A Honeywords Generation Method Based on Deep Learning and Rule-Based Password Attack

Kunyu Yang[1], Xuexian Hu[1(✉)], Qihui Zhang[1], Jianghong Wei[1], and Wenfen Liu[2]

[1] State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou, China
xuexian_hu@hotmail.com

[2] Guilin University of Electronic Technology, Guilin, China

Abstract. Honeywords is a simple and efficient method that can help the authentication server to detect password leaks. The indistinguishability between generated honeywords and real passwords is the key to the honeywords generation methods. However, the current honeywords generation methods are difficult to achieve that and are vulnerable to the Top-PW attack.

In order to improve the security of the honeywords generation method, this paper combines the deep learning and rule-based password attacks to propose a new honeywords generation method called GHDR. The method first builds a deep learning model to learn whether the rules used in the rule-based password attack are effective for different passwords. When a user sets a password, effective rules will be selected by the model according to the entered password, and then these selected rules will be used to transform the user's password to generate honeywords. Experimental results show that the proposed honeywords generation method can better resist Top-PW attacks than the state-of-the-art methods.

Keywords: Honeywords · Deep learning · Password attack · Top-PW attack

1 Introduction

Recently years, network attacks against authentication servers of many well-known websites have occurred frequently, resulting in a large number of data files storing passwords have been stolen. These network incidents pose a great threat to users' information security. Although most websites store passwords in salted-hash to improve security, this method could not prevent attackers from recovering passwords from the leaked password file [22]. With the development of password attack technologies (e.g. PCFG-based model [24], deep-learning-based model [18] and the attack model using personal information [23]) and the enhancement of computer computing power, it becomes easier for an attacker

W. Shi et al. (Eds.): SPNCE 2021, LNICST 423, pp. 294–306, 2022.
https://doi.org/10.1007/978-3-030-96791-8_22

to quickly recover most passwords from a password file. What's more serious is that the behavior of stealing password files usually does not affect the function of the server, leading to these attacks are difficult to be perceived and tracked. The website administrators do not discover the password files leak until the attackers make full use of the data and poster them online, which is usually months or even years after the passwords leak occurred.

In 2013, Juels et al. [15] proposed using honeywords to detect whether the password files in the authentication system have leaked. This method generates multiple decoy passwords called honeywords for each account, and stores them together with the real password. Assume that an attacker can still obtain the password file and recover the passwords from it. After that, the attacker still needs to identify which one is the user's real password from the password list mixed with multiple honeywords. The server can detect passwords leak if the attacker login using honeywords, and then take measures such as restricting login and requiring users to modify passwords to ensure user's information security.

According to whether the password setting interface in the client needs to be modified, the honeywords generation method can be divided into two categories, one is Legacy-UI [2,7,14] and the other is Modified-UI [3–6]. Although studies [4,6] have shown that the Legacy-UI methods are weaker than the Modified-UI methods in terms of security, the Legacy-UI methods are more feasible in practice, because no modification is required on the client without increasing users' learning costs.

In the paper [15], Jules et al. proposed several Legacy-UI methods to generate honeywords by replacing part of characters in passwords. These methods based on character replacement are highly efficient, but the generated honeywords are significantly different from the real passwords, resulting in that the honeywords can be easily identified. In order to achieve the goal of generating honeywords that are indistinguishable from real passwords, Erguler [11] proposed randomly selecting some passwords of registered users as honeywords of a new account. But for the sake of satisfying random selection, the authentication server must continuously regenerate honeywords for all users as the number of users grows, which greatly increases the computational burden of the server. Taking into account the advantages and disadvantages of the above two methods, many researchers have found that using password attack methods to generate honeywords is a feasible way. Wang et al. [22] combined the probabilistic-context-free-based method and Markov-based method to generate honeywords, Fauzi et al. [12]proposed the use of PassGAN-based method, and Dionysiou et al. [10] proposed the use of representation learning. Since the real password satisfies Zipf's distribution [21], and the distribution of honeywords generated by these methods still has a gap with the distribution of real passwords, so these methods are vulnerable to a specific attack called Top-PW attack. The Top-PW attack refers to that an attacker calculates the frequency of each sweetword (including honeywords and the real password) in the known real password dataset, and takes the sweetword with the highest frequency as the real password.

In this paper, in view of the problem that the honeywords generation method is vulnerable to Top-PW attacks, we combine deep learning and rule-based pass-

word attack methods to propose a new honeywords generation method called GHDR. To verify the security of the proposed method, we use the GHDR method to generate honeywords for the accounts in the real password dataset, and simulate the behavior of an attacker using the Top-PW attack to identify the correct password after obtaining the password file. The experimental results show that the proposed honeywords generation method can better resist Top-PW attacks than the state-of-the-art methods. We also test the time required for the GHDR method to generate honeywords, and the results indicate that the method is usable in practice.

2 Preliminaries

2.1 Authentication System with Honeywords

The symbols we used to describe the authentication system with honeywords and their corresponding meanings are given in Table 1.

Table 1. Symbols used to describe the authentication system with honeywords

Symbols	Meanings
$H()$	hash algorithm
u_i	ID of a certain user
p_i	real password of u_i
honeywords	decoy passwords generated by system for u_i
W_i	including p_i and corresponding honeywords, also called sweetwords
k	the number of elements in W_i
c_i	index of p_i in W_i

Compared with the traditional authentication system, the authentication system with honeywords adds a Honey-checker server in addition to the identity authentication server. The Honey-checker server is considered to be safe and reliable. It connects to the identity authentication server through an encrypted and trusted channel, and only allows adding, modifying, and verifying operations. The system stores the user ID and corresponding sweetwords (i.e. (u_i, W_i)) in the identity authentication server, and stores the user ID and the index of real password in the sweetwords (i.e. (u_i, c_i)) in the Honey-checker server.

Figure 1 shows how the authentication system with honeywords works. When a user registers, the identity authentication server generates multiple honeywords for the user u_i and forms W_i together with the password p_i set by the user, where the position of p_i in W_i is random. Then the identity authentication server uses the hash algorithm $H()$ to calculate the value of each element in W_i and store the tuple $(u_i, H(W_i))$. After that, the identity authentication server sends the user ID u_i and c_i (i.e. the index of p_i in W_i) to the Honey-checker, and the tuple (u_i, c_i) will be stored in the Honey-checker server.

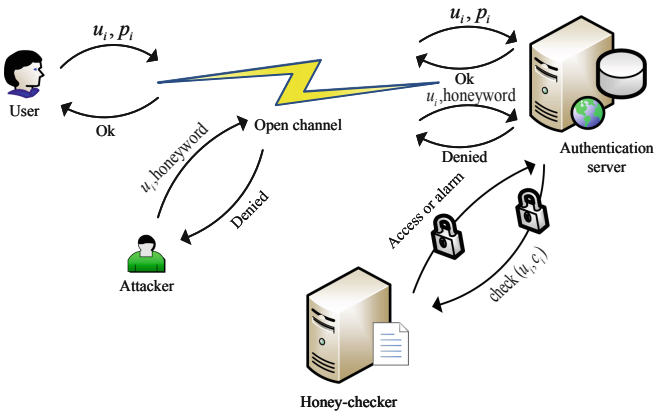


Fig. 1. Illustration of authentication system with honeywords

When a user logs in, the identity authentication server uses $H()$ to calculate the hash value $H(p_i')$ of the password p_i' entered by the user u_i and checks whether $H(p_i')$ is in $H(W_i)$. If not, it means that the user has entered the wrong password. If it is, it means that the user may have entered the correct password or an attacker has entered a honeyword, then the identity authentication server needs to use the honey-checker for further verification. The identity authentication server sends u_i and the index c_i' of $H(p_i')$ in $H(W_i)$ to the Honey-checker. Honey-checker checks whether c_i' and c_i are equal. If true, the user can log in successfully. If not, it means the login operation is using a honeyword, and the system issues an alarm.

2.2 Rule-Based Password Attack

The rule-based password attack is a kind of heuristic password attack algorithm. It can simulate the user's password generation behavior by designing various transformation rules (e.g. insertion, deletion, and reordering) to transform the strings in the carefully designed password dictionary. As early as 1979, Morris et al. [19] proposed the use of the rule-based attack to analyze the strength of passwords. Subsequent proposed password cracking tools (e.g. Hashcat, John and Ripper) are all designed with many professional rule sets and used to rule-based password guessing attacks. The methods used in papers [8,16] to implement password guessing attacks are also rule-based.

We take the rule set used in the well-known password attack tool Hashcat as an example to further illustrate how to execute a rule-based password attack. The rules in Hashcat are to express how to transform a password with custom symbols. Some examples of rules and their corresponding meanings are given in Table 2. Take *co1@* as an example, where *c* means capitalizing the first character

Table 2. Examples of rules in a rule set used by Hashcat

Rules	Meanings
co1@	Capitalize the first letter and lower the rest, and overwrite character at position at 1 with @
dO02	Duplicate entire word, and delete 2 characters starting from the position 0
C]	Lowercase first found character and upper the rest, then delete the last character

of the string and lowercasing other characters; *oNX* means overwrite the character at position N with X. We can get *P@ssword* by applying the rule *co1@* to *password*.

3 GHDR: The Proposed Method

3.1 Inspiration

On the one hand, generating honeywords that are indistinguishable from the real password is the key to a honeyword generation method [1,11,22]. On the other hand, the honeyword generation process must be time-saving, otherwise, using it will cause a poor service experience for users, which is unacceptable for service providers. In this paper, we intend to construct a method that can quickly generate honeywords similar to real passwords and can better resist Top-PW attacks.

The rule-based password attack can transform the given password to generate guessing passwords close to real passwords. Therefore, applying the rule-based password attack to honeywords generation has certain advantages. However, Pasquini et al. [20] pointed out that not all the rules are applicable to a certain password. The generated password obtained by transforming the password with an inapplicable rule is different from the real password. Inspired by the research in paper [20], a deep learning model for finding the appropriate rules for each password is built. Then we propose to use this model to generate honeywords, that is, use the model to match the password entered by a user with appropriate rules, and then take the generated passwords transformed by these rules as honeywords.

3.2 Construction of the Deep Learning Model

Denote the rule set as R, and a rule in it is denoted by r_i; denote the password dictionary as D, and a password in it is denoted by d_i. The goal of our deep learning model is to match the appropriate rules for each password d_i. This can be regarded as a multi-label classification problem, that is, each rule r_i is a label, and each d_i has multiple labels. We use 0 and 1 to indicate that a rule r_i and d_i are inappropriate and appropriate respectively.

In order to construct labeled training samples, we construct a target password set X, which consists of a large number of real passwords. If the generated password obtained by using the rule r_i to transform d_i appears in X, it means that d_i and r_i are appropriate, otherwise, it means that they are inappropriate. It can be expressed as:

$$f(d,r) = \begin{cases} 0 \; r(d) \notin X \\ 1 \; r(d) \in X \end{cases}, \tag{1}$$

where $r(d)$ represents the generated password obtained by using r to transform d. Based on this, a label y_i with $|R|$ dimensions can be generated for a password d_i in D, i.e. $y_i = [f(d_i, r_1), [f(d_i, r_2), \ldots, [f(d_i, r_{|R|})]$.

In order to meet the requirement that the honeywords generation process must be time-saving, we use Gate Convolutional Neural Network (GCNN) [9] to build the model. GCNN is a type of Convolutional Neural Network and it can better exploit GPU hardware to accelerate the training process. The researches in papers [9,13] show that the effect of using the GCNN model on natural language processing exceeds that of partial recurrent neural networks. The neural network structure of the model used in GHDR is shown in Figs. 2, and 3 shows the structure of the GCNN block used in it. During the model training, passwords are divided into character sequences as input. First, the Embedding layer is used to obtain the embedding representation of the characters, and then two GCNN blocks are used to extract features. Finally, the dimensionality of the result is changed through two fully connected layers.

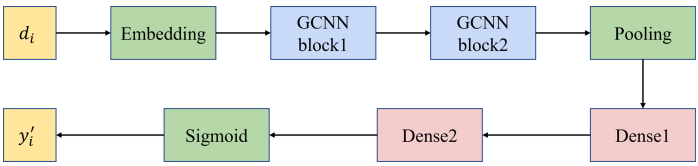


Fig. 2. The neural network structure of the model used in GHDR

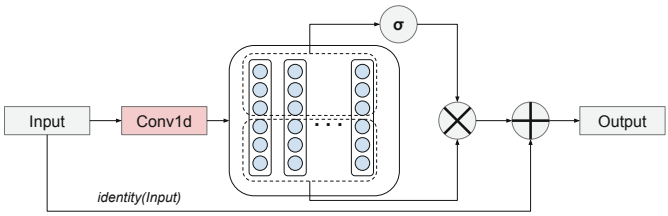


Fig. 3. The neural network structure of the GCNN blocks

3.3 Training of the Deep Learning Model

Dataset. We use the passwords leaked from *163.com* and *Twitter* as the password dictionary D, and the passwords leaked from *Mate1* and *Dodonew* as the target password set X. The above dataset information is shown in Table 3. The rule set used in this article is *generated* constructed in the Hashcat, which contains 14,728 rules in total. After removing the rules that do not have any change to passwords and some rules used to reject plains, we use 13,470 of them.

Table 3. Dataset used in model training

Dataset	Web service	Language	Password nums
163.com	Mailbox	Chinese	36,046,377
Twitter	Social forum	English	38,470,995
Mate1	Online dating	English	25,570,008
Dodonew	E-commerce	Chinese	15,578,470

Training Process. The passwords in D and X that contain non-printable ASCII characters and the password length is not between 6 and 20 are deleted, and we remove the duplicate passwords. Then all the rules in the rule set are executed for each password in D, and the labeled training samples are constructed according to Eq. 1. Finally, we get about 1.6 million valid training samples.

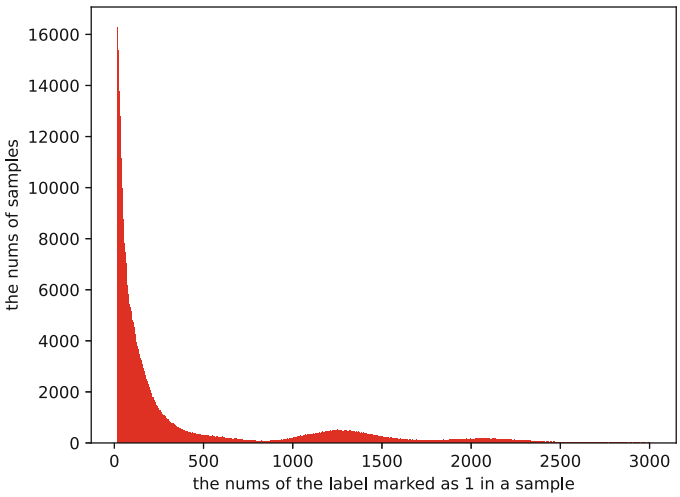


Fig. 4. The distribution of the positive instances in the training samples

Observing the training sample, we find that the positive and negative labels in the samples are seriously unbalanced. Specifically, there are far fewer labels marked as 1 in a sample than labels marked as 0. Figure 4 shows the relationship between the number of the label marked as 1 in a sample and the number of samples. Although there are 13,470 labels for a sample, less than 500 labels are marked as 1 in most samples. Aiming at this problem, Focal loss [17] is adopted as the loss function, it can be expressed as:

$$\mathcal{L} = \begin{cases} -\alpha(1-y_i')^{\gamma} log y_i' & , y_i = 1 \\ -(1-\alpha) y_i'^{\gamma} log(1-y_i') & , y_i = 0 \end{cases}, \tag{2}$$

where y_i is the ground-truth label, y_i' is the result assigned by the model, α is a parameter that can adjust the weight, and γ is an adjustable factor. According to the recommended values in the paper [17], α is set to 0.25, and γ is set to 2 in our experiment.

During training, the passwords in the training samples are input into the model, and the output result $\boldsymbol{y'}$ is obtained through the calculation of the neural network. Then use the focal loss to calculate the loss between $\boldsymbol{y'}$ and the label $\boldsymbol{y}$ of the sample. Finally, the Adam optimization algorithm is used to adjust the model parameters through backpropagation until the value of the loss function no longer decreases.

3.4 Honeywords Generation Using the Deep Learning Model

The trained model is used to generate honeywords, the specific process is shown in Algorithm 1.

Algorithm 1: Honeywords generation

Input: User's password p_u, rule set R, model M, the nums of honeywords $k-1$
Output: Honeywords list *honeywords*

Initialize *honeywords* to an empty list
$pred \leftarrow M(p_u)$
$R_{sorted} \leftarrow$ sort the rules in R in descending order according to the probability in *pred*
while *honeywords*.length $< k$ **do**
    $r \leftarrow R_{sorted}$.pop(0)
    $candidate \leftarrow r(p_u)$
    if *candidate* not in *honeywords* and $candidate \neq p_u$ **then**
        add *candidate* to *honeywords*
    end
end
return *seqs*

When generating honeywords for a user, the password set by the user is input into the model, and the predictions of the model are obtained firstly. The predicted values represent the probabilities that the generated passwords obtained

by transforming the user's password using the corresponding rules appear in the real password dataset. Then all the rules are sorted in descending order of probabilities, and these rules are used in turn to transform the user's password to obtain generated passwords. If a generated password is different from the existing honeywords and the user's password, it can be used as a new honeyword until $k-1$ honeywords are generated. Finally, the password set by the user is randomly inserted into a random position in the honeywords list to form sweetwords. User ID and sweetwords (stored in salt-hash) are stored in the authentication server, and the honey-checker stores the user ID and the index of the user's password in the sweetwords.

4 Security Analysis and Efficiency Evaluation

4.1 Security Analysis

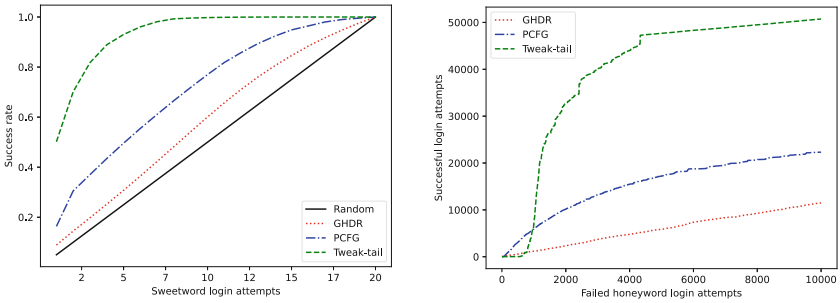
We use the flatness graph and success-number graph proposed by Wang et al. [22] to test the resistance against Normalized Top-PW attack ability of the proposed GHDR method, and compare with the methods Tweaking-tail and Chaffing-with-a-password-model (PCFG-based [24] model is used in this paper) proposed in the paper [15].

We use the GHDR method to generate honeywords for real passwords leaked from *Gmail* and *Mopu*, and assume that the attacker uses passwords leaked from *Rockyou* and *7k7k* as known passwords to perform the Top-PW attack. The information of these datasets is shown in Table 4. The attacker aims to distinguish the real password from k sweetwords by treating the authentication server as a querying oracle. But the number of queries that an attacker can perform is not unlimited. The honey-checker will record the number of login attempts using honeywords. When the number of such attempts against an account exceeds t_1, the attacker will be prohibited from continuing to login to this account; when the total number of such attempts exceeds t_2, the attacker will be prohibited from trying to login to any account.

Table 4. Datasets used in model security analysis

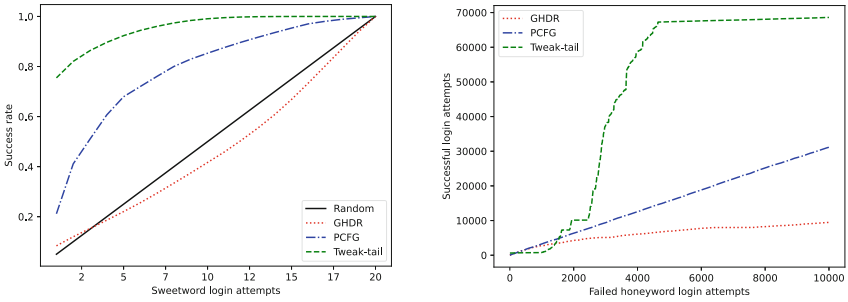
Dataset	Web service	Language	Password nums
RockYou	Social forum	English	32,368,961
7k7k	Gaming	Chinese	18,576,977
Gmail	Mailbox	English	468,992
Maopu	Entertainment forum	Chinese	294,912

In the experiment, k is set to 20, t_1 is set to 3, and t_2 is set to 10000. The results of our experiment are shown in Figs. 5 and 6. The *Random* in the Flatness graph means that an attacker cannot obtain any additional information, and can only randomly select one from sweetwords as the real password.



(a) Flatness graph (b) Success-number graph

Fig. 5. Security analysis on password leaked from Gmail



(a) Flatness graph (b) Success-number graph

Fig. 6. Security analysis on password leaked from Maopu

From the Flatness graphs in Figs. 5(a) and 6(a), we can see that the proposed GHDR method shows higher security against Top-PW attacks on average compared with the Tweak-tail method and the PCFG-based method. Specifically, using the GHDR method to generate honeywords for the *Gmail* dataset, the probability that an attacker can successfully identify the real password with one guess is about 8.89%, while he can successfully login 16.35% accounts against the PCFG-based method and 50.16% accounts against the Tweaking-tail method, which are about 1 time and 5 times higher than the GHDR method respectively. Using the GHDR method to generate honeywords for the *Maopu* dataset, the probability that an attacker can successfully identify the real password with one guess is about 8.36%, while he can successfully login 21.17% accounts against the PCFG-based method and 75.46% accounts against the Tweaking-tail method, which are about 2 times and 8 times higher than the GHDR method respectively. What is interesting is that the number of successful attacks with more than two guesses against GHDR method is less than *Random* in Fig. 5(a), which shows that the honeywords generated by GHDR have a certain negative impact on the Top-PW attack.

From the Success-number graphs in Figs. 5(b) and 6(b), we can find that the proposed GHDR method performs better than the Tweak-tail method and the PCFG-based method in the worst case. Specifically, using the GHDR method to generate honeywords for the *Gmail* dataset, approximately 2.45% of the total accounts can be guessed by an attacker under the limit, while he can successfully guess 4.76% accounts against the PCFG-based method and 10.82% accounts against the Tweaking-tail method, which are about 1 time and 4 times higher than the GHDR method respectively. Using the GHDR method to generate honeywords for the *Maopu* dataset, approximately 3.21% of the total accounts can be guessed by an attacker under the limit, while he can successfully guess 10.56% accounts against the PCFG-based method and 23.26% accounts against the Tweaking-tail method, which are about 2 times and 6 times higher than the GHDR method respectively.

The experimental results show that the proposed GHDR method can resist Top-PW attacks more effectively compared to Tweaking-tail and PCFG-based models.

4.2 Efficiency Evaluation

The use of honeywords in the authentication system is inevitably increasing the time required for users to set passwords. Network application services must consider the time required to generate honeywords. If the consumed time is too long, it will bring a negative service experience to users. We have tested the time required to generate honeywords using the GHDR method and compared it with the time needed by the PCFG method and the Tweak-tail method.

We employ a host with a configuration of CPU Intel(R) Xeon(R) Gold 6136 CPU @ 3.00 GHz, 128 GB DDR4 DRAM, and GeForce RTX 2080 Ti GPU to evaluate the proposed method. The time needed to generate honeywords for an account is shown in Table 5.

Table 5. Dataset used in model security analysis

Honeywords generation method	Average time needed for an account
GHDR	3.06 ms
PCFG-based	0.07 ms
Tweak-tail	0.04 ms

Although the average time required for the proposed GHDR method to generate honeywords for an account is much higher than that of the PCFG-based method and the Tweak-tail method, it still only takes about 3.06 ms. Compared with the general network delay (10 ms–50 ms), it is difficult for users to perceive this time. It can be proved that the proposed GHDR method is feasible in practice.

5 Conclusion

This paper proposes a honeywords method that combines deep learning and rule-based password attacks called GHDR, which can better resist Top-PW attacks. The GHDR method first constructs a neural network model to retrieve appropriate rules from the rule set used in the rule-based password attack for the user's passwords. Then the searched rules are used to transform the user's passwords to generate honeywords. The Flatness graph and Success-number graph are used to test the security of the GHDR method. The experimental results show that the GHDR method has a higher ability to resist Top-PW attacks than the Tweak-tail method and the PCFG-based method. In addition, we analyzed the time required to generate honeywords using the GHDR method, and the results proved that it is usable in practice.

Acknowledgements. This research was supported by the National Natural Science Foundation of China (Grant Nos. 62172433, 62172434, 61862011, 61872449), and Guangxi Natural Science Foundation (Grant No. 2018GXNSFAA138116).

References

1. Camenisch, J., Lehmann, A., Neven, G.: Optimal distributed password verification. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, pp. 182–194 (2015)
2. Catuogno, L., Castiglione, A., Palmieri, F.: A honeypot system with honeyword-driven fake interactive sessions. In: Proceedings of the International Conference on High Performance Computing & Simulation, pp. 187–194 (2015)
3. Chakraborty, N., Mondal, S.: Few notes towards making honeyword system more secure and usable. In: Proceedings of the 8th International Conference on Security of Information and Networks, pp. 237–245 (2015)
4. Chakraborty, N., Mondal, S.: A new storage optimized honeyword generation approach for enhancing security and usability. CoRR abs/1509.06094 (2015). http://arxiv.org/abs/1509.06094
5. Chakraborty, N., Mondal, S.: On designing a modified-UI based honeyword generation approach for overcoming the existing limitations. Comput. Secur. **66**, 155–168 (2017)
6. Chakraborty, N., Singh, S., Mondal, S.: On designing a questionnaire based honeyword generation approach for achieving flatness. In: Proceedings of the 17th IEEE International Conference On Trust, Security and Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering, pp. 444–455 (2018)
7. Chang, D., Goel, A., Mishra, S., Sanadhya, S.K.: Generation of secure and reliable honeywords, preventing false detection. IEEE Trans. Dependable Secur. Comput. **16**(5), 757–769 (2019)
8. Clair, L.S., et al.: Password exhaustion: predicting the end of password usefulness. In: Bagchi, A., Atluri, V. (eds.) ICISS 2006. LNCS, vol. 4332, pp. 37–55. Springer, Heidelberg (2006). https://doi.org/10.1007/11961635_3
9. Dauphin, Y.N., Fan, A., Auli, M., Grangier, D.: Language modeling with gated convolutional networks. CoRR abs/1612.08083 (2016). http://arxiv.org/abs/1612.08083

10. Dionysiou, A., Vassiliades, V., Athanasopoulos, E.: Honeygen: generating honeywords using representation learning. In: Proceedings of the ACM Asia Conference on Computer and Communications Security, pp. 265–279 (2021)
11. Erguler, I.: Achieving flatness: selecting the honeywords from existing user passwords. IEEE Trans. Dependable Secur. Comput. **13**(2), 284–295 (2016)
12. Fauzi, M.A., Yang, B., Martiri, E.: PassGAN based honeywords system for machine-generated passwords database. In: Proceedings of the IEEE 6th International Conference on Big Data Security on Cloud, IEEE International Conference on High Performance and Smart Computing, and IEEE International Conference on Intelligent Data and Security, pp. 214–220 (2020)
13. Gehring, J., Auli, M., Grangier, D., Yarats, D., Dauphin, Y.N.: Convolutional sequence to sequence learning. CoRR abs/1705.03122 (2017). http://arxiv.org/abs/1705.03122
14. Guo, Y., Zhang, Z., Guo, Y.: Superword: a honeyword system for achieving higher security goals. Comput. Secur. **103**, 101689 (2021)
15. Juels, A., Rivest, R.L.: Honeywords: making password-cracking detectable. In: Proceedings of the 20th ACM SIGSAC Conference on Computer and Communications Security, pp. 145–160 (2013)
16. Kuo, C., Romanosky, S., Cranor, L.F.: Human selection of mnemonic phrase-based passwords. In: Proceedings of the Second Symposium on Usable Privacy and Security, pp. 67–78 (2006)
17. Lin, T.Y., Goyal, P., Girshick, R., He, K., Dollar, P.: Focal loss for dense object detection. IEEE Trans. Pattern Anal. Mach. Intell. **42**(2), 318–327 (2020)
18. Melicher, W., et al.: Fast, lean, and accurate: modeling password Guessability using neural networks. In: Proceedings of the 25th USENIX Security Symposium, pp. 175–191 (2016)
19. Morris, R., Thompson, K.: Password security: a case history. Commun. ACM **22**(11), 594–597 (1979)
20. Pasquini, D., Cianfriglia, M., Ateniese, G., Bernaschi, M.: Reducing bias in modeling real-world password strength via deep learning and dynamic dictionaries. In: Proceedings of the 30th USENIX Security Symposium, pp. 821–838 (2020)
21. Wang, D., Cheng, H., Wang, P., Huang, X., Jian, G.: Zipf's law in passwords. IEEE Trans. Inf. Forensics Secur. **12**(11), 2776–2791 (2017)
22. Wang, D., Cheng, H., Wang, P., Yan, J., Huang, X.: A security analysis of honeywords. In: Proceedings of the 2018 Network and Distributed System Security Symposium (2018)
23. Wang, D., Wang, P., He, D., Tian, Y.: Birthday, name and bifacial-security: understanding passwords of Chinese web users. In: Proceedings of the 28th USENIX Security Symposium, pp. 1537–1555 (2019)
24. Weir, M., Aggarwal, S., de Medeiros, B., Glodek, B.: Password cracking using probabilistic context-free grammars. In: Proceedings of the 30th IEEE Symposium on Security and Privacy, pp. 391–405 (2009)