Empecemos iniciando el *gdb*. Tras poner nuestro *layout* empezaremos a ir pasando una tras una hasta llegar a esta parte concreta:

```
0x40120f <main+81>      callq  0x401060 <fgets@plt>
0x401214 <main+86>      test   %rax,%rax
0x401217 <main+89>      je     0x4011e8 <main+42>
```

Desde donde nos pide la contraseña. Ahí pondremos cualquier clave, ya que intentaremos hacerle un bypass. En mi caso voy a probar con *“hola/n”*.
Por el mismo sitio podemos ver esto de aquí:

```
0x40121e <main+96>      callq  0x4011b6 <codificarPass>
0x401223 <main+101>     mov    %al,0x30(%rsp)
0x401227 <main+105>     lea    0x30(%rsp),%rdi
0x40122c <main+110>     mov    $0x8,%edx
0x401231 <main+115>     lea    0x2e30(%rip),%rsi        # 0x404068 <password>
```

Vemos la contraseña y un método denominado codificarPass, o sea, que la contraseña podría estar alterada. Si imprimimos esa dirección de memoria, encontraríamos:

```
(gdb) p (char*) 0x404068
$1 = 0x404068 <password> "qwned!\n"
```

Si vas muy rápido podrías pensar “¡Oye, no ha sido difícil!”. Introduces el código y… ¡boom!. Ok, pues habrá que ver cómo se ha codificado esa clave, ¿no?
Desde el trozo de ASM anterior , si hacemos *`nexti/ni`* dos veces, vemos que en el registro %rax, el cual devuelve lo que se haya hecho en el método *codificarPass*. Es, de hecho, un número de 3 cifras. Podríamos interpretarlo como una letra, como un número, como una cadena…
Si intentamos imprimir ese registro con las diferentes opciones…

```
(gdb) p (int) $rax
$2 = 105
(gdb) p (char*) $rax
$3 = 0x69 <error: No se puede acceder a la memoria en la dirección 0x69>
(gdb) p (char) $rax
$4 = 105 'i'
```

Esa ‘i’ nos puede dar una pista… ¿Pero de dónde sale esa ‘i’?
En mi caso, puse para probar la contraseña “hola/n”. Puede tener sentido que la ‘i’ sea la ‘h’ del principio, pero lo han codificado con la letra siguiente. Pero claro, no todos tienen por qué caer en la cuenta. Vamos a seguir, a ver qué pasa…
Y llegamos al *strcmp*. Según el código en C, esta función usa tres parámetros: un string, otro string que se comparará, y el tamaño de la comparación en bytes. *Recordando el System V ABI de AMD y lo visto en el Tema 2.3 de Teoría de EC, o la regla mnenotécnica “Diane Silk Dress(x) Costs(x) 8-9”, estos parámetros estarían en %rdi, %rsi y %rdx*. Veamos esos registros (como cadenas de caracteres) :

```
(gdb) p (char*) $rdi
$5 = 0x7fffffffded0 "iola\n"
(gdb) p (char*) $rsi
$6 = 0x404068 <password> "qwned!\n"
(gdb) p (int) $rdx
$7 = 8
```

Un momento… ¿no habia introducido yo “hola/n”? ¿por qué ahora es “iola/n”? Hay que ver el código del *codificarPass*. Bueno, al menos tenemos la contraseña alterada, a ver si lo descifro.
Hacemos un *breakpoint* a *codificarPass* y quitamos el del main, de momento (también puedes desactivarla o dejarla).

El método posee este código:
```
0x4011b6 <codificarPass>        lea    0x1(%rdi),%eax
0x4011b9 <codificarPass+3>      retq
```
Traduciendo el código ASM a C nos daría algo como:
```
return ?? + 1 ;
```
Y podríamos ver que este método sólo tiene un parámetro. Volvamos al main,antes de codificarPass. En la zona de la llamada veríamos estas dos líneas:
```
0x401219 <main+91>      movsbl 0x30(%rsp),%edi
0x40121e <main+96>      callq  0x4011b6 <codificarPass>
```
Si me sitúo en la linea de la llamada, sin entrar en la función, e imprimo %edi, sale:
```
(gdb) p (char) $rdi
$3 = 104 'h'
(gdb) p (char*) $rdi
$4 = 0x48 <error: No se puede acceder a la memoria en la dirección 0x48>
```
Ya puede ir encajando todo…
1. Entra a codificarPass la letra ‘h’ de “hola/n”
2. codificarPass hace ‘h’ + 1 = 104+1 = 105 = ‘i’
3. Este método devuelve ‘i’
4. La contraseña ahora es “iola/n”

Entonces, la codificación es poner la letra siguiente al primer carácter de la cadena. Entonces, si lo que se compara es *“qwned!/n”*, la primera letra codificada es *‘q’*, a la cual, si le quito 1, es *‘p’*, y todo junto sale *“pwned!/n”*, siendo esa la contraseña.

Vamos con el pin. Empezaremos en main + 172, tras sortear las comprobaciones de contraseña. Ponemos un numero aleatorio, en mi caso, *2211*. Vamos a ver si hay otros métodos y a ver las comprobaciones:
```
0x4012ad <main+239>     cmp    $0x1,%ebx
0x4012b0 <main+242>     jne    0x40126a <main+172>
0x4012b2 <main+244>     mov    0xc(%rsp),%edi
0x4012b6 <main+248>     callq  0x4011ba <codificarPin>
0x4012bb <main+253>     mov    %eax,0xc(%rsp)
0x4012bf <main+257>     cmp    0x2d9b(%rip),%eax        # 0x404060 <passcode>
0x4012c5 <main+263>     je     0x4012cc <main+270>
0x4012c7 <main+265>     callq  0x401182 <desactivada>
```
Vemos el método *codificarPin* y el *passcode* en la posición indicada. Veamos el *passcode* y ahora pasamos a *codificarPin* (hagamos de paso un *breakpoint*)
codificarPin introduce el pin que hemos puesto en el método, y hace `lea 0x1(%rdi),%eax` , es decir, le suma 1. O sea, en mi caso, que era *2211*, ahora es *2212*. Al llegar a la comparación de pin, veremos que está el *passcode*.
Vamos a imprimirlo, examinarlo… A ver qué sacamos:
```
(gdb) p 0x404060
$1 = 4210784
(gdb) x/1i 0x404060
   0x404060 <passcode>: out    %al,$0x20
(gdb) x/1g 0x404060
0x404060 <passcode>:    0x00000000000020e6
```

Ergo, hay dos opciones, el largo (4210784) o el hexadecimal, el cual traducido es 8422.
Vista la codificación, la clave puede ser 4210783 u 8421. La correcta es la segunda.

Visto el proceso, las claves son.

Código : ***pwned!/n***
Pin: ***8421***

Si estás leyendo esto tras intentar resolverlo por tu cuenta , siento las molestias si te ha sido difícil de resolver, y espero que te esclarezca el proceso.