Filtering Honeywords using Probabilistic Context Free Grammar

by

Alekhya Tanniru

A Capstone Project submitted in partial
fulfillment of the requirements for the
degree of

Masters

in

Information Technology Security in Artificial Intelligence

Ontario Tech University

Supervisor: Dr. Miguel Vargas Martin

October 2023

Capstone Research Project Review Information

Submitted by : **Alekhya Tanniru**

Master of Information Technology Security in **Artificial Intelligence**

Project/Major Paper title: Filtering Honeywords using Probabilistic Context Free Grammar

The Capstone Project was approved on 23rd November 2023 by the following review committee

Review Committee:

Research Supervisor:	Miguel Vargas Martin Professor Faculty of Business and Information Technology
Second Reader:	Patrick Hung Professor Faculty of Business and Information Technology

The above review committee determined that the capstone project is acceptable in form and content and that a satisfactory knowledge of the field was covered by the work submitted. A copy of the Certificate of Approval is available from the School of Graduate and Postdoctoral Studies.

Abstract

With the growing prevalence of cyber threats, effective password policies have become crucial for safeguarding sensitive information. Traditional password-based authentication techniques are open to a number of threats. The idea of honeywords, which was developed to improve password-based security, entails using dummy passwords with real ones to build a defence mechanism based on deceit. The importance of password policies is examined in the context of honeywords in this study, emphasizing how they might improve security and reduce password-related risks. We present the idea of using the existing passwords to extract a policy and using this policy to filter good and strong passwords. Through this capstone project, we aim to contribute to the broader understanding of honeywords and their role in improving password-based authentication systems. I have conducted experiments on Chunk-GPT3 and GPT 4 models, to see which one of the models produces more honeywords which are very similar to the real passwords.

Author's Declaration

I hereby declare that this capstone project consists of original work of which I have authored. This is a true copy of the work, including any required final revisions, as accepted by my committee.

I authorize the University of Ontario Institute of Technology (Ontario Tech University) to lend this work 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 work 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 work may be made electronically available to the public.

ALEKHYA TANNIRU

Acknowledgements

I would like to express my heartfelt gratitude to Dr. Miguel Vargas Martin, for his unwavering support and mentorship throughout this endeavour. His guidance and expertise have been invaluable in shaping the direction of this work. I am incredibly thankful for his belief in the importance of this research and his willingness to provide the necessary resources. I would also like to acknowledge Fangyi Yu for her significant contribution as the first author of the Honeywords paper. Her groundbreaking research forms the foundation of this project, and I am grateful for her innovative insights.

Furthermore, I extend my heartfelt thanks to all the individuals, colleagues, friends, and family members who have stood by me throughout this journey. To all those who contributed in various ways, whether through discussions, proofreading, or emotional support, I extend my appreciation. Your collective efforts have made this thesis a reality.

In writing this acknowledgment, I have tried to express my deep appreciation for the help and support I have received. However, words can only partially convey my gratitude. I am truly indebted to all of you.

Alekhya Tanniru

October, 2023

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Password security is of utmost importance in today's digital world where passwords are the primary means of authentication for various servers and databases. To improve password security now a days user's passwords are being stored as hashed password file. If an adversary can get his hands on this hashed password file, using brute force attack he can find a password whose hash value matches with hash value stored in the user's password file. This is how an attacker can impersonate the user.

There were many cases of password database breaches in the past like that of, LinkedIn in 2012, Yahoo! in 2013, Ashley Madison in 2015, Facebook / Cambridge Analytica in 2018, Microsoft in 2021 etc. At first, Yahoo reported that 1 billion accounts were stolen. But, later in 2017 they have reported that 3 billion accounts were stolen. Until now, we couldn't find any proper solution to detect such password database breaches.

Therefore, to overcome such situations Juels and Rivest [9] have proposed a new method called "honeywords" to protect hashed passwords databases against the attacks. This technique plays an important role in detecting these password database breaches. In this technique, for each user the server generated multiple false honey-

words called honeywords. These honeywords are stored in the same file as the user's password file. So, when attacker gets access to the user's password file, he will not be able to distinguish between the real password from honeywords. Based on the honeywords, there is high probability that the attacker chooses honeywords instead of the real password. When the attacker tries to login with the honeyword, the system sets-off an alarm identifying the compromise of password database. The effectiveness of this approach mostly hinges on the honeyword generating scheme's capacity to produce honeywords that are identical to the actual password.

System policies play a critical role in safeguarding personal data and digital assets from unwanted access in the dynamic world of information technology. They create policies, procedures, and standards to protect the integrity and security of passwords. The initial line of defence against unauthorized people is provided by these policies, which specify and enforce access control methods. Strong passwords are more resistant to brute-force attacks when they contain a combination of capital and lowercase letters, digits, and special characters. System policies improve overall security posture by guaranteeing user compliance with these criteria.

In the context of cybersecurity, this study investigates the relationship between system policies, strong passwords, and honeywords. The idea is that by matching system regulations with honeywords, one can further increase their strength—just like with strong passwords. The lack of specific system policies in the dataset is a problem, though. We suggest a technique to enhance and filter honeywords based on pre-existing password properties in order to close this gap. The goal of this strategy is to set up a misleading environment where it becomes difficult for potential attackers to discern between genuine passwords and honeywords.

Chapter 2

Background And Related Work

Previous papers have conducted various experiments on different types of honeyword generation techniques. Fangyi Yu in [20] proposed a new metric "HWSimilarity" that measures the similarity between honeywords, and real passwords based on the semantics meaning of the passwords. And, also compared Chunk-GPT3 HGT and other two state-of-art HGT's (chaffing-by-tweaking and chaffing-by-fasttext). The results show that Chunk-GPT3 generated honeywords are substantially more difficult to understand, potentially raising the bar for targeted attacks. But this paper has not focused on addressing the security concerns entitled with honeyword generation.

Erguler [4] in his paper mentioned if a strict policy is used to generate the honeywords, there is a chance the whole system might be affected by DoS attack. But if there is soft policy it might affect the strength of honeywords. Erguler's argument was that if the honeyword generator could achieve perfect flatness, more effective honeywords will be generated and making it difficult for the attackers to distinguish between honeywords and the real password.

To strengthen the password storage, Pagar [12] has proposed a system that uses Honeyword technique along with Honeyencryption. Passwords are encrypted using

the Honeyencryption process, which increases password security. Every password decryption attempt yields erroneous or misleading plaintext, confusing the attacker with the original password. The goal of this project is to prevent hackers from misusing user financial and personal information and to promote safe online communication by implementing robust password security.

Research was also conducted on the semantics of passwords. Rafael Veras [14] and Weir et al. [18] have used algorithms to generate password guesses by understanding the semantics of passwords. Weir et al. [18] were the first to use PCFGs in passwords to learn mangling patterns from the RockYou list and create guesses in best probability order. Weir approach used a set of non-terminal symbols for digits, special characters, and alphabets to parse each password and find the probability of each pattern. The main of limitation of this approach is that the probabilities were not assigned to each word. From this paper, I have taken the concept of how much important it is to understand the semantics of the hoenywords to make them strong.

Veras [14] proposed a framework for segmentation and semantic classification that captures the semantic essence of password samples. This framework used Natural Language processing techniques to understand the semantics of passwords. The limitation of this paper was that the algorithm was not very effective because it was generating duplicate guesses.

Since computational recognition of semantic patterns is challenging, Veras [15] also has resorted to visualization to facilitate the identification of intriguing semantic patterns in user choice. With an interactive graphic designed for their in-depth investigation, they have concentrated on the dates found in passwords. They have examined the dataset in a variety of ways with the help of visualization, including the connection between dates and the language that appears alongside them. After experiments the following patterns were found in user's choice: years after 1969 ap-

peared frequently, Name of month, two years after one another, and holidays etc. The research tells to advise against advising customers to set their password to a simple date and number sequence. The results also clearly imply the existence of specific patterns in the dates that users choose.

The well-known security technique of honey word based authentication shields the original password from server-side assaults. Still, the security community remains concerned about a few basic issues with this detention. A primary issue that requires additional attention is attaining flatness or producing honey words that bear a similar likelihood to the real password. To overcome this problem, Shubahm Sawant has introduced a new algorithm in paper [13]. The algorithm is named as 'Chaffing by password model algorithm'. The generator algorithm generates the honeywords by using a probabilistic model of actual passwords, after receiving the password from the user. There are separate character sets in the password. For example, mice3blind can be broken down into four letters plus one digit plus five letters, and then substituted with a composition similar to gold5rings. Upon, seeing these papers that use PCFG's to understand the semantics of different passwords. In upcoming sections, I would like propose a different use of PCFG's in generating more effective honeywords.

In the similar manner, PCFG's were used to segment keyboard pattern [7] and also the personal information pattern [10] were analyzed. The success rate of password guessing has significantly increased thanks to password guessing algorithms based on the aforementioned patterns.

Also, Wang [17] has proposed an algorithm that deeply analyzes the left out part of the integral segments of a password which have no semantics. He introduced Byte Pair Encoding algorithm which is used for password segmentation, extracting non semantical patterns that people often employ inadvertently while creating passwords. And then, built the responsive pass-word generator and gave the PCFGs model based

on BPE-patterns. The success rate of password guessing has significantly increased thanks to password guessing algorithms based on the aforementioned patterns.

After a comprehensive review of the existing literature, I have deduced the pivotal significance of semantic considerations in both authentic passwords and honeywords for optimizing the effectiveness of the latter. In light of the diverse applications of Probabilistic Context-Free Grammar (PCFG) models in password-related research, I advocate for the formulation of custom policies utilizing PCFGs in subsequent sections.

Chapter 3

Contributions

Creating honeywords for each user might be a good approach to prevent password databases having been compromised. However, there are a few security concerns with honeyword generation techniques that needs to be addressed. Some of the security concerns that need to be addressed are,

1. Making it difficult for an attacker to distinguish between a honeyword and a real password,

2. Trying to reduce the success rate of targeted password guessing,

3. Trying to produce honeywords that are very much like the real password.

For example, if the user's password is "$abcd123". Some of the generated honeywords are "@abcd123", "!abce456", "abcde123", "@$abc1234". Among these four honeywords, the first two are same length and almost have same composition as the real password, when compared to other two honeywords which have more length and different substrings as the real password. The attacker will be confused when the real password is placed with the first two honeywords, which means we have made difficult

for the attacker to choose the correct password. At the post processing step, we need to filter out the best honeywords. The filtering is done by following the below steps.

My contribution for this capstone project would be:

1. At first, I would use PCFG to form a structure for each real password using non-terminal symbols (something like, LSD). For example, for password like "Password@123", the structure would be $L_8S_1D_3$.

2. Then, I would similarly make structures for all the honeywords generated.

3. I would compare the honeywords structures with the real password structure and see how many honeywords structure's matched with the real password's structure.

4. Finally, I would produce a count of matched honeywords for each real password.

Chapter 4

Honeywords

According to Juels and Rivest [9], honeywords helps to improve the password security by inserting fake or 'honeyword' passwords along with real user passwords in database. The goal is to confuse attackers and making it difficult for them to identify the correct password, even if have unauthorized access to the database.

4.1 Different attacks

Below we like to see different attack scenarios which would affect the security of honeywords.

4.1.1 Brute Force Attack

In a brute force attack, an adversary tries all possible combinations of passwords until he'll find the correct password. When the honeywords are implemented, a password file includes the real password and several honeywords. So, when the attacker steals the password file, he has to deal with a lot of passwords. As there are many passwords, the attacker has to check for each password, and it takes more time to crack

the password and needs more resources. But if the honeywords generated have a predictable pattern, it will be very easy for attackers.

In this attack scenario, we'll suppose that the adversary obtains access to the password database from the server and that the honeychecker is uncompromised. Then, the flatness of the honeyword generating mechanism determines the attacker's probability of success in obtaining the actual password.

When the "user-profile" model mentioned by Akshima in [1] is employed to generate honeywords, the method will yield nearly perfect flatness since there is a high likelihood that a password will be connected to the user profile. Therefore, in this instance, a brute-force attack is not helpful in determining the password's unique identity. Under the "evolving-password" paradigm, the frequency of the honeywords is created to resemble the user's password selection. Therefore, without extra information that may be obtained from other attacks like social engineering assaults, the adversary has no advantage in guessing the actual password.

To fight against brute force attacks, honeywords must be generated randomly and be indistinguishable from authentic passwords. Implementing account lockout restrictions can also aid in the prevention of repeated login attempts, hence protecting against brute force assaults.

4.1.2 Targeted Password Guessing

Advertisers can identify real passwords from fakes by using the user's personal information. However, if the adversary cannot access the passwords of the same user across numerous sites, using the user-profile model to generate honeywords makes things more difficult for them. If the password in an evolving password model is linked to the user profile, there's a good chance the adversary will be able to tell the password apart from honeywords. A useful attack against weak, widely-used

passwords, passwords that are used on numerous websites, and passwords associated with user profiles is password guessing. In order to generate honeywords, a hybrid technique combining the two models can be used.

The security of honeywords under targeted guessing attacks, in which the attacker possesses personally identifiable information (PII) about the users, has been examined in recent studies. The first security analysis of honeywords under such assaults was carried out by Wang et al. [16], although they only looked at the legacy-UI techniques suggested by Juels and Rivest, demonstrating empirically that these techniques are unable to achieve low false-negative rates.

Zonghao Huang in his paper [8] have conducted experiments on existing honeyword-generation algorithms for users whose account credentials were compromised on other websites. In a model where the attacker has access to passwords for the same users at other sites or, in the case of false-positive attackers, even passwords for users at the defending site , they have formalized the false-positive rate and false-negative rate of honeywords. The study demonstrated that current honeyword-generation algorithms had poor tradeoffs between false positives and false negatives using formal definitions and a dataset of password leaks. Additionally, it discovered that passwords created by algorithms provide only mediocre defence against attackers that use false negatives.

4.1.3 Denial-of-Service Attack

In a DoS attack, the attacker overwhelms the system with traffic or requests, leading it to become unusable or unavailable to legitimate users. When an attacker repeatedly activates honeyword detection mechanism, it will trigger false alarms or cause excessive resource consumption, which in turn results in DoS attack. Designing more intelligent honeyword detection algorithms to ignore multiple requests from same source might help to defend against DoS attacks.

4.1.4 Dictionary Attack

It's possible that the dictionary is just a compilation of word lists that people frequently use to generate mnemonic passwords. Users, however, rarely employ the unaltered items from these lists; instead, they alter the terms in a way that preserves their recallability. A dictionary attack attempts to replicate this common method of choosing passwords by utilizing pre-selected mangling algorithms to process words from an input dictionary and generate variants in a methodical manner. A word-mangling rule that appends the numeral "9" to a dictionary word, for instance, might generate the guess "password9" from the dictionary word "password."

If for two users', the generated honeywords are same. The attacker can easily build custom dictionaries to predict the pattern. Then they launch dictionary attacks, testing each honeyword along with real password to find the correct match.

A dictionary attack cannot succeed unless both the original word and the appropriate word-mangling rule are present in the attacker's input dictionary. Even while dictionary-based attacks are typically faster than brute-force attacks, attackers are still constrained by the number of word-mangling rules they can employ because of time restrictions. As the dictionaries' sizes increase, these limitations get more severe. It becomes crucial in this situation to choose rules that offer a high degree of success while minimizing the amount of guesses needed for each dictionary word.

4.2 Honeychecker Function

The idea behind the honeywords was to use it as a defense mechanism. Real passwords and honeywords both are salted and hashed before being stored into database. Even if two user's same password, salting makes sure both the users honeywords are unique. Juels and Rivest [9] have proposed a honeychecker function that is used to identify

when an attacker is trying to crack passwords.

In this approach, for each user U_i, a list of W_i distinct words (sweet words) are generated by Gen(k) randomly which contains one real password of the user and (k-1) honeywords, where k is fixed system-wide value of k = 20. When a new user registers, the system generates honeywords for this user. Using hashing and salting techniques, the user's real password and honeywords are converted into hexadecimal format and stored into the database.

Next time, when the user logins to the system using their username and password, the server checks if that password is there in the database or not. If the password is not in the database, the system says the login is unsuccessful. If the password is present in the database, the server sends it to honeychecker to verify whether it is real password or honeyword. If the honeychecker says that the password is honeyword, then it sends an alarm to the system administrator. Otherwise, the user can login successfully [11].

4.3 Dataset

The dataset used in this project for password selection process is referred as '4iQ' from [20]. This dataset is taken from DrakWeb2 in 2017 which contains various leaked password breaches from Chegg, Canva, LinkedIn, Yahoo etc. . The dataset has 1.4 billion email-password pairs, with 1.1 billion unique emails and 463 million unique passwords. Duplicate email-password pairs were removed. For the sake of simplification, we have removed the email address suffix and use only usernames.

To obtain valid passwords, we have striction on the length of passwords to be more than 8 characters and less than 32 characters, this resulted in 28,492 username-password combinations. We then used zxcvbn to calculate the strength of passwords.

We found that 24,661 passwords had a score of 4, 2706 passwords had a score of 3, 277 passwords had a score of 1 and 3 passwords had a score of 0.

To compare different honeyword generation technique's, we have divided the dataset into two parts. One part has strong passwords whose zxcvbn value is 4 and the other part has weak passwords whose zxcvbn values ranges from 0 to 2.

Chapter 5

Our Approach

5.1 Pre-processing

For this Capstone project, we are using the honeywords generated by Chunk-GPT3 and GPT4 models. Chunk-GPT3 is a deep learning language model introduced in 2020 which generates script that looks to be written by a person. This model excels at different NLP tasks such as question-answering, translation. This model based trained on a lot of words from documents, then convert the words into vectors, and decodes into human readable texts. The model may be used to do NLP tasks without the need for fine-tuning on specific downstream task datasets, and it can produce texts that are difficult for humans to distinguish from human-written articles. This model segments passwords into chunks and then uses GPT model to generate high-quality honeywords which uses personal information in user's real passwords.

For this model we need to specify prompt which means giving instructions to the model. The better and instructive prompt will give you better honeywords. Using PwdSegment Chunking algorithm the passwords is divided into chunks. Later, these chunks are given as inputs to Chunk-GPT3 to generate honeywords.

5.1.1 A Password-Specific Segmentation (Pwd Segment) Algorithm

In the paper [19], Pwd segment method was presented to divide a password into chunks. In order to obtain chunk vocabulary, PwdSegment conceptually trains a Byte-Pair-Encoding (BPE) algorithm using training data of plain-text passwords. This idea is related to WordSegment, a parser tool that uses a trillion natural language corpora to train an n-gram model. The BPE algorithm, which was first developed in 1994 as a data compression method, is frequently used in machine translation to perform subword segmentation, which separates uncommon words into several units while maintaining the frequency of the common words.

First, the plain-text corpus is used to train the raw BPE algorithm. The most common pair of tokens is then iteratively combined with a single, novel (i.e., unseen) token to form the subword (i.e., chunk) vocabulary. Each merge operation creates a new chunk by substituting a brand-new, unused subword (such as "w0") for the most frequent pair of characters or character sequences (for example, "w", "0"). The merge operation is performed a predetermined number of times (i.e., a hyper-parameter) to produce a chunk of vocabulary that is proportionate in size.

Above figure illustrates the workflow of Pwd Segment as follows:

1. First, we setup the dataset with plain-text passwords and calculate the average length (avg_len) of the resulting chunks.

2. Then we count the occurrences of passwords in the training set and split the passwords into character sequences. For example, the password "last4ever" appears 2 times, we write it as "last4ever : 2" and the password " p@ss0rd123" appears 4 times, we write it as "p@ssw0rd123 : 4".

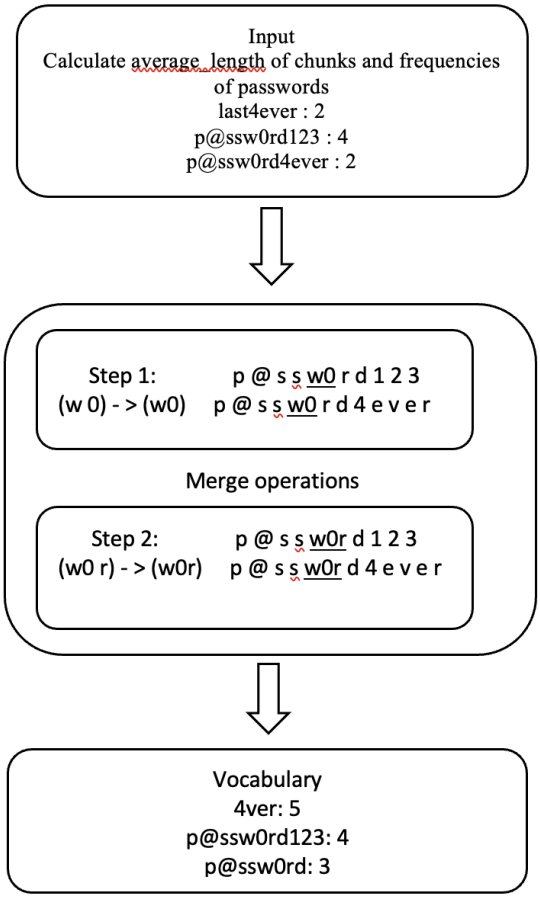

Figure 5.1: An example of Pwd Segment Algorithm

3. Merge character pair sequentially in decreasing order of frequency. The character pair "w 0" is combined as w0 in step 1, where its frequency is highest (7 appearances). As you can see from Figure 1's Merge operation, PwdSegment selects "w 0" based on dictionary sequences even though "p @" also appears seven times; The top character pair "w0 r," which also appears seven times, is merged into "w0r" in step 2. The next step 3, which uses the identical techniques, merges "w0r d" into "w0rd."

4. Continue doing this until either all character pairings have the same frequency or the average length of the generated chunks is equal to or larger than the threshold. Characters or chunks can be found in the final chunk vocabulary. Last but not least, PwdSegment divides passwords into chunks based on chunk vocabulary. For instance, "p@ssw0rd4ever" may be read as "p@ssw0rd, 4ever".

5.2 Probabailistic Context free Grammar

A context-free grammar is a defined as G = (V, Σ, S, P), where: V is a finite set of variables (or non-terminals), Σ is a finite set of terminals, S is the start variable, and P is a finite set of productions of the form :

$(\alpha \rightarrow \beta)$

where β is a string made up of variables or terminals and α is a single variable. The collection of strings made up of every terminal that can be derived from the start symbol is the grammar's language.

To represent this grammar, we only use alpha variables (L_n), digit variables (D_n), special variables (S_n) for specified values of n. For this project, we are using this grammar to learn the compisition of each password.

A probabilistic context free grammar (PCFG) is a context free grammar with

related probability for its output. A PCFG is used to parse or produce sentences from a language's syntax, or how words are grouped together and relate to one another as heads and dependents. Weir et al.'s [18] use of PCFGs in passwords was the first. They learned how to manipulate patterns from the RockYou list and produced guesses in the order of highest probability.

The work in computational linguistics to comprehend the structure of natural languages gave rise to grammar theory to represent symbol strings. Almost forty years after its introduction in computational linguistics, probabilistic context free grammars, or PCFGs, have been applied to probabilistic modelling of RNA structures.

In the same way as hidden Markov models extend regular grammars, PCFGs extend context-free grammars. A probability is assigned to each production. The product of the probabilities of the productions employed in a derivation yields the probability of that derivation (parse). These probabilities can be thought of as model parameters, and it is straightforward to use machine learning to discover these values for huge issues. The context of a probabilistic grammar's training dataset limits its validity.

I have taken this concept of probabilistic context free grammar to understand the semantics of honeywords and real passwords. In a word, Iam representing alphabets as 'L' (not differentiating upper and lower case letter), numbers as 'D', special characters as 'S'. I have written a common function for both real passwords and honeywords which returns the segments as "$L_nS_nD_n$"

5.3 Post Processing

In this section, I'm going to discuss about the post processing steps of the honeywords generated by GPT models. The dataset doesn't have any policy and from previous

sections it was clear that we need to have a good policy that makes strong honeywords. So, we thought of making our own policy.

Our idea is to form policy for each real password by dividing each password into segments which captures the length of observed substrings. For example, if the real password is "Password@123", then PCFG segment would be "$L_8S_1D_3$" . This policy would be the basis for selecting honeywords. Each honeyword is also divided into PCFG segments. Then, we compare real password PCFG segments with each of its honeywords PCFG segments. We are using this PCFG segment methodology, to found out how many honeywords are generated with similar composition as their corresponding real passwords. I am conducting experiment with both Chunk-GPT3 and GPT- 4 models, to find out which model produces more honeywords that are indistinguishable to the real passwords.

Chapter 6

Large Language Models

As mentioned in paper [6], Natural language processing (NLP) requires the critical task of language modelling (LM), which foretells the subsequent word or character in a text sequence. It entails creating models and algorithms that can comprehend and produce coherent human language. Early models employed straightforward statistical methods, but the emergence of deep learning and massive data has prompted the creation of extensive language models. Deep learning's emergence was a turning point in the evolution of Large Language models (LLM).

LLMs are deep neural networks that have been trained on a variety of text data from different sources, including books, social media, and online content. The model can produce language that is cohesive and resembles human writing thanks to this diversity. However, LLMs also demonstrate inventiveness, comprehending the vocabulary, tone, and writing styles of the training dataset. By using various input data, the model is able to produce original and sincere responses to user questions, producing results that are relevant to the query submitted. LLM's origins can be traced back to the earliest studies on Natural Language Processing (NLP). The original language models, which were rule-based and based on manually created linguistic

rules, had some limitations. In the 1980s and 1990s, statistical language models were developed, which estimated word likelihoods using probabilistic techniques. These models were more precise, but they were still incapable of comprehending the context and semantics of the language. Neural language models were developed in the middle of the 2010s, including the recurrent neural network language model in 2010. These models could simulate word context and generate more writing that sounded natural. Google's Google Neural Machine Translation system, which was released in 2015, was the first extensive neural language model.

As per the paper [6], enabling parallel training on several GPUs and learning longer-term linguistic relationships, the Transformer model revolutionized LLMs in 2017. With 117 million parameters, OpenAI's GPT-1 in 2018 showed the potential of transformers in NLP tasks. Despite its shortcomings, it paved the way for more robust models and a new phase of AI research in LLMs. GPT-3, which was able to produce extremely cohesive and natural-sounding text, was made available by OpenAI in 2020. LLMs' promise for a variety of NLP tasks was shown by GPT-3. As a result of GPT-3's success, OpenAI developed the next version of their language model, GPT-4, which can produce texts that are even more coherent and natural-sounding. After the success of GPT-4, Meta also unveiled the Llama. LLMs are a subset of Generative AI that are created with the express purpose of producing human-like language in response to a particular prompt. Examples of LLMs are Google Bard, ChatGPT, and Llama. These models learn the statistical patterns of language by being trained on enormous volumes of data using methods like unsupervised learning.

Due to their outstanding performance across a wide range of NLP tasks, such as text generation, translation, summarization, question-answering, and sentiment analysis, LLMs have attracted a lot of attention in recent years. These models, built on the transformer architecture, have a remarkable ability to process and produce

text that resembles human speech by utilizing enormous amounts of training data on a variety of subjects.

6.1 Chunk-GPT3 Model

GPT-3 was released by OpenAI in 2020. It can produce text with character-level precision and has been trained on a corpus of more than 1 billion words. An encoder and a decoder are the two primary parts of the GPT-3 design. The encoder uses the previous word in the phrase as input to create a vector representation of it. This vector representation is then fed through an attention mechanism to produce the prediction of the following word. The decoder outputs a probability distribution over all possible words given the inputs of the previous word and its vector representation. It employs a context that is 2048 tokens long and has 175 billion parameters, which at the time required 800GB of storage.

A subclass of GPT-3 Models called Generative Pre-trained Transformer 3.5 (GPT-3.5) was developed by OpenAI in 2022. It is a big language model that can produce text that resembles what a human would write because it has been trained on a vast amount of text data. Text generated by GPT-3.5 can be coherent and consistent with the context that is given to it. It can be used to generate text, summarize text, and answer questions, among other things. Now, GPT 3.5 is used to describe the code-davinci-002, text-davinci-002, and text-davinci-003 models. Models that can be referred to as GPT-3.5 are incorporated into the well-known ChatGPT application and other services of OpenAI Payground including text completion.

Based on Oğuz's paper [2] on academic writing with GPT-3.5, academics have recently begun to favour GPT-3.5 and its predecessors, GPT-3, and GPT-2, for their use in many scientific communication contexts, such as tackling writing issues, navigating

dense literature, and offering descriptions of topics . Twitter threads that show how ChatGPT might be used in a "smart" fashion have been published. Independent specialists conducted studies to evaluate the quality of the material, and they discovered that it provides high-quality results that are difficult to discern from human-generated information. GPT models have also been suggested as a tool to help pupils with their academic writing. On a test set of Wikipedia articles, Microsoft's Turing NLG model can produce text with character-level accuracy, but it needs a ton of training data to do it. After its initial pre-training phase, OpenAI asserts that GPT-3.5 can perform at this level without requiring any additional training data. Additionally, compared to older models like Google's BERT and Stanford NLP's Transformer, GPT-3.5 can produce lengthier phrases and paragraphs.

Until now, we have seen the uses of GPT-3 model to generate honeywords. Nilesh Chakraborty in his paper [3], have established a minimal attacker and demonstrate that, given a set of honeywords of its own, GPT-3 can predict the real password with a remarkably high percentage. This work aims to address the concerns related to the usage of GPT-3 for generating honeywords. Specifically, it clarifies how GPT-3 can be effectively utilized to conceal the real password under various attacker models by capitalizing on its strengths. Through the use of real-world datasets in their experiments, they were able to show not only how feasible the threat is, but also offer crucial information on how to effectively mitigate its effects. The work in this paper has provided useful reference for future investigations, permitting the prudent and efficient application of GPT technology to strengthen password security protocols.

6.2 GPT - 4 Model

The most recent accomplishment in OpenAI's endeavour to scale up deep learning is GPT-4, which was established by the OpenAi team. GPT-4 is a sizable multimodal model that accepts image and text inputs and emits text outputs. While less effective than humans in many real-world situations, GPT-4 performs at a human-level on a variety of academic and professional benchmarks. There was an example mentioned in paper [5], where it successfully completes a mock bar exam with a score in the top 10% of test takers, as opposed to GPT-3, whose score was in the bottom 10%. Our adversarial testing program and ChatGPT lessons have been used to iteratively align GPT-4 over the course of six months, and the results are our best-ever ratings for factuality, steerability, and staying inside the boundaries.

We completely rebuilt our deep learning stack over the last two years, and we co-designed a supercomputer for our workload with Azure from the bottom up. We trained GPT-3 as the system's initial "test run" a year ago. We identified several problems, rectified them, and strengthened our theoretical underpinnings. Our GPT-4 training run was consequently unprecedentedly steady, making it our first large model whose training performance we were able to precisely forecast in advance. We intend to improve our methods as we keep our attention on dependable scaling in order to better foresee and plan for future capabilities, which we believe is essential for safety.

The distinction between GPT-3 and GPT-4 can be difficult to make in informal speech. When the difficulty of the task reaches a certain level, GPT-4 distinguishes itself from GPT-3 by being more dependable, inventive, and capable of handling far more complex instructions. On its blog, OpenAI promoted GPT-4 as being more trustworthy, inventive, and able to handle far more complex instructions than GPT-3.

In comparison to GPT-3.5 and GPT-3, which had pop-up windows with a maximum of 4,096 and 2,048 tokens, respectively, the organization has created two versions of GPT-4 with pop-up windows of 8,192 and 32,768 tokens. GPT-4, in contrast to its predecessor, can accept both text and image inputs.

The technical study specifically avoided mentioning the model's size, architecture, hardware, or training technique, reflecting the closed stance that OpenAI takes to the technical aspects of GPT-4. Although The Verge mentioned rumours that GPT-4 would considerably increase the number of GPT-3 parameters from 175 billion to 100,000 billion, the exact number of GPT-4 parameters is yet unknown.

Chapter 7

Experiments

In this section I would like to discuss about the results of the conducted experiments on GPT - 3 and GPT – 4 models.

7.1 Chunk-GPT3

In this project, for GPT-3 we are using text-davinci-002 model because this is the existing model that have been used by Fangyi in her paper [20]. As Honeyword Generation Technique's irreversibility is very crucial, we must make sure that the honeywords we developed are unreplicable even if attackers are aware of our methods and the criteria, we used to construct them, such as the prompt and the temperature. This is made possible through rigorous temperature control and careful prompt engineering . To achieve the maximum randomization, we advise setting temperature to 1 and after experimenting with many prompts, we chose to utilize the following prompt:

"Derive 19 passwords that are similar to " + real_password + " and contain" + chunks +" . The length of the passwords should be at most " + str(len(real_password)) + ". Do not add digits at the end of the passwords."

As this prompt was able to generate more diversified honeywords, when compared to other prompt. I'm using the similar prompt for GPT -4 to generate honeywords.

The arguments passed to the GPT model were the real passwords and the chunks. Chunks were generated by the Pwd Segment algorithm. The output is then cleaned to make sure there is no honeyword with length less than 8. Then they are stored into a csv file. The csv file consists of 20 columns in which first column represents the real password, and the rest 19 columns represent the honeywords. Later all the columns are combined, to an array for each row.

Then each array is passed through the pw_segment function, in which the polices (composition) of each honeyword and the real password is extracted. The first element in each array is the real password. So we compare the first element with all other elements in the array to find out how many matched honeywords are there for each real password.

Below graph shows the number of matched honeywords for the first 50 strong real passwords.

X-axis – represents the real passwords

Y-axis – represents the matched honeywords (value lies in between 0 to 19)

Seeing the graph, we can interpret that most of the real passwords have zero matched honeywords to less than 5 matched honeywords.

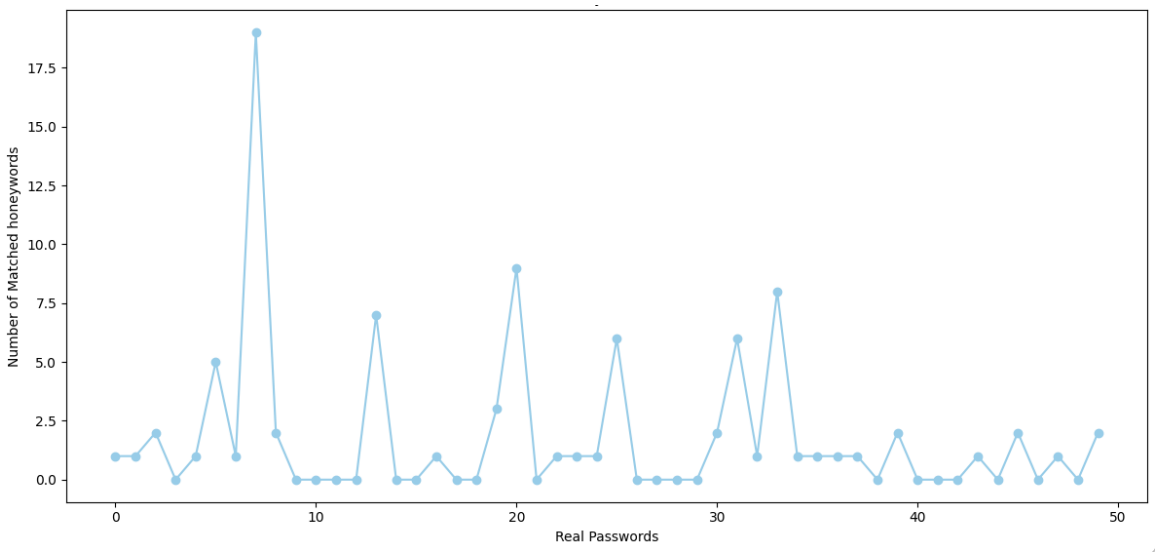

Figure 7.1: Indistinguishable Honeywords generated by Chunk-GPT3

7.2 GPT - 4

For generating honeywords from GPT – 4, we are passing 1000 strong real passwords and not chunks as the argument. The reason behind this would be to check how better honeywords is the GPT-4 generating even without the chunks. The prompt used for GPT -4 to generate honeywords is as follows:

"Derive 19 distinct passwords that are similar to " + real_password + " . The length of the derived passwords should be at most " + str(len(real_password)) + ". Do not add digits at the end of the passwords."

After the honeywords are generated, we clean them and convert them into a csv file. As the file contains 20 columns, we combine all of them to form an array for each row. In each array, the first element is the real password. We send each row into the pw_segments (password segments) function, which results in the policies (compositions) of real password and honeywords. After getting all the policies, we compare the first element policy i.e., the real password policy with all the honeywords

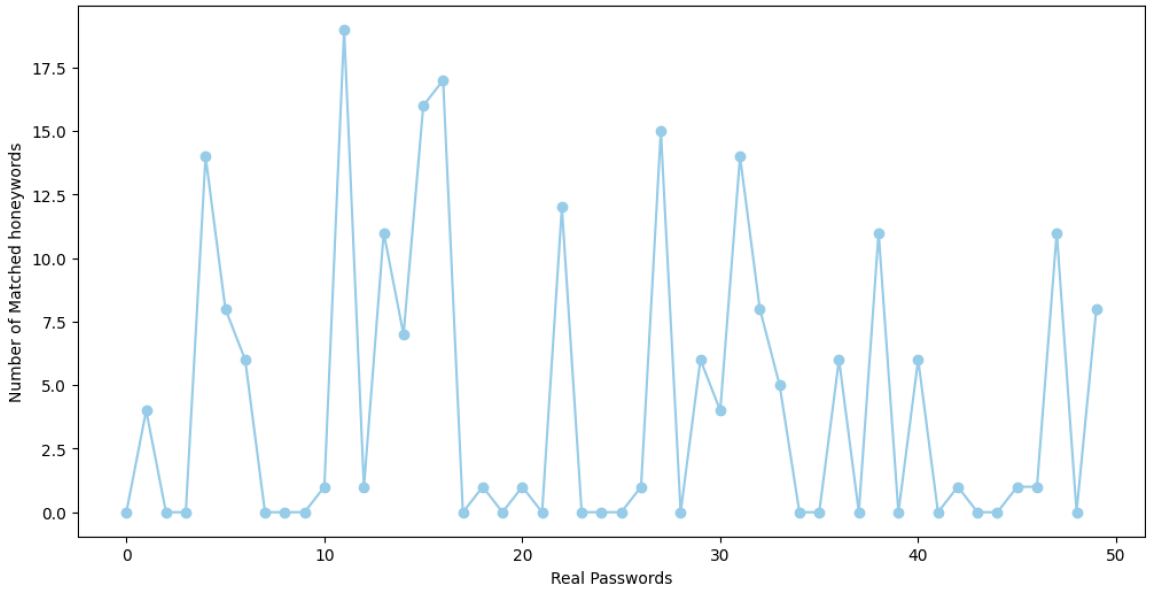

Figure 7.2: Indistinguishable Honeywords generated by GPT-4

policies. At the end we write a count of how many honeywords policies are similar to the real policy for each real password.

Example: [‘emperorpalpateen - 4’]

This above example tells that the real password “emperorpalpateen” has 4 matched honeywords.

Below graph shows the number of matched honeywords for the first 50 strong real passwords.

X-axis – represents the real passwords

Y-axis – represents the matched honeywords (value lies in between 0 to 19)

Seeing the figure 7.2, we can say that most of the real passwords have more than 5 matched honeywords.

Comparing the following graph with the one generated by Chunk-GPT3, we interpret that when compared to the Chunk-GPT3, GPT-4 has generated more honeywords that are similar to their real passwords.

7.3 Comparisons

In this section, I am going to draw comparisons of how many matched honeywords has each of the model produced.

Table 7.1 shows how many honeywords generated by Chunk-GPT3 have the same composition as their real password. The password 'a1069268@bofthew.com' has only '1' honeyword which has the same composition as the real password

Real passwords	**Number of Matched Honeywords**
emperorpalpateen	1
a1069268@bofthew.com	1
karpova.miss-olga	0
zholtovskaya90	0
vitalik3104slon	1

Table 7.1: Matched Honeywords generated by Chunk-GPT3

Table 7.2 shows how many honeywords generated by GPT - 4 have the same composition as their real password. The password 'a1069268@bofthew.com' has '14' honeyword which has the same composition as the real password. The same password had only 1 matched honeyword when generated by Chunk-GPT3. After this experiment, we can say that GPT - 4 model has showed significant increase of generating indistinguishable honeywords when compared to Chunk-GPT3.

Real Passwords	**Number of Matched Honeywords**
emperorpalpateen	4
a1069268@bofthew.com	14
karpova.miss-olga	19
zholtovskaya90	6
vitalik3104slon	11

Table 7.2: Matched Honeywords generated by GPT - 4

Chapter 8

Conclusion and Future Work

In this section, I will be discussing about some limitations and future work that can be done to this project for more improvements.

8.1 System Policy

Over the time, system administrators have created more sophisticated password creation policies to improve security. Even the users have become more security conscious to create better passwords which compile with system policies. In our experiments, we have used PCFG to create our own policies. In future, it would be great if we try to compare the honeywords with the system policies.

8.2 Typing errors

If the honeyword is very similar to the password, there's a chance that a legitimate user would type it by accident. By keeping the password and each generated honeyword at least a minimal distance apart, such problems can be handled with a high probability. We recommend calculating the "Levenshtein distance" (proposed by

"Vladimir Levenshtein") between the password and the honeywords. By calculating the quantity of insertions, deletions, or substitutions needed to change one string into another, the "Lavenshtein distance" is determined. It can be applied to the computation of string length differences. All kinds of human typing errors can be considered in this way.

To improve the creation of honeywords and create a more plausible and reliable false authentication method, this study investigates creative uses of current passwords. Also, we have proposed the use of Probabilistic Context-Free Grammar to improve the generated honeywords by using the length and number of substrings in existing real passwords. This PCFG concept in context with honeywords contributes to the development of password security and provides a framework for further research and development in this domain. After my experiments, with Chunk-GPT 3 and GPT -4 models, I can say that GPT - 4 has generated more indistinguishable honeywords from the real passwords when compared to Chunk-GPT 3.

Bibliography

[1] Akshima, Chang, D., Goel, A., Mishra, S., and Sanadhya, S. K. Generation of Secure and Reliable Honeywords, Preventing False Detection. *IEEE Transactions on Dependable and Secure Computing 16*, 5 (2019), 757–769.

[2] Buruk, O. O. Academic Writing with GPT-3.5: Reflections on Practices, Efficacy and Transparency. arxiv preprint arxiv:2304.11079, 2023.

[3] Chakraborty, N., Yamout, Y., and Zulkernine, M. The Tables Have Turned: GPT-3 Distinguishing Passwords from Honeywords. In *2023 IEEE Conference on Communications and Network Security (CNS)* (2023), pp. 1–5.

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

[5] Fezari, M., Al Dahoud, A., and Al-Dahoud, A. From GPT to AutoGPT: a Brief Attention in NLP Processing using DL. *ResearchGate* (04 2023).

[6] Hadi, M. U., qasem al tashi, Qureshi, R., Shah, A., amgad muneer, Irfan, M., Zafar, A., Shaikh, M. B., Akhtar, N., Wu, J., and Mirjalili, S. Large Language Models: A Comprehensive Survey of its Applications, Challenges, Limitations, and Future Prospects. Published in TechRxiv.

[7] Houshmand, S., Aggarwal, S., and Flood, R. Next Gen PCFG Password Cracking. *IEEE Transactions on Information Forensics and Security 10*, 8 (2015), 1776–1791.

[8] Huang, Z., Bauer, L., and Reiter, M. K. The Impact of Exposed Passwords on Honeyword Efficacy. arxiv preprint arxiv:2309.10323, 2023.

[9] 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.

[10] Li, Y., Wang, H., and Sun, K. A study of personal information in human-chosen passwords and its security implications. In *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications* (2016), pp. 1–9.

[11] Moea, K. S. M., and Winb, T. Selecting the Honeywords from Existing User's Passwords Using Improved Hashing and Salting Algorithm. *International Journal of Computer (IJC) 28*, 1 (2018), 133–142.

[12] Pagar, V. R., and Pise, R. G. Strengthening password security through honeyword and Honeyencryption technique. In *2017 International Conference on Trends in Electronics and Informatics (ICEI)* (2017), pp. 827–831.

[13] Shubham Sawant, Pratik Saptal2, K. L. K. G., and Kaur, P. R. Honeywords: Making Password Cracking Detectable 2018. *International Journal of Engineering Research and Advanced Technology (ijerat) (E-ISSN 2454-6135) 4*, 4 (4 2018), 01–06.

[14] Veras, R., Collins, C., and Thorpe, J. On semantic patterns of passwords and their security impact. In *NDSS* (2014), Citeseer.

[15] Veras, R., Thorpe, J., and Collins, C. Visualizing Semantics in Passwords: The Role of Dates. In *Proceedings of the Ninth International Symposium on Visualization for Cyber Security* (New York, NY, USA, 2012), VizSec '12, Association for Computing Machinery, p. 88–95.

[16] Wang, D., Cheng, H., Wang, P., Yan, J., and Huang, X. A Security Analysis of Honeywords. In *Network and Distributed System Security Symposium* (2018).

[17] Wang, X., Wang, D., Chen, X., Xu, R., Shi, J., and Guo, L. Improving Password Guessing Using Byte Pair Encoding. In *Nguyen, P., Zhou, J. (eds) Information Security. ISC 2017. Lecture Notes in Computer Science(), vol 10599. Springer, Cham* (10 2017), pp. 254–268.

[18] Weir, M., Aggarwal, S., De Medeiros, B., and Glodek, B. Password cracking using probabilistic context-free grammars. In *2009 30th IEEE symposium on security and privacy* (2009), IEEE, pp. 391–405.

[19] 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.

[20] Yu, F., and Martin, M. V. Honey, I Chunked the Passwords: Generating Semantic Honeywords Resistant to Targeted Attacks Using Pre-trained Language Models. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment* (2023), Springer, pp. 89–108.