**ESC BUG HUNTING TEST REPORT**

**Tester**: [Joe] Baarath S/O Sellathurai (1005433)  
**Method of Testing**: Fuzzing  
**Implementation 1’s Author**: [Charles] Lim Thian Yew (1003158)  
**Implementation 2’s Author**: [Daniel] Koh Aik Hong (1005139)  
 **Background on Fuzzer**  
The default fuzzing test has 5 main sections.  
The 1st section generates 1 **valid** CSV (w/ **random characters**), 1 **valid mutated** CSV (based on the 1st valid csv), 1 **invalid** csv (copy of the 1st valid csv w/ an **additional comma** inserted at a random line), another 1 **invalid** csv (copy of the 1st valid csv w/ a randomly selected **line to be reduced** into half).  
  
The 2nd section passes the generated csv files to the implementation program using python’s subprocesses and return the error code and a flag to check if an output csv was generated. For this fuzzing I focus soely on scenario 1 and where only the last column is used for mismatch value comparasion.  
  
The 3rd section compares the 1st valid csv with the 2nd valid mutated csv and repeats again in vice versa order. By default, this section will check the valid error code and output csv for number of mismatches, number of duplicates, and number of missing records. However, realizing that the 3rd party implmentation may not account for errorcode, duplicates and missing records do not display the mismatch types, I have added a flag to ingore the error code and another flag to instead use the length of the output file as the value for the number of mismatches.  
  
The 4th section compares the 1st valid csv with the 1st invalid csv (w/ an additional comma) and repeats again in vice versa order. This section checks if the 3rd party implementation can identitfy the additional comma and throw a valid error.  
  
The 5th section compares the 1st valid csv with the 2nd invalid csv (w/ 1 reduced line) and repeats again in vice versa order. This section checks if the 3rd party implementation can identitfy that 1 of the line is reduced and throw a valid error.  
  
After completing all 5 sections, the fuzzer generates a log file which details the success and failed test cases and the failed csv files content for inspection.

**Testing Implementation 1 {[Charles] Lim Thian Yew (1003158)}  
  
Note**: All Fuzzing test code, log details and bug evidence   
are stored in “Testing\_Bug\_Hunting\Testing\_Charles\_Code”

|  |  |  |
| --- | --- | --- |
|  | **Bugs** | **Comments** |
| **1** | Unhandled valid double quoted csv | While adapting the fuzzing test discovered that the 3rd input for the program which were requesting for the unique header did not handle double quoted header values and treated them as distinct and separate values. |
| **2** | Unhandled valid csv input files with special characters | During Fuzzing tests, all csv inputs with punctuations/special character were not successfully during valid comparasions. |
| **3** | Invalid error code | Only for the scenario with the invalid csv w/ additional comma, the proper error code was not returned even though error message was displayed. |
| **4** | Unhandled duplicate record | Duplicate records are not accounted for. |
| **5** | Unhandled missing record | Missing records are not accounted for. |

**Testing Implementation 2 {[Daniel] Koh Aik Hong (1005139)}  
  
Note**: All Fuzzing test code, log details and bug evidence   
are stored in “Testing\_Bug\_Hunting\Testing\_Daniel\_Code”

|  |  |  |
| --- | --- | --- |
|  | **Bugs** | **Comments** |
| **1** | Unhandled valid params with space | For the 3rd input of the program, after typing the valid column header indexes seperated with comma w/ spaces, the program throws an exception. |
| **2** | Unhandled invalid additional comma | When additional commas were added, the program still resume with comparasion without throwing an error. |
| **3** | Unhandled missing record | Missing records are not accounted for. |
| **4** | Unhandled duplicate record | Duplicate records are not accounted for. |