14.5.3.—Enumerar referencias cruzadas

Iterar a través de referencias cruzadas puede crear confusión debido al número de funciones que tienen acceso a las referencias cruzadas de dato y de hecho a las referencias cruzadas de código las cuales son bidireccionales. A fin de obtener los datos que deseamos, debemos asegurarnos de que estamos accediendo al tipo apropiado de referencia cruzada. En este primer ejemplo de script sobre referencias cruzadas, obtendremos el listado de todas las funciones que son llamadas dentro de la función, esta acción la realizaremos gracias a la iteración a través de cada instrucción en la función seleccionada, para determinar si la instrucción es una llamada a otra función. Un método para realizar esto podría ser analizar los resultados de búsqueda de instrucciones **call** por **GetMnem**. Esta solución, no obstante, no es muy "portable" ya que la forma de escribir la instrucción de llamada a una función puede variar entre tipos de CPU. En segundo lugar, podríamos querer realizar un segundo análisis para determinar exactamente a la función que se está llamando. Debido a dichos inconvenientes utilizaremos las referencias cruzadas que evitan cada uno de los problemas apuntados, ya que ellas son independientes del tipo de CPU y nos informan directamente del objetivo que tiene la referencia cruzada.

Script enumerar referencias cruzadas que referencia llamadas desde una función.

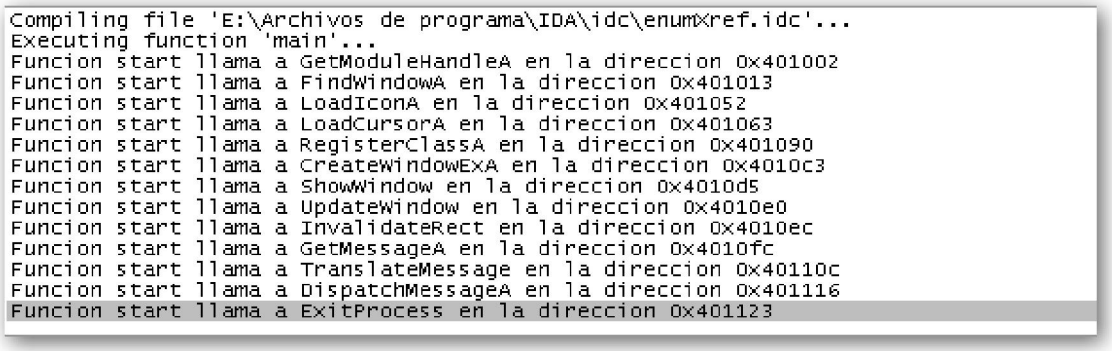
```
#include <idc.idc>
static main ( )  {
   auto funcion, final, objetivo, instruccion, nombre, banderas, refcruzada;
   banderas = SEARCH_DOWN | SEARCH_NEXT;
   funcion = GetFunctionAttr (ScreenEA ( ), FUNCATTR_START);
   if (funcion != -1)   {
            nombre = Name (funcion);
            final = GetFunctionAttr (funcion, FUNCATTR_END);
            for (instrucción = funcion; instruccion < final; instruccion = FindCode (instruccion, banderas))  {
              for (objetivo = Rfirst (instruccion); objetivo != BADADDR; objetivo = Rnext (instruccion, objetivo))  {
                 refcruzada = XrefType ( );
                 if (xref == fl_CN | | xref == fl_CF)   {
                       Message ("Funcion %s llama a %s en la direccion 0x%x\n", nombre, Name (objetivo), instruccion);
                 }
             }
         }
    }
   else   {
      Warning ("No existe funcion en la ubicacion %x", ScreenEA ( ) );
   }
}
```

En este ejemplo, deberemos iterar a través de cada instrucción en la función. Para cada instrucción, deberemos iterar a través de cada referencia cruzada a la función. Como sólo nos interesan las referencias cruzadas que referencia a una llamada de otra función, deberemos verificar que el valor de retorno de **XrefType** corresponda a la búsqueda de referencias cruzadas de tipo **fl_CN** o **Fl_CF**.

Otra posible utilización de las referencias cruzadas es determinar cualquier ubicación que proporcione referencias a una particular ubicación. Si quisiéramos crear un analizador de seguridad sencillo, podría ser interesante remarcar todas las llamadas realizadas a funciones como **strcpy** y **sprintf**. En el "kit-kat teórico" se explica el por qué de estas dos funciones.

Como siempre, cogeremos nuestro script al cual llamaremos **enumXref.idc** y lo colocaremos en la carpeta **idc**, seguidamente podemos cargar nuestro querido **CRACKME.EXE** y con la acción **File > IDC file** ejecutamos el script **enumXref** sobre

la función **start** y si nos fijamos en la ventana de mensajes de IDA podremos ver la siguiente información.



Como podemos observar nos proporciona la relación de todas las llamadas a otras funciones realizadas por la función **start** y la correspondiente dirección en donde se produce dicha llamada.

Kit-kat teórico: Funciones "peligrosas"
Las funciones de C **strcpy** y **sprintf** generalmente son reconocidas como de uso peligroso debido a que permiten la copia ilimitada de datos en su buffer de destino. Cada una de ellas puede ser utilizada siempre y cuando se realice una verificación del tamaño de los buffers tanto fuente como destino, cosa que normalmente se olvidan la mayoría de programadores. La función **strcpy**, por ejemplo, es declarada como sigue:

char *strcpy (char *dest, const char *source);

La definición del comportamiento de la función **strcpy**, es copiar todos los caracteres incluido el carácter **null** final, del buffer fuente al buffer de destino dado **dest**. El problema fundamental es que no existe forma para determinar, en ejecución, el tamaño de cualquier conjunto (array). En este caso, **strcpy** no tiene ningún medio para determinar si el tamaño del buffer destino será suficiente grande como para almacenar todos los datos que serán copiados del buffer fuente. Dichas operaciones de copia sin verificación de tamaños son causa de vulnerabilidades tipo "**buffer overflow**".

Performance Bigundill@