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

Profesor: Daniel Veiga

Ayudantes: Matias Lavista, Nicolas Romero, Juan Carlos Suárez

Entregado: **Lunes 9 de Septiembre**

Entrega: **Martes 17 de Septiembre 11:59 PM**

Entrega tarde (máxima nota 8):

Práctico 2

Este trabajo práctico se basa en un ejercicio desarrollado en la Universidad Carnegie Mellon por los profesores R. Bryant y D. O'Hallaron.

0 - Antes de empezar:

1. El Link de su bomba está en el spreadsheet:
   Bombas y Links
   Hay una única bomba por alumno, los otros links no les sirven.

2. Descarguen toda su carpeta, ya que Google Drive la comprimirá en un archivo ZIP. Luego, sigan los pasos indicados a continuación::

```
# 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. Al descargar las bombas, asegúrense de que todos los siguientes archivos estén en su directorio, ya que la bomba los utiliza 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. En su carpeta hay un archivo llamado `.gdbinit`. El punto al inicio lo convierte en un archivo oculto en Linux. Para verlo, pueden ejecutar el siguiente comando:

```
$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 desafío sin precedentes: han sido seleccionados para participar en una misión crítica. Su objetivo es desactivar un código malicioso conocido como ***'bombas binarias***', diseminado en la red por un grupo de hackers. Si no logran desactivarlas a tiempo, estas bombas explotarán vulnerabilidades en los sistemas, permitiendo el robo masivo de datos de usuarios. La misión es liberar las claves que desactivan las bombas antes de que sea demasiado tarde.

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 participante se le asigna una bomba única para desactivar. Su misión es aplicar sus mejores habilidades de análisis en ensamblador para descifrar la entrada requerida que les permitirá superar cada nivel y desactivar la bomba por completo..

La bomba (**bomb**) es un ejecutable, es decir, código objeto ya compilado. A partir de este código, se deberá trabajar hacia atrás en un proceso conocido como **ingeniería inversa**, para intentar reconstruir una imagen del código fuente original en C. Una vez que se comprenda cómo funciona la bomba asignada, podrá proporcionar la entrada correcta en cada nivel para desactivarla paso a paso.

Los niveles se vuelven progresivamente más complejos, pero la experiencia adquirida al avanzar en cada uno debería compensar esta dificultad. Un aspecto importante a tener en cuenta es que la bomba tiene un disparador extremadamente sensible, que puede explotar ante la menor provocación. Cada vez que la bomba explota, se notifica al personal, lo que resulta en una reducción de puntos en el trabajo práctico. Por lo tanto, **detonar la bomba tiene consecuencias negativas**.

La **ingeniería inversa** exige una combinación de enfoques y técnicas diversas, y ofrece la oportunidad de practicar con una variedad de herramientas, que se listan y explican brevemente en la sección 4. La herramienta más poderosa en este proceso será el **debugger**, y uno de los objetivos es mejorar la destreza con **gdb**. Desarrollar un sólido dominio de **gdb** puede proporcionar grandes beneficios a lo largo de toda la carrera profesional.

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, la corrección es automática cualquier archivo que les demos y que NO sea entregado el TP se encuentra automáticamente desaprobado**. Si están trabajando con git les recomendamos hacer:

   $git add .

   para asegurar que todos los archivos de su repo se están entregando.

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.