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.

Autor¹: Nazaret Román Guerrero

Ejercicio 1.

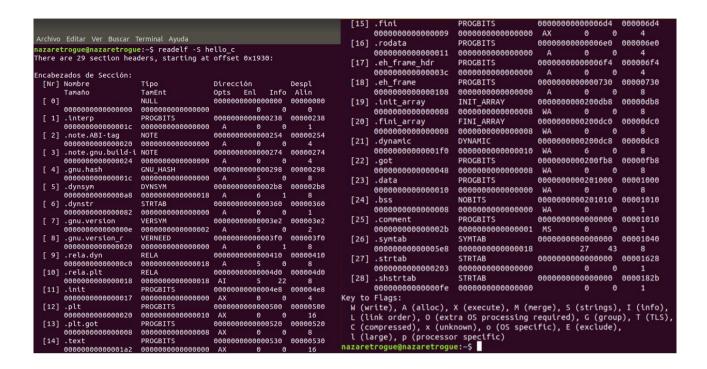
Construye y compila un programa simple, por ejemplo uno similar al "Hola, Mundo", en dos versiones: una en C y otra en C++.

- 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.
 - .qot: 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++.

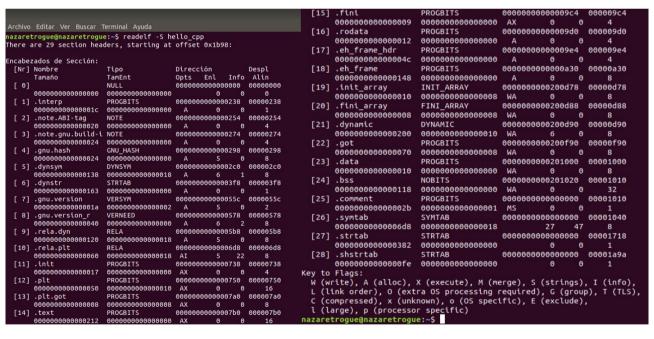
El código de cada programa está añadido al final de este documento, en un anexo con los códigos para los distintos ejercicios.

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:

```
nazaretrogue
 nazaretrogue@nazaretrogue:~$ readelf -r hello c
 a sección de reubicación '.rela.dyn' at offset 0x410 contains 8 entries:
  Desplaz
                        Info
                                             Tipo
                                                                 Val. Símbolo Nom. Símbolo + Adend
 000000200db8 000000000008 R_X86_64_RELATIVE
                                                                                       630
0000000200dc0 000000000008 R_X86_64_RELATIVE 5f0
000000201008 000000000008 R_X86_64_RELATIVE 201008
000000200fd8 000100000006 R_X86_64_GLOB_DAT 00000000000000 _ITM_deregisterTMClone + 0
000000200fe0 000300000006 R_X86_64_GLOB_DAT 00000000000000000000___llbc_start_main@GLIBC_2.2.5 + 0
000000200fe8 000400000006 R_X86_64_GLOB_DAT 000000000000000 __gmon_start__ + 0
000000200ff0 0005000000006 R_X86_64_GLOB_DAT 000000000000000 _ITM_registerTMCloneTa + 0
                                                                                    __cxa_finalize@GLIBC_2.2.5 + 0
000000200ff8
                  000600000006 R_X86_64_GLOB_DAT 0000000000000000
 a sección de reubicación '.rela.plt' at offset 0x4d0 contains 1 entry:
                                                                 Val. Símbolo Nom. Símbolo + Adend
  Desplaz
                        Info
                                             Tipo
000000200fd0 000200000007 R_X86_64_JUMP_SLO 00000000000000 puts@GLIBC_2.2.5 + 0
 nazaretrogue@nazaretrogue:~$
```

Para el programa en C++ la salida es:

```
nazaretroque@naza
nazaretrogue@nazaretrogue:~$ readelf -r hello_cpp
a sección de reubicación '.rela.dyn' at offset 0x5b8 contains 12 entries:
 Desplaz
                                              Val. Símbolo Nom. Símbolo + Adend
               Info
                               Tipo
            000000000008 R_X86_64_RELATIVE
000000000008 R_X86_64_RELATIVE
0000000000008 R_X86_64_RELATIVE
0000000000008 R_X86_64_RELATIVE
.
000000200d78
                                                             8b0
000000200d80
                                                              936
900000200d88
                                                              870
000000201008
                                                              201008
            00000200fc8
900000200fd0
000000200fd8
000000200fe0
000000200fe8
000000200ff0
000000200ff8
000000201020
             000c00000005 R_X86_64_COPY
                                           0000000000201020 ZSt4cout@GLIBCXX_3.4 + 0
a sección de reubicación '.rela.plt' at offset 0x6d8 contains 4 entries:
Desplaz Info Tipo Val. Símbolo Nom. Símbolo + Adend
            000000200fa8
000000200fb0
000000200fb8
000000200fc0
 azaretrogue@nazaretrogue:~$
```

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.

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
- -i <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:

```
nazaretrogue@nazaretrogue: ~
Archivo Editar Ver Buscar Terminal Ayuda
nazaretrogue@nazaretrogue: ~$ gcc hello.c -o hello_c_mod
nazaretrogue@nazaretrogue: ~$ ./hello_c_mod&
[1] 25866
```

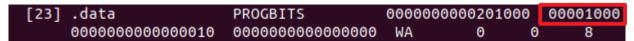
El PID es 25866, que nos sirve para buscar el archivo /proc/25866/maps que necesitamos, cuya salida es la siguiente:

```
nazaretrogue@nazaretrogue
nazaretrogue@nazaretrogue:~$ cat /proc/25866/maps
55ae02a20000-55ae02a21000 r-xp 00000000 08:01 285450
55ae02c20000-55ae02c21000 r--p 00000000 08:01 285450
55ae02c21000-55ae02c22000 rw-p 00001000 08:01 285450
55ae02f26000-55ae02f47000 rw-p 00000000 00:00 0
                                                                                                                                                         /home/nazaretrogue/hello_c_mod
                                                                                                                                                         /home/nazaretrogue/hello_c_mod
/home/nazaretrogue/hello_c_mod
                                                                                                                                                        [leap]
/lib/x86_64-linux-gnu/libc-2.27.so
/lib/x86_64-linux-gnu/libc-2.27.so
/lib/x86_64-linux-gnu/libc-2.27.so
/lib/x86_64-linux-gnu/libc-2.27.so
 ?ff892c7f000-7ff892e66000 r-xp 00000000 08:01 399362
?ff892e66000-7ff893066000 ---p 001e7000 08:01 399362
  ff893066000-7ff89306a000 r--p 001e7000 08:01 399362
 ff89306a000-7ff89306c000 rw-p 001eb000 08:01 399362
 rff893070000-7ff893097000 r-xp 00000000 08:01 399334
rff893280000-7ff893282000 rw-p 00000000 00:00 0
rff893297000-7ff893298000 r--p 00027000 08:01 399334
                                                                                                                                                        /lib/x86 64-linux-gnu/ld-2.27.so
                                                                                                                                                        /lib/x86_64-linux-gnu/ld-2.27.so
/lib/x86_64-linux-gnu/ld-2.27.so
 /ff893298000-7ff893299000 rw-p 00028000 08:01 399334
/ff893299000-7ff89329a000 rw-p 00000000 00:00 0
 fff5ce97000-7fff5ceb8000 rw-p 00000000 00:00 0
fff5cf47000-7fff5cf4a000 r--p 00000000 00:00 0
fff5cf4a000-7fff5cf4b000 r-xp 00000000 00:00 0
                                                                                                                                                         [vvar]
[vdso]
   ffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0
                                                                                                                                                         [vsyscall]
  azaretrogue@nazaretrogue:~$
```

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:

```
nazaretrogue@nazaretrogue:~$ readelf -S hello_c_mod
There are 29 section headers, starting at offset 0x1958:
Encabezados de Sección:
                         Tipo
  [Nr] Nombre
                                          Dirección
                                                           Despl
       Tamaño
                         TamEnt
                                          Opts
                                                 Enl
                                                       Info Alin
   0]
                         NULL
                                          00000000000000000
                                                            00000000
       0000000000000000 00000000000000000
```

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.

```
#include <stdio.h>
int main(){
    printf("Hello world!\n");
    return 0;
}
```

Figure 1: hello.c

```
#include <iostream>
using namespace std;
int main(){
    cout << "Hello world! << endl;
    return 0;
}</pre>
```

Figure 2: hello.cpp

```
#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