End semester exam

There are 5 questions in this exam. 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 3 hours.

Question 1

Write a 64 bit assembly program that prints from 100 to 1, skipping a number i.e. first 100 is printed, then 98, 96 and so on. Use of standard C library functions is permitted. You cannot invoke any additional code besides the assembly code you write and the standard C library functions.

Sample session

```
$ ./print-numbers.out
100
98
96
….
```

How we will test your program

We will compile the assembly code you submit using nasm and gcc. The exact commands that will be executed are:

```
$ nasm -f elf64 print-numbers.asm
$ gcc print-numbers.o -o print-numbers.out
```

We will not pass any arguments to the program.

Score

10 marks

Question 2

Analyze the given binary(2-reverseme.out) and identify what are the inputs it is expecting. Save all expected, correct inputs to a file “input.txt” and upload it along with a file named “writeup.txt”. In the file “writeup.txt”, describe how you found the correct inputs. Make sure you clearly describe how you reverse engineered the binary and found all the correct inputs in “writeup.txt”.

You can verify your input file by checking if the output of the binary when it is given your input file as input matches below output:

```
$ ./2-reverseme.out < input.txt
Woah! You found them all! You are l33t
```

How we will test your program

We will evaluate the input.txt file you upload to see if it has the correct inputs. We will also evaluate your writeup for clarity of explanation, completeness and quality.

Score

15 marks

Question 3

Analyze the given binary(3-reverseme.out) and reimplement the same functionality in another programming language. You are allowed to use any programming language in your solution except assembly programming language. Please include a “run.sh” file which will compile and/or execute your solution by passing the arguments appropriately.

Sample session

This sample session only demonstrates the input and expected output format. The exact output for specific inputs are not displayed.

```
$ ./run.sh ABCD EFGH
Output of binary when run with ABCD and EFGH as arguments
```

How we will test your program

We will invoke the 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 compatible with other shells but if you run some fancy commands or use shell specific tricks in run.sh, expect issues. We will required number of arguments and your solution should print output in the same format as shown in the sample session.

Score

5 marks

Question 4

Carefully study the source code given(4-vulnerable.c) and identify the vulnerability in it. After identifying the vulnerability, programmatically generate an exploit that will overwrite the local variable “variable” with the appropriate value such that the program will display the contents of file “flag.txt”. Save the generated exploit to a file “exploit.txt”. You can use the executable file provided(4-vulnerable.out) for testing your exploit. You can generate the exploit using any programming language. Please include a “run.sh” file which will compile and/or execute your solution to generate the “exploit.txt” file.

Sample session

```
$ ./run.sh                              # Should create the file exploit.txt
$ ./4-vulnerable.out < exploit.txt
<Contents of the file flag.txt>
```

How we will test your program

We will invoke the 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 compatible with other shells but if you run some fancy commands or use shell specific tricks in run.sh, expect issues. You can assume that the file “flag.txt” will exist when your exploit is executed.

Score

5 marks

Question 5

Carefully study the source code given(5-vulnerable.c) and identify the vulnerability in it. After identifying the vulnerability, programmatically generate an exploit that will display the contents of flag.txt by invoking the function “print_flag”. Save the generated exploit to a file “exploit.txt”. You can use the executable file provided(5-vulnerable.out) for testing your exploit. You can generate the exploit using any programming language. Please include a run.sh file which will compile and/or execute your solution to generate the “exploit.txt” file.

Sample session

```
$ ./run.sh                              # Should create the file exploit.txt
$ ./5-vulnerable.out < exploit.txt
<Contents of the file flag.txt>
```

How we will test your program

We will invoke the 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 compatible with other shells but if you run some fancy commands or use shell specific tricks in run.sh, expect issues. You can assume that the file “flag.txt” will exist when your exploit is executed.

Score

10 marks