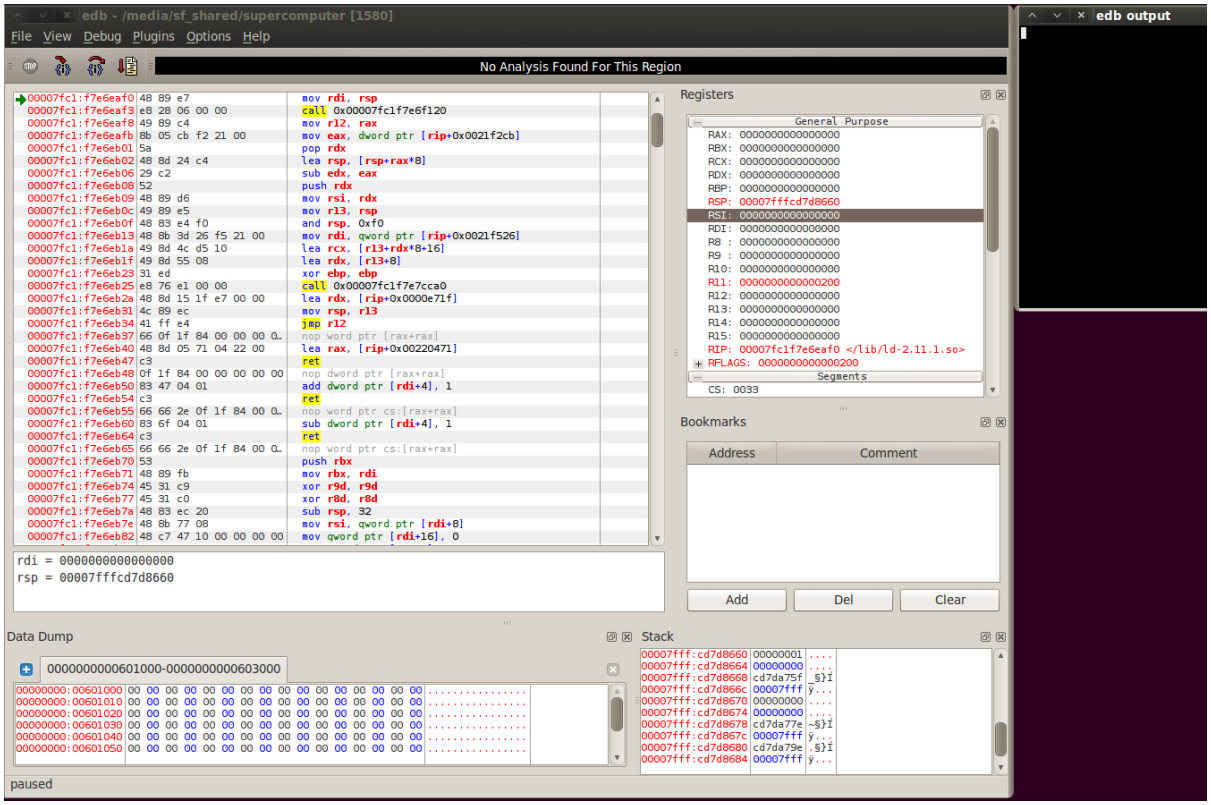
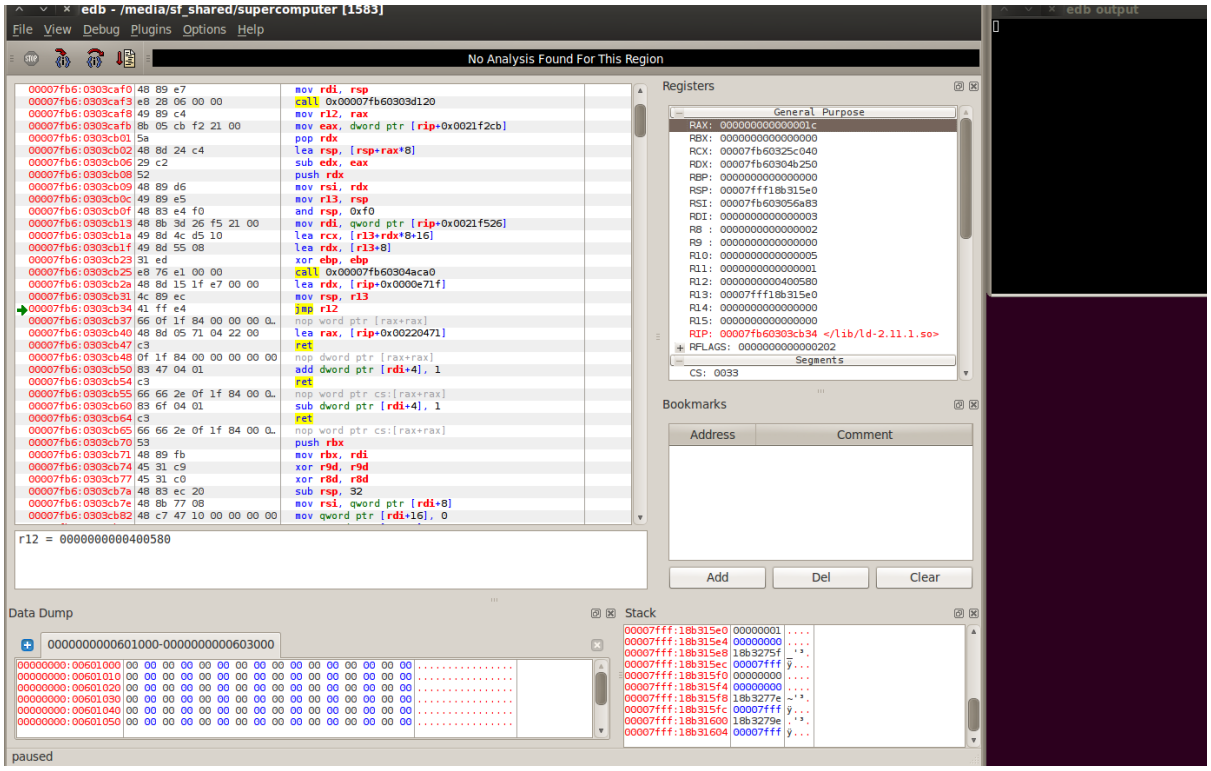
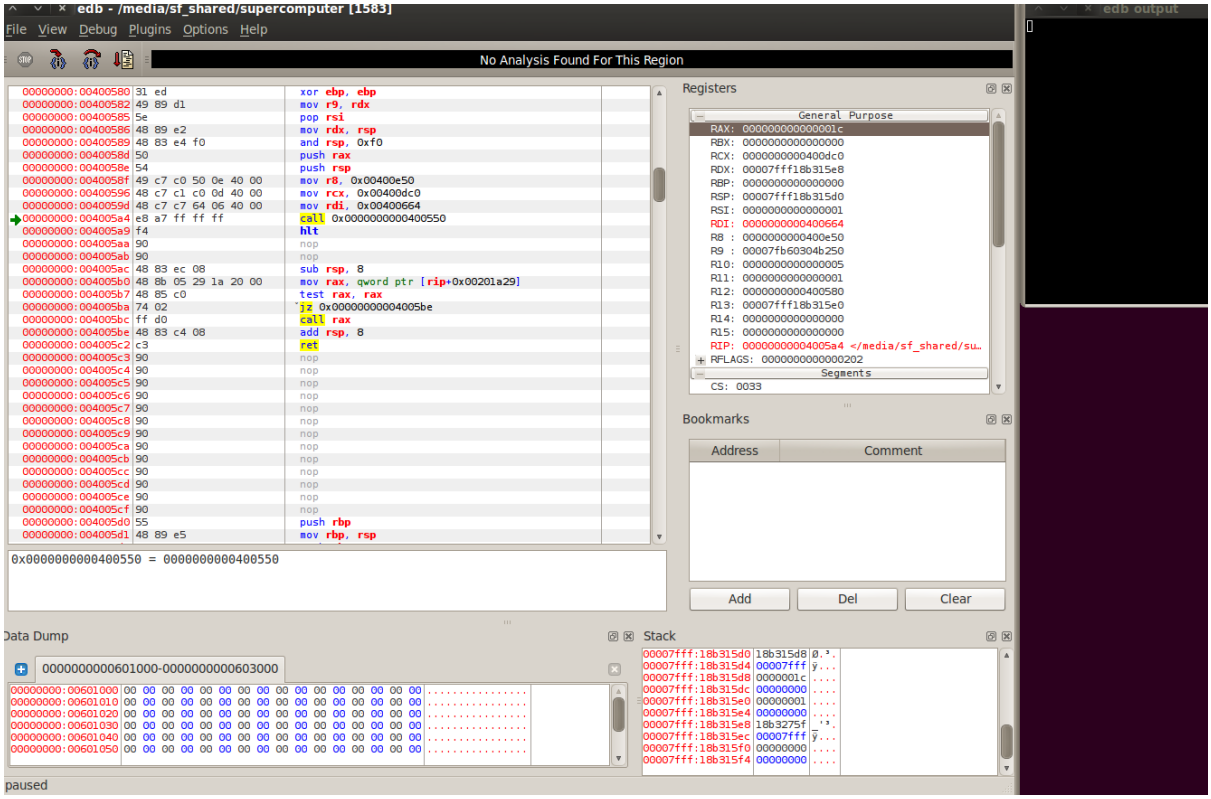


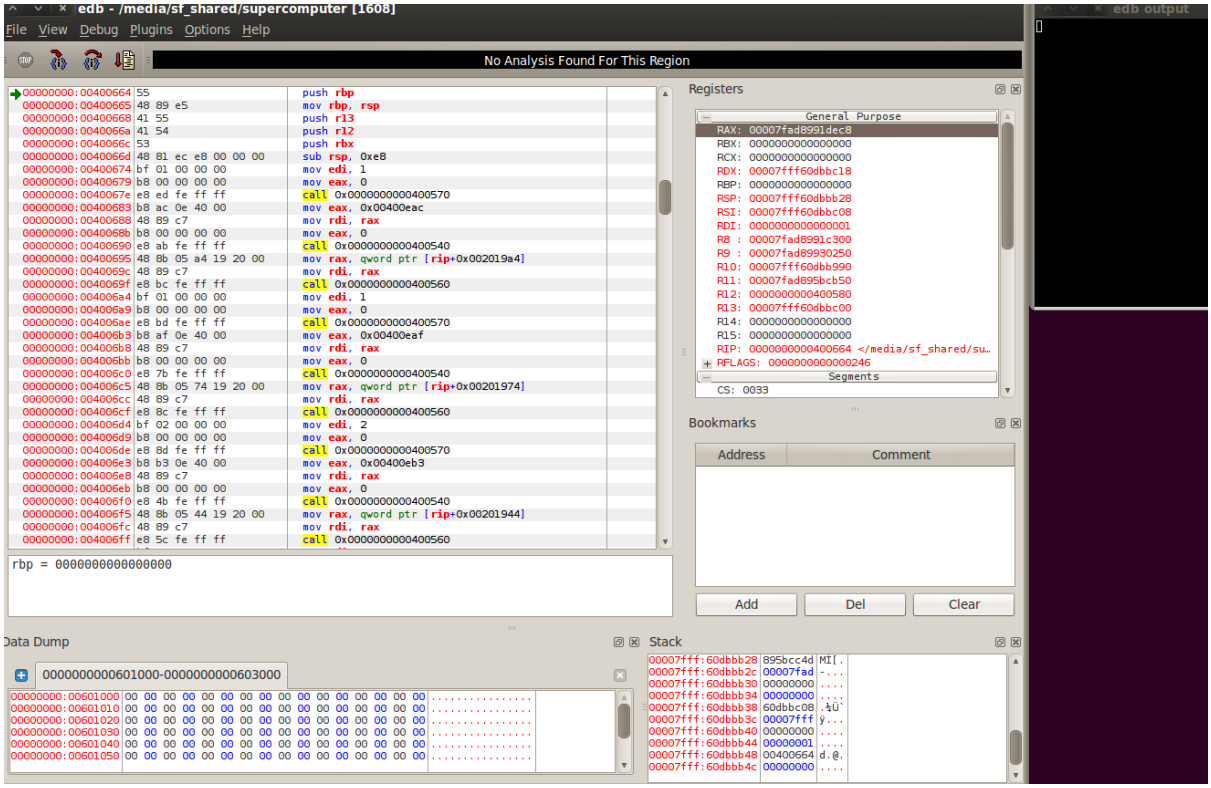
So this is the starting point of the super computer for challenge 0x01. Using Evan's debugger I started by stepping through the code. We can see we're currently in ld library at the moment, so I wanted to find some user code by stepping through with f8, until I got to here -



its jumping to r12, whose value is 0x400580, this address range is common for user code. Stepping through that code we end up getting to here –



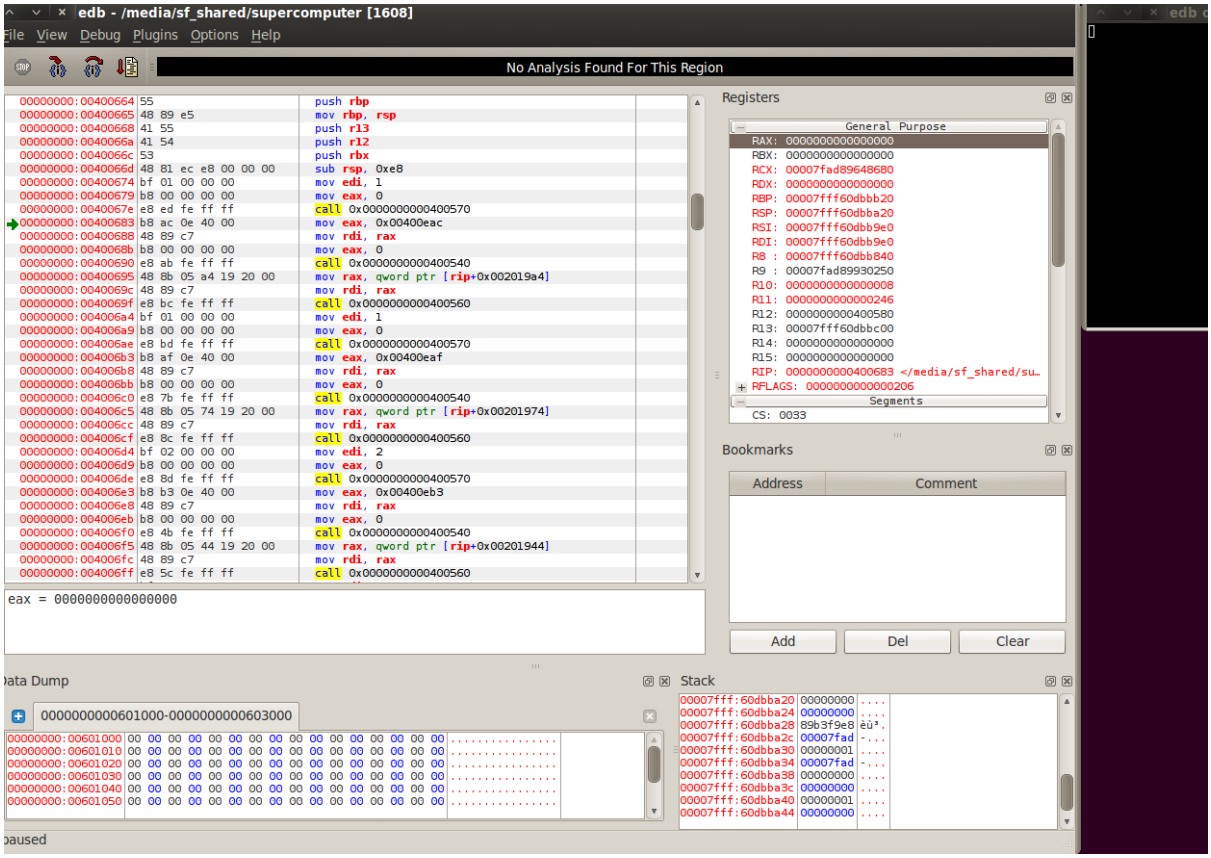
another function inside the user code section (we can also take a look at the memory layout /PE header to check this).



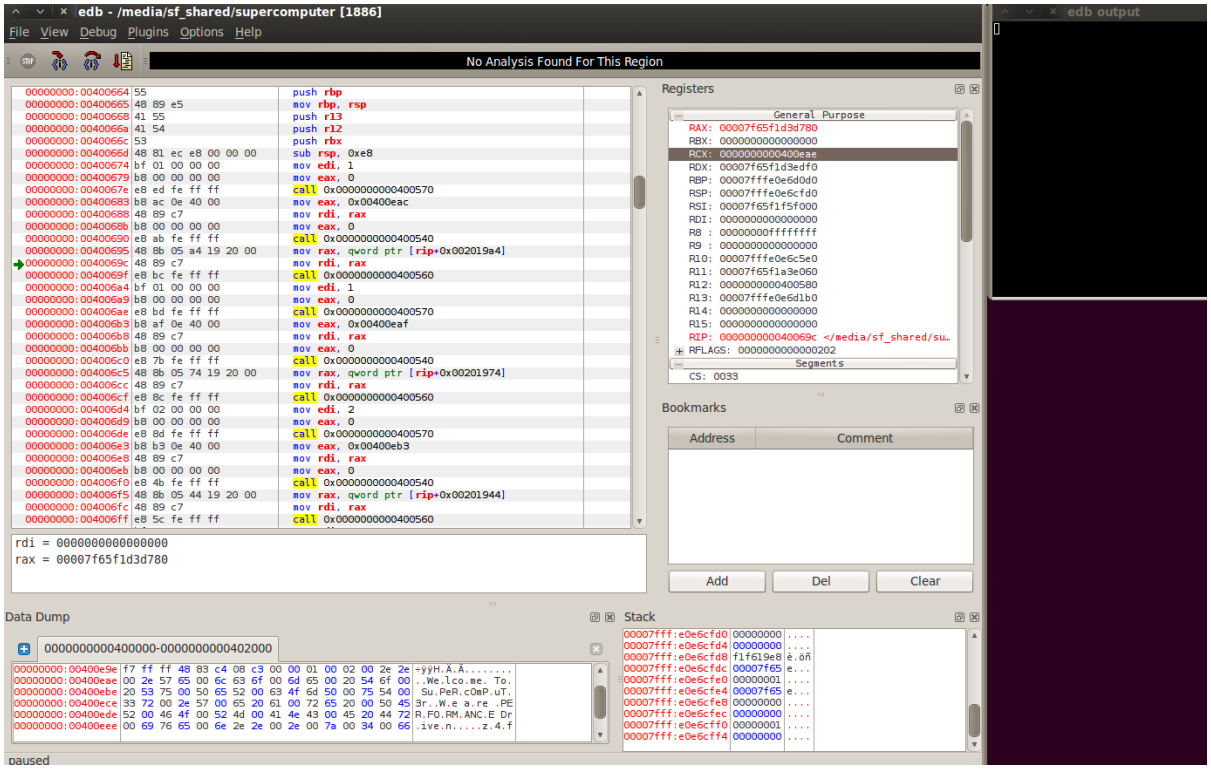
And this is the start of our super computer. You can see it runs through calling the same 3 functions, also note that before the function calls we have mov's which setup the parameters. The function calls are –

0x400540
0x400560
0x400570

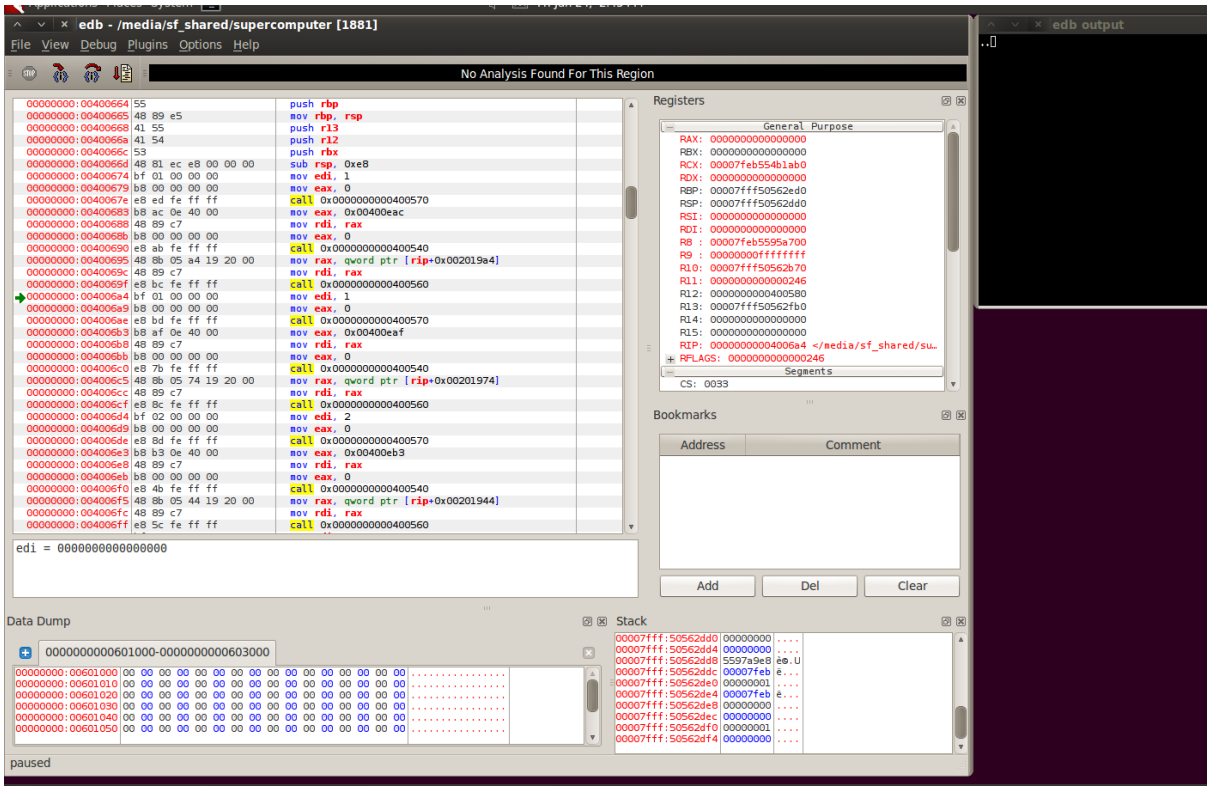
All in user code section. Stepping over the first call we notice that it pauses for a second –



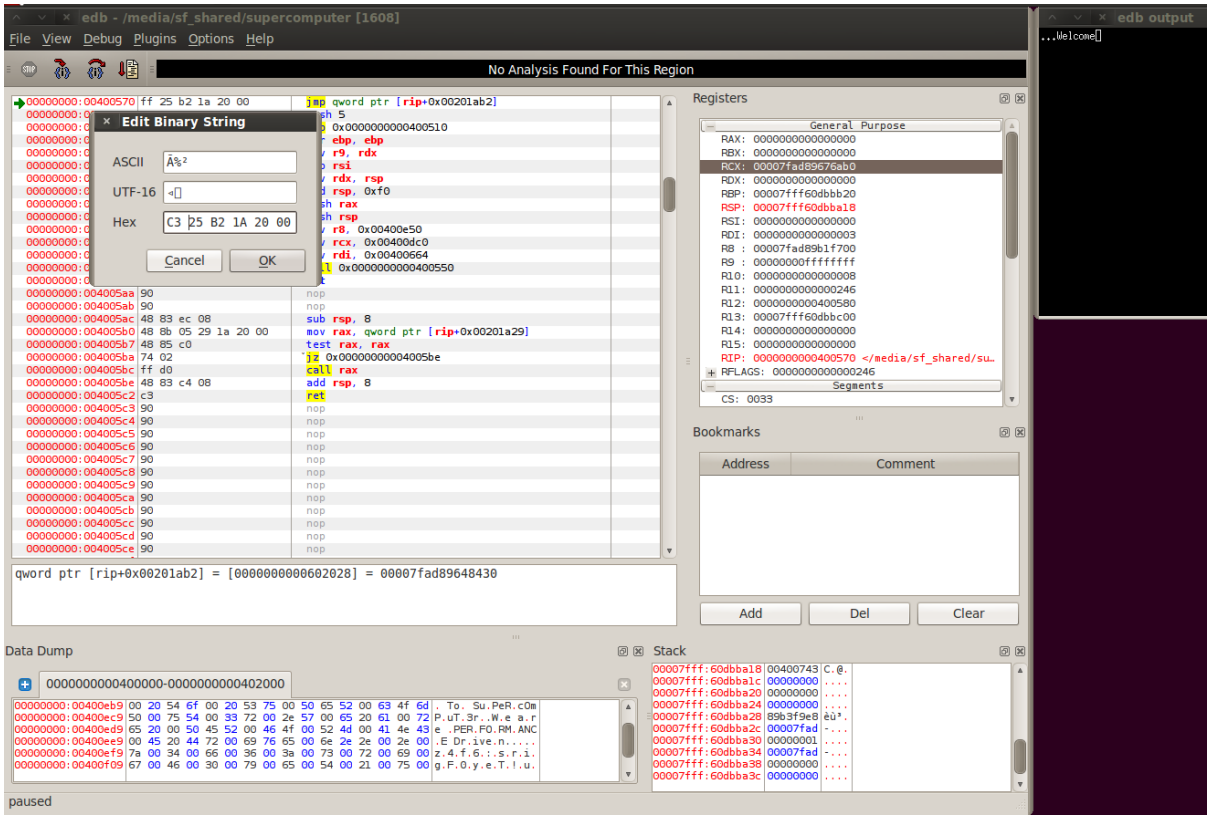
Stepping over the next function and following one of our registers as a pointer we get this result –



Notice the text in the dump and RCX pointing to the start of this. Next function call we see the first 2 characters are displayed, can you see why it's only two? Have a close look at the previous image.

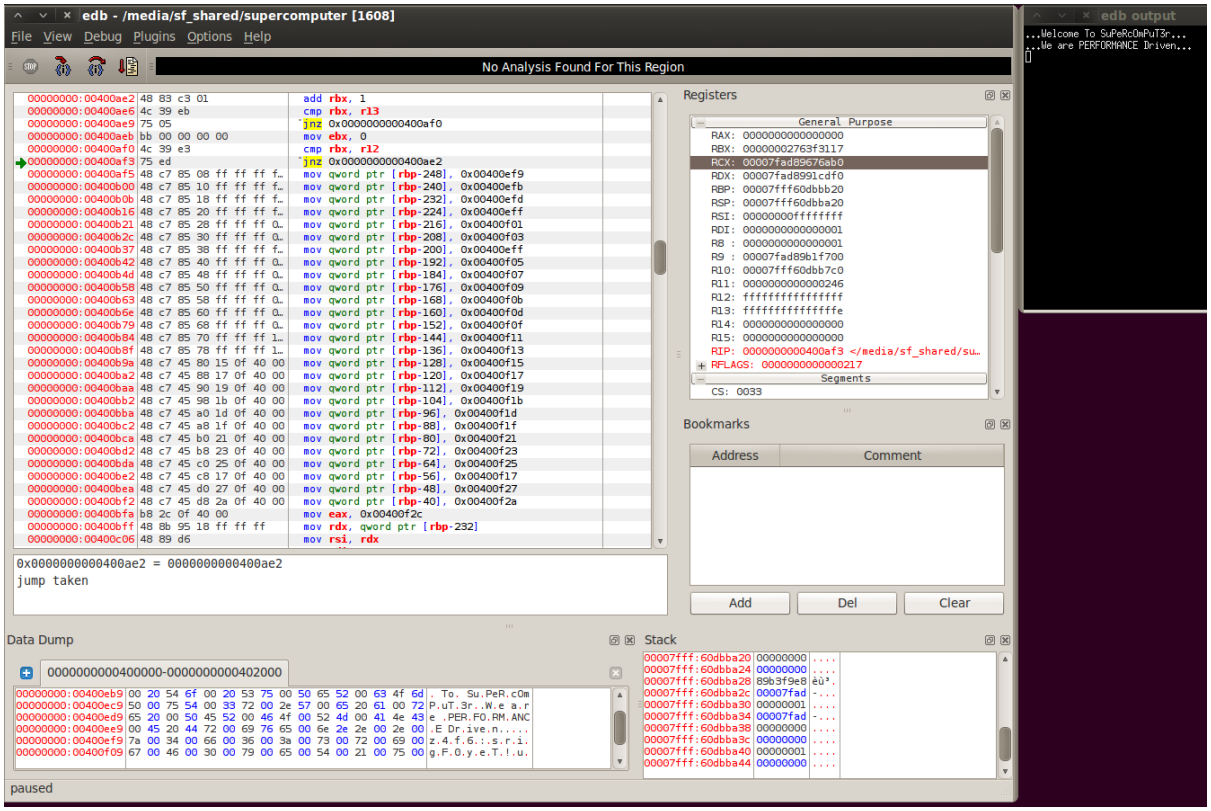


So I want to speed this up first of all. If I go into the 400570 function we can see it goes to a jump table, this is used as an abstraction for common library functions so it is trivial to change their address.

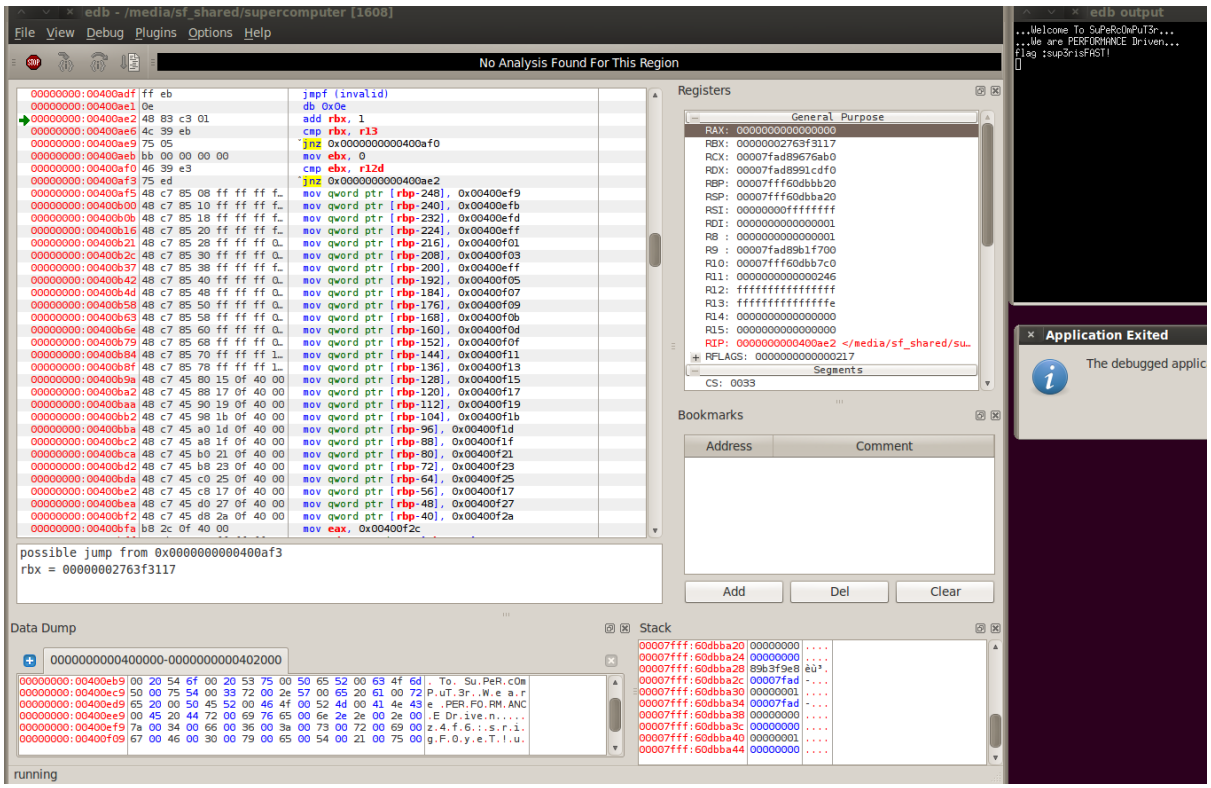


What I'm doing here is changing one byte of the jump (FF) to be a ret (C3). This will return immediately to the calling location (which you'll notice is on top of the stack). This does make the rest of the bytes in that line meaningless, but it should never get to that code (we could nop them if we wanted). Also this method only affects the jump table and doesn't touch library code, so we could still use the sleep function by referencing it directly and skipping the jump table.

This fixes our performance problem. Now we keep going and we notice that we get stuck in a loop which is the top 6 lines here -



It's a never ending loop, take a look at the logic and see why it will never end. Now we could change the JNZ (75) to a JZ (74), but that method was shown in the last talk. Let's see if we can find some other 1-byte solutions. Another one is changing the cmp before the last JNZ to compare something which is equal –



I've changed 4C to a 46 so it compares only the end (low) 32bits of r12 with ebx. This means ebx will reach that after a couple of seconds and the jump will work (as we can see by the flag). But I wanted

something faster. If we change that same byte to 35 it changes the statement to an xor, which takes in the next line with it using it as an operand. This removes our jump altogether and instantly reaches our desired outcome, here is the program with the changed byte (it will now finish instantly)-