Respuestas Apéndice 3 . Saludo.s

#1 Valor de edx tras mov longsaludo, %edx

Contiene la longitud de la cadena de texto (0x1c -> 28)

Necesitamos el valor para saber cuantos bytes hay que mostrar a partir del inicio del primer byte de la cadena

#2 Qúe contiene ecx tras mov $saludo, %ecx

0x8049098, la dirección de memoria en la cual comienza el primer caracter de la cadena de texto

#Hacer dibujo

#3 Significado de $

Usando la variable con el dolar, se refiere a la dirección de memoria y se puede examinar

Ej:

ecx 0x8049098 134516888

x/32cb 0x8049098

0x8049098 <saludo>: 72 'H' 111 'o' 108 'l' 97 'a' 32 ' ' 97 'a' 32 ' ' 116 't'
0x80490a0 <saludo+8>: 111 'o' 100 'd' 111 'o' 115 's' 33 '!' 10 '\n' 72 'H' 101 'e'
0x80490a8 <saludo+16>: 108 'l' 108 'l' 111 'o' 44 ',' 32 ' ' 87 'W' 111 'o' 114 'r'
0x80490b0 <saludo+24>: 108 'l' 100 'd' 33 '!' 10 '\n' 28 '\034' 0 '\000' 0 '\000'0 '\000'

Al usar sin el dolar, se refiere al propio valor, en este caso los 4 primeros bytes de la cadena, que es el ancho de palabra del registro (para 32bits), luego el programa no imprimirá$

#4 Cuantas posinciones de memoria ocupa longsaludo

Al ser un entero debe ocupar 4B

x/xw &longsaludo

0x80490b4 <longsaludo>: 0x0000001c

¿y saludo? Ocupa 7 posiciones, 7 bytes

x/7wx &saludo

0x8049098 <saludo>: 0x616c6f48 0x74206120 0x736f646f 0x65480a21

0x80490a8 <saludo+16>: 0x2c6f6c6c 0x726f5720 0x0a21646c

La sección de datos ocupa en total 8 posiciones, 8B

#6 5 posiciones, con objdump -d

8048079: bb 01 00 00 00 mov $0x1,%ebx

Las posiciones en concreto son: 0x8048079 0x804807a 0x80487b 0x804807c 0x804807d

#7 Que sucede si se elimina int 0x80, y si se elimina la siguiente?

Como se elimina la interrupción para hacer la llamada al sistema write,

no imprime la cadena y sí se llama a la función exit.

Si comentamos mov $1, %eax, eax sigue conteniendo un 4, asociado a la llamada al sistema write, con mov $0, %ebx cambiamos el fd de stdout a stdin, y se produce una interrupción, que $

#8, Número de la llamada al sistema read:

#define __NR_read 3, obtenido del fichero /usr/include/asm/unistd_32.h

Respuestas Apéndice 3 . suma.s

#1 Contenido de EAX antes de RET

Contiene el número decimal 37

0b10 es el 2 decimal, 0x10 el 16 y (.-lista)/4 contiene la longitud de$

#2

Se obtiene el valor -3.

1º iteración) eax = 0

2º) eax = 0xffffffff -1 (PF SF IF)

3º) eax = 0xfffffffe -2 (CF AF SF IF)

4º) eax = 0xfffffffd -3 (CF AF SF IF)

Decimal con signo :

x/dw &resultado

0x80490bc <resultado>: -3

Decimal sin signo :

x/uw &resultado

0x80490bc <resultado>: 4294967293

Es negativo porque está habilitado el flag de signo, y además hay acar$

#3

Dirección de suma : 0x8048095

Dirección de bucle: 0x80480a0

Se puede obtener tanto con gdb como con objdump -d:

(gdb) p suma

$7 = {<text variable, no debug info>} 0x8048095 <suma>

objdump -d suma

08048095 <suma>:

Igual para bucle

#4

EIP es el registro de instrucción (Almacena la dirección de la siguien$

ESP contiene un puntero a la cabeza de la pila.

#5

Valor de ESP antes de call: 0xffffd550

despues de llamar a call : 0xffffd54c

antes de llamar a ret : 0xffffd54c

despues de llamar a ret : 0xffffd550

Se diferencian en 4B, que es lo que ocupa una posición de memoria, est$

#6

El registro EIP, conteniendo la instrucción actual y el esp mencionado$

En este caso, eip contiene 0x08048095, dirección de la instrucción que$

#7

La instruccuón ret modifica esp por la razón del ejercicio 5, y de nue$

#8

Cuando se llama a suma, se introduce en la pila la dirección de retorn$

Dentro de la función suma, push %edx inserta el valor actual de edx a $

#9

mov $0x0,%edx ocupa una sola posición de memoria, y contiene 0x0000$

inc %edx ocupa una posición, y el código es 0x42. Examinando con gd$

#10

Que el programa seguría ejecutando instrucciones sin volver a la direc$

Termina de ejecutar pop %edx e incrementa el registro de instrucción, $

(gdb) si

Program received signal SIGSEGV, Segmentation fault.

0x080480a9 in ?? ()

Respuestas para suma64unsigned

#1

Para N=32, el se necesitarían como máximo 2 bits adicionales para almacenar el resultado. Pues 0xffffffff*32)10 = 0x1FFFFFFE0.

Sumando 32 veces el mayor número que se puede representar (0xffffffff) se producen 31 acarreos, el valor de la suma usando los registros ESI:EAX como acumulador es 0x1effffffe1

#2

Cada elemento debe valer $2^{32}/2^5$ = 134217728, en hexadecimal 0x8000000. El acarreo se produce en la última suma, es decir, al sumar el último elemento de la lista, ya que en ese punto el acumulador tiene 0xf8000000, y al sumar 0x8000000 se produce el desbordamiento, dando como resultado 0x100000000

#3

Los acarreos se producen cada vez que se suman 0x10000000 + 0x20000000 + 0x40000000 + 0x80000000, al volver a sumar 0x10000000 a la suma anterior, hay desbordamiento, luego, la suma es la siguiente: 0x20000000 + 0x40000000 + 0x80000000 + 0x10000000, que vuelve a producir desbordamiento.

La suma es 0x780000000.

Respuestas para suma64signed

#1

El mayor número es el 0x7fffffffffffffff (en 64 bits) 0x7fffffff en 32-b. La suma de los 32 elementos da 0x18FFFFFCE

107374182350÷2^10 KiB

107374182350÷2^20 MiB

107374182350÷2^30 GiB

#2

El menor número negativo 0x80000000.

La suma es 0xffffffff000000000

0xffffffff000000000 / 2^A KiB

0xffffffff000000000 / 2^14 MiB

0xffffffff000000000 / 2^1E GiB

#3

$2^{31}/2^5$ = 0x40000000 El

El acarreo no se produce, la suma total es 0x80000000. Para que solo haya un acarreo hay que seguir el mismo razonamiento del ejercicio 2 de suma64unsigned.

#5

El valor necesario es 0xfc000000 para que quepa en 32 bits, para desbordar es necesario 0xf8000000.

#6

El valor total de la suma es 0xfffffffb80000000

Respuestas media.s

#1

El cociente vale -1 si todos los numeros son -1, y si uno de los elementos es 0, el resto vale 0xfffffffe1, es decir, el resultado de la suma. y el cociente vale 0

Programas

Suma64unsigned

```
# suma.s:       Sumar los elementos de una lista
#               llamando a función, pasando argumentos mediante registros
# retorna:      código retorno 0, comprobar suma en %eax mediante gdb/ddd

# SECCIÓN DE DATOS (.data, variables globales inicializadas)
.section .data
lista:
        .int    0x10000000, 0x20000000, 0x40000000, 0x80000000, 0x10000000, 0x20000000, 0x40000000,
0x80000000,0x10000000, 0x20000000, 0x40000000, 0x80000000, 0x10000000, 0x20000000, 0x40000000,
0x80000000, 0x10000000, 0x20000000, 0x40000000, 0x80000000, 0x10000000, 0x20000000, 0x40000000,
0x80000000, 0x10000000, 0x20000000, 0x40000000, 0x80000000, 0x10000000, 0x20000000, 0x40000000,
0x80000000
longlista:
        .int    (.-lista)/4     # .= contador posiciones. Aritmética de etiquetas.
resultado:
        .quad   0               # 4B a FF para notar cuándo se modifica cada byte
#Para el segundo programa, con signo, hay que usar cltd, cdq
# SECCIÓN DE CÓDIGO (.text, instrucciones máquina)
.section .text
_start:.global _start           # PROGRAMA PRINCIPAL-se puede abreviar de esta forma

        mov     $lista, %ebx    # dirección del array lista
        mov  longlista, %ecx    # número de elementos a sumar
        call suma               # llamar suma(&lista, longlista);
        mov  %eax,resultado     # salvar resultado
        mov  %esi,resultado+4
```

```
        # void _exit(int status);
        mov $1, %eax            #   exit: servicio 1 kernel Linux
        mov $0, %ebx   # status: código a retornar (0=OK)
        int $0x80               # llamar _exit(0);


# SUBRUTINA:   suma(int* lista, int longlista);
# entrada:     1) %ebx = dirección inicio array
#              2) %ecx = número de elementos a sumar
# salida:         %eax = resultado de la suma


suma:
        push     %edx           # preservar %edx (se usa aquí como índice)
        mov  $0, %eax           # poner a 0 acumulador
        mov  $0, %edx           # poner a 0 índice
        xor  %esi, %esi                 # poner a 0 esi
        xor %edi, %edi
bucle:
        add (%ebx, %edx,4), %eax
        adc $0, %esi                    # acumular i-ésimo elemento
        inc      %edx                   # incrementar índice
        cmp %edx,%ecx                   # comparar con longitud
        jne bucle                       # si no iguales, seguir acumulando

        pop %edx                        # recuperar %edx antiguo
        ret
```

Suma64signed

```
# suma.s:      Sumar los elementos de una lista
#              llamando a función, pasando argumentos mediante registros
# retorna:     código retorno 0, comprobar suma en %eax mediante gdb/ddd


# SECCIÓN DE DATOS (.data, variables globales inicializadas)
.section .data
lista:
        #.int   0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff,
0xffffffff
        .int    0xf0000000, 0xe0000000, 0xe0000000, 0xd0000000,0xf0000000, 0xe0000000, 0xe0000000,
0xd0000000,0xf0000000, 0xe0000000, 0xe0000000, 0xd0000000,0xf0000000, 0xe0000000, 0xe0000000,
0xd0000000,0xf0000000, 0xe0000000, 0xe0000000, 0xd0000000,0xf0000000, 0xe0000000, 0xe0000000,
0xd0000000,0xf0000000, 0xe0000000, 0xe0000000, 0xd0000000,0xf0000000, 0xe0000000, 0xe0000000,
0xd0000000,0xf0000000, 0xe0000000, 0xe0000000, 0xd0000000,
        #.int 0x0, 0x7fffffff
longlista:
        .int    (.-lista)/4     # .= contador posiciones. Aritmética de etiquetas.
resultado:
        .quad   0               # 4B a FF para notar cuándo se modifica cada byte


# SECCIÓN DE CÓDIGO (.text, instrucciones máquina)
.section .text
_start:.global _start           # PROGRAMA PRINCIPAL-se puede abreviar de esta forma
```

```
        mov     $lista, %ebx  # dirección del array lista
        mov  longlista, %ecx  # número de elementos a sumar
        call suma             # llamar suma(&lista, longlista);
        mov  %eax,resultado   # salvar resultado
        mov  %edx,resultado+4

        # void _exit(int status);
        mov $1, %eax          #   exit: servicio 1 kernel Linux
        mov $0, %ebx   # status: código a retornar (0=OK)
        int $0x80             # llamar _exit(0);

# SUBRUTINA:   suma(int* lista, int longlista);
# entrada:     1) %ebx = dirección inicio array
#              2) %ecx = número de elementos a sumar
# salida:         %eax = resultado de la suma

suma:
        push     %edx         # preservar %edx (se usa aquí como índice)
        #mov  $0xffffffff, %eax  # poner a 0 acumulador
        mov  $0, %esi         # poner a 0 índice
        mov  $0, %ebp
        mov  $0, %edi
bucle:
        mov  (%ebx, %esi, 4), %eax
        cltd
        add %eax, %ebp        #Acumulador EDI:EBP
        adc %edx, %edi        # usado como acum
        inc      %esi                 # incrementar índice
        cmp %esi,%ecx                 # comparar con longitud
        jne bucle                     # si no iguales, seguir acumulando

        mov %edi, %edx        #Colocamos el resultado donde lo espera _start
        mov %ebp, %eax
        pop %edx                      # recuperar %edx antiguo
        ret
```

Media.s

```
# suma.s:      Sumar los elementos de una lista
#              llamando a función, pasando argumentos mediante registros
# retorna:     código retorno 0, comprobar suma en %eax mediante gdb/ddd

# SECCIÓN DE DATOS (.data, variables globales inicializadas)
.section .data
lista:
        #.int   0xffffff, 0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff,
0xffffffff
        .int
0x0,0xffffffff,0xffffffff,0xffffffff,0xffffffff,0xffffffff,0xffffffff,0xffffffff,0xffffffff
```

```
,0xffffffff,0xffffffff,0xffffffff,0xffffffff,0xffffffff,0xffffffff,0xffffffff,0xffffffff,0xffffffff,0x
ffffffff,0xffffffff,0xffffffff,0xffffffff,0xffffffff,0xffffffff,0xffffffff,0xffffffff,0xffffffff,0xfff
fffff,0xffffffff,0xffffffff,0xffffffff
longlista:
        .int    (.-lista)/4     # .= contador posiciones. Aritmética de etiquetas.
cociente:
        .int    0               # 4B a FF para notar cuándo se modifica cada byte
resto:
        .int    0
# SECCIÓN DE CÓDIGO (.text, instrucciones máquina)
.section .text
_start:.global _start           # PROGRAMA PRINCIPAL-se puede abreviar de esta forma

        mov     $lista, %ebx    # dirección del array lista
        mov  longlista, %ecx    # número de elementos a sumar
        call suma               # llamar suma(&lista, longlista);
        mov  %eax,cociente      # salvar resultado
        mov  %edx,resto

        # void _exit(int status);
        mov $1, %eax            #   exit: servicio 1 kernel Linux
        mov $0, %ebx   # status: código a retornar (0=OK)
        int $0x80               # llamar _exit(0);

# SUBRUTINA:   suma(int* lista, int longlista);
# entrada:     1) %ebx = dirección inicio array
#              2) %ecx = número de elementos a sumar
# salida:         %eax = resultado de la suma

suma:
        push     %edx           # preservar %edx (se usa aquí como índice)
        #mov  $0xffffffff, %eax  # poner a 0 acumulador
        mov  $0, %esi           # poner a 0 índice
        mov  $0, %ebp
        mov  $0, %edi
bucle:
        mov  (%ebx, %esi, 4), %eax
        cltd
        add %eax, %ebp          #Acumulador EDI:EBP
        adc %edx, %edi          # usado como acum
        inc      %esi                   # incrementar índice
        cmp %esi,%ecx                   # comparar con longitud
        jne bucle                       # si no iguales, seguir acumulando

        mov %edi, %edx          #Colocamos el resultado donde lo espera _start
        mov %ebp, %eax
        ## División, idiv espera dividendo en EDX:EAX, q en eax, r en edx

        idivl %ecx

        pop %edx                        # recuperar %edx antiguo
```

```
        ret
```