





# Honey, I Chunked the Passwords: Generating Semantic Honeywords Resistant to Targeted Attacks Using Pre-trained Language Models

Fangyi Yu<sup>(✉)</sup>  and Miguel Vargas Martin<sup>(✉)</sup> 

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. The major challenge is producing honeywords that are difficult to distinguish from real passwords. Although the generation of honeywords has been widely investigated in the past, the majority of existing research assumes attackers have no knowledge of the users. These honeyword generating techniques (HGTs) may utterly fail if attackers exploit users' personal identifiable information (PII) and the real passwords include users' PII. The literature has demonstrated that password guessing is more effective when focusing on each of the chunks that compose a password (e.g., "P@ssword123" contains two chunks: "P@ssword" and "123") and it has been suggested that, when available, PII should be used to generate honeywords. We thus leverage these findings to base our HGT method on any possible PII contained within passwords, and introduce a new, and more robust than its literature counterparts, method to generate honeywords, which consists of generating honeywords with GPT-3 using the semantic chunks of their corresponding real passwords.

Furthermore, we propose a new metric, HWSimilarity, to evaluate the capability of HGTs. HWSimilarity is a pre-trained language model-based similarity metric that considers the semantic meaning of passwords when measuring the indistinguishability of honeywords and their counterparts. Comparing our chunk-level GPT-3 HGT to two state-of-the-art HGTs and using GPT-3 alone, we show that our HGT can generate honeywords that are more indistinguishable than its counterparts.

**Keywords:** authentication · chunking · honeywords · natural language processing · language models

## 1 Introduction

Passwords have dominated the authentication system for decades, despite their security flaws compared to competing techniques such as cognitive authentication [12], biometrics [20] and tokens [22]. Their irreplaceability is primarily due to their incomparable deployability and usability [3]. However, current password-based authentication systems store sensitive password files that make them ideal

targets for attackers because if successfully obtained and cracked (recovering the hashed passwords’ plain-text representations), an adversary may impersonate registered users in an undetectable fashion [26]. Numerous prestigious online services have been infiltrated, for example, Yahoo!, RockYou, Zynga, resulting in the exposure of millions of credentials. Unfortunately, there is often a large delay between a credential database’s breach and its detection; estimates place the average latency at 287 days [1]. The resulting window of vulnerability enables attackers to crack passwords offline and use them directly to extract value or sell them via illicit forums profiting with stolen credentials [25]. Normally, the longer it takes to detect and remediate a data breach, the more expensive it is [1]. As a result, it is vital to have active, timely password-breach detection systems in place to allow immediate counter-actions.

One way to reduce the cost of password breaches is to make offline guessing harder [5]. However, this method has major disadvantages, such as low scalability or a need for large modifications to the server-side and client-side authentication systems, which prevent the community from implementing them. Another promising approach is to shorten the latency between password breaches and detection. Juels and Rivest suggest the use of honeywords as a potential method for efficiently detecting password leaks [13]. According to their proposal, 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 process is critical.

### 1.1 Honeywords for Targeted Attacks

The biggest challenge of designing a HGT is to generate honeywords that are resistant to targeted attacks [28]. For targeted attacks, attackers exploit users’ 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 [1] and people are used to create easy-to-remember passwords using their names, birthdays, and their variants [28]. 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. For example, for a *sweetword* list “*gaby1124, abg71993, australiaisno#1, 10L026378, noviembre9101, Elena1986@327, cken22305*” which are generated using a made-up password “*Elena1986@327*” (suppose this is the real password) and the HGT proposed by Dionysiou et al. [9]. In this case, if the attacker has no information about the user, it will be difficult to determine which of the seven *sweetwords* is the real

password, since all of the honeywords are from data breaches and are legitimate passwords belonging to other users. However, if the attacker knows the user’s first name is “*Elena*”, it is quite straightforward to deduce that “*Elena1986@327*” is this user’s real password and all the others are fake.

**Table 1.** Data breaches containing PII and passwords in the past five years

Dataset	Number of Items	Year	Type of PII breached
Neiman Marcus	4,800,000	2021	Name, Encrypted Password, Security questions, Financial information
CAM4	10,880,000,000	2020	Name, Email, Encrypted Password, Chat transcripts, IP, Payment logs
Canva	137,000,000	2019	Name, Email, Encrypted Password
Quora	100,000,000	2018	Name, Email, Encrypted Password, Questions and answers posted
Yahoo	3,000,000,000	2017	Name, Email, Encrypted Password, DoB, Security question and answer

Following the introduction of the honeywords security mechanism by Juels and Rivest [13], the academic community has been actively exploring the technique. However, to our knowledge, only Wang et al. [29] concentrated on the production of honeywords in a targeted manner. All other works make the invalid assumption that attackers have no knowledge about the users. Each year, as demonstrated in Table 1, billions of password datasets including PII are leaked. Attackers might use the PII to determine which sweetword is the real password. If none of the sweetwords include PII existing in the password breach, the attackers may still create a knowledge map for each user by searching their information purposefully through social media and search engines using the known PII exposed in data breaches. This is especially a concern if the user is a public figure. Compromised accounts may have substantial financial, political, and societal consequences.

## 1.2 Related Work

Numerous studies have been conducted on the non-targeted honeyword generation method. The majority of these HGTs fall into two categories: chaffing-by-tweaking and chaffing-with-a-password-model. Chaffing-by-tweaking is mostly based on the substitution of random letters, digits, and symbols. For instance, given the real password “*deshaun96*”, we could get honeywords “*deshaun87*, *deshAUn66*, *DesHaun56*” via tweaking. However, as Wang et al. [26] demonstrate, this strategy is indeed vulnerable. While honeywords generated using the chaffing-with-a-password-model approach are more resistant to attacks, they do have certain drawbacks. Bojinov et al. [2] proposed *Kamouflage*, which first tokenizes the user’s real passwords into a collection of tokens, and then substitutes each token with a random one that matches the token’s type. For instance, “*jones34monkey*” is tokenized as “*l<sub>5</sub>d<sub>2</sub>l<sub>6</sub>*” (a five-letter word followed by two digits and a six-letter word), indicating that some possible honeywords are “*apple10laptop*, *tired93braces*, *hills28highly*”. This technique, as outlined in [9], demands considerable modifications on the client-side authentication system, which has a significant impact on usability. Additionally, it is incapable of

generating honeywords of varying length or structure, thus limiting the spectrum of possible honeywords.

Yu et al. [36] proposed to generate honeywords using a password-guessing model [34], which is based on an enhanced Generative Adversarial Network. They evaluated their HGT quantitatively and qualitatively, demonstrating that their HGT could generate honeywords more resistant to trawling attacks than other state-of-the-art HGTs.

For targeted honeyword generation, the challenge is to split the real password into tokens while retaining tokens that correspond to PII and replacing tokens that do not correspond to PII with random ones. Consider the real password ‘*Elena1986@327*’, the challenge is to produce honeywords containing the token ‘*Elena*’, which is the user’s first name as indicated by her email address. To do this, we propose to employ a chunking algorithm [32] to divide passwords into semantic chunks consisting of frequently occurring sequences of related characters, and a pre-trained generative model [4] to create desired honeywords based on the semantic chunks retrieved from the chunking step.

### 1.3 Our Contribution

- We are the first to use generative language models to create honeywords that are robust to targeted attacks. We propose a novel HGT, termed *Chunk-GPT3*<sup>1</sup> which generates honeywords by segmenting passwords into semantic chunks and then instructing GPT-3 to construct honeywords containing the given semantic chunks. Without being trained on real passwords, the off-the-shelf GPT-3 model could generate high-quality honeywords that are more indistinguishable from literature counterparts, and thus are more robust to targeted attacks. Furthermore, unlike HGTs from the literature, our model makes no assumptions as to the PII an attacker may use to tell apart honeywords from the real password.
- We are the first to take semantic meaning into consideration to evaluate HGTs. We propose *HWSimilarity*, for measuring an HGT’s capabilities. *HWSimilarity* employs a pre-trained language model MPNet [23] to encode sweetwords into vectors, and then calculates the cosine similarity between each honeyword vector and its real password vector, taking into consideration the semantics of each sweetword.
- We evaluated the capabilities of *Chunk-GPT3* and two state-of-the-art HGTs and demonstrated that *Chunk-GPT3*-generated honeywords are significantly more similar to their real passwords, making them more difficult to differentiate regardless of what PII is available in a targeted attack.

The remainder of the paper is structured as follows: Sect. 2 provides the preliminaries for understanding our work. Section 3 introduces our approach to generating honeywords in a targeted manner. Section 4 evaluates our HGT and other two approaches. Section 5 discusses the limitations of our work and future directions. Section 6 concludes our work.

<sup>1</sup> Source code: <https://github.com/HumanMachineLab/Chunk-GPT3>.

## 2 Preliminaries

In this section, we explain the honeyword generation mechanism and datasets used in this paper.

### 2.1 The Honeyword Mechanism

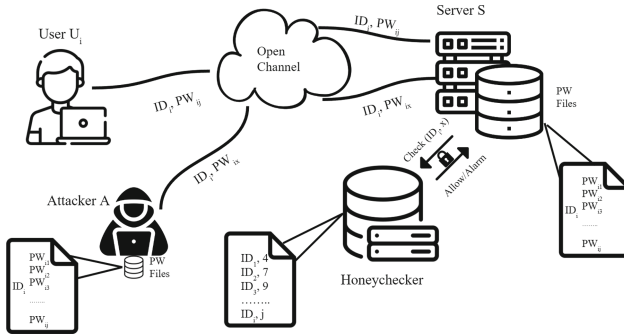


Fig. 1. Password (PW) authentication with honeywords.

Juels and Rivest [13] are the first to introduce the honeyword concept to detect password breaches. The honeyword system is comprised of four entities, as shown in Fig 1 [29]: a user  $U_i$ , an authentication server  $S$ , a *honeychecker*, and an attacker  $A$ . User  $U_i$  initially registers an account  $(ID_i, PW_i)$  on the server  $S$ . Apart from the standard user registration processes,  $S$  runs a command  $GEN(k, PW_i)$  to produce a list of  $k - 1$  unique fake passwords (called honeywords) to be stored alongside  $U_i$ 's true password  $PW_i$ , where  $k = 20$  as recommended in [13].  $PW_i$  and its  $k - 1$  honeywords are referred to as  $k$  sweetwords.

### 2.2 Threat Model

Honeyword-enabled systems could reliably identify a password file leak by pairing each user's account with  $k - 1$  honeywords. The reason for this is that even if attackers obtain a copy of the password file along with its hashing parameters and salts, and successfully recover all the passwords via brute-force or other password guessing techniques [17,30] (be aware that at this stage they know which  $k$  sweetwords are associated with each user), they must first distinguish each user's true password from these  $k$  sweetwords. The system features *honeychecker* to aid in the usage of honeywords, and the computer system could interact with the *honeychecker* whenever a login attempt is made or users change their passwords. Additionally, the *honeychecker* is capable of triggering an alert if an anomaly is

discovered. The warning signal may be sent to an administrator or to a third party [13]. This approach is compatible with existing authentication systems since it needs little adjustments to the server-side systems and no alterations to the client-side systems; nevertheless, it is very reliable due to the high probability of capturing adversaries. For instance, if the likelihood of an attacker selecting each sweetword is uniform, the probability of capturing an attacker is  $3/4 = 75\%$  for  $k = 4$ , and thus the probability grows as  $k$  increases.

Our HGT is designed based on the assumption that attackers have complete knowledge of users' PII, and our technique including the specifics (the following mentioned prompt and temperature). As described in Sect. 5, we ensure that our honeyword generation process is irreversible even when attackers have all of the aforementioned information.

### 2.3 Dataset

This section introduces the password dataset (termed 4iQ) we used in this paper and password selection process. 4iQ contains a leaked compilation of various password breaches over time and was first discovered in the Dark Web<sup>2</sup> in December 2017. The dataset consists of 1.4 billion email-password pairs, with 1.1 billion unique emails and 463 million unique passwords. Duplicate email-password pairs were removed by an unknown curator. The listed leaks are from websites such as Canva, Chegg, Dropbox, LinkedIn, Yahoo!, etc. We eliminated the suffix of each email address and only use the prefix as usernames for simplification.

To acquire legitimate passwords, we excluded those that are too short or too lengthy, with fewer than 8 characters or more than 32 characters, respectively [27], resulting in 28,492 username-password pairs. Such short strings are not permitted by most authentication systems [24], and such lengthy strings are unlikely created by users or password managers owing to their default settings of 12, 16 or 20 characters (LastPass, 1Password and Dashlane) [32]. We further calculated the strength of each password using *zxcvbn* [31], and found that 24,661 passwords have a *zxcvbn* score of 4, 2706 passwords have a *zxcvbn* score of 3, 277 and 3 passwords have a *zxcvbn* score of 1 and 0, respectively.

To compare HGTs' capability on various password strengths, we constructed two sets of username-password combinations depending on the computed *zxcvbn* password strength. One *zxcvbn-weak* set with 1000 username-password pairs whose passwords have the lowest *zxcvbn* score, and one *zxcvbn-strong* set with 1000 username-password pairings whose passwords have the highest strength *zxcvbn* score. Note that all passwords in the *zxcvbn-strong* set have a *zxcvbn* score of 4, and the *zxcvbn-weak* set has passwords with score ranging from 0 to 2. We further analyzed and compared the chunks in the two sets and generated honeywords for both sets with our proposed method and two other HGTs.

---

<sup>2</sup> 1.4 Billion Clear Text Credentials Discovered in a Single Database: <https://medium.com/4iqdelvedeep/1-4-billion-clear-text-credentials-discovered-in-a-single-database-3131d0a1ae14>.

### 3 Our Methodology

To preserve the PII in honeywords, it is necessary to segment passwords into chunks in which the PII is included. The chunks can then be used as inputs for a generative language model to produce honeywords that retain the PII while altering the real passwords.

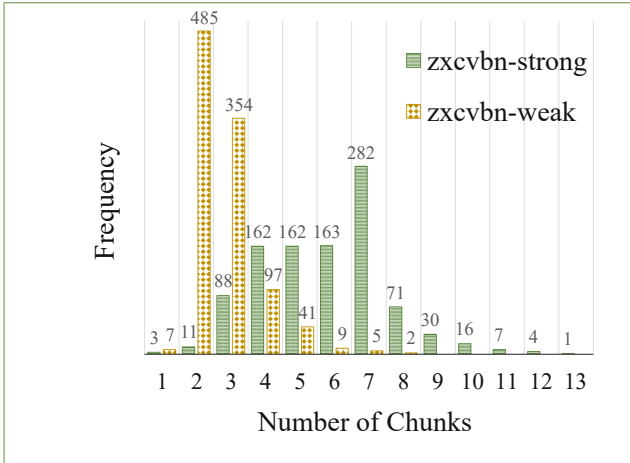
#### 3.1 PII Extraction

PII is rarely a single character. Instead, most PII, such as usernames, birthdays, anniversaries, and pet names carry some semantics. Semantic chunks in passwords may or may not constitute PII, but if users construct passwords including semantic chunks, they risk exposing PII. In order to extract PII from users' real passwords, we first segment the real passwords into semantic chunks using the password-specific segmentation technique PwdSegment [32]. PwdSegment conceptually trains a Byte-Pair-Encoding (BPE) for producing chunk vocabularies using training data of plain-text passwords. The BPE algorithm, which was initially proposed in 1994 as a data compression technique, is widely used in the NLP domain for subword segmentation (e.g., the GPT-2 model [18] proposed by OpenAI and the RoBERTa model [16] proposed by Meta), which preserves the frequent words while dividing the rare ones into multiple units. PwdSegment enhances the BPE technique by substituting the number of merging operations with the configurable parameter average length (*avg\_len*) of chunk vocabulary. PwdSegment counts all character pairs and terminates the merging operation when the *avg\_len* of the resultant chunk vocabulary equals or exceeds the threshold length. PwdSegment could be parameterized with a threshold *avg\_len* to control the segmentation result with varied granularity more simply where a longer *avg\_len* yields a more coarse-grained result.

The PwdSegment algorithm is first trained using a plain-text corpus. Then it repeatedly merges the most common pair of tokens into a single, new (i.e., previously unseen) token comprising the subword (i.e. chunk) vocabulary. Every merging procedure generates a new chunk by exchanging the most common pair of letters or character sequences (for example, “r”, “d”) with a new subword (for example, “rd”). The merging procedure is repeated until *avg\_len* of the resultant chunk vocabulary equals or exceeds a pre-determined threshold length.

#### 3.2 Chunk Analysis for zxcvbn-weak and zxcvbn-strong Password Sets

**Difference of Chunk Numbers.** We segment passwords into chunks for both zxcvbn-weak and zxcvbn-strong password sets using the PwdSegment algorithm. As shown in Fig 2, most passwords in the zxcvbn-strong set contain four to seven chunks, whereas most passwords in the zxcvbn-weak passwords only contain two or three chunks. This suggests that stronger passwords (based on zxcvbn) typically contain more chunks than weak passwords.



**Fig. 2.** The comparison of password chunk numbers in zxcvbn-weak and zxcvbn-strong sets.

**Difference of Common Chunk Frequencies.** To further investigate the differences between zxcvbn-strong password set and zxcvbn-weak password set, we list all chunks in both sets and visualize the result in Fig 3, from which we can observe that most chunks in the zxcvbn-weak password set contain semantics or easy-to-guess patterns, such as English words (“football”, “builder”, “vietnamese”, “microsoft”), phrases (“iloveyou”), Chinese names (“chenchen”, “liang”, “jiang”, “shan”), English names (“benjamin”, “Erick”, “sasha”, “elena”), and patterns (“qwert”, “zxcvbn”, “QWEASDZXC”). Many of them are plausible PII that attackers could take advantage of to compromise users’ accounts. In contrast, the majority of chunks in the zxcvbn-strong password set are random and short combinations of characters with no semantics, whereas semantics still exist in certain chunks (such as “sasha”, “jj” and “wang”). This indicates that although passwords that are zxcvbn-strong in strength are mostly comprised of more chunks and are harder to guess in a trawling scenario, many of them still contain semantic words, which can be PII that is accessible to attackers, thereby increasing the likelihood of passwords being guessed and accounts being compromised. As a result, regardless of the strength of the real password, as long as it contains PII which attackers could utilize all their resources to get, the trawling-honeyword-integrated authentication system will fail since most trawling-generated honeywords do not contain PII, and thus a targeted-honeyword-integrated system is needed.

### 3.3 Honeyword Generation with Chunk-GPT3

Language models can learn the probabilities of occurrences of a series of words in a regularly spoken language and predict the next potential word in that sequence. Generative Pre-trained Transformer 3 (GPT-3) is an autoregressive language





**Fig. 3.** The comparison of common password chunks in zxcvbn-weak (left) and zxcvbn-strong (right) sets.

model that uses deep learning to generate text that appears to be written by a person. It was introduced in 2020 and excels at a variety of NLP tasks, including translation, question-answering, and cloze [4]. The model was trained on trillions of words in text documents. It turns words into vectors or mathematical representations, and then decodes the encoded text into human-readable phrases. The model can be utilized to execute NLP tasks without requiring fine-tuning on particular downstream task datasets and is capable of producing texts that are difficult for humans to differentiate from human-written articles.

Therefore, we propose to use GPT-3 to generate honeywords that are robust to targeted attacks by providing the semantic chunks retrieved in the PII extraction phase. We first specify what the model should do by giving it a prompt, for example, “Derive five passwords that are similar to ‘*toby2009bjs*’ and contain ‘*toby*’, ‘2009’ and ‘*bjs*’. Do not add digits at the end of the passwords.” Here, “*toby*”, “2009” and “*bjs*” are chunks generated by PwdSegment. GPT-3 will then produce outputs “*tobyEmma2009bjs*, *toby2009Katiebjs*, *toby2009bjsKaitlyn*, *toby2009bjsRiley*, *toby2009bjsSavannah*” by following the instruction. The quality and the diversity of the output depend on three attributes: prompt, temperature and examples given to the model.

**The Prompt.** The prompt is the instruction GPT-3 received. The quality of the prompt can determine the quality of the generated honeywords. Usually, the more concise and instructive the prompt is, the better the completion is [15]. Same can be seen in honeyword generation, as shown in Table 2.

**Table 2.** Honeywords generated by GPT-3 when using different prompts. Honeywords generated using Prompt2 are not ideal because they do not contain the potential PII “toby”.

<b>Prompt1</b>	Suggest three passwords that are similar to “toby2009bjs” and contain “toby”
<b>Honeywords</b>	toby2009bjd, toby1998bjx, toby2021bjz
<b>Prompt2</b>	Suggest three passwords that look like “toby2009bjs”.
<b>Honeywords</b>	toy2009bjs, tab2009bjs, boy2009bjs

**The Temperature.** The temperature is a numeric variable between 0 and 1 that effectively regulates the model’s degree of confidence when generating predictions. A lower temperature implies that the model will take fewer risks, and the honeywords created will be more repetitive while increasing the temperature results in more diversified honeywords. The temperature is a vital parameter that determines the irreversibility of our HGT, as discussed in Sect. 5. Table 3 contains examples of honeywords formed at temperatures 0 and 1.

**Table 3.** Honeywords generated by GPT-3 when using different temperatures and given the prompt “Suggest five words that are similar to ‘toby2009bjs’ and contain ‘toby’.” A higher temperature will result in more diverse honeywords.

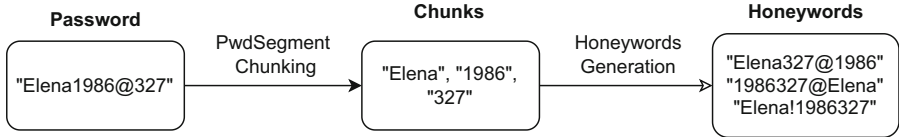
<b>Temperature</b>	<b>Honeywords</b>
0	toby2009bjd, toby2009bjx, toby2009bjz, toby2009bjf, toby2009bjh
1	Toby2009BJS, toby2009bjs1, tobybjs2009, Bjs2009toby, bjs2009toby1

**Zero-Shot and Few-Shot Learning.** Zero-shot learning refers to a situation in which no demonstrations are permitted and the model is given simply a plain language description of the task. In comparison, few-shot learning refers to a situation in which the model is given a few demonstrations of the task during inference time, but the model is not re-trained on them. This is particularly advantageous since many websites have varying policies regarding password creation, such as beginning with letters and requiring uppercase, lowercase, symbols, and numbers. When the operators demonstrate how they want the honeywords to appear, GPT-3 will generate honeywords that match the examples.

Since the introduction of Generative Pre-trained Transformers, they have been extensively investigated in a variety of domains, including creating media dialogues summaries [7], generating code from natural-language instructions [6], generating passphrases [11], and generating graphics from text descriptions [19]. To the best of our knowledge, we are the first to employ GPT-3 in the sphere of computer security, to generate honeywords that are resistant to targeted attacks.

An example of honeyword generation using Chunk-GPT3 is illustrated in Fig. 4, which contains two steps: 1). Passwords are segmented using algorithm PwdSegment, detailed in Sect. 3.1. For example, password “Elena1986@327”

is segmented into chunks “Elena”, “1986” and “327”. 2). The resulting chunks are used as inputs to prompt GPT-3 to generate honeywords. We prompted GPT-3 with instruction “Please derive three passwords that are similar to “Elena1986@327” and contain “Elena”, “1986” and “327”. The length of the passwords should be at most 13 (the length of the real password “Elena1986@327”).”



**Fig. 4.** Honeyword generation with Chunk-GPT3. In this example, the password “Elena1986@327” is segmented into chunks “Elena”, “1986”, and “327” using the PwdSegment Chunking algorithm. The chunks are then used as inputs for GPT-3 to generate honeywords.

## 4 Evaluation

Two common metrics in HGT evaluation are *flatness* and *success-number graphs* which measure HGTs’ resistance against the honeyword distinguishing attacker from the average and worst-case point perspective [26]. The honeyword distinguishing attacker is required for using the two metrics. Previous works [9,10] used the trawling attack algorithm Normalized Top-PW model to construct *flatness* and *success-number graphs* and to evaluate their HGTs, since their HGTs are used to generate honeywords against trawling attacks [35]. The Normalized Top-PW is not applicable to targeted attacks because trawling attackers have no knowledge about users’ PII while targeted attacks do, which make targeted attackers more capable. To the best of our knowledge, the only work proposing targeted attacks [29] construct their attack models based on various kinds of capabilities allowed to an attacker (e.g., birthday, username, email address, and registration order). We do not give these assumptions to attackers since it is typically not know what kind of attackers a system may have when generating honeywords. In fact, attackers may take advantage of any resources they may have, not limiting to PII, registration order and more. A comparison of the assumptions made in our HGT and in Wang et al.’s is shown in Table 4. Wang et al. [29] used flatness and success-number graphs to measure their HGTs. These metrics measure password guessing success rate per user, and the number of successfully identified real passwords, respectively. However, these honeyword evaluation metrics are not compatible with our method since flatness and success-number graphs require the computation of password probabilities as yielded by the HGT method. For example, if our honeywords were generated

using a PCFG-based approach, we would be able to compute honeyword probabilities. However, Chunk-GPT3 is not a probabilistic method but a generalized application of a large language model over password chunks. Thus, we evaluate honeywords from the perspective of word embedding similarities, which are commonly used in the NLP domain, to measure the similarity of two sequences. We propose an evaluation metric that measures the effectiveness of HGTs by comparing the similarity between a honeyword and its real password using another pre-trained language model. We also intend to draw the community’s attention to targeted scenarios, since trawling situations have been intensively studied but targeted honeyword generation and attack models are under-researched yet represent a pressing problem, as outlined in Sect. 1.1 and in [28].

**Table 4.** Assumptions on attackers in our HGT and Wang et al.’s [29].

	PW file	Public info <sup>a</sup>	Limited PII or user info	Any PII or other info
<b>Ours</b>	✓	✓	✓	✓
<b>Wang et al.’s [29]</b>	✓	✓	✓	

<sup>a</sup> Public info may include leaked password lists, password policy, and cryptographic algorithms.

#### 4.1 Metric: HWSimilarity

In this section, we introduce an evaluation metric to measure the indistinguishability of honeywords in terms of their corresponding real passwords.

The similarity between two strings is crucial in HGT since it demonstrates the indistinguishability of a false password from a genuine one. 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. Since passwords may contain PII which contains semantics, hence when measuring the similarity of two sweetwords, the semantics contained in a sweetword have to be considered. Therefore, in this paper, we propose to use a pre-trained language model MPNet [23] to encode passwords since it encodes the semantics in word sequences to word embeddings. MPNet utilizes the interdependence among predicted tokens via permuted language modeling (vs. MLM in BERT [8]) and accepts auxiliary position information as input to help the model view a whole phrase, hence minimizing position discrepancy (vs. PLM in XLNet [33]).

Computing the HWSimilarity of a sweetword list can be done as follows: For a user’s sweetword list  $SW = [sw_1, sw_2, \dots, sw_l]$ , and her honeyword list  $HW = [hw_1, hw_2, \dots, hw_{l-1}]$ , we have  $pw \in SW$ ,  $HW \subsetneq SW$  and  $pw \notin HW$ .

Here  $pw$  is the user’s real password,  $hw_i$  denotes a honeyword and  $l$  is the number of sweetwords.  $HWSimilarity = \frac{\sum_{i=1}^{l-1} \cosin(\Phi(hw_i), \Phi(pw))}{l-1}$ , here  $\Phi$  is the MPNet Neural Network model.

## 4.2 Comparable HGTs and Evaluation Results

We compare our Chunk-GPT3 with other three HGTs: generating honeywords using GPT-3 alone without semantic chunks provided, and two state-of-the-art HGTs chaffing-by-tweaking and *fasttext*.

**Chaffing-by-Tweaking.** Chaffing-by-tweaking (tweaking) HGT was initially presented in [13] and mainly relies on random letter, digit, and symbol substitution. We choose to use chaffing-by-tweaking instead of other recently proposed methods in the literature because other methods are more vulnerable to targeted attacks [2, 28]. Dionysiou et al. [9] highlight the intricacy of developing tweaking rules in such a way that it could be difficult for an attacker to distinguish the password from its changed versions. For example, if a chaffing-by-tweaking strategy randomly perturbs the last three characters of a password, the adversary may easily conclude that the authentic password is the first one in the instances “18!morning”, “18!morniey”, and “18!gorndge”. Thus, they replace all occurrences of a particular symbol in a given password with a randomly chosen alternate symbol, lower-case each letter in a password with probability  $p = 0.3$ , upper-case each letter in a password with probability  $f = 0.03$ , and replace each digit occurrence with probability  $q = 0.05$ . [9] contains the pseudocode and rationale for the assignment of  $p$ ,  $q$ , and  $f$ .

**Table 5.** Honeyword samples generated by the HGTs compared in the paper (Chunk-GPT3, GPT-3, *fasttext* and tweaking). *fasttext* is required to be trained on a real password dataset (the *rockyou* dataset in the paper). Other three HGTs can generate honeywords directly without being trained on a password dataset. Only Chunk-GPT3-generated honeywords retain the PII in the real password.

	HGTs			
	Chunk-GPT3	GPT-3	<i>fasttext</i>	tweaking
h2omega-tania	tania-home123	h2omega-alex	Karert.334	4oMega<tANia
	Tania@home5	h2omega-zoe	Adery993	H2oMega”tAnia
	home!tania12	h2omega-sam	brobe31	h4omega,tania
0000.mila.0000	1111.mila.0000	0000.lila.0000	octavia3	7434~MIIa\$6421
	0000_MILA_0000	0000_lela.0000	Bushido07	364\MIIA-9353
	0000@Mila@0000	0000_lola.0000	Dampire2	3124/MiLa‘2089
007skyblueboy	Skyblueboys007	007skybluegirl	gz152sha	903SkyBIUeboY
	Blueboysky007	007babyblueboy	Calepepi	561SkYblUEbOy
	007blueboysky	007lightblueboy	hajenrai	960SKybluebOy

**Chaffing-by-Fasttext.** This technique was proposed by Dionysiou et al. [9] which uses representation learning for the generation of honeywords. They convert words to vectors using *fasttext* and then assign honeywords to the  $k - 1$  nearest neighbors of an actual password based on cosine similarity.

More specifically, in the chaffing-by-fasttext method, it needs a real password corpus as the training dataset for the *fasttext* model. During the training phase, *fasttext* generates vector representations of each word 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. Following that, Dionysiou et al. loop over each password in their password corpus ( $n$  records in total where  $n$  is the number of users) and return its top  $k - 1$  closest neighbours in decreasing order of cosine similarity to create the list of  $k \times n$  sweetwords. In this way, for each password in the password file, they generate a list of the  $k - 1$  most similar honeywords.

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.

**GPT-3 Without Semantic Chunks.** We conducted an ablation study to assess if GPT-3 can create honeywords containing PII on its own, without any semantic chunks provided. In this case, the prompt we gave GPT-3 is “Derive 19 passwords that are similar to *real\_password*. The length of the passwords should be at most  $len(real\_password)$ . Do not add digits at the end of the passwords.”

A few examples of honeywords generated by Chunk-GPT3, GPT-3, tweaking and *fasttext* are illustrated in Table 5.

**Table 6.** HWSimilarity of honeywords generated by the four techniques (Chunk-GPT3, GPT-3, *fasttext*, and tweaking). Honeywords generated by Chunk-GPT3 have the highest HWSimilarity score compared with other HGTs, indicating that the Chunk-GPT3-generated honeywords are the most similar to their corresponding real passwords taking into account semantics.

	<b>Chunk-GPT3</b>	<b>GPT-3</b>	<b><i>fasttext</i></b>	<b>tweaking</b>
zxcvbn-strong	0.8525	0.8348	0.3441	0.7297
zxcvbn-weak	0.8367	0.8144	0.3445	0.7527

**Results.** The HWSimilarity of honeywords is shown in Table 6. For both zxcvbn-strong and zxcvbn-weak password sets, honeywords generated by *fasttext* and tweaking have a much lower HWSimilarity score than the score of honeywords generated by GPT-3 and Chunk-GPT3, indicating that the majority of *fasttext* and tweaking-generated honeywords do not contain users’ PII.

We also compared GPT-3 and Chunk-GPT3 using paired t-tests, and found the Chunk-GPT3-generated honeywords are significantly more similar to their corresponding real passwords considering semantics contained in passwords, and thus are harder to distinguish by targeted attacks ( $t_{zxcvbn-weak}(999) = 3.935, P < 0.001, t_{zxcvbn-strong}(999) = 3.237, P < 0.001$ ).

**Will HWSimilarity Leak Information About the Real Passwords to Attackers?** Consider this scenario: An attacker takes the 20 sweetwords and creates 20 different sets  $S_1, S_2, \dots, S_{20}$  of 19 sweetwords each (i.e., leaving a different sweetword out every time). Then for each of  $S_1, S_2, \dots, S_{20}$ , the attacker computes the HWSimilarity of each element of  $S_i$  against the sweetwords that are not in  $S_i$ . Will this expose some patterns revealing which of the 20 sweetwords is the real password? In order to examine this, we did a pilot experiment and took a subset of our data with 500 username-password pairs and 4 honeywords per user from the generated honeywords by the 4 HGTs. In this case, in the sweetword files with honeywords generated by different HGTs, each user has 4 honeywords stored along with the real passwords. For each sweetword list, the attacker takes one sweetword as  $p$ , and then 1) Computes the average HWSimilarity score ( $\bar{p}$ ) of each sweetword  $sw_1$  to  $sw_4$  against the target sweetword  $p$ . 2) Then computes the average HWSimilarity score ( $\bar{a}_1$ ) of sweetwords  $sw_2, sw_3, sw_4$ , and  $p$  against  $sw_1$ . 3) Next, computes the average HWSimilarity score ( $\bar{a}_2$ ) of sweetwords  $sw_1, sw_3, sw_4$ , and  $p$  against  $sw_2$ . 4) Then computes the average HWSimilarity score ( $\bar{a}_3$ ) of sweetwords  $sw_1, sw_2, sw_4$ , and  $p$  against  $sw_3$ . 5) Then computes the average HWSimilarity score ( $\bar{a}_4$ ) of sweetwords  $sw_1, sw_2, sw_3$ , and  $p$  against  $sw_4$ . 6) Finally, checks if one of the values (i.e.,  $\bar{p}, \bar{a}_1, \bar{a}_2, \bar{a}_3, \bar{a}_4$ ) is significantly “different” from the other 4 values.

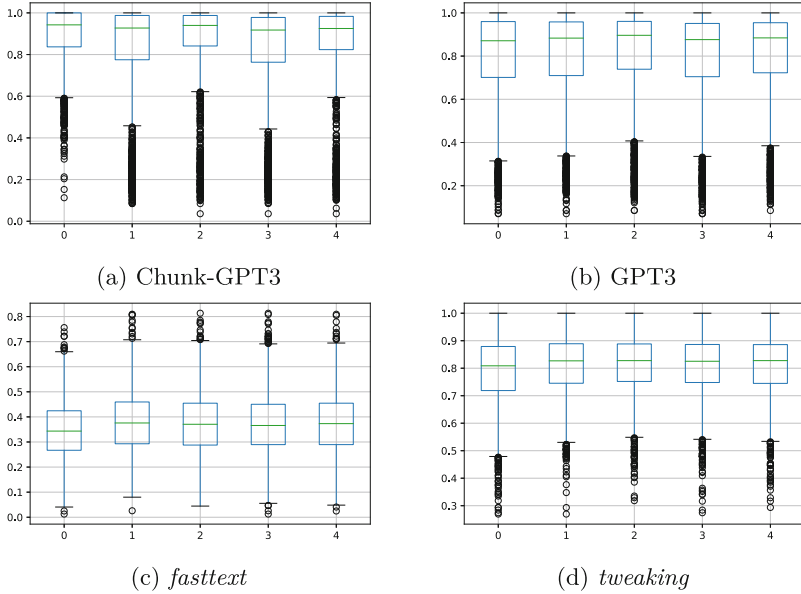
The average similarity scores for each HGT are shown in Fig 5. ANOVA tests on each HGT’s averages did not reject the null hypotheses, concluding that there is no significant difference between the averages, suggesting that HWSimilarity would not reveal the real password.

## 5 Discussion

We talk about the limitations of our study and future directions in this section.

**User Study.** We argue that there is no need to conduct user studies to qualitatively evaluate Chunk-GPT3-generated honeywords. If given a question: “Suppose you are an attacker and know a victim’s user name is ‘mila’, which one in the following list would most probably be his/her password: ‘0000\_mila\_0000, octavia3, Bushido07, Dampire2’ (real password and honeywords generated by *fasttext*).”, the task is easy to complete, while if the choices are “0000\_mila\_0000, 1111\_mila\_0000, 0000\_MILA\_0000, 0000@Mila@0000” (real password and honeywords generated by Chunk-GPT3), the task becomes obviously more difficult.

**Lack of Comparison with [29].** To the best of our knowledge, there is only one publication that discusses how to generate honeywords that are resistant



**Fig. 5.** Each boxplot represents the HWSimilarity scores of sweetwords at all indices with the sweetword at the target index. No significant difference in the average scores at different indices is observed for each HGT.

to targeted attacks, which was published in IEEE S&P’22 by Wang et al. [29]. They first proposed four attack models each representing a potential attacker  $A$ ’s strategy, with each model based on different information available to  $A$  (e.g., public datasets, the victim’s username, email address, birthday and registration order). They further developed four HGTs for each attack strategy, by using various probabilistic password guessing models proposed in previous work [28]. Nonetheless, assuming that attackers have access to only certain PII imposes an important limit since attackers may utilize a superset of PII beyond the PII pieces considered in their study (or a totally different PII set) to guess a user’s password, particularly if the user is a person of interest. What we are proposing is a different yet robust, and generalized approach. Rather than assuming  $A$ ’s attack strategy and creating HGTs accordingly, we assume attackers have white-box access to our HGT, meaning that attackers have complete knowledge of users’ PII, and our technique including the specifics (such as the prompt and temperature). A comparison of the assumptions made in our HGT and in Wang et al.’s can be found in Table 4, along with an explanation as to why generating flatness and success-number graphs is not possible (see Sect. 4). Despite the impossibility of producing a useful comparison between our work and Wang et al.’s, and for completeness, we still attempted to reproduce their HGT and compare with ours, using HWSimilarity. However, they did not make their artifacts public due to intellectual property concerns, and despite our efforts, we



were unable to reproduce their HGT from the description found in their paper. Nonetheless, it is clear that their HGT method does not consider the real password chunks, and this results in honeywords that do not necessarily resemble the real password. This can be readily seen by comparing the honeywords generated with one of their HGTs (TarList) and our Chunk-GPT3 method (see Table 7).

**Table 7.** Honeyword examples generated for real password “tiger81” by our method (Chunk-GPT3) and Wang et al.’s TarList (taken from [29], Fig. 1).

HGTs	Honeywords
<b>Chunk-GPT3</b>	Tig3r81, T1ger81, TigEr81, Tig3r8I, T1g3r81, Tig3r1I, T1gEr81, T1ger8I, TigEr1I, Tig3r8I, T1gEr8I, T1g3r1I
<b>Wang et al.’s</b>	jsmith117, prince00, love123, qwertyu, js128821, bond007, a123456, trustono1, rcv_11n1nj, jan1981, lemein, newy0rk, 1989y2002r

Since our HGT is based on the intuition that honeywords that are more similar to their corresponding real password are of higher quality [13], if there are any PII in the real passwords, the honeywords should include that PII to warrant their indistinguishability. Thus, we evaluate our HGT based on the similarity/word vector distance between honeywords and real passwords.

**Irreversibility.** The irreversibility of an HGT is critical. We need to make sure that even when attackers know our methodology and the specifications we were using for generating honeywords, such as the prompt and the temperature, they still cannot reproduce the honeywords we generated. This is ensured by careful prompt-engineering [14, 21] and temperature setting. We suggest to set temperature to 1 to get the most randomness [4], and after experimenting with various prompts, we decided to use the prompt “Derive 19 passwords that are similar to *real\_password*, and contain *chunks*. The length of the passwords should be at most  $len(real\_password)$ . Do not add digits at the end of the passwords.” since it generates the most diversified honeywords compared with other prompts we experimented with, and the honeywords generated each time are different by our observation.

## 6 Conclusions

In this paper, we proposed a novel HGT, Chunk-GPT3, which segments passwords into semantic chunks and then utilizes GPT-3 to generate high-quality honeywords that contain PII existing in users’ real passwords. Honeywords generated by Chunk-GPT3 are robust to targeted attacks where attackers get access to both breached password databases and users’ PII. Unlike other machine learning-based HGTs, GPT-3 can be easily integrated into any current password-based authentication system without any further training on real passwords. Additionally, we proposed a targeted HGT evaluation metric that incorporates

another pre-trained language model. We compared Chunk-GPT3's performance with GPT-3 alone, and two state-of-the-art HGTs with the proposed metric and demonstrated that Chunk-GPT3-generated honeywords are significantly harder to decipher and thus could raise the bar for targeted attacks.

**Acknowledgement.** The authors thank the assigned shepherd and anonymous reviewers for their valuable comments that improved the quality of the paper. We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC), funding reference number RGPIN-2018-05919.

## References

1. IBM security: Cost of a data breach report 2021 (2021). <https://www.ibm.com/security/data-breach>. Accessed 01 Jan 2022
2. Bojinov, H., Bursztein, E., Boyen, X., Boneh, D.: Kamouflage: loss-resistant password management. In: Gritzalis, D., Preneel, B., Theoharidou, M. (eds.) ESORICS 2010. LNCS, vol. 6345, pp. 286–302. Springer, Heidelberg (2010). [https://doi.org/10.1007/978-3-642-15497-3\\_18](https://doi.org/10.1007/978-3-642-15497-3_18)
3. Bonneau, J., Herley, C., Oorschot, P.C.V., Stajano, F.: The quest to replace passwords: a framework for comparative evaluation of web authentication schemes. In: 2012 IEEE Symposium on Security and Privacy (S&P), pp. 553–567 (2012). <https://doi.org/10.1109/SP.2012.44>
4. Brown, T., et al.: Language models are few-shot learners. In: Advances in Neural Information Processing Systems, vol. 33, pp. 1877–1901. Curran Associates, Inc. (2020). [https://proceedings.neurips.cc/paper\\_files/paper/2020/file/1457c0d6bfc4967418bfb8ac142f64a-Paper.pdf](https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfc4967418bfb8ac142f64a-Paper.pdf)
5. Camenisch, J., Lehmann, A., Neven, G.: Optimal distributed password verification. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. CCS '15, pp. 182–194. Association for Computing Machinery, New York, NY, USA (2015). <https://doi.org/10.1145/2810103.2813722>
6. Chen, M., et al.: Evaluating large language models trained on code. arXiv preprint [arXiv:2107.03374](https://arxiv.org/abs/2107.03374) (2021)
7. Chintagunta, B., Katariya, N., Amatriain, X., Kannan, A.: Medically aware GPT-3 as a data generator for medical dialogue summarization. In: Proceedings of the Second Workshop on Natural Language Processing for Medical Conversations, pp. 66–76. Association for Computational Linguistics, Online, June 2021). <https://doi.org/10.18653/v1/2021.nlpmc-1.9>
8. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: BERT: pre-training of deep bidirectional transformers for language understanding. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pp. 4171–4186. Association for Computational Linguistics, Minneapolis, Minnesota, June 2019. <https://doi.org/10.18653/v1/N19-1423>
9. 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. ASIA CCS '21, pp. 265–279. Association for Computing Machinery, New York, NY, USA (2021). <https://doi.org/10.1145/3433210.3453092>

10. Guo, Y., Zhang, Z., Guo, Y.: Superword: a honeyword system for achieving higher security goals. *Comput. Secur.* **103**, 101689 (2021). <https://doi.org/10.1016/j.cose.2019.101689>
11. Jagadeesh, N., Vargas Martin, M.: Alice in passphraseland: assessing the memorability of familiar vocabularies for system-assigned passphrases (2021). <https://doi.org/10.48550/ARXIV.2112.03359>
12. Joudaki, Z., Thorpe, J., Vargas Martin, M.: Reinforcing system-assigned passphrases through implicit learning. In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. CCS '18*, pp. 1533–1548. Association for Computing Machinery, New York, NY, USA (2018). <https://doi.org/10.1145/3243734.3243764>
13. Juels, A., Rivest, R.L.: Honeywords: making password-cracking detectable. In: *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security. CCS '13*, pp. 145–160. Association for Computing Machinery, New York, NY, USA (2013). <https://doi.org/10.1145/2508859.2516671>
14. Kojima, T., Gu, S.S., Reid, M., Matsuo, Y., Iwasawa, Y.: Large language models are zero-shot reasoners. *arXiv preprint arXiv:2205.11916* (2022)
15. Liu, P., Yuan, W., Fu, J., Jiang, Z., Hayashi, H., Neubig, G.: Pre-train, prompt, and predict: a systematic survey of prompting methods in natural language processing. *ACM Comput. Surv.* **55**(9) (2023). <https://doi.org/10.1145/3560815>
16. Liu, Y., et al.: RoBERTa: a robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692* (2019)
17. Pasquini, D., Gangwal, A., Ateniese, G., Bernaschi, M., Conti, M.: Improving password guessing via representation learning. In: *2021 IEEE Symposium on Security and Privacy (S&P)*, pp. 1382–1399 (2021). <https://doi.org/10.1109/SP40001.2021.00016>
18. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I.: Language models are unsupervised multitask learners (2018). <https://d4mucfpxsywv.cloudfront.net/better-language-models/language-models.pdf>
19. Ramesh, A., et al.: Zero-shot text-to-image generation. In: *International Conference on Machine Learning*, pp. 8821–8831. PMLR (2021)
20. Ratha, N.K., Connell, J.H., Bolle, R.M.: Enhancing security and privacy in biometrics-based authentication systems. *IBM Syst. J.* **40**(3), 614–634 (2001)
21. Reynolds, L., McDonell, K.: Prompt programming for large language models: beyond the few-shot paradigm. In: *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems. CHI EA '21*, Association for Computing Machinery, New York, NY, USA (2021). <https://doi.org/10.1145/3411763.3451760>
22. Roche, T., Lomné, V., Mutschler, C., Imbert, L.: A side journey to Titan. In: *30th USENIX Security Symposium (USENIX Security 21)*, pp. 231–248. USENIX Association, August 2021. <https://www.usenix.org/conference/usenixsecurity21/presentation/roche>
23. Song, K., Tan, X., Qin, T., Lu, J., Liu, T.Y.: MPNet: masked and permuted pre-training for language understanding. In: *Advances in Neural Information Processing Systems*, vol. 33, pp. 16857–16867. Curran Associates, Inc. (2020). [https://proceedings.neurips.cc/paper\\_files/paper/2020/file/c3a690be93aa602ee2dc0cab5b7b67e-Paper.pdf](https://proceedings.neurips.cc/paper_files/paper/2020/file/c3a690be93aa602ee2dc0cab5b7b67e-Paper.pdf)
24. Tan, J., Bauer, L., Christin, N., Cranor, L.F.: Practical recommendations for stronger, more usable passwords combining minimum-strength, minimum-length, and blocklist requirements. *CCS '20*, pp. 1407–1426. Association for Computing Machinery, New York, NY, USA (2020). <https://doi.org/10.1145/3372297.3417882>

25. Thomas, K., et al.: Data breaches, phishing, or malware? Understanding the risks of stolen credentials. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. CCS '17, pp. 1421–1434. Association for Computing Machinery, New York, NY, USA (2017). <https://doi.org/10.1145/3133956.3134067>
26. 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, October 2018. <https://doi.org/10.14722/ndss.2018.12345>
27. Wang, D., Wang, P., He, D., Tian, Y.: Birthday, name and bifacial-security: understanding passwords of Chinese web users. In: 28th USENIX Security Symposium (USENIX Security 19), pp. 1537–1555. USENIX Association, Santa Clara, CA, August 2019. <https://www.usenix.org/conference/usenixsecurity19/presentation/wang-ding>
28. 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. CCS '16, pp. 1242–1254. Association for Computing Machinery, New York, NY, USA (2016). <https://doi.org/10.1145/2976749.2978339>
29. Wang, D., Zou, Y., Dong, Q., Song, Y., Huang, X.: How to attack and generate honeywords. In: 2022 IEEE Symposium on Security and Privacy (S&P), pp. 966–983 (2022). <https://doi.org/10.1109/SP46214.2022.9833598>
30. Weir, M., Aggarwal, S., Medeiros, B.d., Glodek, B.: Password cracking using probabilistic context-free grammars. In: 2009 30th IEEE Symposium on Security and Privacy (S&P), pp. 391–405 (2009). <https://doi.org/10.1109/SP.2009.8>
31. Wheeler, D.L.: zxcvbn: Low-budget password strength estimation. In: 25th USENIX Security Symposium (USENIX Security 16), pp. 157–173. USENIX Association, Austin, TX, August 2016. <https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/wheeler>
32. Xu, M., Wang, C., Yu, J., Zhang, J., Zhang, K., Han, W.: Chunk-level password guessing: towards modeling refined password composition representations. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. CCS '21, pp. 5–20. Association for Computing Machinery, New York, NY, USA (2021). <https://doi.org/10.1145/3460120.3484743>
33. Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R.R., Le, Q.V.: XLNet: generalized autoregressive pretraining for language understanding. In: Advances in Neural Information Processing Systems, vol. 32. Curran Associates, Inc. (2019). [https://proceedings.neurips.cc/paper\\_files/paper/2019/file/dc6a7e655d7e5840e66733e9ee67cc69-Paper.pdf](https://proceedings.neurips.cc/paper_files/paper/2019/file/dc6a7e655d7e5840e66733e9ee67cc69-Paper.pdf)
34. Yu, F.: Raising the bar for password crackers: improving the quality of honeywords with deep neural networks. Master's thesis, Ontario Tech University, Oshawa, Canada (2022). [https://ir.library.ontariotechu.ca/bitstream/handle/10155/1593/Yu\\_Fangyi.pdf?sequence=1&isAllowed=y](https://ir.library.ontariotechu.ca/bitstream/handle/10155/1593/Yu_Fangyi.pdf?sequence=1&isAllowed=y)
35. Yu, F., Vargas Martin, M.: 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>
36. Yu, F., Vargas Martin, M.: HoneyGAN: creating indistinguishable honeywords with improved generative adversarial networks. In: Lenzini, G., Meng, W. (eds.) STM 2022. LNCS, vol. 13867, pp. 189–198. Springer, Cham (2023). [https://doi.org/10.1007/978-3-031-29504-1\\_11](https://doi.org/10.1007/978-3-031-29504-1_11)