SEGURIDAD EN SISTEMAS OPERATIVOS
4º Grado en Informática – Complementos de Ing. del Software
Curso 2018-19

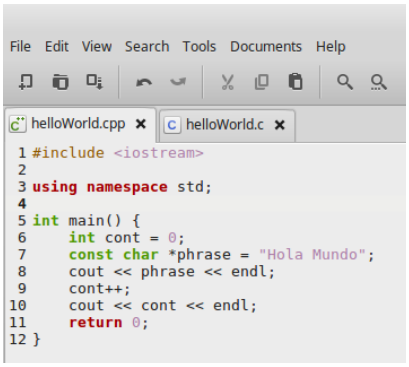
Práctica 2. Ingeniería inversa y vulnerabilidades

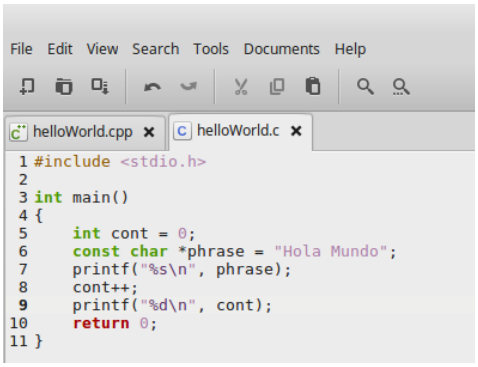
Sesión 1. El formato ELF (*Executable and Linkable Format*) en Linux

Autor[1]: Víctor García Carrera

Ejercicio 1.

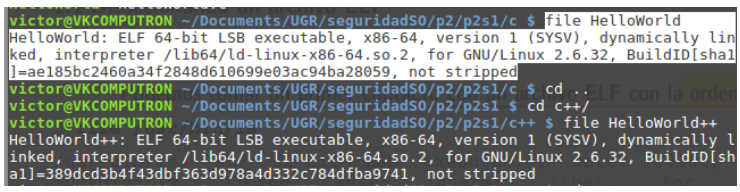
Para conocer la estructura de los archivos ejecutables ELF y las herramientas básicas para ver su contenido, comenzamos creando un programa que imprime por pantalla un mensaje almacenado en una variable de tipo char* *phrase* e imprime por pantalla el valor de una variable de tipo entero *cont* inicializada 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.

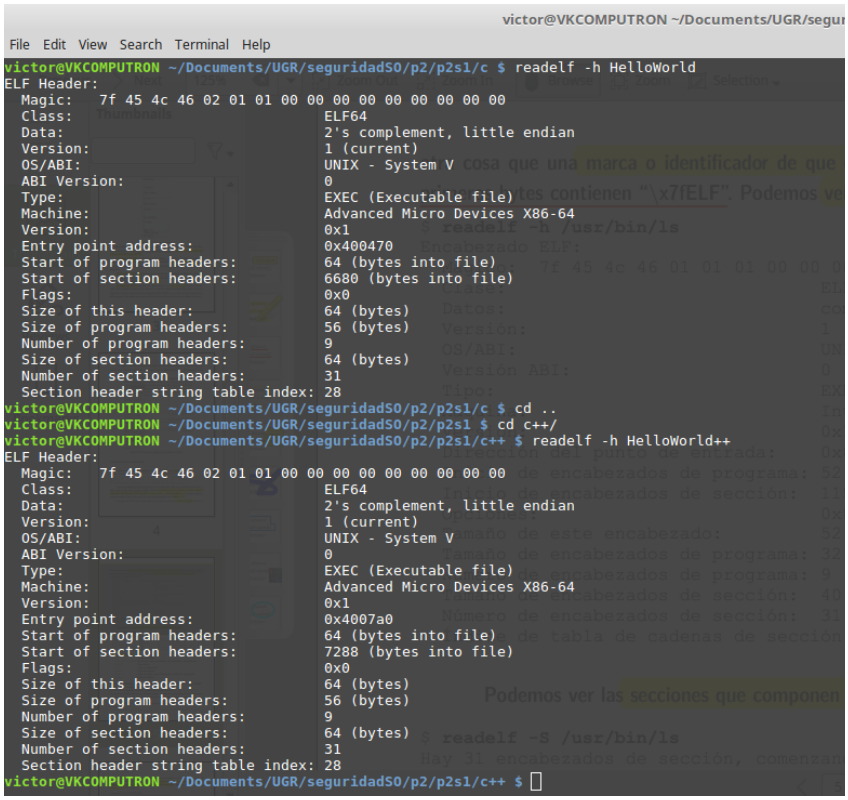
A la hora de observar la estructura y secciones de ambos ejectuables ELF, visualizamos con el comando *file* el tipo de archivos que son, ambos archivos ELF para una arquitectura de 64 bits y little endian (LSB) sin información de depuración (*not stripped*).



---

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++:



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.ptl* 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-i, .gnu.hash, .got* y *.got.plt*, además de un valor distino del campo Info o Align en las secciones *.gnu.version_r* o *.bss*.

```
victor@VKCOMPUTRON ~/Documents/UGR/seguridadSO/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]                   NULL            0000000000000000  00000000
       0000000000000000  0000000000000000           0     0     0
  [ 1] .interp           PROGBITS        0000000000400238  00000238
       000000000000001c  0000000000000000   A       0     0     1
  [ 2] .note.ABI-tag     NOTE            0000000000400254  00000254
       0000000000000020  0000000000000000   A       0     0     4
  [ 3] .note.gnu.build-i NOTE            0000000000400274  00000274
       0000000000000024  0000000000000000   A       0     0     4
  [ 4] .gnu.hash         GNU_HASH        0000000000400298  00000298
       000000000000001c  0000000000000000   A       5     0     8
  [ 5] .dynsym           DYNSYM          00000000004002b8  000002b8
       0000000000000078  0000000000000018   A       6     1     8
  [ 6] .dynstr           STRTAB          0000000000400330  00000330
       0000000000000044  0000000000000000   A       0     0     1
  [ 7] .gnu.version      VERSYM          0000000000400374  00000374
       000000000000000a  0000000000000002   A       5     0     2
  [ 8] .gnu.version_r    VERNEED         0000000000400380  00000380
       0000000000000020  0000000000000000   A       6     1     8
  [ 9] .rela.dyn         RELA            00000000004003a0  000003a0
       0000000000000018  0000000000000018   A       5     0     8
  [10] .rela.plt         RELA            00000000004003b8  000003b8
       0000000000000048  0000000000000018  AI       5    24     8
  [11] .init             PROGBITS        0000000000400400  00000400
       000000000000001a  0000000000000000  AX       0     0     4
  [12] .plt              PROGBITS        0000000000400420  00000420
       0000000000000040  0000000000000010  AX       0     0     16
  [13] .plt.got          PROGBITS        0000000000400460  00000460
       0000000000000008  0000000000000000  AX       0     0     8
  [14] .text             PROGBITS        0000000000400470  00000470
       00000000000001b2  0000000000000000  AX       0     0     16
  [15] .fini             PROGBITS        0000000000400624  00000624
       0000000000000009  0000000000000000  AX       0     0     4
  [16] .rodata           PROGBITS        0000000000400630  00000630
       0000000000000013  0000000000000000   A       0     0     4
  [17] .eh_frame_hdr     PROGBITS        0000000000400644  00000644
       0000000000000034  0000000000000000   A       0     0     4
  [18] .eh_frame         PROGBITS        0000000000400678  00000678
       00000000000000f4  0000000000000000   A       0     0     8
  [19] .init_array       INIT_ARRAY      0000000000600e10  00000e10
```

```
  [19] .init_array       INIT_ARRAY      0000000000600e10  00000e10
       0000000000000008  0000000000000000  WA       0     0     8
  [20] .fini_array       FINI_ARRAY      0000000000600e18  00000e18
       0000000000000008  0000000000000000  WA       0     0     8
  [21] .jcr              PROGBITS        0000000000600e20  00000e20
       0000000000000008  0000000000000000  WA       0     0     8
  [22] .dynamic          DYNAMIC         0000000000600e28  00000e28
       00000000000001d0  0000000000000010  WA       6     0     8
  [23] .got              PROGBITS        0000000000600ff8  00000ff8
       0000000000000008  0000000000000008  WA       0     0     8
  [24] .got.plt          PROGBITS        0000000000601000  00001000
       0000000000000030  0000000000000008  WA       0     0     8
  [25] .data             PROGBITS        0000000000601030  00001030
       0000000000000010  0000000000000000  WA       0     0     8
  [26] .bss              NOBITS          0000000000601040  00001040
       0000000000000008  0000000000000000  WA       0     0     1
  [27] .comment          PROGBITS        0000000000000000  00001040
       0000000000000035  0000000000000001  MS       0     0     1
  [28] .shstrtab         STRTAB          0000000000000000  00001905
       000000000000010c  0000000000000000           0     0     1
  [29] .symtab           SYMTAB          0000000000000000  00001078
       0000000000000660  0000000000000018          30    47     8
  [30] .strtab           STRTAB          0000000000000000  000016d8
       000000000000022d  0000000000000000           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)
  O (extra OS processing required) o (OS specific), p (processor specific)
victor@VKCOMPUTRON ~/Documents/UGR/seguridadSO/p2/p2s1/c $
```

```
victor@VKCOMPUTRON ~/Documents/UGR/seguridadSO/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]                   NULL            0000000000000000  00000000
       0000000000000000  0000000000000000           0     0     0
  [ 1] .interp           PROGBITS        0000000000400238  00000238
       000000000000001c  0000000000000000   A       0     0     1
  [ 2] .note.ABI-tag     NOTE            0000000000400254  00000254
       0000000000000020  0000000000000000   A       0     0     4
  [ 3] .note.gnu.build-i NOTE            0000000000400274  00000274
       0000000000000024  0000000000000000   A       0     0     4
  [ 4] .gnu.hash         GNU_HASH        0000000000400298  00000298
       0000000000000030  0000000000000000   A       5     0     8
  [ 5] .dynsym           DYNSYM          00000000004002c8  000002c8
       0000000000000150  0000000000000018   A       6     1     8
  [ 6] .dynstr           STRTAB          0000000000400418  00000418
       0000000000000172  0000000000000000   A       0     0     1
  [ 7] .gnu.version      VERSYM          000000000040058a  0000058a
       000000000000001c  0000000000000002   A       5     0     2
  [ 8] .gnu.version_r    VERNEED         00000000004005a8  000005a8
       0000000000000040  0000000000000000   A       6     2     8
  [ 9] .rela.dyn         RELA            00000000004005e8  000005e8
       0000000000000030  0000000000000018   A       5     0     8
  [10] .rela.plt         RELA            0000000000400618  00000618
       00000000000000c0  0000000000000018  AI       5    24     8
  [11] .init             PROGBITS        00000000004006d8  000006d8
       000000000000001a  0000000000000000  AX       0     0     4
  [12] .plt              PROGBITS        0000000000400700  00000700
       0000000000000090  0000000000000010  AX       0     0     16
  [13] .plt.got          PROGBITS        0000000000400790  00000790
       0000000000000008  0000000000000000  AX       0     0     8
  [14] .text             PROGBITS        00000000004007a0  000007a0
       0000000000000222  0000000000000000  AX       0     0     16
  [15] .fini             PROGBITS        00000000004009c4  000009c4
       0000000000000009  0000000000000000  AX       0     0     4
  [16] .rodata           PROGBITS        00000000004009d0  000009d0
       000000000000000f  0000000000000000   A       0     0     4
  [17] .eh_frame_hdr     PROGBITS        00000000004009e0  000009e0
       0000000000000044  0000000000000000   A       0     0     4
  [18] .eh_frame         PROGBITS        0000000000400a28  00000a28
       0000000000000134  0000000000000000   A       0     0     8
  [19] .init_array       INIT_ARRAY      0000000000600df8  00000df8
```

```
  [19] .init_array       INIT_ARRAY      0000000000600df8  00000df8
       0000000000000010  0000000000000000  WA       0     0     8
  [20] .fini_array       FINI_ARRAY      0000000000600e08  00000e08
       0000000000000008  0000000000000000  WA       0     0     8
  [21] .jcr              PROGBITS        0000000000600e10  00000e10
       0000000000000008  0000000000000000  WA       0     0     8
  [22] .dynamic          DYNAMIC         0000000000600e18  00000e18
       00000000000001e0  0000000000000010  WA       6     0     8
  [23] .got              PROGBITS        0000000000600ff8  00000ff8
       0000000000000008  0000000000000008  WA       0     0     8
  [24] .got.plt          PROGBITS        0000000000601000  00001000
       0000000000000058  0000000000000008  WA       0     0     8
  [25] .data             PROGBITS        0000000000601058  00001058
       0000000000000010  0000000000000000  WA       0     0     8
  [26] .bss              NOBITS          0000000000601080  00001068
       0000000000000118  0000000000000000  WA       0     0     32
  [27] .comment          PROGBITS        0000000000000000  00001068
       0000000000000035  0000000000000001  MS       0     0     1
  [28] .shstrtab         STRTAB          0000000000000000  00001b6c
       000000000000010c  0000000000000000           0     0     1
  [29] .symtab           SYMTAB          0000000000000000  000010a0
       0000000000000738  0000000000000018          30    50     8
  [30] .strtab           STRTAB          0000000000000000  000017d8
       0000000000000394  0000000000000000           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)
  O (extra OS processing required) o (OS specific), p (processor specific)
victor@VKCOMPUTRON ~/Documents/UGR/seguridadSO/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_SLO*T, 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/seguridadSO/p2/p2s1 $ readelf -r HelloWorld

Relocation section '.rela.dyn' at offset 0x3a0 contains 1 entries:
  Offset          Info           Type           Sym. Value    Sym. Name + Addend
000000600ff8  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
000000601018  000100000007 R_X86_64_JUMP_SLO 0000000000000000 puts@GLIBC_2.2.5 + 0
000000601020  000200000007 R_X86_64_JUMP_SLO 0000000000000000 printf@GLIBC_2.2.5 + 0
000000601028  000300000007 R_X86_64_JUMP_SLO 0000000000000000 __libc_start_main@GLIBC_2.2.5 + 0
victor@VKCOMPUTRON ~/Documents/UGR/seguridadSO/p2/p2s1 $ readelf -r HelloWorld++

Relocation section '.rela.dyn' at offset 0x5e8 contains 2 entries:
  Offset          Info           Type           Sym. Value    Sym. Name + Addend
000000600ff8  000200000006 R_X86_64_GLOB_DAT 0000000000000000 __gmon_start__ + 0
000000601080  000d00000005 R_X86_64_COPY     0000000000601080 _ZSt4cout@GLIBCXX_3.4 + 0

Relocation section '.rela.plt' at offset 0x618 contains 8 entries:
  Offset          Info           Type           Sym. Value    Sym. Name + Addend
000000601018  000100000007 R_X86_64_JUMP_SLO 0000000000000000 _ZNSolsEi@GLIBCXX_3.4 + 0
000000601020  000400000007 R_X86_64_JUMP_SLO 0000000000000000 _ZNSt8ios_base4InitC1E@GLIBCXX_3.4 + 0
000000601028  000500000007 R_X86_64_JUMP_SLO 0000000000000000 __libc_start_main@GLIBC_2.2.5 + 0
000000601030  000600000007 R_X86_64_JUMP_SLO 0000000000000000 __cxa_atexit@GLIBC_2.2.5 + 0
000000601038  000c00000007 R_X86_64_JUMP_SLO 0000000000400750 _ZNSt8ios_base4InitD1E@GLIBCXX_3.4 + 0
000000601040  000800000007 R_X86_64_JUMP_SLO 0000000000000000 _ZStlsISt11char_traits@GLIBCXX_3.4 + 0
000000601048  000a00000007 R_X86_64_JUMP_SLO 0000000000000000 _ZNSolsEPFRSoS_E@GLIBCXX_3.4 + 0
000000601050  000b00000007 R_X86_64_JUMP_SLO 0000000000400780 _ZSt4endlIcSt11char_tr@GLIBCXX_3.4 + 0
victor@VKCOMPUTRON ~/Documents/UGR/seguridadSO/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.

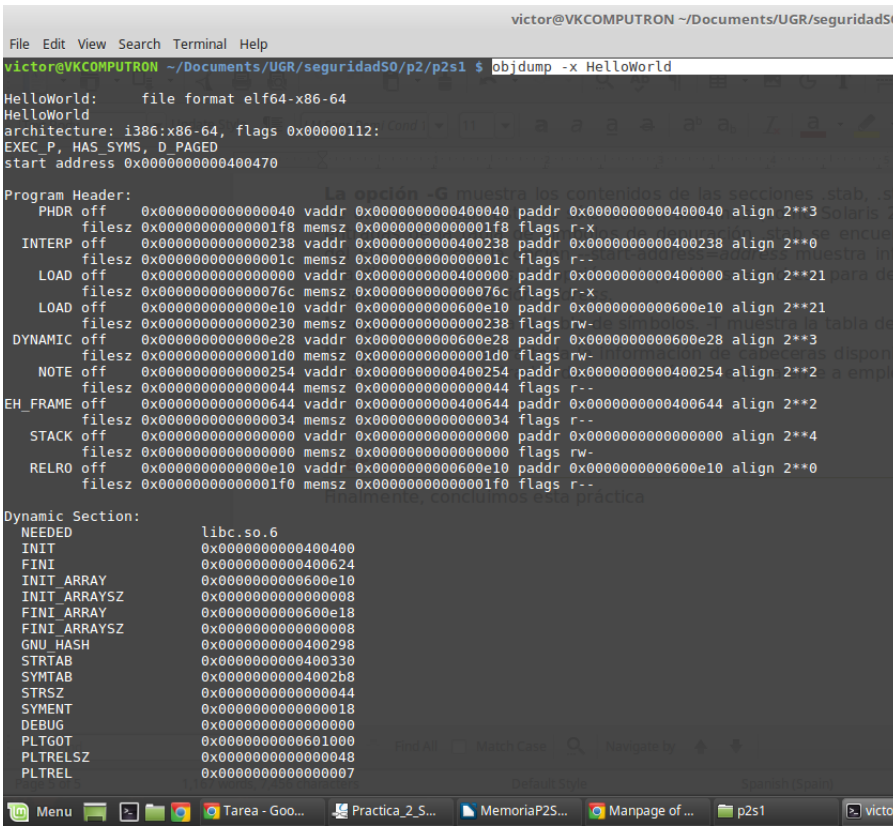
La opción -S --show-raw-ins permite, a la hora de desensamblar instrucciones, imprimirla tanto en hexadecimal como en su forma simbólica.

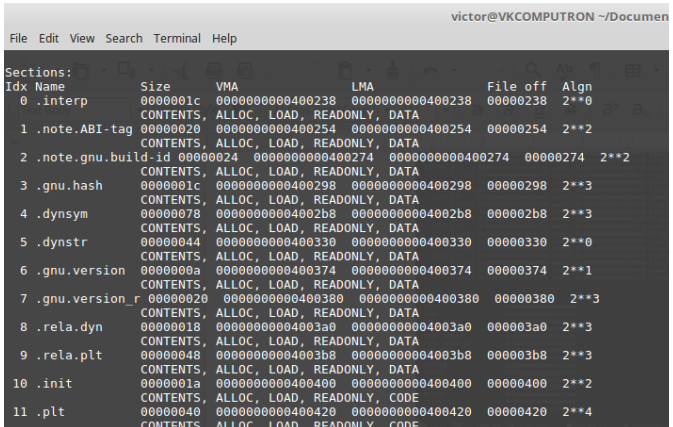
La opción -W muestra el contenido de las secciones de debug del fichero.

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.





Ejercicio 3.

Finalmente, concluimos esta práctica modificando el archivo *helloWorld.c* para crear el ejecutable *HelloWorld2* que ahora imprime por pantalla el *PID* del proceso que está ejecutando el programa, así como se queda bloqueado a fin de poder abrir otra terminal y ejecutar el comando *cat /proc/PID/maps* para visualizar la construcción que se ha llevado a cabo del proceso que ejecuta el programa. Nos interesa ver las direcciones y permisos de las diferentes regiones de memoria del proceso pues se

forman a partir de la información contenida en las cabeceras del fichero ELF. A continuación se muestran las salidas obtenidas:

File Edit View Search Tools Documents Help

helloWorld.c

```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
    int cont = 0;
    const char *phrase = "Hola Mundo";
    printf("%s PID: %d\n", phrase, getpid());
    sleep(10000);
    cont++;
    printf("%d\n", cont);
    return 0;
}
```

victor@VKCOMPUTRON ~/Documents/UGR/seguridadSO/p2/p2s1

File Edit View Search Terminal Help

```
victor@VKCOMPUTRON ~/Documents/UGR/seguridadSO/p2/p2s1 $ ./HelloWorld2
Hola Mundo PID: 29994
```

victor@VKCOMPUTRON ~

File Edit View Search Terminal Help

```
victor@VKCOMPUTRON ~ $ cat /proc/29994/maps
00400000-00401000 r-xp 00000000 08:05 21497115                           /home/victor/Documents/UGR/seguridadSO/p2/p2s1/HelloWorld2
00600000-00601000 r--p 00000000 08:05 21497115                           /home/victor/Documents/UGR/seguridadSO/p2/p2s1/HelloWorld2
00601000-00602000 rw-p 00001000 08:05 21497115                           /home/victor/Documents/UGR/seguridadSO/p2/p2s1/HelloWorld2
013ea000-0140b000 rw-p 00000000 00:00 0                                  [heap]
7f915980f000-7f91599cf000 r-xp 00000000 08:05 17831935                   /lib/x86_64-linux-gnu/libc-2.23.so
7f91599cf000-7f9159bcf000 ---p 001c0000 08:05 17831935                   /lib/x86_64-linux-gnu/libc-2.23.so
7f9159bcf000-7f9159bd3000 r--p 001c0000 08:05 17831935                   /lib/x86_64-linux-gnu/libc-2.23.so
7f9159bd3000-7f9159bd5000 rw-p 001c4000 08:05 17831935                   /lib/x86_64-linux-gnu/libc-2.23.so
7f9159bd5000-7f9159bd9000 rw-p 00000000 00:00 0
7f9159bd9000-7f9159bff000 r-xp 00000000 08:05 17831928                   /lib/x86_64-linux-gnu/ld-2.23.so
7f9159dd1000-7f9159dd4000 rw-p 00000000 00:00 0
7f9159dfe000-7f9159dff000 r--p 00025000 08:05 17831928                   /lib/x86_64-linux-gnu/ld-2.23.so
7f9159dff000-7f9159e00000 rw-p 00026000 08:05 17831928                   /lib/x86_64-linux-gnu/ld-2.23.so
7f9159e00000-7f9159e01000 rw-p 00000000 00:00 0
7fffee464000-7fffee485000 rw-p 00000000 00:00 0                          [stack]
7fffee55b000-7fffee55d000 r--p 00000000 00:00 0                          [vvar]
7fffee55d000-7fffee55f000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
victor@VKCOMPUTRON ~ $
```

Si ejecutamos este proceso repetidas veces, veremos que algunas direcciones referentes a regiones contiguas de memoria virtual (cada linea representa una de estas regiones) no cambian, como por ejemplo las correspondientes a las 3 primeras líneas y la región de *vsyscall*. Esto refleja que tanto la región de *vsyscall* como las regiones de datos del programa y memoria ejecutable siempre son fijas (podríamos modificarlo con PIE, donde incluso las direcciones de estas regiones se aleatorizan), a diferencia del resto que son aleatorias como resultado del método de protección ASLR. Se ha tomado como estandar que los ejecutables ELF de 64 bits sin PIE tienen una dirección de entrada de 0x400000, mientras que los ejecutables ELF de 32 bits sin PIE tienen una dirección de entrada de 0x08048000.

```
victor@VKCOMPUTRON ~
File  Edit  View  Search  Terminal  Help
victor@VKCOMPUTRON ~ $ readelf -S /home/victor/Documents/UGR/seguridadSO/p2/p2s1/HelloWorld2
There are 31 section headers, starting at offset 0x1a48:

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         0000000000400238  00000238
       000000000000001c  0000000000000000   A       0     0     1
  [ 2] .note.ABI-tag     NOTE             0000000000400254  00000254
       0000000000000020  0000000000000000   A       0     0     4
  [ 3] .note.gnu.build-i NOTE             0000000000400274  00000274
       0000000000000024  0000000000000000   A       0     0     4
  [ 4] .gnu.hash         GNU_HASH         0000000000400298  00000298
       000000000000001c  0000000000000000   A       5     0     8
  [ 5] .dynsym           DYNSYM           00000000004002b8  000002b8
       0000000000000090  0000000000000018   A       6     1     8
  [ 6] .dynstr           STRTAB           0000000000400348  00000348
       000000000000004c  0000000000000000   A       0     0     1
  [ 7] .gnu.version      VERSYM           0000000000400394  00000394
       000000000000000c  0000000000000002   A       5     0     2
  [ 8] .gnu.version_r    VERNEED          00000000004003a0  000003a0
       0000000000000020  0000000000000000   A       6     1     8
  [ 9] .rela.dyn         RELA             00000000004003c0  000003c0
       0000000000000018  0000000000000018   A       5     0     8
  [10] .rela.plt         RELA             00000000004003d8  000003d8
       0000000000000060  0000000000000018  AI       5    24     8
  [11] .init             PROGBITS         0000000000400438  00000438
       000000000000001a  0000000000000000  AX       0     0     4
  [12] .plt              PROGBITS         0000000000400460  00000460
       0000000000000050  0000000000000010  AX       0     0     16
  [13] .plt.got          PROGBITS         00000000004004b0  000004b0
       0000000000000008  0000000000000000  AX       0     0     8
  [14] .text             PROGBITS         00000000004004c0  000004c0
       00000000000001d2  0000000000000000  AX       0     0     16
  [15] .fini             PROGBITS         0000000000400694  00000694
       0000000000000009  0000000000000000  AX       0     0     4
  [16] .rodata           PROGBITS         00000000004006a0  000006a0
       000000000000001f  0000000000000000   A       0     0     4
  [17] .eh_frame_hdr     PROGBITS         00000000004006c0  000006c0
       0000000000000034  0000000000000000   A       0     0     4
  [18] .eh_frame         PROGBITS         00000000004006f8  000006f8
       00000000000000f4  0000000000000000   A       0     0     8
  [19] .init_array       INIT_ARRAY       0000000000600e10  00000e10
```

```
victor@VKCOMPUTRON ~
File  Edit  View  Search  Terminal  Help
  [22] .dynamic          DYNAMIC          0000000000600e28  00000e28
       00000000000001d0  0000000000000010  WA       6     0     8
  [23] .got              PROGBITS         0000000000600ff8  00000ff8
       0000000000000008  0000000000000008  WA       0     0     8
  [24] .got.plt          PROGBITS         0000000000601000  00001000
       0000000000000038  0000000000000008  WA       0     0     8
  [25] .data             PROGBITS         0000000000601038  00001038
       0000000000000010  0000000000000000  WA       0     0     8
  [26] .bss              NOBITS           0000000000601048  00001048
       0000000000000008  0000000000000000  WA       0     0     1
```

```
victor@VKCOMPUTRON ~ $ readelf -h /home/victor/Documents/UGR/seguridadSO/p2/p2s1/HelloWorld2
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:               0x4004c0
  Start of program headers:          64 (bytes into file)
  Start of section headers:          6728 (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
```

A partir de la información de las cabeceras ELF, tanto de secciones como de segmentos, podemos deducir y entrever la construcción del proceso que ejecuta nuestro programa. Por ejemplo, la cabecera de programa ELF indica que la dirección de entrada es la 0x4004c0 (**dirección de la sección *.text***). Sin embargo, la información mostrada en proc establecía que el programa empieza en la dirección 0x400000. La dirección proporcionada por *readelf* hace referencia al primer "procedimiento" que es ejecutado por la CPU, en este caso la función main, apuntando por ello a la dirección de la sección *.text*. El espacio entre la dirección 0x400000 y la 0x4004c0 es ocupado por las cabeceras ELF y las cabeceras del programa.

Podemos seguir deduciendo cómo se ha construido finalmente el proceso ejecutable a partir de la información contenida en el fichero ELF. La estructura de este proceso quedaría resumida de la siguiente forma:

0

(Nada aquí, por elección arbitraria del enlazador)

Cabeceras de programa y secciones ELF – 0x400000 en nuestra arquitectura de 64 bits.

Texto del programa (sección .text) – 0x4004c0 Punto de entrada proporcionado por *readelf*

Nada

Desconocido ensamblador y datos – 0x600000

Datos Inicializados (sección .data) – 0x601038

Datos sin inicializar (sección .bss) – 0x601048

Heap

Region mapeada de memoria para librerías compartidas o cualquier otra cosa

Pila de usuario