Reversando Ejercicio 14

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

Cargando con *IDA*

Esta vez se nos enrevesa un poco más la cosa, ahora debemos de reversar un ejercicio con estructuras, arrays y punteros.
Vamos a encararnos con el problema, lo cargamos con *IDA*.
Hacemos click derecho en el código y usamos *Text View*, ya que parece que por ahora no nos hará falta verlo de forma gráfica (muy intuitivo para otros casos).
Si tienes el *Hex Rays* podemos pulsar *F5*, por si alguna vez nos perdemos, pero vamos a hacerlo sin tenerlo en cuenta.
Ponemos un *Breakpoint* en la línea *0x004012C3* y comenzamos a ejecutar para poder ver los cambios en vivo.

Copiando a memoria Juan el Viejo

Campo nombre

```
.text:004012C3          mov    dword ptr [esp+4], offset aJuanElViejo ; "Juan el Viejo\n"
.text:004012CB          lea    eax, [ebp+var_238]
.text:004012D1          mov    [esp], eax     ; char *
.text:004012D4          call   strcpy
```

Lo primero que hacemos es mover a la pila la dirección de memoria del *String* "Juan el Viejo\n".
Más tarde cargamos en *eax* la posición *ebp-0x238*, la cargamos en el *Hex-View*, suponemos que copiará ahí nuestro *string*.
Más tarde tenemos que mueve el valor de *eax* a la pila.
Es decir, tenemos en la pila como parámetros para el próximo *strcpy* un puntero que apunta a una dirección de memoria *ebp-0x238=0x0022FD10* y además también tenemos la dirección de memoria (puntero) a la cadena de texto "Juan el Viejo\n".
Llamamos a *strcpy* y observamos en el *Hex-View* que se ha copiado aquí la cadena de texto.

Campo email

```
.text:004012D9 mov    dword ptr [esp+4], offset aJuan@chiste_co ; "juan@chiste.com :-)\n"
.text:004012E1 lea    eax, [ebp+var_238]
.text:004012E7 add    eax, 130h
.text:004012EC mov    [esp], eax              ; char *
.text:004012EF call   strcpy
```

Pasamos a la pila la dirección de la cadena de texto del email "juan@chiste.com :-)\n"
Más tarde movemos a *eax* la dirección de memoria *ebp-0x238*, recordar, que anteriormente para el nombre habíamos movido esta misma, interesante.
Ya que ahora añadimos *0x130* a la dirección *ebp-0x238*.
Movemos a la pila el valor de *eax* y llamamos a *strcpy*.
Pues bien, podemos vigilar el valor de *ebp-0x238+0x130=0x0022FE40*.

Campo numérico

```
.text:004012F4 mov    [ebp+var_10C], 0C8h
```

Parece ser que movemos a *ebp-0x10C* un número *0xC8=200*.
No sabemos qué significa, por ello lo llamaremos número.

Posible conclusión

Memoria desde ebp-0x238

Pensamos que *nombre* tiene un espacio reservado de *0x130-0x04=0x12C=300*.
Nos sale un número redondo, eso está bien, estamos más seguros.

```
struct {
    char nombre[300];
    long int numero;
    char email[x];
} miestructura;
```

Por ahora no sabemos el valor de *x*, sabemos que como mínimo será 20, ya que este es la longitud del email más uno (El '\0' final).
Lo que haremos será renombrer *var_238* por *estruc_juan*.

Reservando espacio

.text:004012FE mov dword ptr [esp], 22Ch ; size_t
.text:00401305 call malloc

Ahora movemos a *ebp-0x10C* el valor *0x0C8*, más tarde, movemos a la pila el valor *0x22C*.
Llamamos a *malloc*.
Quizás es que nuestra *estructura* ocupe en total *0x22C=556* bytes.
Por lo tanto, el campo *email* ocupará *556-4-300=252* bytes.
Nos queda así la estructura:

```
struct {
    char nombre[300];
    long int numero;
    char email[252];
} miestructura;
```

En total, nuestra estructura ocupa 900 bytes, vaya, casi 1K.
Tras pasar el *malloc* tenemos:

.text:0040130A mov [ebp+var_23C], eax

¿Qué hay en *eax?*
Pues lo que hay es la salida de *malloc*, que es un puntero a una dirección de memoria libre donde ha reservado *0x22C* bytes.

El tamaño para guardar nuestra estructura.

Copiando a memoria Pedrito

.text:0040130A mov [ebp+var_23C], eax
.text:00401310 mov dword ptr [esp+4], offset aPedrito ; "Pedrito\n"
.text:00401318 mov eax, [ebp+var_23C]
.text:0040131E mov [esp], eax ; char *
.text:00401321 call strcpy

Lo explicaré un poco más rápido, ya que es igual que lo anterior.
Por lo tanto, renombraremos *var_23C*, le pondremos de nombre *pestruc_pedrito*.
Lo que hacemos es mover la cadena "Pedrito\n" a *ebp-0x23C*.
Si nos fijamos lo que estamos usando es *ebp+var_23C*, este es el puntero a la estructura de pedrito.
El espacio que habíamos reservado anteriormente de *0x22C* bytes era para la estructura de *Pedrito* que estamos usando ahora, pensamos que es del mismo tipo que la estructura de *Juan*, es decir, tenemos tan solo un tipo de estructura.

.text:00401326 mov dword ptr [esp+4], offset aPedrito@gmail_ ; "pedrito@gmail.com\n"
.text:0040132E mov eax, [ebp+pestruc_pedrito]
.text:00401334 add eax, 130h
.text:00401339 mov [esp], eax ; char *
.text:0040133C call strcpy

Guardamos el *email*, de nuevo vemos un *offset* de *0x130=304* bytes con respecto a *nombre*.

.text:00401341 mov eax, [ebp+pestruc_pedrito]
.text:00401347 mov dword ptr [eax+12Ch], 15h
.text:00401351 mov eax, [ebp+var_10C]
.text:00401357 mov [esp+0Ch], eax

De nuevo tenemos que se guarda otro valor en el campo numérico, en este caso, en el campo numérico de pedrito insertamos *0x15=21*.

Imprimiendo datos de Juan

.text:00401351 mov eax, [ebp+var_10C]
.text:00401357 mov [esp+0Ch], eax
.text:0040135B lea eax, [ebp+estruc_juan]
.text:00401361 add eax, 130h
.text:00401366 mov [esp+8], eax
.text:0040136A lea eax, [ebp+estruc_juan]
.text:00401370 mov [esp+4], eax
.text:00401374 mov dword ptr [esp], offset aPrimeraPersona ; "Primera persona: %s, %s, con edad %d\n"
.text:0040137B call printf

Primero cargamos en *eax* el contenido de la edad de Juan.
Más tarde movemos a la pila este valor.
Cargamos en *eax* la dirección de memoria que apunta al primer campo del nombre de *Juan*, es decir, al nombre.
Pero más tarde le añadimos *0x130*, con lo que ahora *eax* apunta al email de Juan.
Esto lo pasamos a la pila.
Ahora sí cargamos el nombre en la pila pasando por *eax*.
Pasamos a la pila también la cadena con el formato y sus "%s,%s,%d" correspondientes para imprimir los datos.

Imprimiendo datos de Pedrito

.text:00401380 mov eax, [ebp+pestruc_pedrito]
.text:00401386 mov eax, [eax+12Ch]

```
.text:0040138C mov     [esp+0Ch], eax
.text:00401390 mov     eax, [ebp+pestruc_pedrito]
.text:00401396 add     eax, 130h
.text:0040139B mov     [esp+8], eax
.text:0040139F mov     eax, [ebp+pestruc_pedrito]
.text:004013A5 mov     [esp+4], eax
.text:004013A9 mov     dword ptr [esp], offset aSegundaPersona ; "Segunda persona: %s, %s, con edad %d\n"
.text:004013B0 call    printf
```

Aquí hacemos lo mismo con *Pedrito*, creo que no es necesario comentarlo, ya que es igual, tan solo se usa el puntero de la estructura.

Finalizando

```
.text:004013B5 mov     eax, [ebp+pestruc_pedrito]
.text:004013BB mov     [esp], eax              ; void *
.text:004013BE call    free
.text:004013C3 call    getchar
.text:004013C8 leave
.text:004013C9 retn
```

Aquí liberamos el espacio apuntado por el puntero de la estructura *pedrito*.
También usamos *getchar()* para esperar a que se pulse una tecla, el programa se termina.

En resumen

En resumen tenemos un tipo de estructura de este tipo:

```
struct {
    char nombre[300];
    long int edad;
    char email[252];
} miestructura;
```

Hemos llegado a la conclusión de que el segundo es la *edad* por los *printf* que aparecen.
Además, sabemos que declararemos una estructura para *juan* y otra para *pedrito*, esta segunda se definirá a través de un puntero.
Más tarde se imprimirán los datos de estas.

Código en C

```c
#include <stdio.h>
#include <string.h>
//Definimos nuestra estructura
typedef struct {
	char nombre[300];
	int edad;
	char email[252];
} miestructura;
int main(void)
{
   //Definimos Juan sin puntero:
   miestructura juan;
   strcpy(juan.nombre,"Juan el Viejo\n");
   strcpy(juan.email,"juan@chiste.com :-)\n");
   juan.edad=200;
   //Definimos Pedrito con puntero:
   miestructura *pedrito;
   pedrito=(miestructura *)malloc(0x22C);    //Reservamos espacio
   strcpy((*pedrito).nombre,"Pedrito\n");
   strcpy((*pedrito).email,"pedrito@gmail.com\n");
   (*pedrito).edad=21;
   //Imprimimos datos de Juan
   printf("Primera persona: %s,%s, con edad
%d\n",juan.nombre,juan.email,juan.edad);
   //Imprimimos datos de Pedrito
   printf("Segunda persona: %s,%s, con edad
%d\n",(*pedrito).nombre,(*pedrito).email,(*pedrito).edad);

   //Finalizando
   free(pedrito);
   getchar();
   return;
}
```

Al comparar con *TurboDiff* nos muestra todo *identical* tras haberlo compilado con *Dev-C++,* además vemos que el programa ejecuta de forma complementa idéntica.
Hemos llegado a la solución.