Password Guessing-Based Legacy-UI Honeywords Generation Strategies for Achieving Flatness

Muhammad Ali Fauzi
Department of Information Security and Communication Technology
Norwegian University of Science and Technology (NTNU)
Gjøvik, Norway
muhammad.a.fauzi@ntnu.no

Bian Yang
Department of Information Security and Communication Technology
Norwegian University of Science and Technology (NTNU)
Gjøvik, Norway
bian.yang@ntnu.no

Edlira Martiri
Department of Information Security and Communication Technology
Norwegian University of Science and Technology (NTNU)
Gjøvik, Norway
edlira.martiri2@ntnu.no

***Abstract*—The legacy-UI honeywords generation approach is more favored due to its high usability compared to the modified-UI approach that sometimes becomes unusable in practice. However, several prior arts on legacy-UI based honeywords generation methods often fail to obtain the security standard, especially the flatness criterion. In this work, we propose two legacy-UI honeywords generation strategies based on two password guessing methods: PassGAN and Probabilistic Context-Free Grammar (PCFG). Besides, we also introduce two hybrid strategies by combining PassGAN, PCFG, and random-based methods. We empirically examine the flatness of the proposed honeywords generation strategy against Top Password (Top-PW) attack using real-world datasets, instead of only providing heuristic security arguments. The experiment results show that three of the proposed methods (the PassGAN-based and the two hybrid methods) have lower flatness value than all previous legacy-UI methods and able to meet the "perfectly flat" criterion.**

***Index Terms*—Honeywords, PassGAN, Probabilistic Context-Free Grammar, Random, Hybrid, Password.**

I. Introduction

Password database leaks are not a new phenomenon and some of them were experienced by reputable corporations including Dropbox [1], Linkedin [2], Yahoo [3], etc. One of the important concern about the password data breach is the fact that most organisations take a long time to detect the breach [4], [5]. The failure to detect the leak early gives the attacker more than enough time to fully exploit the users' account. Therefore, it is notable to have not only a mechanism to enhance password-based authentication security but also a mechanism that can identify the password data breach immediately [6].

One of the promising approaches to tackle the data leak issue is honeywords system introduced by Juels and Rivet [7]. Honeywords provide a simple solution without significant changes required in both client and server-side [8]. For each user's password, the honeywords system generates some decoy passwords (termed as honeywords) and store them to the password file along with the original password after converting them into a hashed format. If the attacker manages to compromise the password file and decode the hashed password using the inversion attack, the attacker will find several passwords for each user and have to decide which one is the correct password. Once the attacker login using the wrong password, the data breach will be recognized.

The most challenging part for the system administrator to implement the honeywords system is how to create the decoy passwords that fulfill the security and usability standards of the system [9], [10]. Unfortunately, usability and security often collide with each other so that the two standards cannot be met [9]. Based on whether there is a modification on the user interface (UI) for password change, the honeywords generation methods can be divided into two classes: the legacy-UI and the modified-UI approach. Using a legacy-UI method, there is no need for change in UI. This method also does not interfere with the user password selection nor requires to notify the user about the use of honeywords. In contrast, the modified-UI approach needs to modify the registration and password change UI. This method also has to interfere with the user's password selection to obtain the desired security standard making the usability standard deteriorated. Moreover, some modified-UI based methods (e.g. take-a-tail in [7]) sometime become unusable in practice [9]. Therefore, legacy-UI is more recommended due to its high usability. However, this approach often fails to reach the security standard, especially the flatness criterion. In this work, we propose some legacy-UI honeywords generation strategies that are based on two password guessing methods (PassGAN and Probabilistic Context-Free Grammar (PCFG)) to achieve the flatness criterion. Besides, two hybrid strategies combining PassGAN, PCFG, and random-based methods are also introduced. We also evaluate the flatness of the proposed honeywords generation strategy empirically using a real dataset, unlike most of the previous work (e.g. [7], [10]–[12]) that only evaluate the flatness of their methods using heuristic security arguments.

II. Background and Related Work

A. Honeywords System

Honeywords system works by simply generating some artificial passwords (termed as honeywords) for each user account and store them into the password file together with the user's original password (termed as sugarword). This combination is

called sweetwords. The purpose of this approach is to make the attacker's job more complicated even after successfully steal and crack the hashed password file because the attacker still has to decide which password is the original one. Once the attacker entering a honeyword to log in, an alert will be triggered to inform the defender that a password breach had happened.

B. Honeywords Flatness Criterion

All honeywords generated should be indistinguishable from the user's original password so that sweetwords corresponded to the user's account should be equal probability to be chosen by the attacker as the correct password (sugarword). Ideally, since the system administrator has k sweetwords for each account, the success rate of the attacker to guess the account's correct password is $1/k$. However, several cases (e.g., an association between username and password, popular password [13], etc.) can make the attacker's success rate to guess the correct password higher.

The formula to calculate the attacker's success rate is as follows:

$$\epsilon = \frac{NoC}{NoA} \tag{1}$$

where NoC is the number of accounts whose passwords have been guessed correctly by the attacker and NoA is the total number of accounts. A honeywords generation technique is called ϵ-flat when the attacker's success rate given one chance to select a user's original password is ϵ. A "perfectly flat" generation method make the attacker only has success rate $\epsilon = 1/k$ while if the success rate ϵ is not much larger than $= 1/k$, it is considered as "approximately flat" [7]. It is also important to notice that some honeywords may allow more than one guess to minimize a false alarm. Therefore, Wang et al. [13] used a flatness graph as displayed in Fig. 1 to display the attacker's success rate when it is allowed to login using the wrong password more than once. A flatness graph outlines the success rate of the attacker to pick the original password against the number of login efforts per account. A point (x, y) on a curve in Fig. 1 means that the original password is correctly selected by the attacker with a probability y in the first x login trials, where $x \leq k$.

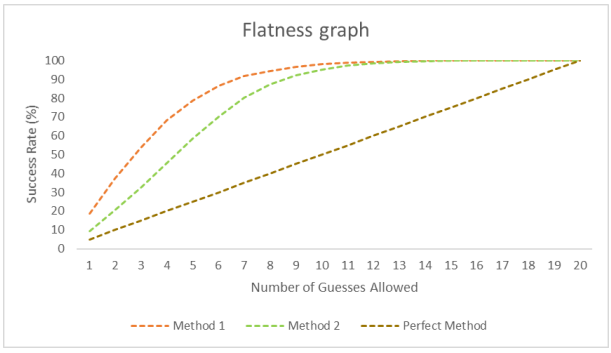

Fig. 1. The flatness graph

C. Legacy-UI Honeywords Generation Strategies

*1) **Chaffing by tweaking (CbT)**:* CbT [7] produces fake passwords by "tweaking" some characters from the user's original password. One of the most popular variants under this method is "chaffing-by-tail-tweaking". Under this approach, for a user's original password p_i, some honeywords are produced by replacing each of the last t characters of p_i with a random character of the same type: a letter is substituted by a letter, a digit by a digit, and a special character by a special character. For instance, if the p_i is "Lakers4%" and $t = 3$ is used, then the produced honeywords can be "Lakerw9@", "Lakeru1?", "Lakerr3*", etc.

*2) **Chaffing-with-a-password-model (CPM)**:* CPM [7] generates fake passwords by applying the same syntax as the password model. One of the variants of CPM is modeling syntax inspired by Bojinov et al. [14] that uses the user's original password p_i as a password model. First, the syntax of p_i is extracted. For example, if the p_i is "Lakers4%", then the syntax is L6D1S1 meaning the password has 6 letters, 1 digit, and 1 special character. The honeywords are then generated based on the syntax so that they can be "TgsRfv3!", "TGygha2?", "ewscpl8*", etc.

*3) **Close-number-formation (CNF)**:* CNF method [15] produces the honeywords by replacing numbers in the original password with other numbers that are close to them. This method maintains two lists: $num = \{1, 2, 3\}$ and $sig = \{+, -\}$. First, the system will find the number dp in p_i. Then, dp will be *tweaked* using the following formula:

$$dh = dp\ e(sig)\ e(num), \tag{2}$$

where $e(sig)$ and $e(num)$ indicate an element from the list sig and num, respectively. For example, if the original password is "Lakers2007", then the honeywords generated are "Lakers2008", "Lakers2009", "Lakers2010", "Lakers2006", "Lakers2005", and "Lakers2004".

*4) **Chaffing with "tough nuts" (CTN)**:* Unlike all previous methods, honeywords generated using CTN [7] is not necessarily related to the user's original password p_i. CTN is used to produce honeywords whose hashed form is hard to crack. This method can be applied by using some pseudo-random generator algorithm. Juels and Rivet [7] provided an example for "tough nut" as "9,50PEe]KV.0?RIOtcL-:IJ"b+Wol¡*[!NWT/pb". The purpose of this method is to make the attacker unable to crack the hashed form of this kind of passwords. It is stated in [7] that this method cannot be used alone, it should be combined with other methods to make the honeyword system more secure. If this method is used alone, the attacker will easily recognize the original password by selecting the one that can be cracked. This method can be powerful when it is combined with another method. The combination approach will give the attacker some sweetwords that can be cracked but some of them will be blank because they cannot be cracked. It is expected that in such a situation the attacker will be in doubt and may hesitate to make a login attempt using the cracked passwords.

*5) **Using passwords of other users (POU):*** The approach introduced by Erguler [11] utilizing other users' passwords as the honeywords. For a newly registered user, $k-1$ passwords are randomly selected from other users' passwords. Under this approach, the sweetwords are stored in a way that requires less storage than other previous methods.

III. The Proposed Approach

In this work, we propose four password guessing based legacy-UI honeywords generation strategies. The first two methods are based on two password guessing methods: PassGAN and PCFG. To make the attacker's task to distinguish the original passwords from the fake one hard, we have to produce honeywords that are similar to the original password. The weakness of most previous methods (e.g. CbT, CPM, and CTN) is they cannot generate honeywords that are similar to most users' passwords. Thus, the original password becomes notable and makes the attacker's job easier. According to some studies, most users use an easy-to-guess password for their account [16], [17]. Another report by Avast in 2019 [18] also show that 83% of American using weak passwords. The challenge for the system administrator in a honeywords system is to generate some easy-to-guess passwords to make the honeywords indistinguishable from widely used passwords. To obtain the objective, we employ PassGAN and PCFG due to its ability to mimic the characteristic of real-world password data. PassGAN and PCFG are originally intended to be a machine learning-based password guessing methods that learn from real password datasets to produce password guesses that have high similarity with passwords in the dataset. These two approaches do not need primary knowledge or intuition from experts about what kind of passwords frequently taken by users since it will autonomously learn from the real data. Therefore, PassGAN and PCFG can be a potential strategy for the system administrator to generate indistinguishable honeywords.

The other two proposed strategies are hybrid approaches. The first hybrid approach (Hybrid 1) combines PassGAN and PCFG to make the honeywords more diverse so that it will be harder for the attacker to predict the honeywords pattern and reveal the method that the defender uses to create the honeywords. Meanwhile, the second hybrid approach combines PassGAN, PCFG, and random-based method. This incorporation of a random-based method is based on the fact that even though most users tend to choose easy-to-guess passwords, there are still many users that select good passwords consist of a random combination of letters, digits, and special characters (e.g. using random password generators). This kind of user also needs to be accommodated. If we use PassGAN and PCFG to create honeywords for this random look-like passwords, the attacker that manages to crack the hashed password can conclude that the original password is the one that is random look like because all passwords in sweetwords look like easy-to-guess passwords, except one that is random look like. Therefore, we also use a random-based method in this hybrid approach to tackle this problem. By adding some random look like passwords into the sweetwords, even tough the user uses a random look-like password, the attacker cannot conclude which type of password is the original one because some passwords are random look like and some of them are easy-to-guess.

A. First Proposed Method: PassGAN-based Honeyword generation approach

PassGAN is a deep learning-based approach formerly designed by Hitaj et al for password guessing [19]. PassGAN uses Generative Adversarial Network (GAN), deep neural net architectures developed by Goodfellow et al. [20], that consist of two neural networks (generator and discriminator) that compete against each other. The generator network learns from the password dataset and then produces some new artificial passwords with a high likeness to the real passwords in the dataset. Then, the real passwords and the generated ones are combined before being fed into the discriminator. The discriminator task is to classify the combined data. The purpose of the generator network is to minimize the chance of the discriminator to correctly distinguishing the real and the artificial data, while on the contrary, the discriminator's objective is to improve the classification performance. Furthermore, the discriminator's classification performance is then employed as the basis to improve both network models. This process is iterated until it reaches a certain condition (e.g. the number of iteration). After the training finished, some honeywords are produced using the generator model of the PassGAN.

B. Second proposed method: PCFG-based Honeyword generation approach

Context-free grammar (CFG) is not a new phenomenon in the natural language field as it frequently used to create strings with particular structures. A CFG is defined as a 5-tuple $G = (S, N, P, \Sigma, R)$ where S is the start variable, N is a finite set of variables (non-terminals), P is a finite set of preterminals, Σ is a finite set of terminals, and R is a finite set of production rules of the form [21]:

$$S \rightarrow A, A \in N \tag{3}$$

$$A \rightarrow BC, A \in N, B, C \in N \cup P \tag{4}$$

$$T \rightarrow \omega, T \in P, \omega \in \Sigma. \tag{5}$$

A probabilistic context-free grammar (PCFG) is simply a CFG with assigned probability in each production. The probability of a string generation (derivation) is the product of the probabilities of the productions used in that generation. The probabilities are learned from the training data. In this work, we used a PCFG method designed by Weir et al [22]. First, a PCFG model is built by training on a real passwords datasets. Then, the model is used to produce some honeywords.

C. Third proposed method: Hybrid 1

Hybrid 1 method combines PassGAN and PCFG to produce honeywords. In this ensemble method, half of the honeywords are generated using PassGAN while the other half are created using PCFG method.

D. Fourth proposed method: Hybrid 2

Hybrid 2 method combines PassGAN, PCFG, and random-based method to produce honeywords. As seen in Fig. 2, since the recommended number of honeywords is 19 [7], we will create n fake passwords using the random-based method, where n is choosen randomly between 2 to 4, and the remaining honeywords will be generated equally by PassGAN-based and PCFG-based method.

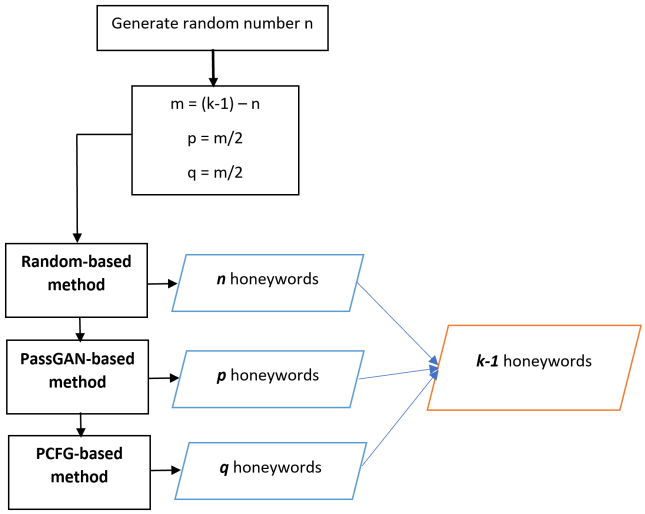

Fig. 2. The Proposed Hybrid 2 Honeywords Generation Approach

The random-based used in Hybrid 2 approach is similar to the 'tough nut' method. The purpose of this method is to generate passwords that look like strong passwords (include some letters, digits, and special characters). To create the honeywords using this random-based method, we generate a combination of random lowercase and uppercase letters, numbers, and punctuations. Unlike the 'tough nuts' method that creates a very long password, the length of the password in this method follows the system's password policy.

IV. Evaluating the Flatness of Honeywords Generation Methods

Most former studies do not assess the flatness of their honeywords generation methods empirically. They only present heuristic security reasons to justify the flatness of their method. In this work, we evaluate some honeywords generation methods empirically using some real-world password datasets. A game between the defender (e.g. system administrator) and the attacker is simulated. The defender produces $k-1$ honeywords for each original password (sugarword) in the password database. The defender can use several generation strategies for this task. We use $k = 20$ because it is the ideal number of k suggested by Juels and Rivet [7]. The sugarword and its honeywords are then joined and rearranged to form sweetword. The sweetwords are then saved into a correlated user account's record in a hashed form. In this game, the attacker is assumed to successfully compromise the password data and convert all the hashed sweetwords into plain texts. The attacker's next task is to guess the original password among the sweetwords. The evaluation used to measure the flatness of the honeywords generation methods are the attacker's success rate and flatness graph as described in subsection II-B.

A. The Defender's Strategies

In this experiment, several prior arts of legacy-UI honeywords generation will be used by the defender. The proposed hybrid method and each of the individual methods that compose it will also be employed by the defender. The following is the honeywords generation methods used by the defender:

1) **Chaffing by tweaking (CbT).** The CbT variant used is "chaffing-by-tail-tweaking" with $t = 3$.
2) **Chaffing-with-a-password-model (CPM).** The CPM variant used in this experiment is "modeling syntax".
3) **Using passwords of other users (POU).** The honeywords used is the other users' password in the database.
4) **Random/'tough nuts' method (Random).** The honeywords are created using a combination of random lowercase and uppercase letters, numbers, and punctuations.
5) **PassGAN.** A PassGAN model is first trained using a real-world password dataset and then it is used to produce the honeywords.
6) **PCFG.** A PCFG model is first trained using a real-world password dataset and then it is employed to generate the honeywords.
7) **Hybrid of PassGAN and PCFG (Hybrid 1).** Half of the $k-1$ honeywords are produced using PassGAN while the other half are generated using PCFG method.
8) **Hybrid of PassGAN, PCFG, and random-based method (Hybrid 2).** Like described in the Section III, 2 to 4 honeywords are generated using the random-based method and the remaining honeywords are generated equally by PassGAN-based andPCFG-based method.

The CNF is not used in this experiment because it cannot be used in a general password dataset. As mentioned before, the CNF cannot be used on passwords that do not contain a number. The CTN is also not used because the assumption of this method is the hashed form of honeywords generated by CTN cannot be cracked. Besides, the CTN method has been represented by the random-based method as they share similar generation techniques

B. The Attacker's Method

The attacker uses a Top Password (Top-PW) attack in this game. The Top-PW attack is a simple attack based on the fact that user passwords follow Zipf's law [23]. Users tend to choose popular passwords (e.g. 12345, password, etc.) to easily remember them. Therefore, among the sweetwords for each user, the most popular password is most likely the original one. Top-PW attack measures the popularity of each sweetwords by calculating its probability distribution in other real-world password datasets (e.g. leaked password datasets). Sweetwords that often appear on other password databases will have a large probability value. The sweetword with the

largest probability value is considered as the most popular password and it will be guessed as the correct password. This method is used for the attacker's method because its simplicity and it expose the weakness of most legacy-UI honeywords generation methods: cannot produce honeywords that are similar to the most users' password so that the original password becomes quite salient.

The formula to calculate each of sweetwords probability distribution in a password dataset is as follow:

$$PD\left(x\right) = \frac{count\left(x\right)}{|D|} \tag{6}$$

where $count(x)$ e number of sweetword x appears in a password dataset and $|D|$ is the total number of users' passwords in the dataset. The sweetwords are then ordered based on the probability value. Eventually, the sweetword with the highest probability is regarded as sugarword.

C. Dataset

Three password datasets are used for this experiment as follows:

1) **Sugarset**. The sugarset is the password datasets to simulate the original passwords (sugarwords) from users. In this work, a leaked password data from LinkedIn [24] is used as the sugarset. This dataset contains 3,000,000 real-world passwords.
2) **The defender's dataset**. The defender needs a password dataset only to train their PassGAN and PCFG models. The defender does not require the dataset for other honeywords generation methods. In this work, a leaked password data from Rockyou [25] is used as the defender's dataset. This dataset contains 21,315,673 real-world passwords.
3) **The attacker's dataset**. The attacker also needs a real-world password dataset to compute the probability distribution of each sweetword. It is assumed that the attacker does not know the dataset that had been used by the defender to generate the honeywords, in case the defender used PassGAN or PCFG method, so that the attacker uses different datasets. In this work, a leaked password data from Dropbox [26] is used as the attacker's dataset. This dataset contains 7,884,855 real-world passwords.

D. The Experiment Results

The experiment results when the attacker is given only one opportunity to guess the correct password are depicted in Fig. 3 while the results when the attacker is given more than one chance to predict the correct one are represented by a flatness graph displayed in Fig. 4. Based on results, as predicted, the stand-alone random-based is the worst method to generate honeywords. This method produces honeywords that are very different from the most users' passwords so that the attacker can guess the original passwords easily with a 94.48% success rate. To achieve the "perfectly flat" criteria, the honeywords generation strategies have to make

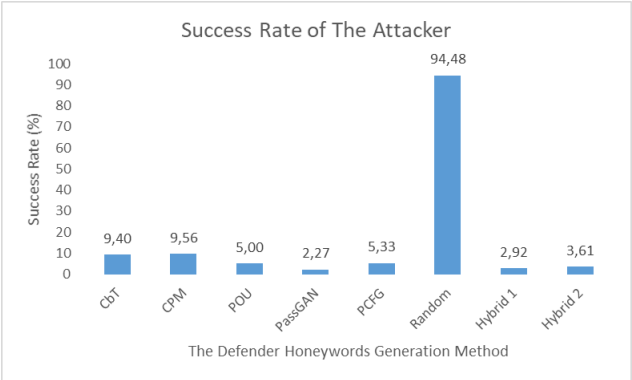

Fig. 3. The attacker's success rate given only one opportunity to guess the correct password.

the attacker's success rate equal to or less than $1/k$. Since $k = 20$ is used in this work, the maximum success rate of 5% should be obtained. To achieve the "perfectly flat" standard, the honeywords generation strategies have to make the attacker's success rate equal to or less than $1/k$. Since $k = 20$ is used in this work, the maximum success rate of 5% should be obtained. The only previous legacy-UI method that meet the criterion is POU with exactly 5% of success rate. Meanwhile, three of our proposed methods (PassGAN, Hybrid 1, and Hybrid 2) achieve the flatness goal with less than 5% of success rate. The proposed PCFG method can only fulfill the "approximately flat" standard as the attacker's success rate is 5.33% when the defender uses this method. The similarity of all these methods is they can produce honeywords that are similar to the most users' passwords.

The smallest success rate is achieved when the defender using the PassGAN-based honeywords generation method. The hybrid between the PassGAN-based and the PCFG-based methods (Hybrid 1) also provide a quite small success rate with 2.92%. Meanwhile, the attacker's success rate is slightly higher with 3.61% when the defender uses the Hybrid 2 method because it combines not only the PassGAN-based and the PCFG-based methods but also the random-based method.

Based on the flatness graph in Fig. 4, the proposed PCFG-based, Hybrid 1, and PassGAN-based methods provide the best flatness graph among all honeywords generation methods. Despite it only meet the "approximately flat" standard when the attacker given only one opportunity to guess the correct password, the PCFG-based method provides the best flatness graph. It means that this method can produce honeywords that have the same quality. A different method such as Hybrid 2 cannot generate honeywords that have the same quality because it contains a random-based method. As a consequence, the flatness graph of Hybrid 2 will rise sharply at a certain point. However, this method will be very useful when the user use random-generated password.

V. Conclusion

A good honeywords generation strategy should meet both the security and usability standards of the system. The legacy-

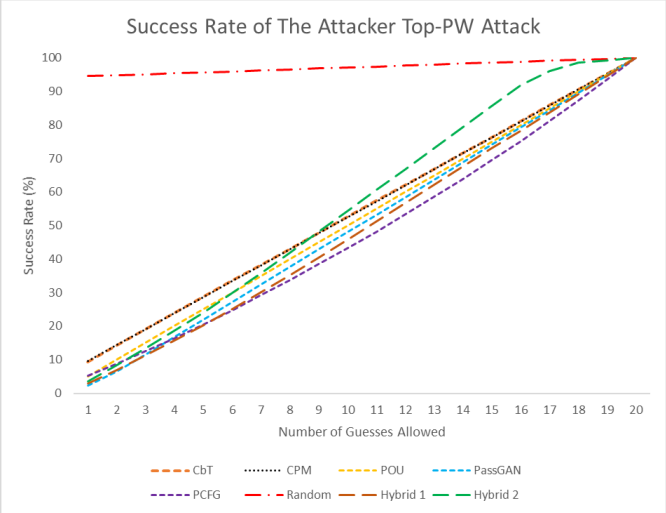

Fig. 4. The flatness graph.

UI approach is more recommended due to its high usability compared to the modified-UI approach that sometimes becomes unusable in practice. However, several prior arts on legacy-UI based honeywords generation methods often fail to obtain the security standard, especially the flatness criterion. In this work, we propose four password guessing based legacy-UI honeywords generation method that can achieve the flatness standard. The two proposed methods are stand-alone methods based on PassGAN and PCFG while the other two are hybrid approaches that combine PassGAN, PCFG, and random-based methods.

We empirically evaluate the flatness of the proposed honeywords generation strategies and several former generation methods against Top-PW attack using real-world datasets, instead of only providing heuristic security arguments. The experiment results show that almost all of the proposed methods have a lower flatness value than all previous methods and able to meet the "perfectly flat" criterion.

In future work, some attacking methods can also be used to evaluate the flatness of honeywords generation methods. One of the significant factors to concerned to design an attack is the user's behavior to choose passwords that are associated with their private data (e.g. favorite band, phone number, etc.). Based on the behavior, the use of targeted guessing attacks to evaluate the flatness of the honeywords generation methods would be relevant for future work.

REFERENCES

[1] P. Heim. (2016, Aug.) Resetting passwords to keep your files safe. [Online]. Available: https://blog.dropbox.com/topics/company/resetting-passwords-to-keep-your-files-safe

[2] M. J. Schwartz. (2016, May) Linkedin breach: Worse than advertised. [Online]. Available: https://www.bankinfosecurity.com/linkedin-breach-worse-than-advertised-a-9113

[3] R. Hackett. (2017, Oct.) Yahoo raises breach estimate to full 3 billion accounts, by far biggest known. [Online]. Available: https://blog.dropbox.com/topics/company/resetting-passwords-to-keep-your-files-safe

[4] S. Goolik. (2018, Sep.) Cyber security threats: Why detection takes so long. [Online]. Available: https://symmetrycorp.com/blog/cyber-security-threats-detection-takes-long/

[5] T. Brewster. (2016, Aug.) Why you shouldn't panic about dropbox leaking 68 million passwords. [Online]. Available: https://www.forbes.com/sites/thomasbrewster/2016/08/31/dropbox-hacked-but-its-not-that-bad/55c5c7ee5576

[6] D. Florêncio, C. Herley, and P. C. Van Oorschot, "An administrator's guide to internet password research," in *28th Large Installation System Administration Conference (LISA14)*, 2014, pp. 44–61.

[7] A. Juels and R. L. Rivest, "Honeywords: Making password-cracking detectable," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013, pp. 145–160.

[8] Z. A. Genc, S. Kardaş, and M. S. Kiraz, "Examination of a new defense mechanism: Honeywords," in *IFIP International Conference on Information Security Theory and Practice*. Springer, 2017, pp. 130–139.

[9] N. Chakraborty, S. Singh, and S. Mondal, "On designing a questionnaire based honeyword generation approach for achieving flatness," in *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. IEEE, 2018, pp. 444–455.

[10] N. Chakraborty and S. Mondal, "On designing a modified-ui based honeyword generation approach for overcoming the existing limitations," *Computers & Security*, vol. 66, pp. 155–168, 2017.

[11] I. Erguler, "Achieving flatness: Selecting the honeywords from existing user passwords," *IEEE Transactions on Dependable and Secure Computing*, vol. 13, no. 2, pp. 284–295, 2015.

[12] D. Chang, A. Goel, S. Mishra, and S. K. Sanadhya, "Generation of secure and reliable honeywords, preventing false detection," *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 5, pp. 757–769, 2018.

[13] D. Wang, H. Cheng, P. Wang, J. Yan, and X. Huang, "A security analysis of honeywords." in *NDSS*, 2018.

[14] H. Bojinov, E. Bursztein, X. Boyen, and D. Boneh, "Kamouflage: Loss-resistant password management," in *European symposium on research in computer security*. Springer, 2010, pp. 286–302.

[15] N. chakraborty and S. Mondal, "Towards improving storage cost and security features of honeyword based approaches," *Procedia Computer Science*, vol. 93, pp. 799–807, 2016.

[16] M. Dell'Amico, P. Michiardi, and Y. Roudier, "Password strength: An empirical analysis," in *2010 Proceedings IEEE INFOCOM*. IEEE, 2010, pp. 1–9.

[17] M. Dürmuth, F. Angelstorf, C. Castelluccia, D. Perito, and A. Chaabane, "Omen: Faster password guessing using an ordered markov enumerator," in *International Symposium on Engineering Secure Software and Systems*. Springer, 2015, pp. 119–132.

[18] Avast. (2019, May) 83 % of americans are using weak passwords. [Online]. Available: https://press.avast.com/83-of-americans-are-using-weak-passwords

[19] B. Hitaj, P. Gasti, G. Ateniese, and F. Perez-Cruz, "Passgan: A deep learning approach for password guessing," in *International Conference on Applied Cryptography and Network Security*. Springer, 2019, pp. 217–237.

[20] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[21] Y. Kim, C. Dyer, and A. M. Rush, "Compound probabilistic context-free grammars for grammar induction," *arXiv preprint arXiv:1906.10225*, 2019.

[22] M. Weir, S. Aggarwal, B. De Medeiros, and B. Glodek, "Password cracking using probabilistic context-free grammars," in *2009 30th IEEE Symposium on Security and Privacy*. IEEE, 2009, pp. 391–405.

[23] D. Wang, H. Cheng, P. Wang, X. Huang, and G. Jian, "Zipf's law in passwords," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 11, pp. 2776–2791, 2017.

[24] Leaks linkedin. [Online]. Available: https://hashes.org/leaks.php?id=68

[25] Brannondorsey. (2018) Passsgan. [Online]. Available: https://github.com/brannondorsey/PassGAN/releases/download/data/rockyou-train.txt

[26] Leaks dropbox. [Online]. Available: https://hashes.org/leaks.php?id=91