SEGURIDAD EN SISTEMAS OPERATIVOS

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

Curso 2017-18

Práctica [2]. Ingeniería inversa y vulnerabilidades

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

Autor[1]: Rubén Calvo Villazán

Ejercicio 1.

a)

La sección .interp es un string ASCII que contiene el nombre del loader/linkeador dinámico. Usualmente ld.so

.got contiene punteros a todas las variables globales que son usadas dentro del programa provenientes de una librería compartida.

.plt viene de Procedure Linkage Table que es la encargada de las llamadas a procedimientos y funciones cuyas direcciones no son conocidas en tiempo de enlazado y se ha dejado para resolver por el enlazador dinámico en tiempo de ejecución.

b)

Hay diferencias en las cabeceras relacionadas con direcciones etc, pero no en las que tienen información sobre la versión gnu, arquitectura etc.

A la hora de buscar .ctors y .dtors vemos que a partir de gcc 4.7, no los genera sino que en su lugar usa .init_array

https://stackoverflow.com/questions/16569495/cant-find-dtors-and-ctors-in-binary

Por lo tanto, usamos

```
$> objdump -dr -j .init_array c/cpp
```

y vemos que

---

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 ..."

con C:

Disassembly of section .init_array:

0000000000200de8 <__frame_dummy_init_array_entry>:

200de8: 30 06 00 00 00 00 00 00 0.....

Y con C++:

Disassembly of section .init_array:

0000000000200da0 <__frame_dummy_init_array_entry>:

200da0: d0 08 00 00 00 00 00 00 56 09 00 00 00 00 00 00 ....V.......

c)

Las secciones de reubicación para el archivo C contienen:

Relocation section '.rela.dyn' at offset 0x410 contains 8 entries:

Offset	Info	Type	Sym. Value	Sym. Name + Addend
000000200de8	000000000008	R_X86_64_RELATIVE		630
000000200df0	000000000008	R_X86_64_RELATIVE		5f0
000000201028	000000000008	R_X86_64_RELATIVE		201028
000000200fd8	000100000006	R_X86_64_GLOB_DAT	0000000000000000	_ITM_deregisterTMClone + 0
000000200fe0	000300000006	R_X86_64_GLOB_DAT	0000000000000000	__libc_start_main@GLIBC_2.2.5 + 0
000000200fe8	000400000006	R_X86_64_GLOB_DAT	0000000000000000	__gmon_start__ + 0
000000200ff0	000500000006	R_X86_64_GLOB_DAT	0000000000000000	_ITM_registerTMCloneTa + 0
000000200ff8	000600000006	R_X86_64_GLOB_DAT	0000000000000000	__cxa_finalize@GLIBC_2.2.5 + 0

Relocation section '.rela.plt' at offset 0x4d0 contains 1 entries:

Offset	Info	Type	Sym. Value	Sym. Name + Addend
000000201018	000200000007	R_X86_64_JUMP_SLO	0000000000000000	puts@GLIBC_2.2.5 + 0

Para el de C++ contienen:

Relocation section '.rela.dyn' at offset 0x5d0 contains 12 entries:

Offset	Info	Type	Sym. Value	Sym. Name + Addend
000000200da0	000000000008	R_X86_64_RELATIVE		8d0
000000200da8	000000000008	R_X86_64_RELATIVE		956
000000200db0	000000000008	R_X86_64_RELATIVE		890
000000201040	000000000008	R_X86_64_RELATIVE		201040
000000200fc8	000100000006	R_X86_64_GLOB_DAT	0000000000000000	__gmon_start__ + 0
000000200fd0	000300000006	R_X86_64_GLOB_DAT	0000000000000000	__libc_start_main@GLIBC_2.2.5 + 0
000000200fd8	000500000006	R_X86_64_GLOB_DAT	0000000000000000	_ZNSt8ios_base4InitD1E@GLIBCXX_3.4 + 0
000000200fe0	000600000006	R_X86_64_GLOB_DAT	0000000000000000	_ITM_deregisterTMClone + 0
000000200fe8	000800000006	R_X86_64_GLOB_DAT	0000000000000000	_ITM_registerTMCloneTa + 0
000000200ff0	000900000006	R_X86_64_GLOB_DAT	0000000000000000	__cxa_finalize@GLIBC_2.2.5 + 0
000000200ff8	000b00000006	R_X86_64_GLOB_DAT	0000000000000000	_ZSt4endlIcSt11char_tr@GLIBCXX_3.4 + 0
000000201060	000c00000005	R_X86_64_COPY	0000000000201060	_ZSt4cout@GLIBCXX_3.4 + 0

Relocation section '.rela.plt' at offset 0x6f0 contains 4 entries:

Offset	Info	Type	Sym. Value	Sym. Name + Addend
000000201018	000200000007	R_X86_64_JUMP_SLO	0000000000000000	_ZNSt8ios_base4InitC1E@GLIBCXX_3.4 + 0
000000201020	000400000007	R_X86_64_JUMP_SLO	0000000000000000	__cxa_atexit@GLIBC_2.2.5 + 0
000000201028	000700000007	R_X86_64_JUMP_SLO	0000000000000000	_ZStlsISt11char_traits@GLIBCXX_3.4 + 0
000000201030	000a00000007	R_X86_64_JUMP_SLO	0000000000000000	_ZNSolsEPFRSoS_E@GLIBCXX_3.4 + 0

Ejercicio 2.

a) Tanto objdump como readelf tienen diversas opciones. Por ejemplo con

$> readelf -a C

$> objdump -x C

Podemos ver todas las cabeceras del archivo.

Hay más opciones en las que coinciden pero en otras cambian en la sintaxis.

El motivo por el que existe readelf es que objdump depende de BFD (Binary File Descriptor Library) y además depende de la arquitectura, luego se hace necesario de un programa que examine el archivo sin consultar la BFD y además que sea independiente de la arquitectura.

b)

En relación a la ingeniería inversa, objdump nos permite por ejemplo ver el código desensamblado del programa.

Por ejemplo con

$> objdump -D C

Podemos desensamblar por completo el programa y ver las direcciones de memoria.

Ejercicio 3.

Si hacemos

$> cat /proc/2048/maps

Vemos que nos aparece:

```
5563b2e10000-5563b2e11000 r-xp 00000000 08:01 7868273                    /root/cpp
5563b3010000-5563b3011000 r--p 00000000 08:01 7868273                    /root/cpp
5563b3011000-5563b3012000 rw-p 00001000 08:01 7868273                    /root/cpp
5563b3ba3000-5563b3bd5000 rw-p 00000000 00:00 0                          [heap]
7f2ebf5ff000-7f2ebf792000 r-xp 00000000 08:01 45484715                   /lib/x86_64-linux-gnu/libc-2.24.so
7f2ebf792000-7f2ebf992000 ---p 00193000 08:01 45484715                   /lib/x86_64-linux-gnu/libc-2.24.so
7f2ebf992000-7f2ebf996000 r--p 00193000 08:01 45484715                   /lib/x86_64-linux-gnu/libc-2.24.so
7f2ebf996000-7f2ebf998000 rw-p 00197000 08:01 45484715                   /lib/x86_64-linux-gnu/libc-2.24.so
7f2ebf998000-7f2ebf99c000 rw-p 00000000 00:00 0
7f2ebf99c000-7f2ebf9b2000 r-xp 00000000 08:01 45484571                   /lib/x86_64-linux-gnu/libgcc_s.so.1
7f2ebf9b2000-7f2ebfbb1000 ---p 00016000 08:01 45484571                   /lib/x86_64-linux-gnu/libgcc_s.so.1
7f2ebfbb1000-7f2ebfbb2000 r--p 00015000 08:01 45484571                   /lib/x86_64-linux-gnu/libgcc_s.so.1
7f2ebfbb2000-7f2ebfbb3000 rw-p 00016000 08:01 45484571                   /lib/x86_64-linux-gnu/libgcc_s.so.1
```

7f2ebfbb3000-7f2ebfcb6000 r-xp 00000000 08:01 45484782 /lib/x86_64-linux-gnu/libm-2.24.so

7f2ebfcb6000-7f2ebfeb5000 ---p 00103000 08:01 45484782 /lib/x86_64-linux-gnu/libm-2.24.so

7f2ebfeb5000-7f2ebfeb6000 r--p 00102000 08:01 45484782 /lib/x86_64-linux-gnu/libm-2.24.so

7f2ebfeb6000-7f2ebfeb7000 rw-p 00103000 08:01 45484782 /lib/x86_64-linux-gnu/libm-2.24.so

7f2ebfeb7000-7f2ec0027000 r-xp 00000000 08:01 2756183 /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.24

7f2ec0027000-7f2ec0227000 ---p 00170000 08:01 2756183 /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.24

7f2ec0227000-7f2ec0231000 r--p 00170000 08:01 2756183 /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.24

7f2ec0231000-7f2ec0233000 rw-p 0017a000 08:01 2756183 /usr/lib/x86_64-linux-gnu/libstdc+
+.so.6.0.24

7f2ec0233000-7f2ec0236000 rw-p 00000000 00:00 0

7f2ec0236000-7f2ec0259000 r-xp 00000000 08:01 45484685 /lib/x86_64-linux-gnu/ld-2.24.so

7f2ec0426000-7f2ec042a000 rw-p 00000000 00:00 0

7f2ec0456000-7f2ec0459000 rw-p 00000000 00:00 0

7f2ec0459000-7f2ec045a000 r--p 00023000 08:01 45484685 /lib/x86_64-linux-gnu/ld-2.24.so

7f2ec045a000-7f2ec045b000 rw-p 00024000 08:01 45484685 /lib/x86_64-linux-gnu/ld-2.24.so

7f2ec045b000-7f2ec045c000 rw-p 00000000 00:00 0

7ffd2f191000-7ffd2f1b2000 rw-p 00000000 00:00 0 [stack]

7ffd2f1d6000-7ffd2f1d9000 r--p 00000000 00:00 0 [vvar]

7ffd2f1d9000-7ffd2f1db000 r-xp 00000000 00:00 0 [vdso]

Encontramos que las regiones del heap tienen permisos de r, rw, o rx, todos en zona privada.

Si hacemos

$> objdump -D C++

vemos que en el main nos aparece:

e8 90 fe ff ff callq 7e0 <sleep@plt>

Que es la llamada al sleep dentro del main.