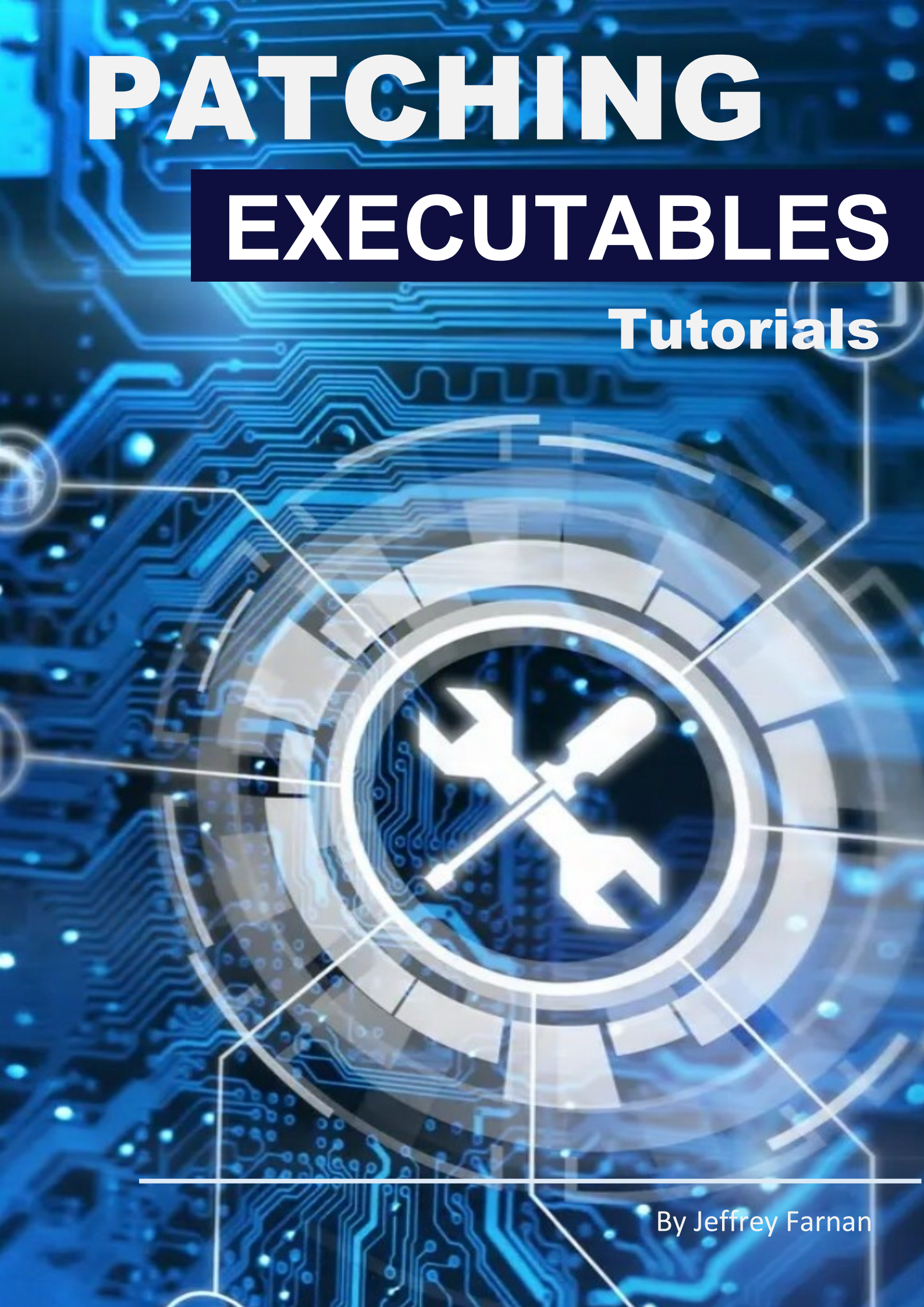
PATCHING EXECUTABLES

Tutorials



By Jeffrey Farnan

INTRODUCTION

This is a tutorial showing basic skills needed to patch an executable in x64dbg. Patching is changing or modifying the assembly instructions to improve the executable, fix problems or modify it to do something different.

The code we will be using is a simple C program to add two numbers in a function and print out the result. Here is the assembly code for this program:

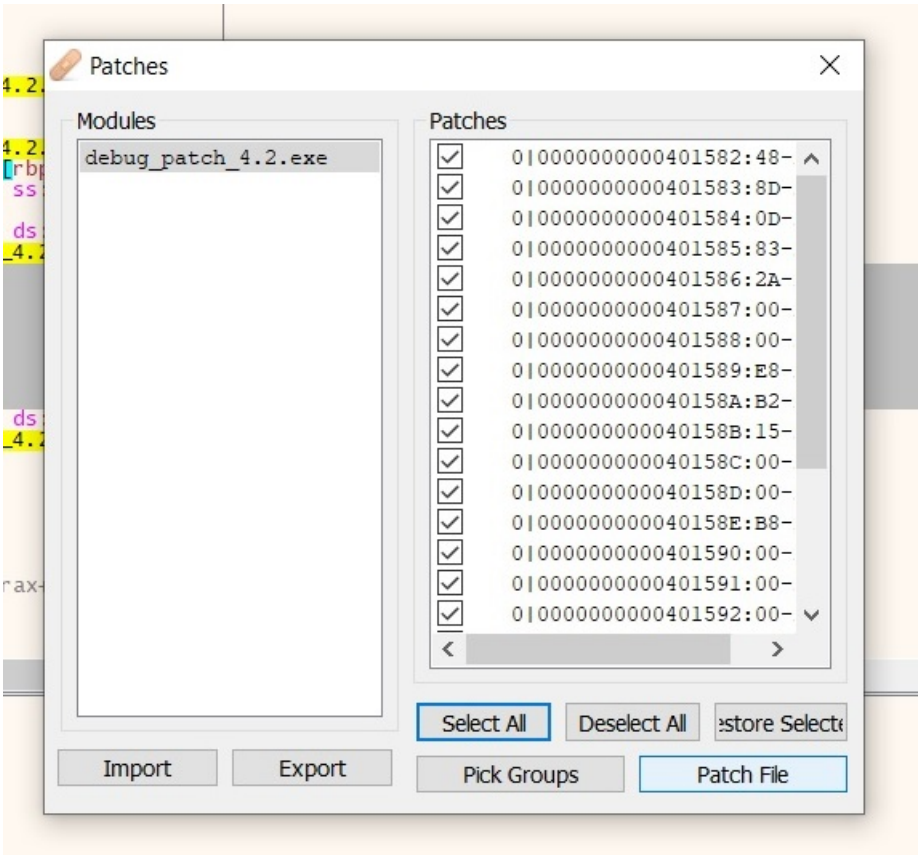
```
90                  nop
55                  push rbp                                add_func
48 89 E5            mov rbp,rsp
48 83 EC 10         sub rsp,10
89 4D 10            mov dword ptr ss:[rbp+10],ecx           num1 = ecx
89 55 18            mov dword ptr ss:[rbp+18],edx           num2 = edx
8B 55 10            mov edx,dword ptr ss:[rbp+10]
8B 45 18            mov eax,dword ptr ss:[rbp+18]
01 D0               add eax,edx                             num1 + num2
89 45 FC            mov dword ptr ss:[rbp-4],eax
8B 45 FC            mov eax,dword ptr ss:[rbp-4]            return eax
48 83 C4 10         add rsp,10
5D                  pop rbp
C3                  ret                                     end
55                  push rbp                                main(){
48 89 E5            mov rbp,rsp
48 83 EC 30         sub rsp,30
E8 C1 0B 00 00      call <debug_patch_2.main>
BA 05 00 00 00      mov edx,5                               int num1 =5;
B9 05 00 00 00      mov ecx,5                               int num2 =5;
E8 C2 FF FF FF      call <debug_patch_2.add_func>           add_function(num1, num2);
89 45 FC            mov dword ptr ss:[rbp-4],eax            int result =
8B 45 FC            mov eax,dword ptr ss:[rbp-4]
89 C2               mov edx,eax
48 8D 0D 83 2A 00   lea rcx,qword ptr ds:[404000]           0x0000000000404000:"5 + 5 = %d\n"
E8 B6 15 00 00      call <debug_patch_2.printf>             printf("5 + 5 = %d\n", result);
48 8D 0D 83 2A 00   lea rcx,qword ptr ds:[40400C]           0x000000000040400C:"pause"
E8 B2 15 00 00      call <debug_patch_2.system>             system("pause");
B8 00 00 00 00      mov eax,0                               return 0;
48 83 C4 30         add rsp,30                              }
5D                  pop rbp
C3                  ret                                     end
90                  nop
```

The main method is on the bottom and the add function on the top. When the program is run the output is as follows.

```
5 + 5 = 10
Press any key to continue . . .
```

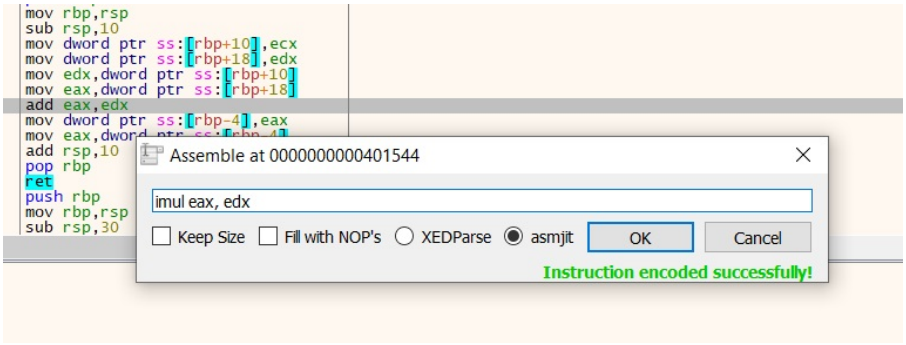
SAVING A PATCH

Saving a patch is easy, when you are done modifying the assembly go to **File – Patch File**. A selection of modifications appear which you have made to the file, select the modifications you want by clicking the checkbox or press **Select All** button and click **Patch File** which will allow to save the file as a new executable.



PATCHING STEP 1

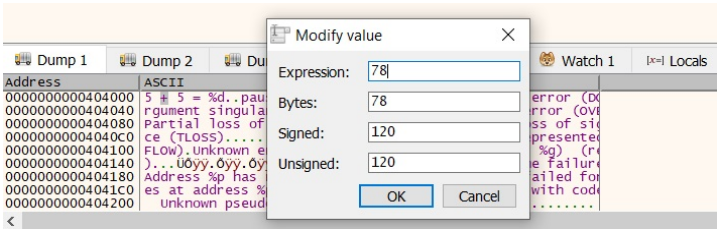
The first patch is very simple, rather than add too variables in the function we can multiply and return the new value to the *printf* statement. This is a fairly basic operations as all I have to do is change the add instruction to a multiple instruction, which in assemble its **imul**.



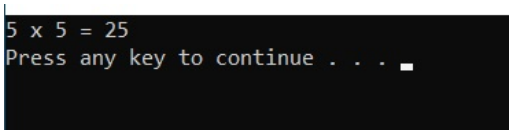
I select the line of assemble and click on the spacebar to modify an assemble instruction. I save my new patch and run the program this time instead of 10 as the answer we have 25.

The last thing we need to do is change the plus symbol to a multiply symbol in the output. We go back into the assembly.

Click on where the String is stored, and right click **Follow in Dump**. To modify the value select the plus symbol and press the spacebar to change it. In hex the “x” is 78.

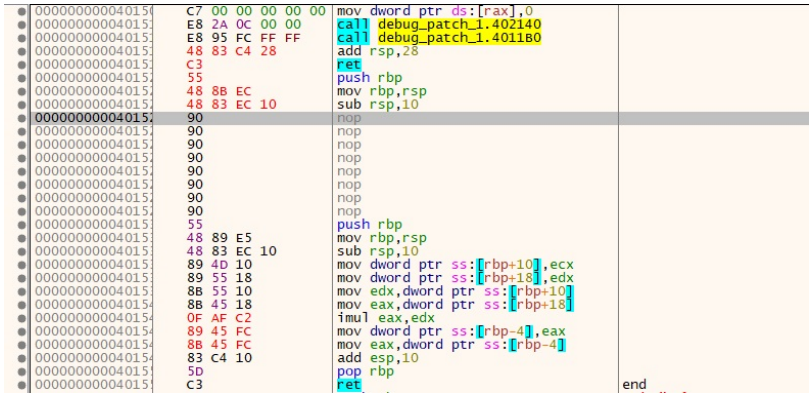


Save the patches again and run the program, we have the following output:



PATCHING STEP 2

For the next set of patches I want to add a *printf* statement to the function. To make more room for extra instructions, I have to move up assemble instructions upwards.



I do this by copying each instruction line by line and pasting the instructions where I have space above.

I copy the *printf* instructions from main into the empty *nop* spaces which is available.

I have successfully moved the *printf* instructions into the function with some extra space filled with *nop's*. As this function is already printing out our mutilation result, we don't need to pass this result to the main method, so insert a **mov eax, 0** so nothing gets returned.

PATCHING STEP 2

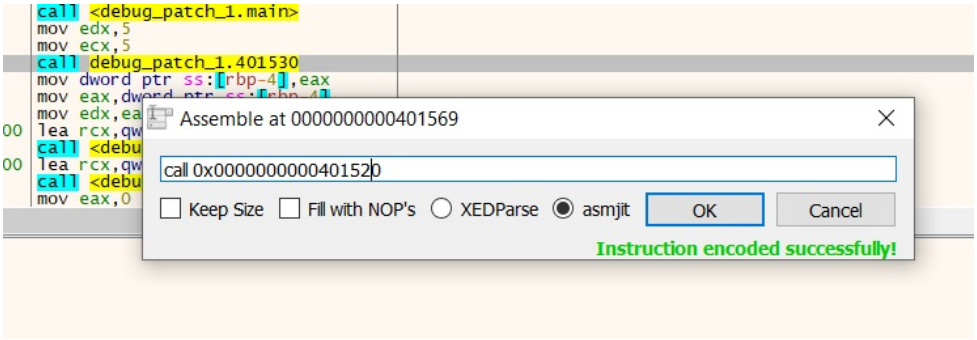
```
nop
push rbp                                    add_func
mov rbp,rsp
sub rsp,10
imul eax,edx
mov edx,eax
lea rcx,qword ptr ds:[404000]               0x0000000000404000:"5 x 5 = %d\n"
call <patch2_test.printf>
mov eax,0                                   return 0
add esp,10
pop rbp
ret                                         end_add_func
```

The function has now changed, the two arguments are multiplied and the result is printed out in its own printout statement.

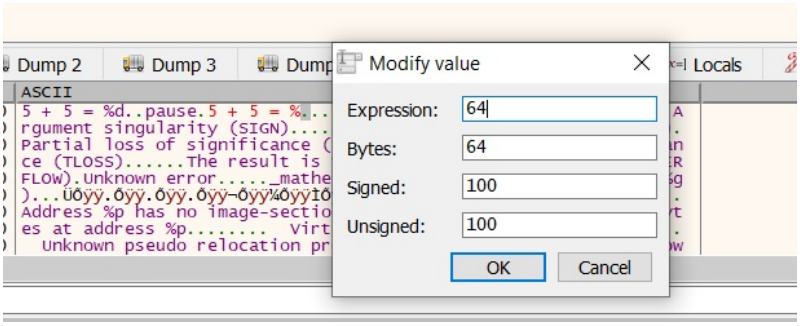
The main method

My next task is to change the function address on the main method, so the function is called and the output result is displayed.

Since we have moved this function we must fix the calling address in which is now **401520**, I can modify it by pressing the spacebar on the call instruction.

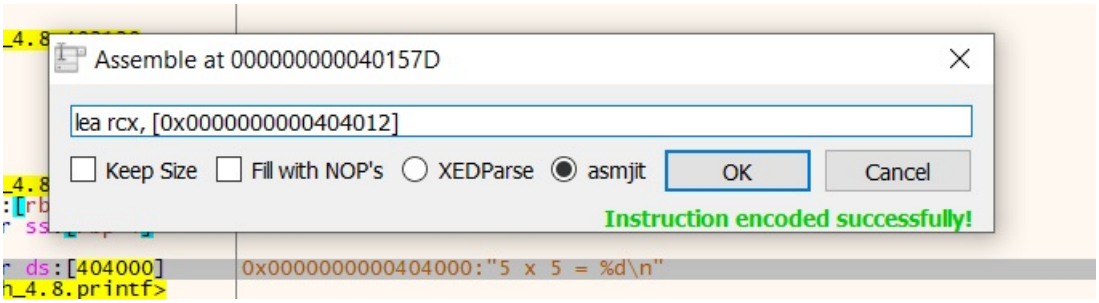


I also need to modify the string in the *printf* statement to change it back to a plus symbol instead of multiply symbol. I have to create a second string in the dump and change this string address.

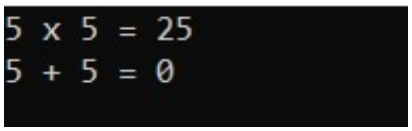


PATCHING STEP 2

I need to change the address of this location so the *printf* will call this string.



We save this patch and run the program. This function returns zero to the main function. We will fix this in the next patch.



PATCHING STEP 3

The third and final patch is a bit more complex. As we took out the add instructions in the function in the first patch, this patch we will now put these add instructions in the main method.

The first thing we do is create free space in the main function so we can add extra instructions.

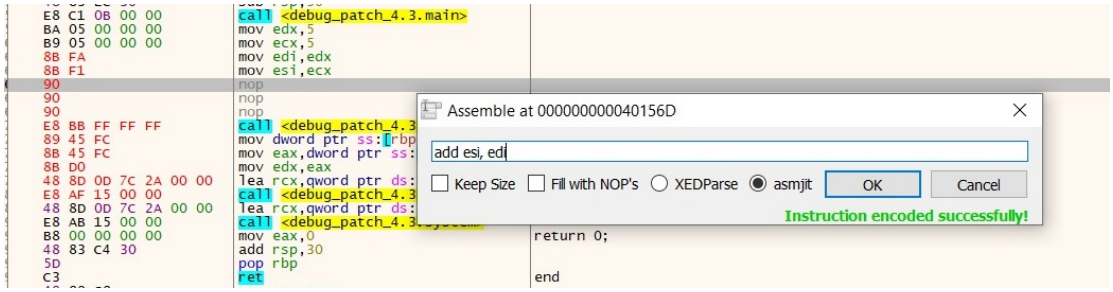
```
55                      push rbp
48 89 E5                mov rbp,rsp
48 83 EC 30             sub rsp,30
E8 C1 0B 00 00          call debug_patch_4.2.402120
BA 05 00 00 00          mov edx,5
B9 05 00 00 00          mov ecx,5
E8 C2 FF FF FF          call debug_patch_4.2.401530
89 45 FC                mov dword ptr ss:[rbp-4],eax
8B 45 FC                mov eax,dword ptr ss:[rbp-4]
89 C2                   mov edx,eax
48 8D 0D 83 2A 00 00    lea rcx,qword ptr ds:[404012]   0x0000000000404000:"5 + 5 = %d\n"
E8 B6 15 00 00          call <debug_patch_4.2.printf>
90                      nop
90                      nop
90                      nop
90                      nop
90                      nop
90                      nop
90                      nop
48 8D 0D 7C 2A 00 00    lea rcx,qword ptr ds:[40400C]   0x000000000040400C:"pause"
E8 AB 15 00 00          call <debug_patch_4.2.system>
B8 00 00 00 00          mov eax,0
48 83 C4 30             add rsp,30
5D                      pop rbp
C3                      ret
```

I need to move instructions down so I have extra space at the top of the function.

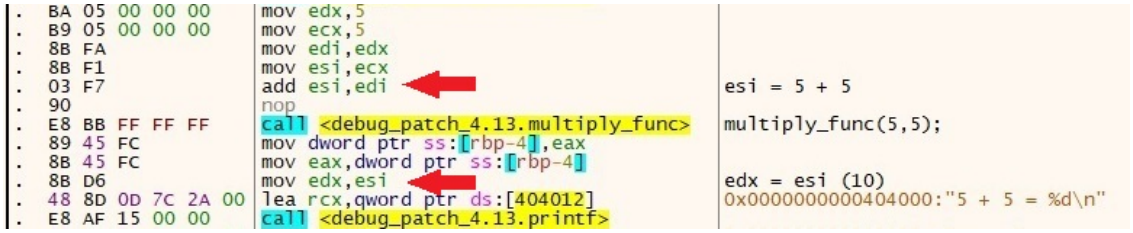
```
55                      push rbp                        main
48 89 E5                mov rbp,rsp
48 83 EC 30             sub rsp,30
E8 C1 0B 00 00          call <debug_patch_4.3.main>
BA 05 00 00 00          mov edx,5
B9 05 00 00 00          mov ecx,5
90                      nop
90                      nop
90                      nop
90                      nop
90                      nop
90                      nop
90                      nop
E8 BB FF FF FF          call <debug_patch_4.3.func>
89 45 FC                mov dword ptr ss:[rbp-4],eax
8B 45 FC                mov eax,dword ptr ss:[rbp-4]
8B D0                   mov edx,eax
48 8D 0D 7C 2A 00 00    lea rcx,qword ptr ds:[404012]   0x0000000000404000:"5 + 5 = %d\n"
E8 AF 15 00 00          call <debug_patch_4.3.printf>
48 8D 0D 7C 2A 00 00    lea rcx,qword ptr ds:[40400C]   0x000000000040400C:"pause"
E8 AB 15 00 00          call <debug_patch_4.3.system>
B8 00 00 00 00          mov eax,0                       return 0;
48 83 C4 30             add rsp,30
5D                      pop rbp
C3                      ret                             end
```

We can see we have too arguments ECX and EDX both holding 5. Both of these registers are needed for the calling function. What we can do is move EDX and ECX into the EDI and ESI registers plus moving the values they hold. We can than add the values in ESI and EDI.

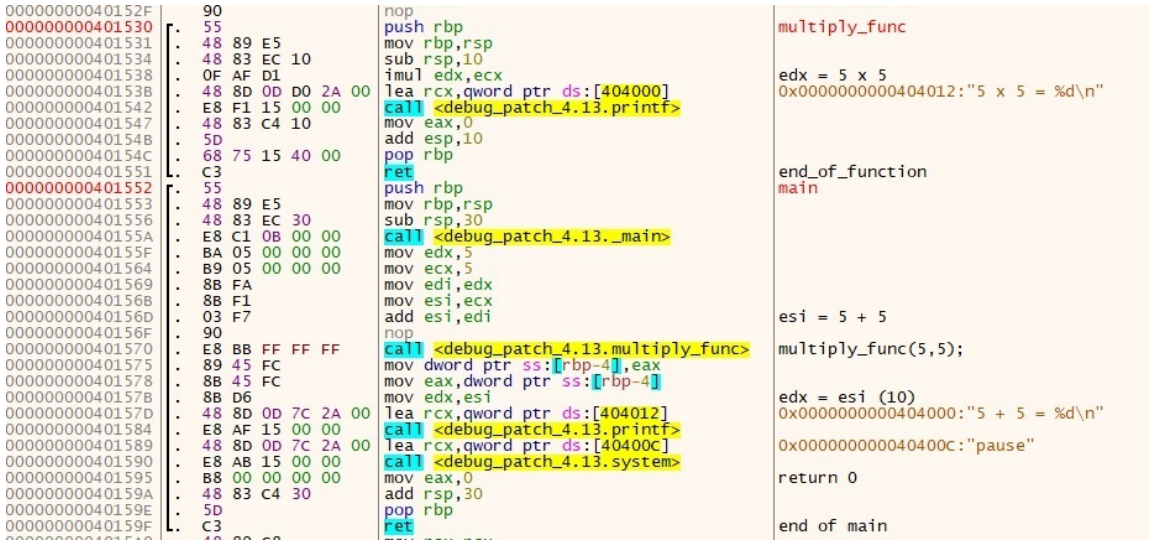
PATCHING STEP 3



When you add ESI and EDI the added value is stored in ESI. ESI can be moved into EDX as the result and printed out in the output string.



We can see here the full modified code.



When I run the program the following is the output to screen: