**Nova Southeastern University**

**College of Computing and Engineering**

**Project Report**

**Dice ware Passphrase Generator**

**Jose Luis Ron Coka**

**CSIS 4010 Computer Security**

**Fall 2022**

**Introduction:**

Today people keep most of their sensitive information online, where it must be kept safe from third parties with bad intentions. User authentication is becoming more and more important. It has evolved from simple passwords to two factor authentication and biometrics; passwords are still the most popular even though they can also be the least safe since there are open-source tools like John the Ripper that can brute force a password in a few seconds. Because of this when creating a password today you must think about making it hard to crack. Most platforms ask for a combination of items to include, like numbers, symbols, capital letters and more. Coming up with a safe password is very important, which is why a lot of passwords generators exist today, you can search one online and get an extremely hard to crack password like: 54vHMW1!, which I got from LastPass.com.

The problem with random generated password is that while they may be safe, but they are not practical because a random combination of characters is impossible to remember. The only practical way to use random generated passwords is by also using a password manager. But there are a few key disadvantages, not all managers keep your passwords across all your devices, the provider company is still vulnerable to attacks where your information can be leaked, if you lose your device you can be completely locked out of the manager, you still need a master password for the manager and if its lost or cracked it would jeopardize all your other passwords, and finally most of the good password managers are not free.

**Solution:**

An alternative is using a passphrase. According to Froehlich “A passphrase is a sentence like string of words used for authentication that is longer than a traditional password, easy to remember and difficult to crack.” (2022) A good passphrase is a combination of random words with no relation between each other. The longer the passphrase is and the more random the words are then the more secure it is. An example would be “plane sage technology bread”. This is as secure as a random password and is much easier for users to remember so a password manager is not needed.

However, when trying to think of a good passphrase we ran into problems. Humans are very bad at creating fully random things, we are a species of patterns, that is why a lot of people might use a quote from a song, book or movie as a passphrase. These are not as strong, because they lack randomness or entropy. Therefore, we require a method that gives us fully random passphrases.

My project, “Dice Ware Passphrase Generator” is a web-based app that generates a secure passphrase using the Dice Ware algorithm. Built with HTML, CSS and JavaScript. The web application software runs on the local computer inside a browser.

To generate a passphrase, the user inputs the Wordlist they wish to use, how many dice throws the Wordlist uses and the desired length of the passphrase to generate a passphrase made up of random words. The project also has the option to create an acronym passphrase, where the user inputs a meaningful word, and the passphrase is generated by words that start with each of the letters of the entered word. So, if the word is "bear" the passphrase could be "basic elephant around rock".

**Dice Ware Algorithm:**

As the name states, the Dice Ware algorithm uses a dice and a wordlist to create a strong and random passphrase. A Dice Ware wordlist is a list of words where in each line you get a code and a word. The main wordlist used for the project was created by the EFF (2018). This is an extract:

21145 corral

21146 correct

21151 corridor

21152 corrode

21153 corroding

21154 corrosive

21155 corsage

21156 corset

As long as you have a wordlist that looks like this you can generate passphrase with just a piece of paper, a pen and a regular dice. To generate the code, you throw the dice a number of times, for the EFF wordlist we have to throw it 5 times. After each throw you write down the number you got. You should have 5 numbers when you finish, that is your code. With your code you go through the wordlist to find the word that matches that code. Once you find it, you write down the first word of the passphrase. Depending on how long you want the passphrase to be you repeat the process the times needed. For example, just using the extract from before, if I throw the dice 5 times and get 21153, then the first word in my passphrase is “corroding”.

The strength of a Dice ware comes from the randomness of its words and its length. By choosing a word out of a Word list of 7776 words it could take an attacker 7776 chances to guess the word, now if you add another word that increases exponentially to 7776\*\*2 and so on. Therefore, just a five-word passphrase would take 14 quintillion tries to guess on average. (Lee, 2015)

We can simulate the same dice throw inside the computer. In the program, the dice throws are simulated with the built-in random function of JavaScript. The function *diceRoll()* returns a random number between 1 and 6.

**Pseudo Code:**

Once the user has entered the required inputs, the JavaScript read the information from the wordlist is read and passed into an array of arrays, where each member array has two elements, the first is the code and the second is the actual word.

Once the Generate button is clicked the *generateFunc()* is called. It takes care of generating the Dice Ware code and calling the match functions.

Using two nested for loops, one runs for every word in the passphrase, another runs the dice roll the times required to generate the code for each word. Using two arrays, one for passphrase another for each dice throw. After the second loop runs the array with the dice throws is turned to a string and saved on the passphrase array. This results on having an array with the codes.

To match those codes from words with the list we use one of the two match functions. *match(number)* searches for the matching word inside the wordlist Array and pass it to the final passphrase array. In case one of the words is 3 letters or shorter then match is called again recursively with the *getNewRoll()* function as its argument. This function generates a new code. Then if the new code matches a proper word it is passed into the final passphrase array and running continues.

The second match function is *matchInit(number,index)* which is used to create the acronym passphrase. It only gets called if the user entered an input for the acronym passphrase. It works in a simar way to the normal match, if the word’s first character is not one of the letters of the meaningful word then *matchInit* is called recursively with *getNewRoll()* as an argument. It generates a new code and checks whether this one matches the char, if not it calls the function again until a correct word is found.

**Manual:**

The process to run is straightforward, all you need is to extract the “DiceWarePassphraseGenerator.html” and the provided “wordlist.txt” and “wordlistEsp.txt” files from the FinalProjectCode.zip. The CSS and JavaScript code is encapsulated inside the HTML file. Once downloaded you can run it by double clicking the file, it may ask for what browser you want to use to run it, the project was tested on Google Chrome, Microsoft Edge and Opera GX.

Once it is running you just have to follow steps on the webpage.

1. “First Input the .txt file with the Wordlist you will use.” The input file must be a .txt file. It also must follow the Dice Ware Algorithm, specifically it must have the same format as the EFF wordlist. So, every line contains a code and a word. The code and the word are separated by a tab or \t space. Otherwise, the app will not work correctly.

2. “Enter how many dice throws your Wordlist requires.” This means how many dice throws are needed to generate the code for the Dice Ware. The Wordlists provided use five Dice throws. But the EFF also has a short Wordlist that require 4 throws. Customized wordlists could take more or less dice throws, that is why its important to input the value.

3. “Enter how many words you want the passphrase to contain.” The desired length of the passphrase depends on the user, the recommended length would be at least 3 words long and not more than 7 words long, to keep it secure and easy to remember.

4. “Leave empty or enter the meaningful word to form the Acronym Passphrase. IT MUST HAVE AS MANY LETTERS AS THE PASSSPHRASE LENGTH YOU ENTERED IN THE PREVIOUS STEP.” The word you enter must be as long as the desired length so each letter can be a word in the passphrase. An acronym passphrase can be much easier to remember since its made up of a meaningful word to you but it is still secure since they are still random words.

5. “Click on the “Generate” Button and see the results.”

“TO RUN AGAIN WITH SAME WORDLIST: Repeat Steps 2 through 5.” As long as the inputs are correct and the same wordlist is used the user can keep clicking on the “Generate” button to create a new passphrase until they find one they like.

“TO RUN AGAIN WITH DIFFERENT WORDLIST: Reload the page and repeat steps 1 through 5.

A small recommendation is displayed at the end of the program of good practices with the generated passphrase. If the user is good with the generated passphrase and wish to use it, we recommend that you save a copy of it, either on a Password Manager or write it in a piece of paper and store it in a safe place. Then take a little time to try to memorize it. To use it for some accounts you might need to meet requirements like adding numbers and symbols. While initially the app would give the function to add these requirements to the passphrase it was removed to keep the passphrase easy to remember. The user can make these additions on their own so they can choose the way to remember it.

**Evaluation:**

There were are many open-source projects that generate passphrases that were used as guides and inspirations. Some provide a lot of different functions which lead them to be more complex and harder to use and some even need to be installed into the user’s computer. Some of my favorites were “Diceware Password Generator” by dmuth and “passphraseme” by micahflee, both available on GitHub and cited on References.

My implementation of the passphrase generator was streamlined and made as simple to use as possible while still achieving the goal. It differentiates from other similar projects mainly in that it lets the user input the wordlist they want to use. If the wordlist, follow the parameters discussed the app should work accordingly. This opens the door for the user to create their own wordlists or look for some online. So, they can create a passphrase even more to their preferences. For example, a wordlist in a different language or a wordlist about a certain property like Star Wars.

To show the use of this a wordlist with words in Spanish was also provided, this was created by Manuel Palao (2020).

However, there are still limitations in the program because of the wordlists, a passphrase is only as good as its words after all. Users have more flexibility when choosing but they have more responsibility on making sure their choice is a good wordlist.

Since this is a project done on a limited amount of time and done by only one person there are many aspects it could improve, like efficiency. The front end could also be improved in future updates to be more concise an to look better. Since it is a web app hosting it on a web server could be a possible improvement.

The main objectives of generating a secure random passphrase and a secure acronym passphrase were achieved. Furthermore, since its web based it can be downloaded and used by anyone without any extra installs. Therefore, I can conclude that while there are many aspects the project can improve its still a good implementation.

**References:**

Froehlich, A., & Fitzgibbons, L. (2022). passphrase. Retrieved December 11, 2022, from Security website: https://www.techtarget.com/searchsecurity/definition/passphrase#:~:text=A%20passphrase%20is%20a%20sentencelike,to%20100%20characters%20or%20more.

‌ Lee, M. (2015, March 26). Passphrases That You Can Memorize — But That Even the NSA Can’t Guess. Retrieved December 11, 2022, from The Intercept website: https://theintercept.com/2015/03/26/passphrases-can-memorize-attackers-cant-guess/

EFF. (2016, July 18). EFF Large Wordlist for Passphrases. Retrieved December 11, 2022, from Electronic Frontier Foundation website: https://www.eff.org/document/passphrase-wordlists

‌

‌ Bonneau, J. (2016, July 19). EFF’s New Wordlists for Random Passphrases. Retrieved December 11, 2022, from Electronic Frontier Foundation website: https://www.eff.org/deeplinks/2016/07/new-wordlists-random-passphrases

‌

dmuth. (2022, December 8). GitHub - dmuth/diceware: Generate secure passwords you can actually remember! Retrieved December 11, 2022, from GitHub website: https://github.com/dmuth/diceware

micahflee. (2022). passphraseme/README.md at master · micahflee/passphraseme. Retrieved December 11, 2022, from GitHub website: https://github.com/micahflee/passphraseme/blob/master/README.md

Palao Manuel (2020). Retrieved December 11, 2022, from Std.com website: https://std.com/~reinhold/diceware\_espanol/DW-Espanol-1.txt

‌

‌