Challenge: Writing on the Wall

# Challenge Description :

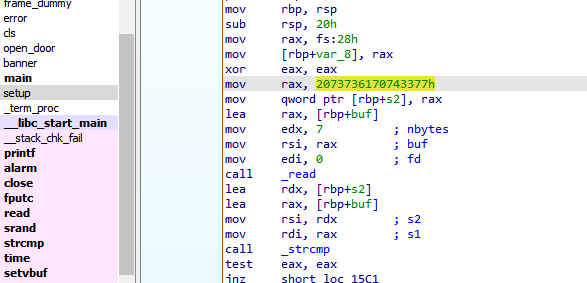
**As you approach a password-protected door, a sense of uncertainty envelops you—no clues, no hints. Yet, just as confusion takes hold, your gaze locks onto cryptic markings adorning the nearby wall. Could this be the elusive password, waiting to unveil the door's secrets?**

# Context :

## You are given a compiled binary file that we need to decompile, during decompiling the file you might find that you need to exploit this file when running it

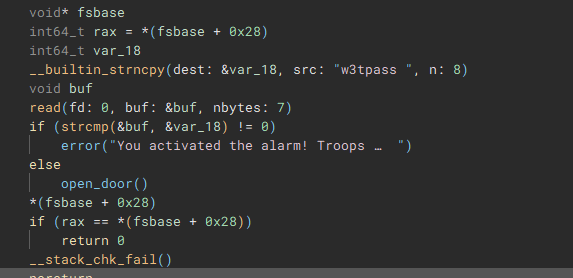
# Flag :

* **First downloading the source files. We are given a binary file named ”writing\_on\_the\_wall”. I first opened it with IDA, with it i found a interesting line:**

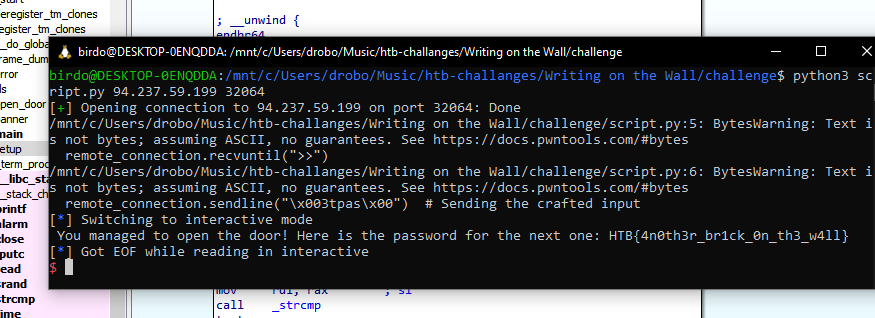
****

* **The hexadecimal value [ 0x2073736170743377 ] corresponds to string [ssapt3w ] and when reversed is [ w3tpass ]. We read 7 bytes into local\_1e, which is only 6 bytes in size. This means that if we input the string [w3tpass ], a null byte will be written at the start of local\_18.**

* **Our goal is to make local\_1e equal to local\_18, Knowing that the first character of local\_18 is a null byte, we can input \x003tpas\x00 to make both strings start with \0.**
* **Using Binary Ninja to get a better understanding we see..**

****

* **Looking at this we can get a much better understanding, the strcmp is trying to get the input to equal the local\_18 [ var\_18 ].**
* **If both are correct the door will open, the destination for the local\_18 variable is above it, as the source for the \_\_builtin\_strncpy.**
* **We will need to run a python script to exploit this to make it easier on us. As we will need to input null bytes. it's easier and more reliable to run a script to print it out properly.**



* **Using the python Pwntools import i was able to make a Running script that will hopefully give us the flag.**

import pwn

import sys

def exploit\_binary(remote\_connection: pwn.remote):

remote\_connection.recvuntil(">>")

remote\_connection.sendline("\x003tpas\x00") # Sending the crafted input

remote\_connection.interactive() # Interact with the shell

def establish\_connection():

if len(sys.argv) != 3:

print(f"Usage: {sys.argv[0]} REMOTE remote-ip remote-port")

sys.exit(1)

return pwn.remote(sys.argv[1], sys.argv[2]) # Establishing remote connection

def main():

remote\_connection = establish\_connection() # Connect to the remote service

exploit\_binary(remote\_connection) # Execute the exploit

if \_\_name\_\_ == "\_\_main\_\_":

main() # Run the script

* **The Flag is finally given as is printed out as:**

**HTB{4n0th3r\_br1ck\_0n\_th3\_w4ll}**