Strong Password Generation Based On User Inputs

Farhana Zaman Glory
Department of Computer Science
University of Manitoba
Winnipeg, Canada
gloryfz@myumanitoba.ca

Atif Ul Aftab
Department of Computer Science
University of Manitoba
Winnipeg, Canada
aftabau@cs.umanitoba.ca

Olivier Tremblay-Savard
Department of Computer Science
University of Manitoba
Winnipeg, Canada
tremblao@cs.umanitoba.ca

Noman Mohammed
Department of Computer Science
University of Manitoba
Winnipeg, Canada
noman@cs.umanitoba.ca

***Abstract*—Every person using different online services is concerned with the security and privacy for protecting individual information from the intruders. Many authentication systems are available for the protection of individuals' data, and the password authentication system is one of them. Due to the increment of information sharing, internet popularization, electronic commerce transactions, and data transferring, both password security and authenticity have become an essential and necessary subject. But it is also mandatory to ensure the strength of the password. For that reason, all cyber experts recommend intricate password patterns. But most of the time, the users forget their passwords because of those complicated patterns. In this paper, we are proposing a unique algorithm that will generate a strong password, unlike other existing random password generators. This password will be based on the information, i.e. (some words and numbers) provided by the users so that they do not feel challenged to remember the password. We have tested our system through various experiments using synthetic input data. We also have checked our generator with four popular online password checkers to verify the strength of the produced passwords. Based on our experiments, the reliability of our generated passwords is entirely satisfactory. We also have examined that our generated passwords can defend against two password cracking attacks named the "Dictionary attack" and the "Brute Force attack". We have implemented our system in Python programming language. In the near future, we have a plan to extend our work by developing an online free to use user interface. The passwords generated by our system are not only user-friendly but also have achieved most of the qualities of being strong as well as non-crackable passwords.**

Index Terms*—*Auto-generated, Password generator, Strong password pattern, Password cracking, Case sensitive words, Combination of numbers, Special characters, Password entropy, Brute Force Attack, Dictionary Attack, Randomization technique, Strength checkers

I. Introduction

Internet security is recently becoming a significant issue with the increasingly wide range of internet applications. Bank and commercial exchanges are now being carried out online in the form of internet banking and commercial electronic transactions. The level of information transmission is becoming more critical for occurring information leakage, and damages due to such leakages are more significant. User authentication is a necessary security element in the open network environment, and the use of simple authentication information also has some severe problems.

One problem is that it is easy for the attackers to guess passwords whenever the users often choose personal information, such as their ID or telephone number as passwords. The users do this to remember them quickly. Sometimes they use the same password for many web sites. They do this so that they do not have to remember too many passwords. Another problem is that users rarely choose passwords that are both hard to guess and easy to remember. To help users in choosing good passwords, many experts proposed different kinds of guidelines for following many policies. Another problem is that many of the deficiencies of password authentication systems arise from human memory limitations. We know that a maximally secure password with maximum entropy consists of a string having numerous random special characters as long as the system permits. But human memory can not remember such long as well as complex passwords. When humans do remember a sequence of items, those items must be familiar pieces of information such as words or familiar symbols. However, most people are much better at recalling the information when we encode them in multiple ways. So, password authentication involves a trade-off. Some passwords may be easy to remember (for example, best friends name, pets name, kids name, etc.) but also easy to crack through dictionary searches. Other passwords may be secure against guessing but challenging to recall. Besides, some users tend to create different passwords for their various online accounts. When users create different passwords for different accounts, they need to remember several passwords which may be problematic or confusing for them during use. In this case, for remembering all the passwords, they sometimes keep insecure written records of them. Having a written document of passwords is a terrible idea because this act is not free of password cracking attacks. Eventually, an attacker can easily guess the password.

978-1-7281-2530-5/19/$31.00 ©2019 IEEE

So to address this issue, in this paper, we have proposed an automated system to generate user-friendly and robust passwords by combining some texts and numbers. These passwords will be made from the user given information. So we can assume that they will be easier to remember than randomly generated online passwords. The generated passwords are unconjecturable because of the features of our methodology and can be used in many and different applications and internet services like social networks, secured systems, distributed systems, and online services. Our proposed password generator can achieve diffusion, randomness, and confusions which are very necessary and required in case an intruder tries to crack the generated passwords.

II. Motivation and Objectives

Every individual who works with different modern online services is concerned with his/her security and privacy for protecting personal information from the attackers. Password authentication is one of the popular authentication systems that has been used for many years for defending online accounts or services. At the same time, the users also need to create a strong password for protecting their services from a real-world attacker. Thus it is recommended to create a unique password with a healthy pattern so that they can protect it from the intruders. But usually, the users forget their passwords because it is not easy to remember a string with a robust and intricate pattern. To recall their passwords, they save them in their own devices, in notebooks or on sticky notes from where the passwords can get easily compromised by the intruders. So, it is always expected that the users should use strong passwords which they can easily remember without the help of writing down or saving them in their own devices or notebooks. Thus, the idea of auto-generating strong passwords based on their inputs will be convenient for them to remember easily.

Let us clarify our goal by describing an example. Suppose a user has been provided with a password like "#XeV65a$PzH08!" from a random online password generator in the web for his usage. These kinds of passwords are definitely secure, but are almost impossible to remember for the average user. On the other hand, our system will ask the user to provide five input texts and any two numbers for generating passwords. Let us assume the user provides this input: "mango, cat, red, kathy, ice, 67, 81". After that using them, our system will generate a password like "81KathY%m@ng0|". We see that both passwords have healthy patterns, so they are safe to use. Now if the user is given a choice of choosing one password from these two passwords "#XeV65a$PzH08!" and "81KathY%m@ng0|", the user should go for the second one. The reason is that the first password from the random generator is clueless to the user and very tough to remember. On the contrary, the second password from our system has two words and one number which they can not easily forget because they are based on the user's suggestions.

In summary, we aim to generate reliable and non-crackable passwords for the online services of the user based on the information provided by the individual which can be remembered easily.

III. Related Work

In [1], Shay et al. found out that the users struggled with new and complex password requirements, and in [2], Mazurek et al. found out that the users who complained about complex password policies used to create vulnerable passwords. Weir et al. [3] attempted to determine the effectiveness of using entropy, as a measurement of the security provided by various password creation policies. Bojinov et al. [4] introduced "Kamouflage", a new architecture for building theft-resistant password managers. This system forced an attacker who stole a laptop or cell phone with a Kamouflage-based password manager to carry out a considerable amount of online work before obtaining any user credentials. Kelley et al. [5] researched on defining metrics to characterize password strength and using them to evaluate password-composition policies. They analyzed 12,000 passwords collected under seven composition policies via an online study. They developed an efficient distributed method for calculating how effectively several heuristic password-guessing algorithms would guess passwords. They investigated on the resistance of passwords created under different conditions to guessing, the relationship between passwords explicitly created under a given composition policy and the relationship between guess-ability, as measured with password-cracking algorithms, and entropy estimates. Their findings could enlighten us on the understanding of both password-composition policies and metrics for quantifying password security. Komanduri et al. [6] also characterized the predictability of passwords by calculating their entropy, and detected that several commonly held beliefs about password composition and strength were inaccurate. Shay et al. [7] identified policies that were both more usable and more secure than commonly used policies that emphasized complexity rather than length requirements. Cormac et al. [8] investigated password authentication using tokens, biometrics, and authentication based on the multi-factor. Inglesent et al. [9] discovered that rather than focusing password policies on maximizing password strength and enforcing frequency alone, policies should be designed using HCI principles to help the user to set an appropriately strong password in a specific context of use.

Kelley et al. [10] developed a method provided for a user to generate a password for a software application accessible from a computer system which included a universal password generator (UPG). Kelsey [11] proposed a system that generated a new password by repeatedly iterating a

TABLE I
OVERVIEW OF THE RELATED WORKS

Year	Author	Background Study On Password	Password Authentication System	Password Generation	Password Crackability Test
1996	Udi Manber[24]	✘	✘	✔	✔
1997	Abadi et al.[23]	✘	✘	✔	✔
1997	Kelsey et al.[11]	✘	✘	✔	✘
2007	Kelley et al.[10]	✔	✘	✔	✘
2008	Marechal et al.[13]	✘	✘	✘	✔
2009	Weir et al.[14]	✘	✘	✘	✔
2009	Helkala et al.[12]	✘	✘	✔	✘
2010	Shay et al.[1]	✔	✘	✘	✘
2010	Weir et al.[3]	✘	✔	✘	✘
2010	Amico et al.[15]	✘	✘	✘	✔
2010	Inglesent et al.[9]	✘	✔	✘	✘
2010	Bojinov et al.[4]	✘	✔	✘	✘
2011	Komanduri et al.[6]	✘	✔	✘	✘
2012	Cormac et al.[8]	✘	✔	✘	✘
2012	Kelley et al.[5]	✘	✔	✘	✘
2012	Agrawal et al.[19]	✘	✘	✔	✘
2013	Mazurek et al.[2]	✔	✘	✘	✘
2015	Huh et al.[22]	✘	✔	✘	✔
2015	Houshmand et al.[16]	✘	✘	✘	✔
2016	Shay et al.[7]	✘	✔	✘	✘
2016	Yang et al.[25]	✘	✘	✔	✘
2017	Hitaj et al.[18]	✘	✘	✘	✔
2018	Masui[20]	✘	✘	✔	✘
2018	Aggarwal et al.[17]	✘	✘	✘	✔
2018	Mohammed[21]	✘	✘	✔	✘

hash function on the original master password. Helkala et al. [12] divided human-generated passwords into three categories: Non-word passwords, Mixture passwords, and Word passwords; depending on their overall structure. Within these categories, they analyzed the search-space reduction of several common password sub-structures.

Marechal et al. [13] surveyed various techniques that had been used in public or private tools in order to enhance the password cracking processes. Weir et al. [14] developed a probabilistic context-free grammar-based training set of previously disclosed passwords. Using this grammar they generated word-mangling rules, and from them, password guesses were used in password cracking. Their work showed that their approach provided a more effective way to crack passwords comparing to traditional methods by testing their tools and techniques on real password sets. In one series of experiments, training on a set of disclosed passwords, their approach was able to crack 28% to 129% more passwords than John the Ripper, a publicly available standard password cracking program. After that Amico et al. [15] developed a system for estimating password strength to be used as a basis for creating more effective proactive password checkers for the users and security auditing tools for the administrators. Following the work of Weir et al., Houshmand et al. [16] worked on defining metrics to help analyze and improve attack dictionaries. Using their approach to improve the dictionary, they achieved an additional improvement of 33% on their previous work by increasing the coverage of a standard attack dictionary. Aggarwal et al. [17] approached some of the cutting edge approaches of password cracking that might become more prevalent in the near future. Hitaj et al. [18] introduced PassGAN, a novel approach that replaced human-generated password rules with theory-grounded machine learning algorithms. Instead of relying on the manual password analysis, PassGAN used a Generative Adversarial Network (GAN) to autonomously learn the distribution of real passwords from actual password leaks and to generate high-quality password guesses.

Agrawal et al. [19] proposed the strong password generation technique by considering multiple input parameters of the cloud paradigm referred to as a multidimensional password. Masui [20] proposed a password generation system EpisoDAS. It took a seed string by a secret pattern drawn by the user's episodic memories and created a password. Mohammed [21] proposed a password generator using the genetic algorithm which achieved the requirements of a standard password. In [22], Huh et al. proposed a system-initiated password scheme and conducted a large-scale usability test. These works showed that usability was an important factor in designing password policies. Abadi et al. [23] and Manber [24] proposed an approach in which password was concatenated with an arbitrary value before hashing was applied. This random value was called password supplement. The mnemonic strategy had been recommended to help users generate secure and memorable passwords. Yang et al. [25] evaluated the security of 6 mnemonic strategy variants in a series of online studies involving 5,484 participants. In addition to applying the standard method of using guess numbers or similar metrics to compare the generated passwords, they also measured the frequencies of the most commonly chosen

sentences as well as the resulting passwords. Differences in the exact instructions had a tremendous impact on the security level of the resulting passwords. They examined the mental workload and memorability of 2 mnemonic strategy variants in another online study with 752 participants. Although perceived workloads for the mnemonic strategy variants were higher than that for the control group where no strategy was required, no significant reduction in password recall after 1 week was obtained. We got inspired by Yang et al.'s [25] work and devised a new way of making the passwords memorable to the users ensuring their fitness rather than using the mnemonic strategy. The full summary and overview of the body of works regarding password generation have been shown in table 1.

IV. Proposed Work

We aim to develop an application for password generation that allows the users to input some pieces of information that are easy for them to remember. Inputs will vary from individual to individual, and for this reason, passwords cannot be compromised easily. Input data provided by the user will consist of different interesting pieces of information such as favourite novel's name, the number of grand mother's children, secret dates etc. For ensuring security from the intruders, these data should ideally never have been exposed to any kind of social media. We propose an efficient algorithm to generate a strong password based on the provided information so that the adversary can not identify the password using different cracking algorithms.

V. Methodology

A. Algorithm

In our methodology, at first, we take input information from the user who is asking for a password. The input information consists of five texts and any two numbers. In our methodology, the number of input numbers "2" is not exact; it is kind of arbitrary. On the other hand, the number of input texts has some base-analogy. If we take only two texts as input, our system will use these two texts for password generation. Then if somehow the adversary can know or detect the input texts he can surely think that both these two texts are used in the password. So from the security perspective, we have increased the number of texts to five where it can ensure some randomness for password generation. Now, think about the case if the number of input texts increases from five to ten for password generation. We can say that this input will be very secured as it will have more randomness than using five input texts. Even if the adversary knows all the ten texts prompted to the system, he can never know the chosen ones from them by our generator for generating passwords. But from the user's perspective, inputting this large number of texts can make the process slow and complicated and also can annoy the users. So we have picked the number "5" for texts as standard for balancing between security and users' comfort.

Algorithm 1 Password Generating Algorithm

Require: Five texts, two numbers as input and the number of demanded passwords to be generated

1: **Begin Procedure**
2: *data* ← five texts and two numbers
3: N ← number of demanded passwords (from 1 to 10)
4: *specialchars* ← {@, $, !, #, %, &, (,), 0, 3, 8, <, |}
5: *punctuation* ← {@,*,+,-,:,",/,\,~,?,[,],{,},$,!,#,%,&,(,),_,<,|}
6: **while** $N > 0$ **do**
7: *string1* ← randomly selected from five texts
8: *string2* ← randomly selected from four texts
9: *Number* ← randomly selected from two numbers
10: *string1* ← capitalize (*string1*) some letters randomly
11: *string2* ← capitalize (*string2*) some letters randomly
12: *Final_string* ← merge *string1* and *string2* randomly
13: *Alphabets* ← {a, s, i, r, x, q, c, j, o, e, b, k, l}
14: *Final_string* ← *Final_string*. replace(Alphabets, *specialchars*)
15: *Final_string* ← randomly insert *Number* in the final string
16: either in the middle, or in the beginning or in the end of it
17: *Final_string* ← randomly append one special character from
18: *punctuation* at the end of the *Final_string*
19: C = len(*Final_string*)
20: **if** $C < 8$ **then**
21: *Final_string* ← append (8 - C) special characters randomly
22: **print** *Final_string*
23: **end if**
24: $N \leftarrow N - 1$
25: **end while**
26: **End Procedure**

Here the term "text" means the user can provide a word or multiple words written together (words should be merged without using space). Both the input texts and numbers should be convenient for the user to remember. For the same input set, the user can generate multiple passwords (up to 10) by our system according to their choice. As this is a run-time system, our generator neither saves any input/output of the user in the memory nor maintains any log. That is why our system guarantees no risks of the privacy breach.

After that, our system randomly picks two texts and one number from the seven choices to generate passwords. In Algorithm 1, this step is covered in lines 7-9. Then, our password generator applies a randomization technique to capitalize some letters of both the texts. After that, the system randomly chooses the positions of the selected texts and concatenates them in lines 10-12 in Algorithm 1.

Then the system appends a special character to the words according to the physical similarity of the special character with the letter. For example, if a word "mango" has the letter "a" and "o" then they are replaced by "@", and "0" because "@" and "0" look similar to "a" and "o". Replacing

some characters by similar special characters has the benefit of ensuring more safety against the adversaries while still generating passwords that are easy to remember. In Algorithm 1, this step is covered from lines 13-14. Subsequently, in lines 15-16, our system randomly inserts the selected number in some chosen positions of the combined text. For example, if the system chooses "178" between two numbers, "178" can be present in the front or in the middle or at the end of the generated password. At last, a random special character is appended at the end of the generated password which we can see in lines 17-18. The passwords will be kept producing depending on the number (from 1 to 10) demanded by the user during the input prompt and among the generated passwords the user can choose their favorite one. We are also keeping track of the run time of each password generation which is very fast. The proposed work is being done by using the Python programming language.

B. *Effectiveness of the Algorithm*

1) Managing ambiguous input: Our system can manage ambiguous inputs given by the user. The user is asked to provide five texts and two numbers. What if the user offers five words, each containing two letters only and two numbers which are of one digit each? As described above, the system takes two words/texts and one number to generate a password, so the length of the password becomes 2+2+1 = 5 characters only. But as far as we know, that does not satisfy the password policy [26]. Our system adds at least three random special characters in the password to fulfill the length policy. In Algorithm 1, we can find this adjustment from lines 20-22.

2) Ensuring User friendliness: Our system is very user-friendly. Our system does not generate a random and harsh password pattern for the users. What is the use of a password if the users can not remember it and need to set another password whenever they try to use it! Our system generates the passwords using the texts and numbers they want to remember. Our task is to create user-friendly passwords maintaining the standard policies [26] of password generation and to make them stronger so that the passwords can ensure safety concern.

3) Defending against various cracking attacks: In our methodology, we are very much concerned about the worth mentioning password cracking attacks named "Brute Force attack" and "Dictionary attack". We tried to safeguard our generated passwords from these two attacks. How we have achieved this is explained below.

Defending against the Brute Force Attack: In the field of cryptography, the brute-force attack means when an attacker submits many passwords or passphrases with the hope of eventually finding out the right one correctly. Brute-force attacks work by determining every possible combination that can make up a password and then test it to see if it is the correct password. The attacker thoroughly checks all possible passwords and passphrases until the correct one is found. During password-guessing, this method is very fast when it is used to check all short passwords. But for long passwords, it takes time. Longer passwords, passphrases and keys have more viable values, making this attack exponentially more challenging to crack than shorter ones. In our system, every generated password has a minimum length of eight. We know that Brute Force can easily crack the passwords that have these properties: repeated characters, consecutive numbers, only number sequences in ascending/descending order, only one sentence without numbers or special characters. So, we have defended our system from this attack by generating passwords with the combination of numbers, case sensitive words and special characters.

Defending against the Dictionary Attack: The dictionary attack is based on trying all the strings in a pre-arranged listing, typically derived from a dictionary (hence the phrase dictionary attack). On the contrary, in a brute force attack, a large proportion of the keyspace is searched meticulously whereas a dictionary attack tries only those possibilities which are deemed most likely to succeed. Dictionary attacks often thrive because many people have a tendency to choose short passwords. These passwords are ordinary words or common passwords or simple variants obtained by appending a digit or punctuation character. Dictionary attacks are relatively easy to defend, e.g. by using a passphrase or choosing a password that is not found in any dictionary or listing of commonly used passwords. The Dictionary attack can quickly occur if the password possesses only one word or name and sometimes only compound words (e.g. "fire works", "base ball", "grand mother" etc.). Our system can easily defend this attack because there is no chance in our system to use only one compound word in the password. If the user gives one compound word as input, there is 20% chance of having it in one piece in the password as our system just takes two texts/words from the five input texts (among them one is compound word), but with this word a minimum of two special characters, two numbers and another word is added. So, our generated passwords can defend against dictionary attacks.

4) Ensuring safety concerns for the user: Our system is a run-time method. It does not ask questions and does not keep any clue which can be tracked down by the adversary for his cracking usage. In our system, the users only give any five texts and two numbers as input, so the adversary can not guess the texts or the numbers provided by the password users. Moreover, the passwords are generated at run time and after using the passwords are gone forever. So there is also no risk of data leakage. We can say that our system is safe for users. They can generate one to ten passwords according to their preference from one input sample (our system will ask how many passwords from 1-10 they want). Then they can

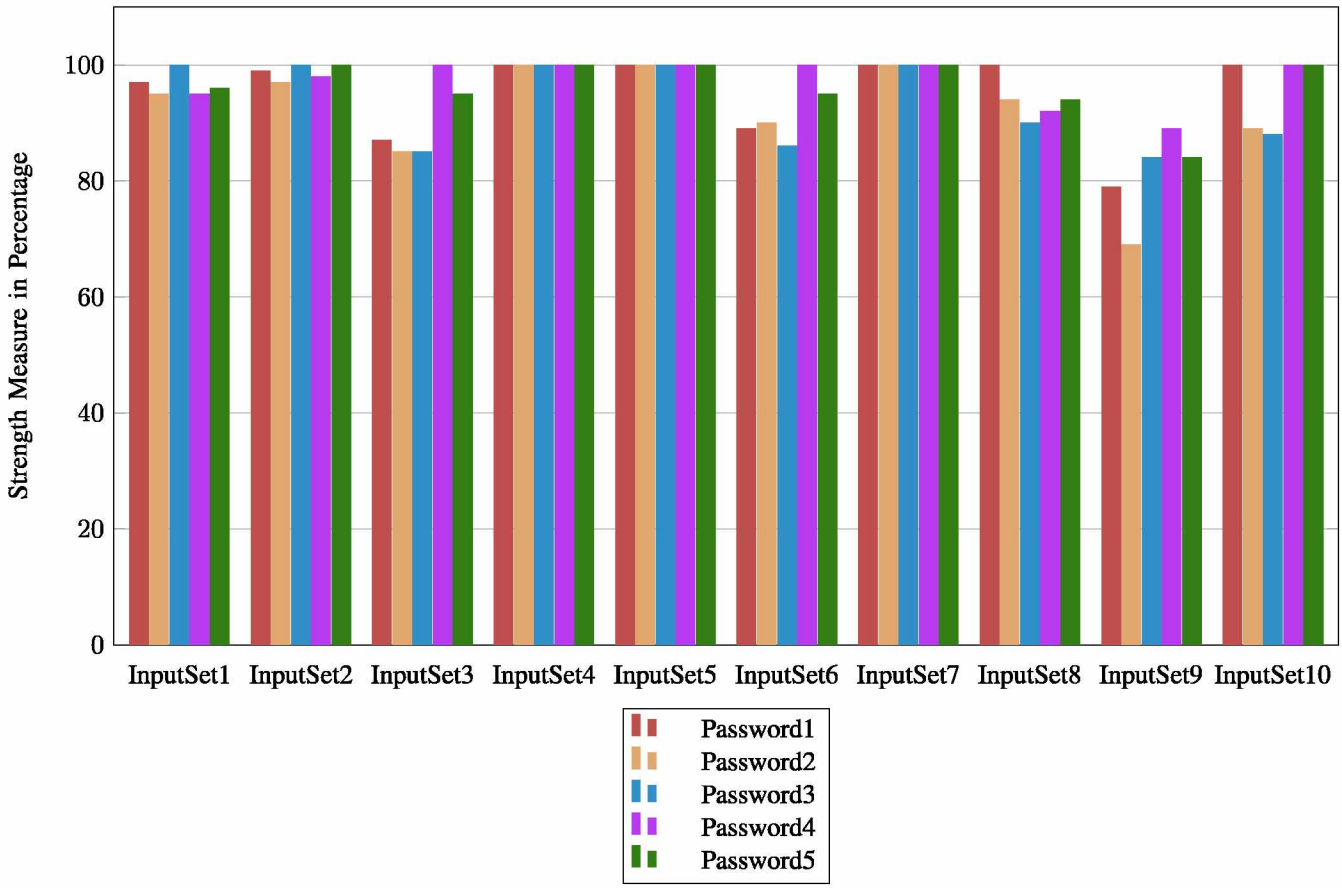

Fig. 1. Experimental results from Password Meter.

TABLE II
INPUT SAMPLES, GENERATED PASSWORDS FROM THEM AND CORRESPONDING RUN TIME IN SECONDS

Input Sample 1	Password	Runtime
asix, anik, irfan, turan, oliver, 29, 12	{urAn29iRfan-	0.0009999275
asix, anik, irfan, turan, oliver, 29, 12	@niK29tUr@n?	0.0009999275
asix, anik, irfan, turan, oliver, 29, 12	@Six29ol!vEr_	0.0009999275
asix, anik, irfan, turan, oliver, 29, 12	!rfAn29An!k]	0.0009999275

Input Sample 2	Password	Runtime
mango, cat, suha, kathy, rice, 7, 81	7kathY*maNgo|	0.0009999354
mango, cat, suha, kathy, rice, 7, 81	81suhA-m@ngO?	0.0010002999
mango, cat, suha, kathy, rice, 7, 81	katHy@81(aT]	0.0010002999
mango, cat, suha, kathy, rice, 7, 81	7cAt+R!(e|	0.0010002999

pick the preferred one from them for their use. For security purposes, our framework neither saves inputs nor generated passwords. So if the user does not like any password from the generated passwords, he/she can retry for another input sample and our system will require a second input from the user.

VI. Experimental Results

We have generated fifty passwords (five passwords for each input) from ten sample input data. We have used synthetic data as our input samples. A sample of generated passwords is shown in table 2 in this section. In our experiment, we at first have generated passwords for each of the respective input sample which consisted of five words/texts and two numbers each. Then to measure the strength and crackability we have tested all of them with four popular password strength checker online services named "The Password Meter" [27], Kaspersky Lab [28], "Password Checker Online" [29] and "Strength Test" [30].

"The Password Meter" checks the strength of the password pattern and gives a score of its strength measure from 0 to 100 including a ranking of four criteria: "Exceptional", "sufficient", "warning" and "failure". If passwords' strengths are good enough, then they will be in "exceptional" and "sufficient" criteria. If the score is not satisfactory, it means that the adversary can compromise the password and it will fall under the "warning" criterion. If the passwords don't meet the minimum standards, then they will be under

the "failure" criterion. Experimenting with "The Password Meter", for our 50 generated passwords we have an average score of 94.88% with a minimum of 69% and a maximum of 100% which denotes that our system-generated passwords are fairly strong. We can see the experimental results from "The Password Meter" in figure 1.

"Kaspersky Lab" checks the crackability of passwords using the Brute Force attack to be accomplished on a home computer. Using this online service for the 50 passwords from sample input data generated by our system, we have obtained a minimum cracking span of 90 days and maximum 1217000 days by the Brute Force attack which again proves that our system is not easily penetrable to the Brute Force attack.

"Password Checker Online" checks the crackability of passwords using both the Brute Force attack (on various machines: standard desktop PC, Fast Desktop PC, GPU, Fast GPU, Parallel GPUs, Medium Size Botnet) and the Dictionary attack. This checker gives an overall comment on the password depending on its strength. It also provides the cracking time estimation by the Brute Force attack and checks whether it is crackable under the Dictionary attack. By using this checker, our observation is that, for every password that we generated, the result for the Dictionary attack is "safe". Another inspection is that the Brute Force attack's cracking period for every password is similar to the results found from "Kaspersky Lab". And the overall comment for all passwords ranged from medium (with 69% strength) to excellent (with 100% strength). We have also checked the results of the Brute force attack and the Dictionary attack, especially for the passwords of medium strength, and the observation is that those are safe from both attacks. For a password whose strength was 69% by using Brute Force in Medium Size Botnet, it has been shown crackable in 14 days. So, this evaluation also proves that our system-generated passwords are robust, maintaining all the password policies and requirement criteria.

"Strength Test" checks the entropy measure of a password. Password entropy is a measurement of the unpredictability of a password. It is the opposite of an ordered pattern. The bigger the entropy is, the harder a password is to crack. In cryptography, password entropy is usually expressed in terms of bits. When a password is already known, has zero bit of entropy. A password that has one bit of entropy can be guessed on the first attempt. Whenever the entropy of a password is between 28-35 bits, it means that the password is very weak. If it is between 36-59, then it is reasonable (meaning somewhat secured for network and company usage). Whenever the password has 60+ entropy, it means it is a robust password. In this checker, our system-generated passwords have entropy ranging from 47 (minimum) to 88.2 (maximum). So, with this check, we can also prove that our generated passwords are hard to crack. We can see the experimental result of the strength test in figure 2.

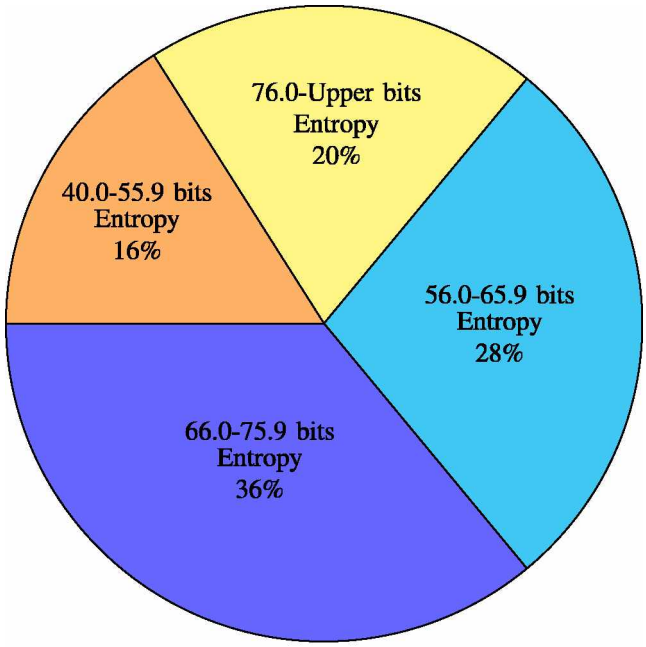

Fig. 2. Experimental results from Strength Test.

VII. Conclusion

In our proposed system, we are generating passwords by prompting the users to provide us with some texts and numbers (which they can easily remember) as inputs. Our generated passwords have ensured the minimum criteria of the password strength. Through various experiments, we have proved that our generated passwords are strong enough. Our generated passwords are different from the passwords of other online random password generators as they are not produced entirely at random. Instead, they are created from the inputs given by the user. They can be easily remembered as they contain only the texts and numbers which the users want to use in their passwords and can not forget easily. We also have experimented that our generated passwords can defend against the dictionary and the Brute Force attacks. However, there is another password cracking attack named "the social attack" which has not been covered in our work. Social attacks take an individual's system and personal information into consideration to try and help speed up the process of dictionary attacks. Say the adversary knows the victim's dog's name as Pickle, mom's name Jane and the birth date 01/07/1980. Then the adversary obviously will give this known data higher priority over other options. So the more the users will use secretive information (which have not been unveiled to any type of social media) as inputs, the more their passwords generated by our system can be safe from the social attack though we have planned nothing yet to defend this attack. In the future, we will extend our work and devise a method so that our generated passwords can defend

against the social attack.

For now, we are using now random data sets as input data. By conducting a user survey, we can collect a real data set which we can use in our generator to generate passwords. Through the use of a real data set, we can also check how much people can remember our generated passwords. Through another user study, we can also get feedback from the users to evaluate whether they can remember the passwords or not. We also have plans to add more features and steps to our algorithm to strengthen our system. We have a goal to add some exceptions in our methodology like (a) if the user inputs compound words like ("living room"; "ice cream"; "flower vase" etc.), our system will not receive these type of data and will prompt the user to provide different contexts as inputs to the system, (b) our framework will not accept the very common words like ("password", "keyword", "pass", "admin", "abc", etc.) as these words are the first choice of the adversary to try for cracking passwords. Our other plan for this work is to experiment with how the length of the input words and numbers affect the strength of the generated passwords. We will make an online application for our system so that the users do not have to wait for installing our system into their machines for use. We will also make it open source for the user's convenience.

ACKNOWLEDGMENT

We want to thank the reviewers for their valuable comments. This research was supported in part by the NSERC Discovery Grants (RGPIN-2015-04147).

REFERENCES

[1] Richard Shay, Saranga Komanduri, Patrick Gage Kelley, Pedro Giovanni Leon, Michelle L Mazurek, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. Encountering stronger password requirements: user attitudes and behaviors. In *Proceedings of the Sixth Symposium on Usable Privacy and Security*, page 2. ACM, 2010.

[2] Michelle L Mazurek, Saranga Komanduri, Timothy Vidas, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, Patrick Gage Kelley, Richard Shay, and Blase Ur. Measuring password guessability for an entire university. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 173–186. ACM, 2013.

[3] Matt Weir, Sudhir Aggarwal, Michael Collins, and Henry Stern. Testing metrics for password creation policies by attacking large sets of revealed passwords. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 162–175. ACM, 2010.

[4] Hristo Bojinov, Elie Bursztein, Xavier Boyen, and Dan Boneh. Kamouflage: Loss-resistant password management. In *European symposium on research in computer security*, pages 286–302. Springer, 2010.

[5] Patrick Gage Kelley, Saranga Komanduri, Michelle L Mazurek, Richard Shay, Timothy Vidas, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, and Julio Lopez. Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms. In *2012 IEEE symposium on security and privacy*, pages 523–537. IEEE, 2012.

[6] Saranga Komanduri, Richard Shay, Patrick Gage Kelley, Michelle L Mazurek, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, and Serge Egelman. Of passwords and people: measuring the effect of password-composition policies. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2595–2604. ACM, 2011.

[7] Richard Shay, Saranga Komanduri, Adam L Durity, Phillip Seyoung Huh, Michelle L Mazurek, Sean M Segreti, Blase Ur, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. Designing password policies for strength and usability. *ACM Transactions on Information and System Security (TISSEC)*, 18(4):13, 2016.

[8] Cormac Herley and Paul Van Oorschot. A research agenda acknowledging the persistence of passwords. *IEEE Security & Privacy*, 10(1):28–36, 2011.

[9] Philip G Inglesant and Martina Angela Sasse. The true cost of unusable password policies: password use in the wild. ACM, 2010.

[10] Edward E Kelley, Franco Motika, and James B Webb. Universal password generation method, January 30 2007. US Patent 7,171,564.

[11] John Kelsey, Bruce Schneier, Chris Hall, and David Wagner. Secure applications of low-entropy keys. In *International Workshop on Information Security*, pages 121–134. Springer, 1997.

[12] Kirsi Helkala and Einar Snekkenes. Password generation and search space reduction. 2009.

[13] Simon Marechal. Advances in password cracking. *Journal in computer virology*, 4(1):73–81, 2008.

[14] Matt Weir, Sudhir Aggarwal, Breno De Medeiros, and Bill Glodek. Password cracking using probabilistic context-free grammars. In *2009 30th IEEE Symposium on Security and Privacy*, pages 391–405. IEEE, 2009.

[15] Matteo Dell'Amico, Pietro Michiardi, and Yves Roudier. Password strength: An empirical analysis. In *2010 Proceedings IEEE INFOCOM*, pages 1–9. IEEE, 2010.

[16] Shiva Houshmand, Sudhir Aggarwal, and Randy Flood. Next gen pcfg password cracking. *IEEE Transactions on Information Forensics and Security*, 10(8):1776–1791, 2015.

[17] Sudhir Aggarwal, Shiva Houshmand, and Matt Weir. New technologies in password cracking techniques. In *Cyber Security: Power and Technology*, pages 179–198. Springer, 2018.

[18] Briland Hitaj, Paolo Gasti, Giuseppe Ateniese, and Fernando Perez-Cruz. Passgan: A deep learning approach for password guessing. In *International Conference on Applied Cryptography and Network Security*, pages 217–237. Springer, 2019.

[19] HA Dinesha and VK Agrawal. Multi-dimensional password generation technique for accessing cloud services. *Special Issue on:" Cloud Computing and Web Services", International Journal on Cloud Computing: Services and Architecture (IJCCSA)*, 2(3):31–39, 2012.

[20] Toshiyuki Masui. Episodas: Das-based password generation using episodic memories. In *Proceedings of the 2018 International Conference on Advanced Visual Interfaces*, page 73. ACM, 2018.

[21] Sura Jasim Mohammed. A new algorithm of automatic complex password generator employing genetic algorithm. *Journal of University of Babylon*, 26(2):295–302, 2018.

[22] Jun Ho Huh, Seongyeol Oh, Hyoungshick Kim, Konstantin Beznosov, Apurva Mohan, and S Raj Rajagopalan. Surpass: System-initiated user-replaceable passwords. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 170–181. ACM, 2015.

[23] Martin Abadi, T Mark Lomas, and Roger Needham. *Strenghtening Passwords*. Digital Equipment Corporation Systems Research Center [SRC], 1997.

[24] Udi Manber. A simple scheme to make passwords based on one-way functions much harder to crack. *Computers & Security*, 15(2):171–176, 1996.

[25] Weining Yang, Ninghui Li, Omar Chowdhury, Aiping Xiong, and Robert W Proctor. An empirical study of mnemonic sentence-based password generation strategies. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1216–1229. ACM, 2016.

[26] https://en.wikipedia.org/wiki/Password_policy.

[27] http://www.passwordmeter.com/.

[28] https://password.kaspersky.com/.

[29] http://password-checker.online-domain-tools.com/.

[30] http://rumkin.com/tools/password/passchk.php.