

RE Challenge

File 1:

Given the fact that this is a crackme-style binary, my eyes first navigate to anything involving the comparison of 2 items

```
0x000007c7  e864feffff  call sym.imp.strcmp ; calls a subroutine, push eip into the stack (esp) ; int strcmp(const char *s1, const char *s2)
0x000007cc  0945eb     mov dword [var_18h], eax ; moves data from src to dst
0x000007cf  017d0000   cmp dword [var_18h], 0 ; compare two operands
0x000007d3  7513      jne 0x7e8 ; jump short if not equal/not zero (zf=0)
0x000007d5  480d3d0000. lea rdi, str.password_is_correct ; 0x8d0 ; "password is correct" ; load effective address; const char *s
0x000007dc  e82ffeffff  call sym.imp.puts ; calls a subroutine, push eip into the stack (esp) ; int puts(const char *s)
0x000007e1  b800000000 mov eax, 0 ; moves data from src to dst
0x000007e6  eb11      jmp 0x7f9 ; jump
; CODE XREF from main @ 0x7d3
0x000007e8  480d3d0000. lea rdi, str.password_is_incorrect ; 0x8ba ; "password is incorrect" ; load effective address; const char *s
```

As we can see, a value is being moved from `eax` into the variable `var_18h` then there is a comparison and if it isn't equal to 0, meaning a previous comparison wasn't equal, then it jumps to the address `0x7e8`. `0x7e` contains this:

```
0x000007e8  480d3d0000. lea rdi, str.password_is_incorrect ; 0x8ba ; "password is incorrect" ; load effective address; const char *s
```

So, being an Assembly noob, my mind goes to thinking that any strings mentioned before may be what was compared to our input then another comparison (between `eax` & 0 to make sure our input was what was expected)

```
0x0000076a  55      push rbp
0x0000076b  4889e5   mov rbp, rsp
0x0000076e  4883ec20 sub rsp, 0x20
0x00000772  64488b042528. mov rax, qword fs:[0x28]
0x0000077b  488945f8  mov qword [canary], rax
0x0000077f  31c0     xor eax, eax
0x00000781  480d3d0c0100. lea rdi, str.enter_password
0x00000788  e883feffff  call sym.imp.puts
0x0000078d  8b053d010000 mov eax, dword [str.hax0r]
```

Within the first few lines of `main`, we have the string `hax0r`, so let's test this as a password:

```
[reing] ./crackme1.bin
enter password
hax0r
password is correct
```

Boom! First challenge solved

File 2:

With this second file, symbols are still have the luxury of symbols like we did in the first so navigating it is quite easy, we are able to quickly identify where our input is and where the comparison occurs

```
0x00000738  e8a3feffff  call sym.imp.puts ; calls a subroutine, push eip into the stack (esp) ; int puts(const char *s)
0x0000073d  480d45f4     lea rax, [var_ch] ; load effective address
0x00000741  4809c6     mov rsi, rax ; moves data from src to dst
0x00000744  480d3d0000. lea rdi, [0x00000030] ; "3d" ; load effective address; const char *format
0x0000074b  b800000000 mov eax, 0 ; moves data from src to dst
0x00000750  e89bfeffff  call sym.imp._isoc99_scanf ; calls a subroutine, push eip into the stack (esp) ; int scanf(const char *format)
0x00000755  8b45f4     mov eax, dword [var_ch] ; moves data from src to dst
```

As we can see, `sym.imp.puts` takes our input and loads it into `var_ch` and then we move the value of `var_ch` into `eax`

```
0x00000758  3d7c130000  cmp eax, 0x137c ; compare two operands
0x0000075d  750e      jne 0x76d ; jump short if not equal/not zero (zf=0)
0x0000075f  480d3d0000. lea rdi, str.password_is_valid ; 0x83b ; "password is valid" ; load effective address; const char *s
0x00000766  e865feffff  call sym.imp.puts ; calls a subroutine, push eip into the stack (esp) ; int puts(const char *s)
0x0000076b  eb0c      jmp 0x779 ; jump
; CODE XREF from main @ 0x75d
0x0000076d  480d3d0000. lea rdi, str.password_is_incorrect ; 0x84d ; "password is incorrect" ; load effective address; const char *s
```

Then our input is compared against `0x137c` which is 4988 in decimal, and if the comparison isn't equal, we get a **password is incorrect** but if the comparison is equal, we get a **password is valid**.

And as we can see, that's our password!

```
[reing] ./crackme2.bin
enter your password
4988
password is valid
```

File 3:

```
0x0000071a 55          push rbp          ; push word, doubleword or quadword onto the stack
0x0000071b 4889e5     mov rbp, rsp      ; moves data from src to dst
0x0000071e 4883ec30   sub rsp, 0x30     ; subtract src and dst, stores result on dst
0x00000722 64488b0c2520. mov rax, qword fs:[0x20] ; moves data from src to dst
0x0000072b 488945f8   mov qword [canary], rax ; moves data from src to dst
0x0000072f 31c8      xor eax, eax      ; logical exclusive or
0x00000731 66c745dd617e. mov word [var_23h], 0x7e61 ; 'az' ; moves data from src to dst
0x00000737 c845df74   mov byte [var_21h], 0x74 ; 't' ; moves data from src to dst
0x0000073b 488d3d120100. lea rdi, str.enter_your_password ; 0x854 ; "enter your password" ; load effective address; const char *s
0x00000742 e889feffff. call sym.imp.puts ; calls a subroutine, push eip into the stack (esp) ; int puts(const char *s)
0x00000747 488d45eb   lea rax, [var_20h] ; load effective address
0x0000074b 4889c8     mov rsi, rax      ; moves data from src to dst
0x0000074e 488d3d130100. lea rdi, [0x00000000] ; "Na" ; load effective address; const char *format
0x00000755 b800000000. mov eax, 0        ; moves data from src to dst
0x0000075a e891feffff. call sym.imp._isoc99_scanf ; calls a subroutine, push eip into the stack (esp) ; int scanf(const char *format)
0x0000075f c74508000000. mov dword [var_28h], 0 ; moves data from src to dst
0x00000766 eb2f      jmp 0x797         ; jump
; CODE XREF from main @ 0x79b
0x00000768 8b4528     mov eax, dword [var_28h] ; moves data from src to dst
0x0000076b 4898      cqqr             ; sign extend eax into rax
0x0000076d 0fb65405eb. movzx edx, byte [rbp + rax - 0x20] ; move dst register size padding with zeroes
0x00000772 8b4528     mov eax, dword [var_28h] ; moves data from src to dst
0x00000775 4898      cqqr             ; sign extend eax into rax
0x00000777 0fb64405dd. movzx eax, byte [rbp + rax - 0x21] ; move dst register size padding with zeroes
0x0000077c 38c2      cmp dl, al        ; compare two operands
0x0000077e 7413      je 0x793         ; jump short if equal (zf=1)
0x00000780 488d3de40000. lea rdi, str.password_is_incorrect ; 0x86b ; "password is incorrect" ; load effective address; const char *s
0x00000787 e844feffff. call sym.imp.puts ; calls a subroutine, push eip into the stack (esp) ; int puts(const char *s)
0x0000078c b800000000. mov eax, 0        ; moves data from src to dst
0x00000791 eb1b      jmp 0x7ae         ; jump
; CODE XREF from main @ 0x77e
0x00000793 83450801   add dword [var_28h], 1 ; adds src and dst, stores result on dst
; CODE XREF from main @ 0x766
0x00000797 837d0802   cmp dword [var_28h], 2 ; compare two operands
0x0000079b 7ecb      jle 0x768        ; jump short if less or equal/not greater (zf=1 or sf=0)
0x0000079d 488d3dd00000. lea rdi, str.password_is_correct ; 0x881 ; "password is correct" ; load effective address; const char *s
0x000007a4 e827feffff. call sym.imp.puts ; calls a subroutine, push eip into the stack (esp) ; int puts(const char *s)
0x000007a9 b800000000. mov eax, 0        ; moves data from src to dst
```

Now this one is more complex, in terms of the assembly.

In terms of the programming logic, it's simple

All it does (to my knowledge) is load the values az & t into variables, loop through & add them into one variable, then compare our input to that variable

Testing this theory...

```
[reing] ./crackme3.bin
enter your password
azt
password is correct
```