Which fuzzers did you select for running a BFF campaign? Why did you choose these fuzzers? For full credit, make sure to base your answer of "why" on the behavior of each fuzzer. "I selected every fuzzer because I wanted to test them all" is not the type of answer we are expecting.

Fuzzer	Effectiveness Reasoning
CRLFMUT	Exploit the reliance on ASCII codes of Carriage-Return [13] and LineFeed [10] that I assumed would be readily present throughout the file. Errors were probably caused more by mutating LineFeed [10] more than Carriage Return [13]. Not having set LineFeed characters would append 2 (or more) lines together which could exceed the size set in the string variables and array bounds
BYTEMUT	Create invalid characters outside the range of acceptable ASCII characters
INSESRT	Depending on where inserted, new characters could change the number of fields and/or lines past the array bounds. Could also create characters outside the range of acceptable ASCII characters.
DROP	Removal of a byte at each location in file would make different/invalid characters more prevalent with the actual content/data. Alterations to markers/delimiters/linefeeds would result in varying line lengths/# of distinct fields per line, etc.

Excluding the verify fuzzer, what are some fuzzers that are ineffective at finding a crash? Why do you think these fuzzers were ineffective at finding crashing test cases?

Fuzzer	Ineffective Reasoning
NULLMUT	Replacing NULL fields with actual values might create an inaccurate output/data file but it crash application as long as within the acceptable character sets. Unclear how frequent NULL locations appear in the file for the fuzzer to even mutate to begin with.
CRMUT	When a CR[13] is replaced with a random value, the application still correctly reads the input from different lines.
TRUNCATE	Since there is no specific "end of file" character, deleting from the end of the file, while might produce incorrect output for the last line/record of data, would not crash program since the "end of file" marker would be adjusted.
SWAP	This approach would not impact the markers for a new line, new field (by removing) or create characters different from the original seed files that worked with the application.

BUG #1

Using the data generated by the BFF campaigns and other debug utilities you ran as a guide, examine the source code in csvParser.cpp and answer the following questions for the first bug that you found in the csvParser application:

Campaign	csvParser 5.2.01
Seed File	sf_d704174fc94c8ca37a7f1aad4f1b2c4e-7rPaif-minimized
Line Crashed Per .GDB	#82 [`fields[count] = *i++;`]
Unique Log Entry	sf_d704174fc94c8ca37a7f1aad4f1b2c4e.csv crash_id=406419b7b77a45a92eb39be8caf77eca bitwise_hd=7 bytewise_hd=2

1. **Which line of code in csvParser.cpp contains the bug?**
   #77: *`string* fields = new string[25];`*
   `newFields[k] = fields[k];`

2. **Explain why the failing input discovered by the BFF caused the application to crash.**
   By mutating the linefeed character it stored what was originally 2 lines as 1 line and doubled the number of fields in the array "NewFields" exceeding the hardcoded upperbound value of 25.

3. **How could you fix the bug so that the program will execute properly?**
   Increase the size of "Fields" array to more than 25 or so that it is equal to the "count" variable or make it's upperbound undefined so that it can handle increases in the # of fields in a line

BUG #2

Using the data generated by the BFF campaigns and other debug utilities you ran as a guide, examine the source code in csvParser.cpp and answer the following questions for the first bug that you found in the csvParser application:

Campaign	csvParser_v5.2.21
Seed File	sf_708da24b63623b1a3f25f3100c93a0fe-0
Line Crashed Per .GDB	#61: std::cout << "Replacing invalid character" << *myCsv[i] << "with blank" << std::endl;
Unique Log Entry	sf_708da24b63623b1a3f25f3100c93a0fe.csv crash_id=df675f37df4186880d488add7e0cd542

1. **Which line of code in csvParser.cpp contains the bug?**
   Line #54: `std::regex ascii("[^\\x00-\\x7F]+");`

2. **Explain why the failing input discovered by the BFF caused the application to crash.**

   The application is not successfully defining the invalid non-ASCII characters and does not exclude the from processing. The test file had an invalid character inserted at the beginning of the file that caused the crash in this particular test case

3. **How could you fix the bug so that the program will execute properly?**

   Ensure that the code that parses for invalid characters is triggered when it is the in the line[] arrary