ThirstyGhost

En este reto se nos presenta un escenario donde se nos pregunta cuanta agua debemos darle a un fantasma, ¿que tiene que ver esto con el binario que tenemos que ejecutar? Bueno, vamos a ejecutar el archivo para ver su comportamiento

Parece que el comportamiento del programa es bastante simple, solo se nos pide un input donde se nos pregunta cuanta agua le queremos dar al fantasma, sin embargo no sabemos que es lo que espera el programa,¿debemos ingresar números, una cantidad en litros o mililitros? Vamos a analizar el código del binario para descubrirlo

```c
void flag() {
    printf("EL vaso del fantasme se derramo\n");
    printf("flag: ShyByte{D3sb0rd4r_3l_Buff3r_es_d1v3rt1d0}\n");
    exit(0);  // Salir después de imprimir la flag
}

void llenar_vaso() {
    char buff[32]; // Buffer de 32 bytes
    int valido = 1337; // Declaramos un numero entero

    printf("\n¿Cuanta agua le quieres poner al vaso?");
    printf("\n ▼ Vacio ---------- Lleno ▼\n");
    printf(">");

    gets(buff); // Funcion gets vulerable

    // Comparar si 'valido' es diferente de 1337
    if(valido != 1337)
    flag();

    printf("El fantasma esta sediento, dale mas agua\n");

    return 0;
}

int main() {
    printf("%s", ghost);
    llenar_vaso();
    return 0;
}
```

Lo primero que podemos ver es que la función **flag()** imprime la flag, te ahorro la decepción, no la vas a encontrar usando el comando **strings**, al menos no en el binario que se proporciono para probar en el reto

Lo siguiente que podemos ver es que no hay ninguna comprobación de que es lo que se recibe en el input, por lo que podemos ingresar cualquier cosa, pero si lo analizamos detalladamente, podemos darnos cuenta que hay una función **gets** que obtiene **buff**, donde buff contiene el input del usuario, que puede almacenar hasta 32 bytes, si realizas una búsqueda rápida en internet, podrás encontrar que la función gets es peligrosa, ya que no hace una comprobación del tamaño del buffer que recibe, esto es peligroso ya que al recibir mas bytes de los que el buffer puede contener, se empiezan a sobrescribir los siguientes valores declarados y posteriormente las direcciones de memoria, esto es conocido como un **buffer overflow**

Sabiendo esto, podemos ver que el siguiente valor declarado después de **buff**, es "**valido"**, donde su valor es 1337, veamos que es lo que se hace con este valor

```
// Comparar si 'valido' es diferente de 1337
if(valido != 1337)
flag();

printf("El fantasma esta sediento, dale mas agua\n");
```

Estas 3 lineas son bastantes simples, únicamente se comprueba si "**valido**" es diferente a **1337**, que es el valor con el que se declaro, si el valor es diferente, se ejecuta la función **flag()**, si el valor es igual, el programa continua su ejecución y te indica que el fantasma sigue con sed

Con esta información, sabemos que hay un buffer overflow que permite sobre escribir el valor de la variable declarada después de **buff**, la cual es "valido" y si logramos hacer que "**valido**" sea diferente de 1337, podemos lograr la ejecución de la función **flag()** que nos da la bandera

Para explotar esta vulnerabilidad, tenemos que sobrepasar el valor de buff, el cual solo puede almacenar 32 bytes, a partir del byte 33 que le pasemos, se va a sobre escribir "**valido**" y si se sobrescribe, sera diferente de 1337, vamos a intentar esto

```
¿Cuanta agua le quieres poner al vaso?
 ▼ Vacio ------------------------- Lleno ▼
>AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
El fantasma esta sediento, dale mas agua
```

Al parecer esto no funciono, la explicación técnica es que al momento de compilar el programa, el compilador puede agregar bytes adicionales para alinear la memoria, también al tener mas variables declaradas como “**valido**”, ocupan espacio en la pila, también conocida como **stack**, por lo que se necesitan mas bytes para poder determinar donde empiezan a sobrescribirse, el punto donde se empiezan a sobrescribir se conoce como **offset**. En general, este concepto es un poco complejo, así que te recomiendo buscar información en internet, ya que este es un reto fácil de resolver, pero para entenderlo completamente, puedes buscar información mas detallada

¿Como podemos determinar el **offset**?
SI te fijas en el mensaje que muestra el binario, podemos ver que se imprime un mensaje que aparentemente parece ser una guía visual de que tantos caracteres tenemos que poner, podemos identificar esto ya que curiosamente hay una flecha indicando donde esta vacío el **stack**, y hay otra flecha mas alejada, intentemos escribir caracteres hasta alcanzar esa flecha

```
¿Cuanta agua le quieres poner al vaso?
 ▼ Vacio --------------------------- Lleno ▼
>AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
El fantasma esta sediento, dale mas agua
```

Parece que aun no, vamos a intentar hacerle caso al ultimo mensaje que nos incida que tenemos que darle mas agua al fantasma o visto de una manera mas literal, agregar mas bytes

```
¿Cuanta agua le quieres poner al vaso?
 ▼ Vacio --------------------------- Lleno ▼
>AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
EL vaso del fantasme se derramo
flag: ShyByte{D3sb0rd4r_3l_Buff3r_es_d1v3rt1d0}
```

Parece que “**valido**” se empieza a sobrescribir un byte después de lo indicado, lo que indica que el **offset** es de 43 bytes y “valido” se empieza a sobrescribir en el byte 44
Existe la posibilidad de que el mensaje se haya desfasado y no coincidan las flechas con los bytes, resolver esto es tan sencillo como contar los caracteres que hay desde la primera flecha hasta la ultima incluyendo los espacios, ya que es igual a los 43 bytes de offset que tenemos.

En otro tipo de retos mas avanzados, determinar el offset es diferente ya que lo mas probable es que no se nos proporcione una guía visual, determinar el offset es lo primero que se debe realizar para explotar un buffer overflow, hay diferentes manera de determinarlo, puede ser ingresar bytes hasta que el programa crashee, debugear el binario hasta encontrar una dirección sobrescrita o simplemente leer el código para ver cuantos bytes como mínimo se necesitan