Microcorruption - New Orleans

New Orleans

Reverse Engineering

10 Points

> The LockIT Pro a.01 is the first of a new series of locks. It is
> controlled by a MSP430 microcontroller. 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.

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

Analysis

I first analyzed the main function of the disassembly to see which other functions were being called and the general behavior of the software.

Disassembly

```
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      $+0xc <main+0x2a>
4458:  3f40 0345      mov      #0x4503 "Invalid password; try again.", r15
445c:  b012 9445      call     #0x4594 <puts>
4460:  063c           jmp      $+0xe <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
4474 <__stop_progExec__>
```

There was a function of interest that stood out immediately: `create_password`

This also connected to another portion of the challenge description as well.

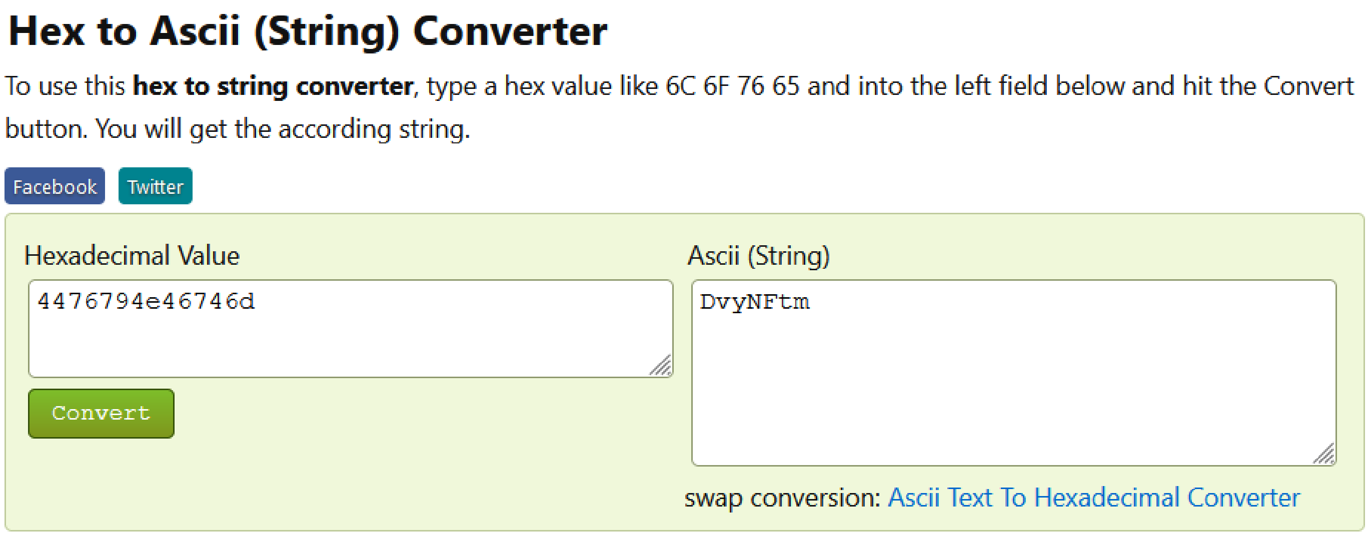
> There is no default password on the LockIT Pro---upon receiving the LockIT Pro, a new password must be set by connecting it to the LockIT Pro App and entering a password when prompted, and then restarting the LockIT Pro using the red button on the back.

This `create_password` function may be the method for storing the lock's set password input by the owner. In the function it appears a hardcoded value is stored into the r15 register:

```
447e <create_password>
447e:  3f40 0024       mov      #0x2400, r15
4482:  ff40 4400 0000 mov.b    #0x44, 0x0(r15)
4488:  ff40 7600 0100 mov.b    #0x76, 0x1(r15)
448e:  ff40 7900 0200 mov.b    #0x79, 0x2(r15)
4494:  ff40 4e00 0300 mov.b    #0x4e, 0x3(r15)
449a:  ff40 4600 0400 mov.b    #0x46, 0x4(r15)
44a0:  ff40 7400 0500 mov.b    #0x74, 0x5(r15)
44a6:  ff40 6d00 0600 mov.b    #0x6d, 0x6(r15)
44ac:  cf43 0700       mov.b    #0x0, 0x7(r15)
44b0:  3041            ret
```

It looks like the function is setting an ASCII character one byte at a time into r15.

- mov.b #0x44, 0x0(r15) // sets 0x44 at the starting byte
- mov.b #0x76, 0x1(r15) // sets 0x76 to the following byte
- the final byte 0x0 is the terminating character



Using an online [Hex to ASCII converter](), I obtained a string that potentially serves as the lock's password: DvyNFtm

Disassembly

```
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     $+0xc <main+0x2a>
4458:  3f40 0345     mov     #0x4503 "Invalid password; try again.", r15
445c:  b012 9445     call    #0x4594 <puts>
4460:  063c          jmp     $+0xe <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
4474 <__stop_progExec__>
```

After generating the hardcoded password, the software will obtain the users' input password and then call `check_password` :

```
44bc <check_password>
44bc:  0e43          clr     r14
44be:  0d4f          mov     r15, r13
44c0:  0d5e          add     r14, r13
44c2:  ee9d 0024     cmp.b   @r13, 0x2400(r14)
44c6:  0520          jnz     $+0xc <check_password+0x16>
44c8:  1e53          inc     r14
44ca:  3e92          cmp     #0x8, r14
44cc:  f823          jnz     $-0xe <check_password+0x2>
44ce:  1f43          mov     #0x1, r15
44d0:  3041          ret
44d2:  0f43          clr     r15
44d4:  3041          ret
```

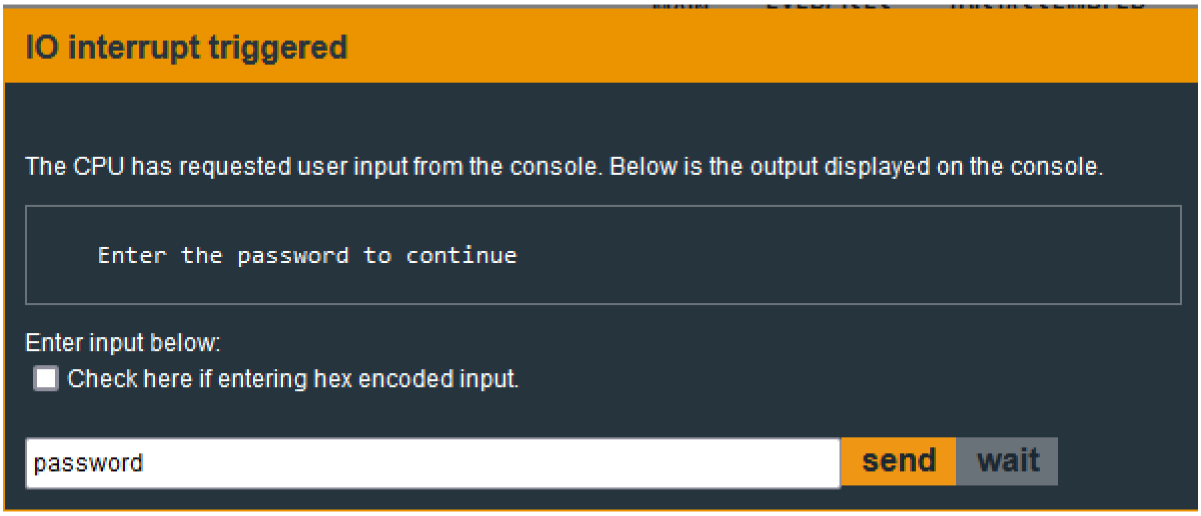
This function is interesting. A counter variable is used to iterate through the bytes stored in memory starting from 0x2400 and compare with the values stored at the address of register r13.

First the register r14 is cleared, and the value of r15 is placed into r13. Then the counter is added to r13, this moves the value to the next byte. Lastly, the current byte of r13 is compared with the i_{th} byte from address 0x2400 where i is the counter value r14.

This is of note because the password must also be stored in memory at the address 0x2400. This can investigated using breakpoints and the live memory dump.

```
44bc <check_password>
44bc:  0e43           clr     r14
44be:  0d4f           mov     r15, r13
44c0:  0d5e           add     r14, r13
44c2:  ee9d 0024      cmp.b   @r13, 0x2400(r14)
44c6:  0520           jnz     $+0xc <check_password+0x16>
44c8:  1e53           inc     r14
44ca:  3e92           cmp     #0x8, r14
44cc:  f823           jnz     $-0xe <check_password+0x2>
44ce:  1f43           mov     #0x1, r15
44d0:  3041           ret
44d2:  0f43           clr     r15
44d4:  3041           ret
44d6 <unlock_door>
```

To break just before this comparison, I executed `break 44bc`.



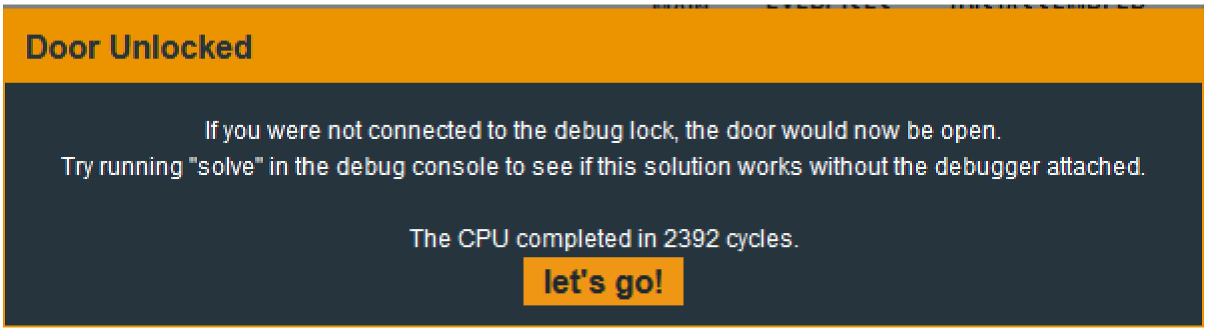
Then after continuing the program a password prompt was given. I entered a temporary value *password* to move past this for now.

Live Memory Dump Download

```
0000: 0000 4400 0000 0000 0000 0000 0000 0000   ..D.............
0010: 3041 0000 0000 0000 0000 0000 0000 0000   0A..............
0020: 0000 0000 0000 0000 0000 0000 0000 0000   ................
0030: *
0150: 0000 0000 0000 0000 0000 0000 085a 0000   .............Z..
0160: 0000 0000 0000 0000 0000 0000 0000 0000   ................
0170: *
2400: 4476 794e 4674 6d00 0000 0000 0000 0000   DvyNFtm.........
2410: 0000 0000 0000 0000 0000 0000 0000 0000   ................
2420: *
4380: 0000 0000 0000 0000 0000 0000 4445 0000   ............DE..
4390: 8e45 0200 9c43 6400 ba44 5444 7061 7373   .E...Cd..DTDpass
43a0: 776f 7264 0000 0000 0000 0000 0000 0000   word............
43b0: 0000 0000 0000 0000 0000 0000 0000 0000   ................
43c0: *
```

Once inside the `check_password` function, I saw the same value found earlier from the ASCII characters at the memory address 0x2400.

Solution



After resetting the CPU and inputting the found password into the prompt, the door successfully unlocked! I then entered `solve` in the terminal to connect to the remote lock and completed the New Orleans challenge.