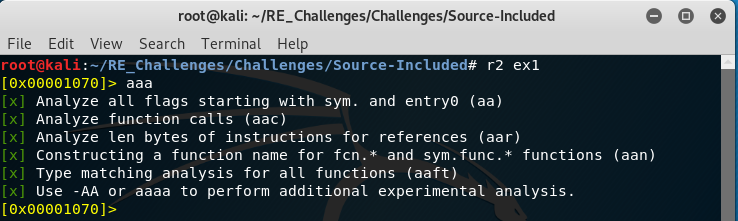
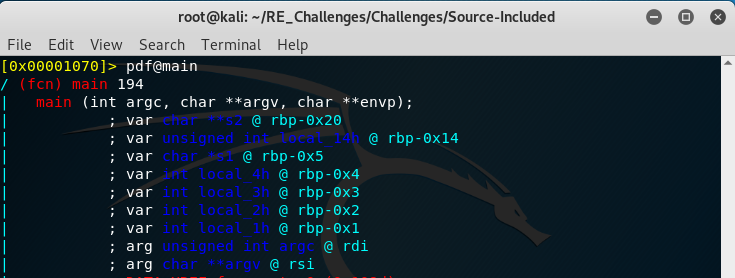
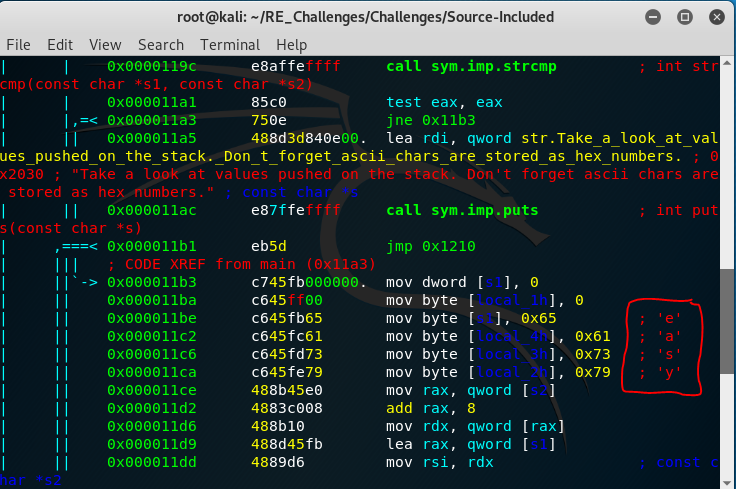
***Ex1***

In this challenge we wrote the code so each character for the comparison string is pushed onto the stack in order then the address of the first character is passed to the “strcmp” function, recall C strings are character arrays represented as a character pointer aka char\*, along with the char\* that represents the first character of our user input string.

To begin lets load the program ex1 into radare2 then perform some analysis on it. To have radare2 perform analysis we use the “a” commands, to view the differing type of analysis available use “a?”, more specifically the “aaa” command will be used and solely for simplicity sake (the commands possible in radare2 can become a little unwieldy)

From here we want to look at the disassembly for the main function in our C program, we accomplish this by telling radare2 to print the disassembly of the function at main “pdf@main”. The command will print a lot to the screen so you can either scroll up with your mouse wheel to the top of the commands output or use the pipe “|” and “less” command combinations to make things easier to navigate “pdf@main|less”. Note that radare2 has given us a listing of the local stack frame variables and their positions relative to the base pointer of the current stack frame along with the argument count and argument vector, “rsi” and “rdi”. Keep in mind the names given to the stack frame variables “local\_Xh”, they will help us understand what code is interacting with the stack in the future.

Notice the password yet? Yes, this was pretty “easy” but a lot is going on so we will talk about some of the details to keep it short and simple. Note the command “mov byte [local\_1h], 0” we will come back to that in a second, after that command we can see that we are moving hexadecimal values into the local\_Xh variables and we see characters written in red as comments to the very far right. This is because radare2 knows that ascii text is just sequences of hexadecimal numbers and gives us comments based on what it thinks these values could be. Why is the code pushing the values onto the stack like it is? Remember a string in C is just a character array, it is pushing the values one by one in order into the stack so it just has to pass a reference to the address of the first character and any other code that receives this address can figure out the remaining elements of the string. How does it know when the string terminates though? Well it doesn’t just “know”, strings have an extra character known as the null terminator, which is why you will hear the term “null terminated strings”. This is where our “mov byte [local\_1h], 0” instruction comes into play. Recall the “y” character was moved to the local stack frame variable “local\_2h” and that strings are character arrays so each character in memory is placed one after the other and “y” being the last character of our string we need a null terminator “0” placed right after this so anything that uses this string will know where it terminates at.

