Andrew Gallagher (arg213)
Computer Security
PA2 Report
4/1/2021

1. Introduction to Code:

Hello! Thank you for reading this document for my code. This project decrypts a mono-alphabetic ciphertext and returns the best key for the cipher it can gather. In this document, I will be explaining my code and my thought processes.

Make sure the input file is next to the python file (The python file is called “main” and uses python 3) in the folder. Also make sure that the “TrainingFiles” folder is also in the same area. In the terminal for the folder, run python3 main.py

Please make sure to follow the prompt in the console to input the encrypted text file!

2. Character Frequency Heatmap:

The first thing I thought about doing was making a heatmap of character frequency for the English language. The way I did this was by making a basic machine learning algorithm where it takes in 8 text files, all containing different texts and stories, gets the character frequencies for each story, then takes the average of all frequencies per character and makes a trained heatmap. This will be used as a baseline / last resort for calculating characters we cannot find using common trigger words.

Functions Used: trainDecrypt(), heatmap()

3. “The” and “And”

In the English language, the words “the” and “and” are the first and second most common 3 letter word, respectively. Utilizing this knowledge, I can try to find these words first and then expand searching for new letters from there. I made the function numWordFilter which inputs the message and the size of the word you are looking for, and then it outputs the first word, the second word, and the array of words from most frequent to least frequent.

Functions used: numWordFilter()

4. New Character Word List:

This is where I used the heart of my code, which is the method wordFilter. wordFilter takes in 5 parameters: The message, a heatmap of the ciphertext, the trained heatmap, the ciphertexted word “the”, and the ciphertexted word “and”. The Idea of this method is to use the word list that houses many common words to pick up unknown characters. Because we generally start with “the” and “and” and we know those are our first words, we how have 6 starting characters to make words from.

So now, for example, in my word list I have the word “each”. Since we already know ‘e’, ‘a’, and ‘h’, we if we find “each” in the document, we will be able to find ‘c’. This idea is repeated for all the words in that list.

My program will get the index of the missing letter in the word. Then it will convert all the other letters in the word to the ciphertext. It then finds the cipher word heatmap list using numWordFilter for the length of the regular word. While searching that heatmap, if it comes across a word that has all the letters in the right order with the one letter missing, it takes that cipher letter in with correspondence to the plaintext letter.

As we find new letters and have confirmed them, we delete them from the original ciphertext heatmap and from the plaintext heatmap. We do this because at the end we will attach the letters we could not find to the end of the key (recall that the ciphertext heatmap is sorted already from greatest frequency to smallest frequency).

All the words in the word list that are being used are in a very particular order. Because there are words like “as”, “is”, and “us”, I had to be very careful about the order of words in the list, or even if the words should not be in the list. A good example of this is making sure that I already had the letter m before I search for “be”, since “me” is a word that could mess up finding the letter ‘b’.

Functions Used: wordFilter(), numWordFilter()

5. Decrypt:

This method is used for bringing all the code together and printing out the key and plaintext. It also helps with using insertion sort to help sort all the arrays that needed to be sorted by frequency or even by letters (which is how we get the key that is alphabetically ordered by the plaintextkey).

Functions Used: numWordFilter(). heatmap(), wordFilter(), decrypt() .

6. Conclusion

Thank you for looking at my code! I thought that pairing the heatmap with the word list helps get my code to a very high accuracy of getting the correct key, if not a couple of letters off. Some vulnerabilities with this code are if the top 2, 3 letter words are not “the” and or “and”, then the algorithm will start thinking the wrong cipher letters are the right cipher letters. Also, the more characters and words there are in the test file, the more accurate my code works. Also, there is the accidental vulnerability of the example I showed in the last paragraph of “4. New Character Word List”, where a cipher word has the same number and letters as well as the same index of the missing letter.