

# VB P-code + dongle

---

*Autor: Alejandro Torres (torrescrack)*

*Objetivo: Modificar y burlar los chequeos  
USB de la aplicación*

*Herramientas: Editor hexadecimal,  
p32dasm (p-code decompiler)*

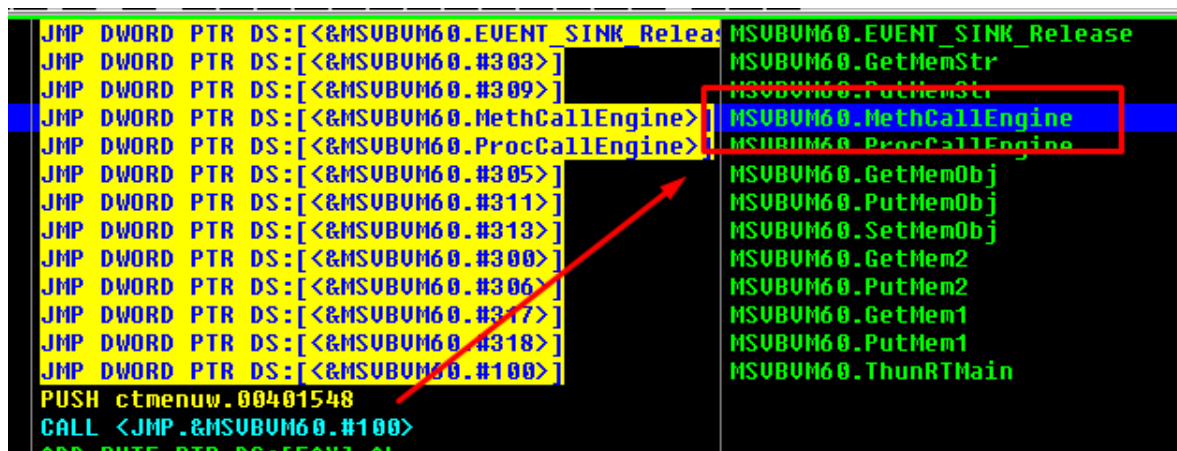
## INTRODUCCION

Después de casi un mes de mi último escrito sobre la primer parte del análisis de malware, aquí sentando en una de esas noches que no puedo dormir (muy común) y para no dejar más tiempo sin movimiento mi blog y los foros decidí escribir algo muy corto acerca de la protección de un programa que me pasaron por correo y este trata de un visual basic p-code + dongle, me intereso escribir sobre esto por dos razones: primero porque muy pocas veces me encuentro con un p-code y segundo: porque para muchos inclusive para mi hace un buen tiempo, al leer “dongle” se piensa que ha de tratarse de algo muy complicado, de hecho de casualidad en la lista si mal no recuerdo preguntaban sobre una protección similar y han hecho un hilo sobre ello así que quedo como anillo al dedo para complementarlo un poco mas, sobre el dongle muchas veces es tan mal utilizado que con un simple cambio de bytes se burlaría la protección y este será uno de los que veremos a continuación así que no los aburriré más y vamos a ver de qué trata.

## EXPLICANDO:

Para los que ya saben acerca de los p-code sáltense esta parte para los que no, les explico, p-code es una forma de compilar nuestros programas en visual basic, este posee dos formas la nativa y la p-code , el compilado nativo es el más común y al que muchos estamos acostumbrados a ver en un vb es el que ejecuta código en la sección .code, la diferencia el compilado p-code únicamente ejecuta códigos llamados p-code los cuales son interpretados por la dll de Visual Basic, estos se pueden interpretar con olly como lo muestra Ricardo Narvaja en su explicación sobre los p-code pero sería una tarea inmensa, y como nos gustan las cosas más fáciles usaremos algunas herramientas que se han creado para decompilar p-code o poder debugearlos, en este caso solo haremos uso de una.

Podemos cargar el ejecutable en olly, el entryptpoint es muy similar en los visual basic nativos y si no conocemos podríamos seguir debuggeando hasta darnos cuenta que algo no anda bien, pero estos tienen una gran característica que los identifica y es el uso de la API METHCALLENGINE:



JMP DWORD PTR DS:[<&MSUBUM60.EVENT_SINK_Release>]	MSUBUM60.EVENT_SINK_Release
JMP DWORD PTR DS:[<&MSUBUM60.#303>]	MSUBUM60.GetMemStr
JMP DWORD PTR DS:[<&MSUBUM60.#309>]	MSUBUM60.PutMemStr
JMP DWORD PTR DS:[<&MSUBUM60.MethCallEngine>]	MSUBUM60.MethCallEngine
JMP DWORD PTR DS:[<&MSUBUM60.ProcCallEngine>]	MSUBUM60.ProcCallEngine
JMP DWORD PTR DS:[<&MSUBUM60.#305>]	MSUBUM60.GetMemObj
JMP DWORD PTR DS:[<&MSUBUM60.#311>]	MSUBUM60.PutMemObj
JMP DWORD PTR DS:[<&MSUBUM60.#313>]	MSUBUM60.SetMemObj
JMP DWORD PTR DS:[<&MSUBUM60.#300>]	MSUBUM60.GetMem2
JMP DWORD PTR DS:[<&MSUBUM60.#306>]	MSUBUM60.PutMem2
JMP DWORD PTR DS:[<&MSUBUM60.#317>]	MSUBUM60.GetMem1
JMP DWORD PTR DS:[<&MSUBUM60.#318>]	MSUBUM60.PutMem1
JMP DWORD PTR DS:[<&MSUBUM60.#100>]	MSUBUM60.ThunRTMain
PUSH ctmenuw.00401548	
CALL <JMP.&MSUBUM60.#100>	
ADD BYTE PTR DS:[EAX], 01	

De esa forma sin duda podemos identificarlos.

### EMPEZANDO CON EL ATAQUE:

Vamos a ejecutar nuestra aplicación para ver el error y tener una idea de lo que buscaremos:



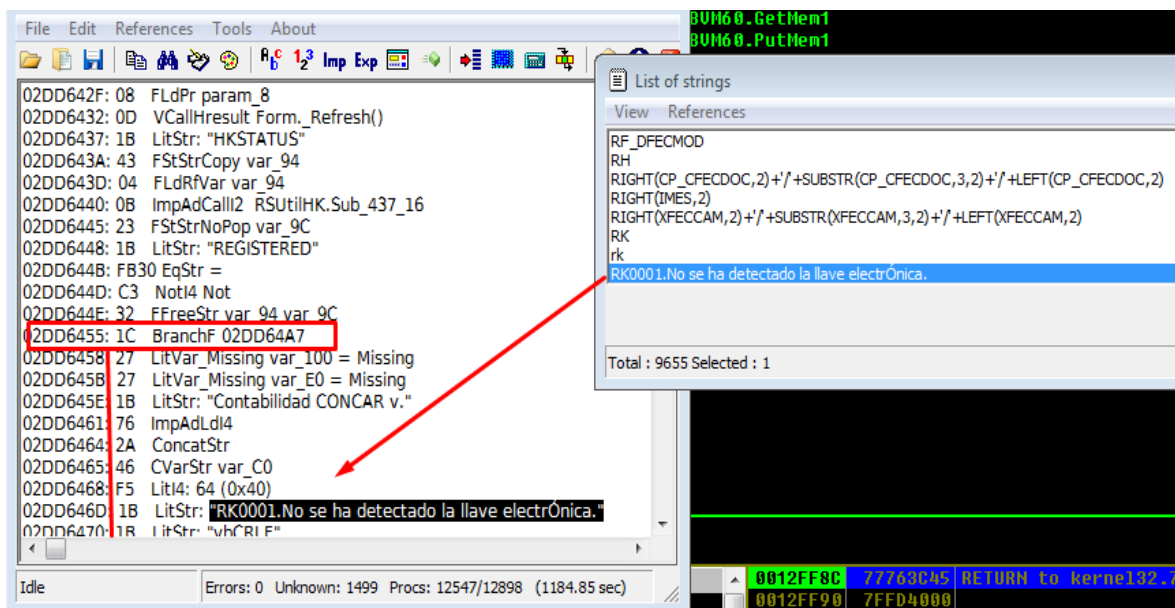
Como verán borre algunos detalles o lo que hacía referencia al software ya que aquí lo importante es tratar la protección y no hacer mención del nombre del software.

Bien ahora con esos detalles abrimos nuestro decompilador p32dasm el cual nos sirve también para p-code:

*“P32Dasm is a Visual Basic 5.0/6.0 PCode + Native code Decompiler. It can generate String, Numbers, Objects, Import and Export function listing. There is also Jump calculator. For VB Native code executables are generated only MSVBVM, External calls and string references. Usefull for setting BPX, you don't need search in debugger where start some Command Button event. You can generate .map files, which you can import to DataRescue IDA (LoadMap plugin) or to Olly Debugger (MapConv plugin)”*

Para mí que es el mejor para p-code, seguimos y lo cargamos con nuestro ejecutable a analizar

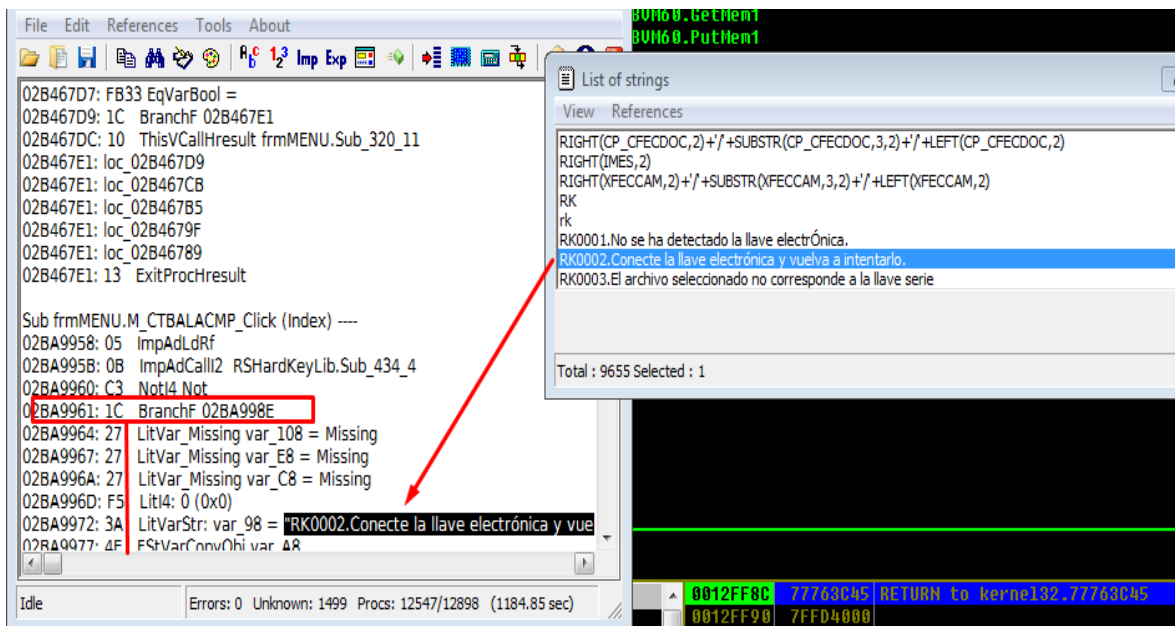
Buscamos el string del error MsgBox y nos encontramos con esto:



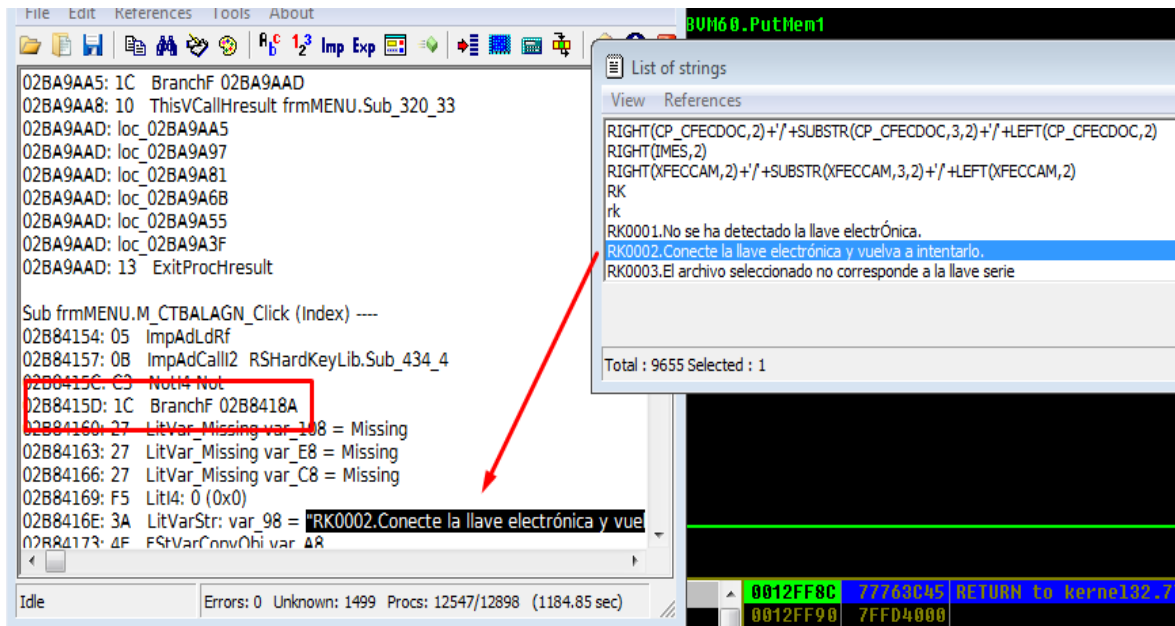
Encontramos el string y la dirección donde muestra el mensaje, al parecer, todo se ve muy simple, más arriba en la dirección 0x2DD6455 hay p-code "BranchF=1C" y es similar a un salto condicional, el salto se realiza si la comprobación anterior es falsa, y nos llevaría a seguir la ejecución del programa burlando la primer protección, si, la primera, ya verán de que hablo, aunque lo que se podría hacer sería tratar de emular un dongle encontrando las llamadas, las regiones donde retornan los valores pero es un labor complicado (mas tiempo) y como dije arriba si hay forma de hacerlo más rapido ahí que intentarlo, por lo tanto se puede cambiar por un p-code "Branch=1E" lo cual es similar a un salto incondicional y eso nos llevaria siempre a la zona deseada, bien ahora lo siguiente es modificar ese byte y con eso podríamos burlar la primer parte, pero no cantemos victoria tan rapido



porque si seguimos avanzando la ejecución del programa aparecerá otro error del tipo “RK0002” diciendo que hay un error con la llave etc, así que vamos directo a buscar ese mensaje:



Bien creo que tenemos la segunda parte que es prácticamente lo mismo, mas arriba en la dirección 0x2BA9961 tenemos un salto condicional que es lo que nos haría burlar este segundo chequeo, ya sabemos con un “Branch=1E” suplantando al “Branchf=1C” lograríamos nuestro objetivo, pero he aquí el truco que me tomo por sorpresa y es que cuando realice los cambios me seguía mostrando el mensaje de error, y ahí algo que no me había dado cuenta y es que si volvemos a hacer doble click en el string que encontramos, nos manda a una dirección nueva con el mismo rollo:



Si volvemos a hacer doble click en el string nos manda a otra dirección, y así unas 8 veces aproximadamente hasta que se vuelven a repetir, es por eso que no me funcionaba al principio, por eso es mejor tratar de emular el dongle (y no tomar caminos fáciles jeje), aquí tuvimos suerte que no fueran unas 100 veces o más las que se llamaban para hacer el chequeo del dongle así que bueno ya tenemos las direcciones que vamos a modificar, para editar o parchear abrimos el editor hexadecimal y buscamos las direcciones:

Original:

	02DD6420	02 00 1A 68 FF F4 FF 08 08
A	02DD6430	08 00 0D A0 02 07 00 1B 08
1	02DD6440	0B 09 00 04 00 23 64 FF 1B
hexadecimal	02DD6450	00 6C FF 64 FF 1C FF 00 27
ANSI ASCII	02DD6460	00 76 01 00 2A 46 40 FF F5
hexadecimal	02DD6470	1B 0D 00 2A 23 6C FF 1B 0E
37x16=592	02DD6480	00 2A 23 60 FF 1B 0F 00 2A
4	02DD6490	00 32 06 00 6C FF 64 FF 60
3	02DD64A0	FF 20 FF 00 FF FC C8 3A 10
	02DD64B0	0C 00 4D 30 FF 08 40 04 50

Parcheado:

rite time: 10/07/2013	02DD6400	18 03 19 68 FF 08 68 FF 0D 54
05:04:56	02DD6410	1B 06 00 21 0F 14 03 19 68 FF
tes: A	02DD6420	02 00 1A 68 FF F4 FF 08 08 00
1	02DD6430	08 00 0D A0 02 07 00 1B 08 00
	02DD6440	0B 09 00 04 00 23 64 FF 1B 0A
hexadecimal	02DD6450	00 6C FF 64 FF 1E FF 00 27 00
cter set: ANSI ASCII	02DD6460	00 76 01 00 2A 46 40 FF F5 40
s: hexadecimal	02DD6470	1B 0D 00 2A 23 6C FF 1B 0E 00
per page: 37x16=592	02DD6480	00 2A 23 60 FF 1B 0F 00 2A 46
ur ff: A	02DD6490	00 32 06 00 6C FF 64 FF 60 FF

Hacemos lo mismo en cada dirección que encontremos como les mencione arriba, después guardamos los cambios en el ejecutable y listo ya tenemos el ejecutable parcheado y desprotegido, de esta forma



Palabras Finales:

Quiero y debo agradecer a muchas personas en especial a ti por leer y llegar hasta aquí, ah y no se olviden de dejar opiniones, consejos, dudas, insultos a [tora\\_248@hotmail.com](mailto:tora_248@hotmail.com)



<http://www.facebook.com/yo.torrescrack>



<https://twitter.com/TorresCrack248>



[www.torrescrack.blogspot.com](http://www.torrescrack.blogspot.com)