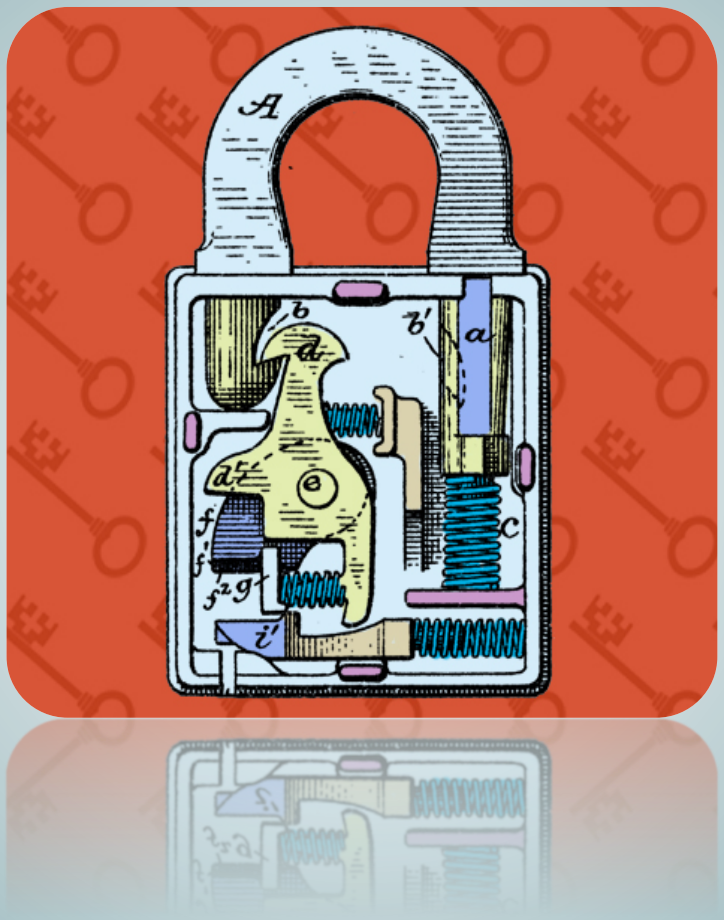
Garantía y Seguridad en Sistemas y Redes

Práctica 6. Detección de Vulnerabilidades de Desbordamiento



Esteban Stafford

Departamento de Ingeniería
Informática y Electrónica

Práctica 6

Detección de vulnerabilidades de desbordamiento

1. Introducción

2. Objetivos

Los objetivos fundamentales que persigue esta práctica son los siguientes:

- Conocer herramientas de *fuzzing*
- Aprender a detectar posibles vulnerabilidades de desbordamiento

3. Desarrollo

3.1. `radamsa`

Una herramienta rudimentaria, aunque efectiva, de *fuzzing* es `radamsa`. Ésta básicamente lee un texto por la entrada estándar, sobre él realiza una serie de transformaciones aleatorias y escribe el resultado en la salida estándar. Esto permite alimentar a un programa que queramos testear con variaciones de los parámetros de entrada. Para ver cómo funciona prueba el ejemplo siguiente:

```
echo hola | radamsa
```

Prueba a ejecutarlo varias veces y observa los diferentes resultados. El comando `radamsa` tiene varias opciones de configuración con las que se puede alterar su funcionamiento. Echa un vistazo a su (escueto) manual `man radamsa`.

3.2. Test de un programa sencillo

El programa adjunto sirve para insertar una cadena dentro de otra en una posición dada. Para ello pide por la entrada estándar dos cadenas y un número entero. Compílalo y ejecutalo con varias cadenas para ver cómo funciona. Una vez hecho esto escribe en un fichero de texto una entrada típica y ejecuta el programa redirigiendo el contenido del fichero a su entrada estándar.

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

Ahora es sencillo intercalar el comando `radamsa` para alterar el contenido de `input.txt` antes de alimentarlo a `insert`.

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

A base de ejecutar el comando anterior repetidas veces, podemos encontrar fallos en `insert`, tratar de corregirlos. Sin embargo, el número de veces que hay que ejecutarlo puede ser muy alto. Por ello conviene hacer uso de la shell para ahorrarnos trabajo. El siguiente script ejecuta tests hasta que sucede un error grave (Seg. fault o similar). Cuando esto sucede, el fichero `last_input.txt` contiene la entrada que causó el error. Lo que nos permite ejecutar el programa en un depurador con la misma entrada 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 del programa `insert.c` y corrígelos. Las soluciones que aborden problemas causados por el usuario deben tratarse de manera que quede claro el error que ha cometido. Es decir, muestra mensajes de error cuando creas que el usuario lo necesite.