Universidad de San Andrés
I301 - Arquitectura de computadoras y Sistemas Operativos

Profesor: Daniel Veiga

Ayudantes: Matias Lavista, Nicolas Romero

Entregado: **Lunes 25 de Marzo**

Entrega: **Martes 9 de Abril 11:59 PM**

Entrega tarde (máxima nota 8): **Miércoles 10 de Abril: 11:59 PM**

Práctico 2

Este práctico está basado en un práctico de Carnegie Mellon University, creado por los profesores R. Bryant and D. O'Hallaron.

0 - Antes de empezar:

1. El Link de su bomba esta en el spreadsheet:

   Bombas y Links

   Hay una única bomba por alumno, los otros links no les sirven.

2. Bajen toda su carpeta que gdrive lo va a hacer un zip file. Y luego hagan los siguientes pasos:

```
# Su file va a ser algo como bomb44-20240326T180253Z-001.zip
# pueden moverlo a donde quieran aca lo pude en home
$mv ~/Downloads/bomb44-20240326T180253Z-001.zip ~/
$cd ~/  #puede ser el directorio que quieran, pero coincida con el mv de arriba
$unzip bomb44-20240326T180253Z-001.zip
$cd bomb44 #Lo que sea su bomba
$sudo chmod +x bomb #Esto hace que la bomba sea ejecutable
#ya esta lista
```

3. Cuando bajen las bombas, deben tener todos los siguientes archivos en su directorio, la bomba los usa a todos:
    a. ID : identificador único por alumno.
    b. bomb: código objeto compilado.
    c. palabras.txt: Diccionario de palabras.
    d. bomb.c: El archivo main, que ejecuta cada una de las fases de la bomba.
    e. README: contiene su nombre
    f. .gdbinit: miren el punto 3 para ponerlo en el lugar correcto
    g. gdb_refcard_gnu.pdf
    La forma rápida de mover su bomba a la VM es con un repositorio git.

4. Hay un archivo llamado "**.gdbinit**" en su carpeta, el punto al comienzo lo hace un oculto en linux, para verlo pueden hacer:

```
    $ls -alh   # la a es all
```

Muevan este archivo a su home, para tener todas las configuraciones correctas de **gdb**, con el siguiente comando:

```
    $mv $dir_con_el_archivo/.gdbinit ~/
```

5. En el caso de que deseen ver la próxima instrucción de assembly a ejecutarse, con gdb corriendo pueden usar los siguientes flags:

```
$gdb bomb
(gdb) set disassemble-next-line on
(gdb) show disassemble-next-line
```

1 - Un poco de contexto:

En un mundo donde la tecnología domina cada aspecto de nuestra vida, un grupo de estudiantes de informática se enfrenta a un reto sin precedentes: han sido elegidos para participar en una misión crítica para desactivar código malicioso denominado ***bombas binarias,*** estas han sido diseminadas en la red por un grupo de hackers. Si estas bombas no son desactivadas, van a explotar vulnerabilidades de los sistemas permitiendo robar datos de usuario. Su misión es liberar las claves que desactivan la bomba.

Sin el código fuente original, no tenemos mucho por dónde empezar, pero hemos observado que los programas parecen operar en una secuencia de niveles o fases. ***Hay 4 fases en total***. Cada nivel desafía al usuario a ***ingresar una cadena de texto, numérica o ambas al mismo tiempo. La bomba también permite que se le provea un archivo de texto con las claves correspondientes***. Si el usuario ingresa la cadena correcta, desactiva el nivel, el código malicioso es evitado, y el programa continúa. Pero si se ingresa la entrada

incorrecta, la bomba explota y termina el programa. Para desactivar toda la bomba, uno necesita desactivar con éxito cada uno de sus niveles.

A cada uno se le asigna una **única** bomba para desactivar y su misión es aplicar tus mejores habilidades de detective de assembly para descifrar la entrada requerida para pasar cada nivel y desactivar la bomba.

La bomba (**bomb**) es un ejecutable, es decir, código objeto ya compilado. A partir de este código ya compilado, trabajarás hacia atrás para intentar reconstruir una imagen del código fuente C original. En un proceso conocido como ***ingeniería inversa***. Una vez que entiendas qué hace que tu bomba ***funcione***, puedes suministrar a cada nivel la entrada que requiere e ir desactivandola. Los niveles se vuelven progresivamente más complejos, pero la experiencia que ganas a medida que avanzas desde cada nivel debería compensar esta dificultad. Un punto importante es que la bomba tiene un disparador muy sensible, propenso a explotar ante la menor provocación. Cada vez que tu bomba explota, notifica al personal, lo que resta puntos al TP. Por lo tanto, **hay consecuencias al detonar la bomba**.

La ***ingeniería inversa*** requiere una mezcla de enfoques y técnicas diferentes y te brindará la oportunidad de practicar con una variedad de herramientas que están listadas, con una breve explicación, en la sección 4. El arma más poderosa en tu arsenal será el ***debugger***, y un objetivo es ampliar tu destreza con ***gdb***. ¡Desarrollar un repertorio robusto de ***gdb*** puede traerte grandes beneficios durante el resto de tu carrera!

2 - Cómo empezar?

1. Lean **TODO** el tp.

2. Cuidado con simplemente ejecutar la bomba ya que probablemente explote. Háganlo primero usando gdb y escriban los breakpoints correspondientes para evitar que explote:

```
$gdb bomb
```

3. **bomb** puede recibir parámetros de la siguiente forma:

```
$gdb --args bomb <arg1> <arg2> ...
```

4. Impriman el assembly de la bomba así pueden hacer un mapa mental:

```
$objdump -M intel -d bomb
```

5. Les recomendamos ***fuertemente*** redirigir la salida hacia un archivo (por ejemplo, assembly_my_bomb.txt), abrirlo con un editor de texto e intenten hacer una especie de mapa del código. Es decir, intentar entender qué hace cada una de las secciones

del código y alguna especie de esquema de las fases de la bomba, les va a ayudar bastante:

```
$objdump -M intel -d bomb > assembly_my_bomb.txt
```

6. ***¡Vean las clases de gdb, pasajes de parámetros y pila! No obstante, les proveemos algunos machetes con las instrucciones de gbd, y algunos otros softwares que les pueden resultar útiles (sección 4). Igualmente, insistimos vean y estudien bien las clases.***

3- Las reglas del juego:

1. ¿Cómo entregar? Deben subir **TODOS** los archivos que les dimos a una carpeta dentro de su carpeta de entrega.

2. **input.txt:** Archivo de inputs contiene los strings que se usan para resolver cada fase de la bomba. Tendríamos que poder correr:

   $./bomb < input.txt

   y desactivar la bomba para que el TP esté aprobado.

3. **respuestas_descripcion.txt:** Acá deben mencionar: **Su nombre + su email.** Luego para cada etapa que desactivaron explicar qué hacía el código, y cómo lo resolvieron. No tiene que escribir una novela, simplemente algunas oraciones describiendo el nivel y su solución.

4 - Herramientas útiles:

Aquí hay algunos posibles puntos de ataque para tu gran aventura de ingeniería inversa:

nm: volcará la tabla de símbolos de un ejecutable. Los símbolos incluyen los nombres de funciones y variables globales y sus direcciones. La tabla de símbolos por sí sola no es mucho por donde empezar, pero simplemente leer los nombres podría darte una ligera idea del terreno.

strings: mostrará todas las cadenas imprimibles en un ejecutable, incluidas todas las constantes de cadena. ¿Qué cadenas encuentras en tu bomba? ¿Alguna de ellas parece relevante para la tarea en cuestión?

objdump: puede volcar el código objeto en su equivalente desensamblado. Leer y rastrear el código desensamblado es de donde vendrá la mayor parte de tu información. Escudriñar el código objeto sin vida sin ejecutarlo es una técnica conocida como listado muerto. Una vez que averigües qué hace el código objeto, puedes, en efecto, traducirlo de vuelta a C y

luego ver qué entrada se espera. Esto funciona bastante bien en pasajes de código simples, pero puede volverse complicado cuando el código es más complejo.

gcc: Si no estás seguro de cómo se traduce una construcción de C particular a ensamblador o cómo acceder a un cierto tipo de datos, otra técnica es intentar comenzar desde el otro lado. Escribe un pequeño programa en C con el código, compila y luego rastrea su desensamblado, ya sea listado en un archivo con objdump o en gdb. Por ejemplo, si no estás seguro de cómo funciona una declaración break o cómo se invoca un puntero a función por qsort, esta sería una buena manera de averiguarlo. Dado que tú mismo escribiste el programa de prueba, tampoco tienes que temer su naturaleza explosiva.

El sitio web interactivo GCC Explorer. Menos pesado que iniciar gcc por ti mismo es el práctico gcc-en-un-sitio-web que adelantamos en el laboratorio 6. Escribe un fragmento de código y obtén su traducción inmediata al ensamblador, ¡fácil! La herramienta está haciendo la misma traducción que podrías hacer a través de gcc, pero de una manera conveniente que fomenta la exploración interactiva.

5 - Algunos tips que pueden llegar a ser útiles:

Disassembly de la sección .text: (la sección de text es la de código)

objdump -d -M intel bomb

Si se desea encontrar una cadena de texto:

strings bomb

Ver la tabla de símbolos del ejecutable. (**No confundir con el comando string)**

nm bomb

Recuerden que siempre lo mejor es usar el manual, por ejemplo:

man nm

6 - Debugger

GDB Quick Reference

Comandos útiles:

r	**run** Ejecuta el programa hasta el primer break
b	**break FILE:LINE** Breakpoint en la línea
b	**break FUNCTION** Breakpoint en la función
info breakpoints	Muestra información sobre los breakpoints
c	**continue** Continua con la ejecución

s		**step** Siguiente linea (Into)
n	|	**next** Siguiente linea (Over)
si	|	**stepi** Siguiente instrucción asm (Into)
ni	|	**nexti** Siguiente instrucción asm (Over)

x/Nuf ADDR Muestra los datos en memoria

N = Cantidad (bytes)

u = Unidad b|h|w|g

b:byte, h:word, w:dword, g:qword

f = Formato x|d|u|o|f|a

x:hex, d:decimal, u:decimal sin signo, o:octal, f:float,
a:direcciones, s: strings, i:inst.

Ejemplos:

- x/3bx **addr** : Tres bytes en hexadecimal.
- x/5wd **addr** : Cinco enteros de 32 bits con signo.
- x/i : $rip : imprimir próxima instrucción
- x/s **addr** : Una string terminada en cero.

Configuración de GDB:

~ /.gdbinit

Para usar sintaxis intel y guardar historial de comandos (el archivo de **.gdbinit** vino con el práctico):

set disassembly-flavor intel
set history save

Correr GDB con argumentos:

gdb --args <ejecutable> <arg1> <arg2> ...

MISC:

- ¿Cómo reconozco un binario?

file bomb

En general, arroja lo siguiente:

ELF 64-bit LSB executable, x86-64, version 1 (SYSV), statically linked, with debug_info, not stripped

ELF, siglas de ***"Executable and Linkable Format"***, es un formato de archivo utilizado para ejecutables, bibliotecas compartidas y objetos en sistemas operativos tipo Unix y Unix-like, como Linux y BSD. ELF proporciona una estructura estándar para organizar y ejecutar programas, incluyendo información sobre el tipo de archivo, segmentación de memoria, símbolos, y otras características necesarias para su ejecución.

En particular, este archivo es un programa ejecutable diseñado para sistemas Unix/Linux de 64 bits. Está creado para la arquitectura x86-64 y sigue el estándar SYSV. Está enlazado estáticamente, lo que significa que todas las bibliotecas necesarias están incluidas dentro del archivo. Además, contiene información de depuración para ayudar en el diagnóstico de errores durante el desarrollo. Es ***Not stripped***, lo que significa que conserva todos los detalles adicionales como símbolos de depuración. En resumen, es un ejecutable completo que puede ejecutarse en sistemas unix compatibles, con capacidad de depuración integrada y sin haber sido optimizado para reducir su tamaño.