Computer Security Capstone

Project 4: Capture The Flag (CTF)

Chi-Yu Li (2023 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 crypto sport: 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 for all the tasks except Task I-2 and Task I-4
  - Solving Task I-2 and Task I-4 locally
- You are NOT allowed to team up: one student one team
  - Discussions are allowed between teams, but any collaboration is prohibited
- TA: Ping-Tsan Liu

How to Proceed?

- Connecting to each CTF server: nc <ip> <port>
  - ip: 140.113.207.243
  - 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
- You can use python with pwntools, too

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: FLAG{xxx}
    - xxx is the flag you need to submit

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 (80%)
- Task II: Advanced CTF problems (20%)
- Download all given executable and source files from e3

Task I: Basic CTF Problems

- Task I-1: math (20%)
- Task I-2: string (20%)
- Task I-3: random (20%)
- Task I-4: meow(20%)

Task I-1: math

- Goal: learn about the details of C language
- Server port: 8881

- Hints
  - cppreference
  - Take time to read the codes

Task I-2: string

- Goal: learn to use tools to inspect binary file
- Local file

- Hints
  - Inspect the binary, what does the encoded string looks like?
  - strings: print the strings of printable characters in files
  - base64

Task I-3: random

- Goal: learn about the glibc PRNG
- Server port: 8883

- Hints
  - Inspect the code, where does it leaks information?

Task I-4: meow

- Goal: learn about file format
- Local file

- Hints
  - Does that image have some additional bytes?
  - grep –obUaP "\x50\x4b\x03\x04"
    - Which file format starts with \x50\x4b\x03\x04 (0x04034b50) magic header?

Task II: Advanced CTF problems

- Task II-1: return2flag (10%)
- Task II-2: echooo (10%)

Task II-1: return2flag

- Goal: learn to identify logic flaw and buffer overflow in source codes
- Server port: 8885

- Hints
  - Inspect the code, where buffer overflow can occur?
  - There is a ***return address check***, how to bypass it?
  - Stack buffer overflow

Task II-2: echooo

- Goal: learn to identify dangerous function usage
- Server port: 8886

- Hints
  - How do you use printf normally?
    - Which conversion specifier can modify variable?

Important: How to Prepare Your Program?

- Must provide a Makefile which compiles your source codes into at least six executable file
- You can use any language and library you want
  - Use your environment to demo
  - Do not hardcode the flag in your program
- Test requirements for your program
  - Do not need user interaction to get flag
    - For online tasks, you can only input server IP and port
    - For local tasks, you can only input file path
  - Must print flag to stdout

Important: How to Demo Your Program?

- Share your screen
- Download your code from e3
- Download new file for task I-2 and task I-4
- Run make
- Run your executables (With new file and different server/port)
- Ask some questions about your code

- Binary file for task I-1, I-3, II-1, II-2 will not change
  - You can hardcode some symbol address if you need

Project Submission

- Due date: 6/7 23:59
- Submission rules
  - Put all your files into a directory and name it using your student ID(s)
  - Zip the directory and upload the zip file to New e3
  - A sample of the zip file: 1234567.zip
    - 1234567
      - Makefile
      - …
      - …
      - …

Questions?

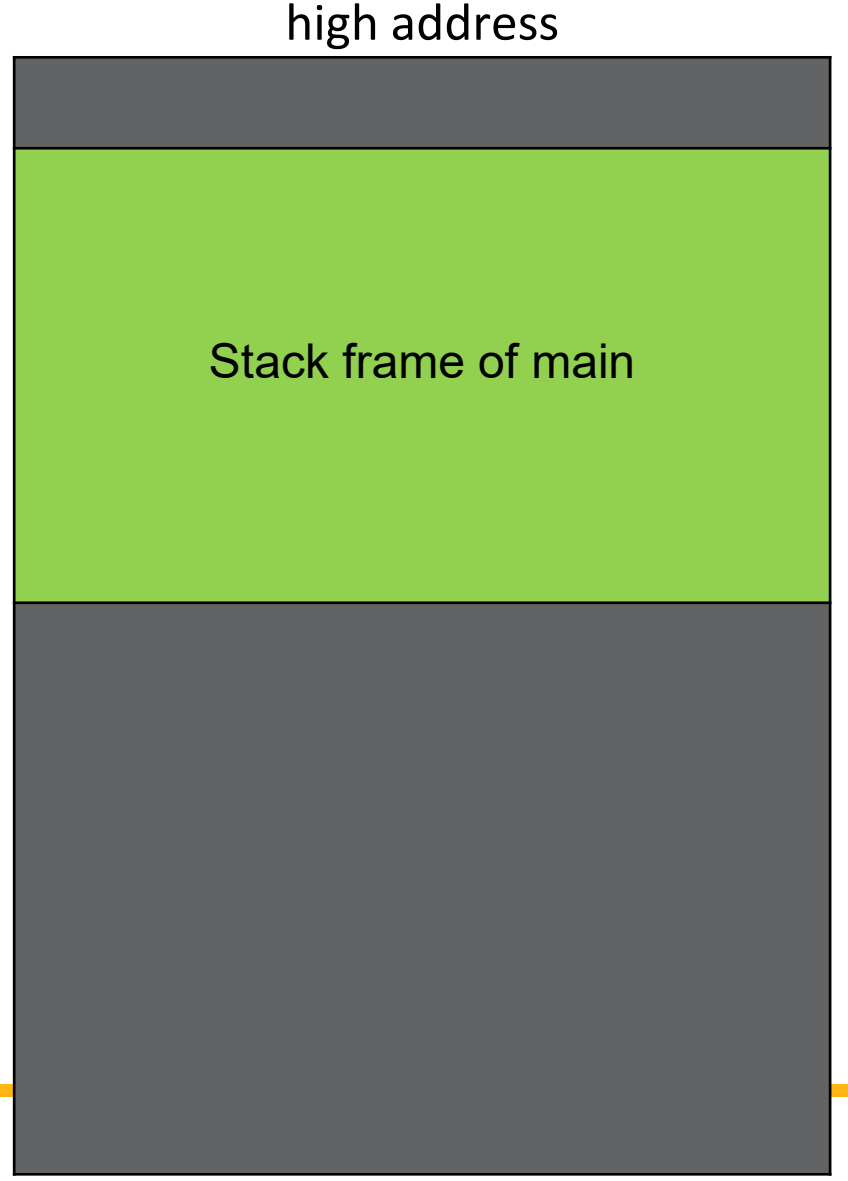
Useful Reference

- https://hackmd.io/@dange/rk9xmgHKX
- https://man7.org/
- https://github.com/longld/peda

Example: Stack frame during a function call

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

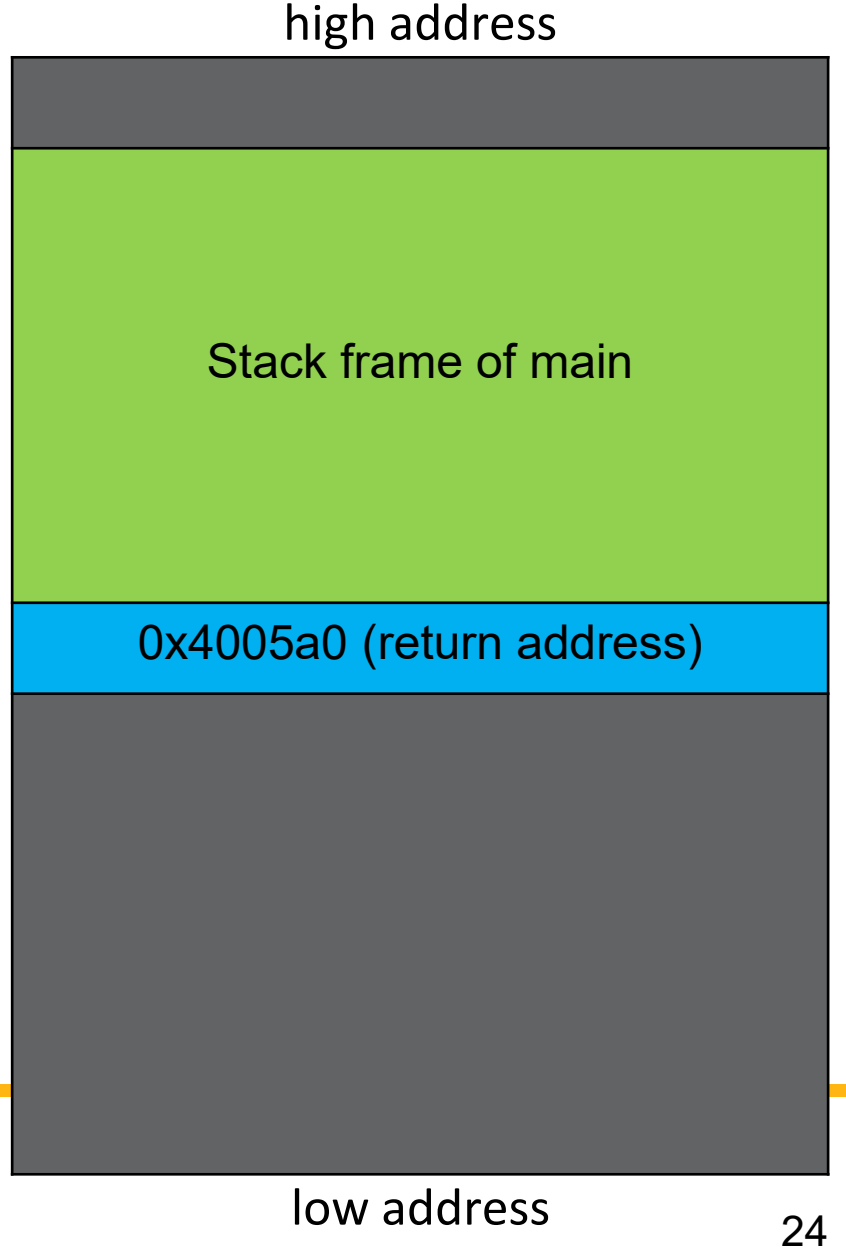
Call fun = push next_rip
jmp func



Example: Stack frame during a function call

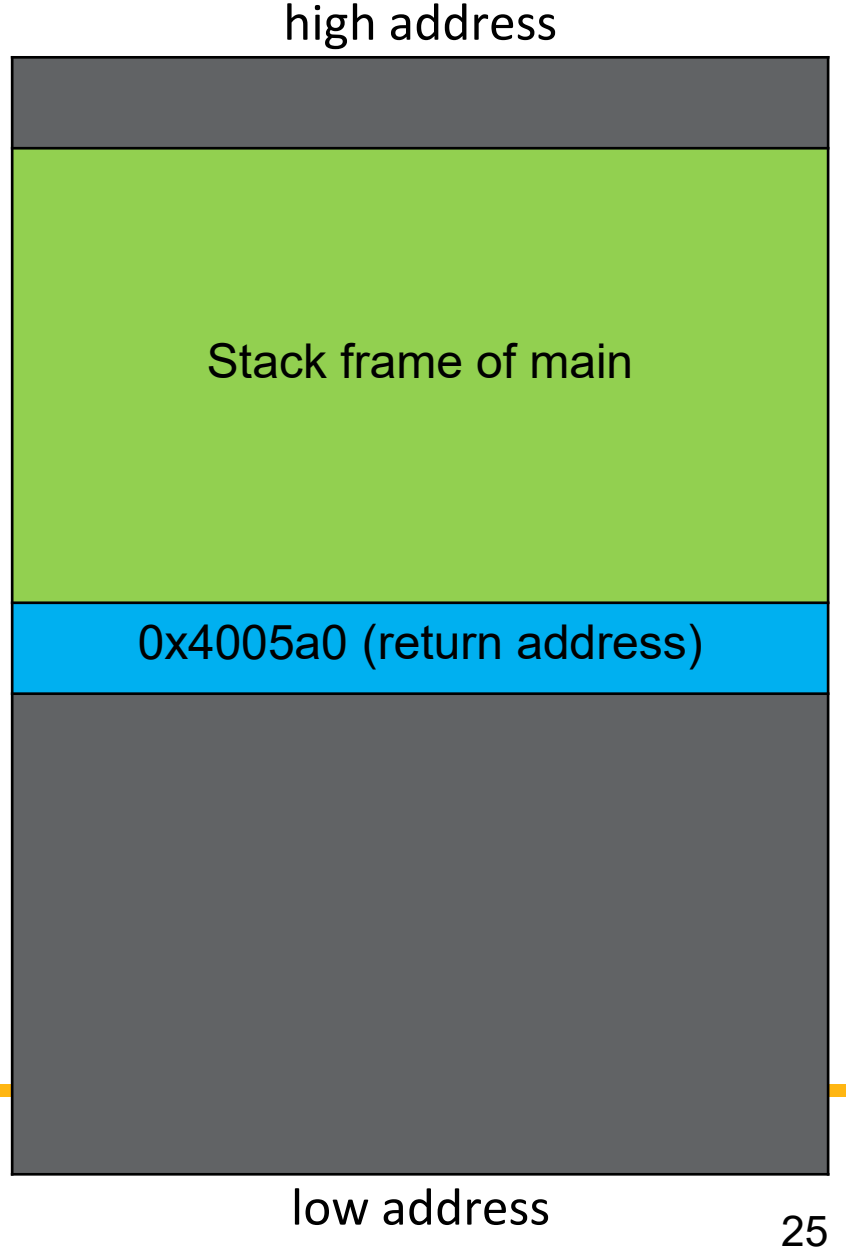
```
func:
        push rbp
        mov rbp, rsp
        sub rsp, 0x30
        …
        move eax, 0x0
        leave
        ret
main:
        …
rip ➔   call func
        mov eax, 0x0 // address 0x4005a0
        …
```

Call fun = push next_rip
jmp func



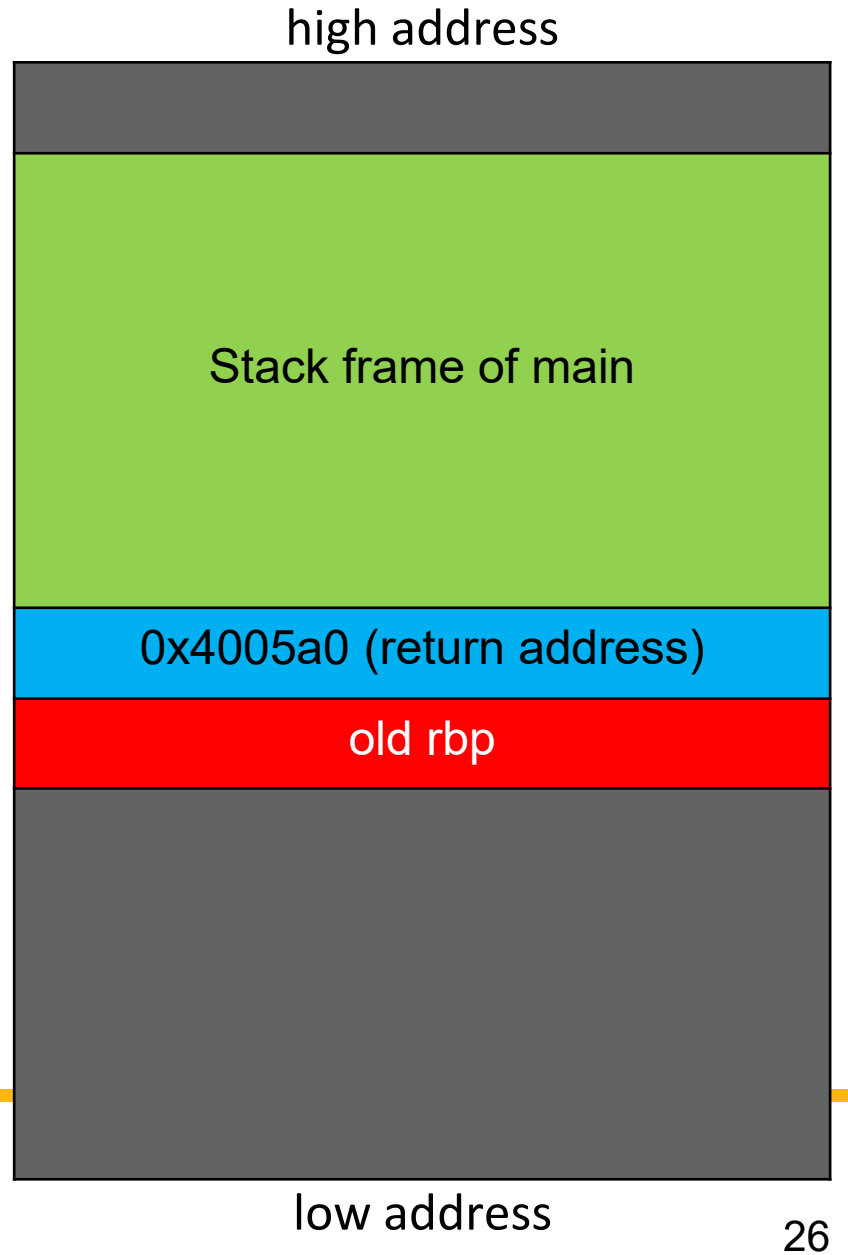
Example: Stack frame during a function call

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



Example: Stack frame during a function call

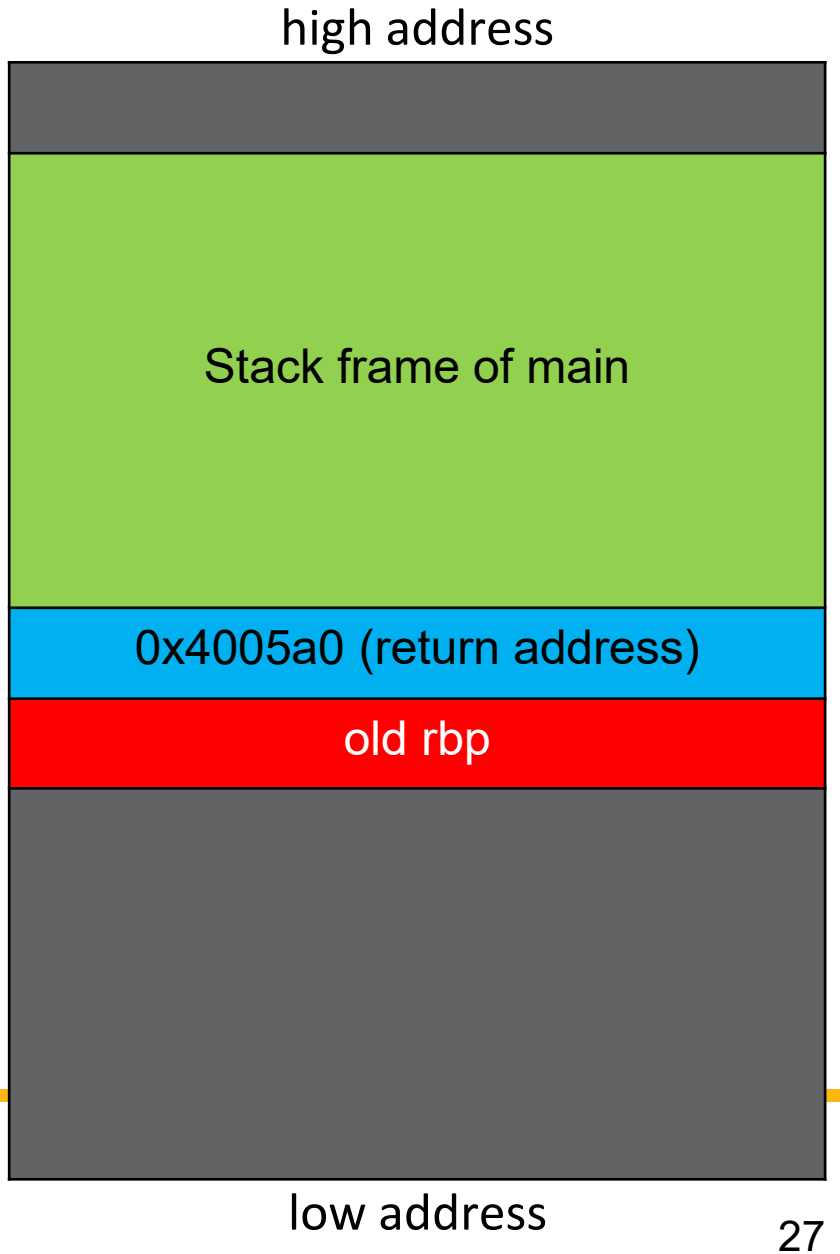
```
func:
        push rbp
rip ➔   mov rbp, rsp
        sub rsp, 0x30
        …
        move eax, 0x0
        leave
        ret
main:
        …
        call func
        mov eax, 0x0 // address 0x4005a0
        …
```



Example: Stack frame during a function call

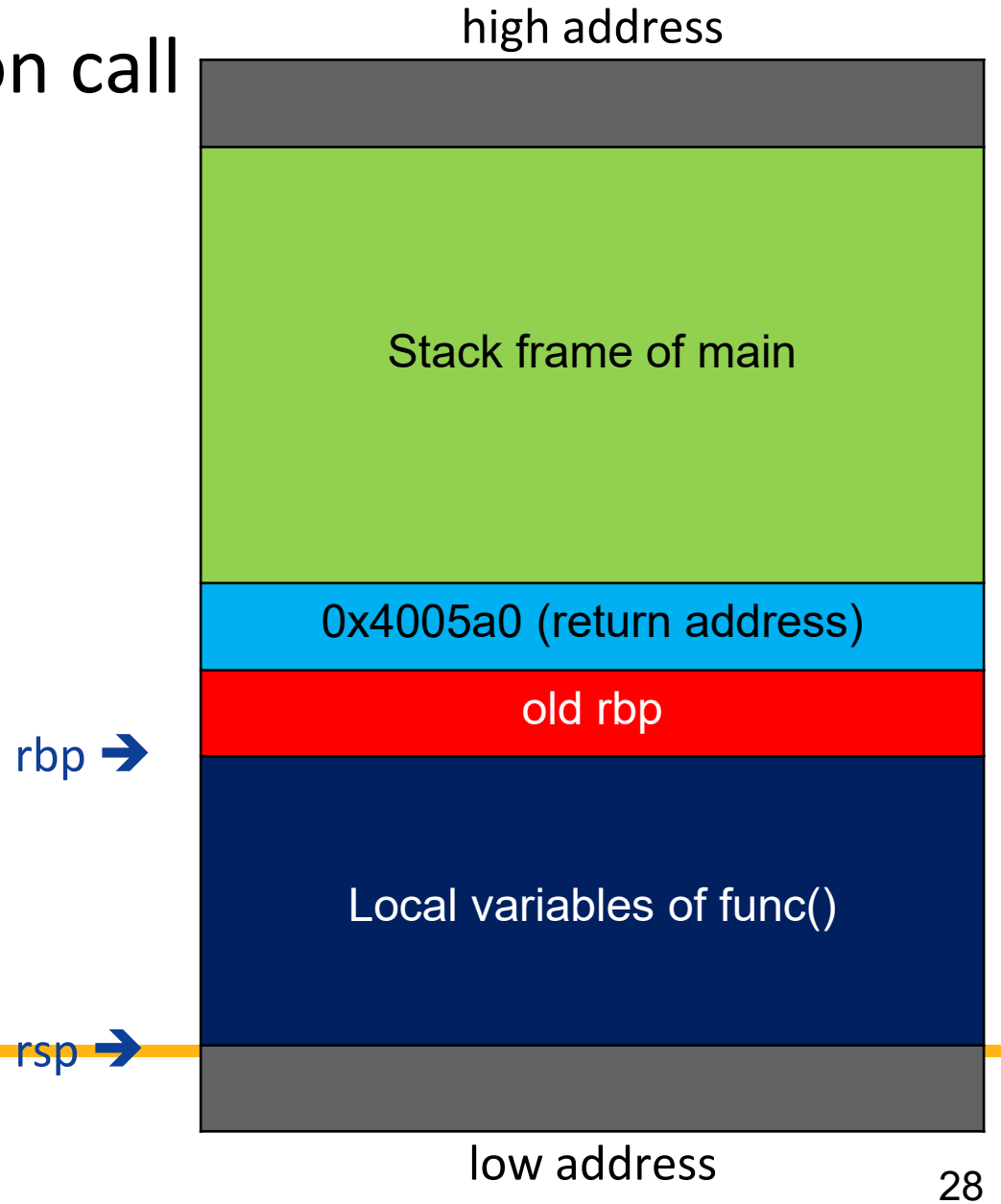
```
func:
        push rbp
        mov rbp, rsp
rip ➔   sub rsp, 0x30
        …
        move eax, 0x0
        leave
        ret
main:
        …
        call func
        mov eax, 0x0 // address 0x4005a0
        …
```

rbp ➔ rsp ➔



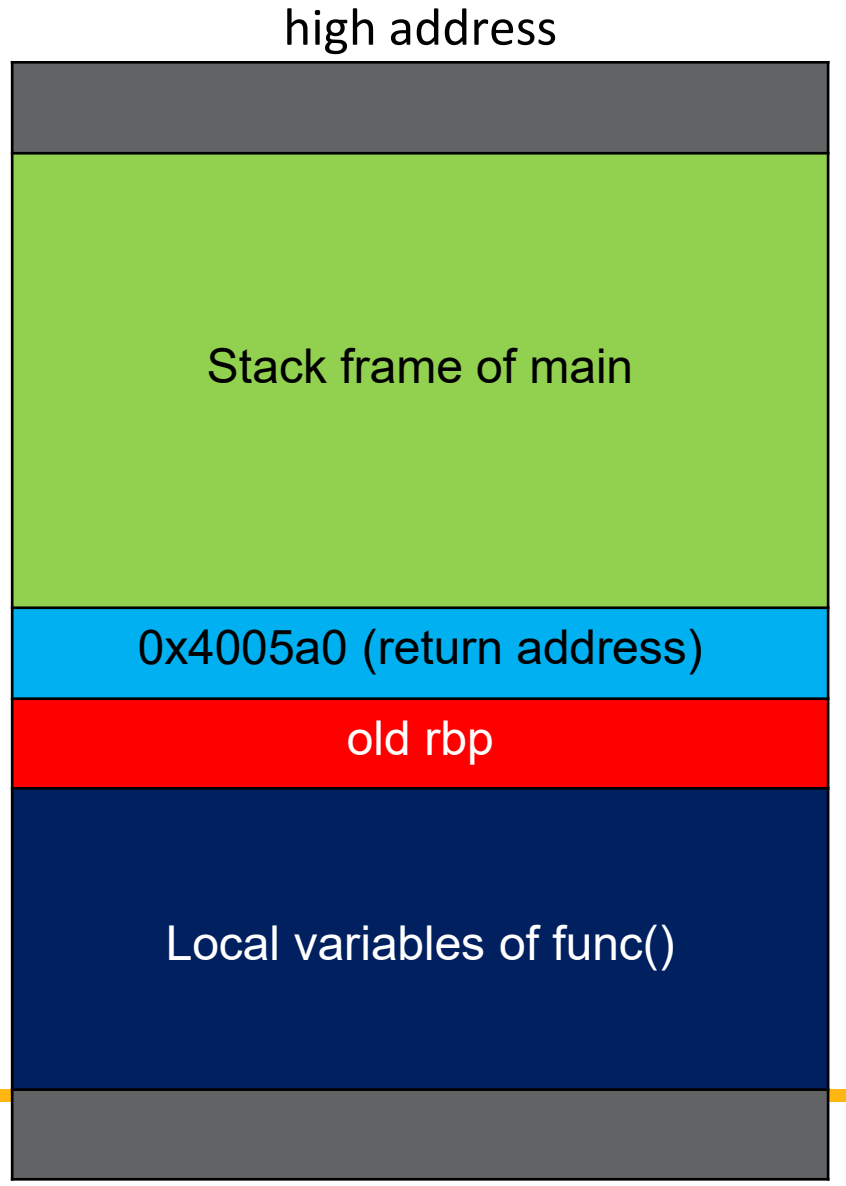
Example: Stack frame during a function call

```
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

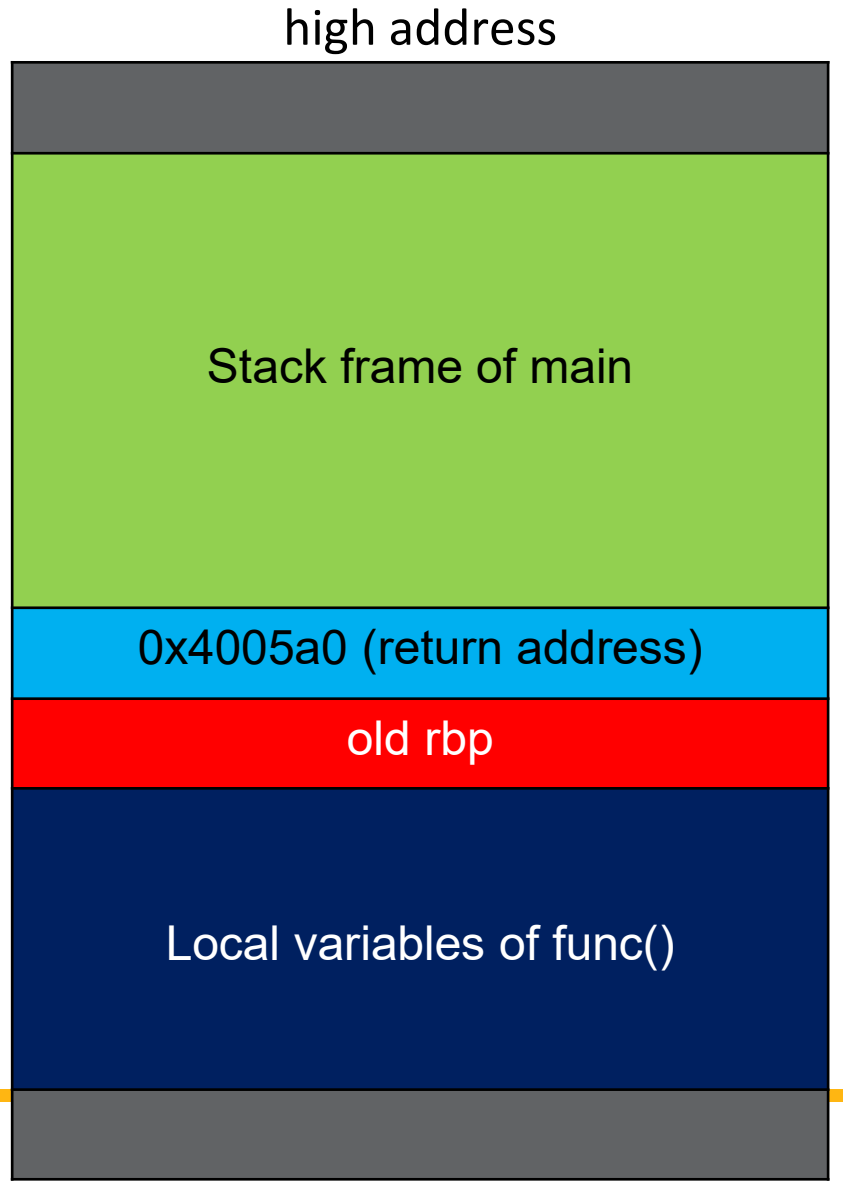
```
func:
        push rbp                    leave = mov rsp, rbp
        mov rbp, rsp                        pop rbp
        sub rsp, 0x30
        …
        move eax, 0x0
rip ➔   leave
        ret
main:
        …
        call func
        mov eax, 0x0 // address 0x4005a0
        …
```



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
        …
```

rbp ➔ rsp ➔

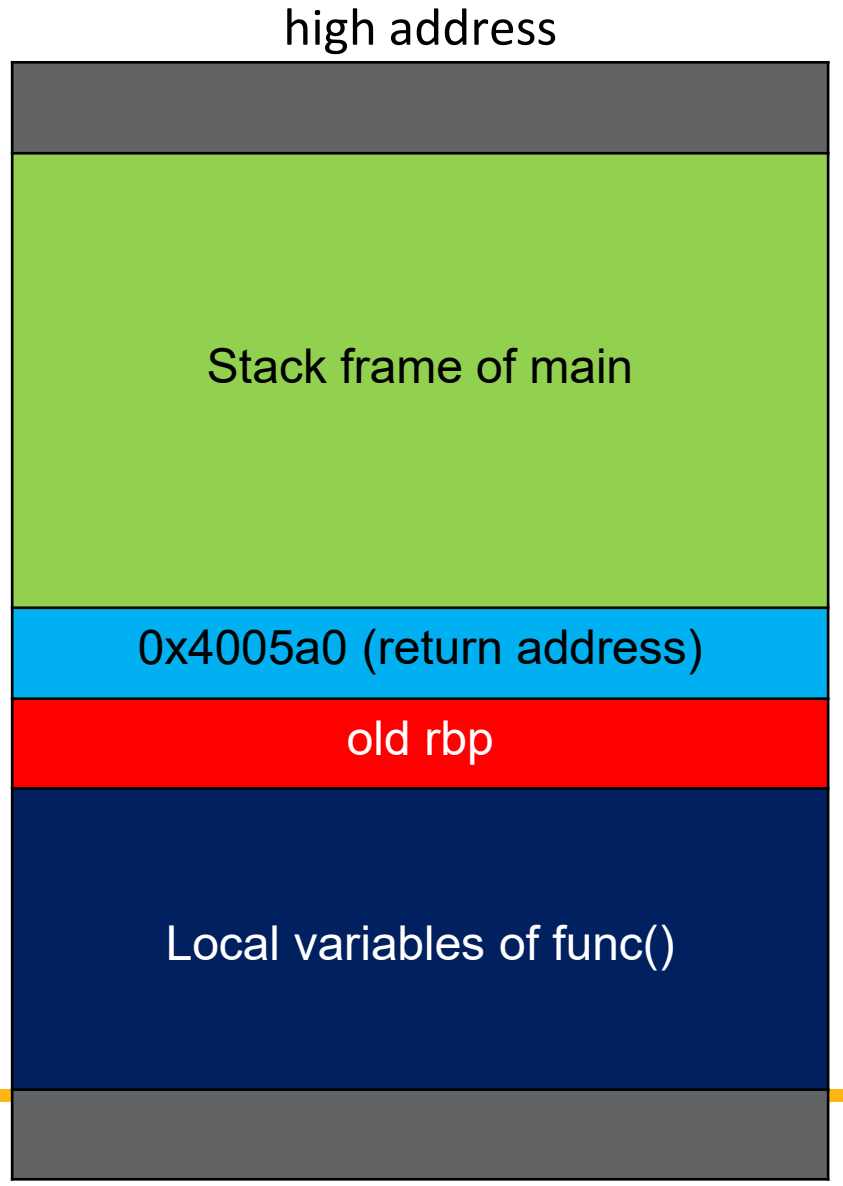


Example: Stack frame during a function call

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

rbp ➔

rsp ➔