15.2.2.—Netnodes

Muchas de las API de IDA están construidas con base a distintas clases de C++, esto modela distintos aspectos del desensamblado de un binario. Por otro lado, la clase **netnode**, está envuelta de misterio debido a que parece no tener ninguna relación directa con los constructores (secciones, funciones, instrucciones, etc.) dentro de un binario.

Los **netnodes** son el mecanismo de almacenamiento de nivel más bajo accesibles dentro de una base de datos de IDA. Como programadores de módulos, raramente necesitaremos trabajar directamente con netnodes. Muchas de las estructuras de datos de nivel más alto, ocultan el hecho de que finalmente dependen de estos para realizar un almacenamiento persistente dentro de una base de datos. Algunas de las formas en que ellos son utilizados dentro de una base de datos, están detalladas en el archivo **nalt.hpp**, del cual aprenderemos por ejemplo, que la información sobre las librerías compartidas y funciones que un binario importa se almacena en un netnode llamado **import_mode**, perplejo, sí los netnodes pueden nombrarse. Estos son también los mecanismos que facilitan el almacenamiento persistente a los conjuntos globales IDC.

Los netnodes se describen con mucho detalle en el archivo **netnode.hpp**. Desde una perspectiva de alto nivel, los netnodes son estructuras utilizadas internamente por IDA para distintos propósitos. Sin embargo su estructura precisa, se mantiene oculta, incluso para los programadores de SDK. Para proporcionar un enlace entre estas estructuras de almacenamiento, el SDK define una clase **netnode**, la cual funciona opacamente alrededor de una estructura de memoria interna. La clase **netnode** contiene un sólo elemento dato llamado **netnodenumber**, el cual es un identificador entero utilizado para acceder a la representación interna de un netnode. Cualquier netnode es identificado únicamente por su netnodenumber. En un sistema de 32-bit el netnodenumber es de un tamaño de 32-bit, permitiendo 2**32 netnodes únicos. En un sistema de 64-bit, un netnodenumber es un entero de 64-bit, lo cual permite 2**64 netnodes únicos. En la mayoría de los casos el netnodenumber representa una dirección virtual dentro de la base de datos, que crea un mapeado natural entre cada dirección dentro de la base de datos y por lo tanto cualquier netnode puede ser requerido para almacenar información asociada con una dirección específica. El texto de un comentario es un ejemplo de información arbitraria que puede asociarse con una dirección y así guardarse dentro de un netnode asociado con la dirección.

La forma recomendada para manipularlos es invocar las funciones de la clase **netnode** utilizando un objeto netnode. Leyendo el archivo **netnode.hpp**, podremos conocer distintas funciones que no pertenecen a la clase netnode y que soportan la manipulación de netnodes. La utilización de dichas funciones no va en detrimento de las funciones pertenecientes a la clase. Observemos sin embargo, que la mayor parte de las funciones que pertenecen a dicha clase giran alrededor de una función no perteneciente a esta.

Internamente, los netnodes pueden ser utilizados para almacenar distintos tipos de información. Cada uno puede asociarse a un nombre de hasta 512 caracteres y a un valor de hasta 1.024 byte. Las funciones pertenecientes a la clase **netnode** se pueden recuperar con **name** o modificar su nombre con **rename**. Otras funciones permiten manejar el valor principal del netnode como un entero (**set_long, long_value**), como una cadena (**set, valstr**) o como un **binary blob** (Blob, es un término a menudo utilizado para referirse a un dato binario de tamaño variable) (**set, valobj**). La función utilizada determina intrínsecamente cómo es tratado el valor principal.

Aquí es donde se complican las cosas. Además de un nombre y un valor principal, cada netnode es también capaz de almacenar 256 conjuntos en los que los elementos pueden clasificarse por tamaños arbitrarios con valores de hasta un máximo de 1.024 byte cada uno. Estos conjuntos pueden agruparse en tres categorías. La primera utiliza conjuntos de valores indexados a **32-bit** que potencialmente pueden sobrepasar los 4 billones de elementos. La segunda utiliza conjuntos de valores indexados a **8-bit** que pueden sobrepasar los 256 elementos. Y la última categoría son en realidad tablas **hash** los cuales utilizan cadenas como claves. Sea cual sea la categoría utilizada, cada elemento del conjunto aceptará valores de una tamaño de hasta 1.024 byte. En resumen, un netnode puede contener una tremenda cantidad de datos, ahora necesitamos aprender que es lo que hace con todos ellos. Si te estás preguntando dónde está guardada toda esta información, no eres el único. En una base de datos de IDA, todo el contenido del netnode es guardado dentro de nodos **btree** (estructura de datos en árbol). Los nodos **Btree** a su vez son guardados en un archivo **ID0**, el cual a su vez es archivado dentro de un archivo **IDB** cuando cerramos la base de datos. Cualquier contenido de un netnode que hayamos creado no será visible en ninguna ventana de IDA; los datos son nuestros para manipularlos a nuestro gusto. Es por esto que los netnodes son un lugar ideal para colocar un almacenamiento persistente de datos para cualquier plug-in y script que pueda querer almacenar resultados para utilizarlos en una próxima invocación de uno de ellos.

15.2.2.1.—Crear netnodes

Un punto desconcertante respecto a los netnodes es que declarar un variable netnode dentro de uno de nuestros módulos, no crea necesariamente una representación interna de ese netnode dentro de la base de datos. Este no es creado internamente mientras uno de los siguientes eventos no tenga lugar:

** Se le asigne un nombre
** Se le asigne un valor primario
** Un valor sea almacenado dentro de uno de los conjuntos internos del netnode.

Estos eventos, son los tres constructores permitidos para declarar netnodes dentro de nuestros módulos. A continuación mostramos los prototipos para cada uno, extraídos de **netnode.hpp**, y ejemplos de su uso:

Ejemplo de declaración de netnodes

```
#ifdef __EA64__
typedef ulonglong nodeidx_t;
#else
typedef ulong nodeidx_t;
#endif
class netnode {
        netnode ( );
        netnode (nodeidx_t num);
        netnode (const char *name, size_t namlen=0, bool do_create=false);
        bool create (const char *name, size_t namlen=0);
        bool create ( );
   // … sigue el resto de la clase netnode
};
netnode n0;                     //uses
netnode n1 (0x00401110);        //uses
netnode n2 ("$ node 2");        //uses
```

netnode n3 (“$ node 3”, 0, true); //uses

En este ejemplo sólo un netnode (**n3**) tiene la garantía de existencia dentro de la base de datos una vez que el código haya sido ejecutado. Los netnodes (**n1**) y (**n2**) podrán existir si previamente son creados y llenados con datos. El netnode (**n1**), exista o no previamente está capacitado en este punto para recibir datos. Por otro lado si (**n2**) no existía, significa que ningún netnode nombrado **$ node 2** puede encontrarse en la base de datos, por lo tanto (**n2**) debe ser explícitamente creado en **bool create (const char *name, size_t namlen=0);** o en **bool create ( );** antes de que puedan ser guardados datos en él. Además si queremos garantizar que podemos almacenar datos en (**n2**), necesitaremos añadir la siguiente verificación de seguridad:

```
if (BADNODE == (nodeidx_t)n2)   {
   n2.create (“$ node 2”);
}
```

El ejemplo anterior muestra la utilización del operador **nodeidx_t**, el cual permite a un netnode formar un **nodeidx_t**. El operador nodeidx_t simplemente retorna el dato **netnodenumber** del netnode asociado y permite que las variables netnode sean convertidas fácilmente en enteros.

Un punto importante que tiene que quedar claro con respecto a los netnodes, es que un netnode debe de tener un **netnodenumber** válido antes de poder almacenar datos dentro de él. Un netnodenumber puede ser asignado explícitamente como se ha hecho en (**n1**) a través de un constructor, como podemos ver en el ejemplo anterior en

```
netnode (nodeidx_t num);
netnode n1 (0x00401110);          //uses
```

Como alternativa, un netnodenumber puede ser generado internamente cuando es creado un netnode utilizando la bandera **create** en un constructor, de la forma que se ha creado (**n3**)

```
netnode (const char *name, size_t namlen=0, bool do_create=false);
netnode n3 (“$ node 3”, 0, true); //uses
```

o de la forma función create como se ha realizado para (n2)

```
netnode n2 (“$ node 2”);          //uses
```

Internamente los netnodenumber asignados empiezan en **0xFF000000** y se incrementa con cada netnode nuevo creado.

Como hemos podido observar hemos dejado al netnode (**n0**) del ejemplo olvidado. En este momento (**n0**) no tiene ni número ni nombre. Podemos crear el nombre (**n0**) con la función **create** de forma similar a (**n2)**. Podemos utilizar la forma alternativa de **create** para crear un **netnode** sin nombre pero con un **netnodenumber** válido generado internamente como podemos ver seguidamente:

```
n0.create ( );      //asigna un netnodenumber a n0 generado internamente
```

Ahora sí es posible almacenar datos dentro de (**n0**), sin embargo no podremos recuperar dichos datos en otra ocasión, a menos que registremos en alguna parte el **netnodenumber** asignado o asignemos un **nombre** a (n0). De hecho, ahora podemos

darnos cuenta lo fácil que es acceder a los netnodes cuando están asociados con una dirección virtual, similar al (**n1**) de nuestro ejemplo. Para todos los demás netnodes, asignarles un nombre hace posible realizar una búsqueda por nombre para futuras referencias a ellos, como en (**n2**) y (**n3**) de nuestro ejemplo.

Observemos que en todos nuestros netnodes nombrados, hemos utilizado nombres con el signo **$** antepuesto a ellos, esta es una práctica recomendada en **netnode.hpp**, para evitar conflictos con nombres utilizados internamente por IDA.

15.2.2.2.—Almacenar datos en netnodes

Una vez que hemos aprendido cómo crear un netnode para poder almacenar datos dentro de él, retomemos la explicación de la capacidad que tienen para almacenar internamente conjuntos. Para guardar un valor en un conjunto dentro de un netnode, necesitamos especificar cinco valores de información: **un valor, un tamaño (8 o 32 bit), un valor a guardar, el número de byte del valor y un conjunto (uno de los 256 disponibles por cada categoría de conjunto) en el cual almacenar el valor**. El parámetro del tamaño es especificado implícitamente por la función que se utilizará para almacenar o recuperar los datos. Los valores restantes son pasados a la función como parámetros. El parámetro que escoge uno de los 256 conjuntos posibles es un valor normalmente almacenado con el nombre **tag** y éste se especifica (no obligadamente) usando un carácter. La documentación de netnode muestra unos cuantos tipos especiales de términos del valor **altvals, supvals y hashvals**. Por defecto, cada uno de dichos valores es normalmente asociado con un **tag** específico al conjunto: **'A' para altval, 'S' para supval y 'H' para hashval**. Un cuarto tipo de valor es el llamado **charval**, el cual no está asociado a ningún tag de conjunto específico.

Es importante comprender que estos tipos de valor están asociados más a una forma específica de almacenar datos en un netnode que con un conjunto específico dentro de un netnode. Es posible almacenar cualquier tipo de valor en cualquier conjunto simplemente especificando un **tag** de conjunto alternativo cuando guardamos datos. En todos los casos, recordemos que el tipo de dato almacenado en una ubicación particular de un conjunto puede utilizar métodos de recuperación apropiados al tipo de dato almacenado.

Altval proporciona un sencillo enlace para almacenar y recuperar datos enteros en un netnode. Altval puede ser almacenado en cualquier conjunto que exista dentro de un netnode, pero por defecto se guarda en el conjunto **'A'**. A pesar de que podemos almacenar enteros en cualquier array deseado, utilizar las funciones relacionadas con altval simplifica mucho la tarea. El siguiente ejemplo muestra como guardar y recuperar datos utilizando altval.

Acceso a netnode con altval:

```
netnode n ("$ tora", 0, true);      //crea el netnode si no existe
sval_t  indice = 1000;              //sval_t es tipo 32-bit, este ejemplo usa índices 32 bit
ulong valor = 0x12345678;
n.altset (indice, valor);           //almacena valor en el conjunto 'A', en el índice indicado
valor = n.altval (indice);          //recupera valor del conjunto 'A' en el índice indicado
n.altset (indice, valor, (char)3);  //ahora lo guarda en el conjunto 3
valor = n.altval (indice, (char)3); // ahora lo recupera del conjunto 3
```

En este ejemplo, vemos un modelo que será repetido por otros tipos de valores netnode, la utilización de una función tipo **XXXset**, en este caso **altset,** almacena un valor dentro de un netnode y la utilización de una función tipo **XXXval**, en este caso **altval**, recupera un valor en un netnode. Si lo que queremos es almacenar valores enteros dentro de un conjunto utilizando valores con indices de 8-bit, necesitaremos utilizar funciones un poco distintas, como podemos ver en el siguiente ejemplo:

```
netnode n ("$ tora", 0, true);
uchar indice = 80;                      //Este ejemplo utiliza indices 8-bit
ulong valor = 0x87654321;
n.altset_idx8 (indice, valor, 'A');     //almacena, con funciones XXX_idx8
valor = n.altval_idx8 (indice, 'A');    //recupera con funciones XXX_idx8
n.altset_idx8 (indice, valor, (char)3); // almacena en el conjunto 3
valor = n.altval_idx8 (indice, (char)3); //recupera del conjunto 3
```

Vemos que el método general para utilizar indices 8-bit es utilizar una función con el sufijo **_idx8**. También observemos que ninguna de las funciones **_idx8** suple el parámetro **tag** por defecto del conjunto.

Supval representa el medio más versátil de almacenar y recuperar datos en los netnode. Supval representa a datos de tamaño arbitrario, desde 1 byte hasta el máximo de 1.024 byte. Cuando utilizamos índices 32-bit, el conjunto por defecto en donde se almacena y recupera supval es el conjunto **'S'**. Sin embargo, otra vez, supval puede ser almacenado en cualquiera de los 256 conjuntos, pero especificando el apropiado valor **tag** del conjunto. Las cadenas son una forma común de datos de longitud arbitraria y por eso son tratadas por funciones de manipulación **supval**. El siguiente ejemplo proporciona la forma de almacenar supval dentro de un netnode:

Almacenar supval en netnode

```
netnode n ("$ tora", 0, true);          //crea el netnode si no existe

char *dato_cadena = "ejemplo dato string supval";
char dato_binario [ ] = {0x43, 0x72, 0x61, 0x63, 0x6B, 0x53,
                         0x4C, 0x61, 0x74, 0x69, 0x6E, 0x6F, 0x53};

//almacenamos dato_binario en el conjunto 'S' con índice 1000 y debemos suministrar
//un puntero al dato y el tamaño del dato
n.supset (1000, dato_binario, sizeof (dato_binario));

//almacenamos dato_cadena en el conjunto 'S' con índice 1001. Si no se suministra el
//tamaño o el tamaño es cero, el tamaño del dato se calcula como:  strlen (dato) + 1
n.supset (1001, dato_cadena);

//almacenamos en otro conjunto que no sea 'S', 200 en este caso y con índice 500
n.supset (500, dato_binario, sizeof (dato_binario), (char)200);
```

La función **subset** requiere un índice de conjunto, un puntero al dato, la longitud del dato en bytes y un tag del conjunto que por defecto es **'S',** si se omite otro cualquiera. Si se omite el parámetro de longitud de dato, por defecto es **cero**. Cuando dicha longitud es especificada como cero, **subset** asume que el dato almacenado es una cadena, el cálculo de la longitud de un dato se realiza como **strlen (dato) + 1**, y se almacena el dato cadena finalizándolo con un **null** al final.

La recuperación de datos con **supval** es un poco más complicado, debido a que no podemos saber la cantidad de datos contenidos dentro de supval antes de recuperarlo. Cuando recuperamos datos de supval, los bytes son copiados fuera del netnode en un **buffer** de salida utilizado temporalmente ¿Cómo podemos saber que ha dicho buffer de salida le hemos asignado la cantidad suficiente de espacio para recibir los datos de supval? Para solucionarlo disponemos de dos métodos. El primer método es recuperar todos los datos supval dentro de un buffer de al menos 1.024 byte. El segundo método es prefijar el tamaño del buffer preguntándonos el posible tamaño de supval, fácil eh. Bien disponemos de dos funciones para recuperar supval. La función **supval** se utiliza para recuperar datos arbitrarios, mientras que la función **supstr** está especializada para recuperar datos cadena. Cada una de estas funciones espera un puntero a nuestro buffer de salida conjuntamente con el tamaño del mismo. El valor retornado por **supval** es el número de byte copiados en el buffer de salida, mientras que el valor retornado por **supstr** es la longitud de la cadena copiada en el buffer de salida sin incluir el **null** final, aunque dicho **null** es copiado en el buffer. Cada una de estas funciones, reconoce el caso especial en que el puntero **NULL** es utilizado substituyendo al puntero al buffer de salida. En tales casos, supval y supstr retornan el número de byte almacenados incluyendo cualquier final null requerido para tomar el dato supval. El siguiente ejemplo muestra la recuperación de un dato supval utilizando las funciones supval y supstr.

Recuperar supval del netnode

```
//determina el tamano del elemento 1000 en el conjunto 'S'. El puntero NULL indica
//que no estamos suministrando ningún buffer de salida
int longitud = n.supval (1000, NULL, 0);

char *buffer_salida = new char [longitud]; //distribuye un buffer de tamaño suficiente
n.supval (1000, buffer_salida, longitud);    //extrae datos del supval

//determina el tamaño del elemento 1001 en el conjunto 'S'. El puntero NULL indica
//que no suministramos ningún buffer de salida
longitud = n.supstr (1001, NULL, 0);

char *cadena_salida = new char [longitud]; //distribuye un buffer de tamaño suficiente
n.supval (1001, cadena_salida, longitud); //extrae datos del supval

//recupera un supval del conjunto 200, indice 500
char buffer [1024];
longitud = n.supval (500, buffer, sizeof (buffer), (char)200);
```

Utilizando **supval,** es posible acceder a cualquier dato almacenado en cualquier conjunto dentro de un netnode. Por ejemplo, las funciones supval pueden utilizarse para almacenar y recuperar datos **altval** limitando las operaciones **subset** y **supval** al tamaño de un **altval**. Sacado de **netnode.hpp**, veremos que esto de hecho es así en el caso de la ejecución de la función **altset**, como mostramos seguidamente:

```
bool altset (sval_t alt, nodeidx_t value, char tag=atag)    {
        Return subset (alt, &value, sizeof (value), tag);
}
```

Otro enlace con los netnodes es el proporcionado por los **hashval**. En vez de estar asociados con índices enteros, están asociados con cadenas clave. Las distintas versiones cargadas de la función **hashset** hacen fácil el poder asociar dato entero o dato

de conjunto con una clave **hash**, mientras que las funciones **hashval, hashstr** y **hashval_long** permiten recuperar hashval cuando se proporcionan con la apropiado clave hash. Los valores **tag** asociados con las funciones **hashXXX** oscilan entre 1 y 256 tablas **hash**, pero la tabla por defecto es **'H'**. Las otras tablas se pueden elegir especificando otro tag que no sea 'H'.

El último enlace a los netnode es **charval**. Las funciones **charval y charset** proporcionan una forma simple de almacenar datos de un byte dentro de un conjunto netnode. No existe ningún conjunto implícito asociado en la recuperación o almacenamiento de charval, por lo tanto es preciso especificar un tag de conjunto para cada operación realizada con charval. Los charval son almacenados en los mismos conjuntos que altval y supval y decir que las funciones charval realizan lo mismo que las de los supval pero con 1-byte.

Otra capacida proporcionada por la clase netnode es la habilidad de iterar por todo el contenido de un conjunto netnode, o tabla hash. La iteración es realizada utilizando las funciones **XXX1st, XXXnxt, XXXlast y XXXprev** disponibles para **altval, supval, hashval y charval**. El siguiente ejemplo muestra la iteración a través del conjunto por defecto **'A'**.

La iteración sobre **supval, charval y hashval** es realizada de forma similar; sin embargo, hallaremos sintaxis diferentes dependiendo del tipo de valores a los que se han de acceder. Por ejemplo, la iteración sobre **hashval** retorna **hashkey** en vez de índices de conjunto, los cuales deberán utilizarse para recuperar hashval.

Enumerar altval del netnode

```
netnode n ("$ tora", 0, true);
//Iterar por los altvals del primero al último
for (nodeidx_t  idx = n.alt1st ( ); idx != BADNODE; idx = n.altnxt (idx))    {
   ulong val = n.altval (idx);
   msg ("Altval encontrados ['A'] [%d] = %d\n", idx, val);
}

//Iterar por los altvals del último al primero
for (nodeidx_t idx = n.altlast ( ); idx != BADNODE; idx = n.altprev (idx))    {
   ulong val = n.altval (idx);
   msg ("Altval encontrados ['A'] [%d] = %d\n", idx, val);
}
```

Kit-kat teórico: NETNODES Y IDC GLOBAL ARRAYS

Si recordamos del escrito 14, el lenguaje de scripts IDC proporciona conjuntos globales persistentes. Los netnodes proporcionan la ayuda necesaria a IDC para poder almacenar dichos global arrays. Si cambiáramos la función de IDC llamada **CreateArray**, por la cadena **$ idc_array** es la forma de nombre para netnode, de la función para IDC. El netnodenumber del nuevo netnode creado es retornado al igual que el identificador de conjunto en IDC. La función de IDC **SetArrayLong** almacena un entero dentro del conjunto **altval ('A'),** mientras que la función **SetArrayString** almacena una cadena dentro del conjunto **supval ('S').** Cuando recuperamos un valor de un conjunto IDC utilizando la función **GetArrayElement**, las tag (**AR_LONG o AR_STR**) representan las tag de conjunto para **altval y supval** utilizados para almacenar el correspondiente entero o cadenas de datos.

15.2.2.3.—Eliminar netnodes y datos netnode

La clase netnode también proporciona funciones para eliminar elementos individuales del conjunto, el contenido completo de un conjunto o el contenido entero de un netnode. Eliminar un netnode completo es muy fácil:

```
netnode n ("$ tora", 0, true);
n.kill ( );                    //todo el contenido de n será eliminado
```

Cuando eliminemos un elemento individual del conjunto, o el contenido entero de un conjunto, deberemos tener el cuidado de escoger la función de eliminación apropiada debido a que los nombres de las funciones son muy similares, con lo cual elegir una forma equivocada puede ocasionar la pérdida de datos. El siguiente ejemplo muestra la eliminación de altval:

```
netnode n ("$ tora", 0, true);
   n.altdel (100);             //elimina el elemento 100 del conjunto por defecto 'A'
n.altdel (100, (char)3);       //elimina el elemento 100 del conjunto altval 3

   n.altdel ( );               //elimina todo el contenido del conjunto altval por defecto
n.altdel_all ('A');            //alternativa para eliminar el contenido del conjunto altval por
defecto
n.altdel_all ((char)3);        //elimina todo el contenido del conjunto altval 3
```

Observemos la similitud entre la sintaxis para eliminar todo el contenido del conjunto por defecto altval **n.altdel ( );** y la sintaxis para eliminar un elemento del conjunto altval por defecto **n.altdel (100);**. Si por alguna razón no especificamos un índice para borrar un elemento, podemos terminar eliminando todo el conjunto. Para eliminar datos supval, charval y hashval, existen funciones similares.

Performance Bigundill@