SISTEMAS OPERATIVOS Y REDES II

Trabajo Práctico 2
Detección de vulnerabilidades de desbordamiento

Profesores: Benjamin Chuquimango
Pablo Rodriguez

Grupo 2

Integrantes: Juan Farias
Pablo Igei Nakagawa
Ezequiel Ravignani
Carlos Caballero
Cristian Yoel Garay

Primer Semestre 2025

Índice

¿Qué es el fuzzing?

El fuzzing es una técnica de pruebas de software que se utiliza para detectar errores o vulnerabilidades de seguridad en aplicaciones. A diferencia de otros métodos más estructurados de análisis, el fuzzing se basa en enviar grandes cantidades de datos aleatorios, inesperados o maliciosos al programa, con el objetivo de hacer que falle o se comporte de forma extraña.

Cuando una aplicación falla durante un proceso, esto indica la existencia de errores en el código, posibles vulnerabilidades o problemas de lógica que podrían ser aprovechados por un atacante. Estos errores en muchos casos no son fáciles de detectar con pruebas adicionales.

El fuzzing tiene la posibilidad de garantizar un estándar de calidad uniforme mediante pruebas separadas, por otra parte refuerza la estabilidad del software, mejora la seguridad.

Una de las herramientas de fuzzing es Radamsa. Se trata de un fuzzer (herramienta) de propósito general que contiene una gran cantidad de algoritmos de mutación para poder generar datos de entrada de forma aleatoria basados en números, cadenas o patrones. Además es posible enviar los datos al sistema analizando los paquetes de TCP/IP.

Instalación de Radamsa

Comenzaremos actualizando el índice de paquetes paquetes locales de modo que se refleje cualquier cambio:

```
alumno@alumno-virtualbox:~/radamsa$ sudo apt update
```

Luego, instalar los paquetes gcc, git, make y wget necesarios para compilar el código fuente en un binario ejecutable:

```
alumno@alumno-virtualbox:~/radamsa$ sudo apt install gcc git make wget
```

Después de confirmar la instalación, se descargará e instalará los paquetes específicos y todas sus dependencias necesarias, en la VM de ubuntu ya estaban instalados.

A continuación se descargara una copia del código fuente de Radamsa clonandolo del repositorio alojado en https://gitlab.com/akihe/radamsa.git

```
alumno@alumno-virtualbox:~$ git clone https://gitlab.com/akihe/radamsa.git
```

Al realizarlo, se creará un directorio llamado radamsa que contendra el código fuente de la aplicación. A continuación nos dirigimos al directorio para compilar el código usando *make:*

```
alumno@alumno-virtualbox:~$ cd radamsa
alumno@alumno-virtualbox:~/radamsa$ make
mkdir -p lib
cd lib && git clone https://gitlab.com/owl-lisp/hex.git
Clonando en 'hex'...
remote: Enumerating objects: 14, done.
remote: Total 14 (delta 0), reused 0 (delta 0), pack-reused 14 (from 1)
Desempaquetando objetos: 100% (14/14), 2.18 KiB | 203.00 KiB/s, listo.
test -x bin/ol || make bin/ol
make[1]: se entra en el directorio '/home/alumno/radamsa'
test -f ol.c.gz || wget -O ol.c.gz https://haltp.org/files/ol-0.2.2.c.gz || c
url -L -o ol.c.gz https://haltp.org/files/ol-0.2.2.c.gz
--2025-04-16 10:51:13--  https://haltp.org/files/ol-0.2.2.c.gz
Resolviendo haltp.org (haltp.org)... 95.216.5.207
Conectando con haltp.org (haltp.org)[95.216.5.207]:443... conectado.
Petición HTTP enviada, esperando respuesta... 200 OK
Longitud: 341773 (334K) [application/octet-stream]
Guardando como: “ol.c.gz”

ol.c.gz             100%[================>] 333,76K   340KB/s    en 1,0s

2025-04-16 10:51:16 (340 KB/s) - “ol.c.gz” guardado [341773/341773]

gzip -d < ol.c.gz > ol.c
mkdir -p bin
cc -Wall -O3 -o bin/ol ol.c
make[1]: se sale del directorio '/home/alumno/radamsa'
bin/ol -O1 -o radamsa.c rad/main.scm
mkdir -p bin
cc -Wall -O3  -o bin/radamsa radamsa.c
```

Por último instalaremos el binario de Radamsa compilado en su $PATH

```
alumno@alumno-virtualbox:~/radamsa$ sudo make install
mkdir -p /usr/bin
cp bin/radamsa /usr/bin
mkdir -p /usr/share/man/man1
cat doc/radamsa.1 | gzip -9 > /usr/share/man/man1/radamsa.1.gz
```

Una vez completado podemos verificar la versión instalada:

```
alumno@alumno-virtualbox:~/radamsa$ radamsa  --version
Radamsa 0.8a
```

Funcionamiento de Radamsa

Para ver como funciona Radamsa, en la terminal ponemos el comando *echo* y entre comillas un texto como parámetro, al que se le aplica la mutación, de tal forma que *echo* muestre el resultado de la misma. De esta forma se manipulan los datos ingresados antes de hacer los casos de prueba.

```
alumno@alumno-virtualbox:~$ echo "hola" | radamsa
hol
```

En este caso Radamsa modificó el último carácter de la cadena reemplazándolo con un símbolo. En algunas aplicaciones este cambio puede generar que los datos se interpreten de manera errónea. A continuación se muestran los resultados de ejecutar este comando varias veces secuencialmente:

```
alumno@alumno-virtualbox:~/Desktop/SOR2/SOR2-TP2$ echo "hola" | radamsa
ho`xcalc`%#x"xcalc\n$'aaaa%d%n\r\nla
alumno@alumno-virtualbox:~/Desktop/SOR2/SOR2-TP2$ echo "hola" | radamsa
h◆◆◆g◆◆◆◆l◆alumno@alumno-virtualbox:~/Desktop/SOR2/SOR2-TP2$ echo "hola" | radamsa
hola
hola
alumno@alumno-virtualbox:~/Desktop/SOR2/SOR2-TP2$ echo "hola" | radamsa
hol␣a
alumno@alumno-virtualbox:~/Desktop/SOR2/SOR2-TP2$ echo "hola" | radamsa
GGG\\la
alumno@alumno-virtualbox:~/Desktop/SOR2/SOR2-TP2$
```

```
alumno@alumno-virtualbox:~$ echo "hola" | radamsa
hola␉
```

```
alumno@alumno-virtualbox:~$ echo "hola" | radamsa
iola
```

```
alumno@alumno-virtualbox:~$ echo "hola" | radamsa
hl␉ollola
```

```
alumno@alumno-virtualbox:~$ echo "hola" | radamsa
hooool␦␦
```

```
alumno@alumno-virtualbox:~$ echo "hola" | radamsa
hol-a—
```

Configuración de radamsa

-n más un valor entero X se generan X salidas por cada entrada.

```
alumno@alumno-virtualbox:~$ echo "hola" | radamsa -n 10
```

-o guarda la salida en un archivo.

```
alumno@alumno-virtualbox:~/Descargas$ echo "hola" | radamsa -o PruebaRadamsa.txt
```

-r lee recursivamente si se dan carpetas como entrada.

```
alumno@alumno-virtualbox:~$ echo "hola" | radamsa -r
```

-s modifica la semilla para la aleatoriedad.

```
alumno@alumno-virtualbox:~$ echo "hola" | radamsa -s 120
```

-p con un valor entero X aplica mutaciones con X% de probabilidad

```
alumno@alumno-virtualbox:~$ echo "hola" | radamsa -p 20
```

Creación de script para encontrar errores

1) Primero creamos el archivo con extensión .sh, ya que se utilizará el intérprete bash para ejecutar el script.

```
alumno@alumno-virtualbox:~/Descargas/TP2_SOR2$ touch fuzzing.sh
alumno@alumno-virtualbox:~/Descargas/TP2_SOR2$ ls
 fuzzing.sh   insert     insert.c.zip      __MACOSX
 input.txt    insert.c   last_input.txt  'Trabajo Práctico 2 - vulnerabilidades de desbordamiento.pdf'
alumno@alumno-virtualbox:~/Descargas/TP2_SOR2$
```

2) Luego lo editamos con un editor de texto (en este caso utilizamos VIM) y guardamos el siguiente script:
   #!/bin/bash
   while true; do
   cat input.txt | radamsa | tee last_input.txt | ./insert
   test $? -gt 127 && break
   done

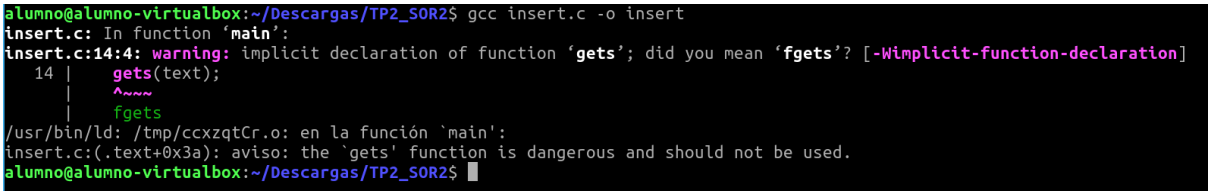
```
alumno@alumno-virtualbox:~/Descargas/TP2_SOR2$ vim fuzzing.sh
alumno@alumno-virtualbox:~/Descargas/TP2_SOR2$ cat fuzzing.sh
#!/bin/bash
while true; do
cat input.txt | radamsa | tee last_input.txt | ./insert
test $? -gt 127 && break
done

alumno@alumno-virtualbox:~/Descargas/TP2_SOR2$
```

3) Hacer el archivo ejecutable: por defecto, los archivos .sh no tienen permisos de ejecución, así que hay que darle permiso con el siguiente comando: chmod +x fuzzing.sh. Y después ya podremos ejecutar el script de fuzzing.

```
alumno@alumno-virtualbox:~/Descargas/TP2_SOR2$ ./fuzzing.sh
bash: ./fuzzing.sh: Permiso denegado
alumno@alumno-virtualbox:~/Descargas/TP2_SOR2$ chmod +x fuzzing.sh
alumno@alumno-virtualbox:~/Descargas/TP2_SOR2$ ./fuzzing.sh
Enter some text
Enter the string to insert
Enter the position to insert
insert
Enter some text
```

Correcciones de errores encontrados en el programa insert.c

Gets



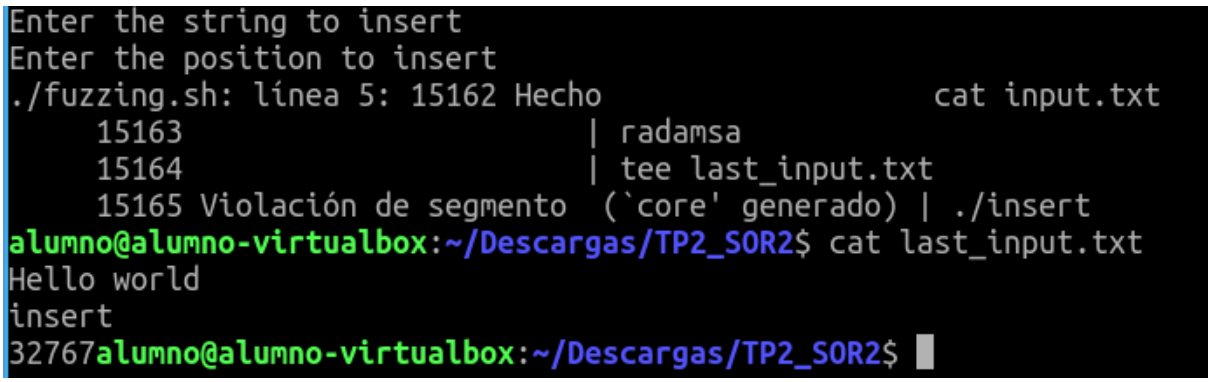
Como se ve en la imagen, el compilador gcc no deja compilar el programa debido al gets. El compilador sugiere utilizar fgets lo cual esto es correcto ya que el fgets() es más seguro porque limita la cantidad de caracteres leídos, evitando que se llene el buffer. En cambio el gets() es más peligroso porque no verifica el tamaño del buffer, lo que puede causar buffer overflow (un error grave de seguridad).

Solución

En el código reemplazamos gets(text); por fgets(substring, sizeof(substring), stdin); y reemplazamos gets(substring); por fgets(substring, sizeof(substring), stdin);
Quedaría así:

```
printf("Enter some text\n");
fgets(text, sizeof(text), stdin);
text[strcspn(text, "\n")] = '\0'; // Elimina el salto de línea si existe

printf("Enter the string to insert\n");
fgets(substring, sizeof(substring), stdin);
substring[strcspn(substring, "\n")] = '\0'; // Elimina el salto de línea si existe
```

Violación de segmento



Este error se genera debido a que se intenta acceder a memoria fuera de los límites.

Solución

Agregamos una validación de la posición para insertar, que esté entre 1 y el tamaño del texto sino printea un mensaje de error.

```
printf("Enter the position to insert\n");
scanf("%d", &position);

// Validación de la posición
if (position < 1 || position > strlen(text)) {
    printf("Error: Posición inválida. Debe estar entre 1 y %ld.\n", strlen(text));
    return 1;  // Termina el programa con error
}
```

Violación de segmento II

También puede ocurrir que al recibir un número negativo que exceda las capacidades de los enteros este se trate de forma incorrecta por el programa. Con Radamsa encontramos que esto ocurre por ejemplo con el valor -11023226954 que genera una violacion de segmento, pero siendo este un caso diferente al anterior.

```
alumno@alumno-virtualbox:~/Desktop/SOR2/SOR2-TP2$ ./insert
Enter some text
Hello, !
Enter the string to insert
world
Enter the position to insert
-11023226954
Violación de segmento (`core' generado)
alumno@alumno-virtualbox:~/Desktop/SOR2/SOR2-TP2$
```

Solución

Ahora el valor de la posición se recibe como una cadena y luego se convierte a un entero para detectar si lo que se recibió en primer lugar era un entero válido. Si no lo es, el programa termina con un mensaje de error apropiado.

```
// Validar si se ingresó un número entero válido
long pos_long = strtol(position_str, &endptr, 10);
if (pos_long < INT_MIN || pos_long > INT_MAX) {
    printf("Error: El número entero está fuera del rango permitido.\n");
    return 1;
}

if (*endptr != '\n') {
    printf("Error: Lo ingresado no es un número entero válido.\n");
    return 1;
}
position = (int)pos_long;
```

Se consideró por separado la validación de si es un entero dentro de los límites que soporta el programa, de la validación de si para empezar se trataba de un entero.

Ventajas y Desventajas

Ventajas:

- **Encuentra errores de forma automática:**
  Con el fuzzing es posible crear datos modificados o inesperados para ver si el programa se rompe. No es necesario saber cómo está hecho por dentro, solo observar y si algo falla, es un posible bug.
- **Muy bueno para encontrar fallos de seguridad:**
  Se puede detectar errores que permiten ejecutar códigos maliciosos, acceso indebido a las memorias.
- **Es automático:**
  Una vez que el fuzzer se está ejecutando trabaja por sí solo. Puede estar horas generando y enviando datos sin que se tenga que intervenir.
- **Detecta errores que no estaban contemplados:**
  El fuzzing crea combinaciones que uno normalmente no probaría.

Desventajas:

- **No llega a probar todo:**
  Por más que se creen millones de datos, esto no significa que cubra todas las posibles situaciones del programa. Hay caminos del código que nunca se exploran si no se guían bien los datos.
- **Falsos positivos:**
  Puede pasar que el programa se rompa, pero no porque sea una falla seria. Causa que se termine revisando muchos casos que quizás no hacían falta.
- **Dificultad de configuración:**
  Si el sistema que queremos probar usa datos complejos, hay que modificar el fuzzer para armar entradas que tengan más posibilidad de datos útiles.
- **Consumo de recursos:**
  Si el fuzzer va a ser ejecutado por muchas horas o si se tiene que probar algo pesado, se necesita una buena máquina, además que los resultados puede ocupar bastante espacio.

Video DEMO

Se creó un video como demostración de lo realizado en este trabajo práctico. Puede acceder al mismo desde el siguiente enlace.

Repositorio

Link: https://github.com/Ging1991/SOR2-TP2