

Ejercicio 1 (4 puntos)

Se le pide realizar un **programa en assembler** (Intel 32 bits) que haga lo siguiente:

- Leer la **humedad ambiente** cada 1 segundo. Use una **system call** para generar el tiempo de espera.
- Ajuste el valor leído de humedad según:  $H_{correcta} = H_{leida}/7$
- Avisar en pantalla si la **humedad** leída en cada momento coincide con algún valor en el arreglo definido en el siguiente fragmento de código

```
sección .data
arreglo dd 10, 20, 30, 40, 50
cant_arreglo dd ($-arreglo)/4
```

(donde esto4 palabra = arreglo) / 4

Otra información que se le brinda

ID	syscall	ebx → tiempo solicitado	ecx → puede ser null
162	sys_nanosleep	const struct timespec *req	struct timespec *Nullable rem
sécción timespec	Cantidad de segundos	Cantidad de nanosegundos	ebx+4
dd	4 bytes	4 bytes	dd

Cómo leer la **humedad**. Linux le permite a usted manejar la entrada analógica donde se lee la **humedad ambiente**, leyendo un archivo. A continuación se detalla cual es el archivo que debe utilizar:

/sys/bus/iio/in\_voltage0\_raw

Usted **no requiere programar la parte de manejo del archivo**; ya se le entrega una función escrita usando convenciones de C para pasaje de argumentos y retorno del valor:

```
RET
BSCD
> filename
int get_humedad(int resolución, char * filename);
// retorna valor de la humedad
// resolución: usar siempre el valor 16 (resolución = 16 bits)
// filename: "/sys/bus/iio/in_voltage0_raw"
```

I.T.B.A

Arquitectura de Computadoras  
Primer Parcial - TEMA 2

2023

Las únicas funciones que puede usar son:

- get\_humedad
- print: funciona exactamente igual que la función de librería estándar de C para intel 32.
- system calls: no hay restricciones

Si requiere alguna otra función, debe implementarla usted.

```
GLOBAL main
GLOBAL belongs_array
EXTERN print
EXTERN get_humedad
section .data
    time_spec dd 1, 0 ; struct
    filename db "/sys/bus/iio/in_voltage0_raw"
    arreglo dd 10, 20, 30, 40, 50 ; arreglo de comparación de valores
    cant_arreglo dd ($-arreglo)/4 ; dimensión
    belongs db "humedad coincide con valor del arreglo", 10, 0
    not_belongs db "humedad NO coincide", 10, 0
```

section .text

main:

```
push ebp
mov ebp, esp ; stackframe
```

```
push edx
push ecx
push ebx
```

sleep:

```
mov eax, 162
mov ebx, time_spec
mov ecx, 0
int 80h
```

humedad\_ambiente:

```
mov eax, 16
push filename
push eax
call get_humedad
add esp, 4*2 ; libero parametros
```

```
; ajuste de valor de humedad  
mov edx, 0 ; condicion de div op  
mov ecx, f ; ajuste  
div ecx ; eax = eax / ecx edx=rest  
; eax contiene valor de comparacion de arreglos  
push eax ; valor  
push count_arreglo ; dim  
push arreglo ; array  
call belongs_array  
add esp, 4 ; "libero" parametro de la función  
  
cmp eax, 0  
je .print_not_belong
```

```
; print belong.  
push belongs  
call print  
add esp, 4  
jmp .sleep
```

```
.print_not_belong:  
push not_belongs  
call print  
add esp, 4  
jmp .sleep.
```

.end:

```
mov eax, 0 ; return cero  
push ebx  
push ecx  
push edx  
  
mov esp, ebp  
pop ebp  
ret.
```

belongs\_array: ; int belongs\_array( int \*array, int dim, int valor)  
push ebp  
mov ebp, esp

push ebx  
push ecx  
push edx  
push esi  
  
mov ebx, [ebp+16] ; valor  
mov ecx, [ebp+8] ; \*array  
mov edx, [ebp+12] ; dim  
mov esi, 0 ; contador / indice  
mov eax, 0 ; si pertenece cambia.

• for:

cmp esi, edx  
je .end  
cmp ebx, duword [ecx + esi \* 4]  
je .belongs  
inc esi  
jmp .for

• belongs:

mov eax, 1

• end:

pop esi  
pop edx  
pop ecx  
pop ebx

mov esp, ebp  
pop ebp  
ret.

Ud. se encuentra analizando un binario en busca de vulnerabilidades en el mismo. Al usar objdump obtuvo la siguiente salida:

```

080497a0 <main>:
080497a0: 55 push    ebp
080497a1: e9 e5 mov     ebp,esp
080497a3: e8 07 00 00 00 call   80497af <print_message>
080497a8: b8 00 00 00 00 mov     eax,0x0
080497ad: c9 2 3 4 5 leave
080497ae: c3 ret

080497af <print_message>:
080497b0: 55 push    ebp
080497b1: e9 e5 mov     ebp,esp
080497b2: 83 ec 10 sub    esp,0x10 ← 1b(0)
080497b5: 68 48 10 0e 08 push   0x80e1048
080497b8: e8 01 30 00 00 call   804c7c0 <_IO_pprintf>
080497b9: 83 c4 04 add    esp,0x4
080497c2: 8d 45 f0 lea    eax,[ebp-0x10] buffer [16]
080497c5: 50 push    eax
080497c6: e8 b5 75 00 00 call   8050d80 <_IO_gets>
080497cb: 83 c4 04 add    esp,0x4
080497cc: 8d 45 f0 lea    eax,[ebp-0x10]
080497d1: 50 push    eax
080497d2: 8d 5c 10 0e 08 push   0x80e105c
080497d7: e8 e4 2f 00 00 call   804c7c0 <_IO_pprintf>
080497df: c9 leave
080497e0: c3 ret

...
080499f0 <run_shell>:
080499f0: 55 push    ebp
080499f1: e9 e5 mov     ebp,esp
080499f3: 53 push    ebx
080499f4: b8 0b 00 00 00 mov     eax,0xb ; execve
080499f9: bb 68 10 0e 08 mov     ebx,0x80e1068 ; "/bin/bash"
080499fe: b9 00 00 00 00 mov     ecx,0x0
08049a03: ba 00 00 00 00 mov     edx,0x0
08049a08: cd 80 int    0x80
08049a0a: 5b pop    ebx

```

```

8049a0b: c9 leave
8049a0c: c3 ret

```

A continuación se muestra el equivalente en C que pudo reconstruir a partir de la salida anterior.

```

#include <stdio.h>
#include <unistd.h>

void print_message() {
    char buffer[16];
    printf("Ingrese su nombre: ");
    gets(buffer);
    printf("Hola %s, %s", buffer);
}

int run_shell() {
    return execve("/bin/bash", NULL, NULL);
}

int main() {
    print_message();
    return 0;
}

```

Además pudo notar la presencia en el binario de una función no usada, `run_shell`.

Pista: Se sabe que el código de operación de la instrucción CALL es E8h. Y que luego del E8 viene el salto relativo que debe hacerse.

Se pide:

1. Explicar cómo podría modificar el binario para que ejecute la función `run_shell` en lugar de `print_message`.
2. Explicar cómo se podría lograr un buffer overflow y ejecutar la función `run_shell`, cambiando la dirección de retorno (sin modificar el binario).
3. Responder: ¿Hay algún mecanismo para evitar que mediante el overflow se permita ejecutar otra función? Explique brevemente cómo funciona.

1) se puede modificar el binario

e7 07 00 00 00

donde se hace el call a `print_message`

se debería modificar cambiando 07 ya que indica el salto relativo para llegar a la dirección de la función a ejecutar

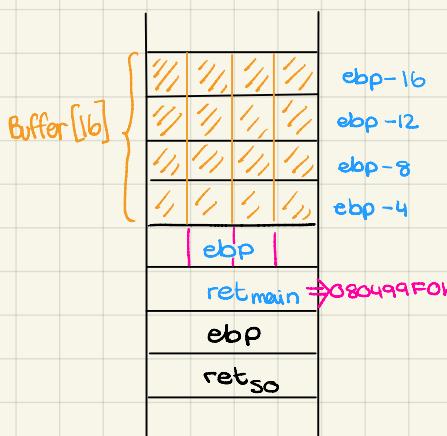
como busco correr `run_shell` esta ubicado en 080499f0 por lo tanto debo hacer una resta para ver el desplazamiento

la instrucción call ocupa 5 bytes. → arranco desde 080497a8



se remplaza e7 07 00 00 00 → e7 47 02 00 00

2) buffer overflow: se logaría ingresando más de 16 caracteres



entonces si busco llamar `run_shell` sin modificar el binario realizando un overflow del buffer de mas de 20 bytes ya que apartir de esa dirección se encuentra el `retmain` se debería llenar con el lugar de memoria 0x80499F0h donde esta `run_shell` de esta forma se sale de `print_message` y se empieza a correr `runshell`

3. gcc crea el canary que se posiciona arriba de esp para evitar la sobre-escritura del ret de la función

gcc le asigna al canary un valor aleatorio que se verifica antes de retornar de una función para chequear si hubo algún mal uso de la página como en este caso un buffer overflow

Ejercicio 3 (3 puntos) AEQ 32

Dado el siguiente programa compilado con GCC sin ningún parámetro adicional más que el archivo fuente:

```
#include<stdio.h>
int resta (int a) {
    int operando=9;
    return operando - a;
}
int main() {
    int a = resta(resta(9));
    printf("Resultado: %d\n", a);
    return ;
}
```

a) Explique con dibujos respectivos el contenido de la pila, durante toda la ejecución del programa. Muestre los valores de los registros del microprocesador que intervienen en los dibujos de la pila.

b) Muestre los valores que retorna cada función y la forma en que lo hace.

c) Muestre la salida en pantalla del programa.

a)

operando=9
Canary
ebp
ret main
9
MMMMm
canary
ebp
retso

resta(a)  
es lo primero  
que se ejecuta.

gcc reserva lugar de  
más  
eax = operando - a

$$eax = 0$$

asumo que  
se ejecuta  
la instrucción  
[add esp, 20]

para "liberar"  
los parámetros  
de la resta

operando=9
Canary
ebp
ret main
0
MMMMm
canary
ebp
retso

$$eax = \text{operando} - a$$

$$eax = 9$$

\* cheque de  
canary antes  
de salir

\* se checea el canary antes  
de salir de la función  
para ver si se rompio el  
stack o no.

asumo que  
se ejecuta  
la instrucción  
[add esp, 20]

para "liberar"  
los parámetros  
de la resta

canary
ebp
retmain
LC0
9

a = 9
MMMMm
canary
ebp
retso

gcc reserva lugar de  
más

\* chequeo del canary

gcc reserva lugar de  
más

$$eax = 13$$

asumo que  
se ejecuta  
la instrucción  
[add esp, 20]

para "liberar"  
los parámetros  
de la resta

$$eax = 13$$

\* chequeo de canary

Se desarma el stack frame,  
el esp = retso

se termina el programa y el main  
retorna lo que devolvio el printf  
igualmente en compilación se indicaria  
la falta de return como un warning

\$ ./ ejecutable  
Resultado: 9  
\$