INTRODUCCIÓN AL REVERSING CON IDA PRO DESDE CERO PARTE 22

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

### DIFFERS

Un differ es un programa que a partir de dos versiones consecutivas de un mismo programa, trata de emparejar o matchear las funciones a pesar de los cambios y nos trata de mostrar cuáles funciones fueron cambiadas y donde.

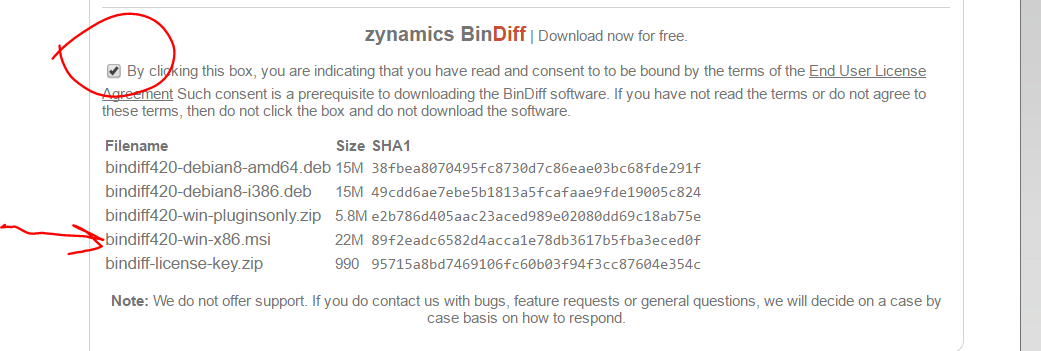
Obviamente este trabajo no es sencillo, sobre todo cuando de una versión a otra ha habido muchos cambios, los cuales pueden ser desde la aplicación de algún parche de seguridad para solucionar alguna vulnerabilidad, como también puede haber mejoras en el programa, agregados en la interfase, cambios generales, etc.

Los que tenemos que trabajar con differs sabemos que cuanto más grande y más cambios haya en el ejecutable, más ingrato es el trabajo ya que el differ comete algunos errores al matchear.

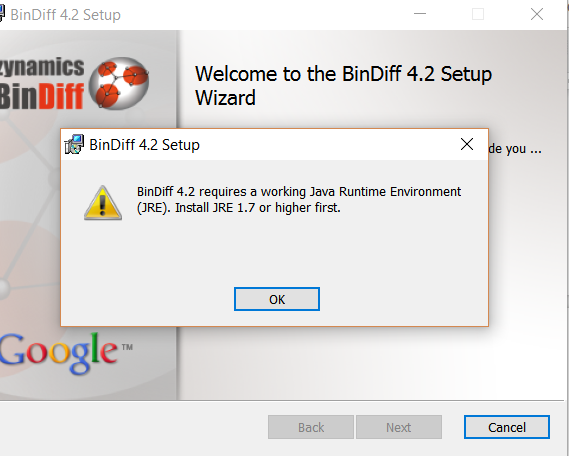
Vamos a ver los tres differs más conocidos que usaremos en general, para ir instalándolos y conociéndolos, cada uno tiene su punto fuerte y su punto débil, la realidad hace que a veces haya que usar más de uno en casos complejos para tratar de aclararse.

El primero que instalaremos será el BINDIFF.

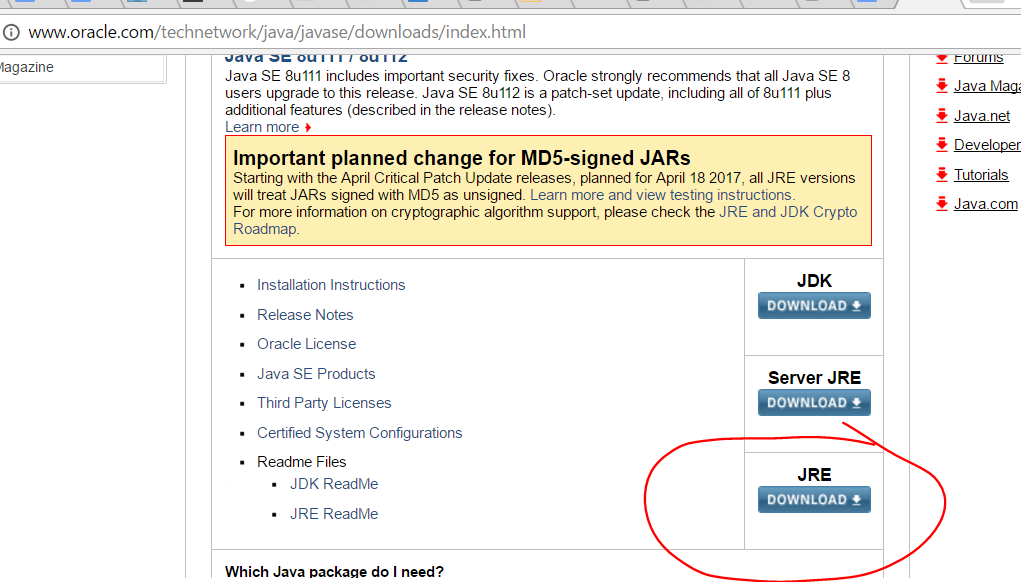
<https://www.zynamics.com/software.html>

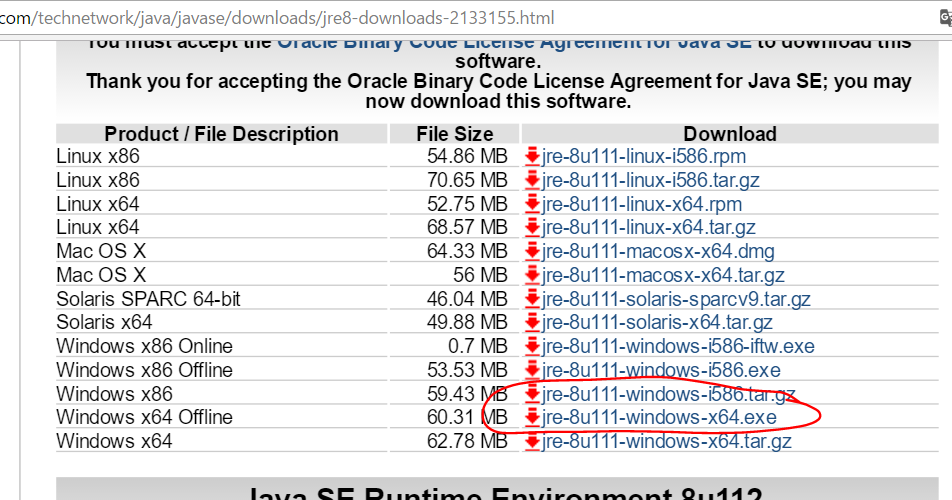


De allí se baja aceptando las condiciones.



Bueno a bajar java si no lo tenemos.



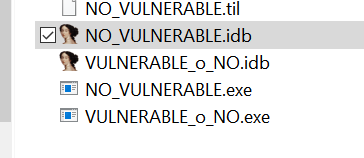


Parece ser esta la versión bajamos e instalamos.

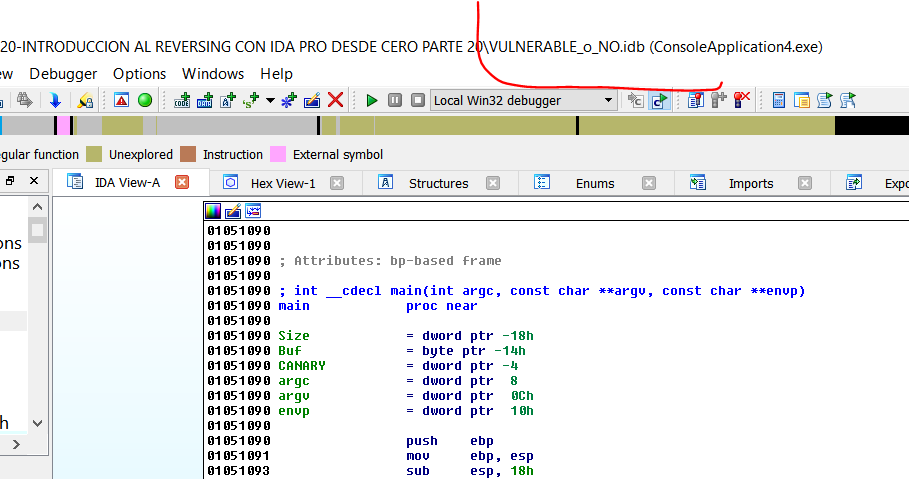


A continuación reintento instalar el bindiff y parece que no hay problema, no se queja.

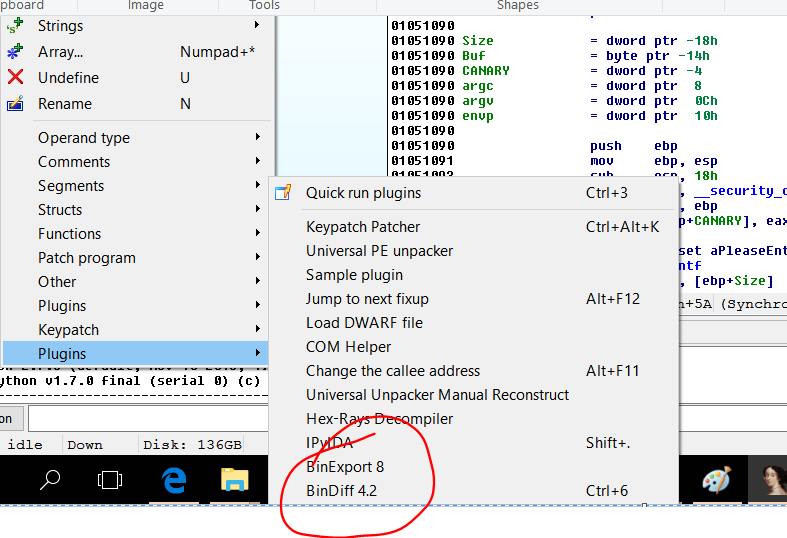
Todavía tengo del ejercicio anterior los ejecutables el vulnerable y el no vulnerable, abro el más nuevo o sea el no vulnerable o parcheado en el LOADER del IDA para que cree el IDB si no lo hice antes.



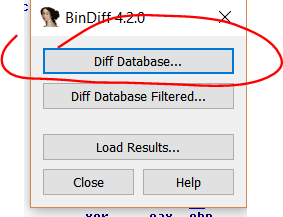
Luego abro el vulnerable.



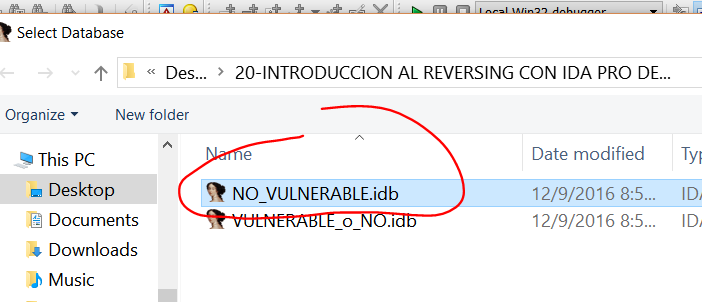
Luego voy a edit-plugins.

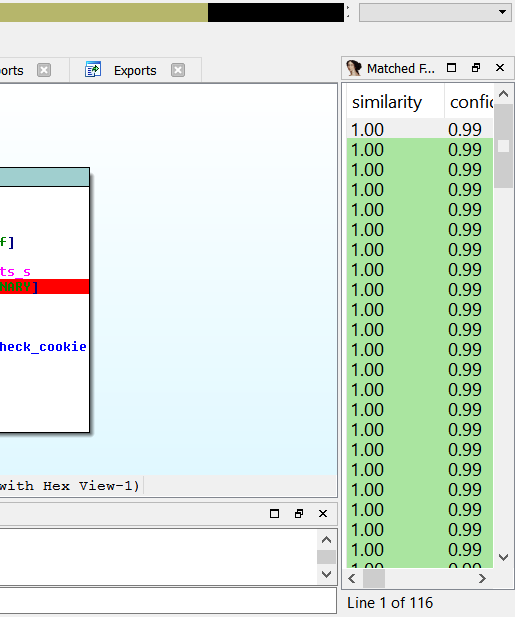


y elijo BINDIFF

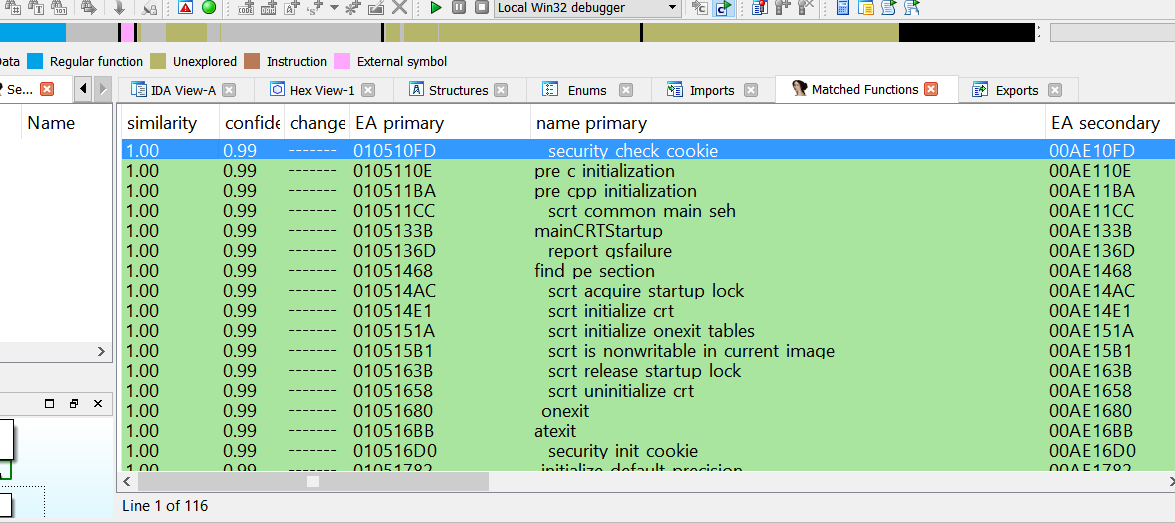


Y apreto DIFF DATABASE y busco la versión parcheada contra la cual comparara.

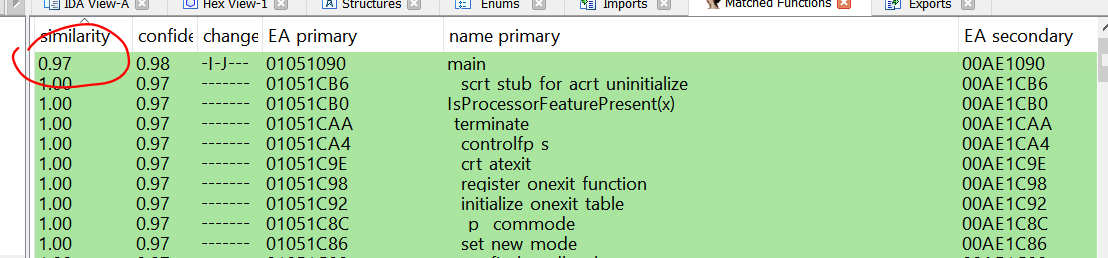




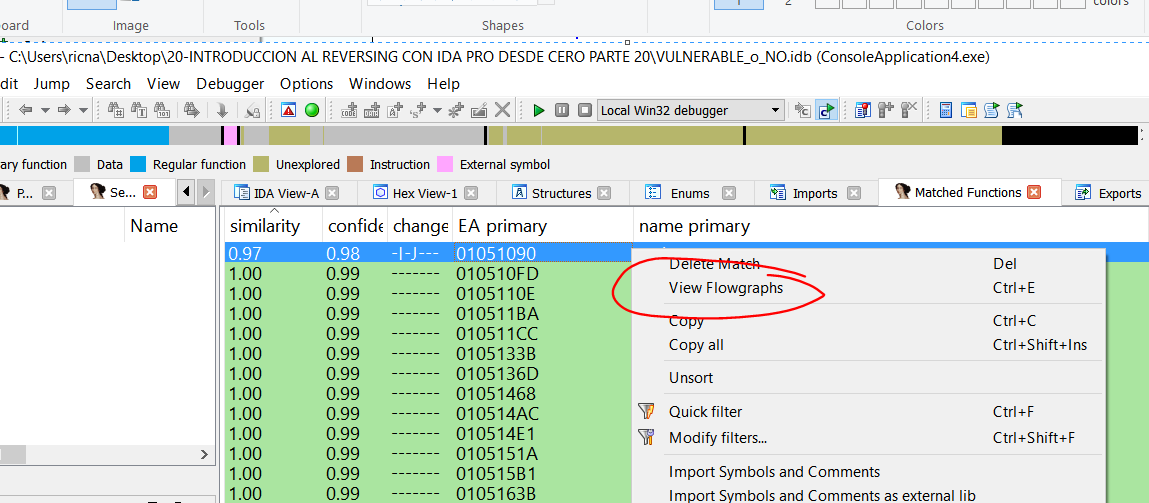
Una vez que termina me muestra los resultados a veces no es cómodo que se vea en una columna, así que puedo arrastrarlo y soltarlo en la barra de pestañas.

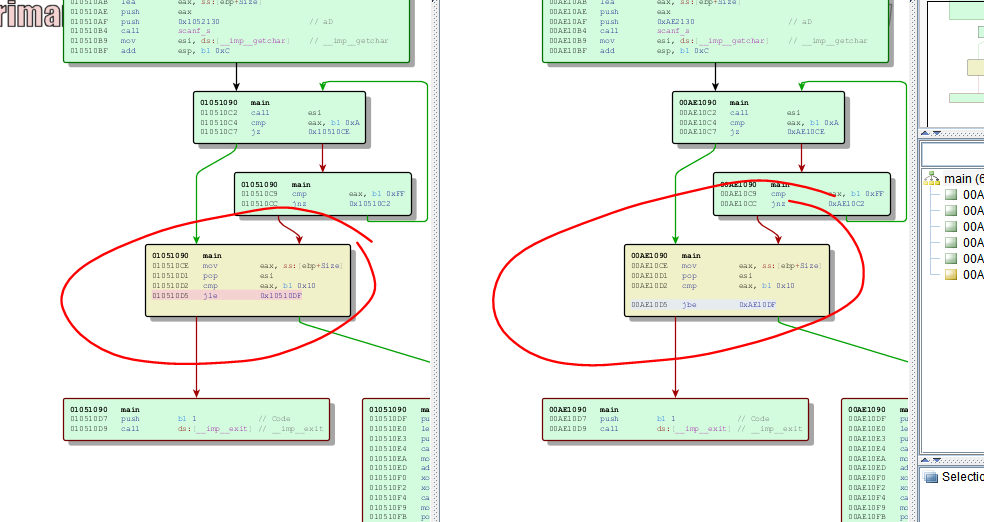


Ahora sí, la primera columna nos muestra la similaridad, las que dicen 1.00 son iguales y cuanto más bajo es el número, más diferentes son, conviene hacer click en la parte superior de esa columna para que las ordene de más diferente a más similar.

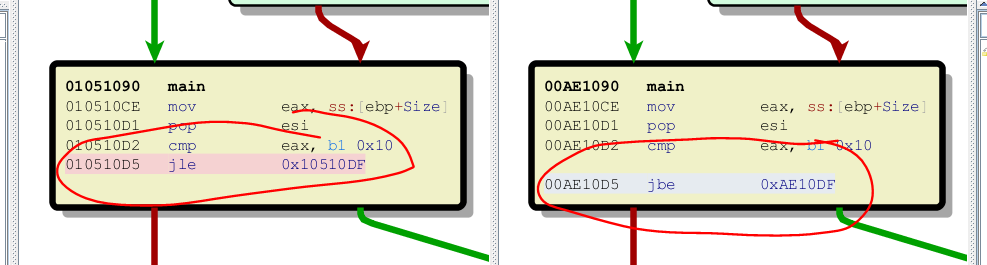


Vemos que hay una sola con similaridad menor que 1.



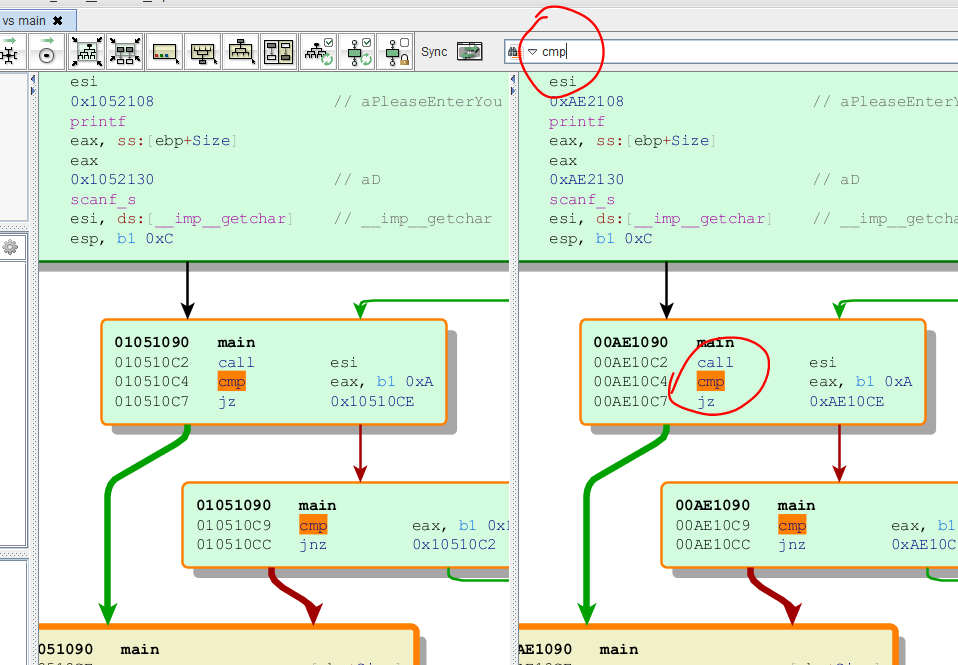


Los bloques verdes son los similares, los amarillos tienen algún cambio y los rojos o grises son agregados.

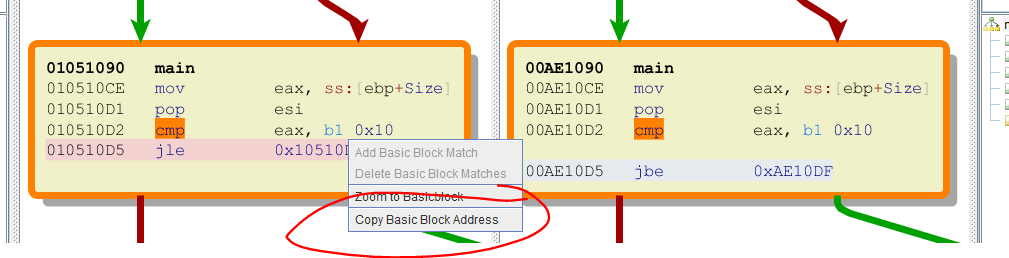


Allí vemos el cambio, como sabemos cambiar un salto JLE por JBE es algo que puede evitar un BUFFER OVERFLOW, por lo tanto si en un programa del cual tenemos ambas la versión vulnerable y la versión parcheada y mirando las funciones cambiadas en alguna encontráramos esto sabremos que tendremos que ir a reversear estáticamente esa función a ver si realmente es la vulnerable del programa.

Una de las ventajas de BINDIFF sobre los otros dos es que el gráfico es interactivo, no una imagen solamente, tiene un buscador arriba.

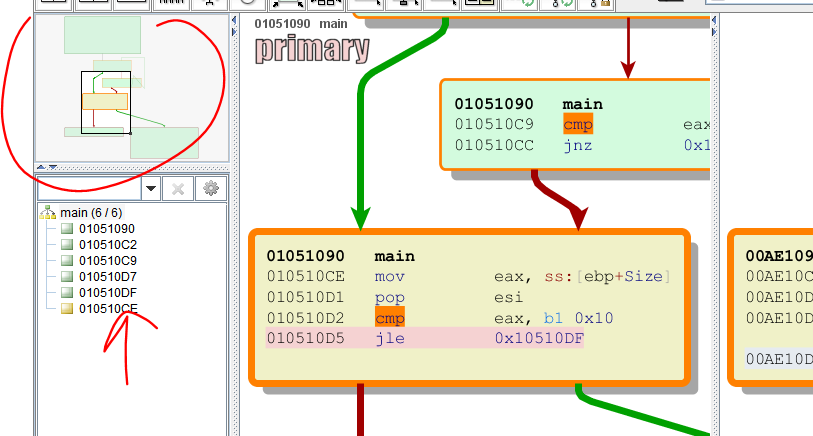


Lo cual es muy útil, y también se pueden buscar direcciones y cualquier texto que este en el gráfico.



Podemos copiar la dirección del bloque para pegarla en IDA e ir allí.

Tenemos también un navegador gráfico para recorrer la función y la lista de bloques.



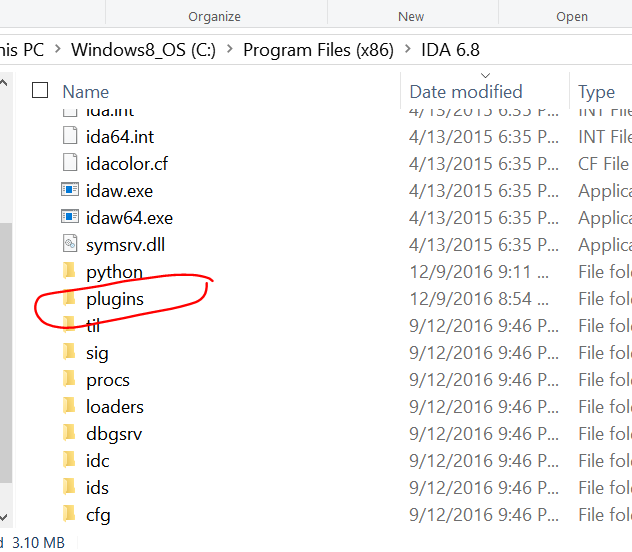
Podemos marcar un bloque y en el menú tenemos select ancestor o select succesors para que nos oscurezca los bloques del camino dentro de la función para llegar al mismo bloque inicial, o desde allí, en este caso es una sencilla función, pero en funciones grandes y complejas encontrar el camino a un bloque es muy importante.

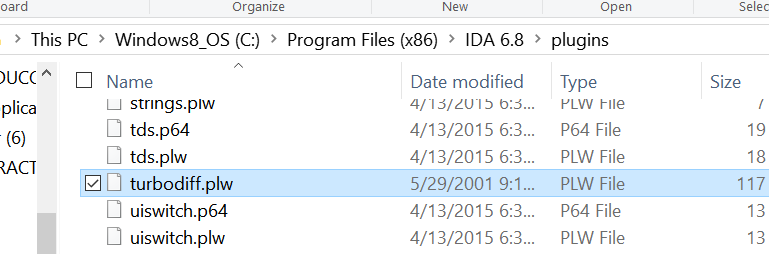
Tiene muchas cosas buenas el bindiff sobretodo en la parte gráfica, tiene algunos problemas de matcheo en programas grandes, pero es una de las mejores opciones a tener en cuenta.

TURBODIFF

Es un differ creado por mi compañero de Core Nicolas Economou, el mismo estará adjunto con el tute, también se puede descargar de la web de CORE SECURITY pero es una versión anterior a la que puse adjunta.

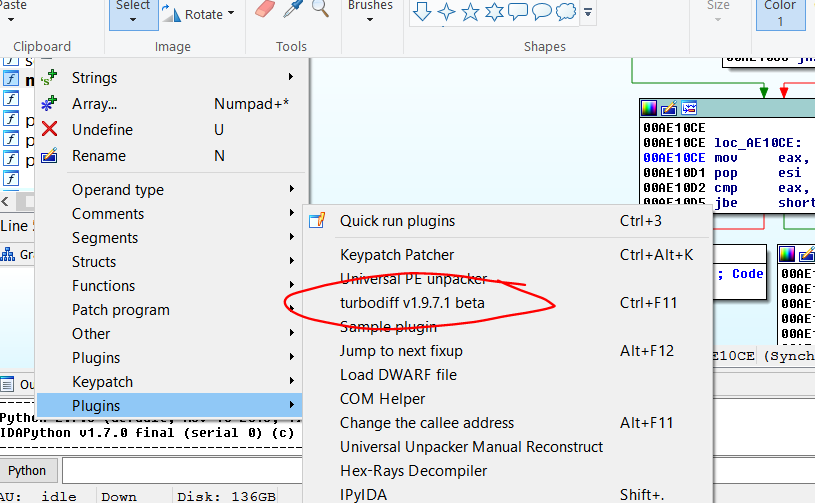
El PLW se copia a la carpeta plugins dentro de la carpeta de instalación del IDA.



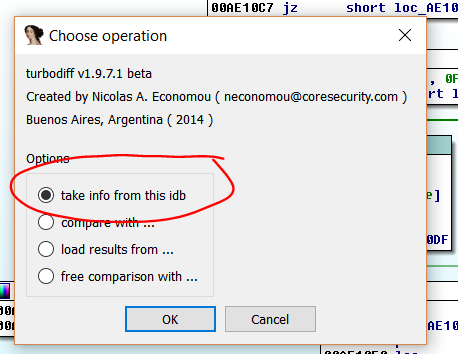


Tendré que volver a arrancar el IDA para que lo cargue.

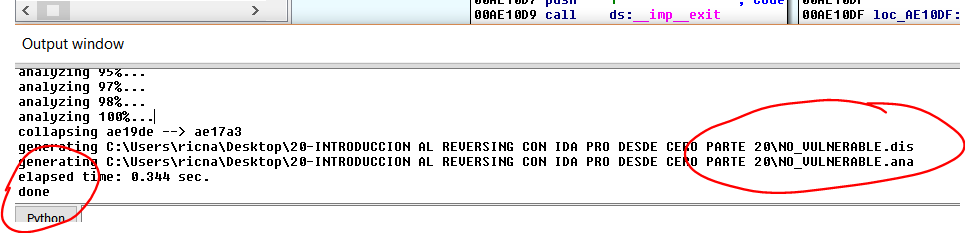
Como siempre cargo la versión NO VULNERABLE primero.



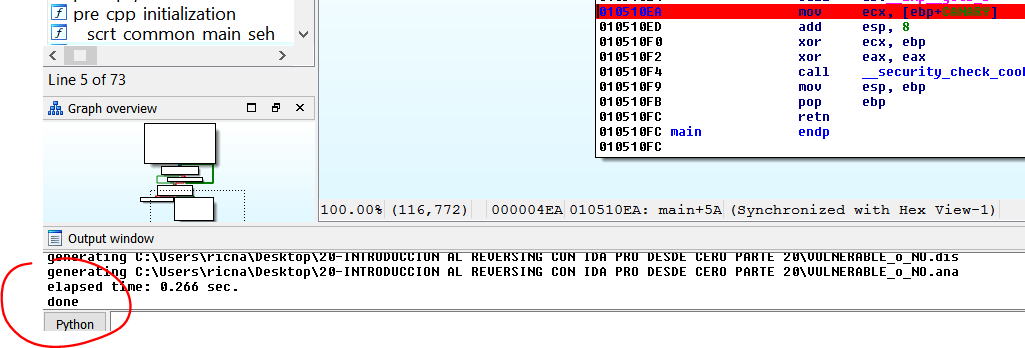
Hay que tomar la información de cada uno de los idb que se van a comparar.



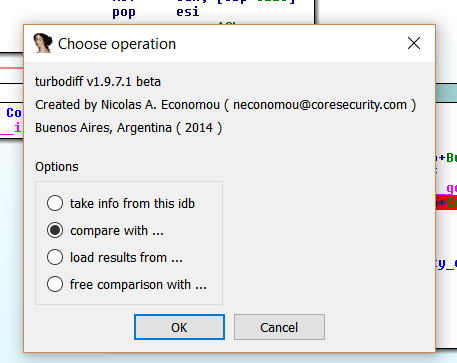
Así que hago eso en este primero.

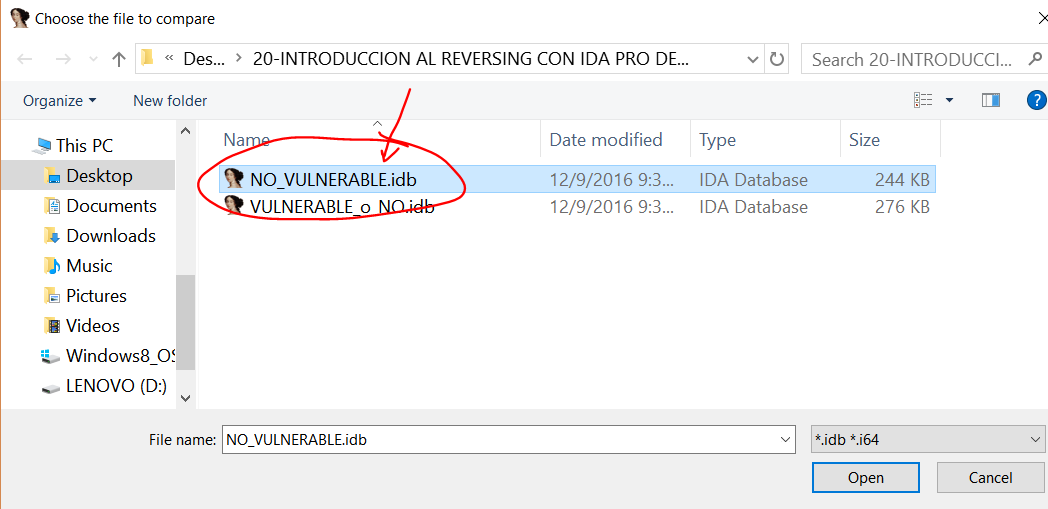


Luego abro el vulnerable y hago lo mismo.

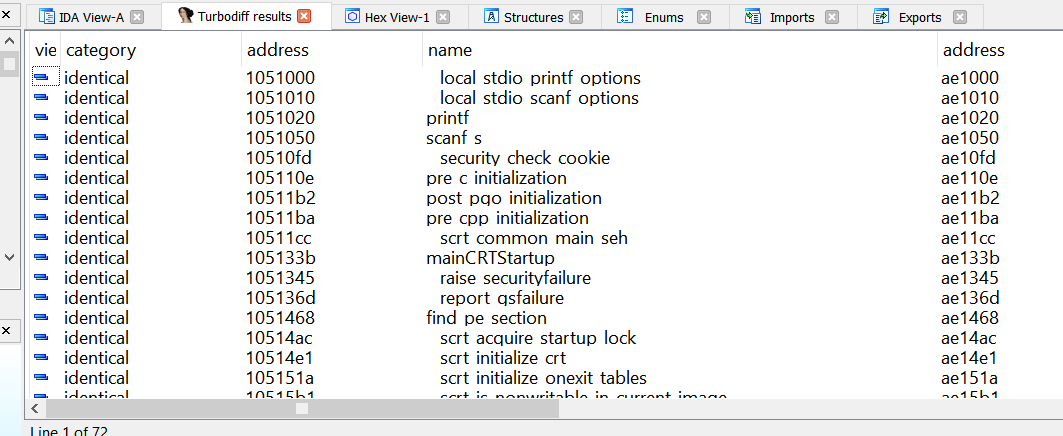


Luego desde el vulnerable vuelvo a llamar al plugin

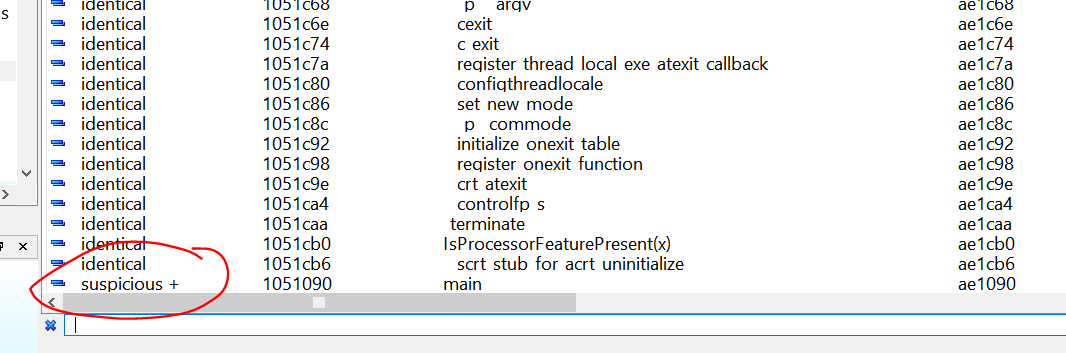




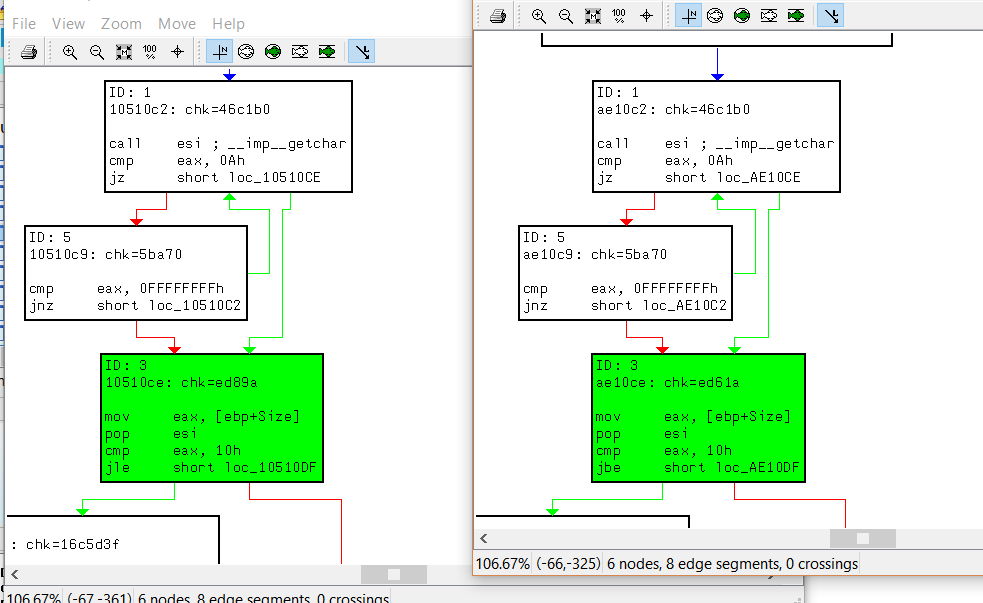
Busco la versión no vulnerable y acepto con las opciones que trae por default.



Allí pudo apretar CTRL mas F y buscar changed o suspicious para que me muestre las cambiadas.



Allí está haciendo doble click



Allí se ven las cambiadas también hay un código de colores según el tipo de cambio, verde para los bloques con mínimos cambios, amarillo para los bloques muy cambiados y rojo para los bloques agregados.

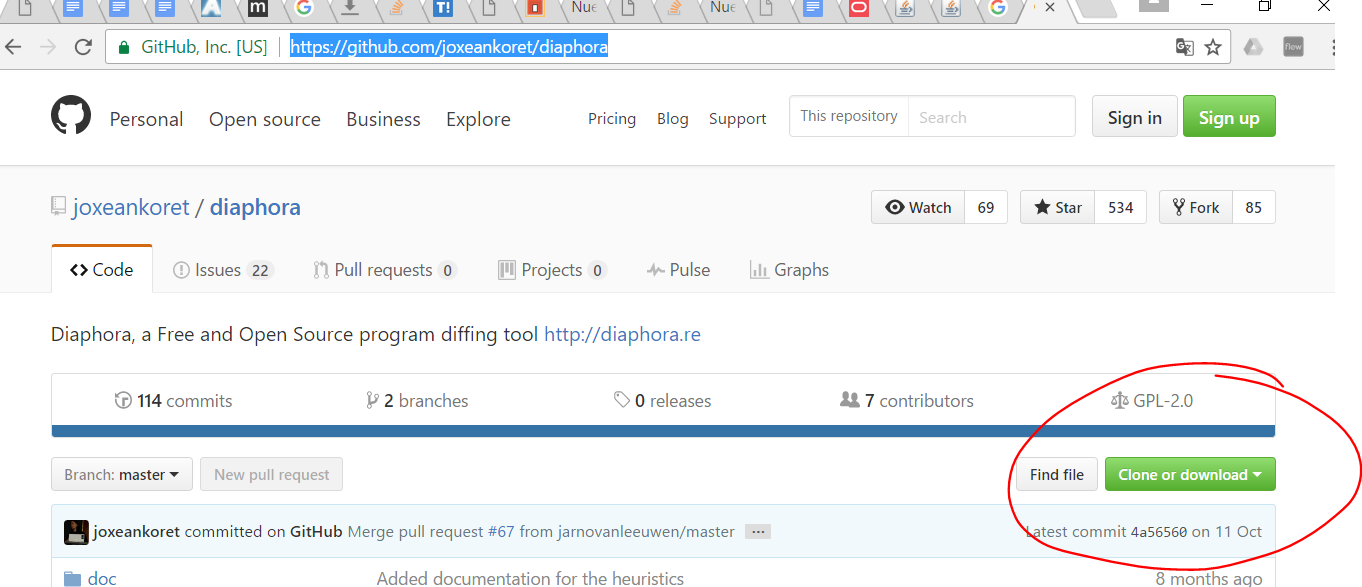
Obviamente los gráficos son imágenes y no son interactivos, pero es un differ muy rápido realmente el más rápido, se nota mucho en ejecutables grandes y que no muestra demasiados cambios tontos como el bindiff, asumiendo muchos como no importantes lo cual en grandes trabajos se agradece.

Si uno no le gusta la forma de los gráficos puede usar ambos differs a la vez y luego ver los resultados en el gráfico del bindiff.

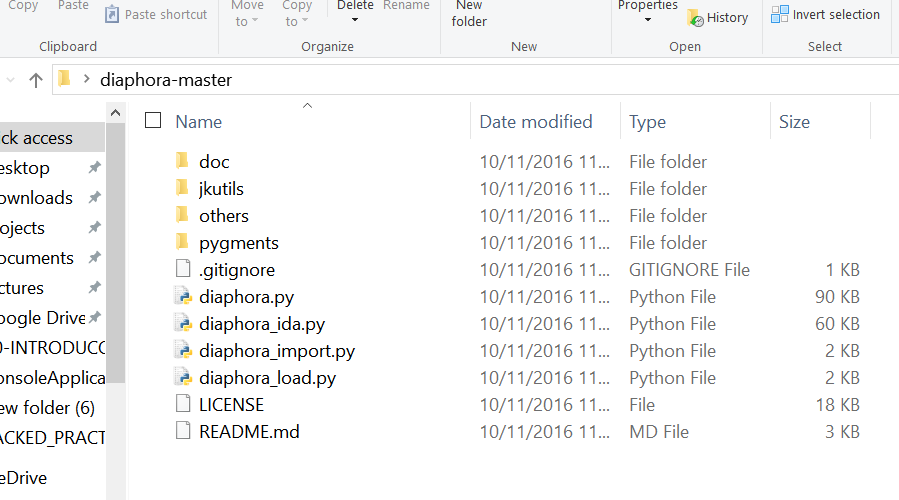
DIAPHORA

El diaphora es un plugin hecho en Python por Joxean Koret

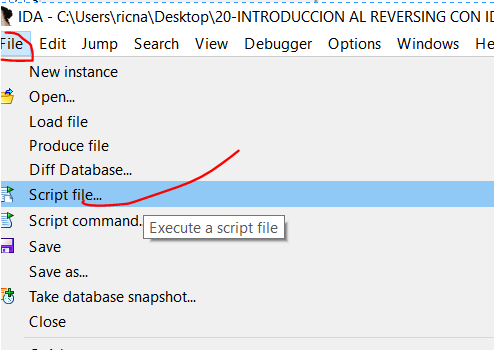
<https://github.com/joxeankoret/diaphora>



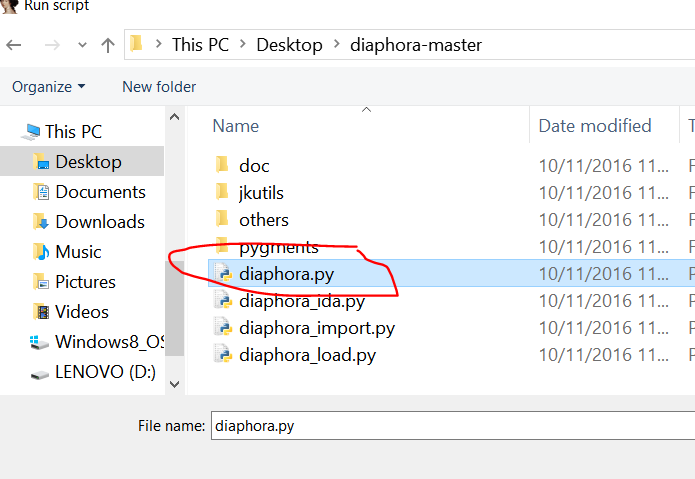
No es necesario instalarlo puedo descomprimirlo en cualquier lugar y solo es necesario tener instalado Python en la máquina lo cual si tenemos IDA ya lo tendremos.

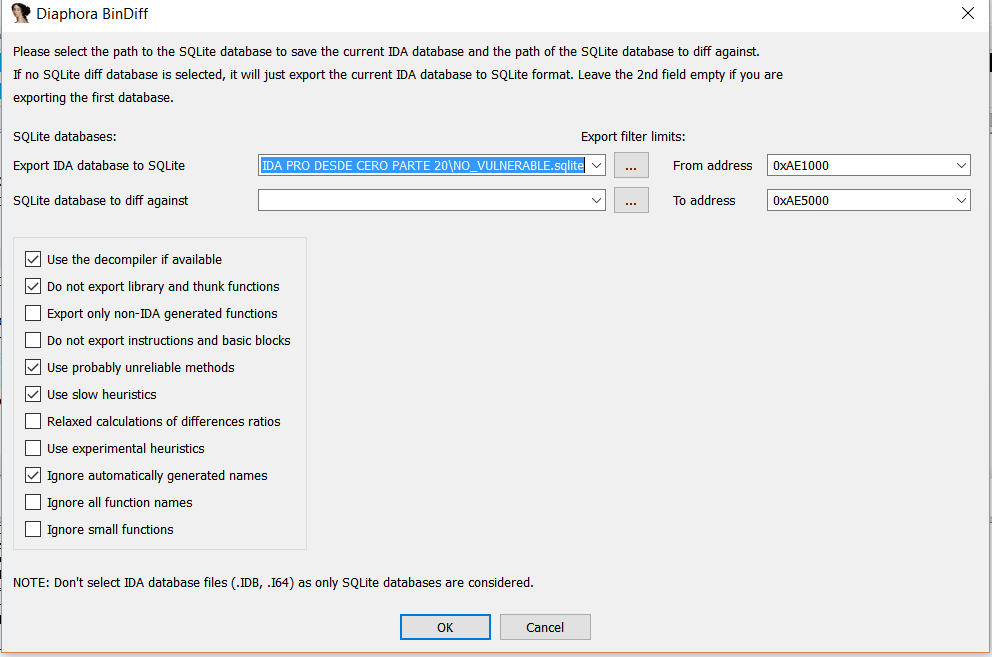


Así que iremos como siempre primero al parcheado o no vulnerable en el IDA.

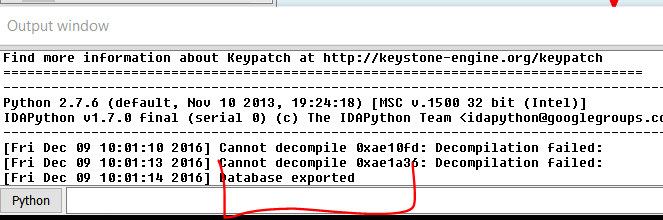


En FILE-SCRIPT FILE abre el buscador y vamos donde los descomprimimos al diaphora y buscamos diaphora.py.

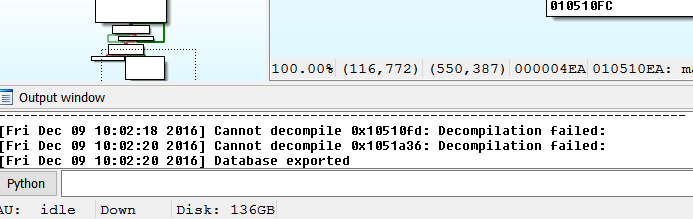




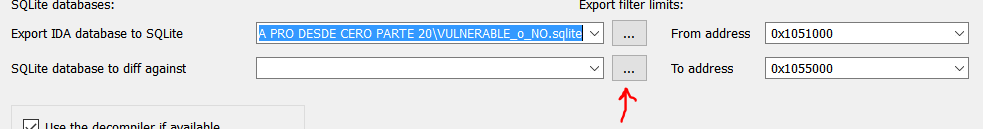
Así como viene apretamos OK para que exporte la database a SQL.



Cuando termina abrimos el vulnerable en IDA y hacemos lo mismo abrimos el diaphora.py y sin cambiar nada exportamos la database.

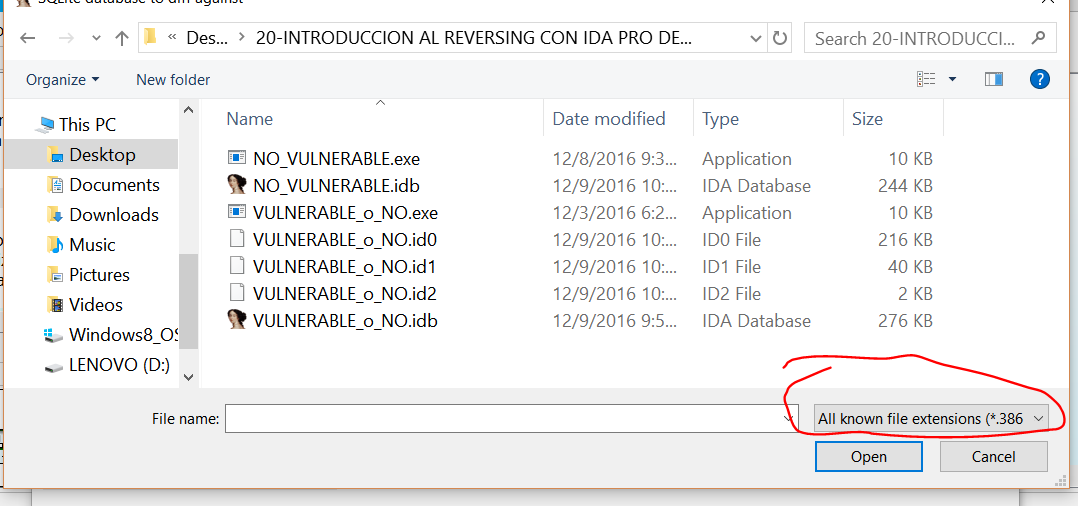


Una vez que hicimos lo mismo en ambas, volvemos a abrir el diaphora.py en la versión vulnerable pero esta vez.

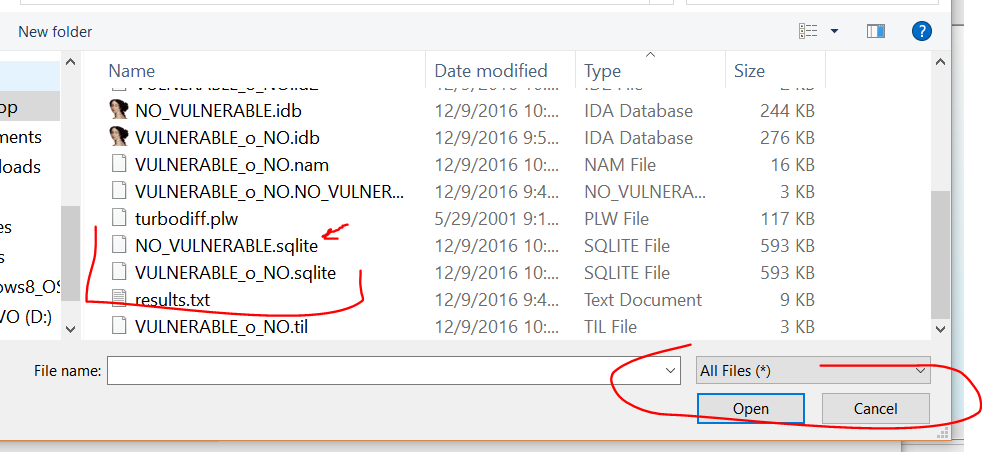


En el segundo lugar buscamos la SQL database del parcheado que exportó antes .

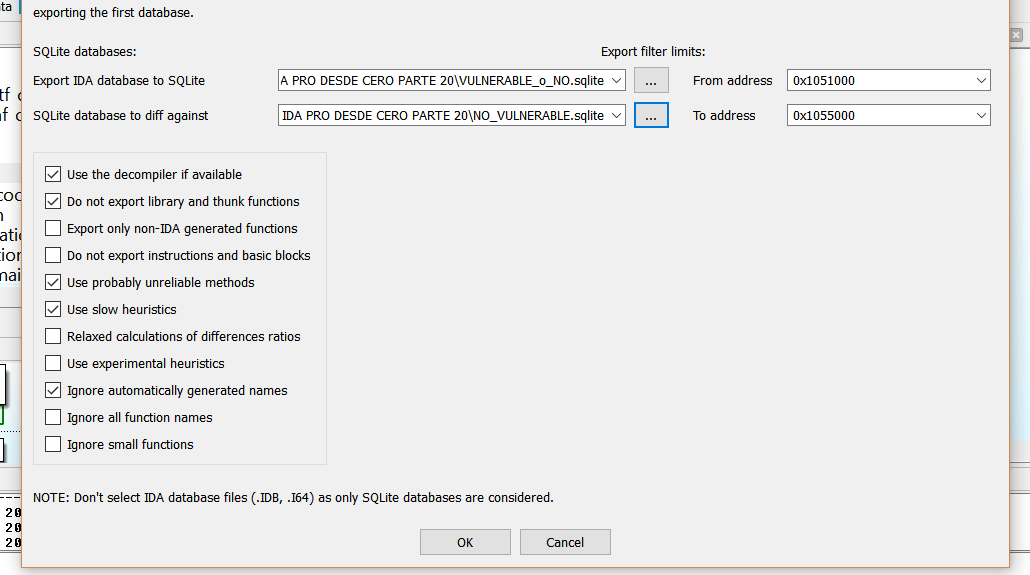
Vemos que cuando vamos a la carpeta parece no haber nada



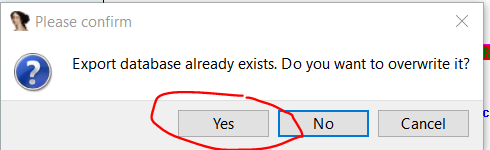
Pero es por el filtro de archivos lo cambiamos para que muestre todo.



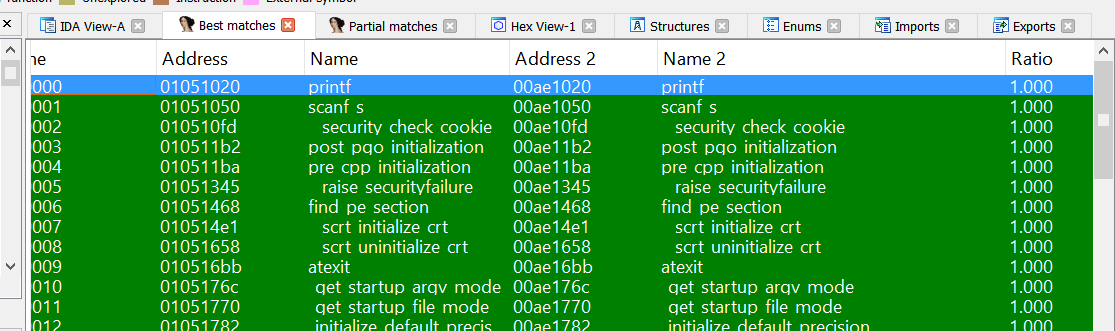
Y buscamos el de la no vulnerable.



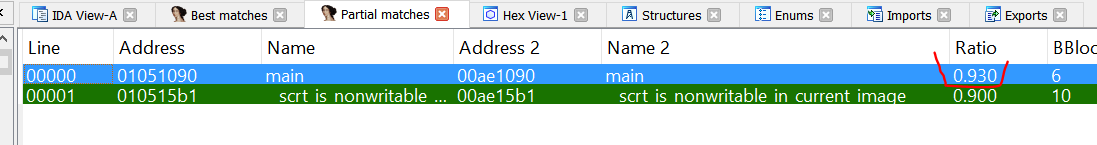
Apreto OK así como esta.



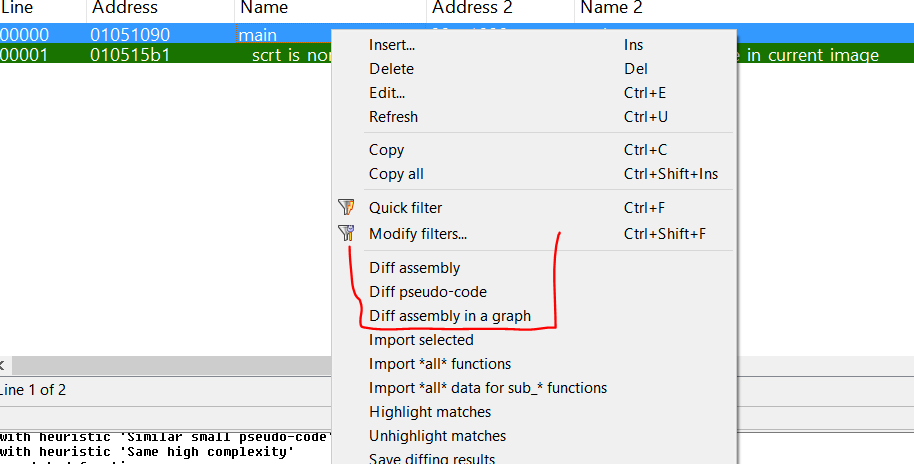
Vemos que hay una pestaña BEST MATCHES con las que no hay duda de que son iguales.



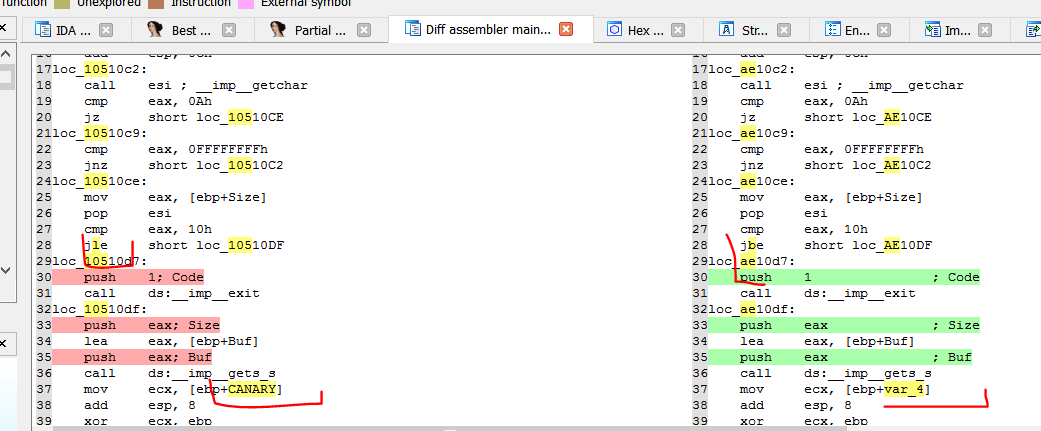
En la pestaña PARTIAL MATCHES vemos las que posiblemente se cambiaron.



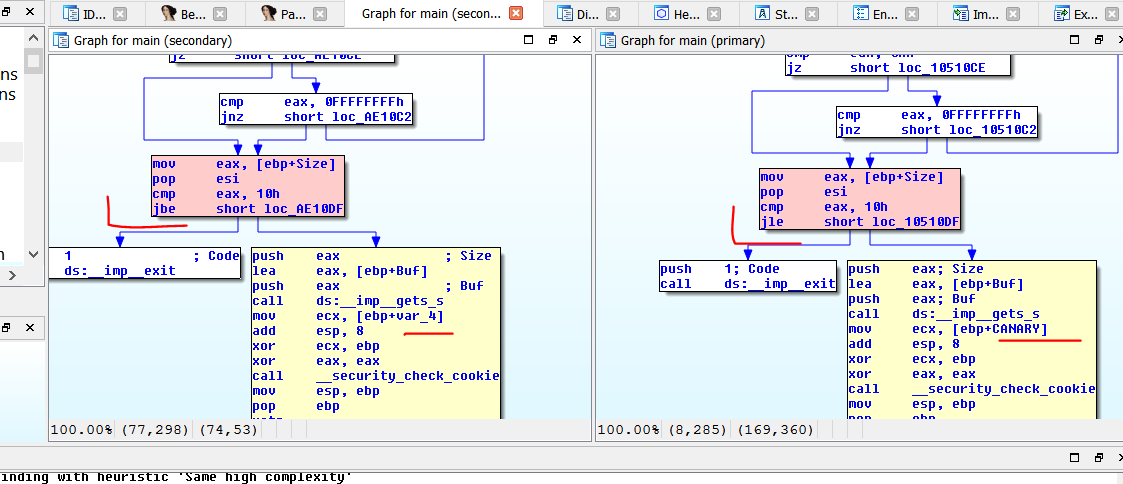
Allí está la versión cambiada, vemos que encontró dos cambiadas, una de las cosas que tiene diaphora es que es muy preciso a veces eso es bueno, pero a veces cuando tienes cientos de funciones, quieres que sea un poco más relajado y no muestre tantas pavadas como cambios.



Vemos que tiene varias opciones para graficar la primera DIFF ASSEMBLY.

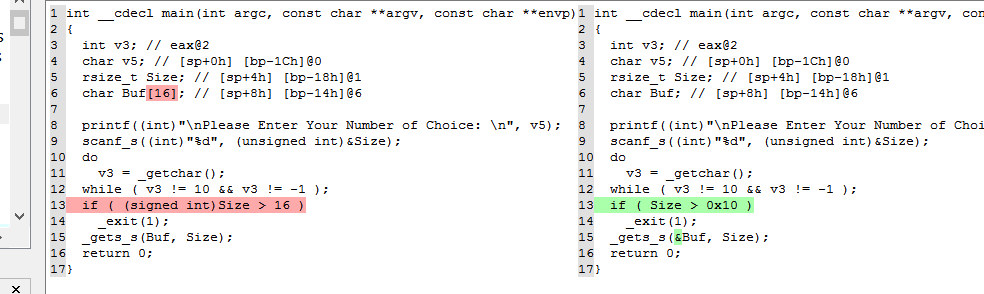


Es como muy preciso y detallado pero cuando ves cien funciones así, te quieres matar jeje, veamos la segunda opción DIFF ASSEMBLY IN A GRAPH.



Este gráfico es un poco mejor, aunque no es interactivo, el bloque importante cambiado está en rojo y en amarillo los que tienen cambios menores como el nombre de una variable.

La otra opción DIFF PSEUDO CODE usa el plugin HEX RAYS que trae el IDA incluido que trata de armar un código fuente a partir del ejecutable.



Vemos que en el vulnerable que habíamos reverseado nosotros a mano y determinado que había un buffer de 16 bytes, a esa variable Buf la detecta como buffer, pero en el otro como no hicimos el mismo trabajo no lo detecta sino como una variable char nada más, también muestra que la variable es signed en el vulnerable y en el otro no dice nada lo que supone que será unsigned.

Otra característica de diaphora es que es el más lento (está programado en python contra C del turbodiff) y en ejecutables grandes es muy largo el análisis y macheo.

Adjunte el archivo IDA1.exe me gustaría que lo analicen y vean si es vulnerable y además si se puede desbordar el buffer y modificar el flujo del programa para que nos muestre el cartel de chico bueno.

Se discute abiertamente el ejercicio tanto en la lista de correo crackslatinos como en nuestro grupo de telegram.

https://telegram.me/CLSExploits

Hasta la parte 23 donde estará la solución del ejercicio.

Ricardo Narvaja