

# 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	num1 = ecx
89 4D 10		mov dword ptr ss:[rbp+10], ecx	num2 = edx
89 55 18		mov dword ptr ss:[rbp+18], edx	
8B 55 10		mov edx, dword ptr ss:[rbp+10]	
8B 45 18		mov eax, dword ptr ss:[rbp+18]	num1 + num2
01 D0		add eax, edx	
89 45 FC		mov dword ptr ss:[rbp-4], eax	return eax
8B 45 FC		mov eax, dword ptr ss:[rbp-4]	
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>	int num1 = 5;
BA 05 00 00 00		mov edx, 5	int num2 = 5;
B9 05 00 00 00		mov ecx, 5	add_function(num1, num2);
E8 C2 FF FF FF		call <debug_patch_2.add_func>	int result =
89 45 FC		mov dword ptr ss:[rbp-4], eax	
8B 45 FC		mov eax, dword ptr ss:[rbp-4]	0x0000000000404000:"5 + 5 = %d\n"
89 C2		mov edx, eax	printf("5 + 5 = %d\n", result);
48 8D 0D 83 2A 00		lea rcx, qword ptr ds:[404000]	0x000000000040400c:"pause"
E8 B6 15 00 00		call <debug_patch_2.printf>	system("pause");
48 8D 0D 83 2A 00		lea rcx, qword ptr ds:[40400c]	return 0;
E8 B2 15 00 00		call <debug_patch_2.system>	}
B8 00 00 00 00		mov eax, 0	
48 83 C4 30		add rsp, 30	
5D		pop rbp	
C3		ret	end
90		nop	

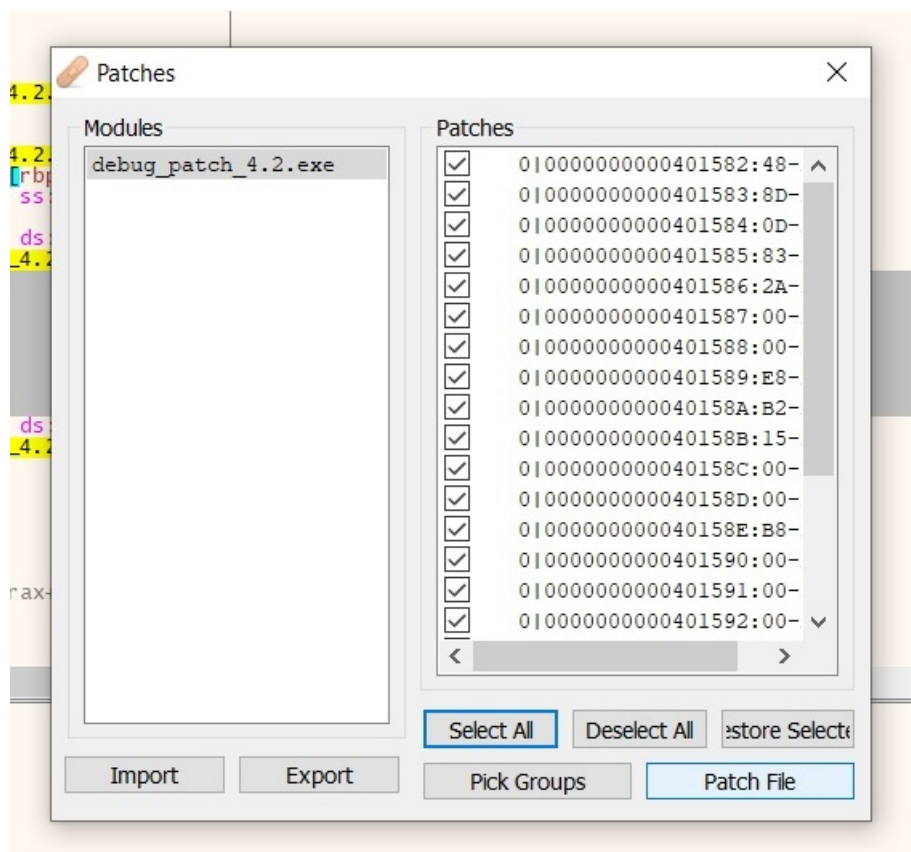
The main method is on the bottom and the add function on the top. When the program is run the output is as follows.

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



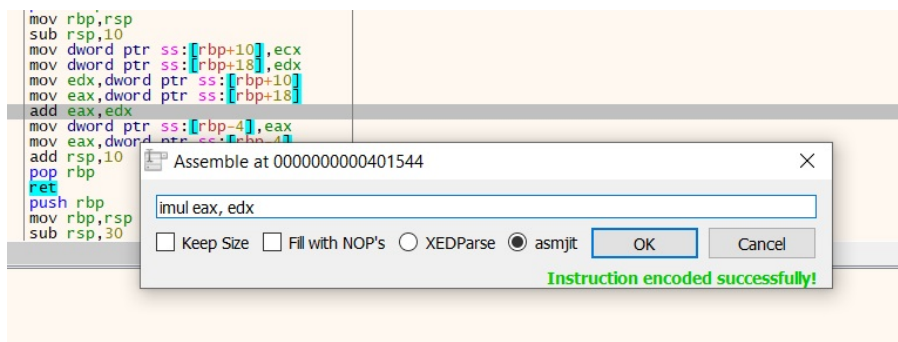
## SAVING A PATCH

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**.

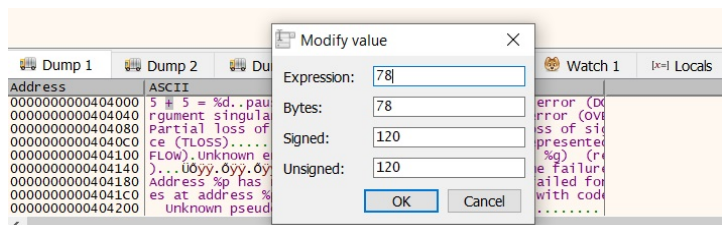


I select the line of assemble and click on the spacebar to modify an assemble instruction. I save my new patch and run the program this time instead of 10 as the answer we have 25.

```
5 + 5 = 25
Press any key to continue . . .
```

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:

```
5 x 5 = 25
Press any key to continue . . .
```

## 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.

0000000000040151	C7 00 00 00 00 00	mov dword ptr ds:[rax],0	
0000000000040151	E8 2A 0C 00 00 00	call debug_patch_1.402140	
0000000000040151	E8 95 FC FF FF	call debug_patch_1.401180	
0000000000040151	48 83 C4 28	add rsp,28	
0000000000040151	C3	ret	
0000000000040151	55	push rbp	
0000000000040151	48 8B EC	mov rbp,esp	
0000000000040151	48 83 EC 10	sub rsp,10	
0000000000040151	90	nop	
0000000000040151	90	nop	
0000000000040151	90	nop	
0000000000040151	90	nop	
0000000000040151	90	nop	
0000000000040151	90	nop	
0000000000040151	90	nop	
0000000000040151	55	push rbp	
0000000000040151	48 89 E5	mov rbp,esp	
0000000000040151	48 83 EC 10	sub rsp,10	
0000000000040151	89 4D 10	mov dword ptr ss:[rbp+10],ecx	
0000000000040151	89 55 18	mov dword ptr ss:[rbp+18],edx	
0000000000040151	8B 55 10	mov edx,dword ptr ss:[rbp+10]	
0000000000040151	8B 45 18	mov eax,dword ptr ss:[rbp+18]	
0000000000040151	0F AF C2	imul eax,edx	
0000000000040151	89 45 FC	mov dword ptr ss:[rbp-4],eax	
0000000000040151	8B 45 FC	mov eax,dword ptr ss:[rbp-4]	
0000000000040151	83 C4 10	add esp,10	
0000000000040151	5D	pop rbp	
0000000000040151	C3	ret	end

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

48 83 C4 28	add rsp,28	
C3	ret	
55	push rbp	
48 8B EC	mov rbp,esp	
48 83 EC 10	sub rsp,10	
89 4D 0A	mov dword ptr ss:[rbp+4],ecx	
89 55 12	mov dword ptr ss:[rbp+12],edx	
8B 55 0A	mov edx,dword ptr ss:[rbp+4]	
8B 45 12	mov eax,dword ptr ss:[rbp+12]	
0F AF C2	imul eax,edx	
8B D0	mov edx,eax	
48 8D 0D C0 2A 00	lea rcx,qword ptr ds:[404000]	0x0000000000040400:"5 x 5 = %d\n"
E8 F3 15 00 00	call <debug_patch_1.printf>	
90	nop	
90	nop	
89 45 FC	mov dword ptr ss:[rbp-4],eax	
8B 45 FC	mov eax,dword ptr ss:[rbp-4]	
83 C4 10	add esp,10	
5D	pop rbp	
C3	ret	end
55	push rbp	main() {
48 89 E5	mov rbp,esp	
48 83 EC 30	sub rsp,30	
E8 C1 0B 00 00	call <debug_patch_1.main>	
BA 05 00 00 00	mov edx,5	
B9 05 00 00 00	mov ecx,5	
E8 C2 FF FF FF	call debug_patch_1.401530	
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	lea rcx,qword ptr ds:[404000]	0x0000000000040400:"5 x 5 = %d\n"
E8 B6 15 00 00	call <debug_patch_1.printf>	
48 8D 0D 83 2A 00	lea rcx,qword ptr ds:[40400C]	0x000000000004040C:"pause"
E8 B2 15 00 00	call <debug_patch_1.system>	
B8 00 00 00 00	mov eax,0	
48 83 C4 30	add rsp,30	
5D	pop rbp	
C3	ret	

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

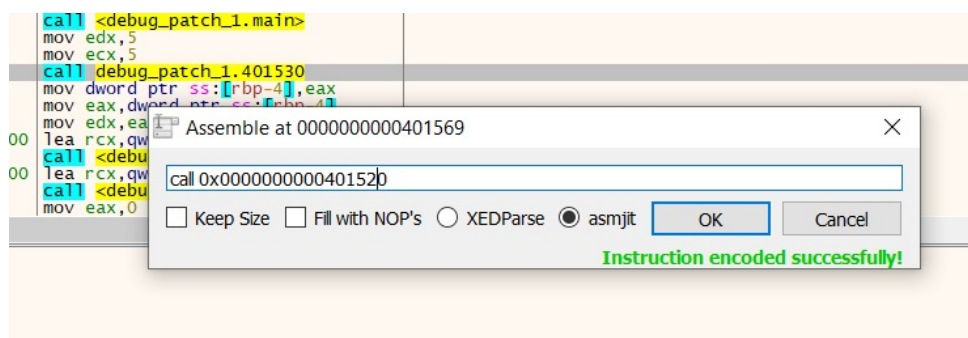
<pre> nop push rbp mov rbp, rsp sub rsp, 10 imul eax, edx mov edx, eax lea rcx, qword ptr ds:[404000] call &lt;patch2_test.printf&gt; mov eax, 0 add esp, 10 pop rbp ret </pre>	<pre> add_func  0x0000000000404000:"5 x 5 = %d\n"  return 0  end_add_func </pre>
---	--

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

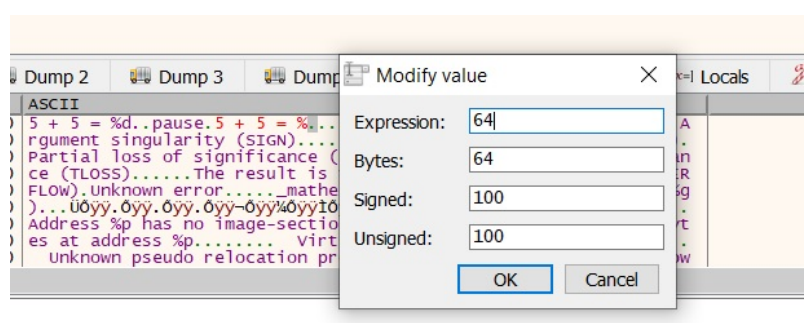
### The main method

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.

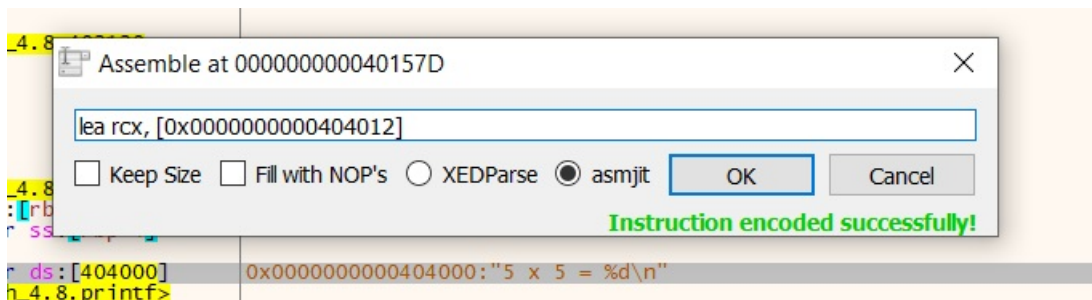


I also need to modify the string in the `printf` statement to change it back to a plus symbol instead of multiply symbol. I have to create a second string in the dump and change this string address.



## 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.

```
5 x 5 = 25
5 + 5 = 0
```

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

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

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



## PATCHING STEP 3

The screenshot shows a debugger window with assembly code. A dialog box titled "Assemble at 000000000040156D" is open, showing the instruction "add esi,edi". The dialog has options for "Keep Size", "Fill with NOP's", "XEDParse", and "asmjit" (which is selected). The "OK" button is highlighted. Below the dialog, the assembly code continues with "return 0;" and "end".

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.

The screenshot shows assembly code with annotations. A red arrow points to the instruction "add esi,edi", with the annotation "esi = 5 + 5". Another red arrow points to the instruction "mov edx,esi", with the annotation "edx = esi (10)". The output string "0x0000000000404000: '5 + 5 = %d\n'" is shown.

We can see here the full modified code.

The screenshot shows the full modified assembly code. The code includes the "multiply\_func" function, the "end\_of\_function" label, and the "main" function. The "main" function calls "multiply\_func(5,5)" and prints the result. The output string "0x0000000000404000: '5 + 5 = %d\n'" is shown.

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

```
5 x 5 = 25
5 + 5 = 10
```