José María Gómez García **D2**

Solución bomba.c

Contraseña: Tamagotchi_01
Pin: 1996

Codify:
Tbodktzjpri;=
3994

main

```
   0x00000000004007c4 <main+0>: push   %rbx
   0x00000000004007c5 <main+1>: sub $0xa0,%rsp
   0x00000000004007cc <main+8>: mov %fs:0x28,%rax
   0x00000000004007d5 <main+17>:          mov   %rax,0x98(%rsp)
   0x00000000004007dd <main+25>:          xor   %eax,%eax
   0x00000000004007df <main+27>:          mov   $0x0,%esi
   0x00000000004007e4 <main+32>:          lea   0x10(%rsp),%rdi
   0x00000000004007e9 <main+37>:          callq  0x400600
<gettimeofday@plt>
   0x00000000004007ee <main+42>:          mov   $0x400a38,%esi
   0x00000000004007f3 <main+47>:          mov   $0x1,%edi
   0x00000000004007f8 <main+52>:          mov   $0x0,%eax
   0x00000000004007fd <main+57>:          callq  0x400630
<__printf_chk@plt>
   0x0000000000400802 <main+62>:          mov   0x200877(%rip),%rdx     #
0x601080 <stdin@@GLIBC_2.2.5>
   0x0000000000400809 <main+69>:          mov   $0x64,%esi
   0x000000000040080e <main+74>:          lea   0x30(%rsp),%rdi
   0x0000000000400813 <main+79>:          callq  0x400620 <fgets@plt>
   0x0000000000400818 <main+84>:          test   %rax,%rax
   0x000000000040081b <main+87>:          je    0x4007ee <main+42>
   0x000000000040081d <main+89>:          lea   0x30(%rsp),%rdi
   0x0000000000400822 <main+94>:          callq  0x400766
<codifypassword>
   0x0000000000400827 <main+99>:          mov   $0xf,%edx
   0x000000000040082c <main+104>:         mov   $0x601070,%esi
   0x0000000000400831 <main+109>:         lea   0x30(%rsp),%rdi
   0x0000000000400836 <main+114>:         callq  0x4005e0 <strncmp@plt>
   0x000000000040083b <main+119>:         test   %eax,%eax
   0x000000000040083d <main+121>:         je    0x400844 <main+128>
   0x000000000040083f <main+123>:         callq  0x400794 <boom>
   0x0000000000400844 <main+128>:         mov   $0x0,%esi
   0x0000000000400849 <main+133>:         lea   0x20(%rsp),%rdi
   0x000000000040084e <main+138>:         callq  0x400600
<gettimeofday@plt>
   0x0000000000400853 <main+143>:         mov   0x20(%rsp),%rax
   0x0000000000400858 <main+148>:         sub   0x10(%rsp),%rax
   0x000000000040085d <main+153>:         cmp   $0xa,%rax
   0x0000000000400861 <main+157>:         jle   0x400868 <main+164>
   0x0000000000400863 <main+159>:         callq  0x400794 <boom>
   0x0000000000400868 <main+164>:         mov   $0x400a54,%esi
   0x000000000040086d <main+169>:         mov   $0x1,%edi
   0x0000000000400872 <main+174>:         mov   $0x0,%eax
```

```
   0x0000000000400877 <main+179>:        callq  0x400630
<__printf_chk@plt>
   0x000000000040087c <main+184>:        lea    0xc(%rsp),%rsi
   0x0000000000400881 <main+189>:        mov    $0x400a68,%edi
   0x0000000000400886 <main+194>:        mov    $0x0,%eax
   0x000000000040088b <main+199>:        callq  0x400640
<__isoc99_scanf@plt>
   0x0000000000400890 <main+204>:        mov    %eax,%ebx
   0x0000000000400892 <main+206>:        test   %eax,%eax
   0x0000000000400894 <main+208>:        jne    0x4008a5 <main+225>
   0x0000000000400896 <main+210>:        mov    $0x400a6b,%edi
   0x000000000040089b <main+215>:        mov    $0x0,%eax
   0x00000000004008a0 <main+220>:        callq  0x400640
<__isoc99_scanf@plt>
   0x00000000004008a5 <main+225>:        cmp    $0x1,%ebx
   0x00000000004008a8 <main+228>:        jne    0x400868 <main+164>
   0x00000000004008aa <main+230>:        lea    0xc(%rsp),%rdi
   0x00000000004008af <main+235>:        callq  0x40078b
<codifypasscode>
   0x00000000004008b4 <main+240>:        mov    0x2007ae(%rip),%eax     #
0x601068 <passcode>
   0x00000000004008ba <main+246>:        cmp    %eax,0xc(%rsp)
   0x00000000004008be <main+250>:        je     0x4008c5 <main+257>
   0x00000000004008c0 <main+252>:        callq  0x400794 <boom>
```

codiffypassword

```
   0x0000000000400766 <+0>:   mov    $0x0,%esi
   0x000000000040076b <+5>:   mov    $0x0,%ecx
   0x0000000000400770 <+10>:  jmp    0x40077c <codifypassword+22>
   0x0000000000400772 <+12>:  add    %ecx,%eax
   0x0000000000400774 <+14>:  mov    %al,(%rdx)
   0x0000000000400776 <+16>:  add    $0x1,%ecx
   0x0000000000400779 <+19>:  add    $0x1,%esi
   0x000000000040077c <+22>:  movslq %esi,%rdx
   0x000000000040077f <+25>:  add    %rdi,%rdx
   0x0000000000400782 <+28>:  movzbl (%rdx),%eax
   0x0000000000400785 <+31>:  cmp    $0xa,%al
   0x0000000000400787 <+33>:  jne    0x400772 <codifypassword+12>
   0x0000000000400789 <+35>:  repz retq
```

codiffypasscode

```
0x000000000040078b <+0>:    mov    (%rdi),%eax
0x000000000040078d <+2>:    lea    0x2(%rax,%rax,1),%eax
0x0000000000400791 <+6>:    mov    %eax,(%rdi)
0x0000000000400793 <+8>:    retq
```

Solución:

Llama a la función codifypassword pasándole como argumento el string leído anteriormente y guardado en 0x30(%rsp):

```
0x000000000040081d <main+89>:       lea    0x30(%rsp),%rdi
0x0000000000400822 <main+94>:       callq  0x400766 <codifypassword>
```

Compara el resultado de codificar la contraseña introducida con $0x601070 ("Tbodktzjpri;=\n"):

```
0x0000000000400827 <main+99>:       mov    $0xf,%edx
0x000000000040082c <main+104>:      mov    $0x601070,%esi
0x0000000000400831 <main+109>:      lea    0x30(%rsp),%rdi
0x0000000000400836 <main+114>:      callq  0x4005e0 <strncmp@plt>
```

Resolver contraseña:

1. Ponemos un punto de ruptura cuando vaya a comparar la contraseña introducida:

```
(gdb) br *main+114
```

3. Ejecutamos e introducimos una cadena simple para ver cómo la ha codificado:

```
(gdb) run

...

Introduce la contraseña: aaaaaa
```

2. Cuando llega al punto de ruptura observamos que ha hecho con nuestra cadena:

```
(gdb) call printf("%s", $rdi)
abcdef
```

Nuestra cadena (“aaaaaa”) a pasado a ser “abcdef”.
La función <codifypassword> debería de estar sumando a cada carácter que forma la cadena su posición en la misma como si de un vector se tratase:

```
cadena[0] += 0;
cadena[1] += 1;
cadena[2] += 2;
        …
cadena[n] += n;
```

Si no nos fiamos de nuestra intuición podemos analizar el código:

```
Dump of assembler code for function codifypassword:
   0x0000000000400766 <+0>:   mov   $0x0,%esi
   0x000000000040076b <+5>:   mov   $0x0,%ecx
   0x0000000000400770 <+10>:  jmp   0x40077c <codifypassword+22>
   0x0000000000400772 <+12>:  add   %ecx,%eax
   0x0000000000400774 <+14>:  mov   %al,(%rdx)
   0x0000000000400776 <+16>:  add   $0x1,%ecx
   0x0000000000400779 <+19>:  add   $0x1,%esi
   0x000000000040077c <+22>:  movslq %esi,%rdx
   0x000000000040077f <+25>:  add   %rdi,%rdx
   0x0000000000400782 <+28>:  movzbl (%rdx),%eax
   0x0000000000400785 <+31>:  cmp   $0xa,%al
   0x0000000000400787 <+33>:  jne   0x400772 <codifypassword+12>
   0x0000000000400789 <+35>:  repz retq
End of assembler dump.
```

```
(gdb) b *main+94
```

Al principio iguala %esi y %ecx a 0, copia la dirección de la cadena en %rdx y mueve el contenido de la dirección a %eax y si %al = $0xa (“\n”) sigue ejecutando el bucle.

Si no fuese una cadena vacía seguirá ejecutándose desde la dirección 0x400772. %ecx y %esi son contadores por lo que valdrán siempre el valor de la iteración actual. Al principio de cada iteración le suma a %eax el valor del contador y lo guarda en la posición de la cadena ya que a %rdx también se le suma el valor del contador cada iteración. Poemos ver cómo varían los valores de la cadena así:

```
(gdb) br *0x400774
```

Y en cada iteración ver el valor de $al:

```
(gdb) p /c $al
$4 = 97 'a'
(gdb) cont
...
(gdb) p /c $al
$5 = 98 'b'
(gdb) cont
...
(gdb) p /c $al
$6 = 99 'c'
(gdb) cont
...
(gdb) p /c $al
$7 = 100 'd'
(gdb) cont
...
(gdb) p /c $al
$8 = 101 'e'
(gdb) cont
...
(gdb) p /c $al
$9 = 102 'f'
```

3. Por la tanto podemos ver cómo es la contraseña codificada:

```
(gdb) call printf("%s", $0x601070)
Tbodktzjpri;=
```

Si aplicamos la función inversa (una que le reste a cada carácter su posición en la cadena) a esa cadena (“Tbodktzjpri;=”) nos daría como resultado “Tamagotchi_01”. Vamos a probar ejecutando de nuevo el programa e introduciendo esta cadena:

```
Introduce la contraseña: Tamagotchi_01

Introduce el pin:
```

Ya hemos resuelto la contraseña.

Después nos pedirá introducir un pin. Podemos ver en el código una función que se llama <codifypasscode> que codifica el pin que hemos introducido:

```
      0x00000000004008aa <main+230>:      lea    0xc(%rsp),%rdi
      0x00000000004008af <main+235>:      callq  0x40078b <codifypasscode>
```

Y luego lo compara con el pin original:

```
      0x00000000004008b4 <main+240>:      mov    0x2007ae(%rip),%eax     #
0x601068 <passcode>
      0x00000000004008ba <main+246>:      cmp    %eax,0xc(%rsp)
```

Resolver el PIN:

Hagamos lo mismo que antes, comprobemos que hace con un pin simple:

1. Pongamos un punto de ruptura en la comparación:

```
(gdb) br *main+246
```

2. Ejecutemos e introducimos un pin sencillo:

```
(gdb) run
Starting program: /home/jose/Escritorio/bomba

Introduce la contraseña: Tamagotchi_01

Introduce el pin: 1111

Breakpoint 1, 0x00000000004008ba in main ()
```

3. Observemos que ha pasado con el pin:

```
(gdb) p /d *(int *)($rsp+0xc)
$11 = 2224
```

4. Vemos que nuestro pin a pasado de ser 1111 a 2224. Podríamos caer en la cuenta de que 2224 = 1111 * 2 + 2, pero por si acaso analicemos el código de la función <>:

```
Dump of assembler code for function codifypasscode:
   0x000000000040078b <+0>:   mov   (%rdi),%eax
   0x000000000040078d <+2>:   lea   0x2(%rax,%rax,1),%eax
   0x0000000000400791 <+6>:   mov   %eax,(%rdi)
   0x0000000000400793 <+8>:   retq
End of assembler dump.
```

```
mov   (%rdi),%eax
```

Copia nuestro pin en %eax.

```
lea   0x2(%rax,%rax,1),%eax
```

%eax = 1 * PIN + PIN + 2

```
mov   %eax,(%rdi)
```

Guarda el resultado dónde estaba nuestro pin. Lo sobreescribe con el resultado.

5. Apliquemos entonces la función inversa al pin original:

```
(gdb) p $eax
$12 = 3994
```

3994 - 2 = 3992
3992 / 2 = 1996

Por lo tanto 1996 debería de ser el pin original. Probemos:

```
Introduce la contraseña: Tamagotchi_01

Introduce el pin: 1996

.........................
··· bomba desactivada ···
.........................
```

¡BOMBA DESACTIVADA!