Computer Security Capstone

Project IV: Capture The Flag (CTF)

Chi-Yu Li (2022 Spring)

Computer Science Department

National Yang Ming Chiao Tung University

Goal

- Understand the exploitation of basic programming bugs, Linux system knowledge, and reverse-engineering

- You will learn about
  - ☐ Solving basic CTF problems
  - ☐ Investigating C/Linux functions deeply instead of simply using them
  - ☐ What buggy codes are and how they can be exploited

What is CTF?

From Wikipedia

- A traditional outdoor game
  - Two teams each have a flag
  - Objective: to capture the other team's flag
- In computer security, it is a type of cryptosport: a computer security competition
  - Giving participants experience in securing a machine
  - Required skills: reverse-engineering, network sniffing, protocol analysis, system administration, programming, etc.
  - How?
    - A set of challenges is given to competitors
    - Each challenge is designed to give a "Flag" when it is countered

A CTF Example

- A toy CTF

$ python -c 'v = input(); ***print("flag:foobar") if v == "1"*** else print("failed")'

- You should enter “1” to pass the *if* statement and get the flag (flag:foobar)
- Otherwise, “failed” is obtained

Requirements

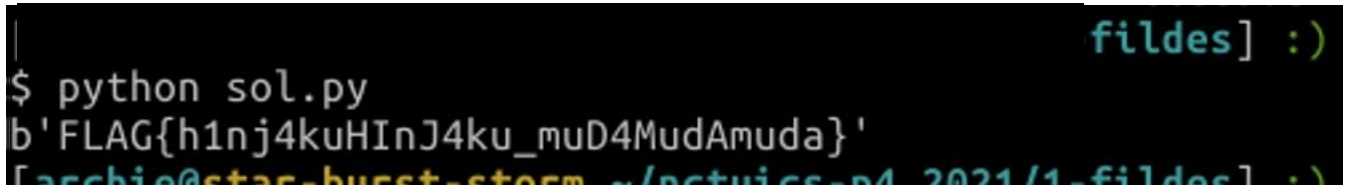
- Linux/Unix environment is required
  - Connecting to our CTF servers using ‘nc’ for all the tasks except Task I-2, Task I-3
  - Solving Task I-2, Task I-3 locally
- You are NOT allowed to team up: one student one team
  - Discussions are allowed between teams, but any collaboration is prohibited
- TA: Kai-Wen Chen

How to Proceed?

- Connecting to each CTF server: nc <ip> <port>
  - IP: 140.113.207.246
  - Port is given at each problem
  - The program of each problem runs as a service at the server
  - You can do whatever you are allowed to do

How to Proceed? (Cont.)

- For each CTF problem, you should
  - Analyze its given executable files or source code files
  - Interact with the server to get a flag
  - The flag format: CSC2022{xxx}



What If Get Stuck?

- Learn to use “man” in UNIX-like systems
  - If you don’t know something, ask “man”
  - e.g., what is man?
    - $ man man

- Learn to find answers with FIRST-HAND INFORMATION/REFERENCE
  - Google is your best friend (Using ENGLISH KEYWORDS!!)
  - First-hand information: Wikipedia, cppreference.com, devel mailing-list, etc.
  - First-hand reference: papers, standards, spec, man, source codes, etc.
  - Second-hand information: blog, medium, ptt, reddit, stackoverflow post, etc.

Two Tasks

- Task I: Basic CTF problems (70%)
- Task II: CTF beginners (30%)
- Download all given executable and source files from the following link
  - http://140.113.207.246:20220

Task I: Basic CTF Problems

- Task I-1: Fildes (20%)
- Task I-2: Timeout (20%)
- Task I-3: cyclicRNG (30%)

Task I-1: Fildes (20%)

- Goal: Learn about Linux fd & standard I/O streams
- Description
  - Read the code carefully and use your knowledge about *read()* function
  - Read the *read()* man page and find a way to solve this
- Server port: 20221
- Hints
  - $ man stdin
  - $ man 2 read
  - $ man 2 atoi

Task I-2: Timeout (20%)

- Goal: Learn to use tools to inspect binary file and library injection
- Description
  - The FLAG will be printed at the end of the program
  - You can start from having a look at the objects inside the binary file
- No server port; solving it locally
- Recommended tools
  - objdump: display object information from a binary file
  - strace: display what functions are used in the program
  - LD_PRELOAD: Library injection
  - GDB - PEDA:
    - Python Exploit Development Assistance for GDB (https://github.com/longld/peda)

Task I-3: cyclicRNG (30%)

- Goal: Try to decrypt the ciphertext
- Description:
  - The PRNG is predictable which will cause the ciphertext easy to be decrypted
- No server port; solving it locally

- Hints
  - Find out the output pattern of the given pseudo random number generator

Task II: CTF Beginner

- Task II-1: Teleportation (10%)
- Task II-2: Notes (10%)
- Task II-3: Secret (10%)

Task II-1: Teleportation (10%)

- Goal: Learn to identify buffer overflow in source codes and overwrite function pointers stored on stack
- Description: Want to know how to teleport to any place you want? Then you got to try this! Give me some spell and the address you want to go, let the magic bring you there!

- Server port: 20224

Task II-1: Teleportation (cont.)

- Recommended tools
  - pwntools (pip install pwntools): a useful python module for pwn
  - GDB - PEDA:
    - Python Exploit Development Assistance for GDB (https://github.com/longld/peda)
    - (gdb-peda) info functions CAN HELP

- Hints
  - No canary
  - No PIE: the address observed in the executable binary file is the virtual address of the process when it's executed

Task II-2: Notes (10%)

- Goal: Learn the vulnerability of ROP(Return Oriented Programming) chaining when only open/read/write is available
- Description: Despite constraining the system call to only open/read/write, data leakage is still possible
- Server port: 20225
- Recommended tools
  - pwntools (pip install pwntools): a useful python module for pwn
  - ROPgadget: helps you find useful instruction sets for rop chain

pop rdi ; ret	pop rsi ; ret
pop rdx ; ret	pop rax ; ret
syscall ; ret	leave ; ret

  - GDB - PEDA:
    - Python Exploit Development Assistance for GDB

Task II-2: GOT (cont.)

- Hints
  - No PIE
  - Learning material: https://www.ired.team/offensive-security/code-injection-process-injection/binary-exploitation/rop-chaining-return-oriented-programming

```
File  Actions  Edit  View  Help
┌──(kali㉿kali)-[~/labs/rop1]
└─$ ./rop1b "$(python -c 'print "A"*108 + "BBBB" + "\xb9\x61\x55\x56" + "\xe4\x61\x55\x56" + "\x1e\x60\x
55\x56" + "\xef\xbe\xef\xbe" + "\x12\x62\x55\x56" + "\x0a\x63\x55\x56" + "\xad\xde\xad\xde" + "\xd3\xc0\
xd3\xc0" + "\x50\x29\xe0\xf7" ')"
ROP 1!
ROP 2: beefbeef!
ROP 3: deaddead, c0d3c0d3!

┌──(kali㉿kali)-[~/labs/rop1]
└─$
```

ROP Chaining: Return Oriented Programming

Task II-3: Secret (10%)

- Goal: learn to run shellcode with buffer overflow
- Description: Like I said, format is a double-edged sword, did you see any in the code again? If you want to get the secret, you will have to pass my test! Give me your words and tell me what you want me to do, I might let you pass the test.

- Server port: 20226

Task II-3: Secret

- Hints#1
  - No canary
  - NX (No-eXecute) is disabled: executing instruction on memory for data storage is possible
  - No PIE: the address observed in the executable binary file is the virtual address of the process when it’s executed

Task II-3: Secret (Cont.)

- Recommended tools
  - pwntools (pip install pwntools): useful python module for pwn
  - objdump: display information in object files
  - GDB - PEDA:
    - Python Exploit Development Assistance for GDB (https://github.com/longld/peda)
- Hints#2
  - Can you find out the address of the beginning of the buf(see the source code)?
  - Another option to generate shellcode, please check the following website
    - http://shell-storm.org/shellcode/ , the version of my machine is linux/x86-64

Task II-3: Secret (Cont.)

- An example: run shellcode using pwntools to get a flag
  - ☐ cd sample-shellcode
  - ☐ python3 sol.py
  - ☐ cat flag

File: **sol.py**

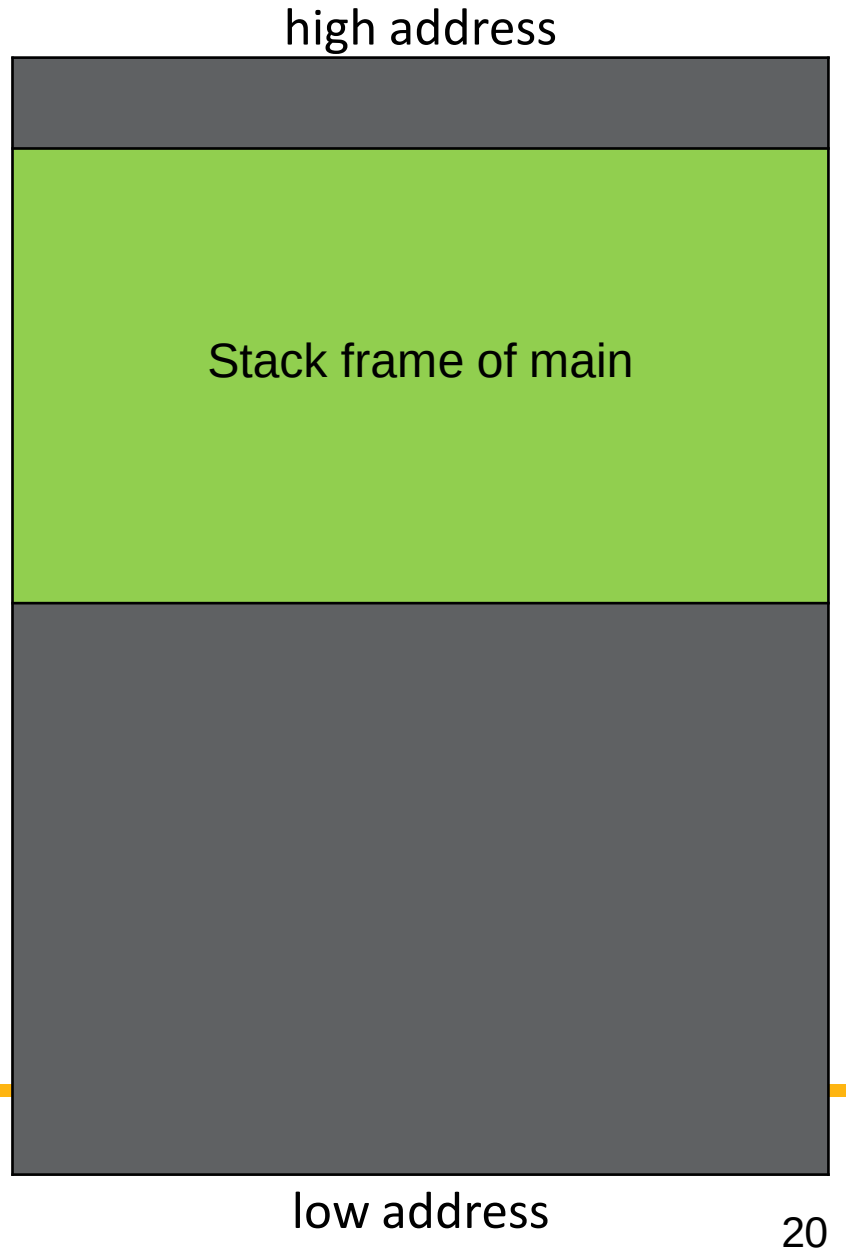
```
1  from pwn import *
2  context.arch = 'amd64'
3  p = process('./shellcode')
4  # To connect to tcp server
5  # p = remote('ip', port)
6  shellcode = asm(shellcraft.amd64.linux.sh())
7  p.send(shellcode)
8  p.interactive()
```

Machine code

Assembly

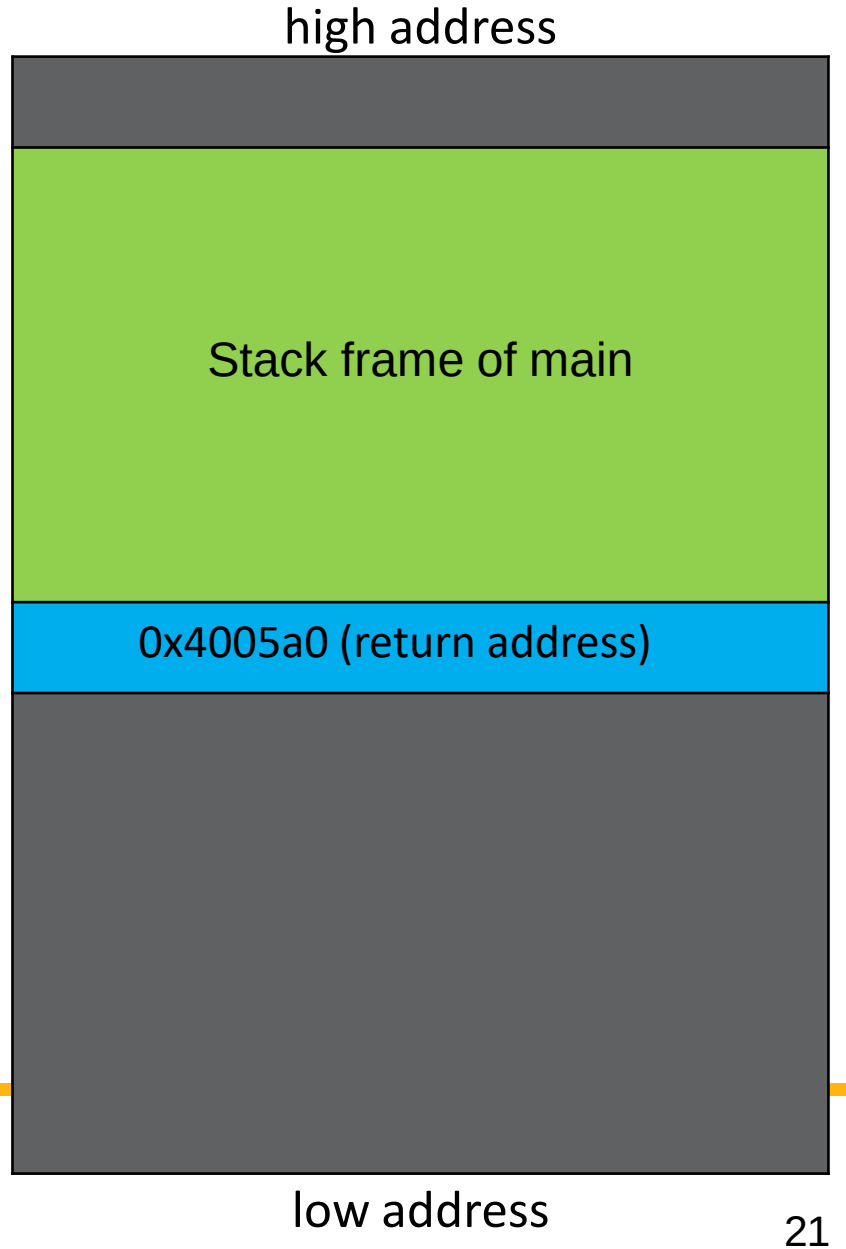
Example: Stack frame during a function call

```
                                                    rbp →
func:
        push rbp
        mov rbp, rsp
        sub rsp, 0x30          Call fun = push next_rip
        …                                 jmp func
        move eax, 0x0                               rsp →
        leave
        ret
main:
        …
rip →   call func
        mov eax, 0x0 // address 0x4005a0
        …
```



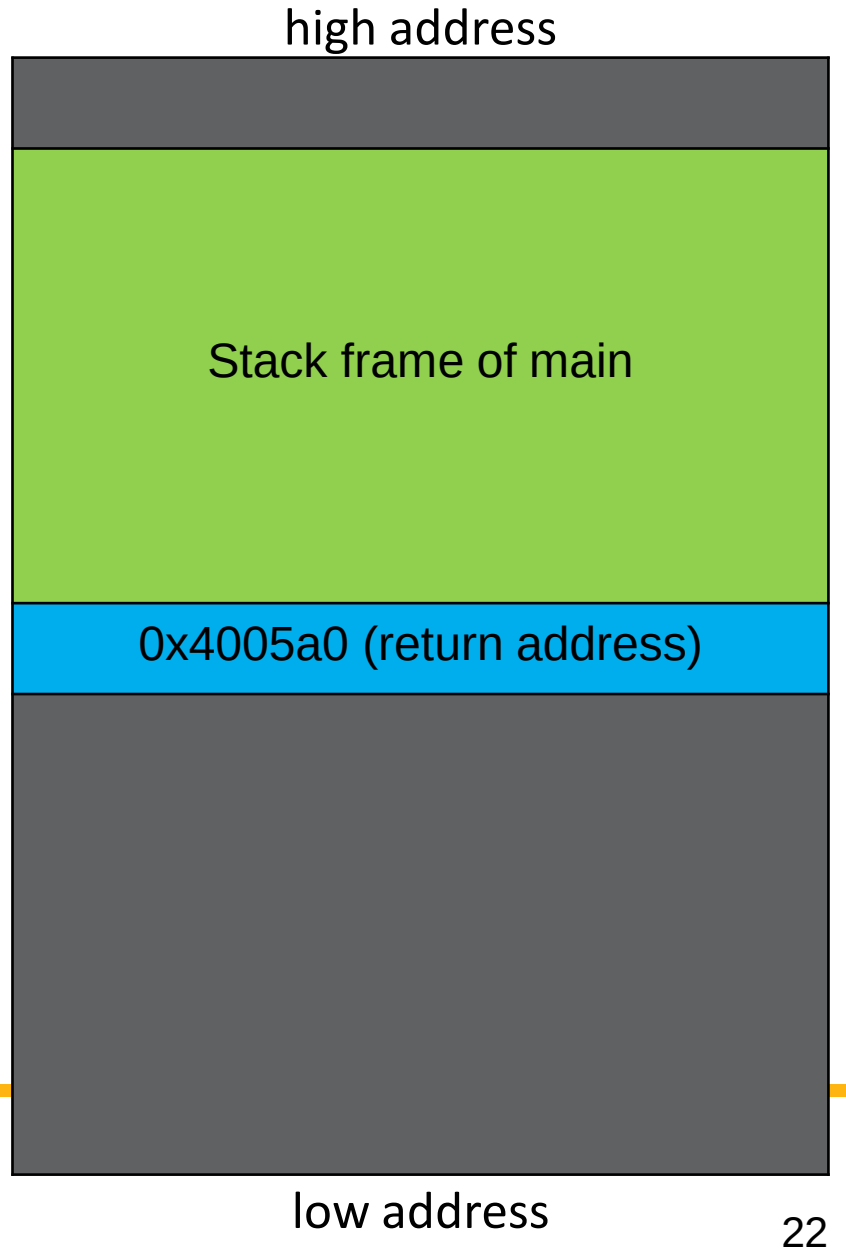
Example: Stack frame during a function call

```
                                                    rbp →
func:
        push rbp
        mov rbp, rsp
        sub rsp, 0x30          Call fun = push next_rip
        …                                jmp func
        move eax, 0x0                               rsp →
        leave                                       rsp →
        ret
main:
        …
rip →   call func
        mov eax, 0x0 // address 0x4005a0
        …
```



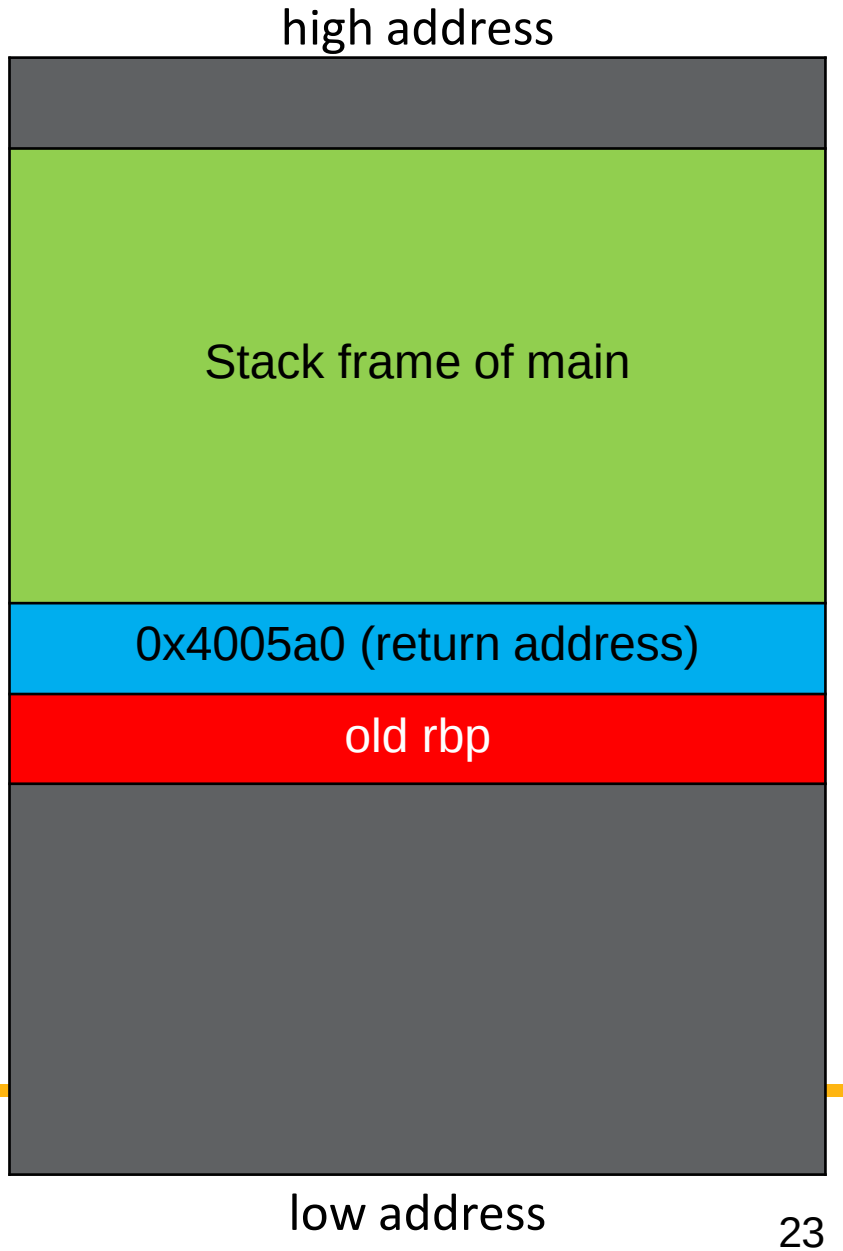
Example: Stack frame during a function call

```
                                        rbp →
func:
rip →   push rbp
        mov rbp, rsp
        sub rsp, 0x30
        …
        move eax, 0x0
        leave
                                        rsp →
        ret
main:
        …
        call func
        mov eax, 0x0 // address 0x4005a0
        …
```



Example: Stack frame during a function call

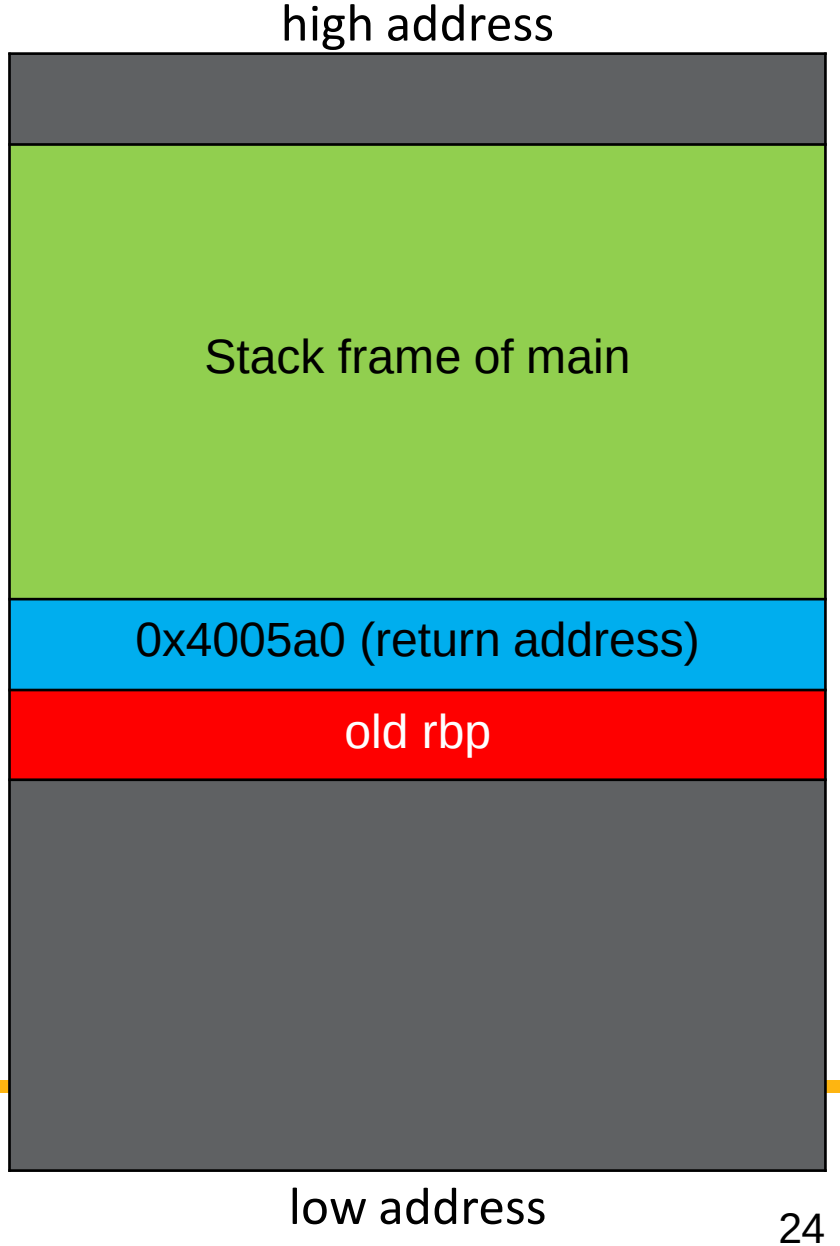
```
                                  rbp →
func:
        push rbp
rip →   mov rbp, rsp
        sub rsp, 0x30
        …
        move eax, 0x0
        leave
        ret                       rsp →
main:
        …
        call func
        mov eax, 0x0 // address 0x4005a0
        …
```



Example: Stack frame during a function call

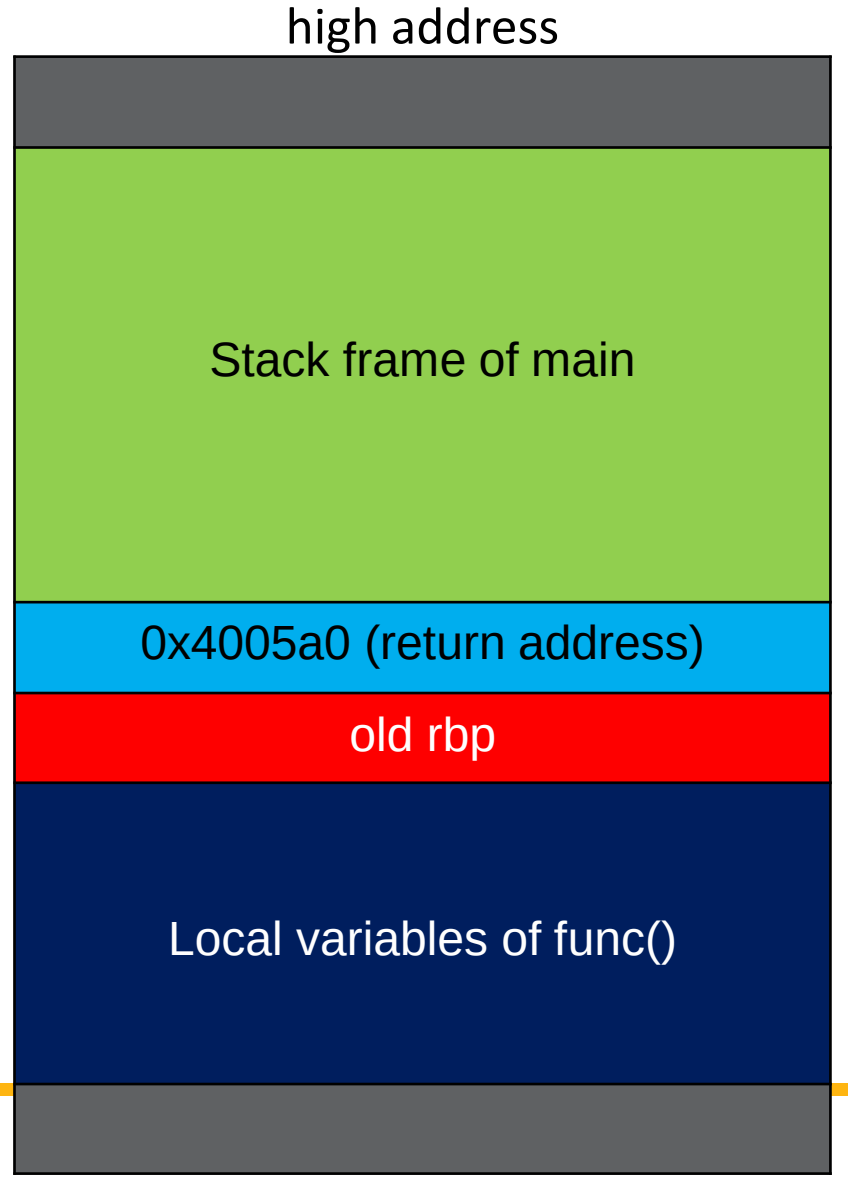
```
func:
        push rbp
        mov rbp, rsp
rip →   sub rsp, 0x30
        …
        move eax, 0x0
        leave
        ret
main:
        …
        call func
        mov eax, 0x0 // address 0x4005a0
        …
```

rbp → rsp →



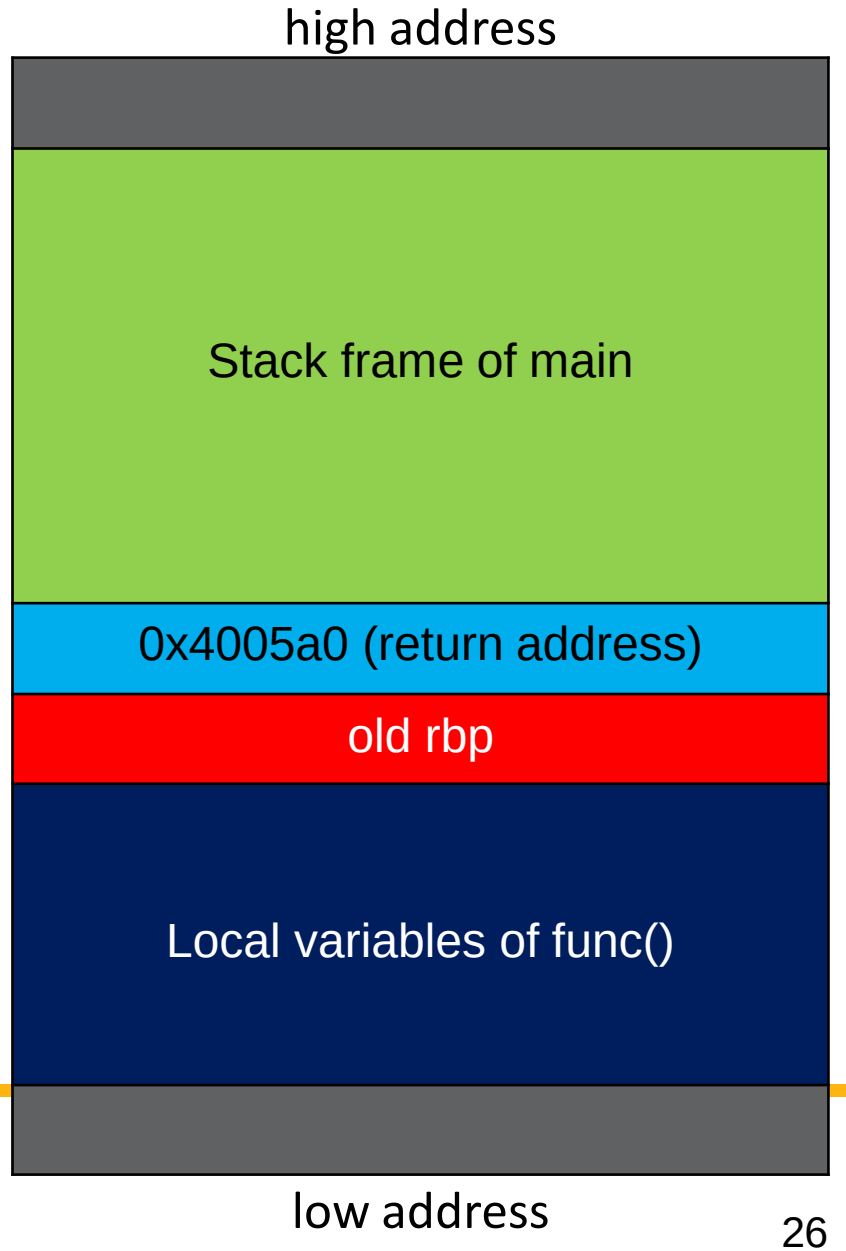
Example: Stack frame during a function call

```
func:
        push rbp
        mov rbp, rsp
        sub rsp, 0x30
rip →   ...
        move eax, 0x0
        leave
        ret
main:
        ...
        call func
        mov eax, 0x0 // address 0x4005a0
```



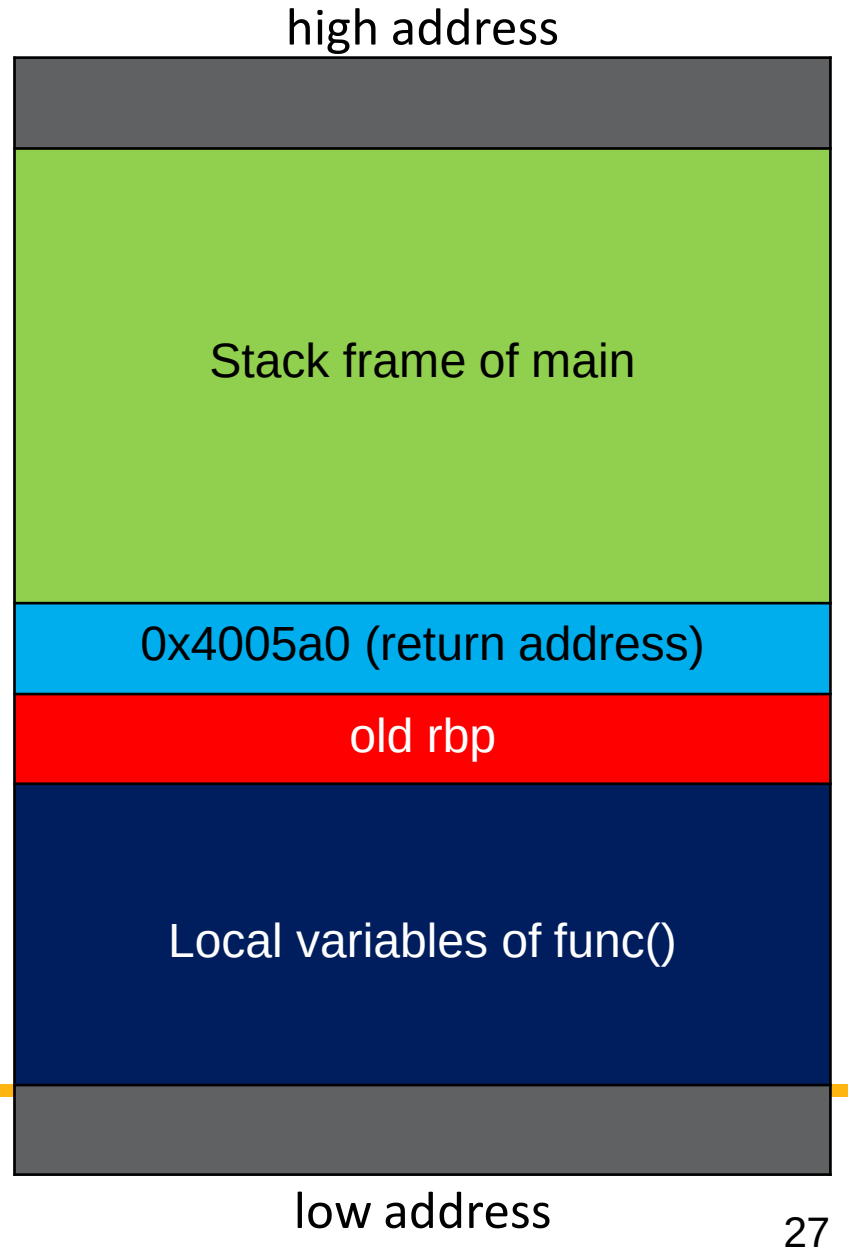
Example: Stack frame during a function call

```
func:
        push rbp                    leave = mov rsp, rbp
        mov rbp, rsp                        pop rbp
        sub rsp, 0x30
        …
        move eax, 0x0
rip →   leave
        ret
main:
        …
        call func
        mov eax, 0x0 // address 0x4005a0
```



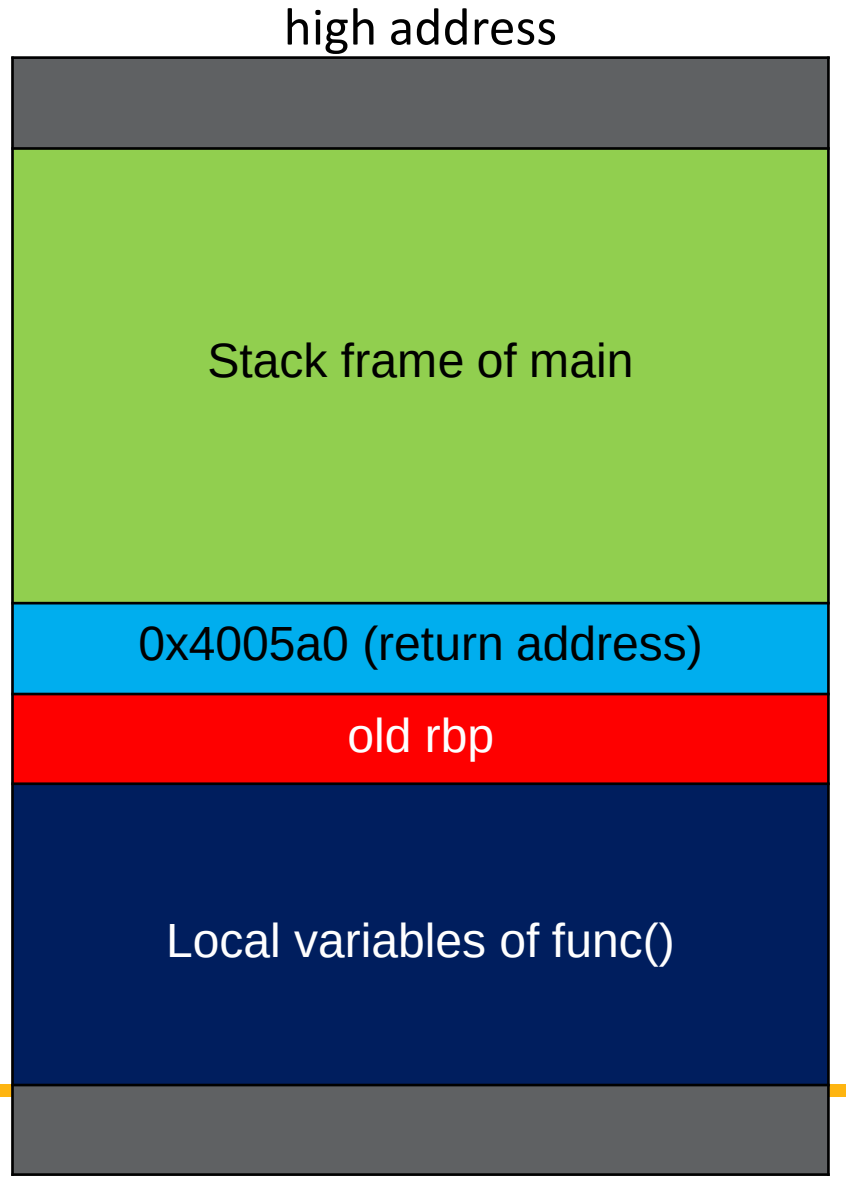
Example: Stack frame during a function call

```
func:
        push rbp                    leave = mov rsp, rbp
        mov rbp, rsp                pop rbp
        sub rsp, 0x30
        …
        move eax, 0x0
rip →   leave
        ret
main:
        …
        call func
        mov eax, 0x0 // address 0x4005a0
```



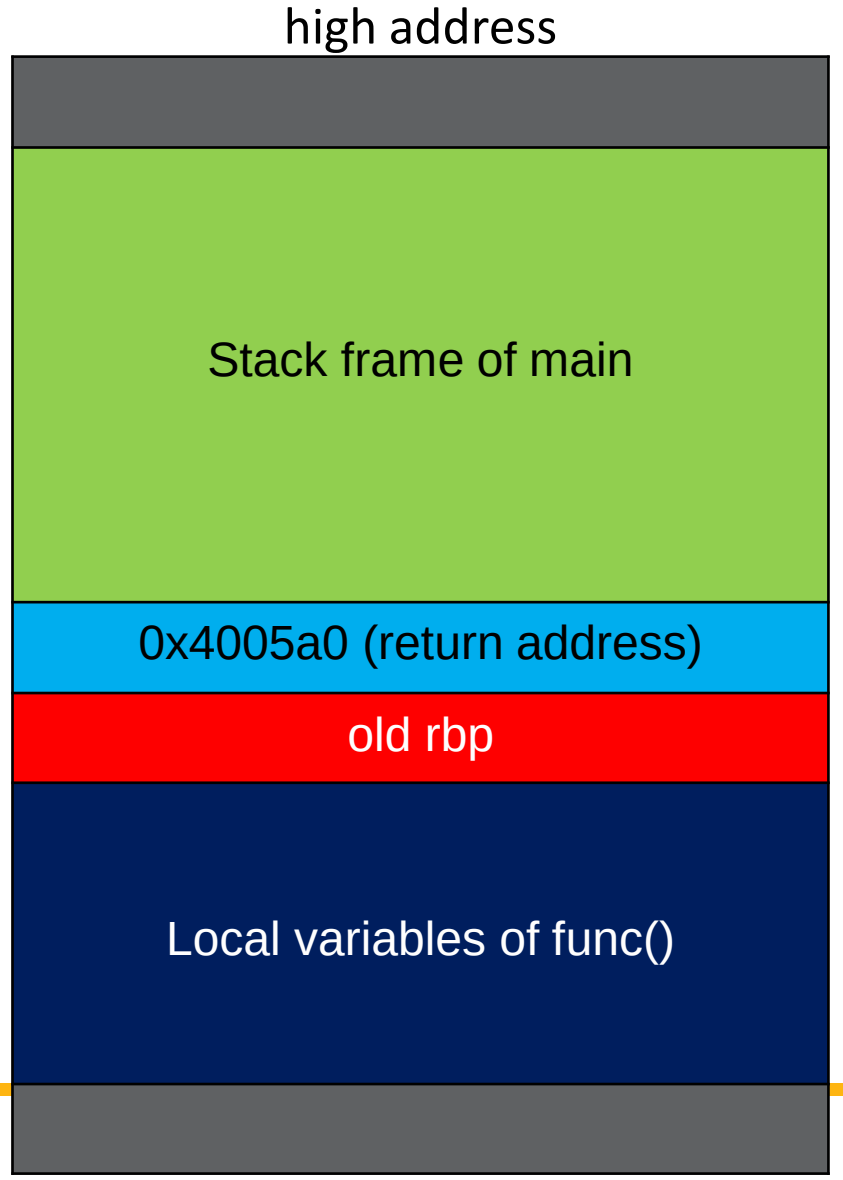
Example: Stack frame during a function call

```
                                                    rbp →
func:
        push rbp
        mov rbp, rsp
        sub rsp, 0x30
        …                          ret = pop rip
        move eax, 0x0
        leave                                       rsp →
rip →   ret
main:
        …
        call func
        mov eax, 0x0 // address 0x4005a0
        …
```



Example: Stack frame during a function call

```
                                                    rbp →
func:
        push rbp
        mov rbp, rsp
        sub rsp, 0x30
        …
        move eax, 0x0                               rsp →
        leave
        ret
main:
        …
        call func
rip →   mov eax, 0x0 // address 0x4005a0
        …
```



Project Submission

- Due date: 6/10 11:59 p.m.
- Makeup submission (75 points at most): TBA (After the final)
- Submission rules
  - Please create a folder for each task and put all the files/scripts needed to get the task's flag into the folder
  - Please put all the folders into a submission folder; zip the submission folder with your student ID and upload the zip file to New e3
  - Sample zip file: 309551234.zip
    - Fildes
      - exp.py
      - …
    - Timeout
      - exp.py
      - …
    - ….

Demo Procedure

- Date: 6/13
- Location: Online with Zoom
- We will run your programs/scripts with you online and see whether flags can be obtained
  - ☐ All the flags will be changed
- Demo schedule and links will be released later

Questions?