

INTRODUCCION AL CRACKING CON OLLYDBG parte 30

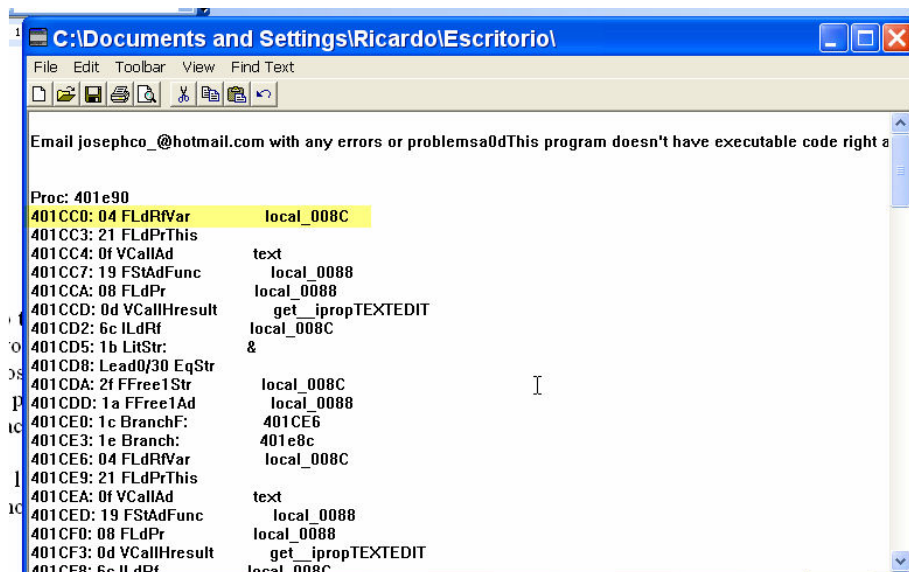
Bueno después de haber practicado y si aun están vivos seguiremos con PCODE.

Tenemos algunos OPCODES más que juntamos por ahí de los tutes de JBDUC

6c -> ILdRf Empuja una dirección a la pila
1b -> LitStr5 Empuja una cadena literal a la pila
fb -> Lead0 Compara dos cadenas (jeje, para que podría servir esto :-)
30 -> EqStr Compara dos cadenas (jeje, para que podría servir esto :-)
2f -> FFree1Str Libera la memoria usada
1a -> FFree1Ad Libera la memoria usada
0f -> VCallAd Ejecuta código dentro de la máquina virtual
1c -> BranchF Salta si la comparación previa era falsa (vamos lo mismo que un jne/jnz de asm)
1d -> BranchT Salta si la comparación previa era cierta (vamos lo mismo que un je/jz de asm)
1e -> Branch Salta incondicionalmente (jeje adivina ke utilidad tiene)
fc -> Lead1 Termina la ejecución del programa (jeje este es bueno...)
c8 -> End Termina la ejecución del programa (jeje este es bueno...)
f3 -> LitI2 Guarda el Integer especificado en la pila
f4 -> LitI2_Byte Convierte un valor de Byte a Integer y lo mete en la pila
70 -> FStI2 Guarda el último Integer que se haya en la pila en la variable global especificada
6b -> FLdI2 Carga en la pila un Integer desde la variable local especificada
a9 -> AddI2 Suma los dos últimos Integers que se empujaron a la pila y mete en pila el resultado
ad -> SubI2 Resta los dos últimos Integers que se empujaron a la pila y mete en pila el resultado
b1 -> MulI2 Multiplica los dos últimos Integers que se empujaron a la pila y mete en pila el resultado como un Integer, creo que si hay desbordamiento se ignora.

Bueno tenemos unos cuantos opcodes mas, que nos evitan tener que investigarlos, además adjunto un archivo llamado P-CODE OPCODES que supuestamente distribuye Microsoft que ayuda un poco y que lista los opcodes y que hace cada uno (no todos jeeje), en una muy escueta descripción (jeje demasiado escueta para mi gusto, pero bueno, antes de entrar a un opcode conviene mirar a ver si en el mismo podemos comprender que hace, así entramos con cierta idea.

Como les prometí, al inicio haremos el crackme clave 2 que quedo de ejercicio la vez anterior, miremos si podemos hallar un listado con el EXDEC.



Ahí tenemos y vemos que comienza en 401cc0, para los que no creen en el EXDEC ya que muchas veces puede ser engañado, veamos si hallamos el primer OPCODE a mano, como vimos en la parte anterior.

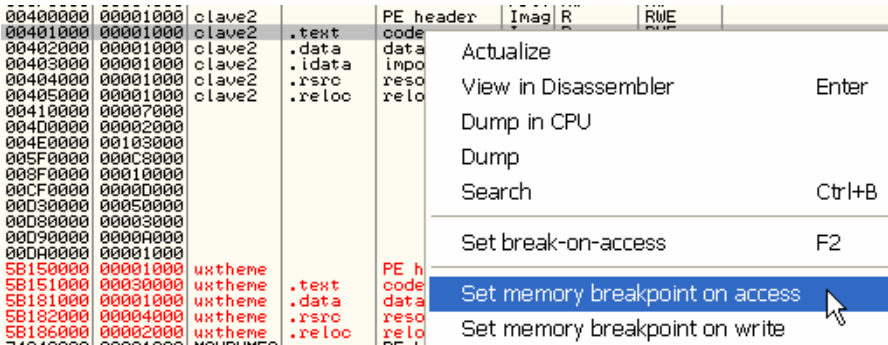
Veamos si esta el JMP a la api MethCallEngine por encima del EP.

Allí esta, por si acaso sea llamada directamente sin usar el JMP; nos posicionamos sobre el y hacemos FOLLOW para ir a la api y ponemos un BP allí también.

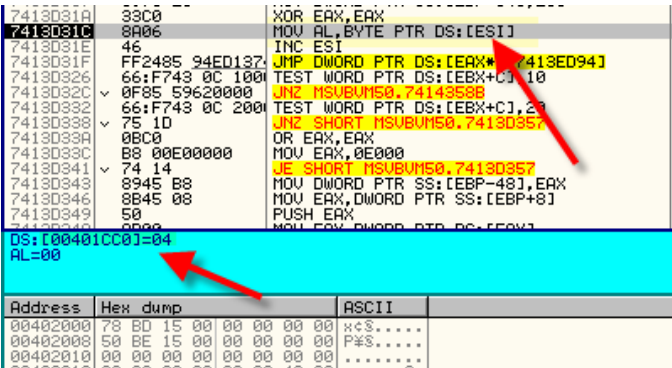
Listo ahora demos RUN hasta que pare en alguno de estos BP que colocamos.

Aparece la ventana antes de parar en el BP, el tema que la ventana aparece antes que entrar a PCODE, debo aclarar que no necesariamente es así, puede parar en el BP y luego aparecer la ventana, en este caso, aparece la ventana antes, coloco un nombre y serial falso.

Y al apretar REGISTRAR para en el BP del JMP ahora voy a la primera sección (recordar no usar el OLLY parchado para OEPs y Visual si no, no funcionara) y le coloco un BPM ON ACCESS en la sección code.



Ahora daré RUN hasta que encuentre la instrucción conocida, de que esta leyendo un OPCODE



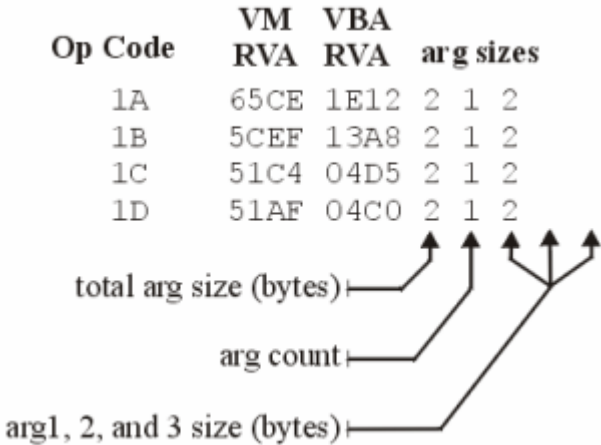
Allí esta, recordemos que siempre ESI debe apuntar al opcode y mover el mismo a AL.

Como vemos el EXDEC no se equivoco y el primer OPCODE esta en 401cc0 y es el 04.

Proc: 401e90
401CC0: 04 FLdRfVar **local_008C**

Recordamos que el 04 era un simple PUSH del argumento que en este caso es EBP-8C si miramos la ayuda que adjuntamos sobre el OPCODE.

04 567B 0B8E 2 1 2 Push arg



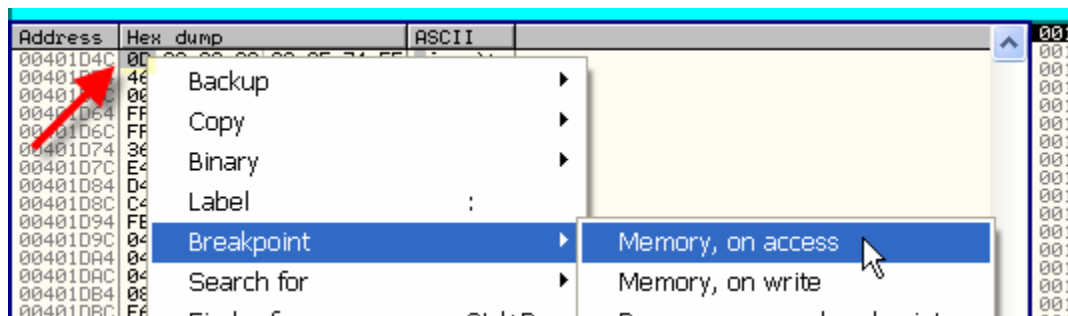
Allí mismo nos aclara que es cada cifra en nuestro caso el 0B8E seria el RVA del opcode, 2 seria la cantidad de bytes que tienen los argumentos en total, 1 nos aclara que tiene un solo argumento, y el ultimo 2 seria el largo en bytes de cada argumento por separado.

Bueno es un PUSH EBP-8C sigamos mirando el listado sin necesidad de tracear uno a uno, vemos estos.

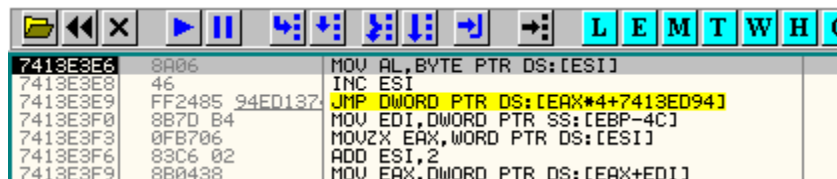
401D4C: 0d VCallHresult get__ipropTEXTEDIT

401DFB: 0d VCallHresult get__ipropTEXTEDIT

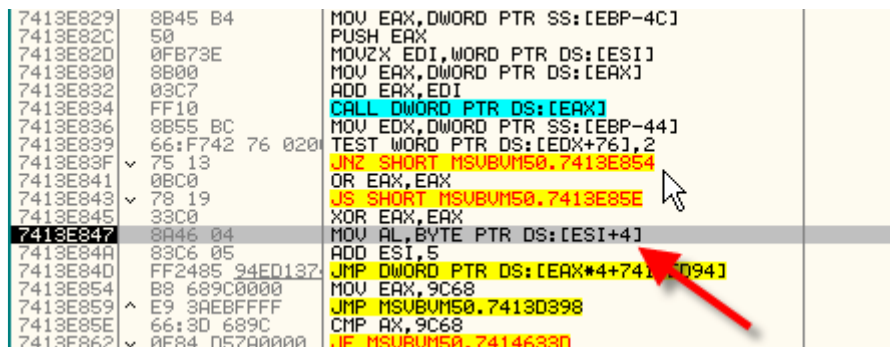
Vemos que hay dos calls para leer lo que ingreso en la ventana de registro, posiblemente el primero lea el nombre y el segundo el serial falso que tipeo, pongamos un BPM ON ACCESS en el primero o sea en 401d4C.



Allí esta el opcode y le coloco el BPM y doy RUN.



Bueno ahora traceemos hasta el siguiente opcode, a ver si ingresa el nombre.



Allí llega al siguiente opcode miremos el stack a ver si lo guardo

401D4C: 0d VCallHresult get__ipropTEXTEDIT

401D51: 3e FLdZeroAd local_008C

Como vemos que a continuación trabaja con la variable local 8c que equivale a EBP-8C me fijo si lo guardo allí, en mi maquina ebp-8c vale 12f454



Busco en el stack esa dirección a ver si esta allí mi nombre.

0012F44C	00000000	
0012F450	00000000	
0012F454	0015CA94	UNICODE "narvaja"
0012F458	00CFBE1C	
0012F45C	0012F4EC	
0012F460	0012F5C8	
0012F464	00CFBB5C	
0012F468	0012F4E8	

Pues si, allí lo veo, así que vamos bien, aquí veo que ingreso mi nombre, lo lógico sería que en el otro, ingrese mi serial falso, pongamos un BPM ON ACCESS en el segundo.

401DFB: 0d VCallHresult get__ipropTEXTEDIT

Address	Hex dump	ASCII
00401DFB	0D	
00401E03	46	
00401E0B	00	
00401E13	FB	
00401E1B	34	
00401E23	F4	
00401E2B	02	
00401E33	F5	
00401E3B	0A	
00401E43	0A	
00401E4B	54	
00401E53	1E	
00401E5B	FE	
00401E63	00	

Listo ahora doy RUN hasta llegar a ese opcode

Address	Hex dump	ASCII
00401DFB	0D A0 00 00 00 3E 74 FF	.ä...>t
00401E03	46 34 FF 5D 3A 64 FF 09	F4]:d.
00401E0B	00 04 74 FE FB EF 54 FF	.+t' T.
00401E13	FB 40 1A 78 FF 36 04 00	'@x 6.
00401F18	34 FF 54 FF 1C 99 01 27	4 T, nA'

Como antes traceemos hasta el siguiente opcode.

7413E83F	75 13	JNZ SHORT MSUBUM50.7413E854
7413E841	0BC0	OR EAX,EAX
7413E843	78 19	JS SHORT MSUBUM50.7413E85E
7413E845	33C0	XOR EAX,EAX
7413E847	8A46 04	MOV AL,BYTE PTR DS:[ESI+4]
7413E849	83C6 05	ADD ESI,5
7413E84D	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]
7413E854	B8 689C0000	MOV EAX,9C68
7413E859	E9 3AEBFFFF	JMP MSUBUM50.7413D398
7413E85E	66:3D 689C	CMP AX,9C68
7413E862	0F84 D57A0000	JE MSUBUM50.7414633D
7413E868	57	PUSH EDI
7413E869	0FB75E 02	MOVZX EBX,WORD PTR DS:[ESI+2]
7413E86D	8B55 AC	MOV EDX,DWORD PTR SS:[EBP-54]

Y veamos si guardo el serial falso en la variable local que usara

401DFB: 0d VCallHresult get__ipropTEXTEDIT
401E00: 3e FLdZeroAd local_008C

Como antes lo guardara en EBP-8c

0012F444	00000000	
0012F448	00000000	
0012F44C	00000000	
0012F450	00000000	
0012F454	0015CA94	UNICODE "989898"
0012F458	00CFBCC4	
0012F45C	0012F4EC	
0012F460	0012F5C8	
0012F464	00CFB85C	
0012F468	0012F4E8	
0012F46C	FFFFFFFF	

Bueno ya llegamos a donde ingresa nuestro serial falso, vemos como conociendo los opcodes no es necesario tracear todo el programa podemos ir poniendo BPM y llegando a las partes calientes sin necesidad de tracear todo, en la parte anterior traceamos todo para que entiendan el mecanismo de cómo funciona PCODE pero normalmente no necesitamos hacer eso, ya lo veremos en la próxima parte cuando crackeemos un extenso programa y localicemos solo la zona caliente que nos interesa y miremos por allí.

401E0F: Lead0/ef ConcatVar
401E13: Lead0/40 NeVarBool
401E15: 1a FFree1Ad **local_0088**
401E18: 36 FFreeVar **local_00CC local_00AC**
401E1F: 1c BranchF: **401E59**

Luego viene esto que tiene todo el aspecto de comparación, luego liberación de la variables locales con FREE, y salto condicional que como vemos va a 401e59 que es el rtcMsgBox de clave correcta y si no salta va al cartel de Clave no Valida_

```

401E0F: Lead0/ef ConcatVar
401E13: Lead0/40 NeVarBool
401E15: 1a FFree1Ad      local_0088
401E18: 36 FFreeVar      local_00CC local_00AC
401E1F: 1c BranchF:      401E59
401E22: 27 LitVar_Missing
401E25: 27 LitVar_Missing
401E28: 3a LitVarStr:      { local_00BC } P-Code
401E2D: 4e FStVarCopyObj local_00CC
401E30: 04 FLdRfVar      local_00CC
401E33: f5 LitI4:      0x10 16 {...}
401E38: 3a LitVarStr:      { local_009C } Clave No Válida!
401E3D: 4e FStVarCopyObj local_00AC
401E40: 04 FLdRfVar      local_00AC
401E43: 0a ImpAdCallIFPR4: rtcMsgBox
401E48: 36 FFreeVar      local_00AC local_00CC local_00EC local_010C
401E53: 1e Branch:      401e8c
401E56: 1e Branch:      401e8c
401E59: 27 LitVar_Missing
401E5C: 27 LitVar_Missing
401E5F: 3a LitVarStr:      { local_00BC } P-Code
401E64: 4e FStVarCopyObj      local_00CC
401E67: 04 FLdRfVar      local_00CC
401E6A: f5 LitI4:      0x30 48 {...}
401E6F: 3a LitVarStr:      { local_009C } Clave Correcta!!
401E74: 4e FStVarCopyObj      local_00AC
401E77: 04 FLdRfVar      local_00AC
401E7A: 0a ImpAdCallIFPR4:      rtcMsgBox
401E7E: 36 FFreeVar      local_00AC local_00CC local_00EC local_010C

```

Allí lo vemos claro, la posible comparación y el salto condicional que decide entre ir a CLAVE NO VALIDA o CLAVE CORRECTA, pues pongamos un BPM ON ACCESS allí, a ver si vemos algo.

401E0F: Lead0/ef ConcatVar
401E13: Lead0/40 NeVarBool

Address	Hex dump	ASCII	
00401E0F	FB EF		
00401E17	FF 36		
00401E1F	1C 99		
00401E27	FF 3A		
00401E2F	FF 04		
00401E37	00 3A		
00401E3F	FF 04		
00401E47	00 36		
00401E4F	14 FF		
00401E57	CC 01		
00401E5F	3A 44		
00401E67	04 34		
00401E6F	3A 64		
00401E77	64 54		

Backup	
Copy	
Binary	
Label	:
Breakpoint	Memory, on access
Search for	Memory, on write

Bueno demos RUN

7413D9AA	8A46 02	MOV AL,BYTE PTR DS:[ESI+2]
7413D9AD	83C6 03	ADD ESI,3
7413D9B0	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]
7413D9B7	8B45 08	MOV EAX,DWORD PTR SS:[EBP+8]
7413D9BA	8945 B4	MOV DWORD PTR SS:[EBP-4C],EAX
7413D9BD	33C0	XOR EAX,EAX
7413D9BF	8A06	MOV AL,BYTE PTR DS:[ESI]
7413D9C1	46	INC ESI
7413D9C2	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]
7413D9C9	0FBF06	MOVSX EAX,WORD PTR DS:[ESI]
7413D9CC	5B	POP EBX
7413D9CD	66:891C28	MOV WORD PTR DS:[EAX+EBP],BX
7413D9D1	33C0	XOR EAX,EAX
7413D9D3	8A46 02	MOV AL,BYTE PTR DS:[ESI+2]
7413D9D6	83C6 03	ADD ESI,3
7413D9D9	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]
7413D9E0	0FBF06	MOVSX EAX,WORD PTR DS:[ESI]
7413D9E3	8F0428	POP DWORD PTR DS:[EAX+EBP]
7413D9E6	33C0	XOR EAX,EAX
7413D9E8	8A46 02	MOV AL,BYTE PTR DS:[ESI+2]
7413D9EB	83C6 03	ADD ESI,3
7413D9EE	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]
7413D9F0	0FBF06	MOVSX EAX,WORD PTR DS:[ESI]

DS:[00401E0F]=FB
AL=00

Allí paro, en el primer opcode, traceemos hasta el segundo que es el que realizara la operación

7413EC87	33C0	XOR EAX,EAX
7413EC89	8A06	MOV AL,BYTE PTR DS:[ESI]
7413EC8B	46	INC ESI
7413EC8C	FF2485 94F1137	JMP DWORD PTR DS:[EAX*4+7413F194]
7413EC93	33C0	XOR EAX,EAX
7413EC95	8A06	MOV AL,BYTE PTR DS:[ESI]
7413EC97	46	INC ESI
7413EC98	FF2485 94F5137	JMP DWORD PTR DS:[EAX*4+7413F594]
7413EC9F	33C0	XOR EAX,EAX
7413ECA1	8A06	MOV AL,BYTE PTR DS:[ESI]
7413ECA3	46	INC ESI
7413ECA4	FF2485 94F9137	JMP DWORD PTR DS:[EAX*4+7413F994]
7413ECAB	33C0	XOR EAX,EAX
7413ECAD	8A06	MOV AL,BYTE PTR DS:[ESI]
7413ECAF	46	INC ESI
7413ECB0	FF2485 94FD137	JMP DWORD PTR DS:[EAX*4+7413FD94]
7413ECB7	33C0	XOR EAX,EAX
7413ECB9	8A06	MOV AL,BYTE PTR DS:[ESI]
7413ECBB	46	INC ESI
7413ECBC	83F8 33	CMP EAX,33
7413ECBF	0F87 D37C0000	JAE MSUBUM50.74146998
7413ECC5	FF2485 9401147	JMP DWORD PTR DS:[EAX*4+74140194]
7413ECC6	8B45 08	MOV EAX,DWORD PTR SS:[EBP+8]

DS:[00401E10]=EF
AL=00

Address	Hex dump	ASCII
00401E0F	FB EF 54 FF FB 40 1A 78	'!T'!@+x
00401E17	FF 36 04 00 34 FF 54 FF	4.4 T

Allí esta el segundo OPCODE es EF

Vemos en la lista de opcode

FB EF 6BAB 25AD 2

FB F0 6B99 259B 0 vbaStrCat

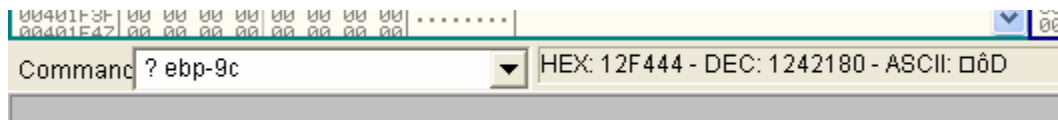
FB F1 C423 B981 0 Push [FC0D134]

Que FB EF no nos aclara que es, igual el EXDEC nos decia algo de concatenar variables, veamos.

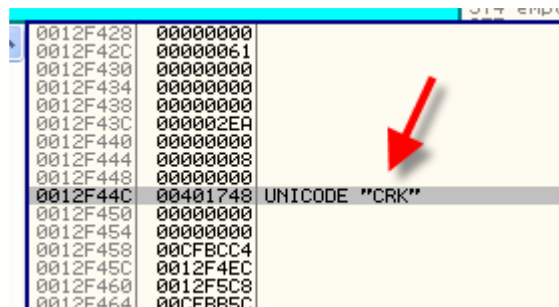
401E07: 3a LitVarStr: (local_009C) CRK

401E0C: 04 FLdRfVar local_018C
 401E0F: Lead0/ef ConcatVar
 401E13: Lead0/40 NeVarBool

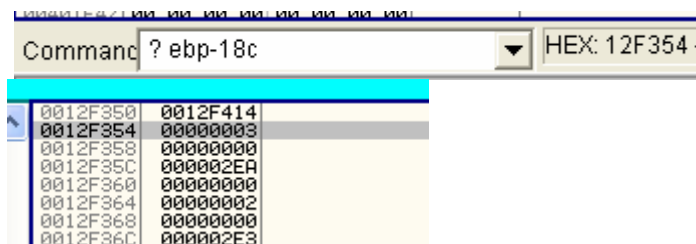
Vemos que justo antes de este opcode trabaja con dos variables locales y en una tiene la STRING CRK y hay otra que esta en EBP-018C veamos que hay en cada variable que menciona.



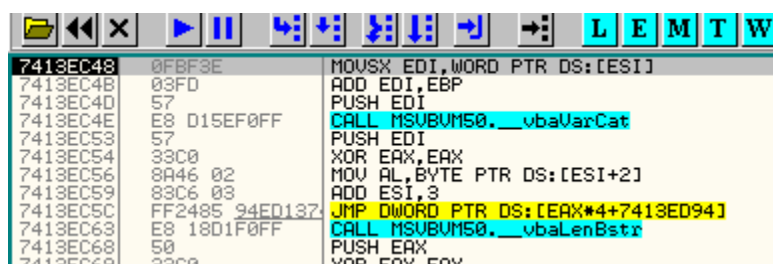
La primera de las dos variables es EBP-9c que en mi maquina es 12F444 y allí esta como dice el EXDEC



Y en la otra sabemos que en el caso de variables hay primero un byte que indica el tipo en este caso el 3 y mas abajo esta la variable realmente que será 2EA.



Entremos en el OPCODE



Allí lee los parámetros

Registers (FPU)		
EAX	000000EF	
ECX	0015CA94	UNIC
EDX	00401824	clavi
EBX	FFFF0008	
ESP	0012F348	
EBP	0012F4E0	
ESI	00401E11	clavi
EDI	0012F434	
EIP	7413EC4D	MSUB
C	1	ES 0023 32bi
P	0	CS 001B 32bi
D	0	SS 0023 32bi

Que luego de sumarle EBP se transforma en 12f434

7413EC48	0FBF3E	MOVX EDI,WORD PTR DS:[ESI]
7413EC4B	03FD	ADD EDI,EBP
7413EC4D	57	PUSH EDI
7413EC4E	E8 D15EF0FF	CALL MSUBUM50. vbaVarCat
7413EC53	57	PUSH EDI
7413EC54	33C0	XOR EAX,EAX

Llegamos a la api vbaVarCat con tres argumentos en el stack

0012F344	0012F434	Arg1 = 0012F434
0012F348	0012F354	Arg2 = 0012F354
0012F34C	0012F444	Arg3 = 0012F444
0012F350	0012F414	
0012F354	00000003	

Veamos que hay en cada uno, recordemos que cuando trabaja con Var como en este caso, hay que mirar una pequeña estructura ya que el primer byte como vimos antes es el tipo de variable, etc.

Address	Hex dump	ASCII
0012F434	00 00 00 00 00 00 00 00
0012F43C	EA 02 00 00 00 00 00 00	00.....
0012F444	08 00 00 00 00 00 00 00	00.....
0012F44C	48 17 40 00 00 00 00 00	H0.....
0012F454	00 00 00 00 C4 BC CF 00

Ese es el primer argumento

Address	Hex dump	ASCII
0012F354	03 00 00 00 00 00 00 00	00.....
0012F35C	EA 02 00 00 00 00 00 00	00.....
0012F364	02 00 00 00 00 00 00 00	00.....
0012F36C	E3 02 00 00 00 00 00 00	00.....
0012F374	02 00 00 00 00 00 00 00	00.....
0012F37C	61 00 00 00 00 00 00 00	a.....

El segundo con el 3 al inicio y 02EA como valor

Address	Hex dump	ASCII
0012F444	08 00 00 00 00 00 00 00	00.....
0012F44C	48 17 40 00 00 00 00 00	H0.....
0012F454	00 00 00 00 C4 BC CF 00
0012F45C	EC F4 12 00 C8 F5 12 00
0012F464	5C BB CF 00 E8 F4 12 00

Y el tercero con el 8 al inicio nos muestra un puntero 401748 a la string CRK

Address	Hex dump	ASCII
00401748	43 00 52 00 4B 00 00 00	C.R.K...
00401750	20 00 00 00 43 00 6C 00	...C.L.
00401758	61 00 76 00 65 00 20 00	a.v.e..
00401760	4E 00 6F 00 20 00 56 00	N.o..U.

Bueno hagamos la ensalada de variables a ver que queda jeje. Si entramos dentro de la api vbaVarCat llegamos a una api interior que nos muestra más claro que va a unir.

74044B46	0F84 95F00500	JE MSUBVM50.740A3BE1
74044B4C	837D F4 FF	CMP DWORD PTR SS:[EBP-C],-1
74044B50	0F84 B4F00500	JE MSUBVM50.740A3C0A
74044B56	FF75 F4	PUSH DWORD PTR SS:[EBP-C]
74044B59	FF75 F8	PUSH DWORD PTR SS:[EBP-8]
74044B5C	E8 2ED5FFFF	CALL MSUBVM50.vbaStrCat
74044B61	8B75 08	MOV ESI,DWORD PTR SS:[EBP+8]
74044B64	837D FC 00	CMP DWORD PTR SS:[EBP-4],0
74044B68	66:C706 0800	MOV WORD PTR DS:[ESI],8

0012F324	0015D83C	Unicode "746"
0012F328	00401748	Unicode "CRK"
0012F32C	00401E11	clave2.00401E11
0012F330	00401748	Unicode "CRK"
0012F334	0015D83C	Unicode "746"
0012F338	0015A828	
0012F33C	0012F4E0	
0012F340	7413EC53	RETURN to MSUBVM50.
0012F344	0012F434	
0012F348	0012F354	

O sea que va a unir CRK con 746 ahora vemos que el parámetro que en la vbaVarCat era 02EA

Address	Hex dump	ASCII
0012F354	03 00 00 00 00 00 00 00
0012F35C	EA 02 00 00 00 00 00 00	02.....
0012F364	02 00 00 00 00 00 00 00	02.....
0012F36C	E3 02 00 00 00 00 00 00	03.....
0012F374	02 00 00 00 00 00 00 00	02.....
0012F37C	61 00 00 00 00 00 00 00	a.....

Lo transforma a string ya que 02EA es

0015D964	FC 00 FD 00 FE 00 FF 00
0015D96C	E0 00 C0 00 80 00 B0 00	0.....
0015D974	C0 00 50 00 B0 00 C0 00	0.....
Command ? 02ea		
HEX: 2EA - DEC: 746 - ASCII: 0ê		
Breakpoint at MSVBVM50.74044B5C		

O sea que la api vbaVarCat toma una string en este caso y una variable numérica que convierte a string y las unirá.

Sigamos traceando con f8

Al llegar al RET de la api vemos

Registers (FPU)		
EAX	0015D88C	Unicode "CRK746"
ECX	00000000	
EDX	0015D608	
EBX	FFFF0000	
ESP	0012F320	
EBP	0012F33C	
ESI	00401E11	clave2.00401E11
EDI	0012F434	
EIP	740420FB	MSUBVM50.740420FB
C 0	ES 0023	32bit 0(FFFFFFFF)
P 0	CS 001B	32bit 0(FFFFFFFF)
A 0	SS 0023	32bit 0(FFFFFFFF)
Z 0	DS 0023	32bit 0(FFFFFFFF)
S 0	FS 003B	32bit 7FFDF000(FFF)

Como unió y formo una linda string con ambas variables.

74044B50	0F84 B4F00500	JE MSUBUM50.740A3C0A
74044B56	FF75 F4	PUSH DWORD PTR SS:[EBP-C]
74044B59	FF75 F8	PUSH DWORD PTR SS:[EBP-8]
74044B5C	E8 2ED5FFFF	CALL MSUBUM50.vbaStrCat
74044B61	8B75 08	MOV ESI,DWORD PTR SS:[EBP+8]
74044B64	837D FC 00	CMP DWORD PTR SS:[EBP-4],0
74044B68	66:C706 0800	MOV WORD PTR DS:[ESI],8
74044B6D	8946 08	MOV DWORD PTR DS:[ESI+8],EAX
74044B70	75 09	JNZ SHORT MSUBUM50.74044B7B
74044B72	8BC6	MOV EAX,ESI
74044B74	5E	POP ESI
74044B75	8BE5	MOV ESP,EBP
74044B77	5D	POP EBP
74044B78	C2 0C00	RET 0C
74044B7B	8B45 FC	MOV EAX,DWORD PTR SS:[EBP-4]
74044B7E	66:8378 50 08	CMP WORD PTR DS:[EAX+50],8
74044B83	75 12	JNZ SHORT MSUBUM50.74044B97
74044B85	FF70 58	PUSH DWORD PTR DS:[EAX+58]
74044B88	FF15 88190474	CALL DWORD PTR DS:[<OLEAUT32.#6>]
74044B8E	8B4D FC	MOV ECX,DWORD PTR SS:[EBP-4]
74044B91	66:C741 50 0000	MOV WORD PTR DS:[ECX+50],0
74044B97	8B45 FC	MOV EAX,DWORD PTR SS:[EBP-4]
74044B99	66:8378 50 08	CMP WORD PTR DS:[EAX+50],8
EAX=0015D88C, (UNICODE "CRK746")		
Stack DS:[0012F43C]=000002EA		
Address	Hex dump	ASCII

Y como antes de llegar al siguiente opcode la guarda en el primer argumento.

Address	Hex dump	ASCII
0012F434	08 00 00 00 00 00 00 00
0012F43C	8C 08 15 00 00 00 00 00	iS.....
0012F444	08 00 00 00 00 00 00 00
0012F44C	48 17 40 00 00 00 00 00	H@.....

Vemos como el primer argumento ahora es del tipo 8 o sea string, y que apunta a 15d88c que esta la string concatenada.

Address	Hex dump	ASCII
0015D88C	43 00 52 00 4B 00 37 00	C.R.K.7.
0015D894	34 00 36 00 00 00 14 20	4.6...¶
0015D89C	DC 02 22 21 61 01 3A 20	¶"1a0:
0015D8A4	53 01 9D 00 EB 00 05 00	S00.0.¶.
0015D8AC	A0 10 A1 00 78 01 15 00	¶i.h0\$.


Pues bien ahora debería venir la comparación, lleguemos hasta el siguiente OPCODE.

7413EC53	57	PUSH EDI
7413EC54	33C0	XOR EAX,EAX
7413EC56	8A46 02	MOV AL,BYTE PTR DS:[ESI+2]
7413EC59	83C6 03	ADD ESI,3
7413EC5C	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]
7413EC63	E8 18D1F0FF	CALL MSUBUM50.vbaLenBstr
7413EC68	50	PUSH EAX
7413EC69	33C0	XOR EAX,EAX
7413EC6B	8A06	MOV AL,BYTE PTR DS:[ESI]
7413EC6D	46	INC ESI
7413EC6E	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]
7413EC75	E8 1AD1F0FF	CALL MSUBUM50.vbaInStr
7413EC7A	50	PUSH EAX
7413EC7B	33C0	XOR EAX,EAX
7413EC7D	8A06	MOV AL,BYTE PTR DS:[ESI]
7413EC7F	46	INC ESI
7413EC80	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]
7413EC87	33C0	XOR EAX,EAX
7413EC89	8A06	MOV AL,BYTE PTR DS:[ESI]
DS:[00401E13]=FB		
AL=00		

401E13: Lead0/40 NeVarBool

Como es doble y es FB40 traceo hasta que lee el segundo OPCODE

Address	Hex dump	ASCII
00401E13	FB 40 1A 78 FF 36 04 00	!@+x 6+
00401E1B	34 FF 54 FF 1C 99 01 27	4 T L00*
00401E23	F4 FE 27 14 FF 3A 44 FF	¶¶'¶:D
00401E2B	02 00 4E 34 FF 04 34 FF	0.N4 ¶4
00401E33	F5 10 00 00 00 3A 64 FF	S¶...:d
00401E3B	0A 00 4E 54 FF 04 54 FF	..NT ¶T

7413EC7B	33C0	XOR EAX,EAX
7413EC7D	8A06	MOV AL,BYTE PTR DS:[ESI]
7413EC7F	46	INC ESI
7413EC80	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]
7413EC87	33C0	XOR EAX,EAX
7413EC89	8A06	MOV AL,BYTE PTR DS:[ESI]
7413EC8B	46	INC ESI
7413EC8C	FF2485 94F1137	JMP DWORD PTR DS:[EAX*4+7413F194]
7413EC93	33C0	XOR EAX,EAX
7413EC95	8A06	MOV AL,BYTE PTR DS:[ESI]
7413EC97	46	INC ESI
7413EC99	FF2485 94FF137	JMP DWORD PTR DS:[EAX*4+7413FF94]
DS:[00401E14]=40 ('@')		
AL=00		
		
Address	Hex dump	ASCII
00401E13	FB 40 1A 78 FF 36 04 00	'@* 6+
00401E1B	34 FF 54 FF 1C 99 01 27	4 T L00'
00401E23	F4 FE 27 14 FF 3A 44 FF	71.'1 :D

Por supuesto la lista de Microsoft no dice nada sobre el, así que traceemos el opcode a ver que hace, jeje.

7413EBF3	BB 84ED1374	MOV EBX,MSUBUM50.7413ED84
7413EBF8	EB 07	JMP SHORT MSUBUM50.7413EC01
7413EBFA	6A 00	PUSH 0
7413EBFC	BB 74ED1374	MOV EBX,MSUBUM50.7413ED74
7413EC01	E8 8E180000	CALL MSUBUM50.74140494
7413EC06	FF3483	PUSH DWORD PTR DS:[EBX+EAX*4]
7413EC09	33C0	XOR EAX,EAX
7413EC0B	8A06	MOV AL,BYTE PTR DS:[ESI]
7413EC0D	46	INC ESI
7413EC0E	FF2485 94ED137	JMP DWORD PTR DS:[EAX*4+7413ED94]

Vemos que el opcode solo tiene una call y luego enseguida termina veamos los argumentos de esta call.

0012F348	00000000	Arg1 = 00000000
0012F34C	0012F434	Arg2 = 0012F434
0012F350	0012F414	Arg3 = 0012F414
0012F354	00000003	
0012F358	00000000	
0012F35C	000002EA	

El primero argumento es cero, luego el segundo es 12f434 miremos que hay alli

Address	Hex dump	ASCII
0012F434	08 00 00 00 00 00 00 00
0012F43C	0C 08 15 00 00 00 00 00	i\$.
0012F444	08 00 00 00 00 00 00 00
0012F44C	48 17 40 00 00 00 00 00	H#0.

Bueno 08 nos indica que es una string miremos cual es ya que nos muestra que 15d88c apunta a ella.

Address	Hex dump	ASCII
0015D88C	43 00 52 00 4B 00 37 00	C.R.K.7.
0015D894	34 00 36 00 00 00 14 20	4.6...1
0015D89C	DC 02 22 21 61 01 3A 20	"!a0: .
0015D8A4	52 01 90 00 50 00 05 00	Sam . .

Allí esta la string veamos el otro argumento.

Address	Hex dump	ASCII
0012F414	08 80 00 00 00 00 00 00	8.
0012F41C	94 CA 15 00 14 F4 12 00	0=3.11#.
0012F424	00 00 00 00 00 00 00 00
0012F42C	74 00 00 00 00 00 00 00

Y en este caso 15ca94 apunta a la string de nuestro serial falso.

Address	Hex dump	ASCII
0015CA94	39 00 38 00 39 00 38 00	9.8.9.8.
0015CA9C	39 00 38 00 00 00 00 00	9.8.
0015CAA4	46 00 6F 00 72 00 6D 00	F.o.r.m.
0015CAAC	0A 0A FF FF 05 0A 05 0A	..-..

Pues parece que va a comparar ambas strings.

7413EBF3	BB 84ED1374	MOV EBX,MSUBUM50.7413ED84
7413EBF8	EB 07	JMP SHORT MSUBUM50.7413EC01
7413EBFA	6A 00	PUSH 0
7413EBFC	BB 74ED1374	MOV EBX,MSUBUM50.7413ED74
7413EC01	E8 8E180000	CALL MSUBUM50.74140494
7413EC06	FF3483	PUSH DWORD PTR DS:[EBX+EAX*4]
7413EC09	33C0	XOR EAX,EAX
7413EC0B	8A06	MOV AL,BYTE PTR DS:[ESI]
7413EC0D	46	INC ESI

Pongamos un BP para sacarnos la duda, pasemos la call con f8 y lleguemos al otro opcode

Vemos que en el stack quedo FFFFFFFF posiblemente porque las strings no son iguales, anotemos el posible serial bueno y probemos hasta que pare en el BP que pusimos nuevamente aquí.

Curso sobre P-Code. Por ...

Usuario:

Serie Núm.:

Registrar

Al apretar registrar

7413EBFA	6A 00	PUSH 0
7413EBFC	BB 74ED1374	MOV EBX,MSUBUM50.7413ED74
7413EC01	E8 8E180000	CALL MSUBUM50.74140494
7413EC06	FF3483	PUSH DWORD PTR DS:[EBX+EAX*4]
7413EC09	33C0	XOR EAX,EAX
7413EC0B	8A06	MOV AL,BYTE PTR DS:[ESI]
7413EC0D	46	INC ESI

Llego de nuevo a la call la paso con f8 y llego al siguiente opcode como antes

0012F350	00000000
0012F354	00000003
0012F358	00000003
0012F35C	00000003
0012F360	00000000
0012F364	00000000
0012F368	00000000
0012F36C	00000000

Vemos que en este caso el resultado es cero al ser las strings iguales.

Curso sobre P-Code. Por ...

Usuario:

Serie Núm.:

P-Code

!

Clave Correcta!!

Aceptar

Como ven sin necesidad de tracear todo, solamente encontrando la parte caliente y traceando algún opcode desconocido, me sirvió para encontrar el serial de este programa.

Para los que quieran el listado mas detallado aquí va un detalle que me enviaron.

```

401CC0: 04 FLdRfVar          local_008C
401CC3: 21 FLdPrThis           ; Load reference pointer into item
pointer.
401CC4: 0f VCallAd            text           ; Accede a metodo ITEM DESCRIPTOR TABLE
401CC7: 19 FStAdFunc          local_0088
401CCA: 08 FLdPr              local_0088
401CCD: 0d VCallHresult       get__ipropTEXTEDIT ; Lee contenido de textbox
401CD2: 6c ILdRf              local_008C      ; NOMBRE
401CD5: 1b LitStr:            &                ; Empuja cadena a la pila
401CD8: Lead0/30 EqStr        ; Compara dos cadenas
401CDA: 2f FFree1Str          local_008C
401CDD: 1a FFree1Ad           local_0088
401CE0: 1c BranchF:           401CE6           ; Salta si comparacion falsa
401CE3: 1e Branch:            401e8c          ; Salto incondicional
401CE6: 04 FLdRfVar          local_008C
401CE9: 21 FLdPrThis
401CEA: 0f VCallAd            text           ; Accede a metodo ITEM DESCRIPTOR
TABLE
401CED: 19 FStAdFunc          local_0088
401CF0: 08 FLdPr              local_0088
401CF3: 0d VCallHresult       get__ipropTEXTEDIT ; Lee contenido de textbox
401CF8: 6c ILdRf              local_008C      ; NOMBRE
401CFB: 4a FnLenStr
401CFC: f5 LitI4:            0x6 6  (....)    ; Pasa entero de 4 bytes (6)
401D01: d1 LtI4              ; ¿Comparacion menor que?
401D02: 2f FFree1Str          local_008C
401D05: 1a FFree1Ad           local_0088
401D08: 1c BranchF:           401D3F           ; Salta si falso (es >= 6)
401DOB: 27 LitVar_Missing
401DOE: 27 LitVar_Missing
401D11: 3a LitVarStr:         ( local_00BC ) P-Code
401D16: 4e FStVarCopyObj      local_00CC
401D19: 04 FLdRfVar           local_00CC
401D1C: f5 LitI4:            0x40 64 (...@)
401D21: 3a LitVarStr:         ( local_009C ) Mínimo 6 caracteres
401D26: 4e FStVarCopyObj      local_00AC
401D29: 04 FLdRfVar           local_00AC
401D2C: 0a ImpAdCallFPR4:    _rtcMsgBox
401D31: 36 FFreeVar           local_00AC local_00CC local_00EC local_010C
401D3C: 1e Branch:            401e8c          ; Si < 6 caracteres a la calle
401D3F: 04 FLdRfVar          local_008C
401D42: 21 FLdPrThis
401D43: 0f VCallAd            text
401D46: 19 FStAdFunc          local_0088
401D49: 08 FLdPr              local_0088
401D4C: 0d VCallHresult       get__ipropTEXTEDIT ; Lee contenido de textbox
401D51: 3e FLdZeroAd          local_008C      ; NOMBRE
401D54: 46 CVarStr            local_00AC
401D57: 04 FLdRfVar           local_00CC
401D5A: 0a ImpAdCallFPR4:    _rtcLowerCaseVar ; Convierte a minusculas
401D5F: 04 FLdRfVar           local_00CC
401D62: 04 FLdRfVar           local_00EC
401D65: 0a ImpAdCallFPR4:    _rtcTrimVar
401D6A: 04 FLdRfVar          local_00EC
401D6D: Lead1/f6 FStVar      local_011C
401D71: 1a FFree1Ad           local_0088
401D74: 36 FFreeVar           local_00AC local_00CC
401D7B: 04 FLdRfVar           local_011C
401D7E: Lead0/eb FnLenVar
401D82: Lead1/f6 FStVar      local_012C

```

```

401D86: 28 LitVarI2:          ( local_00BC ) 0x1  (1)
401D8B: 04 FLdRfVar            local_013C
401D8E: 04 FLdRfVar            local_012C
401D91: Lead3/68 ForVar:      (when done) 401DE0      , Inicio bucle for next
401D97: 28 LitVarI2:          ( local_00AC ) 0x1  (1)
401D9C: 04 FLdRfVar            local_013C
401D9F: Lead1/22 CI4Var
401DA1: 04 FLdRfVar            local_011C
401DA4: 04 FLdRfVar            local_00CC
401DA7: 0a ImpAdCallFPR4:     _rtcMidCharVar      ; Va tomando los caracteres del ...
401DAC: 04 FLdRfVar            local_00CC      ; ... nombre
401DAF: Lead2/fe CStrVarVal   local_008C
401DB3: 0b ImpAdCallI2        _rtcAnsiValueBstr    ; convierte valor carácter a hexadec
401DB8: 44 CVarI2              local_00BC
401DBB: Lead1/f6 FstVar        local_016C
401DBF: 2f FFree1Str           local_008C
401DC2: 36 FFreeVar            local_00AC local_00CC
401DC9: 04 FLdRfVar            local_017C
401DCC: 04 FLdRfVar            local_016C
401DCF: Lead0/94 AddVar     local_00AC
401DD3: Lead1/f6 FstVar        local_017C
401DD7: 04 FLdRfVar            local_013C
401DDA: Lead3/7e NextStepVar: (continue) 401D97      ; Fin bucle for-next?
401DE0: 04 FLdRfVar            local_017C
401DE3: 04 FLdRfVar            local_012C
401DE6: Lead0/94 AddVar     local_00AC
401DEA: Lead1/f6 FstVar        local_018C
401DEE: 04 FLdRfVar            local_008C
401DF1: 21 FLdPrThis
401DF2: 0f VCallAd            text
401DF5: 19 FStAdFunc           local_0088
401DF8: 08 FLdPr               local_0088
401DFB: 0d VCallHresult       get__ipropTEXTEDIT    ; Lee contenido de textbox
401E00: 3e FLdZeroAd           local_008C      ; SERIAL
401E03: 46 CVarStr              local_00CC
401E06: 5d HardType
401E07: 3a LitVarStr:          ( local_009C ) CRK
401E0C: 04 FLdRfVar            local_018C
401E0F: Lead0/ef ConcatVar
401E13: Lead0/40 NeVarBool
401E15: 1a FFree1Ad            local_0088
401E18: 36 FFreeVar            local_00CC local_00AC
401E1F: 1c BranchF:            401E59
401E22: 27 LitVar_Missing
401E25: 27 LitVar_Missing
401E28: 3a LitVarStr:          ( local_00BC ) P-Code
401E2D: 4e FStVarCopyObj       local_00CC
401E30: 04 FLdRfVar            local_00CC
401E33: f5 LitI4:              0x10 16 (....)
401E38: 3a LitVarStr:          ( local_009C ) Clave No Válida!
401E3D: 4e FStVarCopyObj       local_00AC
401E40: 04 FLdRfVar            local_00AC
401E43: 0a ImpAdCallFPR4:     _rtcMsgBox
401E48: 36 FFreeVar            local_00AC local_00CC local_00EC local_010C
401E53: 1e Branch:              401e8c
401E56: 1e Branch:              401e8c
401E59: 27 LitVar_Missing
401E5C: 27 LitVar_Missing
401E5F: 3a LitVarStr:          ( local_00BC ) P-Code
401E64: 4e FStVarCopyObj       local_00CC
401E67: 04 FLdRfVar            local_00CC
401E6A: f5 LitI4:              0x30 48 (...0)
401E6F: 3a LitVarStr:          ( local_009C ) Clave Correcta!!
401E74: 4e FStVarCopyObj       local_00AC
401E77: 04 FLdRfVar            local_00AC
401E7A: 0a ImpAdCallFPR4:     _rtcMsgBox
401E7F: 36 FFreeVar            local_00AC local_00CC local_00EC local_010C
401E8A: Lead1/c8 End

```

401E8C: 13 ExitProcHresult

Bueno allí esta claro todo el trabajo del crackme.

Es bueno saber hacerlo con OLLY, porque hay programas que se protegen contra el WKT contra el EXDEC y al menos con OLLY siempre podremos mientras el programa corra en el mismo, lo cual como ya vimos con los plugins que hay para ocultar OLLYDBG, es bastante sencillo salvo muy contadas excepciones.

Tomemos el crackme adjunto nags1 que nos pide eliminar la nag inicial, veamos su listado en EXDEC.

Proc: 401a40

401A14: 27 LitVar_Missing

401A17: 27 LitVar_Missing

401A1A: 27 LitVar_Missing

401A1D: f5 LitI4: 0x0 0 (....)

401A22: 3a LitVarStr: (local_0094) NAG

401A27: 4e FStVarCopyObj local_00A4

401A2A: 04 FLdRfVar local_00A4

401A2D: 0a ImpAdCallFPR4: _rtcMsgBox

401A32: 36 FFreeVar local_00A4 local_00C4 local_00E4 local_0104

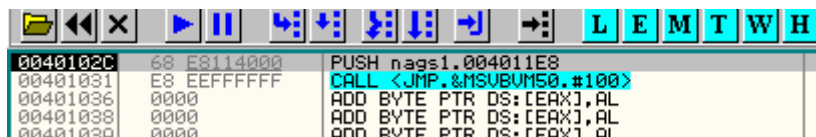
401A3D: 13 ExitProcHresult

Proc: 401958

401954: Lead1/c8 End

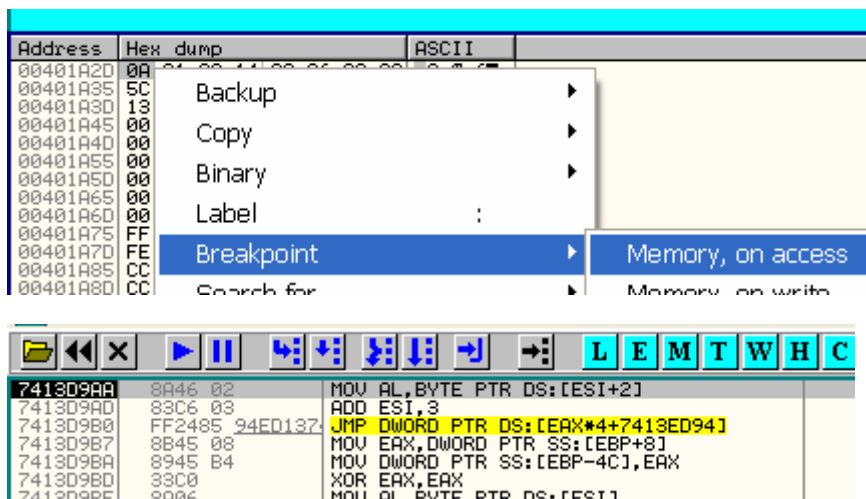
401956: 13 ExitProcHresult

Vemos que la famosa NAG es solo un rtcMsgBox pero por si no saben aquí en PCODE no existe el NOP, jeje así que hay que nopear con OPCODES que no afecten el resto del programa,

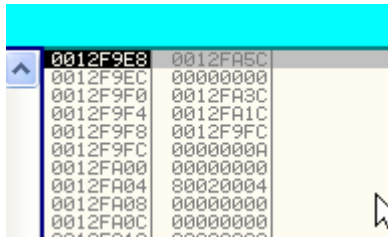


Arranco el programa voy a poner un BPM en el opcode al rtcMsgBox

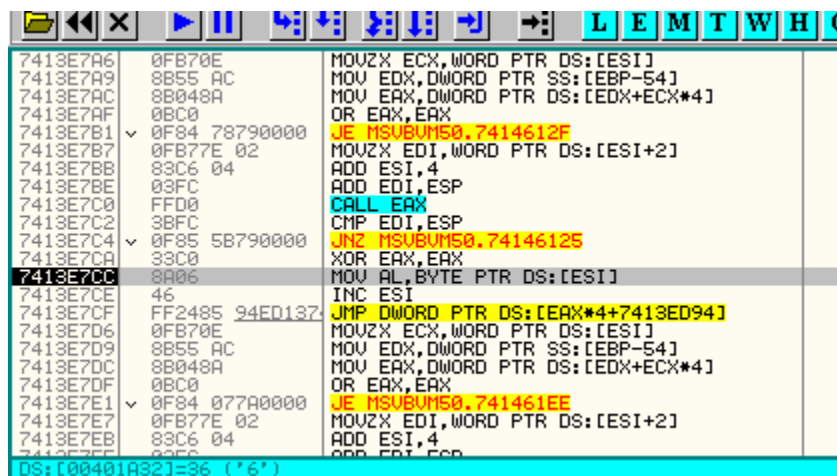
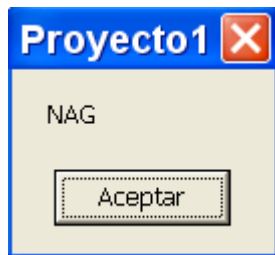
401A2D: 0a ImpAdCallFPR4: _rtcMsgBox



Ali para cuando lee el opcode

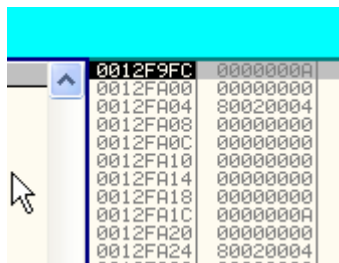


Vemos que el stack esta en 12f9e8, lleguemos hasta el otro opcode por supuesto saldrá la nag que tendremos que aceptar antes de llegar al siguiente opcode.



Allí llego al siguiente opcode luego de pasar por el CALL EAX que es la llamada a la api rtcMsgBox.

El stack esta ahora en 12f9fc



O sea para que quede igual deberíamos hacer varios pops y no nos cabe allí, pues probaremos con un PUSH solo usaremos f5.

F5 5CBE 1377 4 1 4 Push imm#4

Ya que tiene 4 parámetros de largo igual que 0A la que vamos a reemplazar

0A 664E 1F30 4 2 2 2

Probemos cambiemos 0A por F5 y pongamos todos los parámetros a cero

Address	Hex dump	ASCII
00401A20	F5 00 00 00 00 36 08 00	\$...6.
00401A35	5C FF 3C FF 1C FF FC FE	\<L?
00401A3D	13 00 00 80 16 00 00 04	!..C-@.
00401A45	00 00 00 2C 00 00 00 00	.C...\$..

Allí esta el siguiente opcode es el 36 como nos mostraba el EXDEC, siempre debemos fijarnos que el opcode que reemplaza tenga la misma cantidad de parámetros que el que vamos a reemplazar, así no hay problema y seguirá ejecutando el siguiente, guardemos los cambios.

Address	Hex dump	ASCII
00401A20	F5 00 00 00 00 36 08 00	\$...6.
00401A35	5C FF 3C FF 1C FF FC FE	\<L?
00401A3D	13 00 00 80 16 00 00 04	!..C-@.
00401A45	00 00 00 2C 00 00 00 00	.C...\$..

Backup

Undo selection Alt+BkSp

Copy

Binary

Label :

Breakpoint

Search for

Find references Ctrl+R

View executable file

Copy to executable file

Go to

Address	Hex dump	ASCII
00000E20	F5 00 00 00 00 36 08 00	\$...6.
00000E35	5C FF 3C FF 1C FF FC FE	\<L?
00000E3D	13 00 00 80 16 00 00 04	!..C-@.
00000E45	00 00 00 2C 00 00 00 00	.C...\$..
00000E4D	00 00 00 00 00 00 00 00	
00000E55	00 00 00 00 00 00 00 00	
00000E5D	00 00 00 00 00 00 00 00	
00000E65	00 00 00 00 00 00 00 00	
00000E6D	00 00 00 00 00 00 00 00	
00000E75	FF 02 00 1C	
00000E7D	FE 02 00 E9	
00000E85	CC CC CC CC	
00000E8D	CC CC CC 9E	
00000E95	00 00 00 00	
00000E9D	00 00 00 00	
00000EA5	00 00 00 00	
00000EAD	00 00 00 00	
00000EB5	00 00 00 00	
00000EBD	00 00 00 00	
00000EC5	00 00 00 00	

Backup

Copy

Binary

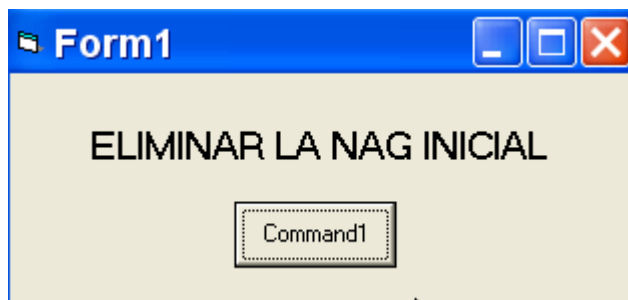
Search for

Save file

Go to offset Ctrl+G

View image in Disassembler

Mis sitios de red		
Nombre:	nags1a.exe	
Tipo:	Executable file (*.exe)	
7413DEB0	00401A20	MOV EAX, DWORD PTR DS:[E
7413DEB1	03FD	ADD EDI, EBP
7413DEB3	66:C707 0800	MOV WORD PTR DS:[EDI], 8



Allí arranca sin la nag inicial, todo es cuestión de probar, quizás en algún caso habrá que usar otro opcode diferente para parchear, ahora como ejercicio me gustaría que quiten la nag del otro crackme que adjunto el nags2.

Y los espero en la parte 31 con el final de PCODE en un programa comercial.

Ricardo Narvaja
08 de febrero de 2006