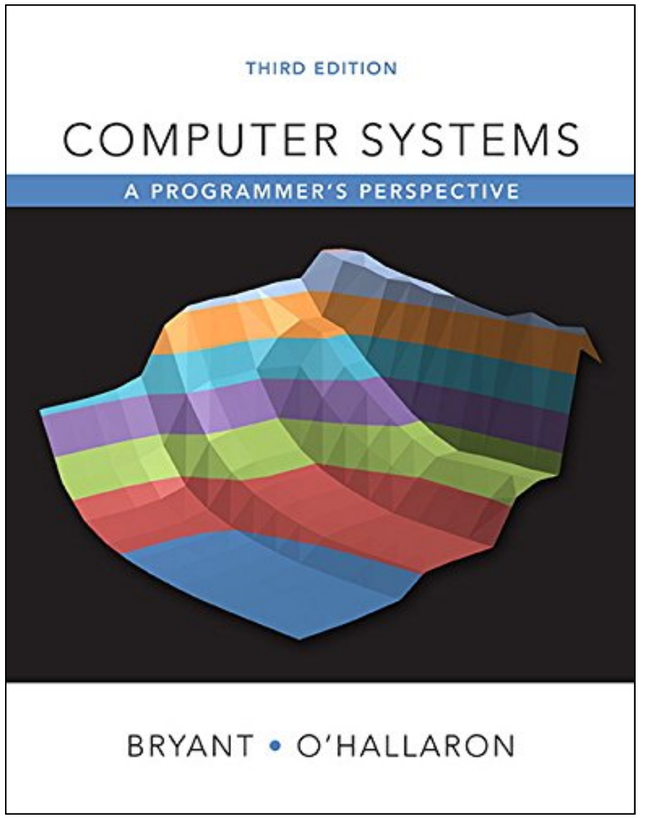
Assembly Code Analysis

Motivation

Motivation

由於前陣子看了一本名為 Computer Systems A Programmer's Perspective (CS:APP)，
讀到關於 x86-64 assembly code 的說明，以及在 function calling 下，
stack 與 registers 會如何變化。

因此想藉此期中報告的實作，觀察 assembly code 與 register
來驗證書中所說明是否為真，也能加強自身對於計算機領域的基本知識理解。



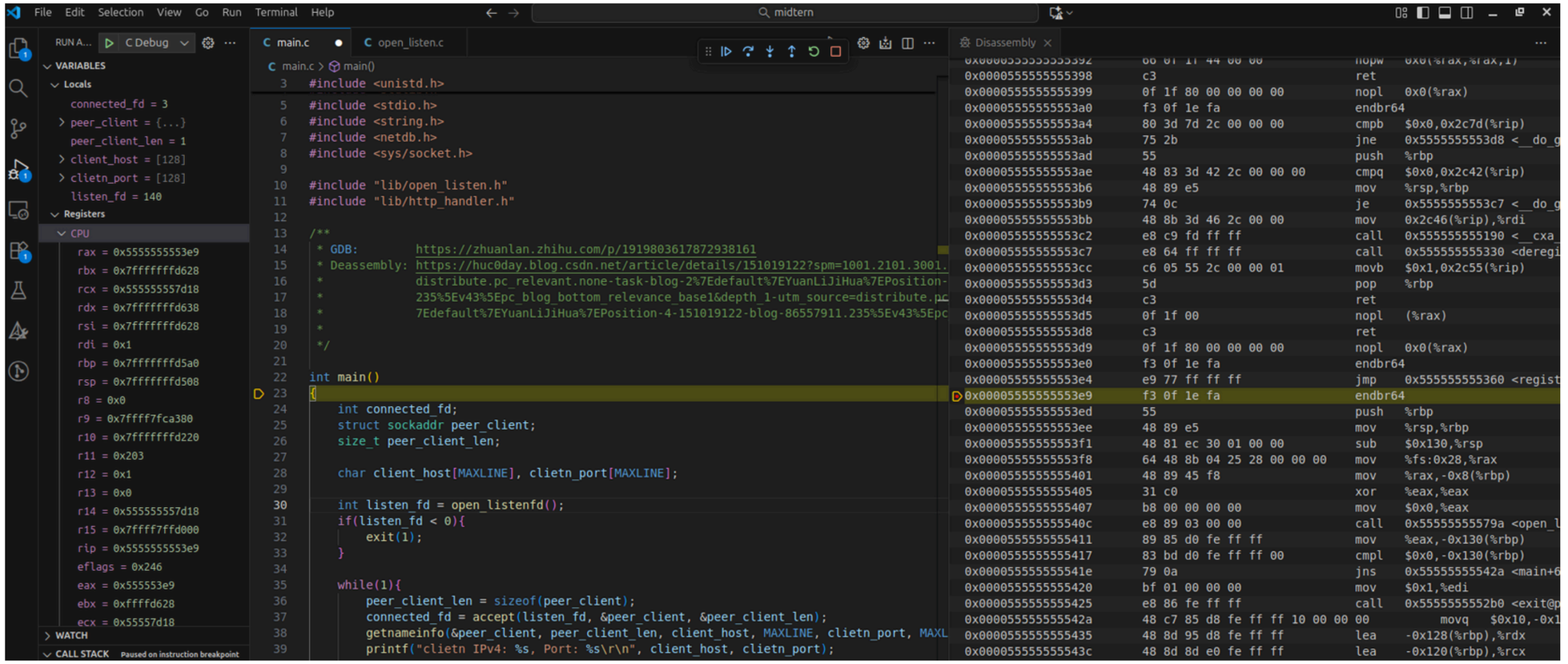
https://www.tenlong.com.tw/products/9780134092669

https://www.cs.sfu.ca/~ashriram/Courses/CS295/assets/books/CSAPP_2016.pdf

Method

Method

藉由 VScode 做調適 (Debug)，並對照可執行檔 (executable file) 的 deassembly (解彙編) 逐行執行並觀察，
藉由理解 assembly code 的執行與猜測 register value 的變化並藉由 debug mode 提供逐行執行驗證猜測。



Trace

Trace

雖然在 x86-64 中有許多通用 register，但只需要注意幾個重要的即可。

- %rax → 儲存 return value
- %rsp → 儲存棧頂部
- %rbp → 儲存當前 function 的起始棧位置
- %rip → 儲存 process 執行位置 (PC)
- %rdi, rsi, rdx, rcx, r8, r9 → 儲存 function 參數

Figure 3.2 **Integer registers.** The low-order portions of all 16 registers can be accessed as byte, word (16-bit), double word (32-bit), and quad word (64-bit) quantities.

Trace

首先來看 main function

13ed → 將 %rbp 內容儲存到 stack 中 (push 會將 %rsp - 0x08 再將 %rbp 內容儲存到 stack)

13ee → 將 %rsp 的值複製到 %rbp 中

13f1 → 將 %rsp 減去 0x130，用於分配 stack space 給 main function

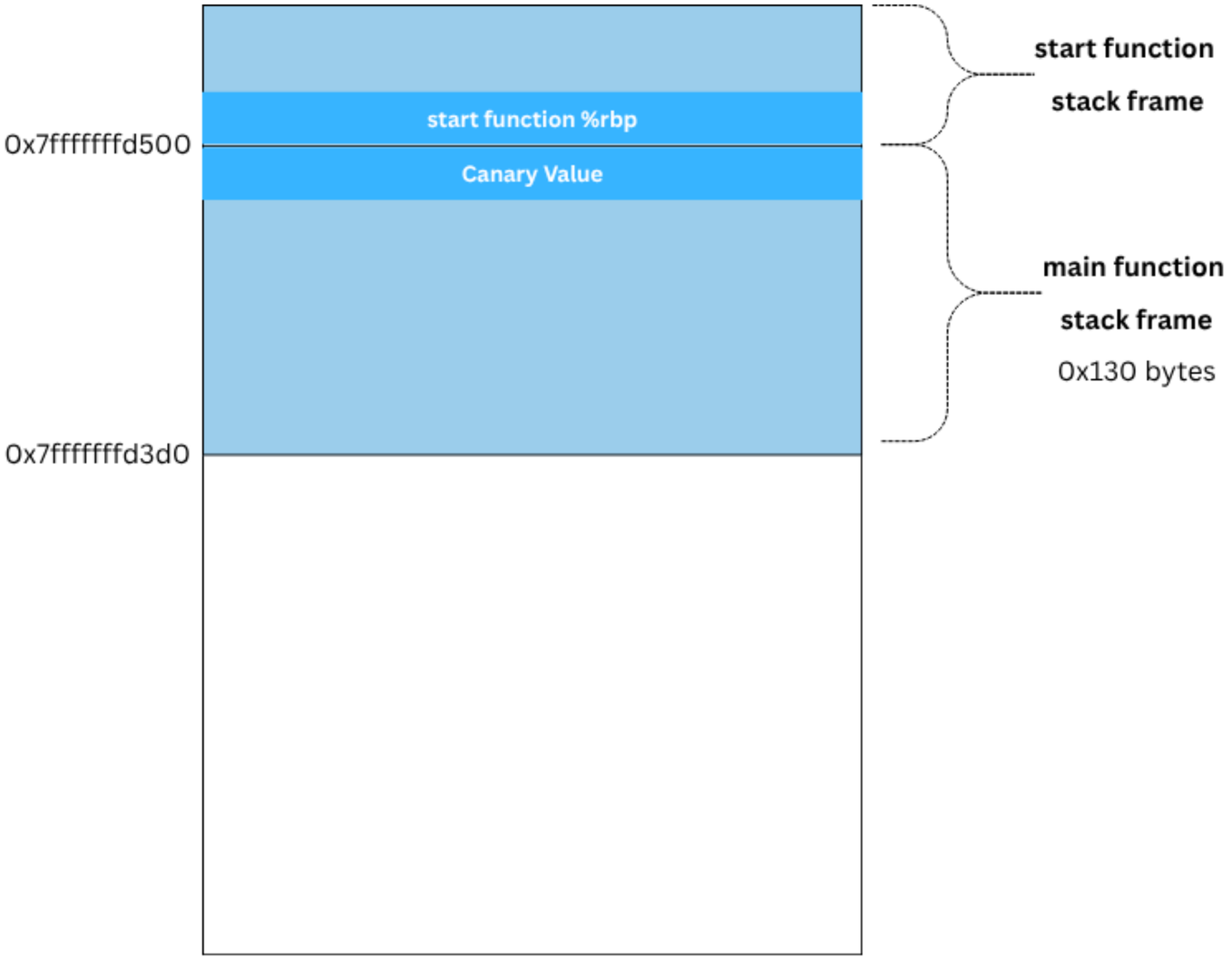
13f8 → 將 Canary value 複製到 %rax 中

1401 → 將 %rax 值複製到 %rbp-0x8 的位置中

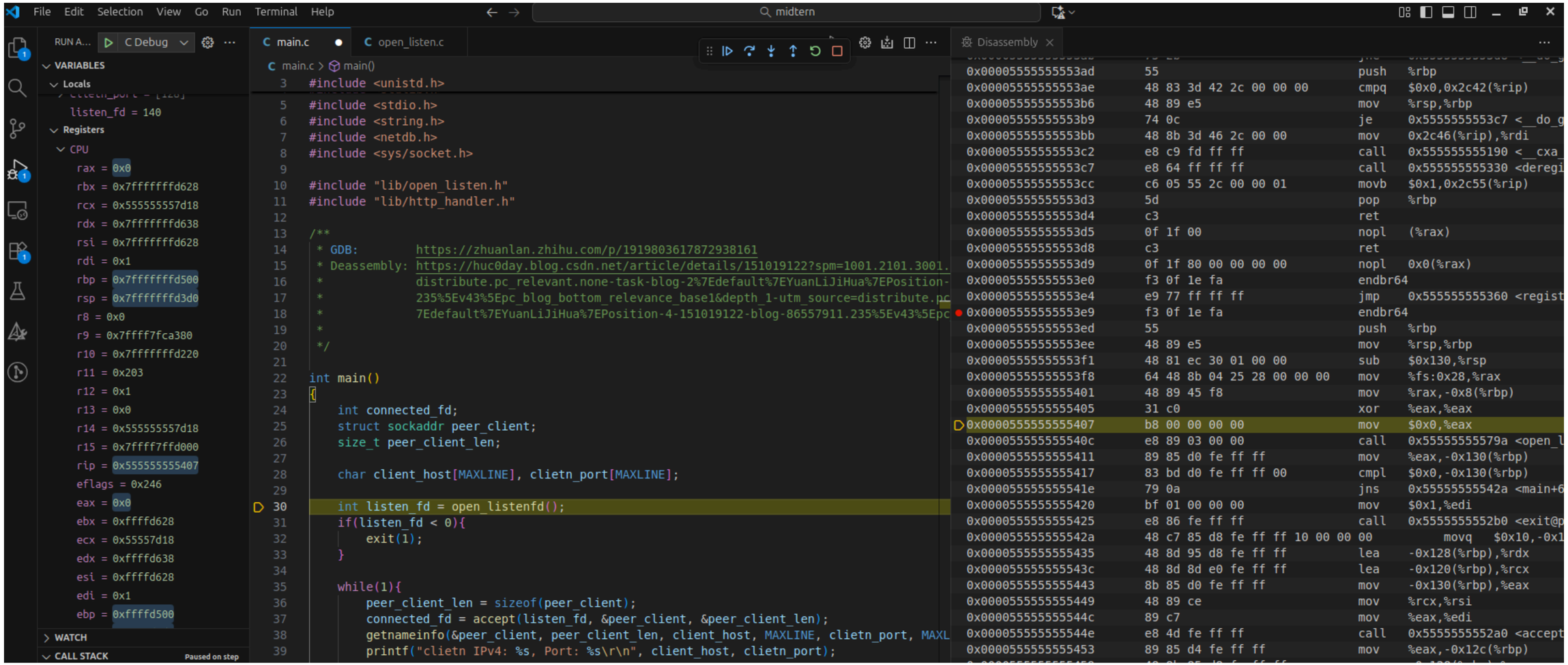
1405 → 將 %rax 清除為 0

```
int main()
{
    13e9:       f3 0f 1e fa             endbr64
    13ed:       55                      push   %rbp
    13ee:       48 89 e5                mov    %rsp,%rbp
    13f1:       48 81 ec 30 01 00 00    sub    $0x130,%rsp
    13f8:       64 48 8b 04 25 28 00    mov    %fs:0x28,%rax
    13ff:       00 00
    1401:       48 89 45 f8             mov    %rax,-0x8(%rbp)
    1405:       31 c0                   xor    %eax,%eax
```

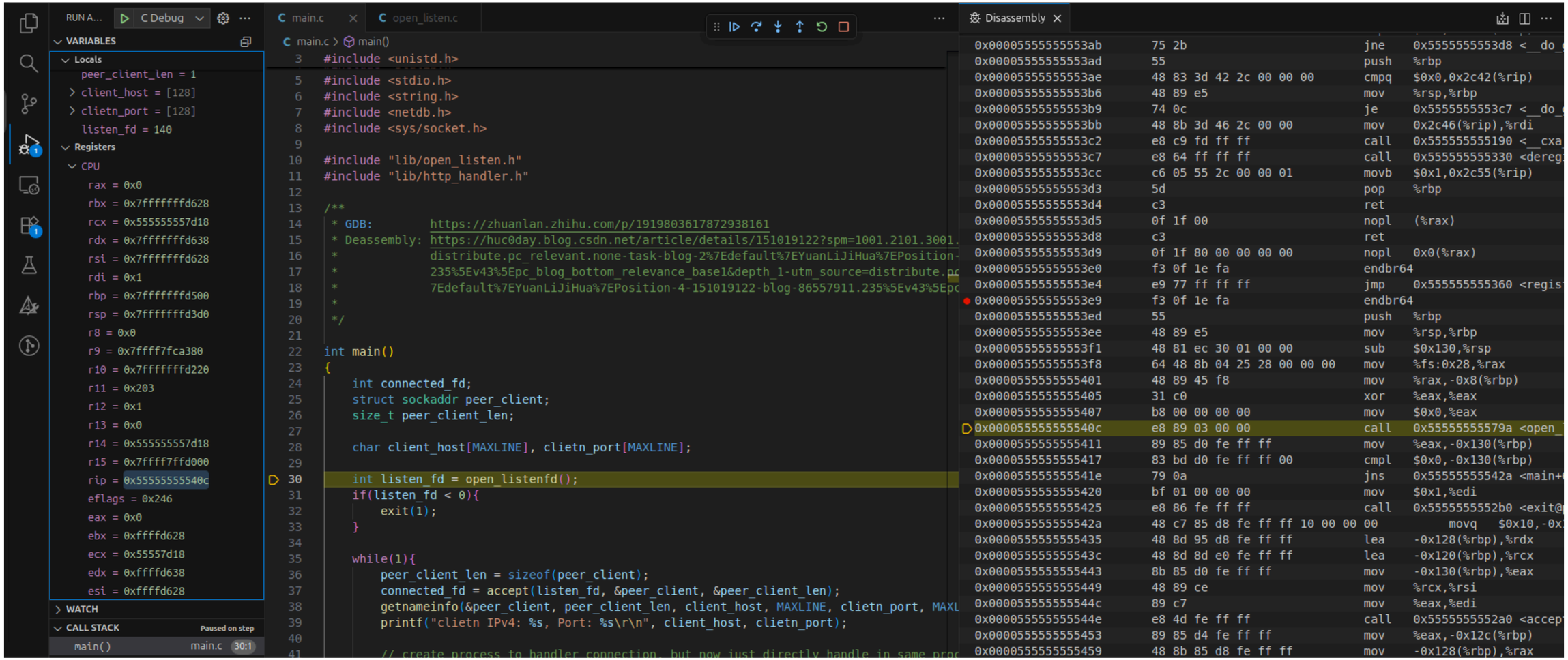
Trace


start function %rbp
Canary Value
0x7fffffffd500
0x7fffffffd3d0
start function
stack frame
main function
stack frame
0x130 bytes

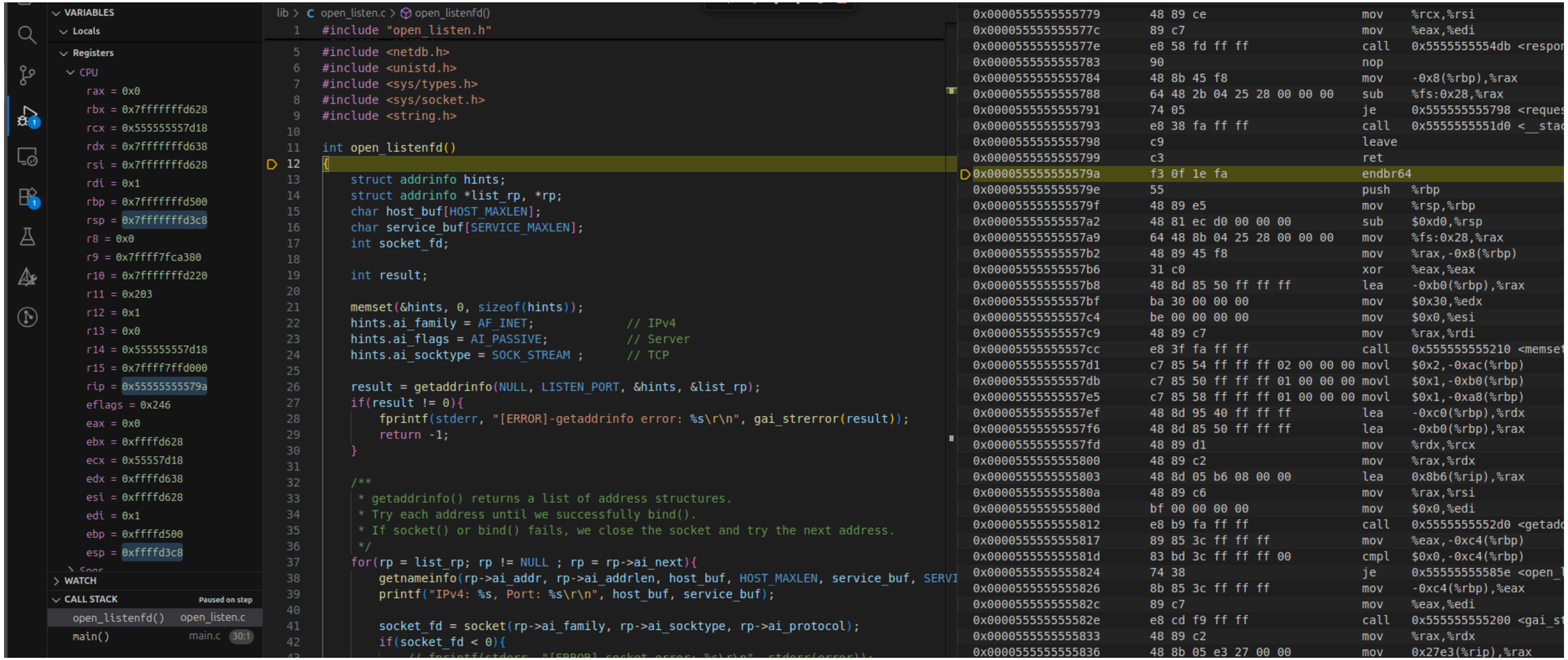
Trace


File Edit Selection View Go Run Terminal Help
midtern
RUN A... C Debug
VARIABLES
Locals
listen_fd = 140
Registers
CPU
rax = 0x0
rbx = 0x7fffffffd628
rcx = 0x555555557d18
rdx = 0x7fffffffd638
rsi = 0x7fffffffd628
rdi = 0x1
rbp = 0x7fffffffd500
rsp = 0x7fffffffd3d0
r8 = 0x0
r9 = 0x7ffff7fca380
r10 = 0x7fffffffd220
r11 = 0x203
r12 = 0x1
r13 = 0x0
r14 = 0x555555557d18
r15 = 0x7ffff7ffd000
rip = 0x555555555407
eflags = 0x246
eax = 0x0
ebx = 0xffffd628
ecx = 0x55557d18
edx = 0xffffd638
esi = 0xffffd628
edi = 0x1
ebp = 0xffffd500
WATCH
CALL STACK
Paused on step
main.c
open_listen.c
main.c > main()
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include <netdb.h>
#include <sys/socket.h>
#include "lib/open_listen.h"
#include "lib/http_handler.h"
/**
* GDB: https://zhuanlan.zhihu.com/p/1919803617872938161
* Deassembly: https://huc0day.blog.csdn.net/article/details/151019122?spm=1001.2101.3001.
* distribute.pc_relevant.none-task-blog-2%7Edefault%7EYuanLiJiHua%7EPosition-
* 235%5Ev43%5Epc_blog_bottom_relevance_base1&depth_1-utm_source=distribute.pc
* 7Edefault%7EYuanLiJiHua%7EPosition-4-151019122-blog-86557911.235%5Ev43%5Epc
*
*/
int main()
{
int connected_fd;
struct sockaddr peer_client;
size_t peer_client_len;
char client_host[MAXLINE], clietn_port[MAXLINE];
int listen_fd = open_listenfd();
if(listen_fd < 0){
exit(1);
}
while(1){
peer_client_len = sizeof(peer_client);
connected_fd = accept(listen_fd, &peer_client, &peer_client_len);
getnameinfo(&peer_client, peer_client_len, client_host, MAXLINE, clietn_port, MAXL
printf("clietn IPv4: %s, Port: %s\r\n", client_host, clietn_port);
Disassembly
0x00005555555553ad 55 push %rbp
0x00005555555553ae 48 83 3d 42 2c 00 00 00 cmpq $0x0,0x2c42(%rip)
0x00005555555553b6 48 89 e5 mov %rsp,%rbp
0x00005555555553b9 74 0c je 0x5555555553c7 <__do_g
0x00005555555553bb 48 8b 3d 46 2c 00 00 mov 0x2c46(%rip),%rdi
0x00005555555553c2 e8 c9 fd ff ff call 0x555555555190 <__cxa_
0x00005555555553c7 e8 64 ff ff ff call 0x555555555330 <deregi
0x00005555555553cc c6 05 55 2c 00 00 01 movb $0x1,0x2c55(%rip)
0x00005555555553d3 5d pop %rbp
0x00005555555553d4 c3 ret
0x00005555555553d5 0f 1f 00 nopl (%rax)
0x00005555555553d8 c3 ret
0x00005555555553d9 0f 1f 80 00 00 00 00 nopl 0x0(%rax)
0x00005555555553e0 f3 0f 1e fa endbr64
0x00005555555553e4 e9 77 ff ff ff jmp 0x555555555360 <regist
0x00005555555553e9 f3 0f 1e fa endbr64
0x00005555555553ed 55 push %rbp
0x00005555555553ee 48 89 e5 mov %rsp,%rbp
0x00005555555553f1 48 81 ec 30 01 00 00 sub $0x130,%rsp
0x00005555555553f8 64 48 8b 04 25 28 00 00 00 mov %fs:0x28,%rax
0x0000555555555401 48 89 45 f8 mov %rax,-0x8(%rbp)
0x0000555555555405 31 c0 xor %eax,%eax
0x0000555555555407 b8 00 00 00 00 mov $0x0,%eax
0x000055555555540c e8 89 03 00 00 call 0x55555555579a <open_l
0x0000555555555411 89 85 d0 fe ff ff mov %eax,-0x130(%rbp)
0x0000555555555417 83 bd d0 fe ff ff 00 cmpl $0x0,-0x130(%rbp)
0x000055555555541e 79 0a jns 0x55555555542a <main+6
0x0000555555555420 bf 01 00 00 00 mov $0x1,%edi
0x0000555555555425 e8 86 fe ff ff call 0x5555555552b0 <exit@p
0x000055555555542a 48 c7 85 d8 fe ff ff 10 00 00 00 movq $0x10,-0x1
0x0000555555555435 48 8d 95 d8 fe ff ff lea -0x128(%rbp),%rdx
0x000055555555543c 48 8d 8d e0 fe ff ff lea -0x120(%rbp),%rcx
0x0000555555555443 8b 85 d0 fe ff ff mov -0x130(%rbp),%eax
0x0000555555555449 48 89 ce mov %rcx,%rsi
0x000055555555544c 89 c7 mov %eax,%edi
0x000055555555544e e8 4d fe ff ff call 0x5555555552a0 <accept
0x0000555555555453 89 85 d4 fe ff ff mov %eax,-0x12c(%rbp)

Trace


RUN A...
C Debug
VARIABLES
Locals
peer_client_len = 1
client_host = [128]
clietn_port = [128]
listen_fd = 140
Registers
CPU
rax = 0x0
rbx = 0x7fffffffd628
rcx = 0x555555557d18
rdx = 0x7fffffffd638
rsi = 0x7fffffffd628
rdi = 0x1
rbp = 0x7fffffffd500
rsp = 0x7fffffffd3d0
r8 = 0x0
r9 = 0x7ffff7fca380
r10 = 0x7fffffffd220
r11 = 0x203
r12 = 0x1
r13 = 0x0
r14 = 0x555555557d18
r15 = 0x7ffff7ffd000
rip = 0x55555555540c
eflags = 0x246
eax = 0x0
ebx = 0xffffd628
ecx = 0x55557d18
edx = 0xffffd638
esi = 0xffffd628
WATCH
CALL STACK
Paused on step
main()
main.c 30:1
main.c
open_listen.c
main.c > main()
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include <netdb.h>
#include <sys/socket.h>
#include "lib/open_listen.h"
#include "lib/http_handler.h"
/**
* GDB: https://zhuanlan.zhihu.com/p/1919803617872938161
* Deassembly: https://huc0day.blog.csdn.net/article/details/151019122?spm=1001.2101.3001.
* distribute.pc_relevant.none-task-blog-2%7Edefault%7EYuanLiJiHua%7EPosition-
* 235%5Ev43%5Epc_blog_bottom_relevance_base1&depth_1-utm_source=distribute.pc
* 7Edefault%7EYuanLiJiHua%7EPosition-4-151019122-blog-86557911.235%5Ev43%5Epc
*
*/
int main()
{
int connected_fd;
struct sockaddr peer_client;
size_t peer_client_len;
char client_host[MAXLINE], clietn_port[MAXLINE];
int listen_fd = open_listenfd();
if(listen_fd < 0){
exit(1);
}
while(1){
peer_client_len = sizeof(peer_client);
connected_fd = accept(listen_fd, &peer_client, &peer_client_len);
getnameinfo(&peer_client, peer_client_len, client_host, MAXLINE, clietn_port, MAXL
printf("clietn IPv4: %s, Port: %s\r\n", client_host, clietn_port);
Disassembly
0x00005555555553ab 75 2b jne 0x5555555553d8 <__do_
0x00005555555553ad 55 push %rbp
0x00005555555553ae 48 83 3d 42 2c 00 00 00 cmpq $0x0,0x2c42(%rip)
0x00005555555553b6 48 89 e5 mov %rsp,%rbp
0x00005555555553b9 74 0c je 0x5555555553c7 <__do_
0x00005555555553bb 48 8b 3d 46 2c 00 00 mov 0x2c46(%rip),%rdi
0x00005555555553c2 e8 c9 fd ff ff call 0x555555555190 <__cxa
0x00005555555553c7 e8 64 ff ff ff call 0x555555555330 <dereg
0x00005555555553cc c6 05 55 2c 00 00 01 movb $0x1,0x2c55(%rip)
0x00005555555553d3 5d pop %rbp
0x00005555555553d4 c3 ret
0x00005555555553d5 0f 1f 00 nopl (%rax)
0x00005555555553d8 c3 ret
0x00005555555553d9 0f 1f 80 00 00 00 00 nopl 0x0(%rax)
0x00005555555553e0 f3 0f 1e fa endbr64
0x00005555555553e4 e9 77 ff ff ff jmp 0x555555555360 <regis
0x00005555555553e9 f3 0f 1e fa endbr64
0x00005555555553ed 55 push %rbp
0x00005555555553ee 48 89 e5 mov %rsp,%rbp
0x00005555555553f1 48 81 ec 30 01 00 00 sub $0x130,%rsp
0x00005555555553f8 64 48 8b 04 25 28 00 00 00 mov %fs:0x28,%rax
0x0000555555555401 48 89 45 f8 mov %rax,-0x8(%rbp)
0x0000555555555405 31 c0 xor %eax,%eax
0x0000555555555407 b8 00 00 00 00 mov $0x0,%eax
0x000055555555540c e8 89 03 00 00 call 0x55555555579a <open_
0x0000555555555411 89 85 d0 fe ff ff mov %eax,-0x130(%rbp)
0x0000555555555417 83 bd d0 fe ff ff 00 cmpl $0x0,-0x130(%rbp)
0x000055555555541e 79 0a jns 0x55555555542a <main+
0x0000555555555420 bf 01 00 00 00 mov $0x1,%edi
0x0000555555555425 e8 86 fe ff ff call 0x5555555552b0 <exit@
0x000055555555542a 48 c7 85 d8 fe ff ff 10 00 00 00 movq $0x10,-0x
0x0000555555555435 48 8d 95 d8 fe ff ff lea -0x128(%rbp),%rdx
0x000055555555543c 48 8d 8d e0 fe ff ff lea -0x120(%rbp),%rcx
0x0000555555555443 8b 85 d0 fe ff ff mov -0x130(%rbp),%eax
0x0000555555555449 48 89 ce mov %rcx,%rsi
0x000055555555544c 89 c7 mov %eax,%edi
0x000055555555544e e8 4d fe ff ff call 0x5555555552a0 <accep
0x0000555555555453 89 85 d4 fe ff ff mov %eax,-0x12c(%rbp)
0x0000555555555459 48 8b 85 d8 fe ff ff mov -0x128(%rbp),%rax

Trace


VARIABLES
Locals
Registers
CPU
rax = 0x0
rbx = 0x7fffffffd628
rcx = 0x555555557d18
rdx = 0x7fffffffd638
rsi = 0x7fffffffd628
rdi = 0x1
rbp = 0x7fffffffd500
rsp = 0x7fffffffd3c8
r8 = 0x0
r9 = 0x7ffff7fca380
r10 = 0x7fffffffd220
r11 = 0x203
r12 = 0x1
r13 = 0x0
r14 = 0x555555557d18
r15 = 0x7ffff7ffd000
rip = 0x55555555579a
eflags = 0x246
eax = 0x0
ebx = 0xffffd628
ecx = 0x55557d18
edx = 0xffffd638
esi = 0xffffd628
edi = 0x1
ebp = 0xffffd500
esp = 0xffffd3c8
WATCH
CALL STACK
Paused on step
open_listenfd() open_listen.c
main() main.c 30:1
lib > C open_listen.c > open_listenfd()
#include "open_listen.h"
#include <netdb.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <string.h>
int open_listenfd()
{
struct addrinfo hints;
struct addrinfo *list_rp, *rp;
char host_buf[HOST_MAXLEN];
char service_buf[SERVICE_MAXLEN];
int socket_fd;
int result;
memset(&hints, 0, sizeof(hints));
hints.ai_family = AF_INET; // IPv4
hints.ai_flags = AI_PASSIVE; // Server
hints.ai_socktype = SOCK_STREAM ; // TCP
result = getaddrinfo(NULL, LISTEN_PORT, &hints, &list_rp);
if(result != 0){
fprintf(stderr, "[ERROR]-getaddrinfo error: %s\r\n", gai_strerror(result));
return -1;
}
/**
* getaddrinfo() returns a list of address structures.
* Try each address until we successfully bind().
* If socket() or bind() fails, we close the socket and try the next address.
*/
for(rp = list_rp; rp != NULL ; rp = rp->ai_next){
getnameinfo(rp->ai_addr, rp->ai_addrlen, host_buf, HOST_MAXLEN, service_buf, SERVI
printf("IPv4: %s, Port: %s\r\n", host_buf, service_buf);
socket_fd = socket(rp->ai_family, rp->ai_socktype, rp->ai_protocol);
if(socket_fd < 0){
0x0000555555555779 48 89 ce mov %rcx,%rsi
0x000055555555577c 89 c7 mov %eax,%edi
0x000055555555577e e8 58 fd ff ff call 0x5555555554db <respor
0x0000555555555783 90 nop
0x0000555555555784 48 8b 45 f8 mov -0x8(%rbp),%rax
0x0000555555555788 64 48 2b 04 25 28 00 00 00 sub %fs:0x28,%rax
0x0000555555555791 74 05 je 0x555555555798 <reques
0x0000555555555793 e8 38 fa ff ff call 0x5555555551d0 <__stac
0x0000555555555798 c9 leave
0x0000555555555799 c3 ret
0x000055555555579a f3 0f 1e fa endbr64
0x000055555555579e 55 push %rbp
0x000055555555579f 48 89 e5 mov %rsp,%rbp
0x00005555555557a2 48 81 ec d0 00 00 00 sub $0xd0,%rsp
0x00005555555557a9 64 48 8b 04 25 28 00 00 00 mov %fs:0x28,%rax
0x00005555555557b2 48 89 45 f8 mov %rax,-0x8(%rbp)
0x00005555555557b6 31 c0 xor %eax,%eax
0x00005555555557b8 48 8d 85 50 ff ff ff lea -0xb0(%rbp),%rax
0x00005555555557bf ba 30 00 00 00 mov $0x30,%edx
0x00005555555557c4 be 00 00 00 00 mov $0x0,%esi
0x00005555555557c9 48 89 c7 mov %rax,%rdi
0x00005555555557cc e8 3f fa ff ff call 0x555555555210 <memset
0x00005555555557d1 c7 85 54 ff ff ff 02 00 00 00 movl $0x2,-0xac(%rbp)
0x00005555555557db c7 85 50 ff ff ff 01 00 00 00 movl $0x1,-0xb0(%rbp)
0x00005555555557e5 c7 85 58 ff ff ff 01 00 00 00 movl $0x1,-0xa8(%rbp)
0x00005555555557ef 48 8d 95 40 ff ff ff lea -0xc0(%rbp),%rdx
0x00005555555557f6 48 8d 85 50 ff ff ff lea -0xb0(%rbp),%rax
0x00005555555557fd 48 89 d1 mov %rdx,%rcx
0x0000555555555800 48 89 c2 mov %rax,%rdx
0x0000555555555803 48 8d 05 b6 08 00 00 lea 0x8b6(%rip),%rax
0x000055555555580a 48 89 c6 mov %rax,%rsi
0x000055555555580d bf 00 00 00 00 mov $0x0,%edi
0x0000555555555812 e8 b9 fa ff ff call 0x5555555552d0 <getadd
0x0000555555555817 89 85 3c ff ff ff mov %eax,-0xc4(%rbp)
0x000055555555581d 83 bd 3c ff ff ff 00 cmpl $0x0,-0xc4(%rbp)
0x0000555555555824 74 38 je 0x55555555585e <open_
0x0000555555555826 8b 85 3c ff ff ff mov -0xc4(%rbp),%eax
0x000055555555582c 89 c7 mov %eax,%edi
0x000055555555582e e8 cd f9 ff ff call 0x555555555200 <gai_st
0x0000555555555833 48 89 c2 mov %rax,%rdx
0x0000555555555836 48 8b 05 e3 27 00 00 mov 0x27e3(%rip),%rax

Trace

首先來看 open_listenfd function

179e → 將 %rbp 內容儲存到 stack 中 (push 會將 %rsp - 0x08 再將 %rbp 內容儲存到 stack)

179f → 將 %rsp 的值複製到 %rbp 中

17a2→ 將 %rsp 減去 0xd0，用於分配 stack space 給 open_listenfd function

17a9 → 將 Canary value 複製到 %rax 中

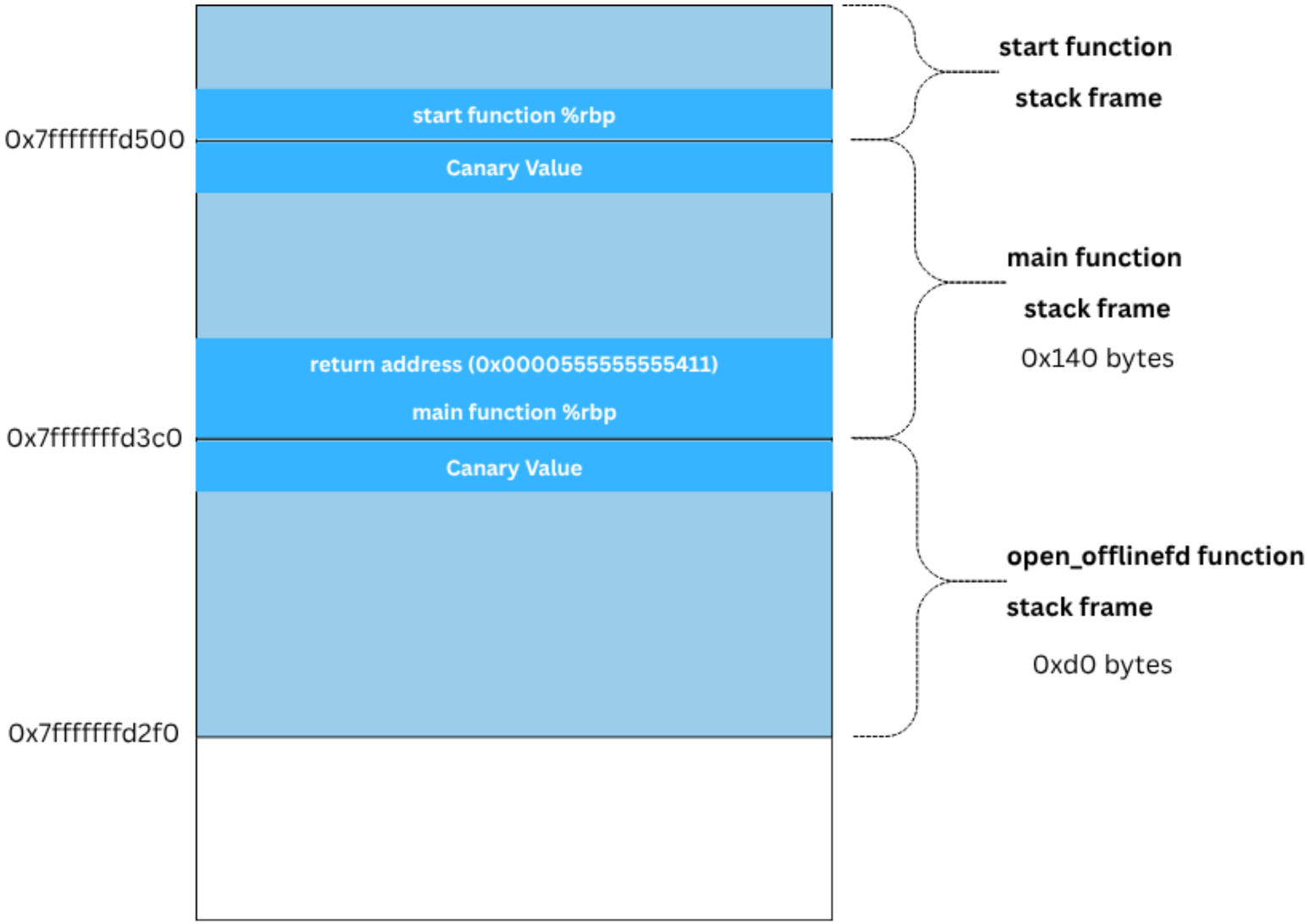
17b0 → 將 %rax 值複製到 %rbp-0x8 的位置中

17b2 → 將 %rax 清除為 0

```
000000000000179a <open_listenfd>:
#include <sys/types.h>
#include <sys/socket.h>
#include <string.h>

int open_listenfd()
{
    179a:       f3 0f 1e fa             endbr64
    179e:       55                      push   %rbp
    179f:       48 89 e5                mov    %rsp,%rbp
    17a2:       48 81 ec d0 00 00 00    sub    $0xd0,%rsp
    17a9:       64 48 8b 04 25 28 00    mov    %fs:0x28,%rax
    17b0:       00 00
    17b2:       48 89 45 f8             mov    %rax,-0x8(%rbp)
    17b6:       31 c0                   xor    %eax,%eax
```

Trace


start function %rbp
0x7fffffffd500
Canary Value
return address (0x0000555555555411)
main function %rbp
0x7fffffffd3c0
Canary Value
0x7fffffffd2f0
start function
stack frame
main function
stack frame
0x140 bytes
open_offlinefd function
stack frame
0xd0 bytes

Trace

再來看 open_listenfd function 呼叫 memset() function 是如何傳遞參數的

17ba→ 將 %rbp 減去 0xb0，並複製到 %rax

17bf → 將 0x30 複製到 %edx 中

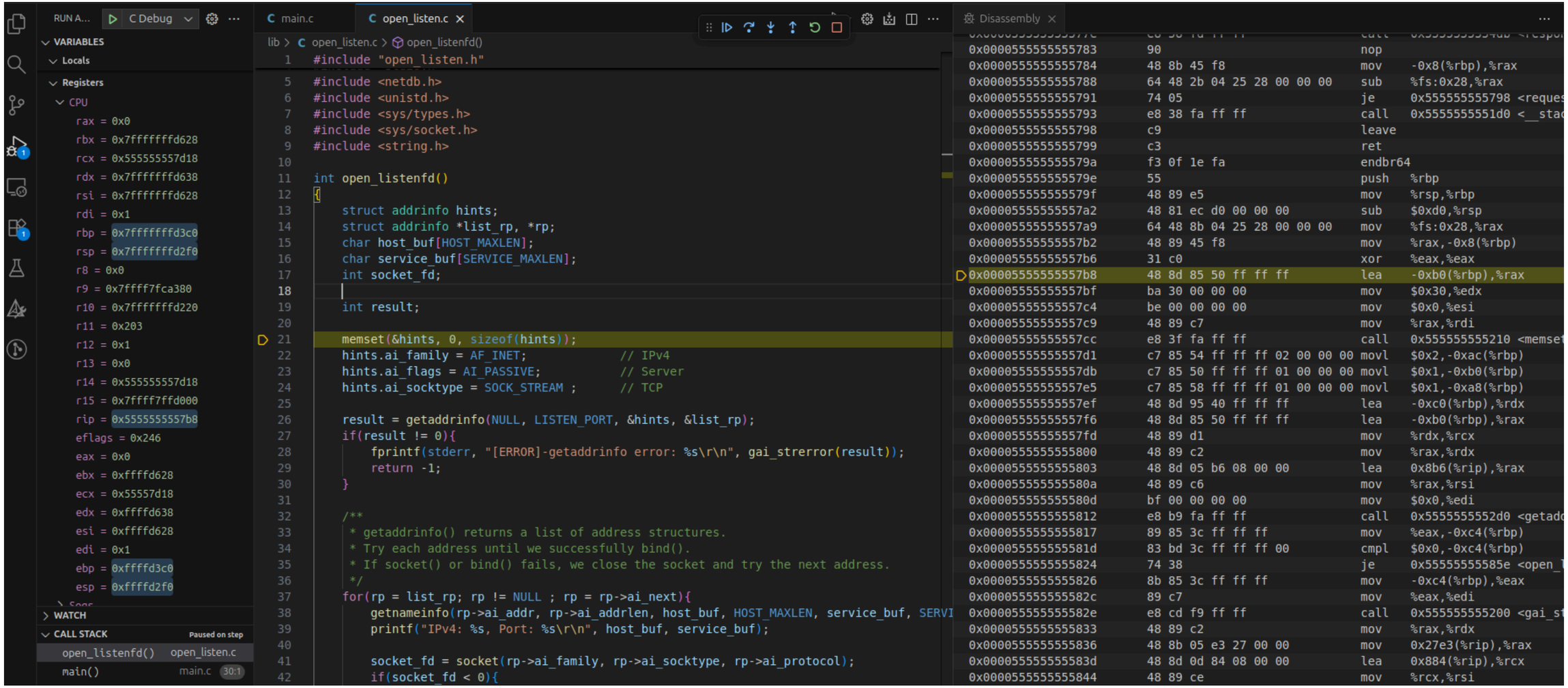
17c4 → 將 0x0 值複製到 %esi

17c9 → 將 %rax 的值複製到 %rdi

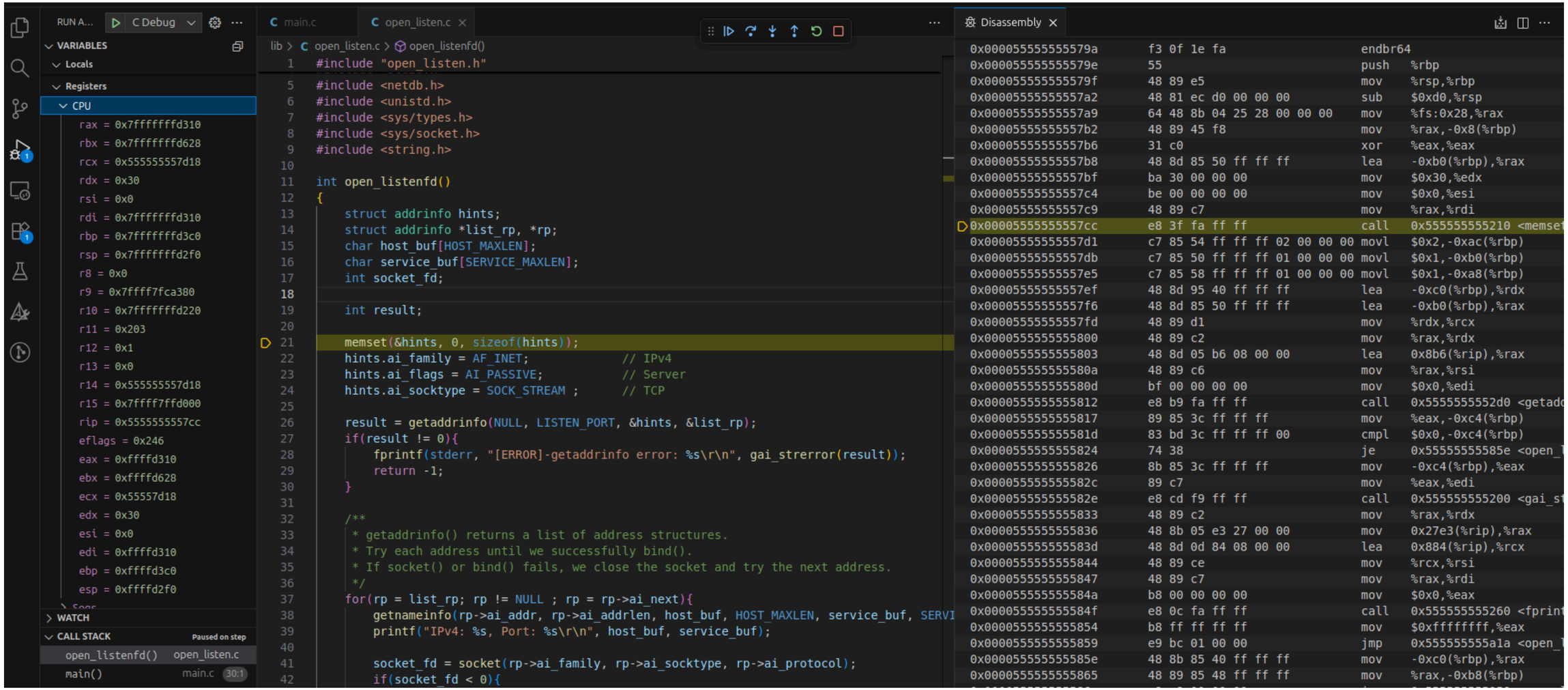
17cc → 呼叫 function memset() ，並將下一個 instruction 的 address 壓入到 stack (%rsp -= 8)，並設置 %rip 為 memset() 的位置。

```
memset(&hints, 0, sizeof(hints));
17b8:       48 8d 85 50 ff ff ff    lea    -0xb0(%rbp),%rax
17bf:       ba 30 00 00 00          mov    $0x30,%edx
17c4:       be 00 00 00 00          mov    $0x0,%esi
17c9:       48 89 c7                mov    %rax,%rdi
17cc:       e8 3f fa ff ff          call   1210 <memset@plt>
```

Trace


RUN A...
C Debug
VARIABLES
Locals
Registers
CPU
rax = 0x0
rbx = 0x7fffffffd628
rcx = 0x555555557d18
rdx = 0x7fffffffd638
rsi = 0x7fffffffd628
rdi = 0x1
rbp = 0x7fffffffd3c0
rsp = 0x7fffffffd2f0
r8 = 0x0
r9 = 0x7ffff7fca380
r10 = 0x7fffffffd220
r11 = 0x203
r12 = 0x1
r13 = 0x0
r14 = 0x555555557d18
r15 = 0x7ffff7ffd000
rip = 0x5555555557b8
eflags = 0x246
eax = 0x0
ebx = 0xffffd628
ecx = 0x55557d18
edx = 0xffffd638
esi = 0xffffd628
edi = 0x1
ebp = 0xffffd3c0
esp = 0xffffd2f0
WATCH
CALL STACK
Paused on step
open_listenfd() open_listen.c
main() main.c 30:1
main.c
open_listen.c
lib > open_listen.c > open_listenfd()
#include "open_listen.h"
#include <netdb.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <string.h>
int open_listenfd()
{
struct addrinfo hints;
struct addrinfo *list_rp, *rp;
char host_buf[HOST_MAXLEN];
char service_buf[SERVICE_MAXLEN];
int socket_fd;
int result;
memset(&hints, 0, sizeof(hints));
hints.ai_family = AF_INET; // IPv4
hints.ai_flags = AI_PASSIVE; // Server
hints.ai_socktype = SOCK_STREAM ; // TCP
result = getaddrinfo(NULL, LISTEN_PORT, &hints, &list_rp);
if(result != 0){
fprintf(stderr, "[ERROR]-getaddrinfo error: %s\r\n", gai_strerror(result));
return -1;
}
/**
* getaddrinfo() returns a list of address structures.
* Try each address until we successfully bind().
* If socket() or bind() fails, we close the socket and try the next address.
*/
for(rp = list_rp; rp != NULL ; rp = rp->ai_next){
getnameinfo(rp->ai_addr, rp->ai_addrlen, host_buf, HOST_MAXLEN, service_buf, SERVI
printf("IPv4: %s, Port: %s\r\n", host_buf, service_buf);
socket_fd = socket(rp->ai_family, rp->ai_socktype, rp->ai_protocol);
if(socket_fd < 0){
Disassembly
0x0000555555555783 90 nop
0x0000555555555784 48 8b 45 f8 mov -0x8(%rbp),%rax
0x0000555555555788 64 48 2b 04 25 28 00 00 00 sub %fs:0x28,%rax
0x0000555555555791 74 05 je 0x555555555798
0x0000555555555793 e8 38 fa ff ff call 0x5555555551d0
0x0000555555555798 c9 leave
0x0000555555555799 c3 ret
0x000055555555579a f3 0f 1e fa endbr64
0x000055555555579e 55 push %rbp
0x000055555555579f 48 89 e5 mov %rsp,%rbp
0x00005555555557a2 48 81 ec d0 00 00 00 sub $0xd0,%rsp
0x00005555555557a9 64 48 8b 04 25 28 00 00 00 mov %fs:0x28,%rax
0x00005555555557b2 48 89 45 f8 mov %rax,-0x8(%rbp)
0x00005555555557b6 31 c0 xor %eax,%eax
0x00005555555557b8 48 8d 85 50 ff ff ff lea -0xb0(%rbp),%rax
0x00005555555557bf ba 30 00 00 00 mov $0x30,%edx
0x00005555555557c4 be 00 00 00 00 mov $0x0,%esi
0x00005555555557c9 48 89 c7 mov %rax,%rdi
0x00005555555557cc e8 3f fa ff ff call 0x555555555210
0x00005555555557d1 c7 85 54 ff ff ff 02 00 00 00 movl $0x2,-0xac(%rbp)
0x00005555555557db c7 85 50 ff ff ff 01 00 00 00 movl $0x1,-0xb0(%rbp)
0x00005555555557e5 c7 85 58 ff ff ff 01 00 00 00 movl $0x1,-0xa8(%rbp)
0x00005555555557ef 48 8d 95 40 ff ff ff lea -0xc0(%rbp),%rdx
0x00005555555557f6 48 8d 85 50 ff ff ff lea -0xb0(%rbp),%rax
0x00005555555557fd 48 89 d1 mov %rdx,%rcx
0x0000555555555800 48 89 c2 mov %rax,%rdx
0x0000555555555803 48 8d 05 b6 08 00 00 lea 0x8b6(%rip),%rax
0x000055555555580a 48 89 c6 mov %rax,%rsi
0x000055555555580d bf 00 00 00 00 mov $0x0,%edi
0x0000555555555812 e8 b9 fa ff ff call 0x5555555552d0
0x0000555555555817 89 85 3c ff ff ff mov %eax,-0xc4(%rbp)
0x000055555555581d 83 bd 3c ff ff ff 00 cmpl $0x0,-0xc4(%rbp)
0x0000555555555824 74 38 je 0x55555555585e
0x0000555555555826 8b 85 3c ff ff ff mov -0xc4(%rbp),%eax
0x000055555555582c 89 c7 mov %eax,%edi
0x000055555555582e e8 cd f9 ff ff call 0x555555555200
0x0000555555555833 48 89 c2 mov %rax,%rdx
0x0000555555555836 48 8b 05 e3 27 00 00 mov 0x27e3(%rip),%rax
0x000055555555583d 48 8d 0d 84 08 00 00 lea 0x884(%rip),%rcx
0x0000555555555844 48 89 ce mov %rcx,%rsi

Trace


RUN A...
C Debug
VARIABLES
Locals
Registers
CPU
rax = 0x7fffffffd310
rbx = 0x7fffffffd628
rcx = 0x555555557d18
rdx = 0x30
rsi = 0x0
rdi = 0x7fffffffd310
rbp = 0x7fffffffd3c0
rsp = 0x7fffffffd2f0
r8 = 0x0
r9 = 0x7ffff7fca380
r10 = 0x7fffffffd220
r11 = 0x203
r12 = 0x1
r13 = 0x0
r14 = 0x555555557d18
r15 = 0x7ffff7ffd000
rip = 0x5555555557cc
eflags = 0x246
eax = 0xffffd310
ebx = 0xffffd628
ecx = 0x55557d18
edx = 0x30
esi = 0x0
edi = 0xffffd310
ebp = 0xffffd3c0
esp = 0xffffd2f0
WATCH
CALL STACK
Paused on step
open_listenfd() open_listen.c
main() main.c 30:1
main.c
open_listen.c
lib > open_listen.c > open_listenfd()
#include "open_listen.h"
#include <netdb.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <string.h>
int open_listenfd()
{
struct addrinfo hints;
struct addrinfo *list_rp, *rp;
char host_buf[HOST_MAXLEN];
char service_buf[SERVICE_MAXLEN];
int socket_fd;
int result;
memset(&hints, 0, sizeof(hints));
hints.ai_family = AF_INET; // IPv4
hints.ai_flags = AI_PASSIVE; // Server
hints.ai_socktype = SOCK_STREAM ; // TCP
result = getaddrinfo(NULL, LISTEN_PORT, &hints, &list_rp);
if(result != 0){
fprintf(stderr, "[ERROR]-getaddrinfo error: %s\r\n", gai_strerror(result));
return -1;
}
/**
* getaddrinfo() returns a list of address structures.
* Try each address until we successfully bind().
* If socket() or bind() fails, we close the socket and try the next address.
*/
for(rp = list_rp; rp != NULL ; rp = rp->ai_next){
getnameinfo(rp->ai_addr, rp->ai_addrlen, host_buf, HOST_MAXLEN, service_buf, SERVI
printf("IPv4: %s, Port: %s\r\n", host_buf, service_buf);
socket_fd = socket(rp->ai_family, rp->ai_socktype, rp->ai_protocol);
if(socket_fd < 0){
Disassembly
0x000055555555579a f3 0f 1e fa endbr64
0x000055555555579e 55 push %rbp
0x000055555555579f 48 89 e5 mov %rsp,%rbp
0x00005555555557a2 48 81 ec d0 00 00 00 sub $0xd0,%rsp
0x00005555555557a9 64 48 8b 04 25 28 00 00 00 mov %fs:0x28,%rax
0x00005555555557b2 48 89 45 f8 mov %rax,-0x8(%rbp)
0x00005555555557b6 31 c0 xor %eax,%eax
0x00005555555557b8 48 8d 85 50 ff ff ff lea -0xb0(%rbp),%rax
0x00005555555557bf ba 30 00 00 00 mov $0x30,%edx
0x00005555555557c4 be 00 00 00 00 mov $0x0,%esi
0x00005555555557c9 48 89 c7 mov %rax,%rdi
0x00005555555557cc e8 3f fa ff ff call 0x555555555210 <memset
0x00005555555557d1 c7 85 54 ff ff ff 02 00 00 00 movl $0x2,-0xac(%rbp)
0x00005555555557db c7 85 50 ff ff ff 01 00 00 00 movl $0x1,-0xb0(%rbp)
0x00005555555557e5 c7 85 58 ff ff ff 01 00 00 00 movl $0x1,-0xa8(%rbp)
0x00005555555557ef 48 8d 95 40 ff ff ff lea -0xc0(%rbp),%rdx
0x00005555555557f6 48 8d 85 50 ff ff ff lea -0xb0(%rbp),%rax
0x00005555555557fd 48 89 d1 mov %rdx,%rcx
0x0000555555555800 48 89 c2 mov %rax,%rdx
0x0000555555555803 48 8d 05 b6 08 00 00 lea 0x8b6(%rip),%rax
0x000055555555580a 48 89 c6 mov %rax,%rsi
0x000055555555580d bf 00 00 00 00 mov $0x0,%edi
0x0000555555555812 e8 b9 fa ff ff call 0x5555555552d0 <getadd
0x0000555555555817 89 85 3c ff ff ff mov %eax,-0xc4(%rbp)
0x000055555555581d 83 bd 3c ff ff ff 00 cmpl $0x0,-0xc4(%rbp)
0x0000555555555824 74 38 je 0x55555555585e <open_
0x0000555555555826 8b 85 3c ff ff ff mov -0xc4(%rbp),%eax
0x000055555555582c 89 c7 mov %eax,%edi
0x000055555555582e e8 cd f9 ff ff call 0x555555555200 <gai_st
0x0000555555555833 48 89 c2 mov %rax,%rdx
0x0000555555555836 48 8b 05 e3 27 00 00 mov 0x27e3(%rip),%rax
0x000055555555583d 48 8d 0d 84 08 00 00 lea 0x884(%rip),%rcx
0x0000555555555844 48 89 ce mov %rcx,%rsi
0x0000555555555847 48 89 c7 mov %rax,%rdi
0x000055555555584a b8 00 00 00 00 mov $0x0,%eax
0x000055555555584f e8 0c fa ff ff call 0x555555555260 <fprint
0x0000555555555854 b8 ff ff ff ff mov $0xffffffff,%eax
0x0000555555555859 e9 bc 01 00 00 jmp 0x555555555a1a <open_
0x000055555555585e 48 8b 85 40 ff ff ff mov -0xc0(%rbp),%rax
0x0000555555555865 48 89 85 48 ff ff ff mov %rax,-0xb8(%rbp)

Thank you