System Security - Attack and Defense for Binaries

CS 4390/5390, Spring *2026*

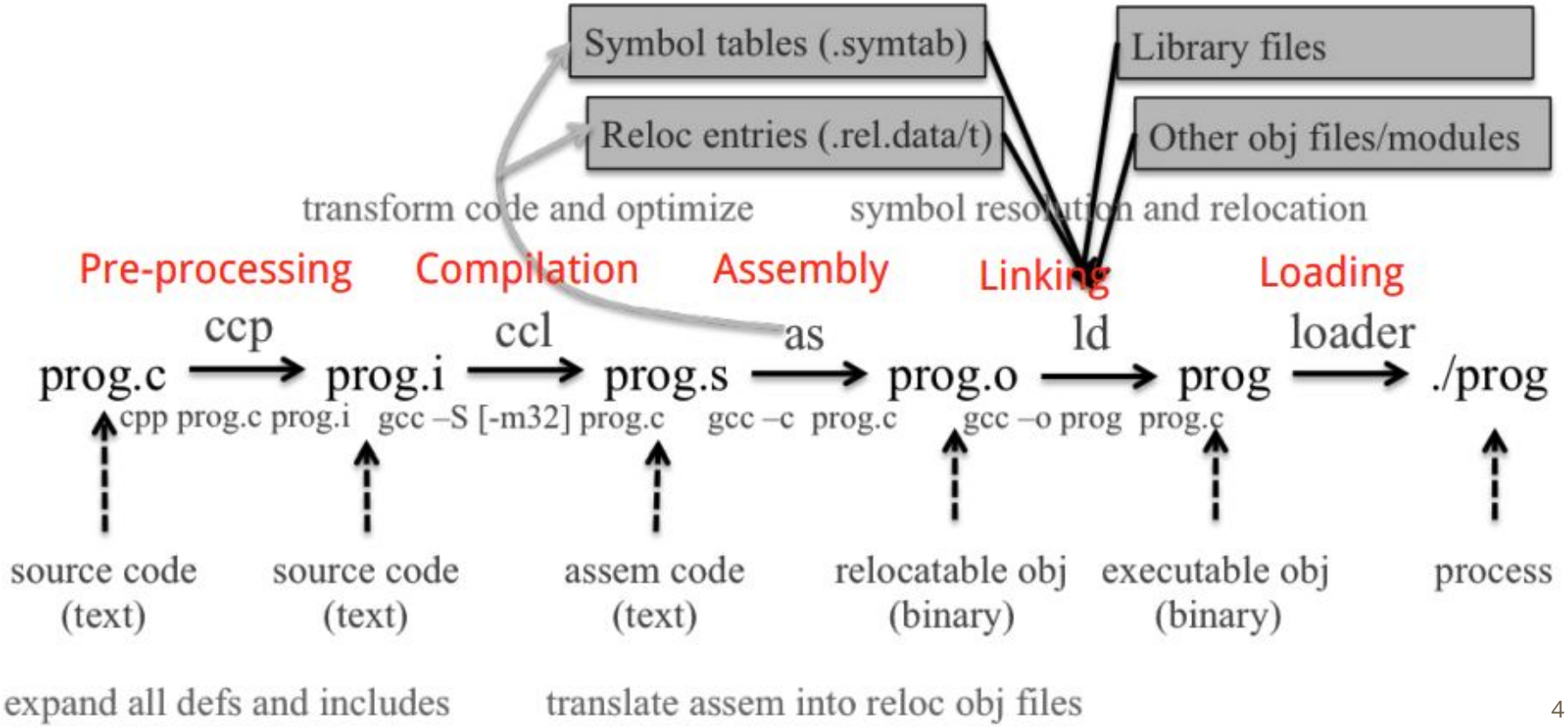
Instructor: MD Armanuzzaman (***Arman***)

Agenda

- Background knowledge
  - Compiler, linker, loader
  - x86 and x86-64 architectures and ISA
  - Linux fundamentals
    - Linux file permissions
    - Set-UID programs
    - Memory map of a Linux process
    - System calls
    - Environment and Shell variables
    - ELF files
    - Reverse engineering tools

Background Knowledge: Compiler, linker, and loader

From a C program to a process



A Shell in a Nutshell

```
int pid = fork();
if (pid == 0) {
        // I am the child process
        exec("ls");
        }
else if (pid == -1) {
        // fork failed
    }
else {
        // I am the parent; continue my business being a cool program
        // I could wait for the child to finish if I want
    }
```

https://github.com/kamalmarhubi/shell-workshop

Loading and Executing a Binary Program on Linux

Validation (permissions, memory requirements etc.)

Operating system starts by setting up a new process for the program to run in, including a virtual address space.

The operating system maps an interpreter into the process's virtual memory.

Interpreter, e.g., /lib/ld-linux.so in Linux

The interpreter loads the binary into its virtual address space (the same space in which the interpreter is loaded).

It then parses the binary to find out (among other things) which dynamic libraries the binary uses.

The interpreter maps these into the virtual address space (using mmap or an equivalent function) and then performs any necessary last-minute relocations in the binary's code sections to fill in the correct addresses for references to the dynamic libraries.

1. Copying the command-line arguments on the stack
2. Initializing registers (e.g., the stack pointer)
3. Jumping to the program entry point (_start)

Compiling a C program behind the scene (add_32 add_64)

add.c

```
#include "add.h"
#define BASE 50

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

add.h

```
#ifndef ADD_H
#define ADD_H

int add(int, int);

#endif
```

```
gcc -Wall -save-temps -P -m32 -O2 add.c main.c -o add_32
```

```
gcc -Wall -save-temps -P -O2 add.c main.c -o add_64
```

main.c

```
/* This program has an integer overflow vulnerability. */
#include "add.h"
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define USAGE "Add two integers with 50. Usage: add a b\n"

int main(int argc, char *argv[])
{
        int a = 0;
        int b = 0;

        if (argc != 3)
        {
                printf(USAGE);
                return 0;  }
        a = atoi(argv[1]);
        b = atoi(argv[2]);
        printf("%d + %d + 50 = %d\n", a, b, add(a, b));
}
```

Background Knowledge: x86 architecture

Data Types

There are 5 integer data types:

Byte – 8 bits.

Word – 16 bits.

Dword, Doubleword – 32 bits.

Quadword – 64 bits.

Double quadword – 128 bits.

Endianness

- Little Endian (Intel, ARM)

  Least significant byte has lowest address

  Dword address: 0x0

  Value: 0x78563412

- Big Endian

  Least significant byte has highest address

  Dword address: 0x0

  Value: 0x12345678

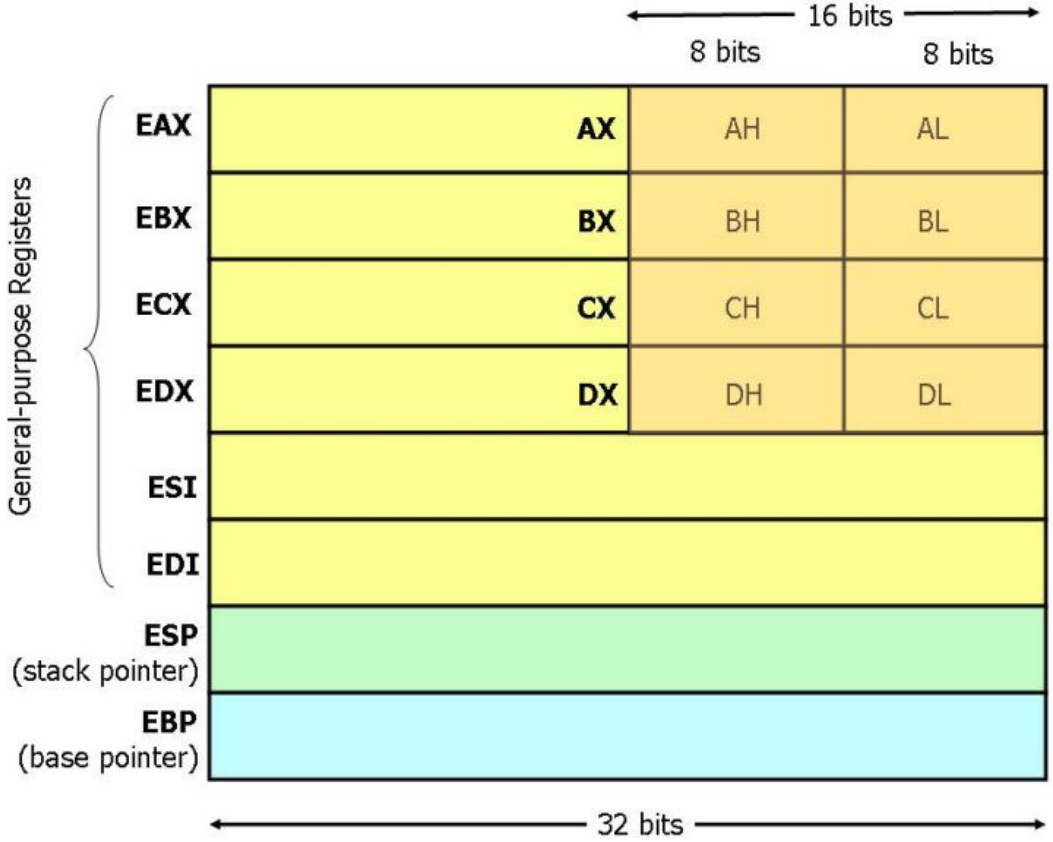
Address 0	0x12
Address 1	0x34
Address 2	0x56
Address 3	0x78

Base Registers

There are

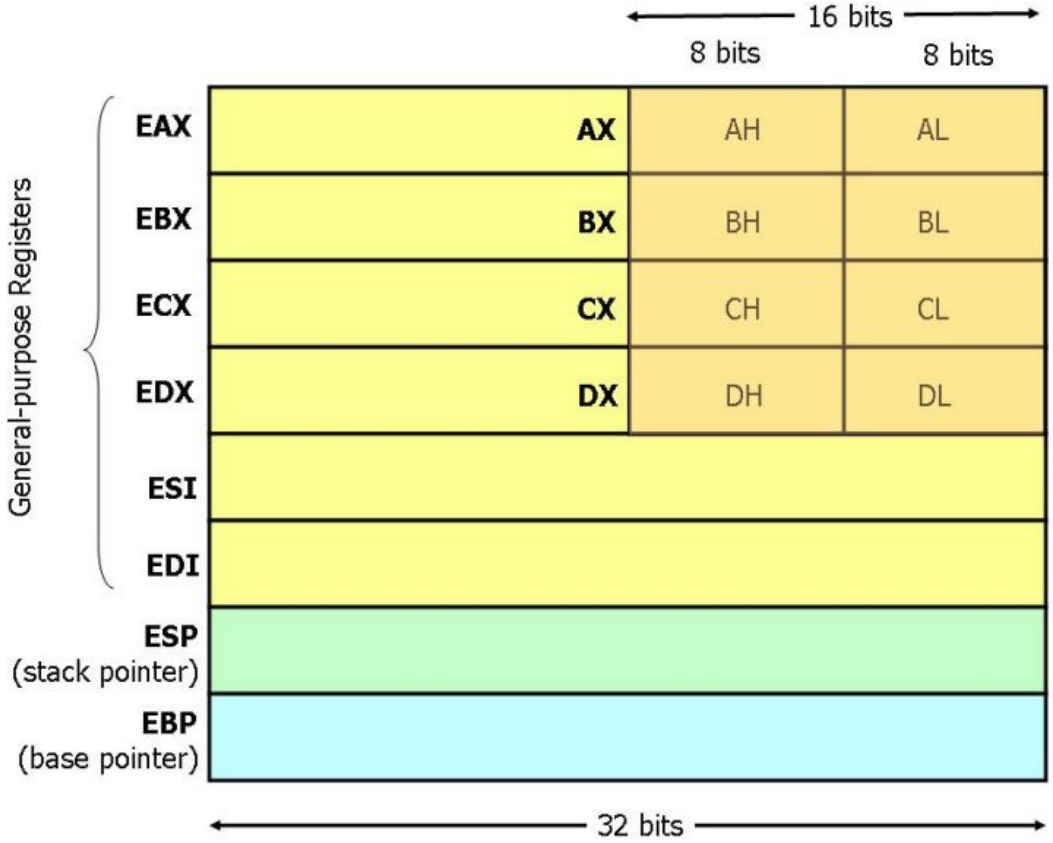
- Eight 32-bit “general-purpose” registers,
- One 32-bit EFLAGS register,
- One 32-bit instruction pointer register (eip), and
- Other special-purpose registers.

The General-Purpose Registers



- 8 general-purpose registers
- esp is the stack pointer
- ebp is the base pointer
- esi and edi are source and destination index registers for array and string operations

The General-Purpose Registers



- The registers eax, ebx, ecx, and edx may be accessed as 32-bit, 16-bit, or 8-bit registers.
- The other four registers can be accessed as 32-bit or 16-bit.

EFLAGS Register

The various bits of the 32-bit EFLAGS register are set (1) or reset/clear (0) according to the results of certain operations.

We will be interested in, at most, the bits

CF – carry flag

PF – parity flag

ZF – zero flag

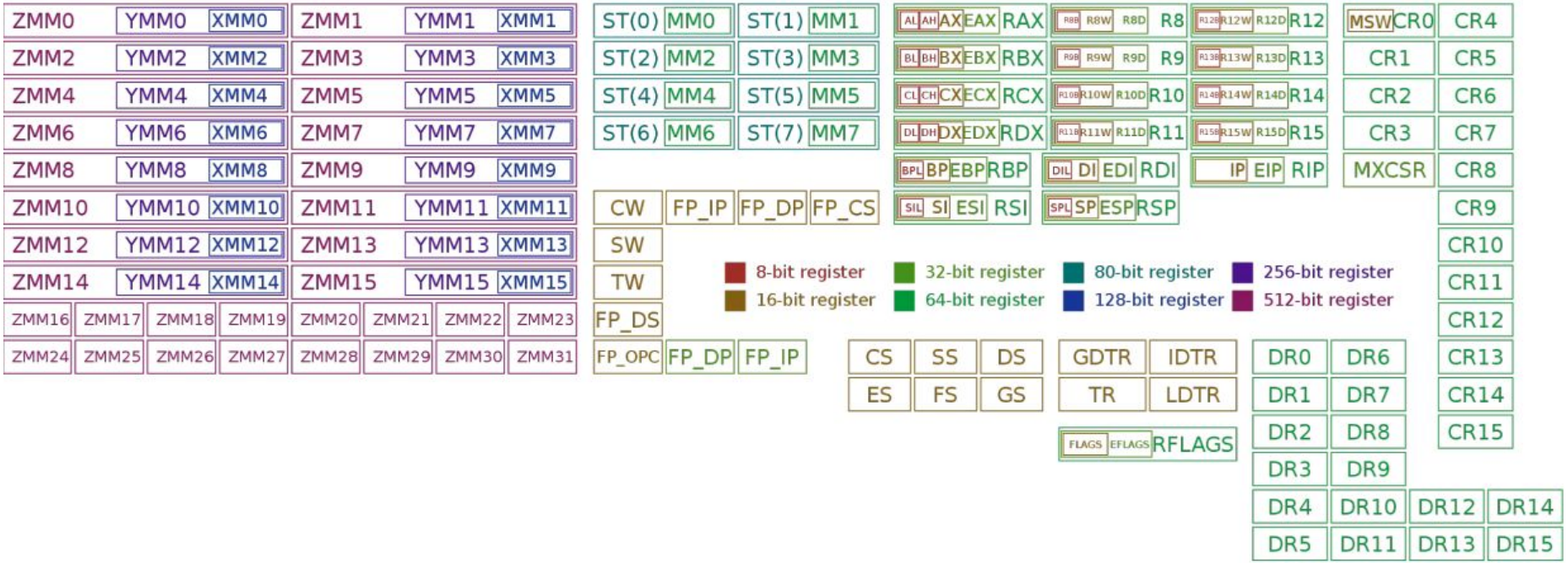
SF – sign flag

Instruction Pointer (EIP)

Finally, there is the EIP register, which is the instruction pointer (program counter).

Register EIP holds the address of the **next** instruction to be executed.

Registers on x86 and amd64



https://en.wikipedia.org/wiki/X86

Instructions

Each instruction is of the form

label: mnemonic operand1, operand2, operand3

The label is optional.

The number of operands is 0, 1, 2, or 3, depending on the mnemonic .

Each operand is either

- An immediate value,
- A register, or
- A memory address.

Source and Destination Operands

Each operand is either a source operand or a destination operand.

A source operand, in general, may be

- An immediate value,
- A register, or
- A memory address.

A destination operand, in general, may be

- A register, or
- A memory address.

Instructions

hlt – 0 operands

halts the central processing unit (CPU) until the next external interrupt is fired

inc – 1 operand; inc <reg>, inc <mem>

add – 2 operands; add <reg>,<reg>

imul – 1, 2, or 3 operands; imul <reg32>,<reg32>,<con>

In Intel syntax the first operand is the destination

Intel Syntax Assembly and Disassembly

Machine instructions generally fall into three categories: data movement, arithmetic/logic, and control-flow.

<reg32> Any 32-bit register (eax, ebx, ecx, edx, esi, edi, esp, or ebp)

<reg16> Any 16-bit register (ax, bx, cx, or dx)

<reg8> Any 8-bit register (ah, bh, ch, dh, al, bl, cl, or dl)

<reg> Any register

<mem> A memory address (e.g., [eax] or [eax + ebx*4]); [] square brackets

<con32> Any 32-bit immediate

<con16> Any 16-bit immediate

<con8> Any 8-bit immediate

<con> Any 8-, 16-, or 32-bit immediate

Addressing Memory

Move from source (operand 2) to destination (operand 1)

Square bracket [] represents memory location.

mov [eax], ebx Copy 4 bytes from register EBX into memory address specified in EAX.

mov eax, [esi - 4] Move 4 bytes at memory address ESI - 4 into EAX.

mov [esi + eax * 1], cl Move the contents of CL into the byte at address ESI+EAX*1.

mov edx, [esi + ebx*4] Move the 4 bytes of data at address ESI+4*EBX into EDX.

Addressing Memory

The size directives **BYTE PTR**, **WORD PTR**, and **DWORD PTR** serve this purpose, indicating sizes of **1, 2,** and **4** bytes respectively.

mov [ebx], 2 isn't this ambiguous? We can have a default.

mov BYTE PTR [ebx], 2 Move 2 into the single byte at the address stored in EBX.

mov WORD PTR [ebx], 2 Move the 16-bit integer representation of 2 into the 2 bytes starting at the address in EBX.

mov DWORD PTR [ebx], 2 Move the 32-bit integer representation of 2 into the 4 bytes starting at the address in EBX.

Data Movement Instructions

mov — Move

Syntax

mov <reg>, <reg>
mov <reg>, <mem>
mov <mem>, <reg>
mov <reg>, <con>
mov <mem>, <con>

Examples

mov eax, ebx — copy the value in EBX into EAX
mov byte ptr [var], 5 — store the value 5 into the byte at location var

Data Movement Instructions

push — Push on stack; **decrements** ESP by 4, then places the operand at the location ESP points to.

Syntax

push <reg32>
push <mem>
push <con32>

Examples

push eax — push eax on the stack
push [var] — push the 4 bytes at address var onto the stack

Data Movement Instructions

pop — Pop from stack

Syntax

pop <reg32>
pop <mem>

Examples

pop edi — pop the top element of the stack into EDI.
push [var] — pop the top element of the stack into memory at the four bytes starting at location EBX.

LEA Instructions

lea — Load effective address; used for quick calculation

Syntax

lea <reg32>, <mem>

Examples

Lea edi, [ebx+4*esi] — the quantity EBX+4*ESI is placed in EDI.

Arithmetic and Logic Instructions

add eax, 10 — EAX is set to EAX + 10
addb byte ptr [eax], 10 — add 10 to the single byte stored at memory address stored in EAX

sub al, ah — AL is set to AL - AH
sub eax, 216 — subtract 216 from the value stored in EAX

dec eax — subtract one from the contents of EAX

imul eax, [ebx] — multiply the contents of EAX by the 32-bit contents of the memory at location EBX. Store the result in EAX.

shr ebx, cl — Store in EBX the floor of result of dividing the value of EBX by $\mathbf{2^n}$ where n is the value in CL.

Control Flow Instructions

jmp — Jump

Transfers program control flow to the instruction at the memory location indicated by the operand.

Syntax

```
jmp <label> # direct jump
jmp <reg32> # indirect jump
```

Examples

jmp begin — Jump to the instruction labeled begin.

Control Flow Instructions

jcondition — Conditional jump

Syntax

je <label> (jump when equal)
jne <label> (jump when not equal)
jz <label> (jump when last result was zero)
jg <label> (jump when greater than)
jge <label> (jump when greater than or equal to)
jl <label> (jump when less than)
jle <label> (jump when less than or equal to)

Examples

cmp ebx, eax
jle done

Control Flow Instructions

cmp — Compare

Syntax

cmp <reg>, <reg>
cmp <mem>, <reg>
cmp <reg>, <mem>
cmp <con>, <reg>

Examples

```
cmp byte ptr [ebx], 10
jeq loop
```

If the byte stored at the memory location in EBX is equal to the integer constant 10, jump to the location labeled loop.

Control Flow Instructions

call — Subroutine call

The call instruction first **pushes the current code location onto the hardware supported stack** in memory, and then performs **an unconditional jump to the code** location indicated by the label operand. *Unlike the simple jump instructions, the call instruction saves the location to return to when the subroutine completes.*

Syntax

call <label>

call <reg32>

call <mem>

Control Flow Instructions

ret — Subroutine return

The ret instruction implements a subroutine return mechanism. This instruction pops a code location off the hardware supported in-memory stack to the program counter.

Syntax

```
ret
```

The Run-time Stack

The run-time stack supports procedure calls and the passing of parameters between procedures.

The stack is located in memory.

The stack grows towards **low memory.**

When we push a value, esp is decremented.

When we pop a value, esp is incremented.

Stack Instructions

enter — Create a function frame

Equivalent to:

push ebp

mov ebp, esp

sub esp, Imm

Stack Instructions

leave — Releases the function frame set up by an earlier ENTER instruction.

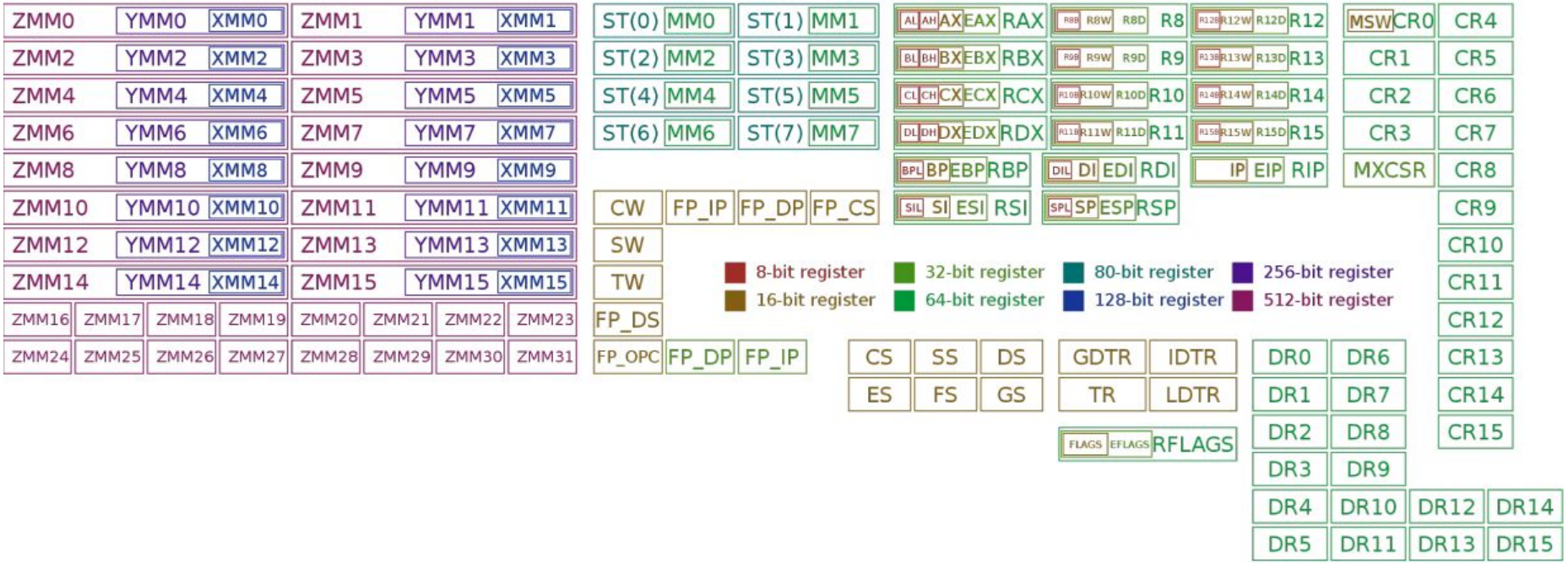
Equivalent to:

mov esp, ebp

pop ebp

Background Knowledge: x86-64/amd64 architecture

Registers on x86 and amd64



https://en.wikipedia.org/wiki/X86

x86 vs. x86-64 (code/ladd)

main.c

```
/*
This program has an integer overflow vulnerability.
 */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
long long ladd(long long *xp, long long y)
{
  long long t = *xp + y;
  return t;
}
```

```
int main(int argc, char *argv[])
{
  long long a = 0;
  long long b = 0;

  if (argc != 3)
  {
   printf("Usage: ladd a b\n");
   return 0;
  }
  printf("The sizeof(long long) is %d\n", sizeof(long  long));
  a = atoll(argv[1]);
  b = atoll(argv[2]);
  printf("%lld + %lld = %lld\n", a, b, ladd(&a, b));
}
```

```
gcc -Wall -m32 -O2 main.c -o ladd
```

```
gcc -Wall -O2 main.c -o ladd64
```

x86 vs. x86-64 (code/ladd)

x86

```
000012c0 <ladd>:
    12c0:    f3 0f 1e fb          endbr32
    12c4:    8b 44 24 04          mov    0x4(%esp),%eax
    12c8:    8b 50 04             mov    0x4(%eax),%edx
    12cb:    8b 00                mov    (%eax),%eax
    12cd:    03 44 24 08          add    0x8(%esp),%eax
    12d1:    13 54 24 0c          adc    0xc(%esp),%edx
    12d5:    c3                   ret
```

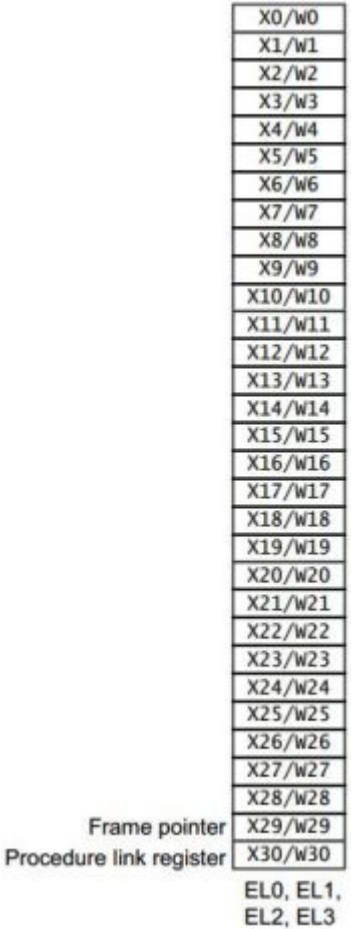
x86-64

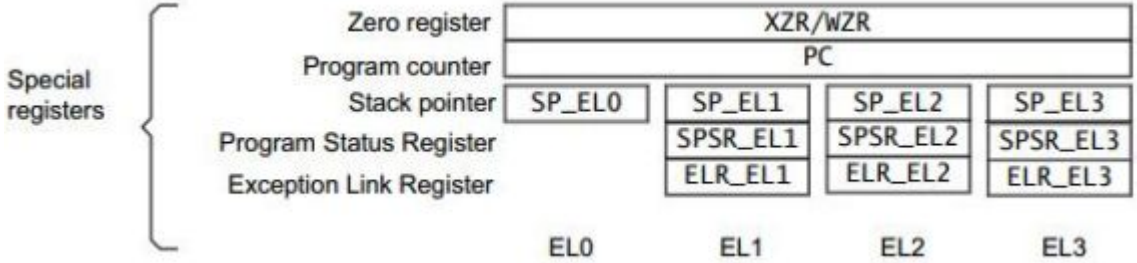
```
0000000000001220 <ladd>:
    1220: f3 0f 1e fa endbr64
    1224: 48 8b 07 mov rax,QWORD PTR [rdi]
    1227: 48 01 f0 add rax,rsi
    122a: c3 ret
```

```
objdump -M intel -d ladd_32
objdump -M intel -d ladd_64
```

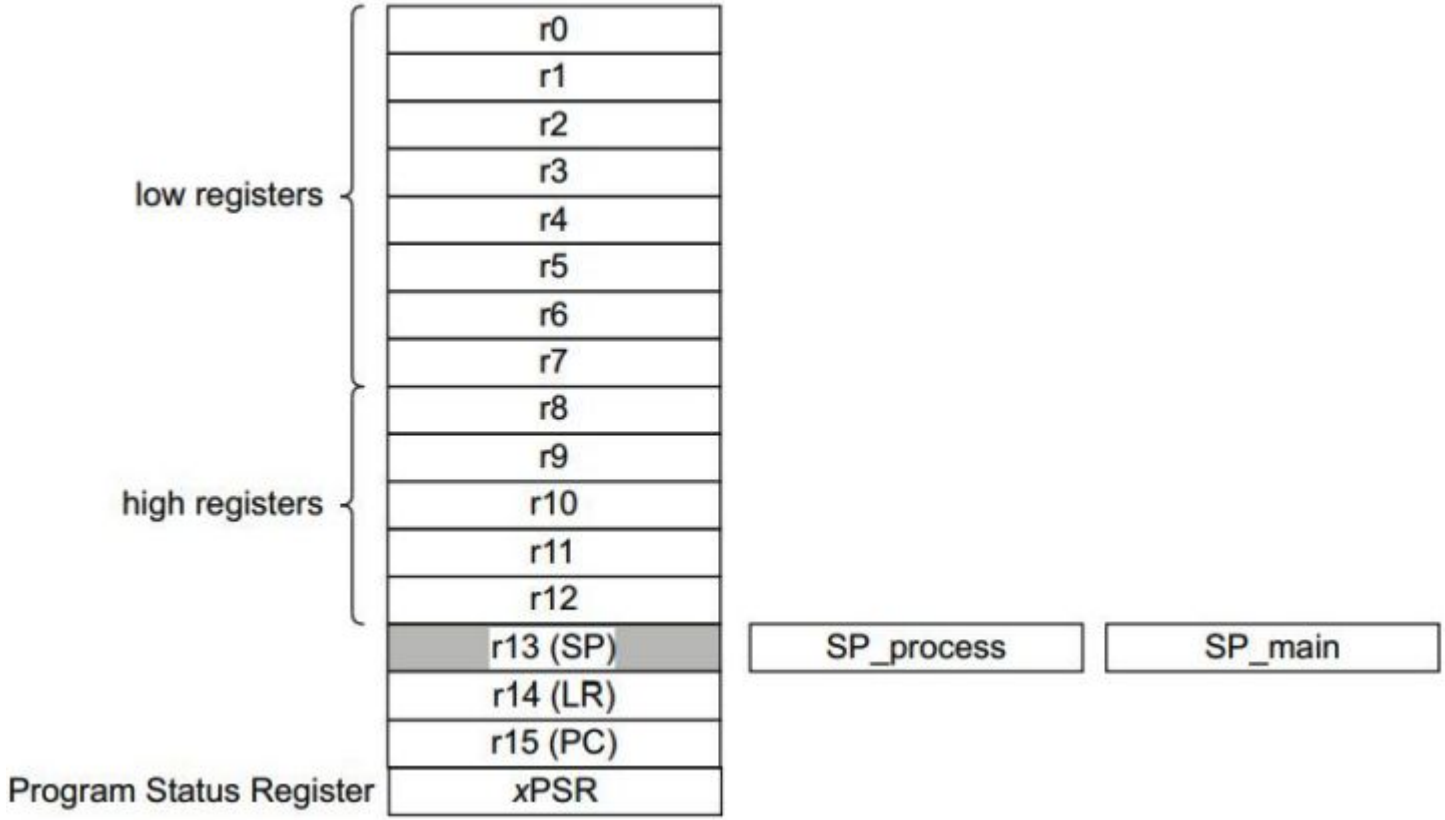
Background Knowledge: ARM Cortex-A/M Architecture

Cortex-A 64 bit





Cortex-M 32 bit



Background Knowledge: Linux File Permissions

Permission Groups

Each file and directory has three user-based permission groups:

Owner – A user is the owner of the file. By default, the person who created a file becomes its owner. The Owner permissions apply only the owner of the file or directory

Group – A group can contain multiple users. All users belonging to a group will have the same access permissions to the file. The Group permissions apply only to the group that has been assigned to the file or directory

Others – The others permissions apply to all other users on the system.

Permission Types

Each file or directory has three basic permission types defined for all the 3 user types:

Read – The Read permission refers to a user's capability to read the contents of the file.

Write – The Write permissions refer to a user's capability to write or modify a file or directory.

Execute – The Execute permission affects a user's capability to execute a file or view the contents of a directory.

File type

First field in the output is file type. If the there is a – it means it is a plain file. If there is d it means it is a directory, c represents a character device, b represents a block device.

```
arman@aserver:~/STC/software-security$ ls -l
total 80
-rwxrwxr-x  1 arman arman 1401 Jan 13 18:09 auth.py
drwxrwxr-x 15 arman arman 4096 Jan 13 18:09 challenges
drwxrwxr-x  3 arman arman 4096 Jan 13 18:09 conf
-rw-rw-r--  1 arman arman  264 Jan 14 02:18 config.env
drwxrwxr-x  7 arman arman 4096 Jan 13 18:09 containers
drwxrwxr-x  7 arman arman 4096 Jan 13 18:09 CTFd
drwxrwxr-x  3 arman arman 4096 Jan 13 18:09 CTFd_plugin
-rw-rw-r--  1 arman arman 2526 Jan 17 23:24 docker-compose.yml
-rwxrwxr-x  1 arman arman  771 Jan 13 18:09 enter.py
drwxrwxr-x  2 arman arman 4096 Jan 13 18:09 home_daemon
-rw-rw-r--  1 arman arman 3074 Jan 13 18:10 README.md
-rwxrwxr-x  1 arman arman 3003 Jan 13 18:31 resetup.sh
-rwxrwxr-x  1 arman arman 1373 Jan 13 18:09 restart.sh
-rwxrwxr-x  1 arman arman 1211 Jan 14 02:07 run.sh
drwxrwxr-x  2 arman arman 4096 Jan 13 18:09 scripts
-rw-rw-r--  1 arman arman 1986 Jan 13 18:09 script.sh
-rwxrwxr-x  1 arman arman 2996 Jan 14 01:40 setup.sh
-rw-rw-r--  1 arman arman 7196 Jan 13 18:09 tips.md
-rw-rw-r--  1 arman arman    0 Jan 13 18:09 todo-list.md
-rwxrwxr-x  1 arman arman  830 Jan 13 18:09 update.sh
```

Permissions for owner, group, and others

```
arman@aserver:~/STC/software-security$ ls -l
total 80
-rwxrwxr-x  1 arman arman 1401 Jan 13 18:09 auth.py
drwxrwxr-x 15 arman arman 4096 Jan 13 18:09 challenges
drwxrwxr-x  3 arman arman 4096 Jan 13 18:09 conf
-rw-rw-r--  1 arman arman  264 Jan 14 02:18 config.env
drwxrwxr-x  7 arman arman 4096 Jan 13 18:09 containers
drwxrwxr-x  7 arman arman 4096 Jan 13 18:09 CTFd
drwxrwxr-x  3 arman arman 4096 Jan 13 18:09 CTFd_plugin
-rw-rw-r--  1 arman arman 2526 Jan 17 23:24 docker-compose.yml
-rwxrwxr-x  1 arman arman  771 Jan 13 18:09 enter.py
drwxrwxr-x  2 arman arman 4096 Jan 13 18:09 home_daemon
-rw-rw-r--  1 arman arman 3074 Jan 13 18:10 README.md
-rwxrwxr-x  1 arman arman 3003 Jan 13 18:31 resetup.sh
-rwxrwxr-x  1 arman arman 1373 Jan 13 18:09 restart.sh
-rwxrwxr-x  1 arman arman 1211 Jan 14 02:07 run.sh
drwxrwxr-x  2 arman arman 4096 Jan 13 18:09 scripts
-rw-rw-r--  1 arman arman 1986 Jan 13 18:09 script.sh
-rwxrwxr-x  1 arman arman 2996 Jan 14 01:40 setup.sh
-rw-rw-r--  1 arman arman 7196 Jan 13 18:09 tips.md
-rw-rw-r--  1 arman arman    0 Jan 13 18:09 todo-list.md
-rwxrwxr-x  1 arman arman  830 Jan 13 18:09 update.sh
```

Link count

```
arman@aserver:~/STC/software-security$ ls -l
total 80
-rwxrwxr-x  1 arman arman 1401 Jan 13 18:09 auth.py
drwxrwxr-x 15 arman arman 4096 Jan 13 18:09 challenges
drwxrwxr-x  3 arman arman 4096 Jan 13 18:09 conf
-rw-rw-r--  1 arman arman  264 Jan 14 02:18 config.env
drwxrwxr-x  7 arman arman 4096 Jan 13 18:09 containers
drwxrwxr-x  7 arman arman 4096 Jan 13 18:09 CTFd
drwxrwxr-x  3 arman arman 4096 Jan 13 18:09 CTFd_plugin
-rw-rw-r--  1 arman arman 2526 Jan 17 23:24 docker-compose.yml
-rwxrwxr-x  1 arman arman  771 Jan 13 18:09 enter.py
drwxrwxr-x  2 arman arman 4096 Jan 13 18:09 home_daemon
-rw-rw-r--  1 arman arman 3074 Jan 13 18:10 README.md
-rwxrwxr-x  1 arman arman 3003 Jan 13 18:31 resetup.sh
-rwxrwxr-x  1 arman arman 1373 Jan 13 18:09 restart.sh
-rwxrwxr-x  1 arman arman 1211 Jan 14 02:07 run.sh
drwxrwxr-x  2 arman arman 4096 Jan 13 18:09 scripts
-rw-rw-r--  1 arman arman 1986 Jan 13 18:09 script.sh
-rwxrwxr-x  1 arman arman 2996 Jan 14 01:40 setup.sh
-rw-rw-r--  1 arman arman 7196 Jan 13 18:09 tips.md
-rw-rw-r--  1 arman arman    0 Jan 13 18:09 todo-list.md
-rwxrwxr-x  1 arman arman  830 Jan 13 18:09 update.sh
```

Owner

This field provide info about the creator of the file.

```
arman@aserver:~/STC/software-security$ ls -l
total 80
-rwxrwxr-x  1 arman arman 1401 Jan 13 18:09 auth.py
drwxrwxr-x 15 arman arman 4096 Jan 13 18:09 challenges
drwxrwxr-x  3 arman arman 4096 Jan 13 18:09 conf
-rw-rw-r--  1 arman arman  264 Jan 14 02:18 config.env
drwxrwxr-x  7 arman arman 4096 Jan 13 18:09 containers
drwxrwxr-x  7 arman arman 4096 Jan 13 18:09 CTFd
drwxrwxr-x  3 arman arman 4096 Jan 13 18:09 CTFd_plugin
-rw-rw-r--  1 arman arman 2526 Jan 17 23:24 docker-compose.yml
-rwxrwxr-x  1 arman arman  771 Jan 13 18:09 enter.py
drwxrwxr-x  2 arman arman 4096 Jan 13 18:09 home_daemon
-rw-rw-r--  1 arman arman 3074 Jan 13 18:10 README.md
-rwxrwxr-x  1 arman arman 3003 Jan 13 18:31 resetup.sh
-rwxrwxr-x  1 arman arman 1373 Jan 13 18:09 restart.sh
-rwxrwxr-x  1 arman arman 1211 Jan 14 02:07 run.sh
drwxrwxr-x  2 arman arman 4096 Jan 13 18:09 scripts
-rw-rw-r--  1 arman arman 1986 Jan 13 18:09 script.sh
-rwxrwxr-x  1 arman arman 2996 Jan 14 01:40 setup.sh
-rw-rw-r--  1 arman arman 7196 Jan 13 18:09 tips.md
-rw-rw-r--  1 arman arman    0 Jan 13 18:09 todo-list.md
-rwxrwxr-x  1 arman arman  830 Jan 13 18:09 update.sh
```

Group

```
arman@aserver:~/STC/software-security$ ls -l
total 80
-rwxrwxr-x  1 arman arman 1401 Jan 13 18:09 auth.py
drwxrwxr-x 15 arman arman 4096 Jan 13 18:09 challenges
drwxrwxr-x  3 arman arman 4096 Jan 13 18:09 conf
-rw-rw-r--  1 arman arman  264 Jan 14 02:18 config.env
drwxrwxr-x  7 arman arman 4096 Jan 13 18:09 containers
drwxrwxr-x  7 arman arman 4096 Jan 13 18:09 CTFd
drwxrwxr-x  3 arman arman 4096 Jan 13 18:09 CTFd_plugin
-rw-rw-r--  1 arman arman 2526 Jan 17 23:24 docker-compose.yml
-rwxrwxr-x  1 arman arman  771 Jan 13 18:09 enter.py
drwxrwxr-x  2 arman arman 4096 Jan 13 18:09 home_daemon
-rw-rw-r--  1 arman arman 3074 Jan 13 18:10 README.md
-rwxrwxr-x  1 arman arman 3003 Jan 13 18:31 resetup.sh
-rwxrwxr-x  1 arman arman 1373 Jan 13 18:09 restart.sh
-rwxrwxr-x  1 arman arman 1211 Jan 14 02:07 run.sh
drwxrwxr-x  2 arman arman 4096 Jan 13 18:09 scripts
-rw-rw-r--  1 arman arman 1986 Jan 13 18:09 script.sh
-rwxrwxr-x  1 arman arman 2996 Jan 14 01:40 setup.sh
-rw-rw-r--  1 arman arman 7196 Jan 13 18:09 tips.md
-rw-rw-r--  1 arman arman    0 Jan 13 18:09 todo-list.md
-rwxrwxr-x  1 arman arman  830 Jan 13 18:09 update.sh
```

File size

```
arman@aserver:~/STC/software-security$ ls -l
total 80
-rwxrwxr-x  1 arman arman 1401 Jan 13 18:09 auth.py
drwxrwxr-x 15 arman arman 4096 Jan 13 18:09 challenges
drwxrwxr-x  3 arman arman 4096 Jan 13 18:09 conf
-rw-rw-r--  1 arman arman  264 Jan 14 02:18 config.env
drwxrwxr-x  7 arman arman 4096 Jan 13 18:09 containers
drwxrwxr-x  7 arman arman 4096 Jan 13 18:09 CTFd
drwxrwxr-x  3 arman arman 4096 Jan 13 18:09 CTFd_plugin
-rw-rw-r--  1 arman arman 2526 Jan 17 23:24 docker-compose.yml
-rwxrwxr-x  1 arman arman  771 Jan 13 18:09 enter.py
drwxrwxr-x  2 arman arman 4096 Jan 13 18:09 home_daemon
-rw-rw-r--  1 arman arman 3074 Jan 13 18:10 README.md
-rwxrwxr-x  1 arman arman 3003 Jan 13 18:31 resetup.sh
-rwxrwxr-x  1 arman arman 1373 Jan 13 18:09 restart.sh
-rwxrwxr-x  1 arman arman 1211 Jan 14 02:07 run.sh
drwxrwxr-x  2 arman arman 4096 Jan 13 18:09 scripts
-rw-rw-r--  1 arman arman 1986 Jan 13 18:09 script.sh
-rwxrwxr-x  1 arman arman 2996 Jan 14 01:40 setup.sh
-rw-rw-r--  1 arman arman 7196 Jan 13 18:09 tips.md
-rw-rw-r--  1 arman arman    0 Jan 13 18:09 todo-list.md
-rwxrwxr-x  1 arman arman  830 Jan 13 18:09 update.sh
```

Last modify time

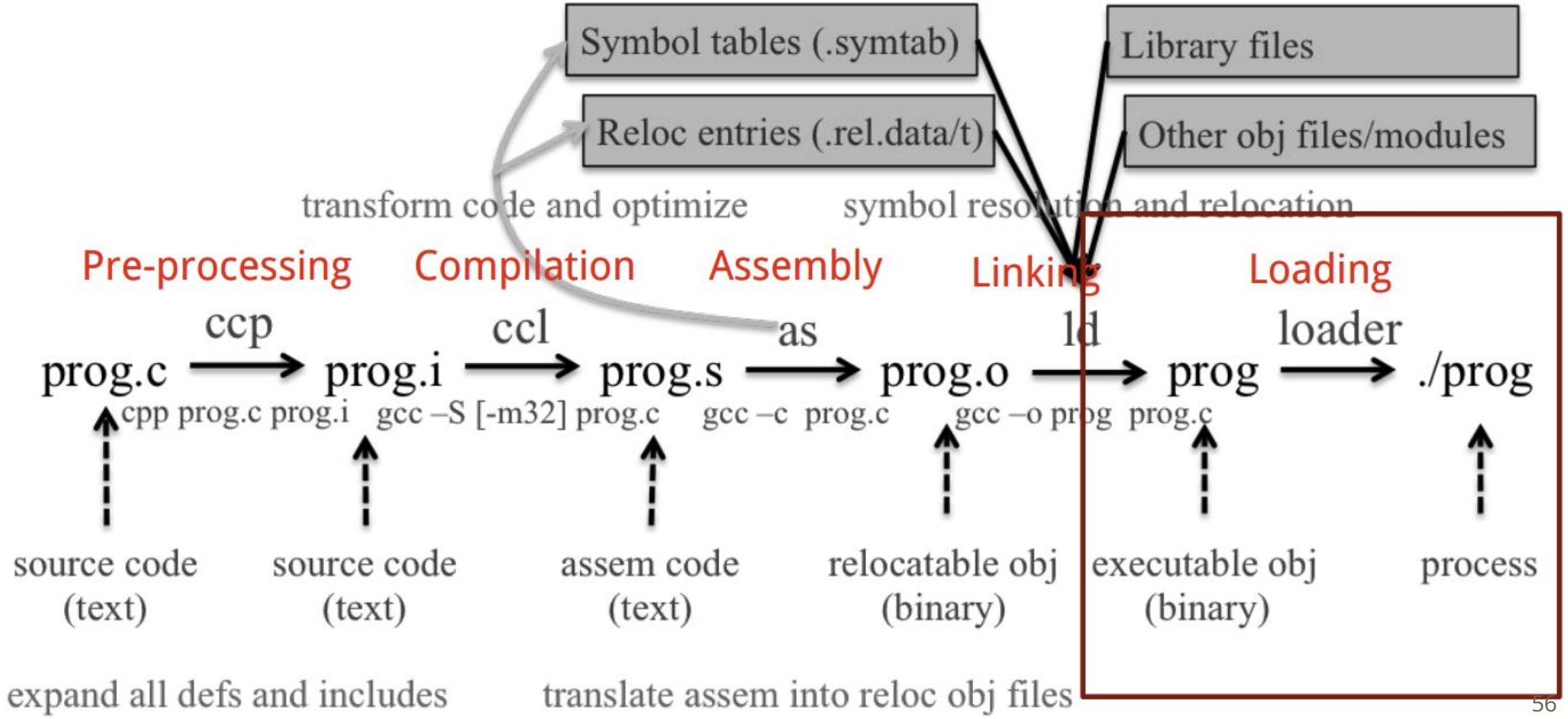
```
arman@aserver:~/STC/software-security$ ls -l
total 80
-rwxrwxr-x  1 arman arman 1401 Jan 13 18:09 auth.py
drwxrwxr-x 15 arman arman 4096 Jan 13 18:09 challenges
drwxrwxr-x  3 arman arman 4096 Jan 13 18:09 conf
-rw-rw-r--  1 arman arman  264 Jan 14 02:18 config.env
drwxrwxr-x  7 arman arman 4096 Jan 13 18:09 containers
drwxrwxr-x  7 arman arman 4096 Jan 13 18:09 CTFd
drwxrwxr-x  3 arman arman 4096 Jan 13 18:09 CTFd_plugin
-rw-rw-r--  1 arman arman 2526 Jan 17 23:24 docker-compose.yml
-rwxrwxr-x  1 arman arman  771 Jan 13 18:09 enter.py
drwxrwxr-x  2 arman arman 4096 Jan 13 18:09 home_daemon
-rw-rw-r--  1 arman arman 3074 Jan 13 18:10 README.md
-rwxrwxr-x  1 arman arman 3003 Jan 13 18:31 resetup.sh
-rwxrwxr-x  1 arman arman 1373 Jan 13 18:09 restart.sh
-rwxrwxr-x  1 arman arman 1211 Jan 14 02:07 run.sh
drwxrwxr-x  2 arman arman 4096 Jan 13 18:09 scripts
-rw-rw-r--  1 arman arman 1986 Jan 13 18:09 script.sh
-rwxrwxr-x  1 arman arman 2996 Jan 14 01:40 setup.sh
-rw-rw-r--  1 arman arman 7196 Jan 13 18:09 tips.md
-rw-rw-r--  1 arman arman    0 Jan 13 18:09 todo-list.md
-rwxrwxr-x  1 arman arman  830 Jan 13 18:09 update.sh
```

filename

```
arman@aserver:~/STC/software-security$ ls -l
total 80
-rwxrwxr-x  1 arman arman 1401 Jan 13 18:09 auth.py
drwxrwxr-x 15 arman arman 4096 Jan 13 18:09 challenges
drwxrwxr-x  3 arman arman 4096 Jan 13 18:09 conf
-rw-rw-r--  1 arman arman  264 Jan 14 02:18 config.env
drwxrwxr-x  7 arman arman 4096 Jan 13 18:09 containers
drwxrwxr-x  7 arman arman 4096 Jan 13 18:09 CTFd
drwxrwxr-x  3 arman arman 4096 Jan 13 18:09 CTFd_plugin
-rw-rw-r--  1 arman arman 2526 Jan 17 23:24 docker-compose.yml
-rwxrwxr-x  1 arman arman  771 Jan 13 18:09 enter.py
drwxrwxr-x  2 arman arman 4096 Jan 13 18:09 home_daemon
-rw-rw-r--  1 arman arman 3074 Jan 13 18:10 README.md
-rwxrwxr-x  1 arman arman 3003 Jan 13 18:31 resetup.sh
-rwxrwxr-x  1 arman arman 1373 Jan 13 18:09 restart.sh
-rwxrwxr-x  1 arman arman 1211 Jan 14 02:07 run.sh
drwxrwxr-x  2 arman arman 4096 Jan 13 18:09 scripts
-rw-rw-r--  1 arman arman 1986 Jan 13 18:09 script.sh
-rwxrwxr-x  1 arman arman 2996 Jan 14 01:40 setup.sh
-rw-rw-r--  1 arman arman 7196 Jan 13 18:09 tips.md
-rw-rw-r--  1 arman arman    0 Jan 13 18:09 todo-list.md
-rwxrwxr-x  1 arman arman  830 Jan 13 18:09 update.sh
```

Background Knowledge: Set-UID Programs

From a C program to a process



Real UID, Effective UID, and Saved UID

Each Linux/Unix process has 3 UIDs associated with it.

Real UID (RUID): This is the UID of the user/process that created THIS process. It can be changed only if the running process has EUID=0.

Effective UID (EUID): This UID is used to evaluate privileges of the process to perform a particular action. EUID can be changed either to RUID, or SUID if EUID!=0. If EUID=0, it can be changed to anything.

Saved UID (SUID): If the binary image file, that was launched has a Set-UID bit on, SUID will be the UID of the owner of the file. Otherwise, SUID will be the RUID.

Set-UID Program

The kernel makes the decision whether a process has the privilege by looking on the **EUID** of the process.

For non Set-UID programs, the effective uid and the real uid are the same. For Set-UID programs, **the effective uid is the owner of the program**, while the real uid is the user of the program.

What will happen is when a setuid binary executes, **the process changes its Effective User ID (EUID) from the default RUID to the owner of this special binary executable file which in this case is - root.**

```
arman@aserver:~/STC/software-security$ ls -al --color=always /usr/bin/ | head -n 100
total 407872
drwxr-xr-x  2 root root       36864 Jan 17 23:11 .
drwxr-xr-x 14 root root        4096 Jan 14 22:41 ..
-rwxr-xr-x  1 root root       55744 Apr  5  2024 [
-rwxr-xr-x  1 root root          94 Nov 12 12:15 2to3
-rwxr-xr-x  1 root root       18744 Mar 19  2025 aa-enabled
-rwxr-xr-x  1 root root       18744 Mar 19  2025 aa-exec
-rwxr-xr-x  1 root root       18736 Mar 19  2025 aa-features-abi
-rwxr-xr-x  1 root root        1622 Nov 18 11:26 acpidbg
-rwxr-xr-x  1 root root       16422 Feb 18  2025 add-apt-repository
-rwxr-xr-x  1 root root       14720 Jun  5  2025 addpart
lrwxrwxrwx  1 root root          26 Dec  3 15:01 addr2line -> x86_64-linux-gnu-addr2line
-rwxr-xr-x  1 root root        2322 Apr 18  2024 apport-bug
-rwxr-xr-x  1 root root       13625 Jul  8  2025 apport-cli
lrwxrwxrwx  1 root root          10 Jul  8  2025 apport-collect -> apport-bug
-rwxr-xr-x  1 root root        3790 Jul  8  2025 apport-unpack
-rwxr-xr-x  1 root root      141544 Apr  8  2024 appstreamcli
lrwxrwxrwx  1 root root           6 Aug  5 17:14 apropos -> whatis
-rwxr-xr-x  1 root root       18824 Oct 22  2024 apt
lrwxrwxrwx  1 root root          18 Feb 18  2025 apt-add-repository -> add-apt-repository
-rwxr-xr-x  1 root root       88544 Oct 22  2024 apt-cache
-rwxr-xr-x  1 root root       27104 Oct 22  2024 apt-cdrom
-rwxr-xr-x  1 root root       31120 Oct 22  2024 apt-config
-rwxr-xr-x  1 root root       59912 Apr  5  2024 chcon
-rwsr-xr-x  1 root root       72792 May 30  2024 chfn
-rwxr-xr-x  1 root root       59912 Apr  5  2024 chgrp
-rwxr-xr-x  1 root root       55816 Apr  5  2024 chmod
-rwxr-xr-x  1 root root       22912 Jun  5  2025 choom
-rwxr-xr-x  1 root root       59912 Apr  5  2024 chown
-rwxr-xr-x  1 root root       31104 Jun  5  2025 chrt
-rwsr-xr-x  1 root root       44760 May 30  2024 chsh
-rwxr-xr-x  1 root root       14712 Mar 31  2024 chvt
-rwxr-xr-x  1 root root       27080 Aug  5 17:14 cifsiostat
-rwxr-xr-x  1 root root      150674 Feb 26  2024 ckbcomp
-rwxr-xr-x  1 root root         227 Aug  5 17:14 ckeygen3
-rwxr-xr-x  1 root root      104984 Apr  5  2024 cksum
-rwxr-xr-x  1 root root       14656 Apr  8  2024 clear
-rwxr-xr-x  1 root root       14568 Mar 31  2024 clear_console
-rwxr-xr-x  1 root root         972 Jun 24  2025 cloud-id
-rwxr-xr-x  1 root root         976 Jun 24  2025 cloud-init
-rwxr-xr-x  1 root root        2108 Jun 24  2025 cloud-init-per
-rwxr-xr-x  1 root root       43408 Apr  8  2024 cmp
-rwxr-xr-x  1 root root       14640 Mar 31  2024 codepage
-rwxr-xr-x  1 root root       22920 Aug  5 17:14 col
-rwxr-xr-x  1 root root         963 Aug  5 17:14 col1
lrwxrwxrwx  1 root root           4 Aug  5 17:14 col2 -> col1
lrwxrwxrwx  1 root root           4 Aug  5 17:14 col3 -> col1
```

```
-rwxr-xr-x  1 root root      80408 Apr  5  2024 stty
-rwsr-xr-x  1 root root      55680 Jun  5  2025 su
-rwsr-xr-x  1 root root     277936 Jun 25  2025 sudo
lrwxrwxrwx  1 root root          4 Jun 25  2025 sudoedit -> sudo
-rwxr-xr-x  1 root root      98256 Jun 25  2025 sudoreplay
-rwxr-xr-x  1 root root      35240 Apr  5  2024 sum
-rwxr-xr-x  1 root root      35240 Apr  5  2024 sync
-rwxr-xr-x  1 root root    1501304 Jul  2  2025 systemctl
lrwxrwxrwx  1 root root         22 Jul  2  2025 systemd -> ../lib/systemd/systemd
-rwxr-xr-x  1 root root      14792 Jul  2  2025 systemd-ac-power
-rwxr-xr-x  1 root root     203624 Jul  2  2025 systemd-analyze
-rwxr-xr-x  1 root root      19024 Jul  2  2025 systemd-ask-password
-rwxr-xr-x  1 root root      18896 Jul  2  2025 systemd-cat
-rwxr-xr-x  1 root root      23112 Jul  2  2025 systemd-cgls
-rwxr-xr-x  1 root root      39392 Jul  2  2025 systemd-cgtop
lrwxrwxrwx  1 root root         14 Jul  2  2025 systemd-confext -> systemd-sysext
-rwxr-xr-x  1 root root      43744 Jul  2  2025 systemd-creds
-rwxr-xr-x  1 root root      72624 Jul  2  2025 systemd-cryptenroll
-rwxr-xr-x  1 root root      80840 Jul  2  2025 systemd-cryptsetup
-rwxr-xr-x  1 root root      27080 Jul  2  2025 systemd-delta
-rwxr-xr-x  1 root root      18888 Jul  2  2025 systemd-detect-virt
-rwxr-xr-x  1 root root      22984 Jul  2  2025 systemd-escape
-rwxr-xr-x  1 root root      60232 Jul  2  2025 systemd-firstboot
-rwxr-xr-x  1 root root     158456 Jul  2  2025 systemd-hwdb
```

Example: rdsecret

main.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <pwd.h>
int main(int argc, char *argv[])
{
 FILE *fp = NULL;
 char buffer[100] = {0};
 // get ruid and euid
 uid_t uid = getuid();
 struct passwd *pw = getpwuid(uid);
 if (pw)
 {
  printf("UID: %d, USER: %s.\n", uid, pw->pw_name);
 }
 uid_t euid = geteuid();
 pw = getpwuid(euid);
 if (pw)
 {
  printf("EUID: %d, EUSER: %s.\n", euid, pw->pw_name);
 }
 print_flag();

 return(0);
}

void print_flag()
{
 FILE *fp;
 char buff[MAX_FLAG_SIZE];
 fp = fopen("flag","r");
 fread(buff, MAX_FLAG_SIZE, 1, fp);
 printf("flag is : %s\n", buff);
 fclose(fp);
}
```

Background Knowledge: ELF Binary Files

ELF Files

The **Executable** and **Linkable Format** (ELF) is a common standard file format for executable files, object code, shared libraries, and core dumps. Filename extension none, .axf, .bin, .elf, .o, .prx, .puff, .ko, .mod and .so

Contains the program and its data. Describes how the program should be loaded (program/segment headers). Contains metadata describing program components (section headers).

Command *file*

```
arman@aserver:~$ file /bin/ls
/bin/ls: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-
64.so.2, BuildID[sha1]=3eca7e3905b37d48cf0a88b576faa7b95cc3097b, for GNU/Linux 3.2.0, stripped
```

file /bin/ls

```
arman@aserver:~$ readelf -a /bin/ls
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              DYN (Position-Independent Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x6d30
  Start of program headers:          64 (bytes into file)
  Start of section headers:          140328 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         13
  Size of section headers:           64 (bytes)
  Number of section headers:         31
  Section header string table index: 30

Section Headers:
  [Nr] Name              Type             Address           Offset
       Size              EntSize          Flags  Link  Info  Align
  [ 0]                   NULL             0000000000000000  00000000
       0000000000000000  0000000000000000           0     0     0
  [ 1] .interp           PROGBITS         0000000000000318  00000318
       000000000000001c  0000000000000000   A       0     0     1
  [ 2] .note.gnu.pr[...] NOTE             0000000000000338  00000338
       0000000000000030  0000000000000000   A       0     0     8
  [ 3] .note.gnu.bu[...] NOTE             0000000000000368  00000368
       0000000000000024  0000000000000000   A       0     0     4
  [ 4] .note.ABI-tag     NOTE             000000000000038c  0000038c
       0000000000000020  0000000000000000   A       0     0     4
  [ 5] .gnu.hash         GNU_HASH         00000000000003b0  000003b0
       0000000000000050  0000000000000000   A       6     0     8
  [ 6] .dynsym           DYNSYM           0000000000000400  00000400
       0000000000000c00  0000000000000018   A       7     1     8
  [ 7] .dynstr           STRTAB           0000000000001000  00001000
       0000000000000614  0000000000000000   A       0     0     1
```

INTERP: defines the library that should be used to load this ELF into memory.

LOAD: defines a part of the file that should be loaded into memory.

Sections:

.text: the executable code of your program.

.plt and **.got**: used to resolve and dispatch library calls.

.data: used for pre-initialized global writable data (such as global arrays with initial values)

.rodata: used for global read-only data (such as string constants)

.bss: used for uninitialized global writable data (such as global arrays without initial values)

Tools for ELF

gcc to make your ELF.

readelf to parse the ELF header.

objdump to parse the ELF header and disassemble the source code.

nm to view your ELF's symbols.

patchelf to change some ELF properties.

objcopy to swap out ELF sections.

strip to remove otherwise-helpful information (such as symbols).

kaitai struct (https://ide.kaitai.io/) to look through your ELF interactively.

Background Knowledge: Memory Map of a Linux Process

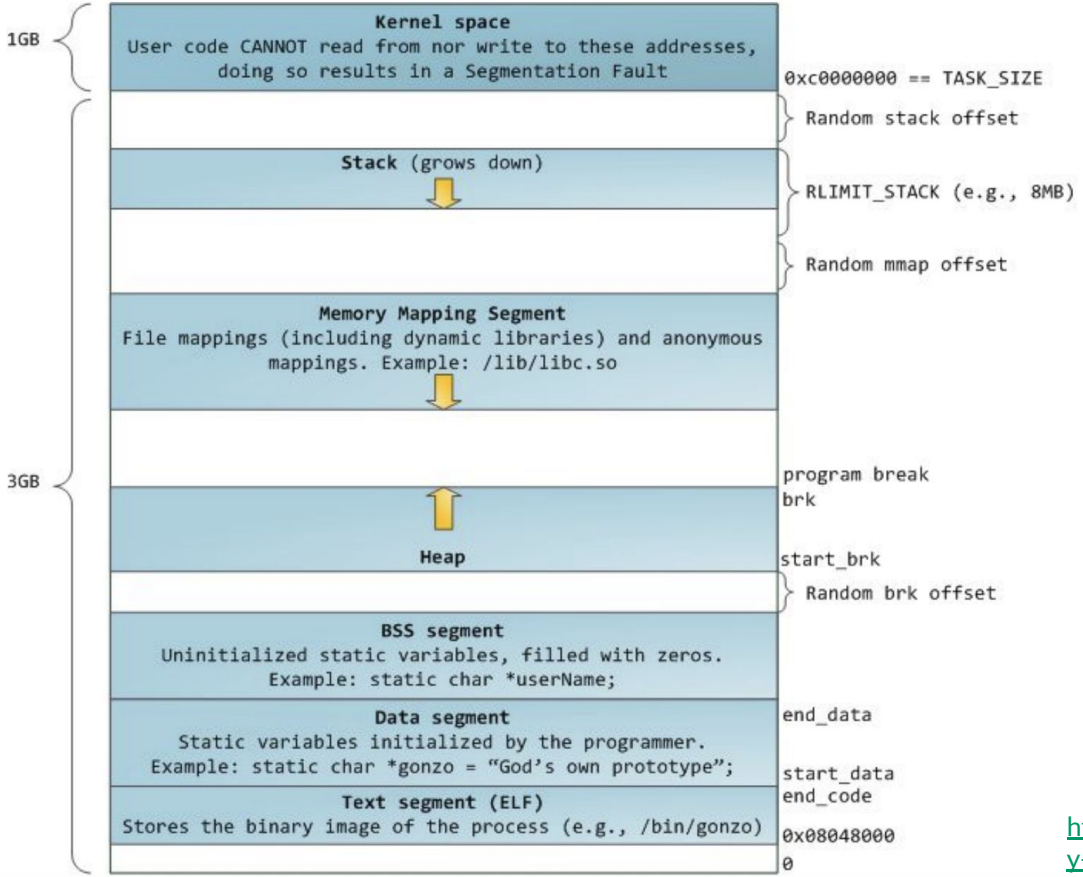
Memory Map of Linux Process (32 bit)

Each process in a multi-tasking OS runs in its own memory sandbox.

This sandbox is the **virtual address space**, which in 32-bit mode is **always a 4GB block of memory addresses**.

These virtual addresses are mapped to physical memory by **page tables**, which are maintained by the operating system kernel and consulted by the processor

Memory Map of Linux Process (32 bit)



https://manybutfinite.com/post/anatomy-of-a-program-in-memory/

NULL Pointer in C/C++

int * pInt = NULL;

In possible definitions of NULL in C/C++:

```
#define NULL ((char *)0)
#define NULL 0

//since C++11
#define NULL nullptr
```

/proc/pid_of_process/maps

Example processmap.c

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
  getchar();
  return 0;
}
```

cat /proc/pid/maps

pmap -X pid

pmap -X `pidof pm`

/proc/pid_of_process/maps

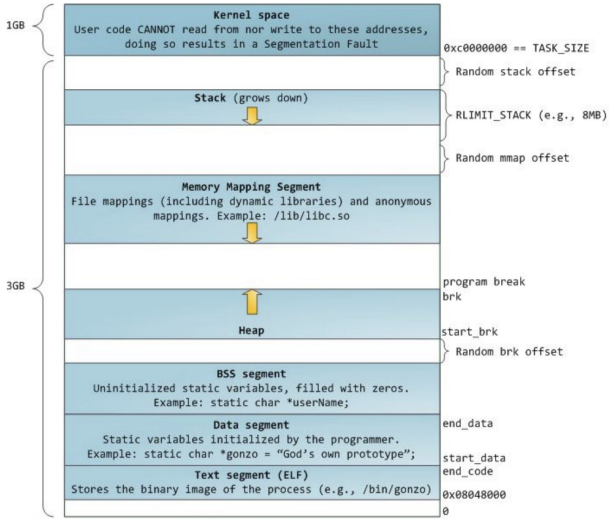
Example processmap.c

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
  getchar();
  return 0;
}
```

cat /proc/pid/maps

pmap -X pid

pmap -X `pidof pm`




```
arman@aserver:~$ pmap -X 746491
746491:   ./processmap_32
  Address Perm   Offset Device   Inode Size  Rss  Pss Pss_Dirty Referenced Anonymous KSM LazyFree ShmemPmdMapped FilePmdMapped Shared_Hugetlb Private_Hugetlb Swap SwapPss Locked THPeligible ProtectionKey Mapping
5d978000 r--p 00000000  fc:00 3145956    4    4    4         0          4         0   0        0              0             0              0               0    0       0      0           0             0 processmap_32
5d979000 r-xp 00001000  fc:00 3145956    4    4    4         0          4         0   0        0              0             0              0               0    0       0      0           0             0 processmap_32
5d97a000 r--p 00002000  fc:00 3145956    4    0    0         0          0         0   0        0              0             0              0               0    0       0      0           0             0 processmap_32
5d97b000 r--p 00002000  fc:00 3145956    4    4    4         4          4         4   0        0              0             0              0               0    0       0      0           0             0 processmap_32
5d97c000 rw-p 00003000  fc:00 3145956    4    4    4         4          4         4   0        0              0             0              0               0    0       0      0           0             0 processmap_32
5e74c000 rw-p 00000000  00:00       0  136    4    4         4          4         4   0        0              0             0              0               0    0       0      0           0             0 [heap]
efde9000 r--p 00000000  fc:00 5544451  140  140  140         0        140         0   0        0              0             0              0               0    0       0      0           0             0 libc.so.6
efe0c000 r-xp 00023000  fc:00 5544451 1532  784  784         0        784         0   0        0              0             0              0               0    0       0      0           0             0 libc.so.6
eff8b000 r--p 001a2000  fc:00 5544451  532   64   64         0         64         0   0        0              0             0              0               0    0       0      0           0             0 libc.so.6
f0010000 r--p 00226000  fc:00 5544451    8    8    8         8          8         8   0        0              0             0              0               0    0       0      0           0             0 libc.so.6
f0012000 rw-p 00228000  fc:00 5544451    4    4    4         4          4         4   0        0              0             0              0               0    0       0      0           0             0 libc.so.6
f0013000 rw-p 00000000  00:00       0   40   16   16        16         16        16   0        0              0             0              0               0    0       0      0           0             0
f0024000 rw-p 00000000  00:00       0    8    8    8         8          8         8   0        0              0             0              0               0    0       0      0           0             0
f0026000 r--p 00000000  00:00       0   16    0    0         0          0         0   0        0              0             0              0               0    0       0      0           0             0 [vvar]
f002a000 r-xp 00000000  00:00       0    8    4    4         0          4         0   0        0              0             0              0               0    0       0      0           0             0 [vdso]
f002c000 r--p 00000000  fc:00 5544448    4    4    4         0          4         0   0        0              0             0              0               0    0       0      0           0             0 ld-linux.so.2
f002d000 r-xp 00001000  fc:00 5544448  140  140  140         0        140         0   0        0              0             0              0               0    0       0      0           0             0 ld-linux.so.2
f0050000 r--p 00024000  fc:00 5544448   56   56   56         0         56         0   0        0              0             0              0               0    0       0      0           0             0 ld-linux.so.2
f005e000 r--p 00031000  fc:00 5544448    8    8    8         8          8         8   0        0              0             0              0               0    0       0      0           0             0 ld-linux.so.2
f0060000 rw-p 00033000  fc:00 5544448    4    4    4         4          4         4   0        0              0             0              0               0    0       0      0           0             0 ld-linux.so.2
ff9bc000 rw-p 00000000  00:00       0  132   12   12        12         12        12   0        0              0             0              0               0    0       0      0           0             0 [stack]
                                     ==== ==== ==== ======== ========== ======== === ======== ============== ============= ============== ============== ==== ======= ====== =========== =============
                                     2788 1272 1272        72       1272        72   0        0              0             0              0               0    0       0      0           0             0 KB
```

Memory Map of Linux Process (64 bit system)

```
arman@aserver:~$ pmap -X 751600
751600:   ./processmap_64
         Address Perm   Offset Device   Inode Size  Rss Pss Pss_Dirty Referenced Anonymous KSM LazyFree ShmemPmdMapped FilePmdMapped Shared_Hugetlb Private_Hugetlb Swap SwapPss Locked THPeligible ProtectionKey Mapping
    56e4fc8ed000 r--p 00000000  fc:00 3146755    4    4   4         0          4         0   0        0              0             0              0               0    0       0      0           0             0 processmap_64
    56e4fc8ee000 r-xp 00001000  fc:00 3146755    4    4   4         0          4         0   0        0              0             0              0               0    0       0      0           0             0 processmap_64
    56e4fc8ef000 r--p 00002000  fc:00 3146755    4    0   0         0          0         0   0        0              0             0              0               0    0       0      0           0             0 processmap_64
    56e4fc8f0000 r--p 00002000  fc:00 3146755    4    4   4         4          4         4   0        0              0             0              0               0    0       0      0           0             0 processmap_64
    56e4fc8f1000 rw-p 00003000  fc:00 3146755    4    4   4         4          4         4   0        0              0             0              0               0    0       0      0           0             0 processmap_64
    56e51f43c000 rw-p 00000000  00:00       0  132    4   4         4          4         4   0        0              0             0              0               0    0       0      0           0             0 [heap]
    7d0ef6400000 r--p 00000000  fc:00 5540086  160  160   3         0        160         0   0        0              0             0              0               0    0       0      0           0             0 libc.so.6
    7d0ef6428000 r-xp 00028000  fc:00 5540086 1568  864  19         0        864         0   0        0              0             0              0               0    0       0      0           0             0 libc.so.6
    7d0ef65b0000 r--p 001b0000  fc:00 5540086  316    0   0         0          0         0   0        0              0             0              0               0    0       0      0           0             0 libc.so.6
    7d0ef65ff000 r--p 001fe000  fc:00 5540086   16   16  16        16         16        16   0        0              0             0              0               0    0       0      0           0             0 libc.so.6
    7d0ef6603000 rw-p 00202000  fc:00 5540086    8    8   8         8          8         8   0        0              0             0              0               0    0       0      0           0             0 libc.so.6
    7d0ef6605000 rw-p 00000000  00:00       0   52   20  20        20         20        20   0        0              0             0              0               0    0       0      0           0             0
    7d0ef67fa000 rw-p 00000000  00:00       0   12    8   8         8          8         8   0        0              0             0              0               0    0       0      0           0             0
    7d0ef6804000 rw-p 00000000  00:00       0    8    4   4         4          4         4   0        0              0             0              0               0    0       0      0           0             0
    7d0ef6806000 r--p 00000000  fc:00 5540083    4    4   0         0          4         0   0        0              0             0              0               0    0       0      0           0             0 ld-linux-x86-64.so.2
    7d0ef6807000 r-xp 00001000  fc:00 5540083  172  172   3         0        172         0   0        0              0             0              0               0    0       0      0           0             0 ld-linux-x86-64.so.2
    7d0ef6832000 r--p 0002c000  fc:00 5540083   40   40   0         0         40         0   0        0              0             0              0               0    0       0      0           0             0 ld-linux-x86-64.so.2
    7d0ef683c000 r--p 00036000  fc:00 5540083    8    8   8         8          8         8   0        0              0             0              0               0    0       0      0           0             0 ld-linux-x86-64.so.2
    7d0ef683e000 rw-p 00038000  fc:00 5540083    8    8   8         8          8         8   0        0              0             0              0               0    0       0      0           0             0 ld-linux-x86-64.so.2
    7ffe0aeac000 rw-p 00000000  00:00       0  132   12  12        12         12        12   0        0              0             0              0               0    0       0      0           0             0 [stack]
    7ffe0afb9000 r--p 00000000  00:00       0   16    0   0         0          0         0   0        0              0             0              0               0    0       0      0           0             0 [vvar]
    7ffe0afbd000 r-xp 00000000  00:00       0    8    4   0         0          4         0   0        0              0             0              0               0    0       0      0           0             0 [vdso]
ffffffffff600000 --xp 00000000  00:00       0    4    0   0         0          0         0   0        0              0             0              0               0    0       0      0           0             0 [vsyscall]
                                            ==== ==== === ======== ========= ======== === ======== ============= ============ ============= ============== ==== ======= ====== ========== ============
                                            2684 1348 129        96       1348        96   0        0              0             0              0               0    0       0      0           0             0 KB
```

Background Knowledge: System Calls

What is System Call?

When a process needs to invoke a kernel service, it invokes a procedure call in the operating system interface using special instructions (not a **call** instruction in x86). Such a procedure is called a system call.

The system call enters the kernel; the kernel performs the service and returns. Thus a process alternates between executing in user space and kernel space.

System calls are generally not invoked directly by a program, but rather via wrapper functions in glibc (or perhaps some other library).

Popular System Call

On Unix, Unix-like and other POSIX-compliant operating systems, popular system calls are open, read, write, close, wait, exec, fork, exit, and kill.

Many modern operating systems have hundreds of system calls. For example, Linux and OpenBSD each have over 300 different calls, FreeBSD has over 500, Windows 7 has close to 700.

Glibc interfaces

Often, but not always, the name of the wrapper function is the same as the name of the system call that it invokes

For example, glibc contains a function **chdir()** which invokes the underlying "chdir" system call.

Tools: strace & ltrace

```
ctf@misc_firstflag_64:/$ strace ./misc_firstflag_64
execve("./misc_firstflag_64", ["./misc_firstflag_64"], 0x7ffc81da94a0 /* 13 vars */) = 0
access("/etc/suid-debug", F_OK)         = -1 ENOENT (No such file or directory)
brk(NULL)                               = 0x5ca81e6ff000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffec0373bf0) = -1 EINVAL (Invalid argument)
fcntl(0, F_GETFD)                       = 0
fcntl(1, F_GETFD)                       = 0
fcntl(2, F_GETFD)                       = 0
access("/etc/suid-debug", F_OK)         = -1 ENOENT (No such file or directory)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=47355, ...}) = 0
mmap(NULL, 47355, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7a8c6a314000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\360q\2\0\0\0\0\0"..., 832) = 832
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
pread64(3, "\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0", 32, 848) = 32
pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\t\233\222%\274\260\320\31\331\326\10\204\276X>\263"..., 68, 880) = 68
fstat(3, {st_mode=S_IFREG|0755, st_size=2029224, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7a8c6a312000
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
pread64(3, "\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0", 32, 848) = 32
pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\t\233\222%\274\260\320\31\331\326\10\204\276X>\263"..., 68, 880) = 68
mmap(NULL, 2036952, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7a8c6a120000
mprotect(0x7a8c6a145000, 1847296, PROT_NONE) = 0
mmap(0x7a8c6a145000, 1540096, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x25000) = 0x7a8c6a145000
mmap(0x7a8c6a2bd000, 303104, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x19d000) = 0x7a8c6a2bd000
mmap(0x7a8c6a308000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e7000) = 0x7a8c6a308000
mmap(0x7a8c6a30e000, 13528, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7a8c6a30e000
close(3)                                = 0
arch_prctl(ARCH_SET_FS, 0x7a8c6a313540) = 0
mprotect(0x7a8c6a308000, 12288, PROT_READ) = 0
mprotect(0x5ca7f9cd0000, 4096, PROT_READ) = 0
mprotect(0x7a8c6a34d000, 4096, PROT_READ) = 0
munmap(0x7a8c6a314000, 47355)           = 0
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x1), ...}) = 0
brk(NULL)                               = 0x5ca81e6ff000
brk(0x5ca81e720000)                     = 0x5ca81e720000
write(1, "Congratulations on getting your "..., 45Congratulations on getting your first flag!!
) = 45
openat(AT_FDCWD, "/flag", O_RDONLY)     = -1 EACCES (Permission denied)
write(1, "Error: Cannot open the flag file"..., 36Error: Cannot open the flag file!!!
) = 36
exit_group(0)                           = ?
+++ exited with 0 +++
```

misc/firstflag

main.c

```
int main(int argc, char *argv[])
{
  printf("Congratulations on getting your first flag!!\n");
  print_flag();
}
```

flag.h

```
int print_flag()
{
        FILE *fp = NULL;
        char buff[MAX_FLAG_SIZE] = {0};
        fp = fopen("/flag", "r");
        if (fp == NULL)
{
         printf("Error: Cannot open the flag file!!!\n");
        return 1;
 }
        fread(buff, MAX_FLAG_SIZE - 2, 1, fp);
        printf("The flag is: %s\n", buff);
        fclose(fp);
        return 0;
}
```

Tools: strace & ltrace

```
ctf@misc_firstflag_64:/$ strace ./misc_firstflag_64
execve("./misc_firstflag_64", ["./misc_firstflag_64"], 0x7ffc81da94a0 /* 13 vars */) = 0
access("/etc/suid-debug", F_OK)         = -1 ENOENT (No such file or directory)
brk(NULL)                               = 0x5ca81e6ff000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffec0373bf0) = -1 EINVAL (Invalid argument)
fcntl(0, F_GETFD)                       = 0
fcntl(1, F_GETFD)                       = 0
fcntl(2, F_GETFD)                       = 0
access("/etc/suid-debug", F_OK)         = -1 ENOENT (No such file or directory)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=47355, ...}) = 0
mmap(NULL, 47355, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7a8c6a314000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\360q\2\0\0\0\0\0"..., 832) = 832
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0"..., 784, 64) = 784
pread64(3, "\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0"..., 32, 848) = 32
pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\t\233\222%\274\260\320\31\331\326\10\204\276X>\263"..., 68, 880) = 68
fstat(3, {st_mode=S_IFREG|0755, st_size=2029224, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7a8c6a312000
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0"..., 784, 64) = 784
pread64(3, "\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0"..., 32, 848) = 32
pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\t\233\222%\274\260\320\31\331\326\10\204\276X>\263"..., 68, 880) = 68
mmap(NULL, 2036952, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7a8c6a120000
mprotect(0x7a8c6a145000, 1847296, PROT_NONE) = 0
mmap(0x7a8c6a145000, 1540096, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x25000) = 0x7a8c6a145000
mmap(0x7a8c6a2bd000, 303104, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x19d000) = 0x7a8c6a2bd000
mmap(0x7a8c6a308000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e7000) = 0x7a8c6a308000
mmap(0x7a8c6a30e000, 13528, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7a8c6a30e000
close(3)                                = 0
arch_prctl(ARCH_SET_FS, 0x7a8c6a313540) = 0
mprotect(0x7a8c6a308000, 12288, PROT_READ) = 0
mprotect(0x5ca7f9cd0000, 4096, PROT_READ) = 0
mprotect(0x7a8c6a34d000, 4096, PROT_READ) = 0
munmap(0x7a8c6a314000, 47355)           = 0
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x1), ...}) = 0
brk(NULL)                               = 0x5ca81e6ff000
brk(0x5ca81e720000)                     = 0x5ca81e720000
write(1, "Congratulations on getting your "..., 45Congratulations on getting your first flag!!
) = 45
openat(AT_FDCWD, "/flag", O_RDONLY)     = -1 EACCES (Permission denied)
write(1, "Error: Cannot open the flag file"..., 36Error: Cannot open the flag file!!!
) = 36
exit_group(0)                           = ?
+++ exited with 0 +++
```

Execve - first system call
Access - check file permission
Brk - check data segment/heap
Arch_prctl - set architecture-specific thread state
Fcntl - manipulate file descriptor
Openat - similar to open
Fstat - get file status
Mmap - map files or devices into memory
Close
Read
Pread64 - similar to read
Mprotect - set protection on a region of memory
Munmap - map files or devices into memory
Write
Exit_group

Use “man 2 syscall_name” to check out its usage

Making a System Call in x86/64 Assembly

On x86/x86-64, most system calls rely on the software interrupt.

A software interrupt is caused either by an exceptional condition in the processor itself, or a special instruction (the **int 0x80** instruction or **syscall** instruction)

For example: a divide-by-zero exception will be thrown if the processor's arithmetic logic unit is commanded to divide a number by zero as this instruction is in error and impossible.

Making a System Call in x86 Assembly (INT 0x80)

x86 (32-bit)

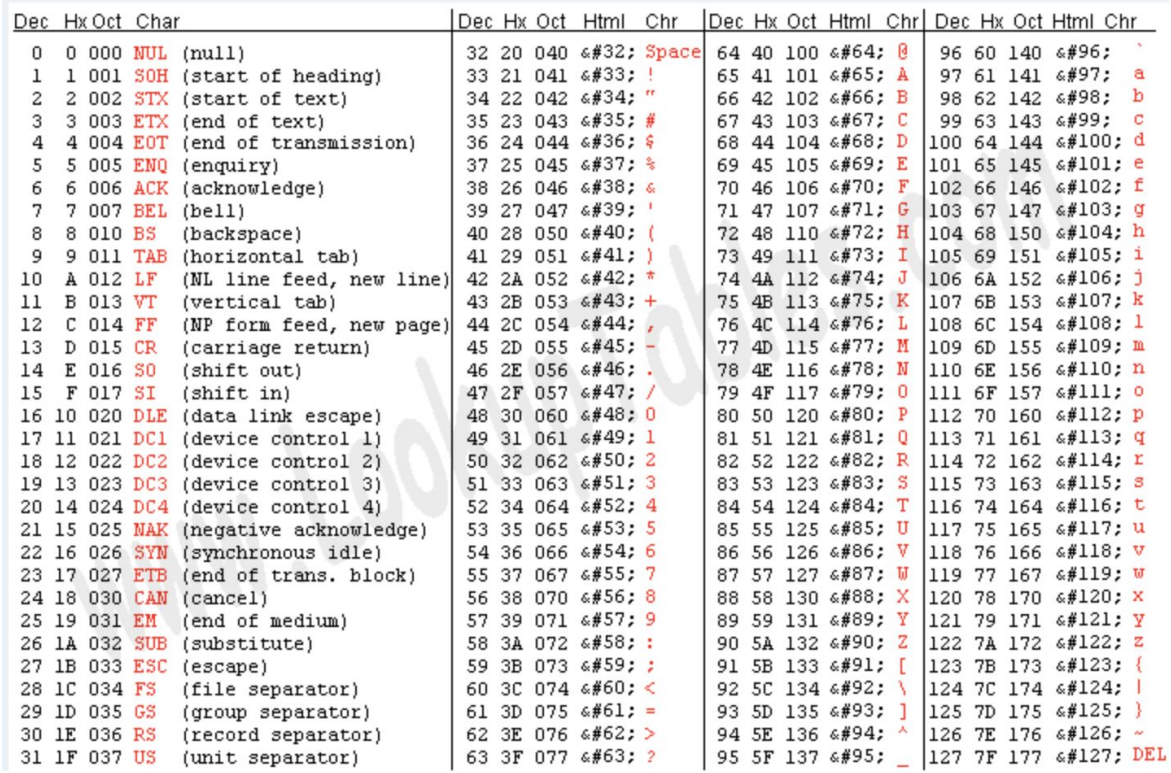
Compiled from Linux 4.14.0 headers.

NR	syscall name	references	%eax	arg0 (%ebx)	arg1 (%ecx)	arg2 (%edx)	arg3 (%esi)	arg4 (%edi)	arg5 (%ebp)
0	restart_syscall	man/ cs/	0x00	-	-	-	-	-	-
1	exit	man/ cs/	0x01	int error_code	-	-	-	-	-
2	fork	man/ cs/	0x02	-	-	-	-	-	-
3	read	man/ cs/	0x03	unsigned int fd	char *buf	size_t count	-	-	-
4	write	man/ cs/	0x04	unsigned int fd	const char *buf	size_t count	-	-	-
5	open	man/ cs/	0x05	const char *filename	int flags	umode_t mode	-	-	-
6	close	man/ cs/	0x06	unsigned int fd	-	-	-	-	-
7	waitpid	man/ cs/	0x07	pid_t pid	int *stat_addr	int options	-	-	-
8	creat	man/ cs/	0x08	const char *pathname	umode_t mode	-	-	-	-
9	link	man/ cs/	0x09	const char *oldname	const char *newname	-	-	-	-
10	unlink	man/ cs/	0x0a	const char *pathname	-	-	-	-	-
11	execve	man/ cs/	0x0b	const char *filename	const char *const *argv	const char *const *envp	-	-	-
12	chdir	man/ cs/	0x0c	const char *filename	-	-	-	-	-
13	time	man/ cs/	0x0d	time_t *tloc	-	-	-	-	-
14	mknod	man/ cs/	0x0e	const char *filename	umode_t mode	unsigned dev	-	-	-
15	chmod	man/ cs/	0x0f	const char *filename	umode_t mode	-	-	-	-
16	lchown	man/ cs/	0x10	const char *filename	uid_t user	gid_t group	-	-	-
17	break	man/ cs/	0x11	?	?	?	?	?	?

https://chromium.googlesource.com/chromiumos/docs/+/master/constants/syscalls.md#x86-32_bit

Making a System Call in x86 Assembly

```
xor eax,eax
push eax
push 0x68732f2f
push 0x6e69622f
mov ebx,esp
push eax
push ebx
mov ecx,esp
mov al,0xb
int 0x80
```

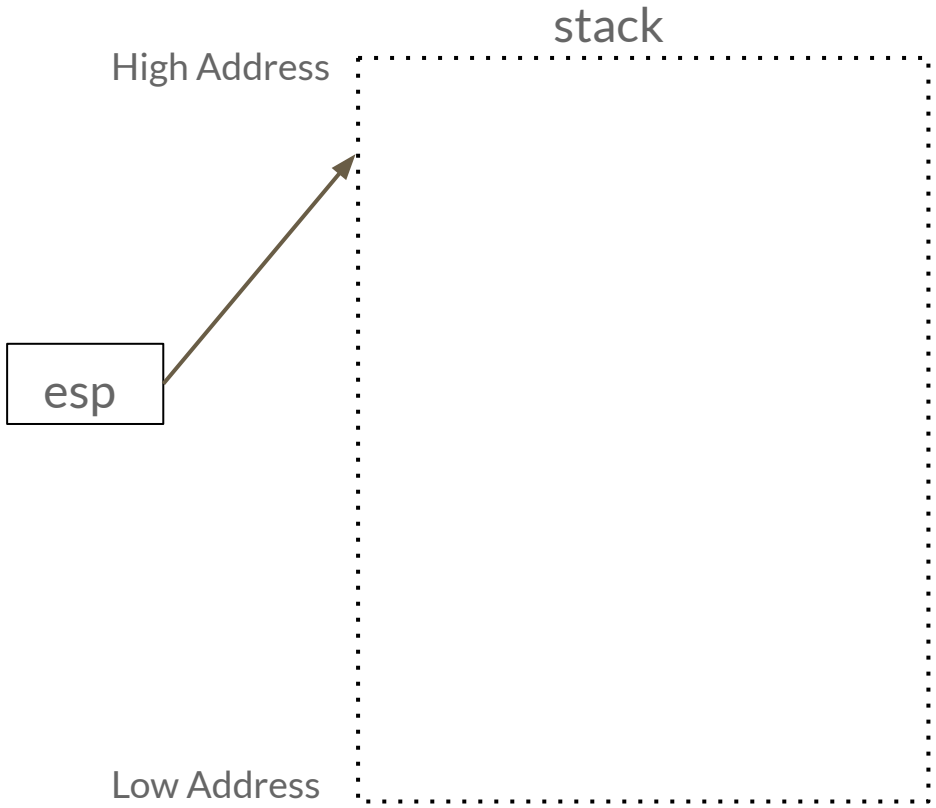


Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

https://shell-storm.org/shellcode/files/shellcode-827.html

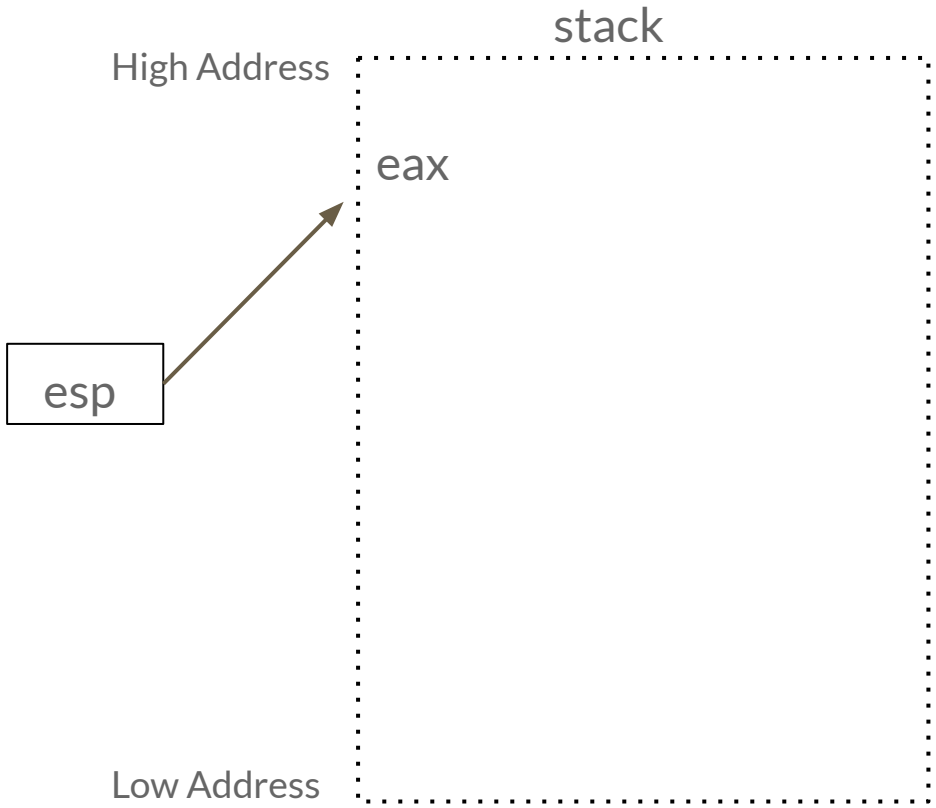
Making a System Call in x86 Assembly

```
xor eax,eax
push eax
push 0x68732f2f
push 0x6e69622f
mov ebx,esp
push eax
push ebx
mov ecx,esp
mov al,0xb
int 0x80
```



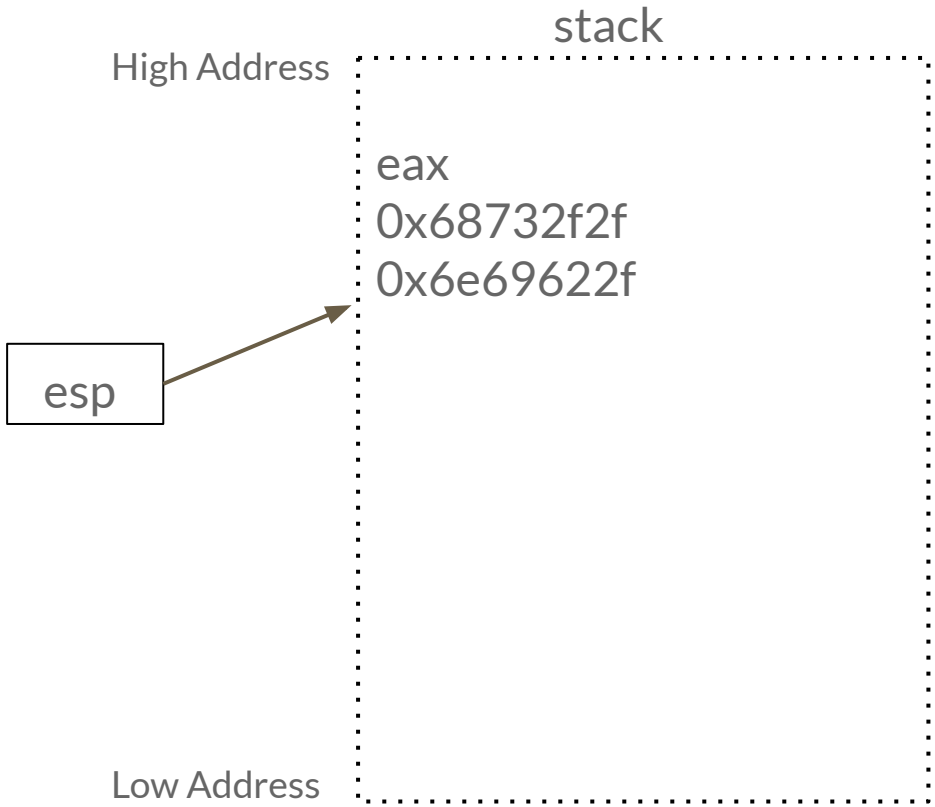
Making a System Call in x86 Assembly

```
xor eax,eax
push eax
push 0x68732f2f
push 0x6e69622f
mov ebx,esp
push eax
push ebx
mov ecx,esp
mov al,0xb
int 0x80
```



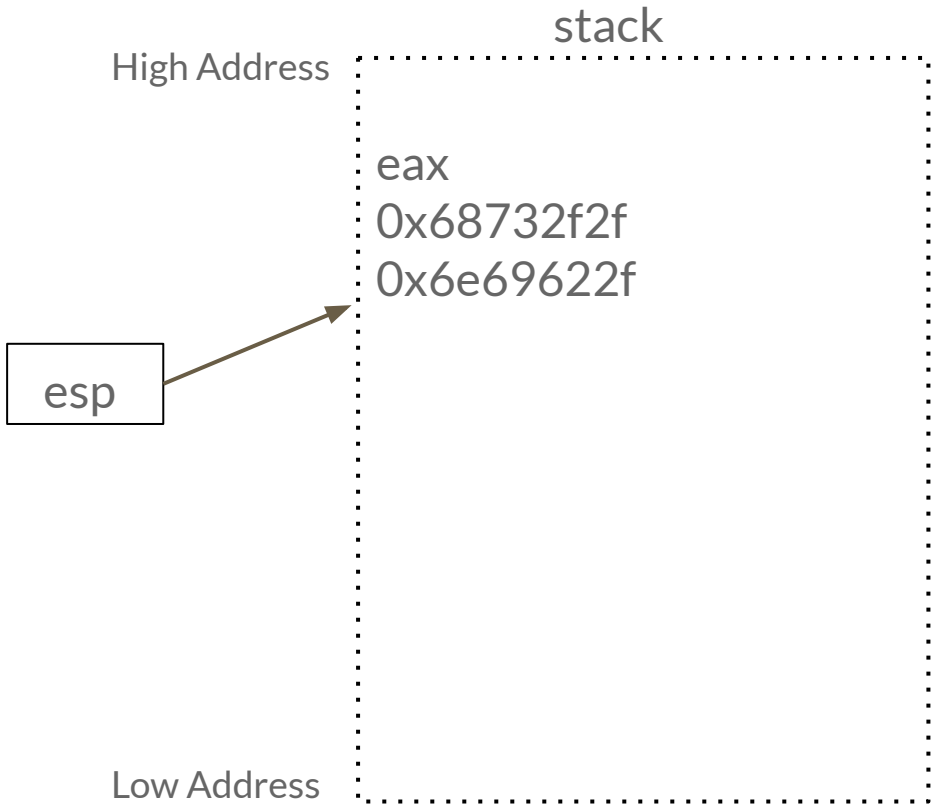
Making a System Call in x86 Assembly

```
xor eax,eax
push eax
push 0x68732f2f
push 0x6e69622f
mov ebx,esp
push eax
push ebx
mov ecx,esp
mov al,0xb
int 0x80
```



Making a System Call in x86 Assembly

```
xor eax,eax
push eax
push 0x68732f2f
push 0x6e69622f
mov ebx,esp
push eax
push ebx
mov ecx,esp
mov al,0xb
int 0x80
```



Making a System Call in x86 Assembly

```
execve(2)                                                    System Calls Manual

NAME
       execve - execute program

LIBRARY
       Standard C library (libc, -lc)

SYNOPSIS
       #include <unistd.h>

       int execve(const char *pathname, char *const _Nullable argv[],
                  char *const _Nullable envp[]);
```

execve(“/bin/sh”, address of string “/bin/sh”, 0)

Making a System Call in x86_64 (64-bit) Assembly

x86_64 (64-bit)

Compiled from Linux 4.14.0 headers.

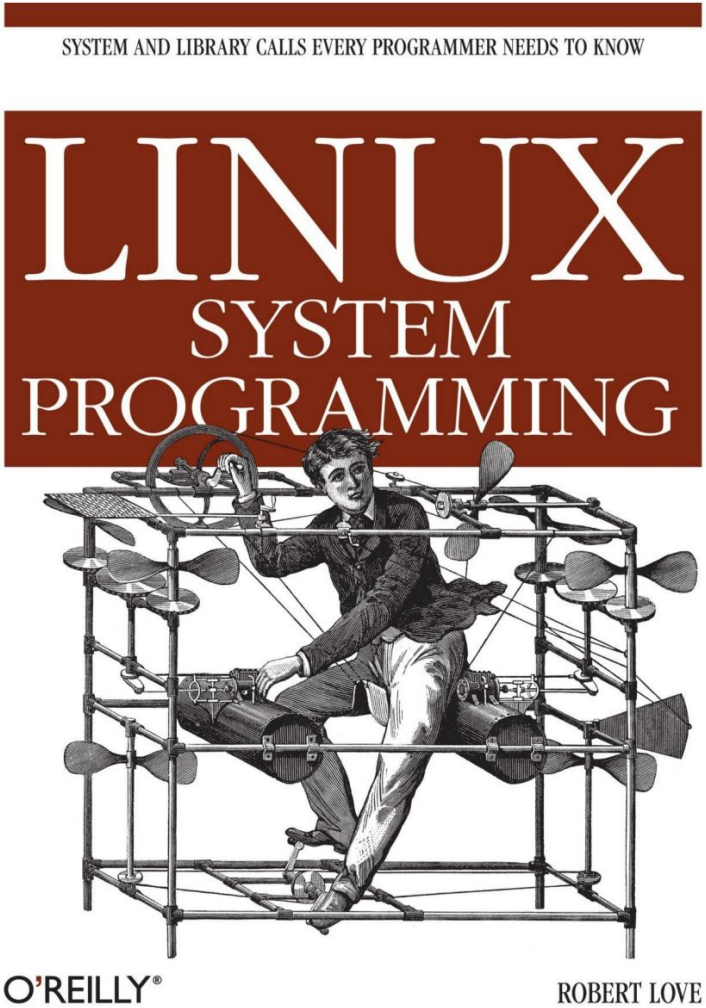
NR	syscall name	references	%rax	arg0 (%rdi)	arg1 (%rsi)	arg2 (%rdx)	arg3 (%r10)	arg4 (%r8)	arg5 (%r9)
0	read	man/ cs/	0x00	unsigned int fd	char *buf	size_t count	-	-	-
1	write	man/ cs/	0x01	unsigned int fd	const char *buf	size_t count	-	-	-
2	open	man/ cs/	0x02	const char *filename	int flags	umode_t mode	-	-	-
3	close	man/ cs/	0x03	unsigned int fd	-	-	-	-	-
4	stat	man/ cs/	0x04	const char *filename	struct __old_kernel_stat *statbuf	-	-	-	-
5	fstat	man/ cs/	0x05	unsigned int fd	struct __old_kernel_stat *statbuf	-	-	-	-
6	lstat	man/ cs/	0x06	const char *filename	struct __old_kernel_stat *statbuf	-	-	-	-
7	poll	man/ cs/	0x07	struct pollfd *ufds	unsigned int nfds	int timeout	-	-	-
8	lseek	man/ cs/	0x08	unsigned int fd	off_t offset	unsigned int whence	-	-	-
9	mmap	man/ cs/	0x09	?	?	?	?	?	?
10	mprotect	man/ cs/	0x0a	unsigned long start	size_t len	unsigned long prot	-	-	-
11	munmap	man/ cs/	0x0b	unsigned long addr	size_t len	-	-	-	-
12	brk	man/ cs/	0x0c	unsigned long brk	-	-	-	-	-
13	rt_sigaction	man/ cs/	0x0d	int	const struct sigaction *	struct sigaction *	size_t	-	-
14	rt_sigprocmask	man/ cs/	0x0e	int how	sigset_t *set	sigset_t *oset	size_t sigsetsize	-	-
15	rt_sigreturn	man/ cs/	0x0f	?	?	?	?	?	?
16	ioctl	man/ cs/	0x10	unsigned int fd	unsigned int cmd	unsigned long arg	-	-	-

https://chromium.googlesource.com/chromiumos/docs/+/master/constants/syscalls.md#x86-32_bit

Making a System Call in x86_64 (64-bit) Assembly

NR	syscall name	references	%rax	arg0 (%rdi)	arg1 (%rsi)	arg2 (%rdx)	arg3 (%r10)	arg4 (%r8)	arg5 (%r9)
59	execve	man/ cs/	0x3b	const char *filename	const char *const *argv	const char *const *envp	-	-	-

```
push rax
xor rdx, rdx
xor rsi, rsi
mov rbx,'/bin//sh'
push rbx
push rsp
pop rdi
mov al, 59
syscall
```


SYSTEM AND LIBRARY CALLS EVERY PROGRAMMER NEEDS TO KNOW
LINUX
SYSTEM
PROGRAMMING
O'REILLY®
ROBERT LOVE

Background Knowledge: System Calls

Channels of Communication for Linux Process

Every process in Linux has three initial, standard channels of communication:

- Standard Input (stdin, fd=0) is the channel through which the process takes input. For example, your shell uses Standard Input to read the commands that you input.
- Standard Output (stdout, fd=1) is the channel through which processes output normal data, such as the flag when it is printed to you in previous challenges or the output of utilities such as ls.
- Standard Error (stderr, fd=2) is the channel through which processes output error details. For example, if you mistype a command, the shell will output, over standard error, that this command does not exist.

Examples

Redirecting output > or 1>

echo hi > asdf echo hi 1> asdf

Appending output >>

echo hi >> asdf

Redirecting errors 2>

/challenge/run 2> errors.log

Redirecting input <

rev < messagefile

Channels of Communication for Linux Process

Process can also take input from command line arguments

ls -al c

at /flag

cat 1.txt 2.txt 3.txt

Pipe

The | (pipe) operator. Standard output from the command to the left of the pipe will be connected to (piped into) the standard input of the command to the right of the pipe.

echo hello-world | wc -c

Background Knowledge: Environment and Shell Variables

Environment and Shell Variables

Environment and Shell variables are a set of dynamic **named values,** stored within the system that are used by applications launched in shells.

KEY=value

KEY="Some other value"

KEY=value1:value2

The names of the variables are case-sensitive (UPPER CASE). Multiple values must be separated by the colon : character. There is no space around the equals = symbol.

Environment and Shell Variables

Environment variables are variables that are available system-wide and are inherited by all spawned child processes and shells.

Shell variables are variables that apply only to the current shell instance. Each shell such as zsh and bash, has its own set of internal shell variables.

Common Environment Variables

USER - The current logged in user.

HOME - The home directory of the current user.

EDITOR - The default file editor to be used. This is the editor that will be used when you type edit in your terminal.

SHELL - The path of the current user's shell, such as bash or zsh.

LOGNAME - The name of the current user.

PATH - A list of directories to be searched when executing commands.

LANG - The current locales settings.

TERM - The current terminal emulation.

MAIL - Location of where the current user's mail is stored.

Commands

env – The command allows you to run another program in a custom environment without modifying the current one. When used without an argument it will print a list of the current environment variables.

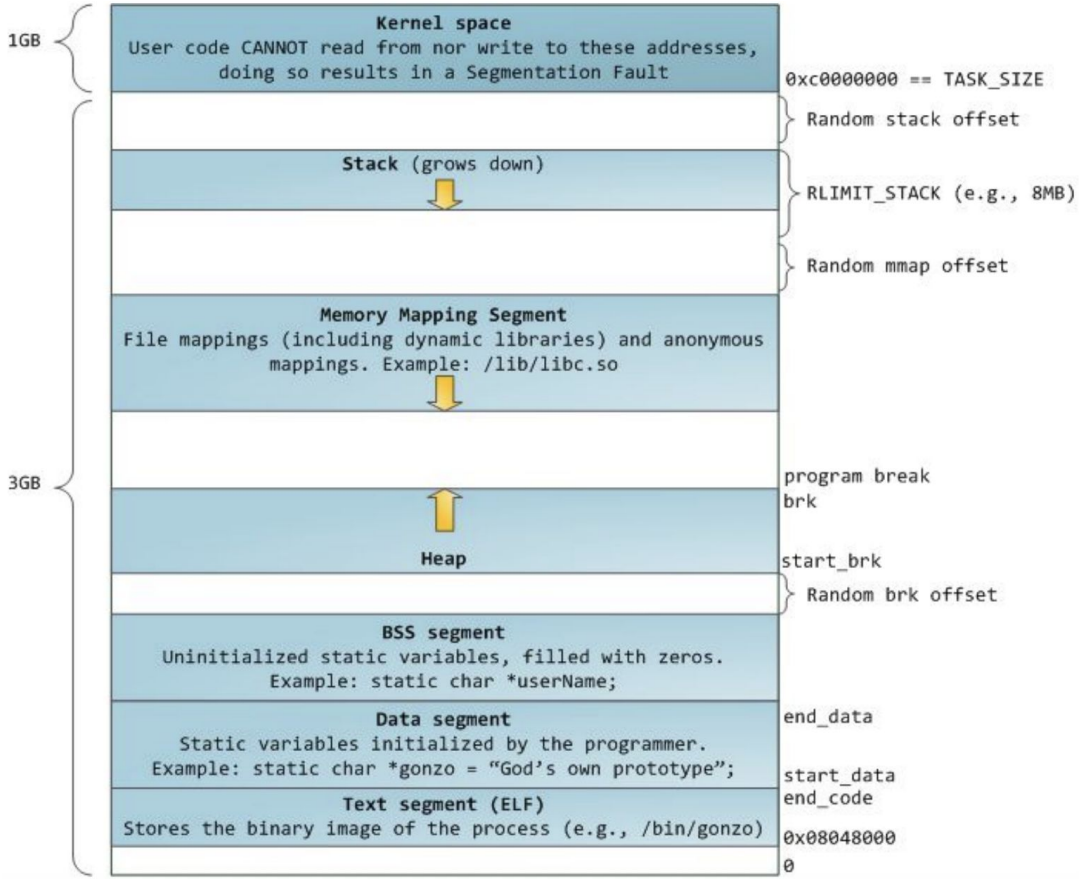
printenv – The command prints all or the specified environment variables.

set – The command sets or unsets shell variables. When used without an argument it will print a list of all variables including environment and shell variables, and shell functions.

unset – The command deletes shell and environment variables.

export – The command sets environment variables

The environment variables live towards the top of the stack, together with command line arguments

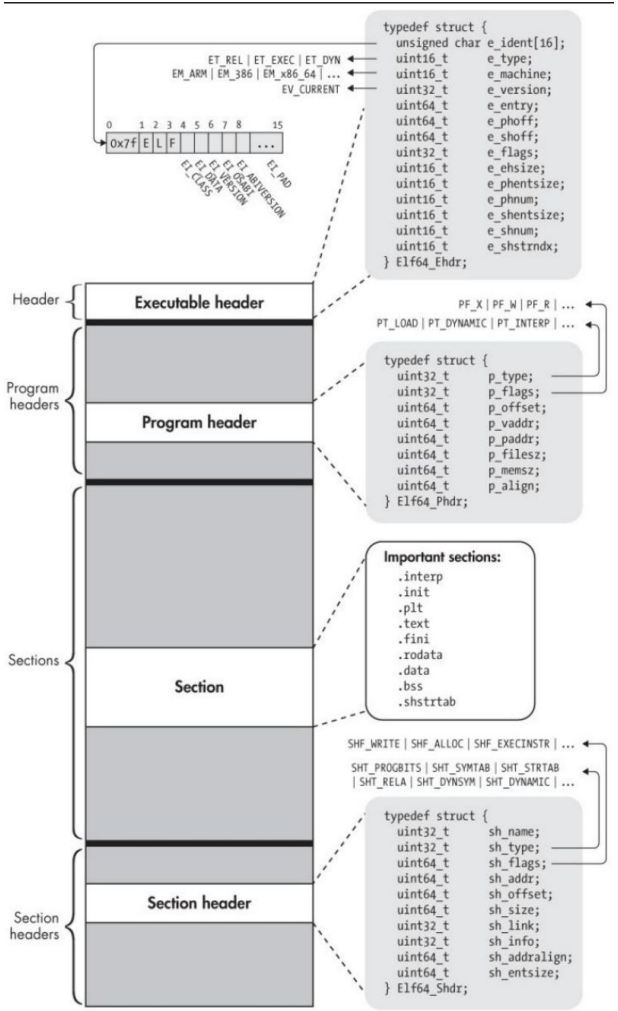


Background Knowledge: Executable and Linkable Format (ELF)

Commands

The **Executable** and **Linkable Format** (**ELF**) is a common standard file format for executable files, object code, shared libraries, and core dumps. Filename extension none, .axf, .bin, .elf, .o, .prx, .puff, .ko, .mod and .so

Contains the program and its data. Describes how the program should be loaded (program/segment headers). Contains metadata describing program components (section headers).



- Executable (a.out), object files (.o), shared libraries (.a), even core dumps.
- Four types of components: an **executable header**, a series of (optional) **program headers**, a number of **sections**, and a series of (optional) **section headers**, one per section.

Executable Header

```
typedef struct {
  unsigned char e_ident[16];   /* Magic number and other info     */ 0x7F ELF ..
  uint16_t      e_type;        /* Object file type Executable, obj, dynamic lib */
  uint16_t      e_machine;     /* Architecture x86-64, Arm                      */
  uint32_t      e_version;     /* Object file version                           */
  uint64_t      e_entry;       /* Entry point virtual address                   */
  uint64_t      e_phoff;       /* Program header table file offset              */
  uint64_t      e_shoff;       /* Section header table file offset              */
  uint32_t      e_flags;       /* Processor-specific flags                      */
  uint16_t      e_ehsize;      /* ELF header size in bytes                      */
  uint16_t      e_phentsize;   /* Program header table entry size               */
  uint16_t      e_phnum;       /* Program header table entry count              */
  uint16_t      e_shentsize;   /* Section header table entry size               */
  uint16_t      e_shnum;       /* Section header table entry count              */
  uint16_t      e_shstrndx;    /* Section header string table index             */
} Elf64_Ehdr;
```

redelf -h a.out

ELF headers

```
arman@aserver:~$ readelf -h /bin/ls
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              DYN (Position-Independent Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x6d30
  Start of program headers:          64 (bytes into file)
  Start of section headers:          140328 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         13
  Size of section headers:           64 (bytes)
  Number of section headers:         31
  Section header string table index: 30
```

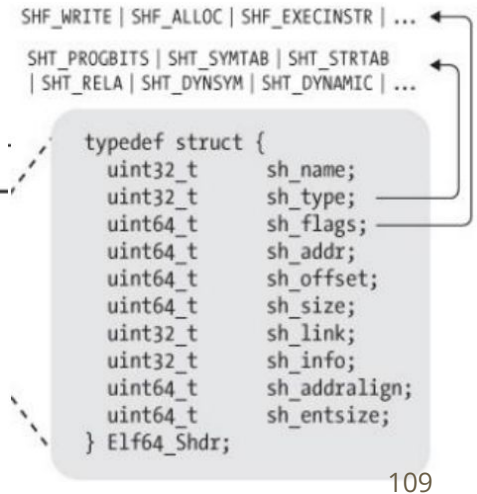
Sections

The code and data in an ELF binary are logically divided into contiguous non-overlapping chunks called sections. The structure of each section varies depending on the contents.

The division into sections is intended to provide a convenient organization for use by the ***linker***.

Section Header Format

```
typedef struct {
  uint32_t  sh_name;      /* Section name (string tbl index)   */
  uint32_t  sh_type;      /* Section type                      */
  uint64_t  sh_flags;     /* Section flags                     */
  uint64_t  sh_addr;      /* Section virtual addr at execution */
  uint64_t  sh_offset;    /* Section file offset               */
  uint64_t  sh_size;      /* Section size in bytes             */
  uint32_t  sh_link;      /* Link to another section           */
  uint32_t  sh_info;      /* Additional section information    */
  uint64_t  sh_addralign; /* Section alignment                 */
  uint64_t  sh_entsize;   /* Entry size if section holds table */
} Elf64_Shdr;
```



Each section is described by its section header

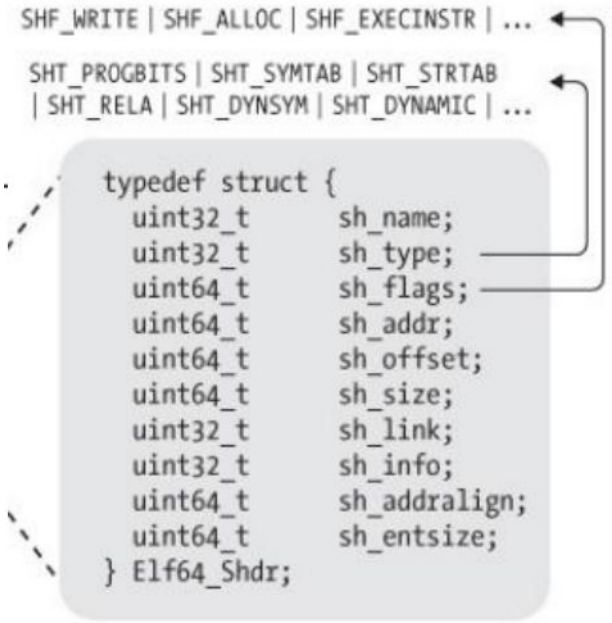
redelf -S a.out

sh_flags

SHF_WRITE: the section is writable at runtime.

SHF_ALLOC: the contents of the section are to be loaded into virtual memory when executing the binary.

SHF_EXECINSTR: the section contains executable instructions.



```
arman@aserver:~/STC/software-security-course-binaries/misc$ readelf -S add_32
There are 31 section headers, starting at offset 0x385c:

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .interp           PROGBITS        000001b4 0001b4 000013 00   A  0   0  1
  [ 2] .note.gnu.bu[...] NOTE            000001c8 0001c8 000024 00   A  0   0  4
  [ 3] .note.gnu.pr[...] NOTE            000001ec 0001ec 00001c 00   A  0   0  4
  [ 4] .note.ABI-tag     NOTE            00000208 000208 000020 00   A  0   0  4
  [ 5] .gnu.hash         GNU_HASH        00000228 000228 000020 04   A  6   0  4
  [ 6] .dynsym           DYNSYM          00000248 000248 0000a0 10   A  7   1  4
  [ 7] .dynstr           STRTAB          000002e8 0002e8 0000bb 00   A  0   0  1
  [ 8] .gnu.version      VERSYM          000003a4 0003a4 000014 02   A  6   0  2
  [ 9] .gnu.version_r    VERNEED         000003b8 0003b8 000040 00   A  7   1  4
  [10] .rel.dyn          REL             000003f8 0003f8 000040 08   A  6   0  4
  [11] .rel.plt          REL             00000438 000438 000020 08  AI  6  24  4
  [12] .init             PROGBITS        00001000 001000 000024 00  AX  0   0  4
  [13] .plt              PROGBITS        00001030 001030 000050 04  AX  0   0 16
  [14] .plt.got          PROGBITS        00001080 001080 000010 10  AX  0   0 16
  [15] .plt.sec          PROGBITS        00001090 001090 000040 10  AX  0   0 16
  [16] .text             PROGBITS        000010d0 0010d0 000259 00  AX  0   0 16
  [17] .fini             PROGBITS        0000132c 00132c 000018 00  AX  0   0  4
  [18] .rodata           PROGBITS        00002000 002000 000044 00   A  0   0  4
  [19] .eh_frame_hdr     PROGBITS        00002044 002044 000054 00   A  0   0  4
  [20] .eh_frame         PROGBITS        00002098 002098 00014c 00   A  0   0  4
  [21] .init_array       INIT_ARRAY      00003ed0 002ed0 000004 04  WA  0   0  4
  [22] .fini_array       FINI_ARRAY      00003ed4 002ed4 000004 04  WA  0   0  4
  [23] .dynamic          DYNAMIC         00003ed8 002ed8 0000f8 08  WA  7   0  4
  [24] .got              PROGBITS        00003fd0 002fd0 000030 04  WA  0   0  4
  [25] .data             PROGBITS        00004000 003000 000008 00  WA  0   0  4
  [26] .bss              NOBITS          00004008 003008 000004 00  WA  0   0  1
  [27] .comment          PROGBITS        00000000 003008 00002a 01  MS  0   0  1
  [28] .symtab           SYMTAB          00000000 003034 000490 10     29  47  4
  [29] .strtab           STRTAB          00000000 0034c4 00027d 00      0   0  1
  [30] .shstrtab         STRTAB          00000000 003741 000118 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings), I (info),
  L (link order), O (extra OS processing required), G (group), T (TLS),
  C (compressed), x (unknown), o (OS specific), E (exclude),
  D (mbind), p (processor specific)
```

redelf -S a.out

Sections

.init: executable code that performs initialization tasks and needs to run before any other code in the binary is executed.

.fini: code that runs after the main program completes.

.text: where the main code of the program resides.

Sections

.rodata section, which stands for “read-only data,” is dedicated to storing constant values. Because it stores constant values, .rodata is not writable.

The default values of initialized variables are stored in the .data section, which is marked as writable since the values of variables may change at runtime.

the .bss section reserves space for uninitialized variables. The name historically stands for “block started by symbol,” referring to the reserving of blocks of memory for (symbolic) variables.

Lazy Binding (.plt, .got, .got.plt Sections)

Binding at Load Time: When a binary is loaded into a process for execution, the dynamic linker resolves references to functions located in shared libraries. The addresses of shared functions were not known at compile time.

In reality - Lazy Binding: many of the relocations are typically not done right away when the binary is loaded but are deferred until the first reference to the unresolved location is actually made.

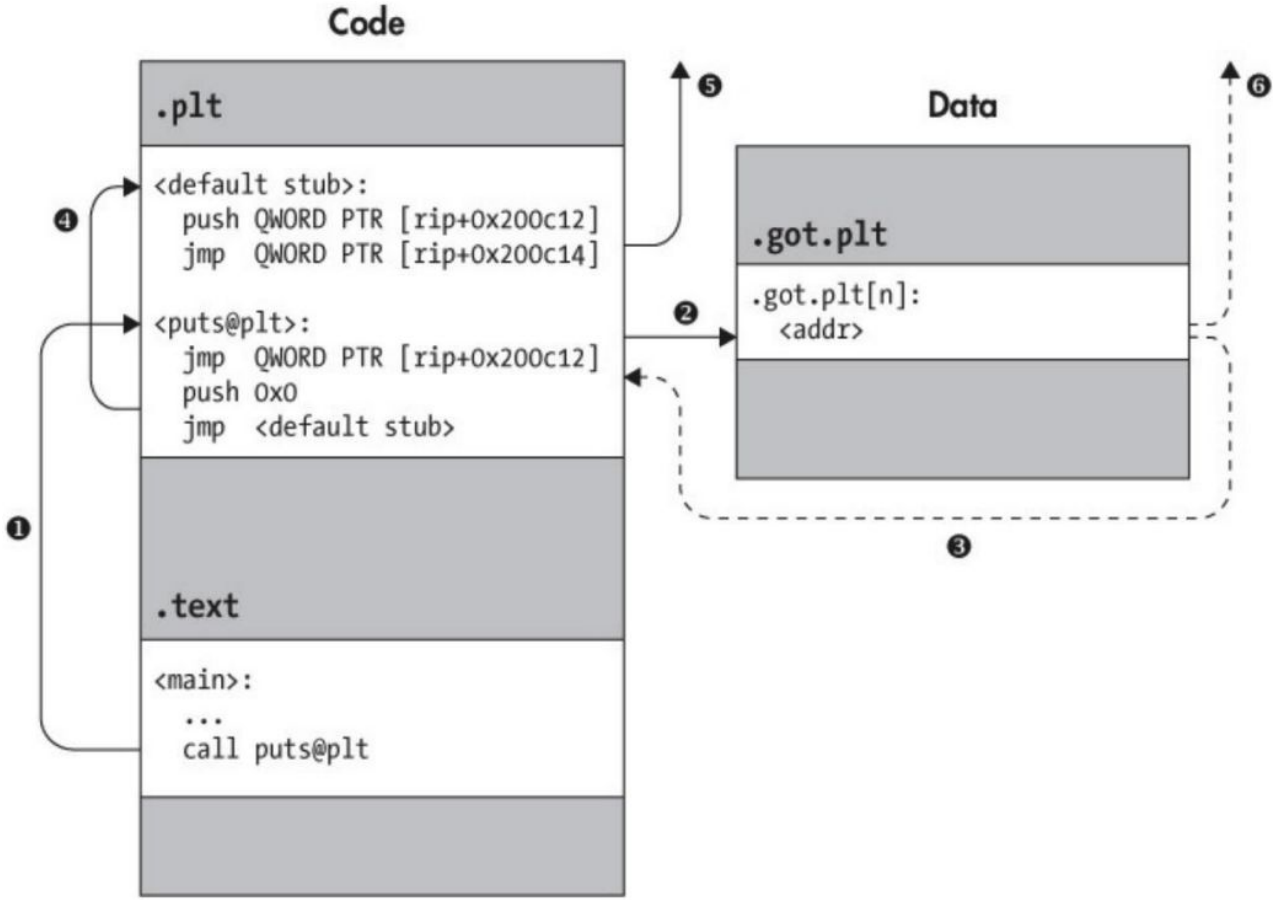
Lazy Binding (.plt, .got, .got.plt Sections)

Lazy binding in Linux ELF binaries is implemented with the help of two special sections, called the Procedure Linkage Table (.plt) and the Global Offset Table (.got).

.plt is a code section that contains executable code. The PLT consists entirely of stubs of a well-defined format, dedicated to directing calls from the .text section to the appropriate library location.

.got.plt is a data section.

Dynamically Resolving a Library Function Using the PLT



Example: Debug misc/lazyb

Section View (Section Header) vs. Segment View (Program Header)

The program header table provides a segment view of the binary, as opposed to the section view provided by the section header table.

The section view of an ELF binary is meant for static linking purposes.

The segment view is used by the operating system and dynamic linker when loading an ELF into a process for execution to locate the relevant code and data and decide what to load into virtual memory.

Segments are simply a bunch of sections bundled together.

Program Header Format

```
typedef struct {
  uint32_t  sh_name;      /* Section name (string tbl index)   */
  uint32_t  sh_type;      /* Section type                      */
  uint64_t  sh_flags;     /* Section flags                     */
  uint64_t  sh_addr;      /* Section virtual addr at execution */
  uint64_t  sh_offset;    /* Section file offset               */
  uint64_t  sh_size;      /* Section size in bytes             */
  uint32_t  sh_link;      /* Link to another section           */
  uint32_t  sh_info;      /* Additional section information    */
  uint64_t  sh_addralign; /* Section alignment                 */
  uint64_t  sh_entsize;   /* Entry size if section holds table */
} Elf64_Shdr;
```

Each section is described by its section header

redelf -S a.out

```
arman@aserver:~/STC/software-security-course-binaries/misc$ readelf -l add_32

Elf file type is DYN (Position-Independent Executable file)
Entry point 0x1160
There are 12 program headers, starting at offset 52

Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  PHDR           0x000034 0x00000034 0x00000034 0x00180 0x00180 R   0x4
  INTERP         0x0001b4 0x000001b4 0x000001b4 0x00013 0x00013 R   0x1
      [Requesting program interpreter: /lib/ld-linux.so.2]
  LOAD           0x000000 0x00000000 0x00000000 0x00458 0x00458 R   0x1000
  LOAD           0x001000 0x00001000 0x00001000 0x00344 0x00344 R E 0x1000
  LOAD           0x002000 0x00002000 0x00002000 0x001e4 0x001e4 R   0x1000
  LOAD           0x002ed0 0x00003ed0 0x00003ed0 0x00138 0x0013c RW  0x1000
  DYNAMIC        0x002ed8 0x00003ed8 0x00003ed8 0x000f8 0x000f8 RW  0x4
  NOTE           0x0001c8 0x000001c8 0x000001c8 0x00060 0x00060 R   0x4
  GNU_PROPERTY   0x0001ec 0x000001ec 0x000001ec 0x0001c 0x0001c R   0x4
  GNU_EH_FRAME   0x002044 0x00002044 0x00002044 0x00054 0x00054 R   0x4
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RW  0x10
  GNU_RELRO      0x002ed0 0x00003ed0 0x00003ed0 0x00130 0x00130 R   0x1

 Section to Segment mapping:
  Segment Sections...
   00
   01     .interp
   02     .interp .note.gnu.build-id .note.gnu.property .note.ABI-tag .gnu.hash .dynsym .dynstr .gnu.version .gnu.version_r .rel.dyn .rel.pl
   03     .init .plt .plt.got .plt.sec .text .fini
   04     .rodata .eh_frame_hdr .eh_frame
   05     .init_array .fini_array .dynamic .got .data .bss
   06     .dynamic
   07     .note.gnu.build-id .note.gnu.property .note.ABI-tag
   08     .note.gnu.property
   09     .eh_frame_hdr
   10
   11     .init_array .fini_array .dynamic .got
```

Background Knowledge: Manual Binary Analysis Tools

Tools for this class

file

readelf

strings

nm

objdump

GDB

[optional] IDA Pro

[optional] ghidra

[optional] Binary Ninja

GDB Cheat Sheet

Start gdb using:

```
gdb <binary>
```

Pass initial commands for gdb through a file

```
gdb <binary> –x <initfile>
```

To start the program and breakpoint at main()

```
start <argv>
```

To start the program and breakpoint at _start

```
start <argv>
```

To run the program without breakpoint

```
r <argv>
```

Use another progrom’s output as stdin in GDB:

```
r <<< $(python2 -c "print '\x12\x34'*5")
```

GDB Cheat Sheet

Set breakpoint at address:

```
b *0x80000000
```

Set breakpoint at beginning of a function:

```
b main
```

....

```
b <filename:line number>
b <line number>
```

Disassemble 10 instructions from an address:

```
x/10i 0x80000000
```

Exam 15 dword (w) from an address; show hex (x):

```
x/15wx 0x80000000
```

Exam 3 qword (g) from an address; show hex (x):

```
x/3gx 0x80000000
```

GDB Cheat Sheet

To show breakpoints

```
info b
```

To remove breakpoints

```
clear <function name>
clear *<instruction address>
clear <filename:line number>
clear <line number>
```

GDB Cheat Sheet

Use “examine” or “x” command

x/32xw <memory location> to see memory contents at memory location, showing 32 hexadecimal words

x/5s <memory location> to show 5 strings (null terminated) at a particular memory location

x/10i <memory location> to show 10 instructions at particular memory location

See registers

info reg

Step an instruction

si

GDB Script

Use “examine” or “x” command

x/32xw <memory location> to see memory contents at memory location, showing 32 hexadecimal words

x/5s <memory location> to show 5 strings (null terminated) at a particular memory location

x/10i <memory location> to show 10 instructions at particular memory location

See registers

info reg

Step an instruction

si

Shell Cheat Sheet

Run a program and use another program’s output as a parameter

```
program $(python2 -c "print '\x12\x34'*5")
```

Reading

1. https://ig.thc.org/how-does-linux-start-a-process