I can code in cow-boy

The challenge was to get a password inside of a binary compiled with cobc. So cobol it is !
The challenge looked harder than it was actually.

By looking at it through IDA, we can see that the main function calls the access_control function

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  unsigned int v3; // eax

  cob_init((unsigned int)argc, argv);
  v3 = ACCESS__CONTROL();
  cob_stop_run(v3);
  return ACCESS__CONTROL();
}
```

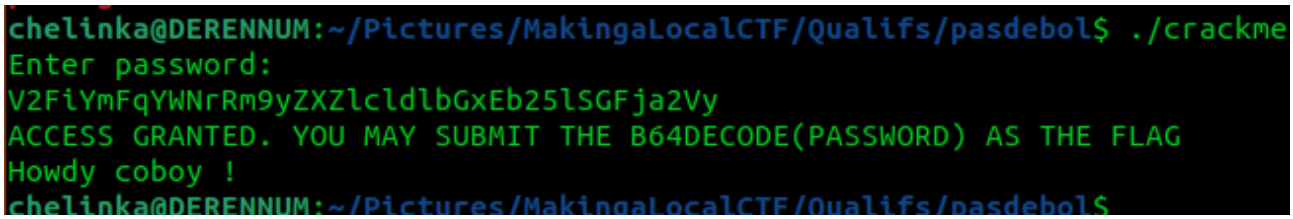
Said access_function contains the main part of the program, where the secret is created from hardcoded strings. It is the compared to the user's input and, if the input corresponds to the secret, the message ACCESS GRANTED is printed to the screen.

```
  cob_string_init(&f_9_26, 0LL);
  cob_string_delimited(0LL);
  cob_string_append(&f_15_25);
  cob_string_append(&f_13_24);
  cob_string_append(&f_17_23);
  cob_string_append(&f_14_22);
  cob_string_append(&f_18_21);
  cob_string_append(&f_19_20);
  cob_string_append(&f_12_19);
  cob_string_append(&f_21_18);
  cob_string_append(&f_22_17);
  cob_string_append(&f_23_16);
  cob_string_append(&f_24_15);
  cob_string_finish();
  if ( !memcmp(&b_8_14, &b_9_13, 0x28uLL) )
  {
    cob_display(0LL, 1LL, 1LL, &f_10_12);
    cob_display(0LL, 1LL, 1LL, &c_2);
  }
  else
  {
    cob_display(0LL, 1LL, 1LL, &f_11_11);
  }
  cob_stop_run();
_12:
  cob_check_version("prog.cob", "4.0-early-dev", 0LL);
  cob_module_path = *(_QWORD *)(v2[0] + 48);
  ACCESS__CONTROL_module_init(module_36);
  *(_QWORD *)(module_36 + 64) = 0LL;
  cob_set_cancel(module_36);
  v3[0] = 4LL;
  v3[1] = (__int64)&b_2_10;
  v3[2] = (__int64)&a_3;
  cob_set_llint(v3, 1000000000LL, 0LL);
  memset(&b_8_14, 32, 0x28uLL);
  memcpy(&b_9_13, "PASSWORD", 8uLL);
  memset(&unk_42F8, 32, 0x20uLL);
  qmemcpy(&b_10_9, "ACCESS GRANTED. YOU MAY SUBMIT THE B64DECODE(PASSWORD) AS THE FLAG", 66);
  qmemcpy(&b_11_8, "ACCESS DENIED", 13);
```

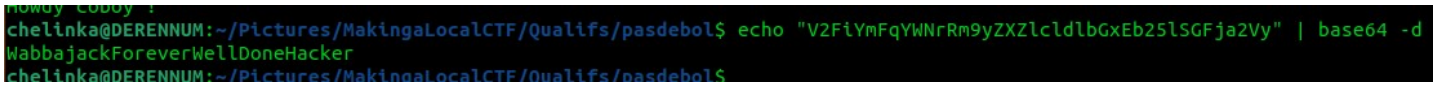
The flag is simply checked against the user's input. Now we may run it through gdb and simply read it from memory after stopping right before the comparison in the ACCESS__CONTROL_ function:

```
  0x5555555555ed <ACCESS__CONTROL_+603>    mov    rdi, rax
► 0x5555555555f0 <ACCESS__CONTROL_+606>    call   memcmp@plt                  <memcmp@plt>
      s1: 0x5555555582c0 (b_8) ◂— '
      s2: 0x5555555582f0 (b_9) ◂— 'V2FiYmFqYWNrRm9yZXZlcldlbGxEb25lSGFja2Vy'
      n: 0x28

  0x5555555555f5 <ACCESS__CONTROL_+611>    test   eax, eax
  0x5555555555f7 <ACCESS__CONTROL_+613>    jne    ACCESS__CONTROL_+687                <ACCESS__

  0x5555555555f9 <ACCESS__CONTROL_+615>    lea    rax, [rip + 0x2bb0]           <f_10>
  0x555555555600 <ACCESS__CONTROL_+622>    mov    rcx, rax
  0x555555555603 <ACCESS__CONTROL_+625>    mov    edx, 1
```

Here we can see exactly what the input string should be. Let’s try it:



We can see that the secret works perfectly. Let’s retrieve the flag !



FLAG: HACKDAY{WabbajackForeverWellDoneHacker}

Written by Chelinka