Resolución bomba propia

Alumno: Juan Sánchez Rodríguez

Para resolver esta práctica he usado la herramienta gdb.

En el terminal escribimos “gdb bomba” para que se nos abra la bomba digital, después hacemos un breakpoint en el main usando “break main”, hacemos “run” y usamos “disas” para mostrar el archivo. La pantalla que vemos es la siguiente:

```
   0x08048627 <+0>:  lea    0x4(%esp),%ecx
   0x0804862b <+4>:  and    $0xfffffff0,%esp
   0x0804862e <+7>:  pushl  -0x4(%ecx)
   0x08048631 <+10>:         push   %ebp
   0x08048632 <+11>:         mov    %esp,%ebp
   0x08048634 <+13>:         push   %ecx
=> 0x08048635 <+14>:         sub    $0x84,%esp
   0x0804863b <+20>:         mov    %gs:0x14,%eax
   0x08048641 <+26>:         mov    %eax,-0xc(%ebp)
   0x08048644 <+29>:         xor    %eax,%eax
   0x08048646 <+31>:         sub    $0x8,%esp
   0x08048649 <+34>:         push   $0x0
   0x0804864b <+36>:         lea    -0x80(%ebp),%eax
   0x0804864e <+39>:         push   %eax
   0x0804864f <+40>:call   0x8048430 <gettimeofday@plt>
   0x08048654 <+45>:         add    $0x10,%esp
   0x08048657 <+48>:         sub    $0xc,%esp
   0x0804865a <+51>:         push   $0x8048874
   0x0804865f <+56>:call   0x8048410 <printf@plt>
   0x08048664 <+61>:         add    $0x10,%esp
   0x08048667 <+64>:         mov    0x804a060,%eax
   0x0804866c <+69>:         sub    $0x4,%esp
   0x0804866f <+72>:push   %eax
   0x08048670 <+73>:         push   $0x64
   0x08048672 <+75>:         lea    -0x70(%ebp),%eax
   0x08048675 <+78>:         push   %eax
   0x08048676 <+79>:         call   0x8048420 <fgets@plt>
   0x0804867b <+84>:         add    $0x10,%esp
   0x0804867e <+87>:         movl   $0x0,-0x84(%ebp)
   0x08048688 <+97>:         jmp    0x80486b6 <main+143>
   0x0804868a <+99>:         lea    -0x70(%ebp),%edx
   0x0804868d <+102>:        mov    -0x84(%ebp),%eax
   0x08048693 <+108>:        add    %edx,%eax
   0x08048695 <+110>:        movzbl (%eax),%edx
   0x08048698 <+113>:        mov    -0x84(%ebp),%eax
   0x0804869e <+119>:        add    $0x804a034,%eax
   0x080486a3 <+124>:        movzbl (%eax),%eax
   0x080486a6 <+127>:        cmp    %al,%dl
   0x080486a8 <+129>:        je     0x80486af <main+136>
```

```
0x080486aa <+131>:      call   0x804859b <boom>
0x080486af <+136>:      addl   $0x1,-0x84(%ebp)
0x080486b6 <+143>:      cmpl   $0xf,-0x84(%ebp)
0x080486bd <+150>:      jle    0x804868a <main+99>
0x080486bf <+152>:      sub    $0x8,%esp
0x080486c2 <+155>:      push   $0x0
0x080486c4 <+157>:      lea    -0x78(%ebp),%eax
0x080486c7 <+160>:      push   %eax
0x080486c8 <+161>:      call   0x8048430 <gettimeofday@plt>
0x080486cd <+166>:      add    $0x10,%esp
0x080486d0 <+169>:      mov    -0x78(%ebp),%edx
0x080486d3 <+172>:      mov    -0x80(%ebp),%eax
0x080486d6 <+175>:      sub    %eax,%edx
0x080486d8 <+177>:      mov    %edx,%eax
0x080486da <+179>:      cmp    $0x5,%eax
0x080486dd <+182>:      jle    0x80486e4 <main+189>
0x080486df <+184>:      call   0x804859b <boom>
0x080486e4 <+189>:      sub    $0xc,%esp
0x080486e7 <+192>:      push   $0x804888f
0x080486ec <+197>:      call   0x8048410 <printf@plt>
0x080486f1 <+202>:      add    $0x10,%esp
0x080486f4 <+205>:      sub    $0x8,%esp
0x080486f7 <+208>:      lea    -0x88(%ebp),%eax
0x080486fd <+214>:      push   %eax
0x080486fe <+215>:      push   $0x80488a6
0x08048703 <+220>:      call   0x8048480 <__isoc99_scanf@plt>
0x08048708 <+225>:      add    $0x10,%esp
0x0804870b <+228>:      mov    -0x88(%ebp),%eax
0x08048711 <+234>:      sub    $0xc,%esp
0x08048714 <+237>:      push   %eax
0x08048715 <+238>:      call   0x804861b <moduloNadaSospechoso>
0x0804871a <+243>:      add    $0x10,%esp
0x0804871d <+246>:      mov    %eax,-0x88(%ebp)
0x08048723 <+252>:      mov    -0x88(%ebp),%edx
0x08048729 <+258>:      mov    0x804a048,%eax
0x0804872e <+263>:      cmp    %eax,%edx
0x08048730 <+265>:      je     0x8048737 <main+272>
0x08048732 <+267>:      call   0x804859b <boom>
0x08048737 <+272>:      sub    $0x8,%esp
0x0804873a <+275>:      push   $0x0
0x0804873c <+277>:      lea    -0x80(%ebp),%eax
0x0804873f <+280>:      push   %eax
0x08048740 <+281>:      call   0x8048430 <gettimeofday@plt>
0x08048745 <+286>:      add    $0x10,%esp
0x08048748 <+289>:      mov    -0x80(%ebp),%edx
0x0804874b <+292>:      mov    -0x78(%ebp),%eax
0x0804874e <+295>:      sub    %eax,%edx
0x08048750 <+297>:      mov    %edx,%eax
0x08048752 <+299>:      cmp    $0x5,%eax
0x08048755 <+302>:      jle    0x804875c <main+309>
0x08048757 <+304>:      call   0x804859b <boom>
0x0804875c <+309>:      call   0x80485db <defused>
```

```
0x08048761 <+314>:      mov    $0x0,%eax
0x08048766 <+319>:      mov    -0xc(%ebp),%ecx
0x08048769 <+322>:      xor    %gs:0x14,%ecx
0x08048770 <+329>:      je     0x8048777 <main+336>
0x08048772 <+331>:      call   0x8048440 <__stack_chk_fail@plt>
0x08048777 <+336>:      mov    -0x4(%ebp),%ecx
0x0804877a <+339>:      leave
0x0804877b <+340>:      lea    -0x4(%ecx),%esp
0x0804877e <+343>:      ret
```

Ahora analizaré esto paso por paso.
Lo primero en lo que me fijo es en lo siguiente:

```
0x0804865a <+51>:   push   $0x8048874
0x0804865f <+56>:   call   0x8048410 <printf@plt>
```

Ya hace una llamada print, supongoque es el mensaje para que introduzcas la contraseña, pero para asegurarme ejecuto la siguiente instrucción “print (char*) 0x8048874”, donde la respuesta del gdb es efectivamente “$1 = 0x8048874 "Introduce la contraseña: "”. Esto quiere decir que no mucho más adelante se realizará la introducción de la contraseña, por ello nos fijamos un poco más adelante en “0x08048676 <+79>:   call   0x8048420 <fgets@plt>”, ya sabemos que esta función es para introducir datos. Bien ahora uso “nexti” hasta que me pida la contraseña, introduzco una cualquiera, en este caso “holamundo” y uso “disas”. Nos encontramos las siguientes instrucciones:

```
0x0804867b <+84>:       add    $0x10,%esp
0x0804867e <+87>:       movl   $0x0,-0x84(%ebp)
0x08048688 <+97>:       jmp    0x80486b6 <main+143>
```

Así que nos dirigimos a <main+143> usando “nexti” hasta después del jmp. Y nos encontramos lo siguiente:

```
0x080486b6 <+143>:      cmpl   $0xf,-0x84(%ebp)
0x080486bd <+150>:      jle    0x804868a <main+99>
```

Así que nos dirigimos a <main+99> usando “nexti” hasta después del jle. Y nos encontramos lo siguiente:

```
0x0804868a <+99>:       lea    -0x70(%ebp),%edx
0x0804868d <+102>:      mov    -0x84(%ebp),%eax
0x08048693 <+108>:      add    %edx,%eax
0x08048695 <+110>:      movzbl (%eax),%edx
0x08048698 <+113>:      mov    -0x84(%ebp),%eax
0x0804869e <+119>:      add    $0x804a034,%eax
0x080486a3 <+124>:      movzbl (%eax),%eax
0x080486a6 <+127>:      cmp    %al,%dl
0x080486a8 <+129>:      je     0x80486af <main+136>
0x080486aa <+131>:      call   0x804859b <boom>
0x080486af <+136>:      addl   $0x1,-0x84(%ebp)
0x080486b6 <+143>:      cmpl   $0xf,-0x84(%ebp)
0x080486bd <+150>:      jle    0x804868a <main+99>
```

Aquí podemos observar que hay un bucle que empieza en <+99>, hace un cmp en <+127>, y después de esto hace un je en <+129> a <main+136> y después en <+143> hace un cmpl y continua con un jle en <+150> a <main+99>. Aquí vemos que se está comparando algo en bucle y si no coincide llama a <boom>. Vamos a usar "nexti" hasta la siguiente instrucción a "0x080486a6 <+127>: cmp %al,%dl" para ver que se está comparando. Usamos la orden "display /c $dl" para ver que contiene, el gdb nos dice lo siguiente "1: /c $dl = 104 'h'", hacemos lo mismo para $al y obtenemos lo siguiente "2: /c $al = 99 'c'", recordamos que nosotros habíamos introducido "holamundo" en la contraseña, y que está empieza por "h", por lo que vemos que lo que hace la bomba es comparar la contraseña letra a letra. Pero claro, con esto vemos que de algún lado tiene que sacar las letras de la susodicha contraseña, así que nos volvemos a fijar en el principio de lo comentado anteriormente:

```
0x0804868d <+102>:    mov    -0x84(%ebp),%eax
0x08048693 <+108>:    add    %edx,%eax
0x08048695 <+110>:    movzbl (%eax),%edx
0x08048698 <+113>:    mov    -0x84(%ebp),%eax
0x0804869e <+119>:    add    $0x804a034,%eax
0x080486a3 <+124>:    movzbl (%eax),%eax
```

Vemos que hace la misma sucesión de instrucciones para dos cosas que luego compara así que vemos lo que hay en $0x804a034 (ya que se puede pensar que aquí está o bien lo que nosotros hemos introducido o o bien la contraseña), usando "print (char*) 0x804a034". La respuesta del gdb es "$10 = 0x804a034 <password> "cuervoinsurgente\n"". Con esto volvemos a hacer "run" donde se nos parará en el breakpoint 1, y usamos "cont", nos pedirá que introduzcamos la contraseña, introducimos "cuervoinsurgente" y vemos que es correcto.

Ahora vamos a buscar la clave numérica, para ello volvemos a hacer "run" y "disas". Ahora nos fijamos más abajo en:

```
0x080486e7 <+192>:    push   $0x804888f
0x080486ec <+197>:    call   0x8048410 <printf@plt>
```

Podemos suponer que este printf es para decirnos que introduzcamos el código, pero para comprobarlo usamos "print (char*) 0x804888f" donde la respuesta del gdb es "$12 = 0x804888f "Introduce el código: """ por lo que vamos que estamos en lo cierto. Vamos a hacer un breakpoint aquí usando "break * 0x080486ec", podemos borrar el primer breakpoint usando "delete 1". Ahora hacemos "cont" y nos saldrá el mensaje de que introduzcamos la clave, introducimos "cuervoinsurgente" y nos salta el breakpoint que acabamos de hacer. Nos fijamos que poco después del printf hay un scanf, aquí es donde se va a introducir el código que metamos, observamos el código:

```
0x080486fe <+215>:    push   $0x80488a6
0x08048703 <+220>:    call   0x8048480 <__isoc99_scanf@plt>
```

Usamos "print (char*) 0x80488a6" y el gdb nos responde con "$13 = 0x80488a6 "%i"" aquí vemos que el tipo de dato que se introduce es un "int" por lo tanto sabemos que es donde se introdice nuestra contraseña. Usamos "nexti" hasta que nos pida que introduzcamos el código, una vez que nos salte ponemos un código cualquiera, por ejemplo "100". Vemos las siguientes instrucciones:

```
0x08048708 <+225>:    add    $0x10,%esp
0x0804870b <+228>:    mov    -0x88(%ebp),%eax
0x08048711 <+234>:    sub    $0xc,%esp
```

```
0x08048714 <+237>:    push   %eax
0x08048715 <+238>:    call   0x804861b <moduloNadaSospechoso>
0x0804871a <+243>:    add    $0x10,%esp
0x0804871d <+246>:    mov    %eax,-0x88(%ebp)
0x08048723 <+252>:    mov    -0x88(%ebp),%edx
0x08048729 <+258>:    mov    0x804a048,%eax
0x0804872e <+263>:    cmp    %eax,%edx
0x08048730 <+265>:    je     0x8048737 <main+272>
0x08048732 <+267>:    call   0x804859b <boom>
```

Usamos “nexti” hasta la siguiente instrucción a “0x0804870b <+228>: mov -0x88(%ebp),%eax” y comprobamos el contenido de “-0x88(%ebp)” usando la instrucción “print *(int*)($ebp-0x88)” y el gdb nos responde con “$14 = 100” por lo que vemos que aquí es donde se ha almacenado nuestra clave. Después vemos que se hace una llamada a “moduloNadaSospechoso” y se hacen unos mov en y después se hace un cmp, un je, y una llamada a explotar, vemos que en ese cmp se están comparando evidentemente los dos números así que hacemo “nexti” hasta el cmp. Una vez aquí hacemos “print *(int*) 0x804a048” donde el gdb nos responde con “$15 = 4593”, esta sabemos que es el código, miramos también lo que hay en “-0x88(%ebp)” y el gdb nos responde con “$16 = 48” vemos que este número es diferente al que hemos introducido, volvemos a hacer “run” al programa introducimos la clave y el número “ 4593” y vemos que la bomba explota. Probamos el proceso anterior con números diferentes al 100 viendo el cambio que hace después del “moduloNadaSospechoso” y nos damos cuenta de que a nuestro código le resta 52. Así que probamos con el código “4645” y efectivamente “Bomba desactivada”.