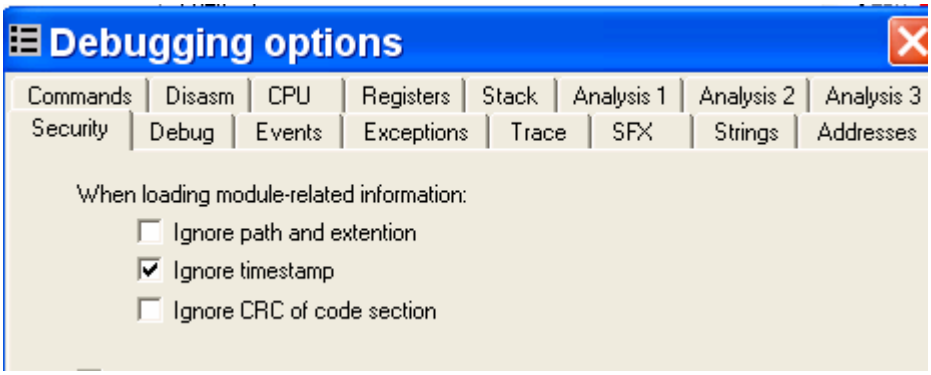
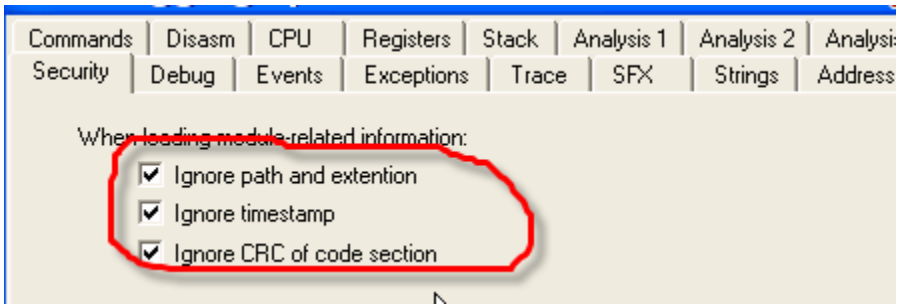
INTRODUCCION AL CRACKING CON OLLYDBG PARTE 20

El siguiente truco ANTIOLLYDBG que veremos en esta oportunidad es la detección del mismo, por el nombre del proceso del OLLYDBG, pero antes de todo les diré una configuración del OLLYDBG que aun no hemos tocado y que es muy importante en este caso.

Si vamos a DEBUGGING OPTIONS-SECURITY



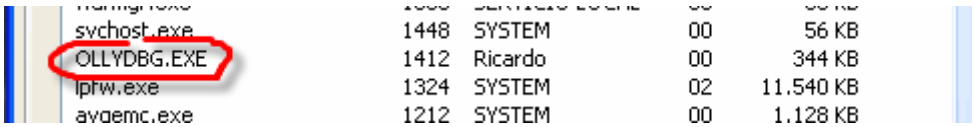
Allí lo mejor es poner las tres tildes y que cargue siempre la información, verán que con las tres tildes puestas cuando pongan BPX en una api y reinicien el OLLYDBG, el BPX continuara puesto, lo cual evita tener que repetir el proceso de ponerlos uno a uno cada vez que reiniciamos.
Estrictamente no conozco el funcionamiento interno completo del OLLYDBG pero en la practica lo que sucede es eso con las tres tildes, los BPX en las apis se mantienen luego de reiniciarlo.



Así esta mejor para mi gusto, es menos molesto, ahora si, comenzamos con el tema de antidebugging por el nombre del proceso.

Deteccion del OLLYDBG por el nombre del proceso

Cuando corremos el OLLYDBG si miramos la lista de procesos con CTRL + ALT + SUPR

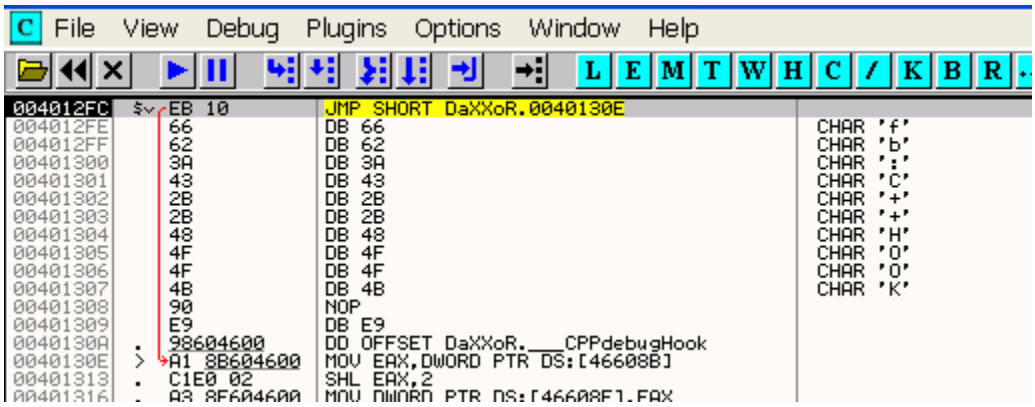


Vemos que el nombre del proceso esta allí muy claro en la lista de los mismos, que le impide a un programa revisar todos los procesos y si encuentra alguno que se llame OLLYDBG, cerrarlo, pues nada, jeje.

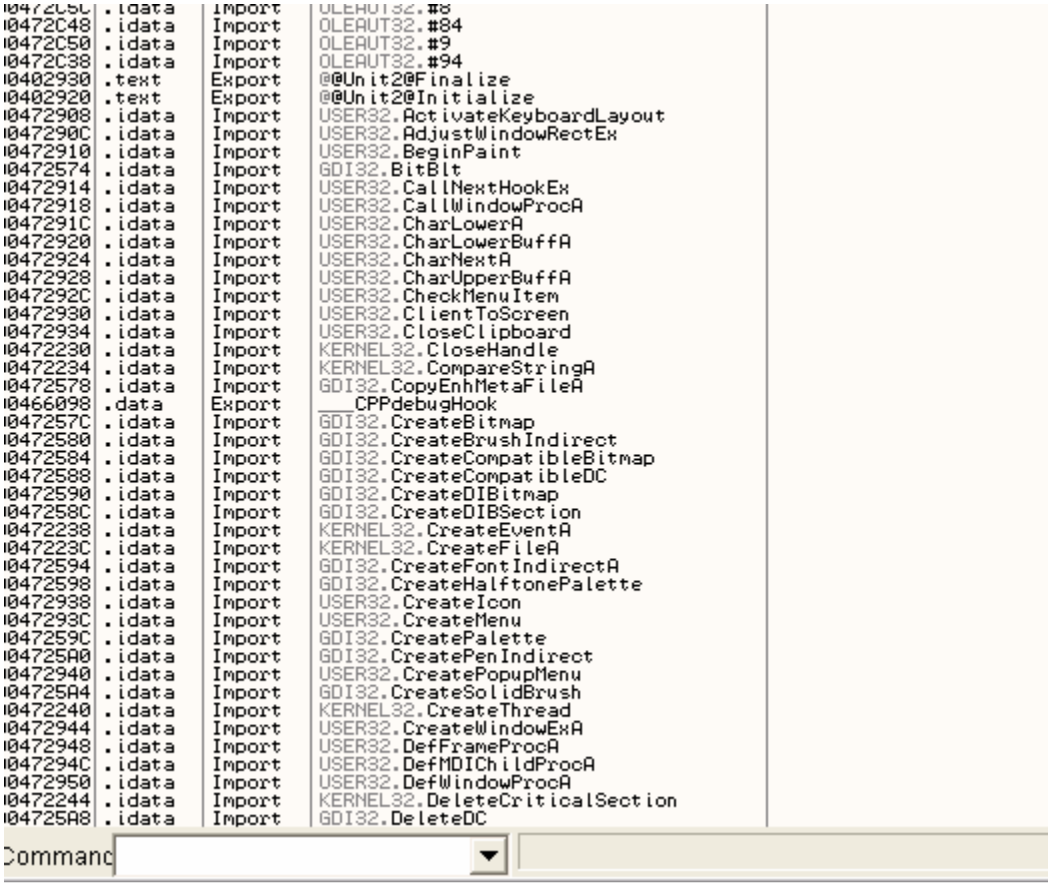
Usaremos un crackme que no resolveremos por ahora porque es un nivel levemente superior al actual, así que lo veremos mas adelante, pero estudiaremos en el la técnica de cómo detectan los programas a OLLYDBG por el nombre del proceso, como evitarlo manualmente y como evitarlo definitivamente jeje.

El crackme que adjunto es el DAXXOR el cual si dejamos un OLLYDBG corriendo vacío y corremos fuera de OLLY el famoso DAXXOR, veremos que el crackme corre pero cierra el OLLYDBG, lo mismo que si lo corremos en OLLYDBG, también lo cierra y por supuesto se acaba todo.

Estudiemos como hace esto, abrámoslo en OLLYDBG.

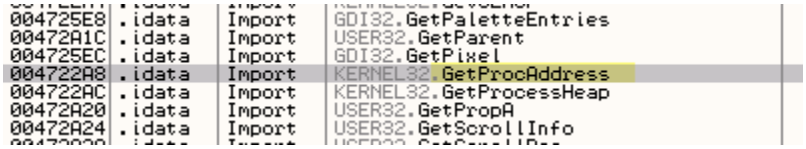


Allí esta en el ENTRY POINT, y veamos las apis que utiliza.

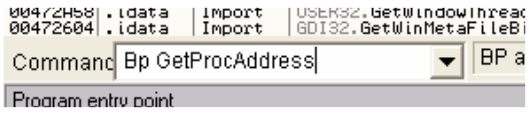


Bueno hay unas cuantas, pero aquí hay otra protección agregada y es que el crackme no tiene cargadas en el inicio las apis que usara para detectar al OLLYDBG, y las cargara a medida que corra, de paso explicaremos también este método de protección que en si, no te permite ver todas las apis que usara, en la lista de NAMES, pero por otra lado tiene como contrapartida que el que se da cuenta del truco y descubre las apis que va cargando el programa a medida que corre, sabrá a ciencia cierta que esas apis son las IMPORTANTES, pues por algo el programa las oculta para que no aparezcan en la lista.

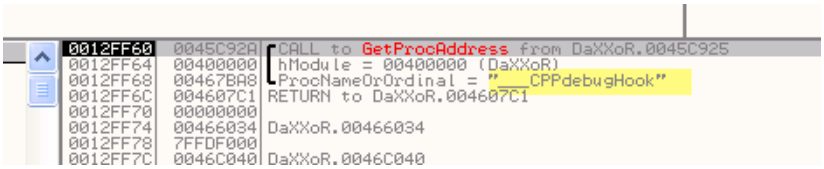
Esto casi siempre es evidente cuando un programa que no esta empacado como en este caso, hace uso de la api GetProcAddress



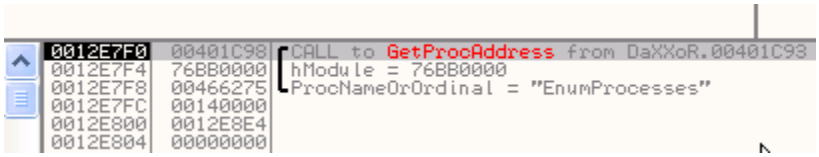
GetProcAddress se utiliza, para que el programa cargue nuevas apis, que no están en la lista para poder usarlas, ya veremos un uso intensivo y mas detallado de esta api en el capitulo de desempacado, pero por ahora pongamos un BP en dicha api.



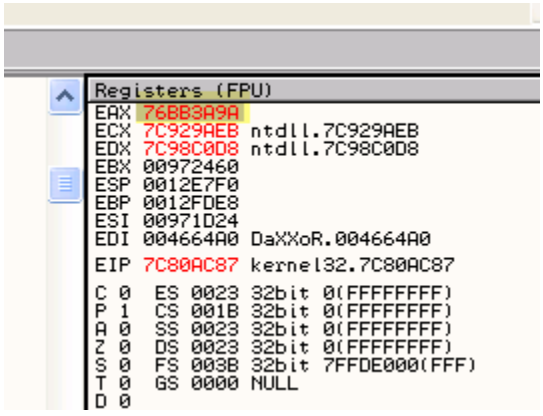
Y demos RUN



Cada vez que para vemos que esta pidiendo por medio de la api GetProcAddress, la dirección en nuestra maquina de una determinada api la primera que pide es en este caso, __CPPdebugHook que no pertenece al truco que estamos estudiando por lo cual damos RUN nuevamente.

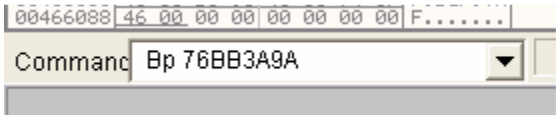


Así vamos pasando con F9 hasta que encontremos apis relacionadas con el truco, aquí vemos la api EnumProcesses que es usada, así que lo que hacemos es llegar hasta el RET de GetProcAddress la cual nos devuelve en EAX la dirección de la api solicitada en nuestra maquina y allí le ponemos un BPX, veamos, lleguemos al RET con EXECUTE TILL RETURN.

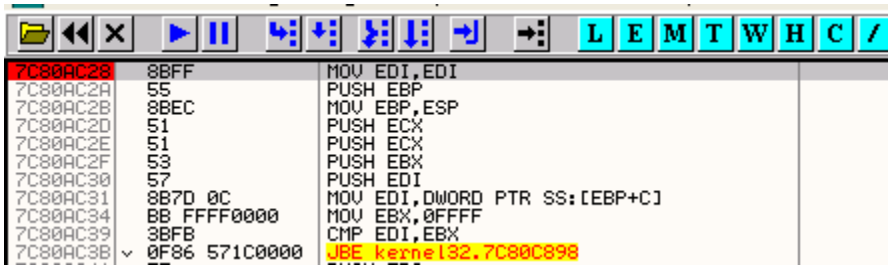


Allí en EAX devuelve la dirección en mi maquina de la api solicitada, en mi caso es 76BB3A9A en sus maquinas puede variar.

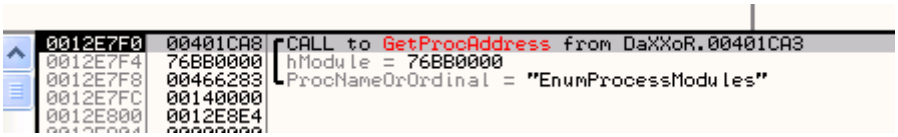
Por otro lado veo que OLLYDBG no identifica la api que no esta en la lista de NAMES, por lo cual no se puede poner BP directos al nombre de la api, si no que hay que poner el BP en la dirección.



Ahora si lo toma

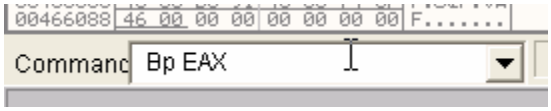


Ya tenemos puesto el BP en la api sospechosa, sigamos corriendo OLLYDBG, para ver si carga mas apis.

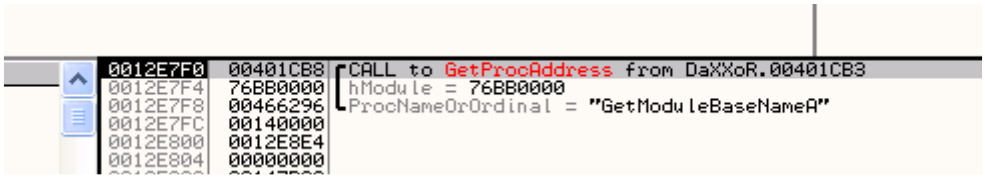


Hmm enumera los módulos de un proceso, hmm repito el procedimiento anterior llego al RET y le pongo un BPX a la dirección que me muestra EAX.

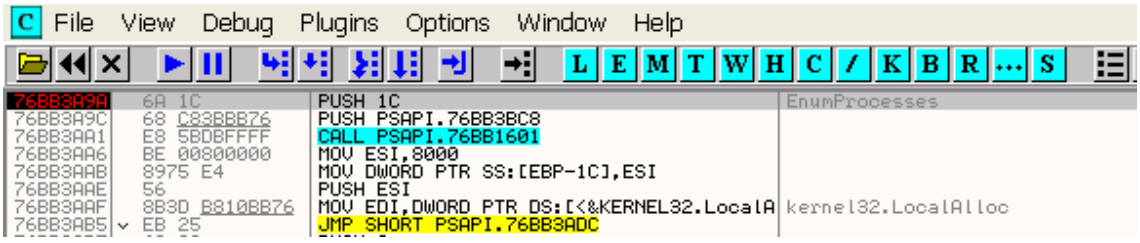
Para no tipear mucho si estoy en el RET directamente pongo



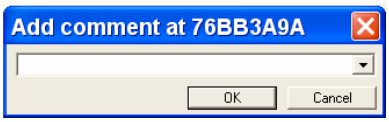
Que me servirá para colocarle BPX a todas las apis cuando este en el RET .

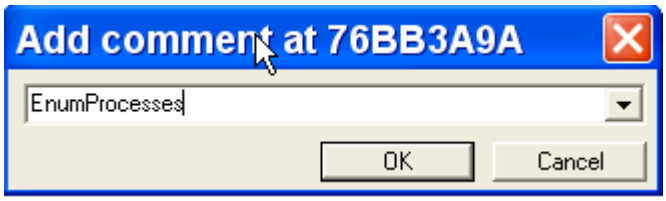


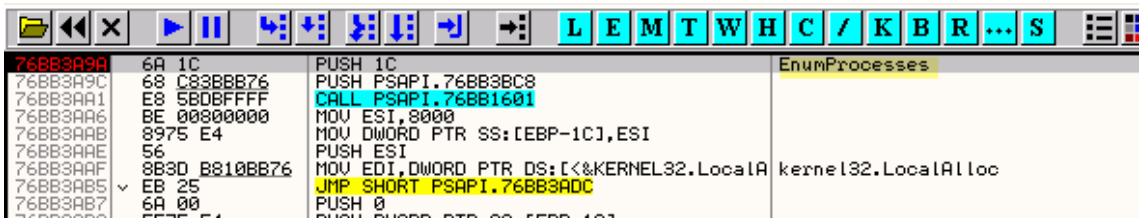
Otra sospechosa le pongo BP de la misma forma que a las anteriores, y doy RUN y para en EnumProcesses



El comentario a la derecha con el nombre de la api se lo agregue yo, haciendo doble click en esa zona nos permite agregar un comentario.

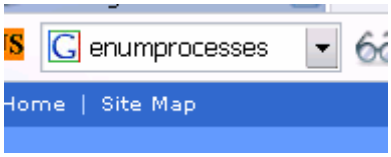






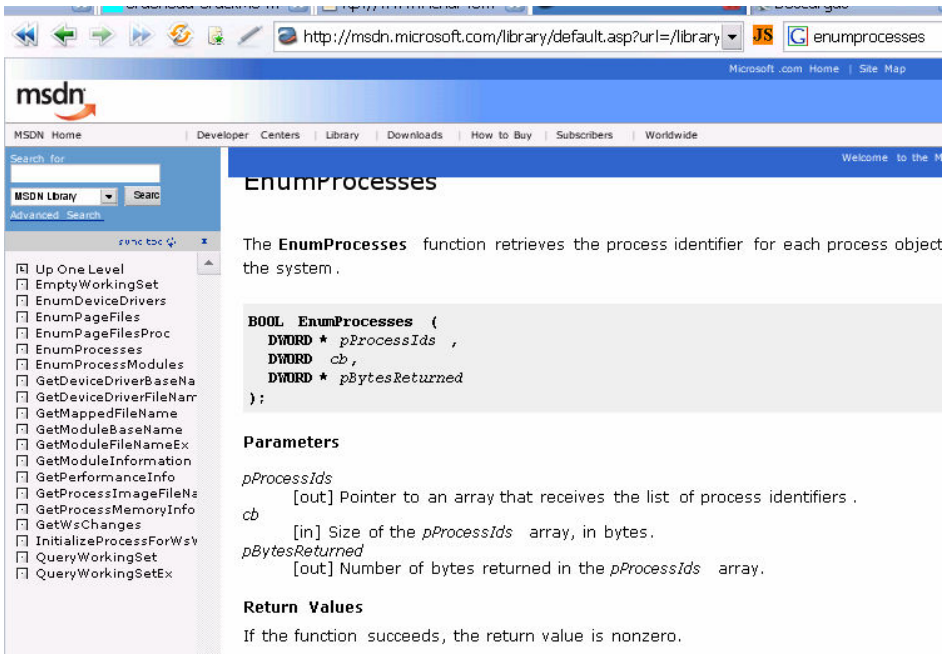
Esto lo realizo en todas las apis que el programa va cargando y veo sospechosas cuando le pongo un BPX cosa de que cuando pare, sepa que api era, pues el OLLYDBG no me aclarara nada al no ser una api de las de la lista.

Si busco en el WINAPIS32 esta api no figura, por lo tanto busco en GOOGLE



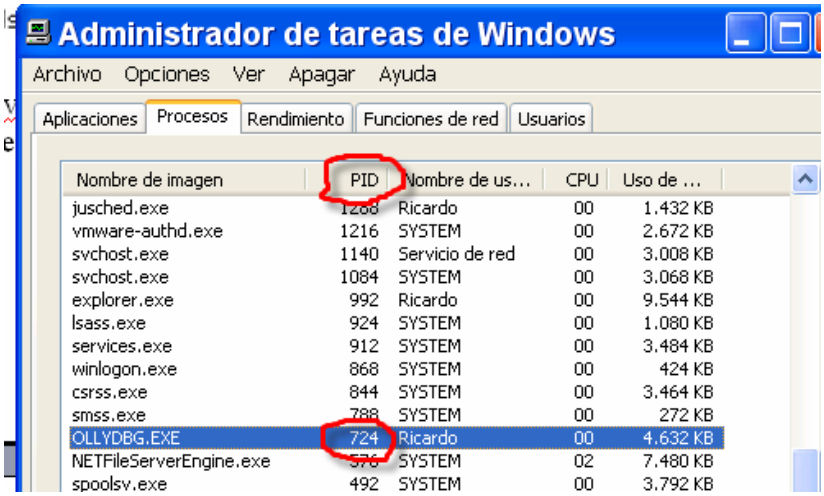
Y en la pagina de Microsoft normalmente se encuentran, es este caso la pagina es

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/perfmon/base/enumprocesses.asp

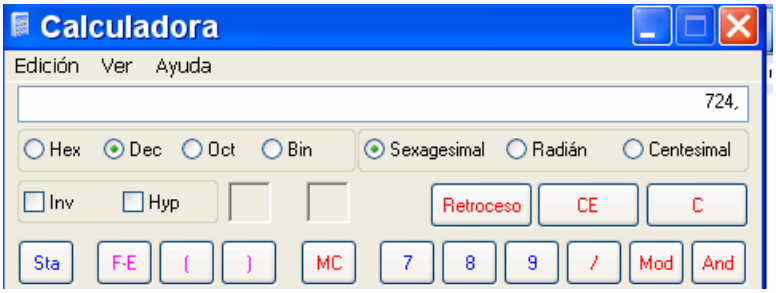


Bueno aquí dice que la susodicha api, nos devolverá el PROCESS IDENTIFIER o PID de cada proceso que esta corriendo, pues bien veamos antes que es el PID ese, jeje.

Cada proceso que corre esta identificado con un número que varia cada vez que se arranca un proceso, si vemos en la lista de procesos



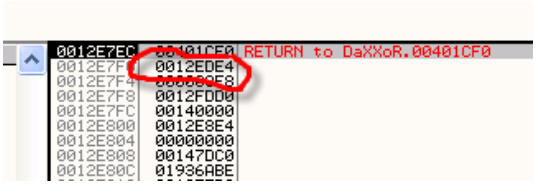
Vemos que el OLLYDBG en este caso tiene un PID de 724 decimal ya que esta utilidad trabaja con números decimales, pero bueno si queremos saber el PID del OLLYDBG en hexa con la calculadora de Windows.



Apreto el boton hex para pasar a HEXA

2D4 será el PID del OLLYDBG, pueden verificar que si lo cierran al OLLYDBG y lo vuelven a abrir el PID variara, pues cada proceso al reiniciarse recibirá otro PID.

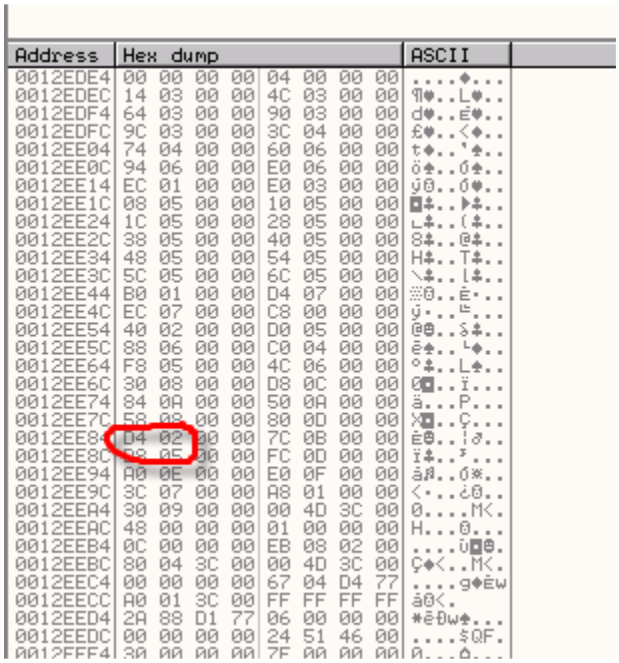
Tampoco tendremos la suerte en este caso de que OLLYDBG nos muestre los parámetros de la api pues para OLLYDBG no existe la misma.



Sabemos por la pagina de Microsoft que los tres parámetros son estos

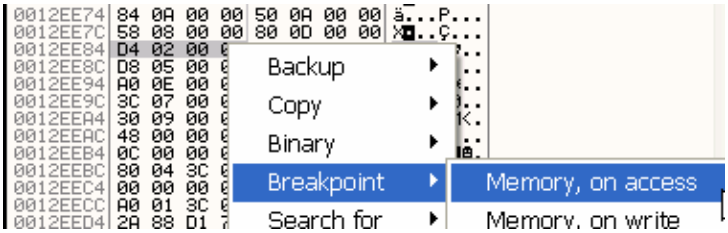
pProcessIds
[out] Pointer to an array that receives the list of process identifiers .
cb
[in] Size of the *pProcessIds* array, in bytes.
pBytesReturned
[out] Number of bytes returned in the *pProcessIds* array.

O sea que en 12eDe4 guardara la lista de PIDs de todos los procesos que corren en mi maquina, hagamos execute till return para llegar al ret de la api y ver en el dump si los guarda alli.

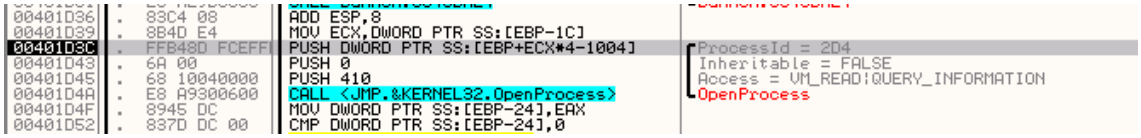


Ahí esta la lista de PIDs y esta el de mi OLLYDBG snif, snif

Pongo un BPM ON ACCESS allí para ver cuando lo usa, veamos



Ahora si doy RUN



Vemos que para allí y va a usar la api OpenProcess que verifica si un proceso esta corriendo, y si esta corriendo te devuelve su handle o manejador.

Que diferencia hay entre el PID y el handle, muy sencillo, el PID es un identificador genérico, en toda tu maquina, en cualquier proceso el PID del OLLYDBG será el mismo mientras no se reinicie, en mi caso será 2d4, ahora el handle, como su nombre lo indica es un manejador, o sea que es un numero que te devuelve el sistema para que tu programa pueda manejar ese proceso, y el numero puede variar para cada programa, es como una solicitud para controlarlo, si no la pedís no tendrás el numerito y no lo podrás controlar, si lo pedís el sistema te devolverá el manejador y lo podrás manejar y hacerle guarradas jeje.

Veamos mas detalladamente la definición de OpenProcess en el WinApis32, la misma tiene muchos parámetros pero lo que nos interesa es esto.

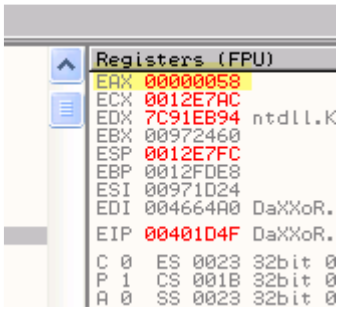
Return Values

If the function succeeds, the return value is an open handle of the specified process.

If the function fails, the return value is NULL. To get extended error information, call GetLastError.

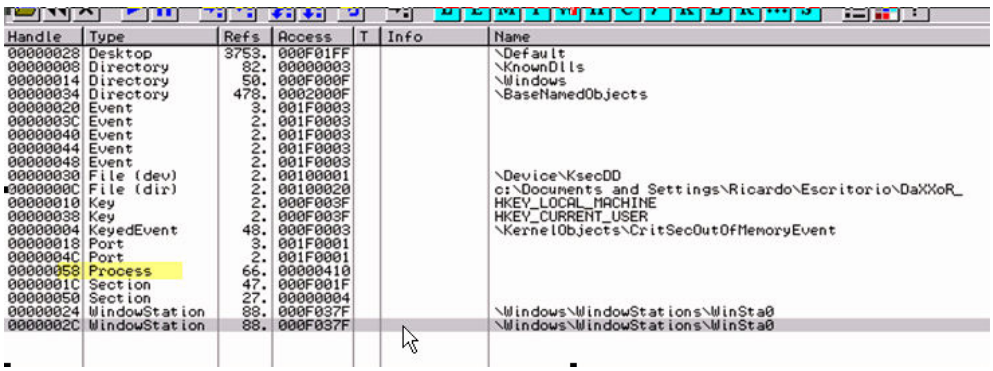
O sea devolverá el manejador del proceso que esta corriendo, en resumidas cuentas es lo que el programa quiere saber en este caso.

Traceemos con f8 hasta pasar la api



Y en EAX devuelve el handle o manejador del OLLYDBG que en mi caso es 58.

El mismo OLLYDBG nos muestra los HANDLES con los cuales esta trabajando el programa en la ventana H.

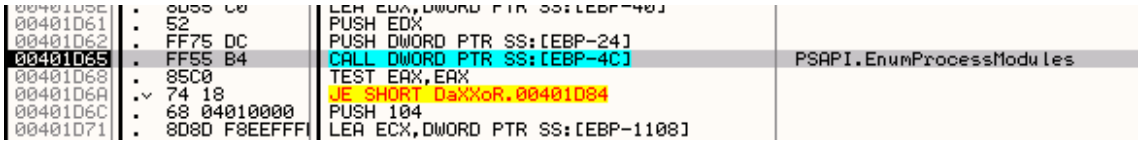


Vemos que allí aparece el 58 y el TYPE o tipo es PROCESS o PROCESO, así que el programa maneja el handle 58 que pertenece a un proceso en este caso al OLLYDBG.

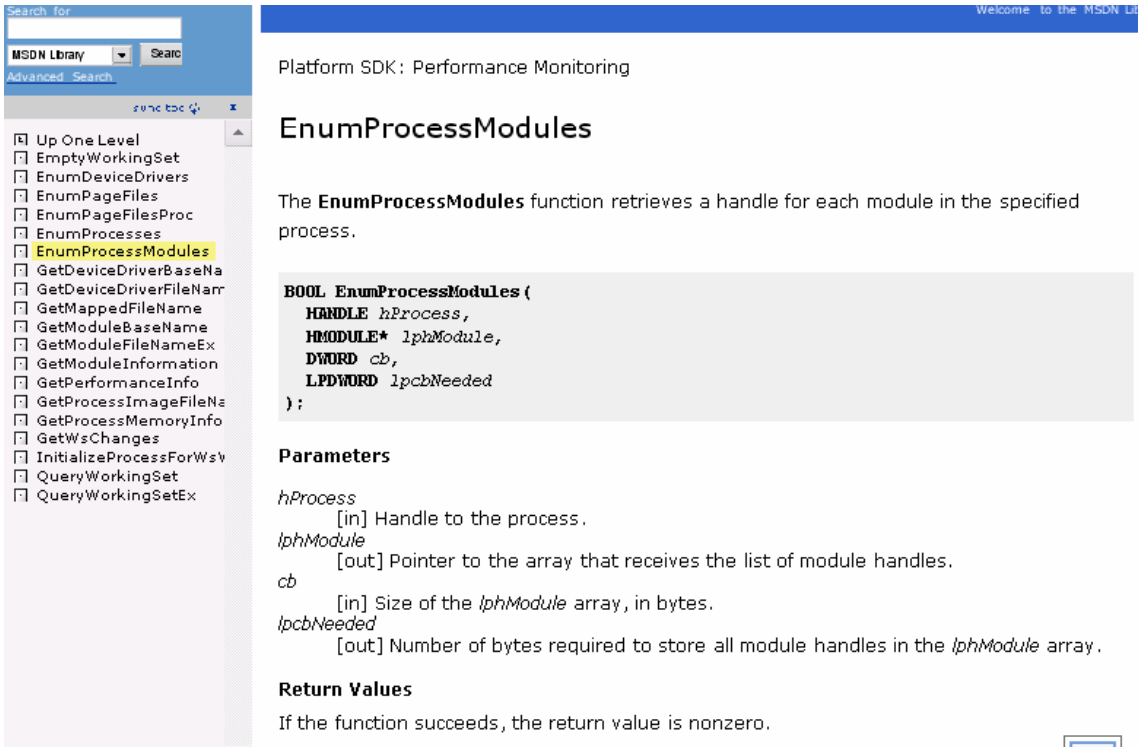
Si otro proceso usara EnumProcess para hallar el PID en este momento seria el mismo 2d4 mientras no se termine el proceso OLLYDBG, ahora si pide al sistema un handle o manejador para dicho proceso, será cualquier otro numero pues, los handles son particulares de cada proceso.

En este caso el peligro para nuestro OLLYDBG es que el programa ya tiene un manejador, con ello puede hacer lo que quiere, lo que si aun no sabe que pertenece a OLLYDBG solo que es un proceso y que esta corriendo, ahora debe verificar el nombre del mismo para determinar si este proceso es OLLYDBG, obviamente esto lo realiza con cada PID que obtiene de la lista de procesos, nosotros salteamos todos y llegamos hasta cuando trabaja con el OLLYDBG al haber puesto un BPM ON ACCESS en su PID.

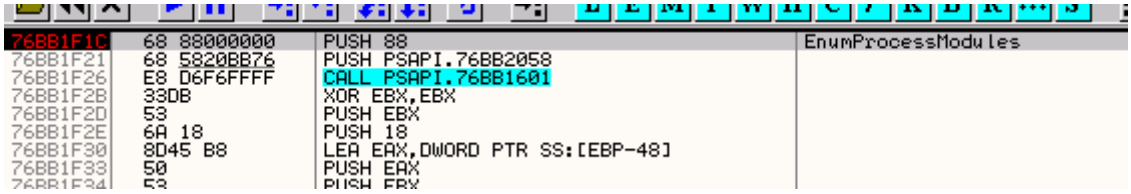
Continuemos traceando con f8



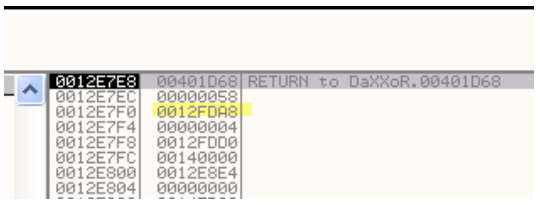
Vemos que allí llega a la otra api que nos quiso ocultar en este caso EnumProcessModules, en la misma pagina de MICROSOFT vemos el detalle de esta api.



O sea que ahora va a buscar de este proceso que esta investigando, la lista de handles de los módulos que usa, lleguemos con f7 hasta la api.

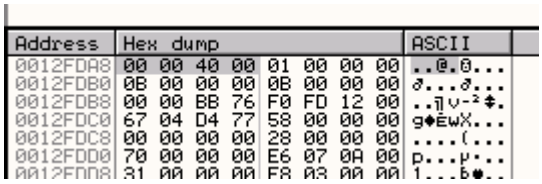


En el stack vemos los parámetros

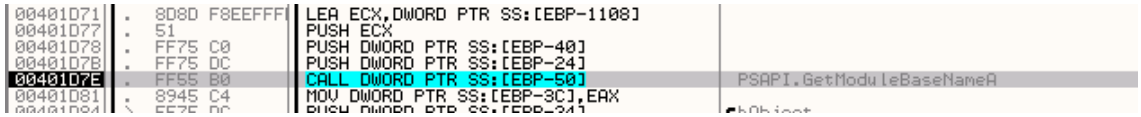


Según nuestro amigo HILL los handles de cada modulo se guardaran allí, vemos justo arriba el 58 perteneciente al handle del proceso OLLYDBG que es el parámetro superior.

Aquí hay una cosita que aclarar, cuando pedimos handle de los módulos el sistema nos devuelve la dirección base o donde comienza dicho proceso en la memoria, en este caso nos devuelve 400000 ya que el proceso OLLYDBG comienza allí.



Ahora continúo traceando y veo que llega a la tercera api ocultada



GetModuleBaseNameA

GetModuleBaseName

The **GetModuleBaseName** function retrieves the base name of the specified module.

```
DWORD GetModuleBaseName(
  HANDLE hProcess,
  HMODULE hModule,
  LPTSTR lpBaseName,
  DWORD nSize
);
```

Parameters

hProcess
[in] Handle to the process that contains the module. If this parameter is NULL, **GetModuleBaseName** uses the current process
The handle must have the PROCESS_QUERY_INFORMATION and PROCESS_VM_READ access rights. For more information, see Process Security and Access Rights.

hModule
[in] Handle to the module. If this parameter is NULL, this function returns the name of the file used to create the calling process

lpBaseName
[out] Pointer to the buffer that receives the base name of the module. If the base name is longer than maximum number of characters specified by the *nSize* parameter, the base name is truncated.

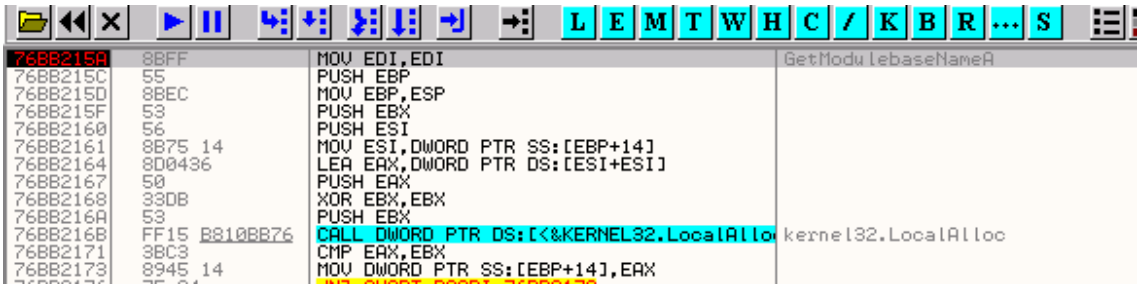
nSize
[in] Size of the *lpBaseName* buffer, in characters

Return Values

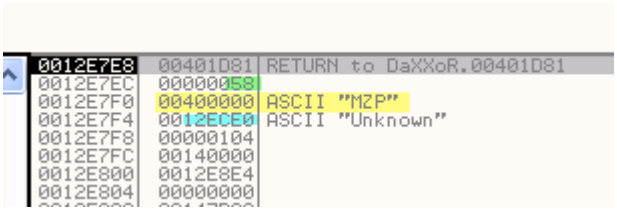
If the function succeeds, the return value specifies the length of the string copied to the buffer, in characters.

If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

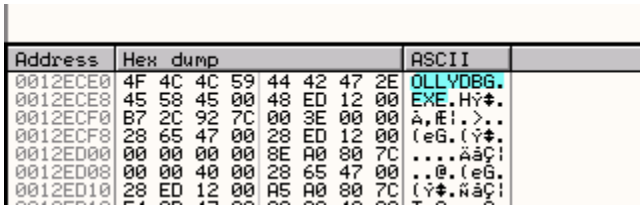
O sea que con esto quiere hallar el nombre del modulo, ya que esta api, en el parámetro lpBaseName, abre un buffer para guardar el nombre del modulo que corresponde a esa base que averiguo antes, veamos lleguemos hasta la api.



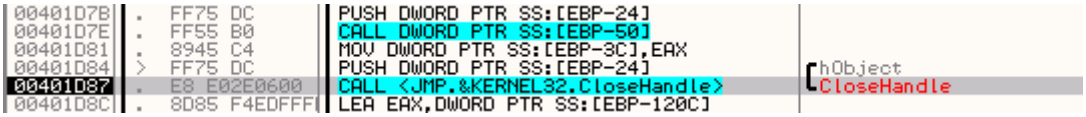
En el stack los parámetros son:



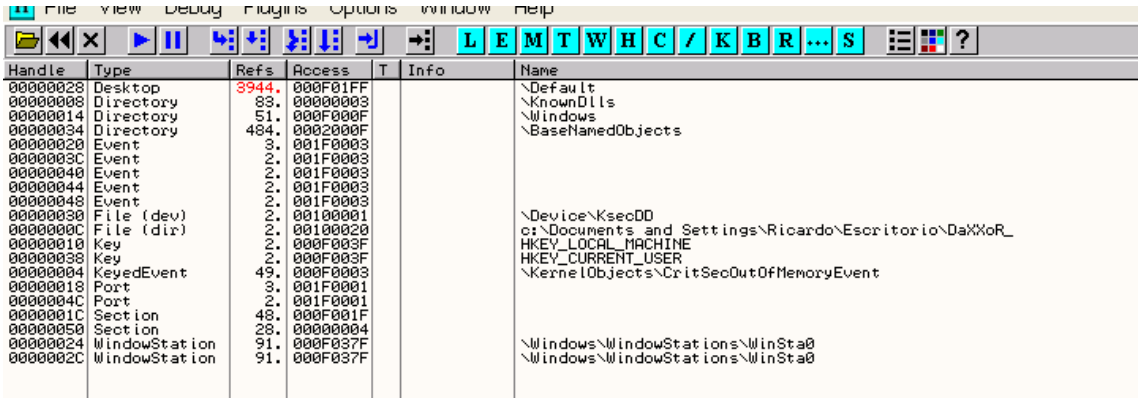
El 58 que es el handle del OLLYDBG, el 400000 que es la base del modulo principal, y el buffer estará guardado en 12ECE0, asi que veamos esa zona en el dump.



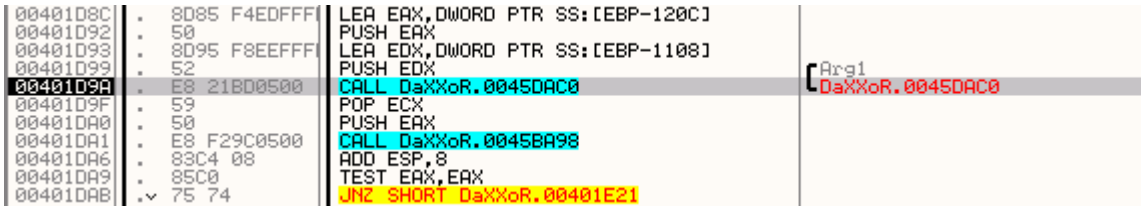
Jeje ya tiene ahora que llega hasta el RET el nombre del proceso y realiza esto con cada uno de los procesos que corren en tu maquina, pues ahora comparara el nombre a ver si es OLLYDBG.exe si en este caso hubiera sido OPERA.EXE por ejemplo, pues lo dejara tranquilo seguramente jeje.



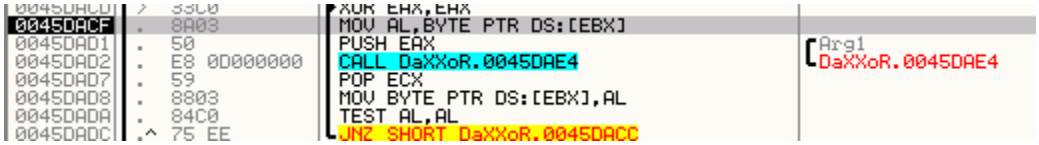
Ahora llega a CloseHandle donde cierra el manejador o sea que el 58 desaparecerá de la lista de handles.

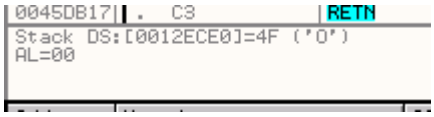


Pues si, por ahora no podrá hacerle trastadas pues no tiene el handle abierto, pero no nos confiemos sigamos adelante.



Vemos que llega a un CALL entremos a el con F7

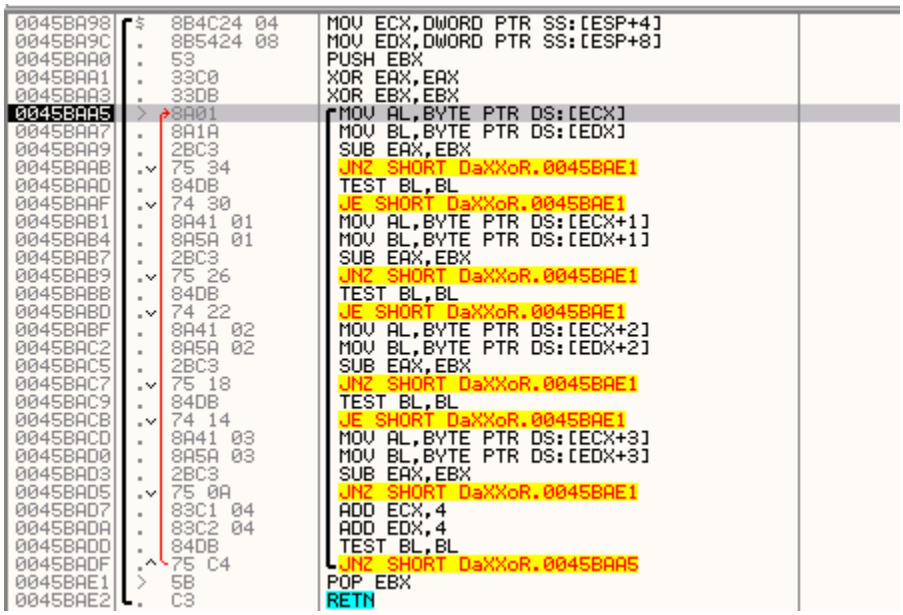


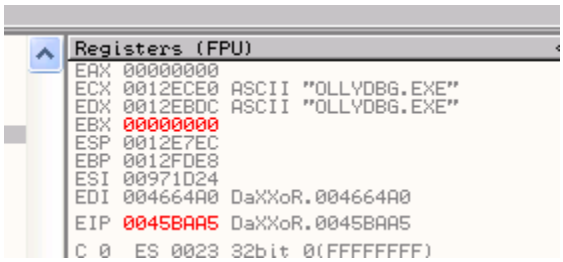


Allí lee la primera letra de OLLYDBG (4F) y la pone en el stack con PUSH y entra a otro CALL entremos también.

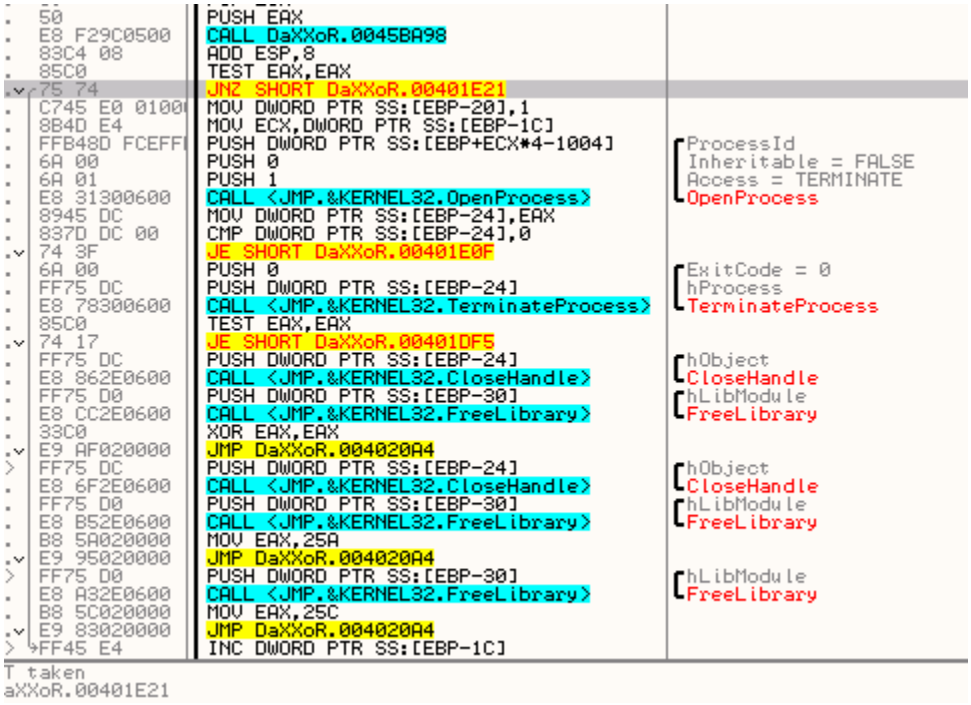
Vemos que en este call no realiza nada demasiado importante salimos y llegamos al segundo.

Entramos en el

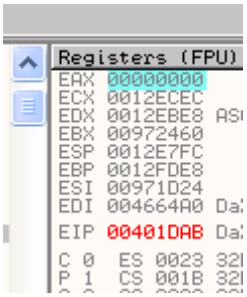




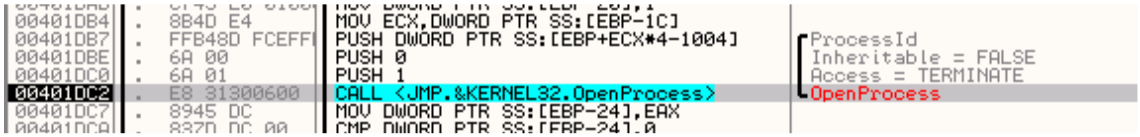
Ahh acá si esta comparando el nombre del proceso que obtuvo, con el nombre OLLYDBG.exe y si son iguales ir a a kaput, jeje en este caso eran iguales veremos que hace, lleguemos al RET



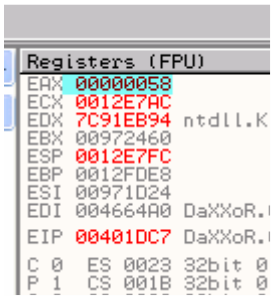
Allí si no son iguales, EAX será diferente de cero y saltara y no pasara nada pero si EAX es igual a cero como en nuestro caso.



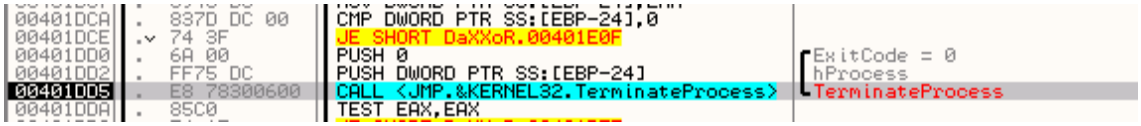
No salta y va a OpenProcess a hallar un handle nuevamente para matarlo.



Si ejecuto con f8



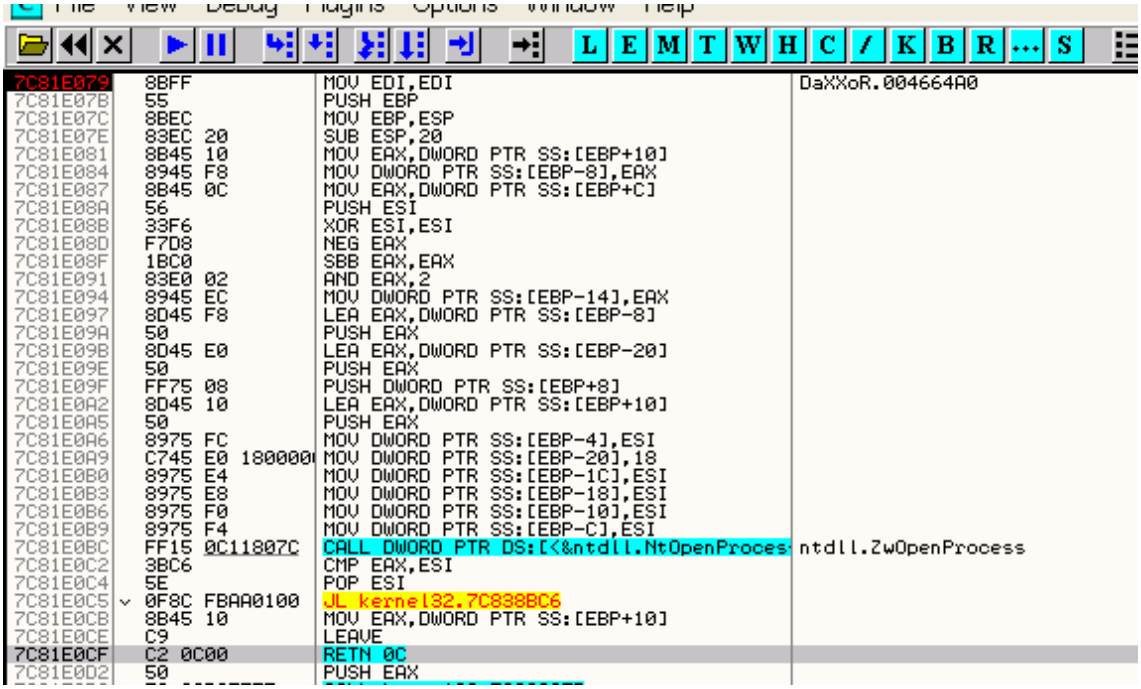
Le volvió a asignar el sistema el numero 58, así que sigamos



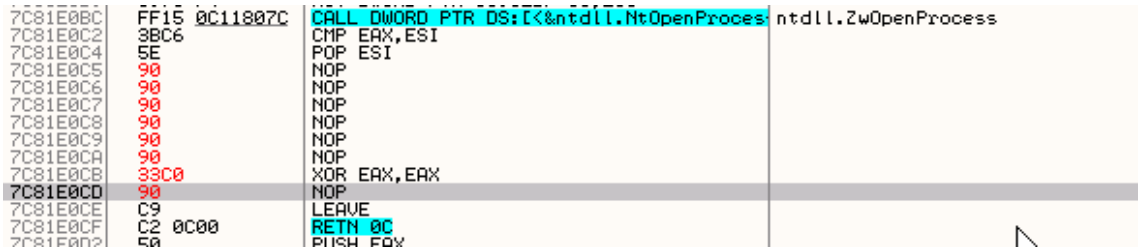
Como vemos llega a la api TerminateProcess auxilio esta por morir mi OLLYDBG a la cual le pasa el handle 58.

Y al apretar f8, adiós OLLYDBG se cerro todo, con eso investigamos como funciona la detección por nombre.

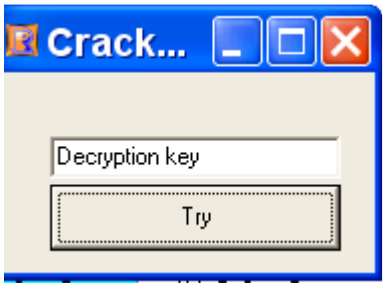
Bueno ya me canse lo haremos un poco a lo maton jeje, reinicio el OLLYDBG y pongo un BP en OpenProcess



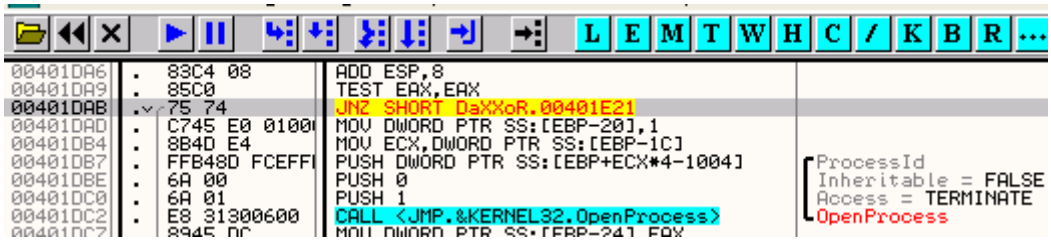
Allí para, si modifico la api para que siempre devuelva cero, el programa pensara que no hay procesos corriendo y no tendrá el handle de ninguno para cerrar, podemos cambiar las ultimas líneas de la api.



Con eso, la api siempre devolverá cero quitemos todos los Bps y demos RUN



Y si así corre perfectamente, pero también podemos hacer lo siguiente:



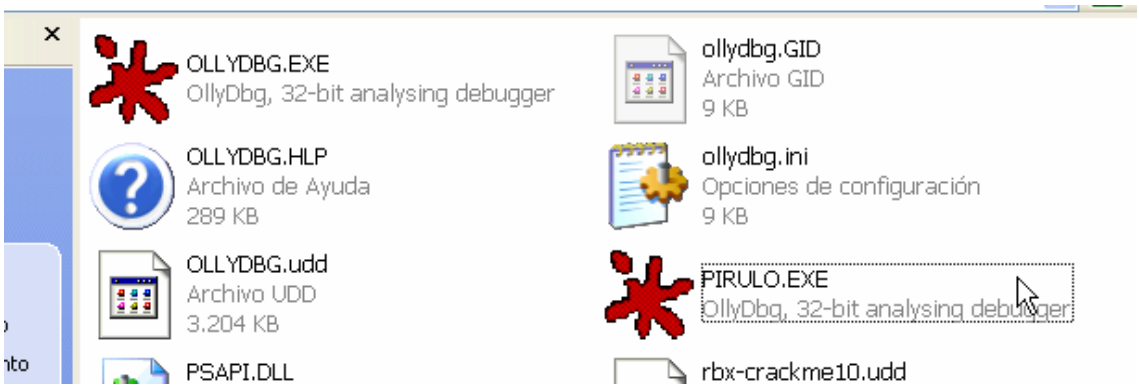
Cambiemos el JNZ por JMP, así evitamos la protección

El programa arranca y muestra la ventana veamos que pasa cuando apreto TRY

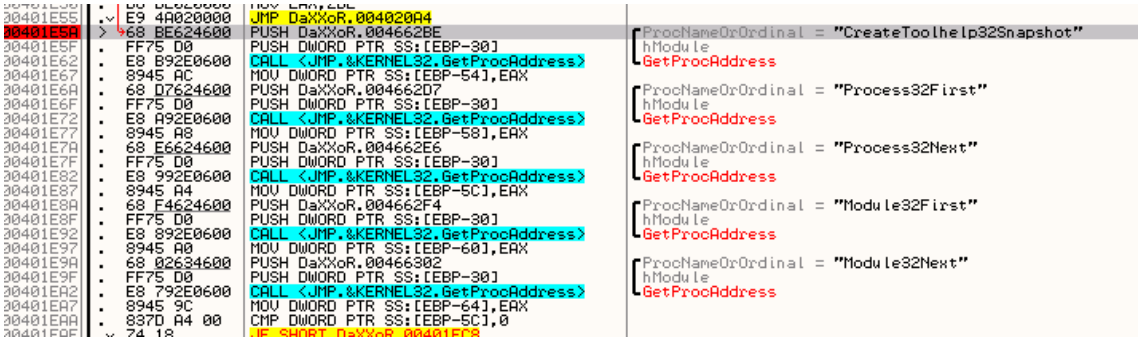
Sale el cartel de error perfectamente eso quiere decir que la protección ANTIDEBUGGER ha sido vencida.

De cualquier manera esto no es lo que se hace habitualmente para vencer esta protección, simplemente copiando el ejecutable OLLYDBG.exe a otra carpeta y cambiándole el nombre por ejemplo a PIRULO.exe y volviéndolo a su carpeta original, de forma que queden ambos el OLLYDBG.exe original y el PIRULO.exe y usando este ultimo, el programa al comparar nunca encontrara el nombre de ningún proceso llamado OLLYDBG, ya que ahora se llamara proceso PIRULO y con eso es vencida esta

protección completamente, de cualquier forma creo que es bueno que sepan como funciona por eso la explicación.



Es importante recordar que aunque usemos un OLLYDBG renombrado, debemos dejar el original en la misma carpeta si no habrá problemas con los plugins.



Hay una parte que el programa posiblemente acceda cuando colocamos un serial bueno, que llama a nuevas apis para una detección diferente, dicha detección la veremos en próximas partes detalladamente.

Hasta la próxima parte 21
Ricardo Narvaja
24 de diciembre de 2005
FELIZ NAVIDAD