

## REVERSEANDO Y PRACTICANDO

Antes de seguir con las practicas quería mostrar un par de detalles que me faltó explicar y que serán necesarios en estos ejemplos de reversing.

Lo primero es como se ingresan en C argumentos por consola.

```
#include <windows.h>
#include <stdio.h>

int main(int argc, char* argv[])
{
    int i;

    if (argc > 1){
        printf("Cantidad de argumentos es %d\n", argc);

        for (i=0; i<argc; i++) printf("argumentos son %s\n", argv[i]);
    }
    else {
        printf("Tipear argumentos para probar\n");
    }

    getchar();

}
```

C provee un mecanismo para pasar argumentos desde la línea de comandos al programa que se va a ejecutar. Cuando el programa comienza su ejecución, la rutina main es llamada con dos argumentos: un contador y un puntero a un array de strings. El contador es llamado por convención argc y el apuntador argv. El uso de de argv es un poco truculento. Dado que argv es un puntero a un array de strings, la primera cadena de caracteres es referenciada por argv[0] (o \*argv). La segunda cadena is referenciada por argv[1] (o \*(argv + 1)), la tercera por argv[2], y así sucesivamente.

La primera cadena de caracteres, **argv[0]**, contiene el nombre del programa. Los argumentos comienzan realmente con **argv[1]**.

Vemos que las argc y argv no necesitan ser declaradas como dijimos siempre el compilador las maneja, si vemos este código en el IDA.

```

; Attributes: up-based frame

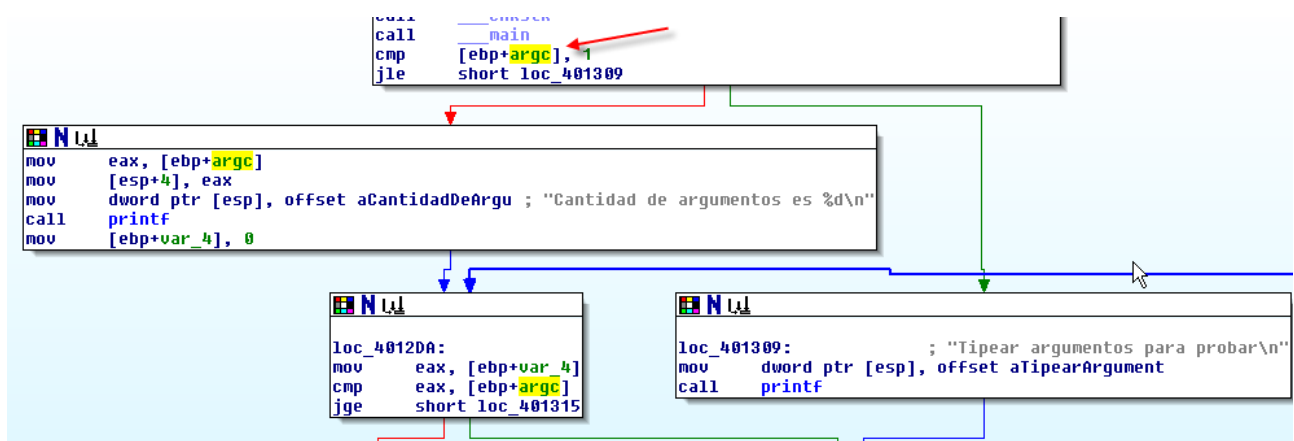
; int __cdecl main(int argc, const char **argv, const char **envp)
_main proc near

var_8= dword ptr -8
var_4= dword ptr -4
argc= dword ptr 8
argv= dword ptr 0Ch
envp= dword ptr 10h

push    ebp
mov     ebp, esp
sub     esp, 18h
and     esp, 0FFFFFFF0h
mov     eax, 0
add     eax, 0Fh
add     eax, 0Fh
chr     eax, h

```

Vemos que el main ya tiene como argumentos **argc** y **argv** que son dos dwords ya que el primero es un contador o sea un int y el segundo es un puntero a un array de punteros a strings, por lo tanto también es un int.

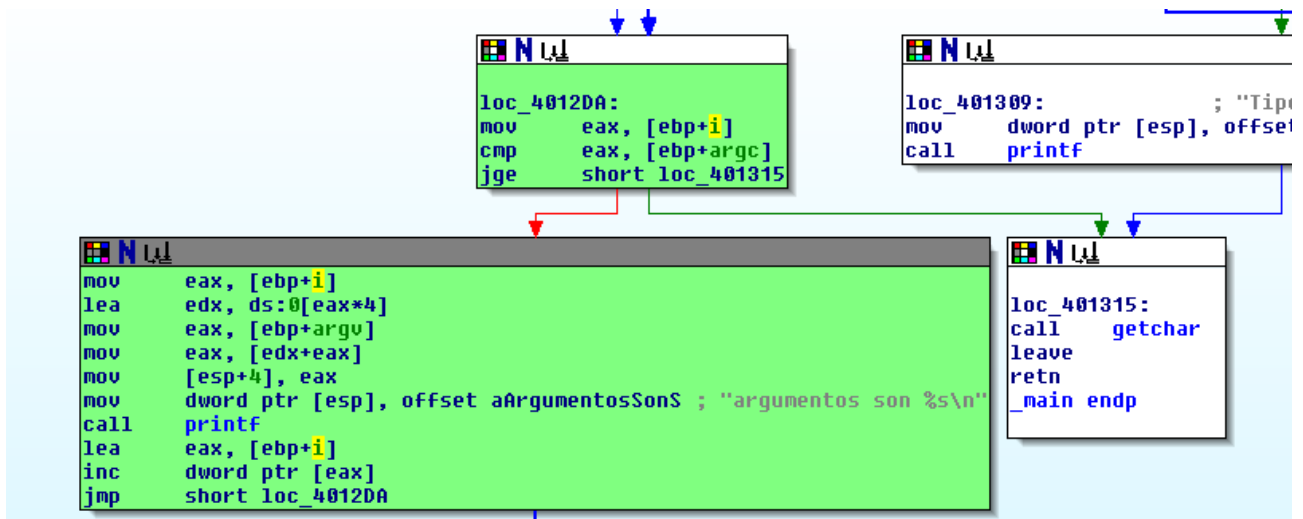


Vemos que compara la cantidad de argumentos con uno, ya que siempre el primer argumento es el mismo nombre del ejecutable o sea que siempre sera 1 como mínimo.

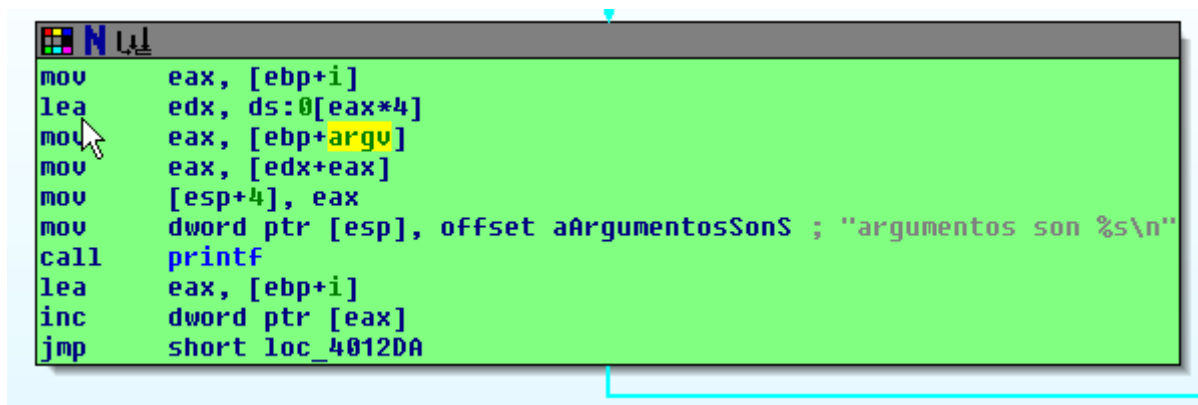
Vemos que si es menor o igual que uno tomara el camino de la flecha verde y me pedirá que tipee argumentos para probar, mientras que si tipee alguno, ira por el camino rojo y imprimirá cuantos argumentos tipeamos.



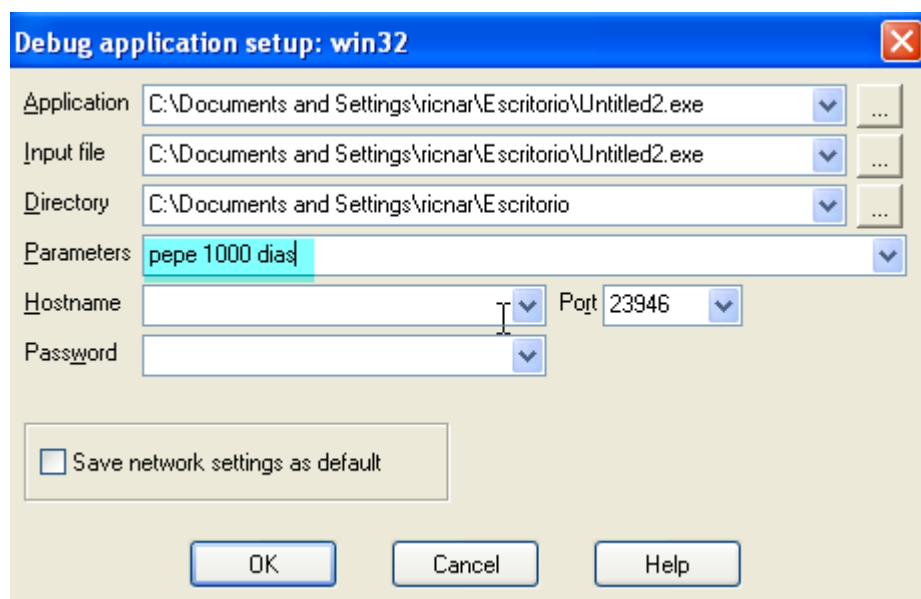
Luego pondrá la **var\_4** a cero que es el contador del for la puedo renombrar a **i**.



Allí esta coloreado el loop se mantendrá loopeando mientras **i** sea menor que **argc**.



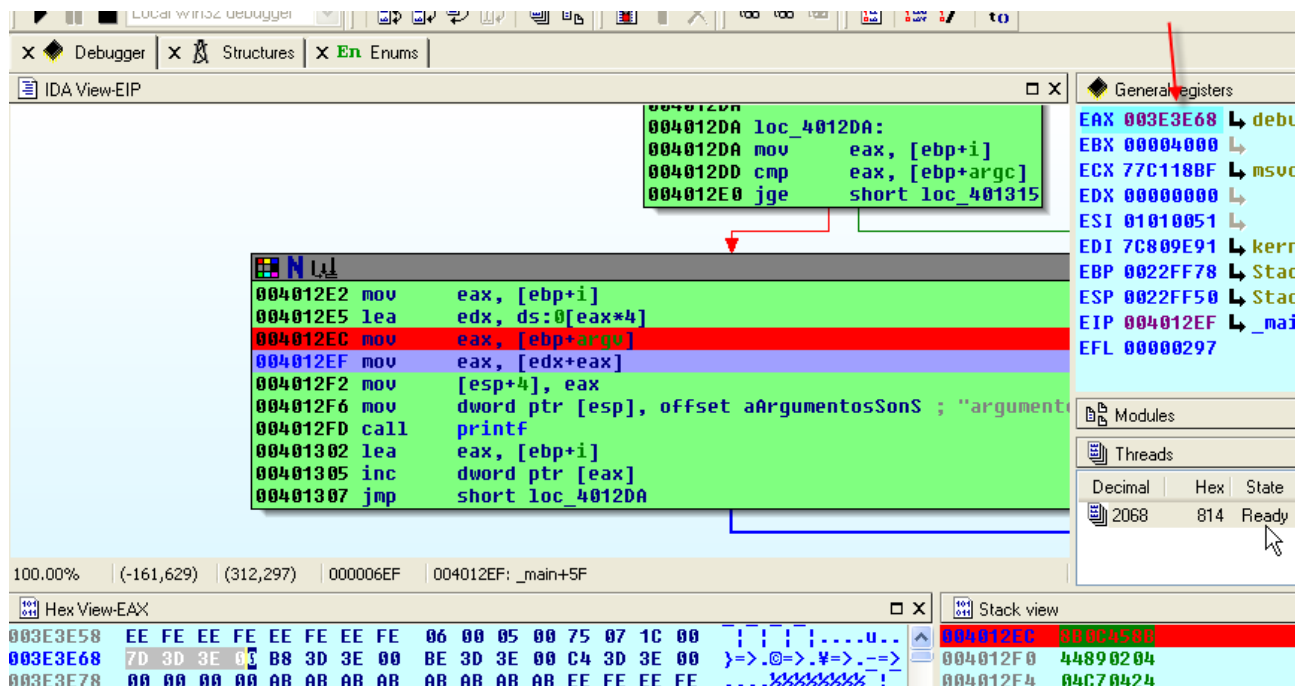
Vemos como toma el contador **i** y lo multiplica por 4 dentro del LEA, o sea que EDX se usara para sumar al puntero, y poder recorrer todos los punteros a las strings, poniendo un BP aquí y tipeando algunos argumentos.



En IDA en las Opciones del proceso o Process Options colocamos los argumentos y arrancamos el debugger.



Vemos que luego de imprimir la cantidad de argumentos para en el BP, a EAX pasa el puntero al array, vemos en el dump que el contenido de EAX o sea el primer campo, es otro puntero.



Como EDX es cero ya que es el primer ciclo del loop

```
mov     eax, [edx+eax]
```

EAX tendrá el valor de dicho puntero que apuntara a la primera string de los argumentos.

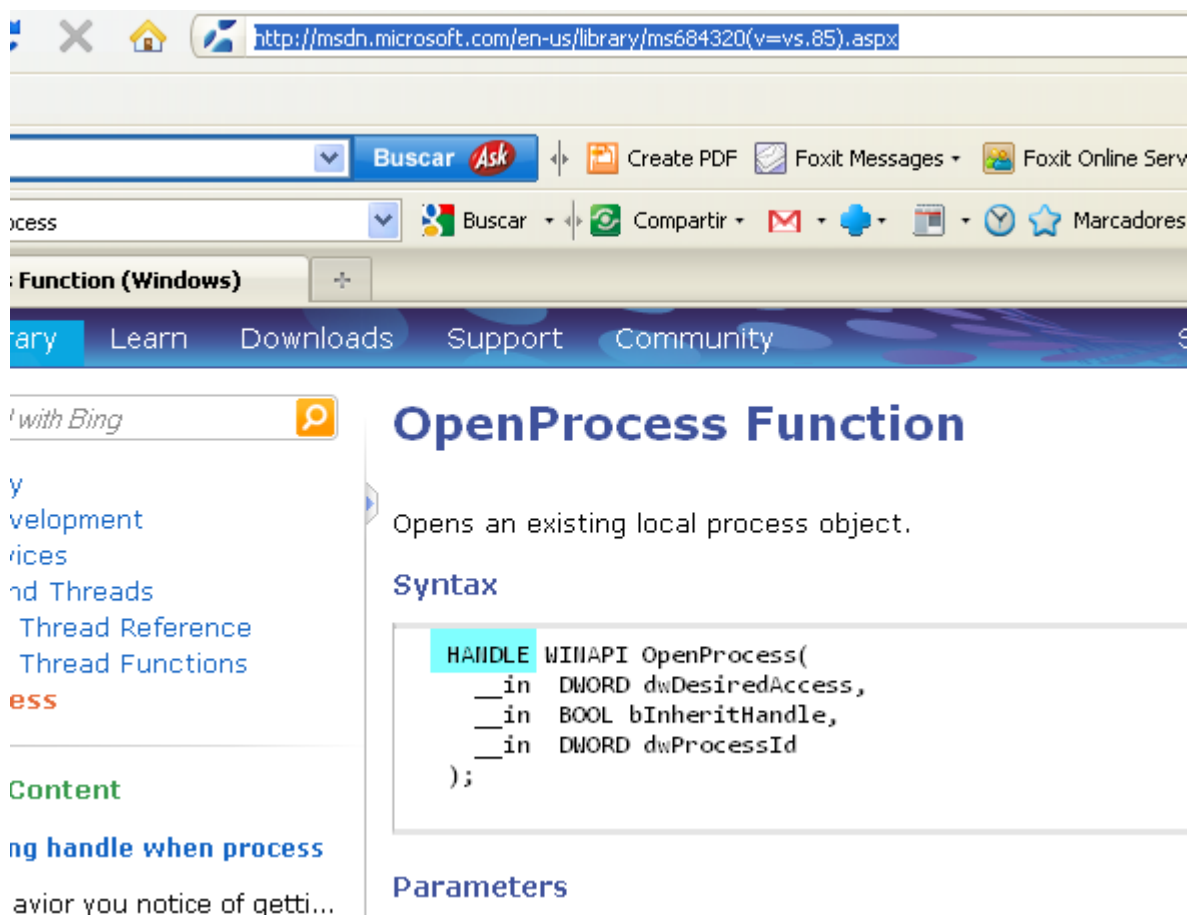


O sea que confirmamos que **argv** es un puntero a un array de punteros a strings.

Bueno quería mostrar un poco como se ingresaban argumentos por consola, ya que algunos ejemplos los usan.

Allí van 3 ejemplos sencillos y uno mas complejo, en este ultimo aparece un tipo de datos avanzado llamado HANDLE, se usa para casos muy específicos como el valor que devuelve OpenProcess en este caso, si ponemos que sea un **int** no funcionara.

**HANDLE** phandle = OpenProcess(PROCESS\_ALL\_ACCESS,0,pid);

A screenshot of a web browser displaying the MSDN page for the OpenProcess function. The browser's address bar shows the URL 'http://msdn.microsoft.com/en-us/library/ms684320(v=vs.85).aspx'. The page has a blue header with navigation links like 'Library', 'Learn', 'Downloads', 'Support', and 'Community'. On the left, there's a sidebar with a search bar and a list of categories including 'Development', 'Processes', 'Threads', 'Thread Reference', 'Thread Functions', and 'Process'. The main content area is titled 'OpenProcess Function' in large blue letters. Below the title, it says 'Opens an existing local process object.' and 'Syntax'. The syntax is shown in a code block with a light blue background, featuring the signature: 'HANDLE WINAPI OpenProcess( \_\_in DWORD dwDesiredAccess, \_\_in BOOL bInheritHandle, \_\_in DWORD dwProcessId );'. Below the code block, the 'Parameters' section is partially visible.

Allí vemos que la api esta definida de esa forma y devuelve un tipo HANDLE, el resto es conocido, como ayuda les diré que antes de pedir el handle a OpenProcess se llama a una funcion que eleva los privilegios de nuestro proceso, si no muchas veces no tendrá los suficientes privilegios para leer la memoria de ciertos procesos.

Hasta la practica 2 y que se diviertan.  
Ricardo Narvaja