Explicación de la bomba

David Infante Casas
75165296P
davidinfante@correo.ugr.es
Grupo de prácticas: D1

Se ha usado la herramienta gdb.

La contraseña y pin originales que se han de escribir son: “agallas” y 1000
La contraseña y pin modificados que lee el programa son: “egallas” y 1024

Comenzamos escribiendo en el terminal gdb -tui bomba_DavidInfanteCasas. Acomodamos la vista con layout asm y layout regs y obtenemos el siguiente código, creamos un breakpoint en main (br main) y vamos avanzando con la orden ni:

```
0x400780 <main>          push   %rbx
0x400781 <main+1>        sub    $0xa0,%rsp
0x400788 <main+8>        mov    %fs:0x28,%rax
0x400791 <main+17>       mov    %rax,0x98(%rsp)
0x400799 <main+25>       xor    %eax,%eax
0x40079b <main+27>       lea    0x10(%rsp),%rdi
0x4007a0 <main+32>       mov    $0x0,%esi
0x4007a5 <main+37>       callq  0x4005f0 <gettimeofday@plt>
0x4007aa <main+42>       lea    0x2ef(%rip),%rsi        # 0x400aa0
```

Comprobamos qué hay en esta dirección de memoria con “x/lsb 0x400aa0” y vemos que es el mensaje de pedir la contraseña.

```
0x4007b1 <main+49>       mov    $0x1,%edi
0x4007b6 <main+54>       mov    $0x0,%eax
0x4007bb <main+59>       callq  0x400610 <__printf_chk@plt>
0x4007c0 <main+64>       lea    0x30(%rsp),%rdi
0x4007c5 <main+69>       mov    0x2008b4(%rip),%rdx        # 0x601080
<stdin@@GLIBC_2.2.5>
0x4007cc <main+76>       mov    $0x64,%esi
0x4007d1 <main+81>       callq  0x400600 <fgets@plt>
```

Nos pide la contraseña e introducimos una aleatoriamente, por ejemplo “hola”. Mirando la pestaña regs vemos que guarda “hola” en %rax.

```
0x4007d6 <main+86>      test   %rax,%rax
0x4007d9 <main+89>      je     0x4007aa <main+42>
0x4007db <main+91>      lea    0x30(%rsp),%rbx
```

Copia lo que hemos introducido “hola” en %rbx.

```
0x4007e0 <main+96>      mov    %rbx,%rdi
```

Llegamos a una instrucción que llama a la función “yoNoModificoPwd” un tanto sospechosa, vamos a usar “si” para entrar a ella y ver qué hace.

```
0x4007e3 <main+99>      callq  0x40075b <yoNoModificoPwd>
```

Comprobamos qué tiene la contraseña después de haber pasado por la función con “x/lsb $rbx” y nos devuelve “eola”. Si probamos con otras contraseñas e inspeccionamos la función como hemos dicho previamente (vemos que hace un movb por 0x65 que en ascii es una ‘e’ minúscula), podemos acabar deduciendo que lo que hace es cambiar el primer carácter por ‘e’.

```
0x4007e8 <main+104>     mov    $0x9,%edx
0x4007ed <main+109>     lea    0x200874(%rip),%rsi        # 0x601068
<password>
```

Leemos el registro de la variable “password” para ver qué contiene con “x/lsb 0x601068” y nos devuelve “egallas\n”. Como ya sabemos que a la contraseña se le pasa la función “yoNoModificoPwd” y esta cambia el primer carácter por ‘e’, la única posible solución es “egallas”.

```
0x4007f4 <main+116>     mov    %rbx,%rdi
0x4007f7 <main+119>     callq  0x4005d0 <strncmp@plt>
0x4007fc <main+124>     test   %eax,%eax
```

La siguiente instrucción es un salto y debemos debemos entender qué hace, para ello comprobamos que llama a la función boom, la cual queremos evitar, entonces tratamos de hacer que el test anterior se cumpla e igualamos los registros poniendo %eax a 0 con “set $eax=0” y avanzamos saltando.

```
0x4007fe <main+126>     je     0x400805 <main+133>
0x400800 <main+128>     callq  0x400727 <boom>
0x400805 <main+133>     lea    0x20(%rsp),%rdi
0x40080a <main+138>     mov    $0x0,%esi
0x40080f <main+143>     callq  0x4005f0 <gettimeofday@plt>
0x400814 <main+148>     mov    0x20(%rsp),%rax
0x400819 <main+153>     sub    0x10(%rsp),%rax
0x40081e <main+158>     cmp    $0x3c,%rax
```

De nuevo, otro salto, usando la pestaña regs comprobamos que compara el valor hexadecimal 0x3c (60 en decimal) con %rax. Como se trata de un salto jle %rax ha de ser menor que 60 y lo pondremos a 1 por ejemplo con “set $rax=1”.

```
0x400822 <main+162>     jle    0x40082e <main+174>
0x400824 <main+164>     callq  0x400727 <boom>
0x400829 <main+169>     cmp    $0x1,%ebx
0x40082c <main+172>     je     0x400873 <main+243>
0x40082e <main+174>     lea    0x289(%rip),%rsi        # 0x400abe
0x400835 <main+181>     mov    $0x1,%edi
0x40083a <main+186>     mov    $0x0,%eax
0x40083f <main+191>     callq  0x400610 <__printf_chk@plt>
0x400844 <main+196>     lea    0xc(%rsp),%rsi
0x400849 <main+201>     lea    0x282(%rip),%rdi        # 0x400ad2
0x400850 <main+208>     mov    $0x0,%eax
0x400855 <main+213>     callq  0x400620 <__isoc99_scanf@plt>
```

Nos pide el pin e introducimos aleatoriamente 1234.

```
0x40085a <main+218>     mov    %eax,%ebx
0x40085c <main+220>     test   %eax,%eax
0x40085e <main+222>     jne    0x400829 <main+169>
0x400860 <main+224>     lea    0x26e(%rip),%rdi        # 0x400ad5
0x400867 <main+231>     mov    $0x0,%eax
0x40086c <main+236>     callq  0x400620 <__isoc99_scanf@plt>
0x400871 <main+241>     jmp    0x400829 <main+169>
0x400873 <main+243>     mov    0xc(%rsp),%edi
```

Comprobamos que hay en el registro con “x/lwu $rsp+0xc” y vemos que es nuestra contraseña “1234”.

```
0x400877 <main+247>     callq  0x40077c <yoNoModificoPin>
0x40087c <main+252>     mov    %eax,0xc(%rsp)    69>
```

De nuevo una función un tanto sospechosa “yoNoModificoPin”, vamos a comprobar que hay en el registro de nuestro pin con “x/lwu $rsp+0xc” y nos da “1258”, que es 1234 + 24. Al probar otros números y entrar a la función vemos que lo único que hace es sumar 24 (0x18 en hexadecimal tal y como pone en la función) al número que se le pasa.

```
0x400880 <main+256>     cmp    0x2007da(%rip),%eax        # 0x601060
<passcode>
```

Comprobamos qué contiene la dirección de memoria que corresponde a la variable “passcode” con “x/lwu 0x601060” y nos devuelve 1024, por tanto ya sabemos el pin con el que lo compara. Si el pin es 1024 y la función pasada antes al pin introducido suma 24, la única solución posible de pin es 1000.

```
0x400886 <main+262>     je     0x40088d <main+269>
0x400888 <main+264>     callq  0x400727 <boom>
0x40088d <main+269>     lea    0x10(%rsp),%rdi
0x400892 <main+274>     mov    $0x0,%esi
0x400897 <main+279>     callq  0x4005f0 <gettimeofday@plt>
0x40089c <main+284>     mov    0x10(%rsp),%rax
0x4008a1 <main+289>     sub    0x20(%rsp),%rax
0x4008a6 <main+294>     cmp    $0x3c,%rax
```

Otro salto salto igual que uno anterior, usando la pestaña regs comprobamos que compara el valor hexadecimal 0x3c (60 en decimal) con %rax. Como se trata de un salto jle %rax ha de ser menor que 60 y lo pondremos a 1 por ejemplo con “set $rax=1”. Aunque solo debemos hacer esto si hemos tardado más de 60 segundos en llegar aquí.

```
0x4008aa <main+298>     jle    0x4008b1 <main+305>
0x4008ac <main+300>     callq  0x400727 <boom>
0x4008b1 <main+305>     callq  0x400741 <defused>
```

Una vez acabada la ejecución del programa ya sabemos la contraseña y el pin y lo que hacen las funciones que los modifican.