EXPLOITING Y REVERSING

USANDO HERRAMIENTAS

GRATUITAS (PARTE 18)

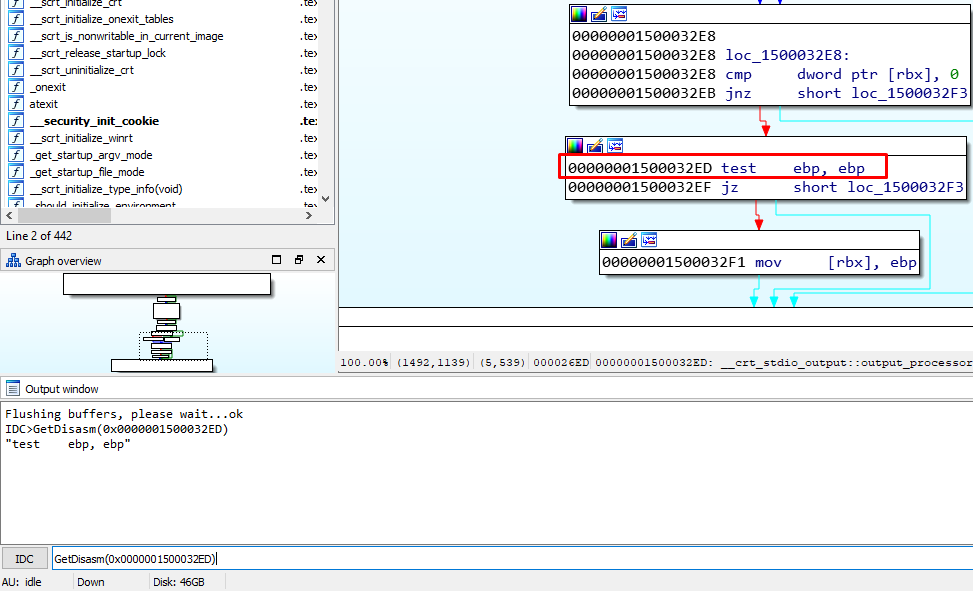
Bueno para hacer una pausa entre tanto reversing y antes de seguir con temas y ejercicios más profundos, veremos el tema del scripting en este caso, como hacer scripts para IDA FREE.

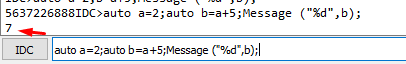
Todos sabemos que IDA PRO incluye IDA PYTHON para poder realizar scripts en IDA, nosotros no tenemos esa posibilidad, pero aprovechando lo que tenemos, que es bastante, IDA FREE trae soporte para hacer scripts con IDC que es una especie de C, pero que sirve para poder realizar scripts para IDA FREE y PRO.

IDC SCRIPTING

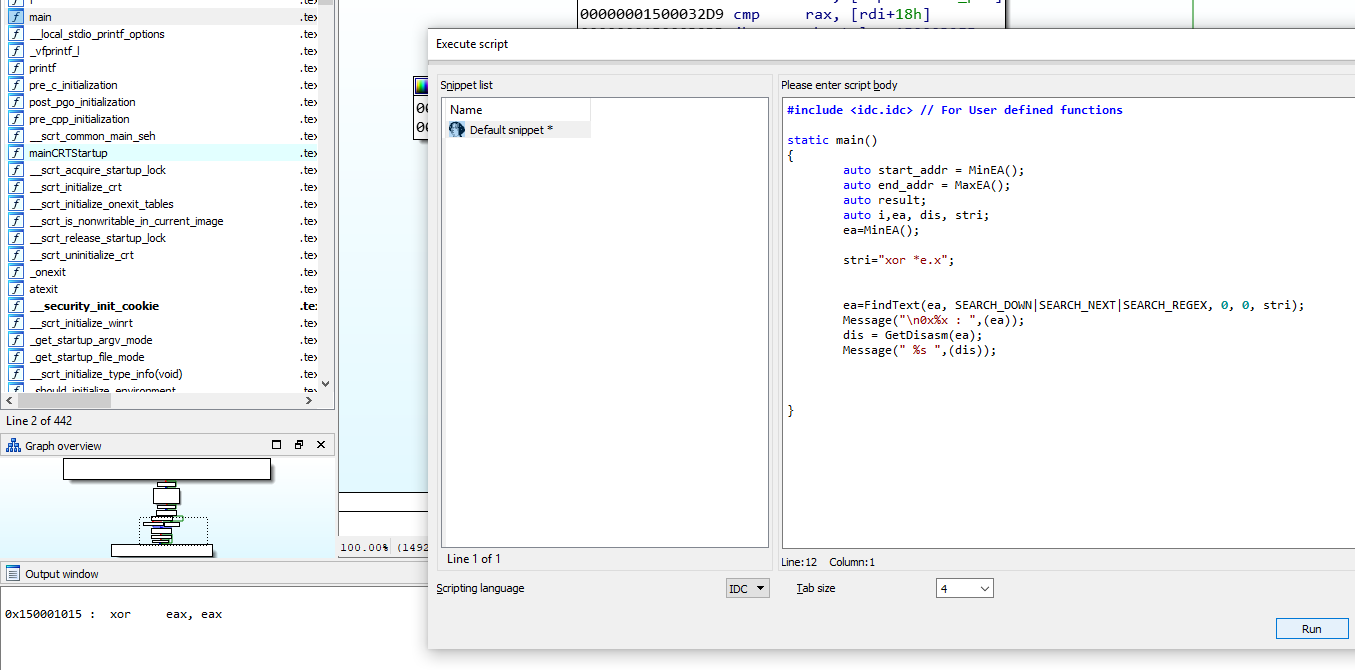
Podemos ejecutar scripts que escribimos en archivos idc, con la opcion del menu FILE-SCRIPT FILE.

También tenemos la barra de IDC debajo, donde se pueden ejecutar comandos sueltos, o varios comandos separados con punto y coma.

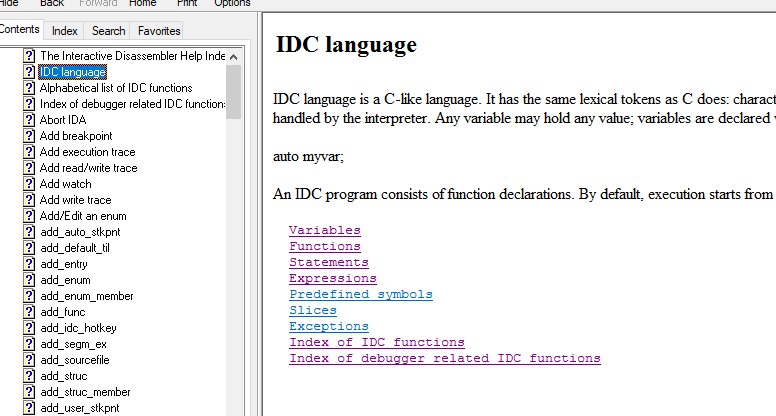


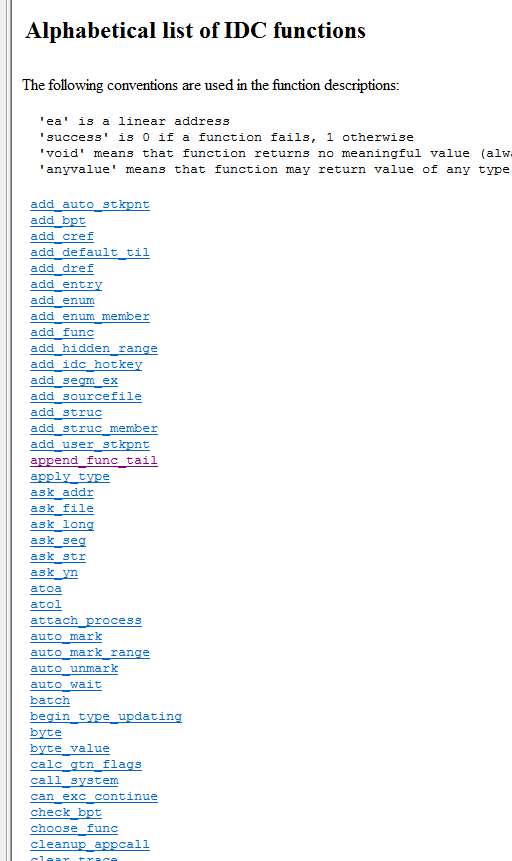


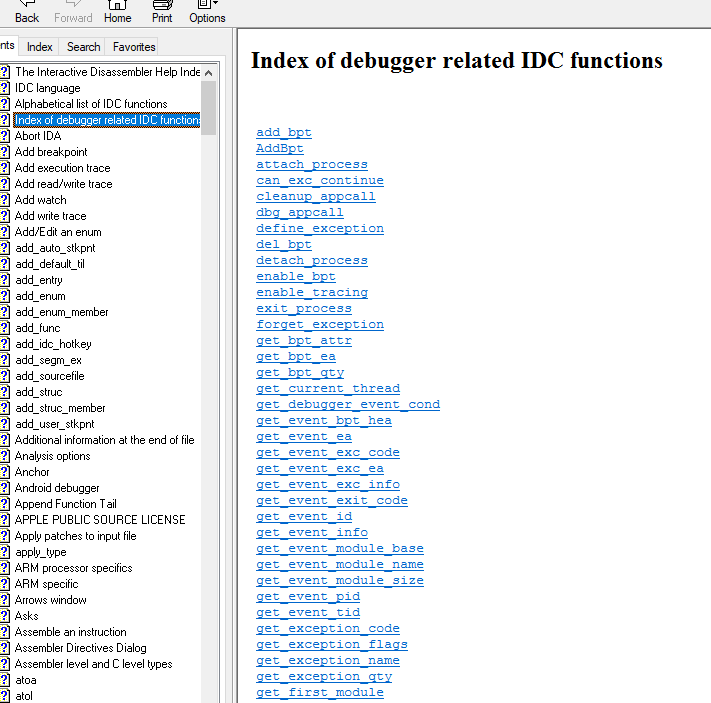
Obviamente no es muy cómoda la barra, pero tenemos una tercera opción para probar pequeños trozos de código.



Bueno comencemos, una lista de funciones que se pueden utilizar en IDC está en el HELP del IDA.



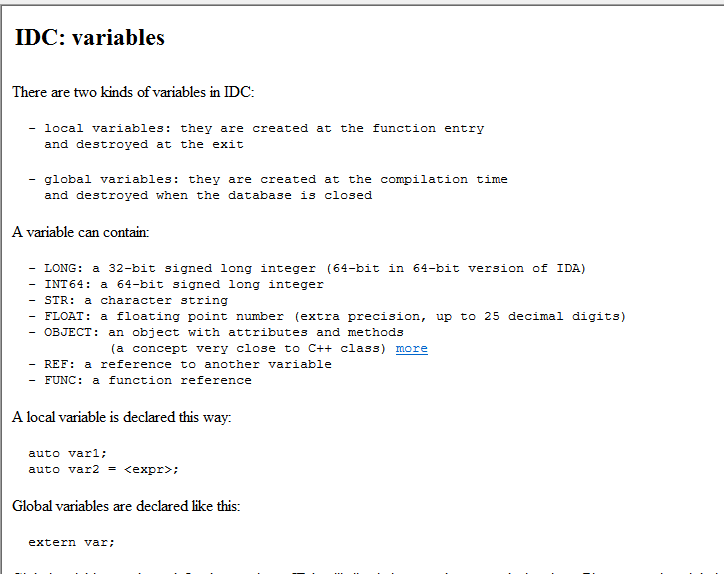




Vemos que tiene muchísimas funciones, algunas para usar al debuggear, otras se pueden usar sin debuggear, tiene muchas posibilidades.

Bueno vayamos viendo como es este “casi C” que es este lenguaje IDC, jeje.

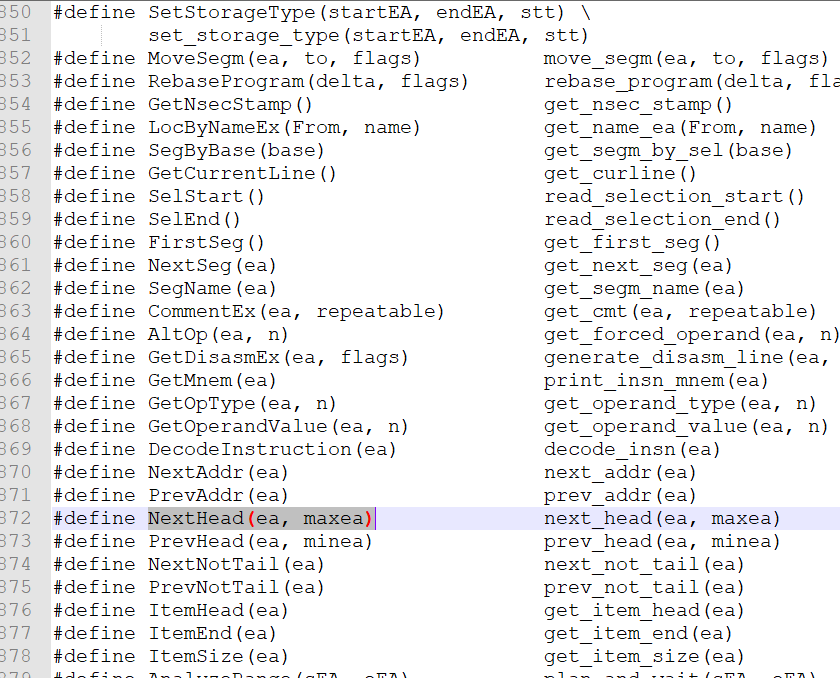
Una de las principales diferencias es que no es un lenguaje donde haya que declarar tipos, aquí el tipo genérico es “auto” y sirve para todos los casos de variables locales del script.



Vemos lo que puede contener cada variable, las variables globales que se declaran con “extern” y no mucho más obviamente es un lenguaje muy sencillo sólo para scripting.

Además de lo que dice la documentación conviene tener abierto en un editor de texto el archivo idc.idc que se encuentra en:

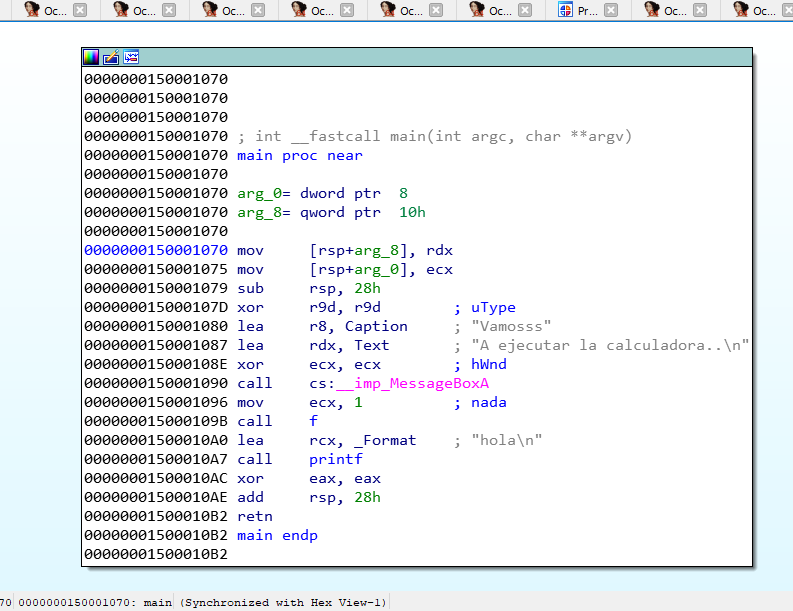
C:\Program Files\IDA Freeware 7.0\idc

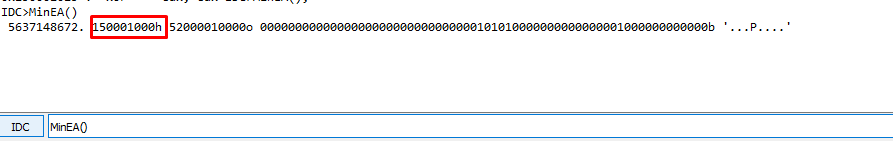


Tiene funciones que no están en el listado de idc que está en el help y que ayudan mucho.

Vamos a ir conociendo y ejecutando algunas funciones, para irnos adentrando en IDC.

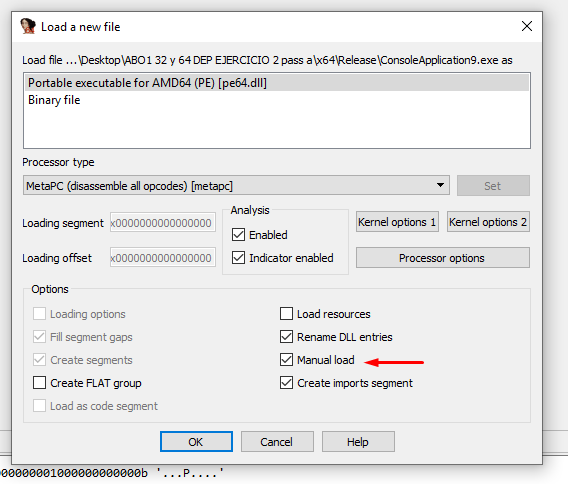
Para probar y trabajar todos con el mismo ejecutable, usaremos el del último ejercicio de 64 bits.

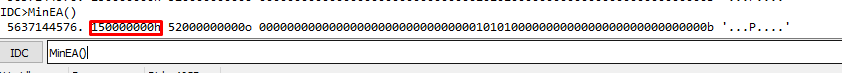




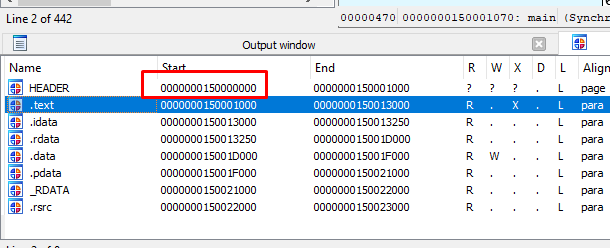
Vemos que la funcion MinEa() nos devuelve el el inicio de la primera sección que tenemos cargada, en este caso la sección de código, ya que no se carga el header por default.

Si recargo el ejecutable con MANUAL LOAD, sobrescribiendo la database anterior, y aceptando que cargue todas las secciones inclusive el header, en ese caso MinEa() me devolvería la imagebase.





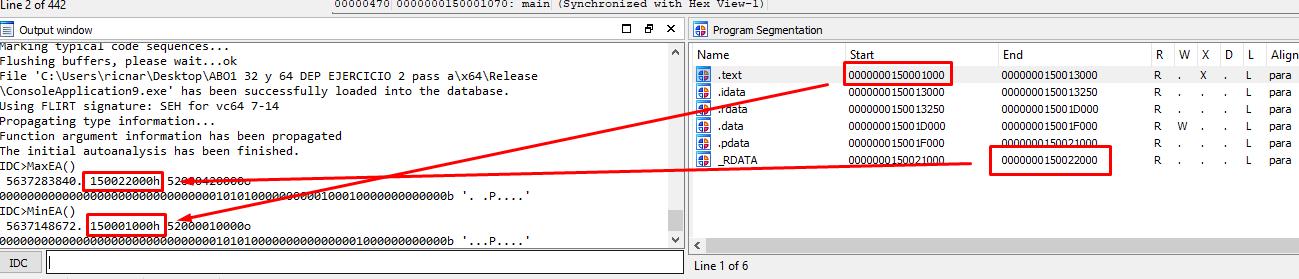
Con VIEW-OPEN SUBVIEWS-SEGMENTS podemos ver las secciones cargadas.



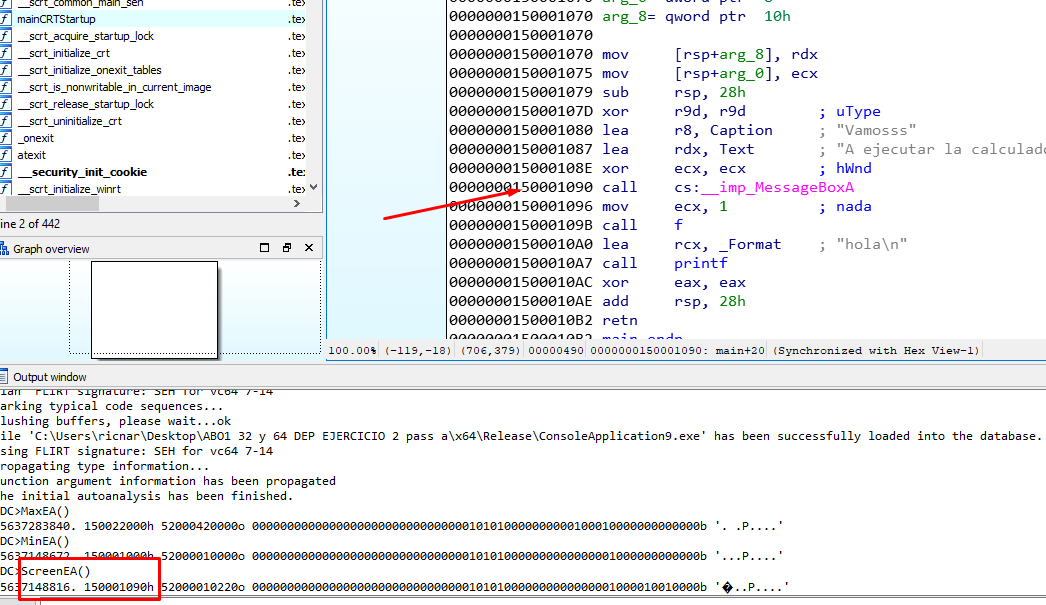
Por supuesto con MaxEA() veremos el final de la última sección cargada.



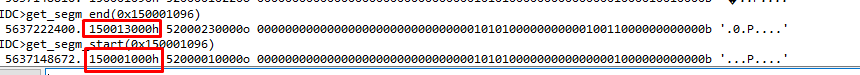
Lo volvere a cargar por default sin el header.



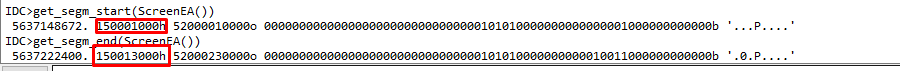
Vemos que siempre muestra el inicio de la sección más baja cargada y el final de la última sección más alta cargada.



ScreenEa() nos muestra la dirección donde está el cursor.



Estas son parecidas pero les debemos pasar una dirección y nos devuelven el inicio y final de la sección a la cual corresponden, se puede concatenar con averiguar el inicio y final de la sección donde está el cursor.

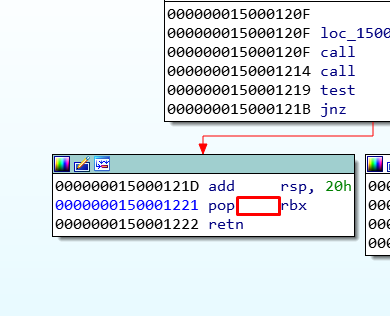


SCRIPT PARA BUSCAR GADGETS

Bueno vamos a ir poco a poco haciendo un script casero que busque algunos gadgets o instrucciones en la sección de código.

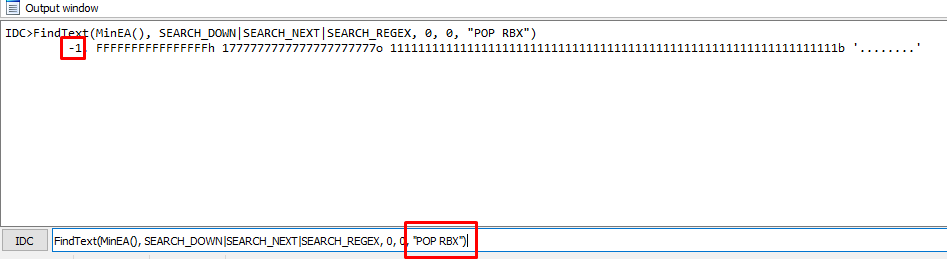
Por supuesto como buscamos texto, solo hallaremos los resultados de instrucciones alineadas y detectadas como instrucciones por IDA. No podremos encontrar instrucciones desalineadas ni en secciones que no sean código, pero para practicar y para algún caso rápido y de pocas instrucciones servirá.

Para ellos usaremos FindText() que puede buscar instrucciones usando REGEX.

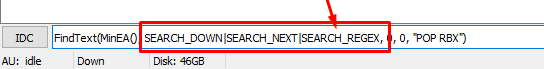


Como buscamos texto si buscáramos por ejemplo “POP RBX” no encontraría el resultado que muestro arriba porque tiene más de un espacio y debería escribirlo exactamente igual con la misma cantidad de espacios.

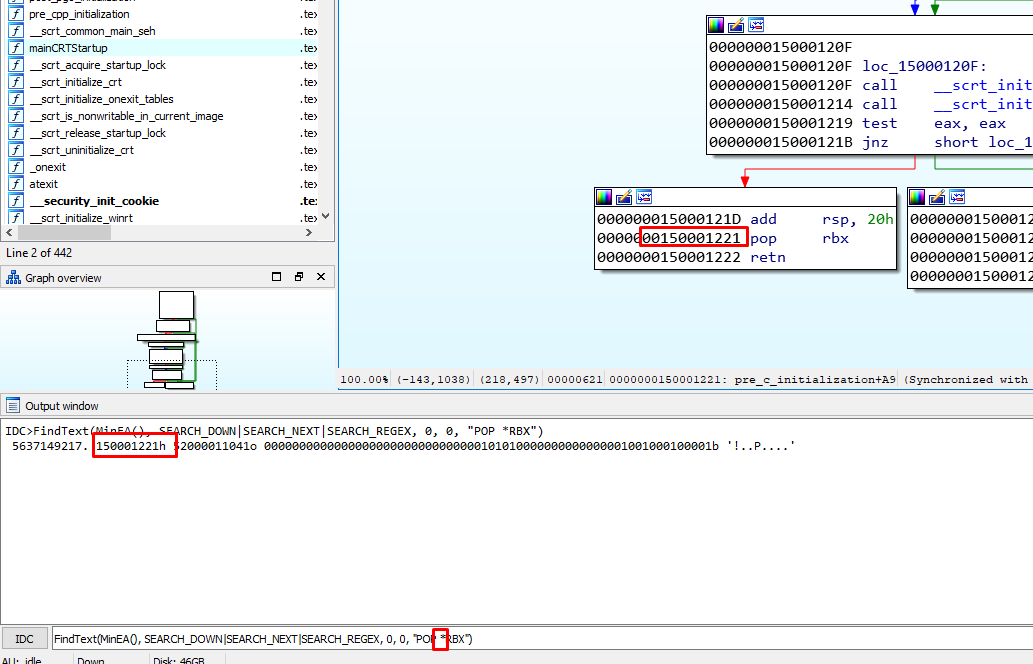
Veo que no lo encuentra.



En cambio como le agregue la opción entre los flags que busque usando REGEX.

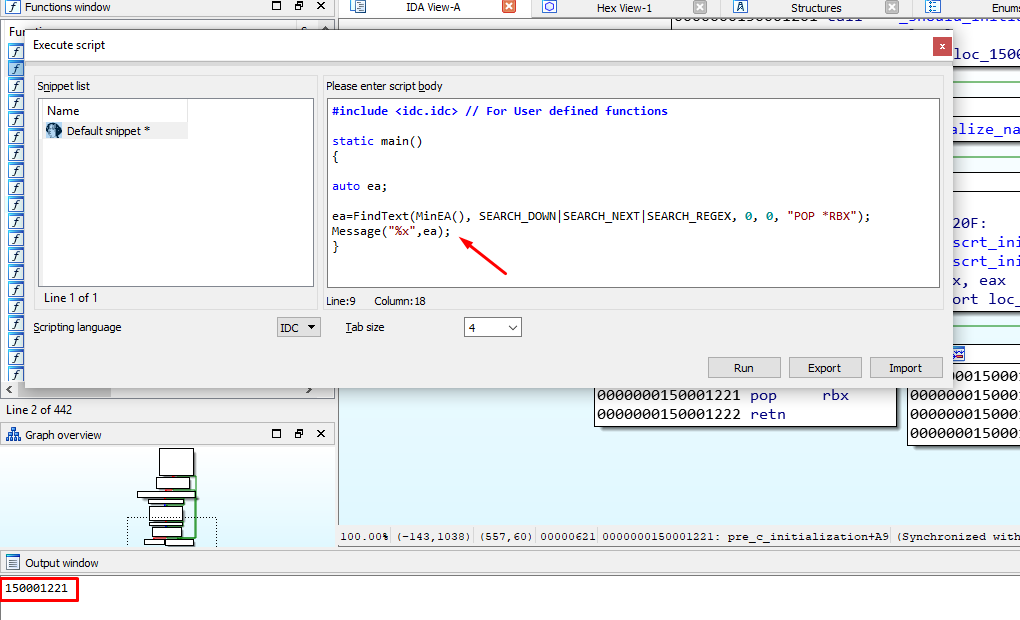


Entonces puedo usar el asterisco que me permite que haya coincidencia, habiendo uno o más del caracter que está justo antes del asterisco.



Esa búsqueda hallará **POP EBX, POP EBX, POP EBX** y coincidirá sin importar la cantidad de espacios entre POP y EBX que haya.

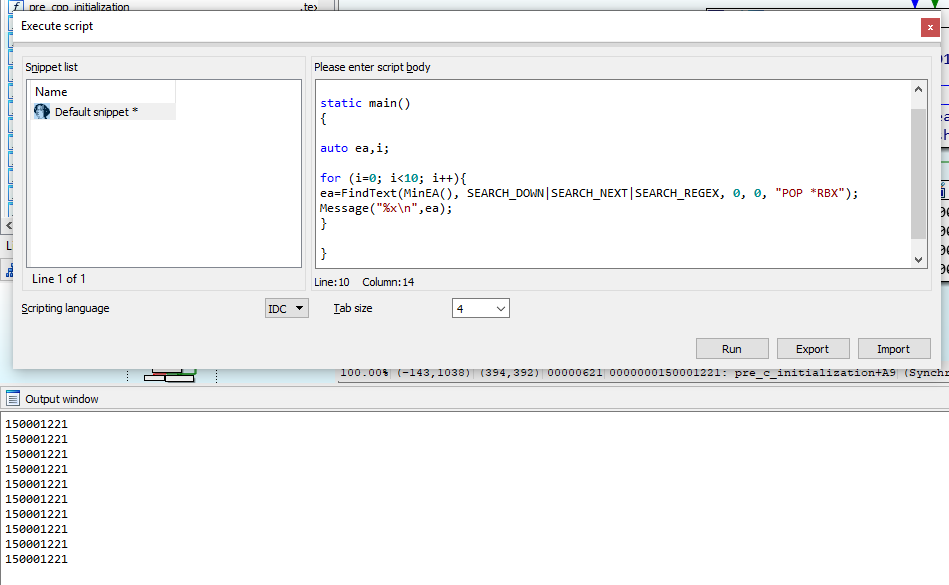
Como haria para encontrar hasta 10 instrucciones que quiera, lo haré en la ventanita.



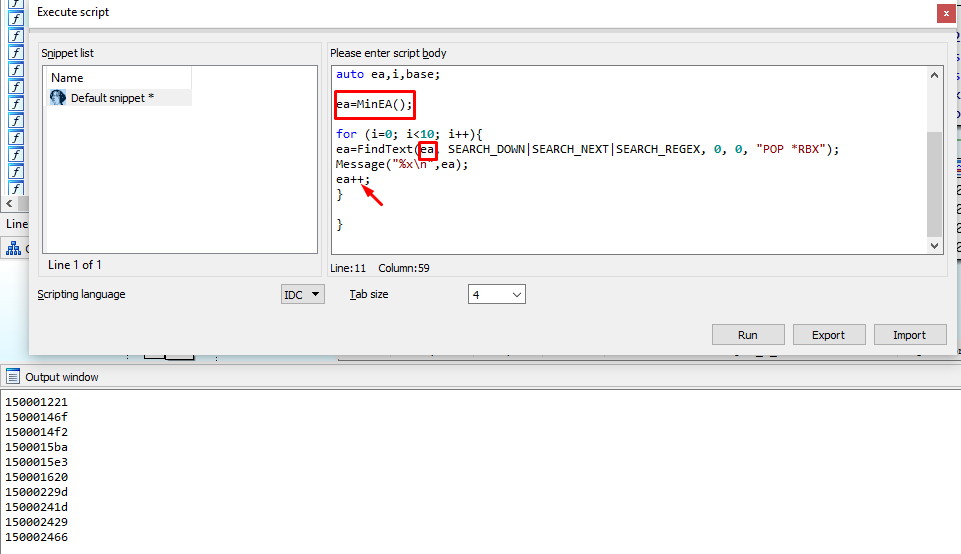
Con eso encuentro una sola dirección, la imprimo con la funcion Message que es similar a un printf.

Veo que declare la variable local ea como auto y no me olvide de los puntos y coma al final de cada instrucción.

Ahora quiero imprimir 10 hare un loop.



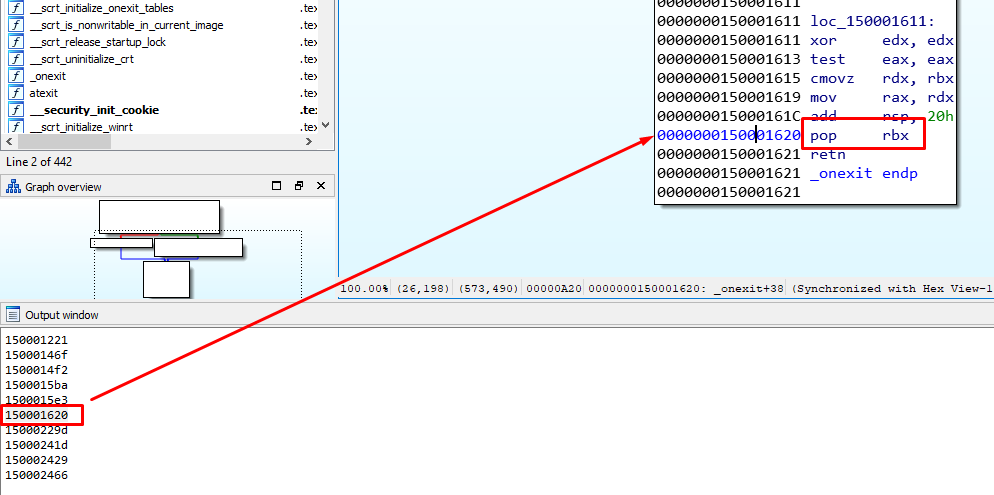
Así siempre encuentra la misma debo aumentar la dirección de búsqueda después de que encuentra alguna, para que busque a continuación.



Por supuesto hay que sacar el MInEA() del loop sino siempre buscara desde alli y usar una variable que se vaya incrementando, uso ea, y después de cada find la incremento en uno para que busque desde el byte siguiente y no repita buscando desde el mismo lugar.

Veo que encontre 10 instrucciones, miro alguna de las otras.

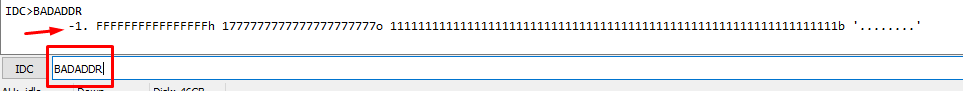
Bueno vamos bien.



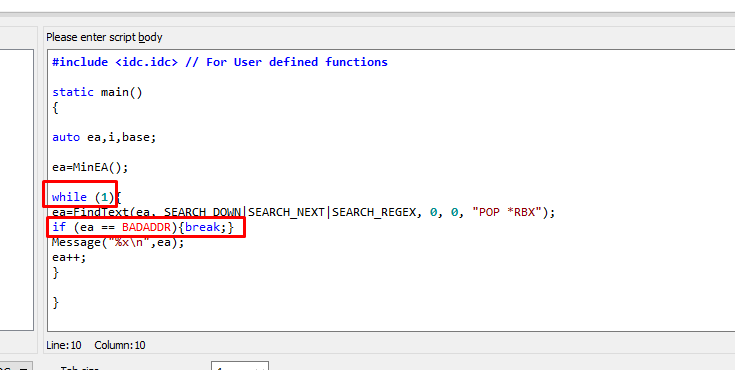
Lo siguiente es imprimir la instrucción que halló, pero antes le agregare este chequeo:

if (ea == BADADDR){break;}

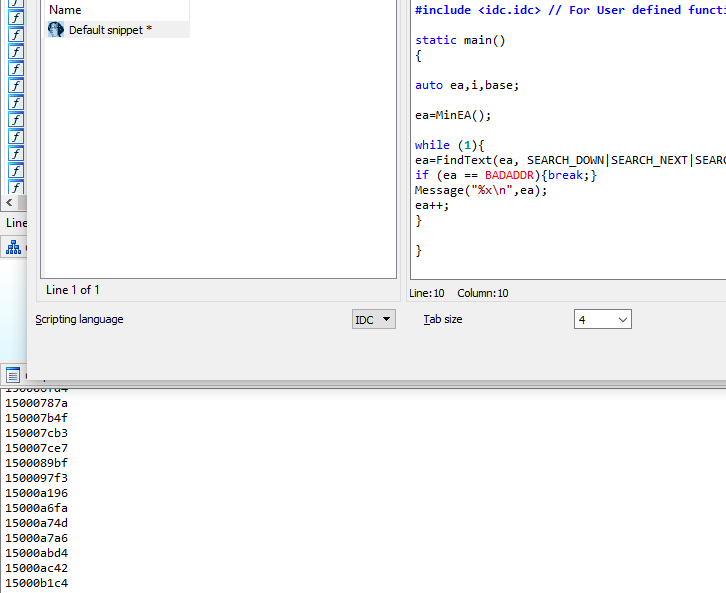
O sea que si ea=-1 quiere decir que no encontró nada termine.



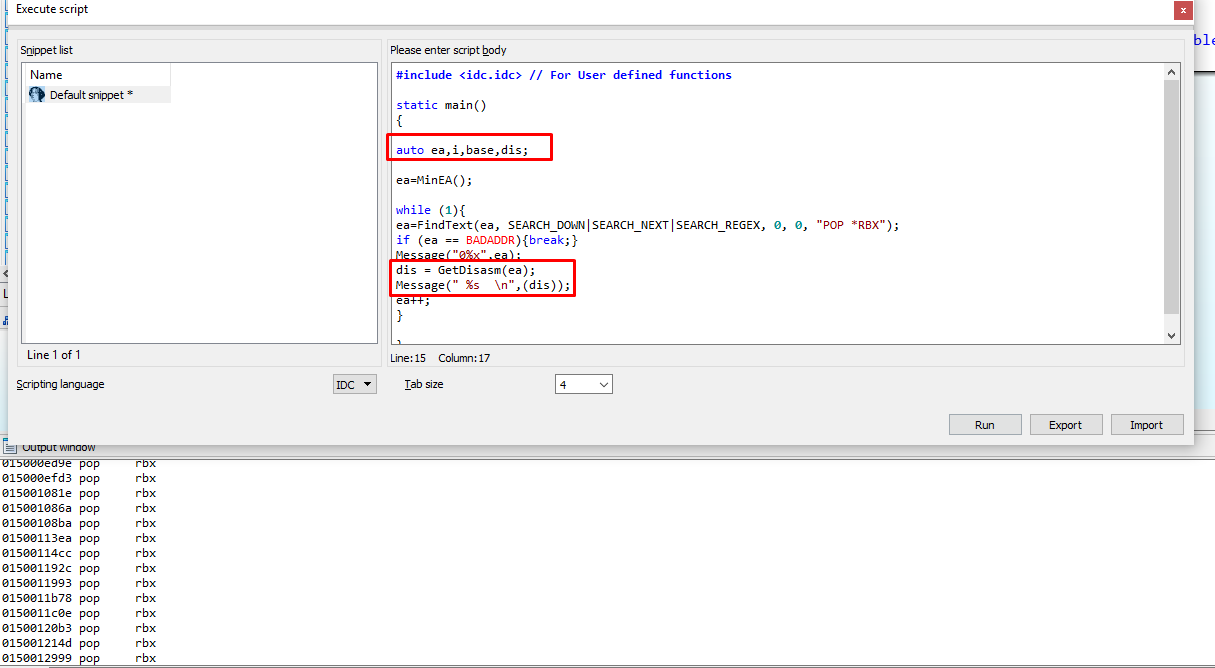
Con eso podría buscar no solo 10 sino todas la que hay.



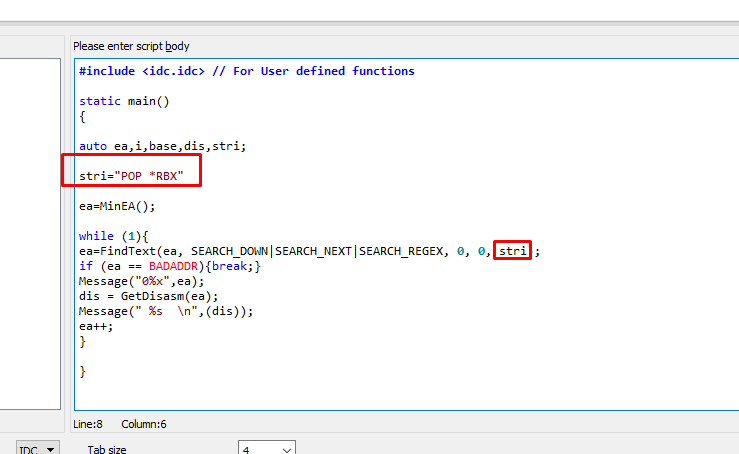
Busca hasta que le devuelve -1 todas las que pueda hallar.



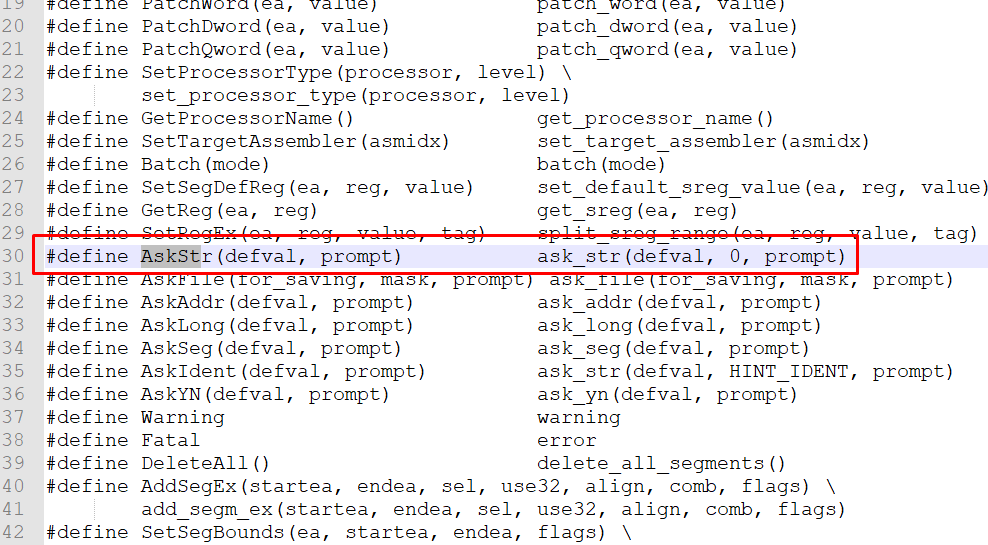
Nos falta imprimir la instrucción hallada uso Get Disasm() para hacerlo como me lo devuelve como string, hago un format string usando Message para imprimirlo.

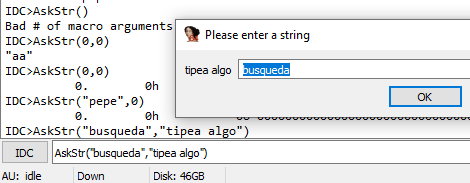


Vamos mejorando jeje.



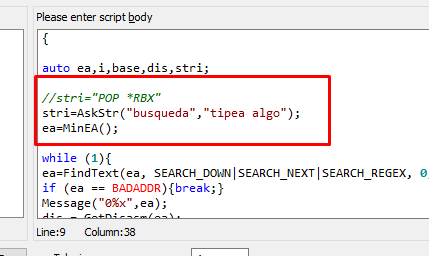
Conviene poner la string a buscar al principio incluso se puede pedir que se la tipee con una ventana para que el usuario ingrese lo que quiere buscar.

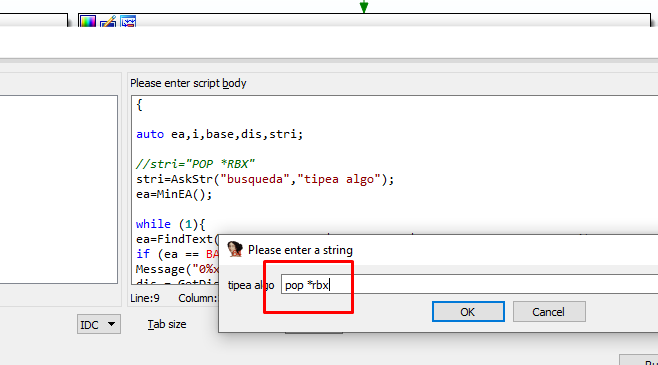




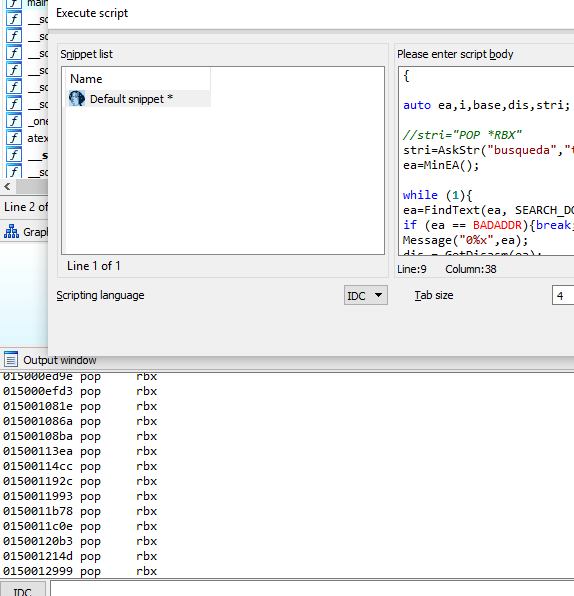
Igual no lo agregare eso por ahora, dejemos que la o las strings se coloque en una variable.

El que le guste puede hacerlo así.

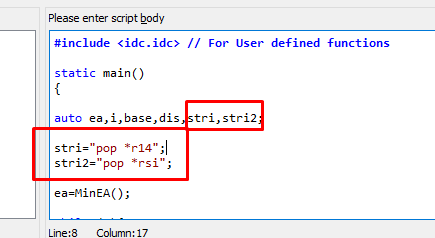




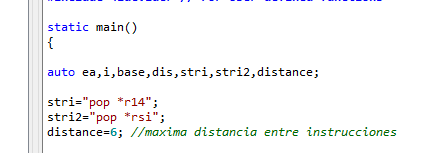
Vemos que funciona de la misma forma.

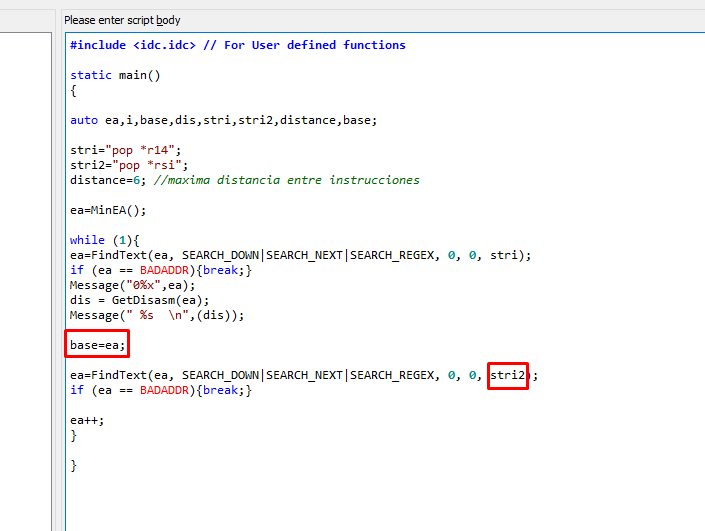


Agregare otra string.



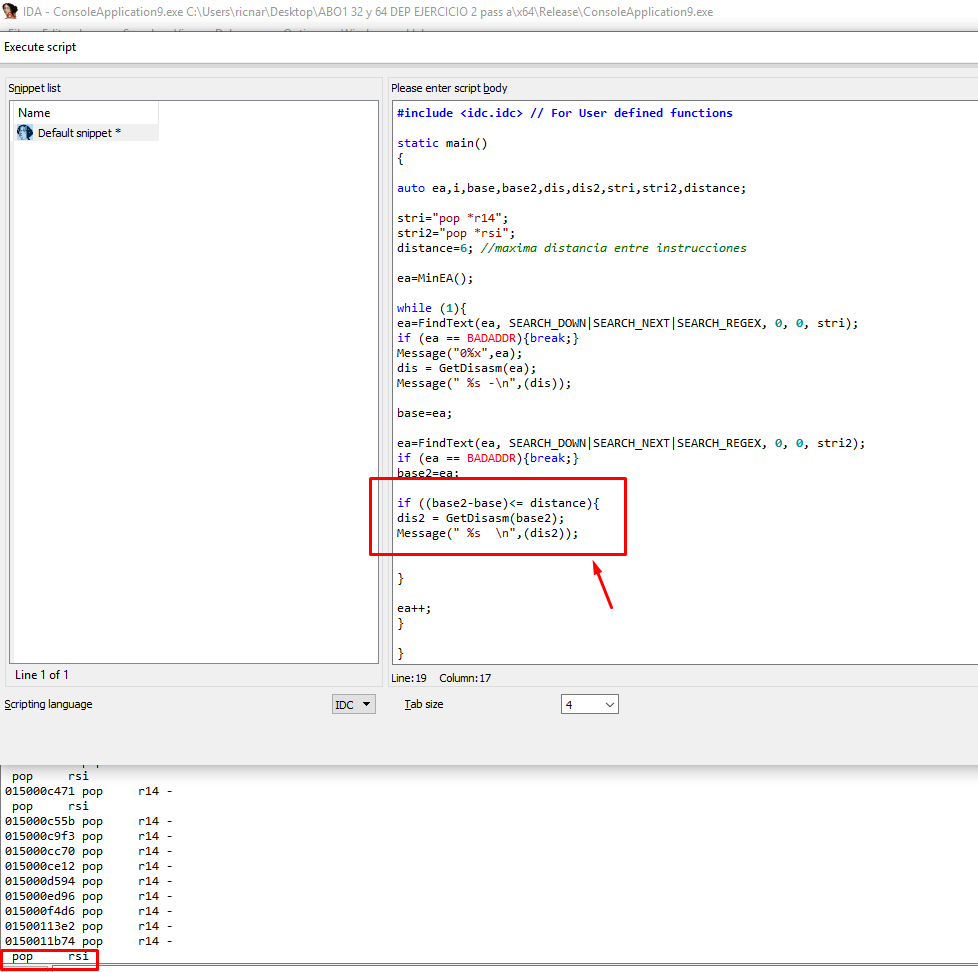
A partir de la dirección que se halle la primera buscaré la segunda, y habrá una distancia máxima que pondré para que haya entre las dos, si la distancia es menor que esa distancia máxima, lo acepto sino no.



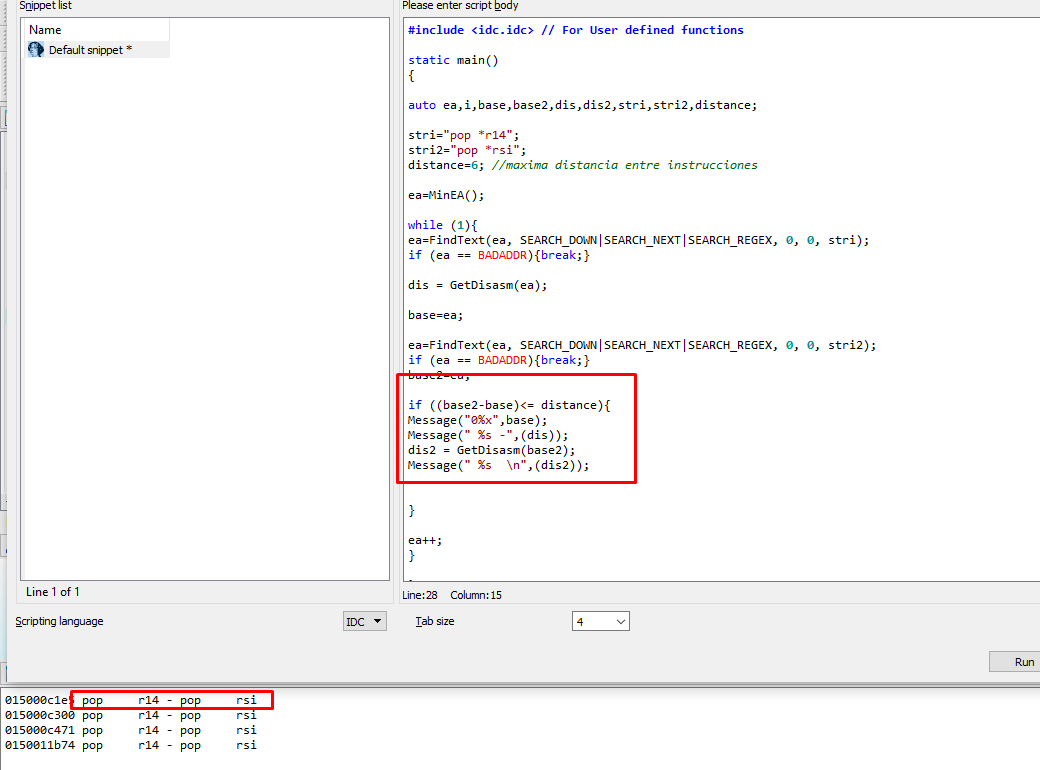


Guardo la dirección de la primera coincidencia en base, y busco la segunda string a partir de la primera ya que ea tiene la dirección de la primera string.

Si encuentra la segunda string a partir de la primera debo chequear la distancia entre ambas.



Vemos que algunas me imprimi solo el POP r14 y no halla el POP RSI y en otras halla ambos, moveré el print al final para que solo imprima cuando halla los dos.



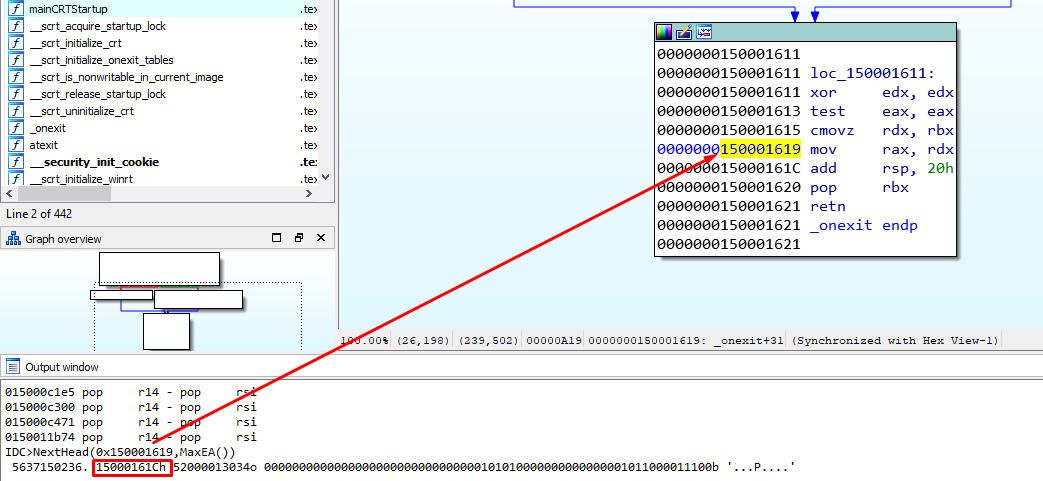
Puse todos los Message abajo para que solo imprima cuando halla ambas strings.

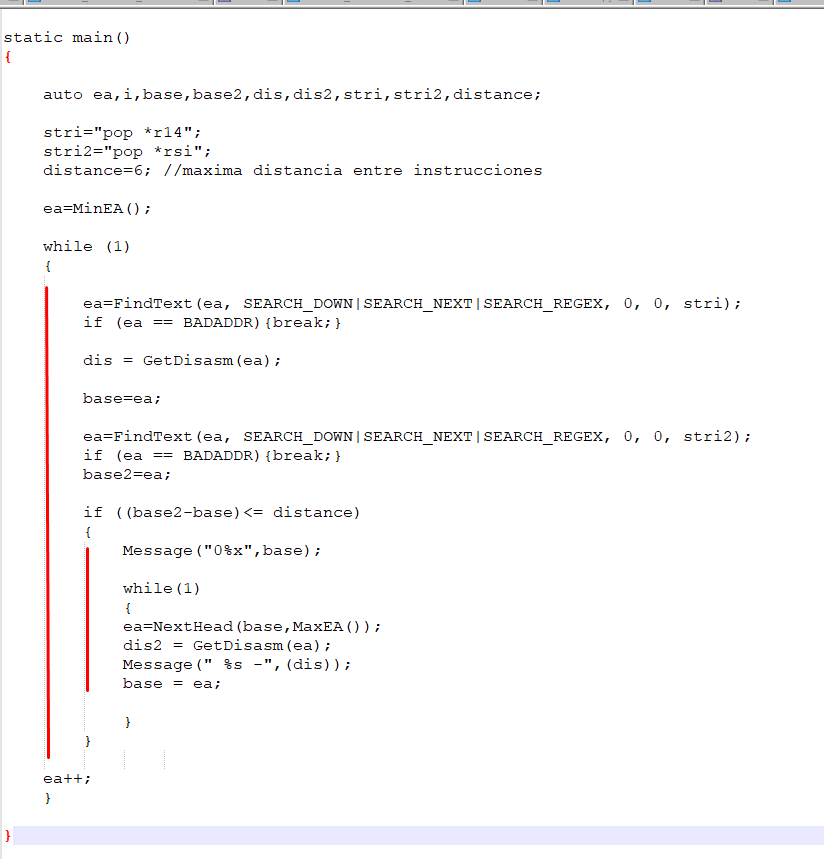
El problema que hay ahora es que si vemos a veces hay instrucciones intermedias entre ambas, ya que pueden estar a 6 de distancia una de otra, estaria bueno imprimir las instrucciones intermedias.

Para ellos usaremos la instrucción.

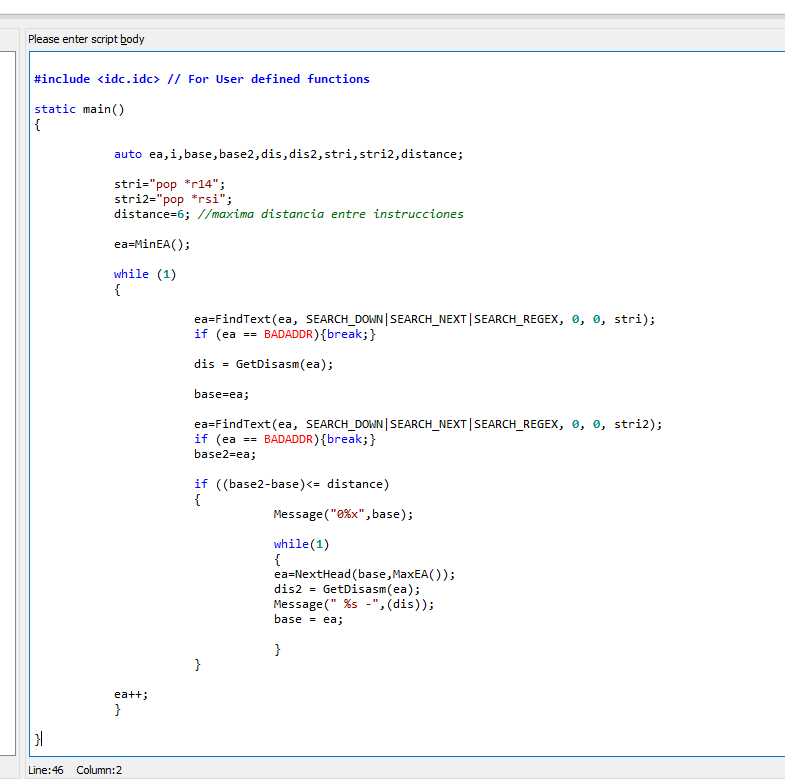
NextHead(base,end\_addr);

Que me da la dirección de la siguiente instrucción a la actual.

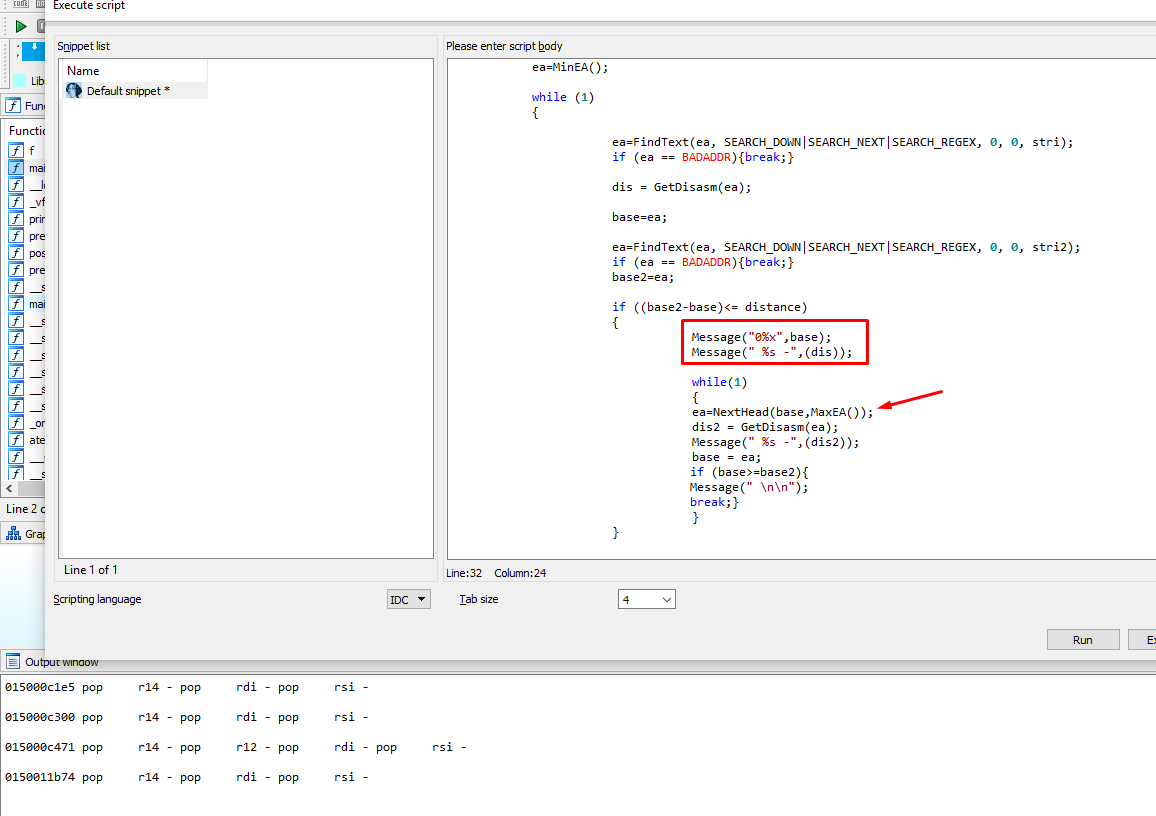




Ahi lo lleve al NOTEPAD ++ y lo alinee un poco para no hacer lio con las llaves.



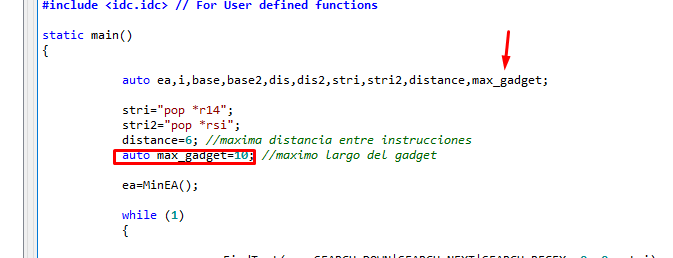
Ahora me falta la condición de salida hasta donde imprima, si no sigue imprimiendo hasta el infinito.



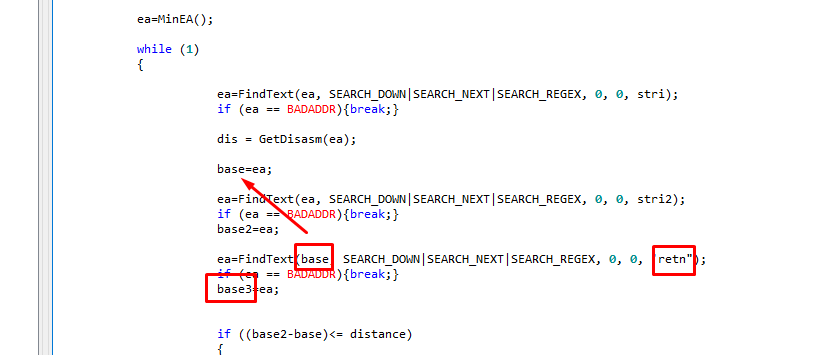
Ahí está ahora imprime todas las instrucciones intermedias entre las dos instrucciones.

Ahora le voy a agregar un máximo largo del gadget hasta el ret

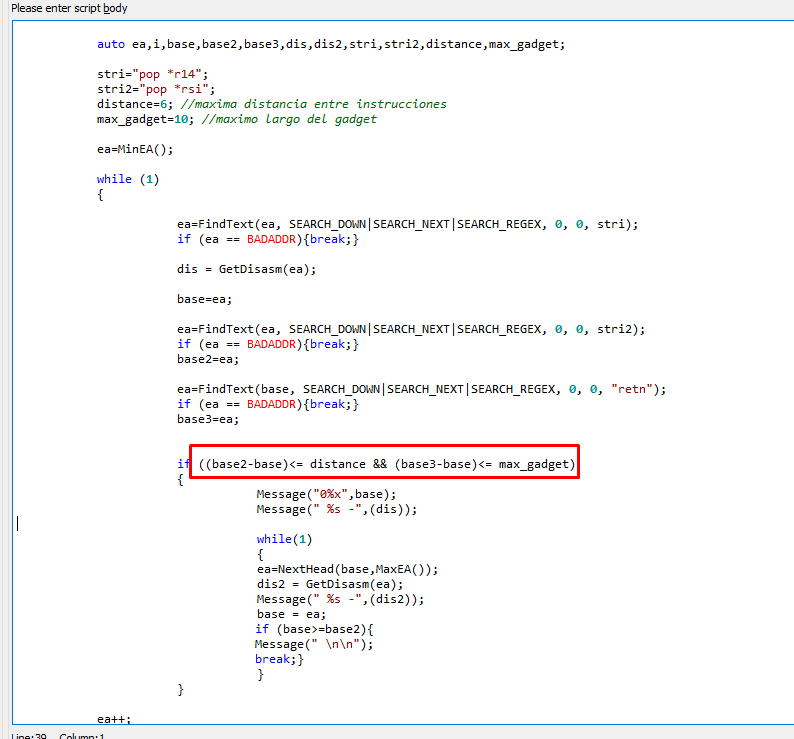
auto max\_gadget=10; //maximo largo del gadget



Entonces también debo buscar un ret desde la base o sea desde la primera coincidencia que encuentra.

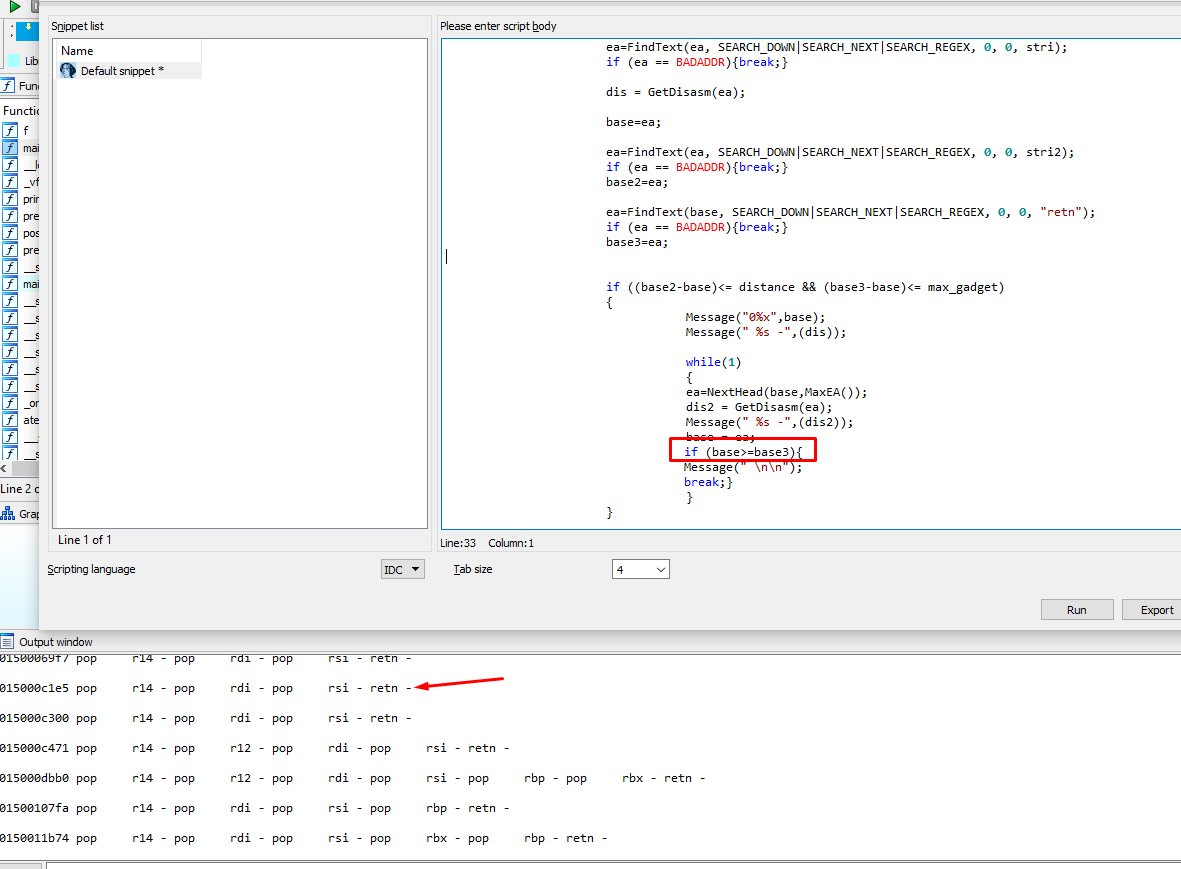


Agrego base3 y hago una busqueda desde base de la string retn y guardo la dirección en base3.



O sea que ahora es que la condición es que ambas instrucciones esten separadas a una distancia menor que distance, y que haya un ret a una distancia menor que max\_gadget desde base.

Ahora debemos hacer que imprima no hasta base2 sino hasta base3 o sea hasta el ret.



No quedó tan mal jeje se puede agrandar y buscar más instrucciones, lo importante aquí es practicar, aquí dejo el script.

SCRIPT FINAL

#include <idc.idc> // For User defined functions

static main()

{

auto ea,i,base,base2,base3,dis,dis2,stri,stri2,distance,max\_gadget;

stri="pop \*r14";

stri2="pop \*rsi";

distance=6; //máxima distancia entre instrucciones

max\_gadget=10; //maximo largo del gadget

ea=MinEA();

while (1)

{

ea=FindText(ea, SEARCH\_DOWN|SEARCH\_NEXT|SEARCH\_REGEX, 0, 0, stri);

if (ea == BADADDR){break;}

dis = GetDisasm(ea);

base=ea;

ea=FindText(ea, SEARCH\_DOWN|SEARCH\_NEXT|SEARCH\_REGEX, 0, 0, stri2);

if (ea == BADADDR){break;}

base2=ea;

ea=FindText(base, SEARCH\_DOWN|SEARCH\_NEXT|SEARCH\_REGEX, 0, 0, "retn");

if (ea == BADADDR){break;}

base3=ea;

if ((base2-base)<= distance && (base3-base)<= max\_gadget)

{

Message("0%x",base);

Message(" %s -",(dis));

while(1)

{

ea=NextHead(base,MaxEA());

dis2 = GetDisasm(ea);

Message(" %s -",(dis2));

base = ea;

if (base>=base3){

Message(" \n\n");

break;}

}

}

ea++;

}

}

Por supuesto lo pueden exportar sino lo perderán, y guardar en un archivo y correrlo desde el menú las veces que quieran.

Creo que es una idea linda para practicar, mejorar este script y agregarle más cosas, es una base para que se diviertan un rato con IDC que no es tan terrible como lo pintan por ahí.

Hasta la parte 19.

2/06/2020

Aun en 40tena jeje