

Software: Alien Shooter.
Empresa: SIGMA Team.
Categoría: Juego.

Objetivo: Desempacar.
Tools: Olly v. 1.10, ImpRec.
ODBScript.

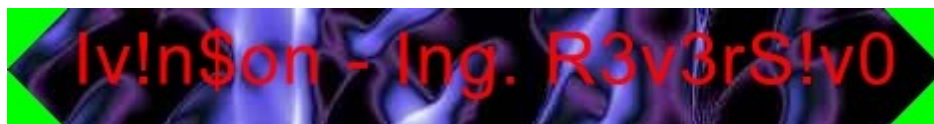
Cracker: Iv!n\$on.
Fecha: 25/02/2012.
Tuto N°: 7.

CracksLatinos

Supertal.com Beach Volley



Download Soft: <http://www.mediafire.com/?jla7nu66jrkmnvn>



Introducción

No hace mucho, lo máximo que conocía de ingeniería inversa era abrir un soft en Resource Hacker y traducirlo. ☺ Ahora, llevo 7 tutoriales con este para el corto tiempo de estudio que le he dedicado al Cracking. Hoy trabajaremos con Asprotect. Si supiera Alexei Solodovnikov que hasta un novato como yo es capaz de desempacarle su protector “Ruso”.

Mi amigo StrongCoder me pasó un juego demo empacado para ver si podía desempacarlo. Para ser sincero, intentaba y nada. Pero aquí, en mi país, decimos: “A cada cochino le llega su sábado para hacerlo chicharrón”.

Así que, veamos como atacaremos. Es hora de encender Olly, poner algo de música, ponerse cómodo y Ni Un Paso Atrás.



Fire in the hole

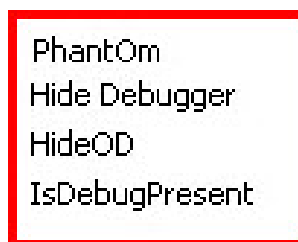
Instalemos el juego (demo). Parece Resident Evil. ☺



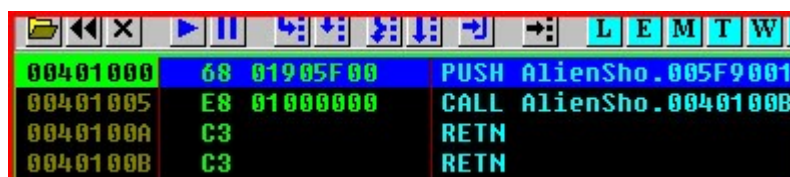
Y luego, lo analizamos como siempre.

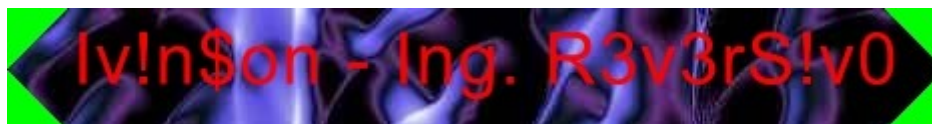


Éstos son los Plugins que tengo por si acaso:



Veamos el EP en Olly.



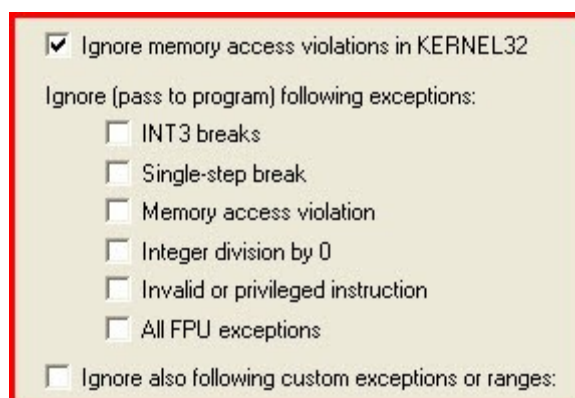


Si le doy Run en un OllySND con las excepciones destildadas, no para y nos muestra la primera ventanita de configuración.



Por lo que, ya sabemos que si un soft no para por excepciones en OllySND, tenemos otra posibilidad de probar con un Olly original.

Destildamos las excepciones así:



Damos Run y vemos que si para en el Olly original. ☺



Si damos Shift+F9 26 veces, el programa arranca. Por lo que, tendríamos que dar Shift+F9 25 veces para llegar a la última excepción, pasarla con Shift+F7, poner un Memory BreakPoint en la primera sección, luego Shift+F9, y ya llegamos al falso OEP.



¿Es mucho proceso dar Shift+F9 25 veces? Por supuesto.

Usen el siguiente Script que los dejará en la última excepción.

```
//Autor: lv!n$on
//Exception Script para cualquier PE.
run
mov buffer, 1 //Iniciamos el bucle con 1.
sigue:
esto //Shift+F9.
inc buffer
cmp buffer, 1A ;Aquí pones el número de excepciones.
jne sigue
ret
```

Probemos el Script.

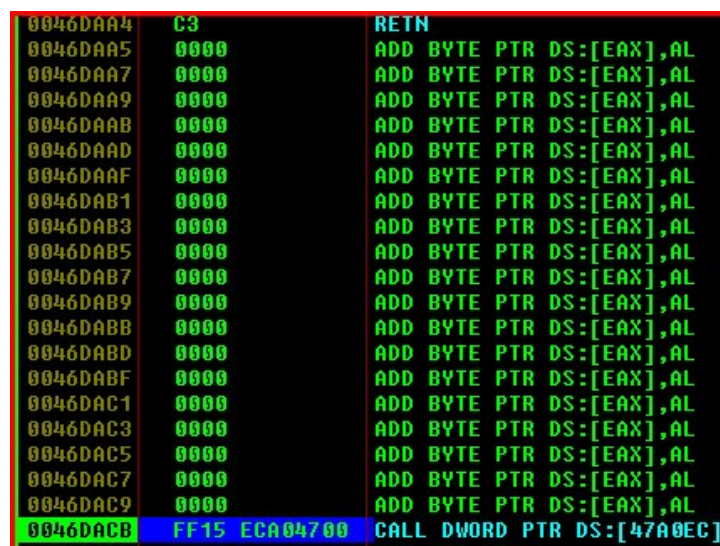


Funciona perfectamente. Paró aquí:



Ahora, solo doy Shift+F7, pongo un BPM en la primera sección, Shift+F9 y listo. ¿Menos trabajo? Se pone mejor aún. Si colocamos un HE en el falso OEP, ya no necesitaremos el Script. ☺

Si miramos por encima del falso OEP, tenemos:





¿38 bytes robados? WTF. Si Ricardo tardo 1 hora en deducir un Push 60 y un Push -1. ¿Cuánto le tomaría a un novato cómo yo encontrar estos supuestos 38 bytes? ☹

Dejémoslo para el final. Y concentrémonos en arreglar la IAT.

En la imagen inferior (Falso OEP) vemos un CALL redirigido.

0046DAC7	0000	ADD BYTE PTR DS:[EAX],AL
0046DAC9	0000	ADD BYTE PTR DS:[EAX],AL
0046DACB	FF15	CALL DWORD PTR DS:[47A0EC]

Démosle Follow in Dump/Memory Address y busquemos los datos de la IAT para el ImpRec.

Inicio de IAT:
47A000.

Fin de IAT:
47A25C.

Inicio de IAT – 400000 = RVA.

47A000 – 400000 = **7A000**.

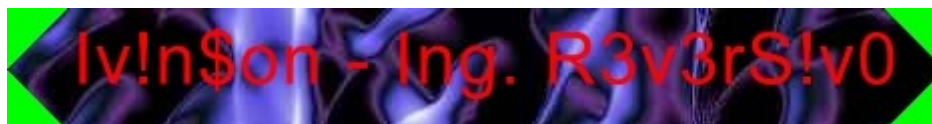
Fin - Inicio = Size o Largo.

47A25C - 47A000 = **25C**.

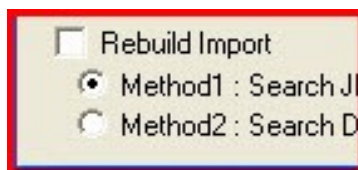
46DACB (Falso OEP) – 400000 = **6DACB**.

Datos concretos para el ImpRec:

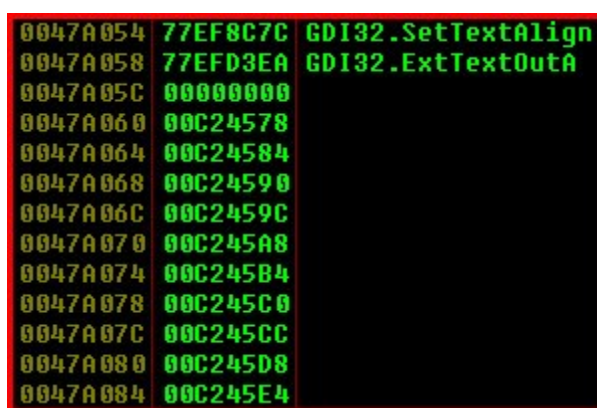
OEP: **6DACB**
RVA: **7A000**
SIZE: **25C**



Probemos el OllyDump a ver si funciona. Destildamos Rebuild:



Arreglemos la IAT. Cuando estábamos buscando los datos para el ImpRec, vimos que el packer destrozó muchas API's. Vean la imagen en modo Long Address.



Pongamos un HBP en **47A060** para buscar la CALL mágica.



Como ya no necesitamos el Script, tildemos todas las excepciones y pongamos un HE en el falso OEP para tenerlo seguro.

HE **46DACB**.

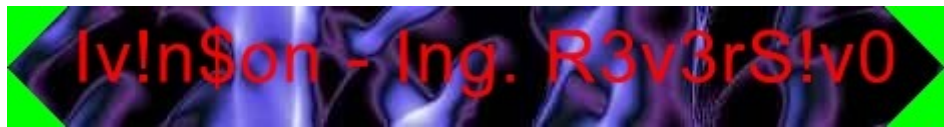
Reiniciemos. F9 como 6 veces hasta caer en un JMP así:



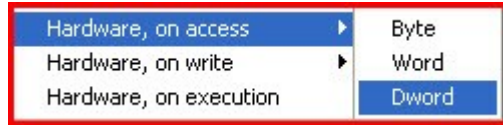
Si miramos un poco hacia arriba, encontraremos 2 CALL's.

La segunda CALL es la mágica.





Si por ejemplo, colocamos un HBP en 47A000 (Inicio de IAT).



Caemos en un sitio similar al de la CALL mágica.

00BE26AD	50	PUSH EAX
00BE26AE	E8 D9FBFFFF	CALL 00BE228C
00BE26B3	8B17	MOV EDX,DWORD PTR DS:[EDI]
00BE26B5	8902	MOV DWORD PTR DS:[EDX],EAX
00BE26B7	EB 09	JMP SHORT 00BE26C2

Pero esa no es. ☺ Solo tiene una CALL. Por lo que, tenemos que buscar otra parte de la IAT, preferiblemente un valor malo.

Entonces, ya tenemos la dirección de la CALL mágica:

0BE2639 CALL BE24C8

Para poder ponerle un HE a la CALL mágica, dejamos solamente el HE del OEP falso. Reiniciamos, F9, para en el OEP y buscamos **0BE2639**. Pongámosle un HE. En total, solo tenemos 2 HE. Uno en el OEP y el otro en la CALL mágica.

Ahora, sí. Reiniciamos y damos F9. Para en la CALL mágica. Nopeémosla.

00BE2633	50	PUSH EAX	
00BE2634	E8 53	CALL 00BE228C	
00BE2639	90	NOP	
00BE263A	90	NOP	
00BE263B	90	NOP	
00BE263C	90	NOP	
00BE263D	90	NOP	
00BE263E	8B17	MOV EDX,DWORD PTR DS:[EDI]	
00BE2640	8902	MOV DWORD PTR DS:[EDX],EAX	
00BE2642	EB 7E	JMP SHORT 00BE26C2	CAIMOS AQUÍ

Tracemos con F8 hasta:

00BE287B	74 DD	JE SHORT 00BE285A	←
00BE287D	80F8	CMP AL,6	
00BE2880	75 06	JNZ SHORT 00BE2888	
00BE2882	8345	ADD DWORD PTR SS:[EBP-8],4	
00BE2886	EB EF	JMP SHORT 00BE2877	



Estando en ese JE, démosle Enter y caemos en:

00BE285A	AD	LDS DWORD PTR DS:[ESI]
00BE285B	09C0	OR EAX,EAX
00BE285D	74 4A	JE SHORT 00BE28A9
00BE285E	00C7	MOV EAX,EAX

Le damos clic a ese JE, luego Enter y caemos en:

00BE28A9	61	POPAD
00BE28AA	E8 3A	CALL 00BE28E9

Le ponemos un HE a ese POPAD. Ya al llegar a este punto, la IAT estará arreglada. Quitamos el HE de **0BE2639** (CALL mágica)

Damos F9 y cuando para en ese POPAD, damos CTRL+G y vamos a **0BE2639** (CALL mágica) seleccionamos los bytes y Undo Selection para restaurar la CALL. Si no restauramos la CALL y damos Run, mira lo que sale: ☺



Igual saldrá si en el POPAD colocamos un BP común porque lo detecta, más no detecta el HE. Ridiculeces del Packer.

Si siguen mi concejo llegaremos al OEP.

0046DACB	FF15 ECA04700	CALL DWORD PTR DS:[47A0EC]
----------	---------------	----------------------------

Otra cosa a tener en cuenta es no dejar que Olly analice el programa porque no se verá bien el código:

0046DACB	FF	DB FF	Falso OEP
0046DACC	15	DB 15	
0046DACD	EC	DB EC	
0046DADE	A0	DB A0	

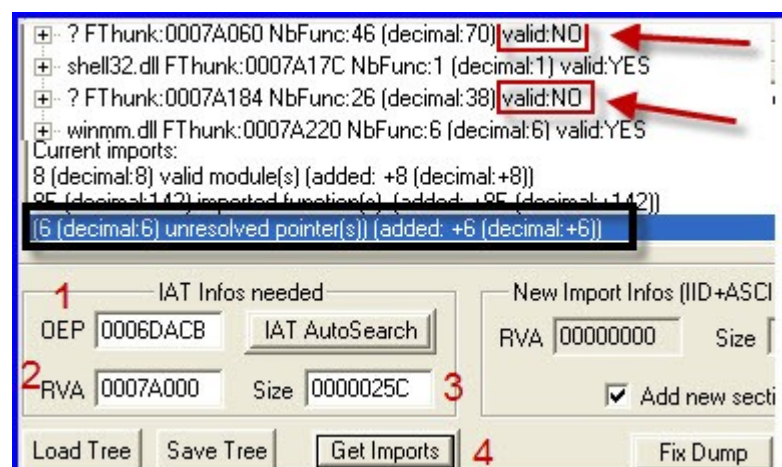
Eso lo resolveremos pulsando la barra espaciadora al cargarlo o:

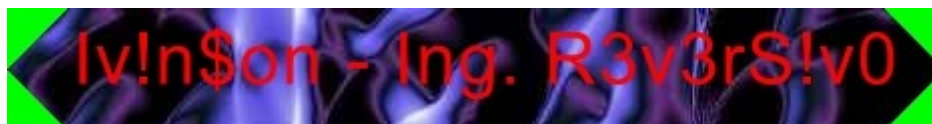


Repasemos:

- 1) HE en el OEP.
- 2) HE en la CALL mágica.
- 3) HE en el POPAD.
- 4) F9 y nopeamos la CALL mágica.
- 5) Quitamos el HE de la CALL mágica.
- 6) F9 y al parar en el POPAD, restauramos la CALL.
- 7) F9 y llegamos al OEP.

Ahora sí abramos ImpRec:

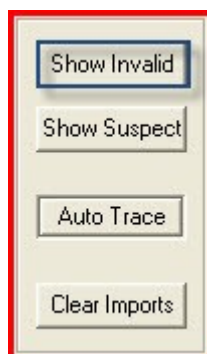




Como

podemos apreciar en la imagen previa, aún quedaron 6 valores sin arreglar. Es por eso que Syxe la llama CALL semi-mágica, pero 6 es un valor decente y mucho más práctico para arreglarlo a mano. Así que, no seamos mal agradecidos. ☺

Veamos cuales son esas API's que faltan. Presionemos el botón Show Invalid:



Vamos a

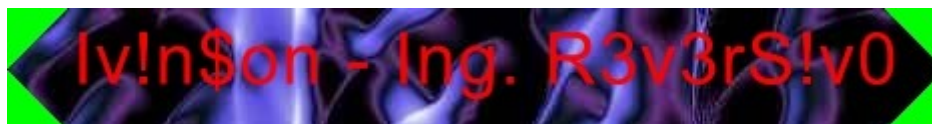
proceder de la siguiente forma: Haremos clic derecho en cada valor malo en el ImpRec para desensamblar y analizar cuál valor le corresponde a cada una.



Valor N° 1: GetProcAddress.

```
00BE0EF0  push ebp
00BE0EF1  mov  ebp,esp
00BE0EF3  mov  edx,[ebp+C]
00BE0EF6  mov  eax,[ebp+8]
00BE0EF9  mov  ecx,[BE543C]    // DWORD value: 00BE5314
00BE0EFF  mov  ecx,[ecx]
00BE0F01  cmp  ecx,eax
00BE0F03  jnz  short 00BE0F0E
00BE0F05  mov  eax,[edx*4+BE5350] // DWORD value: 00000000
00BE0F0C  jmp  short 00BE0F15
00BE0F0E  push edx
00BE0F0F  push eax
00BE0F10  call 00BD5160    // = kernel32.dll/0199/GetProcAddress
```

Pongámoslo como GetProcAddress.



Para corregirla, hacemos doble click en el valor malo y la escribimos así:



Y así, haremos con cada una en ImpRec.

Valor N° 2: GetVersion

```
00BE1388  push 0
00BE138A  call 00BD5158    // = kernel32.dll/0177/GetModuleHandleA
00BE138F  push dword ptr [BE6CE8]    // DWORD value: 0A280105
00BE1395  pop eax
00BE1396  retn
```

En los tutoriales de CracksLatinos nos dicen que ésta es GetVersion.

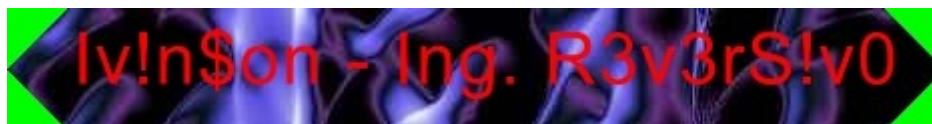
En azul, vemos que PUSHes **0A280105** que es el valor de retorno de GetVersion. Ahora, vean porque no es GetModuleHandleA. Abran en otro Olly el Crackme 1 de CrueHead (conejiillo de Indias ☺). Su primera API es GetModuleHandleA y vean que si trazamos la CALL con F8, EAX retorna el valor de la ImageBase (**400000**).

Assembly	Registers (FPU)
PUSH 0	EAX 00400000 ASCII "MZP"
CALL <JMP.&KERNEL32.GetModuleHandleA>	ECX 0012FFB0
MOV DWORD PTR DS:[4020CA],EAX	EDX 7C91E4F4 ntdll.KiFast
PUSH 0	EBX 7EED4000
PUSH CRACKME.004020F4	

Valor N° 3: GetCommandLineA.

```
00BE13D0  push 0
00BE13D2  call 00BD5158    // = kernel32.dll/0177/GetModuleHandleA
00BE13D7  push dword ptr [BE6CE8]    // DWORD value: 0A280105
00BE13DD  pop eax
00BE13DE  mov eax,[BE6CF8]    // DWORD value: 001523C0
00BE13E4  retn
00BE13E5  retn
```

Al principio, tuve muchos problemas con esta API. Si lo notaron. Las primeras 4 líneas son idénticas a la API GetVersion, pero la diferencia es el MOV EAX,[xxxxxx] //DWORD value: 001523C0.



Así que, ya sabemos la diferencia. ☺ Otra forma de reconocerla, es trazando en Olly, ya que debe retornar el Path del programa.

Valor N° 4: GetModuleHandleA.

```
00BE1360  push ebp
00BE1361  mov  ebp,esp
00BE1363  mov  eax,[ebp+8]
00BE1366  test  eax,eax
00BE1368  jnz  short 00BE137D
00BE136A  cmp  dword ptr [BE6978],400000
00BE1374  jnz  short 00BE137D
00BE1376  mov  eax,[BE6978] // DWORD value: 00400000
00BE137B  jmp  short 00BE1383
00BE137D  push  eax
00BE137E  call 00BD5158 // = kernel32.dll/0177/GetModuleHandleA
00BE1383  pop  ebp
```

El valor 4 si tiene toda la pinta de ser GetModuleHandleA.

Podemos notar como mueve a EAX el valor 400000.

Valor N° 5: GetCurrentProcess.

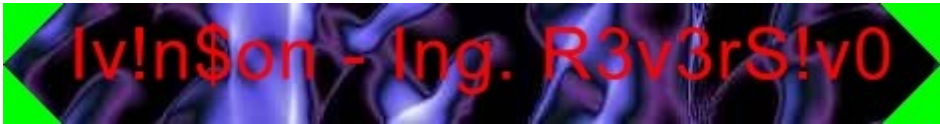
No recuerdo en cual tuto supe que era ese. Aunque, no tengo mucha seguridad. ☺ Retorna -1.

```
00BE13B8  mov  eax,[BE6CEC] // DWORD value: FFFFFFFF
00BE13BD  retn
```

Valor 6: DialogBoxParamA.

```
00BE1414  push  ebp
00BE1415  mov  ebp,esp
00BE1417  push  ebx
00BE1418  mov  ebx,[ebp+8]
00BE141B  mov  eax,[ebp+18]
00BE141E  push  eax
00BE141F  mov  eax,[ebp+14]
00BE1422  push  eax
00BE1423  mov  eax,[ebp+10]
00BE1426  push  eax
00BE1427  push  5
00BE1429  mov  eax,[ebp+C]
00BE142C  push  eax
00BE142D  push  ebx
00BE142E  call 00BD5108 // = kernel32.dll/00E0/FindResourceA
00BE1433  push  eax
00BE1434  push  ebx
00BE1435  call 00BD5188 // = kernel32.dll/024A/LoadResource
00BE143A  push  eax
00BE143B  call 00BD5190 // = kernel32.dll/0258/LockResource
```

La más grande



Vemos que llama a 3 API's relacionadas con recursos. Según lo que he leído Debería ser FreeResource o LockResource, pero que va. Me daba error. Resulta que Guillermo o Syxe, no recuerdo, dijeron en sus tutos que era DialogBoxParamA y no se equivocaron.

Gracias por compartir.

API's reparadas:

- | | |
|----------------------|---------|
| 1) GetProcAddress | -7A0E0- |
| 2) GetVersion | -7A0EC- |
| 3) GetCommandLineA | -7A0F0- |
| 4) GetModuleHandleA | -7A0F8- |
| 5) GetCurrentProcess | -7A104- |
| 6) DialogBoxParamA | -7A1F4- |

Stolen Bytes

Ya tenemos el dumpeado con la tabla arreglada. Seleccionen todos los Stolen Bytes arriba del OEP de tipo:

XXXXXXXX 0000 ADD BYTE PTRS DS:[EAX],AL

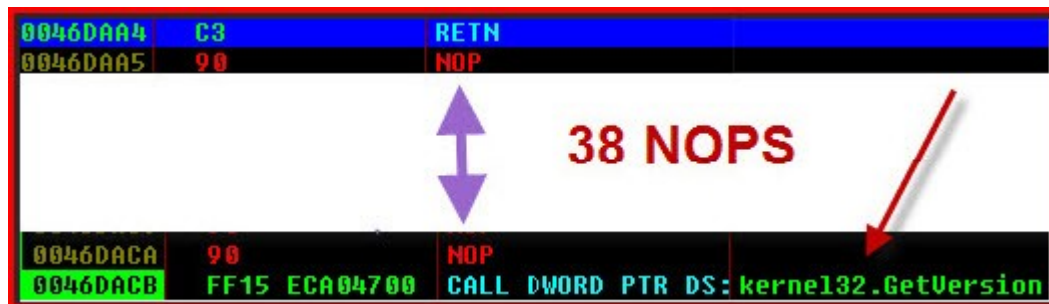
0046DAC5	0000	ADD BYTE PTR DS:[EAX],AL	
0046DAC7	0000	ADD BYTE PTR DS:[EAX],AL	
0046DAC9	0000	ADD BYTE PTR DS:[EAX],AL	
0046DAD3	FF15 ECA04700	CALL DWORD PTR DS:[<kernel32.GetVersion>	kernel32.GetVersion

Clic derecho:

0046DAA3	5B	POP EBX	
0046DAA4	C3	RETN	
0046DAA5	0000	ADD BYT	
0046DAA7	0000	ADD BYT	
0046DAA9	0000	ADD BYT	
0046DAAB	0000	ADD BYT	
0046DAAD	0000	ADD BYT	
0046DAAF	0000	ADD BYT	
0046DAB1	0000	ADD BYT	



Queda con mejor aspecto.



¿Qué tal? Y también podemos apreciar la primera API llamada por C++ 6.0. Seleccionen todos y damos Copy to Executable/Selection/Save File.

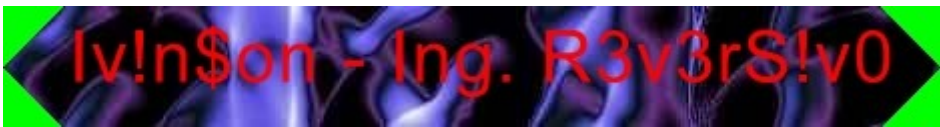
Ahora, llegó el momento de la verdad. ¿Cómo resolveremos los Stolen Bytes? Lo primero que se me ocurrió fue injertar el Stack como lo hizo Ricnar en su tuto.

260-ASPROTECT-STOLEN BYTES CON MUCHA FIACA.

Y vaya sorpresa que me llevé. Después de escribir como 10 páginas explicando el proceso, tuve que borrarlas todas y reescribir el tuto. Ya verán por qué. El siguiente es el injerto-plantilla que usó Ricnar.

```
PUSHAD                ;Guardamos los registros
MOV ESI,XXXXXXXXX ;Origen. Stack del empackado
MOV EDI,XXXXXXXXX ;Destino. Stack del dumpeado.
MOV ECX,0C00          ;3000 / 4 = 0C00. Contador.
REP MOVSD DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]
POPAD                 ;Restauramos los registros.
MOV ESP,XXXXXXXXX ; ESB Y EBP son suficientes.
MOV EBP,XXXXXXXXX ;
NOP
JMP XXXXXXXXX ;salto al falso OEP.
```

Después de hacer todo el procedimiento y explicarlo en este tuto. Cómo agregué las secciones, etc. Y ver como funcionaba este juego, me di cuenta que en realidad, no había colocado el valor correcto del Stack origen a ser copiado ni el valor original de los registros.



En fin, el injerto estaba malo y aún así, funcionaba perfectamente con solo colocar ESP y EBP como los puso Ricardo.

```
MOV ESP,12FF40
MOV EBP,12FFC0
```

Pero, resulta que esos valores son del soft original que usó él en su tuto que no tiene nada que ver con el mío.

Observen:

Original ↔ Unpacked	
Registers (FP)	Registers (FP)
EAX 0012FFE0	EAX 00000000
ECX 00C2414A	ECX 0012FFB0
EDX 4305585B	EDX 7C91E4F4
EBX 00000026	EBX 7FFD6000
ESP 0012FF2C	ESP 0012FFC4
EBP 0012FFA0	EBP 0012FFF0
ESI 0A28FF9F	ESI 7C920208
EDI FFFFFFF4	EDI 7C91E900

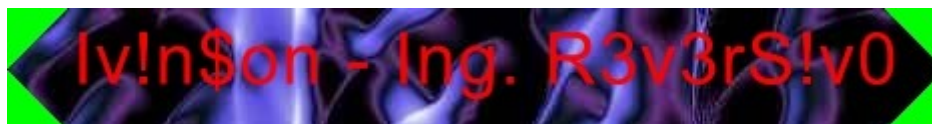
Quiere decir que los valores correctos para que mi dumpeado tenga los registros por lo menos de EBP y ESP iguales, las instrucciones serían:

```
MOV ESP,12FF2C
MOV EBP,12FFA0
```

Pero no parece relevante porque incluso funciona con:

```
MOV ESP,12FF00
MOV EBP,12FF00
```





Sin
de Stack, sin sección nueva y sin Stolen Bytes.

injerto

Solamente:

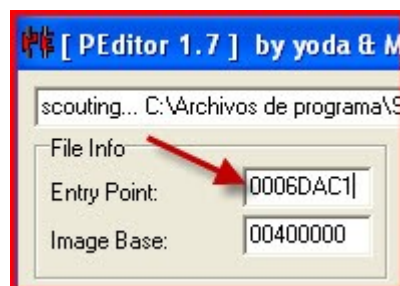
```
MOV ESP,12FF00  
MOV EBP,12FF00
```

¿Descubrí un OEP universal?

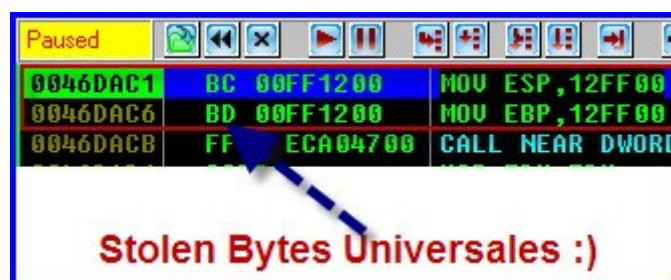
Bueno, terminemos el trabajo. Agreguemos el EntryPoint final con PEEditor.

0046DAC1	BC 00FF1200	MOV ESP,12FF00	← OEP
0046DAC6	BD 00FF1200	MOV EBP,12FF00	
0046DADB	FF15 ECA04700	CALL NEAR DWORD PT	

46DAC1 – 400000 == 6DAC1

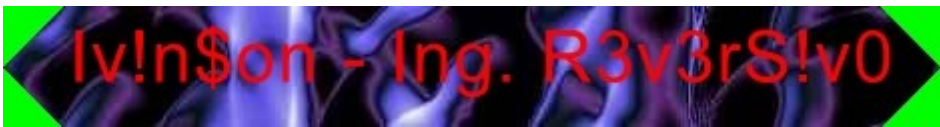


Lo vemos en Olly.



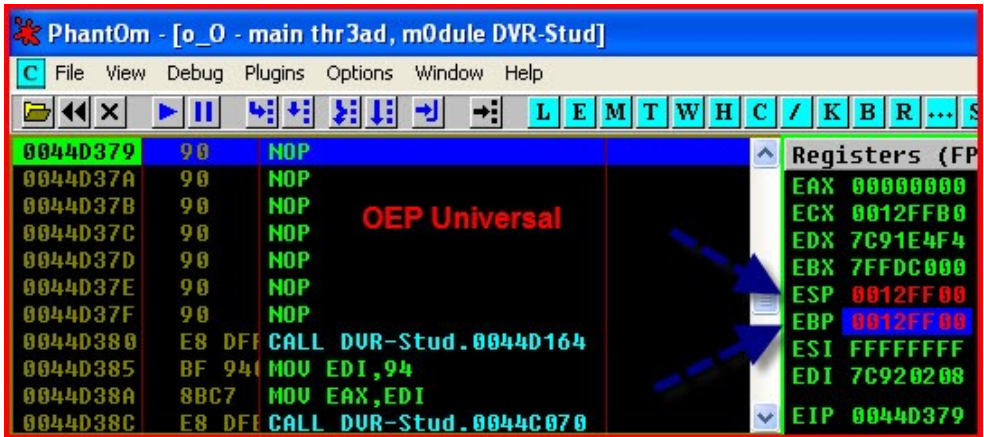
Sí, ya se que estarás pensando que a lo mejor ni siquiera necesite los MOV's para que el programa arranque, pero mira: ☺



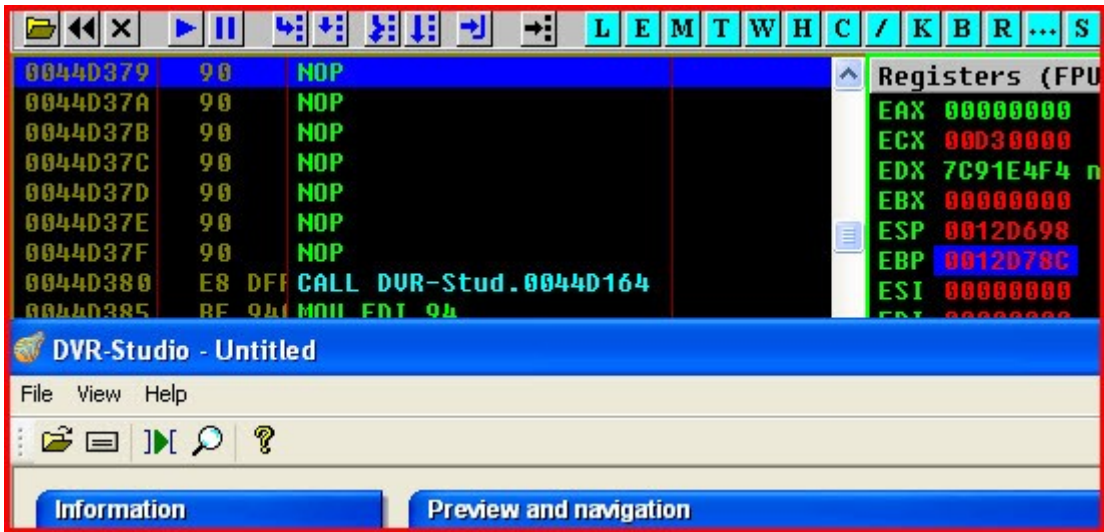


Insisto. ¿Será suerte y esto funciona solo con este soft?

Probemos con DVR-Studio del tuto de Ric. Solo desempaquémoslo, arreglemos la IAT y modifiquemos EBP y ESP a 12FF00. Sin injerto de Stack, sin PUSH 60 ni PUSH -1. (No olviden de quitar los antidebuggers [int 41] e [int 68])



Ahora, demos F9.



¿Convencido?

Iv!n\$on - Ing. R3v3rS!v0

Por fin.

Programa desempacado.



Iv!n\$on - Ing. R3v3rS!v0

Espero
hayan disfrutado este tuto tanto como yo. Bye.

que

lpadilla63@gmail.com

Nos vemos en el tuto 8. Dios mediante.

Mi novia Marjorie. (Broma)

