

SEGURIDAD EN SISTEMAS OPERATIVOS

4º Grado en Informática - Complementos de Ing. del Software

Curso 2018-19

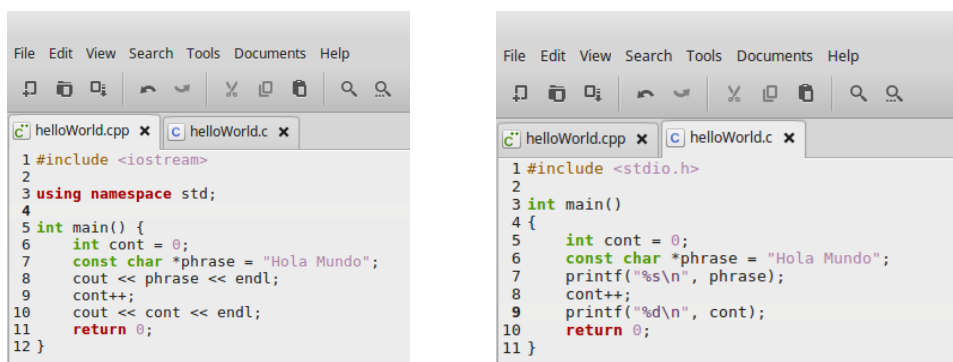
Práctica 2. Ingeniería inversa y vulnerabilidades

Sesión 1. El formato ELF (*Executable and Linkable Format*) en Linux

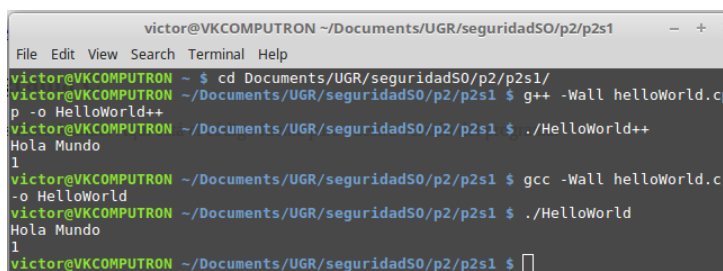
Autor¹: Víctor García Carrera

Ejercicio 1.

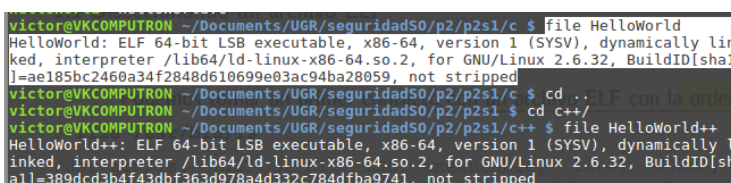
Para conocer la estructura de los archivos ejecutables ELF y las herramientas básicas para ver su contenido, comenzamos creando un programa que imprime por pantalla un mensaje almacenado en una variable de tipo `char*` *phrase* e imprime por pantalla el valor de una variable de tipo entero *cont* inicializada a 0 la cual aumentamos en 1. Desarrollamos dos versiones del mismo en dos lenguajes de programación diferentes, C++ y C, de nombres *helloWorld.cpp* y *helloWorld.c* respectivamente.



Compilamos como se muestra a continuación cada una de las versiones con los compiladores `g++` y `gcc`, y vemos que la ejecución es la misma.



A la hora de observar la estructura y secciones de ambos ejecutables ELF, visualizamos con el comando `file` el tipo de archivos que son, ambos archivos ELF para una arquitectura de 64 bits y little endian (LSB) sin información de depuración (*not stripped*).



¹ Como autor declaro que los contenidos del presente documento son originales y elaborados por mi. De no cumplir con este compromiso, soy consciente de que, de acuerdo con la “[Normativa de evaluación y de calificaciones de los estudiantes de la Universidad de Granada](#)” esto “conllevará la calificación numérica de cero ... independientemente del resto de calificaciones que el estudiante hubiera obtenido ...”

Acto seguido, visualizamos con el comando `readelf -h` la cabecera de los archivos ELF, de nuevo sin cambios remarcables entre ambas versiones a excepción de la dirección de inicio de las cabeceras de sección, superior en la versión de C++:

```
victor@VKCOMPUTRON ~/Documents/UGR/seguridad50/p2/p2s1/c $ readelf -h HelloWorld
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:    EXEC (Executable file)
  Machine: Advanced Micro Devices X86-64
  Version: 0x1
  Entry point address: 0x400470
  Start of program headers: 64 (bytes into file)
  Start of section headers: 6680 (bytes into file)
  Flags:    0x0
  Size of this header: 64 (bytes)
  Size of program headers: 56 (bytes)
  Number of program headers: 9
  Size of section headers: 64 (bytes)
  Number of section headers: 31
  Section header string table index: 28

victor@VKCOMPUTRON ~/Documents/UGR/seguridad50/p2/p2s1/c $ cd ..
victor@VKCOMPUTRON ~/Documents/UGR/seguridad50/p2/p2s1 $ cd c++/
victor@VKCOMPUTRON ~/Documents/UGR/seguridad50/p2/p2s1/c++ $ readelf -h HelloWorld++
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:    EXEC (Executable file)
  Machine: Advanced Micro Devices X86-64
  Version: 0x1
  Entry point address: 0x4007a0
  Start of program headers: 64 (bytes into file)
  Start of section headers: 7288 (bytes into file)
  Flags:    0x0
  Size of this header: 64 (bytes)
  Size of program headers: 56 (bytes)
  Number of program headers: 9
  Size of section headers: 64 (bytes)
  Number of section headers: 31
  Section header string table index: 28

victor@VKCOMPUTRON ~/Documents/UGR/seguridad50/p2/p2s1/c++ $
```

A continuación, vamos a observar las secciones presentes en los archivos ELF, comentando algunas de ellas y comparando las dos versiones.

Una sección de interés es la de `.interp` pues almacena la ruta al intérprete utilizado (basicamente su nombre), en este caso `/lib64/ld-linux-x86-64-.so.2`.

Otra sección interesante es `.got`, la cual contiene la dirección de la *Global Offset Table* o GOT. Esta tabla de desplazamientos globales se utiliza para garantizar el funcionamiento correcto de aquel código que sea independiente de su direccionamiento (*position-independent code* o PIC). El código PIC no puede por lo general almacenar direcciones virtuales absolutas. Esta tabla GOT almacena direcciones absolutas en datos privados para permitir que cuando se utilizan referencias independientes de la posición, éstas sean traducidas a direcciones absolutas. La sección `.got.plt` contiene aquella porción de la tabla GOT que es de sólo lectura o read-only.

La sección `.ctors` contiene una lista de punteros a función de los constructores globales, mientras que la sección `.dtors` contiene una lista de punteros a función de los destructores globales.

Al listar las secciones de ambas versiones, podemos comprobar que **no existe ninguna diferencia relevante, excepto las direcciones** de muchas de las secciones excepto `.interp`, `.note.ABI-tag`, `.note.gnu.build-id`, `.gnu.hash`, `.got` y `.got.plt`, además de un valor distinto del campo Info o Align en las secciones `.gnu.version_r` o `.bss`.

```
victor@VKCOMPUTRON ~/Documents/UGR/seguridadS0/p2/p2s1/c $ readelf -S HelloWorld
There are 31 section headers, starting at offset 0x1a18:

Section Headers:
 [Nr] Name              Type             Address           Offset
     Size              EntSize          Flags Link Info  Align
 [ 0] 0000000000000000 NULL             0000000000000000 0 0 0 0
 [ 1] .interp              PROGBITS         0000000000400238 00000238
     000000000000001c A 0 0 1
 [ 2] .note.ABI-tag        NOTE             0000000000400254 00000254
     0000000000000020 A 0 0 4
 [ 3] .note.gnu.build-id   NOTE             0000000000400274 00000274
     0000000000000024 A 0 0 4
 [ 4] .gnu.hash            GNU_HASH         0000000000400298 00000298
     000000000000001c A 5 0 8
 [ 5] .dynsym              DYNSYM           00000000004002b8 000002b8
     0000000000000018 A 6 1 8
 [ 6] .dynstr              STRTAB           0000000000400330 00000330
     0000000000000000 A 0 0 1
 [ 7] .gnu.version          VERSYM           0000000000400374 00000374
     0000000000000002 A 5 0 2
 [ 8] .gnu.version_r        VERNEED          0000000000400380 00000380
     0000000000000020 A 6 1 8
 [ 9] .rela.dyn             RELA             00000000004003a0 000003a0
     0000000000000018 A 5 0 8
[10] .rela.plt             RELA             00000000004003b8 000003b8
     0000000000000048 AI 5 24 8
[11] .init                 PROGBITS         0000000000400400 00000400
     000000000000001a AX 0 0 4
[12] .plt                  PROGBITS         0000000000400420 00000420
     0000000000000040 AX 0 0 16
[13] .plt.got              PROGBITS         0000000000400460 00000460
     0000000000000008 AX 0 0 8
[14] .text                 PROGBITS         0000000000400470 00000470
     00000000000001b2 AX 0 0 16
[15] .fini                 PROGBITS         0000000000400624 00000624
     0000000000000009 AX 0 0 4
[16] .rodata               PROGBITS         0000000000400630 00000630
     0000000000000013 A 0 0 4
[17] .eh_frame_hdr         PROGBITS         0000000000400644 00000644
     0000000000000034 A 0 0 4
[18] .eh_frame             PROGBITS         0000000000400678 00000678
     00000000000000f4 A 0 0 8
[19] .init_array           INIT_ARRAY        0000000000600e10 00000e10
```

```
[19] .init_array          INIT_ARRAY        0000000000600e10 00000e10
     0000000000000008 WA 0 0 8
[20] .fini_array          FINI_ARRAY        0000000000600e18 00000e18
     0000000000000008 WA 0 0 8
[21] .jcr                  PROGBITS         0000000000600e20 00000e20
     0000000000000008 WA 0 0 8
[22] .dynamic              DYNAMIC          0000000000600e28 00000e28
     00000000000001d0 WA 6 0 8
[23] .got                  PROGBITS         0000000000600ff8 00000ff8
     0000000000000008 WA 0 0 8
[24] .got.plt              PROGBITS         0000000000601000 00001000
     0000000000000030 WA 0 0 8
[25] .data                 PROGBITS         0000000000601030 00001030
     0000000000000010 WA 0 0 8
[26] .bss                  NOBITS           0000000000601040 00001040
     0000000000000008 WA 0 0 1
[27] .comment              PROGBITS         0000000000000000 00001040
     0000000000000035 MS 0 0 1
[28] .shstrtab             STRTAB           0000000000000000 00001905
     000000000000010c 0 0 1
[29] .symtab               SYMTAB           0000000000000000 00001078
     0000000000000660 30 47 8
[30] .strtab               STRTAB           0000000000000000 000016d8
     000000000000022d 0 0 1

Key to Flags:
W (write), A (alloc), X (execute), M (merge), S (strings), l (large)
I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)
0 (extra OS processing required) o (OS specific), p (processor specific)
victor@VKCOMPUTRON ~/Documents/UGR/seguridadS0/p2/p2s1/c $
```

```
victor@VKCOMPUTRON ~/Documents/UGR/seguridadS0/p2/p2s1/c++ $ readelf -S HelloWorld++
There are 31 section headers, starting at offset 0x1c78:

Section Headers:
 [Nr] Name              Type             Address           Offset
     Size              EntSize          Flags Link Info  Align
 [ 0] 0000000000000000 NULL             0000000000000000 0 0 0 0
 [ 1] .interp              PROGBITS         0000000000400238 00000238
     000000000000001c A 0 0 1
 [ 2] .note.ABI-tag        NOTE             0000000000400254 00000254
     0000000000000020 A 0 0 4
 [ 3] .note.gnu.build-id   NOTE             0000000000400274 00000274
     0000000000000024 A 0 0 4
 [ 4] .gnu.hash            GNU_HASH         0000000000400298 00000298
     0000000000000030 A 5 0 8
 [ 5] .dynsym              DYNSYM           00000000004002c8 000002c8
     0000000000000018 A 6 1 8
 [ 6] .dynstr              STRTAB           0000000000400418 00000418
     0000000000000000 A 0 0 1
 [ 7] .gnu.version          VERSYM           00000000004005a8 000005a8
     000000000000001c A 5 0 2
 [ 8] .gnu.version_r        VERNEED          00000000004005e8 000005e8
     0000000000000040 A 6 2 8
 [ 9] .rela.dyn             RELA             00000000004005f8 000005f8
     0000000000000018 A 5 0 8
[10] .rela.plt             RELA             0000000000400618 00000618
     00000000000000c0 AI 5 24 8
[11] .init                 PROGBITS         00000000004006d8 000006d8
     000000000000001a AX 0 0 4
[12] .plt                  PROGBITS         0000000000400700 00000700
     0000000000000090 AX 0 0 16
[13] .plt.got              PROGBITS         0000000000400790 00000790
     0000000000000008 AX 0 0 8
[14] .text                 PROGBITS         00000000004007a0 000007a0
     0000000000000222 AX 0 0 16
[15] .fini                 PROGBITS         00000000004009c4 000009c4
     0000000000000009 AX 0 0 4
[16] .rodata               PROGBITS         00000000004009d0 000009d0
     000000000000000f A 0 0 4
[17] .eh_frame_hdr         PROGBITS         00000000004009e0 000009e0
     0000000000000044 A 0 0 4
[18] .eh_frame             PROGBITS         0000000000400a28 00000a28
     0000000000000134 A 0 0 8
[19] .init_array           INIT_ARRAY        0000000000600df8 00000df8
```

```
[19] .init_array          INIT_ARRAY        0000000000600df8 00000df8
     0000000000000010 WA 0 0 8
[20] .fini_array          FINI_ARRAY        0000000000600e08 00000e08
     0000000000000008 WA 0 0 8
[21] .jcr                  PROGBITS         0000000000600e10 00000e10
     0000000000000008 WA 0 0 8
[22] .dynamic              DYNAMIC          0000000000600e18 00000e18
     00000000000001e0 WA 6 0 8
[23] .got                  PROGBITS         0000000000600ff8 00000ff8
     0000000000000008 WA 0 0 8
[24] .got.plt              PROGBITS         0000000000601000 00001000
     0000000000000058 WA 0 0 8
[25] .data                 PROGBITS         0000000000601058 00001058
     0000000000000010 WA 0 0 8
[26] .bss                  NOBITS           0000000000601080 00001068
     0000000000000118 WA 0 0 32
[27] .comment              PROGBITS         0000000000000000 00001068
     0000000000000035 MS 0 0 1
[28] .shstrtab             STRTAB           0000000000000000 00001b6c
     000000000000010c 0 0 1
[29] .symtab               SYMTAB           0000000000000000 000010a0
     0000000000000738 30 50 8
[30] .strtab               STRTAB           0000000000000000 000017d8
     0000000000000394 0 0 1

Key to Flags:
W (write), A (alloc), X (execute), M (merge), S (strings), l (large)
I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)
0 (extra OS processing required) o (OS specific), p (processor specific)
victor@VKCOMPUTRON ~/Documents/UGR/seguridadS0/p2/p2s1/c++ $
```

Finalmente, con el comando `readelf -r` podemos ver las secciones de reubicación de nuestros archivos ELF. Los binarios ELF contienen una tabla de reubicación que especifica las reubicaciones necesarias para la ejecución del programa. Estas reubicaciones suelen corresponder a símbolos, donde el enlazador dinámico resuelve un símbolo por su nombre y escribe su dirección en el lugar indicado en su entrada de reubicación para poder acceder posteriormente a él. Los tipos de reubicaciones dependen de la arquitectura, siendo las más comunes `R_386_COPY`, que indica básicamente “sólo copia la dirección del símbolo en esa dirección”, y `R_386_JUMP_SLOT`, utilizado por la llamada a la función PLT/GOT.

Este último tipo suele aparecer con símbolos como *printf*, *strcpy*, *getpid*..., reflejado con el caso particular de *printf* en la sección de reubicación *rela.plt* de la versión del programa en C, que utiliza la llamada a esta función para imprimir por pantalla.

El contenido de las secciones de reubicación es, por ende, aquellos símbolos o palabras empleados en el programa junto con sus entradas de reubicación necesarias (con valores de desplazamiento entre otros) para obtener sus direcciones y poder acceder a ellas, permitiendo la correcta ejecución del programa.

```
victor@VKCOMPUTRON ~/Documents/UGR/seguridad50/p2/p2s1 $ readelf -r HelloWorld
Relocation section '.rela.dyn' at offset 0x3a0 contains 1 entries:
  Offset      Info          Type             Sym. Value     Sym. Name + Addend
0000000000ff8 000400000006 R_X86_64_GLOB_DAT 0000000000000000 __gmon_start__ + 0

Relocation section '.rela.plt' at offset 0x3b8 contains 3 entries:
  Offset      Info          Type             Sym. Value     Sym. Name + Addend
0000000001018 000100000007 R_X86_64_JUMP_SLO 0000000000000000 puts@GLIBC_2.2.5 + 0
0000000001020 000200000007 R_X86_64_JUMP_SLO 0000000000000000 printf@GLIBC_2.2.5 + 0
0000000001028 000300000007 R_X86_64_JUMP_SLO 0000000000000000 libc_start_main@GLIBC_2.2.5 + 0
victor@VKCOMPUTRON ~/Documents/UGR/seguridad50/p2/p2s1 $ readelf -r HelloWorld++
Relocation section '.rela.dyn' at offset 0x5e8 contains 2 entries:
  Offset      Info          Type             Sym. Value     Sym. Name + Addend
0000000000ff8 000200000006 R_X86_64_GLOB_DAT 0000000000000000 __gmon_start__ + 0
0000000001080 000000000005 R_X86_64_COPY     00000000000001080 ZSt4cout@GLIBCXX_3.4 + 0

Relocation section '.rela.plt' at offset 0x618 contains 8 entries:
  Offset      Info          Type             Sym. Value     Sym. Name + Addend
0000000001018 000100000007 R_X86_64_JUMP_SLO 0000000000000000 ZNSolsEi@GLIBCXX_3.4 + 0
0000000001020 000400000007 R_X86_64_JUMP_SLO 0000000000000000 ZNSt8ios_base4InitC1E@GLIBCXX_3.4 + 0
0000000001028 000500000007 R_X86_64_JUMP_SLO 0000000000000000 libc_start_main@GLIBC_2.2.5 + 0
0000000001030 000600000007 R_X86_64_JUMP_SLO 0000000000000000 cxa_atexit@GLIBC_2.2.5 + 0
0000000001038 000c00000007 R_X86_64_JUMP_SLO 00000000000400750 ZNSt8ios_base4InitD1E@GLIBCXX_3.4 + 0
0000000001040 000800000007 R_X86_64_JUMP_SLO 0000000000000000 ZStlsISt11char_traits@GLIBCXX_3.4 + 0
0000000001048 000a00000007 R_X86_64_JUMP_SLO 0000000000000000 ZNSolsEPFR5oS_E@GLIBCXX_3.4 + 0
0000000001050 000b00000007 R_X86_64_JUMP_SLO 00000000000400780 ZSt4endlcISt11char_tr@GLIBCXX_3.4 + 0
victor@VKCOMPUTRON ~/Documents/UGR/seguridad50/p2/p2s1 $
```

Ejercicio 2.

En este ejercicio, investigamos el comando *objdump*. Leyendo su entrada en el manual (*man objdump*), nos informa que se trata de un comando que muestra información acerca de un fichero, configurando qué información mostrar en función de diversos parámetros. Su diferencia fundamental con *readelf* es que, mientras que ésta vuelca toda la información de un fichero ELF, *objdump* vuelca información genérica acerca de casi cualquier tipo de archivo (incluyendo ELF). A continuación los diversos parámetros posibles a utilizar con *objdump* para **obtener información similar a la obtenida con *readelf***, junto con otras opciones que nos proporcionan información valiosa desde el punto de vista de la ingeniería inversa:

La opción -a muestra la cabecera del archivo.

Si queremos volcar el contenido de las secciones, podemos especificar la opción *-adjust-vma=offset* para establecer el *offset* a sumar a cada una de las secciones (a su dirección de inicio).

La opción *-C* permite decodificar símbolos de bajo nivel a nombres de alto nivel, útil para hacer legibles los nombres de las funciones en C++.

La opción *-g* muestra información de depuración. Intenta parsear la información de debug en formato STABS o IEEE almacenado en el archivo, o si falla trata de imprimir cualquier información DWARF del mismo.

La opción *-d* está pensada para desensamblar aquellas secciones que se espera que contengan instrucciones. La opción *-D* lo hace de todas las secciones.

La opción -f muestra información acerca de la cabecera general del archivo.

La opción -h muestra información acerca de las cabeceras de secciones del archivo.

La opción -j name muestra información sólo de la sección de nombre name.

La opción -r imprime las entradas de reubicación. -R las de reubicación dinámica.

La opción -s muestra el contenido completo de cualquier sección especificada (si no se especifica ninguna, lo hace de todas las secciones no vacías).

La opción -S source muestra código fuente mezclado con ensamblador si es posible.

La opción -S --show-raw-ins permite, a la hora de desensamblar instrucciones, imprimirla tanto en hexadecimal como en su forma simbólica.

La opción -W muestra el contenido de las secciones de debug del fichero.

La opción -G muestra los contenidos de las secciones .stab, .stab.index y .stab.excl de un fichero ELF. Esto es sólo útil en sistemas (como Solaris 2.0) en los cuales las entradas de la tabla de símbolos de depuración .stab se encuentran en una sección del ELF. Junto con la opción --start-address=address muestra información a partir de esa dirección address. La opción --stop-address=address para de mostrar información a partir de esa dirección address.

La opción -t muestra la tabla de símbolos. -T muestra la tabla de símbolos dinámicos.

La opción -x muestra toda la información de cabeceras disponible, incluyendo tabla de símbolos y las entradas de reubicación. Es equivalente a emplear -a -f -h -r -t.

```
victor@VKCOMPUTRON ~/Documents/UGR/seguridad50
File Edit View Search Terminal Help
victor@VKCOMPUTRON ~/Documents/UGR/seguridad50/p2/p2s1 $ objdump -x HelloWorld

HelloWorld:      file format elf64-x86-64
HelloWorld
architecture: i386:x86-64, flags 0x00000112:  EXEC_P, HAS_SYMS, D_PAGED
start address 0x000000000400470

Program Header:
  PHDR off 0x0000000000000040 vaddr 0x0000000000000040 paddr 0x0000000000000040 align 2**3
  filesz 0x00000000000001f8 memsz 0x00000000000001f8 flags r-x
  INTERP off 0x0000000000000238 vaddr 0x0000000000000238 paddr 0x0000000000000238 align 2**0
  filesz 0x000000000000001c memsz 0x000000000000001c flags r--
  LOAD off 0x0000000000000000 vaddr 0x0000000000000000 paddr 0x0000000000000000 align 2**21
  filesz 0x000000000000076c memsz 0x000000000000076c flags r-x
  LOAD off 0x000000000000010 vaddr 0x000000000000010 paddr 0x000000000000010 align 2**21
  filesz 0x0000000000000238 memsz 0x0000000000000238 flags rw-
  DYNAMIC off 0x000000000000028 vaddr 0x000000000000028 paddr 0x000000000000028 align 2**3
  filesz 0x00000000000001d0 memsz 0x00000000000001d0 flags rw-
  NOTE off 0x0000000000000254 vaddr 0x0000000000000254 paddr 0x0000000000000254 align 2**2
  filesz 0x0000000000000044 memsz 0x0000000000000044 flags r--
  EH_FRAME off 0x0000000000000644 vaddr 0x0000000000000644 paddr 0x0000000000000644 align 2**2
  filesz 0x0000000000000034 memsz 0x0000000000000034 flags r--
  STACK off 0x0000000000000000 vaddr 0x0000000000000000 paddr 0x0000000000000000 align 2**4
  filesz 0x0000000000000000 memsz 0x0000000000000000 flags rw-
  RELRO off 0x0000000000000e10 vaddr 0x0000000000000e10 paddr 0x0000000000000e10 align 2**0
  filesz 0x00000000000001f0 memsz 0x00000000000001f0 flags r--

Dynamic Section:
  lib.so.6
  INIT 0x0000000000000400
  FINI 0x0000000000000624
  INIT_ARRAY 0x0000000000000e10
  INIT_ARRAYSZ 0x0000000000000008
  FINI_ARRAY 0x0000000000000e18
  FINI_ARRAYSZ 0x0000000000000008
  GNU_HASH 0x0000000000000298
  STRTAB 0x0000000000000330
  SYMTAB 0x00000000000002b8
  STRSZ 0x0000000000000044
  SYMENT 0x0000000000000018
  DEBUG 0x0000000000000000
  PLTGOT 0x0000000000000100
  PLTRELSZ 0x0000000000000048
  PLTREL 0x0000000000000007
```

Sections:						
Idx	Name	Size	VMA	LMA	File off	Align
0	.interp	0000001c	000000000400238	000000000400238	00000238	2**0
CONTENTS, ALLOC, LOAD, READONLY, DATA						
1	.note.ABI-tag	00000020	000000000400254	000000000400254	00000254	2**2
CONTENTS, ALLOC, LOAD, READONLY, DATA						
2	.note.gnu.build-id	00000024	000000000400274	000000000400274	00000274	2**2
CONTENTS, ALLOC, LOAD, READONLY, DATA						
3	.gnu.hash	0000001c	000000000400298	000000000400298	00000298	2**3
CONTENTS, ALLOC, LOAD, READONLY, DATA						
4	.dynsym	00000078	0000000004002b8	0000000004002b8	000002b8	2**3
CONTENTS, ALLOC, LOAD, READONLY, DATA						
5	.dynstr	00000044	000000000400330	000000000400330	00000330	2**0
CONTENTS, ALLOC, LOAD, READONLY, DATA						
6	.gnu.version	0000000a	000000000400374	000000000400374	00000374	2**1
CONTENTS, ALLOC, LOAD, READONLY, DATA						
7	.gnu.version_r	00000020	000000000400380	000000000400380	00000380	2**3
CONTENTS, ALLOC, LOAD, READONLY, DATA						
8	.rela.dyn	00000018	0000000004003a0	0000000004003a0	000003a0	2**3
CONTENTS, ALLOC, LOAD, READONLY, DATA						
9	.rela.plt	00000048	0000000004003b8	0000000004003b8	000003b8	2**3
CONTENTS, ALLOC, LOAD, READONLY, DATA						
10	.init	0000001a	000000000400400	000000000400400	00000400	2**2
CONTENTS, ALLOC, LOAD, READONLY, CODE						
11	.plt	00000040	000000000400420	000000000400420	00000420	2**4
CONTENTS, ALLOC, LOAD, READONLY, CODE						
12	.plt.got	00000008	000000000400460	000000000400460	00000460	2**3
CONTENTS, ALLOC, LOAD, READONLY, CODE						
13	.text	000001b2	000000000400470	000000000400470	00000470	2**4
CONTENTS, ALLOC, LOAD, READONLY, CODE						
14	.fini	00000009	000000000400624	000000000400624	00000624	2**2
CONTENTS, ALLOC, LOAD, READONLY, CODE						
15	.rodata	00000013	000000000400630	000000000400630	00000630	2**2
CONTENTS, ALLOC, LOAD, READONLY, DATA						
16	.eh_frame_hdr	00000034	000000000400644	000000000400644	00000644	2**2
CONTENTS, ALLOC, LOAD, READONLY, DATA						
17	.eh_frame	000000f4	000000000400678	000000000400678	00000678	2**3
CONTENTS, ALLOC, LOAD, READONLY, DATA						
18	.init_array	00000008	000000000600e10	000000000600e10	00000e10	2**3
CONTENTS, ALLOC, LOAD, DATA						
19	.fini_array	00000008	000000000600e18	000000000600e18	00000e18	2**3
CONTENTS, ALLOC, LOAD, DATA						
20	.jcr	00000008	000000000600e20	000000000600e20	00000e20	2**3
CONTENTS, ALLOC, LOAD, DATA						

Ejercicio 3.

Finalmente, concluimos esta práctica