Write-up: BLUE - Binary London Underground Entry

Description

This challenge is about analyzing a local Linux program and use the knowledge to find out the password in the online version of the program.

Please compile and analyze the given `blue.s` on your local Linux system and use the secret password against the online version of `blue`.

Find out the secret password the service will accept by analyzing the local source assembly or self-compiled binary `blue`.

Solution

The first step was to compile the binary with gcc.

After that I started analysing it with Ghydra. This way it was easy to find the main function, but the code looked very unreadable. After a couple of hours with no success I tried it with gdb. I set a breakpoint at the main function and stepped through the whole binary. This way I got an abstract idea of the program:

```
26  pcVar3 = fgets((char *)&local_48,0x14,stdin);
27  puVar5 = &local_48;
28  if (pcVar3 != (char *)0x0) {
29    do {
30      puVar4 = puVar5;
31      uVar2 = *puVar4 + 0xfefefeff & ~*puVar4;
32      _bVar1 = uVar2 & 0x80808080;
33      bVar1 = (byte)_bVar1;
34      puVar5 = puVar4 + 1;
35    } while (_bVar1 == 0);
```

On line 26, the string is read. Right after that it is checked if the return value of fgets is a null byte. This just checks if the user inputted something. After that it iterates over a set of 4 characters and does some calculations with it. I still do not quite understand what it does, but if I inputted a string with a length of a multiple of 4 (including the line feed at the end) I was able to proceed further into the code.

```
44      if (((((long)puVar5 + ((-3 - (ulong)CARRY1(bVar1,bVar1)) - (long)&local_48) == 0x10) &&
45           (local_48 == 'l')) && (cStack71 == 'i')) && (cStack70 == 'c')) {
```

10 lines further down is the first indicator to the goal. Here we can see that three variables are checked to be certain characters. While debugging with gdb, it can be seen that these are the first three characters of string we want to find.

```
47        uVar7 = 0;
48        iVar10 = 0;
49        iVar8 = 0;
50        iVar11 = 0x6c;
51        iVar6 = 0x6c;
52        iVar9 = 0;
53        while( true ) {
54          if ((0xc < uVar7) || (((ulong)&local_48 >> (uVar7 & 0x1f) & 1) == 0)) {
55            iVar8 = iVar8 + iVar6;
56          }
57                        /* check if uVar7 is even */
58          if ((uVar7 & 1) == 0) {
59            iVar10 = iVar10 + iVar6;
60          }
61          iVar9 = iVar9 % 0x100;
62          iVar8 = iVar8 % 0x100;
63          iVar10 = iVar10 % 0x100;
64          if (uVar7 == 0xe) break;
65          iVar6 = (int)*(char *)(&local_48 + uVar7 + 1);
66          iVar11 = iVar11 % 0x100 + iVar6;
67          if (((int)uVar7 + 1) % 3 == 1) {
68            iVar9 = iVar9 + iVar6;
69          }
70          uVar7 = uVar7 + 1;
71        }
```

At this point we know that the string starts with “lic” and probably has a length of a multiple of 4. Now the rest of the input gets processed. If we analyse the while loop, it can be seen that it is executed exactly 14 times (line 64).

Variable	Explanation
iVar6	Storage for the current character that is being processed.
uVar7	A counter, which tracks how often the loop was executed and thus iterates over the string.
iVar8	A “checksum”, which is calculated by adding the ascii value of some characters of the string. Till the end I was not quite sure which characters it was using.
iVar9	A “checksum”, which is calculated by adding the ascii value of every third character (as seen in line 67) of the input string. But only the first two bytes are used, thus the modulo 0x100 is taken.
iVar10	Same as iVar9, but every second character is used.
iVar11	Same as iVar9, but every character is used.

```
if (((iVar11 % 0x100 == 0xd0) && (iVar9 == 0xaa)) && ((iVar8 == 0xb3 && (iVar10 == 0xaa)))) {
  puts("SUCCESS! The flag is: changeme_on_prod");
  fflush(stdout);
  return 0;
}
```

Now the “checksums” are used and checked if they have a certain value. If true, the user gets the flag.

To calculate the needed input string, I used a python script and a bit of trial and error. In the end it took me many hours to get to this point, but I learned quite a lot about reverse engineering.

My input string: `licRa(*[DL*Ah[z`

The flag: `hl{welc0me_to_th3_undergr0und}`