

Práctica 4. Estructura de Computadores

Noelia Escalera Mejías. Grupo A3

Apartado 1: Cómo se ha codificado la bomba y código fuente

La contraseña de mi bomba es spmg, no obstante ésta está codificada en ASCII (11511210910310 si incluimos el \n). El pin es 2007, pero este no está codificado. He aquí el código fuente de mi programa:

```
// gcc -Og bomba_NoeliaEscaleraMejias.c -o bomba_NoeliaEscaleraMejias -no-pie

#include <stdio.h>    // para printf(), fgets(), scanf()
#include <stdlib.h>    // para exit()
#include <string.h>    // para strcmp()
#include <sys/time.h> // para gettimeofday(), struct timeval

#define SIZE 100
#define TLIM 5

char password[]="11511210910310"; // contraseña codificada
int passcode = 2007;                // pin

void boom(void){
    printf("\n"
           "*****\n"
           "*** BOOM!!! ***\n"
           "*****\n"
           "\n");
    exit(-1);
}

void defused(void){
    printf("\n"
           ".....\n"
           "... bomba desactivada ...\n"
           ".....\n"
           "\n");
    exit(0);
}

char * codifica (const char * c){
    char * a_devolver = malloc(sizeof(char) * (strlen(c) * 4 +1));
    char * nueva_c = a_devolver;
    *nueva_c = '\0';
    for (;*c; ++c){
        nueva_c += sprintf(nueva_c, "%u ", *c);
    }
    return a_devolver;
}
```

```

int main(){
    char pw[SIZE];
    int pc, n;

    struct timeval tv1,tv2;    // gettimeofday() secs-usecs
    gettimeofday(&tv1,NULL);

    do    printf("\nIntroduce la contraseña: ");
    while (fgets(pw, SIZE, stdin) == NULL );
    char * cad = codifica (pw);

    if    (strncmp(cad,password,sizeof(password)) )
        boom();

    gettimeofday(&tv2,NULL);
    if    ( tv2.tv_sec - tv1.tv_sec > TLIM )
        boom();

    do { printf("\nIntroduce el pin: ");
        if ((n=scanf("%i",&pc))==0)
            scanf("%*s") ==1;
        while (n!=1 );
        if    (pc != passcode )
            boom();

        gettimeofday(&tv1,NULL);
        if    ( tv1.tv_sec - tv2.tv_sec > TLIM )
            boom();

        defused();
    }
}

```

Apartado 2: Cómo descodificar la bomba

Nos metemos en gdb con el ejecutable bomba_NoeliaEscaleraMejias y avanzamos con ni hasta que nos pida la contraseña. Introducimos una contraseña aleatoria, hemos introducido “hola”.

```

Register group: general
rax      0x1b      27
rcx      0x0       0
rsi      0x64      100
rbp      0x4009a0  0x4009a0 <__libc_csu_init>
r8        0x0       0
r10      0x602010  6299664
r12      0x4006e0  4196064
r14      0x0       0
rip      0x4008bd  0x4008bd <main+81>
rbx      0x0       0
rdx      0x7ffff7dcfa00  140737351842304
rdi      0x7fffffffdb10  140737488345872
rsp      0x7fffffffdae0  0x7fffffffdae0
r9        0x0       0
r11      0x246     582
r13      0x7ffff7dc60    140737488346208
r15      0x0       0
eflags   0x206     [ PF IF ]

0x400896 <main+42>    lea    0x24f(%rip),%rsi    # 0x400aec
0x40089d <main+49>    mov    $0x1,%edi
0x4008a2 <main+54>    mov    $0x0,%eax
0x4008a7 <main+59>    callq 0x4006a0 <__printf_chk@plt>
0x4008ac <main+64>    lea    0x30(%rsp),%rdi
0x4008b1 <main+69>    mov    0x2007e8(%rip),%rdx    # 0x6010a0 <stdin@@GLIBC_2.2.5>
0x4008b8 <main+76>    mov    $0x64,%esi
> 0x4008bd <main+81>    callq 0x400680 <fgets@plt>
0x4008c2 <main+86>    test   %rax,%rax
0x4008c5 <main+89>    je     0x400896 <main+42>

native process 13840 In: main                                L??  PC: 0x4008bd
0x00000000004008a7 in main ()
(gdb) ni
0x00000000004008ac in main ()
(gdb) ni
0x00000000004008b1 in main ()
(gdb) ni
0x00000000004008b8 in main ()
(gdb) ni
0x00000000004008bd in main ()
(gdb) ni
Introduce la contraseña: hola

```

Noelia Escalera Mejías

Grupo A3

Si seguimos avanzando, vemos que se va a llamar a una función codificar, vamos a comprobar que se le pasa a la función. Vamos a ver qué se le pasa como argumento:

```
Register group: general
rax      0x7fffffffdb10  140737488345872
rcx      0xa616c6f      174156911
rsi      0x7fffffffdb10  140737488345872
rbp      0x4009a0 0x4009a0 <_libc_csu_init>
r8        0x602675 6301301
r10       0x602010 6299664
r12       0x4006e0 4196064
r14       0x0 0
rip      0x4008cc 0x4008cc <main+96>
rbx      0x0 0
rdx      0x7ffff7dd18d0 140737351850192
rdi      0x7fffffffdb10 140737488345872
rsp      0x7fffffffdae0 0x7fffffffdae0
r9        0x7ffff7fda500 140737353983232
r11       0x246 582
r13       0x7ffff7fddc60 140737488346208
r15       0x0 0
eflags   0x202 [ IF ]

0x4008ac <main+64> lea 0x30(%rsp),%rdi
0x4008b1 <main+69> mov 0x2007e8(%rip),%rdx # 0x6010a0 <stdin@GLIBC_2.2.5>
0x4008b8 <main+76> mov $0x64,%esi
0x4008bd <main+81> callq 0x400680 <fgets@plt>
0x4008c2 <main+86> test %rax,%rax
0x4008c5 <main+89> je 0x400896 <main+42>
0x4008c7 <main+91> lea 0x30(%rsp),%rdi
> 0x4008cc <main+96> callq 0x4007fb <codifica>
0x4008d1 <main+101> mov $0x14,%edx
0x4008d6 <main+106> lea 0x2007a3(%rip),%rsi # 0x601080 <password>

native process 13840 In: main L?? PC: 0x4008cc
(gdb) ni
0x00000000004008c2 in main ()
(gdb) ni
0x00000000004008c5 in main ()
(gdb) ni
0x00000000004008c7 in main ()
(gdb) ni
0x00000000004008cc in main ()
(gdb) x/1sb %rdi
0x7fffffffdb10: "hola\n"
(gdb) █
```

Efectivamente, se le ha pasado la contraseña que hemos introducido. Posteriormente se llama a una función strcmp para comparar dos cadenas, analizamos los argumentos:

```
Register group: general
rax      0x602a80 6302336
rcx      0x0 0
rsi      0x601080 6295680
rbp      0x4009a0 0x4009a0 <_libc_csu_init>
r8        0x0 0
r10       0x0 0
r12       0x4006e0 4196064
r14       0x0 0
rip      0x4008e0 0x4008e0 <main+116>
rbx      0x0 0
rdx      0x14 20
rdi      0x602a80 6302336
rsp      0x7fffffffdae0 0x7fffffffdae0
r9        0x0 0
r11       0x400aea 4197098
r13       0x7ffff7fddc60 140737488346208
r15       0x0 0
eflags   0x246 [ PF ZF IF ]

0x4008c2 <main+86> test %rax,%rax
0x4008c5 <main+89> je 0x400896 <main+42>
0x4008c7 <main+91> lea 0x30(%rsp),%rdi
0x4008cc <main+96> callq 0x4007fb <codifica>
0x4008d1 <main+101> mov $0x14,%edx
0x4008d6 <main+106> lea 0x2007a3(%rip),%rsi # 0x601080 <password>
0x4008dd <main+113> mov %rax,%rdi
> 0x4008e0 <main+116> callq 0x400650 <strcmp@plt>
0x4008e5 <main+121> test %eax,%eax
0x4008e7 <main+123> je 0x4008ee <main+130>

native process 13840 In: main L?? PC: 0x4008e0
(gdb) ni
0x00000000004008dd in main ()
(gdb) ni
0x00000000004008e0 in main ()
(gdb) x/1sb %rsi
El símbolo 'rsi' no está en el contexto actual.
(gdb) x/1sb %rsi
0x601080 <password>: "115 112 109 103 10 "
(gdb) x/1sb %rdi
0x602a80: "104 111 108 97 10 "
(gdb) █
```

El argumento password (se supone que la contraseña real de la bomba) es “115 112 109 103 10 ”. Se le compara con “104 111 108 97 10 ”, sin embargo, esta no es la contraseña que nosotros hemos introducido, es

la contraseña codificada. Vamos a reanudar la depuración, pero esta vez solo introduciremos un carácter (la a) como contraseña para ver qué le hace. Volvemos a chequear los argumentos de strcmp:

```
Register group: general
rax      0x602a80 6302336      rbx      0x0      0
rcx      0x0      0          rdx      0x14     20
rsi      0x601080 6295680      rdi      0x602a80 6302336
rbp      0x4009a0 0x4009a0 <__libc_csu_init>  rsp      0x7fffffffdae0 0x7fffffffdae0
r8        0x0      0          r9        0x0      0
r10       0x0      0          r11       0x400aea 4197098
r12       0x4006e0 4196064      r13       0x7fffffffdc60 140737488346208
r14       0x0      0          r15       0x0      0
rip      0x4008e0 0x4008e0 <main+116>  eflags   0x246     [ PF ZF IF ]

0x4008c2 <main+86>    test    %rax,%rax
0x4008c5 <main+89>    je      0x400896 <main+42>
0x4008c7 <main+91>    lea     0x30(%rsp),%rdi
0x4008cc <main+96>    callq  0x4007fb <codifica>
0x4008d1 <main+101>   mov     $0x14,%edx
0x4008d6 <main+106>   lea     0x2007a3(%rip),%rsi    # 0x601080 <password>
0x4008dd <main+113>   mov     %rax,%rdi
> 0x4008e0 <main+116> callq  0x400650 <strcmp@plt>
0x4008e5 <main+121>   test    %eax,%eax
0x4008e7 <main+123>   je      0x4008ee <main+130>

native process 13899 In: main
(gdb) ni
0x00000000004008dd in main ()
(gdb) ni
0x00000000004008e0 in main ()
(gdb) x/1sb $rsi
El símbolo 'rsi' no está en el contexto actual.
(gdb) x/1sb $rsi
0x601080 <password>:  "115 112 109 103 10 "
(gdb) x/1sb $rdi
0x602a80:  "97 10 "
(gdb) █
```

Obviamente la contraseña original del programa es la misma, pero ahora nuestra contraseña se ha traducido como “97 10 “. 97 es el equivalente ASCII de ‘a’ y 10 el de ‘\n’. Luego la codificación consiste en hacer el ascii de cada carácter de la contraseña y separarlo por un espacio, luego la contraseña del programa es “spmg\n”.

Seguimos avanzando en nuestro programa (modificando los registros convenientes para que no nos salte el boom): primero modificamos %rsi para que valga lo mismo que %rdi y %rax antes del test, así hacemos que %eax valga 0 y es como si hubiéramos introducido bien la contraseña.

```
Register group: general
rax      0x602a80 6302336      rbx      0x0      0
rcx      0x0      0          rdx      0x14     20
rsi      0x602a80 6302336      rdi      0x602a80 6302336
rbp      0x4009a0 0x4009a0 <__libc_csu_init>  rsp      0x7fffffffdae0 0x7fffffffdae0
r8        0x0      0          r9        0x0      0
r10       0x0      0          r11       0x400aea 4197098
r12       0x4006e0 4196064      r13       0x7fffffffdc60 140737488346208
r14       0x0      0          r15       0x0      0
rip      0x4008e0 0x4008e0 <main+116>  eflags   0x246     [ PF ZF IF ]

0x4008c2 <main+86>    test    %rax,%rax
0x4008c5 <main+89>    je      0x400896 <main+42>
0x4008c7 <main+91>    lea     0x30(%rsp),%rdi
0x4008cc <main+96>    callq  0x4007fb <codifica>
0x4008d1 <main+101>   mov     $0x14,%edx
0x4008d6 <main+106>   lea     0x2007a3(%rip),%rsi    # 0x601080 <password>
0x4008dd <main+113>   mov     %rax,%rdi
> 0x4008e0 <main+116> callq  0x400650 <strcmp@plt>
0x4008e5 <main+121>   test    %eax,%eax
0x4008e7 <main+123>   je      0x4008ee <main+130>

native process 14216 In: main
0x00000000004008e0 in main ()
(gdb) x/1sb $rsi
El símbolo 'rsi' no está en el contexto actual.
(gdb) x/1sb $rsi
0x601080 <password>:  "115 112 109 103 10 "
(gdb) x/1sb $rdi
0x602a80:  "97 10 "
(gdb) set $rsi=0x60302336
(gdb) set $rsi=60302336
(gdb) set $rsi=6302336
(gdb) █
```

Nota: Los dos primeros set son accidentales

Noelia Escalera Mejías

Grupo A3

Luego modificamos \$rax para que valga 5 antes del cmp y así no nos salte el boom por tiempo:

```
Register group: general
rax      0x5      5
rcx      0x20c49ba5e353f7cf 2361183241434822607
rsi      0x0      0
rbp      0x4009a0 0x4009a0 <__libc_csu_init>
r8       0x0      0
r10      0x7fffffffdae0 140737488345824
r12      0x4006e0 4196064
r14      0x0      0
rip      0x400902 0x400902 <main+150>
rbx      0x0      0
rdx      0xac61a 706074
rdi      0x7fffffffdb00 140737488345856
rsp      0x7fffffffdae0 0x7fffffffdae0
r9       0x0      0
r11      0x7fffffffdb00 140737488345856
r13      0x7fffffffdc60 140737488346208
r15      0x0      0
eflags   0x202    [ IF ]

0x4008e5 <main+121> test    %eax,%eax
0x4008e7 <main+123> je      0x4008ee <main+130>
0x4008e9 <main+125> callq  0x4007c7 <boom>
0x4008ee <main+130> lea     0x20(%rsp),%rdi
0x4008f3 <main+135> mov     $0x0,%esi
0x4008f8 <main+140> callq  0x400670 <gettimeofday@plt>
0x4008fd <main+145> mov     0x20(%rsp),%rax
> 0x400902 <main+150> sub     0x10(%rsp),%rax
0x400907 <main+155> cmp     $0x5,%rax
0x40090b <main+159> jle     0x400917 <main+171>

native process 14216 In: main L?? PC: 0x400902
0x00000000004008ee in main ()
(gdb) ni
0x00000000004008f3 in main ()
(gdb) ni
0x00000000004008f8 in main ()
(gdb) ni
0x00000000004008fd in main ()
(gdb) ni
0x0000000000400902 in main ()
(gdb) set $rax=5
(gdb) █
```

Ahora tenemos que llegar hasta donde se nos pide el pin, nosotros introduciremos 123.

```
Register group: general
rax      0x0      0
rcx      0x0      0
rsi      0x7fffffffdae0 140737488345836
rbp      0x4009a0 0x4009a0 <__libc_csu_init>
r8       0x0      0
r10      0x400b08 4197128
r12      0x4006e0 4196064
r14      0x0      0
rip      0x40093e 0x40093e <main+210>
rbx      0x0      0
rdx      0x7ffff7dd18c0 140737351850176
rdi      0x400b1c 4197148
rsp      0x7fffffffdae0 0x7fffffffdae0
r9       0x0      0
r11      0x246     582
r13      0x7fffffffdc60 140737488346208
r15      0x0      0
eflags   0x206    [ PF IF ]

0x400917 <main+171> lea     0x1ea(%rip),%rsi    # 0x400b08
0x40091e <main+178> mov     $0x1,%edi
0x400923 <main+183> mov     $0x0,%eax
0x400928 <main+188> callq  0x4006a0 <__printf_chk@plt>
0x40092d <main+193> lea     0xc(%rsp),%rsi
0x400932 <main+198> lea     0x1e3(%rip),%rdi    # 0x400b1c
0x400939 <main+205> mov     $0x0,%eax
> 0x40093e <main+210> callq  0x4006b0 <_isoc99_scanf@plt>
0x400943 <main+215> mov     %eax,%ebx
0x400945 <main+217> test    %eax,%eax

native process 14216 In: main L?? PC: 0x40093e
0x0000000000400928 in main ()
(gdb) ni
0x000000000040092d in main ()
(gdb) ni
0x0000000000400932 in main ()
(gdb) ni
0x0000000000400939 in main ()
(gdb) ni
0x000000000040093e in main ()
(gdb) ni
Introduce el pin: 123█
```

Avanzamos hasta que llegamos a una instrucción cmp a la que se le pasa como argumento un passcode. Comprobamos los argumentos de la operación:

```

--Register group: general--
rax      0x7d7  2007
rcx      0x10   16
rsi      0x1    1
rbp      0x4009a0 0x4009a0 <__libc_csu_init>
r8       0x0    0
r10      0x7fff7b82cc0 140737349430464
r12      0x4006e0 4196064
r14      0x0    0
rip      0x400962 0x400962 <main+246>
rbx      0x1    1
rdx      0x7ffff7dd18d0 140737351850192
rdi      0x0    0
rsp      0x7fffffffdae0 0x7fffffffdae0
r9       0x0    0
r11      0x400b1e 4197150
r13      0x7fffffffdc60 140737488346208
r15      0x0    0
eflags   0x246  [ PF ZF IF ]

0x400949 <main+221> lea 0x1cf(%rip),%rdi # 0x400b1f
0x400950 <main+228> mov $0x0,%eax
0x400955 <main+233> callq 0x4006b0 <__isoc99_scanf@plt>
0x40095a <main+238> jmp 0x400912 <main+166>
0x40095c <main+240> mov 0x20070e(%rip),%eax # 0x601070 <passcode>
> 0x400962 <main+246> cmp %eax,0xc(%rsp)
0x400966 <main+250> je 0x40096d <main+257>
0x400968 <main+252> callq 0x4007c7 <boom>
0x40096d <main+257> lea 0x10(%rsp),%rdi
0x400972 <main+262> mov $0x0,%esi

native process 14216 In: main L?? PC: 0x400962
(gdb) ni
0x000000000040095c in main ()
(gdb) ni
0x0000000000400962 in main ()
(gdb) x/ldw 0x601070
0x601070 <passcode>: 2007
(gdb) x/ldw $rsp+0xc
0x7fffffffdae0: 123
(gdb) x/ldw $eax

```

Efectivamente, uno de los argumentos es nuestro pin, el otro argumento es 2007, que suponemos que es el pin de la bomba. El pin no está codificado.

Ya podemos salir del depurador, vamos a ejecutar el programa para comprobar que hemos hallado los datos correctos:

```

noelia@noelia-HP-ENVY-17-Notebook-PC:~/Escritorio/Universidad/Segundo/EC/Practicas/Practica_4$ ./bomba_NoeliaEscaleraMejias
Introduce la contraseña: spmg
Introduce el pin: 2007
.....
... bomba desactivada ...
.....

```

Como podemos observar, los datos eran correctos y hemos desactivado la bomba.