INTRODUCCIÓN AL CRACKING EN OLLYDBG PARTE 8

Trataremos en esta parte de ver rápidamente algunas instrucciones importantes para el cracking que nos quedaron en el tintero para terminar con las mismas y empezar a crackear.

INTRUCCIONES PARA LOOPS O CICLOS

Ciertamente se pueden realizar ciclos en un programa con las instrucciones ya vistas o sea ejecutando varias instrucciones, poniendo un contador por ejemplo en ECX y al final una comparación de si es cero y un salto condicional que si no es cero vuelva a repetirse el ciclo se disminuya ECX y así se repetirá hasta que ECX sea cero seria algo así:

Xor ECX,ECX
Add ECX,15

Eso seria para inicializar el contador de nuestro loop que sera puesto a 15, aquí comienza el LOOP en si

DEC ECX

Para disminuir ECX cada vez que se ejecute el loop

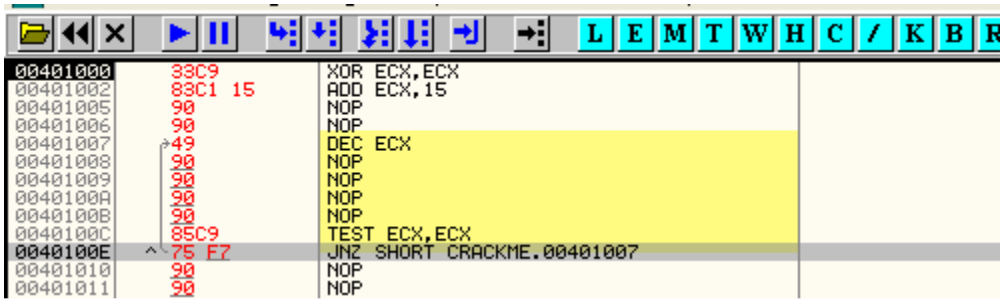
Luego la ejecución de las instrucciones que se deben repetir

Y luego

TEST ECX,ECX
JNE salta hacia el inicio del LOOP

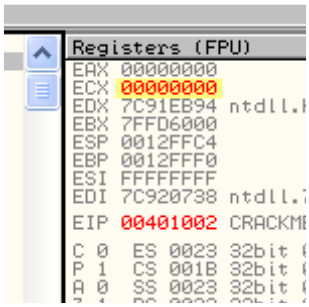
O sea que al testear si ECX es cero la primera vez será 14 ya que lo decremente una vez y al no ser cero volverá a repetirse, y así se repetirá hasta que ECX sea cero donde saldrá fuera del LOOP y continuará con la instrucción siguiente.

Escribámoslo en OLLYDBG

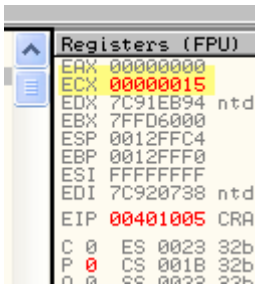


Allí lo vemos la zona resaltada en amarillo es el loop en si, que se repetirá hasta que ECX sea cero, y entre 401008 y 40100C se deberían escribir las instrucciones que el programa desea repetir que aquí no interesan pues solo vemos el mecanismo de iteración del LOOP por eso dejamos NOPS alli.

Si lo traceamos vemos que al apretar F7, ECX se pone a cero

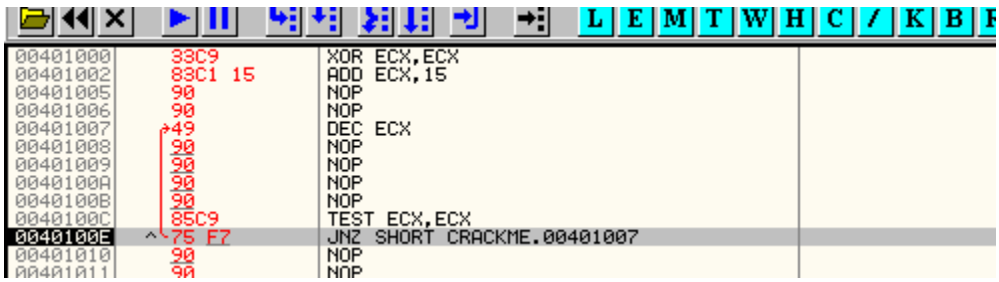


Apreto F7 nuevamente y le sumo 15 para establecer la cantidad de iteraciones en ECX.



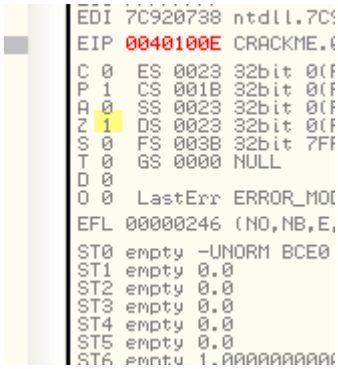
Luego apreto f7 hasta llegar al DEC ECX el cual al ejecutarlo pone ECX en 14

Luego llegamos hasta la comparación TEST ECX,ECX que sabemos que verifica si ECX es cero

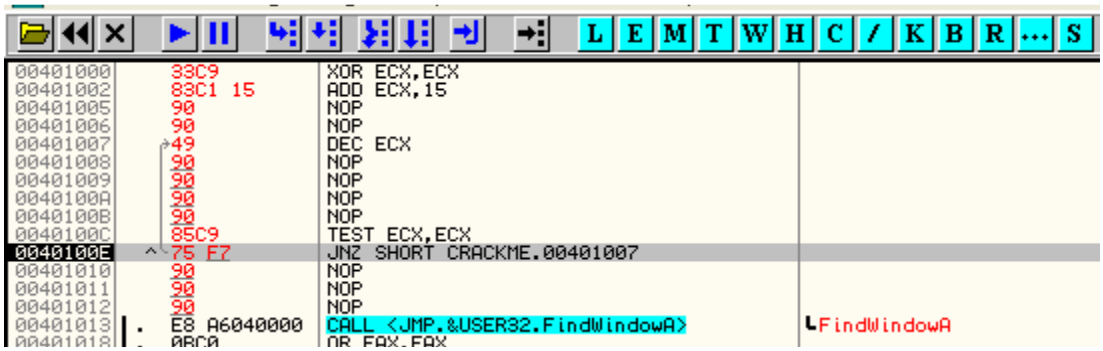


Al no ser cero no se activa el FLAG Z y el JNZ salta hacia 401007, donde vuelve a decrementar ECX quedando en 13, puedo tracear así haciendo los loops hasta llegar al momento que ECX vale CERO

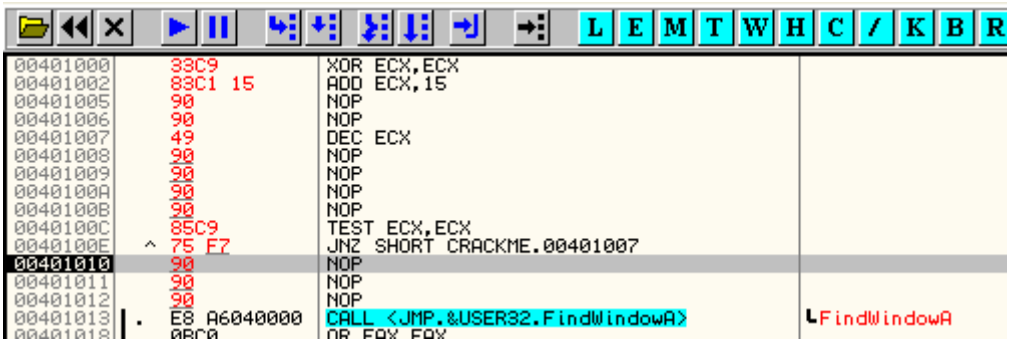
Vemos que al ejecutar la comparación con ECX igual a cero se activa el FLAG Z lo que hace que el JNZ no salte



Recordemos que JNZ es el inverso de JZ que saltaba cuando se activaba el FLAG Z, el JNZ es el opuesto no salta cuando se activa el FLAG Z.



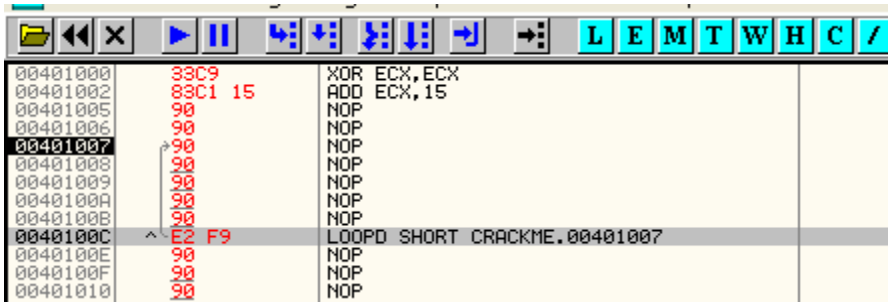
Allí lo vemos la flecha en gris indica que no va a saltar, al apretar F7 nuevamente sale del loop a la instrucción siguiente



Esa seria una forma sencilla de loop con las instrucciones ya vistas, aunque hay instrucciones especiales para eso.

LOOP

La instrucción LOOP nos ayuda a hacer algunas de las tareas que vimos en el ejemplo anterior, reemplazamos

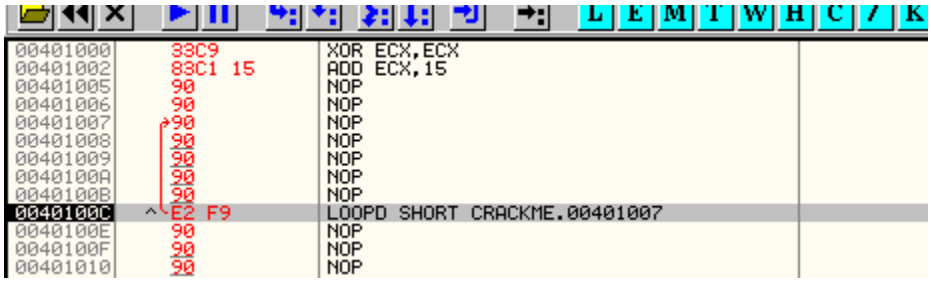


En donde estaba DEC ECX hacemos CLICK DERECHO – BINARY NOP con los cual NOPEARA esa instrucción, lo mismo donde estaban TEST ECX,ECX y JNZ 401007, todas esas instrucciones pueden ser reemplazadas por una sola que es la INSTRUCCIÓN LOOP la cual compara si ECX es cero, si no lo es salta al operando en este caso 401007 y además decrementa ECX .

Hagamos en la primera línea, CLICK DERECHO-NEW ORIGIN HERE para ejecutar mi nuevo LOOP.

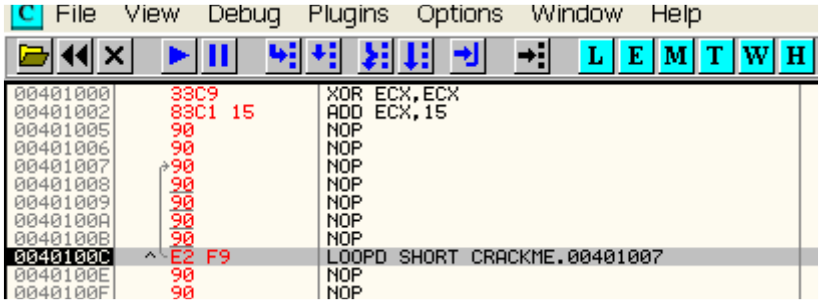
Vemos que al apretar f7 pone ECX a cero luego le suma 15 igual que antes para marcar la cantidad de iteraciones o repeticiones.

Ahora traceo y llego hasta la instrucción LOOP

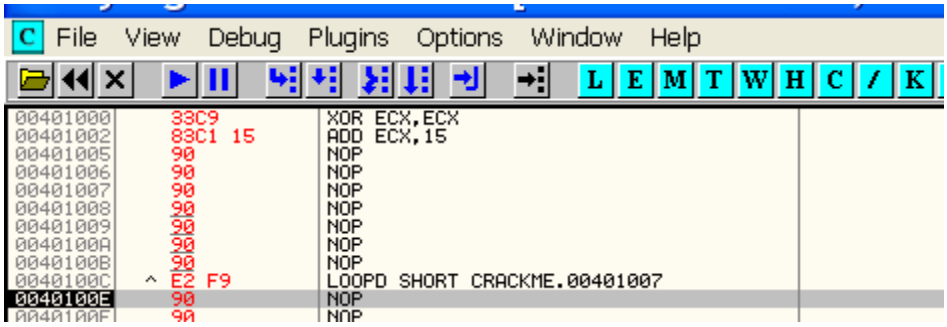


Vemos que igual que antes al no ser ECX igual a cero, salta a 401007, pero no solo compara si es cero y salta, también vemos que decrementa ECX ya que volvió siendo 14.

Si traceamos hasta que llega a LOOP pero valiendo ECX igual a cero veremos como se repite el ciclo.



En el momento que ECX vale cero ya no se repite mas el LOOP y al apretar F7 continua con la ejecución de la instrucción subsiguiente.



Luego tenemos variaciones de la instrucción LOOP estas son

- *LOOPZ,LOOPE* realizar un bucle *si es cero*
- *LOOPNZ,LOOPNE* realizar un bucle *si no es cero*

LOOPZ salta o mantiene dentro del bucle mientras que el flag Z sea cero cada vez que se ejecute la instrucción LOOP, y la opuesta LOOPNZ mientras que el FLAG Z sea 1, en este tipo de ciclo hay ademas contador que se decrementa y se sale por alguna comparación anterior que ponga el FLAG Z a cero en el primer caso o a 1 en el segundo o porque ECX llegue a cero por cualquiera de ambos casos.

INSTRUCCIONES PARA EL MANEJO DE CADENAS de BYTES

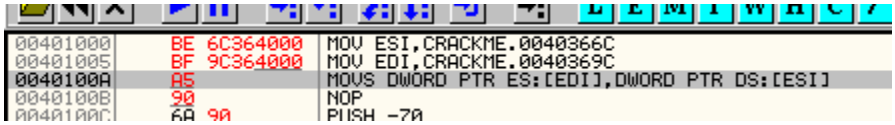
Aquí vemos las mas importantes las aclaramos debajo

MOVS

Esta instrucción, lo que hace es mover el contenido de ESI al contenido de EDI, realmente no necesita ningún parámetro, pero al escribir en OLLY la instrucción MOVS y apretar ASSEMBLE, la completa (innecesariamente) al ensamblar y queda como

MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]

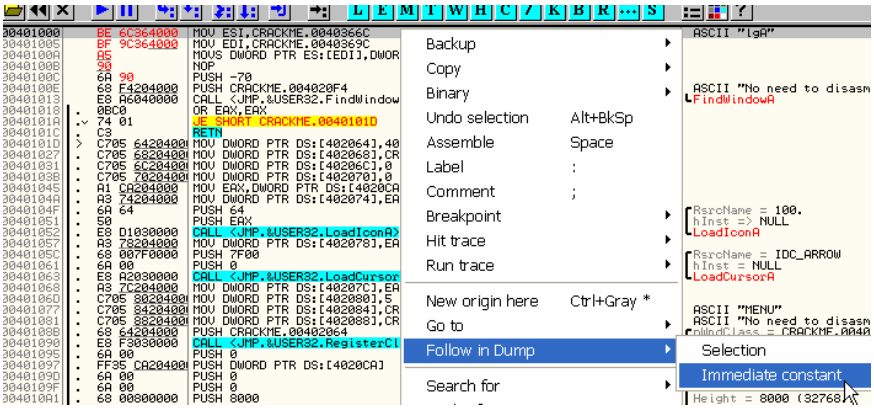
Allí vemos el ejemplo que escribí

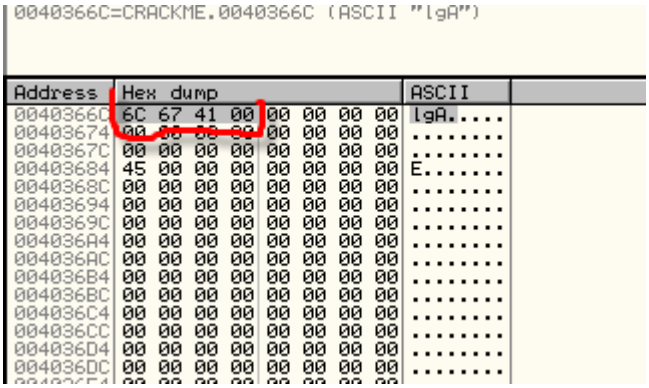


Inicializo antes ESI con la dirección de donde se va a leer o ORIGEN y EDI con el DESTINO o donde se copiará

Podemos mirar en el DUMP el contenido de ambos para ver como va a ser cuando lo ejecute

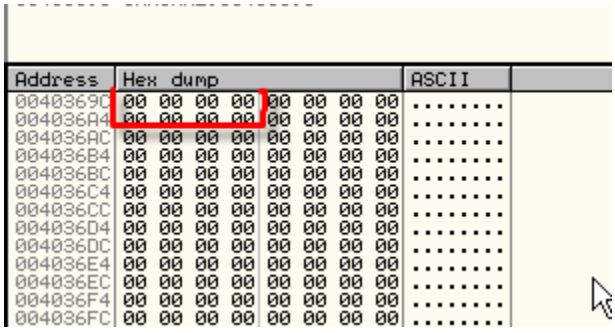
En el DUMP puedo hacer GOTO EXPRESIÓN=40366C o bien lo mismo y mas rápido FOLLOW IN DUMP- INMEDIATE CONSTANT que mostrara en el dump la constante 40366C





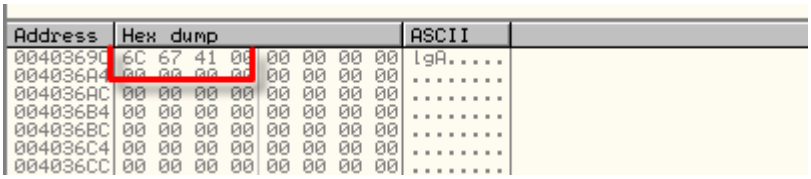
Allí vemos los bytes que apunta ESI o sea los de ORIGEN o que van a ser copiados.

Y EDI apunta a



Allí deberían copiarse seria el DESTINO.

Si apreto F7 hasta que ejecuto el MOVS vemos que se copiaron los 4 bytes.

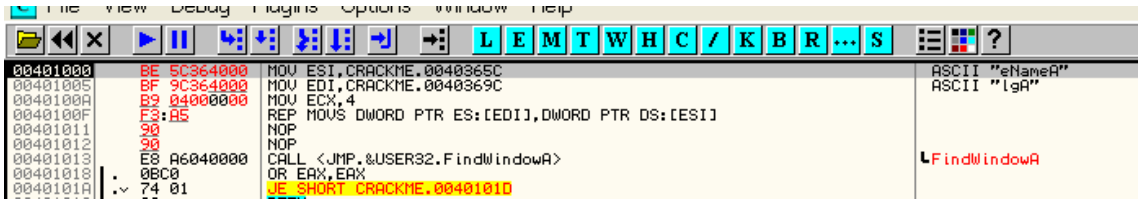


Así mismo como el comando MOVS mueve los 4 bytes o sea el DWORD, tambien existe MOVSW (mueve 2 bytes) y MOVSB (mueve un solo byte) en la misma forma que funciona MOVS.

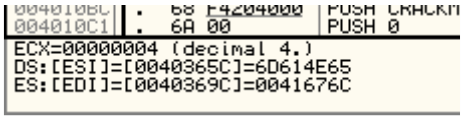
REP

Es un prefijo que se agrega a ciertas instrucciones como la anterior y que significa que la instrucción se repetirá hasta que ECX sea cero, a la vez que cada vez que se ejecute la instrucción se decrementara ECX en uno y aumentaran ESI Y EDI en 4 para apuntar a los siguientes 4 bytes.
Es muy util para copiar grandes cantidades de memoria de una zona del programa a otra.

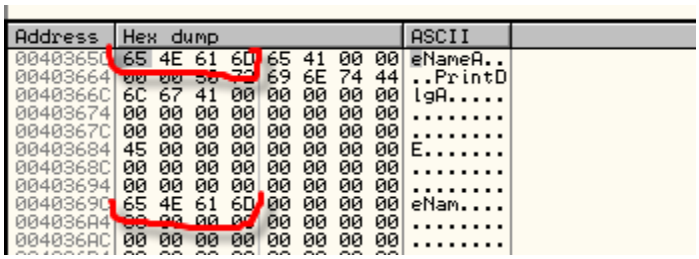
Modifiquemos el caso anterior y agreguémosle el REP



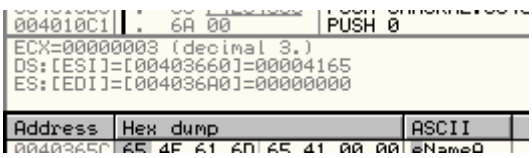
Además modifique la dirección de ORIGEN a 40365C para que copie desde allí, la dirección de destino sigue siendo 40369C, voy a la primera línea con NEW ORIGIN HERE y llego apretando F7 hasta el REP



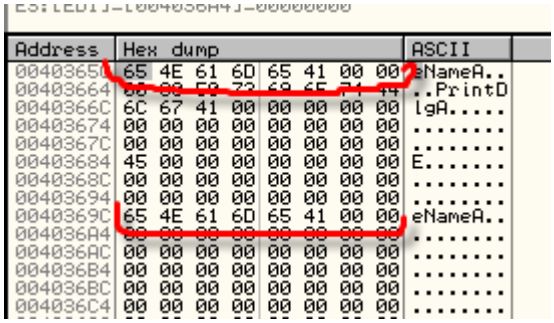
La aclaración del OLLY nos muestra las direcciones de ORIGEN Y DESTINO y los contenidos que copiara, apreto F7



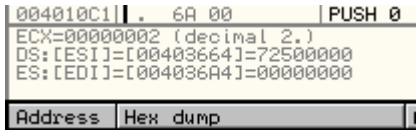
Allí se copiaron los 4 primeros bytes, pero no salimos de la instrucción ya que la misma se repetirá hasta que ECX sea cero y vemos que ahora ECX es 3, o sea disminuyo en uno, además ESI y EDI se incrementaron en 4, para apuntar a los 4 siguientes bytes.



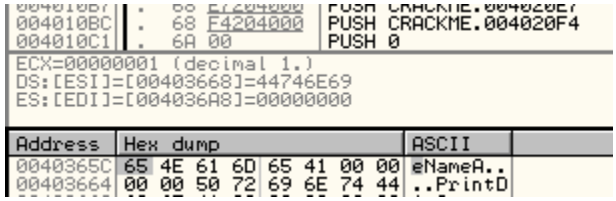
Apreto F7 nuevamente



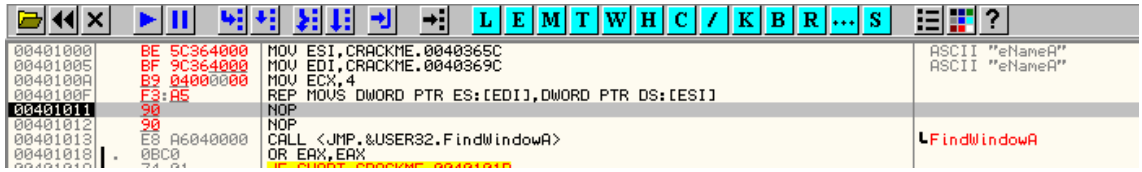
Allí se copiaron los siguientes 4 bytes y ECX quedo en 2



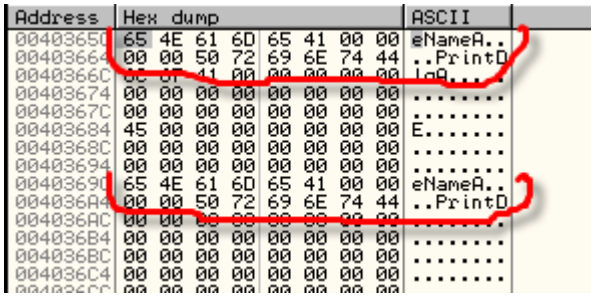
Si sigo apretando F7 una vez mas ECX será 1



Y nuevamente



Vemos que dejo de repetir y paso a la siguiente instrucción ya que ECX vale cero.



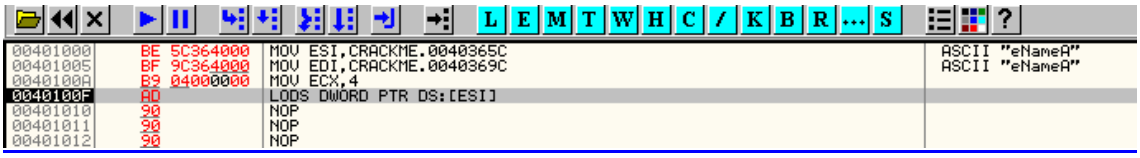
Vemos como se copiaron los bytes del ORIGEN al DESTINO y que se repitió 4 veces la operación gracias a la instrucción REP.

Es de aclarar que esta vez, busque una sección de destino con permiso de escritura ya que si no fuera así, como en veces anteriores al ejecutar la instrucción nos daría una excepción.

Además de la instrucción REP existen variantes como REPE o REPZ que repite hasta que el flag Z se pone a cero y REPNZ repite hasta que el flag Z no sea cero, o si ECX es cero también sale por cualquiera de las dos posibilidades, aunque estas variantes de REP no sirven para el caso de la instrucción MOVS si no para otras que veremos a continuación.

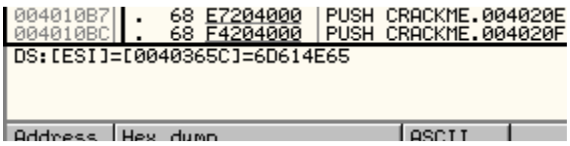
LODS

Esta instrucción lo que hace es mover los bytes que apunta ESI, o sea su contenido a EAX

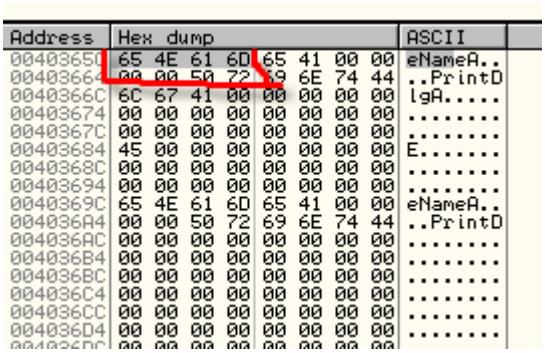


Vemos en este ejemplo que al llegar a LODS (que OLLY escribe como LODS DWORD PTR DS:[ESI])

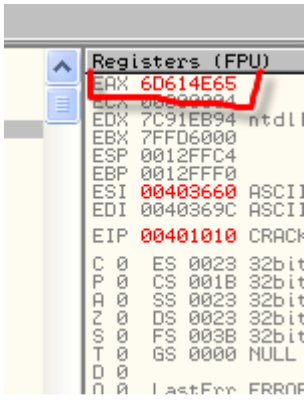
Y vemos que ESI apunta a 40365C lo cual vemos en la aclaración del OLLY



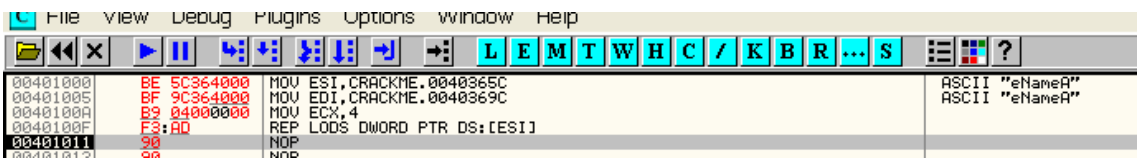
Y en el DUMP miramos



Esos son los 4 bytes que moverá a EAX, apreto F7.

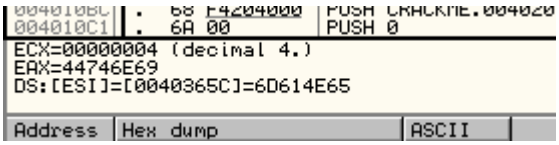


Al ejecutar con F7, vemos que EAX tomo ese valor.

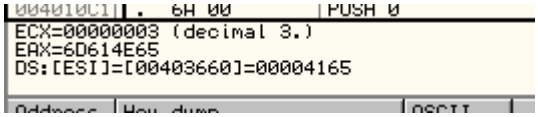


También a LODS se le puede agregar delante el REP y como en el caso anterior repetirá hasta que ECX sea cero, y leerá a partir de ESI los bytes, y los movera a EAX.

Al llegar a REP LODS



Nos muestra los bytes que apunta ESI y que serán transferidos a EAX si apreto F7



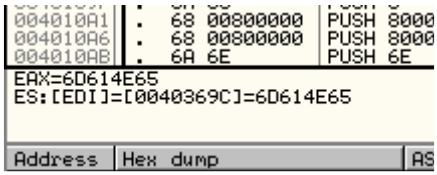
ECX se disminuyo a 3 y ESI se incremento 4 para apuntar a los siguientes 4 bytes que se moverán a EAX y así se repite hasta que ECX vale cero y sigue ejecutando la siguiente línea.

También existen las versiones para copiar 2 bytes LODSW y para copiar 1 byte LODSB

STOS

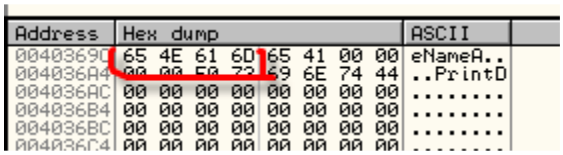
En este caso COPIA al contenido de EDI, el valor que hay en EAX.

Al ejecutar este ejemplo y llegar a STOS



Nos muestra EAX y el destino EDI en su contenido se copiaran.

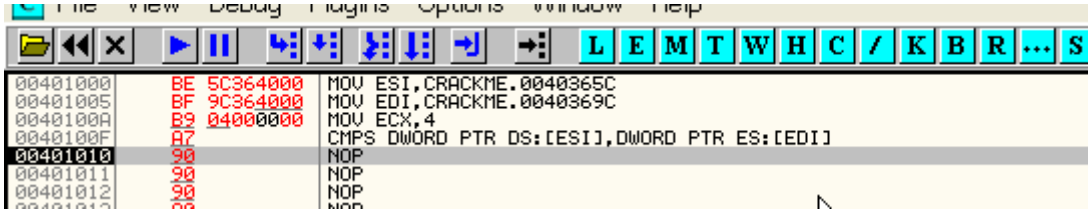
Al ejecutar



Allí se copiaron al DESTINO, también al igual que los casos anteriores se le puede agregar REP delante para repetir y existen STOSW y STOSB para copiar dos bytes o un solo byte.

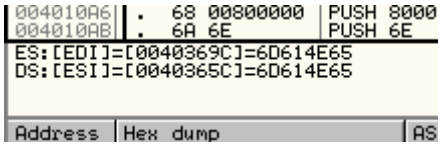
CMPS

Compara el contenido de ESI con el contenido de EDI

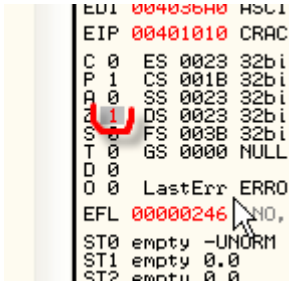


OLLY la escribe como CMPS DWORD PTR DS:[ESI],DWORD PTR ES:[EDI]

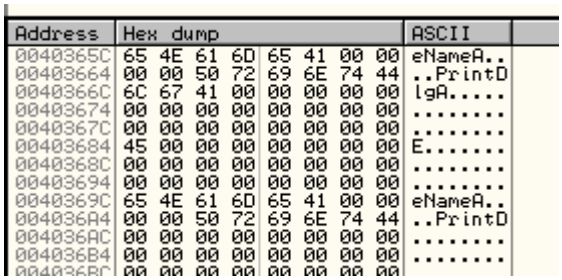
Si llego traceando con F7 hasta el CMPS la aclaración del OLLY nos muestra lo que va a comparar



Como la comparación en si sabemos que es una resta de ambos y como en mi caso son iguales el resultado es cero y activa el FLAGZ

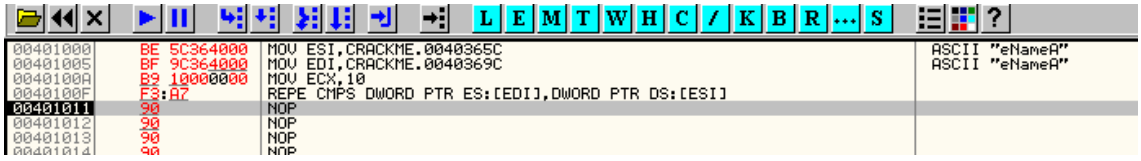


En este caso si se puede utilizar REPE o REPZ que comparara hasta que el FLAGZ sea cero o ECX sea cero en cualquiera de ambos estados saldrá del REPE.

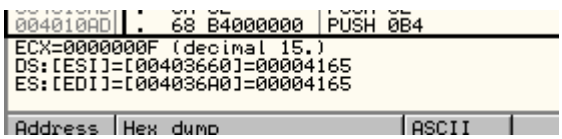


En mi caso, de anteriores ejemplos había quedado en 40365C y 40369c lo que vemos en mi DUMP.

Y escribo, poniendo ECX a 10 ya que en todos los casos al llegar ECX a cero teminará la repetición, solo que en el caso de REPE, también finalizara según el estado del FLAG Z.

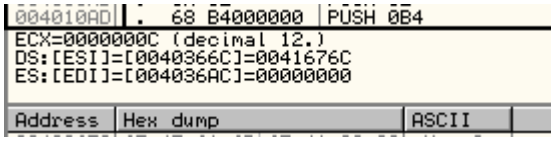


Llego hasta el REPE apretando F7 y en mi caso ambos operandos son iguales por lo cual la diferencia será cero y se activara el FLAG Z .

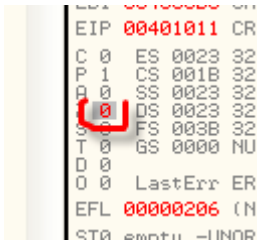


Vemos que para REPE si el FLAG Z esta a 1 o sea son iguales no nos saca de la repetición, apreto F7

Si seguimos apretando llega el momento en que ambos operandos son diferentes



En ese caso al apretar F7 el FLAG Z se pone a 0 y sale de la repetición



Si hubiéramos llegado a ECX igual a cero, siempre comparando operandos iguales, en ese caso saldría al ser ECX igual a cero también.

Como habíamos dicho existe también el REPNZ que salta al ser el flag Z igual a 1 o sea cuando en la comparación ambos operandos son iguales.

Creo que con esto hemos hecho un estudio bastante detallado de las instrucciones mas importantes, solo dejamos para mas adelante las instrucciones de PUNTO FLOTANTE para no complicar por ahora la cosa, espero que lo hayan entendido y que practiquen cada instrucción hasta que sepan bien que hace sin dudar.

MODOS DE DIRECCIONAMIENTO

DIRECTO:

Es el modo más comúnmente utilizado, para referirnos a una dirección de memoria, en la instrucción escribimos su valor.

```
mov dword ptr [00513450], ecx
mov ax, word ptr [00510A25]
mov al, byte ptr [00402811]
CALL 452200
JMP 421000
```

En este caso no tenemos ningún problema de interpretar cual es la dirección de memoria donde el programa guardara, moverá, saltara o ejecutara una rutina pues esta a la vista.

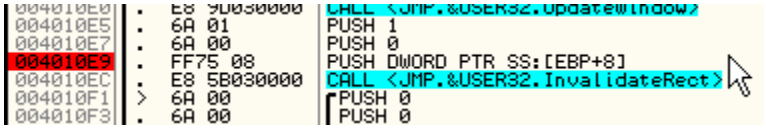
INDIRECTO:

```
mov dword ptr [eax], ecx
CALL EAX
JMP [ebx + 4]
```

Estas instrucciones si las vemos aquí no nos dicen en que dirección se guardaran, o donde saltara o estará la rutina del CALL, solo estando parado debuggeando con OLLYDBG justo en esa instrucción y viendo los valores que en ese momento tienen los registros, se podrá saber y OLLYDBG nos los mostrara en la aclaración cual es la dirección.

En muchos programa se utiliza el direccionamiento indirecto como una forma de complicar el trabajo del cracker ya que el análisis que OLLYDBG hace al inicio del programa, no nos dará información de estas instrucciones ya que hasta que no llegue a ejecutarlas no sabrá cual es el valor circunstancial de los registros.

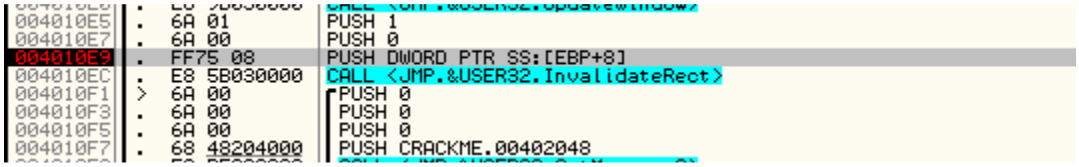
Solo poniendo un BREAKPOINT o llegando traceando hasta alguna de estas instrucciones, en dicho punto se sabrá el valor de la dirección, hagamos un ejemplo, reiniciemos el OLLYDBG con el crackme de cruehead



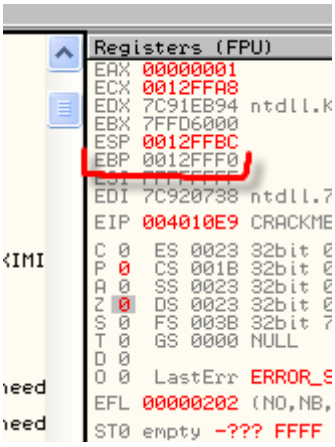
Allí vemos un PUSH [ebp+8]

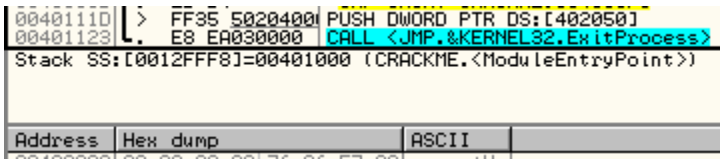
Como estoy en el inicio o ENTRY POINT del programa no se cuanto valdrá EBPcuando llegue a esa instrucción, por lo cual no tengo ni idea de que pusheara allí, voy a la línea con GOTO EXPRESIÓN 401009 y apreto F2 para colocar un BREAKPOINT.

Luego apreto F9 que es RUN y parara en el BREAKPOINT al pasar por alli.

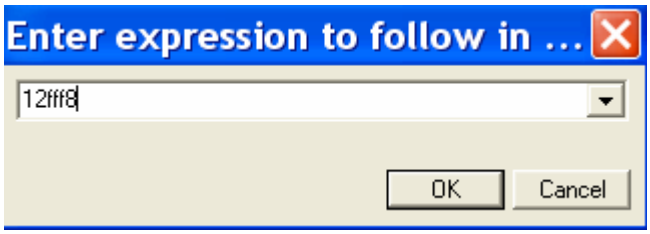


Allí paro y OLLY en la aclaración nos muestra que en mi maquina EBP+8 es 12FFF8 ya que EBP vale en este momento 12FFF0 al sumarle 8 dará 12FFF8 en mi maquina, en la suya puede tener otro valor pero será igual siempre a EBP+8.

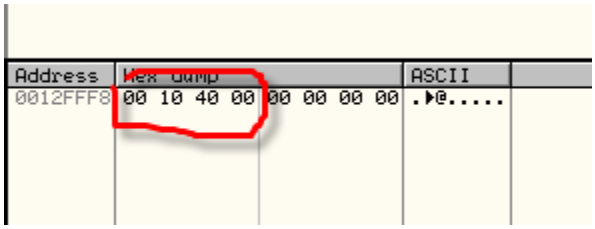




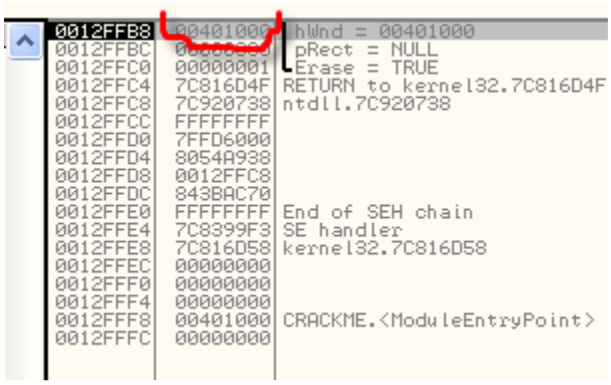
PUSH [ebp+8] es hacer PUSH el contenido de 12FFF8 si veo en el DUMP dicha posición de memoria con GOTO EXPRESIÓN 12FFF8



Y el contenido será



Por lo cual ese será el valor PUSHEADO si apreto F7 veo que lo coloca en el stack



El mismo caso se da en cualquier instrucción indirecta solo podemos calcular las direcciones que utilizara al estar justo ejecutando dicha instrucción.

Existen otros Modos de direccionamiento pero en cierta forma ya fueron explicados junto con las instrucciones por lo cual no repetiremos ni agregaremos mas nada.

Creo que el que sobrevivió a todo lo anterior y lo entendió bien tiene grandes probabilidades de éxito en el mundo del cracking no sin antes practicar y leer mucho pero, lo anterior es lo básico, a partir de la parte 9 ya ingresaremos en el apasionante mundo de la practica del cracking, por favor repasen bien las 8 partes estas a FULL.

Hasta la parte 9
Ricardo Narvaja
20 de noviembre de 2005