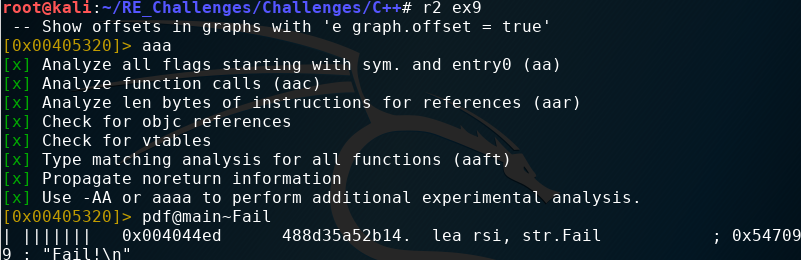
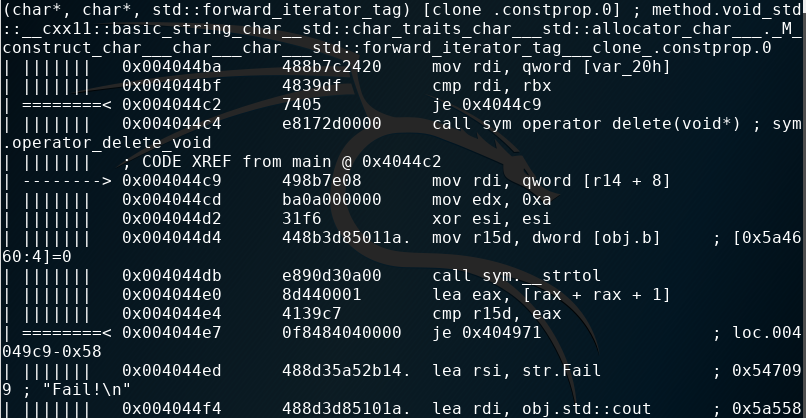
***Ex9***

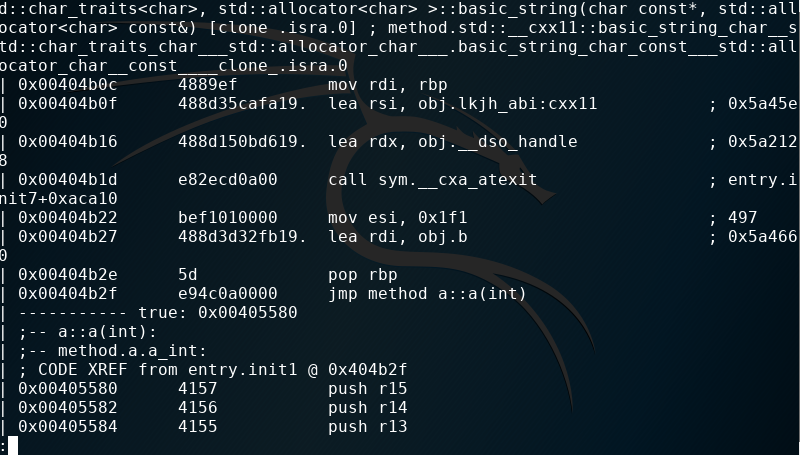
If we approach this challenge like the others and just use a [pdf@main](mailto:pdf@main) we get overwhelmed by how much code is involved in this challenge. Since we know that every challenge has the strings “Fail!” & “Score!” in it and it has been beneficial to start looking around these strings we can filter the output of [pdf@main](mailto:pdf@main) with a more complex variant such as [pdf@main](mailto:pdf@main)~Fail. This will return all lines that include the text Fail which can be used to reference where we should start looking for how out input is being checked.

From here we can use [pdf@main](mailto:pdf@main)|less to scroll through the disassembly of main until we get to the address where the fail string is referenced.

Now things make some sense, we see the base address of some object name b, obj.b, being moved into the register r15d and a call to the string to long , strtol, function followed by a comparison that dictates the printing of our fail message. Knowing that our input to ex9 is translated to a char\* we can with high certainty assume that the strtol call is for our input. The one instruction that is not as straight forward that we have to concern our self with is the lea instruction at 0x004044e0. In the average case linux uses the std calling convention with used the eax/rax register for passing a return value back to a caller from the callee. The typical use of lea is to calculate effective addresses and mov that value into a register since we can not perform arithmetic on the source operand in a mov instruction. However, lea can be used to perform some efficient calculations, which is our case here. After the strtol call rax holds the value returned then, lea is used to calculate rax\*2+1. In other words rax+rax+1!

So we know that our input is being doubled and with an addition of one but in order to figure out what we need to give as input we have to figure out the obj.b previously mentioned. From prior challenges involving c++ objects we know the base pointer will point to the first data member of a class. That means we need to start figuring out the class implementation of obj.b. To start lets take a look at our flag spaces, we see the classes flag space which holds a LOT of entries. How might we be able to efficiently browse through these you may ask. It is quite simple, scrolling around the output we notice a trend with the characters gnu,std, and cxx. This is because the compiler has generated a lot of extra information dealing with the c++ runtime. Imagine if we could filter the output of our classes flag space for everything with out references to the c++ runtime. Well, we can do this easily from inside radare2! The ~, tilda, in radare2 is synonymous with the linux grep command. Grep will filter its input and only output what matches the pattern given to it and even the inverse of a pattern, meaning everything that does not hold to the pattern. All we had to do from the beginning was fs classes;f~!gnu,std,cxx. This command translates as change to flag space classes then print the contents of the current flag space grepping for all things not matching the patterned text gnu, std, and cxx.

Well that was easy, we see there is one user defined class named “a” which only has a constructor that accepts an integer. Let’s try to pdf the method.

This might be a new message to some, not to worry it actually saves you an untold amount of headache and wasted time. There are two ways to disassemble instructions, linearly and recursively. For this block of code we have take the recursive approach since it is not implemented in the typical function prologue, code, and return style we have always seen. The only change that we have to do is use the pdr command!With a quick scroll we see why the pdr command was needed, we actually didn’t print out the constructor a but the function that jumps to a block of code representing this constructor.

We can quickly identify that the integer passed to the constructor is 497. Knowing that our input is being doubled with 1 added to it why not just try this number? When dividing 497 by 2 we end up with a fractional 248.5 which would not translate to a long, recall strtol means, string to long, since longs are not floats or double which can represent decimals numbers. Maybe this is why it requires the addition of 1!

