

## Bomba José Manuel Pérez Léndinez

Se ha realizado con gbd ya que ddd me dio muchos problemas.

### Compresión del mecanismo de encriptación para la contraseña.

Lo primero que vemos al analizar el código es que después de la petición de introducir la contraseña se encuentra un bucle. Analizando su código se puede ver que es el encargado de cifrar la contraseña.

```
0x080486cd <+84>: call 0x8048470 <fgets@plt>
0x080486d2 <+89>: lea 0x38(%esp),%eax
0x080486d6 <+93>: mov %eax,%esp
0x080486d9 <+96>: call 0x80484d0 <strlen@plt>
0x080486de <+101>: sub $0x1,%eax
0x080486e1 <+104>: mov %eax,0x24(%esp)
0x080486e5 <+108>: movl $0x0,0x20(%esp)
0x080486ed <+116>: jmp 0x8048710 <main+151>
0x080486ef <+118>: lea 0x38(%esp),%edx
0x080486f3 <+122>: mov 0x20(%esp),%eax
0x080486f7 <+126>: add %edx,%eax
0x080486f9 <+128>: movzbl (%eax),%eax
0x080486fc <+131>: add $0x1,%eax
0x080486ff <+134>: lea 0x38(%esp),%ecx
0x08048703 <+138>: mov 0x20(%esp),%edx
0x08048707 <+142>: add %ecx,%edx
0x08048709 <+144>: mov %al,(%edx)
0x0804870b <+146>: addl $0x1,0x20(%esp)
0x08048710 <+151>: mov 0x20(%esp),%eax
0x08048714 <+155>: cmp 0x24(%esp),%eax
0x08048718 <+159>: jl 0x80486ef <main+118>
```

Antes de entrar del bucle se ve que en la posición 0x20(%esp) tenemos un 0 y será usada como iterador y en %eax se tiene el tamaño de la cadena -1 (con la instrucción sub para no modificar el \n que señala el fin de la cadena) que se introdujo y se almacena en 0x24(%esp) que se usa para la expresión de fin del bucle.

En la línea 155 junto a la 159 conforman expresión que comprueba que el iterador 0x20(%esp), al que se le suma 1 en la línea 146, almacenado en %eax es menor que el 0x24(%esp) (tamaño de la cadena -1) y si es así continúa con el bucle. Si se cumple eso salta al cuerpo del bucle.

Con la línea 118 trae de memoria la contraseña introducida.

Con la línea 122 eax tiene el valor del iterador.

Con las líneas de la 122 a la 128 nos quedamos con la letra que corresponde a la pasada del bucle.

Con la línea 101 se suma 1 a la letra correspondiente de esta pasada

Después de la línea 134 a la 142 se realiza el mismo proceso de antes pero en vez de cargar en edx la contraseña se almacena en ecx y en edx se carga la pasada actual del bucle. Para luego dejar en la posición de memoria que corresponde a la letra de esta pasada.

Con la línea 144 se añade la letra modificada a la contraseña.

Después se empiezan otra vez las comprobaciones de si el bucle ha terminado o no.

Con esto se ve por tanto que se está haciendo un algoritmo cesar a la contraseña introducida por el usuario en el que se cambia cada carácter de la contraseña por el siguiente carácter en la tabla ascii.

## Obtener contraseña

```
0x0804871a <+161>: movl    $0x804a040, (%esp)
0x08048721 <+168>: call   0x80484d0 <strlen@plt>
=> 0x08048726 <+173>: mov     %eax, 0x8(%esp)
```

La contraseña se pasa a strlen para ver su tamaño justo antes de la llamada colocando la en la pila.

El tamaño de la contraseña no sabremos también justo después de terminar la llamada a strlen y se almacena en eax.

Con estos dos datos podemos mostrar perfectamente la contraseña desde esa dirección y con el tamaño de esta con el siguiente comando.

```
(gdb) display $eax
2: $eax = 5
(gdb) x/5xc 0x804a040
0x804a040 <password>: 105 'i' 112 'p' 109 'm' 98 'b' 10 '\n'
(gdb) █
```

Solo nos quedaria mirar en la tabla ascii el valor de los caracteres anteriores de la contraseña. En nuestro caso corresponderia con hola.

## Compresión del mecanismo de encriptación para la pidcode y descifrado.

```
0x0804878c <+275>: call   0x80484f0 <__isoc99_scanf@plt>
0x08048791 <+280>: mov     0x1c(%esp), %eax
0x08048795 <+284>: add     $0x10, %eax
0x08048798 <+287>: mov     %eax, 0x1c(%esp)
0x0804879c <+291>: mov     0x1c(%esp), %edx
0x080487a0 <+295>: mov     0x804a048, %eax
=> 0x080487a5 <+300>: cmp     %eax, %edx
0x080487a7 <+302>: je      0x80487ae <main+309>
0x080487a9 <+304>: call   0x804860d <boom>
0x080487ae <+309>: movl    $0x0, 0x4(%esp)
0x080487b6 <+317>: lea     0x28(%esp), %eax
0x080487ba <+321>: mov     %eax, (%esp)
0x080487bd <+324>: call   0x8048480 <gettimeofday@plt>
0x080487c2 <+329>: mov     0x28(%esp), %edx
0x080487c6 <+333>: mov     0x30(%esp), %eax
0x080487ca <+337>: sub     %eax, %edx
0x080487cc <+339>: mov     %edx, %eax
0x080487ce <+341>: cmp     $0x3c, %eax
0x080487d1 <+344>: jle     0x80487d8 <main+351>
0x080487d3 <+346>: call   0x804860d <boom>
0x080487d8 <+351>: call   0x8048643 <defused>
0x080487dd <+356>: mov     0x9c(%esp), %ebx
0x080487e4 <+363>: xor     %gs:0x14, %ebx
0x080487eb <+370>: je      0x80487f2 <main+377>
0x080487ed <+372>: call   0x8048490 <__stack_chk_fail@plt>
0x080487f2 <+377>: mov     -0x4(%ebp), %ebx
0x080487f5 <+380>: leave
0x080487f6 <+381>: ret
End of assembler dump.
(gdb) display $edx
4: $edx = 7777
(gdb) █
```

Al pid code introducido por el usuario se le suma 0x10 que corresponde con 16 en la línea 284. Y teniendo en cuenta que en la línea 300 se comprara con %edx que almacena el pidcode de la bomba.

Por tanto para saber el pid code solo hace falta mostrar el registro edx y restarle 16. El pidcode seria 7761.

## Desactivar bomba.

Solo hay que cambiar las sentencia je por jmp con el programa ghx. Primero se busca con objdump -d y el nombre del ejecutable las lineas que aparecen con las cadenas del jn y se modifica la sentencia de je que el 74 por eb de esa forma salta siempre y no llama nunca a la función boom

```
8048740: 74 05          je      8048747 <main+0xce>
8048742: e8 c6 fe ff ff call    804860d <boom>

80487a7: 74 05          je      80487ae <main+0x135>
80487a9: e8 5f fe ff ff call    804860d <boom>
```