Challenge: FlagCasino

Challenge Description :

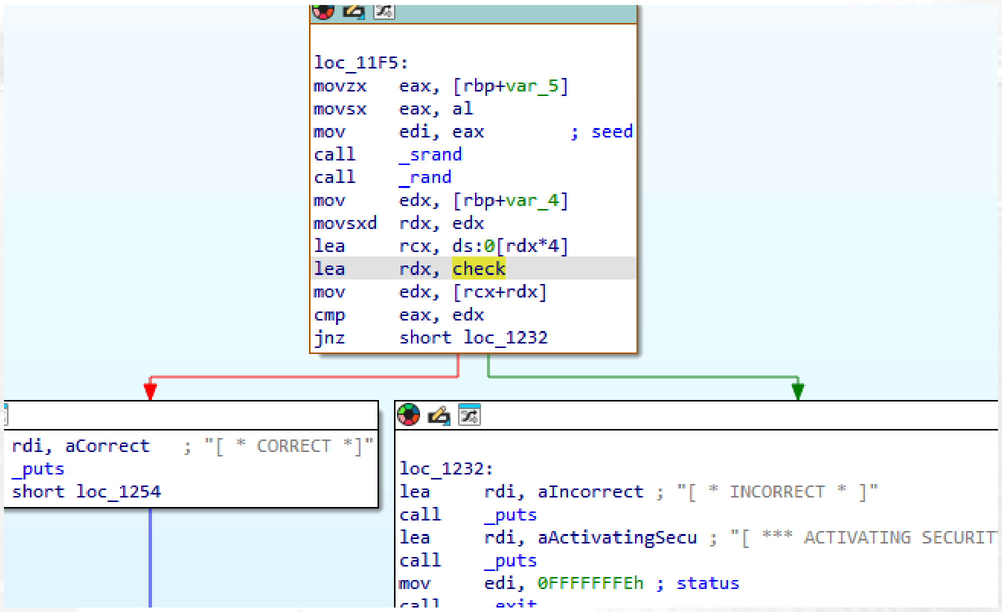
The team stumbles into a long-abandoned casino. As you enter, the lights and music whir to life, and a staff of robots begin moving around and offering games, while skeletons of prewar patrons are slumped at slot machines. A robotic dealer waves you over and promises great wealth if you can win - can you beat the house and gather funds for the mission?

Context :

- You are given a compiled binary file, when looking through the file we can find a selection of 120 bytes of data that we need to format correctly, after formatting correctly we need to structure it and send the data to the IP:PORT that's hosting the binary file to gain the flag.

Flag :

- First I downloaded the file and opened it up with Hex-Ida, with much luck i didn't really find anything too interesting, i did find a bunch of data located in the variable [Check].



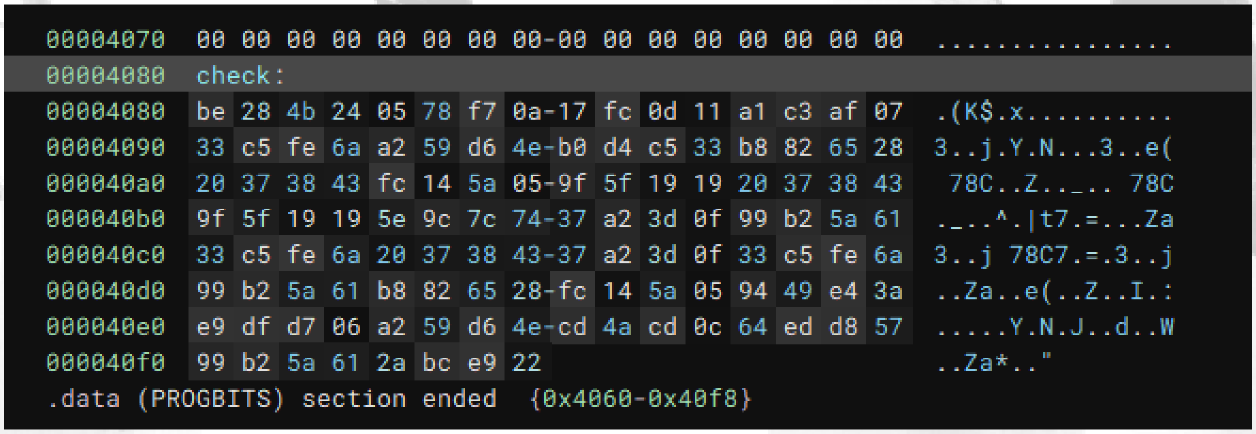
```
loc_11F5:
movzx   eax, [rbp+var_5]
movsx   eax, al
mov     edi, eax        ; seed
call    _srand
call    _rand
mov     edx, [rbp+var_4]
movsxd  rdx, edx
lea     rcx, ds:0[rdx*4]
lea     rdx, check
mov     edx, [rcx+rdx]
cmp     eax, edx
jnz     short loc_1232
```

```
rdi, aCorrect    ; "[ * CORRECT *]"
_puts
short loc_1254
```

```
loc_1232:
lea     rdi, aIncorrect ; "[ * INCORRECT * ]"
call    _puts
lea     rdi, aActivatingSecu ; "[ *** ACTIVATING SECURIT
call    _puts
mov     edi, 0FFFFFFFEh ; status
call    _exit
```

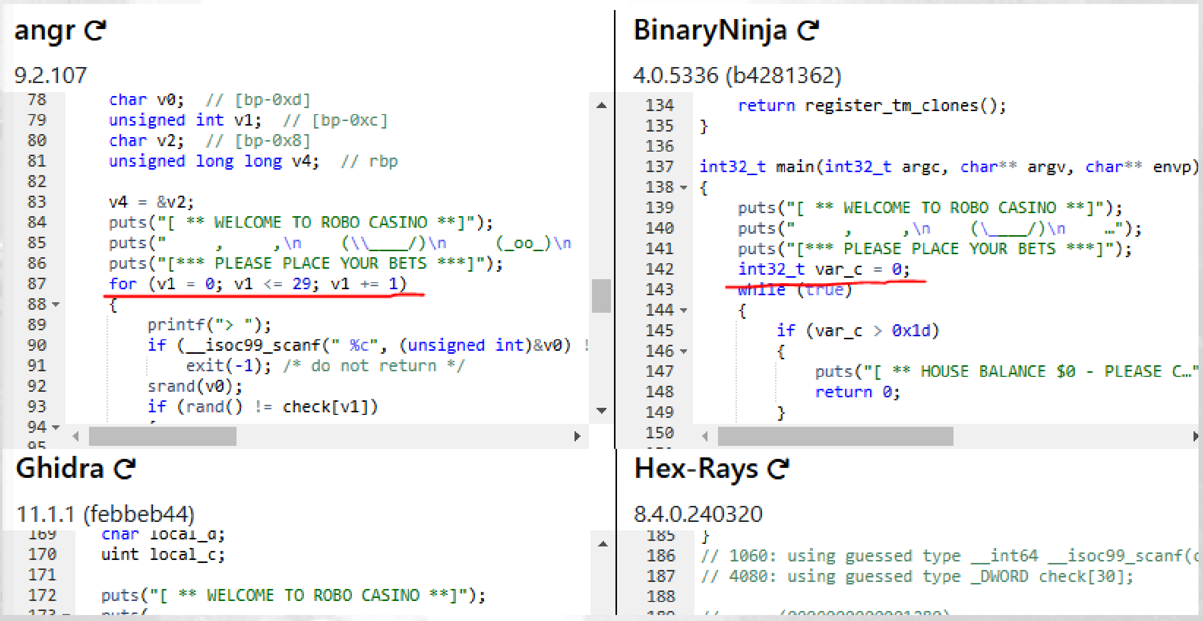
- To get a better look and understanding of the file I opened it up with Binary ninja. With it I went to the main function and double checked the data in the [Check] variable .

- The data within the [Check] variable could be the data I need to input within the Binary file.



```
00004070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
00004080  check:
00004080  be 28 4b 24 05 78 f7 0a-17 fc 0d 11 a1 c3 af 07  .(K$.x..........
00004090  33 c5 fe 6a a2 59 d6 4e-b0 d4 c5 33 b8 82 65 28  3..j.Y.N...3..e(
000040a0  20 37 38 43 fc 14 5a 05-9f 5f 19 19 20 37 38 43   78C..Z.._.. 78C
000040b0  9f 5f 19 19 5e 9c 7c 74-37 a2 3d 0f 99 b2 5a 61  ._..^.|t7.=...Za
000040c0  33 c5 fe 6a 20 37 38 43-37 a2 3d 0f 33 c5 fe 6a  3..j 78C7.=.3..j
000040d0  99 b2 5a 61 b8 82 65 28-fc 14 5a 05 94 49 e4 3a  ..Za..e(..Z..I.:
000040e0  e9 df d7 06 a2 59 d6 4e-cd 4a cd 0c 64 ed d8 57  .....Y.N.J..d..W
000040f0  99 b2 5a 61 2a bc e9 22                          ..Za*.."
.data (PROGBITS) section ended  {0x4060-0x40f8}
```

- Trying to find more information I went to [Dogbolt.org] to decompile it further. I finally found what I needed, I just needed a different decompiler to see it.



angr

9.2.107

```
    char v0;  // [bp-0xd]
    unsigned int v1;  // [bp-0xc]
    char v2;  // [bp-0x8]
    unsigned long long v4;  // rbp

    v4 = &v2;
    puts("[ ** WELCOME TO ROBO CASINO **]");
    puts("     ,     ,\n    (\\____/)\n     (_oo_)\n
    puts("[*** PLEASE PLACE YOUR BETS ***]");
    for (v1 = 0; v1 <= 29; v1 += 1)
    {
        printf("> ");
        if (__isoc99_scanf(" %c", (unsigned int)&v0) !
            exit(-1); /* do not return */
        srand(v0);
        if (rand() != check[v1])
```

BinaryNinja

4.0.5336 (b4281362)

```
    return register_tm_clones();
}

int32_t main(int32_t argc, char** argv, char** envp)
{
    puts("[ ** WELCOME TO ROBO CASINO **]");
    puts("     ,     ,\n    (\____/)\n    …");
    puts("[*** PLEASE PLACE YOUR BETS ***]");
    int32_t var_c = 0;
    while (true)
    {
        if (var_c > 0x1d)
        {
            puts("[ ** HOUSE BALANCE $0 - PLEASE C…"
            return 0;
        }
```

Ghidra

11.1.1 (febbeb44)

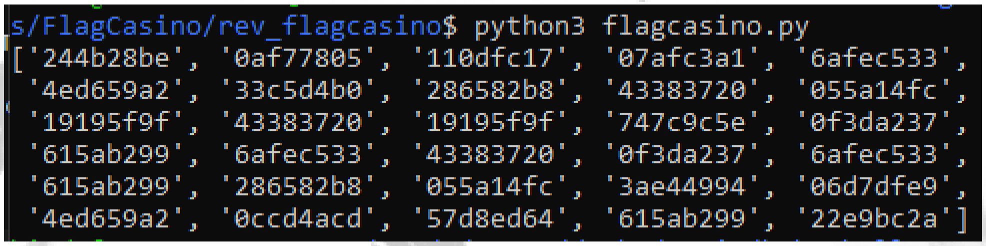
```
  char local_d;
  uint local_c;

  puts("[ ** WELCOME TO ROBO CASINO **]");
```

Hex-Rays

8.4.0.240320

```
}
// 1060: using guessed type __int64 __isoc99_scanf(
// 4080: using guessed type _DWORD check[30];
```

- The Max amount of characters must be 30 because the loop variable **i** stops at 30. Given that there are 120 bytes of data, this means that each check corresponds to 4 bytes. To process the data effectively, we can divide the string into segments of 4 bytes each.

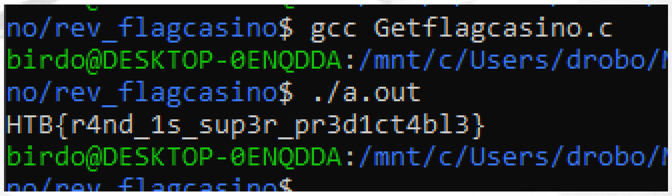


```
s/FlagCasino/rev_flagcasino$ python3 flagcasino.py
['244b28be', '0af77805', '110dfc17', '07afc3a1', '6afec533',
 '4ed659a2', '33c5d4b0', '286582b8', '43383720', '055a14fc',
 '19195f9f', '43383720', '19195f9f', '747c9c5e', '0f3da237',
 '615ab299', '6afec533', '43383720', '0f3da237', '6afec533',
 '615ab299', '286582b8', '055a14fc', '3ae44994', '06d7dfe9',
 '4ed659a2', '0ccd4acd', '57d8ed64', '615ab299', '22e9bc2a']
```

- Instead of converting it manually, let's create a simple python script to achieve this, which will also convert the bytes into a little-endian format.

  [Link]

- It is not read-able but it works. We will need to test every number / letter between 33 and 126, to brute-force the integers to be converted to ASCII characters, which will hopefully give us a better result.



```
no/rev_flagcasino$ gcc Getflagcasino.c
birdo@DESKTOP-0ENQDDA:/mnt/c/Users/drobo/
no/rev_flagcasino$ ./a.out
HTB{r4nd_1s_sup3r_pr3d1ct4bl3}
birdo@DESKTOP-0ENQDDA:/mnt/c/Users/drobo/
no/rev_flagcasino$
```

- Below is the link for the script to gain the flag by converting the little-endian format to ascii characters : HTB{r4nd_1s_sup3r_pr3d1ct4bl3}

  [Link]