Introduction to Computer Security

Project 4: Capture The Flag (CTF)

Chi-Yu Li (2020 Spring)
Computer Science Department
National 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 II-2
  - Solving Task II-2 locally

- You are NOT allowed to team up: one student one team
  - Discussions are allowed between teams, but any collaboration is prohibited

- TA: Po-Yi Chou

How to Proceed?

- Choosing your CTF servers based on your student ID

- Each CTF problem has 4 servers for 4 groups
  - Group ID = student ID % 4

- Connecting to each CTF server: nc <ip> <port>
  - ip: 140.113.207.233
  - 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: 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 (60%)
- Task II: CTF beginners (40%)
- Download all given executable and source files from the following link
  - http://140.113.207.233:9820

Task I: Basic CTF Problems

- Task I-1: Fildes (20%)
- Task I-2: Rand-breaker (20%)
- Task I-3: Nasty-rules(20%)

Task I-1: Fildes

- Goal: learn about Linux fd & standard I/O streams
- Server port: 881x
  - x: Group ID (0, 1, 2, 3)

- Hints
  - $ man stdin
  - $ man 2 read
  - $ man 2 atoi
  - Take time to read the codes

Task I-2: Rand-breaker

- Goal: learn to read a manual carefully
- Server port: 882x
  - x: Group ID (0, 1, 2, 3)

- Hints
  - Do you really know how a C function works?
    - Don't lie to yourself. If you don't know, ask "man" as usual
    - $ man 3 rand
  - Read the manual carefully
    - Manuals/documents/source codes are important

Task I-3: Nasty-rules

- Goal: learn about the details of C language
- Server port: 883x
  - x: Group ID (0, 1, 2, 3)

- Hints
  - Operator precedence

Task II: CTF Beginner

- Task II-1: Time-will-stop (20%)
- Task II-2: Agent-hacker (10%)
- Task II-3: Ret-shellcode (10%)

Task II-1: Time-will-stop

- Goal: learn to use tools to inspect binary file
- Binary executable file: time_will_stop_group[ID]
  - x: Group ID (0, 1, 2, 3)

- Recommended tool
  - objdump: display information in object files
  - strings: print the strings of printable characters in files
  - GDB - PEDA:
    - Python Exploit Development Assistance for GDB
    - https://github.com/longld/peda

Task II-2: Agent-hacker

- Goal: learn to identify basic logic flaw and buffer overflow in source codes
- Server port: 885x
  - x: Group ID (0, 1, 2, 3)
- Recommended tool
  - pwntools (pip install pwntools): a useful python module for pwn
  - GDB - PEDA:
    - Python Exploit Development Assistance for GDB
    - https://github.com/longld/peda

Task II-3: Ret-shellcode

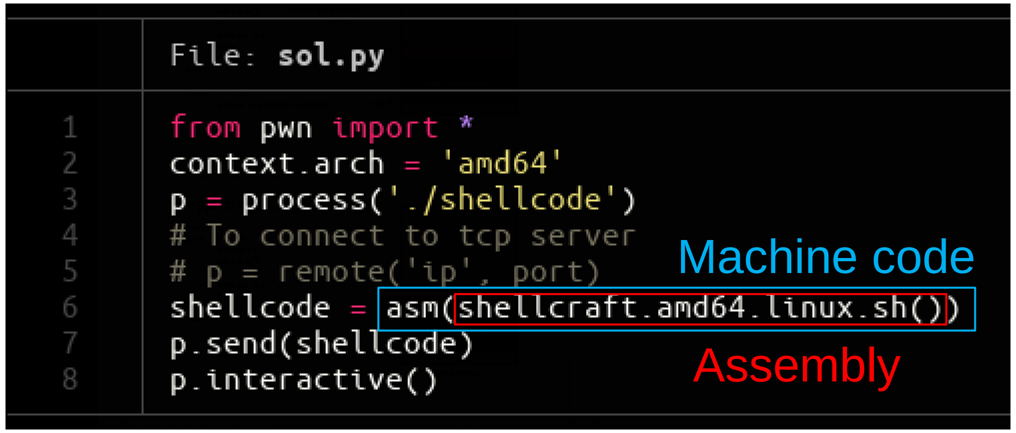
- Goal: learn to run shellcode with buffer overflow
- Server port: 886x
  - x: Group ID (0, 1, 2, 3)

Task II-3: Ret-shellcode (Cont.)

- Recommended tool
  - 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
  - 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: Ret-shellcode (Cont.)

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



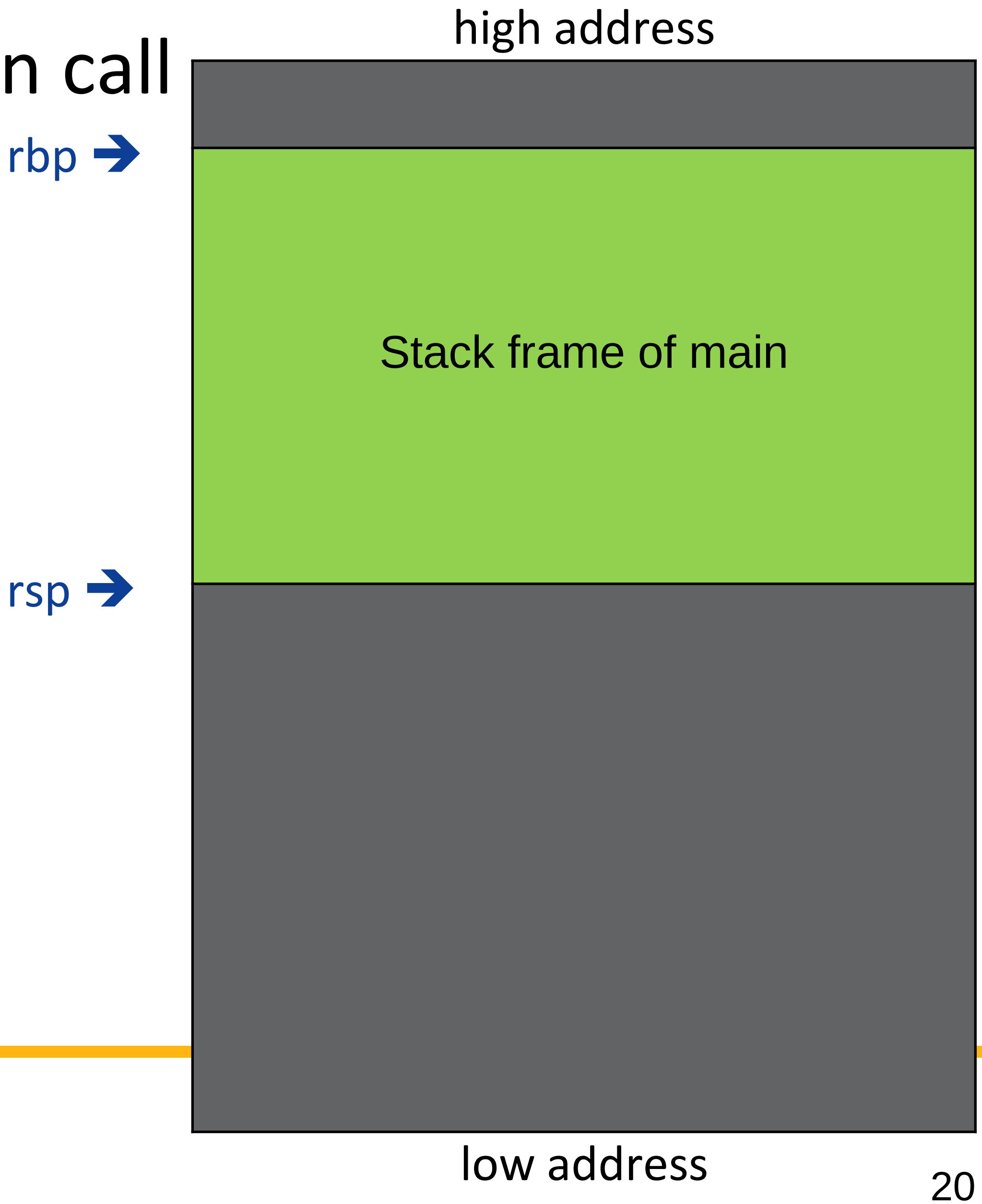
```
File: sol.py

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

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
        …
```

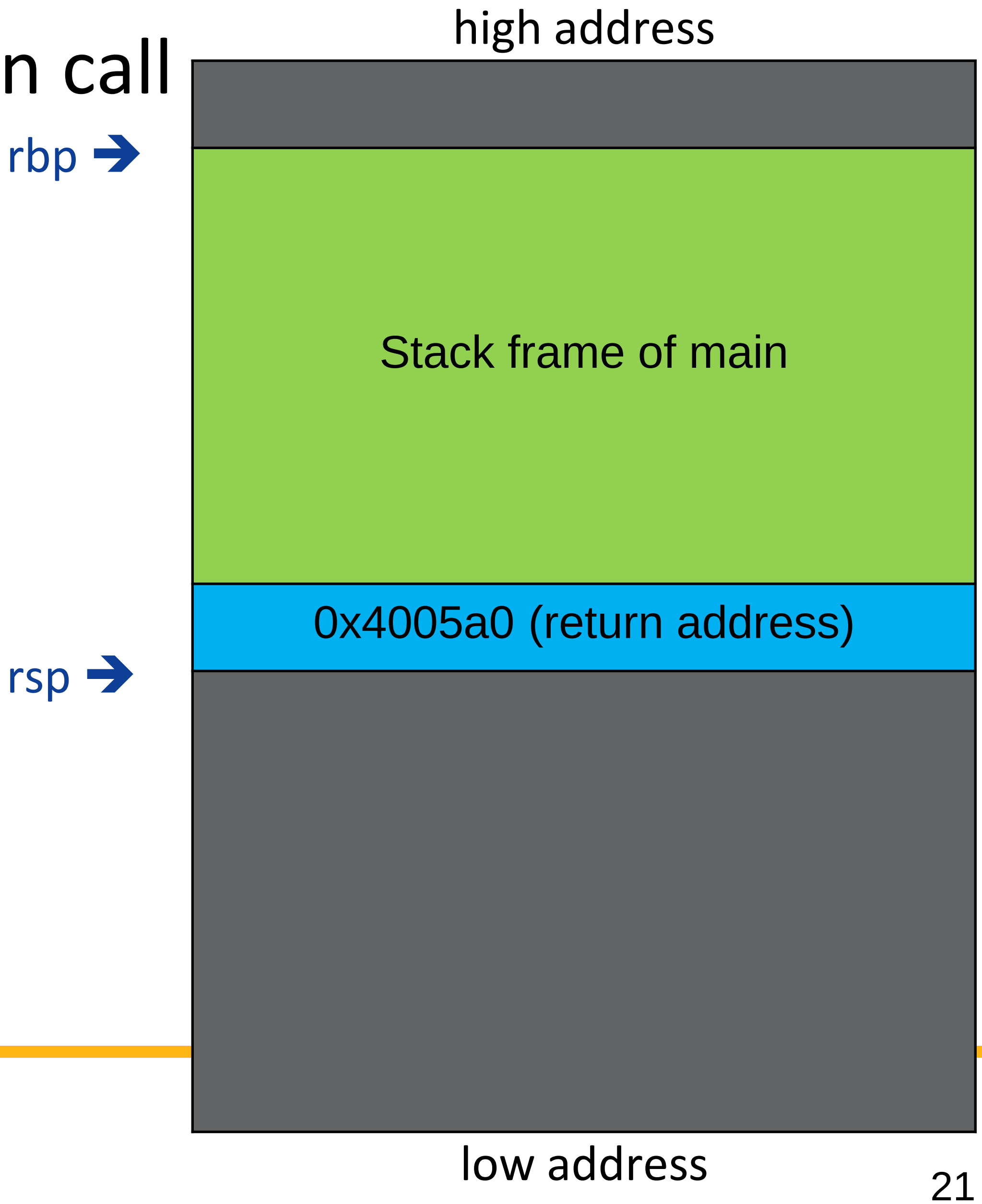
Call fun = push next_rip
           jmp func



Example: Stack frame during a function call

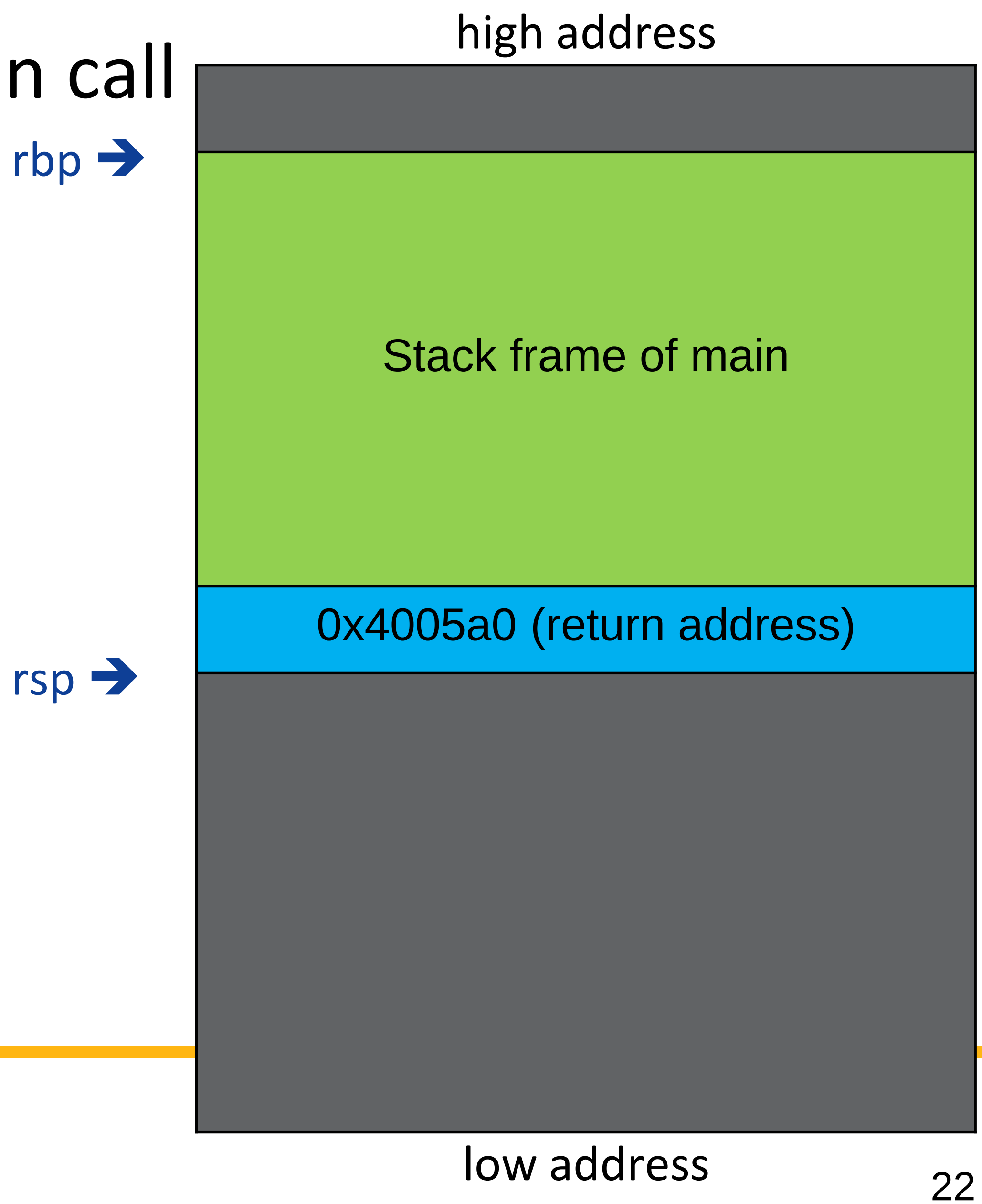
```
func:
        push rbp
        mov rbp, rsp
        sub rsp, 0x30
        …
        move eax, 0x0
        leave
        ret
main:
        …
rip →   call func
        mov eax, 0x0 // address 0x4005a0
        …
```

Call fun = push next_rip
jmp func



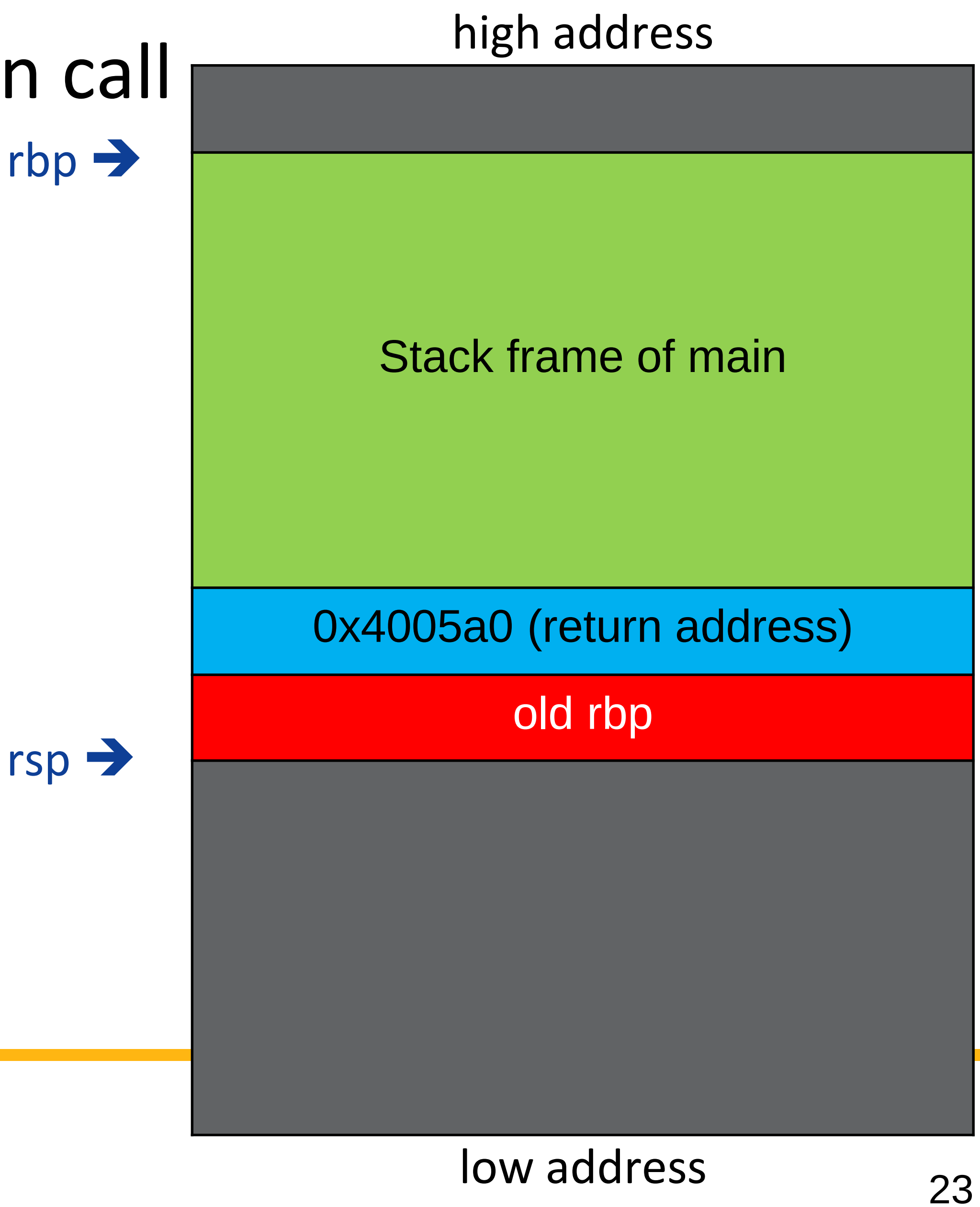
Example: Stack frame during a function call

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



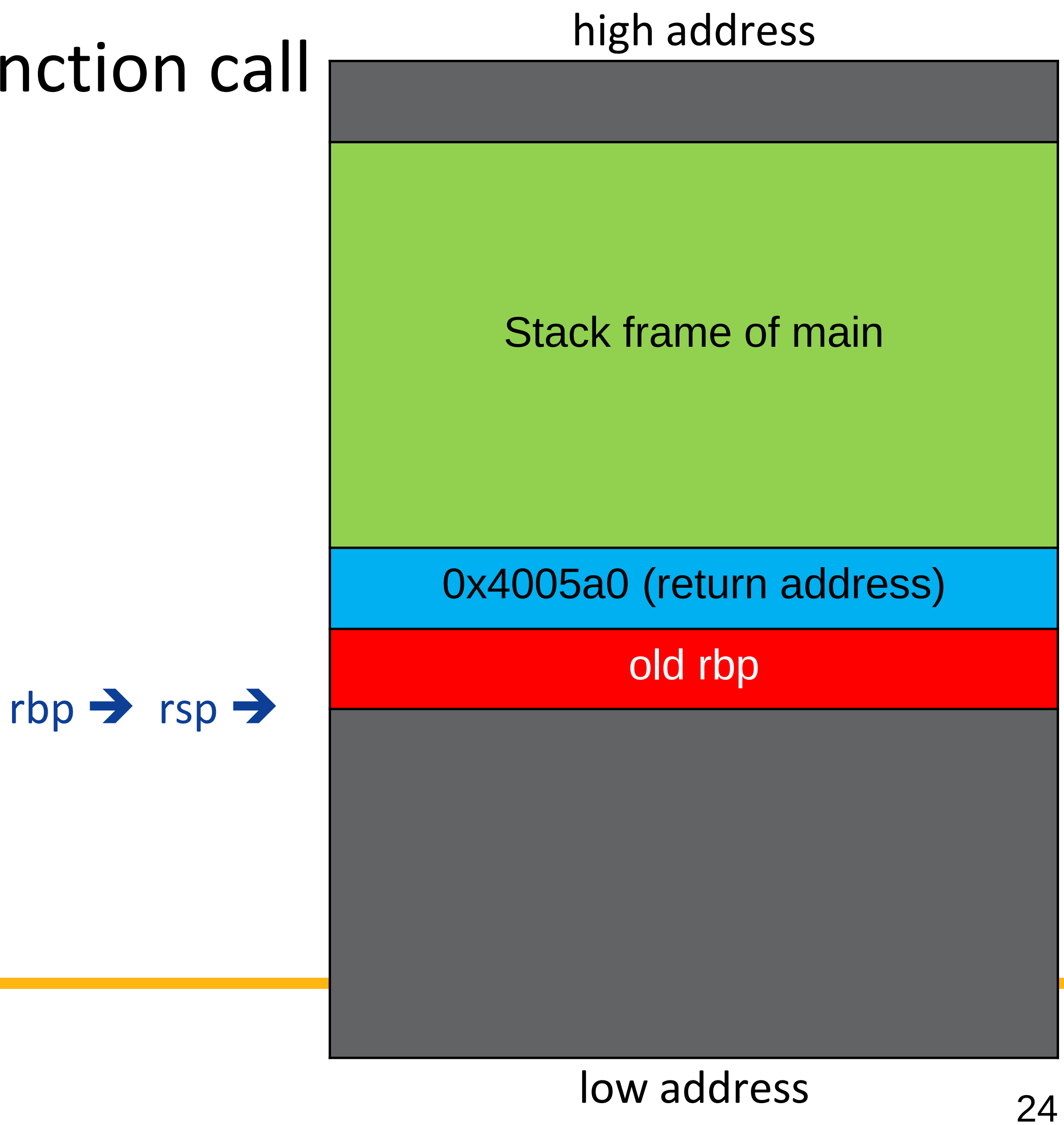
Example: Stack frame during a function call

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



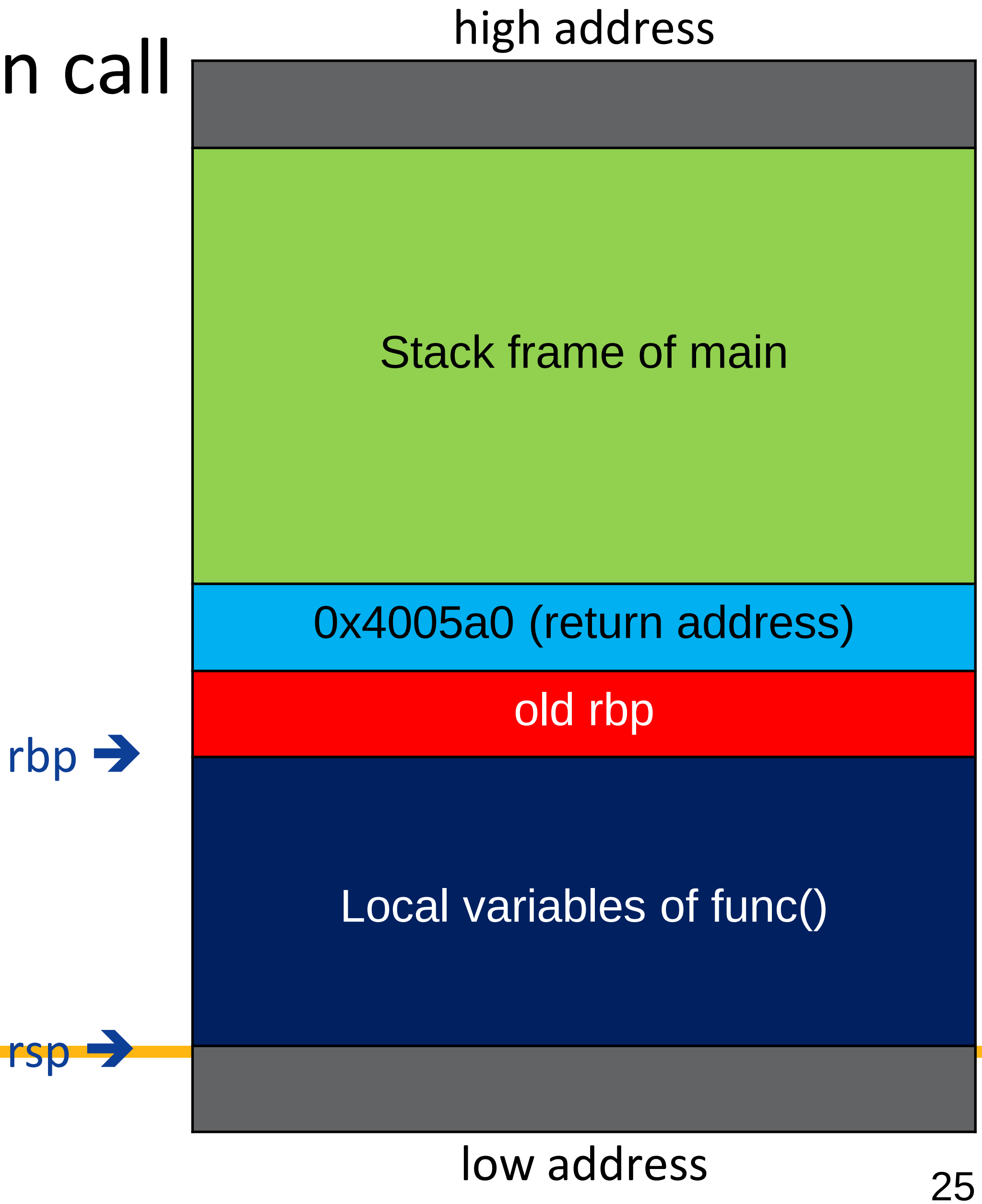
Example: Stack frame during a function call

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



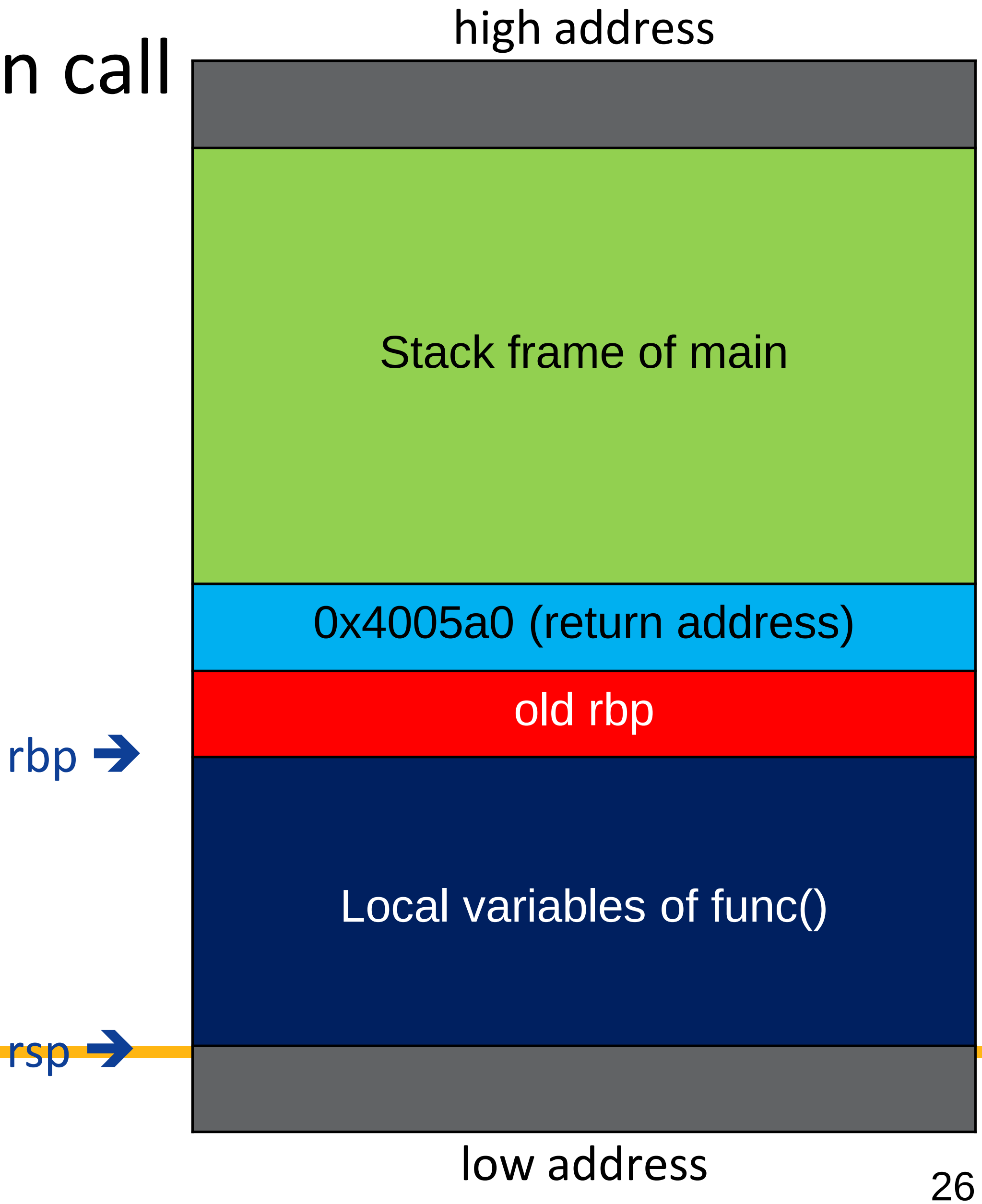
Example: Stack frame during a function call

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



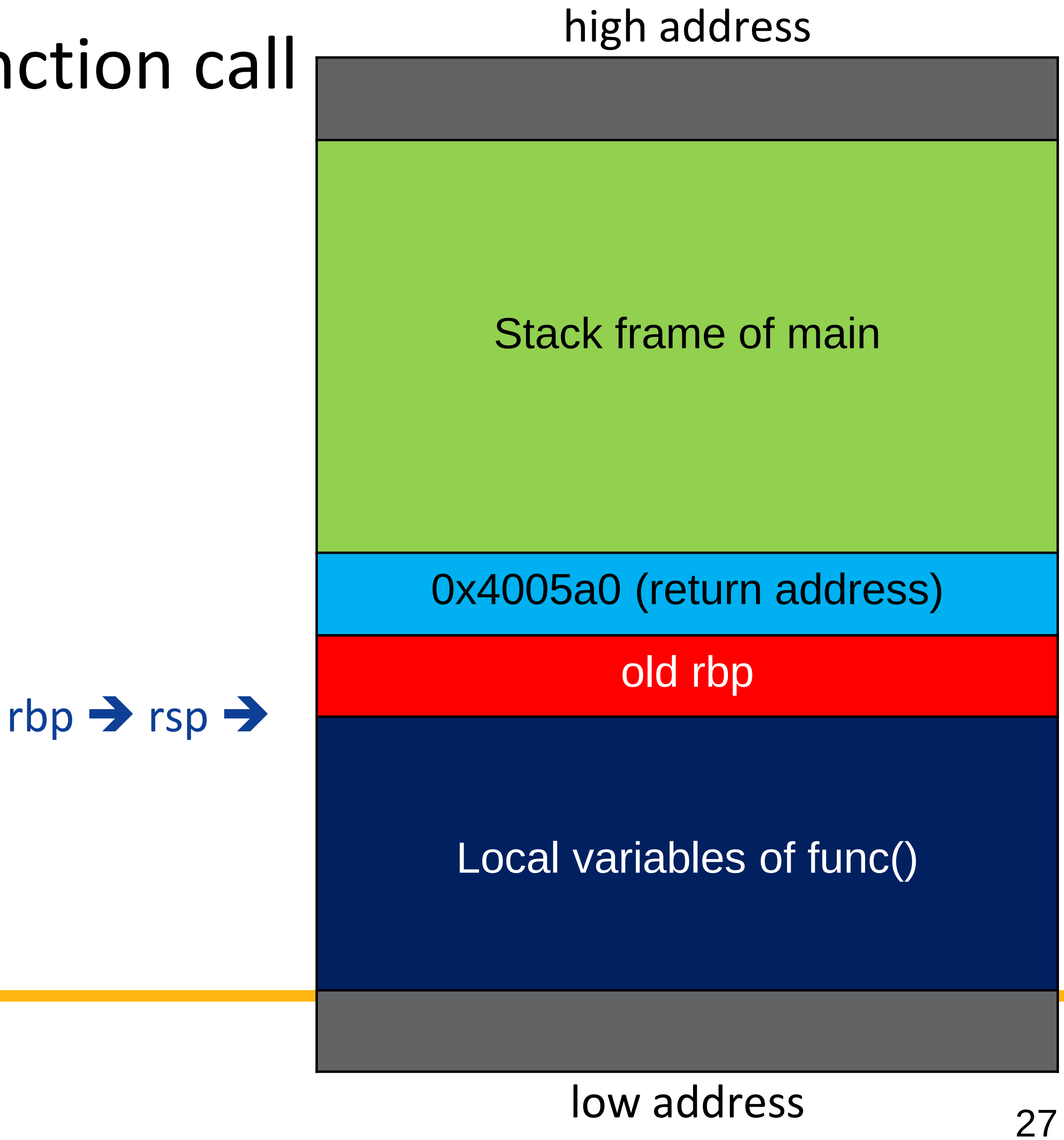
Example: Stack frame during a function call

```
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

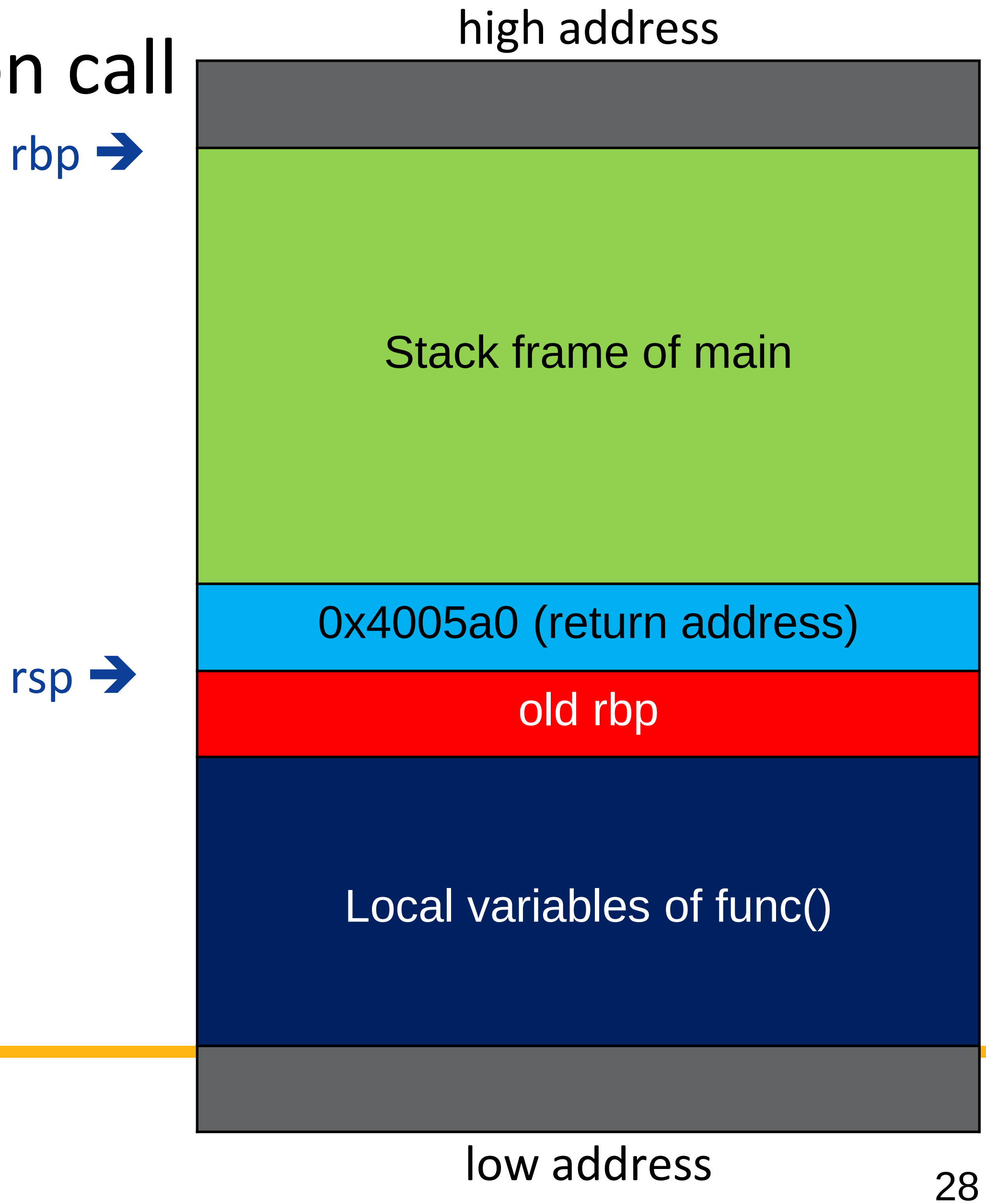
```
func:
        push rbp                leave = mov rsp, rbp
        mov rbp, rsp                    pop rbp
        sub rsp, 0x30
        …
        move eax, 0x0
rip ➔   leave
        ret
main:
        …
        call func
        mov eax, 0x0 // address 0x4005a0
        …
```



Example: Stack frame during a function call

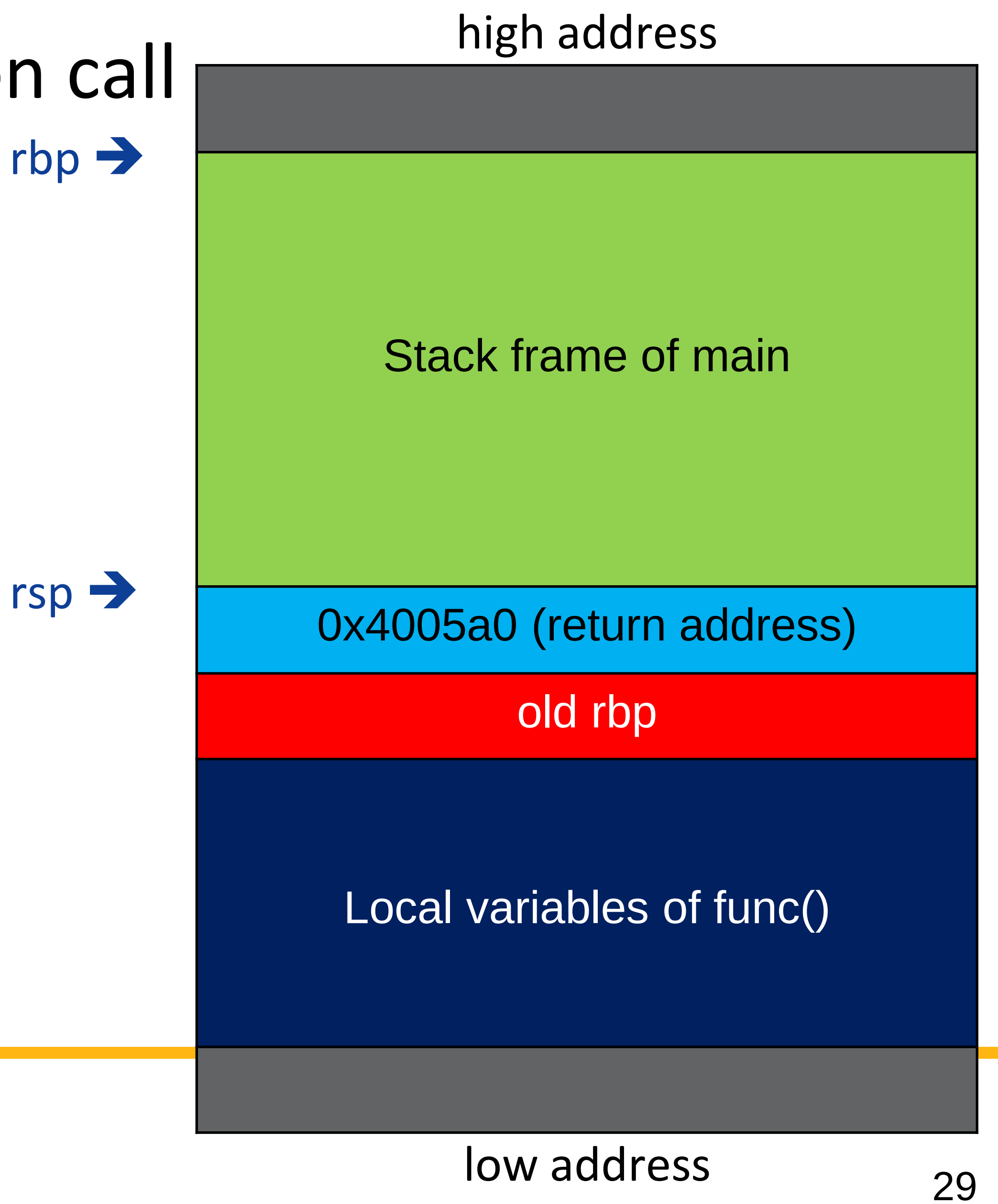
```
func:
        push rbp
        mov rbp, rsp
        sub rsp, 0x30
        …
        move eax, 0x0
        leave
rip →   ret
main:
        …
        call func
        mov eax, 0x0 // address 0x4005a0
        …
```

ret = pop rip



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
        …
```



Project Submission

- Due date: 6/24 11:55 p.m.
- Submission rules
  - Please put your flags in a text file
    - First line: your ID number
    - Next lines: “problem_name|flag”
    - For example
      - 123456789
      - fildes|FLAG_1
      - Rand-breaker|FLAG_2
  - Submit this text file to new E3
    - Filename: ONLY your student ID without “.txt”

Project Submission (Cont.)

- We will grade the text file by a script
  - Any submission that fails the script will get **NO POINTS**
  - Remember that no extension in the filename

- The grading script and an example of your submission file are on GitHub
  - https://github.com/poyichou/nctuics-p3-grade-script

- Make sure you have tested your file by the grading script **Before Submission**

Questions?