Raising the Bar for Password Crackers: Improving the Quality of Honeywords with Deep Neural Networks

by

Fangyi Yu

A thesis submitted to the
School of Graduate and Postdoctoral Studies in partial
fulfillment of the requirements for the degree of

Master of Science in Computer Science

Faculty of Science

University of Ontario Institute of Technology (Ontario Tech University)

Oshawa, Ontario, Canada

December 2022

© Fangyi Yu, 2022

THESIS EXAMINATION INFORMATION

Submitted by: **Fangyi Yu**

Master of Science in **Computer Science**

Thesis title: Raising the Bar for Password Crackers: Improving the Quality of Honeywords with Deep Neural Networks

An oral defense of this thesis took place on November 25th, 2022 in front of the following examining committee:

Examining Committee:

Chair of Examining Committee	Pooria Madani Assistant Professor, Faculty of Business and Information Technology
Research Supervisor	Miguel Vargas Martin Professor, Faculty of Business and Information Technology
Examining Committee Member	Julie Thorpe Associate Professor, Faculty of Business and Information Technology
Thesis Examiner	Akramul Azim Associate Professor, Faculty of Engineering and Applied Science

The above committee determined that the thesis is acceptable in form and content and that a satisfactory knowledge of the field covered by the thesis was demonstrated by the candidate during an oral examination. A signed copy of the Certificate of Approval is available from the School of Graduate and Postdoctoral Studies.

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 time-consuming and sophisticated task, and the majority of existing research assumes that attackers have no knowledge about users, which is a flawed assumption. In this thesis, we introduce two honeyword generation techniques (HGT): Honey-GAN and Chunk-GPT3, which can generate honeywords resistant to trawling attacks and targeted attacks, respectively. In addition, we propose a trawling attack, termed as Normalized Top-SW, to imitate trawling attackers and further assess the resilience of HGTs to the attack. Furthermore, we propose two text similarity-based metrics to evaluate the indistinguishability of honeywords. We analyze our HGTs compared with the other two state-of-the-art HGTs quantitatively and qualitatively and demonstrate that our HGTs can produce honeywords that are substantially more difficult for attackers to distinguish, hence increasing the bar for attackers and accelerating the detection of passwor‘d breaches.

Keywords: generative adversarial networks; honeywords; authentication; security and privacy; natural language processing

Author’s Declaration

I hereby declare that this thesis consists of original work of which I have authored. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners. I authorize the University of Ontario Institute of Technology (Ontario Tech University) to lend this thesis to other institutions or individuals for the purpose of scholarly research. I further authorize the University of Ontario Institute of Technology (Ontario Tech University) to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research. I understand that my thesis will be made electronically available to the public.

The research work in this thesis was performed in compliance with the regulations of the Research Ethics Board under REB Certificate number #16779.

______________________________ Fangyi Yu

Statement of Contributions

The research presented in this thesis is funded by the Natural Sciences and Engineering Research Council of Canada (NSERC), funding reference number RGPIN-2018-05919.

The work described in Chapter 3 has been published as:
F. Yu and M. V. Martin, "GNPassGAN: Improved Generative Adversarial Networks For Trawling Offline Password Guessing," *2022 IEEE European Symposium on Security and Privacy Workshops* (EuroS&PW), 2022, pp. 10-18, doi: 10.1109/EuroSPW55150.2022.00009.
I presented the work virtually at the EuroS&PW'22 conference hosted in Genoa, Italy (June 6-10, 2022).

The work described in Chapter 4 has been published in the 18th International Workshop on Security and Trust Management, co-located with the 27th European Symposium on Research in Computer Security (ESORICS).
I presented the work virtually at the STM'22 workshop hosted in Copenhagen, Denmark (September 26-30, 2022).
I hereby certify that I am the sole author of this thesis. I have used standard referencing practices outlined in the *Publication Manual of the Association for Computing Machinery (2021)* to acknowledge ideas, research techniques, or other materials that belong to others. Furthermore, I hereby certify that I am the sole source of the creative works and/or inventive knowledge described in this thesis.

Acknowledgements

I would like to convey my sincere thanks to Dr. Miguel Vargas Martin, my thesis adviser, and research supervisor, for his ongoing guidance, encouragement, support, patience, and extensive knowledge. Thank you not only for supporting the development of my thesis and providing me with complete academic freedom but also for sharing your insight and experience in the formation of my future professional path. I feel blessed and honored to work and learn under his supervision.

I would also like to express my gratitude to Dr. Julie Thorpe and Dr. Akramul Azim for serving on my thesis committee and as the external examiner, respectively. You can never have too many eyes on a work like this, and your input was vital in making this thesis as solid as possible.

Last but not least, my deepest appreciation goes to my spouse Shengqian for his patience and unwavering support, my parents for their unconditional love, and all of my friends. Without them, it would not have been feasible to complete this task.

Table of Contents

List of Figures

List of Tables

List of Abbreviations and Symbols

GANs Generative Adversarial Networks

PII Personally Identifiable Information

PCFG Probabilistic Context-Free Grammar

RNN Recurrent Neural Networks

LSTM Long Short Term Memory Networks

CNN Convolutional Neural Networks

PSMs Password Strength Meters

WAEs Wasserstein Autencoders

DPG Dynamic Password Guessing

BERT Bidirectional Encoder Representations from Transformers

BiLSTM Bidirectional Long Short-term Memory

FLA Fast, Lean, Accurate

IWGAN Improved Training of Wasserstein GANs

SPRNN Structure Partition and BiLSTM Recurrent Neural Network

HGT Honeyword Generation Technique

GPT-3 Generative Pre-trained Transformer 3

Chapter 1

Introduction

Passwords have dominated the authentication system for decades, despite their security flaws compared to competing techniques such as cognitive authentication [76], biometrics [57] and tokens [60]. Their irreplaceability is primarily due to its incomparable deployability and usability [8]. 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 undetectable [71]. Numerous prestigious online services have been infiltrated, for example, Yahoo![1], RockYou[2], Zynga[3], 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 [2]. The resulting window of vulnerability enables attackers to crack passwords offline (if the stolen credential database contains encrypted passwords rather than plain-text

[1] Yahoo Triples Estimate of Breached Accounts to 3 Billion (2017), https://www.wsj.com/articles/yahoo-triples-estimate-of-breached-accounts-to-3-billion-1507062804

[2] RockYou2021: largest password compilation of all time leaked online with 8.4 billion entries (2022), https://cybernews.com/security/rockyou2021-alltime-largest-password-compilation-leaked/

[3] Password Breach of Game Developer Zynga Compromises 170 Million Accounts (2019), https://www.cpomagazine.com/cyber-security/password-breach-of-game-developer-zynga-compromises-170-million-accounts/

passwords); determine the value of these passwords by probing their associated accounts, and then use them directly to extract value or sell them via illicit forums dealing in stolen credentials [66]. Normally, the longer it takes to detect and remediate a data breach, the more expensive it is. More precisely, data breaches that take more than 200 days to identify and contain cost an average of 4.87 million, compared to 3.61 million for breaches that take fewer than 200 days to identify and manage [2]. 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. A variety of ways have been proposed in the literature, including machine-dependent functions [5], external password-hardening services [38], and distributed cryptography [11]. All of these approaches, however, have 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 [34]. 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* (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.

1.1 Motivation

One challenge of designing an HGT is that 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. Additionally, any suggested HGT should assure its irreversibility, meaning that it should be computationally inefficient (or impossible) to revert to the original password file containing just the true password for each user from the sweetword file [17].

Another challenge is to generate honeywords that are resistant to targeted attacks. For targeted attacks, attackers exploit users' Personally 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 [1, 2], and people are used to creating easy-to-remember passwords using their names, birthdays, and their variants [73]. Once an attacker obtains user's PII, and if only one sweetword in a user's sweetword list contains the user's PII, it is easy to deduce that the only sweetword containing the user's PII is the real password and others are all fake.

To address these challenges, we propose two novel techniques to generate honeywords that are resistant to offline trawling attacks, and targeted attacks, respectively.

1.2 Contributions

Our contributions are as follows:

- We introduce **GNPassGAN**, an offline password guessing tool, which outperforms standalone password guessing methods in the literature on both one-site (models trained and tested on subsets of the same dataset) and cross-site (models trained and tested on various datasets) scenarios. It can serve as (a) a new state-of-the-art benchmark for academics interested in the password guessing area, and can further be utilized to (b) develop password strength meters that encourage users to choose stronger passwords, and (c) to produce honeywords that detect password breaches [34].

- Because GNPassGAN is capable of producing texts with the same character distribution as the training data, and when trained on a real-world password dataset, it can produce authentic-looking passwords. We then introduce **HoneyGAN**, a strategy for generating honeywords that leverages GNPassGAN.

- We introduce two **evaluation metrics** for determining the indistinguishability of honeywords in a trawling scenario 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 about our framework's true resistance to sophisticated attackers.

- We highlight issues with password settings when deep learning algorithms are used to guess passwords. More specifically, previous research mainly focused on guessing passwords less than or equal to 8 characters, which are considered short passwords and would be rejected by websites with a minimum length password policy. This enables future research to focus only on password settings that adhere to password creation policies.

- We propose a novel HGT, termed **Chunk-GPT3**, which generates honeywords by segmenting passwords into semantic chunks and then instructing GPT-3 to construct honeywords containing the given chunks. Without being trained on real passwords, the off-the-shelf GPT-3 model could generate high-quality honeywords that are more resistant than literature counterparts to targeted attacks. To the best of our knowledge, we are the first to use language models to generate honeywords that are robust to targeted attacks.

- We are the first to take semantics into consideration to evaluate HGTs. We propose **HWSimilarity**, for measuring an HGT's capabilities under targeted attacks. HWSimilarity employs a pre-trained language model MPNet 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 passwords are significantly similar to their real passwords, making them more difficult to differentiate even when PII is available to targeted attackers.

- We have made the source code[4] available to the public to facilitate reproducibility.

1.3 Thesis Organization

The remainder of the thesis is structured as follows: Chapter 2 summarizes related work in password guessing models and honeyword generating techniques. Chapter 3 introduces our password guessing model GNPassGAN. Chapter 4 introduces HoneyGAN, our trawling HGT. Chapter 5 introduces Chunk-GPT3, our targeted HGT. Each chapter contains the proposed framework's methodology, evaluations, and discussions. Chapter 6 is the thesis conclusion, followed by the bibliography and appendices.

[4] https://github.com/fangyiyu/GNPassGAN
https://github.com/fangyiyu/HoneyGAN
https://github.com/fangyiyu/Honeyword_GPT3.

Chapter 2

Related Work

2.1 Overview

Despite the prevalence of text-based passwords, the security of user-selected passwords continues to be a significant concern. According to research examining susceptible behaviors that affect password crackability [25], there are three types of user actions that result in the creation of insecure passwords: (1) Users often use basic terms in passwords and perform simple string transformations to comply with websites' password creation policies [47]. (2) Password reuse is prevalent. According to S. Pearman et al. [51], 40% of users reuse their passwords across multiple platforms. (3) Users prefer to use simple-to-remember passwords that include personal information such as their birth date or their pets' name. All of these behaviors expose the user-created passwords to attacks. Additionally, the recent large-scale leakage of passwords on multiple platforms across the world (listed in Table 2.1) raises the alarm for researchers and stakeholders.

As a consequence, it becomes even more critical to aid users in establishing stronger passwords. Due to the fact that password strength is a statistic that reflects a password's resistance to guessing attacks, the first step is to appropriately estimate password strength via password guessing attacks. Password guessing strategies can be categorized as offline and online, or targeted and trawling.

2.1.1 Offline Attacks and Online Attacks

Password guessing attacks fall into two categories: offline and online. Offline attacks occur when attackers get cryptographic hashes of certain users' passwords and attempt to recover them by guessing and testing many passwords. The primary objective is to determine the difficulty of cracking a genuine user's password, or the strength of a user-created password, by producing a list of password guesses and checking for the possibility of the genuine user's password's occurrence. Offline attacks are only considered when the following conditions are met: An attack gains access to the system and extracts the password file, all while remaining unnoticed. Moreover, the file's salting and hashing must be done appropriately. Otherwise, an offline assault is either ineffective (the attacker may get credentials directly without requiring guesses, or an online approach is more effective), or impossible [22].

An online attack occurs when an attacker makes password attempts against users using a web interface or an application. This situation is more constrained for attackers since

Table 2.1: Datasets used for training and evaluating deep learning models. Size is the number of passwords in the dataset.

Name	Size	Brief Description
Yahoo!	4.4×10^5	A web services provider.
phpBB	3×10^5	A software website.
RockYou	1.4×10^7	A gaming platform.
MySpace	5.5×10^4	A social networking platform.
SkullSecurityComp	6.7×10^6	Compilations of passwords lists.
LinkedIn	1.3×10^6	A social online platform.

most authentication systems automatically freeze accounts after several unsuccessful attempts. Therefore, attackers must guess users' passwords successfully within the allotted number of tries, which is the primary difficulty of online password guessing. According to Florêncio et al. [22], 10^6 is a reasonable upper limit for the number of online guesses a secure password must survive, while the number of offline guesses is difficult to quantify considering the attacker's possible usage of unlimited computers each calculating hashes thousands of times quicker than the target site's backend server.

2.1.2 Trawling Attacks and Targeted Attacks

Targeted guessing attacks occur when attackers attempt to break users' passwords using their knowledge of users, specifically their PII, such as name, birth date, anniversary, home address, etc. This is a considerable concern when PII becomes more accessible as a result of constant data breaches. On the contrary, trawling attacks do not assume the users' identities.

In this thesis, we propose two HGTs that can generate honeywords resistant to trawling attacks and targeted attacks, respectively.

2.2 A Survey of Deep Learning in Password Guessing

The three predominant ways of password guessing are rule-based, probability-based, and deep learning-based.

2.2.1 Rule-based Models

A large amount of stolen passwords simplifies the process of collecting password patterns. Following that, other candidate passwords may be produced using these password patterns as guidelines. Hashcat[1] and John the Ripper[2] are two popular open-source password guessing programs that use rule-based password guessing. They provide a variety of ways for cracking passwords, including dictionary attacks, brute-force attacks, and rule-based attacks. Among all these types, the rule-based one is the fastest, and Hashcat is the market leader in terms of speed, hash function compatibility, updates, and community support [31]. However, rule-based systems create passwords solely based on pre-existing rules, and developing rules requires domain expertise. Once rules are defined, passwords that violate those restrictions will not be identified.

2.2.2 Probability-based Models

Apart from rule-based password guessing models, conventional password guessing models are mostly probability-based, with two notable approaches being Markov Models and Probabilistic Context-Free Grammar (PCFG). Markov Models are built on the assumption that all critical password features can be specified in n-grams. Its central principle is to predict the next character based on the preceding characters [47]. PCFG examines the grammatical structures (combinations of special characters, digits, and alphanumerical sequences) in disclosed passwords and generates the distribution probability, after which it uses the distribution probability to produce password candidates [77].

Veras et al. [69] proposed a framework, termed as semantic guesser, that first employs natural language processing techniques to segment, categorize, and generalize semantic categories from passwords, then incorporate the semantic segments into the PCFG model to guess passwords. In the first 3 billion attempts, their semantic guesser was able to guess 67% more LinkedIn passwords and 32% more MySpace passwords than the PCFG technique alone.

[1] https://hashcat.net/wiki/

[2] https://www.openwall.com/john/

2.2.3 Deep Learning-based Models

Unlike rule-based or probability-based password guessing tools, deep learning-based methods make no assumptions about password structure. Deep neural network-generated password samples are not constrained to a particular subset of the password space. Rather than that, neural networks can autonomously encode a broad range of password information beyond the capabilities of human-generated rules and Markovian password-generating methods.

Recurrent Neural Networks

Recurrent Neural Networks (RNN) are neural networks in which inputs are processed sequentially and restored using internal memory. They are often employed to solve sequential tasks such as language translation, natural language processing, and voice recognition. Due to the fact that the vanilla RNN architecture is incapable of processing long-term dependencies due to the vanishing gradient issue [49], therefore, Long Short Term Memory Networks (LSTM) was designed to tackle the problem. LSTM networks make use of a gating mechanism to retain information in memory for long periods of time [30].

To the best of our knowledge, Melicher et al. [44] were the first to utilize RNN to extract and predict password features. They kept their model, named Fast, Lean, Accurate (FLA), as lightweight as possible in order to integrate it into local browsers for proactive password verification. Three LSTM layers and two highly connected layers comprise the proposed Neural Network. Various strategies for training neural networks on passwords were used. It was proven that employing transfer learning [83] significantly improves guessing efficacy, however, adding natural language dictionaries to the training set and tutoring had little impact. Consequently, they discovered that Neural Networks are superior at guessing passwords when the number of guesses is increased and when more complicated or longer password policies are targeted. Nevertheless, because of the Markovian nature of FLA's password generation process, any password feature that is not included within the scope of an n-gram may be omitted from encoding [29].

Zhang et al. [85] presented Structure Partition and BiLSTM Recurrent Neural Network (SPRNN), a hybrid password attack technique based on structural partitioning and Bidirectional Long Short-term Memory (BiLSTM). The PCFG is used for structure partitioning, which seeks to structure the password training set to learn users' habits of password construction and generate a collection of basic structures and string dictionaries ordered by likelihood. The BiLSTM was then trained using the string dictionary produced by PCFG. They compared SPRNN's performance to probability-based approaches (Markov Models and PCFG) on both cross-site (model trained and tested on various datasets) and one-site (model trained and tested on subsets of the same dataset) scenarios. SPRNN outperforms the other two models in all circumstances, albeit it performs worse cross-site than one-site.

Based on Zhang et al.'s work [85], the same year, Liu et al. [41] also developed a hybrid model named GENPass that can be generalized to cross-sites attacks. The model preprocesses a password by encoding it into a series of units that are then given tags based on PCFG (e.g., "password123" can be separated into two units: "L8" and "D3", where "L" refers to letters, and "D" refers to digits.). After that, LSTM is used to create passwords. Additionally, they built a Convolutional Neural Networks (CNN) classifier to determine which wordlist the password is most likely to originate from. The results indicate that GENPass can achieve the same degree of security as the LSTM model alone in a one-site test while generating passwords with a substantially lower rank. GENPass enhanced the matching rate by 16 to 30% when compared to LSTM alone in the cross-site test.

Autoencoders

Autoencoders are any model architecture that is composed of two submodules: an encoder and a decoder. The encoder is responsible for learning the representation of the source text at each time step and generating a latent representation of the whole source sentence, which the decoder uses as an input to build a meaningful output of the original phrase. Typically, autoencoders are employed to deal with sequential data and various NLP tasks, such as machine translation, text summarization, and question answering. RNN and CNN are often used as encoder and decoder components, respectively.

Pasquini et al. [79] applied this strategy to a dataset containing leaked passwords, using Generative Adversarial Networks (GANs) and Wasserstein Autoencoders (WAEs) to develop a suitable representation of the observed password distribution rather than directly predicting it. Their methodology, called Dynamic Password Guessing (DPG), can guess passwords that are unique to the password set. and they are the first to apply completely unsupervised representation learning to the area of password guessing.

Attention-based models

When we use the phrase “Attention” in English, we mean concentrating our focus on something and paying closer attention. The Attention mechanism in Deep Learning is based on this principle, and it prioritizes certain tokens (words, letters, and phrases) while processing text inputs. This, intuitively, aids the model in gaining a better knowledge of the textual structure (e.g., grammar, semantic meaning, word structure, and so on) and hence improve text classification, generation, and interpretability. In language models, attention mechanisms are often utilized in combination with RNN and CNN. However, even with LSTM, these models cannot manage lengthy dependencies since transforming the whole source sentence to a fixed-length context vector is challenging when the source sentence is too long. As a result, Transformers [68] were invented that were built just on Attention, without convolution or recurrent layers. Bidirectional Encoder Representations from Transformers (BERT) [16], ELMO [53], and GPT [54] are all well-known instances of attention-based applications built on top of Transformers.

Li et al. [39] proposed a curated Deep Neural Network architecture consisting of five LSTM layers and an output layer, and then tutored and improved the created model using BERT. They proved that the tutoring process by BERT can help increase the model performance significantly.

GANs

Unlike the previously described deep learning-based algorithms commonly employed in Natural Language Processing tasks, GANs [24] have been used to construct simulations of pictures, texts, and voices across all domains. Behind the scenes, GANs consist of two sub-modules: a discriminator (D) and a generator (G), both of which are built of deep learning neural networks. G accepts noise or random features as input; learns the probability of the input's features; and generates fake data that follows the distribution of the input data. While D makes every effort to discriminate between actual samples and those created artificially by G by estimating the conditional probability of an example being false (or real) given a set of inputs (or features). The model architecture diagram is illustrated in Figure 2.1. This cat-and-mouse game compels D to extract necessary information in training data; this information assists G in precisely replicating the original data distribution. D and G compete against one another during the training phase, which progressively improves their performance with each iteration. Typically, proper gradient descent and regularization techniques must be used to accelerate the whole process. More formally, the optimization problem solved by GANs can be summarized as follows:

$$\min_{\theta_G}\max_{\theta_D}\sum_{i=1}^{n} log f(x_i;\theta_D) + \sum_{i=1}^{n} log(1 - f(g(z_j;\theta_G);\theta_D))$$

where $f(x_i;\theta_D)$ and $g(z_j;\theta_G)$ represents the discriminator D and the generator G respectively.

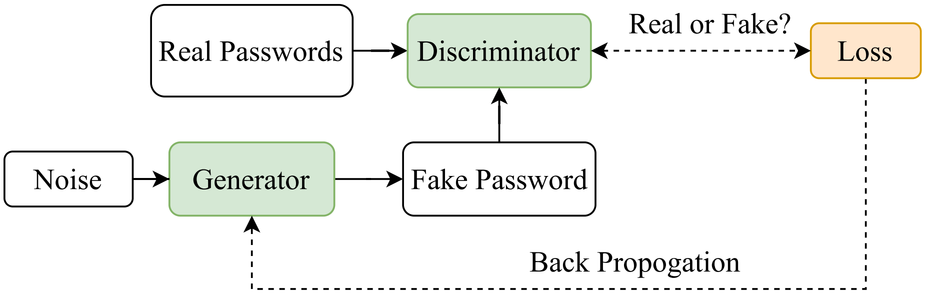


Figure 2.1: GAN's model architecture.

The optimization demonstrates the min-max game between D and G. After the initial GANs work was published in 2014, several enhancements were made, and Hitaj et al. [29] leveraged the Improved Training of Wasserstein GANs (IWGAN) [26] to apply GANs on password guessing, which is the first in literature. They trained the discriminator using a collection of leaked passwords (actual samples). Each iteration brings the generator's output closer to the distribution of genuine passwords, increasing the likelihood of matching real-world users' passwords. Consequently, PassGAN outperformed current rule-based password guessing tools and state-of-the-art machine learning password guessing technologies (FLA) after sufficient passwords were generated (10^9). They matched 51% - 73% of passwords when combining PassGAN with Hashcat, compared to 17.67% when using Hashcat alone and 21% when using PassGAN alone. One disadvantage of PassGAN is that it has intrinsic training instability due to the final softmax activation function in the generator, which may easily result in the network being vulnerable to vanishing gradients, lowering the guessing accuracy.

Following the publication of PassGAN in 2019, other researchers saw the possibilities of using GANs for password guessing, and more refinements have been done on top of PassGAN. In 2020, Nam et al. [46] developed REDPACK that employs a variant of GANs in conjunction with various password generation models for improved cracking performance. They suggested rPassGAN in their prior study, which enhanced PassGAN by altering its fundamental Neural Network architecture. More precisely, they employed RNN in PassGAN instead of ResNet in PassGAN's original paper. However, during rPassGAN's training process, it became unstable at times, and REDPACK introduced the RaSGGAN-GP cost function to stabilize the training process. Nam et al. also introduced a selection phase to REDPACK, during which the password candidates are generated using several password generators (Hashcat, PCFG, and rPassGAN). The discriminator then determines the chance of each generator's password candidates being realistic and sends the candidates with the greatest probability to password cracking tools such as Hashcat. We regard PassGAN to be a good representation of GANs-based password guessing tools, and PassGAN enhancement is the inspiration for our proposed password guessing model.

A comparison of prior published deep learning-based password guessing tools is illustrated in Table2.2.

Table 2.2: A comparison for Deep Learning Models used for Password Guessing.

Category	**Methods**	**Models used**	**Year**
Autoencoders	DPG [79]	WAE, GANs	2021
GANs	REDPACK [46]	IWGAN, RaGAN, HashCat, PCFG	2020
GANs	PassGAN [29]	IWGAN	2019
Attention	Language Model [39]	BERT, LSTM	2019
RNN	GENPass [41]	PCFG, LSTM, CNN	2018
RNN	SPRNN [85]	PCFG, BiLSTM	2018
RNN	FLA [44]	LSTM	2016

2.3 A Survey of Honeyword Generation Techniques

To the best of our knowledge, Bojinov et al. [7] were the first to propose using honeywords for theft-resistant password managers. Their architecture *Kamouflage* produces a fixed number of fake managers with accompanying decoy master passwords and keeps them alongside the real password manager. The first phase of their HGT is tokenization, in which the password manager transforms 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 "$l_5d_2l_6$" (a five-letter word followed by two digits and a six-letter word), indicating that some possible honeywords are "$apple10laptop$", "$tired93braces$", and "$hills28highly$". This technique, as outlined in [17], demands considerable modifications to the client-side authentication system, which has a significant impact on usability. Additionally, it is incapable of generating honeywords of varying lengths or structures, thus limiting the spectrum of possible honeywords. In comparison, our technique needs minimum modifications to the server-side authentication system and also supports the generation of honeywords of varying lengths and structures.

Erguler [20] proposed a different technique in which honeywords are derived from the system's current user passwords. In this case, all honeywords are realistic and adhere to the operator's password-creating policy. However, their HGT is restricted by the limited number of viable honeywords created by selecting genuine passwords from the website's password corpus, which is particularly the case if the website has a small user database. On the other hand, since the generator of GNPassGAN can produce as many passwords as needed, our HGT can thus generate a significantly larger amount of honeywords than selecting real passwords from other corpus.

In contrast to our HGT, which utilizes GNPassGAN for honeyword generation, Fauzi et al. [21] directly generated honeywords using PassGAN. The advantage of our model over Fauzi et al.'s work is twofold: 1) As proven in Section 3, our password guessing model GNPassGAN can better learn the distribution of genuine passwords and create passwords that match the distribution of real passwords. 2) Fauzi et al.'s approach needs that they train their HGT on the attack dataset; future attackers with access to the stolen passwords dataset may significantly enhance their attacking performance. In comparison, we recommend to website administrators that they train our HGT on their password corpus in order to create system-specific honeywords and prevent any security consequences. Finally, Fauzi et al.'s technique should be assessed using more advanced attack that simulates sophisticated real-world attackers (such as Normalized Top-SW) in order to determine whether or not their HGT generates susceptible honeywords for readily distinguishable user accounts.

To the best of our knowledge, there is only one publication that discusses how to generate honeywords that are resistant to targeted attacks by Wang et al. [74]. 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 personal information and registration order). They further develop four HGTs for each attack strategy, by using various representative probabilistic password guessing models proposed in their previous paper [73]. These assumptions about attackers are flawed since the attackers may utilize whatever information they can get to attack users' accounts, particularly if the user is a person of interest. What we are proposing is a much simpler yet robust, and generalized approach. Rather than assuming *A*'s attack strategy and creating HGTs accordingly, we construct honeywords based on the information contained in the real password. The challenge is to partition 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 proposed by Xu et al. [81] to divide passwords into chunks consisting of frequently occurring sequences of related characters, and a language model [10] capable of generating high-quality honeywords incorporating PII.

Chapter 3

GNPassGAN

3.1 Methodology

As mentioned in Section 2.2.3, the IWGAN used in PassGAN implements gradient penalty to impose the 1-Lipschitz continuous of the discriminator; however, Wu et al. [80] proved that to achieve a balance between the Lipschitz restriction required by earth mover distance and the neural network's capacity is a difficult challenge; the combination of gradient penalty and earth mover distance is not the ideal solution to GAN's mode collapse and vanishing gradient issues. To address these issues, our approach adopted a different kind of normalization technique: gradient normalization, as introduced in [80]. Gradient normalization imposes a gradient norm restriction on the GANs discriminator to increase its capacity. Wu et al. [80] proved that GANs trained with gradient normalization outperform previous GANs in the computer vision area by conducting comprehensive experiments. In our study, we expect that in the password guessing domain, we can also outperform previous methods when generating passwords with the same rank by applying gradient normalization to PassGAN.

Fig. 3.1 illustrates the architecture of our model, termed as GNPassGAN. The key

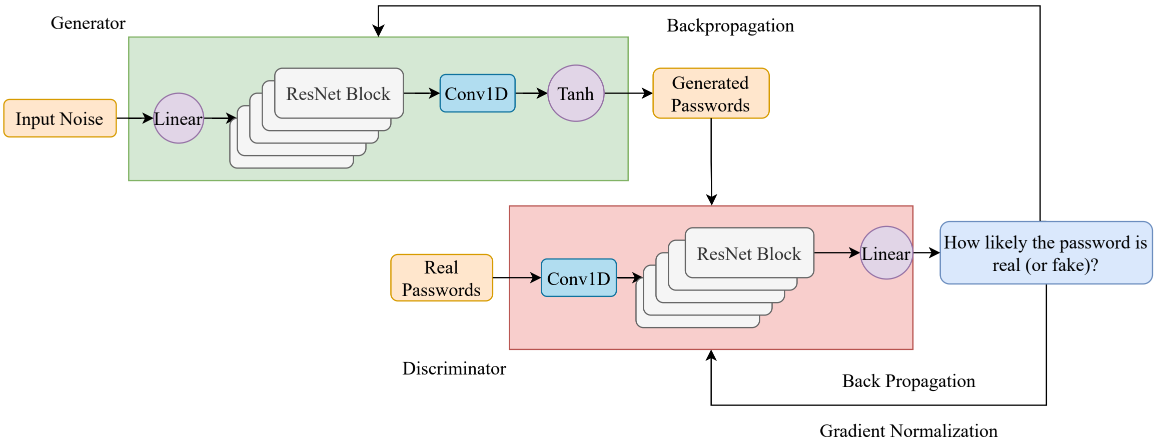


Figure 3.1: GNPassGAN model architecture diagram.

improvements we made to PassGAN are as follows: We add gradient normalization [80] to the discriminator, and the generator's activation function in the last layer is modified from softmax to tanh. Additionally, instead of the Wasserstein loss, we use the binary entropy loss inside a sigmoid layer as the loss function.

3.2 Evaluation

Our model GNPassGAN was implemented using PyTorch 1.10. Our experiments were conducted on a workstation running Ubuntu 20.04.0 LTS, with 30 GB of RAM, an Intel(R) Xeon(R) Silver 4114 CPU, and an NVIDIA Tesla P100 GPU with 16 GB Global Memory. The hyperparameter settings for running GNPassGAN can be found in Table 3.1.

3.2.1 Experimental Design

We only compare our model with PassGAN in this paper since PassGAN has conducted extensive experiments and shown that their work exceeds traditional rule-based and probability-based password guessing tools. As with PassGAN's work, we use the *rockyou* dataset[1] for

[1] http://downloads.skullsecurity.org/passwords/rockyou.txt.bz2

Table 3.1: Hyperparameter Setting for Trainning GNPassGAN

Hyperparameters	**Value**
Batch size	64
Number of iterations	200,000
Number of discriminator iterations for each generator iteration	10
Layer dimension for generator and discriminator	128
Adam learning rate	0.0001
Adam coefficient β_1	0.5
Adam coefficient β_2	0.9

training, and the *phpbb* dataset[2] and a disjoint subset of *rockyou* for testing. The distribution of the two datasets based on length can be found in Table 3.2. Testing on two different sets with varying data distributions allows us to assess if our model generalizes well to cross-site password guessing. We conducted experiments on passwords of two lengths: less than or equal to 10 characters, as most password guessing experiments do; between 8 and 12 characters inclusively, which corresponds to the real-world password setting scenario as most websites require passwords to be at least 8 characters. We refer to these two experimental settings as *Char10* and *Char812*, respectively. To better assess the models' guessing capability, we delete duplicates in the datasets. The training and testing sets are randomly divided by a ratio of 4:1, and there is no overlap between the two sets. Hitaj et al. [29] tested their PassGAN model on passwords with less than 10 characters only; by testing on *Char812*, we can see if the models are capable of properly guessing more complicated passwords. Both PassGAN and GNPassGAN are trained for 200,000 iterations in our experiment, with checkpoints for D and G retained every 10,000 iterations for the purpose of storing the neural network parameters.

Notably, the justification for using GANs to guess passwords is based on the assump-

[2] https://github.com/danielmiessler/SecLists/blob/master/Passwords/Leaked-Databases/phpbb.txt

Table 3.2: The comparison of the data distribution based on length in the *rockyou* and *phpbb* dataset. Around 95% of passwords in *phpbb* are less than or equal to 10 characters.

Range	**RockYou**	**phpBB**
(0,8)	33.025%	48.039%
[8,10]	50.004%	46.864%
(10,12]	9.906%	4.031%
(12,∞)	7.065%	1.066%
Total	100%	100%

tion that the training and testing sets have a similar distribution, and therefore, by simulating samples from the training set, the generated samples may approximate the test set sufficiently. By shuffling before splitting into the two sets, we assume that they have a similar distribution. In Section 3.2.2, we demonstrate empirically that our hypothesis is correct.

3.2.2 Experimental Results

Measuring Guessing Accuracy. GANs are typically applied in the computer vision area. The Inception Score [61] and the Frechet Inception Distance [28] are the most frequently used metrics for evaluating GAN's performance. However, since we are measuring GANs in the context of password guessing, the metrics employed in computer vision are inappropriate for our task. Instead of that, we measure the performance of GANs using the *matching accuracy* as most previous works did. This metric indicates the percentage of actual passwords generated by the algorithms on an unseen dataset (test set).

Assume that the generated file is *FG* and the testing file is *FT*. We define the matching accuracy by dividing the number of unique passwords that exist in both *FG* and *FT*, by the total number of unique passwords in *FT*. The following formula can be used to represent the calculation:

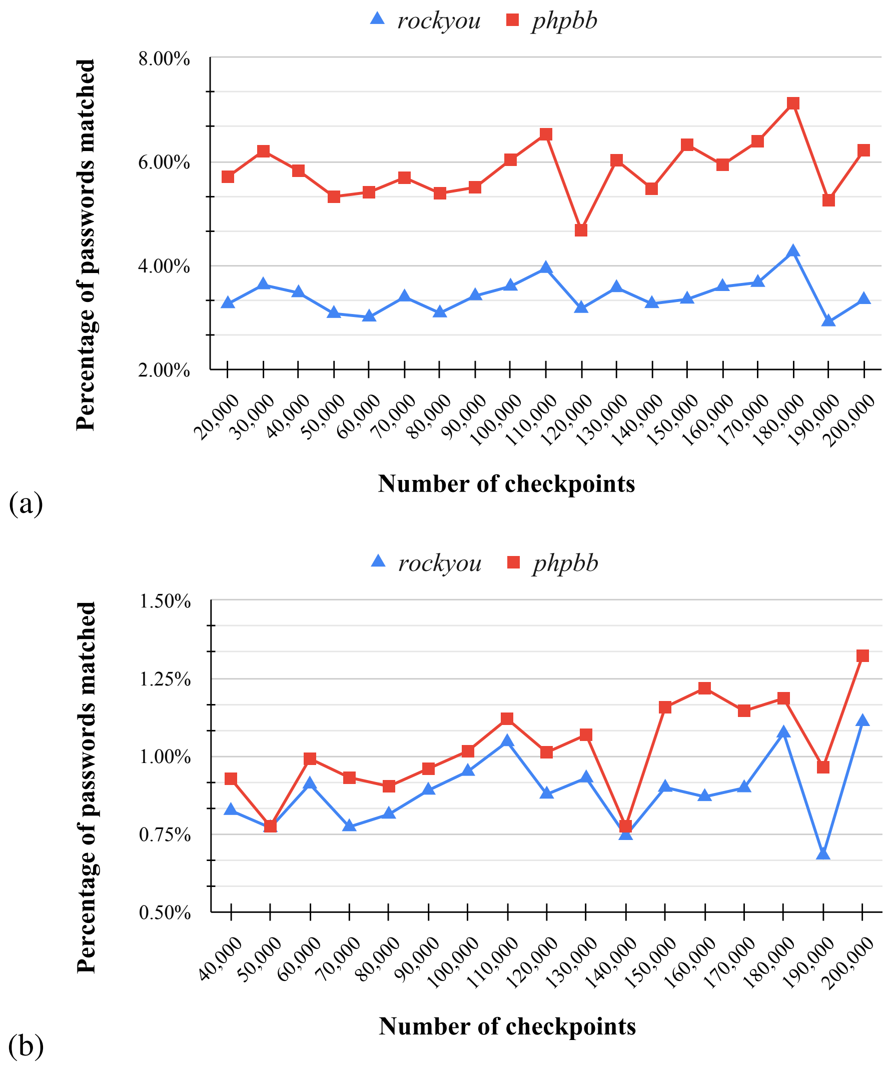


Figure 3.2: Proportion of unique passwords created by GNPassGAN that matched the *rockyou* testing set and the *phpbb* testing set at different checkpoints when experimented on passwords with length $\leqq$ 10 characters (a) and in [8, 12] (b). The *x* axis denotes the checkpoint used in the generating process. For each checkpoint, we sampled 10^7 passwords.

Table 3.3: The comparison of the data distribution based on length in the training set, the *rockyou* testing set and files containing fake passwords generated by PassGAN and GNPassGAN. All passwords are unique and in *Char10*.

Range	**Training**	**Testing**	**PassGAN**	**GNPassGAN**
(0,5]	0.213%	0.213%	0.115%	0.166%
(5,8]	47.661%	47.645%	46.079%	46.178%
(8,10]	52.126%	52.142%	53.801%	53.645%
(10, ∞)	0	0	0.005%	0.011%
Total	100%	100%	100%	100%

Table 3.4: The matched passwords by PassGAN and GNPassGAN over the *rockyou* testing dataset in Char10. When 10^8 passwords are generated, GNPassGAN is able to generate 31.69% fewer duplicates, and match 88.03% more passwords than PassGAN

Passwords Generated	**Models**			
	PassGAN		**GNPassGAN**	
	Unique Passwords	**Matching Accuracy**	**Unique Passwords**	**Matching Accuracy**
10^4	9,738	103 (0.005%)	9,980	153 (0.008%)
10^5	94,400	975 (0.048%)	99,545	1,622 (0.082%)
10^6	855,972	7,543 (0.381%)	973,436	14,328 (0.724%)
10^7	7,064,483	40,320 (2.038%)	8,806,659	48,263 (4.258%)
10^8	52,815,412	133,061 (6.726%)	69,551,549	250,309 (12.647%)

$$Matching\,Accuracy = \frac{Count(set(FG) \cap set(FT))}{Count(set(FT))}$$

GNPassGAN's Output Space. Table 3.3 shows the data distribution of the training set, the *rockyou* testing set, and passwords produced by PassGAN and GNPassGAN on *Char10* by length. As expected, the distributions of the training and testing sets are quite comparable, and the output of PassGAN and GNPassGAN both have a similar distribution to the testing set. The majority of passwords have between 5 to 10 characters. Note that a small percentage of GANs-generated passwords exceed 10 characters. This is because GANs models are attempting to simulate the distribution of the training data while also attempting to achieve sample variety.

Table 3.5: The matched passwords by PassGAN and GNPassGAN over the *rockyou* testing dataset in Char812. When 10^8 passwords are generated, GNPassGAN is able to generate 61.80% fewer duplicates, and match six times more passwords than PassGAN

Passwords Generated	**Models**			
	PassGAN		**GNPassGAN**	
	Unique Passwords	**Matching Accuracy**	**Unique Passwords**	**Matching Accuracy**
10^4	9,969	7 (0.0003%)	9,983	26 (0.0012%)
10^5	98,705	116 (0.0055%)	99,917	281 (0.0133%)
10^6	992,774	974 (0.0462%)	995,713	2803 (0.1329%)
10^7	7,720,173	4,962 (0.2353%)	9,672,555	23,416 (1.1102%)
10^8	53,025,885	16,404 (0.7777%)	85,793,575	115,851 (5.4927%)

Both PassGAN and GNPassGAN were trained for 200,000 iterations, with the discriminator and generator competing and improving throughout each iteration. We want to see how GNPassGAN performs throughout iterations and evaluate if 200,000 is the optimal iteration parameter value for password file generation. To determine the association between iteration and matching accuracy, we display the proportion of unique passwords created by GNPassGAN that match the *rockyou* and *phpbb* testing sets at different checkpoints (Fig. 3.2). As shown in Fig. 3.2 (a), the greatest matching accuracy occurs at the 180,000th checkpoints for both testing sets in *Char10*. Additionally, despite the fact that our model was trained on the *rockyou* dataset, the matching accuracy on the *phpbb* dataset is greater than on *rockyou*. A possible explanation is that the passwords in *phpbb* are weaker in strength, and thus easier to guess. It also indicates that our model can perform well in cross-site guessing situations.

Comparison with PassGAN. For *Char10*, we generated passwords using the 180,000th checkpoints and compared them to the *rockyou* testing set to determine the models' guessing capability. 10^4 up to 10^8 passwords were generated. Table 3.4 compares the matching accuracy of PassGAN with GNPassGAN, and the numbers of PassGAN are taken from the PassGAN publication [29]. Hitaj et al. [29] used the $200,000^{th}$ iteration because PassGAN obtains the highest matching accuracy at the $200,000^{th}$ iteration. It is more appropriate to compare our model's performance with their best performance to demonstrate which one is superior at guessing capability. Our comparison shows that as the number of created passwords increases, both models can successfully guess more passwords appearing in the test dataset. Our model GNPassGAN is capable of guessing more passwords than PassGAN and generates fewer duplicates, which suggests that PassGAN is experiencing mode collapsing. More precisely, when 10^8 passwords are generated, GNPassGAN is able to match 88.03% more passwords than PassGAN, and generate 31.69% fewer duplicates[3].

For *Char812*, as shown in Fig. 3.2 (b), the $200,000^{th}$ checkpoint has the maximum matching accuracy for both testing sets; hence, we utilize the $200,000^{th}$ checkpoint to generate passwords in *Char812*. The performance of PassGAN and GNPassGAN in matching passwords in *Char812* is shown in Table 3.5. As can be observed, GNPassGAN continues to outperform PassGAN in terms of properly guessing more genuine passwords. When 10^8 passwords are generated, GNPassGAN is able to match six times more passwords than PassGAN, and generate 61.80% fewer duplicates. However, when compared with Table 3.4, we can find that when the same rank of passwords (10^8) are generated, the matching accuracy for passwords in *Char812* (5.4927%) is significantly lower than for passwords in *Char10* (12.647%), which demonstrates that lengthier passwords are more difficult to guess, and is consistent with other studies [15, 36]. Therefore, we emphasize the importance of imposing a minimum password length restriction of eight characters to prevent passwords from being readily guessed.

Examining Non-matched Passwords. We examined a list of GNPassGAN-generated passwords that did not match any of the testing sets and discovered that a substantial number of these passwords are plausible candidates for human-generated passwords. As a result, we expect that the passwords created by GNPassGAN might be exploited as honeyword candidates to reduce attackers' success rate at compromising users' accounts and

[3] Both implementations took about the same time to finish the training process.

Table 3.6: Passwords produced by GNPassGAN that did not match the testing sets.

claia02001	cas043712	mannda235	all53002
badanan24	nsha1105	livemilo	namrasbdo
mintesa01	jonern14	tikiocmo	dendiona
maketa11	moritin1	pilk2711	fish1053

detect data breaches.

Honeywords were introduced by Juels and Rivest as a potential method for efficiently detecting password leaks [34]. 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 distinct sweetwords (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. Table 3.6 illustrates some samples of the passwords generated by GNPassGAN that did not match the testing set but seem to be viable honeyword candidates.

3.3 Discussion

3.3.1 Limitation and Future Work

This section discusses the limitations of GNPassGAN, the future work that can be done to enhance GNPassGAN, and potential applications.

Inappropriate Password Setting. PassGAN and GNPassGAN both conduct experiments with passwords that are less than or equal to 10 characters in length, with nearly half of them being less than or equal to 8 characters. This is unworkable in practice, since the majority of websites need a minimum of 8 characters. Additionally, some websites require a mix of numeric characters, special symbols, and special characters in passwords. Previously published research [65] shows that altering password criteria, such as required minimum length and class, may have a considerable positive influence on both usability and security. As a result, while preprocessing datasets and assessing the models, we need to take the password policy and minimum length requirements into account to simulate real-world password generating scenarios.

Hybrid Models. Given that prior work [50] and [29] demonstrated that deep neural networks can mimic the domain knowledge of professional attackers, GNPassGAN can be used in conjunction with rule-based models such as HashCat to provide more accurate dynamic password guessing solutions than GNPassGAN alone.

Honeyword Generation. Because GNPassGAN is capable of synthesizing texts with the same distribution as the training data, it can generate authentic-looking passwords that can be considered honeyword candidates when trained on real-world password datasets. In Section 4, we introduce a technique to generate honeywords robust to offline trawling attacks by utilizing GNPassGAN.

3.3.2 Conclusion

In this chapter, we introduced GNPassGAN, a GANs-based deep learning password guessing tool. The original motivation comes from PassGAN [29], and by applying gradient normalization to the discriminator, modifying the loss function, and tweaking the architecture of the generator, we are able to outperform PassGAN by 88.03% while generating 10^8 passwords and create 31.69% less duplicates. The result indicates that GNPassGAN is superior than PassGAN in terms of resolving the mode collapse issue and achieving a better guessing capability. We argue that there is no need to compare GNPassGAN to traditional rule-based and probability-based password guessing tools, given that Hitaj et al. [29], the authors of PassGAN have conducted extensive experiments and shown that their work outperforms others.

We encourage researchers interested in password guessing with deep learning techniques to adopt GNPassGAN as a new state-of-the-art benchmark. Additionally, the potential for using GNPassGAN to construct password strength meters that encourage users to create stronger passwords, and honeywords that detect password breaches is promising.

Chapter 4

HoneyGAN

4.1 Introduction

In this chapter, we propose HoneyGAN, a trawling attack-resistant HGT. HoneyGAN utilizes GANs to automatically learn distributions for a large collection of unstructured data (leaked password datasets) and then utilize the learnt distribution to produce honeywords that are indistinguishable from actual passwords. Due to the policy-neutral nature of our method, it can be easily integrated into any password-based authentication system. Additionally, our machine-learning-based technique for honeyword production permits the creation of honeywords of any length or structure.

4.2 Preliminaries

4.2.1 The Honeyword Mechanism

According to Juels and Rivest [34], the honeyword system is comprised of four entities, as shown in Figure 4.2: a user U_i, an authentication server S, a *honeychecker*, and the 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 [34]. PW_i and its $k-1$ honeywords are referred to as k sweetwords.

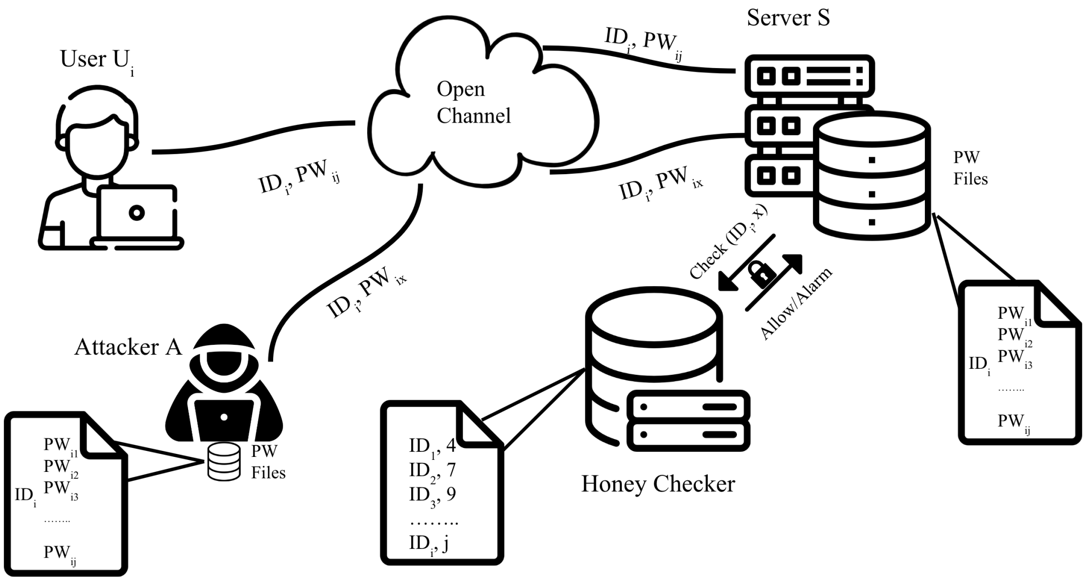


Figure 4.1: Password (PW) authentication with honeywords.

Honeyword-enabled systems could reliably identify a password file leak by pairing each user's account with $k-1$ honeywords. 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 [19, 44, 77, 79], and suppose the attackers know which k sweetwords are associated with each user, then their target is to distinguish each user's true password from the k sweetwords. The honeyword-enabled 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 other than the computer system itself [34]. 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 equal, the probability of capturing an attacker is $3/4 = 75\%$ for $k = 4$, and the probability grows as k increases.

4.2.2 Honeyword Generation and Evaluation

Automatically generating honeywords that are difficult to distinguish from genuine passwords is a difficult and intricate problem, and the strategy fails if an opponent can readily discover the true passwords. Juels and Rivest [34] presented four traditional user-interface (UI) HGTs that heavily rely on random letter, digit, and symbol substitution (chaffing-by-tweaking). These techniques were subsequently shown to be inefficient in meeting anticipated security requirements [71]. In Section 4 and 5, we demonstrate the inefficiency of their methodologies using our metrics and a user study.

In terms of assessment, Juels and Rivest [34] developed a metric for assessing the efficiency of HGTs, namely $\varepsilon{-}flat$, which quantifies the highest success rate that a prospective adversary A may achieve by submitting just one online guess in response to each user's k sweetwords. This statistic, however, is insufficient for assessing HGTs' performance when A is permitted to make multiple online guesses per user. Additionally, this statistic does not represent the system's most susceptible sweetwords which may be instantly discernible due to the fact that different sweetwords have differing odds of being chosen depending on the attacker's experience.

Wang et al. [71] later proposed two alternative evaluation measures, namely *flatness graph* and *success-number graph*, to address the shortcomings of $\varepsilon - flat$. They evaluated Juels and Rivest's HGTs [34] and concluded that they all failed under these two metrics.

Additionally, Wang et al. [71] stated that due to the Zipf-distribution of passwords [70], the probabilistic password guessing model PCFG [77] cannot be used to generate high-quality honeywords. As a result, the community should prioritize the development of new HGTs that properly satisfy the expected security requirements.

4.2.3 Text Similarity

The 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 (line 4 of Algorithm 4.1 and line 5 of Algorithm 4.2). 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 [45], FastText [6], and $TF-IDF$. While these techniques take into account the semantic and syntactic meanings of a word/text, but 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. Consider the following example: "11june1993" is a real password from the *rockyou* dataset, and the password “junte1189” created by HoneyGAN has a similarity score of 0.853 with the real password, whereas “057519189harry” and “nastymarc” have similarity scores of 0.502 and 0.071 with the real password, respectively (the latter two are generated by the *fasttext* HGT proposed in [17]).

4.3 Honeyword Generation Techniques

We propose our trawling HGT design HoneyGAN in this section, and compare (and subsequently assess) HoneyGAN to two baseline models: chaffing-by-fasttext proposed by Dioysiou et al. [17] and chaffing-by-tweaking proposed by Juels and Rivest [34]. We will use the term chaffing-by-fasttext and *fasttext* interchangeably, as well as chaffing-by-tweaking and *tweaking*.

4.3.1 HoneyGAN

The following procedure demonstrates how we generate honeywords for evaluation using GNPassGAN (shown in Figure 4.2). (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 named 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 each of the fake passwords in F and calculate text similarity scores. Here, we convert each password to a vector using BoW described in Section 4, and compute the cosine similarity of two passwords. (4) Finally, we assign the honeywords for a genuine password to the $k-1$ most similar fake passwords in F. The pseudocode of HoneyGAN can be found in Appendeix, Algorithm 4.1.

We clarify that the administrator does not need to collect plaintext passwords for Hon-

Algorithm 4.1 Generate Honeywords Using the GNPassGAN Model (HoneyGAN)

Input : A fake password list *fake* generated by GNPassGAN; a real password list *real* from data breaches, one password per user; the number of sweetwords per user k.

Output: SW, a $2D$ sweetword list $\{SW_1, SW_2, ...SW_n\}$, each user has $k-1$ honeywords.

```
Initialize a $2D$ sweatword list $SW$.
for $real\_password$ in $real$ do
    for $i \leftarrow 0; i < n$ do
        score$_i$ = $cosdis(real\_password, fake_i)$
        $Assign$k-1$fakepasswordswiththehighestsimilarityscoresto$SW$_i$.
    end
end
return $SW$.
```

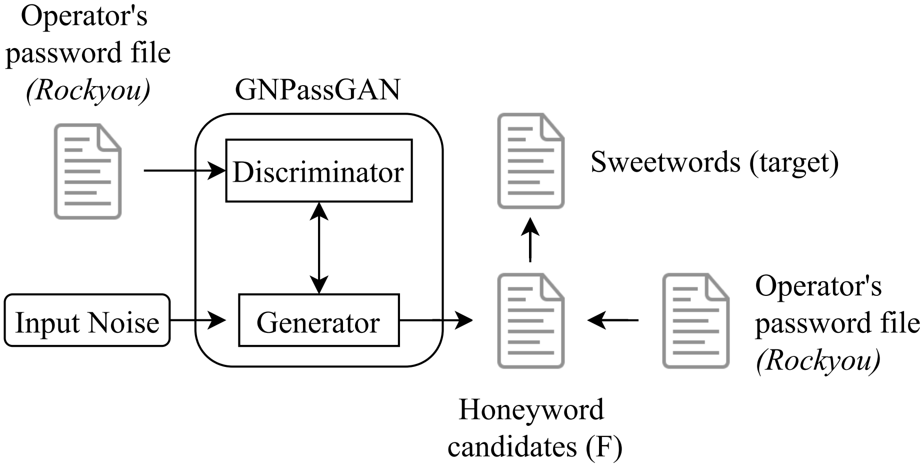


Figure 4.2: HoneyGAN workflow

eyGAN to operate on current websites. When a user creates an account, HoneyGAN associates honeywords to the user based on the similarity scores between the honeyword candidates in the pool and the user's actual password; then all 20 sweetwords are processed through whatever encryption mechanisms are in place, and stored in the system.

4.3.2 Baseline Models

Chaffing-by-fasttext

This technique was proposed by Dionysiou et al. [17] 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-fastext 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 neighbors in decreasing order of cosine similarity to create the list of $k * n$ sweetwords. As a consequence, 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.

Chaffing-by-tweaking

The concept of chaffing-by-tweaking was initially presented in [34], and it is an approach that mainly relies on random letter, digit, and symbol substitution. Dionysiou et al. [17] 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

Table 4.1: 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$

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 $q = 0.05$.

Rather than accepting duplicates as in [17], we eliminate all duplicates in each user's sweetwords, since duplication might indirectly assist attackers in choosing the correct password.

Honeyword examples generated by the three HGTs can be found in Table 4.1.

4.4 HoneyGAN Evaluation

We propose two metrics for assessing the indistinguishability of honeywords which is the second contribution of this paper: one from the standpoint of the sweetwords themselves, and another from the attacker's perspective. The performance of HoneyGAN is then compared to that of chaffing-by-tweaking and chaffing-by-fasttext in producing indistinguishable honeywords. Our attack model is based on overcoming the conceptual ambiguity inherent in Wang et al's work [71].

4.4.1 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 4.2). We analyze these datasets and choose only passwords with a length more than 8 characters, as recommended by the National Institute of Standards and Technology (NIST) in its most recent password length requirement standards[1]. Additionally, we randomly choose 10,000 authentic passwords from each disclosed dataset to facilitate in the assessment of the HGTs without sacrificing generality.

4.4.2 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 discussed in Section 4.2.3 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 differen-

[1] Complying with NIST Password Guidelines in 2021:
https://safepass.me/2021/03/11/complying-with-nist-password-guidelines-in-2021/

tiate genuine 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 [51], more sophisticated attackers would assault the sweetwords using these accessible passwords. As a result, we develop an attack model as described in Section 4.3 and assess the resilience of the HGTs based on the aforementioned assumption on attackers. The performance of an HGT is then determined by combining these two evaluation metrics.

4.4.3 Attack Model: Normalized Top-SW

Our attack model is based on Wang et al.'s work "Normalized Top-PW" [71]. The goal of Normalized Top-PW is to get the probability of recognizing genuine passwords within a user's allowed sweetword login attempts. Given an adversary A with a password file F containing $n * k$ sweetwords (where n represents the total number of users and k denotes the number of sweetwords per user), A tries a maximum of T logins per user to find as many real passwords as possible, where the probability of each sweetword $sw_{i,j}$ ($1 \leq i \leq n$ and $1 \leq j \leq k$) is derived directly from the probability distribution of a leaked password dataset D (attack dataset), such as the ones mentioned in Tabel 4.2. That is, $Pr(sw_{i,j}) = PD(sw_{i,j})$ for each sweetword in D, else $Pr(sw_{i,j}) = 0. \forall x \in D, PD(x) = Count(x)/|D|$, where $Count(x)$ is the number of occurrences of x in D and $|D|$ is the size of the leaked passwords dataset D. If the system permits multiple honeyword login attempts ($T > 1$), when a sweetword is attempted, the probability of all remaining unattempted sweetwords should be re-normalized.

The Normalized Top-PW adversary begins with the most vulnerable user accounts, i.e. those whose most probable honeyword probability is closest to 1. However, since the majority of honeywords never exist in the attack dataset, their probabilities of occurrence are all zero. Wang et al. [71] circumvented this sparsity issue by using the "+1" smoothing.

However, even after smoothing, all honeywords that are not included in the attack dataset have the same probability; thus, which honeyword should the attacker choose in this instance where all remaining sweetwords have the same probability? Should we assume that the attacker is fortunate enough to discover the real passwords for each user in a single attempt, or the opposite, that the attacker fail to compromise users' accounts within all allowed attempts? To avoid this ambiguity, we do not use probability derived on the basis of occurrences to identify the next attempted sweetword; rather, we assign each sweetword the *largest* cosine similarity of it with all genuine passwords in the attack dataset. We are assuming the attacker's best-case scenario in this case: the most sophisticated attacker can identify which honeyword is most likely to be a real password. The approach is similar to Normalized Top-PW, with the primary difference that we replace the probability of a sweetword occurring in the attack dataset with the sweetwords with the highest similarity score to all true passwords in the attack dataset; additionally, our strategy would not encounter the sparsity problem.

Our attack model, Normalized Top-SW, operates as follows: 1) Given a genuine password dataset (attack) obtained from, say, LinkedIn data breach, and the sweetword file (target). The attacker employs the BoW described in Section 2 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 score. 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. Our Normalized Top-SW technique is shown in the Appendix, Algorithm 4.2.

We employ two distinct datasets for target and attack so that we can investigate a more realistic situation in which the target and attack datasets are derived from different distributions, since various systems often use different password policies, resulting in distinct password distributions. Additionally, if the target and attack datasets are owned by the same operator and an attacker has access to both, the attacker can easily discover the genuine passwords by querying the attack dataset directly, negating the need for honeywords. As a result, it is critical to maintain a separate file for the sweetwords and one for the index of the true passwords.

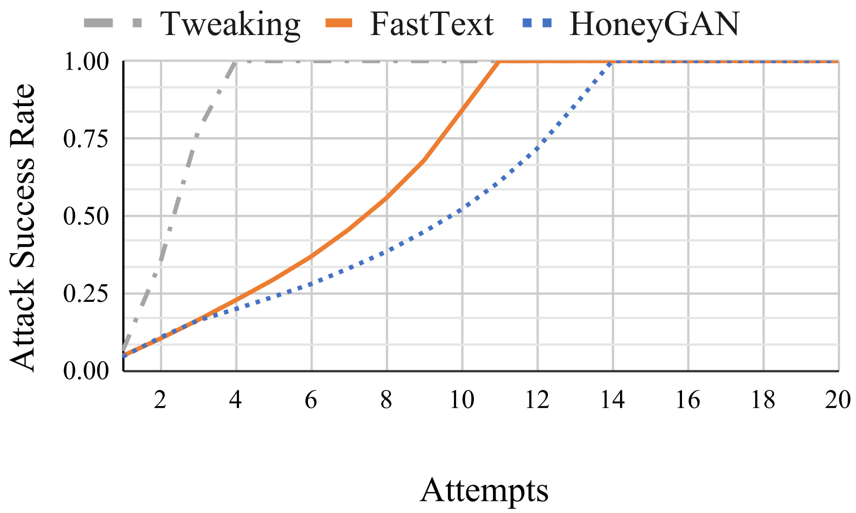
4.4.4 Results

As recommended in [34], 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 4.2. For each user, the attacker has $T = 20$ attempts.

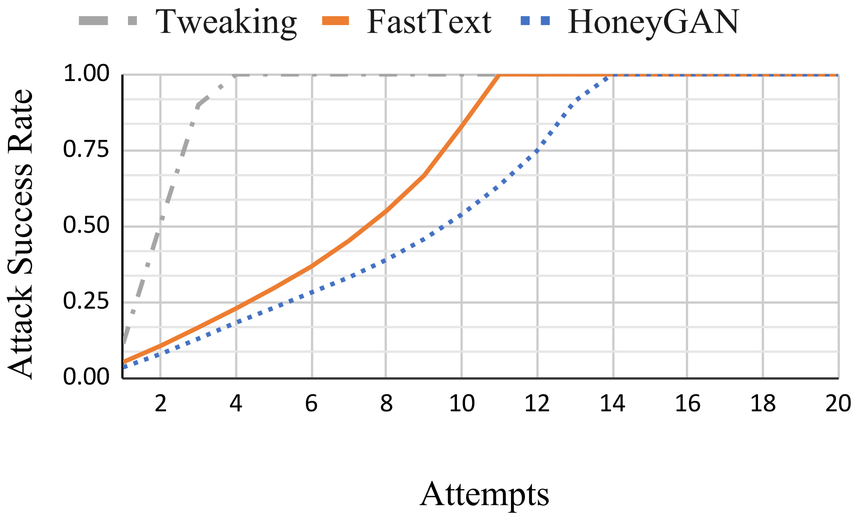
Average Internal Similarity. As a result, the internal similarity score for honeywords created by chaffing-by-GNPassGAN (HoneyGAN) is 0.8193, whereas chaffing-by-fasttext is 0.2620, and chaffing-by-tweaking is 0.6270. These numbers indicate that the honeywords created by HoneyGAN have the shortest average distance to their corresponding genuine passwords, implying that they are more similar to their true passwords and hence more difficult to differentiate.

Algorithm 4.2 The Normalized Top-SW Attack Model

```
Input : A real password list from data breaches $attack$; a $2D$ sweetwords list
        $\{SW_1, SW_2, ....SW_n\}$; the number of sweetwords per user $k$; the number of at-
        tempts allowed per user $T$.
Output: A list $success\_rate$ with the success attack rate under different number of attempts.
Initiate a $2D$ list $similarity$ to store similarity scores for all sweetwords.
 for $i \leftarrow 0; i < n$ do
    for $j \leftarrow 0; j < k$ do
        for $password\ in\ attack$ do
            $score = cosdis(password, SW_{i,j})$
        end
        Assign the highest similarity score to $similarity_{i,j}$.
    end
end
Initialize a list $success\_rate$ to store the rate of accounts being successfully attacked.
 Initialize an integer $count = 0$ to store the number of accounts being successfully attacked.
 for $a \leftarrow 0; a < T$ do
    $u = n$
     while $u > 0$ and $size(similarity_u) > 0$ do
        Get the $column\_index$ of the highest similarity score.
         if $column\_index == 0$ then
            $count++$ and delete this user.
        else
            Assign the highest similarity score to 0.
        end
        $u--$
    end
    $success\_rate_a = count/n$
end
return $success\_rate$.
```



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



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

Figure 4.3: The Attack Success Rate by using the datasets in Table 4.2 (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. 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 HoneyGAN is the hardest.

Table 4.2: The Average Success Rate of attacks on the three HGTs when various attack datasets with *rockyou* as the target dataset are used. The number of allowed attempts per user *T = 20*. 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**

Attack Success Rate (ASR). As illustrated in Figure 4.3, 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 4.2, 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.

Normalized Top-SW vs Normalized Top-PW. Our adversarial Normalized Top-SP model is inspired by Normalized Top-SW [71] and addresses its ambiguity and sparsity problems, as detailed in Section 4.4.3. As a consequence, we anticipate that our adversarial model will be more robust than Normalized Top-PW and capable of properly guessing real passwords in less attempts than Normalized Top-PW. Dionysious et al. [17] conducted experiments and discovered that when honeywords are created via *fasttext*, the adversary attack Normalized Top-PW takes 20 attempts to achieve a 100% attack success rate. In contrast, our Normalized Top-SW attack model needs only 11 attempts to correctly guess all real passwords (as shown in Figure 4.3). In this regard, our adversarial model Normalized Top-SW is more robust than Normalized Top-PW.

4.5 User Study

We introduce our third contribution in this section: apart from evaluating the three HGTs using the two metrics we proposed, we conducted a user study to test the indistinguishability of the sweetwords generated by the three HGTs. To the best of our knowledge, we are the first to conduct a research ethics-approved human participant study related to honeywords. Dionysious et al. [17] also conducted a human study to test their HGT's performance, however, their study is oversimplified: they did not mention the sample size and each survey comprised only ten questions. Additionally, their study was not approved by the research ethics committee.

4.5.1 Research Hypothesis

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*.

4.5.2 Study Design

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

Similar to previous security-related studies [32, 35, 58, 67], 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

Table 4.3: The order of each HGT appearing in the 18 survey questions.

HoneyGAN	FastText	Tweaking
HoneyGAN	Tweaking	FastText
Tweaking	HoneyGAN	FastText
Tweaking	FastText	HoneyGAN
FastText	Tweaking	HoneyGAN
FastText	HoneyGAN	Tweaking

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.

To mitigate the negative impacts of learning effects and fatigue caused by the within-group experiment, we employed the Balanced Latin Square Design [9], in which each HGT appears the same amount of times as the first, second, and third. Table 4.3 illustrates the sequence in which each HGT appears in the survey's 18 questions.

The user study documents can be found in Appendix, including test instructions, Amazon MTurk recruitment parameters, consent form, and TCPS2 certificate.

4.5.3 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 seconds and 211 minutes to complete. To ensure validity, we removed 13 of the 293 replies from participants who finished the exam in less than 3 minutes, as it is possible that they were not concentrating. Additionally, we eliminated outliers with completion time longer than 39 minutes and 30 seconds (boxplot maximum), leaving us with 272 responses to analyze.

The average completion time for the remaining 272 survey was 14 minutes with 58 seconds, with a standard deviation of 7.86 minutes, which is consistent with our expectation. This suggests 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. We analyzed the data using 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, $p \leq 0.001$).

4.6 Discussion

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

Implication of Attempts. As demonstrated in Section 4.4.4, using the Top Normalized-SW attack, attackers need in average 4, 11 and 14 attempts to successfully discover all real passwords in a sweetword file. In this case, if a website administrator sets the default number of allowed login attempts to be between 11 and 13 for each user, then attackers could easily compromise users' accounts without triggering an alert from the honeychecker if honeywords are generated by *fasttext* and *tweaking*. In contrast, if honeywords are created by HoneyGAN, as attackers need 14 attempts to find the real passwords, the honeychecker will notify the administrator of a password breach after the maximum number of attempts has been reached. In other words, the better the HGT, the higher the number of attempts required to identify all legitimate passwords.

Text Similarity Metric. In the design of Top-SW, we assign each sweetword the *highest* cosine similarity to all genuine passwords in the attack dataset to signify its likelihood of being a true password, since we are considering the best-case scenario from the attackers' perspective. To be more realistic and to eliminate outliers in the generated similarity score, we may utilize the mean or medium value rather than the maximum value.

Text similarity could also be employed during the password reset process once the website operator receives an alarm from *honeychecker* or breach notification services [48] and sends a password reset requirement to the specific user whose account has been leaked. Given that users continue to use passwords with equal strength despite the implementation of more secure password mechanisms [18], we can use the text similarity metric to nudge users to construct a stronger password that is not similar to the original one. More specifically, if the new password is too similar to the original password, it will be rejected due to the ease with which attackers might compromise this user's account by *tweaking* their stolen password.

Word Embedding. The choice of word representation technique is critical in evaluating the indistinguishibility of honeywords since it is used in both honeyword generation and evaluation process. In our study, we use the simplest BoW approach since we presume that passwords lack semantic or syntactic significance. This may not be the case for systems that rely on passphrases for authentication. In contrast to passwords, a passphrase is a whole phrase, sentence, or statement comprised of 4 to 10 words with semantic and grammatical implications. Some studies suggest that operators should employ passphrases to aid users in improving memorization [32, 62]. In this case, other word embedding approaches that include semantic meaning, such as Word2vec or GloVe [52], should be used.

F is Required to be Private. The HoneyGAN method we propose requires that the honeyword pool F be stored privately and in a safe place. This is consistent with other prior studies [17, 20]. However, the generator (including the GNPassGAN model and the HoneyGAN mechanism) can be publicly disclosed because GNPassGAN generates different honeyword pools each time; thus, even if the attacker gains access to GNPassGAN and the HoneyGAN mechanism and uses the same training data as the operator does to generate honeywords, the generated honeywords will remain different, preventing attackers from distinguishing real passwords.

Targeted Attacks. We emphasized the threat of targeted attacks in Section 1.1. Notably, 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 "*elaine@11, Mortons11, jaymeg12, gopin542, 487sheba, Newbell1, jacki5304, Makena2two, Win4Kevin, 45wootton*" which are generated using a user's real password[2] "*elaine@11*" and the HGT *fasttext* [17] trained on the *rockyou* dataset. In this case, if the attacker has no information about the user, it will be difficult to determine which of the ten sweetwords is the real password, since all of the fake passwords are from data breaches and are legitimate passwords belonging to other users. However, if the attacker knows the user's username is "*elaine*", it is quite straightforward to deduce that "*elaine@11*" is this user's real password and the others are all fake.

To the best of our knowledge, Wang et al. [74] are the only ones that discuss how to generate honeywords that are resistant to targeted attacks. In Section 5, we propose another approach that utilizes Generative Pre-trained Transformer 3 (GPT-3) to generate honeywords that are resistant to targeted attacks.

4.7 Conclusions

In this chapter, 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

[2] To prevent disclosing authors' information and to avoid using a real password that has been compromised and contains PII, we acquired written consent from one of the author's friends to use one of her real passwords in this study.

HoneyGAN is capable of creating more hard-to-find honeywords and decreasing the success rate of sophisticated attackers.

Chapter 5

Chunk-GPT3

5.1 Introduction

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 HGTs may utterly fail if attackers exploit users' 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's been suggested that, when available, PII should be used to generate honeywords [74]. We thus leverage these findings to base our HGT method on PII, 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 chunks of their corresponding real passwords.

The biggest challenge of designing an HGT is to generate honeywords that are resistant to targeted attacks [73]. 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

Table 5.1: Examples of data breaches containing PII and passwords in the past five years

Dataset	Number of Accounts	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

result of ongoing data breaches [1, 2] and people are used to creating easy-to-remember passwords using their names, birthdays, and their variants [73]. 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, rhin223, 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. [17]. In a nutshell, this HGT is first trained on a real password dataset, and it converts all real passwords in the dataset into vectors using a word embedding technique called *fasttext* [6]. For each user, the HGT assigns $k - 1$ honeywords to the $k - 1$ real passwords that have the closest distance to this user's actual password based on cosine similarity. In this case, if the attacker has no information about the user, it will be difficult to determine which of the eight sweetwords is the real password, since all of 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 the others are all fake.

Following the introduction of the honeywords security mechanism by Juels and Rivest [34], the academic community has been actively exploring the technique. However, to our knowledge, only Wang et al. [74] 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 5.1, billions of password datasets including PII are leaked. Attackers might use the PII to determine which sweetword is the real password. If all the sweetwords do not include any 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 celebrity or a public figure. Compromised accounts may have substantial financial, political, and societal consequences.

5.2 Preliminaries and Password Chunks

5.2.1 Dataset

This section introduces the password dataset (termed 4iQ) we used in this paper and password selection process. The dataset contains a leaked compilation of various password breaches over time and was first discovered by 4iQ in the Dark Web[1]. 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!, Poshmark, etc. We eliminate the suffix of each email address and only use the prefix as usernames for simplification.

To acquire legitimate passwords, we exclude those that are too short or too lengthy, with fewer than 8 characters or more than 32 characters, respectively [43, 72], resulting in 28,492 username-password pairs. Such short strings are not permitted by most authentication systems [65], 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) [81]. We further calculate the strength of each password using zxcvbn [78],

[1] 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

Table 5.2: The number of passwords in each zxcvbn score.

zxcvbn score	**num of passwords**
4	24,661
3	2,706
2	277
1	3

and found that 24,661 passwords have a zxcvbn score of 4, and only 3 passwords have a zxcvbn score of 1 (shown in Table 5.2).

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 has a zxcvbn score of 4, and the zxcvbn-weak set has passwords with score ranging from 0 to 2. We further analyze and compare the chunks in the two sets and generate honeywords for both sets with our proposed method and two other HGTs.

5.2.2 Password Chunking

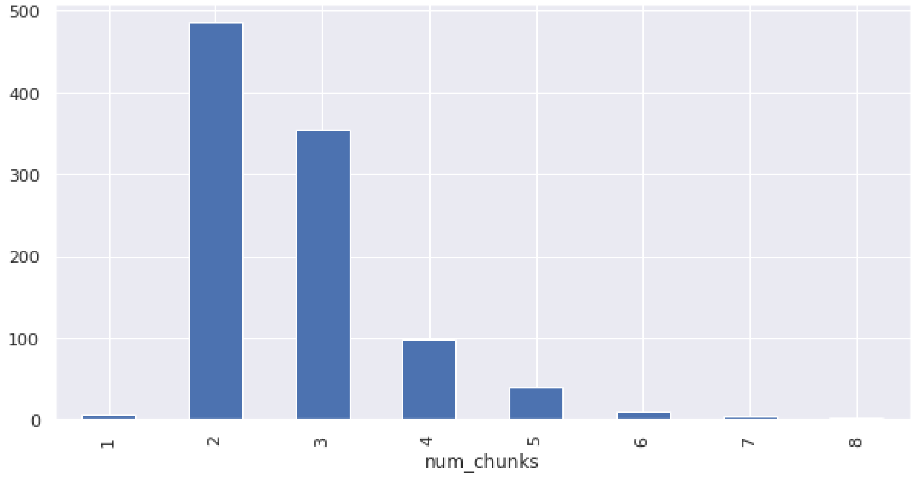
We use the password-specific segmentation technique PwdSegment [81] to interpret a password as a collection of chunks. PwdSegment conceptually trains a Byte-Pair-Encoding (BPE) technique for producing chunk vocabularies using training data of plain-text passwords. The BPE algorithm, which was initially proposed in 1994 as a data compression technique [23], is widely used in machine translation for subword segmentation (e.g., the GPT-2 model [55] proposed by OpenAI and the RoBERTa model [87] 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.

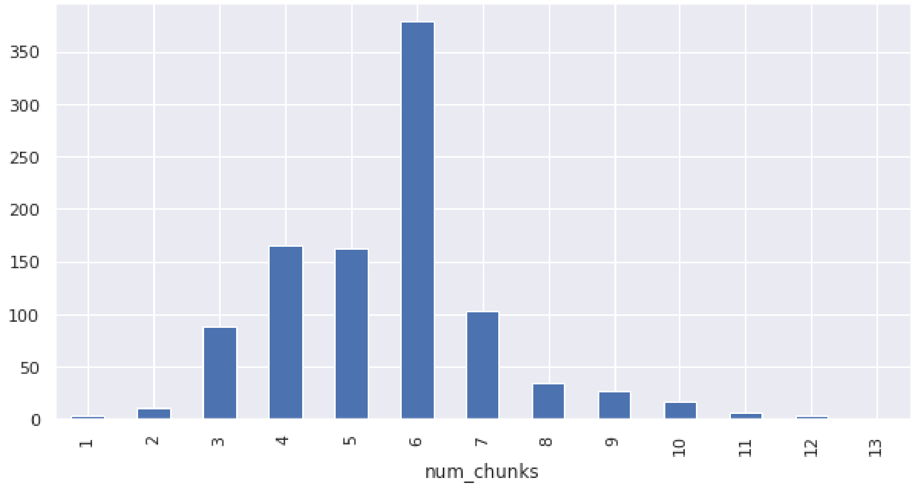
5.2.3 Chunk Analysis for zxcvbn-weak and zxcvbn-strong Password Sets

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 5.1 (b), around 370 out of 1000 zxcvbn-strong passwords contain six chunks, only around 10% contain less than three chunks. Conversely, more than 80% zxcvbn-weak passwords contain 3 chunks or less (shown in Fig 5.1 (a)).

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 5.2, from which we can observe that most chunks in the zxcvbn-weak password set contain semantic meaning or easy-to-guess patterns, such as English words (such as "*football, builder, Vietnamese, microsoft"*), phrases ("*iloveyou*"),

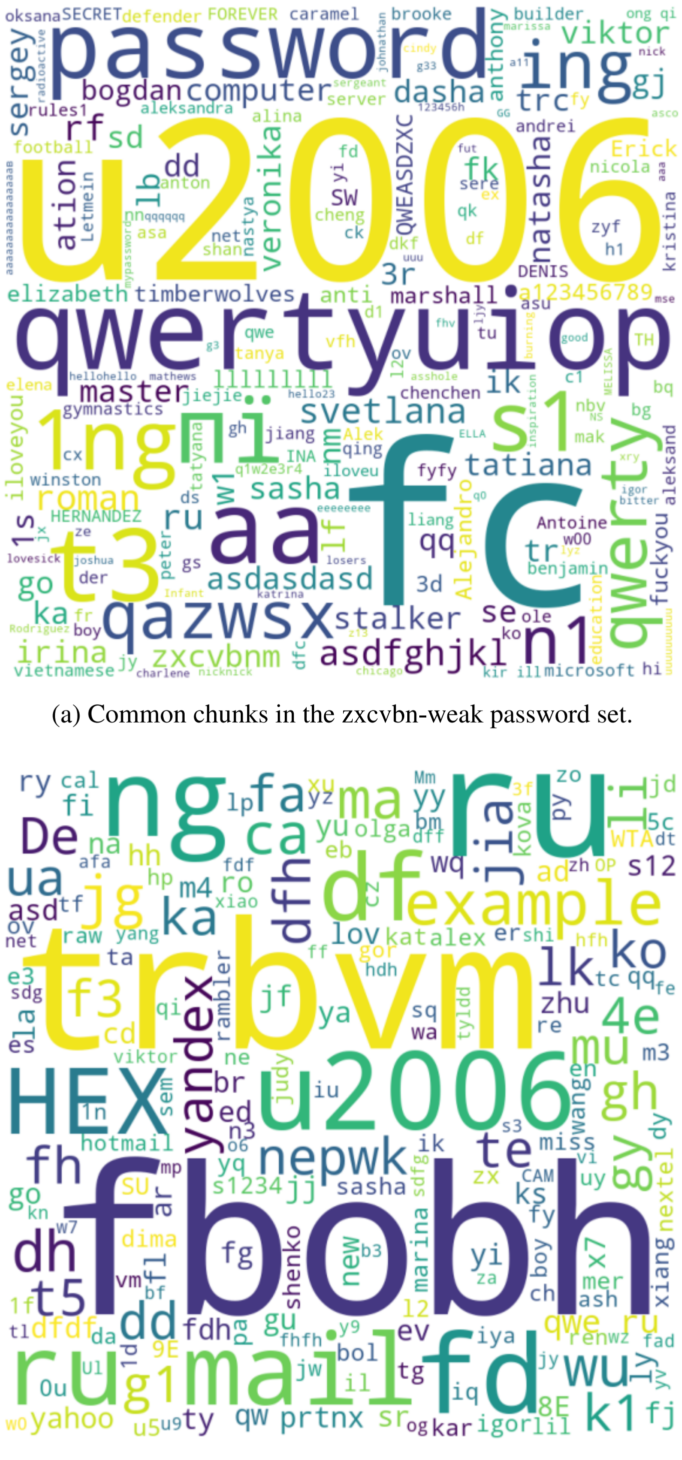


(a) The distribution of the number of chunks in the zxcvbn-weak password set.



(b) The distribution of the number of chunks in the zxcvbn-strong password set.

Figure 5.1: The comparison of password chunk numbers in zxcvbn-weak and zxcvbn-strong sets.



(a) Common chunks in the zxcvbn-weak password set.

(b) Common chunks in the zxcvbn-strong password set.

Figure 5.2: The comparison of common chunks in zxcvbn-weak and zxcvbn-strong sets.

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 apparent semantic meanings, whereas semantic words still exist (such as "*sasha, jj, wang*"). This indicates that although passwords that are zxcvbn-strong in strength are mostly comprised of short 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 a targted-honeyword-integrated system is needed.

5.3 Honeywords Generation with Chunk-GPT3

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 lanauge model to produce honeywords that retain the PII intact while altering the real passwords.

Language models can learn the probabilities of occurrences of a series of words in a regularly spoken language (for example, English) and predict the next potential word in that sequence. Generative GPT-3 is an autoregressive language model that uses deep learning to generate text that appears to be written by a person. It was introduced in 2020 by Elon Musk et al.'s AI research lab, OpenAI, and excels at a variety of NLP tasks, including translation, question-answering, and cloze [10]. The model was trained on trillions of words in text documents. It turns the words into vectors or mathematical representations

Table 5.3: Honeywords generated by GPT-3 when using different prompts. “toby” in the real password is highly likely to be a piece of PII, and honeywords generated in the prompt 2 condition replaced the PII. As we see, a more precise prompt could result in higher-quality honeywords.

Prompt 1	Suggest three passwords that are similar to “toby2009bjs”.
Honeywords	toby2009bjd, toby2009bjx, toby2009bjz
Prompt 2	Suggest three words that look like “toby2009bjs”.
Honeywords	toy2009bjs, tab2009bjs, boy2009bjs

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. It 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. Since honeyword is a new term specified in the computer security domain and does not exist in the GPT-3’s training data, we need to specify what the model should do by giving it a prompt, for example, “Derive five passwords that are similar to “*toby*2009*bjs*” 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 depends 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 [40]. The same can be seen in honeyword generation, as shown in Table 5.3.

The temperature. The temperature is a numeric variable between 0 and 1 that ef-

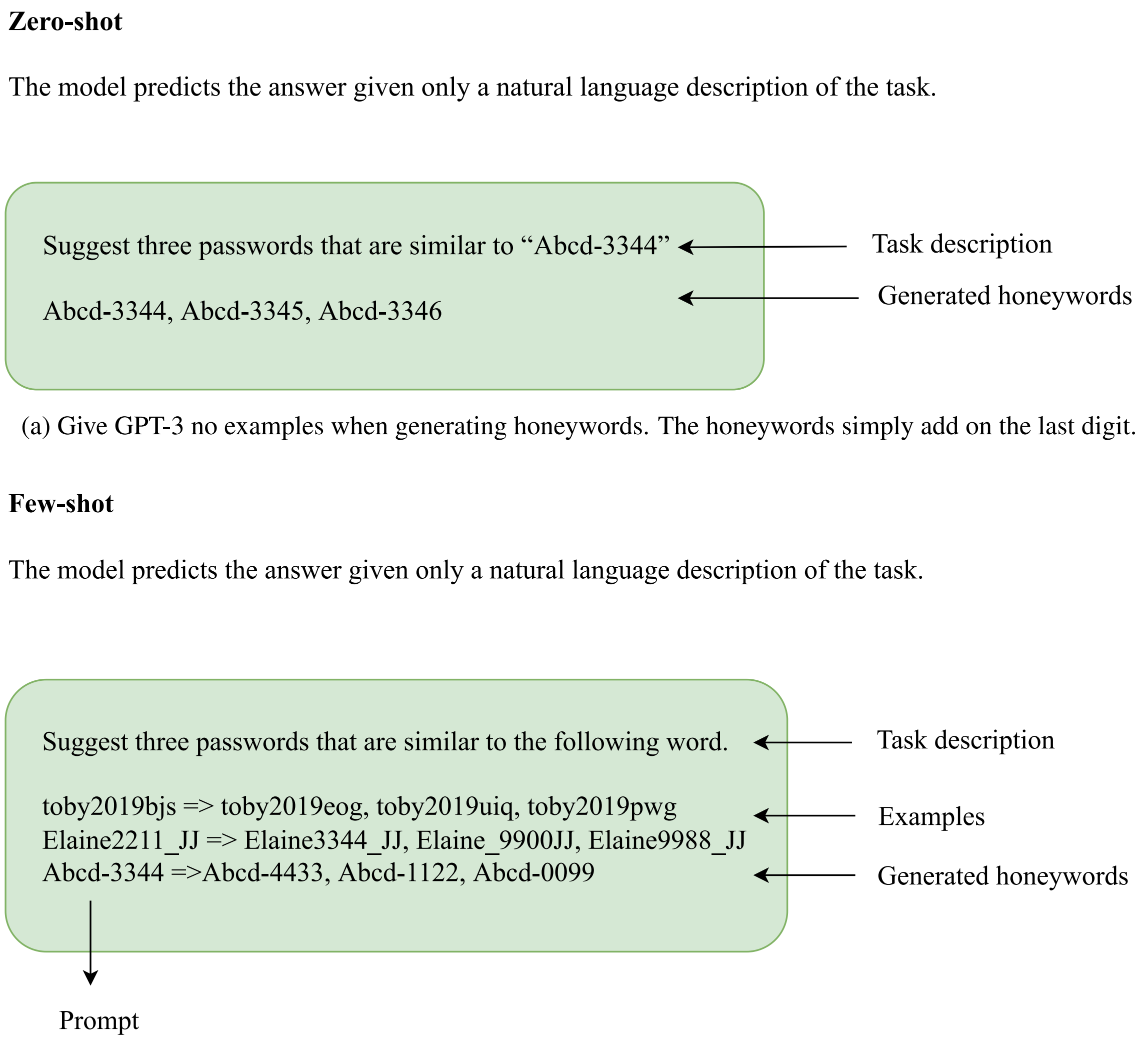
Table 5.4: Honeywords generated by GPT-3 when using different temperatures and given the prompt "Suggest five words that are similar to "toby2009bjs"". A higher temperature will result in more diverse honeywords.

Temperature	Honeywords
0	toby2009bjd, toby2009bjx, toby2009bjz, toby2009bjf, toby2009bjh
1	Toby2009BJS, toby2009bjs1, tobybjs2009, Bjs2009toby, bjs2009toby1

fectively 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 and predictable, 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 Section 5.5. Table 5.4 contains examples of honeywords formed at temperatures 0 and 1.

Zero-shot learning 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 the demonstrations. 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. An image illustration is in Fig 5.3.

Since the introduction of Generative Pre-trained Transformers, they have been extensively investigated in a variety of domains, including creating summaries of media dialogues [13], generating code from natural-language instructions [12], generating passphrases [32], and generating graphics from text descriptions [56]. To the best of our knowledge, we are the first to employ GPT-3 in the sphere of computer security, to generate honeywords



Zero-shot
The model predicts the answer given only a natural language description of the task.
Task description
Generated honeywords

Suggest three passwords that are similar to “Abcd-3344”

Abcd-3344, Abcd-3345, Abcd-3346

(a) Give GPT-3 no examples when generating honeywords. The honeywords simply add on the last digit.

Few-shot
The model predicts the answer given only a natural language description of the task.
Task description
Examples
Generated honeywords
Prompt

Suggest three passwords that are similar to the following word.

toby2019bjs => toby2019eog, toby2019uiq, toby2019pwg
Elaine2211_JJ => Elaine3344_JJ, Elaine_9900JJ, Elaine9988_JJ
Abcd-3344 =>Abcd-4433, Abcd-1122, Abcd-0099

(b) Give GPT-3 two examples for generating better honeywords. Through examples, the model can learn the expected pattern of honeywords and create honeywords with the same pattern as the real password.

Figure 5.3: The comparison of honeywords generated by GPT-3 in two conditions: given no examples and given two examples. The model is able to generate honeywords that conform to the operators’ expectations by given demonstrations.

that are resistant to targeted attacks.

The process of our HGT, termed Chunk-GPT3, is shown in Fig 5.4, and contains two main steps: 1). Passwords are segmented using PwdSegment Chunking algorithm detailed in Section 5.2.2. 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 “*Elena*1986@327”).”

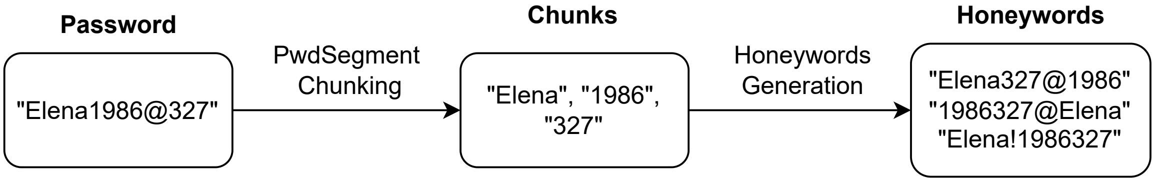


Figure 5.4: Honeyword generation with Chunk-GPT3. In this example, a password “*Elena1986@327*” is segmented to chunks “*Elena*”, “*1986*”, and “*327*” using the PwdSegment Chunking algorithm. The chunks are then used as inputs for GPT-3 to generate honeywords.

5.4 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 worse-case point perspective [71]. The honeyword distinguishing attacker is required for using the two metrics. Previous works [17, 27] 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 [84]. 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 [74] are constructing their attack models based on various kinds of capabilities allowed to an attacker (PII and registration order), we do not give these assumptions to attackers since websites would not know what kind of attackers they 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. Since there is no general targeted attack proposed in the literature, the *flatness* and *success-number graphs* are not used in our HGT evaluation. In these regards, 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 Section 5.1 and in [73].

5.4.1 Metric: HWSimilarity

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

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. Popular word embedding methods include Word2vec [45], *fasttext* [6], and $TF-IDF$ [33]. Since passwords may contain PII which has semantic meanings, while measuring the similarity of two sweetwords, the semantic meanings contained in a sweetword have to be considered. Therefore, in this paper, we propose to use a pre-trained language model MPNet [64] to encode passwords. MPNet utilizes the interdependence among predicted tokens via permuted language modeling (vs. MLM in BERT [16]) and accepts auxiliary position information as input to help the model view a whole phrase, hence minimizing position discrepancy (vs. PLM in XLNet [82]).

5.4.2 Comparable HGTs and Evaluation Results

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

The details of chaffing-by-tweaking and *fasttext* can be found in Section 4.3.2.

GPT-3 without chunks. We also tested when no chunks are given to GPT-3 when generating honeywords to examine the effect of chunks. 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 developed by Chunk-GPT3, GPT-3, *tweaking* and *fasttext* are illustrated in Table 5.5.

Results. The HWSimilarity of honeywords is shown in Tabel 5.6. 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 honeyword 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

Table 5.5: Honeyword samples generated by the four HGTs compared in the chapter (Chunk-GPT3, GPT-3, *fasttext* and chaffing-by-tweaking). *fasttext* is required to be trained on a real password dataset (the *rockyou* dataset in our experiment). Other three HGTs can generate honeywords directly without being trained on a password dataset. Only Chunk-GPT3-generated honeywords preserve the PII in the real password intact.

	HGTs			
	Chunk-GPT3	**GPT-3**	***fasttext***	**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~MIla$6421
	0000_MILA_0000	0000_lela_0000	Bushido07	364\MIlA—9353
	0000@Mila@0000	0000_lola_0000	Dampire2	3124/MiLa‘2089
007skyblueboy	Skyblueboys007	007skybluegirl	gz152sha	903SkyBlUeboY
	Blueboysky007	007babyblueboy	Calepepi	561SkYblUEbOy
	007blueboysky	007lightblueboy	hajenrai	960SKybluebOy

Table 5.6: HWSimilarity of honeywords generated by the four techniques (Chunk-GPT3, GPT-3, *fasttext*, and *tweaking*). Honeywords generated by Chunk-GPT3 have the biggest 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 the semantics contained in pass- words

	Chunk-GPT3	**GPT-3**	***fasttext***	***tweaking***
Strong PW	0.8525	0.8348	0.3441	0.7297
Weak PW	0.8367	0.8144	0.3445	0.7527

passwords considering semantics contained in passwords, and thus are harder to distinguish by targeted attacks ($t_{zxcvbn-weak}(999) = 3.935, P < 0.001$, $t_{zxcvbn-strong}(999) = 3.237, P < 0.001$) .

5.5 Discussion

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

The Chunking Algorithm. It is important that the chunking algorithm always segments chunks correctly. If incorrect chunks are segmented, the final honeywords created by Chunk-GPT3 may leak information of the real password to attackers. For example, the real password "h2omega-tania" in Table 5.5 is segmented to two chunks "home" and "tania" with "2" being removed; hence the generated honeywords contain both "home" and "tania". If all sweetwords except one include the chunks "home" and "tania", a sophisticated attacker may conclude that the sweetword that does not contain the two chunks is the real password, and that the others are all fakely produced.

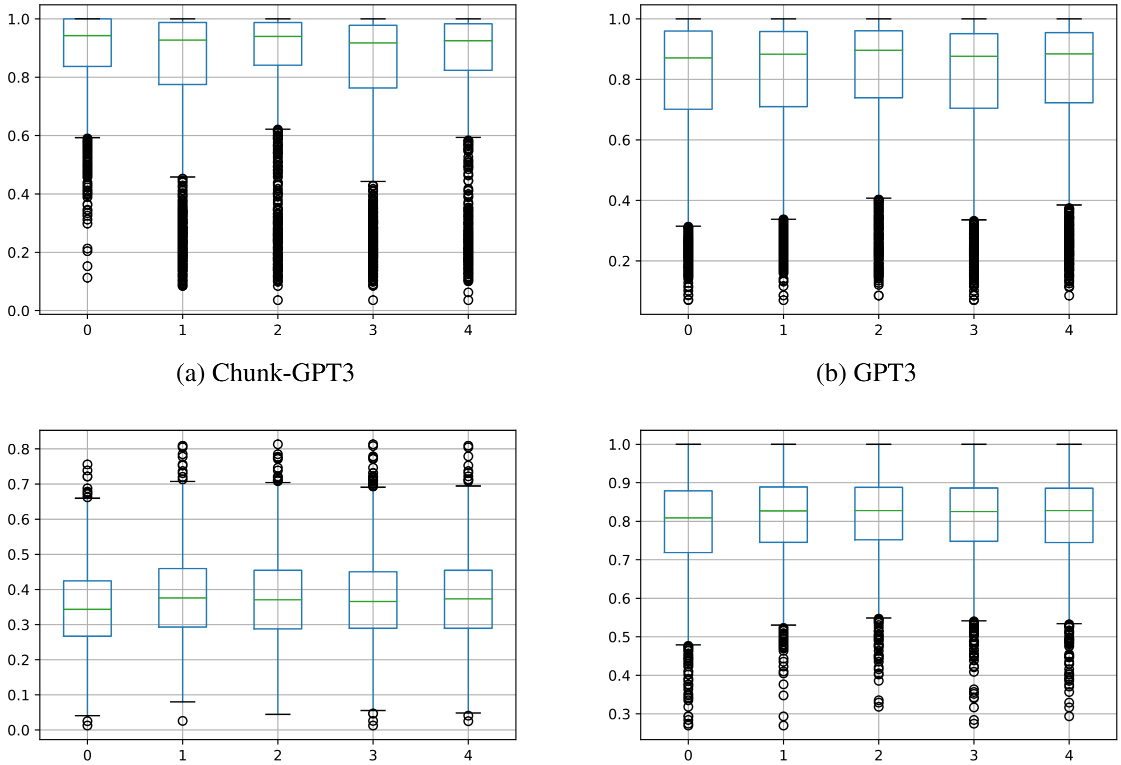
User Study. We argue that there is no need to conduct user studies to qualitatively evaluate Chunk-GPT3-generated honeywords. Since the difference is significant. If being given a question: Suppose you are an attacker and know a victim's user name is "*mila*", which one in the following list would most 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 much more difficult.

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 [37, 59] and temperature setting. We suggest to set temperature to 1 to get the most randomness [10], 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.

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, ..., and S_{20} of 19 sweetwords each (i.e., leaving a different sweetword out every time). Then for each of S_1, S_2, ..., and S_{20}, the attacker computes the HWSimilarity of each element of S_i against the sweetword that is not in S_i. Will this expose some pattern 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 of each sweetword sw_1 to sw_4 against the target sweetword p. Call this average score $\bar{p}$. 2) Then computes the average HWSimilarity score of sweetwords sw_2, sw_3, sw_4, and p against sw_1. Call this average score $\bar{a}_1$. 3) Next, computes the average HWSimilarity score of sweetwords sw_1, sw_3, sw_4, and p against sw_2. Call this average score $\bar{a}_2$. 4) Then computes the average cosine similarity score of sweetwords sw_1, sw_2, sw_4, and p against sw_3. Call this average score $\bar{a}_3$. 5) Then computes the average HWSimilarity score of sweetwords sw_1, sw_2, sw_3, and p against sw_4.



(a) Chunk-GPT3

(b) GPT3

(c) *fasttext*

(d) *tweaking*

Figure 5.5: 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.

Call this average score $\bar{a}_4$. 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 Figure 5.5. 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.

5.6 Conclusions

In this chapter, we propose a novel HGT, Chunk-GPT3, which segments passwords into 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' personal identifiable information. 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 attackers in compromising users' accounts.

Chapter 6

Conclusions

In this dissertation, we first proposed a deep learning-based password guessing model GNPassGAN, which is able to guess 88.3% more real passwords than its counterpart PassGAN when 10^8 passwords are generated. We investigated the passwords created by GNPassGAN that are not included in the test sets and discovered that they are plausible human-derived passwords that could be utilized as honeyword candidates. Hence, we proposed a novel honeyword generation technique, HoneyGAN, which uses GNPassGAN to generate honeyword candidates and selects the highest quality honeywords according to the cosine similarity between each honeyword and its respective real password. When analyzing the resilience of our honeywords by implementing the attack model presented in [71], we found the attack model's weakness and, as a result, proposed the improved attack model Normalized Top-SW. Since HoneyGAN does not take the semantics of passwords into account while producing honeywords, we created Chunk-GPT3, which segments passwords into semantic chunks and then instructs GPT-3 to generate honeywords containing the provided chunks. We evaluated honeywords produced by HoneyGAN and Chunk-GPT3 and proved that HoneyGAN and Chunk-GPT3 can generate honeywords that are more resistant to trawling and targeted assaults, respectively, than their counterparts.

Nevertheless, our work has limitations, and more work could be done in the future.

6.1 Limitations

As mentioned in Chapter 4, we created a honeyword candidate file F by using GNPassGAN in HoneyGAN. One constraint is that F must be privately kept in a secure environment, which is consistent with previous work [17,20]. The generator (including the GNPassGAN model and the HoneyGAN mechanism) can be released publicly because GNPassGAN generates different honeyword pools each time; therefore, even if an attacker gains access to GNPassGAN and the HoneyGAN mechanism and uses the same training data as the operator, the generated honeywords will remain unique, preventing attackers from distinguishing real passwords.

As mentioned in Chapter 5, we only quantitatively evaluated the indistinguishability of Chunk-GPT3-generated honeywords by measuring the cosine similarity between real passwords and their associated honeywords, and we did not conduct a user study to qualitatively evaluate their robustness. This is because when using HGTs other than Chunk-GPT3, it seems too easy for a human to tell the real password apart from its honeywords if PII is known, whereas the same task readily looks quite challenging when using Chunk-GPT3 even when PII is known. Hence we argue that a user study is dispensable.

6.2 Future work

As described in Chapter 3, our GNPassGAN model is a stand-alone deep learning model for password guessing. A future direction is to combine GNPassGAN with rule-based models like HashCat to provide more accurate and dynamic password guessing solutions. GNPassGAN may also be used to create password strength meters.

For the two HGTs we proposed in this dissertation, HoneyGAN does not take the semantics of passwords into account while generating honeywords, but Chunk-GPT3 does. Targeted honeyword generation is a somewhat unexplored field. More efforts may be put into generating honeywords that take into account the semantics of real passwords. The implementation of semantic guesser [69], a password guessing model that contains syntactic and semantic language patterns, is a potential method for generating robust honeywords resistant to targeted attacks.

We are the first to suggest using large language models to generate honeywords that are resistant to targeted attacks, and the potential is enormous. Large neural networks trained for language understanding and generation have achieved impressive results across a wide range of tasks [14]. Therefore, apart from GPT-3, other large language models such as PaLM [14] by Google, BLOOM [3] by BigScience, and YaLM [4] by Yandex could also be experimented with to test if their generated honeywords have better qualities. Moreover, a simple modification to the prompt may result in a significant change in GPT-3's completions [40, 42, 63]. Another intriguing research idea is to conduct prompt engineering to find the most instructive and efficient prompt that could lead to the highest quality honeywords in the few-shot or zero-shot settings. These prompts could be language model-automatically generated prompts [63], human-calibrated prompts [86] or chain-of-thought prompts [37, 75].

Bibliography

[1] Have I Been Pwned. https://haveibeenpwned.com/. [Online; accessed 06-October-2022].

[2] IBM security: Cost of a data breach report 2021. https://www.ibm.com/security/data-breach. [Online; accessed 01-Jan-2022].

[3] Introducing the world's largest open multilingual language model: BLOOM. https://bigscience.huggingface.co/blog/bloom, 2022. [Online; accessed 27-October-2022].

[4] Yandex publishes YaLM 100B. it's the largest GPT-like neural network in open source. https://github.com/yandex/YaLM-100B, 2022. [Online; accessed 27-October-2022].

[5] Almeshekah, M. H., Gutierrez, C. N., Atallah, M. J., and Spafford, E. H. Ersatzpasswords: Ending password cracking and detecting password leakage. In *Proceedings of the 31st Annual Computer Security Applications Conference* (2015), pp. 311–320.

[6] Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics 5* (2017), 135–146.

[7] Bojinov, H., Bursztein, E., Boyen, X., and Boneh, D. Kamouflage: Loss-resistant password management. In *European Symposium on Research in Computer Security* (2010), Springer, pp. 286–302.

[8] Bonneau, J., Herley, C., Van Oorschot, P. C., and Stajano, F. The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In *2012 IEEE Symposium on Security and Privacy* (2012), IEEE, pp. 553–567.

[9] Bradley, J. V. Complete counterbalancing of immediate sequential effects in a Latin square design. *Journal of the American Statistical Association 53*, 282 (1958), 525–528.

[10] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems 33* (2020), 1877–1901.

[11] Camenisch, J., Lehmann, A., and Neven, G. Optimal distributed password verification. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (2015), pp. 182–194.

[12] Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021).

[13] Chintagunta, B., Katariya, N., Amatriain, X., and Kannan, A. Medically aware GPT-3 as a data generator for medical dialogue summarization. In *Machine Learning for Healthcare Conference* (2021), PMLR, pp. 354–372.

[14] Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., et al. PaLM: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311* (2022).

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

[16] Devlin, J., Chang, M.-W., Lee, K., and 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)* (Minneapolis, Minnesota, June 2019), Association for Computational Linguistics, pp. 4171–4186.

[17] Dionysiou, A., Vassiliades, V., and Athanasopoulos, E. HoneyGen: Generating honeywords using representation learning. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security* (2021), pp. 265–279.

[18] Dubey, R., and Martin, M. V. Fool me once: A study of password selection evolution over the past decade. In *2021 18th International Conference on Privacy, Security and Trust (PST)* (2021), IEEE, pp. 1–7.

[19] Dürmuth, M., Angelstorf, F., Castelluccia, C., Perito, D., and Chaabane, A. OMEN: Faster password guessing using an ordered Markov enumerator. In *International Symposium on Engineering Secure Software and Systems* (2015), Springer, pp. 119–132.

[20] Erguler, I. Achieving flatness: Selecting the honeywords from existing user passwords. *IEEE Transactions on Dependable and Secure Computing 13*, 2 (2015), 284–295.

[21] Fauzi, M. A., Yang, B., and Martiri, E. PassGAN-based honeywords system. In *2019 International Conference on Computational Science and Computational Intelligence (CSCI)* (2019), IEEE, pp. 179–184.

[22] Florêncio, D., Herley, C., and Van Oorschot, P. C. An administrator's guide to internet password research. In *Proceedings of the 28th USENIX conference on Large Installation System Administration* (2014), pp. 35–52.

[23] Gage, P. A new algorithm for data compression. *C Users Journal 12*, 2 (1994), 23–38.

[24] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. In *Advances in Neural Information Processing Systems* (2014), Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, Eds., vol. 27, Curran Associates, Inc.

[25] Green, M., and Smith, M. Developers are not the enemy!: The need for usable security APIs. *IEEE Security & Privacy 14*, 5 (2016), 40–46.

[26] Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. Improved training of Wasserstein GANs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Red Hook, NY, USA, 2017), NIPS'17, Curran Associates Inc., p. 5769–5779.

[27] Guo, Y., Zhang, Z., and Guo, Y. Superword: A honeyword system for achieving higher security goals. *Computers & Security 103* (2021), 101689.

[28] Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. GANs trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in Neural Information Processing Systems 30* (2017).

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

[30] Hochreiter, S., and Schmidhuber, J. Long short-term memory. *Neural Computation 9*, 8 (1997), 1735–1780.

[31] Hranický, R., Zobal, L., Ryšavý, O., and Kolář, D. Distributed password cracking with BOINC and Hashcat. *Digit. Investig. 30*, C (sep 2019), 161–172.

[32] Jagadeesh, N., and Martin, M. V. Alice in passphraseland: Assessing the memorability of familiar vocabularies for system-assigned passphrases. *arXiv preprint arXiv:2112.03359* (2021).

[33] Jones, K. S. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation* (1972).

[34] Juels, A., and Rivest, R. L. Honeywords: Making password-cracking detectable. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security* (2013), pp. 145–160.

[35] Kelley, P. G. Conducting usable privacy & security studies with Amazon's Mechanical Turk. In *Symposium on Usable Privacy and Security (SOUPS)(Redmond, WA* (2010).

[36] Kelley, P. G., Komanduri, S., Mazurek, M. L., Shay, R., Vidas, T., Bauer, L., Christin, N., Cranor, L. F., and Lopez, J. Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms. In *2012 IEEE Symposium on Security and Privacy* (2012), IEEE, pp. 523–537.

[37] Kojima, T., Gu, S. S., Reid, M., Matsuo, Y., and Iwasawa, Y. Large language models are zero-shot reasoners. In *ICML 2022 Workshop on Knowledge Retrieval and Language Models* (2022).

[38] Lai, R. W., Egger, C., Schröder, D., and Chow, S. S. Phoenix: Rebirth of a cryptographic password-hardening service. In *26th USENIX Security Symposium (USENIX Security 17)* (2017), pp. 899–916.

[39] Li, H., Chen, M., Yan, S., Jia, C., and Li, Z. Password guessing via neural language modeling. In *International Conference on Machine Learning for Cyber Security* (2019), Springer, pp. 78–93.

[40] Liu, P., Yuan, W., Fu, J., Jiang, Z., Hayashi, H., and Neubig, G. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Comput. Surv.* (Aug 2022).

[41] Liu, Y., Xia, Z., Yi, P., Yao, Y., Xie, T., Wang, W., and Zhu, T. GENPass: A general deep learning model for password guessing with PCFG rules and adversarial generation. In *2018 IEEE International Conference on Communications (ICC)* (2018), IEEE, pp. 1–6.

[42] Lu, Y., Bartolo, M., Moore, A., Riedel, S., and Stenetorp, P. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (Dublin, Ireland, May 2022), Association for Computational Linguistics, pp. 8086–8098.

[43] Ma, J., Yang, W., Luo, M., and Li, N. A study of probabilistic password models. In *2014 IEEE Symposium on Security and Privacy* (2014), pp. 689–704.

[44] Melicher, W., Ur, B., Segreti, S. M., Komanduri, S., Bauer, L., Christin, N., and Cranor, L. F. Fast, lean, and accurate: Modeling password guessability using neural networks. In *Proceedings of the 25th USENIX Conference on Security Symposium* (2016), pp. 175–191.

[45] Mikolov, T., Chen, K., Corrado, G., and Dean, J. Efficient estimation of word representations in vector space. In *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings* (2013).

[46] Nam, S., Jeon, S., and Moon, J. Generating optimized guessing candidates toward better password cracking from multi-dictionaries using relativistic GAN. *Applied Sciences 10*, 20 (2020), 7306.

[47] Narayanan, A., and Shmatikov, V. Fast dictionary attacks on passwords using time-space tradeoff. In *Proceedings of the 12th ACM Conference on Computer and Communications Security* (2005), pp. 364–372.

[48] Pal, B., Daniel, T., Chatterjee, R., and Ristenpart, T. Beyond credential stuffing: Password similarity models using neural networks. In *2019 IEEE Symposium on Security and Privacy (S&P)* (2019), IEEE, pp. 417–434.

[49] Pascanu, R., Mikolov, T., and Bengio, Y. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning* (2013), PMLR, pp. 1310–1318.

[50] Pasquini, D., Cianfriglia, M., Ateniese, G., and Bernaschi, M. Reducing bias in modeling real-world password strength via deep learning and dynamic dictionaries. In *30th USENIX Security Symposium (USENIX Security 21)* (Aug. 2021), USENIX Association, pp. 821–838.

[51] Pearman, S., Thomas, J., Naeini, P. E., Habib, H., Bauer, L., Christin, N., Cranor, L. F., Egelman, S., and Forget, A. 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* (2017), pp. 295–310.

[52] Pennington, J., Socher, R., and Manning, C. D. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (2014), pp. 1532–1543.

[53] Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)* (New Orleans, Louisiana, June 2018), Association for Computational Linguistics, pp. 2227–2237.

[54] Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. Improving language understanding by generative pre-training. `https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf`, 2018.

[55] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. Language models are unsupervised multitask learners. `https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf`, 2019.

[56] Ramesh, A., Pavlov, M., Goh, G., Gray, S., Voss, C., Radford, A., Chen, M., and Sutskever, I. Zero-shot text-to-image generation. In *International Conference on Machine Learning* (2021), PMLR, pp. 8821–8831.

[57] Ratha, N. K., Connell, J. H., and Bolle, R. M. Enhancing security and privacy in biometrics-based authentication systems. *IBM Systems Journal 40*, 3 (2001), 614–634.

[58] Redmiles, E. M., Kross, S., and 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)* (2019), IEEE, pp. 1326–1343.

[59] Reynolds, L., and 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* (2021), pp. 1–7.

[60] Roche, T., Lomné, V., Mutschler, C., and Imbert, L. A side journey to Titan. In *30th USENIX Security Symposium (USENIX Security 21)* (Aug. 2021), USENIX Association, pp. 231–248.

[61] Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., and Chen, X. Improved techniques for training GANs. In *Proceedings of the 30th International Conference on Neural Information Processing Systems* (Red Hook, NY, USA, 2016), NIPS'16, Curran Associates Inc., p. 2234–2242.

[62] Shay, R., Kelley, P. G., Komanduri, S., Mazurek, M. L., Ur, B., Vidas, T., Bauer, L., Christin, N., and Cranor, L. F. Correct horse battery staple: Exploring the usability of system-assigned passphrases. In *Proceedings of the 8th Symposium on Usable Privacy and Security* (2012), pp. 1–20.

[63] Shin, T., Razeghi, Y., IV, R. L. L., Wallace, E., and Singh, S. AutoPrompt: Eliciting knowledge from language models with automatically generated prompts. In *Empirical Methods in Natural Language Processing (EMNLP)* (2020).

[64] Song, K., Tan, X., Qin, T., Lu, J., and Liu, T.-Y. MPNet: Masked and permuted pre-training for language understanding. In *Proceedings of the 34th International Conference on Neural Information Processing Systems* (Red Hook, NY, USA, 2020), NIPS'20, Curran Associates Inc.

[65] Tan, J., Bauer, L., Christin, N., and Cranor, L. F. Practical recommendations for stronger, more usable passwords combining minimum-strength, minimum-length, and blocklist requirements. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security* (2020), pp. 1407–1426.

[66] Thomas, K., Li, F., Zand, A., Barrett, J., Ranieri, J., Invernizzi, L., Markov, Y., Comanescu, O., Eranti, V., Moscicki, A., 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* (2017), pp. 1421–1434.

[67] Tuncay, G. S., Qian, J., and Gunter, C. A. See no evil: Phishing for permissions with false transparency. In *29th USENIX Security Symposium (USENIX Security 20)* (Aug. 2020), USENIX Association, pp. 415–432.

[68] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Red Hook, NY, USA, 2017), NIPS'17, Curran Associates Inc., p. 6000–6010.

[69] Veras, R., Collins, C. M., and Thorpe, J. On semantic patterns of passwords and their security impact. In *Network and Distributed System Security (NDSS) Symposium* (2014).

[70] Wang, D., Cheng, H., Wang, P., Huang, X., and Jian, G. Zipf's law in passwords. *IEEE Transactions on Information Forensics and Security 12*, 11 (2017), 2776–2791.

[71] Wang, D., Cheng, H., Wang, P., Yan, J., and Huang, X. A security analysis of honeywords. In *Network and Distributed System Security (NDSS) Symposium 2018* (10 2018), pp. 1–16.

[72] Wang, D., Wang, P., He, D., and Tian, Y. Birthday, name and bifacial-security: Understanding passwords of Chinese web users. In *28th USENIX Security Symposium (USENIX Security 19)* (Santa Clara, CA, Aug. 2019), USENIX Association, pp. 1537–1555.

[73] Wang, D., Zhang, Z., Wang, P., Yan, J., and Huang, X. Targeted online password guessing: An underestimated threat. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (2016), pp. 1242–1254.

[74] Wang, D., Zou, Y., Dong, Q., Song, Y., and Huang, X. How to attack and generate honeywords. In *2022 IEEE Symposium on Security and Privacy (S&P)* (2022), pp. 966–983.

[75] Wei, J., Wang, X., Schuurmans, D., Bosma, M., Chi, E., Le, Q., and Zhou, D. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903* (2022).

[76] Weinshall, D. Cognitive authentication schemes safe against spyware. In *2006 IEEE Symposium on Security and Privacy (S&P'06)* (2006), pp. 6 pp.–300.

[77] Weir, M., Aggarwal, S., Medeiros, B. D., and Glodek, B. Password cracking using probabilistic context-free grammars. In *2009 30th IEEE Symposium on Security and Privacy* (2009), pp. 391–405.

[78] Wheeler, D. L. zxcvbn: Low-Budget password strength estimation. In *25th USENIX Security Symposium (USENIX Security 16)* (Austin, TX, Aug. 2016), USENIX Association, pp. 157–173.

[79] Wu, Y., Wang, D., Zou, Y., and Huang, Z. Improving deep learning based password guessing models using pre-processing. In *Information and Communications Security*

(Cham, 2022), C. Alcaraz, L. Chen, S. Li, and P. Samarati, Eds., Springer International Publishing, pp. 163–183.

[80] Wu, Y.-L., Shuai, H.-H., Tam, Z.-R., and Chiu, H.-Y. Gradient normalization for generative adversarial networks. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)* (2021), pp. 6353–6362.

[81] Xu, M., Wang, C., Yu, J., Zhang, J., Zhang, K., and 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* (New York, NY, USA, 2021), CCS '21, Association for Computing Machinery, p. 5–20.

[82] Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R. R., and Le, Q. V. XLNet: Generalized autoregressive pretraining for language understanding. *Proceedings of the 33rd International Conference on Neural Information Processing Systems 32* (2019).

[83] Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. How transferable are features in deep neural networks? In *Advances in Neural Information Processing Systems* (2014), pp. 3320–3328.

[84] Yu, F., and 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)* (2022), pp. 10–18.

[85] Zhang, M., Zhang, Q., Hu, X., and Liu, W. A password cracking method based on structure partition and BiLSTM recurrent neural network. In *Proceedings of the 8th International Conference on Communication and Network Security* (2018), pp. 79–83.

[86] Zhao, Z., Wallace, E., Feng, S., Klein, D., and Singh, S. Calibrate before use: Improving few-shot performance of language models. In *ICML* (2021), pp. 12697–12706.

[87] Zhuang, L., Wayne, L., Ya, S., and Jun, Z. A robustly optimized BERT pre-training approach with post-training. In *Proceedings of the 20th Chinese National Conference on Computational Linguistics* (Huhhot, China, Aug. 2021), Chinese Information Processing Society of China, pp. 1218–1227.

Appendix A

Password Authenticity Test Instructions

A.1 Purpose

The purpose of this survey is to test if humans can distinguish real passwords from synthetic ones, where a synthetic password is a password generated using machine learning or statistical models, and a real password is a password found in a public domain dataset of leaked passwords. This research helps detect potential password data breaches. In this survey, you will be asked 18 "rank order" questions. Each question contains up to 19 synthetic passwords and 1 real password. You're going to sort the 20 passwords in each question according to your level of confidence that the password looks like a real one. Order 1 means you have the most confidence that this password is real, and order 20 means you have the least confidence. This survey has no time limit but it shouldn't take more than 20 mins to complete Please ensure that you answer the online survey on your own without any assistance from others. We would kindly ask you to take your time to carefully consider your answers to each of the questions.

A.2 Input format

Please drag and drop the choices (passwords) according to your level of confidence in the password's validity. You should drop the password that you believe is most likely to be real to the top of the list, the second most likely to be real to the second top of the list, and so on. You will repeat the procedure for all 18 questions.

A.3 Sample Question

There are up to 19 synthetic passwords and 1 real password in this question, where a synthetic password is a password generated using machine learning or statistical models, and a real password is a password found in a public domain dataset of leaked passwords.

Please sort the following 20 passwords according to your level of confidence that the password looks like a real one by drag and drop. Order 1 means you have the most confidence that this password is real, and order 20 means you have the least confidence.

Please do the same for the following 17 questions.

Passwords:

1. don05445
2. joseon54
3. roonn5452
4. 5ong4oni43
5. kon55544
6. noa25445
7. hoon4045
8. eon54452
9. jon54jon54

10. 5456roon
11. joo14055
12. deon445
13. cusoon554
14. koan44525
15. jonnre14052
16. jen045547
17. joodra554
18. ons425450
19. tonnn44582
20. oloven5455

Appendix B

Amazon MTurk Recruitment Parameters

B.1 Describle your survey to workers

Project Name: Password Authenticity Test (This name is not displayed to Workers).

Project Title: Answer a survey about password authenticity in your opinion.

Description: You will be asked 18 "rank order" questions in this survey. Each question contains up to 19 synthetic passwords and 1 real password, where a synthetic password is a password generated using machine learning or statistical models, and a real password is a password found in a public domain dataset of leaked passwords. You're going to sort the 20 passwords in each question according to your level of confidence that the password looks like a real one. Order 1 means you have the most confidence that this password is real, and order 20 means you have the least confidence. This survey has no time limit but it shouldn't take more than 20 mins to complete.

Keywords: Password, Authenticity Test, Survey

B.2 Setting up your survey

Reward per response: CA$5.0

Number of respondents: 300

Time allotted per worker: 20 mins

Survey expires in: 7 Days

B.3 Worker Requirements

Require that workers be masters to do your tasks: No.

Specify any additional qualifications Workers must meet to work on your tasks:

1) Location is in one of the following countries: the United States, the UK, Canada, Australia;

2) Hit Approval Rate (%) for all requester's HITs, greater than or equal to 90%;

3) The number of HITs approved is greater than or equal to 1,000;

4) Job Function: Information Technology.

Task Visibility: Hidden (only workers that meet my qualification requirements can see and preview my tasks).

Appendix C

Consent Form

Consent Form to Participate in a Research Study

Title of Research Study: How distinguishable are synthetic passwords from real passwords?

Name of Principal Investigator (PI): Miguel V. Martin

PI's contact email: Miguel.Martin@ontariotechu.ca

Names of Student Lead and contact email: Fangyi Yu (fangyi.yu@ontariotechu.ca)

Departmental and institutional affiliation: Faculty of Business and Information Technology, Ontario Tech University.

Introduction

You are invited to participate in a research study entitled *How distinguishable are synthetic passwords from real passwords?* You are being asked to take part in a research study. Please read the information about the study presented in this form. The form includes details on the study's procedures, risks and benefits that you should know before you decide if you would like to take part. You should take as much time as you need to make your decision. You should ask the Principal Investigator (PI) or study team to explain anything that you do not understand and make sure that all of your questions have been answered before signing this consent form. Before you make your decision, feel free to talk about this study with anyone you wish including your friends and family. Participation in this study is voluntary.

This study has been reviewed by the University of Ontario Institute of Technology (Ontario Tech University) Research Ethics Board [15160] on [2022/03/15].

Purpose and Procedure:

The purpose of this study is to test if humans can distinguish real passwords from synthetic ones, where a synthetic password is a password generated using machine learning or statistical models, and a real password is a password found in a public domain dataset of leaked passwords. This research helps detect potential password data breaches.

You have been invited to participate in this study because your native language is English, have some background in information technology, and you have a perfect track record of completing Amazon MTurk tasks with high-quality answers. Please kindly quit this research if you think you don't meet any of these three requirements.

You are taking this online survey in Amazon Mturk. You will be presented with 18 "rank order" questions. Each question contains up to 19 synthetic passwords and 1 real password. You're going to sort the 20 passwords in each question according to your level of confidence that the password looks like a real password. Order *1* means you have the most confidence that this password is real, and order *20* means you have the least confidence. This survey has no time limit but it shouldn't take more than 20 mins to complete. There are 300 participants in this study.

Please ensure that you answer the online survey on your own without any assistance from others. We would kindly ask you to take your time to carefully consider your answers to each of the questions.

Potential Benefits:

Your participation in the survey may help the research community to build a novel password breach detection system, and further help society timely detects data breaches.

Potential Risk or Discomforts:

Although very unlikely, it is still possible that you may find some of your own passwords in our questions. If that is the case, this would suggest that these passwords were compromised. It is recommended that you change these passwords to secure your accounts.

It is unlikely but possible, that some of the words in the questions have a negative connotation as honeywords are synthetic passwords generated by machines, and real passwords are taken verbatim from password leaks.

Use and Storage of Data:

No personal information will be collected in the survey.

The collection of the data will be occurring from April 17th, 2022 to August 30th, 2023, during which this project will be posted on Amazon Mturk.

We will use Qualtrics to design our survey and embed it in Amazon Mturk. After you finish the survey and submit it, we will receive the data from Qualtrics. The data contain all of your answers in the survey and some descriptive statistics. Amazon Mturk will provide us with your worker ID which is composed of random strings and digits. Your worker ID will be removed from the dataset and deleted by the study team immediately after the data is received from Amazon. ; we will only include your answers in the dataset. The data will be anonymized since the dataset will be stripped of the worker IDs.

After we have downloaded and analyzed the data file from Qualtrics, we will remove it from our Qualtrics account. The data file will be protected by compressing within password-protected RAR (Roshal Archive Compressed file) archives. Anybody who tries to open the encrypted RAR will need to enter the password for the archive for extraction, and the password is only accessible by our student research team member, Fangyi Yu. The RAR file will then be stored in Fangyi Yu's Google Drive and only be shared with the PI, Prof. Martin.

The analysis of the data is to count on average, at which attempt can humans find the real password among all 20 passwords for each question. We will only put the average attempts and results of the statistical analysis in the publication to represent the indistinguishability of the synthetic passwords. No personal information about you will be revealed in our publication.

The data files related to this research will be kept perpetually.

The research activity will be conducted on Amazon Mturk. We recruit participants located in English-speaking countries including Canada, The United States, The UK, and Australia. The use of the data will take place in Canada.

All information collected during this study, including your worker ID and answers to the survey, will be kept confidential and will not be shared with anyone outside the study unless required by law. You will not be named in any reports, publications, or presentations that may come from this study.

Confidentiality:

The study is anonymized given that the data will include your worker ID, which is an indirect identifier, but will be stripped upon data analysis. . Additionally, the data we will collect and analyze from Qualtircs will consist of only the responses you provide to each question and the associated descriptive statistics.

Your privacy shall be respected. No information about your identity will be shared or published without your permission unless required by law. Confidentiality will be provided to the fullest extent possible by law, professional practice, and ethical codes of conduct. Please note that confidentiality cannot be guaranteed while data is in transit over the Internet.

Voluntary Participation:

Your participation in this study is voluntary and you may partake in only those aspects of the study in which you feel comfortable. You may also decide not to be in this study, or to be in the study now, and then change your mind later. You may leave the study at any time.

Right to Withdraw:

If you withdraw from the research project at any time, any data that you have contributed will be removed from the study and you do not need to offer any reason for making this request.

You can simply withdraw by closing the webpage. Your data will not be saved in Amazon Mturk if you close the webpage before clicking submit, but you will get no incentives if you withdraw.

Please be aware that our dataset will be anonymized; we will only save the answer to each question you input in our dataset. The original data will be deleted from our Qualtrics account. As a result, once you click submit, the data cannot be removed from the dataset, and it will be impossible for us to remove your data from Amazon MTurk since we will not have a way to associate the data with your worker ID.

Compensation, Reimbursement, Incentives:

You will use your own personal computer to complete this survey.

There is no expense as a result of your participation, and you will be paid CA$5.0 for completing this online survey on Amazon Mturk after submitting your answers. The submission procedure is controlled by Amazon MTurk and we are unable to compensate you if you withdraw, and Amazon MTurk does not provide us with any information about you if you withdraw.

Debriefing and Dissemination of Results:

After all of the participants submit this online survey, we will analyze on average, at which attempt can participants find the real password among all 20 passwords for each question.

This study was done using Amazon MTurk. We do not collect personal information from you, and Amazon MTurk simply supplies us with the worker ID, which will be removed immediately after we receive the data. As a result, we are unable to offer feedback to you.

Participant Rights and Concerns:

Please read this consent form carefully and feel free to ask the researchers any questions that you might have about the study. If you have any questions about your rights as a participant in this study, complaints, or adverse events, please contact the Research Ethics Office at (905) 721-8668 ext. 3693 or at researchethics@ontariotechu.ca.

If you have any questions concerning the research study or experience any discomfort related to the study, please contact the Principle Investigator Miguel V. Martin at Miguel.Martin@ontariotechu.ca.

By signing this form, you do not give up any of your legal rights against the investigators, sponsor or involved institutions for compensation, nor does this form relieve the investigators, sponsor or involved institutions of their legal and professional responsibilities.

Consent to Participate:

I have read the consent form and understand the study being described.

I freely consent to participate in the research study, understanding that I may discontinue participation at any time without penalty. A copy of this Consent Form has been made available to me.

□ I agree
□ I disagree