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 (return-oriented program.)，漏洞利用技術.

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

允許 attacker 在程式啟用了安全保護技術 (ex: stack 不能執行) 的情況下，執行惡意 code.

- 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

利用 bufferoverflow 控制 stack 的方式改變程式控制流，並執行惡意的機器語言指令序列 (gadgets)

所謂 gadgets，就是以 ret (return) 結尾的指令序列.

因為程式沒有做 boundary check，而導致資料可以超過邊界。

x86 (32 bit) 裡面的暫存器 (register)：

EIP (instruction pointer)：目前執行指令的位址

EBP (base pointer)：目前 stack 的 base

ESP (stack pointer)：目前 stack 的 top.

x64 (64 bit)：前面改成 R，RIP，RBP...

libc.so.6：是 linux 系統中一個核心的共享程式庫(shared library)，是 GNU C library 的其中一部份。

- libc = C standard library
- so：shared object (類似 windows 上的 .dll)

它提供了執行 Linux 絕大多數所須的功能。

ex：

① 記憶體管理 (ex: malloc, free)

② 檔案操作 (ex: open, read, write)

③ 字串處理 (ex: strlen, strcpy)

不應該刪除它，因為整個系統的執行檔都須要它

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

兩者皆為分析 ELF 檔案的工具

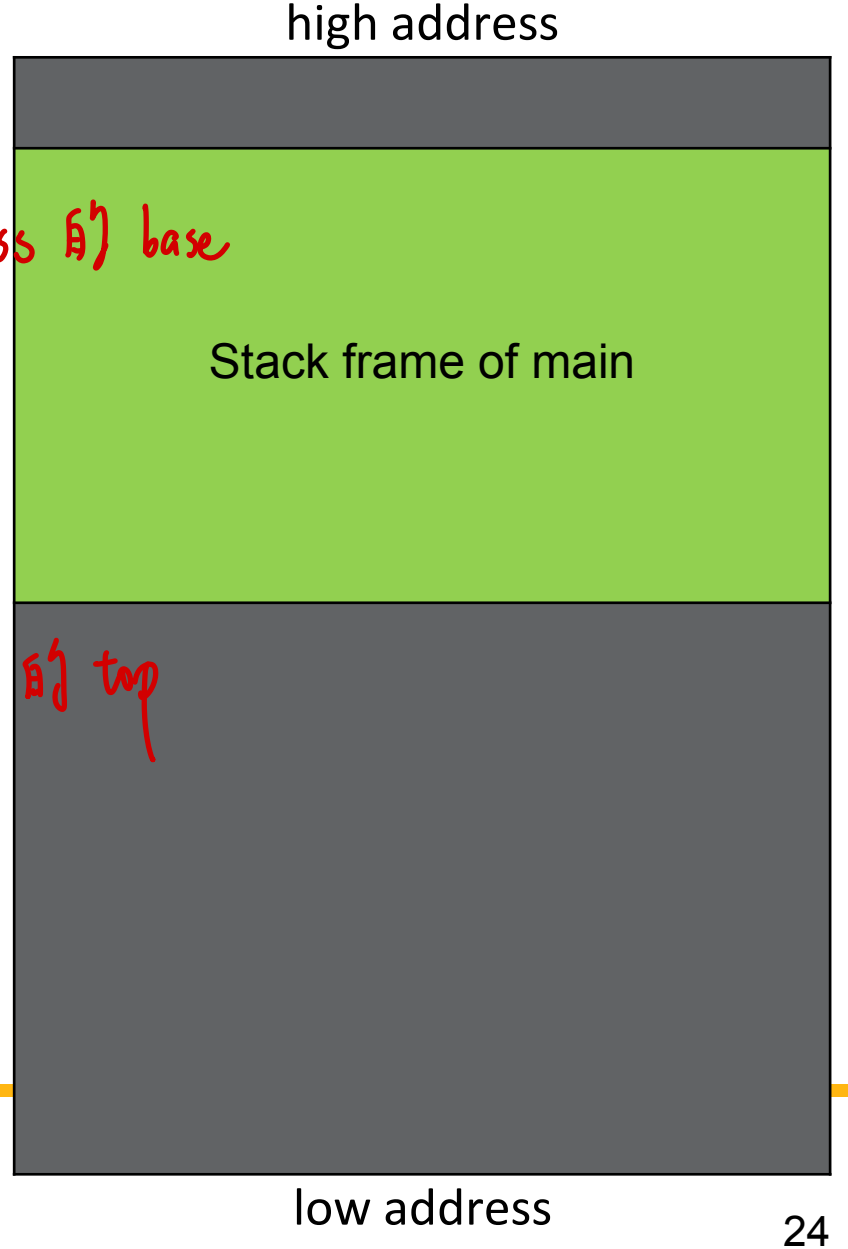
elf 是 Linux & Unix 常見的執行格式，表示程式，library，等可執行或連接的 binary file ex: .out , .so ls.

checksec	低階 ELF 結構，headers, symbols
readelf	高階，檢查 elf 安全機制啟用情況 ex: relro, NX, PIE.

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
        …
```

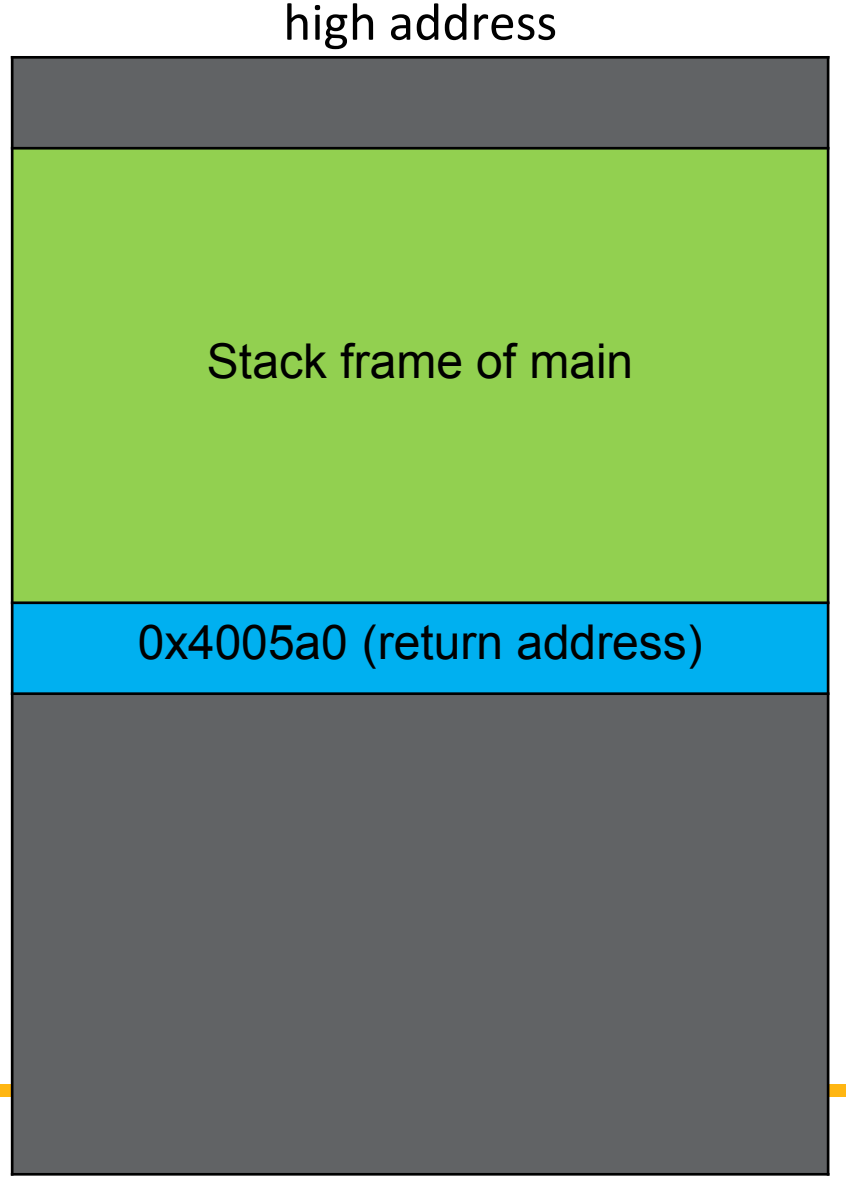
Call fun = push next_rip
jmp func



Example: Stack frame during a function call

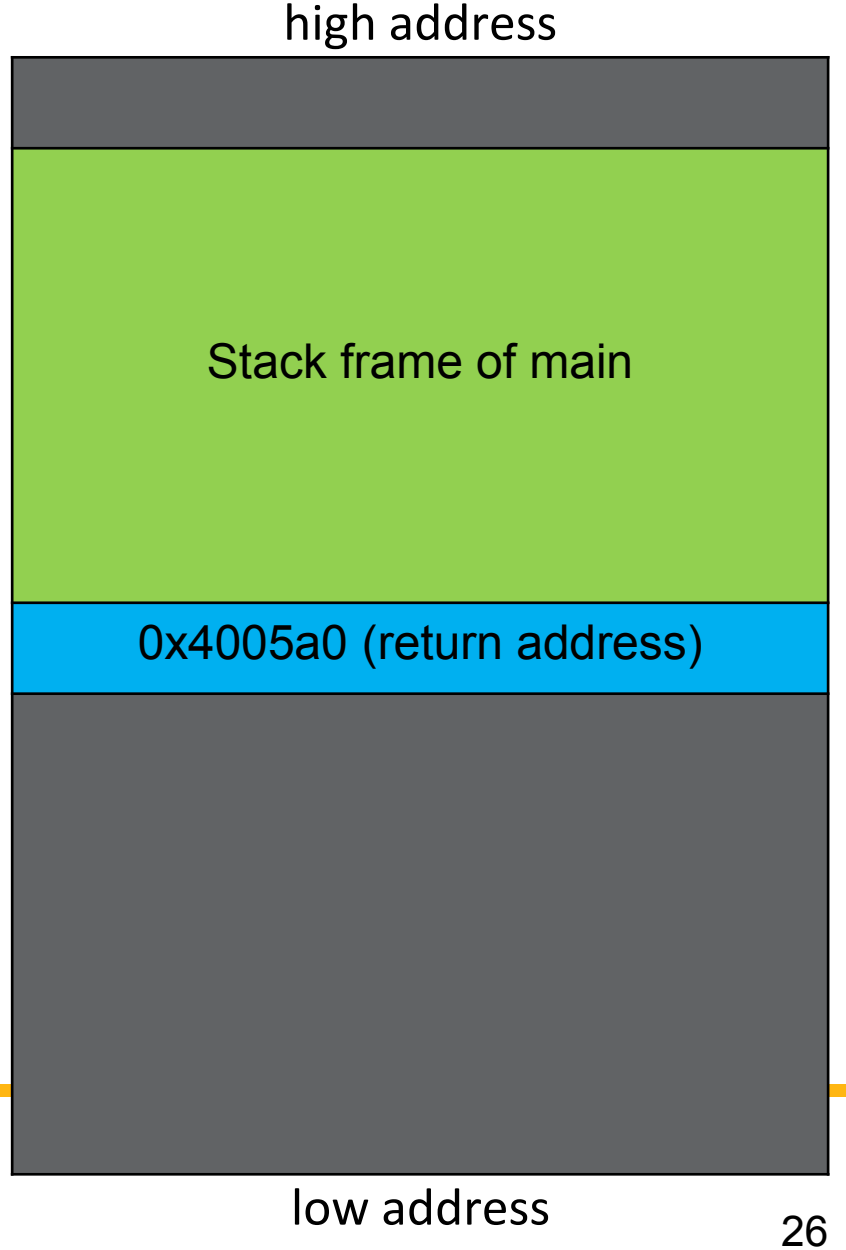
```
func:
        push rbp
        mov rbp, rsp
        sub rsp, 0x30
        …
        move eax, 0x0
        leave
        ret
main:
        …
rip ➔   call func
        mov eax, 0x0 // address 0x4005a0
        …
```

Call fun = push next_rip
jmp func



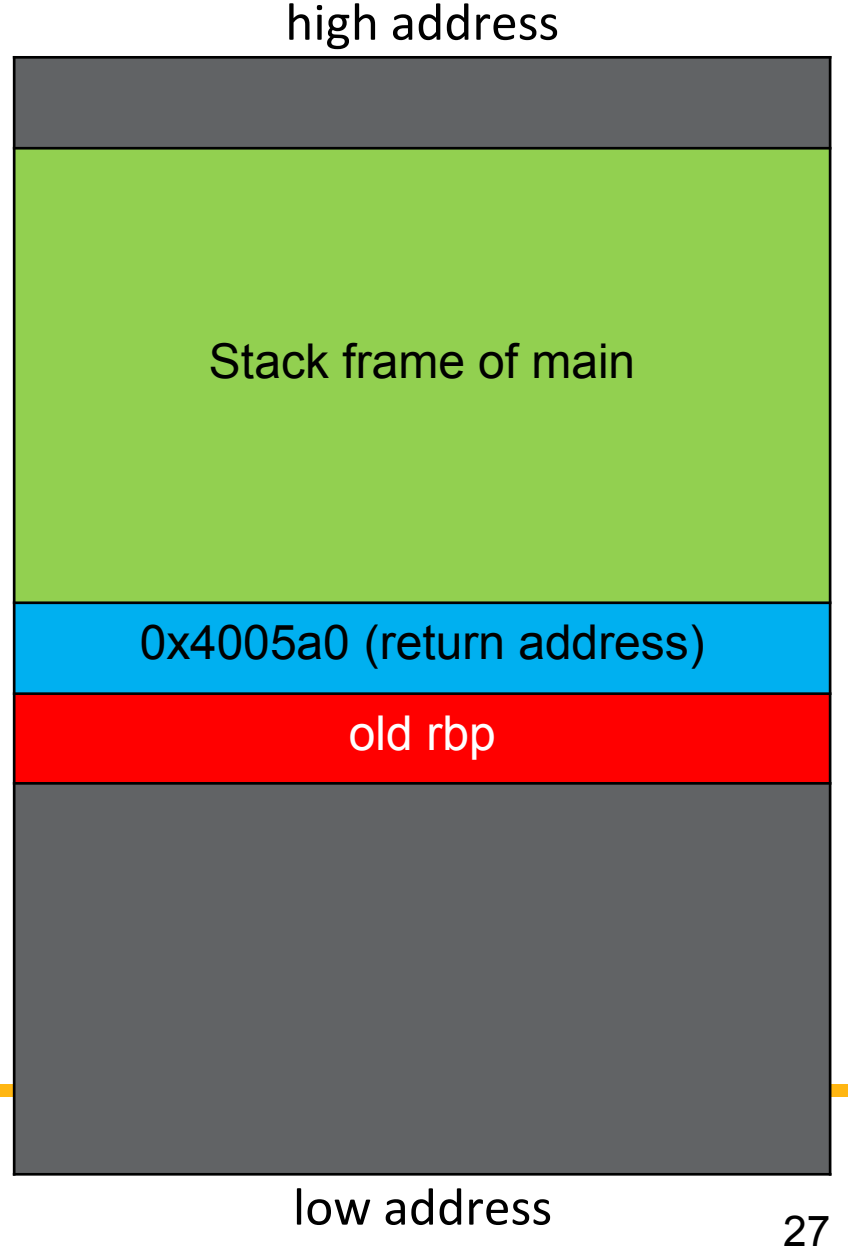
Example: Stack frame during a function call

```
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

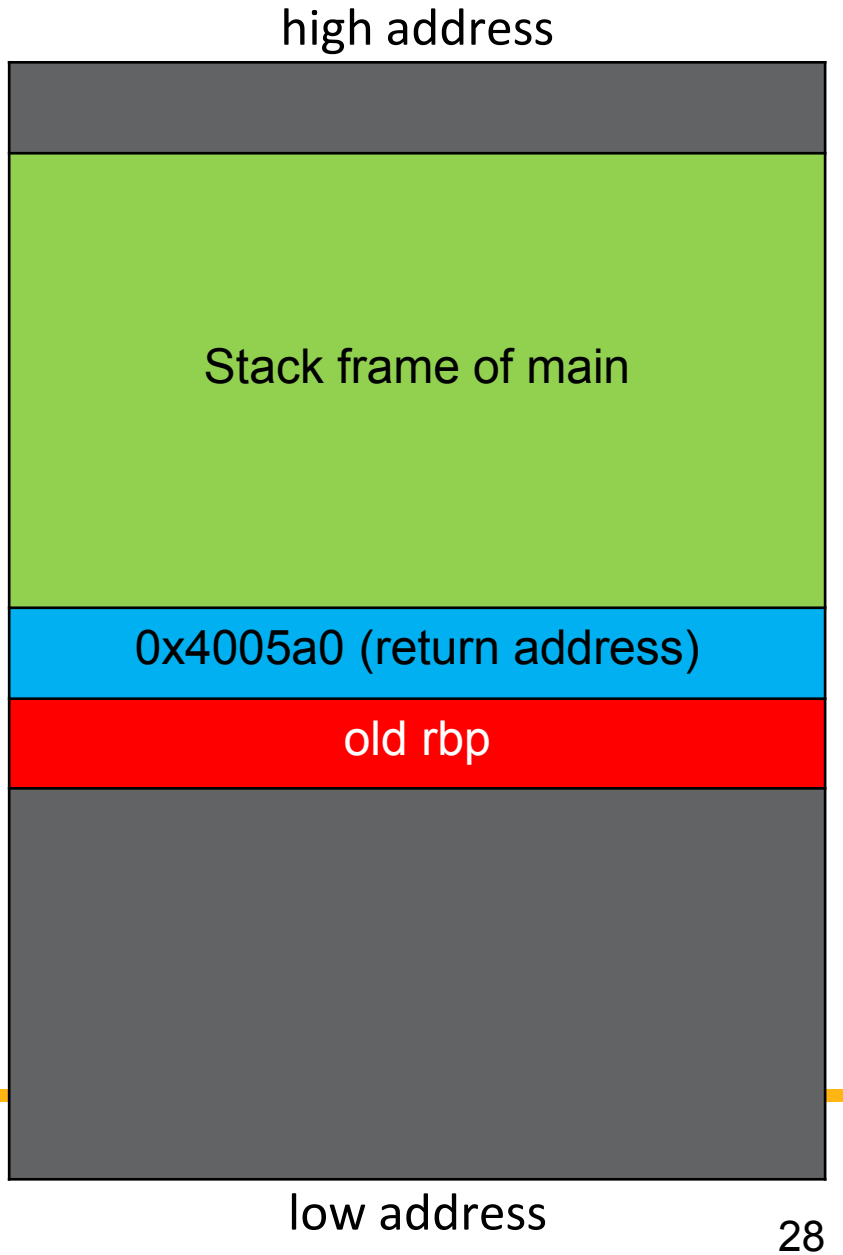
```
func:
        push rbp
rip ➔   mov rbp, rsp
        sub rsp, 0x30
        …
        move eax, 0x0
        leave
        ret
main:
        …
        call func
        mov eax, 0x0 // address 0x4005a0
        …
```



Example: Stack frame during a function call

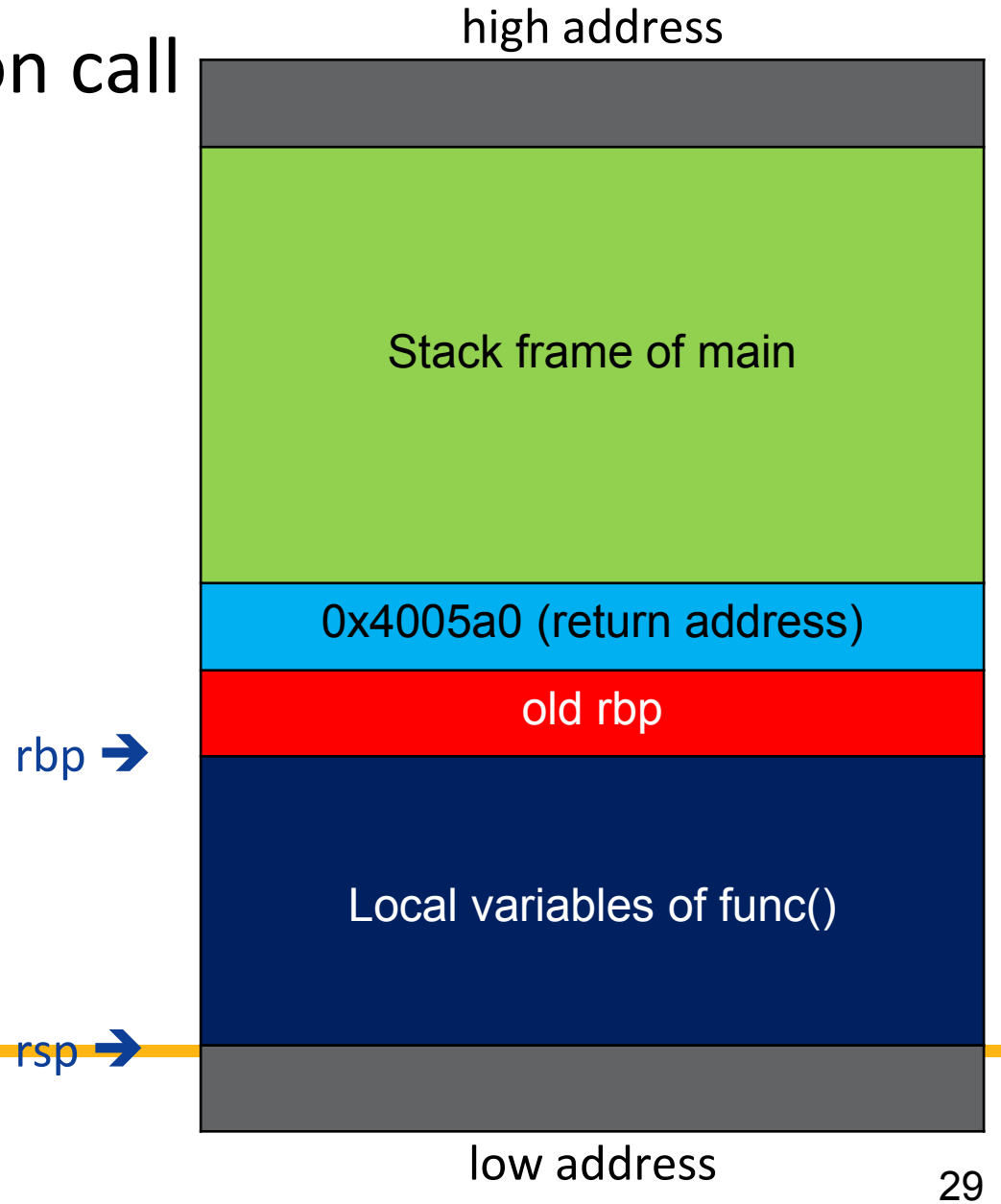
```
func:
        push rbp
        mov rbp, rsp
rip ➔   sub rsp, 0x30
        …
        move eax, 0x0
        leave
        ret
main:
        …
        call func
        mov eax, 0x0 // address 0x4005a0
        …
```

rbp ➔ rsp ➔



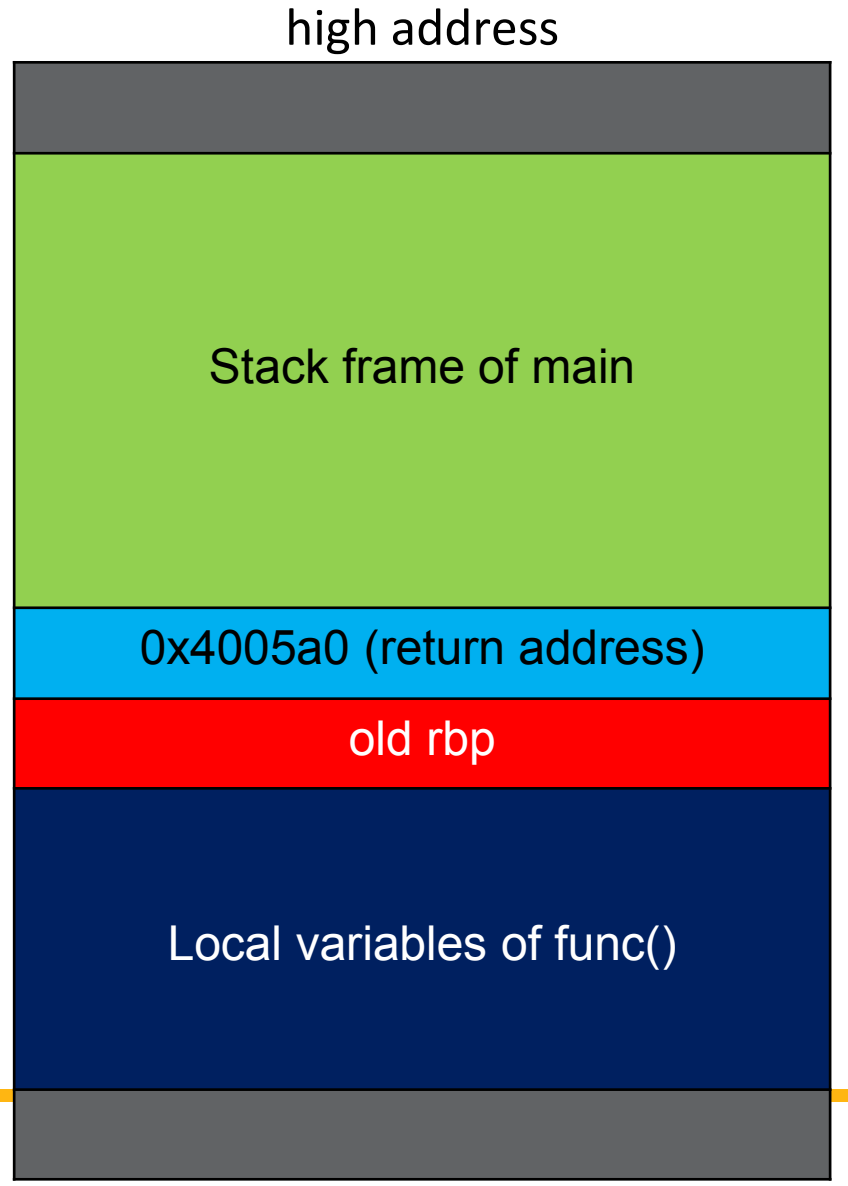
Example: Stack frame during a function call

```
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

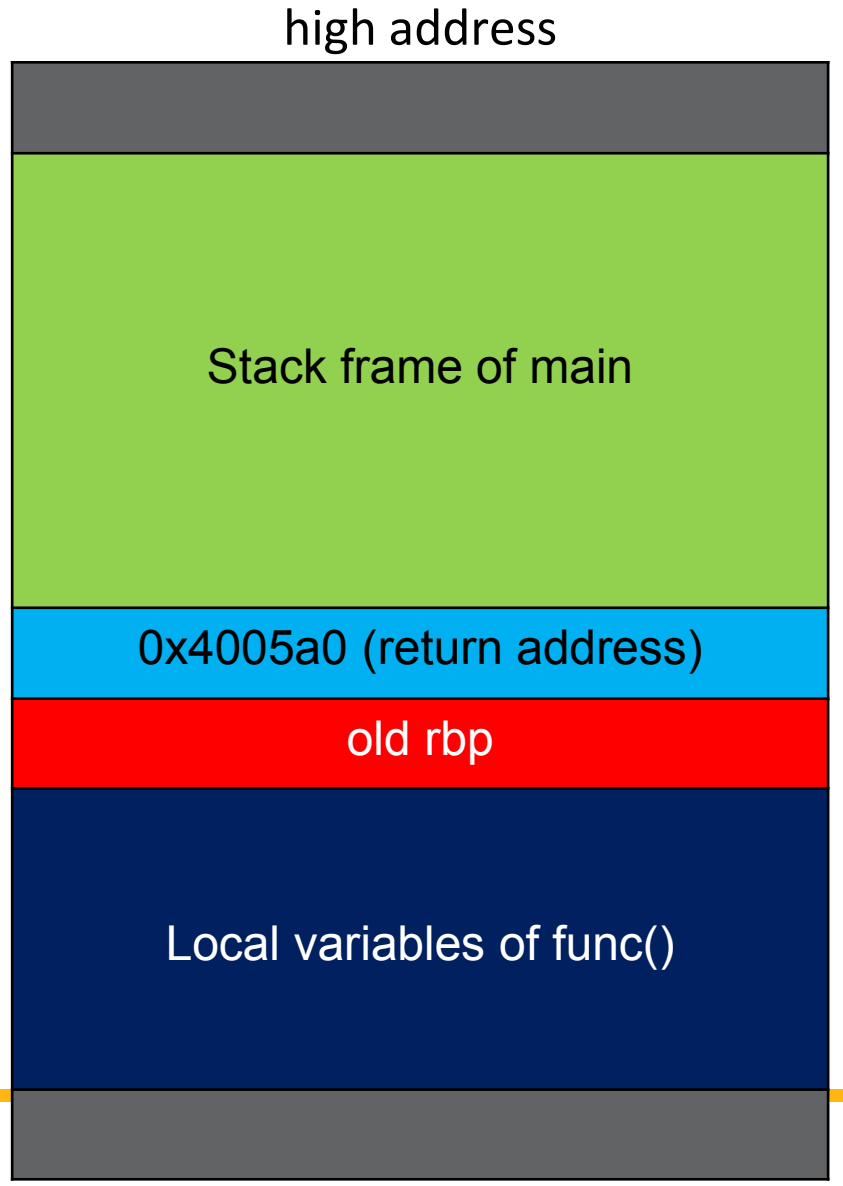
```
func:
        push rbp                    leave = mov rsp, rbp
        mov rbp, rsp                        pop rbp
        sub rsp, 0x30
        …
        move eax, 0x0
rip ➔   leave
        ret
main:
        …
        call func
        mov eax, 0x0 // address 0x4005a0
        …
```



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 ➔

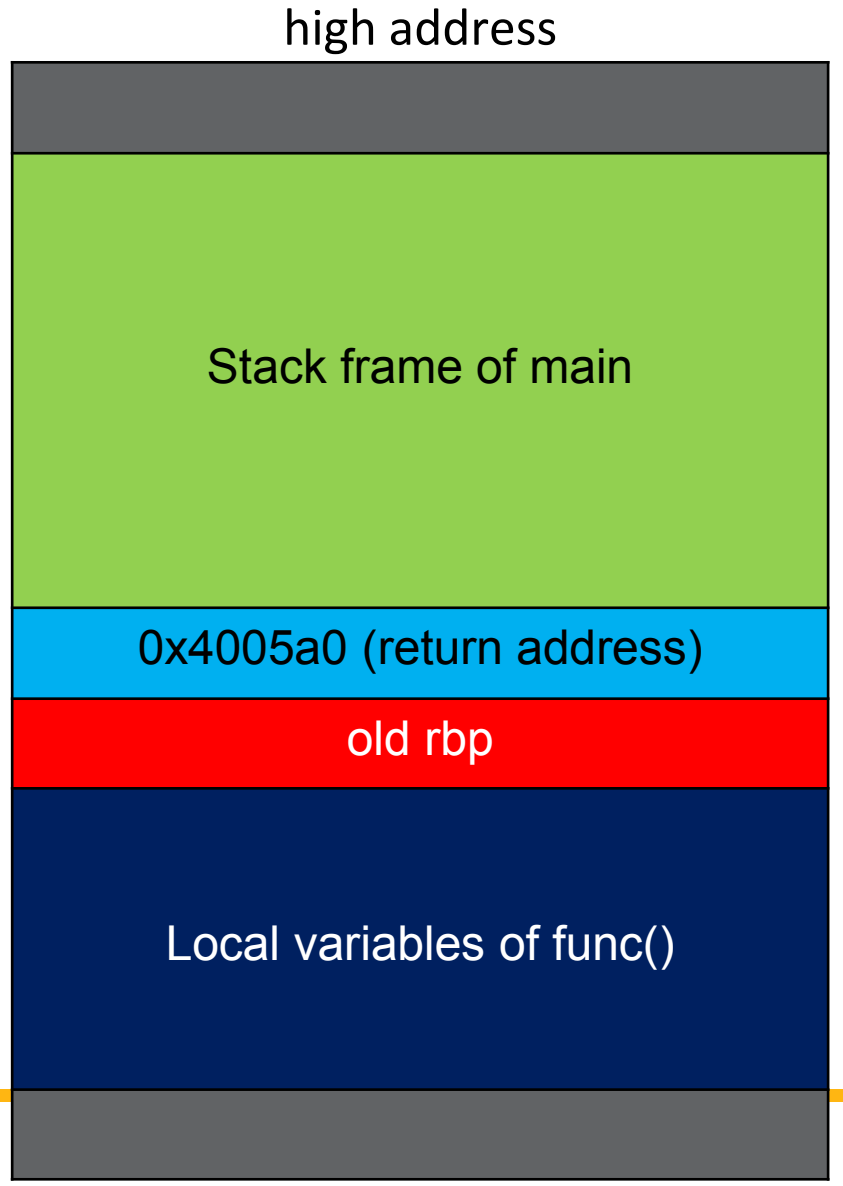


Example: Stack frame during a function call

```
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 ➔

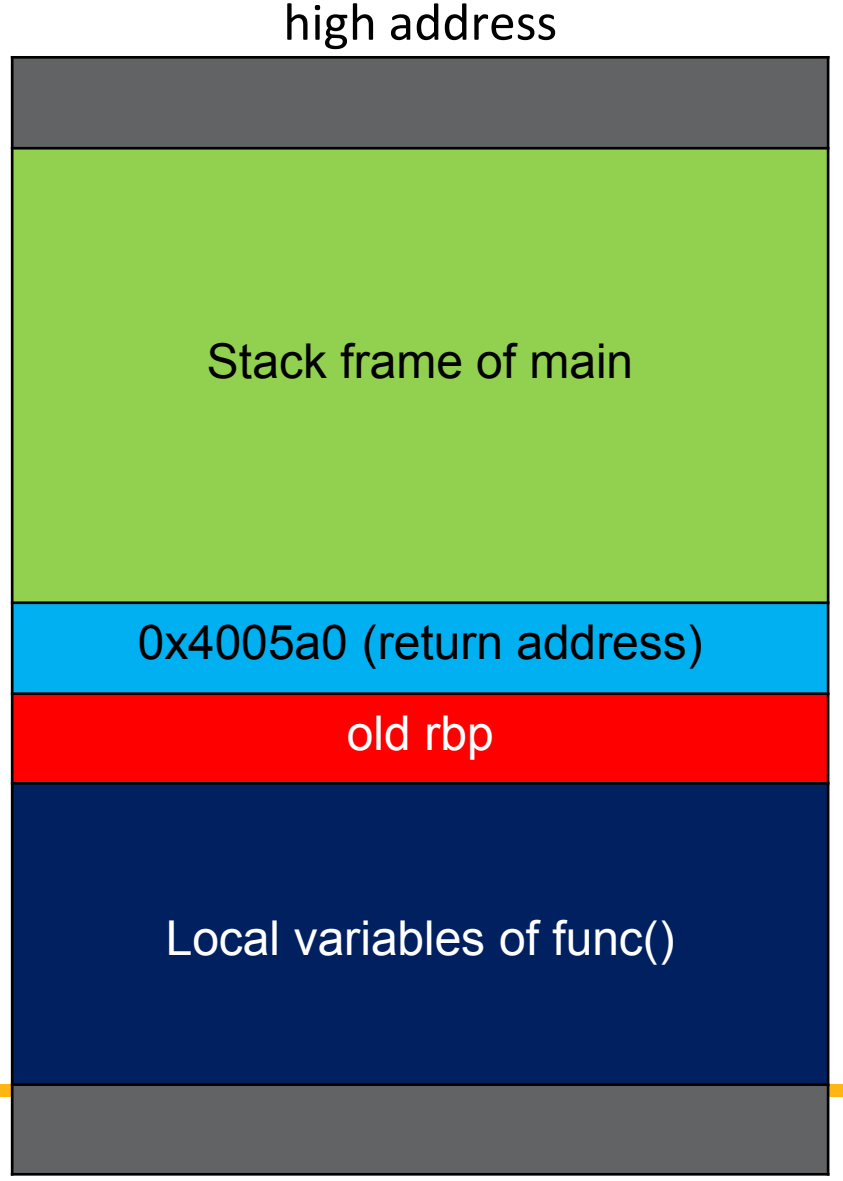


Example: Stack frame during a function call

```
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 ➔



Common Security Protection in Binary

- Canary (堆疊保護：防止 stack buffer overflow 攻擊的保護機制，在 stack 中放置一個"金絲雀值"，在函式返回前查看是否被改寫，以用來偵測攻擊.)
  - Put canary value before old rbp and return address (放在 return address 前面) (canary value)
- PIE/ALSR (Position Independent Executable，增強 OS 安全性的技術。讓可執行檔能夠和 shared library 一樣，在 memory 中任意的位置執行。使用相對位址 (relative addressing))
  - Randomize the address space of a process
    - The offset between different symbol still the same!!

program 之間的記憶体空間是獨立的，都有屬於自己的 virtual memory space，由 OS 和 MMU 隔離出來。

memory manage unit

也就是說，不同程式間的"虛擬位址"可能一樣。但實際位址不同，由 MMU 管理。

+

page Table

給定非 PIE 的 program，程式碼會載入到 0x400000

Common Security Protection in Binary

- Relro Relocation Read-Only，針對 GOT (Global Offset Table) 實施保護的機制。防止 GOT overwrite.
  - Lazy binding option for program GOT 是 ELF 格式中的一個表格，存放動態函式的實際位址 (printf, system)。若 attacker 能改寫 GOT，能讓 program 呼叫任意代碼執行。
    - 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：讓 stack 不可被執行。無法直接丟入 shell code 進行攻擊。
  - 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

file name

該 binary file 的特性

```
(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
```

canary value 被改動
後就知道此程式
有問題 !!