HoneyGAN: Creating Indistinguishable Honeywords with Improved Generative Adversarial Networks

Fangyi Yu(✉) and Miguel Vargas Martin(✉)

Ontario Tech University, Oshawa, ON L1G 0C5, Canada
{fangyi.yu,miguel.martin}@ontariotechu.ca

Abstract. Honeywords are fictitious passwords inserted into databases in order to identify password breaches. Producing honeywords that are difficult to distinguish from actual passwords automatically is a sophisticated task. We propose a honeyword generation technique (HGT) called HoneyGAN and an evaluation metric based on representation learning for measuring the indistinguishability of fake passwords, together with a novel attack model for evaluating the efficiency of HGTs. We compare HoneyGAN to state-of-the-art HGTs proposed in the literature using both evaluation metrics and a human study. Our findings indicate that HoneyGAN creates genuine-looking honeywords, leading to a low success rate for knowledgeable attackers in identifying them. We also demonstrate that our attack model is more capable of finding real passwords among sets of honeywords compared to previous works.

Keywords: authentication · machine learning · honeywords

1 Introduction

Current password-based authentication systems store sensitive password files that make them ideal targets for attackers because if successfully obtained and cracked, an adversary may impersonate registered users undetectable [11]. To effectively detect password leaks, Juels and Rivest [4] suggest that a website could store decoy passwords, called *honeywords*, alongside real passwords in its credential database, so that even if an attacker steals and reverts the password file containing the users' hashed passwords, they must still choose a real password from a set of k distinct *sweetwords*, where a real password and its associated honeywords are referred to as sweetwords. The attacker's use of a honeyword could cause the website to become aware of the breach. Notably, honeywords are only beneficial if they are difficult to distinguish from real-world passwords; otherwise, a knowledgeable attacker may be able to recognize them and compromise their security. Thus, when implementing this security feature into current authentication systems, the honeyword generating technique is critical.

The following are the paper's key contributions:

- We propose HoneyGAN, an HGT leveraging a password guessing model called GNPassGAN [14]. HoneyGAN can create passwords that seem legitimate and could be used in a honeyword system to deceive attackers.
- We introduce two evaluation metrics for determining the indistinguishability of honeywords and compare the honeywords generated by our technique HoneyGAN to those generated by other two state-of-the-art HGTs in the literature, and so could reliably infer our framework's true resistance to sophisticated discriminating attackers.
- We conducted a human study via Amazon Mechanical Turk to test the difficulty of finding the real passwords in sets of honeywords created by our HGT and other two state-of-the-art HGTs. Our findings are consistent with the result of using the two evaluation metrics we proposed. To the best of our knowledge, we are the first to conduct a research ethics-approved human participant study related to honeywords.
- To encourage more research on this area and to improve reproducibility, we have made the source code[1] for HoneyGAN publicly available.

The remainder of the paper is structured as follows: Sect. 2 introduces HoneyGAN, and two other HGTs for comparison. Section 3 is the HGTs evaluation. Section 4 is the user study and Sect. 5 discusses the limitations and future prospects of our study.

2 Honeyword Generation Techniques

2.1 HoneyGAN

GNPassGAN. Our HGT is inspired by a password guessing model GNPassGAN [14]. GNPassGAN is a GAN-based model that consists of a discriminator (D) and a generator (G) that are both constructed using deep learning neural networks. G takes as inputs noise or random features, learns the probability of the input's features, and creates data that follow the distribution of the input data. While D gets both real passwords and samples generated by G and makes every attempt to distinguish the two by calculating the conditional probability of a sample being false (or real) given a set of inputs (or features). This cat-and-mouse game forces D to extract vital information from the training data, and each iteration brings G's output closer to the distribution of real passwords, improving the possibility of matching the passwords of real-world users. GNPassGAN also incorporates gradient normalization to boost its guessing capability.

GNPassGAN is adept at generating realistic passwords, with 12.65% of passwords created by GNPassGAN being confirmed to exist in real-world password breaches (the *Rockyou* test set) [14], and the generated passwords that do not match the test set are plausible candidates for human-generated passwords. Because the primary challenge of honeyword creation is to develop indistinguishable decoy passwords that attackers cannot discern apart from genuine ones, theoretically, we reckon GNPassGAN can be employed for this purpose and demonstrate it quantitatively in Sect. 3 and 4 via experiments. The main difference

[1] https://github.com/fangyiyu/HoneyGAN.

between GNPassGAN and HoneyGAN is that GNPassGAN is used as a password guessing tool in our work, and HoneyGAN is a HGT that utilizes only the generator of GNPassGAN to generate honeyword candidates, and further select the passwords that are most similar to the real password as honeywords.

Text Similarity. Similarity between two strings is crucial in HGT since it demonstrates the indistinguishability of a false password from a genuine one, and is employed in both the honeyword creation and assessment processes. Typically, in natural language processing tasks, the distance/similarity of two strings is determined as follows: the strings are converted to vectors using word embedding techniques, and then the cosine similarity of the two vectors is calculated as the distance. Here, the strings might be composed of letters, symbols, or numbers, similar to how passwords are composed. Popular word embedding methods include Word2vec [6], FastText [1], and $TF-IDF$. While these techniques take into account the semantic and syntactic meanings of a word/text, in our case, the majority of passwords lack such meanings; hence, we choose the simplest but still effective method of vectorization known as bag of words (BoW).

In BoW, the core premise is that documents are similar if they contain comparable information. We examine the histogram of the characters included inside the strings, that is, each character count is considered as a feature. To be more precise, we first count the unique characters and their occurrences in the two strings being compared, then create a vector for each string with a length equal to the number of unique characters the strings contain, assign the vector's value in the associated index to the character's occurrences in each string, and finally compute the cosine similarity of the two vectors by definition. Please note that we do not consider the semantic connotations of passwords in this work.

Generate Honeywords with GNPassGAN. The following procedure demonstrates how we generate honeywords using GNPassGAN. (1) GNPassGAN first needs to be trained on a password corpus, and we train GNPassGAN for 200,000 iterations to get a thorough grasp of the construction pattern of passwords in the training dataset. (2) We use the GNPassGAN generator to produce a file F containing 50,000 fake passwords as honeyword candidates. Notably, F must be stored separately from the authentication system in a secure place. (3) We compare each user's true password to all fake passwords in F and calculate text similarity scores. Here, we convert each password to a vector using BoW and compute the cosine similarity of two passwords. (4) Finally, we assign honeywords for a real password to the $k-1$ most similar fake passwords in F.

2.2 Baseline Models

We utilize two models as comparisons in this work: chaffing-by-fasttext proposed by Dioysiou et al. [2] and chaffing-by-tweaking proposed by Juels and Rivest [4]. We will use the term chaffing-by-fasttext and *fasttext* interchangeably, as well as chaffing-by-tweaking and *tweaking*.

Chaffing-by-fasttext first trains the *fasttext* model with a real password corpus, then *fasttext* generates vector representations of each password in the corpus. After training is complete, the trained model can be queried by providing a real password as input and receiving a multi-dimensional vector representing the provided password's word embedding as a response. Finally, the top $k - 1$ closest neighbours according to cosine similarity are assigned as honeywords for each password.

Notably, the technique's primary weakness is that the produced honeywords are all genuine passwords in the *fasttext* training dataset, which means that if an attacker has access to the training dataset, the honeywords will be readily discovered. Additionally, the size of the training data has a significant impact on the quality of the honeywords created.

Chaffing-by-tweaking is an approach that mainly relies on random letter, digit, and symbol substitution.

Honeyword examples generated by the three HGTs can be found in Table 1.

Table 1. Honeyword samples generated by the three HGTs compared in the paper (HoneyGAN, *fasttext* and *tweaking*). Our password guessing model GNPassGAN and the *fasttext* model have been trained on a subset of the *Rockyou* dataset.

Real Passwords	**deshaun96**	**dafnny_24**	**Shauni16!**
HoneyGAN	masdane69	andey124	nahuas11
	sandesh89	badhyn24	hunhzan1
	naueds09	maydona242	hanilin1
fasttext	boedha21	snuffy22	muchluv!
	cutechica1	Dushido07	cliffordx
	felli1330	Dampire2	10.04.88
tweaking	DeShauN37	dafnny=96	Shauni53+
	deshaun87	dafNnY44	SHaunI73$
	DesHaun56	dAfnny+47	SHaUnI73$

3 Evaluation

3.1 Datasets

We analyze HoneyGAN's performance and compare it to the other two HGTs using 13 datasets containing real-world passwords. Our password datasets include over 828 million plain-text passwords and are derived from 13 different online providers (can be found in Table 2). We analyze these datasets and choose only passwords with a length of more than 8 characters, and we randomly choose 10,000 authentic passwords from each disclosed dataset to facilitate the assessment of the HGTs without sacrificing generality.

3.2 Internal Similarity Between Honeywords and Real Passwords

The primary goal of HGTs is to create indistinguishable fake passwords; that is, the honeywords and their corresponding actual passwords are too close to be differentiated. Consider passwords to be texts; we can determine the similarity of two passwords by comparing their text similarities. The greater the similarity score, the more similar the two passwords are, and the more difficult it is to distinguish them. We use the BoW metric to determine the similarity of two words without considering the semantic and syntactic meanings.

However, this metric is based on the assumption that an attacker attempts to differentiate real passwords using no resources. Indeed, they may have accessed a large number of previously compromised password files from data breaches. Because 40% of users reuse their passwords [7], more sophisticated attackers would assault the sweetwords using these accessible passwords. As a result, we develop an attack model as described in Sect. 3.3 and assess the resilience of the HGTs based on the aforementioned assumption of attackers. The performance of an HGT is then determined by combining these two evaluation metrics.

3.3 Attack Model: Normalized Top-SW

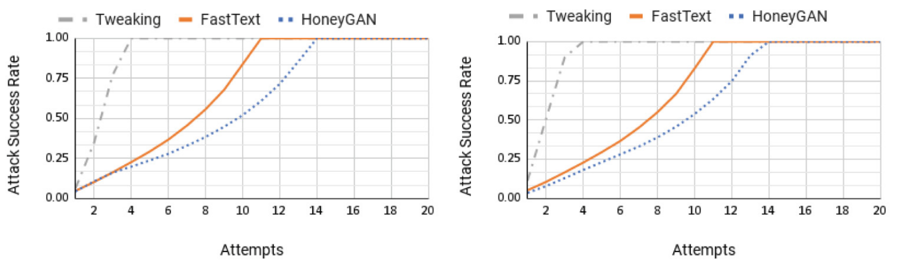
Our attack model, termed Normalized Top-SW is inspired by Wang et al.'s work "Normalized Top-PW" [11], and operates as follows: 1) Consider a genuine password dataset (attack) obtained from a data breach, and the sweetword file (target). The attacker employs the BoW to vectorize all passwords and sweetwords. 2) The attacker calculates the cosine similarity between each sweetword in the target file and all genuine passwords in the attack dataset, and then assigns the maximum similarity score to the sweetword denoting the highest likelihood of it being a true password. 3) The attacker tries the sweetwords of each user in decreasing order of their scores. If the guessed sweetword is a valid password for the associated user, then delete this user from the dataset; otherwise, set the similarity of the guessed sweetword to 0 to prevent it from being tried again.

In our experiment, we determine the efficiency of HGTs by computing the attacker's success rate under various attempts T. More precisely, we count the number of user accounts that are successfully cracked under varying T assignments and divided by the total number of users to get the attack success rate. We place all genuine passwords in the first column of the sweetword file for the simplicity of evaluation; in practice, operators should shuffle the order of sweetwords and securely keep the index of the real passwords.

3.4 Results

As recommended in [4], we assign $k = 19$ honeywords to each user and calculate the internal similarity score for each sweetword file generated by the three HGTs. Assume we are the *Rockyou* system operator and train our GNPassGAN and *fasttext* on our own dataset (*Rockyou*) to create honeywords for our users. We then attack the produced sweetword file using all other datasets in Table 2. For each user, the attacker has $T = 20$ attempts.

Average Internal Similarity. As a result, the internal similarity score for honeywords created by chaffing-by-GNPassGAN (HoneyGAN) is 0.8193, whereas chaffing-by-fasttext is 0.2620, and chaffing-by-tweaking is 0.6270. These numbers



(a) Attack Success Rate using all datasets except for *zynga*.

(b) Attack Success Rate using the *zynga* dataset.

Fig. 1. The Attack Success Rate by using the datasets in Table 2 (except for *Rockyou* as it is the target file) to attack the sweetword file generated by the three HGTs under the Normalized Top-SW attack. A line closer to the y-axis means the HGT is more vulnerable to attacks. As a result, honeywords generated by chaffing-by-tweaking are the easiest to attack, and by HoneyGAN are the hardest.

indicate that the honeywords created by HoneyGAN have the shortest average distance to their corresponding genuine passwords, implying that they are more similar to their true passwords and hence more difficult to differentiate.

Attack Success Rate (*ASR*). As illustrated in Fig. 1, under our Normalized Top-SW attack, when all datasets except *Rockyou* (exclude it since it is the target) are used as the attack dataset, we see the same pattern: we are able to crack all users' accounts in 4 attempts under the chaffing-by-tweaking condition, in 11 attempts under the chaffing-by-fasttext condition, and in 14 attempts under the HoneyGAN condition. Furthermore, 13 attempts are sufficient for the *zynga* dataset under the HoneyGAN condition. As a result, honeywords formed by *tweaking* are the simplest to discern, while those generated using HoneyGAN are the most difficult.

We show the average attack success rate ($AASR$) in Table 2, where $AASR = \frac{1}{20}\sum_{i=1}^{20} ASR^{(i)}$. As can be seen in the table, an attacker could achieve a success rate of around 60% when honeywords are created using HoneyGAN and 68% when honeywords are generated using *fasttext* when given 20 attempts per user, and it is statistically significant ($p = 3.09 * 10^{-12}$ for a one-tale t-test) that the attack success rate is lower when attacking honeywords generated by HoneyGAN than *fasttext*. Honeywords generated by *tweaking* is the most vulnerable with more than 90% attack success rate. Furthermore, HoneyGAN can produce better undetectable honeywords than *fasttext* and *tweaking* regardless of which dataset is used as the resource for attacking.

HoneyGAN outperforms *fastext* and *tweaking* in terms of both average internal similarity and attack success rate, indicating that HoneyGAN-generated honeywords are more similar to real passwords, therefore deceiving attackers and reducing their attack success rate, and alerting honeycheck towards the password breach.

Table 2. The Average Attack Success Rate on the three HGTs when various attack datasets with *Rockyou* as the target dataset are used. A number in bold indicates that the relevant HGT performs the best.

Dataset	**Tweak**	**FastText**	**HoneyGAN**
have-i-been-pwned-v2	0.9149	0.6863	**0.5923**
linkedin	0.9092	0.6863	**0.5943**
myspace	0.9279	0.6857	**0.6090**
youku	0.9072	0.6858	**0.6090**
zynga	0.9300	0.6907	**0.6213**
adultfriendfinder	0.9230	0.6902	**0.6006**
dubsmash	0.9229	0.6886	**0.6138**
last.fm (2016)	0.9226	0.6854	**0.5880**
chegg	0.9123	0.6888	**0.6032**
dropbox	0.9257	0.6928	**0.6096**
yahoo	0.9188	0.6881	**0.5868**
phpbb	0.9260	0.6855	**0.5972**

4 User Study

4.1 Study Design

We want to validate the hypothesis that individuals need more attempts to correctly find the real password when honeywords are generated by HoneyGAN than *tweaking* and *fasttext*.

We conducted a within-subjects experiment with 300 participants where each person performed all three HGTs. In our experiment, we have one independent variable: HGT type; three conditions: HoneyGAN, *tweaking* and *fasttext*; and one dependent variable: the number of attempts required to find the real password. Our study was approved by the Research Ethics Board at our institution.

Similar to previous security-related studies [3,5,8,10], we recruited participants through Amazon Mechanical Turk (AMT), where we embedded a survey designed on an online survey platform called Qualtrics. Qualified respondents were encouraged to complete our survey. We imposed three requirements on participants: (1) To avoid misunderstandings about our instructions, we need participants to be proficient in English; hence, we required participants exclusively from English-speaking countries including Canada, the US, the UK, and Australia. (2) Participants should have general knowledge as to what secure passwords look like, and we would expect that normally people savvy in information technology have such knowledge. So we only recruited those who self-identify as having a job related to information technology. (3) Additionally, we aim to include only individuals who accomplish high task quality on AMT, as measured by two AMT scores: the total number of approved Human Intelligence Tasks (HITs) and the percentage of approved HITs. We selected individuals who have 1,000 or more approved HITs and a 90% or greater approval rate for HITs.

Participants were required to answer 18 rank-order questions, which match 6 sets of honeyword samples produced from each of the three HGTs. Each question has 19 honeywords and 1 real password. The order of the 20 sweetwords is randomized. The participants were asked to sort the 20 sweetwords in each question according to their level of confidence that the sweetword is a real password. We compensated each participant with CAD$5.00 for completing the experiment, and the compensation was prorated using the Ontario minimum wage at the time of the study.

4.2 Results

Our analysis is based on the responses to our survey that each participant provided. We want to determine if there is a significant difference in the average number of attempts required for users to properly guess the real password in the HoneyGAN condition compared with the other two conditions.

Among all 300 responses, 7 responses were detected as robots by Qualtrics, and we deleted these suspicious responses. The remaining 293 responses took between 47 s and 211 min to complete. To ensure validity, we removed 13 of the 293 replies from participants who finished the exam in less than 3 min, as it is possible that they were not concentrating. Additionally, we eliminated outliers with completion time longer than 39 min and 30 s (boxplot maximum), leaving us with 272 responses to analyze.

The average completion time for the remaining 272 surveys was 14 min with 58 s, with a standard deviation of 7.86 min. This would suggest that the remaining participants were diligent in their responses.

We concatenated the responses for each HGT and got a dataset containing three columns (the three HGTs), and 1632 (6×272) rows, where each value represents the attempts needed to find the real password in one of the questions in the corresponding HGT. Since our experiment is a within-group study with non-uniform data, we used two-factor ANOVA without replication to examine the effect that the HGTs have on attempts needed to find the real password. The results indicated that the type of HGT resulted in statistically significant differences in the number of attempts required to find the real password ($F(2, 3262) = 448.276$, $p \leq 0.001$). We also ran two paired-samples t-tests to examine if there are significant differences between attempts required to find the real password for HoneyGAN vs *tweaking*, and HoneyGAN vs *fasttext*. As a result of comparing HoneyGAN and *fasttext*, the mean number of attempts required to find the real password is 12.479 in the HoneyGAN condition, meaning that participants require approximately 13 attempts to find the real password when HoneyGAN generates the honeywords, compared to 6.734 when *fasttext* generates the honeywords. And the result is statistically significant ($t(1631) = 29.767$, $p \leq 0.001$). A similar result can be found in the comparison of HoneyGAN vs *tweaking*: HoneyGAN requires 12.479 attempts while *tweaking* requires 8.89 ($t(1631)=16.948$, $p \leq 0.001$).

5 Discussion

In this section, we highlight the limitations and future work of our study.

Semantics in Passwords. One limitation of our study is that we did not consider the semantic meanings of passwords. This is flawed when authentication systems incorporate passphrases to assist users in memorization [3,9]. A passphrase, as opposed to a password, is typically a 4-to-10-word phrase, sentence, or statement having semantic and grammatical connotations.

Targeted Attacks. For targeted attacks, attackers exploit users' Personal Identifiable Information (PII) to guess passwords, which increases the likelihood of users' accounts being compromised. This is a critical problem because numerous PII and passwords become widely accessible as a result of ongoing data breaches, and people are used to create easy-to-remember passwords using their names, birthdays, and their variants [12]. Once an attacker obtains users' PII, and if only one sweetword in a user's sweetword list contains the user's PII, it is highly likely that this sweetword is the real password and others are fake.

To the best of our knowledge, Wang et al. [13] are the only ones that discuss how to generate honeywords that are resistant to targeted attacks. We are currently investigating how to generate honeywords for the same purpose with Natural Language Processing techniques.

6 Conclusions

In this paper, we propose HoneyGAN, an HGT built on top of GNPassGAN that generates high-quality honeywords capable of luring attackers and detecting password breaches. HoneyGAN can be easily integrated into any current password-based authentication system. Additionally, we present internal text similarity to assess the quality of honeywords and Normalized Top-SW, a honeyword attack model that mimics the real-world attack situation and avoids any ambiguity. We compare HoneyGAN's performance to two state-of-the-art HGTs using these two metrics, as well as a human study and discovered that HoneyGAN is capable of creating more hard-to-find honeywords and decreasing the success rate of sophisticated attackers. Furthermore, we demonstrated that our attack model Normalized Top-SW is more effective than Normalized Top-PW [11] in discovering real passwords.

Acknowledgement. The authors acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC), funding reference number RGPIN-2018-05919.

References

1. Bojanowski, P., Grave, E., Joulin, A., Mikolov, T.: Enriching word vectors with subword information. Trans. Assoc. Computat. Linguist. **5**, 135–146 (2017)

2. Dionysiou, A., Vassiliades, V., Athanasopoulos, E.: HoneyGen: generating honeywords using representation learning. In: Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security, pp. 265–279 (2021)
3. Jagadeesh, N., Martin, M.V.: Alice in passphraseland: assessing the memorability of familiar vocabularies for system-assigned passphrases. arXiv preprint arXiv:2112.03359 (2021)
4. Juels, A., Rivest, R.L.: Honeywords: making password-cracking detectable. In: Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security, pp. 145–160 (2013)
5. Kelley, P.G.: Conducting usable privacy & security studies with Amazon's mechanical turk. In: Symposium on Usable Privacy and Security (SOUPS), Redmond, WA (2010)
6. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781 (2013)
7. Pearman, S., et al.: Let's go in for a closer look: observing passwords in their natural habitat. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pp. 295–310 (2017)
8. Redmiles, E.M., Kross, S., Mazurek, M.L.: How well do my results generalize? Comparing security and privacy survey results from MTurk, web, and telephone samples. In: 2019 IEEE Symposium on Security and Privacy (S&P), pp. 1326–1343. IEEE (2019)
9. Shay, R., et al.: Correct horse battery staple: exploring the usability of system-assigned passphrases. In: Proceedings of the 8th Symposium on Usable Privacy and Security, pp. 1–20 (2012)
10. Tuncay, G.S., Qian, J., Gunter, C.A.: See no evil: phishing for permissions with false transparency. In: 29th USENIX Security Symposium (USENIX Security 2020), pp. 415–432 (2020)
11. Wang, D., Cheng, H., Wang, P., Yan, J., Huang, X.: A security analysis of honeywords. In: Network and Distributed System Security (NDSS) Symposium 2018, pp. 1–16 (2018)
12. Wang, D., Zhang, Z., Wang, P., Yan, J., Huang, X.: Targeted online password guessing: an underestimated threat. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 1242–1254 (2016)
13. Wang, D., Zou, Y., Dong, Q., Song, Y., Huang, X.: How to attack and generate honeywords. In: 2022 IEEE Symposium on Security and Privacy, pp. 489–506. IEEE (2022)
14. Yu, F., Martin, M.V.: GNPassGAN: improved generative adversarial networks for trawling offline password guessing. In: 2022 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), pp. 10–18 (2022). https://doi.org/10.1109/EuroSPW55150.2022.00009