Stack Buffer Overflow

Hacking Ético

©Ismael Ripoll &
Hector Marco

Universidad Politècnica de València

February 1, 2022

Índice

1. Objetivos
2. Programa vulnerable
3. El error
4. Análisis
5. Explotación

Objetivos

- Aprender el impacto que tiene el desbordamiento de un array en pila.
- Ser capaces de construir un exploit en python que redirija el flujo de ejecución.
- Esta práctica es muy compleja. Debes comprender perfectamente cada paso antes de pasar al siguiente.
- Todo lo que has estudiado en Fundamentos de Sistemas Operativos, lo vas a necesitar.
- Debes tener soltura con el uso de GDB.

Programa vulnerable

- El siguiente programa es vulnerable:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
char global[100];
void vulnerable(){
  char local[32];
  gets(local);
  strcpy(global, local);
}
int main(void){
  printf("Yo soy:\n");
  system("/usr/bin/id");
  vulnerable();
  return 0;
}
```

El error

- Compílalo con los flags `-m32 -no-pie -fomit-frame-pointer -fno-stack-protector`
- Intenta forzar el error en el programa anterior.
- Históricamente, se utiliza la letra “A” (ascii 0x41) para forzar los errores.
- Identifica exactamente la longitud de la entrada que hace que el programa falle.
- Utiliza Python desde la línea de ordenes para forzar el fallo.
- Utiliza el comando `strace` para observar el fallo.
- Utiliza GDB y sigue el proceso hasta que falle.

¿Qué podemos controlar?

- Ajusta la entrada para que en el contador de programa (eip) tenga el valor “BBBB” (0x42424242) cuando se produzca el fallo.
- Según strace, la dirección que falla será `si_addr=0x42424242`

Explotación (I)

1. Ver si podemos redirigir donde queremos:
   - ¿Cuál es la dirección de la función `main()`?
     Utiliza `objdump -d` para localizarla.
   - Sobrescribe la dirección de retorno con la de la propia funcion `main()`.
   - ¿Qué deberías ver en la ejecución? Fíjate en la salida, si lo has hecho bien deberías ver DOS VECES la ejecución de `main()`.
2. Queremos llamar a la función `system()`:
   - Busca en el programa dónde hay una *llamada a la función system()*.
   - Utiliza esta dirección y salta a ella. ¿Qué pasa?
3. Paso del parámetro a `system()`:
   - ¿Sabes donde (la dirección) está la variable "`global`"?
   - Búsca su dirección con `readelf -aW`.
   - La función `system()` espera un parámetro.

Explotación (II)

- ¿Puedes utilizar la variable `global`?
- Añade a tu exploit un comando (por ejemplo `ps`) para ejecutar. Deberías tener algo como:

Una vez completado el `exploit.py`, debería funcionar de la siguiente forma:

```
./exploit.py | ./vuln
Yo soy:
uid=1000(iripoll) gid=1000(iripoll) grupos=1000(iripoll)
  PID TTY          TIME CMD
 2245 pts/1    00:00:01 bash
 3664 pts/1    00:00:08 evince
 3800 pts/1    00:00:00 vuln
 3803 pts/1    00:00:00 sh
 3804 pts/1    00:00:00 ps
Violación de segmento (`core' generado)
```

Explotación (III)

4. AVANZADO: Ahora trata de ejecutar un shell interactivo.
   - ¿Qué pasa con la entrada estándar del shell?
   - Fíjate que la entrada estándar del programa vulnerable es una tubería (PIPE), por tanto, no puede leer de la consola.
   - ¿Puedes conseguir tener un shell interactivo operativo?
   - Prueba a utilizar los operadores de REDIRECCIÓN del shell. Lee la sección correspondiente del manual (manpage) de bash para conseguirlo.