Microcorruption - Hanoi

Hanoi

Reverse Engineering

10 Points

> Remember: passwords are between 8 and 16 characters.

> The LockIT Pro can send the LockIT Pro HSM-1 a password, and the HSM will return if the password is correct by setting a flag in memory.

Analysis

main()

```
4438 <main>
4438:  b012 2045      call      #0x4520 <login>
443c:  0f43           clr       r15
```

In this challenge, the main function simply calls the login function, and then clears the contents of the r15 register.

login()

```
4520 <login>
4520:  c243 1024      mov.b     #0x0, &0x2410
4524:  3f40 7e44      mov       #0x447e "Enter the password to continue.",
r15
4528:  b012 de45      call      #0x45de <puts>
452c:  3f40 9e44      mov       #0x449e "Remember: passwords are between 8
and 16 characters.", r15
4530:  b012 de45      call      #0x45de <puts>
4534:  3e40 1c00      mov       #0x1c, r14
4538:  3f40 0024      mov       #0x2400, r15
453c:  b012 ce45      call      #0x45ce <getsn>
4540:  3f40 0024      mov       #0x2400, r15
4544:  b012 5444      call      #0x4454 <test_password_valid>
4548:  0f93           tst       r15
454a:  0324           jz        $+0x8 <login+0x32>
454c:  f240 1700 1024 mov.b     #0x17, &0x2410
```

```
4552:  3f40 d344      mov       #0x44d3 "Testing if password is valid.", r15
4556:  b012 de45      call      #0x45de <puts>
455a:  f290 da00 1024 cmp.b     #0xda, &0x2410
4560:  0720           jnz       $+0x10 <login+0x50>
4562:  3f40 f144      mov       #0x44f1 "Access granted.", r15
4566:  b012 de45      call      #0x45de <puts>
456a:  b012 4844      call      #0x4448 <unlock_door>
456e:  3041           ret
4570:  3f40 0145      mov       #0x4501 "That password is not correct.", r15
4574:  b012 de45      call      #0x45de <puts>
4578:  3041           ret
```

It appears that the login() function calls the test_password_valid() function.

Instructions 455a - 4560 are also of note. At 455a, the value at &0x2410 in memory is compared with 0xda. If the comparison passes, then the zero register is set. If the register is not set, then the next instruction 4560 will jump 16 bytes forward in memory. If the program does not jump, then the door is unlocked. So, we need &0x2410 to be set to 0xda in order to pass this comparison and unlock the door.

The function test_password_valid() is called before this, maybe there is something there?

test_password_valid()

```
4454 <test_password_valid>
4454:  0412           push      r4
4456:  0441           mov       sp, r4
4458:  2453           incd      r4
445a:  2183           decd      sp
445c:  c443 fcff      mov.b     #0x0, -0x4(r4)
4460:  3e40 fcff      mov       #0xfffc, r14
4464:  0e54           add       r4, r14
4466:  0e12           push      r14
4468:  0f12           push      r15
446a:  3012 7d00      push      #0x7d
446e:  b012 7a45      call      #0x457a <INT>
4472:  5f44 fcff      mov.b     -0x4(r4), r15
4476:  8f11           sxt       r15
4478:  3152           add       #0x8, sp
447a:  3441           pop       r4
447c:  3041           ret
```

There are no comparisons made in this function, and nothing immediately stood out to me at first.

```
453c:  b012 ce45      call      #0x45ce <getsn>
4540:  3f40 0024      mov       #0x2400, r15
4544:  b012 5444      call      #0x4454 <test_password_valid>
4548:  0f93           tst       r15
454a:  0324           jz        $+0x8 <login+0x32>
```

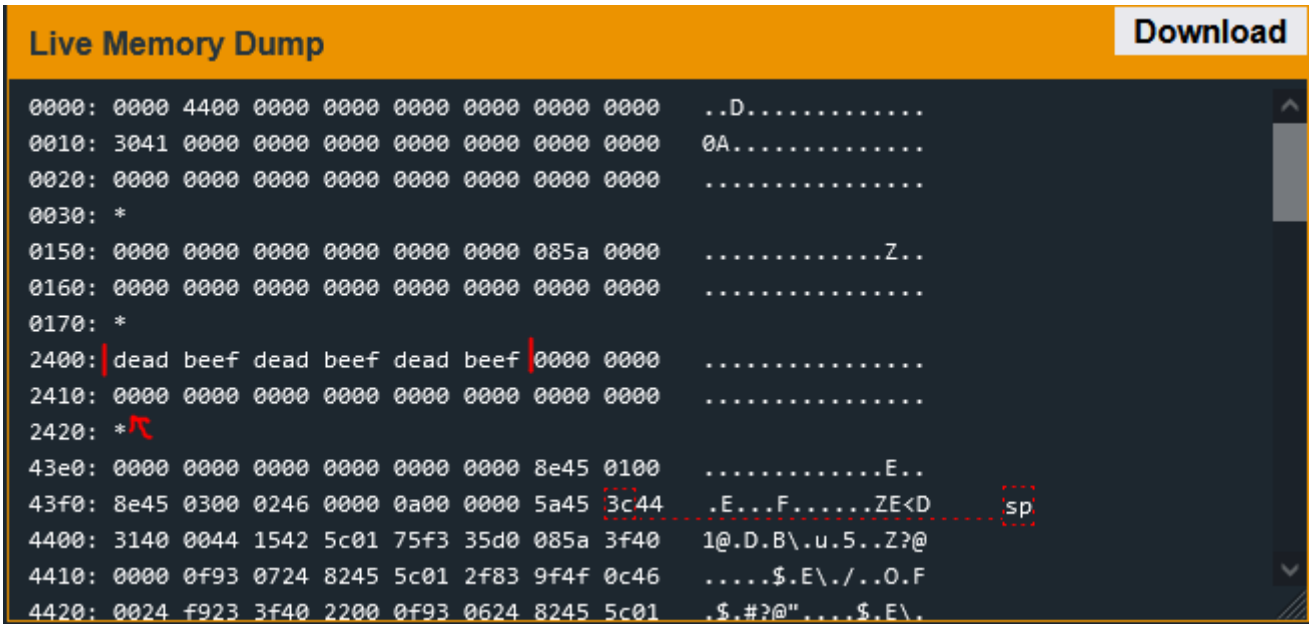
However, there is an interesting section of memory just before this function is called in login(). It appears the value in memory at 0x2400 is moved into register r15 before the function call, and then r15 is compared for a non-zero value at address 4548 just after the function call. Thus, the input may be stored at memory address 0x2400. If the flag being checked for login validation is located at 0x2410 then the distance between input and the variable is very close. This could indicate a buffer overflow vulnerability. This is also interesting because of the note output by the program at the start of execution:

> Remember: passwords are between 8 and 16 characters

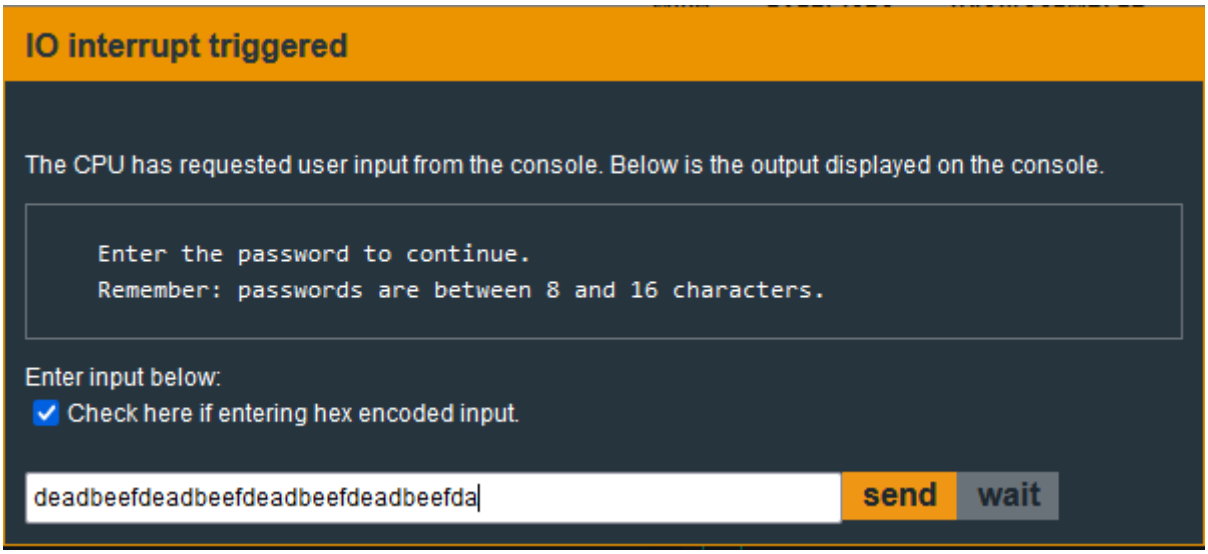
Are we able to reach this memory address from the input prompt? If the input size is not checked, as in some older C code, then we may be able to perform a buffer overflow on the login() function to overwrite this flag variable in memory.

To test this, we can pass in a noticeable input, and then inspect the memory to see where the input ends up getting stored.

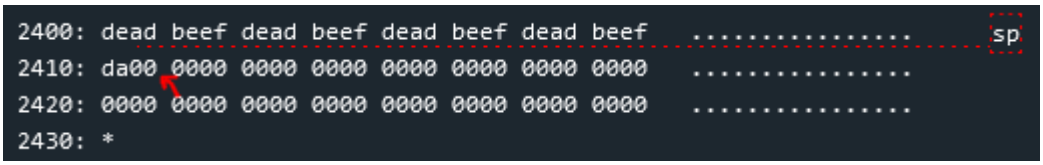
I set a break point just before this jump at instruction 455a which checks the memory at address 0x2410. Then I continued the program inputting 'deadbeefdeadbeefdeadbeef' as a hex encoded password.



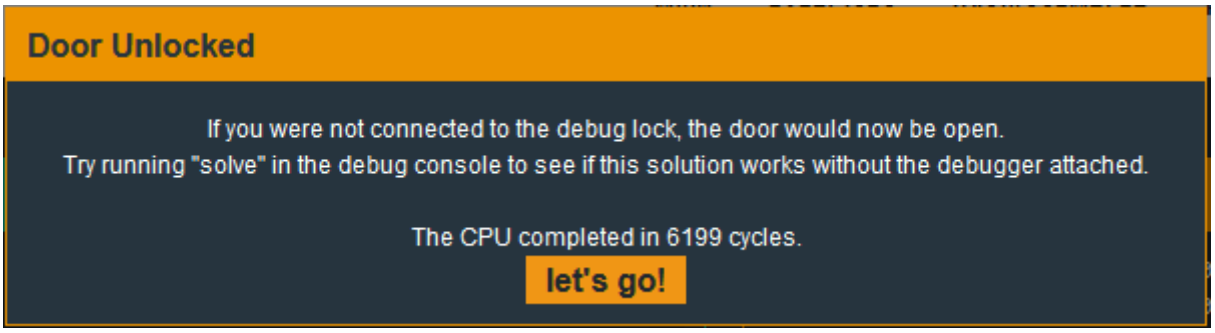
And this almost worked! I only passed in 12 hex characters in my input, however, they were stored just before the memory address that is checked for valid passwords. If I add 4 more hex characters, followed by 0xda then I may be able to overwrite the flag and recreate the success behavior of the test_password_valid() function regardless of whether the password matches the stored password or not.



I then reset the cpu, and for my input added four more hex characters followed by the success flag value 0xda to be stored in the checked memory address 0x2410. I am assuming we do not need to worry about endianness since the program is only checking a single byte.

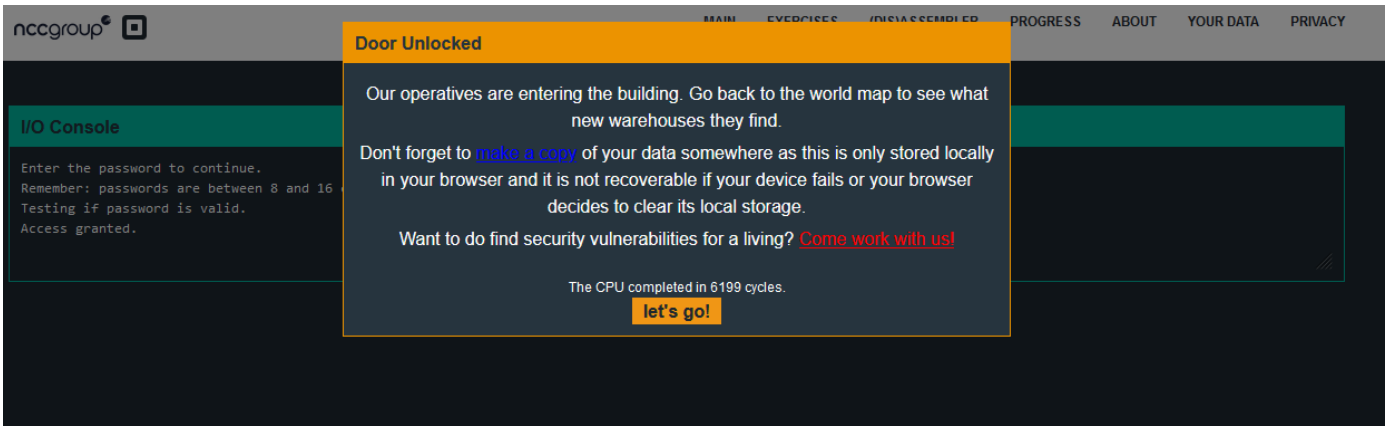


Now, with the updated input, the flag value was stored at the correct memory address!

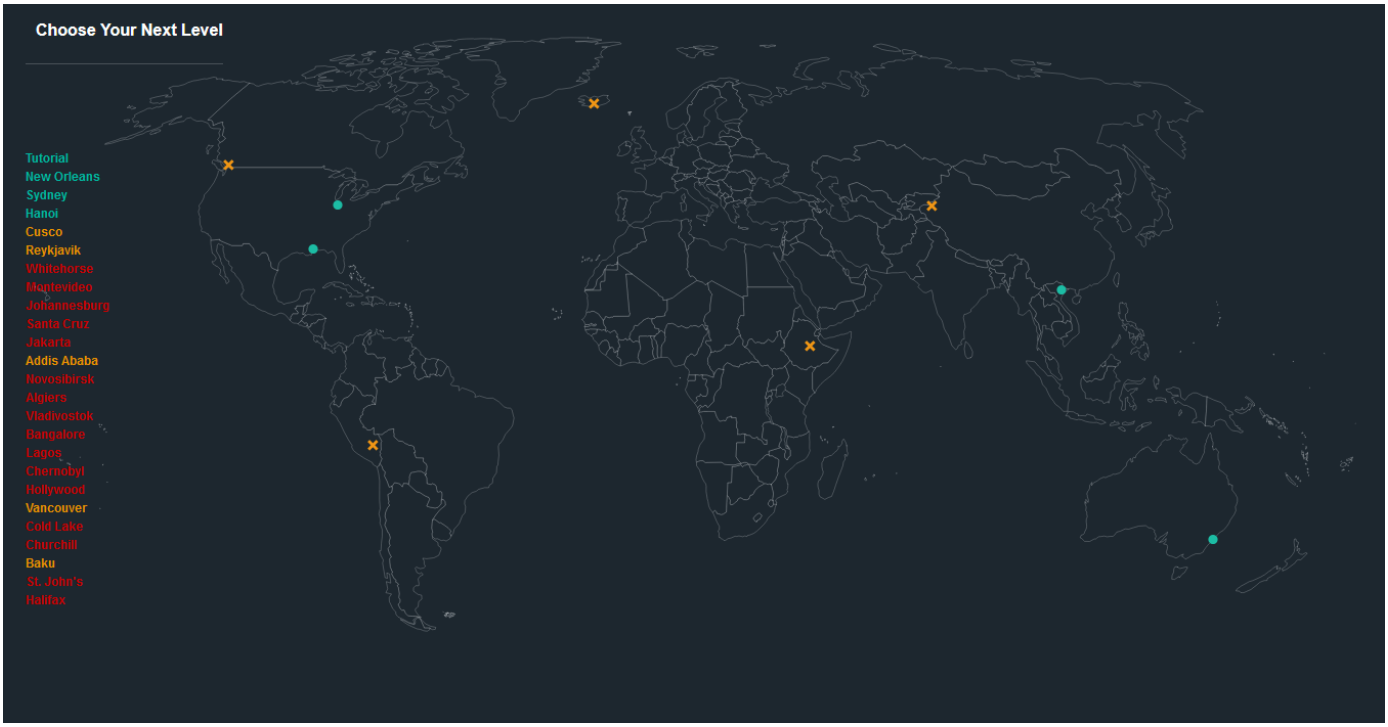


I continued the program's execution, and the door unlocked!

Because the input password was larger than the allocated size, the addresses higher in memory were overwritten with my input. Knowing the address of the flag, and the success value returned when a valid password is input, I constructed a payload that overwrites the flag to be the success value regardless if my input matches the stored password.



With this payload, I then moved on to the real lock, and completed the challenge.



Current Final Metadata

{"cpu":"msp.Txis4VRqFeq3EfttpdQnU4q/g7GMxAzSWKccGHfS9fU=","solutions":"
[{"level_id":1,"input":"7279616e6861636b;"},{"level_id":1,"input":"7279616e6861636b;"},
{"level_id":2,"input":"4476794e46746d;"},{"level_id":2,"input":"4476794e46746d;"},
{"level_id":3,"input":"5a51504276492672;"},{"level_id":2,"input":"4476794e46746d;"},
{"level_id":3,"input":"5a51504276492672;"},
{"level_id":4,"input":"deadbeefdeadbeefdeadbeefdeadbeefda;"}]"}