8.0.—Referencias cruzadas y gráficos

Una de las preguntas más comunes que nos realizamos cuando estamos realizando ingeniería inversa en un binario es ¿Desde donde es llamada esta función? o ¿Qué funciones acceden a estos datos? Las referencias y los recursos a estas y a otras preguntas similares serán las que se traten de catalogar en un programa. Utilizaremos dos ejemplos para mostrar la utilidad de estos tipos de preguntas.

Consideremos un primer caso en el que hemos localizado una función la cual contiene un buffer distribuido en la pila y el cual puede desbordarse (**overflow**), y posiblemente con ello poder explotar el programa. Debido a que dicha función puede estar escondida dentro de una aplicación compleja, el siguiente paso sería determinar cómo podemos llegar a dicha función. Ya sabemos que cualquier función no nos sirve para nada si no conseguimos ejecutarla. Esto nos lleva a realizarnos la siguiente pregunta ¿Qué funciones llaman a esta función vulnerable? También nos podemos preguntar ¿Qué datos y de qué naturaleza, son pasados a la función vulnerable por las otras funciones? Estos razonamientos nos llevarán a trabajar con una cadena de llamadas que potencialmente podrán influir en hallar la solución para poder explotar el **overflow** que hemos descubierto.

Como segundo caso, consideremos un binario el cual contiene un gran número de cadenas ASCII, de las cuales al menos una la encontramos sospechosa, “Contraseña no válida” ¿La presencia de esta cadena indica que el binario nos mostrará siempre “Contraseña no válida”? No. Esto sólo indica que en dicho binario existe esta secuencia de caracteres ASCII. Este mensaje se nos podría mostrar justo después de la petición de una contraseña; por lo tanto lo que necesitamos es encontrar el código relacionado con esta cadena para verificar nuestras sospechas. La respuesta a la pregunta ¿dónde está referenciada esta cadena? Nos ayudará rápidamente a localizar la ubicación o ubicaciones en donde se utiliza dicha cadena en el programa. Una vez allí podremos verificar si existe en el código alguna forma de que dicha cadena no nos sea nunca mostrada después de la petición de una contraseña.

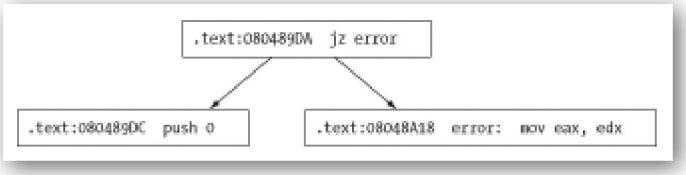
IDA nos ayudará a responder estos tipos de preguntas a través de su característica de referencias cruzadas. IDA nos proporciona varios mecanismos para mostrarnos y acceder a las referencias cruzadas de datos, incluida en estos, está la capacidad de generar un gráfico mostrándonos la relación gráfica entre código y datos. En el resto de esta parte estudiaremos los tipos de información de referencias cruzadas suministradas por IDA, las herramientas para acceder a ellas y cómo interpretar estos datos.

8.1.—Referencias cruzadas

Iniciaremos su explicación apuntando que en IDA normalmente las referencias cruzadas son mencionadas como **xrefs**. En este texto utilizaremos, **xrefs**, solamente cuando lo utilicemos para referirnos a ellas dentro de un menú o diálogo de IDA. En otros casos que no sean los dichos utilizaremos el término referencia cruzada.

En IDA existen dos tipos básicos de referencias cruzadas: referencias cruzadas de código y referencias cruzadas de datos. Dentro de cada categoría, detallaremos varios tipos de referencias cruzadas. A cada referencia cruzada existe asociada a ella una anotación de dirección. Todas las referencias cruzadas están hechas desde una dirección a otra dirección. Las direcciones tanto “desde” como “a” pueden ser tanto de código como de datos. Si estás familiarizado con la teoría gráfica de IDA, puedes considerar

directamente en el gráfico, las direcciones como nodos y las líneas flechadas como referencias cruzadas. La figura siguiente nos proporciona un recordatorio sobre la terminología gráfica. En este gráfico simple existen tres nodos, uno en cada rectángulo los cuales están conectados directamente por dos líneas flechadas.



Observemos que los nodos también pueden llamarse vértices. En las líneas flechadas directas, la disposición de la punta indica la dirección en la cual se mueve el flujo de ejecución. En la figura anterior vemos que es posible que el flujo se mueva del nodo superior a los nodos inferiores, pero no es posible que el flujo vaya de los nodos inferiores al nodo superior.

El concepto de referencia cruzada de código es muy importante, ya que gracias a ella IDA puede generar el gráfico de control de flujo y el gráfico de llamada a función, la estudiaremos en este capítulo.

Antes de acometer los detalles de las referencias cruzadas, es importante entender cómo IDA nos muestra en el listado de desensamblado las información de las referencias cruzadas. La figura siguiente muestra la línea de encabezado del desensamblado de la funcion **WndProc**, la cual contiene una referencia cruzada como un comentario normal, al lado derecho de la figura.

```
CODE:00401128 ; Attributes: bp-based frame
CODE:00401128
CODE:00401128 ; int __stdcall WndProc(HWND hWndParent, UINT Msg, WPARAM wParam, LPARAM lParam)
CODE:00401128                 public WndProc
CODE:00401128 WndProc         proc near               ; DATA XREF: start+27↑o
```

El texto **DATA XREF** indica que es una referencia cruzada de dato, mientras que **CODE XREF** indica que es una referencia cruzada de código. La dirección que sigue al texto **start+27**, en este caso, indica la dirección en la cual se ha originado la referencia cruzada. Digamos que la forma de dirección, **CODE:00401027**, es más descriptiva, que **start+27**. Aunque ambas formas representan la misma ubicación del programa, el formato utilizado por la referencia cruzada nos proporciona información adicional nos dice que esta referencia cruzada se ha hecho dentro de una función llamada **start**, específicamente **0x27 (39)** bytes dentro de la función **start**. La flecha hacia arriba o hacia abajo que siempre sigue a la dirección, indica la dirección relativa a la ubicación que se ha referenciado. Como vemos en la figura anterior la flecha hacia arriba indica que **start+27** depende de una dirección menor a **WndProc**, por lo tanto hay que ir hacia arriba del listado de desensamblado para encontrarla. Por similitud una flecha hacia abajo indicaría que la dirección referenciada sería mayor a la de **WndProc**, con lo cual tendríamos que bajar por el listado para encontrarla. Para finalizar, cualquier comentario de referencia cruzada contiene un sufijo de un carácter para identificar el tipo de la referencia cruzada. Cada sufijo lo describiremos con más detalle cuando hablemos de los tipos de referencia cruzada.

8.1.1.—Referencias cruzadas de código

Una referencia cruzada de código se utiliza para indicar que una instrucción transfiere o puede transferir el control a otra instrucción. A la forma en que las instrucciones

transfieren el control en IDA, le daremos el nombre de **flujo (flow)**. IDA distingue tres tipos de flujo: **ordinario (ordinary), salto (jump) y llamada (call)**. Los flujos de salto y llamada además, se dividen según la dirección del objetivo si es una dirección cercana o lejana. Las direcciones lejanas las podremos encontrar en los binarios que utilicen direcciones segmentadas. Para el estudio siguiente, utilizaremos el desensamblado del siguiente programa:

```
int lee_esto;                   //variable entero lee en main
int escribe_esto;               //variable entero escribe 3 veces en main
int referencia_esto;            //variable entero cuya dirección es tomada en main

void llamada_flujo ( ) { }      //función llamada dos veces desde main
int main ( )
   {
   int *p = &referencia_esto;         //efecto un “offset” estilo referencia dato
   *p = lee_esto;               //efecto un “lee” estilo referencia dato
   escribe_esto = *p;           //efecto un “escribe” estilo referencia dato
   llamada_flujo ( );           //efecto una “llamada” estilo referencia código
   if (lee_esto == 3)           //efecto un “salto” estilo referencia código
         {
         escribe_esto = 2;      //efecto un “escribe” estilo referencia dato
         }
   else                         //efecto un “salto” estilo referencia código
         {
         escribe_esto = 1;              //efecto un “escribe” estilo referencia dato
         }
         llamada_flujo;                 //efecto una “llamada” estilo referencia código
   }
```

Dicho programa contiene todas las características de referencias cruzadas de IDA, como puedes ver en los comentarios de cada línea.

Un flujo ordinario es simplemente un tipo de flujo, el cual representa el flujo secuencial desde una instrucción a otra. Es por defecto el flujo de ejecución para todas las instrucciones que no produzcan desviación de flujo como **ADD**. No existe ningún indicador especial para el flujo ordinario, aparte del orden en que están listadas las instrucciones en el desensamblado. Si una instrucción **A** tiene un flujo ordinario con la instrucción **B,** quiere decir que la instrucción **B** es la que seguirá inmediatamente en el listado de desensamblado a **A**. En el siguiente listado, cada instrucción aparte de estas dos

Tienen asociado un flujo ordinario con su inmediato sucesor:

```
.text:00401010 ; =============== S U B R O U T I N E =======================================
.text:00401010
.text:00401010 ; Attributes: bp-based frame
.text:00401010
.text:00401010 ; int __cdecl main(int argc, const char **argv, const char *envp)
.text:00401010 _main           proc near               ; CODE XREF: ___tmainCRTStartup+15A↓p
.text:00401010
.text:00401010 p               = dword ptr -4
.text:00401010 argc            = dword ptr  8
.text:00401010 argv            = dword ptr  0Ch
.text:00401010 envp            = dword ptr  10h
.text:00401010
.text:00401010                 push    ebp
.text:00401011                 mov     ebp, esp
.text:00401013                 push    ecx
.text:00401014                 mov     [ebp+p], offset referencia_esto
.text:0040101B                 mov     eax, [ebp+p]
.text:0040101E                 mov     ecx, lee_esto
.text:00401024                 mov     [eax], ecx
.text:00401026                 mov     edx, [ebp+p]
.text:00401029                 mov     eax, [edx]
.text:0040102B                 mov     escribe_esto, eax
.text:00401030                 call    llamada_flujo
.text:00401035                 cmp     lee_esto, 3
.text:0040103C                 jnz     short loc_40104A
.text:0040103E                 mov     escribe_esto, 2
.text:00401048                 jmp     short loc_401054
.text:0040104A ; ---------------------------------------------------------------------------
.text:0040104A
.text:0040104A loc_40104A:                             ; CODE XREF: _main+2C↑j
.text:0040104A                 mov     escribe_esto, 1
.text:00401054
.text:00401054 loc_401054:                             ; CODE XREF: _main+38↑j
.text:00401054                 call    llamada_flujo
.text:00401059                 xor     eax, eax
.text:0040105B                 mov     esp, ebp
.text:0040105D                 pop     ebp
.text:0040105E                 retn
.text:0040105E _main           endp
```

Las instrucciones utilizadas para invocar funciones, instrucciones x86 como **call** en,

```
.text:00401030                 call    llamada_flujo
```

```
.text:00401054                 call    llamada_flujo
```

y que están asignadas a la función **llamada_flujo**, indican una transferencia de control de flujo a la función objetivo. En la mayoría de los casos a las instrucciones **call** se les asigna un tipo de flujo ordinario, ya que la mayoría de las funciones retornan a la ubicación siguiente de la llamada. Si IDA cree que una función no retorna, lo cual se determina en la fase de análisis, entonces a estas llamadas a función no se le asignan un flujo ordinario. Los flujos de llamadas son anotados mostrándonos la referencia cruzada a la función objetivo, que es la dirección de destino del flujo. El resultado del desensamblado de la función **llamada_flujo** es el siguiente:

```
.text:00401000 ; =============== S U B R O U T I N E =======================================
.text:00401000
.text:00401000 ; Attributes: library function bp-based frame
.text:00401000
.text:00401000 llamada_flujo   proc near               ; CODE XREF: _main+20↓p
.text:00401000                                         ; _main:loc_401054↓p
.text:00401000                 push    ebp
.text:00401001                 mov     ebp, esp
.text:00401003                 pop     ebp
.text:00401004                 retn
.text:00401004 llamada_flujo   endp
.text:00401004
```

En este ejemplo se nos muestran dos referencias cruzadas de dos direcciones para **llamada_flujo**, lo cual nos indica que esta función es llamada dos veces. La dirección mostrada en la referencia cruzada es mostrada en la función llamada como un **offset** con

un nombre asociado, que en cada caso es el nombre utilizado. Pueden utilizarse ambas formas de direcciones mostradas en estas referencias cruzadas. Las referencias cruzadas consecuencia de la llamada a una función se distinguen de las otras por llevar el sufijo **p,** se utiliza p de la palabra **Procedure**.

Un flujo de salto, **jump flow**, se asigna a cada instrucción de desviación ya sea condicional o incondicional. A las desviaciones condicionales también se les asigna un flujo ordinario para poder explicar el flujo de control cuando la desviación no se produce. Las desviaciones incondicionales no tienen asociado el flujo ordinario ya que la desviación siempre se produce en estos casos. La línea de guiones, figura abajo, es un

```
.text:0040104A ; ---------------------------------------------------------------------------
```

dispositivo que se utiliza para indicarnos que no existe en flujo ordinario entre las dos instrucciones adyacentes a la línea. Los flujos de salto están asociados con las referencias cruzadas estilo **jump** mostrando el objetivo del salto, ver siguiente figura,

```
.text:0040104A loc_40104A:                             ; CODE XREF: _main+2C↑j
.text:0040104A                 mov     escribe_esto, 1
.text:00401054
.text:00401054 loc_401054:                             ; CODE XREF: _main+38↑j
```

al igual que las referencias cruzadas de llamada, las referencias cruzadas de salto, muestran la dirección de la ubicación referida, la fuente del salto. Las referencias cruzadas de salto se distinguen de las demás por utilizar el sufijo **j**, de la palabra **jump**.

8.1.2.—Referencias cruzadas de datos

Las referencias cruzadas de datos se utilizan para seguir la pista a la forma en que los datos son accedidos dentro de un binario. Las referencias cruzadas de datos pueden estar asociadas con cualquier byte, en una base de datos de IDA, el cual esté asociado con una dirección virtual. En otras palabras, las referencias cruzadas de datos nunca están asociadas con las variables de pila. Los tres tipos de referencias cruzadas de datos utilizados normalmente se usan para indicar cuando una ubicación se está **leyendo**, cuando una ubicación se está **escribiendo** y cuando una dirección de una ubicación se está **tomando**. Las variables globales asociadas al ejemplo del listado de desensamblado previo se muestran a continuación, ya que ellas nos proporcionan varios ejemplos de referencias cruzadas de datos.

```
.data:0040B720 lee_esto        dd ?                    ; DATA XREF: _main+E↑r
.data:0040B720                                         ; _main+25↑r
.data:0040B724 escribe_esto    dd ?                    ; DATA XREF: _main+1B↑w
.data:0040B724                                         ; _main+2E↑w ...
.data:0040B728 referencia_esto db    ? ;               ; DATA XREF: _main+4↑o
.data:0040B729                 db    ? ;
.data:0040B72A                 db    ? ;
.data:0040B72B                 db    ? ;
```

Una **referencia cruzada de lectura** se utiliza para indicarnos que el contenidos de una ubicación de memoria está siendo accedida. Las referencias cruzadas de lectura pueden originarse solamente desde una dirección de instrucción, pero puede referirse a cualquier ubicación del programa. La variable local **lee_esto** es leída en las ubicaciones.

```
.text:0040101E                 mov     ecx, lee_esto
```

```
.text:00401035                 cmp     lee_esto, 3
```

Los comentarios asociados a las referencias cruzadas mostradas a continuación, indican

```
.data:0040B720 lee_esto        dd ?                    ; DATA XREF: _main+E↑r
.data:0040B720                                         ; _main+25↑r
```

en qué ubicaciones dentro de **main** se han referenciado a **lee_esto** y además se pueden reconocer que son referencias cruzadas de lectura por el sufijo **r**, de la palabra **read**. La primera lectura ejecutada a **lee_esto** es una lectura de 32 bit la cual se coloca en el registro **ECX**, lo que induce a IDA a formatear a **lee_esto** como un **dword (dd)**. Por regla general IDA toma nota de cualquier señal posible para poder determinar el tamaño y/o tipo de las variables basándose en cómo son accedidas y cómo son utilizadas como parámetros a las funciones.

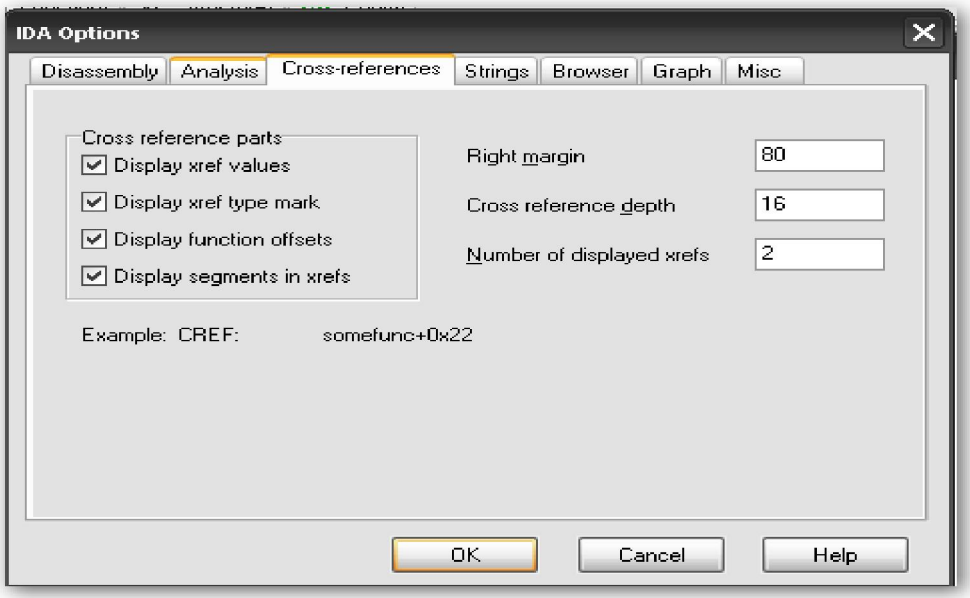
La variable global **escribe_esto** se referencia en las siguientes ubicaciones del ejemplo.

```
.text:0040102B                  mov     escribe_esto, eax

.text:0040103E                  mov     escribe_esto, 2

.text:0040104A                  mov     escribe_esto, 1
```

Las **referencias cruzadas de escritura** generadas y mostradas como comentarios de la variable **escribe_esto**, mostradas a continuación, indican las ubicaciones del programa en donde se modifica el contendido de la variable.

```
.data:0040B724 escribe_esto    dd ?                    ; DATA XREF: _main+1B↑w
.data:0040B724                                         ; _main+2E↑w ...
```

Las referencias cruzadas de escritura utilizan el sufijo **w**, de la palabra **write**. En este caso otra vez IDA ha determinado el tamaño de la variable basándose en los 32 bit del registro **EAX** el cual es copiado a **escribe_esto**. Observemos que la lista de referencias cruzadas mostradas de **escribe_esto** termina con **puntos suspensivos**, figura arriba, lo cual indica que las referencias cruzadas a **escribe_esto** exceden el actual límite de referencias cruzadas. Este límite lo podemos modificar a través de la opción **Number of displayed xrefs** de la solapa **Cross-references** en el **Dialog general** del menú **Options**.

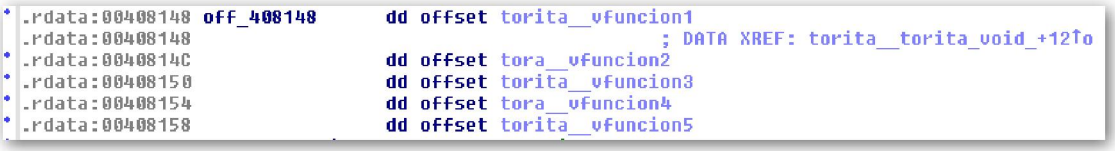


Esto es debido a que tanto las referencias cruzadas de lectura como las referencias cruzadas de escritura sólo se pueden generar desde una instrucción, pero pueden referenciar cualquier ubicación del programa. En términos generales, una referencia cruzada de escritura que tenga como objetivo un byte de una instrucción del programa, es indicativo de auto modificación de código, lo cual normalmente se considera negativo ya que frecuentemente se encuentra en las rutinas de ofuscación utilizadas en el malware.

El tercer tipo de referencia cruzada de dato, es la **referencia cruzada offset**, indica la dirección de una ubicación que está siendo utilizada en vez del contenido de la ubicación. La dirección de la variable global **referencia_esto** es tomada en:

Observemos la referencia cruzada offset del listado ejemplo y veamos que contiene un sufijo **o**, de la palabra **offset**. Normalmente las referencias cruzadas offset son el resultado de operaciones con punteros tanto con código como con datos. Las operaciones de acceso a conjuntos, por ejemplo, son ejecutadas añadiendo un offset a la dirección de inicio del conjunto. Como consecuencia la primera dirección, en la mayoría de conjuntos globales, puede reconocerse por la presencia de una referencia cruzada offset. Por esta razón, la mayoría de cadenas de datos, cadenas que son conjuntos de caracteres en C/C++, es el objetivo de las referencias cruzadas offset.

A diferencia de las referencias cruzadas de lectura y escritura, las cuales sólo se pueden originar en ubicaciones de instrucción, las referencias cruzadas offset pueden originarse de unas a otras tanto en ubicaciones de instrucción como en ubicaciones de datos. Un ejemplo de un offset que puede originarse desde la sección de datos de un programa es cualquier tabla de punteros, tal como una **vtable**, lo cual da como efecto la generación de una referencia cruzada offset desde cada ubicación dentro de la tabla a la ubicación que están apuntando dichas ubicaciones. Esto lo podemos apreciar si examinamos la **vtable** de la clase **torita**, la cual vimos en parte 7, su desensamblado sería el siguiente:



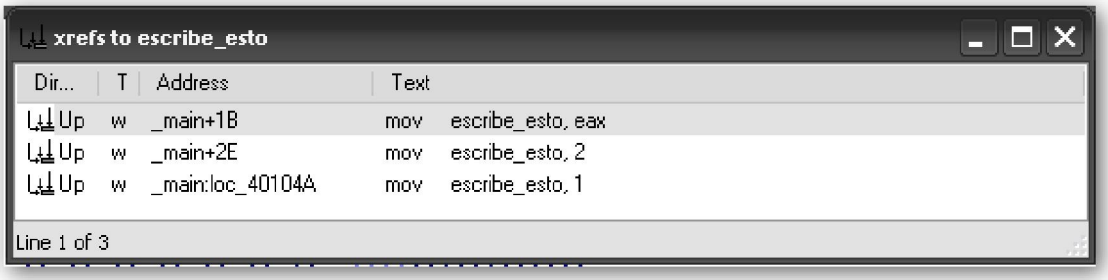
Vemos las direcciones de la **vtable** utilizada en la función **torita::torita(void)** la cual es la clase constructor. La línea de encabezado de la función **torita__vfuncion3**, línea abajo, muestra la referencia cruzada offset la cual enlaza la función con la vtable.

Con este ejemplo se demuestra que las funciones virtuales en C++ nunca son llamadas directamente y nunca son objetivo de una referencia cruzada de llamada. Por el contrario, todas las funciones virtuales de C++ deben de ser referidas al menos por una entrada de la **vtable** y siempre deberá tener al menos una referencia cruzada offset. Recordemos que no es obligado supeditarnos a una función virtual. Por lo tanto, una función virtual puede aparecer en más de una **vtable**, como vimos en la parte 7. El resultado de este desarrollo es; que seguir el rastro hacia atrás de las referencias cruzadas offset es una técnica para localizar fácilmente **vtables** C++ en la sección de datos de un programa.

8.1.3.—Listado de referencias cruzadas

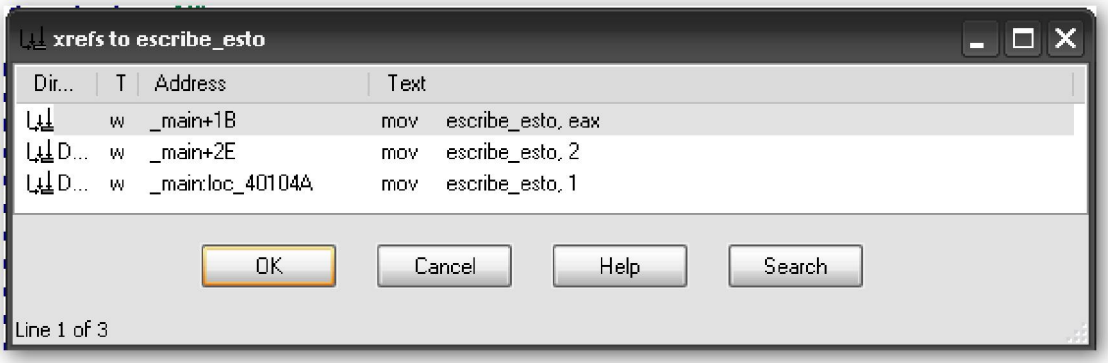
Una vez que ya hemos comprendido que son las referencias cruzadas, podemos ahora aprender de qué forma podemos acceder a todos estos datos en IDA. Como ya mencionamos, el número de comentarios de referencias cruzadas que pueden ser mostrados con respecto a una ubicación está limitado por la configuración por defecto a **2**. Debido a que el número de referencias cruzadas a una ubicación no excederá este límite, para poder trabajar con todas las que contenga es muy fácil. Pasamos el ratón por encima del texto de la referencia cruzada mostrada en el desensamblado con lo cual se nos muestra una herramienta del tipo tool-tip, hacemos doble click en la dirección de salto de la referencia cruzada y en la ventana del desensamblado tendremos la fuente de la referencia cruzada.

Existen dos métodos para ver la lista completa de referencias cruzadas a una ubicación. El primer método es abrir una vista de las referencias cruzadas asociadas a una dirección específica. Lo podemos realizar posicionando el cursor en la dirección deseada la cual es el objetivo de una o más referencias cruzadas y realizaremos la siguiente acción **View > Open Subviews > Cross-References**, con lo cual se nos abrirá un listado completo de las referencias cruzadas de la ubicación deseada. La figura abajo, te muestra el listado completo de las referencias cruzadas a la variable **escribe_esto**.



Las columnas de la ventana indican; **Direction**, la dirección arriba o abajo a la fuente de la referencia cruzada, **T..** el tipo de la referencia cruzada usando los sufijos de tipo explicados anteriormente, **Address** la dirección de la fuente de la referencia cruzada y **Text** el texto correspondiente del desensamblado de la dirección fuente incluido cualquier comentario que exista en la dirección fuente. Al igual que en otras ventanas que muestran un listado de direcciones, si hacemos doble click en cualquier entrada la vista del desensamblado se reposiciona en la vista de la dirección fuente correspondiente. Una vez abierta, la ventana referencias cruzadas permanecerá abierta y se podrá acceder a ella a través de la solapa correspondiente encima del desensamblado.

La segunda forma de acceder al listado de referencias cruzadas es haciendo click derecho en un nombre (name) del cual queramos saber sus referencias cruzadas y eligiremos **Jump to xref** o utilizaremos el atajo **X** para abrir el diálogo de listado de cualquier ubicación que haga referencia al símbolo seleccionado. El diálogo que se nos muestra es idéntico en apariencia al de la vista anterior de referencias cruzadas. Pero en este caso podemos activarlo con el atajo **X** seleccionando **escribe_esto** en la dirección **.text:0040102B**.

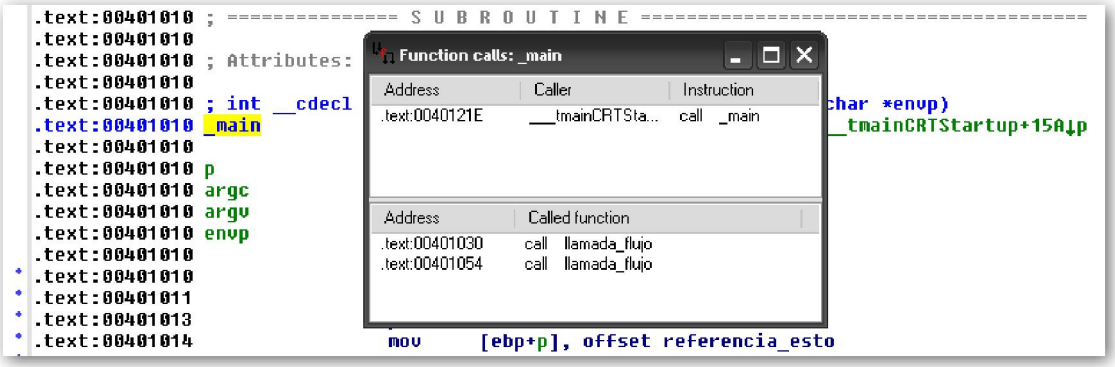


La principal diferencia entre las dos vistas es su comportamiento. La vista, figura anterior, es un diálogo modal (Un diálogo modal es aquel que debe cerrarse antes de que continuemos con la interacción de la aplicación. Los diálogos de modo pueden permanecer abiertos mientras continuamos con la interacción con la aplicación) tiene botones para interactuar y cerrar el diálogo. El propósito principal de este diálogo es seleccionar una ubicación de referencia y saltar a ella. Si hacemos doble click en una de las ubicaciones listadas nos cierra el diálogo y la vista del desensamblado se posiciona en dicha ubicación. Otra diferencia entre los dos diálogos es la forma de poderlos abrir, el segundo podemos abrirlo utilizando el atajo X o con el menú contextual desde cualquier símbolo de un procedimiento, el primero sólo puede ser abierto posicionando el cursor en una dirección la cual es el objetivo de la referencia cruzada y realizando la acción **View > Open Subview > Cross-References**. Otra forma de decirlo es que el diálogo se puede abrir desde la fuente de cualquier referencia cruzada, mientras que la subventana puede ser abierta solamente desde el destino de la referencia cruzada.

Un ejemplo sobre la utilidad de los listados de referencias cruzadas podría ser, localizar rápidamente cada ubicación en las que una función, en particular, es llamada. Veamos un ejemplo, muchas personas consideran el uso de la función C **strcpy**, (La función de C strcpy copia un conjunto fuente de caracteres, incluido un carácter null al final, destinados a un conjunto sin verificar que dicho conjunto destino sea lo suficientemente grande para contener todos los caracteres del conjunto fuente), como peligrosa. Utilizando las referencias cruzadas, podemos localizar cualquier llamada a **strcpy** simplemente hallando una llamada a **strcpy**, y luego utilizando el atajo **X** nos aparecerá el diálogo de referencias cruzadas, y podremos trabajar a través de cualquier referencia cruzada a ella. Si no quieres perder tiempo hallando **strcpy** en el binario, puedes usar el truco de añadir al desensamblado un comentario con el texto **strcpy** y activar el diálogo de referencias cruzadas utilizando dicho comentario. Cuando un nombre de símbolo aparece en un comentario, IDA trata a este símbolo al igual que un operando de una instrucción de desensamblado. Haciendo doble click encima del símbolo nos reposiciona la ventana del desensamblado en él, y haciendo click derecho nos proporciona el menú de contexto.

8.1.4.—Llamadas de función

Un listado de referencias cruzadas especial es el que trata exclusivamente con las llamadas de función al cual podemos acceder realizando la acción **View > Open Subviews > Function Calls** lo cual nos proporciona el diálogo siguiente, el cual nos lista todas las ubicaciones en donde es llamada la función actual, la cual se define con la colocación del cursor en el momento de abrir dicho diálogo. En la parte superior de la ventana se listan todas las llamadas hechas a la función y todas las llamadas realizadas por la función en la mitad inferior de la ventana.



En esta ventana también, podemos utilizar cada referencia cruzada listada para reposicionar el desensamblado en la ubicación correspondiente de dicha referencia. Resumiendo las referencias cruzadas de llamadas nos permiten considerar las relaciones abstractas entre una dirección u otra sin tener en cuenta la relación existente entre ellas como funciones.

Performance Bigundill@