Nanako Chung
Application Security
Brendan Dolan-Gavitt
September 22nd, 2019

REPORT 1

Exactly how the program works:

The spell.c file contains four functions that work to check the spelling of content in a given file. One function takes a "dictionary" file (a file with many words that are spelled correctly) and stores it in a hashtable (i.e. load_dictionary). Another function (i.e. check_words) takes a file, reads in the content line by line, iterates over each word, calls another function to clean the word in case there are punctuation marks and such attached to it (i.e. trim), and assesses whether it is misspelled or not by calling the third function (i.e. check_word), which takes the word and tries to find it in the dictionary/hashtable.

The output of Valgrind & brief explanation about Travis:

```
HEAP SUMMARY:
    in use at exit: 19,725 bytes in 168 blocks
  total heap usage: 189 allocs, 21 frees, 28,173 bytes allocated

LEAK SUMMARY:
   definitely lost: 0 bytes in 0 blocks
   indirectly lost: 0 bytes in 0 blocks
     possibly lost: 48 bytes in 2 blocks
   still reachable: 200 bytes in 6 blocks
        suppressed: 19,477 bytes in 160 blocks
```

These mass amounts of bytes are coming from the fact that we cannot free the newNode until after the hashtable is finished being used. Additionally, these bytes could be coming from using the C string functions such as strcpy() or strlen(), which read a string to find its length until it hits a null character. However, in the case of not having a null character, the pointer may be moving beyond the end of the string, in which this becomes undefined behavior. I use C string functions in many places throughout my code. I could fix these bytes by creating my own strlen function or being extra careful with memory management (which I tried to do by setting all the characters in pointer word to null every time a new word is found in the text file in the check_words function).

Additionally, another testing system was used. Travis CI is a continuous integration testing tool that allows you to see if your entire program builds when it pushes to git. In the context of this project, my Travis builds correctly and passes all the tests under OSX.

What bugs you expect may exist in your code & why those bugs might occur:

- The dictionary may not have as much coverage as we trust it to because it does not list all the possible words (e.g. "shoemaking" is not in the dictionary)

- The dictionary may not have the correct spelling of words because there may be mistakes or alternative spellings (e.g. "colour" vs. "color")
- Words with punctuation marks and special characters may lead to faultiness, especially if there is more than one of these kinds of characters or a mix of both """ and punctuation marks and special characters
- Numeric strings may have commas and decimals in them, which is valid!
- Some strings with punctuation in the middle are technically "spelled right" such as websites.

What steps you took to mitigate or fix those bugs:

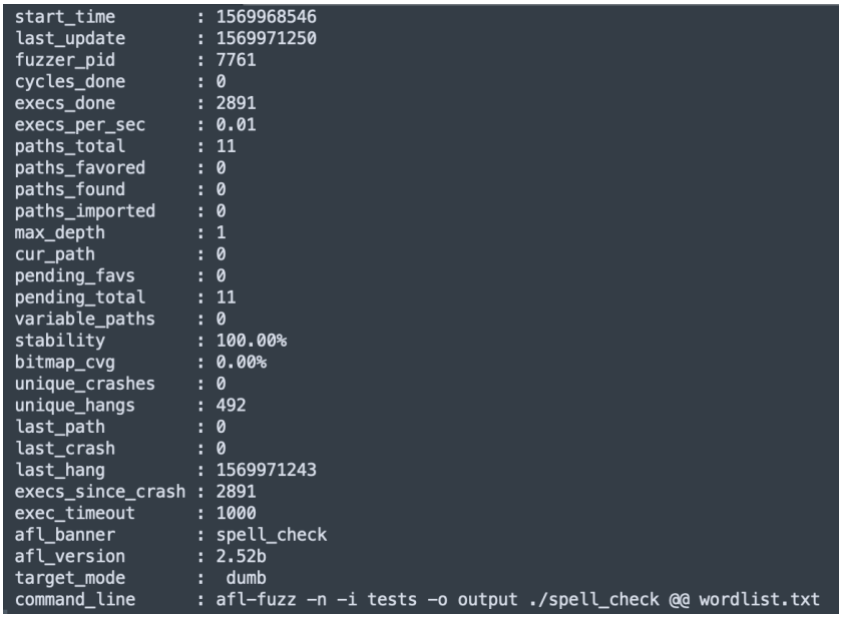
- For the dictionary files, it was tricky and pointless to find a solution because there can never be 100% coverage of the words in the English (and non-English) language...
- In my code, I looked at strings with any wacky punctuation or special character, trimmed them using my trim() function to cut off any non-alphanumeric characters at the beginning and end, and then let it go through the detector to see if it was actually misspelled. Thus, all of the unexpected characters in the ASCII table are handled.
- Numeric strings are handled as well. We trim() the string, count up the digits, commas, and periods, then do a handy math conditional to check if the string is a valid number.

All of the bugs found by your tests:

- Txt files with … WITHOUT SPACE failed to pass (e.g. I did this…and that)
- / and – were not being analyzed correctly (e.g. here/there and self-conscious)

All of the bugs found by fuzzing:

- None! I ran my program for 45 minutes and there were no crashes. It is slow, but even after running for this long, there weren't any crashes.



How the bugs were fixed & how similar bugs can be avoided in the future:
I fixed the / and – bug by just adding a check for c == '/' || c == '-' because it essentially checks it the same way as if the word buffer is full or if there is a space or line break. I did think it was important to fix the … bug because it is arbitrary (no space between the … is a mistake on the user's part), but I think it could be fixed easily by adding c == '…' as well!