Problem 0. Simple errors [18 points]

For each of the following code samples:

* Provide a cogent explanation in 140 characters or less of what author probably intended for the code to do.
* Identify an approach that reliably makes the compiled code [(1)](https://algorithmics.bu.edu/fw/EC521/HomeworkFour#FootNote1note) produce the string ec521.Hint: The string needs only appear *somewhere* during the output of the program." style="vertical-align: super; font-size: 10.5px;"> [(2)](https://algorithmics.bu.edu/fw/EC521/HomeworkFour#FootNote2note)
* Identify the Fortify [kingdom](https://vulncat.hpefod.com/en) of the vulnerability you exploited. For further clarity, you may add the phylum of the error as well, if you wish.
* You may not make any modifications to the source code. Specifically, you may also not:
  + Tamper with the compiled code in any external way.
  + Manipulate the running memory of the code with external programs.
  + Alter any files outside of a temporary directory in which you place the code.

0a. vuln2.C [vulnA.C]

1. **Explanation:** Author likely meant the code to read in a # < 7 digits and print out a list of right justified numbers starting at 1, each on a newline.

2. The code suffers from a buffer overflow, input vulnerability due to a lack of Input Validation and Representation since they do not check for the right type and length. I was able to determine this by analyzing the code and seeing that the arrays that take the user's inputs are only designed 6 space in memory on the stack. This shows that if the users puts in more digits or characters than space initiated for the array, other information on the stack maybe printed when looping through the array. In this case I was able to determine that if the user inputs 16 digits, after the 16th digit anything entered after will print that number of times (if compiled and run on a mac OS then anything after the 6th digit will be printed and repeated), or if the user wants a specific output only once they can enter a single "1" and then 31 additional charaters. After the 31st charater (32 in total) the characters or inputs enter afterward will print once. See examples below:

*./vulnA*  
How should I go? **1aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaec521**  
**ec521**  
# 32 chars in total; 1 and the 31 a's to print once

# Or 16 single numbers and then ec521 to repeat multiple times  
  
#mac - 1234567ec521 - multiple prints

3. **Kingdom: Input Validation and Representation**

Input validation and representation problems ares caused by metacharacters, alternate encodings and numeric representations. Security problems result from trusting input. The issues include: "Buffer Overflows," "Cross-Site Scripting" attacks, "SQL Injection," and many others.

0b. vuln2.C [vulnB.C]

**\*Two of the same programs; reached out to the Professor and he said just to do the analysis and question for the 2 unique files I was given. Please see above answers.\***

0c. vuln4.C [vulnC.C]

1. **Explanation:**Author meant to set the seed for rand() based on epoch time of the system, if the rand() number matches then to print ec521.

2. In order to get the string "ec521" to print, the system time must be set to a specific time that correlates to the epoch time that will set the seed for rand, using srand(time(NULL)), for rand() to produce the correct number that matches. In order to do this I was able to write a C script to find the number that rand() needed to seeded with in srand(). I was also able to see this number in gdb by setting a breakpoint and seeing the value of the register that srand(time(NULL)) was set to. I found that this number was 1478410166. Using an online epoch time converter, I was able to determine that this epoch time correlated to 6 NOV 2016 01:29:26 EST. Since I have root to my machine I was able to set my system time to this time and date by running *"date -s “6 NOV 2016 01:29:26”; ./vul4.out" .*This set my system date to the correct time to seed the rand() function and print "ec521".

3. **Kindom: Time and State Vulnerability**

The code suffers from this vulnerability since if the user has the ability to change the system date or change how the program reads the date, then they can manipulate the date and run the program.

Problem 0 grade: (**GRADER ONLY**)

Comments:

Problem 1. Server-side code **[40 points]**

The (terribly-written) code attached to this topic ( [process.C](https://algorithmics.bu.edu/fw/pub/EC521/HomeworkFour/process.C), and [getpost.h](https://algorithmics.bu.edu/fw/pub/EC521/HomeworkFour/getpost.h)) is also running on 10.241.13.199. Unfortunately, this code has quite a number of vulnerabilities and errors, which you will attempt to identify in this homework.

1a. Warmup [10 points]

First, you need to understand the code. Describe the task of ***each*** function in process.C with a *concise* and *complete* English-language explanation (up to 140 characters).

\*Functions:\*

1. char \*\*topValues(cStringMap &patterns)

topValues() takes in a cStringMap object, finds the highest value of patterns, ensures that it is not chosen again and returns the highest.

2. void parse(cStringMap& patterns, int len, char \*text)

parse() takes in a cStringMap object, the substring length, and value of text, breaks the text into substrings, marking each as seen.

3. int main()

main() get the inputs from the user, calls parse() and topValues() to get the patterns and find the top 10 and displays this to the user

1b. Coding [20 points]

Problems:

* How many Ni 's are there in the plain text of the book ["Pride and Prejudice"](http://www.gutenberg.org/cache/epub/1342/pg1342.txt) by Jane Austen?
  + There are 2 occurences "Ni" in Pride and Preducjice.
  + I was able to get the server to provide this in several ways. First I was able to pull out just the lines that contain "Ni" and enter this into the server text field and parse for a subtring of 2 characters. I was also to grep the text to verify this and check in sublime to make sure the this was correct. I was also able to exploit the overflow and to take in the text in a slightly larger length than 5000 characters and output the finding.
  + 10 most common patterns of length 2:
  + Ni: 2
* Provide an input that produces no patterns at all.
  + Entering a number for length that is over the length of the largest substring entered in the text, or larger than the text entered such as entering a large number and no text input. This will produce no patterns.
  + Also entering a length of 40 or more outputs no matter if there is a substring that actually contains a pattern substring of 40 characters.
  + 10 most common patterns of length 5001:
  + NO patterns found
* Provide an input that produces patterns that are not found in the text.
  + Entering unknown, or non-English characters and images show patterns that are not given in the text. It displays the uni-code pattern of the given length, if length is within range. For example entering a shaded box with pattern of 4 displays its uni-code 9638.
  + ▦
  + 10 most common patterns of length 5:

#9638: 1  
σ: 1  
9638;: 1  
9638;: 1  
9638;: 1  
9638;: 1  
9638;: 1  
9638;: 1  
9638;: 1  
9638;: 1

* Provide an input that produces patterns with a length different from what you specified.
  + I was able to determine that if a text longer than 5005 caharacters give a length number of a very large number no matter what the length was specified by the user. This number is a the addition of the characters converted to binary and added together.
  + len: 2
  + Text: aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbjjghbbbbbbbbbbbbbbbbvgjhgbmnqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfaaaaf
  + 1751413608 most common patterns of length 1987273830:
* Provide an input that produces the 102 most common patterns.
  + I was able to determine this by entering anything over 5005 characters. After this the next character is converted to ACSII value and displayed as length over writing the "10" that is displayed and set in the code.
  + 102 most common patterns of length 1633771873:

len: 0

Text: qweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfqweqerewfsdgvhgfbjdhfnjdgbjgdfbhjgvhgdgfdsvgdvhgdjfbfghvdsvfdsvhsdhbgfvhghgfdhgfsghfdsgfdsvfdsgfdjhfaaaaf

* Provide an input that displays a pattern 9632 even though it does not appear in the text.
  + I noticed by entering other types of characters such as Spanish, Italian, Greek, and Chinese characters that the output gave certian numbers. After investigation I realize these numbers corresponded to the UNI-Code numbers of those charaters. Since I my number to display is 9632. I found this correpsonded to a the uni-code number of a black square image. After entering the black square into the text box and the length of 4 , since I needed the 4 digit number to display I was able to get the desired output.
  + ■
  + 10 most common patterns of length 4:

#963: 1  
`: 1  
632;: 1  
9632: 1  
9632: 1  
9632: 1  
9632: 1  
9632: 1  
9632:

1c. Defense [10 points]

*Minimally* (i.e. least number of character edits) fix up the code so that it has the desired functionality (i.e. shows 10 most common substrings of a given length from a given text) without vulnerabilities. Provide your source code and make sure to explain what you have changed from the original code.

* In regards the vulnerable functions found there are serveral minor tweaks that you can make to the code to assure vaild input. Please refer to the CHANGE # comments on the changed lines.
* First using strncpy and specifying the length. These can be changed in the topValues function and the main function.
* Second at the top of process.C you can initialize TOP\_NUM, followed by length, followed by text, since text may be the largest variables of the 3 slightly preventing overflow of the text onto the length and TOP\_NUM on the stack.
* Another verification is to make sure that all for loops and copies of data are correct.
* There are also other way to error catch that can be altered by, such as throw catach or user input validations, that can be written to ensure that the user only enters 5000 characters, and that the cahracters are acceptable and not symbols. In addition checking that the length is only a digit and not longer than the text so the atoi() function is not exploited and does not overwrite other data on the stack if a long digit or input is entered. Also verifiying that only a digit is entered and not a decimal, characters or other symbols are entered. Alternative error checking functions should be used such as strtol(), strtoul(), or isdigit(0 for verification.

Problem 1 grade: (**GRADER ONLY**)

Comments:

Problem 2 - Reverse Engineering **[42 points]**

Executable 3:

2a. Initial exploration [16 points]

i. What kind of machine code does executable contain?

The type of executable is **ELF64 for x86-64 bit executable [elf64.I64].**

I was able to see this by running readelf slots3, rabin 2 -I slots3 (to list the flags and see the file type) or find it in gdb.

*rabin2 -I slots3*  
pic false  
canary false  
nx true  
crypto false  
va true  
**bintype elf**  
**class ELF64**  
lang cxx  
arch x86  
bits 64  
**machine AMD x86-64 architecture**  
os linux  
subsys linux  
**endian little**  
stripped false  
static false  
linenum true  
lsyms true  
relocs true  
rpath NONE  
binsz 7849

ii. In a fair machine [(3)](https://algorithmics.bu.edu/fw/EC521/HomeworkFour#FootNote3note), what is the [expected](https://en.wikipedia.org/wiki/Expected_value) number of machine uses before seeing [seven] [seven] [seven] ?

In a fair machine the expected value of a slot machine with 2 options for each indiependent slot, like ours being [bar] and [seven], is **0.125 or 12.5% or 1/8** since there are 3 independent slots and 2 options for each slot. Thus there is a 1/2, or 50% chance of seeing a [seven] in each. To get the expected value these probabilites must be multiplied. Thus **0.5\*0.5\*0.5 = 0.125 (12.5%).**

iii. What is the empirical probability of seeing [seven] [seven] [seven] on your machine over one-minute run? [(4)](https://algorithmics.bu.edu/fw/EC521/HomeworkFour#FootNote4note)

The empirical probability is defined as the number of times a desired event occurs divided by the total number of times the experiment is performed. I was able to run the slots continiously back to back in a loop, testing both in python and bash scripts (attaached). I was able to count the number of times the program ran and write the output of each run to a file. Once completed I serached the file using *"grep -o '\[seven\] \[seven\] \[seven\]' slot\_out\_bash.txt | wc -l"*to count the number of occurrences of 777. Once I had these 2 numbers I could determine the empirical probability. After several test runs to compare my findings I found that the empirical probability was a**pproximately 9.5%.**

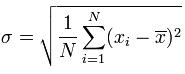
**2763/29558 = 0.093477231**  
**2852/29325 = 0.097254902**

**3088/31456 = 0.098168871**

**.....**

iv. Given your observations above, what is the probability that your machine is fair?

Given m observation I was able to determine that the probability that my machine is fair by comparing the probability of a fair machine to the probability of my sample run for a minute of my machine. To determine this I was able to calculate the standard error which is definied by first finding the number of standard deviation away our sample run was from the mean of a fair machine. Assuming that the distribution should be a normal distribution and the mean is set as the mean (expected value) of a fair machine. My machine has a less likelyhood of winning since the empirical probabilty of the minute sample run concluded that the odds is about 2% less than a fair machine. In order to find the standard error, I was able to calculate the standard deviation of the fair machine and our sample. The std dev of our sample = std dev fair / sqroot(# of run during sample). This gave 0.387/sqroot(31456) = .00218. Then the std dev of our sample is (.125 - .1) / .00218 = ~11.5 stand deviations away from the mean. Concluding the standard error is over 3 std devs away and making our machine unfair. Less likely to win as compared to a fair machine.



http://davidmlane.com/hyperstat/pictures/sem_popN.gif

2b. Disassemble main [26 points]

Dissassemble the main function and answer the following about it.

1. What functions are called in main and what do they do (provide a brief, 140 characters or fewer explanation)?
   1. getRandom(int): \_Z9getRandomi
      1. getRandom opens/runs the linux dev/urandom file, gets a random number, mods the number by 4 and returns that number.
   2. time(): <time@plt>
      1. time(NULL) get the epoch time of the system and stores and provides that number in the stack of the program.
   3. cout: <\_ZStlsISt11char\_traitsIcEERSt13basic\_ostreamIcT\_ES5\_PKc@plt>
      1. cout writes the given parameter to stream buffer, standard out, and displays this to the user.
2. One of the functions (call it foo) computes a uniform random number.
   1. What functions are called within foo ? [(5)](https://algorithmics.bu.edu/fw/EC521/HomeworkFour#FootNote5note)
      1. <open@plt>
      2. <read@plt>
      3. <close@plt>
      4. foo is the get Random function that opens and run the dev/urandom file, gets the random number and mods that number by the buffer size which is 4 and retunrs it to main.
   2. What file does this function access to produce a random number? [(6)](https://algorithmics.bu.edu/fw/EC521/HomeworkFour" \l "FootNote6note)
      1. /dev/urandom
      2. Linux executable to produce a random number.
   3. What is the range of random numbers provided by the function?
      1. mod of the buffer size; or mod 4 = 0-3
      2. I was able to determine this by putting a breakpoint in getRandom function and stepping through the assembly instructions and viewing the values in gdb. I was able to see that the random number given was moded by the buffer size which on our machine is 4. This was determined by also viewing the values of the return and this resulted in a range of 0-3, meaning the number was a result of the random number mod 4.
3. What does the cheater have to do to hit the jackpot consistently? Provide code that produces the desired wins using the attached executable.
   1. In order to hit 777 consistently the user can set their system time and date to an epoch time that results in 0 when moded by 11. So epoch time mod 11 must equal 0.
   2. I was able to determine this by view the register value of the place where the time() function was called. This was my system epoch time. then stepping through in gdb, i was able to see the values for this time checked and compared to a very large number, sifted and then multiplied by 11. If this output was 0 then the result of 777 was displayed. This meant that the epoch time must result in 0 wahen moded by 11. I was able to verify this by running the following continously and getting 777. This make correlates to the winning empirical probability that was found for my machine after running it for a minute. So the probability of a win, if the machine is run at least once a second, there is a win once out of 11runs which gives 0.091 or about 9.1%. This matches the probability of the sample run.
   3. *date +%s -s @134460458; ./slots3*
   4. Another way is by removing the randomization by removing or modifying the dev/urandom file.
4. If you were running this machine, what would you do to make sure it is providing a fair result.
   1. In order to make sure that the it is providing a a fair result I would conclude making a few changes and validation.
   2. If keeping the vulnerabilities I would make sure the date is based on a probability of 1/8 or 12.5%. Since my machine has a win of 10% and is set to auto win if the epoch time mod 11 is 0 then, changing this to the epoch time of the system mod 8 = 0 then we would get a fair machine that produces a win ratio of 1/8 or 12.5%
   3. Although the vulnerabilities I would take out if possible would be the following:
      1. I would make sure that the user does not have root or acces to change the system time in anyway.
      2. I would make sure no part of the code or any functions is based on the system time.
      3. I would make sure that randomization is based on other parameters other than the time and the dev/urandom file since the suser can alter, cahnge or remove this file and win.
      4. I would not relie on comparing the probability on the mod of certian limited numbers.
      5. I would make sure the seed to any ranomization is not finite and not able to be altered.