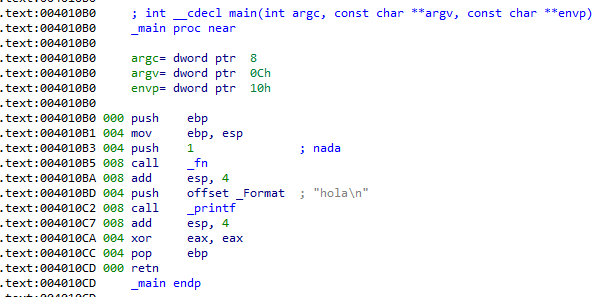
Se me había pasado por alto este **ABO1\_NX**, así que lo vamos hacer.

Antes que nada hay ver que significa el **NX**, segun la wiki:

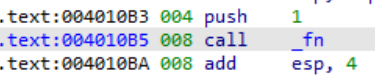
* <https://es.wikipedia.org/wiki/Bit_NX>
* <https://en.wikipedia.org/wiki/Executable_space_protection>

| *NX significa No eXecute (no ejecutar). Es una tecnología utilizada en CPUs para separar las áreas de memoria usadas para albergar las instrucciones del procesador (código) y las de almacenamiento de datos. Cualquier sección de memoria designada con el atributo NX significa que sólo se usa para datos, por lo tanto las instrucciones del procesador no pueden ni deben residir ahí. Es una técnica utilizada para prevenir que cierto tipo de software malicioso tome el control de la máquina insertando su código en el área de almacenamiento de datos de otro programa y ejecute su propio código desde dentro de esta sección; esto se conoce como desbordamiento de búfer, y NX puede prevenirlo.* |
| --- |

Vamos a abrirlo en el **IDA.**

****

Okey, vemos una función **\_fn** que recibe como parámetro el entero **1.**

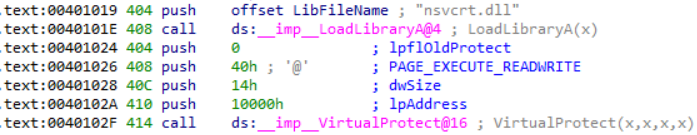
****

Dentro de esta función **fn** vemos una variable local **buf,** más el argumento **arg\_1** que se le paso por parámetro.

**buf** tiene un tamaño de **0x400 (1024) bytes**

****

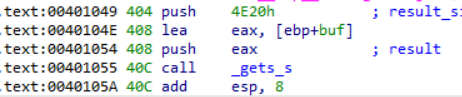
Se carga una librería **nsvcrt.dll** la cual viene en conjunto con el **abo,** y le da permisos de ejecución, lectura y escritura con la función **VirtualProtect.**

****

vemos que hay un función **gets\_s** que es una versión más segura que la **gets.**

| char \*gets\_s(  char \*buffer,  size\_t sizeInCharacters ); |
| --- |

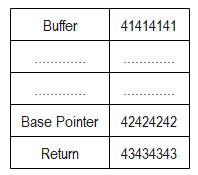
Bien, ahora se debe de pasar el tamaño del buffer. En este caso se le pasa **0x4E20** como tamaño del **buffer.**



**Metodo de Resolucion 1:**

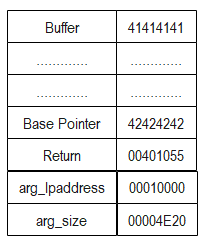
Debido a que tenemos **DEP** con el bit **NX** activado, el código en la pila no va a tener permiso de ejecución.

Lo primero que tenemos que hacer es pisar el **RETN,** esto lo podemos lograr desbordando la variable **buffer** que se llenará con la función **gets.**



¿ahora a dónde saltamos?¿ Pues podemos saltar al **call gets\_s,** y pasarle por argumento la dirección **0x10000** que es la que se da permiso de ejecución con el **VirtualProtect**

Muy bien, cambiemos el **43434343** por **0x10000**

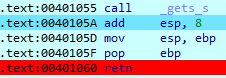


También le mandamos el **size** como argumento, ya que la función **gets\_s** lleva esos dos argumentos, el **size** puse el mismo que ya tenía, **0x4E20.** Hasta aquí toda esta cadena se las enviamos al **primer gets\_s**, por lo que debemos finalizarlo con un **“\n”**.

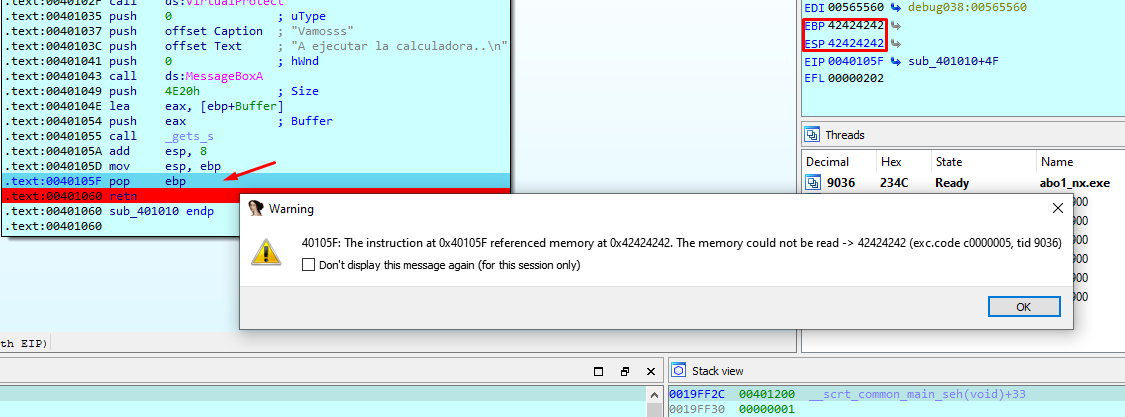
Si vamos traceando por el **epílogo**  en un momento va a recuperar el **viejo base pointer**, en este caso es **0x42424242**, y luego retorna a la dirección que le indicamos, en este caso a **0x401055** que corresponde al **call gets.**

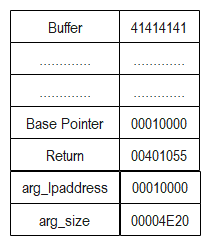
****

Okey, una vez pasemos el **ret** estaremos parados encima del **call gets\_s**, vamos traceando con **F8**

****

Recordemos que ahora **EBP** vale **0x42424242,** y cuando lleguemos nuevamente a **40105D** este valor se pasará al **ESP.** Al ejecutar el **POP EBP** me aparece este error :P

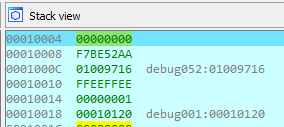
****



Vamos a solucionar esto poniendo un valor real en **ebp,** voy a intentar haciendo que **ebp** sea igual a **0x00010000**, que era la dirección al que se le seteaba los permisos de ejecución con el **VirtualProtect.**

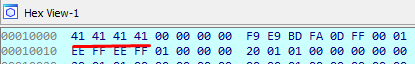
Para que esto suceda, deberemos hacer que el **viejo base pointer** valga **0x00010000.**

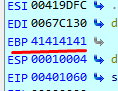
Okey, ya está.., no nos sale más el error al pasar el **pop ebp.** Y lo mejor es que la pila nos queda apuntando a **0x00010000**

****

Recordemos que al final de esta cadena que le pasamos al **primer gets\_s** lo finalizamos con un **“\n”**, así de esa forma lo que escribamos después de este “**\n”** lo tomara como argumento el **segundo gets\_s,**  y escribirá el contenido en la dirección **0x00010000**

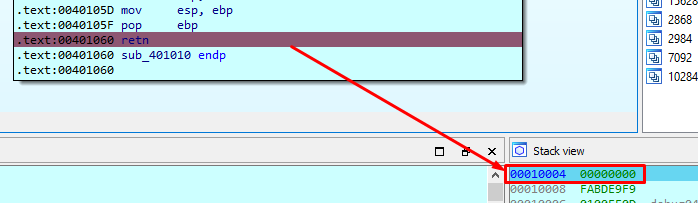
Okey, ahora cuando lleguemos por segunda vez al **retn** y si miramos en el dump **0x00010000** tendremos la segunda parte de la cadena que le pasamos despues del “**\n”,** en estes caso le pase “AAAA”



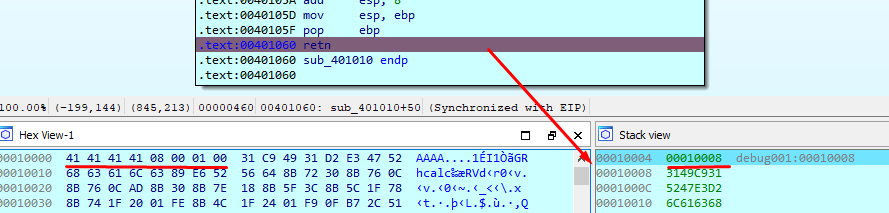


Casualmente **0x41414141** se vuelve a mover a **EBP,** pero por el momento, no nos interesa.

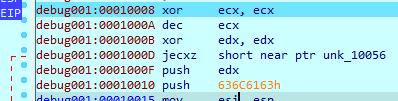
Vemos que estando parados en el **retn** está por retornar al contenido del segundo **dword** de la cadena que le pasamos. Podemos aprovechar para saltar a ejecutar desde el tercer **dword**, y ya que ejecute la **calculadora.** Desde ya estoy hablando de la cadena que va después del **“\n”**, osea la segunda parte de la cadena -.- .



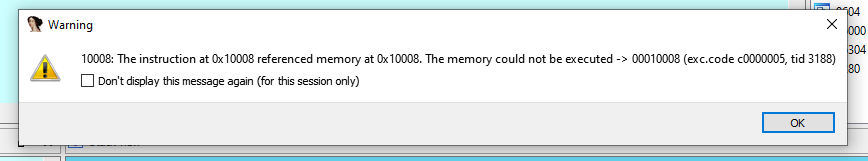
Nos quedara asi, entonces cuando salte a ejecutar **0x00010008** ejecutara el codigo de la calculadora.

****

Aqui estamos en el comienzo de la **shellcode** que ejecutara la calculadora

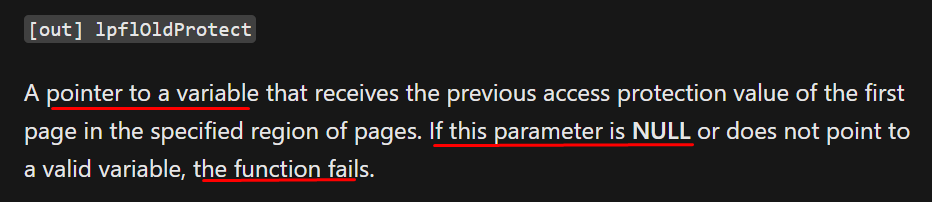
****

El problema es que al parecer por más que **VirtualProtect** le dio permisos de ejecución, el muy piutho sigue sin permisos!!! -.-

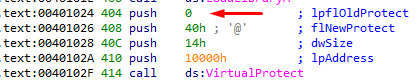


**¿Cual es el problema?¿**

Veamos la descripción de **VirtualProtect**

****

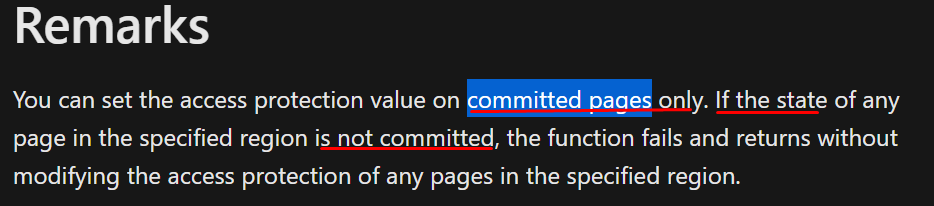
Este parámetro debe ser un puntero a una variable, y si ese parámetro es 0 (NULL) la función fallará.



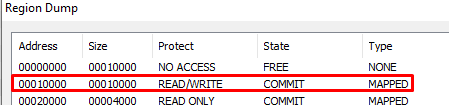
Se esta pusheando un cero, nos metieron una trampita, por eso esta fallando!! -.-

Bueno después de un par de pruebas cambiando la dirección de **lpflOldProtect** a **0x10000**, no funciono, siempre me retorna un cero en eax -.-.

Quizás pueda ser por esto que dice en la descripción de **VirtualProtect**

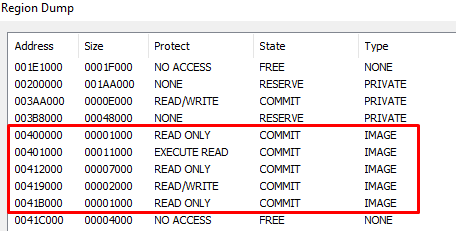


Parado en el **call a VirtualProtect** con un Editor PE podemos ver la región **0x10000** y su protección, se ve que es **READ/WRITE,** y después de pasar el call, sigue igual. Y además vemos que el estado es de **COMMIT**, así que bueno que se yo, lo que dice en la descripción no se estaría cumpliendo. -.-

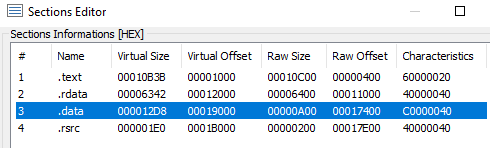


Okey, que la chupe la dirección **0x10000**, vamos a probar con otra diferente.

Tenemos estas que son de tipo **IMAGE,**



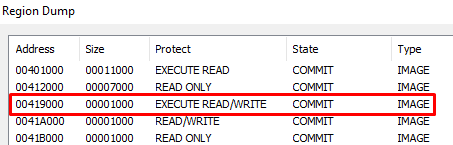
Voy a elegir la **0x419000** que es la sección .data



Okey, bueno… en vez de saltar el **get\_s** vamos hacer que salte al **call VirtualProtect**.

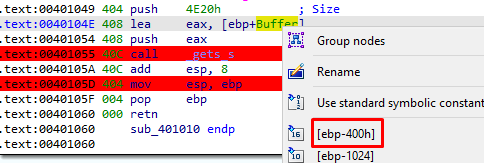


Ok, para que funcionara y me retorne **eax** distinto a **cero**, lo que hice fue setear **lpflOldProctet y lpAddress** en **0x00419000,** ahora si permite ejecución.



Ahora tenemos que hacer que se copie la **shellcode de la calculadora** en esa dirección **0x00419000**, y vamos a usar el **gets\_s** para esto. Como ya comentamos esta sería la **segunda ejecución del gets\_s** por lo tanto tendremos que escribir el resto del **payload** después del **“\n”.**

Al llegar por segunda vez al **gets\_s** en **ebp-0x400** está la dirección que se le pasa como argumento para el **buffer,** tendremos que setear **ebp** para que apunte a **ebp+0x400.** Pondremos en el **ebp** la dirección **0x00419400**, que es la sección data que le acabamos de dar permiso de ejecución, lectura y escritura. Esto lo logramos seteando en el **viejo base pointer.**

****

Okey, en este **buffer** meteremos el **shellcode de la calculadora**.

Bien, estamos por llegar por segunda vez al **retn**, en este punto tenemos que saltar a **0x00419000** para ejecutar la calculadora.