Desempacando un armadillo sencillo

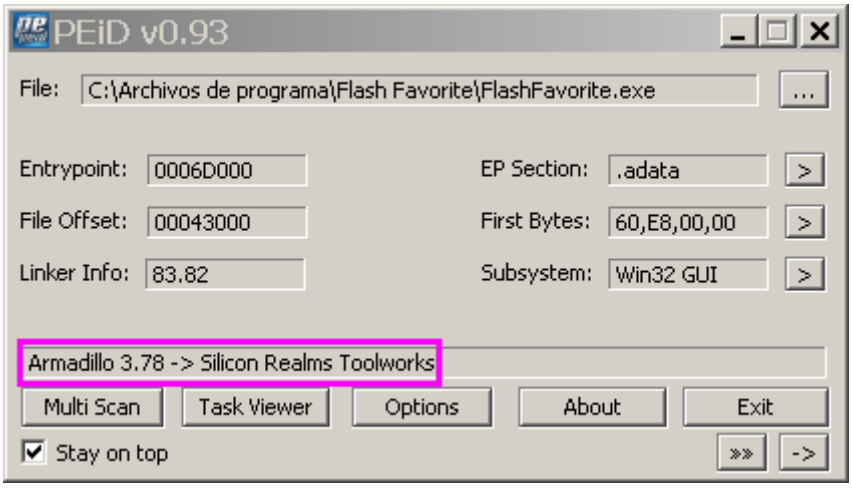
Programa:	Flash Favorite Versión 1.4.8
Download:	http://www.pipisoft.com/flashfavorite.exe
Protección:	Armadillo 3.78 (si no se demuestra otra cosa)
Descripción:	Grabar ficheros Flash de paginas web
Dificultad:	Pequeña
Herramientas:	PEID, OllyDBG, LordPE e Import Reconstructor
Objetivos:	Desempacado del programa
Cracker:	Orniaco (mas bien traductor)

Introducción:

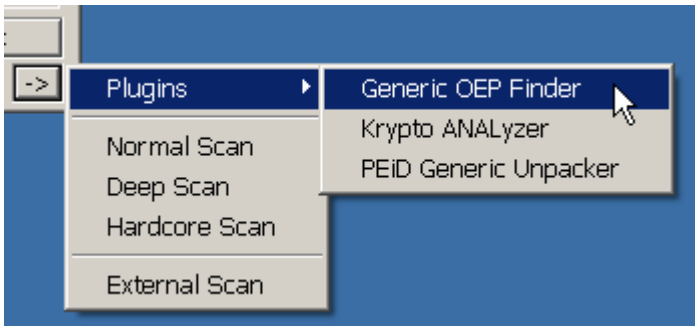
Voy a intentar ilustrar como utilizar un script para desempacar un armadillo sencillo.

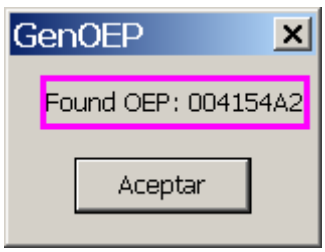
Al atake:

Con PEiD nos damos cuenta que nos enfrentamos al temible armadillo:

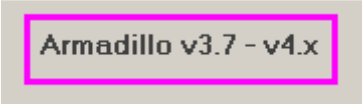


Aprovechamos para intentar conocer el EOP del programa:

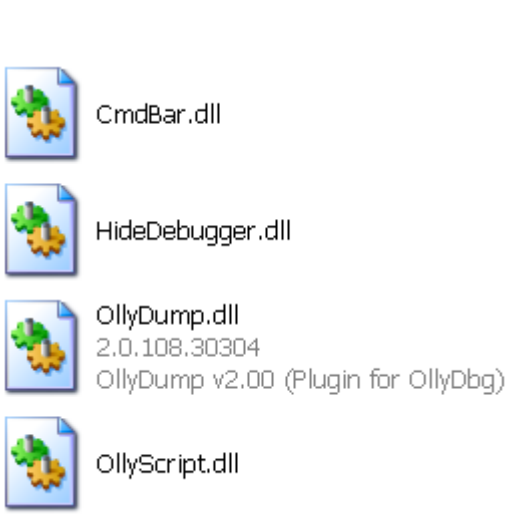


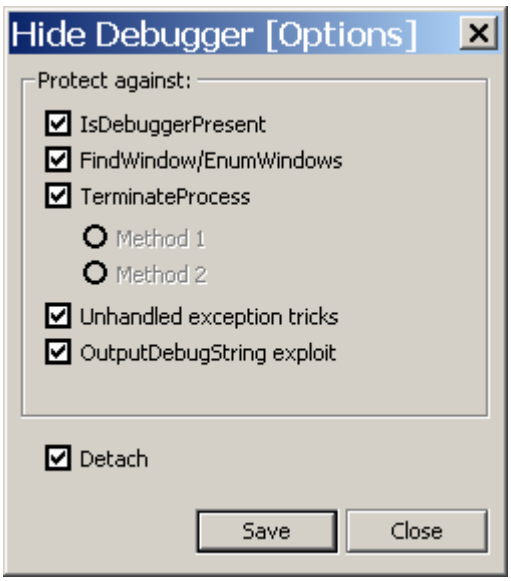


Para mayor seguridad lo comprobamos también con el RDG packer detector:

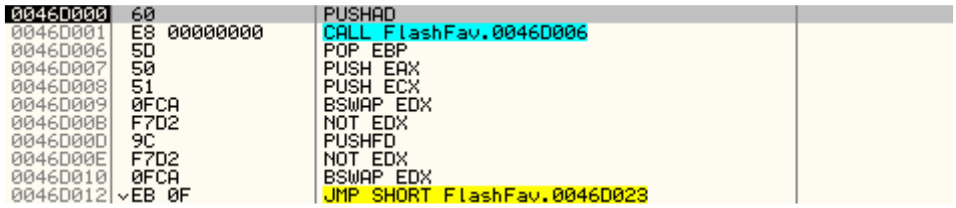


Para que no haya equívocos, vamos a utilizar Ollydbg con solo estos plugins:



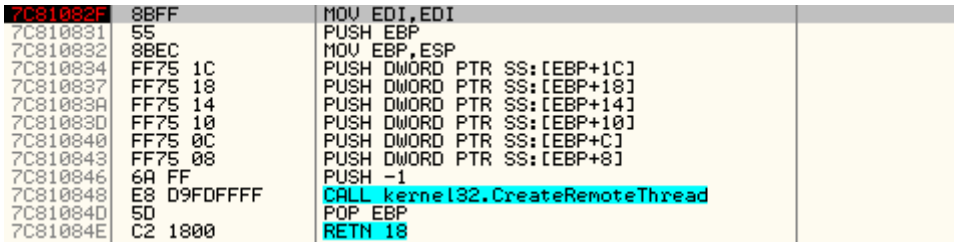


Cargamos Olly y nos aparece la siguiente pantalla:



Nuestro trabajo va a consistir en determinar tres direcciones del programa para luego hacer un pequeño script que nos permita desempacar el programa. La primera dirección es el EOP. Para ello ponemos un BP a la api CreateThread desde la línea de comandos:

Apreto [Shift] + [F9] para arrancar el programa y evitar alguna excepción odiosa (tengo configurado Olly para que salte automáticamente todas las excepciones) y se para en la línea:



Con ayuda de [F8] traceo hasta el RETN 18 llegando a:

```
00C0B7EA  5F              POP EDI                  0012FF04
00C0B7EB  5E              POP ESI
00C0B7EC  C9              LEAVE
00C0B7ED  C3              RETN
00C0B7EE  55              PUSH EBP
00C0B7EF  8BEC            MOV EBP,ESP
00C0B7F1  81EC 28010000   SUB ESP.128
```

Seguimos traceando hasta el RETN con [F8] para salir en:

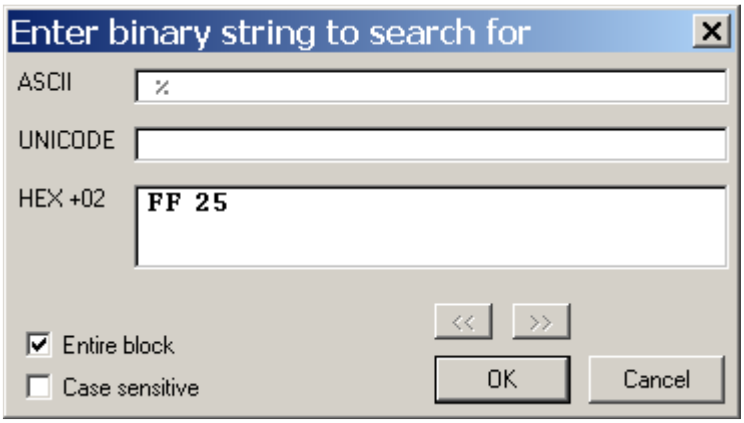
```
00C1DF60  59              POP ECX                  kernel32.7C8107FD
00C1DF61  BF 68D9C200     MOV EDI,0C2D968
00C1DF66  8BCF            MOV ECX,EDI
00C1DF68  E8 3EA4FDFF     CALL 00BF83AB
00C1DF6D  84C0            TEST AL,AL
00C1DF6F  75 09           JNZ SHORT 00C1DF7A
00C1DF71  6A 01           PUSH 1
00C1DF73  8BCF            MOV ECX,EDI
00C1DF75  E8 D9F2FDFF     CALL 00BFD253
00C1DF7A  B9 D0CCC200     MOV ECX,0C2CCD0
00C1DF7F  C705 70A0C200 5 MOV DWORD PTR DS:[C2A070],0C2AF5C
00C1DF89  E8 7F520000     CALL 00C2320D
00C1DF8E  6A 00           PUSH 0
00C1DF90  E8 78520000     CALL 00C2320D
00C1DF95  A1 B0DFC200     MOV EAX,DWORD PTR DS:[C2DFB0]
00C1DF9A  59              POP ECX
00C1DF9B  8B16            MOV EDX,DWORD PTR DS:[ESI]
00C1DF9D  8B88 84000000   MOV ECX,DWORD PTR DS:[EAX+84]
00C1DFA3  3348 10         XOR ECX,DWORD PTR DS:[EAX+10]
00C1DFA6  3348 08         XOR ECX,DWORD PTR DS:[EAX+8]
00C1DFA9  030D C8DFC200   ADD ECX,DWORD PTR DS:[C2DFC8]   FlashFav.00400000
00C1DFAF  85D2            TEST EDX,EDX
00C1DFB1  75 1D           JNZ SHORT 00C1DFD0
00C1DFB3  8B90 8C000000   MOV EDX,DWORD PTR DS:[EAX+8C]
00C1DFB9  FF76 18         PUSH DWORD PTR DS:[ESI+18]
00C1DFBC  3390 84000000   XOR EDX,DWORD PTR DS:[EAX+84]
00C1DFC2  FF76 14         PUSH DWORD PTR DS:[ESI+14]
00C1DFC5  3310            XOR EDX,DWORD PTR DS:[EAX]
00C1DFC7  FF76 10         PUSH DWORD PTR DS:[ESI+10]
00C1DFCA  2BCA            SUB ECX,EDX
00C1DFCC  FFD1            CALL ECX
00C1DFCE  EB 22           JMP SHORT 00C1DFF2
00C1DFD0  83FA 01         CMP EDX,1
00C1DFD3  75 1F           JNZ SHORT 00C1DFF4
00C1DFD5  FF76 04         PUSH DWORD PTR DS:[ESI+4]
00C1DFD8  8B90 8C000000   MOV EDX,DWORD PTR DS:[EAX+8C]
00C1DFDE  3390 84000000   XOR EDX,DWORD PTR DS:[EAX+84]
00C1DFE4  FF76 08         PUSH DWORD PTR DS:[ESI+8]
00C1DFE7  3310            XOR EDX,DWORD PTR DS:[EAX]
00C1DFE9  6A 00           PUSH 0
00C1DFEB  FF76 0C         PUSH DWORD PTR DS:[ESI+C]
00C1DFEE  2BCA            SUB ECX,EDX
00C1DFF0  FFD1            CALL ECX
00C1DFF2  8BD8            MOV EBX,EAX
00C1DFF4  5F              POP EDI
00C1DFF5  8BC3            MOV EAX,EBX
00C1DFF7  5E              POP ESI
00C1DFF8  5B              POP EBX
```

Observamos que existen dos CALL ECX, en el segundo ponemos un BP, apretamos [F9] y luego [F7] para introducirnos dentro de la llamada y llegar al EOP:

```
004154A2  55              PUSH EBP                 EOP
004154A3  8BEC            MOV EBP,ESP
004154A5  6A FF           PUSH -1
004154A7  68 38934100     PUSH FlashFav.00419338
004154AC  68 06564100     PUSH FlashFav.00415606   JMP to msvcrt._except_handler3
004154B1  64:A1 00000000  MOV EAX,DWORD PTR FS:[0]
004154B7  50              PUSH EAX
```

Primera dirección conseguida: EOP = 004154A2.

¿Dónde estará nuestra iat? ¿Habrán sido encriptadas algunas llamadas a apis? La buscamos como tradicionalmente:



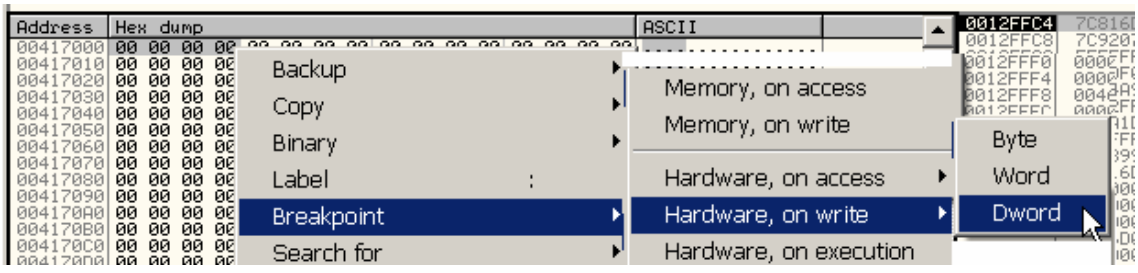

```
004155F7|  FF75 88          |PUSH DWORD PTR SS:[EBP-78]
004155FA|  FF15 74754100    |CALL DWORD PTR DS:[417574]
00415600|-FF25 84754100     |JMP DWORD PTR DS:[417584]
00415606|-FF25 7C754100     |JMP DWORD PTR DS:[41757C]
0041560C|-FF25 78754100     |JMP DWORD PTR DS:[417578]
00415612|-FF25 70754100     |JMP DWORD PTR DS:[417570]
00415618|-FF25 64754100     |JMP DWORD PTR DS:[417564]
0041561E|  68 00000300      |PUSH 30000
00415623|  68 00000100      |PUSH 10000
```

La iat parece estar comprendida entre 00417000 y 00417700:

```
Address  | Hex dump                                          | ASCII
00417000 | F0 6B DA 77 83 78 DA 77 1B 76 DA 77 CF 66 C0 00 | -k rwâx rw+v rwßf └.
00417010 | 2A 58 3A 77 CD 53 3B 77 5C 40 3A 77 8D 51 3B 77 | *X:w═S;w\@:w¡Q;w
00417020 | D0 55 3B 77 37 55 3B 77 48 66 C0 00 12 90 EF 77 | ╨U;w7U;wHf└.╞é'w
00417030 | F1 5F EF 77 A6 6C EF 77 A7 5B EF 77 C0 6D EF 77 | ±_'w╛l'w┬['w└m'w
                                  ...
004176C0 | 1D C7 D1 77 FD BE D1 77 E2 C2 D1 77 78 8E D1 77 | ¶╟╤w²¥╤wΓ╤wx╞╤w
004176D0 | 71 BE D1 77 85 CB D1 77 58 D6 D1 77 E2 16 D2 77 | q¥╤wà╦╤wX╓╤w┌▬╥w
004176E0 | FD 66 C0 00 F7 A8 B1 76 2F 67 C0 00 8A 61 52 77 | ²f└.≈¿▒v/g└.èaRw
004176F0 | 5E 05 4D 77 24 2A 4D 77 2A 67 C0 00 01 9E 28 77 | ^♣Mw$*Mw*g└.Ø×(w
00417700 | 48 66 C0 00 00 00 00 00 0F 00 00 00 00 00 00 00 | Hf└.....*.......
00417710 | 00 00 00 00 00 00 00 00 0C 00 00 00 E9 11 40 00 | ............ú4@.
00417720 | 01 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ☺☻.....↕.......
```

y tiene algunas entradas encriptadas (en rojo).
Reiniciamos el programa con [Control] + [F2], seleccionamos el dump y vamos a la dirección 00417000:

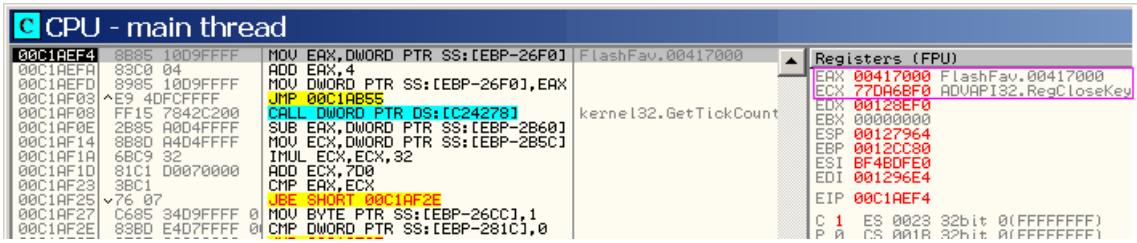
Para poner un Hardware Breakpoint on write dword en la dirección 00417000:



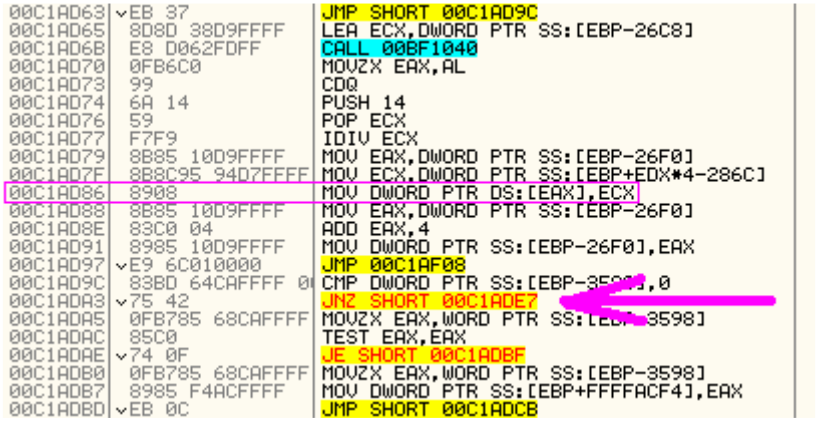
Ejecutamos el programa como inicialmente con [Shift] + [F9] y nos para en:

```
77C16FA3  F3:A5          REP MOVS DWORD PTR ES:[EDI],DWORD PTR DS
77C16FA5  FF2495 B870C177 JMP DWORD PTR DS:[EDX*4+77C170B8]
77C16FAC  8BC7           MOV EAX,EDI
77C16FAE  BA 03000000    MOV EDX,3
77C16FB3  83E9 04        SUB ECX,4
77C16FB6  72 0C          JB SHORT msvcrt.77C16FC4
77C16FB8  83E0 03        AND EAX,3
```

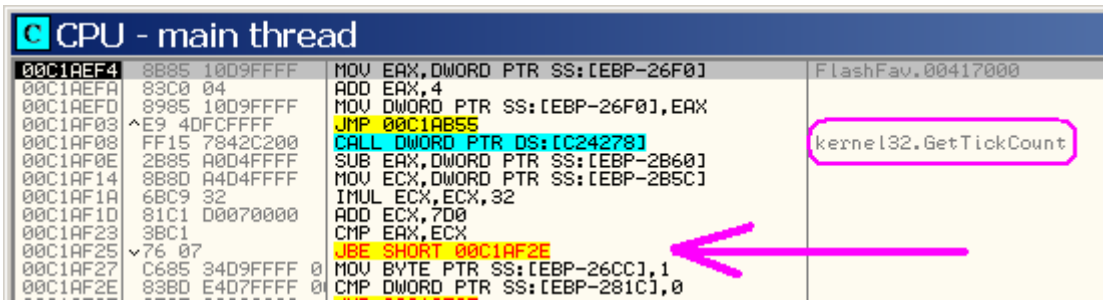
Seguimos con [F9] hasta:



Está escribiendo la iat. Si localizo el salto mágico que evita que encripte algunas rutinas podremos reconstruir el programa. Si analizamos la rutina podemos llegar a la conclusión que si NOPEAMOS el salto 00C1ADA3 el packer no encriptará llamadas de la iat.



El packer también comprueba que discurra el tiempo con normalidad cuando se desempaqueta el programa (pone a 1 un flag [EBP-26CC] si transcurre demasiado tiempo porque estamos depurando). Hasta aquí ninguna idea de cosecha propia, todo gracias a MaDMAn_H3rCuL3s del ARTEAM.



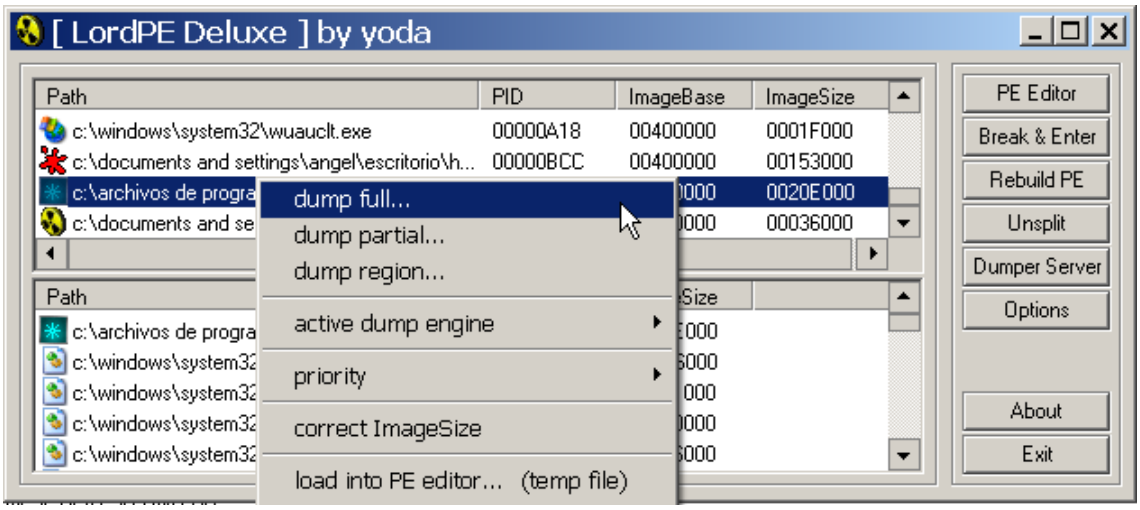
Segunda dirección conseguida: saltoanopear = 00C1ADA3.
Tercera dirección conseguida: saltoaforzar = 00 C1AF25.

Con un pequeño script vamos a superar estas dificultades; consiste en poner tres hardware breakpoints de ejecución en el salto a topear, el salto a forzar y en el EOP.

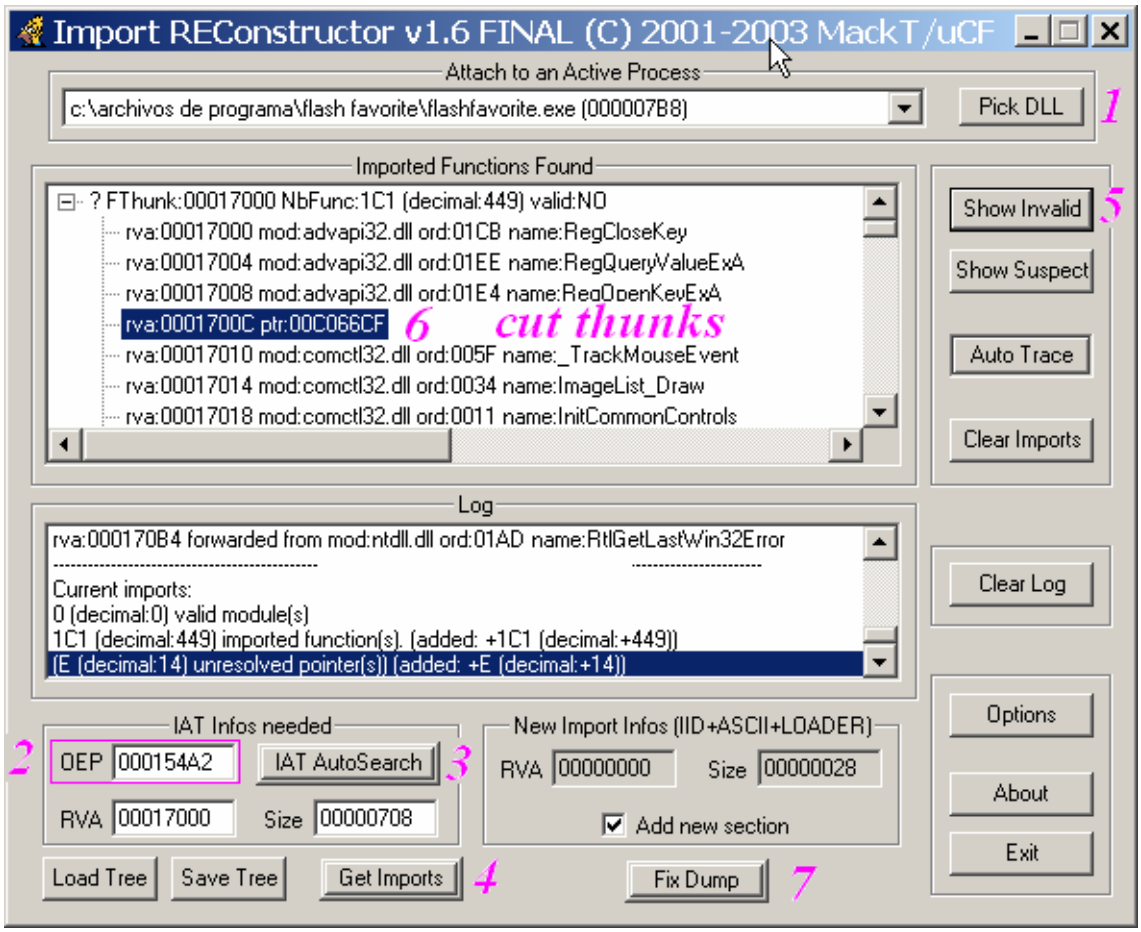
En cada parada por breakpoint comprobamos el eip y diferenciamos si el salto a topear (en el que en vez de nopear hacemos que la instrucción siguiente a ejecutar sea la siguiente al salto sumando 2 a eip), o es el salto a forzar (en el que en vez de forzar escribiendo un JMP hacemos que la instrucción siguiente a ejecutar sea la adecuada al salto sumando 9 a eip), o el eop donde quitamos todos los breakpoints y salimos del script anuciando que estamos en el EOP

```
var saltoanopear
var saltoaforzar
var EOP

mov saltoanopear, 00C1ADA3
mov saltoaforzar, 00C1AF25
mov EOP,          004154A2

  bphws saltoanopear , "x"
  bphws saltoaforzar , "x"
  bphws EOP , "x"
  esto

seguir:
  eob hayparada
  run
  ret

hayparada:
 log eip
  cmp eip, saltoanopear
  je salto1
  cmp eip, saltoaforzar
  je salto2
  cmp eip, EOP
  je salto3
  msg "Error parada inesperada"
  ret

salto1:
  add eip,2
  jmp seguir

salto2:
  add eip,9
  jmp seguir

salto3:
  bphwc saltoanopear
  bphwc saltoaforzar
  bphwc EOP
  msg "Estamos en el EOP"
  ret
```

Una vez en el EOP volcamos con LordPE (con OllyDump se niega posteriormente a arreglar el volcado):



Después del volcado utilizamos el Import Reconstructor. (1) Seleccionamos el proceso de los existentes en memoria (2) ponemos el valor de la EOP descontando 400000 (3) apretamos el botón de búsqueda automática del comienzo y tamaño de la iat (4) hacemos la importación (5) comprobamos si hay alguna entrada inválida (6) seleccionamos una y con el botón derecho seleccionamos cut thunks en el menú emergente y (7) arreglamos el volcado



¡Ejecutamos el programa y funciona! Está desempacado ahora solo hace falta crackearlo.

Agradecimientos

A Ricardo Narvaja, autentico azote de los armadillo, a mis amigos Morales (un monstruo), +NCR y a Daniel y Ans-Gari por sus aportaciones en Radasm y como no a todos los CracksLatinoS.