ANEXO A LA LECCIÓN 6

Tal y como habíamos quedado en la última entrega, se hace necesario un estudio somero de la rutina de chequeo para generar un serial bueno y poder registrar así nuestro programa. Esto sería un crack "limpio", un crack que implica conocimiento y natural al programa, no como los parches, que son cracks "sucios", muy buenos cuando se empieza pero insuficientes cuando ya se tiene algo de nivel.

El código al que habíamos llegado era éste:

```
00005760  2f0a          L1040   MOVE.L   A2,-(A7); carga HotSync en la pila. El fake, en A3
00005762  4e4fa0c7              sysTrapStrLen; calcula tamaño de mi HotSync y lo mete en D0 en hexa
00005766  0c40000a              CMPI.W #10!$a,D0; compara el tamaño de HotSync con 10
0000576a  584f                  ADDQ.W#4,A7; corrige la pila
0000576c  6336                  BLS      L1041; salta, ya que X-Grimator mide 10 (branch less or same)
```

Este trozo de código es si el HotSync es superior a 10 caracteres

```
0000576e  48780005              PEA      $0005.W
00005772  2f0a                  MOVE.L   A2,-(A7)
00005774  486efff4              PEA      -12(A6)
00005778  4e4fa026              sysTrapMemMove
0000577c  2f0a                  MOVE.L   A2,-(A7)
0000577e  4e4fa0c7              sysTrapStrLen
00005782  0640fffb              ADDI.W   #-5,D0
00005786  3600                  MOVE.W   D0,D3
00005788  48780005              PEA      $0005.W
0000578c  48723000              PEA      0(A2,D3.W)
00005790  486efff9              PEA      -7(A6)
00005794  4e4fa026              sysTrapMemMove
00005798  422efffe              CLR.B    -2(A6)
0000579c  7e0a                  MOVEQ    #10,D7
0000579e  4fef001c              LEA      28(A7),A7
000057a2  6016                  BRA      L1042
```

```
000057a4  2f0a          L1041   MOVE.L   A2,-(A7); mi HotSync pasa a la pila
000057a6  4e4fa0c7              sysTrapStrLen; en D0 se mete el tamaño calculado de mi HotSync, en hexa
000057aa  3e00                  MOVE.WD0,D7; D0 pasa a D7 en el último word (xxxx0010 en mi caso)
000057ac  2f0a                  MOVE.L A2,-(A7); HotSync pasa a la pila
000057ae  486efff4              PEA      -12(A6)
000057b2  4e4fa0c5              sysTrapStrCopy; HotSync se mete en D0 en valor ASCII (Así, para el Hot
                                        Sync "Az", sería 41 [que es A] 7A [que es z])
000057b6  4fef000c              LEA      12(A7),A7
000057ba  7800          L1042   MOVEQ    #0,D4; D4 se pone a cero. Es el contador del número
                                        de letras que componen nuestro HotSync
000057bc  7600                  MOVEQ    #0,D3; se pone a cero el contador de cada letra, que es D3
000057be  6048                  BRA      L1044

00005808  b647          L1044   CMP.WD7,D3; mira si la letra que analiza el contador D3 es igual al nú-
                                        mero total de letras de mi HotSync. Así, cuando D3 calcule
                                        la letra 10, coincidirá con la de D7 y acabará el checksum
0000580a  65b4                  BCS      L1043; salta mientres queden letras que analizar

000057c0  7000          L1043   MOVEQ    #0,D0; pone D0 a cero
000057c2  3003                  MOVE.WD3,D0; D3 marca cero en la primera letra, uno en la segunda…y
                                        lo pasa a D0
000057c4  41efff4               LEA      -12(A6),A0
000057c8  7a00                  MOVEQ    #0,D5;  pone D5 a cero
000057ca  1a300800              MOVE.B 0(A0,D0.L),D5;  en D5 se mete el valor ASCII de mi primera le-
                                        tra del HotSync conforme vimos en línea 57b2
000057ce  e248                  LSR.W    #1,D0; los bits de D0 los desplaza un puesto a la derecha en
                                        binario. Si se "cayó" un bit uno, activa el flag Zero. Así,
                                        si tenemos en D0=00000002(hexa) pasaría a ser 00000001
                                        binario, ya que 2 hexa es 10 binario, y el flag no se activa
                                        ya que lo que se "cayó" fue un cero. Esta instrucción per-
                                        mite desplazar de uno a ocho bits (#1 para uno, #2 para
                                        dos...)
000057d0  d040                  ADD.W    D0,D0; D0+D0 y lo guarda en D0
000057d2  3c00                  MOVE.WD0,D6; D0 pasa a D6
000057d4  bc43                  CMP.W    D3,D6;  compara D3 con D6. De no ser igual, D0 pasa a ser FF
```

```
                                                    de ser igual, D0 se convierte en 00
000057d6  56c0                          SNE     D0
000057d8  4400                          NEG.B   D0; 0-D0 (1 bit) y lo guarda en D0
000057da  4880                          EXT.W   D0

        Segunda parte del cálculo, como veis, no es muy diferente la estructura a la primera

000057dc  c0c5                          MULU.WD5,D0; D5 por D0 y lo guarda en D0
000057de  bc43                          CMP.W   D3,D6; compara D3 con D6 y de ser igual, D1 pasa a ser FF. En
                                                    caso contrario, D1 pasa a ser 00
000057e0  57c1                          SEQ     D1
000057e2  4401                          NEG.B   D1; 0-D1 (un bit) y lo guarda en D1
000057e4  4881                          EXT.W   D1

        Tercera  parte del cálculo, como veis, no es muy diferente la estructura a la segunda

000057e6  c2c5                          MULU.WD5,D1; D5 por D1 y lo guarda en D1
000057e8  3403                          MOVE.WD3,D2; mueve D3 y lo pone en D2
000057ea  5242                          ADDQ.W#1,D2; suma uno a D2 y lo guarda en D2
000057ec  c4c1                          MULU.WD1,D2; multiplica D1 por D2 y lo guarda en D2
000057ee  d440                          ADD.W   D0,D2; suma D0 a D2 y lo guarda en D2
000057f0  c4fc0007                      MULU.W#7,D2; multiplica D2 por siete y lo guarda en D2
000057f4  4a43                          TST.W   D3; compara D3 con cero. De ser igual, D0 pasa a ser FF. En
                                                    caso contrario, D0 se pone como 00
000057f6  57c0                          SEQ     D0
000057f8  4400                          NEG.B   D0; 0-D0 (1 bit) y lo guarda en D0
000057fa  4880                          EXT.W   D0

        Tercera parte del cálculo, como veis, no es muy diferente la estructura a la segunda

000057fc  c0fc019d                      MULU.W#413!$19d,D0; D0 multiplicado por 413 (decimal) y guardado en
                                                    D0
00005800  d042                          ADD.W   D2,D0; D0 más D2 y lo guarda en D0
00005802  d044                          ADD.W   D4,D0; D4 más D0 y lo guarda en D0. Suma el cálculo de la letra
                                                    anterior al de ésta.
00005804  3800                          MOVE.WD0,D4;  D0 pasa a D4. Este es el serial total de las letras que lle-
                                                    vemos analizadas.
00005806  5243                          ADDQ.W#1,D3; suma el contador D3 a uno


00005808  b647                  L1044     CMP.WD7,D3;  vuelve a comparar los contadores
0000580a  65b4                          BCS     L1043;  si faltan letras, loopea y vuelve al checksum
0000580c  200b                          MOVE.L  A3,D0
0000580e  b840                          CMP.W   D0,D4; en D4 está el serial correcto y en D0 mi fake
```

Pues así es. Os he detallado paso a paso qué hace el programa para que podáis hacer un keygen con vuestra herramienta favorita de programación. Yo voy de cabeza a hacer el mío en Delphi, así aprendo a programar en Delphi ;o).

Un abrazo.

X-Grimator
U.E. 20/07/02
“Pensad, pensad...siempre pensad”