NETWORK & MULTIMEDIA LAB

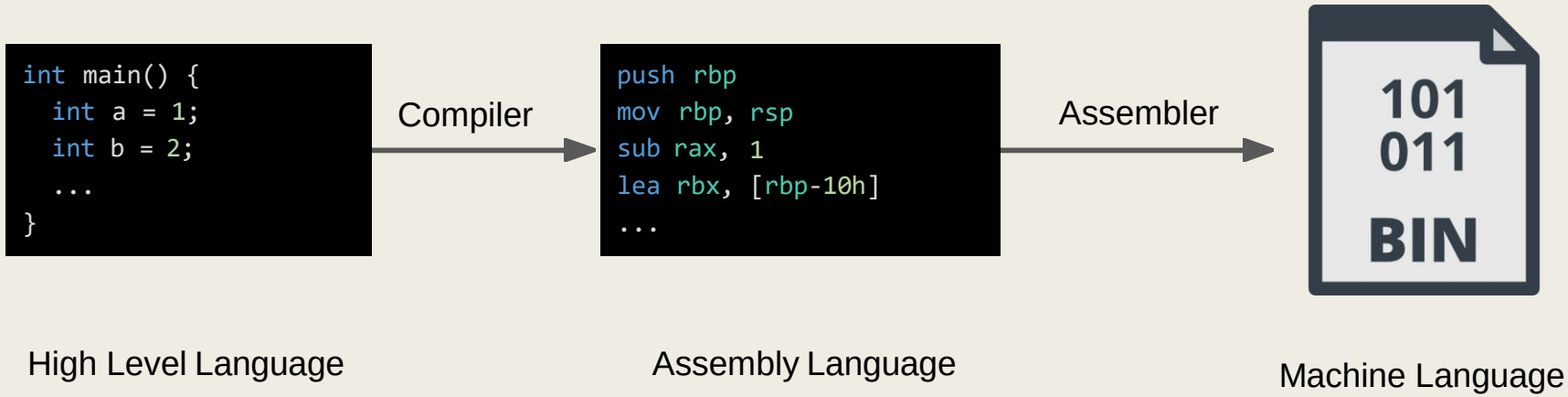
REVERSE ENGINEERING

Fall 2020

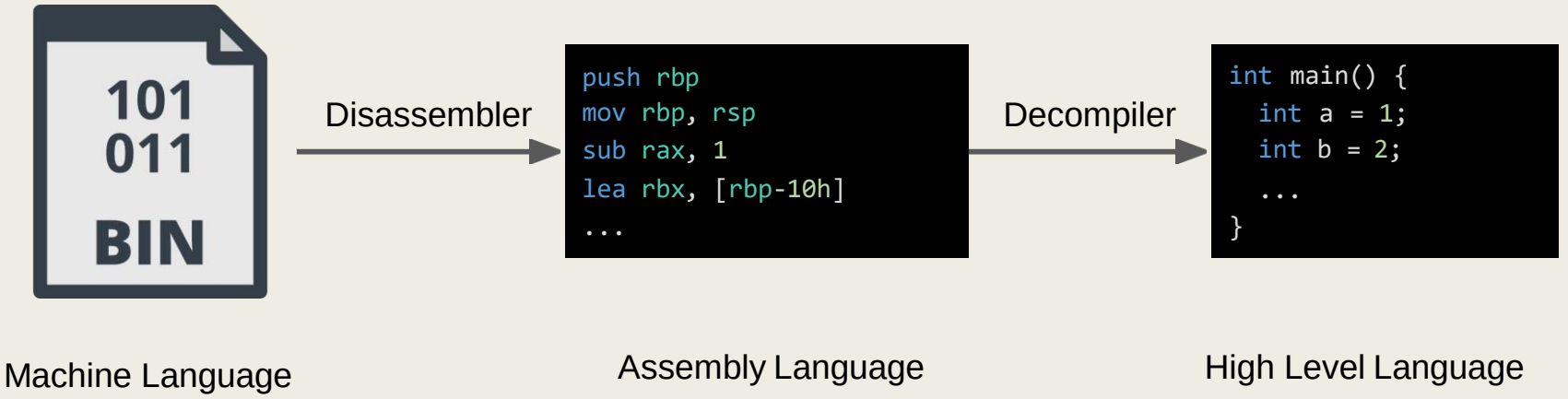
Reverse engineering

- (software) reverse engineering:
  - *The process of analyzing a subject system to identify the system's components and their interrelationships* and to create representations of the system in another form or at a higher level of abstraction.

Compiler, Assembler



Decompiler, Disassembler



Compile C Code

- example.c

```c
#include "stdio.h"

int main(){
    printf("Hello, world!");
    return 0;
}
```

- gcc -masm=intel -S example.c
  - *This will generate example.s*
- gcc -s example.s -o example.o

```
    .file   "example.c"
    .intel_syntax noprefix
    .section    .rodata
.LC0:
    .string "Hello, world!"
    .text
    .globl  main
    .type   main, @function
main:
.LFB0:
    .cfi_startproc
    push    rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    mov rbp, rsp
    .cfi_def_cfa_register 6
    mov edi, OFFSET FLAT:.LC0
    mov eax, 0
    call    printf
    mov eax, 0
    pop rbp
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
.LFE0:
    .size   main, .-main
    .ident  "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.12) 5.4.0 20160609"
    .section    .note.GNU-stack,"",@progbits
```

x86-64 Assembly

- Syntax
  - *AT&T*
  - *intel*

```
Ltmp2:
        .cfi_def_cfa_register %rbp
        movslq  %edi, %rax
        imulq   $1759218605, %rax,
        movq    %rsi, %rax
        shrq    $63, %rax
        sarq    $44, %rsi
        addl    %eax, %esi
        leaq    L_.str(%rip), %rdi
        xorl    %eax, %eax
        callq   _printf
        xorl    %eax, %eax
        popq    %rbp
        retq
        .cfi_endproc
```

```
Ltmp2:
        .cfi_def_cfa_register rbp
        movsxd  rax, edi
        imul    rsi, rax, 175921860
        mov     rax, rsi
        shr     rax, 63
        sar     rsi, 44
        add     esi, eax
        lea     rdi, [rip + L_.str]
        xor     eax, eax
        call    _printf
        xor     eax, eax
```

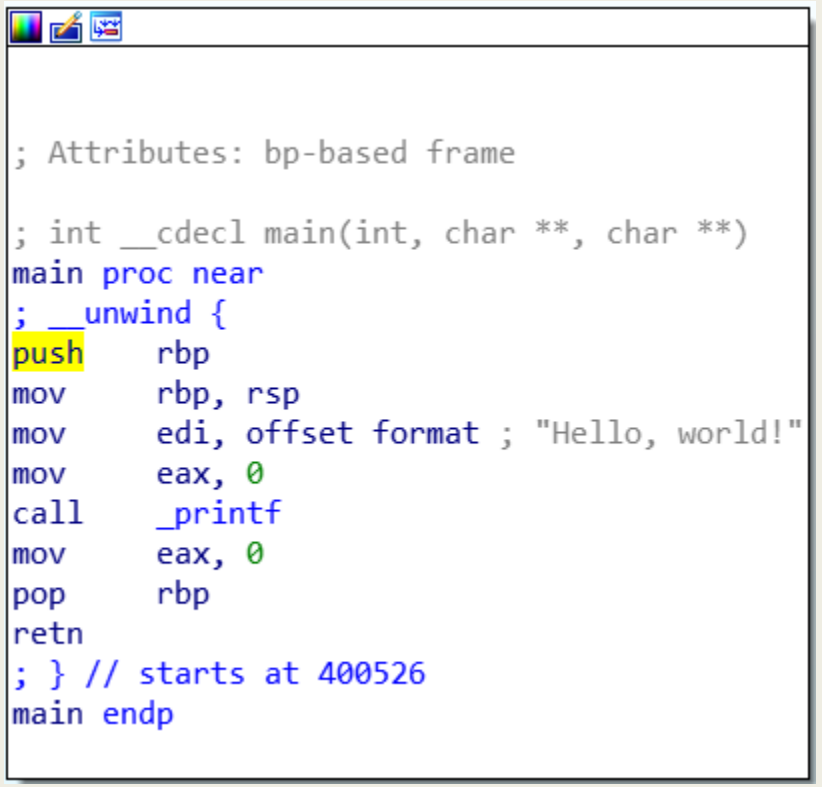
Disassemble in Terminal

- objdump
  - *Mac: objdump -disassemble -x86-asm-syntax=intel <file>*
  - *Linux: objdump -d -M intel <file>*

```
Disassembly of section .text:

0000000000400430 <.text>:
  400430:       31 ed                   xor    ebp,ebp
  400432:       49 89 d1                mov    r9,rdx
  400435:       5e                      pop    rsi
  400436:       48 89 e2                mov    rdx,rsp
  400439:       48 83 e4 f0             and    rsp,0xfffffffffffffff0
  40043d:       50                      push   rax
  40043e:       54                      push   rsp
  40043f:       49 c7 c0 b0 05 40 00    mov    r8,0x4005b0
  400446:       48 c7 c1 40 05 40 00    mov    rcx,0x400540
  40044d:       48 c7 c7 26 05 40 00    mov    rdi,0x400526
  400454:       e8 b7 ff ff ff          call   400410 <__libc_start_main@plt>
  400459:       f4                      hlt
  40045a:       66 0f 1f 44 00 00       nop    WORD PTR [rax+rax*1+0x0]
  400460:       b8 3f 10 60 00          mov    eax,0x60103f
  400465:       55                      push   rbp
```

Disassemble Using IDA Pro

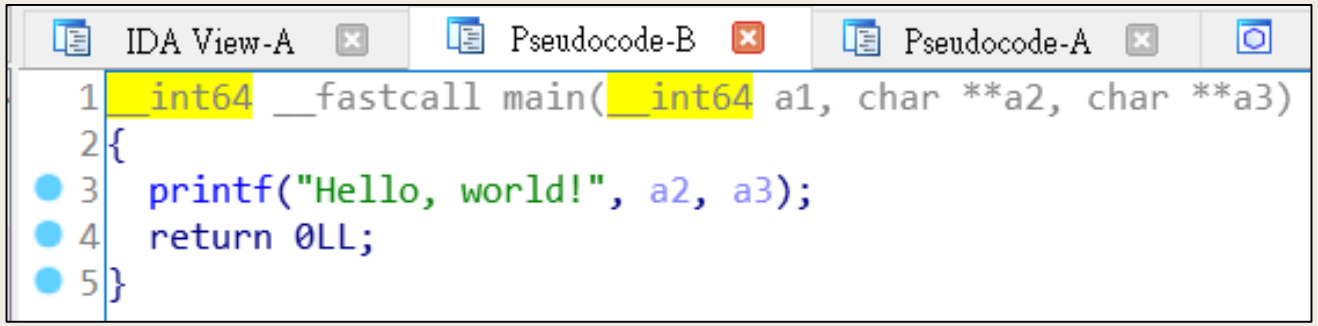
- Open IDA Pro
- Choose the example.o file
- Navigate to main function



```
; Attributes: bp-based frame

; int __cdecl main(int, char **, char **)
main proc near
; __unwind {
push    rbp
mov     rbp, rsp
mov     edi, offset format ; "Hello, world!"
mov     eax, 0
call    _printf
mov     eax, 0
pop     rbp
retn
; } // starts at 400526
main endp
```

Decompile Using IDA Pro

- Go to the function you want to decompile
- Press F5



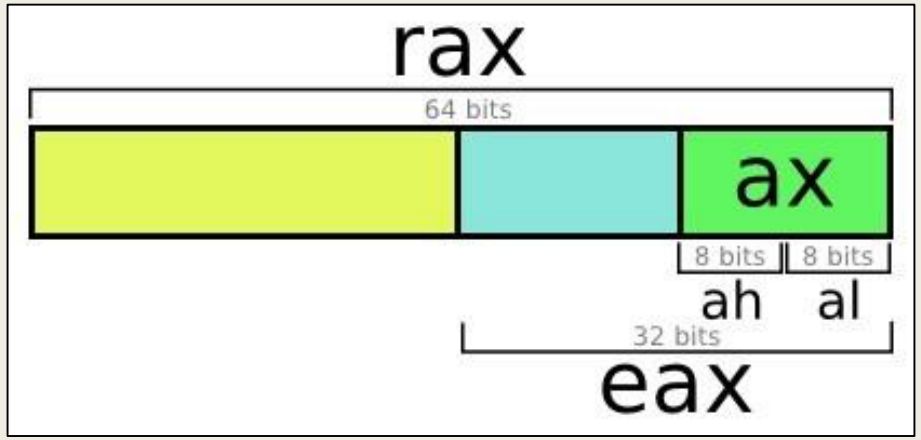
Assembly

- Assembly language
  - *low-level programming language*
  - *instruction set architecture*
    - x86, x86-64
    - ARM32, ARM64
    - MIPS
    - …
- x86-64 (also known as x64, AMD64 and Intel 64)

x86-64 Assembly

- Register
  - 暫存器（Register）是 CPU 內用來暫存指令、數據和位址的電腦記憶體

- Register layout
  - *r[a-d]x*
  - *rsi, rdi*
  - *rbp, rsp*
  - *r[8-15]*
  - *...*



https://wiki.cdot.senecacollege.ca/wiki/X86_64_Register_and_Instruction_Quick_Start

x86-64 Assembly

- Instruction
  - *mov dst, src*
  - *example*
    - mov rax, rbx // rax = rbx
    - mov rax, [rbp - 4] // rax = *(rbp - 4)
    - mov [rax], rbx // *rax = rbx

x86-64 Assembly

- Instruction
  - *add, sub, imul, idiv, and, or, xor dst, src*
  - *example*
    - sub rbx, [rbp - 4] // rbx = rbx - *(rbp - 4)
    - mul rcx, 2 // rcx = rcx * 2
    - xor [rsp], rax // *rsp = (*rsp) ^ rax

x86-64 Assembly

- Instruction
  - *Inc, dec, neg, not dst*
  - *example*
    - dec rbx // rbx -= 1
    - neg rcx // rcx = -rcx
    - not byte [rsp] // convert [rsp] to byte *rsp = ~(*rsp)

x86-64 Assembly

- Instruction
  - *specify the type*
    - if dst is memory address and src is not register
    - need to specify the type of dst
    - byte ⟶ word ⟶ dword ⟶ qword
    - (8 bits ⟶ 2 bytes ⟶ 2 words ⟶ 2 dwords)
    - (8 bits ⟶ 16 bits ⟶ 32 bits ⟶ 64 bits)

x86-64 Assembly

- Instruction
  - *specify the type*
  - *example*
    - mov byte [rax], 0xda
    - mov word [rax], 0xda
    - mov dword [rax], 0xda
    - mov qword [rax], 0xda

x86-64 Assembly

- Instruction
  - *cmp val1, val2*
  - *example*
    - cmp rax, 5 // compare the values and set the flags (zero, carry, sign …)
    - cmp rbx, rcx
    - cmp word [rsp], 0x1234

x86-64 Assembly

- Instruction
  - *jmp label*
  - *example*
    - loop: // set a label
    - ; do something
    - jmp loop // jump to loop label

x86-64 Assembly

- Instruction
  - *ja, jb, jna, jbe, je, jne, jz label*
  - *example*
    - cmp rax, 10
    - je quit

x86-64 Assembly

- Translate between assembly and C/C++
  - *conditional statement*

```c
if (rax < 5) {
    foo1();
}
else if (rax >= 5 && rax < 10) {
    foo2();
}
else {
    foo3();
}
```

```asm
    cmp     rax, 0x4
    jg      Lelseif
    call    foo1
    jmp     Lend
Lelseif:
    cmp     rax, 0x4
    jle     Lelse
    cmp     rax, 0x9
    jg      Lelse
    call    foo2
    jmp     Lend
Lelse:
    call    foo3
Lend:
    ret
```

x86-64 Assembly

- Translate between assembly and C/C++
  - *loop statement*

```c
for (rax = 0; rax < 10; rax++) {
    foo1();
}
```

```asm
mov     rax, 0x0
jmp     Lchk
Lbody:
call    foo1
add     rax, 0x1
Lchk:
cmp     rax, 0x9
jle     Lbody
ret
```

x86-64 Assembly

- Instruction
  - *nop*
  - *example*
    - nop

```
// no operation, do nothing
// this is useful when patching the binary
```

x86-64 Assembly

- Instruction
  - *syscall*
  - *example*

%rax	System call	%rdi	%rsi	%rdx
0	sys_read	unsigned int fd	char *buf	size_t count
1	sys_write	unsigned int fd	const char *buf	size_t count
2	sys_open	const char *filename	int flags	int mode

http://blog.rchapman.org/posts/Linux_System_Call_Table_for_x86_64/

x86-64 calling conventions
呼叫慣例

- System V AMD64 ABI (**Application Binary Interface**)

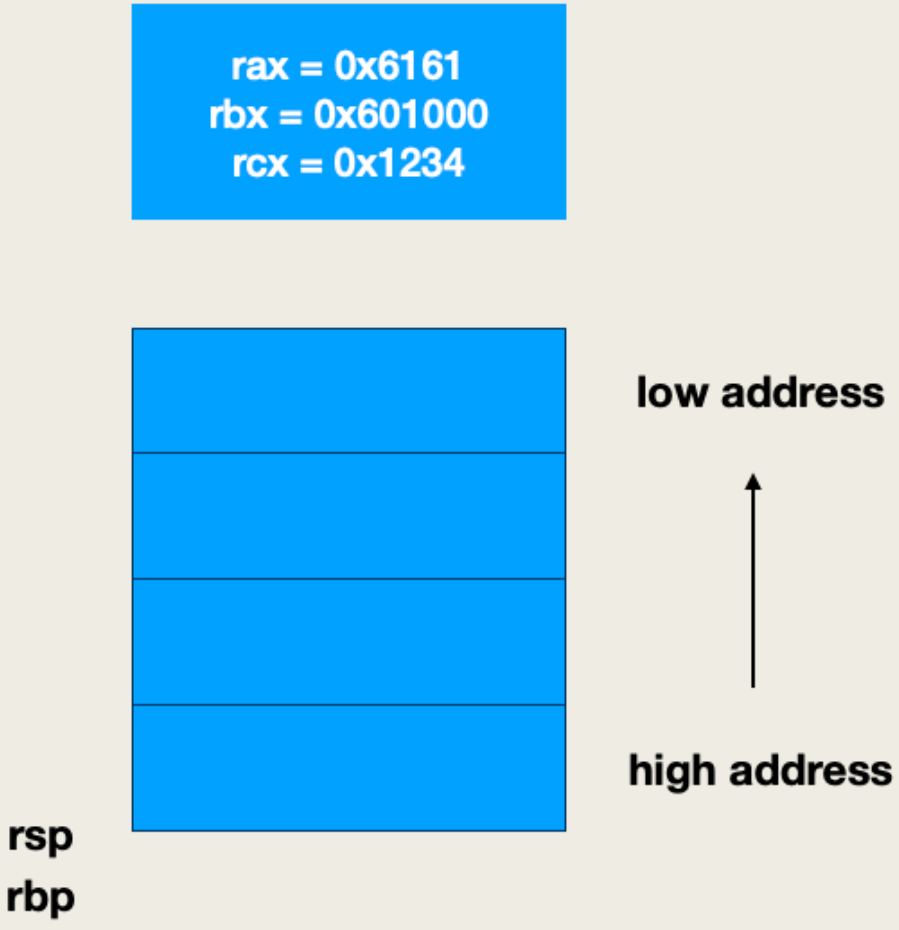
```
//       rdi    rsi    rdx    rcx    r8     r9     stack
int foo(int a, int b, int c, int d, int e, int f, int g)
```

- Microsoft x64 calling convention

```
func1(int a, int b, int c, int d, int e, int f);
// a in RCX, b in RDX, c in R8, d in R9, f then e pushed on stack
```

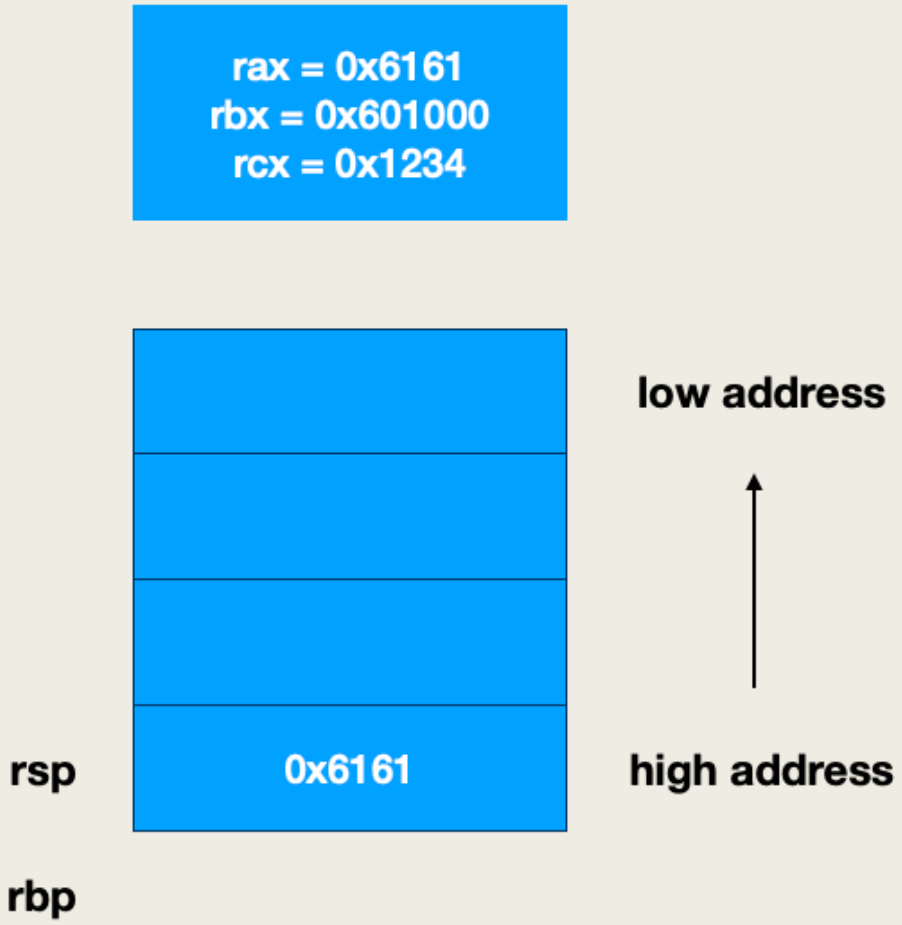
x86-64 Assembly

- Instruction
  - *push, pop val*
  - *example*
    - push rax
    - push 0
    - pop rcx
    - pop word [rbx]
- rsp：stack pointer，指向stack頂端
- rbp：base pointer，指向stack底部



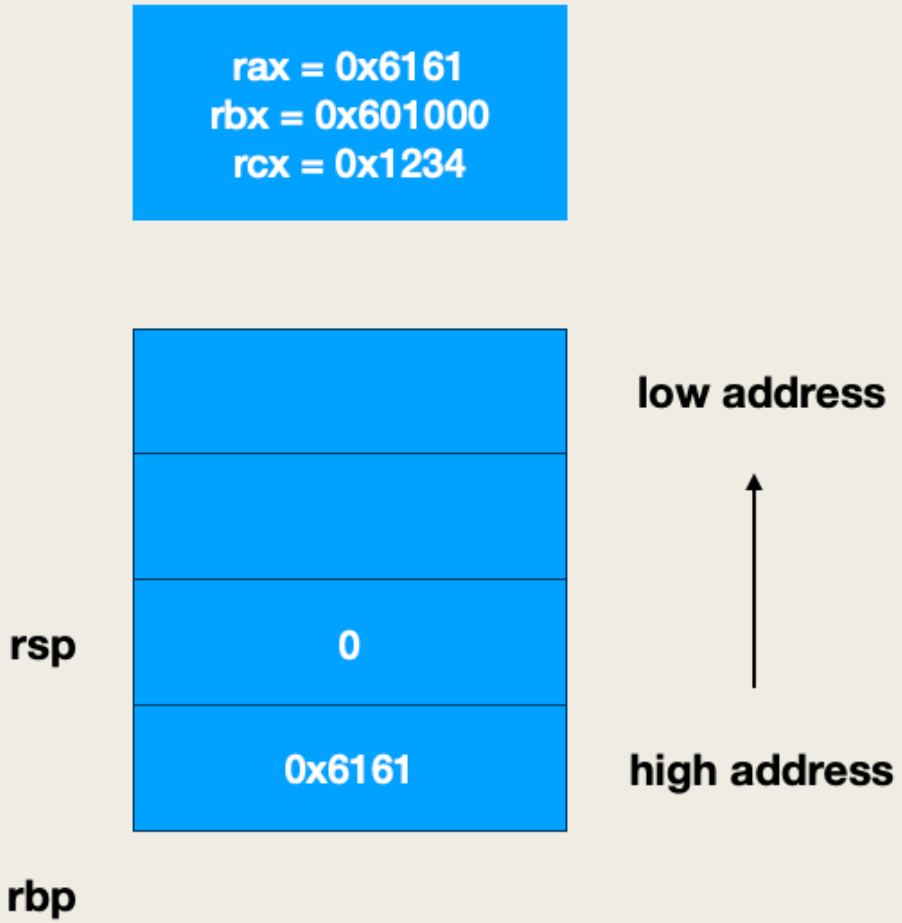
x86-64 Assembly

- Instruction
  - *push, pop val*
  - *example*
    - push rax
    - push 0
    - pop rcx
    - pop word [rbx]



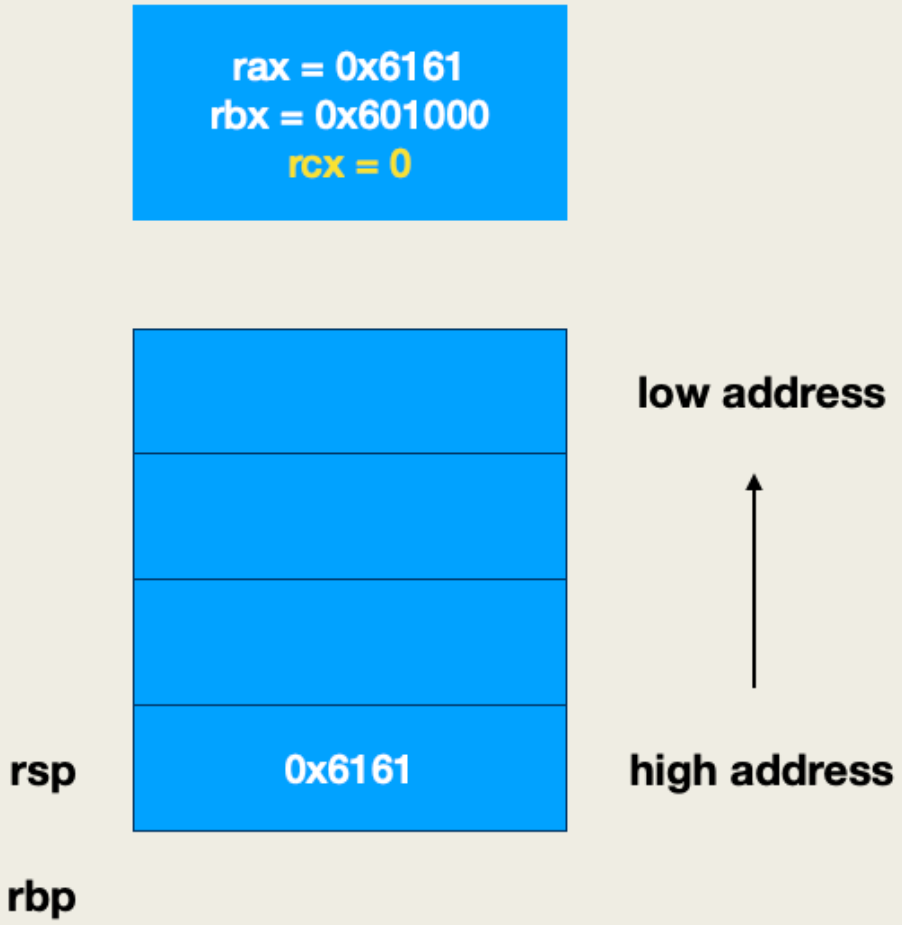
x86-64 Assembly

- Instruction
  - *push, pop val*
  - *example*
    - push rax
    - push 0
    - pop rcx
    - pop word [rbx]



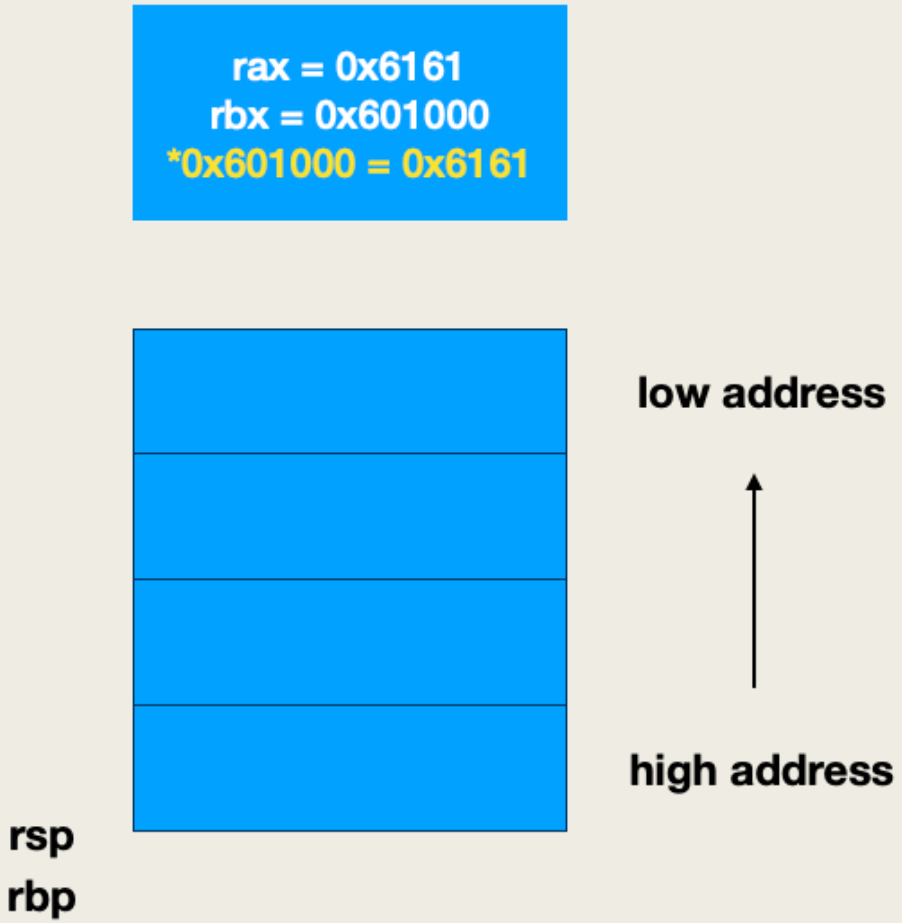
x86-64 Assembly

- Instruction
  - *push, pop val*
  - *example*
    - push rax
    - push 0
    - pop rcx
    - pop word [rbx]



x86-64 Assembly

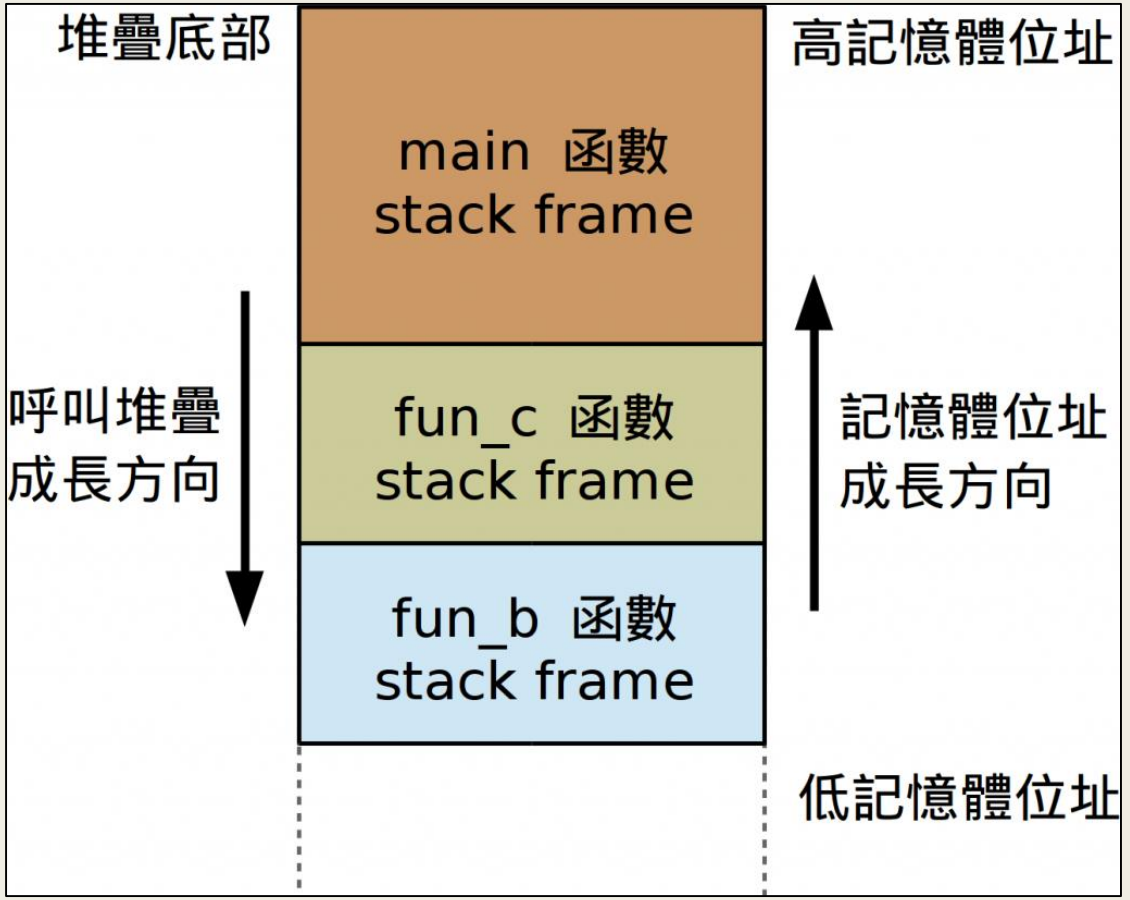
- Instruction
  - *push, pop val*
  - *example*
    - push rax
    - push 0
    - pop rcx
    - pop word [rbx]



x86-64 Assembly

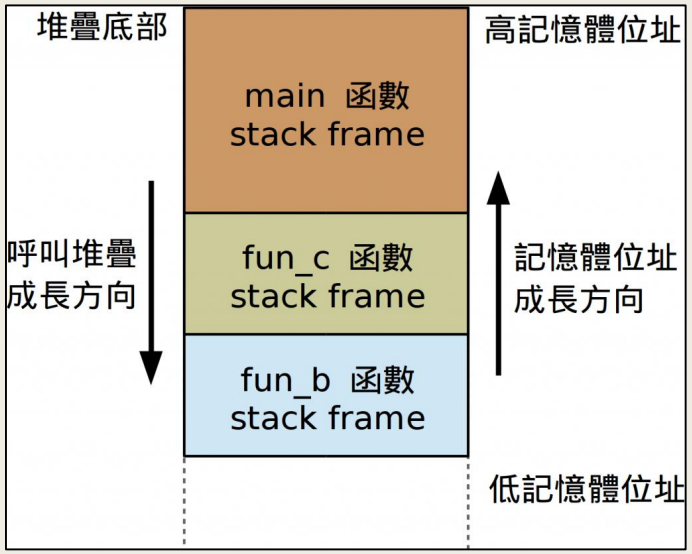
- Stack
  - *the region between rsp (top) and rbp (bottom)*
  - *stores information about the active subroutines*
    - record the local variables
    - record the return address

- Stack frame layout



x86-64 Assembly

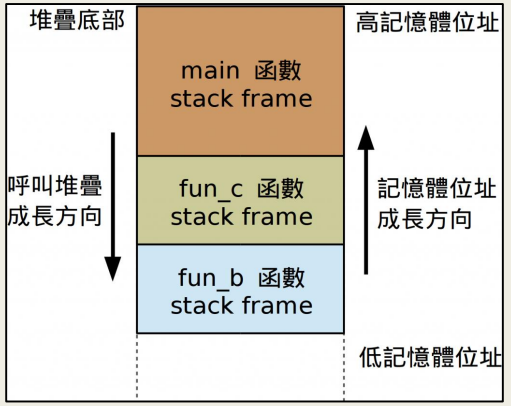
- Translate between assembly and C/C++
  - *function*
    - prologue and epilogue




```asm
push    rbp
mov     rbp, rsp
sub     rsp, N
```

```asm
    mov     rsp, rbp
    pop     rbp
    ret

; also can be written as
    leave
    ret
```

x86-64 Assembly



- Translate between assembly and C/C++
  - *function*

c

```c
int add(int a, int b) {
    return a + b;
}

int main() {
    int a = 1, b = 2;
    printf("%d\n", add(a, b));
    return 0;
}
```

asm

```asm
add:
  push    rbp
  mov     rbp, rsp
  mov     DWORD [rbp - 0x4], edi
  mov     DWORD [rbp - 0x8], esi
  mov     edx, DWORD [rbp - 0x4]
  mov     eax, DWORD [rbp - 0x8]
  add     eax, edx
  pop     rbp
  ret
```

asm

```asm
main:
  push    rbp
  mov     rbp, rsp
  sub     rsp, 0x10
  mov     DWORD [rbp - 0x8], 0x1
  mov     DWORD [rbp - 0x4], 0x2
  mov     edx, DWORD [rbp - 0x4]
  mov     eax, DWORD [rbp - 0x8]
  mov     esi, edx
  mov     edi, eax
  call    add
  mov     esi, eax
  lea     rdi, [rip + 0x9a]
  mov     eax, 0x0
  call    printf
  mov     eax, 0x0
  leave
  ret
```

```c
//        rdi    rsi    rdx    rcx    r8     r9     stack
int foo(int a, int b, int c, int d, int e, int f, int g)
```

x86-64 calling conventions

- C/C++ 中經常見到的三種函數調用方式
  - cdecl (C declaration，即C聲明)是源起C語言的一種呼叫約定，也是C語言的標準。
  - stdcall 是由微軟建立的呼叫約定，是 Windows API 的標準呼叫約定。非微軟的編譯器並不總是支援該呼叫協定。
  - fastcall 約定還未被標準化，不同編譯器的實現也不一致。

	CDECL	STDCALL	FASTCALL
Parameters	Pushed on the stack from right-to-left. Caller must clean up the stack after the call.	Same as CDECL except that the callee must clean the stack.	First two parameters are passed in ECX and EDX. The rest are on the stack.
Return value	Stored in EAX.	Stored in EAX.	Stored in EAX.
Non-volatile registers	EBP, ESP, EBX, ESI, EDI.	EBP, ESP, EBX, ESI, EDI.	EBP, ESP, EBX, ESI, EDI.

same

x86-64 calling conventions
Stack Cleaning

```
400100 func:
400100      push ebp
400101      mov ebp, esp
400103      sub esp, 0x20
            ...
400187      mov eax, 1 ; return 1
40018c      leave
40018d      ret

400000 main:
            ...
4000ab      push 0x2
4000ad      push 0x1
4000af      call func ; func(1, 2)
4000b4      add  esp, 0x8
```

▲CDECL

```
400100 func:
400100      push ebp
400101      mov ebp, esp
400103      sub esp, 0x20
            ...
400187      mov eax, 1 ; return 1
40018c      leave
40018d      ret 8

400000 main:
            ...
4000ab      push 0x2
4000ad      push 0x1
4000af      call func ; func(1, 2)
4000b4      ...
```

▲STDCALL

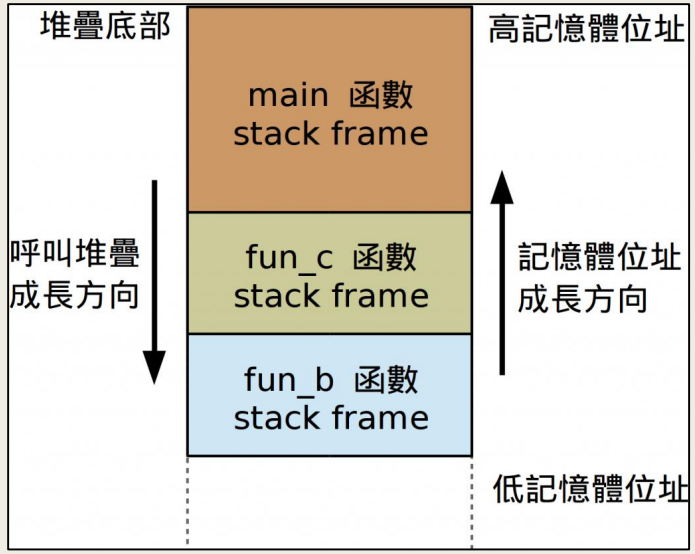
```
400100 func:
400100      push ebp
400101      mov ebp, esp
400103      sub esp, 0x20
            ...
400187      mov eax, 1 ; return 1
40018c      leave
40018d      ret

400000 main:
            ...
4000ab      mov edx, 0x2
4000ad      mov ecx, 0x1
4000af      call func ; func(1, 2)
4000b4      ...
```

▲FASTCALL

On 32-bit x86 systems, the parent function pushes parameters on the stack.

Stack Cleaning

```c
#include "stdio.h"
void a(void) {
    int i;
    printf("%d ", i);
    i = 10;
    printf("%d\n", i);
}
void b(void) {
    int i;
    printf("%d ", i);
    i = 20;
    printf("%d\n", i);
}
int main(){
    printf("Hello, world!\n");
    a();
    b();
    a();
    return 0;
}
```




```
Hello, world!
32767 10
10 20
20 10
```

volatile/non-volatile registers

- volatile registers 顧名思義，這些通用寄存器通常保存臨時（易失）信息，這些信息可以被任何函數覆蓋。因此，caller 有責任在呼叫函數前將這些寄存器中的每一個想要保存的值推入堆疊。
  (由 caller 負責保存)

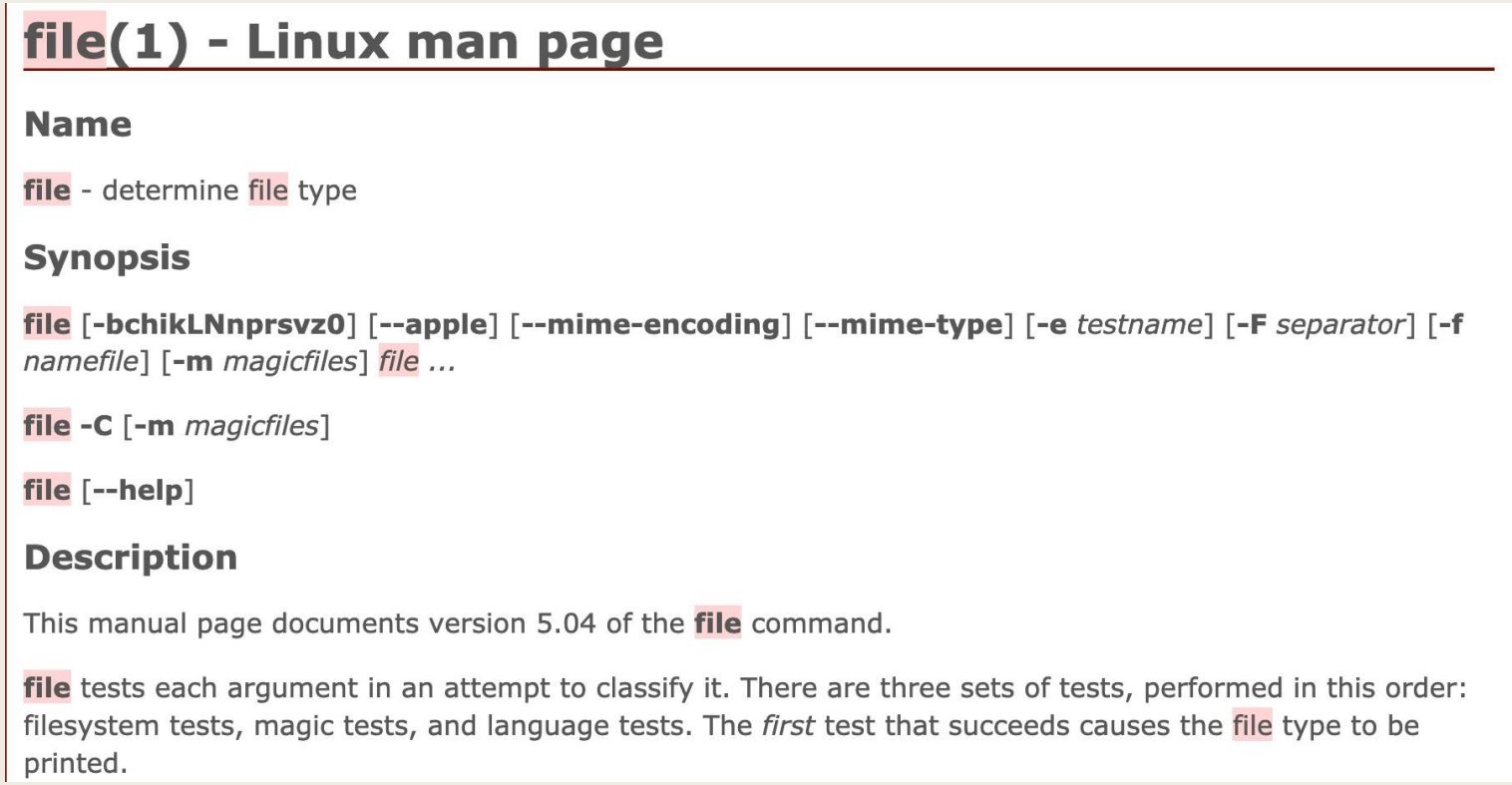
- non-volatile registers 用於保存長期值（非易失性），callee 有責任保存想要動用的寄存器並在返回給 caller 之前恢復它們，當 caller 進行函數呼叫時，可以期望這些寄存器在 callee 返回後將保持相同的值。
  (由 callee 負責保存)

Basic Tools

- Static analysis
  - *file / strings*
  - *readelf / objdump*
- Dynamic analysis
  - *strace / ltrace*
  - *gdb*

Basic Tools

- Static analysis – file



file(1) - Linux man page

Name

file - determine file type

Synopsis

file [**-bchikLNnprsvz0**] [**--apple**] [**--mime-encoding**] [**--mime-type**] [**-e** *testname*] [**-F** *separator*] [**-f** *namefile*] [**-m** *magicfiles*] *file* ...

file -C [**-m** *magicfiles*]

file [**--help**]

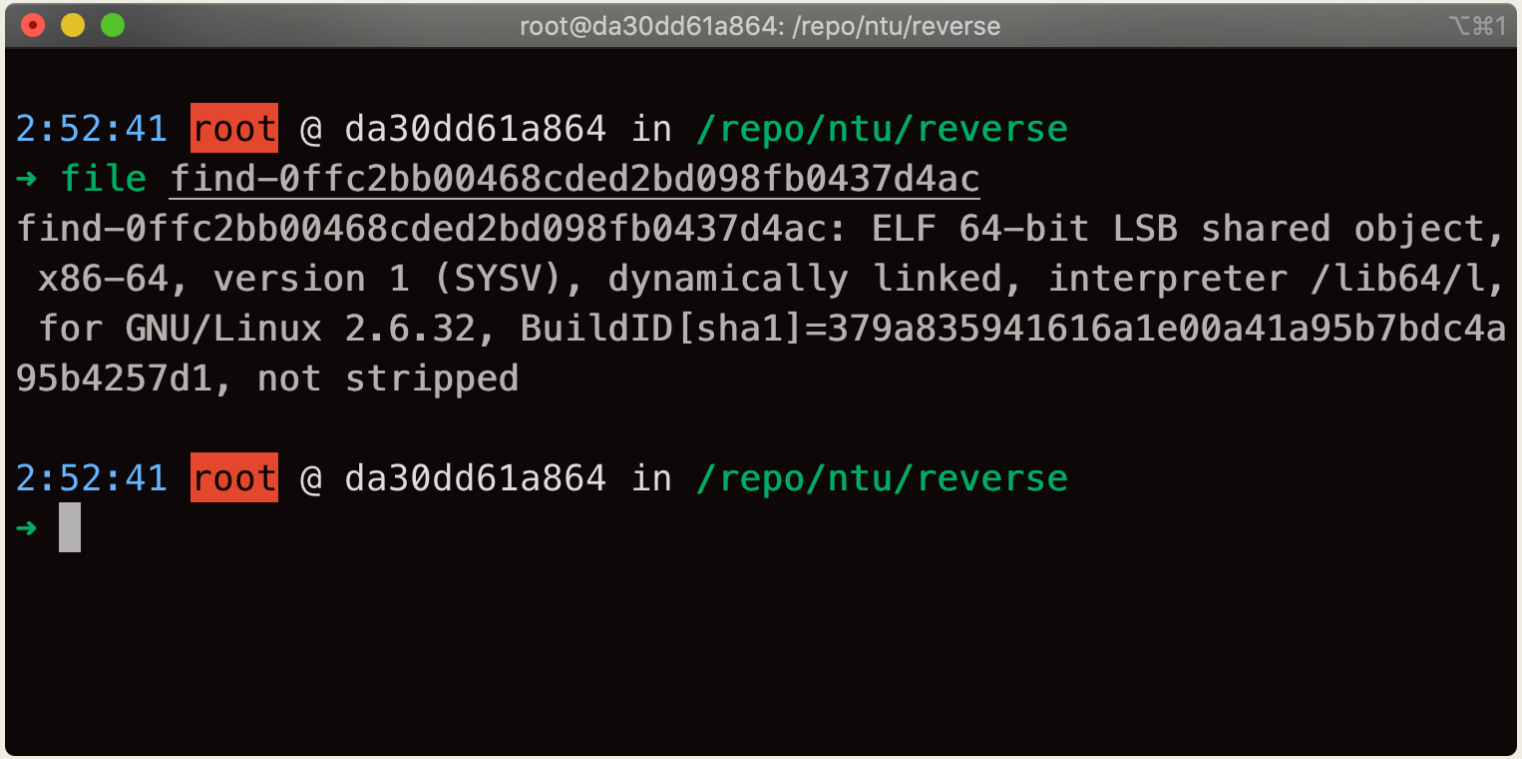
Description

This manual page documents version 5.04 of the **file** command.

file tests each argument in an attempt to classify it. There are three sets of tests, performed in this order: filesystem tests, magic tests, and language tests. The *first* test that succeeds causes the file type to be printed.

Basic Tools

- Static analysis – file



Basic Tools

- Static analysis – strings

strings(1) - Linux man page

Name

strings - print the strings of printable characters in files.

Synopsis

strings [**-afovV**] [**-***min-len*] [**-n** *min-len*] [**--bytes=***min-len*] [**-t** *radix*] [**--radix=***radix*] [**-e** *encoding*] [**--encoding=***encoding*] [**-**] [**--all**] [**--print-file-name**] [**-T** *bfdname*] [**--target=***bfdname*] [**--help**] [**--version**] *file*...

Description

For each *file* given, GNU **strings** prints the printable character sequences that are at least 4 characters long (or the number given with the options below) and are followed by an unprintable character. By default, it only prints the strings from the initialized and loaded sections of object files; for other types of files, it prints the strings from the whole file.

strings is mainly useful for determining the contents of non-text files.

Basic Tools

- Static analysis – strings

Basic Tools

- Static analysis – readelf

readelf(1) - Linux man page

Name

readelf - Displays information about ELF files.

Synopsis

readelf [**-a**|**--all**] [**-h**|**--file-header**] [**-l**|**--program-headers**|**--segments**] [**-S**|**--section-headers**|**--sections**] [**-g**|**--section-groups**] [**-t**|**--section-details**] [**-e**|**--headers**] [**-s**|**--syms**|**--symbols**] [**-n**|**--notes**] [**-r**|**--relocs**] [**-u**|**--unwind**] [**-d**|**--dynamic**] [**-V**|**--version-info**] [**-A**|**--arch-specific**] [**-D**|**--use-dynamic**] [**-x** <number or name>|**--hex-dump=**<number or name>] [**-p** <number or name>|**--string-dump=**<number or name>] [**-R** <number or name>|**--relocated-dump=**<number or name>] [**-c**|**--archive-index**] [**-w[lLiaprmfFsoR]**| **--debug-dump**[=rawline,=decodedline,=info,=abbrev,=pubnames,=aranges,=macro,=frames,=frames-interp,=str,=loc,=Ranges]] [**-I**|**-histogram**] [**-v**|**--version**] [**-W**|**--wide**] [**-H**|**--help**] *elffile*...

Description

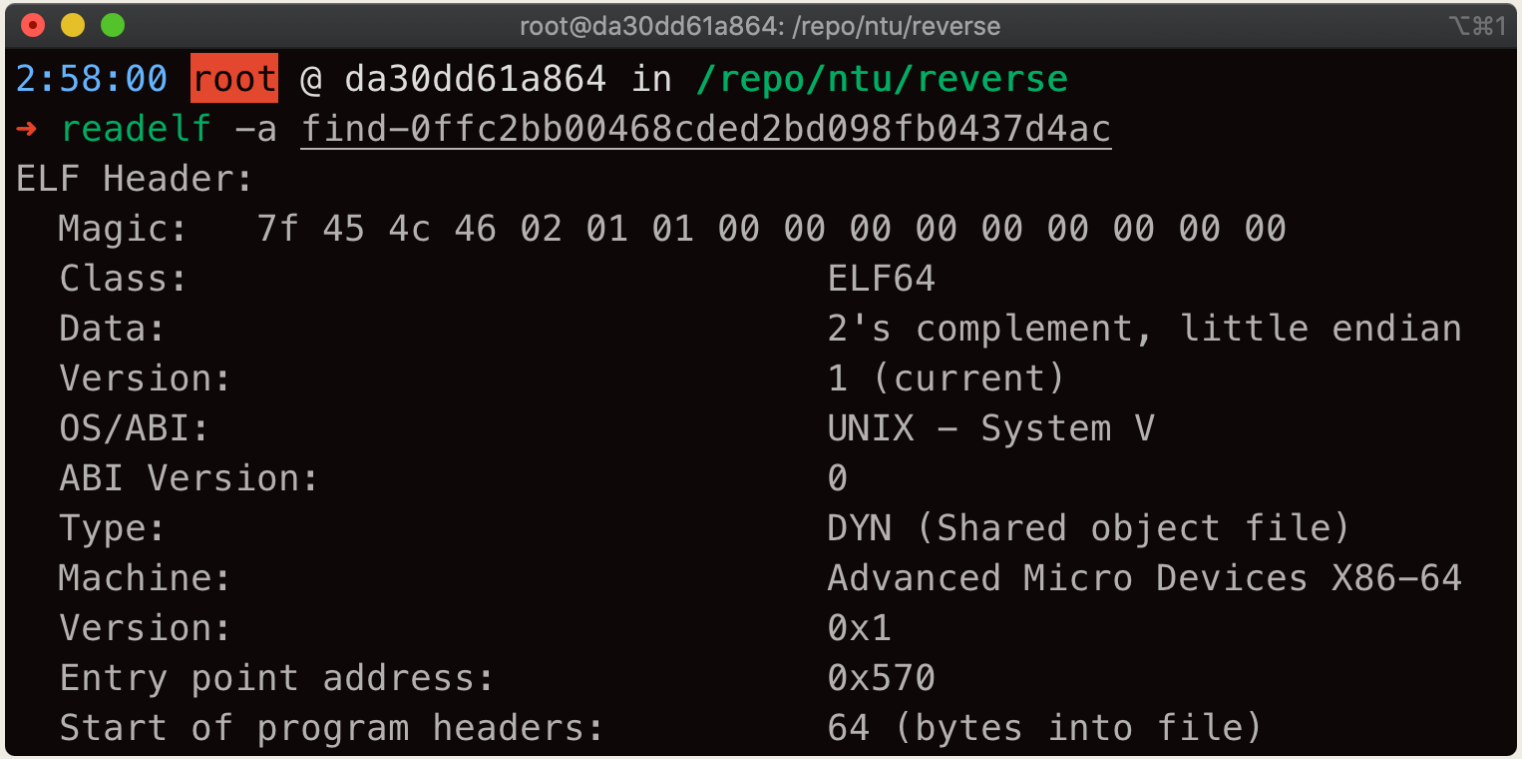
readelf displays information about one or more ELF format object files. The options control what particular information to display.

elffile... are the object files to be examined. 32-bit and 64-bit ELF files are supported, as are archives containing ELF files.

This program performs a similar function to **objdump** but it goes into more detail and it exists independently of the BFD library, so if there is a bug in BFD then readelf will not be affected.

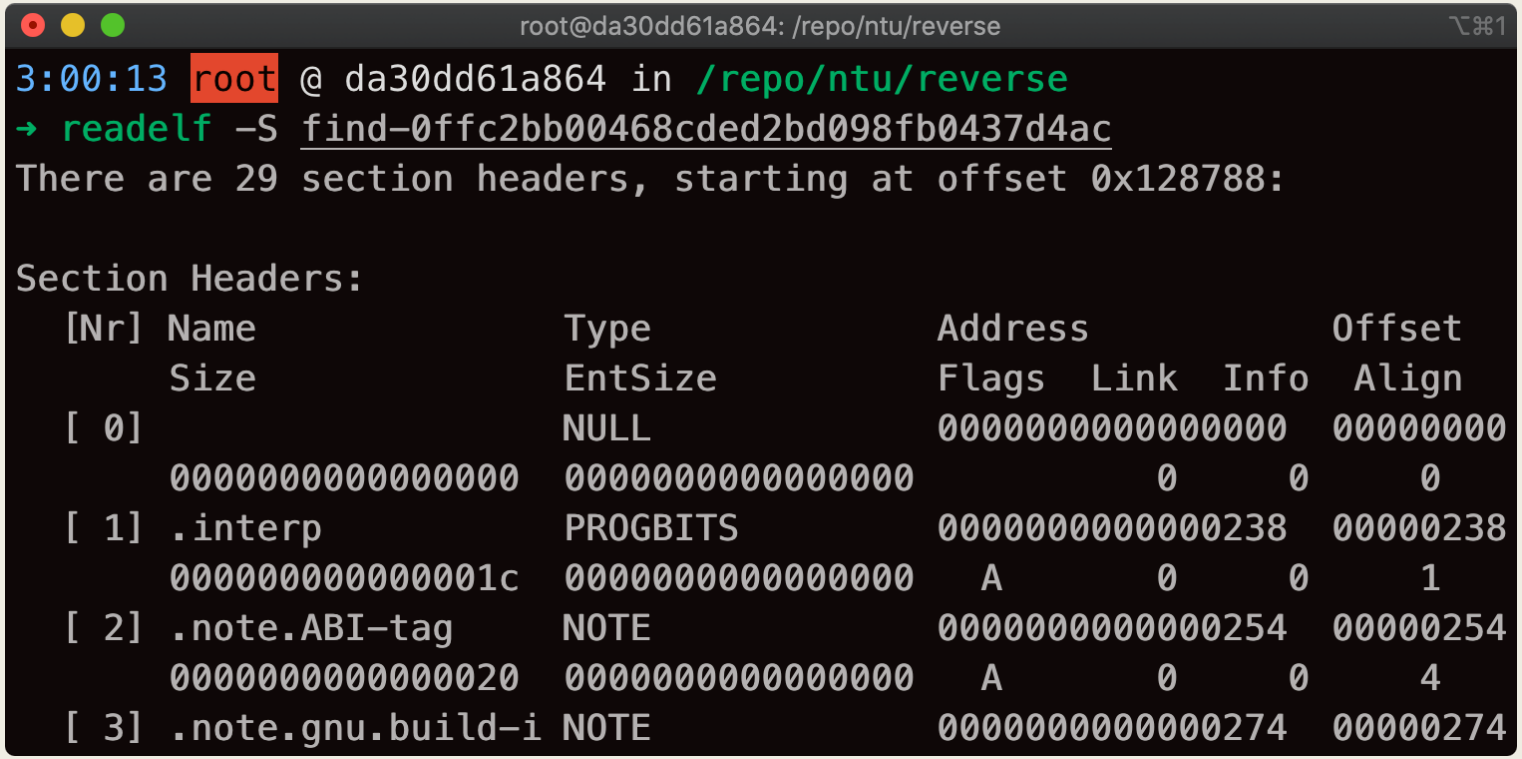
Basic Tools

- Static analysis – readelf (all)



Basic Tools

- Static analysis – readelf (section headers)



Basic Tools

- Static analysis – objdump

objdump(1) - Linux man page

Name

objdump - display information from object files.

Synopsis

objdump [**-a**|**--archive-headers**] [**-b** *bfdname*|**--target=***bfdname*] [**-C**|**--demangle**[=*style*]] [**-d**|**--disassemble**] [**-D**|**--disassemble-all**] [**-z**|**--disassemble-zeroes**] [**-EB**|**-EL**|**--endian=**{big | little }] [**-f**|**--file-headers**] [**-F**|**--file-offsets**] [**--file-start-context**] [**-g**|**--debugging**] [**-e**|**--debugging-tags**] [**-h**|**--section-headers**|**--headers**] [**-i**|**--info**] [**-j** *section*|**--section=***section*] [**-l**|**--line-numbers**] [**-S**|**--source**] [**-m** *machine*|**--architecture=***machine*] [**-M** *options*|**--disassembler-options=***options*] [**-p**|**--private-headers**] [**-r**|**--reloc**] [**-R**|**--dynamic-reloc**] [**-s**|**--full-contents**] [**-W[lLiaprmfFsoR]**|**--dwarf**[=rawline,=decodedline,=info,=abbrev,=pubnames,=aranges,=macro,=frames,=frames-interp,=str,=loc,=Ranges]] [**-G**|**--stabs**] [**-t**|**--syms**] [**-T**|**--dynamic-syms**] [**-x**|**--all-headers**] [**-w**|**--wide**] [**--start-address=***address*] [**--stop-address=***address*] [**--prefix-addresses**] [**--[no-]show-raw-insn**] [**--adjust-vma=***offset*] [**--special-syms**] [**--prefix=***prefix*] [**--prefix-strip=***level*] [**--insn-width=***width*] [**-V**|**--version**] [**-H**|**--help**] *objfile*...

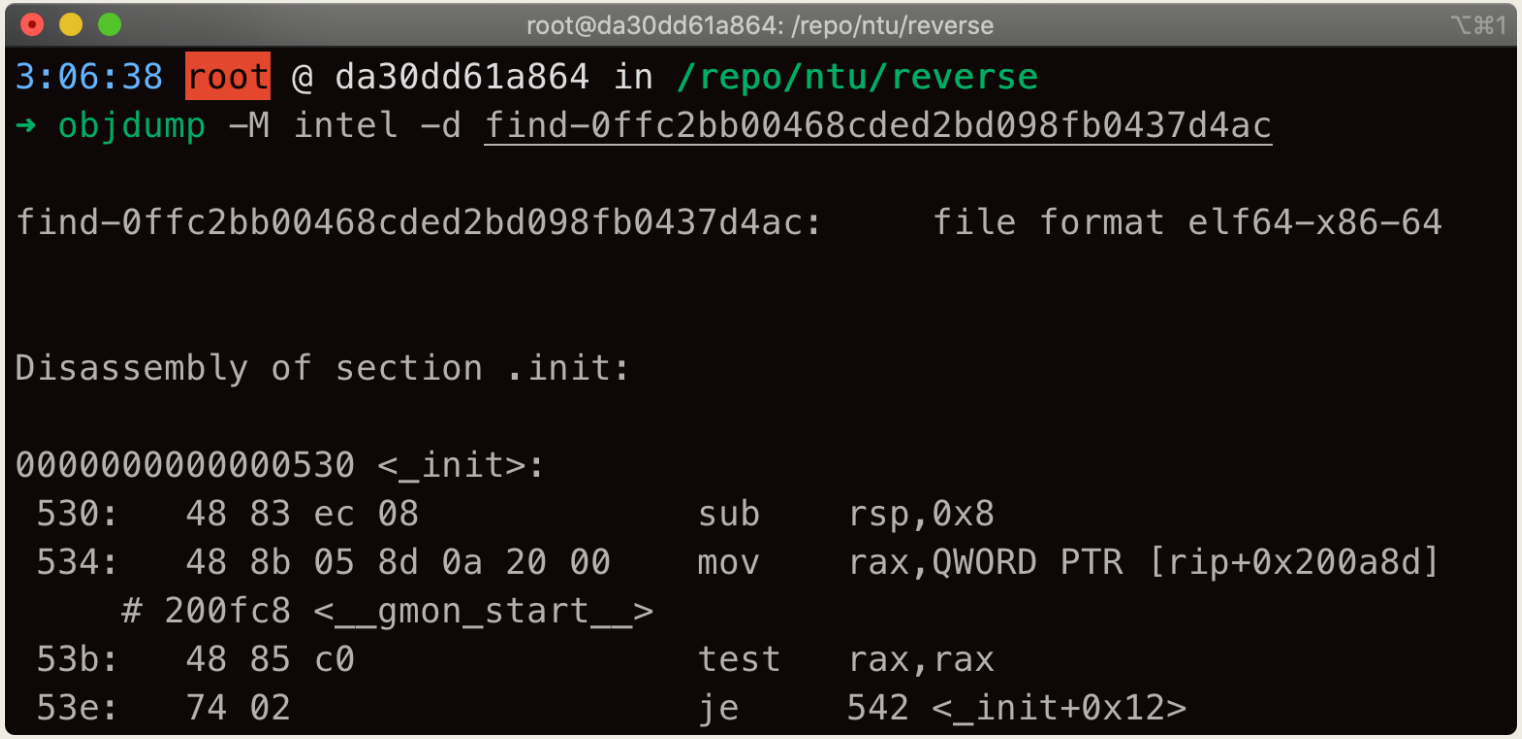
Description

objdump displays information about one or more object files. The options control what particular information to display. This information is mostly useful to programmers who are working on the compilation tools, as opposed to programmers who just want their program to compile and work.

objfile... are the object files to be examined. When you specify archives, **objdump** shows information on each of the member object files.

Basic Tools

- Static analysis – objdump (disassemble)



Basic Tools

- Dynamic analysis – strace

strace(1) - Linux man page

Name

strace - trace system calls and signals

Synopsis

strace [**-dDffhiqrtttTvVxx**] [**-a***column*] [**-e***expr*] ... [**-o***file*] [**-p***pid*] ... [**-s***strsize*] [**-u***username*] [**-E***var*=*val*] ... [**-E***var*] ... [*command* [*arg* ...]]

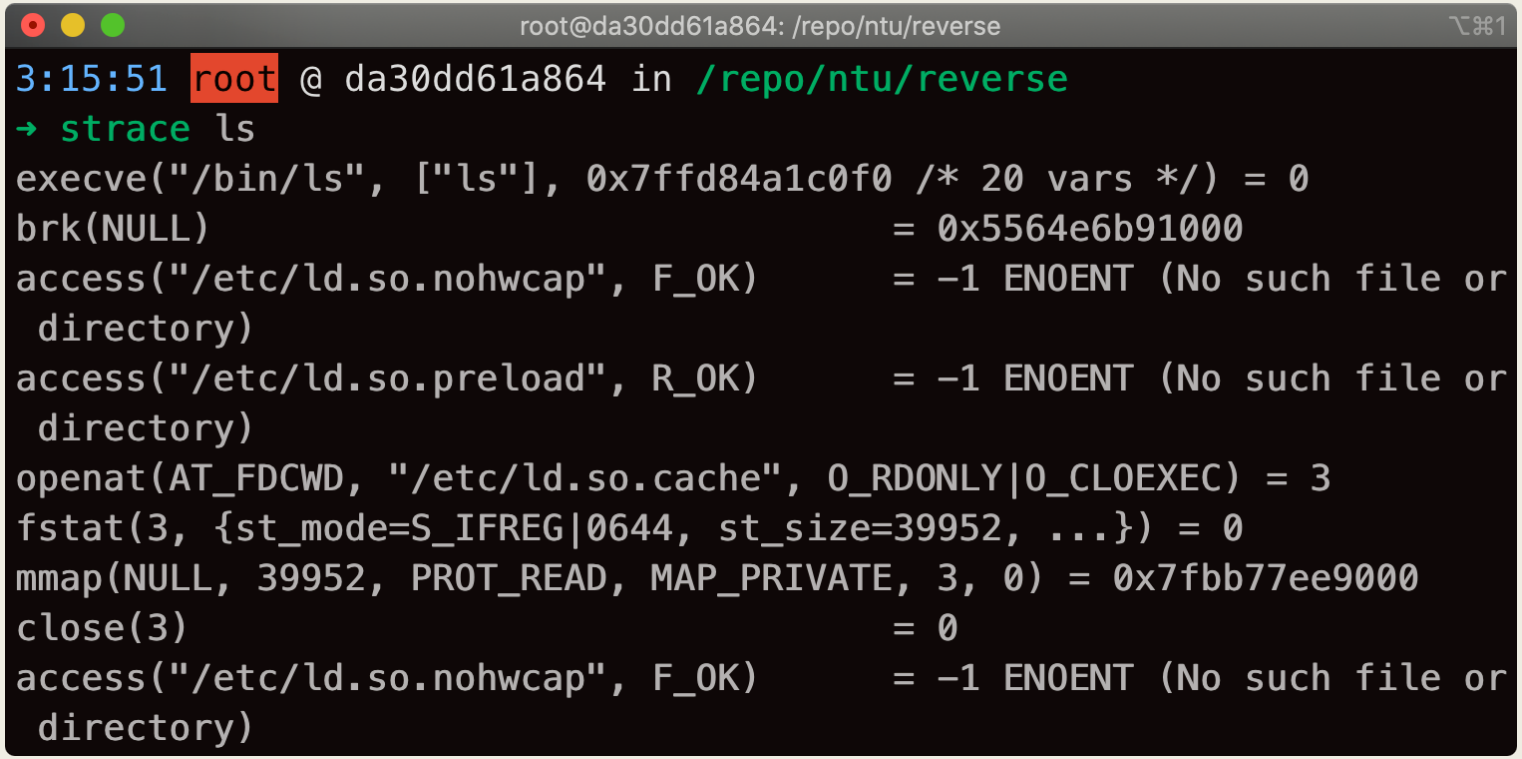
strace -c [**-D**] [**-e***expr*] ... [**-O***overhead*] [**-S***sortby*] [*command* [*arg* ...]]

Description

In the simplest case **strace** runs the specified *command* until it exits. It intercepts and records the system calls which are called by a process and the signals which are received by a process. The name of each system call, its arguments and its return value are printed on standard error or to the file specified with the **-o** option.

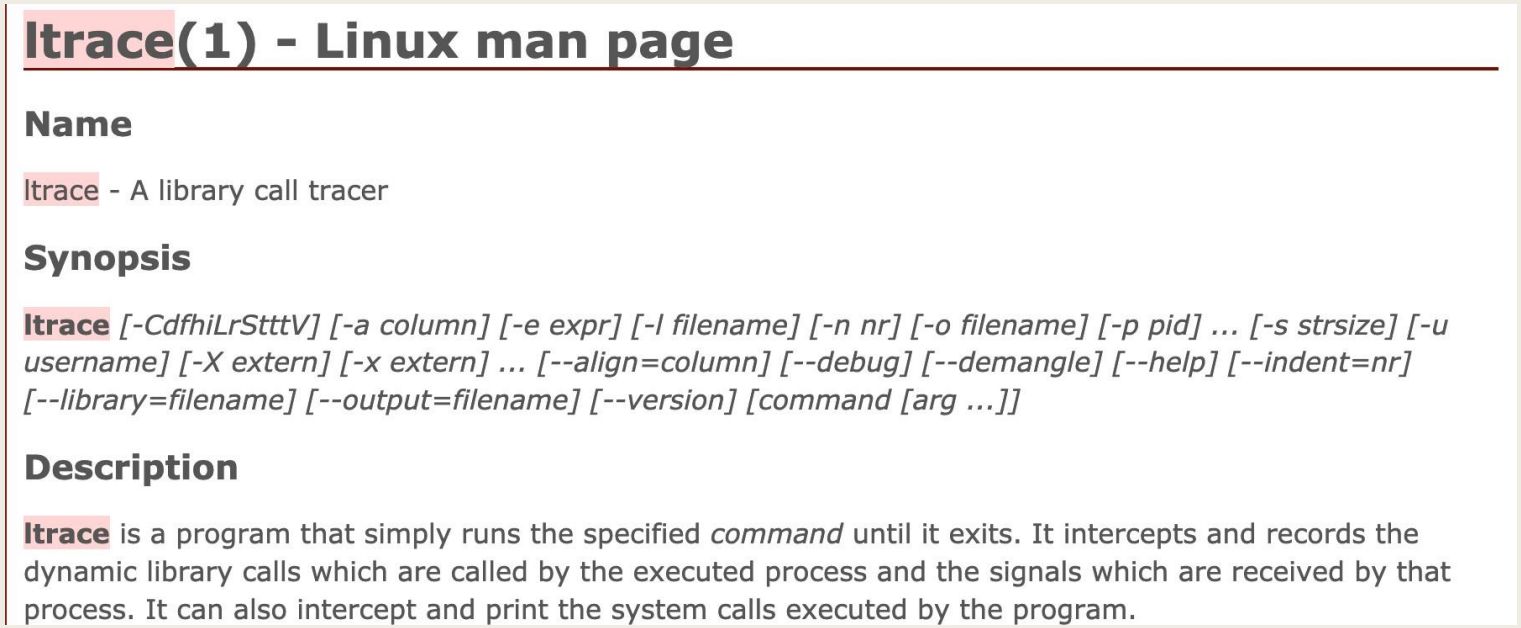
Basic Tools

- Dynamic analysis – strace



Basic Tools

- Dynamic analysis – ltrace



ltrace(1) - Linux man page

Name

ltrace - A library call tracer

Synopsis

ltrace *[-CdfhiLrSTttV] [-a column] [-e expr] [-l filename] [-n nr] [-o filename] [-p pid] ... [-s strsize] [-u username] [-X extern] [-x extern] ... [--align=column] [--debug] [--demangle] [--help] [--indent=nr] [--library=filename] [--output=filename] [--version] [command [arg ...]]*

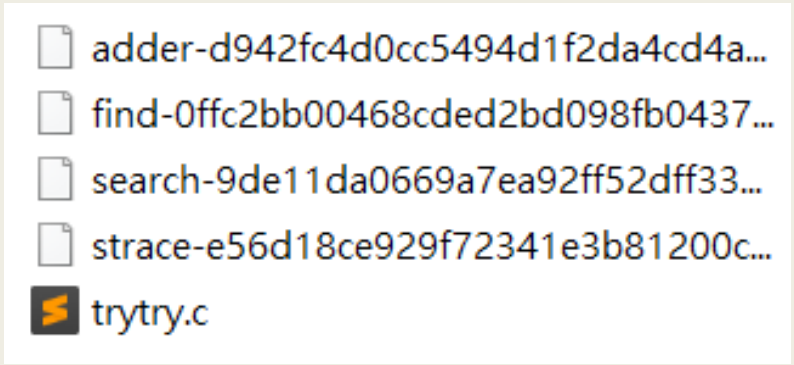
Description

ltrace is a program that simply runs the specified *command* until it exits. It intercepts and records the dynamic library calls which are called by the executed process and the signals which are received by that process. It can also intercept and print the system calls executed by the program.

Basic Tools

- Dynamic analysis – ltrace

Labs


adder-d942fc4d0cc5494d1f2da4cd4a...
find-0ffc2bb00468cded2bd098fb0437...
search-9de11da0669a7ea92ff52dff33...
strace-e56d18ce929f72341e3b81200c...
trytry.c

Labs - find

- 用 strings 找出長度大於 10 的字串，再用 grep 找有沒有匹配的 FLAG

```
CYC[kali]strings find-0ffc2bb00468cded2bd098fb0437d4ac  -n 10 | grep -E 'MyFirstCTF{[a-zA-Z0-9_!@]+}'
MyFirstCTF{Y0U_gOt_th3_flag_bY_sTr1ngS}
CYC[kali]
```

Labs - strace

整數值	名稱	<unistd.h>符號常數[1]	<stdio.h>檔案流[2]
0	Standard input	STDIN_FILENO	stdin
1	Standard output	STDOUT_FILENO	stdout
2	Standard error	STDERR_FILENO	stderr

- 用 strace 找出呼叫的 syscall，並在參數中發現 FLAG

```
CYC[kali]./strace-e56d18ce929f72341e3b81200cc4929f
find the flag in system call!
CYC[kali]strace ./strace-e56d18ce929f72341e3b81200cc4929f
execve("./strace-e56d18ce929f72341e3b81200cc4929f", ["./strace-e56d18ce929f72341e3b812"... ], 0x7ffe04576ea0 /* 20 vars */) = 0
uname({sysname="Linux", nodename="kali", ...}) = 0
brk(NULL)                               = 0xa31000
brk(0xa321c0)                           = 0xa321c0
arch_prctl(ARCH_SET_FS, 0xa31880)       = 0
readlink("/proc/self/exe", "/media/sf_SEEDVM/rev1/strace-e56"..., 4096) = 61
brk(0xa531c0)                           = 0xa531c0
brk(0xa54000)                           = 0xa54000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
close(2)                                = 0
close(0)                                = 0
write(2, "FLAG{____yaaaa_flag_in_the_stack"..., 36) = -1 EBADF (Bad file descriptor)
fstat(1, {st_mode=S_IFCHR|0600, st_rdev=makedev(0x88, 0), ...}) = 0
write(1, "find the flag in system call!\n", 30find the flag in system call!
) = 30
exit_group(0)                           = ?
+++ exited with 0 +++
```

Labs - search

- objdump - 把檔案中可疑的 16 進位組合成字串，再用 grep 找有沒有匹配的 FLAG

```
136
137 00000000000006b7 <do_nothing>:
138     6b7: 55                         push   rbp
139     6b8: 48 89 e5                   mov    rbp,rsp
140     6bb: c6 05 8c 19 21 00 25       mov    BYTE PTR [rip+0x21198c],0x25        # 21204e <s+0xe>
141     6c2: c6 05 9e 19 21 00 25       mov    BYTE PTR [rip+0x21199e],0x25        # 212067 <s+0x27>
142     6c9: c6 05 8e 19 21 00 6a       mov    BYTE PTR [rip+0x21198e],0x6a        # 21205e <s+0x1e>
143     6d0: c6 05 6e 19 21 00 44       mov    BYTE PTR [rip+0x21196e],0x44        # 212045 <s+0x5>
144     6d7: c6 05 79 19 21 00 22       mov    BYTE PTR [rip+0x211979],0x22        # 212057 <s+0x17>
145     6de: c6 05 64 19 21 00 5e       mov    BYTE PTR [rip+0x211964],0x5e        # 212049 <s+0x9>
146     6e5: c6 05 71 19 21 00 5f       mov    BYTE PTR [rip+0x211971],0x5f        # 21205d <s+0x1d>
147     6ec: c6 05 72 19 21 00 69       mov    BYTE PTR [rip+0x211972],0x69        # 212065 <s+0x25>
148     6f3: c6 05 66 19 21 00 4c       mov    BYTE PTR [rip+0x211966],0x4c        # 212060 <s+0x20>
149     6fa: c6 05 53 19 21 00 60       mov    BYTE PTR [rip+0x211953],0x60        # 212054 <s+0x14>
150     701: c6 05 49 19 21 00 2a       mov    BYTE PTR [rip+0x211949],0x2a        # 212051 <s+0x11>
151     708: c6 05 57 19 21 00 71       mov    BYTE PTR [rip+0x211957],0x71        # 212066 <s+0x26>
152     70f: c6 05 2f 19 21 00 4f       mov    BYTE PTR [rip+0x21192f],0x4f        # 212045 <s+0x5>
153     716: c6 05 40 19 21 00 3c       mov    BYTE PTR [rip+0x211940],0x3c        # 21205d <s+0x1d>
154     71d: c6 05 34 19 21 00 7e       mov    BYTE PTR [rip+0x211934],0x7e        # 212058 <s+0x18>
155     724: c6 05 35 19 21 00 45       mov    BYTE PTR [rip+0x211935],0x45        # 212060 <s+0x20>
156     72b: c6 05 31 19 21 00 2e       mov    BYTE PTR [rip+0x211931],0x2e        # 212063 <s+0x23>
157     732: c6 05 1b 19 21 00 57       mov    BYTE PTR [rip+0x21191b],0x57        # 212054 <s+0x14>
158     739: c6 05 00 19 21 00 78       mov    BYTE PTR [rip+0x211900],0x78        # 212040 <s>
159     740: c6 05 0d 19 21 00 29       mov    BYTE PTR [rip+0x21190d],0x29        # 212054 <s+0x14>
160     747: c6 05 f2 18 21 00 4b       mov    BYTE PTR [rip+0x2118f2],0x4b        # 212040 <s>
```

Labs - search

- 把檔案中可疑的 16 進位組合成字串，再用 grep 找有沒有匹配的 FLAG

```
CYC[kali]cat search_objdump.txt | grep 'mov    BYTE PTR \[rip+0x' | awk -F 'x' '{print$3}' | awk '{print$1}' | x
args | tr -d '[:space:]' | python -c 'print(raw_input()[1:].decode("hex"))' | grep 'MyFirstCTF{'
grep: grep: warning: GREP_OPTIONS is deprecated; please use an alias or script
warning: GREP_OPTIONS is deprecated; please use an alias or script
%%jD"^_iL`*qO←~E.Wx)K[eJ+h$YM0~]3shl%@4<p**dIX"^s(fy%d+A?.54h&kCy[VO~$d$/p;~ll⟹ka[(hY-Y~<%e^dThbopn^K~/ypiWR~D_
TlUD_^Q8x?u|6RipKSpEAOpLvRE"fT`E6;%E(mIizk1_(XB:q`snj=.?fqik1EDdd,i0B9YEA<[+KyOz;65+}..n_]!~;sS:WT3vFfI5ONlv<U1*q
DEf6?=F7~1MVX0%i~QQi89$b" &Qn~t`h/mJLhhPD9Nl,UQ;c$?Cr59@o^?5zQ?U9pY_ooNW+mf-1U9DLB?PxZA""5Lgevs=H*FF;vL1^[2^;h$8c
9/aRm+w|+hf~.QA`BBMXP_Lt/0H{X(6{0Q~R& Y4;CSD3<3_C.7LlUox0_^KfAtk:q,xOP>~ gDn"+1^sDV9p9P}gEU`]bR{:B@knX ="Ws]s0Go.
J-oh*p#u XE$CdhO$qKYC+*~H]km[XS#=H[D_Lyq+hr.H2LZ+sbir%Z_Fyo.@?nXr>T&:*Hio-72d;q FDy!mSpo`SL&Cm~(@3:]Zvya*V +c2iHW
]lI(yi?P}.ix|CTc>u~tE^2(@$jech)s@JOyRnU*>%w#vgSgCsWIEC3UlFd5DAYotENV!~o"g4%P[7|$y:5S3$wK(NUJDt0?:#p$RtJ>UL3+ijwe`
UQ1)7_+jaZ;PcUw=RJ$[TW<zbgD,I/ZM=r1LpDW.~d4<_&`43@`^.So7H#_Zb&PP:=Lq&E=!3AgP76c@p6$VQl67.S7;Gt8[_^e${Z(pn[.3QY 1k
0^oUu}nB9uzoo1)n3dpe1AFfHU14#)doM~K[◇x>sgG:F@f:@-N(Lnp]nid>SM~SO)o;{V{mU_JOg<ZjAtz`PA+ _ 54}SI/fj!.iOIeFU/4DN. 4
R@4[m`4g51kLi,P39GxjknAj[/.+a8trDB1]#%VqOU ~VpY7O]i5A8_NoQ4,Ct(l`bmt,3n,dn:a4Hg+b6B/a6~JwXMyFirstCTF{D0_A_1OT_4nd
_f1Nd_F14G}zq8ctJOTY%,$elBSkE$VL0E_."DO]s85X{]26SaBSm.JPSHp%^=vY1N+=~ro?<IPzb "iriv`GfY.{2}!H,am+F~bZORZx*5:pT/b7
```

Labs - adder

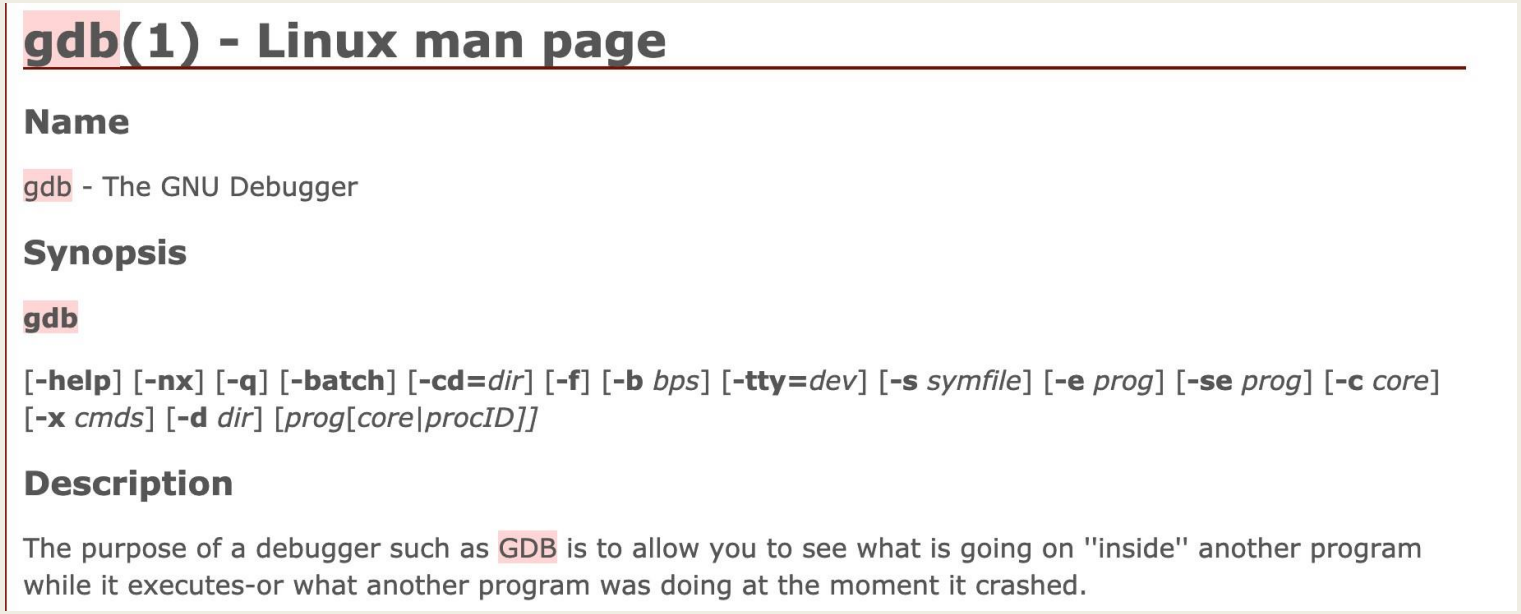
- 用 IDA Pro，發現印 FLAG 的條件是 3 個數字總和為 1337

```
std::operator<<<std::char_traits<char>>(&std::cout, "Enter three numbers!\n", envp);
v3 = std::istream::operator>>(&std::cin, &v9);
v4 = std::istream::operator>>(v3, &v8);
std::istream::operator>>(v4, &v7);
ptr = gen(v8 + v9 + v7);
v5 = (unsigned int)(v8 + v9);
if ( (_DWORD)v5 + v7 == 1337 )
{
  std::operator<<<std::char_traits<char>>(&std::cout, "FLAG{", v5);
  print_ptr((char *)ptr);
  puts("}");
}
```

```
CYC[kali]
CYC[kali]./adder-d942fc4d0cc5494d1f2da4cd4a4d5d39
Enter three numbers!
1337
0
0
FLAG{y0u_added_thr33_nums!}
CYC[kali]
CYC[kali]
CYC[kali]
CYC[kali]
CYC[kali]
CYC[kali]
CYC[kali]
CYC[kali]
```

Basic Tools

- Dynamic analysis - gdb



gdb(1) - Linux man page

Name

gdb - The GNU Debugger

Synopsis

gdb

[**-help**] [**-nx**] [**-q**] [**-batch**] [**-cd=***dir*] [**-f**] [**-b** *bps*] [**-tty=***dev*] [**-s** *symfile*] [**-e** *prog*] [**-se** *prog*] [**-c** *core*] [**-x** *cmds*] [**-d** *dir*] [*prog*[*core*|*procID*]]

Description

The purpose of a debugger such as GDB is to allow you to see what is going on "inside" another program while it executes-or what another program was doing at the moment it crashed.

Basic Tools

- gdb - The GNU Debugger
  - *debug binary with gdb directly*
    - *$ gdb <binary>*
  - *attach and debug running process*
    - *$ gdb*

Basic Tools

- gdb - The GNU Debugger
  - *useful packages*
    - peda: https://github.com/longld/peda
    - Pwngdb: https://github.com/scwuaptx/Pwngdb

Basic Tools

- gdb - The GNU Debugger
  - *installation*
    - *$ git clone https://github.com/longld/peda.git ~/peda*
    - *$ git clone https://github.com/scwuaptx/Pwngdb.git ~/Pwngdb*
    - *$ cp ~/Pwngdb/.gdbinit ~/*

Basic Tools

- gdb - The GNU Debugger

```gdb
# 設定斷點
# break *<address>
break *0x4004d7

# 執行程式
run
```

```gdb
# 執行下一個指令
# 會追進 function 中
step

# 執行下一個指令
# 不會追進 function 中
next
```

```gdb
# 繼續執行
continue

# 執行至 function 結束
finish
```

Basic Tools

- gdb - The GNU Debugger

```gdb
# 查看斷點狀態
info break

# 查看暫存器狀態
info register
```

```gdb
# 刪除斷點
# delete breakpoints <number>
delete breakpoints 1
```

```gdb
# 跳轉
# jump *<address>
jump *0x4004d7
```

Basic Tools

- gdb - The GNU Debugger

gdb

```
# 印出暫存器的值
# print $<register>
print $rax

# 印出記憶體的值
# x <memory address>
x 0x7fffffffe920
```

gdb

```
# 設定暫存器的值
# set $<register> = <value>
set $rsp = 0x7fffffffe920

# 設定記憶體的值
# set {<size>} <memory address> = <value>
set {int} 0x7fffffffe920 = 2
```

Basic Tools

- Try to debug
  - *trace into function*
  - *set values for register / memory*
  - *jump to certain address*
- Compile and debug
  - *$ gcc -g ./trytry.c -o trytry*
  - *$ gdb ./trytry*

```c
#include <stdio.h>

int foo(int a, int b) {
    puts("this is foo");
    int c = a + b;
    return c;
}

int main() {
    int x, y;
    scanf("%d%d", &x, &y);
    int out = foo(x, y);
    if (out == 0xdeadbeef) {
        puts("this is 0xdeadbeef");
    }
    return 0;
}
```

trytry

```
CYC[kali]gdb ./trytry
GNU gdb (Debian 9.1-3) 9.1
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./trytry...
gdb-peda$ b main
Breakpoint 1 at 0x1177: file ./trytry.c, line 11.
gdb-peda$ r
Starting program: /media/sf_SEEDVM/rev1/trytry
[----------------------------------registers-----------------------------------]
RAX: 0x55555555516f (<main>:    push   rbp)
```

trytry

```
gdb-peda$ r
Starting program: /media/sf_SEEDVM/rev1/trytry
[----------------------------------registers-----------------------------------]
RAX: 0x55555555516f (<main>:    push   rbp)
RBX: 0x0
RCX: 0x7ffff7fb3718 --> 0x7ffff7fb5980 --> 0x0
RDX: 0x7fffffffe638 --> 0x7fffffffe874 ("SHELL=/bin/bash")
RSI: 0x7fffffffe628 --> 0x7fffffffe857 ("/media/sf_SEEDVM/rev1/trytry")
RDI: 0x1
RBP: 0x7fffffffe540 --> 0x5555555551d0 (<__libc_csu_init>:      push   r15)
RSP: 0x7fffffffe530 --> 0x7fffffffe620 --> 0x1
RIP: 0x555555555177 (<main+8>:  lea    rdx,[rbp-0xc])
R8 : 0x0
R9 : 0x7ffff7fe3530 (<_dl_fini>:        push   rbp)
R10: 0x0
R11: 0x0
R12: 0x555555555060 (<_start>:  xor    ebp,ebp)
R13: 0x7fffffffe620 --> 0x1
R14: 0x0
R15: 0x0
EFLAGS: 0x206 (carry PARITY adjust zero sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x55555555516f <main>:       push   rbp
   0x555555555170 <main+1>:     mov    rbp,rsp
   0x555555555173 <main+4>:     sub    rsp,0x10
=> 0x555555555177 <main+8>:     lea    rdx,[rbp-0xc]
   0x55555555517b <main+12>:    lea    rax,[rbp-0x8]
   0x55555555517f <main+16>:    mov    rsi,rax
   0x555555555182 <main+19>:    lea    rdi,[rip+0xe87]        # 0x555555556010
   0x555555555189 <main+26>:    mov    eax,0x0
[-------------------------------------stack-------------------------------------]
0000| 0x7fffffffe530 --> 0x7fffffffe620 --> 0x1
0008| 0x7fffffffe538 --> 0x0
0016| 0x7fffffffe540 --> 0x5555555551d0 (<__libc_csu_init>:      push   r15)
0024| 0x7fffffffe548 --> 0x7ffff7e1de0b (<__libc_start_main+235>:       mov    edi,eax)
0032| 0x7fffffffe550 --> 0x0
0040| 0x7fffffffe558 --> 0x7fffffffe628 --> 0x7fffffffe857 ("/media/sf_SEEDVM/rev1/trytry")
0048| 0x7fffffffe560 --> 0x100040000
0056| 0x7fffffffe568 --> 0x55555555516f (<main>:        push   rbp)
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 1, main () at ./trytry.c:11
11          scanf("%d%d", &x, &y);
gdb-peda$
```

trytry

```
Breakpoint 1, main () at ./trytry.c:11
11      scanf("%d%d", &x, &y);
gdb-peda$ n
123 321
[----------------------------------registers-----
RAX: 0x2
RBX: 0x0
RCX: 0x0
RDX: 0x0
RSI: 0x0
RDI: 0x7fffffffdff0 --> 0x313233 ('321')
RBP: 0x7fffffffe540 --> 0x5555555551d0 (<__libc_csu_init>:      push   r15)
RSP: 0x7fffffffe530 --> 0x141ffffe620
RIP: 0x555555555193 (<main+36>: mov    edx,DWORD PTR [rbp-0xc])
R8 : 0x1999999999999999
R9 : 0x0
R10: 0x7
R11: 0x7ffff7f683c0 --> 0x2000200020002
R12: 0x555555555060 (<_start>:  xor    ebp,ebp)
R13: 0x7fffffffe620 --> 0x1
R14: 0x0
R15: 0x0
EFLAGS: 0x206 (carry PARITY adjust zero sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x555555555182 <main+19>:    lea    rdi,[rip+0xe87]        # 0x555555556010
   0x555555555189 <main+26>:    mov    eax,0x0
   0x55555555518e <main+31>:    call   0x555555555040 <__isoc99_scanf@plt>
=> 0x555555555193 <main+36>:    mov    edx,DWORD PTR [rbp-0xc]
   0x555555555196 <main+39>:    mov    eax,DWORD PTR [rbp-0x8]
   0x555555555199 <main+42>:    mov    esi,edx
   0x55555555519b <main+44>:    mov    edi,eax
   0x55555555519d <main+46>:    call   0x555555555145 <foo>
[--------------------------------------stack-------------------------------------]
0000| 0x7fffffffe530 --> 0x141ffffe620
0008| 0x7fffffffe538 --> 0x7b ('{')
0016| 0x7fffffffe540 --> 0x5555555551d0 (<__libc_csu_init>:      push   r15)
0024| 0x7fffffffe548 --> 0x7ffff7e1de0b (<__libc_start_main+235>:       mov    edi,eax)
0032| 0x7fffffffe550 --> 0x0
0040| 0x7fffffffe558 --> 0x7fffffffe628 --> 0x7fffffffe857 ("/media/sf_SEEDVM/rev1/trytry")
0048| 0x7fffffffe560 --> 0x100040000
0056| 0x7fffffffe568 --> 0x55555555516f (<main>:        push   rbp)
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
12          int out = foo(x, y);
gdb-peda$ print $rbp
$1 = (void *) 0x7fffffffe540
gdb-peda$
```

```
(gdb) x/x &gx          print gx in hex
(gdb) x/4wx &main      print 4 longs at start of \fImain\fR in hex
(gdb) x/gf &gd1        print double
```

```
gdb-peda$ x/x 0x7fffffffe534
0x7fffffffe534: 0x0000007b00000141
gdb-peda$ x/x 0x7fffffffe538
0x7fffffffe538: 0x000000000000007b
```

trytry

- ni 直到 call foo 時
- si 進入 foo

```
gdb-peda$ ni
[----------------------------------registers-----------------------------------]
RAX: 0x7b ('{')
RBX: 0x0
RCX: 0x0
RDX: 0x141
RSI: 0x141
RDI: 0x7b ('{')
RBP: 0x7fffffffe540 —→ 0x555555551d0 (<__libc_csu_init>:      push   r15)
RSP: 0x7fffffffe530 —→ 0x141fffe620
RIP: 0x55555555519d (<main+46>: call   0x555555555145 <foo>)
R8 : 0x1999999999999999
R9 : 0x0
R10: 0x7
R11: 0x7ffff7f683c0 —→ 0x2000200020002
R12: 0x555555555060 (<_start>:  xor    ebp,ebp)
R13: 0x7fffffffe620 —→ 0x1
R14: 0x0
R15: 0x0
EFLAGS: 0x206 (carry PARITY adjust zero sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x555555555196 <main+39>:    mov    eax,DWORD PTR [rbp-0x8]
   0x555555555199 <main+42>:    mov    esi,edx
   0x55555555519b <main+44>:    mov    edi,eax
⇒ 0x55555555519d <main+46>:    call   0x555555555145 <foo>
   0x5555555551a2 <main+51>:    mov    DWORD PTR [rbp-0x4],eax
   0x5555555551a5 <main+54>:    cmp    DWORD PTR [rbp-0x4],0xdeadbeef
   0x5555555551ac <main+61>:    jne    0x5555555551ba <main+75>
   0x5555555551ae <main+63>:    lea    rdi,[rip+0xe60]        # 0x555555556015
Guessed arguments:
arg[0]: 0x7b ('{')
arg[1]: 0x141
arg[2]: 0x141
[-------------------------------------stack-------------------------------------]
0000| 0x7fffffffe530 —→ 0x141fffe620
0008| 0x7fffffffe538 —→ 0x7b ('{')
0016| 0x7fffffffe540 —→ 0x5555555551d0 (<__libc_csu_init>:      push   r15)
0024| 0x7fffffffe548 —→ 0x7ffff7e1de0b (<__libc_start_main+235>:       mov    edi,eax)
0032| 0x7fffffffe550 —→ 0x0
0040| 0x7fffffffe558 —→ 0x7fffffffe628 —→ 0x7fffffffe857 ("/media/sf_SEEDVM/rev1/trytry")
0048| 0x7fffffffe560 —→ 0x100040000
0056| 0x7fffffffe568 —→ 0x55555555516f (<main>:        push   rbp)
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
0x000055555555519d      12          int out = foo(x, y);
gdb-peda$ si
```

trytry

```
Legend: code, data, rodata, value
0x000055555555519d        12        int out = foo(x, y);
gdb-peda$ si
[----------------------------------registers-----------------------------------]
RAX: 0x7b ('{')
RBX: 0x0
RCX: 0x0
RDX: 0x141
RSI: 0x141
RDI: 0x7b ('{')
RBP: 0x7fffffffe540 --> 0x5555555551d0 (<__libc_csu_init>:      push   r15)
RSP: 0x7fffffffe528 --> 0x5555555551a2 (<main+51>:      mov    DWORD PTR [rbp-0x4],eax)
RIP: 0x555555555145 (<foo>:     push   rbp)
R8 : 0x1999999999999999
R9 : 0x0
R10: 0x7
R11: 0x7ffff7f683c0 --> 0x2000200020002
R12: 0x555555555060 (<_start>:  xor    ebp,ebp)
R13: 0x7fffffffe620 --> 0x1
R14: 0x0
R15: 0x0
EFLAGS: 0x206 (carry PARITY adjust zero sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x555555555138 <__do_global_dtors_aux+56>:   ret
   0x555555555139 <__do_global_dtors_aux+57>:   nop    DWORD PTR [rax+0x0]
   0x555555555140 <frame_dummy>:        jmp    0x5555555550c0 <register_tm_clones>
=> 0x555555555145 <foo>:        push   rbp
   0x555555555146 <foo+1>:      mov    rbp,rsp
   0x555555555149 <foo+4>:      sub    rsp,0x20
   0x55555555514d <foo+8>:      mov    DWORD PTR [rbp-0x14],edi
   0x555555555150 <foo+11>:     mov    DWORD PTR [rbp-0x18],esi
[--------------------------------------stack-------------------------------------]
0000| 0x7fffffffe528 --> 0x5555555551a2 (<main+51>:     mov    DWORD PTR [rbp-0x4],eax)
0008| 0x7fffffffe530 --> 0x141ffffe620
0016| 0x7fffffffe538 --> 0x7b ('{')
0024| 0x7fffffffe540 --> 0x5555555551d0 (<__libc_csu_init>:     push   r15)
0032| 0x7fffffffe548 --> 0x7ffff7e1de0b (<__libc_start_main+235>:       mov    edi,eax)
0040| 0x7fffffffe550 --> 0x0
0048| 0x7fffffffe558 --> 0x7fffffffe628 --> 0x7fffffffe857 ("/media/sf_SEEDVM/rev1/trytry")
0056| 0x7fffffffe560 --> 0x100040000
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
foo (a=0x5555, b=0x55555215) at ./trytry.c:3
3       int foo(int a, int b) {
gdb-peda$
```

trytry

- 更改 return value

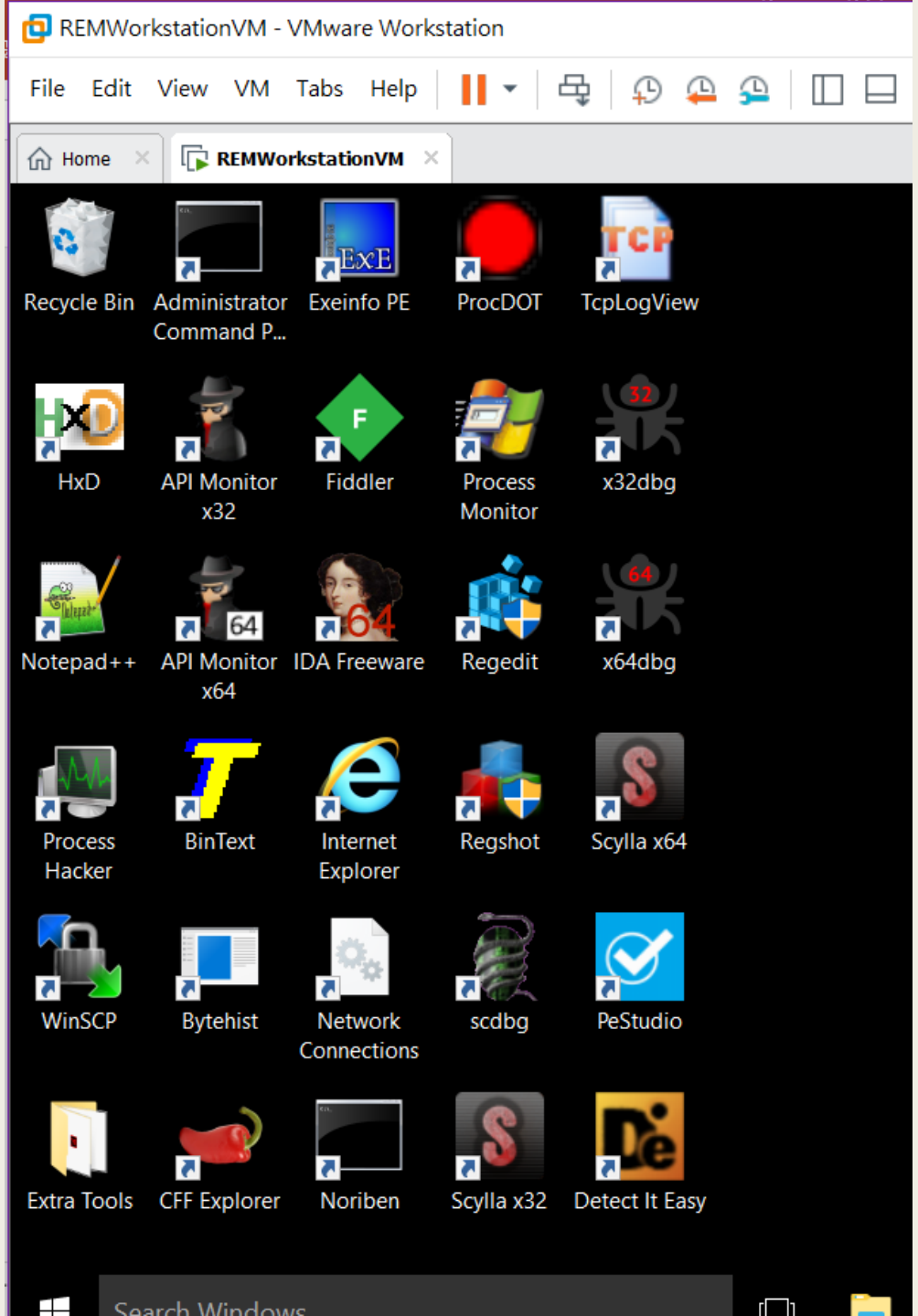
```c
#include <stdio.h>

int foo(int a, int b) {
    puts("this is foo");
    int c = a + b;
    return c;
}

int main() {
    int x, y;
    scanf("%d%d", &x, &y);
    int out = foo(x, y);
    if (out == 0xdeadbeef) {
        puts("this is 0xdeadbeef");
    }
    return 0;
}
```

```
gdb-peda$ ni
[----------------------------------registers-----------------------------------]
RAX: 0x1bc
RBX: 0x0
RCX: 0x7ffff7ee5643 (<__GI___libc_write+19>:    cmp    rax,0xfffffffffffff000)
RDX: 0x7b ('{')
RSI: 0x5555555596b0 ("this is foo\n")
RDI: 0x7ffff7fb64c0 --> 0x0
RBP: 0x7fffffffe520 --> 0x7fffffffe540 --> 0x5555555551d0 (<__libc_csu_init>:   push   r15)
RSP: 0x7fffffffe500 --> 0x0
RIP: 0x55555555516d (<foo+40>:  leave)
R8 : 0xc ('\x0c')
R9 : 0x40 ('@')
R10: 0x410
R11: 0x246
R12: 0x555555555060 (<_start>:  xor    ebp,ebp)
R13: 0x7fffffffe620 --> 0x1
R14: 0x0
R15: 0x0
EFLAGS: 0x202 (carry parity adjust zero sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x555555555165 <foo+32>:     add    eax,edx
   0x555555555167 <foo+34>:     mov    DWORD PTR [rbp-0x4],eax
   0x55555555516a <foo+37>:     mov    eax,DWORD PTR [rbp-0x4]
=> 0x55555555516d <foo+40>:     leave
   0x55555555516e <foo+41>:     ret
   0x55555555516f <main>:       push   rbp
   0x555555555170 <main+1>:     mov    rbp,rsp
   0x555555555173 <main+4>:     sub    rsp,0x10
[-----------------------------------stack------------------------------------]
0000| 0x7fffffffe500 --> 0x0
0008| 0x7fffffffe508 --> 0x7b00000141
0016| 0x7fffffffe510 --> 0x0
0024| 0x7fffffffe518 --> 0x1bcf7ffe190
0032| 0x7fffffffe520 --> 0x7fffffffe540 --> 0x5555555551d0 (<__libc_csu_init>:  push   r15)
0040| 0x7fffffffe528 --> 0x5555555551a2 (<main+51>:     mov    DWORD PTR [rbp-0x4],eax)
0048| 0x7fffffffe530 --> 0x141ffffe620
0056| 0x7fffffffe538 --> 0x7b ('{')
[----------------------------------------------------------------------------]
Legend: code, data, rodata, value
7       }
gdb-peda$ print $eax
$4 = 0x1bc
gdb-peda$ set $eax = 0xdeadbeef
gdb-peda$ c
Continuing.
this is 0xdeadbeef
```

Basic Tools

先安裝 VMWare 在電腦中
Windows 版:
https://www.vmware.com/tw/products/workstation-pro.html
Mac 版:
https://www.vmware.com/tw/products/fusion/fusion-evaluation.html

之後再到
https://drive.google.com/file/d/1ZSIG2iz1sOnSnOd4_JpWJDhVVoGtTAC9/view?usp=sharing
下載 VM 檔案，解壓縮後將
RemWorkStationVM.vmx 匯入 VMWare 中即可

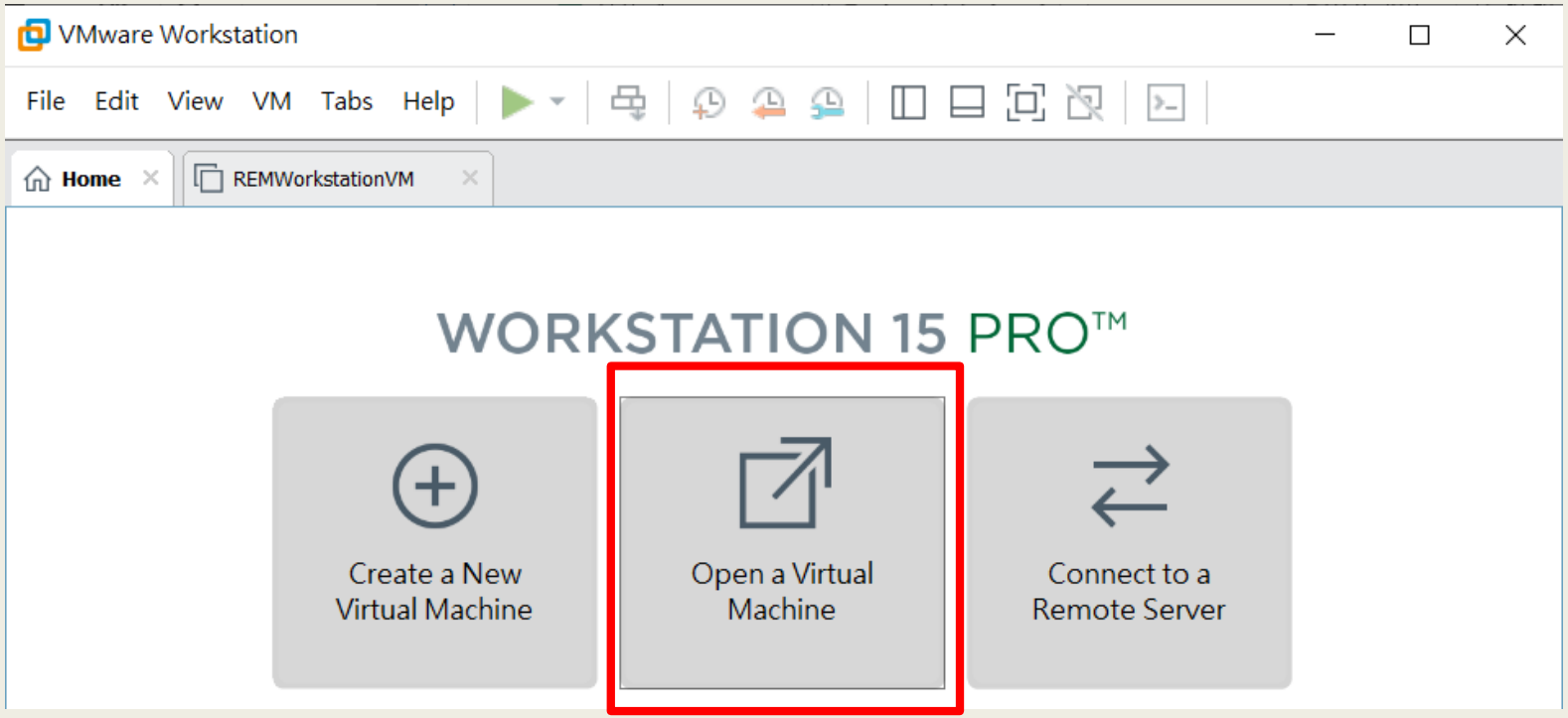


Import Virtual Machine

- In Mac, you can click the RemWorkStationVM.vmx to import it into VMWare.

Import Virtual Machine

- In Windows, you can click the Open a Virtual Machine and choose the RemWorkStationVM.vmx to import it.



HW

- http://reversing.kr/challenge.php
- 挑 3 題來寫
- 上傳 writeup.pdf