,DWORD PTR [EBP-1C]      Serial Part B (i.e. middle portion)
01111268  .   E8 21FFFFFF  CALL KeygenMe.0111118E          [Function Reorder]
0111126D  .   8D75 F0      LEA ESI,DWORD PTR [EBP-10]      Serial Part C (i.e. ending portion)
01111270  .   E8 19FFFFFF  CALL KeygenMe.0111118E          [Function Reorder]
01111275  .   E8 67FEFFFF  CALL KeygenMe.011110E1          [Function] = Generate the correct key for the given Name
0111127A  .   8BF0         MOV ESI,EAX
```

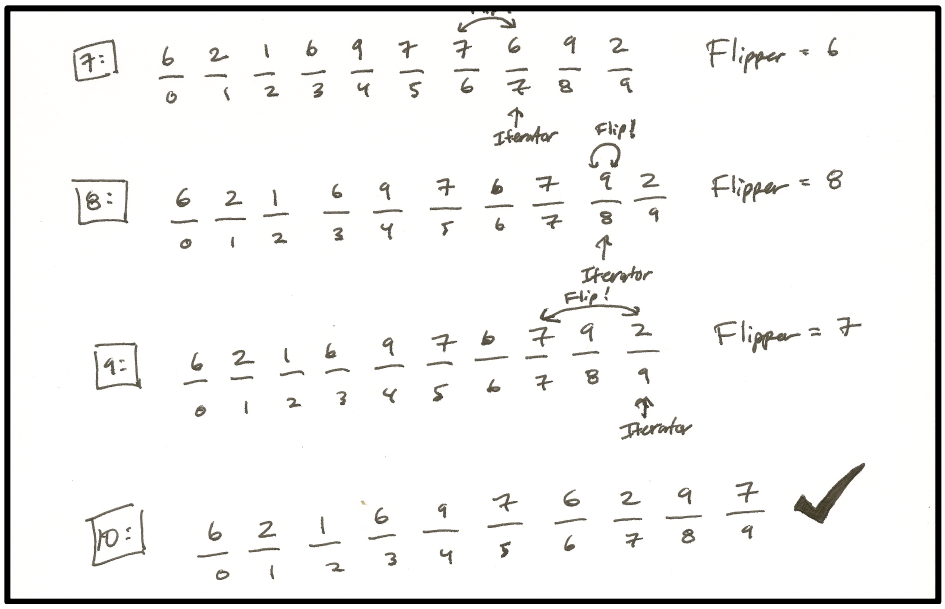
(The comments on the side were put in by me)

11. The first check on the serial is whether we have two instances of the "-" character. Under these constrains, we have three "parts" separated by "-" and that are each 10 characters in length.
12. The next check is checking to make sure our serial only consists of numbers and not alphanumerical letters. If we have any letters in our serial, then we fail automatically.
13. Then the program breaks up our serial on the delimiter "-" and then calls a function on each of these pieces. The function is shown as such:

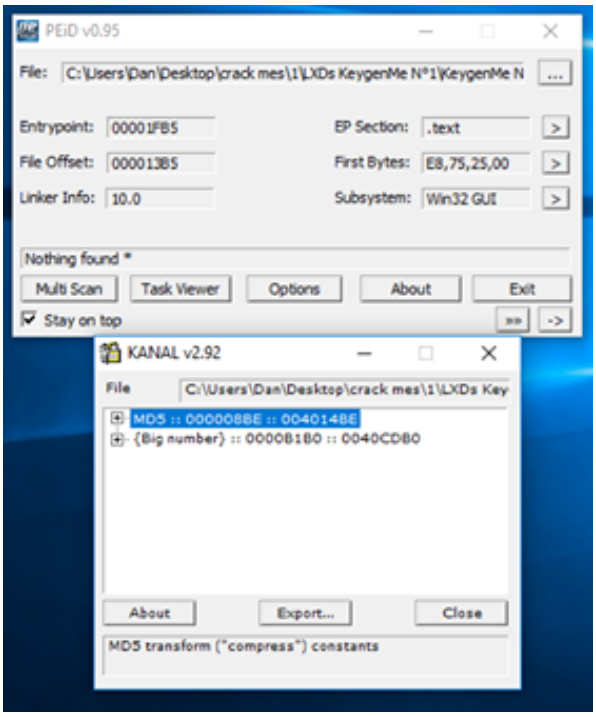
```
0111118D  L.  C3          RET
0111118E  r$  53          PUSH EBX
0111118F  |.  8BD6        MOV EDX,ESI
01111191  |.  B9 D4CD1101  MOV ECX,KeygenMe.0111CDD4     Hard coded scramble key => 3, 5, 9, 4, 2, 1, 0, 6, 8, 7
01111196  |.  57          PUSH EDI
01111197  |>  8B39        MOV EDI,DWORD PTR [ECX]
01111199  |.  8A1C3E      MOV BL,BYTE PTR [ESI+EDI]      Gets byte at a certain position
0111119C  |.  8A02        MOV AL,BYTE PTR [EDX]
0111119E  |.  881A        MOV BYTE PTR [EDX],BL
011111A0  |.  8B39        MOV EDI,DWORD PTR [ECX]
011111A2  |.  83C1 04     ADD ECX,4
011111A5  |.  42          INC EDX
011111A6  |.  88043E      MOV BYTE PTR [ESI+EDI],AL
011111A9  |.  81F9 FCCD110 CMP ECX,KeygenMe.0111CDFC
011111AF  |.^ 7C E6       JL SHORT KeygenMe.01111197
011111B1  |.  5F          POP EDI
011111B2  |.  8BC6        MOV EAX,ESI
011111B4  |.  5B          POP EBX
```

14. A very simple scrambling algorithm is applied to each section of the serial. There are hard-coded swap indices (see above) so this is a very predictable algorithm. Here is an example applied to the following section: "1227676699":





15. Once finished, it goes on to calculate the correct answer, but before I detail that, I must make light of some relevant information:



16. Using a program called PEiD with the [KANAL plugin], it tells me that this Crack-Me is using an MD5 hash algorithm. This information will be very useful in the next few steps!
17. Here is the key calculating function:

```
011110E1  r$  55          PUSH EBP
011110E2  .   8BEC        MOV EBP,ESP
011110E4  .   83EC 20     SUB ESP,20
011110E7  .   A1 04C01101 MOV EAX,DWORD PTR [111C004]
011110EC  .   33C5        XOR EAX,EBP
011110EE  .   8945 FC     MOV DWORD PTR [EBP-4],EAX
011110F1  .   E8 0AFFFFFF CALL KeygenMe.01111000          [Function] = Calculate MD5 Hash on Name (from KANAL) (eax = hash)
011110F6  .   50          PUSH EAX
011110F7  .   8945 E0     MOV DWORD PTR [EBP-20],EAX
011110FA  .   E8 94780000 CALL KeygenMe.01118993          [Function] = Converts hash to upper case letters
011110FF  .   8365 E4 00  AND DWORD PTR [EBP-1C],0
01111103  .   8365 E8 00  AND DWORD PTR [EBP-18],0
01111107  .   59          POP ECX
01111108  .   53          PUSH EBX
01111109  .   56          PUSH ESI
0111110A  .   57          PUSH EDI
0111110B  >   8B45 E8     MOV EAX,DWORD PTR [EBP-18]      #--[Loop1 BEGIN] = Obtains a unqiue byte bound to CPU--#
0111110E  .   33C9        XOR ECX,ECX
01111110  .   0FA2        CPUID                           Infromation (i.e. numbers) about CPU used to find unique number
01111112  .   8D75 EC     LEA ESI,DWORD PTR [EBP-14]
01111115  .   8906        MOV DWORD PTR [ESI],EAX
01111117  .   8B45 E4     MOV EAX,DWORD PTR [EBP-1C]
0111111A  .   895E 04     MOV DWORD PTR [ESI+4],EBX
0111111D  .   894E 08     MOV DWORD PTR [ESI+8],ECX
01111120  .   8956 0C     MOV DWORD PTR [ESI+C],EDX
01111123  .   0345 EC     ADD EAX,DWORD PTR [EBP-14]
01111126  .   FF45 E8     INC DWORD PTR [EBP-18]
01111129  .   837D E8 05  CMP DWORD PTR [EBP-18],5        Loop for 5 turns
0111112D  .   8945 E4     MOV DWORD PTR [EBP-1C],EAX
01111130  .^  7C D9       JL SHORT KeygenMe.0111110B      #--[Loop1 END]--#
01111132  .   8B75 E0     MOV ESI,DWORD PTR [EBP-20]
01111135  .   33C9        XOR ECX,ECX
01111137  >   300431      XOR BYTE PTR [ECX+ESI],AL       #--[Loop2 BEGIN] = XORs all 20 bytes of hash with unique byte above--#
0111113A  .   41          INC ECX
0111113B  .   83F9 20     CMP ECX,20
0111113E  .^  7C F7       JL SHORT KeygenMe.01111137      #--[Loop3 END]--#
01111140  .   B9 B0CD1101 MOV ECX,KeygenMe.0111CDB0       ASCII "d41d8cd98f00b204e9800998ecf8427e"
01111145  .   6A 20       PUSH 20
01111147  .   8BC6        MOV EAX,ESI
01111149  .   2BCE        SUB ECX,ESI
0111114B  .   5F          POP EDI
0111114C  >   8A1401      MOV DL,BYTE PTR [ECX+EAX]       #--[Loop3 BEGIN] = XOR xored hash with secret number above (i.e. @0111CDB0)
0111114F  .   3010        XOR BYTE PTR [EAX],DL
01111151  .   40          INC EAX
01111152  .   4F          DEC EDI
01111153  .^  75 F7       JNZ SHORT KeygenMe.0111114C     #--[Loop3 END]--#
01111155  >   0FBE0437    MOVSX EAX,BYTE PTR [EDI+ESI]    #--[Loop4 BEGIN] = For each byte in new xored hash, uses upper nipple to ascii as next number in seial--#
01111159  .   85C0        TEST EAX,EAX
0111115B  .v  79 02       JNS SHORT KeygenMe.0111115F
0111115D  .   F7D8        NEG EAX                         Use positive version of byte (i.e. sign bit = 0)
0111115F  >   50          PUSH EAX
01111160  .   8D45 F0     LEA EAX,DWORD PTR [EBP-10]
01111163  .   68 B0A91101 PUSH KeygenMe.0111A9B0          ASCII "%1d"
01111168  .   50          PUSH EAX
01111169  .   E8 5F0B0000 CALL KeygenMe.01111CCD          [Function] = Get upper ascii digit of next byte
0111116E  .   8A45 F0     MOV AL,BYTE PTR [EBP-10]
01111171  .   83C4 0C     ADD ESP,0C
01111174  .   880437      MOV BYTE PTR [EDI+ESI],AL
01111177  .   47          INC EDI
01111178  .   83FF 20     CMP EDI,20
0111117B  .^  7C D8       JL SHORT KeygenMe.01111155      #--[Loop4 END]--#
0111117D  .   8B4D FC     MOV ECX,DWORD PTR [EBP-4]
01111180  .   5F          POP EDI
01111181  .   8BC6        MOV EAX,ESI                     EAX = Correct Serial
01111183  .   5E          POP ESI
01111184  .   33CD        XOR ECX,EBP
01111186  .   5B          POP EBX
01111187  .   E8 320B0000 CALL KeygenMe.01111CBE
0111118C  .   C9          LEAVE
0111118D  L.  C3          RET
```

(Again the comments were generated by me)

18. At address 0x011110F1, it makes a call to a function: KeygenMe.01111000. Instead of reverse engineering every single function, let's treat this as a black box and see what it returns:

```
Registers (FPU)                                  <    <    <
EAX 00DF3CB8 ASCII "97c8e6d0d14f4e242c3c37af68cc376c"
ECX E28DCEA2
EDX 00000000
EBX 00000001
ESP 0019F704
EBP 0019F724
ESI 0019F778 ASCII "1976512235"
EDI 000003EA
```

19. Given that we know this Crack-Me is using MD5, the output in EAX looks very much like an MD5 hash. Looking back, one can

notice that one of the parameters for this function is the "Name" we entered in early. So, this function most likely hashed the name!

20. Next, we arrive at "Loop1." This loop is using some system dependent information through the CPUID instruction to generate a unique byte that would be different from computer to computer.
21. It uses this unique byte and XORs it with the md5 hash of the "Name" in loop2.
22. Next, it takes a pre-defined constant ("d41d8cd98f… 427e") and hashes the result of step 21 with this constant in loop3.
23. And finally, for each byte in this new hashed value (from step 22), a conversion to ASCII takes place and then the leading digit is plucked to become the next number in the serial:

- Example: 0xA → 10, therefore "1" is chosen as the next number.

24. After all that, for the name "Dan," I get the following result:

"62169762972171237121211976512235"

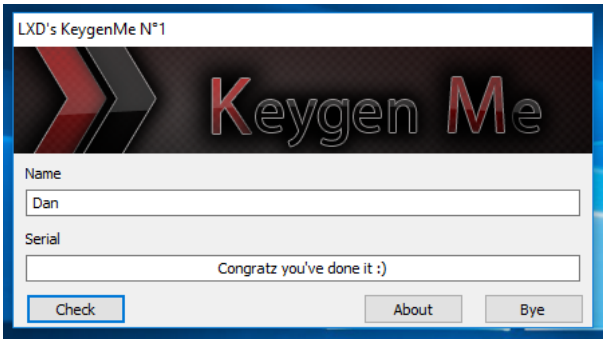
Now let's match the correct format with the dashes (i.e. "-"). We do have to throw away two numbers though:

"6216976297-1712371212-1976512235"

However, this is not the serial, because once we return from this function, we compare this value with our scrambled up serial. Therefore, to get a proper serial, we must apply the scramble algorithm above **in reverse** on this "serial" in order to get the correct answer! In must case, in my case, the correct serial is:

"1227676699-1722271113-7925611235"

25. When I enter in this serial for the name "Dan," I get the following successful screen:



Finished.

Final Remarks:

I did not expect this Crack-Me to take me well over a full day to figure out. Figuring out that MD5 was being used in this Crack-Me was critical for making a speedy solution, but I found out the hard when I had to reverse part of the algorithm to see what the program was doing (not discussed above). Through further research, I came across that PEiD program, which made my life much easier. Creating a keygen program for this Crack-Me is not difficult. The only major thing to keep in mind is generating that unique byte with Loop1 above (i.e. the loop that used the CPUID instruction).