7.0.—Tipos de datos y estructuras de datos (Introducción)

Para que resulte fácil comprender el comportamiento de los binarios, el secreto estriba en catalogar las funciones de librería que el programa llama. Por ejemplo un programa en C llama a la función **connect** para crear una conexión de network. Por otra parte un programa de Windows llama a **RegOpenKey** para acceder al registro de Windows. Por lo tanto para llegar a comprender cómo y porqué estas funciones son llamadas necesitaremos analizarlo.

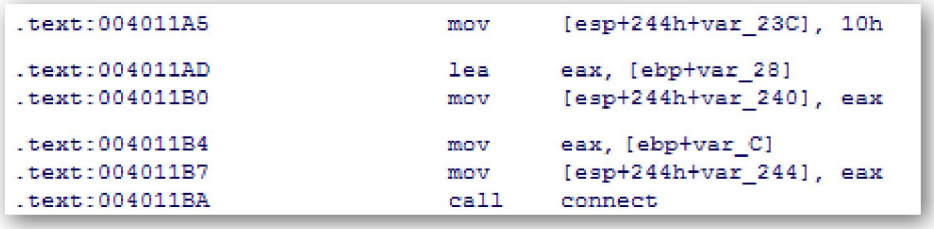
Para descubrir cómo es llamada una función, se requiere aprender qué parámetros serán pasados a la función. Volvamos a nuestro ejemplo de llamada **connect**, más que el hecho de que se esté llamando a la función, lo más importante es conocer exactamente que dirección del programa está vinculada a ella. Además comprender los datos que se están pasando a las funciones es la clave para revertir la firma de una función; número, tipo y secuencia de los parámetros requeridos por una función. También hay que señalar la importancia que conlleva comprender cómo son manipulados los tipos de datos y las estructuras de datos a nivel de lenguaje ensamblador.

En esta parte estudiaremos cómo son guardadas las estructuras de datos en memoria y cómo se accede a los datos que se encuentran dentro de dichas estructuras. El método básico para asociar un tipo de dato específico con una variable, es observar cómo se utiliza dicha variable, como parámetro, por una función de la cual nosotros sepamos algo de ella. Durante la fase de análisis, IDA hace lo posible para anotar todos los tipos de datos posibles los cuales puedan deducirse basándose en la utilización de una variable en una función de la cual IDA tenga su prototipo. Siempre que sea posible IDA utilizará el nombre de un parámetro formal respecto a una función prototipo antes que generar por defecto un **dummy name** para la variable. Esto lo podemos ver en el siguiente desensamblado de una llamada a **connect**:

```
.text:004010F3                  push    10h             ; namelen
.text:004010F5                  lea     ecx, [ebp+name]
.text:004010F8                  push    ecx             ; name
.text:004010F9                  mov     edx, [ebp+s]
.text:004010FF                  push    edx             ; s
.text:00401100                  call    connect
```

En el listado podemos ver como cada **push** contiene un comentario, tomados de la lista de funciones prototipo de IDA, indicando el parámetro que será empujado a la pila. Además las dos variables de pila locales, **[ebp + name] y [ebp + s]**, han sido nombradas con los parámetros que les corresponden. En la mayoría de los casos estos nombres proporcionarán más información que los **dummy names** que IDA hubiera generado.

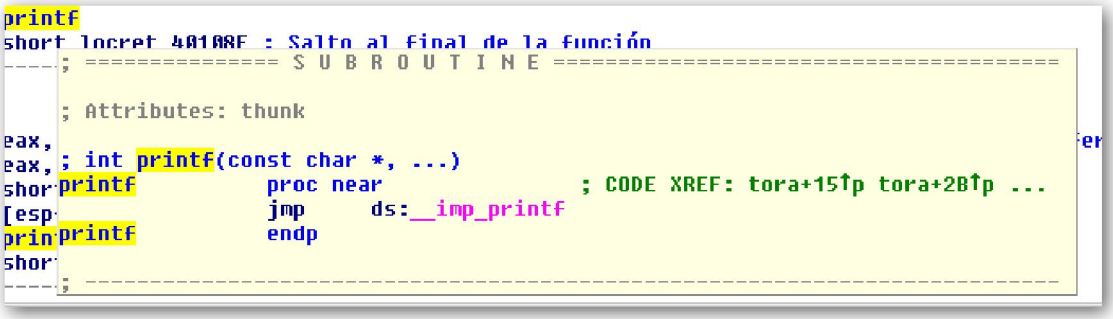
Un defecto que tiene IDA, es que sólo reconoce los parámetros de una función que son colocados en la pila por declaraciones **push**, como hemos visto antes. En el siguiente listado, los parámetros para **connect** son colocados en la pila utilizando declaraciones del tipo **mov**:



En este caso no solamente no nos anota los parámetros que son colocados en la pila, sino que además los nombres de las variables de pila, **[ebp + var_28] y [ebp + var_C]**, no concuerdan con los parámetros que representan; **var_28** corresponde a **name** mientras que **var_C** corresponde a **s**.

Cuando esto ocurra, tendrás que determinar los tipos de dato, para la función librería, que serán llamados. En muchos casos IDA conocerá el prototipo de la función, con lo cual colocando el ratón encima de su nombre se nos mostrará la vista de ésta.

Observación:"Colocar el ratón encima de cualquier nombre en la vista de IDA, produce una ventana emergente en donde se nos muestra diez líneas de desensamblado de donde está ubicado el objetivo. En el caso de nombres de función librería, a menudo incluye el prototipo de la función librería llamada".



Cuando IDA no tiene la secuencia de parámetros de una función, en su base de datos, tu mejor recurso para aprender el comportamiento de la función será la documentación **API** disponible, como **MSDN** online. Cuando todo lo anterior falle, acuérdate, **Google** es tú amigo.

En el resto de esta parte, estudiaremos: cómo reconocer qué estructuras de datos se están utilizando en un programa, cómo descifrar el esquema organigrama de dicha estructura y cómo mejorar el desensamblado cuando se utilicen dichas estructuras.

Performance Bigundill@