Ing. Inversa de troyanos III

Bueno, les dejo la ultima parte de las "clases" de "ing. inversa de troyanos" hecha por **Bloodday**

Ya esta es la tercera clase, os felicito si habéis podido llegar hasta aquí con un profesor tan malo como yo…

Desde ahora empezaremos a practicar… y como su profesor dejo pendiente una explicación de cierta rutina, en cierto stub, aprovechara la clase para explicarla y enseñarles a hacer su propia rutina…

Lo que veremos hoy es conocido como “manual packing” o empaquetamiento manual, en el que lo que haremos será encriptar nuestro ejecutable favorito nosotros mismos directamente con el olly. Pero antes de entrar en el tema explicare la rutina del stub…

Y como seria muy tonto explicar una rutina de un ejecutable sin que los alumnos puedan “sentir que trabajan con el ejecutable” se los adjunto… (Los que entraron al Chat “por obligación” se sentirán algo molestos, pero no se preocupen que casi nadie pasa por este subforo –y probablemente sean recompensados-)

Antes de empezar, veamos un par de APIs que se utilizan aquí y que no se han explicado:

GetModuleHandleA

Obtiene el Handle o manejador de un modulo, siempre y cuando este se encuentre cargado en la memoria.

Código:
```
Push @offset nombre del modulo
Call GetModuleHandleA
```
Returns
La función devolverá el handle del modulo si tiene éxito, de lo contrario devolverá cero

GetProcAddress

Obtiene la dirección de memoria de una función exportada del modulo especificado, siempre y cuando el modulo esté cargado.
[Code]
Push @offset nombre o ordinal de la función
Push Handle del modulo.
[Code]
Returns
La función devolverá la dirección de memoria de la función. De fallar devolverá cero.

Bien, empecemos… apenas abrimos el ejecutable con el olly caemos en un Pushad

Podríamos explicar la rutina así, pero para hacer las cosas más sencillas utilizaremos el plugin del Olly AnalizeThis! (Si no lo tienen lo encontraran adjunto al final)

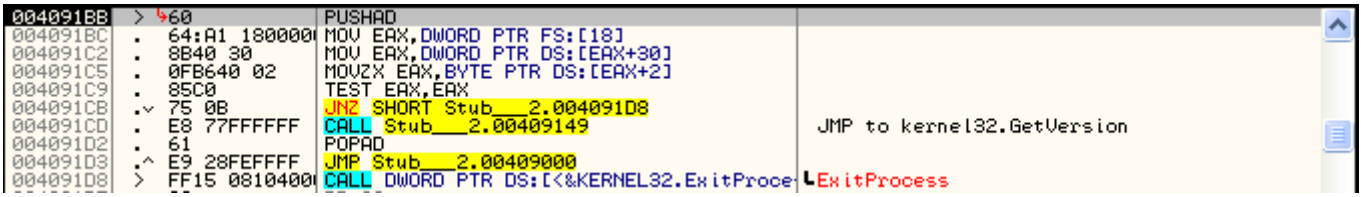
Vemos que ha cambiado un poco, pero gracias a ese cambio podemos ver una tabla de texto… si miramos bien podemos deducir que esta obteniendo las direcciones de memoria de las APIs que se mencionan en el cuadro de texto, y guarda esas direcciones en otro lado, y si nos fijamos bien, esos saltos que están abajo apuntan a hacia donde se guardaron esa direcciones de memoria. Es decir que es como una mini-IAT.

Ahí esta dividido por partes ese código.

En la parte 1 si, se darán cuenta que lo que hace es obtener el Handle o manejador del modulo “Kernel32.dll”

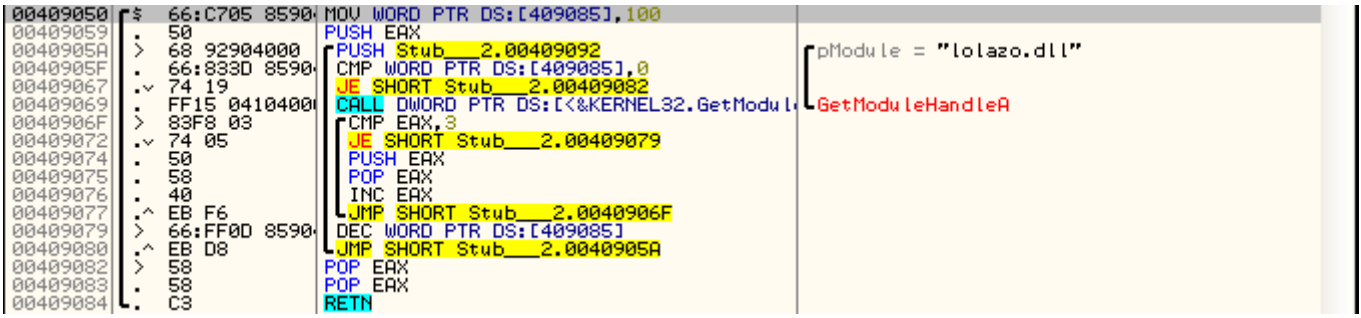
En la parte 2, mete el handle en el stack, pasa al siguiente elemento del cuadro de texto que se ve claramente en la imagen, y si son muy despistados (:ja:), es el que esta marcado con el numero 4. Compara si ese elemento es ultimo de la tabla (6179 son los bytes invertidos de “ya” en ASCII,) y si no lo es, entonces mete el handle de la dll (kernel32) en el stack y mete también, el elemento de la tabla al que apunta EBP para llamar a GetProcAddress, y hacer la mini-IAT.

Luego de eso, chequea el Bit de IsDebuggerPresent.



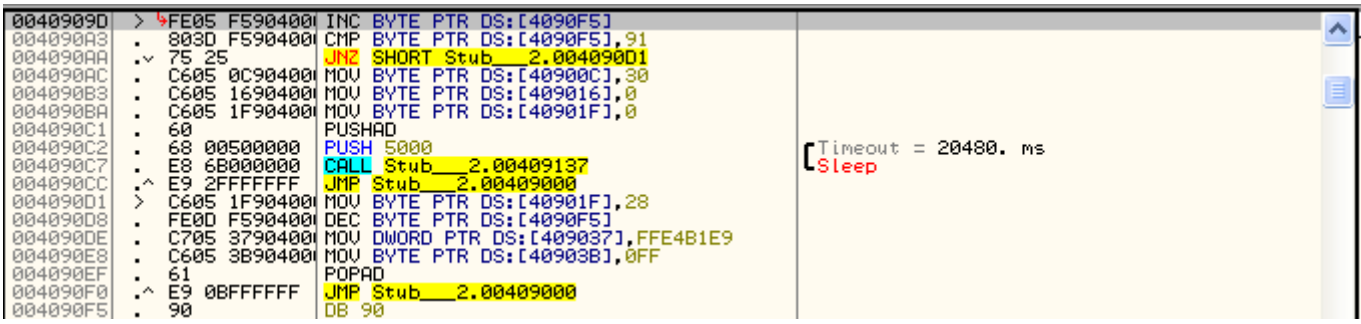
Y si esta a 1, nos lanzara afuera con un salto condicional que se ve que va directo a ExitProcess. Ahora ustedes dirán: como puede Chequear el Bit de IsDebuggerPresent si no llama a esa API? Pues la respuesta es sencilla, si en el olly oprimen Ctrl. + G y ponen ahí “IsDebuggerPresent” (con sus respectivas mayúsculas, o no funcionara) verán que en el lugar en el que caen, están las mismas primeras tres instrucciones que en la imagen que esta arriba de este párrafo (a excepción del pushad), pero si tenemos suerte, o mejor dicho, algún plugin que modifique ese bit (o lo modifican ustedes mismos), se hará una llamada a GetVersion, a la cual se le machacaran los datos que retorna con un popad y nos iremos en un vertiginoso salto hacia el inicio de la rutina de desencriptado (Al fin!!!)

Ahí vemos otro pushad, y vemos también que empezara a desencriptar desde 401020, que al parecer desencriptara el exe por bloques de 179h bytes y que cada tres bytes los encripta de forma distinta. Al llegar ECX a cero saltara hacia la call que esta abajo del salto incondicional, hacia el inicio del bucle encriptador, veamos que hay dentro de ese call…

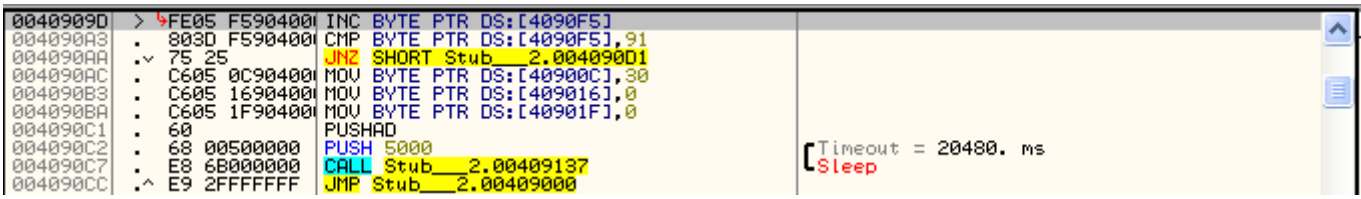


Pues, resumiré diciendo que es una rutina basura, en la cual se llama 100 veces a GetModuleHandleA pidiendo el handle o manejador de una dll Inexistente (lolazo.dll para ser mas exactos) y repite el bucle anidado ahí 300 veces (bucle que hace solo push EAX/pop EAX)

Al retornar vemos que se hace una comparación entre lo que hay en EAX y 4075BD, si no es igual nos manda a seguir desencriptando. Con esto sabemos que va a desencriptar hasta esa dirección, pero luego de eso hay un salto que va hacia abajo… veamos que nos depara la rutina por esos lares…

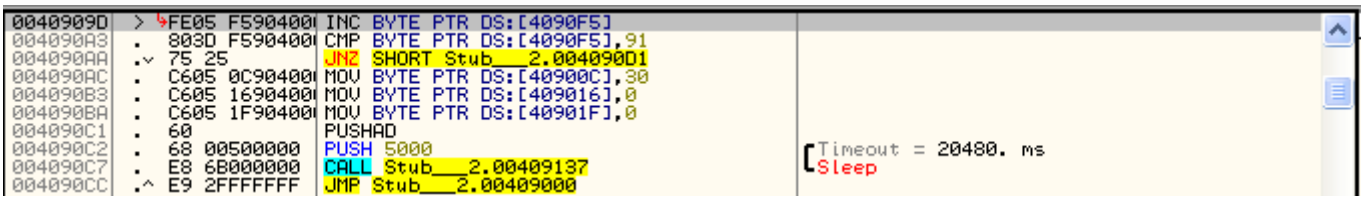


Este trozo de código, lo veremos en dos partes…



Ahí vemos que incrementa en uno el byte almacenado en 4090F5 y lo compara con 91, básicamente es para verificar si se ejecutara la primera parte del código a analizar o la segunda. Primero se ejecutara esta parte, en la que vemos que mueve ciertos valores hacia la rutina de desencriptado, así que lo mejor será ejecutarlo hasta el pushad.

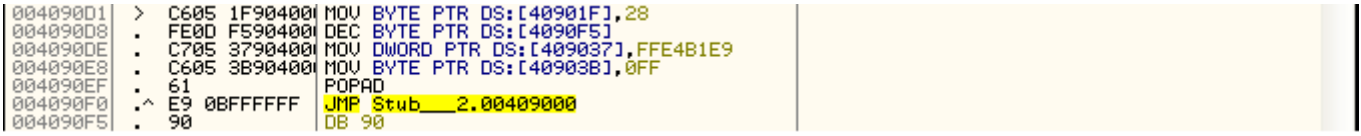
Una vez llegado aquí nos vamos hacia donde empieza la rutina para ver que cambios hizo…



Ahí nos fijamos que ha cambiado las operaciones de la rutina, el antiguo ADD ahora es un XOR, el antiguo ROR ahora es un ROL, y el antiguo XOR ahora es un ADD…

Siguiendo donde estábamos, hay un PUSH 5000 y una llamada a "Sleep", esta API lo que hace es parar la ejecución del programa el tiempo que se le diga, que en este caso son 5000h milisegundos (5000h = 20480d) y se nos manda al inicio de la rutina de desencriptado de nuevo.

Al llegar a desencriptar el byte de la dirección 4075BD vuelve a hacia la rutina que estamos analizando, y el salto decide que se ejecute la segunda parte…



Ahí vemos que cambia de nuevo la rutina, así que ejecutamos para ver que cambios se hacen…

Ahora solo hay dos cambios, el primero es que el antiguo ADD se convirtió en un SUB y el salto que estaba al final de la rutina de desencriptado que nos mandaba al último código analizado, ahora nos manda a 4074ED. Que ciertamente es nuestro entripoint.

Listo… un poco larga la explicación, pero a más de uno le servirá para entender muchas cosas (o eso creo). Ahora pasemos a algo más interesante…

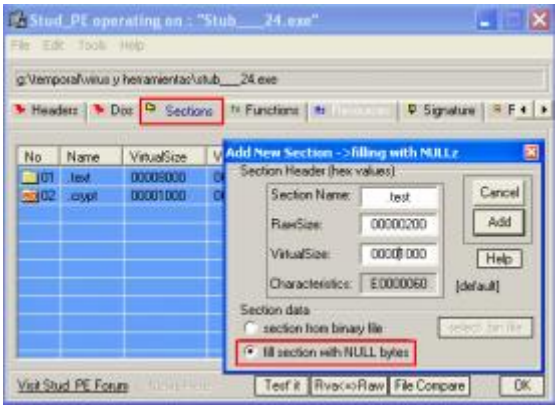
Haciendo Manual-Packing

Yo explicare como hacerlo con el Stub del Bifrost (versión 1.21 desempacada), ustedes pueden hacerlo con el stub que prefieran… (los enlaces de las herramientas utilizadas están al final)

Para no estar trasteando con ningún editor de recursos lo que vamos a hacer es sacar el stub dándole a builder… no importa la configuración que le den, solo denle al botón build que esta abajo a la derecha y les saldrá un mensaje, lo dejaran en el aire, es decir no lo toquen… abran la carpeta donde tienen el bif y copian el ejecutable que se llama Server.exe y lo ponen en otra carpeta, este será el que utilizaremos… ya pueden darle aceptar al cartel y cerrar el cliente y si quieren borrar el Server.exe que esta al lado del cliente (sobra decir que si utilizas otro stub tendrán que sacarlo con el editor de recursos)

Lo que tenemos que hacer ahora es agregar una nueva sección, para esto yo utilizare el Stud_pe, ustedes utilicen el PE Editor que prefieran.

Los que me siguen con el Stud_pe: arrastramos el Stub que acabamos de sacar y nos vamos a la pestaña sections, hacemos click secundario > New Section, y el cuadro que aparece ponemos RAW SIZE: 200 y VIRTUAL SIZE: 1000 (esto es por que en RAW SIZE solo se pueden Poner Múltiplos de 200 y en VIRTUAL SIZE múltiplos de 1000) ahora seleccionamos “fill section with NULL bytes”



Ahora le damos al botón “ADD” y ya tenemos lista la nueva sección, así que abrimos el stub con el olly y nos vamos a donde pusimos la nueva sección.

Para obtener el valor en memoria de la nueva sección lo que tenemos que hacer es sumar el IMAGEBASE y el VIRTUAL OFFSET (normalmente el IMAGEBASE tiene un valor de 400000 en los ejecutables) y la dirección resultante es la que pondremos en el olly. Otra forma de llegar a la dirección virtual de la sección es con el plugin OllyAdvanced en el que se marcara la casilla “enable Advaced Ctrl. + G” y al hacer presionar esa combinación marcamos donde dice offset en el cuadro que nos aparece y ponemos la dirección que nos aparece en RAW OFFSET.

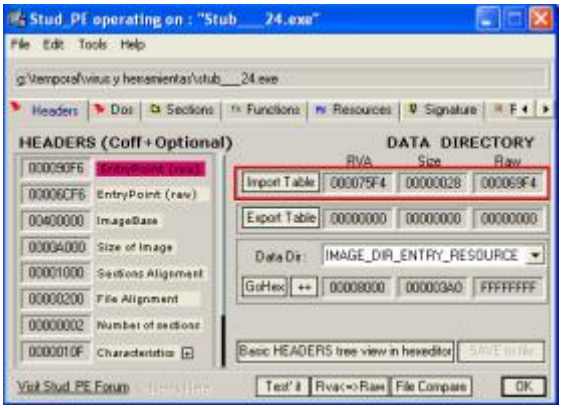
Una vez llegados acá, podremos empezar a escribir código para encriptar, pero deberemos tener en cuenta algo antes de encriptar y eso es la IAT y el IMPORT DESCRIPTOR.

La IAT la hallamos buscando algún call hacia cualquier API, cuando encontremos una miramos la dirección que esta entre corchetes ([]) y nos dirigimos hacia allá en el dump, y nos fijamos donde empieza y donde termina, en este caso empieza en 401000 y termina en 401020 (por eso empezaba a desencriptar desde ahí en la rutina explicada

arriba)

Ahora le toca el turno al IMPORT DESCRIPTOR, pero hay dos caminos a recorrer, el primero es que cuando terminemos de encriptar, dumpeemos, con lo cual el plugin OllyDump pondrá un IMPORT DESCRIPTOR en una nueva sección y no habrá que preocuparse por el, pero esto nos aumentara el tamaño (el original es de 23Kb y lo dejara de 41Kb) y el otro camino es hacer "copy to executable" con lo cual mantendremos el tamaño, pero deberemos preocuparnos por el IMPORT DESCRIPTOR así que elijan su camino.

Camino del "Copy to executable": deberemos hallar el IMPORT DESCRIPTOR y para ello nos ayudara el Stud_pe, en la primera pestaña hay una parte que dice "Import table" y tiene su dirección en RVA (Relative Virtual Address), RAW y el tamaño.



Esas direcciones son las del IMPORT DESCRIPTOR y como dije antes, para obtener su dirección virtual, se suma el RVA al IMAGEBASE. Y el resultado es el que se pone en el olly y para nuestra suerte, no hay nada después del IMPORT DESCRIPTOR, así que hasta ahí encriptaremos.

Aquí los caminos se unen…

La encriptación que haremos será una súper básica, pero que como vimos en la rutina explicada al principio, puede mejorarse.

Movemos hacia algún registro de uso general la dirección desde la que empezamos a encriptar…
Código:
```
MOV EAX,401020
```
Encriptamos el byte de la dirección a la que apunta el registro de uso general con cualquier operación matemática o lógica…
Código:
```
MOV EAX,401020
SUB BYTE PTR DS:[EAX],51
```
Incrementamos el valor del registro en uno.
Código:
```
MOV EAX,401020
SUB BYTE PTR DS:[EAX],51
INC EAX
```

Comparamos si se ha llegado hasta donde queremos encriptar (4075f4 si vamos a hacer "copy to executable", hasta el final de la sección si vamos a dumpear)
Código:
```
MOV EAX,401020
SUB BYTE PTR DS:[EAX],51
INC EAX
CMP EAX, XXXXX // ustedes ponen el valor dependiendo de lo que van a
hacer
```
Hacemos que si no es igual, siga encriptando.
Código:
```
MOV EAX,401020
SUB BYTE PTR DS:[EAX],51
INC EAX
CMP EAX, XXXXX // ustedes ponen el valor dependiendo de lo que van a
hacer
JNE @dirección donde esta el SUB
```
Y por ultimo hacemos que salte al entri point cuando termine de encriptar.
Código:
```
MOV EAX,401020
SUB BYTE PTR DS:[EAX],51
INC EAX
CMP EAX, XXXXX // ustedes ponen el valor dependiendo de lo que van a
hacer
JNE @dirección donde esta el SUB
JMP 4074ED
```
Ahora para que de verdad quede encriptado, vamos a hacer lo siguiente, nos paramos donde esta la instrucción MOV hacemos click Secundario y hacemos click en "New origin here" o lo que es lo mismo Ctrl. + *, Ponemos un breakPoint en el salto hacia el entri point y corremos con F9 al parar deberemos tomar la decisión, dumpeamos o no…

Si dumpean, hagan click secundario > "dump debugged process" y solo pongan como entripoint la dirección del MOV (Sin el IMAGEBASE) y guarden con el nombre que prefieran y listo.

Si no dumpean, seleccionen el código recién escrito, hagan click secundario > Copy to executable > selection, para volver al CPU (a ventana donde estábamos trabajando) hacemos click en el botón "C" de la barra de botones, luego vallan a 401000, hagan click normal en la primera instrucción para que se marque y vallan hacia el inicio del IMPORT DESCRIPTOR (4075f4) y presionando shift hacen click ahí donde aparecieron. Ahora de nuevo click secundario > Copy to executable > selection y ahí damos de nuevo click secundario > save file, Guardemos y con el Stud_pe cambiamos el entri point la dirección del MOV (sin el IMAGEBASE) y listo.

ahora despues de guardado lo abrimos de nuevo para que desencripte, es decir lo que tenemos que hacer es cambiar el sub por un add y hacer copy to executable, fin

Bifrost publico desempacado
Pass: indetect@bles
plugin analizethis
plugin ollyDump
Stud_pe
stub de la explicacion

De verdad disculpen lo tardado en sacar esta tercera clase, pero es que tuve problemas de tiempo y de olly… si he cometido errores les ruego que me los hagan saber, y si NO les gusto la clase me lo dicen también…

Un saludo a todos los que como yo son indetectables y a los que no..!

Un saludos a todos, espero que como yo hayan disfrutado esto y recuerden si van a copiar esto ponerle los creditos a **Bloodday**