Micro Corruption

Tutorial (10 pts)

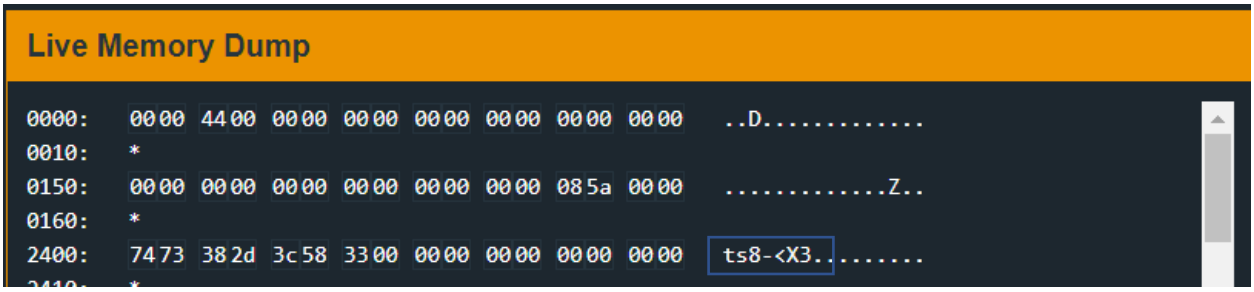
New Orleans (10 pts)

```
4438 <main>
4438:  3150 9cff      add       #0xff9c, sp
443c:  b012 7e44      call      #0x447e <create_password>
4440:  3f40 e444      mov       #0x44e4 "Enter the password to continue", r15
4444:  b012 9445      call      #0x4594 <puts>
4448:  0f41           mov       sp, r15
444a:  b012 b244      call      #0x44b2 <get_password>
444e:  0f41           mov       sp, r15
4450:  b012 bc44      call      #0x44bc <check_password>
4454:  0f93           tst       r15
4456:  0520           jnz       #0x4462 <main+0x2a>
4458:  3f40 0345      mov       #0x4503 "Invalid password; try again.", r15
445c:  b012 9445      call      #0x4594 <puts>
4460:  063c           jmp       #0x446e <main+0x36>
4462:  3f40 2045      mov       #0x4520 "Access Granted!", r15
4466:  b012 9445      call      #0x4594 <puts>
446a:  b012 d644      call      #0x44d6 <unlock_door>
446e:  0f43           clr       r15
4470:  3150 6400      add       #0x64, sp
```

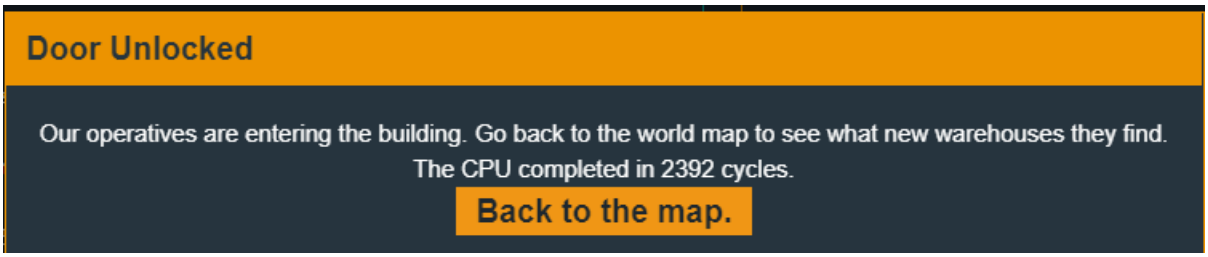
This was the first Challenge that we faced. We needed to crack the password to move on to the next stage. You can see from the main function that it first calls the <create_password> function to create a password. This is where we wanted to look first so we can see how the password is generated and then enter it in and move on.

```
447e <create_password>
447e:  3f40 0024      mov       #0x2400, r15
4482:  ff40 7400 0000 mov.b     #0x74, 0x0(r15)
4488:  ff40 7300 0100 mov.b     #0x73, 0x1(r15)
448e:  ff40 3800 0200 mov.b     #0x38, 0x2(r15)
4494:  ff40 2d00 0300 mov.b     #0x2d, 0x3(r15)
449a:  ff40 3c00 0400 mov.b     #0x3c, 0x4(r15)
44a0:  ff40 5800 0500 mov.b     #0x58, 0x5(r15)
44a6:  ff40 3300 0600 mov.b     #0x33, 0x6(r15)
44ac:  cf43 0700      mov.b     #0x0, 0x7(r15)
```

This is the create password function. It's pretty simple as it just takes a hex value and moves it to the r15 register.



After stepping through the function you can see that our password was generated and we can solve the puzzle.



Sydney (15 pts)

```
The LockIT Pro contains a Bluetooth chip allowing it to
communiciate with the LockIT Pro App, allowing the LockIT Pro to
be inaccessable from the exterior of the building.

There is no default password on the LockIT Pro---upon receiving
the LockIT Pro, a new password must be set by connecting it to the
LockIT Pro App and entering a password when prompted, and then
restarting the LockIT Pro using the red button on the back.

This is Hardware Version A. It contains the Bluetooth connector
built in, and one available port to which the LockIT Pro Deadbolt
should be connected.
```

This was the second challenge that we faced.

```
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       #0x445e <main+0x26>
4454:  3f40 d444      mov       #0x44d4 "Invalid password; try again.", r15
4458:  b012 6645      call      #0x4566 <puts>
445c:  093c           jmp       #0x4470 <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
```

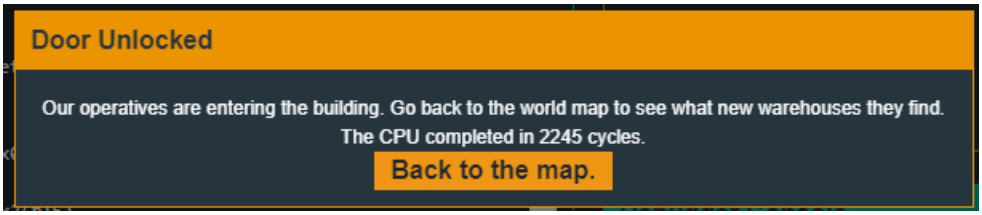
The main function looked somewhat similar to the last challenge being that there is a get password function that is compared to a check password function.

```
4390:   6045 0200 9c43 6400 8844 4a44 7061 7373   `E...Cd..DJDpass
43a0:   776f 7264 0000 0000 0000 0000 0000 0000   word............
```

I first set a break point at the check password function so we could see how the program interacted with the arbratarty password I gave it. I first thought, oh perfect the password is stored in the 4 bytes of memory right before my inputed password. From this I tried to solve the puzzle using 9c43640088444a444 as the password in hex. I was wrong.

```
448a <check_password>
448a:  bf90 2a3b 0000 cmp       #0x3b2a, 0x0(r15)
4490:  0d20           jnz       $+0x1c
4492:  bf90 413e 0200 cmp       #0x3e41, 0x2(r15)
4498:  0920           jnz       $+0x14
449a:  bf90 6859 0400 cmp       #0x5968, 0x4(r15)
44a0:  0520           jne       #0x44ac <check_password+0x22>
44a2:  1e43           mov       #0x1, r14
44a4:  bf90 245f 0600 cmp       #0x5f24, 0x6(r15)
44aa:  0124           jeq       #0x44ae <check_password+0x24>
```

I then looked at the check password function to see what was really happening. I noticed that it couldn't be my last password I tried just because the check password function compares my password to different hex values. I then figured that the password must be the hex values that we are comparing it to. I then tried 3b2b3e4159685f24 as the password but still wasn't to get in. This confused me a bit but then realized that it is probably little-endian. We can see this in the boxes that the addresses are little-endian. I then tried 2a3b413e6859245f as our password.



HOORAY!! We got it and moved on to the next stage.

Hanoi (20 pts)

```
There is no default password on the LockIT Pro HSM-1. Upon
receiving the LockIT Pro, a new password must be set by first
connecting the LockitPRO HSM to output port two, connecting it to
the LockIT Pro App, and entering a new password when prompted, and
then restarting the LockIT Pro using the red button on the back.

LockIT Pro Hardware Security Module 1 stores the login password,
ensuring users can not access the password through other means.
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.

This is Hardware Version B. It contains the Bluetooth connector
built in, and two available ports: the LockIT Pro Deadbolt should
be connected to port 1, and the LockIT Pro HSM-1 should be
connected to port 2.

This is Software Revision 01, allowing it to communicate with the
LockIT Pro HSM-1
```

This was an overview of the next challenge.

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

This was the main function. There was not much here except that it calls the login function which is where most of the program took place.

```
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
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
454c:  f240 5000 1024 mov.b     #0x50, &0x2410
4552:  3f40 d344      mov       #0x44d3 "Testing if password is valid.", r15
4556:  b012 de45      call      #0x45de <puts>
455a:  f290 be00 1024 cmp.b     #0xbe, &0x2410
4560:  0720           jne       #0x4570 <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
```

This is the login function. It was quite more than the last 3 challenges so I knew this might take some time. The first thing I noticed it that it calls a <getsn> function which is similar to a fgets() so I looked at that function.

After looking at the getsn function I decied to set a breakpoint at the call to test_valid_password. This allowed me to input a password of my choice.

```
2400:   4141 4141 4141 4141 4141 4141 4141 4141   AAAAAAAAAAAAAAAA
2410:   0000 0000 0000 0000 0000 0000 0000 0000   ................
```

When prompted I inputted 16 characters since our input had to be between 8 and 16 characters. I then noticed that it was stored in the memory address 2400. I then went to the <test_valid_password> to see what it did.

```
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
```

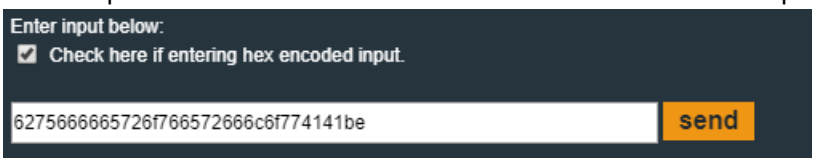
After stepping through this nothing happened except that the r15 register was set back to 0000. I then returned to the login function.

```
4544:  b012 5444      call    #0x4454 <test_password_valid>
4548:  0f93           tst     r15
454a:  0324           jz      $+0x8
454c:  f240 5000 1024 mov.b   #0x50, &0x2410
4552:  3f40 d344      mov     #0x44d3 "Testing if password is valid.", r15
4556:  b012 de45      call    #0x45de <puts>
455a:  f290 be00 1024 cmp.b   #0xbe, &0x2410
4560:  0720           jne     #0x4570 <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
```

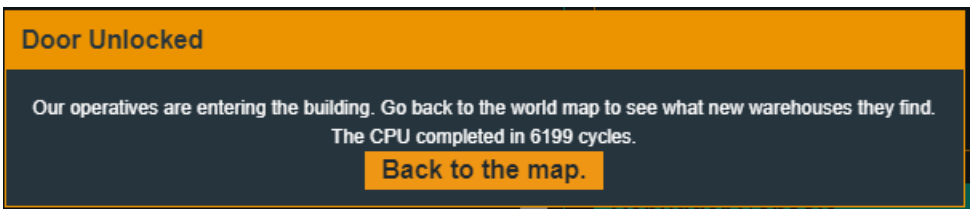
After the test_valid_password function it went through the 4548 and 454a calls but then jumped to 4552 skipping over the mov.b. I didn't worry about this and kept moving through it. I noticed that it does a string compare on the 2410 block of memory. I then tested to see if we imputed 17 characters for the password if the last one would be stored in the first 2410 block of memory.

```
2400:   6161 6161 6161 6161 6161 6161 6161 6161   aaaaaaaaaaaaaaaa
2410:   6100 0000 0000 0000 0000 0000 0000 0000   a...............
```

I was right it was stored in the first block of the 2410 memory block. I then went back to the input function and saw that the compare had to be against the hex value of 0xbe. After googling what hex value that was I found it was not a printable character. I then figured the password had to be sent in hex. I then went to solve the puzzle with the following password



This was kinda cheeky cause if you look at the hex values on an ASCII table it spells out bufferoverflowAAbe. I knew that this was an overflow once we had to check the 2410 block of memory.



Cusco (25 pts)

```
This is Hardware  Version B.  It contains  the Bluetooth connector
built in, and two available  ports: the LockIT Pro Deadbolt should
be  connected to  port  1,  and the  LockIT  Pro  HSM-1 should  be
connected to port 2.

This is Software Revision 02. We have improved the security of the
lock by  removing a conditional  flag that could  accidentally get
set by passwords that were too long.
```

This was the briefing for this level. I decided to do this level for no other reason then it seemed a little easier than the other 2.

```
4500 <login>
4500:  3150 f0ff      add       #0xfff0, sp
4504:  3f40 7c44      mov       #0x447c "Enter the password to continue.", r15
4508:  b012 a645      call      #0x45a6 <puts>
450c:  3f40 9c44      mov       #0x449c "Remember: passwords are between 8 and 16 characters
4510:  b012 a645      call      #0x45a6 <puts>
4514:  3e40 3000      mov       #0x30, r14
4518:  0f41           mov       sp, r15
451a:  b012 9645      call      #0x4596 <getsn>
451e:  0f41           mov       sp, r15
4520:  b012 5244      call      #0x4452 <test_password_valid>
4524:  0f93           tst       r15
4526:  0524           jz        #0x4532 <login+0x32>
4528:  b012 4644      call      #0x4446 <unlock_door>
452c:  3f40 d144      mov       #0x44d1 "Access granted.", r15
4530:  023c           jmp       #0x4536 <login+0x36>
4532:  3f40 e144      mov       #0x44e1 "That password is not correct.", r15
4536:  b012 a645      call      #0x45a6 <puts>
453a:  3150 1000      add       #0x10, sp
453e:  3041           ret
```

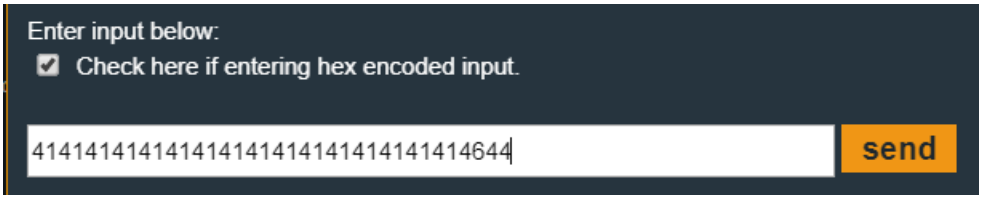
This was the main function that is called that does all the operations. I look very similar to our last problem where we enter a password between 8-16 characters the only difference is that this does not do a compare on the last byte of memory. I believe this to be another buffer overflow attack.

```
4452 <test_password_valid>
4452:  0412           push      r4
4454:  0441           mov       sp, r4
4456:  2453           incd      r4
4458:  2183           decd      sp
445a:  c443 fcff      mov.b     #0x0, -0x4(r4)
445e:  3e40 fcff      mov       #0xfffc, r14
4462:  0e54           add       r4, r14
4464:  0e12           push      r14
4466:  0f12           push      r15
4468:  3012 7d00      push      #0x7d
446c:  b012 4245      call      #0x4542 <INT>
4470:  5f44 fcff      mov.b     -0x4(r4), r15
4474:  8f11           sxt       r15
4476:  3152           add       #0x8, sp
4478:  3441           pop       r4
447a:  3041           ret
```

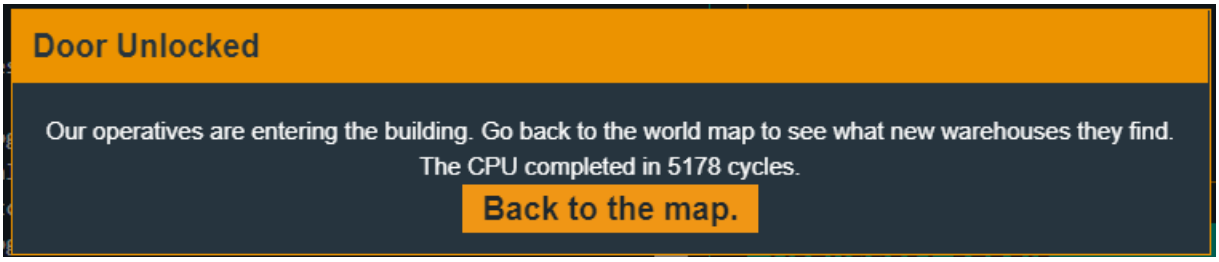
This was the test_password_valid function but if we remember from our last problem that this did not do anything. I went back to see if this function was the same as the last problem and it was the same so I decided to skip over this and to just focus on the main function.

```
4520:  b012 5244      call      #0x4452 <test_password_valid>
4524:  0f93           tst       r15
4526:  0524           jz        #0x4532 <login+0x32>
4528:  b012 4644      call      #0x4446 <unlock_door>
452c:  3f40 d144      mov       #0x44d1 "Access granted.", r15
4530:  023c           jmp       #0x4536 <login+0x36>
4532:  3f40 e144      mov       #0x44e1 "That password is not correct.", r15
4536:  b012 a645      call      #0x45a6 <puts>
453a:  3150 1000      add       #0x10, sp
453e:  3041           ret
```

I set a breakpoint after the test_password_valid_function since we know that it doesn't do much. I then stepped through the program and saw that no matter what after you inputted a password it jumps to 4532 printing out "that password is not correct."



After that, It was pretty simple to figure it out. I knew that it was a buffer overflow so all I did was enter 16 characters in hex followed by the address for the <unlock_door> function and was able to override the buffer and get in.



Johannesburg (20 pts)

```
This is Hardware  Version B.  It contains  the Bluetooth connector
built in, and two available  ports: the LockIT Pro Deadbolt should
be  connected to  port  1,  and the  LockIT  Pro  HSM-1 should  be
connected to port 2.

This is Software Revision 04. We have improved the security of the
lock by ensuring passwords that are too long will be rejected.
```

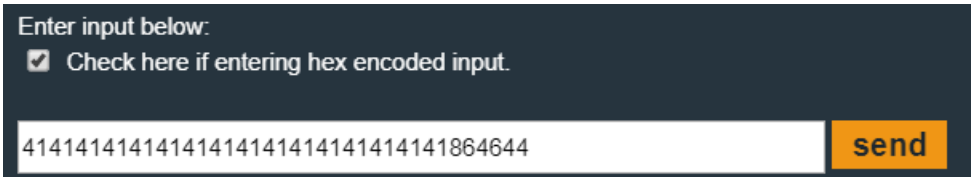
This was the overview for this level. You can see that they pathed the buffer overflows.

```
452c <login>
452c:  3150 eeff      add       #0xffee, sp
4530:  f140 8600 1100 mov.b     #0x86, 0x11(sp)
4536:  3f40 7c44      mov       #0x447c "Enter the password to continue.", r15
453a:  b012 f845      call      #0x45f8 <puts>
453e:  3f40 9c44      mov       #0x449c "Remember: passwords are between 8 and 16 characters
4542:  b012 f845      call      #0x45f8 <puts>
4546:  3e40 3f00      mov       #0x3f, r14
454a:  3f40 0024      mov       #0x2400, r15
454e:  b012 e845      call      #0x45e8 <getsn>
4552:  3e40 0024      mov       #0x2400, r14
4556:  0f41           mov       sp, r15
4558:  b012 2446      call      #0x4624 <strcpy>
455c:  0f41           mov       sp, r15
455e:  b012 5244      call      #0x4452 <test_password_valid>
4562:  0f93           tst       r15
4564:  0524           jz        #0x4570 <login+0x44>
4566:  b012 4644      call      #0x4446 <unlock_door>
456a:  3f40 d144      mov       #0x44d1 "Access granted.", r15
456e:  023c           jmp       #0x4574 <login+0x48>
4570:  3f40 e144      mov       #0x44e1 "That password is not correct.", r15
4574:  b012 f845      call      #0x45f8 <puts>
4578:  f190 8600 1100 cmp.b     #0x86, 0x11(sp)
457e:  0624           jeq       #0x458c <login+0x60>
4580:  3f40 ff44      mov       #0x44ff "Invalid Password Length: password too long.", r15
4584:  b012 f845      call      #0x45f8 <puts>
4588:  3040 3c44      br        #0x443c <__stop_progExec__>
458c:  3150 1200      add       #0x12, sp
4590:  3041           ret
```

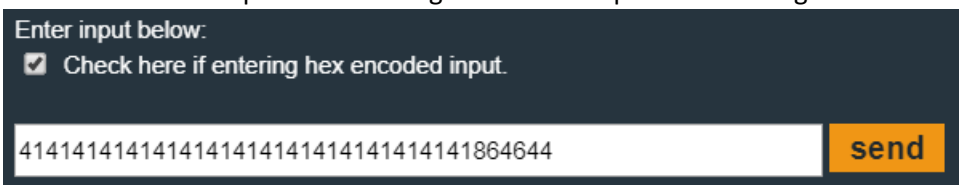
The main function again just called the login function. This looked similar to the last one we did. I saw that it included the test_password_valid function again so I did not even pay attention to that. The first thing that caught my attention was the strcpy call. Was this storing our password in another location for verification or not. I also noticed that it had the same calls from 455e to 4574 as the other programs but then at 4578 it does a compare with the stack pointer and 0x86, this seemed interesting.

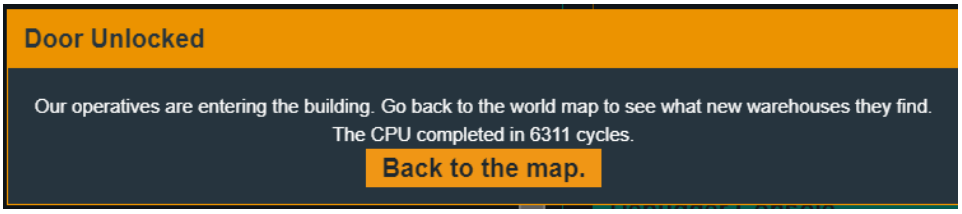
```
456e:  023c           jmp       #0x4574 <login+0x48>
4570:  3f40 e144      mov       #0x44e1 "That password is not correct.", r15
4574:  b012 f845      call      #0x45f8 <puts>
4578:  f190 8600 1100 cmp.b     #0x86, 0x11(sp)
457e:  0624           jeq       #0x458c <login+0x60>
4580:  3f40 ff44      mov       #0x44ff "Invalid Password Length: password too long.", r15
4584:  b012 f845      call      #0x45f8 <puts>
4588:  3040 3c44      br        #0x443c <__stop_progExec__>
458c:  3150 1200      add       #0x12, sp
4590:  3041           ret
```

I then set a breakpoint at that memory address. The next instruction underneath that is a jump if equal call to the memory address 0x458 which adds twelve to the sp. After stepping through the rest of program, it never executed the jeq call so that must mean that we need to have the value of 0x86 at the correct spot in our password for it to trigger. After looking at the compare for a little longer I realized that it is comparing 0x86 to the 17 (0x11) position of the stack pointer. After this, I was able to formulate a password that would possibly work.



This was the first attempt where I had 86 in the 17th position followed by the memory address of the unlock_door function. I was unable to get in. I then quickly realized that the sp starts at 0 and not 1 so I then re-entered the correct password having 86 in the 18th position starting at 1.





Conclusion:

After completing these levels I looked at Rakavjak and I tried to see if I could make some progress on it. After about 3-4 hours and not making any sustainable progress I decied that I should work on my research project instead since I need to start that early. I enjoyed trying to solve each level and will continue to try and solve more this summer.