SEGURIDAD EN SISTEMAS OPERATIVOS

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

Curso 2019-20

Práctica [2]. Ingeniería inversa y vulnerabilidades.

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

Autor[1]: Nazaret Román Guerrero

Ejercicio 1.

Construye y compila un programa simple, por ejemplo uno similar al "Hola, Mundo", en dos versiones: una en C y otra en C++.

a) Consulta los manuales o en Internet qué contienen las secciones .interp, .got y .got.ptl.

b) Compara los ELFs de las dos versiones listando las secciones. ¿Hay alguna diferencia relevante respecto a las secciones de un programa compilado con gcc? ¿Qué contienen las secciones .ctors y .dtors?

c) Con la opción readelf -r podemos ver las secciones de reubicación. Indicar qué contienen estas secciones.

a) Cada sección contiene:

- .interp: contiene el path del intérprete del programa. Si el archivo contiene un segmento cargable que incluya la reubicación, incluirá SHF_ALLOC entre sus atributos.
- .got: contiene los offsets para la reubicación de variables globales.
- .got.ptl: contiene los offsets utilizados para llamar a funciones y procecimientos externos al programa.

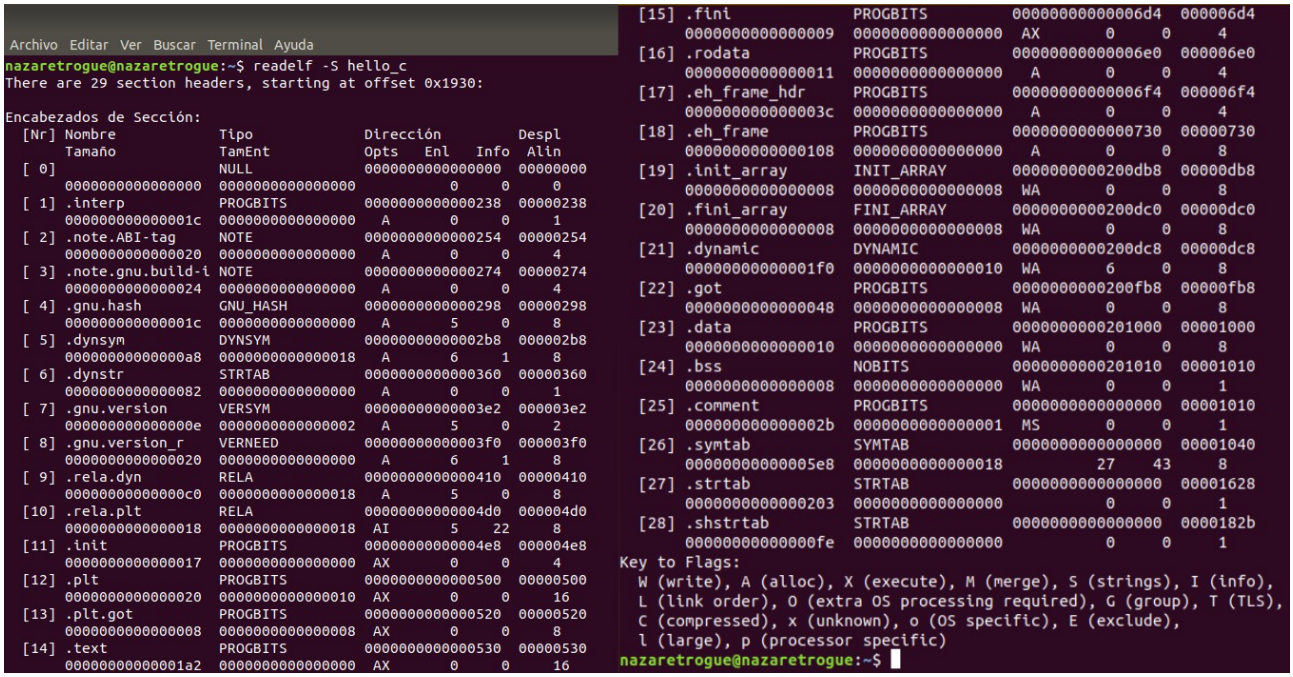
b) Los programas hechos son dos "Hola Mundo" en C y en C++. Las secciones de cada uno se verán a continuación. Antes de nada, para poder compilar los programas es necesario tener instalado en el sistema el compilador de C, gcc, y el de C++, g++.

El código de cada programa está añadido al final de este documento, en un anexo con los códigos para los distintos ejercicios.

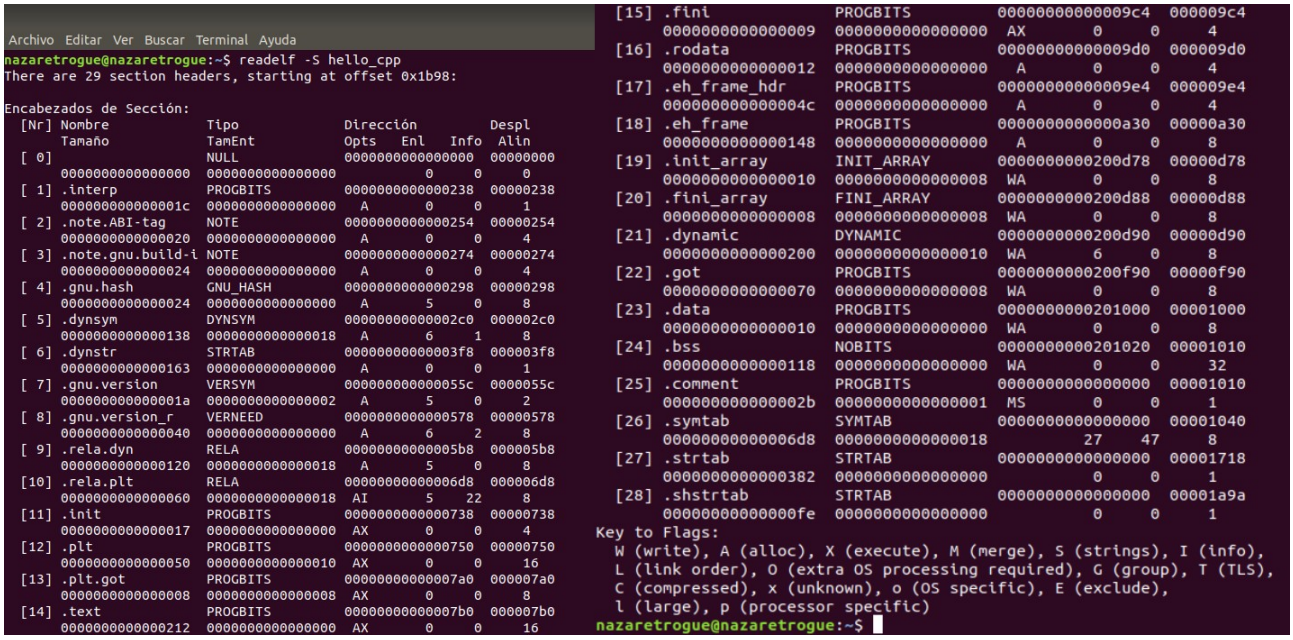
El listado de secciones del programa en C es el siguiente:

---

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 ..."



El listado de secciones para el programa en C++ es el que sigue:



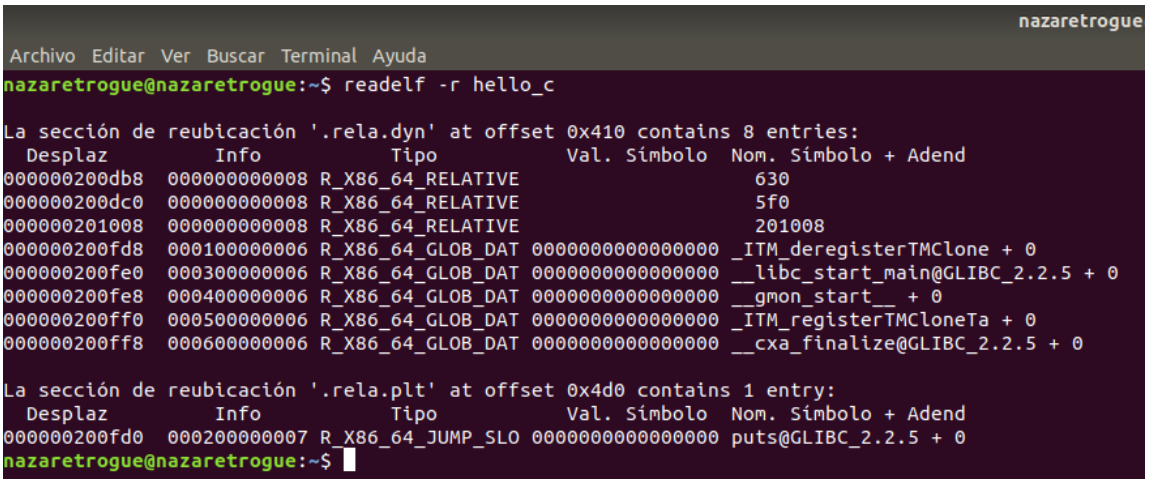
Como se puede observar, no hay diferencias respecto a las secciones; las únicas diferencias se corresponden, como es normal, a la ubicación del programa en memoria, sus variables y procedimientos, ya que son dos programas diferentes.

La sección .ctors contiene una lista global con punteros a los constructores, no obstante no aparece dicha sección aquí porque no hay ninguna declaración de variables ni clases creadas con constructores definidos.

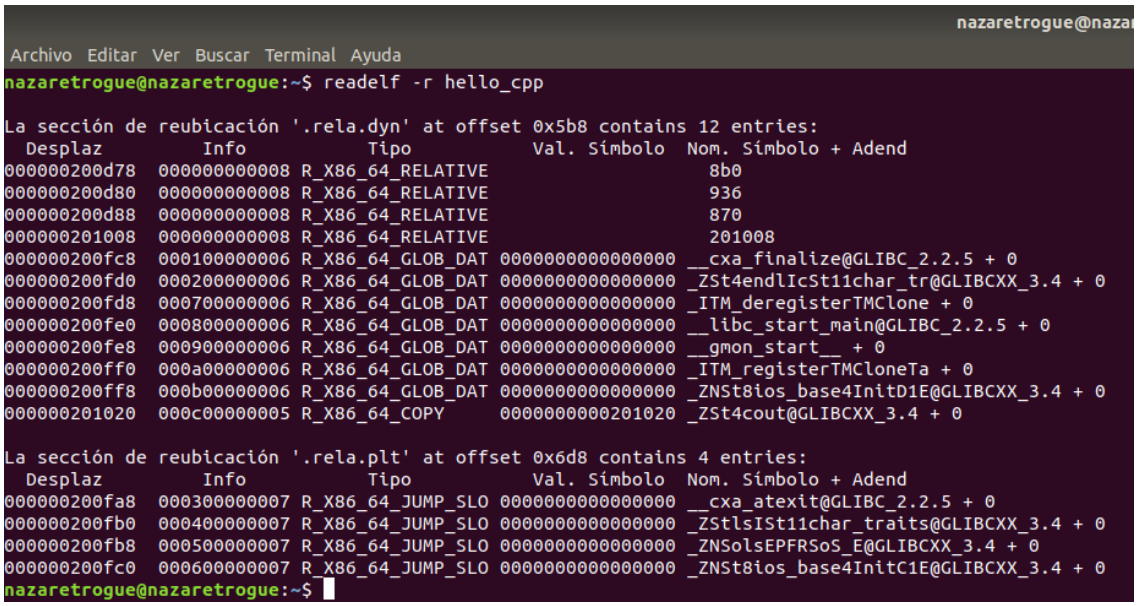
La sección .dtors contiene una lista con los destructores, pero al igual que con .ctors no existe dicha sección porque no hay destructores declarados en el programa ni se están destruyendo variables de otras clases.

c) La salida del comando readelf -r para ambos programas es diferente:

- Para el programa en C la salida es la siguiente:



- Para el programa en C++ la salida es:



En ambos programas, hay dos secciones, de reubicación .rela.dyn y .rela.plt.

La primera, .rela.dyn (runtime/dynamic relocation table), contiene la información sobre la reubicación de las variables en tiempo de carga para aquellos ejecutables que utilizan memoria dinámica.

La segunda, .rela.plt (runtime/dynamic relocation table), es similar a .rela.dyn con la diferencia de que es para funciones y procedimientos en lugar de variables.

Ejercicio 2.

Mira en el manual en línea o en Internet las opciones de la orden objdump, e indica:

a) qué opciones nos permiten ver la información que nos suministra readelf,

b) y qué otras opciones nos permite realizar objdump desde el punto de la ingeniería inversa.

a) Con objdump podemos mostrar la información de archivos .obj.

Si lo que queremos es extraer por ejemplo la información que muestra la orden readelf -r, que muestra las secciones de reubicación, utilizando objdump, debemos utilizar la opción –r y –R (o --reloc y --dynamic-reloc) que muestran la relocation y la dynamic relocation respectivamente, por lo que sería algo como

- readelf -r <archivo> = objdump -rR <archivo>

Si creemos mostrar toda la información igual que lo hace la orden readelf -a, el equivalente sería utilizar las siguientes opciones:

- -f o --file-headers
- -x o --all-headers
- -j <section name> o --section=<section name>
- -t o --syms
- -T o --dynamic-syms
- -r o --reloc
- -R o --dynamic-reloc
- -V o --version
- -m <machine architecture> o --architecture=<machine>

De modo que el comando quedaría como

- readelf -a <archivo> = objdump -fxtTrRV -j <section> -m <architecture> <archivo>

b) La orden objdump nos permite desensamblar un archivo .obj lo que nos permite ver las instrucciones en ensamblador del programa según la máquina en la que se haya compilado. Esto es muy útil a la hora de intentan descubrir el funcionamiento de un programa cuando solo tenemos el binario, y es muy importante en ingeniería inversa precisamente por eso, porque analizando las instrucciones podemos saber si es, por ejemplo, un malware.

Las opciones que permiten desensamblar son -d, -D y -z:

- -d permite desensamblar el programa y sacar aquellas instrucciones que llevan a cabo una función dentro de éste.
- -D desensambla el código completo, no solo las partes que contienen instrucciones, es decir, cuando se utiliza esta opción, la suposición de que los símbolos presentes más allá de los límites de las instrucciones no se desensamblan desaparece, por lo que se está desensamblando más allá del límite del código del propio programa.

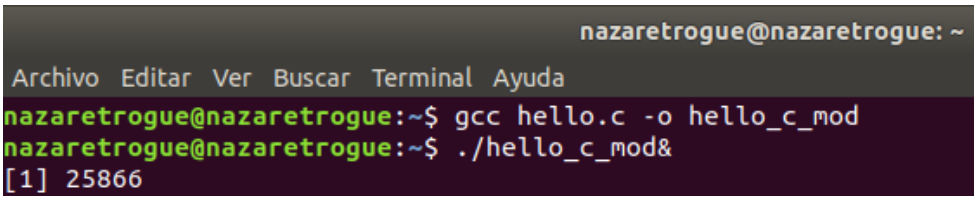
- -z desensambla todo el código, incluyendo los bloques rellenos de 0 del programa, que por lo general no se muestran en la salida del desensamblado.

Ejercicio 3.

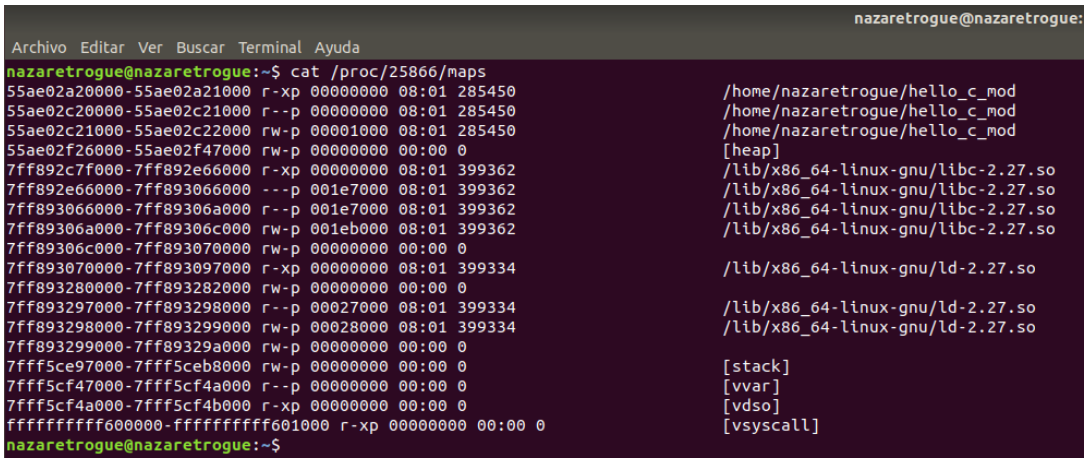
Modifica el programa realizado en el ejercicio anterior para que el programa se detenga durante un rato, por ejemplo, con un sleep(), al objeto de que podamos visualizar el archivo /proc/<PID>/maps de su ejecución. Ahora analiza la información de su ELF para entrever cómo se ha construido dicho proceso a través de la información del ELF, por ejemplo, las direcciones y permisos de las regiones de texto y datos, etc.

El código con la correspondiente modificación se encuentran en el anexo, al final de este mismo documento.

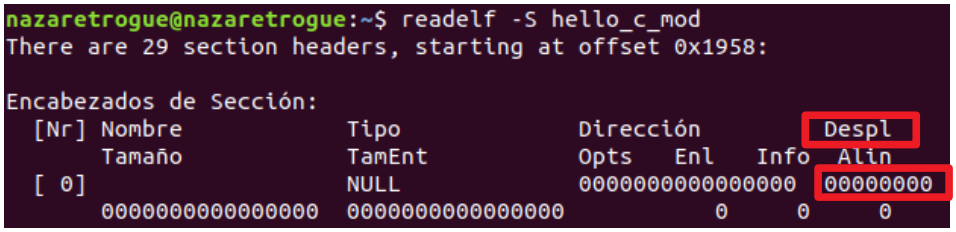
Vamos a ejecutar la versión en C. Para que no se quede la terminal colgada mientras ejecuta el programa, lo lanzamos en background:



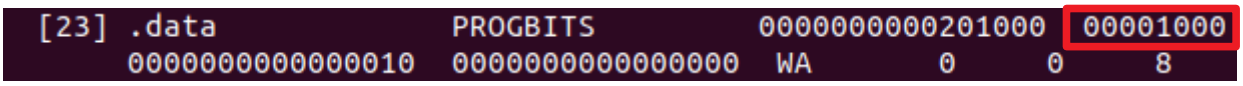
El PID es 25866, que nos sirve para buscar el archivo /proc/25866/maps que necesitamos, cuya salida es la siguiente:



Comparando con la salida del comando readelf -S, podemos ver que la primera línea y segunda línea del archivo maps se corresponden con el código y los datos del programa, puesto que el desplazamiento es 0, como se ve en la imagen:



También podemos apreciar que la tercera línea de maps se corresponde con las variables estáticas del programa, puesto que el desplazamiento es 1000:



Estas secciones tienen permisos de lectura y ejecución y mapeo privado (el código), lectura y mapeo privado para los datos del programa y lectura y escritura y mapeo privado para las variables estáticas.

Anexo.

```c
#include <stdio.h>

int main(){
        printf("Hello world!\n");

        return 0;
}
```

Figure 1: hello.c

```cpp
#include <iostream>

using namespace std;

int main(){
        cout << "Hello world! << endl;

        return 0;
}
```

Figure 2: hello.cpp

```c
#include <stdio.h>
#include <unistd.h>

int main(){
        printf("Hello world!\n");
        sleep(1000); # El programa queda bloqueado durante 1000 segundos, unos 16 minutos
        return 0;
}
```

Figure 3: hello_modificado.c

Bibliografía.

- http://bottomupcs.sourceforge.net/csbu/x3824.htm
- https://reverseengineering.stackexchange.com/questions/1992/what-is-plt-got
- https://stackoverflow.com/questions/11676472/what-is-the-difference-between-got-and-got-plt-section
- https://www.cs.stevens.edu/~jschauma/631A/elf.html