

19: Práctica 2/4

Curso C y reversing Crackslatinos

Por sisco 0

Veamos qué tal se nos da el segundo.

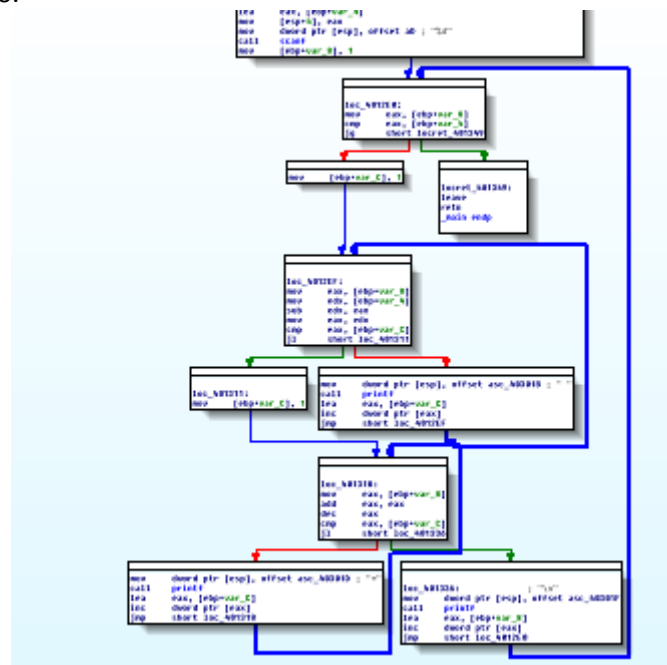
Cargando en IDA

Lo cargamos en IDA y nos fijamos en algo interesante.

```
.text:00401290 var_10      = dword ptr -10h
.text:00401290 var_C      = dword ptr -0Ch
.text:00401290 var_8      = dword ptr -8
.text:00401290 var_4      = dword ptr -4
.text:00401290 argc       = dword ptr 8
.text:00401290 argv       = dword ptr 0Ch
```

A simple vista parece que no tenemos muchas variables, aunque quizás, puede que usemos *argv* y *argc* en alguna parte del programa, para ello lo que haremos será seleccionarlos uno a uno e ir viendo si se marcan otras partes del programa.

Pues parece que no.



Si ya sabía yo...

No iba a ser todo tan fácil, hay tropecientos bucles.

Pues bien, habrá que estudiarlos.

Comenzando por el principio

```
.text:004012BA mov     dword ptr [esp], offset aTama ; "tama"
.text:004012C1 call    printf
```

Aquí hacemos un *printf* a algo, ¿pero realmente estamos imprimiendo tan solo la cadena *tama*?

Me da que no, quizás queremos imprimir *tamaño*, o algo así, ¿no?, para saberlo lo que haremos será hacer doble clic donde pone *aTama* y observamos qué es lo que hay.

Observamos que realmente lo que pone es *"tamaño de la piramide: "*.

Pues bien, podemos volver al *_main* donde estábamos, para eso pulsamos G y escribimos *_main*.

```
.text:004012C6 lea     eax, [ebp+tamanho_pir]
.text:004012C9 mov     [esp+4], eax
.text:004012CD mov     dword ptr [esp], offset aD ; "%d"
.text:004012D4 call    scanf
```

Aquí estamos haciendo un *scanf* para guardar el tamaño de la pirámide, pues bien, lo que haremos será **renombrar** *var_4* por *tamanho_pir*, de esta forma lo tenemos controlado. Recuerdo que para renombrar hay que seleccionar lo que queremos renombrar y pulsar *N*, entonces cambiamos ahí el nombre.

```
.text:004012D9 mov     [ebp+var_8], 1
```

Movemos a *var_8* un *1* antes de entrar en el bucle, curioso, ¿eh? Pues sí, lo es, seguramente sea el contador, vamos, tiene todas las papeletas para serlo. ¿Cómo lo sabemos?

Pues bien, hacemos clic en *var_8* y nos encontramos con que nos señala también una parte del código dentro del bucle.

Observamos que más tarde lo compara con *tamanho_pir* para salir del bucle, por lo tanto será un contador que suponemos que irá incrementando.

Renombramos *var_8* por *contador*.

```
.text:004012E0 mov     eax, [ebp+contador]
.text:004012E3 cmp     eax, [ebp+tamanho_pir]
.text:004012E6 jg      short locret_4
```

Estudiamos esta parte dentro del bucle antes de poner el código fuente de esta zona ya que nos interesa.

He hecho referencia a esta parte en mi última frase anterior (la de esta línea no, la de antes de pegar este código), y por lo tanto pensamos que el contador es algo que va incrementándose, para ello hacemos clic en *contador*.

Vamos a la parte más baja del bucle y observamos:

```
.text:00401342 lea     eax, [ebp+contador]
.text:00401345 inc     dword ptr [eax]
```

Pues sí, parece que se va incrementando, por lo tanto estamos ante un bucle que puede ser por ejemplo de tipo *for*.

Nos queda:

```
printf("tamaño de la piramide: ");
```

```
scanf("%d",&tamanho_pir);
```

```
for(contador=1;contador<=tamanho_pir;contador++) {
```

Por ahora tan solo es una hipótesis la forma de incrementar el contador, ya se verá.

Más contadores (O eso parece)

```
.text:004012E8 mov     [ebp+var_C], 1
```

Movemos un *1* a una variable llamada *var_C*, ¿Qué hará? Por ahora no la renombraremos.

Vemos que entramos en un bucle (Las líneas anchas).

```
.text:004012EF mov     eax, [ebp+contador]
.text:004012F2 mov     edx, [ebp+tamanho_pir]
.text:004012F5 sub     edx, eax
.text:004012F7 mov     eax, edx
.text:004012F9 cmp     eax, [ebp+var_C]
.text:004012FC jl      short loc_401311
```

Lo que hacemos es calcular la diferencia entre el *contador* y *tamanho_pir*.

Más tarde movemos a *eax* este resultado y lo comparamos con *var_C*, en el caso de que sea menor, es decir, que la diferencia entre *contador* y *tamanho_pir* sea menor que 1. (Ahora mismo *var_C* está en *1*, al menos al iniciar) se saltaría al **311*.

```
.text:004012FE mov     dword ptr [esp], offset asc_40301B ; " "
.text:00401305 call    printf
.text:0040130A lea     eax, [ebp+var_C]
.text:0040130D inc     dword ptr [eax]
.text:0040130F jmp     short loc_4012EF
```

Si no es menor, entonces lo que hacemos es imprimir un espacio con *printf* e incrementar *var_C*.

Para movermos a la cabeza del bucle de *var_C*.

Lo que hay en el **311*

```
.text:00401311 mov     [ebp+var_C], 1
```

Aquí estamos moviendo un *1* a nuestro *var_C*.

```
.text:00401318 mov     eax, [ebp+contador]
.text:0040131B add     eax, eax
.text:0040131D dec     eax
```

Cargamos en *eax* el contador y lo multiplicamos por dos.

Entonces le decrementamos uno. De esta forma tenemos un número impar en *eax*.

```
.text:0040131E cmp     eax, [ebp+var_C]
.text:00401321 jl      short loc_401336
```

Compara *eax* con *var_C*, y en el caso de que sea más bajo, es decir, que este número impar sea más bajo que *var_C* saltaríamos fuera de este bucle.

```
.text:00401323 mov     dword ptr [esp], offset asc_40301D ; "*"
.text:0040132A call    printf
.text:0040132F lea     eax, [ebp+var_C]
.text:00401332 inc     dword ptr [eax]
.text:00401334 jmp     short loc_401318
```

En el caso de que no sea así lo que haremos será imprimir un asterisco (Que será el cuerpo de nuestra pirámide) y además incrementar *var_C*.

Nueva línea (O nuevo nivel en la pirámide)

```
.text:00401336 mov     dword ptr [esp], offset asc_40301F ; "\n"
.text:0040133D call    printf
.text:00401342 lea     eax, [ebp+contador]
.text:00401345 inc     dword ptr [eax]
.text:00401347 jmp     short loc_4012E0
```

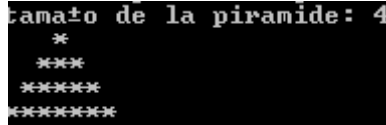
Imprimimos un salto de línea con *printf* e incrementamos nuestro *contador*, entonces, realmente, nuestro contador lo que hace es contar las plantas de la pirámide, es decir, que en *tamanho_pir* tenemos el tamaño de nuestra pirámide que queremos.

Escribiendo el código

He ido un poco rápido, lo sé, pero ahora vamos a ver cómo caigo en errores, uno tras otro y cómo los soluciono xD.

Ah, antes de nada, voy a correr el programa, a ver si es lo que nos esperamos.

Em, lo corro y se me cierra, tengo una ultravisión rápida que me permite verlo antes de que se cierre, pero bueno, lo que haré será irme a *inicio* → *ejecutar* → *cmd* (Si no os aparece ejecutar pulsar *Windows+R*) y cargarlo desde ahí.



Detalle de mi Spectrum

Pues sí, es justo lo que nos esperábamos, incluso hemos acertado en lo del número impar, qué cracks.

Comencemos renombrando primero *var_C* por otra cosa, como por ejemplo: *pos_linea*.

Por cierto, observamos además que no hay ningún valor de retorno al final de todo el *main*, por lo que es de tipo *void*. (Nos saltará un warning al compilar, pero no pasa nada).

```
.text:00401290 pos_linea      = dword ptr -0Ch
.text:00401290 contador   = dword ptr -8
.text:00401290 tamanho_pir = dword ptr -4
```

Código fuente en la siguiente página.

Código fuente:

/ Ejercicio 19 Practica 2 */*

```
void main(void)
{
    int tamanho_pir,contador,pos_linea;
    printf("tamaño de la piramide: ");
    scanf("%d",&tamanho_pir);
    for(contador=1;contador<=tamanho_pir;contador++) {
        for(pos_linea=1;(tamanho_pir-contador)>=pos_linea;pos_linea++) {
            printf(" ");
        }
        for(pos_linea=1;(contador*2-1)>=pos_linea;pos_linea++) {
            printf("*");
        }
        printf("\n");
    }
}
```

Turbodiff

Hacemos *turbodiff* al igual que con la *practica1* y nos da *identical*, ¡sonríe! ☺