x86 Assessment 2: Disassemble Bomb

At the start of your analysis of the bomb project put a summary of the stages and what input you provided to successfully disarm the bomb for that stage. If you were unable to disarm the bomb, clearly indicate all stages in which you could not disarm the bomb. Here is an example of what a summary should look like. The answers given are made up and will not disarm the bomb.

1. Stage 1: Disarmed bomb for this stage by entering: `Fiddle_Sticks`
2. Stage 2: Disarmed bomb for this stage by entering: `Garbage_Collection`
3. Stage 3: Disarmed bomb for this stage by entering: `0xbad`
4. Stage 4: Disarmed bomb for this stage by entering: `31 19 7 13 29 41 61`
5. Stage 5: Disarmed bomb for this stage by entering: `T 113 J`
6. Stage 6: Disarmed bomb for this stage by entering: `431 M 907`
7. Stage 7: Disarmed bomb for this stage by entering: `The_Cat_Is_On_The_Mat`
8. Stage 8: I was unable to disarm the bomb for this stage.
9. Stage ?: I was unable to disarm the bomb for the secret stage.

Be sure to put this summary at the *beginning* of your analysis of project bomb. Even if you only disarm the bomb for 2 or 3 stages, it is imperative that you include this summary before you provide your analysis.

Your analysis should describe how you solved those stages where the bomb was disarmed. Include a careful and clear description of your solution, the disassembled code, your use of a debugger, including the debugger commands, *etc*. It is important to describe your thought process and the techniques you used to solve a stage.

For those stages where you were unable to disarm the bomb, explain what you think the disassembled code is doing. If you use the debugger on one of these stages, explain what debugger commands you used. Note that it is possible to change the contents of registers, *etc*., and bypass a stage in the debugger if you can't get that stage to disarm the bomb.

If you cannot disarm a stage, it is also possible to change the contents of an instruction to a *different* instruction, like a jump instruction, in the debugger to bypass a stage. Note that changing the *instruction pointer* register `rip` can be done to force execution to another instruction. This might enable you to bypass a stage.

Whatever, you do, please carefully explain it. You are being graded on your analysis and explanation, in addition to actually disarming the bomb for every stage.

The examples discussed in class should help you with the bomb project. You may use them or disregard them completely, and try your own methods and approach. The discussions in class were meant as an aide. If you create files `x` and `y` that contain the disassembled code and the read only section, respectively, be sure to include them as part of your solution. You can keep them as separate files and refer to them by citing specific lines, addresses, instructions, *etc*., in your writeup. It is possible that correct input you provide for some stages will not be correct input for someone else.

The writeup should be in a git repository called `bomb`. This is a challenging project. Good luck!