Periodical exam 2

There are 2 questions in this exam worth 7.5 marks each. All questions are mandatory. You are allowed to refer to classroom material but use of internet is not allowed and considered cheating. All other regular examination related rules are applicable here as well. The decision taken by the instructor in any matter related to the exam will be final and binding. The solution will be evaluated on Ubuntu 14.04. The total duration of exam is 2 hours.

Question 1

Analyze the binary “reverseme.out” and reimplement the same functionality in higher level programming language. You are allowed to use any programming language in your solution except assembly programming language. Please include a shell script run.sh which will execute your solution by passing the arguments appropriately. If you are using a compiled language, include a shell script compile.sh that will compile the source files.

Sample session

This sample session only demonstrates the input and expected output format. No inputs or outputs are provided.

```
$ ./run.sh <argument list>
<Output of running reverseme.out with same argument list>
```

How we will test your program

We will invoke compile.sh and run.sh file you upload using sh. We test your solution on Ubuntu 14.04, which symlinks sh to dash. If you use a different *nix OS, please ensure that your run.sh file is compatible with dash. For most simple commands, dash is mostly a drop-in replacement for sh but if you run some fancy commands or use shell specific tricks in run.sh, expect issues. We will pass the same number of arguments as required by the binary to run.sh and your solution should print the same output as binary. All string arguments will contain only lowercase or uppercase alphabets.

Question 2

You are given a program “vulnerable.out”(both binary and source code) that reads 1 line from input from stdin and prints it to stdout. Craft an exploit that can subvert the program execution and execute the function print_flag. Carefully analyse the source code before crafting the exploit. Be sure to debug it locally and ensure it works. Create a file run.sh that will compile and/or execute your exploit generator. Store the generated exploit in a file exploit.txt. We will provide this exploit.txt file as input to the vulnerable binary.

Sample session

We will first execute the run.sh file and then give the generated exploit.txt file as input to the vulnerable executable. You can assume that the flag.txt file will exist when the exploit you created is given as input to the vulnerable executable.

```
$ ls
run.sh
vulnerable.out
<other files>

$ sh ./run.sh
$ ls
run.sh
vulnerable.out
exploit.txt

$ ./vulnerable.out < exploit.txt
<contents of file flag.txt in stdout along with other lines, if any>
```