

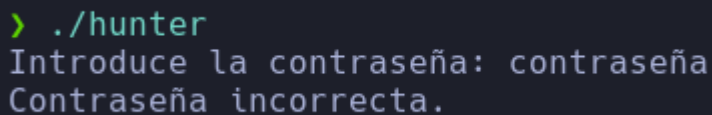
# Password Hunter Writeup

Para comenzar, podemos analizar la descripción que se nos proporciona para entender el reto:

*“Hey tu, si tu, el cazador, necesito que encuentres mi contraseña que se oculto en alguna sección de este binario, es un poco grande y tuvo que modificarse un poco para entrar, tu la reconocerás cuando la veas, la necesito ilesa, así que si esta herida o cambiada, necesito que le muestres tu atención, le gusta estar con su peluche, pero a el no lo necesito, solo quiero a mi contraseña”*

Primero podemos entender que necesitamos encontrar una contraseña que es un poco larga, que esta ligeramente modificada, por lo que no es la contraseña final, por lo que tenemos que modificarla para obtener la contraseña original, además se resalta una “H”, que por ahora no tiene sentido, así que podemos seguir con el reto

Ahora debemos ejecutar el binario para entender cual es su comportamiento



```
> ./hunter
Introduce la contraseña: contraseña
Contraseña incorrecta.
```

Parece ser un binario bastante sencillo, solo nos pide una contraseña, nada mas y por lo que sabemos, la contraseña esta escondida en algún lugar o “sección” del binario

Para empezar a analizar el binario, podemos buscar las cadenas de caracteres legibles que se encuentran en los binarios, para buscar los textos que se observan en la imagen. La sección donde se encuentran los datos legibles es **.data** o **.rodata**, como paso siguiente, podemos leer el contenido de esas secciones con la herramienta **strings** para identificar todos los datos legibles

(Imagen en la pagina siguiente)

```
> strings hunter
/lib64/ld-linux-x86-64.so.2
mgUa
fgets
stdin
puts
putchar
strcspn
__libc_start_main
__cxa_finalize
printf
strcmp
libc.so.6
GLIBC_2.34
GLIBC_2.2.5
_ITM_deregisterTMCloneTable
__gmon_start__
_ITM_registerTMCloneTable
PTE1
u+UH
Mi_pass-H
inseguraH
ra123( )
Flag:
Introduce la contrase
Contrase
a incorrecta.
;*3$"
GCC: (Debian 12.2.0-14) 12.2.0
Sqrt1.o
__abi_tag
crtstuff.c
deregister_tm_clones
__do_global_dtors_aux
completed.0
__do_global_dtors_aux_fini_array_entry
frame_dummy
__frame_dummy_init_array_entry
hunter.c
__FRAME_END__
_DYNAMIC
__GNU_EH_FRAME_HDR
_GLOBAL_OFFSET_TABLE_
putchar@GLIBC_2.2.5
__libc_start_main@GLIBC_2.34
_ITM_deregisterTMCloneTable
puts@GLIBC_2.2.5
stdin@GLIBC_2.2.5
_edata
_fini
printf@GLIBC_2.2.5
```

Aquí hay algunos datos interesantes como nombres de funciones, metadata como el compilador y la versión de OS, los mensajes que podemos ver al ejecutar el binario y una cadena bastante interesante, la cual es:

```
Mi_pass-H  
inseguraH  
ra123()
```

No parece ser la original pero es bastante legible, tiene saltos de linea, podemos quitarlos e intentar utilizar la contraseña para ver si el binario la acepta

```
> ./hunter  
Introduce la contraseña: Mi_pass-HinseguraHra123( )  
Contraseña incorrecta.
```

Al parecer esa no es la contraseña, pero si recordamos las instrucciones, tenemos que modificarla para que nos la acepte, si bien, en un principio la frase *“le gusta estar con su peluche, pero a el no lo necesito, solo quiero a mi contraseña”* no tenia sentido, ahora tiene un poco mas de sentido que la H este resaltada en la palabra “peluche”, vamos a intentar quitar las H’s

```
> ./hunter  
Introduce la contraseña: Mi_pass-insegurara123( )  
Contraseña incorrecta.
```

Parece que aun necesitamos modificar la contraseña, así que veamos, parece que la palabra “insegurara” no tiene mucho sentido, que tal si cambiamos a “insegura”? Quedando como contraseña: **Mi\_pass-insegura123()**

```
> ./hunter  
Introduce la contraseña: Mi_pass-insegura123( )  
Flag: ShyByte{N0_t0d0_Est4_0cuLt0}
```

Y efectivamente, esa es la contraseña, obteniendo la flag

Ahora vamos a analizar el código del binario para entender esta vulnerabilidad:

**(Imagen en la siguiente pagina)**

```

19 int main() {
20     char input[100];
21
22     // Contraseña almacenada como string
23     char correct_password[] = "Mi_pass-insegura123(";
24
25     printf("Introduce la contraseña: ");
26     fgets(input, sizeof(input), stdin);
27     input[strcspn(input, "\n")] = 0;
28
29     // Verificar si la contraseña es correcta
30     if (strcmp(input, correct_password) == 0) {
31         print_flag();
32     } else {
33         printf("Contraseña incorrecta.\n");
34     }
35
36     return 0;
37 }

```

En este pedazo de código podemos ver que la contraseña se almacena como un string en texto plano y posteriormente se compara el input del usuario con el string que contiene la contraseña en texto plano, al ser texto legible, todo esto se almacena en la sección `.data` del binario que como ya había dicho anteriormente, almacena texto plano y legible, por lo que almacenar contraseñas y compararlas de esta manera en un binario es altamente peligroso

Por si te lo preguntabas, así esta almacenada la flag para evitar que se almacene en `.data` y que no sea posible verla utilizando el comando **strings**

```

1 void print_flag() {
2     // Flag almacenada en formato ASCII para evitar su almacenamiento como string
3     int flag[] = {83, 104, 121, 66, 121, 116, 101, 123, 78, 48, 95, 116, 48, 100, 48, 95, 69, 115, 116, 52, 95, 48, 99, 117, 76, 116, 79, 125}; // "ShyByte[N0_t0d0_Est4_0culT0]"
4
5     // Calcular el tamaño de la flag
6     size_t flag_size = sizeof(flag) / sizeof(flag[0]);
7
8     // Imprimir la flag
9     printf("Flag: ");
10    for (size_t i = 0; i < flag_size; i++) {
11        printf("%c", flag[i]);
12    }
13    printf("\n");
14 }

```

Se almacena como números correspondientes a caracteres ASCII, para luego decodificarlos e imprimirlos bastante ingenioso, verdad?

Aquí tienes el código completo del binario por si quieres analizarlo a profundidad

> cat hunter.c

File: hunter.c

```
1  #include <stdio.h>
2  #include <string.h>
3
4  void print_flag() {
5      // Flag almacenada en formato ASCII para evitar su almacenamiento como string
6      int flag[] = {83, 104, 121, 66, 121, 116, 101, 123, 78, 48, 95, 116, 48, 100, 48, 95, 69, 115, 116, 52, 95, 48, 99, 117, 76, 116, 79, 125}; // "ShyByte{N0_t0d0_Est4_0cuLt0}"
7
8      // Calcular el tamaño de la flag
9      size_t flag_size = sizeof(flag) / sizeof(flag[0]);
10
11     // Imprimir la flag
12     printf("Flag: ");
13     for (size_t i = 0; i < flag_size; i++) {
14         printf("%c", flag[i]);
15     }
16     printf("\n");
17 }
18
19 int main() {
20     char input[100];
21
22     // Contraseña almacenada como string
23     char correct_password[] = "Mi_pass-insegura123(";
24
25     printf("Introduce la contraseña: ");
26     fgets(input, sizeof(input), stdin);
27     input[strcspn(input, "\n")] = 0;
28
29     // Verificar si la contraseña es correcta
30     if (strcmp(input, correct_password) == 0) {
31         print_flag();
32     } else {
33         printf("Contraseña incorrecta.\n");
34     }
35     return 0;
36 }
37
```

