Resolución bomba_JuliaCano

1. Introducción

Para resolver esta bomba lo primero que hay que hacer es abrir el ejecutable con el depurador gdb:

```
>$ gdb -tui bomba_JuliaCano
```

Una vez cargado y con las instrucciones visibles podemos ver que realiza una comprobación con dos variablestras leer la password y el passcode introducidos por el usuario. Mediante las órdenes para imprimir por pantalla las variables, y teniendo en cuenta que nuestra entrada pasa por un proceso de codificación podemos determinar que los valores de ambas variables están codificadas:

Variable	Valor Codificado
password	ngxkquc
passcode	2004

Entonces genero las instrucciones del ejecutable mediante `>$ objdump -d bomba_JuliaCano`, y analizo las instrucciones de codificación de password y passcode.

2. Password encoding

La función de encriptado de la password es la siguiente:

```
0000000000401256 <encripta>:
  401256:       f3 0f 1e fa             endbr64
  40125a:       53                      push   %rbx
  40125b:       48 89 f3                mov    %rsi,%rbx
  40125e:       48 89 fe                mov    %rdi,%rsi
  401261:       48 89 df                mov    %rbx,%rdi
  401264:       e8 57 fe ff ff          callq  4010c0 <strcpy@plt>
  401269:       be 00 00 00 00          mov    $0x0,%esi
  40126e:       89 f2                   mov    %esi,%edx
  401270:       48 c7 c1 ff ff ff ff    mov    $0xffffffffffffffff,%rcx
  401277:       b8 00 00 00 00          mov    $0x0,%eax
  40127c:       48 89 df                mov    %rbx,%rdi
  40127f:       f2 ae                   repnz scas %es:(%rdi),%al
  401281:       48 89 c8                mov    %rcx,%rax
  401284:       48 f7 d0                not    %rax
  401287:       48 83 e8 02             sub    $0x2,%rax
  40128b:       48 39 c2                cmp    %rax,%rdx
  40128e:       72 02                   jb     401292 <encripta+0x3c>
  401290:       5b                      pop    %rbx
  401291:       c3                      retq
  401292:       48 01 da                add    %rbx,%rdx
  401295:       0f b6 02                movzbl (%rdx),%eax
  401298:       83 c0 02                add    $0x2,%eax
  40129b:       88 02                   mov    %al,(%rdx)
  40129d:       83 c6 01                add    $0x1,%esi
  4012a0:       eb cc                   jmp    40126e <encripta+0x18>
```

Podemos observar que al principio copia el string introducido en el primer argumento en otro string vacío pasado como segundo argumento, una estructura que se ha repetido bastante en el resto de bombas. Además, tiene una estructura de bucle bastante clara. En concreto las lineas que nos dan una pista sobre lo que realiza el encriptado del string son las siguientes:

```
  401298:       83 c0 02                add    $0x2,%eax
  40129b:       88 02                   mov    %al,(%rdx)
```

Podemos observar claramente como se suma 2 al valor del char y se guarda en su posición de memoria. Por ello la password sin encriptar se puede adivinar fácilmente haciendo una función que haga lo contrario, o mirando una tabla ascii.

3. Passcode encoding

La función de encriptado del passcode es mucho más sencilla:

```
00000000004012a2 <encripta_num>:
  4012a2:       f3 0f 1e fa             endbr64
  4012a6:       8d 47 0f                lea    0xf(%rdi),%eax
  4012a9:       c3                      retq
```

Podemos ver como el pin se encripta transofrmandolo como:

$$pin = pin + 15$$

4. Valores descodificados

Haciendo el cálculo inverso a la encriptación podemos adivinar rápidamente que los valores a introducir son:

Variable	Valor Codificado	Valor Descodificado
password	ngxkquc	**leviosa**
passcode	2004	**1989**