**DESACTIVACIÓN DE BOMBA DIGITAL**

**Leandro Jorge Fernández Vega**

**1.-INTRODUCCIÓN**

Se plantea la desactivación de una bomba virtual mediante la averiguación de una contraseña y un PIN. El algoritmo de cifrado usado es un cifrado César clásico, donde el factor de cifrado se obtiene mediante la multiplicación de un índice y un desplazamiento, a modo de confusión, de manera que la letra cifrada es:

**letra\_cifrada= ’a’ + [(letra - ‘a’ + cifrado) % 26]**

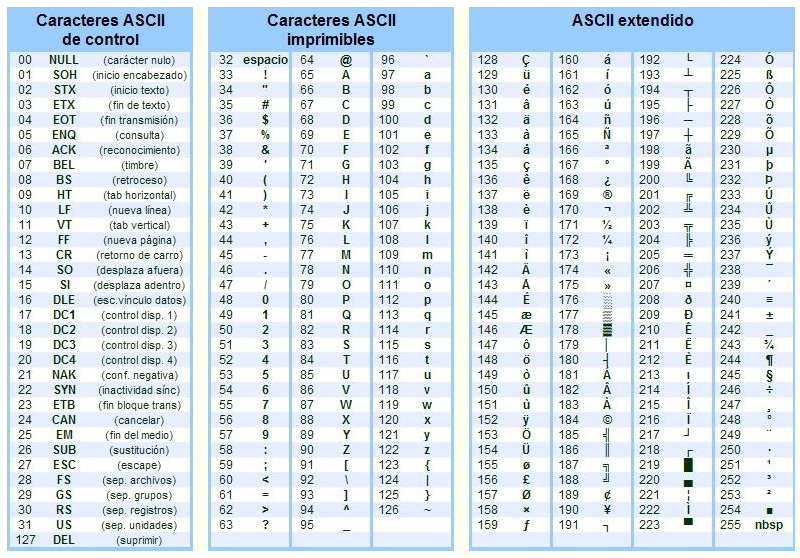
Esto es debido a que si se obtiene una letra posterior a la z vuelva al inicio del abecedario (consideramos alfabeto inglés en minúscula, es decir, sin la ”ñ”).

Elegimos las siguientes variables:

Contraseña= ”perrocarl\n”

PIN= 6980

Desplazamiento=Índice=2 🡺 Cifrado=4

 Para todo ello debemos saber que a cada carácter ASCII le corresponde un entero, lo que permite operar con ellos como si fueran enteros, según la siguiente tabla:

El algoritmo tiene un toque sutil: se modifica lo introducido por el usuario mientras que la contraseña queda intacta, por lo que para poder desactivar la bomba es necesario darse cuenta de que **lo introducido no es lo que verdaderamente se compara con la contraseña correcta.**

**2.- COMPILADO**

Nos centraremos en la desactivación mediante depuración. Para ello compilamos mediante

**gcc –O bombaLFV.c -o bombaLFV -no-pie; gdb -tui bombaLFV**

A continuación, mostramos lo necesario:

**layout asm**

**layout regs**

**3.- EXPLORACIÓN DE BOOM()**

Debemos explorar que ocurre en la explosión primero:

**br main**

**run**

**ni** hasta que toque introducir la contraseña, podemos poner un **breakpoint** ahí.

**br \*main+93**

**ni**

Introducimos una contraseña fácil de recordar, como “hola”.

Ahora debemos ver qué ocurre cuando explota la bomba. Para entrar en **boom()**:

**ni** hasta llegar a la llamada a boom()

**stepi** para entrar

**ni** hasta acabar

Vemos que no hay ninguna otra condición dentro de la propia función de finalización, por lo que la condición de explosión debe producirse antes de llamar a **boom()**.

**4.- OBTENCIÓN DE LA CONTRASEÑA MEDIANTE LA DETECCIÓN DEL ALGORITMO**

La idea es encontrar la orden de comparación de contraseñas. Deberemos darnos cuenta que cada letra se incrementa según un valor previo al bucle.

Para ello, observamos que la orden de comparación es la función strcmp (\*main+217).

**br \*main+217**

En \*main+222 encontramos **test %rax, %rax**. Podemos ver que, si ponemos %eax a 0, la bomba no explota.

**set $eax=0**

**ni**

Antes de strcmp se observa **lea 0x30(%rsp), %rdi** y **lea 0x2d24(%rip), %rsi** (\*main+200 y \*main+210 respectivamente). Es decir, en **%rdi** se encuentra lo que hemos introducido y en **%rsi** la contraseña.

**x/s $rsi** produce la siguiente salida: “perrocarl\n”

**x/s $rdi** produce la siguiente salida: “lspe\n” (==hola)

Si nos damos cuenta, cada letra de la contraseña que hemos introducido ha aumentado en 4 según el código ASCII (salvo “\n” debido a conveniencia en la programación). Por tanto, ya tenemos la contraseña. Debemos introducir lannkywnh (==perrocarl).

Ahora queda sortear un último obstáculo. Lo siguiente que encontramos es una llamada a la función **gettimeofday()** (\*main+241)**.** Vemos que en \*main+261 aparece la instrucción **cmp $0x3c, %rax**. Si el valor de **%rax** es mayor que el otro a comparar explotará.

**br \*main+261**

**set $rax=0**

**ni**

Vemos que la bomba no explota. Podemos intuir que se está comprobando si hemos tardado demasiado en introducir la contraseña.

**5.- OBTENCIÓN DEL PIN**

Lo próximo es la solicitud del PIN. Introduciremos uno fácil de recordar, como “123”.

**br \*main+321**

Ahora debemos encontrar la instrucción que compara el correcto y el nuestro. Observamos las instrucciones **cmp %eax, 0xc(%rsp)** en \*main+341 y **je** en \*main+345.

**p $eax** da como resultado 6980, que es el PIN a adivinar.

Por último, encontramos de nuevo una llamada a **gettimeofday()** y una comprobación de si se ha introducido el PIN lo suficientemente rápido, al igual que antes, por lo que se sortea de la misma forma. Finalmente una llamada a **defused()** finaliza el programa.