Computer Security Capstone

Project 4: Capture The Flag (CTF)

Chi-Yu Li (2024 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 for all the tasks
- You are NOT allowed to team up: one student one team
  - Discussions are allowed between teams, but any collaboration is prohibited
- TA: Cheng-I Hu

How to Proceed?

- Connecting to each CTF server: nc <ip> <port>
  - ip: 140.113.207.245
  - 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: CSC2025{[a-zA-Z0-9_]+}

- You will need to submit the programs
  - run the programs when you demo

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: Advance CTF problems (30%)
- Download all given executable and source files from e3
  - CTF Server using ubuntu 24.04 (for some problem to calculate address)

Task I: Basic CTF Problems

- Task I-1: Password Checker (20%)
- Task I-2: Secure Random (20%)
- Task I-3: Simple Shell (15%)
- Task I-4: Simple ROP (15%)

Task I-1: Password Checker

- Goal: Learn how type conversion works in C/C++
- Server port: 30170

- Hints
  - Implicit conversions of type

Task I-2: Secure Random

- Goal: learn about the glibc PRNG
- Server port: 30171

- Hints
  - Is the random function really random?
    - Make sure you have time synchronization in your environment!!

Task I-3: Simple Shell

- Goal: learn to identify basic logic flaw and buffer overflow in source codes
- Server port: 30172

- Hints
  - Inspect the code, where buffer overflow can occur?
  - What can you modify?
  - Inspect the impact of overflow by using gdb
    - You may want to install gdb extensions like gef

Task I-4: Simple ROP

- Goal: Given buffer overflow, try to find a way to open up a shell for remote command execution!!
- Server port: 30173

- Hints
  - Inspect the code, where buffer overflow can occur?
  - With NX enabled, you cannot write shell code for buffer overflow
  - Stack buffer overflow
  - Return-oriented programming
  - You may want to use tools like ROPgadget to find gadget for ROP

Task II: Advance CTF Problems

- Task II-1: ret2Flag (10%)
- Task II-2: Simple RTOS (10%)
- Task II-3: Hard ROP(10%)

Task II-1: Ret2Flag

- Goal: Learn exploit buffer overflow to control program flow
- Server port: 30174

- Hints
  - Inspect the code, where buffer overflow can occur?
  - How can you bypass canary protection?
  - How can you find the function address?
    - You may want to find address that related with putFlag
  - Try to leak the information you need!!!

Task II-2: Simple RTOS

- Goal: learn to identify dangerous function usage
- Server port: 30175

- Hints
  - How do you use printf normally?
    - Which conversion specifier can modify variable?
  - How can you return to the function you want?

Task II-3: Hard ROP

- Goal: Try to ROP with libc gadget!!
- Server port: 30176

- Hints
  - First, try to leaking every thing you need
  - Try to use libc gadget

Important: How to Prepare Your Program?

- Must provide a Makefile which compiles your source codes into senven 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?

- Download your code from e3
- Run make if needed
- Run your executables
- Ask some questions about your code
- Binary file for all task will not change
  - You can hardcode some symbol address if you need
  - FLAG during demo will change to avoid hardcode the flag

Project Submission

- Due date: 5/28 11:55 p.m.
- 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 (if needed)
    - ├ …
    - ├ …
    - └ …
    - (Please have a studentID folder in your zip)
  - If files are not in a directory after unzip, 10 points will be deducted.

Questions?

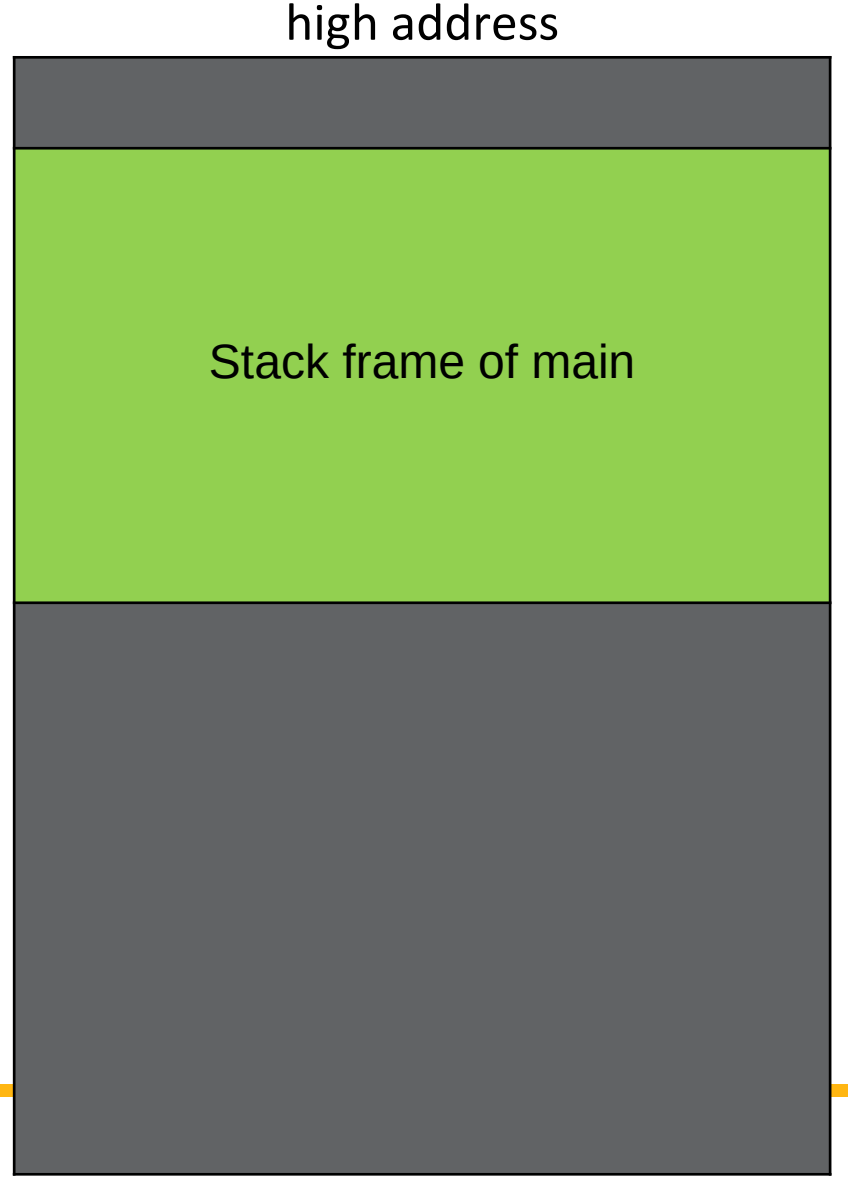
Useful Info

- command
  - checksec
  - readelf
- pwntools
  - connect to server and control what content will be sent to it
  - generate shellcode, attach gdb …etc.
- gdb
  - normal plugins: pwngdb / gef /peda
  - dynamic analysis of the program

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
        …
```

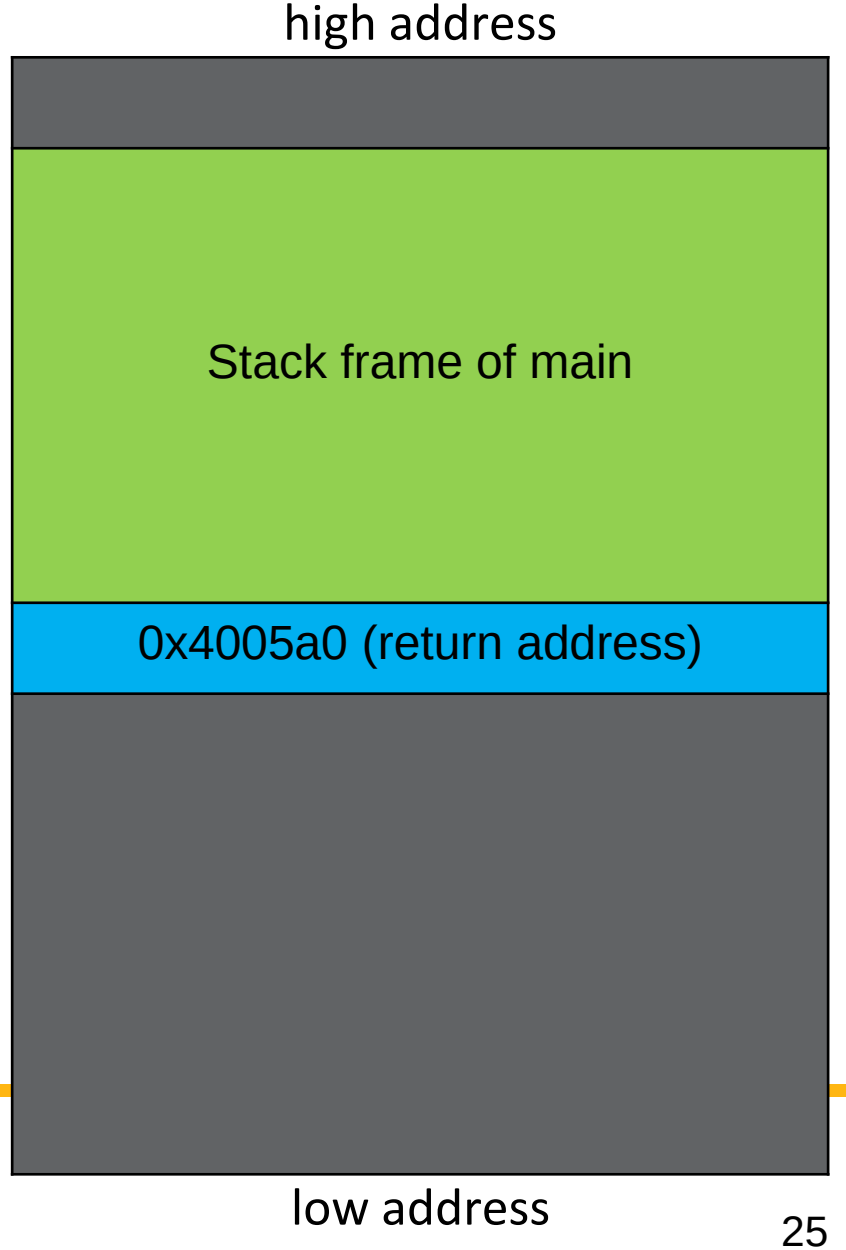
Call fun = push next_rip
jmp func



Example: Stack frame during a function call

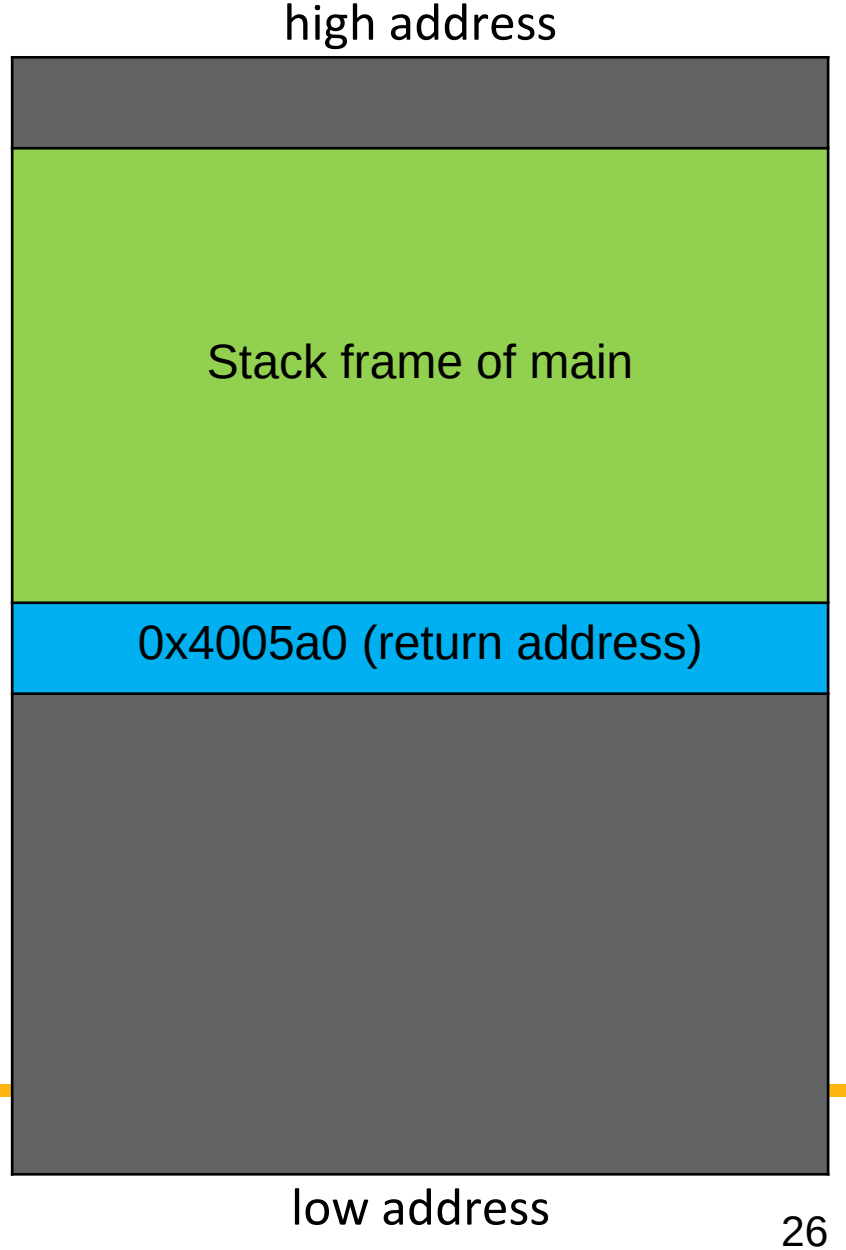
```
func:
        push rbp
        mov rbp, rsp
        sub rsp, 0x30
        …
        move eax, 0x0
        leave
        ret
main:
        …
rip ➔   call func
        mov eax, 0x0 // address 0x4005a0
        …
```

Call fun = push next_rip
jmp func



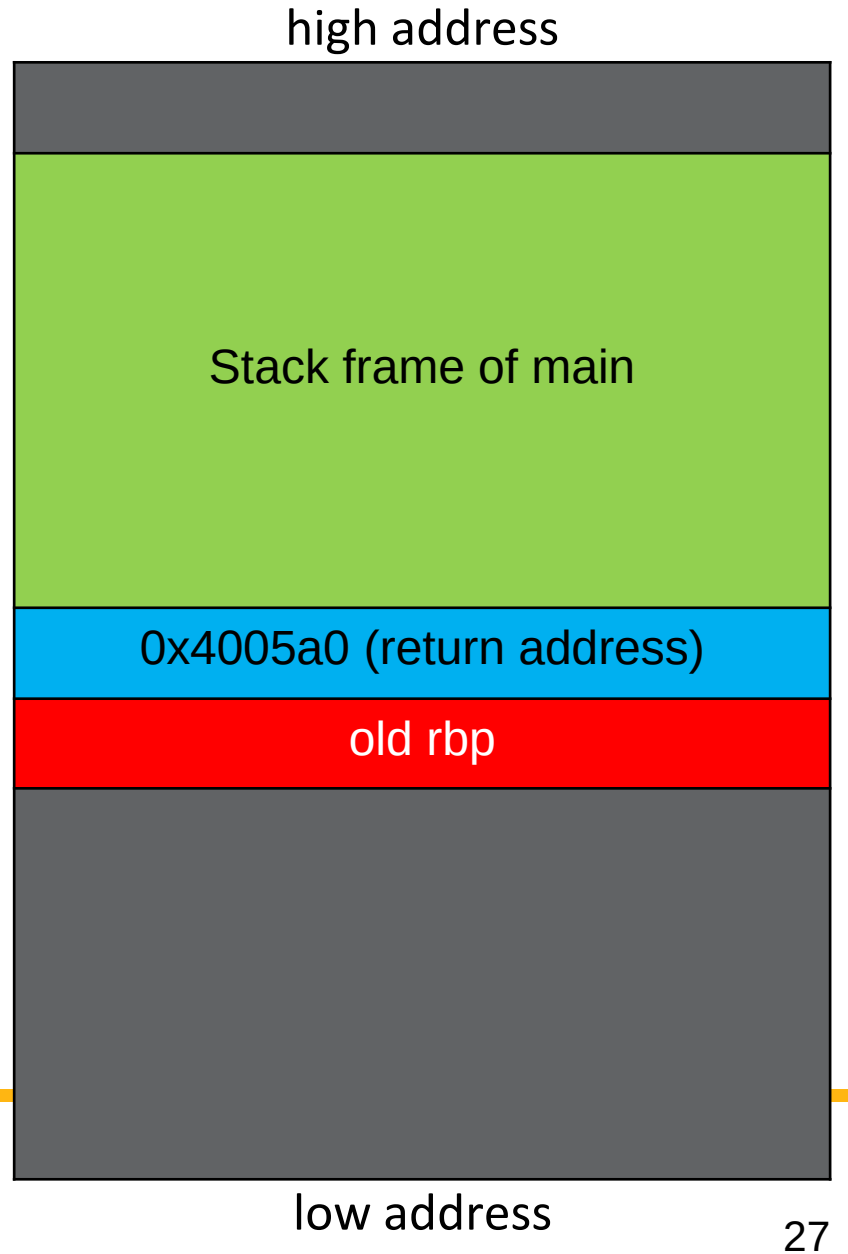
Example: Stack frame during a function call

```
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

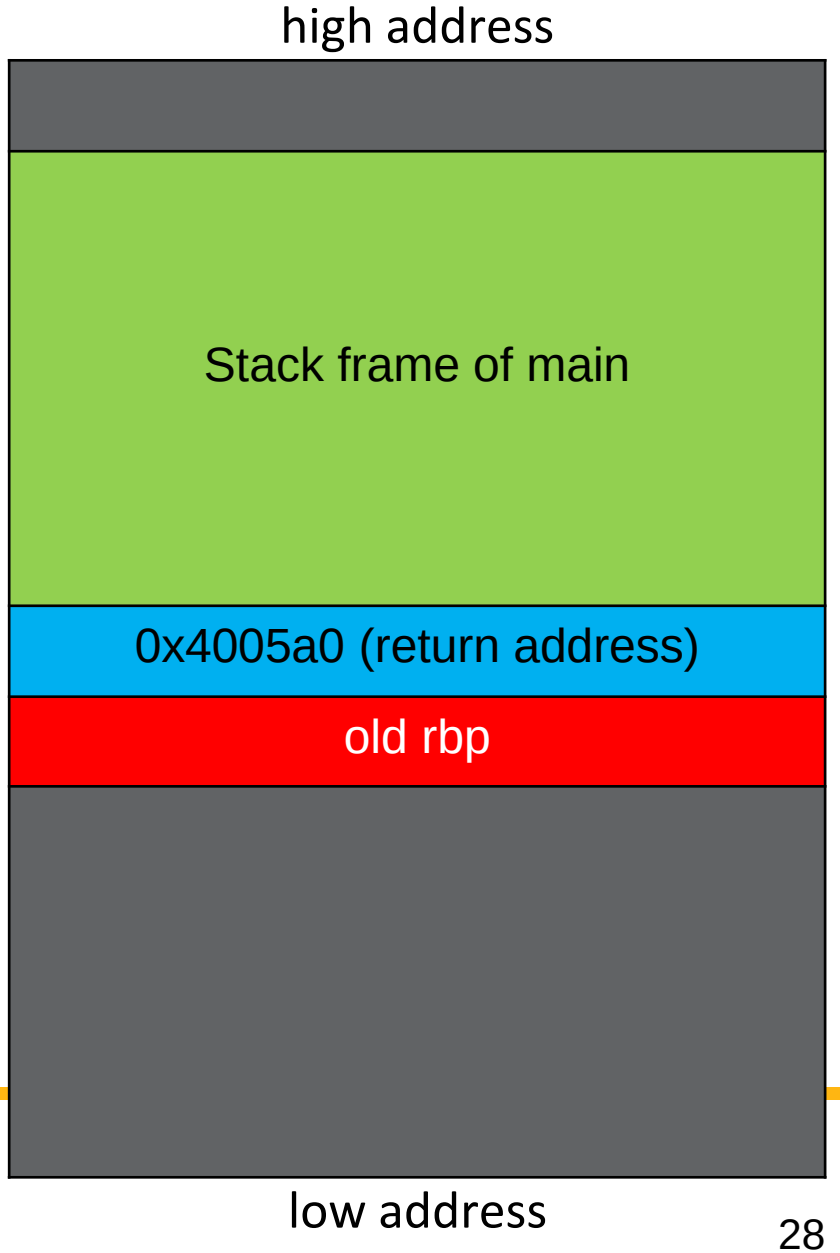
```
func:
        push rbp
rip ➔   mov rbp, rsp
        sub rsp, 0x30
        …
        move eax, 0x0
        leave
        ret
main:
        …
        call func
        mov eax, 0x0 // address 0x4005a0
        …
```



Example: Stack frame during a function call

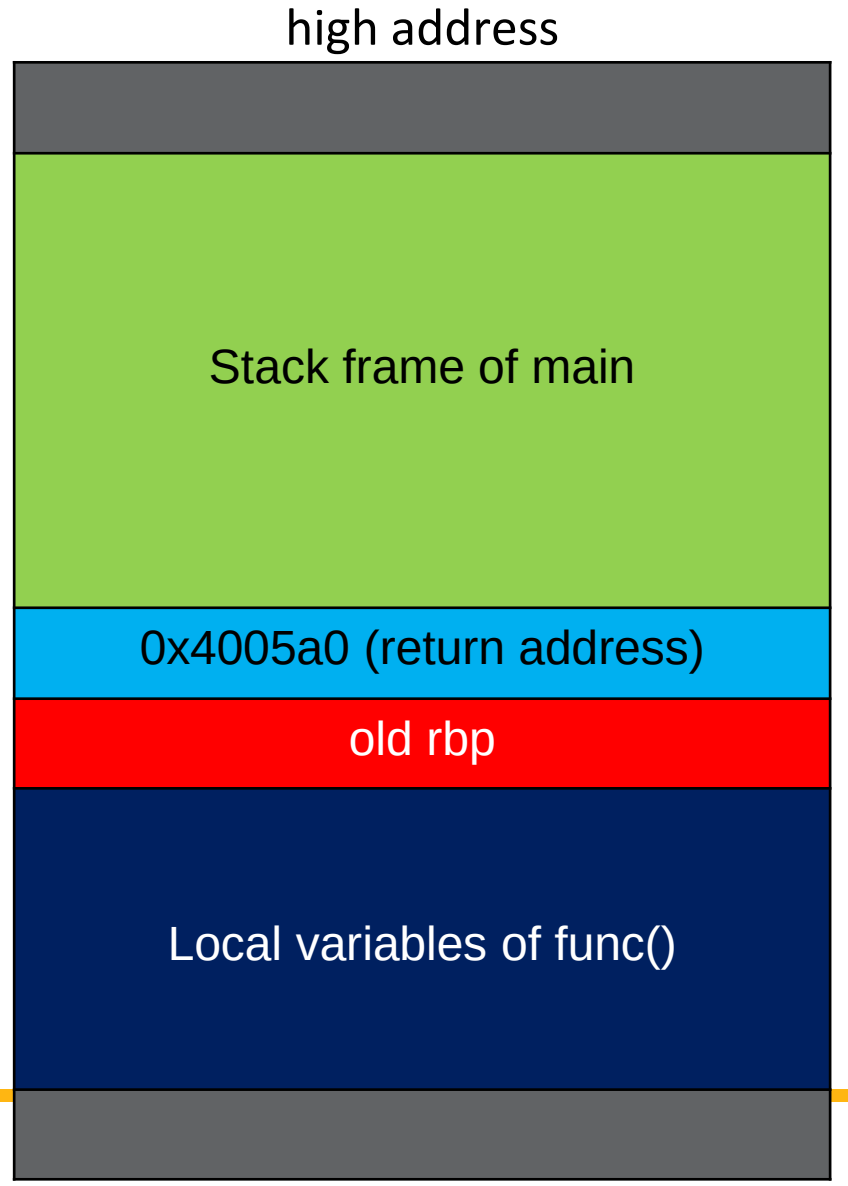
```
func:
        push rbp
        mov rbp, rsp
rip ➔   sub rsp, 0x30
        …
        move eax, 0x0
        leave
        ret
main:
        …
        call func
        mov eax, 0x0 // address 0x4005a0
        …
```

rbp ➔ rsp ➔



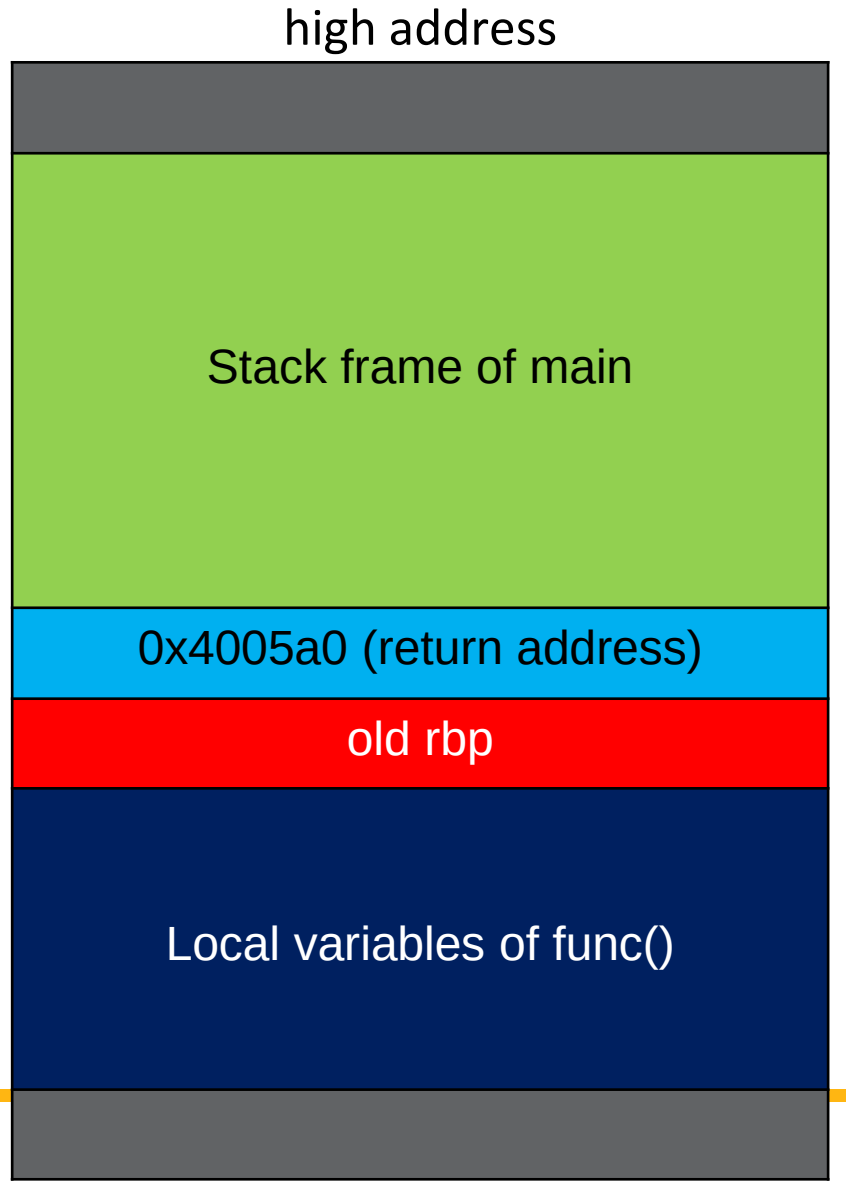
Example: Stack frame during a function call

```
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

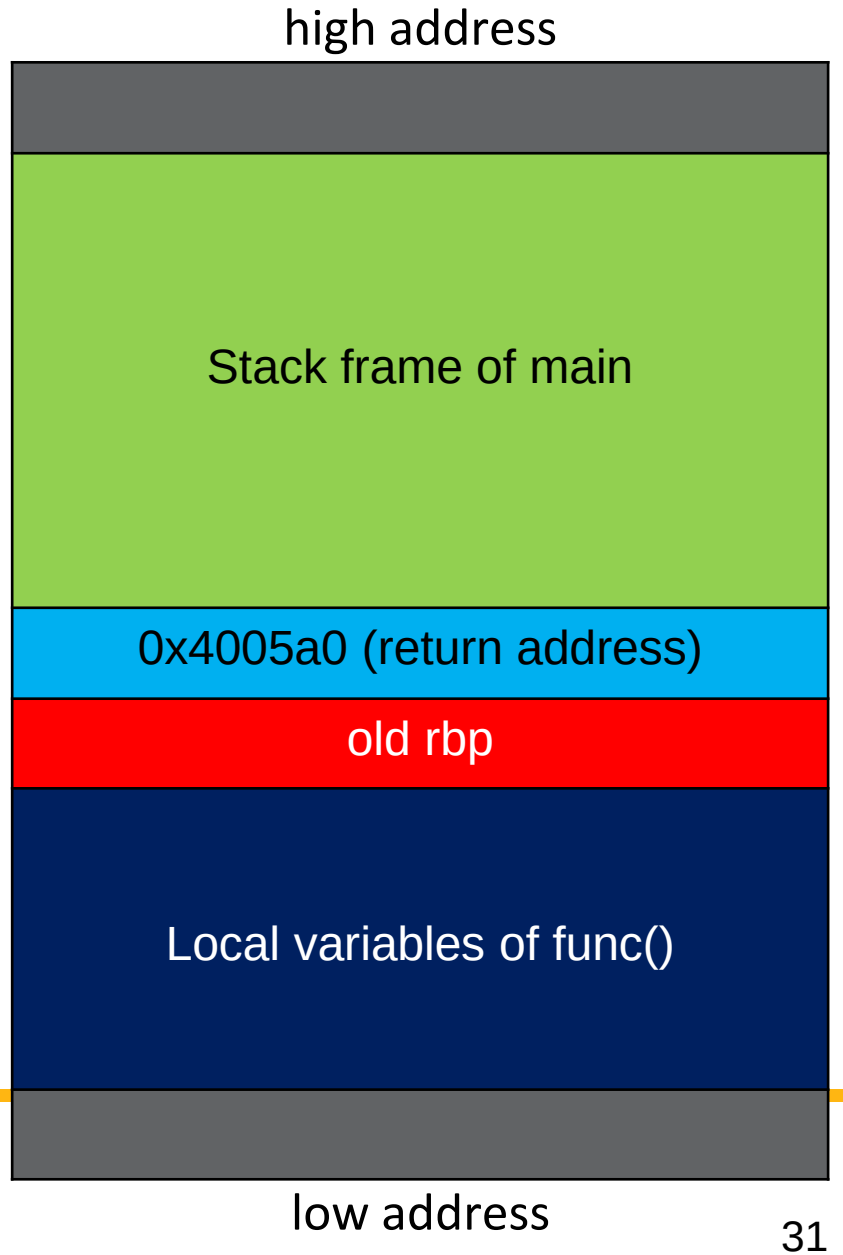
```
func:
        push rbp                    leave = mov rsp, rbp
        mov rbp, rsp                        pop rbp
        sub rsp, 0x30
        …
        move eax, 0x0
rip ➔   leave
        ret
main:
        …
        call func
        mov eax, 0x0 // address 0x4005a0
        …
```



Example: Stack frame during a function call

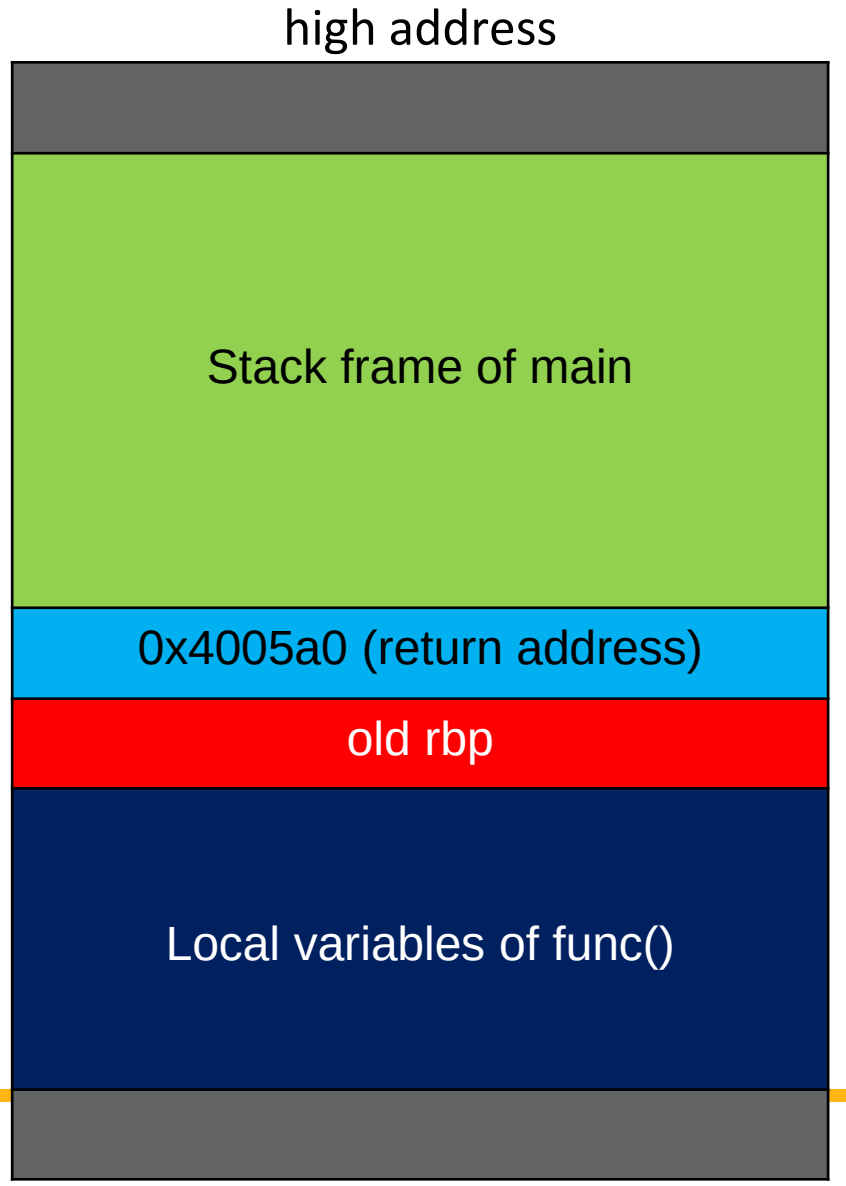
```
func:
        push rbp                    leave = mov rsp, rbp
        mov rbp, rsp                        pop rbp
        sub rsp, 0x30
        …
        move eax, 0x0
rip ➔   leave
        ret
main:
        …
        call func
        mov eax, 0x0 // address 0x4005a0
        …
```

rbp ➔ rsp ➔



Example: Stack frame during a function call

```
                                                    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
        …
```

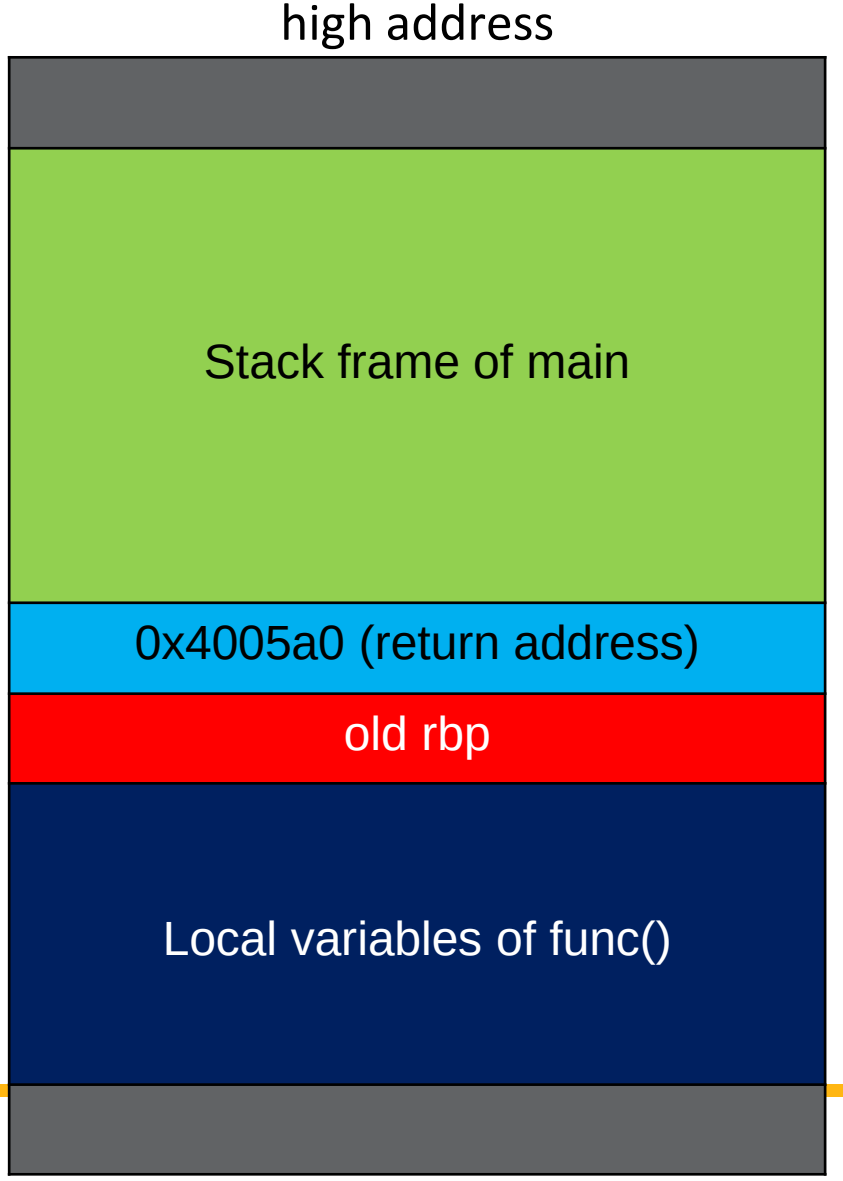


Example: Stack frame during a function call

```
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 ➔



Common Security Protection in Binary

- Canary
  - Put canary value before old rbp and return address
- PIE/ALSR
  - Randomize the address space of a process
    - The offset between different symbol still the same!!

Common Security Protection in Binary

- Relro
  - Lazy binding option for program
    - Full Relro will have GOT table read only before calling main function
    - Partial Relro make GOT table writable and resolve symbol after calling main function
- NX
  - Making stack not executable.
    - Make shellcode not able to run on stack.

Common Security Protection in Binary

- You may check the protection mechanism in binary using checksec
  - slimm609/checksec: Checksec

```
(env) huroy@build-server:~/csc2025-project4/hard_rop$ checksec --file hard_rop
[*] '/home/huroy/csc2025-project4/hard_rop/hard_rop'
    Arch:       amd64-64-little
    RELRO:      Full RELRO
    Stack:      Canary found
    NX:         NX enabled
    PIE:        PIE enabled
    SHSTK:      Enabled
    IBT:        Enabled
    Stripped:   No
    Debuginfo:  Yes
```

Example: Stack frame with canary

```
func:
        push rbp
        mov rbp, rsp
        sub rsp, 0x30
        rax,QWORD PTR fs:0x28
        QWORD PTR [rbp-0x8],rax
rip ➔   ...
        rax,QWORD PTR [rbp-0x8]
        rax,QWORD PTR fs:0x28
        call <__stack_chk_fail@plt>
        leave
        ret
```