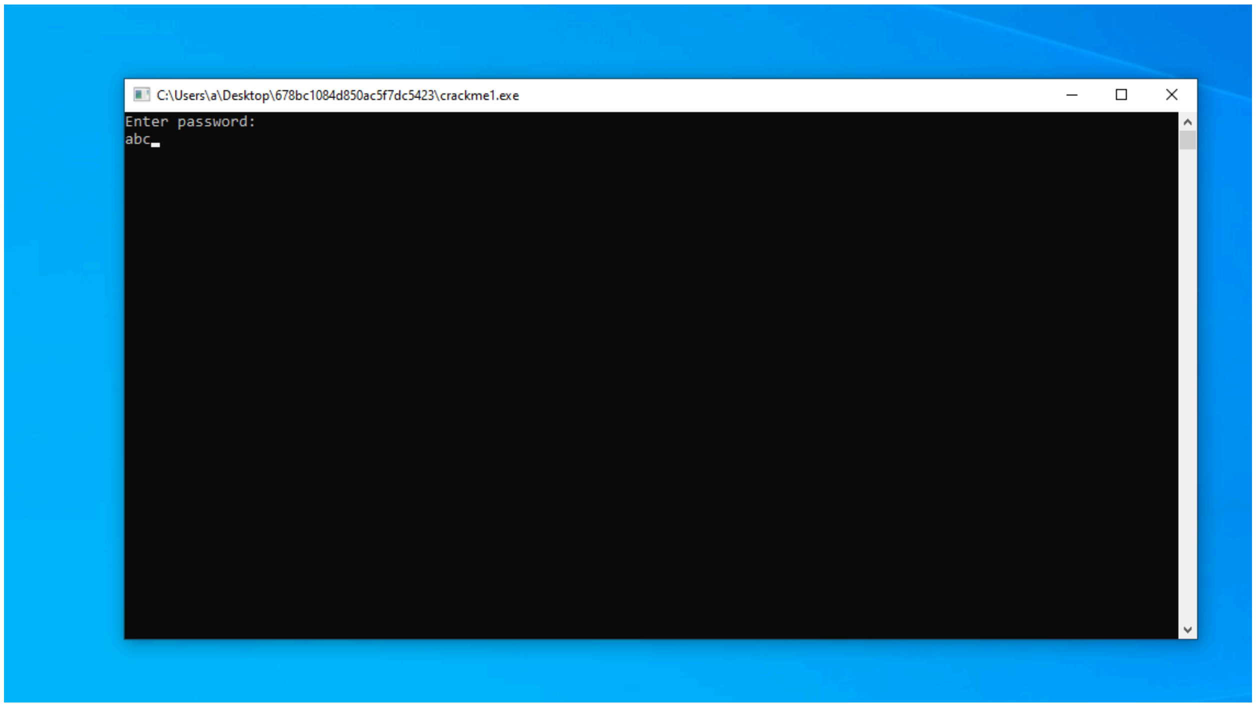
without fantasy

https://crackmes.one/crackme/678bc1084d850ac5f7dc5423

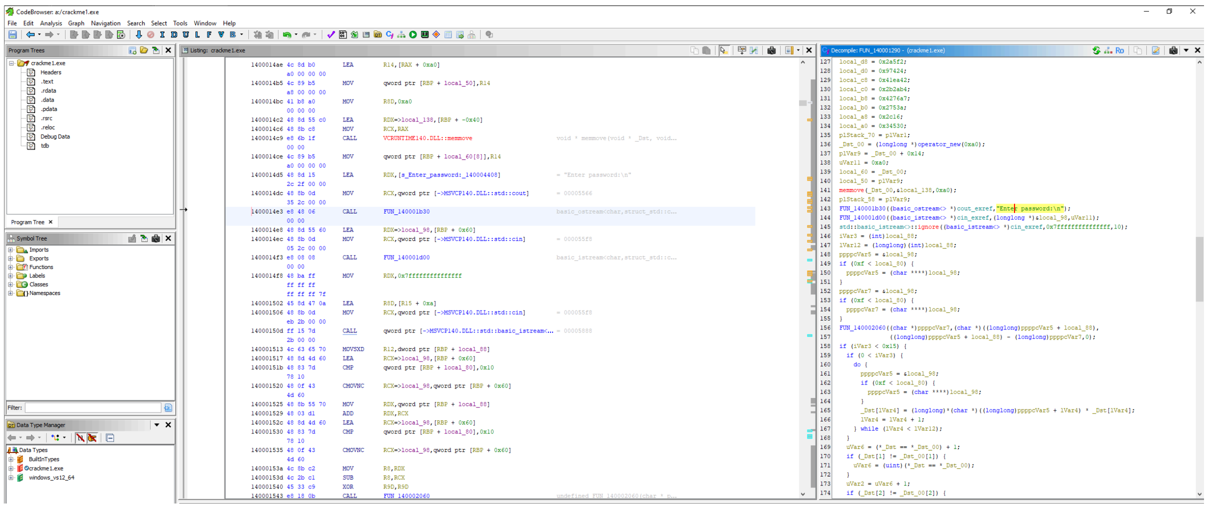
For this reverse engineering challenge, I used a Windows 10 VM, Ghidra for static analysis, and x64dbg as a debugger for dynamic analysis.

Upon download, the first thing I wanted to do was see how I can interact with the program, so I ran it. The program seemed to ask for a password, and upon any user input it just closed the program.



```
Enter password:
abc_
```

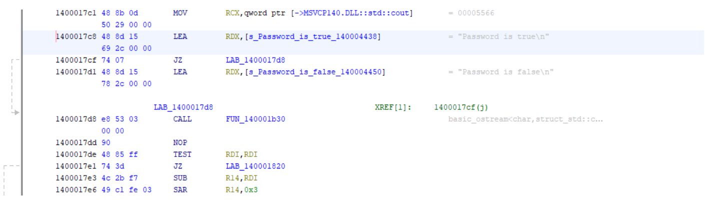
Now I decided to load the program into ghidra. I know I was looking for the string "Enter Password" because the program would need to compare my user input at some point. This seemed like a good place to start.



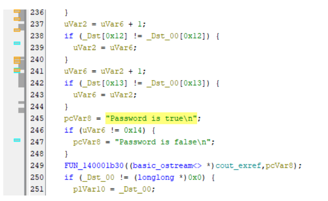
```
1400014ae 4c 8d b0      LEA      R14,[RAX + 0xa0]
          a0 00 00 00
1400014b5 4c 89 b5      MOV      qword ptr [RBP + local_50],R14
          a8 00 00 00
1400014bc 41 b8 a0      MOV      R8D,0xa0
          00 00 00
1400014c2 48 8d 55 c0   LEA      RDX=>local_138,[RBP + -0x40]
1400014c6 48 8b c8      MOV      RCX,RAX
1400014c9 e8 6b 1f      CALL     VCRUNTIME140.DLL::memmove                   void * memmove(void * _Dst, void...
          00 00
1400014ce 4c 89 b5      MOV      qword ptr [RBP + local_60[8]],R14
          a0 00 00 00
1400014d5 48 8d 15      LEA      RDX,[s_Enter_password:_140004408]           = "Enter password:\n"
          2c 2f 00 00
1400014dc 48 8b 0d      MOV      RCX,qword ptr [->MSVCP140.DLL::std::cout]   = 00005566
          35 2c 00 00
1400014e3 e8 48 06      CALL     FUN_140001b30                               basic_ostream<char,struct_std::c...
          00 00
1400014e8 48 8d 55 60   LEA      RDX=>local_98,[RBP + 0x60]
1400014ec 48 8b 0d      MOV      RCX,qword ptr [->MSVCP140.DLL::std::cin]    = 000055f8
          05 2c 00 00
1400014f3 e8 08 08      CALL     FUN_140001d00                               basic_istream<char,struct_std::c...
          00 00
1400014f8 48 ba ff      MOV      RDX,0x7fffffffffffffff
          ff ff ff
          ff ff ff 7f
140001502 45 8d 47 0a   LEA      R8D,[R15 + 0xa]
140001506 48 8b 0d      MOV      RCX,qword ptr [->MSVCP140.DLL::std::cin]    = 000055f8
          eb 2b 00 00
14000150d ff 15 7d      CALL     qword ptr [->MSVCP140.DLL::std::basic_istream<... = 00005888
          2b 00 00
140001513 4c 63 65 70   MOVSXD   R12,dword ptr [RBP + local_88]
140001517 48 8d 4d 60   LEA      RCX=>local_98,[RBP + 0x60]
14000151b 48 83 7d      CMP      qword ptr [RBP + local_80],0x10
          78 10
140001520 48 0f 43      CMOVNC   RCX=>local_98,qword ptr [RBP + 0x60]
          4d 60
140001525 48 8b 55 70   MOV      RDX,qword ptr [RBP + local_88]
140001529 48 03 d1      ADD      RDX,RCX
14000152c 48 8d 4d 60   LEA      RCX=>local_98,[RBP + 0x60]
140001530 48 83 7d      CMP      qword ptr [RBP + local_80],0x10
          78 10
140001535 48 0f 43      CMOVNC   RCX=>local_98,qword ptr [RBP + 0x60]
          4d 60
14000153a 4c 8b c2      MOV      R8,RDX
14000153d 4c 2b c1      SUB      R8,RCX
140001540 45 33 c9      XOR      R9D,R9D
140001543 e8 18 0b      CALL     FUN_140002060                               undefined FUN_140002060(char * p...
```

```
local_d8 = 0x2a5f2;
local_d0 = 0x97424;
local_c8 = 0x41ea42;
local_c0 = 0x2b2ab4;
local_b8 = 0x4276a7;
local_b0 = 0x2753a;
local_a8 = 0x2c16;
local_a0 = 0x34530;
plStack_70 = plVar1;
_Dst_00 = (longlong *)operator_new(0xa0);
plVar9 = _Dst_00 + 0x14;
uVar11 = 0xa0;
local_60 = _Dst_00;
local_50 = plVar9;
memmove(_Dst_00,&local_138,0xa0);
plStack_58 = plVar9;
FUN_140001b30((basic_ostream<> *)cout_exref,"Enter password:\n");
FUN_140001d00((basic_istream<> *)cin_exref,(longlong *)&local_98,uVar11);
std::basic_istream<>::ignore((basic_istream<> *)cin_exref,0x7fffffffffffffff,10);
iVar3 = (int)local_88;
lVar12 = (longlong)(int)local_88;
ppppcVar5 = &local_98;
if (0xf < local_80) {
  ppppcVar5 = (char ****)local_98;
}
ppppcVar7 = &local_98;
if (0xf < local_80) {
  ppppcVar7 = (char ****)local_98;
}
FUN_140002060((char *)ppppcVar7,(char *)((longlong)ppppcVar5 + local_88),
              ((longlong)ppppcVar5 + local_88) - (longlong)ppppcVar7,0);
if (iVar3 < 0x15) {
  if (0 < iVar3) {
    do {
      ppppcVar5 = &local_98;
      if (0xf < local_80) {
        ppppcVar5 = (char ****)local_98;
      }
      _Dst[lVar4] = (longlong)*(char *)((longlong)ppppcVar5 + lVar4) * _Dst[lVar4];
      lVar4 = lVar4 + 1;
    } while (lVar4 < lVar12);
  }
  uVar6 = (*_Dst == *_Dst_00) + 1;
  if (_Dst[1] != _Dst_00[1]) {
    uVar6 = (uint)(*_Dst == *_Dst_00);
  }
  uVar2 = uVar6 + 1;
  if (_Dst[2] != _Dst_00[2]) {
```

Further into the same function, I was able to find a code block that checked if the password was true of false, so I decided to work backwards from there and I knew I was looking in the correct area.

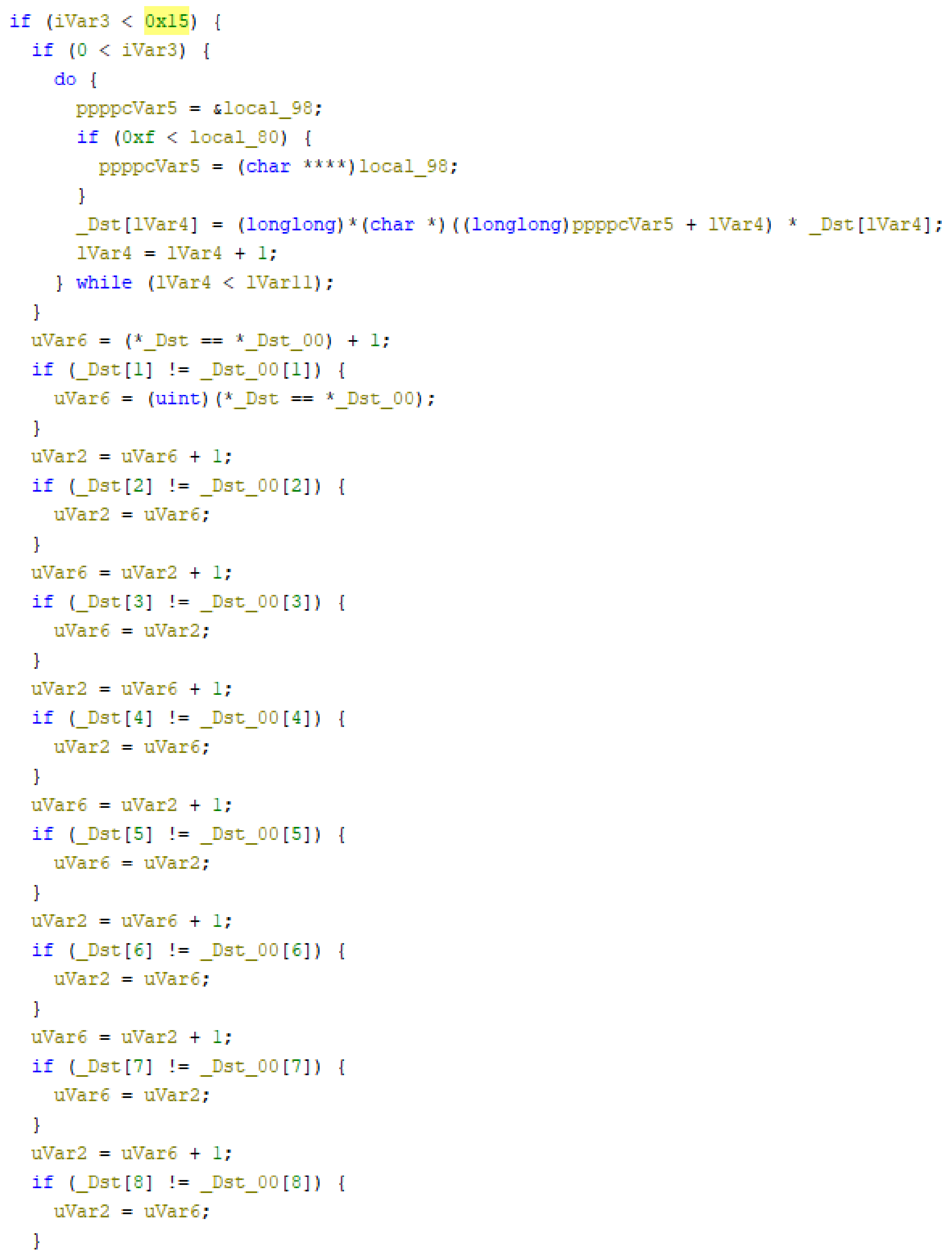


```
1400017c1 48 8b 0d      MOV      RCX,qword ptr [->MSVCP140.DLL::std::cout]   = 00005566
          50 29 00 00
1400017c8 48 8d 15      LEA      RDX,[s_Password_is_true_140004438]          = "Password is true\n"
          69 2c 00 00
1400017cf 74 07         JZ       LAB_1400017d8
1400017d1 48 8d 15      LEA      RDX,[s_Password_is_false_140004450]         = "Password is false\n"
          78 2c 00 00

                     LAB_1400017d8                         XREF[1]:     1400017cf(j)
1400017d8 e8 53 03      CALL     FUN_140001b30                               basic_ostream<char,struct_std::c...
          00 00
1400017dd 90            NOP
1400017de 48 85 ff      TEST     RDI,RDI
1400017e1 74 3d         JZ       LAB_140001820
1400017e3 4c 2b f7      SUB      R14,RDI
1400017e6 49 c1 fe 03   SAR      R14,0x3
```



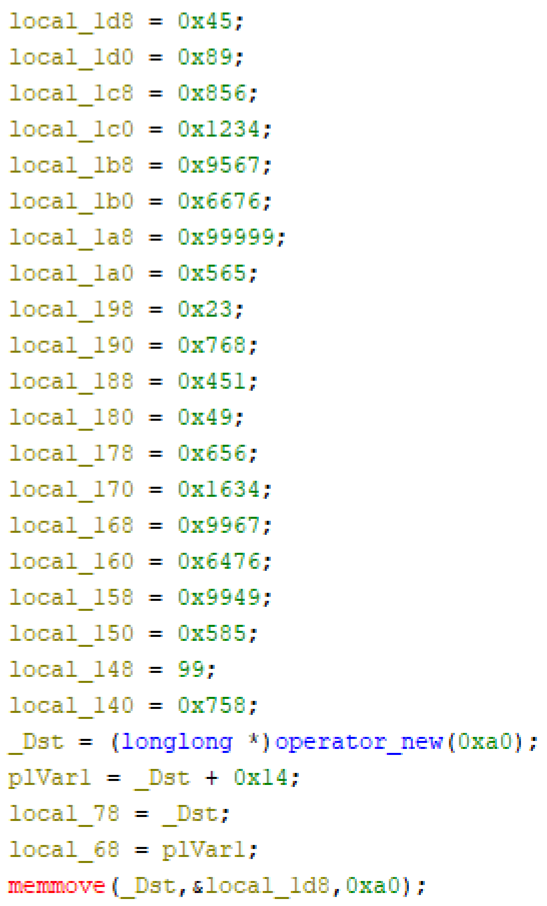
```
  }
  uVar2 = uVar6 + 1;
  if (_Dst[0x12] != _Dst_00[0x12]) {
    uVar2 = uVar6;
  }
  uVar6 = uVar2 + 1;
  if (_Dst[0x13] != _Dst_00[0x13]) {
    uVar6 = uVar2;
  }
  pcVar8 = "Password is true\n";
  if (uVar6 != 0x14) {
    pcVar8 = "Password is false\n";
  }
  FUN_140001b30((basic_ostream<> *)cout_exref,pcVar8);
  if (_Dst_00 != (longlong *)0x0) {
    plVar10 = _Dst_00;
```

In this function I was able to see that there was some sort of comparison happening 20 times between indices of Dst and Dst_00. This would presumably mean that there were 20 characters being compared and the loop exits at 21(0x15). This would lead me to believe that firstly the password is 20 characters long, and secondly that either Dst or Dst_00 contains the password I am looking for while the other contains the user input.

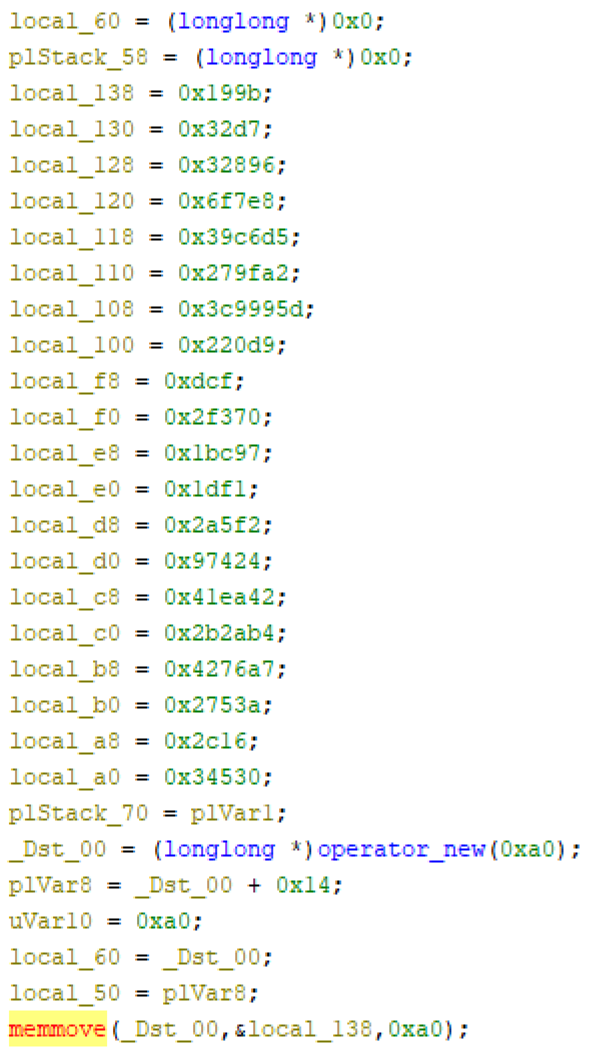


```
if (iVar3 < 0x15) {
  if (0 < iVar3) {
    do {
      ppppcVar5 = &local_98;
      if (0xf < local_80) {
        ppppcVar5 = (char ****)local_98;
      }
      _Dst[lVar4] = (longlong)*(char *)((longlong)ppppcVar5 + lVar4) * _Dst[lVar4];
      lVar4 = lVar4 + 1;
    } while (lVar4 < lVar11);
  }
  uVar6 = (*_Dst == *_Dst_00) + 1;
  if (_Dst[1] != _Dst_00[1]) {
    uVar6 = (uint)(*_Dst == *_Dst_00);
  }
  uVar2 = uVar6 + 1;
  if (_Dst[2] != _Dst_00[2]) {
    uVar2 = uVar6;
  }
  uVar6 = uVar2 + 1;
  if (_Dst[3] != _Dst_00[3]) {
    uVar6 = uVar2;
  }
  uVar2 = uVar6 + 1;
  if (_Dst[4] != _Dst_00[4]) {
    uVar2 = uVar6;
  }
  uVar6 = uVar2 + 1;
  if (_Dst[5] != _Dst_00[5]) {
    uVar6 = uVar2;
  }
  uVar2 = uVar6 + 1;
  if (_Dst[6] != _Dst_00[6]) {
    uVar2 = uVar6;
  }
  uVar6 = uVar2 + 1;
  if (_Dst[7] != _Dst_00[7]) {
    uVar6 = uVar2;
  }
  uVar2 = uVar6 + 1;
  if (_Dst[8] != _Dst_00[8]) {
    uVar2 = uVar6;
  }
```

At the start of the function, I can see that Dst is being initialized with the following values. All but one seem to be hexadecimal values.


```
local_1d8 = 0x45;
local_1d0 = 0x89;
local_1c8 = 0x856;
local_1c0 = 0x1234;
local_1b8 = 0x9567;
local_1b0 = 0x6676;
local_1a8 = 0x99999;
local_1a0 = 0x565;
local_198 = 0x23;
local_190 = 0x768;
local_188 = 0x451;
local_180 = 0x49;
local_178 = 0x656;
local_170 = 0x1634;
local_168 = 0x9967;
local_160 = 0x6476;
local_158 = 0x9949;
local_150 = 0x585;
local_148 = 99;
local_140 = 0x758;
_Dst = (longlong *)operator_new(0xa0);
plVar1 = _Dst + 0x14;
local_78 = _Dst;
local_68 = plVar1;
memmove(_Dst,&local_1d8,0xa0);
```

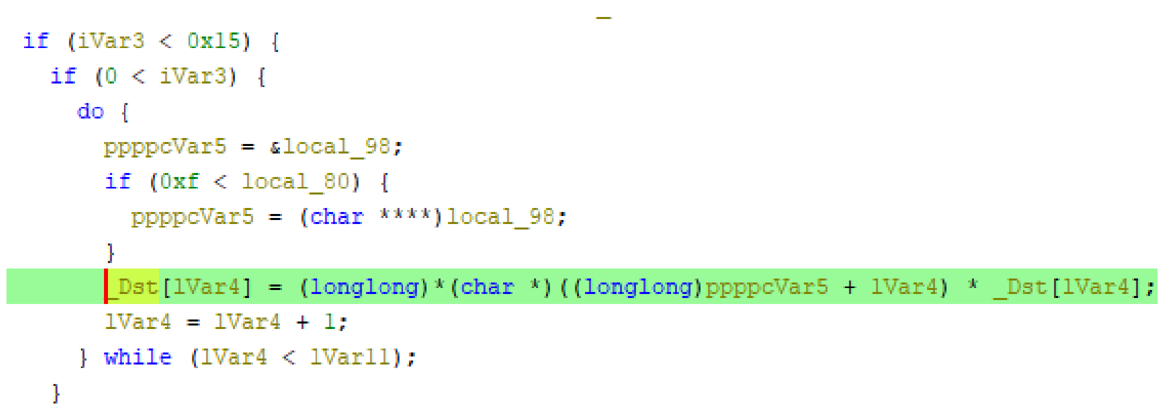
Right after, I can see that Dst_00 is also being initialized with the following values.



```
local_60 = (longlong *)0x0;
plStack_58 = (longlong *)0x0;
local_138 = 0x199b;
local_130 = 0x32d7;
local_128 = 0x32896;
local_120 = 0x6f7e8;
local_118 = 0x39c6d5;
local_110 = 0x279fa2;
local_108 = 0x3c9995d;
local_100 = 0x220d9;
local_f8 = 0xdcf;
local_f0 = 0x2f370;
local_e8 = 0x1bc97;
local_e0 = 0x1df1;
local_d8 = 0x2a5f2;
local_d0 = 0x97424;
local_c8 = 0x41ea42;
local_c0 = 0x2b2ab4;
local_b8 = 0x4276a7;
local_b0 = 0x2753a;
local_a8 = 0x2c16;
local_a0 = 0x34530;
plStack_70 = plVar1;
_Dst_00 = (longlong *)operator_new(0xa0);
plVar8 = _Dst_00 + 0x14;
uVar10 = 0xa0;
local_60 = _Dst_00;
local_50 = plVar8;
memmove(_Dst_00,&local_138,0xa0);
```

When I move back to the comparison loop that checks the 20 characters, there is a block of code that seems to obfuscate Dst values using multiplication and the user input. It then compares the index to Dst_00 index. this means that Dst_00 is initialized with the obfuscated password.

```
_Dst[i] = userInput[i] * _Dst[i]
```



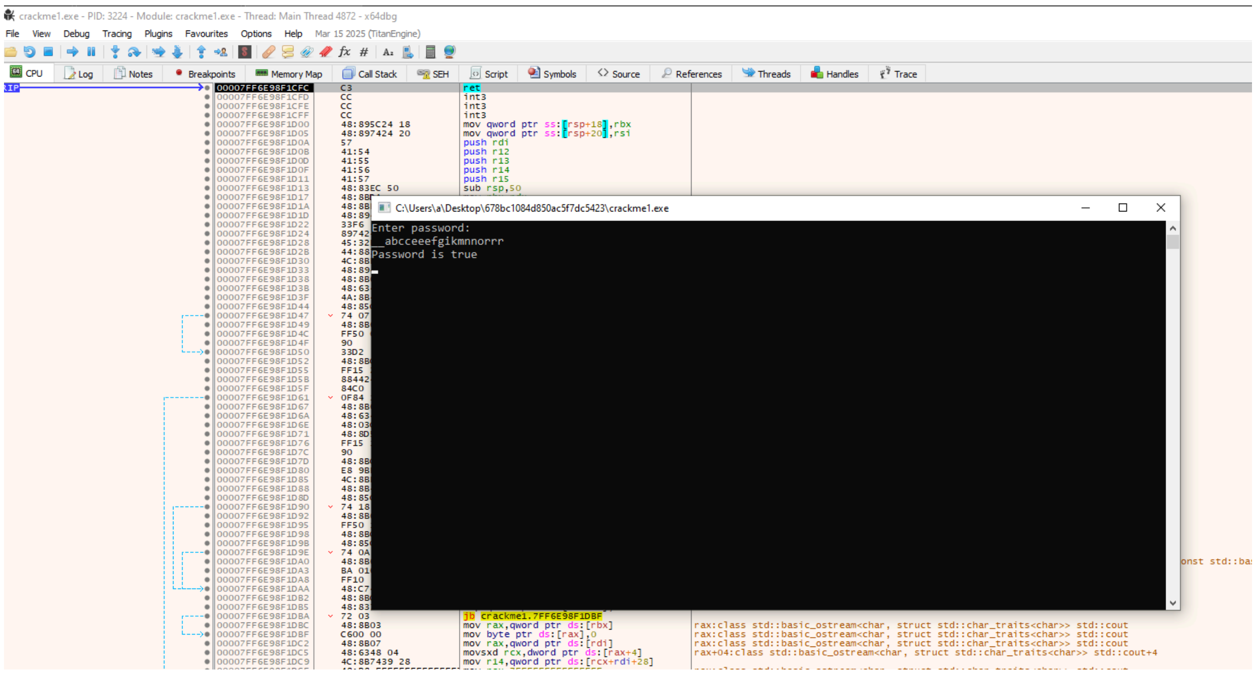
```
if (iVar3 < 0x15) {
  if (0 < iVar3) {
    do {
      ppppcVar5 = &local_98;
      if (0xf < local_80) {
        ppppcVar5 = (char ****)local_98;
      }
      _Dst[lVar4] = (longlong)*(char *)((longlong)ppppcVar5 + lVar4) * _Dst[lVar4];
      lVar4 = lVar4 + 1;
    } while (lVar4 < lVar11);
  }
```

Therefore, we will need to reverse the math to deobfuscate the real password

```
finalVal[i] = userInput[i] * _Dst[i]
# turns to
userInput[i] = finalVal[i] / _Dst[i]
# in this case finalVal == _Dst_00
userInput[i] = _Dst_00[i] / _Dst[i];

_Dst_00 = [
    0x199b, 0x32d7, 0x32896, 0x6f7e8, 0x39c6d5, 0x279fa2, 0x3c9995d, 0x220
    0xdcf, 0x2f370, 0x1bc97, 0x1df1, 0x2a5f2, 0x97424, 0x41ea42, 0x2b2ab4,
    0x4276a7, 0x2753a, 0x2c16, 0x34530
]

_Dst = [
    0x45, 0x89, 0x856, 0x1234, 0x9567, 0x6676, 0x99999, 0x565, 0x23, 0x768,
    0x451, 0x49, 0x656, 0x1634, 0x9967, 0x6476, 0x9949, 0x585, 99, 0x758
]

0x199b / 0x45 = 0x5f = '_'
0x32d7 / 0x89 = 0x5f = '_'
0x32896 / 0x856 = 0x61 = 'a'
0x6f7e8 / 0x1234 = 0x62 = 'b'
0x39c6d5 / 0x9567 = 0x63 = 'c'
0x279fa2 / 0x6676 = 0x63 = 'c'
```

```
0x3c9995d / 0x99999 = 0x65 = 'e'
0x220d9 / 0x565 = 0x65 = 'e'
0xdcf / 0x23 = 0x65 = 'e'
0x2f370 / 0x768 = 0x66 = 'f'
0x1bc97 / 0x451 = 0x67 = 'g'
0x1df1 / 0x49 = 0x69 = 'i'
0x2a5f2 / 0x656 = 0x6b = 'k'
0x97424 / 0x1634 = 0x6d = 'm'
0x41ea42 / 0x9967 = 0x6e = 'n'
0x2b2ab4 / 0x6476 = 0x6e = 'n'
0x4276a7 / 0x9949 = 0x6f = 'o'
0x2753a / 0x585 = 0x72 = 'r'
0x2c16 / 99 = 0x72 = 'r' #this one _Dst[i] is set to decimal 99, not hex
0x34530 / 0x758 = 0x72 = 'r'
```

After the math is done, we get '__abcceeefgikmnnorrr' as the password. Now to check the results. Using x64dbg as the debugger, we can see the success message by setting a breakpoint before the program closes automatically.



crackme1.exe - PID: 3224 - Module: crackme1.exe - Thread: Main Thread 4872 - x64dbg
File View Debug Tracing Plugins Favourites Options Help Mar 15 2025 (TitanEngine)
CPU Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source References Threads Handles Trace

```
00007FF6E98F1CFC   C3              ret
00007FF6E98F1CFD   CC              int3
00007FF6E98F1CFE   CC              int3
00007FF6E98F1CFF   CC              int3
00007FF6E98F1D00   48:895C24 18    mov qword ptr ss:[rsp+18],rbx
00007FF6E98F1D05   48:897424 20    mov qword ptr ss:[rsp+20],rsi
00007FF6E98F1D0A   57              push rdi
00007FF6E98F1D0B   41:54           push r12
00007FF6E98F1D0D   41:55           push r13
00007FF6E98F1D0F   41:56           push r14
00007FF6E98F1D11   41:57           push r15
00007FF6E98F1D13   48:83EC 50      sub rsp,50
...
00007FF6E98F1DBA   72 03           jb crackme1.7FF6E98F1DBF
00007FF6E98F1DBC   48:8B03         mov rax,qword ptr ds:[rbx]              rax:class std::basic_ostream<char, struct std::char_traits<char>> std::cout
00007FF6E98F1DBF   C600 00         mov byte ptr ds:[rax],0                 rax:class std::basic_ostream<char, struct std::char_traits<char>> std::cout
00007FF6E98F1DC2   48:8B07         mov rax,qword ptr ds:[rdi]              rax:class std::basic_ostream<char, struct std::char_traits<char>> std::cout
00007FF6E98F1DC5   48:6348 04      movsxd rcx,dword ptr ds:[rax+4]         rax+04:class std::basic_ostream<char, struct std::char_traits<char>> std::cout+4
00007FF6E98F1DC9   4C:8B7439 28    mov r14,qword ptr ds:[rcx+rdi+28]
```

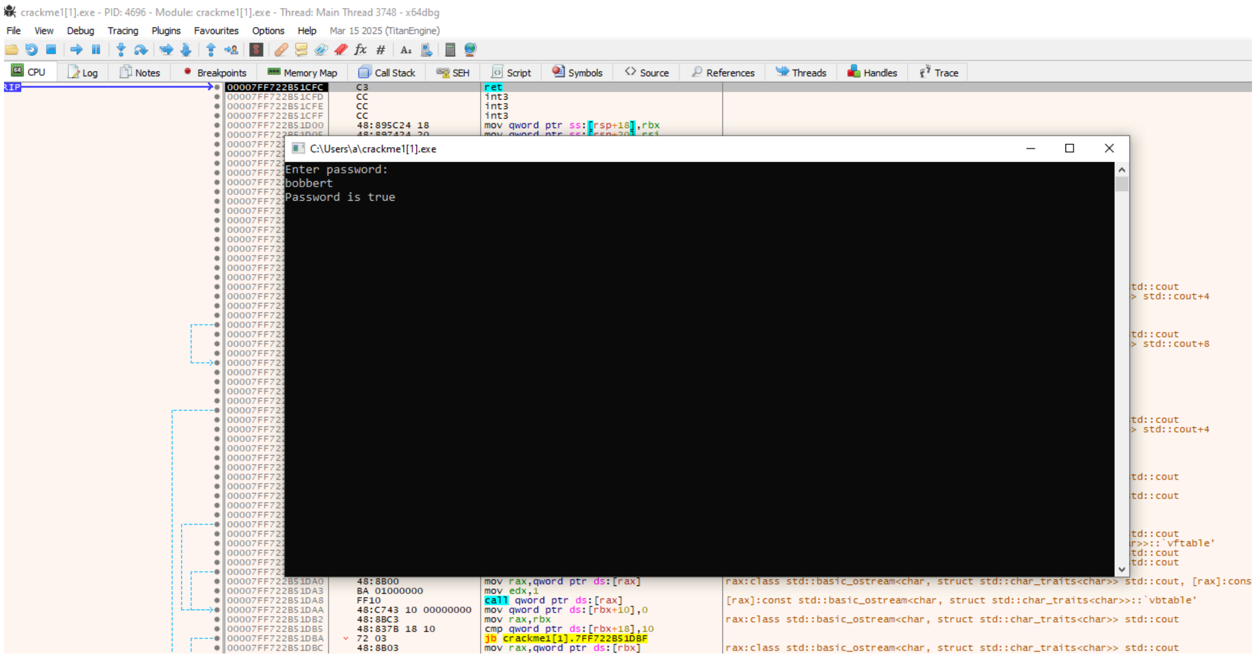
C:\Users\a\Desktop\678bc1084d850ac5f7dc5423\crackme1.exe

```
Enter password:
__abcceeefgikmnnorrr
Password is true
```

Alternatively, I patched the password check so that the machine code jumps over the false password instruction all the time and not only when it is set to zero (JZ → JMP) and allows it to always be set to true

```
             50 29 00 00
    1400017c8 48 8d 15        LEA        resultString,[s_Password_is_true_140004438]      = "Password is true\n"
             69 2c 00 00
    1400017cf eb 07           JMP        LAB_1400017d8
    1400017d1 48 8d 15        LEA        resultString,[s_Password_is_false_140004450]     = "Password is false\n"
             78 2c 00 00
```

This allows me to type in any password and get a success message



crackme1[1].exe - PID: 4696 - Module: crackme1[1].exe - Thread: Main Thread 3748 - x64dbg

```
00007FF722B51CFC   C3                    ret
00007FF722B51CFD   CC                    int3
00007FF722B51CFE   CC                    int3
00007FF722B51CFF   CC                    int3
00007FF722B51D00   48:895C24 18          mov qword ptr ss:[rsp+18],rbx
```

C:\Users\a\crackme1[1].exe

```
Enter password:
bobbert
Password is true
```

```
00007FF722B51DA0   48:8B00               mov rax,qword ptr ds:[rax]          rax:class std::basic_ostream<char, struct std::char_traits<char>> std::cout, [rax]:cons
00007FF722B51DA3   BA 01000000           mov edx,1
00007FF722B51DA8   FF10                  call qword ptr ds:[rax]             [rax]:const std::basic_ostream<char, struct std::char_traits<char>>::`vbtable'
00007FF722B51DAA   48:C743 10 00000000   mov qword ptr ds:[rbx+10],0
00007FF722B51DB2   48:8BC3               mov rax,rbx                         rax:class std::basic_ostream<char, struct std::char_traits<char>> std::cout
00007FF722B51DB5   48:837B 18 10         cmp qword ptr ds:[rbx+18],10
00007FF722B51DBA   72 03                 jb crackme1[1].7FF722B51DBF
00007FF722B51DBC   48:8B03               mov rax,qword ptr ds:[rbx]          rax:class std::basic_ostream<char, struct std::char_traits<char>> std::cout
```