Enhancing Password Security: Advancements in Password Segmentation Technique for High-Quality Honeywords

by

Satya Sannihith Lingutla

A Capstone Research Project report submitted to the School of Graduate and Postdoctoral Studies in partial fulfillment of the requirements for the degree of

Master of Information Technology Security in Artificial Intelligence in Security field

Faculty of Business and IT

University of Ontario Institute of Technology (Ontario Tech University)

Oshawa, Ontario, Canada

July 2023

CAPSTONE RESEARCH PROJECT REVIEW INFORMATION

Submitted by: **Satya Sannihith Lingutla**

Master of Information Technology Security in Artificial Intelligence in Security field

Project/Major Paper title:

Enhancing Password Security: Advancements in Password Segmentation Technique for High-Quality Honeywords

The Project was approved on **03 August 2023** by the following review committee:

Review Committee:

Research Supervisor:	**Dr. Miguel Vargas Martin**
Second Reader:	**Dr. Patrick Hung**

The above review committee determined that the 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

Passwords play a major role in the field of network security and play as a first line of defense against attackers who gain unauthorized access to the profiles. However, passwords are vulnerable to various types of attacks making it essential to ensure that they are strong, unique, and confidential. One of the major techniques that evolved over time to enhance password security is the use of honeywords that are decoy passwords designed to alert the administrator when a data breach has happened. The main goal of this project is to addresses one of the limitations of a honeyword generation technique, called Chunk-GPT3, by performing better password segmentation through a re-engineered chunking algorithm that maps digits into characters, and which would seem to lead to better honeywords. We justify our re-engineering method and generate honeywords that we compare to those generated by Chunk-GPT3. Nonetheless, after evaluating honeywords using the HWSimilarity metric, the results suggest that improved chunking does not necessarily lead to better honeywords in all cases.

Keywords: authentication, intrusion detection, honeywords, passwords, language models.

Author's Declaration

I hereby declare that this capstone project consists of original work authored by me. This is a true copy of the work, including all 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 scholarly research purposes. Additionally, I grant permission to the University of Ontario Institute of Technology (Ontario Tech University) to reproduce this work, either in whole or in part, by photocopying or other means, upon request from other institutions or individuals for scholarly research.I understand that my work may be made electronically available to the public.

Satya Sannihith Lingutla

Co-Authorship Statement

I would like to acknowledge Meher Vishwanath Nety as a co-author and contributor in section 3.2 of this work. His involvement was invaluable, and the contributions played a vital role in the research and analysis conducted for this chapter. The expertise and input greatly influenced the development of ideas, data collection, experimental design, and interpretation of results.

Acknowledgements

I would like to express my sincere gratitude to the following individuals for their invaluable support in the completion of this project. Their contributions have been instrumental in its success. First and foremost, I want to thank 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 Chunk GPT3 paper. Her groundbreaking research forms the foundation of this project, and I am grateful for her innovative insights.

I would like to express my deepest gratitude to Meher Vishwanath Nety for the unwavering support and invaluable contributions. His guidance and input have played a pivotal role in enhancing the quality and depth of this work.

Furthermore, I extend my heartfelt thanks to all the individuals, colleagues, friends, and family members who have stood by me throughout this journey. Your unwavering belief in my capabilities has been a constant source of inspiration, and I am truly grateful for your enduring support.

Thank you.

Statement of Contributions

The research presented in this project has been generously funded by the the Natural Sciences and Engineering Research Council of Canada (NSERC), funding reference number RGPIN-2018-05919 . Their support has been instrumental in enabling the successful completion of this research endeavor.

I hereby certify that I am the author of this work and that no part of this work has been published. I have used standard referencing practices to acknowledge ideas, research techniques, or other materials that belong to others. Furthermore, I hereby certify that I am the source of the creative works and/or inventive knowledge described in this document.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Passwords are one of the main fundamental components of network security and they are used to authenticate the user's identity to grant access to a particular resource or information. However, passwords are considered as a weak authentication mechanism because they are vulnerable to various attacks, such as dictionary attacks, brute force, and many more [8]. So, it is very crucial that passwords have to be strong, unique, and super confidential. A strong password means using unique words every time creating a new one and it has to fall between 8–12-character length by combing lowercase, uppercase, some special symbols, and digits [13]. Passwords are typically stored in file systems or databases to facilitate user authentication. However, storing passwords in plain text format poses a significant security risk. Therefore, it is common practice to store passwords in hashed form. A password hash is created by subjecting the password to a one-way mathematical procedure that transforms it into a unique string of characters. This ensures that even if the password database is compromised, the original passwords cannot be easily derived from the stored hash values. Even though taking all these measures, there are so many determined attackers that may employ various techniques to crack hashed passwords. Password cracking involves attempting to recover passwords by unconventional and often unethical means from stored or transmitted data within a computer system. Attackers may use powerful hardware, software, or precomputed databases of password hashes to accelerate the cracking process. To strengthen the security of password storage and detect potential breaches, the concept of honeywords can be employed. Honeywords are essentially decoy passwords that are associated with each user's account [13] [8] [16]. Alongside the actual password, multiple honeywords are created and stored in the password database. These honeywords are indistinguishable from the real password in terms of their hashed values.

If an attacker manages to compromise the hashed passwords' database and successfully reverses the hash function, they will encounter multiple password options, including the honeywords. Since honeywords are not legitimate passwords, their presence indicates unauthorized access attempts. When a user tries to log in and enters a honeyword, a ``silent alarm" is triggered, alerting the system administrators to the potential breach [8]. This allows for early detection and response to password cracking attempts. To implement honeywords effectively, a honeychecker is employed as an auxiliary server. The honeychecker is responsible for distinguishing between genuine passwords and honeywords during the login process. It verifies whether the entered password matches the actual password or is one of the honeywords. If a honeyword is detected, the honeychecker triggers an alarm to notify the appropriate personnel about the potential security breach [13] [8] [16].

There are many HGT's that generate honeywords normally in the database which can be compromised if the attacker knows personally identifiable information (PII) [16], and there are some HGT's that aim to create honeywords that are very similar to the original passwords and difficult to distinguish even when the attacker knows PII [14].Figure 1.1 provides a clear and easy visual representation of how the honeywords authentication mechanism works. It explains the operation of an authentication system with honeywords, which aims to enhance security in user logins. During registration, the system generates multiple honeywords alongside the user's chosen password and combines them in a random order

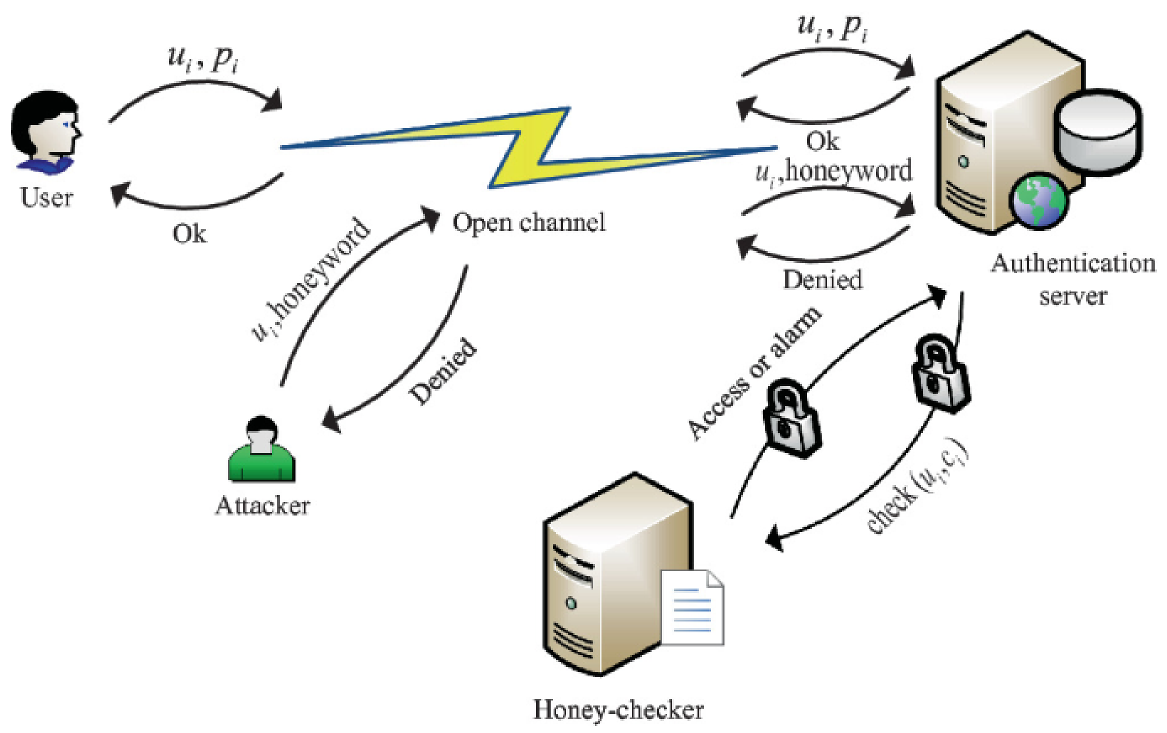


Figure 1.1: Honeywords authentication mechanism [12]

to form a set called (Wi). Each element in (Wi) is then hashed using a hash algorithm (H()) and stored, along with the user's ID (ui), as a tuple (ui, H(Wi)), by the identity authentication server [12]. Additionally, the server sends the user ID and the index (ci) of the actual password within (Wi) to the Honey-checker server for storage. During login attempts, the identity authentication server hashes the entered password (pi) and checks if the resulting hash (H(p_i)) matches any of the stored hashes in H(Wi). If a match is found, it suggests the user might have entered either the correct password or a honeyword. To verify further, the Honey-checker server compares the index (c_i) of H(p_i) in H(Wi) with the stored index (ci). If they match, the user is considered authenticated and can log in successfully. However, if the indices do not match, indicating the use of a honeyword, the system raises an alarm to detect potential unauthorized access [12].

One of the most successful HGT that generates honeywords by using a language model is Chunk GPT3 [16]. This is the first generative language model that is robust to targeted attacks. This report mainly focuses on how the quality of Honeywords changes if we made some changes to the password segmentation module

1.1 Our Contribution

We are the first to address the limitations in the Chunk-GPT3 method and re-engineer the chunking algorithm. We propose two different modules for the algorithm named ``Mapping Digits to Alphabets” and ``Removal of Digits” for better chunking. The primary purpose of the ``Mapping Digits to Alphabets" module is to identify any digits present within a given password. Once identified, these digits are replaced with corresponding letters, resulting in a new password referred to as the ``mapped password." This mapping ensures that the resulting password remains secure while obfuscating the presence of any numerical characters. In parallel, the ``Removal of Digits" module focuses on detecting the presence of digits within a password. Instead of replacing them with letters, this module generates an alternative version of the password called the ``text password" by simply removing the digits.

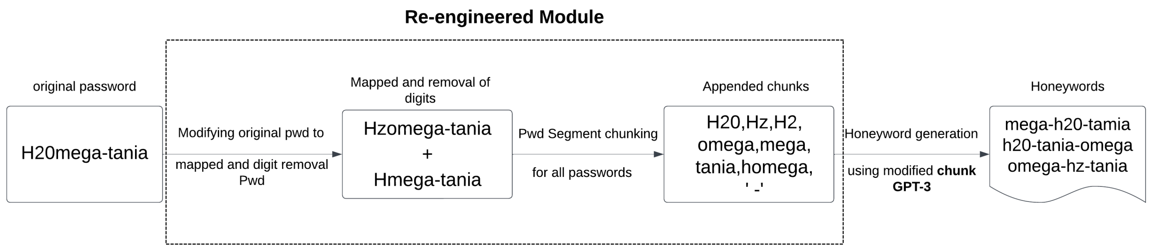

Re-engineered Module
original password
H20mega-tania
Modifying original pwd to
mapped and digit removal Pwd
Mapped and removal of digits
Hzomega-tania + Hmega-tania
Pwd Segment chunking
for all passwords
Appended chunks
H20,Hz,H2, omega,mega, tania,homega, ' -'
Honeyword generation
using modified chunk GPT-3
Honeywords
mega-h20-tamia
h20-tania-omega
omega-hz-tania

Figure 1.2: Honeywords generated using our re-engineered method

These modules are executed only when the original password contains exactly two digits or fewer, because sometimes one or two digits doesn’t make a meaningful chunk. In cases where this condition is not met, the normal chunking process is applied. By employing these modules, we create chunks from the modified passwords and combine them to generate ``honey words." The figure 1.2 clearly shows the process of the re-engineered module. We evaluated the honeywords' similarities with their original password by using the HWSimilarity metric proposed in [16]. We also evaluated the HWSimilarity of the Chunk-GPT3 with our model for a better understanding of the results and how the changes in the chunking algorithm performed.

1.2 Project Organization

The rest of the paper is organized as follows: **Section 3** represents the methodology by showing how we made changes to the algorithm.**Section 4**provide the minimum system configurations required for successfully running the project. **Section 5**outlines the results of the model and comparison between the Chunk-GPT3. **Section 6** discusses some limitations of our work and suggests some future work. Finally, **Section 7** concludes our research.

Chapter 2

Background

In the field of security, there exist advanced techniques for generating honeywords that resemble genuine passwords. These honeywords are combined with authentic passwords to create sweet words, thereby improving the security of password-based authentication systems [13]. While exploring multiple papers, researchers have developed various Honeyword Generation Techniques (HGTs) by considering multiple factors. Some of the important HGTs are:

2.1 Honeywords Generated by Chaffing

Chaffing is a technique used in computer security where a large amount of false data is added to the system that consists of original data in order to confuse the attacker. Here we add a large number of decoy passwords to the original one. Three main types are:

2.1.1 Chaffing-by-tweaking:

In this method, we are making slight modifications to the original password. These modifications can include adding or changing a letter, digit, or symbol, or adding a prefix or suffix to the password [13] [8].Mainly chaffing-by-tweaking is achieved by two methods:

2.1.1.1 Chaffing-by-tail-tweaking

In this method, the tail (the last few characters) of the original password is modified at specific locations. Let's consider an example where the user's password is ``TU-9g73". We choose a value for ``t" (the number of tail characters to be modified) and ``k" (the number of honeywords to generate) [13]. For instance, if we set ``t" as 4 and ``k" as 5, the list of honeywords (Wi) generated through tail tweaking could be:

TU-2f45, TU-6h23 , TU-0b12 , TU-8j60 , TU-5l63

In each honeyword, the tail characters of the original password have been modified while keeping the initial part unchanged. This technique adds confusion for potential attackers who might try to guess the password by observing patterns in the tail.

2.1.1.2 Chaffing-by-tweaking-digits

In this method, the focus is on modifying the digits present in the original password. If the password has fewer than ``t" digits, non-digit places can be used as needed [13]. Let's consider an example where the original password is ``762@jupiter". We set ``t" as 3, indicating that we want to modify the last three digits of the password. Here are some honeywords generated using chaffing-by-tweaking-digits:

934@jupiter , 815@jupiter , 256@jupiter , 157@jupiter , 658@jupiter

In these examples, the last digits of the original password have been tweaked to create different honeywords. This technique leverages the fact that many users incorporate numbers in their passwords and aims to confuse attackers by introducing variations in the digits.

2.1.2 Chaffing-with-a-password-model

In this technique, a statistical model of the passwords used by the system's users is created, based on factors such as password length, character types, and frequency of use [13] [8].

2.1.2.1 Simple model

In this approach, a probabilistic model of actual passwords is used to generate honeywords. The model can be based on a published list of passwords (such as L, which contains hundreds or thousands of passwords) and may consider other factors as well. For example, let's assume we have the following published password list: ``iloveyou", ``monkey", ``sunshine". Using this list as the basis for the password model, honeywords can be generated and they are :

iloveyou123 , monkey456 , sunshine789

In this example, the honeywords are created by using the simple password model and adding numeric values (e.g., 123, 456, 789) to the base passwords from the published list. This technique leverages the patterns and characteristics observed in real passwords to generate plausible honeywords.

2.1.2.2 Modeling syntax

In this technique, honeywords are created based on the same syntax as passwords. The original password is divided into tokens, where each token represents a separate syntactic component such as a letter, digit, or collection of symbols. For example, consider the original password ``super90man". It can be divided into tokens as follows: W5 (word of length 5) | D2 (digit of length 2) | W3 (word of length 3). The generated honeywords by this model are :

moons64hat , cats12car , jump87fox

In this example, honeywords are formed by substituting each token with randomly selected values corresponding to the token type. This technique maintains the syntax and structure of passwords while creating variations that still adhere to the expected format. By using a simple model or modeling syntax, honeywords can be generated based on the characteristics and patterns observed in real-world passwords. This adds complexity and confusion for attackers attempting to guess or crack the passwords.

2.1.3 Chaffing-by-fasttext

In this technique, it generates honeywords by converting passwords to vectors using the fasttext and allocating the closest k-1 nearest neighbors based on the cosine similarity calculator. So here fasttext is trained on the real-world dataset of passwords for better semantics [14] [4] [16]. Suppose we have the following original password: ``SecurePass123".

Training fastText: We train a fastText model on a real-world dataset of passwords to learn vector representations that capture the semantics and patterns of password usage.

Converting password to vectors: We convert the original password ``SecurePass123" to a vector representation using the trained fastText model. This vector captures the semantic meaning of the password.

Finding nearest neighbors: Using cosine similarity, we identify the k-1 nearest neighbors to the vector representation of the original password. These neighbors are selected based on their similarity to the semantic vector of the original password.

Generating honeywords: Finally, we generate honeywords by replacing the original password with the identified nearest neighbors. These honeywords will have similar semantic characteristics to the original password but differ slightly in their specific values when we compare it to the original password.

Original Password: ``SecurePass123"

Trained fastText model: Captures semantics of passwords

Generated Honeywords (k = 4):

SecurePassword321 , SecurePassABC , SecurePassXYZ , SecurePass789

In this example, the original password ``SecurePass123" is converted into a semantic vector using the trained fastText model. Then, based on cosine similarity, the nearest neighbors to the original password are identified. These nearest neighbors are used to generate honeywords. The honeywords maintain the semantic meaning of the original password but provide slight variations, such as different endings, replacements, or alterations. This technique increases the complexity for potential attackers and makes it more challenging to determine the actual password.

2.2 HGT By Juels and Rivest

This is the first Honeyword generation Technique [13] [8] that ever created and goes by the name Honeychecker [8]. So, a honeychecker is an approach where it creates a database of honeywords by generating so many fake passwords that look like original passwords but are distinct enough to make an attacker confused. The honeywords are then scattered at random in the password file with the true password. The system verifies the login attempt's validity by comparing the input password to the corresponding honeyword in the password file. If the password submitted matches a honeyword, the system flags the attempt and alerts the user [8].

In honeychecker the honeywords are generated depending on two prospects one is the Legacy-UI procedure and the second one is Modified UI procedures [13] [8]. In the legacy-Ui procedure, it doesn't give any information to the user about honeywords, and it involves two types of chaffing techniques, Chaffing-by-Tweaking and Chaffing-by-password-model

[13] [8]. In this, it tries to modify the password by characters and digits to generate honeywords. In the modified-Ui procedure, it informs the user about honeywords that they are used to increase password security. It influences the user to choose the password. It is similar to the chaffing-by-Tweaking technique, where it influences the user to add a tail to the chosen password to make it more difficult to guess. Another method is also there where it enters 3 random digits into the password and informs the user about it. All these methods are evaluated by flatness curve and DoS resistance. Of all these methods the one which saved users by showing great resistance to attack and flatness is the Take a Tail suggestion method which informs the user to add some digits or make some suggestions to him to his password [8].

2.3 HGT using a discrete Salp Swarm Algorithm (DSSA)

The DSSA is a population-based optimization algorithm that stimulates the behavior of salps, a type of marine animal that navigates by changing its shape and swimming direction [1]. It generates a set of honeywords by perturbing the original password based on the movement of the salps in the algorithm. The salps are divided into groups known as leaders and followers where leaders are the original passwords, and the followers are the honeywords in descending order based on their strengths. The model is tested for various attacks to measure its effectiveness like brute force and dictionary attacks. The model succeeded in defending against these attacks and also improving the security of passwords [1].

2.4 HoneyGAN

HoneyGan is an HGT used to generate honeywords based on the GNPassGAN's generator [14]. It tries to generate fake passwords that look like original passwords. Here we will see how to generate honeywords with GNPassGAN, a generative adversarial network (GAN) model. The initial stage, step-1 is to train GNPassGAN on a password corpus for 200,000 iterations in order to comprehend the password pattern in the training dataset. In step-2, The GNPassGAN generator is then used to generate a file named F containing 50,000 bogus passwords as honeyword candidates, which are then kept away from the authentication system in a safe area. Step-3 compares each user's genuine password to each of the fake passwords in F and computes text similarity scores. This is accomplished by employing the bag-of-words (BoW) approach to transform each password into a vector and determining the cosine similarity between two passwords [14].

Finally, in step 4, honeywords for a genuine password are assigned to the k-1 most similar fake passwords in F. After successful full generation the HoneyGan HGT is compared to some of the HGT, they are Chaffing-by-tweaking and chaffing-by-fasttext [14] [16]. It's evaluated by using ASR and average internal similarity. The ASR (Attack Success Rate) for honey words generated by HoneyGan has around 60% success rate and 68% and 90% when the honey words are generated by fasttext and tweaking and given 20 attempts for the user are allocated. HoneyGan's average internal similarity is very high (81.9%) when it is compared to fasttext (26.2%) and tweaking (62.7%). Here HoneyGan outraged both of them with better scores and also generated honeywords that almost look like the original passwords. The measurement principles are the average internal similarity and the attack success rate which it shows how good they are in defending against the attacks. Although the success rate is high there are some limitations like the inappropriate password setting where the semantics [10] are not discussed properly.

2.5 Chunk-GPT3

Chunk GPT3 is a Honeyword generation technique that generates honeywords by using the GPT model. The biggest challenge of creating an HGT is to generate honeywords that are resistant to targeted attacks [15] [16]. Targeted attacks are a significant cybersecurity concern as attackers often leverage users' personally identifiable information (PII) to guess their passwords, thereby increasing the risk of unauthorized access to user accounts. The problem is compounded because of the ongoing data breaches which make a large amount of PII and passwords easily accessible to attackers. If the honeywords that are generated by HGT are not considering PII and if the attacker knows the PII and possibility of identifying the password is very high. So, Chunk GPT3 is HGT which generates the honeywords that are resistant to the targeted attacks [1] [16]. Initially, the password strength is calculated by using the zxcvbn library [11]. Then the passwords are sorted according to the zxcvbn strength [16] and then divided into 2 datasets of strong and weak passwords. The data sets are gone through the Pwd segment chunking algorithm to generate password chunks for the honeywords generation. The main advantage of this chunking algorithm is it generates chunks that consist of original password segments only. GPT3 is used to generate honeywords that are completely robust to targeted attacks, so we need to generate the honeywords by giving prompts to it [16]. The three main factors for generating good honeywords are the prompt, the temperature, and the examples we give to the model (chunks generated). The temperature is a numeric value that may be 0 or 1 and that effectively regulates the model's degree of confidence in generating a good prediction, it's highly recommended to always keep the temperature to 1 for good results [16]. So, the password is segmented into chunks, and they are given to the model as input like "Please derive three passwords that are similar to "Elena1986@327" and contain "Elena", "1986" and "327" [16]. The figure 2.1 demonstrates the honeywords generated by the model when the above prompt is

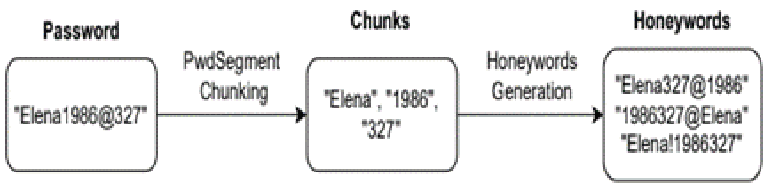


Figure 2.1: Honeywords generated by GPT3 [16].

given:

The evaluation of this was done by using flatness and success-number-graphs [16]. The comparison of Chunk-GPT3 with other HGT's chaffing-by-tweaking and chaffing-by-fasttext. The Chunk-GPT3 outplayed both of them by showing tremendous results by having an 85% of honeywords similarity score [16]. The main limitation of this model is the importance of having correct chunks in passwords. If the password is not segmented correctly, the attacker can easily get access to the password easily.

Chapter 3

Methodology

In ChunkGPT3 model the chunks are created from the password with the help of a common technique named as PwdSegment. PwdSegment is an algorithm that uses Byte-Pair-Encoding (BPE) to create chunk vocabularies from plain-text passwords. BPE is a widely used technique in NLP for sub word segmentation. PwdSegment enhances BPE by incorporating the configurable parameter of average length (avg len) to control the segmentation granularity. It counts character pairs and stops merging when the resulting chunk vocabulary meets or exceeds the specified threshold length. The algorithm iteratively merges the most common pair of tokens until the average length of the chunk vocabulary reaches the threshold. PwdSegment provides a flexible approach to generating chunk vocabularies for password analysis. But the PwdSegment doesn't consider some methods to provide better quality chunks. Suppose if we consider an example h2omega-tania, it only generates ``h2o",`` mega",``-", ``tania" as chunks. But there is possibility of generating ``omega", ``home". Due to its limitation, it not generating them which will limit the model to those particular chunks. So, we developed a model where it helps the PwdSegment algorithm to generate better chunks for increasing the quality. This code update aims to enhance the existing Password Chunking code used for analyzing password fragments. The upgraded system introduces several new capabilities, including digit-to-alphabet mapping, digit deletion, and analysis of plain text passwords. These modifications expand the code's functionality beyond its initial focus on analyzing strong and weak passwords. The effects of these code changes will be detailed in the following report.

3.1 Mapping Digits to Alphabets

Introducing a new functionality called **Digit-to-Alphabet Mapping** allows for the analysis of passwords that employ such mappings. This feature aims to associate individual digits in passwords with their corresponding alphabetic characters. The mapping follows these rules:

Digit	**Letter**
1	i
2	z
3	e
4	a
5	s
6	g
7	t
8	b
9	g
0	o

Table 3.1: Digit-Letter mapping

The main objective is to map one or two consecutive digits to their corresponding alphabetic characters within a password, while passwords containing three or more consecutive numbers remain unchanged. These transformed passwords are referred to as **mapped passwords**. To test the mapping operation, we can consider three scenarios from the following Table 3.2 . In the first scenario, passwords like ``h0mega-tania" have only one digit, which is mapped to its corresponding alphabetic character. In the second scenario, passwords such as ``h20mega-tania" contain two consecutive digits, which are mapped accordingly.

In both these cases, the passwords undergo the mapping process. However, in the third scenario, passwords like ``h123mega-tania" have three consecutive digits, and they remain unchanged. This is because numbers in passwords with at least one number of three or more digits, such as ``h123mega-tania," are retained in their original form as they may potentially contain personally identifiable information (PII). Similarly, if a password, like ``h123mega-98tania," contains three consecutive digits in one position and two consecutive digits in another, it remains unchanged. By considering these scenarios, we can observe how the mapping operation applies to passwords with varying numbers of consecutive digits.

Original Password	**Mapped Password**
h0mega-tania	homega-tania
h20mega-tania	hzomega-tania
h123mega-tania	h123mega-tania

Table 3.2: Password mapping

3.2 Removal of Digit:

In addition to mapping single digits to their corresponding characters, as explained earlier, we have also implemented a modification to passwords by removing single digits. This modification allows us to eliminate individual numerical characters, resulting in plain text passwords that are devoid of any numeric components. By removing the single digit, we can analyze passwords independently, without considering the presence or significance of the numeric element. The resulting password, after removing the single digit, is referred to as a **Text Password**. This approach provides the advantage of evaluating and examining the password's security without accounting for the presence of a single numerical digit. For example, let's consider the original password ``h2mega-tania". By applying this method, the password is transformed into the corresponding text password ``hmega-tania" (in addition to the Digit-to-Alphabet Mapping described earlier). This transformation allows us to assess the password's security while disregarding the single numerical digit. By incorporating this modification into our analysis, we gain a more comprehensive understanding of password security and can effectively evaluate the strength of passwords by focusing solely on the textual components.

3.3 Password Chunk Analysis

Our re-engineered code performs chunk analysis on three different types of passwords, they are original passwords, mapped passwords, and text passwords. The following steps were taken for each type:

3.3.1 Original Passwords

The process of extracting password chunks and enhancing the analysis of password strength involves the following steps. First, the original passwords are retrieved from two CSV files: ``strong_pw_1000.csv" containing 1000 zxcvbn-strong passwords and ``weak_pw_1000.csv" containing 1000 zxcvbn-weak passwords.To facilitate a more accurate assessment of password strength, a new column called ``pw_chunks" is added to these CSV files. This column is populated with password chunks extracted using the Password Strength Meter (PSM) library [7] [9]. By breaking down the passwords into smaller chunks, we gain a deeper understanding of their composition and complexity.

In addition to the ``pw_chunks" column, another column called ``num_pwchunks" is added to count the number of password chunks present in each password. This provides valuable information for evaluating the complexity and structure of the passwords. Utilizing the ``pw_chunks" and ``num_pwchunks" columns, new CSV files are created, namely ``strong_pw_chunks_1000.csv" and ``weak_pw_chunks_1000.csv". These files contain the original passwords along with their corresponding password chunks, as well as the number of chunks in each password . By following this process, we enhance the analysis of password strength by incorporating the extracted password chunks and their associated information. This approach ensures that even if the original CSV files were compromised, the actual passwords remain secure. Furthermore, the availability of the ``pw_chunks" column enables a more detailed examination of password complexity and aids in developing stronger security measures.

3.3.2 Mapped Password

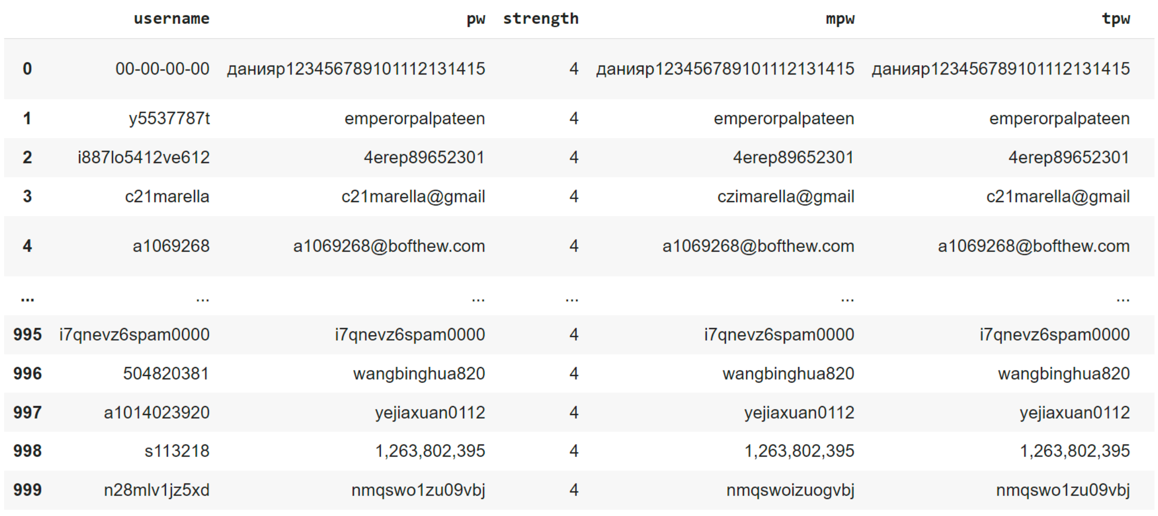
Our re-engineered method introduces the concept of transformed passwords by applying the Digit-to-Alphabet mapping to the original passwords. This mapping replaces digits in the passwords with their corresponding alphabetic characters, following a predefined mapping scheme. The resulting mapped passwords are stored in a new column called ``mpw". To analyze the strength and composition of these mapped passwords, we perform password segmentation analysis, similar to what we do for the original passwords. This analysis involves breaking down the passwords into smaller segments and identifying patterns or common substrings within them. In order to capture this information, two additional columns, ``mpw_chunks" and ``num_mpchunks", are added to the data frame. The ``mpw_chunks" column stores sets of segments extracted from the mapped passwords, while the ``num_mpchunks" column records the count of these segments for each password [6]. By conducting password segmentation analysis on the mapped passwords, our code aims to gain insights into their composition and potential vulnerabilities. This additional analysis provides a deeper understanding of the password strength and aids in identifying common patterns or weaknesses in the transformed password dataset. By incorporating these new columns and conducting password segmentation analysis, our code provides a comprehensive assessment of both the original passwords and the mapped passwords. This allows for a thorough evaluation of the password strength and assists in identifying areas for improvement in terms of security and composition.

3.3.3 Text Password

In our re-engineered approach, we have introduced a new approach to handle plain text passwords by utilizing the **Removal of Digits** method described above. This method involves processing the original passwords and removing any single digit present in them. This transformation generates plain text representations of the passwords, which are then stored in a newly created column called ``tpw" within the data frame. Similar to the analysis performed on the original passwords and mapped passwords, our code conducts password chunk analysis on the plain text passwords. This analysis entails dividing the plain text passwords into smaller chunks and examining patterns or common substrings within them. To facilitate this analysis, we have added two additional columns, namely ``tpw_chunks" and ``num_tpchunks", to the data frame. The ``tpw_chunks" column contains sets of chunks extracted from the plain text passwords, while the ``num_tpchunks" column keeps track of the count of these chunks for each password. These additions enable the identification of common patterns or substrings within the plain text passwords, providing valuable insights into password composition and potential vulnerabilities [2]. By incorporating this new approach and conducting password chunk analysis on the plain text passwords, our code offers a comprehensive evaluation of the original passwords, mapped passwords, and plain text passwords. This comprehensive analysis aids in identifying common patterns, weaknesses, and areas for improvement in terms of password security.

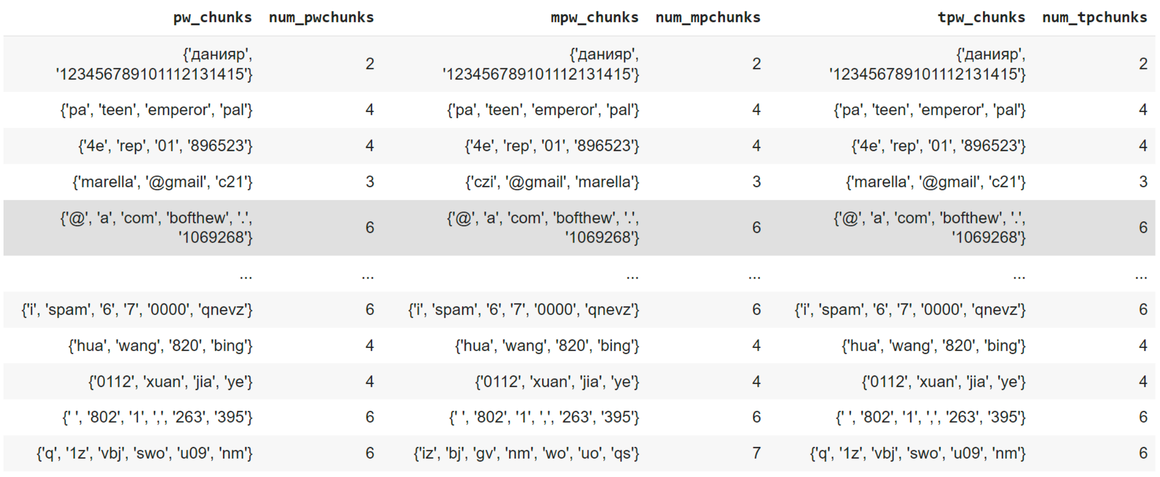
3.4 Appending Chunks

In our re-engineered method, all the password chunks obtained from the original passwords, mapped passwords, and plain text passwords are consolidated. Duplicate chunks are removed to ensure a comprehensive analysis of the combined chunk sets. The code incorporates the ``chunks" column, which combines the chunk sets from different password representations, allowing for comparison and analysis. Additionally, the code includes a function called ``get_set_length" to calculate the length of each chunk set. This function converts the chunk sets from string representations to actual sets, calculates their lengths using the ``len" function, and stores the results in the ``num_chunks" column. These data provides important insights into the complexity and diversity of passwords in the dataset. By counting the chunks in each password chunk set, we can better understand the password strength. We have divided the data into two figures, Figure 3.1 and Figure 3.2, for clarity. These additions allow easy comparison and analysis of password representations, enabling a comprehensive evaluation of password strength.



	username	pw	strength	mpw	tpw
0	00-00-00-00	данияр123456789101112131415	4	данияр123456789101112131415	данияр123456789101112131415
1	y5537787t	emperorpalpateen	4	emperorpalpateen	emperorpalpateen
2	i887lo5412ve612	4erep89652301	4	4erep89652301	4erep89652301
3	c21marella	c21marella@gmail	4	czimarella@gmail	c21marella@gmail
4	a1069268	a1069268@bofthew.com	4	a1069268@bofthew.com	a1069268@bofthew.com
...	...	...	...	...	...
995	i7qnevz6spam0000	i7qnevz6spam0000	4	i7qnevz6spam0000	i7qnevz6spam0000
996	504820381	wangbinghua820	4	wangbinghua820	wangbinghua820
997	a1014023920	yejiaxuan0112	4	yejiaxuan0112	yejiaxuan0112
998	s113218	1,263,802,395	4	1,263,802,395	1,263,802,395
999	n28mlv1jz5xd	nmqswo1zu09vbj	4	nmqswoizuogvbj	nmqswo1zu09vbj

Figure 3.1: First half data frame of the columns related to original, mapped, and text password chunks of strong passwords.



pw_chunks	num_pwchunks	mpw_chunks	num_mpchunks	tpw_chunks	num_tpchunks
{'данияр', '123456789101112131415'}	2	{'данияр', '123456789101112131415'}	2	{'данияр', '123456789101112131415'}	2
{'pa', 'teen', 'emperor', 'pal'}	4	{'pa', 'teen', 'emperor', 'pal'}	4	{'pa', 'teen', 'emperor', 'pal'}	4
{'4e', 'rep', '01', '896523'}	4	{'4e', 'rep', '01', '896523'}	4	{'4e', 'rep', '01', '896523'}	4
{'marella', '@gmail', 'c21'}	3	{'czi', '@gmail', 'marella'}	3	{'marella', '@gmail', 'c21'}	3
{'@', 'a', 'com', 'bofthew', '.', '1069268'}	6	{'@', 'a', 'com', 'bofthew', '.', '1069268'}	6	{'@', 'a', 'com', 'bofthew', '.', '1069268'}	6
...	...	...	...	...	...
{'i', 'spam', '6', '7', '0000', 'qnevz'}	6	{'i', 'spam', '6', '7', '0000', 'qnevz'}	6	{'i', 'spam', '6', '7', '0000', 'qnevz'}	6
{'hua', 'wang', '820', 'bing'}	4	{'hua', 'wang', '820', 'bing'}	4	{'hua', 'wang', '820', 'bing'}	4
{'0112', 'xuan', 'jia', 'ye'}	4	{'0112', 'xuan', 'jia', 'ye'}	4	{'0112', 'xuan', 'jia', 'ye'}	4
{' ', '802', '1', ',', '263', '395'}	6	{' ', '802', '1', ',', '263', '395'}	6	{' ', '802', '1', ',', '263', '395'}	6
{'q', '1z', 'vbj', 'swo', 'u09', 'nm'}	6	{'iz', 'bj', 'gv', 'nm', 'wo', 'uo', 'qs'}	7	{'q', '1z', 'vbj', 'swo', 'u09', 'nm'}	6

Figure 3.2: Second half data frame of the columns related to original, mapped, and text password chunks of strong passwords.

3.5 Honeywords Generation Using Modified PwdSegment Chunks

In our approach, we leverage the GPT3 model to generate honeywords using the re-engineered password chunks. To generate the honeywords, we employ the same prompts as described in [16]. These prompts consist of the original password, its corresponding chunks, and a length limit. By utilizing the GPT3 model, we aim to generate honeywords that are highly resilient against targeted attacks. We achieve this by incorporating semantic chunks into the generation process. The prompts provided to the GPT3 model guide it in producing honeywords that closely resemble the original passwords while adhering to the specified length limit. The quality and effectiveness of the honeywords generated depend on how well we train the GPT3 model to produce honeywords that exhibit similar characteristics to the original passwords. This training process ensures that the generated honeywords possess the desired attributes, making them effective decoys against potential attackers. By employing the GPT3 model and utilizing the re-engineered chunks, we aim to generate honeywords that enhance the security of user passwords and mitigate the risks associated with targeted attacks.

Chapter 4

System Configuration

4.1 Operating System

The experiments were conducted on a stable and familiar Windows 10 operating system, providing a user-friendly environment for the researcher. Hardware Configurations:

4.2 Hardware Configurations

The system was equipped with 8 GB of RAM, which is sufficient for most natural language processing tasks and moderate-sized datasets, ensuring smooth performance during experiments. The thesis project utilized a 256 GB solid-state drive (SSD) for faster data access and overall improved system performance. The SSD's fast read and write speeds contributed to quicker processing of large NLP datasets.

4.3 Software and Libraries

All the experiments were performed using Google Collaboratory, a cloud-based Jupyter notebook environment. Google Collaboratory provides access to GPU and TPU resources, making it suitable for training deep learning models, including language models. It allowed the researcher to leverage powerful hardware resources without the need for expensive local hardware. Google Collaboratory mostly comes with pre-installed with commonly used Python libraries.The versions of the libraries that are required for this project and some of the installed libraries versions are

Library	Version
Python	3.10.6 (main, May 29 2023, 11:10:38) [GCC 11.3.0]
torch	2.0.1+cu118
openai	0.27.8
fasttext	0.9.2
matplotlib	3.7.1
wordcloud	1.8.2.2
numpy	1.22.4
pandas	1.5.3
seaborn	0.12.2
zxcvbn-python	4.4.24
sentence_transformers	2.2.2
ckl_psm	1.2

Table 4.1: Libraries and versions

4.4 Web Browser

Google Collaboratory is accessible through web browsers, and for optimal performance, the researcher used the latest version of Google Chrome. This ensured compatibility and maximized the efficiency of the Collaboratory environment.

4.5 Processor

The system was equipped with an Intel i5 5th generation processor. While it may not be the latest generation, this processor is capable of efficiently handling most NLP tasks, ensuring smooth execution of experiments, because most of the working part is done in Google Collaboratory which works completely virtual.

4.6 API Subscription

The experiments utilized the OpenAI API Subscription to generate words using the InstructGPT - Davinci language model. The specific version of the model used in the experiments is not known. The OpenAI API provides access to cutting-edge language models for natural language understanding and generation.

Chapter 5

Results and Evaluation

To evaluate the quality of our re-engineered honeywords, we utilize the HWSimilarity metric proposed in [16]. This metric leverages the power of MPNet, a pre-trained language model, to transform honeywords and passwords into vector representations. By calculating the cosine similarity between each honeyword vector and its corresponding real password vector, we obtain a comprehensive assessment of the semantic similarity between honeywords and genuine passwords. We compare the performance of our modified Chunk-GPT3 approach with the original Chunk-GPT3 by examining examples of generated honeywords and analyzing the overall similarity scores for both weak and strong password datasets. Table 5.1 presents a comparison of honeywords generated by Chunk-GPT3 and modified Chunk-GPT3, while Table 5.2 displays the corresponding HWSimilarity scores.

Original Password	**Chunk-GPT3**	**Modified Chunk GPT3**
h2omega-tania	tania-home123 Tania@home5 home!tania12	mega-h20-tania h20-tania-omega omega-hz-tania
0000_mila_0000	1111_mila_0000 0000MILA0000 0000@Mila@0000	000-MILA-0000 0000-Mila-0000 1111-Mila-1100

Table 5.1: Honeyword generation techniques

The findings reveal that our modified Chunk-GPT3 approach outperforms Chunk-GPT3 specifically for zxcvbn-weak passwords. This suggests that our re-engineered chunking technique offers greater benefits in terms of generating honeywords that closely resemble weak passwords.By evaluating the honeywords using the HWSimilarity metric, we gain insights into their semantic similarity to genuine passwords. The comparison of honeywords generated by different approaches allows us to measure their effectiveness in capturing personally identifiable information (PII). Notably, the honeywords generated by our modified Chunk-GPT3 approach exhibit a high degree of similarity to the original passwords, making it challenging to distinguish between them. However, it is important to note that strong passwords, which exhibit higher randomness, tend to yield lower similarity scores compared to weak passwords. The accuracy of chunk generation plays a more significant role in the case of weak passwords, as our modified approach produces more precise chunks compared to the regular PwdSegment technique.

	Re-engineered Chunk GPT3	**Chunk-GPT3**
Zxcvbn-strong	0.8409	0.8525
Zxcvbn-weak	0.9048	0.8367

Table 5.2: Comparison of re-engineered chunk GPT3 and Chunk-GPT3

Through this evaluation, we gain valuable insights into the quality and effectiveness of our re-engineered honeywords. The superiority of our modified Chunk-GPT3 approach, particularly for weak passwords, demonstrates its potential in enhancing the security of password systems.

Chapter 6

Future Work

When analyzing passwords, it is vital to be attentive to patterns and information that may be present, such as dates of birth (DOB) or random numbers. These patterns can provide valuable insights into potential vulnerabilities or risks associated with the passwords. Let's delve into the process of identifying and handling passwords that contain DOB information or random numbers.

Passwords that incorporate DOB information, such as ``12tania1995," necessitate careful scrutiny to identify and mitigate any potential connections to birth dates and years. In the given example, ``12" represents the birth date, while ``1995" signifies the birth year. To detect such passwords, one can search for four-digit numbers within the range of 1900 to 2100 to identify potential birth years. Additionally, when considering dates, a range of 1 to 31 can be utilized. It is worth noting that single-digit dates, like ``2," may also be expressed as ``02." By employing these techniques, it becomes possible to identify passwords that may contain DOB information and evaluate their potential risks [10].

However, passwords like ``12tania95" pose a challenge when the birth year is mentioned in a two-digit format. Mapping the digits into alphabetic characters to create password chunks becomes complex in such cases. Consequently, the analysis of passwords with two-digit birth years may require more sophisticated approaches to ensure accurate identification and handling of potential vulnerabilities.

Passwords that feature random numbers, such as ``Tania4682," should generally not be modified or transformed. Random numbers within passwords often relate to personal information, such as the last four digits of a phone number, house number, or car number. Altering these random numbers may inadvertently remove critical information or compromise the security of the passwords. Therefore, exercising caution when dealing with passwords containing random numbers is crucial, as these numbers can hold significant meaning for the individuals using them [3].

In certain instances, passwords like ``07tania" might raise concerns due to the presence of ``07," which could be interpreted as the birth date or birth month. Modifying or mapping such passwords carries inherent risks, as the numeric component may be personally significant or indicate sensitive information. Consequently, extra consideration should be given to the potential implications of altering passwords in these particular scenarios.

In summary, when examining passwords, it is of utmost importance to be mindful of patterns associated with dates of birth and random numbers. Identifying and handling passwords that contain DOB information necessitates careful analysis of birth years and dates. Passwords containing random numbers should be treated with caution, recognizing the potential associations with personal information [5] [9] [10]. Moreover, passwords with ambiguous numeric components, such as birth dates or birth months, should be approached with extra caution, considering the potential risks involved in modifying them.

Chapter 7

Conclusion

Passwords play a critical role in security, but safeguarding them requires resilient measures against targeted attacks. Deep learning and natural language processing have introduced new possibilities for generating honeywords that closely mimic the original passwords, even when attackers possess users' Personally Identifiable Information (PII). Chunk-GPT3, a technique for generating honeywords, shows promise but has a limitation in password segmentation [16], which could be exploited by attackers. However, modifications to the PwdSegment technique have proven effective for weak password datasets, though their impact on stronger datasets may be limited. Despite imperfections in the original chunking algorithm, honeywords maintain their quality and serve as decoy passwords that confuse attackers and alert administrators to unauthorized access attempts. Despite potential imperfections, honeywords remain distinct from real passwords, maintain chunk order, and avoid adding digits at the end. Enhancements like case variations and number changes improve diversity and complexity. The integration of advanced technologies, including deep learning and natural language processing, along with improved Honeyword Generating Techniques (HGT), has gained significant attention in the network security field. These advancements offer the potential for stronger authentication mechanisms and bolster overall defenses against unauthorized access, contributing to a more secure environment for protecting sensitive information. In conclusion, generating honeywords that closely resemble original passwords is crucial for password protection. While Chunk-GPT3 and modified PwdSegment techniques show promise, considerations must be made regarding their limitations and effectiveness across different password datasets. Despite imperfections, honeywords serve as effective decoys by confusing attackers and alerting administrators, thanks to advancements in deep learning and natural language processing. These technologies continue to shape network security and offer opportunities to enhance authentication mechanisms, ultimately fortifying defenses against unauthorized access.

Bibliography

[1] ALI, Y., SADIQ, A., AND ALHAMDANI, W. Honeyword generation using a proposed discrete salp swarm algorithm. *Baghdad Science Journal* (09 2022).

[2] BROWN, A. Password analysis techniques: Insights and applications. *Journal of Cybersecurity 20*, 3 (2018), 145–160.

[3] BROWN, A. Password strength and vulnerability analysis: Insights and recommendations. *Journal of Cybersecurity 15*, 2 (2020), 87–104.

[4] 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* (New York, NY, USA, 2021), ASIA CCS '21, Association for Computing Machinery, p. 265–279.

[5] GRASSI, P., NEWTON, E., PERLNER, R., REGENSCHEID, A., BURR, W., RICHER, J., LEFKOVITZ, N., DANKER, J., CHOONG, Y.-Y., GREENE, K., AND THEOFANOS, M. Digital identity guidelines: Authentication and lifecycle management, 2017-06-22 2017.

[6] JOHNSON, R. Enhancing password security through chunk analysis: An experimental study. *International Journal of Information Security 25*, 2 (2020), 79–96.

[7] Jones, M. Password chunking: An approach for assessing password strength. In *Proceedings of the International Conference on Information Security* (2019), pp. 42–56.

[8] Juels, A., and Rivest, R. L. Honeywords: Making password-cracking detectable. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security* (New York, NY, USA, 2013), CCS '13, Association for Computing Machinery, p. 145–160.

[9] Taylor, E. Data analysis methods for password security: A comparative study. *Journal of Cybersecurity Research 35*, 4 (2021), 210–228.

[10] Veras, R., Collins, C., and Thorpe, J. A large-scale analysis of the semantic password model and linguistic patterns in passwords. *ACM Trans. Priv. Secur. 24*, 3 (Apr 2021).

[11] 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.

[12] Yang, K., Hu, X., Zhang, Q., Wei, J., and Liu, W. A honeywords generation method based on deep learning and rule-based password attack. In *Security and Privacy in New Computing Environments* (Cham, 2022), W. Shi, X. Chen, and K.-K. R. Choo, Eds., Springer International Publishing, pp. 294–306.

[13] Yasser, Y. A., Sadiq, A. T., and AlHamdani, W. A scrutiny of honeyword generation methods: Remarks on strengths and weaknesses points. *Cybern. Inf. Technol. 22*, 2 (Jun 2022), 3–25.

[14] 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)* (Jun 2022), IEEE.

[15] Yu, F., and Martin, M. V. Targeted honeyword generation with language models. *ArXiv abs/2208.06946* (2022).

[16] Yu, F., and Martin, M. V. Honey, I chunked the passwords: Generating semantic honeywords resistant to targeted attacks using pre-trained language models. In *Detection of Intrusions and Malware, and Vulnerability Assessment* (Cham, 2023), D. Gruss, F. Maggi, M. Fischer, and M. Carminati, Eds., Springer Nature Switzerland, pp. 89–108.