PASS2EDIT: A Multi-Step Generative Model for Guessing Edited Passwords

Ding Wang, Yunkai Zou
Nankai University
{wangding, zouyunkai}@nankai.edu.cn

Yuan-An Xiao
Peking University
xiaoyuanan@pku.edu.cn

Siqi Ma
The University of New South Wales
siqi.ma@unsw.edu.au

Xiaofeng Chen
Xidian University
xfchen@xidian.edu.cn

Abstract

While password *stuffing* attacks (that exploit the direct password reuse behavior) have gained considerable attention, only a few studies have examined password *tweaking* attacks, where an attacker exploits users' indirect reuse behaviors (with edit operations like insertion, deletion, and substitution). *For the first time*, we model the password tweaking attack as a multi-class classification problem for characterizing users' password edit/modification processes, and propose a generative model coupled with the *multi-step decision-making* mechanism, called PASS2EDIT, to accurately characterize users' password reuse/modification behaviors.

We demonstrate the effectiveness of PASS2EDIT through extensive experiments, which consist of 12 practical attack scenarios and employ 4.8 billion real-world passwords. The experimental results show that PASS2EDIT and its variant significantly improve over the prior art. More specifically, when the victim's password at site A (namely pw_A) is known, within 100 guesses, the cracking success rate of PASS2EDIT in guessing her password at site B ($pw_B \neq pw_A$) is 24.2% (for common users) and 11.7% (for security-savvy users), respectively, which is 18.2%-33.0% higher than its foremost counterparts. Our results highlight that password tweaking is a much more damaging threat to password security than expected.

1 Introduction

Text passwords are the most prevalent method of user authentication and play an important role in the daily digital lives of today's 5 billion Internet users. Although password-based authentication has some intrinsic security and usability issues (e.g., guessing [33, 60], stuffing [59] and typo [54]), and many alternative authentication technologies (e.g., hardware security key [38], single-sign-on [40], and behavior biometrics [44]) have also been successively proposed, passwords will remain their status as the most widely used authentication method in the foreseeable future due to its simplicity to use, easiness to change and low cost to deploy [12, 13]. This consensus has gradually been reached in both academia [13, 27, 76] and industry [9, 14, 64].

Researchers have reported for decades that a large majority of users, despite good-faith efforts in their information security, struggled to create secure passwords [41, 51, 61]. To address this issue, many service providers have enforced strict password policies, such as restricting the minimum length and the character composition [35, 41, 69]. Besides, current password guidelines suggest that users should create distinct passwords, especially for systems and accounts across different levels of importance [1, 4, 63] (e.g., news subscription accounts and financial accounts). However, the number of accounts a user needs to manage is constantly increasing, and typical Internet users are reported to have 80-107 distinct online accounts [25, 45, 51]. As the memory capacity of the human brain remains stable, users are very likely to cope by reusing existing passwords across different sites.

Password reuse poses a serious security vulnerability: Attackers who compromise one site are likely to compromise other services protected by the same or slightly edited/modified password [53]. The recent large-scale password leaks (e.g., the 3 billion Yahoo [2], 10.88 billion CAM4 [5], and 3.2 billion COMB [6]) do provide ample materials for attackers to conduct cross-site guessing attacks. For example, the 2022 DBIR report [7] shows that there are 4,751 data breaches due to basic web application attacks and "over 80% of the breaches in this pattern can be attributed to stolen credentials" (i.e., password stuffing attacks), and there has been an almost 30% increase in credential stuffing since 2017. The 2022 IBM annual data breach report [8] reveals that compromised credentials are the most common initial attack vector, which is responsible for 19% of breaches at an average breach cost of USD 4.91 million. Some companies (e.g., [17]) even purchase compromised credentials from the darknet market to actively confirm their vulnerable accounts.

Worse still, attackers can also exploit the victim's existing password at one service to guess a *different* password created by the same user at another service. Such attacks that exploit users' password *indirect* reuse behaviors are called *credential tweaking* [46]. Research [18, 51, 67, 68, 71] reveals that 21%-33% of users slightly edit/modify existing passwords when creating passwords for their new accounts.

A few studies [18, 46, 71] have investigated credential tweaking attacks. However, this threat is still largely underestimated, because how to model/characterize users' password reuse behaviors looks deceptively simple, but actually, it is rather challenging. Here we explain why.

If we model users' password modification processes as a series of atomic edit operations (e.g., deletion or insertion of a specific password character), and employ a neural network to predict the sequence of edit operations, then each edit step may have a certain impact on the subsequent edit steps. For example, suppose we modify a password pw_A=`wang123` to pw_B=`wang1!`, then the edit operation set from pw_A to pw_B is {(DEL,5), (DEL,6), (INS,7,!), EOS}, where (DEL,5) means deleting the character `2` at the sixth position of pw_A, (INS,7,!) means inserting an `!` at the eighth position, and EOS represents to terminate the edit process. Note that after the edit operation (DEL,5), the original password `wang123` has *already* been modified to `wang13` and similar situations occur in subsequent editing operations, but the existing password reuse-based models (e.g., the state-of-the-art Pass2Path [46]) cannot capture such critical changes. How to establish a direct connection between the edit operations and the corresponding edit effects is not straightforward for "neural networks".

As various security mechanisms (e.g., rate-limiting and lockout [20]) have been employed by 65% of top sites (see [37]) to prevent a large number of online guessing attempts, password guessing should be effective even when only allowed a small number of guesses (e.g., 100 by NIST 800-63B [23]). How to automatically prioritize password modification behaviors in a personalized manner and fit them in the limited guesses is challenging. To address both challenges, we investigate credential tweaking attacks from a data-driven perspective, and for the first time, model users' password reuse processes as a multi-classification problem (which is essentially different from the sequence to sequence-based model employed by Pass2Path [46]). Fig. 1 provides a high-level view of our guessing model. We call this training mechanism multi-step decision-making. The resulting model is denoted as password-to-edit (i.e., PASS2EDIT), where edit represents not only one step of edit operations but also the edited passwords (i.e., modified/reused passwords).

Our PASS2EDIT (and TarGuess-II [71]) exploits not only users' vulnerable behaviors of password reuse but also choosing popular passwords, and is very effective. Particularly, within 100 guesses, our PASS2EDIT with *no* consideration of users' behavior of choosing popular passwords (i.e., when only considering users' password reuse behavior, denoted as PASS2EDIT-nomix) outperforms the state-of-the-art Pass2Path [46] by 43.39% against common users and by 18.46% against security-savvy users. Furthermore, we consider users' vulnerable behaviors of choosing popular passwords and further improve the success rate of our model by 24.19% in most (10 out of 12) attack scenarios.

There have been dozens of metrics that measure the similarity of strings, e.g., Das et al. [18] and Guo et al. [24] employed edit distance and cosine similarity, respectively, to measure password similarity. Still, to our knowledge, previous research on password guessing (see [46, 71]) have invariably used the canonical metric (i.e., edit distance) to figure out the reused password pairs, and whether other metrics are more effective for password guessing is unknown. Fortunately, in this work, we, for the first time, find that cosine similarity can be more suitable than edit distance for guessing.

1.1 Related work

At NDSS'14, Das et al. [18] proposed the first cross-site password-guessing algorithm, which applies eight transformation rules (namely mangling rules, e.g., insertion, deletion, capitalization, etc.) in a pre-defined order to generate candidate passwords based on one existing passwords of the same user. Although this algorithm's attack success rate outperforms trawling guessing algorithms under a small number of guesses, it has some inherent limitations: It assumes that all users select password transformation rules in a fixed priority, which cannot capture users' complex modification behaviors.

At ACM CCS'16, Wang et al. [71] proposed a probabilistic context-free grammar-based (PCFG [74]) password reuse model, named TarGuess-II, which significantly outperforms that of [18]. Although TarGuess-II is based on a strict statistical model, it considers only six types of structure-level transformation rules, which is quite heuristic. Besides, the inherent limitations of PCFG (such as the weak generalization ability) are difficult to overcome.

At IEEE S&P'19, Pal et al. [46] introduced deep learning techniques to characterize users' password reuse behaviors. More specifically, they trained a sequence-to-sequence (seq2seq) model [58] to predict the modifications needed to transform an existing password into its sister passwords, and achieved the state-of-the-art guessing success rate on large-scale datasets (i.e., the 1.4 billion-sized 4iQ dataset [3]). However, this model (named Pass2Path [46]) is still not optimal: (1) It cannot capture the mutual influence between password edit operations and corresponding transformation effects; (2) Its character substitution operation defined often does not conform to the semantics of password modification; (3) It does not consider the usage of popular passwords (such as `123456789` and `password123`). See more details in Appendix A of our full version paper at `https://bit.ly/3ZjHPaD`.

1.2 Our contributions

The contributions of this work are as follows:

- **Multi-step decision-making mechanism**. In order for the neural network to learn the *reaction* of one-step edit operations to the original password, we propose a targeted password guessing model, called PASS2EDIT-nomix, which *for the first time*, introduces a multi-step

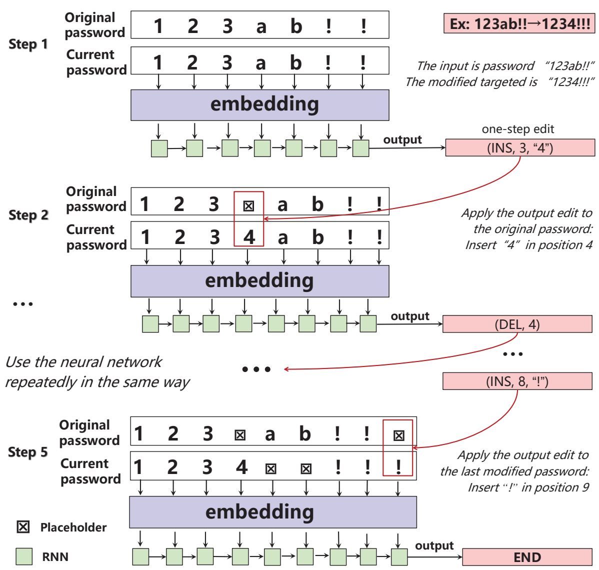


Figure 1: An example of our multi-step decision-making training mechanism. Suppose the original password is `123ab!!`, the target password is `1234!!!`, and the edit operation sequence between this password pair is [(Insert "4" in the position 4), (Delete "a" in position 5), (Delete "b" in position 6), (Insert "!" in position 9), End]. Then, the input of the neural network is both the original password and the currently modified password, and the output is the next one-step edit operation. Here the placeholder is to align the length of the two passwords so that they can be input into the neural network.

decision-making training mechanism to more accurately and practically characterize users' password reuse behaviors. To further exploits users' vulnerable behavior of choosing popular passwords, we have explored a number of methods and preferred the simple yet effective method of mixing globally popular password dictionaries, resulting in our final fully-fledged model PASS2EDIT.

- **Extensive evaluation**. Extensive experiments on 11 large real-world password datasets demonstrate the effectiveness of PASS2EDIT and its variant PASS2EDIT-nomix. Particularly, within 100 guesses, PASS2EDIT-nomix outperforms its foremost counterpart (i.e., Pass2Path [46]) by 36.67% on average, and this value is 35.84% if both further consider users' vulnerable behaviors of choosing popular passwords. Besides, we investigate the passwords independently cracked by PASS2EDIT and its counterparts, and summarize their similarities and differences in terms of length, character composition, structure, and complexity.
- **Some insights**. We introduce a 2-gram cosine similarity metric for password guessing. We show that cosine similarity is more effective than edit distance in most attack scenarios when used as a filter metric. Specifically, after the training set is filtered out by cosine similarity (>0.3), the cracking success rate of PASS2EDIT is 9% higher than using edit distance ($\leq$4) at 1,000 guesses. In addition, we find that in the process of multi-step training, both the current modified password and the existing password can help predict the next edit operation.

2 Background

Now we briefly introduce the background of users' password reuse behaviors and the corresponding guessing attacks.

2.1 Password reuse behaviors

Given the limited cognitive capacity of the human brain, users inevitably reuse or modify/edit their existing passwords across different accounts [57]. In 2007, Florencio and Herley [21] published the first large-scale study of password use and reuse behaviors, and confirmed that reused and poor-strength passwords are a frequent flaw. At NDSS'14, Das et al. [18] highlighted the problem of password reuse by conducting a large-scale data collection through websites. Since then, a number of successive studies have been conducted. For example, Wash et al. [73] carried out a six-week investigation on the password security practices of 134 participants and found that users do tend to reuse passwords, especially those relatively complex and frequently used. Similar to password creation, password reuse is also affected by different factors.

At CCS'17, Pearman et al. [51] observed that the usage of symbols and digits in passwords increases the possibility of reuse behaviors, while password managers have few impacts on password reuse. To protect users from credential-stuffing attacks, researchers have proposed countermeasures from various aspects. For example, Golla et al. [22] discussed some best practices for designing password-reuse notifications; Wang et al. [68] and Pal et al. [46] designed new password strength meters based on password reuse behaviors; Thomas et al. [59] proposed a privacy-preserving protocol that allows users to query whether their login credentials were exposed.

2.2 Password guessing attacks

In a broad sense of natural language processing (NLP), the password generation process can be regarded as a character-level language modeling problem. At CCS'05, Narayanan and Shmatikov [43] first introduced the Markov model into password guessing to improve the dictionary-based cracking tools. This algorithm trains all characters in a password, and calculates the probability of each password through the connection between the characters from left to right. At IEEE S&P'09, Weir et al. [74] proposed a password model based on Probabilistic Context Free Grammar (PCFG), which can automatically learn users' password generation behaviors by dividing the password into different character segments.

Subsequently, a number of successive studies were conducted to improve the attack efficiency and success rate of these two models (e.g., [19, 30, 39, 66]). At USENIX SEC'16, Melicher et al. [42] first introduced deep learning techniques to password guessing and trained a language model with Recurrent Neural Networks (RNNs). Since then, various deep generative models have been applied to password guessing, such as generative adversarial networks (e.g., PassGAN [28]) and conditional/dynamic password guessing frameworks (i.e., CPG/DPG [50]). In addition, some studies (e.g., TarGuess-I [71]) further incorporate personally identifiable information (PII) into the password model, which greatly improves the guessing success rates of trawling password models.

3 PASS2EDIT: A targeted guessing model for password reuse

To characterize users' password reuse behaviors, we first introduce the multi-step decision-making training mechanism. Then, we build our neural network and propose to use the cosine similarity to measure password similarity.

3.1 Modeling password reuse behaviors

As mentioned in Sec. 1.1, there are inherent limitations in existing password models (e.g., the mutual influence issue in Pass2Path [46]; see details in Appendix A at `https://bit.ly/3ZjHPaD`), so we propose a new targeted password model.

We treat password modification as a series of continuous edit operations. When giving a training set with password pairs $\langle pw_A, pw_B \rangle$, the edit sequence $t = t_1, t_2, ..., \text{EOS}$ of each pair from pw_A to pw_B can be obtained by dynamic programming of the edit distance matrix (from pw_A to pw_B), where EOS stands for the end symbol. Unlike Pass2Path [46], our atomic operations only include insertion and deletion operations, without substitution operations, that is: $t = \{(\textsc{Ins}, p, c) | p \in \mathbb{Z}^*, c \in \Sigma\} \cup \{(\textsc{Del}, p) | p \in \mathbb{Z}^*\} \cup \{\text{EOS}\}$, where p and c stand for the position and the inserted character, respectively. This is because substitution can be completely replaced by deletion and insertion. Hence, $\overline{(\textsc{Sub}, p, c)}$ can be demonstrated as $\overline{(\textsc{Del}, p), (\textsc{Ins}, p, c)}$.

In addition, model training is more efficient by excluding the SUB operation because the number of atomic operations is reduced. For example, if we limit the maximum length of the training passwords to 29, removing the SUB operation can reduce at least 29∗47 atomic operation classes (where 47=48 types of EN-US keyboard characters subtract the substituted character itself), thus greatly improving the training and generation efficiency. Also, it is more realistic to fit the scenario of modifying an existing password. For example, if the trained Path2Path model [46] uses the pw_A=`wang123` to generate pw_B=`wang1!`, then a SUB operation ($\textsc{Sub}$, 5, '!') will be required first (i.e., digit `2` in the sixth position is substituted with symbol `!`), and then it deletes character `3` at the end. However, what the user actually does could be first to delete digits `2` and `3`, and then add an `!` to the end.

We agree on the order of atomic operations as follows:

- The EOS operation must be at the end of the sequence, indicating the end of the modification.
- Other edit operations must be sorted in an ascending order of the character position (i.e., p).
- When two operations are conducted at the same position, we make the operations INS prior to DEL because $(\textsc{Ins}, p, c)$ means inserting before position p.

To make the model learn the reaction of transformation t to password pw when the modification is relatively complex, we make a multi-step decision. The input of the model is $\begin{pmatrix} pw^{\text{orig}} \\ pw^{\text{cur}} \end{pmatrix}$, where pw^{orig} and pw^{cur} respectively represent the original password and the current password generated by the previous transformation steps. The output of the model is the next atomic transformation t_i. After the model outputs t_i, we *apply* this transformation to the input. That is $\begin{pmatrix} \tilde{pw}^{\text{orig}} \\ pw_i^{\text{cur}} \end{pmatrix} = \text{apply}\left(t_i, \begin{pmatrix} pw^{\text{orig}} \\ pw_{i-1}^{\text{cur}} \end{pmatrix}\right) (i \geq 1)$, where $\tilde{pw}^{\text{orig}}$ represents the original password pw^{orig} with the corresponding placeholder and $pw_0^{cur} = pw^{orig}$. Since the INS and DEL operations will make the lengths of pw^{orig} and pw_i^{cur} no longer equal, we align them by inserting placeholders ⊠. That is,

$$\text{apply}\left((\textsc{Ins}, p, c), \begin{pmatrix} \overline{c_0^{\text{orig}} \ldots c_{n-1}^{\text{orig}}} \\ \overline{c_0^{\text{cur}} \ldots c_{n-1}^{\text{cur}}} \end{pmatrix}\right) = \begin{pmatrix} \overline{c_0^{\text{orig}} \ldots c_{p-1}^{\text{orig}} \boxtimes c_p^{\text{orig}} \ldots c_{n-1}^{\text{orig}}} \\ \overline{c_0^{\text{cur}} \ldots c_{p-1}^{\text{cur}}\ c\ c_p^{\text{cur}} \ldots c_{n-1}^{\text{cur}}} \end{pmatrix}$$

$$\text{apply}\left((\textsc{Del}, p), \begin{pmatrix} \overline{c_0^{\text{orig}} \ldots c_{n-1}^{\text{orig}}} \\ \overline{c_0^{\text{cur}} \ldots c_{n-1}^{\text{cur}}} \end{pmatrix}\right) = \begin{pmatrix} \overline{c_0^{\text{orig}} \ldots c_{p-1}^{\text{orig}} c_p^{\text{orig}} c_{p+1}^{\text{orig}} \ldots c_{n-1}^{\text{orig}}} \\ \overline{c_0^{\text{cur}} \ldots c_{p-1}^{\text{cur}} \boxtimes c_{p+1}^{\text{cur}} \ldots c_{n-1}^{\text{cur}}} \end{pmatrix},$$

where c_i^{orig} and c_i^{cur} are each single character of password pw^{orig} and pw^{cur}, respectively. Formally, given a user's existing password pw_A, we define the conditional probability of generating a new password pw_B as follows:

$$P(pw_B | pw_A) = \prod_{t_i \in t_{pw_A \to pw_B}} P(t_i | pw^{orig}, pw_{i-1}^{cur}),$$

where $t_{pw_A \to pw_B}$ is an ordered set of transformation operations from pw_A to pw_B and $pw_0^{cur} = pw^{orig}$.

Considering the password pairs of pw_A=`wang123` and pw_B=`wang1!` as an example, the transformation set is {($\textsc{Del}$,5), ($\textsc{Del}$,6), ($\textsc{Ins}$,7,`!`), EOS}, and the process of transforming pw_A to pw_B can be demonstrated as:

$$\begin{aligned} P(pw_B|pw_A) = & P\left((\textsc{Del},5) \middle| \begin{pmatrix} \texttt{wang123} \\ \texttt{wang123} \end{pmatrix}\right) \\ & * P\left((\textsc{Del},6) \middle| \begin{pmatrix} \texttt{wang123} \\ \texttt{wang1}\boxtimes\texttt{3} \end{pmatrix}\right) \\ & * P\left((\textsc{Ins},7,!) \middle| \begin{pmatrix} \texttt{wang123} \\ \texttt{wang1}\boxtimes\boxtimes \end{pmatrix}\right) \\ & * P\left(\text{EOS} \middle| \begin{pmatrix} \texttt{wang123}\boxtimes \\ \texttt{wang1}\boxtimes\boxtimes\texttt{!} \end{pmatrix}\right). \end{aligned}$$

Finally, we take the last transformed pw^{cur} that has undergone the transformation operation (i.e., `wang1⊠⊠!`) as the final generated password and further remove the placeholders.

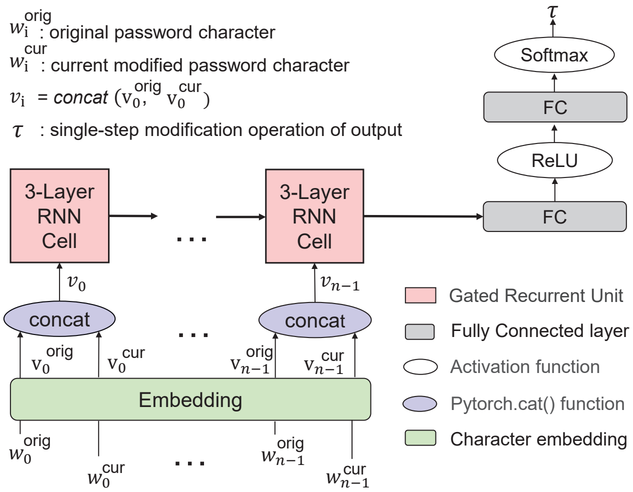


Figure 2: The neural network architecture of our PASS2EDIT. It consists of 3-GRU layers and two fully connected layers, and it is essentially a classifier, where the input is the original and currently modified password pair, and the output is the classification of the single-step modification.

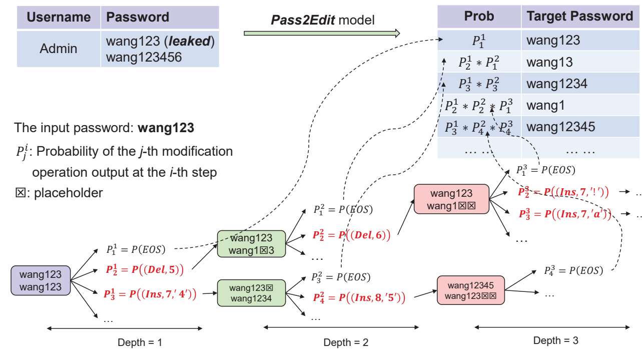


Figure 3: Example of generating reused passwords with beam search algorithm. Here, we suppose the original password is pw = `wang123`, and the beam width is $k = 2$. The red P indicates the probability of the most feasible k valid modification operations in each round (the product of the probability from the root to the current node), and the dotted arrow indicates the process of summarizing the paths to get the final password guess set.

Since Pal et al. [46] showed that the key-sequence representation of passwords performs better when capturing capitalization-related transforms, we consider the caps-lock and shift key on the keyboard when processing characters. Specifically, after each password is transformed into a key sequence, the character set Σ includes 48 types of characters that can be entered through the EN-US standard keyboard, as well as $\langle$shift$\rangle$, $\langle$caps$\rangle$ and $\boxtimes$ (48+3=51). If we limit the length of the password to no more than 30 (i.e., $0 \le p < 30$), then the total number of atomic operations is $|t| = 30*51+30+1 = 1,561$, where $30*51$ is the category # of insertions, 30 is the category # of deletions, and 1 represents the EOS operation. In this light, our one-step prediction process can essentially be seen as a 1,561-class multi-classification problem.

3.2 Neural network building

For sequence tasks with varying lengths, Recurrent Neural Network (RNN) is a commonly used neural network structure, and there are two classical variants: Long Short-Term Memory (LSTM) [29] alleviates the vanilla RNN network's gradient vanishing/explosion problems; Gated Recurrent Unit (GRU) [16] improves the LSTM's calculation efficiency while achieving similar performance. Thus, we use GRU as the basic unit to build our neural network.

As shown in the Fig. 2, the input of the neural network is the password pair (i.e., the original password pw^{orig} and the current password pw^{cur}), and the output is the probability of each transformation state t_i. Firstly, the input passes through the embedding layer, and each one-hot encoded password character is converted into a 256-dimensional vector (i.e., v_i^{orig} and v_i^{cur}). Secondly, we concatenate v_i^{orig} and v_i^{cur} (using the `Pytorch.cat()` function) into v_i and then input it to a 3-layer GRU (the hidden layer dimension is 256). Thirdly, we take the output of the GRU for the last character through a 2-layer FC (i.e., fully connected layer, where the hidden layer dimension is 512), and finally obtain the probability of each transformation t_i through the softmax layer.

Since the neural network we build (see Fig. 2) is essentially a classifier, we use the cross-entropy of the predicted output and the ground truth as the loss function during training, and use Adam optimizer [34] with weight delay strategy to minimize the cross-entropy loss. To alleviate overfitting [55], we set the dropout rate to 0.4. That is, the output of 40% neurons is randomly set to zero during training. See more details of parameter tunings in Sec. 4.5.

When generating reused passwords, we use the beam search algorithm, which is one of the most popular search strategies in NLP. More specifically, we first input pw^{orig} and pw_0^{cur} into the neural network to obtain the probability distribution of the first-step transformation. Then the algorithm selects the top k valid transformations $(t_1, ..., t_k)$ in probability except EOS (valid means that the atomic operations meet the order we agreed in advance), and applies them to the original password input. Here, $\begin{pmatrix} \tilde{pw}^{orig} \\ pw_i^{cur} \end{pmatrix} = \text{apply}\left(t_i, \begin{pmatrix} pw^{orig} \\ pw_{i-1}^{cur} \end{pmatrix}\right)$. We take these intermediate transformations as input for the next round to execute the neural network iteratively. The whole process lasts for several (no more than the beam depth, i.e., a parameter we set in advance) rounds, and the output of each step of the model forms a tree structure, as shown in Fig. 3. We then summarize all the paths ending with the EOS symbol on the tree to get the final guess set.

Note that the input password modified by different transformation sequences may get the same passwords, so the guess set obtained needs to remove the duplicated passwords (we also remove the passwords that are same as the original password because the original password is always the first choice for targeted guessing [71]). Finally, we sort by the total probability in descending order to get the final output.

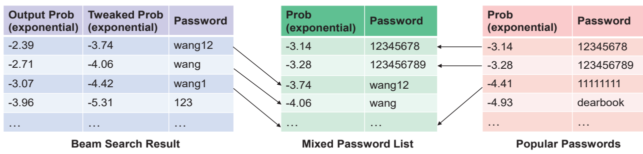


Figure 4: Mix popular passwords on the guess set of our PASS2EDIT. Note that the probability value in this figure is after taking the logarithm.

3.3 Mixing popular passwords

Researches on users' password reuse behaviors [18, 51, 67, 68, 71] show that about 21%-33% of users tend to (slightly) modify their existing passwords when creating new passwords, and about 20%-59% of users tend to directly reuse their existing passwords. While for the rest users, they are likely to create a new password that is not related to their existing password (e.g., just using a popular password, see Table 3). Thus, it is desirable to make password model like Pass2Path [46] and PASS2EDIT-nomix have the ability to characterize users' vulnerable behaviors of choosing popular passwords.

Inspired by TarGuess-II [71], we adopt the method of mixing globally popular passwords, and this practice helps us achieve satisfactory results (see Fig. 4). More specifically, for our guess set output by PASS2EDIT-nomix, we multiply the probability of each password by a factor α, which stands for the fraction of users who do *not* choose popular passwords (e.g., about 0.3 in most of our datasets); For the set of popular passwords, we use the frequency of each password in it to estimate its probability. Then, we merge the two password sets in descending order of probability as the final guess set.

3.4 Cosine similarity metric

Previous studies [18, 46, 71] have invariably used edit distance as the metric of password similarity. For example, Pal et al. [46] used password pairs only with an "edit distance≤4" for training to avoid the negative impacts of futile/distant password pairs. However, this measurement method is not sufficiently accurate to filter out dissimilar password pairs. For example, the minimum edit distances of the following four password pairs are all six: $\langle\texttt{3080124},\texttt{cooper3080124}\rangle$, $\langle\texttt{720710},\texttt{720710720710}\rangle$, $\langle\texttt{wozuixiao},\texttt{leizixi1}\rangle$, and $\langle\texttt{123456789},\texttt{281456}\rangle$. The first two pairs are typical reused passwords, while the latter two are not at all. To filter out dissimilar password pairs more accurately, we introduce the 2-gram cosine similarity as the metric. The similarity between pw_A and pw_B is defined as:

$$\mathrm{sim}(pw_A, pw_B) = \frac{\sum\limits_{g\in\mathbb{G}} \left(\mathrm{count}\left(pw_A, g\right) * \mathrm{count}\left(pw_B, g\right)\right)}{\sqrt{\sum\limits_{g\in\mathbb{G}} \mathrm{count}^2\left(pw_A, g\right)}\ \sqrt{\sum\limits_{g\in\mathbb{G}} \mathrm{count}^2\left(pw_B, g\right)}},$$

where $\mathbb{G}$ is the set of all 2-gram substrings in pw_A and pw_B, and $\mathrm{count}(pw, g)$ represents the number of occurrences of substring g in the password pw. For example, the 2-gram set of password abc is $\{\overline{\mathrm{SOS}a}, \overline{ab}, \overline{bc}, \overline{c\mathrm{EOS}}\}$, and the 2-gram set of password abcabc is $\{\overline{\mathrm{SOS}a}, \overline{ab}, \overline{bc}, \overline{ca}, \overline{ab}, \overline{bc}, \overline{c\mathrm{EOS}}\}$. Therefore, the similarity of these two passwords is $\mathrm{sim}(\texttt{abc}, \texttt{abcabc}) = \frac{1*1+1*2+1*2+1*1}{\sqrt{1+1+1+1}\sqrt{1+4+4+1+1}} = 0.905$. Note that the similarity value of two passwords is between 0 and 1, and the larger the value, the higher the similarity.

In this paper, we choose 0.3 as the threshold of 2-gram cosine similarity between password pairs, because such passwords account for about 30% in most of our datasets. While this paper empirically shows that 0.3 is acceptable as a rule of thumb, one may choose other thresholds according to her own situation. To further confirm the effectiveness of 2-gram cosine similarity, we conduct a series of comparative experiments (see Sec. 4.4) with two different metrics (i.e., sim>0.3 vs. edit distance≤4). For the first time, we show that using sim>0.3 is slightly better (e.g., improving 9% in attack success rate at 1,000 guesses for our PASS2EDIT model) than using edit distance≤4 through large-scale experiments.

4 Experiments

We first elaborate on the experimental setups, and then fairly/comparatively evaluate our PASS2EDIT and its variant Pass2edit-nomix with their foremost counterparts (i.e., Pass2Path [46], TarGuess-II [71] and their variants).

4.1 Our datasets and ethical considerations

Datasets. We evaluate the existing password guessing models and our PASS2EDIT based on 11 large-scale password datasets (see Table 1), containing 4.8 billion passwords. Our password datasets include four from English sites and five from Chinese sites. They were hacked and made public on the Internet between 2011 and 2021. For the password reuse attack, we obtain the datasets composed of password pairs by matching the email. For details of these datasets, see Table 2. Note that 000Webhost is mainly used by web administrators, so its users are likely to be more security-savvy than common users, and this has been confirmed in [71]. Thus, the lists 000Webhost→LinkedIn, Yahoo→000Webhost, LinkedIn→000Webhost and 000Webhost→RedMart (A→B means that: A user's password at service A can be used by an attacker to help attack this user's account at service B) will show more secure reuse behaviors than that of common users (see attacking scenarios #5-#7 and #12 in Table 2). Besides, we count the proportion of 3Class8 passwords (which denotes passwords that must contain at least three character classes, i.e., uppercase/lowercase letters, symbols, and digits, and satisfy *len*≥8) for each dataset, and find that the value of 000Webhost far exceeds that of other datasets.

Ethical considerations. Though ever publicly available and widely used in password studies [18, 46, 49, 50, 70, 71], these datasets contain private data. Therefore, we take special care when dealing with them, e.g., only reporting the aggregated statistical information and treating each individual account as

Table 1: Data Cleaning of the password datasets leaked from various web services ("PWs" stands for passwords).

Dataset	Web service	Language	Leaked Time	Original PWs	Invalid emails	Invalid PWs	Removed %	After cleaning	3Class8† %
Tianya	Social forum	Chinese	Dec. 2011	30,816,592	5,783	3,279	0.03%	30,807,530	2.68%
126	Email	Chinese	Dec. 2011	6,392,568	0	14,995	0.24%	6,377,573	2.66%
Dodonew	E-commerce & Gaming	Chinese	Dec. 2011	16,282,286	225,931	30,085	1.57%	16,026,270	1.08%
Taobao	E-commerce	Chinese	Feb. 2016	15,072,418	1,176	90	0.01%	15,071,153	0.84%
CSDN	Programmer forum	Chinese	Dec. 2011	6,428,410	7	3,157	0.05%	6,425,246	3.67%
000Webhost	Web hosting	English	Oct. 2015	15,299,907	49,061	67,401	0.76%	15,183,445	19.41%
LinkedIn	Job hunting	English	Jan. 2012	54,656,615	0	122,051	0.23%	54,534,564	8.39%
Yahoo	Portal(e.g., E-commerce)	English	Jul. 2012	5,737,798	119	54,105	0.95%	5,683,574	5.32%
RedMart‡	E-commerce	English	Oct. 2020	1,108,774	0	—	0	1,108,774	—
4iQ	Mixed	Mixed	Dec. 2017	1,400,553,869	575,283	18,475,938	1.36%	1,381,502,648	5.56%
COMB	Mixed	Mixed	Feb. 2021	3,279,064,312	81,542,117	15,718,941	2.97%	3,181,803,254	7.95%

†3Class8 means passwords that must contain at least three character classes (i.e., uppercase/lowercase letters, symbols, and digits) and satisfy $len \geq 8$.
‡RedMart dataset is leaked from a Singapore's leading online supermarket. These passwords are in salted-hash and will be used as real targets.

confidential, storing and processing them on computers not linked to the Internet. While these datasets might be already exploited by attackers for misconduct, our use is helpful for security administrators/users to measure password strength and prevent weak ones (Since guessability is found to be a good metric for password strength [15, 33], and those easily guessed by an attacker are considered weak passwords). More specifically, the defenses (e.g., one can design a personalized password strength meter similar to [46]) derived from our guessing model can be in the public interest. As our datasets are all publicly available from various sources over the Internet, the results in this work are reproducible.

Datasets cleaning. We remove the entries with empty passwords, emails that do not contain `@` characters and malformed data (some datasets do not escape special characters). As with [46], we further remove strings that include symbols beyond the 95 printable ASCII characters. Additionally, we also remove strings with $len \geq 30$ because after manually scrutinizing the original datasets, we find that these long strings do not seem to be generated by users but are more likely by password managers or simply junk information.

4.2 Attack scenarios design

To evaluate the effect of our PASS2EDIT model, we need to answer the following three key research questions (RQs):

RQ1: How well does PASS2EDIT perform in password reuse behavior characterizing when comparing with its foremost counterparts (e.g., Pass2Path [46] and TarGuess-II [71])?

RQ2: How effective is our PASS2EDIT model in practical attacking scenarios?

RQ3: Does the efficiency of our PASS2EDIT model meet the needs of the real attacker?

To answer **RQ1** and compare with the existing guessing approaches fairly, we employ the 4iQ dataset [3] (which was also used in the original Pass2Path work [46]) and recently leaked COMB dataset [6] to perform the comparative experiments. Both of them are mixed datasets from multiple sources that contain billions of email and password pairs. We preprocess them with the "email-based" matching method employed by [46]. Specifically, for the same user (identified by the email address), if the email address appears in at least two accounts, then two of her passwords are randomly selected as the original password pw_A and the new password pw_B, respectively. The processed dataset consists of the password pairs $\langle pw_A, pw_B \rangle$. We take 80% of them as the training set, and the rest as the test set (i.e., scenarios #10 and #11 in Table 2). This creates a general attack scenario without considering any realistic factors (e.g., language, policy, and service). Since previous work [36, 70] showed that language plays an important role in the characteristics and strength of passwords, we use the same matching method to create a Chinese mixed dataset consisting of Tianya, Dodonew, and CSDN, and an English mixed dataset consisting of 000Webhost, LinkedIn, and Yahoo (i.e., scenarios #8 and #9 in Table 2).

Although the manually mixed dataset (scenarios #8-#11 in Table 2) can evaluate the scalability of different models (**RQ1**), it cannot show their effects in practical attacking scenarios (**RQ2**). This is because users have different preferences when creating passwords on different types of websites (e.g., users tend to create stronger passwords for financial accounts [10]). Thus, for a real attacker, she can constantly improve her training set to make it as close as possible to the test set. For instance, the target system's password distribution can be largely approximated by a leaked site with the same language, service types, and password policies [70].

To answer **RQ2**, we design a number of practical attack scenarios (see Table 2) to simulate the attacker's selection of a reasonable training set and compare the cracking success rates of different approaches. More specifically, for attack scenario #1: Dodonew and Taobao datasets are related to finance, and neither has any password policy. That is, the password policy and the service type of the training set match those of the test set in this scenario. For attack scenario #2: Since the policy of CSDN requires the created password greater than or equal to 8 (i.e., $len \geq 8$), and the 126 dataset has no password policy, we select the passwords with $len \geq 8$ from Dodonew as the training set to simulate the scenario that users modify from a simple password to a relatively complex one. For attack scenario #3: It is opposite to the scenario #2, changing from a relatively complex password to a simple password. For attack scenario #4: The service of the training set does not match the test set (but the policy matches). For attack scenarios #5-#7: These scenarios are similar to scenarios #2-#4, but the language is English. Among them, scenario #5 is to change

Table 2: Setups of 12 different attacking scenarios (RQ=Research question, see Section 4.2; For evaluation results, see Fig. 5)†

Scenario #	RQ# addressed	Language	Training set setup	Size (pairs)	Test set setup	Size (pairs)
1	RQ2	Chinese	Tianya → Dodonew	624,925	Tianya → Taobao	57,7017
2	RQ2		126 → Dodonew ($len \geq 8$)	188,926	126 → CSDN ($len \geq 8$)	85,206
3	RQ2, RQ3		CSDN → Dodonew	211,385	CSDN → 126	86,104
4	RQ2		Tianya → Dodonew ($len \geq 8$)	434,255	Tianya → CSDN ($len \geq 8$)	826,559
5	RQ2	English	000Webhost → Yahoo ($len \geq 6$)	265,083	000Webhost → LinkedIn ($len \geq 6$)	265,083
6	RQ2		Yahoo → LinkedIn (LD)	40,646	Yahoo → 000Webhost (LD)	37,479
7	RQ2		LinkedIn → Yahoo (LD, $len \geq 6$)*	40,812	LinkedIn → 000Webhost (LD, $len \geq 6$)	259,175
8	RQ1, RQ3	Mixed	80% of 3 mixed English datasets	338,857	20% of 3 mixed English Datasets	84,714
9	RQ1, RQ3		80% of 3 mixed Chinese datasets	434,255	20% of 3 mixed Chinese Datasets	108,564
10	RQ1, RQ3		80% of 4iQ dataset matched by email	116,837,808	20 % 4iQ dataset matched by email	29,209,452
11	RQ1, RQ3		80% of COMB dataset matched by email	342,921,727	20 % COMB dataset matched by email	85,730,432
12 (real)	RQ2	English	000Webhost → Linkedin (LD $len \geq 6$)	213,697	000Webhost → RedMart (LD $len \geq 6$)	6,858

†$A \to B$ means that: A user's password at service A can be used by an attacker to help attack this user's account at service B.
*(LD, $len \geq 6$) means that we only use passwords that contain at least one digit and one letter, and have a minimum length of 6 in the dataset.

from a complex password to a simple one; Scenario #6 is opposite to scenario #5; Scenario #7 is that neither policy nor service of the training set matches the test set.

In scenarios #1-#11, all test sets are in plain text. A natural question arises: Would our model keep effective when cracking "real accounts". We further design scenario #12 to compare the effectiveness of different approaches when cracking Redmart passwords, which are MD5 hashed with salt, leaked from a Singapore's leading online supermarket.

To answer **RQ3**, we conduct training and testing processes of the three models (i.e., Pass2Path [46], TarGuess-II [71] and our PASS2EDIT) on the same workstation and count the running time. In particular, the machine we used for the experiment is equipped with an Intel Xeon Silver processor, 256GB of RAM, NVIDIA RTX 3090 GPU (including 24GB of VRAM), and a 4TB hard drive. We believe that this configuration is not difficult to achieve for practical attackers.

Particularly, for all attack scenarios, if the size of the password pairs in one test set exceeds 20,000, we randomly sample 20,000 pairs from them instead of using the entire test set. We find that, at this time, the cracking success rates of all comparison approaches have already converged (which is consistent with [46]: using 10,000 password pairs as a test set is enough). For COMB, considering the memory consumption caused by a large amount of data, we randomly sampled 100 million data for experiments. Note that the test set may contain identical password pairs (i.e., pw_A=pw_B), and we count the proportion of such password pairs in each test set (see Table 3).

4.3 Guessing approaches for comparison

We now compare our PASS2EDIT model with two leading password reuse models (i.e., TarGuess-II [71] and Pass2Path [46]) and their variants. For a fair comparison, we ensure that all six models work on the same training and test sets, and manage to obtain/use their codes shared/open-sourced by the original authors. For all model parameters, we follow the best recommendations. For a better illustration, we further compare against the basic dictionary attacker that exploits the top-password list obtained from the training set. Details on the specific setup are as follows.

TarGuess-II. This model was proposed by Wang et al. [71] in 2016. The parameter settings retain the default settings in the code provided by the authors. Note that TarGuess-II [71] externally has a structural segment file trained by PCFG [74], two n-gram string files trained by Markov [39], and a popular password file to help generate guesses (see the Sec. 4.2 of [71] for details). These data files are trained in advance by the "three mixed English/Chinese dataset" we construct in scenarios #8 and #9. For popular password dictionaries, the Chinese popular password dictionary is $\mathcal{L}_C$=$\{pw|$the value of $P_{\text{csdn}}(pw) * P_{126}(pw) * P_{\text{Dodonew}}(pw)$ ranks top-$10^4\}$, and the English popular password dictionary is $\mathcal{L}_E$=$\{pw|$the value of $P_{\text{000Webhost}}(pw) * P_{\text{Yahoo}}(pw) * P_{\text{LinkedIn}}(pw)$ ranks top-$10^4\}$. We ensure that all compared models (i.e., Pass2Path-mix [46], TarGuess-II [71], and our PASS2EDIT) in this work use the same popular password dictionaries when mixing their generated guesses.

Pass2Path. This model was proposed by Pal et al. [46] in 2019. We use the open-source code of the paper, and the hyperparameter settings are as recommended. Specifically, the learning rate is 0.0003, the layer of RNN is three, the hidden unit is 128, and the dropout probability is 0.4. As with [46], we randomly select 20% from the training set as the validation set. For the 4iQ/COMB dataset, we follow the recommendations of the original paper and train for three epochs. While for other datasets, considering that their size is much smaller than 4iQ, we set the number of epochs to 20, because we find that the loss has converged at this time and there is no serious overfitting (judged by the validation set).

Pass2Path-bugfix. We notice that Pal et al. [46] have used a data enhancement mechanism: Both $\langle pw_A, pw_B \rangle$ and $\langle pw_B, pw_A \rangle$ are used for training. However, when considering the impact of password policy, this mechanism will interfere with the learning of the model, and sometimes slightly reduce the cracking success rate, so we remove this mechanism. Besides, the guesses finally generated by Pass2Path [46] contains duplicated passwords (because different transformation paths may get the same password), so we have de-duplicated them and only kept the one with the higher probability. The parameter settings are the same as the original Pass2Path. Since the size of training data is reduced to half of the original Pass2Path, we set the training epoch to 40.

Pass2Path-mix. Considering that TarGuess-II [71] and our PASS2EDIT have used popular password dictionaries, we also mix popular passwords in the generated set of Pass2Path [46], and the mixing ratio is 2:1 (this ratio is the best ratio tested in our experiment). Note that, we have also tried a mixing method that is exactly the same as PASS2EDIT but found that the effect is not as good as 2:1 (the specific results can be seen in Table 4). We discuss this issue in Sec. 4.4 (the description of "*Effect of mixing popular passwords*").

PASS2EDIT. It is the targeted password guessing model proposed in this paper. It consists of a 3-layer GRU and a 2-layer FC. The learning rate is 0.001, the dropout rate is 0.4, and the training epochs are 40 (for the 4iQ dataset, it is three).

PASS2EDIT-nomix. This model removes the mixed popular passwords dictionary of our PASS2EDIT.

TarGuess-II-nomix. This model removes the popular password dictionaries employed by TarGuess-II [71]. We notice that after removing the popular password dictionary, TarGuess-II [71] can still generate popular passwords (because of the structure-level transformation; see more details in Sec. 4.2 of [71]), even if the corresponding original password is not similar to the newly generated popular passwords.

Top-PW: We sort the popular passwords in the *training set* in a descending order of probability. In this way, we conduct a basic dictionary attack. Note that this dictionary is different from the popular dictionaries (i.e., $\mathcal{L}_C$ and $\mathcal{L}_E$) used by TarGuess-II [71], PASS2EDIT [46] and Pass2Path-mix [46].

JtR: We enable the John the Ripper toolkit [52] in wordlist mode with word mangling rules. JTR has 57 word-mangling rules in its configuration file, along with a default password list (password.lst). We append one of the passwords from each pair from our data set into this password.lst file.

Combined method. To avoid the bias of a single model when characterizing users' password reuse behaviors, we introduce the Min-auto strategy [62] to represent the upper limit of combining the three models (i.e., TarGuess-II [71], Pass2Path [46], and PASS2EDIT): A password in the test set is considered cracked as long as any of the three models cracks it.

4.4 Evaluation results

Table 3 shows the proportion of identical password pairs (i.e., pw_B=pw_A) in each test set, and the cracking success rates of popular password dictionaries employed by three guessers (i.e., TarGuess-II [71], Pass2Path-mix [46], and our PASS2EDIT). Results show that 26.87%-64.46% of Chinese users directly reuse their existing passwords, while this value is only 4.94%-19.55% for English users. Similarly, with only popular dictionaries, 4.94%-17.17% of Chinese user passwords can be cracked directly within 1,000 guesses, while this value is only 1.95%-5.25% for English users. We list these results separately as they are model-independent.

We use the guess-number-graph (see Fig. 5) to evaluate the effectiveness of our PASS2EDIT with its foremost counterparts (i.e., TarGuess-II [71], Pass2Path [46], and their variants). To accurately show the success rate of all approaches (including the JtR [52]), we further give the concrete results at some specific guess numbers (i.e., 10, 100, 1,000, which are typical values considered by the main-stream literature [46, 71]) for all scenarios in Tables 10 and 11 of our full version paper.

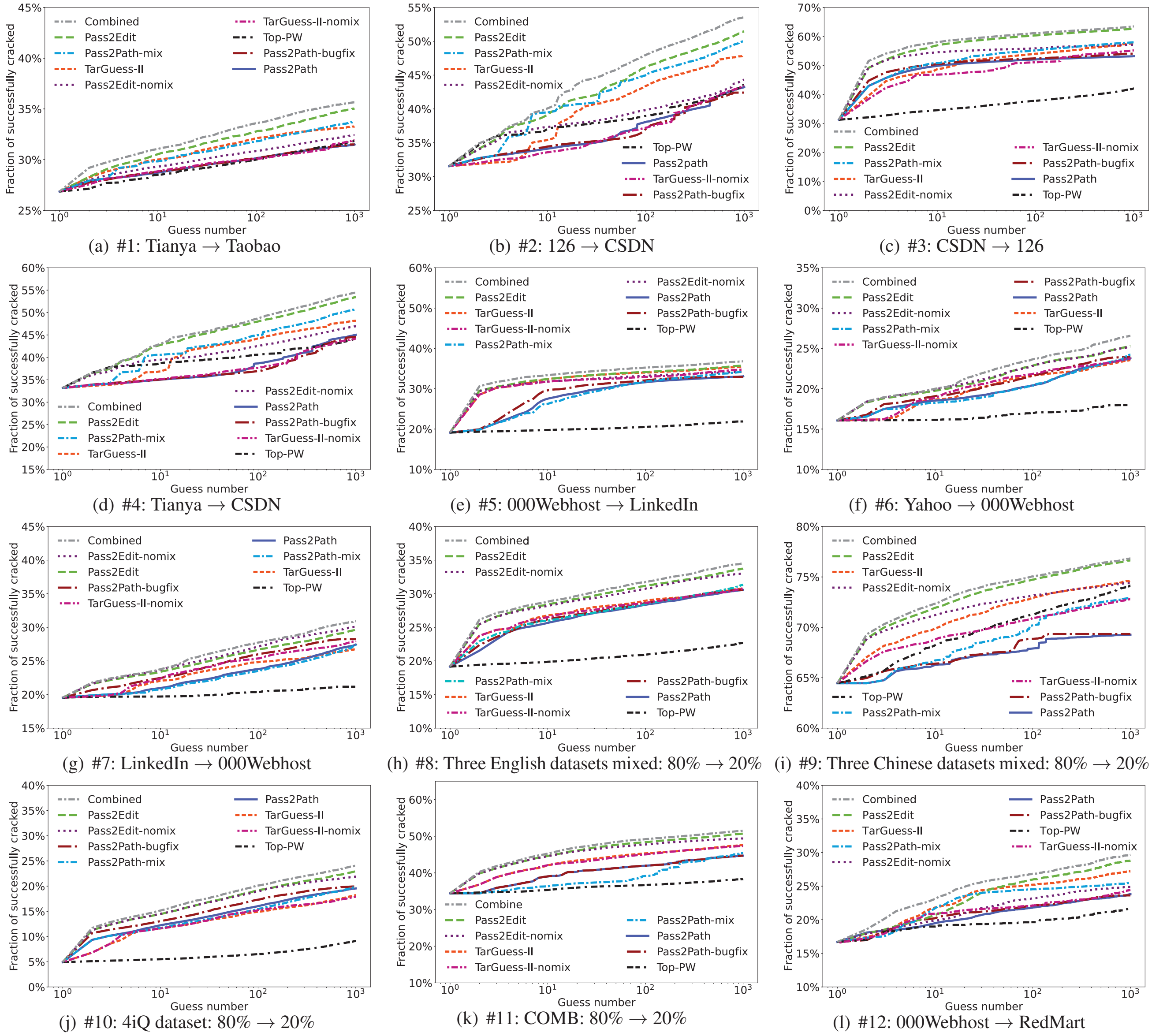
Table 3: Cracking success rates of popular password (PW) dictionaries.

Experimental setup (see Table 2) Attacking scenarios	Guesses #	Cracked by popular PW dictionaries	Identical password pairs
#1: Tianya→Taobao	10 100 1,000	2.67% 4.00% 4.94%	26.87%
#2: 126→CSDN	10 100 1,000	8.42% 12.09% 15.78%	31.55%
#3: CSDN→126	10 100 1,000	9.17% 13.10% 17.17%	31.28%*
#4: Tianya→CSDN	10 100 1,000	8.81% 12.23% 15.92%	33.18%
#5: 000Webhost→LinkedIn	10 100 1,000	1.69% 2.49% 4.39%	19.14%
#6: Yahoo→000Webhost	10 100 1,000	0.00% 0.58% 1.95%	16.07%
#7: LinkedIn→000Webhost	10 100 1,000	0.00% 0.25% 1.37%	19.55%*
#8: Mixed_E: 80%→20%†	10 100 1,000	0.59% 1.59% 3.21%	19.17%
#9: Mixed_C: 80%→20%†	10 100 1,000	7.40% 10.44% 13.38%	64.46%
#10: 4iQ dataset: 80%→20%	10 100 1,000	0.62% 1.38% 3.49%	4.94%
#11: COMB: 80%→20%	10 100 1,000	1.51% 2.46% 4.64%	34.44%
#12: 000Webhost→RedMart	10 100 1,000	2.86% 3.62% 5.25%	16.70%

†Mixed_E=English mixed dataset; Mixed_C=Chinese mixed dataset.
*The value in attack scenario #3/#7 is unequal to scenario #2/#5 because 20,000 test password pairs are randomly chosen for each attack scenario.

Overall analysis. Fig. 5 shows that the performance of our PASS2EDIT(-nomix) is better than all its counterparts in most experimental scenarios. More specifically, if there is no mixture of popular passwords, the cracking success rate of our PASS2EDIT-nomix is 17.04% higher than Pass2Path [46] (which natively does not consider popular passwords), and is 11.58% higher than TarGuess-II-nomix [71] within 1,000 guesses. Only in attack scenarios #5 and #12, the success rate of TarGuess-II-nomix [71] is comparable to our PASS2EDIT-nomix. This is because TarGuess-II-nomix [71] can still generate popular passwords even if the popular dictionary is removed. Our PASS2EDIT significantly improves its advantage when all models employ the popular dictionaries. More specifically, after mixing popular passwords, the success rates of our PASS2EDIT outperform Pass2Path-mix by 18.51%, and outperform TarGuess-II [71] by 22.89% within 1,000 guesses. For the sake of completeness, we also explore the performance of different attack approaches under a relatively larger number of guesses (i.e., 10^4 guesses). As shown in Fig. 6 (take scenario #8 as an example), we can see that our PASS2EDIT still outperforms all its counterparts.

Overall, when allowed 100 guesses, the average success rates of our PASS2EDIT against common users (see Figs. 5(a)-



(a) #1: Tianya → Taobao (b) #2: 126 → CSDN (c) #3: CSDN → 126

(d) #4: Tianya → CSDN (e) #5: 000Webhost → LinkedIn (f) #6: Yahoo → 000Webhost

(g) #7: LinkedIn → 000Webhost (h) #8: Three English datasets mixed: 80% → 20% (i) #9: Three Chinese datasets mixed: 80% → 20%

(j) #10: 4iQ dataset: 80% → 20% (k) #11: COMB: 80% → 20% (l) #12: 000Webhost → RedMart

Figure 5: Experiments for 12 targeted scenarios, for each of which the training set is shown in Table 2 and the test set is as the sub-title. The *combined* curve represents the upper limit of combining TarGuess-II [71], Pass2Path [46], and Pass2Edit. Our Pass2Edit(and -nomix) performs better than its counterparts.

5(d) and Fig. 9) and security-savvy users (see Figs. 5(e)-5(g) and Fig. 5(l)) are 47.81% and 27.42% respectively, while this figure is 45.29% and 25.05% for Pass2Path-mix [46], and is 45.26% and 26.42% for TarGuess-II [71]; When allowed 1,000 guesses, this figure is 52.01% and 29.87% for our Pass2Edit, 49.59% and 27.89% for Pass2Path-mix, and 48.80% and 28.26% for TarGuess-II [71].

When allowed 100 guesses and excluding the cases where the target password equals the original password (i.e., $pw_B \neq pw_A$), the average success rates of Pass2Edit against common users (see Figs. 5(a)-5(d) and Fig. 9) and security-savvy users (see Figs. 5(e)-5(g) and Fig. 5(l)) are 24.18% and 11.68% respectively, while this figure is 20.45% and 8.78% for Pass2Path-mix [46], and is 20.46% and 10.46% for TarGuess-II [71]; When allowed 1,000 gueses, this figure is 30.34% and 15.32% for Pass2Edit, 26.80% and 12.79% for Pass2Path-mix, and 25.65% and 13.03% for TarGuess-II.

Effect of mixing popular passwords. From Fig. 5 (and Table 11 in our full version paper), we can see that Pass2Edit performs better than Pass2Edit-nomix in 10 out of 12 attack scenarios, which shows the effectiveness of the mixed popular dictionary. Specifically, within 1,000 guesses, Pass2Edit outperforms Pass2Edit-nomix by 24.69% (on average) in these 10 scenarios. This is because password pairs containing popular passwords (e.g., `password` and `123456789`) can be quickly cracked through this dictionary. Note that, in scenarios #6 and #7, the attack success rate of Pass2Edit-nomix is slightly higher than that of the mixed model (Pass2Edit), which indicates that the popular dictionary does not work well against 000Webhost. The reason is that the users of 000Web-

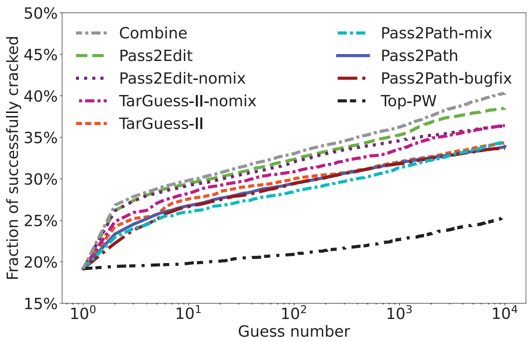


Figure 6: The cracking success rates of all compared approaches within 10^4 guesses (here we take the attack scenario #8 as an example).

Table 4: The results of Pass2Path after mixing popular passwords.†

Attack models	Pass2Path [46]		Our PASS2EDIT
Attack scenarios	Nomix	Mixed‡	Mixed
#1: Tianya → Taobao	6.32%	5.76%	11.18%
#2: 126 → CSDN	17.06%	28.07%	29.10%
#3: CSDN → 126	31.87%	40.34%	45.70%
#4: Tianya → CSDN	17.72%	28.23%	30.31%
#5: 000web → LinkedIn	17.21%	4.39%	20.57%
#6: Yahoo → 000web	9.16%	1.95%	10.96%
#7: LinkedIn → 000web	9.81%	1.37%	12.49%
#8: Mixed: 80% → 20%	14.11%	4.61%	18.02%

†All values are the results of the two models at the guess number of 1,000.
‡The mixed dictionaries and method are the same as our PASS2EDIT.

host are web administrators and generally have stronger security awareness, and popular passwords are less frequent (e.g., the sum of top-10 passwords account for only 0.79%, while this value is 3.28%-10.44% for common Chinese users [70]).

We also find that in all English attacking scenarios (i.e., #5-#8), the cracking success rate of Pass2Path [46] is drastically reduced if we mix the popular passwords *in the same way* as PASS2EDIT (see Table 4). This is because the probability of the password generated by Pass2Path [46] is lower than that of the popular password in the dictionary after the same adjustment as PASS2EDIT (multiply by 0.3), which leads to a large number of popular passwords (they are not very effective in English attacking scenarios) in the final ordered guessing set. To address this issue, we optimize the mixing method of Pass2Path [46] and insert popular passwords in the way of generated passwords: popular password = 2:1 (this ratio is the best ratio tested in our experiments). However, no matter which mixing method is employed, the cracking success rate of our PASS2EDIT is always higher than Pass2Path-mix (see Table 4 and Table 10 in the full version of this paper).

Impact of training set size. We take the attack scenario #3 as an example, and reduce the size of the training set to 1/2, 1/4, and 1/8 of the original training set respectively. Fig. 7 shows that when the size of the training set changes within [10^5,10^6], the cracking success rates of different models are largely unaffected. Among them, the statistics-based model TarGuess-II [71], has the best stability (the average deviation of its cracking success rate is <0.2%). While when the size of the training set becomes extremely large (e.g., >10^8), the advantage of the deep learning based models appears. For example, in attack scenario #10, both PASS2EDIT and Pass2Path [46] outperform TarGuess-II [71].

Table 5: The impact of filtering metrics on different methods.†

Method	Pass2Path [46]		PASS2EDIT		TarGuess-II [71]	
Attack scenarios	sim>0.3	$\delta \le 4$	sim>0.3	$\delta \le 4$	sim>0.3	s >0.5*
#1: Tianya → Taobao	6.79%	6.32%	11.18%	10.27%	9.07%	8.76%
#2: 126 → CSDN	16.20%	17.06%	29.10%	28.02%	24.44%	23.82%
#3: CSDN → 126	37.58%	31.87%	45.70%	41.80%	41.08%	38.38%
#4: Tianya → CSDN	15.43%	17.72%	30.31%	28.31%	23.82%	22.43%
#5: 000web → LinkedIn	18.34%	17.21%	20.57%	18.72%	20.51%	20.21%
#6: Yahoo → 000web	8.81%	9.16%	10.85%	10.64%	9.52%	8.92%
#7: LinkedIn → 000web	10.29%	9.81%	12.49%	8.79%	9.86%	8.96%
#8: Mixed: 80% → 20%	15.00%	14.11%	18.02%	16.37%	14.83%	14.40%

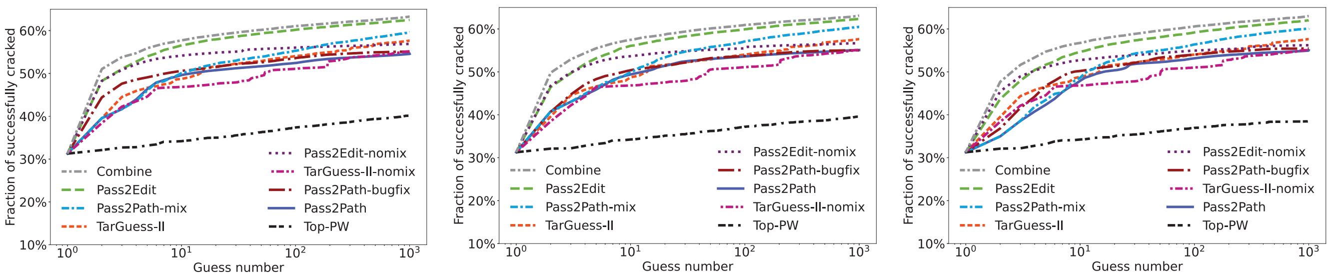
†All values are the results of the three models at the guess number of 1,000.
*$s = 1 - \text{EditDistance}(pw_A, pw_B)/\max(|pw_A|, |pw_B|)$.

Micro-perspective of cracking ability. To more clearly show the ability of leading models and our PASS2EDIT, we take scenario #2 as an example and plot the cosine similarity distribution of the cracked password pairs in the test set. Fig. 8 shows the cracking capabilities of different models from a micro perspective, from which we can see a roughly hierarchical ranking: PASS2EDIT>Pass2Path [46]>TarGuess-II [71].

Basic dictionary attack. In scenarios #2 and #4, the basic dictionary attack (i.e., top-PW) performs even better than advanced password models (for example, within 1,000 guesses, the success rate of top-PW is 10.83%, while the success rate of Pass2Path [46] is only 9.58%). The reason is that the weak passwords of some victims can be guessed directly without acquiring knowledge from their existing passwords, and the service type of the training set and test set do not match in these two scenarios. Thus, in practical attack scenarios, one can give priority to popular passwords, and the training set's language, service type, and password policy need to be fully considered when training a password model.

Impact of filtering metrics. To test the effectiveness of cosine similarity as a filtering metric, we use cosine similarity (>0.3) and edit distance ($\le$4) to filter the training set in attack scenarios #1-#8, respectively, and then use them to train of our PASS2EDIT model. We notice that the TarGuess-II [71] uses a similarity score based on edit distance (ED), calculated as $s = 1 - \text{ED}(pw_A, pw_B)/\max(|pw_A|, |pw_B|)$, and the threshold is set to 0.5. We additionally use cosine similarity to filter the training set in advance. For Pass2Path [46], we find that when using cosine similarity, there will be some password pairs with extremely long edit distance (>10) (e.g., `abc123@hotmail.com` and `abc123`), which makes the generation speed of the trained Pass2Path model extremely slow, so we manually remove these data (about 1%-3%).

Notably, for our PASS2EDIT, there is no similar problem because it only outputs a one-step edit operation at a time. Table 5 shows that using cosine similarity can improve the success rate of PASS2EDIT by 9%, can improve TarGuess-II [71] by 5%, and can improve Pass2Path [46] in five out of eight scenarios (1,000 guesses). A plausible reason is that cosine similarity is particularly good at measuring the structural similarity between two passwords [11, 24], and can preserve more password pairs with a longer edit distance after filtering. Particularly, we find that the training set (used in this paper) filtered by cosine similarity>0.3 is generally 3%-10%



(a) #3A: Trained on $\frac{1}{2}$CSDN $\rightarrow$ Dodonew (b) #3B: Trained on $\frac{1}{4}$CSDN $\rightarrow$ Dodonew (c) #3C: Trained on $\frac{1}{8}$CSDN $\rightarrow$ Dodonew

Figure 7:Influence of different training set sizes on the guessing success rate of Pass2Path [46], TarGuess-II [71] and our PASS2EDIT(-nomix). We take attack scenario #3 as an example (the training set is CSDN $\rightarrow$ Dodonew and the test set is CSDN$\rightarrow$126), reduce the training set size to 1/2, 1/4, and 1/8 of the original, and observe the influence. One can observe that the change in the size of the training set has little effect on the guessing success rates.

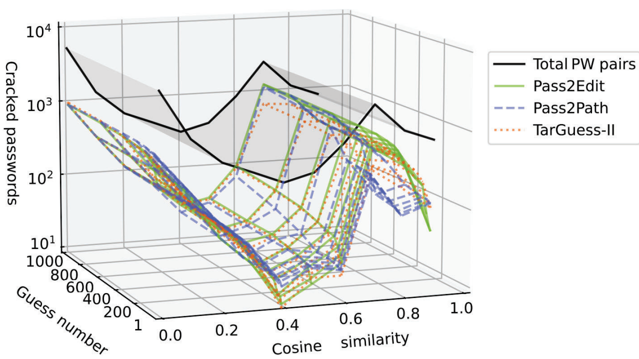


Figure 8: The comparison of Pass2Path [46], TarGuess-II [71] and our PASS2EDIT's ability to crack password pairs with different cosine similarity ranges. Here we take attack scenario #2 (see Table 2) for example.

Table 6: Running time of different attack models.†

Attack method	Training time	Testing time	Generated PW/s ‡
TarGuess-II [71]	00:59:44	00:57:13	5,538
Pass2Path [46]	14:09:45	01:46:42	2,969
PASS2EDIT	09:43:26	02:26:25	2,164

† The timings are taken from attack scenario #10 and their format is "hour:minute:second". All model parameters are consistent with Sec. 4.3.

‡ PW/s is calculated by dividing the total number by the total testing time.

larger than the training set filtered by edit distance$\leq$4, and the overlap ratio between the two is 92%-100%.

Attacking efficiency. Here we take scenario #10 using the 116 million 4iQ dataset as an example to examine the running time of different models. The detailed results are shown in Table 6. We can see that the statistics-based TarGuess-II [71], runs the fastest (both in training and generation process), Pass2Path [46] takes the second place, and PASS2EDIT is the slowest. Fortunately, for online attacks, the performance bottleneck lies in the speed of network requests and the throttling strategy of the websites [20], while computational complexity is not particularly important. Note that, PASS2EDIT is not good at applications where guess generation speed is important, like the compromised credential checking service [47] which per day generally handles millions of user requests and needs to generate billions of password guesses/variants.

Correctness confirmation. Note that in scenario #10, the cracking success rate of Pass2Path [46] within 1,000 guesses

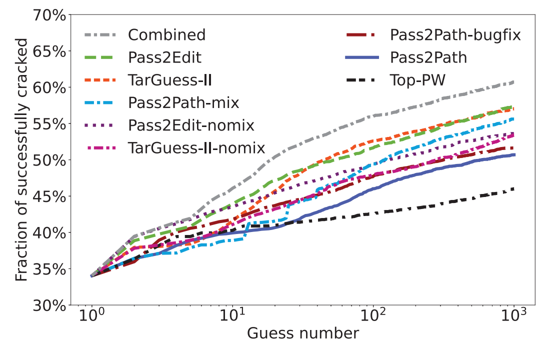


Figure 9: The comparison of our PASS2EDIT with its counterparts in attack scenario consistent with the original TarGuess-II paper [71] (i.e., trained on 68,546 CSDN$\rightarrow$12306, and tested on 5,997 CSDN$\rightarrow$Dodonew).

is 15.77%, which is almost the same as the original paper (i.e., 15.8% of Fig.3 in [46]). This indicates that we have correctly run the Pass2Path model. Similarly, to ensure that we correctly run the TarGuess-II algorithm [71], we use our existing datasets to additionally design the same attack scenario as in [71] (i.e., using CSDN$\rightarrow$126 as the training set and using CSDN$\rightarrow$Dodonew as the test set). The results of TarGuess-II [71] in Fig. 9 indicate that we have run this algorithm correctly, since the cracking success rate within 1,000 guesses in Fig. 13(f) of [71] is also about 57% without removing identical password pairs (i.e., pw_A=pw_B).

4.5 Further exploration

In what follows, we show some explorations we have made to improve our PASS2EDIT or existing password models.

Model input. In our PASS2EDIT, both pw^{orig} and pw_i^{cur} are used as the training input (they are converted to a new vector by using the concatenate function). A natural question may arise: Would our password model still work if we only use the current transformed password pw_i^{cur} as the training input? Our experimental results show that when only pw_i^{cur} is used, the guessing success rate of PASS2EDIT is still higher than Pass2Path [46] and TarGuess-II [71], but it is slightly worse than using pw^{orig} and pw_i^{cur} simultaneously (see Fig. 10). This suggests that both the current edited password pw_i^{cur} and the original password pw^{orig} provide useful knowledge on

Table 7: Examples of using all the training sets vs. using the filtered training sets.†

Examples	ihtfnjing		qwert1234		WANG520025		0112141333	
Training set	All	Filtered	All	Filtered	All	Filtered	All	Filtered
Rank	Unique top-10							
1	ihtfnj	ihtfnj	qwert1	qwert12	520025	g520025	112141333	112141333
2	**123456**	ihtfnji	t1234	rt1234	g520025	wang520025	12141333	2141333
3	ihtfnji	IHTFNJING	1234	qwert1	20025	ng520025	141333	141333
4	**123456789**	ihtfnjin	qwert12	wert1234	WANG5	WANG52	2141333	011214
5	**12345678**	ihtfnjg	**123456**	qwert123	**123456**	WANG52002	011214	12141333
6	ihtfnjg	0ihtfnjing	rt1234	QWERT1234	ng520025	WANG520	11214	0112141
7	htfnjing	fnjing	234	t1234	0025	WANG5200	0112141	01121413
8	IHTFNJING	ihtfnjig	**123456789**	0qwert1234	qwert12	WANG5	41333	011214133
9	**11111111**	htfnjing	**12345678**	**123456**	WANG52002	WANG55	112141	00112141333
10	ihtfnjin	ihnjing	qwert12	ert1234	WANG5200	0WANG520025	12141333	01121413

† "All" means using all password pairs in training set, and "Filter" means using password pairs whose edit distance is ≤4 in the training set. **Bold** passwords in each column mean that they are the popular passwords generated by the Pass2Path model [46] through the corresponding original password in the first row.

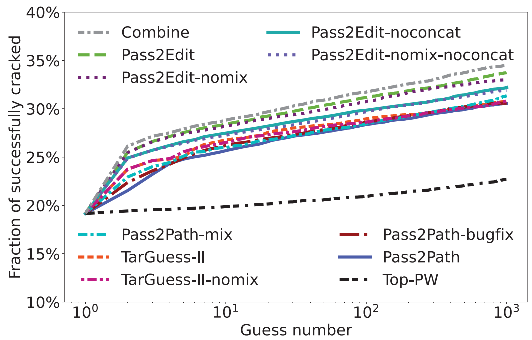


Figure 10: The cracking success rate of PASS2EDIT without concatenating (noconcat) the original password (here we take scenario #8 for example).

predicting the next transformation step. It is likely because the original password provides a benchmark for model training, preventing the model from losing its knowledge of the original password after several editing operations.

Parameter tuning. We summarize the tricks we have used into three categories. (i) Tunings that will improve the success rate: 1) Add 1-2 fully connected layers after the RNN layers, and the cracking success rate of two layers is better than one layer; 2) Transform the password into a key sequence containing caps-lock and shift keys; 3) Use dropout for RNN and fully connected layers, and the cracking success rate is better when the value is 0.4. (ii) Tunings that have little effect on the success rate: 1) Use bidirectional RNN; 2) Replace GRU cell with LSTM cell; 3) Introducing a residual connection for RNN; 4) Use the gradient clipping configuration [48] for RNN; 5) Adjust the number of layers of RNN (in the range {2, 3, 4}); 6) Adjust the dimensions of embedding, RNN, and fully connected layers. (iii) Tunings that will reduce the success rate: 1) Flip the training set, that is, use $pw_A \to pw_B$ and $pw_B \to pw_A$ at the same time; 2) Adding placeholders to achieve the purpose of data enhancement. For example, enrich the password pairs 1234→123 into ⊠1234→⊠123, ⊠⊠1234→⊠⊠123 and ⊠⊠⊠1234→⊠⊠⊠123.

Generation of popular passwords. To make the password model like Pass2Path [46] have the ability to generate popular passwords, we define a new atomic operation (denoted as *Bs*, which represents generating passwords from scratch), and the other atomic operations remain unchanged. For example, the transformation sequence of abcdef → 123 in original Pass2Path [46] is {(SUB,0,1), (SUB,1,2), (SUB,2,3), (DEL,3), (DEL,4), (DEL,5), EOS}. After adding the "start from scratch" atomic operation *Bs*, the transformation sequence of abcdef → 123 becomes {*Bs*, (INS,0,1), (INS,1,2), (INS,2,3), EOS}. However, our experimental results show that the cracking success rate becomes even worse after adding this operation. One possible reason is that this newly added operation interferes with the network's learning of the original insertion operation.

In addition, we have tried to directly use the entire training set to train the neural network without similarity filtering (by cosine similarity or edit distance). Table 7 shows ten examples generated by Pass2Path [46] by using all the training sets compared with using the training sets filtered by the cosine similarity (≥ 0.3). We find that the password model does generate popular passwords like 123456 and 12345678 after training with the entire training set, but the overall improvement in cracking success rate is marginal. The underlying reason is that the mixing of dissimilar password pairs interferes with the learning of the model and weakens the model's ability to characterize users' password reuse behaviors.

Additional approaches. Our PASS2EDIT essentially completes a task of character-level sequences classification. To improve its performance, we have tried some well-known models suitable for *short text classification*, such as Fasttext [32], TextCNN [75], and DPCNN [31], but found that the improvement in characterizing users' password reuse behaviors was marginal. Besides, we have implemented the Encoder structure of the Transformer [65] for this multi-label classification task. Unfortunately, the model's cracking success rate and training efficiency (i.e., training time and password generation speed) are drastically reduced. Additionally, we have introduced a residual network structure [26] based on the original model (i.e., 3-layer GRU+2-fully connected layer), while the success rate still has no obvious change. These attempts show that the existing NLP technique may not be able to *directly* migrate to the field of password guessing. Instead, it needs to be adaptively improved based on task requirements and the characteristics of password characters (e.g., short length, small feature dimensions, and rich/no semantics). For further exploration in these aspects, we leave them as future work.

A combined model. The structure-level and segment-level transformation defined by TarGuess-II [71] are restricted by the training dictionary. Specifically, the structure-level transformation requires the training dictionary to provide the specific password structure to be transformed and its corresponding probabilities (e.g., $\mathsf{L}_6\mathsf{D}_3 \rightarrow \mathsf{L}_6$ with probability 0.002), and the segment-level transformation requires the training dictionary to provide the specific content of the transformation and its corresponding probabilities (e.g., `123`→`1234` with probability 0.3). Unlike TarGuess-II [71], Pass2Path [46] can employ the entire original passwords to give the probability distribution of each step of transformation dynamically. Therefore, we can use Pass2Path [46] to generate transformation paths and corresponding probabilities for the structure-level and segment-level transformations in TarGuess-II [71].

For example, if the password pair (original password, new password) is (`pass123`, `1234@@`), then the segment sequence of the original password is [(0, 4, `pass`), (1, 3, `123`)]. Each item in this sequence (e.g., (0, 4, `pass`)) is called a segment, and each segment has three fixed items, followed by segment types, length, and specific characters. Among them, there are three types of segments, namely letters, digits, and special symbols, which are represented by 0, 1, and 2, respectively. Here, (0, 4, `pass`) means "a letter segment with a length of 4, and the specific content is `pass`". Similarly, the segment sequence of the new password `1234@@` is [(1, 4, `1234`), (2, 2, `@@`)]. On this basis, the modification sequence is represented as [(`d`, None, 0), (`s`, (1, 4, `1234`), 1), (`i`, (2, 2, `@@`), 2)], where each item is called a modification operation. The three sub-items in the modification operation respectively indicate the modification type (d: deletion, s: substitution, i: insertion), specific content, and location. For instance, (`i`, (2, 2, `@@`), 2) means inserting a segment (2, 2, `@@`) at the position marked 2 in the original password segment sequence.

To make the password model have the ability to generate the modification sequence [(`d`, None, 0), (`s`, (1, 4, `1234`), 1), (`i`, (2, 2, `@@`), 2)], we set up three sub-models, called struct_model, segment_model, and insert_dict, to complete the task of structure-level transformation prediction, substitution prediction within a segment, and insertion prediction within a segment, respectively. Among them, struct_model and segment_model are based on Pass2Path [46], and insert_dict employs a training dictionary.

We first use struct_model to predict the structure-level transformations. For example, if the input of struct_model is the original password `pass123`, then the output is a *structure-level* transformation sequence represented as [(`d`, None, 0), (`s`, None, 1), (`i`, (2, 2), 2)]. For the deletion and substitution operations (i.e., `d` and `s`), we only need to determine the position of the operation (i.e., 0 and 1 in items (`d`, None, 0) and (`s`, None, 1)), and do not need to determine the specific deletion/substitution content (i.e., 'None' in these items). For the insertion operation (i.e., `i`), we need to determine the type (L, S, and D segments represented by 0, 1, 2) and length of the inserted segment (i.e., (2, 2) in item (`i`, (2, 2), 2)).

We then use segment_model to predict the specific content to be substituted within a segment. For example, if the input of segment_model is `123` (which is the string before substitution operation, i.e., `123` in the original password `pass123`), then the output is `1234` (which is the string after the substitution operation, i.e., `1234` in the new password `1234@@`). For insert_dict, it outputs the corresponding string within the segment according to the segment type and length. For example, if its input is (2,2), then the corresponding output is `@@`.

Finally, we integrate the outputs of these three models to form a complete modification sequence (e.g., the sequence [(`d`, None, 0), (`s`, (1, 4, `1234`), 1), (`i`, (2, 2, `@@`), 2)] in the above example), and output all possible complete modification sequences in descending order of probability. However, our preliminary experimental results show that the performance of this combined model has not substantially improved compared to the original Pass2Path model [46].

4.6 Analysis of cracked passwords

Now we investigate the passwords cracked by TarGuess-II [71], Pass2Path [46], and our PASS2EDIT in terms of length, character composition, structure, similarity distributions, and complexity. To demonstrate their ability to generate edited passwords, we remove the popular password dictionaries employed by TarGuess-II [71] and PASS2EDIT. Ultimately, here the analysis builds on a total of 56,151 cracked password pairs from all 12 attack scenarios in Table 2.

Overlap. We first count the guesses generated by the three models and find that if each model generates a dictionary containing 1,000 unique guesses for a victim user, the overlap ratio for the three dictionaries is only 2%-10%. This implies that each model *generates* quite different guesses when inputting the same original password (i.e., pw_A). We then count *all cracked* passwords in the test set, and find that 59.5% of them were cracked simultaneously by all three models, and 1.0%-12.9% of passwords were cracked *independently* by each model (i.e., a password is only cracked by one of the three models, and not by the other two; see Fig. 11). This suggests that although the guesses generated by the three models are quite different, the passwords cracked by them have a very large overlap ratio.

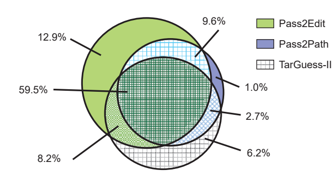


Figure 11: The overlap ratio of passwords cracked by three models.

Note that TarGuess-II [71] can still generate popular passwords even if the popular password dictionary is removed. The reason is that TarGuess-II can transform passwords at the structural level (e.g., $\mathsf{L}_8 \rightarrow \mathsf{D}_9$), and then fill the generated segment (i.e., D_9) with popular strings (e.g., `123456789`). This makes the cracked password pw_B have little similarity with the original password pw_A. We refer to this property of TarGuess-II as "Structured Advantage". Particularly, we find that 3.3% of the 6.2% passwords cracked independently by TarGuess-II are such popular passwords. However, this advantage will be weakened after all models employ the popular password dictionaries (see columns 7-9 of Table 10 in our full version).

Overall, our PASS2EDIT can independently crack the most edited passwords (i.e., 12.9%). To gain a deeper understanding of the differences between the three models, we investigate the password/password pairs *independently* cracked by each model (i.e., we remove the passwords that were cracked simultaneously by all three models). Figs 12-15 and Table 9 show the comparison results. Note that the values in Figs 12-15 and Table 9 are the percentage of passwords cracked for each model, not for all three models.

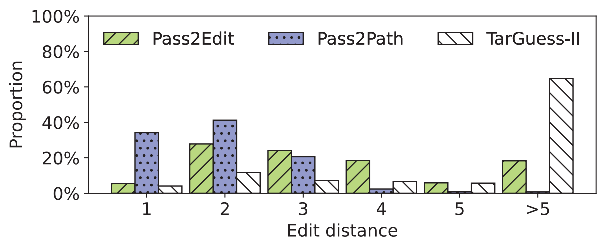


Figure 12: The edit distance distributions of cracked password pairs.

Edit distance. Fig. 12 shows that the independently cracked password pairs via our PASS2EDIT are distributed in each edit distance value. For Pass2Path [46], while it is good at guessing password pairs with edit distance<4, the proportion of cracked passwords decreases significantly as the edit distance value increases (since it cannot capture the connections between the edit operations and the corresponding edit effects). For TarGuess-II [71], due to its "Structured Advantage" mentioned above, the proportion of cracked password pairs with editing distance>5 is extremely high.

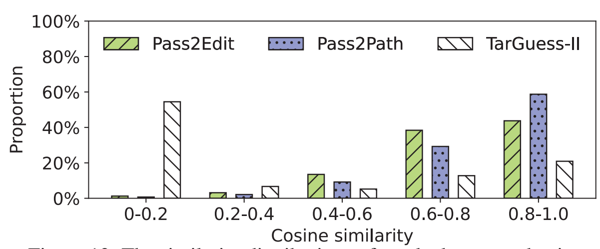


Figure 13: The similarity distributions of cracked password pairs.

Cosine similarity. Fig. 13 shows the similarity distributions of independently cracked password pairs. Since TarGuess-II [71] can generate popular passwords with little similarity to the original passwords (e.g., `seperti*`→`123456789`), the proportion of cracked password pairs with low cosine similarity is extremely high. On the contrary, the password pairs with high cosine similarity cracked by Pass2Path [46] account for the highest proportion, which is also consistent with the results of the edit distance distribution in Fig. 12: It's good at cracking password pairs with edit distance<4. For our PASS2EDIT, it performs well in cracking password pairs with a cosine similarity between 0.4-0.8.

Character position. The previous work [67] showed that users' modification of their existing passwords is related to

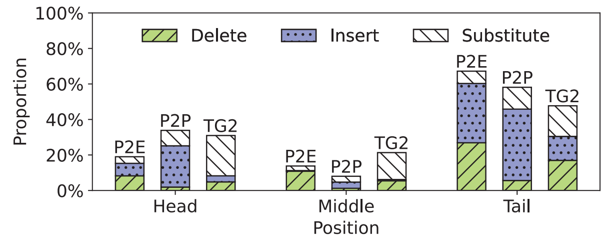


Figure 14: The edited position distributions of the cracked password pairs (P2E=PASS2EDIT, P2P=Pass2Path [46], TG2=TarGuess-II [71]).

the character position (e.g., 87.2% of insertions/deletions happened at the tail). Thus we also explore how different models work in different character positions. More specifically, we divide the cracked password into the head, middle, and tail parts (each of which is one-third of the entire password length), and further count the percentage of different parts being modified (i.e., insertion, deletion, and substitution). Fig. 14 shows that all three models tend to modify the password at the tail part. Particularly, our PASS2EDIT tends to delete characters in the middle part, TarGuess-II [71] tends to substitute characters at the head and middle parts, and Pass2Path [46] tends to insert characters at the head and tail parts.

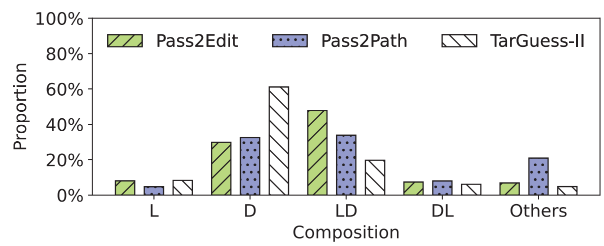


Figure 15: The structure distributions of cracked passwords.

Character compositions and top structures. Since the previous work [70] showed that the top-3 structural patterns account for an overwhelming fraction of users passwords (e.g., the top-3 structures D, LD, DL account for an average of 81.90% for Chinese datasets, and the top-3 structures L, LD, D account for an average of 81.26% for English datasets, where L denotes a lower-case sequence, D for a digit sequence, and S for a symbol sequence), we divide the cracked passwords into five categories (i.e., L, D, LD, DL, and others). Fig. 15 shows that different models are good at cracking passwords of different structures. It is interesting to see that our PASS2EDIT is good at cracking passwords with LD structure (which is the 2nd-ranked structure in English datasets [70]). In contrast, Pass2Path [46] performs well in cracking passwords with relatively complex structures since the proportion of "others" is 10%-15% higher than TarGuess-II [71] and our PASS2EDIT.

Length feature. Fig. 16 shows that the length of the passwords independently cracked by Pass2Path [46] is mainly concentrated in 8-9. Another interesting observation is that the passwords with length=6 account for a considerable proportion (i.e., 26.11%) of TarGuess-II [71]. We manually check the generated guesses, and find that the popular password `123456` accounts for an overwhelming fraction (i.e., 71.76%).

Table 8: Examples of passwords cracked independently by different models.

Attacking models		TarGuess-II [71]		Pass2Path [46]		Our PASS2EDIT	
Number	Language	Existing password	Targeted password	Existing password	Targeted password	Existing password	Targeted password
1	Chinese	`gxb840213`	`gxb1314521`	`biaokng`	`biaoking`	`201212`	`dai201212`
2		`dragonyr`	`123456789`	`ximmy851129`	`ximmy851119`	`9918241`	`zyj9918241`
3		`243586`	`qazwsxedc`	`199185`	`19910805`	`fire2500`	`ling2500`
4		`Tian6253*`	`love6253`	`zhangbig`	`ZHANGbig`	`1314520`	`1314520xl`
5		`2323kbc`	`123123kbc`	`super19771020`	`super19791020`	`6691064`	`6691064wu`
6	English	`seperti*`	`123456`	`JAtt12#$`	`JAtt1234`	`di10ca10040790`	`dica040790`
7		`sergioafull15013320`	`15013320`	`rajivamerica123`	`RAJIVamerica123`	`t@lking1`	`talking`
8		`megahomme@megahomme`	`megahomme`	`Iuliana93LAN`	`Iuliana93LaN`	`9427-078-168`	`9427078168`
9		`ddd786*1987`	`1987*786`	`kornjacica989`	`kornjaca89`	`Denningj11!!`	`denningj7`
10		`301873022iansangbbyboo`	`301873022`	`savone61`	`Savone6!`	`Ritalin!2#`	`ritalin123`

Table 9: The complexity of independently cracked passwords.†

Models	Complexity of cracked passwords %									Total
	1C6‡	1C8	1C12	2C6	2C8	2C12	3C6	3C8	3C12	#
PASS2EDIT	94.54	69.49	6.72	62.09	53.78	6.26	6.45	5.92	0.00	7,213
Pass2Path [46]	98.59	**90.82**	6.82	**63.76**	**59.29**	6.35	**12.00**	**12.00**	**2.59**	376
TarGuess-II [71]	**98.82**	68.18	**9.50**	30.67	28.07	**9.30**	3.92	3.79	1.25	3,490

† A **bold** value in each column means that it is the highest one among the three.
‡ 1C6=1Class6, which means passwords that must contain at least one character classes (i.e., uppercase/lowercase letters, symbols, and digits) and satisfy *len*≥6.

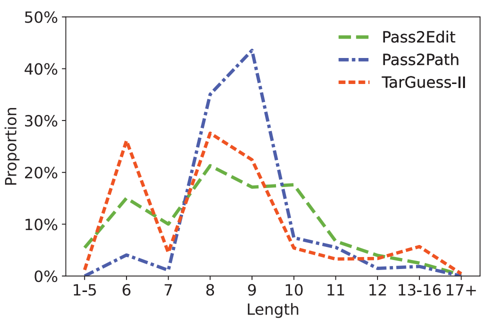


Figure 16: The length distributions of passwords cracked by each model.

For PASS2EDIT, the length distribution of its independently cracked passwords is relatively flat. Particularly, it can crack 31.04% of passwords with length>10, and this value is only 16.25% and 18.91% for Pass2Path [46], and TarGuess-II [71].

Password complexity. Table 9 summarizes the proportion of independently cracked passwords with different password composition rules/policies (PCPs). It is interesting to see that TarGuess-II [71] is good at cracking long passwords with relatively simple PCPs, while Pass2Path [46] performs well in cracking passwords with complex PCPs. For our PASS2EDIT, while the proportion of complex passwords (e.g., 3Class8) it cracked is not the highest, the number of complex passwords it cracked is the largest. Particularly, we give ten examples of passwords cracked independently by each model in Table 8.

5 Potential applications for protection

We now discuss the real-world security implications of our PASS2EDIT, and give some specific ways of how PASS2EDIT could be used to better protect users.

Password file recovery. A major usage of password guessing algorithms is to recover hashed password files. For example, after employing the trawling guessing algorithms/tools (e.g., PCFG [74] and Hashcat [56]) to recover the hashed password file to a certain proportion (e.g., 80%-90%), one can use the PASS2EDIT model proposed in this paper as a supplement to further recover more passwords. Besides, our PASS2EDIT can help recover the encrypted passwords of cyber criminals more quickly, and can also help users recover their forgotten reused passwords based on their existing passwords.

Password reuse identification. At IEEE S&P'19, Pal et al. [46] developed a personalized password strength meter (PPSM) to defend against password reuse attacks. Particularly, they have employed the existing targeted attacks (i.e., Das et al. [18], TarGuess-II [71], and Pass2Path [46]) to determine/label if a password pair is vulnerable to credential tweaking attacks. Considering that PASS2EDIT performs the best when guessing reused passwords, one can simply incorporate PASS2EDIT into existing attacks to improve the label accuracy, and thus improve the performance of this PPSM. Similarly, at CCS'21 [54], Sahin et al. proposed a machine learning-based classifier to predict when a password's security is likely affected by typo tolerance. Training this model requires to determin whether a password is vulnerable to password reuse attacks. Thus, one can also employ PASS2EDIT to help label vulnerable passwords, thereby improving the accuracy of the classifier and better protecting users.

Honeywords. Honeywords are decoy passwords stored together with each user's real password for detecting password file leakage. More specifically, this mechanism generates $k-1$ (e.g., k=40 as recommended by [72]) honeywords for each account to impede attackers from figuring out the real password. Even if the attacker steals the password file, she has to perform a few online login attempts. Once a certain number (e.g., three) of honeywords are checked for login attempts by the server, a password file leakage alarm is triggered. A key issue for the effectiveness of the honeyword mechanism is to generate flat honeywords (which means they are difficult to be distinguished from the real password). As a leading password reuse-based model, our PASS2EDIT has great potential to be employed by web administrators to generate flat honeywords.

6 Conclusion

This paper proposes a targeted password guessing algorithm PASS2EDIT to model the increasingly damaging credential tweaking attack, in which an attacker exploits the victim's leaked passwords to increase her success rate of guessing the victim's passwords at other sites. Particularly, for the first time, we propose a multi-step decision-making training mechanism, and build a classification neural network to learn the reaction of one-step edit operations to the existing password. This provides a brand new technical route to accurately and practically characterize users' password reuse behaviors and a better understanding of users' password security.

Acknowledgement

The authors are grateful to the shepherd and anonymous reviewers for their invaluable comments. Ding Wang is the corresponding author. This research was in part supported by the National Natural Science Foundation of China under Grants Nos. 62172240 and 62222208, and Natural Science Foundation of Tianjin, China under Grant No. 21JCZDJC00190. See the full version of this paper at `https://bit.ly/3ZjHPaD`.

References

[1] *Stick with Security: Require secure passwords and authentication*, Aug. 2017, https://www.ftc.gov/business-guidance/blog/2017/08/stick-security-require-secure-passwords-and-authentication.

[2] *Yahoo's 2013 Email Hack Actually Compromised Three Billion Accounts*, Oct. 2017, https://www.wired.com/story/yahoo-breach-three-billion-accounts/.

[3] *Identities in the Wild: The Tsunami of Breached Identities Continues*, May 2018, https://4iq.com/wp-content/uploads/2018/05/2018IdentityBreachReport4iQ.pdf/.

[4] *Password administration for system owners*, Nov. 2018, https://www.ncsc.gov.uk/collection/passwords/updating-your-approach.

[5] *Hack Brief: An Adult Cam Site Exposed 10.88 Billion Records*, May 2020, https://www.wired.com/story/cam4-adult-cam-data-leak-7tb/.

[6] *COMB: largest breach of all time leaked online with 3.2 billion records*, July 2022, https://cybernews.com/news/largest-compilation-of-emails-and-passwords-leaked-free/.

[7] *Data Breach Investigations Report*, June 2022, https://www.verizon.com/business/resources/reports/2022/dbir/2022-data-breach-investigations-report-dbir.pdf.

[8] *IBM Report: Cost of a Data Breach Hits Record High During Pandemic*, July 2022, https://www.ibm.com/downloads/cas/3R8N1DZJ.

[9] *The Password is Dead, Long Live the Password!*, Oct. 2016, https://www.nccgroup.trust/uk/about-us/newsroom-and-events/blogs/2016/october/the-password-is-dead-long-live-the-password/.

[10] D. V. Bailey, M. Dürmuth, and C. Paar, "Statistics on password re-use and adaptive strength for financial accounts," in *Proc. SCN 2014*.

[11] R. J. Bayardo, Y. Ma, and R. Srikant, "Scaling up all pairs similarity search," in *Proc. WWW 2007*, pp. 131–140.

[12] J. Bonneau, C. Herley, P. van Oorschot, and F. Stajano, "Passwords and the evolution of imperfect authentication," *Commun. ACM*, vol. 58, no. 7, pp. 78–87, 2015.

[13] J. Bonneau, C. Herley, P. C. Van Oorschot, and F. Stajano, "The request to replace passwords: A framework for comparative evaluation of web authentication schemes," in *Proc. IEEE S&P 2012*, pp. 553–567.

[14] M. Burnett, *Is there life after passwords?*, July 2016, https://medium.com/un-hackable/is-there-life-after-passwords-290d50fc6f7d.

[15] C. Castelluccia, M. Dürmuth, and D. Perito, "Adaptive password-strength meters from Markov models," in *Proc. NDSS 2012*.

[16] K. Cho, B. van Merrienboer, C. Gulcehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," in *Proc. EMNLP 2014*.

[17] K. Collins, *Facebook buys black market passwords to keep your account safe*, Nov. 2016, https://www.cnet.com/tech/services-and-software/facebook-chief-security-officer-alex-stamos-web-summit-lisbon-hackers/.

[18] A. Das, J. Bonneau, M. Caesar, N. Borisov, and X. Wang, "The tangled web of password reuse," in *Proc. NDSS 2014*, pp. 1–15.

[19] M. Dürmuth, F. Angelstorf, C. Castelluccia, D. Perito, and C. Abdelberi, "OMEN: faster password guessing using an ordered markov enumerator," in *Proc. ESSoS 2015*, pp. 119–132.

[20] M. Dürmuth, D. Freeman, S. Jain, B. Biggio, and G. Giacinto, "Who are you? A statistical approach to measuring user authenticity," in *Proc. NDSS 2016*, pp. 1–15.

[21] D. Florencio and C. Herley, "A large-scale study of web password habits," in *Proc. WWW 2007*, pp. 657–666.

[22] M. Golla, M. Wei, J. Hainline, L. Filipe, M. Dürmuth, E. Redmiles, and B. Ur, ""what was that site doing with my facebook password?" designing password-reuse notifications," in *Proc. ACM CCS 2018*.

[23] P. A. Grassi, E. M. Newton, R. A. Perlner, and et al., "NIST 800-63B digital identity guidelines: Authentication and lifecycle management," McLean, VA, Tech. Rep., June 2017.

[24] Y. Guo and Z. Zhang, "LPSE: Lightweight password-strength estimation for password meters," *Comput. Secur.*, vol. 73, pp. 507–518, 2018.

[25] A. Hanamsagar, S. S. Woo, C. Kanich, and J. Mirkovic, "Leveraging semantic transformation to investigate password habits and their causes," in *Proc. ACM CHI 2018*, pp. 1–10.

[26] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. CVPR 2016*, pp. 770–778.

[27] C. Herley and P. Van Oorschot, "A research agenda acknowledging the persistence of passwords," *IEEE Secur. Priv.*, vol. 10, pp. 28–36, 2012.

[28] B. Hitaj, P. Gasti, G. Ateniese, and F. Perez-Cruz, "Passgan: A deep learning approach for password guessing," in *Proc. ACNS 2019*.

[29] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[30] S. Houshmand, S. Aggarwal, and R. Flood, "Next gen PCFG password cracking," *IEEE Trans. Inf. Forensics Secur.*, vol. 10, no. 8, pp. 1776–1791, 2015.

[31] R. Johnson and T. Zhang, "Deep pyramid convolutional neural networks for text categorization," in *Proc. ACL 2017*, pp. 562–570.

[32] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification," in *Proc. EACL 2017*, pp. 427–431.

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

[34] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in *Proc. ICLR 2015*, pp. 1–15.

[35] K. Lee, S. Sjöberg, and A. Narayanan, “Password policies of most top websites fail to follow best practices,” in *Proc. SOUPS 2022*.

[36] Z. Li, W. Han, and W. Xu, “A large-scale empirical analysis on chinese web passwords,” in *Proc. USENIX SEC 2014*, pp. 559–574.

[37] B. Lu, X. Zhang, Z. Ling, Y. Zhang, and Z. Lin, “A measurement study of authentication rate-limiting mechanisms of modern websites,” in *Proc. ACSAC 2018*, pp. 89–100.

[38] S. G. Lyastani, M. Schilling, M. Neumayr, M. Backes, and S. Bugiel, “Is FIDO2 the kingslayer of user authentication? A comparative usability study of FIDO2 passwordless authentication,” in *Proc. IEEE S&P 2020*, pp. 268–285.

[39] J. Ma, W. Yang, M. Luo, and N. Li, “A study of probabilistic password models,” in *Proc. IEEE S&P 2014*, pp. 689–704.

[40] C. Mainka, V. Mladenov, J. Schwenk, and T. Wich, “Sok: Single sign-on security-an evaluation of openid connect,” in *Proc. EuroS&P 2017*.

[41] M. L. Mazurek, S. Komanduri, T. Vidas, L. F. Cranor, P. G. Kelley, R. Shay, and B. Ur, “Measuring password guessability for an entire university,” in *Proc. ACM CCS 2013*, pp. 173–186.

[42] W. Melicher, B. Ur, S. Komanduri, L. Bauer, N. Christin, and L. F. Cranor, “Fast, lean and accurate: Modeling password guessability using neural networks,” in *Proc. USENIX SEC 2017*, pp. 175–191.

[43] A. Narayanan and V. Shmatikov, “Fast dictionary attacks on passwords using time-space tradeoff,” in *Proc. ACM CCS 2005*, pp. 364–372.

[44] P. Negi, P. Sharma, V. Jain, and B. Bahmani, “K-means++ vs. Behavioral biometrics: One loop to rule them all,” in *Proc. NDSS 2018*.

[45] M. Nicholas, *68 Million Reasons Why Your Small Business Needs a Password Manager*, Jan. 2017, https://blog.dashlane.com/68-million-reasons-why-your-small-business-needs-a-password-manager/.

[46] B. Pal, T. Daniel, R. Chatterjee, and T. Ristenpart, “Beyond credential stuffing: Password similarity models using neural networks,” in *Proc. IEEE S&P 2019*, pp. 417–434.

[47] B. Pal, M. Islam, M. S. Bohuk, N. Sullivan, L. Valenta, T. Whalen, C. Wood, T. Ristenpart, and R. Chatterjee, “Might I get pwned: A second generation compromised credential checking service,” in *Proc. USENIX SEC 2022*, pp. 1831–1848.

[48] R. Pascanu, T. Mikolov, and Y. Bengio, “On the difficulty of training recurrent neural networks,” in *Proc. ICML 2013*, pp. 1310–1318.

[49] D. Pasquini, M. Cianfriglia, G. Ateniese, and M. Bernaschi, “Reducing bias in modeling real-world password strength via deep learning and dynamic dictionaries,” in *Proc. USENIX SEC 2021*, pp. 821–838.

[50] D. Pasquini, A. Gangwal, G. Ateniese, M. Bernaschi, and M. Conti, “Improving password guessing via representation learning,” in *Proc. IEEE S&P 2021*, pp. 265–282.

[51] S. Pearman, J. Thomas, P. E. Naeini, H. Habib, L. Bauer, N. Christin, L. F. Cranor, S. Egelman, and A. Forget, “Let’s go in for a closer look: Observing passwords in their natural habitat,” in *Proc. ACM CCS 2017*.

[52] A. Peslyak, *John the Ripper password cracker*, Feb. 1996, http://www.openwall.com/john/.

[53] E. M. Redmiles, S. Kross, and M. L. Mazurek, “How I learned to be secure: a Census-representative survey of security advice sources and behavior,” in *Proc. ACM CCS 2016*, pp. 666–677.

[54] S. Sahin and F. Li, “Don’t forget the stuffing! revisiting the security impact of typo-tolerant password authentication,” in *Proc. ACM CCS 2021*, pp. 252–270.

[55] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting,” *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.

[56] J. Steube, *Hashcat*, 2022, https://hashcat.net/hashcat/.

[57] E. Stobert and R. Biddle, “The password life cycle,” *ACM Trans. Priv. Secur.*, vol. 21, no. 3, pp. 1–32, 2018.

[58] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks,” in *Proc. NeurIPS 2014*, pp. 3104–3112.

[59] K. Thomas, J. Pullman, K. Yeo, A. Raghunathan, P. G. Kelley, L. Invernizzi, B. Benko, T. Pietraszek, S. Patel, D. Boneh *et al.*, “Protecting accounts from credential stuffing with password breach alerting,” in *Proc. USENIX SEC 2019*, pp. 1556–1571.

[60] Y. Tian, C. Herley, and S. E. Schechter, “Stopguessing: Using guessed passwords to thwart online password guessing,” *IEEE Secur. Priv.*, vol. 18, no. 3, pp. 38–47, 2020.

[61] B. Ur, F. Noma, J. Bees, S. M. Segreti, R. Shay, L. Bauer, N. Christin, and L. F. Cranor, “" i added’ !’ at the end to make it secure": Observing password creation in the lab,” in *Proc. SOUPS 2015*, pp. 123–140.

[62] B. Ur, S. M. Segreti, L. Bauer, N. Christin, L. F. Cranor, S. Komanduri, D. Kurilova, M. L. Mazurek, W. Melicher, and R. Shay, “Measuring real-world accuracies and biases in modeling password guessability,” in *Proc. USENIX SEC 2015*, pp. 463–481.

[63] US-CERT, *Choosing and Protecting Passwords*, Nov. 2019, https://us-cert.cisa.gov/ncas/tips/ST04-002.

[64] *Passwords are not lame and they’re not dead*, Aug. 2017, https://it.toolbox.com/blogs/itmanagement/passwords-are-not-lame-and-theyre-not-dead-heres-why-072417.

[65] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” in *Proc. NeurIPS 2017*, pp. 5998–6008.

[66] R. Veras, C. Collins, and J. Thorpe, “On the semantic patterns of passwords and their security impact,” in *Proc. NDSS 2014*, pp. 1–16.

[67] C. Wang, S. T. Jan, H. Hu, D. Bossart, and G. Wang, “The next domino to fall: Empirical analysis of user passwords across online services,” in *Proc. CODASPY 2018*, pp. 196–203.

[68] D. Wang, D. He, H. Cheng, and P. Wang, “fuzzyPSM: A new password strength meter using fuzzy probabilistic context-free grammars,” in *Proc. IEEE/IFIP DSN 2016*, pp. 595–606.

[69] D. Wang and P. Wang, “The emperor’s new password creation policies,” in *Proc. ESORICS 2015*, pp. 456–477.

[70] D. Wang, P. Wang, D. He, and Y. Tian, “Birthday, name and bifacial-security: Understanding passwords of Chinese web users,” in *Proc. USENIX SEC 2019*, pp. 1537–1555.

[71] D. Wang, Z. Zhang, P. Wang, J. Yan, and X. Huang, “Targeted online password guessing: An underestimated threat,” in *Proc. ACM CCS 2016*, pp. 1242–1254.

[72] D. Wang, Y. Zou, Q. Dong, Y. Song, and X. Huang, “How to attack and generate honeywords,” in *Proc. IEEE S&P 2022*, pp. 489–506.

[73] R. Wash, E. Rader, R. Berman, and Z. Wellmer, “Understanding password choices: How frequently entered passwords are re-used across websites,” in *Proc. SOUPS 2016*, pp. 175–188.

[74] M. Weir, S. Aggarwal, B. de Medeiros, and B. Glodek, “Password cracking using probabilistic context-free grammars,” in *Proc. IEEE S&P 2009*, pp. 391–405.

[75] X. Zhang, J. Zhao, and Y. LeCun, “Character-level convolutional networks for text classification,” in *Proc. NeurIPS 2015*, pp. 649–657.

[76] V. Zimmermann, “From the quest to replace passwords towards supporting secure and usable password creation,” Ph.D. dissertation, Technical University of Darmstadt, Germany, 2021.