

## Guía para un análisis estructurado y organizado de una Muestra. (PE construida de MASM)

En el presente informe se va a realizar el análisis de una muestra binaria. Un PE construida en MASM como ejercicio para la entrega del proyecto del Ceh. Si el lector está interesado en un repaso de la arquitectura x86 o del formato de los archivos Portable Ejecutable de windows, se recomienda una lectura rápida de la parte introductoria de ese documento (evitando mirar los códigos)

Parece importante para la parte de análisis, ya que se leerá mucho código en ensamblador puro, realizar un pequeño repaso de unas instrucciones básicas de ensamblador que veremos muy a menudo:

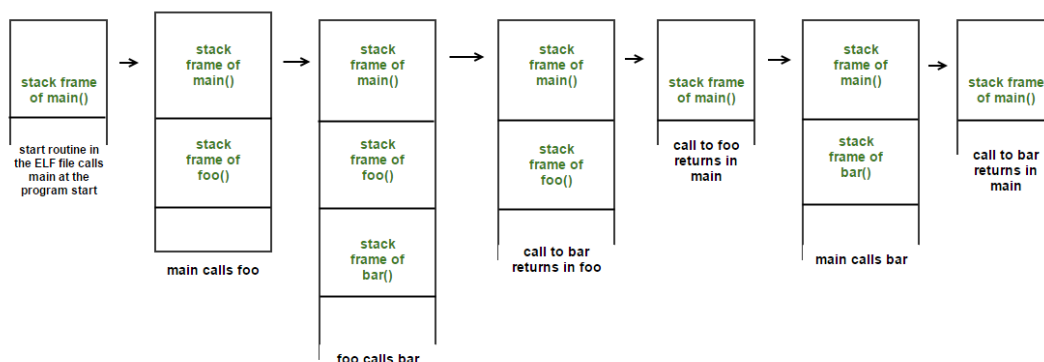
- mov: mover el valor entre direcciones de memoria o registros
- lea: cargar el valor al que apunta una dirección de memoria
- add, sub, mul... : operaciones aritméticas básicas
- flags: las operaciones anteriores pueden modificar los siguientes flags
  - zf: zero flag
  - cf: carry flag
- cmp: compara dos valores. Afecta a zf y cf
- jnz, jz, jge...: instrucciones que permiten saltar de una instrucción a otra. Se basan en los valores de los flags mencionados y vienen de una operación anterior.
- push: introduce un valor en la pila y actualiza el puntero ESP a la nueva dirección de memoria
- pop: recoge el valor de la dirección de memoria a la que apunta ESP y actualiza dicho registro a la dirección anterior.

Estas instrucciones no son mas que alias de los opcodes que la maquina utiliza, y que nos permiten recordar y entender de una manera mas fácil el flujo del programa.

Una de las herramientas que utilizaremos posteriormente, IDA, nos permite mostrar los opcodes de las instrucciones. A continuación los podemos ver en azul junto a la instrucción que corresponden:

|                |                |      |                  |
|----------------|----------------|------|------------------|
| .text:00401040 | 55             | push | ebp              |
| .text:00401041 | 8B EC          | mov  | ebp, esp         |
| .text:00401043 | 51             | push | ecx              |
| .text:00401044 | E8 B7 FF FF FF | call | sub_401000       |
| .text:00401049 | 89 45 FC       | mov  | [ebp+var_4], eax |
| .text:0040104C | 83 7D FC 00    | cmp  | [ebp+var_4], 0   |
| .text:00401050 | 75 04          | jnz  | short loc_401056 |
| .text:00401052 | 33 C0          | xor  | eax, eax         |
| .text:00401054 | EB 05          | jmp  | short loc_40105B |
| .text:00401056 |                |      |                  |

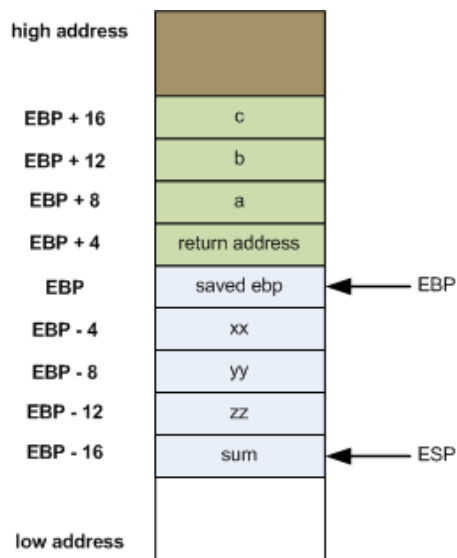
Durante el análisis también nos centraremos mucho en los valores en memoria. El comportamiento de la pila (stack) es un tanto particular, por lo que hagamos un pequeño resumen.



La memoria dentro de la pila funciona por frames: área de memoria utilizada para almacenar la información de una función determinada. Se reserva y se libera memoria, según se llama a nuevas funciones, o vuelve el flujo de las mismas, como se muestra en la imagen anterior.

Tres registros importantes a tener en cuenta para entender el funcionamiento de la pila:

- El puntero base **EBP** es un ancla al comienzo del marco actual de la pila, durante toda la ejecución de la función
- El registro **ESP** guarda una referencia al tope de la pila, última posición en la que podemos encontrar un valor válido. Dicho valor sería el que se recoja cuando se utilice la instrucción 'pop'
- Otro registro importante que no se debe confundir con los anteriores es **EIP**. Este guarda una referencia a la dirección donde está la siguiente instrucción a ejecutar. Por esto mismo se convierte en un objetivo de muchos exploit: si puedes controlar cuál es la dirección de la siguiente instrucción a ejecutar, puedes cambiar el flujo del programa para que empiece a ejecutar otro código malicioso (o no).



```
int foobar(int a, int b, int c)
{
    int xx = a + 2;
    int yy = b + 3;
    int zz = c + 4;
    int sum = xx + yy + zz;

    return xx * yy * zz + sum;
}
```

Entendamos esta imagen, y cuáles han sido los pasos para que la pila/stack tenga esos valores.

Cuando el programa ve que hay una llamada a otra función:

- mete en la pila los valores de los argumentos que va a pasar a la misma
  - nótese que se deben introducir de derecha a izquierda, ya que es una pila LIFO para que la función los pueda leer en el orden correcto. Por eso vemos que primero entró c, luego b y finalmente a
- mete en la pila la dirección de la instrucción a la que tiene que volver el flujo del programa cuando termine la función
- guarda la dirección de los límites del frame actual (ebp)
  - EBP pasa a apuntar al valor actual de ESP
  - ESP decrementa hasta dar cabida a todas las variables locales que se necesite definir

Como se ve en la imagen, los parámetros tienen un offset positivo con respecto al nuevo valor de EBP que se toma como referencia en la función ya que se meten en la pila antes de llamar a la misma. Mientras que las variables locales tienen un valor negativo al meterse después (teniendo en cuenta que la pila crece hacia direcciones de memoria menores)

Este baile y movimiento de valores se puede considerar como el prólogo que veremos en cualquier llamada a una función.

Cuando la función termina, los valores de EBP y ESP se actualizan para descartar las variables locales y moverse a la función que realizó la llamada. Dependiendo de la convención utilizada esto lo puede realizar la función que llama, o la llamada (cdecl, stdcall, fastcall..)

A continuación y como objetivo de este informe, realizaremos el análisis de la muestra en 4 fases:

1. Análisis estático básico
2. Análisis dinámico básico
3. Análisis estático avanzado – Desensamblado
4. Análisis dinámico avanzado – Debugar

## 1.- Análisis estático básico:

Consiste en analizar el binario para sacar información SIN ejecutarlo en ningún momento.

Utilizaremos las siguientes herramientas:

1. VirusTotal
2. ExeInfo
3. PEview
4. DependencyWalker
5. Strings
6. ResourceHacker

### 1.1- Subir el HASH a <http://www.VirusTotal.com>

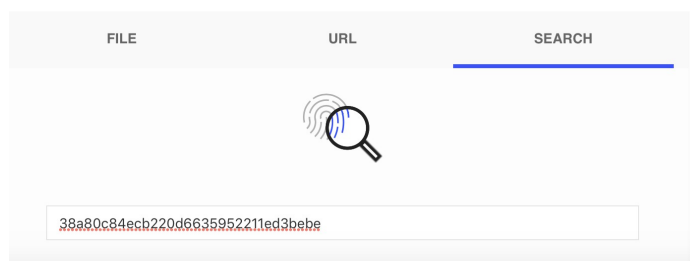
VirusTotal es una plataforma de origen español que, entre otras muchas funcionalidades, contiene una gran base de datos de referencias de malware además de integrarse con muchos antivirus para ofrecer sus servicios

```
vega@Esthers-MBP reversing $ md5 AllTogetherEduProject.exe  
MD5 (AllTogetherEduProject.exe) = 38a80c84ecb220d6635952211ed3bebe
```

En su sitio web se puede subir el archivo entero, aunque lo lógico es sacar el hash md5 del archivo y utilizarlo para comprobar si existe algún registro de la misma muestra de la siguiente manera:



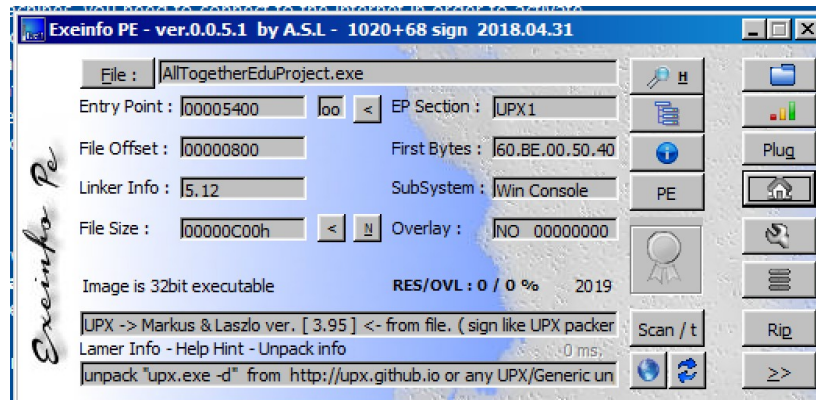
Analyze suspicious files and URLs to detect types of malware,  
automatically share them with the security community



Obviamente en este caso el resultado es “No matches found” como cabía esperar ya que es una muestra creada solo para el presente informe.

## 1.2- Exeinfo PE

Al abrir la muestra con esta herramienta se nos enseña mucha información. Lo principal que nos llama la atención está en la parte de abajo de la pantalla que nos indica que este ejecutable está empaquetado, comprimido.



De esta forma se nos oculta mucha información, es necesario desempaquetar la muestra antes de analizarla. La propia herramienta Exeinfo nos da una pista en la última línea en este caso de cómo realizar esta acción. Cuando se utilicen procesos de empaquetados más profesionales o a medida, no será tan fácil. (<https://github.com/upx/upx>)

```
C:\Users\IEUser\Desktop\finde\reversingUnpacked>upx.exe -d AllTogetherEduProject.exe

      Ultimate Packer for eXecutables
      Copyright (C) 1996 - 2018
      UPX 3.95w      Markus Oberhumer, Laszlo Molnar & John Reiser      Aug 26th 2018

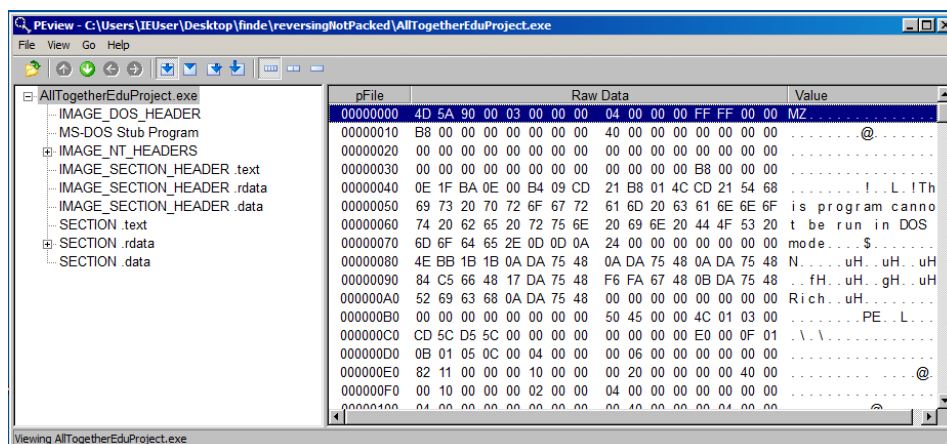
      File size      Ratio      Format      Name
      -----
      3584 <-      3072      85.71%      win32/pe      AllTogetherEduProject.exe

Unpacked 1 file.
C:\Users\IEUser\Desktop\finde\reversingUnpacked>
```

De todas formas, durante el análisis estático básico existen otros marcadores que pueden indicar al analista que estamos tratando con un archivo empaquetado. Se irán remarcando según aparezcan.

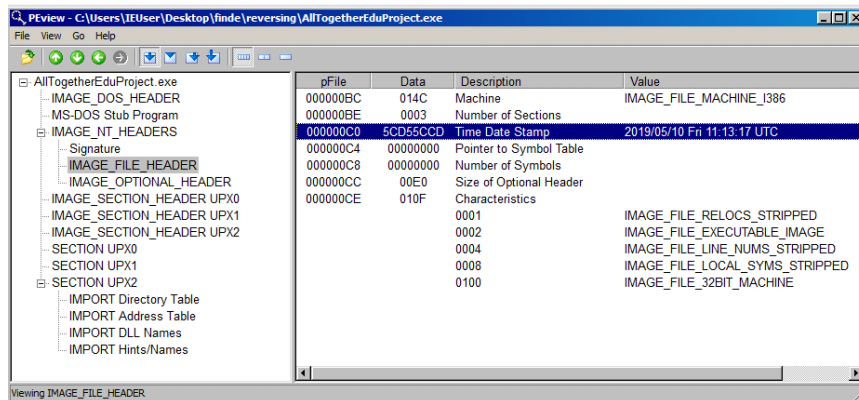
## 1.3- PView

Como curiosidad, lo primero que vemos al abrir la muestra con esta herramienta es que comienza con los bytes propios de todo Portable Ejecutable – MZ (0x4D5A)



Esta herramienta nos proporciona también información temporal de cuándo se compiló la muestra. Nos ayuda a saber si estamos tratando con algo nuevo o no. Esta información aparece en los encabezados como muestra la imagen:

- IMAGE\_NT\_HEADERS / IMAGE\_FILE\_HEADER

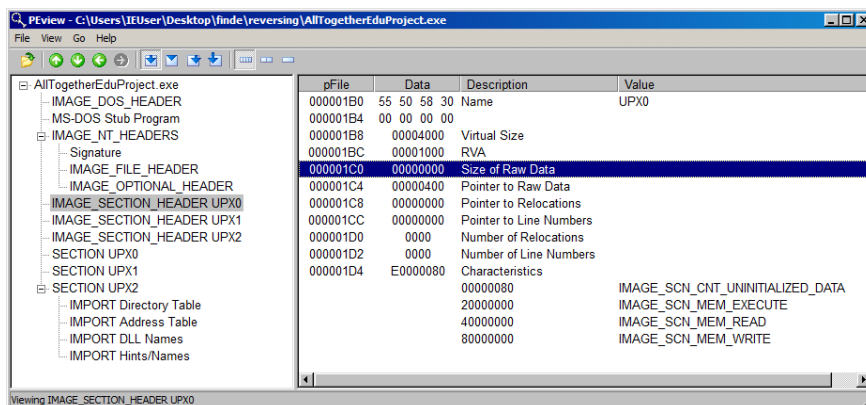


Por último, esta herramienta nos proporciona dos indicadores para saber si un binario está empaquetado.

#### Indicador1.- Memoria Virtual vs Memoria Raw.

Dentro de cada una de las secciones se muestra el tamaño de los datos y el tamaño virtual de los mismos. Si una sección ocupa mucho más espacio en memoria de lo que lo hace en disco, es un buen indicativo de que el binario ha sido empaquetado. Principalmente es bueno fijarse en la sección .text

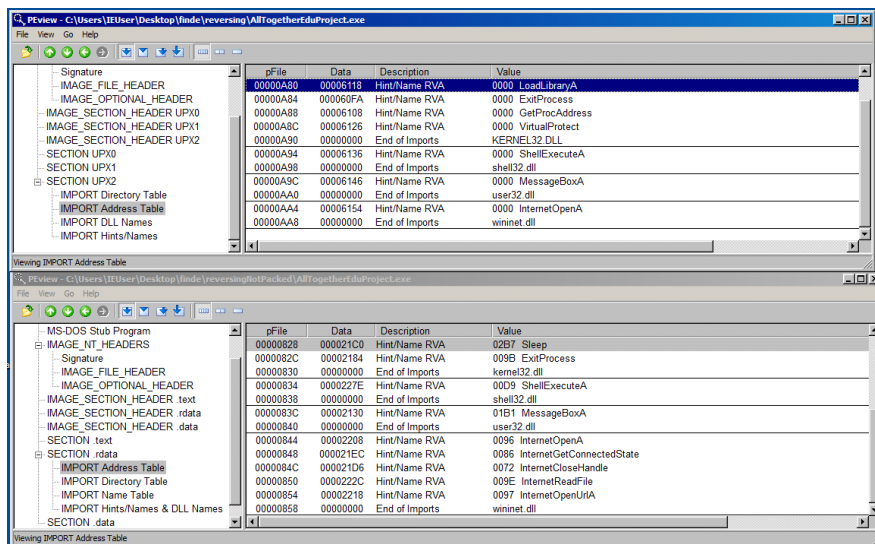
En ciertos casos, como este, al haberse usado UPX, incluso los nombres de las secciones están cambiadas y muestran los nombres que este packer utiliza.



Indicador2.- Esta herramienta nos muestra la tabla de Imports. Para realizar sus algoritmos importa librerías, en este caso podemos ver varias dll de microsoft, y utiliza sus funciones.

Podemos ver una recopilación de dichas funciones en la sección de recursos SECTION .rdata/ IMPORT Address Table

Cuando el archivo está empaquetado, la lista de funciones expuestas en la tabla de IMPORTS es mucho menor. Podemos ver en la siguiente imagen una comparación del archivo empaquetado (arriba) y tras desempaquetarlo (abajo). Para una misma librería que en ambos casos reconoce se esta usando como *wininet.dll* se encuentran muchas mas funciones en la parte inferior al no estar comprimido.

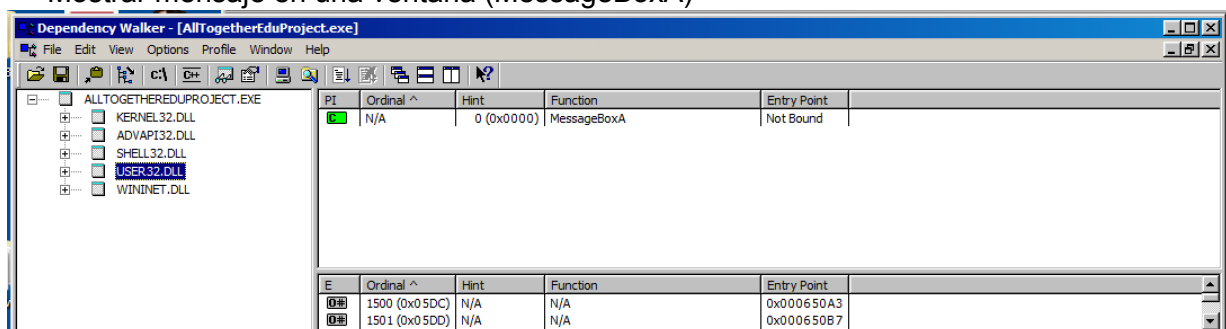


## 1.4- DependencyWalker

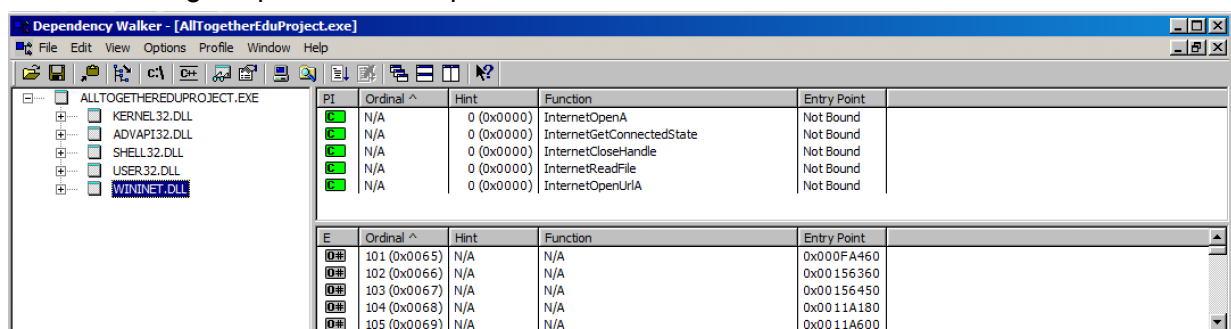
Esta herramienta escanea archivos PE y construye un diagrama jerárquico de todos los módulos que encuentra y su dependencias.

En esta muestra podemos ver que se importan varias librerías (kernel32, advapi32, shell32, user32, y wininet). Si vamos escaneando cada una de ellos, a la derecha en verde nos muestra los métodos de las mismas que se encontraron. Tras revisarlos todos, tenemos indicios para pensar que este binario podría:

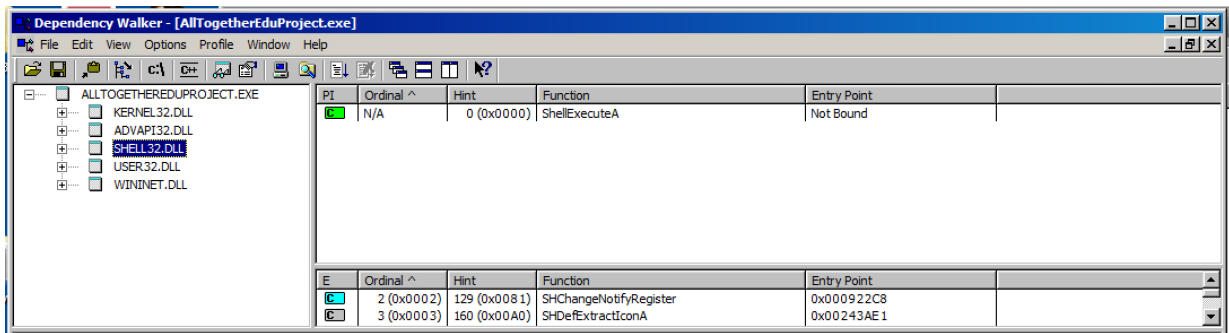
- Mostrar mensaje en una ventana (MessageBoxA)



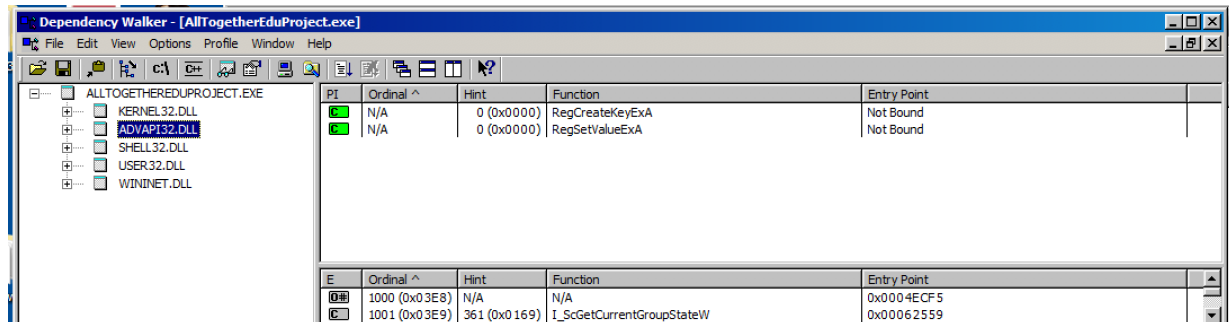
- Realizar algún tipo de llamada por internet



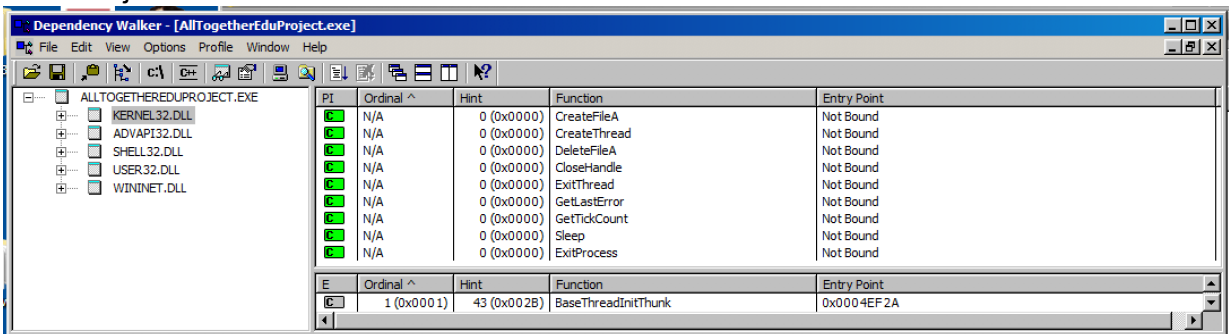
## - Lanzar una Shell



## - Crear una clave en el registro



## - Trabajar con archivos



Toda esta parte de los imports es muy útil cuando lleguemos a la parte de análisis estático avanzado,

### 1.5- Strings (<https://docs.microsoft.com/en-us/sysinternals/downloads/strings>)

Buscar las cadenas de caracteres que un programa utiliza nos puede dar una idea de su funcionalidad, fácilmente mostrarnos marcadores de red o de host.

Utilizamos la herramienta Strings de Microsoft para inspeccionar la muestra: (empaquetada a la izquierda y sin empaquetar a la derecha)

La cantidad de cadenas de caracteres encontrados en un binario se considera otro indicador de compromiso para saber si está empaquetado o no.

En este caso podemos ver que la muestra empaquetada de hecho tiene más valores. Esto a priori podría parecer raro, pero con una simple inspección visual es obvio la diferencia de sentido semántico entre ambas ventanas. La muestra no empaquetada tiene strings con mucho más sentido mientras que a la izquierda la mayoría de ellos no son descifrables.



```

C:\Users\IEUser\Desktop>find /reversing>strings AllTogetherEduProject.exe | wc -l
1
'wc' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\IEUser\Desktop>find /reversing>strings AllTogetherEduProject.exe | find /c /v ""
99

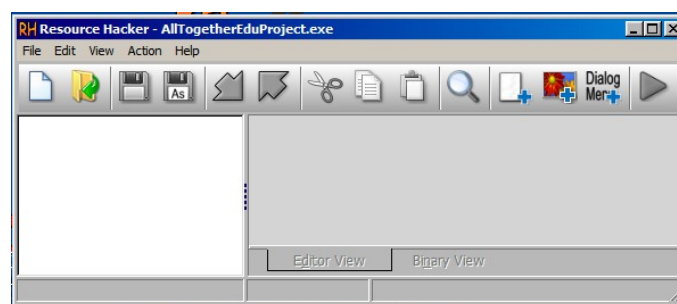
C:\Users\IEUser\Desktop>find /reversing>strings AllTogetherEduProject.exe | find /c /v ""
78

C:\Users\IEUser\Desktop>find /reversing>strings AllTogetherEduProject.exe | find /c /v ""
78

```

## 1.6- ResourceHacker (<https://docs.microsoft.com/en-us/sysinternals/downloads/strings>)

Se nos ocurre también usar otras herramientas como “Resource Hacker” para comprobar si la muestra contiene otros recursos, pero no observamos nada



Esta herramienta es muy útil, algunas muestras contienen otros binarios en su interior, que esta herramienta permite exportar y así realizar un análisis por separado.

**Nota!** → Recopilación de los indicadores a comprobar para intentar identificar si un archivo esta empacado:

- Exeinfo PE
- Memoria Virtual vs Memoria Raw: fácilmente identificable con PEView
- Numero de:
  - Imports identificables
  - Cadenas de texto “con sentido semántico”



## 2.- Análisis dinámico básico:

En un análisis dinámico básico ya se procede a la ejecución de la muestra. Es esencial haber preparado previamente todas las herramientas que vamos a utilizar para obtener marcadores en un análisis a posteriori.

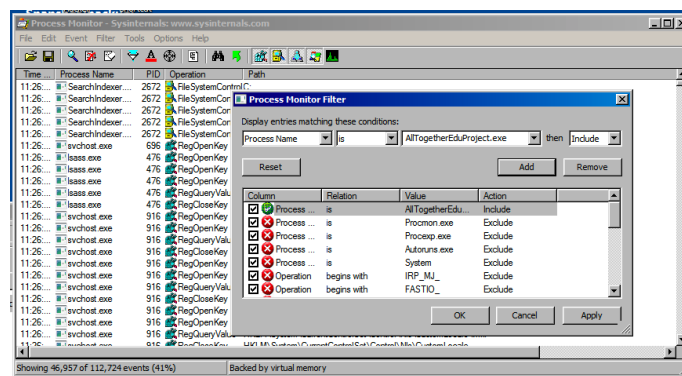
Prepararemos un entorno con las siguientes herramientas:

7. Process Monitor - ProcMon
8. Process Explorer
9. Snapshot of Registry - Regshot
10. Networks
  1. ApateDNS
  2. InetSim
11. Network logging con wireshark

### 2.7- Process Monitor (<https://docs.microsoft.com/en-us/sysinternals/downloads/procmon>)

ProcMon es una herramienta de monitorización avanzada de Windows que permite observar en tiempo real modificaciones en archivos del sistema, registro, procesos y actividades de hilos.

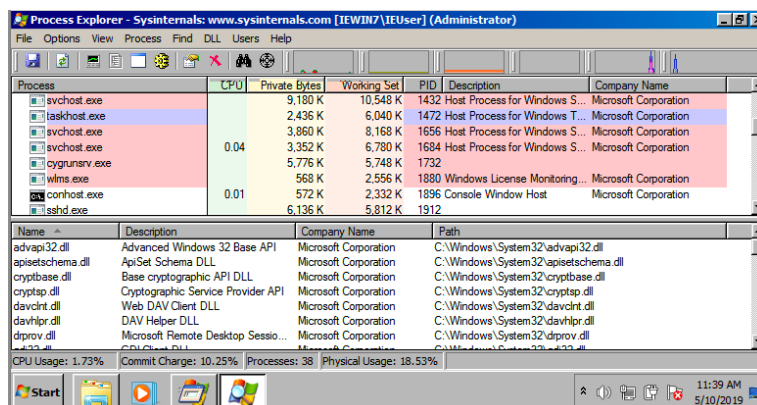
Tiene unos filtros muy útiles para facilitar el entendimiento de lo que está pasando, pero guarda los datos en memoria, así que es mejor no tenerlo activado recibiendo eventos durante mucho tiempo a no ser que sea imprescindible.



Lo dejamos preparado ya para filtrar solo los eventos de nuestro proceso *AllTogetherEduProject.exe*

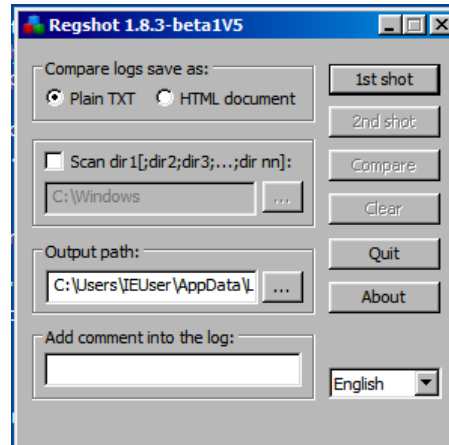
### 2.8- Process Explorer (<https://docs.microsoft.com/en-us/sysinternals/downloads/process-explorer>)

Process Explorer muestra información sobre qué handlers están abiertos y que dls han sido cargadas en un proceso.



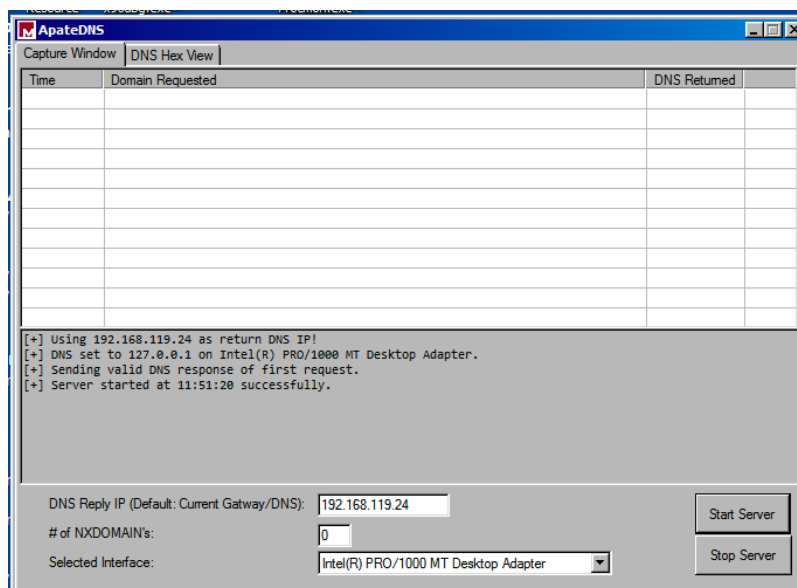
## 2.9- Registry Snapshot – RegShot (<https://sourceforge.net/projects/regshot/> )

Esta herramienta permite comparar el estado del registro de windows entre dos instantes determinados y muestra las diferencias. Para utilizarla, tomaremos una captura antes de ejecutar la muestra y otra después de que todo haya terminado. Y nos permitirá ver si el proceso lanzado ha tenido algún contacto con el registro de Windows.



### 2.10.1- Simular la Red: ApateDNS

ApateDNS nos permite controlar la respuesta a todas las peticiones DNS realizadas en una máquina con una interfaz muy sencilla.



Tan solo especificando la IP 192.168.119.24 en la parte inferior, todas las peticiones DNS que haga nuestra máquina cuando ejecutemos la muestra, serán interceptadas por ApateDNS y reenviadas a la máquina 192.168.119.24 que controlamos nosotros. (además de listarlas y mostrarlas en la cuadrícula de la parte superior de la ventana)

Así en dicha máquina, .24 se podría usar netcat para levantar servicios clásicos y comprobar las peticiones que llegan, o podemos utilizar INETSim

## 2.10.2- Simular la Red: INetSim

InetSim es una herramienta para windows que permite simular servicios comunes de internet en un entorno de laboratorio. La lista es extensa y podemos verlo cuando se lanza:

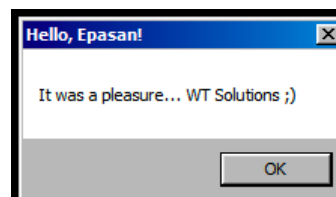
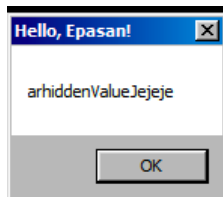
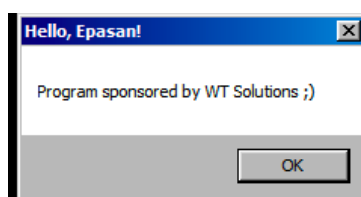
```
neska@neska-VirtualBox:~$ sudo inetsim
[sudo] password for neska:
INetSim 1.2.8 (2018-06-12) by Matthias Eckert & Thomas Hungenberg
Using log directory:      /var/log/inetsim/
Using data directory:     /var/lib/inetsim/
Using report directory:   /var/log/inetsim/report/
Using configuration file: /etc/inetsim/inetsim.conf
Parsing configuration file.
Configuration file parsed successfully.
=== INetSim main process started (PID 6952) ===
Session ID:      6952
Listening on:    127.0.0.1
Real Date/Time:  2019-05-10 20:51:06
Fake Date/Time: 2019-05-10 20:51:06 (Delta: 0 seconds)
Forking services...
* dns_53_tcp_udp - started (PID 6954)
* ident_113_tcp - started (PID 6967)
* time_37_tcp - started (PID 6969)
* syslog_514_udp - started (PID 6968)
* irc_6667_tcp - started (PID 6964)
* finger_79_tcp - started (PID 6966)
* daytime_13_tcp - started (PID 6971)
* echo_7_tcp - started (PID 6973)
* discard_9_tcp - started (PID 6975)
* discard_9_udp - started (PID 6976)
* quotd_17_tcp - started (PID 6977)
* chargen_19_udp - started (PID 6980)
* daytime_13_udp - started (PID 6972)
* echo_7_udp - started (PID 6974)
* time_37_udp - started (PID 6970)
* tftp_69_udp - started (PID 6963)
* chargen_19_tcp - started (PID 6979)
* ntp_123_udp - started (PID 6965)
* ftp_21_tcp - started (PID 6961)
* quotd_17_udp - started (PID 6978)
* dummy_1_udp - started (PID 6982)
* https_443_tcp - started (PID 6956)
* ftps_990_tcp - started (PID 6962)
* pop3s_995_tcp - started (PID 6960)
* pop3_110_tcp - started (PID 6959)
* http_80_tcp - started (PID 6955)
* smtps_465_tcp - started (PID 6958)
* smtp_25_tcp - started (PID 6957)
* dummy_1_tcp - started (PID 6981)
done.
Simulation running.
```

## 2.11- Wireshark

Wireshark es una herramienta de sniffing de tráfico de red muy conocida. Se puede utilizar para analizar la interacción de una muestra con el exterior. En este caso no se ha utilizado ya que no resultaba de gran utilidad y ya se estaban utilizando muchas y variadas aplicaciones para monitorizar la ejecución.

..... Arrancamos todas las herramientas, y lanzamos la muestra. Analicemos los **resultados**....

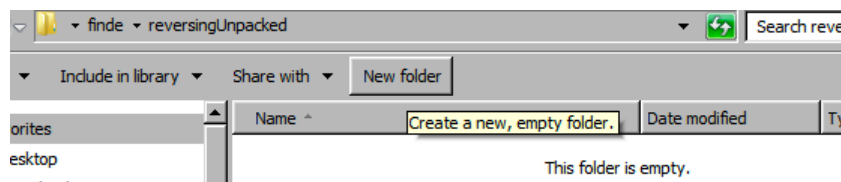
Podemos ver que nos saltan varias ventanas con mensajes:



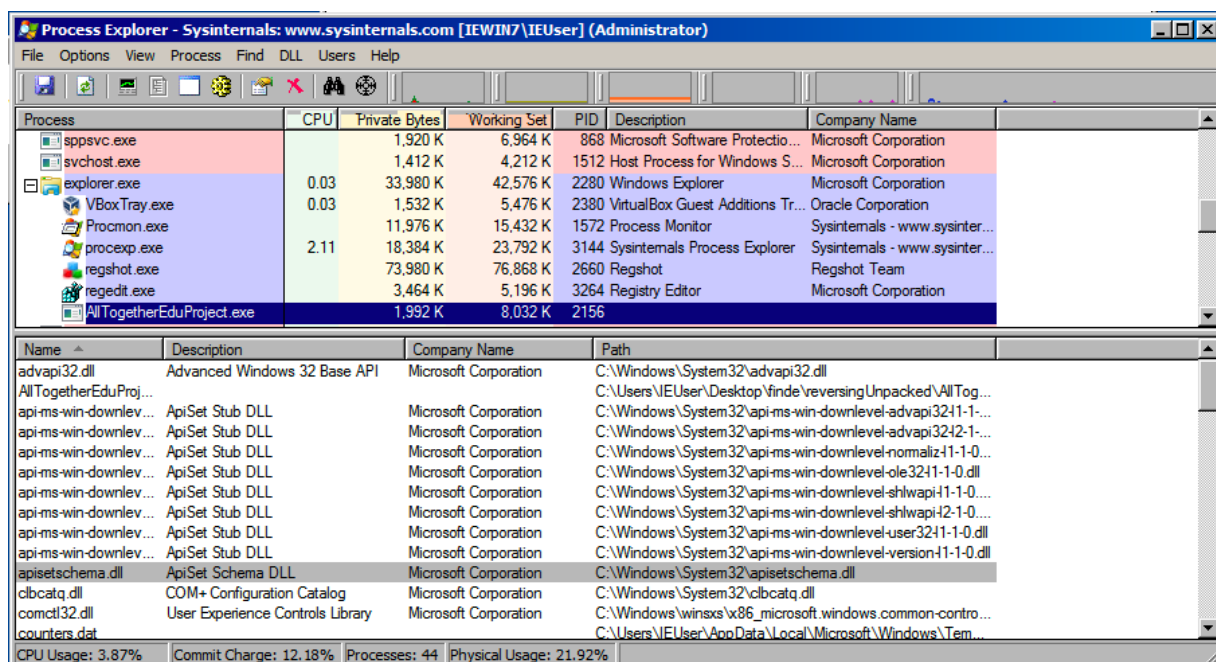
En la carpeta donde se ha ejecutado la muestra vemos que aparece otro archivo de nombre "temp".



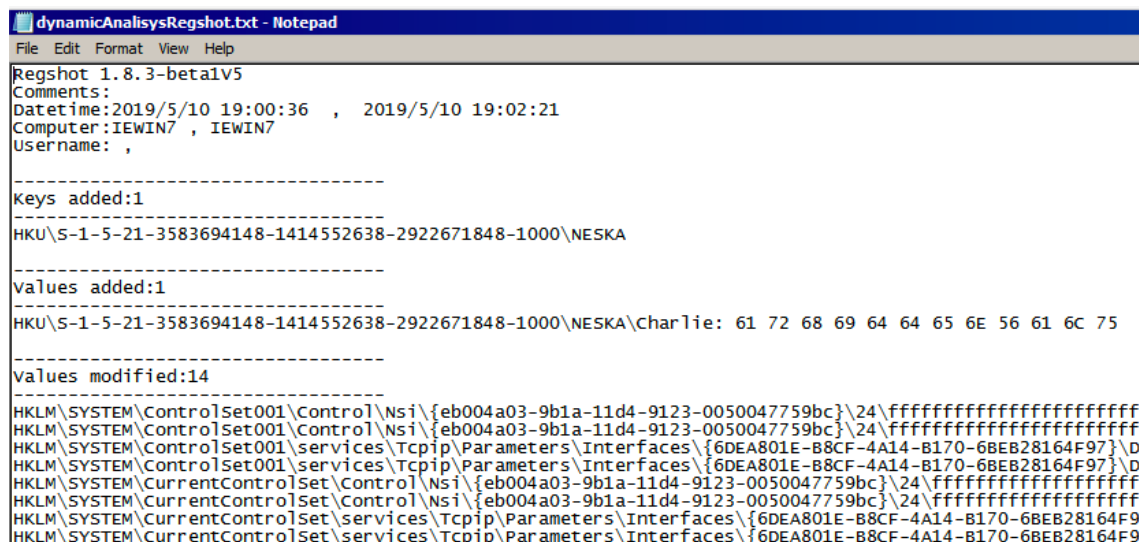
Pero poco después desaparecen tanto el archivo creado como el propio ejecutable una vez que termina la ejecución.



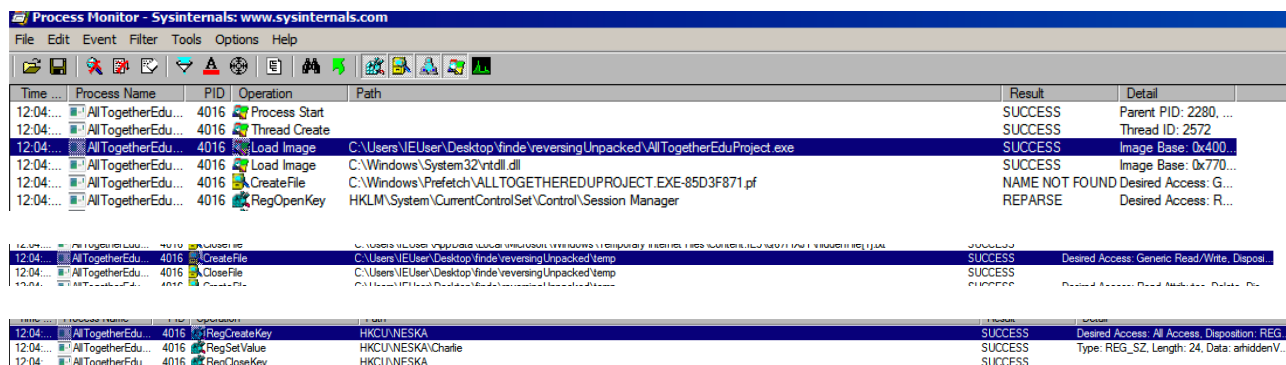
Mientras se estaba ejecutando la muestra se ha podido ver en el Process Explorer, pero no hemos visto que lanzara ningún proceso extra, era una muestra simple.



De la herramienta RegShot observamos que se ha creado una nueva clave *NESKA* en el registro y se le ha dado un valor:



La creación de la clave en el registro, y el archivo temporal, ha dejado su huella en el Process Monitor. Si no nos hubiéramos dado cuenta, siempre estaría ahí Aunque hay que saber leerlo ya que tiene mucha cantidad de información.



The screenshot shows the Process Monitor window with a list of events. The following table represents the data visible in the main pane:

| Time      | Process Name      | PID  | Operation     | Path  | Result         | Detail  |
|-----------|-------------------|------|---------------|---|----------------|---|
| 12:04:... | AllTogetherEdu... | 4016 | Process Start |   | SUCCESS        | Parent PID: 2280, ...                           |
| 12:04:... | AllTogetherEdu... | 4016 | Thread Create |   | SUCCESS        | Thread ID: 2572                                 |
| 12:04:... | AllTogetherEdu... | 4016 | Load Image    | C:\Users\IEUser\Desktop\finde\reversingUnpacked\AllTogetherEduProject.exe | SUCCESS        | Image Base: 0x400...                            |
| 12:04:... | AllTogetherEdu... | 4016 | Load Image    | C:\Windows\System32\ntdll.dll   | SUCCESS        | Image Base: 0x770...                            |
| 12:04:... | AllTogetherEdu... | 4016 | CreateFile    | C:\Windows\Prefetch\ALLTOGETHEREDUPROJECT.EXE-85D3F871.pf                 | NAME NOT FOUND | Desired Access: G...                            |
| 12:04:... | AllTogetherEdu... | 4016 | RegOpenKey    | HKLM\System\CurrentControlSet\Control\Session Manager                     | REPARSE        | Desired Access: R...                            |
| 12:04:... | AllTogetherEdu... | 4016 | CloseFile     | C:\Users\IEUser\Desktop\finde\reversingUnpacked\temp                      | SUCCESS        | Desired Access: Generic Read/Write, Disposi...  |
| 12:04:... | AllTogetherEdu... | 4016 | CloseFile     | C:\Users\IEUser\Desktop\finde\reversingUnpacked\temp                      | SUCCESS        | Desired Access: Generic Read/Write, Disposi...  |
| 12:04:... | AllTogetherEdu... | 4016 | RegOpenKey    | HKCU\NESKA  | SUCCESS        | Desired Access: All Access, Disposition: REG... |
| 12:04:... | AllTogetherEdu... | 4016 | RegSetValue   | HKCU\NESKA\Charlie  | SUCCESS        | Type: REG_SZ, Length: 24, Data: ahidhenV...     |
| 12:04:... | AllTogetherEdu... | 4016 | RegCloseKey   | HKCU\NESKA  | SUCCESS        |   |

De las herramientas de simulación de red no se ha obtenido información ya que la llamada a la url se realizaba a una IP fija. Pero son una herramientas extremadamente útil.

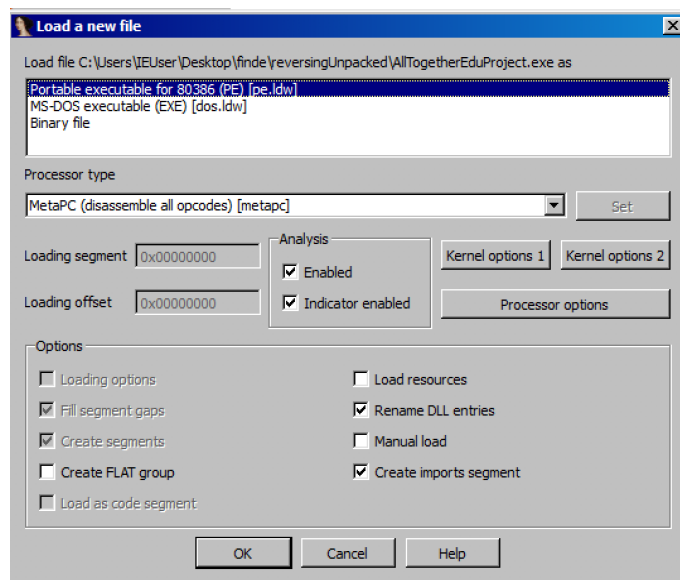
### 3.- Análisis estático avanzado:

La herramienta por excelencia para este apartado es:

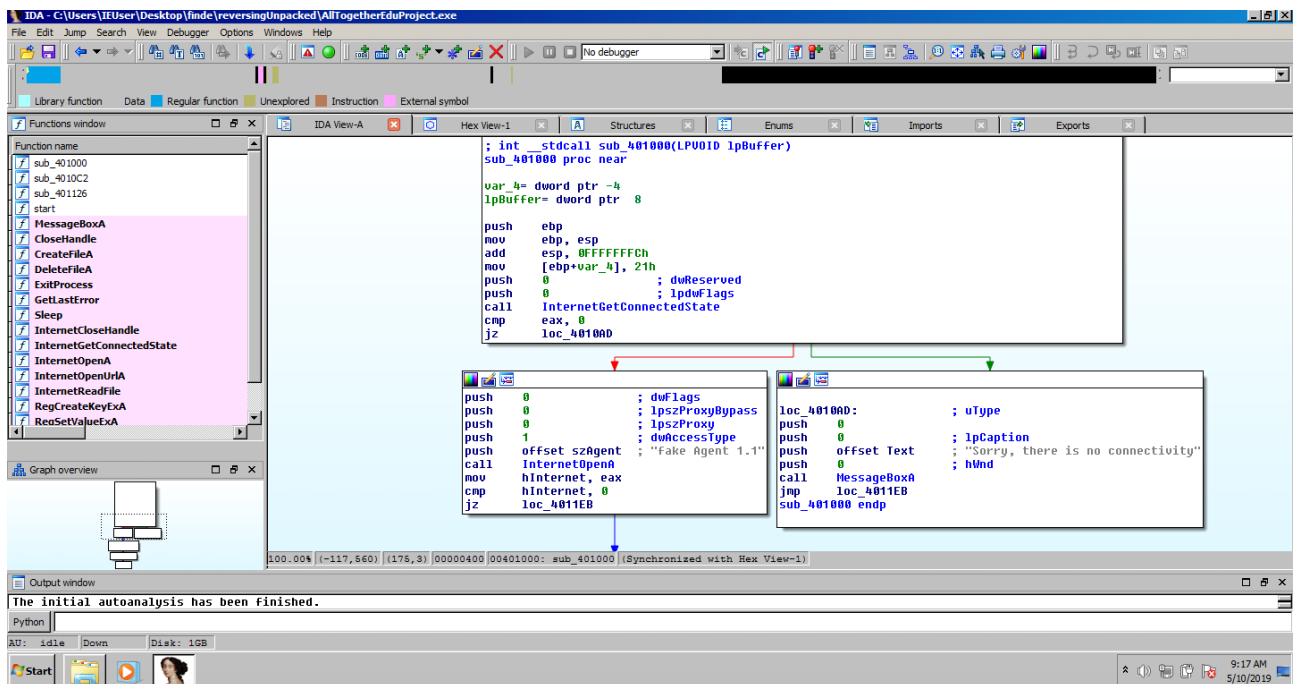
12. IDA pro (32 bits)

IDA es una herramienta que permite obtener el código ensamblador tomando como origen una muestra binaria.

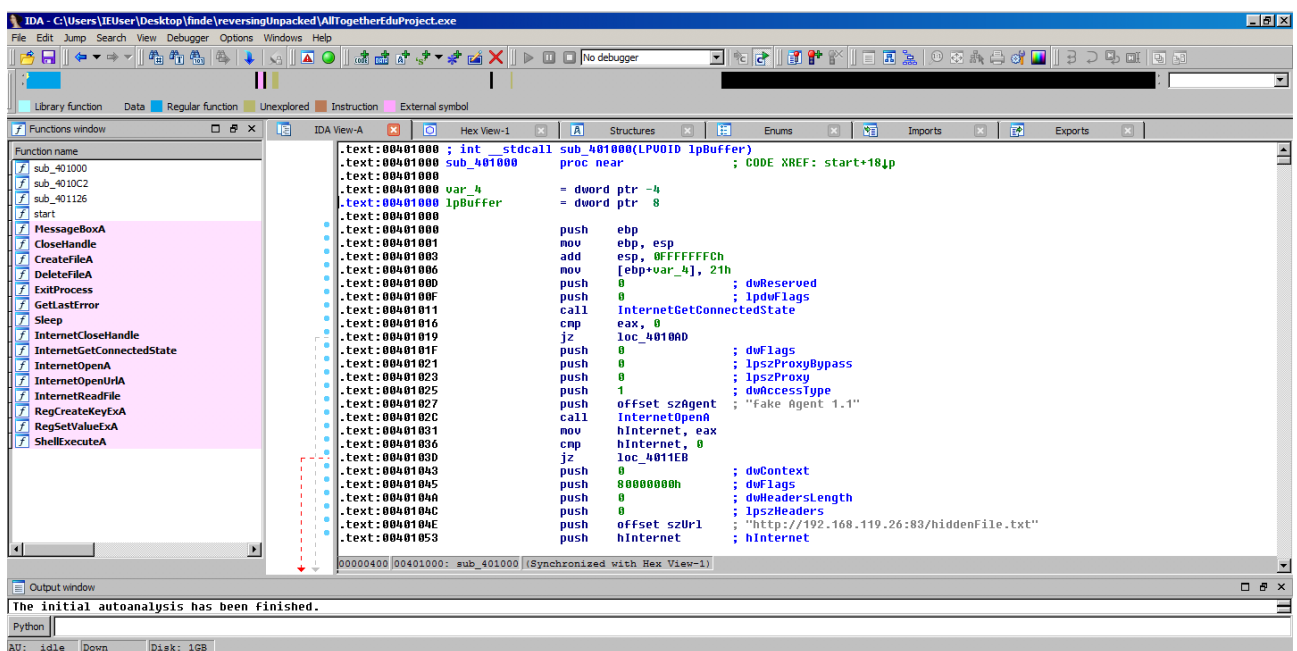
Para desensamblar un binario con IDA tan solo es necesario arrastrar el archivo hasta el icono del programa y se abrirá automáticamente mostrando la siguiente ventana:



Una vez abierto, IDA ...



IDA tiene dos modos para visualizar el código, gráfico (arriba) y modo texto (abajo). Para pasar de uno a otro solo es necesario presionar la barra espaciadora



En la parte de la izquierda vemos una lista de las funciones que la herramienta entiende que existen. Tienen un código de colores. Aquellas que vemos en rosa son propias de microsoft, en principio son funciones conocidas, por lo que a nosotros nos interesa centrarnos en las primeras de la lista sin color.

IDA nos proporciona más información que clasifica en varias ventanas que se muestran en cada una de las pestañas.

IDA ha realizado el desensamblado del binario y nos muestra que nuestro método principal es:

```
public start
start proc near
push 0 ; uType
push offset Caption ; "Hello, Epasan?"
push offset aProgramSponsor ; "Program sponsored by WT Solutions ;)"
push 0 ; hWnd
call MessageBoxA
push offset Data ; lpBuffer
call sub_401000
push 0 ; uType
push offset Caption ; "Hello, Epasan?"
push offset Data ; lpText
push 0 ; hWnd
call MessageBoxA
push offset Data ; lpData
call sub_4010C2
call sub_401126
push 0 ; uType
push offset Caption ; "Hello, Epasan?"
push offset aItWasAPleasure ; "It was a pleasure... WT Solutions ;)"
push 0 ; hWnd
call MessageBoxA
push 0Ah ; nShowCmd
push 0 ; lpDirectory
push offset Parameters ; "Timeout /T 2 & /C del .\\AllTogetherEdu"
push offset File ; "cmd.exe"
push 0 ; lpOperation
push 0 ; hWnd
call ShellExecuteA
```

A la derecha vemos como comentarios los valores de las cadenas de caracteres que ha encontrado. Este programa realiza básicamente 4 llamadas a diferentes funciones:

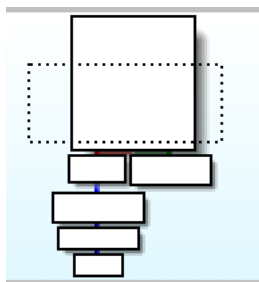
- sub\_401000
- sub\_4010C2
- sub\_401126
- ShellExecuteA

Haciendo doble click en cualquiera de ellas se puede navegar dentro de la función para investigar su contenido. Analicémoslas una por una:

- Analizaremos **sub\_401000**

```
push offset Data ; lpBuffer
call sub_401000
```

Antes de la llamada a la función se mete un valor en la pila, un puntero a lpBuffer. Este es el argumento que la función recibe. Una vez dentro vemos en el gráfico que esta función es un poco más entretenida



Ataquemos al código por partes. Nada más empezar vemos dos variables definidas:

```
.text:00401000 sub_401000 proc near
.text:00401000
.text:00401000 var_4 = dword ptr -4
.text:00401000 lpBuffer = dword ptr 8
```



La segunda la reconocemos, es el argumento que se ha pasado a la función "lpBuffer". Los argumentos se les puede reconocer ya que tienen un offset de memoria positivo (8 en este caso). Esto es debido a que se mete en la pila de memoria (stack) justo antes de llamar a la función. Recordar que la pila crece hacia unidades de memoria menores, por lo que para acceder al valor del argumento hay que sumarle un offset al inicio del frame de la función en la stack.

Por el contrario, var\_4 tiene un valor del offset negativo. Esto indica que se ha añadido a la pila después de llamar a la función, así que debe ser una variable local.

Lo siguiente que vemos es un prologo clásico en todas las funciones, la reorganización de los registros EBP y ESP para adaptarse a un nuevo frame en la pila para la nueva función

```

.text:00401000      push    ebp
.text:00401001      mov     ebp, esp

```

Todo esto cuadra con lo explicado al principio del informe sobre los registros especiales EBP, ESP, EIP y su integración con la pila y las llamadas a funciones.

El resto de la función contiene la lógica, realiza varias llamadas a funciones de windows. El proceso aquí cada vez que nos encontramos una de estas llamadas es ir a la documentación de msdn y comprobar como funciona, que parámetros recibe y cuales devuelve. En nuestro caso:

- Comprobar que existe conexión a internet

```

.text:00401000      push    0          ; dwReserved
.text:0040100F      push    0          ; lpdwFlags
.text:00401011      call    InternetGetConnectedState
.text:00401016      cmp     eax, 0
.text:00401019      jz      loc_4010AD

```

- Intentar abrir una conexión

```

.text:0040101F      push    0          ; dwFlags
.text:00401021      push    0          ; lpzProxyBypass
.text:00401023      push    0          ; lpzProxy
.text:00401025      push    1          ; dwAccessType
.text:00401027      push    offset szAgent ; "fake Agent 1.1"
.text:0040102C      call    InternetOpenA

```

- Acceder a una url específica

```

.text:00401043      push    0          ; dwContext
.text:00401045      push    80000000h   ; dwFlags
.text:0040104A      push    0          ; dwHeadersLength
.text:0040104C      push    0          ; lpzHeaders
.text:0040104E      push    offset szUrl ; "http://192.168.119.26:83/hiddenFile.txt"
.text:00401053      push    hInternet   ; hInternet
.text:00401059      call    InternetOpenUrlA
.text:0040105E      mov     hFile, eax
.text:00401063      cmp     hFile, 0

```

- Leer los datos y guardarlos en el parámetro de entrada

```

.text:0040106A      jz      loc_4010EB
.text:00401070      push    offset dwNumberOfBytesRead ; lpdwNumberOfBytesRead
.text:00401075      push    64h        ; dwNumberOfBytesToRead
.text:00401077      push    [ebp+lpBuffer] ; lpBuffer
.text:0040107A      push    hFile       ; hFile
.text:00401080      call    InternetReadFile
.text:00401085      cmov    eax, 0

```

- Cerrar la conexión, limpiar handlers y volver

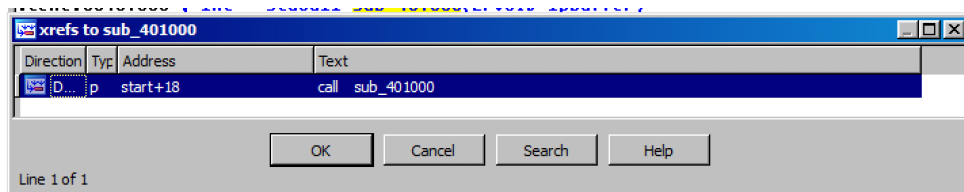
```

.text:0040108E      push    hInternet   ; hInternet
.text:00401094      call    InternetCloseHandle
.text:00401099      push    hFile       ; hInternet
.text:0040109F      call    InternetCloseHandle
.text:004010A4      mov     eax, 1
.text:004010A9      leave
.text:004010AA      retn     4

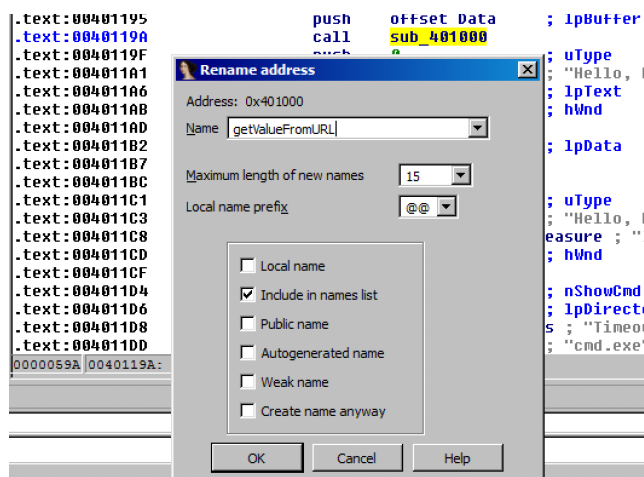
```

Existen diferentes convenciones en las llamadas a funciones. Estamos utilizando stdcall, por lo que vemos que la función llamada es la que se encarga de arreglar los valores en la pila antes de devolver el control a la función principal.

Para volver a la función principal podemos usar las referencias cruzadas. Ponemos el cursor en el nombre de la función `sub_401000` y damos a la tecla X. Nos aparecerá la siguiente ventana indicando todos los sitios en los que se llama a dicha función. Para nuestro caso, solo uno, desde `main`:



Una vez analizada esta función, lo más cómodo es renombrarla con un valor más semántico que nos facilite la tarea. Para ello ponemos el cursor encima del nombre de la función, presionamos la tecla N y ponemos el nuevo valor en la siguiente ventana que aparece.



- Analizaremos `sub_4010C2` sin entrar tan en detalle

Esta función es mas pequeña, se muestra entera:

```
.text:004010C2 sub_4010C2      proc near                               ; CODE XREF: start+351p
.text:004010C2                                     = dword ptr 8
.text:004010C2 lpData
.text:004010C2
.text:004010C2      push     ebp
.text:004010C3      mov      ebp, esp
.text:004010C5      push     offset dwDisposition ; lpdwDisposition
.text:004010CA      push     offset hKey          ; phkResult
.text:004010CF      push     0                    ; lpSecurityAttributes
.text:004010D1      push     0F003Fh              ; samDesired
.text:004010D6      push     0                    ; dwOptions
.text:004010DA      push     0                    ; lpClass
.text:004010DC      push     0                    ; Reserved
.text:004010DE      push     offset SubKey        ; "NESKA"
.text:004010E1      push     80000001h            ; hKey
.text:004010E6      call     RegCreateKeyExA
.text:004010EB      cmp      eax, 0
.text:004010EE      jnz      short loc_401112
.text:004010F0      push     0Ch                  ; cbData
.text:004010F2      push     [ebp+lpData]         ; lpData
.text:004010F5      push     1                    ; dwType
.text:004010F7      push     0                    ; Reserved
.text:004010F9      push     offset ValueName     ; "Charlie"
.text:004010FE      push     hKey                 ; hKey
.text:00401104      call     RegSetValueExA
.text:00401109      cmp      eax, 0
.text:0040110C      jnz      short loc_401112
.text:0040110E      leave
.text:0040110F      retn     4
.text:00401112
```

Dejando aparte el prólogo típico, declaración de variables, y parámetros, esta función llama al registro de windows y crea una Key con nombre "NESKA/Charlie" y con valor el recibido en lpData como parámetro.

Volviendo a la función principal vemos que el parámetro de esta función es el valor que se ha leído del contenido de la URL en el paso anterior.

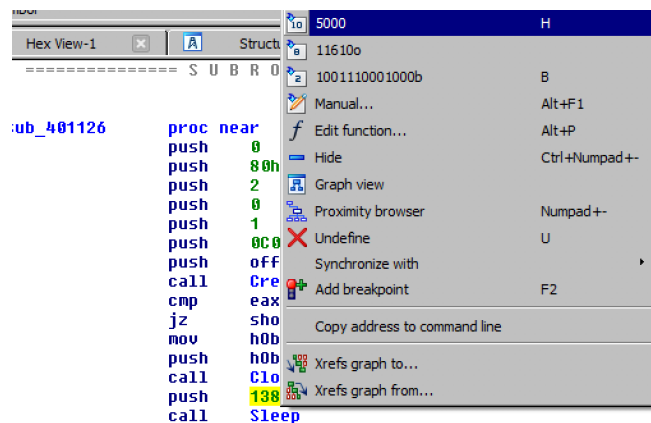
```
.text:00401190      call    MessageBoxA
.text:00401195      push    offset Data      ; lpBuffer
.text:0040119A      call    GetValueFromURL
.text:0040119F      push    0                ; uType
.text:004011A1      push    offset Caption    ; "Hello, Epanan!"
.text:004011A6      push    offset Data      ; lpText
.text:004011AB      push    0                ; hWnd
.text:004011AD      call    MessageBoxA
.text:004011B2      push    offset Data      ; lpData
.text:004011B7      call    sub_4010C2
```

Igualmente renombramos esta función por comodidad

- Analizamos la ultima funcion desconocida **sub\_401126**

Su contenido es bastante fácil de entender. Quitando el control de errores (loc\_40116C) de las llamadas a las funciones:

- crea un archivo llamado *temp* en el mismo directorio donde se ejecute la muestra
- Duerme durante 1388h segundos
- En IDA, haciendo click con el botón derecho en estos números, te permite pasarlo de hexadecimal a binario. Vemos que el valor son 5000 por lo que duerme durante 5 segundos



- Borra el archivo que acaba de crear.

- No quedan más funciones con nombre desconocido

Pero nos llama la atención la última línea dentro de la función principal, ya que realiza una llamada a ShellExecuteA

```

0040118F .text:0040118F      push    0           ; uType
00401191 .text:00401191      push    offset Caption ; "Hello, Epsan!"
00401193 .text:00401193      push    offset Data    ; lpText
00401195 .text:00401195      push    0           ; hWnd
00401197 .text:00401197      call    MessageBoxA
00401199 .text:00401199      push    offset Data    ; lpData
004011A1 .text:004011A1      call    saveRegistryKey
004011A3 .text:004011A3      call    crearBorrarArchivo
004011A5 .text:004011A5      push    0           ; uType
004011A7 .text:004011A7      push    offset Caption ; "Hello, Epsan!"
004011A9 .text:004011A9      push    offset altWasAPleasure ; "It was a pleasure... WT Solutions ;)"
004011AB .text:004011AB      push    0           ; hWnd
004011AD .text:004011AD      call    MessageBoxA
004011AF .text:004011AF      push    0Ah         ; nShowCmd
004011B1 .text:004011B1      push    0           ; lpDirectory
004011B3 .text:004011B3      push    offset Parameters ; "Timeout /T 2 & /C del .\\AllTogetherEdu"...
004011B5 .text:004011B5      push    offset File     ; "cmd.exe"
004011B7 .text:004011B7      push    0           ; lpOperation
004011B9 .text:004011B9      push    0           ; hWnd
004011BB .text:004011BB      call    ShellExecuteA

004011BD .text:004011BD      loc_4011BD:
004011BF .text:004011BF      push    0           ; hWnd
004011C1 .text:004011C1      call    ExitProc
004011C3 .text:004011C3      start
004011C5 .text:004011C5      endp
004011C7 .text:004011C7      ; [00000006 BYTES: COLLAPSED FUNCTION]
004011C9 .text:004011C9      ; [00000006 BYTES: COLLAPSED FUNCTION]
004011CB .text:004011CB      jmp     ds:__imp_ShellExecuteA

```

IDA tan útil como siempre, nos muestra en la parte gris, uno de los parámetros que le vamos a pasar a esta ejecución de Shell. Parece ser que primero ejecuta un Timeout con una espera de dos segundos y a continuación borra un archivo con el mismo nombre que tiene nuestra muestra.

Nos hace pensar que después de ejecutar las acciones que tenga que realizar, se borrará a sí mismo.

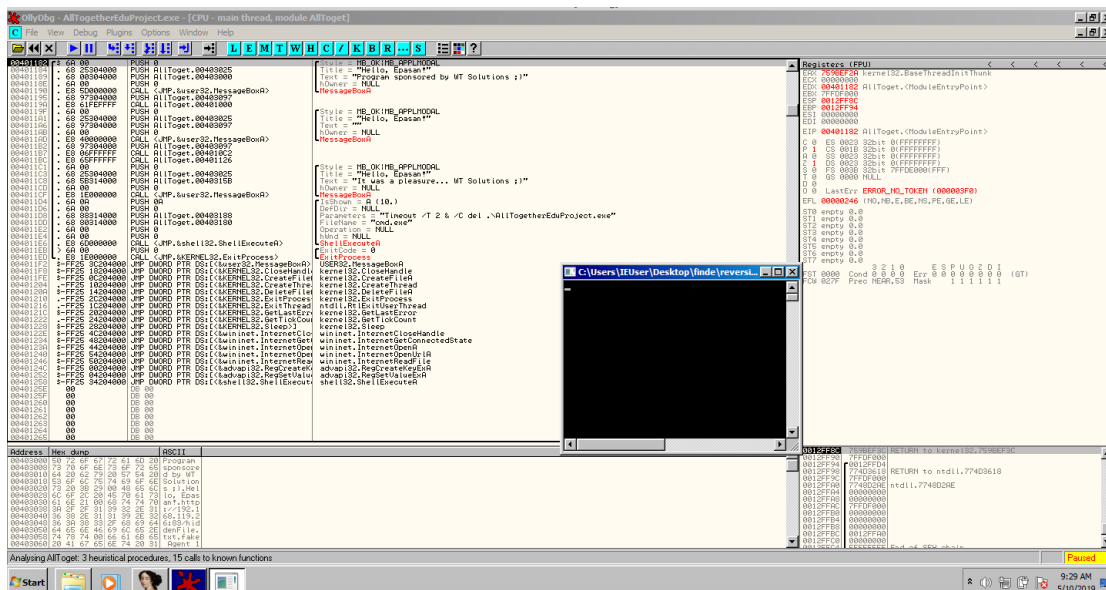
#### 4.- Análisis dinámico avanzado:

Existen varias herramientas que te permiten hacer debugging. Nosotros utilizamos:

##### 13. OllyDbg

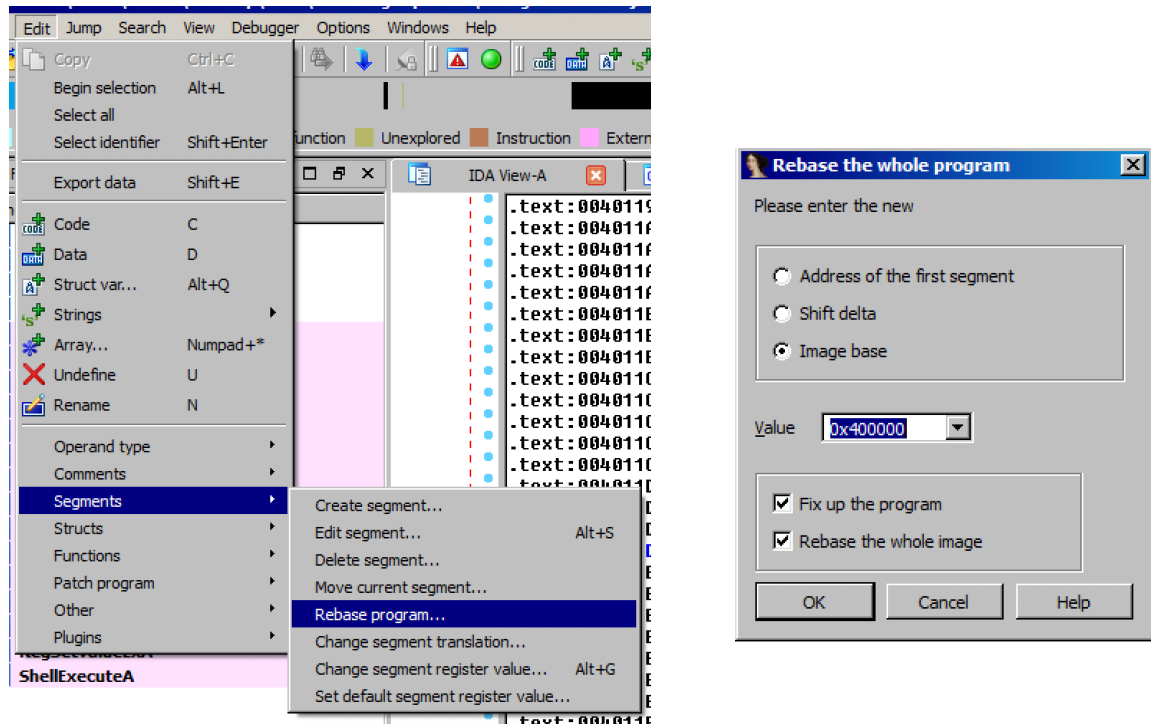
Que se vean funciones, imports, o valores de cadenas de caracteres no implica que de hecho se estén usando. La mejor manera de asegurarnos de lo que hace una muestra es seguir paso a paso sus pasos durante la ejecución.

OllyDbg está dividido en varias pantallas. La principal nos muestra las instrucciones del código. Durante la ejecución, mientras se va pasando por cada una de ellas se puede ver en las otras ventanas cómo los valores de los registros van cambiando y qué valores entran y salen de la pila.



Los puntos 3 y 4 (análisis avanzado) suelen ir de la mano para un analista de malware. Resulta interesante poder ir ejecutando paso a paso las acciones de la muestra mientras IDA muestra sus bondades del código desensamblado.

Para poder relacionar uno con el otro, es necesario que las direcciones de memoria que manejen relativa al programa sean las mismas. Esto se consigue alineando la memoria entre las dos herramientas (rebase). Se toma la dirección de memoria base de OllyDbg y se lleva a IDA:



No se va a explicar de forma práctica OllyDbg ya que para el tamaño de la muestra no se considera necesario. Sí decir que ha sido muy útil durante la creación de la misma en la parte de resolución de problemas, ya que permite poner puntos de ruptura (breakpoint) en las partes conflictivas y observar los valores de variables, registros y memoria.

Con el fin de tener un procedimiento específico y completo de como realizar análisis de malware es posible que el autor del presente informe lo complete con un video demostrativo utilizando el debugger en un futuro. Esta parte sería la más característica para realizar en una presentación o muestra.