
| Victima | VB6.exe (Aimboyd DXL) |
|---|---|
| Protección | Asprotect 2.4 |
| Herramientas | Olly, ODBGscript 1.67, ImportREC |
| Objetivo | Desempacar Asprotect |

## Introducción

Bueno volvemos a la carga con otro Asprotect de los nuevos, esta vez a petición de un amigo de la lista el_chavo.

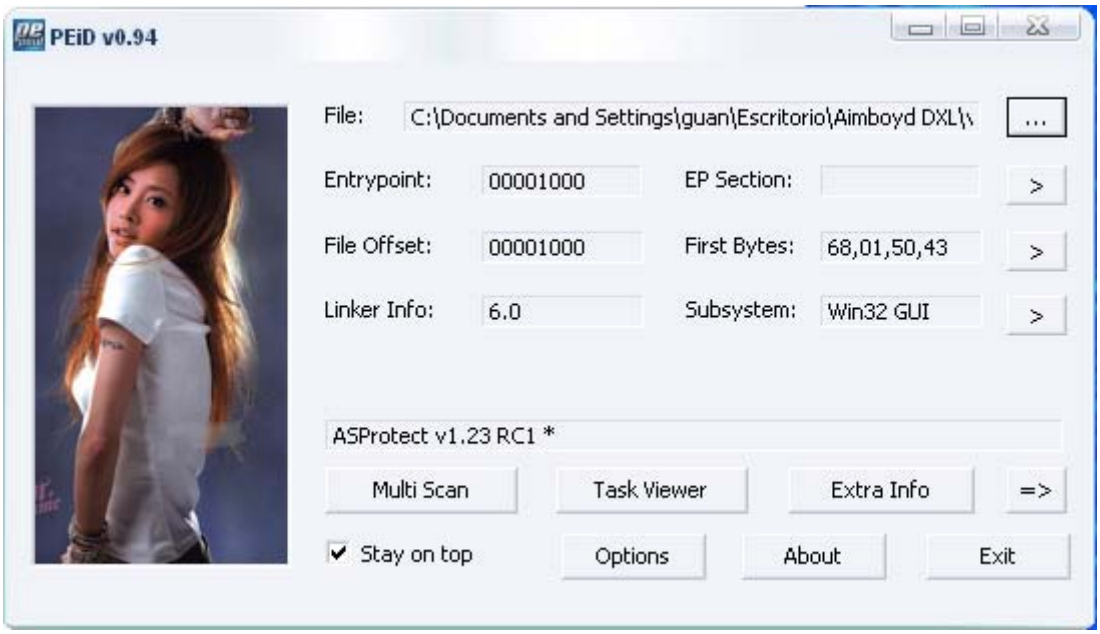
En este tutorial solo trataremos el unpack de protector, no el crackeo del programa.

Esta vez el packer es un poco más sencillo que el de mi último tuto. Como veremos el programa está hecho en VB Nativo lo cual hace más sencillo localizar el OEP, en resumen el packer tiene:

- Detección de depuración
- Byte Stolen
- IAT (una entrada mala)
- Reparación de los famosos CALL

## Conociendo al enemigo

Si pasamos el programa por el PEID vemos esto:



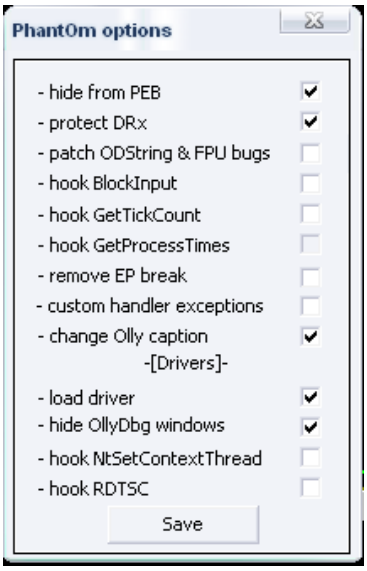

Como este PEID es viejito y no tengo mantenido las signaturas pues lo único que podemos sacar en claro es la protección pero no su versión.

Si lo pasamos por el ASPrINF que es específico para el Asprotect nos da:

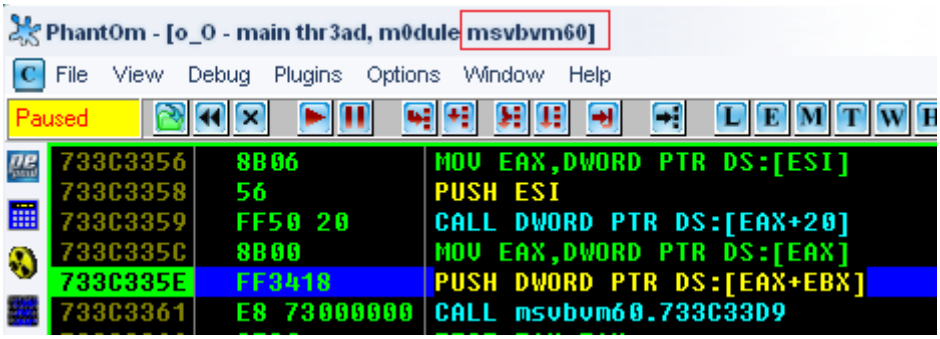

Bueno vemos que es de lo más nuevo que hay, el último es el 2.5

Como el nombre del fichero (vb6.exe) es un tanto sospechoso vamos a ver si realmente es un VB. Para ello simplemente vamos a cargarlo en Olly, lo ejecutamos para que se descomprima en memoria, ponemos un bmp en la sección de código y miraremos la IAT a ver si canta.

Antes de nada este packer si está configurado para comprobar la existencia de debugger, esta vez no me compliqué la vida y usé el plugin phantom configurado de la siguiente manera:




Bueno con esto lo ejecutamos paramos y llegamos aquí.



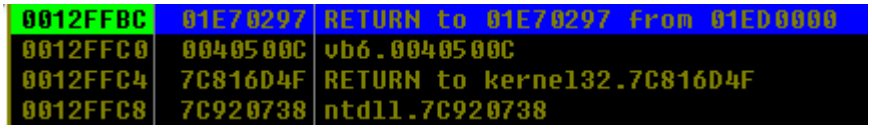

Como podemos ver hemos caído en el módulo msvbvm60, por lo que estamos ante un VB, lo cual nos facilitará mucho la vida.

# Buscando el OEP

Como estamos ante un VB la búsqueda del OEP se facilita muchísimo. Todos los VB llaman al comienzo a la API **ThunRTMain**.

La cosa es sencilla vamos a poner un HE (Hardware Breakpoint on Execute) rearrancamos el Olly y esperamos a que pare, vamos a ver.

Una vez parado vemos la pila



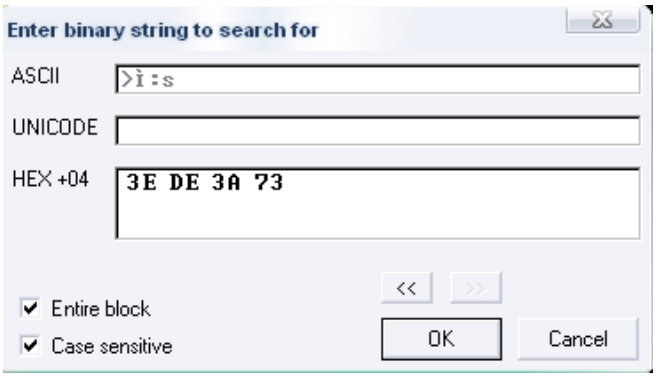

Nos dice que viene de 1ED0000. Eso es malo ya que es una zona del packer, esto quiere decir que estamos ante Stolen Bytes.

Si miramos el EP de un programa en VB tenemos la siguiente estructura:

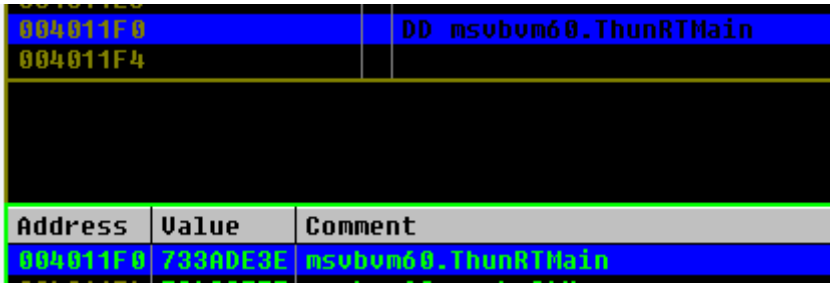
JMP DWORD PTR DS:[dir a IAT]
…
JMP DWORD PTR DS:[dir de IAT a ThunRTMain]
**PUSH Dir a string con la versión del VB**
**CALL a dir del JMP DWORD PTR DS:[dir a IAT a ThunRTMain]**
**4 ceros**

Lo que faltan son las líneas en Rojo y los JMP de la IAT a saber cómo nos lo encontraremos.

Lo que vamos a hacer es buscar a partir de la sección .code la dirección de **ThunRTMain**, en mi PC es la dirección 733ADE3E, como siempre en la búsqueda alteramos los bytes.



Y se detiene aquí

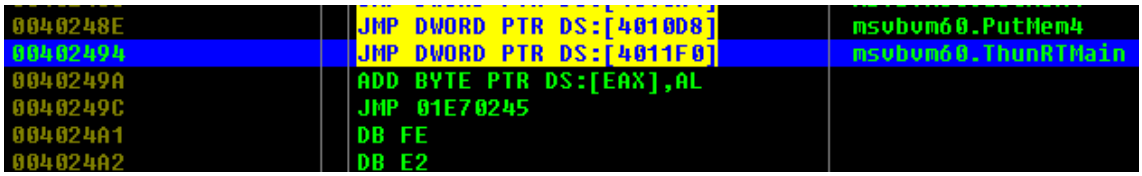



Si vemos la dirección en la ventana Dump queda un poco más bonito.

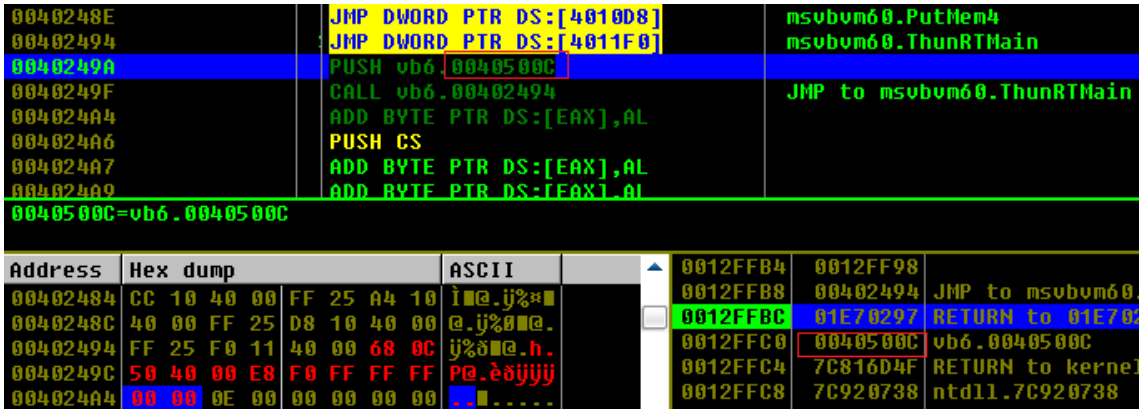
Bueno los siguiente es buscar el JMP DWORD PTR DS:[4011F0]

Lo buscamos en Binario FF 25 F0 22 40 00

Y llegamos aquí



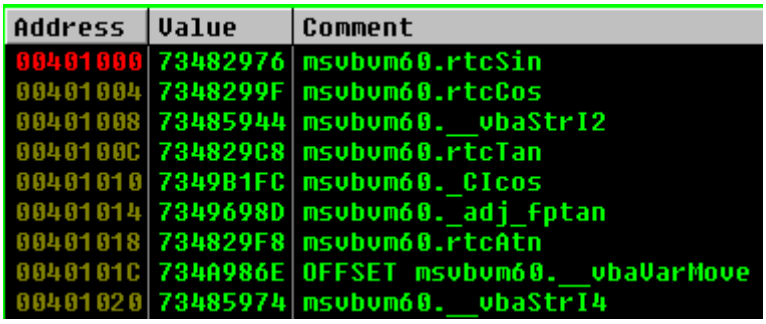

Pues bien el **OEP está en 40249A**. Y reponiendo los Bytes que faltan la cosa debe de quedar así.




El valor del PUSH lo sacamos de la pila, y no olvidar poner los 2 BYTES a 0 después del CALL, sino el VB no arrancará.

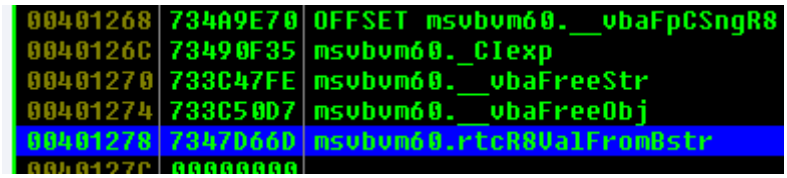
# Reparando la IAT

Bueno sabemos ya que la IAT se encuentra a partir de la dirección 40100 vamos a ver como se encuentra.



| Address | Value | Comment |
|---|---|---|
| 00401000 | 73482976 | msvbvm60.rtcSin |
| 00401004 | 7348299F | msvbvm60.rtcCos |
| 00401008 | 73485944 | msvbvm60.__vbaStrI2 |
| 0040100C | 734829C8 | msvbvm60.rtcTan |
| 00401010 | 7349B1FC | msvbvm60._CIcos |
| 00401014 | 7349698D | msvbvm60._adj_fptan |
| 00401018 | 734829F8 | msvbvm60.rtcAtn |
| 0040101C | 734A986E | OFFSET msvbvm60.__vbaVarMove |
| 00401020 | 73485974 | msvbvm60.__vbaStrI4 |

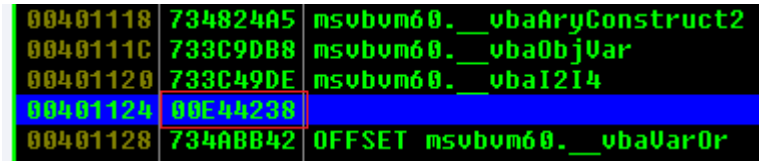
Inicio de IAT



| | | |
|---|---|---|
| 00401268 | 734A9E70 | OFFSET msvbvm60.__vbaFpCSngR8 |
| 0040126C | 73490F35 | msvbvm60._CIexp |
| 00401270 | 733C47FE | msvbvm60.__vbaFreeStr |
| 00401274 | 733C50D7 | msvbvm60.__vbaFreeObj |
| 00401278 | 7347D66D | msvbvm60.rtcR8ValFromBstr |

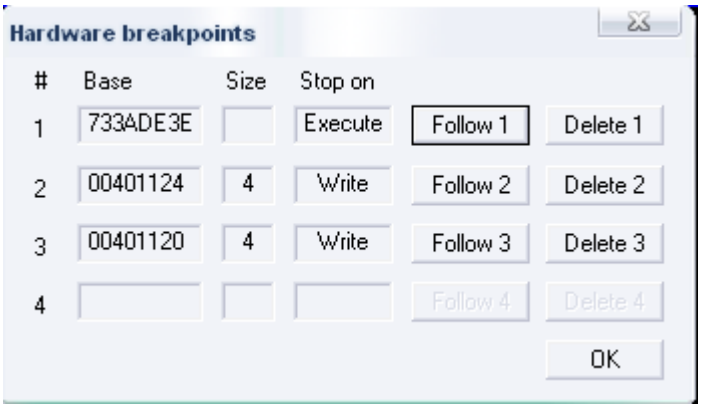
Fin de IAT

Y en medio una entrada mala.

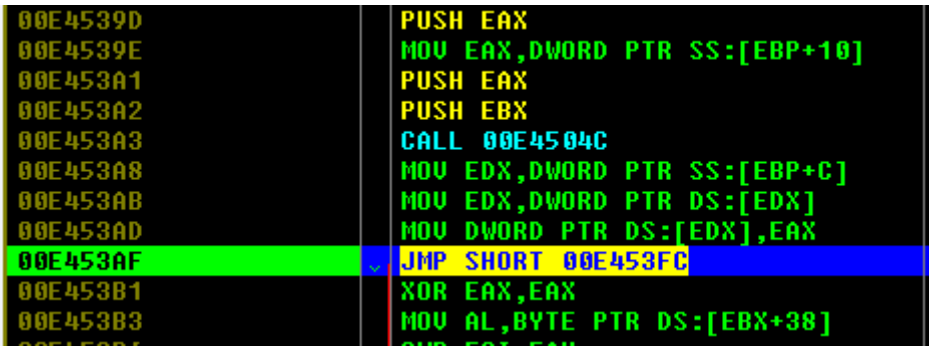


| | | |
|---|---|---|
| 00401118 | 734824A5 | msvbvm60.__vbaAryConstruct2 |
| 0040111C | 733C9DB8 | msvbvm60.__vbaObjVar |
| 00401120 | 733C49DE | msvbvm60.__vbaI2I4 |
| 00401124 | 00E44238 | |
| 00401128 | 734ABB42 | OFFSET msvbvm60.__vbaVarOr |

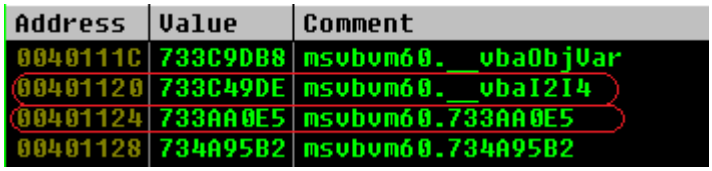
Bueno solo 1 no esta mal. Para averiguarla lo que vamos a hacer es poner un HW on Write DWORD sobre la dirección de la entrada mala (401124) y justo la anterior (401124), para ver donde realiza el cambio y reiniciamos Olly.



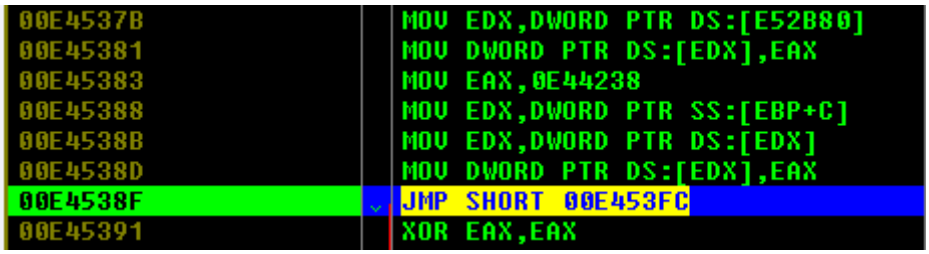

Reiniciamos, ponemos la dirección de la IAT en la ventana DUMP y vamos pasando los BP donde va parando hasta llegar a esta zona




Si vemos el DUMP la IAT ya se está formando




Acabamos de escribir la API buena de la dirección 401120 y lo ha realizado en la dirección 00E453AD. Damos a RUN y veamos donde pone la mala.




Pues la ha escrito un poco más arriba en la dirección 00E4538D

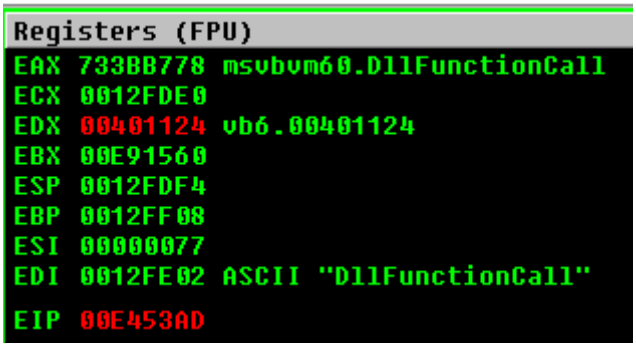
Si miramos un poco arriba vemos que tenemos 2 JMP que va a una función no alineada, es decir tenemos ofuscación.


Vamos a situarnos encima de una de ellas y damos a Enter a ver a donde nos lleva.


Pues caemos justo al XOR EAX,EAX que luego nos lleva al JNZ. Este salto es el salto mágico, si se ejecuta llegamos a 00E453AD donde se escribió la API buena, y si no llegamos a 00E4538D donde nos puso la API mala.

Lo que vamos a hacer es repetir la jugada, y cuando paremos al poner la API buena ponemos un HE en la dirección E4536B donde está el JNZ daremos a RUN y forzaremos el salto, como solo tenemos una entrada no merece la pena usar scripts ni nada por el estilo, así que volvemos a reiniciar.

Cuando forzamos el JNZ vamos traceando y vemos esto:

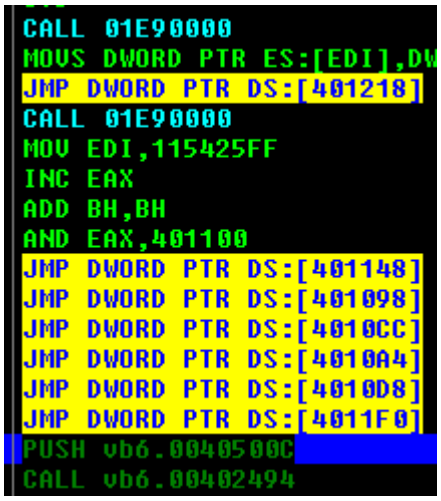



Así que la API que faltaba es la DllFunctionCall y su dirección en mi máquina es la 733BB778, y debe de ir en la dirección 401124 de la IAT.

Bueno con esto quitamos los HW que tenemos y dejamos solo el de la API **ThunRTMain.**

## Reparando los CALL de ASPROTECT

Una vez parados nuevamente sobre la API ThunRTMain, nos vamos al OEP y realizamos la reparación de los Stolen Bytes como vimos al principio y hacemos New Origin here sobre la dirección del OEP.




Una vez parados en el OEP y con la IAT reparada simplemente moviendo la rueda del ratón nos topamos con los famosos CALL, en mi caso CALL 01E90000. Para el que a estas alturas no lo sema Asprotect modificas llamadas a la IAT (JMP indirectos y CALL indirectos) por un CALL a una sección virtual, sección Asprotect por así decirlo. Es un método antidump que vamos a saltar usando un script de mi amigo Solid.

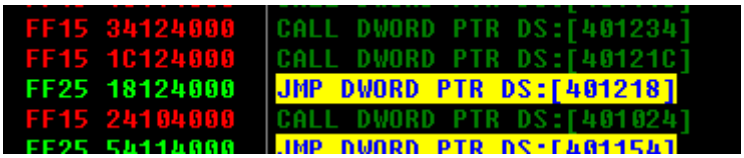
*Asprotect_repara call API PROTECTION_POR_SOLID.txt*

Para que este script funcione hay que estar sobre el OEP, y usar ODBGscript 1.67 o superior.

Los datos que nos hacen falta son:
- Inicio de IAT: 401000
- Fin de IAT: 401278
- Call de Asprotect: Call 001E9000 con las comillas importante

Con estos datos los pasamos al script y le dejamos trabajar tranquilo.


Bueno reparó unos cuantos




Aquí hay un punto a tener en cuenta. Según me comentó Solid el siempre repara con CALL indirectos y nunca a tenido problemas. Si comparamos la zona de JMP a IAT siempre son eso JMP indirectos y no CALL. Hipotéticamente podríamos tener problemas en casos como estos:

CAll dir a JMP de IAT
Y lo que tenemos es un CALL

Esto en la pila se traduce a:

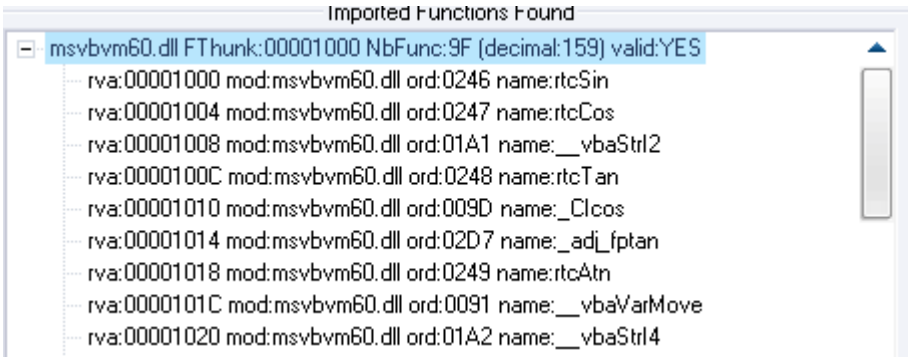
- Return por el CALL reparado por el script
- Return por el CALL a la IAT.

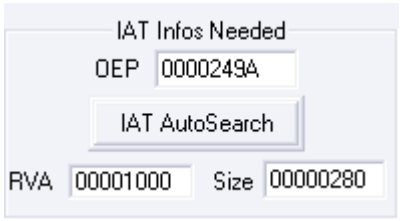
Esto puede provocar que después de ejecutar la API reparada se siga con la siguientes instrucción que será otra API.

¿Cómo reparar? Bueno digamos que si a la instrucción no se le llega por otra (no está referenciado) es un CALL indirector, en cambio si se llega por otra instrucción (un call por ejemplo) es un JMP indirecto.

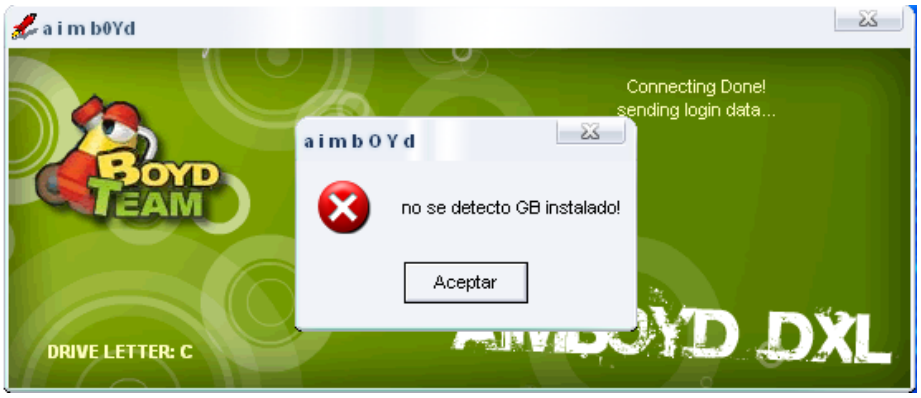
Yo opté por modificar a mano los CALL de la zona de la IAT y el resto los dejé como los solución el script.

Una vez que tenemos esto simplemente dumpeamos y reparamos la IAT con el ImportRect.




Ejecutamos y




Funciona perfecto. Como nota decir que si el programa no tiene su nombre vb6.exe parece que luego no carga una dll, tiene alguna protección por nombre pero eso ya os lo dejo a ustedes.

Esto fue todo, este tutorial está especialmente dirigido a El_Chavo. Agradecer al maravilloso grupo de CracksLatinos las enseñanzas de todos estos años, en especial a Solid por ese maravilloso script ☺.

Bueno hasta la próxima.