FlagCasino

22nd April 2022 / Document No. D24.102.60

Prepared By: `clubby789`

Challenge Author: `clubby789`

Difficulty: Very Easy

Classification: Official

Synopsis

- FlagCasino is a Very Easy reversing challenge. Players will extract a series of integers from a binary, then use the predictable behavior of `rand()` to recover the flag.

Skills Required

- Decompiler usage

Skills Learned

- Using `rand` from Python

Solution

Executing the binary, we're prompted for input.

```
[ ** WELCOME TO ROBO CASINO **]
     ,     ,
    (\____/)
     (_oo_)
       (O)
     __||__    \)
```

```
  []/______\[] /
  / \______/ \/
 /    /__\
(\   /____\
---------------------
[*** PLEASE PLACE YOUR BETS ***]
> 1
[ * INCORRECT * ]
[ *** ACTIVATING SECURITY SYSTEM - PLEASE VACATE *** ]
```

We'll open the binary in a decompiler to uncover the behavior.

Analysis

```
int32_t main(int32_t argc, char** argv, char** envp)
  puts(str: "[ ** WELCOME TO ROBO CASINO **]")
  puts(str: "      ,      ,\n    (\____/)\n     (_oo_)\n       (O)\n     __…")
  puts(str: "[*** PLEASE PLACE YOUR BETS ***]")
  int32_t i = 0
  while (true) {
      if (i u> 0x1c) {
          puts(str: "[ ** HOUSE BALANCE $0 - PLEASE COME BACK LATER ** ]")
          return 0
      }
      printf(format: "> ")
      char inp
      if (__isoc99_scanf(format: " %c", &inp) != 1) {
          exit(status: 0xffffffff)
          noreturn
      }
      srand(x: sx.d(inp))
      if (rand() != check[sx.q(i)]) {
          break
      }
      puts(str: "[ * CORRECT *]")
      i = i + 1
  }
  puts(str: "[ * INCORRECT * ]")
  puts(str: "[ *** ACTIVATING SECURITY SYSTEM - PLEASE VACATE *** ]")
  exit(status: 0xfffffffe)
```

After printing out the banner, the binary iterates from `0` to `0x1c`. At each step, `srand()` is called on the character we input. We then call `rand()` and check the result against a corresponding integer from `check`. If it matches we get a success message then continue.

If all `0x1c` entries are correct, we exit successfully.

Solving

`rand()` is a predictable random number generator - calling `srand(x)` then `rand()` will always give us the same result. We'll first write a script to discover the mapping.

Using the `ctypes` library, we can call C functions from Python.

```
import ctypes

libc = ctypes.CDLL('libc.so.6')

mapping = {}
for i in range(255):
    libc.srand(i)
    mapping[libc.rand()] = chr(i)
```

We'll then use `pwntools` to open up the binary and read out each desired integer.

```
from pwn import *

flag = ""

e = ELF("./casino", checksec=False)
for j in range(29):
    # Index into the `check` array by multiplying the index by 4
    # Use `e.u32()` to extract the 32-bit integer
    val = e.u32(e.sym["check"] + j * 4)
    flag += mapping[val]
print(flag)
```

Running this against the binary gives us the flag.