



## **Sistemas Operativos y Redes II - 2<sup>do</sup> semestre 2024**

### **TP2 Seguridad Informática:**

### **Vulnerabilidades de desbordamiento**



#### **Docente**

- Benjamin Chuquimango

#### **Integrantes**

- Chamizo, Christian
- Espinola, Dario
- Oviedo, Marcelo
- Zavalla Gamarra, Martín Octavio

#### **Fecha de entrega**

- 18/10/2024

# Índice

<b>Introducción</b>	<b>3</b>
<b>Radamsa</b>	<b>4</b>
¿Qué es Radamsa?	4
Instalación	4
Pruebas iniciales	6
<b>Análisis - Programa insert</b>	<b>7</b>
Test manual	7
Tests automatizados y errores encontrados	12
Segmentation Fault	13
Stack Smashing	14
<b>Conclusión</b>	<b>16</b>
<b>Referencias</b>	<b>17</b>

## Introducción

La seguridad informática es un aspecto fundamental en el desarrollo de software, ya que cualquier error puede ser usado para comprometer la integridad, confidencialidad o disponibilidad de los sistemas. Entre las amenazas más conocidas se encuentra el buffer overflow<sup>1</sup>, qué ocurre cuando se escribe más información en un buffer del que puede manejar, lo que puede permitir a un atacante ejecutar código arbitrario o causar fallos en el sistema.

En este proyecto se tiene como objetivo conocer y utilizar herramientas de *fuzzing*<sup>2</sup> para detectar posibles vulnerabilidades a través de la ejecución de pruebas automatizadas. Para esto contamos con un programa *insert.c* que debemos analizar y testear para encontrar todos los fallos posibles.

---

<sup>1</sup> En informática, un buffer es una región de la memoria que se utiliza para almacenar datos temporalmente.

<sup>2</sup> El fuzzing es una técnica que consiste en proporcionar entradas aleatorias a un programa con el fin de descubrir fallos de seguridad.

# Radamsa

## ¿Qué es Radamsa?

Es un *fuzzer* de propósito general, fácil de utilizar y que contiene 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 analizado en paquetes TCP/IP.

## Instalación

El entorno que utilizaremos para las pruebas será una distribución linux. Por lo tanto, por medio de la terminal, obtendremos Radamsa de su repositorio en Git usando los siguientes comandos:

```
git clone https://gitlab.com/akihe/radamsa.git
```

```
cd radamsa
```

```
make
```

```
sudo make install
```

```
radamsa -h
```

```
alumno@alumno-virtualbox:~/radamsa$ radamsa --help
Usage: radamsa [arguments] [file ...]
-h | --help, show this thing
-a | --about, what is this thing?
-V | --version, show program version
-o | --output <arg>, output pattern, e.g. out.bin /tmp/fuzz-%n.%s, -, :80 o
r 127.0.0.1:80 or 127.0.0.1:123/udp [-]
-n | --count <arg>, how many outputs to generate (number or inf) [1]
-s | --seed <arg>, random seed (number, default random)
-m | --mutations <arg>, which mutations to use [ft=2,fo=2,fn,num=5,ld,lds,l
r2,li,ls,lp,lr,lis,lrs,sr,sd,bd,bf,bi,br,bp,bei,bed,ber,uw,ui=2,xp=9,ab]
-p | --patterns <arg>, which mutation patterns to use [od,nd=2,bu]
-g | --generators <arg>, which data generators to use [random,file=1000,jum
p=200,stdin=100000]
-M | --meta <arg>, save metadata about generated files to this file
-r | --recursive, include files in subdirectories
-S | --seek <arg>, start from given testcase
-T | --truncate <arg>, take only first n bytes of each output (mainly inten
ded for UDP)
-d | --delay <arg>, sleep for n milliseconds between outputs
-l | --list, list mutations, patterns and generators
-C | --checksums <arg>, maximum number of checksums in uniqueness filter (0
disables) [10000]
```

Opciones de comandos en Radamsa

```

dario@lenovo:~/Documents/radamsa$ radamsa --help
Usage: radamsa [arguments] [file ...]
-h | --help, show this thing
-a | --about, what is this thing?
-V | --version, show program version
-o | --output <arg>, output pattern, e.g. out.bin /tmp/fuzz-%n.%s, -, :80 or 127.0.0.1:80 or 127.0.0.1:123/udp [-]
-n | --count <arg>, how many outputs to generate (number or inf) [1]
-s | --seed <arg>, random seed (number, default random)
-m | --mutations <arg>, which mutations to use [ft=2,fo=2,fn,num=5,ld,lrs,lr2,li,ls,lp,lr,lis,lrs,sr,sd,bd,bf,bi,br,bp,bei,bed,ber,uw,ui=2,xp=9,ab]
-p | --patterns <arg>, which mutation patterns to use [od,nd=2,bu]
-g | --generators <arg>, which data generators to use [random,file=1000,jump=200,stdin=100000]
-M | --meta <arg>, save metadata about generated files to this file
-r | --recursive, include files in subdirectories
-S | --seek <arg>, start from given testcase
-T | --truncate <arg>, take only first n bytes of each output (mainly intended for UDP)
-d | --delay <arg>, sleep for n milliseconds between outputs
-l | --list, list mutations, patterns and generators
-C | --checksums <arg>, maximum number of checksums in uniqueness filter (0 disables) [10000]
-H | --hash <arg>, hash algorithm for uniqueness checks (stream, sha1 or sha256) [stream]
    --output-template <arg>, Output template. %f is fuzzed data. e.g. "<html>%f</html>"
-v | --verbose, show progress during generation

```

```

radamsa(1)                                General Commands Manual                                radamsa(1)

NAME
    radamsa - a general purpose fuzzer

SYNOPSIS
    radamsa [options] [sample-path] ...

DESCRIPTION
    Radamsa is a general purpose data fuzzer. It reads data from given sample files, or standard input if none are given, and outputs modified data. It is usually used to generate malformed data for testing programs.

OPTIONS
    This program follows the usual GNU command line syntax, with long options starting with two dashes ('-'). A summary of options is included below.

    -h, --help
        Show a shorter summary of options.

    -V, --version
        Show program version.

    -o, --output pattern
        Specify where to write the modified data. By default the data is written to standard output. The pattern can be a path, in which %n is filled to be the test case number if present, :n to have the data sent to TCP clients connecting to local port n, or a.b.c.d:n to have radamsa connect to port n of IPv4 address a.b.c.d to send each output.

    -n, --count n
        How many outputs to generate based on the sample(s). Giving -1 or inf causes data to be generated forever. The default is 1.

    -s, --seed n
        Start from random state n, which is a decimal number. Starting from the same random state with the same command line arguments and samples causes the same outputs to be generated. If no seed is given explicitly radamsa reads a random one from /dev/urandom or takes the current time in milliseconds.

    -M, --meta path
        Write metadata about generated data to given path. - can be used to have the metadata written to standard output. The metadata contains the seed followed by one line per testcase.

    -l, --list
        Show all available mutations, patterns and generators along with their descriptions.

    -r, --recursive
        Include samples from directories recursively.

    -v, --verbose
        Print what is being done.

    -m, --mutations string
        Request the initial mutation probabilities manually. This is generally not recommended and the options will change in the future versions.

    -p, --patterns string
        Request the mutation patterns manually. This is generally not recommended and the options will change in the future versions.

    -g, --generators string
        Request the data stream generator probabilities manually. This is generally not recommended and the options will change in the future versions.

EXAMPLES
    $ cat file | radamsa | program -
    $ radamsa samples/*.foo | program -
    $ radamsa -o test-%n.foo -n 100 samples/*.foo
    $ radamsa -o :80 -n inf samples/*.resp
    $ radamsa -o 127.0.0.1:80 -n inf samples/*.req

```

## Pruebas iniciales

Para garantizar la correcta instalación, se realizaron pruebas de la herramienta. En este caso, pasamos a Radamsa por parámetro una cadena de texto y realizamos esta operación repetidas veces esperando que cada vez nos proporcione un resultado diferente.

```
martin@DESKTOP-1DD67VP:~/CIBERSEC$ echo "hola" | radamsa
//v8/
ho0la
martin@DESKTOP-1DD67VP:~/CIBERSEC$ echo "hola" | radamsa
$1&#000;\u0000hhlhhhllahlolala
martin@DESKTOP-1DD67VP:~/CIBERSEC$ echo "hola" | radamsa
0hole
martin@DESKTOP-1DD67VP:~/CIBERSEC$ echo "hola" | radamsa
0a
martin@DESKTOP-1DD67VP:~/CIBERSEC$ echo "hola" | radamsa
Khola
Khola
Khola
Khola
Khola
Khola
Khola
```

```
marce998@marce-VM:~/Documentos$ echo hola | radamsa
hola
hola
ho0la
marce998@marce-VM:~/Documentos$ echo hola | radamsa
hola
hola
marce998@marce-VM:~/Documentos$ echo hola | radamsa
'holaaaaaamarce998@marce-VM:~/Documentos$ echo hola | radamsa
holb
marce998@marce-VM:~/Documentos$ echo hola | radamsa
hol0a
marce998@marce-VM:~/Documentos$ echo hola | radamsa
0hola
marce998@marce-VM:~/Documentos$
```

```
dario@lenovo:~/Documents/radamsa$ echo "hola"| radamsa
000hola
dario@lenovo:~/Documents/radamsa$ echo "hola"| radamsa
000hola
dario@lenovo:~/Documents/radamsa$ echo "hola"| radamsa
ho la
dario@lenovo:~/Documents/radamsa$ echo "hola"| radamsa
hola
hola
dario@lenovo:~/Documents/radamsa$ echo "hola"| radamsa
la
```

```
alumno@alumno-virtualbox: ~/radamsa
alumno@alumno-virtualbox:~/radamsa$ echo "hola" | radamsa
Ho
alumno@alumno-virtualbox:~/radamsa$ echo "hola" | radamsa
hol
alumno@alumno-virtualbox:~/radamsa$ echo "hola" | radamsa
hola
hlaool
ah
haool
alumno@alumno-virtualbox:~/radamsa$ echo "hola" | radamsa
ho
alumno@alumno-virtualbox:~/radamsa$ echo "hola" | radamsa
lla
alumno@alumno-virtualbox:~/radamsa$ echo "hola" | radamsa
haol
alumno@alumno-virtualbox:~/radamsa$ echo "hola" | radamsa
ho
alumno@alumno-virtualbox:~/radamsa$ echo "hola" | radamsa
Dhola
alumno@alumno-virtualbox:~/radamsa$
```

## Análisis - Programa insert

### Test manual

Una vez corroborado que Radamsa funciona, debemos hacer lo mismo con el programa "insert.c". Para esto, lo compilamos usando el comando "gcc -o insert insert.c" y lo ejecutamos con "./insert" para conocer su funcionamiento.

Cabe resaltar que, aunque el programa se termina compilando, el sistema ya nos está advirtiendo y aconsejando que utilizar la función **gets()** es peligrosa, y que en su lugar, se debería utilizar **fgets()**. Esto se debe a que, **gets()** no tiene en cuenta el tamaño del buffer de memoria cuando se escriben o leen caracteres en él. En cambio, **fgets()** controla los datos que se ingresan en el buffer, permitiendo especificar el tamaño que tendrá el mismo.

```
martin@DESKTOP-1DD67VP:~/CIBERSEC$ gcc -o insert insert.c
insert.c: In function 'main':
insert.c:14:4: warning: implicit declaration of function 'gets'; did you mean 'fgets'? [-Wimplicit-function-declaration]
   14 |     gets(text);
      |     ^~~~~
      |     fgets
/usr/bin/ld: /tmp/ccj7plex.o: in function `main':
insert.c:(.text+0x3d): warning: the `gets' function is dangerous and should not be used.
```

### Advertencia al compilar "insert.c"

En la siguiente imagen se muestra que al ejecutar el programa, se le solicita al usuario ingresar primero una y luego otra cadena de texto. Luego, solicita una posición (en número). Por último, muestra cómo output, un texto nuevo con el segundo texto ubicado en la posición del carácter del primer texto. Este proceso se realizó un par de veces siguiendo el happy path<sup>3</sup>.

```
martin@DESKTOP-1DD67VP:~/CIBERSEC$ ./insert
Enter some text
martin
Enter the string to insert
zavalla
Enter the position to insert
7
martinzavalla
martin@DESKTOP-1DD67VP:~/CIBERSEC$ ./insert
Enter some text
blablabla
Enter the string to insert
sisi
Enter the position to insert
3
blsisiablabla
martin@DESKTOP-1DD67VP:~/CIBERSEC$ ./insert
Enter some text
oro
Enter the string to insert
pepita de
Enter the position to insert
1
pepita de oro
```

```
dario@lenovo:~/Documents$ ./insert
Enter some text
Dario Espinola
Enter the string to insert
Fabian
Enter the position to insert
6
DarioFabian Espinola
dario@lenovo:~/Documents$ ./insert
Enter some text
Esto una prueba
Enter the string to insert
es
Enter the position to insert
6
Esto es una prueba
```

<sup>3</sup> Escenario de uso ideal de un software, en el que nada fuera de lo normal ocurre y todo se desarrolla de forma directa hasta lograr el objetivo deseado por el usuario



```

alumno@alumno-virtualbox:~/Descargas$ ./insert
Enter some text
Chris Chamizo
Enter the string to insert
Chamizo
Enter the position to insert
3
ChChamizoris Chamizo
alumno@alumno-virtualbox:~/Descargas$ ./insert
Enter some text
vacaciones, sol, playa
Enter the string to insert
sol
Enter the position to insert
3
vasolcaciones, sol, playa
alumno@alumno-virtualbox:~/Descargas$ █

```

programa insert en ejecución

En este punto, tenemos el programa y Radamsa funcionando. Para la siguiente prueba, crearemos un archivo "input.txt" el cual contendrá los datos que solicitaba insert.c, cada uno en una línea diferente. El objetivo de este archivo será facilitar la automatización de testing del programa.

Como vemos a continuación, podemos agarrar el contenido de input.txt y pasarlo al programa para que lo utilice. Además, concatenando tareas, modificamos el contenido del archivo con Radamsa y lo pasamos al programa. De esta forma logramos a partir de un input aleatorizar los inputs del programa.

```

martin@DESKTOP-1DD67VP:~/CIBERSEC$ touch input.txt
martin@DESKTOP-1DD67VP:~/CIBERSEC$ nano input.txt
martin@DESKTOP-1DD67VP:~/CIBERSEC$ cat input.txt
taza
kilometraje
3
martin@DESKTOP-1DD67VP:~/CIBERSEC$ cat input.txt | ./insert
Enter some text
Enter the string to insert
Enter the position to insert
takilometrajeza
martin@DESKTOP-1DD67VP:~/CIBERSEC$ cat input.txt | radamsa | ./insert
Enter some text
Enter the string to insert
Enter the position to insert
ta{a
martin@DESKTOP-1DD67VP:~/CIBERSEC$ cat input.txt | radamsa | ./insert
Enter some text
Enter the string to insert
Enter the position to insert
taza
martin@DESKTOP-1DD67VP:~/CIBERSEC$ cat input.txt | radamsa | ./insert
Enter some text
Enter the string to insert
Enter the position to insert
tkkilometrajeaza

```

```
dario@lenovo:~/Documents$ cat input.txt | ./insert
Enter some text
Enter the string to insert
Enter the position to insert
Ejemplo de entrada
dario@lenovo:~/Documents$ cat input.txt | radamsa | ./insert
Enter some text
Enter the string to insert
Enter the position to insert
Segmentation fault (core dumped)
dario@lenovo:~/Documents$ cat input.txt | radamsa | ./insert
Enter some text
Enter the string to insert
Enter the position to insert
dario@lenovo:~/Documents$ cat input.txt | radamsa | ./insert
Enter some text
Enter the string to insert
Enter the position to insert
dario@lenovo:~/Documents$ cat input.txt | radamsa | ./insert
Enter some text
Enter the string to insert
Enter the position to insert
Segmentation fault (core dumped)
dario@lenovo:~/Documents$ cat input.txt | radamsa | ./insert
Enter some text
Enter the string to insert
Enter the position to insert
de entrada
```

programa insert con inputs aleatorios

```
alumno@alumno-virtualbox:~/Descargas$ cat input.txt
remo
playa
arena
alumno@alumno-virtualbox:~/Descargas$ cat input.txt | radamsa | ./insert
Enter some text
Enter the string to insert
Enter the position to insert
playam
alumno@alumno-virtualbox:~/Descargas$ cat input.txt | radamsa | ./insert
Enter some text
Enter the string to insert
Enter the position to insert
playa
```

```
pm4ya
alumno@alumno-virtualbox:~/Descargas$ cat input.txt | radamsa | ./insert
Enter some text
Enter the string to insert
Enter the position to insert
remo
```

```
Enter the position to insert
pm4ya
alumno@alumno-virtualbox:~/Descargas$ cat input.txt | radamsa | ./insert
Enter some text
Enter the string to insert
Enter the position to insert
a
alumno@alumno-virtualbox:~/Descargas$ cat input.txt | radamsa | ./insert
Enter some text
Enter the string to insert
Enter the position to insert
remo
alumno@alumno-virtualbox:~/Descargas$ cat input.txt | radamsa | ./insert
Enter some text
Enter the string to insert
Enter the position to insert
playa
alumno@alumno-virtualbox:~/Descargas$ cat input.txt | radamsa | ./insert
Enter some text
Enter the string to insert
Enter the position to insert
arena
```

```
alumno@alumno-virtualbox:~/Descargas$ cat input.txt | radamsa | ./insert
Enter some text
Enter the string to insert
Enter the position to insert
remo `rena
alumno@alumno-virtualbox:~/Descargas$ cat input.txt | radamsa | ./insert
Enter some text
Enter the string to insert
Enter the position to insert
re`mo
alumno@alumno-virtualbox:~/Descargas$ cat input.txt | radamsa | ./insert
Enter some text
Enter the string to insert
Enter the position to insert
arena
alumno@alumno-virtualbox:~/Descargas$ cat input.txt | radamsa | ./insert
Enter some text
Enter the string to insert
Enter the position to insert
playa
alumno@alumno-virtualbox:~/Descargas$
```

```
alumno@alumno-virtualbox:~/Descargas$ cat input.txt | radamsa | ./insert
Enter some text
Enter the string to insert
Enter the position to insert
a000pl
*** stack smashing detected ***: terminated
Abortado ('core' generado)
```

## Tests automatizados y errores encontrados

Durante estas últimas pruebas, hubo casos donde el programa arrojaba errores. Sin embargo al no tener control sobre cual es el input aleatorio generado por Radamsa, no sabíamos cuál era la que generaba el error (obviamente se puede inferir cuáles podrían ser al revisar el código o aplicar las sugerencia del compilador). Por este motivo, buscamos testear de forma automática el programa y en caso de que ocurra un error poder ir a solucionarlo.

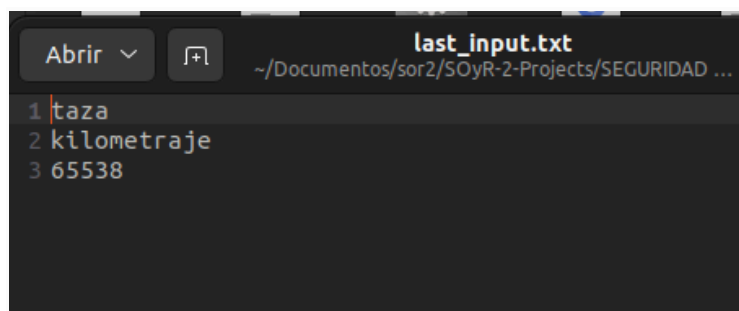
En primer lugar creamos un script en bash llamado “findError.sh” que ejecute el programa insert con los inputs aleatorios hasta que ocurra un error. Una vez que sucede, hay que escribir en un archivo “last\_input.txt” el input aleatorio usado y que se muestre por pantalla el error del programa.

A continuación utilizamos el output del script para solucionar errores:

```
dario@lenovo:~/Documents/sor2_projects/SEGURIDAD INFORMATICA$ sh findError.sh
Enter some text
Enter the string to insert
Enter the position to insert
Ejempm ed entradado
Enter some text
Enter the string to insert
Enter the position to insert
Ejemplo de entrbda
Enter some text
Enter the string to insert
Enter the position to insert
Segmentation fault (core dumped)
dario@lenovo:~/Documents/sor2_projects/SEGURIDAD INFORMATICA$ ls
findError.sh  input.txt  insert  insert.c  last_input.txt  'vulnerabilidades de desbordamiento.pdf'
dario@lenovo:~/Documents/sor2_projects/SEGURIDAD INFORMATICA$ cat last_input.txt
de entrada
Ejemplo
65526
dario@lenovo:~/Documents/sor2_projects/SEGURIDAD INFORMATICA$
```

```
marce998@marce-VM:~/Documentos/sor2/SOyR-2-Projects/SEGURIDAD INFORMATICA$ sh findError.sh
Enter some text
Enter the string to insert
Enter the position to insert
ki
lometraje
Enter some text
Enter the string to insert
Enter the position to insert
tak:ilometrajeza
Enter some text
Enter the string to insert
Enter the position to insert
Segmentation fault (core dumped)
marce998@marce-VM:~/Documentos/sor2/SOyR-2-Projects/SEGURIDAD INFORMATICA$
```

findError.sh encuentra segmentation fault

A screenshot of a text editor window titled 'last\_input.txt'. The window shows a list of three items: '1 |taza', '2 |kilometraje', and '3 |65538'. The first item is highlighted with a mouse cursor. The window's title bar includes a dropdown menu labeled 'Abrir' and a plus icon. The path shown in the title bar is '~/Documentos/sor2/SOyR-2-Projects/SEGURIDAD ...'.

Input aleatorio que generó el segmentation fault

## Segmentation Fault

Podemos observar que el script termina mostrando un mensaje de segmentation fault. Este es un error que la CPU produce cuando alguien intenta acceder a una parte de la memoria que no debería estar accediendo.

Teniendo en cuenta los valores del archivo “last\_input.txt”, el programa recibe un índice mayor al tamaño del primer texto ingresado, intentando agregar contenido en una posición que no es posible de obtener.

Para solucionar este error, se decide modificar el código de insert.c, agregando una condición sobre que el valor válido para una posición sea un número entre el 1 y el tamaño del primer texto ingresado.

Luego de agregar este cambio, se vuelve a ejecutar el script y obtenemos los siguientes resultados:



findError.sh se ejecutaba infinitamente al no encontrar ninguno de los errores que se presentaban antes sobre *segmentation fault* y *stack smashing*.

```
> findError.sh ●
> findError.sh
1  #!/bin/bash
2  count=0
3  while true; do
4      cat input.txt | radamsa | tee last_input.txt | ./insert
5      test $? -gt 127 && break
6      count=$((count + 1)) # Incrementamos el contador
7      if [ "$count" -ge 100 ]; then
8          echo "Se Alcanzó el límite de 100 iteraciones."
9          break
10     fi
11 done
```

Agregado del contador para que se detenga la ejecución del bash en la iteración 100



## Conclusión

Durante el desarrollo del trabajo, pudimos ver más en profundidad de qué se trata el desbordamiento de buffer y qué tan frecuente se puede presentar. Obviamente al utilizar el software de Radamsa, forzamos a que en algún momento se produzca el Buffer Overflow, pero también identificamos que, en otros momentos, en otras cursadas, nos pasaba lo mismo, donde nos encontrábamos con mensajes de *"segmentation fault"* o *"stack smashing"* y en esa situación lo único que nos imaginábamos era que el programa, código no nos andaba bien. Aquí, pudimos adentrarnos en el por qué de esto y a su vez, aplicar técnicas para su mitigación. Aprendimos la importancia de usar la función `fgets()` por sobre `gets()`, cómo alterar datos de archivos con Radamsa, y a su vez, aprender un poco más sobre c.

Con toda la experiencia lograda durante este trabajo, podemos sentirnos más preparados y con la certeza de saber por qué y cuándo podría desbordarse el buffer que vamos sobrescribiendo o leyendo, durante el desarrollo de programas/código en nuestro sistema, y poder aplicar herramientas para disolver su presencia.



## Referencias

[Buffer overflow attacks explained \(coengodegebure.com\)](https://coengodegebure.com)

[Fuzzing y testing en sistemas de control industrial](#)