Reversando Ejercicio 13

Curso C y reversing Crackslatinos
Por sisco 0
Programas necesarios:
IDA Pro+Hex Rays, Dev C++

Viendo cómo se mueve el puchero

Abrimos el programa en *IDA*, a ver qué hay.
Lo ponemos en *Text View* por ahora (esto se hace con el click derecho sobre el fondo de la ventana de *IDA View*).
Lo que nos interesa es a partir de la línea:
.text:004012DC mov [ebp+var_4], 5
Ya que es justo debajo de la llamada a *__main*, lo anterior es código del compilador.

Variables y punteros iniciales

.text:004012DC mov [ebp+var_4], 5
.text:004012E3 mov [ebp+var_8], 0Ch
Parece ser que tenemos dos variables, vamos a renombrarlas, les pondremos de nombre *variable1* y *variable2*.
Por ahora ninguna de las dos parecen ser punteros, pero debemos tener en cuenta que más adelante podemos hacer referencia a sus posiciones en memoria, es decir, lo equivalente en 'C', *&variable1* y *&variable2*.

Cargando datos en la pila y realizando la llamada a una función

.text:004012EA lea eax, [ebp+variable2]
.text:004012ED mov [esp+4], eax
.text:004012F1 lea eax, [ebp+variable1]
.text:004012F4 mov [esp], eax
.text:004012F7 call sub_401290
Vaya vaya, parece que cargamos en *eax* la dirección de *variable2*.
Más tarde pasamos *eax* a la pila.
Ahora cargamos en *eax* la dirección de *variable1* y la pasamos a la pila también.
Así que estamos llamando en la siguiente línea a nuestra función en *0x00401290*, esta recibirá como parámetros *&variable1* y *&variable2*.
Entrando dentro de la función

Exploremos a ver qué hace la función

Hay que destacar que la gracia de hacer todo esto es haciéndolo sin haber ejecutado previamente el programa.
Entramos dentro de la función haciendo *doble click* en *sub_401290*, por ahora no la renombraremos.
.text:00401290 push ebp
.text:00401291 mov ebp, esp
.text:00401293 sub esp, 4
Lo primero que hace es crear una pequeña pila con espacio para guardar 4 bytes, o 32 bits, o un posible puntero.
.text:00401296 mov eax, [ebp+arg_0]
.text:00401299 mov eax, [eax]
Ahora movemos a *eax* el primer argumento que le pasamos, ¿Qué argumento es este?

Pues este es el último argumento que metimos en la pila antes de llamar a la función, es decir, se trata de *&variable1*.
Pues lo que realizamos en la siguiente línea es mover el contenido de *variable1* a *variable1*.
Ahora tenemos en *eax* un *0x05*.
Vamos a renombrar *arg_0* por *_variable_1* y *arg_4* por *_variable_2*.

```
.text:0040129B        mov    [ebp+var_4], eax
```

Ahora movemos *eax* a *var_4*, que es nuestra variable local de la función, para la que habíamos reservado espacio al principio de esta.

```
.text:0040129E        mov    edx, [ebp+_variable_1]
.text:004012A1        mov    eax, [ebp+_variable_2]
.text:004012A4        mov    eax, [eax]
.text:004012A6        mov    [edx], eax
```

Bien, ahora en el código vemos que movemos a *edx* la dirección de *variable_1*, esto es *_variable_1*, el primer parámetro de nuestra función.
Más tarde movemos a *eax* la dirección de *varlable_2*, esto es *_variable_2*, el segundo parámetro de nuestra función.
Ahora movemos a *eax* el contenido de lo apuntado por *eax*, es decir, movemos a *eax* el *0x0C*, el valor de *variable_2*.
Y movemos a lo apuntado por *edx* el valor de *eax*, es decir, estamos moviendo el *0x0C* a la posición de memoria referenciada por *_variable_1*, estamos asignando *0x0C* a *variable_1*.

```
.text:004012A8        mov    edx, [ebp+_variable_2]
.text:004012AB        mov    eax, [ebp+var_4]
.text:004012AE        mov    [edx], eax
.text:004012B0        leave
.text:004012B1        retn
```

Movemos a *edx* la dirección de memoria de *variable_2*.
Más tarde movemos a *eax* el valor de *var_4*, en esta variable habíamos guardado anteriormente un *0x05*, el valor que tenía antes de entrar a la función *variable_1*.
Y ahora movemos a la posición de memoria referencia por *edx* este valor, es decir, estamos metiendo un *0x05* en *variable_2*.

Resumen de la función

En resumen, nuestra función lo que hace es intercambiar los valores de las variables, para ello le pasamos como parámetros las direcciones de memoria de estas dos variables.
La pasamos a renombrar, le pondremos de nombre *intercambiar*.
Salimos de la función.

Fuera de la función, seguimos en main

```
.text:004012FC        mov    eax, [ebp+variable2]
.text:004012FF        mov    [esp+8], eax
.text:00401303        mov    eax, [ebp+variable1]
.text:00401306        mov    [esp+4], eax
.text:0040130A        mov    dword ptr [esp], offset aAhoraAEsDYBEsD ; "Ahora a es"…
.text:00401311        call   printf
```

Primeramente lo que hacemos es mover a *eax* el valor de *variable_2*, debemos tener en cuenta, que tras haber pasado por la función tenemos la situación *variable_2=0x05* y *variable_1=0x0C*.
Pues bien, estamos moviendo a *eax* el valor *0x0C*.
Y más tarde pasamos este valor a la pila, seguramente como parámetro para la función *printf*.
Más tarde pasamos a *eax* el valor de *variable_1*, que es *0x05* y lo pasamos a la pila.
Movemos la dirección de memoria de la cadena para imprimir a la pila, vemos que hace referencia con dos *%d* a cada uno de los parámetros que anteriormente pasamos a la pila, es decir, *variable_1* y *variable_2*, que los llama *a* y *b*.

Debemos tener en cuenta que al tener *%d* el *0x0C* se imprimirá como su correspondiente representación decimal, es decir, *12*.
Llamamos a *printf* para imprimir en pantalla.

Finalizando

```
.text:00401316        call   getchar
.text:0040131B        leave
.text:0040131C        retn
```

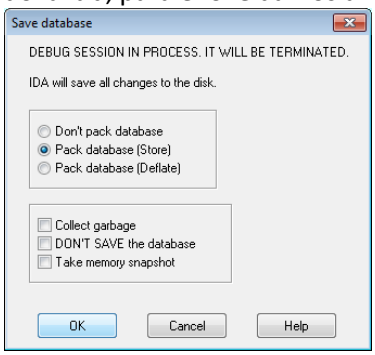
Llamamos a *getchar* para esperar que se pulse una tecla y salimos de la función principal, ya se acabó el programa.

Código en *C*

```
#include <stdio.h>
void intercambio(int *a,int *b);   //Declaración de prototipos de la función
int main (void)
{
   int a=0x05,b=0x0C;    //Declaramos las variables
   intercambio(&a,&b);   //Llamamos a la función
   printf("Ahora a es %d y b es %d\n",a,b);
   getchar();          //Saliendo del programa
   return;
}
void intercambio(int *a,int *b)
{
    int variable_de_paso; //Esta será nuestra variable local
    variable_de_paso=*a;  //Movemos contenido de 'a'
    *a=*b;                //Metemos 'b' en 'a'
    *b=variable_de_paso;  /*Y ahora metemos el antiguo
                            valor de 'a' en 'b'*/
    return;
}
```

Comparando nuestro programa con el original

Ejecutamos ambos programas y vemos que dan la misma salida ambos.
Entonces vamos a realizar una comparación con el *plugin* de *IDA turbodiff*.
Cargamos en *IDA* el programa *Punteros.exe* original (Si es que lo habíamos cerrado antes), entonces, pulsamos en *Edit→Plugins→turbodiff*
Hacemos click en *take info from this idb* y en *OK*, y de nuevo en *OK*.
Ahora vamos a cerrarlo guardando la *idb*, para ello le damos a *File→Close*.



Click en OK.

Pasamos a abrir nuestro ejecutable que compilamos de nuestro código en *C* con el *IDA*.
Vamos de nuevo a *Edit→Plugins→turbodiff* y *take info from this idb*, *OK* y *OK*.
Ahora compararemos ambas extracciones de *turbodiff*, vamos de nuevo a *Edit→Plugins→turbodiff*, pero ahora seleccionamos la opción *compare with*.

Vamos hasta el *path* de nuestro ejecutable original donde teníamos guardado el fichero *idb* de este y lo seleccionamos con doble click, pulsamos *OK*.
Vemos que todo aparece como *identical* a la izquierda, perfecto, no hay ninguna diferencia, hemos conseguido reversar el programa de forma idéntica.