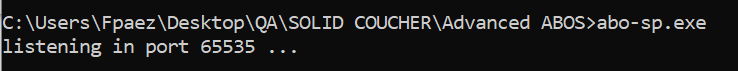
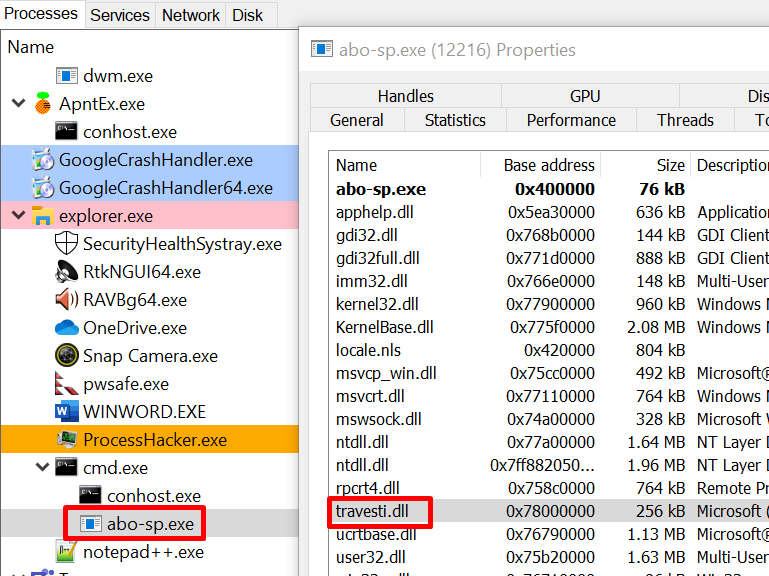
Ponemos a correr el abo



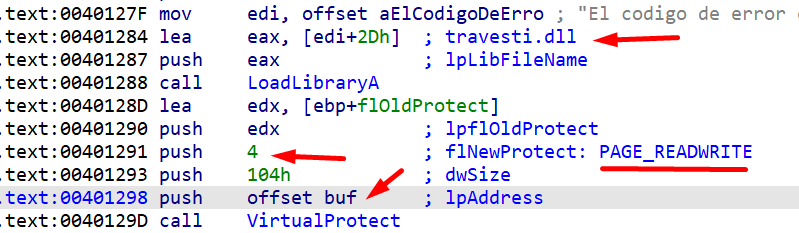
Si lo vemos en proccess hacker nos daremos cuenta de que también carga la librería trapecio.dll



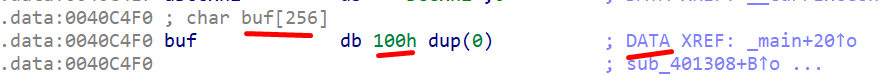
Trapecio.dll

Cargamos el abo en el Ida y vamos al main.

Podemos ver que carga una librería trapecio.dll, y luego a la variable global **buff** le da permisos de solo lectura y escritura, es decir no tendrá permisos de ejecución.



Si hacemos doble clic en la variable **buf** nos daremos cuenta que se encuentra en la sección **data**, y que esta variable **buff** tiene un tamaño de **256 bytes,** por ende es una **variable global**, lo que significa que el espacio de memoria ya no es asignado por medio de la pila local, ósea en el ámbito de la función. Es decir el acceso a esta variable es directamente desde la memoria del proceso.



Okey, bueno vamos al grano, busquemos la función **recv**

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Ahí estamos, como se ve en la imagen se recibe la data en una variable llamada **buff**, y la cantidad de bytes que se leyó con la función **recv** es retornado en el registro **eax** y almacenado en una variable a la que llamare **bytes\_leidos.**

Ahora veamos desde donde es llamado este bloque.

Texto

Descripción generada automáticamente

Es llamado desde dos lugares, analicemos el primero.

Viene de este bloque que comienza con un **memset** inicializando el **buffer** en **CERO**.

Imagen que contiene Tabla

Descripción generada automáticamente

Aquí ya noto algo extraño, primero el tamaño del **buffer** era de **256 bytes,** y como vemos aquí se esta inicializando **260 bytes.** Aquí abajo se ve como son inicializados, los 260 bytes a cero.

Diagrama

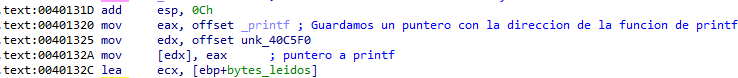
Descripción generada automáticamente con confianza baja

Luego de pasar el **recv** estos 260 bytes serán reemplazados por la data que se reciba, en este caso le mande muchas Aes.

Tabla

Descripción generada automáticamente

Okey, un puntero a la función **printf** es guardado inmediatamente después del **buffer de 256 bytes,** mas estos **4 bytes** del puntero obtenemos los **260 bytes totales.** Esto ocurre antes de que se ejecute el **recv.**



¡¡¡¡Entonces, al ejecutarse la función recv estos 4 bytes serán pisados!!!!

Texto

Descripción generada automáticamente con confianza media

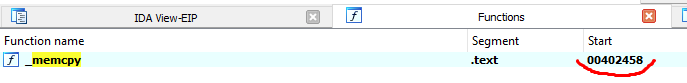
Podemos manipular estos 4 bytes colocando ahí una dirección cualquiera a la que saltará con el **call eax.**

Como vimos al comienzo, la variable global **buffer** NO TIENE PERMISOS DE EJECUCION, es por esto por lo que no podemos saltar ahí, ya que no se ejecutara.

Podemos buscar **gadgets** en la librería **trapecio.dll,** pero se me ocurrió algo mas fácil :D

Una forma de evitar esto es copiando el contenido de la variable **buffer** a otro lugar en donde se pueda ejecutar código.

Para hacer esto podemos usar a api **memcpy** que se encuentra en **0x402458**



Dicha api tiene 3 argumentos, **destino, source, y el tamaño** de lo que se copiara.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Ya cuando llegamos al **call eax** podemos ver en la pila que nos queda afortunadamente así:

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Usaremos a **0x40A115** como el **Destino,** en **0x40C4F0** esta el **buffer** que será el **source**, y **0x104** es el **tamaño,** que son los **260 bytes.**

Entonces el plan es el siguiente, una vez que llegue al **call eax,** este saltara a la api **memcpy** y nos copiara el contenido de la variable **buffer** en el **destino**.

Toda esa primera parte lo hacemos enviando un paquete que hará todo eso, después necesitamos enviar otro paquete, pero esta vez en vez de saltar a **memcpy**, el **call eax** saltará al **destino** en donde copiamos anteriormente el contenido del **buffer.** De esta forma se ejecutará la calculadora :P

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media