System Security - Attack and Defense for Binaries

CS 4390/5390, Spring *2026*

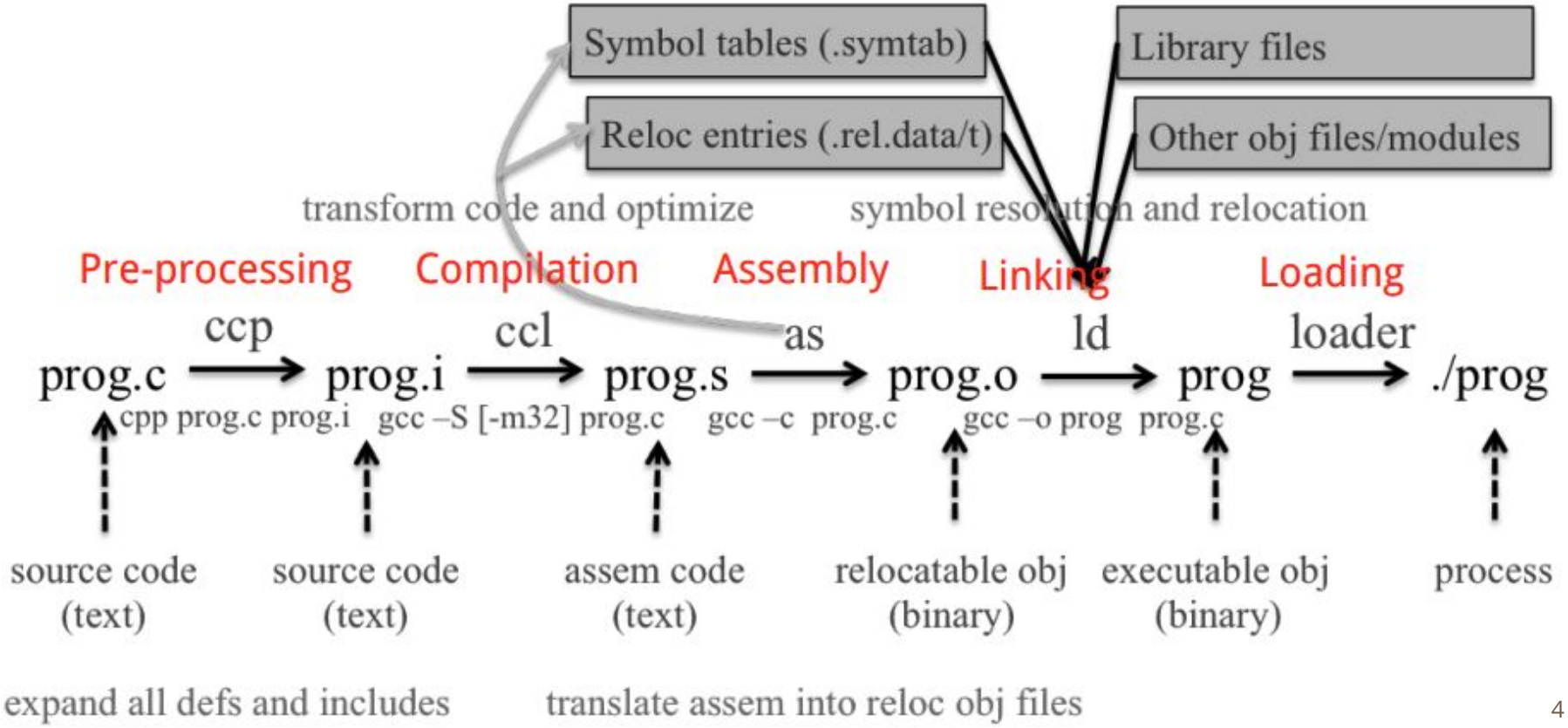
Instructor: MD Armanuzzaman (***Arman***)

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

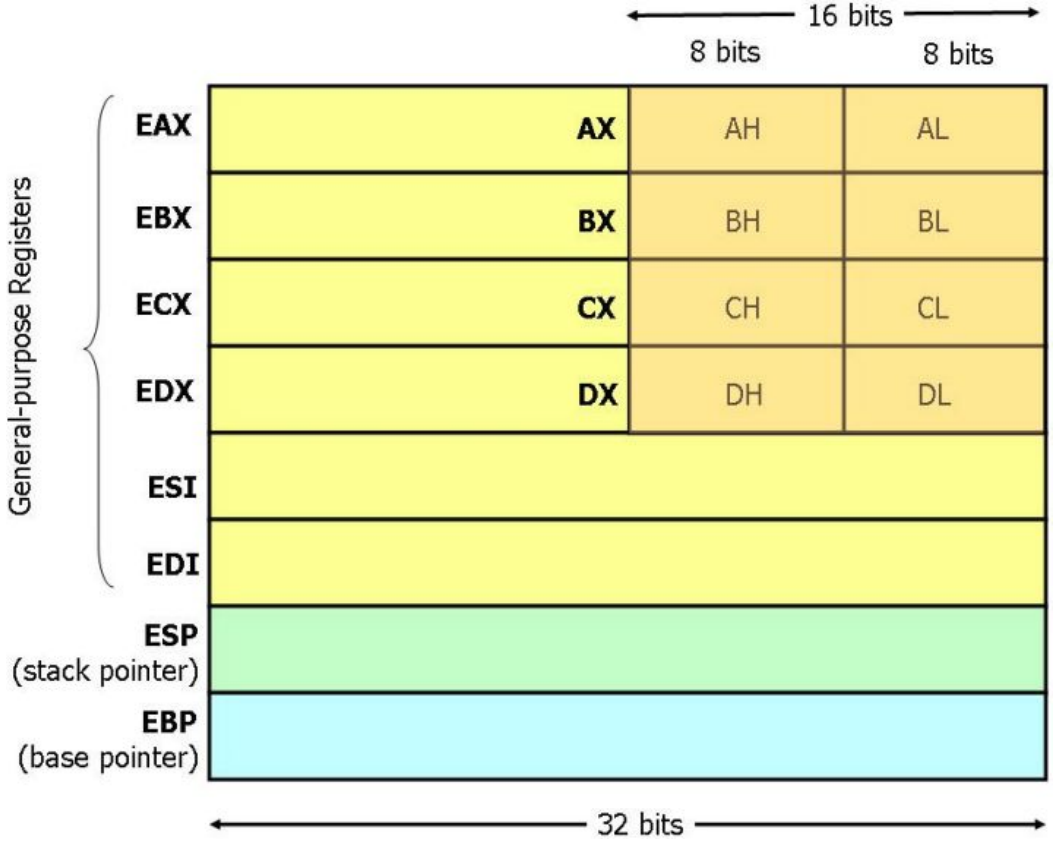
Address 0	0x12
Address 1	0x34
Address 2	0x56
Address 3	0x78

Base Registers

There are

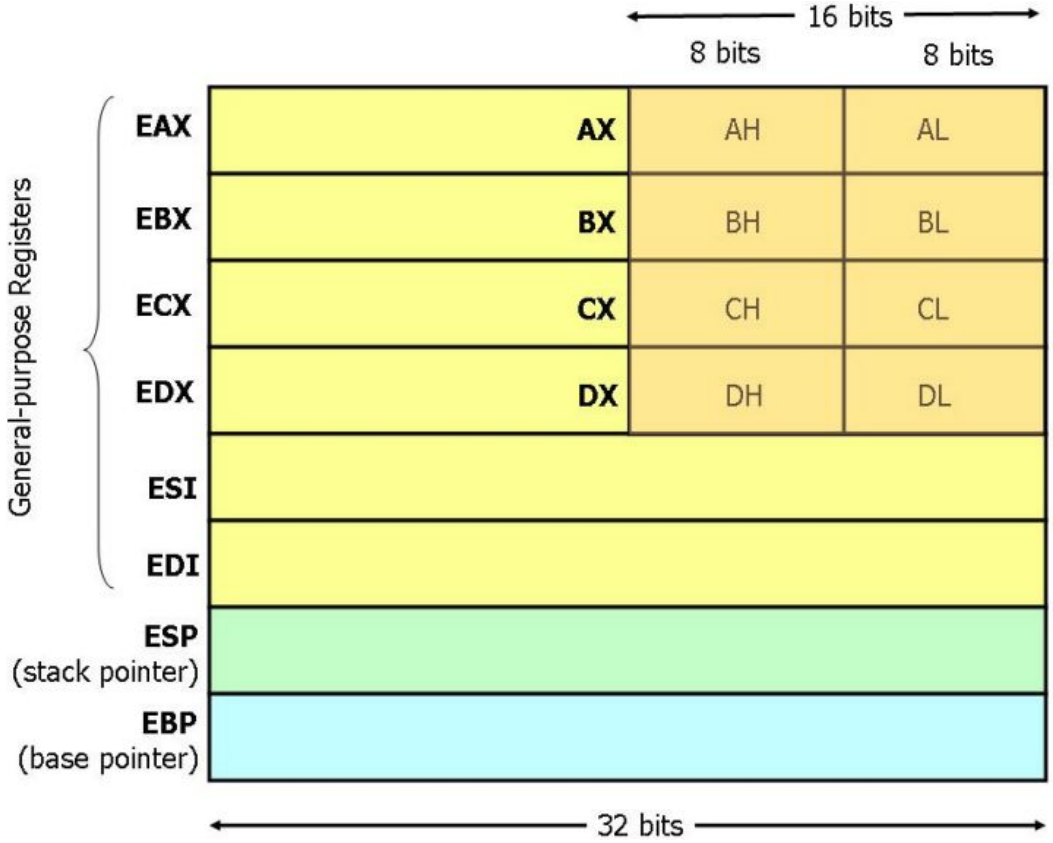
- Eight 32-bit “general-purpose” registers,
- One 32-bit EFLAGS register,
- One 32-bit instruction pointer register (eip), and
- Other special-purpose registers.

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

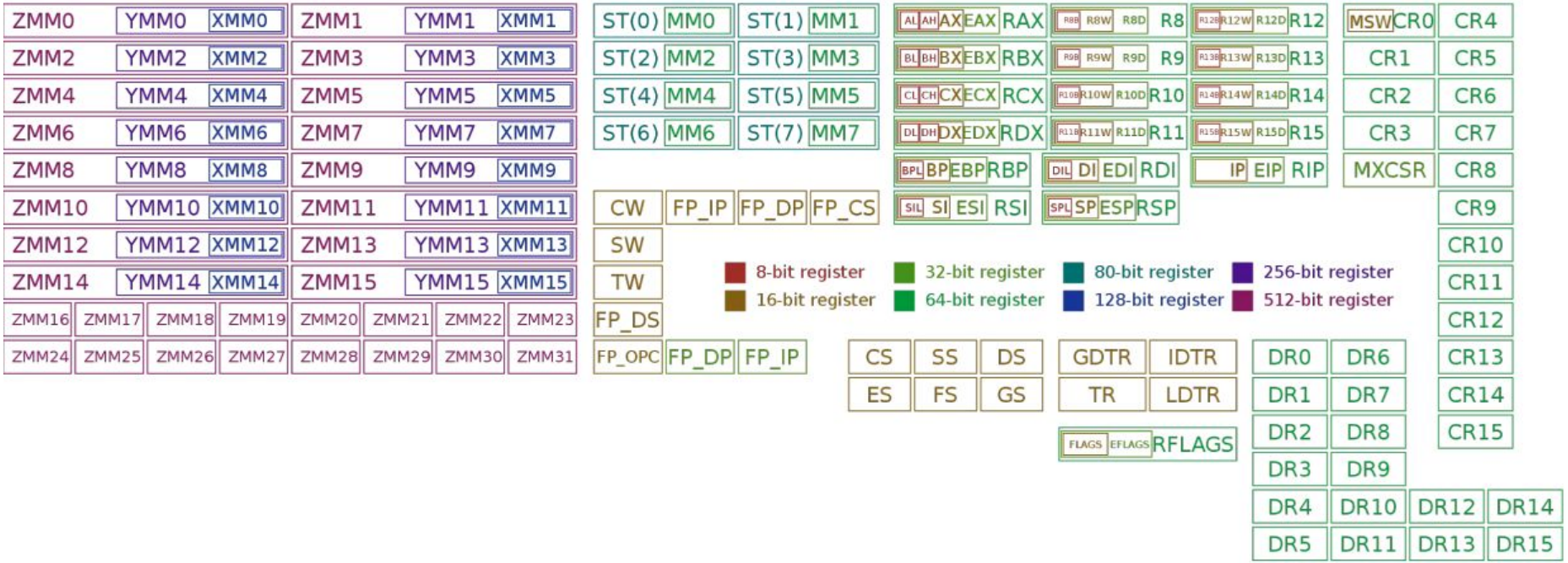
SF – sign flag

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.

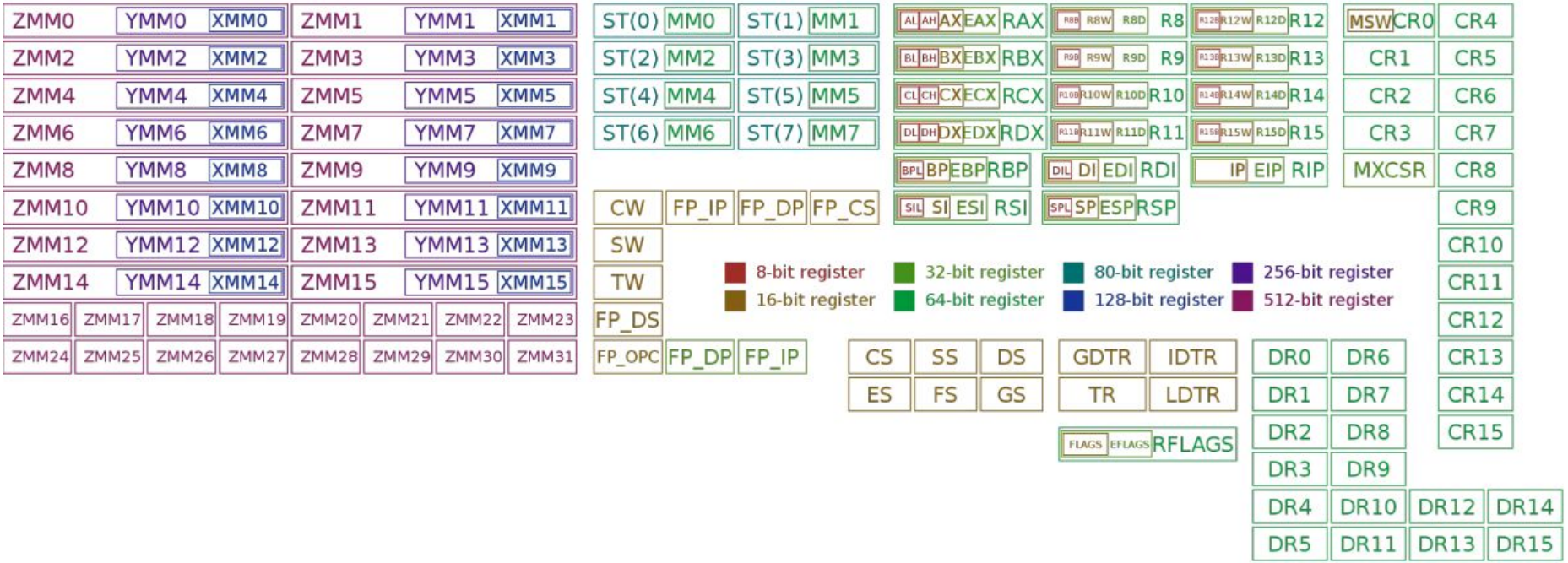
Equivalent to:

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
```

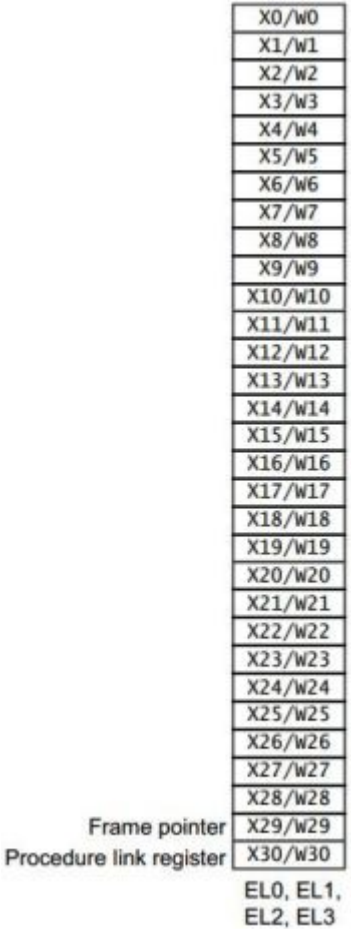
x86-64

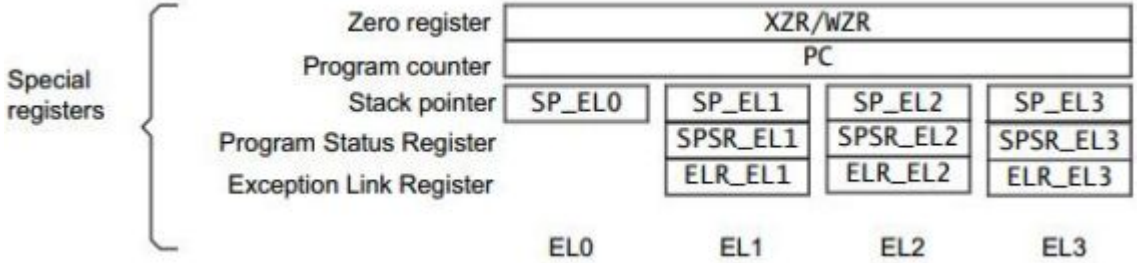
```
0000000000001220 <ladd>:
    1220: f3 0f 1e fa endbr64
    1224: 48 8b 07 mov rax,QWORD PTR [rdi]
    1227: 48 01 f0 add rax,rsi
    122a: c3 ret
```

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

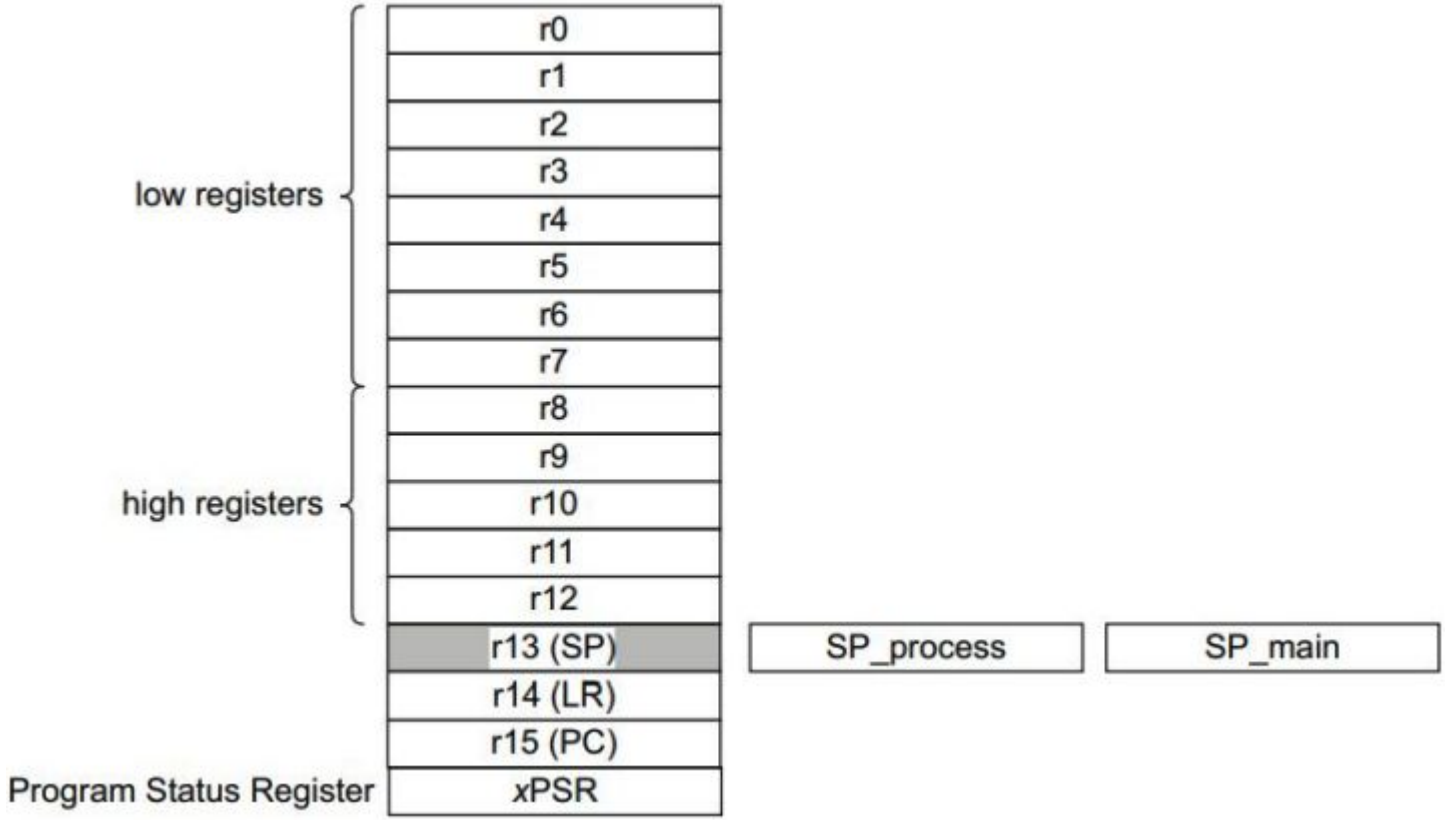
Background Knowledge: ARM Cortex-A/M Architecture

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

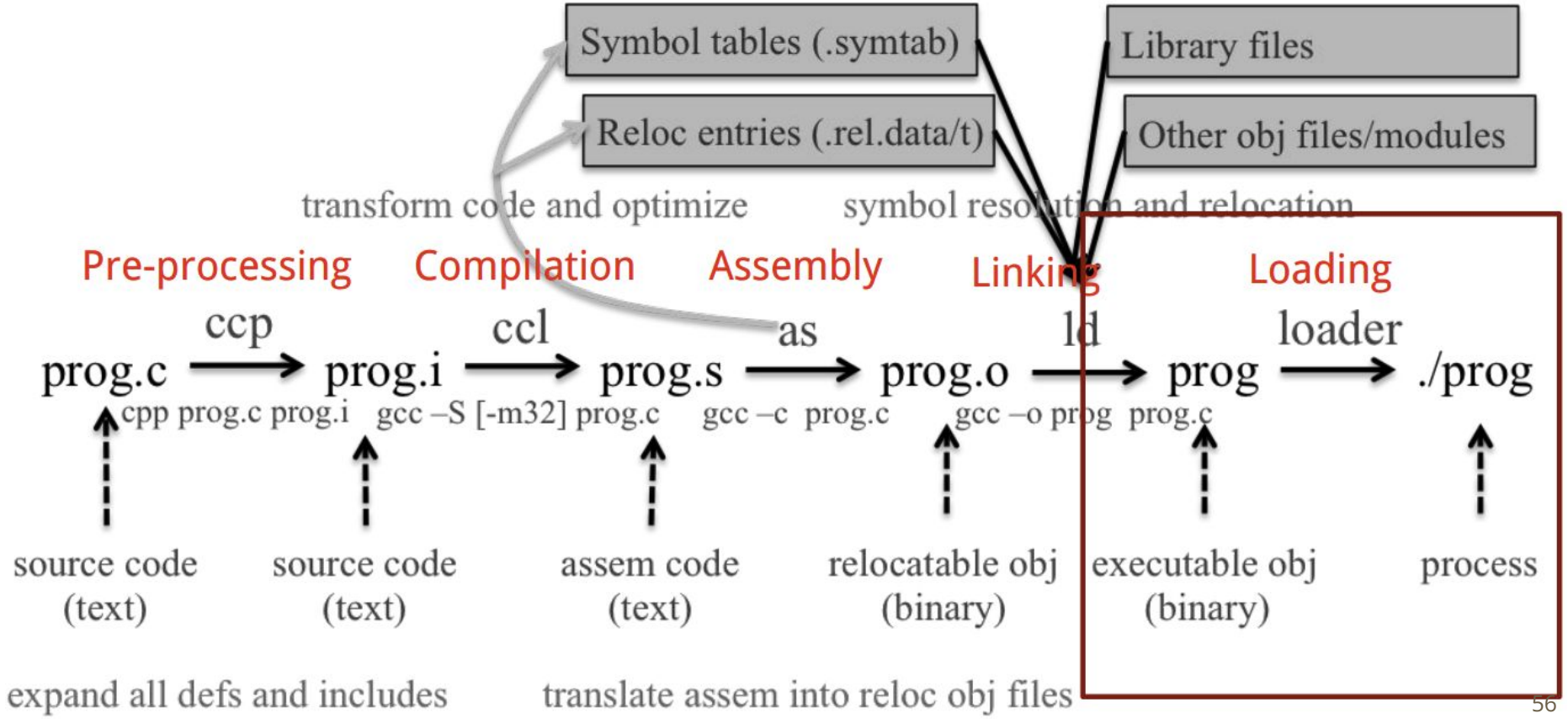
```
arman@aserver:~/STC/software-security$ ls -l
total 80
-rwxrwxr-x  1 arman arman 1401 Jan 13 18:09 auth.py
drwxrwxr-x 15 arman arman 4096 Jan 13 18:09 challenges
drwxrwxr-x  3 arman arman 4096 Jan 13 18:09 conf
-rw-rw-r--  1 arman arman  264 Jan 14 02:18 config.env
drwxrwxr-x  7 arman arman 4096 Jan 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);
}
```

Reading

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