Instituto de Industria Universidad Nacional de General Sarmiento Licenciatura en Sistemas

SISTEMAS OPERATIVOS Y REDES II
Primer Semestre 2025

Detección de vulnerabilidades de desbordamiento

1.Introducción

En esta práctica, aprenderemos a utilizar herramientas de *fuzzing* para detectar posibles vulnerabilidades, específicamente aquellas relacionadas con desbordamientos de memoria. El *fuzzing* es una técnica que consiste en proporcionar entradas aleatorias o malformadas a un programa con el fin de descubrir errores o fallos de seguridad que podrían ser explotados.

2.Objetivos

Los objetivos principales de esta práctica son:

1. Conocer y utilizar herramientas de *fuzzing*.
2. Aprender a identificar vulnerabilidades, como los desbordamientos de memoria, a través de la ejecución de pruebas automatizadas.

3. Desarrollo

3.1. Radamsa

Una herramienta básica pero efectiva para realizar *fuzzing* es *Radamsa*. Esta herramienta toma un texto de entrada, aplica una serie de transformaciones aleatorias sobre él y luego devuelve el resultado, lo que permite simular diferentes variaciones en los parámetros de entrada de un programa que se desea probar.

Para ver cómo funciona, puedes ejecutar el siguiente ejemplo en tu terminal:

```
|  echo hola | radamsa
```

Prueba a ejecutarlo varias veces y observa cómo varían los resultados generados. *Radamsa* ofrece varias opciones de configuración que permiten modificar su comportamiento. Te sugiero revisar su manual con el comando man radamsa para obtener más información sobre estas opciones.

3.2. Prueba de un programa sencillo

El programa que se adjunta a esta práctica permite insertar una cadena dentro de otra en una posición dada. El programa solicita dos cadenas y un número entero a través de la entrada estándar.

Primero, compila y ejecuta el programa con diferentes combinaciones de cadenas para asegurarte de que funciona correctamente. Luego, crea un archivo de texto (input.txt) con una entrada típica, y usa el siguiente comando para redirigir el contenido del archivo hacia la entrada estándar del programa:

```
| cat input.txt | ./insert
```

A continuación, puedes intercalar *Radamsa* para alterar el contenido de input.txt antes de pasarlo al programa:

```
| cat input.txt | radamsa | ./insert
```

Al ejecutar este comando repetidamente, puedes encontrar errores en el programa insert. Sin embargo, dado que puede ser necesario ejecutar el comando muchas veces para identificar un fallo, es recomendable automatizar este proceso con un script en *bash*.

El siguiente script ejecuta el programa en bucle hasta que se encuentra un error grave (por ejemplo, un *segmentation fault*). Cuando esto ocurre, guarda la última entrada que causó el error en un archivo llamado last_input.txt. De esta manera, puedes usar esa entrada específica para depurar el programa y descubrir la causa del fallo:

```
| #!/bin/bash
| while true; do
|         cat input.txt | radamsa | tee last_input.txt | ./insert
|         test $? -gt 127 && break
| done
```

Con este script, trata de encontrar todos los fallos presentes en el programa insert.c y corrígelos. Asegúrate de que las soluciones a los errores provocados por entradas incorrectas del usuario generen mensajes de error claros, indicando al usuario el problema específico que ha cometido, para que la interacción con el programa sea más amigable y robusta.

Entregables

- Informe técnico en PDF con:
- Diagrama de red y configuración.
- Capturas de pantalla de cada paso.
- Resultados de pruebas.
- Análisis sobre ventajas y desventajas
- (Opcional) Grabación de video corto mostrando la demo de autenticación