Reversando Ejercicio 15

Curso C y reversing Crackslatinos

Por sisco 0

Programas necesarios:

IDA Pro+Hex Rays, Dev C++

Cargando el programa en *IDA*

Cargamos el programa en *IDA* y nos posicionamos en la línea siguiente a *call __main*

```
.text:00401385 mov     dword ptr [esp], 5
.text:0040138C call    sub_401290
```

Posicionamos un *Breakpoint* pulsando *F2* en la primera línea que he puesto y pulsamos *F9*, directamente, para comenzar *debuggando* y así asegurar nuestros pasos.

En la primera línea movemos a la pila un *5*, mientras que en la segunda hacemos una llamada a una función *sub_401290*, que ahora veremos qué hace, la podremos renombrar más tarde.

I Primera función *sub_401290*

Entramos dentro de esta pulsando *F7*.

Nos encontramos con el siguiente código:

```
.text:00401290 push    ebp
.text:00401291 mov     ebp, esp
.text:00401293 sub     esp, 8
```

Reservamos un espacio en la pila, este espacio es para dos *Dwords*, es decir, 2 espacios de pila.

```
.text:00401296 mov     dword ptr [esp], 8          ; size_t
.text:0040129D call    malloc
.text:004012A2 mov     [ebp+var_4], eax
```

Ahora lo que hacemos es mover a la pila *8*, para más tarde llamar a *malloc*, por lo que estamos reservando *8 bytes* en memoria.

Debemos tener en cuenta que un entero y un puntero ocupan cada uno *4 bytes*, este dato quizás nos sirva para más tarde.

Llamamos a *malloc*, el cual nos devuelve en *eax* el puntero al espacio de memoria reservado.

Lo que hacemos es mover a *var_4* esta dirección.

Podemos **renombrar** *var_4* a *estruct1*, recuerden que para hacerlo debemos hacer click sobre él y pulsar *N*.

```
.text:004012A5 mov     edx, [ebp+estruct1]
.text:004012A8 mov     eax, [ebp+arg_0]
.text:004012AB mov     [edx], eax
```

Ahora movemos a *edx* esta misma dirección.

Movemos a *eax* el argumento con el que llamamos a la función. Es decir, una cifra, (En este caso es un *5*, tal y como vemos en el *debugging*).

Entonces **renombrarmos** *arg_0* y le pondremos de nombre *cifra_in*.

Más tarde movemos a la dirección apuntada por *edx* esta misma cifra, recordando que esto lo que hará será reemplazar los primeros *4* bytes de la estructura. Lo que nos queda es que la primera parte de la estructura será de tipo *int*.

```
.text:004012AD mov     eax, [ebp+estruct1]
.text:004012B0 mov     dword ptr [eax+4], 0
.text:004012B7 mov     eax, [ebp+estruct1]
.text:004012BA leave
.text:004012BB retn
```

Más tarde movemos a *eax* la dirección de nuestra estructura y movemos un *0* a esta con un *offset* de *4* bytes, por lo que estamos escribiendo en el segundo casillero de nuestra estructura.

Seguramente esta parte de la estructura será la que apunte al siguiente elemento de la lista.

Más tarde pasamos a *eax* la dirección de nuestra estructura y hacemos return, de esta forma estamos retornando la dirección de la estructura.
Por lo tanto podremos **renombrar** esta función a *crear_ficha*.
Nos queda en C así la función:

```
#include <stdio.h>
#include <malloc.h>
typedef struct {
   int cifra;
   int siguiente;
} ficha;
ficha * crear_ficha(int cifra_in);
int main(void)
{
   ficha *mi_ficha;
   mi_ficha=crear_ficha(5);
   return 0;
}
ficha * crear_ficha(int cifra_in)
{
     ficha *ficha_out;
     ficha_out=(ficha *)malloc(8);
     (*ficha_out).cifra=cifra_in;
     (*ficha_out).siguiente=0x0;
     return ficha_out;
}
```

Podemos compararla con *IDA* a ojo y vemos que es concluyente, ambas son idénticas.

II Segunda función sub_4012D0

```
.text:004012ED push   ebp
.text:004012EE mov    ebp, esp
.text:004012F0 sub    esp, 18h
```

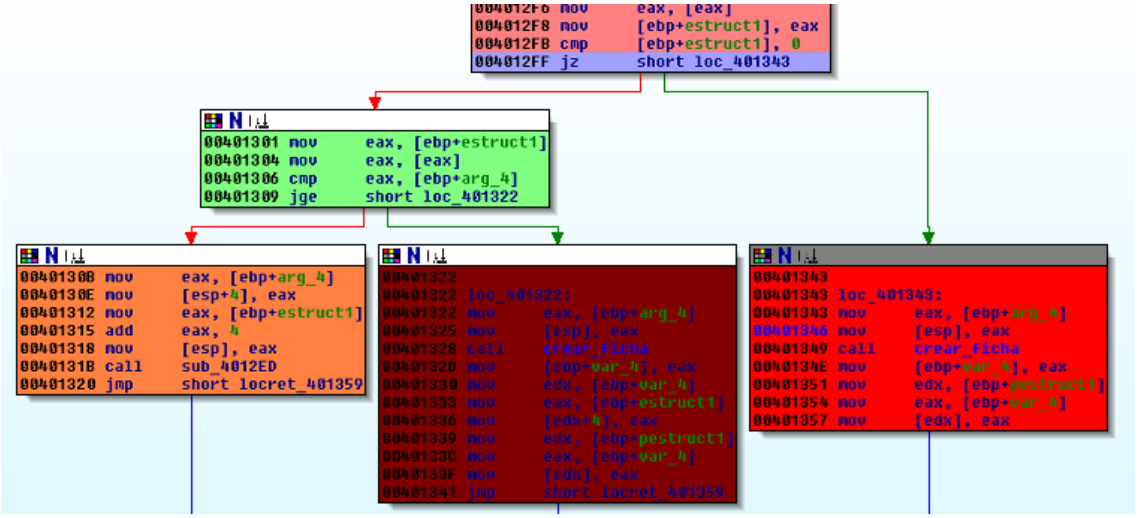
Reservamos un espacio de *0x18 bytes*, es decir, para guardar *6* elementos en la pila.

```
.text:004012F3 mov    eax, [ebp+arg_0]
.text:004012F6 mov    eax, [eax]
```

Movemos a *eax* el primer argumento, que es un puntero a una estructura, por lo que **renombramos** *arg_0*, le pondremos *pestruct1*.
Más tarde lo que hacemos es mover a *eax* el contenido de lo apuntado por *eax*, es decir, movemos la dirección de *estruct1* a *eax*.

```
.text:004012F8 mov    [ebp+var_8], eax
.text:004012FB cmp    [ebp+var_8], 0
.text:004012FF jz     short loc_401343
```

Más tarde movemos *eax* a *var_8*, por lo que estamos moviendo la dirección de la estructura a *var_8*, vamos a **renombrar** *var_8* por *estruct1*.
Más tarde lo compara con *0*, es decir, si es nulo. Esto no parece tener mucho sentido por ahora, pero lo cobrará.
En el caso de que sea cero salta a *0x00401343*.



Detalle de la estructura de la función

Ya hemos analizado la caja rosa, ahora tenemos una rama, que nos puede llevar a la caja verde o a la roja.

Analicemos primeramente la caja roja.

1 Caja roja, el puntero apunta a cero

Llegaremos a esta si nuestro puntero apunta a cero, y no lo hace a otra estructura (Este es el caso de fin de lista).

Observemos las instrucciones aquí:

```
.text:00401343 mov    eax, [ebp+arg_4]
.text:00401346 mov    [esp], eax
.text:00401349 call   crear_ficha
.text:0040134E mov    [ebp+var_4], eax
.text:00401351 mov    edx, [ebp+pestruct1]
.text:00401354 mov    eax, [ebp+var_4]
.text:00401357 mov    [edx], eax
```

Lo que hacemos es mover *arg_4* a *eax*, que es el número constante con el que hemos llamado a la función, lo **renombramos**, le pondremos de nombre *cifra_in*.

Más tarde lo cargamos en el *stack* y llamamos a *crear_ficha*, por lo que estamos creando una nueva ficha con este valor *cifra_in*.

Guardamos esta estructura en *var_4*, por lo que **renombramos** *var_4* a *estruct2*.

Más tarde lo que hacemos es mover a *edx* el puntero que apunta a una estructura (realmente apunta a cero en este caso), y en *eax* emplazamos nuestra nueva estructura *estruct2*.

En el último paso hacemos que nuestro puntero de estructuras apunte a *estruct2*, la nueva estructura.

Observamos que más debajo de todas las cajas simplemente tenemos un *return* que no nos devuelve nada, por lo que la función será de tipo *void*.

Podemos **renombrar** nuestra función, le pondremos de nombre *anyadir_ficha*.

Por lo tanto nuestra función queda por ahora así:

```
#include <stdio.h>
#include <malloc.h>
typedef struct {
   int cifra;
   int siguiente;
} ficha;
ficha * crear_ficha(int cifra_in);
void anyadir_ficha(ficha **pficha,int cifra_in);
int main(void)
{
   ficha *mi_ficha;
   mi_ficha=crear_ficha(5);
```

```
    anyadir_ficha(&mi_ficha,6);
    return 0;
}
ficha * crear_ficha(int cifra_in)
{
     ficha *ficha_out;
     ficha_out=(ficha *)malloc(8);
     (*ficha_out).cifra=cifra_in;
     (*ficha_out).siguiente=0x0;
     return ficha_out;
}
void anyadir_ficha(ficha **pficha,int cifra_in)
{
    ficha *estruct1;
    estruct1=*pficha;
    if ((estruct1)!=0)
    {

    }
    else
    {
        ficha *estruct2=crear_ficha(cifra_in);
        *pficha=estruct2;
    }
}
```

2 Caja verde intermedia

Pasamos a inspeccionar la caja verde.

```
.text:00401301 mov    eax, [ebp+estruct1]
.text:00401304 mov    eax, [eax]
.text:00401306 cmp    eax, [ebp+cifra_in]
.text:00401309 jge    short loc_401322
```

Aquí lo que hace es comparar el número de la estructura que le hemos pasado a la función con *cifra_in*, en el caso de que el número de nuestra estructura sea menor que *cifra_in* saltaríamos a la caja naranja, en caso contrario a la marrón.

Es decir, todo esto es un *if* que irá dentro de nuestro *if*.

Como detalle, podemos ver cómo quedaría parte de nuestro código:

```
    if ((estruct1)!=0)
    {
       if (cifra_in>(*estruct1).cifra)
       {
          //Caja naranja
       }
       else
       {
          //Caja marrón
       }
    }
```

3 Caja naranja

En este caso nuestra estructura tiene una cifra menor que la de entrada.

```
.text:0040130B mov    eax, [ebp+cifra_in]
.text:0040130E mov    [esp+4], eax
.text:00401312 mov    eax, [ebp+estruct1]
.text:00401315 add    eax, 4
.text:00401318 mov    [esp], eax
.text:0040131B call   sub_4012ED
.text:00401320 jmp    short locret_401359
```

Lo que hacemos es mover a *eax* nuestra cifra de entrada.

Más tarde movemos *eax* a la pila.

Movemos a *eax* nuestra estructura y sumamos 4, de esta forma lo que hacemos es referenciar a la parte *siguiente* de nuestra estructura.
Llamamos recursivamente a la función donde nos encontramos.
Es decir, en el caso de que nuestra estructura tenga una cifra menor a la de entrada lo que haremos será ir buscando en la lista aquella en la que no se cumple esta condición.
Más tarde tenemos un *jmp short* que se ejecutará cuando se haya concluido todo el procedimiento recursivo y realizará return, ya que esta es la dirección donde está el return de la función.
Nos queda el siguiente código (vista detalle)

```
if (cifra_in>(*estruct1).cifra)
{
   anyadir_ficha((ficha **)(estruct1)+1,cifra_in);
}
```

4 Caja marrón

Y por fín, llegamos a la caja marrón, donde nos queda el siguiente código ensamblador:

```
.text:00401322 mov    eax, [ebp+cifra_in]
.text:00401325 mov    [esp], eax
.text:00401328 call   crear_ficha
.text:0040132D mov    [ebp+estruct2], eax
.text:00401330 mov    edx, [ebp+estruct2]
.text:00401333 mov    eax, [ebp+estruct1]
.text:00401336 mov    [edx+4], eax
.text:00401339 mov    edx, [ebp+pestruct1]
.text:0040133C mov    eax, [ebp+estruct2]
.text:0040133F mov    [edx], eax
.text:00401341 jmp    short locret_401359
```

Observamos que lo que hace es crear una ficha con *cifra_in*, además, esto lo pasa a *estruct2*, más tarde lo que hará será hacer que *estruct2.siguiente* sea igual a *estruct1*.
También hará que nuestro puntero de estructuras apunte a *estruct2*.
Es decir, este puntero siempre estará apuntando a la cifra mínima.
Finalmente sale saltando al *locret*.
Nos queda el siguiente código completo:

```
#include <stdio.h>
#include <malloc.h>
typedef struct {
   int cifra;
   int *siguiente;
} ficha;
ficha * crear_ficha(int cifra_in);
void anyadir_ficha(ficha **pficha,int cifra_in);
int main(void)
{
   ficha *mi_ficha;
   mi_ficha=crear_ficha(5);
   anyadir_ficha(&mi_ficha,6);
   return 0;
}
ficha * crear_ficha(int cifra_in)
{
    ficha *ficha_out;
    ficha_out=(ficha *)malloc(8);
    (*ficha_out).cifra=cifra_in;
    (*ficha_out).siguiente=0x0;
    return ficha_out;
}
void anyadir_ficha(ficha **pficha,int cifra_in)
{
```

```
    ficha *estruct1;
    estruct1=*pficha;
    if ((estruct1)!=0)
    {
       if (cifra_in>(*estruct1).cifra)
       {
          anyadir_ficha((ficha **)(estruct1)+1,cifra_in);
          //Utilizamos +1 para que separe el add
       }
       else
       {
          ficha *estruct2=crear_ficha(cifra_in);
          (*estruct2).siguiente=(int *)estruct1;
          *pficha=estruct2;
       }
    }
    else
    {
        ficha *estruct2=crear_ficha(cifra_in);
        *pficha=estruct2;
    }
}
```

Ya hemos analizado todas las cajas, tenemos esta rutina totalmente analizado, la **renombramos** y le pondremos de nombre *anyadir_ficha*.
Vemos que lo hace tanto con 3, como con 2, como con 6, lo agregamos a nuestro código.

III Tercera función *sub_4012BC*

Ya tan solo nos queda la función de impresión.
Recordamos que en *mi_ficha* tenemos la dirección de la estructura que se encuentra primera en la lista, esta es la de la cifra más baja.
Tenemos esto en *assembler* (lo pongo todo ya que es trivial):

```
.text:004012BC push    ebp
.text:004012BD mov     ebp, esp
.text:004012BF sub     esp, 8
.text:004012C2 cmp     [ebp+arg_0], 0
.text:004012C6 jz      short locret_4012EB
.text:004012C8 mov     eax, [ebp+arg_0]
.text:004012CB mov     eax, [eax]
.text:004012CD mov     [esp+4], eax
.text:004012D1 mov     dword ptr [esp], offset aD      ; "%d\n"
.text:004012D8 call    printf
.text:004012DD mov     eax, [ebp+arg_0]
.text:004012E0 mov     eax, [eax+4]
.text:004012E3 mov     [esp], eax
.text:004012E6 call    sub_4012BC
```

Lo que hace es, primero, reservar 2 huecos en la pila.
Más tarde lo que hace es comparar el argumento de entrada (esta será la dirección de nuestra estructura) con *cero*.
En el caso de que sea *cero* saltaría a *4012EB*, que es la *locret*, es decir, se sale de la función.
En el caso de que no sea cero tenemos que mueve a *eax* la estructura, y más tarde accede al primer campo de esta mediante [*eax*], moviéndolo a *eax*.
Mueve la cifra a la pila.
Mueve el puntero de la cadena de formato *"%d\n"* a la pila.
Llama a la función *printf* para mostrarlo en pantalla.
Y ahora lo que hace es acceder al campo *siguiente* de nuestra estructura y llamarse a sí misma la función, es otra función recursiva.
Pues bien, nos quedaría el código de la función tal que así:

```
void imprime_lista(ficha *pficha)
```

```
{
    if (pficha!=0)
    {
      printf("%d\n",(*pficha).cifra);
      imprime_lista((ficha *)(*pficha).siguiente);
    }
}
```

IV Finalizando

Finalizando, al volver al *main* simplemente tenemos lo siguiente:

```
.text:004013D8 call    getchar
.text:004013DD call    getchar
.text:004013E2 mov     eax, 0
.text:004013E7 leave
.text:004013E8 retn
```

Dos *getchar()* y un *return 0,* tal y como lo teníamos, agregamos los *getchar()* y ya tenemos nuestro código completo.

Código fuente completo

```
#include <stdio.h>
#include <malloc.h>
typedef struct {
   int cifra;
   int *siguiente;
} ficha;
ficha * crear_ficha(int cifra_in);
void anyadir_ficha(ficha **pficha,int cifra_in);
void imprime_lista(ficha *pficha);
int main(void)
{
   ficha *mi_ficha;
   mi_ficha=crear_ficha(5);
   anyadir_ficha(&mi_ficha,3);
   anyadir_ficha(&mi_ficha,2);
   anyadir_ficha(&mi_ficha,6);
   imprime_lista(mi_ficha);
   getchar();
   getchar();
   return 0;
}
ficha * crear_ficha(int cifra_in)
{
    ficha *ficha_out;
    ficha_out=(ficha *)malloc(8);
    (*ficha_out).cifra=cifra_in;
    (*ficha_out).siguiente=0x0;
    return ficha_out;
}
void anyadir_ficha(ficha **pficha,int cifra_in)
{
   ficha *estruct1;
   estruct1=*pficha;
   if ((estruct1)!=0)
   {
      if (cifra_in>(*estruct1).cifra)
      {
         anyadir_ficha((ficha **)(estruct1)+1,cifra_in);
         //Utilizamos +1 para que separe el add
```

```
        }
        else
        {
           ficha *estruct2=crear_ficha(cifra_in);
           (*estruct2).siguiente=(int *)estruct1;
           *pficha=estruct2;
        }
    }
    else
    {
        ficha *estruct2=crear_ficha(cifra_in);
        *pficha=estruct2;
    }
}
void imprime_lista(ficha *pficha)
{
    if (pficha!=0)
    {
      printf("%d\n",(*pficha).cifra);
      imprime_lista((ficha *)(*pficha).siguiente);
    }
}
```

Podemos ejecutar ambos programas y ver que funcionan exactamente igual, además, vamos a ver con *IDA* el *diff*.

Para ello con el *IDA* abrimos el fichero original y vamos a *Edit→Plugins→turbodiff*

Una vez allí usamos *take info from this idb*. Hacemos click en *OK* y en *OK*.

Entonces hacemos *File→Close*. (NO seleccionar la opción **DON'T SAVE…**).

Abrimos el fichero nuestro compilado con *IDA*.

Realizamos lo mismo:

Edit→Plugins→turbodiff

take info from this idb. *OK*,*OK*.

Ahora compararemos haciendo click en *Edit→Plugins→turbodiff*

compare with… , OK.

Entonces seleccionamos nuestro fichero original y nos aparece una parte sospechosa.

Vemos que lo que ocurre es que ve intercambiadas *var_4* y *var_8*. Interesante.

Simplemente debemos cambiar esto en la función de las cajas.

El código, una vez completamente modificado nos quedaría así:

```
#include <stdio.h>
#include <malloc.h>
typedef struct {
   int cifra;
   int *siguiente;
} ficha;
ficha * crear_ficha(int cifra_in);
void anyadir_ficha(ficha **pficha,int cifra_in);
void imprime_lista(ficha *pficha);
int main(void)
{
   ficha *mi_ficha;
   mi_ficha=crear_ficha(5);
   anyadir_ficha(&mi_ficha,3);
   anyadir_ficha(&mi_ficha,2);
   anyadir_ficha(&mi_ficha,6);
   imprime_lista(mi_ficha);
   getchar();
   getchar();
```

```
   return 0;
}
ficha * crear_ficha(int cifra_in)
{
     ficha *ficha_out;
     ficha_out=(ficha *)malloc(8);
     (*ficha_out).cifra=cifra_in;
     (*ficha_out).siguiente=0x0;
     return ficha_out;
}
void anyadir_ficha(ficha **pficha,int cifra_in)
{
   ficha *estruct2;
   ficha *estruct1;
   estruct1=*pficha;
   if ((estruct1)!=0)
   {
      if (cifra_in>(*estruct1).cifra)
      {
         anyadir_ficha((ficha **)(estruct1)+1,cifra_in);
         //Utilizamos +1 para que separe el add
      }
      else
      {
         estruct2=crear_ficha(cifra_in);
         (*estruct2).siguiente=(int *)estruct1;
         *pficha=estruct2;
      }
   }
   else
   {
       estruct2=crear_ficha(cifra_in);
       *pficha=estruct2;
   }
}
void imprime_lista(ficha *pficha)
{
    if (pficha!=0)
    {
      printf("%d\n",(*pficha).cifra);
      imprime_lista((ficha *)(*pficha).siguiente);
    }
}
```

Y nos queda un *identical* en todo. Perfecto.

Recuerden que el fichero sea *.c*, ya que, en el caso de que sea *.cpp* puede dar lugar a *diffs*.

¡Hasta otra!