CÁTEDRA
INCIBE-UEX-EPCC

> Wifi: HACKTHEBOX
> Pw: Hack2025

CÁTEDRA
INCIBE-UEX-EPCC

> Taller de introducción a la Ingeniería Inversa

> whoami

Alberto López Trigo

> Investigador en la UEx
> Hack The Box Ambassador

/in/chillredchilli

chillredchilli

@ChillRedChilli

CÁTEDRA INCIBE-UEX-EPCC

HACKTHEBOX Meetup Cáceres

> ¿Qué vamos a ver?

01 Introducción

02 Análisis estático

03 Arquitectura x86_64

04 Ghidra

05 Actividad 1

06 Actividad 2

> ¿Qué es la ingeniería inversa de software?

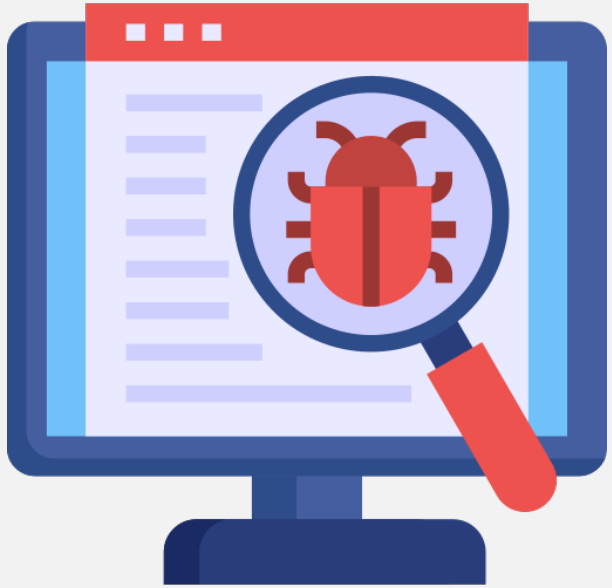
> La ingeniería inversa de software (SRE) es la práctica de analizar un sistema de software, ya sea en su totalidad o en parte, para extraer información sobre su diseño e implementación. Un escenario típico de SRE implicaría un módulo de software que ha funcionado durante años \[...]; desafortunadamente, el código fuente de la aplicación se ha perdido y lo que queda es código «nativo» o «binario».

– Cipresso, T., Stamp, M. (2010). Software Reverse Engineering. In: Stavroulakis, P., Stamp, M. (eds) Handbook of Information and Communication Security.

> ¿Qué es la ingeniería inversa de software?

Lo que quiere decir:

- Analizar un componente de software para extraer información a partir de él, sin tener disponible el código fuente.
- Se usa para recrear y entender la funcionalidad del componente.
- También sirve para encontrar bugs y vulnerabilidades.
- Normalmente se empieza por la capa de abstracción más baja (p. ej. código máquina)



> Aplicaciones prácticas

Análisis del malware

Bug bounty / explotación de binarios

Ingeniería del software / aplicaciones heredadas

> Proceso de compilación

programa.c

```c
#include <stdio.h>

// TODO: Hack the world
int main() {
  printf("Hola, Mundo!");
  return 0;
}
```

Preprocessor

programa.i

- Eliminación de comentarios
- Expansión de macros
- Expansión de los archivos incluidos
- Compilación condicional

Compiler

programa.s

- Instrucciones Ensamblador

Assembler

programa.o

- De ensamblador a código máquina

linker

programa.exe

- Combina archivos .o en un ejecutable.
- Referencias a bibliotecas externas.
- Enlace de funciones y variables.

> Proceso de compilación

Código C

```c
#include <stdio.h>

// TODO: Hack the world
int main() {
  printf("Hola, Mundo!");
  return 0;
}
```

Código C (Expandido)

```c
# 0 "programa.c"
# 0 "<interno>"
# 0 "<línea-de-órdenes>"
# 1 "/usr/include/stdc-predef.h" 1 3 4
[...]
extern int printf (const char *__restrict __format, ...);
[...]
# 973 "/usr/include/stdio.h" 3 4
# 2 "programa.c" 2

# 4 "programa.c"
int main() {
  printf("Hola, Mundo!");
  return 0;
}
```

> Proceso de compilación

Código C (Expandido)

```
# 0 "programa.c"
# 0 "<interno>"
# 0 "<línea-de-órdenes>"
# 1 "/usr/include/stdc-predef.h" 1 3 4
[...]
extern int printf (const char *__restrict __format, ...);
[...]
# 973 "/usr/include/stdio.h" 3 4
# 2 "programa.c" 2

# 4 "programa.c"
int main() {
  printf("Hola, Mundo!");
  return 0;
}
```

Ensamblador

```
  .file  "programa.c"
  .text
  .section  .rodata
.LC0:
  .string  "Hola, Mundo!"
  .text
  .globl  main
  .type  main, @function
main:
.LFB0:
  .cfi_startproc
  pushq  %rbp
  .cfi_def_cfa_offset 16
  .cfi_offset 6, -16
  movq  %rsp, %rbp
  .cfi_def_cfa_register 6
  leaq  .LC0(%rip), %rax
  movq  %rax, %rdi
  movl  $0, %eax
  call  printf@PLT
  movl  $0, %eax
  popq  %rbp
  .cfi_def_cfa 7, 8
  ret
  .cfi_endproc
.LFE0:
  .size  main, .-main
  .ident  "GCC: (GNU) 15.2.1 20250813"
  .section  .note.GNU-stack,"",@progbits
```

> Proceso de compilación

Ensamblador

```
  .file  "programa.c"
  .text
  .section  .rodata
.LC0:
  .string  "Hola, Mundo!"
  .text
  .globl  main
  .type  main, @function
main:
.LFB0:
  .cfi_startproc
  pushq  %rbp
  .cfi_def_cfa_offset 16
  .cfi_offset 6, -16
  movq  %rsp, %rbp
  .cfi_def_cfa_register 6
  leaq  .LC0(%rip), %rax
  movq  %rax, %rdi
  movl  $0, %eax
  call  printf@PLT
  movl  $0, %eax
  popq  %rbp
  .cfi_def_cfa 7, 8
  ret
  .cfi_endproc
.LFE0:
  .size  main, .-main
  .ident  "GCC: (GNU) 15.2.1 20250813"
  .section  .note.GNU-stack,"",@progbits
```

Bytecode

```
000000C0 01 7A 52 00:01 78 10 01|1B 0C 07 08:90 01 00 00
000000D0 1C 00 00 00:1C 00 00 00|00 00 00 00:1F 00 00 00
```

> Proceso de compilación

Bytecode

```
000000C0 01 7A 52 00:01 78 10 01|1B 0C 07 08:90 01 00 00
000000D0 1C 00 00 00:1C 00 00 00|00 00 00 00:1F 00 00 00
```

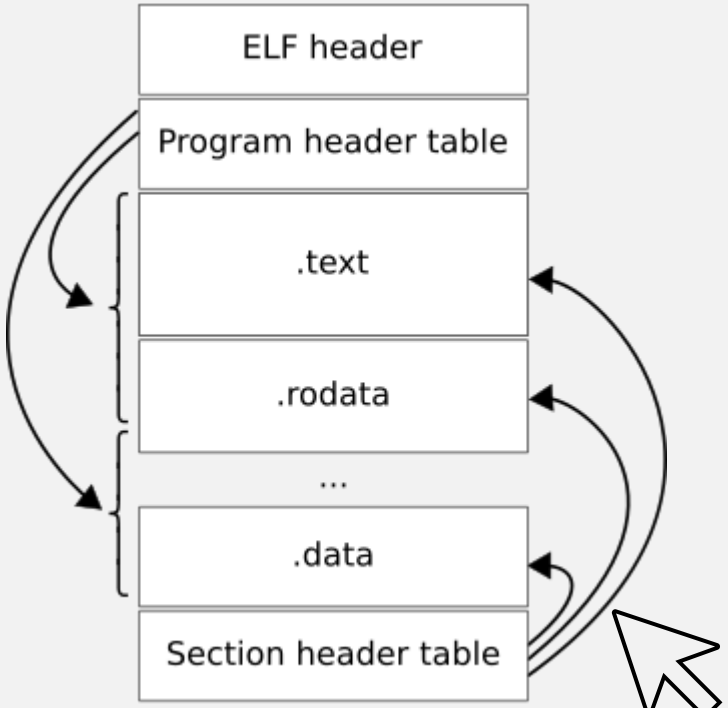
Archivo ELF

```
000000C0 01 7A 52 00:01 78 10 01|1B 0C 07 08:90 01 00 00
000000D0 1C 00 00 00:1C 00 00 00|00 00 00 00:1F 00 00 00
```

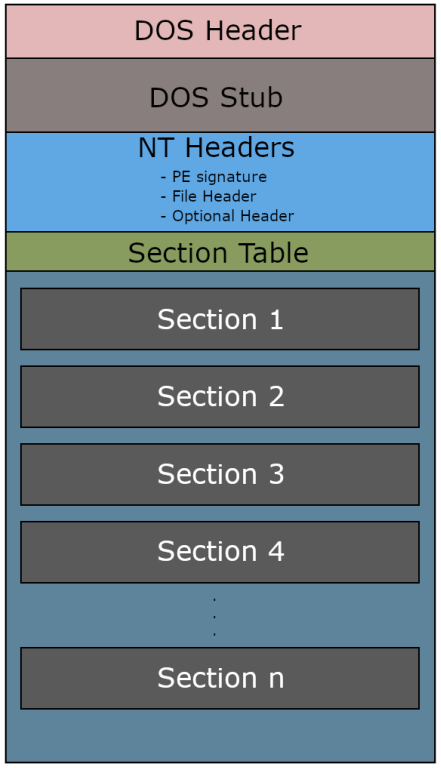
> Formatos de salida

Principales formatos de archivos ejetubales

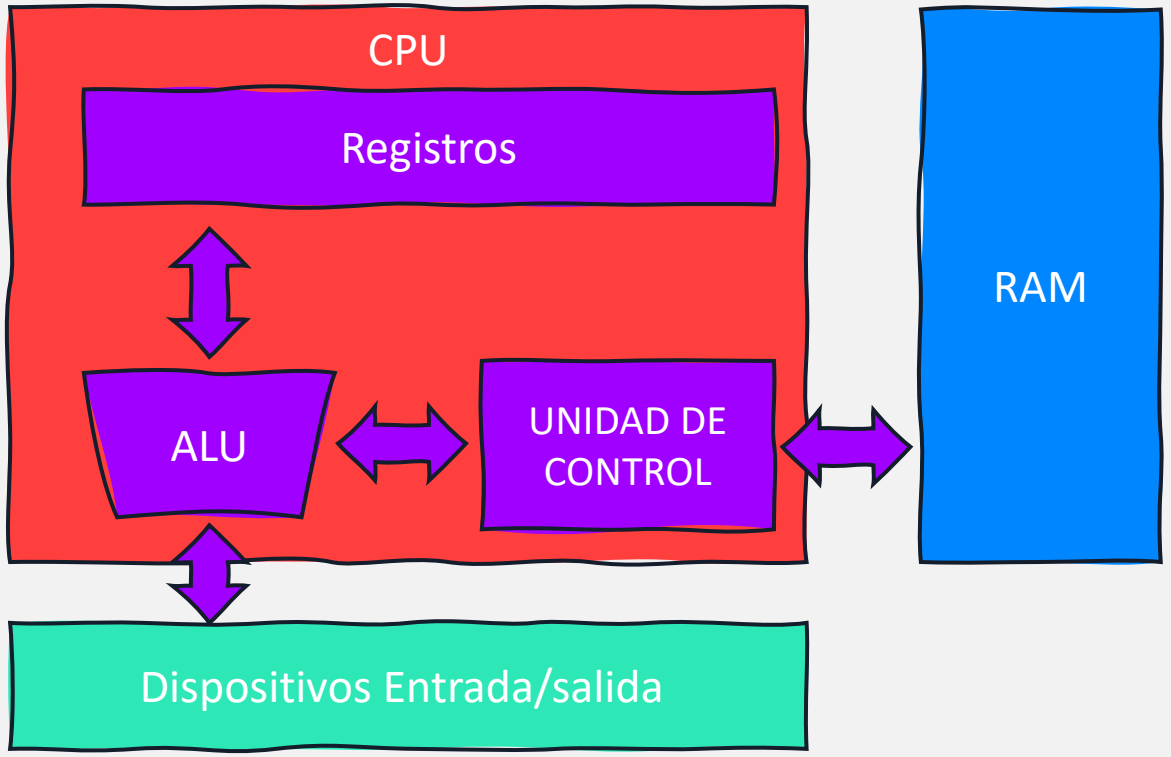
- **Portable Executable** (PE) para Windows.
- **Extensible Linkable Format** (ELF) para Linux.
- **Mach object** (Mach-O) para macOS.
- Etc.



Hoy nos centraremos en archivos ELF



> Arquitectura Von Neumann

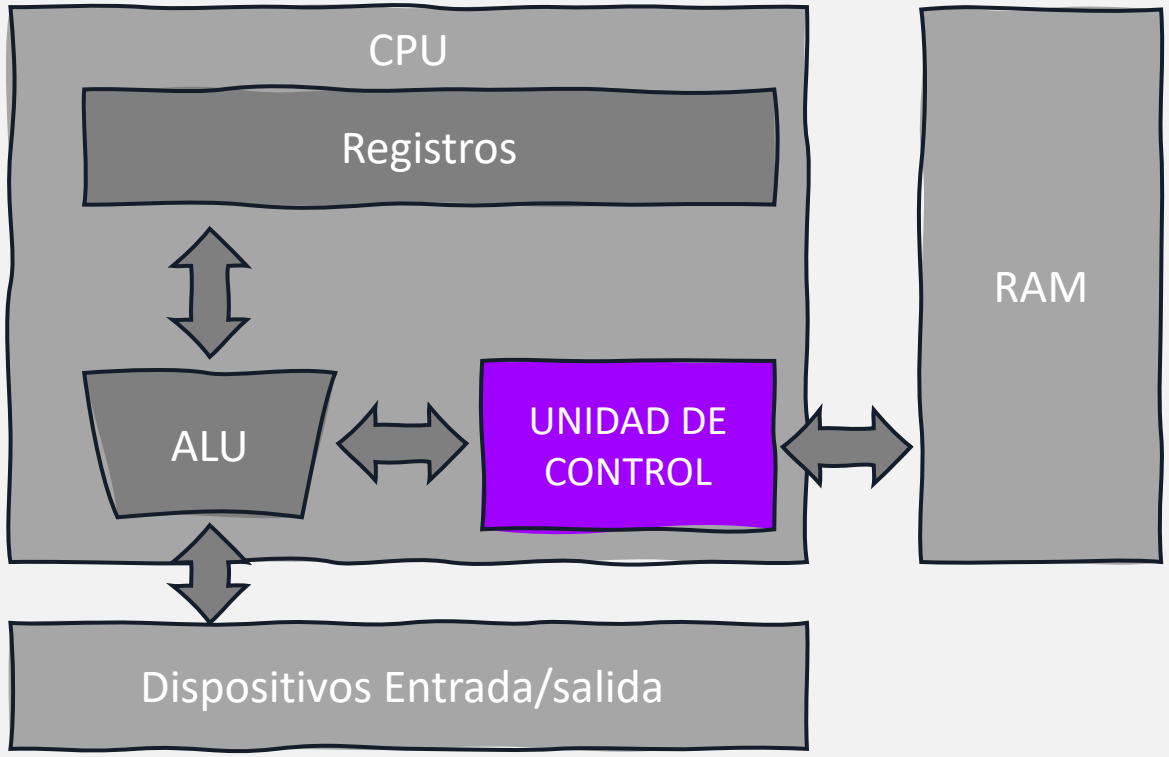


> Arquitectura Von Neumann

Unidad de control

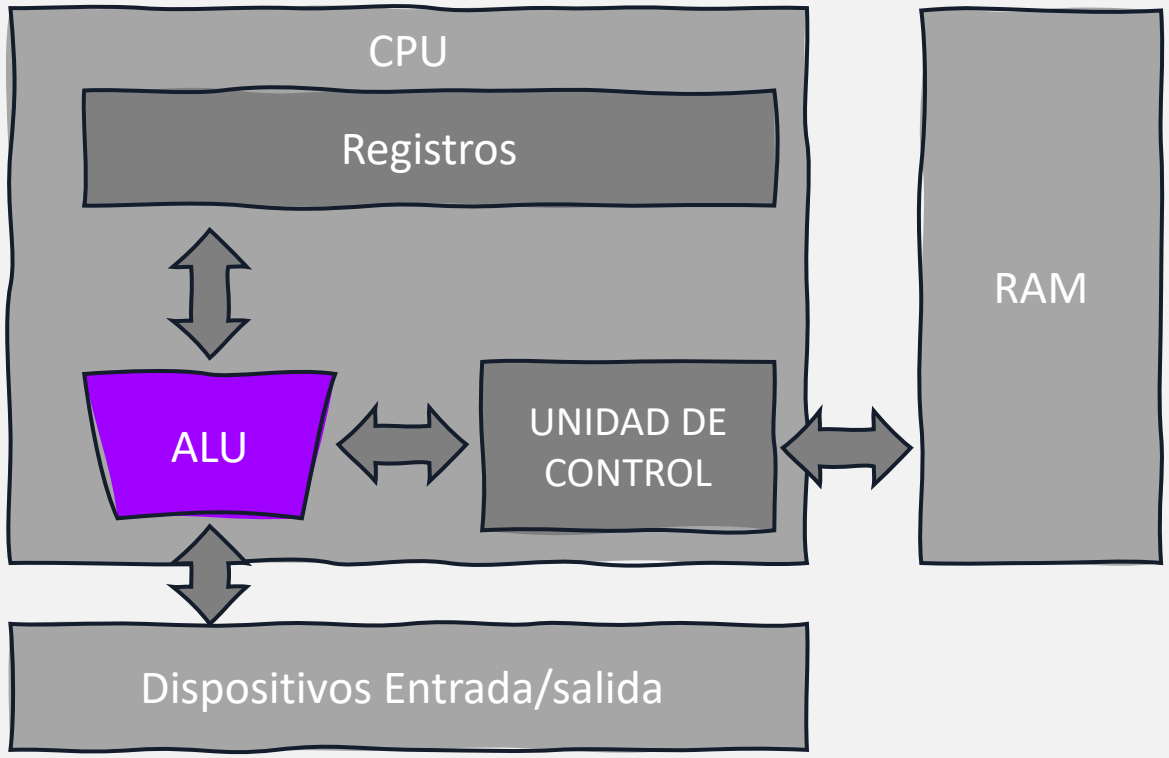
La unidad de control gestiona el procesador enviando señales de control.

- Decide cómo deben moverse los datos dentro del ordenador
- controla las operaciones de entrada y salida
- Obtiene instrucciones desde la RAM con un puntero de instrucción



Unidad Lógica Aritmética

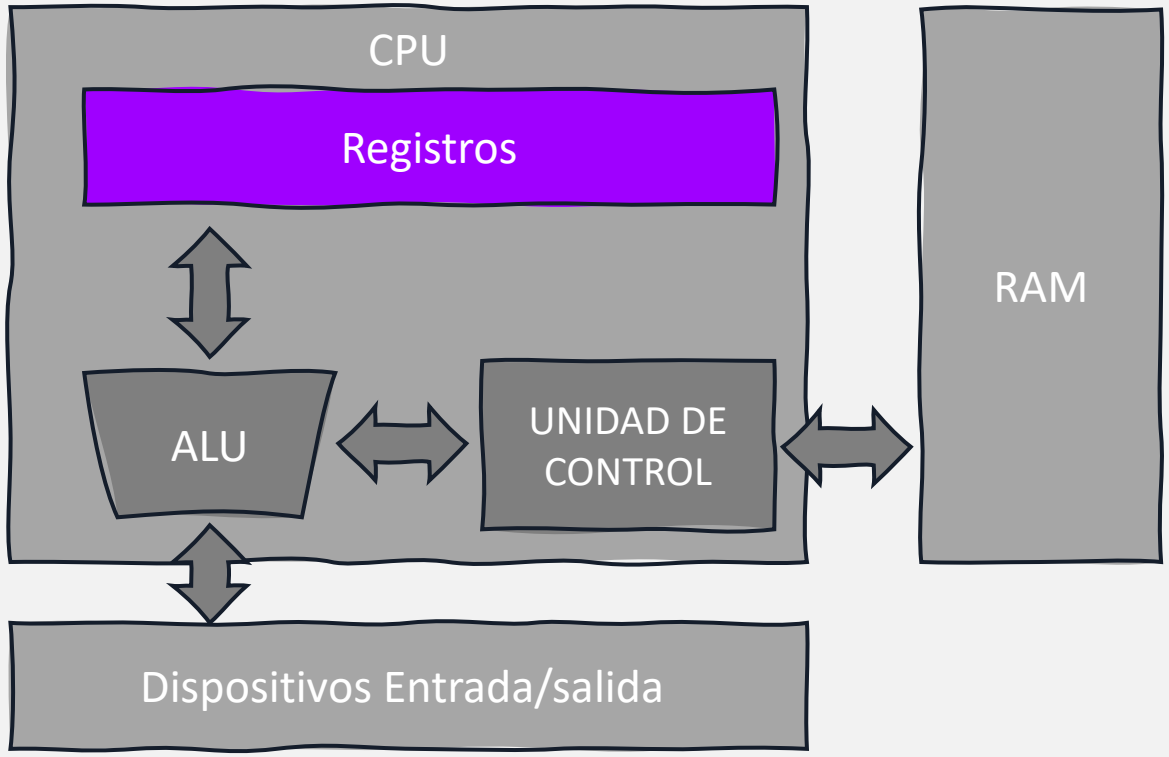
La unidad de lógica aritmética ejecuta las instrucciones y genera resultados que se guardan en registros o en memoria



> Arquitectura Von Neumann

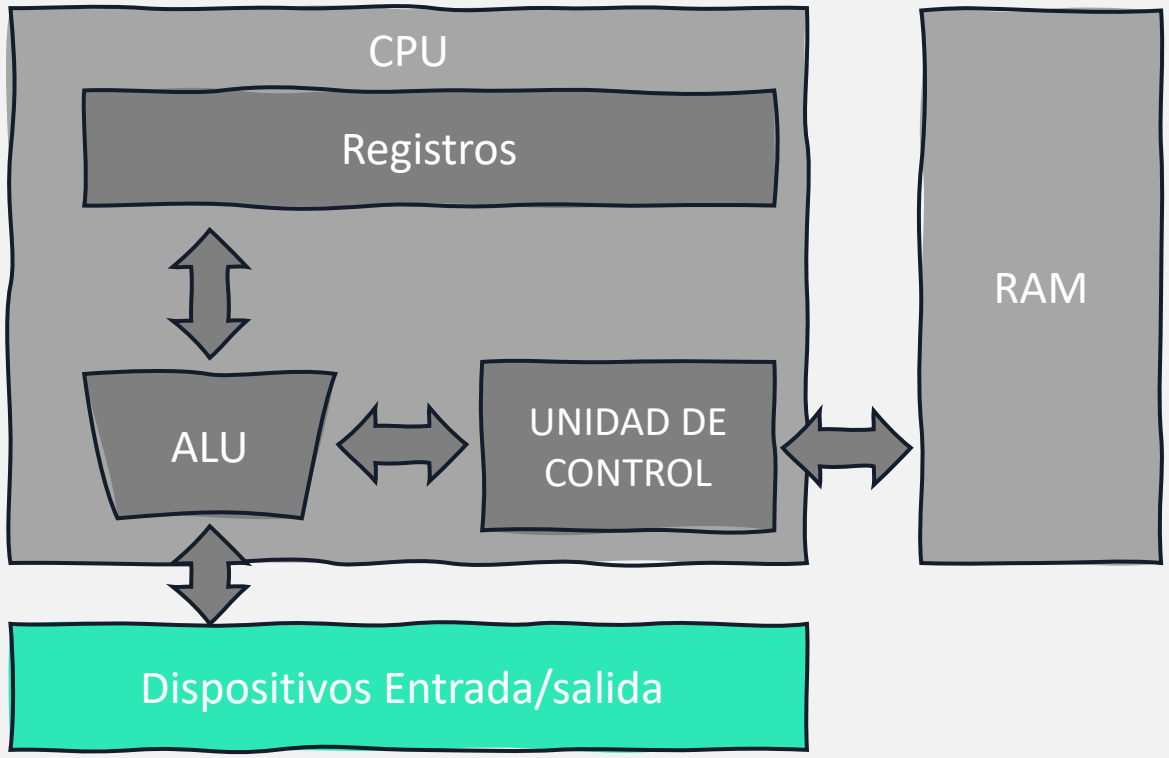
Registros

Los registros son las unidades básicas de almacenamiento de datos de la CPU y se utilizan a menudo para ahorrar tiempo, de modo que la CPU no tenga que acceder a la RAM.



Dispositivos Entrada/salida

Una interfaz de entrada/salida (E/S) interactúa con dispositivos como discos duros, teclados y monitores.

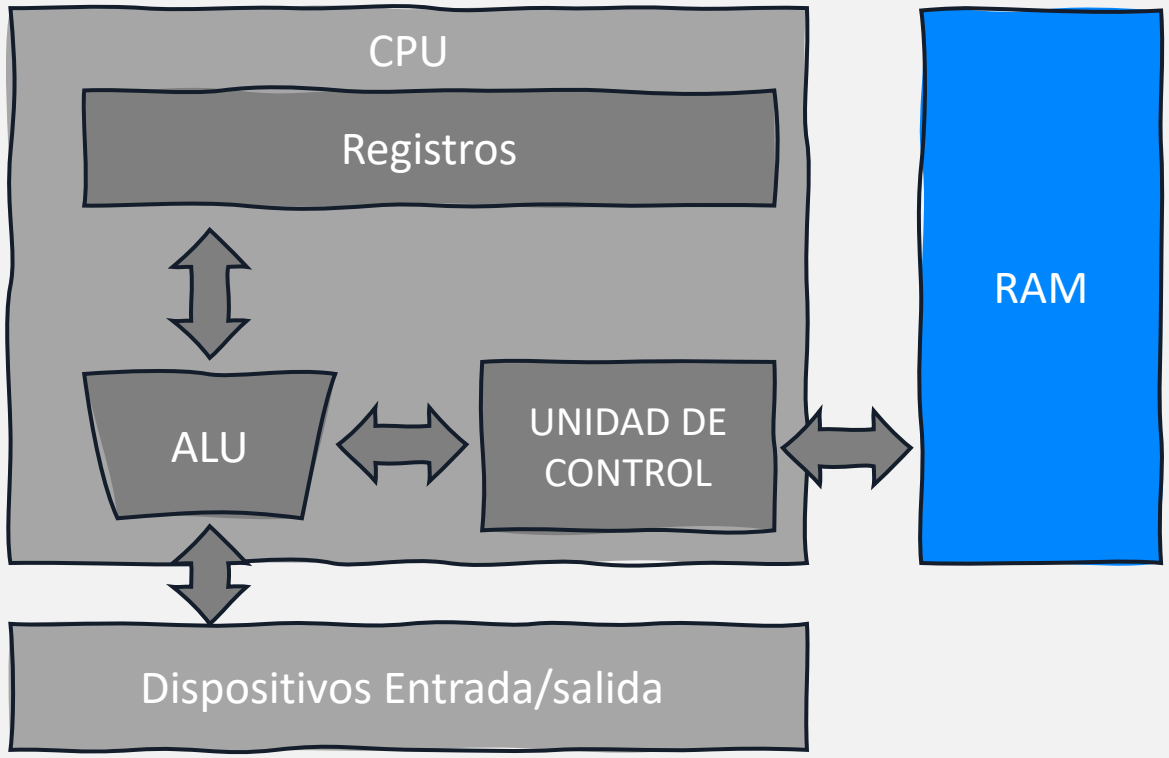


> Arquitectura Von Neumann

Random Access Memory (RAM)

La memoria principal del sistema (RAM) almacena todos los datos y el código.

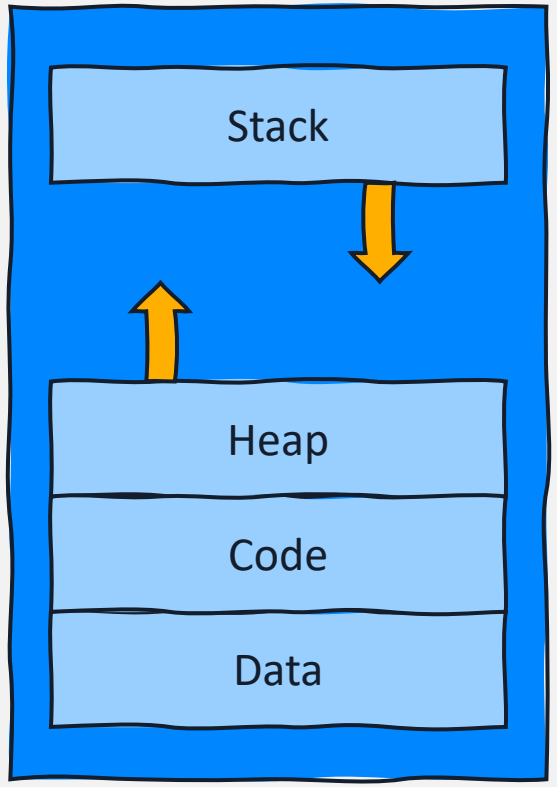
- Se pierden los datos al apagar el ordenador.
- Es el lugar en el que se cargan los procesos
  - La memoria se puede dividir en varias partes



> Regiones de la memoria

Partes de un programa cargado en memoria:

- Stack (Pila): Variables locales y parámetros
- Heap: Reserva de memoria dinámica en tiempo de ejecución
- Code: Contiene las instrucciones que se pasan a la CPU para ser ejecutadas
- Data: Valores estáticos o globales que no cambias y son usados por el programa desde distintas partes



Formato Intel

Mnemómico	Operador destino	Operador fuente
mov	ecx	0x42

- Mueve el valor 66 (0x42) al registro ECX
- En memoria sería: B9 42 00 00 00
  - B9 → mov ecx
  - 42 00 00 00 → valor 0x42

Formato Intel vs AT&T

- Intel
  - Sin prefijos para registros
  - El destino es el primer operando
  - La memoria se especifica con []
- AT&T
  - Los registros utilizan %, las constantes $
  - El destino es el segundo operando
  - La memoria se especifica con ()

```
Mov eax, [ecx]
```

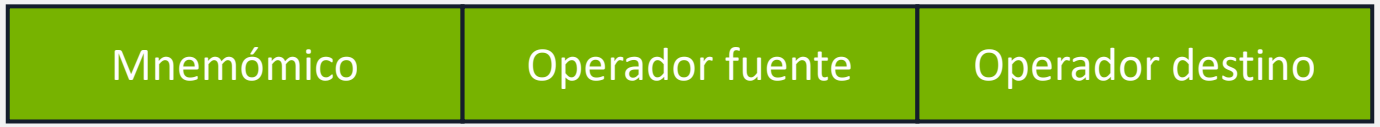
```
Movl (%ecx), %eax
```

Formato AT&T

Mnemómico	Operador fuente	Operador destino
mov	$0x42	%ecx

- Mueve el valor 66 (0x42) al registro ECX
- En memoria sería: B9 42 00 00 00
  - B9 → mov ecx
  - 42 00 00 00 → valor 0x42

Composición de una instrucción



Mnemónico

- Especifica a la CPU la operación que debe hacer
  - Add, sub, mov, push, pop, jmp, call, lea, etc.

Operandos

- Pueden ser:
  - Registros
  - Dirección de memoria
  - Constantes
- Depende de la instrucción, puede haber 2, 1 o Ningún operando.
- Solo se permite un operando de memoria explícito

> Endianness

El *endianness* de los datos describe si el byte más significativo (big-endian) o el menos significativo (little-endian) se ordena primero (en la dirección más pequeña) dentro de un elemento de datos más grande.

X86_64 utiliza Little-endian

127.0.0.1 → 0x7F000001

Big-endian

Dirección en memoria	+0	+1	+2	+3
Valor del Byte	0x7F	0x00	0x00	0x01

Little-endian

Dirección en memoria	+0	+1	+2	+3
Valor del Byte	0x01	0x00	0x00	0x7F

> Registros

Los registros son regiones de memoria pequeñas que se encuentran dentro de la CPU.

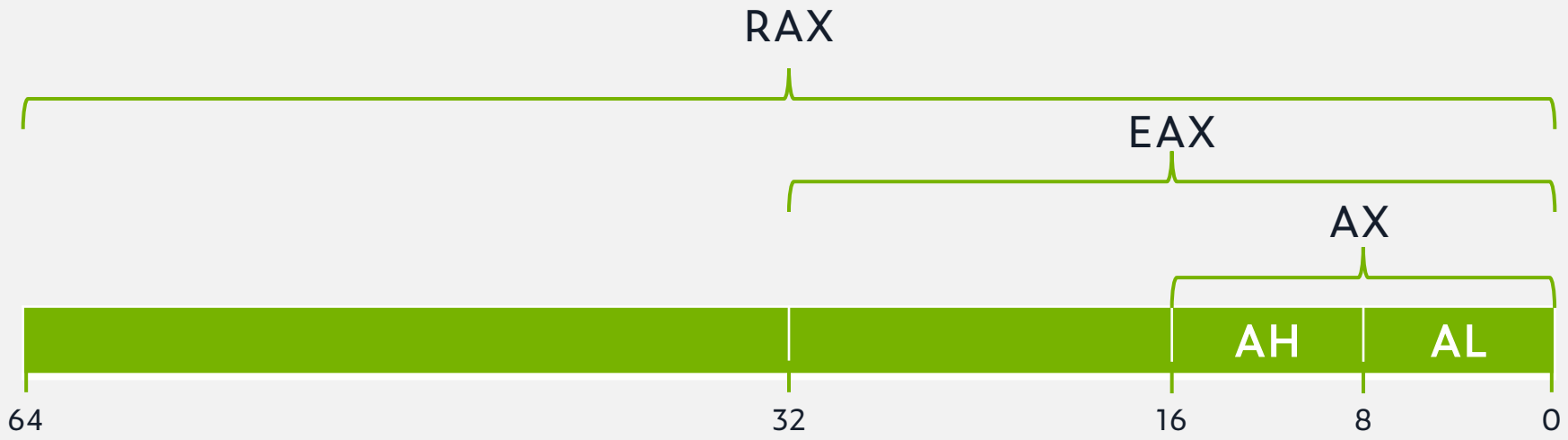
- Los registros generales son utilizados por la CPU durante la ejecución.
- Los registros de segmento se utilizan para identificar secciones de la memoria.
- Los indicadores de estado (flags) se utilizan para tomar decisiones.
- El puntero de instrucción apunta a la siguiente instrucción que se va a ejecutar.

Generales	De segmento	De estado	Puntero a instrucción
RAX	CS	RFLAGS	RIP
RBX	DS		
RCX	SS		
RDX	ES		
RSP	FS		
RBP	GS		
RSI			
RDI			
R8-R15			

> Registros

Registros generales

- Se suele almacenar en ellos datos o direcciones de memoria
- Cada registro de 64-bits se puede dividir en subregistros más pequeños (32, 16 y 8 bits)
- EAX suele guardar el resultado de las funciones
- Algunas instrucciones utilizan registros específicos (p. ej. EAX y EDX para MUL y DIV)



Flags

- El registro RFLAGS contiene un bit por flag (valores binarios)
- Los flags más importantes son:
  - ZF (Zero flag): indica si el resultado de la operación es 0.
  - CF (Carry flag): sirve para indicar que el resultado de una operación es demasiado grande o demasiado pequeño para el operador de destino.
  - SF (Sign flag): indica que el resultado de la operación es negativo (bit más significativo).
  - TF (Trap flag): se usa para debugging e indica que la CPU solo va a ejecutar una instrucción a la vez.

Instruction pointer (RIP)

- El registro RIP contiene la dirección de la próxima instrucción a ejecutar.
- Puede ser manipulado por ataques de buffer overflow (puede que lo veamos en otro meetup).

> Instrucciones simples

Mov

Copia datos entre dos operandos. Sintaxis: `mov destino, origen`

Instrucción	Descripción
mov eax, ebx	Copia el contenido de EBX en EAX.
mov eax, 0x42	Copia 0x42 en el registro EAX.
mov eax, [0x4037C4]	Copia los 4 de la ubicación de memoria 0x4037C4 en EAX.
mov eax, [ebx]	Copia los 4 bytes que hay en la dirección almacenada en EBX al registro EAX.
mov eax, [ebx+esi*4]	Copia los 4 bytes que hay en la dirección de memoria definida por ebx+esi*4 en EAX.

Lea (Load Effective Address)

Calcula una dirección de memoria y lo guarda en destino. Sintaxis: `lea destino, origen`

Ejemplo:

```
lea eax, [ebx+8]
```

Calcula y guarda en EAX el resultado de EBX + 8

El equivalente sería `mov eax, ebx+8` (pero es inválido)

`mov eax, [ebx+8]`: Copia el contenido que hay en la dirección EBX+8

> Instrucciones simples

Sumas y restas

- add, sub, inc y dec
- El resultado se guarda siempre en destino
- Sub deja valores en ZF (resultado 0) y en CF (Si el destino es menor que el valor a sustraer)

Instrucción	Descripción
sub eax, 0x10	Resta 0x10 de EAX.
add eax, ebx	Suma EBX a EAX y almacena el resultado en EAX.
inc edx	Incrementa EDX en 1.
dec ecx	Decrementa ECX en 1.

> Instrucciones simples

Multiplicación

- Sintaxis: `mul val`
- Multiplica el contenido de RAX por `val` (Operador implícito)
- Si hay overflow se guarda también en RDX

División

- Sintaxis: `div val`
- Divide el contenido de RDX:RAX entre `val` (Operador implícito)
- Resultado en RAX y resto en EDX

> Instrucciones simples

Operaciones lógicas

- `or, and y xor`
  - Sintaxis: `or|and|xor destino origen`
- `shr y shl`
  - Sintaxis: `shr|shl registro posiciones`
  - Desplaza el valor con el registro X posiciones a la izquierda (`shl`) o derecha (`shr`)
- `ror y rol`
  - Sintaxis: `ror|rol registro posiciones`
  - Rota el valor con el registro X posiciones a la izquierda (`rol`) o derecha (`ror`)

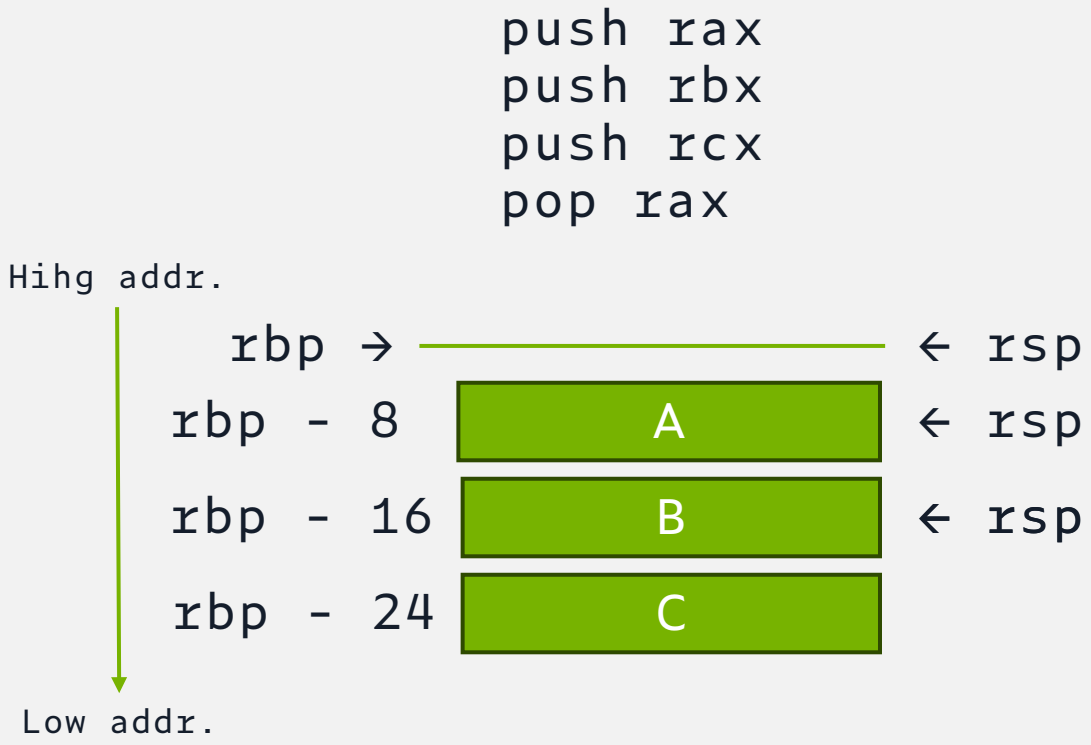
Not operation

- Sintaxis: nop
- Pasa a la siguiente instrucción

> La pila

La pila (stack) es una estructura de datos en memoria que se utiliza para almacenar información de manera temporal

- Sigue el principio LIFO (last in, first out)
- Almacena parámetros, variables, direcciones de retorno, etc.
- Instrucciones para operar la pila:
  - push: Copia el registro/dirección a la pila.
  - pop: Mueve el primer elemento de la pila al registro/dirección.
  - call: Llamada a función
  - ret: retorno de función



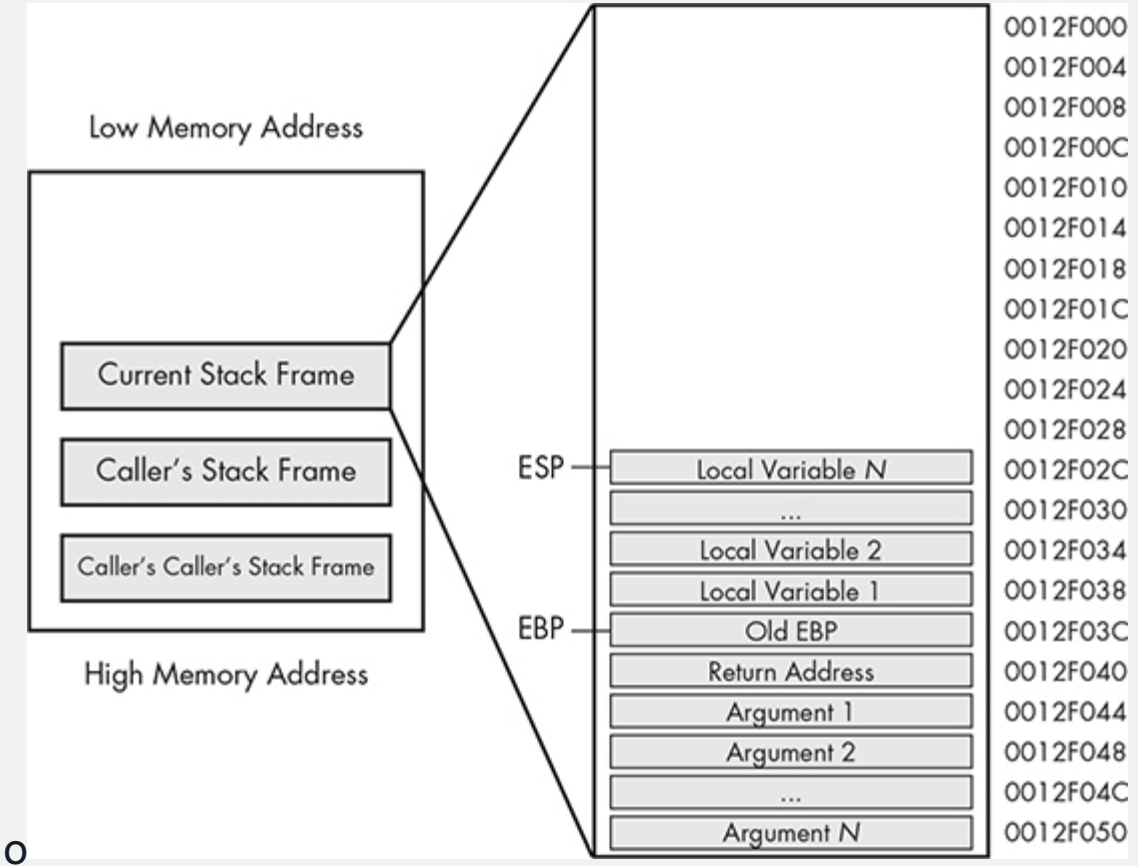
> Stack frame

La pila está compuesta por frames. Un stack frame es generado cada vez que se llama a una función y contiene:

- Los parámetros de la función que se está llamando
- La dirección de retorno
- El puntero al inicio del anterior frame
- Las variables locales de la función

Cada vez que se hace una llamada a una función:

1. Los argumentos se pasan a la pila (push)
2. Se hace push del RIP (Puntero a instrucción) en la pila para volver al acabar de ejecutar la función (return address)
3. Se ejecuta el function prologue
    1. Se guarda el *frame pointer* en la pila (`push ebp`)
    2. Se hace que el nuevo frame pointer apunte al inicio de la pila (`mov ebp esp`)
    3. Se guarda espacio para variables locales (`sub esp $n`)



> Saltos y condicionales

Condicionales

- `test`
  - Sintaxis: `test op1 op2`
  - Igual que una operación and sin afectar a los operandos.
  - Guarda los resultados en flags
  - `test eax, eax`: permite saber si es NULL
- `cmp`
  - Sintaxis: `cmp op1 op2`
  - Igual que una operación sub sin afectar a los operandos.
  - Otra instrucción para comparar (si son iguales ZERO FLAG = 1)

Saltos

Define un salto a una dirección concreta

jz loc	Salta si ZF = 1
jnz loc	Salta si ZF = 0
je loc	Salta en caso de valor igual
jne loc	Salta en caso de valor no igual
jg loc	Salto si el destino es mayor que el origen
jge loc	Salto si el destino es mayor o igual que el origen
ja loc	jg pero sin signo
jae loc	jge pero sin signo
jl loc	Salto si el destino es menor que el origen
jle loc	Salto si el destino es menor o igual que el origen
jb loc	jl pero sin signo
jbe loc	jle pero sin signo
jo loc	salto si OF = 1 (Overflow)
js loc	Salto si SF = 1 (signo)
jecxz loc	Salto si ECX = 0

Ghidra es un framework para la ingeniería inversa de software (SRE)

- Creado y mantenido por la National Security Agency (NSA)
- Open source
- Programado en Java

Ofrece un dissasembler y un decompiler

- Tiene una gran librería de procesadores/arquitecturas compatibles
- Soporte para plugins

Instalación

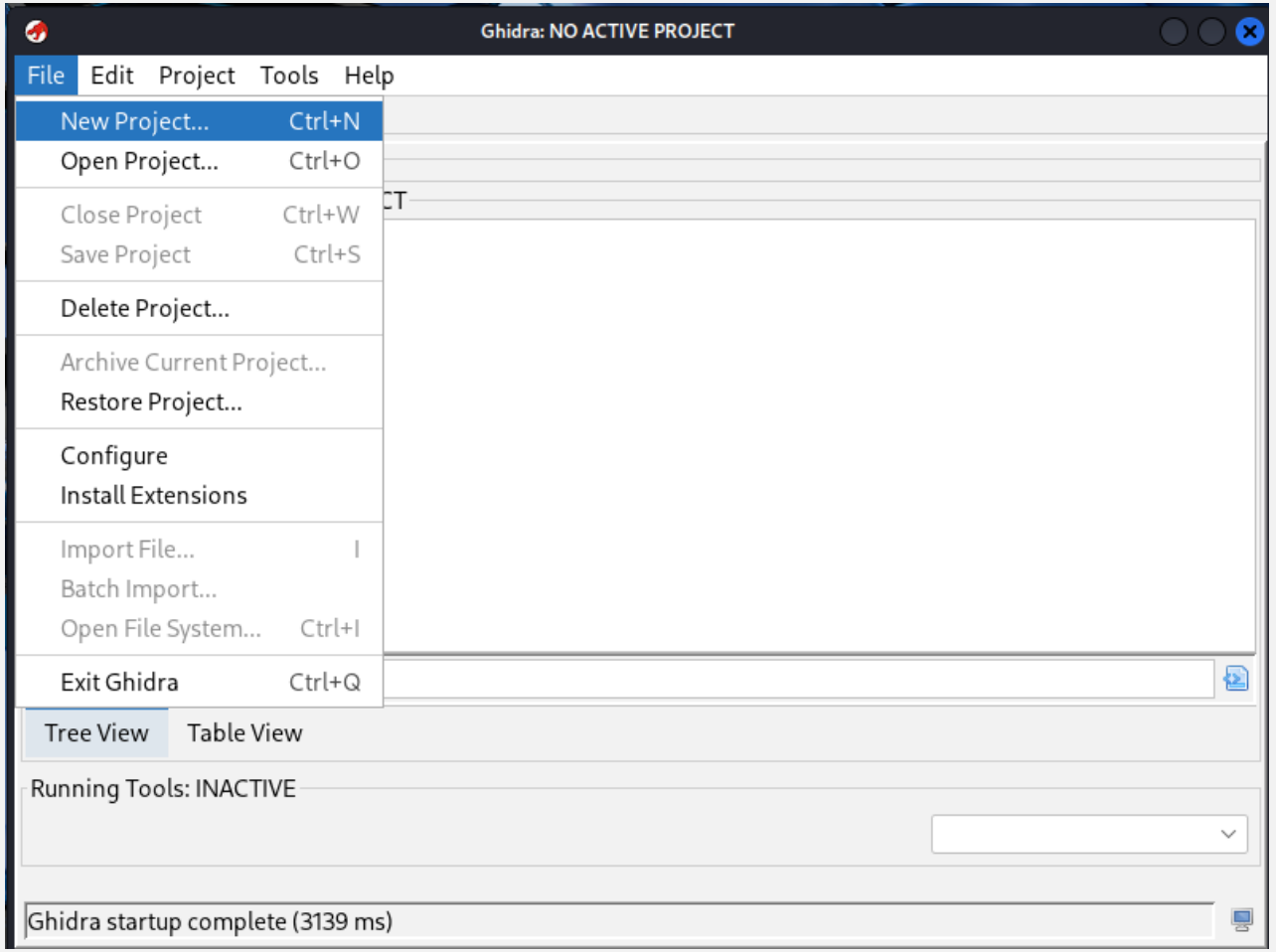
Opción 1: Desde el gestor de paquetes (Kali/Parrot)

```
apt get install openjdk-21 ghidra
```

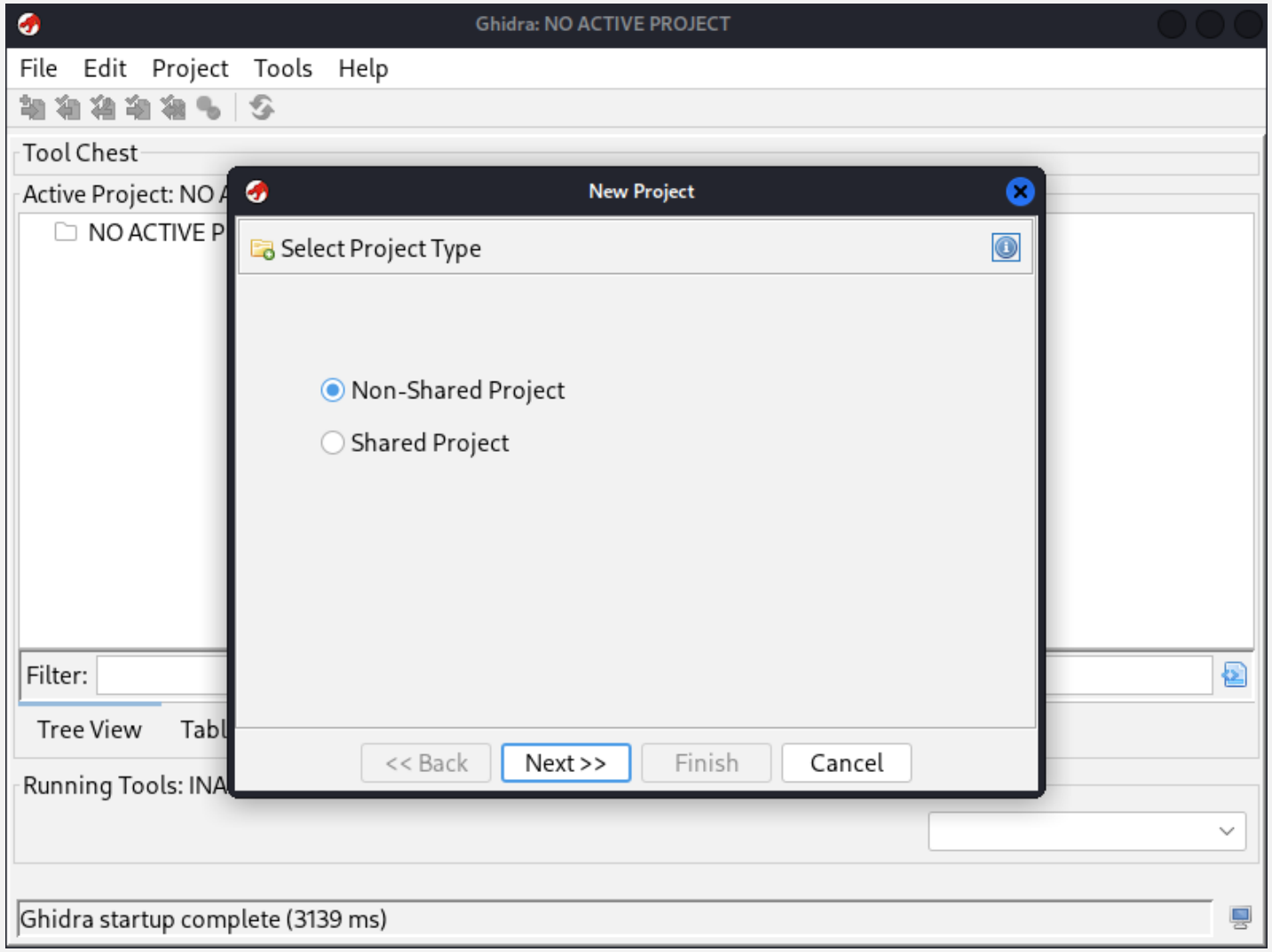
Opción 2: Manual (Linux y Windows)

1. Instalar JDK 21 64-bit (OpenJDK o tu favorito)
2. Descarga la última versión de Ghidra https://github.com/NationalSecurityAgency/ghidra/releases
3. Lanza Ghidra desde ./ghidraRun.sh (Linux) o ./ghidraRun.bat (Windows)

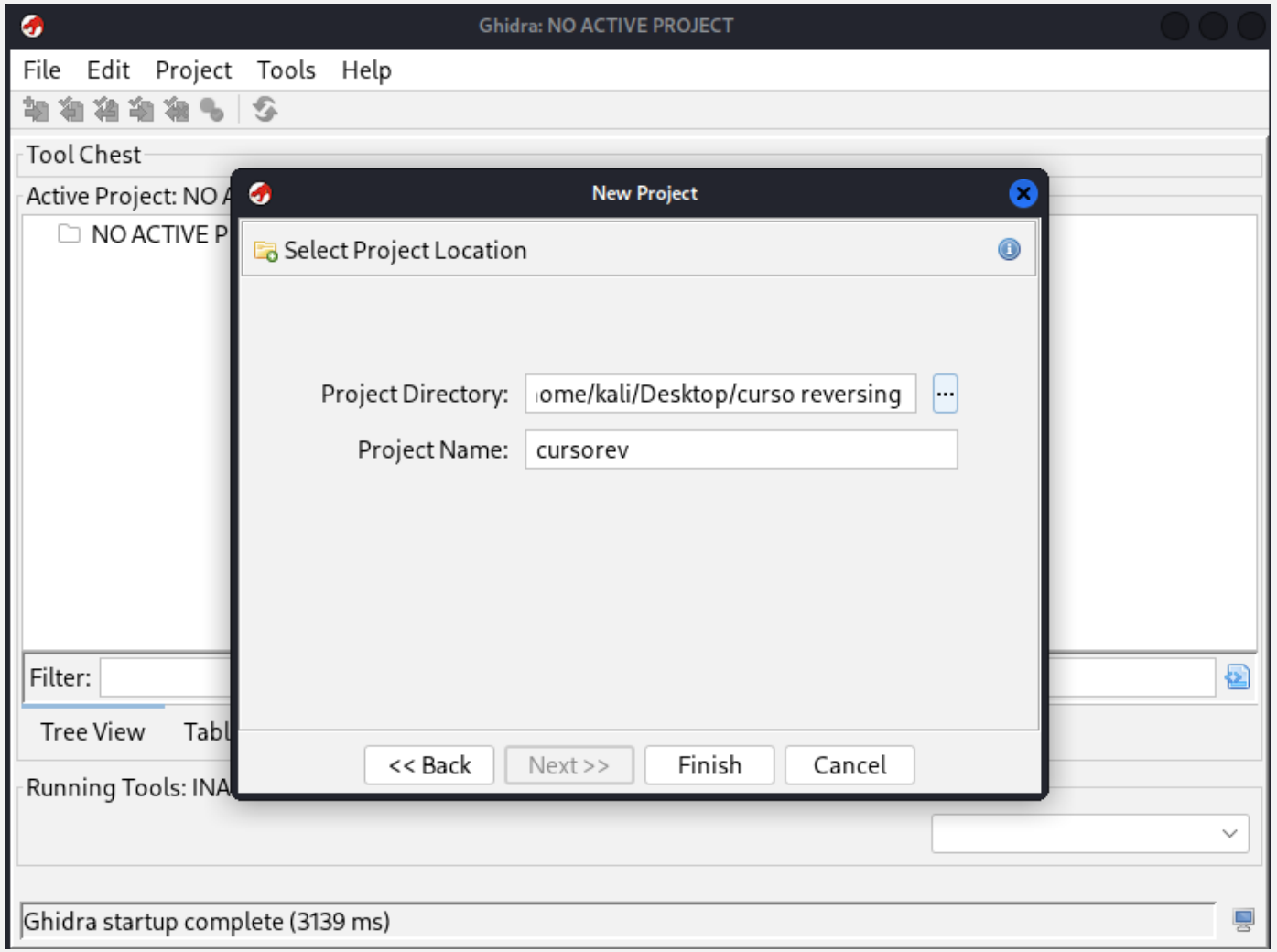
Nuevo proyecto



Nuevo proyecto



Nuevo proyecto

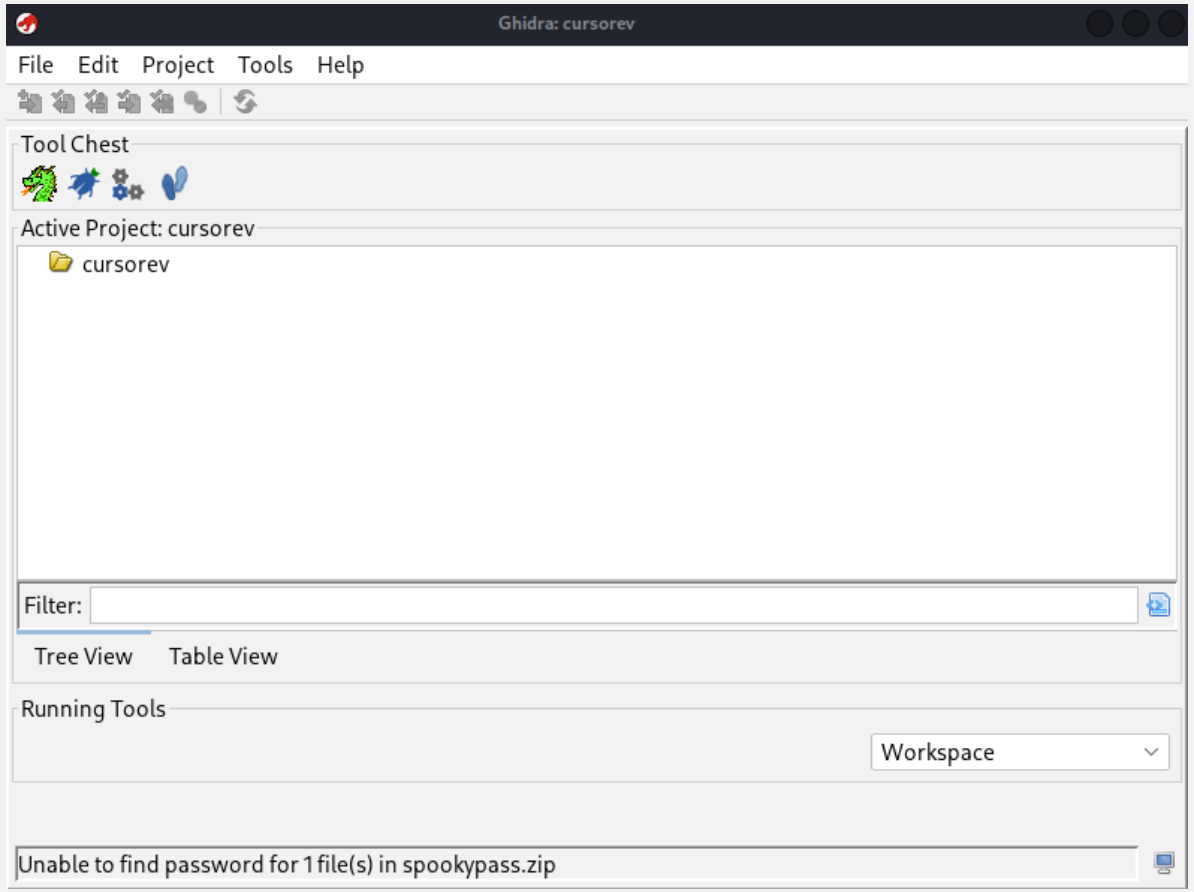


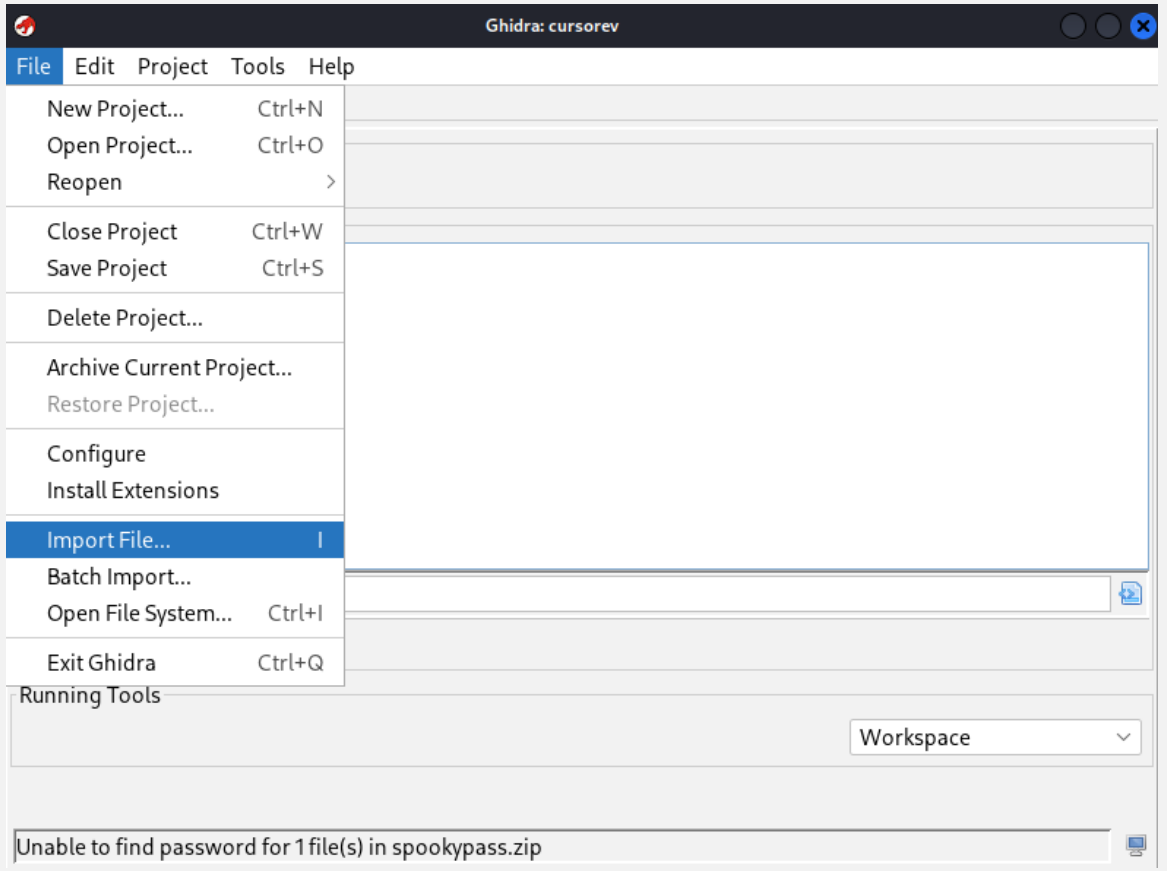
HACK THE BOX

SpookyPass

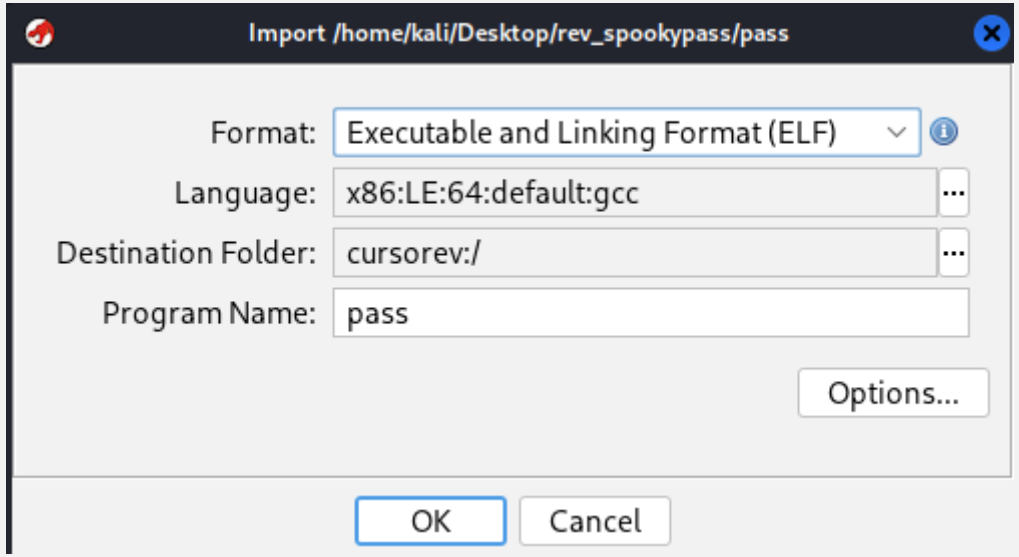
REVERSING VERY EASY

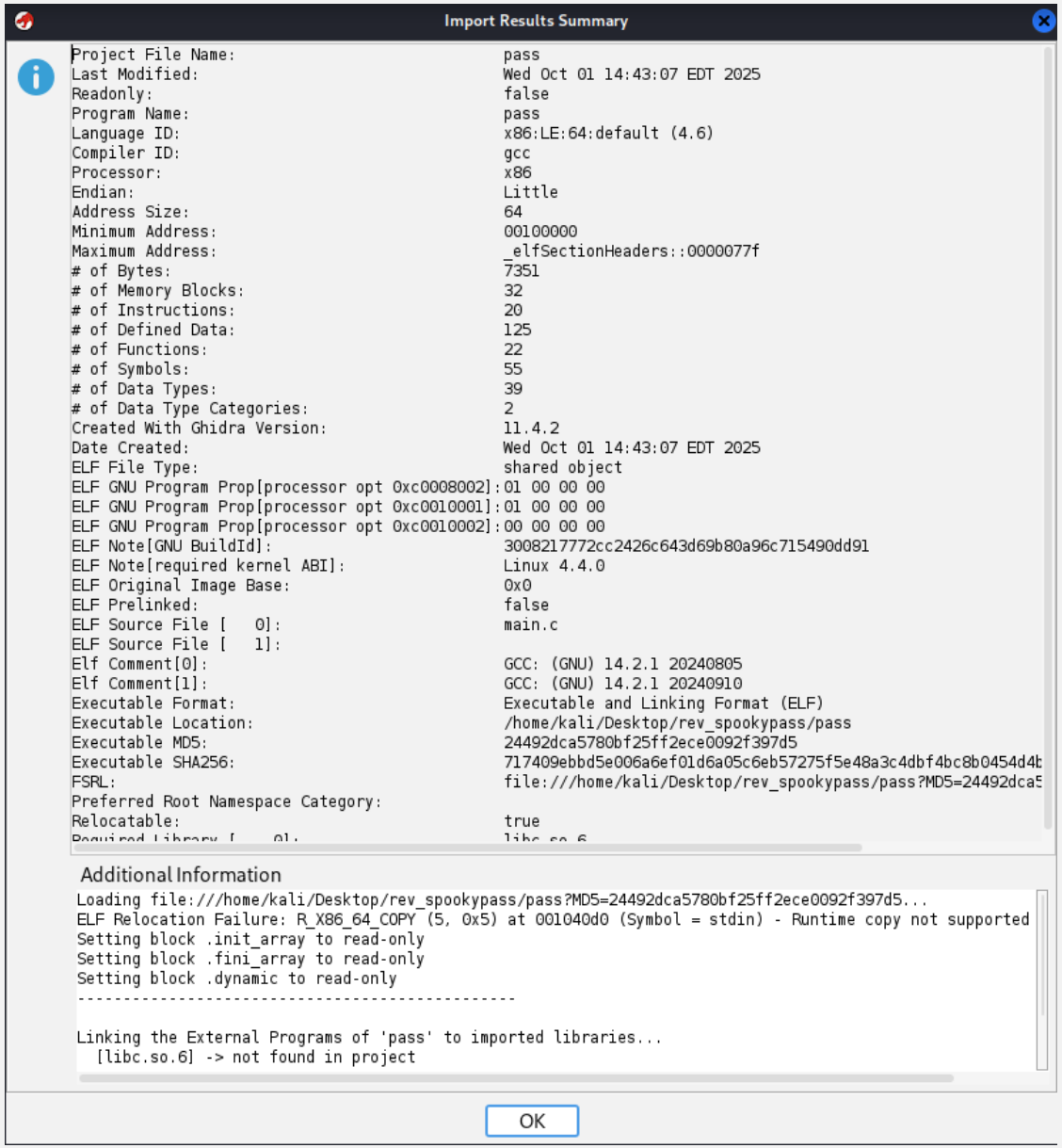
Importar binario



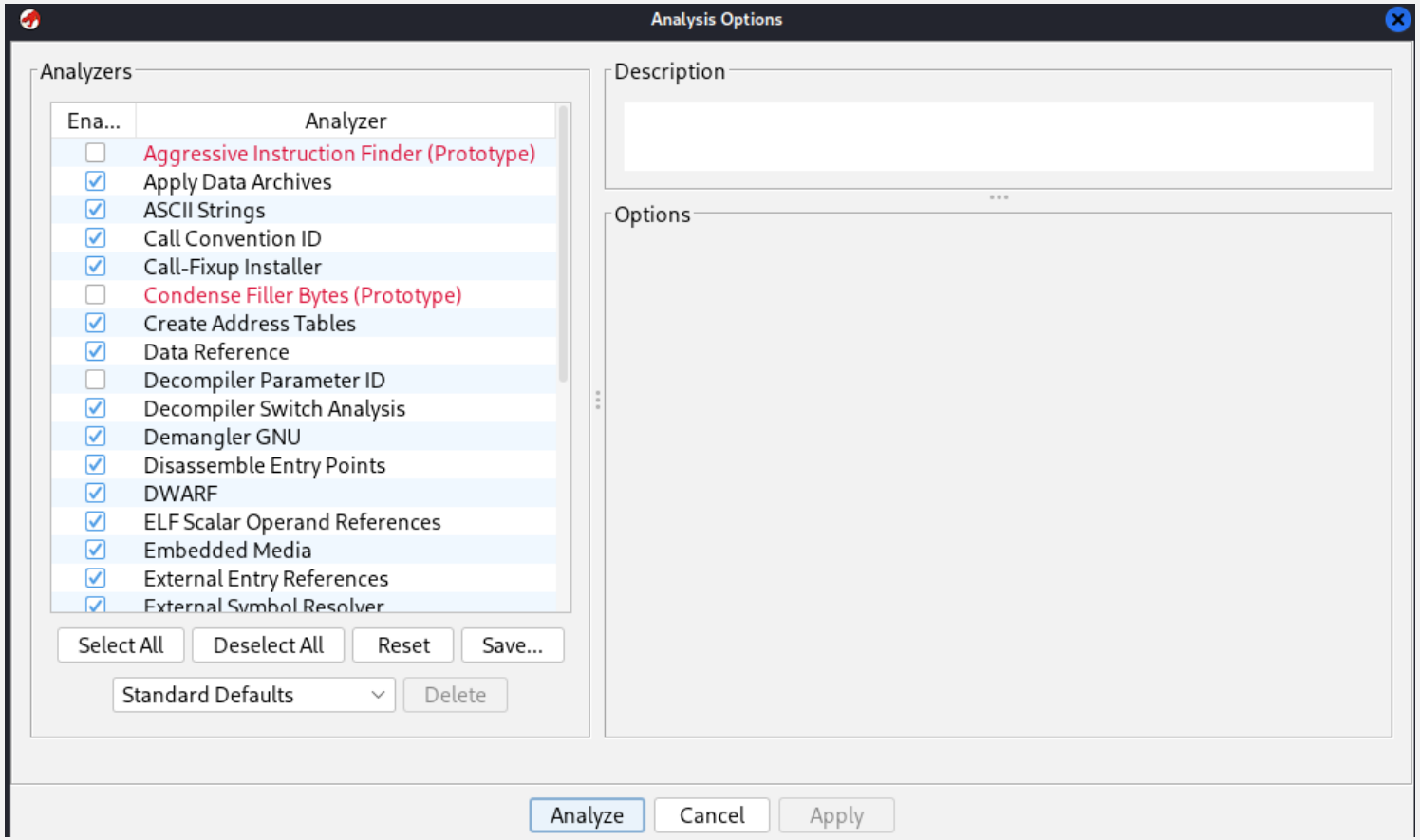


Importar binario

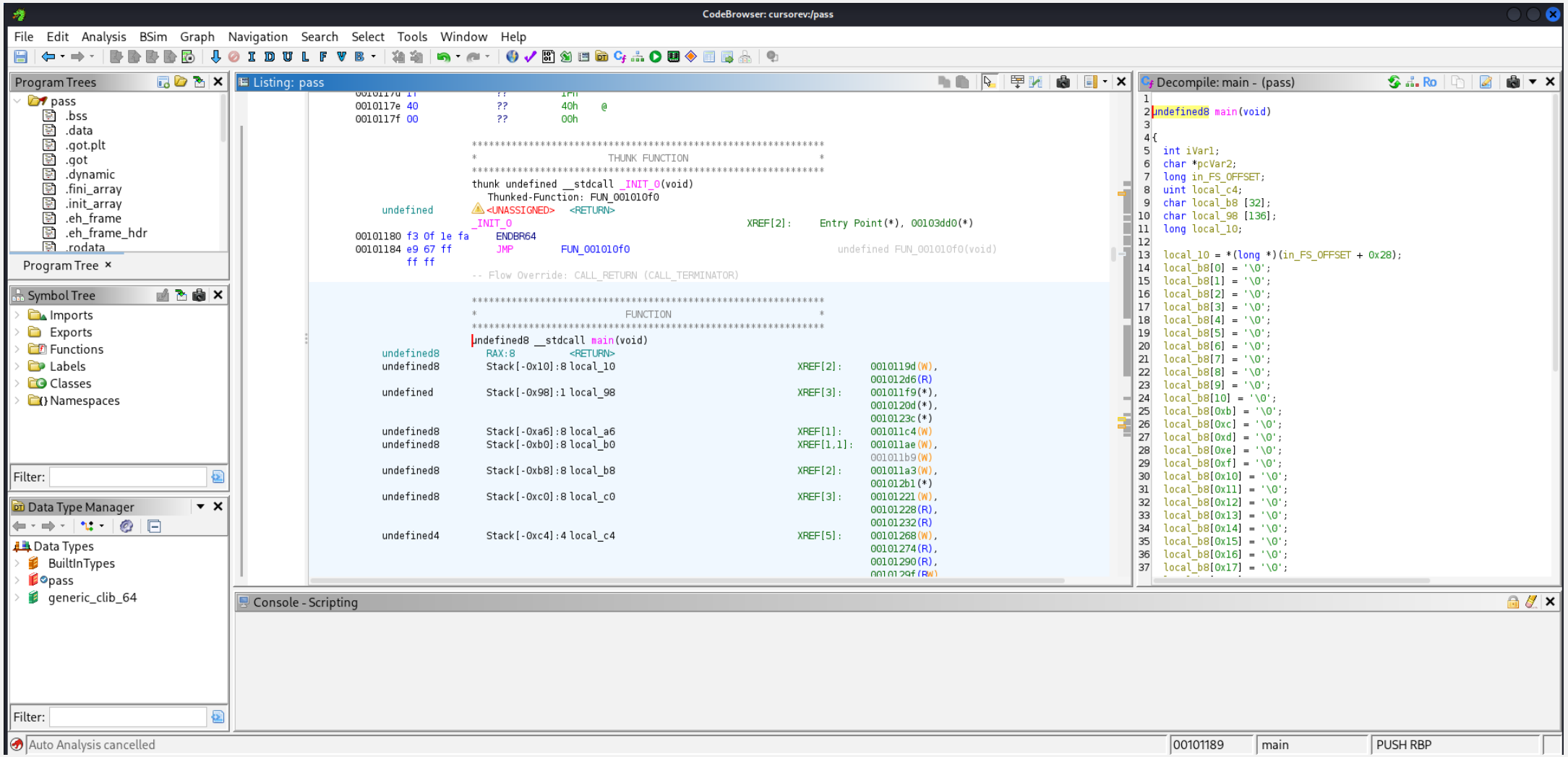




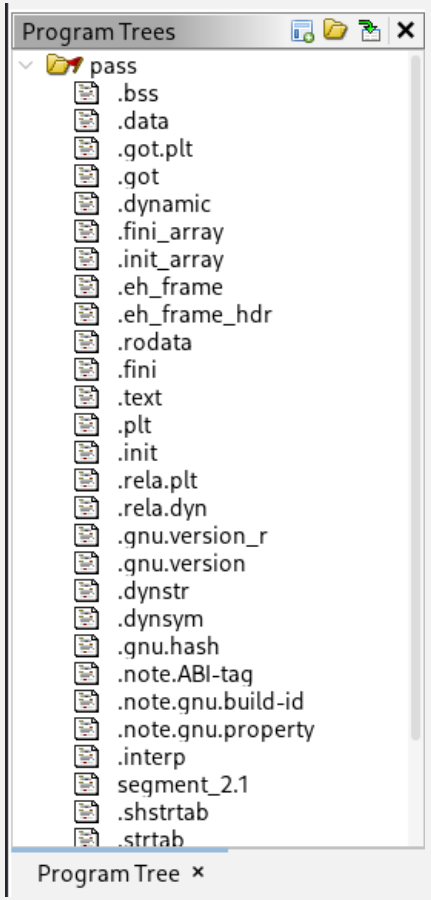
Analizar binario



Code Browser

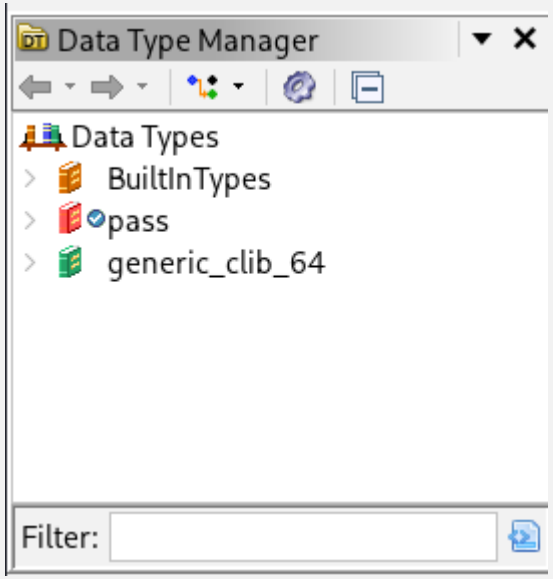


Program tree

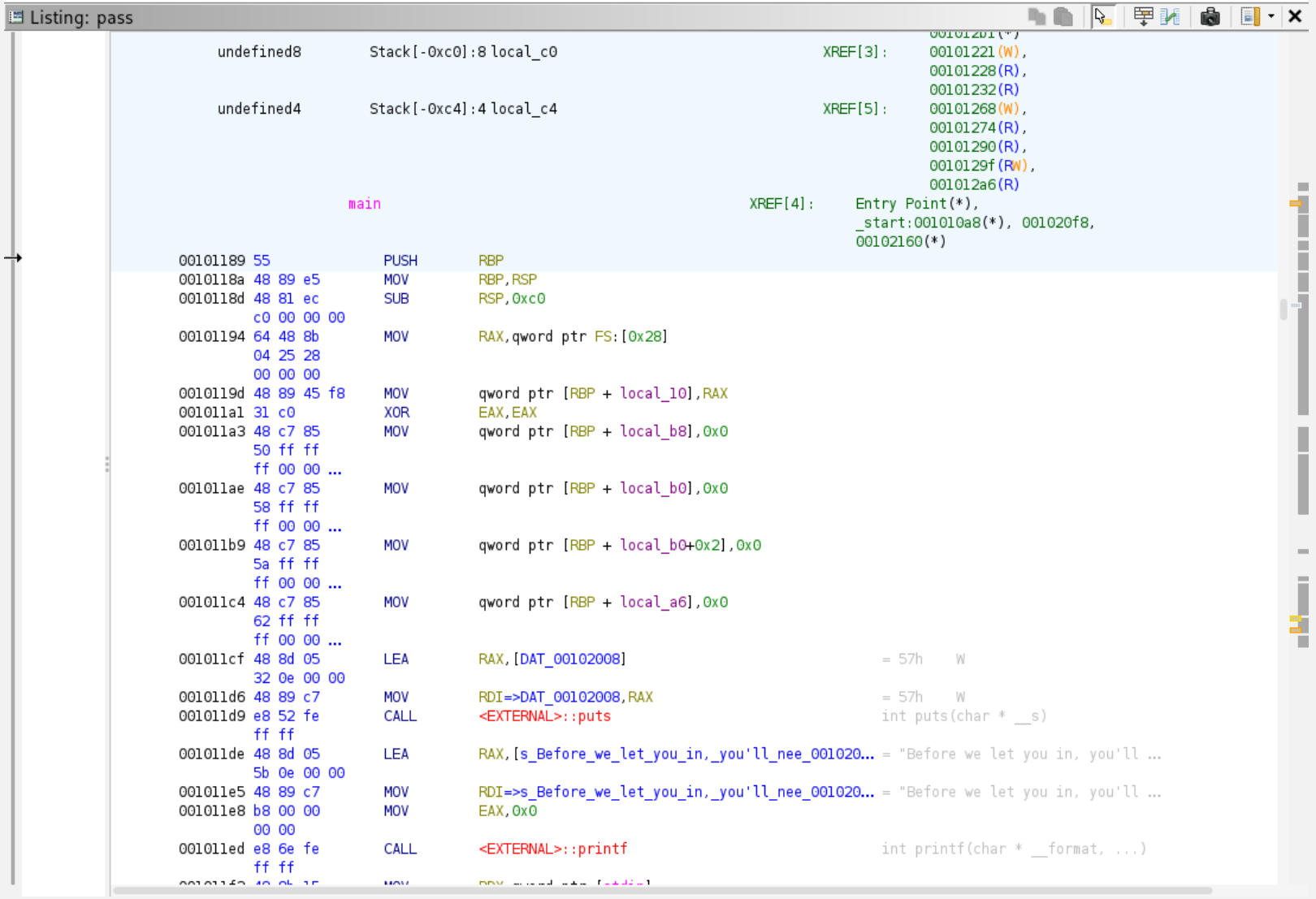


Symbol tree

Data type manager



Listing


Listing: pass
undefined8 Stack[-0xc0]:8 local_c0 XREF[3]: 00101221(W), 00101228(R), 00101232(R)
undefined4 Stack[-0xc4]:4 local_c4 XREF[5]: 00101268(W), 00101274(R), 00101290(R), 0010129f(RW), 001012a6(R)
main XREF[4]: Entry Point(*), _start:001010a8(*), 001020f8, 00102160(*)
00101189 55 PUSH RBP
0010118a 48 89 e5 MOV RBP,RSP
0010118d 48 81 ec c0 00 00 00 SUB RSP,0xc0
00101194 64 48 8b 04 25 28 00 00 00 MOV RAX,qword ptr FS:[0x28]
0010119d 48 89 45 f8 MOV qword ptr [RBP + local_10],RAX
001011a1 31 c0 XOR EAX,EAX
001011a3 48 c7 85 50 ff ff ff 00 00 00 ... MOV qword ptr [RBP + local_b8],0x0
001011ae 48 c7 85 58 ff ff ff 00 00 00 ... MOV qword ptr [RBP + local_b0],0x0
001011b9 48 c7 85 5a ff ff ff 00 00 00 ... MOV qword ptr [RBP + local_b0+0x2],0x0
001011c4 48 c7 85 62 ff ff ff 00 00 00 ... MOV qword ptr [RBP + local_a6],0x0
001011cf 48 8d 05 32 0e 00 00 LEA RAX,[DAT_00102008] = 57h W
001011d6 48 89 c7 MOV RDI=>DAT_00102008,RAX = 57h W
001011d9 e8 52 fe ff ff CALL <EXTERNAL>::puts int puts(char * __s)
001011de 48 8d 05 5b 0e 00 00 LEA RAX,[s_Before_we_let_you_in,_you'll_nee_001020... = "Before we let you in, you'll ...
001011e5 48 89 c7 MOV RDI=>s_Before_we_let_you_in,_you'll_nee_001020... = "Before we let you in, you'll ...
001011e8 b8 00 00 00 00 MOV EAX,0x0
001011ed e8 6e fe ff ff CALL <EXTERNAL>::printf int printf(char * __format, ...)

>Ghidra

Decompiler

Decompile: main - (pass)

```
1
2 undefined8 main(void)
3
4 {
5   int iVar1;
6   char *pcVar2;
7   long in_FS_OFFSET;
8   uint local_c4;
9   char local_b8 [32];
10  char local_98 [136];
11  long local_10;
12
13  local_10 = *(long *)(in_FS_OFFSET + 0x28);
14  local_b8[0] = '\0';
15  local_b8[1] = '\0';
16  local_b8[2] = '\0';
17  local_b8[3] = '\0';
18  local_b8[4] = '\0';
19  local_b8[5] = '\0';
20  local_b8[6] = '\0';
21  local_b8[7] = '\0';
22  local_b8[8] = '\0';
23  local_b8[9] = '\0';
24  local_b8[10] = '\0';
25  local_b8[0xb] = '\0';
26  local_b8[0xc] = '\0';
27  local_b8[0xd] = '\0';
28  local_b8[0xe] = '\0';
29  local_b8[0xf] = '\0';
30  local_b8[0x10] = '\0';
31  local_b8[0x11] = '\0';
32  local_b8[0x12] = '\0';
33  local_b8[0x13] = '\0';
34  local_b8[0x14] = '\0';
35  local_b8[0x15] = '\0';
36  local_b8[0x16] = '\0';
37  local_b8[0x17] = '\0';
38  local_b8[0x18] = '\0';
39  local_b8[0x19] = '\0';
40  puts("Welcome to the \x1b[1;3mSPOOKIEST\x1b[0m party of the year.");
41  printf("Before we let you in, you\'ll need to give us the password:
42  fgets(local_98,0x80,stdin);
43  pcVar2 = strchr(local_98,10);
44  if (pcVar2 != (char *)0x0) {
45    *pcVar2 = '\0';
```

Strings

Defined Strings - 85 items

Location	String Value	String Representa...	Data Type
.strtab::0000012b	__dso_handle	u8"__dso_handle"	utf8
.strtab::00000138	_IO_stdin_used	u8"_IO_stdin_used"	utf8
.strtab::00000147	_end	u8"_end"	utf8
.strtab::0000014c	__bss_start	u8"__bss_start"	utf8
.strtab::00000158	main	u8"main"	utf8
.strtab::0000015d	__TMC_END__	u8"__TMC_END__"	utf8
.strtab::00000169	_ITM_registerTMC...	u8"_ITM_registerT...	utf8
.strtab::00000183	__cxa_finalize@G...	u8"__cxa_finalize...	utf8
.strtab::0000019e	_init	u8"_init"	utf8
00100001	ELF	"ELF"	ds
00100318	/lib64/ld-linux-x86...	"/lib64/ld-linux-x8...	ds
00100344	GNU	"GNU"	ds
00100384	GNU	/lib64/ld-linux-x86-64.so.2	ds
001003a8	GNU	"GNU"	ds
00100521	fgets	u8"fgets"	utf8
00100527	stdin	u8"stdin"	utf8
0010052d	puts	u8"puts"	utf8
00100532	__stack_chk_fail	u8"__stack_chk_fail"	utf8
00100543	__libc_start_main	u8"__libc_start_m...	utf8
00100555	__cxa_finalize	u8"__cxa_finalize"	utf8
00100564	strchr	u8"strchr"	utf8
0010056b	printf	u8"printf"	utf8
00100572	strcmp	u8"strcmp"	utf8
00100579	libc.so.6	u8"libc.so.6"	utf8
00100583	GLIBC_2.4	u8"GLIBC_2.4"	utf8
0010058d	GLIBC_2.2.5	u8"GLIBC_2.2.5"	utf8
00100599	GLIBC_2.34	u8"GLIBC_2.34"	utf8
001005a4	_ITM_deregisterT...	u8"_ITM_deregiste...	utf8
001005c0	__gmon_start__	u8"__gmon_start__"	utf8
001005cf	_ITM_registerTMC...	u8"_ITM_registerT...	utf8
00102027	[0m party of the ye...	"[0m party of the y...	ds
00102040	Before we let you i...	"Before we let you i...	ds

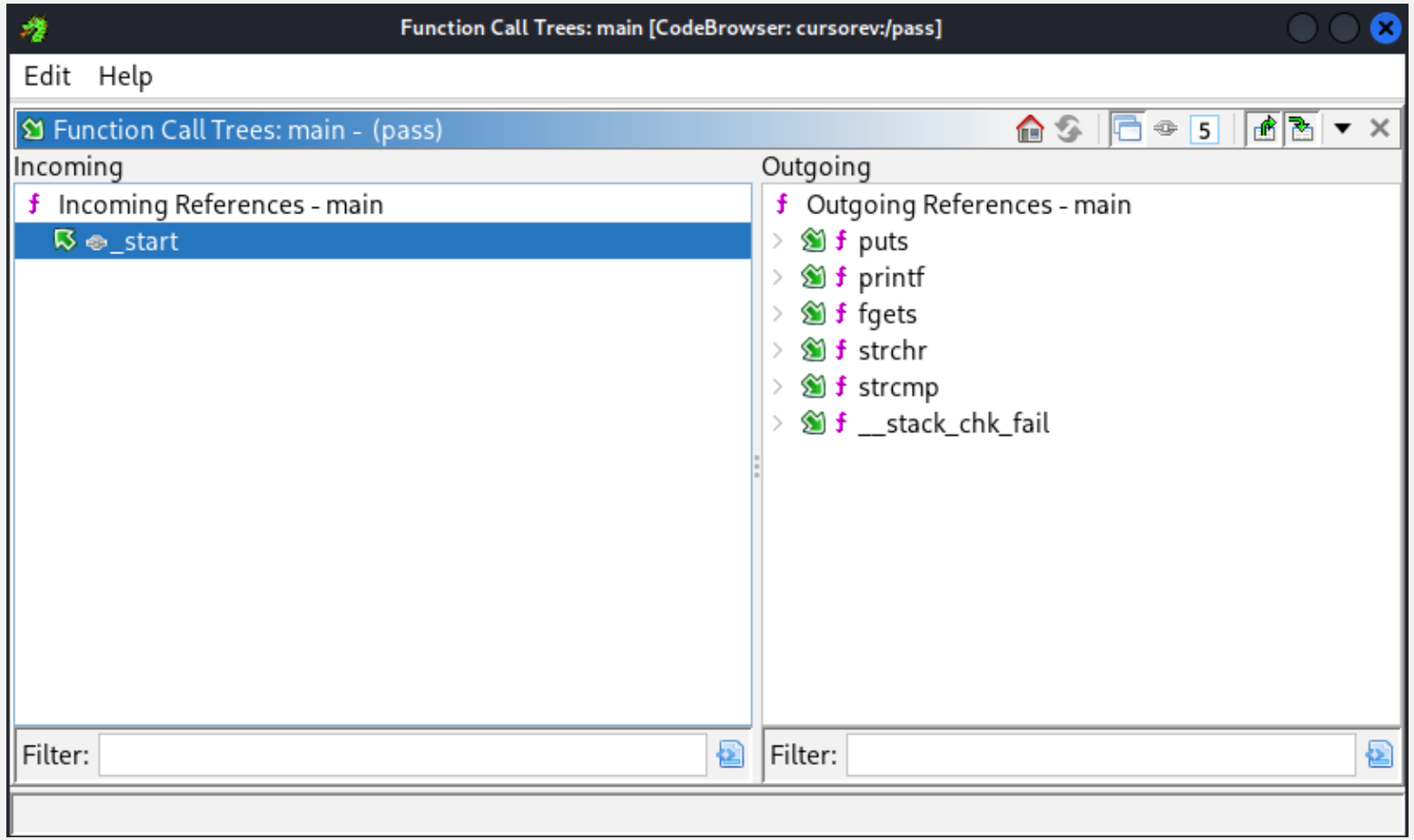
Byte view

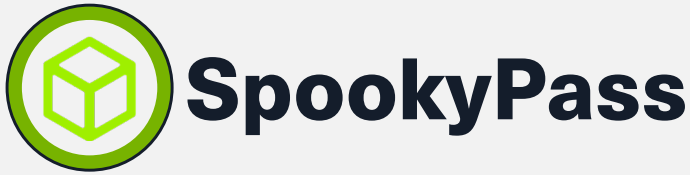
Bytes: pass

Addresses	Hex
00101080	ff 25 a2 2f 00 00 68 05 00 00 00 e9 90 ff ff ff
.....	.. ..
00101090	f3 0f 1e fa 31 ed 49 89 d1 5e 48 89 e2 48 83 e4
001010a0	f0 50 54 45 31 c0 31 c9 48 8d 3d da 00 00 00 ff
001010b0	15 0b 2f 00 00 f4 66 2e 0f 1f 84 00 00 00 00 00
001010c0	48 8d 3d 01 30 00 00 48 8d 05 fa 2f 00 00 48 39
001010d0	f8 74 15 48 8b 05 ee 2e 00 00 48 85 c0 74 09 ff
001010e0	e0 0f 1f 80 00 00 00 00 c3 0f 1f 80 00 00 00 00
001010f0	48 8d 3d d1 2f 00 00 48 8d 35 ca 2f 00 00 48 29
00101100	fe 48 89 f0 48 c1 ee 3f 48 c1 f8 03 48 01 c6 48
00101110	d1 fe 74 14 48 8b 05 bd 2e 00 00 48 85 c0 74 08
00101120	ff e0 66 0f 1f 44 00 00 c3 0f 1f 80 00 00 00 00
00101130	f3 0f 1e fa 80 3d 9d 2f 00 00 00 75 33 55 48 83
00101140	3d 9a 2e 00 00 00 48 89 e5 74 0d 48 8b 3d f6 2e
00101150	00 00 ff 15 88 2e 00 00 e8 63 ff ff ff c6 05 74
00101160	2f 00 00 01 5d c3 66 2e 0f 1f 84 00 00 00 00 00
00101170	c3 66 66 2e 0f 1f 84 00 00 00 00 00 0f 1f 40 00
00101180	f3 0f 1e fa e9 67 ff ff ff 55 48 89 e5 48 81 ec
00101190	c0 00 00 00 64 48 8b 04 25 28 00 00 00 48 89 45
001011a0	f8 31 c0 48 c7 85 50 ff ff ff 00 00 00 00 48 c7
001011b0	85 58 ff ff ff 00 00 00 00 48 c7 85 5a ff ff ff
001011c0	00 00 00 00 48 c7 85 62 ff ff ff 00 00 00 00 48
001011d0	8d 05 32 0e 00 00 48 89 c7 e8 52 fe ff ff 48 8d
001011e0	05 5b 0e 00 00 48 89 c7 b8 00 00 00 00 e8 6e fe
001011f0	ff ff 48 8b 15 d7 2e 00 00 48 8d 85 70 ff ff ff
00101200	be 80 00 00 00 48 89 c7 e8 63 fe ff ff 48 8d 85
00101210	70 ff ff ff be 0a 00 00 00 48 89 c7 e8 2f fe ff
00101220	ff 48 89 85 48 ff ff ff 48 83 bd 48 ff ff ff 00
00101230	74 0a 48 8b 85 48 ff ff ff c6 00 00 48 8d 85 70
00101240	ff ff ff 48 8d 15 36 0e 00 00 48 89 d6 48 89 c7
00101250	e8 2b fe ff ff 85 c0 75 69 48 8d 05 42 0e 00 00
00101260	48 89 c7 e8 c8 fd ff ff c7 85 44 ff ff ff 00 00
00101270	00 00 eb 32 8b 85 44 ff ff ff 48 98 48 8d 14 85
00101280	00 00 00 00 48 8d 05 d5 2d 00 00 8b 04 02 89 c2
00101290	8b 85 44 ff ff ff 48 98 88 94 05 50 ff ff ff 83
001012a0	85 44 ff ff ff 01 8b 85 44 ff ff ff 83 f8 19 76
001012b0	c3 48 8d 85 50 ff ff ff 48 89 c7 e8 70 fd ff ff
001012c0	eb 0f 48 8d 05 ef 0d 00 00 48 89 c7 e8 5f fd ff
001012d0	ff b8 00 00 00 00 48 8b 55 f8 64 48 2b 14 25 28
001012e0	00 00 00 74 05 e8 56 fd ff ff c9 c3
.....	.. ..
001012ec	f3 0f 1e fa
001012f0	48 83 ec 08 48 83 c4 08 c3

Start: 00100000 End: _elfSectionHeaders::0000077f Offset: 00000000 Insertion: 00101189

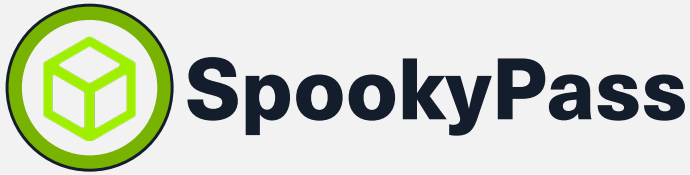
Function call tree





TAREA 1

¿Cuál es la contraseña? ¿Qué hace el programa?



TAREA 1

¿Cuál es la contraseña? ¿Qué hace el programa?

```
iVar1 = strcmp(local_98,"s3cr3t_p455_f0r_gh05t5_4nd_gh0ul5");
if (iVar1 == 0) {
  puts("Welcome inside!");
```

HACK THE BOX

Simple Encryptor

REVERSING VERY EASY

> ¡Sigue practicando!

HackTheBox Labs/Academy

https://referral.hackthebox.com/mzw43pf

Comunidad HTB Cáceres

meetup

GRACIAS

htb.caceres

HTBCaceres

HACKTHEBOX

UEX

CÁTEDRA INCIBE-UEX-EPCC