Laboratorio 3

Buffer Overflow

Profesores: Alejandro Hevia y Eduardo Riveros
Auxiliares: Sergio Rojas
Ayudantes: Darlene Sobarzo y Tomás Alvarado

- Trabajo personal o en parejas
- Entrega: lunes 9 de junio a las 23:59 hrs.

Cuando un programa tiene una vulnerabilidad de **Buffer Overflow**, una forma común de explotarlo es inyectar código arbitrario (shellcode) en la memoria. Sin embargo, muchos sistemas modernos **previenen la ejecución de código en ciertas áreas de memoria**, como la pila, usando protecciones como **NX (No eXecute)**.

Frente a esto, surge una técnica llamada **ret2libc**, donde en lugar de inyectar instrucciones nuevas, el atacante **redirige la ejecución del programa a funciones que ya existen en memoria**, particularmente, en la **biblioteca estándar de C** (libc). Un ejemplo clásico es redirigir la ejecución a `system("/bin/sh")`, lo que lanza una shell.

Trabajaremos con el siguiente código que encontrarán en material docente:

```c
#include <stdio.h>
#include <string.h>

void vulnerable() {
  char buffer[512];
  printf("Introduce tu mensaje:\n");
  gets(buffer); // No creo que genere problemas...
  printf("Mensaje recibido: %s\n", buffer);
}

void secreto() {
    printf("¡Lograste redirigir la ejecución!\n");

int main() {
  vulnerable();
  return 0;
}
```

Con la siguiente compilación:

```
gcc -fno-stack-protector -z execstack -no-pie -o vuln vuln.c
```

1. Ejercicios

1.1. [1 pt.]

Con gdb, descubre cúantos bytes necesitas para sobreescribir la dirección de retorno.

1.2. [1.5 pt.]

Realizando un ataque tipo Buffer Overflow, redirige la ejecución hacia `secreto()`. No se permite modificar la lógica de `vulnerable()`.

1.3. [2.5 pt.]

Usa gdb para encontrar:

- Dirección de `system` en `libc`.
- Dirección de `exit`.
- Dirección de la cadena `"/bin/sh"` en memoria (`libc`).

Construye un payload que:

- Llama a `system("/bin/sh")`.
- Luego a `exit()` (para terminar limpiamente).

Si lo haces bien, deberías obtener una shell interactiva. Se tomará como válido que el gdb señale que se ejecutó `/bin/sh`.

1.4. [1 pt.]

Señale en el informe cuales fueron las implicancias de realizar la compilación como se señala en este enunciado. En su análisis, responda las siguientes preguntas:

- ¿Qué realiza cada flag de la compilación? Investigue y responda **brevemente**.
- ¿En qué se diferencia con la compilación normal que se realizaría? Investique y responda **brevemente**.

Además, incluya en el informe otras mitigaciones (aparte de la compilación) que se podrían realizar para prevenir la vulnerabilidad.

2. Indicaciones

- Para obtener la dirección de una función del programa, en gdb: `info address <funcion>`.
- Para obtener las direcciones de `system` y `exit`, en gdb:

```
start
p system
p exit
```

- Para obtener la dirección de la cadena `"/bin/sh"`:
  - En gdb, realiza: `info proc mappings`.
  - Podrás ver el inicio de `libc`.
  - Fuera de gdb, realiza: `strings -a -t x /lib/x86_64-linux-gnu/libc.so.6 | grep /bin/sh`
  - Lo anterior te entregará la dirección relativa de `/bin/sh`
  - Con eso, pueden calcular su dirección real como: `dir_libc + dir_rel_binsh`.
- gdb genera nuevos procesos hijos, para ver lo que ocurre en estos ejecutar en gdb: `set follow-fork-mode child`.
- Al momento de generar el payload para el tercer ejercicio, se deberá tener la siguiente estructura de este:

[padding hasta `return address`]

[dirección de `system()`]

[dirección de `exit()`]

[dirección de `"/bin/sh"`]

3. Entregable

Deberán entregar un informe donde expliquen el paso a paso de sus ataques, adjuntando fotos de la terminal donde se señale cómo van obteniendo la información.