Microcorruption - Sydney

Sydney

Reverse Engineering

10 Points

> This is Software Revision 02. We have received reports that the prior version of the lock was bypassable without knowing the password. We have fixed this and removed the password from memory.

> From the software disassembly, memory dump, and terminal output of the program we must obtain the password for the lock.

Analysis

Disassembly

```
4438 <main>
4438:  3150 9cff      add       #0xff9c, sp
443c:  3f40 b444      mov       #0x44b4 "Enter the password to continue.", r15
4440:  b012 6645      call      #0x4566 <puts>
4444:  0f41           mov       sp, r15
4446:  b012 8044      call      #0x4480 <get_password>
444a:  0f41           mov       sp, r15
444c:  b012 8a44      call      #0x448a <check_password>
4450:  0f93           tst       r15
4452:  0520           jnz       $+0xc <main+0x26>
4454:  3f40 d444      mov       #0x44d4 "Invalid password; try again.", r15
4458:  b012 6645      call      #0x4566 <puts>
445c:  093c           jmp       $+0x14 <main+0x38>
445e:  3f40 f144      mov       #0x44f1 "Access Granted!", r15
4462:  b012 6645      call      #0x4566 <puts>
4466:  3012 7f00      push      #0x7f
446a:  b012 0245      call      #0x4502 <INT>
446e:  2153           incd      sp
4470:  0f43           clr       r15
4472:  3150 6400      add       #0x64, sp
```

In the main function, there is no longer a `create_password` function.

However, there is still a `check_password` method that may be of interest:

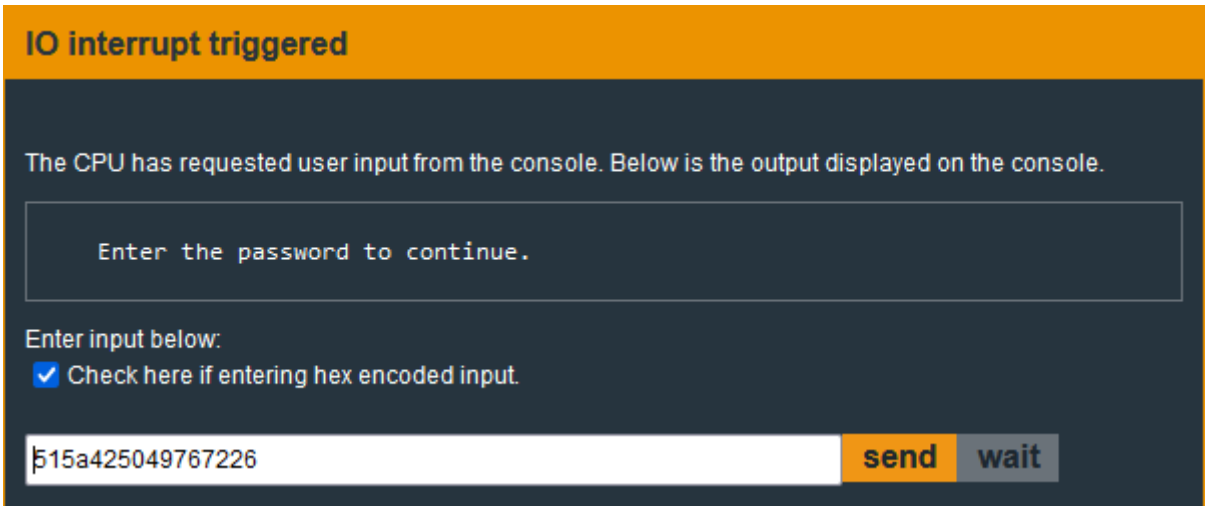
```
448a <check_password>
448a:  bf90 5a51 0000 cmp       #0x515a, 0x0(r15)
4490:  0d20           jnz       $+0x1c <check_password+0x22>
4492:  bf90 5042 0200 cmp       #0x4250, 0x2(r15)
```

```
4498:  0920           jnz       $+0x14 <check_password+0x22>
449a:  bf90 7649 0400 cmp       #0x4976, 0x4(r15)
44a0:  0520           jnz       $+0xc <check_password+0x22>
44a2:  1e43           mov       #0x1, r14
44a4:  bf90 2672 0600 cmp       #0x7226, 0x6(r15)
44aa:  0124           jz        $+0x4 <check_password+0x24>
44ac:  0e43           clr       r14
44ae:  0f4e           mov       r14, r15
44b0:  3041           ret
```

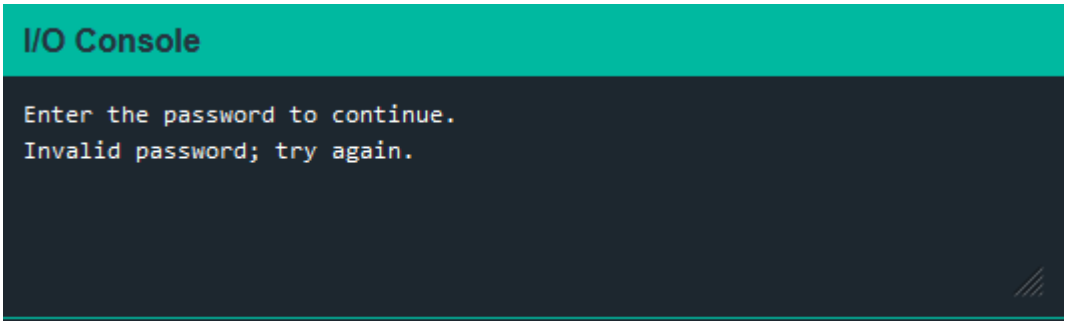
It appears the function is comparing a value to certain bytes of the value contained in register r15.

- cmp #0x515a, 0x0(r15) // compares the first and second bytes with 0x515a
- cmp #0x4250, 0x2(r15) // compares the third and fourth bytes with 0x4250
- cmp #0x4976, 0x4(r15) // compares the fifth and sixth bytes with 0x4976
- cmp #0x7226, 0x6(r15) // compares the seventh and eigth bytes with 0x7226

Thus, if the data is stored in big endian format, then the password will be:
0x 515a 4250 4976 7226



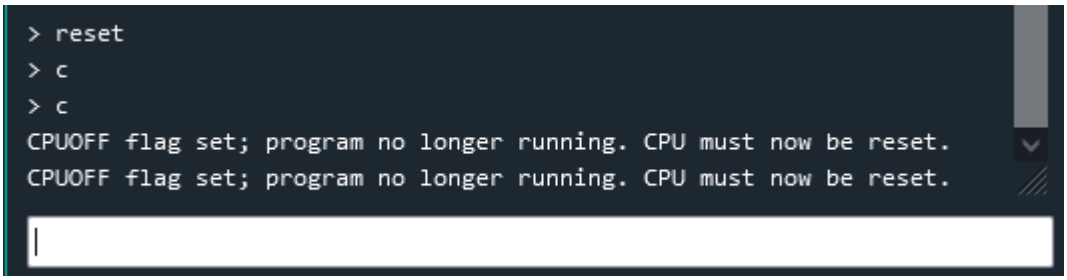
Because no other functions are called after `check_password`, I then tested this value in the password prompt. I also utilized the option to submit hex encoded input, so that it would not need to be translated into its ASCII representation.



However, the input value failed to unlock the door. My next thought was to investigate if the data was being stored in little endian format.

```
448a <check_password>
448a:  bf90 5a51 0000 cmp       #0x515a, 0x0(r15)
4490:  0d20           jnz       $+0x1c <check_password+0x22>
4492:  bf90 5042 0200 cmp       #0x4250, 0x2(r15)
4498:  0920           jnz       $+0x14 <check_password+0x22>
449a:  bf90 7649 0400 cmp       #0x4976, 0x4(r15)
44a0:  0520           jnz       $+0xc <check_password+0x22>
44a2:  1e43           mov       #0x1, r14
44a4:  bf90 2672 0600 cmp       #0x7226, 0x6(r15)
44aa:  0124           jz        $+0x4 <check_password+0x24>
44ac:  0e43           clr       r14
44ae:  0f4e           mov       r14, r15
44b0:  3041           ret
```

I then reanalyzed the `check_password` function, and constructed the next password attempt starting from the last value and working backwards. This resulted in the value: 0x 7226 4976 4250 515a

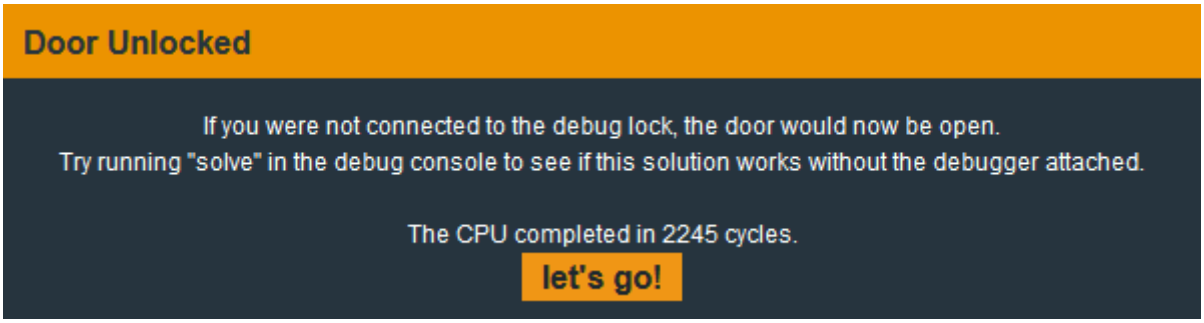


However, after resetting the CPU and inputting this value, a failure resulted again.

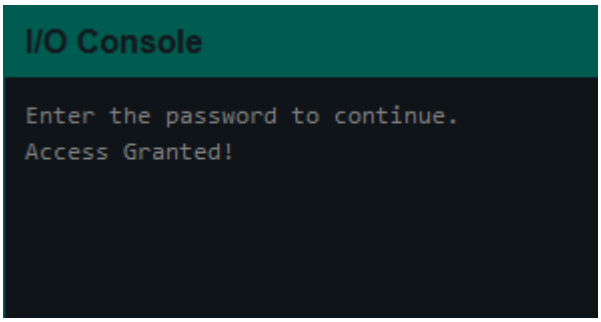
I then realized I was converting the data into little endian format incorrectly.

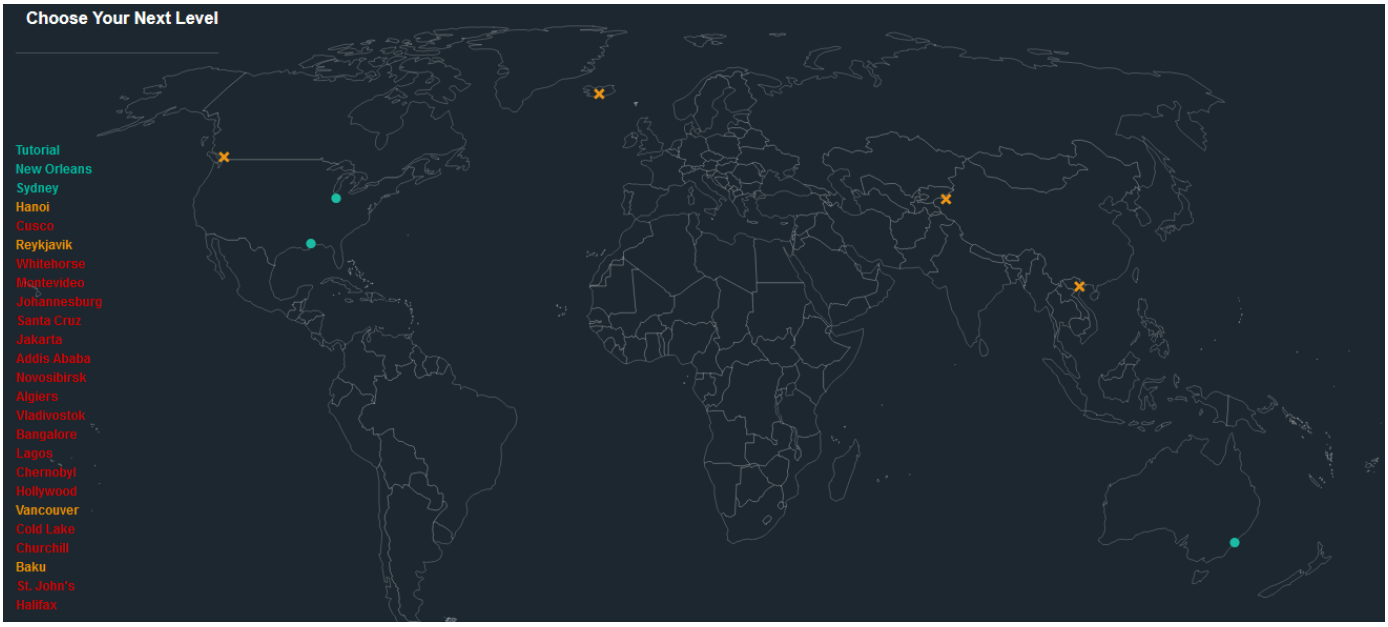
Solution

Each individual hex value within each cmp statement in the `check_password` funciton should be swapped rather than the entire values across different positions. Therefore, the correct little endian value is:
0x 5a51 5042 7649 2672



After resetting the CPU and entering this value, the door unlocked! Lastly, I connected to the remote lock by executing `solve` in the terminal to submit the password and complete the Sydney 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;"}]"}