

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

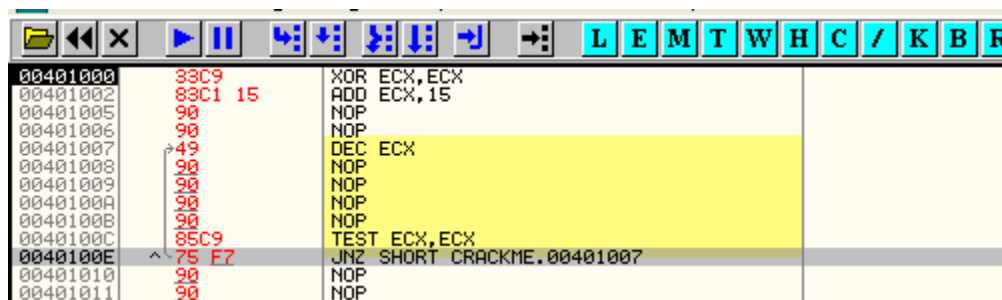
Luego la ejecución de las instrucciones que se deben repetir

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 allí.

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

```

Registers (FPU)
EAX 00000000
ECX 00000000
EDX 7C91EB94 ntdll.
EBX 7FFD6000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 ntdll.
EIP 00401002 CRACKME
C 0 ES 0023 32bit
P 1 CS 001B 32bit
A 0 SS 0023 32bit

```

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

```

Registers (FPU)
EAX 00000000
ECX 00000015
EDX 7C91EB94 ntd
EBX 7FFD6000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 ntd
EIP 00401005 CRA
C 0 ES 0023 32b
P 0 CS 001B 32b
A 0 SS 0023 32b

```

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

Address	Disassembly	Comment
00401000	XOR ECX,ECX	
00401002	ADD ECX,15	
00401005	NOP	
00401006	NOP	
00401007	DEC ECX	
00401008	NOP	
00401009	NOP	
0040100A	NOP	
0040100B	NOP	
0040100C	TEST ECX,ECX	
0040100E	JNZ SHORT CRACKME.00401007	
00401010	NOP	
00401011	NOP	

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

```

EDI 7C920738 ntdll.7C9
EIP 0040100E CRACKME.0
C 0 ES 0023 32bit 0000
P 1 CS 001B 32bit 0000
A 0 SS 0023 32bit 0000
Z 1 DS 0023 32bit 0000
S 0 FS 003B 32bit 7FF0
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR_MOM
EFL 00000246 (NO,NB,E,
ST0 empty -UNORM BCE0
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 1.000000000000

```

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.

Address	Hex	Assembly	Comment
00401000	33C9	XOR ECX,ECX	
00401002	83C1 15	ADD ECX,15	
00401005	90	NOP	
00401006	90	NOP	
00401007	49	DEC ECX	
00401008	90	NOP	
00401009	90	NOP	
0040100A	90	NOP	
0040100B	90	NOP	
0040100C	85C9	TEST ECX,ECX	
0040100E	75 F7	JNZ SHORT CRACKME.00401007	
00401010	90	NOP	
00401011	90	NOP	
00401012	90	NOP	
00401013	E8 A6040000	CALL <JMP.&USER32.FindWindowA>	FindWindowA
00401018	90	NOP	

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

Address	Hex	Assembly	Comment
00401000	33C9	XOR ECX,ECX	
00401002	83C1 15	ADD ECX,15	
00401005	90	NOP	
00401006	90	NOP	
00401007	49	DEC ECX	
00401008	90	NOP	
00401009	90	NOP	
0040100A	90	NOP	
0040100B	90	NOP	
0040100C	85C9	TEST ECX,ECX	
0040100E	75 F7	JNZ SHORT CRACKME.00401007	
00401010	90	NOP	
00401011	90	NOP	
00401012	90	NOP	
00401013	E8 A6040000	CALL <JMP.&USER32.FindWindowA>	FindWindowA
00401018	90	NOP	

Esa sería 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

Address	Hex	Assembly	Comment
00401000	33C9	XOR ECX,ECX	
00401002	83C1 15	ADD ECX,15	
00401005	90	NOP	
00401006	90	NOP	
00401007	90	NOP	
00401008	90	NOP	
00401009	90	NOP	
0040100A	90	NOP	
0040100B	90	NOP	
0040100C	E2 F9	LOOPD SHORT CRACKME.00401007	
0040100E	90	NOP	
0040100F	90	NOP	
00401010	90	NOP	

En donde estaba DEC ECX hacemos CLICK DERECHO – BINARY NOP con lo cual NOPEARÁ 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

00401000	33C9	XOR ECX,ECX
00401002	83C1 15	ADD ECX,15
00401005	90	NOP
00401006	90	NOP
00401007	90	NOP
00401008	90	NOP
00401009	90	NOP
0040100A	90	NOP
0040100B	90	NOP
0040100C	E2 F9	LOOPD SHORT CRACKME.00401007
0040100E	90	NOP
0040100F	90	NOP
00401010	90	NOP

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.

00401000	33C9	XOR ECX,ECX
00401002	83C1 15	ADD ECX,15
00401005	90	NOP
00401006	90	NOP
00401007	90	NOP
00401008	90	NOP
00401009	90	NOP
0040100A	90	NOP
0040100B	90	NOP
0040100C	E2 F9	LOOPD SHORT CRACKME.00401007
0040100E	90	NOP
0040100F	90	NOP

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.

00401000	33C9	XOR ECX,ECX
00401002	83C1 15	ADD ECX,15
00401005	90	NOP
00401006	90	NOP
00401007	90	NOP
00401008	90	NOP
00401009	90	NOP
0040100A	90	NOP
0040100B	90	NOP
0040100C	E2 F9	LOOPD SHORT CRACKME.00401007
0040100E	90	NOP
0040100F	90	NOP

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

00401000	BE 6C364000	MOV ESI,CRACKME.0040366C
00401005	BF 9C364000	MOV EDI,CRACKME.0040369C
0040100A	AS	MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]
0040100B	90	NOP
0040100C	EA 9A	PIUSH -7A

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- IMMEDIATE CONSTANT que mostrara en el dump la constante 40366C

Backup
Copy
Binary
Undo selection Alt+BkSp
Assemble Space
Label
Comment
Breakpoint
Hit trace
Run trace
New origin here Ctrl+Gray *
Go to
Follow in Dump
Search for

ASCII "lgA"
ASCII "No need to disasm
FindWindowA
RsroName = 100.
hInst => NULL
LoadIconA
RsroName = IDC_ARROW
hInst = NULL
LoadCursorA
ASCII "MENU"
ASCII "No need to disasm
FindWindowA = CRACKME.0040
Selection
Immediate constant
Height = 8000 (32768)

0040366C=CRACKME.0040366C (ASCII "lgA")			
Address	Hex dump	ASCII	
0040366C	6C 67 41 00 00 00 00 00	lgA....	
00403674	00 00 00 00 00 00 00 00	
0040367C	00 00 00 00 00 00 00 00	
00403684	45 00 00 00 00 00 00 00	E.....	
0040368C	00 00 00 00 00 00 00 00	
00403694	00 00 00 00 00 00 00 00	
0040369C	00 00 00 00 00 00 00 00	
004036A4	00 00 00 00 00 00 00 00	
004036AC	00 00 00 00 00 00 00 00	
004036B4	00 00 00 00 00 00 00 00	
004036BC	00 00 00 00 00 00 00 00	
004036C4	00 00 00 00 00 00 00 00	
004036CC	00 00 00 00 00 00 00 00	
004036D4	00 00 00 00 00 00 00 00	
004036DC	00 00 00 00 00 00 00 00	

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

Y EDI apunta a

Address	Hex dump	ASCII
0040369C	00 00 00 00 00 00 00 00
004036A4	00 00 00 00 00 00 00 00
004036AC	00 00 00 00 00 00 00 00
004036B4	00 00 00 00 00 00 00 00
004036BC	00 00 00 00 00 00 00 00
004036C4	00 00 00 00 00 00 00 00
004036CC	00 00 00 00 00 00 00 00
004036D4	00 00 00 00 00 00 00 00
004036DC	00 00 00 00 00 00 00 00
004036E4	00 00 00 00 00 00 00 00
004036EC	00 00 00 00 00 00 00 00
004036F4	00 00 00 00 00 00 00 00
004036FC	00 00 00 00 00 00 00 00

Allí deberían copiarse sería el DESTINO.

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

Address	Hex dump	ASCII
0040369C	6C 67 41 00 00 00 00 00	lgA.....
004036A0	00 00 00 00 00 00 00 00
004036AC	00 00 00 00 00 00 00 00
004036B4	00 00 00 00 00 00 00 00
004036BC	00 00 00 00 00 00 00 00
004036C4	00 00 00 00 00 00 00 00
004036CC	00 00 00 00 00 00 00 00

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

File View Debug Plugins Options Window Help		
L E M T W H C / K B R ... S		
00401000	BE 5C364000	MOV ESI,CRACKME.0040365C
00401005	BF 9C364000	MOV EDI,CRACKME.0040369C
0040100A	B9 04000000	MOV ECX,4
0040100F	F3:85	REP MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]
00401011	90	NOP
00401012	90	NOP
00401013	E8 A6040000	CALL <JMP.&USER32.FindWindowA>
00401018	0BC0	OR EAX,EAX
0040101A	74 01	JE SHORT CRACKME.0040101D

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 luego apretando F7 hasta el REP

004010B0	: 58 F4040000	PUSH CRACKME
004010C1	: 6A 00	PUSH 0
ECX=00000004 (decimal 4.)		
DS:[ESI]=0040365C=6D614E65		
ES:[EDI]=0040369C=0041676C		

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

Address	Hex dump	ASCII
0040365C	65 4E 61 6D 65 41 00 00	eNameA..
00403664	00 00 00 00 69 6E 74 44	..PrintD
0040366C	6C 67 41 00 00 00 00 00	lgA.....
00403674	00 00 00 00 00 00 00 00
0040367C	00 00 00 00 00 00 00 00
00403684	45 00 00 00 00 00 00 00	E.....
0040368C	00 00 00 00 00 00 00 00
00403694	00 00 00 00 00 00 00 00
0040369C	65 4E 61 6D 00 00 00 00	eNam....
004036A4	00 00 00 00 00 00 00 00
004036AC	00 00 00 00 00 00 00 00

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.

004010C1	: 6A 00	PUSH 0
ECX=00000003 (decimal 3.)		
DS:[ESI]=00403660=00004165		
ES:[EDI]=004036A0=00000000		
Address	Hex dump	ASCII
0040365C	65 4E 61 6D 65 41 00 00	eNameA..

Apreto F7 nuevamente

Address	Hex dump	ASCII
00403655	65 4E 61 6D 65 41 00 00	eNameA..
00403664	00 00 50 72 69 65 74 44	..PrintD
0040366C	6C 67 41 00 00 00 00 00	lgA.....
00403674	00 00 00 00 00 00 00 00
0040367C	00 00 00 00 00 00 00 00
00403684	45 00 00 00 00 00 00 00	E.....
0040368C	00 00 00 00 00 00 00 00
00403694	00 00 00 00 00 00 00 00
0040369C	65 4E 61 6D 65 41 00 00	eNameA..
004036A4	00 00 50 72 69 6E 74 44	..PrintD
004036AC	00 00 00 00 00 00 00 00
004036B4	00 00 00 00 00 00 00 00
004036BC	00 00 00 00 00 00 00 00
004036C4	00 00 00 00 00 00 00 00

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

004010C1	6A 00	PUSH 0
ECX=00000002 (decimal 2.)		
DS:[ESI]=[00403664]=72500000		
ES:[EDI]=[004036A4]=00000000		
Address	Hex dump	

Si sigo apretando F7 una vez mas ECX será 1

004010B7	68 F4204000	PUSH CRACKME.004020E7
004010BC	6A 00	PUSH CRACKME.004020F4
004010C1	6A 00	PUSH 0
ECX=00000001 (decimal 1.)		
DS:[ESI]=[00403668]=44746E69		
ES:[EDI]=[004036A8]=00000000		
Address	Hex dump	ASCII
0040365C	65 4E 61 6D 65 41 00 00	eNameA..
00403664	00 00 50 72 69 6E 74 44	..PrintD

Y nuevamente

00401000	BE 5C364000	MOV ESI,CRACKME.0040365C	ASCII "eNameA"
00401005	BF 9C364000	MOV EDI,CRACKME.0040369C	ASCII "eNameA"
0040100A	B9 04000000	MOV ECX,4	
0040100F	F3:05	REP MOVSD DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]	
00401011	90	NOP	
00401012	90	NOP	
00401013	E8 A6040000	CALL <JMP.<USER32.FindWindowA>	FindWindowA
00401018	0BC0	OR EAX,EAX	
0040101D	74 04	JE SHORT CRACKME.00401023	

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

Address	Hex dump	ASCII
00403655	65 4E 61 6D 65 41 00 00	eNameA..
00403664	00 00 50 72 69 6E 74 44	..PrintD
0040366C	6C 67 41 00 00 00 00 00	lgA.....
00403674	00 00 00 00 00 00 00 00
0040367C	00 00 00 00 00 00 00 00
00403684	45 00 00 00 00 00 00 00	E.....
0040368C	00 00 00 00 00 00 00 00
00403694	00 00 00 00 00 00 00 00
0040369C	65 4E 61 6D 65 41 00 00	eNameA..
004036A4	00 00 50 72 69 6E 74 44	..PrintD
004036AC	00 00 00 00 00 00 00 00
004036B4	00 00 00 00 00 00 00 00
004036BC	00 00 00 00 00 00 00 00
004036C4	00 00 00 00 00 00 00 00

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 MOVSD 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

00401000	BE 5C364000	MOV ESI, CRACKME.0040365C	ASCII "eNameA"
00401005	BF 9C364000	MOV EDI, CRACKME.0040369C	ASCII "eNameA"
0040100A	B3 04000000	MOV ECX, 4	
0040100F	FD	LODS DWORD PTR DS:[ESI]	
00401010	90	NOP	
00401011	90	NOP	
00401012	90	NOP	

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

004010B7	: 68 E7204000	PUSH CRACKME.004020E
004010BC	: 68 F4204000	PUSH CRACKME.004020F
DS:[ESI]=0040365C]=6D614E65		
Address	Hex dump	ASCII

Y en el DUMP miramos

Address	Hex dump	ASCII
00403651	65 4E 61 60 65 41 00 00	eNameA..
00403656	00 00 50 72 69 6E 74 44	..PrintD
0040365C	6C 67 41 00 00 00 00 00	lgA.....
00403674	00 00 00 00 00 00 00 00
0040367C	00 00 00 00 00 00 00 00
00403684	45 00 00 00 00 00 00 00	E.....
0040368C	00 00 00 00 00 00 00 00
00403694	00 00 00 00 00 00 00 00
0040369C	65 4E 61 60 65 41 00 00	eNameA..
004036A4	00 00 50 72 69 6E 74 44	..PrintD
004036AC	00 00 00 00 00 00 00 00
004036B4	00 00 00 00 00 00 00 00
004036BC	00 00 00 00 00 00 00 00
004036C4	00 00 00 00 00 00 00 00
004036CC	00 00 00 00 00 00 00 00
004036D4	00 00 00 00 00 00 00 00
004036DC	00 00 00 00 00 00 00 00

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

Registers (FPU)	
EAX	6D614E65
ECX	00000004
EDX	7C91EB94 ntdll
EBX	7FFD6000
ESP	0012FFC4
EBP	0012FFF0
ESI	00403660 ASCII
EDI	0040369C ASCII
EIP	00401010 CRACK
C 0	ES 0023 32bit
P 0	CS 001B 32bit
A 0	SS 0023 32bit
Z 0	DS 0023 32bit
S 0	FS 003B 32bit
T 0	GS 0000 NULL
O 0	
D 0	LastErr FRRNF

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

File view Debug Plugins Options window Help			
00401000	BE 5C364000	MOV ESI, CRACKME.0040365C	ASCII "eNameA"
00401005	BF 9C364000	MOV EDI, CRACKME.0040369C	ASCII "eNameA"
0040100A	B3 04000000	MOV ECX, 4	
0040100F	F3:BD	REP LODS DWORD PTR DS:[ESI]	
00401011	90	NOP	
00401012	90	NOP	

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

004010B0	: 68 14 4E 65	PUSH CRACKME.004020
004010C1	: 6A 00	PUSH 0
ECX=00000004 (decimal 4.)		
EAX=44746E69		
DS:[ESI]=[0040365C]=6D614E65		
Address	Hex dump	ASCII

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

004010C1	: 6A 00	PUSH 0
ECX=00000003 (decimal 3.)		
EAX=6D614E65		
DS:[ESI]=[00403660]=00004165		
Address	Hex dump	ASCII

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.

00401000	BE 5C364000	MOV ESI,CRACKME.0040365C
00401005	BF 9C364000	MOV EDI,CRACKME.0040369C
0040100A	B9 04000000	MOV ECX,4
0040100F	AB	STOS DWORD PTR ES:[EDI]
00401010	90	NOP
00401011	90	NOP
00401012	90	NOP

Al ejecutar este ejemplo y llegar a STOS

004010A1	: 68 00 00 00 00	PUSH 0000
004010A6	: 68 00 00 00 00	PUSH 0000
004010AB	: 6A 6E	PUSH 6E
EAX=6D614E65		
ES:[EDI]=[0040369C]=6D614E65		
Address	Hex dump	AS

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

Al ejecutar

Address	Hex dump	ASCII
00403690	65 4E 61 6D 65 41 00 00	eNameA..
004036A4	00 00 50 73 69 6E 74 44	..PrintD
004036AC	00 00 00 00 00 00 00 00
004036B4	00 00 00 00 00 00 00 00
004036BC	00 00 00 00 00 00 00 00
004036C4	00 00 00 00 00 00 00 00

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

```

00401000 BE 5C364000 MOV ESI,CRACKME.0040365C
00401005 BF 9C364000 MOV EDI,CRACKME.0040369C
0040100A B9 04000000 MOV ECX,4
0040100F A7 CMPS DWORD PTR DS:[ESI],DWORD PTR ES:[EDI]
00401010 90 NOP
00401011 90 NOP
00401012 90 NOP
00401013 90 NOP

```

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

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

```

004010A6 .: 68 00000000 PUSH 8000
004010AB .: 6A 6E PUSH 6E
ES:[EDI]=[0040369C]=6D614E65
DS:[ESI]=[0040365C]=6D614E65

```

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

```

EDI 0040369C ASCII
EIP 00401010 CRAC
C 0 ES 0023 32bi
P 1 CS 001B 32bi
A 0 SS 0023 32bi
Z 1 DS 0023 32bi
S 0 FS 003B 32bi
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR
EFL 00000246 NO,
ST0 empty -UNORM
ST1 empty 0.0
ST2 empty 0.0

```

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.

Address	Hex dump	ASCII
0040365C	65 4E 61 6D 65 41 00 00	eNameA..
00403664	00 00 50 72 69 6E 74 44	..PrintD
0040366C	6C 67 41 00 00 00 00 00	lgA.....
00403674	00 00 00 00 00 00 00 00
0040367C	00 00 00 00 00 00 00 00
00403684	45 00 00 00 00 00 00 00	E.....
0040368C	00 00 00 00 00 00 00 00
00403694	00 00 00 00 00 00 00 00
0040369C	65 4E 61 6D 65 41 00 00	eNameA..
004036A4	00 00 50 72 69 6E 74 44	..PrintD
004036AC	00 00 00 00 00 00 00 00
004036B4	00 00 00 00 00 00 00 00
004036BC	00 00 00 00 00 00 00 00

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.

```

00401000 BE 5C364000 MOV ESI,CRACKME.0040365C
00401005 BF 9C364000 MOV EDI,CRACKME.0040369C
0040100A B9 10000000 MOV ECX,10
0040100F A7 REPE CMPS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]
00401010 90 NOP
00401011 90 NOP
00401012 90 NOP
00401013 90 NOP
00401014 90 NOP

```

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.

```

004010AD .: 68 B4000000 PUSH 0B4
ECX=0000000F (decimal 15.)
DS:[ESI]=[00403660]=00004165
ES:[EDI]=[004036A0]=00004165

```

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

004010AD	68 B4000000	PUSH 0B4
ECX=0000000C (decimal 12.)		
DS:[ESI]=[0040366C]=0041676C		
ES:[EDI]=[004036AC]=00000000		
Address	Hex dump	ASCII

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

EIP	00401011	CR
C 0	ES 0023	32
P 1	CS 001B	32
A 0	SS 0023	32
S 0	DS 0023	32
T 0	FS 003B	32
D 0	GS 0000	NU
O 0	LastErr	ER
EFL	00000206	(N
CTL	empty	-HNR

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 guardarán, 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 programas 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

```

004010E0 . E8 50030000 CALL <JMP.&USER32.UpdateWindow>
004010E5 . 6A 01      PUSH 1
004010E7 . 6A 00      PUSH 0
004010E9 . FF75 08    PUSH DWORD PTR SS:[EBP+8]
004010EC . E8 5B030000 CALL <JMP.&USER32.InvalidRect>
004010F1 . 6A 00      PUSH 0
004010F3 . 6A 00      PUSH 0

```

Allí vemos un PUSH [ebp+8]

Como estoy en el inicio o ENTRY POINT del programa no se cuanto valdrá EBP cuando 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.

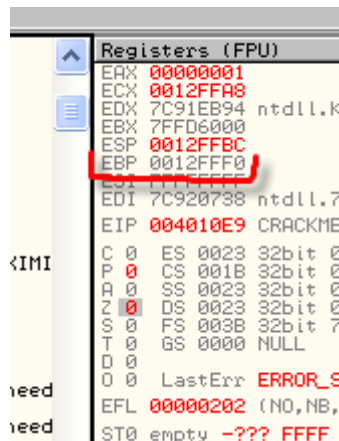
Luego apreto F9 que es RUN y parara en el BREAKPOINT al pasar por allí.

```

004010E0 . E8 50030000 CALL <JMP.&USER32.UpdateWindow>
004010E5 . 6A 01      PUSH 1
004010E7 . 6A 00      PUSH 0
004010E9 . FF75 08    PUSH DWORD PTR SS:[EBP+8]
004010EC . E8 5B030000 CALL <JMP.&USER32.InvalidRect>
004010F1 . 6A 00      PUSH 0
004010F3 . 6A 00      PUSH 0
004010F5 . 6A 00      PUSH 0
004010F7 . 68 48204000 PUSH CRACKME.00402048

```

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.



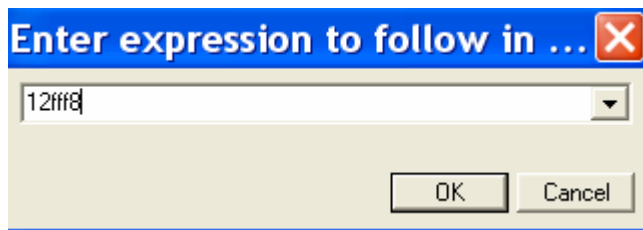
```

00401110 . FF35 50204000 PUSH DWORD PTR DS:[402050]
00401123 . E8 EA030000 CALL <JMP.&KERNEL32.ExitProcess>
Stack SS:[0012FFF8]=00401000 (CRACKME.<ModuleEntryPoint>)

```

Address	Hex dump	ASCII
00401110	FF 35 50 20 40 00	
00401123	E8 EA 03 00 00	

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

Address	Hex dump	ASCII
0012FFF8	00 10 40 00 00 00 00 00	..@.....

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

0012FFB8	00401000	hWnd = 00401000
0012FFBC	00000000	pRect = NULL
0012FFC0	00000001	Erase = TRUE
0012FFC4	7C816D4F	RETURN to kernel32.7C816D4F
0012FFC8	7C920738	ntdll.7C920738
0012FFCC	FFFFFFFF	
0012FFD0	7FFD6000	
0012FFD4	8054A938	
0012FFD8	0012FFC8	
0012FFDC	843BAC70	
0012FFE0	FFFFFFFF	End of SEH chain
0012FFE4	7C8399F3	SE handler
0012FFE8	7C816D58	kernel32.7C816D58
0012FFEC	00000000	
0012FFF0	00000000	
0012FFF4	00000000	
0012FFF8	00401000	CRACKME.<ModuleEntryPoint>
0012FFFC	00000000	

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