**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.**

I ran all 9 Fuzzers for approximately 10 to 15 minutes each. I used the following order as a priority based on which Fuzzer Option would yield the most issues and uncover the most bugs.

|  |  |  |  |
| --- | --- | --- | --- |
| Order of Fuzzer for csvParser | Fuzzer | Errors  Found? | Justification for Prioritization |
| 1 | CRLFMUT | Yes | 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] 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 |
|  | BYTEMUT | Yes | Create invalid characters outside the range of acceptable ASCII characters |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**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 would not cause the application to crash so long as the new values were within the acceptable character sets |
| **CRMUT** | When a CR[13] is replaced with a random value, the application still correctly read the input from different lines. |
| **TRUNCATE** | Since there is no specific “end of file” character, deleting bytes/characters from the end of the file, while might produce incorrect output for the last line/record of data, would not cause the program to crash since the “end of file” marker would be adjusted to the new version of that test case/seed file’s last row. |

**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:**

1. **Which line of code in csvParser.cpp contains the bug?**
2. **Explain why the failing input discovered by the BFF caused the application to crash.**
3. **How could you fix the bug so that the program will execute properly?**

**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:**

1. **Which line of code in csvParser.cpp contains the bug?**
2. **Explain why the failing input discovered by the BFF caused the application to crash.**
3. **How could you fix the bug so that the program will execute properly?**

* **bytemut**: replace bytes with random values
* **swap**: swap adjacent bytes
* **wave**: cycle through every possible single-byte value, sequentially
* **drop**: removes one byte from the file for each position in the file
* **insert**: inserts a random byte for each position in the file
* **truncate**: truncates bytes from the end of the file
* **crmut**: replace carriage return bytes with random values
* **crlfmut**: replace carriage return and linefeed bytes with random values
* **nullmut**: replace null bytes with random values
* **verify**: do not mutate file. Used for verifying crashing testcases