Information Systems Frontiers
https://doi.org/10.1007/s10796-022-10325-x

Password and Passphrase Guessing with Recurrent Neural Networks

Alex Nosenko[1] · **Yuan Cheng**[2] · **Haiquan Chen**[2]

Accepted: 12 August 2022

Abstract
Most online services continue their reliance on text-based passwords as the primary authentication mechanism. With a growing number of these services and the limited creativity to devise new memorable passwords, users tend to reuse their passwords across multiple platforms. These factors, combined with the increasing number of leaked passwords, make passwords vulnerable to cross-site guessing attacks. Over the years, researchers have proposed several prevalent methods to predict subsequently used passwords, such as dictionary attacks, rule-based approaches, neural networks, and combinations of the above. We exploit the correlation between the similarity and predictability of these subsequent passwords in a dataset of 28.8 million users and their 61.5 million passwords. We use a rule-based approach but delegate rule derivation, classification, and prediction to a Recurrent Neural Network (RNN). We limit the number of guessing attempts to ten yet get an astonishingly high prediction accuracy of up to 83% in under five attempts, twice as much as any other known model. The result makes our model effective for targeted online password guessing without getting spotted or locked out. To the best of our knowledge, this study is the first attempt of its kind using RNN. We also explore the use of RNN models in passphrase guessing. Passphrases are perceived to be more secure and easier to remember than passwords of the same length. We use a dataset that contains around 100,000 distinct phrases. We demonstrate that RNN models can predict complete passphrases given the initial word with rate up to 40%, which is twice better than other known approaches. Furthermore, our predictions can succeed in under 5,000 attempts, a 100% improvement compared to existing algorithms. In addition, this approach provides ease of deployment and low resource consumption. To our knowledge, it is the first attempt to exploit RNN for passphrase guessing.

Keywords Authentication · Passwords · Passphrases · Recurrent neural networks

1 Introduction

Text-based passwords remain the first and sometimes the only line of defense for most online services. Therefore, having a strong, unique password is extremely important to keep users' data safe. Government agencies, especially those storing users' personally identifiable information (PII), medical and legal records, follow the password guidance of the National Institute of Standards and Technology (NIST). Different online services have independent definitions of secure passwords. They also enforce different password composition, expiration, and reuse policies. This puts a lot of responsibility on users to create and maintain a large number of passwords. To cope with this burden, a user can either use a password manager, create and remember a unique password for each account, or create a very strong but memorable passphrase that follows all the guidelines and use it across all platforms. According to a survey by a cybersecurity company NordPass, 50% of the respondents in the United Kingdom find it extremely difficult to remember unique passwords for multiple accounts (Rawlings, 2020). The problem worsens when a user is required to change passwords due to password expiration or known security breaches. Based on numerous studies, most people

✉ Yuan Cheng
yuan.cheng@csus.edu

Alex Nosenko
alex.benliron@gmail.com

Haiquan Chen
haiquan.chen@csus.edu

[1] Santa Clara County Office of Education, 1290 Ridder Park Dr, San Jose 95131, CA, USA

[2] Department of Computer Science, California State University, Sacramento, 6000 J Street, Sacramento 95819, CA, USA

Published online: 27 August 2022

use commonly used phrases as their passphrases or simply reuse their passwords by modifying them slightly every time a new password is required (Wang et al., 2018; Florencio & Herley, 2007; Haque et al., 2013; Notoatmodjo & Thomborson, 2009; Keith et al., 2005). Since most of these studies were conducted in academic institutions and involved participants with higher education levels and better security awareness, the situation for the rest of the Internet community is probably even worse.

The habit of password reuse is detrimental to account security due to the increasing threat of cross-site password guessing attacks. In this form of attack, an attacker leverages previously leaked password datasets to guess passwords potentially used by the same user at different sites (Das et al., 2014). With an abundance of password leaks and data breaches, a large pool of publicly available passwords exists for a swarm of users. In addition, attackers have various tools at their disposal, such as dictionary-based attacks, rule-based attacks, and machine learning models for effective and automated guessing. Suppose each online service allows up to three attempts to enter a password before locking down the account. Consider that each user has registered on at least five popular online services. An attacker thus has at least 15 attempts to guess a password before being spotted or flagged. Due to the rate limit, the traditional brute-force and dictionary attacks will not be effective in this setting. However, rule-based and neural network-based predictions can still yield a high probability of successful guesses with rate-limiting enforced (Hitaj et al., 2019; Li et al., 2019; Liu et al., 2018; Melicher et al., 2016).

In this paper, we leverage the rule-based approach and automate the guessing process using a neural network model to derive modification patterns, complete the classification, and generate a password guess. We resort to neural networks, which outperform traditional classifiers like Naïve Bayes and k-nearest neighbors (KNN) used in prior research (Wang et al., 2018) to solve the classification problem. We use a character-based bidirectional long short-term memory (BiLSTM) model to generate passwords for each modification category. We then build an experimental model that can make predictions without knowing the modification patterns. We use a character-based LSTM encoder-decoder model as it is a common tool when one sequence of characters (e.g., the original password) needs to be transformed into another sequence of characters (e.g., a subsequent password). This model delivers outstanding prediction results for a significant amount of password pairs.

Additionally, we apply Recurrent Neural Networks (RNNs) and generative pre-trained transformers to the less researched area of passphrase guessing. We build a bidirectional LSTM model with an attention mechanism that can pick up linguistic patterns in passphrases and complete a multi-word phrase by knowing just the first few characters. We conduct a comparative analysis between this novel approach and traditional methods and see a significant reduction in the number of attempts required (up to 10 times) to generate a passphrase. Moreover, we see a remarkable 50%-100% improvement in successful prediction rate. Finally, we demonstrate how this approach can eliminate the constant need to acquire and maintain large passphrase dictionaries and how the entire process is easily deployable on platforms such as Google Colab.

The main contributions of our study can be summarized as follows:

- We created a neural network-based classifier for automated password modification category prediction.
- We built an LSTM-based model for password generation for each category.
- We designed an LSTM-based model that predicts a subsequent password based on the original password with up to 83% accuracy in under five attempts.
- We quantified the vulnerability of password reuse based on Levenshtein distance, Jaro-Winkler distance, and modification patterns.
- We built an attention-based LSTM model to predict targeted passphrases reaching a 40% prediction rate in under 5,000 attempts.
- We compared the passphrase prediction rate between transformers, LSTM, and Markov Chain models.
- We made recommendations for online services to enhance their authentication security.

The remainder of this paper is organized as follows. First, we discuss the related work in Section 2. Section 3 elaborates on the password prediction process and compares our results with the results from the existing models. Section 4 provides an overview of passphrases, reviews the related work, describes our passphrase prediction model, and analyzes the experimental results. Section 5 discusses our key findings on password and passphrase security and suggests some possible directions for future work. We conclude this study in Section 6.

2 Related Work

The problem of password guessing is not new. And over time, there has been an abundance of methods proposed to solve it. Among the most prominent guessing approaches are dictionary-based attacks, rule-based attacks, and neural network-based attacks. While some methods train and test their prediction models on the same dataset, other methods extract rules from one dataset and predict passwords from another dataset. The first type of method is called the single-site password guessing attack, where attackers crack

passwords from a single password leak. The second type is known as the cross-site password guessing attack, which exploits leaked passwords from multiple online services. Our research falls into the second type, where we aim to guess users' subsequent passwords from their known passwords from the same or a different site.

Next, we will provide an overview of different password guessing methods, including the most recent advance in using neural networks for this purpose.

2.1 Dictionary Attacks

Dictionary attacks depend upon the assumption that a password is either a word that belongs to a pre-compiled word list or dictionary (Haque et al., 2013), a valid, complete word (used in vocabulary attacks), or a valid passphrase (used by Markov model-based methods). The first two attacks may need many attempts to make a correct guess, require constant maintenance of dictionaries, and cost a tremendous amount of time and resources. Markov model-based methods, on the other hand, enable efficient password and passphrase cracking by only generating and testing linguistically likely passwords (Narayanan & Shmatikov, 2005) or linguistically correct phrases (Sparell & Simovits, 2016). These methods are commonly used for single-site password guessing attacks but can also instigate cross-site attacks with slight modifications. Unfortunately, most of the passwords in our dataset do not fall under the category of valid dictionary words or linguistically correct phrases. Thus, these methods are of limited use in solving our problem.

2.2 Rule-Based Attacks

Rule-based attacks rely on password creation and reuse patterns extracted from previously leaked datasets or user surveys and are widely used for cross-site attacks. Researchers conducted statistical analyses of leaked password datasets and discovered that most users stick to simple and easily memorable patterns (Weir et al., 2009; Zhang et al. 2010; Das et al., 2014; Wang et al., 2016, 2018). Based on the patterns, researchers were able to build algorithms and prediction trees that indicate the most probable modification categories and the most likely transformations. These data-driven algorithms aim to minimize the number of guesses and maximize prediction accuracy.

One of the most promising rule-based approaches employs probabilistic context-free grammars (PCFG) (Weir et al., 2009). This method analyzes leaked datasets and existing wordlists to create grammars for generating word-mangling rules. The next generation of rule-based guessing mechanisms is based on PCFG but leverages previously used passwords as an additional input to help predict subsequent passwords. This targeted online guessing framework is known as TarGuess (Wang et al., 2016). Among the four TarGuess models (I~IV), TarGuess-II is the closest to our work. It aims at cross-site targeted online guessing, where attackers make online guesses of a user's password at one site when given the user's one sister password from elsewhere. TarGuess-I and Xie et al.'s TarGuess-I^{+KPX} (Xie et al., 2020), on the other hand, require some PII, such as name and birthday. Zhang et al. developed a generic algorithmic framework for searching out possible transformations that convert a user's previous passwords to future ones (Zhang et al. 2010). Their optimal search strategy successfully cracked an average of 13% of the accounts in the experiment within five online guesses and 18% within ten attempts. Wang et al. introduced the next iteration of rule-based predictors by breaking a process into two steps (Wang et al., 2018). The first step uses a Naïve Bayes classifier to guess a modification category, and the second step applies the rule-based mechanism to predict the actual password. This approach significantly improves prediction, but the accuracy within ten attempts is still below 30%.

2.3 Neural Network-Based Guessing

The relatively new neural network-based approach surfaced in 2016 with the premise that RNNs can predict the next symbol in a character string if provided with enough training data (Melicher et al., 2016). The so-called FLA (Fast, Lean, and Accurate) method presented promising results in single-site and cross-site attacks and was used in combination with rule-based approaches (Liu et al., 2018; Xu et al., 2021) and the Markov model (Li et al., 2019; Xu et al., 2021). It helped overcome several previous limitations, such as the utilization of fixed-length context, by using a long short-term memory (LSTM) network (Hochreiter & Schmidhuber, 1997).

Generative Adversarial Networks (GANs), a subset of neural networks, were recently used to train neural network models to eliminate the need for learning modification patterns on a single-site attack (Hitaj et al., 2019; Pasquini et al., 2021; Fu et al., 2021). Despite showing a good prediction rate, most of these studies did not limit the number of guessing attempts and ran models until they exhausted every possibility. This assumption makes these models impractical in online password guessing in the real world.

Xu et al. proposed a novel method to segment passwords into chunks and then built three chunk-level password guessing models based on the Markov, PCFG, and FLA methods (Xu et al., 2021). Although this method showed state-of-the-art results in untargeted password guessing with LSTMs, it was not optimized for targeted online guessing.

Our research is built on the understanding of password modification patterns. We specifically focus on the targeted online password guessing problem - guessing subsequent passwords of a user from the given user's preceding sister

passwords. This makes our work distinct from many proposals in the existing literature (Hitaj et al., 2019; Pasquini et al., 2021; Fu et al., 2021; Xu et al., 2021). Our RNN-based approach lowers the number of attempts for cross-site targeted online password guessing. We generate two models, one with rules provided and the other one that derives the rules itself, and explore the prediction accuracy of the two models in under ten attempts. Furthermore, our models do not require PII or any knowledge besides sister passwords.

3 Password Guessing

This section first describes the password dataset used in the project and how we process it. We then elaborate the password prediction pipeline in Section 3.2. Finally, we present the experiment results in Section 3.3.

3.1 Password Dataset

The password dataset was provided by Wang et al. (2018), which consists of 61,552,446 individual passwords that belong to 28,836,775 users from 107 online services over eight years. The dataset has been sanitized and anonymized to protect personally identifiable information. Every user in the dataset has at least two passwords; thus, we can form at least one pair of subsequently used passwords. Although some users have more than two passwords, we only choose two passwords (i.e., one pair) for each user for simplicity.

We then pre-process the dataset to eliminate pairs of identical passwords and those that appear more than once. The resulting dataset contains 17,133,333 unique pairs. This process helps us identify a set of the most common passwords for further analysis. Among the most popular passwords are "123456," "password," "qwerty," "111111," "123123," "dragon," "monkey," "shadow," and "love."

When looking at the length of the passwords, we discover that 99% of the passwords are 5-17 characters long, which seems to be consistent with the common password requirements enforced by online services as well as the general human memory capacity (Stobert & Biddle, 2014). The passwords that are longer than 17 characters (hard to memorize) or shorter than 5 (not acceptable by most online services) and those that contain non-ASCII characters are considered outliers and are thus removed from further consideration.

We analyze the distribution of passwords based on two metrics, Levenshtein distance and Jaro-Winkler distance.

The Levenshtein distance is the number of edits (e.g., substitution, insertion, or deletion) needed to transform one string into another. For example, transforming "rain" to "shine" requires three steps, consisting of two substitutions and one insertion: "rain" → "sain" → "shin" → "shine." These operations could have been done in other orders, but at least three steps are needed (Hardeniya, 2015). In our dataset, the Levenshtein distance between passwords ranges from 0 to 17, as shown in Fig. 1. And the majority of password pairs have a Levenshtein distance in the range of 1 to 11. The Levenshtein distance can help set up password reuse rules that are easy for users to understand (e.g., make sure the subsequent password is different from the original by three characters). However, it suffers from a major limitation. For example, the Levenshtein distance between the words "password" and "password12345678" is 8, which is

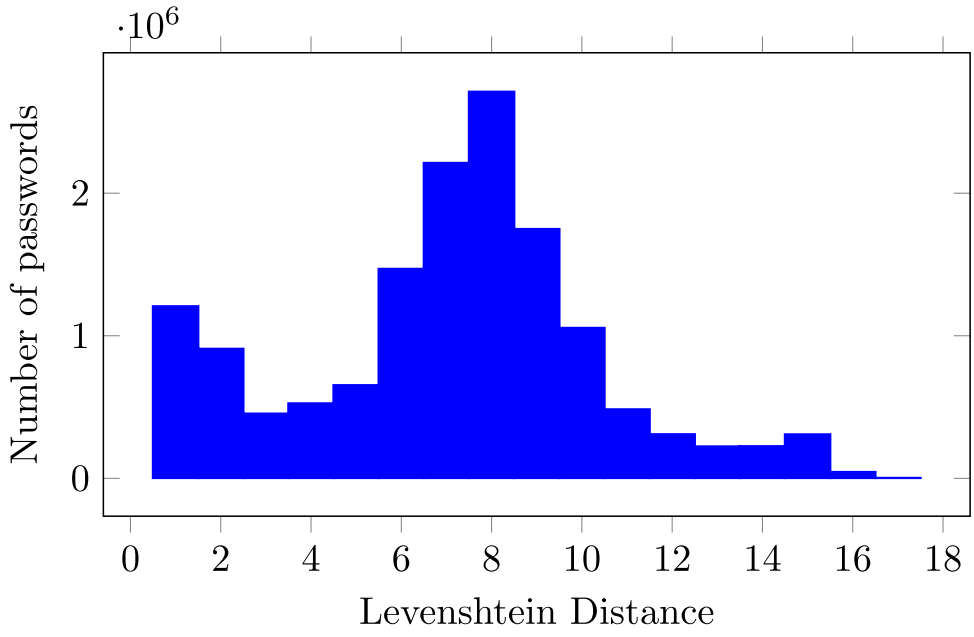

Fig. 1 Password distribution by Levenshtein distance

relatively high, although both words exhibit an easy-to-guess pattern.

The passwords that exhibit the highest similarity will have the smallest Levenshtein distance and the highest Jaro-Winkler distance and will be the best candidates for cross-site guessing attacks. We use the NLTK Python library for the Levenshtein distance and the StrsimPy library for the Jaro-Winkler distance to implement both metrics.

In prior research, several most common modification patterns were identified, including Substring, Common Substring, Capitalized, Leet, and Sequential Keys (Wang et al., 2018; Walia et al., 2020). We label each password pair based on these five patterns to create a labeled dataset. Finally, the pairs that do not fit into any of these rules are dropped. The labeled dataset contains 3,006,871 unique password pairs.

Figure 2 demonstrates the number of password pairs for the original dataset ("All passwords"), the set where an original password is not the same as its subsequent password ("Unique pairs"), the set of unique pairs where each password is between 5 to 17 characters long and contains only valid ASCII characters that will be used for most of this research ("Working set"), and the labeled set ("Known rules").

3.2 Password Prediction Process

So far, we have reviewed the data pre-processing steps necessary for password prediction. Next, we want to build a pipeline to take the resulting dataset, identify and tag modification patterns, classify passwords into appropriate buckets, and generate predictions for each original password. We also seek to take a step further by skipping tagging and classification and going straight to password prediction. We will refer to this process as direct password prediction. This approach will be especially beneficial when users combine multiple modification patterns or no rules can be identified. To the best of our knowledge, this approach has not yet been used for launching a single-user cross-site password guessing attack.

The prediction pipeline consists of four steps. During the first step, common modification patterns are defined, and each password pair is analyzed and labeled with a corresponding category. We use a neural network model to assign each original password into a single modification category during the second step. This process is known as the single-label prediction problem. During the third step, we build a second model to learn about possible modifications within each category. With 90% accuracy on the test data, the model can understand and generate all possible modifications for each category. Both models are then combined, and the resulting pipeline is assembled in the last step. Our approach can take just one original password as an input, classify it into a modification category, and generate password guesses. We will now review each step in further detail.

3.2.1 Tagging

Before tagging password pairs, let us first introduce the common password modification patterns we borrowed from Wang et al.'s prior work (Wang et al., 2018). Leet refers to transforming alphanumeric characters into visually similar symbols and vice versa. The Substring category includes password pairs where one password is a substring of the other. Adding symbols to the head or tail of a string is the most common modification of this category. Capitalization is where one or more symbols are in uppercase. The Common Substring category contains password pairs that share common letter combinations. The Sequential Keys category consists of passwords that contain alphabetically ordered letters (e.g., "abcd"), sequential numbers (e.g., "1234"), and adjacent keys on the keyboard (e.g., "qwert," "asdfg," etc.). We define a function to identify which pattern each password pair fits in and tag each pair with a corresponding modification pattern category. After the tagging is completed, we drop the passwords that do not match any rules

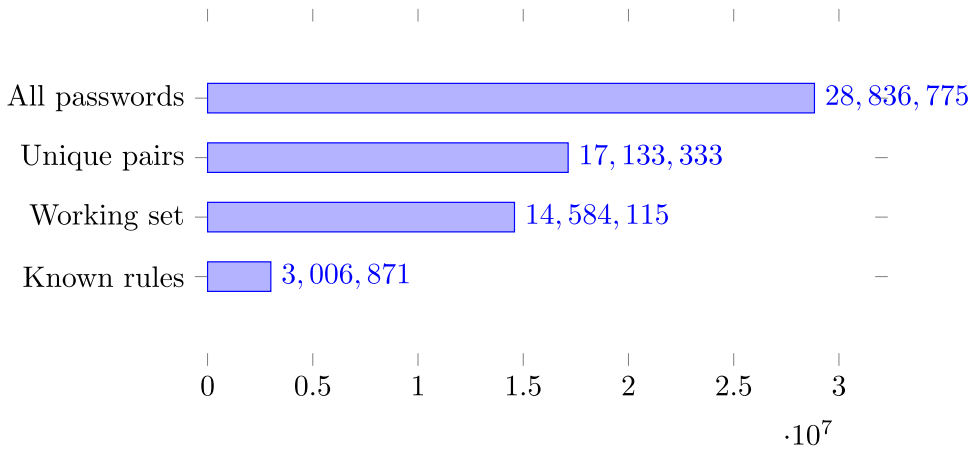

Fig. 2 The number of password pairs in each dataset

or contain non-ASCII symbols. The resulting dataset has 3,006,871 pairs of passwords. Figure 3 shows the distribution of each password modification category in the tagged dataset. The most common patterns found in the dataset are Substring and Common Substring. Together they cover 2,674,521 password pairs, around 89% of the dataset. These password pairs provide sufficient training data for our proposed model. The other three categories represent about 10% of the dataset. Except for the Sequential Keys category, we still collect enough training data from the categories of Capitalization and Leet.

3.2.2 Classification

Conventional classifiers have been used to solve the classification problem as they are easy to use and do not require heavy data pre-processing. However, neural networks can enable automated tagging and classification without manual interventions. Furthermore, they have proven to deliver a better prediction rate, especially on larger datasets (Schumacher et al., 1996).

We build a 4-layer classifier for automated password modification category prediction using the Keras Python library. The Input Layer takes a single sequence of characters with the same length as the longest password in the dataset, which is 17 characters long. The One-Hot Encoder processes every password as a sequence of characters and transforms those sequences into a one-hot numeric array. It assigns the value of 1 if the character is present in a given word or 0 if otherwise. The encoding is passed to the LSTM units. We use a character-level Bidirectional LSTM (BiLSTM) Layer, which is an extension of traditional LSTMs and can improve the model performance on sequence classification problems. The BiLSTM Layer runs inputs in two directions, one from the past to the future and the other from the future to the past. Unlike unidirectional LSTM, BiLSTM uses two hidden states and can preserve information from the past and future at any point in time. Because of these qualities, BiLSTM can better understand the context around each character in the sequence (Xu et al., 2019). The output of the BiLSTM cells is fed to a dense Activation Layer. The Activation Layer contains an activation function, which defines how the weighted sum of the input is transformed into an output. To ensure that a model learns features and does not converge prematurely, we use the Adam optimizer with a small learning rate (Kingma & Ba, 2014). This optimizer is used to update network weights during each training iteration.

After trying out different numbers of layers for hyperparameter tuning, the model with one LSTM layer yields the best performance.

3.2.3 Password Generation

In this step, we train a character-level BiLSTM model to generate passwords within each modification category. The model contains two Input Layers. Input Layer #1

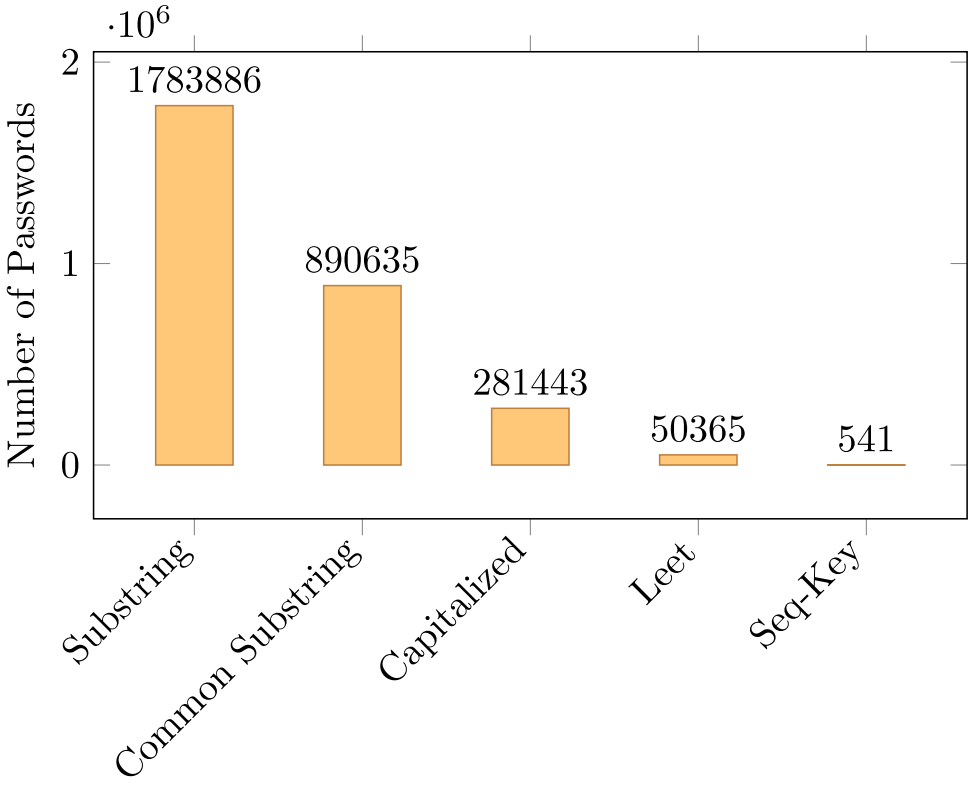

Fig. 3 Category distribution of the tagged dataset

takes as input the previously used passwords in the form of a sequence of characters, while Input Layer #2 takes the password modification categories as input. The first stream includes Input Layer #1, a One-Hot encoder, and a BiLSTM layer. These three layers act similarly to what is described in Section 3.2.2. The second input stream includes Input Layer #2 and a Repeat Vector Layer, both of which are used to add a list of modification patterns to the model as each prediction is generated within a single category. Finally, the Concatenation Layer combines the outputs of both streams and feeds the combined outputs into an Activation Layer. The Activation Layer acts the same as described in Section 3.2.2. The resulting model can generate highly accurate password guesses for each modification category.

To generate password guesses for all categories, we combine the classification model and the password generation model into one prediction pipeline, namely Pipeline Prediction Mechanism (PPM). The first model predicts a modification category when an original password enters the pipeline. Then, the predicted category and the original password are fed into the second model, which generates password candidates and chooses the top ten candidates with the highest probability. Once the prediction completes, we verify if a password guess is correct.

We end up with this architecture as it generates the best performance during hyperparameter tuning. However, adding more LSTM layers may overfit the model.

3.2.4 Direct Password Prediction

To our knowledge, RNN has never been used to solve cross-site password guessing for the same user. This problem, however, is similar to the problem of machine translation. In both problems, the source may vary in length and character dictionary. The model architecture consists of at least two LSTM layers, an encoder and a decoder. The encoder takes the input sequence and summarizes the information into a context vector or hidden states of LSTM (Cho et al., 2014). The outputs are unimportant and thus are dropped, but the hidden states are saved. This context vector encapsulates the information for all input elements and will be used by the decoder to generate predictions. The decoder is also an LSTM layer that takes the encoder output as an initial state and produces an output sequence. Our model uses the Softmax layer from the Keras library as an Activation Layer. The Softmax function is based on a normalized exponential function and is used as the last layer on a neural network to normalize the output to a probability distribution over predicted output classes (Kouretas & Paliouras, 2019). In our model, it will determine the output modified password.

Since a model can generate multiple predictions with different degrees of confidence, we need an algorithm to choose the top ten most probable outputs. We use the Beam search algorithm, one of the most widely used for sequence-to-sequence machine translation problems (Yoo et al., 2020), to help us identify the most probable predictions. Direct password prediction is the most promising approach as it eliminates the need for constant rule derivation and distribution analysis and eases dataset pre-processing. The resulting model, the Direct Prediction Mechanism (DPM), is rule-independent and delivers a high prediction rate.

3.3 Experimental Results

3.3.1 Hardware Requirements

We ran most of this project on Google Colab Pro, a cloud-based Jupyter notebook environment. The hosted runtime environment uses Tesla P100-PCIE-16GB GPU, Intel(R) Xeon(R) CPU @ 2.20GHz processor, 25GB of RAM, and 109GB of disk space. The most hardware-demanding parts of the experiment were dataset pre-processing, model training, and direct password prediction. Direct password prediction was the most computationally expensive. Based on Google Colab measurements, it took 13GB of RAM to train a model on 400k records and 20GB for 500k records. Even though it might sound like a significant amount of resources, it is not unreachable for a sophisticated attacker. Better hardware resources will yield better performance results since we can train the model on a larger dataset and make faster predictions with higher accuracy.

3.3.2 Results

We first compared the results of an LSTM classifier and a Naïve Bayes classifier. Table 1 shows that the LSTM-based model delivers better results than the Naïve Bayes classifier, especially in underrepresented categories, such as Leet and Common Substring.

LSTM outperforms traditional classifiers because it captures and preserves the sequential order (i.e., the order in

Table 1 LSTM vs. Naïve Bayes classifier

		Precision	Recall	F1-Score
LSTM	Capitalized	0.65	0.58	0.61
	Common Substring	0.58	0.31	0.4
	Leet	0.49	0.23	0.31
	Seq-Key	0	0	0
	Substring	0.73	0.91	0.81
Naïve Bayes	Capitalized	0.6	0.39	0.48
	Common Substring	0.38	0.05	0.08
	Leet	0.14	0.01	0.01
	Seq-Key	0	0	0
	Substring	0.66	0.9	0.79

which the characters appear in a password), while the classical models do not. For example, "abac" is different than "aabc" for LSTM, while it is the same for a classical model as it only considers the frequency of the featured characters 'a,' 'b,' 'c,' which are the same for both strings.

We then evaluated the performance of our password generation model. The closest existing studies are the ones that estimate how many guesses are needed to predict a password correctly (Das et al., 2014; Wang et al., 2018, 2016). All of these papers published their prediction rate within the first ten attempts. Therefore, we use ten attempts as part of the experiment parameters. We compared the results of our Pipeline Prediction Mechanism (PPM) and Direct Prediction Mechanism (DPM) with three existing algorithms from Wang et al. (2018) (referred to as "Domino" hereafter), Wang et al. (2016) (referred to as "TarGuess-II" hereafter), and Das et al. (2014) (referred to as "Tangled" hereafter).

On average, both of our models exhibited a 5% improvement in overall prediction rate compared to "Domino" and three times more accurate predictions than TarGuess-II and "Tangled" as shown in Fig. 4. Although the improvement does not seem drastic, if we can predict 5% more passwords out of 6 million, that is around 300,000 more compromised accounts.

A larger difference in prediction rate between "Domino," PPM, and DPM can be observed in the first four attempts, as shown in Fig. 5. Our PPM and DPM models can predict twice as many passwords as "Domino" within the first four attempts. This is especially surprising considering that DPM was only provided with the original password and no other prior information. We also observed that the DPM model makes most of the predictions during the first few attempts, and then its confidence decreases as well as the number of predicted passwords (see Fig. 6). On the other

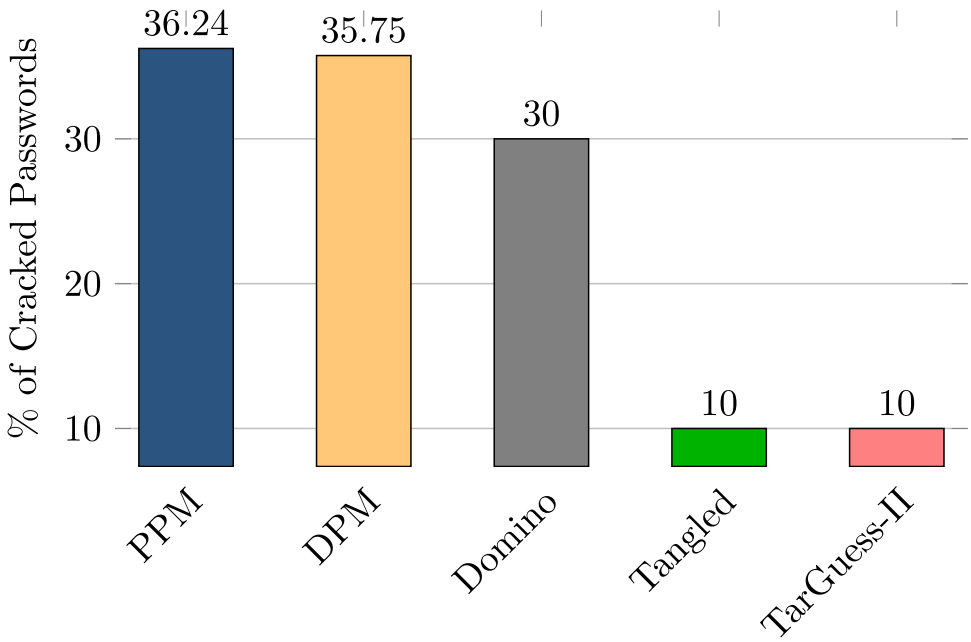

Fig. 4 The percentage of cracked passwords at ten attempts

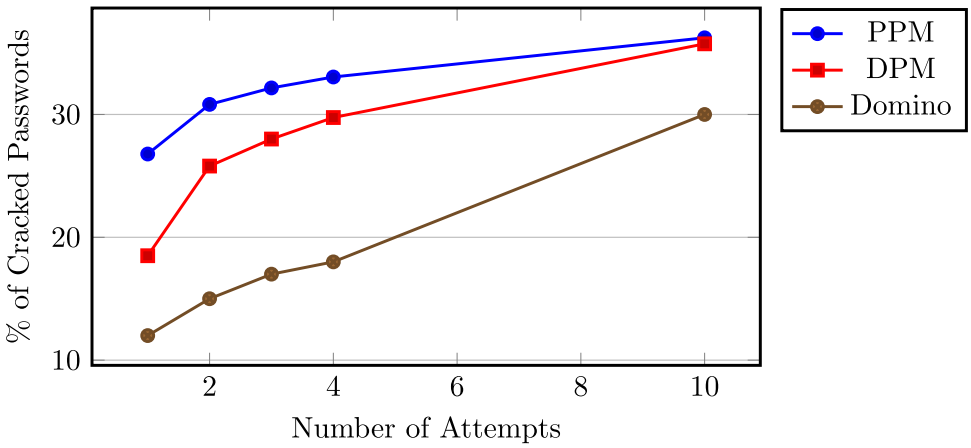

Fig. 5 The growth of the percentage of cracked passwords under ten attempts

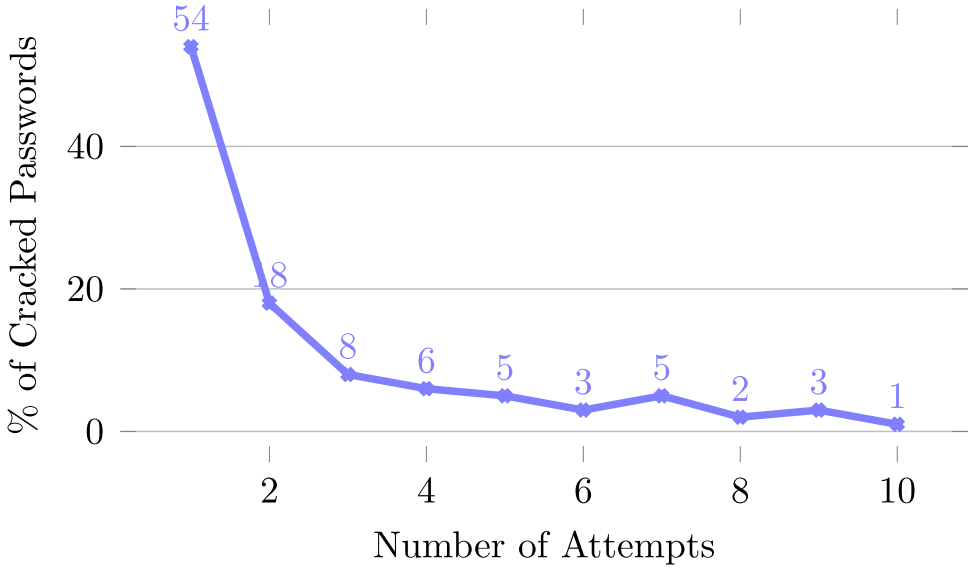

Fig. 6 DPM prediction rate for each attempt

hand, traditional approaches exhaust all possible combinations and thus predict more as they try more. This reveals a fundamental difference between our method and those traditional ones.

Next, we zoomed in on the results of the DPM predictions to quantify for the first time the correlation between password predictability and Levenshtein distance for subsequent passwords. The prediction rate, as shown in Fig. 7, is broken down by the Levenshtein distance on the x-axis (x-coordinates correspond to the distance values 1, 2, 3, 4, 5, and 6, respectfully). The y-axis refers to the percentage of guessed passwords. We found that it is 6-8 times harder to predict a subsequent password when the Levenshtein distance is 4 compared to the same task when the distance is 1. Changing only one character in subsequent passwords does almost nothing to improve the overall password strength since the prediction rate is as high as 83%. In other words, almost 8 out of 10 subsequent passwords can be predicted regardless of modification patterns if the two passwords only differ in one character.

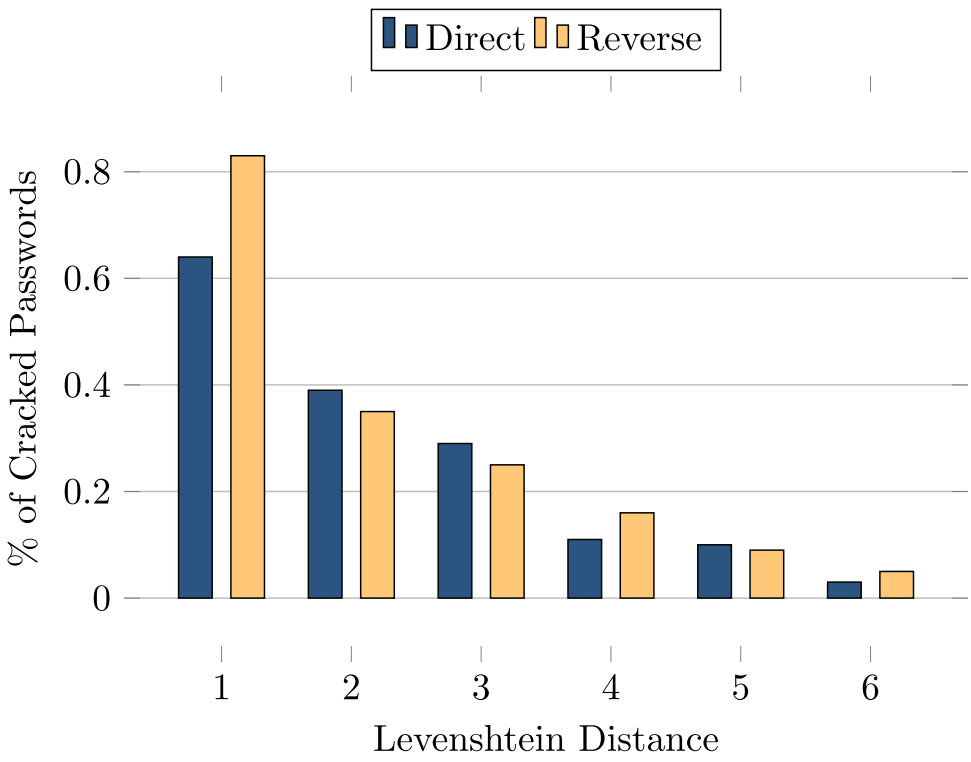

Fig. 7 Prediction rate w.r.t. Levenshtein distance

We also examined the association between password predictability and Jaro-Winkler distance by running a DPM model on each distance interval, as shown in Fig. 8. Passwords with less than 0.7 Jaro-Winkler distance result in a prediction rate of below 1% under five guesses, compared to passwords with the same metric of above 0.9, which have a prediction rate of around 50%.

We then investigated the prediction results to see how specific modification patterns contribute to the overall password predictability, as most predicted passwords exhibit some pattern. The unrelated password pairs with no syntactic or semantic similarity proved to be the hardest to predict. Figure 9 shows that Capitalization and having a subsequent password being a substring or containing a substring of the original are the easiest categories to break and are at least

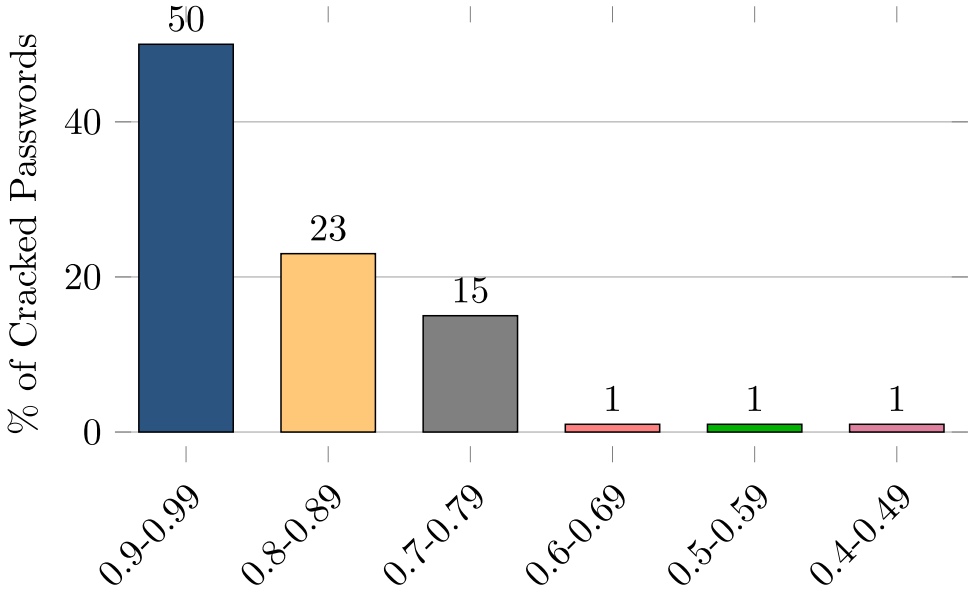

Fig. 8 Prediction rate w.r.t. Jaro-Winkler distance

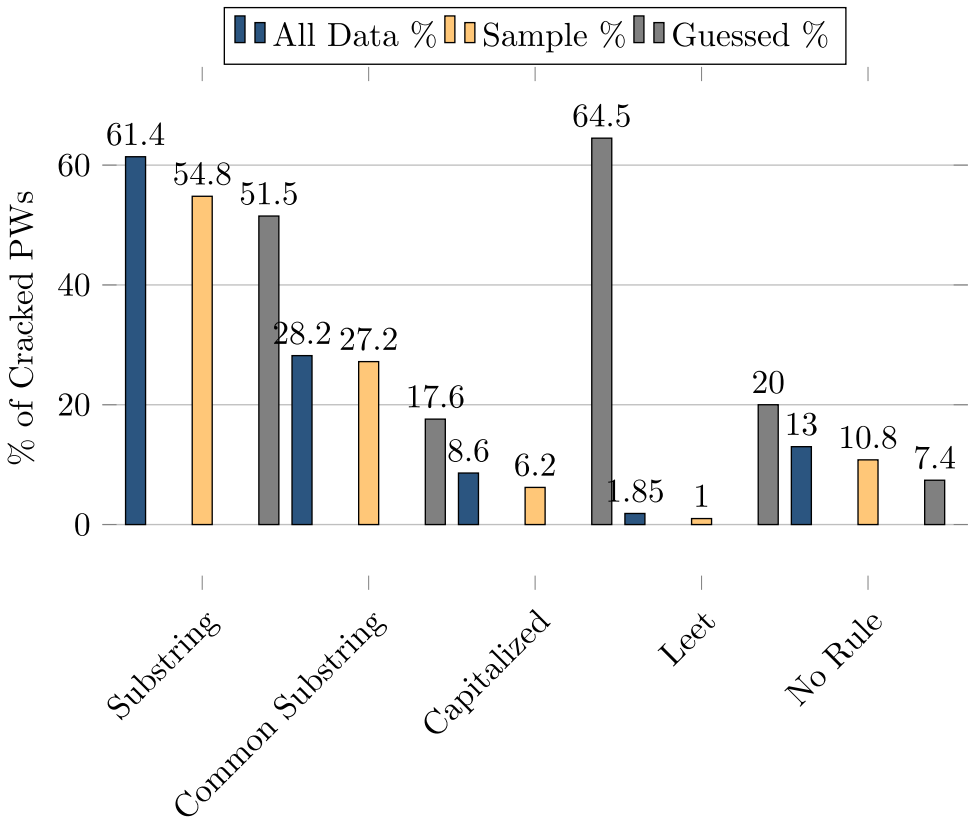

Fig. 9 Prediction within each category

20%-50% less secure. Considering how easy it is for a user to remember a password with a few added characters or some capitalization changes, it is as easy for the model to guess it. Since Substring is the most represented category in the original dataset, it is not surprising to see that DPM is well trained in this category and is thus able to make a successful guess around 50% of the time, as shown in Fig. 9. However, the interesting fact is that Capitalization represents only around 9% of the dataset, yet a model successfully predicts around 64.5% of those passwords. It is also worth noticing that DPM can predict around 7.4% (or 25,000) of passwords that do not follow any known modification patterns.

Both models demonstrated a great prediction rate, beating the close competitors 2:1 in scenarios where only a few guessing attempts are allowed. Our models adopt a more fine-grained prediction approach and can predict 83% of passwords in some common modification categories. With more data available, the performance of our models improves. But our models can also work well on a smaller training set (e.g., 50k size).

4 Passphrase Guessing

Passphrases were first introduced by IBM in 1985 (Kurzban, 1985). The original design suggested that a user selects a combination of system-generated words and then uses it as an authentication method while executing jobs on mainframes. Amazon introduced the first well-known commercial utilization of passphrases for authentication. In 2009, they implemented Amazon Payphrases (Bonneau & Shutova, 2012). Customers were offered a way to complete a purchase without signing in and clicking the "proceed to checkout" button by simply entering a passphrase and a four-digit PIN. The passphrases were 4 to 100 characters long and could be used on partners' retail sites such as Buy.com, Car Toys, DKNY, J &R Electronics, Jockey, and Patagonia. However, Amazon quickly abandoned this system as the phrases were easily guessable and thus not secure. In addition, the entire system was not user-friendly.

In 2019, NIST recommended that the password fields accept up to 64 characters to accommodate the use of passphrases (Burr et al., 2013). In another publication, Cybersecurity and Infrastructure Security Agency (CISA) advocated using passphrases as a more secure alternative to easily breakable passwords (Huth et al., 2012). The use and wide adoption of passphrases is hindered by the same issue that passwords suffer from. The passphrases that users create are commonly used phrases and are susceptible to guessing attacks. The system-generated phrases are more secure but hard to remember and maintain. There are a lot of survey-based studies that dig deeper into the understanding of what makes passphrases memorable while keeping them secure (Keith et al., 2005; Joudaki et al., 2018; Blanchard et al., 2018). When discussing memorability, the experiments measured how different groups of respondents recall the phrases generated by various methods (Keith et al., 2005; Joudaki et al., 2018; Blanchard et al., 2018; Woo et al., 2016). When considering security, the researchers performed dictionary attacks (Labrande, 2015), rule-based attacks (Rao et al., 2013; Bonneau & Shutova, 2012), and used the Markov Chain method to break the passphrases (Sparell & Simovits, 2016).

4.1 Related Work on Passphrase Guessing

Passphrase guessing is a sub-field of password guessing. Researchers have applied similar methods and attack vectors in both scenarios. We start with a review of the most trivial technique, i.e., the dictionary-based brute-force attack. We then analyze the improved approach where dictionary attack is supplemented by tools like word mangling, rules, and grammar analysis. Then, we discuss the Markov Chain method that accounts for statistical word association in constructing passphrases. Each method is valuable and can be beneficial in specific attack scenarios. Unfortunately, there is no uniform way to compare these methods for the best one, as most studies use different datasets and settings in their experiments. Our study aims not to find the best approach but to offer additional tools to examine the security of passphrases while streamlining the process.

In a dictionary attack, an attacker assembles a list of common words and phrases and then conducts an exhaustive search to find possible matches. The predictions are often hashed and then compared to datasets of leaked passwords. The success of this method relies on the fact that most passphrases are derived from natural language corpora, such as quotes, proverbs, lyrics, etc. (Labrande, 2015). This method requires a low skill level from attackers, and its tools are highly available. However, it requires heavy computational resources and needs to maintain large dictionaries.

In rule-based approaches, passphrases are built based on the rules of grammar (Rao et al., 2013; Bonneau & Shutova, 2012; Han et al., 2020). Rao et al. reduced the password search space and generated more accurate guesses from Parts-of-Speech tag rules (Rao et al., 2013). In another rule-based approach, Bonneau et al. analyzed the syntactic construction of phrases and the factors that make users combine words into phrases (Bonneau & Shutova, 2012). The TransPCFG framework (Han et al., 2020) applied the grammar learned from short passwords to long password guessing using PCFG (Weir et al., 2009). These studies showed that rule-based methods deliver better prediction results in fewer attempts than the traditional dictionary attacks. However, this approach is very dataset-specific, requires a lot of manual processing, and still relies on exhaustive searches.

The Markov Chain method is commonly used in both password and passphrase guessing. It can predict the entire passphrase based on the initial word. The consecutive words are constructed by analyzing the probability distribution and the current state (Sparell & Simovits, 2016). The process does not consider the meaning of the words or their relations with each other. Instead, it only relies on the probability of them occurring together in the given dataset. We use this method to establish a baseline for prediction result comparison.

4.2 Passphrase Prediction Process

Our work is based on understanding the underlying structure of passphrases as natural language phrases. The problem we tackle here is targeted offline passphrase guessing as opposed to targeted online password guessing discussed in Section 3. It is also different from the existing literature discussed in Section 4.1 as they addressed untargeted offline passphrase guessing. We hypothesize that sequential and attention-based language models can be useful in generating real-world passphrases. Our neural network-based approach focuses on lowering the number of attempts and eliminating the need to maintain passphrases dictionaries. It also simplifies the prediction pipeline and utilizes less storage and fewer computational resources. Our first method employs a word-level attention-based LSTM model, which is different from the character-level LSTM models we developed in Section 3 with no attention mechanism. Our second approach utilizes the Generative Pre-trained Transformer (GPT-2) provided by OpenAI and is available for public use Radfordetal. (2019).

4.2.1 Passphrase Dataset

The dataset for passphrase prediction is from the Corpus of Contemporary American English (COCA) (Davies, 2009). It is a large, recent, genre-balanced corpus of English. It comprises more than one billion words in 485,202 texts, including 20 million words each year from 1990 to 2019. For each year, the corpus is evenly divided between the genres of TV and movie subtitles, spoken, fiction, popular magazines, newspapers, and academic journals. We use three samples of the dataset that are available for free download. Each sample consists of fifty thousand phrases, with the frequency of each phrase. The first sample contains two-word phrases; the second sample consists of three-word phrases; the third sample comprises four-word phrases. We use frequency as a fair indicator of a phrase commonality in a natural language. To account for each phrase frequency without unnecessary bloating of the dataset, we take the log of each frequency and add the corresponding amounts of duplicated phrases to the dataset. The resulting dataset contains 574,531 phrases, out of which 99,953 are distinct.

4.2.2 Attack Vector

An attacker needs to know the beginning of a passphrase to run the prediction models. The more of a passphrase is known, the easier it is to predict the entire phrase. Several attack vectors can be used to obtain the initial information, including shoulder-surfing, eavesdropping, or other forms of social engineering. Shoulder-surfing is a social engineering technique used to unlawfully gather confidential information by looking over a victim's shoulder. Other methods may include acquiring unhashed credentials from online data leaks. The beginning of the phrase or the first word becomes an input to generate the entire phrase.

We begin our experiment by establishing a baseline using the Markov Chain model. The model takes as input the first word of a bi-gram or 3-gram and uses the Markov algorithm to generate the rest of the phrase. Then, the model builds a transition matrix based on the statistical probability of two words co-occurring. As a result, the model generates predictions that we compare to the target phrases to see if they match. For our experiment, we used a generic Python implementation of the Markov model available online (Sparell & Simovits, 2016).

4.2.3 LSTM-Based Passphrase Prediction

The attention-based LSTM model architecture consists of an encoder and a decoder, as shown in Fig. 10. The encoder takes an input sequence and summarizes the information into a context vector, which encapsulates the information for all input elements and is used by the decoder to generate predictions. Our implementation of the encoder consists of several layers.

The input layer takes a numeric representation of the first word in the phrase as the starting point of the prediction. The embedding layer encodes the words into a real-value vector. We use the GloVe model to generate word embeddings, which contains 400,000 most used words crawled from Wikipedia and other sources and the matrix of their concurrences (Pennington et al., 2014). The encoder layer based on the Bidirectional LSTM layer compresses the input and reduces the vector's dimension. The resulting vector is fed to the decoder for translation.

The decoder is based on the LSTM layer that takes the encoder output as an initial state and produces an output sequence. The decoder consists of several important layers. The input layer takes the vector generated in the previous step. Multiple attention layers form the attention mechanism. Instead of having a single hidden state to encode and decode, the sequence attention mechanism uses a function.

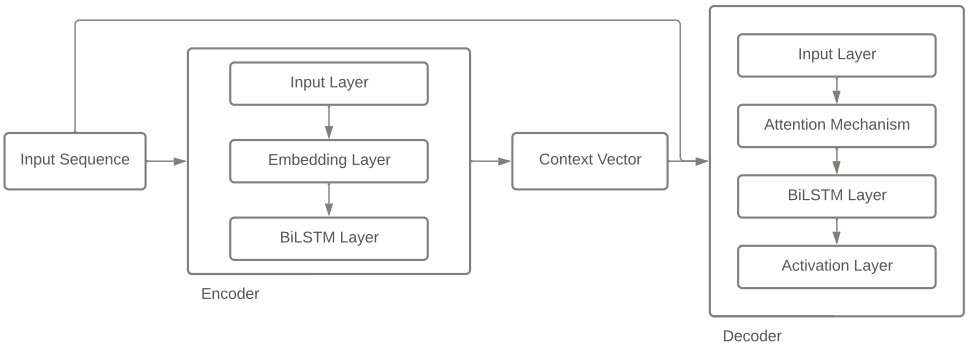

Fig. 10 The passphrase generation model

This function can be described as a query that maps a set of key-value pairs to an output. The query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key (Vaswani et al., 2017). In other words, the context vector takes all the encoder cells' outputs as inputs to compute the probability distribution for each single word decoder wants to generate. It helps the decoder capture the overall information about the input. Then decoder layer produces the output word sequence by looking at the context vector and the input word. We used Adam optimizer and sparse categorical cross-entropy as a loss function. One advantage of using sparse categorical cross-entropy is better memory and computational resource utilization as it uses a single integer for each class and not an entire vector.

4.2.4 GPT-2-Based Passphrase Prediction

Transformers have been used for phrase auto-completion for a long time (Vaswani et al., 2017). We use GPT-2, an open-source language model (Radford et al., 2019). The model's architecture is very similar to the decoder-only transformer. It consists of multiple decoder blocks stacked together. Each decoder block has a Feed Forward Neural Network and a Masked Self-Attention layer. The masked self-attention layer is based on the model's understanding of relevant and associated words as well as the context necessary before processing that word. This layer assigns scores defining the relevancy of each word in the segment and adds up their vector representation (Vaswani et al., 2017). The vector representation is passed to the fully connected Neural Network layer for processing.

We use the GPT-2 Small model version with 124M parameters. Parameters represent the weights of each connection in the neural network and are updated during the model training stage. We re-train it on the wordlist extracted from the 4-gram dataset sample used in this research. This allows the model to generate more relevant predictions. Aside from fine-tuning the model, we utilize some hyper-parameters provided by the model authors. The `prefix` parameter is used to provide the model with the beginning of the phrase. The `temperature` parameter is a float number controlling randomness. Lower temperature makes prediction more repetitive. Length controls how many words the generated phrase will contain. Parameter `top_p` helps narrow the pool of predictions to choose the best candidates. The number of phrases we generated is controlled by the `N_samples` parameter and is set to 50.

4.3 Passphrase Prediction Results

We complete the passphrase prediction experiment on the same hardware setup as described in Section 3.3.1: Google Colab Pro with Tesla P100-PCIE-16GB CPU, Intel(R) Xeon(R) CPU @ 2.20GHz processor, 25GB of RAM, and 109GB of disk space.

Table 2 shows the results of some existing passphrase breaking methods in terms of maximum prediction rate and the number of attempts used (Labrande, 2015; Bonneau & Shutova, 2012; Rao et al., 2013; Sparell & Simovits, 2016). Unfortunately, most of these models require a huge number of attempts to guess a passphrase correctly.

Note that most authors of the existing work did not disclose every detail and parameter of their experiments. Moreover, conducting a fair one-to-one comparison between the existing models and ours is very difficult as they often use different datasets that are not always provided.

Table 2 Passphrase prediction comparison

Algorithm	Max. prediction rate	Max. number of attempts
Dictionary-based (Labrande, 2015)	10%	2×10^{12}
Rule-based (Bonneau & Shutova, 2012; Rao et al., 2013)	10-20%	2×10^{6}
Markov (Sparell & Simovits, 2016)	11%	2×10^{27}

Figure 11 demonstrates the LSTM model prediction rate over time. The model cracked 24% of the passphrases with about one thousand attempts. It eventually reached the maximum prediction rate of 40% at around five thousand attempts. Then the prediction rate starts to flatten up.

In addition to the LSTM and GPT-2 models we developed, we also implemented a Markov-based model to serve as a baseline. Table 3 provides a detailed comparison between the models we developed and configured. Our implementation of the Markov-based model was not optimized for effective GPU and resource utilization and stopped converging at around 20 attempts. At that point, we were unable to record the prediction rate. The model started timing out if we set the number of attempts to higher than 20. With LSTM, we saw a proportional increase of resources for a higher number of attempts; however, the overall utilization remained low. The GPT-2 model performed the best in predicting longer phrases but required more input to provide accurate guesses faster. LSTM showed an exceptional prediction rate even on less frequently occurred phrases, which are typically harder to break. This experiment asserts that longer phrases and those with three or more words are harder to predict for Markov-based models and LSTMs. All of our models delivered a significant increase in correct predictions while reducing the number of attempts. It demonstrates that our models are highly effective in breaking passphrases.

Table 3 Prediction rate and setup comparison

Description	Markov	GPT-2	LSTM
Effective # of guesses	20	20	5000
Max prediction rate	12%	21%	40%
Resource utilization	Low	Low	Low
Time of task completion	Very High	Very Low	Low
Average frequency of phrases	65%	44%	22%
Average phrase length	15	14	11
Average length of prediction	6	6	5
Dataset (2,3,4-grams)	2, 3	4	2, 3
Initial input	1 word	2 words	1 word
Predicted words	1	2	1

5 Discussion and Future Work

5.1 Insights on Passwords

Predicting slightly modified subsequent passwords is getting easier. With more data breaches occurring each year, there is an abundance of leaked passwords, including sister passwords for the same users. This allows us to extract more fine-grained modification patterns and train more robust prediction models. Our experiment showed that having the original password alone is enough to predict a subsequent password with a high probability in just a few attempts. With most online services allowing up to 10 attempts (Brostoff & Sasse, 2003), an attacker can use the proposed mechanisms and models to generate subsequent passwords and try to compromise an account without raising the alarm. With the availability of more powerful computational resources and companies delaying the public release of data breach details, an attacker can re-train their models to account for newly acquired data and enhance the chance of success.

The length and complexity requirements recommended by the NIST guideline (Burr et al., 2013) are outdated in practice today. Having a long password that consists of various symbols might add extra security only when it comes to single-site dictionary attacks and eavesdropping (Grassi

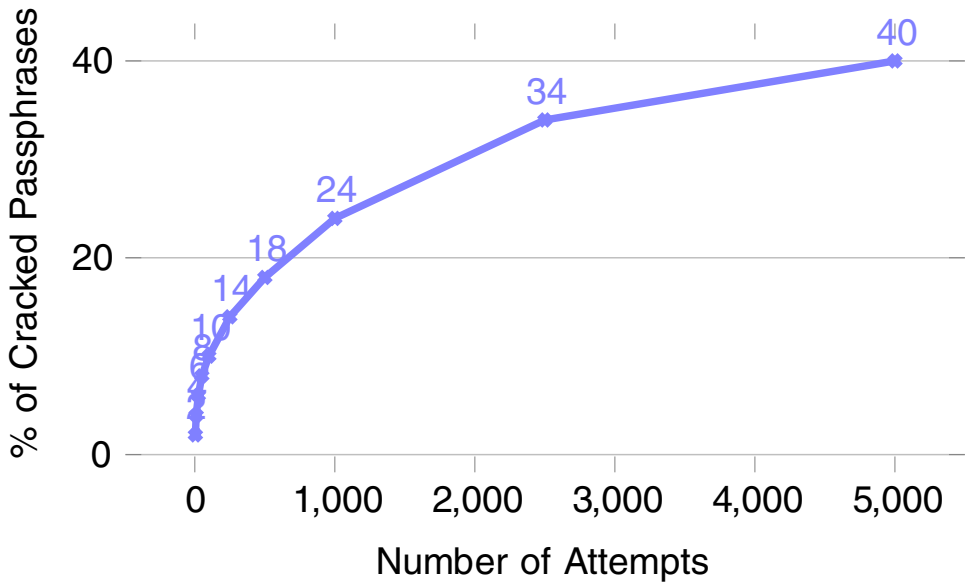

Fig. 11 Passphrase prediction rate by LSTM model

et al., 2017). Making slight modifications based on an original password provides little to no additional security in cross-site attacks. It may even give users a false sense of security as these changes seem to align with the characteristics of "strong passwords" defined by the guideline. Our studies showed that we need to consider the similarity between an original password and its successors as a new requirement in a future edition of user authentication guidelines. The less similarity there is between sister passwords, the more secure user data is Murray & Malone (2018). Even with sufficient training data, neural network-based guessing algorithms require much more attempts to crack passwords that are distant in terms of similarity metrics. For example, we observed that passwords four edits away are at least three times harder to break than those with just one modification apart. Having completely unrelated passwords can mitigate cross-site attacks in the presence of a password breach.

Updating user authentication guidelines also helps standardize web frameworks and development tools to include libraries that support similarity-based proactive password checking. A proactive password checker can prevent users from choosing an easy-to-guess subsequent password, especially when it is similar to the used ones. This process, however, should not have a detrimental effect on user experience and may include a system to suggest a secure password if a user struggles to create one.

Service providers should consider using passwords alone in favor of two-factor authentication, biometrics, behavioral authentication, and other alternative means. Users' creativity, memory capacity, or the level of education in computer security should not be the decisive factors in data security. Most survey participants in the password guessing studies represent a younger and higher educated cohort compared to the general population (Das et al., 2014; Von Zezschwitz et al., 2013). The safe guess would be that a general population would have even more insecure password habits. It is hard to change users' habits as we consistently see users reusing the same passwords or modifying them slightly. As users entrust their data and private information to more and more online services, these services should step up and carry more responsibilities to ensure that users' negligence or lack of education does not jeopardize the safety of their data.

5.2 Insights on Passphrases

Passphrases remain appealing as they are easy to implement and maintain on an enterprise level. They are also user and application-friendly and provide moderate security levels if combined with user training. They appear more advantageous to traditional passwords as they tend to be longer and more memorable. However, memorable passphrases are inherently insecure due to non-secure patterns. These phrases are usually derived from common sentences in mainstream culture or natural language corpora (Kuo et al., 2006; Labrande, 2015), correctly following the grammar (Rao et al., 2013; Sparell & Simovits, 2016), or just made of mnemonic words (Bonneau & Shutova, 2012; Labrande, 2015). The same factor that makes passphrases memorable makes them less secure and more susceptible to guessing attacks.

The rapid development of neural networks and artificial intelligence and a decrease in hardware prices allow malicious actors to execute more sophisticated attacks. Over the last few years, Microsoft, Nvidia, and OpenAI released powerful Natural Generation Models capable of language comprehension and passphrase generation (Radford et al., 2019). In addition, numerous data leaks allow researchers to build powerful dictionaries to boost passphrase prediction and model training. Passphrases alone are essentially no longer a reliable alternative to ordinary passwords. Enterprises and service providers should supplement the use of passphrases with other tools like we mentioned in Section 5.1 to enhance their authentication mechanism. And users should avoid natural language tendencies and be more creative when constructing passphrases (e.g., mangling passphrases Labrande, 2015, randomly choosing words Bonneau & Shutova, 2012, ignoring grammatical structures Rao et al., 2013; Bonneau & Shutova, 2012, etc.).

5.3 Future Work

In the future, we plan to extend the LSTM models to predict password pairs that are semantically similar (e.g., synonyms, associated words). However, the prediction of these passwords was out of the scope of this research. We may also improve the password prediction by training the model on a synthetic balanced dataset that contains various under-represented modification patterns found in this research (e.g., common names). We want to build a neural network model to classify the passwords that involve more than one type of modification pattern (e.g., Capitalization and Leet, Substring and Leet, etc.). Lastly, we would like to introduce and define new metrics to calculate a password or passphrase's intrinsic security value. The score can be based on entropy and other factors such as the number of attempts to predict and the name of the best suitable model, the use of resources to generate predictions, commonality in contemporary language, its prevalence online, semantic similarity, presence in the leaked datasets, and several other factors. It can provide a more detailed understanding of passphrase security than the current entropy score. This score can help design systems that make security recommendations and lower the crackability score of user-generated passwords and passphrases while keeping them memorable.

6 Conclusion

In this research, we first investigated the problem of targeted online password guessing. We built a password prediction pipeline using RNNs to automate password categorization and generation. The prediction results were superior to those of traditional classification and guessing algorithms. The performance boost was especially significant when we limited guessing attempts to five. We combined the understanding of rule-based prediction algorithms and the power of LSTM neural networks to solve the problem of cross-site prediction for passwords created by the same user. It is a relatively new approach and perhaps one of the first attempts to use RNNs for this specific task. We could quantify the correlation between the similarity, modification patterns, and predictability of subsequent passwords. In addition, we demonstrated the ease of prediction and high accuracy of the most common modification strategies, such as adding head or tail symbols to the original password or Capitalization. We showcased that such a prediction process could be facilitated by affordable hardware or online computing resources, such as Google Colab, due to the low complexity and shallow nature of the RNN model. In addition, prediction efficiency allows it to run on platforms where five or fewer attempts are allowed before an account gets locked. We also discussed the concrete steps the online services should take to improve the security of the authentication process.

We then explored the problem of targeted offline passphrase guessing using RNNs. We built an attention-based LSTM model and a fine-tuned GPT-2 model to predict commonly used passphrases. We analyzed the results and compared them to the most utilized methods, such as dictionary-based, rule-based, and Markov Chain-based prediction algorithms. We achieved a significantly better prediction rate, especially considering the number of attempts used, time to deploy, resource utilization, and ease of operation. Even though these approaches are not radically new, they are novel in the passphrase prediction domain and yield a competitive prediction rate.

Funding No funding was received to assist with the preparation of this manuscript.

Declarations

A preliminary version of this article was presented at SKM '21 (Nosenko et al., 2021).

Financial and non-financial interests The authors have no relevant financial or non-financial interests to declare that are relevant to the content of this manuscript.

References

Blanchard, N.K., Malaingre, C., & Selker, T. (2018). Improving security and usability of passphrases with guided word choice. In *34th annual computer security applications conference* (pp. 723–732)

Bonneau, J., & Shutova, E. (2012). Linguistic properties of multi-word passphrases. In *International conference on financial cryptography and data security* (pp. 1–12). Springer

Brostoff, S., & Sasse, M.A. (2003). Ten strikes and you're out": Increasing the number of login attempts can improve password usability. In *CHI 2003 workshop on human-computer interaction and security systems*

Burr, W., Dodson, D., Perlner, R., Gupta, S., & Nabbus, E. (2013). NIST SP800-63-2: Electronic authentication guideline. Technical report, National Institute of Standards and Technology, Reston, VA

Cho, K., Van Merriënboer, B., Bahdanau, D., & Bengio, Y. (2014). On the properties of neural machine translation: Encoder-decoder approaches. arXiv:1409.1259

Das, A., Bonneau, J., Caesar, M., Borisov, N., & Wang, X. (2014). The tangled web of password reuse. In: *NDSS* (Vol. 14, pp. 23–26)

Davies, M. (2009). The 385+ million word corpus of contemporary American English (1990–2008+): Design, architecture, and linguistic insights. *International Journal of Corpus Linguistics, 14*(2), 159–190.

Florencio, D., & Herley, C. (2007). A large-scale study of web password habits. In *16th International conference on World Wide Web* (pp. 657–666)

Fu, C., Duan, M., Dai, X., Wei, Q., Wu, Q., & Zhou, R. (2021). Densegan: A password guessing model based on densenet and passgan. In *International conference on information security practice and experience* (pp. 296–305). Springer

Grassi, P. A., Garcia, M. E., & Fenton, J. L. (2017). DRAFT NIST SP800-63-3 digital identity guidelines. Technical report, National Institute of Standards and Technology, Los Altos, CA

Han, W., Xu, M., Zhang, J., Wang, C., Zhang, K., & Wang, X. S. (2020). TransPCFG: transferring the grammars from short passwords to guess long passwords effectively. *IEEE Transactions on Information Forensics and Security, 16,* 451–465.

Haque, S.T., Wright, M., & Scielzo, S. (2013). A study of user password strategy for multiple accounts. In *3rd ACM conference on data and application security and privacy* (pp. 173–176)

Hardeniya, N. (2015). NLTK Essentials, p. 28. Packt Publishing Ltd

Hitaj, B., Gasti, P., Ateniese, G., & Perez-Cruz, F. (2019). Passgan: A deep learning approach for password guessing. In *International conference on applied cryptography and network security* (pp. 217–237). Springer

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

Huth, A., Orlando, M., & Pesante, L. (2012). Password security, protection, and management. United States Computer Emergency Readiness Team

Joudaki, Z., Thorpe, J., & Martin, M.V. (2018). Reinforcing system-assigned passphrases through implicit learning. In *2018 ACM conference on computer and communications security* (pp. 1533–1548)

Keith, M., Shao, B., & Steinbart, P. (2005). The effectiveness and usability of passphrases for authentication. In *11th Americas conference on information systems* (pp. 3354–3357)

Kingma, D.P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv:1412.6980

Kouretas, I., & Paliouras, V. (2019). Simplified hardware implementation of the softmax activation function. In *2019 8th international conference on modern circuits and systems technologies (MOCAST)* (pp. 1–4). IEEE

Kuo, C., Romanosky, S., & Cranor, L.F. (2006). Human selection of mnemonic phrase-based passwords. In *Second symposium on usable privacy and security (SOUPS)* (pp. 67–78)

Kurzban, S. A. (1985). Easily remembered passphrases: a better approach. *ACM SIGSAC Review, 3*(2–4), 10–21.

Labrande, H. (2015). Crack me I'm famous: cracking weak passphrases using publicly-available sources. In *2015 Information and Communications Technology Security Symposium (SSTIC)* (pp. 479–484).

Li, H., Chen, M., Yan, S., Jia, C., & Li, Z. (2019). Password guessing via neural language modeling. In *Proceedings of the international conference on machine learning for cyber security* (pp. 78–93). Springer

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

Melicher, W., Ur, B., Segreti, S.M., Komanduri, S., Bauer, L., Christin, N., & Cranor, L.F. (2016). Fast, lean, and accurate: Modeling password guessability using neural networks. In *25th USENIX security symposium* (pp. 175–191)

Murray, H., & Malone, D. (2018). Exploring the impact of password dataset distribution on guessing. In: *2018 16th annual conference on privacy, security and trust (PST)* (pp. 1–8). IEEE

Narayanan, A., & Shmatikov, V. (2005). Fast dictionary attacks on passwords using time-space tradeoff. In *12th ACM conference on computer and communications security* (pp. 364–372)

Nosenko, A., Cheng, Y., & Chen, H. (2021). Learning password modification patterns with recurrent neural networks. In *International conference on secure knowledge management in artificial intelligence era* (pp. 110–129). Springer

Notoatmodjo, G., & Thomborson, C. (2009). Passwords and perceptions. In: *Seventh Australasian conference on information security* (pp. 71–78)

Pasquini, D., Gangwal, A., Ateniese, G., Bernaschi, M., & Conti, M. (2021). Improving password guessing via representation learning. In *2021 42nd IEEE symposium on security and privacy* (pp. 1382–1399). IEEE

Pennington, J., Socher, R., & Manning, C.D. (2014). Glove: Global vectors for word representation. In *2014 conference on empirical methods in natural language processing (EMNLP)* (pp. 1532–1543)

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. (2019). Language models are unsupervised multitask learners. *OpenAI blog, 1*(8), 9.

Rao, A., Jha, B., & Kini, G. (2013). Effect of grammar on security of long passwords. In *Third ACM conference on data and application security and privacy* (pp. 317–324)

Rawlings, R. (2020). Password Habits in the US and the UK: This Is What We Found.https://nordpass.com/blog/password-habits-statistics/. Accessed 26 July 2022.

Schumacher, M., Roßner, R., & Vach, W. (1996). Neural networks and logistic regression: Part I. *Computational Statistics & Data Analysis, 21*(6), 661–682.

Sparell, P., & Simovits, M. (2016). Linguistic cracking of passphrases using Markov chains. Cryptology ePrint Archive

Stobert, E., & Biddle, R. (2014). The password life cycle: user behaviour in managing passwords. In: *10th symposium on usable privacy and security (SOUPS)* (pp. 243–255)

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., & Polosukhin, I (2017). Attention is all you need. *Advances in Neural Information Processing Systems, 30*

Von Zezschwitz, E., De Luca, A., & Hussmann, H. (2013). Survival of the shortest: A retrospective analysis of influencing factors on password composition. In *IFIP Conference on Human-Computer Interaction* (pp. 460–467). Springer

Walia, K.S., Shenoy, S., & Cheng, Y. (2020). An empirical analysis on the usability and security of passwords. In *2020 21st IEEE international conference on information reuse and integration for data science (IRI)* (pp. 1–8). IEEE

Wang, C., Jan, S.T., Hu, H., Bossart, D., & Wang, G. (2018). The next domino to fall: Empirical analysis of user passwords across online services. In *8th ACM conference on data and application security and privacy* (pp. 196–203)

Wang, D., Zhang, Z., Wang, P., Yan, J., & Huang, X. (2016). Targeted online password guessing: An underestimated threat. In *2016 ACM conference on computer and communications security* (pp. 1242–1254)

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

Woo, S.S., & Mirkovic, J. (2016). Improving recall and security of passphrases through use of mnemonics. In *10th International conference on passwords*

Xie, Z., Zhang, M., Yin, A., & Li, Z. (2020). A new targeted password guessing model. In *Australasian conference on information security and privacy* (pp. 350–368). Springer

Xu, M., Wang, C., Yu, J., Zhang, J., Zhang, K., & Han, W. (2021). Chunk-level password guessing: Towards modeling refined password composition representations. In *2021 ACM conference on computer and communications security* (pp. 5–20)

Xu, G., Meng, Y., Qiu, X., Yu, Z., & Wu, X. (2019). Sentiment analysis of comment texts based on BiLSTM. *IEEE Access, 7,* 51522–51532.

Yoo, J.Y., Morris, J.X., Lifland, E., & Qi, Y. (2020). Searching for a search method: Benchmarking search algorithms for generating NLP adversarial examples. arXiv:2009.06368

Zhang, Y., Monrose, F., & Reiter, M.K. (2010). The security of modern password expiration: An algorithmic framework and empirical analysis. In *17th ACM conference on computer and communications security* (pp. 176–186)

Alex Nosenko is an Application Systems Analyst for the Santa Clara County Office of Education. He received his MS degree in Computer Science from California State University, Sacramento, in 2022. His research interests include data engineering and cybersecurity.

Yuan Cheng is an Associate Professor in the Department of Computer Science at California State University, Sacramento. He obtained a Ph.D. in Computer Science from the University of Texas at San Antonio in 2014. He also received a B.Eng. degree in Information Security from Huazhong University of Science and Technology in 2008. His research interests include access control, authentication, and security and privacy issues in emerging technologies.

Haiquan Chen is an Associate Professor in the Department of Computer Science at California State University, Sacramento. Previously, he was an Assistant/Associate Professor with the Department of Computer Science at Valdosta State University, Valdosta, GA, from 2011 to 2017. He obtained his Ph.D. degree in Computer Science from Auburn University in 2011. His research interests embrace database and big data management, with emphasis on data mining, machine learning, location-based services, and social network analysis.